Description of STM32N6 HAL and low-layer drivers

Introduction

STM32Cube is an STMicroelectronics original initiative to significantly improve developer productivity by reducing development effort, time, and cost. STM32Cube covers the STM32 portfolio.

STM32Cube includes:

- STM32CubeMX, a graphical software configuration tool that allows the generation of C initialization code using graphical wizards.
- A comprehensive embedded software platform, delivered per series (such as STM32CubeN6 for STM32N6 devices)
  - The STM32Cube HAL, STM32 abstraction layer embedded software ensuring maximized portability across the STM32 portfolio. HAL APIs are available for all peripherals.
  - Low-layer APIs (LL) offering a fast light-weight expert-oriented layer which is closer to the hardware than the HAL. LL APIs are available only for a set of peripherals.
  - A consistent set of middleware components such as USB, H.264 video encoder, image signal processing library, Azure® RTOS, OpenBootloader, external memory manager and loader.
  - All embedded software utilities, delivered with a full set of examples.

The HAL driver layer provides a simple, generic multi-instance set of APIs (application programming interfaces) to interact with the upper layer (application, libraries, and stacks). The HAL driver APIs are split into two categories: generic APIs, which provide common and generic functions for all the STM32 series and extension APIs, which include specific and customized functions for a given line or part number. The HAL drivers include a complete set of ready-to-use APIs that simplify the user application implementation. For example, the communication peripherals contain APIs to initialize and configure the peripheral, manage data transfers in polling mode, handle interrupts or DMA, and manage communication errors. The HAL drivers are feature-oriented instead of peripheral-oriented. For example, the timer APIs are split into several categories following the peripheral functions, such as basic timer, capture, and pulse width modulation (PWM). The HAL driver layer implements runtime failure detection by checking the input values of all functions. Such dynamic checking enhances the firmware robustness. Runtime detection is also suitable for user application development and debugging.

The LL drivers offer hardware services based on the available features of the STM32 peripherals. These services reflect exactly the hardware capabilities, and provide atomic operations that must be called by following the programming model described in the product line reference manual. As a result, the LL services are not based on standalone processes and do not require any additional memory resources to save their states, counter, or data pointers. All operations are performed by changing the content of the associated peripheral registers. Unlike the HAL, LL APIs are not provided for peripherals for which optimized access is not a key feature, or for those requiring heavy software configuration and/or a complex upper-level stack.

The HAL and LL are complementary and cover a wide range of application requirements:

- The HAL offers high-level and feature-oriented APIs with a high-portability level. These hide the MCU and peripheral complexity from the end user.
- The LL offers low-level APIs at register level, with better optimization but less portability. These require deep knowledge of the MCU and peripheral specifications.

The HAL- and LL-driver source code is developed in Strict ANSI-C, which makes it independent of the development tools. It is checked with the Coverity static analysis tool. It is fully documented.

It is compliant with MISRA C®:2012 standard.

This user manual is structured as follows:

- Overview of HAL drivers
- Overview of low-layer drivers
- Cohabiting of HAL and LL drivers
- Detailed description of each peripheral driver: configuration structures, functions, and how to use the given API to build your application



1 General information

The STM32CubeN6 MCU Package runs on STM32N6 32-bit microcontrollers based on the Arm® Cortex®-M processor.

Note: Arm is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

2 Acronyms and definitions

Table 1. **Acronyms and definitions**

Acronym	Definition
ADC	Analog-to-digital converter
AES	Advanced encryption standard
ANSI	American national standards institute
API	Application programming interface
BSEC	Boot and security control
BSP	Board support package
CAN	Controller area network
CMSIS	Cortex microcontroller software interface standard
CPU	Central processing unit
CRC	CRC calculation unit
CRYP	Cryptographic processor
DLYB	Delay block
DCMI	Digital camera interface
DCMIPP	Digital camera interface pixel pipeline
DMA	Direct memory access
DMA2D	DMA dedicated to image manipulation (Chrom-ART)
DTS	Digital temperature sensor
ETH	Ethernet controller
EXTI	External interrupt/event controller
FDCAN	Flexible data-rate controller area network unit
FLASH	Flash memory
FMC	Flexible memory controller
FSBL	First stage bootloader
GFXMMU	Graphical oriented memory management unit (Chrom-GRC)
GFXTIM	Graphic timer
GPDMA	General-purpose DMA
GPIO	General-purpose I/Os
GPU2D	Graphic processing unit (Neo-Chrom)
HAL	Hardware abstraction layer
HASH	Hash processor
HCD	USB host controller driver
HPDMA	High-performance DMA
I2C	Inter-integrated circuit
I3C	Improved inter-integrated circuit
I2S	Inter-integrated sound
ICACHE	Instruction cache
IRDA	Infrared data association
IWDG	Independent watchdog

Acronym	Definition
LCD	Liquid crystal display controller
LTDC	Liquid crystal display - thin film transistor display controller
LPTIM	Low-power timer
LPUART	Low-power universal asynchronous receiver/transmitter
LRUN	Load and run
MCE	Memory cipher engine
MCO	Microcontroller clock output
MDF	Multifunction digital filter
MDIOS	Management data input/output (MDIO) slave
MMC	MultiMediaCard
MPU	Memory protection unit
MSP	MCU specific package
NAND	NAND flash memory
NOR	NOR flash memory
NVIC	Nested vectored interrupt controller
PKA	Public key accelerator
PCD	USB peripheral controller driver
PPP	STM32 peripheral or block
PSSI	Parallel synchronous slave interface
PWR	Power controller
RAMECC	RAM ECC monitoring
RCC	Reset and clock controller
RIF	Resource isolation framework
RIFSC	Resource isolation framework security controller
RISAF	Resource isolation unit for address space protection
RNG	Random number generator
RTC	Real-time clock
SAI	Serial audio interface
SAES	Secure advanced encryption standard
SBS	System configuration, boot and security
SD	Secure digital
SDMMC	SD/SDIO/MultiMediaCard card host interface
SFDP	Serial flash discovery protocol
SMARTCARD	Smartcard IC
SMBUS	System management bus
SPI	Serial peripheral interface
SPDIFRX	S/PDIF audio protocol receiver interface
SRAM	SRAM external memory
SysTick	System tick timer
TIM	Advanced-control, general-purpose or basic timer
UART	Universal asynchronous receiver/transmitter

Acronym	Definition
UCPD	USB Type-C® and Power Delivery interface
USART	Universal synchronous receiver/transmitter
USB	Universal serial bus
VENC	Video encoder
VREFBUF	Voltage reference buffer
WWDG	Window watchdog
XIP	Execute in place
XSPI	Extended SPI interface
XSPIM	XSPI I/O manager

3 Overview of HAL drivers

The HAL drivers are designed to offer a rich set of APIs and to interact easily with the application upper layers.

Each driver consists of a set of functions covering the most common peripheral features. The development of each driver is driven by a common API which standardizes the driver structure, the functions, and the parameter names.

The HAL drivers include a set of driver modules, each module being linked to a standalone peripheral. However, in some cases, the module is linked to a peripheral functional mode. As an example, several modules exist for the USART peripheral: UART driver module, USART driver module, SMARTCARD driver module and IRDA driver module.

The HAL main features are the following:

- Cross-family portable set of APIs covering the common peripheral features as well as extension APIs in case of specific peripheral features.
- Three API programming models: polling, interrupt, and DMA.
- APIs are RTOS-compliant:
  - Fully reentrant APIs
  - Systematic usage of timeouts in polling mode
- Support of peripheral multi-instance allowing concurrent API calls for multiple instances of a given peripheral (such as USART1 or USART2)
- All HAL APIs implement a user-callback functions mechanism:
  - Peripheral Init/DeInit HAL APIs can call user-callback functions to perform peripheral system level Initialization/De-Initialization (clock, GPIOs, interrupt, DMA)
  - Peripherals interrupt events
  - Error events
- Object locking mechanism: safe hardware access to prevent multiple spurious accesses to shared resources.
- Timeout used for all blocking processes: the timeout can be a simple counter or a timebase.

3.1 HAL and user-application files

3.1.1 HAL driver files

HAL drivers are composed of the following set of files:

Table 2. HAL driver files

File	Description
stm32n6xx_hal_ppp.c	Main peripheral/module driver file It includes the APIs that are common to all STM32 devices. *Example: stm32n6xx_hal_adc.c, stm32n6xx_hal_irda.c.*
stm32n6xx_hal_ppp.h	Header file of the main driver C file It includes common data, handle and enumeration structures, define statements and macros, as well as the exported generic APIs. *Example:stm32n6xx_hal_adc.h,stm32n6xx_hal_irda.h.*
stm32n6xx_hal_ppp_ex.c	Extension file of a peripheral/module driver. It includes the specific APIs for a given part number or family, as well as the newly defined APIs that overwrite the default generic APIs if the internal process is implemented in a different way. *Example:stm32n6xx_hal_adc_ex.c,stm32n6xx_hal_flash_ex.c.*
stm32n6xx_hal_ppp_ex.h	Header file of the extension C file It includes the specific data and enumeration structures, define statements, and macros, as well as the exported device part number specific APIs *Example: stm32n6xx_hal_adc_ex.h,stm32n6xx_hal_flash_ex.h.*
stm32n6xx_hal.c	This file is used for HAL initialization and contains VREFBUF, SBS, DBGMCU, remap and time delay based on SysTick APIs.
stm32n6xx_hal.h	stm32n6xx_hal.c header file
stm32n6xx_hal_msp_template.c	Template file to be copied to the user application folder It contains the MSP initialization and deinitialization (main routine and callbacks) of the peripheral used in the user application.
stm32n6xx_hal_conf_template.h	Template file allowing the drivers to be customized for a given application
stm32n6xx_hal_def.h	Common HAL resources such as common define statements, enumerations, structures, and macros

3.1.2 User-application files

The minimum files required to build an application using the HAL are listed in the table below:

Table 3. User-application files

File	Description
system_stm32n6xx.c	This file contains SystemInit() that is called at startup just after reset and before branching to the main program. It does not configure the system clock at startup (contrary to the standard library). This is to be done using the HAL APIs in the user files. It allows the relocation of the vector table in internal SRAM.
startup_stm32n657xx/n655xx/ n647xx/n645xx.s	Toolchain specific file that contains reset handler and exception vectors. For some toolchains, it allows the adaptation of the stack/heap size to fit the application requirements.
stm32n657xx_axisram.icf	This file is the linker file for the EWARM toolchain. Its main function is the adaptation of the stack/heap size to fit the application requirements in internal SRAM 1.
stm32n657xx_axisram2_fsbl.icf	This file is the linker file for the EWARM toolchain. Its main function is the adaptation of the stack/heap size to fit the application requirements in internal SRAM 2, at the location where the FSBL binary is copied by the boot ROM.

File	Description
stm32n6xx_hal_msp.c	This file contains the MSP initialization and deinitialization (main routine and callbacks) of the peripheral used in the user application.
stm32n6xx_hal_conf.h	This file allows the HAL drivers to be customized for a specific application. It is not mandatory to modify this configuration. The application can use the default configuration without any modification.
stm32n6xx_it.c/.h	This file contains the exceptions handler and peripherals interrupt service routine, and calls HAL_IncTick() at regular time intervals to increment a local variable (declared in *stm32n6xx*_hal.c) used as HAL timebase. By default, this function is called each 1 ms in SysTick ISR. The PPP_IRQHandler() routine must call HAL_PPP_IRQHandler() if an interrupt-based process is used within the application.
main.c/.h	This file contains the main program routine, mainly: • Call to HAL_Init() • assert_failed() implementation • system clock configuration • peripheral HAL initialization and user application code.

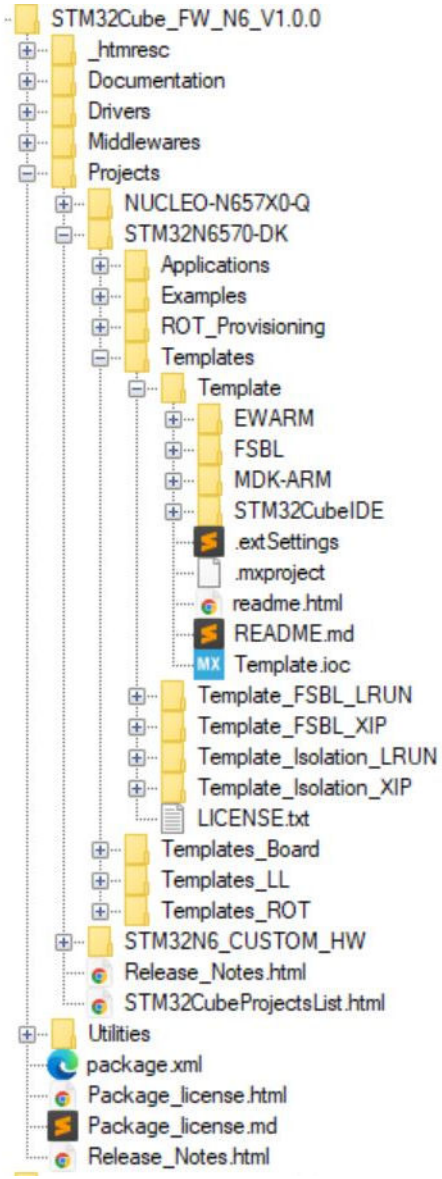
The STM32Cube package comes with ready-to-use project templates, one for each supported board. Each project contains the files listed above and a preconfigured project for the supported toolchains.

Each project template provides an empty main loop function and can be used as a starting point to get familiar with project settings for STM32Cube. Its features are the following:

- It contains the sources of HAL, CMSIS, and BSP drivers which are the minimal components to develop a code on a given board.
- It contains the include paths for all the firmware components.
- It defines the STM32 device supported, and configures the CMSIS and HAL drivers accordingly.
- It provides ready to use user files preconfigured as defined below:
  - HAL is initialized
  - SysTick ISR implemented for HAL_GetTick()
  - System clock configured with the selected device frequency.

Note: If an existing project is copied to another location, then include paths must be updated.

Figure 1. **Example of project template**



3.2 HAL data structures

Each HAL driver can contain the following data structures:

- Peripheral handle structures
- Initialization and configuration structures
- Specific process structures.

3.2.1 Peripheral handle structures

The APIs have a modular generic multi-instance architecture that enables working with several peripheral instances simultaneously.

PPP_HandleTypeDef *handle is the main structure that is implemented in the HAL drivers. It handles the peripheral/module configuration and registers and embeds all the structures and variables needed to follow the peripheral device flow.

The peripheral handle is used for the following purposes:

- Multi-instance support: each peripheral/module instance has its own handle. As a result instance resources are independent.
- Peripheral process intercommunication: the handle is used to manage shared data resources between the process routines.
  Example: global pointers, DMA handles, state machine.
- Storage: this handle is also used to manage global variables within a given HAL driver.

An example of peripheral structure is shown below:

```
typedef struct
{
USART_TypeDef *Instance; /* USART registers base address */
USART_InitTypeDef Init; /* Usart communication parameters */
uint8_t *pTxBuffPtr;/* Pointer to Usart Tx transfer Buffer */
uint16_t TxXferSize; /* Usart Tx Transfer size */
__IO uint16_t TxXferCount;/* Usart Tx Transfer Counter */
uint8_t *pRxBuffPtr;/* Pointer to Usart Rx transfer Buffer */
uint16_t RxXferSize; /* Usart Rx Transfer size */
__IO uint16_t RxXferCount; /* Usart Rx Transfer Counter */
DMA_HandleTypeDef *hdmatx; /* Usart Tx DMA Handle parameters */
DMA_HandleTypeDef *hdmarx; /* Usart Rx DMA Handle parameters */
HAL_LockTypeDef Lock; /* Locking object */
__IO HAL_USART_StateTypeDef State; /* Usart communication state */
__IO HAL_USART_ErrorTypeDef ErrorCode;/* USART Error code */
}USART_HandleTypeDef;
```

Note:

1. *The multi-instance feature implies that all the APIs used in the application are reentrant and avoid using global variables because subroutines can fail to be reentrant if they rely on a global variable to remain unchanged but that variable is modified when the subroutine is recursively invoked. For this reason, the following rules are respected:*
   - *Reentrant code does not hold any static (or global) nonconstant data: reentrant functions can work with global data. For example, a reentrant interrupt service routine can grab a piece of hardware status to work with (for example serial port read buffer) which is not only global, but volatile. Still, typical use of static variables and global data is not advised, in the sense that only atomic read-modify-write instructions should be used in these variables. It should not be possible for an interrupt or signal to occur during the execution of such an instruction.*
   - *Reentrant code does not modify its own code.*
2. *When a peripheral can manage several processes simultaneously using the DMA (full duplex case), the DMA interface handle for each process is added in the PPP_HandleTypeDef.*
3. *For the shared and system peripherals, no handle or instance object is used. The peripherals concerned by this exception are the following:*
   - *GPIO*
   - *SYSTICK*
   - *NVIC*
   - *PWR*
   - *RCC*

3.2.2 Initialization and configuration structure

These structures are defined in the generic driver header file when it is common to all part numbers. When they can change from one part number to another, the structures are defined in the extension header file for each part number.

```
typedef struct
{
uint32_t BaudRate;      /*!< This member configures the Usart communication
                             baud rate. The baud rate is computed using the
                             following formula: Baud Rate Register[15:4]=((2 *
                             fclk_pres)/((huart->Init.BaudRate)))[15:4] BaudRate
                             Register[3]=0 Baud Rate Register[2:0]=(((2*
                             fclk_pres)/((huart->Init.BaudRate)))[3:0]) >> 1
                             where fclk_pres is the USART input clock frequency
                             (fclk) divided by a prescaler. The oversampling by 8
                             is systematically applied to achieve high baudrates.*/
uint32_t WordLength;    /*!< Specifies the number of data bits transmitted or
                             received in a frame.This parameter can be a value of
                             USAR TEx_Word_Length. */
uint32_t StopBits;      /*!< Specifies the number of stop bits transmitted. This
                             parameter can be a value of USART_Stop_Bits. */
uint32_t Parity;        /*!< Specifies the parity mode. This parameter can be a value
                             of USART_Parity @note When parity is enabled, the
                             computed parity is inserted at the MSB position of the
                             transmitted data (9th bit when the word length is set to
                             9 data bits; 8th bit when the word length is set to 8
                             data bits). */
uint32_t Mode;          /*!< Specifies whether the Receive or Transmit mode is
                             enabled or disabled. This parameter can be a value of
                             USART_Mode. */
uint32_t CLKPolarity;   /*!< Specifies the steady state of the serial clock. This
                             parameter can be a value of USART_Clock_Polarity. */ uint32_t CL
KPhase;      /*!< Specifies the clock transition on which the bit capture
                             is made. This parameter can be a value of
                             USART_Clock_Phase. */
uint32_t CLKLastBit;    /*!< Specifies whether the clock pulse corresponding to the
                             last transmitted data bit (MSB) has to be output on the
                             SCLK pin in synchronous mode. This parameter can be a
                             value of USART_Last_Bit. */
uint32_t ClockPrescaler; /*!< Specifies the prescaler value used to divide the USART
                             clock source. This parameter can be a value of
                             USART_ClockPrescaler. */
} USART_InitTypeDef;
```

Note: The config structure is used to initialize the submodules or subinstances. See below example:

```
HAL_ADC_ConfigChannel (ADC_HandleTypeDef* hadc, const ADC_ChannelConfTypeDef *pConfig)
```

3.2.3 Specific process structures

The specific process structures are used for specific process (common APIs). They are defined in the generic driver header file.

Example:

```
HAL_PPP_Process (PPP_HandleTypeDef* hadc,PPP_ProcessConfig* sConfig)
```

3.3 API classification

The HAL APIs are classified into the following categories:

- **Generic APIs:** common generic APIs applying to all STM32 devices. These APIs are consequently present in the generic HAL driver files of all STM32 microcontrollers.

```
HAL_StatusTypeDef HAL_ADC_Init(ADC_HandleTypeDef* hadc);
HAL_StatusTypeDef HAL_ADC_DeInit(ADC_HandleTypeDef *hadc);
HAL_StatusTypeDef HAL_ADC_Start(ADC_HandleTypeDef* hadc);
HAL_StatusTypeDef HAL_ADC_Stop(ADC_HandleTypeDef* hadc);
HAL_StatusTypeDef HAL_ADC_Start_IT(ADC_HandleTypeDef* hadc);
HAL_StatusTypeDef HAL_ADC_Stop_IT(ADC_HandleTypeDef* hadc);
void HAL_ADC_IRQHandler(ADC_HandleTypeDef* hadc);
```

- **Extension APIs:**
  This set of API is divided into two subcategories :
  - **Family specific APIs:** APIs applying to a given family. They are located in the extension HAL driver file (see example below related to the ADC).

```
HAL_StatusTypeDef HAL_ADCEx_Calibration_Start(ADC_HandleTypeDef* hadc,uint32_t Sin
gleDiff);
uint32_t HAL_ADCEx_Calibration_GetValue(ADC_HandleTypeDef* hadc, uint32_t SingleDi
ff);
```

  - **Feature/Device specific APIs**: these APIs are implemented in the extension file and delimited by specific define statements relative to CMSIS bits or features and depending on device part number.

Note: The data structure related to the specific APIs is delimited by the device part number define statement. It is located in the corresponding extension header C file.

The following table summarizes the location of the different categories of HAL APIs in the driver files.

Table 4. API classification

	Generic file	Extension file
Common APIs	X	X
Family specific APIs	-	X
Device specific APIs	-	X

Note: Family specific APIs are only related to a given family. This means that if a specific API is implemented in another family, and the arguments of this latter family are different, additional structures and arguments might need to be added.

The IRQ handlers are used for common and family specific processes.

3.4 Devices supported by HAL drivers

Table 5. List of devices supported by HAL/LL drivers

Peripheral/Module	STM32N657/655xx	STM32N647xx/645xx
stm32n6xx_hal.c	Yes	Yes
stm32n6xx_hal_adc.c	Yes	Yes
stm32n6xx_hal_adc_ex.c	Yes	Yes
stm32n6xx_hal_bsec.c	Yes	Yes
stm32n6xx_hal_cacheaxi.c	Yes	Yes
stm32n6xx_hal_cortex.c	Yes	Yes
stm32n6xx_hal_crc.c	Yes	Yes
stm32n6xx_hal_crc_ex.c	Yes	Yes
stm32n6xx_hal_cryp.c	Yes	no
stm32n6xx_hal_cryp_ex.c	Yes	no
stm32n6xx_hal_dcmipp.c	Yes	Yes
stm32n6xx_hal_dcmipp.c	Yes	Yes
stm32n6xx_hal_dma.c	Yes	Yes
stm32n6xx_hal_dma_ex.c	Yes	Yes
stm32n6xx_hal_dma2d.c	Yes	Yes
stm32n6xx_hal_dts.c	Yes	Yes
stm32n6xx_hal_eth.c	Yes	Yes
stm32n6xx_hal_eth_ex.c	Yes	Yes
stm32n6xx_hal_exti.c	Yes	Yes
stm32n6xx_hal_fdcan.c	Yes	Yes
stm32n6xx_hal_gfxmmu.c	Yes	Yes
stm32n6xx_hal_gfxtim.c	Yes	Yes
stm32n6xx_hal_gpio.c	Yes	Yes
stm32n6xx_hal_gpu2d.c	no	no
stm32n6xx_hal_hash.c	Yes	Yes
stm32n6xx_hal_hcd.c	Yes	Yes
stm32n6xx_hal_i2c.c	Yes	Yes
stm32n6xx_hal_i2c_ex.c	Yes	Yes
stm32n6xx_hal_i2s.c	Yes	Yes
stm32n6xx_hal_i2s_ex.c	Yes	Yes
stm32n6xx_hal_i3c.c	Yes	Yes
stm32n6xx_hal_icache.c	Yes	Yes
stm32n6xx_hal_irda.c	Yes	Yes
stm32n6xx_hal_iwdg.c	Yes	Yes
stm32n6xx_hal_jpeg.c	Yes	Yes
stm32n6xx_hal_lptim.c	Yes	Yes
stm32n6xx_hal_ltdc.c	Yes	Yes
stm32n6xx_hal_ltdc_ex.c	Yes	Yes

Peripheral/Module	STM32N657/655xx	STM32N647xx/645xx
stm32n6xx_hal_mce.c	Yes	No
stm32n6xx_hal_mdf.c	Yes	Yes
stm32n6xx_hal_mdios.c	Yes	Yes
stm32n6xx_hal_mmc.c	Yes	Yes
stm32n6xx_hal_mmc_ex.c	Yes	Yes
stm32n6xx_hal_msp_template.c	Yes	Yes
stm32n6xx_hal_nand.c	Yes	Yes
stm32n6xx_hal_nor.c	Yes	Yes
stm32n6xx_hal_pcd.c	Yes	Yes
stm32n6xx_hal_pcd_ex.c	Yes	Yes
stm32n6xx_hal_pka.c	Yes	Yes
stm32n6xx_hal_pssi.c	Yes	Yes
stm32n6xx_hal_pwr.c	Yes	Yes
stm32n6xx_hal_pwr_ex.c	Yes	Yes
stm32n6xx_hal_ramcfg.c	Yes	Yes
stm32n6xx_hal_rcc.c	Yes	Yes
stm32n6xx_hal_rcc_ex.c	Yes	Yes
stm32n6xx_hal_rng.c	Yes	Yes
stm32n6xx_hal_rng_ex.c	Yes	Yes
stm32n6xx_hal_rtc.c	Yes	Yes
stm32n6xx_hal_rtc_ex.c	Yes	Yes
stm32n6xx_hal_sai.c	Yes	Yes
stm32n6xx_hal_sai_ex.c	Yes	Yes
stm32n6xx_hal_sd.c	Yes	Yes
stm32n6xx_hal_sd_ex.c	Yes	Yes
stm32n6xx_hal_smartcard.c	Yes	Yes
stm32n6xx_hal_smartcard_ex.c	Yes	Yes
stm32n6xx_hal_smbus.c	Yes	Yes
stm32n6xx_hal_smbus_ex.c	Yes	Yes
stm32n6xx_hal_spdifrx.c	Yes	Yes
stm32n6xx_hal_spi.c	Yes	Yes
stm32n6xx_hal_spi_ex.c	Yes	Yes
stm32n6xx_hal_sram.c	Yes	Yes
stm32n6xx_hal_tim.c	Yes	Yes
stm32n6xx_hal_tim_ex.c	Yes	Yes
stm32n6xx_hal_uart.c	Yes	Yes
stm32n6xx_hal_uart_ex.c	Yes	Yes
stm32n6xx_hal_usart.c	Yes	Yes
stm32n6xx_hal_usart_ex.c	Yes	Yes
stm32n6xx_hal_wwdg.c	Yes	Yes
stm32n6xx_hal_xspi.c	Yes	Yes

Peripheral/Module	STM32N657/655xx	STM32N647xx/645xx
stm32n6xx_ll_adc.c	Yes	Yes
stm32n6xx_ll_bus.h	Yes	Yes
stm32n6xx_ll_cortex.h	Yes	Yes
stm32n6xx_ll_crc.c	Yes	Yes
stm32n6xx_ll_dlyb.c	Yes	Yes
stm32n6xx_ll_dma.c	Yes	Yes
stm32n6xx_ll_dma2d.c	Yes	Yes
stm32n6xx_ll_exti.c	Yes	Yes
stm32n6xx_ll_fmc.c	Yes	Yes
stm32n6xx_ll_gpio.c	Yes	Yes
stm32n6xx_ll_i2c.c	Yes	Yes
stm32n6xx_ll_i3c.c	Yes	Yes
stm32n6xx_ll_lptim.c	Yes	Yes
stm32n6xx_ll_lpuart.c	Yes	Yes
stm32n6xx_ll_pka.c	Yes	Yes
stm32n6xx_ll_pwr.c	Yes	Yes
stm32n6xx_ll_rcc.c	Yes	Yes
stm32n6xx_ll_rng.c	Yes	Yes
stm32n6xx_ll_rtc.c	Yes	Yes
stm32n6xx_ll_sdmmc.c	Yes	Yes
stm32n6xx_ll_system.h	Yes	Yes
stm32n6xx_ll_spi.c	Yes	Yes
stm32n6xx_ll_tim.c	Yes	Yes
stm32n6xx_ll_ucpd.c	Yes	Yes
stm32n6xx_ll_usart.c	Yes	Yes
stm32n6xx_ll_usb.c	Yes	Yes
stm32n6xx_ll_utils.c	Yes	Yes
stm32n6xx_ll_venc.c	Yes	Yes
stm32n6xx_util_i3c.c	Yes	Yes

3.5 HAL driver rules

3.5.1 HAL API naming rules

The following naming rules are used in HAL drivers:

Table 6. HAL API naming rules

	Generic	Family specific APIs	Device specific APIs
File names	*stm32n6xx_hal_ppp (c/h)*	*stm32n6xx_hal_ppp_ex (c/h)*	*stm32n6xx_ hal_ppp_ex (c/h)*
Module name	*HAL_PPP_ MODULE*		
Function name	*HAL_PPP_Function* *HAL_PPP_FeatureFunction_MODE*	*HAL_PPPEx_Function* *HAL_PPPEx_FeatureFunction_MODE*	*HAL_PPPEx_Function* *HAL_PPPEx_FeatureFunction_MODE*
Handle name	*PPP_HandleTypedef*	*NA*	*NA*
Init structure name	*PPP_InitTypeDef*	*NA*	*PPP_InitTypeDef*
Enum name	*HAL_PPP_StructnameTypeDef*	*NA*	*NA*

- The **PPP** prefix refers to the peripheral functional mode and not to the peripheral itself. For example, if the USART, PPP can be USART, IRDA, UART, or SMARTCARD depending on the peripheral mode.
- The constants used in one file are defined within this file. A constant used in several files is defined in a header file. All constants are written in uppercase, except for peripheral driver function parameters.
- typedef variable names should be suffixed with _TypeDef.
- Registers are considered as constants. In most cases, their name is in uppercase and uses the same acronyms as in the STM32N6 reference manuals.
- Peripheral registers are declared in the PPP_TypeDef structure (for example ADC_TypeDef) in the stm32n6xxx.h header file:
stm32n6xxx.h corresponds to stm32n657xx.h, stm32n655xx.h, stm32n647xx.h and stm32n645xx.h
- Peripheral function names are prefixed by HAL_, then the corresponding peripheral acronym in uppercase followed by an underscore. The first letter of each word is in uppercase (for example HAL_UART_Transmit()). Only one underscore is allowed in a function name to separate the peripheral acronym from the rest of the function name.
- The structure containing the PPP peripheral initialization parameters are named PPP_InitTypeDef (for example ADC_InitTypeDef).
- The structure containing the specific configuration parameters for the PPP peripheral are named PPP_xxxxConfTypeDef (for example ADC_ChannelConfTypeDef).
- Peripheral handle structures are named PPP_HandleTypedef (e.g DMA_HandleTypeDef)
- The functions used to initialize the PPP peripheral according to parameters specified in PPP_InitTypeDef are named HAL_PPP_Init (for example HAL_TIM_Init()).
- The functions used to reset the PPP peripheral registers to their default values are named HAL_PPP_DeInit (for example HAL_TIM_DeInit()).
- The **MODE** suffix refers to the process mode, which can be polling, interrupt or DMA. As an example, when the DMA is used in addition to the native resources, the function should be called:
HAL_PPP_Function_DMA().
- The **Feature** prefix should refer to the new feature.
Example: *HAL_ADCEx_InjectedStart()*() refers to the injection mode.

3.5.2 HAL general naming rules

- For the shared and system peripherals, no handle or instance object is used. This rule applies to the following peripherals:
  - GPIO
  - SYSTICK
  - NVIC
  - RCC

Example: The *HAL_GPIO_Init()* requires only the GPIO address and its configuration parameters.

```
HAL_StatusTypeDef HAL_GPIO_Init (GPIO_TypeDef* GPIOx, GPIO_InitTypeDef *Init)
{
/*GPIO Initialization body */
}
```

- The macros that handle interrupts and specific clock configurations are defined in each peripheral/module driver. These macros are exported in the peripheral driver header files so that they can be used by the extension file. The list of these macros is defined below:

Note: This list is not exhaustive and other macros related to peripheral features can be added, so that they can be used in the user application.

Table 7. Macros handling interrupts and specific clock configurations

Macros	Description
__HAL_PPP_ENABLE_IT(__HANDLE__, __INTERRUPT__)	Enables a specific peripheral interrupt
__HAL_PPP_DISABLE_IT(__HANDLE__, __INTERRUPT__)	Disables a specific peripheral interrupt
__HAL_PPP_GET_IT (__HANDLE__, __ INTERRUPT __)	Gets a specific peripheral interrupt status
__HAL_PPP_CLEAR_IT (__HANDLE__, __ INTERRUPT __)	Clears a specific peripheral interrupt status
__HAL_PPP_GET_FLAG (__HANDLE__, __FLAG__)	Gets a specific peripheral flag status
__HAL_PPP_CLEAR_FLAG (__HANDLE__, __FLAG__)	Clears a specific peripheral flag status
__HAL_PPP_ENABLE(__HANDLE__)	Enables a peripheral
__HAL_PPP_DISABLE(__HANDLE__)	Disables a peripheral
__HAL_PPP_XXXX (__HANDLE__, __PARAM__)	Specific PPP HAL driver macro
__HAL_PPP_GET_ IT_SOURCE (__HANDLE__, __ INTERRUPT __)	Checks the source of the specified interrupt

- NVIC and SYSTICK are two Arm® Cortex® core features. The APIs related to these features are located in the stm32n6xx_hal_cortex.c file.
- The PPP handles are valid before using the HAL_PPP_Init() API. The init function performs a check before modifying the handle fields.

```
 HAL_PPP_Init(PPP_HandleTypeDef)
if(hppp == NULL)
 {
 return HAL_ERROR;
 }
```

- The macros defined below are used:
  - Conditional macro:

```
#define ABS(x) (((x) > 0) ? (x) : -(x))
```

  - Pseudocode macro (multiple instructions macro):

```
#define __HAL_LINKDMA(__HANDLE__, __PPP_DMA_FIELD_, __DMA_HANDLE_) \
do{ \
(__HANDLE__)->__PPP_DMA_FIELD_ = &(__DMA_HANDLE_); \
(__DMA_HANDLE_).Parent = (__HANDLE__); \
} while(0)
```

3.5.3 HAL interrupt handler and callback functions

Besides the APIs, HAL peripheral drivers include:

- HAL_PPP_IRQHandler() peripheral interrupt handler that should be called from stm32n6xx_it.c
- User callback functions.

The user callback functions are defined as empty functions with “weak” attribute. They have to be defined in the user code.

There are three types of user callbacks functions:

- Peripheral system level initialization/ de-Initialization callbacks: HAL_PPP_MspInit() and HAL_PPP_MspDeInit
- Process complete callbacks: HAL_PPP_ProcessCpltCallback
- Error callback: HAL_PPP_ErrorCallback.

Table 8. Callback functions

Callback functions	Example
HAL_PPP_MspInit() / _DeInit()	Example: HAL_USART_MspInit() Called from HAL_PPP_Init() API function to perform peripheral system level initialization (GPIOs, clock, DMA, interrupt)
HAL_PPP_ProcessCpltCallback	Example: HAL_USART_TxCpltCallback Called by the peripheral or DMA interrupt handler when the process completes
HAL_PPP_ErrorCallback	Example: HAL_USART_ErrorCallback Called by the peripheral or DMA interrupt handler when an error occurs

3.6 HAL generic APIs

The generic APIs provide common generic functions applying to all STM32 devices. They are composed of four APIs groups:

- **Initialization and deinitialization functions:**HAL_PPP_Init(), HAL_PPP_DeInit()
- **I/O operation functions**: HAL_PPP_Read(), HAL_PPP_Write(),HAL_PPP_Transmit(), HAL_PPP_Receive()
- **Control functions**: HAL_PPP_Set (), HAL_PPP_Get ().
- **State and error functions**: HAL_PPP_GetState (), HAL_PPP_GetError ().

For some peripheral/module drivers, these groups are modified depending on the peripheral/module implementation.

Example: in the timer driver, the API grouping is based on timer features (such as PWM, OC, and IC).

The initialization and deinitialization functions initialize a given peripheral and configure the low-level resources, mainly clocks, GPIO, alternate functions (AF) and possibly DMA and interrupts. The *HAL_DeInit()* function restores the peripheral default state, frees the low-level resources and removes any direct dependency with the hardware.

The I/O operation functions perform a row access to the peripheral payload data in write and read modes.

The control functions are used to dynamically change the peripheral configuration and set another operating mode.

The peripheral state and errors functions retrieve in runtime the peripheral and data flow states, and identifying the type of errors that occurred. The example below is based on the ADC peripheral. The list of generic APIs is not exhaustive. It is only given as an example.

Table 9. HAL generic APIs

Function group	Common API name	Description
Initialization group	*HAL_ADC_Init()*	This function initializes the peripheral and configures the low-level resources (such as clocks, GPIO, AF)
	HAL_ADC_DeInit()	This function restores the peripheral default state, frees the low-level resources and removes any direct dependency with the hardware.
I/O operation group	*HAL_ADC_Start ()*	This function starts ADC conversions when the polling method is used
	HAL_ADC_Stop ()	This function stops ADC conversions when the polling method is used
	HAL_ADC_PollForConversion()	This function waits for the end of conversions when the polling method is used. In this case, a timeout value is specified by the user according to the application.
	HAL_ADC_Start_IT()	This function starts ADC conversions when the interrupt method is used
	HAL_ADC_Stop_IT()	This function stops ADC conversions when the interrupt method is used
	HAL_ADC_IRQHandler()	This function handles ADC interrupt requests
	HAL_ADC_ConvCpltCallback()	Callback function called in the IT subroutine to indicate the end of the current process or when a DMA transfer has been completed
	HAL_ADC_ErrorCallback()	Callback function called in the IT subroutine if a peripheral error or a DMA transfer error occurred
Control group	*HAL_ADC_ConfigChannel()*	This function configures the selected ADC regular channel, the corresponding rank in the sequencer and the sample time
	HAL_ADC_AnalogWDGConfig	This function configures the analog watchdog for the selected ADC
State and errors group	*HAL_ADC_GetState()*	This function gets in runtime the peripheral and the data flow states.
	HAL_ADC_GetError()	This fuction gets in runtime the error that occurred during an IT routine

3.7 HAL extension APIs

3.7.1 HAL extension model overview

The extension APIs provide specific functions or overwrite modified APIs for a specific family (series) or specific part number within the same family.

The extension model consists of an additional file, stm32n6xx_hal_ppp_ex.c, that includes all the specific functions and define statements (stm32n6xx_hal_ppp_ex.h) for a given part number.

Below is an example based on the ADC peripheral:

Table 10. HAL extension APIs

Function group	Common API name
HAL_ADCEx_CalibrationStart()	This function is used to start the automatic ADC calibration
HAL_ADCEx_InjectedStart	This function is used to start the conversion of injected groups.

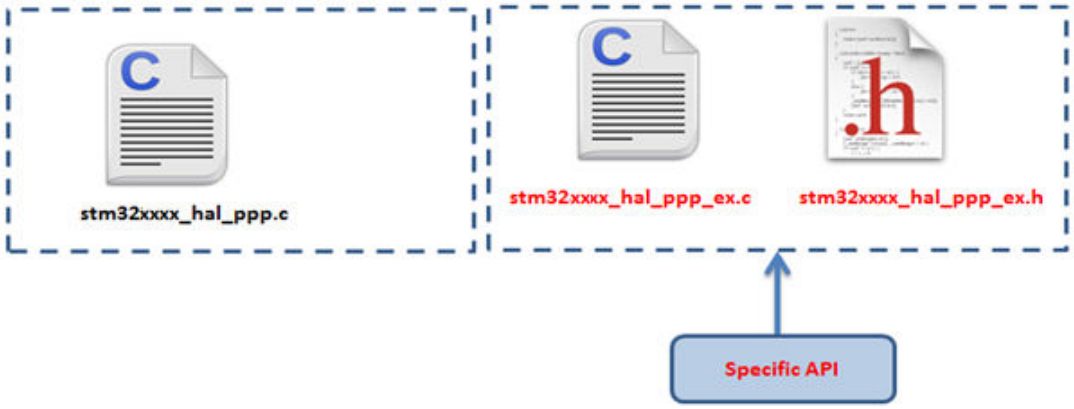
3.7.2 HAL extension model cases

The specific peripheral features can be handled by the HAL drivers in five different ways. They are described below.

Adding a part number-specific function

When a new feature specific to a given device is required, the new APIs are added in the stm32n6xx_hal_ppp_ex.c extension file. They are named HAL_PPPEx_Function().

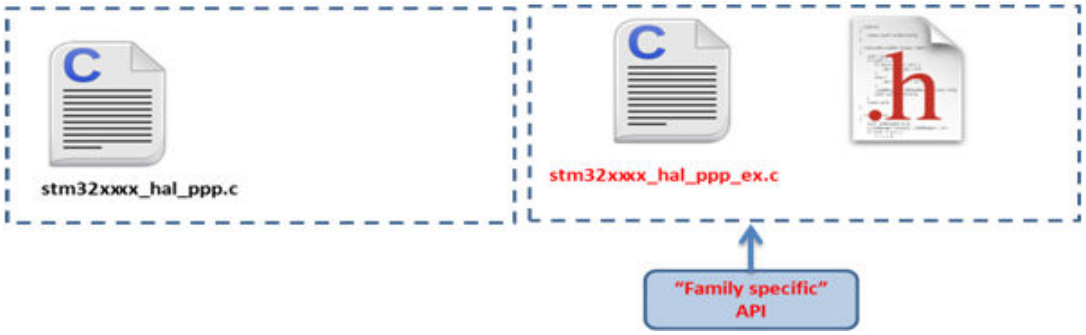
Figure 2. Adding device-specific functions



Adding a family-specific function

In this case, the API is added in the extension driver C file and named HAL_PPPEx_Function ().

Figure 3. Adding family-specific functions

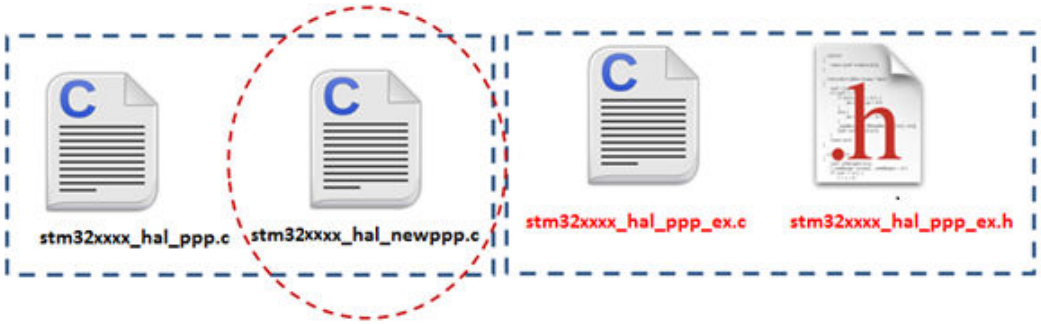


Adding a new peripheral (specific to a device belonging to a given family)

When a peripheral which is available only in a specific device is required, the APIs corresponding to this new peripheral/module (newPPP) are added in a new stm32n6xx_hal_newppp.c. However, the inclusion of this file is selected in the stm32n6xx_hal_conf.h using the macro:

```
#define HAL_NEWPPP_MODULE_ENABLED
```

Figure 4. **Adding new peripherals**

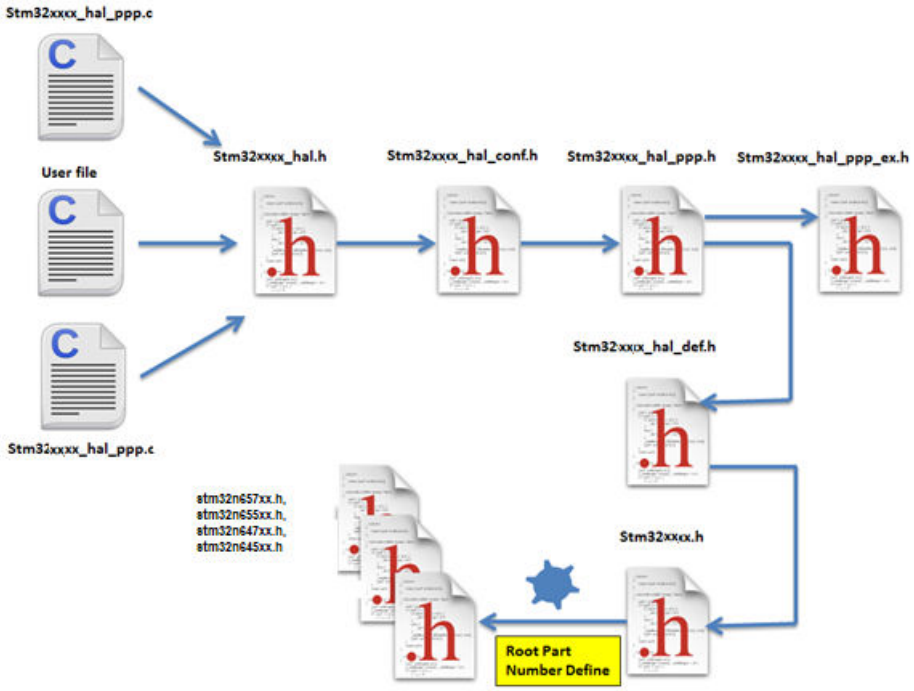


Example: stm32n6xx_hal_adc.c/h

3.8 File inclusion model

The header of the common HAL driver file (stm32n6xx_hal.h) includes the common configurations for the whole HAL library. It is the only header file that is included in the user sources and the HAL C sources files to be able to use the HAL resources.

Figure 5. **File inclusion model**



A PPP driver is a standalone module which is used in a project. The user must enable the corresponding USE_HAL_PPP_MODULE define statement in the configuration file.

```
/**********************************************************************
* @file stm32n6xx_hal_conf.h
* @author MCD Application Team
* @version VX.Y.Z * @date dd-mm-yyyy
* @brief This file contains the modules to be used
**********************************************************************
(…)
#define USE_HAL_USART_MODULE
#define USE_HAL_IRDA_MODULE
#define USE_HAL_DMA_MODULE
```

```
#define USE_HAL_RCC_MODULE
(…)
```

3.9 HAL common resources

The common HAL resources, such as common define enumerations, structures, and macros, are defined in *stm32n6xx_hal_def.h.*The main common define enumeration is *HAL_StatusTypeDef*.

- **HAL status**
  The HAL status is used by almost all HAL APIs, except for boolean functions and the IRQ handler. It returns the status of the current API operations. It has four possible values as described below:

```
Typedef enum
{
HAL_OK = 0x00,
HAL_ERROR = 0x01,
HAL_BUSY = 0x02,
HAL_TIMEOUT = 0x03
} HAL_StatusTypeDef;
```

- **HAL locked**
  The HAL lock is used by all HAL APIs to prevent accessing by accident shared resources.

```
typedef enum
{
HAL_UNLOCKED = 0x00, /*!<Resources unlocked */
HAL_LOCKED = 0x01 /*!< Resources locked */
} HAL_LockTypeDef;
```

  In addition to common resources, the stm32n6xx_hal_def.h file calls the stm32n6xx.h file in the CMSIS library to get the data structures and the address mapping for all peripherals:

  – Declarations of peripheral registers and bits definition.
  – Macros to access peripheral registers hardware (such as write register or read register).

- **Common macros**
  – Macro defining HAL_MAX_DELAY

```
#define HAL_MAX_DELAY 0xFFFFFFFF
```

  – Macro linking a PPP peripheral to a DMA structure pointer:

```
#define __HAL_LINKDMA(__HANDLE__, __PPP_DMA_FIELD_, __DMA_HANDLE_) \
do{ \
  (__HANDLE__)->__PPP_DMA_FIELD_ = &(__DMA_HANDLE_); \
  (__DMA_HANDLE_).Parent = (__HANDLE__); \
} while(0)
```

3.10 HAL configuration

The configuration file, stm32n6xx_hal_conf.h, performs the customization of the drivers for the user application. Modifying this configuration is not mandatory: the application can use the default configuration without any modification.

To configure these parameters, the user should enable, disable, or modify some options by uncommenting, commenting, or modifying the values of the related define statements as described in the table below:

Table 11. Define statements used for HAL configuration

Configuration item	Description	Default Value
HSE_VALUE	Value of the external oscillator (HSE) expressed in Hz. The user must adjust this define statement when using a different crystal value.	48 000 000 Hz
HSE_STARTUP_TIMEOUT	Timeout for HSE startup, expressed in ms	100 ms
HSI_VALUE	Value of the internal oscillator (HSI) expressed in Hz.	64 000 000 Hz
LSI_VALUE	Default value of the low-speed internal oscillator (LSI) expressed in Hz.	32000 Hz
LSE_VALUE	Value of the external oscillator (LSE) expressed in Hz. The user must adjust this define statement when using a different crystal value.	32768 Hz

Note: The stm32n6xx_hal_conf_template.h file is located in the HAL drivers Inc folder. It should be copied to the user folder, renamed and modified as described above.

By default, the values defined in the stm32n6xx_hal_conf_template.h file are the same as the ones used for the examples and demonstrations. All HAL include files are enabled so that they can be used in the user code without modifications.

3.11 HAL system peripheral handling

This section gives an overview of how the system peripherals are handled by the HAL drivers. The full API list is provided within each peripheral driver description section.

3.11.1 Clocks

Two main functions can be used to configure the system clock:

- HAL_RCC_OscConfig (RCC_OscInitTypeDef *RCC_OscInitStruct). This function configures/enables multiple clock sources (HSE, HSI, LSE, LSI, PLL).
- HAL_RCC_ClockConfig (RCC_ClkInitTypeDef *RCC_ClkInitStruct, uint32_t FLatency). This function
  - Selects the system clock source
  - Configures AHB, APB1, APB2, APB4, and APB5 clock dividers
  - Updates the SysTick configuration when the HCLK clock changes.

Some peripheral clocks are not derived from the system clock (such as RTC, USB). In this case, the clock configuration is performed by an extended API defined in stm32n6xx_hal_rcc_ex.c: *HAL_RCCEx_PeriphCLKConfig(RCC_PeriphCLKInitTypeDef *PeriphClkInit)*.

Additional RCC HAL driver functions are available:

- HAL_RCC_DeInit() clock deinitialization function that returns clock configuration to the reset state
- Get clock functions that retrieve the various clock configurations (such as system clock)
- MCO configuration functions

A set of macros are defined in stm32n6xx_hal_rcc.h and stm32n6xx_hal_rcc_ex.h. They enable the execution of elementary operations on RCC block registers, such as peripheral clock gating/reset control:

- __HAL_PPP_CLK_ENABLE/__HAL_PPP_CLK_DISABLE to enable/disable the peripheral clock.
- __HAL_PPP_FORCE_RESET/__HAL_PPP_RELEASE_RESET to force/release peripheral reset
- __HAL_PPP_CLK_SLEEP_ENABLE/__HAL_PPP_CLK_SLEEP_DISABLE to enable/disable the peripheral clock during Sleep mode.
- __HAL_PPP_IS_CLK_ENABLED/__HAL_PPP_IS_CLK_DISABLED to query about the enabled/disabled status of the peripheral clock.
- __HAL_PPP_IS_CLK_SLEEP_ENABLED/__HAL_PPP_IS_CLK_SLEEP_DISABLED to query about the enabled/disabled status of the peripheral clock during Sleep mode.

3.11.2 GPIOs

GPIO HAL APIs are the following:

- HAL_GPIO_Init() / HAL_GPIO_DeInit()
- HAL_GPIO_ReadPin() / HAL_GPIO_WritePin()
- HAL_GPIO_TogglePin ().
- HAL_GPIO_LockPin()
- HAL_GPIO_ConfigPinAttributes()/HAL_GPIO_GetConfigPinAttributes()

In addition to standard GPIO modes (input, output, analog), the pin mode can be configured as EXTI with interrupt or event generation.

When selecting EXTI mode with interrupt generation, the user must call HAL_GPIO_EXTI_IRQHandler() from stm32n6xx_it.c and implement HAL_GPIO_EXTI_Callback().

The table below describes the GPIO_InitTypeDef structure field.

Table 12. Description of GPIO_InitTypeDef structure

Structure field	Description
Pin	Specifies the GPIO pins to be configured. Possible values: GPIO_PIN_x or GPIO_PIN_All, where x[0..15]
Mode	Specifies the operating mode for the selected pins: GPIO mode or EXTI mode. Possible values are: • GPIO mode – GPIO_MODE_INPUT: Input floating – GPIO_MODE_OUTPUT_PP: Output push-pull – GPIO_MODE_OUTPUT_OD: Output open drain – GPIO_MODE_AF_PP: Alternate function push-pull – GPIO_MODE_AF_OD: Alternate function open drain – GPIO_MODE_ANALOG: Analog mode • External interrupt mode – GPIO_MODE_IT_RISING: Rising edge trigger detection – GPIO_MODE_IT_FALLING: Falling edge trigger detection – GPIO_MODE_IT_RISING_FALLING: Rising/Falling edge trigger detection • External event mode – GPIO_MODE_EVT_RISING: Rising edge trigger detection – GPIO_MODE_EVT_FALLING: Falling edge trigger detection – GPIO_MODE_EVT_RISING_FALLING: Rising/Falling edge trigger detection
Pull	Specifies the pull-up or pull-down activation for the selected pins. Possible values are: GPIO_NOPULL GPIO_PULLUP GPIO_PULLDOWN
Speed	Specifies the speed for the selected pins Possible values are: GPIO_SPEED_FREQ_LOW GPIO_SPEED_FREQ_MEDIUM GPIO_SPEED_FREQ_HIGH GPIO_SPEED_FREQ_VERY_HIGH

Please find below typical GPIO configuration examples:

- Configuring GPIOs as output push-pull to drive external LEDs:

```
GPIO_InitStruct.Pin = GPIO_PIN_12 | GPIO_PIN_13 | GPIO_PIN_14 | GPIO_PIN_15;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_PULLUP;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_MEDIUM;
HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);
```

- Configuring PA0 as an external interrupt with falling edge sensitivity:

```
GPIO_InitStructure.Mode = GPIO_MODE_IT_FALLING;
GPIO_InitStructure.Pull = GPIO_NOPULL;
GPIO_InitStructure.Pin = GPIO_PIN_0;
HAL_GPIO_Init(GPIOA, &GPIO_InitStructure);
```

3.11.3 Cortex® NVIC and SysTick timer

The Cortex® HAL driver, stm32n6xx_hal_cortex.c, provides APIs to handle NVIC and SysTick. The supported APIs include:

- HAL_NVIC_SetPriority()/ HAL_NVIC_SetPriorityGrouping()
- HAL_NVIC_GetPriority() / HAL_NVIC_GetPriorityGrouping()

- HAL_NVIC_EnableIRQ()/HAL_NVIC_DisableIRQ()
- HAL_NVIC_SystemReset()
- HAL_SYSTICK_IRQHandler()
- HAL_NVIC_GetPendingIRQ() / HAL_NVIC_SetPendingIRQ () / HAL_NVIC_ClearPendingIRQ()
- HAL_NVIC_GetActive(IRQn)
- HAL_SYSTICK_Config()
- HAL_SYSTICK_CLKSourceConfig()
- HAL_SYSTICK_Callback()
- HAL_MPU_Enable()
- HAL_MPU_Disable()
- HAL_MPU_ConfigRegion()
- HAL_MPU_EnableRegion()
- HAL_MPU_DisableRegion()
- HAL_CORTEX_ClearEvent()

3.11.4 PWR

The PWR HAL driver handles power management. The features shared between all STM32 series are listed below:

- PVD configuration, enabling/disabling
  - HAL_PWR_ConfigPVD()
  - HAL_PWR_EnablePVD() / HAL_PWR_DisablePVD()
- Wake-up pin configuration
  - HAL_PWR_EnableWakeUpPin() / HAL_PWR_DisableWakeUpPin()
- Low-power mode entry
  - HAL_PWR_EnterSLEEPMode()
  - HAL_PWR_EnterSTOPMode()
  - HAL_PWR_EnterSTANDBYMode()
- STM32N6 low-power configuration features:
  - HAL_PWR_EnableSleepOnExit() / HAL_PWR_DisableSleepOnExit: (low-power management Sleep-on-entry)
  - HAL_PWR_EnableSEVOnPend() / HAL_PWR_DisableSEVOnPend (signal event on pending interrupt)

3.11.5 EXTI

The EXTI is not considered as a standalone peripheral but rather as a service used by other peripheral, that are handled through EXTI HAL APIs. In addition, each peripheral HAL driver implements the associated EXTI configuration and function as macros in its header file.

The first 16 EXTI lines connected to the GPIOs are managed within the GPIO driver. The GPIO_InitTypeDef structure configures a given I/O as an external interrupt or external event.

The EXTI lines connected internally to the PVD, RTC, USB, and Ethernet, are configured within the HAL drivers of these peripheral through the macros given in the table below.

The EXTI internal connections depend on the targeted STM32 microcontroller (refer to the product datasheet for more details):

Table 13. Description of EXTI configuration macros

Macros	Description
__HAL_PPP_{SUBLOCK}__EXTI_ENABLE_IT()	Enables a given EXTI line interrupt Example: __HAL_PWR_PVD_EXTI_ENABLE_IT()

Macros	Description
__HAL_PPP_{SUBLOCK}__EXTI_DISABLE_IT()	Disables a given EXTI line. Example: __HAL_PWR_PVD_EXTI_DISABLE_IT()
__HAL_ PPP_{SUBLOCK}__EXTI_GET_FLAG()	Gets a given EXTI line interrupt flag pending bit status. Example: __HAL_PWR_PVD_EXTI_GET_FLAG()
__HAL_ PPP_{SUBLOCK}_EXTI_CLEAR_FLAG()	Clears a given EXTI line interrupt flag pending bit. Example; __HAL_PWR_PVD_EXTI_CLEAR_FLAG()
__HAL_PPP_SUBBLOCK_EXTI_ENABLE_EVENT()	Enable a given EXTI line event Example: __HAL_PWR_PVD_EXTI_ENABLE_EVENT()
__HAL_PPP_SUBBLOCK_EXTI_DISABLE_EVENT()	Disable a given EXTI line event Example: __HAL_PWR_PVD_EXTI_DISABLE_EVENT()
__HAL_ PPP_SUBBLOCK_EXTI_ENABLE_RISING_EDGE()	Configure an EXTI interrupt or event on rising edge
__HAL_ PPP_SUBBLOCK_EXTI_ENABLE_FALLING_EDGE()	Enable an EXTI interrupt or event on falling edge
__HAL_ PPP_SUBBLOCK_EXTI_DISABLE_RISING_EDGE()	Disable an EXTI interrupt or event on rising edge
__HAL_ PPP_SUBBLOCK_EXTI_DISABLE_FALLING_EDGE()	Disable an EXTI interrupt or event on falling edge
__HAL_ PPP_SUBBLOCK_EXTI_ENABLE_RISING_FALLING_EDGE()	Enable an EXTI interrupt or event on rising/falling edge
__HAL_ PPP_SUBBLOCK_EXTI_DISABLE_RISING_FALLING_EDGE()	Disable an EXTI interrupt or event on rising/falling edge

If the EXTI interrupt mode is selected, the user application must call HAL_PPP_FUNCTION_IRQHandler() (for example HAL_PWR_PVD_IRQHandler()), from stm32n6xx_it.c file, and implement HAL_PPP_FUNCTIONCallback() callback function (for example HAL_PWR_PVDCallback()).

3.11.6 DMA

DMA transfer modes are divided into two main categories:

- Normal transfers (as for legacy series)
- Linked-list transfers

In addition to linked-list capability, all the advanced DMA features are managed and configured via the extended API as extensions to normal mode. The advanced features are repeated block, trigger, and data handling.

For a given channel, enable and configure the peripheral to be connected to the DMA channel (except for internal SRAM and flash memories for which no initialization is required). Refer to the device reference manuals for details on the connection between peripherals and DMA requests.

The normal transfer mode is initialized via the standard API while the Linked-list mode is configured via the extended API.

Alternatively to the normal programming mode, a DMA channel can be programmed by a list of transfers, also known as linked-list (list of node items), each node being defined by its data structure and specifying a standalone DMA channel. When enabled, the DMA channel fetches the first linked-list node from SRAM (also known as head node). When the execution is complete, it fetches and executes the next linked-list node. This operation is repeated until the end of the linked-list queue. The linked-list can be linear or circular:

- Linear linked-list:
  The last linked-list queue node is not linked to another queue node. The DMA channel fetches and executes the whole DMA linked-list queue from the first node (head node) to the last (tail node). When the last node execution is complete, the DMA channel remains in idle state and another transfer can be launched.
- Circular linked-list:
  The last linked-list node can be linked to any linked-list queue node. The DMA channel fetches and executes the whole DMA linked-list queue from the first node to the last. When the last node is executed, the DMA channel fetches again the first circular node and repeats the same sequence in an infinite loop (circular transfer). An abort operation is required to stop the DMA channel. The circular linked-list mode replaces the legacy circular transfer mode.

To reduce linked-list queue execution time and power consumption, the DMA channel supports dynamic linked-list execution in two types of formats, static and dynamic:

- Static linked-list format: this format refers to the full linked-list node where all DMA channel parameters are fetched and executed independently of the redundancy of information.
- Dynamic linked-list: this format refers to the customized linked-list node where only the necessary DMA channel parameters are fetched and executed (for example, no update is required if data size = 20 on previous node, and data size = 20 on the current node).

For linked-list transfers, the DMA channel can execute the linked-list queue node by node. This feature is named link-step mode. When activated, enabling the DMA channel for the first time fetches the head node from memory and stops. Another DMA channel enable is then required to execute the node. After that, the DMA channel must remain enabled to execute each node until the end of the linked-list queue. When the linked-list queue is circular, the DMA channel must be enabled in an infinite loop to keep the DMA channel running. This feature is useful for debug purposes or for executing queue nodes asynchronously.

Each DMA channel transfer (normal or linked-list) can be configured depending on the DMA channel instance embedded in the devices. The configurations are the following:

- Repeated block configuration:
  This feature is also named two-dimension addressing transfer. If it is supported, the DMA channel can perform repeated transfers of programmed blocks (the size of the block transfer is the legacy data size). In addition to the block repeat count, the DMA channel addresses can jump at burst or block level. The jump length is a programmable parameter defined in the user application:
  - Jump at burst level: the DMA channel keeps an empty area between two consecutive transmitted bursts.
  - Jump at block level; the DMA channel keeps an empty area between two consecutive transmitted blocks.
- Trigger configuration:
  The DMA channel transfers can be triggered on the rising or falling edge of a given signal (hardware triggers). The trigger condition can be applied at single/burst level, block level, repeated block level or node level.
  The DMA channel can report a trigger overrun when more than two trigger signal events are detected before executing the current transfer.
- Data handling:
  The data handling feature is based on a FIFO capability that can be:
  - Padding pattern: the padding pattern (zero padding or sign extension) is selected when the source data width is smaller than the destination data width.
  - Truncation: the source data is truncated when the source data width is bigger than the destination data width.
  - Pack/unpack: a set of data can be packed when the source data width is smaller than the destination data width or unpacked when the source data width is bigger than the destination data width.
  - Exchange: data can be exchanged at byte and half-word level on the destination, and byte level on the source.

Each ongoing DMA channel transfer (normal or linked-list) can be suspended and resumed at runtime by the application. When trying to suspend an ongoing transfer, the DMA channel is not suspended instantly. Instead, the ongoing single/burst transfer completes and then stops.

When the DMA channel is suspended, the current transfer can be resumed instantly.

The DMA channel that supports the FIFO can report in real time the number of beats remaining on the destination FIFO ((output).

Two operating modes are available:

- Polling mode I/O operation
  1. Use HAL_DMA_Start() to start DMA transfer when the source and destination addresses and the length of data to be transferred have been configured.
  2. Use HAL_DMA_PollForTransfer() to poll for the end of the current transfer. In this case, a fixed timeout can be configured depending on the user application.
- Interrupt mode I/O operation
  1. Configure the DMA interrupt priority using HAL_NVIC_SetPriority().
  2. Enable the DMA IRQ handler using HAL_NVIC_EnableIRQ().
  3. Use HAL_DMA_Start_IT() to start DMA transfer when the source and destination addresses and the length of data to be transferred have been configured. In this case, the DMA interrupt is configured.
  4. Use HAL_DMA_IRQHandler() called under DMA_IRQHandler() interrupt subroutine.
  5. When data transfer is complete, HAL_DMA_IRQHandler() function is executed and a user function can be called by customizing XferCpltCallback and XferErrorCallback function pointer (that is a member of DMA handle structure).

Additional functions and macros are available to ensure efficient DMA management:

- Use the HAL_DMA_GetState() function to return the DMA state and HAL_DMA_GetError() in case of error detection.
- Use the HAL_DMA_Abort() function to abort the current transfer.

The most used DMA HAL driver macros are the following:

- __HAL_DMA_ENABLE: enables the specified DMA channel
- __HAL_DMA_DISABLE: disables the specified DMA channel
- __HAL_DMA_GET_FLAG: gets the DMA channel pending flags
- __HAL_DMA_CLEAR_FLAG: clears the DMA channel pending flags
- __HAL_DMA_ENABLE_IT: enables the specified DMA channel interrupts
- __HAL_DMA_DISABLE_IT: disables the specified DMA channel interrupts
- __HAL_DMA_GET_IT_SOURCE: checks whether the specified DMA channel interrupt has been enabled or not

To correctly use the DMA linked-list feature, apply the following call sequences:

- Linear transfer:
  - Linked-list queue building
    - HAL_DMAEx_List_BuildNode()
    - HAL_DMAEx_List_InsertNode_Tail()
    - HAL_DMAEx_List_BuildNode()
    - HAL_DMAEx_List_InsertNode_Tail()
    - HAL_DMAEx_List_ConvertQToDynamic()
  - Linked-list queue execution
    - HAL_DMAEx_List_Init()
    - HAL_DMAEx_List_LinkQ()
    - HAL_DMAEx_List_Start() / HAL_DMAEx_List_Start_IT()
    - HAL_DMAEx_List_UnLinkQ()
    - HAL_DMAEx_List_DeInit()
- Circular transfer:
  - Linked-list queue building
    - HAL_DMAEx_List_BuildNode()
    - HAL_DMAEx_List_InsertNode_Tail()
    - HAL_DMAEx_List_BuildNode()
    - HAL_DMAEx_List_InsertNode_Tail()
    - HAL_DMAEx_List_SetCircularModeConfig()/HAL_DMAEx_List_SetCircularMode()
    - HAL_DMAEx_List_ConvertQToDynamic()
  - Linked-list queue execution
    - HAL_DMAEx_List_Init()
    - HAL_DMAEx_List_LinkQ()
    - HAL_DMAEx_List_Start() / HAL_DMAEx_List_Start_IT()
    - HAL_DMA_Abort() / HAL_DMA_Abort_IT()
    - HAL_DMAEx_List_UnLinkQ()
    - HAL_DMAEx_List_DeInit()

Note: When a peripheral is used in DMA mode, the DMA initialization must be done in the HAL_PPP_MspInit() callback. In addition, the user application must associate the DMA handle to the PPP handle (refer to section "HAL I/O operation functions").

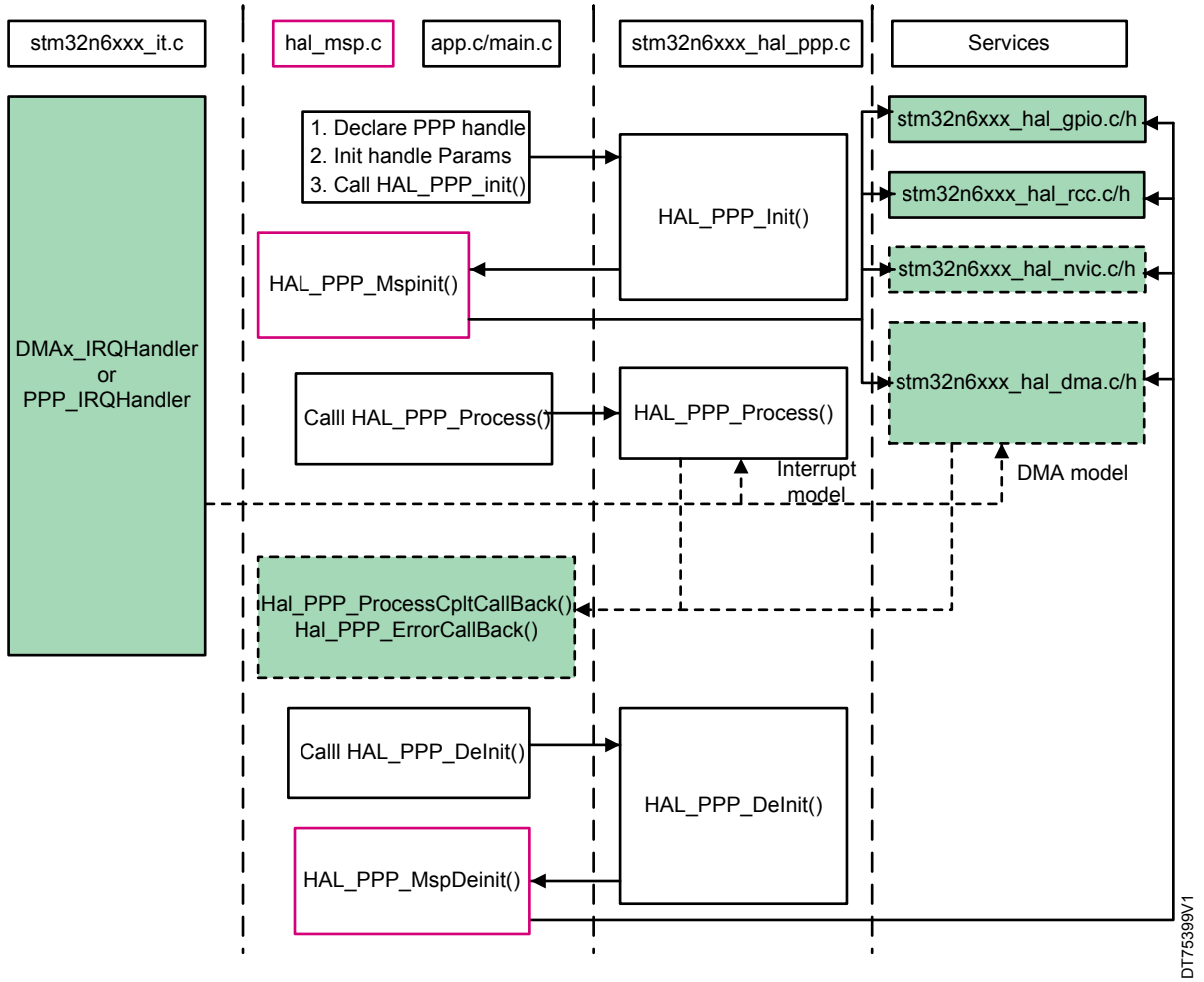
Note: DMA channel callbacks need to be initialized by the user application only in case of memory-to-memory transfer. However, when peripheral-to-memory transfers are used, these callbacks are automatically initialized by calling a process API function that uses the DMA.

3.12 How to use HAL drivers

3.12.1 HAL usage models

The following figure shows the typical use of the HAL driver and the interaction between the application user, the HAL driver and the interrupts.

Figure 6. **HAL driver model**



Note: The functions implemented in the HAL driver are shown in green, the functions called from interrupt handlers in dotted lines, and the msp functions implemented in the user application in red. Nondotted lines represent the interactions between the user application functions.

Basically, the HAL driver APIs are called from user files and optionally from interrupt handlers file when the APIs based on the DMA or the PPP peripheral dedicated interrupts are used.

When DMA or PPP peripheral interrupts are used, the PPP process complete callbacks are called to inform the user about the process completion in real-time event mode (interrupts). Note that the same process completion callbacks are used for DMA in interrupt mode.

3.12.2 HAL initialization

3.12.2.1 HAL global initialization

In addition to the peripheral initialization and deinitialization functions, a set of APIs are provided to initialize the HAL core implemented in file stm32n6xx_hal.c.

- HAL_Init(): this function must be called at application startup to
  - Initialize data/instruction cache and prefetch queue
  - Set SysTick timer to generate an interrupt each 1 ms (based on HSI clock) with the lowest priority
  - Call HAL_MspInit() user callback function to perform system level initializations (clock, GPIOs, DMA, interrupts). HAL_MspInit() is defined as a “weak” empty function in the HAL drivers.

- HAL_DeInit()
  - Resets all peripherals
  - Calls function HAL_MspDeInit() which is a user callback function to do system level deinitializations.
- HAL_GetTick(): this function gets the current SysTick counter value (incremented in SysTick interrupt) used by peripheral drivers to handle timeouts.
- HAL_Delay(): this function implements a delay (expressed in milliseconds) using the SysTick timer.
  Care must be taken when using HAL_Delay() since this function provides an accurate delay (expressed in milliseconds) based on a variable incremented in SysTick ISR. This means that if HAL_Delay() is called from a peripheral ISR, then the SysTick interrupt must have the highest priority (numerically lower) than the peripheral interrupt, otherwise the caller ISR is blocked.

3.12.2.2 *System clock initialization*

The clock configuration is done at the beginning of the user code. However, the user can change the configuration of the clock in his own code.

Below the typical clock configuration sequence to reach the maximum clock frequency of 600 MHz based on the HSI clock.

```
/**
  * @brief  System Clock Configuration
  *         The system Clock is configured as follows :
  *            CPU Clock source               = IC1_CK
  *            System bus Clock source        = IC2_IC6_IC11_CK
  *            CPUCLK (sysa_ck) (Hz)          = 600000000
  *            SYSCLK AXI (sysb_ck) (Hz)      = 400000000
  *            SYSCLK NPU (sysc_ck) (Hz)      = 300000000
  *            SYSCLK AXISRAM3/4/5/6 (sysd_ck) (Hz) = 400000000
  *            HCLKx(Hz)                      = 200000000
  *            PCLKx(Hz)                      = 200000000
  *            AHB Prescaler                  = 2
  *            APB1 Prescaler                 = 1
  *            APB2 Prescaler                 = 1
  *            APB4 Prescaler                 = 1
  *            APB5 Prescaler                 = 1
  *            PLL1 State                     = ON
  *            PLL1 clock source              = HSI
  *            PLL1 M                         = 4
  *            PLL1 N                         = 75
  *            PLL1 P1                        = 1
  *            PLL1 P2                        = 1
  *            PLL1 FRACN                     = 0
  *            PLL2 State                     = BYPASS
  *            PLL2 clock source              = HSI
  *            PLL3 State                     = BYPASS
  *            PLL3 clock source              = HSI
  *            PLL4 State                     = BYPASS
  *            PLL4 clock source              = HSI
  * @retval None
  */


/** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_NONE;
  RCC_OscInitStruct.PLL1.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL1.PLLSource = RCC_PLLSOURCE_HSI;
  RCC_OscInitStruct.PLL1.PLLM = 4;
  RCC_OscInitStruct.PLL1.PLLN = 75;
  RCC_OscInitStruct.PLL1.PLLFractional = 0;
  RCC_OscInitStruct.PLL1.PLLP1 = 1;
  RCC_OscInitStruct.PLL1.PLLP2 = 1;
  RCC_OscInitStruct.PLL2.PLLState = RCC_PLL_NONE;
  RCC_OscInitStruct.PLL3.PLLState = RCC_PLL_NONE;
  RCC_OscInitStruct.PLL4.PLLState = RCC_PLL_NONE;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
```

```
    Error_Handler();
  }

  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_CPUCLK|RCC_CLOCKTYPE_HCLK
                              |RCC_CLOCKTYPE_SYSCLK|RCC_CLOCKTYPE_PCLK1
                              |RCC_CLOCKTYPE_PCLK2|RCC_CLOCKTYPE_PCLK5
                              |RCC_CLOCKTYPE_PCLK4;
  RCC_ClkInitStruct.CPUCLKSource = RCC_CPUCLKSOURCE_IC1;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_IC2_IC6_IC11;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_APB1_DIV1;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_APB2_DIV1;
  RCC_ClkInitStruct.APB4CLKDivider = RCC_APB4_DIV1;
  RCC_ClkInitStruct.APB5CLKDivider = RCC_APB5_DIV1;
  RCC_ClkInitStruct.IC1Selection.ClockSelection = RCC_ICCLKSOURCE_PLL1;
  RCC_ClkInitStruct.IC1Selection.ClockDivider = 2;
  RCC_ClkInitStruct.IC2Selection.ClockSelection = RCC_ICCLKSOURCE_PLL1;
  RCC_ClkInitStruct.IC2Selection.ClockDivider = 3;
  RCC_ClkInitStruct.IC6Selection.ClockSelection = RCC_ICCLKSOURCE_PLL1;
  RCC_ClkInitStruct.IC6Selection.ClockDivider = 4;
  RCC_ClkInitStruct.IC11Selection.ClockSelection = RCC_ICCLKSOURCE_PLL1;
  RCC_ClkInitStruct.IC11Selection.ClockDivider = 3;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct) != HAL_OK)
  {
    Error_Handler();
  }
```

3.12.2.3 *HAL MSP initialization process*

The peripheral initialization is done through *HAL_PPP_Init()* while the hardware resources initialization used by a peripheral (PPP) is performed during this initialization by calling the MSP callback function *HAL_PPP_MspInit()*.

The MspInit callback performs the low-level initialization related to the different additional hardware resources: RCC, GPIO, NVIC, and DMA.

All the HAL drivers with handles include two MSP callbacks for initialization and deinitialization:

```
/**
* @brief Initializes the PPP MSP.
* @param hppp: PPP handle
* @retval None */
void __weak HAL_PPP_MspInit(PPP_HandleTypeDef *hppp) {
/* NOTE : This function Should not be modified, when the callback is needed,
the HAL_PPP_MspInit could be implemented in the user file */
}
/**
* @brief DeInitializes PPP MSP.
* @param hppp: PPP handle
* @retval None */
void __weak HAL_PPP_MspDeInit(PPP_HandleTypeDef *hppp) {
/* NOTE : This function Should not be modified, when the callback is needed,
the HAL_PPP_MspDeInit could be implemented in the user file */
}
```

The MSP callbacks are declared empty as weak functions in each peripheral driver. The user can use them to set the low-level initialization code or omit them and use their own initialization routine.

The HAL MSP callback is implemented inside the *stm32n6xx_hal_msp.c* file in the user folders. An *stm32n6xx_hal_msp.c* file template is located in the HAL folder and should be copied to the user folder. It can be generated automatically by the STM32CubeMX tool and further modified. Note that all the routines are declared as weak functions and could be overwritten or removed to use user low-level initialization code.

stm32n6xx_hal_msp.c file contains the following functions:

Table 14. **MSP functions**

Routine	Description
void HAL_MspInit()	Global MSP initialization routine
void HAL_MspDeInit()	Global MSP deinitialization routine
void HAL_PPP_MspInit()	PPP MSP initialization routine
void HAL_PPP_MspDeInit()	PPP MSP deinitialization routine

By default, if no peripheral needs to be deinitialized during the program execution, the whole MSP initialization is done in *Hal_MspInit()* and MSP deinitialization in the *Hal_MspDeInit()*. In this case, the *HAL_PPP_MspInit()* and *HAL_PPP_MspDeInit()* are not implemented.

When one or more peripherals needs to be deinitialized in runtime and the low-level resources of a given peripheral need to be released and used by another peripheral, HAL_PPP_MspDeInit() and *HAL_PPP_MspInit()* are implemented for the concerned peripheral and other peripherals initialization and de-Initialization are kept in the global *HAL_MspInit()* and the *HAL_MspDeInit()*.

If there is nothing to be initialized by the global *HAL_MspInit()* and *HAL_MspDeInit()*, the two routines can simply be omitted.

3.12.3 HAL I/O operation process

The HAL functions with internal data processing like transmit, receive, write, and read are generally provided with three data processing modes as follows:

- Polling mode
- Interrupt mode
- DMA mode

3.12.3.1 Polling mode

In polling mode, the HAL functions return the process status when the data processing in blocking mode is complete. The operation is considered complete when the function returns the HAL_OK status, otherwise an error status is returned. The user can get more information through the *HAL_PPP_GetState()* function. The data processing is handled internally in a loop. A timeout (expressed in ms) is used to prevent process hanging.

The example below shows the typical polling mode processing sequence:

```
HAL_StatusTypeDef HAL_PPP_Transmit ( PPP_HandleTypeDef * phandle, uint8_t pData,
int16_tSize,uint32_tTimeout)
{
if((pData == NULL ) || (Size == 0))
{
return HAL_ERROR;
}
(…) while (data processing is running)
{
if( timeout reached )
{
return HAL_TIMEOUT;
}
}
(…)
return HAL_OK; }
```

3.12.3.2 Interrupt mode

In interrupt mode, the HAL function returns the process status after starting the data processing and enabling the appropriate interruption. The end of the operation is indicated by a callback declared as a weak function. It can be customized by the user to be informed in real time about the process completion. The user can also get the process status through the *HAL_PPP_GetState()* function.

In interrupt mode, four functions are declared in the driver:

- *HAL_PPP_Process_IT()*: launches the process
- *HAL_PPP_IRQHandler()*: global PPP peripheral interruption

- *__weak HAL_PPP_ProcessCpltCallback ()*: callback relative to the process completion.
- *__weak HAL_PPP_ProcessErrorCallback()*: callback relative to the process error.

To use a process in interrupt mode, *HAL_PPP_Process_IT()* is called in the user file and HAL_PPP_IRQHandler in *stm32n6xx_it.c.*

The *HAL_PPP_ProcessCpltCallback()* function is declared as a weak function in the driver. This means that the user can declare it again in the application. The function in the driver is not modified.

An example of use is illustrated below:

main.c file:

```
UART_HandleTypeDef UartHandle;
int main(void)
{
/* Set User Parameters */
UartHandle.Init.BaudRate = 9600;
UartHandle.Init.WordLength = UART_DATABITS_8;
UartHandle.Init.StopBits = UART_STOPBITS_1;
UartHandle.Init.Parity = UART_PARITY_NONE;
UartHandle.Init.HwFlowCtl = UART_HWCONTROL_NONE;
UartHandle.Init.Mode = UART_MODE_TX_RX;
UartHandle.Init.Instance = USART1;
HAL_UART_Init(&UartHandle);
HAL_UART_SendIT(&UartHandle, TxBuffer, sizeof(TxBuffer));
while (1);
}
void HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart)
{
}
void HAL_UART_ErrorCallback(UART_HandleTypeDef *huart)
{
}
```

stm32n6xx_it.c file:

3.12.3.3 DMA mode

In DMA mode, the HAL function returns the process status after starting the data processing through the DMA and after enabling the appropriate DMA interruption. The end of the operation is indicated by a callback declared as a weak function and can be customized by the user to be informed in real time about the process completion. The user can also get the process status through the *HAL_PPP_GetState()* function. For the DMA mode, three functions are declared in the driver:

- *HAL_PPP_Process_DMA()*: launch the process
- *HAL_PPP_DMA_IRQHandler()*: the DMA interruption used by the PPP peripheral
- *__weak HAL_PPP_ProcessCpltCallback()*: the callback relative to the process completion.
- *__weak HAL_PPP_ErrorCpltCallback()*: the callback relative to the process error.

To use a process in DMA mode, *HAL_PPP_Process_DMA()* is called in the user file and the *HAL_PPP_DMA_IRQHandler()* is placed in the *stm32n6xx_it.c*. When DMA mode is used, the DMA initialization is done in the *HAL_PPP_MspInit()* callback. The user should also associate the DMA handle with the PPP handle. For this purpose, the handles of all the peripheral drivers that use the DMA must be declared as follows:

```
typedef struct
{
PPP_TypeDef *Instance; /* Register base address */
PPP_InitTypeDef Init; /* PPP communication parameters */
HAL_StateTypeDef State; /* PPP communication state */
(…)
DMA_HandleTypeDef *hdma; /* associated DMA handle */
} PPP_HandleTypeDef;
```

The initialization is done as follows (UART example):

```
int main(void)
{
/* Set User Parameters */
UartHandle.Init.BaudRate = 9600;
UartHandle.Init.WordLength = UART_DATABITS_8;
UartHandle.Init.StopBits = UART_STOPBITS_1;
UartHandle.Init.Parity = UART_PARITY_NONE;
UartHandle.Init.HwFlowCtl = UART_HWCONTROL_NONE;
UartHandle.Init.Mode = UART_MODE_TX_RX;
UartHandle.Init.Instance = UART1;
HAL_UART_Init(&UartHandle);
(..)
}
void HAL_USART_MspInit (UART_HandleTypeDef * huart)
{
static DMA_HandleTypeDef hdma_tx;
static DMA_HandleTypeDef hdma_rx;
(…)
__HAL_LINKDMA(UartHandle, DMA_Handle_tx, hdma_tx);
__HAL_LINKDMA(UartHandle, DMA_Handle_rx, hdma_rx);
(…)
}
```

The *HAL_PPP_ProcessCpltCallback()* function is declared as a weak function in the driver that means. The user can declare it again in the application code. The function in the driver must not be modified.

An example of use is illustrated below:

main.c file:

```
UART_HandleTypeDef UartHandle;
int main(void)
{
/* Set User Paramaters */
UartHandle.Init.BaudRate = 9600;
UartHandle.Init.WordLength = UART_DATABITS_8;
UartHandle.Init.StopBits = UART_STOPBITS_1;
UartHandle.Init.Parity = UART_PARITY_NONE;
UartHandle.Init.HwFlowCtl = UART_HWCONTROL_NONE;
UartHandle.Init.Mode = UART_MODE_TX_RX; UartHandle.Init.Instance = USART1;
HAL_UART_Init(&UartHandle);
HAL_UART_Send_DMA(&UartHandle, TxBuffer, sizeof(TxBuffer));
while (1);
}
void HAL_UART_TxCpltCallback(UART_HandleTypeDef *phuart)
{
}
void HAL_UART_TxErrorCallback(UART_HandleTypeDef *phuart)
{
}
```

stm32n6xx_it.c file:

```
extern UART_HandleTypeDef UartHandle;
void DMAx_IRQHandler(void)
{
HAL_DMA_IRQHandler(&UartHandle.DMA_Handle_tx);
}
```

HAL_USART_TxCpltCallback() and *HAL_USART_ErrorCallback()* should be linked in the *HAL_PPP_Process_DMA()* function to the DMA transfer complete callback and the DMA transfer error callback by using the following statement:

```
HAL_PPP_Process_DMA (PPP_HandleTypeDef *hppp, Params….)
{
(…)
hppp->DMA_Handle->XferCpltCallback = HAL_UART_TxCpltCallback ;
hppp->DMA_Handle->XferErrorCallback = HAL_UART_ErrorCallback ;
(…)
}
```

3.12.4 Timeout and error management

3.12.4.1 Timeout management

The timeout is often used for the APIs that operate in polling mode. It defines the delay during which a blocking process should wait until an error is returned. An example is provided below:

```
HAL_StatusTypeDef HAL_DMA_PollForTransfer(DMA_HandleTypeDef *hdma, uint32_t CompleteLevel, ui
nt32_t Timeout)
```

The timeout possible values are the following:

Table 15. Timeout values

Timeout value	Description
0	No poll: Immediate process check and exit
1 ... (HAL_MAX_DELAY -1) (1)	Timeout in ms
HAL_MAX_DELAY	Infinite poll until the process is successful

1. *HAL_MAX_DELAY is defined in the stm32n6xx_hal_def.h as 0xFFFFFFFF*

However, in some cases, a fixed timeout is used for system peripherals or internal HAL driver processes. In these cases, the timeout has the same meaning and is used in the same way, except when it is defined locally in the drivers and cannot be modified or introduced as an argument in the user application.

Example of fixed timeout:

```
#define LOCAL_PROCESS_TIMEOUT 100
HAL_StatusTypeDef HAL_PPP_Process(PPP_HandleTypeDef *hppp)
{
  (…)
  tickstart = HAL_GetTick();
  (…)
 while (ProcessOngoing)
  {
    (…)
   if ((HAL_GetTick() - tickstart) > LOCAL_PROCESS_TIMEOUT)
    {
      hppp->ErrorCode |= HAL_PPP_ERROR_TIMEOUT;
      hppp->State= HAL_PPP_STATE_READY;
      return HAL_ERROR;
    }
  }
  (…)
}
```

The following example shows how to use the timeout inside the polling functions:

```
HAL_StatusTypeDef HAL_PPP_Poll(PPP_HandleTypeDef *hppp, uint32_t Timeout)
{
  (…)
  tickstart = HAL_GetTick();
  (…)
 while (ProcessOngoing)
  {
    (…)
   if (((HAL_GetTick() - tickstart) > Timeout) || (Timeout == 0))
    {
      hppp->ErrorCode |= HAL_PPP_ERROR_TIMEOUT;
      hppp->State= HAL_PPP_STATE_READY;
      return HAL_ERROR;
    }
  }
  (…)
}
```

3.12.4.2 *Error management*

The HAL drivers implement a check on the following items:

- Valid parameters: for some process the used parameters should be valid and already defined, otherwise the system may crash or go into an undefined state. These critical parameters are checked before being used (see example below).

```
HAL_StatusTypeDef HAL_PPP_Process(PPP_HandleTypeDef* hppp, uint32_t *pdata, uint32 Size
)
{
if ((pData == NULL ) || (Size == 0))
{
return HAL_ERROR;
}
}
```

- Valid handle: the PPP peripheral handle is the most important argument since it keeps the PPP driver vital parameters. It is always checked in the beginning of the *HAL_PPP_Init()* function.

```
HAL_StatusTypeDef HAL_PPP_Init(PPP_HandleTypeDef* hppp)
{
if (hppp == NULL) //the handle should be already allocated
{
return HAL_ERROR;
}
}
```

- Timeout error: the following statement is used when a timeout error occurs:

```
while (Process ongoing)
{
timeout = HAL_GetTick() + Timeout; while (data processing is running)
{
if(timeout) { return HAL_TIMEOUT;
}
}
```

When an error occurs during a peripheral process, *HAL_PPP_Process ()* returns with a *HAL_ERROR* status. The HAL PPP driver implements the *HAL_PPP_GetError ()* to retrieve the origin of the error.

```
HAL_PPP_ErrorTypeDef HAL_PPP_GetError (PPP_HandleTypeDef *hppp);
```

In all peripheral handles, a *HAL_PPP_ErrorTypeDef* is defined and used to store the last error code.

```
typedef struct
{
PPP_TypeDef * Instance; /* PPP registers base address */
PPP_InitTypeDef Init; /* PPP initialization parameters */
HAL_LockTypeDef Lock; /* PPP locking object */
__IO HAL_PPP_StateTypeDef State; /* PPP state */
__IO HAL_PPP_ErrorTypeDef ErrorCode; /* PPP Error code */
(…)
/* PPP specific parameters */
}
PPP_HandleTypeDef;
```

The error state and the peripheral global state are always updated before returning an error:

```
PPP->State = HAL_PPP_READY;  /* Set the peripheral ready */
PP->ErrorCode = HAL_ERRORCODE ; /* Set the error code */
_HAL_UNLOCK(PPP) ; /* Unlock the PPP resources */
return HAL_ERROR;  /*return with HAL error */
```

HAL_PPP_GetError () must be used in interrupt mode in the error callback:

```
void HAL_PPP_ProcessCpltCallback(PPP_HandleTypeDef *hspi)
{
ErrorCode = HAL_PPP_GetError (hppp); /* retreive error code */
}
```

3.12.4.3 *Runtime checking*

The HAL implements runtime failure detection by checking the input values of all HAL driver functions. The runtime checking is achieved by using an *assert_param* macro. This macro is used in all the HAL driver functions which have an input parameter. It verifies that the input value lies within the parameter allowed values.

To enable the runtime checking, use the *assert_param* macro, and leave the define ***USE_FULL_ASSERT*** uncommented in the *stm32n6xx_hal_conf.h* file.

```
void HAL_UART_Init(UART_HandleTypeDef *huart)
{
(..) /* Check the parameters */
assert_param(IS_UART_INSTANCE(huart->Instance));
assert_param(IS_UART_BAUDRATE(huart->Init.BaudRate));
assert_param(IS_UART_WORD_LENGTH(huart->Init.WordLength));
assert_param(IS_UART_STOPBITS(huart->Init.StopBits));
assert_param(IS_UART_PARITY(huart->Init.Parity));
assert_param(IS_UART_MODE(huart->Init.Mode));
assert_param(IS_UART_HARDWARE_FLOW_CONTROL(huart->Init.HwFlowCtl));
(..)
```

```
/** @defgroup UART_Word_Length *
@{
*/
#define UART_WORDLENGTH_8B ((uint32_t)0x00000000)
#define UART_WORDLENGTH_9B ((uint32_t)USART_CR1_M)
#define IS_UART_WORD_LENGTH(LENGTH) (((LENGTH) == UART_WORDLENGTH_8B) ||
\ ((LENGTH) == UART_WORDLENGTH_9B))
```

If the expression passed to the assert_param macro is false, the*assert_failed* function is called and returns the name of the source file and the source line number of the call that failed. If the expression is true, no value is returned.

The *assert_param* macro is implemented in stm32n6xx_hal_conf.h:

```
/* Exported macro ------------------------------------------------------------*/
#ifdef USE_FULL_ASSERT
/**
* @brief The assert_param macro is used for function's parameters check.
* @param expr: If expr is false, it calls assert_failed function
* which reports the name of the source file and the source
* line number of the call that failed.
* If expr is true, it returns no value.
* @retval None */
#define assert_param(expr) ((expr)?(void)0:assert_failed((uint8_t *)__FILE__, __LINE__))
/* Exported functions --------------------------------------*/
void assert_failed(uint8_t* file, uint32_t line);
#else
#define assert_param(expr)((void)0)
#endif /* USE_FULL_ASSERT */
```

The *assert_failed* function is implemented in the main.c file or in any other user C file:

```
#ifdef USE_FULL_ASSERT /**
* @brief Reports the name of the source file and the source line number
* where the assert_param error has occurred.
* @param file: pointer to the source file name
* @param line: assert_param error line source number
* @retval None */
void assert_failed(uint8_t* file, uint32_t line)
{
/* User can add his own implementation to report the file name and line number,
ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
/* Infinite loop */
while (1)
{
}
}
```

Attention: ***Because of the overhead runtime checking introduces, it is recommended to use it during application code development and debugging, and to remove it from the final application to improve code size and speed.***

4 Overview of low-layer drivers

The low-layer (LL) drivers are designed to offer a fast light-weight expert-oriented layer which is closer to the hardware than the HAL. Contrary to the HAL, LL APIs are not provided for peripherals where optimized access is not a key feature, or those requiring heavy software configuration and/or complex upper-level stack (such as USB).

The LL drivers feature:

- A set of functions to initialize peripheral main features according to the parameters specified in data structures
- A set of functions used to fill initialization data structures with the reset values of each field
- Functions to perform peripheral deinitialization (peripheral registers restored to their default values)
- A set of inline functions for direct and atomic register access
- Full independence from HAL since LL drivers can be used either in standalone mode (without HAL drivers) or in mixed mode (with HAL drivers)
- Full coverage of the supported peripheral features

The low-layer drivers provide hardware services based on the available features of the STM32 peripherals. These services reflect exactly the hardware capabilities and provide one-shot operations that must be called following the programming model described in the microcontroller line reference manual. As a result, the LL services do not implement any processing and do not require any additional memory resources to save their states, counter, or data pointers: all the operations are performed by changing the associated peripheral registers content.

4.1 Low-layer files

The low-layer drivers are built around header/C files (one per each supported peripheral) plus five header files for some system and Cortex related features.

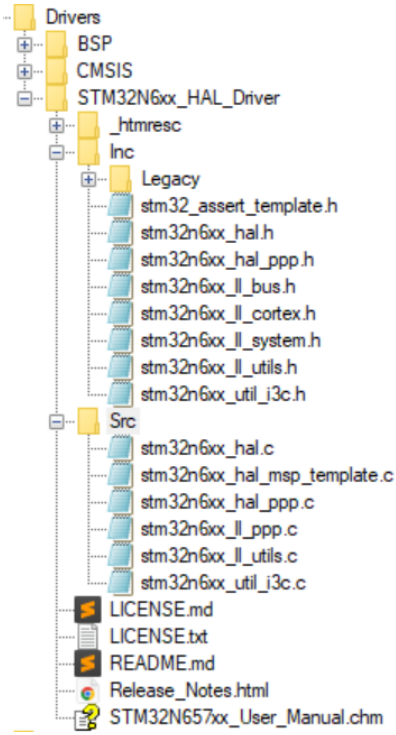
Table 16. LL driver files

File	Description
stm32n6xx_ll_bus.h	This is the h-source file for core bus control and peripheral clock activation and deactivation *Example: LL_AHB2_GRP1_EnableClock*
stm32n6xx_ll_ppp.h/.c	stm32n6xx_ll_ppp.c provides peripheral initialization functions such as LL_PPP_Init(), LL_PPP_StructInit(), LL_PPP_DeInit(). All the other APIs are defined within the stm32n6xx_ll_ppp.h file. The low-layer PPP driver is a standalone module. To use it, the application must include it in the stm32n6xx_ll_ppp.h file.
stm32n6xx_ll_cortex.h	Cortex-M related register operation APIs including the SysTick, low power (such as LL_SYSTICK_xxxxx and LL_LPM_xxxxx "Low Power Mode")
stm32n6xx_ll_utils.h/.c	This file covers the generic APIs: • Read of device unique ID and electronic signature • Timebase and delay management • System clock configuration.
stm32n6xx_ll_system.h	System-related operations. Example: LL_SBS_xxx, LL_DBGMCU_xxx and LL_FLASH_xxx
stm32_assert_template.h	Template file defines the assert_param macro that is used when runtime checking is enabled. This file is required only when the LL drivers are used in standalone mode (without calling the HAL APIs). It should be copied to the application folder and renamed to stm32_assert.h.

Note: There is no configuration file for the LL drivers.

The low-layer files are located in the same HAL driver folder.

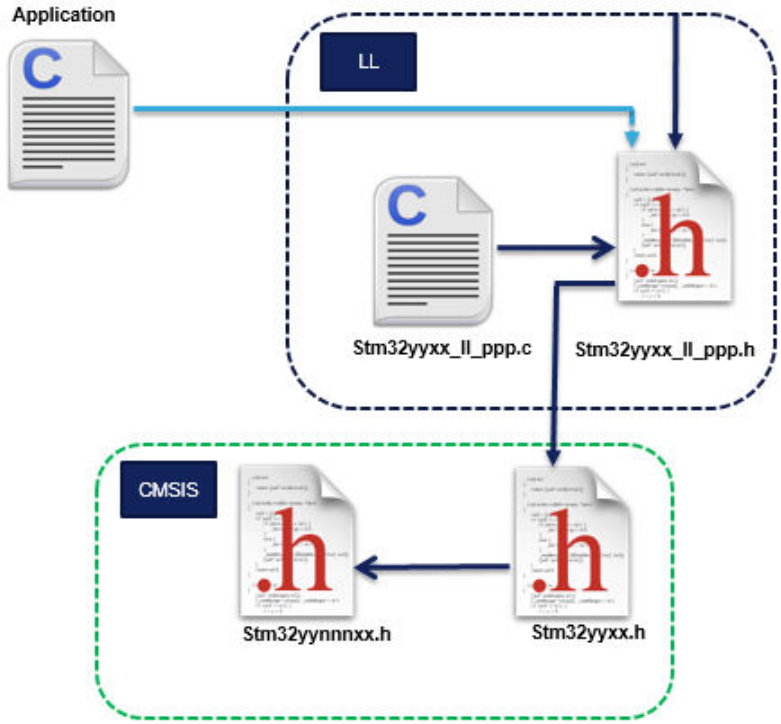
Figure 7. **Low-layer driver folders**



In general, low-layer drivers include only the STM32 CMSIS device file.

```
#include "stm32yyxx.h"
```

Figure 8. **Low-layer driver CMSIS files**



Application files have to include only the used low-layer driver header files.

4.2 Overview of low-layer APIs and naming rules

4.2.1 Peripheral initialization functions

The LL drivers offer three sets of initialization functions. They are defined in the stm32n6xx_ll_ppp.c file:

- Functions to initialize peripheral main features according to the parameters specified in data structures
- A set of functions used to fill initialization data structures with the reset values of each field
- Function for peripheral deinitialization (peripheral registers restored to their default values)

The definition of these LL initialization functions and associated resources (structure, literals, and prototypes) is conditioned by a compilation switch: *USE_FULL_LL_DRIVER*. To use these functions, this switch must be added in the toolchain compiler preprocessor or to any generic header file which is processed before the LL drivers.

The below table shows the list of the common functions provided for all the supported peripherals:

Table 17. Common peripheral initialization functions

Functions	Return type	Parameters	Description
LL_PPP_Init	*ErrorStatus*	• *PPP_TypeDef* PPPx* • *LL_PPP_InitTypeDef* PPP_InitStruct*	Initializes the peripheral main features according to the parameters specified in PPP_InitStruct. Example: LL_USART_Init(USART_TypeDef *USARTx, LL_USART_InitTypeDef *USART_InitStruct)
LL_PPP_StructInit	*void*	• *LL_PPP_InitTypeDef* PPP_InitStruct*	Fills each PPP_InitStruct member with its default value. Example. LL_USART_StructInit(LL_USART_InitTypeDef *USART_InitStruct)
LL_PPP_DeInit	*ErrorStatus*	• *PPP_TypeDef* PPPx*	Deinitializes the peripheral registers that is restore them to their default reset values. Example. LL_USART_DeInit(USART_TypeDef *USARTx)

Additional functions are available for some peripherals (refer to Table 18. Optional peripheral initialization functions).

Table 18. Optional peripheral initialization functions

Functions	Return type	Parameters	Examples
LL_PPP{_CATEGORY}_Init	*ErrorStatus*	• *PPP_TypeDef* PPPx* • *LL_PPP{_CATEGORY}_InitTypeDef* PPP{_CATEGORY}_InitStruct*	Initializes peripheral features according to the parameters specified in PPP_InitStruct. Example: LL_ADC_INJ_Init(ADC_TypeDef *ADCx, LL_ADC_INJ_InitTypeDef *ADC_INJ_InitStruct) LL_RTC_TIME_Init(RTC_TypeDef *RTCx, uint32_t RTC_Format, LL_RTC_TimeTypeDef *RTC_TimeStruct) LL_RTC_DATE_Init(RTC_TypeDef *RTCx, uint32_t RTC_Format, LL_RTC_DateTypeDef *RTC_DateStruct) LL_TIM_IC_Init(TIM_TypeDef* TIMx, uint32_t Channel, LL_TIM_IC_InitTypeDef* TIM_IC_InitStruct) LL_TIM_ENCODER_Init(TIM_TypeDef* TIMx, LL_TIM_ENCODER_InitTypeDef* TIM_EncoderInitStruct)
LL_PPP{_CATEGORY}_StructInit	*void*	*LL_PPP{_CATEGORY}_InitTypeDef* PPP{_CATEGORY}_InitStruct*	Fills each *PPP{_CATEGORY}_InitStruct* member with its default value. Example: LL_ADC_INJ_StructInit(LL_ADC_INJ_InitTypeDef *ADC_INJ_InitStruct)

Functions	Return type	Parameters	Examples
LL_PPP_CommonInit	*ErrorStatus*	• *PPP_TypeDef* PPPx* • *LL_PPP_CommonInitTypeDef* PPP_CommonInitStruct*	Initializes the common features shared between different instances of the same peripheral. Example: LL_ADC_CommonInit(ADC_Common_TypeDef *ADCxy_COMMON, LL_ADC_CommonInitTypeDef *ADC_CommonInitStruct)
LL_PPP_CommonStructInit	*void*	*LL_PPP_CommonInitTypeDef* PPP_CommonInitStruct*	Fills each *PPP_CommonInitStruct* member with its default value Example: LL_ADC_CommonStructInit(LL_ADC_CommonInitTypeDef *ADC_CommonInitStruct)
LL_PPP_ClockInit	*ErrorStatus*	• *PPP_TypeDef* PPPx* • *LL_PPP_ClockInitTypeDef* PPP_ClockInitStruct*	Initializes the peripheral clock configuration in synchronous mode. Example: LL_USART_ClockInit(USART_TypeDef *USARTx, LL_USART_ClockInitTypeDef *USART_ClockInitStruct)
LL_PPP_ClockStructInit	*void*	*LL_PPP_ClockInitTypeDef* PPP_ClockInitStruct*	Fills each *PPP_ClockInitStruct* member with its default value Example: LL_USART_ClockStructInit(LL_USART_ClockInitTypeDef *USART_ClockInitStruct)

4.2.1.1 *Runtime checking*

Like HAL drivers, LL initialization functions implement runtime failure detection by checking the input values of all LL driver functions. For more details, refer to Section 3.12.4.3: Runtime checking.

When using the LL drivers in standalone mode (without calling HAL functions), the following actions are required to use runtime checking:

1. Copy stm32_assert_template.h to the application folder and rename it to stm32_assert.h. This file defines the assert_param macro which is used when runtime checking is enabled.
2. Include the stm32_assert.h file within the application main header file.
3. Add the USE_FULL_ASSERT compilation switch in the toolchain compiler preprocessor or in any generic header file which is processed before the stm32_assert.h driver.

Note: Runtime checking is not available for LL inline functions.

4.2.2 Peripheral register-level configuration functions

On top of the peripheral initialization functions, the LL drivers offer a set of inline functions for direct atomic register access. Their format is as follows:

```
__STATIC_INLINE return_type LL_PPP_Function (PPPx_TypeDef *PPPx, args)
```

The “Function” naming is defined depending on the action category:

- **Specific interrupt, DMA request and status flags management**: Set/Get/Clear/Enable/Disable flags on interrupt and status registers

Table 19. Specific interrupt, DMA request and status flags management

Name	Examples
LL_PPP_{_CATEGORY}_ActionItem_BITNAME *LL_PPP{_CATEGORY}_IsItem_BITNAME_Action*	• LL_RCC_IsActiveFlag_LSIRDY • LL_RCC_IsActiveFlag_FWRST() • LL_ADC_ClearFlag_EOC(ADC1) • LL_DMA_ClearFlag_TCx(DMA_TypeDef* DMAx)

Table 20. Available function formats

Item	Action	Format
Flag	Get	*LL_PPP_IsActiveFlag_BITNAME*
	Clear	*LL_PPP_ClearFlag_BITNAME*

Item	Action	Format
Interrupts	Enable	*LL_PPP_EnableIT_BITNAME*
	Disable	*LL_PPP_DisableIT_BITNAME*
	Get	*LL_PPP_IsEnabledIT_BITNAME*
DMA	Enable	*LL_PPP_EnableDMAReq_BITNAME*
	Disable	*LL_PPP_DisableDMAReq_BITNAME*
	Get	*LL_PPP_IsEnabledDMAReq_BITNAME*

Note: BITNAME refers to the peripheral register bit name as described in the product line reference manual.

- **Peripheral clock activation/deactivation management**: Enable/Disable/Reset a peripheral clock

Table 21. Peripheral clock activation/deactivation management

Name	Examples
LL_BUS_GRPx_ActionClock{Mode}	• *LL_AHB2_GRP1_EnableClock (LL_AHB2_GRP1_PERIPH_GPIOA| LL_AHB2_GRP1_PERIPH_GPIOB)* • *LL_APB1_GRP1_EnableClockSleep (LL_APB1_GRP1_PERIPH_DAC1)*

Note: 'x' corresponds to the group index and refers to the index of the modified register on a given bus. 'bus' corresponds to the bus name.

- ***Peripheral activation/deactivation management:*** Enable/disable a peripheral or activate/deactivate specific peripheral features

Table 22. Peripheral activation/deactivation management

Name	Examples
LL_PPP{_CATEGORY}_Action{Item} *LL_PPP{_CATEGORY}_IsItemAction*	• *LL_ADC_Enable ()* • *LL_ADC_StartCalibration();* • *LL_ADC_IsCalibrationOnGoing;* • *LL_RCC_HSI_Enable ()* • *LL_RCC_HSI_IsReady()*

- ***Peripheral configuration management:*** Set/get a peripheral configuration setting

Table 23. Peripheral configuration management

Name	Examples
LL_PPP{_CATEGORY}_Set{ or Get}ConfigItem	*LL_USART_SetBaudRate (USART2, Clock, LL_USART_BAUDRATE_9600)*

- ***Peripheral register management:*** Write/read the content of a register/retrun DMA relative register address

Table 24. Peripheral register management

Name
LL_PPP_WriteReg(__INSTANCE__, __REG__, __VALUE__)
LL_PPP_ReadReg(__INSTANCE__, __REG__)
*LL_PPP_DMA_GetRegAddr (PPP_TypeDef *PPPx,{Sub Instance if any ex: Channel} , {uint32_t Propriety})*

Note: The Propriety is a variable used to identify the DMA transfer direction or the data register type.

5 Cohabiting of HAL and LL

The low-ayer APIs are designed to be used in standalone mode or combined with the HAL. They cannot be automatically used with the HAL for the same peripheral instance. If you use the LL APIs for a specific instance, you can still use the HAL APIs for other instances. Be careful that the low-layer APIs might overwrite some registers which content is mirrored in the HAL handles.

5.1 Low-layer driver used in standalone mode

The low-layer APIs can be used without calling the HAL driver services. This is done by simply including stm32n6xx_ll_ppp.h in the application files. The LL APIs for a given peripheral are called by executing the same sequence as the one recommended by the programming model in the corresponding product line reference manual. In this case, the HAL drivers associated to the used peripheral can be removed from the workspace. However, the STM32CubeN6 framework should be used in the same way as in the HAL drivers case which means that system file, startup file and CMSIS should always be used.

Note: When the BSP drivers are included, the used HAL drivers associated with the BSP functions drivers should be included in the workspace, even if they are not used by the application layer.

5.2 Mixed use of low-layer APIs and HAL drivers

In this case, the low-layer APIs are used in conjunction with the HAL drivers to achieve direct and register level-based operations.

Mixed use is allowed, however some consideration should be taken into account:

- It is recommended to avoid using simultaneously the HAL APIs and the combination of low-layer APIs for a given peripheral instance. If this is the case, one or more private fields in the HAL PPP handle structure should be updated accordingly.
- For operations and processes that do not alter the handle fields including the initialization structure, the HAL driver APIs and the low-layer services can be used together for the same peripheral instance.
- The low-layer drivers can be used without any restriction with all the HAL drivers that are not based on handle objects (RCC, common HAL, flash, and GPIO).

Several examples showing how to use HAL and LL in the same application are provided within the stm32n6xx firmware package (refer to Examples_MIX projects).

Note:

1. *When the HAL Init/DeInit APIs are not used and are replaced by the low-layer macros, the InitMsp() functions are not called and the MSP initialization should be done in the user application.*
2. *When process APIs are not used and the corresponding function is performed through the low-layer APIs, the callbacks are not called and post processing or error management should be done by the user application.*
3. *When the LL APIs is used for process operations, the IRQ handler HAL APIs cannot be called and the IRQ should be implemented by the user application. Each LL driver implements the macros needed to read and clear the associated interrupt flags.*

6 HAL System Driver

6.1 HAL Firmware driver API description

The following section lists the various functions of the HAL library.

6.1.1 How to use this driver

The common HAL driver contains a set of generic and common APIs that can be used by the PPP peripheral drivers and the user to start using the HAL.

The HAL contains two APIs' categories:

- Common HAL APIs
- Services HAL APIs

All the SYSCFG functions have the same privilege and security attributes.

- Functions can operate in non-privileged code should privileged and unprivileged data access are granted to the peripheral (see HAL_RIF_RISC_SetSlaveSecureAttributes)
- Functions can operate in non-secure code should secure and non-secure data access are granted to the peripheral (see HAL_RIF_RISC_SetSlaveSecureAttributes).

6.1.2 Initialization and de-initialization functions

This section provides functions allowing to:

- Initializes the NVIC allocation and initial clock configuration. It initializes the systick also when timeout is needed and the backup domain when enabled.
- De-Initializes common part of the HAL.
- Configure The time base source to have 1ms time base with a dedicated Tick interrupt priority.
  - SysTick timer is used by default as source of time base, but user can eventually implement his proper time base source (a general purpose timer for example or other time source), keeping in mind that Time base duration should be kept 1ms since PPP_TIMEOUT_VALUEs are defined and handled in milliseconds basis.
  - Time base configuration function (HAL_InitTick ()) is called automatically at the beginning of the program after reset by HAL_Init() or at any time when clock is configured, by HAL_RCC_ClockConfig().
  - Source of time base is configured to generate interrupts at regular time intervals. Care must be taken if HAL_Delay() is called from a peripheral ISR process, the Tick interrupt line must have higher priority (numerically lower) than the peripheral interrupt. Otherwise the caller ISR process will be blocked.
  - functions affecting time base configurations are declared as __weak to make override possible in case of other implementations in user file.

This section contains the following APIs:

- ***HAL_Init()***
- ***HAL_DeInit()***
- ***HAL_MspInit()***
- ***HAL_MspDeInit()***
- ***HAL_InitTick()***

6.1.3 HAL Control functions

This section provides functions allowing to:

- Provide a tick value in millisecond
- Provide a blocking delay in millisecond
- Suspend the time base source interrupt
- Resume the time base source interrupt
- Get the HAL API driver version

- Get the device identifier
- Get the device revision identifier
- Enable/Disable Debug module during SLEEP mode
- Enable/Disable Debug module during STOP mode
- Enable/Disable Debug module during STANDBY mode

This section contains the following APIs:

- ***HAL_IncTick()***
- ***HAL_GetTick()***
- ***HAL_GetTickPrio()***
- ***HAL_SetTickFreq()***
- ***HAL_GetTickFreq()***
- ***HAL_Delay()***
- ***HAL_SuspendTick()***
- ***HAL_ResumeTick()***
- ***HAL_GetHalVersion()***
- ***HAL_GetREVID()***
- ***HAL_GetDEVID()***
- ***HAL_GetUIDw0()***
- ***HAL_GetUIDw1()***
- ***HAL_GetUIDw2()***

6.1.4 HAL Debug functions

This section provides functions allowing to:

- Enable/Disable Debug module during Sleep mode
- Enable/Disable Debug module during Stop modes
- Enable/Disable Debug module during Stanby mode

This section contains the following APIs:

- ***HAL_DBGMCU_EnableDBGSleepMode()***
- ***HAL_DBGMCU_DisableDBGSleepMode()***
- ***HAL_DBGMCU_EnableDBGStopMode()***
- ***HAL_DBGMCU_DisableDBGStopMode()***
- ***HAL_DBGMCU_EnableDBGStandbyMode()***
- ***HAL_DBGMCU_DisableDBGStandbyMode()***

6.1.5 HAL SYSCFG configuration functions

This section provides functions allowing to:

- Enable or disable BOOT pins pull-down.
- Lock and get locks status of system items.
- Set (and get) TCM size and DTCM size.
- Lock (and get) write accesses.
- Enable or disable wait state on extended ITCM and DTCM.
- Enable or disable external RW margin input for TCM memories.
- Set and get the TCM biasing level.
- Enable or disable the external RW for caches memories.
- Set and get the cache biasing level.
- Enable or disable the Power-on reset upon SW system request.
- Enable or disable that Lockup requests a warm reset to the RCC
- Enable or disable that Lockup generates a NMI on the core
- Clear Error capture in write posting buffer

- Reserve VENCRAM for VENC or release it (any use allowed)
- Enable or disable PKA, SAES, CRYP1/2 and HASH reset in case of potential tamper
- Set/Get write/read Qos for NP1, NP2 or CPU.
- Enable or disable SDMMC early-write response.
- Enable or disable USB early-write response.
- Enable or disable the NPU NIC, NPU NOC or CPU NOC clock gating.
- Enable or disable the VDDIOx compensation cell.
- Configure or get VDDIOx compensation cells.
- Get VDDIOx compensation ready status.
- Enable or disable the VDD compensation cell.
- Configure or get VDD compensation cells.
- Configure or get the Timer Break input for error flag
- Set and get secure OS allocation of specific CID to the DMA channel
- Set and get non-secure OS allocation of specific CID to the DMA channel
- Enable or disable Retiming on RX path.
- Enable or disable Retiming on TX path.
- Set and get Delay on feedback clock.
- Enable or disable the interleaving on NPU RAM.
- Get BOOT pin connection status
- Get the address of first error in P-AHB write-posting buffer

This section contains the following APIs:

- ***HAL_SYSCFG_EnablePullDown()***
- ***HAL_SYSCFG_DisablePullDown()***
- ***HAL_SYSCFG_Lock()***
- ***HAL_SYSCFG_GetLock()***
- ***HAL_SYSCFG_SetTCMSize()***
- ***HAL_SYSCFG_GetTCMSize()***
- ***HAL_SYSCFG_LockWriteAccess()***
- ***HAL_SYSCFG_GetLockWriteAccess()***
- ***HAL_SYSCFG_EnableITCMWaiteState()***
- ***HAL_SYSCFG_DisableITCMWaiteState()***
- ***HAL_SYSCFG_EnableDTCMWaiteState()***
- ***HAL_SYSCFG_DisableDTCMWaiteState()***
- ***HAL_SYSCFG_EnableTCMExternalMargin()***
- ***HAL_SYSCFG_DisableTCMExternalMargin()***
- ***HAL_SYSCFG_SetTCMRWMarginInput()***
- ***HAL_SYSCFG_GetTCMRWMarginInput()***
- ***HAL_SYSCFG_SetTCMBiasingLevel()***
- ***HAL_SYSCFG_GetTCMBiasingLevel()***
- ***HAL_SYSCFG_EnableCacheExternalMargin()***
- ***HAL_SYSCFG_DisableCacheExternalMargin()***
- ***HAL_SYSCFG_SetCacheRWMarginInput()***
- ***HAL_SYSCFG_GetCacheRWMarginInput()***
- ***HAL_SYSCFG_SetCacheBiasingLevel()***
- ***HAL_SYSCFG_GetCacheBiasingLevel()***
- ***HAL_SYSCFG_SetSVTORAddress()***
- ***HAL_SYSCFG_GetSVTORAddress()***
- ***HAL_SYSCFG_SetNSVTORAddress()***
- ***HAL_SYSCFG_GetNSVTORAddress()***
- ***HAL_SYSCFG_EnablePowerOnReset()***

- *HAL_SYSCFG_DisablePowerOnReset()*
- *HAL_SYSCFG_EnableLockupWarmResetonRCC()*
- *HAL_SYSCFG_DisableLockupWarmResetonRCC()*
- *HAL_SYSCFG_EnableLockupGenerateNMI()*
- *HAL_SYSCFG_DisableLockupGenerateNMI()*
- *HAL_SYSCFG_ReEnableWritePostingErrorCapture()*
- *HAL_SYSCFG_EnableVENCRAMReserved()*
- *HAL_SYSCFG_DisableVENCRAMReserved()*
- *HAL_SYSCFG_EnableCRYPPotentialTamper()*
- *HAL_SYSCFG_DisableCRYPPotentialTamper()*
- *HAL_SYSCFG_SetWriteQosNP1()*
- *HAL_SYSCFG_GetWriteQosNP1()*
- *HAL_SYSCFG_SetReadQosNP1()*
- *HAL_SYSCFG_GetReadQosNP1()*
- *HAL_SYSCFG_SetWriteQosNP2()*
- *HAL_SYSCFG_GetWriteQosNP2()*
- *HAL_SYSCFG_SetReadQosNP2()*
- *HAL_SYSCFG_GetReadQosNP2()*
- *HAL_SYSCFG_SetWriteQosCPUSS()*
- *HAL_SYSCFG_GetwriteQosCPUSS()*
- *HAL_SYSCFG_SetReadQosCPUSS()*
- *HAL_SYSCFG_GetReadQosCPUSS()*
- *HAL_SYSCFG_EnableSDMMCEarlyWRRESP()*
- *HAL_SYSCFG_DisableSDMMCEarlyWRRSP()*
- *HAL_SYSCFG_EnableUSBEarlyEarlyWRRESP()*
- *HAL_SYSCFG_DisableUSBEarlyEarlyWRRESP()*
- *HAL_SYSCFG_EnablexPUClockGating()*
- *HAL_SYSCFG_DisablexPUClockGating()*
- *HAL_SYSCFG_EnableVDDIO2CompensationCell()*
- *HAL_SYSCFG_DisableVDDIO2CompensationCell()*
- *HAL_SYSCFG_EnableVDDIO3CompensationCell()*
- *HAL_SYSCFG_DisableVDDIO3CompensationCell()*
- *HAL_SYSCFG_EnableVDDIO4CompensationCell()*
- *HAL_SYSCFG_DisableVDDIO4CompensationCell()*
- *HAL_SYSCFG_EnableVDDIO5CompensationCell()*
- *HAL_SYSCFG_DisableVDDIO5CompensationCell()*
- *HAL_SYSCFG_ConfigVDDIOCompensationCell()*
- *HAL_SYSCFG_GetVDDIOCompensationCell()*
- *HAL_SYSCFG_GetCompVDDIO2CellReadyStatus()*
- *HAL_SYSCFG_GetCompVDDIO3CellReadyStatus()*
- *HAL_SYSCFG_GetCompVDDIO4CellReadyStatus()*
- *HAL_SYSCFG_GetCompVDDIO5CellReadyStatus()*
- *HAL_SYSCFG_ConfigVDDCompensationCell()*
- *HAL_SYSCFG_EnableVDDCompensationCell()*
- *HAL_SYSCFG_DisableVDDCompensationCell()*
- *HAL_SYSCFG_GetVDDCompensationCell()*
- *HAL_SYSCFG_GetCompensationVDDCellReadyStatus()*
- *HAL_SYSCFG_ConfigTimerBreakInput()*
- *HAL_SYSCFG_GetTimerBreakInputConfig()*
- *HAL_SYSCFG_SetPerceivedCID()*

- *HAL_SYSCFG_GetPerceivedCID()*
- *HAL_SYSCFG_SetPerceivedPrivCID()*
- *HAL_SYSCFG_GetPerceivedPrivCID()*
- *HAL_SYSCFG_EnableReTimingRXpath()*
- *HAL_SYSCFG_DisableReTimingRXpath()*
- *HAL_SYSCFG_EnableReTimingTXpath()*
- *HAL_SYSCFG_DisableReTimingTXpath()*
- *HAL_SYSCFG_SetDelayOnFeedbackClock()*
- *HAL_SYSCFG_GetDelayOnFeedbackClock()*
- *HAL_SYSCFG_EnableInterleavingCpuRam()*
- *HAL_SYSCFG_DisableInterleavingCpuRam()*
- *HAL_SYSCFG_GetBootPinConnection()*
- *HAL_SYSCFG_GetAddressWritePostingBuffer()*

6.1.6 Detailed description of functions

HAL_Init

Function name

HAL_StatusTypeDef HAL_Init (void)

Function description

Configure the time base source, NVIC and any required global low level hardware by calling the HAL_MspInit() callback function to be optionally defined in user file stm32n6xx_hal_msp.c.

Return values

- **HAL:** status

Notes

- HAL_Init() function is called at the beginning of program after reset and before the clock configuration.
- In the default implementation the System Timer (Systick) is used as source of time base. The Systick configuration is based on HSI clock, as HSI is the clock used after a system Reset and the NVIC configuration is set to Priority group 4. Once done, time base tick starts incrementing: the tick variable counter is incremented each 1ms in the SysTick_Handler() interrupt handler.

HAL_DeInit

Function name

HAL_StatusTypeDef HAL_DeInit (void)

Function description

This function de-initializes all peripherals and low level hardware resources.

Return values

- **HAL:** status

Notes

- This function should only be called from code running in internal RAM since it resets the external peripheral memory interfaces.
- This function is optional.

HAL_MspInit

Function name

void HAL_MspInit (void)

Function description

Initializes the MSP.

Return values

- **None:**

HAL_MspDeInit

Function name

void HAL_MspDeInit (void)

Function description

DeInitializes the MSP.

Return values

- **None:**

HAL_InitTick

Function name

HAL_StatusTypeDef HAL_InitTick (uint32_t TickPriority)

Function description

This function configures the source of the time base.

Parameters

- **TickPriority:** Tick interrupt priority.

Return values

- **HAL:** status

Notes

- This function is called automatically at the beginning of program after reset by HAL_Init() or at any time when clock is reconfigured by HAL_RCC_ClockConfig().
- In the default implementation, SysTick timer is the source of time base. It is used to generate interrupts at regular time intervals. Care must be taken if HAL_Delay() is called from a peripheral ISR process, The SysTick interrupt must have higher priority (numerically lower) than the peripheral interrupt. Otherwise the caller ISR process will be blocked. The function is declared as __weak to be overwritten in case of other implementation in user file.

HAL_IncTick

Function name

void HAL_IncTick (void)

Function description

This function is called to increment a global variable "uwTick" used as application time base.

Return values

- **None:**

Notes

- In the default implementation, this variable is incremented each 1ms in Systick ISR.
- This function is declared as __weak to be overwritten in case of other implementations in user file.

HAL_Delay

Function name

void HAL_Delay (uint32_t Delay)

Function description

This function provides minimum delay (in milliseconds) based on variable incremented.

Parameters

- **Delay:** Specifies the delay time length, in milliseconds.

Return values

- **None:**

Notes

- In the default implementation , SysTick timer is the source of time base. It is used to generate interrupts at regular time intervals where uwTick is incremented.
- This function is declared as __weak to be overwritten in case of other implementations in user file.

HAL_GetTick

Function name

uint32_t HAL_GetTick (void)

Function description

Provides a tick value in millisecond.

Return values

- **tick:** value

Notes

- This function is declared as __weak to be overwritten in case of other implementations in user file.

HAL_GetTickPrio

Function name

uint32_t HAL_GetTickPrio (void)

Function description

This function returns a tick priority.

Return values

- **tick:** priority

HAL_SetTickFreq

Function name

HAL_StatusTypeDef HAL_SetTickFreq (HAL_TickFreqTypeDef Freq)

Function description

Set new tick Freq.

Parameters

- **Freq:** Tick frequency, value of HAL_TickFreqTypeDef.

Return values

- **HAL:** status

HAL_GetTickFreq

Function name

HAL_TickFreqTypeDef HAL_GetTickFreq (void)

Function description

Return tick frequency.

Return values

- **Tick:** frequency. Value of HAL_TickFreqTypeDef.

HAL_SuspendTick

Function name

void HAL_SuspendTick (void)

Function description

Suspend Tick increment.

Return values

- **None:**

Notes

- In the default implementation , SysTick timer is the source of time base. It is used to generate interrupts at regular time intervals. Once HAL_SuspendTick() is called, the SysTick interrupt will be disabled and so Tick increment is suspended.
- This function is declared as __weak to be overwritten in case of other implementations in user file.

HAL_ResumeTick

Function name

void HAL_ResumeTick (void)

Function description

Resume Tick increment.

Return values

- **None:**

Notes

- In the default implementation , SysTick timer is the source of time base. It is used to generate interrupts at regular time intervals. Once HAL_ResumeTick() is called, the SysTick interrupt will be enabled and so Tick increment is resumed.
- This function is declared as __weak to be overwritten in case of other implementations in user file.

HAL_GetHalVersion

Function name

uint32_t HAL_GetHalVersion (void)

Function description

Returns the HAL revision.

Return values

- **version:** : 0xXYZR (8bits for each decimal, R for RC)

HAL_GetREVID

Function name

uint32_t HAL_GetREVID (void)

Function description

Returns the device revision identifier.

Return values

- **Device:** revision identifier

Notes

- Returned Revision ID can be 0x00000100 for Cut1.0 0x00000101 for Cut1.1 0x00000200 for Cut2.0

HAL_GetDEVID

Function name

uint32_t HAL_GetDEVID (void)

Function description

Returns the device identifier.

Return values

- **Device:** identifier

Notes

- This function can only be used if the HAL BSEC module is enabled in the hal configuration file. The BSEC clock must be enabled before calling this function Returned Device ID can be 0x00006200 for STM32N645xx 0x00006000 for STM32N655xx 0x00002200 for STM32N647xx 0x00002000 for STM32N657xx 0xFFFFFFFF if an error occurs

HAL_GetUIDw0

Function name

uint32_t HAL_GetUIDw0 (void)

Function description

Return the first word of the unique device identifier (UID based on 96 bits)

Return values

- **Device:** identifier

Notes

- The application must ensures that SYSCFG clock is enabled.

HAL_GetUIDw1

Function name

uint32_t HAL_GetUIDw1 (void)

Function description

Return the second word of the unique device identifier (UID based on 96 bits)

Return values

- **Device:** identifier

Notes

- The application must ensures that SYSCFG clock is enabled.

HAL_GetUIDw2

Function name

uint32_t HAL_GetUIDw2 (void)

Function description

Return the third word of the unique device identifier (UID based on 96 bits)

Return values

- **Device:** identifier

Notes

- The application must ensures that SYSCFG clock is enabled.

HAL_DBGMCU_EnableDBGSleepMode

Function name

void HAL_DBGMCU_EnableDBGSleepMode (void)

Function description

Enable the Debug Module during Sleep mode.

Return values

- **None:**

HAL_DBGMCU_DisableDBGSleepMode

Function name

void HAL_DBGMCU_DisableDBGSleepMode (void)

Function description

Disable the Debug Module during Sleep mode.

Return values

- **None:**

HAL_DBGMCU_EnableDBGStopMode

Function name

void HAL_DBGMCU_EnableDBGStopMode (void)

Function description

Enable the Debug Module during Stop mode.

Return values

- **None:**

HAL_DBGMCU_DisableDBGStopMode

Function name

void HAL_DBGMCU_DisableDBGStopMode (void)

Function description

Disable the Debug Module during Stop mode.

Return values

- **None:**

HAL_DBGMCU_EnableDBGStandbyMode

Function name

void HAL_DBGMCU_EnableDBGStandbyMode (void)

Function description

Enable the Debug Module during Stanby mode.

Return values

- **None:**

HAL_DBGMCU_DisableDBGStandbyMode

Function name

void HAL_DBGMCU_DisableDBGStandbyMode (void)

Function description

Disable the Debug Module during Stanby mode.

Return values

- **None:**

HAL_SYSCFG_EnablePullDown

Function name

void HAL_SYSCFG_EnablePullDown (uint32_t BootId)

Function description

Enable the Pull Down on BOOT pin.

Parameters

- **BootId:** Specifies the boot pins. This parameter can one of the following values:
  - SYSCFG_BOOT_0 boot 0 pin
  - SYSCFG_BOOT_1 boot 1 pin

Return values

- **None:**

HAL_SYSCFG_DisablePullDown

Function name

void HAL_SYSCFG_DisablePullDown (uint32_t BootId)

Function description

Disable the Pull Down on BOOT pin.

Parameters

- **BootId:** Specifies the boot pins. This parameter can one of the following values:
  - SYSCFG_BOOT_0 boot 0 pin
  - SYSCFG_BOOT_1 boot 1 pin

Return values

- **None:**

HAL_SYSCFG_Lock

Function name

void HAL_SYSCFG_Lock (uint32_t Item)

Function description

Lock the SYSCFG item(s).

Parameters

- **Item:** Item(s) to set lock on. This parameter can be a combination of SYSCFG Lock items

Return values

- **None:**

Notes

- Lock(s) cleared only at system reset

HAL_SYSCFG_GetLock

Function name

HAL_StatusTypeDef HAL_SYSCFG_GetLock (uint32_t * pItem)

Function description

Get the lock state of SYSCFG item.

Parameters

- **pItem:** Pointer to return locked items the return value can be a combination of SYSCFG Lock items

Return values

- **HAL:** status

HAL_SYSCFG_SetTCMSize

Function name

void HAL_SYSCFG_SetTCMSize (uint32_t DtcmSize, uint32_t ItcmSize)

Function description

Set the DTCM and ITCM memory sizes.

Parameters

- **DtcmSize:** DTCM memory size This parameter can one of the following values:
  - SYSCFG_DTCM_128K 128K
  - SYSCFG_DTCM_256K 256K
- **ItcmSize:** ITCM memory size This parameter can one of the following values:
  - SYSCFG_ITCM_64K 64K
  - SYSCFG_ITCM_128K 128K
  - SYSCFG_ITCM_256K 256K

Return values

- **None:**

Notes

- Write-once TCM configuration change.
- Effective change applied upon new power-on reset by calling HAL_SYSCFG_EnablePowerOnReset() and NVIC_SystemReset()

HAL_SYSCFG_GetTCMSize

Function name

HAL_StatusTypeDef HAL_SYSCFG_GetTCMSize (uint32_t * pDtcmSize, uint32_t * pItcmSize)

Function description

Get the DTCM and ITCM memory sizes.

Parameters

- **pDtcmSize:** Pointer to return DTCM memory size This return value can be one of
  - SYSCFG_DTCM_128K 128K
  - SYSCFG_DTCM_256K 256K
- **pItcmSize:** Pointer to return ITCM memory size This return value can be one of
  - SYSCFG_ITCM_64K 64K
  - SYSCFG_ITCM_128K 128K
  - SYSCFG_ITCM_256K 256K

Return values

- **HAL:** status

HAL_SYSCFG_LockWriteAccess

Function name

void HAL_SYSCFG_LockWriteAccess (uint32_t Item)

Function description

Lock the SYSCFG write access item(s).

Parameters

- **Item:** Item(s) to set lock on. This parameter can be a combination of WRITE access selection

Return values

- **None:**

Notes

- Lock(s) cleared only at system reset

HAL_SYSCFG_GetLockWriteAccess

Function name

HAL_StatusTypeDef HAL_SYSCFG_GetLockWriteAccess (uint32_t * pItem)

Function description

Get the lock state of SYSCFG write access item.

Parameters

- **pItem:** pointer to return locked items the return value can be a combination of WRITE access selection

Return values

- **HAL:** status

HAL_SYSCFG_EnableITCMWaiteState

Function name

void HAL_SYSCFG_EnableITCMWaiteState (void)

Function description

Enable the ITCM Wait State.

Return values

- **None:**

HAL_SYSCFG_DisableITCMWaiteState

Function name

void HAL_SYSCFG_DisableITCMWaiteState (void)

Function description

Disable the ITCM Wait State.

Return values

- **None:**

HAL_SYSCFG_EnableDTCMWaiteState

Function name

void HAL_SYSCFG_EnableDTCMWaiteState (void)

Function description

Enable the DTCM Wait State.

Return values

- **None:**

HAL_SYSCFG_DisableDTCMWaiteState

Function name

void HAL_SYSCFG_DisableDTCMWaiteState (void)

Function description

Disable the DTCM Wait State.

Return values

- **None:**

HAL_SYSCFG_EnableTCMExternalMargin

Function name

void HAL_SYSCFG_EnableTCMExternalMargin (void)

Function description

Enable external RW margin inputs for TCM memories.

Return values

- **None:**

HAL_SYSCFG_DisableTCMExternalMargin

Function name

void HAL_SYSCFG_DisableTCMExternalMargin (void)

Function description

Disable external RW margin inputs for TCM memories.

Return values

- **None:**

HAL_SYSCFG_SetTCMRWMarginInput

Function name

void HAL_SYSCFG_SetTCMRWMarginInput (uint32_t TcmRwMarginInput)

Function description

Set external RW margin inputs for TCM memories.

Parameters

- **TcmRwMarginInput:** Input Margin.

Return values

- **None:**

HAL_SYSCFG_GetTCMRWMarginInput

Function name

HAL_StatusTypeDef HAL_SYSCFG_GetTCMRWMarginInput (uint32_t * pTcmRwMarginInput)

Function description

Get the External RW margin inputs for TCM memories.

Parameters

- **pTcmRwMarginInput:** pointer to return Input Margin.

Return values

- **HAL:** status.

HAL_SYSCFG_SetTCMBiasingLevel

Function name

void HAL_SYSCFG_SetTCMBiasingLevel (uint32_t Level)

Function description

Set the TCM biasing level adjust input.

Parameters

- **Level:** Biaising level adjust input This parameter can one of the following values:
  - SYSCFG_TCM_BIAS_VNOM : Biasing level adjust input recommended for Vnom
  - SYSCFG_TCM_BIAS_VNOM_10_PERCENT : Biasing level adjust input recommended for Vnom + 10%

Return values

- **None:**

HAL_SYSCFG_GetTCMBiasingLevel

Function name

HAL_StatusTypeDef HAL_SYSCFG_GetTCMBiasingLevel (uint32_t * pLevel)

Function description

Get the TCM biasing level adjust input.

Parameters

- **pLevel:** Pointer to return biasing level adjust input. The return value can be one of the following values:
  - SYSCFG_CACHE_BIAS_VNOM boot 0 pin: Biasing level adjust input recommended for Vnom
  - SYSCFG_CACHE_BIAS_VNOM_10_PERCENT: Biasing level adjust input recommended for Vnom + 10%

Return values

- **HAL:** status

HAL_SYSCFG_EnableCacheExternalMargin

Function name

void HAL_SYSCFG_EnableCacheExternalMargin (void)

Function description

Enable external RW margin inputs for Cache memories.

Return values

- **None:**

HAL_SYSCFG_DisableCacheExternalMargin

Function name

void HAL_SYSCFG_DisableCacheExternalMargin (void)

Function description

Disable external RW margin inputs for Cache memories.

Return values

- **None:**

HAL_SYSCFG_SetCacheRWMarginInput

Function name

void HAL_SYSCFG_SetCacheRWMarginInput (uint32_t CacheRWMarginInput)

Function description

Set External RW margin inputs for Cache memories.

Parameters

- **CacheRWMarginInput:** Input Margin.

Return values

- **None:**

HAL_SYSCFG_GetCacheRWMarginInput

Function name

HAL_StatusTypeDef HAL_SYSCFG_GetCacheRWMarginInput (uint32_t * pCacheRWMarginInput)

Function description

Get the External RW margin inputs for Cache memories.

Parameters

- **pCacheRWMarginInput:** Pointer to return Input Margin.

Return values

- **HAL:** status

HAL_SYSCFG_SetCacheBiasingLevel

Function name

void HAL_SYSCFG_SetCacheBiasingLevel (uint32_t Level)

Function description

Set the Cache biasing level adjust input.

Parameters

- **Level:** the biaising level adjust input This parameter can one of the following values:
  - SYSCFG_CACHE_BIAS_VNOM: Biasing level adjust input recommended for Vnom
  - SYSCFG_CACHE_BIAS_VNOM_10_PERCENT: Biasing level adjust input recommended for Vnom + 10%

Return values

- **None:**

HAL_SYSCFG_GetCacheBiasingLevel

Function name

HAL_StatusTypeDef HAL_SYSCFG_GetCacheBiasingLevel (uint32_t * pLevel)

Function description

Get the Cache biasing level adjust input.

Parameters

- **pLevel:** Pointer to return Cache biasing level adjust input. The return value can be onr of the following values:
  - SYSCFG_CACHE_BIAS_VNOM boot: Biasing level adjust input recommended for Vnom
  - SYSCFG_CACHE_BIAS_VNOM_10_PERCENT: Biasing level adjust input recommended for Vnom + 10%

Return values

- **HAL:** status

HAL_SYSCFG_SetSVTORAddress

Function name

void HAL_SYSCFG_SetSVTORAddress (uint32_t Address)

Function description

Set the secure vector table (VTOR) address.

Parameters

- **Address:** Secure VTOR address

Return values

- **None:**

Notes

- Secure vector table address must be set to ensure return from STANDBY state.

HAL_SYSCFG_GetSVTORAddress

Function name

HAL_StatusTypeDef HAL_SYSCFG_GetSVTORAddress (uint32_t * pAddress)

Function description

Get the secure vector table (VTOR) address.

Parameters

- **pAddress:** Pointer to return the secure vector table address

Return values

- **HAL:** status

HAL_SYSCFG_SetNSVTORAddress

Function name

void HAL_SYSCFG_SetNSVTORAddress (uint32_t Address)

Function description

Set the non secure vector table (VTOR) address.

Parameters

- **Address:** Non-secure VTOR address

Return values

- **None:**

HAL_SYSCFG_GetNSVTORAddress

Function name

HAL_StatusTypeDef HAL_SYSCFG_GetNSVTORAddress (uint32_t * pAddress)

Function description

Get the non secure VTOR address.

Parameters

- **pAddress:** Pointer to return the non-secure vector table address

Return values

- **HAL:** status

HAL_SYSCFG_EnablePowerOnReset

Function name

void HAL_SYSCFG_EnablePowerOnReset (void)

Function description

Enable the Power-on reset.

Return values

- **None:**

Notes

- Select power-on reset to apply on core upon SYSRESETREQ (NVIC_SystemReset())

HAL_SYSCFG_DisablePowerOnReset

Function name

void HAL_SYSCFG_DisablePowerOnReset (void)

Function description

Disable the Power-on reset.

Return values

- **None:**

Notes

- Select warm reset to apply on core upon SYSRESETREQ (NVIC_SystemReset())

HAL_SYSCFG_EnableLockupWarmResetonRCC

Function name

void HAL_SYSCFG_EnableLockupWarmResetonRCC (void)

Function description

Enable Lockup requests a warm reset to the RCC.

Return values

- **None:**

Notes

- Select action to perform on a lockup state on the core

HAL_SYSCFG_DisableLockupWarmResetonRCC

Function name

void HAL_SYSCFG_DisableLockupWarmResetonRCC (void)

Function description

Disable Lockup requests a warm reset to the RCC.

Return values

- **None:**

Notes

- Select action to perform on a lockup state on the core

HAL_SYSCFG_EnableLockupGenerateNMI

Function name

void HAL_SYSCFG_EnableLockupGenerateNMI (void)

Function description

Enable Lockup generates a NMI on the core.

Return values

- **None:**

Notes

- Select action to perform on a lockup state on the core

HAL_SYSCFG_DisableLockupGenerateNMI

Function name

void HAL_SYSCFG_DisableLockupGenerateNMI (void)

Function description

Disable Lockup generates a NMI on the core.

Return values

- **None:**

Notes

- Select action to perform on a lockup state on the core

HAL_SYSCFG_ReEnableWritePostingErrorCapture

Function name

HAL_StatusTypeDef HAL_SYSCFG_ReEnableWritePostingErrorCapture (void)

Function description

Enable the capture mechanism upon write posting buffer error.

Return values

- **HAL:** status.

Notes

- Upon error detection, the erroneous address is logged on and the capture mechanism is on hold. The address error can be read thanks to HAL_SYSCFG_GetAddressWritePostingBuffer.
- This function enables the error capture mechanism again.

HAL_SYSCFG_EnableVENCRAMReserved

Function name

void HAL_SYSCFG_EnableVENCRAMReserved (void)

Function description

Reserve the VENCRAM allocation to VENC.

Return values

- **None:**

HAL_SYSCFG_DisableVENCRAMReserved

Function name

void HAL_SYSCFG_DisableVENCRAMReserved (void)

Function description

Release the VENCRAM allocation to VENC (any use allowed).

Return values

- **None:**

HAL_SYSCFG_EnableCRYPPotentialTamper

Function name

void HAL_SYSCFG_EnableCRYPPotentialTamper (void)

Function description

Enable PKA, SAES, CRYP1/2, and HASH reset, in case of potential tamper.

Return values

- **None:**

HAL_SYSCFG_DisableCRYPPotentialTamper

Function name

void HAL_SYSCFG_DisableCRYPPotentialTamper (void)

Function description

Disable PKA, SAES, CRYP1/2, and HASH reset, in case of potential tamper.

Return values

- **None:**

HAL_SYSCFG_SetWriteQosNP1

Function name

void HAL_SYSCFG_SetWriteQosNP1 (uint32_t QosValue)

Function description

Set write QoS information for master port from NP1 NPUNIC.

Parameters

- **QosValue:** Write QoS value (0 to 15).

Return values

- **None:**

HAL_SYSCFG_GetWriteQosNP1

Function name

HAL_StatusTypeDef HAL_SYSCFG_GetWriteQosNP1 (uint32_t * pQosValue)

Function description

Get write QoS information for master port from NP1 NPUNIC.

Parameters

- **pQosValue:** Pointer to return write QOS value (0 to 15)

Return values

- **HAL:** status

HAL_SYSCFG_SetReadQosNP1

Function name

void HAL_SYSCFG_SetReadQosNP1 (uint32_t QosValue)

Function description

Set read QoS information for master port from NP1 NPUNIC.

Parameters

- **QosValue:** Read QoS value (0 to 15)

Return values

- **None:**

HAL_SYSCFG_GetReadQosNP1

Function name

HAL_StatusTypeDef HAL_SYSCFG_GetReadQosNP1 (uint32_t * pQosValue)

Function description

Get read QoS information for master port from NP1 NPUNIC.

Parameters

- **pQosValue:** Pointer to return read QOS value (0 to 15)

Return values

- **HAL:** status

HAL_SYSCFG_SetWriteQosNP2

Function name

void HAL_SYSCFG_SetWriteQosNP2 (uint32_t QosValue)

Function description

Set write QoS information for master port from NP2 NPUNIC.

Parameters

- **QosValue:** Write Qos value (0 to 15)

Return values

- **None:**

HAL_SYSCFG_GetWriteQosNP2

Function name

HAL_StatusTypeDef HAL_SYSCFG_GetWriteQosNP2 (uint32_t * pQosValue)

Function description

Get write QoS information for master port from NP2 NPUNIC.

Parameters

- **pQosValue:** Pointer to return write QOS value (0 to 15)

Return values

- **HAL:** status

HAL_SYSCFG_SetReadQosNP2

Function name

void HAL_SYSCFG_SetReadQosNP2 (uint32_t QosValue)

Function description

Set read QoS information for master port from NP2 NPUNIC.

Parameters

- **QosValue:** Read Qos value (0 to 15)

Return values

- **None:**

HAL_SYSCFG_GetReadQosNP2

Function name

HAL_StatusTypeDef HAL_SYSCFG_GetReadQosNP2 (uint32_t * pQosValue)

Function description

Get read QoS information for master port from NP2 NPUNIC.

Parameters

- **pQosValue:** Pointer to return read QOS value (0 to 15)

Return values

- **HAL:** status

HAL_SYSCFG_SetWriteQosCPUSS

Function name

void HAL_SYSCFG_SetWriteQosCPUSS (uint32_t QosValue)

Function description

Set write QoS information for master port from CPUSS NPUNIC.

Parameters

- **QosValue:** Write QoS value (0 to 15)

Return values

- **None:**

HAL_SYSCFG_GetwriteQosCPUSS

Function name

HAL_StatusTypeDef HAL_SYSCFG_GetwriteQosCPUSS (uint32_t * pQosValue)

Function description

Get write QoS information for master port from CPUSS NPUNIC.

Parameters

- **pQosValue:** Pointer to return write QOS value (0 to 15)

Return values

- **HAL:** status

HAL_SYSCFG_SetReadQosCPUSS

Function name

void HAL_SYSCFG_SetReadQosCPUSS (uint32_t QosValue)

Function description

Set read QoS information for master port from CPUSS NPUNIC.

Parameters

- **QosValue:** Read QoS value (0 to 15)

Return values

- **None:**

HAL_SYSCFG_GetReadQosCPUSS

Function name

HAL_StatusTypeDef HAL_SYSCFG_GetReadQosCPUSS (uint32_t * pQosValue)

Function description

Get read QoS information for master port from CPUSS NPUNIC.

Parameters

- **pQosValue:** Pointer to return read QOS value (0 to 15)

Return values

- **HAL:** status

HAL_SYSCFG_EnableSDMMCEarlyWRRESP

Function name

void HAL_SYSCFG_EnableSDMMCEarlyWRRESP (uint32_t Sdmmc)

Function description

Enable SDMMC early-write response.

Parameters

- **Sdmmc:** Selected SDMMC instance(s). This parameter can be a one of SDMMC Id

Return values

- **None:**

HAL_SYSCFG_DisableSDMMCEarlyWRRSP

Function name

void HAL_SYSCFG_DisableSDMMCEarlyWRRSP (uint32_t Sdmmc)

Function description

Disable SDMMC early-write response.

Parameters

- **Sdmmc:** Selected SDMMC instance(s). This parameter can be a one of SDMMC Id

Return values

- **None:**

HAL_SYSCFG_EnableUSBEarlyEarlyWRRESP

Function name

void HAL_SYSCFG_EnableUSBEarlyEarlyWRRESP (uint32_t Usb)

Function description

Enable USB early-write response.

Parameters

- **Usb:** Selected USB instance(s). This parameter can be a one of USB Id

Return values

- **None:**

HAL_SYSCFG_DisableUSBEarlyEarlyWRRESP

Function name

void HAL_SYSCFG_DisableUSBEarlyEarlyWRRESP (uint32_t Usb)

Function description

Disable USB early-write response.

Parameters

- **Usb:** Selected USB instance(s). This parameter can be a one of USB Id

Return values

- **None:**

HAL_SYSCFG_EnablexPUClockGating

Function name

void HAL_SYSCFG_EnablexPUClockGating (uint32_t Xpu)

Function description

Enable xPU clock gating.

Parameters

- **Xpu:** Selected CPU(s). This parameter can be a one of XPU Id

Return values

- **None:**

HAL_SYSCFG_DisablexPUClockGating

Function name

void HAL_SYSCFG_DisablexPUClockGating (uint32_t Xpu)

Function description

Disable xPU clock gating.

Parameters

- **Xpu:** Selected CPU(s). This parameter can be a one of XPU Id

Return values

- **None:**

HAL_SYSCFG_EnableVDDIO2CompensationCell

Function name

void HAL_SYSCFG_EnableVDDIO2CompensationCell (void)

Function description

Enable the VDDIO2 compensation cell.

Return values

- **None:**

HAL_SYSCFG_DisableVDDIO2CompensationCell

Function name

void HAL_SYSCFG_DisableVDDIO2CompensationCell (void)

Function description

Disable the VDDIO2 compensation cell.

Return values

- **None:**

HAL_SYSCFG_EnableVDDIO3CompensationCell

Function name

void HAL_SYSCFG_EnableVDDIO3CompensationCell (void)

Function description

Enable the VDDIO3 compensation cell.

Return values

- **None:**

HAL_SYSCFG_DisableVDDIO3CompensationCell

Function name

void HAL_SYSCFG_DisableVDDIO3CompensationCell (void)

Function description

Disable the VDDIO3 compensation cell.

Return values

- **None:**

HAL_SYSCFG_EnableVDDIO4CompensationCell

Function name

void HAL_SYSCFG_EnableVDDIO4CompensationCell (void)

Function description

Enable the VDDIO4 compensation cell.

Return values

- **None:**

HAL_SYSCFG_DisableVDDIO4CompensationCell

Function name

void HAL_SYSCFG_DisableVDDIO4CompensationCell (void)

Function description

Disable the VDDIO4 compensation cell.

Return values

- **None:**

HAL_SYSCFG_EnableVDDIO5CompensationCell

Function name

void HAL_SYSCFG_EnableVDDIO5CompensationCell (void)

Function description

Enable the VDDIO5 compensation cell.

Return values

- **None:**

HAL_SYSCFG_DisableVDDIO5CompensationCell

Function name

void HAL_SYSCFG_DisableVDDIO5CompensationCell (void)

Function description

Disable the VDDIO5 compensation cell.

Return values

- **None:**

HAL_SYSCFG_ConfigVDDIOCompensationCell

Function name

HAL_StatusTypeDef HAL_SYSCFG_ConfigVDDIOCompensationCell (uint32_t Selection, uint32_t Code, uint32_t NmosValue, uint32_t PmosValue)

Function description

Configure the code selection for the compensation cell.

Parameters

- **Selection:** specifies the concerned compensation cell This parameter can one of the following values:
  - SYSCFG_IO_VDDIO2_CELL Compensation cell for the VDDIO2
  - SYSCFG_IO_VDDIO3_CELL Compensation cell for the VDDIO3
  - SYSCFG_IO_VDDIO4_CELL Compensation cell for the VDDIO4
  - SYSCFG_IO_VDDIO5_CELL Compensation cell for the VDDIO5
- **Code:** code selection to be applied for the I/O compensation cell This parameter can be one of the following values:
  - SYSCFG_IO_CELL_CODE Code from the cell (available in the SYSCFG_VDDIOxCCSR)
  - SYSCFG_IO_REGISTER_CODE Code from the compensation cell code register ((SYSCFG_VDDIOxCCCR)
- **NmosValue:** In case SYSCFG_IO_REGISTER_CODE is selected, it provides the Nmos value to apply in range 0 to 15 else this parameter is not used
- **PmosValue:** In case SYSCFG_IO_REGISTER_CODE is selected, it provides the Pmos value to apply in range 0 to 15 else this parameter is not used

Return values

- **HAL:** status

HAL_SYSCFG_GetVDDIOCompensationCell

Function name

HAL_StatusTypeDef HAL_SYSCFG_GetVDDIOCompensationCell (uint32_t Selection, const uint32_t * pCode, uint32_t * pNmosValue, uint32_t * pPmosValue)

Function description

Get the code selection for the compensation cell.

Parameters

- **Selection:** specifies the concerned compensation cell This parameter can one of the following values:
  - SYSCFG_IO_VDDIO2_CELL Compensation cell for the VDDIO2
  - SYSCFG_IO_VDDIO3_CELL Compensation cell for the VDDIO3
  - SYSCFG_IO_VDDIO4_CELL Compensation cell for the VDDIO4
  - SYSCFG_IO_VDDIO5_CELL Compensation cell for the VDDIO5
- **pCode:** pointer to code selection This parameter can be one of the following values:
  - SYSCFG_IO_CELL_CODE Code from the cell (available in the SYSCFG_VDDIOxCCSR)
  - SYSCFG_IO_REGISTER_CODE Code from the compensation cell code register ((SYSCFG_VDDIOxCCCR)
- **pNmosValue:** pointer to the Nmos value in range 0 to 15
- **pPmosValue:** pointer to the Pmos value in range 0 to 15

Return values

- **HAL_OK:** (all values available) or HAL_ERROR (check parameters)

HAL_SYSCFG_GetCompVDDIO2CellReadyStatus

Function name

uint32_t HAL_SYSCFG_GetCompVDDIO2CellReadyStatus (void)

Function description

Get the compensation cell ready status of VDDIO2.

Return values

- **Ready:** status (0 (not ready) or !=0 (ready))

HAL_SYSCFG_GetCompVDDIO3CellReadyStatus

Function name

uint32_t HAL_SYSCFG_GetCompVDDIO3CellReadyStatus (void)

Function description

Get the compensation cell ready status of VDDIO3.

Return values

- **Ready:** status (0 (not ready) or !=0 (ready))

HAL_SYSCFG_GetCompVDDIO4CellReadyStatus

Function name

uint32_t HAL_SYSCFG_GetCompVDDIO4CellReadyStatus (void)

Function description

Get the compensation cell ready status of VDDIO4.

Return values

- **Ready:** status (0 (not ready) or !=0 (ready))

HAL_SYSCFG_GetCompVDDIO5CellReadyStatus

Function name

uint32_t HAL_SYSCFG_GetCompVDDIO5CellReadyStatus (void)

Function description

Get the compensation cell ready status of VDDIO5.

Return values

- **Ready:** status (0 (not ready) or !=0 (ready))

HAL_SYSCFG_ConfigVDDCompensationCell

Function name

HAL_StatusTypeDef HAL_SYSCFG_ConfigVDDCompensationCell (uint32_t Code, uint32_t NmosValue, uint32_t PmosValue)

Function description

Configure the code selection for the compensation cell of VDD.

Parameters

- **Code:** code selection to be applied for the I/O compensation cell This parameter can be one of the following values:
  - SYSCFG_IO_CELL_CODE Code from the cell (available in the SYSCFG_VDDIOxCCSR)
  - SYSCFG_IO_REGISTER_CODE Code from the compensation cell code register ((SYSCFG_VDDIOxCCCR)
- **NmosValue:** In case SYSCFG_IO_REGISTER_CODE is selected, it provides the Nmos value to apply in range 0 to 15 else this parameter is not used
- **PmosValue:** In case SYSCFG_IO_REGISTER_CODE is selected, it provides the Pmos value to apply in range 0 to 15 else this parameter is not used

Return values

- **HAL:** status

HAL_SYSCFG_EnableVDDCompensationCell

Function name

void HAL_SYSCFG_EnableVDDCompensationCell (void)

Function description

Enable the VDD compensation cell.

Return values

- **None:**

HAL_SYSCFG_DisableVDDCompensationCell

Function name

void HAL_SYSCFG_DisableVDDCompensationCell (void)

Function description

Disable the VDD compensation cell.

Return values

- **None:**

HAL_SYSCFG_GetVDDCompensationCell

Function name

HAL_StatusTypeDef HAL_SYSCFG_GetVDDCompensationCell (uint32_t Code, uint32_t * pNmosValue, uint32_t * pPmosValue)

Function description

Get the code selection for the compensation cell of VDD.

Parameters

- **Code:** code selection This parameter can be one of the following values:
  - SYSCFG_IO_CELL_CODE Code from the cell (available in the SYSCFG_VDDIOxCCSR)
  - SYSCFG_IO_REGISTER_CODE Code from the compensation cell code register ((SYSCFG_VDDIOxCCCR)
- **pNmosValue:** pointer to the Nmos value in range 0 to 15
- **pPmosValue:** pointer to the Pmos value in range 0 to 15

Return values

- **HAL_OK:** (all values available) or HAL_ERROR (check parameters)

HAL_SYSCFG_GetCompensationVDDCellReadyStatus

Function name

uint32_t HAL_SYSCFG_GetCompensationVDDCellReadyStatus (void)

Function description

Get the compensation cell ready status of VDD.

Return values

- **Ready:** status (0 (not ready) or !=0 (ready))

HAL_SYSCFG_ConfigTimerBreakInput

Function name

void HAL_SYSCFG_ConfigTimerBreakInput (uint32_t Input)

Function description

Configure the Timer Break input for error flag(s).

Parameters

- **Input:** Input configuration This parameter can be one or a combination of the following values:
  - SYSCFG_CBR_BREAK_LOCK_CORE Cortex-CM55 lockup
  - SYSCFG_CBR_BREAK_LOCK_PVD PVD lock
  - SYSCFG_CBR_BREAK_LOCK_BKPRAM Backup SRAM ECC error
  - SYSCFG_CBR_BREAK_LOCK_CM55_CACHE CM55 cache double ECC error
  - SYSCFG_CBR_BREAK_LOCK_CM55_TCM DTCM double ECC error

Return values

- **None:**

Notes

- When a configuration is set, only a system reset can reset it.

HAL_SYSCFG_GetTimerBreakInputConfig

Function name

uint32_t HAL_SYSCFG_GetTimerBreakInputConfig (void)

Function description

Get the Timer Break input configuration.

Return values

- **Timer:** break input configuration This return value can be one or a combination of the following values:
  - SYSCFG_CBR_BREAK_LOCK_CORE Cortex-CM55 lockup
  - SYSCFG_CBR_BREAK_LOCK_PVD PVD lock
  - SYSCFG_CBR_BREAK_LOCK_BKPRAM Backup SRAM ECC error
  - SYSCFG_CBR_BREAK_LOCK_CM55_CACHE CM55 cache double ECC error
  - SYSCFG_CBR_BREAK_LOCK_CM55_TCM DTCM double ECC error

Notes

- When a configuration is set, only a system reset can reset it.

HAL_SYSCFG_SetPerceivedCID

Function name

void HAL_SYSCFG_SetPerceivedCID (uint32_t Cid)

Function description

Set the perceived CID of the peripheral.

Parameters

- **Cid:** CID in range 0 to 7

Return values

- **None:**

Notes

- Set the perceived CID so that all accesses which arrive at the peripheral (here HPDMA) appear to carry that CID. See Multi-tenancy section of the reference manual for further information.

HAL_SYSCFG_GetPerceivedCID

Function name

uint32_t HAL_SYSCFG_GetPerceivedCID (void)

Function description

Get the perceived CID of the peripheral.

Return values

- **CID:** in range 0 to 7

HAL_SYSCFG_SetPerceivedPrivCID

Function name

void HAL_SYSCFG_SetPerceivedPrivCID (uint32_t Cid)

Function description

Set the perceived CID of the peripheral as privileged.

Parameters

- **Cid:** in range 0 to 7

Return values

- **None:**

Notes

- Set the perceived CID so that all accesses which arrive at the peripheral (here HPDMA) appear to carry that CID. This function shall be called in case of DMA channel interrupt (privileged) code to read the registers and clear the interrupt of this DMA channel. See Multi-tenancy section of the reference manual for forther information.

HAL_SYSCFG_GetPerceivedPrivCID

Function name

uint32_t HAL_SYSCFG_GetPerceivedPrivCID (void)

Function description

Get the perceived CID of the peripheral as privileged.

Return values

- **CID:** in range 0 to 7

HAL_SYSCFG_EnableReTimingRXpath

Function name

void HAL_SYSCFG_EnableReTimingRXpath (void)

Function description

Enable Retiming on RX path.

Return values

- **None:**

HAL_SYSCFG_DisableReTimingRXpath

Function name

void HAL_SYSCFG_DisableReTimingRXpath (void)

Function description

Disable Retiming on RX path.

Return values

- **None:**

HAL_SYSCFG_EnableReTimingTXpath

Function name

void HAL_SYSCFG_EnableReTimingTXpath (void)

Function description

Enable Retiming on TX path.

Return values

- **None:**

HAL_SYSCFG_DisableReTimingTXpath

Function name

void HAL_SYSCFG_DisableReTimingTXpath (void)

Function description

Disable Retiming on TX path.

Return values

- **None:**

HAL_SYSCFG_SetDelayOnFeedbackClock

Function name

void HAL_SYSCFG_SetDelayOnFeedbackClock (uint32_t Delay)

Function description

Set delay on feedback clock.

Parameters

- **Delay:** Delay on feedback clock
  - SYSCFG_DELAY_FEEDBACK_NONE None
  - SYSCFG_DELAY_FEEDBACK_HALF_CYCLE Half a cycle delay

Return values

- **None:**

HAL_SYSCFG_GetDelayOnFeedbackClock

Function name

uint32_t HAL_SYSCFG_GetDelayOnFeedbackClock (void)

Function description

Get delay on feedback clock.

Return values

- **Delay:** on feedback clock
  - SYSCFG_DELAY_FEEDBACK_NONE None
  - SYSCFG_DELAY_FEEDBACK_HALF_CYCLE Half a cycle delay

HAL_SYSCFG_EnableInterleavingCpuRam

Function name

void HAL_SYSCFG_EnableInterleavingCpuRam (void)

Function description

Enable the interleaving on NPU RAM.

Return values

- **None:**

HAL_SYSCFG_DisableInterleavingCpuRam

Function name

void HAL_SYSCFG_DisableInterleavingCpuRam (void)

Function description

Disable the interleaving on NPU RAM.

Return values

- **None:**

HAL_SYSCFG_GetBootPinConnection

Function name

uint32_t HAL_SYSCFG_GetBootPinConnection (uint32_t BootId)

Function description

Get BOOT pin connection status.

Parameters

- **BootId:** specifies the boot pins.

Return values

- **BOOT:** pin connection configuration This return value can be one of the following values:
  - SYSCFG_BOOT_CONNECTION_VSS pin connected to VSS
  - SYSCFG_BOOT_CONNECTION_VDD pin connected to VDD

HAL_SYSCFG_GetAddressWritePostingBuffer

Function name

uint32_t HAL_SYSCFG_GetAddressWritePostingBuffer (void)

Function description

Get address of first error in P-AHB write-posting buffer.

Return values

- **Address:** of the first error in P-AHB write-posting buffer

6.2 HAL Firmware driver defines

The following section lists the various define and macros of the module.

6.2.1 HAL

HAL

DBGMCU Exported Macros

__HAL_DBGMCU_FREEZE_TIM2

__HAL_DBGMCU_UNFREEZE_TIM2

__HAL_DBGMCU_FREEZE_TIM3

__HAL_DBGMCU_UNFREEZE_TIM3

__HAL_DBGMCU_FREEZE_TIM4

__HAL_DBGMCU_UNFREEZE_TIM4

__HAL_DBGMCU_FREEZE_TIM5

__HAL_DBGMCU_UNFREEZE_TIM5

__HAL_DBGMCU_FREEZE_TIM6

__HAL_DBGMCU_UNFREEZE_TIM6

__HAL_DBGMCU_FREEZE_TIM7

__HAL_DBGMCU_UNFREEZE_TIM7

__HAL_DBGMCU_FREEZE_TIM10

__HAL_DBGMCU_UNFREEZE_TIM10

__HAL_DBGMCU_FREEZE_TIM11

__HAL_DBGMCU_UNFREEZE_TIM11

__HAL_DBGMCU_FREEZE_TIM12

__HAL_DBGMCU_UNFREEZE_TIM12

__HAL_DBGMCU_FREEZE_TIM13

__HAL_DBGMCU_UNFREEZE_TIM13

__HAL_DBGMCU_FREEZE_TIM14

__HAL_DBGMCU_UNFREEZE_TIM14

__HAL_DBGMCU_FREEZE_LPTIM1

__HAL_DBGMCU_UNFREEZE_LPTIM1

__HAL_DBGMCU_FREEZE_WWDG

__HAL_DBGMCU_UNFREEZE_WWDG

__HAL_DBGMCU_FREEZE_I2C1

__HAL_DBGMCU_UNFREEZE_I2C1

__HAL_DBGMCU_FREEZE_I2C2

__HAL_DBGMCU_UNFREEZE_I2C2

__HAL_DBGMCU_FREEZE_I2C3

__HAL_DBGMCU_UNFREEZE_I2C3

__HAL_DBGMCU_FREEZE_I3C1

__HAL_DBGMCU_UNFREEZE_I3C1

__HAL_DBGMCU_FREEZE_I3C2

__HAL_DBGMCU_UNFREEZE_I3C2

__HAL_DBGMCU_FREEZE_FDCAN

__HAL_DBGMCU_UNFREEZE_FDCAN

__HAL_DBGMCU_FREEZE_TIM1

__HAL_DBGMCU_UNFREEZE_TIM1

__HAL_DBGMCU_FREEZE_TIM8

__HAL_DBGMCU_UNFREEZE_TIM8

__HAL_DBGMCU_FREEZE_TIM18

__HAL_DBGMCU_UNFREEZE_TIM18

__HAL_DBGMCU_FREEZE_TIM15

__HAL_DBGMCU_UNFREEZE_TIM15

__HAL_DBGMCU_FREEZE_TIM16

__HAL_DBGMCU_UNFREEZE_TIM16

__HAL_DBGMCU_FREEZE_TIM17

__HAL_DBGMCU_UNFREEZE_TIM17

__HAL_DBGMCU_FREEZE_TIM9

__HAL_DBGMCU_UNFREEZE_TIM9

__HAL_DBGMCU_FREEZE_I2C4

__HAL_DBGMCU_UNFREEZE_I2C4

__HAL_DBGMCU_FREEZE_LPTIM2

__HAL_DBGMCU_UNFREEZE_LPTIM2

__HAL_DBGMCU_FREEZE_LPTIM3

__HAL_DBGMCU_UNFREEZE_LPTIM3

__HAL_DBGMCU_FREEZE_LPTIM4

__HAL_DBGMCU_UNFREEZE_LPTIM4

__HAL_DBGMCU_FREEZE_LPTIM5

__HAL_DBGMCU_UNFREEZE_LPTIM5

__HAL_DBGMCU_FREEZE_RTC

__HAL_DBGMCU_UNFREEZE_RTC

__HAL_DBGMCU_FREEZE_IWDG

__HAL_DBGMCU_UNFREEZE_IWDG

__HAL_DBGMCU_FREEZE_GFXTIM

__HAL_DBGMCU_UNFREEZE_GFXTIM

__HAL_DBGMCU_FREEZE_GPDMA1_CH0

__HAL_DBGMCU_UNFREEZE_GPDMA1_CH0

__HAL_DBGMCU_FREEZE_GPDMA1_CH1

__HAL_DBGMCU_UNFREEZE_GPDMA1_CH1

__HAL_DBGMCU_FREEZE_GPDMA1_CH2

__HAL_DBGMCU_UNFREEZE_GPDMA1_CH2

__HAL_DBGMCU_FREEZE_GPDMA1_CH3

__HAL_DBGMCU_UNFREEZE_GPDMA1_CH3

__HAL_DBGMCU_FREEZE_GPDMA1_CH4

__HAL_DBGMCU_UNFREEZE_GPDMA1_CH4

__HAL_DBGMCU_FREEZE_GPDMA1_CH5

__HAL_DBGMCU_UNFREEZE_GPDMA1_CH5

__HAL_DBGMCU_FREEZE_GPDMA1_CH6

__HAL_DBGMCU_UNFREEZE_GPDMA1_CH6

__HAL_DBGMCU_FREEZE_GPDMA1_CH7

__HAL_DBGMCU_UNFREEZE_GPDMA1_CH7

__HAL_DBGMCU_FREEZE_GPDMA1_CH8

__HAL_DBGMCU_UNFREEZE_GPDMA1_CH8

__HAL_DBGMCU_FREEZE_GPDMA1_CH9

__HAL_DBGMCU_UNFREEZE_GPDMA1_CH9

__HAL_DBGMCU_FREEZE_GPDMA1_CH10

__HAL_DBGMCU_UNFREEZE_GPDMA1_CH10

__HAL_DBGMCU_FREEZE_GPDMA1_CH11

__HAL_DBGMCU_UNFREEZE_GPDMA1_CH11

__HAL_DBGMCU_FREEZE_GPDMA1_CH12

__HAL_DBGMCU_UNFREEZE_GPDMA1_CH12

__HAL_DBGMCU_FREEZE_GPDMA1_CH13

__HAL_DBGMCU_UNFREEZE_GPDMA1_CH13

__HAL_DBGMCU_FREEZE_GPDMA1_CH14

__HAL_DBGMCU_UNFREEZE_GPDMA1_CH14

__HAL_DBGMCU_FREEZE_GPDMA1_CH15

__HAL_DBGMCU_UNFREEZE_GPDMA1_CH15

__HAL_DBGMCU_FREEZE_HPDMA1_CH0

__HAL_DBGMCU_UNFREEZE_HPDMA1_CH0

__HAL_DBGMCU_FREEZE_HPDMA1_CH1

__HAL_DBGMCU_UNFREEZE_HPDMA1_CH1

__HAL_DBGMCU_FREEZE_HPDMA1_CH2

__HAL_DBGMCU_UNFREEZE_HPDMA1_CH2

__HAL_DBGMCU_FREEZE_HPDMA1_CH3

__HAL_DBGMCU_UNFREEZE_HPDMA1_CH3

__HAL_DBGMCU_FREEZE_HPDMA1_CH4

__HAL_DBGMCU_UNFREEZE_HPDMA1_CH4

__HAL_DBGMCU_FREEZE_HPDMA1_CH5

__HAL_DBGMCU_UNFREEZE_HPDMA1_CH5

__HAL_DBGMCU_FREEZE_HPDMA1_CH6

__HAL_DBGMCU_UNFREEZE_HPDMA1_CH6

__HAL_DBGMCU_FREEZE_HPDMA1_CH7

__HAL_DBGMCU_UNFREEZE_HPDMA1_CH7

__HAL_DBGMCU_FREEZE_HPDMA1_CH8

__HAL_DBGMCU_UNFREEZE_HPDMA1_CH8

__HAL_DBGMCU_FREEZE_HPDMA1_CH9

__HAL_DBGMCU_UNFREEZE_HPDMA1_CH9

__HAL_DBGMCU_FREEZE_HPDMA1_CH10

__HAL_DBGMCU_UNFREEZE_HPDMA1_CH10

__HAL_DBGMCU_FREEZE_HPDMA1_CH11

__HAL_DBGMCU_UNFREEZE_HPDMA1_CH11

__HAL_DBGMCU_FREEZE_HPDMA1_CH12

__HAL_DBGMCU_UNFREEZE_HPDMA1_CH12

__HAL_DBGMCU_FREEZE_HPDMA1_CH13

__HAL_DBGMCU_UNFREEZE_HPDMA1_CH13

__HAL_DBGMCU_FREEZE_HPDMA1_CH14

__HAL_DBGMCU_UNFREEZE_HPDMA1_CH14

__HAL_DBGMCU_FREEZE_HPDMA1_CH15

__HAL_DBGMCU_UNFREEZE_HPDMA1_CH15

__HAL_DBGMCU_FREEZE_NPU

__HAL_DBGMCU_UNFREEZE_NPU

HAL Exported Constants

__STM32N6xx_HAL_VERSION_MAIN

[31:24] main version

__STM32N6xx_HAL_VERSION_SUB1

[23:16] sub1 version

__STM32N6xx_HAL_VERSION_SUB2

[15:8] sub2 version

__STM32N6xx_HAL_VERSION_RC

[7:0] release candidate

__STM32N6xx_HAL_VERSION

HAL FDCAN Error Code

HAL_FDCAN_ERROR_NONE

No error

HAL_FDCAN_ERROR_TIMEOUT

Timeout error

HAL_FDCAN_ERROR_NOT_INITIALIZED

Peripheral not initialized

HAL_FDCAN_ERROR_NOT_READY

Peripheral not ready

HAL_FDCAN_ERROR_NOT_STARTED

Peripheral not started

HAL_FDCAN_ERROR_NOT_SUPPORTED

Mode not supported

HAL_FDCAN_ERROR_PARAM

Parameter error

HAL_FDCAN_ERROR_PENDING

Pending operation

HAL_FDCAN_ERROR_RAM_ACCESS

Message RAM Access Failure

HAL_FDCAN_ERROR_FIFO_EMPTY

Put element in full FIFO

HAL_FDCAN_ERROR_FIFO_FULL

Get element from empty FIFO

HAL_FDCAN_ERROR_LOG_OVERFLOW

Overflow of CAN Error Logging Counter

HAL_FDCAN_ERROR_RAM_WDG

Message RAM Watchdog event occurred

HAL_FDCAN_ERROR_PROTOCOL_ARBT

Protocol Error in Arbitration Phase (Nominal Bit Time is used)

HAL_FDCAN_ERROR_PROTOCOL_DATA

Protocol Error in Data Phase (Data Bit Time is used)

HAL_FDCAN_ERROR_RESERVED_AREA

Access to Reserved Address

HAL_FDCAN_ERROR_TT_GLOBAL_TIME

Global Time Error : Synchronization deviation exceeded limit

HAL_FDCAN_ERROR_TT_TX_UNDERFLOW

Tx Count Underflow : Less Tx trigger than expected in one matrix cycle

HAL_FDCAN_ERROR_TT_TX_OVERFLOW

Tx Count Overflow : More Tx trigger than expected in one matrix cycle

HAL_FDCAN_ERROR_TT_SCHEDULE1

Scheduling error 1

HAL_FDCAN_ERROR_TT_SCHEDULE2

Scheduling error 2

HAL_FDCAN_ERROR_TT_NO_INIT_REF

No system startup due to missing reference message

HAL_FDCAN_ERROR_TT_NO_REF

Missing reference message

HAL_FDCAN_ERROR_TT_APPL_WDG

Application watchdog not served in time

HAL_FDCAN_ERROR_TT_CONFIG

Error found in trigger list

HAL_FDCAN_ERROR_INVALID_CALLBACK

Invalid Callback error

I3C Notification ID definition

EVENT_ID_GETACCCR

I3C target complete controller-role hand-off (direct GETACCR CCC) event

EVENT_ID_IBIEND

I3C target IBI end process event

EVENT_ID_DAU

I3C target receive a dynamic address update (ENTDAA/RSTDAA/SETNEWDA CCC) event

EVENT_ID_GETx

I3C target receive any direct GETxxx CCC event

EVENT_ID_GETSTATUS

I3C target receive get status command (direct GETSTATUS CCC) event

EVENT_ID_SETMWL

I3C target receive maximum write length update (direct SETMWL CCC) event

EVENT_ID_SETMRL

I3C target receive maximum read length update(direct SETMRL CCC) event

EVENT_ID_RSTACT

I3C target detect reset pattern (broadcast or direct RSTACT CCC) event

EVENT_ID_ENTASx

I3C target receive activity state update (direct or broadcast ENTASx) event

EVENT_ID_ENEC_DISEC

I3C target receive a direct or broadcast ENEC/DISEC CCC event

EVENT_ID_DEFTGTS

I3C target receive a broadcast DEFTGTS CCC event

EVENT_ID_DEFGRPA

I3C target receive a group addressing (broadcast DEFGRPA CCC) event

EVENT_ID_WKP

I3C target wakeup event

EVENT_ID_IBI

I3C controller receive IBI event

EVENT_ID_CR

I3C controller controller-role request event

EVENT_ID_HJ

I3C controller hot-join event

HAL state definition

HAL_SMBUS_STATE_RESET

SMBUS not yet initialized or disabled

HAL_SMBUS_STATE_READY

SMBUS initialized and ready for use

HAL_SMBUS_STATE_BUSY

SMBUS internal process is ongoing

HAL_SMBUS_STATE_MASTER_BUSY_TX

Master Data Transmission process is ongoing

HAL_SMBUS_STATE_MASTER_BUSY_RX

Master Data Reception process is ongoing

HAL_SMBUS_STATE_SLAVE_BUSY_TX

Slave Data Transmission process is ongoing

HAL_SMBUS_STATE_SLAVE_BUSY_RX

Slave Data Reception process is ongoing

HAL_SMBUS_STATE_LISTEN

Address Listen Mode is ongoing

Boot Id

SYSCFG_BOOT_0

Boot 0 selection

SYSCFG_BOOT_1

Boot 1 selection

Cache biasing level adjust input selection

SYSCFG_CACHE_BIAS_VNOM

SYSCFG_CACHE_BIAS_VNOM_10_PERCENT

Compensation Cell Selection

SYSCFG_IO_VDDIO2_CELL

I/O Compensation cell for VDDIO2

SYSCFG_IO_VDDIO3_CELL

I/O Compensation cell for VDDIO3

SYSCFG_IO_VDDIO4_CELL

I/O Compensation cell for VDDIO4

SYSCFG_IO_VDDIO5_CELL

I/O Compensation cell for VDDIO5

Delay on feedback clock

SYSCFG_DELAY_FEEDBACK_NONE

No delay on the feedback clock

SYSCFG_DELAY_FEEDBACK_HALF_CYCLE

Half a cycle delay on the feedback clock

SYSCFG DTCM size

SYSCFG_DTCM_128K

DTCM memory size is 128K

SYSCFG_DTCM_256K

DTCM memory size is 256K

SYSCFG Exported Macros

__HAL_SYSCFG_FPU_INTERRUPT_ENABLE

Description:

- Floating Point Unit interrupt enable/disable macros.

Parameters:

- __INTERRUPT__: This parameter can be a value of

__HAL_SYSCFG_FPU_INTERRUPT_DISABLE

FPU Interrupts

SYSCFG_IT_FPU_IOC

Floating Point Unit Invalid operation Interrupt

SYSCFG_IT_FPU_DZC

Floating Point Unit Divide-by-zero Interrupt

SYSCFG_IT_FPU_UFC

Floating Point Unit Underflow Interrupt

SYSCFG_IT_FPU_OFC

Floating Point Unit Overflow Interrupt

SYSCFG_IT_FPU_IDC

Floating Point Unit Input denormal Interrupt

SYSCFG_IT_FPU_IXC

Floating Point Unit Inexact Interrupt

SYSCFG_IT_FPU_ALL

IO Compensation Code config

SYSCFG_IO_CELL_CODE

Code from the cell

SYSCFG_IO_REGISTER_CODE

Code from the values in the cell code register

SYSCFG ITCM size

SYSCFG_ITCM_64K

ITCM memory size is 64K

SYSCFG_ITCM_128K

ITCM memory size is 128K

SYSCFG_ITCM_256K

ITCM memory size is 256K

SYSCFG Lock items

SYSCFG_MPU_NSEC

Non-secure MPU lock (privileged secure or non-secure only)

SYSCFG_VTOR_NSEC

Non-secure VTOR lock (privileged secure or non-secure only)

SYSCFG_DCAIC

Disable access to the instruction cache direct cache access registers DCAICLR and DCAICRR

SYSCFG_SAU

SAU lock (privileged secure code only)

SYSCFG_MPU_SEC

Secure MPU lock (privileged secure code only)

SYSCFG_VTOR_AIRCR_SEC

VTOR_S and AIRCR lock (privileged secure code only)

SYSCFG_LOCK_ALL

All

SDMMC Id

SYSCFG_SDMMC1

SDMMC 1 selection

SYSCFG_SDMMC2

SDMMC 2 selection

SYSCFG_SDMMC_ALL

SDMMC 1 & 2 selection

biasing level selection

SYSCFG_TCM_BIAS_VNOM

SYSCFG_TCM_BIAS_VNOM_10_PERCENT

USB Id

SYSCFG_USB1

USB 1 selection

SYSCFG_USB2

USB 2 selection

SYSCFG_USB_ALL

USB 1 & 2 selection

WRITE access selection

SYSCFG_LOCK_WR_TCM

ITCM memory size is 64K

SYSCFG_LOCK_WR_ITGU

Disable writes to registers associated with the ITCM interface security/gating.

SYSCFG_LOCK_WR_DTGU

Disable writes to registers associated with the DTCM interface security/gating

SYSCFG_LOCK_WR_ALL

All writes accesses

XPU Id

SYSCFG_NPU_NIC

NPU_NIC clock gating disable selection

SYSCFG_CPU_NIC

CPU_NIC clock gating disable selection

SYSCFG_CPU_NOC

CPU_NOC clock gating disable selection

SYSCFG_CPU_ALL

7 HAL ADC Generic Driver

7.1 ADC Firmware driver registers structures

7.1.1 ADC_OversamplingTypeDef

ADC_OversamplingTypeDef is defined in the stm32n6xx_hal_adc.h

Data Fields

- ***uint32_t Ratio***
- ***uint32_t RightBitShift***
- ***uint32_t TriggeredMode***
- ***uint32_t OversamplingStopReset***

Field Documentation

- ***uint32_t ADC_OversamplingTypeDef::Ratio***
  Configures the oversampling ratio. This parameter can be a value from 1 to 1024
- ***uint32_t ADC_OversamplingTypeDef::RightBitShift***
  Configures the division coefficient for the Oversampler. This parameter can be a value of ***ADC_HAL_EC_OVS_SHIFT***
- ***uint32_t ADC_OversamplingTypeDef::TriggeredMode***
  Selects the regular triggered oversampling mode. This parameter can be a value of ***ADC_HAL_EC_OVS_DISCONT_MODE***
- ***uint32_t ADC_OversamplingTypeDef::OversamplingStopReset***
  Selects the regular oversampling mode. The oversampling is either temporary stopped or reset upon an injected sequence interruption. If oversampling is enabled on both regular and injected groups, this parameter is discarded and forced to setting "ADC_REGOVERSAMPLING_RESUMED_MODE" (the oversampling buffer is zeroed during injection sequence). This parameter can be a value of ***ADC_HAL_EC_OVS_SCOPE_REG***

7.1.2 ADC_InitTypeDef

ADC_InitTypeDef is defined in the stm32n6xx_hal_adc.h

Data Fields

- ***uint32_t Resolution***
- ***uint32_t GainCompensation***
- ***uint32_t ScanConvMode***
- ***uint32_t EOCSelection***
- ***FunctionalState LowPowerAutoWait***
- ***FunctionalState ContinuousConvMode***
- ***uint32_t NbrOfConversion***
- ***FunctionalState DiscontinuousConvMode***
- ***uint32_t NbrOfDiscConversion***
- ***uint32_t ExternalTrigConv***
- ***uint32_t ExternalTrigConvEdge***
- ***uint32_t SamplingMode***
- ***uint32_t ConversionDataManagement***
- ***uint32_t Overrun***
- ***uint32_t LeftBitShift***
- ***FunctionalState OversamplingMode***
- ***ADC_OversamplingTypeDef Oversampling***

Field Documentation

- ***uint32_t ADC_InitTypeDef::Resolution***
  Configure the ADC resolution. This parameter can be a value of ***ADC_HAL_EC_RESOLUTION***

- ***uint32_t ADC_InitTypeDef::GainCompensation***
  Specify the ADC gain compensation coefficient to be applied to ADC raw conversion data, based on following formula: DATA = DATA(raw) * (gain compensation coef) / 4096 12 bit format, unsigned: 2 bits exponents / 12 bits mantissa Gain step is 1/4096 = 0.000244 Gain range is 0.0000 to 3.999756 This parameter value can be:
  - value "0": Gain compensation will be disabled and coefficient set to 0
  - value in range [0x0001; 0x3FFF]: Gain compensation will enabled and coefficient set to specified value
- ***uint32_t ADC_InitTypeDef::ScanConvMode***
  Configure the sequencer of ADC groups regular and injected. This parameter can be associated to parameter 'DiscontinuousConvMode' to have main sequence subdivided in successive parts. If disabled: Conversion is performed in single mode (one channel converted, the one defined in rank 1). Parameters 'NbrOfConversion' and 'InjectedNbrOfConversion' are discarded (equivalent to set to 1). If enabled: Conversions are performed in sequence mode (multiple ranks defined by 'NbrOfConversion' or 'InjectedNbrOfConversion' and rank of each channel in sequencer). Scan direction is upward: from rank 1 to rank 'n'. This parameter can be a value of ***ADC_Scan_mode***
- ***uint32_t ADC_InitTypeDef::EOCSelection***
  Specify which EOC (End Of Conversion) flag is used for conversion by polling and interruption: end of unitary conversion or end of sequence conversions. This parameter can be a value of ***ADC_EOCSelection***.
- ***FunctionalState ADC_InitTypeDef::LowPowerAutoWait***
  Select the dynamic low power Auto Delay: new conversion start only when the previous conversion (for ADC group regular) or previous sequence (for ADC group injected) has been retrieved by user software, using function **HAL_ADC_GetValue()** or **HAL_ADCEx_InjectedGetValue()**. This feature automatically adapts the frequency of ADC conversions triggers to the speed of the system that reads the data. Moreover, this avoids risk of overrun for low frequency applications. This parameter can be set to ENABLE or DISABLE. Note: It is not recommended to use with interruption or DMA (**HAL_ADC_Start_IT()**, **HAL_ADC_Start_DMA()**) since these modes have to clear immediately the EOC flag (by CPU to free the IRQ pending event or by DMA). Auto wait will work but fort a very short time, discarding its intended benefit (except specific case of high load of CPU or DMA transfers which can justify usage of auto wait). Do use with polling: 1. Start conversion with **HAL_ADC_Start()**, 2. Later on, when ADC conversion data is needed: use **HAL_ADC_PollForConversion()** to ensure that conversion is completed and **HAL_ADC_GetValue()** to retrieve conversion result and trig another conversion start. (in case of usage of ADC group injected, use the equivalent functions HAL_ADCExInjected_Start(), **HAL_ADCEx_InjectedGetValue()**, ...).
- ***FunctionalState ADC_InitTypeDef::ContinuousConvMode***
  Specify whether the conversion is performed in single mode (one conversion) or continuous mode for ADC group regular, after the first ADC conversion start trigger occurred (software start or external trigger). This parameter can be set to ENABLE or DISABLE.
- ***uint32_t ADC_InitTypeDef::NbrOfConversion***
  Specify the number of ranks that will be converted within the regular group sequencer. To use the regular group sequencer and convert several ranks, parameter 'ScanConvMode' must be enabled. This parameter must be a number between Min_Data = 1 and Max_Data = 16. Note: This parameter must be modified when no conversion is on going on regular group (ADC disabled, or ADC enabled without continuous mode or external trigger that could launch a conversion).
- ***FunctionalState ADC_InitTypeDef::DiscontinuousConvMode***
  Specify whether the conversions sequence of ADC group regular is performed in Complete-sequence/ Discontinuous-sequence (main sequence subdivided in successive parts). Discontinuous mode is used only if sequencer is enabled (parameter 'ScanConvMode'). If sequencer is disabled, this parameter is discarded. Discontinuous mode can be enabled only if continuous mode is disabled. If continuous mode is enabled, this parameter setting is discarded. This parameter can be set to ENABLE or DISABLE.
- ***uint32_t ADC_InitTypeDef::NbrOfDiscConversion***
  Specifies the number of discontinuous conversions in which the main sequence of ADC group regular (parameter NbrOfConversion) will be subdivided. If parameter 'DiscontinuousConvMode' is disabled, this parameter is discarded. This parameter must be a number between Min_Data = 1 and Max_Data = 8.
- ***uint32_t ADC_InitTypeDef::ExternalTrigConv***
  Select the external event source used to trigger ADC group regular conversion start. If set to ADC_SOFTWARE_START, external triggers are disabled and software trigger is used instead. This parameter can be a value of ***ADC_regular_external_trigger_source***. Caution: external trigger source is common to all ADC instances.

- ***uint32_t ADC_InitTypeDef::ExternalTrigConvEdge***
  Select the external event edge used to trigger ADC group regular conversion start. If trigger source is set to ADC_SOFTWARE_START, this parameter is discarded. This parameter can be a value of ***ADC_regular_external_trigger_edge***
- ***uint32_t ADC_InitTypeDef::SamplingMode***
  Select the sampling mode to be used for ADC group regular conversion. This parameter can be a value of ***ADC_regular_sampling_mode***
- ***uint32_t ADC_InitTypeDef::ConversionDataManagement***
  Specifies whether the Data conversion data is managed: using the DMA (oneshot or circular), or stored in the DR register or transferred to MDF register. Note: In continuous mode, DMA must be configured in circular mode. Otherwise an overrun will be triggered when DMA buffer maximum pointer is reached. This parameter can be a value of ***ADC_ConversionDataManagement***. Note: This parameter must be modified when no conversion is on going on both regular and injected groups (ADC disabled, or ADC enabled without continuous mode or external trigger that could launch a conversion).
- ***uint32_t ADC_InitTypeDef::Overrun***
  Select the behavior in case of overrun: data overwritten or preserved (default). This parameter applies to ADC group regular only. This parameter can be a value of ***ADC_HAL_EC_REG_OVR_DATA_BEHAVIOR***. Note: In case of overrun set to data preserved and usage with programming model with interruption (HAL_Start_IT()): ADC IRQ handler has to clear end of conversion flags, this induces the release of the preserved data. If needed, this data can be saved in function **HAL_ADC_ConvCpltCallback()**, placed in user program code (called before end of conversion flags clear). Note: Error reporting with respect to the conversion mode:
  - Usage with ADC conversion by polling for event or interruption: Error is reported only if overrun is set to data preserved. If overrun is set to data overwritten, user can willingly not read all the converted data, this is not considered as an erroneous case.
  - Usage with ADC conversion by DMA: Error is reported whatever overrun setting (DMA is expected to process all data from data register).
- ***uint32_t ADC_InitTypeDef::LeftBitShift***
  Configures the left shifting applied to the final result with or without oversampling. This parameter can be a value of ***ADCEx_Left_Bit_Shift***
- ***FunctionalState ADC_InitTypeDef::OversamplingMode***
  Specify whether the oversampling feature is enabled or disabled. This parameter can be set to ENABLE or DISABLE. Note: This parameter can be modified only if there is no conversion is ongoing on ADC groups regular and injected
- ***ADC_OversamplingTypeDef ADC_InitTypeDef::Oversampling***
  Specify the Oversampling parameters. Caution: this setting overwrites the previous oversampling configuration if oversampling is already enabled.

7.1.3 ADC_ChannelConfTypeDef

ADC_ChannelConfTypeDef is defined in the stm32n6xx_hal_adc.h

Data Fields

- ***uint32_t Channel***
- ***uint32_t Rank***
- ***uint32_t SamplingTime***
- ***uint32_t SingleDiff***
- ***uint32_t OffsetNumber***
- ***uint32_t Offset***
- ***FunctionalState OffsetSignedSaturation***
- ***FunctionalState OffsetSaturation***
- ***uint32_t OffsetSign***

Field Documentation

- ***uint32_t ADC_ChannelConfTypeDef::Channel***
  Specify the channel to configure into ADC regular group. This parameter can be a value of ***ADC_HAL_EC_CHANNEL*** Note: Depending on devices and ADC instances, some channels may not be available on device package pins. Refer to device datasheet for channels availability.

- ***uint32_t ADC_ChannelConfTypeDef::Rank***
  Specify the rank in the regular group sequencer. This parameter can be a value of ***ADC_HAL_EC_REG_SEQ_RANKS*** Note: to disable a channel or change order of conversion sequencer, rank containing a previous channel setting can be overwritten by the new channel setting (or parameter number of conversions adjusted)
- ***uint32_t ADC_ChannelConfTypeDef::SamplingTime***
  Sampling time value to be set for the selected channel. Unit: ADC clock cycles Conversion time is the addition of sampling time and processing time (13.5 ADC clock cycles at ADC resolution 12 bits, 11.5 cycles at 10 bits, 8.5 cycles at 8 bits, 6.5 cycles at 6 bits). This parameter can be a value of ***ADC_HAL_EC_CHANNEL_SAMPLINGTIME*** Caution: This parameter applies to a channel that can be used into regular and/or injected group. It overwrites the last setting. Note: In case of usage of internal measurement channels (VrefInt/Vbat/TempSensor), sampling time constraints must be respected (sampling time can be adjusted in function of ADC clock frequency and sampling time setting). Refer to device datasheet for timings values.
- ***uint32_t ADC_ChannelConfTypeDef::SingleDiff***
  Select single-ended or differential input. In differential mode: Differential measurement is carried out between the selected channel (positive input) and another channel (negative input). Refer to reference manual for corresponding channel. Only channel of positive input has to be configured, the other channel is configured automatically. This parameter must be a value of ***ADC_HAL_EC_CHANNEL_SINGLE_DIFF_ENDING*** Caution: This parameter applies to a channel that can be used in a regular and/or injected group. It overwrites the last setting. Note: Refer to Reference Manual to ensure the selected channel is available in differential mode. Note: When configuring a channel 'i' in differential mode, the channel 'i+1' is not usable separately. Note: This parameter must be modified when ADC is disabled (before ADC start conversion or after ADC stop conversion). If ADC is enabled, this parameter setting is bypassed without error reporting (as it can be the expected behavior in case of another parameter update on the fly)
- ***uint32_t ADC_ChannelConfTypeDef::OffsetNumber***
  Select the offset number This parameter can be a value of ***ADC_HAL_EC_OFFSET_NB*** Caution: Only one offset is allowed per channel. This parameter overwrites the last setting.
- ***uint32_t ADC_ChannelConfTypeDef::Offset***
  Define the offset to be subtracted from the raw converted data. Offset value must be a positive number. Depending of ADC resolution selected (12, 10, 8 or 6 bits), this parameter must be a number between Min_Data = 0x000 and Max_Data = 0xFFF, 0xFFF, 0x3FF, 0xFF or 0x3F respectively. Note: This parameter must be modified when no conversion is on going on both regular and injected groups (ADC disabled, or ADC enabled without continuous mode or external trigger that could launch a conversion).
- ***FunctionalState ADC_ChannelConfTypeDef::OffsetSignedSaturation***
  Specify whether the Signed saturation feature is used or not. This parameter is applied only for 16-bit or 8-bit resolution. This parameter can be set to ENABLE or DISABLE.
- ***FunctionalState ADC_ChannelConfTypeDef::OffsetSaturation***
  Define if the offset should be saturated upon under or over flow. This parameter value can be ENABLE or DISABLE. Note:
  - This parameter must be modified when no conversion is on going on both regular and injected groups (ADC disabled, or ADC enabled without continuous mode or external trigger that could launch a conversion).
- ***uint32_t ADC_ChannelConfTypeDef::OffsetSign***
  Define if the offset should be subtracted (negative sign) or added (positive sign) from or to the raw converted data. This parameter can be a value of ***ADCEx_OffsetSign***. Note:
  - This parameter must be modified when no conversion is on going on both regular and injected groups (ADC disabled, or ADC enabled without continuous mode or external trigger that could launch a conversion).

7.1.4 ADC_AnalogWDGConfTypeDef

ADC_AnalogWDGConfTypeDef is defined in the stm32n6xx_hal_adc.h

Data Fields

- ***uint32_t WatchdogNumber***
- ***uint32_t WatchdogMode***
- ***uint32_t Channel***
- ***FunctionalState ITMode***

- ***uint32_t HighThreshold***
- ***uint32_t LowThreshold***
- ***uint32_t FilteringConfig***

Field Documentation

- ***uint32_t ADC_AnalogWDGConfTypeDef::WatchdogNumber***
  Select which ADC analog watchdog is monitoring the selected channel. For Analog Watchdog 1: Only 1 channel can be monitored (or overall group of channels by setting parameter 'WatchdogMode') For Analog Watchdog 2 and 3: Several channels can be monitored (by successive calls of '**HAL_ADC_AnalogWDGConfig()**' for each channel) This parameter can be a value of ***ADC_HAL_EC_AWD_NUMBER***.
- ***uint32_t ADC_AnalogWDGConfTypeDef::WatchdogMode***
  Configure the ADC analog watchdog mode: single/all/none channels. For Analog Watchdog 1: Configure the ADC analog watchdog mode: single channel or all channels, ADC groups regular and-or injected. For Analog Watchdog 2 and 3: Several channels can be monitored by applying successively the AWD init structure. Channels on ADC group regular and injected are not differentiated: Set value 'ADC_ANALOGWATCHDOG_SINGLE_xxx' to monitor 1 channel, value 'ADC_ANALOGWATCHDOG_ALL_xxx' to monitor all channels, 'ADC_ANALOGWATCHDOG_NONE' to monitor no channel. This parameter can be a value of ***ADC_analog_watchdog_mode***.
- ***uint32_t ADC_AnalogWDGConfTypeDef::Channel***
  Select which ADC channel to monitor by analog watchdog. For Analog Watchdog 1: this parameter has an effect only if parameter 'WatchdogMode' is configured on single channel (only 1 channel can be monitored). For Analog Watchdog 2 and 3: Several channels can be monitored. To use this feature, call successively the function **HAL_ADC_AnalogWDGConfig()** for each channel to be added (or removed with value 'ADC_ANALOGWATCHDOG_NONE'). This parameter can be a value of ***ADC_HAL_EC_CHANNEL***.
- ***FunctionalState ADC_AnalogWDGConfTypeDef::ITMode***
  Specify whether the analog watchdog is configured in interrupt or polling mode. This parameter can be set to ENABLE or DISABLE
- ***uint32_t ADC_AnalogWDGConfTypeDef::HighThreshold***
  Configure the ADC analog watchdog High threshold value. Depending of ADC resolution selected (12, 10, 8 or 6 bits), this parameter must be a number between Min_Data = 0x000 and Max_Data = 0xFFF, 0x3FF, 0xFF or 0x3F respectively.
- ***uint32_t ADC_AnalogWDGConfTypeDef::LowThreshold***
  Configures the ADC analog watchdog Low threshold value. Depending of ADC resolution selected (12, 10, 8 or 6 bits), this parameter must be a number between Min_Data = 0x000 and Max_Data = 0xFFF, 0x3FF, 0xFF or 0x3F respectively.
- ***uint32_t ADC_AnalogWDGConfTypeDef::FilteringConfig***
  Specify whether filtering should be use and the number of samples to consider. Before setting flag or raising interrupt, analog watchdog can wait to have several consecutive out-of-window samples. This parameter allows to configure this number. This parameter only applies to Analog watchdog 1. For others, use value ADC_AWD_FILTERING_NONE. This parameter can be a value of ***ADC_analog_watchdog_filtering_config***.

7.1.5 __ADC_HandleTypeDef

__ADC_HandleTypeDef is defined in the stm32n6xx_hal_adc.h

Data Fields

- ***ADC_TypeDef * Instance***
- ***ADC_InitTypeDef Init***
- ***DMA_HandleTypeDef * DMA_Handle***
- ***HAL_LockTypeDef Lock***
- ***__IO uint32_t State***
- ***__IO uint32_t ErrorCode***
- ***void(* ConvCpltCallback***
- ***void(* ConvHalfCpltCallback***
- ***void(* LevelOutOfWindowCallback***
- ***void(* ErrorCallback***
- ***void(* InjectedConvCpltCallback***

- ***void(* LevelOutOfWindow2Callback***
- ***void(* LevelOutOfWindow3Callback***
- ***void(* EndOfSamplingCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***ADC_TypeDef* __ADC_HandleTypeDef::Instance***
  Register base address
- ***ADC_InitTypeDef __ADC_HandleTypeDef::Init***
  ADC initialization parameters and regular conversions setting
- ***DMA_HandleTypeDef* __ADC_HandleTypeDef::DMA_Handle***
  Pointer DMA Handler
- ***HAL_LockTypeDef __ADC_HandleTypeDef::Lock***
  ADC locking object
- ***__IO uint32_t __ADC_HandleTypeDef::State***
  ADC communication state (bitmap of ADC states)
- ***__IO uint32_t __ADC_HandleTypeDef::ErrorCode***
  ADC Error code
- ***void(* __ADC_HandleTypeDef::ConvCpltCallback)(struct __ADC_HandleTypeDef *hadc)***
  ADC conversion complete callback
- ***void(* __ADC_HandleTypeDef::ConvHalfCpltCallback)(struct __ADC_HandleTypeDef *hadc)***
  ADC conversion DMA half-transfer callback
- ***void(* __ADC_HandleTypeDef::LevelOutOfWindowCallback)(struct __ADC_HandleTypeDef *hadc)***
  ADC analog watchdog 1 callback
- ***void(* __ADC_HandleTypeDef::ErrorCallback)(struct __ADC_HandleTypeDef *hadc)***
  ADC error callback
- ***void(* __ADC_HandleTypeDef::InjectedConvCpltCallback)(struct __ADC_HandleTypeDef *hadc)***
  ADC group injected conversion complete callback
- ***void(* __ADC_HandleTypeDef::LevelOutOfWindow2Callback)(struct __ADC_HandleTypeDef *hadc)***
  ADC analog watchdog 2 callback
- ***void(* __ADC_HandleTypeDef::LevelOutOfWindow3Callback)(struct __ADC_HandleTypeDef *hadc)***
  ADC analog watchdog 3 callback
- ***void(* __ADC_HandleTypeDef::EndOfSamplingCallback)(struct __ADC_HandleTypeDef *hadc)***
  ADC end of sampling callback
- ***void(* __ADC_HandleTypeDef::MspInitCallback)(struct __ADC_HandleTypeDef *hadc)***
  ADC Msp Init callback
- ***void(* __ADC_HandleTypeDef::MspDeInitCallback)(struct __ADC_HandleTypeDef *hadc)***
  ADC Msp DeInit callback

7.2 ADC Firmware driver API description

The following section lists the various functions of the ADC library.

7.2.1 ADC peripheral features

- 12-bit, 10-bit, 8-bit or 6-bit configurable resolution.
- Interrupt generation at the end of regular conversion and in case of analog watchdog or overrun events.
- Single and continuous conversion modes.
- Scan mode for conversion of several channels sequentially.
- Data alignment with in-built data coherency.
- Programmable sampling time (channel wise)
- External trigger (timer or EXTI) with configurable polarity
- DMA request generation for transfer of conversions data of regular group.
- Configurable delay between conversions in Dual interleaved mode.
- ADC channels selectable single/differential input.

- ADC offset shared on 4 offset instances.
- ADC gain compensation
- ADC calibration
- ADC conversion of regular group.
- ADC supply requirements: 1.62 V to 3.6 V.
- ADC input range: from Vref- (connected to Vssa) to Vref+ (connected to Vdda or to an external voltage reference).

7.2.2 How to use this driver

Configuration of top level parameters related to ADC

1. Enable the ADC interface
   - As prerequisite, ADC clock must be configured at RCC top level.
   - Two clock settings are mandatory:
     - ADC clock (core clock, also possibly conversion clock).
     - ADC clock (conversions clock). Two possible clock sources: synchronous clock derived from AHB clock or asynchronous clock derived from system clock, PLL2 or PLL3.
     - Example: Into HAL_ADC_MspInit() (recommended code location) or with other device clock parameters configuration:
     - __HAL_RCC_ADC_CLK_ENABLE(); (mandatory) RCC_ADCCLKSOURCE_PLL2 enable: (optional: if asynchronous clock selected)
     - RCC_PeriphClkInitTypeDef RCC_PeriphClkInit;
     - PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_ADC;
     - PeriphClkInit.AdcClockSelection = RCC_ADCCLKSOURCE_PLL2;
     - HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit);
   - ADC clock source and clock prescaler are configured at ADC level with parameter "ClockPrescaler" using function HAL_ADC_Init().
2. ADC pins configuration
   - Enable the clock for the ADC GPIOs using macro __HAL_RCC_GPIOx_CLK_ENABLE()
   - Configure these ADC pins in analog mode using function HAL_GPIO_Init()
3. Optionally, in case of usage of ADC with interruptions:
   - Configure the NVIC for ADC using function HAL_NVIC_EnableIRQ(ADCx_IRQn)
   - Insert the ADC interruption handler function HAL_ADC_IRQHandler() into the function of corresponding ADC interruption vector ADCx_IRQHandler().
4. Optionally, in case of usage of DMA:
   - Configure the DMA (DMA channel, mode normal or circular, ...) using function HAL_DMA_Init().
   - Configure the NVIC for DMA using function HAL_NVIC_EnableIRQ(DMAx_Channelx_IRQn)
   - Insert the ADC interruption handler function HAL_ADC_IRQHandler() into the function of corresponding DMA interruption vector DMAx_Channelx_IRQHandler().

Configuration of ADC, group regular, channels parameters

1. Configure the ADC parameters (resolution, data alignment, ...) and regular group parameters (conversion trigger, sequencer, ...) using function HAL_ADC_Init().
2. Configure the channels for regular group parameters (channel number, channel rank into sequencer, ..., into regular group) using function HAL_ADC_ConfigChannel().
3. Optionally, configure the analog watchdog parameters (channels monitored, thresholds, ...) using function HAL_ADC_AnalogWDGConfig().

Execution of ADC conversions

1. Optionally, perform an automatic ADC calibration to improve the conversion accuracy using function HAL_ADCEx_Calibration_Start().

2. ADC driver can be used among three modes: polling, interruption, transfer by DMA.
   - ADC conversion by polling:
     - Activate the ADC peripheral and start conversions using function HAL_ADC_Start()
     - Wait for ADC conversion completion using function HAL_ADC_PollForConversion()
     - Retrieve conversion results using function HAL_ADC_GetValue()
     - Stop conversion and disable the ADC peripheral using function HAL_ADC_Stop()
   - ADC conversion by interruption:
     - Activate the ADC peripheral and start conversions using function HAL_ADC_Start_IT()
     - Wait for ADC conversion completion by call of function HAL_ADC_ConvCpltCallback() (this function must be implemented in user program)
     - Retrieve conversion results using function HAL_ADC_GetValue()
     - Stop conversion and disable the ADC peripheral using function HAL_ADC_Stop_IT()
   - ADC conversion with transfer by DMA:
     - Activate the ADC peripheral and start conversions using function HAL_ADC_Start_DMA()
     - Wait for ADC conversion completion by call of function HAL_ADC_ConvCpltCallback() or HAL_ADC_ConvHalfCpltCallback() (these functions must be implemented in user program)
     - Conversion results are automatically transferred by DMA into destination variable address.
     - Stop conversion and disable the ADC peripheral using function HAL_ADC_Stop_DMA()

Note: Callback functions must be implemented in user program:

- *HAL_ADC_ErrorCallback()*
- *HAL_ADC_LevelOutOfWindowCallback() (callback of analog watchdog)*
- *HAL_ADC_ConvCpltCallback()*
- *HAL_ADC_ConvHalfCpltCallback*

Deinitialization of ADC

1. Disable the ADC interface
   - ADC clock can be hard reset and disabled at RCC top level.
   - Hard reset of ADC peripherals using macro __HAL_RCC_ADCx_FORCE_RESET(), __HAL_RCC_ADCx_RELEASE_RESET().
   - ADC clock disable using the equivalent macro/functions as configuration step.
     - Example: Into HAL_ADC_MspDeInit() (recommended code location) or with other device clock parameters configuration:
     - __HAL_RCC_ADC_CLK_DISABLE(); (if not used anymore) RCC_ADCCLKSOURCE_CLKP restore: (optional)
     - RCC_PeriphClkInitTypeDef RCC_PeriphClkInit;
     - PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_ADC;
     - PeriphClkInit.AdcClockSelection = RCC_ADCCLKSOURCE_CLKP;
     - HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit);
2. ADC pins configuration
   - Disable the clock for the ADC GPIOs using macro __HAL_RCC_GPIOx_CLK_DISABLE()
3. Optionally, in case of usage of ADC with interruptions:
   - Disable the NVIC for ADC using function HAL_NVIC_EnableIRQ(ADCx_IRQn)
4. Optionally, in case of usage of DMA:
   - Deinitialize the DMA using function HAL_DMA_Init().
   - Disable the NVIC for DMA using function HAL_NVIC_EnableIRQ(DMAx_Channelx_IRQn)

Callback registration

The compilation flag USE_HAL_ADC_REGISTER_CALLBACKS, when set to 1, allows the user to configure dynamically the driver callbacks. Use Functions HAL_ADC_RegisterCallback() to register an interrupt callback.

Function HAL_ADC_RegisterCallback() allows to register following callbacks:

- ConvCpltCallback : ADC conversion complete callback
- ConvHalfCpltCallback : ADC conversion DMA half-transfer callback
- LevelOutOfWindowCallback : ADC analog watchdog 1 callback
- ErrorCallback : ADC error callback
- InjectedConvCpltCallback : ADC group injected conversion complete callback
- LevelOutOfWindow2Callback : ADC analog watchdog 2 callback
- LevelOutOfWindow3Callback : ADC analog watchdog 3 callback
- EndOfSamplingCallback : ADC end of sampling callback
- MspInitCallback : ADC Msp Init callback
- MspDeInitCallback : ADC Msp DeInit callback This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

Use function HAL_ADC_UnRegisterCallback to reset a callback to the default weak function.

HAL_ADC_UnRegisterCallback takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks:

- ConvCpltCallback : ADC conversion complete callback
- ConvHalfCpltCallback : ADC conversion DMA half-transfer callback
- LevelOutOfWindowCallback : ADC analog watchdog 1 callback
- ErrorCallback : ADC error callback
- InjectedConvCpltCallback : ADC group injected conversion complete callback
- LevelOutOfWindow2Callback : ADC analog watchdog 2 callback
- LevelOutOfWindow3Callback : ADC analog watchdog 3 callback
- EndOfSamplingCallback : ADC end of sampling callback
- MspInitCallback : ADC Msp Init callback
- MspDeInitCallback : ADC Msp DeInit callback

By default, after the HAL_ADC_Init() and when the state is HAL_ADC_STATE_RESET all callbacks are set to the corresponding weak functions: examples HAL_ADC_ConvCpltCallback(), HAL_ADC_ErrorCallback(). Exception done for MspInit and MspDeInit functions that are reset to the legacy weak functions in the HAL_ADC_Init()/ HAL_ADC_DeInit() only when these callbacks are null (not registered beforehand).

If MspInit or MspDeInit are not null, the HAL_ADC_Init()/ HAL_ADC_DeInit() keep and use the user MspInit/ MspDeInit callbacks (registered beforehand) whatever the state.

Callbacks can be registered/unregistered in HAL_ADC_STATE_READY state only. Exception done MspInit/ MspDeInit functions that can be registered/unregistered in HAL_ADC_STATE_READY or HAL_ADC_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/ DeInit.

Then, the user first registers the MspInit/MspDeInit user callbacks using HAL_ADC_RegisterCallback() before calling HAL_ADC_DeInit() or HAL_ADC_Init() function.

When the compilation flag USE_HAL_ADC_REGISTER_CALLBACKS is set to 0 or not defined, the callback registration feature is not available and all callbacks are set to the corresponding weak functions.

7.2.3 Detailed description of functions

HAL_ADC_Init

Function name

HAL_StatusTypeDef HAL_ADC_Init (ADC_HandleTypeDef * hadc)

Function description

HAL_ADC_DeInit

Function name

HAL_StatusTypeDef HAL_ADC_DeInit (ADC_HandleTypeDef * hadc)

Function description

HAL_ADC_MspInit

Function name

void HAL_ADC_MspInit (ADC_HandleTypeDef * hadc)

Function description

HAL_ADC_MspDeInit

Function name

void HAL_ADC_MspDeInit (ADC_HandleTypeDef * hadc)

Function description

HAL_ADC_RegisterCallback

Function name

HAL_StatusTypeDef HAL_ADC_RegisterCallback (ADC_HandleTypeDef * hadc, HAL_ADC_CallbackIDTypeDef CallbackID, pADC_CallbackTypeDef pCallback)

Function description

HAL_ADC_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_ADC_UnRegisterCallback (ADC_HandleTypeDef * hadc, HAL_ADC_CallbackIDTypeDef CallbackID)

Function description

HAL_ADC_Start

Function name

HAL_StatusTypeDef HAL_ADC_Start (ADC_HandleTypeDef * hadc)

Function description

HAL_ADC_Stop

Function name

HAL_StatusTypeDef HAL_ADC_Stop (ADC_HandleTypeDef * hadc)

Function description

HAL_ADC_PollForConversion

Function name

HAL_StatusTypeDef HAL_ADC_PollForConversion (ADC_HandleTypeDef * hadc, uint32_t Timeout)

Function description

HAL_ADC_PollForEvent

Function name

HAL_StatusTypeDef HAL_ADC_PollForEvent (ADC_HandleTypeDef * hadc, uint32_t EventType, uint32_t Timeout)

Function description

HAL_ADC_Start_IT

Function name

HAL_StatusTypeDef HAL_ADC_Start_IT (ADC_HandleTypeDef * hadc)

Function description

HAL_ADC_Stop_IT

Function name

HAL_StatusTypeDef HAL_ADC_Stop_IT (ADC_HandleTypeDef * hadc)

Function description

HAL_ADC_Start_DMA

Function name

HAL_StatusTypeDef HAL_ADC_Start_DMA (ADC_HandleTypeDef * hadc, const uint32_t * pData, uint32_t Length)

Function description

HAL_ADC_Stop_DMA

Function name

HAL_StatusTypeDef HAL_ADC_Stop_DMA (ADC_HandleTypeDef * hadc)

Function description

HAL_ADC_GetValue

Function name

uint32_t HAL_ADC_GetValue (const ADC_HandleTypeDef * hadc)

Function description

HAL_ADC_GetSignedValue

Function name

int32_t HAL_ADC_GetSignedValue (const ADC_HandleTypeDef * hadc)

Function description

HAL_ADC_StartSampling

Function name

HAL_StatusTypeDef HAL_ADC_StartSampling (ADC_HandleTypeDef * hadc)

Function description

HAL_ADC_StopSampling

Function name

HAL_StatusTypeDef HAL_ADC_StopSampling (ADC_HandleTypeDef * hadc)

Function description

HAL_ADC_IRQHandler

Function name

void HAL_ADC_IRQHandler (ADC_HandleTypeDef * hadc)

Function description

HAL_ADC_ConvCpltCallback

Function name

void HAL_ADC_ConvCpltCallback (ADC_HandleTypeDef * hadc)

Function description

HAL_ADC_ConvHalfCpltCallback

Function name

void HAL_ADC_ConvHalfCpltCallback (ADC_HandleTypeDef * hadc)

Function description

HAL_ADC_LevelOutOfWindowCallback

Function name

void HAL_ADC_LevelOutOfWindowCallback (ADC_HandleTypeDef * hadc)

Function description

HAL_ADC_ErrorCallback

Function name

void HAL_ADC_ErrorCallback (ADC_HandleTypeDef * hadc)

Function description

HAL_ADC_ConfigChannel

Function name

HAL_StatusTypeDef HAL_ADC_ConfigChannel (ADC_HandleTypeDef * hadc, const ADC_ChannelConfTypeDef * pConfig)

Function description

HAL_ADC_AnalogWDGConfig

Function name

HAL_StatusTypeDef HAL_ADC_AnalogWDGConfig (ADC_HandleTypeDef * hadc, const ADC_AnalogWDGConfTypeDef * pAnalogWDGConfig)

Function description

HAL_ADC_GetState

Function name

uint32_t HAL_ADC_GetState (const ADC_HandleTypeDef * hadc)

Function description

HAL_ADC_GetError

Function name

uint32_t HAL_ADC_GetError (const ADC_HandleTypeDef * hadc)

Function description

ADC_ConversionStop

Function name

HAL_StatusTypeDef ADC_ConversionStop (ADC_HandleTypeDef * hadc, uint32_t ConversionGroup)

Function description

ADC_Enable

Function name

HAL_StatusTypeDef ADC_Enable (ADC_HandleTypeDef * hadc)

Function description

ADC_Disable

Function name

HAL_StatusTypeDef ADC_Disable (ADC_HandleTypeDef * hadc)

Function description

ADC_DMAConvCplt

Function name

void ADC_DMAConvCplt (DMA_HandleTypeDef * hdma)

Function description

ADC_DMAHalfConvCplt

Function name

void ADC_DMAHalfConvCplt (DMA_HandleTypeDef * hdma)

Function description

ADC_DMAError

Function name

void ADC_DMAError (DMA_HandleTypeDef * hdma)

Function description

ADC_ConfigureBoostMode

Function name

void ADC_ConfigureBoostMode (ADC_HandleTypeDef * hadc)

Function description

7.3 ADC Firmware driver defines

The following section lists the various define and macros of the module.

7.3.1 ADC

ADC

ADC Analog Watchdog filtering configuration

ADC_AWD_FILTERING_NONE

ADC analog watchdog no filtering, one out-of-window sample is needed to raise flag or interrupt

ADC_AWD_FILTERING_2SAMPLES

ADC analog watchdog 2 consecutives out-of-window samples are needed to raise flag or interrupt

ADC_AWD_FILTERING_3SAMPLES

ADC analog watchdog 3 consecutives out-of-window samples are needed to raise flag or interrupt

ADC_AWD_FILTERING_4SAMPLES

ADC analog watchdog 4 consecutives out-of-window samples are needed to raise flag or interrupt

ADC_AWD_FILTERING_5SAMPLES

ADC analog watchdog 5 consecutives out-of-window samples are needed to raise flag or interrupt

ADC_AWD_FILTERING_6SAMPLES

ADC analog watchdog 6 consecutives out-of-window samples are needed to raise flag or interrupt

ADC_AWD_FILTERING_7SAMPLES

ADC analog watchdog 7 consecutives out-of-window samples are needed to raise flag or interrupt

ADC_AWD_FILTERING_8SAMPLES

ADC analog watchdog 8 consecutives out-of-window samples are needed to raise flag or interrupt

ADC analog watchdog (AWD) mode

ADC_ANALOGWATCHDOG_NONE

ADC AWD not selected

ADC_ANALOGWATCHDOG_SINGLE_REG

ADC AWD applied to a regular group single channel

ADC_ANALOGWATCHDOG_SINGLE_INJEC

ADC AWD applied to an injected group single channel

ADC_ANALOGWATCHDOG_SINGLE_REGINJEC

ADC AWD applied to a regular and injected groups single channel

ADC_ANALOGWATCHDOG_ALL_REG

ADC AWD applied to regular group all channels

ADC_ANALOGWATCHDOG_ALL_INJEC

ADC AWD applied to injected group all channels

ADC_ANALOGWATCHDOG_ALL_REGINJEC

ADC AWD applied to regular and injected groups all channels

ADCx CFGR fields

ADC_CFGR1_FIELDS

ADCx CFGR sub fields

ADC_CFGR1_FIELDS_2

ADC Conversion Data Management

ADC_CONVERSIONDATA_DR

ADC conversions data are available in ADC data register only

ADC_CONVERSIONDATA_DMA_ONESHOT

ADC conversion data are transferred by DMA, in limited mode (one shot mode): DMA transfer requests are stopped when number of DMA data transfers (number of ADC conversions) is reached. This ADC mode is intended to be used with DMA mode non-circular.

ADC_CONVERSIONDATA_DMA_CIRCULAR

ADC conversion data are transferred by DMA, in unlimited mode: DMA transfer requests are unlimited, whatever number of DMA data transferred (number of ADC conversions). This ADC mode is intended to be used with DMA mode circular.

ADC_CONVERSIONDATA_MDF

ADC conversion data are transferred to MDF

ADC sequencer end of unitary conversion or sequence conversions

ADC_EOC_SINGLE_CONV

End of unitary conversion flag

ADC_EOC_SEQ_CONV

End of sequence conversions flag

ADC Error Code

HAL_ADC_ERROR_NONE

No error

HAL_ADC_ERROR_INTERNAL

ADC peripheral internal error (problem of clocking, enable/disable, erroneous state, ...)

HAL_ADC_ERROR_OVR

Overrun error

HAL_ADC_ERROR_DMA

DMA transfer error

HAL_ADC_ERROR_INVALID_CALLBACK

Invalid Callback error

ADC Event type

ADC_EOSMP_EVENT

Notes:

- Analog watchdog 1 is available on all stm32 series Analog watchdog 2 and 3 are not available on all series ADC End of Sampling event

ADC_AWD1_EVENT

ADC Analog watchdog 1 event (main analog watchdog)

ADC_AWD2_EVENT

ADC Analog watchdog 2 event (additional analog watchdog)

ADC_AWD3_EVENT

ADC Analog watchdog 3 event (additional analog watchdog)

ADC_OVR_EVENT

ADC overrun event

ADC Exported Constants

ADC_AWD_EVENT

ADC Analog watchdog 1 event: Naming for compatibility with other STM32 devices having only one analog watchdog

ADC flags definition

ADC_FLAG_RDY

ADC Ready flag

ADC_FLAG_EOSMP

ADC End of Sampling flag

ADC_FLAG_EOC

ADC End of Regular Conversion flag

ADC_FLAG_EOS

ADC End of Regular sequence of Conversions flag

ADC_FLAG_OVR

ADC overrun flag

ADC_FLAG_JEOC

ADC End of Injected Conversion flag

ADC_FLAG_JEOS

ADC End of Injected sequence of Conversions flag

ADC_FLAG_AWD1

ADC Analog watchdog 1 flag (main analog watchdog)

ADC_FLAG_AWD2

ADC Analog watchdog 2 flag (additional analog watchdog)

ADC_FLAG_AWD3

ADC Analog watchdog 3 flag (additional analog watchdog)

Analog watchdog - ADC analog watchdog (AWD) number

ADC_ANALOGWATCHDOG_1

ADC analog watchdog number 1

ADC_ANALOGWATCHDOG_2

ADC analog watchdog number 2

ADC_ANALOGWATCHDOG_3

ADC analog watchdog number 3

ADC instance - Channel number

ADC_CHANNEL_0

ADC external channel (connected to GPIO pin) ADCx_IN0

ADC_CHANNEL_1

ADC external channel (connected to GPIO pin) ADCx_IN1

ADC_CHANNEL_2

ADC external channel (connected to GPIO pin) ADCx_IN2

ADC_CHANNEL_3

ADC external channel (connected to GPIO pin) ADCx_IN3

ADC_CHANNEL_4

ADC external channel (connected to GPIO pin) ADCx_IN4

ADC_CHANNEL_5

ADC external channel (connected to GPIO pin) ADCx_IN5

ADC_CHANNEL_6

ADC external channel (connected to GPIO pin) ADCx_IN6

ADC_CHANNEL_7

ADC external channel (connected to GPIO pin) ADCx_IN7

ADC_CHANNEL_8

ADC external channel (connected to GPIO pin) ADCx_IN8

ADC_CHANNEL_9

ADC external channel (connected to GPIO pin) ADCx_IN9

ADC_CHANNEL_10

ADC external channel (connected to GPIO pin) ADCx_IN10

ADC_CHANNEL_11

ADC external channel (connected to GPIO pin) ADCx_IN11

ADC_CHANNEL_12

ADC external channel (connected to GPIO pin) ADCx_IN12

ADC_CHANNEL_13

ADC external channel (connected to GPIO pin) ADCx_IN13

ADC_CHANNEL_14

ADC external channel (connected to GPIO pin) ADCx_IN14

ADC_CHANNEL_15

ADC external channel (connected to GPIO pin) ADCx_IN15

ADC_CHANNEL_16

ADC external channel (connected to GPIO pin) ADCx_IN16

ADC_CHANNEL_17

ADC external channel (connected to GPIO pin) ADCx_IN17

ADC_CHANNEL_18

ADC external channel (connected to GPIO pin) ADCx_IN18

ADC_CHANNEL_19

ADC external channel (connected to GPIO pin) ADCx_IN19

ADC_CHANNEL_VREFINT

ADC internal channel connected to VrefInt: Internal voltage reference. On this STM32 series, ADC channel available only on ADC instance: ADC1.

ADC_CHANNEL_VBAT

ADC internal channel connected to Vbat/4: Vbat voltage through a divider ladder of factor 1/4 to have Vbat always below Vdda. On this STM32 series, ADC channel available only on ADC instance: ADC2.

ADC_CHANNEL_VDDCORE

ADC internal channel connected to VddCore. On this STM32 series, ADC channel available only on ADC instance: ADC2.

Channel - Sampling time

ADC_SAMPLETIME_1CYCLE_5

Sampling time 1.5 ADC clock cycle

ADC_SAMPLETIME_2CYCLES_5

Sampling time 2.5 ADC clock cycles

ADC_SAMPLETIME_6CYCLES_5

Sampling time 6.5 ADC clock cycles

ADC_SAMPLETIME_11CYCLES_5

Sampling time 11.5 ADC clock cycles

ADC_SAMPLETIME_23CYCLES_5

Sampling time 23.5 ADC clock cycles

ADC_SAMPLETIME_46CYCLES_5

Sampling time 46.5 ADC clock cycles

ADC_SAMPLETIME_246CYCLES_5

Sampling time 246.5 ADC clock cycles

ADC_SAMPLETIME_1499CYCLES_5

Sampling time 1499.5 ADC clock cycles

Channel - Single or differential ending

ADC_SINGLE_ENDED

ADC channel ending set to single ended

ADC_DIFFERENTIAL_ENDED

ADC channel ending set to differential

ADC instance - Groups

ADC_REGULAR_GROUP

ADC group regular (available on all STM32 devices)

ADC_INJECTED_GROUP

ADC group injected (not available on all STM32 devices)

ADC_REGULAR_INJECTED_GROUP

ADC both groups regular and injected

Multimode - Mode

ADC_MODE_INDEPENDENT

ADC dual mode disabled (ADC independent mode)

ADC_DUALMODE_REGSIMULT

ADC dual mode enabled: group regular simultaneous

ADC_DUALMODE_INTERL

ADC dual mode enabled: Combined group regular interleaved

ADC_DUALMODE_INJECSIMULT

ADC dual mode enabled: group injected simultaneous

ADC_DUALMODE_ALTERTRIG

ADC dual mode enabled: group injected alternate trigger. Works only with external triggers (not internal SW start)

ADC_DUALMODE_REGSIMULT_INJECSIMULT

ADC dual mode enabled: Combined group regular simultaneous + group injected simultaneous

ADC_DUALMODE_REGSIMULT_ALTERTRIG

ADC dual mode enabled: Combined group regular simultaneous + group injected alternate trigger

ADC_DUALMODE_REGINTERL_INJECSIMULT

ADC dual mode enabled: Combined group regular interleaved + group injected simultaneous

Multimode - Delay between two sampling phases

ADC_TWOSAMPLINGDELAY_1CYCLE

ADC multimode delay between two sampling phases: 1 ADC clock cycle

ADC_TWOSAMPLINGDELAY_2CYCLES

ADC multimode delay between two sampling phases: 2 ADC clock cycles

ADC_TWOSAMPLINGDELAY_3CYCLES

ADC multimode delay between two sampling phases: 3 ADC clock cycles

ADC_TWOSAMPLINGDELAY_4CYCLES

ADC multimode delay between two sampling phases: 4 ADC clock cycles

ADC_TWOSAMPLINGDELAY_5CYCLES

ADC multimode delay between two sampling phases: 5 ADC clock cycles

ADC_TWOSAMPLINGDELAY_6CYCLES

ADC multimode delay between two sampling phases: 6 ADC clock cycles

ADC_TWOSAMPLINGDELAY_7CYCLES

ADC multimode delay between two sampling phases: 7 ADC clock cycles

ADC_TWOSAMPLINGDELAY_8CYCLES

ADC multimode delay between two sampling phases: 8 ADC clock cycles

ADC_TWOSAMPLINGDELAY_9CYCLES

ADC multimode delay between two sampling phases: 9 ADC clock cycles

ADC_TWOSAMPLINGDELAY_10CYCLES

ADC multimode delay between two sampling phases: 10 ADC clock cycles

ADC_TWOSAMPLINGDELAY_11CYCLES

ADC multimode delay between two sampling phases: 11 ADC clock cycles

ADC_TWOSAMPLINGDELAY_12CYCLES

ADC multimode delay between two sampling phases: 12 ADC clock cycles

ADC_TWOSAMPLINGDELAY_13CYCLES

ADC multimode delay between two sampling phases: 13 ADC clock cycles

ADC instance - Offset number

ADC_OFFSET_NONE

ADC offset disabled: no offset correction for the selected ADC channel

ADC_OFFSET_1

ADC offset number 1: ADC channel and offset level to which the offset programmed will be applied (independently of channel mapped on ADC group regular or group injected)

ADC_OFFSET_2

ADC offset number 2: ADC channel and offset level to which the offset programmed will be applied (independently of channel mapped on ADC group regular or group injected)

ADC_OFFSET_3

ADC offset number 3: ADC channel and offset level to which the offset programmed will be applied (independently of channel mapped on ADC group regular or group injected)

ADC_OFFSET_4

ADC offset number 4: ADC channel and offset level to which the offset programmed will be applied (independently of channel mapped on ADC group regular or group injected)

Oversampling - Discontinuous mode

ADC_TRIGGEREDMODE_SINGLE_TRIGGER

ADC oversampling discontinuous mode: continuous mode (all conversions of OVS ratio are done from 1 trigger)

ADC_TRIGGEREDMODE_MULTI_TRIGGER

ADC oversampling discontinuous mode: discontinuous mode (each conversion of OVS ratio needs a trigger)

Oversampling - Oversampling scope for ADC group regular

ADC_REGOVERSAMPLING_CONTINUED_MODE

Oversampling buffer maintained during injection sequence

ADC_REGOVERSAMPLING_RESUMED_MODE

Oversampling buffer zeroed during injection sequence

Oversampling - Data shift

ADC_RIGHTBITSHIFT_NONE

ADC oversampling no shift

ADC_RIGHTBITSHIFT_1

ADC oversampling right shift of 1 ranks

ADC_RIGHTBITSHIFT_2

ADC oversampling right shift of 2 ranks

ADC_RIGHTBITSHIFT_3

ADC oversampling right shift of 3 ranks

ADC_RIGHTBITSHIFT_4

ADC oversampling right shift of 4 ranks

ADC_RIGHTBITSHIFT_5

ADC oversampling right shift of 5 ranks

ADC_RIGHTBITSHIFT_6

ADC oversampling right shift of 6 ranks

ADC_RIGHTBITSHIFT_7

ADC oversampling right shift of 7 ranks

ADC_RIGHTBITSHIFT_8

ADC oversampling right shift of 8 ranks

ADC_RIGHTBITSHIFT_9

ADC oversampling right shift of 9 ranks

ADC_RIGHTBITSHIFT_10

ADC oversampling right shift of 10 ranks

ADC group regular - Overrun behavior on conversion data

ADC_OVR_DATA_PRESERVED

ADC group regular behavior in case of overrun: data preserved

ADC_OVR_DATA_OVERWRITTEN

ADC group regular behavior in case of overrun: data overwritten

ADC group regular - Sequencer ranks

ADC_REGULAR_RANK_1

ADC group regular sequencer rank 1

ADC_REGULAR_RANK_2

ADC group regular sequencer rank 2

ADC_REGULAR_RANK_3

ADC group regular sequencer rank 3

ADC_REGULAR_RANK_4

ADC group regular sequencer rank 4

ADC_REGULAR_RANK_5

ADC group regular sequencer rank 5

ADC_REGULAR_RANK_6

ADC group regular sequencer rank 6

ADC_REGULAR_RANK_7

ADC group regular sequencer rank 7

ADC_REGULAR_RANK_8

ADC group regular sequencer rank 8

ADC_REGULAR_RANK_9

ADC group regular sequencer rank 9

ADC_REGULAR_RANK_10

ADC group regular sequencer rank 10

ADC_REGULAR_RANK_11

ADC group regular sequencer rank 11

ADC_REGULAR_RANK_12

ADC group regular sequencer rank 12

ADC_REGULAR_RANK_13

ADC group regular sequencer rank 13

ADC_REGULAR_RANK_14

ADC group regular sequencer rank 14

ADC_REGULAR_RANK_15

ADC group regular sequencer rank 15

ADC_REGULAR_RANK_16

ADC group regular sequencer rank 16

ADC instance - Resolution

ADC_RESOLUTION_12B

ADC resolution 12 bits

ADC_RESOLUTION_10B

ADC resolution 10 bits

ADC_RESOLUTION_8B

ADC resolution 8 bits

ADC_RESOLUTION_6B

ADC resolution 6 bits

HAL ADC macro to manage HAL ADC handle, IT and flags.

__HAL_ADC_RESET_HANDLE_STATE

Description:

- Reset ADC handle state.

Parameters:

- __HANDLE__: ADC handle

Return value:

- None

__HAL_ADC_ENABLE_IT

Description:

- Enable ADC interrupt.

Parameters:

- __HANDLE__: ADC handle
- __INTERRUPT__: ADC Interrupt This parameter can be one of the following values:
  - ADC_IT_RDY ADC Ready interrupt source
  - ADC_IT_EOSMP ADC End of Sampling interrupt source
  - ADC_IT_EOC ADC End of Regular Conversion interrupt source
  - ADC_IT_EOS ADC End of Regular sequence of Conversions interrupt source
  - ADC_IT_OVR ADC overrun interrupt source
  - ADC_IT_JEOC ADC End of Injected Conversion interrupt source
  - ADC_IT_JEOS ADC End of Injected sequence of Conversions interrupt source
  - ADC_IT_AWD1 ADC Analog watchdog 1 interrupt source (main analog watchdog)
  - ADC_IT_AWD2 ADC Analog watchdog 2 interrupt source (additional analog watchdog)
  - ADC_IT_AWD3 ADC Analog watchdog 3 interrupt source (additional analog watchdog)

Return value:

- None

__HAL_ADC_DISABLE_IT

Description:

- Disable ADC interrupt.

Parameters:

- __HANDLE__: ADC handle
- __INTERRUPT__: ADC Interrupt This parameter can be one of the following values:
  - ADC_IT_RDY ADC Ready interrupt source
  - ADC_IT_EOSMP ADC End of Sampling interrupt source
  - ADC_IT_EOC ADC End of Regular Conversion interrupt source
  - ADC_IT_EOS ADC End of Regular sequence of Conversions interrupt source
  - ADC_IT_OVR ADC overrun interrupt source
  - ADC_IT_JEOC ADC End of Injected Conversion interrupt source
  - ADC_IT_JEOS ADC End of Injected sequence of Conversions interrupt source
  - ADC_IT_AWD1 ADC Analog watchdog 1 interrupt source (main analog watchdog)
  - ADC_IT_AWD2 ADC Analog watchdog 2 interrupt source (additional analog watchdog)
  - ADC_IT_AWD3 ADC Analog watchdog 3 interrupt source (additional analog watchdog)

Return value:

- None

__HAL_ADC_GET_IT_SOURCE

Description:

- Checks if the specified ADC interrupt source is enabled or disabled.

Parameters:

- __HANDLE__: ADC handle
- __INTERRUPT__: ADC interrupt source to check This parameter can be one of the following values:
  - ADC_IT_RDY ADC Ready interrupt source
  - ADC_IT_EOSMP ADC End of Sampling interrupt source
  - ADC_IT_EOC ADC End of Regular Conversion interrupt source
  - ADC_IT_EOS ADC End of Regular sequence of Conversions interrupt source
  - ADC_IT_OVR ADC overrun interrupt source
  - ADC_IT_JEOC ADC End of Injected Conversion interrupt source
  - ADC_IT_JEOS ADC End of Injected sequence of Conversions interrupt source
  - ADC_IT_AWD1 ADC Analog watchdog 1 interrupt source (main analog watchdog)
  - ADC_IT_AWD2 ADC Analog watchdog 2 interrupt source (additional analog watchdog)
  - ADC_IT_AWD3 ADC Analog watchdog 3 interrupt source (additional analog watchdog)

Return value:

- State: of interruption (SET or RESET)

__HAL_ADC_GET_FLAG

Description:

- Check whether the specified ADC flag is set or not.

Parameters:

- __HANDLE__: ADC handle
- __FLAG__: ADC flag This parameter can be one of the following values:
  - ADC_FLAG_RDY ADC Ready flag
  - ADC_FLAG_EOSMP ADC End of Sampling flag
  - ADC_FLAG_EOC ADC End of Regular Conversion flag
  - ADC_FLAG_EOS ADC End of Regular sequence of Conversions flag
  - ADC_FLAG_OVR ADC overrun flag
  - ADC_FLAG_JEOC ADC End of Injected Conversion flag
  - ADC_FLAG_JEOS ADC End of Injected sequence of Conversions flag
  - ADC_FLAG_AWD1 ADC Analog watchdog 1 flag (main analog watchdog)
  - ADC_FLAG_AWD2 ADC Analog watchdog 2 flag (additional analog watchdog)
  - ADC_FLAG_AWD3 ADC Analog watchdog 3 flag (additional analog watchdog)

Return value:

- State: of flag (TRUE or FALSE).

__HAL_ADC_CLEAR_FLAG

Description:

- Clear the specified ADC flag.

Parameters:

- __HANDLE__: ADC handle
- __FLAG__: ADC flag This parameter can be one of the following values:
  - ADC_FLAG_RDY ADC Ready flag
  - ADC_FLAG_EOSMP ADC End of Sampling flag
  - ADC_FLAG_EOC ADC End of Regular Conversion flag
  - ADC_FLAG_EOS ADC End of Regular sequence of Conversions flag
  - ADC_FLAG_OVR ADC overrun flag
  - ADC_FLAG_JEOC ADC End of Injected Conversion flag
  - ADC_FLAG_JEOS ADC End of Injected sequence of Conversions flag
  - ADC_FLAG_AWD1 ADC Analog watchdog 1 flag (main analog watchdog)
  - ADC_FLAG_AWD2 ADC Analog watchdog 2 flag (additional analog watchdog)
  - ADC_FLAG_AWD3 ADC Analog watchdog 3 flag (additional analog watchdog)

Return value:

- None

HAL ADC helper macro

__HAL_ADC_CHANNEL_TO_DECIMAL_NB

Description:

- Helper macro to get ADC channel number in decimal format from literals ADC_CHANNEL_x.

Parameters:

- __CHANNEL__: This parameter can be one of the following values:
  - ADC_CHANNEL_0
  - ADC_CHANNEL_1
  - ADC_CHANNEL_2
  - ADC_CHANNEL_3
  - ADC_CHANNEL_4
  - ADC_CHANNEL_5
  - ADC_CHANNEL_6
  - ADC_CHANNEL_7
  - ADC_CHANNEL_8
  - ADC_CHANNEL_9
  - ADC_CHANNEL_10
  - ADC_CHANNEL_11
  - ADC_CHANNEL_12
  - ADC_CHANNEL_13
  - ADC_CHANNEL_14
  - ADC_CHANNEL_15
  - ADC_CHANNEL_16
  - ADC_CHANNEL_17
  - ADC_CHANNEL_18
  - ADC_CHANNEL_19
  - ADC_CHANNEL_VREFINT (1)(4)
  - ADC_CHANNEL_VBAT (2)(4)
  - ADC_CHANNEL_VDDCORE (2)(4)

Return value:

- Value: between Min_Data=0 and Max_Data=18

Notes:

- Example: __HAL_ADC_CHANNEL_TO_DECIMAL_NB(ADC_CHANNEL_4) will return decimal number "4". The input can be a value from functions where a channel number is returned, either defined with number or with bitfield (only one bit must be set).

__HAL_ADC_DECIMAL_NB_TO_CHANNEL

Description:

- Helper macro to get ADC channel in literal format ADC_CHANNEL_x from number in decimal format.

Parameters:

- __DECIMAL_NB__: Value between Min_Data=0 and Max_Data=18

Return value:

- Returned: value can be one of the following values:
  - ADC_CHANNEL_0
  - ADC_CHANNEL_1
  - ADC_CHANNEL_2
  - ADC_CHANNEL_3
  - ADC_CHANNEL_4
  - ADC_CHANNEL_5
  - ADC_CHANNEL_6
  - ADC_CHANNEL_7
  - ADC_CHANNEL_8
  - ADC_CHANNEL_9
  - ADC_CHANNEL_10
  - ADC_CHANNEL_11
  - ADC_CHANNEL_12
  - ADC_CHANNEL_13
  - ADC_CHANNEL_14
  - ADC_CHANNEL_15
  - ADC_CHANNEL_16
  - ADC_CHANNEL_17
  - ADC_CHANNEL_18
  - ADC_CHANNEL_19
  - ADC_CHANNEL_VREFINT (1)(3)
  - ADC_CHANNEL_VBAT (2)(3)
  - ADC_CHANNEL_VDDCORE (2)(3)

Notes:

- Example: __HAL_ADC_DECIMAL_NB_TO_CHANNEL(4) will return a data equivalent to "ADC_CHANNEL_4".

__HAL_ADC_IS_CHANNEL_INTERNAL

Description:

- Helper macro to determine whether the selected channel corresponds to literal definitions of driver.

Parameters:

- __CHANNEL__: This parameter can be one of the following values:
  - ADC_CHANNEL_0
  - ADC_CHANNEL_1
  - ADC_CHANNEL_2
  - ADC_CHANNEL_3
  - ADC_CHANNEL_4
  - ADC_CHANNEL_5
  - ADC_CHANNEL_6
  - ADC_CHANNEL_7
  - ADC_CHANNEL_8
  - ADC_CHANNEL_9
  - ADC_CHANNEL_10
  - ADC_CHANNEL_11
  - ADC_CHANNEL_12
  - ADC_CHANNEL_13
  - ADC_CHANNEL_14
  - ADC_CHANNEL_15
  - ADC_CHANNEL_16
  - ADC_CHANNEL_17
  - ADC_CHANNEL_18
  - ADC_CHANNEL_19
  - ADC_CHANNEL_VREFINT (1)(4)
  - ADC_CHANNEL_VBAT (2)(4)
  - ADC_CHANNEL_VDDCORE (2)(4)

Return value:

- Value: "0" if the channel corresponds to a parameter definition of a ADC external channel (channel connected to a GPIO pin). Value "1" if the channel corresponds to a parameter definition of a ADC internal channel.

Notes:

- The different literal definitions of ADC channels are: ADC internal channel: ADC_CHANNEL_VREFINT, ...ADC external channel (channel connected to a GPIO pin): ADC_CHANNEL_1, ADC_CHANNEL_2, ... The channel parameter must be a value defined from literal definition of a ADC internal channel (ADC_CHANNEL_VREFINT, ...), ADC external channel (ADC_CHANNEL_1, ADC_CHANNEL_2, ...), must not be a value from functions where a channel number is returned from ADC registers, because internal and external channels share the same channel number in ADC registers. The differentiation is made only with parameters definitions of driver.

__HAL_ADC_CHANNEL_INTERNAL_TO_EXTERNAL

Description:

- Helper macro to convert a channel defined from parameter definition of a ADC internal channel (ADC_CHANNEL_VREFINT, ...), to its equivalent parameter definition of a ADC external channel (ADC_CHANNEL_1, ADC_CHANNEL_2, ...).

Parameters:

- __CHANNEL__: This parameter can be one of the following values:
  - ADC_CHANNEL_0
  - ADC_CHANNEL_1
  - ADC_CHANNEL_2
  - ADC_CHANNEL_3
  - ADC_CHANNEL_4
  - ADC_CHANNEL_5
  - ADC_CHANNEL_6
  - ADC_CHANNEL_7
  - ADC_CHANNEL_8
  - ADC_CHANNEL_9
  - ADC_CHANNEL_10
  - ADC_CHANNEL_11
  - ADC_CHANNEL_12
  - ADC_CHANNEL_13
  - ADC_CHANNEL_14
  - ADC_CHANNEL_15
  - ADC_CHANNEL_16
  - ADC_CHANNEL_17
  - ADC_CHANNEL_18
  - ADC_CHANNEL_19
  - ADC_CHANNEL_VREFINT (1)(4)
  - ADC_CHANNEL_VBAT (2)(4)
  - ADC_CHANNEL_VDDCORE (2)(4)

Return value:

- Returned: value can be one of the following values:
  - ADC_CHANNEL_0
  - ADC_CHANNEL_1
  - ADC_CHANNEL_2
  - ADC_CHANNEL_3
  - ADC_CHANNEL_4
  - ADC_CHANNEL_5
  - ADC_CHANNEL_6
  - ADC_CHANNEL_7
  - ADC_CHANNEL_8
  - ADC_CHANNEL_9
  - ADC_CHANNEL_10
  - ADC_CHANNEL_11
  - ADC_CHANNEL_12
  - ADC_CHANNEL_13
  - ADC_CHANNEL_14
  - ADC_CHANNEL_15
  - ADC_CHANNEL_16
  - ADC_CHANNEL_17
  - ADC_CHANNEL_18
  - ADC_CHANNEL_19

Notes:

- The channel parameter can be, additionally to a value defined from parameter definition of a ADC internal channel (ADC_CHANNEL_VREFINT, ...), a value defined from parameter definition of ADC external channel (ADC_CHANNEL_1, ADC_CHANNEL_2, ...) or a value from functions where a channel number is returned from ADC registers.

__HAL_ADC_IS_CHANNEL_INTERNAL_AVAILABLE

Description:

- Helper macro to determine whether the internal channel selected is available on the ADC instance selected.

Parameters:

- __ADC_INSTANCE__: ADC instance
- __CHANNEL__: This parameter can be one of the following values:
  - ADC_CHANNEL_VREFINT (1)
  - ADC_CHANNEL_VBAT (2)
  - ADC_CHANNEL_VDDCORE (2)

Return value:

- Value: "0" if the internal channel selected is not available on the ADC instance selected. Value "1" if the internal channel selected is available on the ADC instance selected.

Notes:

- The channel parameter must be a value defined from parameter definition of a ADC internal channel (ADC_CHANNEL_VREFINT, ...), must not be a value defined from parameter definition of ADC external channel (ADC_CHANNEL_1, ADC_CHANNEL_2, ...) or a value from functions where a channel number is returned from ADC registers, because internal and external channels share the same channel number in ADC registers. The differentiation is made only with parameters definitions of driver.

__HAL_ADC_MULTI_CONV_DATA_MASTER_SLAVE

Description:

- Helper macro to get the ADC multimode conversion data of ADC master or ADC slave from raw value with both ADC conversion data concatenated.

Parameters:

- __ADC_MULTI_MASTER_SLAVE__: This parameter can be one of the following values:
  - LL_ADC_MULTI_MASTER
  - LL_ADC_MULTI_SLAVE
- __ADC_MULTI_CONV_DATA__: Value between Min_Data=0x000 and Max_Data=0xFFF

Return value:

- Value: between Min_Data=0x000 and Max_Data=0xFFF

Notes:

- This macro is intended to be used when multimode transfer by DMA is enabled: refer to function __LL_ADC_MULTI_CONV_DATA_MASTER_SLAVE(). In this case the transferred data need to processed with this macro to separate the conversion data of ADC master and ADC slave.

__HAL_ADC_COMMON_INSTANCE

Description:

- Helper macro to select the ADC common instance to which is belonging the selected ADC instance.

Parameters:

- __ADCx__: ADC instance

Return value:

- ADC: common register instance

Notes:

- ADC common register instance can be used for: Set parameters common to several ADC instancesMultimode (for devices with several ADC instances) Refer to functions having argument "ADCxy_COMMON" as parameter.

__HAL_ADC_IS_ENABLED_ALL_COMMON_INSTANCE

Description:

- Helper macro to check if all ADC instances sharing the same ADC common instance are disabled.

Parameters:

- __ADCXY_COMMON__: ADC common instance (can be set directly from CMSIS definition or by using helper macro

Return value:

- Value: "0" if all ADC instances sharing the same ADC common instance are disabled. Value "1" if at least one ADC instance sharing the same ADC common instance is enabled.

Notes:

- This check is required by functions with setting conditioned to ADC state: All ADC instances of the ADC common group must be disabled. Refer to functions having argument "ADCxy_COMMON" as parameter. On devices with only 1 ADC common instance, parameter of this macro is useless and can be ignored (parameter kept for compatibility with devices featuring several ADC common instances).

__HAL_ADC_DIGITAL_SCALE

Description:

- Helper macro to define the ADC conversion data full-scale digital value corresponding to the selected ADC resolution.

Parameters:

- __ADC_RESOLUTION__: This parameter can be one of the following values:
  - ADC_RESOLUTION_12B
  - ADC_RESOLUTION_10B
  - ADC_RESOLUTION_8B
  - ADC_RESOLUTION_6B

Return value:

- ADC: conversion data full-scale digital value

Notes:

- ADC conversion data full-scale corresponds to voltage range determined by analog voltage references Vref+ and Vref- (refer to reference manual).

__HAL_ADC_CONVERT_DATA_RESOLUTION

Description:

- Helper macro to convert the ADC conversion data from a resolution to another resolution.

Parameters:

- __DATA__: ADC conversion data to be converted
- __ADC_RESOLUTION_CURRENT__: Resolution of to the data to be converted This parameter can be one of the following values:
  - ADC_RESOLUTION_12B
  - ADC_RESOLUTION_10B
  - ADC_RESOLUTION_8B
  - ADC_RESOLUTION_6B
- __ADC_RESOLUTION_TARGET__: Resolution of the data after conversion This parameter can be one of the following values:
  - ADC_RESOLUTION_12B
  - ADC_RESOLUTION_10B
  - ADC_RESOLUTION_8B
  - ADC_RESOLUTION_6B

Return value:

- ADC: conversion data to the requested resolution

__HAL_ADC_CALC_DATA_TO_VOLTAGE

Description:

- Helper macro to calculate the voltage (unit: mVolt) corresponding to a ADC conversion data (unit: digital value).

Parameters:

- __VREFANALOG_VOLTAGE__: Analog reference voltage (unit: mV)
- __ADC_DATA__: ADC conversion data (resolution 12 bits) (unit: digital value).
- __ADC_RESOLUTION__: This parameter can be one of the following values:
  - ADC_RESOLUTION_12B
  - ADC_RESOLUTION_10B
  - ADC_RESOLUTION_8B
  - ADC_RESOLUTION_6B

Return value:

- ADC: conversion data equivalent voltage value (unit: mVolt)

Notes:

- Analog reference voltage (Vref+) must be either known from user board environment.

__HAL_ADC_CALC_DIFF_DATA_TO_VOLTAGE

Description:

- Helper macro to calculate the voltage (unit: mVolt) corresponding to a ADC conversion data (unit: digital value) in differential ended mode.

Parameters:

- __VREFANALOG_VOLTAGE__: Analog reference voltage (unit: mV)
- __ADC_DATA__: ADC conversion data (resolution 12 bits) (unit: digital value).
- __ADC_RESOLUTION__: This parameter can be one of the following values:
  - ADC_RESOLUTION_12B
  - ADC_RESOLUTION_10B
  - ADC_RESOLUTION_8B
  - ADC_RESOLUTION_6B

Return value:

- ADC: conversion data equivalent voltage value (unit: mVolt)

Notes:

- Analog reference voltage (Vref+) must be either known from user board environment.

ADC group injected trigger edge (when external trigger is selected)

ADC_EXTERNALTRIGINJECCONV_EDGE_NONE

Injected conversions trigger disabled (SW start)

ADC_EXTERNALTRIGINJECCONV_EDGE_RISING

Injected conversions trigger polarity set to rising edge

ADC_EXTERNALTRIGINJECCONV_EDGE_FALLING

Injected conversions trigger polarity set to falling edge

ADC_EXTERNALTRIGINJECCONV_EDGE_RISINGFALLING

Injected conversions trigger polarity set to both rising and falling edges

ADC group injected trigger source

ADC_INJECTED_SOFTWARE_START

ADC group injected conversion trigger internal: SW start.

ADC_EXTERNALTRIGINJEC_EXT_IT15

ADC group injected conversion trigger from external peripheral: external interrupt line 15.

ADC_EXTERNALTRIGINJEC_T1_CC4

ADC group injected conversion trigger from external peripheral: TIM1 channel 4 event (capture compare).

ADC_EXTERNALTRIGINJEC_T1_TRGO

ADC group injected conversion trigger from external peripheral: TIM1 TRGO event.

ADC_EXTERNALTRIGINJEC_T1_TRGO2

ADC group injected conversion trigger from external peripheral: TIM1 TRGO2 event.

ADC_EXTERNALTRIGINJEC_T2_CC1

ADC group injected conversion trigger from external peripheral: TIM2 channel 1 event (capture compare).

ADC_EXTERNALTRIGINJEC_T2_TRGO

ADC group injected conversion trigger from external peripheral: TIM2 TRGO event.

ADC_EXTERNALTRIGINJEC_T3_CC1

ADC group injected conversion trigger from external peripheral: TIM3 channel 1 event (capture compare).

ADC_EXTERNALTRIGINJEC_T3_CC3

ADC group injected conversion trigger from external peripheral: TIM3 channel 3 event (capture compare).

ADC_EXTERNALTRIGINJEC_T3_CC4

ADC group injected conversion trigger from external peripheral: TIM3 channel 4 event (capture compare).

ADC_EXTERNALTRIGINJEC_T3_TRGO

ADC group injected conversion trigger from external peripheral: TIM3 TRGO event.

ADC_EXTERNALTRIGINJEC_T4_TRGO

ADC group injected conversion trigger from external peripheral: TIM4 TRGO event.

ADC_EXTERNALTRIGINJEC_T5_TRGO

ADC group injected conversion trigger from external peripheral: TIM5 TRGO event.

ADC_EXTERNALTRIGINJEC_T6_TRGO

ADC group injected conversion trigger from external peripheral: TIM6 TRGO event.

ADC_EXTERNALTRIGINJEC_T7_TRGO

ADC group injected conversion trigger from external peripheral: TIM7 TRGO event.

ADC_EXTERNALTRIGINJEC_T8_TRGO

ADC group injected conversion trigger from external peripheral: TIM8 TRGO event.

ADC_EXTERNALTRIGINJEC_T8_TRGO2

ADC group injected conversion trigger from external peripheral: TIM8 TRGO2 event.

ADC_EXTERNALTRIGINJEC_T9_CC2

ADC group injected conversion trigger from external peripheral: TIM9 channel 2 event (capture compare).

ADC_EXTERNALTRIGINJEC_T9_TRGO

ADC group injected conversion trigger from external peripheral: TIM9 TRGO event.

ADC_EXTERNALTRIGINJEC_T12_TRGO

ADC group injected conversion trigger from external peripheral: TIM12 TRGO event.

ADC_EXTERNALTRIGINJEC_T12_CC4

ADC group injected conversion trigger from external peripheral: TIM12 channel 4 event (capture compare).

ADC_EXTERNALTRIGINJEC_T15_TRGO

ADC group injected conversion trigger from external peripheral: TIM12 TRGO event.

ADC_EXTERNALTRIGINJEC_T18_TRGO

ADC group injected conversion trigger from external peripheral: TIM12 TRGO event.

ADC_EXTERNALTRIGINJEC_LPTIM1_CH2

ADC group injected conversion trigger from external peripheral: LPTIM1 channel 2 event.

ADC_EXTERNALTRIGINJEC_LPTIM2_CH2

ADC group injected conversion trigger from external peripheral: LPTIM2 channel 2 event.

ADC_EXTERNALTRIGINJEC_LPTIM3_CH2

ADC group injected conversion trigger from external peripheral: LPTIM3 channel 2 event.

ADC group injected - Sequencer ranks

ADC_INJECTED_RANK_1

ADC group injected sequencer rank 1

ADC_INJECTED_RANK_2

ADC group injected sequencer rank 2

ADC_INJECTED_RANK_3

ADC group injected sequencer rank 3

ADC_INJECTED_RANK_4

ADC group injected sequencer rank 4

ADC interrupts definition

ADC_IT_RDY

ADC Ready interrupt source

ADC_IT_EOSMP

ADC End of sampling interrupt source

ADC_IT_EOC

ADC End of regular conversion interrupt source

ADC_IT_EOS

ADC End of regular sequence of conversions interrupt source

ADC_IT_OVR

ADC overrun interrupt source

ADC_IT_JEOC

ADC End of injected conversion interrupt source

ADC_IT_JEOS

ADC End of injected sequence of conversions interrupt source

ADC_IT_AWD1

ADC Analog watchdog 1 interrupt source (main analog watchdog)

ADC_IT_AWD2

ADC Analog watchdog 2 interrupt source (additional analog watchdog)

ADC_IT_AWD3

ADC Analog watchdog 3 interrupt source (additional analog watchdog)

ADC group regular trigger edge (when external trigger is selected)

ADC_EXTERNALTRIGCONVEDGE_NONE

ADC group regular trigger disabled (SW start)

ADC_EXTERNALTRIGCONVEDGE_RISING

ADC group regular conversion trigger polarity set to rising edge

ADC_EXTERNALTRIGCONVEDGE_FALLING

ADC group regular conversion trigger polarity set to falling edge

ADC_EXTERNALTRIGCONVEDGE_RISINGFALLING

ADC group regular conversion trigger polarity set to both rising and falling edges

ADC group regular trigger source

ADC_SOFTWARE_START

ADC group regular conversion trigger internal: SW start.

ADC_EXTERNALTRIG_EXT_IT11

ADC group regular conversion trigger from external peripheral: external interrupt line 11.

ADC_EXTERNALTRIG_T1_CC1

ADC group regular conversion trigger from external peripheral: TIM1 channel 1 event (capture compare).

ADC_EXTERNALTRIG_T1_CC2

ADC group regular conversion trigger from external peripheral: TIM1 channel 2 event (capture compare).

ADC_EXTERNALTRIG_T1_CC3

ADC group regular conversion trigger from external peripheral: TIM1 channel 3 event (capture compare).

ADC_EXTERNALTRIG_T1_TRGO

ADC group regular conversion trigger from external peripheral: TIM1 TRGO event.

ADC_EXTERNALTRIG_T1_TRGO2

ADC group regular conversion trigger from external peripheral: TIM1 TRGO2 event.

ADC_EXTERNALTRIG_T2_CC2

ADC group regular conversion trigger from external peripheral: TIM2 channel 2 event (capture compare).

ADC_EXTERNALTRIG_T2_TRGO

ADC group regular conversion trigger from external peripheral: TIM2 TRGO event.

ADC_EXTERNALTRIG_T3_CC4

ADC group regular conversion trigger from external peripheral: TIM3 channel 4 event (capture compare).

ADC_EXTERNALTRIG_T3_TRGO

ADC group regular conversion trigger from external peripheral: TIM3 TRGO event.

ADC_EXTERNALTRIG_T4_CC4

ADC group regular conversion trigger from external peripheral: TIM4 channel 4 event (capture compare).

ADC_EXTERNALTRIG_T4_TRGO

ADC group regular conversion trigger from external peripheral: TIM4 TRGO event.

ADC_EXTERNALTRIG_T5_TRGO

ADC group regular conversion trigger from external peripheral: TIM5 TRGO event.

ADC_EXTERNALTRIG_T6_TRGO

ADC group regular conversion trigger from external peripheral: TIM6 TRGO event.

ADC_EXTERNALTRIG_T7_TRGO

ADC group regular conversion trigger from external peripheral: TIM7 TRGO event.

ADC_EXTERNALTRIG_T8_TRGO

ADC group regular conversion trigger from external peripheral: TIM8 TRGO event.

ADC_EXTERNALTRIG_T8_TRGO2

ADC group regular conversion trigger from external peripheral: TIM8 TRGO2 event.

ADC_EXTERNALTRIG_T9_CC1

ADC group regular conversion trigger from external peripheral: TIM9 channel 1 event (capture compare).

ADC_EXTERNALTRIG_T9_TRGO

ADC group regular conversion trigger from external peripheral: TIM9 TRGO event.

ADC_EXTERNALTRIG_T12_TRGO

ADC group regular conversion trigger from external peripheral: TIM12 TRGO event.

ADC_EXTERNALTRIG_T15_TRGO

ADC group regular conversion trigger from external peripheral: TIM15 TRGO event.

ADC_EXTERNALTRIG_T18_TRGO

ADC group regular conversion trigger from external peripheral: TIM18 TRGO event.

ADC_EXTERNALTRIG_LPTIM1_CH1

ADC group regular conversion trigger from external peripheral: LPTIM1 channel 1 event.

ADC_EXTERNALTRIG_LPTIM2_CH1

ADC group regular conversion trigger from external peripheral: LPTIM2 channel 2 event.

ADC_EXTERNALTRIG_LPTIM3_CH1

ADC group regular conversion trigger from external peripheral: LPTIM3 channel 3 event.

ADC group regular sampling mode

ADC_SAMPLING_MODE_NORMAL

ADC conversions sampling phase duration is defined using

ADC_SAMPLING_MODE_BULB

ADC conversions sampling phase starts immediately after end of conversion, and stops upon trigger event. Note: First conversion is using minimal sampling time (see

ADC_SAMPLING_MODE_TRIGGER_CONTROLED

ADC conversions sampling phase is controlled by trigger events: Trigger rising edge = start sampling Trigger falling edge = stop sampling and start conversion

ADC sequencer scan mode

ADC_SCAN_DISABLE

Scan mode disabled

ADC_SCAN_ENABLE

Scan mode enabled

ADC_CHSELR_SQ2_TO_SQ8

ADCx SMPR1 fields

ADC_SMPR1_FIELDS

ADC States

HAL_ADC_STATE_RESET

Notes:

- ADC state machine is managed by bitfields, state must be compared with bit by bit. For example: " if ((HAL_ADC_GetState(hadc1) & HAL_ADC_STATE_REG_BUSY) != 0UL) " " if ((HAL_ADC_GetState(hadc1) & HAL_ADC_STATE_AWD1) != 0UL) " ADC not yet initialized or disabled

HAL_ADC_STATE_READY

ADC peripheral ready for use

HAL_ADC_STATE_BUSY_INTERNAL

ADC is busy due to an internal process (initialization, calibration, ...)

HAL_ADC_STATE_TIMEOUT

TimeOut occurrence

HAL_ADC_STATE_ERROR_INTERNAL

Internal error occurrence

HAL_ADC_STATE_ERROR_CONFIG

Configuration error occurrence

HAL_ADC_STATE_ERROR_DMA

DMA error occurrence

HAL_ADC_STATE_REG_BUSY

A conversion on ADC group regular is ongoing or can occur (either by continuous mode, external trigger, low power auto power-on (if feature available), multimode ADC master control (if feature available))

HAL_ADC_STATE_REG_EOC

Conversion data available on group regular

HAL_ADC_STATE_REG_OVR

Overrun occurrence

HAL_ADC_STATE_REG_EOSMP

Not available on this STM32 series: End Of Sampling flag raised

HAL_ADC_STATE_INJ_BUSY

A conversion on ADC group injected is ongoing or can occur (either by auto-injection mode, external trigger, low power auto power-on (if feature available), multimode ADC master control (if feature available))

HAL_ADC_STATE_INJ_EOC

Conversion data available on group injected

HAL_ADC_STATE_AWD1

Out-of-window occurrence of ADC analog watchdog 1

HAL_ADC_STATE_AWD2

Out-of-window occurrence of ADC analog watchdog 2

HAL_ADC_STATE_AWD3

Out-of-window occurrence of ADC analog watchdog 3

HAL_ADC_STATE_MULTIMODE_SLAVE

ADC in multimode slave state, controlled by another ADC master (when feature available)

8 HAL ADC Extension Driver

8.1 ADCEx Firmware driver registers structures

8.1.1 ADC_InjOversamplingTypeDef

ADC_InjOversamplingTypeDef is defined in the stm32n6xx_hal_adc_ex.h

Data Fields

- ***uint32_t Ratio***
- ***uint32_t RightBitShift***

Field Documentation

- ***uint32_t ADC_InjOversamplingTypeDef::Ratio***
  Configures the oversampling ratio. This parameter can be a value from 1 to 1024
- ***uint32_t ADC_InjOversamplingTypeDef::RightBitShift***
  Configures the division coefficient for the Oversampler. This parameter can be a value of ***ADC_HAL_EC_OVS_SHIFT***

8.1.2 ADC_InjectionConfTypeDef

ADC_InjectionConfTypeDef is defined in the stm32n6xx_hal_adc_ex.h

Data Fields

- ***uint32_t InjectedChannel***
- ***uint32_t InjectedRank***
- ***uint32_t InjectedSamplingTime***
- ***uint32_t InjectedSingleDiff***
- ***uint32_t InjectedOffsetNumber***
- ***uint32_t InjectedOffset***
- ***FunctionalState InjectedOffsetSignedSaturation***
- ***uint32_t InjectedOffsetSign***
- ***FunctionalState InjectedOffsetSaturation***
- ***uint32_t InjectedNbrOfConversion***
- ***FunctionalState InjectedDiscontinuousConvMode***
- ***FunctionalState AutoInjectedConv***
- ***uint32_t ExternalTrigInjecConv***
- ***uint32_t ExternalTrigInjecConvEdge***
- ***FunctionalState InjecOversamplingMode***
- ***ADC_InjOversamplingTypeDef InjecOversampling***

Field Documentation

- ***uint32_t ADC_InjectionConfTypeDef::InjectedChannel***
  Specifies the channel to configure into ADC group injected. This parameter can be a value of ***ADC_HAL_EC_CHANNEL*** Note: Depending on devices and ADC instances, some channels may not be available on device package pins. Refer to device datasheet for channels availability.
- ***uint32_t ADC_InjectionConfTypeDef::InjectedRank***
  Specifies the rank in the ADC group injected sequencer. This parameter must be a value of ***ADC_INJ_SEQ_RANKS***. Note: to disable a channel or change order of conversion sequencer, rank containing a previous channel setting can be overwritten by the new channel setting (or parameter number of conversions adjusted)

- ***uint32_t ADC_InjectionConfTypeDef::InjectedSamplingTime***
  Sampling time value to be set for the selected channel. Unit: ADC clock cycles. Conversion time is the addition of sampling time and processing time (13.5 ADC clock cycles at ADC resolution 12 bits, 11.5 cycles at 10 bits, 8.5 cycles at 8 bits, 6.5 cycles at 6 bits). This parameter can be a value of ***ADC_HAL_EC_CHANNEL_SAMPLINGTIME***. Caution: This parameter applies to a channel that can be used in a regular and/or injected group. It overwrites the last setting. Note: In case of usage of internal measurement channels (VrefInt/Vbat/TempSensor), sampling time constraints must be respected (sampling time can be adjusted in function of ADC clock frequency and sampling time setting); Refer to device datasheet for timings values.
- ***uint32_t ADC_InjectionConfTypeDef::InjectedSingleDiff***
  Selection of single-ended or differential input. In differential mode: Differential measurement is carried out between the selected channel (positive input) and another channel (negative input). Refer to reference manual for corresponding channel. Only channel of positive input has to be configured, the other channel is configured automatically. This parameter must be a value of:
  ADC_HAL_EC_CHANNEL_SINGLE_DIFF_ENDING. Caution: This parameter applies to a channel that can be used in a regular and/or injected group. It overwrites the last setting. Note: Refer to Reference Manual to ensure the selected channel is available in differential mode. Note: This parameter must be modified when ADC is disabled (before ADC start conversion or after ADC stop conversion). If ADC is enabled, this parameter setting is bypassed without error reporting (as it can be the expected behavior in case of another parameter update on the fly)
- ***uint32_t ADC_InjectionConfTypeDef::InjectedOffsetNumber***
  Selects the offset number. This parameter can be a value of ***ADC_HAL_EC_OFFSET_NB***. Caution: Only one offset is allowed per channel. This parameter overwrites the last setting.
- ***uint32_t ADC_InjectionConfTypeDef::InjectedOffset***
  Defines the offset to be subtracted from the raw converted data. Offset value must be a positive number. Depending of ADC resolution selected (12, 10, 8 or 6 bits), this parameter must be a number between Min_Data = 0x000 and Max_Data = 0xFFF, 0x3FF, 0xFF or 0x3F respectively. Note: This parameter must be modified when no conversion is on going on both regular and injected groups (ADC disabled, or ADC enabled without continuous mode or external trigger that could launch a conversion).
- ***FunctionalState ADC_InjectionConfTypeDef::InjectedOffsetSignedSaturation***
  Specifies whether the Signed saturation feature is used or not. This parameter is applied only for 14-bit or 8-bit resolution. This parameter can be set to ENABLE or DISABLE.
- ***uint32_t ADC_InjectionConfTypeDef::InjectedOffsetSign***
  Define if the offset should be subtracted (negative sign) or added (positive sign) from or to the raw converted data. This parameter can be a value of ***ADCEx_OffsetSign***. Note:
  - This parameter must be modified when no conversion is on going on both regular and injected groups (ADC disabled or ADC enabled without continuous mode or external trigger that could launch a conversion).
  - This parameter is specific to ADC1 only.
- ***FunctionalState ADC_InjectionConfTypeDef::InjectedOffsetSaturation***
  Define if the offset should be saturated upon under or over flow. This parameter value can be ENABLE or DISABLE. Note: This parameter must be modified when no conversion is on going on both regular and injected groups (ADC disabled, or ADC enabled without continuous mode or external trigger that could launch a conversion).
- ***uint32_t ADC_InjectionConfTypeDef::InjectedNbrOfConversion***
  Specifies the number of ranks that will be converted within the ADC group injected sequencer. To use the injected group sequencer and convert several ranks, parameter 'ScanConvMode' must be enabled. This parameter must be a number between Min_Data = 1 and Max_Data = 4. Caution: this setting impacts the entire injected group. Therefore, call of **HAL_ADCEx_InjectedConfigChannel()** to configure a channel on injected group can impact the configuration of other channels previously set.
- ***FunctionalState ADC_InjectionConfTypeDef::InjectedDiscontinuousConvMode***
  Specifies whether the conversions sequence of ADC group injected is performed in Complete-sequence/ Discontinuous-sequence (main sequence subdivided in successive parts). Discontinuous mode is used only if sequencer is enabled (parameter 'ScanConvMode'). If sequencer is disabled, this parameter is discarded. Discontinuous mode can be enabled only if continuous mode is disabled. This parameter can be set to ENABLE or DISABLE. Note: This parameter must be modified when ADC is disabled (before ADC start conversion or after ADC stop conversion). Note: For injected group, discontinuous mode converts the sequence channel by channel (discontinuous length fixed to 1 rank). Caution: this setting impacts the entire injected group. Therefore, call of **HAL_ADCEx_InjectedConfigChannel()** to configure a channel on injected group can impact the configuration of other channels previously set.

- ***FunctionalState ADC_InjectionConfTypeDef::AutoInjectedConv***
  Enables or disables the selected ADC group injected automatic conversion after regular one This parameter can be set to ENABLE or DISABLE. Note: To use Automatic injected conversion, discontinuous mode must be disabled ('DiscontinuousConvMode' and 'InjectedDiscontinuousConvMode' set to DISABLE) Note: To use Automatic injected conversion, injected group external triggers must be disabled ('ExternalTrigInjecConv' set to ADC_INJECTED_SOFTWARE_START) Note: In case of DMA used with regular group: if DMA configured in normal mode (single shot) JAUTO will be stopped upon DMA transfer complete. To maintain JAUTO always enabled, DMA must be configured in circular mode. Caution: this setting impacts the entire injected group. Therefore, call of **HAL_ADCEx_InjectedConfigChannel()** to configure a channel on injected group can impact the configuration of other channels previously set.
- ***uint32_t ADC_InjectionConfTypeDef::ExternalTrigInjecConv***
  Selects the external event used to trigger the conversion start of injected group. If set to ADC_INJECTED_SOFTWARE_START, external triggers are disabled and software trigger is used instead. This parameter can be a value of ***ADC_injected_external_trigger_source***. Caution: this setting impacts the entire injected group. Therefore, call of **HAL_ADCEx_InjectedConfigChannel()** to configure a channel on injected group can impact the configuration of other channels previously set.
- ***uint32_t ADC_InjectionConfTypeDef::ExternalTrigInjecConvEdge***
  Selects the external trigger edge of injected group. This parameter can be a value of ***ADC_injected_external_trigger_edge***. If trigger source is set to ADC_INJECTED_SOFTWARE_START, this parameter is discarded. Caution: this setting impacts the entire injected group. Therefore, call of **HAL_ADCEx_InjectedConfigChannel()** to configure a channel on injected group can impact the configuration of other channels previously set.
- ***FunctionalState ADC_InjectionConfTypeDef::InjecOversamplingMode***
  Specifies whether the oversampling feature is enabled or disabled. This parameter can be set to ENABLE or DISABLE. Note: This parameter can be modified only if there is no conversion is ongoing (both ADSTART and JADSTART cleared).
- ***ADC_InjOversamplingTypeDef ADC_InjectionConfTypeDef::InjecOversampling***
  Specifies the Oversampling parameters. Caution: this setting overwrites the previous oversampling configuration if oversampling already enabled. Note: This parameter can be modified only if there is no conversion is ongoing (both ADSTART and JADSTART cleared).

8.1.3 ADC_MultiModeTypeDef

ADC_MultiModeTypeDef is defined in the stm32n6xx_hal_adc_ex.h

Data Fields

- ***uint32_t Mode***
- ***uint32_t DualModeData***
- ***uint32_t TwoSamplingDelay***

Field Documentation

- ***uint32_t ADC_MultiModeTypeDef::Mode***
  Configures the ADC to operate in independent or multimode. This parameter can be a value of ***ADC_HAL_EC_MULTI_MODE***.
- ***uint32_t ADC_MultiModeTypeDef::DualModeData***
  Configures the Dual ADC Mode Data Format: This parameter can be a value of ***ADCEx_Dual_Mode_Data_Format***
- ***uint32_t ADC_MultiModeTypeDef::TwoSamplingDelay***
  Configures the Delay between 2 sampling phases. This parameter can be a value of ***ADC_HAL_EC_MULTI_TWOSMP_DELAY***

8.2 ADCEx Firmware driver API description

The following section lists the various functions of the ADCEx library.

8.2.1 Detailed description of functions

HAL_ADCEx_Calibration_Start

Function name

HAL_StatusTypeDef HAL_ADCEx_Calibration_Start (ADC_HandleTypeDef * hadc, uint32_t SingleDiff)

Function description

HAL_ADCEx_Calibration_GetValue

Function name

uint32_t HAL_ADCEx_Calibration_GetValue (const ADC_HandleTypeDef * hadc, uint32_t SingleDiff)

Function description

HAL_ADCEx_Calibration_SetValue

Function name

HAL_StatusTypeDef HAL_ADCEx_Calibration_SetValue (ADC_HandleTypeDef * hadc, uint32_t SingleDiff, uint32_t CalibrationFactor)

Function description

HAL_ADCEx_InjectedStart

Function name

HAL_StatusTypeDef HAL_ADCEx_InjectedStart (ADC_HandleTypeDef * hadc)

Function description

HAL_ADCEx_InjectedStop

Function name

HAL_StatusTypeDef HAL_ADCEx_InjectedStop (ADC_HandleTypeDef * hadc)

Function description

HAL_ADCEx_InjectedPollForConversion

Function name

HAL_StatusTypeDef HAL_ADCEx_InjectedPollForConversion (ADC_HandleTypeDef * hadc, uint32_t Timeout)

Function description

HAL_ADCEx_InjectedStart_IT

Function name

HAL_StatusTypeDef HAL_ADCEx_InjectedStart_IT (ADC_HandleTypeDef * hadc)

Function description

HAL_ADCEx_InjectedStop_IT

Function name

HAL_StatusTypeDef HAL_ADCEx_InjectedStop_IT (ADC_HandleTypeDef * hadc)

Function description

HAL_ADCEx_MultiModeStart_DMA

Function name

HAL_StatusTypeDef HAL_ADCEx_MultiModeStart_DMA (ADC_HandleTypeDef * hadc, uint32_t * pData, uint32_t Length)

Function description

HAL_ADCEx_MultiModeStop_DMA

Function name

HAL_StatusTypeDef HAL_ADCEx_MultiModeStop_DMA (ADC_HandleTypeDef * hadc)

Function description

HAL_ADCEx_MultiModeGetValue

Function name

uint32_t HAL_ADCEx_MultiModeGetValue (const ADC_HandleTypeDef * hadc)

Function description

HAL_ADCEx_InjectedGetValue

Function name

uint32_t HAL_ADCEx_InjectedGetValue (const ADC_HandleTypeDef * hadc, uint32_t InjectedRank)

Function description

HAL_ADCEx_InjectedGetSignedValue

Function name

int32_t HAL_ADCEx_InjectedGetSignedValue (const ADC_HandleTypeDef * hadc, uint32_t InjectedRank)

Function description

HAL_ADCEx_InjectedConvCpltCallback

Function name

void HAL_ADCEx_InjectedConvCpltCallback (ADC_HandleTypeDef * hadc)

Function description

HAL_ADCEx_LevelOutOfWindow2Callback

Function name

void HAL_ADCEx_LevelOutOfWindow2Callback (ADC_HandleTypeDef * hadc)

Function description

HAL_ADCEx_LevelOutOfWindow3Callback

Function name

void HAL_ADCEx_LevelOutOfWindow3Callback (ADC_HandleTypeDef * hadc)

Function description

HAL_ADCEx_EndOfSamplingCallback

Function name

void HAL_ADCEx_EndOfSamplingCallback (ADC_HandleTypeDef * hadc)

Function description

HAL_ADCEx_RegularStop

Function name

HAL_StatusTypeDef HAL_ADCEx_RegularStop (ADC_HandleTypeDef * hadc)

Function description

HAL_ADCEx_RegularStop_IT

Function name

HAL_StatusTypeDef HAL_ADCEx_RegularStop_IT (ADC_HandleTypeDef * hadc)

Function description

HAL_ADCEx_RegularStop_DMA

Function name

HAL_StatusTypeDef HAL_ADCEx_RegularStop_DMA (ADC_HandleTypeDef * hadc)

Function description

HAL_ADCEx_RegularMultiModeStop_DMA

Function name

HAL_StatusTypeDef HAL_ADCEx_RegularMultiModeStop_DMA (ADC_HandleTypeDef * hadc)

Function description

HAL_ADCEx_InjectedConfigChannel

Function name

HAL_StatusTypeDef HAL_ADCEx_InjectedConfigChannel (ADC_HandleTypeDef * hadc, const ADC_InjectionConfTypeDef * pConfigInjected)

Function description

HAL_ADCEx_MultiModeConfigChannel

Function name

HAL_StatusTypeDef HAL_ADCEx_MultiModeConfigChannel (ADC_HandleTypeDef * hadc, const ADC_MultiModeTypeDef * pMultimode)

Function description

HAL_ADCEx_EnterADCDeepPowerDownMode

Function name

HAL_StatusTypeDef HAL_ADCEx_EnterADCDeepPowerDownMode (ADC_HandleTypeDef * hadc)

Function description

8.3 ADCEx Firmware driver defines

The following section lists the various define and macros of the module.

8.3.1 ADCEx

ADCEx

ADC Extended Dual Mode Data Formatting

ADC_DUALMODEDATAFORMAT_DISABLED

ADC multimode group regular data format: conversion data in data register of each ADC instance. If ADC data transfer by DMA is used: each ADC uses its own DMA channel, with its individual DMA transfer settings.

ADC_DUALMODEDATAFORMAT_32_BIT

ADC multimode group regular data format: conversion data in two ADC common instance data registers (CDR, CDR2) with packing option on 32 bit. In register CDR, data packing on 32 bit: ADC master and slave data are concatenated (data master in [15; 0], data slave in [31; 16]), therefore data width must be lower than 16 bit (even with ADC resolution 12 bit, higher width reachable by post processing: oversampling, offset, ...). In register CDR2, data of master and slave are alternatively set in full register width 32 bit, therefore no constraint on data width. In case of usage with DMA, CDR generate ones transfer request and CDR2 two transfer requests per conversion.

ADC_DUALMODEDATAFORMAT_8_BIT

ADC multimode group regular data format: conversion data in two ADC common instance data registers (CDR, CDR2) with packing option on 16 bit. In register CDR, data packing on 16 bit: ADC master and slave data are concatenated (data master in [7; 0], data slave in [15; 8]), therefore data width must be lower than 8 bit (even with ADC resolution 8 bit, higher width reachable by post processing: oversampling, offset, ...). In register CDR2, data of master and slave are alternatively set in full register width 32 bit, therefore no constraint on data width. In case of usage with DMA, CDR generate ones transfer request and CDR2 two transfer requests per conversion.

ADC_DUALMODEDATAFORMAT_32_10_BITS

ADC_DUALMODEDATAFORMAT_8_BITS

ADC Extended Exported Macros

ADC_FORCE_MODE_INDEPENDENT

Description:

- Force ADC instance in multimode mode independent (multimode disable).

Parameters:

- __HANDLE__: ADC handle.

Return value:

- None

Notes:

- This macro must be used only in case of transition from multimode to mode independent and in case of unknown previous state, to ensure ADC configuration is in mode independent. Standard way of multimode configuration change is done from HAL ADC handle of ADC master using function "HAL_ADCEx_MultiModeConfigChannel(..., ADC_MODE_INDEPENDENT)")". Usage of this macro is not the Standard way of multimode configuration and can lead to have HAL ADC handles status misaligned. Usage of this macro must be limited to cases mentioned above.

__HAL_ADC_CHANNEL_DIFF_NEG_INPUT

Description:

- Helper macro to determine the selected channel corresponding negative input on the ADC instance selected.

Parameters:

- __HANDLE__: ADC handle.
- __CHANNEL__: This parameter can be one of the following values:
  - ADC_CHANNEL_0
  - ADC_CHANNEL_1
  - ADC_CHANNEL_2
  - ADC_CHANNEL_3
  - ADC_CHANNEL_4
  - ADC_CHANNEL_5
  - ADC_CHANNEL_6
  - ADC_CHANNEL_7
  - ADC_CHANNEL_8
  - ADC_CHANNEL_9
  - ADC_CHANNEL_10
  - ADC_CHANNEL_11
  - ADC_CHANNEL_12
  - ADC_CHANNEL_13
  - ADC_CHANNEL_14
  - ADC_CHANNEL_15
  - ADC_CHANNEL_16
  - ADC_CHANNEL_17
  - ADC_CHANNEL_18
  - ADC_CHANNEL_19
  - ADC_CHANNEL_VREFINT (1)(4)
  - ADC_CHANNEL_VBAT (2)(4)
  - ADC_CHANNEL_VDDCORE (2)(4)

Return value:

- Returned: value can be one of the following values:
  - ADC_CHANNEL_0
  - ADC_CHANNEL_1
  - ADC_CHANNEL_2
  - ADC_CHANNEL_3
  - ADC_CHANNEL_4
  - ADC_CHANNEL_5
  - ADC_CHANNEL_6
  - ADC_CHANNEL_7
  - ADC_CHANNEL_8
  - ADC_CHANNEL_9
  - ADC_CHANNEL_10
  - ADC_CHANNEL_11
  - ADC_CHANNEL_12
  - ADC_CHANNEL_13
  - ADC_CHANNEL_14
  - ADC_CHANNEL_15
  - ADC_CHANNEL_16
  - ADC_CHANNEL_17
  - ADC_CHANNEL_18
  - ADC_CHANNEL_19
  - ADC_CHANNEL_VREFINT (1)(3)
  - ADC_CHANNEL_VBAT (2)(3)
  - ADC_CHANNEL_VDDCORE (2)(3)

ADC Extended Oversampling left Shift

ADC_LEFTBITSHIFT_NONE

ADC No bit shift

ADC_LEFTBITSHIFT_1

ADC 1 bit shift

ADC_LEFTBITSHIFT_2

ADC 2 bits shift

ADC_LEFTBITSHIFT_3

ADC 3 bits shift

ADC_LEFTBITSHIFT_4

ADC 4 bits shift

ADC_LEFTBITSHIFT_5

ADC 5 bits shift

ADC_LEFTBITSHIFT_6

ADC 6 bits shift

ADC_LEFTBITSHIFT_7

ADC 7 bits shift

ADC_LEFTBITSHIFT_8

ADC 8 bits shift

ADC_LEFTBITSHIFT_9

ADC 9 bits shift

ADC_LEFTBITSHIFT_10

ADC 10 bits shift

ADC_LEFTBITSHIFT_11

ADC 11 bits shift

ADC_LEFTBITSHIFT_12

ADC 12 bits shift

ADC_LEFTBITSHIFT_13

ADC 13 bits shift

ADC_LEFTBITSHIFT_14

ADC 14 bits shift

ADC_LEFTBITSHIFT_15

ADC 15 bits shift

ADC Extended Offset Sign

ADC_OFFSET_SIGN_NEGATIVE

Offset sign negative, offset is subtracted

ADC_OFFSET_SIGN_POSITIVE

Offset sign positive, offset is added

9 HAL BSEC Generic Driver

9.1 BSEC Firmware driver registers structures

9.1.1 BSEC_HandleTypeDef

BSEC_HandleTypeDef is defined in the stm32n6xx_hal_bsec.h

Data Fields

- ***BSEC_TypeDef * Instance***
- ***uint32_t ErrorCode***

Field Documentation

- ***BSEC_TypeDef* BSEC_HandleTypeDef::Instance***
  BSEC registers base address
- ***uint32_t BSEC_HandleTypeDef::ErrorCode***
  Error code in case of HAL driver internal error

9.1.2 BSEC_ScratchRegTypeDef

BSEC_ScratchRegTypeDef is defined in the stm32n6xx_hal_bsec.h

Data Fields

- ***uint32_t RegType***
- ***uint32_t RegNumber***

Field Documentation

- ***uint32_t BSEC_ScratchRegTypeDef::RegType***
  Type of the register This parameter can be a value of ***BSEC_Register_Type***
- ***uint32_t BSEC_ScratchRegTypeDef::RegNumber***
  Number of the register This parameter can be a value between 0 and 3 if RegType is HAL_BSEC_SCRATCH_REG and between 0 and 7 if RegType is HAL_BSEC_WRITE_ONCE_SCRATCH_REG

9.1.3 BSEC_DebugCfgTypeDef

BSEC_DebugCfgTypeDef is defined in the stm32n6xx_hal_bsec.h

Data Fields

- ***uint32_t HDPL_Open_Dbg***
- ***uint32_t Sec_Dbg_Auth***
- ***uint32_t NonSec_Dbg_Auth***

Field Documentation

- ***uint32_t BSEC_DebugCfgTypeDef::HDPL_Open_Dbg***
  Level at which debug may be opened This parameter can be a value of ***BSEC_Open_Dbg***
- ***uint32_t BSEC_DebugCfgTypeDef::Sec_Dbg_Auth***
  Secure debug authorization This parameter can be a value of ***BSEC_Sec_Dbg_Auth***
- ***uint32_t BSEC_DebugCfgTypeDef::NonSec_Dbg_Auth***
  Non-secure debug authorization This parameter can be a value of ***BSEC_NonSec_Dbg_Auth***

9.2 BSEC Firmware driver API description

The following section lists the various functions of the BSEC library.

9.2.1 How to use this driver

General Configuration

- Call the function HAL_BSEC_GetStatus() to get the status of BSEC peripheral.

- Call the function HAL_BSEC_GlobalLock() to lock the write to BSEC registers.
- Call the function HAL_BSEC_GetGlobalLockStatus() to get the global write registers lock status.
- Call the function HAL_BSEC_GetErrorCode() to get the error code raised when API return HAL_ERROR.

OTP Management

- Call the function HAL_BSEC_OTP_Read() to read the value of an OTP fuse.
- Call the function HAL_BSEC_OTP_Program() to program an OTP fuse with or without permanent lock.
- Call the function HAL_BSEC_OTP_Reload() to reload an OTP fuse value.
- Call the function HAL_BSEC_OTP_Lock() to sticky lock an OTP fuse programming, writing shadowed register or reload.
- Call the function HAL_BSEC_OTP_GetState() to get the sticky lock status (programming, writing shadowed register or reload), the permanent lock status, the shadow configuration, the ECC or hidden status of an OTP fuse.

Shadow fuse register management

- Call the function HAL_BSEC_OTP_ReadShadow() to read the value of a shadow fuse register.
- Call the function HAL_BSEC_OTP_WriteShadow() to write in a shadow fuse register.
- Call the function HAL_BSEC_OTP_GetShadowState() to get the validity status of a shadow fuse register.

Device lifecycle management

- Call the function HAL_BSEC_GetDeviceLifeCycleState() to get the BSEC device lifecycle state.
- Call the function HAL_BSEC_ReadEpochCounter() to read the value of epoch counter.
- Call the function HAL_BSEC_SelectEpochCounter() to select the epoch counter used by SAES.
- Call the function HAL_BSEC_GetEpochCounterSelection() to get the epoch counter selection for SAES.

HDPL management

- Call the function HAL_BSEC_GetHDPLValue() to get the current HDPL.
- Call the function HAL_BSEC_IncrementHDPLValue() to increment the HDPL.
- Call the function HAL_BSEC_ConfigSAESHDPLIncrementValue() to configure the increment of HDPL sent to SAES.
- Call the function HAL_BSEC_GetSAESHDPLIncrementValue() to get the SAES increment to HDPL.

Scratch register management

- Call the function HAL_BSEC_WriteScratchValue() to write a value in write once scratch or scratch register.
- Call the function HAL_BSEC_ReadScratchValue() to read a value from write once scratch or scratch register.

Debug management

- Call the function HAL_BSEC_GetDebugRequest() to get the debug request of host debugger.
- Call the function HAL_BSEC_SendJTAGData() to send data via JTAG.
- Call the function HAL_BSEC_ReceiveJTAGData() to receive data via JTAG.
- Call the function HAL_BSEC_ConfigDebug() to configure the HDPL, secure and non-secure authorization for debugger.
- Call the function HAL_BSEC_GetDebugConfig() to get the HDPL, secure and non-secure authorization of debug.
- Call the function HAL_BSEC_LockDebug() to lock the debug.
- Call the function HAL_BSEC_UnlockDebug() to unlock the debug.
- Call the function HAL_BSEC_GetDebugLockState() to get the debug lock status.

DHUK management

- Call the function HAL_BSEC_GetDHUKValidity() to get the validity of DHUK.
- Call the function HAL_BSEC_LockDHUKUse() to lock the DHUK use.

- Call the function HAL_BSEC_GetDHUKLockStatus() to get the DHUK lock status.

Reset counter

- Call the function HAL_BSEC_GetNumberOfResets() to get the number of hot or warm resets.

9.2.2 Detailed description of functions

HAL_BSEC_GetStatus

Function name

HAL_StatusTypeDef HAL_BSEC_GetStatus (BSEC_HandleTypeDef * hbsec, uint32_t * pStatus)

Function description

HAL_BSEC_GlobalLock

Function name

HAL_StatusTypeDef HAL_BSEC_GlobalLock (BSEC_HandleTypeDef * hbsec)

Function description

HAL_BSEC_GetGlobalLockStatus

Function name

HAL_StatusTypeDef HAL_BSEC_GetGlobalLockStatus (BSEC_HandleTypeDef * hbsec, uint32_t * pStatus)

Function description

HAL_BSEC_GetErrorCode

Function name

HAL_StatusTypeDef HAL_BSEC_GetErrorCode (BSEC_HandleTypeDef * hbsec, uint32_t * pError)

Function description

HAL_BSEC_OTP_Read

Function name

HAL_StatusTypeDef HAL_BSEC_OTP_Read (BSEC_HandleTypeDef * hbsec, uint32_t FuseId, uint32_t * pFuseData)

Function description

HAL_BSEC_OTP_Program

Function name

HAL_StatusTypeDef HAL_BSEC_OTP_Program (BSEC_HandleTypeDef * hbsec, uint32_t FuseId, uint32_t FuseData, uint32_t Lock)

Function description

HAL_BSEC_OTP_Reload

Function name

HAL_StatusTypeDef HAL_BSEC_OTP_Reload (BSEC_HandleTypeDef * hbsec, uint32_t FuseId)

Function description

HAL_BSEC_OTP_Lock

Function name

HAL_StatusTypeDef HAL_BSEC_OTP_Lock (BSEC_HandleTypeDef * hbsec, uint32_t FuseId, uint32_t Lock)

Function description

HAL_BSEC_OTP_GetState

Function name

HAL_StatusTypeDef HAL_BSEC_OTP_GetState (BSEC_HandleTypeDef * hbsec, uint32_t FuseId, uint32_t * pState)

Function description

HAL_BSEC_OTP_ReadShadow

Function name

HAL_StatusTypeDef HAL_BSEC_OTP_ReadShadow (BSEC_HandleTypeDef * hbsec, uint32_t RegId, uint32_t * pRegData)

Function description

HAL_BSEC_OTP_WriteShadow

Function name

HAL_StatusTypeDef HAL_BSEC_OTP_WriteShadow (BSEC_HandleTypeDef * hbsec, uint32_t RegId, uint32_t RegData)

Function description

HAL_BSEC_OTP_GetShadowState

Function name

HAL_StatusTypeDef HAL_BSEC_OTP_GetShadowState (BSEC_HandleTypeDef * hbsec, uint32_t RegId, uint32_t * pValidity)

Function description

HAL_BSEC_GetDeviceLifeCycleState

Function name

HAL_StatusTypeDef HAL_BSEC_GetDeviceLifeCycleState (BSEC_HandleTypeDef * hbsec, uint32_t * pState)

Function description

HAL_BSEC_ReadEpochCounter

Function name

HAL_StatusTypeDef HAL_BSEC_ReadEpochCounter (BSEC_HandleTypeDef * hbsec, uint32_t CounterId, uint32_t * pCounterData)

Function description

HAL_BSEC_SelectEpochCounter

Function name

HAL_StatusTypeDef HAL_BSEC_SelectEpochCounter (BSEC_HandleTypeDef * hbsec, uint32_t SelectedCounter)

Function description

HAL_BSEC_GetEpochCounterSelection

Function name

HAL_StatusTypeDef HAL_BSEC_GetEpochCounterSelection (BSEC_HandleTypeDef * hbsec, uint32_t * pSelectedCounter)

Function description

HAL_BSEC_GetHDPLValue

Function name

HAL_StatusTypeDef HAL_BSEC_GetHDPLValue (BSEC_HandleTypeDef * hbsec, uint32_t * pHDPL)

Function description

HAL_BSEC_IncrementHDPLValue

Function name

HAL_StatusTypeDef HAL_BSEC_IncrementHDPLValue (BSEC_HandleTypeDef * hbsec)

Function description

HAL_BSEC_ConfigSAESHDPLIncrementValue

Function name

HAL_StatusTypeDef HAL_BSEC_ConfigSAESHDPLIncrementValue (BSEC_HandleTypeDef * hbsec, uint32_t Increment)

Function description

HAL_BSEC_GetSAESHDPLIncrementValue

Function name

HAL_StatusTypeDef HAL_BSEC_GetSAESHDPLIncrementValue (BSEC_HandleTypeDef * hbsec, uint32_t * pIncrement)

Function description

HAL_BSEC_WriteScratchValue

Function name

HAL_StatusTypeDef HAL_BSEC_WriteScratchValue (BSEC_HandleTypeDef * hbsec, const BSEC_ScratchRegTypeDef * pRegAddr, uint32_t Value)

Function description

HAL_BSEC_ReadScratchValue

Function name

HAL_StatusTypeDef HAL_BSEC_ReadScratchValue (BSEC_HandleTypeDef * hbsec, const BSEC_ScratchRegTypeDef * pRegAddr, uint32_t * pValue)

Function description

HAL_BSEC_GetDebugRequest

Function name

HAL_StatusTypeDef HAL_BSEC_GetDebugRequest (BSEC_HandleTypeDef * hbsec, uint32_t * pDbgReq)

Function description

HAL_BSEC_SendJTAGData

Function name

HAL_StatusTypeDef HAL_BSEC_SendJTAGData (BSEC_HandleTypeDef * hbsec, uint32_t Data)

Function description

HAL_BSEC_ReceiveJTAGData

Function name

HAL_StatusTypeDef HAL_BSEC_ReceiveJTAGData (BSEC_HandleTypeDef * hbsec, uint32_t * pData)

Function description

HAL_BSEC_ConfigDebug

Function name

HAL_StatusTypeDef HAL_BSEC_ConfigDebug (BSEC_HandleTypeDef * hbsec, const BSEC_DebugCfgTypeDef * pCfg)

Function description

HAL_BSEC_GetDebugConfig

Function name

HAL_StatusTypeDef HAL_BSEC_GetDebugConfig (BSEC_HandleTypeDef * hbsec, BSEC_DebugCfgTypeDef * pDbgCfg)

Function description

HAL_BSEC_LockDebug

Function name

HAL_StatusTypeDef HAL_BSEC_LockDebug (BSEC_HandleTypeDef * hbsec)

Function description

HAL_BSEC_UnlockDebug

Function name

HAL_StatusTypeDef HAL_BSEC_UnlockDebug (BSEC_HandleTypeDef * hbsec)

Function description

HAL_BSEC_GetDebugLockState

Function name

HAL_StatusTypeDef HAL_BSEC_GetDebugLockState (BSEC_HandleTypeDef * hbsec, uint32_t * pStatus)

Function description

HAL_BSEC_GetDHUKValidity

Function name

HAL_StatusTypeDef HAL_BSEC_GetDHUKValidity (BSEC_HandleTypeDef * hbsec, uint32_t * pValidity)

Function description

HAL_BSEC_LockDHUKUse

Function name

HAL_StatusTypeDef HAL_BSEC_LockDHUKUse (BSEC_HandleTypeDef * hbsec)

Function description

HAL_BSEC_GetDHUKLockStatus

Function name

HAL_StatusTypeDef HAL_BSEC_GetDHUKLockStatus (BSEC_HandleTypeDef * hbsec, uint32_t * pStatus)

Function description

HAL_BSEC_GetNumberOfResets

Function name

HAL_StatusTypeDef HAL_BSEC_GetNumberOfResets (BSEC_HandleTypeDef * hbsec, uint32_t ResetType, uint32_t * pResetNumber)

Function description

9.3 BSEC Firmware driver defines

The following section lists the various define and macros of the module.

9.3.1 BSEC

BSEC

BSEC Debug lock

HAL_BSEC_DEBUG_LOCKED

DBG_MCU AP interface locked

HAL_BSEC_DEBUG_UNLOCKED

DBG_MCU AP interface unlocked

BSEC Debug request

HAL_BSEC_DBG_NOT_REQUESTED

Host debugger is not requesting debug

HAL_BSEC_DBG_REQUESTED

Host debugger is requesting debug

BSEC Derived hardware unique key lock

HAL_BSEC_DHUK_USABLE

DHUK in SAES is usable

HAL_BSEC_DHUK_NOT_USABLE

DHUK in SAES is not usable

BSEC Derived hardware unique key validity

HAL_BSEC_DHUK_INVALID

DHUK can't be used in SAES

HAL_BSEC_DHUK_VALID

DHUK can be used in SAES

BSEC Epoch selected register

HAL_BSEC_EPOCH_COUNTER_0

Epoch counter 0

HAL_BSEC_EPOCH_COUNTER_1

Epoch counter 1

BSEC Error code

HAL_BSEC_ERROR_NONE

No error

HAL_BSEC_ERROR_INVALID_PARAM

Parameter error

HAL_BSEC_ERROR_UNDEFINED_VALUE

Undefined value error

HAL_BSEC_ERROR_LOCK

Locked operation error

HAL_BSEC_ERROR_UNALLOWED

Not possible operation error

HAL_BSEC_ERROR_TIMEOUT

Timeout error

HAL_BSEC_ERROR_PROGFAIL

Programming failed

HAL_BSEC_ERROR_DISTURB

Disturb flag (unexpected error)

HAL_BSEC_ERROR_DED

Double error detection flag

HAL_BSEC_ERROR_SEC

Single error detection flag

HAL_BSEC_ERROR_PPL

Permanent programming lock flag

HAL_BSEC_ERROR_PPLM

Permanent programming lock mismatch flag

HAL_BSEC_ERROR_AME

Addresses mismatch error flag

BSEC Global write lock

HAL_BSEC_WRITE_REG_ALLOWED

Write to BSEC registers are allowed

HAL_BSEC_WRITE_REG_IGNORED

Write to BSEC registers are ignored

BSEC Hide protection level

HAL_BSEC_HDPL_VALUE_0

Level 0

HAL_BSEC_HDPL_VALUE_1

Level 1

HAL_BSEC_HDPL_VALUE_2

Level 2

HAL_BSEC_HDPL_VALUE_3

Level 3

BSEC Increment of Hide protection level sent to SAES

HAL_BSEC_HDPL_INCR_VALUE_0

HDPL sent to SAES incremented by 0

HAL_BSEC_HDPL_INCR_VALUE_1

HDPL sent to SAES incremented by 1

HAL_BSEC_HDPL_INCR_VALUE_2

HDPL sent to SAES incremented by 2

HAL_BSEC_HDPL_INCR_VALUE_3

HDPL sent to SAES incremented by 3

BSEC Device lifecycle state

HAL_BSEC_OPEN_STATE

BSEC is in open state

HAL_BSEC_CLOSED_STATE

BSEC is in closed state

HAL_BSEC_INVALID_STATE

BSEC is in invalid state

HAL_BSEC_INVALID_STATE_WITH_TAMPER

BSEC is in invalid state with an active confirmed tamper triggered

BSEC Non-secure debug authorization

HAL_BSEC_NONSEC_DBG_UNAUTH

Non-secure debug not authorized

HAL_BSEC_NONSEC_DBG_AUTH

Non-secure debug authorized

BSEC Open debug level

HAL_BSEC_OPEN_DBG_LEVEL_0

Debug opened for HDPL0

HAL_BSEC_OPEN_DBG_LEVEL_1

Debug opened for HDPL1

HAL_BSEC_OPEN_DBG_LEVEL_2

Debug opened for HDPL2

HAL_BSEC_OPEN_DBG_LEVEL_3

Debug opened for HDPL3

BSEC Permanent lock

HAL_BSEC_NORMAL_PROG

Fuse word is programmed normally

HAL_BSEC_LOCK_PROG

Fuse word is locked, preventing further modifications

BSEC Type of scratch register

HAL_BSEC_SCRATCH_REG

Type of register is scratch register

HAL_BSEC_WRITE_ONCE_SCRATCH_REG

Type of register is write once scratch register

BSEC Validity of shadow register reload

HAL_BSEC_RELOAD_ERROR

An error occurred while fuse word was last reloaded

HAL_BSEC_RELOAD_DONE

Last reload of fuse word was done without error

BSEC Type of reset

HAL_BSEC_HOT_RESET

Type of reset is hot reset

HAL_BSEC_WARM_RESET

Type of reset is warm reset

BSEC Secure debug authorization

HAL_BSEC_SEC_DBG_UNAUTH

Secure debug not authorized

HAL_BSEC_SEC_DBG_AUTH

Secure debug authorized

BSEC State

HAL_BSEC_FUSE_PROG_LOCKED

Fuse word programming is silently ignored until next reset

HAL_BSEC_FUSE_WRITE_LOCKED

Shadow register writing is silently ignored until next reset

HAL_BSEC_FUSE_RELOAD_LOCKED

Fuse word loading is denied until next reset

HAL_BSEC_FUSE_LOCKED

Fuse word is locked, preventing further modifications

HAL_BSEC_FUSE_SHADOWED

Fuse word is shadowed

HAL_BSEC_FUSE_HIDDEN

Fuse word is hidden

HAL_BSEC_FUSE_ERROR_DETECTED

At least one error is detected during reset operations

HAL_BSEC_FUSE_ERROR_CORRECTED

At least one single-error correction is detected during reset operations

BSEC Peripheral status

HAL_BSEC_INIT_DONE

BSEC initialized following a cold or warm reset

HAL_BSEC_UPPER_FUSES_HIDDEN

Upper fuses are not accessible to the application

HAL_BSEC_OTP_NOT_VIRGIN

BSEC is not virgin

10 HAL CACHEAXI Generic Driver

10.1 CACHEAXI Firmware driver registers structures

10.1.1 __CACHEAXI_HandleTypeDef

__CACHEAXI_HandleTypeDef is defined in the stm32n6xx_hal_cacheaxi.h

Data Fields

- ***CACHEAXI_TypeDef * Instance***
- ***__IO HAL_CACHEAXI_StateTypeDef State***
- ***__IO uint32_t ErrorCode***
- ***void(* ErrorCallback***
- ***void(* CleanByAddrCallback***
- ***void(* InvalidateCompleteCallback***
- ***void(* CleanAndInvalidateByAddrCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***CACHEAXI_TypeDef* __CACHEAXI_HandleTypeDef::Instance***
  CACHEAXI register base address.
- ***__IO HAL_CACHEAXI_StateTypeDef __CACHEAXI_HandleTypeDef::State***
- ***__IO uint32_t __CACHEAXI_HandleTypeDef::ErrorCode***
- ***void(* __CACHEAXI_HandleTypeDef::ErrorCallback)(struct __CACHEAXI_HandleTypeDef *hcacheaxi)***
- ***void(* __CACHEAXI_HandleTypeDef::CleanByAddrCallback)(struct __CACHEAXI_HandleTypeDef *hcacheaxi)***
- ***void(* __CACHEAXI_HandleTypeDef::InvalidateCompleteCallback)(struct __CACHEAXI_HandleTypeDef *hcacheaxi)***
- ***void(* __CACHEAXI_HandleTypeDef::CleanAndInvalidateByAddrCallback)(struct __CACHEAXI_HandleTypeDef *hcacheaxi)***
- ***void(* __CACHEAXI_HandleTypeDef::MspInitCallback)(struct __CACHEAXI_HandleTypeDef *hcacheaxi)***
- ***void(* __CACHEAXI_HandleTypeDef::MspDeInitCallback)(struct __CACHEAXI_HandleTypeDef *hcacheaxi)***

10.2 CACHEAXI Firmware driver API description

The following section lists the various functions of the CACHEAXI library.

10.2.1 How to use this driver

1. Configure and enable the MPU to override default config if needed, please refers to ARM manual for default memory attribute. Then enable CACHEAXI with HAL_CACHEAXI_Init() to cache memories data when accessed by a master peripheral through an AXI interconnect.

- Use HAL_CACHEAXI_Invalidate() to invalidate the full cache content:
  - Cache content is lost, and reloaded when needed.
  - Blocking call until operation is done.
- Use HAL_CACHEAXI_CleanByAddr() to clean cache content for a specific memory range:
  - Cache content for specific range is written back to specific memory range.
  - Used when buffer is updated by CPU before usage by a peripheral (typically DMA transfer)
  - Blocking call until operation is done.

- Use HAL_CACHEAXI_CleanInvalidateByAddr() to clean and invalidate cache content for a specific memory range:
  - Cache content for specific range is written back to specific memory range, and reloaded when needed.
  - Recommended to use for MPU reprogramming.
  - Blocking call until operation is done.

Interrupt mode IO operation

- Configure the CACHEAXI interrupt priority using HAL_NVIC_SetPriority()
- Enable the CACHEAXI IRQ handler using HAL_NVIC_EnableIRQ()
- Override weak definition for following callbacks (if needed):
  - HAL_CACHEAXI_CleanAndInvalidateByAddrCallback()
  - HAL_CACHEAXI_InvalidateCompleteCallback()
  - HAL_CACHEAXI_CleanByAddrCallback()
  - HAL_CACHEAXI_ErrorCallback()
- Use HAL_CACHEAXI_<COMMAND>_IT() to start a CACHEAXI operation with IT enabled.
- Use HAL_CACHEAXI_IRQHandler() called under CACHEAXIx_IRQHandler() Interrupt subroutine

Use HAL_CACHEAXI_GetState() function to return the CACHEAXI state and HAL_CACHEAXI_GetError() in case of error detection.

CACHEAXI HAL driver macros list

Below the list of macros defined in the CACHEAXI HAL driver.

- __HAL_CACHEAXI_ENABLE_IT : Enable CACHEAXI interrupts.
- __HAL_CACHEAXI_DISABLE_IT : Disable CACHEAXI interrupts.
- __HAL_CACHEAXI_GET_IT_SOURCE: Check whether the specified CACHEAXI interrupt source is enabled or not.
- __HAL_CACHEAXI_GET_FLAG : Check whether the selected CACHEAXI flag is set or not.
- __HAL_CACHEAXI_CLEAR_FLAG : Clear the selected CACHEAXI flags.

Note: You can refer to the header file of the CACHEAXI HAL driver for more useful macros.

10.2.2 Detailed description of functions

HAL_CACHEAXI_Init

Function name

HAL_StatusTypeDef HAL_CACHEAXI_Init (CACHEAXI_HandleTypeDef * hcacheaxi)

Function description

HAL_CACHEAXI_DeInit

Function name

HAL_StatusTypeDef HAL_CACHEAXI_DeInit (CACHEAXI_HandleTypeDef * hcacheaxi)

Function description

HAL_CACHEAXI_MspInit

Function name

void HAL_CACHEAXI_MspInit (CACHEAXI_HandleTypeDef * hcacheaxi)

Function description

HAL_CACHEAXI_MspDeInit

Function name

void HAL_CACHEAXI_MspDeInit (CACHEAXI_HandleTypeDef * hcacheaxi)

Function description

HAL_CACHEAXI_Enable

Function name

HAL_StatusTypeDef HAL_CACHEAXI_Enable (CACHEAXI_HandleTypeDef * hcacheaxi)

Function description

HAL_CACHEAXI_Disable

Function name

HAL_StatusTypeDef HAL_CACHEAXI_Disable (CACHEAXI_HandleTypeDef * hcacheaxi)

Function description

HAL_CACHEAXI_IsEnabled

Function name

uint32_t HAL_CACHEAXI_IsEnabled (const CACHEAXI_HandleTypeDef * hcacheaxi)

Function description

HAL_CACHEAXI_Invalidate

Function name

HAL_StatusTypeDef HAL_CACHEAXI_Invalidate (CACHEAXI_HandleTypeDef * hcacheaxi)

Function description

HAL_CACHEAXI_CleanByAddr

Function name

HAL_StatusTypeDef HAL_CACHEAXI_CleanByAddr (CACHEAXI_HandleTypeDef * hcacheaxi, const uint32_t *const pAddr, uint32_t dSize)

Function description

HAL_CACHEAXI_CleanInvalidByAddr

Function name

HAL_StatusTypeDef HAL_CACHEAXI_CleanInvalidByAddr (CACHEAXI_HandleTypeDef * hcacheaxi, const uint32_t *const pAddr, uint32_t dSize)

Function description

HAL_CACHEAXI_Invalidate_IT

Function name

HAL_StatusTypeDef HAL_CACHEAXI_Invalidate_IT (CACHEAXI_HandleTypeDef * hcacheaxi)

Function description

HAL_CACHEAXI_CleanByAddr_IT

Function name

HAL_StatusTypeDef HAL_CACHEAXI_CleanByAddr_IT (CACHEAXI_HandleTypeDef * hcacheaxi, const uint32_t *const pAddr, uint32_t dSize)

Function description

HAL_CACHEAXI_CleanInvalidByAddr_IT

Function name

HAL_StatusTypeDef HAL_CACHEAXI_CleanInvalidByAddr_IT (CACHEAXI_HandleTypeDef * hcacheaxi, const uint32_t *const pAddr, uint32_t dSize)

Function description

HAL_CACHEAXI_IRQHandler

Function name

void HAL_CACHEAXI_IRQHandler (CACHEAXI_HandleTypeDef * hcacheaxi)

Function description

HAL_CACHEAXI_ErrorCallback

Function name

void HAL_CACHEAXI_ErrorCallback (CACHEAXI_HandleTypeDef * hcacheaxi)

Function description

HAL_CACHEAXI_CleanByAddrCallback

Function name

void HAL_CACHEAXI_CleanByAddrCallback (CACHEAXI_HandleTypeDef * hcacheaxi)

Function description

HAL_CACHEAXI_InvalidateCompleteCallback

Function name

void HAL_CACHEAXI_InvalidateCompleteCallback (CACHEAXI_HandleTypeDef * hcacheaxi)

Function description

HAL_CACHEAXI_CleanAndInvalidateByAddrCallback

Function name

void HAL_CACHEAXI_CleanAndInvalidateByAddrCallback (CACHEAXI_HandleTypeDef * hcacheaxi)

Function description

HAL_CACHEAXI_RegisterCallback

Function name

HAL_StatusTypeDef HAL_CACHEAXI_RegisterCallback (CACHEAXI_HandleTypeDef * hcacheaxi, HAL_CACHEAXI_CallbackIDTypeDef CallbackID, pCACHEAXI_CallbackTypeDef pCallback)

Function description

HAL_CACHEAXI_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_CACHEAXI_UnRegisterCallback (CACHEAXI_HandleTypeDef * hcacheaxi, HAL_CACHEAXI_CallbackIDTypeDef CallbackID)

Function description

HAL_CACHEAXI_Monitor_GetReadHitValue

Function name

uint32_t HAL_CACHEAXI_Monitor_GetReadHitValue (const CACHEAXI_HandleTypeDef * hcacheaxi)

Function description

HAL_CACHEAXI_Monitor_GetReadMissValue

Function name

uint32_t HAL_CACHEAXI_Monitor_GetReadMissValue (const CACHEAXI_HandleTypeDef * hcacheaxi)

Function description

HAL_CACHEAXI_Monitor_GetWriteHitValue

Function name

uint32_t HAL_CACHEAXI_Monitor_GetWriteHitValue (const CACHEAXI_HandleTypeDef * hcacheaxi)

Function description

HAL_CACHEAXI_Monitor_GetWriteMissValue

Function name

uint32_t HAL_CACHEAXI_Monitor_GetWriteMissValue (const CACHEAXI_HandleTypeDef * hcacheaxi)

Function description

HAL_CACHEAXI_Monitor_GetReadAllocMissValue

Function name

uint32_t HAL_CACHEAXI_Monitor_GetReadAllocMissValue (const CACHEAXI_HandleTypeDef * hcacheaxi)

Function description

HAL_CACHEAXI_Monitor_GetWriteAllocMissValue

Function name

uint32_t HAL_CACHEAXI_Monitor_GetWriteAllocMissValue (const CACHEAXI_HandleTypeDef * hcacheaxi)

Function description

HAL_CACHEAXI_Monitor_GetWriteThroughValue

Function name

uint32_t HAL_CACHEAXI_Monitor_GetWriteThroughValue (const CACHEAXI_HandleTypeDef * hcacheaxi)

Function description

HAL_CACHEAXI_Monitor_GetEvictionValue

Function name

uint32_t HAL_CACHEAXI_Monitor_GetEvictionValue (const CACHEAXI_HandleTypeDef * hcacheaxi)

Function description

HAL_CACHEAXI_Monitor_Reset

Function name

HAL_StatusTypeDef HAL_CACHEAXI_Monitor_Reset (CACHEAXI_HandleTypeDef * hcacheaxi, uint32_t MonitorType)

Function description

HAL_CACHEAXI_Monitor_Start

Function name

HAL_StatusTypeDef HAL_CACHEAXI_Monitor_Start (CACHEAXI_HandleTypeDef * hcacheaxi, uint32_t MonitorType)

Function description

HAL_CACHEAXI_Monitor_Stop

Function name

HAL_StatusTypeDef HAL_CACHEAXI_Monitor_Stop (CACHEAXI_HandleTypeDef * hcacheaxi, uint32_t MonitorType)

Function description

HAL_CACHEAXI_GetState

Function name

HAL_CACHEAXI_StateTypeDef HAL_CACHEAXI_GetState (const CACHEAXI_HandleTypeDef * hcacheaxi)

Function description

HAL_CACHEAXI_GetError

Function name

uint32_t HAL_CACHEAXI_GetError (const CACHEAXI_HandleTypeDef * hcacheaxi)

Function description

10.3 CACHEAXI Firmware driver defines

The following section lists the various define and macros of the module.

10.3.1 CACHEAXI

CACHEAXI

Command Operation

LL_CACHEAXI_COMMAND_NO_OPERATION

LL_CACHEAXI_COMMAND_CLEAN_BY_ADDR

LL_CACHEAXI_COMMAND_CLEAN_INVALIDATE_BY_ADDR

CACHEAXI Error Code

HAL_CACHEAXI_ERROR_NONE

No error

HAL_CACHEAXI_ERROR_TIMEOUT

Timeout error

HAL_CACHEAXI_ERROR_EVICTION_CLEAN

Eviction or clean operation write-back error

HAL_CACHEAXI_ERROR_INVALID_OPERATION

Invalid operation

HAL_CACHEAXI_ERROR_INVALID_CALLBACK

Invalid callback error

CACHEAXI Exported Macros

__HAL_CACHEAXI_ENABLE_IT

Description:

- Enable CACHEAXI interrupts.

Parameters:

- __HANDLE__: specifies the CACHEAXI handle.
- __INTERRUPT__: specifies the CACHEAXI interrupt sources to be enabled. This parameter can be any combination of the following values:
  - CACHEAXI_IT_BUSYEND Busy end interrupt
  - CACHEAXI_IT_ERROR Cache error interrupt
  - CACHEAXI_IT_CMDEND Cache Command end interrupt

Return value:

- None

__HAL_CACHEAXI_DISABLE_IT

Description:

- Disable CACHEAXI interrupts.

Parameters:

- __HANDLE__: specifies the CACHEAXI handle.
- __INTERRUPT__: specifies the CACHEAXI interrupt sources to be disabled. This parameter can be any combination of the following values:
  - CACHEAXI_IT_BUSYEND Busy end interrupt
  - CACHEAXI_IT_ERROR Cache error interrupt
  - CACHEAXI_IT_CMDEND Cache Command end interrupt

Return value:

- None

__HAL_CACHEAXI_GET_IT_SOURCE

Description:

- Check whether the specified CACHEAXI interrupt source is enabled or not.

Parameters:

- __HANDLE__: specifies the CACHEAXI handle.
- __INTERRUPT__: specifies the CACHEAXI interrupt source to check. This parameter can be any combination of the following values:
  - CACHEAXI_IT_BUSYEND Busy end interrupt
  - CACHEAXI_IT_ERROR Cache error interrupt
  - CACHEAXI_IT_CMDEND Cache Command end interrupt

Return value:

- The: state of __INTERRUPT__ (SET or RESET).

__HAL_CACHEAXI_GET_FLAG

Description:

- Check whether the selected CACHEAXI flag is set or not.

Parameters:

- __HANDLE__: specifies the CACHEAXI handle.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - CACHEAXI_FLAG_BUSY Busy flag
  - CACHEAXI_FLAG_BUSYEND Busy end flag
  - CACHEAXI_FLAG_ERROR Cache error flag
  - CACHEAXI_FLAG_BUSYCMD Cache Busy command flag
  - CACHEAXI_FLAG_CMDEND Cache command end flag

Return value:

- The: state of __FLAG__ (0 or 1).

__HAL_CACHEAXI_CLEAR_FLAG

Description:

- Clear the selected CACHEAXI flags.

Parameters:

- __HANDLE__: specifies the CACHEAXI handle.
- __FLAG__: specifies the CACHEAXI flags to clear. This parameter can be any combination of the following values:
  - CACHEAXI_FLAG_BUSYEND Busy end flag
  - CACHEAXI_FLAG_ERROR Cache error flag
  - CACHEAXI_FLAG_CMDEND Cache command end flag

Flags

CACHEAXI_FLAG_BUSY

Busy flag

CACHEAXI_FLAG_BUSYEND

Busy end flag

CACHEAXI_FLAG_ERROR

Cache error flag

CACHEAXI_FLAG_BUSYCMD

Busy command flag

CACHEAXI_FLAG_CMDEND

Command end flag

Interrupts

CACHEAXI_IT_BUSYEND

Busy end interrupt

CACHEAXI_IT_ERROR

Cache error interrupt

CACHEAXI_IT_CMDEND

Command end interrupt

Monitor type

CACHEAXI_MONITOR_READ_HIT

Read Hit monitoring

CACHEAXI_MONITOR_READ_MISS

Read Miss monitoring

CACHEAXI_MONITOR_WRITE_HIT

Write Hit monitoring

CACHEAXI_MONITOR_WRITE_MISS

Write Miss monitoring

CACHEAXI_MONITOR_READALLOC_MISS

Read-allocate Miss monitoring

CACHEAXI_MONITOR_WRITEALLOC_MISS

Write-allocate Miss monitoring

CACHEAXI_MONITOR_WRITETHROUGH

Write-through monitoring

CACHEAXI_MONITOR_EVICTION

Eviction monitoring

CACHEAXI_MONITOR_ALL

LL_CACHEAXI_MONITOR_READ_HIT

Read Hit monitoring

LL_CACHEAXI_MONITOR_READ_MISS

Read Miss monitoring

LL_CACHEAXI_MONITOR_WRITE_HIT

Write Hit monitoring

LL_CACHEAXI_MONITOR_WRITE_MISS

Write Miss monitoring

LL_CACHEAXI_MONITOR_READALLOC_MISS

Read-alloc Miss monitoring

LL_CACHEAXI_MONITOR_WRITEALLOC_MISS

Write-alloc Miss monitoring

LL_CACHEAXI_MONITOR_WRITETHROUGH

Write-through monitoring

LL_CACHEAXI_MONITOR_EVICTION

Eviction monitoring

LL_CACHEAXI_MONITOR_ALL

11 HAL CORTEX Generic Driver

11.1 CORTEX Firmware driver registers structures

11.1.1 MPU_Region_InitTypeDef

MPU_Region_InitTypeDef is defined in the stm32n6xx_hal_cortex.h

Data Fields

- ***uint8_t Enable***
- ***uint8_t Number***
- ***uint32_t BaseAddress***
- ***uint32_t LimitAddress***
- ***uint8_t AttributesIndex***
- ***uint8_t AccessPermission***
- ***uint8_t DisableExec***
- ***uint8_t DisablePrivExec***
- ***uint8_t IsShareable***

Field Documentation

- ***uint8_t MPU_Region_InitTypeDef::Enable***
  Specifies the status of the region. This parameter can be a value of ***CORTEX_MPU_Region_Enable***
- ***uint8_t MPU_Region_InitTypeDef::Number***
  Specifies the number of the region to protect. This parameter can be a value of ***CORTEX_MPU_Region_Number***
- ***uint32_t MPU_Region_InitTypeDef::BaseAddress***
  Specifies the base address of the region to protect.
- ***uint32_t MPU_Region_InitTypeDef::LimitAddress***
  Specifies the limit address of the region to protect.
- ***uint8_t MPU_Region_InitTypeDef::AttributesIndex***
  Specifies the memory attributes index. This parameter can be a value of ***CORTEX_MPU_Attributes_Number***
- ***uint8_t MPU_Region_InitTypeDef::AccessPermission***
  Specifies the region access permission type. This parameter can be a value of ***CORTEX_MPU_Region_Permission_Attributes***
- ***uint8_t MPU_Region_InitTypeDef::DisableExec***
  Specifies the instruction access status. This parameter can be a value of ***CORTEX_MPU_Instruction_Access***
- ***uint8_t MPU_Region_InitTypeDef::DisablePrivExec***
  Specifies if privileged software can execute instructions from this region. This parameter can be a value of ***CORTEX_MPU_Priv_Instruction_Access***
- ***uint8_t MPU_Region_InitTypeDef::IsShareable***
  Specifies the shareability status of the protected region. This parameter can be a value of ***CORTEX_MPU_Access_Shareable***

11.1.2 MPU_Attributes_InitTypeDef

MPU_Attributes_InitTypeDef is defined in the stm32n6xx_hal_cortex.h

Data Fields

- ***uint8_t Number***
- ***uint8_t Attributes***

Field Documentation

- ***uint8_t MPU_Attributes_InitTypeDef::Number***
  Specifies the number of the memory attributes to configure. This parameter can be a value of ***CORTEX_MPU_Attributes_Number***

- ***uint8_t MPU_Attributes_InitTypeDef::Attributes***
  Specifies the memory attributes vue. This parameter must be a number between Min_Data = 0x0 and Max_Data = 0xFFFF

11.2 CORTEX Firmware driver API description

The following section lists the various functions of the CORTEX library.

11.2.1 How to use this driver

How to configure Interrupts using CORTEX HAL driver

This section provides functions allowing to configure the NVIC interrupts (IRQ). The Cortex-M55 exceptions are managed by CMSIS functions.

1. Configure the NVIC Priority Grouping using HAL_NVIC_SetPriorityGrouping() function.
2. Configure the priority of the selected IRQ Channels using HAL_NVIC_SetPriority().
3. Enable the selected IRQ Channels using HAL_NVIC_EnableIRQ().

Note: When the NVIC_PRIORITYGROUP_0 is selected, IRQ pre-emption is no more possible. The pending IRQ priority will be managed only by the sub priority.

Note: IRQ priority order (sorted by highest to lowest priority):

- *Lowest pre-emption priority*
- *Lowest sub priority*
- *Lowest hardware priority (IRQ number)*

How to configure SysTick using CORTEX HAL driver

Setup SysTick Timer for time base.

- The HAL_SYSTICK_Config() function calls the SysTick_Config() function which is a CMSIS function that:
  - Configures the SysTick Reload register with value passed as function parameter.
  - Configures the SysTick IRQ priority to the lowest value (0x0F).
  - Resets the SysTick Counter register.
  - Configures the SysTick Counter clock source to be Core Clock Source (HCLK).
  - Enables the SysTick Interrupt.
  - Starts the SysTick Counter.
- You can change the SysTick Clock source to be HCLK_Div8 by calling the macro __HAL_CORTEX_SYSTICKCLK_CONFIG(SYSTICK_CLKSOURCE_HCLK_DIV8) just after the HAL_SYSTICK_Config() function call. The __HAL_CORTEX_SYSTICKCLK_CONFIG() macro is defined inside the stm32n6xx_hal_cortex.h file.
- You can change the SysTick IRQ priority by calling the HAL_NVIC_SetPriority(SysTick_IRQn,...) function just after the HAL_SYSTICK_Config() function call. The HAL_NVIC_SetPriority() call the NVIC_SetPriority() function which is a CMSIS function.
- To adjust the SysTick time base, use the following formula: Reload Value = SysTick Counter Clock (Hz) x Desired Time base (s)
  - Reload Value is the parameter to be passed for HAL_SYSTICK_Config() function
  - Reload Value should not exceed 0xFFFFFF

How to configure MPU (secure and non secure) using CORTEX HAL driver

This section provides functions allowing to configure both secure and non-secure MPUs.

1. Enable the MPU using HAL_MPU_Enable() function.
2. Disable the MPU using HAL_MPU_Disable() function.
3. Enable the MPU using HAL_MPU_Enable_NS() function to address the non secure MPU.
4. Disable the MPU using HAL_MPU_Disable_NS() function to address the non secure MPU.
5. Configure the MPU region using HAL_MPU_ConfigRegion() and HAL_MPU_ConfigRegion_NS() to address the non secure MPU.

6. Configure the MPU Memory attributes using HAL_MPU_ConfigMemoryAttributes() and HAL_MPU_ConfigMemoryAttributes_NS() to address the non secure MPU. A device memory has the following attributes:
7. G or nG: gathering or non-gathering. (multiple accesses to a device can be merged into a single transaction except for operations with memory ordering semantics, for example, memory barrier instructions, load acquire/store release).
8. R or nR: reordering
9. E or nE: early write acknowledge (similar to bufferable) Only four combinations of these attributes are valid:
10. device-nGnRnE: equivalent to Armv7-M strongly ordered memory type
11. device-nGnRE: equivalent to Armv7-M device memory
12. device-nGRE: new to Armv8-M
13. device-GRE: new to Armv8-M A normal memory has the following attributes:
14. Cache Allocation attribute : set when a cache line is allocated (no allocation, read/write/read-write allocation)
15. Cache write policy : write through (write to cache AND memory), write back (memory is written when the cache line is evicted)
16. Transient : indicates that the region will be used for a short period of time For normal memory, attributes can be set for inner and outer caches separately. Note that outer attributes set to 0 change the memory to device mode. Both inner and outer attributes should be set for normal memory. Sample configurations
17. Inner-outer cacheable, write back, read-write allocate INNER_OUTER(MPU_RW_ALLOCATE | MPU_WRITE_BACK)
18. Inner write back, read allocation, outer non-cacheable (MPU_R_ALLOCATE | MPU_WRITE_BACK) | OUTER(MPU_NOT_CACHEABLE) For detail on memory attributes, refer to the ARMv8-m MPU documentation. On STM32N6xx, the MPUs are split memory into regions (up to sixteen for both Secure and Non-Secure domains) The secure MPU is only available when TrustZone is activated.
19. Enable the MPU using HAL_MPU_Enable() function or HAL_MPU_Enable_NS function for non-secure MPU.
20. Disable the MPU using HAL_MPU_Disable() function or HAL_MPU_Disable_NS function for non-secure MPU.
21. Enable the MPU region using HAL_MPU_EnableRegion() function or HAL_MPU_EnableRegion_NS function for non-secure MPU region.
22. Disable the MPU region using HAL_MPU_DisableRegion() function or HAL_MPU_DisableRegion_NS function for non-secure MPU region.
23. Configure the MPU region using HAL_MPU_ConfigRegion() function or HAL_MPU_ConfigRegion_NS function for non-secure MPU.
24. Configure the MPU memory attributes using HAL_MPU_ConfigMemoryAttributes() function or HAL_MPU_ConfigMemoryAttributes_NS function for non-secure MPU.
25. The HAL_MPU_XXX_NS functions are only available when TrustZone is activated and CPU in secure state. _NS functions are targeting non secure MPU, in any other cases APIs without NS shall be used.

11.2.2 Initialization and Configuration functions

This section provides the CORTEX HAL driver functions allowing to configure Interrupts SysTick functionalities

This section contains the following APIs:

- ***HAL_NVIC_SetPriorityGrouping()***
- ***HAL_NVIC_SetPriority()***
- ***HAL_NVIC_EnableIRQ()***
- ***HAL_NVIC_DisableIRQ()***
- ***HAL_NVIC_SystemReset()***
- ***HAL_SYSTICK_Config()***

11.2.3 Peripheral Control functions

This subsection provides a set of functions allowing to control the CORTEX (NVIC, SYSTICK, MPU) functionalities.

This section contains the following APIs:

- ***HAL_NVIC_GetPriorityGrouping()***
- ***HAL_NVIC_GetPriority()***

- *HAL_NVIC_SetPendingIRQ()*
- *HAL_NVIC_GetPendingIRQ()*
- *HAL_NVIC_ClearPendingIRQ()*
- *HAL_NVIC_GetActive()*
- *HAL_SYSTICK_CLKSourceConfig()*
- *HAL_SYSTICK_IRQHandler()*
- *HAL_SYSTICK_Callback()*
- *HAL_CORTEX_ClearEvent()*
- *HAL_MPU_Enable()*
- *Section 11.2.4: HAL_MPU_Enable_NS()*
- *HAL_MPU_Disable()*
- *Section 11.2.4: HAL_MPU_Disable_NS()*
- *HAL_MPU_EnableRegion()*
- *Section 11.2.4: HAL_MPU_EnableRegion_NS()*
- *HAL_MPU_DisableRegion()*
- *Section 11.2.4: HAL_MPU_DisableRegion_NS()*
- *HAL_MPU_ConfigRegion()*
- *Section 11.2.4: HAL_MPU_ConfigRegion_NS()*
- *HAL_MPU_ConfigMemoryAttributes()*
- *Section 11.2.4: HAL_MPU_ConfigMemoryAttributes_NS()*

11.2.4 Detailed description of functions

HAL_NVIC_SetPriorityGrouping

Function name

void HAL_NVIC_SetPriorityGrouping (uint32_t PriorityGroup)

Function description

Set the priority grouping field (pre-emption priority and subpriority) using the required unlock sequence.

Parameters

- **PriorityGroup:** The priority grouping bits length. This parameter can be one of the following values:
  - NVIC_PRIORITYGROUP_0 0 bit for pre-emption priority, 4 bits for subpriority
  - NVIC_PRIORITYGROUP_1 1 bit for pre-emption priority, 3 bits for subpriority
  - NVIC_PRIORITYGROUP_2 2 bits for pre-emption priority, 2 bits for subpriority
  - NVIC_PRIORITYGROUP_3 3 bits for pre-emption priority, 1 bit for subpriority
  - NVIC_PRIORITYGROUP_4 4 bits for pre-emption priority, 0 bit for subpriority

Return values

- **None:**

Notes

- When the NVIC_PriorityGroup_0 is selected, IRQ pre-emption is no more possible. The pending IRQ priority will be managed only by the subpriority.

HAL_NVIC_SetPriority

Function name

void HAL_NVIC_SetPriority (IRQn_Type IRQn, uint32_t PreemptPriority, uint32_t SubPriority)

Function description

Set the priority of an interrupt.

Parameters

- **IRQn:** External interrupt number. This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32n6xxxx.h))
- **PreemptPriority:** The pre-emption priority for the IRQn channel. This parameter can be a value between 0 and 15 A lower priority value indicates a higher priority
- **SubPriority:** the subpriority level for the IRQ channel. This parameter can be a value between 0 and 15 A lower priority value indicates a higher priority.

Return values

- **None:**

HAL_NVIC_EnableIRQ

Function name

void HAL_NVIC_EnableIRQ (IRQn_Type IRQn)

Function description

Enable a device specific interrupt in the NVIC interrupt controller.

Parameters

- **IRQn:** External interrupt number. This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32n6xxxx.h))

Return values

- **None:**

Notes

- To configure interrupts priority correctly, the NVIC_PriorityGroupConfig() function should be called before.

HAL_NVIC_DisableIRQ

Function name

void HAL_NVIC_DisableIRQ (IRQn_Type IRQn)

Function description

Disable a device specific interrupt in the NVIC interrupt controller.

Parameters

- **IRQn:** External interrupt number. This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32n6xxxx.h))

Return values

- **None:**

HAL_NVIC_SystemReset

Function name

void HAL_NVIC_SystemReset (void)

Function description

Initiate a system reset request to reset the MCU.

Return values

- **None:**

HAL_SYSTICK_Config

Function name

uint32_t HAL_SYSTICK_Config (uint32_t TicksNumb)

Function description

Initialize the System Timer with interrupt enabled and start the System Tick Timer (SysTick): Counter is in free running mode to generate periodic interrupts.

Parameters

- **TicksNumb:** Specifies the ticks Number of ticks between two interrupts.

Return values

- **status:** - 0 Function succeeded.
  - 1 Function failed.

HAL_NVIC_GetPriorityGrouping

Function name

uint32_t HAL_NVIC_GetPriorityGrouping (void)

Function description

Get the priority grouping field from the NVIC Interrupt Controller.

Return values

- **Priority:** grouping field (SCB->AIRCR [10:8] PRIGROUP field)

HAL_NVIC_GetPriority

Function name

void HAL_NVIC_GetPriority (IRQn_Type IRQn, uint32_t PriorityGroup, uint32_t * pPreemptPriority, uint32_t * pSubPriority)

Function description

Get the priority of an interrupt.

Parameters

- **IRQn:** External interrupt number. This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32n6xxxx.h))
- **PriorityGroup:** the priority grouping bits length. This parameter can be one of the following values:
  - NVIC_PRIORITYGROUP_0 0 bit for pre-emption priority, 4 bits for subpriority
  - NVIC_PRIORITYGROUP_1 1 bit for pre-emption priority, 3 bits for subpriority
  - NVIC_PRIORITYGROUP_2 2 bits for pre-emption priority, 2 bits for subpriority
  - NVIC_PRIORITYGROUP_3 3 bits for pre-emption priority, 1 bit for subpriority
  - NVIC_PRIORITYGROUP_4 4 bits for pre-emption priority, 0 bit for subpriority
- **pPreemptPriority:** Pointer on the Preemptive priority value (starting from 0).
- **pSubPriority:** Pointer on the Subpriority value (starting from 0).

Return values

- **None:**

HAL_NVIC_GetPendingIRQ

Function name

uint32_t HAL_NVIC_GetPendingIRQ (IRQn_Type IRQn)

Function description

Get Pending Interrupt (read the pending register in the NVIC and return the pending bit for the specified interrupt).

Parameters

- **IRQn:** External interrupt number. This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32n6xxxx.h))

Return values

- **status:** - 0 Interrupt status is not pending.
  - 1 Interrupt status is pending.

HAL_NVIC_SetPendingIRQ

Function name

void HAL_NVIC_SetPendingIRQ (IRQn_Type IRQn)

Function description

Set Pending bit of an external interrupt.

Parameters

- **IRQn:** External interrupt number This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32n6xxxx.h))

Return values

- **None:**

HAL_NVIC_ClearPendingIRQ

Function name

void HAL_NVIC_ClearPendingIRQ (IRQn_Type IRQn)

Function description

Clear the pending bit of an external interrupt.

Parameters

- **IRQn:** External interrupt number. This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32n6xxxx.h))

Return values

- **None:**

HAL_NVIC_GetActive

Function name

uint32_t HAL_NVIC_GetActive (IRQn_Type IRQn)

Function description

Get active interrupt (read the active register in NVIC and return the active bit).

Parameters

- **IRQn:** External interrupt number This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32n6xxxx.h))

Return values

- **status:** - 0 Interrupt status is not pending.
  - 1 Interrupt status is pending.

HAL_SYSTICK_CLKSourceConfig

Function name

void HAL_SYSTICK_CLKSourceConfig (uint32_t CLKSource)

Function description

Configure the SysTick clock source.

Parameters

- **CLKSource:** specifies the SysTick clock source. This parameter can be one of the following values:
  - SYSTICK_CLKSOURCE_HCLK AHB clock selected as SysTick clock source.
  - SYSTICK_CLKSOURCE_HCLK_DIV8 AHB clock divided by 8 selected as SysTick clock source.

Return values

- **None:**

HAL_SYSTICK_IRQHandler

Function name

void HAL_SYSTICK_IRQHandler (void)

Function description

Handle SYSTICK interrupt request.

Return values

- **None:**

HAL_SYSTICK_Callback

Function name

void HAL_SYSTICK_Callback (void)

Function description

SYSTICK callback.

Return values

- **None:**

HAL_CORTEX_ClearEvent

Function name

void HAL_CORTEX_ClearEvent (void)

Function description

Clear pending event(s).

Return values

- **None:**

HAL_MPU_Enable

Function name

void HAL_MPU_Enable (uint32_t MPU_Control)

Function description

Enable the MPU.

Parameters

- **MPU_Control:** Specifies the control mode of the MPU during hard fault, NMI, FAULTMASK and privileged access to the default memory This parameter can be one of the following values:
  - MPU_HFNMI_PRIVDEF_NONE
  - MPU_HARDFAULT_NMI
  - MPU_PRIVILEGED_DEFAULT
  - MPU_HFNMI_PRIVDEF

Return values

- **None:**

HAL_MPU_Disable

Function name

void HAL_MPU_Disable (void)

Function description

Disable the MPU.

Return values

- **None:**

HAL_MPU_EnableRegion

Function name

void HAL_MPU_EnableRegion (uint32_t RegionNumber)

Function description

Enable the MPU Region.

Return values

- **None:**

HAL_MPU_DisableRegion

Function name

void HAL_MPU_DisableRegion (uint32_t RegionNumber)

Function description

Disable the MPU Region.

Return values

- **None:**

HAL_MPU_ConfigRegion

Function name

void HAL_MPU_ConfigRegion (const MPU_Region_InitTypeDef * pMPU_RegionInit)

Function description

Initialize and configure the Region and the memory to be protected.

Parameters

- **pMPU_RegionInit:** Pointer to a MPU_Region_InitTypeDef structure that contains the initialization and configuration information.

Return values

- **None:**

HAL_MPU_ConfigMemoryAttributes

Function name

void HAL_MPU_ConfigMemoryAttributes (const MPU_Attributes_InitTypeDef * pMPU_AttributesInit)

Function description

Initialize and configure the memory attributes.

Parameters

- **pMPU_AttributesInit:** Pointer to a MPU_Attributes_InitTypeDef structure that contains the initialization and configuration information.

Return values

- **None:**

11.3 CORTEX Firmware driver defines

The following section lists the various define and macros of the module.

11.3.1 CORTEX

CORTEX

CORTEX MPU Instruction Access Shareable

MPU_ACCESS_NOT_SHAREABLE

MPU_ACCESS_OUTER_SHAREABLE

MPU_ACCESS_INNER_SHAREABLE

CORTEX MPU Attributes

MPU_DEVICE_NGNRNE

MPU_DEVICE_NGNRE

MPU_DEVICE_NGRE

MPU_DEVICE_GRE

MPU_NOT_CACHEABLE

MPU_WRITE_THROUGH

MPU_WRITE_BACK

MPU_TRANSIENT

MPU_NON_TRANSIENT

MPU_NO_ALLOCATE

MPU_W_ALLOCATE

MPU_R_ALLOCATE

MPU_RW_ALLOCATE

CORTEX MPU Memory Attributes Number

MPU_ATTRIBUTES_NUMBER0

MPU_ATTRIBUTES_NUMBER1

MPU_ATTRIBUTES_NUMBER2

MPU_ATTRIBUTES_NUMBER3

MPU_ATTRIBUTES_NUMBER4

MPU_ATTRIBUTES_NUMBER5

MPU_ATTRIBUTES_NUMBER6

MPU_ATTRIBUTES_NUMBER7

CORTEX MPU HFNMI and PRIVILEGED Access control

MPU_HFNMI_PRIVDEF_NONE

MPU_HARDFAULT_NMI

MPU_PRIVILEGED_DEFAULT

MPU_HFNMI_PRIVDEF

CORTEX MPU Instruction Access

MPU_INSTRUCTION_ACCESS_ENABLE

MPU_INSTRUCTION_ACCESS_DISABLE

CORTEX MPU Normal Memory Attributes

OUTER

INNER_OUTER

CORTEX MPU Privileged Instruction Access

MPU_PRIV_INSTRUCTION_ACCESS_ENABLE

MPU_PRIV_INSTRUCTION_ACCESS_DISABLE

CORTEX MPU Region Enable

MPU_REGION_ENABLE

MPU_REGION_DISABLE

CORTEX MPU Region Number

MPU_REGION_NUMBER0

MPU_REGION_NUMBER1

MPU_REGION_NUMBER2

MPU_REGION_NUMBER3

MPU_REGION_NUMBER4

MPU_REGION_NUMBER5

MPU_REGION_NUMBER6

MPU_REGION_NUMBER7

MPU_REGION_NUMBER8

MPU_REGION_NUMBER9

MPU_REGION_NUMBER10

MPU_REGION_NUMBER11

MPU_REGION_NUMBER12

MPU_REGION_NUMBER13

MPU_REGION_NUMBER14

MPU_REGION_NUMBER15

CORTEX MPU Region Permission Attributes

MPU_REGION_PRIV_RW

MPU_REGION_ALL_RW

MPU_REGION_PRIV_RO

MPU_REGION_ALL_RO

CORTEX Preemption Priority Group

NVIC_PRIORITYGROUP_0

0 bit for pre-emption priority, 4 bits for subpriority

NVIC_PRIORITYGROUP_1

1 bit for pre-emption priority, 3 bits for subpriority

NVIC_PRIORITYGROUP_2

2 bits for pre-emption priority, 2 bits for subpriority

NVIC_PRIORITYGROUP_3

3 bits for pre-emption priority, 1 bit for subpriority

NVIC_PRIORITYGROUP_4

4 bits for pre-emption priority, 0 bit for subpriority

CORTEX SysTick clock source

SYSTICK_CLKSOURCE_HCLK_DIV8

SYSTICK_CLKSOURCE_HCLK

12 HAL CRC Generic Driver

12.1 CRC Firmware driver registers structures

12.1.1 CRC_InitTypeDef

CRC_InitTypeDef is defined in the stm32n6xx_hal_crc.h

Data Fields

- ***uint8_t DefaultPolynomialUse***
- ***uint8_t DefaultInitValueUse***
- ***uint32_t GeneratingPolynomial***
- ***uint32_t CRCLength***
- ***uint32_t InitValue***
- ***uint32_t InputDataInversionMode***
- ***uint32_t OutputDataInversionMode***

Field Documentation

- ***uint8_t CRC_InitTypeDef::DefaultPolynomialUse***
  This parameter is a value of ***CRC_Default_Polynomial*** and indicates if default polynomial is used. If set to DEFAULT_POLYNOMIAL_ENABLE, resort to default X^32 + X^26 + X^23 + X^22 + X^16 + X^12 + X^11 + X^10 +X^8 + X^7 + X^5 + X^4 + X^2+ X +1. In that case, there is no need to set GeneratingPolynomial field. If otherwise set to DEFAULT_POLYNOMIAL_DISABLE, GeneratingPolynomial and CRCLength fields must be set.
- ***uint8_t CRC_InitTypeDef::DefaultInitValueUse***
  This parameter is a value of ***CRC_Default_InitValue_Use*** and indicates if default init value is used. If set to DEFAULT_INIT_VALUE_ENABLE, resort to default 0xFFFFFFFF value. In that case, there is no need to set InitValue field. If otherwise set to DEFAULT_INIT_VALUE_DISABLE, InitValue field must be set.
- ***uint32_t CRC_InitTypeDef::GeneratingPolynomial***
  Set CRC generating polynomial as a 7, 8, 16 or 32-bit long value for a polynomial degree respectively equal to 7, 8, 16 or 32. This field is written in normal, representation e.g., for a polynomial of degree 7, X^7 + X^6 + X^5 + X^2 + 1 is written 0x65. No need to specify it if DefaultPolynomialUse is set to DEFAULT_POLYNOMIAL_ENABLE.
- ***uint32_t CRC_InitTypeDef::CRCLength***
  This parameter is a value of ***CRC_Polynomial_Sizes*** and indicates CRC length. Value can be either one of
  - **CRC_POLYLENGTH_32B** (32-bit CRC),
  - **CRC_POLYLENGTH_16B** (16-bit CRC),
  - **CRC_POLYLENGTH_8B** (8-bit CRC),
  - **CRC_POLYLENGTH_7B** (7-bit CRC).
- ***uint32_t CRC_InitTypeDef::InitValue***
  Init value to initiate CRC computation. No need to specify it if DefaultInitValueUse is set to DEFAULT_INIT_VALUE_ENABLE.
- ***uint32_t CRC_InitTypeDef::InputDataInversionMode***
  This parameter is a value of ***CRCEx_Input_Data_Inversion*** and specifies input data inversion mode. Can be either one of the following values
  - **CRC_INPUTDATA_INVERSION_NONE** no input data inversion
  - **CRC_INPUTDATA_INVERSION_BYTE** byte-wise inversion, 0x1A2B3C4D becomes 0x58D43CB2
  - **CRC_INPUTDATA_INVERSION_HALFWORD** halfword-wise inversion, 0x1A2B3C4D becomes 0xD458B23C
  - **CRC_INPUTDATA_INVERSION_WORD** word-wise inversion, 0x1A2B3C4D becomes 0xB23CD458

- ***uint32_t CRC_InitTypeDef::OutputDataInversionMode***
  This parameter is a value of ***CRCEx_Output_Data_Inversion*** and specifies output data (i.e. CRC) inversion mode. Can be either
  - **CRC_OUTPUTDATA_INVERSION_DISABLE** no CRC inversion,
  - **CRC_OUTPUTDATA_INVERSION_ENABLE** CRC 0x11223344 is converted into 0x22CC4488

12.1.2 CRC_HandleTypeDef

CRC_HandleTypeDef is defined in the stm32n6xx_hal_crc.h

Data Fields

- ***CRC_TypeDef * Instance***
- ***CRC_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_CRC_StateTypeDef State***
- ***uint32_t InputDataFormat***

Field Documentation

- ***CRC_TypeDef* CRC_HandleTypeDef::Instance***
  Register base address
- ***CRC_InitTypeDef CRC_HandleTypeDef::Init***
  CRC configuration parameters
- ***HAL_LockTypeDef CRC_HandleTypeDef::Lock***
  CRC Locking object
- ***__IO HAL_CRC_StateTypeDef CRC_HandleTypeDef::State***
  CRC communication state
- ***uint32_t CRC_HandleTypeDef::InputDataFormat***
  This parameter is a value of ***CRC_Input_Buffer_Format*** and specifies input data format. Can be either
  - **CRC_INPUTDATA_FORMAT_BYTES** input data is a stream of bytes (8-bit data)
  - **CRC_INPUTDATA_FORMAT_HALFWORDS** input data is a stream of half-words (16-bit data)
  - **CRC_INPUTDATA_FORMAT_WORDS** input data is a stream of words (32-bit data)

  Note that constant CRC_INPUT_FORMAT_UNDEFINED is defined but an initialization error must occur if InputBufferFormat is not one of the three values listed above

12.2 CRC Firmware driver API description

The following section lists the various functions of the CRC library.

12.2.1 How to use this driver

- Enable CRC AHB clock using __HAL_RCC_CRC_CLK_ENABLE();
- Initialize CRC calculator
  - specify generating polynomial (peripheral default or non-default one)
  - specify initialization value (peripheral default or non-default one)
  - specify input data format
  - specify input or output data inversion mode if any
- Use HAL_CRC_Accumulate() function to compute the CRC value of the input data buffer starting with the previously computed CRC as initialization value
- Use HAL_CRC_Calculate() function to compute the CRC value of the input data buffer starting with the defined initialization value (default or non-default) to initiate CRC calculation

12.2.2 Detailed description of functions

HAL_CRC_Init

Function name

HAL_StatusTypeDef HAL_CRC_Init (CRC_HandleTypeDef * hcrc)

Function description

HAL_CRC_DeInit

Function name

HAL_StatusTypeDef HAL_CRC_DeInit (CRC_HandleTypeDef * hcrc)

Function description

HAL_CRC_MspInit

Function name

void HAL_CRC_MspInit (CRC_HandleTypeDef * hcrc)

Function description

HAL_CRC_MspDeInit

Function name

void HAL_CRC_MspDeInit (CRC_HandleTypeDef * hcrc)

Function description

HAL_CRC_Accumulate

Function name

uint32_t HAL_CRC_Accumulate (CRC_HandleTypeDef * hcrc, uint32_t pBuffer, uint32_t BufferLength)

Function description

HAL_CRC_Calculate

Function name

uint32_t HAL_CRC_Calculate (CRC_HandleTypeDef * hcrc, uint32_t pBuffer, uint32_t BufferLength)

Function description

HAL_CRC_GetState

Function name

HAL_CRC_StateTypeDef HAL_CRC_GetState (const CRC_HandleTypeDef * hcrc)

Function description

12.3 CRC Firmware driver defines

The following section lists the various define and macros of the module.

12.3.1 CRC

CRC

Default CRC computation initialization value

DEFAULT_CRC_INITVALUE

Initial CRC default value

Indicates whether or not default init value is used

DEFAULT_INIT_VALUE_ENABLE

Enable initial CRC default value

DEFAULT_INIT_VALUE_DISABLE

Disable initial CRC default value

Indicates whether or not default polynomial is used

DEFAULT_POLYNOMIAL_ENABLE

Enable default generating polynomial 0x04C11DB7

DEFAULT_POLYNOMIAL_DISABLE

Disable default generating polynomial 0x04C11DB7

Default CRC generating polynomial

DEFAULT_CRC32_POLY

X^32 + X^26 + X^23 + X^22 + X^16 + X^12 + X^11 + X^10 +X^8 + X^7 + X^5 + X^4 + X^2+ X +1

CRC Exported Macros

__HAL_CRC_RESET_HANDLE_STATE

Description:

- Reset CRC handle state.

Parameters:

- __HANDLE__: CRC handle.

Return value:

- None

__HAL_CRC_DR_RESET

Description:

- Reset CRC Data Register.

Parameters:

- __HANDLE__: CRC handle

Return value:

- None

__HAL_CRC_INITIALCRCVALUE_CONFIG

Description:

- Set CRC INIT non-default value.

Parameters:

- __HANDLE__: CRC handle
- __INIT__: 32-bit initial value

Return value:

- None

__HAL_CRC_SET_IDR

Description:

- Store data in the Independent Data (ID) register.

Parameters:

- __HANDLE__: CRC handle
- __VALUE__: Value to be stored in the ID register

Return value:

- None

Notes:

- Refer to the Reference Manual to get the authorized __VALUE__ length in bits

__HAL_CRC_GET_IDR

Description:

- Return the data stored in the Independent Data (ID) register.

Parameters:

- __HANDLE__: CRC handle

Return value:

- Value: of the ID register

Notes:

- Refer to the Reference Manual to get the authorized __VALUE__ length in bits

Input Buffer Format

CRC_INPUTDATA_FORMAT_UNDEFINED

Undefined input data format

CRC_INPUTDATA_FORMAT_BYTES

Input data in byte format

CRC_INPUTDATA_FORMAT_HALFWORDS

Input data in half-word format

CRC_INPUTDATA_FORMAT_WORDS

Input data in word format

Polynomial sizes to configure the peripheral

CRC_POLYLENGTH_32B

Resort to a 32-bit long generating polynomial

CRC_POLYLENGTH_16B

Resort to a 16-bit long generating polynomial

CRC_POLYLENGTH_8B

Resort to a 8-bit long generating polynomial

CRC_POLYLENGTH_7B

Resort to a 7-bit long generating polynomial

CRC polynomial possible sizes actual definitions

HAL_CRC_LENGTH_32B

32-bit long CRC

HAL_CRC_LENGTH_16B

16-bit long CRC

HAL_CRC_LENGTH_8B

8-bit long CRC

HAL_CRC_LENGTH_7B

7-bit long CRC

13 HAL CRC Extension Driver

13.1 CRCEx Firmware driver API description

The following section lists the various functions of the CRCEx library.

13.1.1 How to use this driver

- Set user-defined generating polynomial through HAL_CRCEx_Polynomial_Set()
- Configure Input or Output data inversion

13.1.2 Detailed description of functions

HAL_CRCEx_Polynomial_Set

Function name

HAL_StatusTypeDef HAL_CRCEx_Polynomial_Set (CRC_HandleTypeDef * hcrc, uint32_t Pol, uint32_t PolyLength)

Function description

HAL_CRCEx_Input_Data_Reverse

Function name

HAL_StatusTypeDef HAL_CRCEx_Input_Data_Reverse (CRC_HandleTypeDef * hcrc, uint32_t InputReverseMode)

Function description

HAL_CRCEx_Output_Data_Reverse

Function name

HAL_StatusTypeDef HAL_CRCEx_Output_Data_Reverse (CRC_HandleTypeDef * hcrc, uint32_t OutputReverseMode)

Function description

13.2 CRCEx Firmware driver defines

The following section lists the various define and macros of the module.

13.2.1 CRCEx

CRCEx

CRC Extended Exported Macros

__HAL_CRC_OUTPUTREVERSAL_BIT_ENABLE

Description:

- Set CRC output bit-reversal.

Parameters:

- __HANDLE__: CRC handle

Return value:

- None

__HAL_CRC_OUTPUTREVERSAL_HALFWORD_ENABLE

Description:

- Set CRC output halfword-reversal.

Parameters:

- __HANDLE__: CRC handle

Return value:

- None

__HAL_CRC_OUTPUTREVERSAL_BYTE_ENABLE

Description:

- Set CRC output byte-reversal.

Parameters:

- __HANDLE__: CRC handle

Return value:

- None

__HAL_CRC_OUTPUTREVERSAL_ENABLE

__HAL_CRC_OUTPUTREVERSAL_DISABLE

Description:

- Unset CRC output reversal.

Parameters:

- __HANDLE__: CRC handle

Return value:

- None

__HAL_CRC_POLYNOMIAL_CONFIG

Description:

- Set CRC non-default polynomial.

Parameters:

- __HANDLE__: CRC handle
- __POLYNOMIAL__: 7, 8, 16 or 32-bit polynomial

Return value:

- None

Input Data Inversion Modes

CRC_INPUTDATA_INVERSION_NONE

No input data inversion

CRC_INPUTDATA_INVERSION_HALFWORD_BYWORD

Input data half-word-reversal done by word

CRC_INPUTDATA_INVERSION_BYTE_BYWORD

Input data byte-reversal done by word

CRC_INPUTDATA_INVERSION_BIT_BYBYTE

Input data bit-reversal done by byte

CRC_INPUTDATA_INVERSION_BIT_BYHALFWORD

Input data bit-reversal done by half-word

CRC_INPUTDATA_INVERSION_BIT_BYWORD

Input data bit-reversal done by word

CRC_INPUTDATA_INVERSION_BYTE

Definition for compatibility with legacy code

CRC_INPUTDATA_INVERSION_HALFWORD

Definition for compatibility with legacy code

CRC_INPUTDATA_INVERSION_WORD

Definition for compatibility with legacy code

Output Data Inversion Modes

CRC_OUTPUTDATA_INVERSION_DISABLE

No output data inversion

CRC_OUTPUTDATA_INVERSION_BIT

Output data bit-reversal

CRC_OUTPUTDATA_INVERSION_HALFWORD

Output data half-word-reversal done by word

CRC_OUTPUTDATA_INVERSION_BYTE

Output data byte-reversal done by word

CRC_OUTPUTDATA_INVERSION_ENABLE

Definition for compatibility with legacy code

14 HAL CRYP Generic Driver

14.1 CRYP Firmware driver registers structures

14.1.1 CRYP_ConfigTypeDef

CRYP_ConfigTypeDef is defined in the stm32n6xx_hal_cryp.h

Data Fields

- ***uint32_t DataType***
- ***uint32_t KeySize***
- ***uint32_t * pKey***
- ***uint32_t * pInitVect***
- ***uint32_t Algorithm***
- ***uint32_t * Header***
- ***uint32_t HeaderSize***
- ***uint32_t * B0***
- ***uint32_t DataWidthUnit***
- ***uint32_t HeaderWidthUnit***
- ***uint32_t KeyIVConfigSkip***
- ***uint32_t KeyMode***
- ***uint32_t KeySelect***
- ***uint32_t KeyProtection***

Field Documentation

- ***uint32_t CRYP_ConfigTypeDef::DataType***
  32-bit data, 16-bit data, 8-bit data or 1-bit string. This parameter can be a value of ***CRYP_Data_Type***
- ***uint32_t CRYP_ConfigTypeDef::KeySize***
  Used only in AES mode : 128, 192 or 256 bit key length in CRYP1. This parameter can be a value of ***CRYP_Key_Size***
- ***uint32_t* CRYP_ConfigTypeDef::pKey***
  The key used for encryption/decryption
- ***uint32_t* CRYP_ConfigTypeDef::pInitVect***
  The initialization vector used also as initialization counter in CTR mode
- ***uint32_t CRYP_ConfigTypeDef::Algorithm***
  AES Algorithm ECB/CBC/CTR/GCM or CCM This parameter can be a value of ***CRYP_CR_ALGOMODE***
- ***uint32_t* CRYP_ConfigTypeDef::Header***
  used only in AES GCM and CCM Algorithm for authentication, GCM : also known as Additional Authentication Data CCM : named B1 composed of the associated data length and Associated Data.
- ***uint32_t CRYP_ConfigTypeDef::HeaderSize***
  The size of header buffer in word
- ***uint32_t* CRYP_ConfigTypeDef::B0***
  B0 is first authentication block used only in AES CCM mode
- ***uint32_t CRYP_ConfigTypeDef::DataWidthUnit***
  Data With Unit, this parameter can be value of ***CRYP_Data_Width_Unit***
- ***uint32_t CRYP_ConfigTypeDef::HeaderWidthUnit***
  Header Width Unit, this parameter can be value of ***CRYP_Header_Width_Unit***
- ***uint32_t CRYP_ConfigTypeDef::KeyIVConfigSkip***
  CRYP peripheral Key and IV configuration skip, to configure Key and Initialization Vector only once and to skip configuration for consecutive processing. This parameter can be a value of ***CRYP_Configuration_Skip***
- ***uint32_t CRYP_ConfigTypeDef::KeyMode***
  Key mode selection, this parameter can be a value of ***CRYP_Key_Mode***
- ***uint32_t CRYP_ConfigTypeDef::KeySelect***
  Only for SAES : Key selection, this parameter can be a value of ***CRYP_Key_Select***

- ***uint32_t CRYP_ConfigTypeDef::KeyProtection***
  Only for SAES : Key protection, this parameter can be a value of ***CRYP_Key_Protection***

14.1.2 __CRYP_HandleTypeDef

__CRYP_HandleTypeDef is defined in the stm32n6xx_hal_cryp.h

Data Fields

- ***void * Instance***
- ***CRYP_ConfigTypeDef Init***
- ***uint32_t * pCrypInBuffPtr***
- ***uint32_t * pCrypOutBuffPtr***
- ***__IO uint16_t CrypHeaderCount***
- ***__IO uint16_t CrypInCount***
- ***__IO uint16_t CrypOutCount***
- ***uint16_t Size***
- ***uint32_t Phase***
- ***DMA_HandleTypeDef * hdmain***
- ***DMA_HandleTypeDef * hdmaout***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_CRYP_STATETypeDef State***
- ***__IO uint32_t ErrorCode***
- ***uint32_t Version***
- ***uint32_t KeyIVConfig***
- ***uint32_t SizesSum***
- ***void(* InCpltCallback***
- ***void(* OutCpltCallback***
- ***void(* ErrorCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***void* __CRYP_HandleTypeDef::Instance***
  CRYP or SAES registers base address
- ***CRYP_ConfigTypeDef __CRYP_HandleTypeDef::Init***
  CRYP required parameters
- ***uint32_t* __CRYP_HandleTypeDef::pCrypInBuffPtr***
  Pointer to CRYP processing (encryption, decryption,...) buffer
- ***uint32_t* __CRYP_HandleTypeDef::pCrypOutBuffPtr***
  Pointer to CRYP processing (encryption, decryption,...) buffer
- ***__IO uint16_t __CRYP_HandleTypeDef::CrypHeaderCount***
  Counter of header data
- ***__IO uint16_t __CRYP_HandleTypeDef::CrypInCount***
  Counter of input data
- ***__IO uint16_t __CRYP_HandleTypeDef::CrypOutCount***
  Counter of output data
- ***uint16_t __CRYP_HandleTypeDef::Size***
  length of input data in word or in byte, according to DataWidthUnit
- ***uint32_t __CRYP_HandleTypeDef::Phase***
  CRYP peripheral phase
- ***DMA_HandleTypeDef* __CRYP_HandleTypeDef::hdmain***
  CRYP In DMA handle parameters
- ***DMA_HandleTypeDef* __CRYP_HandleTypeDef::hdmaout***
  CRYP Out DMA handle parameters
- ***HAL_LockTypeDef __CRYP_HandleTypeDef::Lock***
  CRYP locking object

- ***__IO HAL_CRYP_STATETypeDef __CRYP_HandleTypeDef::State***
  CRYP peripheral state
- ***__IO uint32_t __CRYP_HandleTypeDef::ErrorCode***
  CRYP peripheral error code
- ***uint32_t __CRYP_HandleTypeDef::Version***
  CRYP1 IP version
- ***uint32_t __CRYP_HandleTypeDef::KeyIVConfig***
  CRYP peripheral Key and IV configuration flag, used when configuration can be skipped
- ***uint32_t __CRYP_HandleTypeDef::SizesSum***
  Sum of successive payloads lengths (in bytes), stored for a single signature computation after several messages processing
- ***void(* __CRYP_HandleTypeDef::InCpltCallback)(struct __CRYP_HandleTypeDef *hcryp)***
  CRYP Input FIFO transfer completed callback
- ***void(* __CRYP_HandleTypeDef::OutCpltCallback)(struct __CRYP_HandleTypeDef *hcryp)***
  CRYP Output FIFO transfer completed callback
- ***void(* __CRYP_HandleTypeDef::ErrorCallback)(struct __CRYP_HandleTypeDef *hcryp)***
  CRYP Error callback
- ***void(* __CRYP_HandleTypeDef::MspInitCallback)(struct __CRYP_HandleTypeDef *hcryp)***
  CRYP Msp Init callback
- ***void(* __CRYP_HandleTypeDef::MspDeInitCallback)(struct __CRYP_HandleTypeDef *hcryp)***
  CRYP Msp DeInit callback

14.2 CRYP Firmware driver defines

The following section lists the various define and macros of the module.

14.2.1 CRYP

CRYP

SAES peripheral Clear Flags

CRYP_CLEAR_CCF

SAES peripheral clear Computation Complete Flag

CRYP_CLEAR_RWEIF

SAES peripheral clear Error Flag : RWEIF in SAES_ISR and both RDERR and WRERR flags in SAES_SR

CRYP_CLEAR_KEIF

SAES peripheral clear Key Error Flag: KEIF in SAES_ISR

CRYP_CLEAR_RNGEIF

SAES peripheral clear RNG error Flag

CRYP Key and IV Configuration Skip Mode

CRYP_KEYIVCONFIG_ALWAYS

Peripheral Key and IV configuration to do systematically

CRYP_KEYIVCONFIG_ONCE

Peripheral Key and IV configuration to do only once

CRYP_KEYNOCONFIG

Peripheral Key configuration to not do

CRYP Algorithm Mode

CRYP_AES_ECB

CRYP_AES_CBC

CRYP_AES_CTR

CRYP_AES_KEY

CRYP_AES_GCM

CRYP_AES_CCM

CRYP Data Type

CRYP_DATATYPE_32B

CRYP_DATATYPE_16B

CRYP_DATATYPE_8B

CRYP_DATATYPE_1B

CRYP_NO_SWAP
: 32-bit data type (no swapping)

CRYP_HALFWORD_SWAP
: 16-bit data type (half-word swapping)

CRYP_BYTE_SWAP
: 8-bit data type (byte swapping)

CRYP_BIT_SWAP
: 1-bit data type (bit swapping)

CRYP Data Width Unit

CRYP_DATAWIDTHUNIT_WORD
: By default, size unit is word

CRYP_DATAWIDTHUNIT_BYTE
: By default, size unit is byte

CRYP Error Definition

HAL_CRYP_ERROR_NONE
: No error

HAL_CRYP_ERROR_WRITE
: Write error

HAL_CRYP_ERROR_READ
: Read error

HAL_CRYP_ERROR_DMA
: DMA error

HAL_CRYP_ERROR_BUSY
: Busy flag error

HAL_CRYP_ERROR_TIMEOUT

Timeout error

HAL_CRYP_ERROR_NOT_SUPPORTED

Not supported mode

HAL_CRYP_ERROR_AUTH_TAG_SEQUENCE

Sequence are not respected only for GCM or CCM

HAL_CRYP_ERROR_INVALID_CALLBACK

Invalid Callback error

HAL_CRYP_ERROR_KEY

Key error

HAL_CRYP_ERROR_RNG

RNG error

CRYP Exported Macros

IS_CRYP_INSTANCE

Description:

- Inform about which IP is the current INSTANCE: CRYP or SAES.

Parameters:

- INSTANCE: specifies the HW instance.

Return value:

- None

IS_SAES_INSTANCE

__HAL_CRYP_RESET_HANDLE_STATE

Description:

- Reset CRYP handle state.

Parameters:

- __HANDLE__: specifies the CRYP handle.

Return value:

- None

__HAL_CRYP_ENABLE

Description:

- Enable/Disable the CRYP or SAES peripheral.

Parameters:

- __HANDLE__: specifies the CRYP handle.

Return value:

- None

__HAL_CRYP_DISABLE

CRYP_FLAG_MASK

Description:

- Check whether the specified CRYP or SAES peripheral status flag is set or not.

Parameters:

- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - CRYP_FLAG_BUSY: The CRYP core is currently processing a block of data or a key preparation (for AES decryption)
  - CRYP_FLAG_KEYVALID Key valid flag
  - CRYP_FLAG_KEIF Key error flag
  - CRYP_FLAG_IFEM: Input FIFO is empty (CRYP peripheral only)
  - CRYP_FLAG_IFNF: Input FIFO is not full (CRYP peripheral only)
  - CRYP_FLAG_INRIS: Input FIFO service raw interrupt is pending (CRYP peripheral only)
  - CRYP_FLAG_OFNE: Output FIFO is not empty (CRYP peripheral only)
  - CRYP_FLAG_OFFU: Output FIFO is full (CRYP peripheral only)
  - CRYP_FLAG_OUTRIS: Input FIFO service raw interrupt is pending (CRYP peripheral only)
  - CRYP_FLAG_WRERR Write Error flag (SAES peripheral only)
  - CRYP_FLAG_RDERR Read Error flag (SAES peripheral only)
  - CRYP_FLAG_CCF Computation Complete flag (SAES peripheral only)
  - CRYP_FLAG_RWEIF Read/write Error flag (SAES peripheral only)
  - CRYP_FLAG_RNGEIF RNG Error flag (SAES peripheral only)

Return value:

- The: state of __FLAG__ (TRUE or FALSE).

__HAL_CRYP_GET_FLAG

__HAL_CRYP_CLEAR_FLAG

Description:

- Clear the SAES peripheral pending status flag.

Parameters:

- __HANDLE__: specifies the SAES handle.
- __FLAG__: specifies the flag to clear. This parameter can be one of the following values:
  - CRYP_CLEAR_RWEIF Read (RDERR), Write (WRERR) or Read/write (RWEIF) Error Flag Clear
  - CRYP_CLEAR_CCF Computation Complete Flag (CCF) Clear
  - CRYP_CLEAR_KEIF Key error interrupt flag clear
  - CRYP_CLEAR_RNGEIF RNG error interrupt flag clear

Return value:

- None

__HAL_CRYP_GET_IT

Description:

- Check whether the specified CRYP peripheral interrupt is set or not.

Parameters:

- __HANDLE__: specifies the CRYP handle.
- __INTERRUPT__: specifies the interrupt to check. This parameter can be one of the following values:
  - CRYP_IT_INI: Input FIFO service masked interrupt status
  - CRYP_IT_OUTI: Output FIFO service masked interrupt status

Return value:

- The: state of __INTERRUPT__ (TRUE or FALSE).

__HAL_CRYP_ENABLE_IT

Description:

- Enable the CRYP or SAES peripheral interrupt.

Parameters:

- __HANDLE__: specifies the CRYP handle.
- __INTERRUPT__: Interrupt. This parameter can be one of the following values:
  - CRYP_IT_INI Input FIFO service interrupt mask (CRYP peripheral only)
  - CRYP_IT_OUTI Output FIFO service interrupt mask (CRYP peripheral only)
  - CRYP_IT_RWEIE Error interrupt (used for RDERR and WRERR) (SAES peripheral only)
  - CRYP_IT_CCFIE Computation Complete interrupt (SAES peripheral only)
  - CRYP_IT_KEIE Key error interrupt (SAES peripheral only)
  - CRYP_IT_RNGEIE RNG interrupt (SAES peripheral only)

Return value:

- None

__HAL_CRYP_DISABLE_IT

Description:

- Disable the CRYP or SAES peripheral interrupt.

Parameters:

- __HANDLE__: specifies the CRYP handle.
- __INTERRUPT__: Interrupt. This parameter can be one of the following values:
  - CRYP_IT_INI Input FIFO service interrupt mask (CRYP peripheral only)
  - CRYP_IT_OUTI Output FIFO service interrupt mask (CRYP peripheral only)
  - CRYP_IT_RWEIE Error interrupt (used for RDERR and WRERR) (SAES peripheral only)
  - CRYP_IT_CCFIE Computation Complete interrupt (SAES peripheral only)
  - CRYP_IT_KEIE Key error interrupt (SAES peripheral only)
  - CRYP_IT_RNGEIE RNG interrupt (SAES peripheral only)

Return value:

- None

__HAL_CRYP_GET_IT_SOURCE

Description:

- Check whether the specified SAES peripheral interrupt source is enabled or not.

Parameters:

- __HANDLE__: specifies the CRYP handle.
- __INTERRUPT__: interrupt source to check This parameter can be one of the following values for TinyAES:
  - CRYP_IT_RWEIE Error interrupt (used for RDERR and WRERR)
  - CRYP_IT_CCFIE Computation Complete interrupt
  - CRYP_IT_KEIE Key error interrupt
  - CRYP_IT_RNGEIE RNG error interrupt

Return value:

- State: of interruption (TRUE or FALSE).

CRYP & SAES Flags

CRYP_FLAG_IFEM

CRYP peripheral Input FIFO is empty

CRYP_FLAG_IFNF

CRYP peripheral Input FIFO is not Full

CRYP_FLAG_OFNE

CRYP peripheral Output FIFO is not empty

CRYP_FLAG_OFFU

CRYP peripheral Output FIFO is Full

CRYP_FLAG_KERF

CRYP peripheral Key error flag

CRYP_FLAG_OUTRIS

Output FIFO service raw interrupt status

CRYP_FLAG_INRIS

Input FIFO service raw interrupt status

CRYP_FLAG_BUSY

The CRYP peripheral is currently processing a block of data or a key preparation (for AES decryption).

CRYP_FLAG_KEYVALID

CRYP or SAES peripheral Key valid flag

SAES_FLAG_BUSY

The SAES peripheral is currently processing a block of data or a key preparation (for AES decryption).

CRYP_FLAG_WRERR

SAES peripheral Write Error flag

CRYP_FLAG_RDERR

SAES peripheral Read error flag

CRYP_FLAG_CCF

SAES peripheral Computation completed flag as AES_ISR_CCF

CRYP_FLAG_KEIF

SAES peripheral Key error interrupt flag

CRYP_FLAG_RWEIF

SAES peripheral Read or Write error Interrupt flag

CRYP_FLAG_RNGEIF

SAES peripheral RNG error Interrupt flag

CRYP Header Width Unit

CRYP_HEADERWIDTHUNIT_WORD

By default, size unit is word

CRYP_HEADERWIDTHUNIT_BYTE

By default, size unit is byte

CRYP and SAES peripherals interrupts

CRYP_IT_INI

CRYP peripheral input FIFO Interrupt

CRYP_IT_OUTI

CRYP peripheral output FIFO Interrupt

CRYP_IT_CCFIE

SAES peripheral computation Complete interrupt enable

CRYP_IT_RWEIE

SAES peripheral read or write Error interrupt enable

CRYP_IT_KEIE

SAES peripheral key error interrupt enable

CRYP_IT_RNGEIE

SAES peripheral RNG error interrupt enable

CRYP Private macros to check input parameters

IS_CRYP_ALGORITHM

IS_CRYP_KEYSIZE

IS_CRYP_DATATYPE

IS_CRYP_INIT

IS_CRYP_KEYIVCONFIG

IS_CRYP_KEYMODE

IS_SAES_ALGORITHM

IS_SAES_KEYSIZE

IS_SAES_DATATYPE

IS_SAES_KEYMODE

IS_SAES_KEYPROT

IS_SAES_KEYSEL

IS_SAES_KEYSHARED

CRYP or SAES Key Mode

CRYP_KEYMODE_NORMAL

Normal key usage, Key registers are freely usable

CRYP_KEYMODE_WRAPPED

Only for SAES, Wrapped key: to encrypt or decrypt AES keys

CRYP_KEYMODE_SHARED

Key shared by SAES peripheral

SAES Key Protection

CRYP_KEYPROT_ENABLE

Only for SAES, Key can't be shared between two applications with different security contexts

CRYP_KEYPROT_DISABLE

Only for SAES, Key can be shared between two applications with different security contexts

SAES Key Select

CRYP_KEYSEL_NORMAL

Normal key, key registers SAES_KEYx or CRYP_KEYx

CRYP_KEYSEL_HW

Only for SAES, Hardware key : derived hardware unique key (DHUK 256-bit)

CRYP_KEYSEL_SW

Only for SAES, Software key : boot hardware key BHK (256-bit)

CRYP_KEYSEL_HSW

Only for SAES, DHUK XOR BHK Hardware unique key XOR software key

CRYP_KEYSEL_AHK

Only for SAES, Software key : application hardware key AHK (128- or 256-bit)

CRYP_KEYSEL_DUK_AHK

Only for SAES, DHUK XOR AHK

CRYP_KEYSEL_TEST_KEY

Test mode key (256-bit hardware constant key 0xA5A5...A5A5)

SAES Key Shared with Peripheral

CRYP_KEYSHARED_CRYP

Only for SAES, key is shared with CRYP peripheral

CRYP Key Size

CRYP_KEYSIZE_128B

CRYP_KEYSIZE_192B

CRYP_KEYSIZE_256B

SAES processing mode

CRYP_MODE_ENCRYPT

SAES peripheral encryption mode

CRYP_MODE_KEY_DERIVATION

SAES peripheral key derivation

CRYP_MODE_DECRYPT

SAES peripheral decryption mode

CRYP_OPERATINGMODE_ENCRYPT

CRYP peripheral encryption mode

CRYP_OPERATINGMODE_DECRYPT

CRYP peripheral decryption mode

15 HAL CRYP Extension Driver

15.1 CRYPEx Firmware driver API description

The following section lists the various functions of the CRYPEx library.

15.1.1 How to use this driver

The CRYP extension HAL driver can be used after AES-GCM or AES-CCM Encryption/Decryption to get the authentication messages.

15.1.2 Detailed description of functions

HAL_CRYPEx_AESGCM_GenerateAuthTAG

Function name

HAL_StatusTypeDef HAL_CRYPEx_AESGCM_GenerateAuthTAG (CRYP_HandleTypeDef * hcryp, const uint32_t * pAuthTag, uint32_t Timeout)

Function description

HAL_CRYPEx_AESCCM_GenerateAuthTAG

Function name

HAL_StatusTypeDef HAL_CRYPEx_AESCCM_GenerateAuthTAG (CRYP_HandleTypeDef * hcryp, const uint32_t * pAuthTag, uint32_t Timeout)

Function description

HAL_CRYPEx_EncryptSharedKey

Function name

HAL_StatusTypeDef HAL_CRYPEx_EncryptSharedKey (CRYP_HandleTypeDef * hcryp, uint32_t * Key, uint32_t * Output, uint32_t ID, uint32_t Timeout)

Function description

HAL_CRYPEx_DecryptSharedKey

Function name

HAL_StatusTypeDef HAL_CRYPEx_DecryptSharedKey (CRYP_HandleTypeDef * hcryp, uint32_t * Key, uint32_t ID, uint32_t Timeout)

Function description

HAL_CRYPEx_UnwrapKey

Function name

HAL_StatusTypeDef HAL_CRYPEx_UnwrapKey (CRYP_HandleTypeDef * hcryp, uint32_t * Input, uint32_t Timeout)

Function description

HAL_CRYPEx_WrapKey

Function name

HAL_StatusTypeDef HAL_CRYPEx_WrapKey (CRYP_HandleTypeDef * hcryp, uint32_t * Input, uint32_t * Output, uint32_t Timeout)

Function description

16 HAL DCMIPP Generic Driver

16.1 DCMIPP Firmware driver registers structures

16.1.1 DCMIPP_EmbeddedSyncUnmaskTypeDef

DCMIPP_EmbeddedSyncUnmaskTypeDef is defined in the stm32n6xx_hal_dcmipp.h

Data Fields

- ***uint8_t FrameStartUnmask***
- ***uint8_t LineStartUnmask***
- ***uint8_t LineEndUnmask***
- ***uint8_t FrameEndUnmask***

Field Documentation

- ***uint8_t DCMIPP_EmbeddedSyncUnmaskTypeDef::FrameStartUnmask***
  Specifies the frame start delimiter unmask.
- ***uint8_t DCMIPP_EmbeddedSyncUnmaskTypeDef::LineStartUnmask***
  Specifies the line start delimiter unmask.
- ***uint8_t DCMIPP_EmbeddedSyncUnmaskTypeDef::LineEndUnmask***
  Specifies the line end delimiter unmask.
- ***uint8_t DCMIPP_EmbeddedSyncUnmaskTypeDef::FrameEndUnmask***
  Specifies the frame end delimiter unmask.

16.1.2 DCMIPP_EmbeddedSyncCodesTypeDef

DCMIPP_EmbeddedSyncCodesTypeDef is defined in the stm32n6xx_hal_dcmipp.h

Data Fields

- ***uint8_t FrameStartCode***
- ***uint8_t LineStartCode***
- ***uint8_t LineEndCode***
- ***uint8_t FrameEndCode***

Field Documentation

- ***uint8_t DCMIPP_EmbeddedSyncCodesTypeDef::FrameStartCode***
  Specifies the code of the frame start delimiter.
- ***uint8_t DCMIPP_EmbeddedSyncCodesTypeDef::LineStartCode***
  Specifies the code of the line start delimiter.
- ***uint8_t DCMIPP_EmbeddedSyncCodesTypeDef::LineEndCode***
  Specifies the code of the line end delimiter.
- ***uint8_t DCMIPP_EmbeddedSyncCodesTypeDef::FrameEndCode***
  Specifies the code of the frame end delimiter.

16.1.3 DCMIPP_CSI_PIPE_ConfTypeDef

DCMIPP_CSI_PIPE_ConfTypeDef is defined in the stm32n6xx_hal_dcmipp.h

Data Fields

- ***uint32_t DataTypeMode***
- ***uint32_t DataTypeIDA***
- ***uint32_t DataTypeIDB***

Field Documentation

- ***uint32_t DCMIPP_CSI_PIPE_ConfTypeDef::DataTypeMode***
  Configures the Data Type Mode This parameter can be a value from ***DCMIPP_DataTypeMode***
- ***uint32_t DCMIPP_CSI_PIPE_ConfTypeDef::DataTypeIDA***
  Configures the Data type selection ID-A This parameter can be a value from ***DCMIPP_DataType***

- ***uint32_t DCMIPP_CSI_PIPE_ConfTypeDef::DataTypeIDB***
  Configures the Data type selection ID-B This parameter can be a value from ***DCMIPP_DataType***

16.1.4 DCMIPP_CSI_ConfTypeDef

DCMIPP_CSI_ConfTypeDef is defined in the stm32n6xx_hal_dcmipp.h

Data Fields

- ***uint32_t NumberOfLanes***
- ***uint32_t DataLaneMapping***
- ***uint32_t PHYBitrate***

Field Documentation

- ***uint32_t DCMIPP_CSI_ConfTypeDef::NumberOfLanes***
  Configures the Number of Lanes This parameter can be a value from ***DCMIPP_CSI_Number_Of_Lanes***
- ***uint32_t DCMIPP_CSI_ConfTypeDef::DataLaneMapping***
  Configures the Data Lane Mapping This parameter can be a value from ***DCMIPP_CSI_DataLaneMapping***
- ***uint32_t DCMIPP_CSI_ConfTypeDef::PHYBitrate***
  Configures the Phy Bitrate This parameter can be a value from ***DCMIPP_CSI_PHYBitRate***

16.1.5 DCMIPP_CSI_VCFilteringConfTypeDef

DCMIPP_CSI_VCFilteringConfTypeDef is defined in the stm32n6xx_hal_dcmipp.h

Data Fields

- ***uint32_t DataTypeNB***
- ***uint32_t DataTypeClass***
- ***uint32_t DataTypeFormat***

Field Documentation

- ***uint32_t DCMIPP_CSI_VCFilteringConfTypeDef::DataTypeNB***
  Configures the Data Type number This parameter can be a value from ***DCMIPP_CSI_DataTypeID***
- ***uint32_t DCMIPP_CSI_VCFilteringConfTypeDef::DataTypeClass[MAX_DATATYPE_NB]***
  Configures the Data Type Class This parameter can be a value between 0 and 63
- ***uint32_t DCMIPP_CSI_VCFilteringConfTypeDef::DataTypeFormat[MAX_DATATYPE_NB]***
  Configures the Data Type Class This parameter can be a value from ***DCMIPP_CSI_DataTypeFormat***

16.1.6 DCMIPP_CSI_TimerConfTypeDef

DCMIPP_CSI_TimerConfTypeDef is defined in the stm32n6xx_hal_dcmipp.h

Data Fields

- ***uint32_t VirtualChannel***
- ***uint32_t StartPoint***
- ***uint32_t Count***

Field Documentation

- ***uint32_t DCMIPP_CSI_TimerConfTypeDef::VirtualChannel***
  The Virtual Channel ID This parameter can be a value from ***DCMIPP_Virtual_Channel***
- ***uint32_t DCMIPP_CSI_TimerConfTypeDef::StartPoint***
  The Start Point of the counter This parameter can be a value from ***DCMIPP_CSI_Timer_StartPoint***
- ***uint32_t DCMIPP_CSI_TimerConfTypeDef::Count***
  Number of clock cycle to count from the start point This parameter can be a value between 0 and 0x1FFFFFF

16.1.7 DCMIPP_CSI_LineByteCounterConfTypeDef

DCMIPP_CSI_LineByteCounterConfTypeDef is defined in the stm32n6xx_hal_dcmipp.h

Data Fields

- ***uint32_t VirtualChannel***
- ***uint32_t LineCounter***
- ***uint32_t ByteCounter***

Field Documentation

- ***uint32_t DCMIPP_CSI_LineByteCounterConfTypeDef::VirtualChannel***
  Configures the Virtual Channel ID This parameter can be a value from ***DCMIPP_Virtual_Channel***
- ***uint32_t DCMIPP_CSI_LineByteCounterConfTypeDef::LineCounter***
  Configures the Line Number This parameter can be a value between 0 and 65535
- ***uint32_t DCMIPP_CSI_LineByteCounterConfTypeDef::ByteCounter***
  Configures the Byte Number This parameter can be a value between 0 and 65535

16.1.8 DCMIPP_ParallelConfTypeDef

DCMIPP_ParallelConfTypeDef is defined in the stm32n6xx_hal_dcmipp.h

Data Fields

- ***uint32_t Format***
- ***uint32_t VSPolarity***
- ***uint32_t HSPolarity***
- ***uint32_t PCKPolarity***
- ***uint32_t ExtendedDataMode***
- ***uint32_t SynchroMode***
- ***DCMIPP_EmbeddedSyncCodesTypeDef SynchroCodes***
- ***uint32_t SwapBits***
- ***uint32_t SwapCycles***

Field Documentation

- ***uint32_t DCMIPP_ParallelConfTypeDef::Format***
  Configures the DCMIPP Format This parameter can be one value of ***DCMIPP_Format***
- ***uint32_t DCMIPP_ParallelConfTypeDef::VSPolarity***
  Specifies the Vertical synchronization polarity: High or Low. This parameter can be a value of ***DCMIPP_VSYNC_Polarity***
- ***uint32_t DCMIPP_ParallelConfTypeDef::HSPolarity***
  Specifies the Horizontal synchronization polarity: High or Low. This parameter can be a value of ***DCMIPP_HSYNC_Polarity***
- ***uint32_t DCMIPP_ParallelConfTypeDef::PCKPolarity***
  Specifies the Pixel clock polarity: Falling or Rising. This parameter can be a value of ***DCMIPP_PIXCK_Polarity***
- ***uint32_t DCMIPP_ParallelConfTypeDef::ExtendedDataMode***
  Specifies the data width: 8-bit, 10-bit, 12-bit 14-bit or 16-bits. This parameter can be a value of ***DCMIPP_Extended_Data_Mode***
- ***uint32_t DCMIPP_ParallelConfTypeDef::SynchroMode***
  Specifies the Synchronization Mode: Hardware or Embedded. This parameter can be a value of ***DCMIPP_Synchronization_Mode***
- ***DCMIPP_EmbeddedSyncCodesTypeDef DCMIPP_ParallelConfTypeDef::SynchroCodes***
  Specifies the code of the line/frame start delimiter and the line/frame end delimiter
- ***uint32_t DCMIPP_ParallelConfTypeDef::SwapBits***
  Enable or Disable the Swap Bits. This parameter can be a value of ***DCMIPP_SWAP_BITS***
- ***uint32_t DCMIPP_ParallelConfTypeDef::SwapCycles***
  Enable or Disable the Swap Cycles. This parameter can be a value of ***DCMIPP_SWAP_CYCLES***

16.1.9 DCMIPP_PipeConfTypeDef

DCMIPP_PipeConfTypeDef is defined in the stm32n6xx_hal_dcmipp.h

Data Fields

- ***uint32_t FrameRate***
- ***uint32_t PixelPipePitch***
- ***uint32_t PixelPackerFormat***

Field Documentation

- ***uint32_t DCMIPP_PipeConfTypeDef::FrameRate***
  Configures the DCMIPP Pipe Frame Rate This parameter can be one value of ***DCMIPP_Frame_Rates***
- ***uint32_t DCMIPP_PipeConfTypeDef::PixelPipePitch***
  Configures the DCMIPP Pixel Pipe Pitch This parameter can be one value from
- ***uint32_t DCMIPP_PipeConfTypeDef::PixelPackerFormat***
  Configures the DCMIPP Pixel Pipe Format This parameter can be one value from ***DCMIPP_Pixel_Packer_Format***

16.1.10 DCMIPP_IPPlugConfTypeDef

DCMIPP_IPPlugConfTypeDef is defined in the stm32n6xx_hal_dcmipp.h

Data Fields

- ***uint32_t Client***
- ***uint32_t MemoryPageSize***
- ***uint32_t Traffic***
- ***uint32_t MaxOutstandingTransactions***
- ***uint32_t DPREGStart***
- ***uint32_t DPREGEnd***
- ***uint32_t WLRURatio***

Field Documentation

- ***uint32_t DCMIPP_IPPlugConfTypeDef::Client***
  Configures the DCMIPP IPPLUG Client This parameter can be a value from ***DCMIPP_IPPLUG_Client***
- ***uint32_t DCMIPP_IPPlugConfTypeDef::MemoryPageSize***
  Configures the DCMIPP IPPLUG Memory page size This parameter can be a value from ***DCMIPP_Memory_Page_Size***
- ***uint32_t DCMIPP_IPPlugConfTypeDef::Traffic***
  Configures the DCMIPP IPPLUG Traffic This parameter can be a value from ***DCMIPP_Traffic_Burst_Size***
- ***uint32_t DCMIPP_IPPlugConfTypeDef::MaxOutstandingTransactions***
  Configures the DCMIPP IPPLUG Maximum outstanding transactions This parameter can be a value from DCMIPP_Maximum_Outstanding_Transactions
- ***uint32_t DCMIPP_IPPlugConfTypeDef::DPREGStart***
  Configures the End word of the FIFO of Clientx This parameter can be a value between 0 and 0x3FF
- ***uint32_t DCMIPP_IPPlugConfTypeDef::DPREGEnd***
  Configures the Start word of the FIFO of Clientx This parameter can be a value between 0 and 0x3FF
- ***uint32_t DCMIPP_IPPlugConfTypeDef::WLRURatio***
  Configures the DCMIPP Ratio for WLRU arbitration This parameter can be a value between 0 and 15

16.1.11 DCMIPP_CropConfTypeDef

DCMIPP_CropConfTypeDef is defined in the stm32n6xx_hal_dcmipp.h

Data Fields

- ***uint32_t VStart***
- ***uint32_t HStart***
- ***uint32_t VSize***
- ***uint32_t HSize***
- ***uint32_t PipeArea***

Field Documentation

- ***uint32_t DCMIPP_CropConfTypeDef::VStart***
  Configures the DCMIPP Crop Vertical Start This parameter can be one value between 0 and 4095
- ***uint32_t DCMIPP_CropConfTypeDef::HStart***
  Configures the DCMIPP Crop Horizontal Start This parameter can be one value between 0 and 4095
- ***uint32_t DCMIPP_CropConfTypeDef::VSize***
  Configures the DCMIPP Crop Vertical Size This parameter can be one value between 0 and 4095
- ***uint32_t DCMIPP_CropConfTypeDef::HSize***
  Configures the DCMIPP Crop Horizontal Size This parameter can be one value between 1 and 4095

- ***uint32_t DCMIPP_CropConfTypeDef::PipeArea***
  Configures the DCMIPP Crop Area for the pipe0 This parameter can be one value of ***DCMIPP_Crop_Area***

16.1.12 DCMIPP_StatisticExtractionAreaConfTypeDef

DCMIPP_StatisticExtractionAreaConfTypeDef is defined in the stm32n6xx_hal_dcmipp.h

Data Fields

- ***uint32_t VStart***
- ***uint32_t HStart***
- ***uint32_t VSize***
- ***uint32_t HSize***

Field Documentation

- ***uint32_t DCMIPP_StatisticExtractionAreaConfTypeDef::VStart***
  Configures the DCMIPP Statistic Extraction Vertical Start This parameter can be one value between 0 and 4095
- ***uint32_t DCMIPP_StatisticExtractionAreaConfTypeDef::HStart***
  Configures the DCMIPP Statistic Extraction Horizontal Start This parameter can be one value between 0 and 4095
- ***uint32_t DCMIPP_StatisticExtractionAreaConfTypeDef::VSize***
  Configures the DCMIPP Statistic Extraction Vertical Size This parameter can be one value between 0 and 4095
- ***uint32_t DCMIPP_StatisticExtractionAreaConfTypeDef::HSize***
  Configures the DCMIPP Statistic Extraction Horizontal Size This parameter can be one value between 0 and 4095

16.1.13 DCMIPP_StatisticExtractionConfTypeDef

DCMIPP_StatisticExtractionConfTypeDef is defined in the stm32n6xx_hal_dcmipp.h

Data Fields

- ***uint32_t Mode***
- ***uint32_t Source***
- ***uint32_t Bins***

Field Documentation

- ***uint32_t DCMIPP_StatisticExtractionConfTypeDef::Mode***
  Configures the DCMIPP Statistic Extraction Mode This parameter can be a value from ***DCMIPP_Statistics_Extraction_Mode***
- ***uint32_t DCMIPP_StatisticExtractionConfTypeDef::Source***
  Configures the DCMIPP Statistic Extraction Source This parameter can be a value from ***DCMIPP_Statistics_Extraction_Source***
- ***uint32_t DCMIPP_StatisticExtractionConfTypeDef::Bins***
  Configures the DCMIPP Statistic Extraction Bins This parameter can be a value from ***DCMIPP_Statistics_Extraction_Bins***

16.1.14 DCMIPP_ExposureConfTypeDef

DCMIPP_ExposureConfTypeDef is defined in the stm32n6xx_hal_dcmipp.h

Data Fields

- ***uint8_t ShiftRed***
- ***uint8_t MultiplierRed***
- ***uint8_t ShiftGreen***
- ***uint8_t MultiplierGreen***
- ***uint8_t ShiftBlue***
- ***uint8_t MultiplierBlue***

Field Documentation

- ***uint8_t DCMIPP_ExposureConfTypeDef::ShiftRed***
  Configures the DCMIPP Exposure Shift Red This parameter can be one value between 0 and 7

- ***uint8_t DCMIPP_ExposureConfTypeDef::MultiplierRed***
  Configures the DCMIPP Exposure Multiplier Red This parameter can be one value between 0 and 127
- ***uint8_t DCMIPP_ExposureConfTypeDef::ShiftGreen***
  Configures the DCMIPP Exposure Shift Green This parameter can be one value between 0 and 7
- ***uint8_t DCMIPP_ExposureConfTypeDef::MultiplierGreen***
  Configures the DCMIPP Exposure Multiplier Green This parameter can be one value between 0 and 127
- ***uint8_t DCMIPP_ExposureConfTypeDef::ShiftBlue***
  Configures the DCMIPP Exposure Shift Blue This parameter can be one value between 0 and 7
- ***uint8_t DCMIPP_ExposureConfTypeDef::MultiplierBlue***
  Configures the DCMIPP Exposure Multiplier Blue This parameter can be one value between 0 and 127

16.1.15 DCMIPP_ContrastConfTypeDef

DCMIPP_ContrastConfTypeDef is defined in the stm32n6xx_hal_dcmipp.h

Data Fields

- ***uint8_t LUM_0***
- ***uint8_t LUM_32***
- ***uint8_t LUM_64***
- ***uint8_t LUM_96***
- ***uint8_t LUM_128***
- ***uint8_t LUM_160***
- ***uint8_t LUM_192***
- ***uint8_t LUM_224***
- ***uint8_t LUM_256***

Field Documentation

- ***uint8_t DCMIPP_ContrastConfTypeDef::LUM_0***
  Luminance increase for input luminance of 0 This parameter can be one value between 0 and 31
- ***uint8_t DCMIPP_ContrastConfTypeDef::LUM_32***
  Luminance increase for input luminance of 32 This parameter can be one value between 0 and 31
- ***uint8_t DCMIPP_ContrastConfTypeDef::LUM_64***
  Luminance increase for input luminance of 64 This parameter can be one value between 0 and 31
- ***uint8_t DCMIPP_ContrastConfTypeDef::LUM_96***
  Luminance increase for input luminance of 96 This parameter can be one value between 0 and 31
- ***uint8_t DCMIPP_ContrastConfTypeDef::LUM_128***
  Luminance increase for input luminance of 128 This parameter can be one value between 0 and 31
- ***uint8_t DCMIPP_ContrastConfTypeDef::LUM_160***
  Luminance increase for input luminance of 160 This parameter can be one value between 0 and 31
- ***uint8_t DCMIPP_ContrastConfTypeDef::LUM_192***
  Luminance increase for input luminance of 192 This parameter can be one value between 0 and 31
- ***uint8_t DCMIPP_ContrastConfTypeDef::LUM_224***
  Luminance increase for input luminance of 224 This parameter can be one value between 0 and 31
- ***uint8_t DCMIPP_ContrastConfTypeDef::LUM_256***
  Luminance increase for input luminance of 256 This parameter can be one value between 0 and 31

16.1.16 DCMIPP_RawBayer2RGBConfTypeDef

DCMIPP_RawBayer2RGBConfTypeDef is defined in the stm32n6xx_hal_dcmipp.h

Data Fields

- ***uint32_t VLineStrength***
- ***uint32_t HLineStrength***
- ***uint32_t RawBayerType***
- ***uint32_t PeakStrength***
- ***uint32_t EdgeStrength***

Field Documentation

- ***uint32_t DCMIPP_RawBayer2RGBConfTypeDef::VLineStrength***
  Strength of the vertical line detection This parameter can be a value from ***DCMIPP_RAWBayer2RGB_Strength***
- ***uint32_t DCMIPP_RawBayer2RGBConfTypeDef::HLineStrength***
  Strength of the horizontal line detection This parameter can be a value from ***DCMIPP_RAWBayer2RGB_Strength***
- ***uint32_t DCMIPP_RawBayer2RGBConfTypeDef::RawBayerType***
  Raw Bayer type This parameter can be a value from ***DCMIPP_RAWBayer2RGB_Type***
- ***uint32_t DCMIPP_RawBayer2RGBConfTypeDef::PeakStrength***
  Strength of the peak detection This parameter can be a value from ***DCMIPP_RAWBayer2RGB_Strength***
- ***uint32_t DCMIPP_RawBayer2RGBConfTypeDef::EdgeStrength***
  Strength of the edge detection This parameter can be a value from ***DCMIPP_RAWBayer2RGB_Strength***

16.1.17 DCMIPP_ColorConversionConfTypeDef

DCMIPP_ColorConversionConfTypeDef is defined in the stm32n6xx_hal_dcmipp.h

Data Fields

- ***FunctionalState ClampOutputSamples***
- ***uint8_t OutputSamplesType***
- ***int16_t RR***
- ***int16_t RG***
- ***int16_t RB***
- ***int16_t RA***
- ***int16_t GR***
- ***int16_t GG***
- ***int16_t GB***
- ***int16_t GA***
- ***int16_t BR***
- ***int16_t BG***
- ***int16_t BB***
- ***int16_t BA***

Field Documentation

- ***FunctionalState DCMIPP_ColorConversionConfTypeDef::ClampOutputSamples***
  Clamp the output samples depending on type This parameter can be ENABLE or DISABLE
- ***uint8_t DCMIPP_ColorConversionConfTypeDef::OutputSamplesType***
  Output samples type used while clamp is activated This parameter can be a value from ***DCMIPP_OutputSamplesType***
- ***int16_t DCMIPP_ColorConversionConfTypeDef::RR***
  Coefficient Row 1 Column 1 of the matrix This parameter can be a value from -1023 to 1023
- ***int16_t DCMIPP_ColorConversionConfTypeDef::RG***
  Coefficient Row 1 Column 2 of the matrix This parameter can be a value from -1023 to 1023
- ***int16_t DCMIPP_ColorConversionConfTypeDef::RB***
  Coefficient Row 1 Column 3 of the matrix This parameter can be a value from -511 to 511
- ***int16_t DCMIPP_ColorConversionConfTypeDef::RA***
  Coefficient Row 1 of the added Column This parameter can be a value from -511 to 511
- ***int16_t DCMIPP_ColorConversionConfTypeDef::GR***
  Coefficient Row 2 Column 1 of the matrix This parameter can be a value from -1023 to 1023
- ***int16_t DCMIPP_ColorConversionConfTypeDef::GG***
  Coefficient Row 2 Column 2 of the matrix This parameter can be a value from -1023 to 1023
- ***int16_t DCMIPP_ColorConversionConfTypeDef::GB***
  Coefficient Row 2 Column 3 of the matrix This parameter can be a value from -511 to 511
- ***int16_t DCMIPP_ColorConversionConfTypeDef::GA***
  Coefficient Row 2 of the added Column This parameter can be a value from -511 to 511
- ***int16_t DCMIPP_ColorConversionConfTypeDef::BR***
  Coefficient Row 3 Column 1 of the matrix This parameter can be a value from -1023 to 1023

- ***int16_t DCMIPP_ColorConversionConfTypeDef::BG***
  Coefficient Row 3 Column 2 of the matrix This parameter can be a value -1023 to 1023
- ***int16_t DCMIPP_ColorConversionConfTypeDef::BB***
  Coefficient Row 3 Column 3 of the matrix This parameter can be a value from -511 to 511
- ***int16_t DCMIPP_ColorConversionConfTypeDef::BA***
  Coefficient Row 3 of the added Column This parameter can be a value from -511 to 511

16.1.18 DCMIPP_RegionOfInterestConfTypeDef

DCMIPP_RegionOfInterestConfTypeDef is defined in the stm32n6xx_hal_dcmipp.h

Data Fields

- ***uint32_t VStart***
- ***uint32_t HStart***
- ***uint32_t VSize***
- ***uint32_t HSize***
- ***uint32_t LineSizeWidth***
- ***uint32_t RegionOfInterest***
- ***uint8_t ColorLineRed***
- ***uint8_t ColorLineGreen***
- ***uint8_t ColorLineBlue***

Field Documentation

- ***uint32_t DCMIPP_RegionOfInterestConfTypeDef::VStart***
  Configures the DCMIPP Region Of Interest Vertical Start This parameter can be one value between 0 and 4095
- ***uint32_t DCMIPP_RegionOfInterestConfTypeDef::HStart***
  Configures the DCMIPP Region Of Interest Horizontal Start This parameter can be one value between 0 and 4095
- ***uint32_t DCMIPP_RegionOfInterestConfTypeDef::VSize***
  Configures the DCMIPP Region Of Interest Vertical Size This parameter can be one value between 0 and 4095
- ***uint32_t DCMIPP_RegionOfInterestConfTypeDef::HSize***
  Configures the DCMIPP Region Of Interest Horizontal Size This parameter can be one value between 0 and 4095
- ***uint32_t DCMIPP_RegionOfInterestConfTypeDef::LineSizeWidth***
  Configures the DCMIPP Region Of Interest Line Size Width This parameter can be a value from ***DCMIPP_Region_Of_Interest_Line_Width***
- ***uint32_t DCMIPP_RegionOfInterestConfTypeDef::RegionOfInterest***
  Configures the DCMIPP Region Of Interest This parameter can be a value from ***DCMIPP_Region_Of_Interest***
- ***uint8_t DCMIPP_RegionOfInterestConfTypeDef::ColorLineRed***
  Configures the DCMIPP Region Of Interest Line Color : Red Component This parameter can be one value between 0 and 4095
- ***uint8_t DCMIPP_RegionOfInterestConfTypeDef::ColorLineGreen***
  Configures the DCMIPP Region Of Interest Line Color : Green Component This parameter can be one value between 0 and 4095
- ***uint8_t DCMIPP_RegionOfInterestConfTypeDef::ColorLineBlue***
  Configures the DCMIPP Region Of Interest Line Color : Blue Component This parameter can be one value between 0 and 4095

16.1.19 DCMIPP_BlackLevelConfTypeDef

DCMIPP_BlackLevelConfTypeDef is defined in the stm32n6xx_hal_dcmipp.h

Data Fields

- ***uint8_t RedCompBlackLevel***
- ***uint8_t GreenCompBlackLevel***
- ***uint8_t BlueCompBlackLevel***

Field Documentation

- ***uint8_t DCMIPP_BlackLevelConfTypeDef::RedCompBlackLevel***
  Black value register to red component This parameter can be one value between 0 and 255
- ***uint8_t DCMIPP_BlackLevelConfTypeDef::GreenCompBlackLevel***
  Black value register to green component This parameter can be one value between 0 and 255
- ***uint8_t DCMIPP_BlackLevelConfTypeDef::BlueCompBlackLevel***
  Black value register to blue component This parameter can be one value between 0 and 255

16.1.20 DCMIPP_DownsizeTypeDef

DCMIPP_DownsizeTypeDef is defined in the stm32n6xx_hal_dcmipp.h

Data Fields

- ***uint32_t VSize***
- ***uint32_t HSize***
- ***uint32_t VRatio***
- ***uint32_t HRatio***
- ***uint32_t VDivFactor***
- ***uint32_t HDivFactor***

Field Documentation

- ***uint32_t DCMIPP_DownsizeTypeDef::VSize***
  Configures the DCMIPP Downsize Vertical Size This parameter can be one value between 0 and 4095
- ***uint32_t DCMIPP_DownsizeTypeDef::HSize***
  Configures the DCMIPP Downsize Horizontal Size This parameter can be one value between 0 and 4095
- ***uint32_t DCMIPP_DownsizeTypeDef::VRatio***
  Configures the DCMIPP Downsize Vertical Ratio This parameter can be one value between 0 and 4095
- ***uint32_t DCMIPP_DownsizeTypeDef::HRatio***
  Configures the DCMIPP Downsize Horizontal Ratio This parameter can be one value between 0 and 4095
- ***uint32_t DCMIPP_DownsizeTypeDef::VDivFactor***
  Configures the DCMIPP Downsize Vertical Division Factor This parameter can be one value between 0 and 4095
- ***uint32_t DCMIPP_DownsizeTypeDef::HDivFactor***
  Configures the DCMIPP Downsize Horizontal Division Factor This parameter can be one value between 0 and 4095

16.1.21 DCMIPP_DecimationConfTypeDef

DCMIPP_DecimationConfTypeDef is defined in the stm32n6xx_hal_dcmipp.h

Data Fields

- ***uint32_t VRatio***
- ***uint32_t HRatio***

Field Documentation

- ***uint32_t DCMIPP_DecimationConfTypeDef::VRatio***
  Configures the DCMIPP Downsize Vertical Size This parameter can be a value from ***DCMIPP_PIPE_Vertical_Decimation_Ratio***
- ***uint32_t DCMIPP_DecimationConfTypeDef::HRatio***
  Configures the DCMIPP Downsize Vertical Size This parameter can be a value from ***DCMIPP_PIPE_Horizontal_Decimation_Ratio***

16.1.22 DCMIPP_SemiPlanarDstAddressTypeDef

DCMIPP_SemiPlanarDstAddressTypeDef is defined in the stm32n6xx_hal_dcmipp.h

Data Fields

- ***uint32_t YAddress***
- ***uint32_t UVAddress***

Field Documentation

- ***uint32_t DCMIPP_SemiPlanarDstAddressTypeDef::YAddress***
  Y Frame Buffer address
- ***uint32_t DCMIPP_SemiPlanarDstAddressTypeDef::UVAddress***
  UV Frame Buffer address

16.1.23 DCMIPP_FullPlanarDstAddressTypeDef

DCMIPP_FullPlanarDstAddressTypeDef is defined in the stm32n6xx_hal_dcmipp.h

Data Fields

- ***uint32_t YAddress***
- ***uint32_t UAddress***
- ***uint32_t VAddress***

Field Documentation

- ***uint32_t DCMIPP_FullPlanarDstAddressTypeDef::YAddress***
  Y Frame Buffer address
- ***uint32_t DCMIPP_FullPlanarDstAddressTypeDef::UAddress***
  U Frame Buffer address
- ***uint32_t DCMIPP_FullPlanarDstAddressTypeDef::VAddress***
  V Frame Buffer address

16.1.24 __DCMIPP_HandleTypeDef

__DCMIPP_HandleTypeDef is defined in the stm32n6xx_hal_dcmipp.h

Data Fields

- ***DCMIPP_TypeDef * Instance***
- ***__IO HAL_DCMIPP_StateTypeDef State***
- ***__IO HAL_DCMIPP_PipeStateTypeDef PipeState***
- ***__IO uint32_t ErrorCode***
- ***void(* PIPE_FrameEventCallback***
- ***void(* PIPE_VsyncEventCallback***
- ***void(* PIPE_LineEventCallback***
- ***void(* PIPE_LimitEventCallback***
- ***void(* PIPE_ErrorCallback***
- ***void(* ErrorCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***
- ***void(* StartOfFrameEventCallback***
- ***void(* EndOfFrameEventCallback***
- ***void(* TimerCounterEventCallback***
- ***void(* LineByteEventCallback***
- ***void(* LineErrorCallback***
- ***void(* ClockChangerFifoFullEventCallback***
- ***void(* ShortPacketDetectionEventCallback***

Field Documentation

- ***DCMIPP_TypeDef* __DCMIPP_HandleTypeDef::Instance***
  Register the DCMIPP base address
- ***__IO HAL_DCMIPP_StateTypeDef __DCMIPP_HandleTypeDef::State***
  DCMIPP state
- ***__IO HAL_DCMIPP_PipeStateTypeDef***
  __DCMIPP_HandleTypeDef::PipeState[DCMIPP_NUM_OF_PIPES]
  DCMIPP Pipes state
- ***__IO uint32_t __DCMIPP_HandleTypeDef::ErrorCode***
  DCMIPP Error code

- ***void(* __DCMIPP_HandleTypeDef::PIPE_FrameEventCallback)(struct __DCMIPP_HandleTypeDef *hdcmipp, uint32_t Pipe)***
  DCMIPP Pipe Frame Event Callback
- ***void(* __DCMIPP_HandleTypeDef::PIPE_VsyncEventCallback)(struct __DCMIPP_HandleTypeDef *hdcmipp, uint32_t Pipe)***
  DCMIPP Pipe Vsync Event Callback
- ***void(* __DCMIPP_HandleTypeDef::PIPE_LineEventCallback)(struct __DCMIPP_HandleTypeDef *hdcmipp, uint32_t Pipe)***
  DCMIPP Pipe Line Event Callback
- ***void(* __DCMIPP_HandleTypeDef::PIPE_LimitEventCallback)(struct __DCMIPP_HandleTypeDef *hdcmipp, uint32_t Pipe)***
  DCMIPP Pipe Limit Event Callback
- ***void(* __DCMIPP_HandleTypeDef::PIPE_ErrorCallback)(struct __DCMIPP_HandleTypeDef *hdcmipp, uint32_t Pipe)***
  DCMIPP Pipe Error Callback
- ***void(* __DCMIPP_HandleTypeDef::ErrorCallback)(struct __DCMIPP_HandleTypeDef *hdcmipp)***
  DCMIPP Error Callback
- ***void(* __DCMIPP_HandleTypeDef::MspInitCallback)(struct __DCMIPP_HandleTypeDef *hdcmipp)***
  DCMIPP Msp Init Callback
- ***void(* __DCMIPP_HandleTypeDef::MspDeInitCallback)(struct __DCMIPP_HandleTypeDef *hdcmipp)***
  DCMIPP Msp DeInit Callback
- ***void(* __DCMIPP_HandleTypeDef::StartOfFrameEventCallback)(struct __DCMIPP_HandleTypeDef *hdcmipp, uint32_t VirtualChannel)***
  DCMIPP CSI Start Of Frame Event Callback
- ***void(* __DCMIPP_HandleTypeDef::EndOfFrameEventCallback)(struct __DCMIPP_HandleTypeDef *hdcmipp, uint32_t VirtualChannel)***
  DCMIPP CSI End Of Frame Event Callback
- ***void(* __DCMIPP_HandleTypeDef::TimerCounterEventCallback)(struct __DCMIPP_HandleTypeDef *hdcmipp, uint32_t Timer)***
  DCMIPP CSI Timer Counter Event Callback
- ***void(* __DCMIPP_HandleTypeDef::LineByteEventCallback)(struct __DCMIPP_HandleTypeDef *hdcmipp, uint32_t Counter)***
  DCMIPP CSI Line/Byte Event Callback
- ***void(* __DCMIPP_HandleTypeDef::LineErrorCallback)(struct __DCMIPP_HandleTypeDef *hdcmipp, uint32_t DataLane)***
  DCMIPP CSI Line Error Event Callback
- ***void(* __DCMIPP_HandleTypeDef::ClockChangerFifoFullEventCallback)(struct __DCMIPP_HandleTypeDef *hdcmipp)***
  DCMIPP CSI Clock changer Fifo Event Callback
- ***void(* __DCMIPP_HandleTypeDef::ShortPacketDetectionEventCallback)(struct __DCMIPP_HandleTypeDef *hdcmipp)***
  DCMIPP CSI Short Packet detection Event Callback

16.2 DCMIPP Firmware driver API description

The following section lists the various functions of the DCMIPP library.

16.2.1 How to use this driver

The sequence below describes how to use this driver to capture image from a camera module connected to the DCMIPP Interface. This sequence does not take into account the configuration of the camera module, which should be made before to configure and enable the DCMIPP to capture images. Initialize the DCMIPP and the CSI through HAL_DCMIPP_Init() function.

Mandatory Configuration

Interrupt mode IO operation

DCMIPP HAL driver macros list

Below the list of most used macros in DCMIPP HAL driver :

- __HAL_DCMIPP_GET_FLAG: Get the DCMIPP pending flags.
- __HAL_DCMIPP_CLEAR_FLAG: Clear the DCMIPP pending flags.
- __HAL_DCMIPP_ENABLE_IT: Enable the specified DCMIPP interrupts.
- __HAL_DCMIPP_DISABLE_IT: Disable the specified DCMIPP interrupts.
- __HAL_DCMIPP_GET_IT_SOURCE: Check whether the specified DCMIPP interrupt is enabled or not.
- __HAL_DCMIPP_CSI_GET_FLAG: Get the CSI pending flags.
- __HAL_DCMIPP_CSI_CLEAR_FLAG: Clear the CSI pending flags.
- __HAL_DCMIPP_CSI_ENABLE_IT: Enable the specified CSI interrupts.
- __HAL_DCMIPP_CSI_DISABLE_IT: Disable the specified CSI interrupts.
- __HAL_DCMIPP_CSI_GET_IT_SOURCE: Check whether the specified CSI interrupt is enabled or not.

Callback registration

1. The compilation define USE_HAL_DCMIPP_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks.
   – Use function HAL_DCMIPP_RegisterCallback() to register a user callback.
   – Use function HAL_DCMIPP_PIPE_RegisterCallback() to register a user pipe callback.
2. Function HAL_DCMIPP_RegisterCallback() allows to register following callbacks:
   – ErrorCallback : callback for Error
   – MspInitCallback : DCMIPP MspInit.
   – MspDeInitCallback : DCMIPP MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.
3. Function HAL_DCMIPP_PIPE_RegisterCallback() allows to register following callbacks:
   – PIPE_FrameEventCallback : callback for Pipe Frame Event.
   – PIPE_VsyncEventCallback : callback for Pipe Vsync Event.
   – PIPE_LineEventCallback : callback for Pipe Line Event.
   – PIPE_LimitEventCallback : callback for Pipe Limit Event.
   – PIPE_ErrorCallback : callback for Pipe Error This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.
4. Use function HAL_DCMIPP_UnRegisterCallback() to reset a callback to the default weak (surcharged) function. HAL_DCMIPP_UnRegisterCallback() takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks:
   – ErrorCallback : callback for Error
   – MspInitCallback : DCMIPP MspInit.
   – MspDeInitCallback : DCMIPP MspDeInit.
5. Use function HAL_DCMIPP_PIPE_UnRegisterCallback() to reset a pipe callback to the default weak (surcharged) function. HAL_DCMIPP_PIPE_UnRegisterCallback() takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks:
   – PIPE_FrameEventCallback : callback for Pipe Frame Event.
   – PIPE_VsyncEventCallback : callback for Pipe Vsync Event.
   – PIPE_LineEventCallback : callback for Pipe Line Event.
   – PIPE_LimitEventCallback : callback for Pipe Limit Event.
   – PIPE_ErrorCallback : callback for Pipe Error

6. By default, after the HAL_DCMIPP_Init and if the state is HAL_DCMIPP_STATE_RESET all callbacks are reset to the corresponding legacy weak (surcharged) functions: examples PipeFrameEventCallback(), PipeVsyncEventCallback() Exception done for MspInit and MspDeInit callbacks that are respectively reset to the legacy weak (surcharged) functions in the HAL_DCMIPP_Init and HAL_DCMIPP_DeInit only when these callbacks are null (not registered beforehand) If not, MspInit or MspDeInit are not null, the HAL_DCMIPP_Init and HAL_DCMIPP_DeInit keep and use the user MspInit/MspDeInit callbacks (registered beforehand).
7. Callbacks can be registered/unregistered in READY state only. Exception done for MspInit/MspDeInit callbacks that can be registered/unregistered in READY or RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_DCMIPP_RegisterCallback before calling HAL_DCMIPP_DeInit or HAL_DCMIPP_Init function.
8. When The compilation define USE_HAL_DCMIPP_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak (surcharged) callbacks are used.

Note: You can refer to the DCMIPP HAL driver header file for more useful macros

16.2.2 Detailed description of functions

HAL_DCMIPP_Init

Function name

HAL_StatusTypeDef HAL_DCMIPP_Init (DCMIPP_HandleTypeDef * hdcmipp)

Function description

HAL_DCMIPP_DeInit

Function name

HAL_StatusTypeDef HAL_DCMIPP_DeInit (DCMIPP_HandleTypeDef * hdcmipp)

Function description

HAL_DCMIPP_MspInit

Function name

void HAL_DCMIPP_MspInit (DCMIPP_HandleTypeDef * hdcmipp)

Function description

HAL_DCMIPP_MspDeInit

Function name

void HAL_DCMIPP_MspDeInit (DCMIPP_HandleTypeDef * hdcmipp)

Function description

HAL_DCMIPP_PARALLEL_SetConfig

Function name

HAL_StatusTypeDef HAL_DCMIPP_PARALLEL_SetConfig (DCMIPP_HandleTypeDef * hdcmipp, const DCMIPP_ParallelConfTypeDef * pParallelConfig)

Function description

HAL_DCMIPP_CSI_PIPE_SetConfig

Function name

HAL_StatusTypeDef HAL_DCMIPP_CSI_PIPE_SetConfig (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe, const DCMIPP_CSI_PIPE_ConfTypeDef * pCSI_PipeConfig)

Function description

HAL_DCMIPP_CSI_SetConfig

Function name

HAL_StatusTypeDef HAL_DCMIPP_CSI_SetConfig (DCMIPP_HandleTypeDef * hdcmipp, const DCMIPP_CSI_ConfTypeDef * pCSI_Config)

Function description

HAL_DCMIPP_CSI_SetVCFilteringConfig

Function name

HAL_StatusTypeDef HAL_DCMIPP_CSI_SetVCFilteringConfig (DCMIPP_HandleTypeDef * hdcmipp, uint32_t VirtualChannel, const DCMIPP_CSI_VCFilteringConfTypeDef * pVCFilteringConfig)

Function description

HAL_DCMIPP_CSI_SetVCConfig

Function name

HAL_StatusTypeDef HAL_DCMIPP_CSI_SetVCConfig (DCMIPP_HandleTypeDef * hdcmipp, uint32_t VirtualChannel, uint32_t DataTypeFormat)

Function description

HAL_DCMIPP_PIPE_SetConfig

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_SetConfig (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe, const DCMIPP_PipeConfTypeDef * pPipeConfig)

Function description

HAL_DCMIPP_SetIPPlugConfig

Function name

HAL_StatusTypeDef HAL_DCMIPP_SetIPPlugConfig (DCMIPP_HandleTypeDef * hdcmipp, const DCMIPP_IPPlugConfTypeDef * pIPPlugConfig)

Function description

HAL_DCMIPP_PIPE_Start

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_Start (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe, uint32_t DstAddress, uint32_t CaptureMode)

Function description

HAL_DCMIPP_PIPE_DoubleBufferStart

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_DoubleBufferStart (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe, uint32_t DstAddress0, uint32_t DstAddress1, uint32_t CaptureMode)

Function description

HAL_DCMIPP_CSI_PIPE_Start

Function name

HAL_StatusTypeDef HAL_DCMIPP_CSI_PIPE_Start (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe, uint32_t VirtualChannel, uint32_t DstAddress, uint32_t CaptureMode)

Function description

HAL_DCMIPP_CSI_PIPE_DoubleBufferStart

Function name

HAL_StatusTypeDef HAL_DCMIPP_CSI_PIPE_DoubleBufferStart (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe, uint32_t VirtualChannel, uint32_t DstAddress0, uint32_t DstAddress1, uint32_t CaptureMode)

Function description

HAL_DCMIPP_PIPE_SemiPlanarStart

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_SemiPlanarStart (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe, DCMIPP_SemiPlanarDstAddressTypeDef * pSemiPlanarDstAddress, uint32_t CaptureMode)

Function description

HAL_DCMIPP_PIPE_SemiPlanarDoubleBufferStart

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_SemiPlanarDoubleBufferStart (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe, DCMIPP_SemiPlanarDstAddressTypeDef * pSemiPlanarDstAddress0, DCMIPP_SemiPlanarDstAddressTypeDef * pSemiPlanarDstAddress1, uint32_t CaptureMode)

Function description

HAL_DCMIPP_PIPE_FullPlanarStart

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_FullPlanarStart (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe, DCMIPP_FullPlanarDstAddressTypeDef * pFullPlanarDstAddress, uint32_t CaptureMode)

Function description

HAL_DCMIPP_PIPE_FullPlanarDoubleBufferStart

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_FullPlanarDoubleBufferStart (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe, DCMIPP_FullPlanarDstAddressTypeDef * pFullPlanarDstAddress0, DCMIPP_FullPlanarDstAddressTypeDef * pFullPlanarDstAddress1, uint32_t CaptureMode)

Function description

HAL_DCMIPP_CSI_PIPE_SemiPlanarStart

Function name

HAL_StatusTypeDef HAL_DCMIPP_CSI_PIPE_SemiPlanarStart (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe, uint32_t VirtualChannel, DCMIPP_SemiPlanarDstAddressTypeDef * pSemiPlanarDstAddress, uint32_t CaptureMode)

Function description

HAL_DCMIPP_CSI_PIPE_SemiPlanarDoubleBufferStart

Function name

HAL_StatusTypeDef HAL_DCMIPP_CSI_PIPE_SemiPlanarDoubleBufferStart (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe, uint32_t VirtualChannel, DCMIPP_SemiPlanarDstAddressTypeDef * pSemiPlanarDstAddress0, DCMIPP_SemiPlanarDstAddressTypeDef * pSemiPlanarDstAddress1, uint32_t CaptureMode)

Function description

HAL_DCMIPP_CSI_PIPE_FullPlanarStart

Function name

HAL_StatusTypeDef HAL_DCMIPP_CSI_PIPE_FullPlanarStart (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe, uint32_t VirtualChannel, DCMIPP_FullPlanarDstAddressTypeDef * pFullPlanarDstAddress, uint32_t CaptureMode)

Function description

HAL_DCMIPP_CSI_PIPE_FullPlanarDoubleBufferStart

Function name

HAL_StatusTypeDef HAL_DCMIPP_CSI_PIPE_FullPlanarDoubleBufferStart (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe, uint32_t VirtualChannel, DCMIPP_FullPlanarDstAddressTypeDef * pFullPlanarDstAddress0, DCMIPP_FullPlanarDstAddressTypeDef * pFullPlanarDstAddress1, uint32_t CaptureMode)

Function description

HAL_DCMIPP_PIPE_Stop

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_Stop (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe)

Function description

HAL_DCMIPP_CSI_PIPE_Stop

Function name

HAL_StatusTypeDef HAL_DCMIPP_CSI_PIPE_Stop (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe, uint32_t VirtualChannel)

Function description

HAL_DCMIPP_PIPE_Suspend

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_Suspend (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe)

Function description

HAL_DCMIPP_PIPE_Resume

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_Resume (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe)

Function description

HAL_DCMIPP_IRQHandler

Function name

void HAL_DCMIPP_IRQHandler (DCMIPP_HandleTypeDef * hdcmipp)

Function description

HAL_DCMIPP_CSI_IRQHandler

Function name

void HAL_DCMIPP_CSI_IRQHandler (DCMIPP_HandleTypeDef * hdcmipp)

Function description

HAL_DCMIPP_PIPE_FrameEventCallback

Function name

void HAL_DCMIPP_PIPE_FrameEventCallback (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe)

Function description

HAL_DCMIPP_PIPE_VsyncEventCallback

Function name

void HAL_DCMIPP_PIPE_VsyncEventCallback (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe)

Function description

HAL_DCMIPP_PIPE_LineEventCallback

Function name

void HAL_DCMIPP_PIPE_LineEventCallback (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe)

Function description

HAL_DCMIPP_PIPE_LimitEventCallback

Function name

void HAL_DCMIPP_PIPE_LimitEventCallback (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe)

Function description

HAL_DCMIPP_PIPE_ErrorCallback

Function name

void HAL_DCMIPP_PIPE_ErrorCallback (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe)

Function description

HAL_DCMIPP_ErrorCallback

Function name

void HAL_DCMIPP_ErrorCallback (DCMIPP_HandleTypeDef * hdcmipp)

Function description

HAL_DCMIPP_CSI_StartOfFrameEventCallback

Function name

void HAL_DCMIPP_CSI_StartOfFrameEventCallback (DCMIPP_HandleTypeDef * hdcmipp, uint32_t VirtualChannel)

Function description

HAL_DCMIPP_CSI_EndOfFrameEventCallback

Function name

void HAL_DCMIPP_CSI_EndOfFrameEventCallback (DCMIPP_HandleTypeDef * hdcmipp, uint32_t VirtualChannel)

Function description

HAL_DCMIPP_CSI_TimerCounterEventCallback

Function name

void HAL_DCMIPP_CSI_TimerCounterEventCallback (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Timer)

Function description

HAL_DCMIPP_CSI_LineByteEventCallback

Function name

void HAL_DCMIPP_CSI_LineByteEventCallback (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Counter)

Function description

HAL_DCMIPP_CSI_LineErrorCallback

Function name

void HAL_DCMIPP_CSI_LineErrorCallback (DCMIPP_HandleTypeDef * hdcmipp, uint32_t DataLane)

Function description

HAL_DCMIPP_CSI_ClockChangerFifoFullEventCallback

Function name

void HAL_DCMIPP_CSI_ClockChangerFifoFullEventCallback (DCMIPP_HandleTypeDef * hdcmipp)

Function description

HAL_DCMIPP_CSI_ShortPacketDetectionEventCallback

Function name

void HAL_DCMIPP_CSI_ShortPacketDetectionEventCallback (DCMIPP_HandleTypeDef * hdcmipp)

Function description

HAL_DCMIPP_RegisterCallback

Function name

HAL_StatusTypeDef HAL_DCMIPP_RegisterCallback (DCMIPP_HandleTypeDef * hdcmipp, HAL_DCMIPP_CallbackIDTypeDef CallbackID, pDCMIPP_CallbackTypeDef pCallback)

Function description

HAL_DCMIPP_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_DCMIPP_UnRegisterCallback (DCMIPP_HandleTypeDef * hdcmipp, HAL_DCMIPP_CallbackIDTypeDef CallbackID)

Function description

HAL_DCMIPP_PIPE_RegisterCallback

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_RegisterCallback (DCMIPP_HandleTypeDef * hdcmipp, HAL_DCMIPP_PIPE_CallbackIDTypeDef CallbackID, pDCMIPP_PIPE_CallbackTypeDef pCallback)

Function description

HAL_DCMIPP_PIPE_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_UnRegisterCallback (DCMIPP_HandleTypeDef * hdcmipp, HAL_DCMIPP_PIPE_CallbackIDTypeDef CallbackID)

Function description

HAL_DCMIPP_PIPE_SetBytesDecimationConfig

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_SetBytesDecimationConfig (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe, uint32_t SelectStart, uint32_t SelectMode)

Function description

HAL_DCMIPP_PIPE_SetLinesDecimationConfig

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_SetLinesDecimationConfig (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe, uint32_t SelectStart, uint32_t SelectMode)

Function description

HAL_DCMIPP_PIPE_SetISPDecimationConfig

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_SetISPDecimationConfig (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe, const DCMIPP_DecimationConfTypeDef * pDecConfig)

Function description

HAL_DCMIPP_PIPE_EnableISPDecimation

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_EnableISPDecimation (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe)

Function description

HAL_DCMIPP_PIPE_DisableISPDecimation

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_DisableISPDecimation (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe)

Function description

HAL_DCMIPP_PIPE_SetDecimationConfig

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_SetDecimationConfig (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe, const DCMIPP_DecimationConfTypeDef * pDecConfig)

Function description

HAL_DCMIPP_PIPE_EnableDecimation

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_EnableDecimation (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe)

Function description

HAL_DCMIPP_PIPE_DisableDecimation

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_DisableDecimation (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe)

Function description

HAL_DCMIPP_PIPE_SetCropConfig

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_SetCropConfig (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe, const DCMIPP_CropConfTypeDef * pCropConfig)

Function description

HAL_DCMIPP_PIPE_EnableCrop

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_EnableCrop (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe)

Function description

HAL_DCMIPP_PIPE_DisableCrop

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_DisableCrop (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe)

Function description

HAL_DCMIPP_PIPE_EnableLineEvent

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_EnableLineEvent (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe, uint32_t Line)

Function description

HAL_DCMIPP_PIPE_DisableLineEvent

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_DisableLineEvent (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe)

Function description

HAL_DCMIPP_PIPE_EnableLimitEvent

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_EnableLimitEvent (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe, uint32_t Limit)

Function description

HAL_DCMIPP_PIPE_DisableLimitEvent

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_DisableLimitEvent (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe)

Function description

HAL_DCMIPP_PIPE_SetISPStatisticExtractionConfig

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_SetISPStatisticExtractionConfig (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe, uint8_t ModuleID, const DCMIPP_StatisticExtractionConfTypeDef * pStatisticExtractionConfig)

Function description

HAL_DCMIPP_PIPE_SetISPAreaStatisticExtractionConfig

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_SetISPAreaStatisticExtractionConfig (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe, const DCMIPP_StatisticExtractionAreaConfTypeDef * pStatisticExtractionAreaConfig)

Function description

HAL_DCMIPP_PIPE_EnableISPStatisticExtraction

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_EnableISPStatisticExtraction (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe, uint8_t ModuleID)

Function description

HAL_DCMIPP_PIPE_DisableISPStatisticExtraction

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_DisableISPStatisticExtraction (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe, uint8_t ModuleID)

Function description

HAL_DCMIPP_PIPE_EnableISPAreaStatisticExtraction

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_EnableISPAreaStatisticExtraction (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe)

Function description

HAL_DCMIPP_PIPE_DisableISPAreaStatisticExtraction

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_DisableISPAreaStatisticExtraction (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe)

Function description

HAL_DCMIPP_PIPE_SetDownsizeConfig

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_SetDownsizeConfig (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe, const DCMIPP_DownsizeTypeDef * pDownsizeConfig)

Function description

HAL_DCMIPP_PIPE_DisableDownsize

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_DisableDownsize (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe)

Function description

HAL_DCMIPP_PIPE_EnableDownsize

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_EnableDownsize (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe)

Function description

HAL_DCMIPP_PIPE_SetISPBlackLevelCalibrationConfig

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_SetISPBlackLevelCalibrationConfig (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe, const DCMIPP_BlackLevelConfTypeDef * pBlackLevelConfig)

Function description

HAL_DCMIPP_PIPE_DisableISPBlackLevelCalibration

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_DisableISPBlackLevelCalibration (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe)

Function description

HAL_DCMIPP_PIPE_EnableISPBlackLevelCalibration

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_EnableISPBlackLevelCalibration (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe)

Function description

HAL_DCMIPP_PIPE_EnableGammaConversion

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_EnableGammaConversion (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe)

Function description

HAL_DCMIPP_PIPE_DisableGammaConversion

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_DisableGammaConversion (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe)

Function description

HAL_DCMIPP_PIPE_IsEnabledGammaConversion

Function name

uint32_t HAL_DCMIPP_PIPE_IsEnabledGammaConversion (const DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe)

Function description

HAL_DCMIPP_PIPE_SetISPBadPixelRemovalConfig

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_SetISPBadPixelRemovalConfig (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe, uint32_t Strength)

Function description

HAL_DCMIPP_PIPE_DisableISPBadPixelRemoval

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_DisableISPBadPixelRemoval (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe)

Function description

HAL_DCMIPP_PIPE_EnableISPBadPixelRemoval

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_EnableISPBadPixelRemoval (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe)

Function description

HAL_DCMIPP_PIPE_SetISPCtrlContrastConfig

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_SetISPCtrlContrastConfig (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe, const DCMIPP_ContrastConfTypeDef * pContrastConfig)

Function description

HAL_DCMIPP_PIPE_EnableISPCtrlContrast

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_EnableISPCtrlContrast (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe)

Function description

HAL_DCMIPP_PIPE_DisableISPCtrlContrast

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_DisableISPCtrlContrast (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe)

Function description

HAL_DCMIPP_PIPE_SetRegionOfInterestConfig

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_SetRegionOfInterestConfig (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe, const DCMIPP_RegionOfInterestConfTypeDef * pROIConfig)

Function description

HAL_DCMIPP_PIPE_EnableRegionOfInterest

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_EnableRegionOfInterest (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe, uint32_t Region)

Function description

HAL_DCMIPP_PIPE_DisableRegionOfInterest

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_DisableRegionOfInterest (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe, uint32_t Region)

Function description

HAL_DCMIPP_PIPE_SetLineWrappingConfig

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_SetLineWrappingConfig (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe, uint32_t AddressWrap)

Function description

HAL_DCMIPP_PIPE_EnableLineWrapping

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_EnableLineWrapping (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe)

Function description

HAL_DCMIPP_PIPE_DisableLineWrapping

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_DisableLineWrapping (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe)

Function description

HAL_DCMIPP_PIPE_SetISPExposureConfig

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_SetISPExposureConfig (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe, const DCMIPP_ExposureConfTypeDef * pExposureConfig)

Function description

HAL_DCMIPP_PIPE_EnableISPExposure

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_EnableISPExposure (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe)

Function description

HAL_DCMIPP_PIPE_DisableISPExposure

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_DisableISPExposure (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe)

Function description

HAL_DCMIPP_PIPE_SetISPRemovalStatisticConfig

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_SetISPRemovalStatisticConfig (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe, uint32_t NbFirstLines, uint32_t NbLastLines)

Function description

HAL_DCMIPP_PIPE_EnableISPRemovalStatistic

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_EnableISPRemovalStatistic (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe)

Function description

HAL_DCMIPP_PIPE_DisableISPRemovalStatistic

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_DisableISPRemovalStatistic (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe)

Function description

HAL_DCMIPP_PIPE_SetISPRawBayer2RGBConfig

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_SetISPRawBayer2RGBConfig (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe, const DCMIPP_RawBayer2RGBConfTypeDef * pRawBayer2RGBConfig)

Function description

HAL_DCMIPP_PIPE_DisableISPRawBayer2RGB

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_DisableISPRawBayer2RGB (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe)

Function description

HAL_DCMIPP_PIPE_EnableISPRawBayer2RGB

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_EnableISPRawBayer2RGB (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe)

Function description

HAL_DCMIPP_PIPE_SetYUVConversionConfig

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_SetYUVConversionConfig (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe, const DCMIPP_ColorConversionConfTypeDef * pColorConversionConfig)

Function description

HAL_DCMIPP_PIPE_DisableYUVConversion

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_DisableYUVConversion (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe)

Function description

HAL_DCMIPP_PIPE_EnableYUVConversion

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_EnableYUVConversion (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe)

Function description

HAL_DCMIPP_PIPE_SetISPColorConversionConfig

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_SetISPColorConversionConfig (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe, const DCMIPP_ColorConversionConfTypeDef * pColorConversionConfig)

Function description

HAL_DCMIPP_PIPE_EnableISPColorConversion

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_EnableISPColorConversion (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe)

Function description

HAL_DCMIPP_PIPE_DisableISPColorConversion

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_DisableISPColorConversion (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe)

Function description

HAL_DCMIPP_PIPE_GetISPDecimationConfig

Function name

void HAL_DCMIPP_PIPE_GetISPDecimationConfig (const DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe, DCMIPP_DecimationConfTypeDef * pDecConfig)

Function description

HAL_DCMIPP_PIPE_GetISPBlackLevelCalibrationConfig

Function name

void HAL_DCMIPP_PIPE_GetISPBlackLevelCalibrationConfig (const DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe, DCMIPP_BlackLevelConfTypeDef * pBlackLevelConfig)

Function description

HAL_DCMIPP_PIPE_GetISPBadPixelRemovalConfig

Function name

uint32_t HAL_DCMIPP_PIPE_GetISPBadPixelRemovalConfig (const DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe)

Function description

HAL_DCMIPP_PIPE_GetISPStatisticExtractionConfig

Function name

void HAL_DCMIPP_PIPE_GetISPStatisticExtractionConfig (const DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe, uint8_t ModuleID, DCMIPP_StatisticExtractionConfTypeDef * pStatisticExtractionConfig)

Function description

HAL_DCMIPP_PIPE_GetISPAreaStatisticExtractionConfig

Function name

void HAL_DCMIPP_PIPE_GetISPAreaStatisticExtractionConfig (const DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe, DCMIPP_StatisticExtractionAreaConfTypeDef * pStatisticExtractionAreaConfig)

Function description

HAL_DCMIPP_PIPE_GetISPCtrlContrastConfig

Function name

void HAL_DCMIPP_PIPE_GetISPCtrlContrastConfig (const DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe, DCMIPP_ContrastConfTypeDef * pContrastConfig)

Function description

HAL_DCMIPP_PIPE_GetISPExposureConfig

Function name

void HAL_DCMIPP_PIPE_GetISPExposureConfig (const DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe, DCMIPP_ExposureConfTypeDef * pExposureConfig)

Function description

HAL_DCMIPP_PIPE_GetISPRawBayer2RGBConfig

Function name

void HAL_DCMIPP_PIPE_GetISPRawBayer2RGBConfig (const DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe, DCMIPP_RawBayer2RGBConfTypeDef * pRawBayer2RGBConfig)

Function description

HAL_DCMIPP_PIPE_GetISPColorConversionConfig

Function name

void HAL_DCMIPP_PIPE_GetISPColorConversionConfig (const DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe, DCMIPP_ColorConversionConfTypeDef * pColorConversionConfig)

Function description

HAL_DCMIPP_PIPE_GetISPRemovalStatisticConfig

Function name

void HAL_DCMIPP_PIPE_GetISPRemovalStatisticConfig (const DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe, uint32_t * NbFirstLines, uint32_t * NbLastLines)

Function description

HAL_DCMIPP_PIPE_IsEnabledISPRemovalStatistic

Function name

uint32_t HAL_DCMIPP_PIPE_IsEnabledISPRemovalStatistic (const DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe)

Function description

HAL_DCMIPP_PIPE_IsEnabledISPBadPixelRemoval

Function name

uint32_t HAL_DCMIPP_PIPE_IsEnabledISPBadPixelRemoval (const DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe)

Function description

HAL_DCMIPP_PIPE_IsEnabledISPDecimation

Function name

uint32_t HAL_DCMIPP_PIPE_IsEnabledISPDecimation (const DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe)

Function description

HAL_DCMIPP_PIPE_IsEnabledISPBlackLevelCalibration

Function name

uint32_t HAL_DCMIPP_PIPE_IsEnabledISPBlackLevelCalibration (const DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe)

Function description

HAL_DCMIPP_PIPE_IsEnabledISPExposure

Function name

uint32_t HAL_DCMIPP_PIPE_IsEnabledISPExposure (const DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe)

Function description

HAL_DCMIPP_PIPE_IsEnabledISPRawBayer2RGB

Function name

uint32_t HAL_DCMIPP_PIPE_IsEnabledISPRawBayer2RGB (const DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe)

Function description

HAL_DCMIPP_PIPE_IsEnabledISPColorConversion

Function name

uint32_t HAL_DCMIPP_PIPE_IsEnabledISPColorConversion (const DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe)

Function description

HAL_DCMIPP_PIPE_IsEnabledISPCtrlContrast

Function name

uint32_t HAL_DCMIPP_PIPE_IsEnabledISPCtrlContrast (const DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe)

Function description

HAL_DCMIPP_PIPE_IsEnabledISPStatisticExtraction

Function name

uint32_t HAL_DCMIPP_PIPE_IsEnabledISPStatisticExtraction (const DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe, uint8_t ModuleID)

Function description

HAL_DCMIPP_PIPE_IsEnabledISPAreaStatisticExtraction

Function name

uint32_t HAL_DCMIPP_PIPE_IsEnabledISPAreaStatisticExtraction (const DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe)

Function description

HAL_DCMIPP_PIPE_SetFrameRate

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_SetFrameRate (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe, uint32_t FrameRate)

Function description

HAL_DCMIPP_PIPE_SetCaptureMode

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_SetCaptureMode (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe, uint32_t CaptureMode)

Function description

HAL_DCMIPP_PIPE_EnableCapture

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_EnableCapture (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe)

Function description

HAL_DCMIPP_PIPE_SetMemoryAddress

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_SetMemoryAddress (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe, uint32_t Memory, uint32_t DstAddress)

Function description

HAL_DCMIPP_PIPE_PARALLEL_SetInputPixelFormat

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_PARALLEL_SetInputPixelFormat (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe, uint32_t InputPixelFormat)

Function description

HAL_DCMIPP_PIPE_SetPitch

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_SetPitch (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe, uint32_t LinePitch)

Function description

HAL_DCMIPP_PIPE_SetPixelPackerFormat

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_SetPixelPackerFormat (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe, uint32_t PixelPackerFormat)

Function description

HAL_DCMIPP_PIPE_CSI_EnableShare

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_CSI_EnableShare (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe)

Function description

HAL_DCMIPP_PIPE_CSI_DisableShare

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_CSI_DisableShare (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe)

Function description

HAL_DCMIPP_PIPE_CSI_ForceDataTypeFormat

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_CSI_ForceDataTypeFormat (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe, uint32_t DataTypeFormat)

Function description

HAL_DCMIPP_PIPE_CSI_SetDTMode

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_CSI_SetDTMode (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe, uint32_t DataTypeMode)

Function description

HAL_DCMIPP_PIPE_CSI_SetDTSelection

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_CSI_SetDTSelection (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe, uint32_t DataTypeID, uint32_t DataType)

Function description

HAL_DCMIPP_PIPE_CSI_EnableHeader

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_CSI_EnableHeader (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe)

Function description

HAL_DCMIPP_PIPE_CSI_DisableHeader

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_CSI_DisableHeader (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe)

Function description

HAL_DCMIPP_PARALLEL_SetSyncUnmask

Function name

HAL_StatusTypeDef HAL_DCMIPP_PARALLEL_SetSyncUnmask (DCMIPP_HandleTypeDef * hdcmipp, const DCMIPP_EmbeddedSyncUnmaskTypeDef * SyncUnmask)

Function description

HAL_DCMIPP_CSI_SetLineByteCounterConfig

Function name

HAL_StatusTypeDef HAL_DCMIPP_CSI_SetLineByteCounterConfig (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Counter, const DCMIPP_CSI_LineByteCounterConfTypeDef * pLineByteConfig)

Function description

HAL_DCMIPP_CSI_EnableLineByteCounter

Function name

HAL_StatusTypeDef HAL_DCMIPP_CSI_EnableLineByteCounter (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Counter)

Function description

HAL_DCMIPP_CSI_DisableLineByteCounter

Function name

HAL_StatusTypeDef HAL_DCMIPP_CSI_DisableLineByteCounter (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Counter)

Function description

HAL_DCMIPP_CSI_SetTimerConfig

Function name

HAL_StatusTypeDef HAL_DCMIPP_CSI_SetTimerConfig (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Timer, const DCMIPP_CSI_TimerConfTypeDef * TimerConfig)

Function description

HAL_DCMIPP_CSI_EnableTimer

Function name

HAL_StatusTypeDef HAL_DCMIPP_CSI_EnableTimer (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Timer)

Function description

HAL_DCMIPP_CSI_DisableTimer

Function name

HAL_StatusTypeDef HAL_DCMIPP_CSI_DisableTimer (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Timer)

Function description

HAL_DCMIPP_CSI_SetWatchdogCounterConfig

Function name

HAL_StatusTypeDef HAL_DCMIPP_CSI_SetWatchdogCounterConfig (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Counter)

Function description

HAL_DCMIPP_PIPE_EnableComponentsSwap

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_EnableComponentsSwap (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe)

Function description

HAL_DCMIPP_PIPE_DisableComponentsSwap

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_DisableComponentsSwap (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe)

Function description

HAL_DCMIPP_PIPE_DisableRedBlueSwap

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_DisableRedBlueSwap (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe)

Function description

HAL_DCMIPP_PIPE_EnableRedBlueSwap

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_EnableRedBlueSwap (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe)

Function description

HAL_DCMIPP_PIPE_DisableYUVSwap

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_DisableYUVSwap (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe)

Function description

HAL_DCMIPP_PIPE_EnableYUVSwap

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_EnableYUVSwap (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe)

Function description

HAL_DCMIPP_PIPE_SetFrameCounterConfig

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_SetFrameCounterConfig (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe)

Function description

HAL_DCMIPP_PIPE_ResetFrameCounter

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_ResetFrameCounter (DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe)

Function description

HAL_DCMIPP_PIPE_ReadFrameCounter

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_ReadFrameCounter (const DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe, uint32_t * pCounter)

Function description

HAL_DCMIPP_PIPE_GetDataCounter

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_GetDataCounter (const DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe, uint32_t * pCounter)

Function description

HAL_DCMIPP_PIPE_GetISPRemovedBadPixelCounter

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_GetISPRemovedBadPixelCounter (const DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe, uint32_t * pCounter)

Function description

HAL_DCMIPP_PIPE_GetISPAccumulatedStatisticsCounter

Function name

HAL_StatusTypeDef HAL_DCMIPP_PIPE_GetISPAccumulatedStatisticsCounter (const DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe, uint8_t ModuleID, uint32_t * pCounter)

Function description

HAL_DCMIPP_GetMode

Function name

uint32_t HAL_DCMIPP_GetMode (const DCMIPP_HandleTypeDef * hdcmipp)

Function description

HAL_DCMIPP_PIPE_GetMemoryAddress

Function name

uint32_t HAL_DCMIPP_PIPE_GetMemoryAddress (const DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe, uint32_t MemoryAddress)

Function description

HAL_DCMIPP_GetState

Function name

HAL_DCMIPP_StateTypeDef HAL_DCMIPP_GetState (const DCMIPP_HandleTypeDef * hdcmipp)

Function description

HAL_DCMIPP_PIPE_GetState

Function name

HAL_DCMIPP_PipeStateTypeDef HAL_DCMIPP_PIPE_GetState (const DCMIPP_HandleTypeDef * hdcmipp, uint32_t Pipe)

Function description

HAL_DCMIPP_GetError

Function name

uint32_t HAL_DCMIPP_GetError (const DCMIPP_HandleTypeDef * hdcmipp)

Function description

16.3 DCMIPP Firmware driver defines

The following section lists the various define and macros of the module.

16.3.1 DCMIPP

DCMIPP

DCMIPP Bad Pixel Removal Strength

DCMIPP_BAD_PXL_REM_SRENGTH_0

The filter is fairly tolerant: only a few pixels are declared bad, due to their high noise

DCMIPP_BAD_PXL_REM_SRENGTH_1

Bad Pixel detection strength 1

DCMIPP_BAD_PXL_REM_SRENGTH_2

Bad Pixel detection strength 2

DCMIPP_BAD_PXL_REM_SRENGTH_3

Bad Pixel detection strength 3

DCMIPP_BAD_PXL_REM_SRENGTH_4

Bad Pixel detection strength 4

DCMIPP_BAD_PXL_REM_SRENGTH_5

Bad Pixel detection strength 5

DCMIPP_BAD_PXL_REM_SRENGTH_6

Bad Pixel detection strength 6

DCMIPP_BAD_PXL_REM_SRENGTH_7

Bad Pixel detection strength 7

DCMIPP Byte Select Mode

DCMIPP_BSM_ALL

Interface captures all received data

DCMIPP_BSM_DATA_OUT_2

Interface captures 1 data out of 2

DCMIPP_BSM_BYTE_OUT_4

Interface captures 1 byte out of 4

DCMIPP_BSM_2BYTE_OUT_4

Interface captures 2 byte out of 4

DCMIPP Byte Start Mode

DCMIPP_OEBS_ODD

Interface captures first data (byte or double byte) from the frame/line start,second one being dropped

DCMIPP_OEBS_EVEN

Interface captures second data (byte or double byte) from the frame/line start, first one is dropped

DCMIPP Capture Mode

DCMIPP_MODE_CONTINUOUS

DCMIPP continuous mode (preview)

DCMIPP_MODE_SNAPSHOT

DCMIPP snapshot mode

DCMIPP Crop Area

DCMIPP_POSITIVE_AREA

Positive Area chosen for crop

DCMIPP_NEGATIVE_AREA

Negative Area chosen for crop

DCMIPP CSI Counter

DCMIPP_CSI_COUNTER0

DCMIPP CSI Counter 0

DCMIPP_CSI_COUNTER1

DCMIPP CSI Counter 1

DCMIPP_CSI_COUNTER2

DCMIPP CSI Counter 2

DCMIPP_CSI_COUNTER3

DCMIPP CSI Counter 3

DCMIPP CSI Data Lane

DCMIPP_CSI_DATA_LANE0

DCMIPP CSI Data Lane 0

DCMIPP_CSI_DATA_LANE1

DCMIPP CSI Data Lane 1

DCMIPP CSI Data Lane Mapping

DCMIPP_CSI_PHYSICAL_DATA_LANES

Physical data lane 0 connected to logical data lane 0 and Physical data lane 1 connected to logical data lane 1

DCMIPP_CSI_INVERTED_DATA_LANES

Physical data lane 1 connected to logical data lane 0 and Physical data lane 0 connected to logical data lane 1

DCMIPP CSI Data Type Format

DCMIPP_CSI_DT_BPP6

DCMIPP CSI Data Type Format 6 bit words

DCMIPP_CSI_DT_BPP7

DCMIPP CSI Data Type Format 7 bit words

DCMIPP_CSI_DT_BPP8

DCMIPP CSI Data Type Format 8 bit words

DCMIPP_CSI_DT_BPP10

DCMIPP CSI Data Type Format 10 bit words

DCMIPP_CSI_DT_BPP12

DCMIPP CSI Data Type Format 12 bit words

DCMIPP_CSI_DT_BPP14

DCMIPP CSI Data Type Format 14 bit words

DCMIPP_CSI_DT_BPP16

DCMIPP CSI Data Type Format 16 bit words

DCMIPP CSI Data Type ID

DCMIPP_CSI_DT1

DCMIPP Data Type ID 1

DCMIPP_CSI_DT2

DCMIPP Data Type ID 2

DCMIPP_CSI_DT3

DCMIPP Data Type ID 3

DCMIPP_CSI_DT4

DCMIPP Data Type ID 4

DCMIPP_CSI_DT5

DCMIPP Data Type ID 5

DCMIPP_CSI_DT6

DCMIPP Data Type ID 6

DCMIPP_CSI_DT7

DCMIPP Data Type ID 7

DCMIPP CSI Interrupt Flags

DCMIPP_CSI_FLAG_SYNCERR

Synchronization error flag

DCMIPP_CSI_FLAG_WDERR

Watchdog error flag

DCMIPP_CSI_FLAG_SPKTERR

Shorter packet than expected error flag

DCMIPP_CSI_FLAG_IDERR

Data ID information error flag

DCMIPP_CSI_FLAG_CECCERR

Corrected ECC error flag

DCMIPP_CSI_FLAG_ECCERR

ECC Error flag

DCMIPP_CSI_FLAG_CRCERR

CRC Error flag

DCMIPP_CSI_FLAG_CCFIFO

Clock changer FIFO full event flag

DCMIPP_CSI_FLAG_SPKT

Short packet detection flag

DCMIPP_CSI_FLAG_EOF3

EOF on virtual channel 3 flag

DCMIPP_CSI_FLAG_EOF2

EOF on virtual channel 2 flag

DCMIPP_CSI_FLAG_EOF1

EOF on virtual channel 1 flag

DCMIPP_CSI_FLAG_EOF0

EOF on virtual channel 0 flag

DCMIPP_CSI_FLAG_SOF3

SOF on virtual channel 3 flag

DCMIPP_CSI_FLAG_SOF2

SOF on virtual channel 2 flag

DCMIPP_CSI_FLAG_SOF1

SOF on virtual channel 1 flag

DCMIPP_CSI_FLAG_SOF0

SOF on virtual channel 0 flag

DCMIPP_CSI_FLAG_TIM3

Timer 3 counter value reached flag

DCMIPP_CSI_FLAG_TIM2

Timer 2 counter value reached flag

DCMIPP_CSI_FLAG_TIM1

Timer 1 counter value reached flag

DCMIPP_CSI_FLAG_TIM0

Timer 0 counter value reached flag

DCMIPP_CSI_FLAG_LB3

Line/byte event 3 reached flag

DCMIPP_CSI_FLAG_LB2

Line/byte event 2 reached flag

DCMIPP_CSI_FLAG_LB1

Line/byte event 1 reached flag

DCMIPP_CSI_FLAG_LB0

Line/byte event 0 reached flag

DCMIPP_CSI_FLAG_ECTRLDL1

Error control on data line 1 flag

DCMIPP_CSI_FLAG_ESYNCESCDL1

Low-power data transmission synchronization error on data line 1 flag

DCMIPP_CSI_FLAG_EESCDL1

Escape entry error on data line 1 flag

DCMIPP_CSI_FLAG_ESOTSYNCDL1

SOT synchronization error on data line 1 flag

DCMIPP_CSI_FLAG_ESOTDL1

SOT error on data line 1 flag

DCMIPP_CSI_FLAG_ECTRLDL0

Error control on data line 0 flag

DCMIPP_CSI_FLAG_ESYNCESCDL0

Low-power data transmission synchronization error on data line 0 flag

DCMIPP_CSI_FLAG_EESCDL0

Escape entry error on data line 0 flag

DCMIPP_CSI_FLAG_ESOTSYNCDL0

SOT synchronization error on data line 0 flag

DCMIPP_CSI_FLAG_ESOTDL0

SOT error on data line 0 flag

DCMIPP CSI Interrupt sources

DCMIPP_CSI_IT_SYNCERR

Synchronization error interrupt

DCMIPP_CSI_IT_WDERR

Watchdog error interrupt

DCMIPP_CSI_IT_SPKTERR

Shorter packet than expected error interrupt

DCMIPP_CSI_IT_IDERR

Data ID information error interrupt

DCMIPP_CSI_IT_CECCERR

Corrected ECC error interrupt

DCMIPP_CSI_IT_ECCERR

ECC Error interrupt

DCMIPP_CSI_IT_CRCERR

CRC Error interrupt

DCMIPP_CSI_IT_CCFIFO

Clock changer FIFO full event interrupt

DCMIPP_CSI_IT_SPKT

Short packet detection interrupt

DCMIPP_CSI_IT_EOF3

EOF on virtual channel 3 interrupt

DCMIPP_CSI_IT_EOF2

EOF on virtual channel 2 interrupt

DCMIPP_CSI_IT_EOF1

EOF on virtual channel 1 interrupt

DCMIPP_CSI_IT_EOF0

EOF on virtual channel 0 interrupt

DCMIPP_CSI_IT_SOF3

SOF on virtual channel 3 interrupt

DCMIPP_CSI_IT_SOF2

SOF on virtual channel 2 interrupt

DCMIPP_CSI_IT_SOF1

SOF on virtual channel 1 interrupt

DCMIPP_CSI_IT_SOF0

SOF on virtual channel 0 interrupt

DCMIPP_CSI_IT_TIM3

Timer 3 counter value reached interrupt

DCMIPP_CSI_IT_TIM2

Timer 2 counter value reached interrupt

DCMIPP_CSI_IT_TIM1

Timer 1 counter value reached interrupt

DCMIPP_CSI_IT_TIM0

Timer 0 counter value reached interrupt

DCMIPP_CSI_IT_LB3

Line/byte event 3 reached interrupt

DCMIPP_CSI_IT_LB2

Line/byte event 2 reached interrupt

DCMIPP_CSI_IT_LB1

Line/byte event 1 reached interrupt

DCMIPP_CSI_IT_LB0

Line/byte event 0 reached interrupt

DCMIPP_CSI_IT_ECTRLDL1

Error control on data line 1 interrupt

DCMIPP_CSI_IT_ESYNCESCDL1

Low-power data transmission synchronization error on data line 1 interrupt

DCMIPP_CSI_IT_EESCDL1

Escape entry error on data line 1 interrupt

DCMIPP_CSI_IT_ESOTSYNCDL1

SOT synchronization error on data line 1 interrupt

DCMIPP_CSI_IT_ESOTDL1

SOT error on data line 1 interrupt

DCMIPP_CSI_IT_ECTRLDL0

Error control on data line 0 interrupt

DCMIPP_CSI_IT_ESYNCESCDL0

Low-power data transmission synchronization error on data line 0 interrupt

DCMIPP_CSI_IT_EESCDL0

Escape entry error on data line 0 interrupt

DCMIPP_CSI_IT_ESOTSYNCDL0

SOT synchronization error on data line 0 interrupt

DCMIPP_CSI_IT_ESOTDL0

SOT error on data line 0 interrupt

DCMIPP CSI Number Of Lanes

DCMIPP_CSI_ONE_DATA_LANE

DCMIPP CSI One Data Lane

DCMIPP_CSI_TWO_DATA_LANES

DCMIPP CSI 2 Data Lanes

DCMIPP CSI PHY BitRate

DCMIPP_CSI_PHY_BT_80

PHY BitRate 80

DCMIPP_CSI_PHY_BT_90

PHY BitRate 90

DCMIPP_CSI_PHY_BT_100

PHY BitRate 100

DCMIPP_CSI_PHY_BT_110

PHY BitRate 110

DCMIPP_CSI_PHY_BT_120

PHY BitRate 120

DCMIPP_CSI_PHY_BT_130

PHY BitRate 130

DCMIPP_CSI_PHY_BT_140

PHY BitRate 140

DCMIPP_CSI_PHY_BT_150

PHY BitRate 150

DCMIPP_CSI_PHY_BT_160

PHY BitRate 160

DCMIPP_CSI_PHY_BT_170

PHY BitRate 170

DCMIPP_CSI_PHY_BT_180

PHY BitRate 180

DCMIPP_CSI_PHY_BT_190

PHY BitRate 190

DCMIPP_CSI_PHY_BT_205

PHY BitRate 205

DCMIPP_CSI_PHY_BT_220

PHY BitRate 220

DCMIPP_CSI_PHY_BT_235

PHY BitRate 235

DCMIPP_CSI_PHY_BT_250

PHY BitRate 250

DCMIPP_CSI_PHY_BT_275

PHY BitRate 275

DCMIPP_CSI_PHY_BT_300

PHY BitRate 300

DCMIPP_CSI_PHY_BT_325

PHY BitRate 325

DCMIPP_CSI_PHY_BT_350

PHY BitRate 350

DCMIPP_CSI_PHY_BT_400

PHY BitRate 400

DCMIPP_CSI_PHY_BT_450

PHY BitRate 450

DCMIPP_CSI_PHY_BT_500

PHY BitRate 500

DCMIPP_CSI_PHY_BT_550

PHY BitRate 550

DCMIPP_CSI_PHY_BT_600

PHY BitRate 600

DCMIPP_CSI_PHY_BT_650

PHY BitRate 650

DCMIPP_CSI_PHY_BT_700

PHY BitRate 700

DCMIPP_CSI_PHY_BT_750

PHY BitRate 750

DCMIPP_CSI_PHY_BT_800

PHY BitRate 800

DCMIPP_CSI_PHY_BT_850

PHY BitRate 850

DCMIPP_CSI_PHY_BT_900

PHY BitRate 900

DCMIPP_CSI_PHY_BT_950

PHY BitRate 950

DCMIPP_CSI_PHY_BT_1000

PHY BitRate 1000

DCMIPP_CSI_PHY_BT_1050

PHY BitRate 1050

DCMIPP_CSI_PHY_BT_1100

PHY BitRate 1100

DCMIPP_CSI_PHY_BT_1150

PHY BitRate 1150

DCMIPP_CSI_PHY_BT_1200

PHY BitRate 1200

DCMIPP_CSI_PHY_BT_1250

PHY BitRate 1250

DCMIPP_CSI_PHY_BT_1300

PHY BitRate 1300

DCMIPP_CSI_PHY_BT_1350

PHY BitRate 1350

DCMIPP_CSI_PHY_BT_1400

PHY BitRate 1400

DCMIPP_CSI_PHY_BT_1450

PHY BitRate 1450

DCMIPP_CSI_PHY_BT_1500

PHY BitRate 1500

DCMIPP_CSI_PHY_BT_1550

PHY BitRate 1550

DCMIPP_CSI_PHY_BT_1600

PHY BitRate 1600

DCMIPP_CSI_PHY_BT_1650

PHY BitRate 1650

DCMIPP_CSI_PHY_BT_1700

PHY BitRate 1700

DCMIPP_CSI_PHY_BT_1750

PHY BitRate 1750

DCMIPP_CSI_PHY_BT_1800

PHY BitRate 1800

DCMIPP_CSI_PHY_BT_1850

PHY BitRate 1850

DCMIPP_CSI_PHY_BT_1900

PHY BitRate 1900

DCMIPP_CSI_PHY_BT_1950

PHY BitRate 1950

DCMIPP_CSI_PHY_BT_2000

PHY BitRate 2000

DCMIPP_CSI_PHY_BT_2050

PHY BitRate 2050

DCMIPP_CSI_PHY_BT_2100

PHY BitRate 2100

DCMIPP_CSI_PHY_BT_2150

PHY BitRate 2150

DCMIPP_CSI_PHY_BT_2200

PHY BitRate 2200

DCMIPP_CSI_PHY_BT_2250

PHY BitRate 2250

DCMIPP_CSI_PHY_BT_2300

PHY BitRate 2300

DCMIPP_CSI_PHY_BT_2350

PHY BitRate 2350

DCMIPP_CSI_PHY_BT_2400

PHY BitRate 2400

DCMIPP_CSI_PHY_BT_2450

PHY BitRate 2450

DCMIPP_CSI_PHY_BT_2500

PHY BitRate 2500

DCMIPP CSI Timer

DCMIPP_CSI_TIMER0

DCMIPP CSI Timer0

DCMIPP_CSI_TIMER1

DCMIPP CSI Timer1

DCMIPP_CSI_TIMER2

DCMIPP CSI Timer2

DCMIPP_CSI_TIMER3

DCMIPP CSI Timer3

DCMIPP CSI Timer StartPoint

DCMIPP_CSI_TIMER_START_SOF

DCMIPP CSI Start timer counter at Start Of Frame

DCMIPP_CSI_TIMER_START_EOF

DCMIPP CSI Start timer counter at End Of Frame

DCMIPP Data Type

DCMIPP_DT_YUV420_8

DCMIPP Data Type YUV420 8bit

DCMIPP_DT_YUV420_10

DCMIPP Data Type YUV420 8bit

DCMIPP_DT_YUV422_8

DCMIPP Data Type YUV422 8bit

DCMIPP_DT_YUV422_10

DCMIPP Data Type YUV422 10bit

DCMIPP_DT_RGB444

DCMIPP Data Type RGB444

DCMIPP_DT_RGB555

DCMIPP Data Type RGB555

DCMIPP_DT_RGB565

DCMIPP Data Type RGB565

DCMIPP_DT_RGB666

DCMIPP Data Type RGB666

DCMIPP_DT_RGB888

DCMIPP Data Type RGB888

DCMIPP_DT_RAW8

DCMIPP Data Type RawBayer8

DCMIPP_DT_RAW10

DCMIPP Data Type RawBayer10

DCMIPP_DT_RAW12

DCMIPP Data Type RawBayer12

DCMIPP_DT_RAW14

DCMIPP Data Type RawBayer14

DCMIPP DCMIPP Data Type Mode

DCMIPP_DTMODE_DTIDA

Only flow DTIDA from the selected VC is forwarded in the pipe

DCMIPP_DTMODE_DTIDA_OR_DTIDB

Flows DTIDA and/or DTIDB from the selected VC are forwarded in the pipe

DCMIPP_DTMODE_ALL_EXCEPT_DTIA_DTIB

All data types from the selected VC (except the DTIDA or DTIDB) are forwarded in the pipe, only for Pipe0

DCMIPP_DTMODE_ALL

All data types of the selected virtual channel VC are forwarded in the pipe, only for Pipe0

DCMIPP Data Type Selection

DCMIPP_DTSELECT_IDA

Data type selection ID A

DCMIPP_DTSELECT_IDB

Data type selection ID B

DCMIPP Error Codes

HAL_DCMIPP_ERROR_NONE

No error

HAL_DCMIPP_ERROR_AXI_TRANSFER

IPPLUG AXI Transfer error

HAL_DCMIPP_ERROR_PARALLEL_SYNC

Synchronization error

HAL_DCMIPP_ERROR_PIPE0_LIMIT

Limit error on pipe0

HAL_DCMIPP_ERROR_PIPE0_OVR

Overrun error on pipe0

HAL_DCMIPP_ERROR_PIPE1_OVR

Overrun error on pipe1

HAL_DCMIPP_ERROR_PIPE2_OVR

Overrun error on pipe2

HAL_DCMIPP_ERROR_INVALID_CALLBACK

Invalid Callback error

HAL_DCMIPP_CSI_ERROR_NONE

No error

HAL_DCMIPP_CSI_ERROR_SYNC

Synchronization error

HAL_DCMIPP_CSI_ERROR_WDG

Watchdog Error

HAL_DCMIPP_CSI_ERROR_SPKT

Shorter Packet than expected Error

HAL_DCMIPP_CSI_ERROR_DATA_ID

Data ID Information Error

HAL_DCMIPP_CSI_ERROR_CECC

Corrected ECC Error

HAL_DCMIPP_CSI_ERROR_ECC

ECC Error

HAL_DCMIPP_CSI_ERROR_CRC

CRC Error

HAL_DCMIPP_CSI_ERROR_DPHY_CTRL

Error Control on data line (0 OR 1)

HAL_DCMIPP_CSI_ERROR_DPHY_LP_SYNC

Low-Power data transmssion Synchronisation Error on data line (0 OR 1)

HAL_DCMIPP_CSI_ERROR_DPHY_ESCAPE

Escape Entry Error on data line (0 OR 1)

HAL_DCMIPP_CSI_ERROR_SOT_SYNC

SOT Synchronisation Error on data line(0 OR 1)

HAL_DCMIPP_CSI_ERROR_SOT

SOT Error on data line (0 OR 1)

DCMIPP Exported Macros

__HAL_DCMIPP_ENABLE_IT

Description:

- Enable the specified DCMIPP interrupts.

Parameters:

- __HANDLE__: DCMIPP handle
- __INTERRUPT__: specifies the DCMIPP interrupt sources to be enabled. This parameter can be any combination of the following values:
  - DCMIPP_IT_AXI_TRANSFER_ERR IPPLUG AXI Transfer error interrupt
  - DCMIPP_IT_PARALLEL_SYNC_ERR Synchronization error interrupt on parallel interface
  - DCMIPP_IT_PIPE0_FRAME Frame capture complete interrupt for the pipe0
  - DCMIPP_IT_PIPE0_VSYNC Vertical sync interrupt for the pipe0
  - DCMIPP_IT_PIPE0_LINE Multi-line capture complete interrupt for the pipe0
  - DCMIPP_IT_PIPE0_LIMIT Limit interrupt for the pipe0
  - DCMIPP_IT_PIPE0_OVR Overrun interrupt for the pipe0
  - DCMIPP_IT_PIPE1_LINE Multi-line capture complete interrupt for the pipe1
  - DCMIPP_IT_PIPE1_FRAME Frame capture complete interrupt for the pipe1
  - DCMIPP_IT_PIPE1_VSYNC Vertical sync interrupt for the pipe1
  - DCMIPP_IT_PIPE1_OVR Overrun interrupt for the pipe1
  - DCMIPP_IT_PIPE2_LINE Multi-line capture complete interrupt for the pipe2
  - DCMIPP_IT_PIPE2_FRAME Frame capture complete interrupt for the pipe2
  - DCMIPP_IT_PIPE2_VSYNC Vertical sync interrupt for the pipe2
  - DCMIPP_IT_PIPE2_OVR Overrun interrupt for the pipe2

Return value:

- None

__HAL_DCMIPP_DISABLE_IT

Description:

- Disable the specified DCMIPP interrupts.

Parameters:

- __HANDLE__: DCMIPP handle
- __INTERRUPT__: specifies the DCMIPP interrupt sources to be disabled. This parameter can be any combination of the following values:
  - DCMIPP_IT_AXI_TRANSFER_ERR IPPLUG AXI Transfer error interrupt
  - DCMIPP_IT_PARALLEL_SYNC_ERR Synchronization error interrupt on parallel interface
  - DCMIPP_IT_PIPE0_FRAME Frame capture complete interrupt for the pipe0
  - DCMIPP_IT_PIPE0_VSYNC Vertical sync interrupt for the pipe0
  - DCMIPP_IT_PIPE0_LINE Multi-line capture complete interrupt for the pipe0
  - DCMIPP_IT_PIPE0_LIMIT Limit interrupt for the pipe0
  - DCMIPP_IT_PIPE0_OVR Overrun interrupt for the pipe0
  - DCMIPP_IT_PIPE1_LINE Multi-line capture complete interrupt for the pipe1
  - DCMIPP_IT_PIPE1_FRAME Frame capture complete interrupt for the pipe1
  - DCMIPP_IT_PIPE1_VSYNC Vertical sync interrupt for the pipe1
  - DCMIPP_IT_PIPE1_OVR Overrun interrupt for the pipe1
  - DCMIPP_IT_PIPE2_LINE Multi-line capture complete interrupt for the pipe2
  - DCMIPP_IT_PIPE2_FRAME Frame capture complete interrupt for the pipe2
  - DCMIPP_IT_PIPE2_VSYNC Vertical sync interrupt for the pipe2
  - DCMIPP_IT_PIPE2_OVR Overrun interrupt for the pipe2

Return value:

- None

__HAL_DCMIPP_GET_FLAG

Description:

- Get the DCMIPP pending interrupt flags.

Parameters:

- __HANDLE__: DCMIPP handle
- __FLAG__: Get the specified flag. This parameter can be any combination of the following values:
  - DCMIPP_FLAG_AXI_TRANSFER_ERR IPPLUG AXI Transfer error interrupt flag
  - DCMIPP_FLAG_PARALLEL_SYNC_ERR Synchronization error interrupt flag on parallel interface
  - DCMIPP_FLAG_PIPE0_FRAME Frame capture complete interrupt flag for the pipe0
  - DCMIPP_FLAG_PIPE0_VSYNC Vertical sync interrupt flag for the pipe0
  - DCMIPP_FLAG_PIPE0_LINE Multi-line capture complete interrupt flag for the pipe0
  - DCMIPP_FLAG_PIPE0_LIMIT Limit interrupt flag for the pipe0
  - DCMIPP_FLAG_PIPE0_OVR Overrun interrupt flag for the pipe0
  - DCMIPP_FLAG_PIPE1_LINE Multi-line capture complete interrupt flag for the pipe1
  - DCMIPP_FLAG_PIPE1_FRAME Frame capture complete interrupt flag for the pipe1
  - DCMIPP_FLAG_PIPE1_VSYNC Vertical sync interrupt flag for the pipe1
  - DCMIPP_FLAG_PIPE1_OVR Overrun interrupt flag for the pipe1
  - DCMIPP_FLAG_PIPE2_LINE Multi-line capture complete interrupt flag for the pipe2
  - DCMIPP_FLAG_PIPE2_FRAME Frame capture complete interrupt flag for the pipe2
  - DCMIPP_FLAG_PIPE2_VSYNC Vertical sync interrupt flag for the pipe2
  - DCMIPP_FLAG_PIPE2_OVR Overrun interrupt flag for the pipe2

Return value:

- The: state of FLAG (SET or RESET).

__HAL_DCMIPP_CLEAR_FLAG

Description:

- Clear the DCMIPP pending interrupt flags.

Parameters:

- __HANDLE__: DCMIPP handle
- __FLAG__: specifies the flag to clear. This parameter can be any combination of the following values:
  - DCMIPP_FLAG_AXI_TRANSFER_ERR IPPLUG AXI Transfer error interrupt
  - DCMIPP_FLAG_PARALLEL_SYNC_ERR Synchronization error interrupt on parallel interface
  - DCMIPP_FLAG_PIPE0_FRAME Frame capture complete interrupt for the pipe0
  - DCMIPP_FLAG_PIPE0_VSYNC Vertical sync interrupt for the pipe0
  - DCMIPP_FLAG_PIPE0_LINE Multi-line capture complete interrupt for the pipe0
  - DCMIPP_FLAG_PIPE0_LIMIT Limit interrupt for the pipe0
  - DCMIPP_FLAG_PIPE0_OVR Overrun interrupt for the pipe0
  - DCMIPP_FLAG_PIPE1_LINE Multi-line capture complete interrupt for the pipe1
  - DCMIPP_FLAG_PIPE1_FRAME Frame capture complete interrupt for the pipe1
  - DCMIPP_FLAG_PIPE1_VSYNC Vertical sync interrupt for the pipe1
  - DCMIPP_FLAG_PIPE1_OVR Overrun interrupt for the pipe1
  - DCMIPP_FLAG_PIPE2_LINE Multi-line capture complete interrupt for the pipe2
  - DCMIPP_FLAG_PIPE2_FRAME Frame capture complete interrupt for the pipe2
  - DCMIPP_FLAG_PIPE2_VSYNC Vertical sync interrupt for the pipe2
  - DCMIPP_FLAG_PIPE2_OVR Overrun interrupt for the pipe2

Return value:

- None

__HAL_DCMIPP_GET_IT_SOURCE

Description:

- Checks whether the specified DCMIPP interrupt is enabled or not.

Parameters:

- __HANDLE__: DCMIPP handle
- __INTERRUPT__: specifies the DCMIPP interrupt sources to be checked. This parameter can be any combination of the following values:
  - DCMIPP_IT_AXI_TRANSFER_ERR IPPLUG AXI Transfer error interrupt
  - DCMIPP_IT_PARALLEL_SYNC_ERR Synchronization error interrupt on parallel interface
  - DCMIPP_IT_PIPE0_FRAME Frame capture complete interrupt for the pipe0
  - DCMIPP_IT_PIPE0_VSYNC Vertical sync interrupt for the pipe0
  - DCMIPP_IT_PIPE0_LINE Multi-line capture complete interrupt for the pipe0
  - DCMIPP_IT_PIPE0_LIMIT Limit interrupt for the pipe0
  - DCMIPP_IT_PIPE0_OVR Overrun interrupt for the pipe0
  - DCMIPP_IT_PIPE1_LINE Multi-line capture complete interrupt for the pipe1
  - DCMIPP_IT_PIPE1_FRAME Frame capture complete interrupt for the pipe1
  - DCMIPP_IT_PIPE1_VSYNC Vertical sync interrupt for the pipe1
  - DCMIPP_IT_PIPE1_OVR Overrun interrupt for the pipe1
  - DCMIPP_IT_PIPE2_LINE Multi-line capture complete interrupt for the pipe2
  - DCMIPP_IT_PIPE2_FRAME Frame capture complete interrupt for the pipe2
  - DCMIPP_IT_PIPE2_VSYNC Vertical sync interrupt for the pipe2
  - DCMIPP_IT_PIPE2_OVR Overrun interrupt for the pipe2

Return value:

- The: state of DCMIPP interrupt (SET or RESET).

__HAL_DCMIPP_CSI_ENABLE_IT

Description:

- Enable the specified DCMIPP CSI interrupts.

Parameters:

- __HANDLE__: CSI handle
- __INTERRUPT__: specifies the DCMIPP CSI interrupt sources to be enabled. This parameter can be any combination of the following values:
  - DCMIPP_CSI_IT_SYNCERR Synchronization error interrupt
  - DCMIPP_CSI_IT_WDERR Watchdog error interrupt
  - DCMIPP_CSI_IT_SPKTERR Shorter packet than expected error interrupt
  - DCMIPP_CSI_IT_IDERR Data ID information error interrupt
  - DCMIPP_CSI_IT_CECCERR Corrected ECC error interrupt
  - DCMIPP_CSI_IT_ECCERR ECC Error interrupt
  - DCMIPP_CSI_IT_CRCERR CRC Error interrupt
  - DCMIPP_CSI_IT_CCFIFO Clock changer FIFO full event interrupt
  - DCMIPP_CSI_IT_SPKT Short packet detection interrupt
  - DCMIPP_CSI_IT_EOF3 EOF on virtual channel 3 interrupt
  - DCMIPP_CSI_IT_EOF2 EOF on virtual channel 2 interrupt
  - DCMIPP_CSI_IT_EOF1 EOF on virtual channel 1 interrupt
  - DCMIPP_CSI_IT_EOF0 EOF on virtual channel 0 interrupt
  - DCMIPP_CSI_IT_SOF3 SOF on virtual channel 3 interrupt
  - DCMIPP_CSI_IT_SOF2 SOF on virtual channel 2 interrupt
  - DCMIPP_CSI_IT_SOF1 SOF on virtual channel 1 interrupt
  - DCMIPP_CSI_IT_SOF0 SOF on virtual channel 0 interrupt
  - DCMIPP_CSI_IT_TIM3 Timer 3 counter value reached interrupt
  - DCMIPP_CSI_IT_TIM2 Timer 2 counter value reached interrupt
  - DCMIPP_CSI_IT_TIM1 Timer 1 counter value reached interrupt
  - DCMIPP_CSI_IT_TIM0 Timer 0 counter value reached interrupt
  - DCMIPP_CSI_IT_LB3 Line/byte event 3 reached interrupt
  - DCMIPP_CSI_IT_LB2 Line/byte event 2 reached interrupt
  - DCMIPP_CSI_IT_LB1 Line/byte event 1 reached interrupt
  - DCMIPP_CSI_IT_LB0 Line/byte event 0 reached interrupt

Return value:

- None

__HAL_DCMIPP_CSI_DISABLE_IT

Description:

- Disable the specified DCMIPP CSI interrupts.

Parameters:

- __HANDLE__: CSI handle
- __INTERRUPT__: specifies the DCMIPP CSI interrupt sources to be enabled. This parameter can be any combination of the following values:
  - DCMIPP_CSI_IT_SYNCERR Synchronization error interrupt
  - DCMIPP_CSI_IT_WDERR Watchdog error interrupt
  - DCMIPP_CSI_IT_SPKTERR Shorter packet than expected error interrupt
  - DCMIPP_CSI_IT_IDERR Data ID information error interrupt
  - DCMIPP_CSI_IT_CECCERR Corrected ECC error interrupt
  - DCMIPP_CSI_IT_ECCERR ECC Error interrupt
  - DCMIPP_CSI_IT_CRCERR CRC Error interrupt
  - DCMIPP_CSI_IT_CCFIFO Clock changer FIFO full event interrupt
  - DCMIPP_CSI_IT_SPKT Short packet detection interrupt
  - DCMIPP_CSI_IT_EOF3 EOF on virtual channel 3 interrupt
  - DCMIPP_CSI_IT_EOF2 EOF on virtual channel 2 interrupt
  - DCMIPP_CSI_IT_EOF1 EOF on virtual channel 1 interrupt
  - DCMIPP_CSI_IT_EOF0 EOF on virtual channel 0 interrupt
  - DCMIPP_CSI_IT_SOF3 SOF on virtual channel 3 interrupt
  - DCMIPP_CSI_IT_SOF2 SOF on virtual channel 2 interrupt
  - DCMIPP_CSI_IT_SOF1 SOF on virtual channel 1 interrupt
  - DCMIPP_CSI_IT_SOF0 SOF on virtual channel 0 interrupt
  - DCMIPP_CSI_IT_TIM3 Timer 3 counter value reached interrupt
  - DCMIPP_CSI_IT_TIM2 Timer 2 counter value reached interrupt
  - DCMIPP_CSI_IT_TIM1 Timer 1 counter value reached interrupt
  - DCMIPP_CSI_IT_TIM0 Timer 0 counter value reached interrupt
  - DCMIPP_CSI_IT_LB3 Line/byte event 3 reached interrupt
  - DCMIPP_CSI_IT_LB2 Line/byte event 2 reached interrupt
  - DCMIPP_CSI_IT_LB1 Line/byte event 1 reached interrupt
  - DCMIPP_CSI_IT_LB0 Line/byte event 0 reached interrupt

Return value:

- None

__HAL_DCMIPP_CSI_GET_FLAG

Description:

- Get the DCMIPP CSI pending interrupt flags.

Parameters:

- __HANDLE__: CSI handle
- __FLAG__: Get the specified flag. This parameter can be any combination of the following values:
  - DCMIPP_CSI_FLAG_SYNCERR Synchronization error flag
  - DCMIPP_CSI_FLAG_WDERR Watchdog error flag
  - DCMIPP_CSI_FLAG_SPKTERR Shorter packet than expected error flag
  - DCMIPP_CSI_FLAG_IDERR Data ID information error flag
  - DCMIPP_CSI_FLAG_CECCERR Corrected ECC error flag
  - DCMIPP_CSI_FLAG_ECCERR ECC Error flag
  - DCMIPP_CSI_FLAG_CRCERR CRC Error flag
  - DCMIPP_CSI_FLAG_CCFIFO Clock changer FIFO full event flag
  - DCMIPP_CSI_FLAG_SPKT Short packet detection flag
  - DCMIPP_CSI_FLAG_EOF3 EOF on virtual channel 3 flag
  - DCMIPP_CSI_FLAG_EOF2 EOF on virtual channel 2 flag
  - DCMIPP_CSI_FLAG_EOF1 EOF on virtual channel 1 flag
  - DCMIPP_CSI_FLAG_EOF0 EOF on virtual channel 0 flag
  - DCMIPP_CSI_FLAG_SOF3 SOF on virtual channel 3 flag
  - DCMIPP_CSI_FLAG_SOF2 SOF on virtual channel 2 flag
  - DCMIPP_CSI_FLAG_SOF1 SOF on virtual channel 1 flag
  - DCMIPP_CSI_FLAG_SOF0 SOF on virtual channel 0 flag
  - DCMIPP_CSI_FLAG_TIM3 Timer 3 counter value reached flag
  - DCMIPP_CSI_FLAG_TIM2 Timer 2 counter value reached flag
  - DCMIPP_CSI_FLAG_TIM1 Timer 1 counter value reached flag
  - DCMIPP_CSI_FLAG_TIM0 Timer 0 counter value reached flag
  - DCMIPP_CSI_FLAG_LB3 Line/byte event 3 reached flag
  - DCMIPP_CSI_FLAG_LB2 Line/byte event 2 reached flag
  - DCMIPP_CSI_FLAG_LB1 Line/byte event 1 reached flag
  - DCMIPP_CSI_FLAG_LB0 Line/byte event 0 reached flag

Return value:

- None

__HAL_DCMIPP_CSI_CLEAR_FLAG

Description:

- Clear the DCMIPP CSI pending interrupt flags.

Parameters:

- __HANDLE__: CSI handle
- __FLAG__: specifies the flag to clear. This parameter can be any combination of the following values:
  - DCMIPP_CSI_FLAG_SYNCERR Synchronization error flag
  - DCMIPP_CSI_FLAG_WDERR Watchdog error flag
  - DCMIPP_CSI_FLAG_SPKTERR Shorter packet than expected error flag
  - DCMIPP_CSI_FLAG_IDERR Data ID information error flag
  - DCMIPP_CSI_FLAG_CECCERR Corrected ECC error flag
  - DCMIPP_CSI_FLAG_ECCERR ECC Error flag
  - DCMIPP_CSI_FLAG_CRCERR CRC Error flag
  - DCMIPP_CSI_FLAG_CCFIFO Clock changer FIFO full event flag
  - DCMIPP_CSI_FLAG_SPKT Short packet detection flag
  - DCMIPP_CSI_FLAG_EOF3 EOF on virtual channel 3 flag
  - DCMIPP_CSI_FLAG_EOF2 EOF on virtual channel 2 flag
  - DCMIPP_CSI_FLAG_EOF1 EOF on virtual channel 1 flag
  - DCMIPP_CSI_FLAG_EOF0 EOF on virtual channel 0 flag
  - DCMIPP_CSI_FLAG_SOF3 SOF on virtual channel 3 flag
  - DCMIPP_CSI_FLAG_SOF2 SOF on virtual channel 2 flag
  - DCMIPP_CSI_FLAG_SOF1 SOF on virtual channel 1 flag
  - DCMIPP_CSI_FLAG_SOF0 SOF on virtual channel 0 flag
  - DCMIPP_CSI_FLAG_TIM3 Timer 3 counter value reached flag
  - DCMIPP_CSI_FLAG_TIM2 Timer 2 counter value reached flag
  - DCMIPP_CSI_FLAG_TIM1 Timer 1 counter value reached flag
  - DCMIPP_CSI_FLAG_TIM0 Timer 0 counter value reached flag
  - DCMIPP_CSI_FLAG_LB3 Line/byte event 3 reached flag
  - DCMIPP_CSI_FLAG_LB2 Line/byte event 2 reached flag
  - DCMIPP_CSI_FLAG_LB1 Line/byte event 1 reached flag
  - DCMIPP_CSI_FLAG_LB0 Line/byte event 0 reached flag

Return value:

- None

__HAL_DCMIPP_CSI_GET_IT_SOURCE

Description:

- Checks whether the specified DCMIPP interrupt is enabled or not.

Parameters:

- __HANDLE__: DCMIPP handle
- __INTERRUPT__: specifies the DCMIPP interrupt sources to be checked. This parameter can be any combination of the following values:
  - DCMIPP_CSI_IT_SYNCERR Synchronization error interrupt
  - DCMIPP_CSI_IT_WDERR Watchdog error interrupt
  - DCMIPP_CSI_IT_SPKTERR Shorter packet than expected error interrupt
  - DCMIPP_CSI_IT_IDERR Data ID information error interrupt
  - DCMIPP_CSI_IT_CECCERR Corrected ECC error interrupt
  - DCMIPP_CSI_IT_ECCERR ECC Error interrupt
  - DCMIPP_CSI_IT_CRCERR CRC Error interrupt
  - DCMIPP_CSI_IT_CCFIFO Clock changer FIFO full event interrupt
  - DCMIPP_CSI_IT_SPKT Short packet detection interrupt
  - DCMIPP_CSI_IT_EOF3 EOF on virtual channel 3 interrupt
  - DCMIPP_CSI_IT_EOF2 EOF on virtual channel 2 interrupt
  - DCMIPP_CSI_IT_EOF1 EOF on virtual channel 1 interrupt
  - DCMIPP_CSI_IT_EOF0 EOF on virtual channel 0 interrupt
  - DCMIPP_CSI_IT_SOF3 SOF on virtual channel 3 interrupt
  - DCMIPP_CSI_IT_SOF2 SOF on virtual channel 2 interrupt
  - DCMIPP_CSI_IT_SOF1 SOF on virtual channel 1 interrupt
  - DCMIPP_CSI_IT_SOF0 SOF on virtual channel 0 interrupt
  - DCMIPP_CSI_IT_TIM3 Timer 3 counter value reached interrupt
  - DCMIPP_CSI_IT_TIM2 Timer 2 counter value reached interrupt
  - DCMIPP_CSI_IT_TIM1 Timer 1 counter value reached interrupt
  - DCMIPP_CSI_IT_TIM0 Timer 0 counter value reached interrupt
  - DCMIPP_CSI_IT_LB3 Line/byte event 3 reached interrupt
  - DCMIPP_CSI_IT_LB2 Line/byte event 2 reached interrupt
  - DCMIPP_CSI_IT_LB1 Line/byte event 1 reached interrupt
  - DCMIPP_CSI_IT_LB0 Line/byte event 0 reached interrupt

Return value:

- The: state of DCMIPP interrupt (SET or RESET).

__HAL_DCMIPP_CSI_DPHY_ENABLE_IT

Description:

- Enable the specified DCMIPP CSI DPHY interrupts.

Parameters:

- __HANDLE__: CSI handle
- __INTERRUPT__: specifies the DCMIPP CSI interrupt sources to be enabled. This parameter can be any combination of the following values:
  - DCMIPP_CSI_IT_ECTRLDL1 Error control on data line 1 interrupt
  - DCMIPP_CSI_IT_ESYNCESCDL1 Low-power data transmission synchronization error on data line 1 interrupt
  - DCMIPP_CSI_IT_EESCDL1 Escape entry error on data line 1 interrupt
  - DCMIPP_CSI_IT_ESOTSYNCDL1 SOT synchronization error on data line 1 interrupt
  - DCMIPP_CSI_IT_ESOTDL1 SOT error on data line 1 interrupt
  - DCMIPP_CSI_IT_ECTRLDL0 Error control on data line 0 interrupt
  - DCMIPP_CSI_IT_ESYNCESCDL0 Low-power data transmission synchronization error on data line 0 interrupt
  - DCMIPP_CSI_IT_EESCDL0 Escape entry error on data line 0 interrupt
  - DCMIPP_CSI_IT_ESOTSYNCDL0 SOT synchronization error on data line 0 interrupt
  - DCMIPP_CSI_IT_ESOTDL0 SOT error on data line 0 interrupt

Return value:

- None

__HAL_DCMIPP_CSI_DPHY_DISABLE_IT

Description:

- Disable the specified DCMIPP CSI DPHY interrupts.

Parameters:

- __HANDLE__: CSI handle
- __INTERRUPT__: specifies the DCMIPP CSI interrupt sources to be enabled. This parameter can be any combination of the following values:
  - DCMIPP_CSI_IT_ECTRLDL1 Error control on data line 1 interrupt
  - DCMIPP_CSI_IT_ESYNCESCDL1 Low-power data transmission synchronization error on data line 1 interrupt
  - DCMIPP_CSI_IT_EESCDL1 Escape entry error on data line 1 interrupt
  - DCMIPP_CSI_IT_ESOTSYNCDL1 SOT synchronization error on data line 1 interrupt
  - DCMIPP_CSI_IT_ESOTDL1 SOT error on data line 1 interrupt
  - DCMIPP_CSI_IT_ECTRLDL0 Error control on data line 0 interrupt
  - DCMIPP_CSI_IT_ESYNCESCDL0 Low-power data transmission synchronization error on data line 0 interrupt
  - DCMIPP_CSI_IT_EESCDL0 Escape entry error on data line 0 interrupt
  - DCMIPP_CSI_IT_ESOTSYNCDL0 SOT synchronization error on data line 0 interrupt
  - DCMIPP_CSI_IT_ESOTDL0 SOT error on data line 0 interrupt

Return value:

- None

__HAL_DCMIPP_CSI_GET_DPHY_FLAG

Description:

- Get the DCMIPP CSI DPHY pending interrupt flags.

Parameters:

- __HANDLE__: CSI handle
- __FLAG__: Get the specified flag. This parameter can be any combination of the following values:
  - DCMIPP_CSI_FLAG_ECTRLDL1 Error control on data line 1 flag
  - DCMIPP_CSI_FLAG_ESYNCESCDL1 Low-power data transmission synchronization error on data line 1 flag
  - DCMIPP_CSI_FLAG_EESCDL1 Escape entry error on data line 1 flag
  - DCMIPP_CSI_FLAG_ESOTSYNCDL1 SOT synchronization error on data line 1 flag
  - DCMIPP_CSI_FLAG_ESOTDL1 SOT error on data line 1 flag
  - DCMIPP_CSI_FLAG_ECTRLDL0 Error control on data line 0 flag
  - DCMIPP_CSI_FLAG_ESYNCESCDL0 Low-power data transmission synchronization error on data line 0 flag
  - DCMIPP_CSI_FLAG_EESCDL0 Escape entry error on data line 0 flag
  - DCMIPP_CSI_FLAG_ESOTSYNCDL0 SOT synchronization error on data line 0 flag
  - DCMIPP_CSI_FLAG_ESOTDL0 SOT error on data line 0 flag

Return value:

- None

__HAL_DCMIPP_CSI_CLEAR_DPHY_FLAG

Description:

- Clear the DCMIPP CSI DPHY pending interrupt flags.

Parameters:

- __HANDLE__: CSI handle
- __FLAG__: specifies the flag to clear. This parameter can be any combination of the following values:
  - DCMIPP_CSI_FLAG_ECTRLDL1 Error control on data line 1 flag
  - DCMIPP_CSI_FLAG_ESYNCESCDL1 Low-power data transmission synchronization error on data line 1 flag
  - DCMIPP_CSI_FLAG_EESCDL1 Escape entry error on data line 1 flag
  - DCMIPP_CSI_FLAG_ESOTSYNCDL1 SOT synchronization error on data line 1 flag
  - DCMIPP_CSI_FLAG_ESOTDL1 SOT error on data line 1 flag
  - DCMIPP_CSI_FLAG_ECTRLDL0 Error control on data line 0 flag
  - DCMIPP_CSI_FLAG_ESYNCESCDL0 Low-power data transmission synchronization error on data line 0 flag
  - DCMIPP_CSI_FLAG_EESCDL0 Escape entry error on data line 0 flag
  - DCMIPP_CSI_FLAG_ESOTSYNCDL0 SOT synchronization error on data line 0 flag
  - DCMIPP_CSI_FLAG_ESOTDL0 SOT error on data line 0 flag

Return value:

- None

__HAL_DCMIPP_CSI_GET_DPHY_IT_SOURCE

Description:

- Checks whether the specified DCMIPP CSI DPHY interrupt is enabled or not.

Parameters:

- __HANDLE__: CSI handle
- __INTERRUPT__: specifies the DCMIPP CSI D-PHY interrupt sources to be checked. This parameter can be any combination of the following values:
  - DCMIPP_CSI_IT_ECTRLDL1 Error control on data line 1 interrupt
  - DCMIPP_CSI_IT_ESYNCESCDL1 Low-power data transmission synchronization error on data line 1 interrupt
  - DCMIPP_CSI_IT_EESCDL1 Escape entry error on data line 1 interrupt
  - DCMIPP_CSI_IT_ESOTSYNCDL1 SOT synchronization error on data line 1 interrupt
  - DCMIPP_CSI_IT_ESOTDL1 SOT error on data line 1 interrupt
  - DCMIPP_CSI_IT_ECTRLDL0 Error control on data line 0 interrupt
  - DCMIPP_CSI_IT_ESYNCESCDL0 Low-power data transmission synchronization error on data line 0 interrupt
  - DCMIPP_CSI_IT_EESCDL0 Escape entry error on data line 0 interrupt
  - DCMIPP_CSI_IT_ESOTSYNCDL0 SOT synchronization error on data line 0 interrupt
  - DCMIPP_CSI_IT_ESOTDL0 SOT error on data line 0 interrupt

Return value:

- None

DCMIPP Exported Types

MAX_DATATYPE_NB

DCMIPP CSI maximum number of data types

DCMIPP Extended Data Mode

DCMIPP_INTERFACE_8BITS

Interface captures 8bits on every pixel clock

DCMIPP_INTERFACE_10BITS

Interface captures 10bits on every pixel clock

DCMIPP_INTERFACE_12BITS

Interface captures 12bits on every pixel clock

DCMIPP_INTERFACE_14BITS

Interface captures 14bits on every pixel clock

DCMIPP_INTERFACE_16BITS

Interface captures 16bits on every pixel clock

DCMIPP Format

DCMIPP_FORMAT_BYTE

DCMIPP Format BYTE

DCMIPP_FORMAT_YUV422

DCMIPP Format YUV422

DCMIPP_FORMAT_RGB565

DCMIPP Format RGB565

DCMIPP_FORMAT_RGB666

DCMIPP Format RGB666

DCMIPP_FORMAT_RGB888

DCMIPP Format RGB888

DCMIPP_FORMAT_RAW8

DCMIPP Format RAW8

DCMIPP_FORMAT_RAW10

DCMIPP Format RAW10

DCMIPP_FORMAT_RAW12

DCMIPP Format RAW12

DCMIPP_FORMAT_RAW14

DCMIPP Format RAW14

DCMIPP_FORMAT_MONOCHROME_8B

DCMIPP Format 8-bits

DCMIPP_FORMAT_MONOCHROME_10B

DCMIPP Format 10-bits

DCMIPP_FORMAT_MONOCHROME_12B

DCMIPP Format 12-bits

DCMIPP_FORMAT_MONOCHROME_14B

DCMIPP Format 14-bits

DCMIPP Frame Rates

DCMIPP_FRAME_RATE_ALL

All frames captured

DCMIPP_FRAME_RATE_1_OVER_2

1 frame over 2 captured

DCMIPP_FRAME_RATE_1_OVER_4

1 frame over 4 captured

DCMIPP_FRAME_RATE_1_OVER_8

1 frame over 8 captured

DCMIPP HSYNC Polarity

DCMIPP_HSPOLARITY_LOW

Horizontal synchronization active Low

DCMIPP_HSPOLARITY_HIGH

Horizontal synchronization active High

DCMIPP Interrupt Flags

DCMIPP_FLAG_AXI_TRANSFER_ERROR

IPPLUG AXI Transfer error interrupt flag

DCMIPP_FLAG_PARALLEL_SYNC_ERROR

Synchronization error interrupt on parallel interface flag

DCMIPP_FLAG_PIPE0_FRAME

Frame capture interrupt complete for pipe0 flag

DCMIPP_FLAG_PIPE0_VSYNC

Vertical synch interrupt for pipe0 flag

DCMIPP_FLAG_PIPE0_LINE

Multiline interrupt for pipe0 flag

DCMIPP_FLAG_PIPE0_LIMIT

Limit interrupt for pipe0 flag

DCMIPP_FLAG_PIPE0_OVR

Overrun interrupt for pipe0 flag

DCMIPP_FLAG_PIPE1_LINE

Multiline capture interrupt for pipe1 flag

DCMIPP_FLAG_PIPE1_FRAME

Frame capture interrupt complete for pipe1 flag

DCMIPP_FLAG_PIPE1_VSYNC

Vertical synch interrupt for pipe1 flag

DCMIPP_FLAG_PIPE1_OVR

Overrun interrupt for pipe1 flag

DCMIPP_FLAG_PIPE2_LINE

Multiline capture interrupt for pipe2 flag

DCMIPP_FLAG_PIPE2_FRAME

Frame capture interrupt complete for pipe2 flag

DCMIPP_FLAG_PIPE2_VSYNC

Vertical synch interrupt for pipe2 flag

DCMIPP_FLAG_PIPE2_OVR

Overrun interrupt for pipe2 flag

DCMIPP Interrupt sources

DCMIPP_IT_AXI_TRANSFER_ERROR

IPPLUG AXI Transfer error interrupt

DCMIPP_IT_PARALLEL_SYNC_ERROR

Sync error interrupt on parallel interface

DCMIPP_IT_PIPE0_FRAME

Frame capture interrupt complete for pipe0

DCMIPP_IT_PIPE0_VSYNC

Vertical sync interrupt for pipe0

DCMIPP_IT_PIPE0_LINE

Multiline interrupt for pipe0

DCMIPP_IT_PIPE0_LIMIT

Limit interrupt for pipe0

DCMIPP_IT_PIPE0_OVR

Overrun interrupt for pipe0

DCMIPP_IT_PIPE1_LINE

Multiline capture interrupt for pipe1

DCMIPP_IT_PIPE1_FRAME

Frame capture interrupt complete for pipe1

DCMIPP_IT_PIPE1_VSYNC

Vertical sync interrupt for pipe1

DCMIPP_IT_PIPE1_OVR

Overrun interrupt for pipe1

DCMIPP_IT_PIPE2_LINE

Multiline capture interrupt for pipe2

DCMIPP_IT_PIPE2_FRAME

Frame capture interrupt complete for pipe2

DCMIPP_IT_PIPE2_VSYNC

Vertical synch interrupt for pipe2

DCMIPP_IT_PIPE2_OVR

Overrun interrupt for pipe2

DCMIPP IPPLUG Client

DCMIPP_CLIENT1

Client 1 identifier

DCMIPP_CLIENT2

Client 2 identifier

DCMIPP_CLIENT3

Client 3 identifier

DCMIPP_CLIENT4

Client 4 identifier

DCMIPP_CLIENT5

Client 5 identifier

DCMIPP Line Mult

DCMIPP_MULTILINE_1_LINE

Event after every 1 line

DCMIPP_MULTILINE_2_LINES

Event after every 2 lines

DCMIPP_MULTILINE_4_LINES

Event after every 4 lines

DCMIPP_MULTILINE_8_LINES

Event after every 8 lines

DCMIPP_MULTILINE_16_LINES

Event after every 16 lines

DCMIPP_MULTILINE_32_LINES

Event after every 32 lines

DCMIPP_MULTILINE_64_LINES

Event after every 64 lines

DCMIPP_MULTILINE_128_LINES

Event after every 128 lines

DCMIPP line Wrap Address

DCMIPP_WRAP_ADDRESS_1_LINE

Wraps address after 1 line

DCMIPP_WRAP_ADDRESS_2_LINES

Wraps address after 2 lines

DCMIPP_WRAP_ADDRESS_4_LINES

Wraps address after 4 lines

DCMIPP_WRAP_ADDRESS_8_LINES

Wraps address after 8 lines

DCMIPP_WRAP_ADDRESS_16_LINES

Wraps address after 16 lines

DCMIPP_WRAP_ADDRESS_32_LINES

Wraps address after 32 lines

DCMIPP_WRAP_ADDRESS_64_LINES

Wraps address after 64 lines

DCMIPP_WRAP_ADDRESS_128_LINES

Wraps address after 128 lines

DCMIPP Line Select Mode

DCMIPP_LSM_ALL

Interface captures all received lines

DCMIPP_LSM_ALTERNATE_2

Interface captures one line out of two

DCMIPP Line Start Mode

DCMIPP_OELS_ODD

Interface captures first line from the frame start, second one is dropped

DCMIPP_OELS_EVEN

Interface captures second line from the frame start, first one is dropped

DCMIPP Maximum Outstanding Transactions

DCMIPP_OUTSTANDING_TRANSACTION_NONE

Nooutstanding transaction limitation

DCMIPP_OUTSTANDING_TRANSACTION_2

Two outstanding transactions

DCMIPP_OUTSTANDING_TRANSACTION_3

Three outstanding transactions

DCMIPP_OUTSTANDING_TRANSACTION_4

Four outstanding transactions

DCMIPP_OUTSTANDING_TRANSACTION_5

5 outstanding transactions

DCMIPP_OUTSTANDING_TRANSACTION_6

6 outstanding transactions

DCMIPP_OUTSTANDING_TRANSACTION_7

7 outstanding transactions

DCMIPP_OUTSTANDING_TRANSACTION_8

8 outstanding transactions

DCMIPP_OUTSTANDING_TRANSACTION_9

9 outstanding transactions

DCMIPP_OUTSTANDING_TRANSACTION_10

10 outstanding transactions

DCMIPP_OUTSTANDING_TRANSACTION_11

11 outstanding transactions

DCMIPP_OUTSTANDING_TRANSACTION_12

12 outstanding transactions

DCMIPP_OUTSTANDING_TRANSACTION_13

13 outstanding transactions

DCMIPP_OUTSTANDING_TRANSACTION_14

14 outstanding transactions

DCMIPP_OUTSTANDING_TRANSACTION_15

15 outstanding transactions

DCMIPP_OUTSTANDING_TRANSACTION_16

16 outstanding transactions

DCMIPP Memory

DCMIPP_MEMORY_ADDRESS_0

Base destination address

DCMIPP_MEMORY_ADDRESS_1

Second destination address

DCMIPP Memory Page Size

DCMIPP_MEMORY_PAGE_SIZE_64BYTES

Memory Page size 64 Bytes

DCMIPP_MEMORY_PAGE_SIZE_128BYTES

Memory Page size 128 Bytes

DCMIPP_MEMORY_PAGE_SIZE_256BYTES

Memory Page size 256 Bytes

DCMIPP_MEMORY_PAGE_SIZE_512BYTES

Memory Page size 512 Bytes

DCMIPP_MEMORY_PAGE_SIZE_1KBYTES

Memory Page size 1 Bytes

DCMIPP_MEMORY_PAGE_SIZE_2KBYTES

Memory Page size 2 Bytes

DCMIPP_MEMORY_PAGE_SIZE_4KBYTES

Memory Page size 4 Bytes

DCMIPP_MEMORY_PAGE_SIZE_8KBYTES

Memory Page size 8 Bytes

(DCMI or CSI): modes are exclusive

DCMIPP_PARALLEL_MODE

DCMIPP Parallel (DCMI) mode

DCMIPP_SERIAL_MODE

DCMIPP CSI mode

DCMIPP Multiply

DCMIPP_MULTIPY_BY_1

DCMIPP multiply R/G/B by 1

DCMIPP_MULTIPY_BY_2

DCMIPP multiply R/G/B by 2

DCMIPP_MULTIPY_BY_4

DCMIPP multiply R/G/B by 4

DCMIPP_MULTIPY_BY_8

DCMIPP multiply R/G/B by 8

DCMIPP OutputSamplesType

DCMIPP_CLAMP_YUV

Output samples type : Clamped to [16;235] for Y and to [16;240] for U and V

DCMIPP_CLAMP_RGB

Output samples type:Clamped to [16;235] for R, G and B

DCMIPP Pipes

DCMIPP_PIPE0

DCMIPP Pipe0 (Dump pipe)

DCMIPP_PIPE1

DCMIPP Pipe1 (Main Pipe)

DCMIPP_PIPE2

DCMIPP Pipe2 (Ancillary pipe)

DCMIPP Pipe Horizontal Decimation Ratio

DCMIPP_HDEC_ALL

All pixels captured , no horizontal decimation

DCMIPP_HDEC_1_OUT_2

One line out of two transmitted

DCMIPP_HDEC_1_OUT_4

One line out of four transmitted

DCMIPP_HDEC_1_OUT_8

One line out of eight transmitted

DCMIPP Pipe Vertical Decimation Ratio

DCMIPP_VDEC_ALL

All lines captured , no vertical decimation

DCMIPP_VDEC_1_OUT_2

One line out of two transmitted

DCMIPP_VDEC_1_OUT_4

One line out of four transmitted

DCMIPP_VDEC_1_OUT_8

One line out of eight transmitted

DCMIPP PIXCK Polarity

DCMIPP_PCKPOLARITY_FALLING

Pixel clock active on Falling edge

DCMIPP_PCKPOLARITY_RISING

Pixel clock active on Rising edge

DCMIPP Pixel Packer Format

DCMIPP_PIXEL_PACKER_FORMAT_RGB888_YUV444_1

RGB888 or YUV422 1-buffer

DCMIPP_PIXEL_PACKER_FORMAT_RGB565_1

RGB565 1-buffer

DCMIPP_PIXEL_PACKER_FORMAT_ARGB8888

ARGB888 A=0xff

DCMIPP_PIXEL_PACKER_FORMAT_RGBA888

RGBA888 A=0xff)

DCMIPP_PIXEL_PACKER_FORMAT_MONO_Y8_G8_1

Monochrome Y8 or G8

DCMIPP_PIXEL_PACKER_FORMAT_YUV444_1

YUV444 1-buffer 32bpp A=0xff)

DCMIPP_PIXEL_PACKER_FORMAT_YUV422_1

YUV422 1-buffer 16bpp

DCMIPP_PIXEL_PACKER_FORMAT_YUV422_2

YUV422 2-buffer 16bpp

DCMIPP_PIXEL_PACKER_FORMAT_YUV420_2

YUV420 2-buffer 12bpp

DCMIPP_PIXEL_PACKER_FORMAT_YUV420_3

YUV420 3-buffer 12bpp

DCMIPP RAWBayer2RGB Strength

DCMIPP_RAWBAYER_ALGO_NONE

No edge detection, pure linear interpolation

DCMIPP_RAWBAYER_ALGO_STRENGTH_3

RawBayer relative algorithm strength 3

DCMIPP_RAWBAYER_ALGO_STRENGTH_4

RawBayer relative algorithm strength 4

DCMIPP_RAWBAYER_ALGO_STRENGTH_6

RawBayer relative algorithm strength 6

DCMIPP_RAWBAYER_ALGO_STRENGTH_8

RawBayer relative algorithm strength 8

DCMIPP_RAWBAYER_ALGO_STRENGTH_12

RawBayer relative algorithm strength 12

DCMIPP_RAWBAYER_ALGO_STRENGTH_16

RawBayer relative algorithm strength 16

DCMIPP_RAWBAYER_ALGO_STRENGTH_24

RawBayer relative algorithm strength 24

DCMIPP RAWBayer2RGB Type

DCMIPP_RAWBAYER_RGGB

Raw Bayer Type RGGB

DCMIPP_RAWBAYER_GRBG

Raw Bayer Type GRBG

DCMIPP_RAWBAYER_GBRG

Raw Bayer Type GBRG

DCMIPP_RAWBAYER_BGGR

Raw Bayer Type BGGR

DCMIPP Region Of Interest

DCMIPP_REGION_OF_INTEREST1

Region Of Interest 1

DCMIPP_REGION_OF_INTEREST2

Region Of Interest 2

DCMIPP_REGION_OF_INTEREST3

Region Of Interest 3

DCMIPP_REGION_OF_INTEREST4

Region Of Interest 4

DCMIPP_REGION_OF_INTEREST5

Region Of Interest 5

DCMIPP_REGION_OF_INTEREST6

Region Of Interest 6

DCMIPP_REGION_OF_INTEREST7

Region Of Interest 7

DCMIPP_REGION_OF_INTEREST8

Region Of Interest 8

DCMIPP Region Of Interest Line Width

DCMIPP_LINE_WIDTH_1PXL

Line width 1 pixel

DCMIPP_LINE_WIDTH_2PXL

Line width 2 pixels

DCMIPP_LINE_WIDTH_4PXL

Line width 4 pixels

DCMIPP_LINE_WIDTH_8PXL

Line width 8 pixels

DCMIPP Statistics extraction Bins

DCMIPP_STAT_EXT_BINS_MODE_LOWER_BINS

LowerBins: Accu0 (1, 2) is incremented of 256 if Component <4 (<8, <16)

DCMIPP_STAT_EXT_BINS_MODE_LOWMID_BINS

LowMid Bins: Accu0 (1, 2) is incremented of 256 if Component <32 (<64, <128)

DCMIPP_STAT_EXT_BINS_MODE_UPMID_BINS

UpMid Bins: Accu0 (1, 2) is incremented of 256 if Component >127 (>191, >224)

DCMIPP_STAT_EXT_BINS_MODE_UP_BINS

UpBins: Accu0 (1, 2) is incremented of 256 if Component >239 (>247, >251) Condition: MODE = Average

DCMIPP_STAT_EXT_AVER_MODE_ALL_PIXELS

All Pixels: Accu is incremented of Component, if 0 <= Component < 256

DCMIPP_STAT_EXT_AVER_MODE_NOEXT16

NoExt16: Accu is incremented of Component, if 16 <= Component < 240

DCMIPP_STAT_EXT_AVER_MODE_NOEXT32

NoExt32: Accu is incremented of Component, if 32 <= Component < 224

DCMIPP_STAT_EXT_AVER_MODE_NOEXT64

NoExt64: Accu is incremented of Component, if 64 <= Component < 192

DCMIPP Statistics extraction Mode

DCMIPP_STAT_EXT_MODE_AVERAGE

Values of pixels are accumulated as-is

DCMIPP_STAT_EXT_MODE_BINS

Values of pixels are used to see if the pixel fits one of the 12 bins.

DCMIPP Statistics Extraction Module ID

DCMIPP_STATEXT_MODULE1

Statistic Extraction Module 1

DCMIPP_STATEXT_MODULE2

Statistic Extraction Module 2

DCMIPP_STATEXT_MODULE3

Statistic Extraction Module 3

DCMIPP Statistics extraction Source

DCMIPP_STAT_EXT_SOURCE_PRE_BLKLVL_R

Statistics Extraction Source : Red is sampled before the black level calibration and exposure

DCMIPP_STAT_EXT_SOURCE_PRE_BLKLVL_G

Statistics Extraction Source : Green is sampled before the black level calibration and exposure

DCMIPP_STAT_EXT_SOURCE_PRE_BLKLVL_B

Statistics Extraction Source : Blue is sampled before the black level calibration and exposure

DCMIPP_STAT_EXT_SOURCE_PRE_BLKLVL_L

Statistics Extraction Source : Luminance is sampled before the black level calibration and exposure

DCMIPP_STAT_EXT_SOURCE_POST_DEMOS_R

Statistics Extraction Source : Red is sampled after the black level calibration and exposure

DCMIPP_STAT_EXT_SOURCE_POST_DEMOS_G

Statistics Extraction Source : Green is sampled after the black level calibration and exposure

DCMIPP_STAT_EXT_SOURCE_POST_DEMOS_B

Statistics Extraction Source : Blue is sampled after the black level calibration and exposure

DCMIPP_STAT_EXT_SOURCE_POST_DEMOS_L

Statistics Extraction Source : Luminance is sampled after the black level calibration and exposure

DCMIPP Swap Bits

DCMIPP_SWAPBITS_DISABLE

swap lsb vs msb within each received component

DCMIPP_SWAPBITS_ENABLE

swap lsb vs msb within each received component

DCMIPP Swap Component

DCMIPP_SWAP_COMPONENT_DISABLE

Disable Swap R/U and B/V

DCMIPP_SWAP_COMPONENT_ENABLE

Enable Swap R/U and B/V

DCMIPP Swap Cycles

DCMIPP_SWAPCYCLES_DISABLE

swap data from cycle 0 vs cycle 1

DCMIPP_SWAPCYCLES_ENABLE

swap data from cycle 0 vs cycle 1

DCMIPP Synchronization Mode

DCMIPP_SYNCHRO_HARDWARE

Hardware Synchronization

DCMIPP_SYNCHRO_EMBEDDED

Embedded Synchronization

DCMIPP Traffic Burst Size

DCMIPP_TRAFFIC_BURST_SIZE_8BYTES

Traffic Burst size 8 Bytes

DCMIPP_TRAFFIC_BURST_SIZE_16BYTES

Traffic Burst size 16 Bytes

DCMIPP_TRAFFIC_BURST_SIZE_32BYTES

Traffic Burst size 32 Bytes

DCMIPP_TRAFFIC_BURST_SIZE_64BYTES

Traffic Burst size 64 Bytes

DCMIPP_TRAFFIC_BURST_SIZE_128BYTES

Traffic Burst size 128 Bytes

DCMIPP Virtual Channel

DCMIPP_VIRTUAL_CHANNEL0

DCMIPP Virtual Channel 0

DCMIPP_VIRTUAL_CHANNEL1

DCMIPP Virtual Channel 1

DCMIPP_VIRTUAL_CHANNEL2

DCMIPP Virtual Channel 2

DCMIPP_VIRTUAL_CHANNEL3

DCMIPP Virtual Channel 3

DCMIPP VSYNC Polarity

DCMIPP_VSPOLARITY_LOW

Vertical synchronization active Low

DCMIPP_VSPOLARITY_HIGH

Vertical synchronization active High

17 HAL DCMI Generic Driver

17.1 DCMI Firmware driver registers structures

17.1.1 DCMI_CodesInitTypeDef

DCMI_CodesInitTypeDef is defined in the stm32n6xx_hal_dcmi.h

Data Fields

- ***uint8_t FrameStartCode***
- ***uint8_t LineStartCode***
- ***uint8_t LineEndCode***
- ***uint8_t FrameEndCode***

Field Documentation

- ***uint8_t DCMI_CodesInitTypeDef::FrameStartCode***
  Specifies the code of the frame start delimiter.
- ***uint8_t DCMI_CodesInitTypeDef::LineStartCode***
  Specifies the code of the line start delimiter.
- ***uint8_t DCMI_CodesInitTypeDef::LineEndCode***
  Specifies the code of the line end delimiter.
- ***uint8_t DCMI_CodesInitTypeDef::FrameEndCode***
  Specifies the code of the frame end delimiter.

17.1.2 DCMI_SyncUnmaskTypeDef

DCMI_SyncUnmaskTypeDef is defined in the stm32n6xx_hal_dcmi.h

Data Fields

- ***uint8_t FrameStartUnmask***
- ***uint8_t LineStartUnmask***
- ***uint8_t LineEndUnmask***
- ***uint8_t FrameEndUnmask***

Field Documentation

- ***uint8_t DCMI_SyncUnmaskTypeDef::FrameStartUnmask***
  Specifies the frame start delimiter unmask.
- ***uint8_t DCMI_SyncUnmaskTypeDef::LineStartUnmask***
  Specifies the line start delimiter unmask.
- ***uint8_t DCMI_SyncUnmaskTypeDef::LineEndUnmask***
  Specifies the line end delimiter unmask.
- ***uint8_t DCMI_SyncUnmaskTypeDef::FrameEndUnmask***
  Specifies the frame end delimiter unmask.

17.1.3 DCMI_InitTypeDef

DCMI_InitTypeDef is defined in the stm32n6xx_hal_dcmi.h

Data Fields

- ***uint32_t SynchroMode***
- ***uint32_t PCKPolarity***
- ***uint32_t VSPolarity***
- ***uint32_t HSPolarity***
- ***uint32_t CaptureRate***
- ***uint32_t ExtendedDataMode***
- ***DCMI_CodesInitTypeDef SyncroCode***
- ***uint32_t JPEGMode***
- ***uint32_t ByteSelectMode***

- *uint32_t ByteSelectStart*
- *uint32_t LineSelectMode*
- *uint32_t LineSelectStart*

Field Documentation

- ***uint32_t DCMI_InitTypeDef::SynchroMode***
  Specifies the Synchronization Mode: Hardware or Embedded. This parameter can be a value of *DCMI_Synchronization_Mode*
- ***uint32_t DCMI_InitTypeDef::PCKPolarity***
  Specifies the Pixel clock polarity: Falling or Rising. This parameter can be a value of *DCMI_PIXCK_Polarity*
- ***uint32_t DCMI_InitTypeDef::VSPolarity***
  Specifies the Vertical synchronization polarity: High or Low. This parameter can be a value of *DCMI_VSYNC_Polarity*
- ***uint32_t DCMI_InitTypeDef::HSPolarity***
  Specifies the Horizontal synchronization polarity: High or Low. This parameter can be a value of *DCMI_HSYNC_Polarity*
- ***uint32_t DCMI_InitTypeDef::CaptureRate***
  Specifies the frequency of frame capture: All, 1/2 or 1/4. This parameter can be a value of *DCMI_Capture_Rate*
- ***uint32_t DCMI_InitTypeDef::ExtendedDataMode***
  Specifies the data width: 8-bit, 10-bit, 12-bit or 14-bit. This parameter can be a value of *DCMI_Extended_Data_Mode*
- ***DCMI_CodesInitTypeDef DCMI_InitTypeDef::SyncroCode***
  Specifies the code of the line/frame start delimiter and the line/frame end delimiter
- ***uint32_t DCMI_InitTypeDef::JPEGMode***
  Enable or Disable the JPEG mode. This parameter can be a value of *DCMI_MODE_JPEG*
- ***uint32_t DCMI_InitTypeDef::ByteSelectMode***
  Specifies the data to be captured by the interface This parameter can be a value of *DCMI_Byte_Select_Mode*
- ***uint32_t DCMI_InitTypeDef::ByteSelectStart***
  Specifies if the data to be captured by the interface is even or odd This parameter can be a value of *DCMI_Byte_Select_Start*
- ***uint32_t DCMI_InitTypeDef::LineSelectMode***
  Specifies the line of data to be captured by the interface This parameter can be a value of *DCMI_Line_Select_Mode*
- ***uint32_t DCMI_InitTypeDef::LineSelectStart***
  Specifies if the line of data to be captured by the interface is even or odd This parameter can be a value of *DCMI_Line_Select_Start*

17.1.4 __DCMI_HandleTypeDef

__DCMI_HandleTypeDef is defined in the stm32n6xx_hal_dcmi.h

Data Fields

- *DCMI_TypeDef * Instance*
- *DCMI_InitTypeDef Init*
- *HAL_LockTypeDef Lock*
- *__IO HAL_DCMI_StateTypeDef State*
- *__IO uint32_t XferCount*
- *__IO uint32_t XferSize*
- *uint32_t XferTransferNumber*
- *uint32_t pBuffPtr*
- *DMA_HandleTypeDef * DMA_Handle*
- *__IO uint32_t ErrorCode*
- *void(* FrameEventCallback*
- *void(* VsyncEventCallback*

- ***void(* LineEventCallback***
- ***void(* ErrorCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***DCMI_TypeDef* __DCMI_HandleTypeDef::Instance***
  DCMI Register base address
- ***DCMI_InitTypeDef __DCMI_HandleTypeDef::Init***
  DCMI parameters
- ***HAL_LockTypeDef __DCMI_HandleTypeDef::Lock***
  DCMI locking object
- ***__IO HAL_DCMI_StateTypeDef __DCMI_HandleTypeDef::State***
  DCMI state
- ***__IO uint32_t __DCMI_HandleTypeDef::XferCount***
  DMA transfer counter
- ***__IO uint32_t __DCMI_HandleTypeDef::XferSize***
  DMA transfer size
- ***uint32_t __DCMI_HandleTypeDef::XferTransferNumber***
  DMA transfer number
- ***uint32_t __DCMI_HandleTypeDef::pBuffPtr***
  Pointer to DMA output buffer
- ***DMA_HandleTypeDef* __DCMI_HandleTypeDef::DMA_Handle***
  Pointer to the DMA handler
- ***__IO uint32_t __DCMI_HandleTypeDef::ErrorCode***
  DCMI Error code
- ***void(* __DCMI_HandleTypeDef::FrameEventCallback)(struct __DCMI_HandleTypeDef *hdcmi)***
  DCMI Frame Event Callback
- ***void(* __DCMI_HandleTypeDef::VsyncEventCallback)(struct __DCMI_HandleTypeDef *hdcmi)***
  DCMI Vsync Event Callback
- ***void(* __DCMI_HandleTypeDef::LineEventCallback)(struct __DCMI_HandleTypeDef *hdcmi)***
  DCMI Line Event Callback
- ***void(* __DCMI_HandleTypeDef::ErrorCallback)(struct __DCMI_HandleTypeDef *hdcmi)***
  DCMI Error Callback
- ***void(* __DCMI_HandleTypeDef::MspInitCallback)(struct __DCMI_HandleTypeDef *hdcmi)***
  DCMI Msp Init callback
- ***void(* __DCMI_HandleTypeDef::MspDeInitCallback)(struct __DCMI_HandleTypeDef *hdcmi)***
  DCMI Msp DeInit callback

17.2 DCMI Firmware driver API description

The following section lists the various functions of the DCMI library.

17.2.1 How to use this driver

The sequence below describes how to use this driver to capture image from a camera module connected to the DCMI Interface. This sequence does not take into account the configuration of the camera module, which should be made before to configure and enable the DCMI to capture images.

1. Program the required configuration through following parameters: horizontal and vertical polarity, pixel clock polarity, Capture Rate, Synchronization Mode, code of the frame delimiter and data width using HAL_DCMI_Init() function.
2. Configure the selected DMA stream to transfer Data from DCMI DR register to the destination memory buffer.
3. Program the required configuration through following parameters: DCMI mode, destination memory Buffer address and the data length and enable capture using HAL_DCMI_Start_DMA() function.
4. Optionally, configure and Enable the CROP feature to select a rectangular window from the received image using HAL_DCMI_ConfigCrop() and HAL_DCMI_EnableCrop() functions
5. The capture can be stopped using HAL_DCMI_Stop() function.

6. To control DCMI state you can use the function HAL_DCMI_GetState().

DCMI HAL driver macros list

Below the list of most used macros in DCMI HAL driver.

- __HAL_DCMI_ENABLE: Enable the DCMI peripheral.
- __HAL_DCMI_DISABLE: Disable the DCMI peripheral.
- __HAL_DCMI_GET_FLAG: Get the DCMI pending flags.
- __HAL_DCMI_CLEAR_FLAG: Clear the DCMI pending flags.
- __HAL_DCMI_ENABLE_IT: Enable the specified DCMI interrupts.
- __HAL_DCMI_DISABLE_IT: Disable the specified DCMI interrupts.
- __HAL_DCMI_GET_IT_SOURCE: Check whether the specified DCMI interrupt has occurred or not.

Note: You can refer to the DCMI HAL driver header file for more useful macros

Callback registration

17.2.2 Detailed description of functions

HAL_DCMI_Init

Function name

HAL_StatusTypeDef HAL_DCMI_Init (DCMI_HandleTypeDef * hdcmi)

Function description

HAL_DCMI_DeInit

Function name

HAL_StatusTypeDef HAL_DCMI_DeInit (DCMI_HandleTypeDef * hdcmi)

Function description

HAL_DCMI_MspInit

Function name

void HAL_DCMI_MspInit (DCMI_HandleTypeDef * hdcmi)

Function description

HAL_DCMI_MspDeInit

Function name

void HAL_DCMI_MspDeInit (DCMI_HandleTypeDef * hdcmi)

Function description

HAL_DCMI_RegisterCallback

Function name

HAL_StatusTypeDef HAL_DCMI_RegisterCallback (DCMI_HandleTypeDef * hdcmi, HAL_DCMI_CallbackIDTypeDef CallbackID, pDCMI_CallbackTypeDef pCallback)

Function description

HAL_DCMI_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_DCMI_UnRegisterCallback (DCMI_HandleTypeDef * hdcmi, HAL_DCMI_CallbackIDTypeDef CallbackID)

Function description

HAL_DCMI_Start_DMA

Function name

HAL_StatusTypeDef HAL_DCMI_Start_DMA (DCMI_HandleTypeDef * hdcmi, uint32_t DCMI_Mode, uint32_t pData, uint32_t Length)

Function description

HAL_DCMI_Stop

Function name

HAL_StatusTypeDef HAL_DCMI_Stop (DCMI_HandleTypeDef * hdcmi)

Function description

HAL_DCMI_Suspend

Function name

HAL_StatusTypeDef HAL_DCMI_Suspend (DCMI_HandleTypeDef * hdcmi)

Function description

HAL_DCMI_Resume

Function name

HAL_StatusTypeDef HAL_DCMI_Resume (DCMI_HandleTypeDef * hdcmi)

Function description

HAL_DCMI_ErrorCallback

Function name

void HAL_DCMI_ErrorCallback (DCMI_HandleTypeDef * hdcmi)

Function description

HAL_DCMI_LineEventCallback

Function name

void HAL_DCMI_LineEventCallback (DCMI_HandleTypeDef * hdcmi)

Function description

HAL_DCMI_FrameEventCallback

Function name

void HAL_DCMI_FrameEventCallback (DCMI_HandleTypeDef * hdcmi)

Function description

HAL_DCMI_VsyncEventCallback

Function name

void HAL_DCMI_VsyncEventCallback (DCMI_HandleTypeDef * hdcmi)

Function description

HAL_DCMI_IRQHandler

Function name

void HAL_DCMI_IRQHandler (DCMI_HandleTypeDef * hdcmi)

Function description

HAL_DCMI_ConfigCrop

Function name

HAL_StatusTypeDef HAL_DCMI_ConfigCrop (DCMI_HandleTypeDef * hdcmi, uint32_t X0, uint32_t Y0, uint32_t XSize, uint32_t YSize)

Function description

HAL_DCMI_EnableCrop

Function name

HAL_StatusTypeDef HAL_DCMI_EnableCrop (DCMI_HandleTypeDef * hdcmi)

Function description

HAL_DCMI_DisableCrop

Function name

HAL_StatusTypeDef HAL_DCMI_DisableCrop (DCMI_HandleTypeDef * hdcmi)

Function description

HAL_DCMI_ConfigSyncUnmask

Function name

HAL_StatusTypeDef HAL_DCMI_ConfigSyncUnmask (DCMI_HandleTypeDef * hdcmi, const DCMI_SyncUnmaskTypeDef * SyncUnmask)

Function description

HAL_DCMI_GetState

Function name

HAL_DCMI_StateTypeDef HAL_DCMI_GetState (const DCMI_HandleTypeDef * hdcmi)

Function description

HAL_DCMI_GetError

Function name

uint32_t HAL_DCMI_GetError (const DCMI_HandleTypeDef * hdcmi)

Function description

17.3 DCMI Firmware driver defines

The following section lists the various define and macros of the module.

17.3.1 DCMI

DCMI

DCMI Byte Select Mode

DCMI_BSM_ALL

Interface captures all received data

DCMI_BSM_OTHER

Interface captures every other byte from the received data

DCMI_BSM_ALTERNATE_4

Interface captures one byte out of four

DCMI_BSM_ALTERNATE_2

Interface captures two bytes out of four

DCMI Byte Select Start

DCMI_OEBS_ODD

Interface captures first data from the frame/line start, second one being dropped

DCMI_OEBS_EVEN

Interface captures second data from the frame/line start, first one being dropped

DCMI Capture Mode

DCMI_MODE_CONTINUOUS

The received data are transferred continuously into the destination memory through the DMA

DCMI_MODE_SNAPSHOT

Once activated, the interface waits for the start of frame and then transfers a single frame through the DMA

DCMI Capture Rate

DCMI_CR_ALL_FRAME

All frames are captured

DCMI_CR_ALTERNATE_2_FRAME

Every alternate frame captured

DCMI_CR_ALTERNATE_4_FRAME

One frame in 4 frames captured

DCMI Error Code

HAL_DCMI_ERROR_NONE

No error

HAL_DCMI_ERROR_OVR

Overrun error

HAL_DCMI_ERROR_SYNC

Synchronization error

HAL_DCMI_ERROR_TIMEOUT

Timeout error

HAL_DCMI_ERROR_DMA

DMA error

HAL_DCMI_ERROR_INVALID_CALLBACK

Invalid callback error

DCMI Exported Macros

__HAL_DCMI_RESET_HANDLE_STATE

Description:

- Reset DCMI handle state.

Parameters:

- __HANDLE__: specifies the DCMI handle.

Return value:

- None

__HAL_DCMI_ENABLE

Description:

- Enable the DCMI.

Parameters:

- __HANDLE__: DCMI handle

Return value:

- None

__HAL_DCMI_DISABLE

Description:

- Disable the DCMI.

Parameters:

- __HANDLE__: DCMI handle

Return value:

- None

__HAL_DCMI_GET_FLAG

Description:

- Get the DCMI pending flag.

Parameters:

- __HANDLE__: DCMI handle
- __FLAG__: Get the specified flag. This parameter can be one of the following values (no combination allowed)
  - DCMI_FLAG_HSYNC: HSYNC pin state (active line / synchronization between lines)
  - DCMI_FLAG_VSYNC: VSYNC pin state (active frame / synchronization between frames)
  - DCMI_FLAG_FNE: FIFO empty flag
  - DCMI_FLAG_FRAMERI: Frame capture complete flag mask
  - DCMI_FLAG_OVRRI: Overrun flag mask
  - DCMI_FLAG_ERRRI: Synchronization error flag mask
  - DCMI_FLAG_VSYNCRI: VSYNC flag mask
  - DCMI_FLAG_LINERI: Line flag mask
  - DCMI_FLAG_FRAMEMI: DCMI Capture complete masked interrupt status
  - DCMI_FLAG_OVRMI: DCMI Overrun masked interrupt status
  - DCMI_FLAG_ERRMI: DCMI Synchronization error masked interrupt status
  - DCMI_FLAG_VSYNCMI: DCMI VSYNC masked interrupt status
  - DCMI_FLAG_LINEMI: DCMI Line masked interrupt status

Return value:

- The: state of FLAG.

__HAL_DCMI_CLEAR_FLAG

Description:

- Clear the DCMI pending flags.

Parameters:

- __HANDLE__: DCMI handle
- __FLAG__: specifies the flag to clear. This parameter can be any combination of the following values:
  - DCMI_FLAG_FRAMERI: Frame capture complete flag mask
  - DCMI_FLAG_OVFRI: Overflow flag mask
  - DCMI_FLAG_ERRRI: Synchronization error flag mask
  - DCMI_FLAG_VSYNCRI: VSYNC flag mask
  - DCMI_FLAG_LINERI: Line flag mask

Return value:

- None

__HAL_DCMI_ENABLE_IT

Description:

- Enable the specified DCMI interrupts.

Parameters:

- __HANDLE__: DCMI handle
- __INTERRUPT__: specifies the DCMI interrupt sources to be enabled. This parameter can be any combination of the following values:
  - DCMI_IT_FRAME: Frame capture complete interrupt mask
  - DCMI_IT_OVF: Overflow interrupt mask
  - DCMI_IT_ERR: Synchronization error interrupt mask
  - DCMI_IT_VSYNC: VSYNC interrupt mask
  - DCMI_IT_LINE: Line interrupt mask

Return value:

- None

__HAL_DCMI_DISABLE_IT

Description:

- Disable the specified DCMI interrupts.

Parameters:

- __HANDLE__: DCMI handle
- __INTERRUPT__: specifies the DCMI interrupt sources to be enabled. This parameter can be any combination of the following values:
  - DCMI_IT_FRAME: Frame capture complete interrupt mask
  - DCMI_IT_OVF: Overflow interrupt mask
  - DCMI_IT_ERR: Synchronization error interrupt mask
  - DCMI_IT_VSYNC: VSYNC interrupt mask
  - DCMI_IT_LINE: Line interrupt mask

Return value:

- None

__HAL_DCMI_GET_IT_SOURCE

Description:

- Check whether the specified DCMI interrupt has occurred or not.

Parameters:

- __HANDLE__: DCMI handle
- __INTERRUPT__: specifies the DCMI interrupt source to check. This parameter can be one of the following values:
  - DCMI_IT_FRAME: Frame capture complete interrupt mask
  - DCMI_IT_OVF: Overflow interrupt mask
  - DCMI_IT_ERR: Synchronization error interrupt mask
  - DCMI_IT_VSYNC: VSYNC interrupt mask
  - DCMI_IT_LINE: Line interrupt mask

Return value:

- The: state of INTERRUPT.

DCMI Extended Data Mode

DCMI_EXTEND_DATA_8B

Interface captures 8-bit data on every pixel clock

DCMI_EXTEND_DATA_10B

Interface captures 10-bit data on every pixel clock

DCMI_EXTEND_DATA_12B

Interface captures 12-bit data on every pixel clock

DCMI_EXTEND_DATA_14B

Interface captures 14-bit data on every pixel clock

DCMI Flags

DCMI_FLAG_HSYNC

HSYNC pin state (active line / synchronization between lines)

DCMI_FLAG_VSYNC

VSYNC pin state (active frame / synchronization between frames)

DCMI_FLAG_FNE

FIFO not empty flag

DCMI_FLAG_FRAMERI

Frame capture complete interrupt flag

DCMI_FLAG_OVRRI

Overrun interrupt flag

DCMI_FLAG_ERRRI

Synchronization error interrupt flag

DCMI_FLAG_VSYNCRI

VSYNC interrupt flag

DCMI_FLAG_LINERI

Line interrupt flag

DCMI_FLAG_FRAMEMI

DCMI Frame capture complete masked interrupt status

DCMI_FLAG_OVRMI

DCMI Overrun masked interrupt status

DCMI_FLAG_ERRMI

DCMI Synchronization error masked interrupt status

DCMI_FLAG_VSYNCMI

DCMI VSYNC masked interrupt status

DCMI_FLAG_LINEMI

DCMI Line masked interrupt status

DCMI HSYNC Polarity

DCMI_HSPOLARITY_LOW

Horizontal synchronization active Low

DCMI_HSPOLARITY_HIGH

Horizontal synchronization active High

DCMI interrupt sources

DCMI_IT_FRAME

Capture complete interrupt

DCMI_IT_OVR

Overrun interrupt

DCMI_IT_ERR

Synchronization error interrupt

DCMI_IT_VSYNC

VSYNC interrupt

DCMI_IT_LINE

Line interrupt

DCMI Line Select Mode

DCMI_LSM_ALL

Interface captures all received lines

DCMI_LSM_ALTERNATE_2

Interface captures one line out of two

DCMI Line Select Start

DCMI_OELS_ODD

Interface captures first line from the frame start, second one being dropped

DCMI_OELS_EVEN

Interface captures second line from the frame start, first one being dropped

DCMI Mis Index

DCMI_MIS_INDEX

DCMI MIS register index

DCMI MODE JPEG

DCMI_JPEG_DISABLE

Mode JPEG Disabled

DCMI_JPEG_ENABLE

Mode JPEG Enabled

DCMI PIXCK Polarity

DCMI_PCKPOLARITY_FALLING

Pixel clock active on Falling edge

DCMI_PCKPOLARITY_RISING

Pixel clock active on Rising edge

DCMI SR Index

DCMI_SR_INDEX

DCMI SR register index

DCMI Synchronization Mode

DCMI_SYNCHRO_HARDWARE

Hardware synchronization data capture (frame/line start/stop) is synchronized with the HSYNC/VSYNC signals

DCMI_SYNCHRO_EMBEDDED

Embedded synchronization data capture is synchronized with synchronization codes embedded in the data flow

DCMI VSYNC Polarity

DCMI_VSPOLARITY_LOW

Vertical synchronization active Low

DCMI_VSPOLARITY_HIGH

Vertical synchronization active High

DCMI Window Coordinate

DCMI_WINDOW_COORDINATE

Window coordinate

DCMI Window Height

DCMI_WINDOW_HEIGHT

Window Height

18 HAL DMA2D Generic Driver

18.1 DMA2D Firmware driver registers structures

18.1.1 DMA2D_CLUTCfgTypeDef

DMA2D_CLUTCfgTypeDef is defined in the stm32n6xx_hal_dma2d.h

Data Fields

- ***uint32_t * pCLUT***
- ***uint32_t CLUTColorMode***
- ***uint32_t Size***

Field Documentation

- ***uint32_t* DMA2D_CLUTCfgTypeDef::pCLUT***
  Configures the DMA2D CLUT memory address.
- ***uint32_t DMA2D_CLUTCfgTypeDef::CLUTColorMode***
  Configures the DMA2D CLUT color mode. This parameter can be one value of ***DMA2D_CLUT_CM***.
- ***uint32_t DMA2D_CLUTCfgTypeDef::Size***
  Configures the DMA2D CLUT size. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF.

18.1.2 DMA2D_InitTypeDef

DMA2D_InitTypeDef is defined in the stm32n6xx_hal_dma2d.h

Data Fields

- ***uint32_t Mode***
- ***uint32_t ColorMode***
- ***uint32_t OutputOffset***
- ***uint32_t AlphaInverted***
- ***uint32_t RedBlueSwap***
- ***uint32_t BytesSwap***
- ***uint32_t LineOffsetMode***

Field Documentation

- ***uint32_t DMA2D_InitTypeDef::Mode***
  Configures the DMA2D transfer mode. This parameter can be one value of ***DMA2D_Mode***.
- ***uint32_t DMA2D_InitTypeDef::ColorMode***
  Configures the color format of the output image. This parameter can be one value of ***DMA2D_Output_Color_Mode***.
- ***uint32_t DMA2D_InitTypeDef::OutputOffset***
  Specifies the Offset value. This parameter must be a number between Min_Data = 0x0000 and Max_Data = 0x3FFF.
- ***uint32_t DMA2D_InitTypeDef::AlphaInverted***
  Select regular or inverted alpha value for the output pixel format converter. This parameter can be one value of ***DMA2D_Alpha_Inverted***.
- ***uint32_t DMA2D_InitTypeDef::RedBlueSwap***
  Select regular mode (RGB or ARGB) or swap mode (BGR or ABGR) for the output pixel format converter. This parameter can be one value of ***DMA2D_RB_Swap***.
- ***uint32_t DMA2D_InitTypeDef::BytesSwap***
  Select byte regular mode or bytes swap mode (two by two). This parameter can be one value of ***DMA2D_Bytes_Swap***.
- ***uint32_t DMA2D_InitTypeDef::LineOffsetMode***
  Configures how is expressed the line offset for the foreground, background and output. This parameter can be one value of ***DMA2D_Line_Offset_Mode***.

18.1.3 DMA2D_LayerCfgTypeDef

DMA2D_LayerCfgTypeDef is defined in the stm32n6xx_hal_dma2d.h

Data Fields

- ***uint32_t InputOffset***
- ***uint32_t InputColorMode***
- ***uint32_t AlphaMode***
- ***uint32_t InputAlpha***
- ***uint32_t AlphaInverted***
- ***uint32_t RedBlueSwap***
- ***uint32_t ChromaSubSampling***

Field Documentation

- ***uint32_t DMA2D_LayerCfgTypeDef::InputOffset***
  Configures the DMA2D foreground or background offset. This parameter must be a number between Min_Data = 0x0000 and Max_Data = 0x3FFF.
- ***uint32_t DMA2D_LayerCfgTypeDef::InputColorMode***
  Configures the DMA2D foreground or background color mode. This parameter can be one value of ***DMA2D_Input_Color_Mode***.
- ***uint32_t DMA2D_LayerCfgTypeDef::AlphaMode***
  Configures the DMA2D foreground or background alpha mode. This parameter can be one value of ***DMA2D_Alpha_Mode***.
- ***uint32_t DMA2D_LayerCfgTypeDef::InputAlpha***
  Specifies the DMA2D foreground or background alpha value and color value in case of A8 or A4 color mode. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF except for the color modes detailed below.
  Note:
  – In case of A8 or A4 color mode (ARGB), this parameter must be a number between Min_Data = 0x00000000 and Max_Data = 0xFFFFFFFF where
    ◦ InputAlpha[24:31] is the alpha value ALPHA[0:7]
    ◦ InputAlpha[16:23] is the red value RED[0:7]
    ◦ InputAlpha[8:15] is the green value GREEN[0:7]
    ◦ InputAlpha[0:7] is the blue value BLUE[0:7].
- ***uint32_t DMA2D_LayerCfgTypeDef::AlphaInverted***
  Select regular or inverted alpha value. This parameter can be one value of ***DMA2D_Alpha_Inverted***.
- ***uint32_t DMA2D_LayerCfgTypeDef::RedBlueSwap***
  Select regular mode (RGB or ARGB) or swap mode (BGR or ABGR). This parameter can be one value of ***DMA2D_RB_Swap***.
- ***uint32_t DMA2D_LayerCfgTypeDef::ChromaSubSampling***
  Configure the chroma sub-sampling mode for the YCbCr color mode This parameter can be one value of ***DMA2D_Chroma_Sub_Sampling***

18.1.4 __DMA2D_HandleTypeDef

__DMA2D_HandleTypeDef is defined in the stm32n6xx_hal_dma2d.h

Data Fields

- ***DMA2D_TypeDef * Instance***
- ***DMA2D_InitTypeDef Init***
- ***void(* XferCpltCallback***
- ***void(* XferErrorCallback***
- ***void(* LineEventCallback***
- ***void(* CLUTLoadingCpltCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***
- ***DMA2D_LayerCfgTypeDef LayerCfg***
- ***HAL_LockTypeDef Lock***

- ***__IO HAL_DMA2D_StateTypeDef State***
- ***__IO uint32_t ErrorCode***

Field Documentation

- ***DMA2D_TypeDef* __DMA2D_HandleTypeDef::Instance***
  DMA2D register base address.
- ***DMA2D_InitTypeDef __DMA2D_HandleTypeDef::Init***
  DMA2D communication parameters.
- ***void(* __DMA2D_HandleTypeDef::XferCpltCallback)(struct __DMA2D_HandleTypeDef *hdma2d)***
  DMA2D transfer complete callback.
- ***void(* __DMA2D_HandleTypeDef::XferErrorCallback)(struct __DMA2D_HandleTypeDef *hdma2d)***
  DMA2D transfer error callback.
- ***void(* __DMA2D_HandleTypeDef::LineEventCallback)(struct __DMA2D_HandleTypeDef *hdma2d)***
  DMA2D line event callback.
- ***void(* __DMA2D_HandleTypeDef::CLUTLoadingCpltCallback)(struct __DMA2D_HandleTypeDef *hdma2d)***
  DMA2D CLUT loading completion callback
- ***void(* __DMA2D_HandleTypeDef::MspInitCallback)(struct __DMA2D_HandleTypeDef *hdma2d)***
  DMA2D Msp Init callback.
- ***void(* __DMA2D_HandleTypeDef::MspDeInitCallback)(struct __DMA2D_HandleTypeDef *hdma2d)***
  DMA2D Msp DeInit callback.
- ***DMA2D_LayerCfgTypeDef __DMA2D_HandleTypeDef::LayerCfg[MAX_DMA2D_LAYER]***
  DMA2D Layers parameters
- ***HAL_LockTypeDef __DMA2D_HandleTypeDef::Lock***
  DMA2D lock.
- ***__IO HAL_DMA2D_StateTypeDef __DMA2D_HandleTypeDef::State***
  DMA2D transfer state.
- ***__IO uint32_t __DMA2D_HandleTypeDef::ErrorCode***
  DMA2D error code.

18.2 DMA2D Firmware driver API description

The following section lists the various functions of the DMA2D library.

18.2.1 How to use this driver

1. Program the required configuration through the following parameters: the transfer mode, the output color mode and the output offset using HAL_DMA2D_Init() function.
2. Program the required configuration through the following parameters: the input color mode, the input color, the input alpha value, the alpha mode, the red/blue swap mode, the inverted alpha mode and the input offset using HAL_DMA2D_ConfigLayer() function for foreground or/and background layer.

Polling mode IO operation

1. Configure pdata parameter (explained hereafter), destination and data length and enable the transfer using HAL_DMA2D_Start().
2. Wait for end of transfer using HAL_DMA2D_PollForTransfer(), at this stage user can specify the value of timeout according to his end application.

Interrupt mode IO operation

1. Configure pdata parameter, destination and data length and enable the transfer using HAL_DMA2D_Start_IT().
2. Use HAL_DMA2D_IRQHandler() called under DMA2D_IRQHandler() interrupt subroutine.
3. At the end of data transfer HAL_DMA2D_IRQHandler() function is executed and user can add his own function by customization of function pointer XferCpltCallback (member of DMA2D handle structure).

4. In case of error, the HAL_DMA2D_IRQHandler() function calls the callback XferErrorCallback.

Note: In Register-to-Memory transfer mode, pdata parameter is the register color, in Memory-to-memory or Memory-to-Memory with pixel format conversion pdata is the source address.

Note: Configure the foreground source address, the background source address, the destination and data length then Enable the transfer using HAL_DMA2D_BlendingStart() in polling mode and HAL_DMA2D_BlendingStart_IT() in interrupt mode.

Note: HAL_DMA2D_BlendingStart() and HAL_DMA2D_BlendingStart_IT() functions are used if the memory to memory with blending transfer mode is selected.

5. Optionally, configure and enable the CLUT using HAL_DMA2D_CLUTLoad() in polling mode or HAL_DMA2D_CLUTLoad_IT() in interrupt mode.
6. Optionally, configure the line watermark in using the API HAL_DMA2D_ProgramLineEvent().
7. Optionally, configure the dead time value in the AHB clock cycle inserted between two consecutive accesses on the AHB master port in using the API HAL_DMA2D_ConfigDeadTime() and enable/disable the functionality with the APIs HAL_DMA2D_EnableDeadTime() or HAL_DMA2D_DisableDeadTime().
8. The transfer can be suspended, resumed and aborted using the following functions: HAL_DMA2D_Suspend(), HAL_DMA2D_Resume(), HAL_DMA2D_Abort().
9. The CLUT loading can be suspended, resumed and aborted using the following functions: HAL_DMA2D_CLUTLoading_Suspend(), HAL_DMA2D_CLUTLoading_Resume(), HAL_DMA2D_CLUTLoading_Abort().
10. To control the DMA2D state, use the following function: HAL_DMA2D_GetState().
11. To read the DMA2D error code, use the following function: HAL_DMA2D_GetError().

DMA2D HAL driver macros list

Below the list of most used macros in DMA2D HAL driver :

- __HAL_DMA2D_ENABLE: Enable the DMA2D peripheral.
- __HAL_DMA2D_GET_FLAG: Get the DMA2D pending flags.
- __HAL_DMA2D_CLEAR_FLAG: Clear the DMA2D pending flags.
- __HAL_DMA2D_ENABLE_IT: Enable the specified DMA2D interrupts.
- __HAL_DMA2D_DISABLE_IT: Disable the specified DMA2D interrupts.
- __HAL_DMA2D_GET_IT_SOURCE: Check whether the specified DMA2D interrupt is enabled or not.

Callback registration

1. The compilation define USE_HAL_DMA2D_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use function HAL_DMA2D_RegisterCallback() to register a user callback.
2. Function HAL_DMA2D_RegisterCallback() allows to register following callbacks:
   - XferCpltCallback : callback for transfer complete.
   - XferErrorCallback : callback for transfer error.
   - LineEventCallback : callback for line event.
   - CLUTLoadingCpltCallback : callback for CLUT loading completion.
   - MspInitCallback : DMA2D MspInit.
   - MspDeInitCallback : DMA2D MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.
3. Use function HAL_DMA2D_UnRegisterCallback() to reset a callback to the default weak (surcharged) function. HAL_DMA2D_UnRegisterCallback() takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks:
   - XferCpltCallback : callback for transfer complete.
   - XferErrorCallback : callback for transfer error.
   - LineEventCallback : callback for line event.
   - CLUTLoadingCpltCallback : callback for CLUT loading completion.
   - MspInitCallback : DMA2D MspInit.
   - MspDeInitCallback : DMA2D MspDeInit.

4. By default, after the HAL_DMA2D_Init and if the state is HAL_DMA2D_STATE_RESET all callbacks are reset to the corresponding legacy weak (surcharged) functions: examples HAL_DMA2D_LineEventCallback(), HAL_DMA2D_CLUTLoadingCpltCallback() Exception done for MspInit and MspDeInit callbacks that are respectively reset to the legacy weak (surcharged) functions in the HAL_DMA2D_Init and HAL_DMA2D_DeInit only when these callbacks are null (not registered beforehand) If not, MspInit or MspDeInit are not null, the HAL_DMA2D_Init and HAL_DMA2D_DeInit keep and use the user MspInit/MspDeInit callbacks (registered beforehand). Exception as well for Transfer Completion and Transfer Error callbacks that are not defined as weak (surcharged) functions. They must be defined by the user to be resorted to.
5. Callbacks can be registered/unregistered in READY state only. Exception done for MspInit/MspDeInit callbacks that can be registered/unregistered in READY or RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_DMA2D_RegisterCallback before calling HAL_DMA2D_DeInit or HAL_DMA2D_Init function.
6. When The compilation define USE_HAL_DMA2D_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak (surcharged) callbacks are used.

Note: You can refer to the DMA2D HAL driver header file for more useful macros

18.2.2 Detailed description of functions

HAL_DMA2D_Init

Function name

HAL_StatusTypeDef HAL_DMA2D_Init (DMA2D_HandleTypeDef * hdma2d)

Function description

HAL_DMA2D_DeInit

Function name

HAL_StatusTypeDef HAL_DMA2D_DeInit (DMA2D_HandleTypeDef * hdma2d)

Function description

HAL_DMA2D_MspInit

Function name

void HAL_DMA2D_MspInit (DMA2D_HandleTypeDef * hdma2d)

Function description

HAL_DMA2D_MspDeInit

Function name

void HAL_DMA2D_MspDeInit (DMA2D_HandleTypeDef * hdma2d)

Function description

HAL_DMA2D_RegisterCallback

Function name

HAL_StatusTypeDef HAL_DMA2D_RegisterCallback (DMA2D_HandleTypeDef * hdma2d, HAL_DMA2D_CallbackIDTypeDef CallbackID, pDMA2D_CallbackTypeDef pCallback)

Function description

HAL_DMA2D_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_DMA2D_UnRegisterCallback (DMA2D_HandleTypeDef * hdma2d, HAL_DMA2D_CallbackIDTypeDef CallbackID)

Function description

HAL_DMA2D_Start

Function name

HAL_StatusTypeDef HAL_DMA2D_Start (DMA2D_HandleTypeDef * hdma2d, uint32_t pdata, uint32_t DstAddress, uint32_t Width, uint32_t Height)

Function description

HAL_DMA2D_BlendingStart

Function name

HAL_StatusTypeDef HAL_DMA2D_BlendingStart (DMA2D_HandleTypeDef * hdma2d, uint32_t SrcAddress1, uint32_t SrcAddress2, uint32_t DstAddress, uint32_t Width, uint32_t Height)

Function description

HAL_DMA2D_Start_IT

Function name

HAL_StatusTypeDef HAL_DMA2D_Start_IT (DMA2D_HandleTypeDef * hdma2d, uint32_t pdata, uint32_t DstAddress, uint32_t Width, uint32_t Height)

Function description

HAL_DMA2D_BlendingStart_IT

Function name

HAL_StatusTypeDef HAL_DMA2D_BlendingStart_IT (DMA2D_HandleTypeDef * hdma2d, uint32_t SrcAddress1, uint32_t SrcAddress2, uint32_t DstAddress, uint32_t Width, uint32_t Height)

Function description

HAL_DMA2D_Suspend

Function name

HAL_StatusTypeDef HAL_DMA2D_Suspend (DMA2D_HandleTypeDef * hdma2d)

Function description

HAL_DMA2D_Resume

Function name

HAL_StatusTypeDef HAL_DMA2D_Resume (DMA2D_HandleTypeDef * hdma2d)

Function description

HAL_DMA2D_Abort

Function name

HAL_StatusTypeDef HAL_DMA2D_Abort (DMA2D_HandleTypeDef * hdma2d)

Function description

HAL_DMA2D_EnableCLUT

Function name

HAL_StatusTypeDef HAL_DMA2D_EnableCLUT (DMA2D_HandleTypeDef * hdma2d, uint32_t LayerIdx)

Function description

HAL_DMA2D_CLUTStartLoad

Function name

HAL_StatusTypeDef HAL_DMA2D_CLUTStartLoad (DMA2D_HandleTypeDef * hdma2d, const DMA2D_CLUTCfgTypeDef * CLUTCfg, uint32_t LayerIdx)

Function description

HAL_DMA2D_CLUTStartLoad_IT

Function name

HAL_StatusTypeDef HAL_DMA2D_CLUTStartLoad_IT (DMA2D_HandleTypeDef * hdma2d, const DMA2D_CLUTCfgTypeDef * CLUTCfg, uint32_t LayerIdx)

Function description

HAL_DMA2D_CLUTLoad

Function name

HAL_StatusTypeDef HAL_DMA2D_CLUTLoad (DMA2D_HandleTypeDef * hdma2d, DMA2D_CLUTCfgTypeDef CLUTCfg, uint32_t LayerIdx)

Function description

HAL_DMA2D_CLUTLoad_IT

Function name

HAL_StatusTypeDef HAL_DMA2D_CLUTLoad_IT (DMA2D_HandleTypeDef * hdma2d, DMA2D_CLUTCfgTypeDef CLUTCfg, uint32_t LayerIdx)

Function description

HAL_DMA2D_CLUTLoading_Abort

Function name

HAL_StatusTypeDef HAL_DMA2D_CLUTLoading_Abort (DMA2D_HandleTypeDef * hdma2d, uint32_t LayerIdx)

Function description

HAL_DMA2D_CLUTLoading_Suspend

Function name

HAL_StatusTypeDef HAL_DMA2D_CLUTLoading_Suspend (DMA2D_HandleTypeDef * hdma2d, uint32_t LayerIdx)

Function description

HAL_DMA2D_CLUTLoading_Resume

Function name

HAL_StatusTypeDef HAL_DMA2D_CLUTLoading_Resume (DMA2D_HandleTypeDef * hdma2d, uint32_t LayerIdx)

Function description

HAL_DMA2D_PollForTransfer

Function name

HAL_StatusTypeDef HAL_DMA2D_PollForTransfer (DMA2D_HandleTypeDef * hdma2d, uint32_t Timeout)

Function description

HAL_DMA2D_IRQHandler

Function name

void HAL_DMA2D_IRQHandler (DMA2D_HandleTypeDef * hdma2d)

Function description

HAL_DMA2D_LineEventCallback

Function name

void HAL_DMA2D_LineEventCallback (DMA2D_HandleTypeDef * hdma2d)

Function description

HAL_DMA2D_CLUTLoadingCpltCallback

Function name

void HAL_DMA2D_CLUTLoadingCpltCallback (DMA2D_HandleTypeDef * hdma2d)

Function description

HAL_DMA2D_ConfigLayer

Function name

HAL_StatusTypeDef HAL_DMA2D_ConfigLayer (DMA2D_HandleTypeDef * hdma2d, uint32_t LayerIdx)

Function description

HAL_DMA2D_ConfigCLUT

Function name

HAL_StatusTypeDef HAL_DMA2D_ConfigCLUT (DMA2D_HandleTypeDef * hdma2d, DMA2D_CLUTCfgTypeDef CLUTCfg, uint32_t LayerIdx)

Function description

HAL_DMA2D_ProgramLineEvent

Function name

HAL_StatusTypeDef HAL_DMA2D_ProgramLineEvent (DMA2D_HandleTypeDef * hdma2d, uint32_t Line)

Function description

HAL_DMA2D_EnableDeadTime

Function name

HAL_StatusTypeDef HAL_DMA2D_EnableDeadTime (DMA2D_HandleTypeDef * hdma2d)

Function description

HAL_DMA2D_DisableDeadTime

Function name

HAL_StatusTypeDef HAL_DMA2D_DisableDeadTime (DMA2D_HandleTypeDef * hdma2d)

Function description

HAL_DMA2D_ConfigDeadTime

Function name

HAL_StatusTypeDef HAL_DMA2D_ConfigDeadTime (DMA2D_HandleTypeDef * hdma2d, uint8_t DeadTime)

Function description

HAL_DMA2D_GetState

Function name

HAL_DMA2D_StateTypeDef HAL_DMA2D_GetState (const DMA2D_HandleTypeDef * hdma2d)

Function description

HAL_DMA2D_GetError

Function name

uint32_t HAL_DMA2D_GetError (const DMA2D_HandleTypeDef * hdma2d)

Function description

18.3 DMA2D Firmware driver defines

The following section lists the various define and macros of the module.

18.3.1 DMA2D

DMA2D

DMA2D Alpha Inversion

DMA2D_REGULAR_ALPHA

No modification of the alpha channel value

DMA2D_INVERTED_ALPHA

Invert the alpha channel value

DMA2D Alpha Mode

DMA2D_NO_MODIF_ALPHA

No modification of the alpha channel value

DMA2D_REPLACE_ALPHA

Replace original alpha channel value by programmed alpha value

DMA2D_COMBINE_ALPHA

Replace original alpha channel value by programmed alpha value with original alpha channel value

DMA2D Bytes Swap

DMA2D_BYTES_REGULAR

Bytes in regular order in output FIFO

DMA2D_BYTES_SWAP

Bytes are swapped two by two in output FIFO

DMA2D Chroma Sub Sampling

DMA2D_NO_CSS

No chroma sub-sampling 4:4:4

DMA2D_CSS_422

chroma sub-sampling 4:2:2

DMA2D_CSS_420

chroma sub-sampling 4:2:0

DMA2D CLUT Color Mode

DMA2D_CCM_ARGB8888

ARGB8888 DMA2D CLUT color mode

DMA2D_CCM_RGB888

RGB888 DMA2D CLUT color mode

DMA2D CLUT Size

DMA2D_CLUT_SIZE

DMA2D maximum CLUT size

DMA2D Color Value

DMA2D_COLOR_VALUE

Color value mask

DMA2D Error Code

HAL_DMA2D_ERROR_NONE

No error

HAL_DMA2D_ERROR_TE

Transfer error

HAL_DMA2D_ERROR_CE

Configuration error

HAL_DMA2D_ERROR_CAE

CLUT access error

HAL_DMA2D_ERROR_TIMEOUT

Timeout error

HAL_DMA2D_ERROR_INVALID_CALLBACK

Invalid callback error

DMA2D Exported Macros

__HAL_DMA2D_RESET_HANDLE_STATE

Description:

- Reset DMA2D handle state.

Parameters:

- __HANDLE__: specifies the DMA2D handle.

Return value:

- None

__HAL_DMA2D_ENABLE

Description:

- Enable the DMA2D.

Parameters:

- __HANDLE__: DMA2D handle

Return value:

- None.

__HAL_DMA2D_GET_FLAG

Description:

- Get the DMA2D pending flags.

Parameters:

- __HANDLE__: DMA2D handle
- __FLAG__: flag to check. This parameter can be any combination of the following values:
  - DMA2D_FLAG_CE: Configuration error flag
  - DMA2D_FLAG_CTC: CLUT transfer complete flag
  - DMA2D_FLAG_CAE: CLUT access error flag
  - DMA2D_FLAG_TW: Transfer Watermark flag
  - DMA2D_FLAG_TC: Transfer complete flag
  - DMA2D_FLAG_TE: Transfer error flag

Return value:

- The: state of FLAG.

__HAL_DMA2D_CLEAR_FLAG

Description:

- Clear the DMA2D pending flags.

Parameters:

- __HANDLE__: DMA2D handle
- __FLAG__: specifies the flag to clear. This parameter can be any combination of the following values:
  - DMA2D_FLAG_CE: Configuration error flag
  - DMA2D_FLAG_CTC: CLUT transfer complete flag
  - DMA2D_FLAG_CAE: CLUT access error flag
  - DMA2D_FLAG_TW: Transfer Watermark flag
  - DMA2D_FLAG_TC: Transfer complete flag
  - DMA2D_FLAG_TE: Transfer error flag

Return value:

- None

__HAL_DMA2D_ENABLE_IT

Description:

- Enable the specified DMA2D interrupts.

Parameters:

- __HANDLE__: DMA2D handle
- __INTERRUPT__: specifies the DMA2D interrupt sources to be enabled. This parameter can be any combination of the following values:
  - DMA2D_IT_CE: Configuration error interrupt mask
  - DMA2D_IT_CTC: CLUT transfer complete interrupt mask
  - DMA2D_IT_CAE: CLUT access error interrupt mask
  - DMA2D_IT_TW: Transfer Watermark interrupt mask
  - DMA2D_IT_TC: Transfer complete interrupt mask
  - DMA2D_IT_TE: Transfer error interrupt mask

Return value:

- None

__HAL_DMA2D_DISABLE_IT

Description:

- Disable the specified DMA2D interrupts.

Parameters:

- __HANDLE__: DMA2D handle
- __INTERRUPT__: specifies the DMA2D interrupt sources to be disabled. This parameter can be any combination of the following values:
  - DMA2D_IT_CE: Configuration error interrupt mask
  - DMA2D_IT_CTC: CLUT transfer complete interrupt mask
  - DMA2D_IT_CAE: CLUT access error interrupt mask
  - DMA2D_IT_TW: Transfer Watermark interrupt mask
  - DMA2D_IT_TC: Transfer complete interrupt mask
  - DMA2D_IT_TE: Transfer error interrupt mask

Return value:

- None

__HAL_DMA2D_GET_IT_SOURCE

Description:

- Check whether the specified DMA2D interrupt source is enabled or not.

Parameters:

- __HANDLE__: DMA2D handle
- __INTERRUPT__: specifies the DMA2D interrupt source to check. This parameter can be one of the following values:
  - DMA2D_IT_CE: Configuration error interrupt mask
  - DMA2D_IT_CTC: CLUT transfer complete interrupt mask
  - DMA2D_IT_CAE: CLUT access error interrupt mask
  - DMA2D_IT_TW: Transfer Watermark interrupt mask
  - DMA2D_IT_TC: Transfer complete interrupt mask
  - DMA2D_IT_TE: Transfer error interrupt mask

Return value:

- The: state of INTERRUPT source.

DMA2D Exported Types

MAX_DMA2D_LAYER

DMA2D maximum number of layers

DMA2D Flags

DMA2D_FLAG_CE

Configuration Error Interrupt Flag

DMA2D_FLAG_CTC

CLUT Transfer Complete Interrupt Flag

DMA2D_FLAG_CAE

CLUT Access Error Interrupt Flag

DMA2D_FLAG_TW

Transfer Watermark Interrupt Flag

DMA2D_FLAG_TC

Transfer Complete Interrupt Flag

DMA2D_FLAG_TE

Transfer Error Interrupt Flag

DMA2D Input Color Mode

DMA2D_INPUT_ARGB8888

ARGB8888 color mode

DMA2D_INPUT_RGB888

RGB888 color mode

DMA2D_INPUT_RGB565

RGB565 color mode

DMA2D_INPUT_ARGB1555

ARGB1555 color mode

DMA2D_INPUT_ARGB4444

ARGB4444 color mode

DMA2D_INPUT_L8

L8 color mode

DMA2D_INPUT_AL44

AL44 color mode

DMA2D_INPUT_AL88

AL88 color mode

DMA2D_INPUT_L4

L4 color mode

DMA2D_INPUT_A8

A8 color mode

DMA2D_INPUT_A4

A4 color mode

DMA2D_INPUT_YCBCR

YCbCr color mode

DMA2D Interrupts

DMA2D_IT_CE

Configuration Error Interrupt

DMA2D_IT_CTC

CLUT Transfer Complete Interrupt

DMA2D_IT_CAE

CLUT Access Error Interrupt

DMA2D_IT_TW

Transfer Watermark Interrupt

DMA2D_IT_TC

Transfer Complete Interrupt

DMA2D_IT_TE

Transfer Error Interrupt

DMA2D Layers

DMA2D_BACKGROUND_LAYER

DMA2D Background Layer (layer 0)

DMA2D_FOREGROUND_LAYER

DMA2D Foreground Layer (layer 1)

DMA2D Line Offset Mode

DMA2D_LOM_PIXELS

Line offsets expressed in pixels

DMA2D_LOM_BYTES

Line offsets expressed in bytes

DMA2D Maximum Line Watermark

DMA2D_LINE_WATERMARK_MAX

DMA2D maximum line watermark

DMA2D Maximum Number of Layers

DMA2D_MAX_LAYER

DMA2D maximum number of layers

DMA2D Mode

DMA2D_M2M

DMA2D memory to memory transfer mode

DMA2D_M2M_PFC

DMA2D memory to memory with pixel format conversion transfer mode

DMA2D_M2M_BLEND

DMA2D memory to memory with blending transfer mode

DMA2D_R2M

DMA2D register to memory transfer mode

DMA2D_M2M_BLEND_FG

DMA2D memory to memory with blending transfer mode and fixed color FG

DMA2D_M2M_BLEND_BG

DMA2D memory to memory with blending transfer mode and fixed color BG

DMA2D Offset

DMA2D_OFFSET

maximum Line Offset

DMA2D Output Color Mode

DMA2D_OUTPUT_ARGB8888

ARGB8888 DMA2D color mode

DMA2D_OUTPUT_RGB888

RGB888 DMA2D color mode

DMA2D_OUTPUT_RGB565

RGB565 DMA2D color mode

DMA2D_OUTPUT_ARGB1555

ARGB1555 DMA2D color mode

DMA2D_OUTPUT_ARGB4444

ARGB4444 DMA2D color mode

DMA2D Red and Blue Swap

DMA2D_RB_REGULAR

Select regular mode (RGB or ARGB)

DMA2D_RB_SWAP

Select swap mode (BGR or ABGR)

DMA2D Size

DMA2D_PIXEL

DMA2D maximum number of pixels per line

DMA2D_LINE

DMA2D maximum number of lines

19 HAL DMA Generic Driver

19.1 DMA Firmware driver registers structures

19.1.1 DMA_InitTypeDef

DMA_InitTypeDef is defined in the stm32n6xx_hal_dma.h

Data Fields

- ***uint32_t Request***
- ***uint32_t BlkHWRequest***
- ***uint32_t Direction***
- ***uint32_t SrcInc***
- ***uint32_t DestInc***
- ***uint32_t SrcDataWidth***
- ***uint32_t DestDataWidth***
- ***uint32_t Priority***
- ***uint32_t SrcBurstLength***
- ***uint32_t DestBurstLength***
- ***uint32_t TransferAllocatedPort***
- ***uint32_t TransferEventMode***
- ***uint32_t Mode***

Field Documentation

- ***uint32_t DMA_InitTypeDef::Request***
  Specifies the DMA channel request. This parameter can be a value of ***DMA_Request_Selection***
- ***uint32_t DMA_InitTypeDef::BlkHWRequest***
  Specifies the Block hardware request mode for DMA channel. Block Hardware request feature can be used only with dedicated peripherals. This parameter can be a value of ***DMA_Block_Request***
- ***uint32_t DMA_InitTypeDef::Direction***
  Specifies the transfer direction for DMA channel. This parameter can be a value of ***DMA_Transfer_Direction***
- ***uint32_t DMA_InitTypeDef::SrcInc***
  Specifies the source increment mode for the DMA channel. This parameter can be a value of ***DMA_Source_Increment_Mode***
- ***uint32_t DMA_InitTypeDef::DestInc***
  Specifies the destination increment mode for the DMA channel. This parameter can be a value of ***DMA_Destination_Increment_Mode***
- ***uint32_t DMA_InitTypeDef::SrcDataWidth***
  Specifies the source data width for the DMA channel. This parameter can be a value of ***DMA_Source_Data_Width***
- ***uint32_t DMA_InitTypeDef::DestDataWidth***
  Specifies the destination data width for the DMA channel. This parameter can be a value of ***DMA_Destination_Data_Width***
- ***uint32_t DMA_InitTypeDef::Priority***
  Specifies the priority level for the DMA channel. This parameter can be a value of ***DMA_Priority_Level***
- ***uint32_t DMA_InitTypeDef::SrcBurstLength***
  Specifies the source burst length (number of beats within a burst) for the DMA channel. This parameter can be a value between 1 and 64
- ***uint32_t DMA_InitTypeDef::DestBurstLength***
  Specifies the destination burst length (number of beats within a burst) for the DMA channel. This parameter can be a value between 1 and 64
- ***uint32_t DMA_InitTypeDef::TransferAllocatedPort***
  Specifies the transfer allocated ports. This parameter can be a combination of ***DMA_Transfer_Allocated_Port***

- ***uint32_t DMA_InitTypeDef::TransferEventMode***
  Specifies the transfer event mode for the DMA channel. This parameter can be a value of ***DMA_Transfer_Event_Mode***
- ***uint32_t DMA_InitTypeDef::Mode***
  Specifies the transfer mode for the DMA channel. This parameter can be a value of ***DMA_Transfer_Mode***

19.1.2 DMA_InitLinkedListTypeDef

DMA_InitLinkedListTypeDef is defined in the stm32n6xx_hal_dma.h

Data Fields

- ***uint32_t Priority***
- ***uint32_t LinkStepMode***
- ***uint32_t LinkAllocatedPort***
- ***uint32_t TransferEventMode***
- ***uint32_t LinkedListMode***

Field Documentation

- ***uint32_t DMA_InitLinkedListTypeDef::Priority***
  Specifies the priority level for the DMA channel. This parameter can be a value of ***DMA_Priority_Level***
- ***uint32_t DMA_InitLinkedListTypeDef::LinkStepMode***
  Specifies the link step mode for the DMA channel. This parameter can be a value of ***DMAEx_Link_Step_Mode***
- ***uint32_t DMA_InitLinkedListTypeDef::LinkAllocatedPort***
  Specifies the linked-list allocated port for the DMA channel. This parameter can be a value of ***DMAEx_Link_Allocated_Port***
- ***uint32_t DMA_InitLinkedListTypeDef::TransferEventMode***
  Specifies the transfer event mode for the DMA channel. This parameter can be a value of ***DMA_Transfer_Event_Mode***
- ***uint32_t DMA_InitLinkedListTypeDef::LinkedListMode***
  Specifies linked-list transfer mode for the DMA channel. This parameter can be a value of ***DMAEx_LinkedList_Mode***

19.1.3 DMA_IsolationConfigTypeDef

DMA_IsolationConfigTypeDef is defined in the stm32n6xx_hal_dma.h

Data Fields

- ***uint32_t CidFiltering***
- ***uint32_t StaticCid***

Field Documentation

- ***uint32_t DMA_IsolationConfigTypeDef::CidFiltering***
  Specified the CID filtering mode. This parameter can be a value of ***DMA_ISOLATION_MODE***
- ***uint32_t DMA_IsolationConfigTypeDef::StaticCid***
  Specified the static CID. This parameter can be a value of ***DMA_Channel_CID_Filtering_Static***

19.1.4 __DMA_HandleTypeDef

__DMA_HandleTypeDef is defined in the stm32n6xx_hal_dma.h

Data Fields

- ***DMA_Channel_TypeDef * Instance***
- ***DMA_InitTypeDef Init***
- ***DMA_InitLinkedListTypeDef InitLinkedList***
- ***HAL_LockTypeDef Lock***
- ***uint32_t Mode***
- ***__IO HAL_DMA_StateTypeDef State***
- ***__IO uint32_t ErrorCode***
- ***void * Parent***
- ***void(* XferCpltCallback***

- ***void(* XferHalfCpltCallback***
- ***void(* XferErrorCallback***
- ***void(* XferAbortCallback***
- ***void(* XferSuspendCallback***
- ***struct __DMA_QListTypeDef * LinkedListQueue***

Field Documentation

- ***DMA_Channel_TypeDef* __DMA_HandleTypeDef::Instance***
  Register the DMA channel base address
- ***DMA_InitTypeDef __DMA_HandleTypeDef::Init***
  DMA channel init parameters
- ***DMA_InitLinkedListTypeDef __DMA_HandleTypeDef::InitLinkedList***
  DMA channel linked-list init parameters
- ***HAL_LockTypeDef __DMA_HandleTypeDef::Lock***
  DMA locking object
- ***uint32_t __DMA_HandleTypeDef::Mode***
  DMA transfer mode
- ***__IO HAL_DMA_StateTypeDef __DMA_HandleTypeDef::State***
  DMA transfer state
- ***__IO uint32_t __DMA_HandleTypeDef::ErrorCode***
  DMA error code
- ***void* __DMA_HandleTypeDef::Parent***
  Parent object state
- ***void(* __DMA_HandleTypeDef::XferCpltCallback)(struct __DMA_HandleTypeDef *hdma)***
  DMA transfer complete callback
- ***void(* __DMA_HandleTypeDef::XferHalfCpltCallback)(struct __DMA_HandleTypeDef *hdma)***
  DMA half transfer complete callback
- ***void(* __DMA_HandleTypeDef::XferErrorCallback)(struct __DMA_HandleTypeDef *hdma)***
  DMA transfer error callback
- ***void(* __DMA_HandleTypeDef::XferAbortCallback)(struct __DMA_HandleTypeDef *hdma)***
  DMA transfer Abort callback
- ***void(* __DMA_HandleTypeDef::XferSuspendCallback)(struct __DMA_HandleTypeDef *hdma)***
  DMA transfer Suspend callback
- ***struct __DMA_QListTypeDef* __DMA_HandleTypeDef::LinkedListQueue***
  DMA linked-list queue

19.2 DMA Firmware driver API description

The following section lists the various functions of the DMA library.

19.2.1 How to use this driver

1. GPDMA and HPDMA are made available on this series. Transfer could be done thanks to AXI or AHB ports. HPDMA support transmission through: AXI port 0. AHB port 1. GPDMA allows transmission through: GPDMA_P = port-0. GPDMA_M = port-1 The supported configurations for GPDMA and HPDMA can be found in the Connectivity matrix table of the reference manual. Mind that DTCM and ITCM can be only be accessed thanks to HPDMA AXI port 0.

DMA transfer modes are divided to 2 major categories :

- Normal transfers (legacy)
- Linked-list transfers

Normal transfers mode is initialized via the standard module and linked-list mode is configured via the extended module.

Additionally to linked-list capability, all advanced DMA features are managed and configured via the extended module as extensions to normal mode. Advanced features are :

- Repeated block feature.
- Trigger feature.

- Data handling feature.

DMA Legacy circular transfer, is replaced by circular linked-list configuration.

Initialization and De-Initialization

For a given channel, enable and configure the peripheral to be connected to the DMA Channel (except for internal SRAM/FLASH memories: no initialization is necessary) please refer to Reference manual for connection between peripherals and DMA requests.

For a given channel, use HAL_DMA_Init function to program the required configuration for normal transfer through the following parameters:

- Request : Specifies the DMA channel request Request parameters :
  - can be a value of DMA_Request_Selection
- BlkHWRequest : Specifies the Block hardware request mode for DMA channel
  - can be a value of DMA_Block_Request
- Direction : Specifies the transfer direction for DMA channel
  - can be a value of DMA_Transfer_Direction
- SrcInc : Specifies the source increment mode for the DMA channel
  - can be a value of DMA_Source_Increment_Mode
- DestInc : Specifies the destination increment mode for the DMA channel
  - can be a value of DMA_Destination_Increment_Mode
- SrcDataWidth : Specifies the source data width for the DMA channel
  - can be a value of DMA_Source_Data_Width
- DestDataWidth : Specifies the destination data width for the DMA channel
  - can be a value of DMA_Destination_Data_Width
- Priority : Specifies the priority for the DMA channel
  - can be a value of DMA_Priority_Level
- SrcBurstLength : Specifies the source burst length (number of beats) for the DMA channel
  - can be a value of between 1 and 64
- DestBurstLength : Specifies the destination burst length (number of beats) for the DMA channel
  - can be a value of between 1 and 64
- TransferAllocatedPort : Specifies the source and destination allocated ports
  - can be a value of DMA_Transfer_Allocated_Port
- TransferEventMode : Specifies the transfer event mode for the DMA channel
  - can be a value of DMA_Transfer_Event_Mode
- Mode : Specifies the transfer mode for the DMA channel
  - can be one of the following modes :
    - DMA_NORMAL : Normal Mode
    - DMA_PFCTRL : Peripheral Flow Control (peripheral early termination) Mode

Polling mode IO operation

- Use HAL_DMA_Start() to start a DMA normal transfer after the configuration of source address, destination address and the size of data to be transferred.
- Use HAL_DMA_PollForTransfer() to poll for selected transfer level. In this case a fixed Timeout can be configured by User depending on his application. Transfer level can be :
  - HAL_DMA_HALF_TRANSFER
  - HAL_DMA_FULL_TRANSFER For circular transfer, this API returns an HAL_ERROR with HAL_DMA_ERROR_NOT_SUPPORTED error code.
- Use HAL_DMA_Abort() function to abort any ongoing DMA transfer in blocking mode. This API returns HAL_ERROR when there is no ongoing transfer or timeout is reached when disabling the DMA channel. (This API should not be called from an interrupt service routine)

Interrupt mode IO operation

- Configure the DMA interrupt priority using HAL_NVIC_SetPriority()
- Enable the DMA IRQ handler using HAL_NVIC_EnableIRQ()
- Use HAL_DMA_RegisterCallback() function to register user callbacks from the following list :
  - XferCpltCallback : transfer complete callback.
  - XferHalfCpltCallback : half transfer complete callback.
  - XferErrorCallback : transfer error callback.
  - XferAbortCallback : transfer abort complete callback.
  - XferSuspendCallback : transfer suspend complete callback.
- Use HAL_DMA_Start_IT() to start the DMA transfer after the enable of DMA interrupts and the configuration of source address,destination address and the size of data to be transferred.
- Use HAL_DMA_IRQHandler() called under DMA_IRQHandler() interrupt subroutine to handle any DMA interrupt.
- Use HAL_DMA_Abort_IT() function to abort any on-going DMA transfer in non-blocking mode. This API will suspend immediately the DMA channel execution. When the transfer is effectively suspended, an interrupt is generated and HAL_DMA_IRQHandler() will reset the channel and execute the callback XferAbortCallback. (This API could be called from an interrupt service routine)

State and errors

- Use HAL_DMA_GetState() function to get the DMA state.
- Use HAL_DMA_GetError() function to get the DMA error code.

Security and privilege attributes

- Use HAL_DMA_ConfigChannelAttributes() function to configure DMA channel security and privilege attributes.
  - Security : at channel level, at source level and at destination level.
  - Privilege : at channel level.
- Use HAL_DMA_GetConfigChannelAttributes() function to get the DMA channel attributes.
- Use HAL_DMA_LockChannelAttributes() function to lock the DMA channel security and privilege attributes configuration. This API can be called once after each system boot. If called again, HAL_DMA_ConfigChannelAttributes() API has no effect. Unlock is done either by a system boot or a by an RCC reset.
- Use HAL_DMA_GetLockChannelAttributes() function to get the attributes lock status.
- Use HAL_DMA_SetIsolationAttributes() function to configure the HPDMA channel isolation attribute.
- Use HAL_DMA_GetIsolationAttributes() function to get the HPDMA channel isolation attribute.

Isolation attributes

DMA HAL driver macros list

Below the list of most used macros in DMA HAL driver.

- __HAL_DMA_ENABLE : Enable the specified DMA Channel.
- __HAL_DMA_DISABLE : Disable the specified DMA Channel.
- __HAL_DMA_GET_FLAG : Get the DMA Channel pending flags.
- __HAL_DMA_CLEAR_FLAG : Clear the DMA Channel pending flags.
- __HAL_DMA_ENABLE_IT : Enable the specified DMA Channel interrupts.
- __HAL_DMA_DISABLE_IT : Disable the specified DMA Channel interrupts.
- __HAL_DMA_GET_IT_SOURCE : Check whether the specified DMA Channel interrupt has occurred or not.

Valid burst length

HPDMA allowed AXI maximum burst length

- The maximum allowed AXI burst length shall not exceed 16.
  - If either selected source or destination length is above 16, HAL_DMA_Init will return HAL_ERROR.

Note: You can refer to the header file of the DMA HAL driver for more useful macros.

19.2.2 Initialization and de-initialization functions

This section provides functions allowing to initialize and de-initialize the DMA channel in normal mode.

- The HAL_DMA_Init() function follows the DMA channel configuration procedures as described in reference manual.
- The HAL_DMA_DeInit() function allows to de-initialize the DMA channel.

This section contains the following APIs:

- ***HAL_DMA_Init()***
- ***HAL_DMA_DeInit()***

19.2.3 IO operation functions

This section provides functions allowing to :

- Configure the source, destination address and data size and Start DMA transfer in normal mode
- Abort DMA transfer
- Poll for transfer complete
- Handle DMA interrupt request
- Register and Unregister DMA callbacks
- The HAL_DMA_Start() function allows to start the DMA channel transfer in normal mode (Blocking mode).
- The HAL_DMA_Start_IT() function allows to start the DMA channel transfer in normal mode (Non-blocking mode).
- The HAL_DMA_Abort() function allows to abort any on-going transfer (Blocking mode).
- The HAL_DMA_Abort_IT() function allows to abort any on-going transfer (Non-blocking mode).
- The HAL_DMA_PollForTransfer() function allows to poll on half transfer and transfer complete (Blocking mode). This API cannot be used for circular transfers.
- The HAL_DMA_IRQHandler() function allows to handle any DMA channel interrupt (Non-blocking mode).
- The HAL_DMA_RegisterCallback() and HAL_DMA_UnRegisterCallback() functions allow respectively to register and unregister user customized callbacks. User callbacks are called under HAL_DMA_IRQHandler().

This section contains the following APIs:

- ***HAL_DMA_Start()***
- ***HAL_DMA_Start_IT()***
- ***HAL_DMA_Abort()***
- ***HAL_DMA_Abort_IT()***
- ***HAL_DMA_PollForTransfer()***
- ***HAL_DMA_IRQHandler()***
- ***HAL_DMA_RegisterCallback()***
- ***HAL_DMA_UnRegisterCallback()***

19.2.4 State and Errors functions

This section provides functions allowing to :

- Check the DMA state
- Get error code
- The HAL_DMA_GetState() function allows to get the DMA channel state.
- The HAL_DMA_DeInit() function allows to get the DMA channel error code.

This section contains the following APIs:

- ***HAL_DMA_GetState()***
- ***HAL_DMA_GetError()***

19.2.5 DMA Attributes functions

This section provides functions allowing to :

- Configure DMA channel secure and privilege attributes.
- Get DMA channel secure and privilege attributes.
- Lock DMA channel secure and privilege attributes configuration.
- Check whether DMA channel secure and privilege attributes configuration is locked or not.
- The HAL_DMA_ConfigChannelAttributes() function allows to configure DMA channel security and privilege attributes.
- The HAL_DMA_GetConfigChannelAttributes() function allows to get DMA channel security and privilege attributes configuration.
- The HAL_DMA_LockChannelAttributes() function allows to lock the DMA channel security and privilege attributes.
- The HAL_DMA_GetLockChannelAttributes() function allows to get the DMA channel security and privilege attributes lock status.

This section contains the following APIs:

- ***HAL_DMA_ConfigChannelAttributes()***
- ***HAL_DMA_GetConfigChannelAttributes()***
- ***HAL_DMA_SetIsolationAttributes()***
- ***HAL_DMA_GetIsolationAttributes()***
- ***HAL_DMA_LockChannelAttributes()***
- ***HAL_DMA_GetLockChannelAttributes()***

19.2.6 Detailed description of functions

HAL_DMA_Init

Function name

HAL_StatusTypeDef HAL_DMA_Init (DMA_HandleTypeDef *const hdma)

Function description

Initialize the DMA channel in normal mode according to the specified parameters in the DMA_InitTypeDef and create the associated handle.

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.

Return values

- **HAL:** status.

Notes

- Warning: the maximum allowed AXI burst length shall not exceed 16. Otherwise, an error will be returned and no initialization performed.

HAL_DMA_DeInit

Function name

HAL_StatusTypeDef HAL_DMA_DeInit (DMA_HandleTypeDef *const hdma)

Function description

DeInitialize the DMA channel when it is configured in normal mode.

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.

Return values

- **HAL:** status.

HAL_DMA_Start

Function name

HAL_StatusTypeDef HAL_DMA_Start (DMA_HandleTypeDef *const hdma, uint32_t SrcAddress, uint32_t DstAddress, uint32_t SrcDataSize)

Function description

Start the DMA channel transfer in normal mode (Blocking mode).

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
- **SrcAddress:** : The source data address.
- **DstAddress:** : The destination data address.
- **SrcDataSize:** : The length of data to be transferred from source to destination in bytes.

Return values

- **HAL:** status.

HAL_DMA_Start_IT

Function name

HAL_StatusTypeDef HAL_DMA_Start_IT (DMA_HandleTypeDef *const hdma, uint32_t SrcAddress, uint32_t DstAddress, uint32_t SrcDataSize)

Function description

Starts the DMA channel transfer in normal mode with interrupts enabled (Non-blocking mode).

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
- **SrcAddress:** : The source data address.
- **DstAddress:** : The destination data address.
- **SrcDataSize:** : The length of data to be transferred from source to destination in bytes.

Return values

- **HAL:** status.

HAL_DMA_Abort

Function name

HAL_StatusTypeDef HAL_DMA_Abort (DMA_HandleTypeDef *const hdma)

Function description

Abort any on-going DMA channel transfer (Blocking mode).

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.

Return values

- **HAL:** status.

Notes

- After suspending a DMA channel, a wait until the DMA channel is effectively stopped is added. If a channel is suspended while a data transfer is on-going, the current data will be transferred and the channel will be effectively suspended only after the transfer of any on-going data is finished.

HAL_DMA_Abort_IT

Function name

HAL_StatusTypeDef HAL_DMA_Abort_IT (DMA_HandleTypeDef *const hdma)

Function description

Abort any on-going DMA channel transfer in interrupt mode (Non-blocking mode).

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.

Return values

- **HAL:** status.

HAL_DMA_PollForTransfer

Function name

HAL_StatusTypeDef HAL_DMA_PollForTransfer (DMA_HandleTypeDef *const hdma, HAL_DMA_LevelCompleteTypeDef CompleteLevel, uint32_t Timeout)

Function description

Polling for transfer status (Blocking mode).

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
- **CompleteLevel:** : Specifies the DMA level complete.
- **Timeout:** : Timeout duration.

Return values

- **HAL:** status

HAL_DMA_IRQHandler

Function name

void HAL_DMA_IRQHandler (DMA_HandleTypeDef *const hdma)

Function description

Handle DMA interrupt request (Non-blocking mode).

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.

Return values

- **None.:**

HAL_DMA_RegisterCallback

Function name

HAL_StatusTypeDef HAL_DMA_RegisterCallback (DMA_HandleTypeDef *const hdma, HAL_DMA_CallbackIDTypeDef CallbackID, void(*)(DMA_HandleTypeDef *const _hdma) pCallback)

Function description

Register callback according to specified ID.

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
- **CallbackID:** : User Callback identifier which could be a value of HAL_DMA_CallbackIDTypeDef enumeration.
- **pCallback:** : Pointer to private callback function.

Return values

- **HAL:** status.

Notes

- The HAL_DMA_RegisterCallback() may be called before HAL_DMA_Init() in HAL_DMA_STATE_RESET to register callbacks for HAL_DMA_MSPINIT_CB_ID and HAL_DMA_MSPDEINIT_CB_ID.

HAL_DMA_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_DMA_UnRegisterCallback (DMA_HandleTypeDef *const hdma, HAL_DMA_CallbackIDTypeDef CallbackID)

Function description

Unregister callback according to specified ID.

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
- **CallbackID:** : User Callback identifier which could be a value of HAL_DMA_CallbackIDTypeDef enum.

Return values

- **HAL:** status.

Notes

- The HAL_DMA_UnRegisterCallback() may be called before HAL_DMA_Init() in HAL_DMA_STATE_RESET to un-register callbacks for HAL_DMA_MSPINIT_CB_ID and HAL_DMA_MSPDEINIT_CB_ID.

HAL_DMA_GetState

Function name

HAL_DMA_StateTypeDef HAL_DMA_GetState (DMA_HandleTypeDef const *const hdma)

Function description

Returns the DMA channel state.

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.

Return values

- **DMA:** state.

HAL_DMA_GetError

Function name

uint32_t HAL_DMA_GetError (DMA_HandleTypeDef const *const hdma)

Function description

Return the DMA channel error code.

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.

Return values

- **DMA:** Error Code.

HAL_DMA_ConfigChannelAttributes

Function name

HAL_StatusTypeDef HAL_DMA_ConfigChannelAttributes (DMA_HandleTypeDef *const hdma, uint32_t ChannelAttributes)

Function description

Configure the DMA channel security and privilege attribute(s).

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
- **ChannelAttributes:** : Specifies the DMA channel secure/privilege attributes. This parameter can be a one or a combination of DMA Channel Attributes.

Return values

- **HAL:** Status.

Notes

- These attributes cannot be modified when the corresponding lock state is enabled.

HAL_DMA_GetConfigChannelAttributes

Function name

HAL_StatusTypeDef HAL_DMA_GetConfigChannelAttributes (DMA_HandleTypeDef const *const hdma, uint32_t *const pChannelAttributes)

Function description

Get the DMA channel security and privilege attributes.

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
- **pChannelAttributes:** : Pointer to the returned attributes.

Return values

- **HAL:** Status.

HAL_DMA_LockChannelAttributes

Function name

HAL_StatusTypeDef HAL_DMA_LockChannelAttributes (DMA_HandleTypeDef const *const hdma)

Function description

Lock the DMA channel security and privilege attribute(s).

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.

Return values

- **HAL:** Status.

HAL_DMA_GetLockChannelAttributes

Function name

HAL_StatusTypeDef HAL_DMA_GetLockChannelAttributes (DMA_HandleTypeDef const *const hdma, uint32_t *const pLockState)

Function description

Get the security and privilege attribute lock state of a DMA channel.

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
- **pLockState:** : Pointer to lock state (returned value can be DMA_CHANNEL_ATTRIBUTE_UNLOCKED or DMA_CHANNEL_ATTRIBUTE_LOCKED).

Return values

- **HAL:** status.

HAL_DMA_SetIsolationAttributes

Function name

HAL_StatusTypeDef HAL_DMA_SetIsolationAttributes (DMA_HandleTypeDef *const hdma, DMA_IsolationConfigTypeDef const *const pConfig)

Function description

Set the DMA channel filtering CID (Isolation configuration).

HAL_DMA_GetIsolationAttributes

Function name

HAL_StatusTypeDef HAL_DMA_GetIsolationAttributes (DMA_HandleTypeDef const *const hdma, DMA_IsolationConfigTypeDef *const pConfig)

Function description

Read the DMA channel filtering CID (Isolation configuration).

19.3 DMA Firmware driver defines

The following section lists the various define and macros of the module.

19.3.1 DMA

DMA

DMA Block Request

DMA_BREQ_SINGLE_BURST

Hardware request protocol at a single / burst level

DMA_BREQ_BLOCK

Hardware request protocol at a block level

DMA Channel Attributes

DMA_CHANNEL_PRIV

Channel is privileged

DMA_CHANNEL_NPRIV

Channel is unprivileged

DMA_CHANNEL_SEC

Channel is secure

DMA_CHANNEL_NSEC

Channel is non-secure

DMA_CHANNEL_SRC_SEC

Channel source is secure

DMA_CHANNEL_SRC_NSEC

Channel source is non-secure

DMA_CHANNEL_DEST_SEC

Channel destination is secure

DMA_CHANNEL_DEST_NSEC

Channel destination is non-secure

DMA_CHANNEL_ATTRIBUTE_UNLOCKED

Channel attribute is unlocked

DMA_CHANNEL_ATTRIBUTE_LOCKED

Channel attribute is locked

Static DMA Channel Filtering CID (isolation)

DMA_CHANNEL_STATIC_CID_0

Channel control to CID0 in static mode

DMA_CHANNEL_STATIC_CID_1

Channel control to CID1 in static mode

DMA_CHANNEL_STATIC_CID_2

Channel control to CID2 in static mode

DMA_CHANNEL_STATIC_CID_3

Channel control to CID3 in static mode

DMA_CHANNEL_STATIC_CID_4

Channel control to CID4 in static mode

DMA_CHANNEL_STATIC_CID_5

Channel control to CID5 in static mode

DMA_CHANNEL_STATIC_CID_6

Channel control to CID6 in static mode

DMA destination Data Width

DMA_DEST_DATAWIDTH_BYTE

Destination data width : Byte

DMA_DEST_DATAWIDTH_HALFWORD

Destination data width : HalfWord

DMA_DEST_DATAWIDTH_WORD

Destination data width : Word

DMA_DEST_DATAWIDTH_DOUBLEWORD

Destination data width : DoubleWord

DMA Destination Increment Mode

DMA_DINC_FIXED

Destination fixed single / burst

DMA_DINC_INCREMENTED

Destination incremented single / burst

DMA Error Codes

HAL_DMA_ERROR_NONE

No error

HAL_DMA_ERROR_DTE

Data transfer error

HAL_DMA_ERROR_ULE

Update linked-list item error

HAL_DMA_ERROR_USE

User setting error

HAL_DMA_ERROR_TO

Trigger overrun error

HAL_DMA_ERROR_TIMEOUT

Timeout error

HAL_DMA_ERROR_NO_XFER

No transfer ongoing error

HAL_DMA_ERROR_BUSY

Busy error

HAL_DMA_ERROR_INVALID_CALLBACK

Invalid callback error

HAL_DMA_ERROR_NOT_SUPPORTED

Not supported mode

DMA Exported Macros

__HAL_DMA_RESET_HANDLE_STATE

Description:

- Reset DMA handle state.

Parameters:

- __HANDLE__: : DMA handle.

Return value:

- None.

__HAL_DMA_ENABLE

Description:

- Enable the specified DMA Channel.

Parameters:

- __HANDLE__: : DMA handle.

Return value:

- None

__HAL_DMA_DISABLE

Description:

- Disable the specified DMA Channel.

Parameters:

- __HANDLE__: : DMA handle.

Return value:

- None

__HAL_DMA_GET_FLAG

Description:

- Get the DMA channel pending flags.

Parameters:

- __HANDLE__: : DMA handle.
- __FLAG__: : Get the specified flag. This parameter can be any combination of the following values:
  - DMA_FLAG_TC : Transfer Complete flag.
  - DMA_FLAG_HT : Half Transfer Complete flag.
  - DMA_FLAG_DTE : Data Transfer Error flag.
  - DMA_FLAG_ULE : Update linked-list Error flag.
  - DMA_FLAG_USE : User Setting Error flag.
  - DMA_FLAG_TO : Trigger Overrun flag.
  - DMA_FLAG_SUSP : Completed Suspension flag.
  - DMA_FLAG_IDLEF : Idle flag.

Return value:

- The: state of FLAG (SET or RESET).

__HAL_DMA_CLEAR_FLAG

Description:

- Clear the DMA Channel pending flags.

Parameters:

- __HANDLE__: : DMA handle.
- __FLAG__: : Specifies the flag to clear. This parameter can be any combination of the following values:
  - DMA_FLAG_TC : Transfer Complete flag.
  - DMA_FLAG_HT : Half Transfer Complete flag.
  - DMA_FLAG_DTE : Data Transfer Error flag.
  - DMA_FLAG_ULE : Update Linked-List Error flag.
  - DMA_FLAG_USE : User Setting Error flag.
  - DMA_FLAG_TO : Trigger Overrun flag.
  - DMA_FLAG_SUSP : Completed Suspension flag.

Return value:

- None

__HAL_DMA_ENABLE_IT

Description:

- Enable the specified DMA Channel interrupts.

Parameters:

- __HANDLE__: : DMA handle.
- __INTERRUPT__: : Specifies the DMA interrupt sources to be enabled. This parameter can be any combination of the following values:
  - DMA_IT_TC : Transfer Complete interrupt.
  - DMA_IT_HT : Half Transfer Complete interrupt.
  - DMA_IT_DTE : Data Transfer Error interrupt.
  - DMA_IT_ULE : Update Linked-List Error interrupt.
  - DMA_IT_USE : User Setting Error interrupt.
  - DMA_IT_TO : Trigger Overrun interrupt.
  - DMA_IT_SUSP : Completed Suspension interrupt.

Return value:

- None

__HAL_DMA_DISABLE_IT

Description:

- Disable the specified DMA Channel interrupts.

Parameters:

- __HANDLE__: : DMA handle.
- __INTERRUPT__: : specifies the DMA interrupt sources to be disabled. This parameter can be any combination of the following values:
  - DMA_IT_TC : Transfer Complete interrupt.
  - DMA_IT_HT : Half Transfer Complete interrupt.
  - DMA_IT_DTE : Data Transfer Error interrupt.
  - DMA_IT_ULE : Update Linked-List Error interrupt.
  - DMA_IT_USE : User Setting Error interrupt.
  - DMA_IT_TO : Trigger Overrun interrupt.
  - DMA_IT_SUSP : Completed Suspension interrupt.

Return value:

- None

__HAL_DMA_GET_IT_SOURCE

Description:

- Checks whether the specified DMA Channel interrupt is enabled or not.

Parameters:

- __HANDLE__: : DMA handle.
- __INTERRUPT__: : specifies the DMA interrupt source to check.
  - DMA_IT_TC : Transfer Complete interrupt.
  - DMA_IT_HT : Half Transfer Complete interrupt.
  - DMA_IT_DTE : Data Transfer Error interrupt.
  - DMA_IT_ULE : Update Linked-List Error interrupt.
  - DMA_IT_USE : User Setting Error interrupt.
  - DMA_IT_TO : Trigger Overrun interrupt.
  - DMA_IT_SUSP : Completed Suspension interrupt.

Return value:

- The: state of DMA_IT (SET or RESET).

__HAL_DMA_SET_COUNTER

Description:

- Writes the block number of bytes to be transferred from the source on the DMA Channel.

Parameters:

- __HANDLE__: : DMA handle.
- __COUNTER__: : Number of data bytes to be transferred from the source (from 0 to 65535).

__HAL_DMA_GET_COUNTER

Description:

- Returns the number of remaining data bytes in the current DMA Channel transfer.

Parameters:

- __HANDLE__: : DMA handle.

Return value:

- The: number of remaining data units in the current DMA Stream transfer.

DMA Flag Definition

DMA_FLAG_IDLE

Idle flag

DMA_FLAG_TC

Transfer complete flag

DMA_FLAG_HT

Half transfer complete flag

DMA_FLAG_DTE

Data transfer error flag

DMA_FLAG_ULE

Update linked-list item error flag

DMA_FLAG_USE

User setting error flag

DMA_FLAG_SUSP

Completed suspension flag

DMA_FLAG_TO

Trigger overrun flag

TIM DMA Handle Index

TIM_DMA_ID_UPDATE

Index of the DMA handle used for Update DMA requests

TIM_DMA_ID_CC1

Index of the DMA handle used for Capture/Compare 1 DMA requests

TIM_DMA_ID_CC2

Index of the DMA handle used for Capture/Compare 2 DMA requests

TIM_DMA_ID_CC3

Index of the DMA handle used for Capture/Compare 3 DMA requests

TIM_DMA_ID_CC4

Index of the DMA handle used for Capture/Compare 4 DMA requests

TIM_DMA_ID_COMMUTATION

Index of the DMA handle used for Commutation DMA requests

TIM_DMA_ID_TRIGGER

Index of the DMA handle used for Trigger DMA requests

DMA Interrupt Enable Definition

DMA_IT_TC

Transfer complete interrupt

DMA_IT_HT

Half transfer complete interrupt

DMA_IT_DTE

Data transfer error interrupt

DMA_IT_ULE

Update linked-list item error interrupt

DMA_IT_USE

User eetting error interrupt

DMA_IT_SUSP

Completed suspension interrupt

DMA_IT_TO

Trigger overrun interrupt

DMA Channel isolation mode (CID filtering)

DMA_ISOLATION_OFF

DMA Channel isolation (CID filtering) is disabled

DMA_ISOLATION_ON

DMA Channel isolation (CID filtering) is enabled

DMA Priority Level

DMA_LOW_PRIORITY_LOW_WEIGHT

Priority level : Low Priority, Low weight

DMA_LOW_PRIORITY_MID_WEIGHT

Priority level : Low Priority, Mid weight

DMA_LOW_PRIORITY_HIGH_WEIGHT

Priority level : Low Priority, High weight

DMA_HIGH_PRIORITY

Priority level : HIGH Priority

DMA Request Selection

HPDMA1_REQUEST_JPEG_RX

HPDMA1 HW request is JPEG_RX

HPDMA1_REQUEST_JPEG_TX

HPDMA1 HW request is JPEG_TX

HPDMA1_REQUEST_XSPI1

HPDMA1 HW request is XSPI1

HPDMA1_REQUEST_XSPI2

HPDMA1 HW request is XSPI2

HPDMA1_REQUEST_XSPI3

HPDMA1 HW request is XSPI3

HPDMA1_REQUEST_FMC2_TXRX

HPDMA1 HW request is FMC2_TXRX

HPDMA1_REQUEST_FMC2_BCH

HPDMA1 HW request is FMC2_BCH

HPDMA1_REQUEST_ADC1

HPDMA1 HW request is ADC1

HPDMA1_REQUEST_ADC2

HPDMA1 HW request is ADC2

HPDMA1_REQUEST_CRYP_IN

HPDMA1 HW request is CRYP_IN

HPDMA1_REQUEST_CRYP_OUT

HPDMA1 HW request is CRYP_OUT

HPDMA1_REQUEST_SAES_OUT

HPDMA1 HW request is SAES_OUT

HPDMA1_REQUEST_SAES_IN

HPDMA1 HW request is SAES_IN

HPDMA1_REQUEST_HASH_IN

HPDMA1 HW request is HASH_IN

HPDMA1_REQUEST_TIM1_CH1

HPDMA1 HW request is TIM1_CH1

HPDMA1_REQUEST_TIM1_CH2

HPDMA1 HW request is TIM1_CH2

HPDMA1_REQUEST_TIM1_CH3

HPDMA1 HW request is TIM1_CH3

HPDMA1_REQUEST_TIM1_CH4

HPDMA1 HW request is TIM1_CH4

HPDMA1_REQUEST_TIM1_UP

HPDMA1 HW request is TIM1_UP

HPDMA1_REQUEST_TIM1_TRIG

HPDMA1 HW request is TIM1_TRIG

HPDMA1_REQUEST_TIM1_COM

HPDMA1 HW request is TIM1_COM

HPDMA1_REQUEST_TIM2_CH1

HPDMA1 HW request is TIM2_CH1

HPDMA1_REQUEST_TIM2_CH2

HPDMA1 HW request is TIM2_CH2

HPDMA1_REQUEST_TIM2_CH3

HPDMA1 HW request is TIM2_CH3

HPDMA1_REQUEST_TIM2_CH4

HPDMA1 HW request is TIM2_CH4

HPDMA1_REQUEST_TIM2_UP

HPDMA1 HW request is TIM2_UP

HPDMA1_REQUEST_TIM2_TRIG

HPDMA1 HW request is TIM2_TRIG

HPDMA1_REQUEST_TIM3_CH1

HPDMA1 HW request is TIM3_CH1

HPDMA1_REQUEST_TIM3_CH2

HPDMA1 HW request is TIM3_CH2

HPDMA1_REQUEST_TIM3_CH3

HPDMA1 HW request is TIM3_CH3

HPDMA1_REQUEST_TIM3_CH4

HPDMA1 HW request is TIM3_CH4

HPDMA1_REQUEST_TIM3_UP

HPDMA1 HW request is TIM3_UP

HPDMA1_REQUEST_TIM3_TRIG

HPDMA1 HW request is TIM3_TRIG

HPDMA1_REQUEST_TIM4_CH1

HPDMA1 HW request is TIM4_CH1

HPDMA1_REQUEST_TIM4_CH2

HPDMA1 HW request is TIM4_CH2

HPDMA1_REQUEST_TIM4_CH3

HPDMA1 HW request is TIM4_CH3

HPDMA1_REQUEST_TIM4_CH4

HPDMA1 HW request is TIM4_CH4

HPDMA1_REQUEST_TIM4_UP

HPDMA1 HW request is TIM4_UP

HPDMA1_REQUEST_TIM4_TRIG

HPDMA1 HW request is TIM4_TRIG

HPDMA1_REQUEST_TIM5_CH1

HPDMA1 HW request is TIM5_CH1

HPDMA1_REQUEST_TIM5_CH2

HPDMA1 HW request is TIM5_CH2

HPDMA1_REQUEST_TIM5_CH3

HPDMA1 HW request is TIM5_CH3

HPDMA1_REQUEST_TIM5_CH4

HPDMA1 HW request is TIM5_CH4

HPDMA1_REQUEST_TIM5_UP

HPDMA1 HW request is TIM5_UP

HPDMA1_REQUEST_TIM5_TRIG

HPDMA1 HW request is TIM5_TRIG

HPDMA1_REQUEST_TIM6_UP

HPDMA1 HW request is TIM6_UP

HPDMA1_REQUEST_TIM7_UP

HPDMA1 HW request is TIM7_UP

HPDMA1_REQUEST_TIM8_CH1

HPDMA1 HW request is TIM8_CH1

HPDMA1_REQUEST_TIM8_CH2

HPDMA1 HW request is TIM8_CH2

HPDMA1_REQUEST_TIM8_CH3

HPDMA1 HW request is TIM8_CH3

HPDMA1_REQUEST_TIM8_CH4

HPDMA1 HW request is TIM8_CH4

HPDMA1_REQUEST_TIM8_UP

HPDMA1 HW request is TIM8_UP

HPDMA1_REQUEST_TIM8_TRIG

HPDMA1 HW request is TIM8_TRIG

HPDMA1_REQUEST_TIM8_COM

HPDMA1 HW request is TIM8_COM

HPDMA1_REQUEST_TIM15_CH1

HPDMA1 HW request is TIM15_CH1

HPDMA1_REQUEST_TIM15_CH2

HPDMA1 HW request is TIM15_CH2

HPDMA1_REQUEST_TIM15_UP

HPDMA1 HW request is TIM15_UP

HPDMA1_REQUEST_TIM15_TRIG

HPDMA1 HW request is TIM15_TRIG

HPDMA1_REQUEST_TIM15_COM

HPDMA1 HW request is TIM15_COM

HPDMA1_REQUEST_TIM16_CH1

HPDMA1 HW request is TIM16_CH1

HPDMA1_REQUEST_TIM16_UP

HPDMA1 HW request is TIM16_UP

HPDMA1_REQUEST_TIM16_COM

HPDMA1 HW request is TIM16_COM

HPDMA1_REQUEST_TIM17_CH1

HPDMA1 HW request is TIM17_CH1

HPDMA1_REQUEST_TIM17_UP

HPDMA1 HW request is TIM17_UP

HPDMA1_REQUEST_TIM17_COM

HPDMA1 HW request is TIM17_COM

HPDMA1_REQUEST_TIM18_CH1

HPDMA1 HW request is TIM18_CH1

HPDMA1_REQUEST_TIM18_UP

HPDMA1 HW request is TIM18_UP

HPDMA1_REQUEST_TIM18_COM

HPDMA1 HW request is TIM18_COM

HPDMA1_REQUEST_LPTIM1_IC1

HPDMA1 HW request is LPTIM1_IC1

HPDMA1_REQUEST_LPTIM1_IC2

HPDMA1 HW request is LPTIM1_IC2

HPDMA1_REQUEST_LPTIM1_UE

HPDMA1 HW request is LPTIM1_UE

HPDMA1_REQUEST_LPTIM2_IC1

HPDMA1 HW request is LPTIM2_IC1

HPDMA1_REQUEST_LPTIM2_IC2

HPDMA1 HW request is LPTIM2_IC2

HPDMA1_REQUEST_LPTIM2_UE

HPDMA1 HW request is LPTIM2_UE

HPDMA1_REQUEST_LPTIM3_IC1

HPDMA1 HW request is LPTIM3_IC1

HPDMA1_REQUEST_LPTIM3_IC2

HPDMA1 HW request is LPTIM3_IC2

HPDMA1_REQUEST_LPTIM3_UE

HPDMA1 HW request is LPTIM3_UE

HPDMA1_REQUEST_SPI1_RX

HPDMA1 HW request is SPI1_RX

HPDMA1_REQUEST_SPI1_TX

HPDMA1 HW request is SPI1_TX

HPDMA1_REQUEST_SPI2_RX

HPDMA1 HW request is SPI2_RX

HPDMA1_REQUEST_SPI2_TX

HPDMA1 HW request is SPI2_TX

HPDMA1_REQUEST_SPI3_RX

HPDMA1 HW request is SPI3_RX

HPDMA1_REQUEST_SPI3_TX

HPDMA1 HW request is SPI3_TX

HPDMA1_REQUEST_SPI4_RX

HPDMA1 HW request is SPI4_RX

HPDMA1_REQUEST_SPI4_TX

HPDMA1 HW request is SPI4_TX

HPDMA1_REQUEST_SPI5_RX

HPDMA1 HW request is SPI5_RX

HPDMA1_REQUEST_SPI5_TX

HPDMA1 HW request is SPI5_TX

HPDMA1_REQUEST_SPI6_RX

HPDMA1 HW request is SPI6_RX

HPDMA1_REQUEST_SPI6_TX

HPDMA1 HW request is SPI6_TX

HPDMA1_REQUEST_SAI1_A

HPDMA1 HW request is SAI1_A

HPDMA1_REQUEST_SAI1_B

HPDMA1 HW request is SAI1_B

HPDMA1_REQUEST_SAI2_A

HPDMA1 HW request is SAI2_A

HPDMA1_REQUEST_SAI2_B

HPDMA1 HW request is SAI2_B

HPDMA1_REQUEST_I2C1_RX

HPDMA1 HW request is I2C1_RX

HPDMA1_REQUEST_I2C1_TX

HPDMA1 HW request is I2C1_TX

HPDMA1_REQUEST_I2C2_RX

HPDMA1 HW request is I2C2_RX

HPDMA1_REQUEST_I2C2_TX

HPDMA1 HW request is I2C2_TX

HPDMA1_REQUEST_I2C3_RX

HPDMA1 HW request is I2C3_RX

HPDMA1_REQUEST_I2C3_TX

HPDMA1 HW request is I2C3_TX

HPDMA1_REQUEST_I2C4_RX

HPDMA1 HW request is I2C4_RX

HPDMA1_REQUEST_I2C4_TX

HPDMA1 HW request is I2C4_TX

HPDMA1_REQUEST_I3C1_RX

HPDMA1 HW request is I3C1_RX

HPDMA1_REQUEST_I3C1_TX

HPDMA1 HW request is I3C1_TX

HPDMA1_REQUEST_I3C2_RX

HPDMA1 HW request is I3C2_RX

HPDMA1_REQUEST_I3C2_TX

HPDMA1 HW request is I3C2_TX

HPDMA1_REQUEST_USART1_RX

HPDMA1 HW request is USART1_RX

HPDMA1_REQUEST_USART1_TX

HPDMA1 HW request is USART1_TX

HPDMA1_REQUEST_USART2_RX

HPDMA1 HW request is USART2_RX

HPDMA1_REQUEST_USART2_TX

HPDMA1 HW request is USART2_TX

HPDMA1_REQUEST_USART3_RX

HPDMA1 HW request is USART3_RX

HPDMA1_REQUEST_USART3_TX

HPDMA1 HW request is USART3_TX

HPDMA1_REQUEST_UART4_RX

HPDMA1 HW request is UART4_RX

HPDMA1_REQUEST_UART4_TX

HPDMA1 HW request is UART4_TX

HPDMA1_REQUEST_UART5_RX

HPDMA1 HW request is UART5_RX

HPDMA1_REQUEST_UART5_TX

HPDMA1 HW request is UART5_TX

HPDMA1_REQUEST_USART6_RX

HPDMA1 HW request is USART6_RX

HPDMA1_REQUEST_USART6_TX

HPDMA1 HW request is USART6_TX

HPDMA1_REQUEST_UART7_RX

HPDMA1 HW request is UART7_RX

HPDMA1_REQUEST_UART7_TX

HPDMA1 HW request is UART7_TX

HPDMA1_REQUEST_UART8_RX

HPDMA1 HW request is UART8_RX

HPDMA1_REQUEST_UART8_TX

HPDMA1 HW request is UART8_TX

HPDMA1_REQUEST_UART9_RX

HPDMA1 HW request is UART9_RX

HPDMA1_REQUEST_UART9_TX

HPDMA1 HW request is UART9_TX

HPDMA1_REQUEST_USART10_RX

HPDMA1 HW request is USART10_RX

HPDMA1_REQUEST_USART10_TX

HPDMA1 HW request is USART10_TX

HPDMA1_REQUEST_LPUART1_RX

HPDMA1 HW request is LPUART1_RX

HPDMA1_REQUEST_LPUART1_TX

HPDMA1 HW request is LPUART1_TX

HPDMA1_REQUEST_SPDIFRX_CS

HPDMA1 HW request is SPDIFRX_CS

HPDMA1_REQUEST_SPDIFRX_DT

HPDMA1 HW request is SPDIFRX_DT

HPDMA1_REQUEST_ADF1_FLT0

HPDMA1 HW request is ADF1_FLT0

HPDMA1_REQUEST_MDF1_FLT0

HPDMA1 HW request is MDF1_FLT0

HPDMA1_REQUEST_MDF1_FLT1

HPDMA1 HW request is MDF1_FLT1

HPDMA1_REQUEST_MDF1_FLT2

HPDMA1 HW request is MDF1_FLT2

HPDMA1_REQUEST_MDF1_FLT3

HPDMA1 HW request is MDF1_FLT3

HPDMA1_REQUEST_MDF1_FLT4

HPDMA1 HW request is MDF1_FLT4

HPDMA1_REQUEST_MDF1_FLT5

HPDMA1 HW request is MDF1_FLT5

HPDMA1_REQUEST_UCPD1_TX

HPDMA1 HW request is UCPD1_TX

HPDMA1_REQUEST_UCPD1_RX

HPDMA1 HW request is UCPD1_RX

HPDMA1_REQUEST_DCMI_PSSI

HPDMA1 HW request is DCMI_PSSI

HPDMA1_REQUEST_I3C1_TC

HPDMA1 HW request is I3C1_TC

HPDMA1_REQUEST_I3C1_RS

HPDMA1 HW request is I3C1_RS

HPDMA1_REQUEST_I3C2_TC

HPDMA1 HW request is I3C2_TC

HPDMA1_REQUEST_I3C2_RS

HPDMA1 HW request is I3C2_RS

GPDMA1_REQUEST_JPEG_RX

GPDMA1 HW request is JPEG_DMA_RX

GPDMA1_REQUEST_JPEG_TX

GPDMA1 HW request is JPEG_DMA_TX

GPDMA1_REQUEST_XSPI1

GPDMA1 HW request is XSPI1

GPDMA1_REQUEST_XSPI2

GPDMA1 HW request is XSPI2

GPDMA1_REQUEST_XSPI3

GPDMA1 HW request is XSPI3

GPDMA1_REQUEST_FMC2_TXRX

GPDMA1 HW request is FMC2_TXRX

GPDMA1_REQUEST_FMC2_BCH

GPDMA1 HW request is FMC2_BCH

GPDMA1_REQUEST_ADC1

GPDMA1 HW request is ADC1

GPDMA1_REQUEST_ADC2

GPDMA1 HW request is ADC2

GPDMA1_REQUEST_CRYP_IN

GPDMA1 HW request is CRYP_IN

GPDMA1_REQUEST_CRYP_OUT

GPDMA1 HW request is CRYP_OUT

GPDMA1_REQUEST_SAES_OUT

GPDMA1 HW request is SAES_OUT

GPDMA1_REQUEST_SAES_IN

GPDMA1 HW request is SAES_IN

GPDMA1_REQUEST_HASH_IN

GPDMA1 HW request is HASH_IN

GPDMA1_REQUEST_TIM1_CH1

GPDMA1 HW request is TIM1_CH1

GPDMA1_REQUEST_TIM1_CH2

GPDMA1 HW request is TIM1_CH2

GPDMA1_REQUEST_TIM1_CH3

GPDMA1 HW request is TIM1_CH3

GPDMA1_REQUEST_TIM1_CH4

GPDMA1 HW request is TIM1_CH4

GPDMA1_REQUEST_TIM1_UP

GPDMA1 HW request is TIM1_UP

GPDMA1_REQUEST_TIM1_TRIG

GPDMA1 HW request is TIM1_TRIG

GPDMA1_REQUEST_TIM1_COM

GPDMA1 HW request is TIM1_COM

GPDMA1_REQUEST_TIM2_CH1

GPDMA1 HW request is TIM2_CH1

GPDMA1_REQUEST_TIM2_CH2

GPDMA1 HW request is TIM2_CH2

GPDMA1_REQUEST_TIM2_CH3

GPDMA1 HW request is TIM2_CH3

GPDMA1_REQUEST_TIM2_CH4

GPDMA1 HW request is TIM2_CH4

GPDMA1_REQUEST_TIM2_UP

GPDMA1 HW request is TIM2_UP

GPDMA1_REQUEST_TIM2_TRIG

GPDMA1 HW request is TIM2_TRIG

GPDMA1_REQUEST_TIM3_CH1

GPDMA1 HW request is TIM3_CH1

GPDMA1_REQUEST_TIM3_CH2

GPDMA1 HW request is TIM3_CH2

GPDMA1_REQUEST_TIM3_CH3

GPDMA1 HW request is TIM3_CH3

GPDMA1_REQUEST_TIM3_CH4

GPDMA1 HW request is TIM3_CH4

GPDMA1_REQUEST_TIM3_UP

GPDMA1 HW request is TIM3_UP

GPDMA1_REQUEST_TIM3_TRIG

GPDMA1 HW request is TIM3_TRIG

GPDMA1_REQUEST_TIM4_CH1

GPDMA1 HW request is TIM4_CH1

GPDMA1_REQUEST_TIM4_CH2

GPDMA1 HW request is TIM4_CH2

GPDMA1_REQUEST_TIM4_CH3

GPDMA1 HW request is TIM4_CH3

GPDMA1_REQUEST_TIM4_CH4

GPDMA1 HW request is TIM4_CH4

GPDMA1_REQUEST_TIM4_UP

GPDMA1 HW request is TIM4_UP

GPDMA1_REQUEST_TIM4_TRIG

GPDMA1 HW request is TIM4_TRIG

GPDMA1_REQUEST_TIM5_CH1

GPDMA1 HW request is TIM5_CH1

GPDMA1_REQUEST_TIM5_CH2

GPDMA1 HW request is TIM5_CH2

GPDMA1_REQUEST_TIM5_CH3

GPDMA1 HW request is TIM5_CH3

GPDMA1_REQUEST_TIM5_CH4

GPDMA1 HW request is TIM5_CH4

GPDMA1_REQUEST_TIM5_UP

GPDMA1 HW request is TIM5_UP

GPDMA1_REQUEST_TIM5_TRIG

GPDMA1 HW request is TIM5_TRIG

GPDMA1_REQUEST_TIM6_UP

GPDMA1 HW request is TIM6_UP

GPDMA1_REQUEST_TIM7_UP

GPDMA1 HW request is TIM6_UP

GPDMA1_REQUEST_TIM8_CH1

GPDMA1 HW request is TIM8_CH1

GPDMA1_REQUEST_TIM8_CH2

GPDMA1 HW request is TIM8_CH2

GPDMA1_REQUEST_TIM8_CH3

GPDMA1 HW request is TIM8_CH3

GPDMA1_REQUEST_TIM8_CH4

GPDMA1 HW request is TIM8_CH4

GPDMA1_REQUEST_TIM8_UP

GPDMA1 HW request is TIM8_UP

GPDMA1_REQUEST_TIM8_TRIG

GPDMA1 HW request is TIM8_TRIG

GPDMA1_REQUEST_TIM8_COM

GPDMA1 HW request is TIM8_COM

GPDMA1_REQUEST_TIM15_CH1

GPDMA1 HW request is TIM15_CH1

GPDMA1_REQUEST_TIM15_CH2

GPDMA1 HW request is TIM15_CH2

GPDMA1_REQUEST_TIM15_UP

GPDMA1 HW request is TIM15_UP

GPDMA1_REQUEST_TIM15_TRIG

GPDMA1 HW request is TIM15_TRIG

GPDMA1_REQUEST_TIM15_COM

GPDMA1 HW request is TIM15_COM

GPDMA1_REQUEST_TIM16_CH1

GPDMA1 HW request is TIM16_CH1

GPDMA1_REQUEST_TIM16_UP

GPDMA1 HW request is TIM16_UP

GPDMA1_REQUEST_TIM16_COM

GPDMA1 HW request is TIM16_COM

GPDMA1_REQUEST_TIM17_CH1

GPDMA1 HW request is TIM17_CH1

GPDMA1_REQUEST_TIM17_UP

GPDMA1 HW request is TIM17_UP

GPDMA1_REQUEST_TIM17_COM

GPDMA1 HW request is TIM17_COM

GPDMA1_REQUEST_TIM18_CH1

GPDMA1 HW request is TIM18_CH1

GPDMA1_REQUEST_TIM18_UP

GPDMA1 HW request is TIM18_UP

GPDMA1_REQUEST_TIM18_COM

GPDMA1 HW request is TIM18_COM

GPDMA1_REQUEST_LPTIM1_IC1

GPDMA1 HW request is LPTIM1_IC1

GPDMA1_REQUEST_LPTIM1_IC2

GPDMA1 HW request is LPTIM1_IC2

GPDMA1_REQUEST_LPTIM1_UE

GPDMA1 HW request is LPTIM1_UE

GPDMA1_REQUEST_LPTIM2_IC1

GPDMA1 HW request is LPTIM2_IC1

GPDMA1_REQUEST_LPTIM2_IC2

GPDMA1 HW request is LPTIM2_IC2

GPDMA1_REQUEST_LPTIM2_UE

GPDMA1 HW request is LPTIM2_UE

GPDMA1_REQUEST_LPTIM3_IC1

GPDMA1 HW request is LPTIM3_IC1

GPDMA1_REQUEST_LPTIM3_IC2

GPDMA1 HW request is LPTIM3_IC2

GPDMA1_REQUEST_LPTIM3_UE

GPDMA1 HW request is LPTIM3_UE

GPDMA1_REQUEST_SPI1_RX

GPDMA1 HW request is SPI1_RX

GPDMA1_REQUEST_SPI1_TX

GPDMA1 HW request is SPI1_TX

GPDMA1_REQUEST_SPI2_RX

GPDMA1 HW request is SPI2_RX

GPDMA1_REQUEST_SPI2_TX

GPDMA1 HW request is SPI2_TX

GPDMA1_REQUEST_SPI3_RX

GPDMA1 HW request is SPI3_RX

GPDMA1_REQUEST_SPI3_TX

GPDMA1 HW request is SPI3_TX

GPDMA1_REQUEST_SPI4_RX

GPDMA1 HW request is SPI4_RX

GPDMA1_REQUEST_SPI4_TX

GPDMA1 HW request is SPI4_TX

GPDMA1_REQUEST_SPI5_RX

GPDMA1 HW request is SPI5_RX

GPDMA1_REQUEST_SPI5_TX

GPDMA1 HW request is SPI5_TX

GPDMA1_REQUEST_SPI6_RX

GPDMA1 HW request is SPI6_RX

GPDMA1_REQUEST_SPI6_TX

GPDMA1 HW request is SPI6_TX

GPDMA1_REQUEST_SAI1_A

GPDMA1 HW request is SAI1_A

GPDMA1_REQUEST_SAI1_B

GPDMA1 HW request is SAI1_B

GPDMA1_REQUEST_SAI2_A

GPDMA1 HW request is SAI2_A

GPDMA1_REQUEST_SAI2_B

GPDMA1 HW request is SAI2_B

GPDMA1_REQUEST_I2C1_RX

GPDMA1 HW request is I2C1_RX

GPDMA1_REQUEST_I2C1_TX

GPDMA1 HW request is I2C1_TX

GPDMA1_REQUEST_I2C2_RX

GPDMA1 HW request is I2C2_RX

GPDMA1_REQUEST_I2C2_TX

GPDMA1 HW request is I2C2_TX

GPDMA1_REQUEST_I2C3_RX

GPDMA1 HW request is I2C3_RX

GPDMA1_REQUEST_I2C3_TX

GPDMA1 HW request is I2C3_TX

GPDMA1_REQUEST_I2C4_RX

GPDMA1 HW request is I2C4_RX

GPDMA1_REQUEST_I2C4_TX

GPDMA1 HW request is I2C4_TX

GPDMA1_REQUEST_I3C1_RX

GPDMA1 HW request is I3C1_RX

GPDMA1_REQUEST_I3C1_TX

GPDMA1 HW request is I3C1_TX

GPDMA1_REQUEST_I3C2_RX

GPDMA1 HW request is I3C2_RX

GPDMA1_REQUEST_I3C2_TX

GPDMA1 HW request is I3C2_TX

GPDMA1_REQUEST_USART1_RX

GPDMA1 HW request is USART1_RX

GPDMA1_REQUEST_USART1_TX

GPDMA1 HW request is USART1_TX

GPDMA1_REQUEST_USART2_RX

GPDMA1 HW request is USART2_RX

GPDMA1_REQUEST_USART2_TX

GPDMA1 HW request is USART2_TX

GPDMA1_REQUEST_USART3_RX

GPDMA1 HW request is USART3_RX

GPDMA1_REQUEST_USART3_TX

GPDMA1 HW request is USART3_TX

GPDMA1_REQUEST_UART4_RX

GPDMA1 HW request is UART4_RX

GPDMA1_REQUEST_UART4_TX

GPDMA1 HW request is UART4_TX

GPDMA1_REQUEST_UART5_RX

GPDMA1 HW request is UART5_RX

GPDMA1_REQUEST_UART5_TX

GPDMA1 HW request is UART5_TX

GPDMA1_REQUEST_USART6_RX

GPDMA1 HW request is USART6_RX

GPDMA1_REQUEST_USART6_TX

GPDMA1 HW request is USART6_TX

GPDMA1_REQUEST_UART7_RX

GPDMA1 HW request is UART7_RX

GPDMA1_REQUEST_UART7_TX

GPDMA1 HW request is UART7_TX

GPDMA1_REQUEST_UART8_RX

GPDMA1 HW request is UART8_RX

GPDMA1_REQUEST_UART8_TX

GPDMA1 HW request is UART8_TX

GPDMA1_REQUEST_UART9_RX

GPDMA1 HW request is UART9_RX

GPDMA1_REQUEST_UART9_TX

GPDMA1 HW request is UART9_TX

GPDMA1_REQUEST_USART10_RX

GPDMA1 HW request is USART10_RX

GPDMA1_REQUEST_USART10_TX

GPDMA1 HW request is USART10_TX

GPDMA1_REQUEST_LPUART1_RX

GPDMA1 HW request is LPUART1_RX

GPDMA1_REQUEST_LPUART1_TX

GPDMA1 HW request is LPUART1_TX

GPDMA1_REQUEST_SPDIFRX_CS

GPDMA1 HW request is SPDIFRX_CS

GPDMA1_REQUEST_SPDIFRX_DT

GPDMA1 HW request is SPDIFRX_DT

GPDMA1_REQUEST_ADF1_FLT0

GPDMA1 HW request is ADF1_FLT0

GPDMA1_REQUEST_MDF1_FLT0

GPDMA1 HW request is MDF1_FLT0

GPDMA1_REQUEST_MDF1_FLT1

GPDMA1 HW request is MDF1_FLT1

GPDMA1_REQUEST_MDF1_FLT2

GPDMA1 HW request is MDF1_FLT2

GPDMA1_REQUEST_MDF1_FLT3

GPDMA1 HW request is MDF1_FLT3

GPDMA1_REQUEST_MDF1_FLT4

GPDMA1 HW request is MDF1_FLT4

GPDMA1_REQUEST_MDF1_FLT5

GPDMA1 HW request is MDF1_FLT5

GPDMA1_REQUEST_UCPD1_TX

GPDMA1 HW request is UCPD1_TX

GPDMA1_REQUEST_UCPD1_RX

GPDMA1 HW request is UCPD1_RX

GPDMA1_REQUEST_DCMI_PSSI

GPDMA1 HW request is DCMI_PSSI

GPDMA1_REQUEST_I3C1_TC

GPDMA1 HW request is I3C1_TC

GPDMA1_REQUEST_I3C1_RS

GPDMA1 HW request is I3C1_RS

GPDMA1_REQUEST_I3C2_TC

GPDMA1 HW request is I3C2_TC

GPDMA1_REQUEST_I3C2_RS

GPDMA1 HW request is I3C2_RS

DMA_REQUEST_SW

DMA SW request

DMA Source Data Width

DMA_SRC_DATAWIDTH_BYTE

Source data width : Byte

DMA_SRC_DATAWIDTH_HALFWORD

Source data width : HalfWord

DMA_SRC_DATAWIDTH_WORD

Source data width : Word

DMA_SRC_DATAWIDTH_DOUBLEWORD

Source data width : DoubleWord

DMA Source Increment Mode

DMA_SINC_FIXED

Source fixed single / burst

DMA_SINC_INCREMENTED

Source incremented single / burst

DMA Transfer Allocated Port

DMA_SRC_ALLOCATED_PORT0

Source allocated Port 0

DMA_SRC_ALLOCATED_PORT1

Source allocated Port 1

DMA_DEST_ALLOCATED_PORT0

Destination allocated Port 0

DMA_DEST_ALLOCATED_PORT1

Destination allocated Port 1

DMA Transfer Direction

DMA_PERIPH_TO_MEMORY

Peripheral to memory direction

DMA_MEMORY_TO_PERIPH

Memory to peripheral direction

DMA_MEMORY_TO_MEMORY

Memory to memory direction

DMA Transfer Event Mode

DMA_TCEM_BLOCK_TRANSFER

The TC event is generated at the end of each block and the HT event is generated at the half of each block

DMA_TCEM_REPEATED_BLOCK_TRANSFER

The TC event is generated at the end of the repeated block and the HT event is generated at the half of the repeated block

DMA_TCEM_EACH_LL_ITEM_TRANSFER

The TC event is generated at the end of each linked-list item and the HT event is generated at the half of each linked-list item

DMA_TCEM_LAST_LL_ITEM_TRANSFER

The TC event is generated at the end of the last linked-list item and the HT event is generated at the half of the last linked-list item

DMA Transfer Mode

DMA_NORMAL

Normal DMA transfer

DMA_PFCTRL

HW request peripheral flow control mode

20 HAL DMA Extension Driver

20.1 DMAEx Firmware driver registers structures

20.1.1 DMA_DataHandlingConfTypeDef

DMA_DataHandlingConfTypeDef is defined in the stm32n6xx_hal_dma_ex.h

Data Fields

- ***uint32_t DataExchange***
- ***uint32_t DataAlignment***

Field Documentation

- ***uint32_t DMA_DataHandlingConfTypeDef::DataExchange***
  Specifies the DMA channel data exchange mode. This parameter can be a value of ***DMAEx_Data_Exchange***
- ***uint32_t DMA_DataHandlingConfTypeDef::DataAlignment***
  Specifies the DMA channel data padding and alignment mode This parameter can be a value of ***DMAEx_Data_Alignment***

20.1.2 DMA_TriggerConfTypeDef

DMA_TriggerConfTypeDef is defined in the stm32n6xx_hal_dma_ex.h

Data Fields

- ***uint32_t TriggerMode***
- ***uint32_t TriggerPolarity***
- ***uint32_t TriggerSelection***

Field Documentation

- ***uint32_t DMA_TriggerConfTypeDef::TriggerMode***
  Specifies the DMA channel trigger mode. This parameter can be a value of ***DMAEx_Trigger_Mode***
- ***uint32_t DMA_TriggerConfTypeDef::TriggerPolarity***
  Specifies the DMA channel trigger event polarity. This parameter can be a value of ***DMAEx_Trigger_Polarity***
- ***uint32_t DMA_TriggerConfTypeDef::TriggerSelection***
  Specifies the DMA channel trigger event selection. This parameter can be a value of ***DMAEx_Trigger_Selection***

20.1.3 DMA_RepeatBlockConfTypeDef

DMA_RepeatBlockConfTypeDef is defined in the stm32n6xx_hal_dma_ex.h

Data Fields

- ***uint32_t RepeatCount***
- ***int32_t SrcAddrOffset***
- ***int32_t DestAddrOffset***
- ***int32_t BlkSrcAddrOffset***
- ***int32_t BlkDestAddrOffset***

Field Documentation

- ***uint32_t DMA_RepeatBlockConfTypeDef::RepeatCount***
  Specifies the DMA channel repeat count (the number of repetitions of block). This parameter can be a value between 1 and 2048
- ***int32_t DMA_RepeatBlockConfTypeDef::SrcAddrOffset***
  Specifies the DMA channel single/burst source address offset : This parameter can be a value between -8191 and 8191. If source address offset > 0 => Increment the source address by offset from where the last single/burst transfer ends. If source address offset < 0 => Decrement the source address by offset from where the last single/burst transfer ends. If source address offset == 0 => The next single/burst source address starts from where the last transfer ends

- ***int32_t DMA_RepeatBlockConfTypeDef::DestAddrOffset***
  Specifies the DMA channel single/burst destination address offset signed value : This parameter can be a value between -8191 and 8191. If destination address offset > 0 => Increment the destination address by offset from where the last single/burst transfer ends. If destination address offset < 0 => Decrement the destination address by offset from where the last single/burst transfer ends. If destination address offset == 0 => The next single/burst destination address starts from where the last transfer ends.
- ***int32_t DMA_RepeatBlockConfTypeDef::BlkSrcAddrOffset***
  Specifies the DMA channel block source address offset signed value : This parameter can be a value between -65535 and 65535. If block source address offset > 0 => Increment the block source address by offset from where the last block ends. If block source address offset < 0 => Decrement the next block source address by offset from where the last block ends. If block source address offset == 0 => the next block source address starts from where the last block ends
- ***int32_t DMA_RepeatBlockConfTypeDef::BlkDestAddrOffset***
  Specifies the DMA channel block destination address offset signed value : This parameter can be a value between -65535 and 65535. If block destination address offset > 0 => Increment the block destination address by offset from where the last block ends. If block destination address offset < 0 => Decrement the next block destination address by offset from where the last block ends. If block destination address offset == 0 => the next block destination address starts from where the last block ends

20.1.4 DMA_NodeConfTypeDef

DMA_NodeConfTypeDef is defined in the stm32n6xx_hal_dma_ex.h

Data Fields

- ***uint32_t NodeType***
- ***DMA_InitTypeDef Init***
- ***DMA_DataHandlingConfTypeDef DataHandlingConfig***
- ***DMA_TriggerConfTypeDef TriggerConfig***
- ***DMA_RepeatBlockConfTypeDef RepeatBlockConfig***
- ***uint32_t SrcAddress***
- ***uint32_t DstAddress***
- ***uint32_t DataSize***
- ***uint32_t SrcSecure***
- ***uint32_t DestSecure***

Field Documentation

- ***uint32_t DMA_NodeConfTypeDef::NodeType***
  Specifies the DMA channel node type. This parameter can be a value of ***DMAEx_Node_Type***
- ***DMA_InitTypeDef DMA_NodeConfTypeDef::Init***
  Specifies the DMA channel basic configuration
- ***DMA_DataHandlingConfTypeDef DMA_NodeConfTypeDef::DataHandlingConfig***
  Specifies the DMA channel data handling channel configuration
- ***DMA_TriggerConfTypeDef DMA_NodeConfTypeDef::TriggerConfig***
  Specifies the DMA channel trigger configuration
- ***DMA_RepeatBlockConfTypeDef DMA_NodeConfTypeDef::RepeatBlockConfig***
  Specifies the DMA channel repeated block configuration
- ***uint32_t DMA_NodeConfTypeDef::SrcAddress***
  Specifies the source memory address
- ***uint32_t DMA_NodeConfTypeDef::DstAddress***
  Specifies the destination memory address
- ***uint32_t DMA_NodeConfTypeDef::DataSize***
  Specifies the source data size in bytes
- ***uint32_t DMA_NodeConfTypeDef::SrcSecure***
  Specifies the source security attribute
- ***uint32_t DMA_NodeConfTypeDef::DestSecure***
  Specifies the destination security attribute

20.1.5 DMA_NodeTypeDef

DMA_NodeTypeDef is defined in the stm32n6xx_hal_dma_ex.h

Data Fields

- ***uint32_t LinkRegisters***
- ***uint32_t NodeInfo***

Field Documentation

- ***uint32_t DMA_NodeTypeDef::LinkRegisters[8U]***
  Physical Node register description
- ***uint32_t DMA_NodeTypeDef::NodeInfo***
  Node information

20.1.6 __DMA_QListTypeDef

__DMA_QListTypeDef is defined in the stm32n6xx_hal_dma_ex.h

Data Fields

- ***DMA_NodeTypeDef * Head***
- ***DMA_NodeTypeDef * FirstCircularNode***
- ***uint32_t NodeNumber***
- ***__IO HAL_DMA_QStateTypeDef State***
- ***__IO uint32_t ErrorCode***
- ***__IO uint32_t Type***

Field Documentation

- ***DMA_NodeTypeDef* __DMA_QListTypeDef::Head***
  Specifies the queue head node
- ***DMA_NodeTypeDef* __DMA_QListTypeDef::FirstCircularNode***
  Specifies the queue first circular node
- ***uint32_t __DMA_QListTypeDef::NodeNumber***
  Specifies the queue node number
- ***__IO HAL_DMA_QStateTypeDef __DMA_QListTypeDef::State***
  Specifies the queue state
- ***__IO uint32_t __DMA_QListTypeDef::ErrorCode***
  Specifies the queue error code
- ***__IO uint32_t __DMA_QListTypeDef::Type***
  Specifies whether the queue is static or dynamic

20.1.7 DMA_NodeInQInfoTypeDef

DMA_NodeInQInfoTypeDef is defined in the stm32n6xx_hal_dma_ex.h

Data Fields

- ***uint32_t cllr_offset***
- ***uint32_t previousnode_addr***
- ***uint32_t currentnode_pos***
- ***uint32_t currentnode_addr***
- ***uint32_t nextnode_addr***

Field Documentation

- ***uint32_t DMA_NodeInQInfoTypeDef::cllr_offset***
- ***uint32_t DMA_NodeInQInfoTypeDef::previousnode_addr***
- ***uint32_t DMA_NodeInQInfoTypeDef::currentnode_pos***
- ***uint32_t DMA_NodeInQInfoTypeDef::currentnode_addr***
- ***uint32_t DMA_NodeInQInfoTypeDef::nextnode_addr***

20.2 DMAEx Firmware driver API description

The following section lists the various functions of the DMAEx library.

20.2.1 How to use this driver

Alternatively to the normal programming mode, a DMA channel can be programmed by a list of transfers, known as linked-list (list of Node items). Each node is defined by its data structure. Each node specifies a standalone DMA channel. When enabled, the DMA channel fetch the first linked-list node from SRAM (known as head node). When executed, the next linked list node will be fetched and executed. This operation is repeated until the end of the whole linked-list queue. Optionally, the linked-list can be linear where the last linked-list queue node is not linked to another queue node or circular where the last linked-list node is linked to any linked-list queue node.

- Linear linked-list: The DMA channel fetch and execute all DMA linked-list queue from first node (head node) to last node (tail node) ones. When the last node is completed, the DMA channel remains in idle state and another transfer can be lunched.
- Circular linked-list: The DMA channel fetch and execute all DMA linked-list queue from first node (head node) to last node (tail node). When last node is executed, the DMA channel fetches the first circular node another time and repeat the same sequence in an infinite loop (Circular transfer). To stop the DMA channel, an abort operation is required. This linked-list mode replaces the legacy circular transfers.

In order to reduce linked-list queue executing time and power consumption, the DMA channel supports executing the dynamic linked-list format. In fact, the DMA supports the execution of 2 types of linked-list formats : static and dynamic.

- Static linked-list: The static linked-list format refers to the full linked-list node where all DMA channel parameters are fetched and executed independently of the redundancy of information.
- Dynamic linked-list: The dynamic linked-list format refer to the customized linked-list node where only DMA channel necessary parameters are fetched and executed (Example: data size = 20 on previous node, and data size = 20 on the current node => No need to update it). For linked-list transfers, the DMA channel can execute the linked-list queue node by node. This feature is named link step mode. When activated, enabling the DMA channel first time allows to fetch the head node from memory then it stops. Then, another DMA channel enable is needed to execute the node. After that, keeping enabling the DMA channel is needed to execute each node until the end of linked-list queue. When the linked-list queue is circular, enabling the DMA channel in an infinite loop is required to keep the DMA channel running. This feature is useful for debug purpose or asynchronously executing queue nodes.

Each DMA channel transfer (normal or linked-list), is highly configurable according to DMA channel instance integrated in devices. These configuration can be :

- Repeated block configuration : If the feature is supported, the DMA channel can performs a repeated block transfers. Named also 2 dimension addressing transfers, this feature can transfer n iteration of programmed block transfer (Block transfer is the legacy data size). Additional to the repeat count of a block, DMA channel addresses can jump after at burst and block level. The jump length is a programmable parameter defined by DMA user.
  - Jump at burst level : The DMA channel keep an empty area, between each 2 consecutive bursts transmitted.
  - Jump at block level : The DMA channel keep an empty area, between each 2 consecutive blocks transmitted.
- Trigger : The DMA channel transfers can be conditioned by hardware signals edges (rising or falling) named hardware triggers. Trigger condition can be applied at :
  - Single/Burst level : Each single/burst data transmission is conditioned by a signal trigger hit.
  - Block level : Each block data transmission is conditioned by a signal trigger hit.
  - Repeated block level : Each repeated block data transmission is conditioned by a signal trigger hit.
  - Node level : Each node execution is conditioned by a signal trigger hit. The DMA channel can report a trigger overrun when detects more than 2 trigger signal edges before executing the current transfer.
- Data handling : The data handling feature is a FIFO capability that can be :
  - Padding pattern : Padding selected pattern (zero padding or sign extension) when the source data width is smaller than the destination data width at single level.
  - Truncation : Truncate section from the source data single when the source data width is bigger than the destination data width.
  - Pack/Unpack : Pack a set of data when source data width is smaller than the destination data width. Unpack a set of data when source data width is bigger than the destination data width.
  - Exchange : Exchange data at byte, half-word and word on the destination and at byte level on the source.

Each DMA channel transfer (normal or linked-list) when it is active, can be suspended and resumed at run time application. When trying to suspend an ongoing transfer, the DMA channel isn't suspended instantly but complete the current ongoing single/burst then it stops. When the DMA channel is suspended, the current transfer can be resumed instantly.

The DMA channel that supports FIFO, can report in real time the number of beats remains on destination (Output) FIFO level.

Linked-List Initialization and De-Initialization operation

Differently from normal transfers, DMA channel initialization and de-initialization need less parameters as the remaining transfer parameters are defined by linked-list nodes.

- Use HAL_DMAEx_List_Init() to initialize a DMA channel in linked-list mode according to programmed fields. When called, the DMA channel will be ready to execute linked-list queues.
- Use HAL_DMAEx_List_DeInit() to de-initialize a DMA channel in linked-list mode. When called, the DMA channel will be in reset. It is mandatory to reinitialize it for next transfer.

Linked-List I/O Operation

- Use HAL_DMAEx_List_Start() to start a DMA transfer in linked-list mode after the configuration of linked-list queue base address and offset in polling mode (Blocking mode).
- Use HAL_DMAEx_List_Start_IT() to start a DMA transfer in linked-list mode after the configuration of linked-list queue base address and offset in interrupt mode (Non-blocking mode).

Linked-List Management

The linked-list management is a software processing independently of DMA channel hardware. It allows to reset, build, create, insert, remove, replace, circularize, convert both nodes and queue in order to perform DMA channel transfers in linked-list mode. Linked-list APIs and types are adapted to reduce memory footprint.

Linked-list nodes building

At node level, the operations that can be done are building a new linked-list node or get a linked-list node information from a built node. The linked-list nodes have two forms according to 2 dimensions addressing capability. The linear addressing nodes contains the information of all DMA channel features except the 2 dimension addressing features and the 2 dimensions addressing nodes contain the information of all available features.

- Use HAL_DMAEx_List_BuildNode() to build the DMA linked-list node according to the specified parameters. Build operation allow to convert the specified parameter in values known by the DMA channel and place them in memory. Placing DMA linked-list in SRAM must be done in accordance to product specification to ensure that the link access port can access to the specified SRAM.
  - The DMA linked-list node parameter address should be 32bit aligned and should not exceed the 64 KByte addressable space.
- Use HAL_DMAEx_List_GetNodeConfig() to get the specified configuration parameter on building node. This API can be used when need to change few parameter to build new node.

Inserting nodes to linked-list queue

In order to build a sequence of DMA transaction with different configuration, we need to insert built node at linked-list queue (node present an elementary DMA transaction) in linked-list queue on any position to have the full flexibility of ordering nodes or extend the sequence of queue transactions.

- Use HAL_DMAEx_List_InsertNode() to insert new built node in any queue position of linked-list queue according to selecting previous node. When calling this API with previous node parameter is NULL, the inserted node will be placed at the head of the linked-list queue.
  - This API must be used after HAL_DMAEx_List_BuildNode() otherwise an error will be returned.
  - This API must be called for static queues format.
  - This API shall be avoided when adding new node at the head or the tail of queue (overhead of footprint and performance : use HAL_DMAEx_List_InsertNode_Head() or HAL_DMAEx_List_InsertNode_Tail() instead).

- Use HAL_DMAEx_List_InsertNode_Head() to insert new built node at the head of linked-list queue. The head node will not be overwritten but will be the second queue node.
  - This API must be used after HAL_DMAEx_List_BuildNode() otherwise an error will be returned.
  - This API must be called for static queues format.
- Use HAL_DMAEx_List_InsertNode_Tail() to insert new built node at the tail of linked-list queue. The tail node will not be overwritten but will be the penultimate queue node.
  - This API must be used after HAL_DMAEx_List_BuildNode() otherwise an error will be returned.
  - This API must be called for static queues format.

Removing nodes from linked-list queue

There is some cases when removing a node from linked-list queue is needed (need to remove an elementary DMA transaction). Removing node allows to unlink a node from DMA linked-list queue (NOT DELETED), so the removed node can be reused for another queue or to be added to the same queue without need to rebuild it in next step.

- Use HAL_DMAEx_List_RemoveNode() to remove any yet built and inserted node from linked-list queue according to selected node.
  - This API must be called for static queues format.
  - This API shall be avoided when removing the head or the tail of linked-list queue (overhead of footprint and performance : use HAL_DMAEx_List_RemoveNode_Head() or HAL_DMAEx_List_RemoveNode_Tail() instead).
- Use HAL_DMAEx_List_RemoveNode_Head() to remove the head node from linked-list queue.
  - This API must be called for static queues format.
- Use HAL_DMAEx_List_RemoveNode_Tail() to remove the tail node from linked-list queue.
  - This API must be called for static queues format.

Replacing nodes on linked-list queue

There is some cases when replacing a node from linked-list queue is needed (need to replace an elementary DMA transfer, by another one that have not the same configuration). Replacing node allows to unlink the node to be replaced from DMA linked-list queue (NOT DELETED) and link instead a new node. So the replaced node can be reused for another queue or to be added to the same queue without need to rebuild it in next step and the new node cannot be reused except when remove it or replaced in next step.

- Use HAL_DMAEx_List_ReplaceNode() to replace any yet built and inserted node on linked-list queue according to selected node.
  - This API must be called for static queues format.
  - This API shall be avoided when replacing the head or the tail linked-list queue (overhead of footprint and performance : use HAL_DMAEx_List_ReplaceNode_Head() or HAL_DMAEx_List_ReplaceNode_Tail() instead).
- Use HAL_DMAEx_List_ReplaceNode_Head() to replace the head node of linked-list queue.
  - This API must be called for static queues format.
- Use HAL_DMAEx_List_ReplaceNode_Tail() to replace the tail node from linked-list queue.
  - This API must be called for static queues format.

Reset linked-list queue

After finishing using a linked-list queue, it can be reset and cleared and it's content nodes will be unlinked (NOT DELETED) and reused on another queue.

- Use HAL_DMAEx_List_ResetQ() to reset a linked-list queue and unlink all it's content nodes.
  - This API must be called for ready state queues.
  - This API must be called for static queues format.

Inserting linked-list queue

To ensure the flexibility of building linked-list queue by their targeted functionalities (Example: 3 nodes for action 1 and 5 nodes for action 2), it is possible to build a queue for action 1 that contains action 1 nodes and a queue for action 2 that contains action 2 nodes then concatenating the 2 queues. So, there are some cases where the management of linked-list at queue granularity is needed.

- Use HAL_DMAEx_List_InsertQ() to insert source linked-list queue to a destination linked-list queue according to selecting previous node.
  - This API must be called for static queues format.
  - This API shall be avoided when inserting source linked-list queue at the head or the tail of destination queue (overhead of footprint and performance : use HAL_DMAEx_List_InsertQ_Head() or HAL_DMAEx_List_InsertQ_Tail() instead).
- Use HAL_DMAEx_List_InsertQ_Head() to insert a source linked-list queue at the head of linked-list destination queue.
  - This API must be called for static queues format.
- Use HAL_DMAEx_List_InsertQ_Tail() to insert a source linked-list queue at the tail of linked-list destination queue.
  - This API must be called for static queues format.

Circularizing linked-list queue

In order to perform tasks in infinite loop with DMA channel, it is possible to circularize the linked-list queues. Circularizing queue allows to link last linked-list queue node to any previous node of the same queue (This node is named first circular queue). When the first circular node is the head node, all linked-list queue nodes will be executed in infinite loop. When the first circular node is not the head nodes, all precedent nodes are executed once and all remaining nodes are executed in an infinite loop.

- Use HAL_DMAEx_List_SetCircularModeConfig() to circularize the linked-list queue according to first circular node selected.
  - This API must be called for static queues format.
  - This API shall be avoided when first circular node is the head linked-list queue node (overhead of footprint and performance : use HAL_DMAEx_List_SetCircularMode() instead).
- Use HAL_DMAEx_List_SetCircularMode() to circularize the linked-list queue with linking last queue node with first queue node.
  - This API must be called for static queues format.
- Use HAL_DMAEx_List_ClearCircularMode() to clear any linked-list queue circular configuration.
  - This API must be called for static queues format.

Converting linked-list queue

To have the best DMA channel linked-list queue execution, it is recommended to convert yet build linked-list queue to dynamic format (Static is the default format). When linked-list queue becomes dynamic, all queue nodes are optimized and only changed parameters will be updated between nodes. So, the DMA will fetch only changes parameters instead of the whole node.

- Use HAL_DMAEx_List_ConvertQToDynamic() to convert a linked-list queue to dynamic format.
  - This API must be called for ready state queues.
  - This API must be called for static queues format.
  - This API must be called as the last API before starting the DMA channel in linked-list mode.
- Use HAL_DMAEx_List_ConvertQToStatic() to convert a linked-list queue to static format.
  - This API must be called for ready state queues.
  - This API must be called for dynamic queues format.
  - This API must be called as the first API after the full execution of linked-list queue when the execution mode is linear (not circular) if it is dynamic and a linked-list queue management is needed.
  - This API must be called as the first API after the aborting the execution of the current linked-list queue when the execution mode is linear (not circular) if it is dynamic and a linked-list queue management is needed.

When converting a circular queue to dynamic format and when the first circular node is the last queue node, it is recommended to duplicate the last circular node in order to ensure the full optimization when calling HAL_DMAEx_List_ConvertQToDynamic() API. In this case, updated information are only addresses which allow to reduce 4 words of update for linear nodes per node execution and 6 words update for 2 dimensions addressing nodes per node execution.

Linking linked-list queue to DMA channel

In order to have the possibility of the creation of an infinity queues (limited by available memory size), the building of linked-list queue is fully independent from DMA channels. It is possible to build all needed queues if their size is less then available memory at startup time, then linking each time when needed a linked-list queue to an idle DMA channel.

- Use HAL_DMAEx_List_LinkQ() to link a ready linked-list queue to ready DMA channel.
  - This API supports the two format of linked-list (Static and dynamic).
  - This API must be called for ready state queues and DMA channels.
- Use HAL_DMAEx_List_ConvertQToStatic() to unlink a ready linked-list queue to ready DMA channel.
  - This API supports the two format of linked-list (Static and dynamic).
  - This API must be called for ready state queues and DMA channels.

User sequence

To use cleanly the DMA linked-list library, ensure to apply the following call sequences :

- Linear transfer : Linked-list queue building
  - HAL_DMAEx_List_BuildNode()
  - HAL_DMAEx_List_InsertNode_Tail() . . .
  - HAL_DMAEx_List_BuildNode()
  - HAL_DMAEx_List_InsertNode_Tail()
  - HAL_DMAEx_List_ConvertQToDynamic() Linked-list queue execution
  - HAL_DMAEx_List_Init()
  - HAL_DMAEx_List_LinkQ()
  - HAL_DMAEx_List_Start() / HAL_DMAEx_List_Start_IT()
  - HAL_DMAEx_List_UnLinkQ()
  - HAL_DMAEx_List_DeInit()
- Circular transfer : Linked-list queue building
  - HAL_DMAEx_List_BuildNode()
  - HAL_DMAEx_List_InsertNode_Tail() . . .
  - HAL_DMAEx_List_BuildNode()
  - HAL_DMAEx_List_InsertNode_Tail()
  - HAL_DMAEx_List_SetCircularModeConfig() / HAL_DMAEx_List_SetCircularMode()
  - HAL_DMAEx_List_ConvertQToDynamic() Linked-list queue execution
  - HAL_DMAEx_List_Init()
  - HAL_DMAEx_List_LinkQ()
  - HAL_DMAEx_List_Start() / HAL_DMAEx_List_Start_IT()
  - HAL_DMA_Abort() / HAL_DMA_Abort_IT()
  - HAL_DMAEx_List_UnLinkQ()
  - HAL_DMAEx_List_DeInit()

Data Handling

- In order to avoid some CPU data processing in several cases, the DMA channel provides some features related to FIFO capabilities titled data handling.
  - Padding pattern
  - Padding selected pattern (zero padding or sign extension) when the source data width is smaller than the destination data width at single level.
  - Zero padding
    - (Source : 0xABAB ------> Destination : 0xABAB0000)
  - Sign bit extension
    - (Source : 0x0ABA ------> Destination : 0x00000ABA)
    - (Source : 0xFABA ------> Destination : 0xFFFFFABA)
  - Truncation :
  - Truncate section from the source data single when the source data width is bigger than the destination data width.
  - Left truncation
    - (Source : 0xABABCDCD ------> Destination : 0xCDCD)
  - Right truncation
    - (Source : 0xABABCDCD ------> Destination : 0xABAB)
  - Pack/Unpack :
  - Pack a set of data when source data width is smaller than the destination data width.
  - Unpack a set of data when source data width is bigger than the destination data width.
  - Pack
    - (Source : 0xAB, 0xCD ------> Destination : 0xABCD)
  - UnPack
    - (Source : 0xABCD ------> Destination : 0xAB, 0xCD)
  - Exchange :
  - Exchange data at byte and half-word on the destination and at byte level on the source. Considering source and destination are both word type. Exchange operation can be as follows. In examples below, one exchange setting is enabled at a time.
  - Source byte exchange only
    - (Source : 0xAB12CD34 ------> Destination : 0xABCD1234)
  - Destination byte exchange only
    - (Source : 0xAB12CD34 ------> Destination : 0x12AB34CD)
  - Destination half-word exchange only
    - (Source : 0xAB12CD34 ------> Destination : 0xCD34AB12)
  - In addition, in case of double-word, Exchange data at word level on the destination is also available.
  - Considering source and destination are both double-word type. word exchange only
    - (Source : 0xAB12CD34EF567890 ------> Destination : 0xEF567890AB12CD34)
- Use HAL_DMAEx_ConfigDataHandling() to configure data handling features. Previous elementary explained can be combined according to application needs.
  - This API is complementary of normal transfers.
  - This API must not be called for linked-list transfers as data handling information are configured at node level.

User sequence

To configure cleanly the DMA channel data handling, ensure to apply the following call sequence :

- Linear transfer :
  - HAL_DMA_Init()
  - HAL_DMAEx_ConfigDataHandling()
  - HAL_DMA_Start()

Repeated Block

- When available, this feature is used when the data size is higher then 65535 bytes (Maximum block size) or for scattering / gathering data.
  - Gather data
    - Source Destination
    - 0xAA 0xBB 0xAA 0xCC 0xAA ==> 0xAA 0xAA 0xAA
  - Scatter data
    - Source Destination
    - 0xAA 0xAA 0xAA ==> 0xAA 0xBB 0xAA 0xBB 0xAA
- Use HAL_DMAEx_ConfigRepeatBlock() to configure data repeated block feature. Jump addresses and incrementing or decrementing on source and destination can be combined to have the need application behavior.
  - This API is complementary of normal transfers.
  - This API must not be called for linked-list transfers as repeated block information are configured at node level.
  - This API must be called only for DMA channel that supports repeated block feature.

User sequence

To configure cleanly the DMA channel repeated block, ensure to apply the following call sequence :

- Linear transfer :
  - HAL_DMA_Init()
  - HAL_DMAEx_ConfigRepeatBlock()
  - HAL_DMA_Start()

Trigger Configuration

When application needs that DMA transfers are conditioned by internal or external events, the trigger feature can do that. Trigger signals are a set of device signal that are linked to DMA trigger inputs that allows to start the DMA transfers. To setup a trigger transfers, three DMA channel parameters are needed:

- Trigger mode This parameter specifies the trig level.
  - Block level
  - Repeated block level
  - Node level
  - Single / Burst level
- Trigger polarity This parameter specifies the DMA trigger sensitivity (Rising or falling).
- Trigger selection This parameter specifies the DMA trigger hardware signal.
- Use HAL_DMAEx_ConfigTrigger() to configure trigger feature.
  - This API is complementary to normal transfers APIs.
  - This API must not be called for linked-list transfers as trigger information are configured at node level.

User sequence

To configure cleanly the DMA channel trigger, ensure to apply the following call sequence :

- Linear transfer :
  - HAL_DMA_Init()
  - HAL_DMAEx_ConfigTrigger()
  - HAL_DMA_Start()

Suspend and resume operation

There are several cases when needs to suspend a DMA current transfer (Example: liberate bandwidth for more priority DMA channel transfer). Suspending DMA channel (same as abort) is available in polling (blocking mode) and interrupt (non-blocking mode) modes. When suspended, a DMA channel can be instantly resumed.

- Use HAL_DMAEx_Suspend() to suspend an ongoing DMA channel transfer in polling mode (Blocking mode).
- Use HAL_DMAEx_Suspend_IT() to suspend an ongoing DMA channel transfer in interrupt mode (Non-blocking mode).
- Use HAL_DMAEx_Resume() to resume a suspended DMA channel transfer execution.

FIFO status

In several cases, the information of FIFO level is useful to inform at application level how to process remaining data. When not empty, the DMA channel FIFO cannot be flashed only by reset.

- Use HAL_DMAEx_GetFifoLevel() to get the DMA channel FIFO level (available beats in FIFO).

20.2.2 Linked-List Initialization and De-Initialization Functions

This section provides functions allowing to initialize and de-initialize the DMA channel in linked-list mode.

- The HAL_DMAEx_List_Init() function follows the DMA channel linked-list mode configuration procedures as described in reference manual.
- The HAL_DMAEx_List_DeInit() function allows to de-initialize the DMA channel in linked-list mode.

This section contains the following APIs:

- ***HAL_DMAEx_List_Init()***
- ***HAL_DMAEx_List_DeInit()***

20.2.3 Linked-List IO Operation Functions

This section provides functions allowing to :

- Configure to start DMA transfer in linked-list mode.
- The HAL_DMAEx_List_Start() function allows to start the DMA channel transfer in linked-list mode (Blocking mode).
- The HAL_DMAEx_List_Start_IT() function allows to start the DMA channel transfer in linked-list mode (Non-blocking mode).
  - It is mandatory to register a linked-list queue to be executed by a DMA channel before starting transfer otherwise a HAL_ERROR will be returned.

This section contains the following APIs:

- ***HAL_DMAEx_List_Start()***
- ***HAL_DMAEx_List_Start_IT()***

20.2.4 Linked-List Management Functions

This section provides functions allowing to :

- Build linked-list node.
- Get linked-list node configuration.
- Insert node to linked-list queue in any queue position.
- Remove any node from linked-list queue.
- Replace any node from linked-list queue.
- Reset linked-list queue.
- Insert linked-list queue in any queue position.
- Set circular mode configuration to linked-list queue.
- Clear circular mode configuration from linked-list queue.
- Convert static linked-list queue to dynamic format.
- Convert dynamic linked-list queue to static format.
- Link linked-list queue to DMA channel.
- Unlink linked-list queue from DMA channel.

- The HAL_DMAEx_List_BuildNode() function allows to build linked-list node. Node type can be :
  - 2 dimensions addressing node.
  - Linear addressing node.
- The HAL_DMAEx_List_GetNodeConfig() function allows to get the linked-list node configuration from built node.
- The HAL_DMAEx_List_InsertNode() function allows to insert built linked-list node to static linked-list queue according to selected position.
- The HAL_DMAEx_List_InsertNode_Head() and HAL_DMAEx_List_InsertNode_Tail() functions allow to insert built linked-list node to the head (respectively the tail) of static linked-list queue.
- The HAL_DMAEx_List_RemoveNode() function allows to remove selected built linked-list node from static linked-list queue.
- The HAL_DMAEx_List_RemoveNode_Head() and HAL_DMAEx_List_RemoveNode_Tail() functions allow to remove the head (respectively the tail) built linked-list node from static linked-list queue.
- The HAL_DMAEx_List_ReplaceNode() function allows to replace selected built linked-list node from static linked-list queue.
- The HAL_DMAEx_List_ReplaceNode_Head() and HAL_DMAEx_List_ReplaceNode_Tail() functions allow to replace the head (respectively the tail) built linked-list node of static linked-list queue.
- The HAL_DMAEx_List_ResetQ() function allows to reset static linked-list queue and unlink all built linked-list nodes.
- The HAL_DMAEx_List_InsertQ() function allows to insert static linked-list source queue to static linked-list destination queue according to selected position.
- The HAL_DMAEx_List_InsertQ_Head() and HAL_DMAEx_List_InsertQ_Tail() functions allow to insert static linked-list source queue to the head (respectively the tail) of static linked-list destination queue.
- The HAL_DMAEx_List_SetCircularModeConfig() function allows to link the last static linked-list queue node to the selected first circular node.
- The HAL_DMAEx_List_SetCircularMode() function allows to link the last static linked-list queue node to the first static linked-list queue node.
- The HAL_DMAEx_List_ClearCircularMode() function allows to unlink the last static linked-list queue node from any first circular node position.
- The HAL_DMAEx_List_ConvertQToDynamic() function allows to convert the static linked-list queue to dynamic format. (Optimized queue execution)
- The HAL_DMAEx_List_ConvertQToStatic() function allows to convert the dynamic linked-list queue to static format. (Not optimized queue execution)
- The HAL_DMAEx_List_LinkQ() function allows to link the (Dynamic / Static) linked-list queue to DMA channel to be executed.
- The HAL_DMAEx_List_UnLinkQ() function allows to unlink the (Dynamic / Static) linked-list queue from DMA channel when execution is completed.

This section contains the following APIs:

- ***HAL_DMAEx_List_BuildNode()***
- ***HAL_DMAEx_List_GetNodeConfig()***
- ***HAL_DMAEx_List_InsertNode()***
- ***HAL_DMAEx_List_InsertNode_Head()***
- ***HAL_DMAEx_List_InsertNode_Tail()***
- ***HAL_DMAEx_List_RemoveNode()***
- ***HAL_DMAEx_List_RemoveNode_Head()***
- ***HAL_DMAEx_List_RemoveNode_Tail()***
- ***HAL_DMAEx_List_ReplaceNode()***
- ***HAL_DMAEx_List_ReplaceNode_Head()***
- ***HAL_DMAEx_List_ReplaceNode_Tail()***
- ***HAL_DMAEx_List_ResetQ()***
- ***HAL_DMAEx_List_InsertQ()***
- ***HAL_DMAEx_List_InsertQ_Head()***
- ***HAL_DMAEx_List_InsertQ_Tail()***

- *HAL_DMAEx_List_SetCircularModeConfig()*
- *HAL_DMAEx_List_SetCircularMode()*
- *HAL_DMAEx_List_ClearCircularMode()*
- *HAL_DMAEx_List_ConvertQToDynamic()*
- *HAL_DMAEx_List_ConvertQToStatic()*
- *HAL_DMAEx_List_LinkQ()*
- *HAL_DMAEx_List_UnLinkQ()*

20.2.5 Data handling, repeated block and trigger configuration functions

This section provides functions allowing to :

- Configure DMA channel data handling.
- Configure DMA channel repeated block.
- Configure DMA channel trigger.
- The HAL_DMAEx_ConfigDataHandling() function allows to configure DMA channel data handling.
  - GPDMA or HPDMA data handling : byte-based reordering, packing/unpacking, padding/truncation, sign extension and left/right alignment.
- The HAL_DMAEx_ConfigTrigger() function allows to configure DMA channel HW triggers.
- The HAL_DMAEx_ConfigRepeatBlock() function allows to configure DMA channel repeated block.
  - This feature is available only for channel that supports 2 dimensions addressing capability.

This section contains the following APIs:

- *HAL_DMAEx_ConfigDataHandling()*
- *HAL_DMAEx_ConfigTrigger()*
- *HAL_DMAEx_ConfigRepeatBlock()*

20.2.6 Suspend and resume operation functions

This section provides functions allowing to :

- Suspend any ongoing DMA channel transfer.
- Resume any suspended DMA channel transfer.
- The HAL_DMAEx_Suspend() function allows to suspend any ongoing DMA channel transfer in polling mode (Blocking mode).
- The HAL_DMAEx_Suspend_IT() function allows to suspend any ongoing DMA channel transfer in interrupt mode (Non-blocking mode).
- The HAL_DMAEx_Resume() function allows to resume any suspended DMA channel transfer.

This section contains the following APIs:

- *HAL_DMAEx_Suspend()*
- *HAL_DMAEx_Suspend_IT()*
- *HAL_DMAEx_Resume()*

20.2.7 Fifo status function

This section provides function allowing to get DMA channel FIFO level.

- The HAL_DMAEx_GetFifoLevel() function allows to return the number of available write beats in the FIFO, in units of the programmed destination data.
  - This API is available only for DMA channels that supports FIFO.

This section contains the following APIs:

- *HAL_DMAEx_GetFifoLevel()*

20.2.8 Detailed description of functions

HAL_DMAEx_List_Init

Function name

HAL_StatusTypeDef HAL_DMAEx_List_Init (DMA_HandleTypeDef *const hdma)

Function description

Initialize the DMA channel in linked-list mode according to the specified parameters in the DMA_InitLinkedListTypeDef and create the associated handle.

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.

Return values

- **HAL:** status.

HAL_DMAEx_List_DeInit

Function name

HAL_StatusTypeDef HAL_DMAEx_List_DeInit (DMA_HandleTypeDef *const hdma)

Function description

DeInitialize the DMA channel when it is configured in linked-list mode.

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.

Return values

- **HAL:** status.

HAL_DMAEx_List_Start

Function name

HAL_StatusTypeDef HAL_DMAEx_List_Start (DMA_HandleTypeDef *const hdma)

Function description

Start the DMA channel transfer in linked-list mode (Blocking mode).

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.

Return values

- **HAL:** status.

HAL_DMAEx_List_Start_IT

Function name

HAL_StatusTypeDef HAL_DMAEx_List_Start_IT (DMA_HandleTypeDef *const hdma)

Function description

Starts the DMA channel transfer in linked-list mode with interrupts enabled (Non-blocking mode).

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.

Return values

- **HAL:** status.

HAL_DMAEx_List_BuildNode

Function name

HAL_StatusTypeDef HAL_DMAEx_List_BuildNode (DMA_NodeConfTypeDef const *const pNodeConfig, DMA_NodeTypeDef *const pNode)

Function description

Build a DMA channel node according to the specified parameters in the DMA_NodeConfTypeDef.

Parameters

- **pNodeConfig:** : Pointer to a DMA_NodeConfTypeDef structure that contains the configuration information for the specified DMA linked-list Node.
- **pNode:** : Pointer to a DMA_NodeTypeDef structure that contains linked-list node registers configurations.

Return values

- **HAL:** status.

Notes

- The DMA linked-list node parameter address should be 32bit aligned and should not exceed the 64 KByte addressable space.
- Warning if AXI port is selected with HPDMA, the maximum source (and destination) length should be less than 17. Otherwise, an error will be returned and no initialization performed.

HAL_DMAEx_List_GetNodeConfig

Function name

HAL_StatusTypeDef HAL_DMAEx_List_GetNodeConfig (DMA_NodeConfTypeDef *const pNodeConfig, DMA_NodeTypeDef const *const pNode)

Function description

Get a DMA channel node configuration.

Parameters

- **pNodeConfig:** : Pointer to a DMA_NodeConfTypeDef structure that contains the configuration information for the specified DMA linked-list Node.
- **pNode:** : Pointer to a DMA_NodeTypeDef structure that contains linked-list node registers configurations.

Return values

- **HAL:** status.

HAL_DMAEx_List_InsertNode

Function name

HAL_StatusTypeDef HAL_DMAEx_List_InsertNode (DMA_QListTypeDef *const pQList, DMA_NodeTypeDef *const pPrevNode, DMA_NodeTypeDef *const pNewNode)

Function description

Insert new node in any queue position of linked-list queue according to selecting previous node.

Parameters

- **pQList:** : Pointer to a DMA_QListTypeDef structure that contains queue information.
- **pPrevNode:** : Pointer to a DMA_NodeTypeDef structure that contains linked-list previous node registers configurations.
- **pNewNode:** : Pointer to a DMA_NodeTypeDef structure that contains linked-list new node registers configurations.

Return values

- **HAL:** status.

HAL_DMAEx_List_InsertNode_Head

Function name

HAL_StatusTypeDef HAL_DMAEx_List_InsertNode_Head (DMA_QListTypeDef *const pQList, DMA_NodeTypeDef *const pNewNode)

Function description

Insert new node at the head of linked-list queue.

Parameters

- **pQList:** : Pointer to a DMA_QListTypeDef structure that contains queue information.
- **pNewNode:** : Pointer to a DMA_NodeTypeDef structure that contains linked-list new node registers configurations.

Return values

- **HAL:** status.

HAL_DMAEx_List_InsertNode_Tail

Function name

HAL_StatusTypeDef HAL_DMAEx_List_InsertNode_Tail (DMA_QListTypeDef *const pQList, DMA_NodeTypeDef *const pNewNode)

Function description

Insert new node at the tail of linked-list queue.

Parameters

- **pQList:** : Pointer to a DMA_QListTypeDef structure that contains queue information.
- **pNewNode:** : Pointer to a DMA_NodeTypeDef structure that contains linked-list new node registers configurations.

Return values

- **HAL:** status.

HAL_DMAEx_List_RemoveNode

Function name

HAL_StatusTypeDef HAL_DMAEx_List_RemoveNode (DMA_QListTypeDef *const pQList, DMA_NodeTypeDef *const pNode)

Function description

Remove node from any linked-list queue position.

Parameters

- **pQList:** : Pointer to a DMA_QListTypeDef structure that contains queue information.
- **pNode:** : Pointer to a DMA_NodeTypeDef structure that contains linked-list previous node registers configurations.

Return values

- **HAL:** status.

HAL_DMAEx_List_RemoveNode_Head

Function name

HAL_StatusTypeDef HAL_DMAEx_List_RemoveNode_Head (DMA_QListTypeDef *const pQList)

Function description

Remove the head node from linked-list queue.

Parameters

- **pQList:** : Pointer to a DMA_QListTypeDef structure that contains queue information.

Return values

- **HAL:** status.

HAL_DMAEx_List_RemoveNode_Tail

Function name

HAL_StatusTypeDef HAL_DMAEx_List_RemoveNode_Tail (DMA_QListTypeDef *const pQList)

Function description

Remove the tail node from linked-list queue.

Parameters

- **pQList:** : Pointer to a DMA_QListTypeDef structure that contains queue information.

Return values

- **HAL:** status.

HAL_DMAEx_List_ReplaceNode

Function name

HAL_StatusTypeDef HAL_DMAEx_List_ReplaceNode (DMA_QListTypeDef *const pQList, DMA_NodeTypeDef *const pOldNode, DMA_NodeTypeDef *const pNewNode)

Function description

Replace node in linked-list queue.

Parameters

- **pQList:** : Pointer to a DMA_QListTypeDef structure that contains queue information.
- **pOldNode:** : Pointer to a DMA_NodeTypeDef structure that contains linked-list old node registers configurations.
- **pNewNode:** : Pointer to a DMA_NodeTypeDef structure that contains linked-list new node registers configurations.

Return values

- **HAL:** status.

HAL_DMAEx_List_ReplaceNode_Head

Function name

HAL_StatusTypeDef HAL_DMAEx_List_ReplaceNode_Head (DMA_QListTypeDef *const pQList, DMA_NodeTypeDef *const pNewNode)

Function description

Replace the head node of linked-list queue.

Parameters

- **pQList:** : Pointer to a DMA_QListTypeDef structure that contains queue information.
- **pNewNode:** : Pointer to a DMA_NodeTypeDef structure that contains linked-list new node registers configurations.

Return values

- **HAL:** status.

HAL_DMAEx_List_ReplaceNode_Tail

Function name

HAL_StatusTypeDef HAL_DMAEx_List_ReplaceNode_Tail (DMA_QListTypeDef *const pQList, DMA_NodeTypeDef *const pNewNode)

Function description

Replace the tail node of linked-list queue.

Parameters

- **pQList:** : Pointer to a DMA_QListTypeDef structure that contains queue information.
- **pNewNode:** : Pointer to a DMA_NodeTypeDef structure that contains linked-list new node registers configurations.

Return values

- **HAL:** status.

HAL_DMAEx_List_ResetQ

Function name

HAL_StatusTypeDef HAL_DMAEx_List_ResetQ (DMA_QListTypeDef *const pQList)

Function description

Reset the linked-list queue and unlink queue nodes.

Parameters

- **pQList:** : Pointer to a DMA_QListTypeDef structure that contains queue information.

Return values

- **HAL:** status.

HAL_DMAEx_List_InsertQ

Function name

HAL_StatusTypeDef HAL_DMAEx_List_InsertQ (DMA_QListTypeDef *const pSrcQList, DMA_NodeTypeDef const *const pPrevNode, DMA_QListTypeDef *const pDestQList)

Function description

Insert a source linked-list queue to a destination linked-list queue according to selecting previous node.

Parameters

- **pSrcQList:** : Pointer to a DMA_QListTypeDef structure that contains source queue information.
- **pPrevNode:** : Pointer to a DMA_NodeTypeDef structure that contains linked-list previous node registers configurations.
- **pDestQList:** : Pointer to a DMA_QListTypeDef structure that contains destination queue information.

Return values

- **HAL:** status.

HAL_DMAEx_List_InsertQ_Head

Function name

HAL_StatusTypeDef HAL_DMAEx_List_InsertQ_Head (DMA_QListTypeDef *const pSrcQList, DMA_QListTypeDef *const pDestQList)

Function description

Insert a source linked-list queue at the head of destination queue.

Parameters

- **pSrcQList:** : Pointer to a DMA_QListTypeDef structure that contains source queue information.
- **pDestQList:** : Pointer to a DMA_QListTypeDef structure that contains destination queue information.

Return values

- **HAL:** status.

HAL_DMAEx_List_InsertQ_Tail

Function name

HAL_StatusTypeDef HAL_DMAEx_List_InsertQ_Tail (DMA_QListTypeDef *const pSrcQList, DMA_QListTypeDef *const pDestQList)

Function description

Insert a source linked-list queue at the tail of destination queue.

Parameters

- **pSrcQList:** : Pointer to a DMA_QListTypeDef structure that contains source queue information.
- **pDestQList:** : Pointer to a DMA_QListTypeDef structure that contains destination queue information.

Return values

- **HAL:** status.

HAL_DMAEx_List_SetCircularModeConfig

Function name

HAL_StatusTypeDef HAL_DMAEx_List_SetCircularModeConfig (DMA_QListTypeDef *const pQList, DMA_NodeTypeDef *const pFirstCircularNode)

Function description

Set circular mode configuration for linked-list queue.

Parameters

- **pQList:** : Pointer to a DMA_QListTypeDef structure that contains queue information.
- **pFirstCircularNode:** : Pointer to a DMA_NodeTypeDef structure that contains linked-list first circular node registers configurations.

Return values

- **HAL:** status.

HAL_DMAEx_List_SetCircularMode

Function name

HAL_StatusTypeDef HAL_DMAEx_List_SetCircularMode (DMA_QListTypeDef *const pQList)

Function description

Set circular mode for linked-list queue.

Parameters

- **pQList:** : Pointer to a DMA_QListTypeDef structure that contains queue information.

Return values

- **HAL:** status.

HAL_DMAEx_List_ClearCircularMode

Function name

HAL_StatusTypeDef HAL_DMAEx_List_ClearCircularMode (DMA_QListTypeDef *const pQList)

Function description

Clear circular mode for linked-list queue.

Parameters

- **pQList:** : Pointer to a DMA_QListTypeDef structure that contains queue information.

Return values

- **HAL:** status.

HAL_DMAEx_List_ConvertQToDynamic

Function name

HAL_StatusTypeDef HAL_DMAEx_List_ConvertQToDynamic (DMA_QListTypeDef *const pQList)

Function description

Convert a linked-list queue to dynamic (Optimized DMA queue execution).

Parameters

- **pQList:** : Pointer to a DMA_QListTypeDef structure that contains queue information.

Return values

- **HAL:** status.

HAL_DMAEx_List_ConvertQToStatic

Function name

HAL_StatusTypeDef HAL_DMAEx_List_ConvertQToStatic (DMA_QListTypeDef *const pQList)

Function description

Convert a linked-list queue to static (Not optimized DMA queue execution).

Parameters

- **pQList:** : Pointer to a DMA_QListTypeDef structure that contains queue information.

Return values

- **HAL:** status.

HAL_DMAEx_List_LinkQ

Function name

HAL_StatusTypeDef HAL_DMAEx_List_LinkQ (DMA_HandleTypeDef *const hdma, DMA_QListTypeDef *const pQList)

Function description

Link linked-list queue to a DMA channel.

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
- **pQList:** : Pointer to a DMA_QListTypeDef structure that contains queue information.

Return values

- **HAL:** status.

HAL_DMAEx_List_UnLinkQ

Function name

HAL_StatusTypeDef HAL_DMAEx_List_UnLinkQ (DMA_HandleTypeDef *const hdma)

Function description

Unlink linked-list queue from a DMA channel.

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.

Return values

- **HAL:** status.

HAL_DMAEx_ConfigDataHandling

Function name

HAL_StatusTypeDef HAL_DMAEx_ConfigDataHandling (DMA_HandleTypeDef *const hdma, DMA_DataHandlingConfTypeDef const *const pConfigDataHandling)

Function description

Configure the DMA channel data handling according to the specified parameters in the DMA_DataHandlingConfTypeDef.

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
- **pConfigDataHandling:** : Pointer to a DMA_DataHandlingConfTypeDef structure that contains the data handling configuration.

Return values

- **HAL:** status.

HAL_DMAEx_ConfigTrigger

Function name

HAL_StatusTypeDef HAL_DMAEx_ConfigTrigger (DMA_HandleTypeDef *const hdma, DMA_TriggerConfTypeDef const *const pConfigTrigger)

Function description

Configure the DMA channel trigger according to the specified parameters in the DMA_TriggerConfTypeDef.

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
- **pConfigTrigger:** : Pointer to a DMA_TriggerConfTypeDef structure that contains the trigger configuration.

Return values

- **HAL:** status.

HAL_DMAEx_ConfigRepeatBlock

Function name

HAL_StatusTypeDef HAL_DMAEx_ConfigRepeatBlock (DMA_HandleTypeDef *const hdma, DMA_RepeatBlockConfTypeDef const *const pConfigRepeatBlock)

Function description

Configure the DMA channel repeated block according to the specified parameters in the DMA_RepeatBlockConfTypeDef.

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
- **pConfigRepeatBlock:** : Pointer to a DMA_RepeatBlockConfTypeDef structure that contains the repeated block configuration.

Return values

- **HAL:** status.

HAL_DMAEx_Suspend

Function name

HAL_StatusTypeDef HAL_DMAEx_Suspend (DMA_HandleTypeDef *const hdma)

Function description

Suspend any ongoing DMA channel transfer in polling mode (Blocking mode).

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA channel.

Return values

- **HAL:** status.

Notes

- After suspending a DMA channel, a check for wait until the DMA channel is effectively suspended is added. If a channel is suspended while a data transfer is ongoing, the current data will be transferred and the channel will be effectively suspended only after the transfer of this single/burst data is finished.

HAL_DMAEx_Suspend_IT

Function name

HAL_StatusTypeDef HAL_DMAEx_Suspend_IT (DMA_HandleTypeDef *const hdma)

Function description

Suspend any ongoing DMA channel transfer in polling mode (Non-blocking mode).

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.

Return values

- **HAL:** status.

HAL_DMAEx_Resume

Function name

HAL_StatusTypeDef HAL_DMAEx_Resume (DMA_HandleTypeDef *const hdma)

Function description

Resume any suspended DMA channel transfer.

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.

Return values

- **HAL:** status.

HAL_DMAEx_GetFifoLevel

Function name

uint32_t HAL_DMAEx_GetFifoLevel (DMA_HandleTypeDef const *const hdma)

Function description

Get and returns the DMA channel FIFO level.

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.

Return values

- **Returns:** the number of available beats in FIFO.

20.3 DMAEx Firmware driver defines

The following section lists the various define and macros of the module.

20.3.1 DMAEx

DMAEx

DMAEx Data Alignment

DMA_DATA_RIGHTALIGN_ZEROPADDED

If source data width < destination data width => Right aligned padded with 0 up to destination data width

DMA_DATA_RIGHTALIGN_LEFTTRUNC

If source data width > destination data width => Right aligned left Truncated down to destination data width

DMA_DATA_RIGHTALIGN_SIGNEXT

If source data width < destination data width => Right Aligned padded with sign extended up to destination data width

DMA_DATA_LEFTALIGN_RIGHTTRUNC

If source data width > destination data width => Left Aligned Right Truncated down to the destination data width

DMA_DATA_PACK

If source data width < destination data width => Packed at the destination data width

DMA_DATA_UNPACK

If source data width > destination data width => Unpacked at the destination data width

DMAEx Data Exchange

DMA_EXCHANGE_NONE

No data exchange

DMA_EXCHANGE_DEST_BYTE

Destination Byte exchange when destination data width is > Byte

DMA_EXCHANGE_DEST_HALFWORD

Destination Half-Word exchange when destination data width is > Half-Word

DMA_EXCHANGE_SRC_BYTE

Source Byte endianness exchange when source data width is word

DMA_EXCHANGE_DEST_WORD

Destination Word exchange when destination data width is > Word

DMAEx LinkedList Mode

DMA_LINKEDLIST_NORMAL

Linear linked-list DMA channel transfer

DMA_LINKEDLIST_CIRCULAR

Circular linked-list DMA channel transfer

DMAEx Linked-List Allocated Port

DMA_LINK_ALLOCATED_PORT0

Link allocated port 0

DMA_LINK_ALLOCATED_PORT1

Link allocated port 1

DMAEx Link Step Mode

DMA_LSM_FULL_EXECUTION

Channel is executed for the full linked-list

DMA_LSM_1LINK_EXECUTION

Channel is executed once for the current LLI

DMAEx Node Type

DMA_GPDMA_LINEAR_NODE

Defines the GPDMA linear addressing node type

DMA_GPDMA_2D_NODE

Defines the GPDMA 2 dimension addressing node type

DMA_HPDMA_LINEAR_NODE

Defines the HPDMA linear addressing node type

DMA_HPDMA_2D_NODE

Defines the HPDMA 2 dimension addressing node type

DMAEx Trigger Mode

DMA_TRIGM_BLOCK_TRANSFER

A block transfer is conditioned by (at least) one hit trigger

DMA_TRIGM_REPEATED_BLOCK_TRANSFER

A repeated block transfer is conditioned by (at least) one hit trigger

DMA_TRIGM_LLI_LINK_TRANSFER

A LLI link transfer is conditioned by (at least) one hit trigger

DMA_TRIGM_SINGLE_BURST_TRANSFER

A single/burst transfer is conditioned by (at least) one hit trigger

DMAEx Trigger Polarity

DMA_TRIG_POLARITY_MASKED

No trigger of the selected DMA request. Masked trigger event

DMA_TRIG_POLARITY_RISING

Trigger of the selected DMA request on the rising edge of the selected trigger event input

DMA_TRIG_POLARITY_FALLING

Trigger of the selected DMA request on the falling edge of the selected trigger event input

DMAEx Trigger Selection

HPDMA1_TRIGGER_DCMIPP_P0_FEND

HPDMA1 HW Trigger signal is DCMIPP_P0_FEND

HPDMA1_TRIGGER_DCMIPP_P0_LEND

HPDMA1 HW Trigger signal is DCMIPP_P0_LEND

HPDMA1_TRIGGER_DCMIPP_P0_HSYNC

HPDMA1 HW Trigger signal is DCMIPP_P0_HSYNC

HPDMA1_TRIGGER_DCMIPP_P0_VSYNC

HPDMA1 HW Trigger signal is DCMIPP_P0_VSYNC

HPDMA1_TRIGGER_DCMIPP_P1_FEND

HPDMA1 HW Trigger signal is DCMIPP_P1_FEND

HPDMA1_TRIGGER_DCMIPP_P1_LEND

HPDMA1 HW Trigger signal is DCMIPP_P1_LEND

HPDMA1_TRIGGER_DCMIPP_P1_HSYNC

HPDMA1 HW Trigger signal is DCMIPP_P1_HSYNC

HPDMA1_TRIGGER_DCMIPP_P1_VSYNC

HPDMA1 HW Trigger signal is DCMIPP_P1_VSYNC

HPDMA1_TRIGGER_DCMIPP_P2_FEND

HPDMA1 HW Trigger signal is DCMIPP_P2_FEND

HPDMA1_TRIGGER_DCMIPP_P2_LEND

HPDMA1 HW Trigger signal is DCMIPP_P2_LEND

HPDMA1_TRIGGER_DCMIPP_P2_HSYNC

HPDMA1 HW Trigger signal is DCMIPP_P2_HSYNC

HPDMA1_TRIGGER_DCMIPP_P2_VSYNC

HPDMA1 HW Trigger signal is DCMIPP_P2_VSYNC

HPDMA1_TRIGGER_DMA2D_CTC

HPDMA1 HW Trigger signal is DMA2D_CTC

HPDMA1_TRIGGER_DMA2D_TC

HPDMA1 HW Trigger signal is DMA2D_TC

HPDMA1_TRIGGER_DMA2D_TW

HPDMA1 HW Trigger signal is DMA2D_TW

HPDMA1_TRIGGER_JPEG_EOC

HPDMA1 HW Trigger signal is JPEG_EOC

HPDMA1_TRIGGER_JPEG_IFNF

HPDMA1 HW Trigger signal is JPEG_IFNF

HPDMA1_TRIGGER_JPEG_IFT

HPDMA1 HW Trigger signal is JPEG_IFT

HPDMA1_TRIGGER_JPEG_OFNE

HPDMA1 HW Trigger signal is JPEG_OFNE

HPDMA1_TRIGGER_JPEG_OFT

HPDMA1 HW Trigger signal is JPEG_OFT

HPDMA1_LCD_LI

HPDMA1 HW Trigger signal is LCD_LI

HPDMA1_TRIGGER_GPU2D1_GP_0

HPDMA1 HW Trigger signal is GPU2D1_GP_0

HPDMA1_TRIGGER_GPU2D1_GP_1

HPDMA1 HW Trigger signal is GPU2D1_GP_1

HPDMA1_TRIGGER_GPU2D1_GP_2

HPDMA1 HW Trigger signal is GPU2D1_GP_2

HPDMA1_TRIGGER_GPU2D1_GP_3

HPDMA1 HW Trigger signal is GPU2D1_GP_3

HPDMA1_TRIGGER_GFXTIM_3

HPDMA1 HW Trigger signal is GFXTIM_3

HPDMA1_TRIGGER_GFXTIM_2

HPDMA1 HW Trigger signal is GFXTIM_2

HPDMA1_TRIGGER_GFXTIM_1

HPDMA1 HW Trigger signal is GFXTIM_1

HPDMA1_TRIGGER_GFXTIM_0

HPDMA1 HW Trigger signal is GFXTIM_0

HPDMA1_TRIGGER_LPTIM1_CH1

HPDMA1 HW Trigger signal is LPTIM1_CH1

HPDMA1_TRIGGER_LPTIM1_CH2

HPDMA1 HW Trigger signal is LPTIM1_CH2

HPDMA1_TRIGGER_LPTIM2_CH1

HPDMA1 HW Trigger signal is LPTIM2_CH1

HPDMA1_TRIGGER_LPTIM2_CH2

HPDMA1 HW Trigger signal is LPTIM2_CH2

HPDMA1_TRIGGER_LPTIM3_CH1

HPDMA1 HW Trigger signal is LPTIM3_CH1

HPDMA1_TRIGGER_LPTIM3_CH2

HPDMA1 HW Trigger signal is LPTIM3_CH2

HPDMA1_TRIGGER_LPTIM4_OUT

HPDMA1 HW Trigger signal is LPTIM4_OUT

HPDMA1_TRIGGER_LPTIM5_OUT

HPDMA1 HW Trigger signal is LPTIM5_OUT

HPDMA1_TRIGGER_RTC_WKUP

HPDMA1 HW Trigger signal is RTC_WKUP

HPDMA1_TRIGGER_LPUART1_IT_R_WUP_ASYNC

HPDMA1 HW Trigger signal is IT_R_WUP_ASYNC

HPDMA1_TRIGGER_LPUART1_IT_T_WUP_ASYNC

HPDMA1 HW Trigger signal is IT_T_WUP_ASYNC

HPDMA1_TRIGGER_SPI6_IT_OR_SPI6_AIT_SYNC

HPDMA1 HW Trigger signal is SPI6_IT_OR_SPI6_AIT_SYNC

HPDMA1_TRIGGER_TIM1_TRGO

HPDMA1 HW Trigger signal is TIM1_TRGO

HPDMA1_TRIGGER_TIM1_TRGO2

HPDMA1 HW Trigger signal is TIM1_TRGO2

HPDMA1_TRIGGER_TIM2_TRGO

HPDMA1 HW Trigger signal is TIM2_TRGO

HPDMA1_TRIGGER_TIM3_TRGO

HPDMA1 HW Trigger signal is TIM3_TRGO

HPDMA1_TRIGGER_TIM4_TRGO

HPDMA1 HW Trigger signal is TIM4_TRGO

HPDMA1_TRIGGER_TIM5_TRGO

HPDMA1 HW Trigger signal is TIM5_TRGO

HPDMA1_TRIGGER_TIM6_TRGO

HPDMA1 HW Trigger signal is TIM6_TRGO

HPDMA1_TRIGGER_TIM7_TRGO

HPDMA1 HW Trigger signal is TIM7_TRGO

HPDMA1_TRIGGER_TIM8_TRGO

HPDMA1 HW Trigger signal is TIM8_TRGO

HPDMA1_TRIGGER_TIM8_TRGO2

HPDMA1 HW Trigger signal is TIM8_TRGO2

HPDMA1_TRIGGER_TIM12_TRGO

HPDMA1 HW Trigger signal is TIM12_TRGO

HPDMA1_TRIGGER_TIM15_TRGO

HPDMA1 HW Trigger signal is TIM15_TRGO

HPDMA1_TRIGGER_HPDMA1_CH0_TCF

HPDMA1 HW Trigger signal is HPDMA1_CH0_TCF

HPDMA1_TRIGGER_HPDMA1_CH1_TCF

HPDMA1 HW Trigger signal is HPDMA1_CH1_TCF

HPDMA1_TRIGGER_HPDMA1_CH2_TCF

HPDMA1 HW Trigger signal is HPDMA1_CH2_TCF

HPDMA1_TRIGGER_HPDMA1_CH3_TCF

HPDMA1 HW Trigger signal is HPDMA1_CH3_TCF

HPDMA1_TRIGGER_HPDMA1_CH4_TCF

HPDMA1 HW Trigger signal is HPDMA1_CH4_TCF

HPDMA1_TRIGGER_HPDMA1_CH5_TCF

HPDMA1 HW Trigger signal is HPDMA1_CH5_TCF

HPDMA1_TRIGGER_HPDMA1_CH6_TCF

HPDMA1 HW Trigger signal is HPDMA1_CH6_TCF

HPDMA1_TRIGGER_HPDMA1_CH7_TCF

HPDMA1 HW Trigger signal is HPDMA1_CH7_TCF

HPDMA1_TRIGGER_HPDMA1_CH8_TCF

HPDMA1 HW Trigger signal is HPDMA1_CH8_TCF

HPDMA1_TRIGGER_HPDMA1_CH9_TCF

HPDMA1 HW Trigger signal is HPDMA1_CH9_TCF

HPDMA1_TRIGGER_HPDMA1_CH10_TCF

HPDMA1 HW Trigger signal is HPDMA1_CH10_TCF

HPDMA1_TRIGGER_HPDMA1_CH11_TCF

HPDMA1 HW Trigger signal is HPDMA1_CH11_TCF

HPDMA1_TRIGGER_HPDMA1_CH12_TCF

HPDMA1 HW Trigger signal is HPDMA1_CH12_TCF

HPDMA1_TRIGGER_HPDMA1_CH13_TCF

HPDMA1 HW Trigger signal is HPDMA1_CH13_TCF

HPDMA1_TRIGGER_HPDMA1_CH14_TCF

HPDMA1 HW Trigger signal is HPDMA1_CH14_TCF

HPDMA1_TRIGGER_HPDMA1_CH15_TCF

HPDMA1 HW Trigger signal is HPDMA1_CH15_TCF

HPDMA1_TRIGGER_GPDMA1_CH0_TCF

HPDMA1 HW Trigger signal is GPDMA1_CH0_TCF

HPDMA1_TRIGGER_GPDMA1_CH1_TCF

HPDMA1 HW Trigger signal is GPDMA1_CH1_TCF

HPDMA1_TRIGGER_GPDMA1_CH2_TCF

HPDMA1 HW Trigger signal is GPDMA1_CH2_TCF

HPDMA1_TRIGGER_GPDMA1_CH3_TCF

HPDMA1 HW Trigger signal is GPDMA1_CH3_TCF

HPDMA1_TRIGGER_GPDMA1_CH4_TCF

HPDMA1 HW Trigger signal is GPDMA1_CH4_TCF

HPDMA1_TRIGGER_GPDMA1_CH5_TCF

HPDMA1 HW Trigger signal is GPDMA1_CH5_TCF

HPDMA1_TRIGGER_GPDMA1_CH6_TCF

HPDMA1 HW Trigger signal is GPDMA1_CH6_TCF

HPDMA1_TRIGGER_GPDMA1_CH7_TCF

HPDMA1 HW Trigger signal is GPDMA1_CH7_TCF

HPDMA1_TRIGGER_GPDMA1_CH8_TCF

HPDMA1 HW Trigger signal is GPDMA1_CH8_TCF

HPDMA1_TRIGGER_GPDMA1_CH9_TCF

HPDMA1 HW Trigger signal is GPDMA1_CH9_TCF

HPDMA1_TRIGGER_GPDMA1_CH10_TCF

HPDMA1 HW Trigger signal is GPDMA1_CH10_TCF

HPDMA1_TRIGGER_GPDMA1_CH11_TCF

HPDMA1 HW Trigger signal is GPDMA1_CH11_TCF

HPDMA1_TRIGGER_GPDMA1_CH12_TCF

HPDMA1 HW Trigger signal is GPDMA1_CH12_TCF

HPDMA1_TRIGGER_GPDMA1_CH13_TCF

HPDMA1 HW Trigger signal is GPDMA1_CH13_TCF

HPDMA1_TRIGGER_GPDMA1_CH14_TCF

HPDMA1 HW Trigger signal is GPDMA1_CH14_TCF

HPDMA1_TRIGGER_GPDMA1_CH15_TCF

HPDMA1 HW Trigger signal is GPDMA1_CH15_TCF

HPDMA1_TRIGGER_EXTIT0_SYNC

HPDMA1 HW Trigger signal is EXTIT0_SYNC

HPDMA1_TRIGGER_EXTIT1_SYNC

HPDMA1 HW Trigger signal is EXTIT1_SYNC

HPDMA1_TRIGGER_EXTIT2_SYNC

HPDMA1 HW Trigger signal is EXTIT2_SYNC

HPDMA1_TRIGGER_EXTIT3_SYNC

HPDMA1 HW Trigger signal is EXTIT3_SYNC

HPDMA1_TRIGGER_EXTIT4_SYNC

HPDMA1 HW Trigger signal is EXTIT4_SYNC

HPDMA1_TRIGGER_EXTIT5_SYNC

HPDMA1 HW Trigger signal is EXTIT5_SYNC

HPDMA1_TRIGGER_EXTIT6_SYNC

HPDMA1 HW Trigger signal is EXTIT6_SYNC

HPDMA1_TRIGGER_EXTIT7_SYNC

HPDMA1 HW Trigger signal is EXTIT7_SYNC

HPDMA1_TRIGGER_EXTIT8_SYNC

HPDMA1 HW Trigger signal is EXTIT8_SYNC

HPDMA1_TRIGGER_EXTIT9_SYNC

HPDMA1 HW Trigger signal is EXTIT9_SYNC

HPDMA1_TRIGGER_EXTIT10_SYNC

HPDMA1 HW Trigger signal is EXTIT10_SYNC

HPDMA1_TRIGGER_EXTIT11_SYNC

HPDMA1 HW Trigger signal is EXTIT11_SYNC

HPDMA1_TRIGGER_EXTIT12_SYNC

HPDMA1 HW Trigger signal is EXTIT12_SYNC

HPDMA1_TRIGGER_EXTIT13_SYNC

HPDMA1 HW Trigger signal is EXTIT13_SYNC

HPDMA1_TRIGGER_EXTIT14_SYNC

HPDMA1 HW Trigger signal is EXTIT14_SYNC

HPDMA1_TRIGGER_EXTIT15_SYNC

HPDMA1 HW Trigger signal is EXTIT15_SYNC

GPDMA1_TRIGGER_DCMIPP_P0_FEND

GPDMA1 HW Trigger signal is DCMIPP_P0_FEND

GPDMA1_TRIGGER_DCMIPP_P0_LEND

GPDMA1 HW Trigger signal is DCMIPP_P0_LEND

GPDMA1_TRIGGER_DCMIPP_P0_HSYNC

GPDMA1 HW Trigger signal is DCMIPP_P0_HSYNC

GPDMA1_TRIGGER_DCMIPP_P0_VSYNC

GPDMA1 HW Trigger signal is DCMIPP_P0_VSYNC

GPDMA1_TRIGGER_DCMIPP_P1_FEND

GPDMA1 HW Trigger signal is DCMIPP_P1_FEND

GPDMA1_TRIGGER_DCMIPP_P1_LEND

GPDMA1 HW Trigger signal is DCMIPP_P1_LEND

GPDMA1_TRIGGER_DCMIPP_P1_HSYNC

GPDMA1 HW Trigger signal is DCMIPP_P1_HSYNC

GPDMA1_TRIGGER_DCMIPP_P1_VSYNC

GPDMA1 HW Trigger signal is DCMIPP_P1_VSYNC

GPDMA1_TRIGGER_DCMIPP_P2_FEND

GPDMA1 HW Trigger signal is DCMIPP_P2_FEND

GPDMA1_TRIGGER_DCMIPP_P2_LEND

GPDMA1 HW Trigger signal is DCMIPP_P2_LEND

GPDMA1_TRIGGER_DCMIPP_P2_HSYNC

GPDMA1 HW Trigger signal is DCMIPP_P2_HSYNC

GPDMA1_TRIGGER_DCMIPP_P2_VSYNC

GPDMA1 HW Trigger signal is DCMIPP_P2_VSYNC

GPDMA1_TRIGGER_DMA2D_CTC

GPDMA1 HW Trigger signal is DMA2D_CTC

GPDMA1_TRIGGER_DMA2D_TC

GPDMA1 HW Trigger signal is DMA2D_TC

GPDMA1_TRIGGER_DMA2D_TW

GPDMA1 HW Trigger signal is DMA2D_TW

GPDMA1_TRIGGER_JPEG_EOC

GPDMA1 HW Trigger signal is JPEG_EOC

GPDMA1_TRIGGER_JPEG_IFNF

GPDMA1 HW Trigger signal is JPEG_IFNF

GPDMA1_TRIGGER_JPEG_IFT

GPDMA1 HW Trigger signal is JPEG_IFT

GPDMA1_TRIGGER_JPEG_OFNE

GPDMA1 HW Trigger signal is JPEG_OFNE

GPDMA1_TRIGGER_JPEG_OFT

GPDMA1 HW Trigger signal is JPEG_OFT

GPDMA1_LCD_LI

GPDMA1 HW Trigger signal is LCD_LI

GPDMA1_TRIGGER_GPU2D1_GP_0

GPDMA1 HW Trigger signal is GPU2D1_GP_0

GPDMA1_TRIGGER_GPU2D1_GP_1

GPDMA1 HW Trigger signal is GPU2D1_GP_1

GPDMA1_TRIGGER_GPU2D1_GP_2

GPDMA1 HW Trigger signal is GPU2D1_GP_2

GPDMA1_TRIGGER_GPU2D1_GP_3

GPDMA1 HW Trigger signal is GPU2D1_GP_3

GPDMA1_TRIGGER_GFXTIM_3

GPDMA1 HW Trigger signal is GFXTIM_3

GPDMA1_TRIGGER_GFXTIM_2

GPDMA1 HW Trigger signal is GFXTIM_2

GPDMA1_TRIGGER_GFXTIM_1

GPDMA1 HW Trigger signal is GFXTIM_1

GPDMA1_TRIGGER_GFXTIM_0

GPDMA1 HW Trigger signal is GFXTIM_0

GPDMA1_TRIGGER_LPTIM1_CH1

GPDMA1 HW Trigger signal is LPTIM1_CH1

GPDMA1_TRIGGER_LPTIM1_CH2

GPDMA1 HW Trigger signal is LPTIM1_CH2

GPDMA1_TRIGGER_LPTIM2_CH1

GPDMA1 HW Trigger signal is LPTIM2_CH1

GPDMA1_TRIGGER_LPTIM2_CH2

GPDMA1 HW Trigger signal is LPTIM2_CH2

GPDMA1_TRIGGER_LPTIM3_CH1

GPDMA1 HW Trigger signal is LPTIM3_CH1

GPDMA1_TRIGGER_LPTIM3_CH2

GPDMA1 HW Trigger signal is LPTIM3_CH2

GPDMA1_TRIGGER_LPTIM4_OUT

GPDMA1 HW Trigger signal is LPTIM4_OUT

GPDMA1_TRIGGER_LPTIM5_OUT

GPDMA1 HW Trigger signal is LPTIM5_OUT

GPDMA1_TRIGGER_RTC_WKUP

GPDMA1 HW Trigger signal is RTC_WKUP

GPDMA1_TRIGGER_LPUART1_IT_R_WUP_ASYNC

GPDMA1 HW Trigger signal is IT_R_WUP_ASYNC

GPDMA1_TRIGGER_LPUART1_IT_T_WUP_ASYNC

GPDMA1 HW Trigger signal is IT_T_WUP_ASYNC

GPDMA1_TRIGGER_SPI6_IT_OR_SPI6_AIT_SYNC

GPDMA1 HW Trigger signal is SPI6_IT_OR_SPI6_AIT_SYNC

GPDMA1_TRIGGER_TIM1_TRGO

GPDMA1 HW Trigger signal is TIM1_TRGO

GPDMA1_TRIGGER_TIM1_TRGO2

GPDMA1 HW Trigger signal is TIM1_TRGO2

GPDMA1_TRIGGER_TIM2_TRGO

GPDMA1 HW Trigger signal is TIM2_TRGO

GPDMA1_TRIGGER_TIM3_TRGO

GPDMA1 HW Trigger signal is TIM3_TRGO

GPDMA1_TRIGGER_TIM4_TRGO

GPDMA1 HW Trigger signal is TIM4_TRGO

GPDMA1_TRIGGER_TIM5_TRGO

GPDMA1 HW Trigger signal is TIM5_TRGO

GPDMA1_TRIGGER_TIM6_TRGO

GPDMA1 HW Trigger signal is TIM6_TRGO

GPDMA1_TRIGGER_TIM7_TRGO

GPDMA1 HW Trigger signal is TIM7_TRGO

GPDMA1_TRIGGER_TIM8_TRGO

GPDMA1 HW Trigger signal is TIM8_TRGO

GPDMA1_TRIGGER_TIM8_TRGO2

GPDMA1 HW Trigger signal is TIM8_TRGO2

GPDMA1_TRIGGER_TIM12_TRGO

GPDMA1 HW Trigger signal is TIM12_TRGO

GPDMA1_TRIGGER_TIM15_TRGO

GPDMA1 HW Trigger signal is TIM15_TRGO

GPDMA1_TRIGGER_HPDMA1_CH0_TCF

GPDMA1 HW Trigger signal is HPDMA1_CH0_TCF

GPDMA1_TRIGGER_HPDMA1_CH1_TCF

GPDMA1 HW Trigger signal is HPDMA1_CH1_TCF

GPDMA1_TRIGGER_HPDMA1_CH2_TCF

GPDMA1 HW Trigger signal is HPDMA1_CH2_TCF

GPDMA1_TRIGGER_HPDMA1_CH3_TCF

GPDMA1 HW Trigger signal is HPDMA1_CH3_TCF

GPDMA1_TRIGGER_HPDMA1_CH4_TCF

GPDMA1 HW Trigger signal is HPDMA1_CH4_TCF

GPDMA1_TRIGGER_HPDMA1_CH5_TCF

GPDMA1 HW Trigger signal is HPDMA1_CH5_TCF

GPDMA1_TRIGGER_HPDMA1_CH6_TCF

GPDMA1 HW Trigger signal is HPDMA1_CH6_TCF

GPDMA1_TRIGGER_HPDMA1_CH7_TCF

GPDMA1 HW Trigger signal is HPDMA1_CH7_TCF

GPDMA1_TRIGGER_HPDMA1_CH8_TCF

GPDMA1 HW Trigger signal is HPDMA1_CH8_TCF

GPDMA1_TRIGGER_HPDMA1_CH9_TCF

GPDMA1 HW Trigger signal is HPDMA1_CH9_TCF

GPDMA1_TRIGGER_HPDMA1_CH10_TCF

GPDMA1 HW Trigger signal is HPDMA1_CH10_TCF

GPDMA1_TRIGGER_HPDMA1_CH11_TCF

GPDMA1 HW Trigger signal is HPDMA1_CH11_TCF

GPDMA1_TRIGGER_HPDMA1_CH12_TCF

GPDMA1 HW Trigger signal is HPDMA1_CH12_TCF

GPDMA1_TRIGGER_HPDMA1_CH13_TCF

GPDMA1 HW Trigger signal is HPDMA1_CH13_TCF

GPDMA1_TRIGGER_HPDMA1_CH14_TCF

GPDMA1 HW Trigger signal is HPDMA1_CH14_TCF

GPDMA1_TRIGGER_HPDMA1_CH15_TCF

GPDMA1 HW Trigger signal is HPDMA1_CH15_TCF

GPDMA1_TRIGGER_GPDMA1_CH0_TCF

GPDMA1 HW Trigger signal is GPDMA1_CH0_TCF

GPDMA1_TRIGGER_GPDMA1_CH1_TCF

GPDMA1 HW Trigger signal is GPDMA1_CH1_TCF

GPDMA1_TRIGGER_GPDMA1_CH2_TCF

GPDMA1 HW Trigger signal is GPDMA1_CH2_TCF

GPDMA1_TRIGGER_GPDMA1_CH3_TCF

GPDMA1 HW Trigger signal is GPDMA1_CH3_TCF

GPDMA1_TRIGGER_GPDMA1_CH4_TCF

GPDMA1 HW Trigger signal is GPDMA1_CH4_TCF

GPDMA1_TRIGGER_GPDMA1_CH5_TCF

GPDMA1 HW Trigger signal is GPDMA1_CH5_TCF

GPDMA1_TRIGGER_GPDMA1_CH6_TCF

GPDMA1 HW Trigger signal is GPDMA1_CH6_TCF

GPDMA1_TRIGGER_GPDMA1_CH7_TCF

GPDMA1 HW Trigger signal is GPDMA1_CH7_TCF

GPDMA1_TRIGGER_GPDMA1_CH8_TCF

GPDMA1 HW Trigger signal is GPDMA1_CH8_TCF

GPDMA1_TRIGGER_GPDMA1_CH9_TCF

GPDMA1 HW Trigger signal is GPDMA1_CH9_TCF

GPDMA1_TRIGGER_GPDMA1_CH10_TCF

GPDMA1 HW Trigger signal is GPDMA1_CH10_TCF

GPDMA1_TRIGGER_GPDMA1_CH11_TCF

GPDMA1 HW Trigger signal is GPDMA1_CH11_TCF

GPDMA1_TRIGGER_GPDMA1_CH12_TCF

GPDMA1 HW Trigger signal is GPDMA1_CH12_TCF

GPDMA1_TRIGGER_GPDMA1_CH13_TCF

GPDMA1 HW Trigger signal is GPDMA1_CH13_TCF

GPDMA1_TRIGGER_GPDMA1_CH14_TCF

GPDMA1 HW Trigger signal is GPDMA1_CH14_TCF

GPDMA1_TRIGGER_GPDMA1_CH15_TCF

GPDMA1 HW Trigger signal is GPDMA1_CH15_TCF

GPDMA1_TRIGGER_EXTIT0_SYNC

GPDMA1 HW Trigger signal is EXTIT0_SYNC

GPDMA1_TRIGGER_EXTIT1_SYNC

GPDMA1 HW Trigger signal is EXTIT1_SYNC

GPDMA1_TRIGGER_EXTIT2_SYNC

GPDMA1 HW Trigger signal is EXTIT2_SYNC

GPDMA1_TRIGGER_EXTIT3_SYNC

GPDMA1 HW Trigger signal is EXTIT3_SYNC

GPDMA1_TRIGGER_EXTIT4_SYNC

GPDMA1 HW Trigger signal is EXTIT4_SYNC

GPDMA1_TRIGGER_EXTIT5_SYNC

GPDMA1 HW Trigger signal is EXTIT5_SYNC

GPDMA1_TRIGGER_EXTIT6_SYNC

GPDMA1 HW Trigger signal is EXTIT6_SYNC

GPDMA1_TRIGGER_EXTIT7_SYNC

GPDMA1 HW Trigger signal is EXTIT7_SYNC

GPDMA1_TRIGGER_EXTIT8_SYNC

GPDMA1 HW Trigger signal is EXTIT8_SYNC

GPDMA1_TRIGGER_EXTIT9_SYNC

GPDMA1 HW Trigger signal is EXTIT9_SYNC

GPDMA1_TRIGGER_EXTIT10_SYNC

GPDMA1 HW Trigger signal is EXTIT10_SYNC

GPDMA1_TRIGGER_EXTIT11_SYNC

GPDMA1 HW Trigger signal is EXTIT11_SYNC

GPDMA1_TRIGGER_EXTIT12_SYNC

GPDMA1 HW Trigger signal is EXTIT12_SYNC

GPDMA1_TRIGGER_EXTIT13_SYNC

GPDMA1 HW Trigger signal is EXTIT13_SYNC

GPDMA1_TRIGGER_EXTIT14_SYNC

GPDMA1 HW Trigger signal is EXTIT14_SYNC

GPDMA1_TRIGGER_EXTIT15_SYNC

GPDMA1 HW Trigger signal is EXTIT15_SYNC

21 HAL DTS Generic Driver

21.1 DTS Firmware driver registers structures

21.1.1 __DTS_HandleTypeDef

__DTS_HandleTypeDef is defined in the stm32n6xx_hal_dts.h

Data Fields

- ***DTS_TypeDef * Instance***
- ***__IO HAL_DTS_StateTypeDef State***
- ***__IO uint32_t ErrorCode***
- ***uint32_t SensorMode***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***
- ***void(* TemperatureCallback***
- ***void(* AlarmACallback***
- ***void(* AlarmBCallback***
- ***void(* ErrorCallback***

Field Documentation

- ***DTS_TypeDef* __DTS_HandleTypeDef::Instance***
  DTS instance
- ***__IO HAL_DTS_StateTypeDef __DTS_HandleTypeDef::State***
  DTS state
- ***__IO uint32_t __DTS_HandleTypeDef::ErrorCode***
  DTS error code
- ***uint32_t __DTS_HandleTypeDef::SensorMode[2U]***
  DTS sensor modes
- ***void(* __DTS_HandleTypeDef::MspInitCallback)(struct __DTS_HandleTypeDef *hdts)***
  DTS MSP init callback
- ***void(* __DTS_HandleTypeDef::MspDeInitCallback)(struct __DTS_HandleTypeDef *hdts)***
  DTS MSP de-init callback
- ***void(* __DTS_HandleTypeDef::TemperatureCallback)(struct __DTS_HandleTypeDef *hdts, HAL_DTS_Sensor Sensor)***
  DTS temperature callback
- ***void(* __DTS_HandleTypeDef::AlarmACallback)(struct __DTS_HandleTypeDef *hdts, HAL_DTS_Sensor Sensor)***
  DTS alarm A callback
- ***void(* __DTS_HandleTypeDef::AlarmBCallback)(struct __DTS_HandleTypeDef *hdts, HAL_DTS_Sensor Sensor)***
  DTS alarm B callback
- ***void(* __DTS_HandleTypeDef::ErrorCallback)(struct __DTS_HandleTypeDef *hdts, HAL_DTS_Sensor Sensor)***
  DTS error callback

21.1.2 DTS_SensorConfigTypeDef

DTS_SensorConfigTypeDef is defined in the stm32n6xx_hal_dts.h

Data Fields

- ***uint32_t Mode***
- ***uint32_t Resolution***
- ***uint32_t Trigger***

Field Documentation

- ***uint32_t DTS_SensorConfigTypeDef::Mode***
  Sensor mode. This parameter can be a value of ***DTS_SensorMode***
- ***uint32_t DTS_SensorConfigTypeDef::Resolution***
  Sensor resolution. This parameter can be a value of ***DTS_SensorResolution***
- ***uint32_t DTS_SensorConfigTypeDef::Trigger***
  Sensor trigger. This parameter can be a value of ***DTS_SensorTrigger***.
  Note:
  – This parameter is used only if Mode is set to DTS_SENSOR_MODE_TRIGGER

21.1.3 DTS_AlarmConfigTypeDef

DTS_AlarmConfigTypeDef is defined in the stm32n6xx_hal_dts.h

Data Fields

- ***FunctionalState Enable***
- ***float_t Threshold***
- ***float_t Hysteresis***

Field Documentation

- ***FunctionalState DTS_AlarmConfigTypeDef::Enable***
  Alarm enable/disable
- ***float_t DTS_AlarmConfigTypeDef::Threshold***
  Alarm threshold in celsius degree. This parameter must be a number between Min_Data = -40 and Max_Data = 125
- ***float_t DTS_AlarmConfigTypeDef::Hysteresis***
  Alarm hysteresis in celsius degree. This parameter must be a number between Min_Data = -40 and Max_Data = 125

21.2 DTS Firmware driver API description

The following section lists the various functions of the DTS library.

21.2.1 How to use this driver

The DTS hardware IP is a high-precision low-power junction temperature sensor (TS). It is composed of a configurable controller plus two embedded temperature sensors. In addition of temperature measurements, we have following features for each temperature sensor:

- Two programmable alarms incorporating hysteresis.
- Recording of the minimum and maximum temperatures measured. A sample counter is also available.

Initialization and de-initialization

1. User has first to initialize DTS instance.
2. As prerequisite, fill in the HAL_DTS_MspInit():
   – Enable DTS clock interface with __HAL_RCC_DTS_CLK_ENABLE().
   – If interrupt mode is used, enable and configure DTS interrupt with HAL_NVIC_SetPriority() and HAL_NVIC_EnableIRQ().
3. Call HAL_DTS_Init() function.
1. User can de-initialize DTS instance with HAL_DTS_DeInit() function.

Temperature measurement

1. Configure senors using HAL_DTS_ConfigSensor() for each sensor.
2. In polling mode:
   – Use HAL_DTS_Start() to start temperature measurement.
   – Use HAL_DTS_PollForTemperature() to detect the end of measurement. Use HAL_DTS_GetTemperature to get measured temperature.

3. In interrupt mode:
   – Use HAL_DTS_Start_IT() to start temperature measurement.
   – HAL_DTS_TemperatureCallback() will be called at the end of measurement. Use HAL_DTS_GetTemperature to get measured temperature.
4. Use HAL_DTS_GetExtremeTemperatures() to get the extreme measured temperatures.
5. Stop measurement using HAL_DTS_Stop() or HAL_DTS_Stop_IT().
6. Configure sample counter using HAL_DTS_ConfigSampleCounter().
7. Get sample counter value using HAL_DTS_GetSampleCounter().
8. Activate sample discard using HAL_DTS_ConfigSampleDiscard().

Alarms usage

1. Configure alarms using HAL_DTS_ConfigAlarmX() for each sensor (X is A or B). (#] HAL_DTS_AlarmXCallback() will be called when alarm X occurs (X is A or B).
2. Disable alarms using HAL_DTS_ConfigAlarmX() (X is A or B).

generic functions

1. HAL_DTS_IRQHandler will be called when DTS interrupt occurs.
2. HAL_DTS_ErrorCallback will be called when DTS error occurs.
3. Use HAL_DTS_GetState() to get the current DTS instance state.

Callback registration

The compilation define USE_HAL_DTS_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use functions HAL_DTS_RegisterCallback() to register a user callback.

Functions HAL_DTS_RegisterCallback() and HAL_DTS_UnRegisterCallback allow to register and unregister following callbacks:

- MspInitCallback : MSP init callback.
- MspDeInitCallback : MSP de-init callback.

For temperature measurement complete callback use dedicated register and unregister callback functions:

- HAL_DTS_RegisterTemperatureCallback().
- HAL_DTS_UnRegisterTemperatureCallback().

For alarms callback use dedicated register and unregister callback functions:

- HAL_DTS_RegisterAlarmACallback().
- HAL_DTS_RegisterAlarmBCallback().
- HAL_DTS_UnRegisterAlarmACallback().
- HAL_DTS_UnRegisterAlarmBCallback().

For error callback use dedicated register and unregister callback functions:

- HAL_DTS_RegisterErrorCallback().
- HAL_DTS_UnRegisterErrorCallback().

By default, after the call of init function and if the state is RESET all callbacks are reset to the corresponding legacy weak functions. Exception done for MspInit and MspDeInit callbacks that are respectively reset to the legacy weak functions in the init and de-init only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the init and de-init keep and use the user MspInit/MspDeInit callbacks (registered beforehand).

Callbacks can be registered/unregistered in READY state only. Exception done for MspInit/MspDeInit callbacks that can be registered/unregistered in READY or RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the init/de-init. In that case first register the MspInit/MspDeInit user callbacks using HAL_DTS_RegisterCallback() before calling init or de-init function.

When the compilation define USE_HAL_DTS_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak callbacks are used.

21.2.2 Detailed description of functions

HAL_DTS_Init

Function name

HAL_StatusTypeDef HAL_DTS_Init (DTS_HandleTypeDef * hdts)

Function description

HAL_DTS_DeInit

Function name

HAL_StatusTypeDef HAL_DTS_DeInit (DTS_HandleTypeDef * hdts)

Function description

HAL_DTS_MspInit

Function name

void HAL_DTS_MspInit (DTS_HandleTypeDef * hdts)

Function description

HAL_DTS_MspDeInit

Function name

void HAL_DTS_MspDeInit (DTS_HandleTypeDef * hdts)

Function description

HAL_DTS_RegisterCallback

Function name

**HAL_StatusTypeDef HAL_DTS_RegisterCallback (DTS_HandleTypeDef * hdts,
HAL_DTS_CallbackIDTypeDef CallbackID, pDTS_CallbackTypeDef pCallback)**

Function description

HAL_DTS_UnRegisterCallback

Function name

**HAL_StatusTypeDef HAL_DTS_UnRegisterCallback (DTS_HandleTypeDef * hdts,
HAL_DTS_CallbackIDTypeDef CallbackID)**

Function description

HAL_DTS_RegisterSensorCallback

Function name

**HAL_StatusTypeDef HAL_DTS_RegisterSensorCallback (DTS_HandleTypeDef * hdts,
HAL_DTS_SensorCallbackIDTypeDef CallbackID, pDTS_SensorCallbackTypeDef pCallback)**

Function description

HAL_DTS_UnRegisterSensorCallback

Function name

HAL_StatusTypeDef HAL_DTS_UnRegisterSensorCallback (DTS_HandleTypeDef * hdts, HAL_DTS_SensorCallbackIDTypeDef CallbackID)

Function description

HAL_DTS_ConfigSensor

Function name

HAL_StatusTypeDef HAL_DTS_ConfigSensor (DTS_HandleTypeDef * hdts, HAL_DTS_Sensor Sensor, DTS_SensorConfigTypeDef * pSensorParams)

Function description

HAL_DTS_Start

Function name

HAL_StatusTypeDef HAL_DTS_Start (DTS_HandleTypeDef * hdts, HAL_DTS_Sensor Sensor)

Function description

HAL_DTS_PollForTemperature

Function name

HAL_StatusTypeDef HAL_DTS_PollForTemperature (DTS_HandleTypeDef * hdts, HAL_DTS_Sensor Sensor, uint32_t Timeout)

Function description

HAL_DTS_GetTemperature

Function name

HAL_StatusTypeDef HAL_DTS_GetTemperature (DTS_HandleTypeDef * hdts, HAL_DTS_Sensor Sensor, float_t * pTemperature)

Function description

HAL_DTS_Stop

Function name

HAL_StatusTypeDef HAL_DTS_Stop (DTS_HandleTypeDef * hdts, HAL_DTS_Sensor Sensor)

Function description

HAL_DTS_Start_IT

Function name

HAL_StatusTypeDef HAL_DTS_Start_IT (DTS_HandleTypeDef * hdts, HAL_DTS_Sensor Sensor)

Function description

HAL_DTS_Stop_IT

Function name

HAL_StatusTypeDef HAL_DTS_Stop_IT (DTS_HandleTypeDef * hdts, HAL_DTS_Sensor Sensor)

Function description

HAL_DTS_TemperatureCallback

Function name

void HAL_DTS_TemperatureCallback (DTS_HandleTypeDef * hdts, HAL_DTS_Sensor Sensor)

Function description

HAL_DTS_GetExtremeTemperatures

Function name

HAL_StatusTypeDef HAL_DTS_GetExtremeTemperatures (DTS_HandleTypeDef * hdts, HAL_DTS_Sensor Sensor, float_t * pMin, float_t * pMax)

Function description

HAL_DTS_ConfigSampleCounter

Function name

HAL_StatusTypeDef HAL_DTS_ConfigSampleCounter (DTS_HandleTypeDef * hdts, FunctionalState Enable, FunctionalState Clear, FunctionalState Hold)

Function description

HAL_DTS_GetSampleCounterValue

Function name

HAL_StatusTypeDef HAL_DTS_GetSampleCounterValue (DTS_HandleTypeDef * hdts, uint32_t * pNumber)

Function description

HAL_DTS_ConfigSampleDiscard

Function name

HAL_StatusTypeDef HAL_DTS_ConfigSampleDiscard (DTS_HandleTypeDef * hdts, FunctionalState Status)

Function description

HAL_DTS_ConfigAlarmA

Function name

HAL_StatusTypeDef HAL_DTS_ConfigAlarmA (DTS_HandleTypeDef * hdts, HAL_DTS_Sensor Sensor, const DTS_AlarmConfigTypeDef * pAlarmParams)

Function description

HAL_DTS_ConfigAlarmB

Function name

HAL_StatusTypeDef HAL_DTS_ConfigAlarmB (DTS_HandleTypeDef * hdts, HAL_DTS_Sensor Sensor, const DTS_AlarmConfigTypeDef * pAlarmParams)

Function description

HAL_DTS_AlarmACallback

Function name

void HAL_DTS_AlarmACallback (DTS_HandleTypeDef * hdts, HAL_DTS_Sensor Sensor)

Function description

HAL_DTS_AlarmBCallback

Function name

void HAL_DTS_AlarmBCallback (DTS_HandleTypeDef * hdts, HAL_DTS_Sensor Sensor)

Function description

HAL_DTS_IRQHandler

Function name

void HAL_DTS_IRQHandler (DTS_HandleTypeDef * hdts)

Function description

HAL_DTS_ErrorCallback

Function name

void HAL_DTS_ErrorCallback (DTS_HandleTypeDef * hdts, HAL_DTS_Sensor Sensor)

Function description

HAL_DTS_GetState

Function name

HAL_DTS_StateTypeDef HAL_DTS_GetState (const DTS_HandleTypeDef * hdts)

Function description

HAL_DTS_GetError

Function name

uint32_t HAL_DTS_GetError (const DTS_HandleTypeDef * hdts)

Function description

21.3 DTS Firmware driver defines

The following section lists the various define and macros of the module.

21.3.1 DTS

DTS

DTS error code

HAL_DTS_ERROR_NONE

No error

HAL_DTS_ERROR_INVALID_PARAM

Invalid parameter

HAL_DTS_ERROR_INVALID_STATE

Invalid state

HAL_DTS_ERROR_INVALID_SENSOR_MODE

Invalid sensor mode

HAL_DTS_ERROR_INVALID_SAMPLE

Invalid sample

HAL_DTS_ERROR_FAULT

Fault

HAL_DTS_ERROR_INVALID_CALLBACK

Invalid callback error

DTS Exported Macros

__HAL_DTS_RESET_HANDLE_STATE

IS_DTS_SENSOR_MODE

IS_DTS_SENSOR_RESOLUTION

IS_DTS_SENSOR_TRIGGER

IS_DTS_ALARM_PARAM

DTS sensor mode

DTS_SENSOR_MODE_DISABLE

No acquisition

DTS_SENSOR_MODE_SINGLE

Single acquisition

DTS_SENSOR_MODE_CONTINUOUS

Continuous acquisitions

DTS_SENSOR_MODE_TRIGGER

Acquisition with trigger

DTS sensor resolution

DTS_SENSOR_RESOLUTION_12BITS

Sensor resolution of 12 bits

DTS_SENSOR_RESOLUTION_10BITS

Sensor resolution of 10 bits

DTS_SENSOR_RESOLUTION_8BITS

Sensor resolution of 8 bits

DTS sensor trigger

DTS_SENSOR_TRIGGER_LPTIM4_OUT

LPTIM4 output signal

DTS_SENSOR_TRIGGER_LPTIM2_CH1

LPTIM2 channel 1 signal

DTS_SENSOR_TRIGGER_LPTIM3_CH1

LPTIM3 channel 1 signal

DTS_SENSOR_TRIGGER_EXTI13

EXTI13 signal

22 HAL ETH Generic Driver

22.1 ETH Firmware driver registers structures

22.1.1 ETH_DMADescTypeDef

ETH_DMADescTypeDef is defined in the stm32n6xx_hal_eth.h

Data Fields

- ***__IO uint32_t DESC0***
- ***__IO uint32_t DESC1***
- ***__IO uint32_t DESC2***
- ***__IO uint32_t DESC3***
- ***uint32_t BackupAddr0***
- ***uint32_t BackupAddr1***

Field Documentation

- ***__IO uint32_t ETH_DMADescTypeDef::DESC0***
- ***__IO uint32_t ETH_DMADescTypeDef::DESC1***
- ***__IO uint32_t ETH_DMADescTypeDef::DESC2***
- ***__IO uint32_t ETH_DMADescTypeDef::DESC3***
- ***uint32_t ETH_DMADescTypeDef::BackupAddr0***
- ***uint32_t ETH_DMADescTypeDef::BackupAddr1***

22.1.2 __ETH_BufferTypeDef

__ETH_BufferTypeDef is defined in the stm32n6xx_hal_eth.h

Data Fields

- ***uint8_t * buffer***
- ***uint32_t len***
- ***struct __ETH_BufferTypeDef * next***

Field Documentation

- ***uint8_t* __ETH_BufferTypeDef::buffer***
- ***uint32_t __ETH_BufferTypeDef::len***
- ***struct __ETH_BufferTypeDef* __ETH_BufferTypeDef::next***

22.1.3 ETH_TxDescListTypeDef

ETH_TxDescListTypeDef is defined in the stm32n6xx_hal_eth.h

Data Fields

- ***uint32_t TxDesc***
- ***uint32_t CurTxDesc***
- ***uint32_t * PacketAddress***
- ***uint32_t * CurrentPacketAddress***
- ***uint32_t BuffersInUse***
- ***uint32_t releaseIndex***

Field Documentation

- ***uint32_t ETH_TxDescListTypeDef::TxDesc[ETH_TX_DESC_CNT]***
- ***uint32_t ETH_TxDescListTypeDef::CurTxDesc***
- ***uint32_t* ETH_TxDescListTypeDef::PacketAddress[ETH_TX_DESC_CNT]***
- ***uint32_t* ETH_TxDescListTypeDef::CurrentPacketAddress***
- ***uint32_t ETH_TxDescListTypeDef::BuffersInUse***
- ***uint32_t ETH_TxDescListTypeDef::releaseIndex***

22.1.4 ETH_TxPacketConfigTypeDef

ETH_TxPacketConfigTypeDef is defined in the stm32n6xx_hal_eth.h

Data Fields

- ***uint32_t TxDMACh***
- ***uint32_t Attributes***
- ***uint32_t Length***
- ***ETH_BufferTypeDef * TxBuffer***
- ***uint32_t SrcAddrCtrl***
- ***uint32_t CRCPadCtrl***
- ***uint32_t ChecksumCtrl***
- ***uint32_t MaxSegmentSize***
- ***uint32_t PayloadLen***
- ***uint32_t TCPHeaderLen***
- ***uint32_t VlanTag***
- ***uint32_t VlanCtrl***
- ***uint32_t InnerVlanTag***
- ***uint32_t InnerVlanCtrl***
- ***void * pData***

Field Documentation

- ***uint32_t ETH_TxPacketConfigTypeDef::TxDMACh***
  Sets the Tx Queue/Channel which will transmit the packet
- ***uint32_t ETH_TxPacketConfigTypeDef::Attributes***
  Tx packet HW features capabilities. This parameter can be a combination of ***ETH_Tx_Packet_Attributes***
- ***uint32_t ETH_TxPacketConfigTypeDef::Length***
  Total packet length
- ***ETH_BufferTypeDef* ETH_TxPacketConfigTypeDef::TxBuffer***
  Tx buffers pointers
- ***uint32_t ETH_TxPacketConfigTypeDef::SrcAddrCtrl***
  Specifies the source address insertion control. This parameter can be a value of ***ETH_Tx_Packet_Source_Addr_Control***
- ***uint32_t ETH_TxPacketConfigTypeDef::CRCPadCtrl***
  Specifies the CRC and Pad insertion and replacement control. This parameter can be a value of ***ETH_Tx_Packet_CRC_Pad_Control***
- ***uint32_t ETH_TxPacketConfigTypeDef::ChecksumCtrl***
  Specifies the checksum insertion control. This parameter can be a value of ***ETH_Tx_Packet_Checksum_Control***
- ***uint32_t ETH_TxPacketConfigTypeDef::MaxSegmentSize***
  Sets TCP maximum segment size only when TCP segmentation is enabled. This parameter can be a value from 0x0 to 0x3FFF
- ***uint32_t ETH_TxPacketConfigTypeDef::PayloadLen***
  Sets Total payload length only when TCP segmentation is enabled. This parameter can be a value from 0x0 to 0x3FFFF
- ***uint32_t ETH_TxPacketConfigTypeDef::TCPHeaderLen***
  Sets TCP header length only when TCP segmentation is enabled. This parameter can be a value from 0x5 to 0xF
- ***uint32_t ETH_TxPacketConfigTypeDef::VlanTag***
  Sets VLAN Tag only when VLAN is enabled. This parameter can be a value from 0x0 to 0xFFFF
- ***uint32_t ETH_TxPacketConfigTypeDef::VlanCtrl***
  Specifies VLAN Tag insertion control only when VLAN is enabled. This parameter can be a value of ***ETH_Tx_Packet_VLAN_Control***
- ***uint32_t ETH_TxPacketConfigTypeDef::InnerVlanTag***
  Sets Inner VLAN Tag only when Inner VLAN is enabled. This parameter can be a value from 0x0 to 0x3FFFF

- ***uint32_t ETH_TxPacketConfigTypeDef::InnerVlanCtrl***
  Specifies Inner VLAN Tag insertion control only when Inner VLAN is enabled. This parameter can be a value of ***ETH_Tx_Packet_Inner_VLAN_Control***
- ***void* ETH_TxPacketConfigTypeDef::pData***
  Specifies Application packet pointer to save

22.1.5 ETH_TimeStampTypeDef

ETH_TimeStampTypeDef is defined in the stm32n6xx_hal_eth.h

Data Fields

- ***uint32_t TimeStampLow***
- ***uint32_t TimeStampHigh***

Field Documentation

- ***uint32_t ETH_TimeStampTypeDef::TimeStampLow***
- ***uint32_t ETH_TimeStampTypeDef::TimeStampHigh***

22.1.6 ETH_RxDescListTypeDef

ETH_RxDescListTypeDef is defined in the stm32n6xx_hal_eth.h

Data Fields

- ***uint32_t RxDesc***
- ***uint32_t ItMode***
- ***uint32_t RxDescIdx***
- ***uint32_t RxDescCnt***
- ***uint32_t RxDataLength***
- ***uint32_t RxBuildDescIdx***
- ***uint32_t RxBuildDescCnt***
- ***uint32_t pRxLastRxDesc***
- ***ETH_TimeStampTypeDef TimeStamp***
- ***void * pRxStart***
- ***void * pRxEnd***

Field Documentation

- ***uint32_t ETH_RxDescListTypeDef::RxDesc[ETH_RX_DESC_CNT]***
- ***uint32_t ETH_RxDescListTypeDef::ItMode***
- ***uint32_t ETH_RxDescListTypeDef::RxDescIdx***
- ***uint32_t ETH_RxDescListTypeDef::RxDescCnt***
- ***uint32_t ETH_RxDescListTypeDef::RxDataLength***
- ***uint32_t ETH_RxDescListTypeDef::RxBuildDescIdx***
- ***uint32_t ETH_RxDescListTypeDef::RxBuildDescCnt***
- ***uint32_t ETH_RxDescListTypeDef::pRxLastRxDesc***
- ***ETH_TimeStampTypeDef ETH_RxDescListTypeDef::TimeStamp***
- ***void* ETH_RxDescListTypeDef::pRxStart***
- ***void* ETH_RxDescListTypeDef::pRxEnd***

22.1.7 ETH_MACConfigTypeDef

ETH_MACConfigTypeDef is defined in the stm32n6xx_hal_eth.h

Data Fields

- ***uint32_t SourceAddrControl***
- ***FunctionalState ChecksumOffload***
- ***uint32_t InterPacketGapVal***
- ***FunctionalState GiantPacketSizeLimitControl***
- ***FunctionalState Support2KPacket***

- ***FunctionalState CRCStripTypePacket***
- ***FunctionalState AutomaticPadCRCStrip***
- ***FunctionalState Watchdog***
- ***FunctionalState Jabber***
- ***FunctionalState JumboPacket***
- ***FunctionalState PortSelect***
- ***uint32_t Speed***
- ***uint32_t DuplexMode***
- ***FunctionalState LoopbackMode***
- ***FunctionalState CarrierSenseBeforeTransmit***
- ***FunctionalState ReceiveOwn***
- ***FunctionalState CarrierSenseDuringTransmit***
- ***FunctionalState RetryTransmission***
- ***uint32_t BackOffLimit***
- ***FunctionalState DeferralCheck***
- ***uint32_t PreambleLength***
- ***FunctionalState UnicastSlowProtocolPacketDetect***
- ***FunctionalState SlowProtocolDetect***
- ***FunctionalState CRCCheckingRxPackets***
- ***uint32_t GiantPacketSizeLimit***
- ***FunctionalState ExtendedInterPacketGap***
- ***uint32_t ExtendedInterPacketGapVal***
- ***FunctionalState ProgrammableWatchdog***
- ***uint32_t WatchdogTimeout***
- ***uint32_t PauseTime***
- ***FunctionalState ZeroQuantaPause***
- ***uint32_t PauseLowThreshold***
- ***FunctionalState TransmitFlowControl***
- ***FunctionalState UnicastPausePacketDetect***
- ***FunctionalState ReceiveFlowControl***
- ***FunctionalState PacketBurst***

Field Documentation

- ***uint32_t ETH_MACConfigTypeDef::SourceAddrControl***
  Selects the Source Address Insertion or Replacement Control. This parameter can be a value of *ETH_Source_Addr_Control*
- ***FunctionalState ETH_MACConfigTypeDef::ChecksumOffload***
  Enables or Disable the checksum checking for received packet payloads TCP, UDP or ICMP headers
- ***uint32_t ETH_MACConfigTypeDef::InterPacketGapVal***
  Sets the minimum IPG between Packet during transmission. This parameter can be a value of *ETH_Inter_Packet_Gap*
- ***FunctionalState ETH_MACConfigTypeDef::GiantPacketSizeLimitControl***
  Enables or disables the Giant Packet Size Limit Control.
- ***FunctionalState ETH_MACConfigTypeDef::Support2KPacket***
  Enables or disables the IEEE 802.3as Support for 2K length Packets
- ***FunctionalState ETH_MACConfigTypeDef::CRCStripTypePacket***
  Enables or disables the CRC stripping for Type packets.
- ***FunctionalState ETH_MACConfigTypeDef::AutomaticPadCRCStrip***
  Enables or disables the Automatic MAC Pad/CRC Stripping.
- ***FunctionalState ETH_MACConfigTypeDef::Watchdog***
  Enables or disables the Watchdog timer on Rx path.
- ***FunctionalState ETH_MACConfigTypeDef::Jabber***
  Enables or disables Jabber timer on Tx path.

- ***FunctionalState ETH_MACConfigTypeDef::JumboPacket***
  Enables or disables receiving Jumbo Packet When enabled, the MAC allows jumbo packets of 9,018 bytes without reporting a giant packet error
- ***FunctionalState ETH_MACConfigTypeDef::PortSelect***
  Sets the Port Select When disabled, the line speed is for 1000Mbps operations When enabled, it is for 10 or 100Mbps operations
- ***uint32_t ETH_MACConfigTypeDef::Speed***
  Sets the Ethernet speed: 10/100 Mbps. This parameter can be a value of ***ETH_Speed***
- ***uint32_t ETH_MACConfigTypeDef::DuplexMode***
  Selects the MAC duplex mode: Half-Duplex or Full-Duplex mode This parameter can be a value of ***ETH_Duplex_Mode***
- ***FunctionalState ETH_MACConfigTypeDef::LoopbackMode***
  Enables or disables the loopback mode
- ***FunctionalState ETH_MACConfigTypeDef::CarrierSenseBeforeTransmit***
  Enables or disables the Carrier Sense Before Transmission in Full Duplex Mode.
- ***FunctionalState ETH_MACConfigTypeDef::ReceiveOwn***
  Enables or disables the Receive Own in Half Duplex mode.
- ***FunctionalState ETH_MACConfigTypeDef::CarrierSenseDuringTransmit***
  Enables or disables the Carrier Sense During Transmission in the Half Duplex mode
- ***FunctionalState ETH_MACConfigTypeDef::RetryTransmission***
  Enables or disables the MAC retry transmission, when a collision occurs in Half Duplex mode.
- ***uint32_t ETH_MACConfigTypeDef::BackOffLimit***
  Selects the BackOff limit value. This parameter can be a value of ***ETH_Back_Off_Limit***
- ***FunctionalState ETH_MACConfigTypeDef::DeferralCheck***
  Enables or disables the deferral check function in Half Duplex mode.
- ***uint32_t ETH_MACConfigTypeDef::PreambleLength***
  Selects or not the Preamble Length for Transmit packets (Full Duplex mode). This parameter can be a value of ***ETH_Preamble_Length***
- ***FunctionalState ETH_MACConfigTypeDef::UnicastSlowProtocolPacketDetect***
  Enable or disables the Detection of Slow Protocol Packets with unicast address.
- ***FunctionalState ETH_MACConfigTypeDef::SlowProtocolDetect***
  Enable or disables the Slow Protocol Detection.
- ***FunctionalState ETH_MACConfigTypeDef::CRCCheckingRxPackets***
  Enable or disables the CRC Checking for Received Packets.
- ***uint32_t ETH_MACConfigTypeDef::GiantPacketSizeLimit***
  Specifies the packet size that the MAC will declare it as Giant, If it's size is greater than the value programmed in this field in units of bytes This parameter must be a number between Min_Data = 0x618 (1518 byte) and Max_Data = 0x3FFF (32 Kbyte).
- ***FunctionalState ETH_MACConfigTypeDef::ExtendedInterPacketGap***
  Enable or disables the extended inter packet gap.
- ***uint32_t ETH_MACConfigTypeDef::ExtendedInterPacketGapVal***
  Sets the Extended IPG between Packet during transmission. This parameter can be a value from 0x0 to 0xFF
- ***FunctionalState ETH_MACConfigTypeDef::ProgrammableWatchdog***
  Enable or disables the Programmable Watchdog.
- ***uint32_t ETH_MACConfigTypeDef::WatchdogTimeout***
  This field is used as watchdog timeout for a received packet This parameter can be a value of ***ETH_Watchdog_Timeout***
- ***uint32_t ETH_MACConfigTypeDef::PauseTime***
  This field holds the value to be used in the Pause Time field in the transmit control packet. This parameter must be a number between Min_Data = 0x0 and Max_Data = 0xFFFF.
- ***FunctionalState ETH_MACConfigTypeDef::ZeroQuantaPause***
  Enable or disables the automatic generation of Zero Quanta Pause Control packets.
- ***uint32_t ETH_MACConfigTypeDef::PauseLowThreshold***
  This field configures the threshold of the PAUSE to be checked for automatic retransmission of PAUSE Packet. This parameter can be a value of ***ETH_Pause_Low_Threshold***

- ***FunctionalState ETH_MACConfigTypeDef::TransmitFlowControl***
  Enables or disables the MAC to transmit Pause packets in Full Duplex mode or the MAC back pressure operation in Half Duplex mode
- ***FunctionalState ETH_MACConfigTypeDef::UnicastPausePacketDetect***
  Enables or disables the MAC to detect Pause packets with unicast address of the station
- ***FunctionalState ETH_MACConfigTypeDef::ReceiveFlowControl***
  Enables or disables the MAC to decodes the received Pause packet and disables its transmitter for a specified (Pause) time
- ***FunctionalState ETH_MACConfigTypeDef::PacketBurst***
  Enables or disables the MAC to allow packet bursting during transmission in the GMII Halfduplex mode

22.1.8 DMAChannelInstanceTypeDef

DMAChannelInstanceTypeDef is defined in the stm32n6xx_hal_eth.h

Data Fields

- ***FunctionalState PBLx8Mode***
- ***uint32_t TxDMABurstLength***
- ***FunctionalState SecondPacketOperate***
- ***FunctionalState TCPSegmentation***
- ***uint32_t DescriptorSkipLength***
- ***FunctionalState FlushRxPacket***
- ***uint32_t RxDMABurstLength***
- ***uint32_t MaximumSegmentSize***

Field Documentation

- ***FunctionalState DMAChannelInstanceTypeDef::PBLx8Mode***
  Enables or disables the PBL multiplication by eight.
- ***uint32_t DMAChannelInstanceTypeDef::TxDMABurstLength***
  Indicates the maximum number of beats to be transferred in one Tx DMA transaction. This parameter can be a value of ***ETH_Tx_DMA_Burst_Length***
- ***FunctionalState DMAChannelInstanceTypeDef::SecondPacketOperate***
  Enables or disables the Operate on second Packet mode, which allows the DMA to process a second Packet of Transmit data even before obtaining the status for the first one.
- ***FunctionalState DMAChannelInstanceTypeDef::TCPSegmentation***
  Enables or disables the TCP Segmentation
- ***uint32_t DMAChannelInstanceTypeDef::DescriptorSkipLength***
  Sets the Descriptor Skip Length
- ***FunctionalState DMAChannelInstanceTypeDef::FlushRxPacket***
  Enables or disables the Rx Packet Flush
- ***uint32_t DMAChannelInstanceTypeDef::RxDMABurstLength***
  Indicates the maximum number of beats to be transferred in one Rx DMA transaction. This parameter can be a value of ***ETH_Rx_DMA_Burst_Length***
- ***uint32_t DMAChannelInstanceTypeDef::MaximumSegmentSize***
  Sets the maximum segment size that should be used while segmenting the packet This parameter can be a value from 0x40 to 0x3FFF

22.1.9 ETH_DMAConfigTypeDef

ETH_DMAConfigTypeDef is defined in the stm32n6xx_hal_eth.h

Data Fields

- ***DMAChannelInstanceTypeDef DMACh***
- ***uint32_t TransmitArbitrationAlgorithm***
- ***FunctionalState AddressAlignedBeats***
- ***uint32_t BurstMode***
- ***uint32_t RxOSRLimit***
- ***uint32_t TxOSRLimit***
- ***uint32_t AXIBLENMaxSize***

- ***FunctionalState TransmitPriority***

Field Documentation

- ***DMAChannelInstanceTypeDef ETH_DMAConfigTypeDef::DMACh[ETH_DMA_CH_CNT]***
  DMA Channel x Configuration
- ***uint32_t ETH_DMAConfigTypeDef::TransmitArbitrationAlgorithm***
  Speicifies the arbitration algorithm for the Transmit side
- ***FunctionalState ETH_DMAConfigTypeDef::AddressAlignedBeats***
  Enables or disables the AHB Master interface address aligned burst transfers on Read and Write channels
- ***uint32_t ETH_DMAConfigTypeDef::BurstMode***
  Sets the AHB Master interface burst transfers. This parameter can be a value of ***ETH_Burst_Mode***
- ***uint32_t ETH_DMAConfigTypeDef::RxOSRLimit***
  Sets the maximum outstanding request on the AXI read interface.
- ***uint32_t ETH_DMAConfigTypeDef::TxOSRLimit***
  Sets the maximum outstanding request on the AXI write interface.
- ***uint32_t ETH_DMAConfigTypeDef::AXIBLENMaxSize***
  Sets the maximum BLEN max size that should be used on th AXI interface to initiate burst transfers of specified lengths
- ***FunctionalState ETH_DMAConfigTypeDef::TransmitPriority***
  Sets Transmit priority over Receive

22.1.10 ETH_InitTypeDef

ETH_InitTypeDef is defined in the stm32n6xx_hal_eth.h

Data Fields

- ***uint8_t * MACAddr***
- ***ETH_MediaInterfaceTypeDef MediaInterface***
- ***ETH_DMADescTypeDef * TxDesc***
- ***ETH_DMADescTypeDef * RxDesc***
- ***uint32_t RxBuffLen***

Field Documentation

- ***uint8_t* ETH_InitTypeDef::MACAddr***
  MAC Address of used Hardware: must be pointer on an array of 6 bytes
- ***ETH_MediaInterfaceTypeDef ETH_InitTypeDef::MediaInterface***
  Selects the MII/RMII/RGMII interface.
- ***ETH_DMADescTypeDef* ETH_InitTypeDef::TxDesc[ETH_DMA_TX_CH_CNT]***
  Provides the address of the first DMA CH Tx descriptor in the list
- ***ETH_DMADescTypeDef* ETH_InitTypeDef::RxDesc[ETH_DMA_RX_CH_CNT]***
  Provides the address of the first DMA Rx CH descriptor in the list
- ***uint32_t ETH_InitTypeDef::RxBuffLen***
  Provides the length of Rx buffers size

22.1.11 __ETH_HandleTypeDef

__ETH_HandleTypeDef is defined in the stm32n6xx_hal_eth.h

Data Fields

- ***ETH_TypeDef * Instance***
- ***ETH_InitTypeDef Init***
- ***ETH_TxDescListTypeDef TxDescList***
- ***ETH_RxDescListTypeDef RxDescList***
- ***__IO HAL_ETH_StateTypeDef gState***
- ***__IO uint32_t ErrorCode***
- ***__IO uint32_t DMAErrorCode***
- ***__IO uint32_t MACErrorCode***
- ***__IO uint32_t MACWakeUpEvent***
- ***__IO uint32_t MACLPIEvent***

- *__IO uint32_t IsPtpConfigured*
- *__IO uint32_t TxCH*
- *__IO uint32_t RxCH*
- *__IO uint32_t TxOpCH*
- *__IO uint32_t RxOpCH*
- *void(* TxCpltCallback*
- *void(* RxCpltCallback*
- *void(* ErrorCallback*
- *void(* PMTCallback*
- *void(* EEECallback*
- *void(* WakeUpCallback*
- *void(* MspInitCallback*
- *void(* MspDeInitCallback*
- *pETH_rxAllocateCallbackTypeDef rxAllocateCallback*
- *pETH_rxLinkCallbackTypeDef rxLinkCallback*
- *pETH_txFreeCallbackTypeDef txFreeCallback*
- *pETH_txPtpCallbackTypeDef txPtpCallback*

Field Documentation

- ***ETH_TypeDef* __ETH_HandleTypeDef::Instance***
  Register base address
- ***ETH_InitTypeDef __ETH_HandleTypeDef::Init***
  Ethernet Init Configuration
- ***ETH_TxDescListTypeDef __ETH_HandleTypeDef::TxDescList[ETH_DMA_TX_CH_CNT]***
  DMA Channel Tx descriptor wrapper: holds all Tx descriptors list addresses and current descriptor index
- ***ETH_RxDescListTypeDef __ETH_HandleTypeDef::RxDescList[ETH_DMA_RX_CH_CNT]***
  DMA Channel Rx descriptor wrapper: holds all Rx descriptors list addresses and current descriptor index
- ***__IO HAL_ETH_StateTypeDef __ETH_HandleTypeDef::gState***
  ETH state information related to global Handle management and also related to Tx operations. This parameter can be a value of ***ETH_State_Codes***
- ***__IO uint32_t __ETH_HandleTypeDef::ErrorCode***
  Holds the global Error code of the ETH HAL status machine This parameter can be a value of ***ETH_Error_Code***.
- ***__IO uint32_t __ETH_HandleTypeDef::DMAErrorCode***
  Holds the DMA CH0/CH1 Rx Tx Error code when a DMA AIS interrupt occurs This parameter can be a combination of ***ETH_DMA_Status_Flags***
- ***__IO uint32_t __ETH_HandleTypeDef::MACErrorCode***
  Holds the MAC Rx Tx Error code when a MAC Rx or Tx status interrupt occurs This parameter can be a combination of ***ETH_MAC_Rx_Tx_Status***
- ***__IO uint32_t __ETH_HandleTypeDef::MACWakeUpEvent***
  Holds the Wake Up event when the MAC exit the power down mode This parameter can be a value of ***ETH_MAC_Wake_Up_Event***
- ***__IO uint32_t __ETH_HandleTypeDef::MACLPIEvent***
  Holds the LPI event when the an LPI status interrupt occurs. This parameter can be a value of ***ETHEx_LPI_Event***
- ***__IO uint32_t __ETH_HandleTypeDef::IsPtpConfigured***
  Holds the PTP configuration status. This parameter can be a value of ***ETH_PTP_Config_Status***
- ***__IO uint32_t __ETH_HandleTypeDef::TxCH***
  Holds the Tx DMA Channels Number events. This parameter can be a value of ***ETH_DMA_Channel_Number_Tx_Rx_Selection***
- ***__IO uint32_t __ETH_HandleTypeDef::RxCH***
  Holds the Rx DMA Channels Number events. This parameter can be a value of ***ETH_DMA_Channel_Number_Tx_Rx_Selection***

- ***__IO uint32_t __ETH_HandleTypeDef::TxOpCH***
  Holds the Tx DMA Channel Index which transfer data. This parameter can be a value of ***ETH_DMA_Channel_Number_TxOp_RxOp_Selection***
- ***__IO uint32_t __ETH_HandleTypeDef::RxOpCH***
  Holds the Rx DMA Channel Index which receive data. This parameter can be a value of ***ETH_DMA_Channel_Number_TxOp_RxOp_Selection***
- ***void(* __ETH_HandleTypeDef::TxCpltCallback)(struct __ETH_HandleTypeDef *heth)***
  ETH Tx Complete Callback
- ***void(* __ETH_HandleTypeDef::RxCpltCallback)(struct __ETH_HandleTypeDef *heth)***
  ETH Rx Complete Callback
- ***void(* __ETH_HandleTypeDef::ErrorCallback)(struct __ETH_HandleTypeDef *heth)***
  ETH Error Callback
- ***void(* __ETH_HandleTypeDef::PMTCallback)(struct __ETH_HandleTypeDef *heth)***
  ETH Power Management Callback
- ***void(* __ETH_HandleTypeDef::EEECallback)(struct __ETH_HandleTypeDef *heth)***
  ETH EEE Callback
- ***void(* __ETH_HandleTypeDef::WakeUpCallback)(struct __ETH_HandleTypeDef *heth)***
  ETH Wake UP Callback
- ***void(* __ETH_HandleTypeDef::MspInitCallback)(struct __ETH_HandleTypeDef *heth)***
  ETH Msp Init callback
- ***void(* __ETH_HandleTypeDef::MspDeInitCallback)(struct __ETH_HandleTypeDef *heth)***
  ETH Msp DeInit callback
- ***pETH_rxAllocateCallbackTypeDef __ETH_HandleTypeDef::rxAllocateCallback***
  ETH Rx Get Buffer Function
- ***pETH_rxLinkCallbackTypeDef __ETH_HandleTypeDef::rxLinkCallback***
  ETH Rx Set App Data Function
- ***pETH_txFreeCallbackTypeDef __ETH_HandleTypeDef::txFreeCallback***
  ETH Tx Free Function
- ***pETH_txPtpCallbackTypeDef __ETH_HandleTypeDef::txPtpCallback***
  ETH Tx Handle Ptp Function

22.1.12 ETH_MACFilterConfigTypeDef

ETH_MACFilterConfigTypeDef is defined in the stm32n6xx_hal_eth.h

Data Fields

- ***FunctionalState PromiscuousMode***
- ***FunctionalState ReceiveAllMode***
- ***FunctionalState HachOrPerfectFilter***
- ***FunctionalState HashUnicast***
- ***FunctionalState HashMulticast***
- ***FunctionalState PassAllMulticast***
- ***FunctionalState SrcAddrFiltering***
- ***FunctionalState SrcAddrInverseFiltering***
- ***FunctionalState DestAddrInverseFiltering***
- ***FunctionalState BroadcastFilter***
- ***uint32_t ControlPacketsFilter***

Field Documentation

- ***FunctionalState ETH_MACFilterConfigTypeDef::PromiscuousMode***
  Enable or Disable Promiscuous Mode
- ***FunctionalState ETH_MACFilterConfigTypeDef::ReceiveAllMode***
  Enable or Disable Receive All Mode
- ***FunctionalState ETH_MACFilterConfigTypeDef::HachOrPerfectFilter***
  Enable or Disable Perfect filtering in addition to Hash filtering
- ***FunctionalState ETH_MACFilterConfigTypeDef::HashUnicast***
  Enable or Disable Hash filtering on unicast packets

- ***FunctionalState ETH_MACFilterConfigTypeDef::HashMulticast***
  Enable or Disable Hash filtering on multicast packets
- ***FunctionalState ETH_MACFilterConfigTypeDef::PassAllMulticast***
  Enable or Disable passing all multicast packets
- ***FunctionalState ETH_MACFilterConfigTypeDef::SrcAddrFiltering***
  Enable or Disable source address filtering module
- ***FunctionalState ETH_MACFilterConfigTypeDef::SrcAddrInverseFiltering***
  Enable or Disable source address inverse filtering
- ***FunctionalState ETH_MACFilterConfigTypeDef::DestAddrInverseFiltering***
  Enable or Disable destination address inverse filtering
- ***FunctionalState ETH_MACFilterConfigTypeDef::BroadcastFilter***
  Enable or Disable broadcast filter
- ***uint32_t ETH_MACFilterConfigTypeDef::ControlPacketsFilter***
  Set the control packets filter This parameter can be a value of ***ETH_Control_Packets_Filter***

22.1.13 ETH_PowerDownConfigTypeDef

ETH_PowerDownConfigTypeDef is defined in the stm32n6xx_hal_eth.h

Data Fields

- ***FunctionalState WakeUpPacket***
- ***FunctionalState MagicPacket***
- ***FunctionalState GlobalUnicast***
- ***FunctionalState WakeUpForward***

Field Documentation

- ***FunctionalState ETH_PowerDownConfigTypeDef::WakeUpPacket***
  Enable or Disable Wake up packet detection in power down mode
- ***FunctionalState ETH_PowerDownConfigTypeDef::MagicPacket***
  Enable or Disable Magic packet detection in power down mode
- ***FunctionalState ETH_PowerDownConfigTypeDef::GlobalUnicast***
  Enable or Disable Global unicast packet detection in power down mode
- ***FunctionalState ETH_PowerDownConfigTypeDef::WakeUpForward***
  Enable or Disable Forwarding Wake up packets

22.2 ETH Firmware driver API description

The following section lists the various functions of the ETH library.

22.2.1 How to use this driver

The ETH HAL driver can be used as follows:

1. Declare a ETH_HandleTypeDef handle structure, for example: ETH_HandleTypeDef heth;
2. Fill parameters of Init structure in heth handle
3. Call HAL_ETH_Init() API to initialize the Ethernet peripheral (MAC, DMA, ...)
4. Initialize the ETH low level resources through the HAL_ETH_MspInit() API:
   a. Enable the Ethernet interface clock using
      - __HAL_RCC_ETH1MAC_CLK_ENABLE()
      - __HAL_RCC_ETH1TX_CLK_ENABLE()
      - __HAL_RCC_ETH1RX_CLK_ENABLE()
   b. Initialize the related GPIO clocks
   c. Configure Ethernet pinout
   d. Configure Ethernet NVIC interrupt (in Interrupt mode)

5. Ethernet data reception is asynchronous, so call the following API to start the listening mode:
   a. HAL_ETH_Start(): This API starts the MAC and DMA transmission and reception process, without enabling end of transfer interrupts, in this mode user has to poll for data reception by calling HAL_ETH_ReadData()
   b. HAL_ETH_Start_IT(): This API starts the MAC and DMA transmission and reception process, end of transfer interrupts are enabled in this mode, HAL_ETH_RxCpltCallback() will be executed when an Ethernet packet is received
6. When data is received user can call the following API to get received data:
   a. HAL_ETH_ReadData(): Read a received packet
7. For transmission path, two APIs are available:
   a. HAL_ETH_Transmit(): Transmit an ETH frame in blocking mode
   b. HAL_ETH_Transmit_IT(): Transmit an ETH frame in interrupt mode, HAL_ETH_TxCpltCallback() will be executed when end of transfer occur
8. Communication with an external PHY device:
   a. HAL_ETH_ReadPHYRegister(): Read a register from an external PHY
   b. HAL_ETH_WritePHYRegister(): Write data to an external RHY register
9. Configure the Ethernet MAC after ETH peripheral initialization
   a. HAL_ETH_GetMACConfig(): Get MAC actual configuration into ETH_MACConfigTypeDef
   b. HAL_ETH_SetMACConfig(): Set MAC configuration based on ETH_MACConfigTypeDef
10. Configure the Ethernet DMA after ETH peripheral initialization
    a. HAL_ETH_GetDMAConfig(): Get DMA actual configuration into ETH_DMAConfigTypeDef
    b. HAL_ETH_SetDMAConfig(): Set DMA configuration based on ETH_DMAConfigTypeDef
11. Configure the Ethernet PTP after ETH peripheral initialization
    a. Define HAL_ETH_USE_PTP to use PTP APIs.
    b. HAL_ETH_PTP_GetConfig(): Get PTP actual configuration into ETH_PTP_ConfigTypeDef
    c. HAL_ETH_PTP_SetConfig(): Set PTP configuration based on ETH_PTP_ConfigTypeDef
    d. HAL_ETH_PTP_GetTime(): Get Seconds and Nanoseconds for the Ethernet PTP registers
    e. HAL_ETH_PTP_SetTime(): Set Seconds and Nanoseconds for the Ethernet PTP registers
    f. HAL_ETH_PTP_AddTimeOffset(): Add Seconds and Nanoseconds offset for the Ethernet PTP registers
    g. HAL_ETH_PTP_AddendUpdate(): Update the Addend register
    h. HAL_ETH_PTP_InsertTxTimestamp(): Insert Timestamp in transmission
    i. HAL_ETH_PTP_GetTxTimestamp(): Get transmission timestamp
    j. HAL_ETH_PTP_GetRxTimestamp(): Get reception timestamp -@- The ARP offload feature is not supported in this driver. -@- The PTP offload feature is not supported in this driver.

Callback registration

22.2.2 Detailed description of functions

HAL_ETH_Init

Function name

HAL_StatusTypeDef HAL_ETH_Init (ETH_HandleTypeDef * heth)

Function description

HAL_ETH_DeInit

Function name

HAL_StatusTypeDef HAL_ETH_DeInit (ETH_HandleTypeDef * heth)

Function description

HAL_ETH_MspInit

Function name

void HAL_ETH_MspInit (ETH_HandleTypeDef * heth)

Function description

HAL_ETH_MspDeInit

Function name

void HAL_ETH_MspDeInit (ETH_HandleTypeDef * heth)

Function description

HAL_ETH_RegisterCallback

Function name

HAL_StatusTypeDef HAL_ETH_RegisterCallback (ETH_HandleTypeDef * heth, HAL_ETH_CallbackIDTypeDef CallbackID, pETH_CallbackTypeDef pCallback)

Function description

HAL_ETH_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_ETH_UnRegisterCallback (ETH_HandleTypeDef * heth, HAL_ETH_CallbackIDTypeDef CallbackID)

Function description

HAL_ETH_Start

Function name

HAL_StatusTypeDef HAL_ETH_Start (ETH_HandleTypeDef * heth)

Function description

HAL_ETH_Start_IT

Function name

HAL_StatusTypeDef HAL_ETH_Start_IT (ETH_HandleTypeDef * heth)

Function description

HAL_ETH_Stop

Function name

HAL_StatusTypeDef HAL_ETH_Stop (ETH_HandleTypeDef * heth)

Function description

HAL_ETH_Stop_IT

Function name

HAL_StatusTypeDef HAL_ETH_Stop_IT (ETH_HandleTypeDef * heth)

Function description

HAL_ETH_ReadData

Function name

HAL_StatusTypeDef HAL_ETH_ReadData (ETH_HandleTypeDef * heth, void ** pAppBuff)

Function description

HAL_ETH_RegisterRxAllocateCallback

Function name

HAL_StatusTypeDef HAL_ETH_RegisterRxAllocateCallback (ETH_HandleTypeDef * heth, pETH_rxAllocateCallbackTypeDef rxAllocateCallback)

Function description

HAL_ETH_UnRegisterRxAllocateCallback

Function name

HAL_StatusTypeDef HAL_ETH_UnRegisterRxAllocateCallback (ETH_HandleTypeDef * heth)

Function description

HAL_ETH_RegisterRxLinkCallback

Function name

HAL_StatusTypeDef HAL_ETH_RegisterRxLinkCallback (ETH_HandleTypeDef * heth, pETH_rxLinkCallbackTypeDef rxLinkCallback)

Function description

HAL_ETH_UnRegisterRxLinkCallback

Function name

HAL_StatusTypeDef HAL_ETH_UnRegisterRxLinkCallback (ETH_HandleTypeDef * heth)

Function description

HAL_ETH_GetRxDataErrorCode

Function name

HAL_StatusTypeDef HAL_ETH_GetRxDataErrorCode (const ETH_HandleTypeDef * heth, uint32_t * pErrorCode)

Function description

HAL_ETH_RegisterTxFreeCallback

Function name

HAL_StatusTypeDef HAL_ETH_RegisterTxFreeCallback (ETH_HandleTypeDef * heth, pETH_txFreeCallbackTypeDef txFreeCallback)

Function description

HAL_ETH_UnRegisterTxFreeCallback

Function name

HAL_StatusTypeDef HAL_ETH_UnRegisterTxFreeCallback (ETH_HandleTypeDef * heth)

Function description

HAL_ETH_ReleaseTxPacket

Function name

HAL_StatusTypeDef HAL_ETH_ReleaseTxPacket (ETH_HandleTypeDef * heth)

Function description

HAL_ETH_Transmit

Function name

HAL_StatusTypeDef HAL_ETH_Transmit (ETH_HandleTypeDef * heth, ETH_TxPacketConfigTypeDef * pTxConfig, uint32_t Timeout)

Function description

HAL_ETH_Transmit_IT

Function name

HAL_StatusTypeDef HAL_ETH_Transmit_IT (ETH_HandleTypeDef * heth, ETH_TxPacketConfigTypeDef * pTxConfig)

Function description

HAL_ETH_WritePHYRegister

Function name

HAL_StatusTypeDef HAL_ETH_WritePHYRegister (const ETH_HandleTypeDef * heth, uint32_t PHYAddr, uint32_t PHYReg, uint32_t RegValue)

Function description

HAL_ETH_ReadPHYRegister

Function name

HAL_StatusTypeDef HAL_ETH_ReadPHYRegister (ETH_HandleTypeDef * heth, uint32_t PHYAddr, uint32_t PHYReg, uint32_t * pRegValue)

Function description

HAL_ETH_IRQHandler

Function name

void HAL_ETH_IRQHandler (ETH_HandleTypeDef * heth)

Function description

HAL_ETH_TxCpltCallback

Function name

void HAL_ETH_TxCpltCallback (ETH_HandleTypeDef * heth)

Function description

HAL_ETH_RxCpltCallback

Function name

void HAL_ETH_RxCpltCallback (ETH_HandleTypeDef * heth)

Function description

HAL_ETH_ErrorCallback

Function name

void HAL_ETH_ErrorCallback (ETH_HandleTypeDef * heth)

Function description

HAL_ETH_PMTCallback

Function name

void HAL_ETH_PMTCallback (ETH_HandleTypeDef * heth)

Function description

HAL_ETH_EEECallback

Function name

void HAL_ETH_EEECallback (ETH_HandleTypeDef * heth)

Function description

HAL_ETH_WakeUpCallback

Function name

void HAL_ETH_WakeUpCallback (ETH_HandleTypeDef * heth)

Function description

HAL_ETH_RxAllocateCallback

Function name

void HAL_ETH_RxAllocateCallback (uint8_t ** buff)

Function description

HAL_ETH_RxLinkCallback

Function name

void HAL_ETH_RxLinkCallback (void ** pStart, void ** pEnd, uint8_t * buff, uint16_t Length)

Function description

HAL_ETH_TxFreeCallback

Function name

void HAL_ETH_TxFreeCallback (uint32_t * buff)

Function description

HAL_ETH_TxPtpCallback

Function name

void HAL_ETH_TxPtpCallback (uint32_t * buff, ETH_TimeStampTypeDef * timestamp)

Function description

HAL_ETH_GetMACConfig

Function name

HAL_StatusTypeDef HAL_ETH_GetMACConfig (const ETH_HandleTypeDef * heth, ETH_MACConfigTypeDef * macconf)

Function description

HAL_ETH_GetDMAConfig

Function name

HAL_StatusTypeDef HAL_ETH_GetDMAConfig (const ETH_HandleTypeDef * heth, ETH_DMAConfigTypeDef * dmaconf)

Function description

HAL_ETH_SetMACConfig

Function name

HAL_StatusTypeDef HAL_ETH_SetMACConfig (ETH_HandleTypeDef * heth, ETH_MACConfigTypeDef * macconf)

Function description

HAL_ETH_SetDMAConfig

Function name

HAL_StatusTypeDef HAL_ETH_SetDMAConfig (ETH_HandleTypeDef * heth, ETH_DMAConfigTypeDef * dmaconf)

Function description

HAL_ETH_SetMDIOClockRange

Function name

void HAL_ETH_SetMDIOClockRange (ETH_HandleTypeDef * heth)

Function description

HAL_ETH_SetRxVLANIdentifier

Function name

void HAL_ETH_SetRxVLANIdentifier (ETH_HandleTypeDef * heth, uint32_t ComparisonBits, uint32_t VLANIdentifier)

Function description

HAL_ETH_GetMACFilterConfig

Function name

HAL_StatusTypeDef HAL_ETH_GetMACFilterConfig (const ETH_HandleTypeDef * heth, ETH_MACFilterConfigTypeDef * pFilterConfig)

Function description

HAL_ETH_SetMACFilterConfig

Function name

HAL_StatusTypeDef HAL_ETH_SetMACFilterConfig (ETH_HandleTypeDef * heth, const ETH_MACFilterConfigTypeDef * pFilterConfig)

Function description

HAL_ETH_SetHashTable

Function name

HAL_StatusTypeDef HAL_ETH_SetHashTable (ETH_HandleTypeDef * heth, uint32_t * pHashTable)

Function description

HAL_ETH_SetSourceMACAddrMatch

Function name

HAL_StatusTypeDef HAL_ETH_SetSourceMACAddrMatch (const ETH_HandleTypeDef * heth, uint32_t AddrNbr, const uint8_t * pMACAddr)

Function description

HAL_ETH_EnterPowerDownMode

Function name

void HAL_ETH_EnterPowerDownMode (ETH_HandleTypeDef * heth, const ETH_PowerDownConfigTypeDef * pPowerDownConfig)

Function description

HAL_ETH_ExitPowerDownMode

Function name

void HAL_ETH_ExitPowerDownMode (ETH_HandleTypeDef * heth)

Function description

HAL_ETH_SetWakeUpFilter

Function name

HAL_StatusTypeDef HAL_ETH_SetWakeUpFilter (ETH_HandleTypeDef * heth, uint32_t * pFilter, uint32_t Count)

Function description

HAL_ETH_GetState

Function name

HAL_ETH_StateTypeDef HAL_ETH_GetState (const ETH_HandleTypeDef * heth)

Function description

HAL_ETH_GetError

Function name

uint32_t HAL_ETH_GetError (const ETH_HandleTypeDef * heth)

Function description

HAL_ETH_GetDMAError

Function name

uint32_t HAL_ETH_GetDMAError (const ETH_HandleTypeDef * heth)

Function description

HAL_ETH_GetMACError

Function name

uint32_t HAL_ETH_GetMACError (const ETH_HandleTypeDef * heth)

Function description

HAL_ETH_GetMACWakeUpSource

Function name

uint32_t HAL_ETH_GetMACWakeUpSource (const ETH_HandleTypeDef * heth)

Function description

HAL_ETH_GetTxBuffersNumber

Function name

uint32_t HAL_ETH_GetTxBuffersNumber (const ETH_HandleTypeDef * heth)

Function description

22.3 ETH Firmware driver defines

The following section lists the various define and macros of the module.

22.3.1 ETH

ETH

ETH Back Off Limit

ETH_BACKOFFLIMIT_10

ETH_BACKOFFLIMIT_8

ETH_BACKOFFLIMIT_4

ETH_BACKOFFLIMIT_1

ETH Blen size

ETH_BLEN_MAX_SIZE_256

ETH_BLEN_MAX_SIZE_128

ETH_BLEN_MAX_SIZE_64

ETH_BLEN_MAX_SIZE_32

ETH_BLEN_MAX_SIZE_16

ETH_BLEN_MAX_SIZE_8

ETH_BLEN_MAX_SIZE_4

ETH Burst Mode

ETH_BURSTLENGTH_FIXED

ETH_BURSTLENGTH_UNSPECIFIED

ETH Control Packets Filter

ETH_CTRLPACKETS_BLOCK_ALL

ETH_CTRLPACKETS_FORWARD_ALL_EXCEPT_PA

ETH_CTRLPACKETS_FORWARD_ALL

ETH_CTRLPACKETS_FORWARD_PASSED_ADDR_FILTER

ETH DMA Arbitration

ETH_DMAARBITRATION_TX

ETH DMA Channel Number Tx Rx Selection Index

ETH_DMA_CH0_IDX

ETH_DMA_CH1_IDX

ETH DMA Channel Number Tx Rx Selection

ETH_DMA_CH0

ETH_DMA_CH1

ETH DMA Descriptor Skip Length

ETH_DMA_DESC_SKIP_LENGTH_0

ETH_DMA_DESC_SKIP_LENGTH_32

ETH_DMA_DESC_SKIP_LENGTH_64

ETH_DMA_DESC_SKIP_LENGTH_128

ETH DMA Interrupts

ETH_DMA_NORMAL_IT

ETH_DMA_ABNORMAL_IT

ETH_DMA_CONTEXT_DESC_ERROR_IT

ETH_DMA_FATAL_BUS_ERROR_IT

ETH_DMA_EARLY_RX_IT

ETH_DMA_EARLY_TX_IT

ETH_DMA_RX_WATCHDOG_TIMEOUT_IT

ETH_DMA_RX_PROCESS_STOPPED_IT

ETH_DMA_RX_BUFFER_UNAVAILABLE_IT

ETH_DMA_RX_IT

ETH_DMA_TX_BUFFER_UNAVAILABLE_IT

ETH_DMA_TX_PROCESS_STOPPED_IT

ETH_DMA_TX_IT

ETH DMA Rx Descriptor Bit Definition

ETH_DMARXNDESCRF_BUF1AP
: Header or Buffer 1 Address Pointer

ETH_DMARXNDESCRF_BUF2AP
: Buffer 2 Address Pointer

ETH_DMARXNDESCRF_OWN
: OWN bit: descriptor is owned by DMA engine

ETH_DMARXNDESCRF_IOC
: Interrupt Enabled on Completion

ETH_DMARXNDESCRF_BUF2V
: Buffer 2 Address Valid

ETH_DMARXNDESCRF_BUF1V
: Buffer 1 Address Valid

ETH_DMARXNDESCWBF_IVT
: Inner VLAN Tag

ETH_DMARXNDESCWBF_OVT
: Outer VLAN Tag

ETH_DMARXNDESCWBF_OPC

OAM Sub-Type Code, or MAC Control Packet opcode

ETH_DMARXNDESCWBF_TD

Timestamp Dropped

ETH_DMARXNDESCWBF_TSA

Timestamp Available

ETH_DMARXNDESCWBF_PV

PTP Version

ETH_DMARXNDESCWBF_PFT

PTP Packet Type

ETH_DMARXNDESCWBF_PMT_NO

PTP Message Type: No PTP message received

ETH_DMARXNDESCWBF_PMT_SYNC

PTP Message Type: SYNC (all clock types)

ETH_DMARXNDESCWBF_PMT_FUP

PTP Message Type: Follow_Up (all clock types)

ETH_DMARXNDESCWBF_PMT_DREQ

PTP Message Type: Delay_Req (all clock types)

ETH_DMARXNDESCWBF_PMT_DRESP

PTP Message Type: Delay_Resp (all clock types)

ETH_DMARXNDESCWBF_PMT_PDREQ

PTP Message Type: Pdelay_Req (in peer-to-peer transparent clock)

ETH_DMARXNDESCWBF_PMT_PDRESP

PTP Message Type: Pdelay_Resp (in peer-to-peer transparent clock)

ETH_DMARXNDESCWBF_PMT_PDRESPFUP

PTP Message Type: Pdelay_Resp_Follow_Up (in peer-to-peer transparent clock)

ETH_DMARXNDESCWBF_PMT_ANNOUNCE

PTP Message Type: Announce

ETH_DMARXNDESCWBF_PMT_MANAG

PTP Message Type: Management

ETH_DMARXNDESCWBF_PMT_SIGN

PTP Message Type: Signaling

ETH_DMARXNDESCWBF_PMT_RESERVED

PTP Message Type: PTP packet with Reserved message type

ETH_DMARXNDESCWBF_IPCE

IP Payload Error

ETH_DMARXNDESCWBF_IPCB

IP Checksum Bypassed

ETH_DMARXNDESCWBF_IPV6

IPv6 header Present

ETH_DMARXNDESCWBF_IPV4

IPv4 header Present

ETH_DMARXNDESCWBF_IPHE

IP Header Error

ETH_DMARXNDESCWBF_PT

Payload Type mask

ETH_DMARXNDESCWBF_PT_UNKNOWN

Payload Type: Unknown type or IP/AV payload not processed

ETH_DMARXNDESCWBF_PT_UDP

Payload Type: UDP

ETH_DMARXNDESCWBF_PT_TCP

Payload Type: TCP

ETH_DMARXNDESCWBF_PT_ICMP

Payload Type: ICMP

ETH_DMARXNDESCWBF_L3L4FM

L3 and L4 Filter Number Matched: if reset filter 0 is matched , if set filter 1 is matched

ETH_DMARXNDESCWBF_L4FM

Layer 4 Filter Match

ETH_DMARXNDESCWBF_L3FM

Layer 3 Filter Match

ETH_DMARXNDESCWBF_MADRM

MAC Address Match or Hash Value

ETH_DMARXNDESCWBF_HF

Hash Filter Status

ETH_DMARXNDESCWBF_DAF

Destination Address Filter Fail

ETH_DMARXNDESCWBF_SAF

SA Address Filter Fail

ETH_DMARXNDESCWBF_VF

VLAN Filter Status

ETH_DMARXNDESCWBF_ARPNR

ARP Reply Not Generated

ETH_DMARXNDESCWBF_OWN

Own Bit

ETH_DMARXNDESCWBF_CTXT

Receive Context Descriptor

ETH_DMARXNDESCWBF_FD

First Descriptor

ETH_DMARXNDESCWBF_LD

Last Descriptor

ETH_DMARXNDESCWBF_RS2V

Receive Status RDES2 Valid

ETH_DMARXNDESCWBF_RS1V

Receive Status RDES1 Valid

ETH_DMARXNDESCWBF_RS0V

Receive Status RDES0 Valid

ETH_DMARXNDESCWBF_CE

CRC Error

ETH_DMARXNDESCWBF_GP

Giant Packet

ETH_DMARXNDESCWBF_RWT

Receive Watchdog Timeout

ETH_DMARXNDESCWBF_OE

Overflow Error

ETH_DMARXNDESCWBF_RE

Receive Error

ETH_DMARXNDESCWBF_DE

Dribble Bit Error

ETH_DMARXNDESCWBF_LT

Length/Type Field

ETH_DMARXNDESCWBF_LT_LP

The packet is a length packet

ETH_DMARXNDESCWBF_LT_TP

The packet is a type packet

ETH_DMARXNDESCWBF_LT_ARP

The packet is a ARP Request packet type

ETH_DMARXNDESCWBF_LT_VLAN

The packet is a type packet with VLAN Tag

ETH_DMARXNDESCWBF_LT_DVLAN

The packet is a type packet with Double VLAN Tag

ETH_DMARXNDESCWBF_LT_MAC

The packet is a MAC Control packet type

ETH_DMARXNDESCWBF_LT_OAM

The packet is a OAM packet type

ETH_DMARXNDESCWBF_ES

Error Summary

ETH_DMARXNDESCWBF_PL

Packet Length

ETH_DMARXCDESC_RTSL

Receive Packet Timestamp Low

ETH_DMARXCDESC_RTSH

Receive Packet Timestamp High

ETH_DMARXCDESC_OWN

Own Bit

ETH_DMARXCDESC_CTXT

Receive Context Descriptor

ETH DMA Status Flags

ETH_DMA_RX_NO_ERROR_FLAG

ETH_DMA_TX_NO_ERROR_FLAG

ETH_DMA_CONTEXT_DESC_ERROR_FLAG

ETH_DMA_FATAL_BUS_ERROR_FLAG

ETH_DMA_EARLY_TX_IT_FLAG

ETH_DMA_RX_WATCHDOG_TIMEOUT_FLAG

ETH_DMA_RX_PROCESS_STOPPED_FLAG

ETH_DMA_RX_BUFFER_UNAVAILABLE_FLAG

ETH_DMA_TX_PROCESS_STOPPED_FLAG

ETH DMA Transmit Arbitration Algorithm

ETH_DMATXARBITRATION_FIXED_PRIO

ETH_DMATXARBITRATION_WEIGHTED_SP

ETH_DMATXARBITRATION_WEIGHTED_RR

ETH DMA Tx Descriptor Bit Definition

ETH_DMATXNDESCRF_B1AP

Transmit Packet Timestamp Low

ETH_DMATXNDESCRF_B2AP

Transmit Packet Timestamp High

ETH_DMATXNDESCRF_IOC

Interrupt on Completion

ETH_DMATXNDESCRF_TTSE

Transmit Timestamp Enable

ETH_DMATXNDESCRF_B2L

Buffer 2 Length

ETH_DMATXNDESCRF_VTIR

VLAN Tag Insertion or Replacement mask

ETH_DMATXNDESCRF_VTIR_DISABLE

Do not add a VLAN tag.

ETH_DMATXNDESCRF_VTIR_REMOVE

Remove the VLAN tag from the packets before transmission.

ETH_DMATXNDESCRF_VTIR_INSERT

Insert a VLAN tag.

ETH_DMATXNDESCRF_VTIR_REPLACE

Replace the VLAN tag.

ETH_DMATXNDESCRF_B1L

Buffer 1 Length

ETH_DMATXNDESCRF_HL

Header Length

ETH_DMATXNDESCRF_OWN

OWN bit: descriptor is owned by DMA engine

ETH_DMATXNDESCRF_CTXT

Context Type

ETH_DMATXNDESCRF_FD

First Descriptor

ETH_DMATXNDESCRF_LD

Last Descriptor

ETH_DMATXNDESCRF_CPC

CRC Pad Control mask

ETH_DMATXNDESCRF_CPC_CRCPAD_INSERT

CRC Pad Control: CRC and Pad Insertion

ETH_DMATXNDESCRF_CPC_CRC_INSERT

CRC Pad Control: CRC Insertion (Disable Pad Insertion)

ETH_DMATXNDESCRF_CPC_DISABLE

CRC Pad Control: Disable CRC Insertion

ETH_DMATXNDESCRF_CPC_CRC_REPLACE

CRC Pad Control: CRC Replacement

ETH_DMATXNDESCRF_SAIC

SA Insertion Control mask

ETH_DMATXNDESCRF_SAIC_DISABLE

SA Insertion Control: Do not include the source address

ETH_DMATXNDESCRF_SAIC_INSERT

SA Insertion Control: Include or insert the source address

ETH_DMATXNDESCRF_SAIC_REPLACE

SA Insertion Control: Replace the source address

ETH_DMATXNDESCRF_THL

TCP Header Length

ETH_DMATXNDESCRF_SLOTNUM

Slot Number Control

ETH_DMATXNDESCRF_TSE

TCP segmentation enable

ETH_DMATXNDESCRF_CIC

Checksum Insertion Control: 4 cases

ETH_DMATXNDESCRF_CIC_DISABLE

Do Nothing: Checksum Engine is disabled

ETH_DMATXNDESCRF_CIC_IPHDR_INSERT

Only IP header checksum calculation and insertion are enabled.

ETH_DMATXNDESCRF_CIC_IPHDR_PAYLOAD_INSERT

IP header checksum and payload checksum calculation and insertion are enabled, but pseudo header checksum is not calculated in hardware

ETH_DMATXNDESCRF_CIC_IPHDR_PAYLOAD_INSERT_PHDR_CALC

IP Header checksum and payload checksum calculation and insertion are enabled, and pseudo header checksum is calculated in hardware.

ETH_DMATXNDESCRF_TPL

TCP Payload Length

ETH_DMATXNDESCRF_FL

Transmit End of Ring

ETH_DMATXNDESCWBF_TTSL

Buffer1 Address Pointer or TSO Header Address Pointer

ETH_DMATXNDESCWBF_TTSH

Buffer2 Address Pointer

ETH_DMATXNDESCWBF_OWN

OWN bit: descriptor is owned by DMA engine

ETH_DMATXNDESCWBF_CTXT

Context Type

ETH_DMATXNDESCWBF_FD

First Descriptor

ETH_DMATXNDESCWBF_LD

Last Descriptor

ETH_DMATXNDESCWBF_TTSS

Tx Timestamp Status

ETH_DMATXNDESCWBF_DP

Disable Padding

ETH_DMATXNDESCWBF_TTSE

Transmit Timestamp Enable

ETH_DMATXNDESCWBF_ES

Error summary: OR of the following bits: IHE || UF || ED || EC || LCO || PCE || NC || LCA || FF || JT

ETH_DMATXNDESCWBF_JT

Jabber Timeout

ETH_DMATXNDESCWBF_FF

Packet Flushed: DMA/MTL flushed the packet due to SW flush

ETH_DMATXNDESCWBF_PCE

Payload Checksum Error

ETH_DMATXNDESCWBF_LCA

Loss of Carrier: carrier lost during transmission

ETH_DMATXNDESCWBF_NC

No Carrier: no carrier signal from the transceiver

ETH_DMATXNDESCWBF_LCO

Late Collision: transmission aborted due to collision

ETH_DMATXNDESCWBF_EC

Excessive Collision: transmission aborted after 16 collisions

ETH_DMATXNDESCWBF_CC

Collision Count

ETH_DMATXNDESCWBF_ED

Excessive Deferral

ETH_DMATXNDESCWBF_UF

Underflow Error: late data arrival from the memory

ETH_DMATXNDESCWBF_DB

Deferred Bit

ETH_DMATXNDESCWBF_IHE

IP Header Error

ETH_DMATXCDESC_TTSL

Transmit Packet Timestamp Low

ETH_DMATXCDESC_TTSH

Transmit Packet Timestamp High

ETH_DMATXCDESC_IVT

Inner VLAN Tag

ETH_DMATXCDESC_MSS

Maximum Segment Size

ETH_DMATXCDESC_OWN

OWN bit: descriptor is owned by DMA engine

ETH_DMATXCDESC_CTXT

Context Type

ETH_DMATXCDESC_OSTC

One-Step Timestamp Correction Enable

ETH_DMATXCDESC_TCMSSV

One-Step Timestamp Correction Input or MSS Valid

ETH_DMATXCDESC_CDE

Context Descriptor Error

ETH_DMATXCDESC_IVTIR

Inner VLAN Tag Insert or Replace Mask

ETH_DMATXCDESC_IVTIR_DISABLE

Do not add the inner VLAN tag.

ETH_DMATXCDESC_IVTIR_REMOVE

Remove the inner VLAN tag from the packets before transmission.

ETH_DMATXCDESC_IVTIR_INSERT

Insert the inner VLAN tag.

ETH_DMATXCDESC_IVTIR_REPLACE

Replace the inner VLAN tag.

ETH_DMATXCDESC_IVLTV

Inner VLAN Tag Valid

ETH_DMATXCDESC_VLTV

VLAN Tag Valid

ETH_DMATXCDESC_VT

VLAN Tag

ETH Duplex Mode

ETH_FULLDUPLEX_MODE

ETH_HALFDUPLEX_MODE

ETH Error Code

HAL_ETH_ERROR_NONE

No error

HAL_ETH_ERROR_PARAM

Busy error

HAL_ETH_ERROR_BUSY

Parameter error

HAL_ETH_ERROR_TIMEOUT

Timeout error

HAL_ETH_ERROR_MTL_Q0

MTL Q0 error

HAL_ETH_ERROR_MTL_Q1

MTL Q1 error

HAL_ETH_ERROR_MAC

MAC transfer error

HAL_ETH_ERROR_INVALID_CALLBACK

Invalid Callback error

ETH Exported Macros

__HAL_ETH_RESET_HANDLE_STATE

Description:

- Reset ETH handle state.

Parameters:

- __HANDLE__: specifies the ETH handle.

Return value:

- None

__HAL_ETH_DMA_CH_ENABLE_IT

Description:

- Enables the specified ETHERNET DMA interrupts.

Parameters:

- __HANDLE__: : ETH Handle
- __CH__: : DMA Channel
- __INTERRUPT__: specifies the ETHERNET DMA interrupt sources to be enabled

Return value:

- None

__HAL_ETH_DMA_CH_DISABLE_IT

Description:

- Disables the specified ETHERNET DMA interrupts.

Parameters:

- __HANDLE__: : ETH Handle
- __CH__: : DMA Channel
- __INTERRUPT__: specifies the ETHERNET DMA interrupt sources to be disabled.

Return value:

- None

__HAL_ETH_DMA_CH_GET_IT_SOURCE

Description:

- Gets the ETHERNET DMA IT source enabled or disabled.

Parameters:

- __HANDLE__: : ETH Handle
- __CH__: : DMA Channel
- __INTERRUPT__: specifies the interrupt source to get .

Return value:

- The: ETH DMA IT Source enabled or disabled

__HAL_ETH_DMA_CH_GET_IT

Description:

- Gets the ETHERNET DMA IT pending bit.

Parameters:

- __HANDLE__: : ETH Handle
- __CH__: : DMA Channel
- __INTERRUPT__: specifies the interrupt source to get .

Return value:

- The: state of ETH DMA IT (SET or RESET)

__HAL_ETH_DMA_CH_CLEAR_IT

Description:

- Clears the ETHERNET DMA IT pending bit.

Parameters:

- __HANDLE__: : ETH Handle
- __CH__: : DMA Channel
- __INTERRUPT__: specifies the interrupt pending bit to clear.

Return value:

- None

__HAL_ETH_DMA_GET_FLAG

Description:

- Checks whether the specified ETHERNET DMA flag is set or not.

Parameters:

- __HANDLE__: ETH Handle
- __CH__: : DMA Channel
- __FLAG__: specifies the flag to check.

Return value:

- The: state of ETH DMA FLAG (SET or RESET).

__HAL_ETH_DMA_CLEAR_FLAG

Description:

- Clears the specified ETHERNET DMA flag.

Parameters:

- __HANDLE__: ETH Handle
- __CH__: : DMA Channel
- __FLAG__: specifies the flag to check.

Return value:

- The: state of ETH DMA FLAG (SET or RESET).

__HAL_ETH_MTL_Q_ENABLE_IT

Description:

- Enables the specified ETHERNET MTL interrupts.

Parameters:

- __HANDLE__: : ETH Handle
- __Q__: : MTL Queue
- __INTERRUPT__: specifies the ETHERNET MTL interrupt sources to be enabled

Return value:

- None

__HAL_ETH_MTL_GET_IT

Description:

- Gets the ETHERNET MTL IT source enabled or disabled.

Parameters:

- __HANDLE__: : ETH Handle
- __INTERRUPT__: specifies the interrupt source to get .

Return value:

- The: ETH MTL IT Source enabled or disabled

__HAL_ETH_MTL_Q_GET_IT_SOURCE

Description:

- Gets the ETHERNET MTL IT pending bit.

Parameters:

- __HANDLE__: : ETH Handle
- __Q__: : MTL Queue
- __INTERRUPT__: specifies the interrupt source to get .

Return value:

- The: state of ETH MTL IT (SET or RESET)

__HAL_ETH_MTL_Q_DISABLE_IT

Description:

- Disables the specified ETHERNET MTL interrupts.

Parameters:

- __HANDLE__: : ETH Handle
- __Q__: : MTL Queue
- __INTERRUPT__: specifies the ETHERNET MTL interrupt sources to be disabled

Return value:

- None

__HAL_ETH_MAC_ENABLE_IT

Description:

- Enables the specified ETHERNET MAC interrupts.

Parameters:

- __HANDLE__: : ETH Handle
- __INTERRUPT__: specifies the ETHERNET MAC interrupt sources to be enabled

Return value:

- None

__HAL_ETH_MAC_DISABLE_IT

Description:

- Disables the specified ETHERNET MAC interrupts.

Parameters:

- __HANDLE__: : ETH Handle
- __INTERRUPT__: specifies the ETHERNET MAC interrupt sources to be enabled

Return value:

- None

__HAL_ETH_MAC_GET_IT

Description:

- Checks whether the specified ETHERNET MAC flag is set or not.

Parameters:

- __HANDLE__: ETH Handle
- __INTERRUPT__: specifies the flag to check.

Return value:

- The: state of ETH MAC IT (SET or RESET). External interrupt line 51 Connected to the ETH wakeup EXTI Line

ETH_WAKEUP_EXTI_LINE

__HAL_ETH_WAKEUP_EXTI_ENABLE_IT

Description:

- Enable the ETH WAKEUP Exti Line.

Parameters:

- __EXTI_LINE__: specifies the ETH WAKEUP Exti sources to be enabled.
  - ETH_WAKEUP_EXTI_LINE

Return value:

- None.

__HAL_ETH_WAKEUP_EXTI_GET_FLAG

Description:

- checks whether the specified ETH WAKEUP Exti interrupt flag is set or not.

Parameters:

- __EXTI_LINE__: specifies the ETH WAKEUP Exti sources to be cleared.
  - ETH_WAKEUP_EXTI_LINE

Return value:

- EXTI: ETH WAKEUP Line Status.

__HAL_ETH_WAKEUP_EXTI_CLEAR_FLAG

Description:

- Clear the ETH WAKEUP Exti flag.

Parameters:

- __EXTI_LINE__: specifies the ETH WAKEUP Exti sources to be cleared.
  - ETH_WAKEUP_EXTI_LINE

Return value:

- None.

__HAL_ETH_WAKEUP_EXTI_ENABLE_RISING_EDGE

Description:

- enable rising edge interrupt on selected EXTI line.

Parameters:

- __EXTI_LINE__: specifies the ETH WAKEUP EXTI sources to be disabled.
  - ETH_WAKEUP_EXTI_LINE

Return value:

- None

__HAL_ETH_WAKEUP_EXTI_ENABLE_FALLING_EDGE

Description:

- enable falling edge interrupt on selected EXTI line.

Parameters:

- __EXTI_LINE__: specifies the ETH WAKEUP EXTI sources to be disabled.
  - ETH_WAKEUP_EXTI_LINE

Return value:

- None

__HAL_ETH_WAKEUP_EXTI_ENABLE_RISING_FALLING_EDGE

Description:

- enable falling edge interrupt on selected EXTI line.

Parameters:

- __EXTI_LINE__: specifies the ETH WAKEUP EXTI sources to be disabled.
  - ETH_WAKEUP_EXTI_LINE

Return value:

- None

__HAL_ETH_WAKEUP_EXTI_GENERATE_SWIT

Description:

- Generates a Software interrupt on selected EXTI line.

Parameters:

- __EXTI_LINE__: specifies the ETH WAKEUP EXTI sources to be disabled.
  - ETH_WAKEUP_EXTI_LINE

Return value:

- None

__HAL_ETH_GET_PTP_CONTROL

__HAL_ETH_SET_PTP_CONTROL

ETH frame settings

ETH_MAX_PACKET_SIZE

ETH_HEADER + 2*VLAN_TAG + MAX_ETH_PAYLOAD + ETH_CRC

ETH_HEADER

6 byte Dest addr, 6 byte Src addr, 2 byte length/type

ETH_CRC

Ethernet CRC

ETH_VLAN_TAG

optional 802.1q VLAN Tag

ETH_MIN_PAYLOAD

Minimum Ethernet payload size

ETH_MAX_PAYLOAD

Maximum Ethernet payload size

ETH_JUMBO_FRAME_PAYLOAD

Jumbo frame payload size

ETH Inter Packet Gap

ETH_INTERPACKETGAP_96BIT

ETH_INTERPACKETGAP_88BIT

ETH_INTERPACKETGAP_80BIT

ETH_INTERPACKETGAP_72BIT

ETH_INTERPACKETGAP_64BIT

ETH_INTERPACKETGAP_56BIT

ETH_INTERPACKETGAP_48BIT

ETH_INTERPACKETGAP_40BIT

ETH MAC addresses

ETH_MAC_ADDRESS0

ETH_MAC_ADDRESS1

ETH_MAC_ADDRESS2

ETH_MAC_ADDRESS3

ETH MAC Interrupts

ETH_MAC_RX_STATUS_IT

ETH_MAC_TX_STATUS_IT

ETH_MAC_TIMESTAMP_IT

ETH_MAC_LPI_IT

ETH_MAC_PMT_IT

ETH_MAC_PHY_IT

ETH MAC Rx Tx Status

ETH_RECEIVE_WATCHDOG_TIMEOUT

ETH_EXECESSIVE_COLLISIONS

ETH_LATE_COLLISIONS

ETH_EXECESSIVE_DEFERRAL

ETH_LOSS_OF_CARRIER

ETH_NO_CARRIER

ETH_TRANSMIT_JABBR_TIMEOUT

ETH MAC Wake Up Event

ETH_WAKEUP_PACKET_RECIEVED

ETH_MAGIC_PACKET_RECIEVED

ETH MTL Interrupts

ETH_RX_QUEUE_OVERFLOW_IT

ETH_AVERAGE_BITS_PER_SLOT_IT

ETH_TX_QUEUE_UNDERFLOW_IT

ETH Pause Low Threshold

ETH_PAUSELOWTHRESHOLD_MINUS_4

ETH_PAUSELOWTHRESHOLD_MINUS_28

ETH_PAUSELOWTHRESHOLD_MINUS_36

ETH_PAUSELOWTHRESHOLD_MINUS_144

ETH_PAUSELOWTHRESHOLD_MINUS_256

ETH_PAUSELOWTHRESHOLD_MINUS_512

ETH Preamble Length

ETH_PREAMBLELENGTH_7

ETH_PREAMBLELENGTH_5

ETH_PREAMBLELENGTH_3

ETH PTP Config Status

HAL_ETH_PTP_NOT_CONFIGURED
: ETH PTP Configuration not done

HAL_ETH_PTP_CONFIGURED
: ETH PTP Configuration done

ETH Queue1 AV Algo

ETH_TX_QUEUE_AV_ALGO_CBS

ETH_TX_QUEUE_AV_ALGO_SP

ETH Queue Mapped DMA Channel

ETH_MTL_Q0_MAPPED_TO_DMA_CH0

ETH_MTL_Q0_MAPPED_TO_DMA_CH1

ETH_MTL_Q1_MAPPED_TO_DMA_CH0

ETH_MTL_Q1_MAPPED_TO_DMA_CH1

ETH Queue Operation Mode

ETH_TX_QUEUE_ENABLED

ETH_TX_QUEUE_DISABLED

ETH_TX_QUEUE_AV_ENABLED

ETH_RX_QUEUE0_ENABLED

ETH_RX_QUEUE0_DISABLED

ETH_RX_QUEUE0_AV_ENABLED

ETH_RX_QUEUE1_ENABLED

ETH_RX_QUEUE1_DISABLED

ETH_RX_QUEUE1_AV_ENABLED

ETH Receive Mode

ETH_RECEIVESTOREFORWARD

ETH_RECEIVETHRESHOLD8_64

ETH_RECEIVETHRESHOLD8_32

ETH_RECEIVETHRESHOLD8_96

ETH_RECEIVETHRESHOLD8_128

ETH Receive Queue Size

ETH_RECEIVE_QUEUE_SIZE_256

ETH_RECEIVE_QUEUE_SIZE_2048

ETH_RECEIVE_QUEUE_SIZE_4096

ETH Rx Checksum Status

ETH_CHECKSUM_BYPASSED

ETH_CHECKSUM_IP_HEADER_ERROR

ETH_CHECKSUM_IP_PAYLOAD_ERROR

ETH Rx DMA Burst Length

ETH_RXDMABURSTLENGTH_1BEAT

ETH_RXDMABURSTLENGTH_2BEAT

ETH_RXDMABURSTLENGTH_4BEAT

ETH_RXDMABURSTLENGTH_8BEAT

ETH_RXDMABURSTLENGTH_16BEAT

ETH_RXDMABURSTLENGTH_32BEAT

ETH Rx Error Code

ETH_DRIBBLE_BIT_ERROR

ETH_RECEIVE_ERROR

ETH_RECEIVE_OVERFLOW

ETH_WATCHDOG_TIMEOUT

ETH_GIANT_PACKET

ETH_CRC_ERROR

ETH Rx IP Header Type

ETH_IP_HEADER_IPV4

ETH_IP_HEADER_IPV6

ETH Rx L3 Filter Status

ETH_L3_FILTER0_MATCH

ETH_L3_FILTER1_MATCH

ETH Rx L4 Filter Status

ETH_L4_FILTER0_MATCH

ETH_L4_FILTER1_MATCH

ETH Rx MAC Filter Status

ETH_HASH_FILTER_PASS

ETH_VLAN_FILTER_PASS

ETH_DEST_ADDRESS_FAIL

ETH_SOURCE_ADDRESS_FAIL

ETH Rx maximum read outstanding request limit

ETH_RX_OSR_LIMIT_1

ETH_RX_OSR_LIMIT_2

ETH_RX_OSR_LIMIT_3

ETH Rx Payload Type

ETH_IP_PAYLOAD_UNKNOWN

ETH_IP_PAYLOAD_UDP

ETH_IP_PAYLOAD_TCP

ETH_IP_PAYLOAD_ICMPN

ETH RX Queue

ETH_RX_QUEUE0

ETH_RX_QUEUE1

ETH Source Addr Control

ETH_SOURCEADDRESS_DISABLE

ETH_SOURCEADDRESS_INSERT_ADDR0

ETH_SOURCEADDRESS_INSERT_ADDR1

ETH_SOURCEADDRESS_REPLACE_ADDR0

ETH_SOURCEADDRESS_REPLACE_ADDR1

ETH Speed

ETH_SPEED_10M

ETH_SPEED_100M

ETH_SPEED_1000M

ETH States

HAL_ETH_STATE_RESET

Peripheral not yet Initialized or disabled

HAL_ETH_STATE_READY

Peripheral Communication started

HAL_ETH_STATE_BUSY

an internal process is ongoing

HAL_ETH_STATE_STARTED

an internal process is started

HAL_ETH_STATE_ERROR

Error State

ETH Transmit Mode

ETH_TRANSMITSTOREFORWARD

ETH_TRANSMITTHRESHOLD_32

ETH_TRANSMITTHRESHOLD_64

ETH_TRANSMITTHRESHOLD_96

ETH_TRANSMITTHRESHOLD_128

ETH_TRANSMITTHRESHOLD_192

ETH_TRANSMITTHRESHOLD_256

ETH_TRANSMITTHRESHOLD_384

ETH_TRANSMITTHRESHOLD_512

ETH Transmit Queue Size

ETH_TRANSMIT_QUEUE_SIZE_256

ETH_TRANSMIT_QUEUE_SIZE_2048

ETH_TRANSMIT_QUEUE_SIZE_4096

ETH Tx DMA Burst Length

ETH_TXDMABURSTLENGTH_1BEAT

ETH_TXDMABURSTLENGTH_2BEAT

ETH_TXDMABURSTLENGTH_4BEAT

ETH_TXDMABURSTLENGTH_8BEAT

ETH_TXDMABURSTLENGTH_16BEAT

ETH_TXDMABURSTLENGTH_32BEAT

ETH Tx DMA Channel Weight

ETH_TXCHANNELWEIGHT_1

ETH_TXCHANNELWEIGHT_2

ETH_TXCHANNELWEIGHT_3

ETH_TXCHANNELWEIGHT_4

ETH_TXCHANNELWEIGHT_5

ETH_TXCHANNELWEIGHT_6

ETH_TXCHANNELWEIGHT_7

ETH Tx maximum write outstanding request limit

ETH_TX_OSR_LIMIT_1

ETH_TX_OSR_LIMIT_2

ETH_TX_OSR_LIMIT_3

ETH Tx Packet Attributes

ETH_TX_PACKETS_FEATURES_CSUM

ETH_TX_PACKETS_FEATURES_SAIC

ETH_TX_PACKETS_FEATURES_VLANTAG

ETH_TX_PACKETS_FEATURES_INNERVLANTAG

ETH_TX_PACKETS_FEATURES_TSO

ETH_TX_PACKETS_FEATURES_CRCPAD

ETH Tx Packet Checksum Control

ETH_CHECKSUM_DISABLE

ETH_CHECKSUM_IPHDR_INSERT

ETH_CHECKSUM_IPHDR_PAYLOAD_INSERT

ETH_CHECKSUM_IPHDR_PAYLOAD_INSERT_PHDR_CALC

ETH Tx Packet CRC Pad Control

ETH_CRC_PAD_DISABLE

ETH_CRC_PAD_INSERT

ETH_CRC_INSERT

ETH_CRC_REPLACE

ETH Tx Packet Inner VLAN Control

ETH_INNER_VLAN_DISABLE

ETH_INNER_VLAN_REMOVE

ETH_INNER_VLAN_INSERT

ETH_INNER_VLAN_REPLACE

ETH Tx Packet Source Addr Control

ETH_SRC_ADDR_CONTROL_DISABLE

ETH_SRC_ADDR_INSERT

ETH_SRC_ADDR_REPLACE

ETH Tx Packet VLAN Control

ETH_VLAN_DISABLE

ETH_VLAN_REMOVE

ETH_VLAN_INSERT

ETH_VLAN_REPLACE

ETH VLAN Tag Comparison

ETH_VLANTAGCOMPARISON_16BIT

ETH_VLANTAGCOMPARISON_12BIT

ETH Watchdog Timeout

ETH_WATCHDOGTIMEOUT_2KB

ETH_WATCHDOGTIMEOUT_3KB

ETH_WATCHDOGTIMEOUT_4KB

ETH_WATCHDOGTIMEOUT_5KB

ETH_WATCHDOGTIMEOUT_6KB

ETH_WATCHDOGTIMEOUT_7KB

ETH_WATCHDOGTIMEOUT_8KB

ETH_WATCHDOGTIMEOUT_9KB

ETH_WATCHDOGTIMEOUT_10KB

ETH_WATCHDOGTIMEOUT_11KB

ETH_WATCHDOGTIMEOUT_12KB

ETH_WATCHDOGTIMEOUT_13KB

ETH_WATCHDOGTIMEOUT_14KB

ETH_WATCHDOGTIMEOUT_15KB

ETH_WATCHDOGTIMEOUT_16KB

23 HAL ETH Extension Driver

23.1 ETHEx Firmware driver registers structures

23.1.1 ETH_RxVLANConfigTypeDef

ETH_RxVLANConfigTypeDef is defined in the stm32n6xx_hal_eth_ex.h

Data Fields

- ***FunctionalState InnerVLANTagInStatus***
- ***uint32_t StripInnerVLANTag***
- ***FunctionalState InnerVLANTag***
- ***FunctionalState DoubleVLANProcessing***
- ***FunctionalState VLANTagHashTableMatch***
- ***FunctionalState VLANTagInStatus***
- ***uint32_t StripVLANTag***
- ***uint32_t VLANTypeCheck***
- ***FunctionalState VLANTagInverseMatch***
- ***FunctionalState BitVLANcomparison***
- ***uint32_t VLANcomparison***
- ***FunctionalState VLANTagEnable***
- ***uint32_t VLANTagID***
- ***uint32_t RxDMAChannelNumber***
- ***FunctionalState DMAChannelNumberEnable***

Field Documentation

- ***FunctionalState ETH_RxVLANConfigTypeDef::InnerVLANTagInStatus***
  Enables or disables Inner VLAN Tag in Rx Status
- ***uint32_t ETH_RxVLANConfigTypeDef::StripInnerVLANTag***
  Sets the Inner VLAN Tag Stripping on Receive This parameter can be a value of ***ETHEx_Rx_Inner_VLAN_Tag_Stripping***
- ***FunctionalState ETH_RxVLANConfigTypeDef::InnerVLANTag***
  Enables or disables Inner VLAN Tag
- ***FunctionalState ETH_RxVLANConfigTypeDef::DoubleVLANProcessing***
  Enable or Disable double VLAN processing
- ***FunctionalState ETH_RxVLANConfigTypeDef::VLANTagHashTableMatch***
  Enable or Disable VLAN Tag Hash Table Match
- ***FunctionalState ETH_RxVLANConfigTypeDef::VLANTagInStatus***
  Enable or Disable VLAN Tag in Rx status
- ***uint32_t ETH_RxVLANConfigTypeDef::StripVLANTag***
  Set the VLAN Tag Stripping on Receive This parameter can be a value of ***ETHEx_Rx_VLAN_Tag_Stripping***
- ***uint32_t ETH_RxVLANConfigTypeDef::VLANTypeCheck***
  Enable or Disable VLAN Type Check This parameter can be a value of ***ETHEx_VLAN_Type_Check***
- ***FunctionalState ETH_RxVLANConfigTypeDef::VLANTagInverseMatch***
  Enable or disable VLAN Tag Inverse Match
- ***FunctionalState ETH_RxVLANConfigTypeDef::BitVLANcomparison***
  Enable 12-Bit VLAN Tag Comparison
- ***uint32_t ETH_RxVLANConfigTypeDef::VLANcomparison***
  12-bit or 16-bit VLAN comparison
- ***FunctionalState ETH_RxVLANConfigTypeDef::VLANTagEnable***
  VLAN Tag Enable
- ***uint32_t ETH_RxVLANConfigTypeDef::VLANTagID***
  VLAN Tag ID

- ***uint32_t ETH_RxVLANConfigTypeDef::RxDMAChannelNumber***
  Rx DMA Channel Number
- ***FunctionalState ETH_RxVLANConfigTypeDef::DMAChannelNumberEnable***
  Enable Rx DMA Channel Number

23.1.2 ETH_MACMTLMappingTypeDef

ETH_MACMTLMappingTypeDef is defined in the stm32n6xx_hal_eth_ex.h

Data Fields

- ***uint32_t VLANTagFilterFailPacketsQueue***
- ***FunctionalState VLANTagFilterFailPacketsQueuingEnable***
- ***uint32_t MulticastAddFilterFailPacketsQueue***
- ***FunctionalState MulticastAddrFilterFailPacketsQueuingEnable***
- ***uint32_t UnicastAddrFilterFailPacketsQueue***
- ***FunctionalState UnicastAddrFilterFailPacketsQueuingEnable***
- ***FunctionalState TypeFieldBasedRxQueuingEnable***
- ***uint32_t OverridingMCBCQueuePrioritySelect***
- ***uint32_t FramePreemptionResidueQueue***
- ***uint32_t TaggedPTPoEPacketsQueuingControl***
- ***FunctionalState TaggedAVControlPacketsQueuingEnable***
- ***FunctionalState MulticastBroadcastQueueEnable***
- ***uint32_t MulticastBroadcastQueue***
- ***uint32_t UntaggedPacketQueue***
- ***uint32_t PTPPacketsQueue***
- ***uint32_t AVUntaggedControlPacketsQueue***
- ***uint32_t PrioritiesSelectedRxQ0***
- ***uint32_t PrioritiesSelectedRxQ1***

Field Documentation

- ***uint32_t ETH_MACMTLMappingTypeDef::VLANTagFilterFailPacketsQueue***
  Specifies the Rx queue to which the tagged packets fail DA/SA/VLANtag filter must be routed to
- ***FunctionalState ETH_MACMTLMappingTypeDef::VLANTagFilterFailPacketsQueuingEnable***
  Enables routing the tagged packets fail DA/SA/VLANtag filter to Rx queue
- ***uint32_t ETH_MACMTLMappingTypeDef::MulticastAddFilterFailPacketsQueue***
  Specifies the Rx queue to which the Multicast packets fail the DA/SA filter are routed to
- ***FunctionalState ETH_MACMTLMappingTypeDef::MulticastAddrFilterFailPacketsQueuingEnable***
  Enables routing Multicast Packets fail DA/SA filter to Rx Queue programmed in MFFQ
- ***uint32_t ETH_MACMTLMappingTypeDef::UnicastAddrFilterFailPacketsQueue***
  Specifies the Rx queue to which the Unicast packets fail the DA/SA filter are routed to
- ***FunctionalState ETH_MACMTLMappingTypeDef::UnicastAddrFilterFailPacketsQueuingEnable***
  Enables routing Unicast Packets fail DA/SA filter to Rx Queue programmed in UFFQ
- ***FunctionalState ETH_MACMTLMappingTypeDef::TypeFieldBasedRxQueuingEnable***
  Enables Type field based Rx queuing
- ***uint32_t ETH_MACMTLMappingTypeDef::OverridingMCBCQueuePrioritySelect***
  Select the Overriding MC-BC queue priority
- ***uint32_t ETH_MACMTLMappingTypeDef::FramePreemptionResidueQueue***
  Specifies the Rx queue to which the residual preemption frames must be forwarded
- ***uint32_t ETH_MACMTLMappingTypeDef::TaggedPTPoEPacketsQueuingControl***
  Specifies the routing of the VLAN tagged PTPoE packets
- ***FunctionalState ETH_MACMTLMappingTypeDef::TaggedAVControlPacketsQueuingEnable***
  Enable MAC routes the received Tagged AV control packets to Rx queue specified by AVCPQ field
- ***FunctionalState ETH_MACMTLMappingTypeDef::MulticastBroadcastQueueEnable***
  Specifies that Multicast or Broadcast packets routing to the Rx queue is enabled
- ***uint32_t ETH_MACMTLMappingTypeDef::MulticastBroadcastQueue***
  Specifies the Rx queue onto which Multicast or Broadcast packets are routed

- ***uint32_t ETH_MACMTLMappingTypeDef::UntaggedPacketQueue***
  Specifies the Rx queue to which Untagged Packets are to be routed
- ***uint32_t ETH_MACMTLMappingTypeDef::PTPPacketsQueue***
  Specifies the Rx queue on which the PTP packets sent over the Ethernet payload
- ***uint32_t ETH_MACMTLMappingTypeDef::AVUntaggedControlPacketsQueue***
  Specifies the Receive queue to receive AV untagged control packets
- ***uint32_t ETH_MACMTLMappingTypeDef::PrioritiesSelectedRxQ0***
  Specifies the Priorities Selected in the Receive Queue 0 ***ETHEx_Rx_VLAN_PRIO***
- ***uint32_t ETH_MACMTLMappingTypeDef::PrioritiesSelectedRxQ1***
  Specifies the Priorities Selected in the Receive Queue 1 ***ETHEx_Rx_VLAN_PRIO***

23.1.3 MTL_TxQueueInstance_t

MTL_TxQueueInstance_t is defined in the stm32n6xx_hal_eth_ex.h

Data Fields

- ***uint32_t QueueOpMode***
- ***uint32_t AVAlgorithm***
- ***uint32_t TxQueueSize***
- ***uint32_t TransmitQueueMode***

Field Documentation

- ***uint32_t MTL_TxQueueInstance_t::QueueOpMode***
  Queue Disabled, Enabled or AV Mode.
- ***uint32_t MTL_TxQueueInstance_t::AVAlgorithm***
  Queue Disabled, Enabled or AV Mode.
- ***uint32_t MTL_TxQueueInstance_t::TxQueueSize***
  Specifies the Tx Queue Size
- ***uint32_t MTL_TxQueueInstance_t::TransmitQueueMode***
  Specifies the Transmit Queue operating mode.

23.1.4 MTL_RxQueueInstance_t

MTL_RxQueueInstance_t is defined in the stm32n6xx_hal_eth_ex.h

Data Fields

- ***uint32_t QueueOpMode***
- ***uint32_t RxQueueSize***
- ***FunctionalState DropTCPIPChecksumErrorPacket***
- ***FunctionalState ForwardRxErrorPacket***
- ***FunctionalState ForwardRxUndersizedGoodPacket***
- ***uint32_t ReceiveQueueMode***
- ***uint32_t MappedToDMACh***

Field Documentation

- ***uint32_t MTL_RxQueueInstance_t::QueueOpMode***
  Queue Disabled, Enabled or AV Mode.
- ***uint32_t MTL_RxQueueInstance_t::RxQueueSize***
  Specifies the Rx Queue Size
- ***FunctionalState MTL_RxQueueInstance_t::DropTCPIPChecksumErrorPacket***
  Enables or disables Dropping of TCPIP Checksum Error Packets.
- ***FunctionalState MTL_RxQueueInstance_t::ForwardRxErrorPacket***
  Enables or disables forwarding Error Packets.
- ***FunctionalState MTL_RxQueueInstance_t::ForwardRxUndersizedGoodPacket***
  Enables or disables forwarding Undersized Good Packets.
- ***uint32_t MTL_RxQueueInstance_t::ReceiveQueueMode***
  Specifies the Receive Queue operating mode
- ***uint32_t MTL_RxQueueInstance_t::MappedToDMACh***
  Specifies the DMA Channel to which MTL Q is mapped

23.1.5 ETH_MTLConfigTypeDef

ETH_MTLConfigTypeDef is defined in the stm32n6xx_hal_eth_ex.h

Data Fields

- ***uint32_t TxSchedulingAlgorithm***
- ***uint32_t ReceiveArbitrationAlgorithm***
- ***MTL_TxQueueInstance_t TxQ***
- ***MTL_RxQueueInstance_t RxQ***
- ***FunctionalState TransmitStatus***

Field Documentation

- ***uint32_t ETH_MTLConfigTypeDef::TxSchedulingAlgorithm***
  Specifies the algorithm for Tx scheduling
- ***uint32_t ETH_MTLConfigTypeDef::ReceiveArbitrationAlgorithm***
  Specifies the arbitration algorithm for the Rx side
- ***MTL_TxQueueInstance_t ETH_MTLConfigTypeDef::TxQ[ETH_MTL_TX_Q_CNT]***
  MTL Tx Queue Configuration
- ***MTL_RxQueueInstance_t ETH_MTLConfigTypeDef::RxQ[ETH_MTL_RX_Q_CNT]***
  MTL Rx Queue Configuration
- ***FunctionalState ETH_MTLConfigTypeDef::TransmitStatus***
  Enables or disables forwarding Tx Packet Status to the application.

23.1.6 ETH_PacketTypeQueueConfigTypeDef

ETH_PacketTypeQueueConfigTypeDef is defined in the stm32n6xx_hal_eth_ex.h

Data Fields

- ***ETH_IDRegTypeDef Address***
- ***uint32_t Type***
- ***uint32_t Queue***
- ***uint32_t Preemption***

Field Documentation

- ***ETH_IDRegTypeDef ETH_PacketTypeQueueConfigTypeDef::Address***
  Sets Address Offset for indirect accesses to ETH_MAC_TMRQR
- ***uint32_t ETH_PacketTypeQueueConfigTypeDef::Type***
  Indicates the type value of packet that needs to be compared with the received packet.
- ***uint32_t ETH_PacketTypeQueueConfigTypeDef::Queue***
  Indicates the receive queue number to which the packet needs to be forwarded
- ***uint32_t ETH_PacketTypeQueueConfigTypeDef::Preemption***
  Preemption or Express Packet This parameter can be a value of ***ETHEx_Preemption_Packet***

23.1.7 ETH_TxVLANConfigTypeDef

ETH_TxVLANConfigTypeDef is defined in the stm32n6xx_hal_eth_ex.h

Data Fields

- ***FunctionalState SourceTxDesc***
- ***FunctionalState SVLANType***
- ***uint32_t VLANTagControl***

Field Documentation

- ***FunctionalState ETH_TxVLANConfigTypeDef::SourceTxDesc***
  Enable or Disable VLAN tag source from DMA tx descriptors
- ***FunctionalState ETH_TxVLANConfigTypeDef::SVLANType***
  Enable or Disable insertion of SVLAN type
- ***uint32_t ETH_TxVLANConfigTypeDef::VLANTagControl***
  Sets the VLAN tag control in tx packets This parameter can be a value of ***ETHEx_VLAN_Tag_Control***

23.1.8 ETH_L3FilterConfigTypeDef

ETH_L3FilterConfigTypeDef is defined in the stm32n6xx_hal_eth_ex.h

Data Fields

- ***uint32_t Protocol***
- ***uint32_t SrcAddrFilterMatch***
- ***uint32_t DestAddrFilterMatch***
- ***uint32_t SrcAddrHigherBitsMatch***
- ***uint32_t DestAddrHigherBitsMatch***
- ***uint32_t Ip4SrcAddr***
- ***uint32_t Ip4DestAddr***
- ***uint32_t Ip6Addr***

Field Documentation

- ***uint32_t ETH_L3FilterConfigTypeDef::Protocol***
  Sets the L3 filter protocol to IPv4 or IPv6 This parameter can be a value of ***ETHEx_L3_Protocol***
- ***uint32_t ETH_L3FilterConfigTypeDef::SrcAddrFilterMatch***
  Sets the L3 filter source address match This parameter can be a value of ***ETHEx_L3_Source_Match***
- ***uint32_t ETH_L3FilterConfigTypeDef::DestAddrFilterMatch***
  Sets the L3 filter destination address match This parameter can be a value of ***ETHEx_L3_Destination_Match***
- ***uint32_t ETH_L3FilterConfigTypeDef::SrcAddrHigherBitsMatch***
  Sets the L3 filter source address higher bits match This parameter can be a value from 0 to 31
- ***uint32_t ETH_L3FilterConfigTypeDef::DestAddrHigherBitsMatch***
  Sets the L3 filter destination address higher bits match This parameter can be a value from 0 to 31
- ***uint32_t ETH_L3FilterConfigTypeDef::Ip4SrcAddr***
  Sets the L3 filter IPv4 source address if IPv4 protocol is used This parameter can be a value from 0x0 to 0xFFFFFFFF
- ***uint32_t ETH_L3FilterConfigTypeDef::Ip4DestAddr***
  Sets the L3 filter IPv4 destination address if IPv4 protocol is used This parameter can be a value from 0 to 0xFFFFFFFF
- ***uint32_t ETH_L3FilterConfigTypeDef::Ip6Addr[4]***
  Sets the L3 filter IPv6 address if IPv6 protocol is used This parameter must be a table of 4 words (4* 32 bits)

23.1.9 ETH_L4FilterConfigTypeDef

ETH_L4FilterConfigTypeDef is defined in the stm32n6xx_hal_eth_ex.h

Data Fields

- ***uint32_t Protocol***
- ***uint32_t SrcPortFilterMatch***
- ***uint32_t DestPortFilterMatch***
- ***uint32_t SourcePort***
- ***uint32_t DestinationPort***

Field Documentation

- ***uint32_t ETH_L4FilterConfigTypeDef::Protocol***
  Sets the L4 filter protocol to TCP or UDP This parameter can be a value of ***ETHEx_L4_Protocol***
- ***uint32_t ETH_L4FilterConfigTypeDef::SrcPortFilterMatch***
  Sets the L4 filter source port match This parameter can be a value of ***ETHEx_L4_Source_Match***
- ***uint32_t ETH_L4FilterConfigTypeDef::DestPortFilterMatch***
  Sets the L4 filter destination port match This parameter can be a value of ***ETHEx_L4_Destination_Match***
- ***uint32_t ETH_L4FilterConfigTypeDef::SourcePort***
  Sets the L4 filter source port This parameter must be a value from 0x0 to 0xFFFF
- ***uint32_t ETH_L4FilterConfigTypeDef::DestinationPort***
  Sets the L4 filter destination port This parameter must be a value from 0x0 to 0xFFFF

23.2 ETHEx Firmware driver API description

The following section lists the various functions of the ETHEx library.

23.2.1 Detailed description of functions

HAL_ETHEx_EnableARPOffload

Function name

void HAL_ETHEx_EnableARPOffload (ETH_HandleTypeDef * heth)

Function description

HAL_ETHEx_DisableARPOffload

Function name

void HAL_ETHEx_DisableARPOffload (ETH_HandleTypeDef * heth)

Function description

HAL_ETHEx_SetARPAddressMatch

Function name

void HAL_ETHEx_SetARPAddressMatch (ETH_HandleTypeDef * heth, uint32_t IpAddress)

Function description

HAL_ETHEx_EnableL3L4Filtering

Function name

void HAL_ETHEx_EnableL3L4Filtering (ETH_HandleTypeDef * heth)

Function description

HAL_ETHEx_DisableL3L4Filtering

Function name

void HAL_ETHEx_DisableL3L4Filtering (ETH_HandleTypeDef * heth)

Function description

HAL_ETHEx_GetL3FilterConfig

Function name

HAL_StatusTypeDef HAL_ETHEx_GetL3FilterConfig (const ETH_HandleTypeDef * heth, uint32_t Filter, ETH_L3FilterConfigTypeDef * pL3FilterConfig)

Function description

HAL_ETHEx_GetL4FilterConfig

Function name

HAL_StatusTypeDef HAL_ETHEx_GetL4FilterConfig (const ETH_HandleTypeDef * heth, uint32_t Filter, ETH_L4FilterConfigTypeDef * pL4FilterConfig)

Function description

HAL_ETHEx_SetL3FilterConfig

Function name

HAL_StatusTypeDef HAL_ETHEx_SetL3FilterConfig (ETH_HandleTypeDef * heth, uint32_t Filter, const ETH_L3FilterConfigTypeDef * pL3FilterConfig)

Function description

HAL_ETHEx_SetL4FilterConfig

Function name

HAL_StatusTypeDef HAL_ETHEx_SetL4FilterConfig (ETH_HandleTypeDef * heth, uint32_t Filter, const ETH_L4FilterConfigTypeDef * pL4FilterConfig)

Function description

HAL_ETHEx_EnableVLANProcessing

Function name

void HAL_ETHEx_EnableVLANProcessing (ETH_HandleTypeDef * heth)

Function description

HAL_ETHEx_DisableVLANProcessing

Function name

void HAL_ETHEx_DisableVLANProcessing (ETH_HandleTypeDef * heth)

Function description

HAL_ETHEx_GetRxVLANConfig

Function name

HAL_StatusTypeDef HAL_ETHEx_GetRxVLANConfig (const ETH_HandleTypeDef * heth, ETH_RxVLANConfigTypeDef * pVlanConfig)

Function description

HAL_ETHEx_SetRxVLANConfig

Function name

HAL_StatusTypeDef HAL_ETHEx_SetRxVLANConfig (ETH_HandleTypeDef * heth, ETH_RxVLANConfigTypeDef * pVlanConfig)

Function description

HAL_ETHEx_SetVLANHashTable

Function name

void HAL_ETHEx_SetVLANHashTable (ETH_HandleTypeDef * heth, uint32_t VLANHashTable)

Function description

HAL_ETHEx_GetTxVLANConfig

Function name

HAL_StatusTypeDef HAL_ETHEx_GetTxVLANConfig (const ETH_HandleTypeDef * heth, uint32_t VLANTag, ETH_TxVLANConfigTypeDef * pVlanConfig)

Function description

HAL_ETHEx_SetTxVLANConfig

Function name

HAL_StatusTypeDef HAL_ETHEx_SetTxVLANConfig (ETH_HandleTypeDef * heth, uint32_t VLANTag, const ETH_TxVLANConfigTypeDef * pVlanConfig)

Function description

HAL_ETHEx_SetTxVLANIdentifier

Function name

void HAL_ETHEx_SetTxVLANIdentifier (ETH_HandleTypeDef * heth, uint32_t VLANTag, uint32_t VLANIdentifier)

Function description

HAL_ETHEx_EnterLPIMode

Function name

void HAL_ETHEx_EnterLPIMode (ETH_HandleTypeDef * heth, FunctionalState TxAutomate, FunctionalState TxClockStop)

Function description

HAL_ETHEx_ExitLPIMode

Function name

void HAL_ETHEx_ExitLPIMode (ETH_HandleTypeDef * heth)

Function description

HAL_ETHEx_GetMACLPIEvent

Function name

uint32_t HAL_ETHEx_GetMACLPIEvent (const ETH_HandleTypeDef * heth)

Function description

HAL_ETHEx_GetRxDMAChNumber

Function name

uint32_t HAL_ETHEx_GetRxDMAChNumber (const ETH_HandleTypeDef * heth)

Function description

HAL_ETHEx_GetTxDMAChNumber

Function name

uint32_t HAL_ETHEx_GetTxDMAChNumber (const ETH_HandleTypeDef * heth)

Function description

HAL_ETHEx_GetRxMTLQNumber

Function name

uint32_t HAL_ETHEx_GetRxMTLQNumber (const ETH_HandleTypeDef * heth)

Function description

HAL_ETHEx_GetTxMTLQNumber

Function name

uint32_t HAL_ETHEx_GetTxMTLQNumber (const ETH_HandleTypeDef * heth)

Function description

HAL_ETHEx_GetMTLConfig

Function name

HAL_StatusTypeDef HAL_ETHEx_GetMTLConfig (const ETH_HandleTypeDef * heth, ETH_MTLConfigTypeDef * mtlconf)

Function description

HAL_ETHEx_SetMTLConfig

Function name

HAL_StatusTypeDef HAL_ETHEx_SetMTLConfig (ETH_HandleTypeDef * heth, ETH_MTLConfigTypeDef * mtlconf)

Function description

HAL_ETHEx_SetMACMTLMappingConfig

Function name

HAL_StatusTypeDef HAL_ETHEx_SetMACMTLMappingConfig (ETH_HandleTypeDef * heth, const ETH_MACMTLMappingTypeDef * macmtlconf)

Function description

HAL_ETHEx_GetMACMTLMappingConfig

Function name

HAL_StatusTypeDef HAL_ETHEx_GetMACMTLMappingConfig (const ETH_HandleTypeDef * heth, ETH_MACMTLMappingTypeDef * macmtlconf)

Function description

ETHEx_SetMTLConfig

Function name

void ETHEx_SetMTLConfig (ETH_HandleTypeDef * heth, const ETH_MTLConfigTypeDef * mtlconf)

Function description

ETHEx_SetMACMTLMappingConfig

Function name

HAL_StatusTypeDef ETHEx_SetMACMTLMappingConfig (ETH_HandleTypeDef * heth, const ETH_MACMTLMappingTypeDef * macmtlconf)

Function description

HAL_ETHEx_SetUserTagPriorityQueue

Function name

HAL_StatusTypeDef HAL_ETHEx_SetUserTagPriorityQueue (ETH_HandleTypeDef * heth, uint32_t psrq, uint32_t queue)

Function description

HAL_ETHEx_GetUserTagPriorityQueue

Function name

HAL_StatusTypeDef HAL_ETHEx_GetUserTagPriorityQueue (const ETH_HandleTypeDef * heth, uint32_t * psrq, uint32_t queue)

Function description

HAL_ETHEx_SetPacketTypeQueue

Function name

HAL_StatusTypeDef HAL_ETHEx_SetPacketTypeQueue (ETH_HandleTypeDef * heth, const ETH_PacketTypeQueueConfigTypeDef * typequeueconf)

Function description

HAL_ETHEx_GetPacketTypeQueue

Function name

HAL_StatusTypeDef HAL_ETHEx_GetPacketTypeQueue (ETH_HandleTypeDef * heth, ETH_PacketTypeQueueConfigTypeDef * typequeueconf)

Function description

23.3 ETHEx Firmware driver defines

The following section lists the various define and macros of the module.

23.3.1 ETHEx

ETHEx

ETHEx Command Type

ETH_WRITE_OPERATION

ETH_READ_OPERATION

ETHEx L3 Destination Match

ETH_L3_DEST_ADDR_PERFECT_MATCH_ENABLE

ETH_L3_DEST_ADDR_INVERSE_MATCH_ENABLE

ETH_L3_DEST_ADDR_MATCH_DISABLE

ETHEx L3 Filter

ETH_L3_FILTER_0

ETH_L3_FILTER_1

ETHEx L3 Protocol

ETH_L3_IPV6_MATCH

ETH_L3_IPV4_MATCH

ETHEx L3 Source Match

ETH_L3_SRC_ADDR_PERFECT_MATCH_ENABLE

ETH_L3_SRC_ADDR_INVERSE_MATCH_ENABLE

ETH_L3_SRC_ADDR_MATCH_DISABLE

ETHEx L4 Destination Match

ETH_L4_DEST_PORT_PERFECT_MATCH_ENABLE

ETH_L4_DEST_PORT_INVERSE_MATCH_ENABLE

ETH_L4_DEST_PORT_MATCH_DISABLE

ETHEx L4 Filter

ETH_L4_FILTER_0

ETH_L4_FILTER_1

ETHEx L4 Protocol

ETH_L4_UDP_MATCH

ETH_L4_TCP_MATCH

ETHEx L4 Source Match

ETH_L4_SRC_PORT_PERFECT_MATCH_ENABLE

ETH_L4_SRC_PORT_INVERSE_MATCH_ENABLE

ETH_L4_SRC_PORT_MATCH_DISABLE

ETHEx LPI Event

ETH_TX_LPI_ENTRY

ETH_TX_LPI_EXIT

ETH_RX_LPI_ENTRY

ETH_RX_LPI_EXIT

ETHEx Preemption Packet

ETH_EXPRESS_PACKET

ETH_PREEMPTION_PACKET

ETHEx Rx Inner VLAN Tag Stripping

ETH_INNERVLANTAGRXSTRIPPING_NONE

ETH_INNERVLANTAGRXSTRIPPING_IFPASS

ETH_INNERVLANTAGRXSTRIPPING_IFFAILS

ETH_INNERVLANTAGRXSTRIPPING_ALWAYS

ETHEx Rx VLAN PRIO

ETH_RX_QUEUE_PRIO_0
Rx VLAN User Tag Priority 0

ETH_RX_QUEUE_PRIO_1
Rx VLAN User Tag Priority 1

ETH_RX_QUEUE_PRIO_2
Rx VLAN User Tag Priority 2

ETH_RX_QUEUE_PRIO_3
Rx VLAN User Tag Priority 3

ETH_RX_QUEUE_PRIO_4
Rx VLAN User Tag Priority 4

ETH_RX_QUEUE_PRIO_5
Rx VLAN User Tag Priority 5

ETH_RX_QUEUE_PRIO_6
Rx VLAN User Tag Priority 6

ETH_RX_QUEUE_PRIO_7
Rx VLAN User Tag Priority 7

ETHEx Rx VLAN Tag Stripping

ETH_VLANTAGRXSTRIPPING_NONE

ETH_VLANTAGRXSTRIPPING_IFPASS

ETH_VLANTAGRXSTRIPPING_IFFAILS

ETH_VLANTAGRXSTRIPPING_ALWAYS

ETHEx Tx VLAN Tag

ETH_INNER_TX_VLANTAG

ETH_OUTER_TX_VLANTAG

ETHEx_VLAN_Tag_Control

ETH_VLANTAGCONTROL_NONE

ETH_VLANTAGCONTROL_DELETE

ETH_VLANTAGCONTROL_INSERT

ETH_VLANTAGCONTROL_REPLACE

ETHEx VLAN Type Check

ETH_VLANTYPECHECK_DISABLE

ETH_VLANTYPECHECK_SVLAN

ETH_VLANTYPECHECK_CVLAN

24 HAL EXTI Generic Driver

24.1 EXTI Firmware driver registers structures

24.1.1 EXTI_HandleTypeDef

EXTI_HandleTypeDef is defined in the stm32n6xx_hal_exti.h

Data Fields

- ***uint32_t Line***
- ***void(* RisingCallback***
- ***void(* FallingCallback***

Field Documentation

- ***uint32_t EXTI_HandleTypeDef::Line***
  Exti line number
- ***void(* EXTI_HandleTypeDef::RisingCallback)(void)***
  Exti rising callback
- ***void(* EXTI_HandleTypeDef::FallingCallback)(void)***
  Exti falling callback

24.1.2 EXTI_ConfigTypeDef

EXTI_ConfigTypeDef is defined in the stm32n6xx_hal_exti.h

Data Fields

- ***uint32_t Line***
- ***uint32_t Mode***
- ***uint32_t Trigger***
- ***uint32_t GPIOSel***

Field Documentation

- ***uint32_t EXTI_ConfigTypeDef::Line***
  The Exti line to be configured. This parameter can be a value of ***EXTI_Line***
- ***uint32_t EXTI_ConfigTypeDef::Mode***
  The Exit Mode to be configured for a core. This parameter can be a combination of ***EXTI_Mode***
- ***uint32_t EXTI_ConfigTypeDef::Trigger***
  The Exti Trigger to be configured. This parameter can be a value of ***EXTI_Trigger***
- ***uint32_t EXTI_ConfigTypeDef::GPIOSel***
  The Exti GPIO multiplexer selection to be configured. This parameter is only possible for line 0 to 15. It can be a value of ***EXTI_GPIOSel***

24.2 EXTI Firmware driver API description

The following section lists the various functions of the EXTI library.

24.2.1 EXTI Peripheral features

- Each Exti line can be configured within this driver.
- Exti line can be configured in 3 different modes
  - Interrupt
  - Event
  - Both of them
- Configurable Exti lines can be configured with 3 different triggers
  - Rising
  - Falling
  - Both of them

- When set in interrupt mode, configurable Exti lines have two different interrupt pending registers which allow to distinguish which transition occurs:
  - Rising edge pending interrupt
  - Falling
- Exti lines 0 to 15 are linked to gpio pin number 0 to 15. Gpio port can be selected through multiplexer.

24.2.2 How to use this driver

1. Configure the EXTI line using HAL_EXTI_SetConfigLine().
   - Choose the interrupt line number by setting "Line" member from EXTI_ConfigTypeDef structure.
   - Configure the interrupt and/or event mode using "Mode" member from EXTI_ConfigTypeDef structure.
   - For configurable lines, configure rising and/or falling trigger "Trigger" member from EXTI_ConfigTypeDef structure.
   - For Exti lines linked to gpio, choose gpio port using "GPIOSel" member from GPIO_InitTypeDef structure.
2. Get current Exti configuration of a dedicated line using HAL_EXTI_GetConfigLine().
   - Provide exiting handle as parameter.
   - Provide pointer on EXTI_ConfigTypeDef structure as second parameter.
3. Clear Exti configuration of a dedicated line using HAL_EXTI_ClearConfigLine().
   - Provide exiting handle as parameter.
4. Register callback to treat Exti interrupts using HAL_EXTI_RegisterCallback().
   - Provide exiting handle as first parameter.
   - Provide which callback will be registered using one value from EXTI_CallbackIDTypeDef.
   - Provide callback function pointer.
5. Get interrupt pending bit using HAL_EXTI_GetPending().
6. Clear interrupt pending bit using HAL_EXTI_ClearPending().
7. Generate software interrupt using HAL_EXTI_GenerateSWI().

24.2.3 EXTI attributes functions

This section contains the following APIs:

- ***HAL_EXTI_ConfigLineAttributes()***
- ***HAL_EXTI_GetConfigLineAttributes()***
- ***HAL_EXTI_LockAttributes()***
- ***HAL_EXTI_GetLockAttributes()***

24.2.4 Configuration functions

This section contains the following APIs:

- ***HAL_EXTI_SetConfigLine()***
- ***HAL_EXTI_GetConfigLine()***
- ***HAL_EXTI_ClearConfigLine()***
- ***HAL_EXTI_RegisterCallback()***
- ***HAL_EXTI_GetHandle()***

24.2.5 Detailed description of functions

HAL_EXTI_SetConfigLine

Function name

HAL_StatusTypeDef HAL_EXTI_SetConfigLine (EXTI_HandleTypeDef * hexti, const EXTI_ConfigTypeDef * pExtiConfig)

Function description

Set configuration of a dedicated Exti line.

Parameters

- **hexti:** Exti handle.
- **pExtiConfig:** Pointer on EXTI configuration to be set.

Return values

- **HAL:** Status.

HAL_EXTI_GetConfigLine

Function name

HAL_StatusTypeDef HAL_EXTI_GetConfigLine (const EXTI_HandleTypeDef * hexti, EXTI_ConfigTypeDef * pExtiConfig)

Function description

Get configuration of a dedicated Exti line.

Parameters

- **hexti:** Exti handle.
- **pExtiConfig:** Pointer on structure to store Exti configuration.

Return values

- **HAL:** Status.

HAL_EXTI_ClearConfigLine

Function name

HAL_StatusTypeDef HAL_EXTI_ClearConfigLine (const EXTI_HandleTypeDef * hexti)

Function description

Clear whole configuration of a dedicated Exti line.

Parameters

- **hexti:** Exti handle.

Return values

- **HAL:** Status.

HAL_EXTI_RegisterCallback

Function name

HAL_StatusTypeDef HAL_EXTI_RegisterCallback (EXTI_HandleTypeDef * hexti, EXTI_CallbackIDTypeDef CallbackID, void(*)(void) pPendingCbfn)

Function description

Register callback for a dedicated Exti line.

Parameters

- **hexti:** Exti handle.
- **CallbackID:** User callback identifier. This parameter can be one of
  - EXTI_CallbackIDTypeDef values.
- **pPendingCbfn:** function pointer to be stored as callback.

Return values

- **HAL:** Status.

HAL_EXTI_GetHandle

Function name

HAL_StatusTypeDef HAL_EXTI_GetHandle (EXTI_HandleTypeDef * hexti, uint32_t ExtiLine)

Function description

Store line number as handle private field.

Parameters

- **hexti:** Exti handle.
- **ExtiLine:** Exti line number. This parameter can be from 0 to EXTI_LINE_NB.

Return values

- **HAL:** Status.

HAL_EXTI_IRQHandler

Function name

void HAL_EXTI_IRQHandler (const EXTI_HandleTypeDef * hexti)

Function description

Handle EXTI interrupt request.

Parameters

- **hexti:** Exti handle.

Return values

- **none.:**

HAL_EXTI_GetPending

Function name

uint32_t HAL_EXTI_GetPending (const EXTI_HandleTypeDef * hexti, uint32_t Edge)

Function description

Get interrupt pending bit of a dedicated line.

Parameters

- **hexti:** Exti handle.
- **Edge:** Specify which pending edge as to be checked. This parameter can be one of the following values:
  - EXTI_TRIGGER_RISING
  - EXTI_TRIGGER_FALLING

Return values

- **1:** if interrupt is pending else 0.

HAL_EXTI_ClearPending

Function name

void HAL_EXTI_ClearPending (const EXTI_HandleTypeDef * hexti, uint32_t Edge)

Function description

Clear interrupt pending bit of a dedicated line.

Parameters

- **hexti:** Exti handle.
- **Edge:** Specify which pending edge as to be clear. This parameter can be one of the following values:
  - EXTI_TRIGGER_RISING
  - EXTI_TRIGGER_FALLING

Return values

- **None.:**

HAL_EXTI_GenerateSWI

Function name

void HAL_EXTI_GenerateSWI (const EXTI_HandleTypeDef * hexti)

Function description

Generate a software interrupt for a dedicated line.

Parameters

- **hexti:** Exti handle.

Return values

- **None.:**

HAL_EXTI_ConfigLineAttributes

Function name

void HAL_EXTI_ConfigLineAttributes (uint32_t ExtiLine, uint32_t LineAttributes)

Function description

Configure the EXTI line attribute(s).

Parameters

- **ExtiLine:** Exti line number. This parameter can be from 0 to EXTI_LINE_NB.
- **LineAttributes:** can be one or a combination of the following values:
  - EXTI_LINE_PRIV Privileged-only access
  - EXTI_LINE_NPRIV Privileged/Non-privileged access
  - EXTI_LINE_SEC Secure-only access
  - EXTI_LINE_NSEC Secure/Non-secure access

Return values

- **None:**

Notes

- Available attributes are to secure EXTI line and set EXTI line as privileged. Default state is not secure and unprivileged access allowed.
- Set an EXTI line to secure is only available in secure and privilege Set an EXTI line to privilege is only available in privilege
- Security and privilege attributes can be set independently.

HAL_EXTI_GetConfigLineAttributes

Function name

HAL_StatusTypeDef HAL_EXTI_GetConfigLineAttributes (uint32_t ExtiLine, uint32_t * pLineAttributes)

Function description

Get the EXTI line attribute(s).

Parameters

- **ExtiLine:** Exti line number. This parameter can be from 0 to EXTI_LINE_NB.
- **pLineAttributes:** pointer to return line attributes.

Return values

- **HAL:** Status.

HAL_EXTI_LockAttributes

Function name

void HAL_EXTI_LockAttributes (void)

Function description

Lock the secure and privilege configuration registers.

Return values

- **None:**

HAL_EXTI_GetLockAttributes

Function name

uint32_t HAL_EXTI_GetLockAttributes (void)

Function description

Return the secure and privilege configuration registers LOCK status.

Return values

- **1:** if the secure and privilege configuration registers have been locked else 0.

24.3 EXTI Firmware driver defines

The following section lists the various define and macros of the module.

24.3.1 EXTI

EXTI

EXTI GPIOSel

EXTI_GPIOA

EXTI_GPIOB

EXTI_GPIOC

EXTI_GPIOD

EXTI_GPIOE

EXTI_GPIOF

EXTI_GPIOG

EXTI_GPIOH

EXTI_GPION

EXTI_GPIOO

EXTI_GPIOP

EXTI_GPIOQ

EXTI Line

EXTI_LINE_0

EXTI_LINE_1

EXTI_LINE_2

EXTI_LINE_3

EXTI_LINE_4

EXTI_LINE_5

EXTI_LINE_6

EXTI_LINE_7

EXTI_LINE_8

EXTI_LINE_9

EXTI_LINE_10

EXTI_LINE_11

EXTI_LINE_12

EXTI_LINE_13

EXTI_LINE_14

EXTI_LINE_15

EXTI_LINE_16

EXTI_LINE_17

EXTI_LINE_18

EXTI_LINE_19

EXTI_LINE_20

EXTI_LINE_21

EXTI_LINE_22

EXTI_LINE_23

EXTI_LINE_24

EXTI_LINE_25

EXTI_LINE_26

EXTI_LINE_27

EXTI_LINE_28

EXTI_LINE_29

EXTI_LINE_30

EXTI_LINE_31

EXTI_LINE_32

EXTI_LINE_33

EXTI_LINE_34

EXTI_LINE_35

EXTI_LINE_36

EXTI_LINE_37

EXTI_LINE_38

EXTI_LINE_39

EXTI_LINE_40

EXTI_LINE_41

EXTI_LINE_42

EXTI_LINE_43

EXTI_LINE_44

EXTI_LINE_45

EXTI_LINE_46

EXTI_LINE_47

EXTI_LINE_48

EXTI_LINE_49

EXTI_LINE_50

EXTI_LINE_51

EXTI_LINE_52

EXTI_LINE_53

EXTI_LINE_54

EXTI_LINE_55

EXTI_LINE_56

EXTI_LINE_57

EXTI_LINE_58

EXTI_LINE_59

EXTI_LINE_60

EXTI_LINE_61

EXTI_LINE_62

EXTI_LINE_63

EXTI_LINE_64

EXTI_LINE_65

EXTI_LINE_66

EXTI_LINE_67

EXTI_LINE_68

EXTI_LINE_69

EXTI_LINE_70

EXTI_LINE_71

EXTI_LINE_72

EXTI_LINE_73

EXTI_LINE_74

EXTI_LINE_75

EXTI_LINE_76

EXTI_LINE_77

EXTI line attributes

EXTI_LINE_SEC

Secure line attribute

EXTI_LINE_NSEC

Non-secure line attribute

EXTI_LINE_PRIV

Privileged line attribute

EXTI_LINE_NPRIV

Non-privileged line attribute

EXTI Mode

EXTI_MODE_NONE

EXTI_MODE_INTERRUPT

EXTI_MODE_EVENT

EXTI Trigger

EXTI_TRIGGER_NONE

EXTI_TRIGGER_RISING

EXTI_TRIGGER_FALLING

EXTI_TRIGGER_RISING_FALLING

25 HAL FDCAN Generic Driver

25.1 FDCAN Firmware driver registers structures

25.1.1 FDCAN_InitTypeDef

FDCAN_InitTypeDef is defined in the stm32n6xx_hal_fdcan.h

Data Fields

- ***uint32_t FrameFormat***
- ***uint32_t Mode***
- ***FunctionalState AutoRetransmission***
- ***FunctionalState TransmitPause***
- ***FunctionalState ProtocolException***
- ***uint32_t NominalPrescaler***
- ***uint32_t NominalSyncJumpWidth***
- ***uint32_t NominalTimeSeg1***
- ***uint32_t NominalTimeSeg2***
- ***uint32_t DataPrescaler***
- ***uint32_t DataSyncJumpWidth***
- ***uint32_t DataTimeSeg1***
- ***uint32_t DataTimeSeg2***
- ***uint32_t MessageRAMOffset***
- ***uint32_t StdFiltersNbr***
- ***uint32_t ExtFiltersNbr***
- ***uint32_t RxFifo0ElmtsNbr***
- ***uint32_t RxFifo0ElmtSize***
- ***uint32_t RxFifo1ElmtsNbr***
- ***uint32_t RxFifo1ElmtSize***
- ***uint32_t RxBuffersNbr***
- ***uint32_t RxBufferSize***
- ***uint32_t TxEventsNbr***
- ***uint32_t TxBuffersNbr***
- ***uint32_t TxFifoQueueElmtsNbr***
- ***uint32_t TxFifoQueueMode***
- ***uint32_t TxElmtSize***

Field Documentation

- ***uint32_t FDCAN_InitTypeDef::FrameFormat***
  Specifies the FDCAN frame format. This parameter can be a value of ***FDCAN_frame_format***
- ***uint32_t FDCAN_InitTypeDef::Mode***
  Specifies the FDCAN mode. This parameter can be a value of ***FDCAN_operating_mode***
- ***FunctionalState FDCAN_InitTypeDef::AutoRetransmission***
  Enable or disable the automatic retransmission mode. This parameter can be set to ENABLE or DISABLE
- ***FunctionalState FDCAN_InitTypeDef::TransmitPause***
  Enable or disable the Transmit Pause feature. This parameter can be set to ENABLE or DISABLE
- ***FunctionalState FDCAN_InitTypeDef::ProtocolException***
  Enable or disable the Protocol Exception Handling. This parameter can be set to ENABLE or DISABLE
- ***uint32_t FDCAN_InitTypeDef::NominalPrescaler***
  Specifies the value by which the oscillator frequency is divided for generating the nominal bit time quanta. This parameter must be a number between 1 and 512

- ***uint32_t FDCAN_InitTypeDef::NominalSyncJumpWidth***
  Specifies the maximum number of time quanta the FDCAN hardware is allowed to lengthen or shorten a bit to perform resynchronization. This parameter must be a number between 1 and 128
- ***uint32_t FDCAN_InitTypeDef::NominalTimeSeg1***
  Specifies the number of time quanta in Bit Segment 1. This parameter must be a number between 2 and 256
- ***uint32_t FDCAN_InitTypeDef::NominalTimeSeg2***
  Specifies the number of time quanta in Bit Segment 2. This parameter must be a number between 2 and 128
- ***uint32_t FDCAN_InitTypeDef::DataPrescaler***
  Specifies the value by which the oscillator frequency is divided for generating the data bit time quanta. This parameter must be a number between 1 and 32
- ***uint32_t FDCAN_InitTypeDef::DataSyncJumpWidth***
  Specifies the maximum number of time quanta the FDCAN hardware is allowed to lengthen or shorten a data bit to perform resynchronization. This parameter must be a number between 1 and 16
- ***uint32_t FDCAN_InitTypeDef::DataTimeSeg1***
  Specifies the number of time quanta in Data Bit Segment 1. This parameter must be a number between 1 and 32
- ***uint32_t FDCAN_InitTypeDef::DataTimeSeg2***
  Specifies the number of time quanta in Data Bit Segment 2. This parameter must be a number between 1 and 16
- ***uint32_t FDCAN_InitTypeDef::MessageRAMOffset***
  Specifies the message RAM start address. This parameter must be a number between 0 and 2560
- ***uint32_t FDCAN_InitTypeDef::StdFiltersNbr***
  Specifies the number of standard Message ID filters. This parameter must be a number between 0 and 128
- ***uint32_t FDCAN_InitTypeDef::ExtFiltersNbr***
  Specifies the number of extended Message ID filters. This parameter must be a number between 0 and 64
- ***uint32_t FDCAN_InitTypeDef::RxFifo0ElmtsNbr***
  Specifies the number of Rx FIFO0 Elements. This parameter must be a number between 0 and 64
- ***uint32_t FDCAN_InitTypeDef::RxFifo0ElmtSize***
  Specifies the Data Field Size in an Rx FIFO 0 element. This parameter can be a value of ***FDCAN_data_field_size***
- ***uint32_t FDCAN_InitTypeDef::RxFifo1ElmtsNbr***
  Specifies the number of Rx FIFO 1 Elements. This parameter must be a number between 0 and 64
- ***uint32_t FDCAN_InitTypeDef::RxFifo1ElmtSize***
  Specifies the Data Field Size in an Rx FIFO 1 element. This parameter can be a value of ***FDCAN_data_field_size***
- ***uint32_t FDCAN_InitTypeDef::RxBuffersNbr***
  Specifies the number of Dedicated Rx Buffer elements. This parameter must be a number between 0 and 64
- ***uint32_t FDCAN_InitTypeDef::RxBufferSize***
  Specifies the Data Field Size in an Rx Buffer element. This parameter can be a value of ***FDCAN_data_field_size***
- ***uint32_t FDCAN_InitTypeDef::TxEventsNbr***
  Specifies the number of Tx Event FIFO elements. This parameter must be a number between 0 and 32
- ***uint32_t FDCAN_InitTypeDef::TxBuffersNbr***
  Specifies the number of Dedicated Tx Buffers. This parameter must be a number between 0 and 32
- ***uint32_t FDCAN_InitTypeDef::TxFifoQueueElmtsNbr***
  Specifies the number of Tx Buffers used for Tx FIFO/Queue. This parameter must be a number between 0 and 32
- ***uint32_t FDCAN_InitTypeDef::TxFifoQueueMode***
  Tx FIFO/Queue Mode selection. This parameter can be a value of ***FDCAN_txFifoQueue_Mode***
- ***uint32_t FDCAN_InitTypeDef::TxElmtSize***
  Specifies the Data Field Size in a Tx Element. This parameter can be a value of ***FDCAN_data_field_size***

25.1.2 FDCAN_ClkCalUnitTypeDef

FDCAN_ClkCalUnitTypeDef is defined in the stm32n6xx_hal_fdcan.h

Data Fields

- ***uint32_t ClockCalibration***
- ***uint32_t ClockDivider***
- ***uint32_t MinOscClkPeriods***
- ***uint32_t CalFieldLength***
- ***uint32_t TimeQuantaPerBitTime***
- ***uint32_t WatchdogStartValue***

Field Documentation

- ***uint32_t FDCAN_ClkCalUnitTypeDef::ClockCalibration***
  Enable or disable the clock calibration. This parameter can be a value of ***FDCAN_clock_calibration***.
- ***uint32_t FDCAN_ClkCalUnitTypeDef::ClockDivider***
  Specifies the FDCAN kernel clock divider when the clock calibration is bypassed. This parameter can be a value of ***FDCAN_clock_divider***
- ***uint32_t FDCAN_ClkCalUnitTypeDef::MinOscClkPeriods***
  Configures the minimum number of periods in two CAN bit times. The actual configured number of periods is MinOscClkPeriods x 32. This parameter must be a number between 0x00 and 0xFF
- ***uint32_t FDCAN_ClkCalUnitTypeDef::CalFieldLength***
  Specifies the calibration field length. This parameter can be a value of ***FDCAN_calibration_field_length***
- ***uint32_t FDCAN_ClkCalUnitTypeDef::TimeQuantaPerBitTime***
  Configures the number of time quanta per bit time. This parameter must be a number between 4 and 25
- ***uint32_t FDCAN_ClkCalUnitTypeDef::WatchdogStartValue***
  Start value of the Calibration Watchdog Counter. If set to zero the counter is disabled. This parameter must be a number between 0x0000 and 0xFFFF

25.1.3 FDCAN_FilterTypeDef

FDCAN_FilterTypeDef is defined in the stm32n6xx_hal_fdcan.h

Data Fields

- ***uint32_t IdType***
- ***uint32_t FilterIndex***
- ***uint32_t FilterType***
- ***uint32_t FilterConfig***
- ***uint32_t FilterID1***
- ***uint32_t FilterID2***
- ***uint32_t RxBufferIndex***
- ***uint32_t IsCalibrationMsg***

Field Documentation

- ***uint32_t FDCAN_FilterTypeDef::IdType***
  Specifies the identifier type. This parameter can be a value of ***FDCAN_id_type***
- ***uint32_t FDCAN_FilterTypeDef::FilterIndex***
  Specifies the filter which will be initialized. This parameter must be a number between:
  - 0 and 127, if IdType is FDCAN_STANDARD_ID
  - 0 and 63, if IdType is FDCAN_EXTENDED_ID
- ***uint32_t FDCAN_FilterTypeDef::FilterType***
  Specifies the filter type. This parameter can be a value of ***FDCAN_filter_type***. The value FDCAN_EXT_FILTER_RANGE_NO_EIDM is permitted only when IdType is FDCAN_EXTENDED_ID. This parameter is ignored if FilterConfig is set to FDCAN_FILTER_TO_RXBUFFER
- ***uint32_t FDCAN_FilterTypeDef::FilterConfig***
  Specifies the filter configuration. This parameter can be a value of ***FDCAN_filter_config***
- ***uint32_t FDCAN_FilterTypeDef::FilterID1***
  Specifies the filter identification 1. This parameter must be a number between:
  - 0 and 0x7FF, if IdType is FDCAN_STANDARD_ID
  - 0 and 0x1FFFFFFF, if IdType is FDCAN_EXTENDED_ID

- ***uint32_t FDCAN_FilterTypeDef::FilterID2***
  Specifies the filter identification 2. This parameter is ignored if FilterConfig is set to FDCAN_FILTER_TO_RXBUFFER. This parameter must be a number between:
  - 0 and 0x7FF, if IdType is FDCAN_STANDARD_ID
  - 0 and 0x1FFFFFFF, if IdType is FDCAN_EXTENDED_ID
- ***uint32_t FDCAN_FilterTypeDef::RxBufferIndex***
  Contains the index of the Rx buffer in which the matching message will be stored. This parameter must be a number between 0 and 63. This parameter is ignored if FilterConfig is different from FDCAN_FILTER_TO_RXBUFFER
- ***uint32_t FDCAN_FilterTypeDef::IsCalibrationMsg***
  Specifies whether the filter is configured for calibration messages. This parameter is ignored if FilterConfig is different from FDCAN_FILTER_TO_RXBUFFER. This parameter can be:
  - 0 : ordinary message
  - 1 : calibration message

25.1.4 FDCAN_TxHeaderTypeDef

FDCAN_TxHeaderTypeDef is defined in the stm32n6xx_hal_fdcan.h

Data Fields

- ***uint32_t Identifier***
- ***uint32_t IdType***
- ***uint32_t TxFrameType***
- ***uint32_t DataLength***
- ***uint32_t ErrorStateIndicator***
- ***uint32_t BitRateSwitch***
- ***uint32_t FDFormat***
- ***uint32_t TxEventFifoControl***
- ***uint32_t MessageMarker***

Field Documentation

- ***uint32_t FDCAN_TxHeaderTypeDef::Identifier***
  Specifies the identifier. This parameter must be a number between:
  - 0 and 0x7FF, if IdType is FDCAN_STANDARD_ID
  - 0 and 0x1FFFFFFF, if IdType is FDCAN_EXTENDED_ID
- ***uint32_t FDCAN_TxHeaderTypeDef::IdType***
  Specifies the identifier type for the message that will be transmitted. This parameter can be a value of ***FDCAN_id_type***
- ***uint32_t FDCAN_TxHeaderTypeDef::TxFrameType***
  Specifies the frame type of the message that will be transmitted. This parameter can be a value of ***FDCAN_frame_type***
- ***uint32_t FDCAN_TxHeaderTypeDef::DataLength***
  Specifies the length of the frame that will be transmitted. This parameter can be a value of ***FDCAN_data_length_code***
- ***uint32_t FDCAN_TxHeaderTypeDef::ErrorStateIndicator***
  Specifies the error state indicator. This parameter can be a value of ***FDCAN_error_state_indicator***
- ***uint32_t FDCAN_TxHeaderTypeDef::BitRateSwitch***
  Specifies whether the Tx frame will be transmitted with or without bit rate switching. This parameter can be a value of ***FDCAN_bit_rate_switching***
- ***uint32_t FDCAN_TxHeaderTypeDef::FDFormat***
  Specifies whether the Tx frame will be transmitted in classic or FD format. This parameter can be a value of ***FDCAN_format***
- ***uint32_t FDCAN_TxHeaderTypeDef::TxEventFifoControl***
  Specifies the event FIFO control. This parameter can be a value of ***Section 25.3.1: FDCAN_EFC***
- ***uint32_t FDCAN_TxHeaderTypeDef::MessageMarker***
  Specifies the message marker to be copied into Tx Event FIFO element for identification of Tx message status. This parameter must be a number between 0 and 0xFF

25.1.5 FDCAN_RxHeaderTypeDef

FDCAN_RxHeaderTypeDef is defined in the stm32n6xx_hal_fdcan.h

Data Fields

- ***uint32_t Identifier***
- ***uint32_t IdType***
- ***uint32_t RxFrameType***
- ***uint32_t DataLength***
- ***uint32_t ErrorStateIndicator***
- ***uint32_t BitRateSwitch***
- ***uint32_t FDFormat***
- ***uint32_t RxTimestamp***
- ***uint32_t FilterIndex***
- ***uint32_t IsFilterMatchingFrame***

Field Documentation

- ***uint32_t FDCAN_RxHeaderTypeDef::Identifier***
  Specifies the identifier. This parameter must be a number between:
  – 0 and 0x7FF, if IdType is FDCAN_STANDARD_ID
  – 0 and 0x1FFFFFFF, if IdType is FDCAN_EXTENDED_ID
- ***uint32_t FDCAN_RxHeaderTypeDef::IdType***
  Specifies the identifier type of the received message. This parameter can be a value of ***FDCAN_id_type***
- ***uint32_t FDCAN_RxHeaderTypeDef::RxFrameType***
  Specifies the the received message frame type. This parameter can be a value of ***FDCAN_frame_type***
- ***uint32_t FDCAN_RxHeaderTypeDef::DataLength***
  Specifies the received frame length. This parameter can be a value of ***FDCAN_data_length_code***
- ***uint32_t FDCAN_RxHeaderTypeDef::ErrorStateIndicator***
  Specifies the error state indicator. This parameter can be a value of ***FDCAN_error_state_indicator***
- ***uint32_t FDCAN_RxHeaderTypeDef::BitRateSwitch***
  Specifies whether the Rx frame is received with or without bit rate switching. This parameter can be a value of ***FDCAN_bit_rate_switching***
- ***uint32_t FDCAN_RxHeaderTypeDef::FDFormat***
  Specifies whether the Rx frame is received in classic or FD format. This parameter can be a value of ***FDCAN_format***
- ***uint32_t FDCAN_RxHeaderTypeDef::RxTimestamp***
  Specifies the timestamp counter value captured on start of frame reception. This parameter must be a number between 0 and 0xFFFF
- ***uint32_t FDCAN_RxHeaderTypeDef::FilterIndex***
  Specifies the index of matching Rx acceptance filter element. This parameter must be a number between:
  – 0 and 127, if IdType is FDCAN_STANDARD_ID
  – 0 and 63, if IdType is FDCAN_EXTENDED_ID When the frame is a Non-Filter matching frame, this parameter is unused.
- ***uint32_t FDCAN_RxHeaderTypeDef::IsFilterMatchingFrame***
  Specifies whether the accepted frame did not match any Rx filter. Acceptance of non-matching frames may be enabled via **HAL_FDCAN_ConfigGlobalFilter()**. This parameter takes 0 if the frame matched an Rx filter or 1 if it did not match any Rx filter

25.1.6 FDCAN_TxEventFifoTypeDef

FDCAN_TxEventFifoTypeDef is defined in the stm32n6xx_hal_fdcan.h

Data Fields

- ***uint32_t Identifier***
- ***uint32_t IdType***
- ***uint32_t TxFrameType***
- ***uint32_t DataLength***
- ***uint32_t ErrorStateIndicator***

- ***uint32_t BitRateSwitch***
- ***uint32_t FDFormat***
- ***uint32_t TxTimestamp***
- ***uint32_t MessageMarker***
- ***uint32_t EventType***

Field Documentation

- ***uint32_t FDCAN_TxEventFifoTypeDef::Identifier***
  Specifies the identifier. This parameter must be a number between:
  - 0 and 0x7FF, if IdType is FDCAN_STANDARD_ID
  - 0 and 0x1FFFFFFF, if IdType is FDCAN_EXTENDED_ID
- ***uint32_t FDCAN_TxEventFifoTypeDef::IdType***
  Specifies the identifier type for the transmitted message. This parameter can be a value of ***FDCAN_id_type***
- ***uint32_t FDCAN_TxEventFifoTypeDef::TxFrameType***
  Specifies the frame type of the transmitted message. This parameter can be a value of ***FDCAN_frame_type***
- ***uint32_t FDCAN_TxEventFifoTypeDef::DataLength***
  Specifies the length of the transmitted frame. This parameter can be a value of ***FDCAN_data_length_code***
- ***uint32_t FDCAN_TxEventFifoTypeDef::ErrorStateIndicator***
  Specifies the error state indicator. This parameter can be a value of ***FDCAN_error_state_indicator***
- ***uint32_t FDCAN_TxEventFifoTypeDef::BitRateSwitch***
  Specifies whether the Tx frame is transmitted with or without bit rate switching. This parameter can be a value of ***FDCAN_bit_rate_switching***
- ***uint32_t FDCAN_TxEventFifoTypeDef::FDFormat***
  Specifies whether the Tx frame is transmitted in classic or FD format. This parameter can be a value of ***FDCAN_format***
- ***uint32_t FDCAN_TxEventFifoTypeDef::TxTimestamp***
  Specifies the timestamp counter value captured on start of frame transmission. This parameter must be a number between 0 and 0xFFFF
- ***uint32_t FDCAN_TxEventFifoTypeDef::MessageMarker***
  Specifies the message marker copied into Tx Event FIFO element for identification of Tx message status. This parameter must be a number between 0 and 0xFF
- ***uint32_t FDCAN_TxEventFifoTypeDef::EventType***
  Specifies the event type. This parameter can be a value of ***FDCAN_event_type***

25.1.7 FDCAN_HpMsgStatusTypeDef

FDCAN_HpMsgStatusTypeDef is defined in the stm32n6xx_hal_fdcan.h

Data Fields

- ***uint32_t FilterList***
- ***uint32_t FilterIndex***
- ***uint32_t MessageStorage***
- ***uint32_t MessageIndex***

Field Documentation

- ***uint32_t FDCAN_HpMsgStatusTypeDef::FilterList***
  Specifies the filter list of the matching filter element. This parameter can be:
  - 0 : Standard Filter List
  - 1 : Extended Filter List
- ***uint32_t FDCAN_HpMsgStatusTypeDef::FilterIndex***
  Specifies the index of matching filter element. This parameter can be a number between:
  - 0 and 127, if FilterList is 0 (Standard)
  - 0 and 63, if FilterList is 1 (Extended)
- ***uint32_t FDCAN_HpMsgStatusTypeDef::MessageStorage***
  Specifies the HP Message Storage. This parameter can be a value of ***FDCAN_hp_msg_storage***

- ***uint32_t FDCAN_HpMsgStatusTypeDef::MessageIndex***
  Specifies the Index of Rx FIFO element to which the message was stored. This parameter is valid only when MessageStorage is: FDCAN_HP_STORAGE_RXFIFO0 or FDCAN_HP_STORAGE_RXFIFO1

25.1.8 FDCAN_ProtocolStatusTypeDef

FDCAN_ProtocolStatusTypeDef is defined in the stm32n6xx_hal_fdcan.h

Data Fields

- ***uint32_t LastErrorCode***
- ***uint32_t DataLastErrorCode***
- ***uint32_t Activity***
- ***uint32_t ErrorPassive***
- ***uint32_t Warning***
- ***uint32_t BusOff***
- ***uint32_t RxESIflag***
- ***uint32_t RxBRSflag***
- ***uint32_t RxFDFflag***
- ***uint32_t ProtocolException***
- ***uint32_t TDCvalue***

Field Documentation

- ***uint32_t FDCAN_ProtocolStatusTypeDef::LastErrorCode***
  Specifies the type of the last error that occurred on the FDCAN bus. This parameter can be a value of ***FDCAN_protocol_error_code***
- ***uint32_t FDCAN_ProtocolStatusTypeDef::DataLastErrorCode***
  Specifies the type of the last error that occurred in the data phase of a CAN FD format frame with its BRS flag set. This parameter can be a value of ***FDCAN_protocol_error_code***
- ***uint32_t FDCAN_ProtocolStatusTypeDef::Activity***
  Specifies the FDCAN module communication state. This parameter can be a value of ***FDCAN_communication_state***
- ***uint32_t FDCAN_ProtocolStatusTypeDef::ErrorPassive***
  Specifies the FDCAN module error status. This parameter can be:
  - 0 : The FDCAN is in Error_Active state
  - 1 : The FDCAN is in Error_Passive state
- ***uint32_t FDCAN_ProtocolStatusTypeDef::Warning***
  Specifies the FDCAN module warning status. This parameter can be:
  - 0 : error counters (RxErrorCnt and TxErrorCnt) are below the Error_Warning limit of 96
  - 1 : at least one of error counters has reached the Error_Warning limit of 96
- ***uint32_t FDCAN_ProtocolStatusTypeDef::BusOff***
  Specifies the FDCAN module Bus_Off status. This parameter can be:
  - 0 : The FDCAN is not in Bus_Off state
  - 1 : The FDCAN is in Bus_Off state
- ***uint32_t FDCAN_ProtocolStatusTypeDef::RxESIflag***
  Specifies ESI flag of last received CAN FD message. This parameter can be:
  - 0 : Last received CAN FD message did not have its ESI flag set
  - 1 : Last received CAN FD message had its ESI flag set
- ***uint32_t FDCAN_ProtocolStatusTypeDef::RxBRSflag***
  Specifies BRS flag of last received CAN FD message. This parameter can be:
  - 0 : Last received CAN FD message did not have its BRS flag set
  - 1 : Last received CAN FD message had its BRS flag set
- ***uint32_t FDCAN_ProtocolStatusTypeDef::RxFDFflag***
  Specifies if CAN FD message (FDF flag set) has been received since last protocol status. This parameter can be:
  - 0 : No CAN FD message received
  - 1 : CAN FD message received

- ***uint32_t FDCAN_ProtocolStatusTypeDef::ProtocolException***
  Specifies the FDCAN module Protocol Exception status. This parameter can be:
  - 0 : No protocol exception event occurred since last read access
  - 1 : Protocol exception event occurred
- ***uint32_t FDCAN_ProtocolStatusTypeDef::TDCvalue***
  Specifies the Transmitter Delay Compensation Value. This parameter can be a number between 0 and 127

25.1.9 FDCAN_ErrorCountersTypeDef

FDCAN_ErrorCountersTypeDef is defined in the stm32n6xx_hal_fdcan.h

Data Fields

- ***uint32_t TxErrorCnt***
- ***uint32_t RxErrorCnt***
- ***uint32_t RxErrorPassive***
- ***uint32_t ErrorLogging***

Field Documentation

- ***uint32_t FDCAN_ErrorCountersTypeDef::TxErrorCnt***
  Specifies the Transmit Error Counter Value. This parameter can be a number between 0 and 255
- ***uint32_t FDCAN_ErrorCountersTypeDef::RxErrorCnt***
  Specifies the Receive Error Counter Value. This parameter can be a number between 0 and 127
- ***uint32_t FDCAN_ErrorCountersTypeDef::RxErrorPassive***
  Specifies the Receive Error Passive status. This parameter can be:
  - 0 : The Receive Error Counter (RxErrorCnt) is below the error passive level of 128
  - 1 : The Receive Error Counter (RxErrorCnt) has reached the error passive level of 128
- ***uint32_t FDCAN_ErrorCountersTypeDef::ErrorLogging***
  Specifies the Transmit/Receive error logging counter value. This parameter can be a number between 0 and 255. This counter is incremented each time when a FDCAN protocol error causes the TxErrorCnt or the RxErrorCnt to be incremented. The counter stops at 255; the next increment of TxErrorCnt or RxErrorCnt sets interrupt flag FDCAN_FLAG_ERROR_LOGGING_OVERFLOW

25.1.10 FDCAN_TT_ConfigTypeDef

FDCAN_TT_ConfigTypeDef is defined in the stm32n6xx_hal_fdcan.h

Data Fields

- ***uint32_t OperationMode***
- ***uint32_t GapEnable***
- ***uint32_t TimeMaster***
- ***uint32_t SyncDevLimit***
- ***uint32_t InitRefTrigOffset***
- ***uint32_t ExternalClkSync***
- ***uint32_t AppWdgLimit***
- ***uint32_t GlobalTimeFilter***
- ***uint32_t ClockCalibration***
- ***uint32_t EvtTrigPolarity***
- ***uint32_t BasicCyclesNbr***
- ***uint32_t CycleStartSync***
- ***uint32_t TxEnableWindow***
- ***uint32_t ExpTxTrigNbr***
- ***uint32_t TURNumerator***
- ***uint32_t TURDenominator***
- ***uint32_t TriggerMemoryNbr***
- ***uint32_t StopWatchTrigSel***
- ***uint32_t EventTrigSel***

Field Documentation

- ***uint32_t FDCAN_TT_ConfigTypeDef::OperationMode***
  Specifies the FDCAN Operation Mode. This parameter can be a value of ***FDCAN_operation_mode***
- ***uint32_t FDCAN_TT_ConfigTypeDef::GapEnable***
  Specifies the FDCAN TT Operation. This parameter can be a value of ***FDCAN_TT_operation***. This parameter is ignored if OperationMode is set to FDCAN_TT_COMMUNICATION_LEVEL0
- ***uint32_t FDCAN_TT_ConfigTypeDef::TimeMaster***
  Specifies whether the instance is a slave or a potential master. This parameter can be a value of ***FDCAN_TT_time_master***
- ***uint32_t FDCAN_TT_ConfigTypeDef::SyncDevLimit***
  Specifies the Synchronization Deviation Limit SDL of the TUR numerator : TUR = (Numerator +/- SDL) / Denominator. With : SDL = 2^(SyncDevLimit+5). This parameter must be a number between 0 and 7
- ***uint32_t FDCAN_TT_ConfigTypeDef::InitRefTrigOffset***
  Specifies the Initial Reference Trigger Offset. This parameter must be a number between 0 and 127
- ***uint32_t FDCAN_TT_ConfigTypeDef::ExternalClkSync***
  Enable or disable External Clock Synchronization. This parameter can be a value of ***FDCAN_TT_external_clk_sync***. This parameter is ignored if OperationMode is set to FDCAN_TT_COMMUNICATION_LEVEL1
- ***uint32_t FDCAN_TT_ConfigTypeDef::AppWdgLimit***
  Specifies the Application Watchdog Limit : maximum time after which the application has to serve the application watchdog. The application watchdog is incremented once each 256 NTUs. The application watchdog can be disabled by setting AppWdgLimit to 0. This parameter must be a number between 0 and 255. This parameter is ignored if OperationMode is set to FDCAN_TT_COMMUNICATION_LEVEL0
- ***uint32_t FDCAN_TT_ConfigTypeDef::GlobalTimeFilter***
  Enable or disable Global Time Filtering. This parameter can be a value of ***FDCAN_TT_global_time_filtering***. This parameter is ignored if OperationMode is set to FDCAN_TT_COMMUNICATION_LEVEL1
- ***uint32_t FDCAN_TT_ConfigTypeDef::ClockCalibration***
  Enable or disable Automatic Clock Calibration. This parameter can be a value of ***FDCAN_TT_auto_clk_calibration***. This parameter is ignored if OperationMode is set to FDCAN_TT_COMMUNICATION_LEVEL1
- ***uint32_t FDCAN_TT_ConfigTypeDef::EvtTrigPolarity***
  Specifies the Event Trigger Polarity. This parameter can be a value of ***FDCAN_TT_event_trig_polarity***. This parameter is ignored if OperationMode is set to FDCAN_TT_COMMUNICATION_LEVEL0
- ***uint32_t FDCAN_TT_ConfigTypeDef::BasicCyclesNbr***
  Specifies the number of basic cycles in the system matrix. This parameter can be a value of ***FDCAN_TT_basic_cycle_number***
- ***uint32_t FDCAN_TT_ConfigTypeDef::CycleStartSync***
  Enable or disable synchronization pulse output at pin fdcan1_soc. This parameter can be a value of ***FDCAN_TT_cycle_start_sync***
- ***uint32_t FDCAN_TT_ConfigTypeDef::TxEnableWindow***
  Specifies the length of Tx enable window in NTUs. This parameter must be a number between 1 and 16
- ***uint32_t FDCAN_TT_ConfigTypeDef::ExpTxTrigNbr***
  Specifies the number of expected Tx_Triggers in the system matrix. This is the sum of Tx_Triggers for exclusive, single arbitrating and merged arbitrating windows. This parameter must be a number between 0 and 4095
- ***uint32_t FDCAN_TT_ConfigTypeDef::TURNumerator***
  Specifies the TUR (Time Unit Ratio) numerator. It is advised to set this parameter to the largest applicable value. This parameter must be a number between 0x10000 and 0x1FFFF
- ***uint32_t FDCAN_TT_ConfigTypeDef::TURDenominator***
  Specifies the TUR (Time Unit Ratio) denominator. This parameter must be a number between 0x0001 and 0x3FFF
- ***uint32_t FDCAN_TT_ConfigTypeDef::TriggerMemoryNbr***
  Specifies the number of trigger memory elements. This parameter must be a number between 0 and 64
- ***uint32_t FDCAN_TT_ConfigTypeDef::StopWatchTrigSel***
  Specifies the input to be used as stop watch trigger. This parameter can be a value of ***FDCAN_TT_stop_watch_trig_selection***

- ***uint32_t FDCAN_TT_ConfigTypeDef::EventTrigSel***
  Specifies the input to be used as event trigger. This parameter can be a value of ***FDCAN_TT_event_trig_selection***

25.1.11 FDCAN_TriggerTypeDef

FDCAN_TriggerTypeDef is defined in the stm32n6xx_hal_fdcan.h

Data Fields

- ***uint32_t TriggerIndex***
- ***uint32_t TimeMark***
- ***uint32_t RepeatFactor***
- ***uint32_t StartCycle***
- ***uint32_t TmEventInt***
- ***uint32_t TmEventExt***
- ***uint32_t TriggerType***
- ***uint32_t FilterType***
- ***uint32_t TxBufferIndex***
- ***uint32_t FilterIndex***

Field Documentation

- ***uint32_t FDCAN_TriggerTypeDef::TriggerIndex***
  Specifies the trigger which will be configured. This parameter must be a number between 0 and 63
- ***uint32_t FDCAN_TriggerTypeDef::TimeMark***
  Specifies the cycle time for which the trigger becomes active. This parameter must be a number between 0 and 0xFFFF
- ***uint32_t FDCAN_TriggerTypeDef::RepeatFactor***
  Specifies the trigger repeat factor. This parameter can be a value of ***FDCAN_TT_Repeat_Factor***
- ***uint32_t FDCAN_TriggerTypeDef::StartCycle***
  Specifies the index of the first cycle in which the trigger becomes active. This parameter is ignored if RepeatFactor is set to FDCAN_TT_REPEAT_EVERY_CYCLE. This parameter must be a number between 0 and RepeatFactor
- ***uint32_t FDCAN_TriggerTypeDef::TmEventInt***
  Enable or disable the internal time mark event. If enabled, FDCAN_TT_FLAG_TRIG_TIME_MARK flag is set when trigger memory element becomes active. This parameter can be a value of ***FDCAN_TT_Time_Mark_Event_Internal***
- ***uint32_t FDCAN_TriggerTypeDef::TmEventExt***
  Enable or disable the external time mark event. If enabled, and if TTOCN.TTIE is set, a pulse is generated at fdcan1_tmp when trigger memory element becomes active. This parameter can be a value of ***FDCAN_TT_Time_Mark_Event_External***
- ***uint32_t FDCAN_TriggerTypeDef::TriggerType***
  Specifies the trigger type. This parameter can be a value of ***FDCAN_TT_Trigger_Type***
- ***uint32_t FDCAN_TriggerTypeDef::FilterType***
  Specifies the filter identifier type. This parameter can be a value of ***FDCAN_id_type***
- ***uint32_t FDCAN_TriggerTypeDef::TxBufferIndex***
  Specifies the index of the Tx buffer for which the trigger is valid. This parameter can be a value of ***FDCAN_Tx_location***. This parameter is taken in consideration only if the trigger is configured for transmission.
- ***uint32_t FDCAN_TriggerTypeDef::FilterIndex***
  Specifies the filter for which the trigger is valid. This parameter is taken in consideration only if the trigger is configured for reception. This parameter must be a number between:
  - 0 and 127, if FilterType is FDCAN_STANDARD_ID
  - 0 and 63, if FilterType is FDCAN_EXTENDED_ID

25.1.12 FDCAN_TTOperationStatusTypeDef

FDCAN_TTOperationStatusTypeDef is defined in the stm32n6xx_hal_fdcan.h

Data Fields

- ***uint32_t ErrorLevel***
- ***uint32_t MasterState***
- ***uint32_t SyncState***
- ***uint32_t GTimeQuality***
- ***uint32_t ClockQuality***
- ***uint32_t RefTrigOffset***
- ***uint32_t GTimeDiscPending***
- ***uint32_t GapFinished***
- ***uint32_t MasterPriority***
- ***uint32_t GapStarted***
- ***uint32_t WaitForEvt***
- ***uint32_t AppWdgEvt***
- ***uint32_t ECSPending***
- ***uint32_t PhaseLock***

Field Documentation

- ***uint32_t FDCAN_TTOperationStatusTypeDef::ErrorLevel***
  Specifies the type of the TT operation error level. This parameter can be a value of ***FDCAN_TT_error_level***
- ***uint32_t FDCAN_TTOperationStatusTypeDef::MasterState***
  Specifies the type of the TT master state. This parameter can be a value of ***FDCAN_TT_master_state***
- ***uint32_t FDCAN_TTOperationStatusTypeDef::SyncState***
  Specifies the type of the TT synchronization state. This parameter can be a value of ***FDCAN_TT_sync_state***
- ***uint32_t FDCAN_TTOperationStatusTypeDef::GTimeQuality***
  Specifies the Quality of Global Time Phase. This parameter is only relevant in Level 0 and Level 2, otherwise fixed to 0. This parameter can be:
  – 0 : Global time not valid
  – 1 : Global time in phase with Time Master
- ***uint32_t FDCAN_TTOperationStatusTypeDef::ClockQuality***
  Specifies the Quality of Clock Speed. This parameter is only relevant in Level 0 and Level 2, otherwise fixed to 1. This parameter can be:
  – 0 : Local clock speed not synchronized to Time Master clock speed
  – 1 : Synchronization Deviation = SDL
- ***uint32_t FDCAN_TTOperationStatusTypeDef::RefTrigOffset***
  Specifies the Actual Reference Trigger Offset Value. This parameter can be a number between 0 and 0xFF
- ***uint32_t FDCAN_TTOperationStatusTypeDef::GTimeDiscPending***
  Specifies the Global Time Discontinuity State. This parameter can be:
  – 0 : No global time preset pending
  – 1 : Node waits for the global time preset to take effect
- ***uint32_t FDCAN_TTOperationStatusTypeDef::GapFinished***
  Specifies whether a Gap is finished. This parameter can be:
  – 0 : Reset at the end of each reference message
  – 1 : Gap finished
- ***uint32_t FDCAN_TTOperationStatusTypeDef::MasterPriority***
  Specifies the Priority of actual Time Master. This parameter can be a number between 0 and 0x7
- ***uint32_t FDCAN_TTOperationStatusTypeDef::GapStarted***
  Specifies whether a Gap is started. This parameter can be:
  – 0 : No Gap in schedule
  – 1 : Gap time after Basic Cycle has started
- ***uint32_t FDCAN_TTOperationStatusTypeDef::WaitForEvt***
  Specifies whether a Gap is announced. This parameter can be:
  – 0 : No Gap announced, reset by a reference message with Next_is_Gap = 0
  – 1 : Reference message with Next_is_Gap = 1 received

- ***uint32_t FDCAN_TTOperationStatusTypeDef::AppWdgEvt***
  Specifies the Application Watchdog State. This parameter can be:
  - 0 : Application Watchdog served in time
  - 1 : Failed to serve Application Watchdog in time
- ***uint32_t FDCAN_TTOperationStatusTypeDef::ECSPending***
  Specifies the External Clock Synchronization State. This parameter can be:
  - 0 : No external clock synchronization pending
  - 1 : Node waits for external clock synchronization to take effect
- ***uint32_t FDCAN_TTOperationStatusTypeDef::PhaseLock***
  Specifies the Phase Lock State. This parameter can be:
  - 0 : Phase outside range
  - 1 : Phase inside range

25.1.13 FDCAN_MsgRamAddressTypeDef

FDCAN_MsgRamAddressTypeDef is defined in the stm32n6xx_hal_fdcan.h

Data Fields

- ***uint32_t StandardFilterSA***
- ***uint32_t ExtendedFilterSA***
- ***uint32_t RxFIFO0SA***
- ***uint32_t RxFIFO1SA***
- ***uint32_t RxBufferSA***
- ***uint32_t TxEventFIFOSA***
- ***uint32_t TxBufferSA***
- ***uint32_t TxFIFOQSA***
- ***uint32_t TTMemorySA***
- ***uint32_t EndAddress***

Field Documentation

- ***uint32_t FDCAN_MsgRamAddressTypeDef::StandardFilterSA***
  Specifies the Standard Filter List Start Address. This parameter must be a 32-bit word address
- ***uint32_t FDCAN_MsgRamAddressTypeDef::ExtendedFilterSA***
  Specifies the Extended Filter List Start Address. This parameter must be a 32-bit word address
- ***uint32_t FDCAN_MsgRamAddressTypeDef::RxFIFO0SA***
  Specifies the Rx FIFO 0 Start Address. This parameter must be a 32-bit word address
- ***uint32_t FDCAN_MsgRamAddressTypeDef::RxFIFO1SA***
  Specifies the Rx FIFO 1 Start Address. This parameter must be a 32-bit word address
- ***uint32_t FDCAN_MsgRamAddressTypeDef::RxBufferSA***
  Specifies the Rx Buffer Start Address. This parameter must be a 32-bit word address
- ***uint32_t FDCAN_MsgRamAddressTypeDef::TxEventFIFOSA***
  Specifies the Tx Event FIFO Start Address. This parameter must be a 32-bit word address
- ***uint32_t FDCAN_MsgRamAddressTypeDef::TxBufferSA***
  Specifies the Tx Buffers Start Address. This parameter must be a 32-bit word address
- ***uint32_t FDCAN_MsgRamAddressTypeDef::TxFIFOQSA***
  Specifies the Tx FIFO/Queue Start Address. This parameter must be a 32-bit word address
- ***uint32_t FDCAN_MsgRamAddressTypeDef::TTMemorySA***
  Specifies the Trigger Memory Start Address. This parameter must be a 32-bit word address
- ***uint32_t FDCAN_MsgRamAddressTypeDef::EndAddress***
  Specifies the End Address of the allocated RAM. This parameter must be a 32-bit word address

25.1.14 __FDCAN_HandleTypeDef

__FDCAN_HandleTypeDef is defined in the stm32n6xx_hal_fdcan.h

Data Fields

- ***FDCAN_GlobalTypeDef * Instance***

- ***TTCAN_TypeDef * ttcan***
- ***FDCAN_InitTypeDef Init***
- ***FDCAN_MsgRamAddressTypeDef msgRam***
- ***uint32_t LatestTxFifoQRequest***
- ***__IO HAL_FDCAN_StateTypeDef State***
- ***HAL_LockTypeDef Lock***
- ***__IO uint32_t ErrorCode***
- ***void(* ClockCalibrationCallback***
- ***void(* TxEventFifoCallback***
- ***void(* RxFifo0Callback***
- ***void(* RxFifo1Callback***
- ***void(* TxFifoEmptyCallback***
- ***void(* TxBufferCompleteCallback***
- ***void(* TxBufferAbortCallback***
- ***void(* RxBufferNewMessageCallback***
- ***void(* HighPriorityMessageCallback***
- ***void(* TimestampWraparoundCallback***
- ***void(* TimeoutOccurredCallback***
- ***void(* ErrorCallback***
- ***void(* ErrorStatusCallback***
- ***void(* TT_ScheduleSyncCallback***
- ***void(* TT_TimeMarkCallback***
- ***void(* TT_StopWatchCallback***
- ***void(* TT_GlobalTimeCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***FDCAN_GlobalTypeDef* __FDCAN_HandleTypeDef::Instance***
  Register base address
- ***TTCAN_TypeDef* __FDCAN_HandleTypeDef::ttcan***
  TT register base address
- ***FDCAN_InitTypeDef __FDCAN_HandleTypeDef::Init***
  FDCAN required parameters
- ***FDCAN_MsgRamAddressTypeDef __FDCAN_HandleTypeDef::msgRam***
  FDCAN Message RAM blocks
- ***uint32_t __FDCAN_HandleTypeDef::LatestTxFifoQRequest***
  FDCAN Tx buffer index of latest Tx FIFO/Queue request
- ***__IO HAL_FDCAN_StateTypeDef __FDCAN_HandleTypeDef::State***
  FDCAN communication state
- ***HAL_LockTypeDef __FDCAN_HandleTypeDef::Lock***
  FDCAN locking object
- ***__IO uint32_t __FDCAN_HandleTypeDef::ErrorCode***
  FDCAN Error code
- ***void(* __FDCAN_HandleTypeDef::ClockCalibrationCallback)(struct __FDCAN_HandleTypeDef *hfdcan, uint32_t ClkCalibrationITs)***
  FDCAN Clock Calibration callback
- ***void(* __FDCAN_HandleTypeDef::TxEventFifoCallback)(struct __FDCAN_HandleTypeDef *hfdcan, uint32_t TxEventFifoITs)***
  FDCAN Tx Event Fifo callback
- ***void(* __FDCAN_HandleTypeDef::RxFifo0Callback)(struct __FDCAN_HandleTypeDef *hfdcan, uint32_t RxFifo0ITs)***
  FDCAN Rx Fifo 0 callback

- ***void(* __FDCAN_HandleTypeDef::RxFifo1Callback)(struct __FDCAN_HandleTypeDef *hfdcan, uint32_t RxFifo1ITs)***
  FDCAN Rx Fifo 1 callback
- ***void(* __FDCAN_HandleTypeDef::TxFifoEmptyCallback)(struct __FDCAN_HandleTypeDef *hfdcan)***
  FDCAN Tx Fifo Empty callback
- ***void(* __FDCAN_HandleTypeDef::TxBufferCompleteCallback)(struct __FDCAN_HandleTypeDef *hfdcan, uint32_t BufferIndexes)***
  FDCAN Tx Buffer complete callback
- ***void(* __FDCAN_HandleTypeDef::TxBufferAbortCallback)(struct __FDCAN_HandleTypeDef *hfdcan, uint32_t BufferIndexes)***
  FDCAN Tx Buffer abort callback
- ***void(* __FDCAN_HandleTypeDef::RxBufferNewMessageCallback)(struct __FDCAN_HandleTypeDef *hfdcan)***
  FDCAN Rx Buffer New Message callback
- ***void(* __FDCAN_HandleTypeDef::HighPriorityMessageCallback)(struct __FDCAN_HandleTypeDef *hfdcan)***
  FDCAN High priority message callback
- ***void(* __FDCAN_HandleTypeDef::TimestampWraparoundCallback)(struct __FDCAN_HandleTypeDef *hfdcan)***
  FDCAN Timestamp wraparound callback
- ***void(* __FDCAN_HandleTypeDef::TimeoutOccurredCallback)(struct __FDCAN_HandleTypeDef *hfdcan)***
  FDCAN Timeout occurred callback
- ***void(* __FDCAN_HandleTypeDef::ErrorCallback)(struct __FDCAN_HandleTypeDef *hfdcan)***
  FDCAN Error callback
- ***void(* __FDCAN_HandleTypeDef::ErrorStatusCallback)(struct __FDCAN_HandleTypeDef *hfdcan, uint32_t ErrorStatusITs)***
  FDCAN Error status callback
- ***void(* __FDCAN_HandleTypeDef::TT_ScheduleSyncCallback)(struct __FDCAN_HandleTypeDef *hfdcan, uint32_t TTSchedSyncITs)***
  FDCAN T Schedule Synchronization callback
- ***void(* __FDCAN_HandleTypeDef::TT_TimeMarkCallback)(struct __FDCAN_HandleTypeDef *hfdcan, uint32_t TTTimeMarkITs)***
  FDCAN TT Time Mark callback
- ***void(* __FDCAN_HandleTypeDef::TT_StopWatchCallback)(struct __FDCAN_HandleTypeDef *hfdcan, uint32_t SWTime, uint32_t SWCycleCount)***
  FDCAN TT Stop Watch callback
- ***void(* __FDCAN_HandleTypeDef::TT_GlobalTimeCallback)(struct __FDCAN_HandleTypeDef *hfdcan, uint32_t TTGlobTimeITs)***
  FDCAN TT Global Time callback
- ***void(* __FDCAN_HandleTypeDef::MspInitCallback)(struct __FDCAN_HandleTypeDef *hfdcan)***
  FDCAN Msp Init callback
- ***void(* __FDCAN_HandleTypeDef::MspDeInitCallback)(struct __FDCAN_HandleTypeDef *hfdcan)***
  FDCAN Msp DeInit callback

25.2 FDCAN Firmware driver API description

The following section lists the various functions of the FDCAN library.

25.2.1 How to use this driver

1. Initialize the FDCAN peripheral using HAL_FDCAN_Init function.

2. If needed , configure the reception filters and optional features using the following configuration functions:
   - HAL_FDCAN_ConfigClockCalibration
   - HAL_FDCAN_ConfigFilter
   - HAL_FDCAN_ConfigGlobalFilter
   - HAL_FDCAN_ConfigExtendedIdMask
   - HAL_FDCAN_ConfigRxFifoOverwrite
   - HAL_FDCAN_ConfigFifoWatermark
   - HAL_FDCAN_ConfigRamWatchdog
   - HAL_FDCAN_ConfigTimestampCounter
   - HAL_FDCAN_EnableTimestampCounter
   - HAL_FDCAN_DisableTimestampCounter
   - HAL_FDCAN_ConfigTimeoutCounter
   - HAL_FDCAN_EnableTimeoutCounter
   - HAL_FDCAN_DisableTimeoutCounter
   - HAL_FDCAN_ConfigTxDelayCompensation
   - HAL_FDCAN_EnableTxDelayCompensation
   - HAL_FDCAN_DisableTxDelayCompensation
   - HAL_FDCAN_EnableISOMode
   - HAL_FDCAN_DisableISOMode
   - HAL_FDCAN_EnableEdgeFiltering
   - HAL_FDCAN_DisableEdgeFiltering
   - HAL_FDCAN_TT_ConfigOperation
   - HAL_FDCAN_TT_ConfigReferenceMessage
   - HAL_FDCAN_TT_ConfigTrigger
3. Start the FDCAN module using HAL_FDCAN_Start function. At this level the node is active on the bus: it can send and receive messages.
4. The following Tx control functions can only be called when the FDCAN module is started:
   - HAL_FDCAN_AddMessageToTxFifoQ
   - HAL_FDCAN_EnableTxBufferRequest
   - HAL_FDCAN_AbortTxRequest
5. After having submitted a Tx request in Tx Fifo or Queue, it is possible to get Tx buffer location used to place the Tx request thanks to HAL_FDCAN_GetLatestTxFifoQRequestBuffer API. It is then possible to abort later on the corresponding Tx Request using HAL_FDCAN_AbortTxRequest API.
6. When a message is received into the FDCAN message RAM, it can be retrieved using the HAL_FDCAN_GetRxMessage function.
7. Calling the HAL_FDCAN_Stop function stops the FDCAN module by entering it to initialization mode and re-enabling access to configuration registers through the configuration functions listed here above.
8. All other control functions can be called any time after initialization phase, no matter if the FDCAN module is started or stopped.

Polling mode operation

1. Reception and transmission states can be monitored via the following functions:
   - HAL_FDCAN_IsRxBufferMessageAvailable
   - HAL_FDCAN_IsTxBufferMessagePending
   - HAL_FDCAN_GetRxFifoFillLevel
   - HAL_FDCAN_GetTxFifoFreeLevel

Interrupt mode operation

1. There are two interrupt lines: line 0 and 1. By default, all interrupts are assigned to line 0. Interrupt lines can be configured using HAL_FDCAN_ConfigInterruptLines function.
2. Notifications are activated using HAL_FDCAN_ActivateNotification function. Then, the process can be controlled through one of the available user callbacks: HAL_FDCAN_xxxCallback.

Callback registration

25.2.2 Detailed description of functions

HAL_FDCAN_Init

Function name

HAL_StatusTypeDef HAL_FDCAN_Init (FDCAN_HandleTypeDef * hfdcan)

Function description

HAL_FDCAN_DeInit

Function name

HAL_StatusTypeDef HAL_FDCAN_DeInit (FDCAN_HandleTypeDef * hfdcan)

Function description

HAL_FDCAN_MspInit

Function name

void HAL_FDCAN_MspInit (FDCAN_HandleTypeDef * hfdcan)

Function description

HAL_FDCAN_MspDeInit

Function name

void HAL_FDCAN_MspDeInit (FDCAN_HandleTypeDef * hfdcan)

Function description

HAL_FDCAN_EnterPowerDownMode

Function name

HAL_StatusTypeDef HAL_FDCAN_EnterPowerDownMode (FDCAN_HandleTypeDef * hfdcan)

Function description

HAL_FDCAN_ExitPowerDownMode

Function name

HAL_StatusTypeDef HAL_FDCAN_ExitPowerDownMode (FDCAN_HandleTypeDef * hfdcan)

Function description

HAL_FDCAN_RegisterCallback

Function name

HAL_StatusTypeDef HAL_FDCAN_RegisterCallback (FDCAN_HandleTypeDef * hfdcan, HAL_FDCAN_CallbackIDTypeDef CallbackID, pFDCAN_CallbackTypeDef pCallback)

Function description

HAL_FDCAN_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_FDCAN_UnRegisterCallback (FDCAN_HandleTypeDef * hfdcan, HAL_FDCAN_CallbackIDTypeDef CallbackID)

Function description

HAL_FDCAN_RegisterClockCalibrationCallback

Function name

HAL_StatusTypeDef HAL_FDCAN_RegisterClockCalibrationCallback (FDCAN_HandleTypeDef * hfdcan, pFDCAN_ClockCalibrationCallbackTypeDef pCallback)

Function description

HAL_FDCAN_UnRegisterClockCalibrationCallback

Function name

HAL_StatusTypeDef HAL_FDCAN_UnRegisterClockCalibrationCallback (FDCAN_HandleTypeDef * hfdcan)

Function description

HAL_FDCAN_RegisterTxEventFifoCallback

Function name

HAL_StatusTypeDef HAL_FDCAN_RegisterTxEventFifoCallback (FDCAN_HandleTypeDef * hfdcan, pFDCAN_TxEventFifoCallbackTypeDef pCallback)

Function description

HAL_FDCAN_UnRegisterTxEventFifoCallback

Function name

HAL_StatusTypeDef HAL_FDCAN_UnRegisterTxEventFifoCallback (FDCAN_HandleTypeDef * hfdcan)

Function description

HAL_FDCAN_RegisterRxFifo0Callback

Function name

HAL_StatusTypeDef HAL_FDCAN_RegisterRxFifo0Callback (FDCAN_HandleTypeDef * hfdcan, pFDCAN_RxFifo0CallbackTypeDef pCallback)

Function description

HAL_FDCAN_UnRegisterRxFifo0Callback

Function name

HAL_StatusTypeDef HAL_FDCAN_UnRegisterRxFifo0Callback (FDCAN_HandleTypeDef * hfdcan)

Function description

HAL_FDCAN_RegisterRxFifo1Callback

Function name

HAL_StatusTypeDef HAL_FDCAN_RegisterRxFifo1Callback (FDCAN_HandleTypeDef * hfdcan, pFDCAN_RxFifo1CallbackTypeDef pCallback)

Function description

HAL_FDCAN_UnRegisterRxFifo1Callback

Function name

HAL_StatusTypeDef HAL_FDCAN_UnRegisterRxFifo1Callback (FDCAN_HandleTypeDef * hfdcan)

Function description

HAL_FDCAN_RegisterTxBufferCompleteCallback

Function name

HAL_StatusTypeDef HAL_FDCAN_RegisterTxBufferCompleteCallback (FDCAN_HandleTypeDef * hfdcan, pFDCAN_TxBufferCompleteCallbackTypeDef pCallback)

Function description

HAL_FDCAN_UnRegisterTxBufferCompleteCallback

Function name

HAL_StatusTypeDef HAL_FDCAN_UnRegisterTxBufferCompleteCallback (FDCAN_HandleTypeDef * hfdcan)

Function description

HAL_FDCAN_RegisterTxBufferAbortCallback

Function name

HAL_StatusTypeDef HAL_FDCAN_RegisterTxBufferAbortCallback (FDCAN_HandleTypeDef * hfdcan, pFDCAN_TxBufferAbortCallbackTypeDef pCallback)

Function description

HAL_FDCAN_UnRegisterTxBufferAbortCallback

Function name

HAL_StatusTypeDef HAL_FDCAN_UnRegisterTxBufferAbortCallback (FDCAN_HandleTypeDef * hfdcan)

Function description

HAL_FDCAN_RegisterErrorStatusCallback

Function name

HAL_StatusTypeDef HAL_FDCAN_RegisterErrorStatusCallback (FDCAN_HandleTypeDef * hfdcan, pFDCAN_ErrorStatusCallbackTypeDef pCallback)

Function description

HAL_FDCAN_UnRegisterErrorStatusCallback

Function name

HAL_StatusTypeDef HAL_FDCAN_UnRegisterErrorStatusCallback (FDCAN_HandleTypeDef * hfdcan)

Function description

HAL_FDCAN_RegisterTTScheduleSyncCallback

Function name

HAL_StatusTypeDef HAL_FDCAN_RegisterTTScheduleSyncCallback (FDCAN_HandleTypeDef * hfdcan, pFDCAN_TT_ScheduleSyncCallbackTypeDef pCallback)

Function description

HAL_FDCAN_UnRegisterTTScheduleSyncCallback

Function name

HAL_StatusTypeDef HAL_FDCAN_UnRegisterTTScheduleSyncCallback (FDCAN_HandleTypeDef * hfdcan)

Function description

HAL_FDCAN_RegisterTTTimeMarkCallback

Function name

HAL_StatusTypeDef HAL_FDCAN_RegisterTTTimeMarkCallback (FDCAN_HandleTypeDef * hfdcan, pFDCAN_TT_TimeMarkCallbackTypeDef pCallback)

Function description

HAL_FDCAN_UnRegisterTTTimeMarkCallback

Function name

HAL_StatusTypeDef HAL_FDCAN_UnRegisterTTTimeMarkCallback (FDCAN_HandleTypeDef * hfdcan)

Function description

HAL_FDCAN_RegisterTTStopWatchCallback

Function name

HAL_StatusTypeDef HAL_FDCAN_RegisterTTStopWatchCallback (FDCAN_HandleTypeDef * hfdcan, pFDCAN_TT_StopWatchCallbackTypeDef pCallback)

Function description

HAL_FDCAN_UnRegisterTTStopWatchCallback

Function name

HAL_StatusTypeDef HAL_FDCAN_UnRegisterTTStopWatchCallback (FDCAN_HandleTypeDef * hfdcan)

Function description

HAL_FDCAN_RegisterTTGlobalTimeCallback

Function name

HAL_StatusTypeDef HAL_FDCAN_RegisterTTGlobalTimeCallback (FDCAN_HandleTypeDef * hfdcan, pFDCAN_TT_GlobalTimeCallbackTypeDef pCallback)

Function description

HAL_FDCAN_UnRegisterTTGlobalTimeCallback

Function name

HAL_StatusTypeDef HAL_FDCAN_UnRegisterTTGlobalTimeCallback (FDCAN_HandleTypeDef * hfdcan)

Function description

HAL_FDCAN_ConfigClockCalibration

Function name

HAL_StatusTypeDef HAL_FDCAN_ConfigClockCalibration (FDCAN_HandleTypeDef * hfdcan, const FDCAN_ClkCalUnitTypeDef * sCcuConfig)

Function description

HAL_FDCAN_GetClockCalibrationState

Function name

uint32_t HAL_FDCAN_GetClockCalibrationState (const FDCAN_HandleTypeDef * hfdcan)

Function description

HAL_FDCAN_ResetClockCalibrationState

Function name

HAL_StatusTypeDef HAL_FDCAN_ResetClockCalibrationState (FDCAN_HandleTypeDef * hfdcan)

Function description

HAL_FDCAN_GetClockCalibrationCounter

Function name

uint32_t HAL_FDCAN_GetClockCalibrationCounter (const FDCAN_HandleTypeDef * hfdcan, uint32_t Counter)

Function description

HAL_FDCAN_ConfigFilter

Function name

HAL_StatusTypeDef HAL_FDCAN_ConfigFilter (FDCAN_HandleTypeDef * hfdcan, const FDCAN_FilterTypeDef * sFilterConfig)

Function description

HAL_FDCAN_ConfigGlobalFilter

Function name

HAL_StatusTypeDef HAL_FDCAN_ConfigGlobalFilter (FDCAN_HandleTypeDef * hfdcan, uint32_t NonMatchingStd, uint32_t NonMatchingExt, uint32_t RejectRemoteStd, uint32_t RejectRemoteExt)

Function description

HAL_FDCAN_ConfigExtendedIdMask

Function name

HAL_StatusTypeDef HAL_FDCAN_ConfigExtendedIdMask (FDCAN_HandleTypeDef * hfdcan, uint32_t Mask)

Function description

HAL_FDCAN_ConfigRxFifoOverwrite

Function name

HAL_StatusTypeDef HAL_FDCAN_ConfigRxFifoOverwrite (FDCAN_HandleTypeDef * hfdcan, uint32_t RxFifo, uint32_t OperationMode)

Function description

HAL_FDCAN_ConfigFifoWatermark

Function name

HAL_StatusTypeDef HAL_FDCAN_ConfigFifoWatermark (FDCAN_HandleTypeDef * hfdcan, uint32_t FIFO, uint32_t Watermark)

Function description

HAL_FDCAN_ConfigRamWatchdog

Function name

HAL_StatusTypeDef HAL_FDCAN_ConfigRamWatchdog (FDCAN_HandleTypeDef * hfdcan, uint32_t CounterStartValue)

Function description

HAL_FDCAN_ConfigTimestampCounter

Function name

HAL_StatusTypeDef HAL_FDCAN_ConfigTimestampCounter (FDCAN_HandleTypeDef * hfdcan, uint32_t TimestampPrescaler)

Function description

HAL_FDCAN_EnableTimestampCounter

Function name

HAL_StatusTypeDef HAL_FDCAN_EnableTimestampCounter (FDCAN_HandleTypeDef * hfdcan, uint32_t TimestampOperation)

Function description

HAL_FDCAN_DisableTimestampCounter

Function name

HAL_StatusTypeDef HAL_FDCAN_DisableTimestampCounter (FDCAN_HandleTypeDef * hfdcan)

Function description

HAL_FDCAN_GetTimestampCounter

Function name

uint16_t HAL_FDCAN_GetTimestampCounter (const FDCAN_HandleTypeDef * hfdcan)

Function description

HAL_FDCAN_ResetTimestampCounter

Function name

HAL_StatusTypeDef HAL_FDCAN_ResetTimestampCounter (FDCAN_HandleTypeDef * hfdcan)

Function description

HAL_FDCAN_ConfigTimeoutCounter

Function name

HAL_StatusTypeDef HAL_FDCAN_ConfigTimeoutCounter (FDCAN_HandleTypeDef * hfdcan, uint32_t TimeoutOperation, uint32_t TimeoutPeriod)

Function description

HAL_FDCAN_EnableTimeoutCounter

Function name

HAL_StatusTypeDef HAL_FDCAN_EnableTimeoutCounter (FDCAN_HandleTypeDef * hfdcan)

Function description

HAL_FDCAN_DisableTimeoutCounter

Function name

HAL_StatusTypeDef HAL_FDCAN_DisableTimeoutCounter (FDCAN_HandleTypeDef * hfdcan)

Function description

HAL_FDCAN_GetTimeoutCounter

Function name

uint16_t HAL_FDCAN_GetTimeoutCounter (const FDCAN_HandleTypeDef * hfdcan)

Function description

HAL_FDCAN_ResetTimeoutCounter

Function name

HAL_StatusTypeDef HAL_FDCAN_ResetTimeoutCounter (FDCAN_HandleTypeDef * hfdcan)

Function description

HAL_FDCAN_ConfigTxDelayCompensation

Function name

HAL_StatusTypeDef HAL_FDCAN_ConfigTxDelayCompensation (FDCAN_HandleTypeDef * hfdcan, uint32_t TdcOffset, uint32_t TdcFilter)

Function description

HAL_FDCAN_EnableTxDelayCompensation

Function name

HAL_StatusTypeDef HAL_FDCAN_EnableTxDelayCompensation (FDCAN_HandleTypeDef * hfdcan)

Function description

HAL_FDCAN_DisableTxDelayCompensation

Function name

HAL_StatusTypeDef HAL_FDCAN_DisableTxDelayCompensation (FDCAN_HandleTypeDef * hfdcan)

Function description

HAL_FDCAN_EnableISOMode

Function name

HAL_StatusTypeDef HAL_FDCAN_EnableISOMode (FDCAN_HandleTypeDef * hfdcan)

Function description

HAL_FDCAN_DisableISOMode

Function name

HAL_StatusTypeDef HAL_FDCAN_DisableISOMode (FDCAN_HandleTypeDef * hfdcan)

Function description

HAL_FDCAN_EnableEdgeFiltering

Function name

HAL_StatusTypeDef HAL_FDCAN_EnableEdgeFiltering (FDCAN_HandleTypeDef * hfdcan)

Function description

HAL_FDCAN_DisableEdgeFiltering

Function name

HAL_StatusTypeDef HAL_FDCAN_DisableEdgeFiltering (FDCAN_HandleTypeDef * hfdcan)

Function description

HAL_FDCAN_Start

Function name

HAL_StatusTypeDef HAL_FDCAN_Start (FDCAN_HandleTypeDef * hfdcan)

Function description

HAL_FDCAN_Stop

Function name

HAL_StatusTypeDef HAL_FDCAN_Stop (FDCAN_HandleTypeDef * hfdcan)

Function description

HAL_FDCAN_AddMessageToTxFifoQ

Function name

HAL_StatusTypeDef HAL_FDCAN_AddMessageToTxFifoQ (FDCAN_HandleTypeDef * hfdcan, const FDCAN_TxHeaderTypeDef * pTxHeader, const uint8_t * pTxData)

Function description

HAL_FDCAN_AddMessageToTxBuffer

Function name

HAL_StatusTypeDef HAL_FDCAN_AddMessageToTxBuffer (FDCAN_HandleTypeDef * hfdcan, const FDCAN_TxHeaderTypeDef * pTxHeader, const uint8_t * pTxData, uint32_t BufferIndex)

Function description

HAL_FDCAN_EnableTxBufferRequest

Function name

HAL_StatusTypeDef HAL_FDCAN_EnableTxBufferRequest (FDCAN_HandleTypeDef * hfdcan, uint32_t BufferIndex)

Function description

HAL_FDCAN_GetLatestTxFifoQRequestBuffer

Function name

uint32_t HAL_FDCAN_GetLatestTxFifoQRequestBuffer (const FDCAN_HandleTypeDef * hfdcan)

Function description

HAL_FDCAN_AbortTxRequest

Function name

HAL_StatusTypeDef HAL_FDCAN_AbortTxRequest (FDCAN_HandleTypeDef * hfdcan, uint32_t BufferIndex)

Function description

HAL_FDCAN_GetRxMessage

Function name

HAL_StatusTypeDef HAL_FDCAN_GetRxMessage (FDCAN_HandleTypeDef * hfdcan, uint32_t RxLocation, FDCAN_RxHeaderTypeDef * pRxHeader, uint8_t * pRxData)

Function description

HAL_FDCAN_GetTxEvent

Function name

HAL_StatusTypeDef HAL_FDCAN_GetTxEvent (FDCAN_HandleTypeDef * hfdcan, FDCAN_TxEventFifoTypeDef * pTxEvent)

Function description

HAL_FDCAN_GetHighPriorityMessageStatus

Function name

HAL_StatusTypeDef HAL_FDCAN_GetHighPriorityMessageStatus (const FDCAN_HandleTypeDef * hfdcan, FDCAN_HpMsgStatusTypeDef * HpMsgStatus)

Function description

HAL_FDCAN_GetProtocolStatus

Function name

HAL_StatusTypeDef HAL_FDCAN_GetProtocolStatus (const FDCAN_HandleTypeDef * hfdcan, FDCAN_ProtocolStatusTypeDef * ProtocolStatus)

Function description

HAL_FDCAN_GetErrorCounters

Function name

HAL_StatusTypeDef HAL_FDCAN_GetErrorCounters (const FDCAN_HandleTypeDef * hfdcan, FDCAN_ErrorCountersTypeDef * ErrorCounters)

Function description

HAL_FDCAN_IsRxBufferMessageAvailable

Function name

uint32_t HAL_FDCAN_IsRxBufferMessageAvailable (FDCAN_HandleTypeDef * hfdcan, uint32_t RxBufferIndex)

Function description

HAL_FDCAN_IsTxBufferMessagePending

Function name

uint32_t HAL_FDCAN_IsTxBufferMessagePending (const FDCAN_HandleTypeDef * hfdcan, uint32_t TxBufferIndex)

Function description

HAL_FDCAN_GetRxFifoFillLevel

Function name

uint32_t HAL_FDCAN_GetRxFifoFillLevel (const FDCAN_HandleTypeDef * hfdcan, uint32_t RxFifo)

Function description

HAL_FDCAN_GetTxFifoFreeLevel

Function name

uint32_t HAL_FDCAN_GetTxFifoFreeLevel (const FDCAN_HandleTypeDef * hfdcan)

Function description

HAL_FDCAN_IsRestrictedOperationMode

Function name

uint32_t HAL_FDCAN_IsRestrictedOperationMode (const FDCAN_HandleTypeDef * hfdcan)

Function description

HAL_FDCAN_ExitRestrictedOperationMode

Function name

HAL_StatusTypeDef HAL_FDCAN_ExitRestrictedOperationMode (FDCAN_HandleTypeDef * hfdcan)

Function description

HAL_FDCAN_TT_ConfigOperation

Function name

HAL_StatusTypeDef HAL_FDCAN_TT_ConfigOperation (FDCAN_HandleTypeDef * hfdcan, const FDCAN_TT_ConfigTypeDef * pTTParams)

Function description

HAL_FDCAN_TT_ConfigReferenceMessage

Function name

HAL_StatusTypeDef HAL_FDCAN_TT_ConfigReferenceMessage (FDCAN_HandleTypeDef * hfdcan, uint32_t IdType, uint32_t Identifier, uint32_t Payload)

Function description

HAL_FDCAN_TT_ConfigTrigger

Function name

HAL_StatusTypeDef HAL_FDCAN_TT_ConfigTrigger (FDCAN_HandleTypeDef * hfdcan, const FDCAN_TriggerTypeDef * sTriggerConfig)

Function description

HAL_FDCAN_TT_SetGlobalTime

Function name

HAL_StatusTypeDef HAL_FDCAN_TT_SetGlobalTime (FDCAN_HandleTypeDef * hfdcan, uint32_t TimePreset)

Function description

HAL_FDCAN_TT_SetClockSynchronization

Function name

HAL_StatusTypeDef HAL_FDCAN_TT_SetClockSynchronization (FDCAN_HandleTypeDef * hfdcan, uint32_t NewTURNumerator)

Function description

HAL_FDCAN_TT_ConfigStopWatch

Function name

HAL_StatusTypeDef HAL_FDCAN_TT_ConfigStopWatch (FDCAN_HandleTypeDef * hfdcan, uint32_t Source, uint32_t Polarity)

Function description

HAL_FDCAN_TT_ConfigRegisterTimeMark

Function name

HAL_StatusTypeDef HAL_FDCAN_TT_ConfigRegisterTimeMark (FDCAN_HandleTypeDef * hfdcan, uint32_t TimeMarkSource, uint32_t TimeMarkValue, uint32_t RepeatFactor, uint32_t StartCycle)

Function description

HAL_FDCAN_TT_EnableRegisterTimeMarkPulse

Function name

HAL_StatusTypeDef HAL_FDCAN_TT_EnableRegisterTimeMarkPulse (FDCAN_HandleTypeDef * hfdcan)

Function description

HAL_FDCAN_TT_DisableRegisterTimeMarkPulse

Function name

HAL_StatusTypeDef HAL_FDCAN_TT_DisableRegisterTimeMarkPulse (FDCAN_HandleTypeDef * hfdcan)

Function description

HAL_FDCAN_TT_EnableTriggerTimeMarkPulse

Function name

HAL_StatusTypeDef HAL_FDCAN_TT_EnableTriggerTimeMarkPulse (FDCAN_HandleTypeDef * hfdcan)

Function description

HAL_FDCAN_TT_DisableTriggerTimeMarkPulse

Function name

HAL_StatusTypeDef HAL_FDCAN_TT_DisableTriggerTimeMarkPulse (FDCAN_HandleTypeDef * hfdcan)

Function description

HAL_FDCAN_TT_EnableHardwareGapControl

Function name

HAL_StatusTypeDef HAL_FDCAN_TT_EnableHardwareGapControl (FDCAN_HandleTypeDef * hfdcan)

Function description

HAL_FDCAN_TT_DisableHardwareGapControl

Function name

HAL_StatusTypeDef HAL_FDCAN_TT_DisableHardwareGapControl (FDCAN_HandleTypeDef * hfdcan)

Function description

HAL_FDCAN_TT_EnableTimeMarkGapControl

Function name

HAL_StatusTypeDef HAL_FDCAN_TT_EnableTimeMarkGapControl (FDCAN_HandleTypeDef * hfdcan)

Function description

HAL_FDCAN_TT_DisableTimeMarkGapControl

Function name

HAL_StatusTypeDef HAL_FDCAN_TT_DisableTimeMarkGapControl (FDCAN_HandleTypeDef * hfdcan)

Function description

HAL_FDCAN_TT_SetNextIsGap

Function name

HAL_StatusTypeDef HAL_FDCAN_TT_SetNextIsGap (FDCAN_HandleTypeDef * hfdcan)

Function description

HAL_FDCAN_TT_SetEndOfGap

Function name

HAL_StatusTypeDef HAL_FDCAN_TT_SetEndOfGap (FDCAN_HandleTypeDef * hfdcan)

Function description

HAL_FDCAN_TT_ConfigExternalSyncPhase

Function name

HAL_StatusTypeDef HAL_FDCAN_TT_ConfigExternalSyncPhase (FDCAN_HandleTypeDef * hfdcan, uint32_t TargetPhase)

Function description

HAL_FDCAN_TT_EnableExternalSynchronization

Function name

HAL_StatusTypeDef HAL_FDCAN_TT_EnableExternalSynchronization (FDCAN_HandleTypeDef * hfdcan)

Function description

HAL_FDCAN_TT_DisableExternalSynchronization

Function name

HAL_StatusTypeDef HAL_FDCAN_TT_DisableExternalSynchronization (FDCAN_HandleTypeDef * hfdcan)

Function description

HAL_FDCAN_TT_GetOperationStatus

Function name

HAL_StatusTypeDef HAL_FDCAN_TT_GetOperationStatus (const FDCAN_HandleTypeDef * hfdcan, FDCAN_TTOperationStatusTypeDef * TTOpStatus)

Function description

HAL_FDCAN_ConfigInterruptLines

Function name

HAL_StatusTypeDef HAL_FDCAN_ConfigInterruptLines (FDCAN_HandleTypeDef * hfdcan, uint32_t ITList, uint32_t InterruptLine)

Function description

HAL_FDCAN_TT_ConfigInterruptLines

Function name

HAL_StatusTypeDef HAL_FDCAN_TT_ConfigInterruptLines (FDCAN_HandleTypeDef * hfdcan, uint32_t TTITList, uint32_t InterruptLine)

Function description

HAL_FDCAN_ActivateNotification

Function name

HAL_StatusTypeDef HAL_FDCAN_ActivateNotification (FDCAN_HandleTypeDef * hfdcan, uint32_t ActiveITs, uint32_t BufferIndexes)

Function description

HAL_FDCAN_DeactivateNotification

Function name

HAL_StatusTypeDef HAL_FDCAN_DeactivateNotification (FDCAN_HandleTypeDef * hfdcan, uint32_t InactiveITs)

Function description

HAL_FDCAN_TT_ActivateNotification

Function name

HAL_StatusTypeDef HAL_FDCAN_TT_ActivateNotification (FDCAN_HandleTypeDef * hfdcan, uint32_t ActiveTTITs)

Function description

HAL_FDCAN_TT_DeactivateNotification

Function name

HAL_StatusTypeDef HAL_FDCAN_TT_DeactivateNotification (FDCAN_HandleTypeDef * hfdcan, uint32_t InactiveTTITs)

Function description

HAL_FDCAN_IRQHandler

Function name

void HAL_FDCAN_IRQHandler (FDCAN_HandleTypeDef * hfdcan)

Function description

HAL_FDCAN_ClockCalibrationCallback

Function name

void HAL_FDCAN_ClockCalibrationCallback (FDCAN_HandleTypeDef * hfdcan, uint32_t ClkCalibrationITs)

Function description

HAL_FDCAN_TxEventFifoCallback

Function name

void HAL_FDCAN_TxEventFifoCallback (FDCAN_HandleTypeDef * hfdcan, uint32_t TxEventFifoITs)

Function description

HAL_FDCAN_RxFifo0Callback

Function name

void HAL_FDCAN_RxFifo0Callback (FDCAN_HandleTypeDef * hfdcan, uint32_t RxFifo0ITs)

Function description

HAL_FDCAN_RxFifo1Callback

Function name

void HAL_FDCAN_RxFifo1Callback (FDCAN_HandleTypeDef * hfdcan, uint32_t RxFifo1ITs)

Function description

HAL_FDCAN_TxFifoEmptyCallback

Function name

void HAL_FDCAN_TxFifoEmptyCallback (FDCAN_HandleTypeDef * hfdcan)

Function description

HAL_FDCAN_TxBufferCompleteCallback

Function name

void HAL_FDCAN_TxBufferCompleteCallback (FDCAN_HandleTypeDef * hfdcan, uint32_t BufferIndexes)

Function description

HAL_FDCAN_TxBufferAbortCallback

Function name

void HAL_FDCAN_TxBufferAbortCallback (FDCAN_HandleTypeDef * hfdcan, uint32_t BufferIndexes)

Function description

HAL_FDCAN_RxBufferNewMessageCallback

Function name

void HAL_FDCAN_RxBufferNewMessageCallback (FDCAN_HandleTypeDef * hfdcan)

Function description

HAL_FDCAN_HighPriorityMessageCallback

Function name

void HAL_FDCAN_HighPriorityMessageCallback (FDCAN_HandleTypeDef * hfdcan)

Function description

HAL_FDCAN_TimestampWraparoundCallback

Function name

void HAL_FDCAN_TimestampWraparoundCallback (FDCAN_HandleTypeDef * hfdcan)

Function description

HAL_FDCAN_TimeoutOccurredCallback

Function name

void HAL_FDCAN_TimeoutOccurredCallback (FDCAN_HandleTypeDef * hfdcan)

Function description

HAL_FDCAN_ErrorCallback

Function name

void HAL_FDCAN_ErrorCallback (FDCAN_HandleTypeDef * hfdcan)

Function description

HAL_FDCAN_ErrorStatusCallback

Function name

void HAL_FDCAN_ErrorStatusCallback (FDCAN_HandleTypeDef * hfdcan, uint32_t ErrorStatusITs)

Function description

HAL_FDCAN_TT_ScheduleSyncCallback

Function name

void HAL_FDCAN_TT_ScheduleSyncCallback (FDCAN_HandleTypeDef * hfdcan, uint32_t TTSchedSyncITs)

Function description

HAL_FDCAN_TT_TimeMarkCallback

Function name

void HAL_FDCAN_TT_TimeMarkCallback (FDCAN_HandleTypeDef * hfdcan, uint32_t TTTimeMarkITs)

Function description

HAL_FDCAN_TT_StopWatchCallback

Function name

void HAL_FDCAN_TT_StopWatchCallback (FDCAN_HandleTypeDef * hfdcan, uint32_t SWTime, uint32_t SWCycleCount)

Function description

HAL_FDCAN_TT_GlobalTimeCallback

Function name

void HAL_FDCAN_TT_GlobalTimeCallback (FDCAN_HandleTypeDef * hfdcan, uint32_t TTGlobTimeIts)

Function description

HAL_FDCAN_GetError

Function name

uint32_t HAL_FDCAN_GetError (const FDCAN_HandleTypeDef * hfdcan)

Function description

HAL_FDCAN_GetState

Function name

HAL_FDCAN_StateTypeDef HAL_FDCAN_GetState (const FDCAN_HandleTypeDef * hfdcan)

Function description

25.3 FDCAN Firmware driver defines

The following section lists the various define and macros of the module.

25.3.1 FDCAN

FDCAN

FDCAN Bit Rate Switching

FDCAN_BRS_OFF

FDCAN frames transmitted/received without bit rate switching

FDCAN_BRS_ON

FDCAN frames transmitted/received with bit rate switching

FDCAN Calibration Counter

FDCAN_CALIB_TIME_QUANTA_COUNTER

Time Quanta Counter

FDCAN_CALIB_CLOCK_PERIOD_COUNTER

Oscillator Clock Period Counter

FDCAN_CALIB_WATCHDOG_COUNTER

Calibration Watchdog Counter

FDCAN Calibration Field Length

FDCAN_CALIB_FIELD_LENGTH_32

Calibration field length is 32 bits

FDCAN_CALIB_FIELD_LENGTH_64

Calibration field length is 64 bits

FDCAN Calibration State

FDCAN_CLOCK_NOT_CALIBRATED

Clock not calibrated

FDCAN_CLOCK_BASIC_CALIBRATED

Clock basic calibrated

FDCAN_CLOCK_PRECISION_CALIBRATED

Clock precision calibrated

FDCAN Clock Calibration

FDCAN_CLOCK_CALIBRATION_DISABLE

Disable Clock Calibration

FDCAN_CLOCK_CALIBRATION_ENABLE

Enable Clock Calibration

Clock Calibration Interrupts

FDCAN_IT_CALIB_STATE_CHANGED

Clock calibration state changed

FDCAN_IT_CALIB_WATCHDOG_EVENT

Clock calibration watchdog event occurred

FDCAN Clock Divider

FDCAN_CLOCK_DIV1

Divide kernel clock by 1

FDCAN_CLOCK_DIV2

Divide kernel clock by 2

FDCAN_CLOCK_DIV4

Divide kernel clock by 4

FDCAN_CLOCK_DIV6

Divide kernel clock by 6

FDCAN_CLOCK_DIV8

Divide kernel clock by 8

FDCAN_CLOCK_DIV10

Divide kernel clock by 10

FDCAN_CLOCK_DIV12

Divide kernel clock by 12

FDCAN_CLOCK_DIV14

Divide kernel clock by 14

FDCAN_CLOCK_DIV16

Divide kernel clock by 16

FDCAN_CLOCK_DIV18

Divide kernel clock by 18

FDCAN_CLOCK_DIV20

Divide kernel clock by 20

FDCAN_CLOCK_DIV22

Divide kernel clock by 22

FDCAN_CLOCK_DIV24

Divide kernel clock by 24

FDCAN_CLOCK_DIV26

Divide kernel clock by 26

FDCAN_CLOCK_DIV28

Divide kernel clock by 28

FDCAN_CLOCK_DIV30

Divide kernel clock by 30

FDCAN communication state

FDCAN_COM_STATE_SYNC

Node is synchronizing on CAN communication

FDCAN_COM_STATE_IDLE

Node is neither receiver nor transmitter

FDCAN_COM_STATE_RX

Node is operating as receiver

FDCAN_COM_STATE_TX

Node is operating as transmitter

FDCAN Counter Interrupts

FDCAN_IT_TIMESTAMP_WRAPAROUND

Timestamp counter wrapped around

FDCAN_IT_TIMEOUT_OCCURRED

Timeout reached

FDCAN Data Field Size

FDCAN_DATA_BYTES_8

8 bytes data field

FDCAN_DATA_BYTES_12

12 bytes data field

FDCAN_DATA_BYTES_16

16 bytes data field

FDCAN_DATA_BYTES_20

20 bytes data field

FDCAN_DATA_BYTES_24

24 bytes data field

FDCAN_DATA_BYTES_32

32 bytes data field

FDCAN_DATA_BYTES_48

48 bytes data field

FDCAN_DATA_BYTES_64

64 bytes data field

FDCAN Data Length Code

FDCAN_DLC_BYTES_0

0 bytes data field

FDCAN_DLC_BYTES_1

1 bytes data field

FDCAN_DLC_BYTES_2

2 bytes data field

FDCAN_DLC_BYTES_3

3 bytes data field

FDCAN_DLC_BYTES_4

4 bytes data field

FDCAN_DLC_BYTES_5

5 bytes data field

FDCAN_DLC_BYTES_6

6 bytes data field

FDCAN_DLC_BYTES_7

7 bytes data field

FDCAN_DLC_BYTES_8

8 bytes data field

FDCAN_DLC_BYTES_12

12 bytes data field

FDCAN_DLC_BYTES_16

16 bytes data field

FDCAN_DLC_BYTES_20

20 bytes data field

FDCAN_DLC_BYTES_24

24 bytes data field

FDCAN_DLC_BYTES_32

32 bytes data field

FDCAN_DLC_BYTES_48

48 bytes data field

FDCAN_DLC_BYTES_64

64 bytes data field

FDCAN Error Interrupts

FDCAN_IT_RAM_ACCESS_FAILURE

Message RAM access failure occurred

FDCAN_IT_ERROR_LOGGING_OVERFLOW

Overflow of FDCAN Error Logging Counter occurred

FDCAN_IT_RAM_WATCHDOG

Message RAM Watchdog event due to missing READY

FDCAN_IT_ARB_PROTOCOL_ERROR

Protocol error in arbitration phase detected

FDCAN_IT_DATA_PROTOCOL_ERROR

Protocol error in data phase detected

FDCAN_IT_RESERVED_ADDRESS_ACCESS

Access to reserved address occurred

FDCAN Error State Indicator

FDCAN_ESI_ACTIVE

Transmitting node is error active

FDCAN_ESI_PASSIVE

Transmitting node is error passive

FDCAN Error Status Interrupts

FDCAN_IT_ERROR_PASSIVE

Error_Passive status changed

FDCAN_IT_ERROR_WARNING

Error_Warning status changed

FDCAN_IT_BUS_OFF

Bus_Off status changed

FDCAN Event Type

FDCAN_TX_EVENT

Tx event

FDCAN_TX_IN_SPITE_OF_ABORT

Transmission in spite of cancellation

FDCAN Exported Macros

__HAL_FDCAN_RESET_HANDLE_STATE

Description:

- Reset FDCAN handle state.

Parameters:

- __HANDLE__: FDCAN handle.

Return value:

- None

__HAL_FDCAN_ENABLE_IT

Description:

- Enable the specified FDCAN interrupts.

Parameters:

- __HANDLE__: FDCAN handle.
- __INTERRUPT__: FDCAN interrupt. This parameter can be any combination of
  - FDCAN_Interrupts

Return value:

- None

__HAL_FDCAN_DISABLE_IT

Description:

- Disable the specified FDCAN interrupts.

Parameters:

- __HANDLE__: FDCAN handle.
- __INTERRUPT__: FDCAN interrupt. This parameter can be any combination of
  - FDCAN_Interrupts

Return value:

- None

__HAL_FDCAN_GET_IT

Description:

- Check whether the specified FDCAN interrupt is set or not.

Parameters:

- __HANDLE__: FDCAN handle.
- __INTERRUPT__: FDCAN interrupt. This parameter can be one of
  - FDCAN_Interrupts

Return value:

- ITStatus

__HAL_FDCAN_CLEAR_IT

Description:

- Clear the specified FDCAN interrupts.

Parameters:

- __HANDLE__: FDCAN handle.
- __INTERRUPT__: specifies the interrupts to clear. This parameter can be any combination of
  - FDCAN_Interrupts

Return value:

- None

__HAL_FDCAN_GET_FLAG

Description:

- Check whether the specified FDCAN flag is set or not.

Parameters:

- __HANDLE__: FDCAN handle.
- __FLAG__: FDCAN flag. This parameter can be one of
  - FDCAN_flags

Return value:

- FlagStatus

__HAL_FDCAN_CLEAR_FLAG

Description:

- Clear the specified FDCAN flags.

Parameters:

- __HANDLE__: FDCAN handle.
- __FLAG__: specifies the flags to clear. This parameter can be any combination of
  - FDCAN_flags

Return value:

- None

__HAL_FDCAN_GET_IT_SOURCE

Description:

- Check if the specified FDCAN interrupt source is enabled or disabled.

Parameters:

- __HANDLE__: FDCAN handle.
- __INTERRUPT__: specifies the FDCAN interrupt source to check. This parameter can be a value of
  - FDCAN_Interrupts

Return value:

- ITStatus

__HAL_FDCAN_TT_ENABLE_IT

Description:

- Enable the specified FDCAN TT interrupts.

Parameters:

- __HANDLE__: FDCAN handle.
- __INTERRUPT__: FDCAN TT interrupt. This parameter can be any combination of
  - FDCAN_TTInterrupts

Return value:

- None

__HAL_FDCAN_TT_DISABLE_IT

Description:

- Disable the specified FDCAN TT interrupts.

Parameters:

- __HANDLE__: FDCAN handle.
- __INTERRUPT__: FDCAN TT interrupt. This parameter can be any combination of
  - FDCAN_TTInterrupts

Return value:

- None

__HAL_FDCAN_TT_GET_IT

Description:

- Check whether the specified FDCAN TT interrupt is set or not.

Parameters:

- __HANDLE__: FDCAN handle.
- __INTERRUPT__: FDCAN TT interrupt. This parameter can be one of
  - FDCAN_TTInterrupts

Return value:

- ITStatus

__HAL_FDCAN_TT_CLEAR_IT

Description:

- Clear the specified FDCAN TT interrupts.

Parameters:

- __HANDLE__: FDCAN handle.
- __INTERRUPT__: specifies the TT interrupts to clear. This parameter can be any combination of
  - FDCAN_TTInterrupts

Return value:

- None

__HAL_FDCAN_TT_GET_FLAG

Description:

- Check whether the specified FDCAN TT flag is set or not.

Parameters:

- __HANDLE__: FDCAN handle.
- __FLAG__: FDCAN TT flag. This parameter can be one of
  - FDCAN_TTflags

Return value:

- FlagStatus

__HAL_FDCAN_TT_CLEAR_FLAG

Description:

- Clear the specified FDCAN TT flags.

Parameters:

- __HANDLE__: FDCAN handle.
- __FLAG__: specifies the TT flags to clear. This parameter can be any combination of
  - FDCAN_TTflags

Return value:

- None

__HAL_FDCAN_TT_GET_IT_SOURCE

Description:

- Check if the specified FDCAN TT interrupt source is enabled or disabled.

Parameters:

- __HANDLE__: FDCAN handle.
- __INTERRUPT__: specifies the FDCAN TT interrupt source to check. This parameter can be a value of
  - FDCAN_TTInterrupts

Return value:

- ITStatus

FDCAN FIFO watermark

FDCAN_CFG_TX_EVENT_FIFO

Tx event FIFO

FDCAN_CFG_RX_FIFO0

Rx FIFO0

FDCAN_CFG_RX_FIFO1

Rx FIFO1

FDCAN Filter Configuration

FDCAN_FILTER_DISABLE

Disable filter element

FDCAN_FILTER_TO_RXFIFO0

Store in Rx FIFO 0 if filter matches

FDCAN_FILTER_TO_RXFIFO1

Store in Rx FIFO 1 if filter matches

FDCAN_FILTER_REJECT

Reject ID if filter matches

FDCAN_FILTER_HP

Set high priority if filter matches

FDCAN_FILTER_TO_RXFIFO0_HP

Set high priority and store in FIFO 0 if filter matches

FDCAN_FILTER_TO_RXFIFO1_HP

Set high priority and store in FIFO 1 if filter matches

FDCAN_FILTER_TO_RXBUFFER

Store into Rx Buffer, configuration of FilterType ignored

FDCAN Filter Type

FDCAN_FILTER_RANGE

Range filter from FilterID1 to FilterID2

FDCAN_FILTER_DUAL

Dual ID filter for FilterID1 or FilterID2

FDCAN_FILTER_MASK

Classic filter: FilterID1 = filter, FilterID2 = mask

FDCAN_FILTER_RANGE_NO_EIDM

Range filter from FilterID1 to FilterID2, EIDM mask not applied

FDCAN Flags

FDCAN_FLAG_TX_COMPLETE

Transmission Completed

FDCAN_FLAG_TX_ABORT_COMPLETE

Transmission Cancellation Finished

FDCAN_FLAG_TX_FIFO_EMPTY

Tx FIFO Empty

FDCAN_FLAG_RX_HIGH_PRIORITY_MSG

High priority message received

FDCAN_FLAG_RX_BUFFER_NEW_MESSAGE

At least one received message stored into a Rx Buffer

FDCAN_FLAG_TX_EVT_FIFO_ELT_LOST

Tx Event FIFO element lost

FDCAN_FLAG_TX_EVT_FIFO_FULL

Tx Event FIFO full

FDCAN_FLAG_TX_EVT_FIFO_WATERMARK

Tx Event FIFO fill level reached watermark

FDCAN_FLAG_TX_EVT_FIFO_NEW_DATA

Tx Handler wrote Tx Event FIFO element

FDCAN_FLAG_RX_FIFO0_MESSAGE_LOST

Rx FIFO 0 message lost

FDCAN_FLAG_RX_FIFO0_FULL

Rx FIFO 0 full

FDCAN_FLAG_RX_FIFO0_WATERMARK

Rx FIFO 0 fill level reached watermark

FDCAN_FLAG_RX_FIFO0_NEW_MESSAGE

New message written to Rx FIFO 0

FDCAN_FLAG_RX_FIFO1_MESSAGE_LOST

Rx FIFO 1 message lost

FDCAN_FLAG_RX_FIFO1_FULL

Rx FIFO 1 full

FDCAN_FLAG_RX_FIFO1_WATERMARK

Rx FIFO 1 fill level reached watermark

FDCAN_FLAG_RX_FIFO1_NEW_MESSAGE

New message written to Rx FIFO 1

FDCAN_FLAG_RAM_ACCESS_FAILURE

Message RAM access failure occurred

FDCAN_FLAG_ERROR_LOGGING_OVERFLOW

Overflow of FDCAN Error Logging Counter occurred

FDCAN_FLAG_ERROR_PASSIVE

Error_Passive status changed

FDCAN_FLAG_ERROR_WARNING

Error_Warning status changed

FDCAN_FLAG_BUS_OFF

Bus_Off status changed

FDCAN_FLAG_RAM_WATCHDOG

Message RAM Watchdog event due to missing READY

FDCAN_FLAG_ARB_PROTOCOL_ERROR

Protocol error in arbitration phase detected

FDCAN_FLAG_DATA_PROTOCOL_ERROR

Protocol error in data phase detected

FDCAN_FLAG_RESERVED_ADDRESS_ACCESS

Access to reserved address occurred

FDCAN_FLAG_TIMESTAMP_WRAPAROUND

Timestamp counter wrapped around

FDCAN_FLAG_TIMEOUT_OCCURRED

Timeout reached

FDCAN_FLAG_CALIB_STATE_CHANGED

Clock calibration state changed

FDCAN_FLAG_CALIB_WATCHDOG_EVENT

Clock calibration watchdog event occurred

FDCAN format

FDCAN_CLASSIC_CAN

Frame transmitted/received in Classic CAN format

FDCAN_FD_CAN

Frame transmitted/received in FDCAN format

FDCAN Frame Format

FDCAN_FRAME_CLASSIC

Classic mode

FDCAN_FRAME_FD_NO_BRS

FD mode without BitRate Switching

FDCAN_FRAME_FD_BRS

FD mode with BitRate Switching

FDCAN Frame Type

FDCAN_DATA_FRAME

Data frame

FDCAN_REMOTE_FRAME

Remote frame

FDCAN High Priority Message Storage

FDCAN_HP_STORAGE_NO_FIFO

No FIFO selected

FDCAN_HP_STORAGE_MSG_LOST

FIFO message lost

FDCAN_HP_STORAGE_RXFIFO0

Message stored in FIFO 0

FDCAN_HP_STORAGE_RXFIFO1

Message stored in FIFO 1

FDCAN ID Type

FDCAN_STANDARD_ID

Standard ID element

FDCAN_EXTENDED_ID

Extended ID element

FDCAN interrupt line

FDCAN_INTERRUPT_LINE0

Interrupt Line 0

FDCAN_INTERRUPT_LINE1

Interrupt Line 1

FDCAN non-matching frames

FDCAN_ACCEPT_IN_RX_FIFO0

Accept in Rx FIFO 0

FDCAN_ACCEPT_IN_RX_FIFO1

Accept in Rx FIFO 1

FDCAN_REJECT

Reject

FDCAN Operating Mode

FDCAN_MODE_NORMAL

Normal mode

FDCAN_MODE_RESTRICTED_OPERATION

Restricted Operation mode

FDCAN_MODE_BUS_MONITORING

Bus Monitoring mode

FDCAN_MODE_INTERNAL_LOOPBACK

Internal LoopBack mode

FDCAN_MODE_EXTERNAL_LOOPBACK

External LoopBack mode

FDCAN Operation Mode

FDCAN_TT_COMMUNICATION_LEVEL1

Time triggered communication, level 1

FDCAN_TT_COMMUNICATION_LEVEL2

Time triggered communication, level 2

FDCAN_TT_COMMUNICATION_LEVEL0

Time triggered communication, level 0

FDCAN protocol error code

FDCAN_PROTOCOL_ERROR_NONE

No error occurred

FDCAN_PROTOCOL_ERROR_STUFF

Stuff error

FDCAN_PROTOCOL_ERROR_FORM

Form error

FDCAN_PROTOCOL_ERROR_ACK

Acknowledge error

FDCAN_PROTOCOL_ERROR_BIT1

Bit 1 (recessive) error

FDCAN_PROTOCOL_ERROR_BIT0

Bit 0 (dominant) error

FDCAN_PROTOCOL_ERROR_CRC

CRC check sum error

FDCAN_PROTOCOL_ERROR_NO_CHANGE

No change since last read

FDCAN reject remote frames

FDCAN_FILTER_REMOTE

Filter remote frames

FDCAN_REJECT_REMOTE

Reject all remote frames

FDCAN Rx FIFO 0 Interrupts

FDCAN_IT_RX_FIFO0_MESSAGE_LOST

Rx FIFO 0 message lost

FDCAN_IT_RX_FIFO0_FULL

Rx FIFO 0 full

FDCAN_IT_RX_FIFO0_WATERMARK

Rx FIFO 0 fill level reached watermark

FDCAN_IT_RX_FIFO0_NEW_MESSAGE

New message written to Rx FIFO 0

FDCAN Rx FIFO 1 Interrupts

FDCAN_IT_RX_FIFO1_MESSAGE_LOST

Rx FIFO 1 message lost

FDCAN_IT_RX_FIFO1_FULL

Rx FIFO 1 full

FDCAN_IT_RX_FIFO1_WATERMARK

Rx FIFO 1 fill level reached watermark

FDCAN_IT_RX_FIFO1_NEW_MESSAGE

New message written to Rx FIFO 1

FDCAN FIFO operation mode

FDCAN_RX_FIFO_BLOCKING

Rx FIFO blocking mode

FDCAN_RX_FIFO_OVERWRITE

Rx FIFO overwrite mode

FDCAN Rx Interrupts

FDCAN_IT_RX_HIGH_PRIORITY_MSG

High priority message received

FDCAN_IT_RX_BUFFER_NEW_MESSAGE

At least one received message stored into a Rx Buffer

FDCAN Rx Location

FDCAN_RX_FIFO0

Get received message from Rx FIFO 0

FDCAN_RX_FIFO1

Get received message from Rx FIFO 1

FDCAN_RX_BUFFER0

Get received message from Rx Buffer 0

FDCAN_RX_BUFFER1

Get received message from Rx Buffer 1

FDCAN_RX_BUFFER2

Get received message from Rx Buffer 2

FDCAN_RX_BUFFER3

Get received message from Rx Buffer 3

FDCAN_RX_BUFFER4

Get received message from Rx Buffer 4

FDCAN_RX_BUFFER5

Get received message from Rx Buffer 5

FDCAN_RX_BUFFER6

Get received message from Rx Buffer 6

FDCAN_RX_BUFFER7

Get received message from Rx Buffer 7

FDCAN_RX_BUFFER8

Get received message from Rx Buffer 8

FDCAN_RX_BUFFER9

Get received message from Rx Buffer 9

FDCAN_RX_BUFFER10

Get received message from Rx Buffer 10

FDCAN_RX_BUFFER11

Get received message from Rx Buffer 11

FDCAN_RX_BUFFER12

Get received message from Rx Buffer 12

FDCAN_RX_BUFFER13

Get received message from Rx Buffer 13

FDCAN_RX_BUFFER14

Get received message from Rx Buffer 14

FDCAN_RX_BUFFER15

Get received message from Rx Buffer 15

FDCAN_RX_BUFFER16

Get received message from Rx Buffer 16

FDCAN_RX_BUFFER17

Get received message from Rx Buffer 17

FDCAN_RX_BUFFER18

Get received message from Rx Buffer 18

FDCAN_RX_BUFFER19

Get received message from Rx Buffer 19

FDCAN_RX_BUFFER20

Get received message from Rx Buffer 20

FDCAN_RX_BUFFER21

Get received message from Rx Buffer 21

FDCAN_RX_BUFFER22

Get received message from Rx Buffer 22

FDCAN_RX_BUFFER23

Get received message from Rx Buffer 23

FDCAN_RX_BUFFER24

Get received message from Rx Buffer 24

FDCAN_RX_BUFFER25

Get received message from Rx Buffer 25

FDCAN_RX_BUFFER26

Get received message from Rx Buffer 26

FDCAN_RX_BUFFER27

Get received message from Rx Buffer 27

FDCAN_RX_BUFFER28

Get received message from Rx Buffer 28

FDCAN_RX_BUFFER29

Get received message from Rx Buffer 29

FDCAN_RX_BUFFER30

Get received message from Rx Buffer 30

FDCAN_RX_BUFFER31

Get received message from Rx Buffer 31

FDCAN_RX_BUFFER32

Get received message from Rx Buffer 32

FDCAN_RX_BUFFER33

Get received message from Rx Buffer 33

FDCAN_RX_BUFFER34

Get received message from Rx Buffer 34

FDCAN_RX_BUFFER35

Get received message from Rx Buffer 35

FDCAN_RX_BUFFER36

Get received message from Rx Buffer 36

FDCAN_RX_BUFFER37

Get received message from Rx Buffer 37

FDCAN_RX_BUFFER38

Get received message from Rx Buffer 38

FDCAN_RX_BUFFER39

Get received message from Rx Buffer 39

FDCAN_RX_BUFFER40

Get received message from Rx Buffer 40

FDCAN_RX_BUFFER41

Get received message from Rx Buffer 41

FDCAN_RX_BUFFER42

Get received message from Rx Buffer 42

FDCAN_RX_BUFFER43

Get received message from Rx Buffer 43

FDCAN_RX_BUFFER44

Get received message from Rx Buffer 44

FDCAN_RX_BUFFER45

Get received message from Rx Buffer 45

FDCAN_RX_BUFFER46

Get received message from Rx Buffer 46

FDCAN_RX_BUFFER47

Get received message from Rx Buffer 47

FDCAN_RX_BUFFER48

Get received message from Rx Buffer 48

FDCAN_RX_BUFFER49

Get received message from Rx Buffer 49

FDCAN_RX_BUFFER50

Get received message from Rx Buffer 50

FDCAN_RX_BUFFER51

Get received message from Rx Buffer 51

FDCAN_RX_BUFFER52

Get received message from Rx Buffer 52

FDCAN_RX_BUFFER53

Get received message from Rx Buffer 53

FDCAN_RX_BUFFER54

Get received message from Rx Buffer 54

FDCAN_RX_BUFFER55

Get received message from Rx Buffer 55

FDCAN_RX_BUFFER56

Get received message from Rx Buffer 56

FDCAN_RX_BUFFER57

Get received message from Rx Buffer 57

FDCAN_RX_BUFFER58

Get received message from Rx Buffer 58

FDCAN_RX_BUFFER59

Get received message from Rx Buffer 59

FDCAN_RX_BUFFER60

Get received message from Rx Buffer 60

FDCAN_RX_BUFFER61

Get received message from Rx Buffer 61

FDCAN_RX_BUFFER62

Get received message from Rx Buffer 62

FDCAN_RX_BUFFER63

Get received message from Rx Buffer 63

FDCAN timeout operation

FDCAN_TIMEOUT_CONTINUOUS

Timeout continuous operation

FDCAN_TIMEOUT_TX_EVENT_FIFO

Timeout controlled by Tx Event FIFO

FDCAN_TIMEOUT_RX_FIFO0

Timeout controlled by Rx FIFO 0

FDCAN_TIMEOUT_RX_FIFO1

Timeout controlled by Rx FIFO 1

FDCAN timestamp

FDCAN_TIMESTAMP_INTERNAL

Timestamp counter value incremented according to TCP

FDCAN_TIMESTAMP_EXTERNAL

External timestamp counter value used

FDCAN timestamp prescaler

FDCAN_TIMESTAMP_PRESC_1

Timestamp counter time unit in equal to CAN bit time

FDCAN_TIMESTAMP_PRESC_2

Timestamp counter time unit in equal to CAN bit time multiplied by 2

FDCAN_TIMESTAMP_PRESC_3

Timestamp counter time unit in equal to CAN bit time multiplied by 3

FDCAN_TIMESTAMP_PRESC_4

Timestamp counter time unit in equal to CAN bit time multiplied by 4

FDCAN_TIMESTAMP_PRESC_5

Timestamp counter time unit in equal to CAN bit time multiplied by 5

FDCAN_TIMESTAMP_PRESC_6

Timestamp counter time unit in equal to CAN bit time multiplied by 6

FDCAN_TIMESTAMP_PRESC_7

Timestamp counter time unit in equal to CAN bit time multiplied by 7

FDCAN_TIMESTAMP_PRESC_8

Timestamp counter time unit in equal to CAN bit time multiplied by 8

FDCAN_TIMESTAMP_PRESC_9

Timestamp counter time unit in equal to CAN bit time multiplied by 9

FDCAN_TIMESTAMP_PRESC_10

Timestamp counter time unit in equal to CAN bit time multiplied by 10

FDCAN_TIMESTAMP_PRESC_11

Timestamp counter time unit in equal to CAN bit time multiplied by 11

FDCAN_TIMESTAMP_PRESC_12

Timestamp counter time unit in equal to CAN bit time multiplied by 12

FDCAN_TIMESTAMP_PRESC_13

Timestamp counter time unit in equal to CAN bit time multiplied by 13

FDCAN_TIMESTAMP_PRESC_14

Timestamp counter time unit in equal to CAN bit time multiplied by 14

FDCAN_TIMESTAMP_PRESC_15

Timestamp counter time unit in equal to CAN bit time multiplied by 15

FDCAN_TIMESTAMP_PRESC_16

Timestamp counter time unit in equal to CAN bit time multiplied by 16

FDCAN TT Disturbing Error Interrupts

FDCAN_TT_IT_GLOBAL_TIME_ERROR

Global Time Error

FDCAN_TT_IT_TX_COUNT_UNDERFLOW

Tx Count Underflow

FDCAN_TT_IT_TX_COUNT_OVERFLOW

Tx Count Overflow

FDCAN_TT_IT_SCHEDULING_ERROR_1

Scheduling Error 1

FDCAN_TT_IT_SCHEDULING_ERROR_2

Scheduling Error 2

FDCAN_TT_IT_ERROR_LEVEL_CHANGE

Error Level Changed

FDCAN TT Fatal Error Interrupts

FDCAN_TT_IT_INIT_WATCH_TRIGGER

Initialization Watch Trigger

FDCAN_TT_IT_WATCH_TRIGGER

Watch Trigger

FDCAN_TT_IT_APPLICATION_WATCHDOG

Application Watchdog

FDCAN_TT_IT_CONFIG_ERROR

Configuration Error

FDCAN TT Flags

FDCAN_TT_FLAG_BASIC_CYCLE_START

Start of Basic Cycle

FDCAN_TT_FLAG_MATRIX_CYCLE_START

Start of Matrix Cycle

FDCAN_TT_FLAG_SYNC_MODE_CHANGE

Change of Synchronization Mode

FDCAN_TT_FLAG_START_OF_GAP

Start of Gap

FDCAN_TT_FLAG_REG_TIME_MARK

Register Time Mark Interrupt

FDCAN_TT_FLAG_TRIG_TIME_MARK

Trigger Time Mark Event Internal

FDCAN_TT_FLAG_STOP_WATCH

Stop Watch Event

FDCAN_TT_FLAG_GLOBAL_TIME_WRAP

Global Time Wrap

FDCAN_TT_FLAG_GLOBAL_TIME_DISC

Global Time Discontinuity

FDCAN_TT_FLAG_GLOBAL_TIME_ERROR

Global Time Error

FDCAN_TT_FLAG_TX_COUNT_UNDERFLOW

Tx Count Underflow

FDCAN_TT_FLAG_TX_COUNT_OVERFLOW

Tx Count Overflow

FDCAN_TT_FLAG_SCHEDULING_ERROR_1

Scheduling Error 1

FDCAN_TT_FLAG_SCHEDULING_ERROR_2

Scheduling Error 2

FDCAN_TT_FLAG_ERROR_LEVEL_CHANGE

Error Level Changed

FDCAN_TT_FLAG_INIT_WATCH_TRIGGER

Initialization Watch Trigger

FDCAN_TT_FLAG_WATCH_TRIGGER

Watch Trigger

FDCAN_TT_FLAG_APPLICATION_WATCHDOG

Application Watchdog

FDCAN_TT_FLAG_CONFIG_ERROR

Configuration Error

FDCAN TT Global Time Interrupts

FDCAN_TT_IT_GLOBAL_TIME_WRAP

Global Time Wrap

FDCAN_TT_IT_GLOBAL_TIME_DISC

Global Time Discontinuity

FDCAN TT Schedule Synchronization Interrupts

FDCAN_TT_IT_BASIC_CYCLE_START

Start of Basic Cycle

FDCAN_TT_IT_MATRIX_CYCLE_START

Start of Matrix Cycle

FDCAN_TT_IT_SYNC_MODE_CHANGE

Change of Synchronization Mode

FDCAN_TT_IT_START_OF_GAP

Start of Gap

FDCAN TT Stop Watch Interrupt

FDCAN_TT_IT_STOP_WATCH

Stop Watch Event

FDCAN TT Time Mark Interrupts

FDCAN_TT_IT_REG_TIME_MARK

Register Time Mark Interrupt

FDCAN_TT_IT_TRIG_TIME_MARK

Trigger Time Mark Event Internal

FDCAN TT Automatic Clock Calibration

FDCAN_TT_AUTO_CLK_CALIB_DISABLE

Automatic clock calibration in Level 0,2 disabled

FDCAN_TT_AUTO_CLK_CALIB_ENABLE

Automatic clock calibration in Level 0,2 enabled

FDCAN TT Basic Cycle Number

FDCAN_TT_CYCLES_PER_MATRIX_1

1 Basic Cycle per Matrix

FDCAN_TT_CYCLES_PER_MATRIX_2

2 Basic Cycles per Matrix

FDCAN_TT_CYCLES_PER_MATRIX_4

4 Basic Cycles per Matrix

FDCAN_TT_CYCLES_PER_MATRIX_8

8 Basic Cycles per Matrix

FDCAN_TT_CYCLES_PER_MATRIX_16

16 Basic Cycles per Matrix

FDCAN_TT_CYCLES_PER_MATRIX_32

32 Basic Cycles per Matrix

FDCAN_TT_CYCLES_PER_MATRIX_64

64 Basic Cycles per Matrix

FDCAN TT Cycle Start Sync

FDCAN_TT_NO_SYNC_PULSE

No sync pulse

FDCAN_TT_SYNC_BASIC_CYCLE_START

Sync pulse at start of basic cycle

FDCAN_TT_SYNC_MATRIX_START

Sync pulse at start of matrix

FDCAN TT Error Level

FDCAN_TT_NO_ERROR

Severity 0 - No Error

FDCAN_TT_WARNING

Severity 1 - Warning

FDCAN_TT_ERROR

Severity 2 - Error

FDCAN_TT_SEVERE_ERROR

Severity 3 - Severe Error

FDCAN TT Event Trigger Polarity

FDCAN_TT_EVT_TRIG_POL_RISING

Rising edge trigger

FDCAN_TT_EVT_TRIG_POL_FALLING

Falling edge trigger

FDCAN TT Event Trigger Selection

FDCAN_TT_EVENT_TRIGGER_0

TIM2 selected as event trigger

FDCAN_TT_EVENT_TRIGGER_1

TIM3 selected as event trigger

FDCAN_TT_EVENT_TRIGGER_2

ETH selected as event trigger

FDCAN_TT_EVENT_TRIGGER_3

HRTIM selected as event trigger

FDCAN TT External Clock Synchronization

FDCAN_TT_EXT_CLK_SYNC_DISABLE

External clock synchronization in Level 0,2 disabled

FDCAN_TT_EXT_CLK_SYNC_ENABLE

External clock synchronization in Level 0,2 enabled

FDCAN TT Global Time Filtering

FDCAN_TT_GLOB_TIME_FILT_DISABLE

Global time filtering in Level 0,2 disabled

FDCAN_TT_GLOB_TIME_FILT_ENABLE

Global time filtering in Level 0,2 enabled

FDCAN TT Master State

FDCAN_TT_MASTER_OFF

Master_Off, no master properties relevant

FDCAN_TT_TIME_SLAVE

Operating as Time Slave

FDCAN_TT_BACKUP_TIME_MASTER

Operating as Backup Time Master

FDCAN_TT_CURRENT_TIME_MASTER

Operating as current Time Master

FDCAN TT Operation

FDCAN_STRICTLY_TT_OPERATION

Strictly time-triggered operation

FDCAN_EXT_EVT_SYNC_TT_OPERATION

External event-synchronized time-triggered operation

FDCAN TT reference message payload

FDCAN_TT_REF_MESSAGE_NO_PAYLOAD

Reference message has no additional payload

FDCAN_TT_REF_MESSAGE_ADD_PAYLOAD

Additional payload is taken from Tx Buffer 0

FDCAN TT repeat factor

FDCAN_TT_REPEAT_EVERY_CYCLE

Trigger valid for all cycles

FDCAN_TT_REPEAT_EVERY_2ND_CYCLE

Trigger valid every 2dn cycle

FDCAN_TT_REPEAT_EVERY_4TH_CYCLE

Trigger valid every 4th cycle

FDCAN_TT_REPEAT_EVERY_8TH_CYCLE

Trigger valid every 8th cycle

FDCAN_TT_REPEAT_EVERY_16TH_CYCLE

Trigger valid every 16th cycle

FDCAN_TT_REPEAT_EVERY_32ND_CYCLE

Trigger valid every 32nd cycle

FDCAN_TT_REPEAT_EVERY_64TH_CYCLE

Trigger valid every 64th cycle

FDCAN TT Stop Watch Polarity

FDCAN_TT_STOP_WATCH_RISING

Selected stop watch source is captured at rising edge of fdcan1_swt

FDCAN_TT_STOP_WATCH_FALLING

Selected stop watch source is captured at falling edge of fdcan1_swt

FDCAN TT Stop Watch Source

FDCAN_TT_STOP_WATCH_DISABLED

Stop Watch disabled

FDCAN_TT_STOP_WATCH_CYCLE_TIME

Actual value of cycle time is copied to Capture Time register (TTCPT.SWV)

FDCAN_TT_STOP_WATCH_LOCAL_TIME

Actual value of local time is copied to Capture Time register (TTCPT.SWV)

FDCAN_TT_STOP_WATCH_GLOBAL_TIME

Actual value of global time is copied to Capture Time register (TTCPT.SWV)

FDCAN TT Stop Watch Trigger Selection

FDCAN_TT_STOP_WATCH_TRIGGER_0

TIM2 selected as stop watch trigger

FDCAN_TT_STOP_WATCH_TRIGGER_1

TIM3 selected as stop watch trigger

FDCAN_TT_STOP_WATCH_TRIGGER_2

ETH selected as stop watch trigger

FDCAN_TT_STOP_WATCH_TRIGGER_3

HRTIM selected as stop watch trigger

FDCAN TT Synchronization State

FDCAN_TT_OUT_OF_SYNC

Out of Synchronization

FDCAN_TT_SYNCHRONIZING

Synchronizing to communication

FDCAN_TT_IN_GAP

Schedule suspended by Gap

FDCAN_TT_IN_SCHEDULE

Synchronized to schedule

FDCAN TT time mark event external

FDCAN_TT_TM_NO_EXTERNAL_EVENT

No action

FDCAN_TT_TM_GEN_EXTERNAL_EVENT

External event (pulse) is generated when trigger becomes active

FDCAN TT time mark event internal

FDCAN_TT_TM_NO_INTERNAL_EVENT

No action

FDCAN_TT_TM_GEN_INTERNAL_EVENT

Internal event is generated when trigger becomes active

FDCAN TT Time Mark Source

FDCAN_TT_REG_TIMEMARK_DIABLED

No Register Time Mark Interrupt generated

FDCAN_TT_REG_TIMEMARK_CYC_TIME

Register Time Mark Interrupt if Time Mark = cycle time

FDCAN_TT_REG_TIMEMARK_LOC_TIME

Register Time Mark Interrupt if Time Mark = local time

FDCAN_TT_REG_TIMEMARK_GLO_TIME

Register Time Mark Interrupt if Time Mark = global time

FDCAN TT Time Master

FDCAN_TT_SLAVE

Time slave

FDCAN_TT_POTENTIAL_MASTER

Potential time master

FDCAN TT trigger type

FDCAN_TT_TX_REF_TRIGGER

Transmit reference message in strictly time-triggered operation

FDCAN_TT_TX_REF_TRIGGER_GAP

Transmit reference message in external event-synchronized time-triggered operation

FDCAN_TT_TX_TRIGGER_SINGLE

Start a single transmission in an exclusive time window

FDCAN_TT_TX_TRIGGER_CONTINUOUS

Start a continuous transmission in an exclusive time window

FDCAN_TT_TX_TRIGGER_ARBITRATION

Start a transmission in an arbitration time window

FDCAN_TT_TX_TRIGGER_MERGED

Start a merged arbitration window

FDCAN_TT_WATCH_TRIGGER

Check for missing reference messages in strictly time-triggered operation

FDCAN_TT_WATCH_TRIGGER_GAP

Check for missing reference messages in external event-synchronized time-triggered operation

FDCAN_TT_RX_TRIGGER

Check for the reception of periodic messages in exclusive time windows

FDCAN_TT_TIME_BASE_TRIGGER

Generate internal/external events depending on TmEventInt/TmEventExt configuration

FDCAN_TT_END_OF_LIST

Illegal trigger, to be assigned to the unused triggers after a FDCAN_TT_WATCH_TRIGGER or FDCAN_TT_WATCH_TRIGGER_GAP

FDCAN Tx FIFO/Queue Mode

FDCAN_TX_FIFO_OPERATION

FIFO mode

FDCAN_TX_QUEUE_OPERATION

Queue mode

FDCAN Tx Event FIFO Interrupts

FDCAN_IT_TX_EVT_FIFO_ELT_LOST

Tx Event FIFO element lost

FDCAN_IT_TX_EVT_FIFO_FULL

Tx Event FIFO full

FDCAN_IT_TX_EVT_FIFO_WATERMARK

Tx Event FIFO fill level reached watermark

FDCAN_IT_TX_EVT_FIFO_NEW_DATA

Tx Handler wrote Tx Event FIFO element

FDCAN Tx Interrupts

FDCAN_IT_TX_COMPLETE

Transmission Completed

FDCAN_IT_TX_ABORT_COMPLETE

Transmission Cancellation Finished

FDCAN_IT_TX_FIFO_EMPTY

Tx FIFO Empty

FDCAN Tx Location

FDCAN_TX_BUFFER0

Add message to Tx Buffer 0

FDCAN_TX_BUFFER1

Add message to Tx Buffer 1

FDCAN_TX_BUFFER2

Add message to Tx Buffer 2

FDCAN_TX_BUFFER3

Add message to Tx Buffer 3

FDCAN_TX_BUFFER4

Add message to Tx Buffer 4

FDCAN_TX_BUFFER5

Add message to Tx Buffer 5

FDCAN_TX_BUFFER6

Add message to Tx Buffer 6

FDCAN_TX_BUFFER7

Add message to Tx Buffer 7

FDCAN_TX_BUFFER8

Add message to Tx Buffer 8

FDCAN_TX_BUFFER9

Add message to Tx Buffer 9

FDCAN_TX_BUFFER10

Add message to Tx Buffer 10

FDCAN_TX_BUFFER11

Add message to Tx Buffer 11

FDCAN_TX_BUFFER12

Add message to Tx Buffer 12

FDCAN_TX_BUFFER13

Add message to Tx Buffer 13

FDCAN_TX_BUFFER14

Add message to Tx Buffer 14

FDCAN_TX_BUFFER15

Add message to Tx Buffer 15

FDCAN_TX_BUFFER16

Add message to Tx Buffer 16

FDCAN_TX_BUFFER17

Add message to Tx Buffer 17

FDCAN_TX_BUFFER18

Add message to Tx Buffer 18

FDCAN_TX_BUFFER19

Add message to Tx Buffer 19

FDCAN_TX_BUFFER20

Add message to Tx Buffer 20

FDCAN_TX_BUFFER21

Add message to Tx Buffer 21

FDCAN_TX_BUFFER22

Add message to Tx Buffer 22

FDCAN_TX_BUFFER23

Add message to Tx Buffer 23

FDCAN_TX_BUFFER24

Add message to Tx Buffer 24

FDCAN_TX_BUFFER25

Add message to Tx Buffer 25

FDCAN_TX_BUFFER26

Add message to Tx Buffer 26

FDCAN_TX_BUFFER27

Add message to Tx Buffer 27

FDCAN_TX_BUFFER28

Add message to Tx Buffer 28

FDCAN_TX_BUFFER29

Add message to Tx Buffer 29

FDCAN_TX_BUFFER30

Add message to Tx Buffer 30

FDCAN_TX_BUFFER31

Add message to Tx Buffer 31

26 HAL GFXMMU Generic Driver

26.1 GFXMMU Firmware driver registers structures

26.1.1 GFXMMU_BuffersTypeDef

GFXMMU_BuffersTypeDef is defined in the stm32n6xx_hal_gfxmmu.h

Data Fields

- ***uint32_t Buf0Address***
- ***uint32_t Buf1Address***
- ***uint32_t Buf2Address***
- ***uint32_t Buf3Address***

Field Documentation

- ***uint32_t GFXMMU_BuffersTypeDef::Buf0Address***
  Physical address of buffer 0.
- ***uint32_t GFXMMU_BuffersTypeDef::Buf1Address***
  Physical address of buffer 1.
- ***uint32_t GFXMMU_BuffersTypeDef::Buf2Address***
  Physical address of buffer 2.
- ***uint32_t GFXMMU_BuffersTypeDef::Buf3Address***
  Physical address of buffer 3.

26.1.2 GFXMMU_InterruptsTypeDef

GFXMMU_InterruptsTypeDef is defined in the stm32n6xx_hal_gfxmmu.h

Data Fields

- ***FunctionalState Activation***
- ***uint32_t UsedInterrupts***

Field Documentation

- ***FunctionalState GFXMMU_InterruptsTypeDef::Activation***
  Interrupts enable/disable.
- ***uint32_t GFXMMU_InterruptsTypeDef::UsedInterrupts***
  Interrupts used. This parameter can be a values combination of ***GFXMMU_Interrupts***.
  Note:
  – : Useful only when interrupts are enabled.

26.1.3 GFXMMU_InitTypeDef

GFXMMU_InitTypeDef is defined in the stm32n6xx_hal_gfxmmu.h

Data Fields

- ***uint32_t BlockSize***
- ***uint32_t DefaultValue***
- ***FunctionalState AddressTranslation***
- ***GFXMMU_BuffersTypeDef Buffers***
- ***GFXMMU_InterruptsTypeDef Interrupts***

Field Documentation

- ***uint32_t GFXMMU_InitTypeDef::BlockSize***
  Size of virtual memory block. This parameter can be a value of ***GFXMMU_BlockSize***.
- ***uint32_t GFXMMU_InitTypeDef::DefaultValue***
  Value returned when virtual memory location not physically mapped.
- ***FunctionalState GFXMMU_InitTypeDef::AddressTranslation***
  Address translation enable/disable.

- ***GFXMMU_BuffersTypeDef GFXMMU_InitTypeDef::Buffers***
  Physical buffers addresses.
- ***GFXMMU_InterruptsTypeDef GFXMMU_InitTypeDef::Interrupts***
  Interrupts parameters.

26.1.4 __GFXMMU_HandleTypeDef

__GFXMMU_HandleTypeDef is defined in the stm32n6xx_hal_gfxmmu.h

Data Fields

- ***GFXMMU_TypeDef * Instance***
- ***GFXMMU_InitTypeDef Init***
- ***HAL_GFXMMU_StateTypeDef State***
- ***__IO uint32_t ErrorCode***
- ***void(* ErrorCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***GFXMMU_TypeDef* __GFXMMU_HandleTypeDef::Instance***
  GFXMMU instance.
- ***GFXMMU_InitTypeDef __GFXMMU_HandleTypeDef::Init***
  GFXMMU init parameters.
- ***HAL_GFXMMU_StateTypeDef __GFXMMU_HandleTypeDef::State***
  GFXMMU state.
- ***__IO uint32_t __GFXMMU_HandleTypeDef::ErrorCode***
  GFXMMU error code.
- ***void(* __GFXMMU_HandleTypeDef::ErrorCallback)(struct __GFXMMU_HandleTypeDef *hgfxmmu)***
  GFXMMU error callback.
- ***void(* __GFXMMU_HandleTypeDef::MspInitCallback)(struct __GFXMMU_HandleTypeDef *hgfxmmu)***
  GFXMMU MSP init callback.
- ***void(* __GFXMMU_HandleTypeDef::MspDeInitCallback)(struct __GFXMMU_HandleTypeDef *hgfxmmu)***
  GFXMMU MSP de-init callback.

26.1.5 GFXMMU_LutLineTypeDef

GFXMMU_LutLineTypeDef is defined in the stm32n6xx_hal_gfxmmu.h

Data Fields

- ***uint32_t LineNumber***
- ***uint32_t LineStatus***
- ***uint32_t FirstVisibleBlock***
- ***uint32_t LastVisibleBlock***
- ***int32_t LineOffset***

Field Documentation

- ***uint32_t GFXMMU_LutLineTypeDef::LineNumber***
  LUT line number. This parameter must be a number between Min_Data = 0 and Max_Data = 1023.
- ***uint32_t GFXMMU_LutLineTypeDef::LineStatus***
  LUT line enable/disable. This parameter can be a value of ***GFXMMU_LutLineStatus***.
- ***uint32_t GFXMMU_LutLineTypeDef::FirstVisibleBlock***
  First visible block on this line. This parameter must be a number between Min_Data = 0 and Max_Data = 255.
- ***uint32_t GFXMMU_LutLineTypeDef::LastVisibleBlock***
  Last visible block on this line. This parameter must be a number between Min_Data = 0 and Max_Data = 255.

- ***int32_t GFXMMU_LutLineTypeDef::LineOffset***
  Offset of block 0 of the current line in physical buffer. This parameter must be a number between Min_Data = -255 and Max_Data = 261888.
  Note:
  – : Line offset has to be computed with the following formula: LineOffset = [(Blocks already used) - (1st visible block)].

26.1.6 GFXMMU_PackingTypeDef

GFXMMU_PackingTypeDef is defined in the stm32n6xx_hal_gfxmmu.h

Data Fields

- ***FunctionalState Buffer0Activation***
- ***uint32_t Buffer0Mode***
- ***FunctionalState Buffer1Activation***
- ***uint32_t Buffer1Mode***
- ***FunctionalState Buffer2Activation***
- ***uint32_t Buffer2Mode***
- ***FunctionalState Buffer3Activation***
- ***uint32_t Buffer3Mode***
- ***uint32_t DefaultAlpha***

Field Documentation

- ***FunctionalState GFXMMU_PackingTypeDef::Buffer0Activation***
  Packing on buffer 0 enable/disable.
- ***uint32_t GFXMMU_PackingTypeDef::Buffer0Mode***
  Buffer 0 packing mode. This parameter can be a value of ***GFXMMU_PackingModes***.
- ***FunctionalState GFXMMU_PackingTypeDef::Buffer1Activation***
  Packing on buffer 1 enable/disable.
- ***uint32_t GFXMMU_PackingTypeDef::Buffer1Mode***
  Buffer 1 packing mode. This parameter can be a value of ***GFXMMU_PackingModes***.
- ***FunctionalState GFXMMU_PackingTypeDef::Buffer2Activation***
  Packing on buffer 2 enable/disable.
- ***uint32_t GFXMMU_PackingTypeDef::Buffer2Mode***
  Buffer 2 packing mode. This parameter can be a value of ***GFXMMU_PackingModes***.
- ***FunctionalState GFXMMU_PackingTypeDef::Buffer3Activation***
  Packing on buffer 3 enable/disable.
- ***uint32_t GFXMMU_PackingTypeDef::Buffer3Mode***
  Buffer 3 packing mode. This parameter can be a value of ***GFXMMU_PackingModes***.
- ***uint32_t GFXMMU_PackingTypeDef::DefaultAlpha***
  Default alpha value. This parameter must be a number between Min_Data = 0 and Max_Data = 255.

26.2 GFXMMU Firmware driver API description

The following section lists the various functions of the GFXMMU library.

26.2.1 How to use this driver

Initialization

1. As prerequisite, fill in the HAL_GFXMMU_MspInit() :
   – Enable GFXMMU clock interface with __HAL_RCC_GFXMMU_CLK_ENABLE().
   – If interrupts are used, enable and configure GFXMMU global interrupt with HAL_NVIC_SetPriority() and HAL_NVIC_EnableIRQ().
2. Configure the number of blocks per line, default value, physical buffer addresses and interrupts using the HAL_GFXMMU_Init() function.

LUT configuration

1. Use HAL_GFXMMU_DisableLutLines() to deactivate all LUT lines (or a range of lines).
2. Use HAL_GFXMMU_ConfigLut() to copy LUT from flash to look up RAM.
3. Use HAL_GFXMMU_ConfigLutLine() to configure one line of LUT.

Modify physical buffer addresses

1. Use HAL_GFXMMU_ModifyBuffers() to modify physical buffer addresses.

Packing configuration

1. Use HAL_GFXMMU_ConfigPacking() to configure packing.

Error management

1. If interrupts are used, HAL_GFXMMU_IRQHandler() will be called when an error occurs. This function will call HAL_GFXMMU_ErrorCallback(). Use HAL_GFXMMU_GetError() to get the error code.

De-initialization

1. As prerequisite, fill in the HAL_GFXMMU_MspDeInit() :
   – Disable GFXMMU clock interface with __HAL_RCC_GFXMMU_CLK_ENABLE().
   – If interrupts has been used, disable GFXMMU global interrupt with HAL_NVIC_DisableIRQ().
2. De-initialize GFXMMU using the HAL_GFXMMU_DeInit() function.

Callback registration

The compilation define USE_HAL_GFXMMU_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use functions HAL_GFXMMU_RegisterCallback() to register a user callback.

Function HAL_GFXMMU_RegisterCallback() allows to register following callbacks:

- ErrorCallback : GFXMMU error.
- MspInitCallback : GFXMMU MspInit.
- MspDeInitCallback : GFXMMU MspDeInit.

This function takes as parameters the HAL peripheral handle, the callback ID and a pointer to the user callback function.

Use function HAL_GFXMMU_UnRegisterCallback() to reset a callback to the default weak (overridden) function. HAL_GFXMMU_UnRegisterCallback() takes as parameters the HAL peripheral handle, and the callback ID.

This function allows to reset following callbacks:

- ErrorCallback : GFXMMU error.
- MspInitCallback : GFXMMU MspInit.
- MspDeInitCallback : GFXMMU MspDeInit.

By default, after the HAL_GFXMMU_Init and if the state is HAL_GFXMMU_STATE_RESET all callbacks are reset to the corresponding legacy weak (overridden) functions: examples HAL_GFXMMU_ErrorCallback(). Exception done for MspInit and MspDeInit callbacks that are respectively reset to the legacy weak (overridden) functions in the HAL_GFXMMU_Init and HAL_GFXMMU_DeInit only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the HAL_GFXMMU_Init and HAL_GFXMMU_DeInit keep and use the user MspInit/MspDeInit callbacks (registered beforehand).

Callbacks can be registered/unregistered in READY state only. Exception done for MspInit/MspDeInit callbacks that can be registered/unregistered in READY or RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_GFXMMU_RegisterCallback before calling HAL_GFXMMU_DeInit or HAL_GFXMMU_Init function.

When the compilation define USE_HAL_GFXMMU_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak (overridden) callbacks are used.

26.2.2 Detailed description of functions

HAL_GFXMMU_Init

Function name

HAL_StatusTypeDef HAL_GFXMMU_Init (GFXMMU_HandleTypeDef * hgfxmmu)

Function description

HAL_GFXMMU_DeInit

Function name

HAL_StatusTypeDef HAL_GFXMMU_DeInit (GFXMMU_HandleTypeDef * hgfxmmu)

Function description

HAL_GFXMMU_MspInit

Function name

void HAL_GFXMMU_MspInit (GFXMMU_HandleTypeDef * hgfxmmu)

Function description

HAL_GFXMMU_MspDeInit

Function name

void HAL_GFXMMU_MspDeInit (GFXMMU_HandleTypeDef * hgfxmmu)

Function description

HAL_GFXMMU_RegisterCallback

Function name

HAL_StatusTypeDef HAL_GFXMMU_RegisterCallback (GFXMMU_HandleTypeDef * hgfxmmu, HAL_GFXMMU_CallbackIDTypeDef CallbackID, pGFXMMU_CallbackTypeDef pCallback)

Function description

HAL_GFXMMU_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_GFXMMU_UnRegisterCallback (GFXMMU_HandleTypeDef * hgfxmmu, HAL_GFXMMU_CallbackIDTypeDef CallbackID)

Function description

HAL_GFXMMU_ConfigLut

Function name

HAL_StatusTypeDef HAL_GFXMMU_ConfigLut (const GFXMMU_HandleTypeDef * hgfxmmu, uint32_t FirstLine, uint32_t LinesNumber, uint32_t Address)

Function description

HAL_GFXMMU_DisableLutLines

Function name

HAL_StatusTypeDef HAL_GFXMMU_DisableLutLines (const GFXMMU_HandleTypeDef * hgfxmmu, uint32_t FirstLine, uint32_t LinesNumber)

Function description

HAL_GFXMMU_ConfigLutLine

Function name

HAL_StatusTypeDef HAL_GFXMMU_ConfigLutLine (const GFXMMU_HandleTypeDef * hgfxmmu, const GFXMMU_LutLineTypeDef * lutLine)

Function description

HAL_GFXMMU_ModifyBuffers

Function name

HAL_StatusTypeDef HAL_GFXMMU_ModifyBuffers (GFXMMU_HandleTypeDef * hgfxmmu, const GFXMMU_BuffersTypeDef * Buffers)

Function description

HAL_GFXMMU_ConfigPacking

Function name

HAL_StatusTypeDef HAL_GFXMMU_ConfigPacking (GFXMMU_HandleTypeDef * hgfxmmu, const GFXMMU_PackingTypeDef * pPacking)

Function description

HAL_GFXMMU_IRQHandler

Function name

void HAL_GFXMMU_IRQHandler (GFXMMU_HandleTypeDef * hgfxmmu)

Function description

HAL_GFXMMU_ErrorCallback

Function name

void HAL_GFXMMU_ErrorCallback (GFXMMU_HandleTypeDef * hgfxmmu)

Function description

HAL_GFXMMU_GetState

Function name

HAL_GFXMMU_StateTypeDef HAL_GFXMMU_GetState (const GFXMMU_HandleTypeDef * hgfxmmu)

Function description

HAL_GFXMMU_GetError

Function name

uint32_t HAL_GFXMMU_GetError (GFXMMU_HandleTypeDef * hgfxmmu)

Function description

26.3 GFXMMU Firmware driver defines

The following section lists the various define and macros of the module.

26.3.1 GFXMMU

GFXMMU

GFXMMU block size

GFXMMU_12BYTE_BLOCKS

Blocks of 12-byte.

GFXMMU_16BYTE_BLOCKS

Blocks of 16-byte.

GFXMMU Error Code

GFXMMU_ERROR_NONE

No error.

GFXMMU_ERROR_BUFFER0_OVERFLOW

Buffer 0 overflow.

GFXMMU_ERROR_BUFFER1_OVERFLOW

Buffer 1 overflow.

GFXMMU_ERROR_BUFFER2_OVERFLOW

Buffer 2 overflow.

GFXMMU_ERROR_BUFFER3_OVERFLOW

Buffer 3 overflow.

GFXMMU_ERROR_BUS_MASTER

Bus master error.

GFXMMU_ERROR_INVALID_CALLBACK

Invalid callback error.

GFXMMU Exported Macros

__HAL_GFXMMU_RESET_HANDLE_STATE

Description:

- Reset GFXMMU handle state.

Parameters:

- __HANDLE__: GFXMMU handle.

Return value:

- None

GFXMMU interrupts

GFXMMU_BUS_MASTER_ERROR_IT

Bus master error interrupt.

GFXMMU_BUFFER0_OVERFLOW_IT

Buffer 0 overflow interrupt.

GFXMMU_BUFFER1_OVERFLOW_IT

Buffer 1 overflow interrupt.

GFXMMU_BUFFER2_OVERFLOW_IT

Buffer 2 overflow interrupt.

GFXMMU_BUFFER3_OVERFLOW_IT

Buffer 3 overflow interrupt.

GFXMMU LUT line status

GFXMMU_LUT_LINE_DISABLE

LUT line disabled.

GFXMMU_LUT_LINE_ENABLE

LUT line enabled.

GFXMMU packing modes

GFXMMU_PACKING_MSB_REMOVE

Remove MSB during packing operation.

GFXMMU_PACKING_LSB_REMOVE

Remove LSB during packing operation.

27 HAL GFXTIM Generic Driver

27.1 GFXTIM Firmware driver registers structures

27.1.1 GFXTIM_InitTypeDef

GFXTIM_InitTypeDef is defined in the stm32n6xx_hal_gfxtim.h

Data Fields

- ***uint32_t SynchroSrc***
- ***uint32_t TearingEffectSrc***
- ***uint32_t TearingEffectPolarity***
- ***uint32_t TearingEffectInterrupt***

Field Documentation

- ***uint32_t GFXTIM_InitTypeDef::SynchroSrc***
  Synchronization signals (HSYNC and VSYNC) sources. This parameter can be a value of ***GFXTIM_SynchroSrc***
- ***uint32_t GFXTIM_InitTypeDef::TearingEffectSrc***
  Tearing effect source This parameter can be a value of ***GFXTIM_TearingEffectSrc***
- ***uint32_t GFXTIM_InitTypeDef::TearingEffectPolarity***
  Tearing effect source This parameter can be a value of ***GFXTIM_TearingEffectPolarity***
- ***uint32_t GFXTIM_InitTypeDef::TearingEffectInterrupt***
  Tearing effect interrupt Enable or Disable This parameter can be a value of ***GFXTIM_Interrupt***

27.1.2 __GFXTIM_HandleTypeDef

__GFXTIM_HandleTypeDef is defined in the stm32n6xx_hal_gfxtim.h

Data Fields

- ***GFXTIM_TypeDef * Instance***
- ***__IO HAL_GFXTIM_StateTypeDef State***
- ***__IO uint32_t ErrorCode***
- ***GFXTIM_InitTypeDef Init***
- ***void(* HAL_GFXTIM_AbsoluteTimer_AFCC1Callback***
- ***void(* HAL_GFXTIM_AbsoluteTimer_AFCOFCallback***
- ***void(* HAL_GFXTIM_AbsoluteTimer_ALCC1Callback***
- ***void(* HAL_GFXTIM_AbsoluteTimer_ALCC2Callback***
- ***void(* HAL_GFXTIM_AbsoluteTimer_ALCOFCallback***
- ***void(* HAL_GFXTIM_RelativeTimer_RFC1RCallback***
- ***void(* HAL_GFXTIM_RelativeTimer_RFC2RCallback***
- ***void(* HAL_GFXTIM_TECallback***
- ***void(* HAL_GFXTIM_EventGenerator_EV1Callback***
- ***void(* HAL_GFXTIM_EventGenerator_EV2Callback***
- ***void(* HAL_GFXTIM_EventGenerator_EV3Callback***
- ***void(* HAL_GFXTIM_EventGenerator_EV4Callback***
- ***void(* HAL_GFXTIM_WatchdogTimer_AlarmCallback***
- ***void(* HAL_GFXTIM_WatchdogTimer_PreAlarmCallback***
- ***void(* ErrorCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***GFXTIM_TypeDef* __GFXTIM_HandleTypeDef::Instance***
  GFXTIM instance

- ***__IO HAL_GFXTIM_StateTypeDef __GFXTIM_HandleTypeDef::State***
  GFXTIM state
- ***__IO uint32_t __GFXTIM_HandleTypeDef::ErrorCode***
  GFXTIM error code
- ***GFXTIM_InitTypeDef __GFXTIM_HandleTypeDef::Init***
  GFXTIM initialization
- ***void(* __GFXTIM_HandleTypeDef::HAL_GFXTIM_AbsoluteTimer_AFCC1Callback)(struct __GFXTIM_HandleTypeDef *hgfxtim)***
  GFXTIM Absolute frame counter compare 1 callback
- ***void(* __GFXTIM_HandleTypeDef::HAL_GFXTIM_AbsoluteTimer_AFCOFCallback)(struct __GFXTIM_HandleTypeDef *hgfxtim)***
  GFXTIM Absolute frame counter overflow callback
- ***void(* __GFXTIM_HandleTypeDef::HAL_GFXTIM_AbsoluteTimer_ALCC1Callback)(struct __GFXTIM_HandleTypeDef *hgfxtim)***
  GFXTIM Absolute line counter compare 1 callback
- ***void(* __GFXTIM_HandleTypeDef::HAL_GFXTIM_AbsoluteTimer_ALCC2Callback)(struct __GFXTIM_HandleTypeDef *hgfxtim)***
  GFXTIM Absolute line counter compare 2 callback
- ***void(* __GFXTIM_HandleTypeDef::HAL_GFXTIM_AbsoluteTimer_ALCOFCallback)(struct __GFXTIM_HandleTypeDef *hgfxtim)***
  GFXTIM Absolute line counter overflow callback
- ***void(* __GFXTIM_HandleTypeDef::HAL_GFXTIM_RelativeTimer_RFC1RCallback)(struct __GFXTIM_HandleTypeDef *hgfxtim)***
  GFXTIM Relative frame counter 1 reload callback
- ***void(* __GFXTIM_HandleTypeDef::HAL_GFXTIM_RelativeTimer_RFC2RCallback)(struct __GFXTIM_HandleTypeDef *hgfxtim)***
  GFXTIM Relative frame counter 2 reload callback
- ***void(* __GFXTIM_HandleTypeDef::HAL_GFXTIM_TECallback)(struct __GFXTIM_HandleTypeDef *hgfxtim)***
  GFXTIM Tearing effect callback
- ***void(* __GFXTIM_HandleTypeDef::HAL_GFXTIM_EventGenerator_EV1Callback)(struct __GFXTIM_HandleTypeDef *hgfxtim)***
  GFXTIM Event events 1 callback
- ***void(* __GFXTIM_HandleTypeDef::HAL_GFXTIM_EventGenerator_EV2Callback)(struct __GFXTIM_HandleTypeDef *hgfxtim)***
  GFXTIM Event events 2 callback
- ***void(* __GFXTIM_HandleTypeDef::HAL_GFXTIM_EventGenerator_EV3Callback)(struct __GFXTIM_HandleTypeDef *hgfxtim)***
  GFXTIM Event events 3 callback
- ***void(* __GFXTIM_HandleTypeDef::HAL_GFXTIM_EventGenerator_EV4Callback)(struct __GFXTIM_HandleTypeDef *hgfxtim)***
  GFXTIM Event events 4 callback
- ***void(* __GFXTIM_HandleTypeDef::HAL_GFXTIM_WatchdogTimer_AlarmCallback)(struct __GFXTIM_HandleTypeDef *hgfxtim)***
  GFXTIM Watchdog alarm callback
- ***void(* __GFXTIM_HandleTypeDef::HAL_GFXTIM_WatchdogTimer_PreAlarmCallback)(struct __GFXTIM_HandleTypeDef *hgfxtim)***
  GFXTIM Watchdog pre alarm callback
- ***void(* __GFXTIM_HandleTypeDef::ErrorCallback)(struct __GFXTIM_HandleTypeDef *hgfxtim)***
  GFXTIM error callback
- ***void(* __GFXTIM_HandleTypeDef::MspInitCallback)(struct __GFXTIM_HandleTypeDef *hgfxtim)***
  GFXTIM MSP initialization user callback
- ***void(* __GFXTIM_HandleTypeDef::MspDeInitCallback)(struct __GFXTIM_HandleTypeDef *hgfxtim)***
  GFXTIM MSP de-initialization user callback

27.1.3 GFXTIM_ClockGeneratorConfigTypeDef

GFXTIM_ClockGeneratorConfigTypeDef is defined in the stm32n6xx_hal_gfxtim.h

Data Fields

- ***uint32_t LCCHwReloadSrc***
- ***uint32_t LCCReloadValue***
- ***uint32_t LCCClockSrc***
- ***uint32_t LineClockSrc***
- ***uint32_t FCCHwReloadSrc***
- ***uint32_t FCCReloadValue***
- ***uint32_t FCCClockSrc***
- ***uint32_t FrameClockSrc***
- ***uint32_t LineClockCalib***
- ***uint32_t FrameClockCalib***

Field Documentation

- ***uint32_t GFXTIM_ClockGeneratorConfigTypeDef::LCCHwReloadSrc***
  Line Clock Counter hardware reload source This parameter can be a value of ***GFXTIM_LCCHwReloadSrc***
- ***uint32_t GFXTIM_ClockGeneratorConfigTypeDef::LCCReloadValue***
  Line Clock Counter reload value (22 bits) This parameter must be a number between Min_Data = 1 and Max_Data = 4194303
- ***uint32_t GFXTIM_ClockGeneratorConfigTypeDef::LCCClockSrc***
  Line Clock Counter Clock Source This parameter can be a value of ***GFXTIM_LCCClockSrc***
- ***uint32_t GFXTIM_ClockGeneratorConfigTypeDef::LineClockSrc***
  Line Clock Source This parameter can be a value of ***GFXTIM_LineClockSrc***
- ***uint32_t GFXTIM_ClockGeneratorConfigTypeDef::FCCHwReloadSrc***
  Frame Clock Counter hardware reload source This parameter can be a value of ***GFXTIM_FCCHwReloadSrc***
- ***uint32_t GFXTIM_ClockGeneratorConfigTypeDef::FCCReloadValue***
  Frame Clock Counter reload value (12 bits) This parameter must be a number between Min_Data = 1 and Max_Data = 4095
- ***uint32_t GFXTIM_ClockGeneratorConfigTypeDef::FCCClockSrc***
  Frame Clock Counter Clock Source This parameter can be a value of ***GFXTIM_FCCClockSrc***
- ***uint32_t GFXTIM_ClockGeneratorConfigTypeDef::FrameClockSrc***
  Frame Clock Source This parameter can be a value of ***GFXTIM_FrameClockSrc***
- ***uint32_t GFXTIM_ClockGeneratorConfigTypeDef::LineClockCalib***
  Debug purpose This parameter can be a value of ***GFXTIM_LineClockCalib***
- ***uint32_t GFXTIM_ClockGeneratorConfigTypeDef::FrameClockCalib***
  Debug purpose This parameter can be a value of ***GFXTIM_FrameClockCalib***

27.1.4 GFXTIM_AbsoluteTimerConfigTypeDef

GFXTIM_AbsoluteTimerConfigTypeDef is defined in the stm32n6xx_hal_gfxtim.h

Data Fields

- ***uint32_t FrameCompare1Value***
- ***uint32_t FrameCounterValue***
- ***uint32_t FrameOverflowInterrupt***
- ***uint32_t FrameCompare1Interrupt***
- ***uint32_t LineCompare1Value***
- ***uint32_t LineCompare2Value***
- ***uint32_t LineCounterValue***
- ***uint32_t LineOverflowInterrupt***
- ***uint32_t LineCompare1Interrupt***
- ***uint32_t LineCompare2Interrupt***

Field Documentation

- ***uint32_t GFXTIM_AbsoluteTimerConfigTypeDef::FrameCompare1Value***
  Absolute Frame Compare 1 value (20 bits) This parameter must be a number between 1 and 1048575

- ***uint32_t GFXTIM_AbsoluteTimerConfigTypeDef::FrameCounterValue***
  Absolute Frame Counter initial value (20 bits) This parameter must be a number between 1 and 1048575
- ***uint32_t GFXTIM_AbsoluteTimerConfigTypeDef::FrameOverflowInterrupt***
  Absolute Frame Counter Overflow Interrupt Enable or Disable This parameter can be a value of ***GFXTIM_Interrupt***
- ***uint32_t GFXTIM_AbsoluteTimerConfigTypeDef::FrameCompare1Interrupt***
  Absolute Frame Compare 1 Interrupt Enable or Disable This parameter can be a value of ***GFXTIM_Interrupt***
- ***uint32_t GFXTIM_AbsoluteTimerConfigTypeDef::LineCompare1Value***
  Absolute Line Compare 1 value (12 bits) This parameter must be a number between 1 and 4095
- ***uint32_t GFXTIM_AbsoluteTimerConfigTypeDef::LineCompare2Value***
  Absolute Line Compare 2 value (12 bits) This parameter must be a number between 1 and 4095
- ***uint32_t GFXTIM_AbsoluteTimerConfigTypeDef::LineCounterValue***
  Absolute Line Counter value (12 bits) This parameter must be a number between 1 and 4095
- ***uint32_t GFXTIM_AbsoluteTimerConfigTypeDef::LineOverflowInterrupt***
  Absolute Line Counter Overflow Interrupt Enable or Disable This parameter can be a value of ***GFXTIM_Interrupt***
- ***uint32_t GFXTIM_AbsoluteTimerConfigTypeDef::LineCompare1Interrupt***
  Absolute Line Compare 1 Interrupt Enable or Disable This parameter can be a value of ***GFXTIM_Interrupt***
- ***uint32_t GFXTIM_AbsoluteTimerConfigTypeDef::LineCompare2Interrupt***
  Absolute Line Compare 2 Interrupt Enable or Disable This parameter can be a value of ***GFXTIM_Interrupt***

27.1.5 GFXTIM_RelativeTimerConfigTypeDef

GFXTIM_RelativeTimerConfigTypeDef is defined in the stm32n6xx_hal_gfxtim.h

Data Fields

- ***uint32_t AutoReloadValue***
- ***uint32_t CounterMode***
- ***uint32_t ReloadInterrupt***

Field Documentation

- ***uint32_t GFXTIM_RelativeTimerConfigTypeDef::AutoReloadValue***
  Auto reload value (12 bits) This parameter must be a number between 1 and 4095
- ***uint32_t GFXTIM_RelativeTimerConfigTypeDef::CounterMode***
  Counter Mode This parameter can be a value of GFXTIM_RelativeCounterMode
- ***uint32_t GFXTIM_RelativeTimerConfigTypeDef::ReloadInterrupt***
  Relative Frame Counter Reload Interrupt Enable or Disable This parameter can be a value of ***GFXTIM_Interrupt***

27.1.6 GFXTIM_EventGeneratorConfigTypeDef

GFXTIM_EventGeneratorConfigTypeDef is defined in the stm32n6xx_hal_gfxtim.h

Data Fields

- ***uint32_t LineEvent***
- ***uint32_t FrameEvent***
- ***uint32_t EventInterrupt***

Field Documentation

- ***uint32_t GFXTIM_EventGeneratorConfigTypeDef::LineEvent***
  Line event selection This parameter can be a value of GFXTIM_EventLine
- ***uint32_t GFXTIM_EventGeneratorConfigTypeDef::FrameEvent***
  Frmae event selection This parameter can be a value of GFXTIM_EventFrame
- ***uint32_t GFXTIM_EventGeneratorConfigTypeDef::EventInterrupt***
  Event interrupt Enable or Disable This parameter can be a value of ***GFXTIM_Interrupt***

27.1.7 GFXTIM_WatchdogConfigTypeDef

GFXTIM_WatchdogConfigTypeDef is defined in the stm32n6xx_hal_gfxtim.h

Data Fields

- ***uint32_t ClockSrc***
- ***uint32_t AutoReloadValue***
- ***uint32_t HwReloadConfig***
- ***uint32_t PreAlarmValue***
- ***uint32_t AlarmInterrupt***
- ***uint32_t PreAlarmInterrupt***

Field Documentation

- ***uint32_t GFXTIM_WatchdogConfigTypeDef::ClockSrc***
  Clock source This parameter can be a value of GFXTIM_WatchdogClockSrc
- ***uint32_t GFXTIM_WatchdogConfigTypeDef::AutoReloadValue***
  Reload value (16 bits) This parameter must be a number between 1 and 65535
- ***uint32_t GFXTIM_WatchdogConfigTypeDef::HwReloadConfig***
  Hardware reload configuration This parameter can be a value of GFXTIM_WatchdogHwReloadConfig
- ***uint32_t GFXTIM_WatchdogConfigTypeDef::PreAlarmValue***
  Pre-alarm value (16 bits) This parameter must be a number between 1 and 65535
- ***uint32_t GFXTIM_WatchdogConfigTypeDef::AlarmInterrupt***
  Interrupt Enable or Disable when watchdog counter reaches 0 This parameter can be a value of ***GFXTIM_Interrupt***
- ***uint32_t GFXTIM_WatchdogConfigTypeDef::PreAlarmInterrupt***
  Interrupt Enable or Disable when watchdog counter reaches pre-alarm value This parameter can be a value of ***GFXTIM_Interrupt***

27.2 GFXTIM Firmware driver API description

The following section lists the various functions of the GFXTIM library.

27.2.1 How to use this driver

Initialization and de-initialization

1. User has first to initialize GFXTIM.
2. As prerequisite, fill in the HAL_GFXTIM_MspInit() :
   – Enable GFXTIM with __HAL_RCC_GFXTIM_CLK_ENABLE
   – Enable the clocks for the used GPIOS with __HAL_RCC_GPIOx_CLK_ENABLE().
   – Configure these pins in alternate mode using HAL_GPIO_Init().
   – If interrupt mode is used, enable and configure GFXTIM interrupt with HAL_NVIC_SetPriority() and HAL_NVIC_EnableIRQ().
1. User can de-initialize GFXTIM with HAL_GFXTIM_DeInit() function.

generic functions

1. HAL_GFXTIM_IRQHandler will be called when GFXTIM interrupt occurs.
2. HAL_GFXTIM_ErrorCallback will be called when GFXTIM or ADF error occurs.
3. Use HAL_GFXTIM_GetState() to get the current GFXTIM or ADF instance state.
4. Use HAL_GFXTIM_GetErrorCode() to get the current GFXTIM or ADF instance error code. #if defined(GENERATOR_CALLBACKS_REGISTERING_AVAILABLE)

Callback registration

27.2.2 Detailed description of functions

HAL_GFXTIM_Init

Function name

HAL_StatusTypeDef HAL_GFXTIM_Init (GFXTIM_HandleTypeDef * hgfxtim)

Function description

HAL_GFXTIM_DeInit

Function name

HAL_StatusTypeDef HAL_GFXTIM_DeInit (GFXTIM_HandleTypeDef * hgfxtim)

Function description

HAL_GFXTIM_MspInit

Function name

void HAL_GFXTIM_MspInit (GFXTIM_HandleTypeDef * hgfxtim)

Function description

HAL_GFXTIM_MspDeInit

Function name

void HAL_GFXTIM_MspDeInit (GFXTIM_HandleTypeDef * hgfxtim)

Function description

HAL_GFXTIM_RegisterCallback

Function name

HAL_StatusTypeDef HAL_GFXTIM_RegisterCallback (GFXTIM_HandleTypeDef * hgfxtim, HAL_GFXTIM_CallbackIDTypeDef CallbackID, pGFXTIM_CallbackTypeDef pCallback)

Function description

HAL_GFXTIM_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_GFXTIM_UnRegisterCallback (GFXTIM_HandleTypeDef * hgfxtim, HAL_GFXTIM_CallbackIDTypeDef CallbackID)

Function description

HAL_GFXTIM_TECallback

Function name

void HAL_GFXTIM_TECallback (GFXTIM_HandleTypeDef * hgfxtim)

Function description

HAL_GFXTIM_ClockGenerator_Config

Function name

HAL_StatusTypeDef HAL_GFXTIM_ClockGenerator_Config (GFXTIM_HandleTypeDef * hgfxtim, const GFXTIM_ClockGeneratorConfigTypeDef * pClockGeneratorConfig)

Function description

HAL_GFXTIM_ClockGenerator_Reload

Function name

HAL_StatusTypeDef HAL_GFXTIM_ClockGenerator_Reload (GFXTIM_HandleTypeDef * hgfxtim, uint32_t ClockGeneratorCounter)

Function description

HAL_GFXTIM_AbsoluteTimer_Config

Function name

HAL_StatusTypeDef HAL_GFXTIM_AbsoluteTimer_Config (GFXTIM_HandleTypeDef * hgfxtim, const GFXTIM_AbsoluteTimerConfigTypeDef * pAbsoluteTimerConfig)

Function description

HAL_GFXTIM_AbsoluteTimer_Start

Function name

HAL_StatusTypeDef HAL_GFXTIM_AbsoluteTimer_Start (GFXTIM_HandleTypeDef * hgfxtim)

Function description

HAL_GFXTIM_AbsoluteTimer_Stop

Function name

HAL_StatusTypeDef HAL_GFXTIM_AbsoluteTimer_Stop (GFXTIM_HandleTypeDef * hgfxtim)

Function description

HAL_GFXTIM_AbsoluteTimer_Reset

Function name

HAL_StatusTypeDef HAL_GFXTIM_AbsoluteTimer_Reset (GFXTIM_HandleTypeDef * hgfxtim)

Function description

HAL_GFXTIM_AbsoluteTimer_GetCounter

Function name

HAL_StatusTypeDef HAL_GFXTIM_AbsoluteTimer_GetCounter (const GFXTIM_HandleTypeDef * hgfxtim, uint32_t AbsoluteTime, uint32_t * pValue)

Function description

HAL_GFXTIM_AbsoluteTimer_SetFrameCompare

Function name

HAL_StatusTypeDef HAL_GFXTIM_AbsoluteTimer_SetFrameCompare (GFXTIM_HandleTypeDef * hgfxtim, uint32_t Value)

Function description

HAL_GFXTIM_AbsoluteTimer_SetLineCompare

Function name

HAL_StatusTypeDef HAL_GFXTIM_AbsoluteTimer_SetLineCompare (GFXTIM_HandleTypeDef * hgfxtim, uint32_t AbsoluteLineComparator, uint32_t Value)

Function description

HAL_GFXTIM_AbsoluteTimer_AFCC1Callback

Function name

void HAL_GFXTIM_AbsoluteTimer_AFCC1Callback (GFXTIM_HandleTypeDef * hgfxtim)

Function description

HAL_GFXTIM_AbsoluteTimer_AFCOFCallback

Function name

void HAL_GFXTIM_AbsoluteTimer_AFCOFCallback (GFXTIM_HandleTypeDef * hgfxtim)

Function description

HAL_GFXTIM_AbsoluteTimer_ALCC1Callback

Function name

void HAL_GFXTIM_AbsoluteTimer_ALCC1Callback (GFXTIM_HandleTypeDef * hgfxtim)

Function description

HAL_GFXTIM_AbsoluteTimer_ALCC2Callback

Function name

void HAL_GFXTIM_AbsoluteTimer_ALCC2Callback (GFXTIM_HandleTypeDef * hgfxtim)

Function description

HAL_GFXTIM_AbsoluteTimer_ALCOFCallback

Function name

void HAL_GFXTIM_AbsoluteTimer_ALCOFCallback (GFXTIM_HandleTypeDef * hgfxtim)

Function description

HAL_GFXTIM_RelativeTimer_Config

Function name

HAL_StatusTypeDef HAL_GFXTIM_RelativeTimer_Config (GFXTIM_HandleTypeDef * hgfxtim, const GFXTIM_RelativeTimerConfigTypeDef * pRelativeTimerConfig, uint32_t RelativeTimer)

Function description

HAL_GFXTIM_RelativeTimer_Start

Function name

HAL_StatusTypeDef HAL_GFXTIM_RelativeTimer_Start (GFXTIM_HandleTypeDef * hgfxtim, uint32_t RelativeTimer)

Function description

HAL_GFXTIM_RelativeTimer_Stop

Function name

HAL_StatusTypeDef HAL_GFXTIM_RelativeTimer_Stop (GFXTIM_HandleTypeDef * hgfxtim, uint32_t RelativeTimer)

Function description

HAL_GFXTIM_RelativeTimer_ForceReload

Function name

HAL_StatusTypeDef HAL_GFXTIM_RelativeTimer_ForceReload (GFXTIM_HandleTypeDef * hgfxtim, uint32_t RelativeTimer)

Function description

HAL_GFXTIM_RelativeTimer_SetReload

Function name

HAL_StatusTypeDef HAL_GFXTIM_RelativeTimer_SetReload (GFXTIM_HandleTypeDef * hgfxtim, uint32_t RelativeTimer, uint32_t Value)

Function description

HAL_GFXTIM_RelativeTimer_GetCounter

Function name

HAL_StatusTypeDef HAL_GFXTIM_RelativeTimer_GetCounter (const GFXTIM_HandleTypeDef * hgfxtim, uint32_t RelativeTimer, uint32_t * pValue)

Function description

HAL_GFXTIM_RelativeTimer_RFC1RCallback

Function name

void HAL_GFXTIM_RelativeTimer_RFC1RCallback (GFXTIM_HandleTypeDef * hgfxtim)

Function description

HAL_GFXTIM_RelativeTimer_RFC2RCallback

Function name

void HAL_GFXTIM_RelativeTimer_RFC2RCallback (GFXTIM_HandleTypeDef * hgfxtim)

Function description

HAL_GFXTIM_EventGenerator_Config

Function name

HAL_StatusTypeDef HAL_GFXTIM_EventGenerator_Config (GFXTIM_HandleTypeDef * hgfxtim, uint32_t EventGenerator, const GFXTIM_EventGeneratorConfigTypeDef * pEventGeneratorConfig)

Function description

HAL_GFXTIM_EventGenerator_Enable

Function name

HAL_StatusTypeDef HAL_GFXTIM_EventGenerator_Enable (GFXTIM_HandleTypeDef * hgfxtim, uint32_t EventGenerator)

Function description

HAL_GFXTIM_EventGenerator_Disable

Function name

HAL_StatusTypeDef HAL_GFXTIM_EventGenerator_Disable (GFXTIM_HandleTypeDef * hgfxtim, uint32_t EventGenerator)

Function description

HAL_GFXTIM_EventGenerator_EV1Callback

Function name

void HAL_GFXTIM_EventGenerator_EV1Callback (GFXTIM_HandleTypeDef * hgfxtim)

Function description

HAL_GFXTIM_EventGenerator_EV2Callback

Function name

void HAL_GFXTIM_EventGenerator_EV2Callback (GFXTIM_HandleTypeDef * hgfxtim)

Function description

HAL_GFXTIM_EventGenerator_EV3Callback

Function name

void HAL_GFXTIM_EventGenerator_EV3Callback (GFXTIM_HandleTypeDef * hgfxtim)

Function description

HAL_GFXTIM_EventGenerator_EV4Callback

Function name

void HAL_GFXTIM_EventGenerator_EV4Callback (GFXTIM_HandleTypeDef * hgfxtim)

Function description

HAL_GFXTIM_WatchdogTimer_Config

Function name

HAL_StatusTypeDef HAL_GFXTIM_WatchdogTimer_Config (GFXTIM_HandleTypeDef * hgfxtim, const GFXTIM_WatchdogConfigTypeDef * pWatchdogConfig)

Function description

HAL_GFXTIM_WatchdogTimer_Enable

Function name

HAL_StatusTypeDef HAL_GFXTIM_WatchdogTimer_Enable (GFXTIM_HandleTypeDef * hgfxtim)

Function description

HAL_GFXTIM_WatchdogTimer_Disable

Function name

HAL_StatusTypeDef HAL_GFXTIM_WatchdogTimer_Disable (GFXTIM_HandleTypeDef * hgfxtim)

Function description

HAL_GFXTIM_WatchdogTimer_Refresh

Function name

HAL_StatusTypeDef HAL_GFXTIM_WatchdogTimer_Refresh (GFXTIM_HandleTypeDef * hgfxtim)

Function description

HAL_GFXTIM_WatchdogTimer_AlarmCallback

Function name

void HAL_GFXTIM_WatchdogTimer_AlarmCallback (GFXTIM_HandleTypeDef * hgfxtim)

Function description

HAL_GFXTIM_WatchdogTimer_PreAlarmCallback

Function name

void HAL_GFXTIM_WatchdogTimer_PreAlarmCallback (GFXTIM_HandleTypeDef * hgfxtim)

Function description

HAL_GFXTIM_IRQHandler

Function name

void HAL_GFXTIM_IRQHandler (GFXTIM_HandleTypeDef * hgfxtim)

Function description

HAL_GFXTIM_ErrorCallback

Function name

void HAL_GFXTIM_ErrorCallback (GFXTIM_HandleTypeDef * hgfxtim)

Function description

HAL_GFXTIM_GetError

Function name

uint32_t HAL_GFXTIM_GetError (const GFXTIM_HandleTypeDef * hgfxtim)

Function description

HAL_GFXTIM_GetState

Function name

HAL_GFXTIM_StateTypeDef HAL_GFXTIM_GetState (const GFXTIM_HandleTypeDef * hgfxtim)

Function description

27.3 GFXTIM Firmware driver defines

The following section lists the various define and macros of the module.

27.3.1 GFXTIM

GFXTIM

GFXTIM Absolute Line Comparator

GFXTIM_ABSOLUTE_LINE_COMPARE1

Absolute line compare 1

GFXTIM_ABSOLUTE_LINE_COMPARE2

Absolute line compare 2

GFXTIM Absolute Time

GFXTIM_ABSOLUTE_GLOBAL_TIME

Absolute global time (frame and line) counters ATR

GFXTIM_ABSOLUTE_FRAME_TIME

Absolute frame counter AFCR

GFXTIM_ABSOLUTE_LINE_TIME

Absolute line counter ALCR

GFXTIM Clock Generator Counter

GFXTIM_LINE_CLK_COUNTER

Line clock counter

GFXTIM_FRAME_CLK_COUNTER

Frame clock counter

GFXTIM Error Code

GFXTIM_ERROR_NONE

No error

GFXTIM_ERROR_STATE

State error

GFXTIM_ERROR_INVALID_CALLBACK

Invalid callback error occurs

GFXTIM Event generator Frame Source selection

GFXTIM_FRAME_EVENT_NONE

None

GFXTIM_FRAME_EVENT_AFC_OVERFLOW

Absolute frame counter overflow

GFXTIM_FRAME_EVENT_AFC_COMPARE

Absolute frame counter compare

GFXTIM_FRAME_EVENT_RFC1_RELOAD

Relative frame counter 1 reload

GFXTIM_FRAME_EVENT_RFC2_RELOAD

Relative frame counter 1 reload

GFXTIM Event Generator ID

GFXTIM_EVENT_GENERATOR_1

Event Generator 1

GFXTIM_EVENT_GENERATOR_2

Event Generator 2

GFXTIM_EVENT_GENERATOR_3

Event Generator 3

GFXTIM_EVENT_GENERATOR_4

Event Generator 4

GFXTIM Event generator Line source selection

GFXTIM_LINE_EVENT_NONE

None

GFXTIM_LINE_EVENT_ALC_OVERFLOW

Absolute line counter overflow

GFXTIM_LINE_EVENT_TE

Tearing effect

GFXTIM_LINE_EVENT_ALC1_COMPARE

Absolute line counter 1 compare

GFXTIM_LINE_EVENT_ALC2_COMPARE

Absolute line counter 2 compare

GFXTIM Exported Macros

__HAL_GFXTIM_RESET_HANDLE_STATE

Description:

- Reset GFXTIM handle state.

Parameters:

- __HANDLE__: GFXTIM handle.

Return value:

- None

__HAL_GFXTIM_GET_FLAG

Description:

- Check whether the specified GFXTIM flag is set or not.

Parameters:

- __HANDLE__: GFXTIM handle
- __FLAG__: GFXTIM flag This parameter can be one or a combination of the following values:
  - GFXTIM_FLAG_AFCO Absolute Frame Counter Overflow Flag
  - GFXTIM_FLAG_ALCO Absolute Line Counter Overflow Flag
  - GFXTIM_FLAG_TE Tearing Effect Flag
  - GFXTIM_FLAG_AFCC1 Absolute Frame Counter Compare 1 Flag
  - GFXTIM_FLAG_ALCC1 Absolute Line Counter Compare 1 Flag
  - GFXTIM_FLAG_ALCC2 Absolute Line Counter Compare 2 Flag
  - GFXTIM_FLAG_RFC1R Relative Frame Counter 1 Reload Flag
  - GFXTIM_FLAG_RFC2R Relative Frame Counter 2 Reload Flag
  - GFXTIM_FLAG_EV1 Event 1 Flag
  - GFXTIM_FLAG_EV2 Event 2 Flag
  - GFXTIM_FLAG_EV3 Event 3 Flag
  - GFXTIM_FLAG_EV4 Event 4 Flag
  - GFXTIM_FLAG_WDGA Watchdog Alarm Flag
  - GFXTIM_FLAG_WDGP Watchdog Pre-alarm Flag

Return value:

- State: of flag (TRUE or FALSE).

__HAL_GFXTIM_CLEAR_FLAG

Description:

- Clear the specified GFXTIM flag.

Parameters:

- __HANDLE__: GFXTIM handle
- __FLAG__: GFXTIM flag This parameter can be one or a combination of the following values:
  - GFXTIM_FLAG_AFCO Absolute Frame Counter Overflow Flag
  - GFXTIM_FLAG_ALCO Absolute Line Counter Overflow Flag
  - GFXTIM_FLAG_TE Tearing Effect Flag
  - GFXTIM_FLAG_AFCC1 Absolute Frame Counter Compare 1 Flag
  - GFXTIM_FLAG_ALCC1 Absolute Line Counter Compare 1 Flag
  - GFXTIM_FLAG_ALCC2 Absolute Line Counter Compare 2 Flag
  - GFXTIM_FLAG_RFC1R Relative Frame Counter 1 Reload Flag
  - GFXTIM_FLAG_RFC2R Relative Frame Counter 2 Reload Flag
  - GFXTIM_FLAG_EV1 Event 1 Flag
  - GFXTIM_FLAG_EV2 Event 2 Flag
  - GFXTIM_FLAG_EV3 Event 3 Flag
  - GFXTIM_FLAG_EV4 Event 4 Flag
  - GFXTIM_FLAG_WDGA Watchdog Alarm Flag
  - GFXTIM_FLAG_WDGP Watchdog Pre-alarm Flag

Return value:

- None

GFXTIM Frame CLock Counter Clock Source

GFXTIM_FCC_CLK_SRC_DISABLE

Disable

GFXTIM_FCC_CLK_SRC_LCC_UNDERFLOW

Line Clock Counter underflow

GFXTIM_FCC_CLK_SRC_HSYNC_RISING

HSYNC rising edge

GFXTIM_FCC_CLK_SRC_HSYNC_FALLING

HSYNC falling edge

GFXTIM_FCC_CLK_SRC_VSYNC_RISING

VSYNC rising edge

GFXTIM_FCC_CLK_SRC_VSYNC_FALLING

VSYNC falling edge

GFXTIM_FCC_CLK_SRC_TE_RISING

TE rising edge

GFXTIM_FCC_CLK_SRC_TE_FALLING

TE falling edge

GFXTIM Frame Clock Counter Hardware Reload source

GFXTIM_FCC_HW_RELOAD_SRC_NONE

No hardware reload

GFXTIM_FCC_HW_RELOAD_SRC_LCC_UNDERFLOW

Line Clock Counter underflow

GFXTIM_FCC_HW_RELOAD_SRC_HSYNC_RISING

HSYNC rising edge

GFXTIM_FCC_HW_RELOAD_SRC_HSYNC_FALLING

HSYNC falling edge

GFXTIM_FCC_HW_RELOAD_SRC_VSYNC_RISING

VSYNC rising edge

GFXTIM_FCC_HW_RELOAD_SRC_VSYNC_FALLING

VSYNC falling edge

GFXTIM_FCC_HW_RELOAD_SRC_TE_RISING

TE rising edge

GFXTIM_FCC_HW_RELOAD_SRC_TE_FALLING

TE falling edge

GFXTIM flags

GFXTIM_FLAG_AFCO

GFXTIM_FLAG_ALCO

GFXTIM_FLAG_TE

GFXTIM_FLAG_AFCC1

GFXTIM_FLAG_ALCC1

GFXTIM_FLAG_ALCC2

GFXTIM_FLAG_RFC1R

GFXTIM_FLAG_RFC2R

GFXTIM_FLAG_EV1

GFXTIM_FLAG_EV2

GFXTIM_FLAG_EV3

GFXTIM_FLAG_EV4

GFXTIM_FLAG_WDGA

GFXTIM_FLAG_WDGP

GFXTIM Frame Clock Calibration Output (for debug purpose)

GFXTIM_FRAME_CLK_CALIB_DISABLE

Frame clock output calibration Disable

GFXTIM_FRAME_CLK_CALIB_ENABLE

Frame clock output calibration Enable

GFXTIM GFXTIM Frame Clock Source

GFXTIM_FRAME_CLK_SRC_LCC_UNDERFLOW

Line Clock Counter underflow

GFXTIM_FRAME_CLK_SRC_FCC_UNDERFLOW

Frame Clock Counter underflow

GFXTIM_FRAME_CLK_SRC_HSYNC_RISING

HSYNC rising edge

GFXTIM_FRAME_CLK_SRC_HSYNC_FALLING

HSYNC falling edge

GFXTIM_FRAME_CLK_SRC_VSYNC_RISING

VSYNC rising edge

GFXTIM_FRAME_CLK_SRC_VSYNC_FALLING

VSYNC falling edge

GFXTIM_FRAME_CLK_SRC_TE_RISING

TE rising edge

GFXTIM_FRAME_CLK_SRC_TE_FALLING

TE falling edge

GFXTIM Interrupt

GFXTIM_IT_DISABLE

gfxtim_interrupt disable

GFXTIM_IT_ENABLE

gfxtim_interrupt enable

GFXTIM Line Clock Counter Clock Source

GFXTIM_LCC_CLK_SRC_DISABLE

Disable line clock counter

GFXTIM_LCC_CLK_SRC_SYSCLOCK

System clock as line clock counter source

GFXTIM Line Clock Counter Hardware Reload Source

GFXTIM_LCC_HW_RELOAD_SRC_NONE

No hardware reload

GFXTIM_LCC_HW_RELOAD_SRC_FCC_UNDERFLOW

FCC underflow

GFXTIM_LCC_HW_RELOAD_SRC_HSYNC_RISING

HSYNC rising

GFXTIM_LCC_HW_RELOAD_SRC_HSYNC_FALLING

HSYNC falling

GFXTIM_LCC_HW_RELOAD_SRC_VSYNC_RISING

VSYNC rising

GFXTIM_LCC_HW_RELOAD_SRC_VSYNC_FALLING

VSYNC falling

GFXTIM_LCC_HW_RELOAD_SRC_TE_RISING

TE rising

GFXTIM_LCC_HW_RELOAD_SRC_TE_FALLING

TE falling

GFXTIM Line Clock Calibration Output

GFXTIM_LINE_CLK_CALIB_DISABLE

Disable Line clock calibration

GFXTIM_LINE_CLK_CALIB_ENABLE

Enable Line clock calibration

GFXTIM Line Clock Source

GFXTIM_LINE_CLK_SRC_LCC_UNDERFLOW

Line Clock Counter underflow

GFXTIM_LINE_CLK_SRC_FCC_UNDERFLOW

Frame Clock Counter underflow

GFXTIM_LINE_CLK_SRC_HSYNC_RISING

HSYNC rising edge

GFXTIM_LINE_CLK_SRC_HSYNC_FALLING

HSYNC falling edge

GFXTIM_LINE_CLK_SRC_VSYNC_RISING

VSYNC rising edge

GFXTIM_LINE_CLK_SRC_VSYNC_FALLING

VSYNC falling edge

GFXTIM_LINE_CLK_SRC_TE_RISING

TE rising edge

GFXTIM_LINE_CLK_SRC_TE_FALLING

TE falling edge

GFXTIM Relative Frame Counter Mode

GFXTIM_MODE_ONE_SHOT

Relative Frame Counter One Shot Mode

GFXTIM_MODE_CONTINUOUS

Relative Frame Counter Continuous Mode

GFXTIM Relative Timer

GFXTIM_RELATIVE_TIMER1

Relative Timer 1

GFXTIM_RELATIVE_TIMER2

Relative Timer 2

GFXTIM Synchronization Source

GFXTIM_SYNC_SRC_HSYNC_VSYNC_0

gfxtim_hsync[0] and gfxtim_vsync[0] are used as synchronization source

GFXTIM_SYNC_SRC_HSYNC_VSYNC_1

gfxtim_hsync[1] and gfxtim_vsync[1] are used as synchronization source

GFXTIM_SYNC_SRC_HSYNC_VSYNC_2

gfxtim_hsync[2] and gfxtim_vsync[2] are used as synchronization source

GFXTIM_SYNC_SRC_HSYNC_VSYNC_3

gfxtim_hsync[3] and gfxtim_vsync[3] are used as synchronization source

GFXTIM Tearing Effect Polarity

GFXTIM_TE_RISING_EDGE

Tearing Effect active on rizing edge

GFXTIM_TE_FALLING_EDGE

Tearing Effect active on falling edge

GFXTIM Tearing Effect Source

GFXTIM_TE_SRC_GPIO

Input pad rising

GFXTIM_TE_SRC_ITE

gfxtim_ite rising

GFXTIM_TE_SRC_HSYNC

HSYNC (see SynchroSrc) rising

GFXTIM_TE_SRC_VSYNC

VSYNC (see SynchroSrc) rising

GFXTIM Watchdog clock source

GFXTIM_WATCHDOG_CLK_SRC_LINE_CLK

Line Clock

GFXTIM_WATCHDOG_CLK_SRC_FRAME_CLK

Frame Clock

GFXTIM_WATCHDOG_CLK_SRC_HSYNC_RISING

HSYNC rising edge

GFXTIM_WATCHDOG_CLK_SRC_HSYNC_FALLING

HSYNC falling edge

GFXTIM_WATCHDOG_CLK_SRC_VSYNC_RISING

VSYNC rising edge

GFXTIM_WATCHDOG_CLK_SRC_VSYNC_FALLING

VSYNC falling edge

GFXTIM_WATCHDOG_CLK_SRC_TE_RISING

Tearing Effect rising edge

GFXTIM_WATCHDOG_CLK_SRC_TE_FALLING

Tearing Effect falling edge

GFXTIM_WATCHDOG_CLK_SRC_EVENT_1

Event Generator 1 output

GFXTIM_WATCHDOG_CLK_SRC_EVENT_2

Event Generator 2 output

GFXTIM_WATCHDOG_CLK_SRC_EVENT_3

Event Generator 3 output

GFXTIM_WATCHDOG_CLK_SRC_EVENT_4

Event Generator 4 output

GFXTIM Watchdog hardware reload configuration

GFXTIM_WATCHDOG_HW_RELOAD_DISABLE

Watchdog hardware reload is disable

GFXTIM_WATCHDOG_HW_RELOAD_RISING_EDGE

Watchdog is reload on rising edge of gfxtim_wrld

GFXTIM_WATCHDOG_HW_RELOAD_FALLING_EDGE

Watchdog is reload on falling edge of gfxtim_wrld

28 HAL GPIO Generic Driver

28.1 GPIO Firmware driver registers structures

28.1.1 GPIO_InitTypeDef

GPIO_InitTypeDef is defined in the stm32n6xx_hal_gpio.h

Data Fields

- ***uint32_t Pin***
- ***uint32_t Mode***
- ***uint32_t Pull***
- ***uint32_t Speed***
- ***uint32_t Alternate***

Field Documentation

- ***uint32_t GPIO_InitTypeDef::Pin***
  Specifies the GPIO pins to be configured. This parameter can be any value of ***GPIO_pins***
- ***uint32_t GPIO_InitTypeDef::Mode***
  Specifies the operating mode for the selected pins. This parameter can be a value of ***GPIO_mode***
- ***uint32_t GPIO_InitTypeDef::Pull***
  Specifies the Pull-up or Pull-Down activation for the selected pins. This parameter can be a value of ***GPIO_pull***
- ***uint32_t GPIO_InitTypeDef::Speed***
  Specifies the speed for the selected pins. This parameter can be a value of ***GPIO_speed***
- ***uint32_t GPIO_InitTypeDef::Alternate***
  Peripheral to be connected to the selected pins This parameter can be a value of ***GPIOEx_Alternate_function_selection***

28.1.2 GPIO_DelayTypeDef

GPIO_DelayTypeDef is defined in the stm32n6xx_hal_gpio.h

Data Fields

- ***uint32_t Delay***
- ***uint32_t Path***

Field Documentation

- ***uint32_t GPIO_DelayTypeDef::Delay***
  Specifies the delay to apply on signal of a pin. This parameter can be a value of ***GPIO_delay***
- ***uint32_t GPIO_DelayTypeDef::Path***
  Specifies the path of the delay. This parameter can be a value of ***GPIO_path***

28.1.3 GPIO_RetimeTypeDef

GPIO_RetimeTypeDef is defined in the stm32n6xx_hal_gpio.h

Data Fields

- ***uint32_t Retime***
- ***uint32_t Edge***

Field Documentation

- ***uint32_t GPIO_RetimeTypeDef::Retime***
  Enable or disable the retime functionality on a pin This parameter can be any value of ***GPIO_retime***
- ***uint32_t GPIO_RetimeTypeDef::Edge***
  Specifies the clock edge for retime functionality. This parameter can be a value of ***GPIO_clock***

28.2 GPIO Firmware driver API description

The following section lists the various functions of the GPIO library.

28.2.1 GPIO Peripheral features

- Each port bit of the general-purpose I/O (GPIO) ports can be individually configured by software in several modes:
  - Input mode
  - Analog mode
  - Output mode
  - Alternate function mode
  - External interrupt/event lines
- During and just after reset, the alternate functions and external interrupt lines are not active and the I/O ports are configured in analog mode.
- All GPIO pins have weak internal pull-up and pull-down resistors, which can be activated or not.
- In Output or Alternate mode, each IO can be configured on open-drain or push-pull type and the IO speed can be selected depending on the VDD value.
- The microcontroller IO pins are connected to onboard peripherals/modules through a multiplexer that allows only one peripheral alternate function (AF) connected to an IO pin at a time. In this way, there can be no conflict between peripherals sharing the same IO pin.
- All ports have external interrupt/event capability. To use external interrupt lines, the port must be configured in input mode. All available GPIO pins are connected to the 16 external interrupt/event lines from EXTI0 to EXTI15.
- The external interrupt/event controller consists of up to 73 edge detectors (16 lines are connected to GPIO) for generating event/interrupt requests (each input line can be independently configured to select the type (interrupt or event) and the corresponding trigger event (rising or falling or both). Each line can also be masked independently.

28.2.2 How to use this driver

1. Enable the GPIO AHB clock using the following function: __HAL_RCC_GPIOx_CLK_ENABLE().
2. Configure the GPIO pin(s) using HAL_GPIO_Init().
   - Configure the IO mode using "Mode" member from GPIO_InitTypeDef structure
   - Activate Pull-up, Pull-down resistor using "Pull" member from GPIO_InitTypeDef structure.
   - In case of Output or alternate function mode selection: the speed is configured through "Speed" member from GPIO_InitTypeDef structure.
   - In alternate mode is selection, the alternate function connected to the IO is configured through "Alternate" member from GPIO_InitTypeDef structure.
   - Analog mode is required when a pin is to be used as ADC channel or DAC output.
   - In case of external interrupt/event selection the "Mode" member from GPIO_InitTypeDef structure select the type (interrupt or event) and the corresponding trigger event (rising or falling or both).
3. In case of external interrupt/event mode selection, configure NVIC IRQ priority mapped to the EXTI line using HAL_NVIC_SetPriority() and enable it using HAL_NVIC_EnableIRQ().
4. To get the level of a pin configured in input mode use HAL_GPIO_ReadPin().
5. To set/reset the level of a pin configured in output mode use HAL_GPIO_WritePin()/HAL_GPIO_TogglePin().
6. To set the level of several pins and reset level of several other pins in same cycle, use HAL_GPIO_WriteMultipleStatePin().
7. To lock pin configuration until next reset use HAL_GPIO_LockPin().
8. During and just after reset, the alternate functions are not active and the GPIO pins are configured in analog mode (except JTAG pins).
9. The LSE oscillator pins OSC32_IN and OSC32_OUT can be used as general purpose (PC14 and PC15, respectively) when the LSE oscillator is off. The LSE has priority over the GPIO function.
10. The HSE oscillator pins OSC_IN/OSC_OUT can be used as general purpose PH0 and PH1, respectively, when the HSE oscillator is off. The HSE has priority over the GPIO function.

28.2.3 IO attributes functions

This section contains the following APIs:

- ***HAL_GPIO_LockPinAttributes()***

- *HAL_GPIO_GetLockPinAttributes()*
- *HAL_GPIO_ConfigPinAttributes()*
- *HAL_GPIO_GetConfigPinAttributes()*

28.2.4 Initialization and de-initialization functions

This section contains the following APIs:

- *HAL_GPIO_Init()*
- *HAL_GPIO_DeInit()*

28.2.5 Detailed description of functions

HAL_GPIO_Init

Function name

void HAL_GPIO_Init (GPIO_TypeDef * GPIOx, const GPIO_InitTypeDef * GPIO_Init)

Function description

Initialize the GPIOx peripheral according to the specified parameters in the GPIO_Init.

Parameters

- **GPIOx:** where x can be (A..H and N..Q) to select the GPIO peripheral for STM32N6xx family
- **GPIO_Init:** pointer to a GPIO_InitTypeDef structure that contains the configuration information for the specified GPIO peripheral.

Return values

- **None:**

Notes

- If GPIOx peripheral pin is used in EXTI_MODE and the pin is secure/privilege, it is up to the application to insure that the corresponding EXTI line is set secure/privilege.

HAL_GPIO_DeInit

Function name

void HAL_GPIO_DeInit (GPIO_TypeDef * GPIOx, uint32_t GPIO_Pin)

Function description

De-initialize the GPIOx peripheral registers to their default reset values.

Parameters

- **GPIOx:** where x can be (A..H and N..Q) to select the GPIO peripheral for STM32N6xx family
- **GPIO_Pin:** specifies the port bit to be written. This parameter can be any combination of GPIO_Pin_x where x can be (0..15).

Return values

- **None:**

HAL_GPIO_ReadPin

Function name

GPIO_PinState HAL_GPIO_ReadPin (const GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)

Function description

Read the specified input port pin.

Parameters

- **GPIOx:** where x can be (A..H and N..Q) to select the GPIO peripheral for STM32N6xx family
- **GPIO_Pin:** specifies the port bit to read. This parameter can be any combination of GPIO_Pin_x where x can be (0..15).

Return values

- **The:** input port pin value.

HAL_GPIO_WritePin

Function name

void HAL_GPIO_WritePin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin, GPIO_PinState PinState)

Function description

Set or clear the selected data port bit.

Parameters

- **GPIOx:** where x can be (A..H and N..Q) to select the GPIO peripheral for STM32N6xx family
- **GPIO_Pin:** specifies the port bit to be written. This parameter can be any combination of GPIO_Pin_x where x can be (0..15).
- **PinState:** specifies the value to be written to the selected bit. This parameter can be one of the GPIO_PinState enum values:
  - GPIO_PIN_RESET: to clear the port pin
  - GPIO_PIN_SET: to set the port pin

Return values

- **None:**

Notes

- This function uses GPIOx_BSRR and GPIOx_BRR registers to allow atomic read/modify accesses. In this way, there is no risk of an IRQ occurring between the read and the modify access.

HAL_GPIO_TogglePin

Function name

void HAL_GPIO_TogglePin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)

Function description

Toggle the specified GPIO pin.

Parameters

- **GPIOx:** where x can be (A..H and N..Q) to select the GPIO peripheral for STM32N6xx family
- **GPIO_Pin:** specifies the pin to be toggled. This parameter can be any combination of GPIO_Pin_x where x can be (0..15).

Return values

- **None:**

HAL_GPIO_WriteMultipleStatePin

Function name

void HAL_GPIO_WriteMultipleStatePin (GPIO_TypeDef * GPIOx, uint16_t PinReset, uint16_t PinSet)

Function description

Set and clear several pins of a dedicated port in same cycle.

Parameters

- **GPIOx:** where x can be (A..H and N..Q) to select the GPIO peripheral for STM32N6xx family
- **PinReset:** specifies the port bits to be reset This parameter can be any combination of GPIO_Pin_x where x can be (0..15) or zero.
- **PinSet:** specifies the port bits to be set This parameter can be any combination of GPIO_Pin_x where x can be (0..15) or zero.

Return values

- **None:**

Notes

- Both PinReset and PinSet combinations shall not get any common bit, else assert would be triggered.
- At least one of the two parameters used to set or reset shall be different from zero.

HAL_GPIO_LockPin

Function name

HAL_StatusTypeDef HAL_GPIO_LockPin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)

Function description

Lock GPIO Pins configuration registers.

Parameters

- **GPIOx:** where x can be (A..H and N..Q) to select the GPIO peripheral for STM32N6xx family
- **GPIO_Pin:** specifies the port bits to be locked. This parameter can be any combination of GPIO_Pin_x where x can be (0..15).

Return values

- **HAL_OK:** if success, HAL_ERROR otherwise

Notes

- The locked registers are GPIOx_MODER, GPIOx_OTYPER, GPIOx_OSPEEDR, GPIOx_PUPDR, GPIOx_AFRL, GPIOx_AFRH, GPIOx_DELAYRL, GPIOx_DELAYRH, GPIOx_ADVCFGRL, GPIOx_ADVCFGRH.
- The configuration of the locked GPIO pins can no longer be modified until the next reset.

HAL_GPIO_SetRetime

Function name

void HAL_GPIO_SetRetime (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin, const GPIO_RetimeTypeDef * pRet_Init)

Function description

Configure GPIO retime on specified GPIO pin.

Parameters

- **GPIOx:** where x can be (A..H and N..Q)
- **GPIO_Pin:** specifies the port bit to be written. This parameter can be any combination of GPIO_Pin_x where x can be (0..15).
- **pRet_Init:** pointer to a GPIO_RetimeTypeDef structure that contains the retime configuration information for the specified GPIO peripheral.

Return values

- **None:**

HAL_GPIO_GetRetime

Function name

HAL_StatusTypeDef HAL_GPIO_GetRetime (const GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin, GPIO_RetimeTypeDef * pRet_Init)

Function description

Get GPIO retime configuration on specified GPIO pin.

Parameters

- **GPIOx:** where x can be (A..H and N..Q)
- **GPIO_Pin:** specifies the port bit to read. This parameter can be GPIO_PIN_x where x can be (0..15).
- **pRet_Init:** pointer to the return GPIO_RetimeTypeDef structure that contains the retime information

Return values

- **HAL:** Status

HAL_GPIO_SetDelay

Function name

void HAL_GPIO_SetDelay (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin, const GPIO_DelayTypeDef * pDly_Init)

Function description

Configure GPIO delay on specified GPIO pin.

Parameters

- **GPIOx:** where x can be (A..H and N..Q)
- **GPIO_Pin:** specifies the port bit to be written. This parameter can be any combination of GPIO_Pin_x where x can be (0..15).
- **pDly_Init:** pointer to a GPIO_DelayTypeDef structure that contains the delay configuration information for the specified GPIO peripheral.

Return values

- **None:**

HAL_GPIO_GetDelay

Function name

HAL_StatusTypeDef HAL_GPIO_GetDelay (const GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin, GPIO_DelayTypeDef * pDly_Init)

Function description

Get GPIO delay configuration on specified GPIO pin.

Parameters

- **GPIOx:** where x can be (A..H and N..Q)
- **GPIO_Pin:** specifies the port bit to read. This parameter can be GPIO_PIN_x where x can be (0..15).
- **pDly_Init:** pointer to the return GPIO_DelayTypeDef structure that contains the delay information

Return values

- **HAL:** Status

HAL_GPIO_EXTI_IRQHandler

Function name

void HAL_GPIO_EXTI_IRQHandler (uint16_t GPIO_Pin)

Function description

Handle EXTI interrupt request.

Parameters

- **GPIO_Pin:** Specifies the port pin connected to corresponding EXTI line.

Return values

- **None:**

HAL_GPIO_EXTI_Rising_Callback

Function name

void HAL_GPIO_EXTI_Rising_Callback (uint16_t GPIO_Pin)

Function description

EXTI line rising detection callback.

Parameters

- **GPIO_Pin:** Specifies the port pin connected to corresponding EXTI line.

Return values

- **None:**

HAL_GPIO_EXTI_Falling_Callback

Function name

void HAL_GPIO_EXTI_Falling_Callback (uint16_t GPIO_Pin)

Function description

EXTI line falling detection callback.

Parameters

- **GPIO_Pin:** Specifies the port pin connected to corresponding EXTI line.

Return values

- **None:**

HAL_GPIO_LockPinAttributes

Function name

void HAL_GPIO_LockPinAttributes (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)

Function description

Lock security and privilege configuration of several pins for a dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **GPIO_Pin:** specifies the port bit to be locked. This parameter can be any combination of GPIO_Pin_x where x can be (0..15).

Return values

- **None:**

Notes

- When the lock sequence has been applied on a port bit, the value of this port bit can no longer be modified until the next reset. Set sequence only available in secure and privilege.
- Each lock bit freezes a security configuration register (control and alternate function registers).

HAL_GPIO_GetLockPinAttributes

Function name

uint32_t HAL_GPIO_GetLockPinAttributes (const GPIO_TypeDef * GPIOx)

Function description

Get lock security and privilege configuration of several pins for a dedicated port.

Parameters

- **GPIOx:** GPIO Port

Return values

- **Lock:** status for all pins. Bitx are set to 1 for pinx locked, otherwise 0.

HAL_GPIO_ConfigPinAttributes

Function name

void HAL_GPIO_ConfigPinAttributes (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin, uint32_t PinAttributes)

Function description

Configure the GPIO pins attributes.

Parameters

- **GPIOx:** where x can be (A..H and N..Q) to select the GPIO peripheral for STM32N6xx family
- **GPIO_Pin:** : GPIO_Pin specifies the pin(s) to configure the secure/privilege attribute
- **PinAttributes:** PinAttributes can be one or a combination of the following values :
  - GPIO_PIN_PRIV Privileged-only access
  - GPIO_PIN_NPRIV Privileged/Non-privileged access
  - GPIO_PIN_SEC Secure-only access
  - GPIO_PIN_NSEC Secure/Non-secure access

Return values

- **None.:**

Notes

- Set a pin to secure is only available in secure and privilege Set a pin to privilege is only available in privilege

HAL_GPIO_GetConfigPinAttributes

Function name

HAL_StatusTypeDef HAL_GPIO_GetConfigPinAttributes (const GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin, uint32_t * pPinAttributes)

Function description

Get the GPIO pins attributes.

Parameters

- **GPIOx:** where x can be (A..H and N..Q) to select the GPIO peripheral for STM32N6xx family
- **GPIO_Pin:** specifies the port bit to read. This parameter can be GPIO_PIN_x where x can be (0..15).
- **pPinAttributes:** pPinAttributes pointer to return the pin attributes.

Return values

- **HAL:** Status.

28.3 GPIO Firmware driver defines

The following section lists the various define and macros of the module.

28.3.1 GPIO

GPIO

GPIO attributes

GPIO_PIN_SEC

Secure pin attribute

GPIO_PIN_NSEC

Non-secure pin attribute

GPIO_PIN_PRIV

Privileged pin attribute

GPIO_PIN_NPRIV

Non-privileged pin attribute

GPIO clock control

GPIO_CLOCK_RISING

Retime on rising edge

GPIO_CLOCK_FALLING

Retime on falling edge

GPIO_CLOCK_RISING_FALLING

Retime on both edge

GPIO delay

GPIO_DELAY_PS_0

GPIO_DELAY_PS_300

GPIO_DELAY_PS_500

GPIO_DELAY_PS_750

GPIO_DELAY_PS_1000

GPIO_DELAY_PS_1250

GPIO_DELAY_PS_1500

GPIO_DELAY_PS_1750

GPIO_DELAY_PS_2000

GPIO_DELAY_PS_2250

GPIO_DELAY_PS_2500

GPIO_DELAY_PS_2750

GPIO_DELAY_PS_3000

GPIO_DELAY_PS_3250

GPIO Exported Macros

__HAL_GPIO_EXTI_GET_RISING_IT

Description:

- Check whether the specified EXTI line is rising edge asserted or not.

Parameters:

- __EXTI_LINE__: specifies the EXTI line to check. This parameter can be GPIO_PIN_x where x can be(0..15)

Return value:

- The: new state of __EXTI_LINE__ (SET or RESET).

__HAL_GPIO_EXTI_CLEAR_RISING_IT

Description:

- Clear the EXTI line rising pending bits.

Parameters:

- __EXTI_LINE__: specifies the EXTI lines to clear. This parameter can be any combination of GPIO_PIN_x where x can be (0..15)

Return value:

- None

__HAL_GPIO_EXTI_GET_FALLING_IT

Description:

- Check whether the specified EXTI line is falling edge asserted or not.

Parameters:

- __EXTI_LINE__: specifies the EXTI line to check. This parameter can be GPIO_PIN_x where x can be(0..15)

Return value:

- The: new state of __EXTI_LINE__ (SET or RESET).

__HAL_GPIO_EXTI_CLEAR_FALLING_IT

Description:

- Clear the EXTI line falling pending bits.

Parameters:

- __EXTI_LINE__: specifies the EXTI lines to clear. This parameter can be any combination of GPIO_PIN_x where x can be (0..15)

Return value:

- None

__HAL_GPIO_EXTI_GET_IT

Description:

- Check whether the specified EXTI line is asserted or not.

Parameters:

- __EXTI_LINE__: specifies the EXTI line to check. This parameter can be GPIO_PIN_x where x can be(0..15)

Return value:

- The: new state of __EXTI_LINE__ (SET or RESET).

__HAL_GPIO_EXTI_CLEAR_IT

Description:

- Clear the EXTI's line pending bits.

Parameters:

- __EXTI_LINE__: specifies the EXTI lines to clear. This parameter can be any combination of GPIO_PIN_x where x can be (0..15)

Return value:

- None

__HAL_GPIO_EXTI_GENERATE_SWIT

Description:

- Generate a Software interrupt on selected EXTI line.

Parameters:

- __EXTI_LINE__: specifies the EXTI line to check. This parameter can be GPIO_PIN_x where x can be(0..15)

Return value:

- None

__HAL_GPIO_EXTI_GET_FLAG

Description:

- Check whether the specified EXTI line flag is set or not.

Parameters:

- __EXTI_LINE__: specifies the EXTI line flag to check. This parameter can be GPIO_PIN_x where x can be(0..15)

Return value:

- The: new state of __EXTI_LINE__ (SET or RESET).

__HAL_GPIO_EXTI_CLEAR_FLAG

Description:

- Clear the EXTI line pending flags.

Parameters:

- __EXTI_LINE__: specifies the EXTI lines flags to clear. This parameter can be any combination of GPIO_PIN_x where x can be (0..15)

Return value:

- None

GPIO mode

GPIO_MODE_INPUT

Input Floating Mode

GPIO_MODE_OUTPUT_PP

Output Push Pull Mode

GPIO_MODE_OUTPUT_OD

Output Open Drain Mode

GPIO_MODE_AF_PP

Alternate Function Push Pull Mode

GPIO_MODE_AF_OD

Alternate Function Open Drain Mode

GPIO_MODE_ANALOG

Analog Mode

GPIO_MODE_IT_RISING

External Interrupt Mode with Rising edge trigger detection

GPIO_MODE_IT_FALLING

External Interrupt Mode with Falling edge trigger detection

GPIO_MODE_IT_RISING_FALLING

External Interrupt Mode with Rising/Falling edge trigger detection

GPIO_MODE_EVT_RISING

External Event Mode with Rising edge trigger detection

GPIO_MODE_EVT_FALLING

External Event Mode with Falling edge trigger detection

GPIO_MODE_EVT_RISING_FALLING

External Event Mode with Rising/Falling edge trigger detection

GPIO path control

GPIO_PATH_OUT

Delay applied to output signal

GPIO_PATH_IN

Delay applied to input signal

GPIO pins

GPIO_PIN_0

GPIO_PIN_1

GPIO_PIN_2

GPIO_PIN_3

GPIO_PIN_4

GPIO_PIN_5

GPIO_PIN_6

GPIO_PIN_7

GPIO_PIN_8

GPIO_PIN_9

GPIO_PIN_10

GPIO_PIN_11

GPIO_PIN_12

GPIO_PIN_13

GPIO_PIN_14

GPIO_PIN_15

GPIO_PIN_ALL

GPIO_PIN_MASK

GPIO pull

GPIO_NOPULL

No Pull-up or Pull-down activation

GPIO_PULLUP

Pull-up activation

GPIO_PULLDOWN

Pull-down activation

GPIO retime control

GPIO_RETIME_OFF

Retime disable at pad level

GPIO_RETIME_ON

Retime enable at pad level

GPIO speed

GPIO_SPEED_FREQ_LOW

Low speed

GPIO_SPEED_FREQ_MEDIUM

Medium speed

GPIO_SPEED_FREQ_HIGH

High speed

GPIO_SPEED_FREQ_VERY_HIGH

Very high speed

29 HAL GPIO Extension Driver

29.1 GPIOEx Firmware driver defines

The following section lists the various define and macros of the module.

29.1.1 GPIOEx

GPIOEx

GPIOEx Alternate function selection

GPIO_AF0_PWR

PWR Alternate Function mapping

GPIO_AF0_MCO

MCO Alternate Function mapping

GPIO_AF0_RTC

RTC Alternate Function mapping

GPIO_AF0_SWJ

SWJ (SWD and JTAG) Alternate Function mapping

GPIO_AF0_TRACE

TRACE Alternate Function mapping

GPIO_AF0_SYS

BOOT1 Alternate Function mapping

GPIO_AF1_LPTIM1

LPTIM1 Alternate Function mapping

GPIO_AF1_LPTIM5

LPTIM5 Alternate Function mapping

GPIO_AF1_TIM1

TIM1 Alternate Function mapping

GPIO_AF1_TIM2

TIM2 Alternate Function mapping

GPIO_AF1_TIM16

TIM16 Alternate Function mapping

GPIO_AF1_TIM17

TIM17 Alternate Function mapping

GPIO_AF2_GFXTIM

GFXTIM Alternate Function mapping

GPIO_AF2_I3C2

I3C2 Alternate Function mapping

GPIO_AF2_SAI1

SAI1 Alternate Function mapping

GPIO_AF2_TIM2

TIM2 Alternate Function mapping

GPIO_AF2_TIM3

TIM3 Alternate Function mapping

GPIO_AF2_TIM4

TIM4 Alternate Function mapping

GPIO_AF2_TIM5

TIM5 Alternate Function mapping

GPIO_AF2_TIM8

TIM8 Alternate Function mapping

GPIO_AF2_TIM12

TIM12 Alternate Function mapping

GPIO_AF3_ADF1

ADF1 Alternate Function mapping

GPIO_AF3_LPTIM2

LPTIM2 Alternate Function mapping

GPIO_AF3_LPTIM3

LPTIM3 Alternate Function mapping

GPIO_AF3_LPTIM4

LPTIM4 Alternate Function mapping

GPIO_AF3_LPUART1

LPUART1 Alternate Function mapping

GPIO_AF3_TIM9

TIM9 Alternate Function mapping

GPIO_AF3_USART3

USART3 Alternate Function mapping

GPIO_AF4_I2C1

I2C1 Alternate Function mapping

GPIO_AF4_I2C2

I2C2 Alternate Function mapping

GPIO_AF4_I2C3

I2C3 Alternate Function mapping

GPIO_AF4_I2C4

I2C4 Alternate Function mapping

GPIO_AF4_I3C1

I3C1 Alternate Function mapping

GPIO_AF4_MDF1

MDF1 Alternate Function mapping

GPIO_AF4_TIM15

TIM15 Alternate Function mapping

GPIO_AF4_USART1

UASRT1 Alternate Function mapping

GPIO_AF5_I3C1

I3C1 Alternate Function mapping

GPIO_AF5_SPI1

SPI1 Alternate Function mapping

GPIO_AF5_SPI2

SPI2 Alternate Function mapping

GPIO_AF5_SPI3

SPI3 Alternate Function mapping

GPIO_AF5_SPI4

SPI4 Alternate Function mapping

GPIO_AF5_SPI5

SPI5 Alternate Function mapping

GPIO_AF5_SPI6

SPI6 Alternate Function mapping

GPIO_AF6_FDCAN1

FDCAN1 Alternate Function mapping

GPIO_AF6_FDCAN2

FDCAN2 Alternate Function mapping

GPIO_AF6_FDCAN3

FDCAN3 Alternate Function mapping

GPIO_AF6_SAI1

SAI1 Alternate Function mapping

GPIO_AF6_SPI3

SPI3 Alternate Function mapping

GPIO_AF6_SPI4

SPI4 Alternate Function mapping

GPIO_AF6_UCPD1

UCPD1 Alternate Function mapping

GPIO_AF6_USART10

USART10 Alternate Function mapping

GPIO_AF7_I3C1

I3C1 Alternate Function mapping

GPIO_AF7_SPI2

SPI2 Alternate Function mapping

GPIO_AF7_SPI3

SPI3 Alternate Function mapping

GPIO_AF7_SPI6

SPI6 Alternate Function mapping

GPIO_AF7_TIM15

TIM15 Alternate Function mapping

GPIO_AF7_UART9

UART9 Alternate Function mapping

GPIO_AF7_UCPD1

UCPD1 Alternate Function mapping

GPIO_AF7_USART1

USART1 Alternate Function mapping

GPIO_AF7_USART2

USART2 Alternate Function mapping

GPIO_AF7_USART3

USART3 Alternate Function mapping

GPIO_AF7_USART6

USART6 Alternate Function mapping

GPIO_AF8_GFXTIM

GFXTIM Alternate Function mapping

GPIO_AF8_SPDIFRX

SPDIFRX Alternate Function mapping

GPIO_AF8_SPI6

SPI6 Alternate Function mapping

GPIO_AF8_UART4

UART4 Alternate Function mapping

GPIO_AF8_UART7

UART7 Alternate Function mapping

GPIO_AF8_UART8

UART8 Alternate Function mapping

GPIO_AF8_USART2

USART2 Alternate Function mapping

GPIO_AF9_XSPIM_P1

XSPIM Manager Port 1 Alternate Function mapping

GPIO_AF9_XSPIM_P2

XSPIM Manager Port 2 Alternate Function mapping

GPIO_AF9_PSSI

PSSI Alternate Function mapping

GPIO_AF9_SPI5

SPI5 Alternate Function mapping

GPIO_AF9_TIM11

TIM11 Alternate Function mapping

GPIO_AF9_DCMIPP

DCMIPP Alternate Function mapping

GPIO_AF9_DCMI

DCMI Alternate Function mapping

GPIO_AF10_LCD

LCD Alternate Function mapping

GPIO_AF10_MDIOS

MDIOS Alternate Function mapping

GPIO_AF10_PSSI

PSSI Alternate Function mapping

GPIO_AF10_SAI2

SAI2 Alternate Function mapping

GPIO_AF10_SDMMC1

SDMMC1 Alternate Function mapping

GPIO_AF10_SPI1

SPI1 Alternate Function mapping

GPIO_AF10_TIM10

TIM10 Alternate Function mapping

GPIO_AF10_TIM13

TIM13 Alternate Function mapping

GPIO_AF10_UART7

UART7 Alternate Function mapping

GPIO_AF10_FMC

FMC Alternate Function mapping

GPIO_AF10_DCMIPP

DCMIPP Alternate Function mapping

GPIO_AF10_DCMI

DCMI Alternate Function mapping

GPIO_AF11_ETH1

ETH1 Alternate Function mapping

GPIO_AF11_MDIOS

MDIOS Alternate Function mapping

GPIO_AF11_SDMMC2

SDMMC2 Alternate Function mapping

GPIO_AF11_UART5

UART5 Alternate Function mapping

GPIO_AF11_TIM14

TIM14 Alternate Function mapping

GPIO_AF12_ETH1

ETH1 Alternate Function mapping

GPIO_AF12_FMC

FMC Alternate Function mapping

GPIO_AF12_GFXTIM

GFXTIM Alternate Function mapping

GPIO_AF12_LCD

LCD Alternate Function mapping

GPIO_AF12_SDMMC1

SDMMC1 Alternate Function mapping

GPIO_AF13_GFXTIM

GFXTIM Alternate Function mapping

GPIO_AF13_TIM1

TIM1 Alternate Function mapping

GPIO_AF14_FMC

FMC Alternate Function mapping

GPIO_AF14_LCD

LCD Alternate Function mapping

GPIO_AF15_EVENTOUT

EVENTOUT Alternate Function mapping

GPIO_AF15_HDP

HDP Alternate Function mapping

IS_GPIO_AF

GPIOEx Get Port Index

GPIO_GET_INDEX

30 HAL GPU2D Generic Driver

30.1 GPU2D Firmware driver registers structures

30.1.1 GPU2D_HandleTypeDef

GPU2D_HandleTypeDef is defined in the stm32n6xx_hal_gpu2d.h

Data Fields

- ***GPU2D_TypeDef Instance***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_GPU2D_StateTypeDef State***
- ***__IO uint32_t ErrorCode***

Field Documentation

- ***GPU2D_TypeDef GPU2D_HandleTypeDef::Instance***
  GPU2D register base address.
- ***HAL_LockTypeDef GPU2D_HandleTypeDef::Lock***
  GPU2D lock.
- ***__IO HAL_GPU2D_StateTypeDef GPU2D_HandleTypeDef::State***
  GPU2D transfer state.
- ***__IO uint32_t GPU2D_HandleTypeDef::ErrorCode***
  GPU2D error code.

30.2 GPU2D Firmware driver API description

The following section lists the various functions of the GPU2D library.

30.2.1 How to use this driver

1. Peripheral control is exclusively done by the accompanying middleware library.

Interrupt mode IO operation

1. Configure the GPU2D hardware to perform graphics operation using the Third Party MW Library APIs.
2. Submit command List to the hardware.
3. Wait indefinitely for the completion of submitted Command List by GPU2D hardware.
4. Use HAL_GPU2D_IRQHandler() called under GPU2D_IRQHandler() interrupt subroutine.
5. At the end of Command List execution HAL_GPU2D_IRQHandler() function is executed and user can add his own function by customization of function pointer
6. CommandListCpltCallback (member of GPU2D handle structure) to notify the upper level about the completion of Command List execution.
7. Callback HAL_GPU2D_CommandListCpltCallback is invoked when the GPU2D hardware executes the programmed command list (Command List execution completion).
   – This callback is called when the compilation defines USE_HAL_GPU2D_REGISTER_CALLBACKS is set to 0 or not defined.
   – This callback should be implemented in the application side. It should notify the upper level that the programmed command list is completed.
8. To control the GPU2D state, use the following function: HAL_GPU2D_GetState().
9. To read the GPU2D error code, use the following function: HAL_GPU2D_GetError().

GPU2D HAL driver macros list

Below the list of most used macros in GPU2D HAL driver :

- __HAL_GPU2D_RESET_HANDLE_STATE: Reset GPU2D handle state.
- __HAL_GPU2D_GET_FLAG: Get the GPU2D pending flags.
- __HAL_GPU2D_CLEAR_FLAG: Clear the GPU2D pending flags.

- __HAL_GPU2D_GET_IT_SOURCE: Check whether the specified GPU2D interrupt is enabled or not.

Callback registration

1. The compilation define USE_HAL_GPU2D_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use function HAL_GPU2D_RegisterCallback() to register a user callback.
2. Function HAL_GPU2D_RegisterCallback() allows to register following callbacks:
   – CommandListCpltCallback : callback for Command List completion.
   – MspInitCallback : GPU2D MspInit.
   – MspDeInitCallback : GPU2D MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.
3. Use function HAL_GPU2D_UnRegisterCallback() to reset a callback to the default weak (surcharged) function. HAL_GPU2D_UnRegisterCallback() takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks:
   – CommandListCpltCallback : callback for Command List completion.
   – MspInitCallback : GPU2D MspInit.
   – MspDeInitCallback : GPU2D MspDeInit.
4. Callbacks can be registered/unregistered in READY state only. Exception done for MspInit/MspDeInit callbacks that can be registered/unregistered in READY or RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_GPU2D_RegisterCallback before calling HAL_GPU2D_DeInit or HAL_GPU2D_Init function.
5. When The compilation define USE_HAL_GPU2D_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak (surcharged) callbacks are used.

Note: You can refer to the GPU2D HAL driver header file for more useful macros

30.2.2 Detailed description of functions

HAL_GPU2D_Init

Function name

HAL_StatusTypeDef HAL_GPU2D_Init (GPU2D_HandleTypeDef * hgpu2d)

Function description

HAL_GPU2D_DeInit

Function name

HAL_StatusTypeDef HAL_GPU2D_DeInit (GPU2D_HandleTypeDef * hgpu2d)

Function description

HAL_GPU2D_MspInit

Function name

void HAL_GPU2D_MspInit (GPU2D_HandleTypeDef * hgpu2d)

Function description

HAL_GPU2D_MspDeInit

Function name

void HAL_GPU2D_MspDeInit (GPU2D_HandleTypeDef * hgpu2d)

Function description

HAL_GPU2D_ReadRegister

Function name

uint32_t HAL_GPU2D_ReadRegister (const GPU2D_HandleTypeDef * hgpu2d, uint32_t offset)

Function description

HAL_GPU2D_WriteRegister

Function name

HAL_StatusTypeDef HAL_GPU2D_WriteRegister (GPU2D_HandleTypeDef * hgpu2d, uint32_t offset, uint32_t value)

Function description

HAL_GPU2D_IRQHandler

Function name

void HAL_GPU2D_IRQHandler (GPU2D_HandleTypeDef * hgpu2d)

Function description

HAL_GPU2D_ER_IRQHandler

Function name

void HAL_GPU2D_ER_IRQHandler (GPU2D_HandleTypeDef * hgpu2d)

Function description

HAL_GPU2D_CommandListCpltCallback

Function name

void HAL_GPU2D_CommandListCpltCallback (GPU2D_HandleTypeDef * hgpu2d, uint32_t CmdListID)

Function description

HAL_GPU2D_ErrorCallback

Function name

void HAL_GPU2D_ErrorCallback (GPU2D_HandleTypeDef * hgpu2d)

Function description

HAL_GPU2D_GetState

Function name

HAL_GPU2D_StateTypeDef HAL_GPU2D_GetState (GPU2D_HandleTypeDef const *const hgpu2d)

Function description

HAL_GPU2D_GetError

Function name

uint32_t HAL_GPU2D_GetError (GPU2D_HandleTypeDef const *const hgpu2d)

Function description

30.3 GPU2D Firmware driver defines

The following section lists the various define and macros of the module.

30.3.1 GPU2D

GPU2D

GPU2D Error Code definition

HAL_GPU2D_ERROR_NONE

No error

HAL_GPU2D_ERROR_TIMEOUT

Timeout error

GPU2D Exported Macros

__HAL_GPU2D_RESET_HANDLE_STATE

Description:

- Reset GPU2D handle state.

Parameters:

- __HANDLE__: specifies the GPU2D handle.

Return value:

- None

__HAL_GPU2D_GET_FLAG

Description:

- Get the GPU2D pending flags.

Parameters:

- __HANDLE__: GPU2D handle
- __FLAG__: flag to check. This parameter can be one of the following values:
  - GPU2D_FLAG_CLC: Command List Complete interrupt mask

Return value:

- The: state of FLAG.

__HAL_GPU2D_CLEAR_FLAG

Description:

- Clear the GPU2D pending flags.

Parameters:

- __HANDLE__: GPU2D handle
- __FLAG__: specifies the flag to clear. This parameter can be one of the following values:
  - GPU2D_FLAG_CLC: Command List Complete interrupt mask

Return value:

- None

__HAL_GPU2D_GET_IT_SOURCE

Description:

- Check whether the specified GPU2D interrupt source is enabled or not.

Parameters:

- __HANDLE__: GPU2D handle
- __INTERRUPT__: specifies the GPU2D interrupt source to check. This parameter can be one of the following values:
  - GPU2D_IT_CLC: Command List Complete interrupt mask

Return value:

- The: state of INTERRUPT source.

GPU2D Flag definition

GPU2D_FLAG_CLC

Command List Complete Interrupt Flag

GPU2D Interrupt configuration definition

GPU2D_IT_CLC

Command List Complete Interrupt

GPU2D Last Register Offset

GPU2D_OFFSET

Last GPU2D Register Offset

31 HAL HASH Generic Driver

31.1 HASH Firmware driver registers structures

31.1.1 HASH_ConfigTypeDef

HASH_ConfigTypeDef is defined in the stm32n6xx_hal_hash.h

Data Fields

- ***uint32_t DataType***
- ***uint32_t KeySize***
- ***uint8_t * pKey***
- ***uint32_t Algorithm***

Field Documentation

- ***uint32_t HASH_ConfigTypeDef::DataType***
  no swap (32-bit data), half word swap (16-bit data), byte swap (8-bit data) or bit swap (1-bit data). This parameter can be a value of ***HASH_Data_Type***.
- ***uint32_t HASH_ConfigTypeDef::KeySize***
  The key size is used only in HMAC operation.
- ***uint8_t* HASH_ConfigTypeDef::pKey***
  The key is used only in HMAC operation.
- ***uint32_t HASH_ConfigTypeDef::Algorithm***
  HASH algorithm MD5, SHA1 or SHA2. This parameter can be a value of ***HASH_Algorithm_Selection***

31.1.2 __HASH_HandleTypeDef

__HASH_HandleTypeDef is defined in the stm32n6xx_hal_hash.h

Data Fields

- ***HASH_TypeDef * Instance***
- ***HASH_ConfigTypeDef Init***
- ***uint8_t const * pHashInBuffPtr***
- ***uint8_t * pHashOutBuffPtr***
- ***__IO uint32_t HashInCount***
- ***uint32_t Size***
- ***uint8_t * pHashKeyBuffPtr***
- ***HAL_HASH_PhaseTypeDef Phase***
- ***DMA_HandleTypeDef * hdmain***
- ***HAL_LockTypeDef Lock***
- ***__IO uint32_t ErrorCode***
- ***__IO HAL_HASH_StateTypeDef State***
- ***__IO uint32_t Accumulation***
- ***void(* InCpltCallback***
- ***void(* DgstCpltCallback***
- ***void(* ErrorCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***HASH_TypeDef* __HASH_HandleTypeDef::Instance***
  HASH Register base address
- ***HASH_ConfigTypeDef __HASH_HandleTypeDef::Init***
  HASH required parameters
- ***uint8_t const* __HASH_HandleTypeDef::pHashInBuffPtr***
  Pointer to input buffer

- ***uint8_t* __HASH_HandleTypeDef::pHashOutBuffPtr***
  Pointer to output buffer (digest)
- ***__IO uint32_t __HASH_HandleTypeDef::HashInCount***
  Counter of inputted data
- ***uint32_t __HASH_HandleTypeDef::Size***
  Size of buffer to be processed in bytes
- ***uint8_t* __HASH_HandleTypeDef::pHashKeyBuffPtr***
  Pointer to key buffer (HMAC only)
- ***HAL_HASH_PhaseTypeDef __HASH_HandleTypeDef::Phase***
  HASH peripheral phase
- ***DMA_HandleTypeDef* __HASH_HandleTypeDef::hdmain***
  HASH In DMA Handle parameters
- ***HAL_LockTypeDef __HASH_HandleTypeDef::Lock***
  Locking object
- ***__IO uint32_t __HASH_HandleTypeDef::ErrorCode***
  HASH Error code
- ***__IO HAL_HASH_StateTypeDef __HASH_HandleTypeDef::State***
  HASH peripheral state
- ***__IO uint32_t __HASH_HandleTypeDef::Accumulation***
  HASH multi buffers accumulation flag
- ***void(* __HASH_HandleTypeDef::InCpltCallback)(struct __HASH_HandleTypeDef *hhash)***
  HASH input completion callback
- ***void(* __HASH_HandleTypeDef::DgstCpltCallback)(struct __HASH_HandleTypeDef *hhash)***
  HASH digest computation complete callback
- ***void(* __HASH_HandleTypeDef::ErrorCallback)(struct __HASH_HandleTypeDef *hhash)***
  HASH error callback
- ***void(* __HASH_HandleTypeDef::MspInitCallback)(struct __HASH_HandleTypeDef *hhash)***
  HASH Msp Init callback
- ***void(* __HASH_HandleTypeDef::MspDeInitCallback)(struct __HASH_HandleTypeDef *hhash)***
  HASH Msp DeInit callback

31.2 HASH Firmware driver API description

The following section lists the various functions of the HASH library.

31.2.1 How to use this driver

The HASH HAL driver can be used as follows:

1. Initialize the HASH low level resources by implementing the HAL_HASH_MspInit():
   a. Enable the HASH interface clock using __HAL_RCC_HASH_CLK_ENABLE()
   b. When resorting to interrupt-based APIs (e.g. HAL_HASH_Start_IT())
      - Configure the HASH interrupt priority using HAL_NVIC_SetPriority()
      - Enable the HASH IRQ handler using HAL_NVIC_EnableIRQ()
      - In HASH IRQ handler, call HAL_HASH_IRQHandler() API
   c. When resorting to DMA-based APIs (e.g. HAL_HASH_Start_DMA())
      - Enable the DMA interface clock
      - Configure and enable one DMA to manage data transfer from memory to peripheral (input DMA). Managing data transfer from peripheral to memory can be performed only using CPU.
      - Associate the initialized DMA handle to the HASH DMA handle using __HAL_LINKDMA()
      - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA: use HAL_NVIC_SetPriority() and HAL_NVIC_EnableIRQ()
2. Initialize the HASH HAL using HAL_HASH_Init(). This function:
   a. resorts to HAL_HASH_MspInit() for low-level initialization,
   b. configures the data type: no swap, half word swap, bit swap or byte swap,
   c. configures the Algorithm : MD5, SHA1 or SHA2

3. Three processing schemes are available:
   a. Polling mode: processing APIs are blocking functions i.e. they process the data and wait till the digest computation is finished, e.g. HAL_HASH_Start() for HASH or HAL_HMAC_Start() for HMAC
   b. Interrupt mode: processing APIs are not blocking functions i.e. they process the data under interrupt, e.g. HAL_HASH_Start_IT() for HASH or HAL_HMAC_Start_IT() for HMAC
   c. DMA mode: processing APIs are not blocking functions and the CPU is not used for data transfer i.e. the data transfer is ensured by DMA, e.g. HAL_HASH_Start_DMA() for HASH or HAL_HMAC_Start_DMA() for HMAC.
4. When the processing function is called after HAL_HASH_Init(), the HASH peripheral is initialized and processes the buffer fed in input. When the input data have all been fed to the Peripheral, the digest computation can start.
5. Multi-buffer processing HASH and HMAC are possible in polling, interrupt and DMA modes.
   a. In polling mode, API HAL_HASH_Accumulate()/HAL_HASH_HMAC_Accumulate() must be called for each input buffer, except for the last one. User must resort to HAL_HASH_AccumulateLast()/ HAL_HASH_HMAC_AccumulateLast() to enter the last one and retrieve as well the computed digest.
   b. In interrupt mode, API HAL_HASH_Accumulate_IT()/HAL_HASH_HMAC_Accumulate_IT() must be called for each input buffer, except for the last one. User must resort to HAL_HASH_AccumulateLast_IT()/ HAL_HASH_HMAC_AccumulateLast_IT() to enter the last one and retrieve as well the computed digest.
   c. In DMA mode, once initialization is done, MDMAT bit must be set through __HAL_HASH_SET_MDMAT() macro. From that point, each buffer can be fed to the Peripheral through HAL_HASH_Start_DMA() API for HASH and HAL_HASH_HMAC_Start_DMA() API for HMAC . Before entering the last buffer, reset the MDMAT bit with __HAL_HASH_RESET_MDMAT() macro then wrap-up the HASH processing in feeding the last input buffer through the same API HAL_HASH_Start_DMA()for HASH and HAL_HASH_HMAC_Start_DMA() API for HMAC and retrieve as well the computed digest.
6. To use this driver (version 2.0.0) with application developed with old driver (version 1.0.0) user have to:
   a. Add Algorithm as parameter like DataType or KeySize.
   b. Use new API HAL_HASH_Start() for HASH and HAL_HASH_HMAC_Start() for HMAC processing instead of old API like HAL_HASH_SHA1_Start and HAL_HMAC_SHA1_Start.

31.2.2 Detailed description of functions

HAL_HASH_Init

Function name

HAL_StatusTypeDef HAL_HASH_Init (HASH_HandleTypeDef * hhash)

Function description

HAL_HASH_DeInit

Function name

HAL_StatusTypeDef HAL_HASH_DeInit (HASH_HandleTypeDef * hhash)

Function description

HAL_HASH_MspInit

Function name

void HAL_HASH_MspInit (HASH_HandleTypeDef * hhash)

Function description

HAL_HASH_MspDeInit

Function name

void HAL_HASH_MspDeInit (HASH_HandleTypeDef * hhash)

Function description

HAL_HASH_GetConfig

Function name

HAL_StatusTypeDef HAL_HASH_GetConfig (HASH_HandleTypeDef * hhash, HASH_ConfigTypeDef * pConf)

Function description

HAL_HASH_SetConfig

Function name

HAL_StatusTypeDef HAL_HASH_SetConfig (HASH_HandleTypeDef * hhash, const HASH_ConfigTypeDef * pConf)

Function description

HAL_HASH_RegisterCallback

Function name

HAL_StatusTypeDef HAL_HASH_RegisterCallback (HASH_HandleTypeDef * hhash, HAL_HASH_CallbackIDTypeDef CallbackID, pHASH_CallbackTypeDef pCallback)

Function description

HAL_HASH_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_HASH_UnRegisterCallback (HASH_HandleTypeDef * hhash, HAL_HASH_CallbackIDTypeDef CallbackID)

Function description

HAL_HASH_ProcessSuspend

Function name

HAL_StatusTypeDef HAL_HASH_ProcessSuspend (HASH_HandleTypeDef * hhash)

Function description

HAL_HASH_Resume

Function name

void HAL_HASH_Resume (HASH_HandleTypeDef * hhash, uint8_t * pMemBuffer)

Function description

HAL_HASH_Suspend

Function name

void HAL_HASH_Suspend (HASH_HandleTypeDef * hhash, uint8_t * pMemBuffer)

Function description

HAL_HASH_Start

Function name

HAL_StatusTypeDef HAL_HASH_Start (HASH_HandleTypeDef * hhash, const uint8_t *const pInBuffer, uint32_t Size, uint8_t *const pOutBuffer, uint32_t Timeout)

Function description

HAL_HASH_Start_IT

Function name

HAL_StatusTypeDef HAL_HASH_Start_IT (HASH_HandleTypeDef * hhash, const uint8_t *const pInBuffer, uint32_t Size, uint8_t *const pOutBuffer)

Function description

HAL_HASH_Start_DMA

Function name

HAL_StatusTypeDef HAL_HASH_Start_DMA (HASH_HandleTypeDef * hhash, const uint8_t *const pInBuffer, uint32_t Size, uint8_t *const pOutBuffer)

Function description

HAL_HASH_Accumulate

Function name

HAL_StatusTypeDef HAL_HASH_Accumulate (HASH_HandleTypeDef * hhash, const uint8_t *const pInBuffer, uint32_t Size, uint32_t Timeout)

Function description

HAL_HASH_AccumulateLast

Function name

HAL_StatusTypeDef HAL_HASH_AccumulateLast (HASH_HandleTypeDef * hhash, const uint8_t *const pInBuffer, uint32_t Size, uint8_t *const pOutBuffer, uint32_t Timeout)

Function description

HAL_HASH_AccumulateLast_IT

Function name

HAL_StatusTypeDef HAL_HASH_AccumulateLast_IT (HASH_HandleTypeDef * hhash, const uint8_t *const pInBuffer, uint32_t Size, uint8_t *const pOutBuffer)

Function description

HAL_HASH_Accumulate_IT

Function name

HAL_StatusTypeDef HAL_HASH_Accumulate_IT (HASH_HandleTypeDef * hhash, const uint8_t *const pInBuffer, uint32_t Size)

Function description

HAL_HASH_HMAC_Start

Function name

HAL_StatusTypeDef HAL_HASH_HMAC_Start (HASH_HandleTypeDef * hhash, const uint8_t *const pInBuffer, uint32_t Size, uint8_t *const pOutBuffer, uint32_t Timeout)

Function description

HAL_HASH_HMAC_Start_DMA

Function name

HAL_StatusTypeDef HAL_HASH_HMAC_Start_DMA (HASH_HandleTypeDef * hhash, const uint8_t *const pInBuffer, uint32_t Size, uint8_t *const pOutBuffer)

Function description

HAL_HASH_HMAC_Start_IT

Function name

HAL_StatusTypeDef HAL_HASH_HMAC_Start_IT (HASH_HandleTypeDef * hhash, const uint8_t *const pInBuffer, uint32_t Size, uint8_t *const pOutBuffer)

Function description

HAL_HASH_HMAC_Accumulate

Function name

HAL_StatusTypeDef HAL_HASH_HMAC_Accumulate (HASH_HandleTypeDef * hhash, const uint8_t *const pInBuffer, uint32_t Size, uint32_t Timeout)

Function description

HAL_HASH_HMAC_AccumulateLast

Function name

HAL_StatusTypeDef HAL_HASH_HMAC_AccumulateLast (HASH_HandleTypeDef * hhash, const uint8_t *const pInBuffer, uint32_t Size, uint8_t *const pOutBuffer, uint32_t Timeout)

Function description

HAL_HASH_HMAC_Accumulate_IT

Function name

HAL_StatusTypeDef HAL_HASH_HMAC_Accumulate_IT (HASH_HandleTypeDef * hhash, const uint8_t *const pInBuffer, uint32_t Size)

Function description

HAL_HASH_HMAC_AccumulateLast_IT

Function name

HAL_StatusTypeDef HAL_HASH_HMAC_AccumulateLast_IT (HASH_HandleTypeDef * hhash, const uint8_t *const pInBuffer, uint32_t Size, uint8_t *const pOutBuffer)

Function description

HAL_HASH_IRQHandler

Function name

void HAL_HASH_IRQHandler (HASH_HandleTypeDef * hhash)

Function description

HAL_HASH_InCpltCallback

Function name

void HAL_HASH_InCpltCallback (HASH_HandleTypeDef * hhash)

Function description

HAL_HASH_DgstCpltCallback

Function name

void HAL_HASH_DgstCpltCallback (HASH_HandleTypeDef * hhash)

Function description

HAL_HASH_ErrorCallback

Function name

void HAL_HASH_ErrorCallback (HASH_HandleTypeDef * hhash)

Function description

HAL_HASH_GetState

Function name

HAL_HASH_StateTypeDef HAL_HASH_GetState (const HASH_HandleTypeDef * hhash)

Function description

HAL_HASH_GetError

Function name

uint32_t HAL_HASH_GetError (const HASH_HandleTypeDef * hhash)

Function description

31.3 HASH Firmware driver defines

The following section lists the various define and macros of the module.

31.3.1 HASH

HASH

HASH algorithm selection

HASH_ALGOSELECTION_SHA1

HASH function is SHA1

HASH_ALGOSELECTION_SHA224

HASH function is SHA224

HASH_ALGOSELECTION_SHA256

HASH function is SHA256

HASH_ALGOSELECTION_SHA384

HASH function is SHA384

HASH_ALGOSELECTION_SHA512_224

HASH function is SHA512_224

HASH_ALGOSELECTION_SHA512_256

HASH function is SHA512_256

HASH_ALGOSELECTION_SHA512

HASH function is SHA512

HASH Data Type

HASH_NO_SWAP

32-bit data. No swapping

HASH_HALFWORD_SWAP

16-bit data. Each half word is swapped

HASH_BYTE_SWAP

8-bit data. All bytes are swapped

HASH_BIT_SWAP

1-bit data. In the word all bits are swapped

HASH Error Definition

HAL_HASH_ERROR_NONE

No error

HAL_HASH_ERROR_BUSY

Busy flag error

HAL_HASH_ERROR_DMA

DMA-based process error

HAL_HASH_ERROR_TIMEOUT

Timeout error

HAL_HASH_ERROR_INVALID_CALLBACK

Invalid Callback error

HASH Exported Macros

__HAL_HASH_GET_FLAG

Description:

- Check whether or not the specified HASH flag is set.

Parameters:

- __HANDLE__: specifies the HASH handle.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - HASH_FLAG_DINIS A new block can be entered into the input buffer.
  - HASH_FLAG_DCIS Digest calculation complete.
  - HASH_FLAG_DMAS DMA interface is enabled (DMAE=1) or a transfer is ongoing.
  - HASH_FLAG_BUSY The hash core is Busy : processing a block of data.
  - HASH_FLAG_DINNE DIN not empty : the input buffer contains at least one word of data.

Return value:

- The: new state of __FLAG__ (SET or RESET).

__HAL_HASH_CLEAR_FLAG

Description:

- Clear the specified HASH flag.

Parameters:

- __HANDLE__: specifies the HASH handle.
- __FLAG__: specifies the flag to clear. This parameter can be one of the following values:
  - HASH_FLAG_DINIS A new block can be entered into the input buffer.
  - HASH_FLAG_DCIS Digest calculation complete

Return value:

- None

__HAL_HASH_GET_IT_SOURCE

Description:

- Check whether the specified HASH interrupt source is enabled or not.

Parameters:

- __HANDLE__: specifies the HASH handle.
- __INTERRUPT__: HASH interrupt source to check This parameter can be one of the following values :
  - HASH_IT_DINI A new block can be entered into the input buffer (DIN)
  - HASH_IT_DCI Digest calculation complete

Return value:

- State: of interruption (TRUE or FALSE).

__HAL_HASH_ENABLE_IT

Description:

- Enable the specified HASH interrupt.

Parameters:

- __HANDLE__: specifies the HASH handle.
- __INTERRUPT__: specifies the HASH interrupt source to enable. This parameter can be one of the following values:
  - HASH_IT_DINI A new block can be entered into the input buffer (DIN)
  - HASH_IT_DCI Digest calculation complete

Return value:

- None

__HAL_HASH_DISABLE_IT

Description:

- Disable the specified HASH interrupt.

Parameters:

- __HANDLE__: specifies the HASH handle.
- __INTERRUPT__: specifies the HASH interrupt source to disable. This parameter can be one of the following values:
  - HASH_IT_DINI A new block can be entered into the input buffer (DIN)
  - HASH_IT_DCI Digest calculation complete

Return value:

- None

__HAL_HASH_RESET_HANDLE_STATE

Description:

- Reset HASH handle state.

Parameters:

- __HANDLE__: HASH handle.

Return value:

- None

__HAL_HASH_SET_MDMAT

Description:

- Enable the multi-buffer DMA transfer mode.

Return value:

- None

Notes:

- This bit is set when hashing large files when multiple DMA transfers are needed.

__HAL_HASH_RESET_MDMAT

Description:

- Disable the multi-buffer DMA transfer mode.

Return value:

- None

HAL_HASH_VERSION

Description:

- HAL HASH driver version.

Return value:

- None: HAL HASH driver version 2.0.0

HASH flags definitions

HASH_FLAG_DINIS

16 locations are free in the DIN : new block can be entered in the Peripheral

HASH_FLAG_DCIS

Digest calculation complete

HASH_FLAG_DMAS

DMA interface is enabled (DMAE=1) or a transfer is ongoing

HASH_FLAG_BUSY

The hash core is Busy, processing a block of data

HASH_FLAG_DINNE

DIN not empty : input buffer contains at least one word of data

key length only for HMAC mode

HASH_SHORTKEY

HMAC Key size is <= block size (64 or 128 bytes)

HASH_LONGKEY

HMAC Key size is > block size (64 or 128 bytes)

HASH interrupts definitions

HASH_IT_DINI

A new block can be entered into the input buffer (DIN)

HASH_IT_DCI

Digest calculation complete

HASH Mode

HASH_ALGOMODE_HASH

HASH mode

HASH_ALGOMODE_HMAC

HMAC mode

32 HAL HCD Generic Driver

32.1 HCD Firmware driver registers structures

32.1.1 __HCD_HandleTypeDef

__HCD_HandleTypeDef is defined in the stm32n6xx_hal_hcd.h

Data Fields

- ***HCD_TypeDef * Instance***
- ***HCD_InitTypeDef Init***
- ***HCD_HCTypeDef hc***
- ***HAL_LockTypeDef Lock***
- ***__IO HCD_StateTypeDef State***
- ***__IO uint32_t ErrorCode***
- ***void * pData***
- ***void(* SOFCallback***
- ***void(* ConnectCallback***
- ***void(* DisconnectCallback***
- ***void(* PortEnabledCallback***
- ***void(* PortDisabledCallback***
- ***void(* HC_NotifyURBChangeCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***HCD_TypeDef* __HCD_HandleTypeDef::Instance***
  Register base address
- ***HCD_InitTypeDef __HCD_HandleTypeDef::Init***
  HCD required parameters
- ***HCD_HCTypeDef __HCD_HandleTypeDef::hc[16]***
  Host channels parameters
- ***HAL_LockTypeDef __HCD_HandleTypeDef::Lock***
  HCD peripheral status
- ***__IO HCD_StateTypeDef __HCD_HandleTypeDef::State***
  HCD communication state
- ***__IO uint32_t __HCD_HandleTypeDef::ErrorCode***
  HCD Error code
- ***void* __HCD_HandleTypeDef::pData***
  Pointer Stack Handler
- ***void(* __HCD_HandleTypeDef::SOFCallback)(struct __HCD_HandleTypeDef *hhcd)***
  USB OTG HCD SOF callback
- ***void(* __HCD_HandleTypeDef::ConnectCallback)(struct __HCD_HandleTypeDef *hhcd)***
  USB OTG HCD Connect callback
- ***void(* __HCD_HandleTypeDef::DisconnectCallback)(struct __HCD_HandleTypeDef *hhcd)***
  USB OTG HCD Disconnect callback
- ***void(* __HCD_HandleTypeDef::PortEnabledCallback)(struct __HCD_HandleTypeDef *hhcd)***
  USB OTG HCD Port Enable callback
- ***void(* __HCD_HandleTypeDef::PortDisabledCallback)(struct __HCD_HandleTypeDef *hhcd)***
  USB OTG HCD Port Disable callback
- ***void(* __HCD_HandleTypeDef::HC_NotifyURBChangeCallback)(struct __HCD_HandleTypeDef *hhcd, uint8_t chnum, HCD_URBStateTypeDef urb_state)***
  USB OTG HCD Host Channel Notify URB Change callback

- ***void(* __HCD_HandleTypeDef::MspInitCallback)(struct __HCD_HandleTypeDef *hhcd)***
  USB OTG HCD Msp Init callback
- ***void(* __HCD_HandleTypeDef::MspDeInitCallback)(struct __HCD_HandleTypeDef *hhcd)***
  USB OTG HCD Msp DeInit callback

32.2 HCD Firmware driver API description

The following section lists the various functions of the HCD library.

32.2.1 How to use this driver

1. Declare a HCD_HandleTypeDef handle structure, for example: HCD_HandleTypeDef hhcd;
2. Fill parameters of Init structure in HCD handle
3. Call HAL_HCD_Init() API to initialize the HCD peripheral (Core, Host core, ...)
4. Initialize the HCD low level resources through the HAL_HCD_MspInit() API:
   a. Enable the HCD/USB Low Level interface clock using the following macros
      - __HAL_RCC_USB_OTG_FS_CLK_ENABLE();
      - __HAL_RCC_USB_OTG_HS_CLK_ENABLE(); (For High Speed Mode)
      - __HAL_RCC_USB_OTG_HS_ULPI_CLK_ENABLE(); (For High Speed Mode)
   b. Initialize the related GPIO clocks
   c. Configure HCD pin-out
   d. Configure HCD NVIC interrupt
5. Associate the Upper USB Host stack to the HAL HCD Driver:
   a. hhcd.pData = phost;
6. Enable HCD transmission and reception:
   a. HAL_HCD_Start();

32.2.2 Detailed description of functions

HAL_HCD_Init

Function name

HAL_StatusTypeDef HAL_HCD_Init (HCD_HandleTypeDef * hhcd)

Function description

HAL_HCD_DeInit

Function name

HAL_StatusTypeDef HAL_HCD_DeInit (HCD_HandleTypeDef * hhcd)

Function description

HAL_HCD_HC_Init

Function name

HAL_StatusTypeDef HAL_HCD_HC_Init (HCD_HandleTypeDef * hhcd, uint8_t ch_num, uint8_t epnum, uint8_t dev_address, uint8_t speed, uint8_t ep_type, uint16_t mps)

Function description

HAL_HCD_HC_Halt

Function name

HAL_StatusTypeDef HAL_HCD_HC_Halt (HCD_HandleTypeDef * hhcd, uint8_t ch_num)

Function description

HAL_HCD_MspInit

Function name

void HAL_HCD_MspInit (HCD_HandleTypeDef * hhcd)

Function description

HAL_HCD_MspDeInit

Function name

void HAL_HCD_MspDeInit (HCD_HandleTypeDef * hhcd)

Function description

HAL_HCD_RegisterCallback

Function name

HAL_StatusTypeDef HAL_HCD_RegisterCallback (HCD_HandleTypeDef * hhcd, HAL_HCD_CallbackIDTypeDef CallbackID, pHCD_CallbackTypeDef pCallback)

Function description

HAL_HCD_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_HCD_UnRegisterCallback (HCD_HandleTypeDef * hhcd, HAL_HCD_CallbackIDTypeDef CallbackID)

Function description

HAL_HCD_RegisterHC_NotifyURBChangeCallback

Function name

HAL_StatusTypeDef HAL_HCD_RegisterHC_NotifyURBChangeCallback (HCD_HandleTypeDef * hhcd, pHCD_HC_NotifyURBChangeCallbackTypeDef pCallback)

Function description

HAL_HCD_UnRegisterHC_NotifyURBChangeCallback

Function name

HAL_StatusTypeDef HAL_HCD_UnRegisterHC_NotifyURBChangeCallback (HCD_HandleTypeDef * hhcd)

Function description

HAL_HCD_HC_SubmitRequest

Function name

HAL_StatusTypeDef HAL_HCD_HC_SubmitRequest (HCD_HandleTypeDef * hhcd, uint8_t ch_num, uint8_t direction, uint8_t ep_type, uint8_t token, uint8_t * pbuff, uint16_t length, uint8_t do_ping)

Function description

HAL_HCD_HC_SetHubInfo

Function name

HAL_StatusTypeDef HAL_HCD_HC_SetHubInfo (HCD_HandleTypeDef * hhcd, uint8_t ch_num, uint8_t addr, uint8_t PortNbr)

Function description

HAL_HCD_HC_ClearHubInfo

Function name

HAL_StatusTypeDef HAL_HCD_HC_ClearHubInfo (HCD_HandleTypeDef * hhcd, uint8_t ch_num)

Function description

HAL_HCD_IRQHandler

Function name

void HAL_HCD_IRQHandler (HCD_HandleTypeDef * hhcd)

Function description

HAL_HCD_SOF_Callback

Function name

void HAL_HCD_SOF_Callback (HCD_HandleTypeDef * hhcd)

Function description

HAL_HCD_Connect_Callback

Function name

void HAL_HCD_Connect_Callback (HCD_HandleTypeDef * hhcd)

Function description

HAL_HCD_Disconnect_Callback

Function name

void HAL_HCD_Disconnect_Callback (HCD_HandleTypeDef * hhcd)

Function description

HAL_HCD_PortEnabled_Callback

Function name

void HAL_HCD_PortEnabled_Callback (HCD_HandleTypeDef * hhcd)

Function description

HAL_HCD_PortDisabled_Callback

Function name

void HAL_HCD_PortDisabled_Callback (HCD_HandleTypeDef * hhcd)

Function description

HAL_HCD_HC_NotifyURBChange_Callback

Function name

void HAL_HCD_HC_NotifyURBChange_Callback (HCD_HandleTypeDef * hhcd, uint8_t chnum, HCD_URBStateTypeDef urb_state)

Function description

HAL_HCD_ResetPort

Function name

HAL_StatusTypeDef HAL_HCD_ResetPort (HCD_HandleTypeDef * hhcd)

Function description

HAL_HCD_Start

Function name

HAL_StatusTypeDef HAL_HCD_Start (HCD_HandleTypeDef * hhcd)

Function description

HAL_HCD_Stop

Function name

HAL_StatusTypeDef HAL_HCD_Stop (HCD_HandleTypeDef * hhcd)

Function description

HAL_HCD_GetState

Function name

HCD_StateTypeDef HAL_HCD_GetState (HCD_HandleTypeDef const * hhcd)

Function description

HAL_HCD_HC_GetURBState

Function name

HCD_URBStateTypeDef HAL_HCD_HC_GetURBState (HCD_HandleTypeDef const * hhcd, uint8_t chnum)

Function description

HAL_HCD_HC_GetState

Function name

HCD_HCStateTypeDef HAL_HCD_HC_GetState (HCD_HandleTypeDef const * hhcd, uint8_t chnum)

Function description

HAL_HCD_HC_GetXferCount

Function name

uint32_t HAL_HCD_HC_GetXferCount (HCD_HandleTypeDef const * hhcd, uint8_t chnum)

Function description

HAL_HCD_GetCurrentFrame

Function name

uint32_t HAL_HCD_GetCurrentFrame (HCD_HandleTypeDef * hhcd)

Function description

HAL_HCD_GetCurrentSpeed

Function name

uint32_t HAL_HCD_GetCurrentSpeed (HCD_HandleTypeDef * hhcd)

Function description

32.3 HCD Firmware driver defines

The following section lists the various define and macros of the module.

32.3.1 HCD

HCD

HCD Device Speed

HCD_DEVICE_SPEED_HIGH

HCD_DEVICE_SPEED_FULL

HCD_DEVICE_SPEED_LOW

HCD Error Code definition

HAL_HCD_ERROR_INVALID_CALLBACK

Invalid Callback error

HCD Exported Macros

__HAL_HCD_ENABLE

__HAL_HCD_DISABLE

__HAL_HCD_GET_FLAG

__HAL_HCD_GET_CH_FLAG

__HAL_HCD_CLEAR_FLAG

__HAL_HCD_IS_INVALID_INTERRUPT

__HAL_HCD_CLEAR_HC_INT

__HAL_HCD_MASK_HALT_HC_INT

__HAL_HCD_UNMASK_HALT_HC_INT

__HAL_HCD_MASK_ACK_HC_INT

__HAL_HCD_UNMASK_ACK_HC_INT

__HAL_HCD_SET_HC_CSPLT

__HAL_HCD_CLEAR_HC_CSPLT

__HAL_HCD_CLEAR_HC_SSPLT

HCD PHY Module

HCD_PHY_ULPI

HCD_PHY_EMBEDDED

HCD Speed

HCD_SPEED_HIGH

HCD_SPEED_FULL

HCD_SPEED_LOW

33 HAL I2C Generic Driver

33.1 I2C Firmware driver registers structures

33.1.1 I2C_InitTypeDef

I2C_InitTypeDef is defined in the stm32n6xx_hal_i2c.h

Data Fields

- ***uint32_t Timing***
- ***uint32_t OwnAddress1***
- ***uint32_t AddressingMode***
- ***uint32_t DualAddressMode***
- ***uint32_t OwnAddress2***
- ***uint32_t OwnAddress2Masks***
- ***uint32_t GeneralCallMode***
- ***uint32_t NoStretchMode***

Field Documentation

- ***uint32_t I2C_InitTypeDef::Timing***
  Specifies the I2C_TIMINGR_register value. This parameter calculated by referring to I2C initialization section in Reference manual
- ***uint32_t I2C_InitTypeDef::OwnAddress1***
  Specifies the first device own address. This parameter can be a 7-bit or 10-bit address.
- ***uint32_t I2C_InitTypeDef::AddressingMode***
  Specifies if 7-bit or 10-bit addressing mode is selected. This parameter can be a value of ***I2C_ADDRESSING_MODE***
- ***uint32_t I2C_InitTypeDef::DualAddressMode***
  Specifies if dual addressing mode is selected. This parameter can be a value of ***I2C_DUAL_ADDRESSING_MODE***
- ***uint32_t I2C_InitTypeDef::OwnAddress2***
  Specifies the second device own address if dual addressing mode is selected This parameter can be a 7-bit address.
- ***uint32_t I2C_InitTypeDef::OwnAddress2Masks***
  Specifies the acknowledge mask address second device own address if dual addressing mode is selected. This parameter can be a value of ***I2C_OWN_ADDRESS2_MASKS***
- ***uint32_t I2C_InitTypeDef::GeneralCallMode***
  Specifies if general call mode is selected. This parameter can be a value of ***I2C_GENERAL_CALL_ADDRESSING_MODE***
- ***uint32_t I2C_InitTypeDef::NoStretchMode***
  Specifies if nostretch mode is selected. This parameter can be a value of ***I2C_NOSTRETCH_MODE***

33.1.2 __I2C_HandleTypeDef

__I2C_HandleTypeDef is defined in the stm32n6xx_hal_i2c.h

Data Fields

- ***I2C_TypeDef * Instance***
- ***I2C_InitTypeDef Init***
- ***uint8_t * pBuffPtr***
- ***uint16_t XferSize***
- ***__IO uint16_t XferCount***
- ***__IO uint32_t XferOptions***
- ***__IO uint32_t PreviousState***
- ***HAL_StatusTypeDef(* XferISR***
- ***DMA_HandleTypeDef * hdmatx***

- ***DMA_HandleTypeDef * hdmarx***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_I2C_StateTypeDef State***
- ***__IO HAL_I2C_ModeTypeDef Mode***
- ***__IO uint32_t ErrorCode***
- ***__IO uint32_t AddrEventCount***
- ***__IO uint32_t Devaddress***
- ***__IO uint32_t Memaddress***
- ***void(* MasterTxCpltCallback***
- ***void(* MasterRxCpltCallback***
- ***void(* SlaveTxCpltCallback***
- ***void(* SlaveRxCpltCallback***
- ***void(* ListenCpltCallback***
- ***void(* MemTxCpltCallback***
- ***void(* MemRxCpltCallback***
- ***void(* ErrorCallback***
- ***void(* AbortCpltCallback***
- ***void(* AddrCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***I2C_TypeDef* __I2C_HandleTypeDef::Instance***
  I2C registers base address
- ***I2C_InitTypeDef __I2C_HandleTypeDef::Init***
  I2C communication parameters
- ***uint8_t* __I2C_HandleTypeDef::pBuffPtr***
  Pointer to I2C transfer buffer
- ***uint16_t __I2C_HandleTypeDef::XferSize***
  I2C transfer size
- ***__IO uint16_t __I2C_HandleTypeDef::XferCount***
  I2C transfer counter
- ***__IO uint32_t __I2C_HandleTypeDef::XferOptions***
  I2C sequantial transfer options, this parameter can be a value of ***I2C_XFEROPTIONS***
- ***__IO uint32_t __I2C_HandleTypeDef::PreviousState***
  I2C communication Previous state
- ***HAL_StatusTypeDef(* __I2C_HandleTypeDef::XferISR)(struct __I2C_HandleTypeDef *hi2c, uint32_t ITFlags, uint32_t ITSources)***
  I2C transfer IRQ handler function pointer
- ***DMA_HandleTypeDef* __I2C_HandleTypeDef::hdmatx***
  I2C Tx DMA handle parameters
- ***DMA_HandleTypeDef* __I2C_HandleTypeDef::hdmarx***
  I2C Rx DMA handle parameters
- ***HAL_LockTypeDef __I2C_HandleTypeDef::Lock***
  I2C locking object
- ***__IO HAL_I2C_StateTypeDef __I2C_HandleTypeDef::State***
  I2C communication state
- ***__IO HAL_I2C_ModeTypeDef __I2C_HandleTypeDef::Mode***
  I2C communication mode
- ***__IO uint32_t __I2C_HandleTypeDef::ErrorCode***
  I2C Error code
- ***__IO uint32_t __I2C_HandleTypeDef::AddrEventCount***
  I2C Address Event counter

- ***__IO uint32_t __I2C_HandleTypeDef::Devaddress***
  I2C Target device address
- ***__IO uint32_t __I2C_HandleTypeDef::Memaddress***
  I2C Target memory address
- ***void(* __I2C_HandleTypeDef::MasterTxCpltCallback)(struct __I2C_HandleTypeDef *hi2c)***
  I2C Master Tx Transfer completed callback
- ***void(* __I2C_HandleTypeDef::MasterRxCpltCallback)(struct __I2C_HandleTypeDef *hi2c)***
  I2C Master Rx Transfer completed callback
- ***void(* __I2C_HandleTypeDef::SlaveTxCpltCallback)(struct __I2C_HandleTypeDef *hi2c)***
  I2C Slave Tx Transfer completed callback
- ***void(* __I2C_HandleTypeDef::SlaveRxCpltCallback)(struct __I2C_HandleTypeDef *hi2c)***
  I2C Slave Rx Transfer completed callback
- ***void(* __I2C_HandleTypeDef::ListenCpltCallback)(struct __I2C_HandleTypeDef *hi2c)***
  I2C Listen Complete callback
- ***void(* __I2C_HandleTypeDef::MemTxCpltCallback)(struct __I2C_HandleTypeDef *hi2c)***
  I2C Memory Tx Transfer completed callback
- ***void(* __I2C_HandleTypeDef::MemRxCpltCallback)(struct __I2C_HandleTypeDef *hi2c)***
  I2C Memory Rx Transfer completed callback
- ***void(* __I2C_HandleTypeDef::ErrorCallback)(struct __I2C_HandleTypeDef *hi2c)***
  I2C Error callback
- ***void(* __I2C_HandleTypeDef::AbortCpltCallback)(struct __I2C_HandleTypeDef *hi2c)***
  I2C Abort callback
- ***void(* __I2C_HandleTypeDef::AddrCallback)(struct __I2C_HandleTypeDef *hi2c, uint8_t TransferDirection, uint16_t AddrMatchCode)***
  I2C Slave Address Match callback
- ***void(* __I2C_HandleTypeDef::MspInitCallback)(struct __I2C_HandleTypeDef *hi2c)***
  I2C Msp Init callback
- ***void(* __I2C_HandleTypeDef::MspDeInitCallback)(struct __I2C_HandleTypeDef *hi2c)***
  I2C Msp DeInit callback

33.2 I2C Firmware driver API description

The following section lists the various functions of the I2C library.

33.2.1 How to use this driver

The I2C HAL driver can be used as follows:

1. Declare a I2C_HandleTypeDef handle structure, for example: I2C_HandleTypeDef hi2c;
2. Initialize the I2C low level resources by implementing the HAL_I2C_MspInit() API:
   a. Enable the I2Cx interface clock
   b. I2C pins configuration
      - Enable the clock for the I2C GPIOs
      - Configure I2C pins as alternate function open-drain
   c. NVIC configuration if you need to use interrupt process
      - Configure the I2Cx interrupt priority
      - Enable the NVIC I2C IRQ Channel
   d. DMA Configuration if you need to use DMA process
      - Declare a DMA_HandleTypeDef handle structure for the transmit or receive channel
      - Enable the DMAx interface clock using
      - Configure the DMA handle parameters
      - Configure the DMA Tx or Rx channel
      - Associate the initialized DMA handle to the hi2c DMA Tx or Rx handle
      - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx or Rx channel

3. Configure the Communication Clock Timing, Own Address1, Master Addressing mode, Dual Addressing mode, Own Address2, Own Address2 Mask, General call and Nostretch mode in the hi2c Init structure.
4. Initialize the I2C registers by calling the HAL_I2C_Init(), configures also the low level Hardware (GPIO, CLOCK, NVIC...etc) by calling the customized HAL_I2C_MspInit(&hi2c) API.
5. To check if target device is ready for communication, use the function HAL_I2C_IsDeviceReady()
6. For I2C IO and IO MEM operations, three operation modes are available within this driver :

Polling mode IO operation

- Transmit in master mode an amount of data in blocking mode using HAL_I2C_Master_Transmit()
- Receive in master mode an amount of data in blocking mode using HAL_I2C_Master_Receive()
- Transmit in slave mode an amount of data in blocking mode using HAL_I2C_Slave_Transmit()
- Receive in slave mode an amount of data in blocking mode using HAL_I2C_Slave_Receive()

Polling mode IO MEM operation

- Write an amount of data in blocking mode to a specific memory address using HAL_I2C_Mem_Write()
- Read an amount of data in blocking mode from a specific memory address using HAL_I2C_Mem_Read()

Interrupt mode IO operation

- Transmit in master mode an amount of data in non-blocking mode using HAL_I2C_Master_Transmit_IT()
- At transmission end of transfer, HAL_I2C_MasterTxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_MasterTxCpltCallback()
- Receive in master mode an amount of data in non-blocking mode using HAL_I2C_Master_Receive_IT()
- At reception end of transfer, HAL_I2C_MasterRxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_MasterRxCpltCallback()
- Transmit in slave mode an amount of data in non-blocking mode using HAL_I2C_Slave_Transmit_IT()
- At transmission end of transfer, HAL_I2C_SlaveTxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_SlaveTxCpltCallback()
- Receive in slave mode an amount of data in non-blocking mode using HAL_I2C_Slave_Receive_IT()
- At reception end of transfer, HAL_I2C_SlaveRxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_SlaveRxCpltCallback()
- In case of transfer Error, HAL_I2C_ErrorCallback() function is executed and users can add their own code by customization of function pointer HAL_I2C_ErrorCallback()
- Abort a master or memory I2C process communication with Interrupt using HAL_I2C_Master_Abort_IT()
- End of abort process, HAL_I2C_AbortCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_AbortCpltCallback()
- Discard a slave I2C process communication using __HAL_I2C_GENERATE_NACK() macro. This action will inform Master to generate a Stop condition to discard the communication.

Interrupt mode or DMA mode IO sequential operation

Note: These interfaces allow to manage a sequential transfer with a repeated start condition when a direction change during transfer

- A specific option field manage the different steps of a sequential transfer

- Option field values are defined through I2C_XFEROPTIONS and are listed below:
  - I2C_FIRST_AND_LAST_FRAME: No sequential usage, functional is same as associated interfaces in no sequential mode
  - I2C_FIRST_FRAME: Sequential usage, this option allow to manage a sequence with start condition, address and data to transfer without a final stop condition
  - I2C_FIRST_AND_NEXT_FRAME: Sequential usage (Master only), this option allow to manage a sequence with start condition, address and data to transfer without a final stop condition, an then permit a call the same master sequential interface several times (like HAL_I2C_Master_Seq_Transmit_IT() then HAL_I2C_Master_Seq_Transmit_IT() or HAL_I2C_Master_Seq_Transmit_DMA() then HAL_I2C_Master_Seq_Transmit_DMA())
  - I2C_NEXT_FRAME: Sequential usage, this option allow to manage a sequence with a restart condition, address and with new data to transfer if the direction change or manage only the new data to transfer if no direction change and without a final stop condition in both cases
  - I2C_LAST_FRAME: Sequential usage, this option allow to manage a sequance with a restart condition, address and with new data to transfer if the direction change or manage only the new data to transfer if no direction change and with a final stop condition in both cases
  - I2C_LAST_FRAME_NO_STOP: Sequential usage (Master only), this option allow to manage a restart condition after several call of the same master sequential interface several times (link with option I2C_FIRST_AND_NEXT_FRAME). Usage can, transfer several bytes one by one using HAL_I2C_Master_Seq_Transmit_IT or HAL_I2C_Master_Seq_Receive_IT or HAL_I2C_Master_Seq_Transmit_DMA or HAL_I2C_Master_Seq_Receive_DMA with option I2C_FIRST_AND_NEXT_FRAME then I2C_NEXT_FRAME. Then usage of this option I2C_LAST_FRAME_NO_STOP at the last Transmit or Receive sequence permit to call the opposite interface Receive or Transmit without stopping the communication and so generate a restart condition.
  - I2C_OTHER_FRAME: Sequential usage (Master only), this option allow to manage a restart condition after each call of the same master sequential interface. Usage can, transfer several bytes one by one with a restart with slave address between each bytes using HAL_I2C_Master_Seq_Transmit_IT or HAL_I2C_Master_Seq_Receive_IT or HAL_I2C_Master_Seq_Transmit_DMA or HAL_I2C_Master_Seq_Receive_DMA with option I2C_FIRST_FRAME then I2C_OTHER_FRAME. Then usage of this option I2C_OTHER_AND_LAST_FRAME at the last frame to help automatic generation of STOP condition.

- Different sequential I2C interfaces are listed below:
  - Sequential transmit in master I2C mode an amount of data in non-blocking mode using HAL_I2C_Master_Seq_Transmit_IT() or using HAL_I2C_Master_Seq_Transmit_DMA()
    - At transmission end of current frame transfer, HAL_I2C_MasterTxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_MasterTxCpltCallback()
  - Sequential receive in master I2C mode an amount of data in non-blocking mode using HAL_I2C_Master_Seq_Receive_IT() or using HAL_I2C_Master_Seq_Receive_DMA()
    - At reception end of current frame transfer, HAL_I2C_MasterRxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_MasterRxCpltCallback()
  - Abort a master or memory IT or DMA I2C process communication with Interrupt using HAL_I2C_Master_Abort_IT()
    - End of abort process, HAL_I2C_AbortCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_AbortCpltCallback()
  - Enable/disable the Address listen mode in slave I2C mode using HAL_I2C_EnableListen_IT() HAL_I2C_DisableListen_IT()
    - When address slave I2C match, HAL_I2C_AddrCallback() is executed and users can add their own code to check the Address Match Code and the transmission direction request by master (Write/Read).
    - At Listen mode end HAL_I2C_ListenCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_ListenCpltCallback()
  - Sequential transmit in slave I2C mode an amount of data in non-blocking mode using HAL_I2C_Slave_Seq_Transmit_IT() or using HAL_I2C_Slave_Seq_Transmit_DMA()
    - At transmission end of current frame transfer, HAL_I2C_SlaveTxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_SlaveTxCpltCallback()
  - Sequential receive in slave I2C mode an amount of data in non-blocking mode using HAL_I2C_Slave_Seq_Receive_IT() or using HAL_I2C_Slave_Seq_Receive_DMA()
    - At reception end of current frame transfer, HAL_I2C_SlaveRxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_SlaveRxCpltCallback()
  - In case of transfer Error, HAL_I2C_ErrorCallback() function is executed and users can add their own code by customization of function pointer HAL_I2C_ErrorCallback()
  - Discard a slave I2C process communication using __HAL_I2C_GENERATE_NACK() macro. This action will inform Master to generate a Stop condition to discard the communication.

Interrupt mode IO MEM operation

- Write an amount of data in non-blocking mode with Interrupt to a specific memory address using HAL_I2C_Mem_Write_IT()
- At Memory end of write transfer, HAL_I2C_MemTxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_MemTxCpltCallback()
- Read an amount of data in non-blocking mode with Interrupt from a specific memory address using HAL_I2C_Mem_Read_IT()
- At Memory end of read transfer, HAL_I2C_MemRxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_MemRxCpltCallback()
- In case of transfer Error, HAL_I2C_ErrorCallback() function is executed and users can add their own code by customization of function pointer HAL_I2C_ErrorCallback()

DMA mode IO operation

- Transmit in master mode an amount of data in non-blocking mode (DMA) using HAL_I2C_Master_Transmit_DMA()
- At transmission end of transfer, HAL_I2C_MasterTxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_MasterTxCpltCallback()

- Receive in master mode an amount of data in non-blocking mode (DMA) using HAL_I2C_Master_Receive_DMA()
- At reception end of transfer, HAL_I2C_MasterRxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_MasterRxCpltCallback()
- Transmit in slave mode an amount of data in non-blocking mode (DMA) using HAL_I2C_Slave_Transmit_DMA()
- At transmission end of transfer, HAL_I2C_SlaveTxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_SlaveTxCpltCallback()
- Receive in slave mode an amount of data in non-blocking mode (DMA) using HAL_I2C_Slave_Receive_DMA()
- At reception end of transfer, HAL_I2C_SlaveRxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_SlaveRxCpltCallback()
- In case of transfer Error, HAL_I2C_ErrorCallback() function is executed and users can add their own code by customization of function pointer HAL_I2C_ErrorCallback()
- Abort a master or memory I2C process communication with Interrupt using HAL_I2C_Master_Abort_IT()
- End of abort process, HAL_I2C_AbortCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_AbortCpltCallback()
- Discard a slave I2C process communication using __HAL_I2C_GENERATE_NACK() macro. This action will inform Master to generate a Stop condition to discard the communication.

DMA mode IO MEM operation

- Write an amount of data in non-blocking mode with DMA to a specific memory address using HAL_I2C_Mem_Write_DMA()
- At Memory end of write transfer, HAL_I2C_MemTxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_MemTxCpltCallback()
- Read an amount of data in non-blocking mode with DMA from a specific memory address using HAL_I2C_Mem_Read_DMA()
- At Memory end of read transfer, HAL_I2C_MemRxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_MemRxCpltCallback()
- In case of transfer Error, HAL_I2C_ErrorCallback() function is executed and users can add their own code by customization of function pointer HAL_I2C_ErrorCallback()

I2C HAL driver macros list

Below the list of most used macros in I2C HAL driver.

- __HAL_I2C_ENABLE: Enable the I2C peripheral
- __HAL_I2C_DISABLE: Disable the I2C peripheral
- __HAL_I2C_GENERATE_NACK: Generate a Non-Acknowledge I2C peripheral in Slave mode
- __HAL_I2C_GET_FLAG: Check whether the specified I2C flag is set or not
- __HAL_I2C_CLEAR_FLAG: Clear the specified I2C pending flag
- __HAL_I2C_ENABLE_IT: Enable the specified I2C interrupt
- __HAL_I2C_DISABLE_IT: Disable the specified I2C interrupt

Callback registration

The compilation flag USE_HAL_I2C_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use Functions HAL_I2C_RegisterCallback() or HAL_I2C_RegisterAddrCallback() to register an interrupt callback.

Function HAL_I2C_RegisterCallback() allows to register following callbacks:

- MasterTxCpltCallback : callback for Master transmission end of transfer.
- MasterRxCpltCallback : callback for Master reception end of transfer.
- SlaveTxCpltCallback : callback for Slave transmission end of transfer.
- SlaveRxCpltCallback : callback for Slave reception end of transfer.
- ListenCpltCallback : callback for end of listen mode.
- MemTxCpltCallback : callback for Memory transmission end of transfer.

- MemRxCpltCallback : callback for Memory reception end of transfer.
- ErrorCallback : callback for error detection.
- AbortCpltCallback : callback for abort completion process.
- MspInitCallback : callback for Msp Init.
- MspDeInitCallback : callback for Msp DeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

For specific callback AddrCallback use dedicated register callbacks : HAL_I2C_RegisterAddrCallback().

Use function HAL_I2C_UnRegisterCallback to reset a callback to the default weak function. HAL_I2C_UnRegisterCallback takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks:

- MasterTxCpltCallback : callback for Master transmission end of transfer.
- MasterRxCpltCallback : callback for Master reception end of transfer.
- SlaveTxCpltCallback : callback for Slave transmission end of transfer.
- SlaveRxCpltCallback : callback for Slave reception end of transfer.
- ListenCpltCallback : callback for end of listen mode.
- MemTxCpltCallback : callback for Memory transmission end of transfer.
- MemRxCpltCallback : callback for Memory reception end of transfer.
- ErrorCallback : callback for error detection.
- AbortCpltCallback : callback for abort completion process.
- MspInitCallback : callback for Msp Init.
- MspDeInitCallback : callback for Msp DeInit.

For callback AddrCallback use dedicated register callbacks : HAL_I2C_UnRegisterAddrCallback().

By default, after the HAL_I2C_Init() and when the state is HAL_I2C_STATE_RESET all callbacks are set to the corresponding weak functions: examples HAL_I2C_MasterTxCpltCallback(), HAL_I2C_MasterRxCpltCallback(). Exception done for MspInit and MspDeInit functions that are reset to the legacy weak functions in the HAL_I2C_Init()/ HAL_I2C_DeInit() only when these callbacks are null (not registered beforehand). If MspInit or MspDeInit are not null, the HAL_I2C_Init()/ HAL_I2C_DeInit() keep and use the user MspInit/MspDeInit callbacks (registered beforehand) whatever the state.

Callbacks can be registered/unregistered in HAL_I2C_STATE_READY state only. Exception done MspInit/ MspDeInit functions that can be registered/unregistered in HAL_I2C_STATE_READY or HAL_I2C_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. Then, the user first registers the MspInit/MspDeInit user callbacks using HAL_I2C_RegisterCallback() before calling HAL_I2C_DeInit() or HAL_I2C_Init() function.

When the compilation flag USE_HAL_I2C_REGISTER_CALLBACKS is set to 0 or not defined, the callback registration feature is not available and all callbacks are set to the corresponding weak functions.

Note: You can refer to the I2C HAL driver header file for more useful macros

33.2.2 Detailed description of functions

HAL_I2C_Init

Function name

HAL_StatusTypeDef HAL_I2C_Init (I2C_HandleTypeDef * hi2c)

Function description

HAL_I2C_DeInit

Function name

HAL_StatusTypeDef HAL_I2C_DeInit (I2C_HandleTypeDef * hi2c)

Function description

HAL_I2C_MspInit

Function name

void HAL_I2C_MspInit (I2C_HandleTypeDef * hi2c)

Function description

HAL_I2C_MspDeInit

Function name

void HAL_I2C_MspDeInit (I2C_HandleTypeDef * hi2c)

Function description

HAL_I2C_RegisterCallback

Function name

HAL_StatusTypeDef HAL_I2C_RegisterCallback (I2C_HandleTypeDef * hi2c, HAL_I2C_CallbackIDTypeDef CallbackID, pI2C_CallbackTypeDef pCallback)

Function description

HAL_I2C_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_I2C_UnRegisterCallback (I2C_HandleTypeDef * hi2c, HAL_I2C_CallbackIDTypeDef CallbackID)

Function description

HAL_I2C_RegisterAddrCallback

Function name

HAL_StatusTypeDef HAL_I2C_RegisterAddrCallback (I2C_HandleTypeDef * hi2c, pI2C_AddrCallbackTypeDef pCallback)

Function description

HAL_I2C_UnRegisterAddrCallback

Function name

HAL_StatusTypeDef HAL_I2C_UnRegisterAddrCallback (I2C_HandleTypeDef * hi2c)

Function description

HAL_I2C_Master_Transmit

Function name

HAL_StatusTypeDef HAL_I2C_Master_Transmit (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

HAL_I2C_Master_Receive

Function name

HAL_StatusTypeDef HAL_I2C_Master_Receive (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

HAL_I2C_Slave_Transmit

Function name

HAL_StatusTypeDef HAL_I2C_Slave_Transmit (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

HAL_I2C_Slave_Receive

Function name

HAL_StatusTypeDef HAL_I2C_Slave_Receive (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

HAL_I2C_Mem_Write

Function name

HAL_StatusTypeDef HAL_I2C_Mem_Write (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

HAL_I2C_Mem_Read

Function name

HAL_StatusTypeDef HAL_I2C_Mem_Read (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

HAL_I2C_IsDeviceReady

Function name

HAL_StatusTypeDef HAL_I2C_IsDeviceReady (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint32_t Trials, uint32_t Timeout)

Function description

HAL_I2C_Master_Transmit_IT

Function name

HAL_StatusTypeDef HAL_I2C_Master_Transmit_IT (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size)

Function description

HAL_I2C_Master_Receive_IT

Function name

HAL_StatusTypeDef HAL_I2C_Master_Receive_IT (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size)

Function description

HAL_I2C_Slave_Transmit_IT

Function name

HAL_StatusTypeDef HAL_I2C_Slave_Transmit_IT (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size)

Function description

HAL_I2C_Slave_Receive_IT

Function name

HAL_StatusTypeDef HAL_I2C_Slave_Receive_IT (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size)

Function description

HAL_I2C_Mem_Write_IT

Function name

HAL_StatusTypeDef HAL_I2C_Mem_Write_IT (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size)

Function description

HAL_I2C_Mem_Read_IT

Function name

HAL_StatusTypeDef HAL_I2C_Mem_Read_IT (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size)

Function description

HAL_I2C_Master_Seq_Transmit_IT

Function name

HAL_StatusTypeDef HAL_I2C_Master_Seq_Transmit_IT (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t XferOptions)

Function description

HAL_I2C_Master_Seq_Receive_IT

Function name

HAL_StatusTypeDef HAL_I2C_Master_Seq_Receive_IT (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t XferOptions)

Function description

HAL_I2C_Slave_Seq_Transmit_IT

Function name

HAL_StatusTypeDef HAL_I2C_Slave_Seq_Transmit_IT (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size, uint32_t XferOptions)

Function description

HAL_I2C_Slave_Seq_Receive_IT

Function name

HAL_StatusTypeDef HAL_I2C_Slave_Seq_Receive_IT (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size, uint32_t XferOptions)

Function description

HAL_I2C_EnableListen_IT

Function name

HAL_StatusTypeDef HAL_I2C_EnableListen_IT (I2C_HandleTypeDef * hi2c)

Function description

HAL_I2C_DisableListen_IT

Function name

HAL_StatusTypeDef HAL_I2C_DisableListen_IT (I2C_HandleTypeDef * hi2c)

Function description

HAL_I2C_Master_Abort_IT

Function name

HAL_StatusTypeDef HAL_I2C_Master_Abort_IT (I2C_HandleTypeDef * hi2c, uint16_t DevAddress)

Function description

HAL_I2C_Master_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_I2C_Master_Transmit_DMA (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size)

Function description

HAL_I2C_Master_Receive_DMA

Function name

HAL_StatusTypeDef HAL_I2C_Master_Receive_DMA (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size)

Function description

HAL_I2C_Slave_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_I2C_Slave_Transmit_DMA (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size)

Function description

HAL_I2C_Slave_Receive_DMA

Function name

HAL_StatusTypeDef HAL_I2C_Slave_Receive_DMA (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size)

Function description

HAL_I2C_Mem_Write_DMA

Function name

HAL_StatusTypeDef HAL_I2C_Mem_Write_DMA (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size)

Function description

HAL_I2C_Mem_Read_DMA

Function name

HAL_StatusTypeDef HAL_I2C_Mem_Read_DMA (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size)

Function description

HAL_I2C_Master_Seq_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_I2C_Master_Seq_Transmit_DMA (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t XferOptions)

Function description

HAL_I2C_Master_Seq_Receive_DMA

Function name

HAL_StatusTypeDef HAL_I2C_Master_Seq_Receive_DMA (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t XferOptions)

Function description

HAL_I2C_Slave_Seq_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_I2C_Slave_Seq_Transmit_DMA (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size, uint32_t XferOptions)

Function description

HAL_I2C_Slave_Seq_Receive_DMA

Function name

HAL_StatusTypeDef HAL_I2C_Slave_Seq_Receive_DMA (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size, uint32_t XferOptions)

Function description

HAL_I2C_EV_IRQHandler

Function name

void HAL_I2C_EV_IRQHandler (I2C_HandleTypeDef * hi2c)

Function description

HAL_I2C_ER_IRQHandler

Function name

void HAL_I2C_ER_IRQHandler (I2C_HandleTypeDef * hi2c)

Function description

HAL_I2C_MasterTxCpltCallback

Function name

void HAL_I2C_MasterTxCpltCallback (I2C_HandleTypeDef * hi2c)

Function description

HAL_I2C_MasterRxCpltCallback

Function name

void HAL_I2C_MasterRxCpltCallback (I2C_HandleTypeDef * hi2c)

Function description

HAL_I2C_SlaveTxCpltCallback

Function name

void HAL_I2C_SlaveTxCpltCallback (I2C_HandleTypeDef * hi2c)

Function description

HAL_I2C_SlaveRxCpltCallback

Function name

void HAL_I2C_SlaveRxCpltCallback (I2C_HandleTypeDef * hi2c)

Function description

HAL_I2C_AddrCallback

Function name

void HAL_I2C_AddrCallback (I2C_HandleTypeDef * hi2c, uint8_t TransferDirection, uint16_t AddrMatchCode)

Function description

HAL_I2C_ListenCpltCallback

Function name

void HAL_I2C_ListenCpltCallback (I2C_HandleTypeDef * hi2c)

Function description

HAL_I2C_MemTxCpltCallback

Function name

void HAL_I2C_MemTxCpltCallback (I2C_HandleTypeDef * hi2c)

Function description

HAL_I2C_MemRxCpltCallback

Function name

void HAL_I2C_MemRxCpltCallback (I2C_HandleTypeDef * hi2c)

Function description

HAL_I2C_ErrorCallback

Function name

void HAL_I2C_ErrorCallback (I2C_HandleTypeDef * hi2c)

Function description

HAL_I2C_AbortCpltCallback

Function name

void HAL_I2C_AbortCpltCallback (I2C_HandleTypeDef * hi2c)

Function description

HAL_I2C_GetState

Function name

HAL_I2C_StateTypeDef HAL_I2C_GetState (const I2C_HandleTypeDef * hi2c)

Function description

HAL_I2C_GetMode

Function name

HAL_I2C_ModeTypeDef HAL_I2C_GetMode (const I2C_HandleTypeDef * hi2c)

Function description

HAL_I2C_GetError

Function name

uint32_t HAL_I2C_GetError (const I2C_HandleTypeDef * hi2c)

Function description

33.3 I2C Firmware driver defines

The following section lists the various define and macros of the module.

33.3.1 I2C

I2C

I2C Addressing Mode

I2C_ADDRESSINGMODE_7BIT

I2C_ADDRESSINGMODE_10BIT

I2C Dual Addressing Mode

I2C_DUALADDRESS_DISABLE

I2C_DUALADDRESS_ENABLE

I2C Error Code definition

HAL_I2C_ERROR_NONE
: No error

HAL_I2C_ERROR_BERR
: BERR error

HAL_I2C_ERROR_ARLO
: ARLO error

HAL_I2C_ERROR_AF
: ACKF error

HAL_I2C_ERROR_OVR
: OVR error

HAL_I2C_ERROR_DMA
: DMA transfer error

HAL_I2C_ERROR_TIMEOUT
: Timeout error

HAL_I2C_ERROR_SIZE
: Size Management error

HAL_I2C_ERROR_DMA_PARAM
: DMA Parameter Error

HAL_I2C_ERROR_INVALID_CALLBACK
: Invalid Callback error

HAL_I2C_ERROR_INVALID_PARAM
: Invalid Parameters error

I2C Exported Macros

__HAL_I2C_RESET_HANDLE_STATE

Description:

- Reset I2C handle state.

Parameters:

- __HANDLE__: specifies the I2C Handle.

Return value:

- None

__HAL_I2C_ENABLE_IT

Description:

- Enable the specified I2C interrupt.

Parameters:

- __HANDLE__: specifies the I2C Handle.
- __INTERRUPT__: specifies the interrupt source to enable. This parameter can be one of the following values:
  - I2C_IT_ERRI Errors interrupt enable
  - I2C_IT_TCI Transfer complete interrupt enable
  - I2C_IT_STOPI STOP detection interrupt enable
  - I2C_IT_NACKI NACK received interrupt enable
  - I2C_IT_ADDRI Address match interrupt enable
  - I2C_IT_RXI RX interrupt enable
  - I2C_IT_TXI TX interrupt enable

Return value:

- None

__HAL_I2C_DISABLE_IT

Description:

- Disable the specified I2C interrupt.

Parameters:

- __HANDLE__: specifies the I2C Handle.
- __INTERRUPT__: specifies the interrupt source to disable. This parameter can be one of the following values:
  - I2C_IT_ERRI Errors interrupt enable
  - I2C_IT_TCI Transfer complete interrupt enable
  - I2C_IT_STOPI STOP detection interrupt enable
  - I2C_IT_NACKI NACK received interrupt enable
  - I2C_IT_ADDRI Address match interrupt enable
  - I2C_IT_RXI RX interrupt enable
  - I2C_IT_TXI TX interrupt enable

Return value:

- None

__HAL_I2C_GET_IT_SOURCE

Description:

- Check whether the specified I2C interrupt source is enabled or not.

Parameters:

- __HANDLE__: specifies the I2C Handle.
- __INTERRUPT__: specifies the I2C interrupt source to check. This parameter can be one of the following values:
  - I2C_IT_ERRI Errors interrupt enable
  - I2C_IT_TCI Transfer complete interrupt enable
  - I2C_IT_STOPI STOP detection interrupt enable
  - I2C_IT_NACKI NACK received interrupt enable
  - I2C_IT_ADDRI Address match interrupt enable
  - I2C_IT_RXI RX interrupt enable
  - I2C_IT_TXI TX interrupt enable

Return value:

- The: new state of __INTERRUPT__ (SET or RESET).

I2C_FLAG_MASK

Description:

- Check whether the specified I2C flag is set or not.

Parameters:

- __HANDLE__: specifies the I2C Handle.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - I2C_FLAG_TXE Transmit data register empty
  - I2C_FLAG_TXIS Transmit interrupt status
  - I2C_FLAG_RXNE Receive data register not empty
  - I2C_FLAG_ADDR Address matched (slave mode)
  - I2C_FLAG_AF Acknowledge failure received flag
  - I2C_FLAG_STOPF STOP detection flag
  - I2C_FLAG_TC Transfer complete (master mode)
  - I2C_FLAG_TCR Transfer complete reload
  - I2C_FLAG_BERR Bus error
  - I2C_FLAG_ARLO Arbitration lost
  - I2C_FLAG_OVR Overrun/Underrun
  - I2C_FLAG_PECERR PEC error in reception
  - I2C_FLAG_TIMEOUT Timeout or Tlow detection flag
  - I2C_FLAG_ALERT SMBus alert
  - I2C_FLAG_BUSY Bus busy
  - I2C_FLAG_DIR Transfer direction (slave mode)

Return value:

- The: new state of __FLAG__ (SET or RESET).

__HAL_I2C_GET_FLAG

__HAL_I2C_CLEAR_FLAG

Description:

- Clear the I2C pending flags which are cleared by writing 1 in a specific bit.

Parameters:

- __HANDLE__: specifies the I2C Handle.
- __FLAG__: specifies the flag to clear. This parameter can be any combination of the following values:
  - I2C_FLAG_TXE Transmit data register empty
  - I2C_FLAG_ADDR Address matched (slave mode)
  - I2C_FLAG_AF Acknowledge failure received flag
  - I2C_FLAG_STOPF STOP detection flag
  - I2C_FLAG_BERR Bus error
  - I2C_FLAG_ARLO Arbitration lost
  - I2C_FLAG_OVR Overrun/Underrun
  - I2C_FLAG_PECERR PEC error in reception
  - I2C_FLAG_TIMEOUT Timeout or Tlow detection flag
  - I2C_FLAG_ALERT SMBus alert

Return value:

- None

__HAL_I2C_ENABLE

Description:

- Enable the specified I2C peripheral.

Parameters:

- __HANDLE__: specifies the I2C Handle.

Return value:

- None

__HAL_I2C_DISABLE

Description:

- Disable the specified I2C peripheral.

Parameters:

- __HANDLE__: specifies the I2C Handle.

Return value:

- None

__HAL_I2C_GENERATE_NACK

Description:

- Generate a Non-Acknowledge I2C peripheral in Slave mode.

Parameters:

- __HANDLE__: specifies the I2C Handle.

Return value:

- None

I2C Flag definition

I2C_FLAG_TXE

I2C_FLAG_TXIS

I2C_FLAG_RXNE

I2C_FLAG_ADDR

I2C_FLAG_AF

I2C_FLAG_STOPF

I2C_FLAG_TC

I2C_FLAG_TCR

I2C_FLAG_BERR

I2C_FLAG_ARLO

I2C_FLAG_OVR

I2C_FLAG_PECERR

I2C_FLAG_TIMEOUT

I2C_FLAG_ALERT

I2C_FLAG_BUSY

I2C_FLAG_DIR

I2C General Call Addressing Mode

I2C_GENERALCALL_DISABLE

I2C_GENERALCALL_ENABLE

I2C Interrupt configuration definition

I2C_IT_ERRI

I2C_IT_TCI

I2C_IT_STOPI

I2C_IT_NACKI

I2C_IT_ADDRI

I2C_IT_RXI

I2C_IT_TXI

I2C Memory Address Size

I2C_MEMADD_SIZE_8BIT

I2C_MEMADD_SIZE_16BIT

I2C No-Stretch Mode

I2C_NOSTRETCH_DISABLE

I2C_NOSTRETCH_ENABLE

I2C Own Address2 Masks

I2C_OA2_NOMASK

I2C_OA2_MASK01

I2C_OA2_MASK02

I2C_OA2_MASK03

I2C_OA2_MASK04

I2C_OA2_MASK05

I2C_OA2_MASK06

I2C_OA2_MASK07

I2C Reload End Mode

I2C_RELOAD_MODE

I2C_AUTOEND_MODE

I2C_SOFTEND_MODE

I2C Start or Stop Mode

I2C_NO_STARTSTOP

I2C_GENERATE_STOP

I2C_GENERATE_START_READ

I2C_GENERATE_START_WRITE

I2C Transfer Direction Master Point of View

I2C_DIRECTION_TRANSMIT

I2C_DIRECTION_RECEIVE

I2C Sequential Transfer Options

I2C_FIRST_FRAME

I2C_FIRST_AND_NEXT_FRAME

I2C_NEXT_FRAME

I2C_FIRST_AND_LAST_FRAME

I2C_LAST_FRAME

I2C_LAST_FRAME_NO_STOP

I2C_OTHER_FRAME

I2C_OTHER_AND_LAST_FRAME

34 HAL I2C Extension Driver

34.1 I2CEx Firmware driver API description

The following section lists the various functions of the I2CEx library.

34.1.1 I2C peripheral Extended features

Comparing to other previous devices, the I2C interface for STM32N6xx devices contains the following additional features

- Possibility to disable or enable Analog Noise Filter
- Use of a configured Digital Noise Filter
- Disable or enable wakeup from Stop mode(s)
- Disable or enable Fast Mode Plus

34.1.2 How to use this driver

This driver provides functions to configure Noise Filter and Wake Up Feature

1. Configure I2C Analog noise filter using the function HAL_I2CEx_ConfigAnalogFilter()
2. Configure I2C Digital noise filter using the function HAL_I2CEx_ConfigDigitalFilter()
3. Configure the enable or disable of I2C Wake Up Mode using the functions :
   - HAL_I2CEx_EnableWakeUp()
   - HAL_I2CEx_DisableWakeUp()
4. Configure the enable or disable of fast mode plus driving capability using the functions :
   - HAL_I2CEx_ConfigFastModePlus()

34.1.3 Detailed description of functions

HAL_I2CEx_ConfigAnalogFilter

Function name

HAL_StatusTypeDef HAL_I2CEx_ConfigAnalogFilter (I2C_HandleTypeDef * hi2c, uint32_t AnalogFilter)

Function description

HAL_I2CEx_ConfigDigitalFilter

Function name

HAL_StatusTypeDef HAL_I2CEx_ConfigDigitalFilter (I2C_HandleTypeDef * hi2c, uint32_t DigitalFilter)

Function description

HAL_I2CEx_EnableWakeUp

Function name

HAL_StatusTypeDef HAL_I2CEx_EnableWakeUp (I2C_HandleTypeDef * hi2c)

Function description

HAL_I2CEx_DisableWakeUp

Function name

HAL_StatusTypeDef HAL_I2CEx_DisableWakeUp (I2C_HandleTypeDef * hi2c)

Function description

HAL_I2CEx_ConfigFastModePlus

Function name

HAL_StatusTypeDef HAL_I2CEx_ConfigFastModePlus (I2C_HandleTypeDef * hi2c, uint32_t FastModePlus)

Function description

34.2 I2CEx Firmware driver defines

The following section lists the various define and macros of the module.

34.2.1 I2CEx

I2CEx

I2C Extended Analog Filter

I2C_ANALOGFILTER_ENABLE

I2C_ANALOGFILTER_DISABLE

I2C Extended Fast Mode Plus

I2C_FASTMODEPLUS_ENABLE

Enable Fast Mode Plus

I2C_FASTMODEPLUS_DISABLE

Disable Fast Mode Plus

35 HAL I2S Generic Driver

35.1 I2S Firmware driver registers structures

35.1.1 I2S_InitTypeDef

I2S_InitTypeDef is defined in the stm32n6xx_hal_i2s.h

Data Fields

- ***uint32_t Mode***
- ***uint32_t Standard***
- ***uint32_t DataFormat***
- ***uint32_t MCLKOutput***
- ***uint32_t AudioFreq***
- ***uint32_t CPOL***
- ***uint32_t FirstBit***
- ***uint32_t WSInversion***
- ***uint32_t Data24BitAlignment***
- ***uint32_t MasterKeepIOState***

Field Documentation

- ***uint32_t I2S_InitTypeDef::Mode***
  Specifies the I2S operating mode. This parameter can be a value of ***I2S_Mode***
- ***uint32_t I2S_InitTypeDef::Standard***
  Specifies the standard used for the I2S communication. This parameter can be a value of ***I2S_Standard***
- ***uint32_t I2S_InitTypeDef::DataFormat***
  Specifies the data format for the I2S communication. This parameter can be a value of ***I2S_Data_Format***
- ***uint32_t I2S_InitTypeDef::MCLKOutput***
  Specifies whether the I2S MCLK output is enabled or not. This parameter can be a value of ***I2S_MCLK_Output***
- ***uint32_t I2S_InitTypeDef::AudioFreq***
  Specifies the frequency selected for the I2S communication. This parameter can be a value of ***I2S_Audio_Frequency***
- ***uint32_t I2S_InitTypeDef::CPOL***
  Specifies the idle state of the I2S clock. This parameter can be a value of ***I2S_Clock_Polarity***
- ***uint32_t I2S_InitTypeDef::FirstBit***
  Specifies whether data transfers start from MSB or LSB bit. This parameter can be a value of ***I2S_MSB_LSB_Transmission***
- ***uint32_t I2S_InitTypeDef::WSInversion***
  Control the Word Select Inversion. This parameter can be a value of ***I2S_WSInversion***
- ***uint32_t I2S_InitTypeDef::Data24BitAlignment***
  Specifies the Data Padding for 24 bits data length This parameter can be a value of ***I2S_Data_24Bit_Alignment***
- ***uint32_t I2S_InitTypeDef::MasterKeepIOState***
  Control of Alternate function GPIOs state This parameter can be a value of ***I2S_Master_Keep_IO_State***

35.1.2 __I2S_HandleTypeDef

__I2S_HandleTypeDef is defined in the stm32n6xx_hal_i2s.h

Data Fields

- ***SPI_TypeDef * Instance***
- ***I2S_InitTypeDef Init***
- ***const uint16_t * pTxBuffPtr***
- ***__IO uint16_t TxXferSize***
- ***__IO uint16_t TxXferCount***

- ***uint16_t * pRxBuffPtr***
- ***__IO uint16_t RxXferSize***
- ***__IO uint16_t RxXferCount***
- ***void(* RxISR***
- ***void(* TxISR***
- ***DMA_HandleTypeDef * hdmatx***
- ***DMA_HandleTypeDef * hdmarx***
- ***__IO HAL_LockTypeDef Lock***
- ***__IO HAL_I2S_StateTypeDef State***
- ***__IO uint32_t ErrorCode***
- ***void(* TxCpltCallback***
- ***void(* RxCpltCallback***
- ***void(* TxRxCpltCallback***
- ***void(* TxHalfCpltCallback***
- ***void(* RxHalfCpltCallback***
- ***void(* TxRxHalfCpltCallback***
- ***void(* ErrorCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***SPI_TypeDef* __I2S_HandleTypeDef::Instance***
  I2S registers base address
- ***I2S_InitTypeDef __I2S_HandleTypeDef::Init***
  I2S communication parameters
- ***const uint16_t* __I2S_HandleTypeDef::pTxBuffPtr***
  Pointer to I2S Tx transfer buffer
- ***__IO uint16_t __I2S_HandleTypeDef::TxXferSize***
  I2S Tx transfer size
- ***__IO uint16_t __I2S_HandleTypeDef::TxXferCount***
  I2S Tx transfer Counter
- ***uint16_t* __I2S_HandleTypeDef::pRxBuffPtr***
  Pointer to I2S Rx transfer buffer
- ***__IO uint16_t __I2S_HandleTypeDef::RxXferSize***
  I2S Rx transfer size
- ***__IO uint16_t __I2S_HandleTypeDef::RxXferCount***
  I2S Rx transfer counter (This field is initialized at the same value as transfer size at the beginning of the transfer and decremented when a sample is received NbSamplesReceived = RxBufferSize-RxBufferCount)
- ***void(* __I2S_HandleTypeDef::RxISR)(struct __I2S_HandleTypeDef *hi2s)***
  function pointer on Rx ISR
- ***void(* __I2S_HandleTypeDef::TxISR)(struct __I2S_HandleTypeDef *hi2s)***
  function pointer on Tx ISR
- ***DMA_HandleTypeDef* __I2S_HandleTypeDef::hdmatx***
  I2S Tx DMA handle parameters
- ***DMA_HandleTypeDef* __I2S_HandleTypeDef::hdmarx***
  I2S Rx DMA handle parameters
- ***__IO HAL_LockTypeDef __I2S_HandleTypeDef::Lock***
  I2S locking object
- ***__IO HAL_I2S_StateTypeDef __I2S_HandleTypeDef::State***
  I2S communication state
- ***__IO uint32_t __I2S_HandleTypeDef::ErrorCode***
  I2S Error code This parameter can be a value of ***I2S_Error***
- ***void(* __I2S_HandleTypeDef::TxCpltCallback)(struct __I2S_HandleTypeDef *hi2s)***
  I2S Tx Completed callback

- ***void(* __I2S_HandleTypeDef::RxCpltCallback)(struct __I2S_HandleTypeDef *hi2s)***
  I2S Rx Completed callback
- ***void(* __I2S_HandleTypeDef::TxRxCpltCallback)(struct __I2S_HandleTypeDef *hi2s)***
  I2S TxRx Completed callback
- ***void(* __I2S_HandleTypeDef::TxHalfCpltCallback)(struct __I2S_HandleTypeDef *hi2s)***
  I2S Tx Half Completed callback
- ***void(* __I2S_HandleTypeDef::RxHalfCpltCallback)(struct __I2S_HandleTypeDef *hi2s)***
  I2S Rx Half Completed callback
- ***void(* __I2S_HandleTypeDef::TxRxHalfCpltCallback)(struct __I2S_HandleTypeDef *hi2s)***
  I2S TxRx Half Completed callback
- ***void(* __I2S_HandleTypeDef::ErrorCallback)(struct __I2S_HandleTypeDef *hi2s)***
  I2S Error callback
- ***void(* __I2S_HandleTypeDef::MspInitCallback)(struct __I2S_HandleTypeDef *hi2s)***
  I2S Msp Init callback
- ***void(* __I2S_HandleTypeDef::MspDeInitCallback)(struct __I2S_HandleTypeDef *hi2s)***
  I2S Msp DeInit callback

35.2 I2S Firmware driver API description

The following section lists the various functions of the I2S library.

35.2.1 How to use this driver

The I2S HAL driver can be used as follow:

1. Declare a I2S_HandleTypeDef handle structure.
2. Initialize the I2S low level resources by implement the HAL_I2S_MspInit() API:
   a. Enable the SPIx interface clock.
   b. I2S pins configuration:
      - Enable the clock for the I2S GPIOs.
      - Configure these I2S pins as alternate function pull-up.
   c. NVIC configuration if you need to use interrupt process (HAL_I2S_Transmit_IT() and HAL_I2S_Receive_IT() APIs).
      - Configure the I2Sx interrupt priority.
      - Enable the NVIC I2S IRQ handle.
   d. DMA Configuration if you need to use DMA process (HAL_I2S_Transmit_DMA() and HAL_I2S_Receive_DMA() APIs:
      - Declare a DMA handle structure for the Tx/Rx Stream/Channel.
      - Enable the DMAx interface clock.
      - Configure the declared DMA handle structure with the required Tx/Rx parameters.
      - Configure the DMA Tx/Rx Stream/Channel.
      - Associate the initialized DMA handle to the I2S DMA Tx/Rx handle.
      - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx/Rx Stream/Channel.
3. Program the Mode, Standard, Data Format, MCLK Output, Audio frequency and Polarity using HAL_I2S_Init() function.

Note: The specific I2S interrupts (Transmission complete interrupt, RXNE interrupt and Error Interrupts) will be managed using the macros __HAL_I2S_ENABLE_IT() and __HAL_I2S_DISABLE_IT() inside the transmit and receive process.

– *External clock source is configured after setting correctly the define constant EXTERNAL_CLOCK_VALUE in the stm32n6xx_hal_conf.h file.*

4. Three mode of operations are available within this driver :

Polling mode IO operation

- Send an amount of data in blocking mode using HAL_I2S_Transmit()

- Receive an amount of data in blocking mode using HAL_I2S_Receive()

Interrupt mode IO operation

- Send an amount of data in non blocking mode using HAL_I2S_Transmit_IT()
- At transmission end of half transfer HAL_I2S_TxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2S_TxHalfCpltCallback
- At transmission end of transfer HAL_I2S_TxCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2S_TxCpltCallback
- Receive an amount of data in non blocking mode using HAL_I2S_Receive_IT()
- At reception end of half transfer HAL_I2S_RxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2S_RxHalfCpltCallback
- At reception end of transfer HAL_I2S_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2S_RxCpltCallback
- In case of transfer Error, HAL_I2S_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_I2S_ErrorCallback

DMA mode IO operation

- Send an amount of data in non blocking mode (DMA) using HAL_I2S_Transmit_DMA()
- At transmission end of half transfer HAL_I2S_TxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2S_TxHalfCpltCallback
- At transmission end of transfer HAL_I2S_TxCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2S_TxCpltCallback
- Receive an amount of data in non blocking mode (DMA) using HAL_I2S_Receive_DMA()
- At reception end of half transfer HAL_I2S_RxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2S_RxHalfCpltCallback
- At reception end of transfer HAL_I2S_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2S_RxCpltCallback
- In case of transfer Error, HAL_I2S_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_I2S_ErrorCallback
- Pause the DMA Transfer using HAL_I2S_DMAPause()
- Resume the DMA Transfer using HAL_I2S_DMAResume()
- Stop the DMA Transfer using HAL_I2S_DMAStop()

I2S HAL driver macros list

Below the list of most used macros in I2S HAL driver.

- __HAL_I2S_ENABLE: Enable the specified SPI peripheral (in I2S mode)
- __HAL_I2S_DISABLE: Disable the specified SPI peripheral (in I2S mode)
- __HAL_I2S_ENABLE_IT : Enable the specified I2S interrupts
- __HAL_I2S_DISABLE_IT : Disable the specified I2S interrupts
- __HAL_I2S_GET_FLAG: Check whether the specified I2S flag is set or not

Note: You can refer to the I2S HAL driver header file for more useful macros

I2S HAL driver macros list

Callback registration:

1. The compilation flag USE_HAL_I2S_REGISTER_CALLBACKS when set to 1UL allows the user to configure dynamically the driver callbacks. Use Functions HAL_I2S_RegisterCallback() to register an interrupt callback.

2. Function HAL_I2S_RegisterCallback() allows to register following callbacks.
   - TxCpltCallback : I2S Tx Completed callback
   - RxCpltCallback : I2S Rx Completed callback
   - TxRxCpltCallback : I2S TxRx Completed callback
   - TxHalfCpltCallback : I2S Tx Half Completed callback
   - RxHalfCpltCallback : I2S Rx Half Completed callback
   - TxRxHalfCpltCallback : I2S TxRx Half Completed callback
   - ErrorCallback : I2S Error callback
   - MspInitCallback : I2S Msp Init callback
   - MspDeInitCallback : I2S Msp DeInit callback This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.
3. Use function HAL_I2S_UnRegisterCallback to reset a callback to the default weak function. HAL_I2S_UnRegisterCallback takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks:
   - TxCpltCallback : I2S Tx Completed callback
   - RxCpltCallback : I2S Rx Completed callback
   - TxRxCpltCallback : I2S TxRx Completed callback
   - TxHalfCpltCallback : I2S Tx Half Completed callback
   - RxHalfCpltCallback : I2S Rx Half Completed callback
   - TxRxHalfCpltCallback : I2S TxRx Half Completed callback
   - ErrorCallback : I2S Error callback
   - MspInitCallback : I2S Msp Init callback
   - MspDeInitCallback : I2S Msp DeInit callback
4. By default, after the HAL_I2S_Init() and when the state is HAL_I2S_STATE_RESET all callbacks are set to the corresponding weak functions: examples HAL_I2S_MasterTxCpltCallback(), HAL_I2S_MasterRxCpltCallback(). Exception done for MspInit and MspDeInit functions that are reset to the legacy weak functions in the HAL_I2S_Init()/ HAL_I2S_DeInit() only when these callbacks are null (not registered beforehand). If MspInit or MspDeInit are not null, the HAL_I2S_Init()/ HAL_I2S_DeInit() keep and use the user MspInit/MspDeInit callbacks (registered beforehand) whatever the state.
5. Callbacks can be registered/unregistered in HAL_I2S_STATE_READY state only. Exception done MspInit/ MspDeInit functions that can be registered/unregistered in HAL_I2S_STATE_READY or HAL_I2S_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/ DeInit. Then, the user first registers the MspInit/MspDeInit user callbacks using HAL_I2S_RegisterCallback() before calling HAL_I2S_DeInit() or HAL_I2S_Init() function.
6. When The compilation define USE_HAL_I2S_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak callbacks are used.

35.2.2 Detailed description of functions

HAL_I2S_Init

Function name

HAL_StatusTypeDef HAL_I2S_Init (I2S_HandleTypeDef * hi2s)

Function description

HAL_I2S_DeInit

Function name

HAL_StatusTypeDef HAL_I2S_DeInit (I2S_HandleTypeDef * hi2s)

Function description

HAL_I2S_MspInit

Function name

void HAL_I2S_MspInit (I2S_HandleTypeDef * hi2s)

Function description

HAL_I2S_MspDeInit

Function name

void HAL_I2S_MspDeInit (I2S_HandleTypeDef * hi2s)

Function description

HAL_I2S_RegisterCallback

Function name

HAL_StatusTypeDef HAL_I2S_RegisterCallback (I2S_HandleTypeDef * hi2s, HAL_I2S_CallbackIDTypeDef CallbackID, pI2S_CallbackTypeDef pCallback)

Function description

HAL_I2S_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_I2S_UnRegisterCallback (I2S_HandleTypeDef * hi2s, HAL_I2S_CallbackIDTypeDef CallbackID)

Function description

HAL_I2S_Transmit

Function name

HAL_StatusTypeDef HAL_I2S_Transmit (I2S_HandleTypeDef * hi2s, const uint16_t * pData, uint16_t Size, uint32_t Timeout)

Function description

HAL_I2S_Receive

Function name

HAL_StatusTypeDef HAL_I2S_Receive (I2S_HandleTypeDef * hi2s, uint16_t * pData, uint16_t Size, uint32_t Timeout)

Function description

HAL_I2SEx_TransmitReceive

Function name

HAL_StatusTypeDef HAL_I2SEx_TransmitReceive (I2S_HandleTypeDef * hi2s, const uint16_t * pTxData, uint16_t * pRxData, uint16_t Size, uint32_t Timeout)

Function description

HAL_I2S_Transmit_IT

Function name

HAL_StatusTypeDef HAL_I2S_Transmit_IT (I2S_HandleTypeDef * hi2s, const uint16_t * pData, uint16_t Size)

Function description

HAL_I2S_Receive_IT

Function name

HAL_StatusTypeDef HAL_I2S_Receive_IT (I2S_HandleTypeDef * hi2s, uint16_t * pData, uint16_t Size)

Function description

HAL_I2SEx_TransmitReceive_IT

Function name

HAL_StatusTypeDef HAL_I2SEx_TransmitReceive_IT (I2S_HandleTypeDef * hi2s, const uint16_t * pTxData, uint16_t * pRxData, uint16_t Size)

Function description

HAL_I2S_IRQHandler

Function name

void HAL_I2S_IRQHandler (I2S_HandleTypeDef * hi2s)

Function description

HAL_I2S_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_I2S_Transmit_DMA (I2S_HandleTypeDef * hi2s, const uint16_t * pData, uint16_t Size)

Function description

HAL_I2S_Receive_DMA

Function name

HAL_StatusTypeDef HAL_I2S_Receive_DMA (I2S_HandleTypeDef * hi2s, uint16_t * pData, uint16_t Size)

Function description

HAL_I2SEx_TransmitReceive_DMA

Function name

HAL_StatusTypeDef HAL_I2SEx_TransmitReceive_DMA (I2S_HandleTypeDef * hi2s, const uint16_t * pTxData, uint16_t * pRxData, uint16_t Size)

Function description

HAL_I2S_DMAPause

Function name

HAL_StatusTypeDef HAL_I2S_DMAPause (I2S_HandleTypeDef * hi2s)

Function description

HAL_I2S_DMAResume

Function name

HAL_StatusTypeDef HAL_I2S_DMAResume (I2S_HandleTypeDef * hi2s)

Function description

HAL_I2S_DMAStop

Function name

HAL_StatusTypeDef HAL_I2S_DMAStop (I2S_HandleTypeDef * hi2s)

Function description

HAL_I2S_TxHalfCpltCallback

Function name

void HAL_I2S_TxHalfCpltCallback (I2S_HandleTypeDef * hi2s)

Function description

HAL_I2S_TxCpltCallback

Function name

void HAL_I2S_TxCpltCallback (I2S_HandleTypeDef * hi2s)

Function description

HAL_I2S_RxHalfCpltCallback

Function name

void HAL_I2S_RxHalfCpltCallback (I2S_HandleTypeDef * hi2s)

Function description

HAL_I2S_RxCpltCallback

Function name

void HAL_I2S_RxCpltCallback (I2S_HandleTypeDef * hi2s)

Function description

HAL_I2SEx_TxRxHalfCpltCallback

Function name

void HAL_I2SEx_TxRxHalfCpltCallback (I2S_HandleTypeDef * hi2s)

Function description

HAL_I2SEx_TxRxCpltCallback

Function name

void HAL_I2SEx_TxRxCpltCallback (I2S_HandleTypeDef * hi2s)

Function description

HAL_I2S_ErrorCallback

Function name

void HAL_I2S_ErrorCallback (I2S_HandleTypeDef * hi2s)

Function description

HAL_I2S_GetState

Function name

HAL_I2S_StateTypeDef HAL_I2S_GetState (const I2S_HandleTypeDef * hi2s)

Function description

HAL_I2S_GetError

Function name

uint32_t HAL_I2S_GetError (const I2S_HandleTypeDef * hi2s)

Function description

35.3 I2S Firmware driver defines

The following section lists the various define and macros of the module.

35.3.1 I2S

I2S

I2S Audio Frequency

I2S_AUDIOFREQ_192K

I2S_AUDIOFREQ_96K

I2S_AUDIOFREQ_48K

I2S_AUDIOFREQ_44K

I2S_AUDIOFREQ_32K

I2S_AUDIOFREQ_22K

I2S_AUDIOFREQ_16K

I2S_AUDIOFREQ_11K

I2S_AUDIOFREQ_8K

I2S_AUDIOFREQ_DEFAULT

I2S FullDuplex Mode

I2S_CPOL_LOW

I2S_CPOL_HIGH

Data Padding 24Bit

I2S_DATA_24BIT_ALIGNMENT_RIGHT

I2S_DATA_24BIT_ALIGNMENT_LEFT

I2S Data Format

I2S_DATAFORMAT_16B

I2S_DATAFORMAT_16B_EXTENDED

I2S_DATAFORMAT_24B

I2S_DATAFORMAT_32B

I2S Error

HAL_I2S_ERROR_NONE

No error

HAL_I2S_ERROR_TIMEOUT

Timeout error

HAL_I2S_ERROR_OVR

OVR error

HAL_I2S_ERROR_UDR

UDR error

HAL_I2S_ERROR_DMA

DMA transfer error

HAL_I2S_ERROR_PRESCALER

Prescaler Calculation error

HAL_I2S_ERROR_FRE

FRE error

HAL_I2S_ERROR_NO_OGT

No On Going Transfer error

HAL_I2S_ERROR_NOT_SUPPORTED

Requested operation not supported

HAL_I2S_ERROR_INVALID_CALLBACK

Invalid Callback error

I2S Exported Macros

__HAL_I2S_RESET_HANDLE_STATE

Description:

- Reset I2S handle state.

Parameters:

- __HANDLE__: specifies the I2S Handle.

Return value:

- None

__HAL_I2S_ENABLE

Description:

- Enable the specified SPI peripheral (in I2S mode).

Parameters:

- __HANDLE__: specifies the I2S Handle.

Return value:

- None

__HAL_I2S_DISABLE

Description:

- Disable the specified SPI peripheral (in I2S mode).

Parameters:

- __HANDLE__: specifies the I2S Handle.

Return value:

- None

__HAL_I2S_ENABLE_IT

Description:

- Enable the specified I2S interrupts.

Parameters:

- __HANDLE__: specifies the I2S Handle. This parameter can be I2S where x: 1, 2 or 3 to select the I2S peripheral.
- __INTERRUPT__: specifies the interrupt source to enable or disable. This parameter can be one of the following values:
  - I2S_IT_RXP : Rx-Packet available interrupt
  - I2S_IT_TXP : Tx-Packet space available interrupt
  - I2S_IT_UDR : Underrun interrupt
  - I2S_IT_OVR : Overrun interrupt
  - I2S_IT_FRE : TI mode frame format error interrupt
  - I2S_IT_ERR : Error interrupt enable

Return value:

- None

__HAL_I2S_DISABLE_IT

Description:

- Disable the specified I2S interrupts.

Parameters:

- __HANDLE__: specifies the I2S Handle. This parameter can be I2S where x: 1, 2 or 3 to select the I2S peripheral.
- __INTERRUPT__: specifies the interrupt source to enable or disable. This parameter can be one of the following values:
  - I2S_IT_RXP : Rx-Packet available interrupt
  - I2S_IT_TXP : Tx-Packet space available interrupt
  - I2S_IT_UDR : Underrun interrupt
  - I2S_IT_OVR : Overrun interrupt
  - I2S_IT_FRE : TI mode frame format error interrupt
  - I2S_IT_ERR : Error interrupt enable

Return value:

- None

__HAL_I2S_GET_IT_SOURCE

Description:

- Check if the specified I2S interrupt source is enabled or disabled.

Parameters:

- __HANDLE__: specifies the I2S Handle. This parameter can be I2S where x: 1, 2 or 3 to select the I2S peripheral.
- __INTERRUPT__: specifies the I2S interrupt source to check. This parameter can be one of the following values:
  - I2S_IT_RXP : Rx-Packet available interrupt
  - I2S_IT_TXP : Tx-Packet space available interrupt
  - I2S_IT_DXP : Tx-Packet space available interrupt
  - I2S_IT_UDR : Underrun interrupt
  - I2S_IT_OVR : Overrun interrupt
  - I2S_IT_FRE : TI mode frame format error interrupt
  - I2S_IT_ERR : Error interrupt enable

Return value:

- The: new state of __IT__ (TRUE or FALSE).

__HAL_I2S_GET_FLAG

Description:

- Check whether the specified I2S flag is set or not.

Parameters:

- __HANDLE__: specifies the I2S Handle. This parameter can be I2S where x: 1, 2 or 3 to select the I2S peripheral.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - I2S_FLAG_RXP : Rx-Packet available flag
  - I2S_FLAG_TXP : Tx-Packet space available flag
  - I2S_FLAG_UDR : Underrun flag
  - I2S_FLAG_OVR : Overrun flag
  - I2S_FLAG_FRE : TI mode frame format error flag

Return value:

- The: new state of __FLAG__ (TRUE or FALSE).

__HAL_I2S_CLEAR_OVRFLAG

Description:

- Clear the I2S OVR pending flag.

Parameters:

- __HANDLE__: specifies the I2S Handle.

Return value:

- None

__HAL_I2S_CLEAR_UDRFLAG

Description:

- Clear the I2S UDR pending flag.

Parameters:

- __HANDLE__: specifies the I2S Handle.

Return value:

- None

__HAL_I2S_CLEAR_TIFREFLAG

Description:

- Clear the I2S FRE pending flag.

Parameters:

- __HANDLE__: specifies the I2S Handle.

Return value:

- None

I2S Flags Definition

I2S_FLAG_RXP

I2S_FLAG_TXP

I2S_FLAG_DXP

I2S_FLAG_UDR

I2S_FLAG_OVR

I2S_FLAG_FRE

I2S_FLAG_MASK

I2S Interrupts Definition

I2S_IT_RXP

I2S_IT_TXP

I2S_IT_DXP

I2S_IT_UDR

I2S_IT_OVR

I2S_IT_FRE

I2S_IT_ERR

Keep IO State

I2S_MASTER_KEEP_IO_STATE_DISABLE

I2S_MASTER_KEEP_IO_STATE_ENABLE

I2S MCLK Output

I2S_MCLKOUTPUT_ENABLE

I2S_MCLKOUTPUT_DISABLE

I2S Mode

I2S_MODE_SLAVE_TX

I2S_MODE_SLAVE_RX

I2S_MODE_MASTER_TX

I2S_MODE_MASTER_RX

I2S_MODE_SLAVE_FULLDUPLEX

I2S_MODE_MASTER_FULLDUPLEX

I2S MSB LSB Transmission

I2S_FIRSTBIT_MSB

I2S_FIRSTBIT_LSB

I2S Standard

I2S_STANDARD_PHILIPS

I2S_STANDARD_MSB

I2S_STANDARD_LSB

I2S_STANDARD_PCM_SHORT

I2S_STANDARD_PCM_LONG

I2S Word Select Inversion

I2S_WS_INVERSION_DISABLE

I2S_WS_INVERSION_ENABLE

36 HAL I3C Generic Driver

36.1 I3C Firmware driver registers structures

36.1.1 I3C_InitTypeDef

I3C_InitTypeDef is defined in the stm32n6xx_hal_i3c.h

Data Fields

- ***LL_I3C_CtrlBusConfTypeDef CtrlBusCharacteristic***
- ***LL_I3C_TgtBusConfTypeDef TgtBusCharacteristic***

Field Documentation

- ***LL_I3C_CtrlBusConfTypeDef I3C_InitTypeDef::CtrlBusCharacteristic***
  Specifies the I3C controller bus characteristic configuration when Controller mode
- ***LL_I3C_TgtBusConfTypeDef I3C_InitTypeDef::TgtBusCharacteristic***
  Specifies the I3C target bus characteristic configuration when Target mode

36.1.2 I3C_FifoConfTypeDef

I3C_FifoConfTypeDef is defined in the stm32n6xx_hal_i3c.h

Data Fields

- ***uint32_t RxFifoThreshold***
- ***uint32_t TxFifoThreshold***
- ***uint32_t ControlFifo***
- ***uint32_t StatusFifo***

Field Documentation

- ***uint32_t I3C_FifoConfTypeDef::RxFifoThreshold***
  Specifies the I3C Rx FIFO threshold level. This parameter must be a value of ***I3C_RX_FIFO_THRESHOLD***
- ***uint32_t I3C_FifoConfTypeDef::TxFifoThreshold***
  Specifies the I3C Tx FIFO threshold level. This parameter must be a value of ***I3C_TX_FIFO_THRESHOLD***
- ***uint32_t I3C_FifoConfTypeDef::ControlFifo***
  Specifies the I3C control FIFO enable/disable state. This parameter is configured only with controller mode and it must be a value of ***I3C_CONTROL_FIFO_STATE***
- ***uint32_t I3C_FifoConfTypeDef::StatusFifo***
  Specifies the I3C status FIFO enable/disable state. This parameter is configured only with controller mode and it must be a value of ***I3C_STATUS_FIFO_STATE***

36.1.3 I3C_CtrlConfTypeDef

I3C_CtrlConfTypeDef is defined in the stm32n6xx_hal_i3c.h

Data Fields

- ***uint8_t DynamicAddr***
- ***uint8_t StallTime***
- ***FunctionalState HotJoinAllowed***
- ***FunctionalState ACKStallState***
- ***FunctionalState CCCStallState***
- ***FunctionalState TxStallState***
- ***FunctionalState RxStallState***
- ***FunctionalState HighKeeperSDA***

Field Documentation

- ***uint8_t I3C_CtrlConfTypeDef::DynamicAddr***
  Specifies the dynamic address of the controller when goes in target mode. This parameter must be a number between Min_Data=0x00 and Max_Data=0x7F

- ***uint8_t I3C_CtrlConfTypeDef::StallTime***
  Specifies the controller clock stall time in number of kernel clock cycles. This parameter must be a number between Min_Data=0x00 and Max_Data=0xFF
- ***FunctionalState I3C_CtrlConfTypeDef::HotJoinAllowed***
  Specifies the Enable/Disable state of the controller Hot Join acknowledgement when receiving a hot join request from target. This parameter can be set to ENABLE or DISABLE
- ***FunctionalState I3C_CtrlConfTypeDef::ACKStallState***
  Specifies the Enable/Disable state of the controller clock stall on the ACK phase. This parameter can be set to ENABLE or DISABLE
- ***FunctionalState I3C_CtrlConfTypeDef::CCCStallState***
  Specifies the Enable/Disable state of the controller clock stall on the T bit phase of a CCC communication to allow the target to decode command. This parameter can be set to ENABLE or DISABLE
- ***FunctionalState I3C_CtrlConfTypeDef::TxStallState***
  Specifies the Enable/Disable state of the controller clock stall on parity phase of data to allow the target to read received data. This parameter can be set to ENABLE or DISABLE
- ***FunctionalState I3C_CtrlConfTypeDef::RxStallState***
  Specifies the Enable/Disable state of the controller clock stall on the T bit phase of data enable to allow the target to prepare data to be sent. This parameter can be set to ENABLE or DISABLE
- ***FunctionalState I3C_CtrlConfTypeDef::HighKeeperSDA***
  Specifies the Enable/Disable state of the controller SDA high keeper. This parameter can be set to ENABLE or DISABLE

36.1.4 I3C_TgtConfTypeDef

I3C_TgtConfTypeDef is defined in the stm32n6xx_hal_i3c.h

Data Fields

- ***uint8_t Identifier***
- ***uint8_t MIPIIdentifier***
- ***FunctionalState CtrlRoleRequest***
- ***FunctionalState HotJoinRequest***
- ***FunctionalState IBIRequest***
- ***FunctionalState IBIPayload***
- ***uint32_t IBIPayloadSize***
- ***uint16_t MaxReadDataSize***
- ***uint16_t MaxWriteDataSize***
- ***FunctionalState CtrlCapability***
- ***FunctionalState GroupAddrCapability***
- ***uint32_t DataTurnAroundDuration***
- ***uint8_t MaxReadTurnAround***
- ***uint32_t MaxDataSpeed***
- ***FunctionalState MaxSpeedLimitation***
- ***uint32_t HandOffActivityState***
- ***FunctionalState HandOffDelay***
- ***FunctionalState PendingReadMDB***

Field Documentation

- ***uint8_t I3C_TgtConfTypeDef::Identifier***
  Specifies the target characteristic ID (MIPI named reference DCR). This parameter must be a number between Min_Data=0x00 and Max_Data=0xFF
- ***uint8_t I3C_TgtConfTypeDef::MIPIIdentifier***
  Specifies the bits [12-15] of the 48-provisioned ID (MIPI named reference PID), other 48-provisioned ID are hardcoded. This parameter must be a number between Min_Data=0x00 and Max_Data=0x0F
- ***FunctionalState I3C_TgtConfTypeDef::CtrlRoleRequest***
  Specifies the Enable/Disable state of the target authorization request for a second master Chip. This parameter can be set to ENABLE or DISABLE

- ***FunctionalState I3C_TgtConfTypeDef::HotJoinRequest***
  Specifies the Enable/Disable state of the target hot join authorization request. This parameter can be set to ENABLE or DISABLE
- ***FunctionalState I3C_TgtConfTypeDef::IBIRequest***
  Specifies the Enable/Disable state of the target in Band Interrupt authorization request. This parameter can be set to ENABLE or DISABLE
- ***FunctionalState I3C_TgtConfTypeDef::IBIPayload***
  Specifies the Enable/Disable state of sending data payload after an accepted IBI. This parameter can be set to ENABLE or DISABLE
- ***uint32_t I3C_TgtConfTypeDef::IBIPayloadSize***
  Specifies the I3C target payload data size. This parameter must be a value of ***I3C_PAYLOAD_SIZE***
- ***uint16_t I3C_TgtConfTypeDef::MaxReadDataSize***
  Specifies the numbers of data bytes that the target can read at maximum. This parameter must be a number between Min_Data=0x00 and Max_Data=0xFFFF
- ***uint16_t I3C_TgtConfTypeDef::MaxWriteDataSize***
  Specifies the numbers of data bytes that the target can write at maximum. This parameter must be a number between Min_Data=0x00 and Max_Data=0xFFFF
- ***FunctionalState I3C_TgtConfTypeDef::CtrlCapability***
  Specifies the Enable/Disable state of the target controller capability. This parameter can be set to ENABLE or DISABLE
- ***FunctionalState I3C_TgtConfTypeDef::GroupAddrCapability***
  Specifies the Enable/Disable state of the target support of group address after a controller role hand-off. This parameter can be set to ENABLE or DISABLE
- ***uint32_t I3C_TgtConfTypeDef::DataTurnAroundDuration***
  Specifies the I3C target clock-to-data turnaround time. This parameter must be a value of ***I3C_TURNAROUND_TIME_TSCO***
- ***uint8_t I3C_TgtConfTypeDef::MaxReadTurnAround***
  Specifies the target maximum read turnaround byte. This parameter must be a number between Min_Data=0x00 and Max_Data=0xFF
- ***uint32_t I3C_TgtConfTypeDef::MaxDataSpeed***
  Specifies the I3C target returned GETMXDS CCC format. This parameter must be a value of ***I3C_GETMXDS_FORMAT***
- ***FunctionalState I3C_TgtConfTypeDef::MaxSpeedLimitation***
  Specifies the Enable/Disable state of the target max data speed limitation. This parameter can be set to ENABLE or DISABLE
- ***uint32_t I3C_TgtConfTypeDef::HandOffActivityState***
  Specifies the I3C target activity state when becoming controller. This parameter must be a value of ***I3C_HANDOFF_ACTIVITY_STATE***
- ***FunctionalState I3C_TgtConfTypeDef::HandOffDelay***
  Specifies the Enable/Disable state of the target need of delay to process the controller role hand-off. This parameter can be set to ENABLE or DISABLE
- ***FunctionalState I3C_TgtConfTypeDef::PendingReadMDB***
  Specifies the Enable/Disable state of the transmission of a mandatory data bytes indicating a pending read notification for GETCAPR CCC command. This parameter can be set to ENABLE or DISABLE

36.1.5 I3C_DeviceConfTypeDef

I3C_DeviceConfTypeDef is defined in the stm32n6xx_hal_i3c.h

Data Fields

- ***uint8_t DeviceIndex***
- ***uint8_t TargetDynamicAddr***
- ***FunctionalState IBIAck***
- ***FunctionalState IBIPayload***
- ***FunctionalState CtrlRoleReqAck***
- ***FunctionalState CtrlStopTransfer***

Field Documentation

- ***uint8_t I3C_DeviceConfTypeDef::DeviceIndex***
  Specifies the index value of the device in the DEVRx register. This parameter must be a number between Min_Data=1 and Max_Data=4
- ***uint8_t I3C_DeviceConfTypeDef::TargetDynamicAddr***
  Specifies the dynamic address of the target x (1 to 4) connected on the bus. This parameter must be a number between Min_Data=0x00 and Max_Data=0x7F
- ***FunctionalState I3C_DeviceConfTypeDef::IBIAck***
  Specifies the Enable/Disable state of the controller's ACK when receiving an IBI from a target x (1 to 4) connected on the bus. This parameter can be set to ENABLE or DISABLE
- ***FunctionalState I3C_DeviceConfTypeDef::IBIPayload***
  Specifies the Enable/Disable state of the controller's receiving IBI payload after acknowledging an IBI requested from a target x (1 to 4) connected on the bus. This parameter can be set to ENABLE or DISABLE
- ***FunctionalState I3C_DeviceConfTypeDef::CtrlRoleReqAck***
  Specifies the Enable/Disable state of the controller's ACK when receiving a control request from a target x (1 to 4) connected on the bus. This parameter can be set to ENABLE or DISABLE
- ***FunctionalState I3C_DeviceConfTypeDef::CtrlStopTransfer***
  Specifies the Enable/Disable state of the controller's stop transfer after receiving an IBI request from a target x (1 to 4) connected on the bus. This parameter can be set to ENABLE or DISABLE

36.1.6 I3C_CCCInfoTypeDef

I3C_CCCInfoTypeDef is defined in the stm32n6xx_hal_i3c.h

Data Fields

- ***uint32_t DynamicAddrValid***
- ***uint32_t DynamicAddr***
- ***uint32_t MaxWriteLength***
- ***uint32_t MaxReadLength***
- ***uint32_t ResetAction***
- ***uint32_t ActivityState***
- ***uint32_t HotJoinAllowed***
- ***uint32_t InBandAllowed***
- ***uint32_t CtrlRoleAllowed***
- ***uint32_t IBICRTgtAddr***
- ***uint32_t IBITgtNbPayload***
- ***uint32_t IBITgtPayload***

Field Documentation

- ***uint32_t I3C_CCCInfoTypeDef::DynamicAddrValid***
  I3C target Dynamic Address Valid (updated during ENTDAA/RSTDAA/SETNEWDA CCC) This parameter can be Valid=1U or Not Valid=0U
- ***uint32_t I3C_CCCInfoTypeDef::DynamicAddr***
  I3C target Dynamic Address (updated during ENTDAA/RSTDAA/SETNEWDA CCC)
- ***uint32_t I3C_CCCInfoTypeDef::MaxWriteLength***
  I3C target Maximum Write Length (updated during SETMWL CCC)
- ***uint32_t I3C_CCCInfoTypeDef::MaxReadLength***
  I3C target Maximum Read Length (updated during SETMRL CCC)
- ***uint32_t I3C_CCCInfoTypeDef::ResetAction***
  I3C target Reset Action level (updated during RSTACT CCC)
- ***uint32_t I3C_CCCInfoTypeDef::ActivityState***
  I3C target Activity State (updated during ENTASx CCC)
- ***uint32_t I3C_CCCInfoTypeDef::HotJoinAllowed***
  I3C target Hot Join (updated during ENEC/DISEC CCC) This parameter can be Allowed=1U or Not Allowed=0U
- ***uint32_t I3C_CCCInfoTypeDef::InBandAllowed***
  I3C target In Band Interrupt (updated during ENEC/DISEC CCC) This parameter can be Allowed=1U or Not Allowed=0U

- ***uint32_t I3C_CCCInfoTypeDef::CtrlRoleAllowed***
  I3C target Controller Role Request (updated during ENEC/DISEC CCC) This parameter can be Allowed=1U or Not Allowed=0U
- ***uint32_t I3C_CCCInfoTypeDef::IBICRTgtAddr***
  I3C controller receive Target Address during IBI or Controller Role Request event
- ***uint32_t I3C_CCCInfoTypeDef::IBITgtNbPayload***
  I3C controller get Number of Data Payload after an IBI event
- ***uint32_t I3C_CCCInfoTypeDef::IBITgtPayload***
  I3C controller receive IBI Payload after an IBI event

36.1.7 I3C_ControlTypeDef

I3C_ControlTypeDef is defined in the stm32n6xx_hal_i3c.h

Data Fields

- ***uint32_t * pBuffer***
- ***uint32_t Size***

Field Documentation

- ***uint32_t* I3C_ControlTypeDef::pBuffer***
  Pointer to the buffer containing the control or status register values
- ***uint32_t I3C_ControlTypeDef::Size***
  The size of pBuffer in words

36.1.8 I3C_DataTypeDef

I3C_DataTypeDef is defined in the stm32n6xx_hal_i3c.h

Data Fields

- ***uint8_t * pBuffer***
- ***uint32_t Size***

Field Documentation

- ***uint8_t* I3C_DataTypeDef::pBuffer***
  Pointer to the buffer containing all data values to transfer
- ***uint32_t I3C_DataTypeDef::Size***
  The size of pBuffer in bytes

36.1.9 I3C_CCCTypeDef

I3C_CCCTypeDef is defined in the stm32n6xx_hal_i3c.h

Data Fields

- ***uint8_t TargetAddr***
- ***uint8_t CCC***
- ***I3C_DataTypeDef CCCBuf***
- ***uint32_t Direction***

Field Documentation

- ***uint8_t I3C_CCCTypeDef::TargetAddr***
  Dynamic or Static target Address
- ***uint8_t I3C_CCCTypeDef::CCC***
  CCC value code
- ***I3C_DataTypeDef I3C_CCCTypeDef::CCCBuf***
  Contain size of associated data and size of defining byte if any. Contain pointer to CCC associated data
- ***uint32_t I3C_CCCTypeDef::Direction***
  CCC read and/or write direction message

36.1.10 I3C_BCRTypeDef

I3C_BCRTypeDef is defined in the stm32n6xx_hal_i3c.h

Data Fields

- ***FunctionalState MaxDataSpeedLimitation***
- ***FunctionalState IBIRequestCapable***
- ***FunctionalState IBIPayload***
- ***FunctionalState OfflineCapable***
- ***FunctionalState VirtualTargetSupport***
- ***FunctionalState AdvancedCapabilities***
- ***FunctionalState DeviceRole***

Field Documentation

- ***FunctionalState I3C_BCRTypeDef::MaxDataSpeedLimitation***
  Max data speed limitation
- ***FunctionalState I3C_BCRTypeDef::IBIRequestCapable***
  IBI request capable
- ***FunctionalState I3C_BCRTypeDef::IBIPayload***
  IBI payload data
- ***FunctionalState I3C_BCRTypeDef::OfflineCapable***
  Offline capable
- ***FunctionalState I3C_BCRTypeDef::VirtualTargetSupport***
  Virtual target support
- ***FunctionalState I3C_BCRTypeDef::AdvancedCapabilities***
  Advanced capabilities
- ***FunctionalState I3C_BCRTypeDef::DeviceRole***
  Device role

36.1.11 I3C_PIDTypeDef

I3C_PIDTypeDef is defined in the stm32n6xx_hal_i3c.h

Data Fields

- ***uint16_t MIPIMID***
- ***uint8_t IDTSEL***
- ***uint16_t PartID***
- ***uint8_t MIPIID***

Field Documentation

- ***uint16_t I3C_PIDTypeDef::MIPIMID***
  MIPI Manufacturer ID
- ***uint8_t I3C_PIDTypeDef::IDTSEL***
  Provisioned ID Type Selector
- ***uint16_t I3C_PIDTypeDef::PartID***
  Part ID device vendor to define
- ***uint8_t I3C_PIDTypeDef::MIPIID***
  Instance ID

36.1.12 I3C_ENTDAAPayloadTypeDef

I3C_ENTDAAPayloadTypeDef is defined in the stm32n6xx_hal_i3c.h

Data Fields

- ***I3C_BCRTypeDef BCR***
- ***uint32_t DCR***
- ***I3C_PIDTypeDef PID***

Field Documentation

- ***I3C_BCRTypeDef I3C_ENTDAAPayloadTypeDef::BCR***
  Bus Characteristics Register
- ***uint32_t I3C_ENTDAAPayloadTypeDef::DCR***
  Device Characteristics Register

- ***I3C_PIDTypeDef I3C_ENTDAAPayloadTypeDef::PID***
  Provisioned ID

36.1.13 I3C_PrivateTypeDef

I3C_PrivateTypeDef is defined in the stm32n6xx_hal_i3c.h

Data Fields

- ***uint8_t TargetAddr***
- ***I3C_DataTypeDef TxBuf***
- ***I3C_DataTypeDef RxBuf***
- ***uint32_t Direction***

Field Documentation

- ***uint8_t I3C_PrivateTypeDef::TargetAddr***
  Dynamic or Static target Address
- ***I3C_DataTypeDef I3C_PrivateTypeDef::TxBuf***
  Buffer structure containing the data to transmit (little endian)
- ***I3C_DataTypeDef I3C_PrivateTypeDef::RxBuf***
  Buffer structure containing the data to receive (little endian)
- ***uint32_t I3C_PrivateTypeDef::Direction***
  Read and/or write message

36.1.14 I3C_XferTypeDef

I3C_XferTypeDef is defined in the stm32n6xx_hal_i3c.h

Data Fields

- ***I3C_ControlTypeDef CtrlBuf***
- ***I3C_ControlTypeDef StatusBuf***
- ***I3C_DataTypeDef TxBuf***
- ***I3C_DataTypeDef RxBuf***

Field Documentation

- ***I3C_ControlTypeDef I3C_XferTypeDef::CtrlBuf***
  Buffer structure containing the control register values
- ***I3C_ControlTypeDef I3C_XferTypeDef::StatusBuf***
  Buffer structure containing the status register values
- ***I3C_DataTypeDef I3C_XferTypeDef::TxBuf***
  Buffer structure containing the data to transmit
- ***I3C_DataTypeDef I3C_XferTypeDef::RxBuf***
  Buffer structure containing the data to receive

36.1.15 __I3C_HandleTypeDef

__I3C_HandleTypeDef is defined in the stm32n6xx_hal_i3c.h

Data Fields

- ***I3C_TypeDef * Instance***
- ***I3C_InitTypeDef Init***
- ***HAL_I3C_ModeTypeDef Mode***
- ***I3C_XferTypeDef * pXferData***
- ***const I3C_CCCTypeDef * pCCCDesc***
- ***const I3C_PrivateTypeDef * pPrivateDesc***
- ***uint32_t ControlXferCount***
- ***uint32_t RxXferCount***
- ***uint32_t TxXferCount***
- ***DMA_HandleTypeDef * hdmacr***
- ***DMA_HandleTypeDef * hdmatx***
- ***DMA_HandleTypeDef * hdmarx***

- *DMA_HandleTypeDef * hdmasr*
- *HAL_LockTypeDef Lock*
- *__IO HAL_I3C_StateTypeDef State*
- *__IO HAL_I3C_StateTypeDef PreviousState*
- *__IO uint32_t ErrorCode*
- *HAL_StatusTypeDef(* XferISR*
- *void(* ptrTxFunc*
- *void(* ptrRxFunc*
- *void(* CtrlTxCpltCallback*
- *void(* CtrlRxCpltCallback*
- *void(* CtrlMultipleXferCpltCallback*
- *void(* CtrlDAACpltCallback*
- *void(* TgtReqDynamicAddrCallback*
- *void(* TgtTxCpltCallback*
- *void(* TgtRxCpltCallback*
- *void(* TgtHotJoinCallback*
- *void(* NotifyCallback*
- *void(* ErrorCallback*
- *void(* AbortCpltCallback*
- *void(* MspInitCallback*
- *void(* MspDeInitCallback*

Field Documentation

- ***I3C_TypeDef* __I3C_HandleTypeDef::Instance***
  I3C registers base address
- ***I3C_InitTypeDef __I3C_HandleTypeDef::Init***
  I3C communication parameters
- ***HAL_I3C_ModeTypeDef __I3C_HandleTypeDef::Mode***
  I3C communication mode. This parameter must be a value of
  Section 36.3.1: I3C_mode_structure_definition
- ***I3C_XferTypeDef* __I3C_HandleTypeDef::pXferData***
  I3C transfer buffers pointer
- ***const I3C_CCCTypeDef* __I3C_HandleTypeDef::pCCCDesc***
  I3C CCC descriptor pointer
- ***const I3C_PrivateTypeDef* __I3C_HandleTypeDef::pPrivateDesc***
  I3C private transfer descriptor pointer
- ***uint32_t __I3C_HandleTypeDef::ControlXferCount***
  I3C counter indicating the remaining control data bytes to write in the control register
- ***uint32_t __I3C_HandleTypeDef::RxXferCount***
  I3C counter indicating the remaining data bytes to receive
- ***uint32_t __I3C_HandleTypeDef::TxXferCount***
  I3C counter indicating the remaining data bytes to transmit
- ***DMA_HandleTypeDef* __I3C_HandleTypeDef::hdmacr***
  I3C control DMA handle parameters
- ***DMA_HandleTypeDef* __I3C_HandleTypeDef::hdmatx***
  I3C Tx DMA handle parameters
- ***DMA_HandleTypeDef* __I3C_HandleTypeDef::hdmarx***
  I3C Rx DMA handle parameters
- ***DMA_HandleTypeDef* __I3C_HandleTypeDef::hdmasr***
  I3C status DMA handle parameters
- ***HAL_LockTypeDef __I3C_HandleTypeDef::Lock***
  I3C locking object
- ***__IO HAL_I3C_StateTypeDef __I3C_HandleTypeDef::State***
  I3C communication state

- ***__IO HAL_I3C_StateTypeDef __I3C_HandleTypeDef::PreviousState***
  I3C communication previous state
- ***__IO uint32_t __I3C_HandleTypeDef::ErrorCode***
  I3C Error code
- ***HAL_StatusTypeDef(* __I3C_HandleTypeDef::XferISR)(struct __I3C_HandleTypeDef *hi3c, uint32_t itMasks)***
  I3C transfer IRQ handler function pointer
- ***void(* __I3C_HandleTypeDef::ptrTxFunc)(struct __I3C_HandleTypeDef *hi3c)***
  I3C transmit function pointer
- ***void(* __I3C_HandleTypeDef::ptrRxFunc)(struct __I3C_HandleTypeDef *hi3c)***
  I3C receive function pointer
- ***void(* __I3C_HandleTypeDef::CtrlTxCpltCallback)(struct __I3C_HandleTypeDef *hi3c)***
  I3C Controller private data and CCC Tx Transfer complete callback
- ***void(* __I3C_HandleTypeDef::CtrlRxCpltCallback)(struct __I3C_HandleTypeDef *hi3c)***
  I3C Controller private data and CCC Rx Transfer completed callback
- ***void(* __I3C_HandleTypeDef::CtrlMultipleXferCpltCallback)(struct __I3C_HandleTypeDef *hi3c)***
  I3C Controller multiple Direct CCC, I3C private or I2C Transfer completed callback
- ***void(* __I3C_HandleTypeDef::CtrlDAACpltCallback)(struct __I3C_HandleTypeDef *hi3c)***
  I3C Controller Dynamic Address Assignment completed callback
- ***void(* __I3C_HandleTypeDef::TgtReqDynamicAddrCallback)(struct __I3C_HandleTypeDef *hi3c, uint64_t targetPayload)***
  I3C Controller request dynamic address callback during Dynamic Address Assignment processus
- ***void(* __I3C_HandleTypeDef::TgtTxCpltCallback)(struct __I3C_HandleTypeDef *hi3c)***
  I3C Target private data Tx Transfer completed callback
- ***void(* __I3C_HandleTypeDef::TgtRxCpltCallback)(struct __I3C_HandleTypeDef *hi3c)***
  I3C Target private data Rx Transfer completed callback
- ***void(* __I3C_HandleTypeDef::TgtHotJoinCallback)(struct __I3C_HandleTypeDef *hi3c, uint8_t dynamicAddress)***
  I3C Target Hot-Join callback
- ***void(* __I3C_HandleTypeDef::NotifyCallback)(struct __I3C_HandleTypeDef *hi3c, uint32_t eventId)***
  I3C Target or Controller asynchronous events callback
- ***void(* __I3C_HandleTypeDef::ErrorCallback)(struct __I3C_HandleTypeDef *hi3c)***
  I3C Error callback
- ***void(* __I3C_HandleTypeDef::AbortCpltCallback)(struct __I3C_HandleTypeDef *hi3c)***
  I3C Abort complete callback
- ***void(* __I3C_HandleTypeDef::MspInitCallback)(struct __I3C_HandleTypeDef *hi3c)***
  I3C Msp Init callback
- ***void(* __I3C_HandleTypeDef::MspDeInitCallback)(struct __I3C_HandleTypeDef *hi3c)***
  I3C Msp DeInit callback

36.2 I3C Firmware driver API description

The following section lists the various functions of the I3C library.

36.2.1 How to use this driver

The I3C HAL driver can be used as follows:

1. Declare a I3C_HandleTypeDef handle structure, for example: I3C_HandleTypeDef hi3c;
2. Declare a I3C_XferTypeDef transfer descriptor structure, for example: I3C_XferTypeDef ContextBuffers;

3. Initialize the I3C low level resources by implementing the HAL_I3C_MspInit() API:
   a. Enable the I3Cx interface clock
   b. I3C pins configuration
      - Enable the clock for the I3C GPIOs
      - Configure I3C pins as alternate function push-pull with no-pull
   c. NVIC configuration if you need to use interrupt process
      - Configure the I3Cx interrupt priority
      - Enable the NVIC I3C IRQ Channel
   d. DMA Configuration if you need to use DMA process
      - Declare a DMA_HandleTypeDef handle structure for the Command Common Code (CCC) management channel
      - Declare a DMA_HandleTypeDef handle structure for the transmit channel
      - Declare a DMA_HandleTypeDef handle structure for the receive channel
      - Declare a DMA_HandleTypeDef handle structure for the status channel
      - Enable the DMAx interface clock
      - Configure the DMA handle parameters
      - Configure the DMA Command Common Code (CCC) channel
      - Configure the DMA Tx channel
      - Configure the DMA Rx channel
      - Configure the DMA Status channel
      - Associate the initialized DMA handle to the hi3c DMA CCC, Tx, Rx or Status handle as necessary
      - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA CCC, Tx, Rx or Status instance
4. Configure the HAL I3C Communication Mode as Controller or Target in the hi3c Init structure.
5. Configure the Controller Communication Bus characterics for Controller mode. This mean, configure the parameters SDAHoldTime, WaitTime, SCLPPLowDuration, SCLI3CHighDuration, SCLODLowDuration, SCLI2CHighDuration, BusFreeDuration, BusIdleDuration in the LL_I3C_CtrlBusConfTypeDef structure through h3c Init structure.
6. Configure the Target Communication Bus characterics for Target mode. This mean, configure the parameter BusAvailableDuration in the LL_I3C_TgtBusConfTypeDef structure through h3c Init structure. All these parameters for Controller or Target can be configured directly in user code or by using CubeMx generation. To help the computation of the different parameters, the recommendation is to use CubeMx. Those parameters can be modified after the hi3c initialization by using HAL_I3C_Ctrl_BusCharacteristicConfig() for controller and HAL_I3C_Tgt_BusCharacteristicConfig() for target.
7. Initialize the I3C registers by calling the HAL_I3C_Init(), configures also the low level Hardware (GPIO, CLOCK, NVIC...etc) by calling the customized HAL_I3C_MspInit(&hi3c) API.
8. Configure the different FIFO parameters in I3C_FifoConfTypeDef structure as RxFifoThreshold, TxFifoThreshold for Controller or Target mode. And enable/disable the Control or Status FIFO only for Controller Mode. Use HAL_I3C_SetConfigFifo() function to finalize the configuration, and HAL_I3C_GetConfigFifo() to retrieve FIFO configuration. Possibility to clear the FIFO configuration by using HAL_I3C_ClearConfigFifo() which reset the configuration FIFO to their default hardware value
9. Configure the different additional Controller configuration in I3C_CtrlConfTypeDef structure as DynamicAddr, StallTime, HotJoinAllowed, ACKStallState, CCCStallState, TxStallState, RxStallState, HighKeeperSDA. Use HAL_I3C_Ctrl_Config() function to finalize the Controller configuration.
10. Configure the different additional Target configuration in I3C_TgtConfTypeDef structure as Identifier, MIPIIdentifier, CtrlRoleRequest, HotJoinRequest, IBIRequest, IBIPayload, IBIPayloadSize, MaxReadDataSize, MaxWriteDataSize, CtrlCapability, GroupAddrCapability, DataTurnAroundDuration, MaxReadTurnAround, MaxDataSpeed, MaxSpeedLimitation, HandOffActivityState, HandOffDelay, PendingReadMDB. Use HAL_I3C_Tgt_Config() function to finalize the Target configuration.

11. Before initiate any IO operation, the application must launch an assignment of the different Target dynamic address by using HAL_I3C_Ctrl_DynAddrAssign() in polling mode or HAL_I3C_Ctrl_DynAddrAssign_IT() in interrupt mode. This procedure is named Enter Dynamic Address Assignment (ENTDAA CCC command). For the initiation of ENTDAA procedure from the controller, each target connected and powered on the I3C bus must repond to this particular Command Common Code by sending its proper Payload (a amount of 48bits which contain the target characteristics) Each time a target responds to ENTDAA sequence, the application is informed through HAL_I3C_TgtReqDynamicAddrCallback() of the reception of the target paylaod. And then application must send a associated dynamic address through HAL_I3C_Ctrl_SetDynAddr(). This procedure in loop automatically in hardware side until a target respond to repeated ENTDAA sequence. The application is informed of the end of the procedure at reception of HAL_I3C_CtrlDAACpltCallback(). Then application can easily retrieve ENTDAA payload information through HAL_I3C_Get_ENTDAA_Payload_Info() function. At the end of procedure, the function HAL_I3C_Ctrl_ConfigBusDevices() must be called to store in hardware register part the target capabilities as Dynamic address, IBI support with or without additional data byte, Controller role request support, Controller stop transfer after IBI through I3C_DeviceConfTypeDef structure.
12. Other action to be done, before initiate any IO operation, the application must prepare the different frame descriptor with its associated buffer allocation in their side. Configure the different information related to CCC transfer through I3C_CCCTypeDef structure Configure the different information related to Private or I2C transfer through I3C_PrivateTypeDef structure Configure the different buffer pointers and associated size needed for the driver communication through I3C_XferTypeDef structure The I3C_XferTypeDef structure contains different parameters about Control, Status buffer, and Transmit and Receive buffer. Use HAL_I3C_AddDescToFrame() function each time application add a descriptor in the frame before call a Controller IO operation interface One element of the frame descriptor correspond to one frame to manage through IO operation.
13. To check if I3C target device is ready for communication, use the function HAL_I3C_Ctrl_IsDeviceI3C_Ready()
14. To check if I2C target device is ready for communication, use the function HAL_I3C_Ctrl_IsDeviceI2C_Ready()
15. To send a message header {S + 0x7E + W + STOP}, use the function HAL_I3C_Ctrl_GenerateArbitration().
16. To insert a target reset pattern before the STOP of a transmitted frame containing a RSTACT CCC command, the application must enable the reset pattern configuration using HAL_I3C_Ctrl_SetConfigResetPattern() before calling HAL_I3C_Ctrl_TransmitCCC() or HAL_I3C_Ctrl_ReceiveCCC() interfaces. To have a standard STOP emitted at the end of a frame containing a RSTACT CCC command, the application must disable the reset pattern configuration using HAL_I3C_Ctrl_SetConfigResetPattern() before calling HAL_I3C_Ctrl_TransmitCCC() or HAL_I3C_Ctrl_ReceiveCCC() interfaces. Use HAL_I3C_Ctrl_SetConfigResetPattern() function to configure the insertion of the reset pattern at the end of a Frame, and HAL_I3C_Ctrl_GetConfigResetPattern() to retrieve reset pattern configuration.
17. For I3C IO operations, three operation modes are available within this driver :

Polling mode IO operation

- Activate asynchronous event in controller or target mode a Common Command Code in a broadcast or a direct communication in blocking mode using HAL_I3C_Ctrl_TransmitCCC()
- Transmit in controller mode a Common Command Code in a broadcast or a direct communication in blocking mode using HAL_I3C_Ctrl_TransmitCCC()
- Receive in controller mode a Common Command Code in a direct communication in blocking mode using HAL_I3C_Ctrl_ReceiveCCC()
- Transmit in controller mode an amount of private data in an I3C or an I2C communication in blocking mode using HAL_I3C_Ctrl_Transmit()
- Receive in controller mode an amount of private data in an I3C or an I2C communication in blocking mode using HAL_I3C_Ctrl_Receive()
- Transmit in target mode an amount of private data in an I3C communication in blocking mode using HAL_I3C_Tgt_Transmit()
- Receive in target mode an amount of private data in an I3C communication in blocking mode using HAL_I3C_Tgt_Receive()
- At the end of a transfer, the different FIFO can be flushed if necessary by using HAL_I3C_FlushAllFifo() for flush all the FIFO, or flush individually y using HAL_I3C_FlushTxFifo(), HAL_I3C_FlushRxFifo(), HAL_I3C_FlushControlFifo(), HAL_I3C_FlushStatusFifo().
- Request a HotJoin in target mode in blocking mode using HAL_I3C_Tgt_HotJoinReq()
- Request a In Band Interrupt in target mode in blocking mode using HAL_I3C_Tgt_IBIReq()

- Request a Controller Role in target mode in blocking mode using HAL_I3C_Tgt_ControlRoleReq()

DMA and Interrupt mode IO operation

- Transmit in controller mode a Common Command Code in a broadcast or a direct communication in non-blocking mode using HAL_I3C_Ctrl_TransmitCCC_IT() or HAL_I3C_Ctrl_TransmitCCC_DMA()
- At transmission end of transfer, HAL_I3C_CtrlTxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I3C_CtrlTxCpltCallback()
- Receive in controller mode a Common Command Code in a direct communication in non-blocking mode using HAL_I3C_Ctrl_ReceiveCCC_IT() or HAL_I3C_Ctrl_ReceiveCCC_DMA()
- At reception end of transfer, HAL_I3C_CtrlRxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I3C_CtrlRxCpltCallback()
- Transmit in controller mode an amount of private data in an I3C or an I2C communication in non-blocking mode using HAL_I3C_Ctrl_Transmit_IT() or HAL_I3C_Ctrl_Transmit_DMA()
- At transmission end of transfer, HAL_I3C_CtrlTxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I3C_CtrlTxCpltCallback()
- Receive in controller mode an amount of private data in an I3C or an I2C communication in non-blocking mode using HAL_I3C_Ctrl_Receive_IT() or HAL_I3C_Ctrl_Receive_DMA()
- At reception end of transfer, HAL_I3C_CtrlRxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I3C_CtrlRxCpltCallback()
- Transfer in multiple direction (transmit/receive) in controller mode a Common Command Code in a direct communication or an amount of private data in an I2C or I3C communication in non-blocking mode using HAL_I3C_Ctrl_MultipleTransfer_IT() or HAL_I3C_Ctrl_MultipleTransfer_DMA()
- At the end of transfer, HAL_I3C_CtrlMultipleXferCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I3C_CtrlMultipleXferCpltCallback()
- Transmit in target mode an amount of private data in an I3C communication in non-blocking mode using HAL_I3C_Tgt_Transmit_IT() or HAL_I3C_Tgt_Transmit_DMA()
- At transmission end of transfer, HAL_I3C_TgtTxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I3C_TgtTxCpltCallback()
- Receive in target mode an amount of private data in an I3C communication in non-blocking mode using HAL_I3C_Tgt_Receive_IT() or HAL_I3C_Tgt_Receive_DMA()
- At reception end of transfer, HAL_I3C_TgtRxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I3C_TgtRxCpltCallback()
- To treat asynchronous event, HAL_I3C_ActivateNotification() or HAL_I3C_DeactivateNotification() function is used for enable or disable one or more notification related to specific asynchronous event. Each time one or more event detected by hardware the associated HAL_I3C_NotifyCallback() is executed and users can add their own code by customization of function pointer HAL_I3C_NotifyCallback(). Then application can easily retrieve some specific associated event data through HAL_I3C_GetCCCInfo() function
- At the end of a transfer, the different FIFO can be flushed if necessary by using HAL_I3C_FlushAllFifo() for flush all the FIFO, or flush individually y using HAL_I3C_FlushTxFifo(), HAL_I3C_FlushRxFifo(), HAL_I3C_FlushControlFifo(), HAL_I3C_FlushStatusFifo().
- Request a HotJoin in target mode in non-blocking mode using HAL_I3C_Tgt_HotJoinReq_IT
- At completion, HAL_I3C_TgtHotJoinCallback() is executed and users can add their own code by customization of function pointer HAL_I3C_TgtHotJoinCallback()
- Request an In Band Interrupt in target mode in non-blocking mode using HAL_I3C_Tgt_IBIReq_IT()
- At completion, HAL_I3C_NotifyCallback() is executed and users can add their own code by customization of function pointer HAL_I3C_NotifyCallback()
- Request a Controller Role in target mode in non-blocking mode using HAL_I3C_Tgt_ControlRoleReq_IT()
- At completion, HAL_I3C_NotifyCallback() is executed and users can add their own code by customization of function pointer HAL_I3C_NotifyCallback()
- To manage the wakeup capability, HAL_I3C_ActivateNotification() or HAL_I3C_DeactivateNotification() function is used for enable or disable Wake Up interrupt. At wakeup detection the associated HAL_I3C_NotifyCallback() is executed.
- In case of transfer Error, HAL_I3C_ErrorCallback() function is executed and users can add their own code by customization of function pointer HAL_I3C_ErrorCallback()
- Abort an I3C process communication with Interrupt using HAL_I3C_Abort_IT()

- End of abort process, HAL_I3C_AbortCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I3C_AbortCpltCallback()

I3C HAL driver macros list

Below the list of most used macros in I3C HAL driver.

- __HAL_I3C_ENABLE: Enable the I3C peripheral
- __HAL_I3C_DISABLE: Disable the I3C peripheral
- __HAL_I3C_RESET_HANDLE_STATE: Reset the I3C handle state
- __HAL_I3C_GET_FLAG: Check whether the specified I3C flag is set or not

Callback registration

The compilation flag USE_HAL_I3C_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use Functions HAL_I3C_RegisterCallback() or HAL_I3C_RegisterNotifyCallback() or HAL_I3C_RegisterTgtReqDynamicAddrCallback() or HAL_I3C_RegisterTgtHotJoinCallback() to register an interrupt callback.

Function HAL_I3C_RegisterCallback() allows to register following callbacks:

- CtrlTxCpltCallback : callback for Controller transmission CCC, I3C private or I2C end of transfer.
- CtrlRxCpltCallback : callback for Controller reception CCC, I3C private or I2C end of transfer.
- CtrlMultipleXferCpltCallback : callback for Controller multiple Direct CCC, I3C private or I2C end of transfer.
- CtrlDAACpltCallback : callback for Controller Enter Dynamic Address Assignment end of transfer.
- TgtTxCpltCallback : callback for Target transmission I3C private end of transfer.
- TgtRxCpltCallback : callback for Target reception I3C private end of transfer.
- ErrorCallback : callback for error detection.
- AbortCpltCallback : callback for abort completion process.
- MspInitCallback : callback for Msp Init.
- MspDeInitCallback : callback for Msp DeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

For specific callback NotifyCallback use dedicated register callbacks : HAL_I3C_RegisterNotifyCallback().

For specific callback TgtReqDynamicAddrCallback use dedicated register callbacks : HAL_I3C_RegisterTgtReqDynamicAddrCallback().

For specific callback TgtHotJoinCallback use dedicated register callbacks : HAL_I3C_RegisterTgtHotJoinCallback().

Use function HAL_I3C_UnRegisterCallback to reset a callback to the default weak function. HAL_I3C_UnRegisterCallback takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks:

- CtrlTxCpltCallback : callback for Controller transmission CCC, I3C private or I2C end of transfer.
- CtrlRxCpltCallback : callback for Controller reception CCC, I3C private or I2C end of transfer.
- CtrlMultipleXferCpltCallback : callback for Controller multiple Direct CCC, I3C private or I2C end of transfer.
- CtrlDAACpltCallback : callback for Controller Enter Dynamic Address Assignment end of transfer.
- TgtTxCpltCallback : callback for Target transmission I3C private end of transfer.
- TgtRxCpltCallback : callback for Target reception I3C private end of transfer.
- ErrorCallback : callback for error detection.
- AbortCpltCallback : callback for abort completion process.
- MspInitCallback : callback for Msp Init.
- MspDeInitCallback : callback for Msp DeInit.
- NotifyCallback : callback for Controller and Target notification process.
- TgtReqDynamicAddrCallback : callback for controller application when a target sent its payload to the controller during Dynamic Address Assignment process.
- TgtHotJoinCallback : callback for Target Hotjoin completion process.

By default, after the HAL_I3C_Init() and when the state is HAL_I3C_STATE_RESET all callbacks are set to the corresponding weak functions: examples HAL_I3C_CtrlTxCpltCallback(), HAL_I3C_CtrlRxCpltCallback(). Exception done for MspInit and MspDeInit functions that are reset to the legacy weak functions in the HAL_I3C_Init()/ HAL_I3C_DeInit() only when these callbacks are null (not registered beforehand). If MspInit or MspDeInit are not null, the HAL_I3C_Init()/ HAL_I3C_DeInit() keep and use the user MspInit/MspDeInit callbacks (registered beforehand) whatever the state.

Callbacks can be registered/unregistered in HAL_I3C_STATE_READY state only. Exception done MspInit/ MspDeInit functions that can be registered/unregistered in HAL_I3C_STATE_READY or HAL_I3C_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. Then, the user first registers the MspInit/MspDeInit user callbacks using HAL_I3C_RegisterCallback() before calling HAL_I3C_DeInit() or HAL_I3C_Init() function.

When the compilation flag USE_HAL_I3C_REGISTER_CALLBACKS is set to 0 or not defined, the callback registration feature is not available and all callbacks are set to the corresponding weak functions.

Note: You can refer to the I3C HAL driver header file for more useful macros

36.2.2 Detailed description of functions

HAL_I3C_Init

Function name

HAL_StatusTypeDef HAL_I3C_Init (I3C_HandleTypeDef * hi3c)

Function description

HAL_I3C_DeInit

Function name

HAL_StatusTypeDef HAL_I3C_DeInit (I3C_HandleTypeDef * hi3c)

Function description

HAL_I3C_MspInit

Function name

void HAL_I3C_MspInit (I3C_HandleTypeDef * hi3c)

Function description

HAL_I3C_MspDeInit

Function name

void HAL_I3C_MspDeInit (I3C_HandleTypeDef * hi3c)

Function description

HAL_I3C_RegisterCallback

Function name

HAL_StatusTypeDef HAL_I3C_RegisterCallback (I3C_HandleTypeDef * hi3c, HAL_I3C_CallbackIDTypeDef callbackID, pI3C_CallbackTypeDef pCallback)

Function description

HAL_I3C_RegisterNotifyCallback

Function name

HAL_StatusTypeDef HAL_I3C_RegisterNotifyCallback (I3C_HandleTypeDef * hi3c, pI3C_NotifyCallbackTypeDef pNotifyCallback)

Function description

HAL_I3C_RegisterTgtReqDynamicAddrCallback

Function name

HAL_StatusTypeDef HAL_I3C_RegisterTgtReqDynamicAddrCallback (I3C_HandleTypeDef * hi3c, pI3C_TgtReqDynamicAddrCallbackTypeDef pTgtReqAddrCallback)

Function description

HAL_I3C_RegisterTgtHotJoinCallback

Function name

HAL_StatusTypeDef HAL_I3C_RegisterTgtHotJoinCallback (I3C_HandleTypeDef * hi3c, pI3C_TgtHotJoinCallbackTypeDef pTgtHotJoinCallback)

Function description

HAL_I3C_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_I3C_UnRegisterCallback (I3C_HandleTypeDef * hi3c, HAL_I3C_CallbackIDTypeDef callbackID)

Function description

HAL_I3C_ActivateNotification

Function name

HAL_StatusTypeDef HAL_I3C_ActivateNotification (I3C_HandleTypeDef * hi3c, I3C_XferTypeDef * pXferData, uint32_t interruptMask)

Function description

HAL_I3C_DeactivateNotification

Function name

HAL_StatusTypeDef HAL_I3C_DeactivateNotification (I3C_HandleTypeDef * hi3c, uint32_t interruptMask)

Function description

HAL_I3C_CtrlTxCpltCallback

Function name

void HAL_I3C_CtrlTxCpltCallback (I3C_HandleTypeDef * hi3c)

Function description

HAL_I3C_CtrlRxCpltCallback

Function name

void HAL_I3C_CtrlRxCpltCallback (I3C_HandleTypeDef * hi3c)

Function description

HAL_I3C_CtrlMultipleXferCpltCallback

Function name

void HAL_I3C_CtrlMultipleXferCpltCallback (I3C_HandleTypeDef * hi3c)

Function description

HAL_I3C_CtrlDAACpltCallback

Function name

void HAL_I3C_CtrlDAACpltCallback (I3C_HandleTypeDef * hi3c)

Function description

HAL_I3C_TgtReqDynamicAddrCallback

Function name

void HAL_I3C_TgtReqDynamicAddrCallback (I3C_HandleTypeDef * hi3c, uint64_t targetPayload)

Function description

HAL_I3C_TgtTxCpltCallback

Function name

void HAL_I3C_TgtTxCpltCallback (I3C_HandleTypeDef * hi3c)

Function description

HAL_I3C_TgtRxCpltCallback

Function name

void HAL_I3C_TgtRxCpltCallback (I3C_HandleTypeDef * hi3c)

Function description

HAL_I3C_TgtHotJoinCallback

Function name

void HAL_I3C_TgtHotJoinCallback (I3C_HandleTypeDef * hi3c, uint8_t dynamicAddress)

Function description

HAL_I3C_NotifyCallback

Function name

void HAL_I3C_NotifyCallback (I3C_HandleTypeDef * hi3c, uint32_t eventId)

Function description

HAL_I3C_AbortCpltCallback

Function name

void HAL_I3C_AbortCpltCallback (I3C_HandleTypeDef * hi3c)

Function description

HAL_I3C_ErrorCallback

Function name

void HAL_I3C_ErrorCallback (I3C_HandleTypeDef * hi3c)

Function description

HAL_I3C_ER_IRQHandler

Function name

void HAL_I3C_ER_IRQHandler (I3C_HandleTypeDef * hi3c)

Function description

HAL_I3C_EV_IRQHandler

Function name

void HAL_I3C_EV_IRQHandler (I3C_HandleTypeDef * hi3c)

Function description

HAL_I3C_Ctrl_BusCharacteristicConfig

Function name

HAL_StatusTypeDef HAL_I3C_Ctrl_BusCharacteristicConfig (I3C_HandleTypeDef * hi3c, const LL_I3C_CtrlBusConfTypeDef * pConfig)

Function description

HAL_I3C_Tgt_BusCharacteristicConfig

Function name

HAL_StatusTypeDef HAL_I3C_Tgt_BusCharacteristicConfig (I3C_HandleTypeDef * hi3c, const LL_I3C_TgtBusConfTypeDef * pConfig)

Function description

HAL_I3C_SetConfigFifo

Function name

HAL_StatusTypeDef HAL_I3C_SetConfigFifo (I3C_HandleTypeDef * hi3c, const I3C_FifoConfTypeDef * pConfig)

Function description

HAL_I3C_Ctrl_Config

Function name

HAL_StatusTypeDef HAL_I3C_Ctrl_Config (I3C_HandleTypeDef * hi3c, const I3C_CtrlConfTypeDef * pConfig)

Function description

HAL_I3C_Tgt_Config

Function name

HAL_StatusTypeDef HAL_I3C_Tgt_Config (I3C_HandleTypeDef * hi3c, const I3C_TgtConfTypeDef * pConfig)

Function description

HAL_I3C_Ctrl_ConfigBusDevices

Function name

HAL_StatusTypeDef HAL_I3C_Ctrl_ConfigBusDevices (I3C_HandleTypeDef * hi3c, const I3C_DeviceConfTypeDef * pDesc, uint8_t nbDevice)

Function description

HAL_I3C_AddDescToFrame

Function name

HAL_StatusTypeDef HAL_I3C_AddDescToFrame (I3C_HandleTypeDef * hi3c, const I3C_CCCTypeDef * pCCCDesc, const I3C_PrivateTypeDef * pPrivateDesc, I3C_XferTypeDef * pXferData, uint8_t nbFrame, uint32_t option)

Function description

HAL_I3C_Ctrl_SetConfigResetPattern

Function name

HAL_StatusTypeDef HAL_I3C_Ctrl_SetConfigResetPattern (I3C_HandleTypeDef * hi3c, uint32_t resetPattern)

Function description

HAL_I3C_Ctrl_GetConfigResetPattern

Function name

HAL_StatusTypeDef HAL_I3C_Ctrl_GetConfigResetPattern (I3C_HandleTypeDef * hi3c, uint32_t * pResetPattern)

Function description

HAL_I3C_FlushAllFifo

Function name

HAL_StatusTypeDef HAL_I3C_FlushAllFifo (I3C_HandleTypeDef * hi3c)

Function description

HAL_I3C_FlushTxFifo

Function name

HAL_StatusTypeDef HAL_I3C_FlushTxFifo (I3C_HandleTypeDef * hi3c)

Function description

HAL_I3C_FlushRxFifo

Function name

HAL_StatusTypeDef HAL_I3C_FlushRxFifo (I3C_HandleTypeDef * hi3c)

Function description

HAL_I3C_FlushControlFifo

Function name

HAL_StatusTypeDef HAL_I3C_FlushControlFifo (I3C_HandleTypeDef * hi3c)

Function description

HAL_I3C_FlushStatusFifo

Function name

HAL_StatusTypeDef HAL_I3C_FlushStatusFifo (I3C_HandleTypeDef * hi3c)

Function description

HAL_I3C_ClearConfigFifo

Function name

HAL_StatusTypeDef HAL_I3C_ClearConfigFifo (I3C_HandleTypeDef * hi3c)

Function description

HAL_I3C_GetConfigFifo

Function name

HAL_StatusTypeDef HAL_I3C_GetConfigFifo (I3C_HandleTypeDef * hi3c, I3C_FifoConfTypeDef * pConfig)

Function description

HAL_I3C_Ctrl_TransmitCCC

Function name

HAL_StatusTypeDef HAL_I3C_Ctrl_TransmitCCC (I3C_HandleTypeDef * hi3c, I3C_XferTypeDef * pXferData, uint32_t timeout)

Function description

HAL_I3C_Ctrl_TransmitCCC_IT

Function name

HAL_StatusTypeDef HAL_I3C_Ctrl_TransmitCCC_IT (I3C_HandleTypeDef * hi3c, I3C_XferTypeDef * pXferData)

Function description

HAL_I3C_Ctrl_TransmitCCC_DMA

Function name

HAL_StatusTypeDef HAL_I3C_Ctrl_TransmitCCC_DMA (I3C_HandleTypeDef * hi3c, I3C_XferTypeDef * pXferData)

Function description

HAL_I3C_Ctrl_ReceiveCCC

Function name

HAL_StatusTypeDef HAL_I3C_Ctrl_ReceiveCCC (I3C_HandleTypeDef * hi3c, I3C_XferTypeDef * pXferData, uint32_t timeout)

Function description

HAL_I3C_Ctrl_ReceiveCCC_IT

Function name

HAL_StatusTypeDef HAL_I3C_Ctrl_ReceiveCCC_IT (I3C_HandleTypeDef * hi3c, I3C_XferTypeDef * pXferData)

Function description

HAL_I3C_Ctrl_ReceiveCCC_DMA

Function name

HAL_StatusTypeDef HAL_I3C_Ctrl_ReceiveCCC_DMA (I3C_HandleTypeDef * hi3c, I3C_XferTypeDef * pXferData)

Function description

HAL_I3C_Ctrl_Transmit

Function name

HAL_StatusTypeDef HAL_I3C_Ctrl_Transmit (I3C_HandleTypeDef * hi3c, I3C_XferTypeDef * pXferData, uint32_t timeout)

Function description

HAL_I3C_Ctrl_Transmit_IT

Function name

HAL_StatusTypeDef HAL_I3C_Ctrl_Transmit_IT (I3C_HandleTypeDef * hi3c, I3C_XferTypeDef * pXferData)

Function description

HAL_I3C_Ctrl_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_I3C_Ctrl_Transmit_DMA (I3C_HandleTypeDef * hi3c, I3C_XferTypeDef * pXferData)

Function description

HAL_I3C_Ctrl_Receive

Function name

HAL_StatusTypeDef HAL_I3C_Ctrl_Receive (I3C_HandleTypeDef * hi3c, I3C_XferTypeDef * pXferData, uint32_t timeout)

Function description

HAL_I3C_Ctrl_Receive_IT

Function name

HAL_StatusTypeDef HAL_I3C_Ctrl_Receive_IT (I3C_HandleTypeDef * hi3c, I3C_XferTypeDef * pXferData)

Function description

HAL_I3C_Ctrl_Receive_DMA

Function name

HAL_StatusTypeDef HAL_I3C_Ctrl_Receive_DMA (I3C_HandleTypeDef * hi3c, I3C_XferTypeDef * pXferData)

Function description

HAL_I3C_Ctrl_MultipleTransfer_IT

Function name

HAL_StatusTypeDef HAL_I3C_Ctrl_MultipleTransfer_IT (I3C_HandleTypeDef * hi3c, I3C_XferTypeDef * pXferData)

Function description

HAL_I3C_Ctrl_MultipleTransfer_DMA

Function name

HAL_StatusTypeDef HAL_I3C_Ctrl_MultipleTransfer_DMA (I3C_HandleTypeDef * hi3c, I3C_XferTypeDef * pXferData)

Function description

HAL_I3C_Ctrl_SetDynAddr

Function name

HAL_StatusTypeDef HAL_I3C_Ctrl_SetDynAddr (I3C_HandleTypeDef * hi3c, uint8_t devAddress)

Function description

HAL_I3C_Ctrl_DynAddrAssign_IT

Function name

HAL_StatusTypeDef HAL_I3C_Ctrl_DynAddrAssign_IT (I3C_HandleTypeDef * hi3c, uint32_t dynOption)

Function description

HAL_I3C_Ctrl_DynAddrAssign

Function name

HAL_StatusTypeDef HAL_I3C_Ctrl_DynAddrAssign (I3C_HandleTypeDef * hi3c, uint64_t * target_payload, uint32_t dynOption, uint32_t timeout)

Function description

HAL_I3C_Ctrl_IsDeviceI3C_Ready

Function name

HAL_StatusTypeDef HAL_I3C_Ctrl_IsDeviceI3C_Ready (I3C_HandleTypeDef * hi3c, uint8_t devAddress, uint32_t trials, uint32_t timeout)

Function description

HAL_I3C_Ctrl_IsDeviceI2C_Ready

Function name

HAL_StatusTypeDef HAL_I3C_Ctrl_IsDeviceI2C_Ready (I3C_HandleTypeDef * hi3c, uint8_t devAddress, uint32_t trials, uint32_t timeout)

Function description

HAL_I3C_Ctrl_GenerateArbitration

Function name

HAL_StatusTypeDef HAL_I3C_Ctrl_GenerateArbitration (I3C_HandleTypeDef * hi3c, uint32_t timeout)

Function description

HAL_I3C_Tgt_Transmit

Function name

HAL_StatusTypeDef HAL_I3C_Tgt_Transmit (I3C_HandleTypeDef * hi3c, I3C_XferTypeDef * pXferData, uint32_t timeout)

Function description

HAL_I3C_Tgt_Transmit_IT

Function name

HAL_StatusTypeDef HAL_I3C_Tgt_Transmit_IT (I3C_HandleTypeDef * hi3c, I3C_XferTypeDef * pXferData)

Function description

HAL_I3C_Tgt_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_I3C_Tgt_Transmit_DMA (I3C_HandleTypeDef * hi3c, I3C_XferTypeDef * pXferData)

Function description

HAL_I3C_Tgt_Receive

Function name

HAL_StatusTypeDef HAL_I3C_Tgt_Receive (I3C_HandleTypeDef * hi3c, I3C_XferTypeDef * pXferData, uint32_t timeout)

Function description

HAL_I3C_Tgt_Receive_IT

Function name

HAL_StatusTypeDef HAL_I3C_Tgt_Receive_IT (I3C_HandleTypeDef * hi3c, I3C_XferTypeDef * pXferData)

Function description

HAL_I3C_Tgt_Receive_DMA

Function name

HAL_StatusTypeDef HAL_I3C_Tgt_Receive_DMA (I3C_HandleTypeDef * hi3c, I3C_XferTypeDef * pXferData)

Function description

HAL_I3C_Tgt_ControlRoleReq

Function name

HAL_StatusTypeDef HAL_I3C_Tgt_ControlRoleReq (I3C_HandleTypeDef * hi3c, uint32_t timeout)

Function description

HAL_I3C_Tgt_ControlRoleReq_IT

Function name

HAL_StatusTypeDef HAL_I3C_Tgt_ControlRoleReq_IT (I3C_HandleTypeDef * hi3c)

Function description

HAL_I3C_Tgt_HotJoinReq

Function name

HAL_StatusTypeDef HAL_I3C_Tgt_HotJoinReq (I3C_HandleTypeDef * hi3c, uint8_t * pAddress, uint32_t timeout)

Function description

HAL_I3C_Tgt_HotJoinReq_IT

Function name

HAL_StatusTypeDef HAL_I3C_Tgt_HotJoinReq_IT (I3C_HandleTypeDef * hi3c)

Function description

HAL_I3C_Tgt_IBIReq

Function name

HAL_StatusTypeDef HAL_I3C_Tgt_IBIReq (I3C_HandleTypeDef * hi3c, const uint8_t * pPayload, uint8_t payloadSize, uint32_t timeout)

Function description

HAL_I3C_Tgt_IBIReq_IT

Function name

HAL_StatusTypeDef HAL_I3C_Tgt_IBIReq_IT (I3C_HandleTypeDef * hi3c, const uint8_t * pPayload, uint8_t payloadSize)

Function description

HAL_I3C_Abort_IT

Function name

HAL_StatusTypeDef HAL_I3C_Abort_IT (I3C_HandleTypeDef * hi3c)

Function description

HAL_I3C_GetState

Function name

HAL_I3C_StateTypeDef HAL_I3C_GetState (const I3C_HandleTypeDef * hi3c)

Function description

HAL_I3C_GetMode

Function name

HAL_I3C_ModeTypeDef HAL_I3C_GetMode (const I3C_HandleTypeDef * hi3c)

Function description

HAL_I3C_GetError

Function name

uint32_t HAL_I3C_GetError (const I3C_HandleTypeDef * hi3c)

Function description

HAL_I3C_GetCCCInfo

Function name

HAL_StatusTypeDef HAL_I3C_GetCCCInfo (I3C_HandleTypeDef * hi3c, uint32_t notifyId, I3C_CCCInfoTypeDef * pCCCInfo)

Function description

HAL_I3C_Get_ENTDAA_Payload_Info

Function name

HAL_StatusTypeDef HAL_I3C_Get_ENTDAA_Payload_Info (I3C_HandleTypeDef * hi3c, uint64_t ENTDAA_payload, I3C_ENTDAAPayloadTypeDef * pENTDAA_payload)

Function description

36.3 I3C Firmware driver defines

The following section lists the various define and macros of the module.

36.3.1 I3C

I3C

I3C BCR IN PAYLOAD

HAL_I3C_BCR_IN_PAYLOAD_SHIFT

BCR field in target payload

LL_I3C_BCR_IN_PAYLOAD_SHIFT

BCR field in target payload

I3C COMMON INTERRUPT

HAL_I3C_IT_TXFNFIE

Tx FIFO not full interrupt enable

HAL_I3C_IT_RXFNEIE

Rx FIFO not empty interrupt enable

HAL_I3C_IT_FCIE

Frame complete interrupt enable

HAL_I3C_IT_ERRIE

Error interrupt enable

HAL_I3C_ALL_COMMON_ITS

I3C CONTROLLER INTERRUPT

HAL_I3C_IT_CFNFIE

Control FIFO not full interrupt enable

HAL_I3C_IT_SFNEIE

Status FIFO not empty interrupt enable

HAL_I3C_IT_HJIE

Hot-join interrupt enable

HAL_I3C_IT_CRIE

Controller-role request interrupt enable

HAL_I3C_IT_IBIIE

IBI request interrupt enable

HAL_I3C_IT_RXTGTENDIE

Target-initiated read end interrupt enable

HAL_I3C_ALL_CTRL_ITS

CONTROL FIFO STATE

HAL_I3C_CONTROLFIFO_DISABLE

Control FIFO mode disable

HAL_I3C_CONTROLFIFO_ENABLE

Control FIFO mode enable

DIRECTION

HAL_I3C_DIRECTION_WRITE

Write transfer

HAL_I3C_DIRECTION_READ

Read transfer

HAL_I3C_DIRECTION_BOTH

Read and Write transfer

I3C DYNAMIC ADDRESS OPTION DEFINITION

I3C_RSTDAA_THEN_ENTDAA

Initiate a RSTDAA before a ENTDAA procedure

I3C_ONLY_ENTDAA

Initiate a ENTDAA without RSTDAA

ERROR CODE DEFINITION

HAL_I3C_ERROR_NONE

No error

HAL_I3C_ERROR_CE0

Controller detected an illegally formatted CCC

HAL_I3C_ERROR_CE1

Controller detected that transmitted data on the bus is different than expected

HAL_I3C_ERROR_CE2

Controller detected that broadcast address 7'h7E has been nacked

HAL_I3C_ERROR_CE3

Controller detected that new Controller did not drive the bus after Controller-role handoff

HAL_I3C_ERROR_TE0

Target detected an invalid broadcast address

HAL_I3C_ERROR_TE1

Target detected an invalid CCC Code

HAL_I3C_ERROR_TE2

Target detected a parity error during a write data

HAL_I3C_ERROR_TE3

Target detected a parity error on assigned address during dynamic address arbitration

HAL_I3C_ERROR_TE4

Target detected 7'h7E missing after Restart during Dynamic Address Assignment procedure

HAL_I3C_ERROR_TE5

Target detected an illegally formatted CCC

HAL_I3C_ERROR_TE6

Target detected that transmitted data on the bus is different than expected

HAL_I3C_ERROR_DATA_HAND_OFF

I3C data error during controller-role hand-off process

HAL_I3C_ERROR_DATA_NACK

I3C data not acknowledged error

HAL_I3C_ERROR_ADDRESS_NACK

I3C address not acknowledged error

HAL_I3C_ERROR_COVR

I3C S FIFO Over-Run or C FIFO Under-Run error

HAL_I3C_ERROR_DOVR

I3C Rx FIFO Over-Run or Tx FIFO Under-Run error

HAL_I3C_ERROR_STALL

I3C SCL stall error

HAL_I3C_ERROR_DMA

DMA transfer error

HAL_I3C_ERROR_TIMEOUT

Timeout error

HAL_I3C_ERROR_DMA_PARAM

DMA Parameter Error

HAL_I3C_ERROR_INVALID_PARAM

Invalid Parameters error

HAL_I3C_ERROR_SIZE

I3C size management error

HAL_I3C_ERROR_NOT_ALLOWED

I3C operation is not allowed

HAL_I3C_ERROR_DYNAMIC_ADDR

I3C dynamic address error

HAL_I3C_ERROR_INVALID_CALLBACK

Invalid Callback error

I3C Exported Macros

__HAL_I3C_RESET_HANDLE_STATE

Description:

- Reset I3C handle state.

Parameters:

- __HANDLE__: specifies the I3C Handle.

Return value:

- None

__HAL_I3C_ENABLE_IT

Description:

- Enable the specified I3C interrupt.

Parameters:

- __HANDLE__: specifies the I3C Handle.
- __INTERRUPT__: specifies the interrupt source to enable. This parameter can be one value or a combination of the following group's values:
  - I3C CONTROLLER INTERRUPT
  - I3C TARGET INTERRUPT
  - I3C COMMON INTERRUPT

Return value:

- None

__HAL_I3C_DISABLE_IT

Description:

- Disable the specified I3C interrupt.

Parameters:

- __HANDLE__: specifies the I3C Handle.
- __INTERRUPT__: specifies the interrupt source to disable. This parameter can be one value or a combination of the following group's values:
  - I3C CONTROLLER INTERRUPT
  - I3C TARGET INTERRUPT
  - I3C COMMON INTERRUPT

Return value:

- None

__HAL_I3C_GET_FLAG

Description:

- Check whether the specified I3C flag is set or not.

Parameters:

- __HANDLE__: specifies the I3C Handle.
- __FLAG__: specifies the flag to check. This parameter can be one value of the group
  - I3C FLAGS values.

Return value:

- The: new state of __FLAG__ (SET or RESET).

__HAL_I3C_GET_BCR

Description:

- Get Bus Characterics in payload (64bits) receive during ENTDAA procedure.

Parameters:

- __PAYLOAD__: specifies the Bus Characteristics capabilities retrieve during ENTDAA procedure. This parameter must be a number between Min_Data=0x00(uint64_t) and Max_Data=0xFFFFFFFFFFFFFFFF.

Return value:

- The: value of BCR Return value between Min_Data=0x00 and Max_Data=0xFF.

__HAL_I3C_GET_IBI_CAPABLE

Description:

- Check IBI request capabilities.

Parameters:

- __BCR__: specifies the Bus Characteristics capabilities retrieve during ENTDAA procedure. This parameter must be a number between Min_Data=0x00 and Max_Data=0xFF.

Return value:

- The: state of IBI request capabilities (ENABLE or DISABLE).

__HAL_I3C_GET_IBI_PAYLOAD

Description:

- Check IBI additional data byte capabilities.

Parameters:

- __BCR__: specifies the Bus Characteristics capabilities retrieve during ENTDAA procedure. This parameter must be a number between Min_Data=0x00 and Max_Data=0xFF.

Return value:

- The: state of IBI additional data byte capabilities (ENABLE or DISABLE).

__HAL_I3C_GET_CR_CAPABLE

Description:

- Check Controller role request capabilities.

Parameters:

- __BCR__: specifies the Bus Characteristics capabilities retrieve during ENTDAA procedure. This parameter must be a number between Min_Data=0x00 and Max_Data=0xFF.

Return value:

- The: state of Controller role request capabilities (ENABLE or DISABLE).

I3C FLAGS

HAL_I3C_FLAG_CFEF

Control FIFO not empty flag

HAL_I3C_FLAG_TXFEF

Tx FIFO empty flag

HAL_I3C_FLAG_CFNFF

Control FIFO not full flag

HAL_I3C_FLAG_SFNEF

Status FIFO not empty flag

HAL_I3C_FLAG_TXFNFF

Tx FIFO not full flag

HAL_I3C_FLAG_RXFNEF

Rx FIFO not empty flag

HAL_I3C_FLAG_RXLASTF

Last read data byte/word flag

HAL_I3C_FLAG_TXLASTF

Last written data byte/word flag

HAL_I3C_FLAG_FCF

Frame complete flag

HAL_I3C_FLAG_RXTGTENDF

Target-initiated read end flag

HAL_I3C_FLAG_ERRF

Error flag

HAL_I3C_FLAG_IBIF

IBI request flag

HAL_I3C_FLAG_IBIENDF

IBI end flag

HAL_I3C_FLAG_CRF

Controller-role request flag

HAL_I3C_FLAG_CRUPDF

Controller-role update flag

HAL_I3C_FLAG_HJF

Hot-join flag

HAL_I3C_FLAG_WKPF

Wakeup flag

HAL_I3C_FLAG_GETF

GETxxx CCC flag

HAL_I3C_FLAG_STAF

Format 1 GETSTATUS CCC flag

HAL_I3C_FLAG_DAUPDF

ENTDAA/RSTDAA/SETNEWDA CCC flag

HAL_I3C_FLAG_MWLUPDF

SETMWL CCC flag

HAL_I3C_FLAG_MRLUPDF

SETMRL CCC flag

HAL_I3C_FLAG_RSTF

Reset pattern flag

HAL_I3C_FLAG_ASUPDF

ENTASx CCC flag

HAL_I3C_FLAG_INTUPDF

ENEC/DISEC CCC flag

HAL_I3C_FLAG_DEFF

DEFTGTS CCC flag

HAL_I3C_FLAG_GRPF

DEFGRPA CCC flag

GETMXDS FORMAT

HAL_I3C_GETMXDS_FORMAT_1

GETMXDS CCC Format 1 is used, no MaxRdTurn field in response

HAL_I3C_GETMXDS_FORMAT_2_LSB

GETMXDS CCC Format 2 is used, MaxRdTurn field in response, LSB = RDTURN[7:0]

HAL_I3C_GETMXDS_FORMAT_2_MID

GETMXDS CCC Format 2 is used, MaxRdTurn field in response, Middle byte = RDTURN[7:0]

HAL_I3C_GETMXDS_FORMAT_2_MSB

GETMXDS CCC Format 2 is used, MaxRdTurn field in response, MSB = RDTURN[7:0]

HANDOFF ACTIVITY STATE

HAL_I3C_HANDOFF_ACTIVITY_STATE_0

Activity state 0 after handoff

HAL_I3C_HANDOFF_ACTIVITY_STATE_1

Activity state 1 after handoff

HAL_I3C_HANDOFF_ACTIVITY_STATE_2

Activity state 2 after handoff

HAL_I3C_HANDOFF_ACTIVITY_STATE_3

Activity state 3 after handoff

OPTION DEFINITION

I3C_DIRECT_WITH_DEFBYTE_RESTART

Restart between each Direct Command then Stop request for last command. Each Command have an associated defining byte

I3C_DIRECT_WITH_DEFBYTE_STOP

Stop between each Direct Command. Each Command have an associated defining byte

I3C_DIRECT_WITHOUT_DEFBYTE_RESTART

Restart between each Direct Command then Stop request for last command. Each Command have not an associated defining byte

I3C_DIRECT_WITHOUT_DEFBYTE_STOP

Stop between each Direct Command. Each Command have not an associated defining byte

I3C_BROADCAST_WITH_DEFBYTE_RESTART

Restart between each Broadcast Command then Stop request for last command. Each Command have an associated defining byte

I3C_BROADCAST_WITH_DEFBYTE_STOP

Stop between each Broadcast Command. Each Command have an associated defining byte

I3C_BROADCAST_WITHOUT_DEFBYTE_RESTART

Restart between each Broadcast Command then Stop request for last command. Each Command have not an associated defining byte

I3C_BROADCAST_WITHOUT_DEFBYTE_STOP

Stop between each Broadcast Command. Each Command have not an associated defining byte

I3C_PRIVATE_WITH_ARB_RESTART

Restart between each I3C Private message then Stop request for last message. Each Message start with an arbitration header after start bit condition

I3C_PRIVATE_WITH_ARB_STOP

Stop between each I3C Private message. Each Message start with an arbitration header after start bit condition

I3C_PRIVATE_WITHOUT_ARB_RESTART

Restart between each I3C message then Stop request for last message. Each Message start with Target address after start bit condition

I3C_PRIVATE_WITHOUT_ARB_STOP

Stop between each I3C Private message. Each Message start with Target address after start bit condition

I2C_PRIVATE_WITH_ARB_RESTART

Restart between each I2C Private message then Stop request for last message. Each Message start with an arbitration header after start bit condition

I2C_PRIVATE_WITH_ARB_STOP

Stop between each I2C Private message. Each Message start with an arbitration header after start bit condition

I2C_PRIVATE_WITHOUT_ARB_RESTART

Restart between each I2C message then Stop request for last message. Each Message start with Target address after start bit condition

I2C_PRIVATE_WITHOUT_ARB_STOP

Stop between each I2C Private message. Each Message start with Target address after start bit condition

OWN ACTIVITY STATE

HAL_I3C_OWN_ACTIVITY_STATE_0

Own Controller Activity state 0

HAL_I3C_OWN_ACTIVITY_STATE_1

Own Controller Activity state 1

HAL_I3C_OWN_ACTIVITY_STATE_2

Own Controller Activity state 2

HAL_I3C_OWN_ACTIVITY_STATE_3

Own Controller Activity state 3

I3C PATTERN CONFIGURATION

HAL_I3C_TARGET_RESET_PATTERN

Target reset pattern

HAL_I3C_HDR_EXIT_PATTERN

HDR exit pattern

PAYLOAD SIZE

HAL_I3C_PAYLOAD_EMPTY

Empty payload, no additional data after IBI acknowledge

HAL_I3C_PAYLOAD_1_BYTE

One additional data byte after IBI acknowledge

HAL_I3C_PAYLOAD_2_BYTES

Two additional data bytes after IBI acknowledge

HAL_I3C_PAYLOAD_3_BYTES

Three additional data bytes after IBI acknowledge

HAL_I3C_PAYLOAD_4_BYTES

Four additional data bytes after IBI acknowledge

RESET PATTERN

HAL_I3C_RESET_PATTERN_DISABLE

Standard STOP condition emitted at the end of a frame

HAL_I3C_RESET_PATTERN_ENABLE

Reset pattern is inserted before the STOP condition of any emitted frame

RX FIFO THRESHOLD

HAL_I3C_RXFIFO_THRESHOLD_1_4

Rx Fifo Threshold is 1 byte

HAL_I3C_RXFIFO_THRESHOLD_4_4

Rx Fifo Threshold is 4 bytes

SDA HOLD TIME

HAL_I3C_SDA_HOLD_TIME_0_5

SDA hold time equal to 0.5 x ti3cclk

HAL_I3C_SDA_HOLD_TIME_1_5

SDA hold time equal to 1.5 x ti3cclk

STATUS FIFO STATE

HAL_I3C_STATUSFIFO_DISABLE

Status FIFO mode disable

HAL_I3C_STATUSFIFO_ENABLE

Status FIFO mode enable

I3C TARGET INTERRUPT

HAL_I3C_IT_IBIENDIE

IBI end interrupt enable

HAL_I3C_IT_CRUPDIE

controller-role update interrupt enable

HAL_I3C_IT_WKPIE

wakeup interrupt enable

HAL_I3C_IT_GETIE

GETxxx CCC interrupt enable

HAL_I3C_IT_STAIE

GETSTATUS CCC interrupt enable

HAL_I3C_IT_DAUPDIE

ENTDAA/RSTDAA/SETNEWDA CCC interrupt enable

HAL_I3C_IT_MWLUPDIE

SETMWL CCC interrupt enable

HAL_I3C_IT_MRLUPDIE

SETMRL CCC interrupt enable

HAL_I3C_IT_RSTIE

reset pattern interrupt enable

HAL_I3C_IT_ASUPDIE

ENTASx CCC interrupt enable

HAL_I3C_IT_INTUPDIE

ENEC/DISEC CCC interrupt enable

HAL_I3C_IT_DEFIE

DEFTGTS CCC interrupt enable

HAL_I3C_IT_GRPIE

DEFGRPA CCC interrupt enable

HAL_I3C_ALL_TGT_ITS

TURNAROUND TIME TSCO

HAL_I3C_TURNAROUND_TIME_TSCO_LESS_12NS

clock-to-data turnaround time tSCO <= 12ns

HAL_I3C_TURNAROUND_TIME_TSCO_GREATER_12NS

clock-to-data turnaround time tSCO > 12ns

TX FIFO THRESHOLD

HAL_I3C_TXFIFO_THRESHOLD_1_4

Tx Fifo Threshold is 1 byte

HAL_I3C_TXFIFO_THRESHOLD_4_4

Tx Fifo Threshold is 4 bytes

I3C Utility Bus Type

I3C_PURE_I3C_BUS

Pure I3C bus, no I2C

I3C_MIXED_BUS

Mixed bus I3C and I2C

37 HAL ICACHE Generic Driver

37.1 ICACHE Firmware driver API description

The following section lists the various functions of the ICACHE library.

37.1.1 How to use this driver

The Texture Cache (ICACHE) is introduced on AXI read-only texture port of the GPU2D to improve performance when reading texture data from internal and external memories.

The ICACHE HAL driver can be used as follows:

1. Optionally configure the Instruction Cache mode with HAL_ICACHE_ConfigAssociativityMode() if the default configuration does not suit the application requirements.
2. Enable and disable the Instruction Cache with respectively HAL_ICACHE_Enable() and HAL_ICACHE_Disable(). Use HAL_ICACHE_IsEnabled() to get the Instruction Cache status. To ensure a deterministic cache behavior after power on, system reset or after a call to HAL_ICACHE_Disable(), the application must call HAL_ICACHE_WaitForInvalidateComplete(). Indeed on power on, system reset or cache disable, an automatic cache invalidation procedure is launched and the cache is bypassed until the operation completes.
3. Initiate the cache maintenance invalidation procedure with either HAL_ICACHE_Invalidate() (blocking mode) or HAL_ICACHE_Invalidate_IT() (interrupt mode). When interrupt mode is used, the callback function HAL_ICACHE_InvalidateCompleteCallback() is called when the invalidate procedure is complete. The function HAL_ICACHE_WaitForInvalidateComplete() may be called to wait for the end of the invalidate procedure automatically initiated when disabling the Instruction Cache with HAL_ICACHE_Disable(). The cache operation is bypassed during the invalidation procedure.
4. Use the performance monitoring counters for Hit and Miss with the following functions: HAL_ICACHE_Monitor_Start(), HAL_ICACHE_Monitor_Stop(), HAL_ICACHE_Monitor_Reset(), HAL_ICACHE_Monitor_GetHitValue() and HAL_ICACHE_Monitor_GetMissValue()

37.1.2 Detailed description of functions

HAL_ICACHE_Enable

Function name

HAL_StatusTypeDef HAL_ICACHE_Enable (void)

Function description

HAL_ICACHE_Disable

Function name

HAL_StatusTypeDef HAL_ICACHE_Disable (void)

Function description

HAL_ICACHE_IsEnabled

Function name

uint32_t HAL_ICACHE_IsEnabled (void)

Function description

HAL_ICACHE_ConfigAssociativityMode

Function name

HAL_StatusTypeDef HAL_ICACHE_ConfigAssociativityMode (uint32_t AssociativityMode)

Function description

HAL_ICACHE_DeInit

Function name

HAL_StatusTypeDef HAL_ICACHE_DeInit (void)

Function description

HAL_ICACHE_Invalidate

Function name

HAL_StatusTypeDef HAL_ICACHE_Invalidate (void)

Function description

HAL_ICACHE_Invalidate_IT

Function name

HAL_StatusTypeDef HAL_ICACHE_Invalidate_IT (void)

Function description

HAL_ICACHE_WaitForInvalidateComplete

Function name

HAL_StatusTypeDef HAL_ICACHE_WaitForInvalidateComplete (void)

Function description

HAL_ICACHE_Monitor_Start

Function name

HAL_StatusTypeDef HAL_ICACHE_Monitor_Start (uint32_t MonitorType)

Function description

HAL_ICACHE_Monitor_Stop

Function name

HAL_StatusTypeDef HAL_ICACHE_Monitor_Stop (uint32_t MonitorType)

Function description

HAL_ICACHE_Monitor_Reset

Function name

HAL_StatusTypeDef HAL_ICACHE_Monitor_Reset (uint32_t MonitorType)

Function description

HAL_ICACHE_Monitor_GetHitValue

Function name

uint32_t HAL_ICACHE_Monitor_GetHitValue (void)

Function description

HAL_ICACHE_Monitor_GetMissValue

Function name

uint32_t HAL_ICACHE_Monitor_GetMissValue (void)

Function description

HAL_ICACHE_IRQHandler

Function name

void HAL_ICACHE_IRQHandler (void)

Function description

HAL_ICACHE_InvalidateCompleteCallback

Function name

void HAL_ICACHE_InvalidateCompleteCallback (void)

Function description

HAL_ICACHE_ErrorCallback

Function name

void HAL_ICACHE_ErrorCallback (void)

Function description

37.2 ICACHE Firmware driver defines

The following section lists the various define and macros of the module.

37.2.1 ICACHE

ICACHE

Flags

ICACHE_FLAG_BUSY

Busy flag

ICACHE_FLAG_BUSYEND

Busy end flag

ICACHE_FLAG_ERROR

Cache error flag

Flags and Interrupts Management

__HAL_ICACHE_ENABLE_IT

Description:

- Enable ICACHE interrupts.

Parameters:

- __INTERRUPT__: specifies the ICACHE interrupt sources to be enabled. This parameter can be any combination of the following values:
  - ICACHE_IT_BUSYEND Busy end interrupt
  - ICACHE_IT_ERROR Cache error interrupt

__HAL_ICACHE_DISABLE_IT

Description:

- Disable ICACHE interrupts.

Parameters:

- __INTERRUPT__: specifies the ICACHE interrupt sources to be disabled. This parameter can be any combination of the following values:
  - ICACHE_IT_BUSYEND Busy end interrupt
  - ICACHE_IT_ERROR Cache error interrupt

__HAL_ICACHE_GET_IT_SOURCE

Description:

- Check whether the specified ICACHE interrupt source is enabled or not.

Parameters:

- __INTERRUPT__: specifies the ICACHE interrupt source to check. This parameter can be any combination of the following values:
  - ICACHE_IT_BUSYEND Busy end interrupt
  - ICACHE_IT_ERROR Cache error interrupt

Return value:

- The: state of __INTERRUPT__ (0 or 1).

__HAL_ICACHE_GET_FLAG

Description:

- Check whether the selected ICACHE flag is set or not.

Parameters:

- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - ICACHE_FLAG_BUSY Busy flag
  - ICACHE_FLAG_BUSYEND Busy end flag
  - ICACHE_FLAG_ERROR Cache error flag

Return value:

- The: state of __FLAG__ (0 or 1).

__HAL_ICACHE_CLEAR_FLAG

Description:

- Clear the selected ICACHE flags.

Parameters:

- __FLAG__: specifies the ICACHE flags to clear. This parameter can be any combination of the following values:
  - ICACHE_FLAG_BUSYEND Busy end flag
  - ICACHE_FLAG_ERROR Cache error flag

Interrupts

ICACHE_IT_BUSYEND

Busy end interrupt

ICACHE_IT_ERROR

Cache error interrupt

Monitor type

ICACHE_MONITOR_HIT_MISS

Hit & Miss monitoring

ICACHE_MONITOR_HIT

Hit monitoring

ICACHE_MONITOR_MISS

Miss monitoring

Ways selection

ICACHE_1WAY

1-way cache (direct mapped cache)

ICACHE_4WAYS

4-ways set associative cache (default)

38 HAL IRDA Generic Driver

38.1 IRDA Firmware driver registers structures

38.1.1 IRDA_InitTypeDef

IRDA_InitTypeDef is defined in the stm32n6xx_hal_irda.h

Data Fields

- ***uint32_t BaudRate***
- ***uint32_t WordLength***
- ***uint32_t Parity***
- ***uint32_t Mode***
- ***uint8_t Prescaler***
- ***uint16_t PowerMode***
- ***uint32_t ClockPrescaler***

Field Documentation

- ***uint32_t IRDA_InitTypeDef::BaudRate***
  This member configures the IRDA communication baud rate. The baud rate register is computed using the following formula: Baud Rate Register = ((usart_ker_ckpres) / ((hirda->Init.BaudRate))) where usart_ker_ckpres is the IRDA input clock divided by a prescaler
- ***uint32_t IRDA_InitTypeDef::WordLength***
  Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of ***IRDAEx_Word_Length***
- ***uint32_t IRDA_InitTypeDef::Parity***
  Specifies the parity mode. This parameter can be a value of ***IRDA_Parity***
  Note:
  – When parity is enabled, the computed parity is inserted at the MSB position of the transmitted data (9th bit when the word length is set to 9 data bits; 8th bit when the word length is set to 8 data bits).
- ***uint32_t IRDA_InitTypeDef::Mode***
  Specifies whether the Receive or Transmit mode is enabled or disabled. This parameter can be a value of ***IRDA_Transfer_Mode***
- ***uint8_t IRDA_InitTypeDef::Prescaler***
  Specifies the Prescaler value for dividing the UART/USART source clock to achieve low-power frequency.
  Note:
  – Prescaler value 0 is forbidden
- ***uint16_t IRDA_InitTypeDef::PowerMode***
  Specifies the IRDA power mode. This parameter can be a value of ***IRDA_Low_Power***
- ***uint32_t IRDA_InitTypeDef::ClockPrescaler***
  Specifies the prescaler value used to divide the IRDA clock source. This parameter can be a value of ***IRDA_ClockPrescaler***.

38.1.2 __IRDA_HandleTypeDef

__IRDA_HandleTypeDef is defined in the stm32n6xx_hal_irda.h

Data Fields

- ***USART_TypeDef * Instance***
- ***IRDA_InitTypeDef Init***
- ***const uint8_t * pTxBuffPtr***
- ***uint16_t TxXferSize***
- ***__IO uint16_t TxXferCount***
- ***uint8_t * pRxBuffPtr***
- ***uint16_t RxXferSize***
- ***__IO uint16_t RxXferCount***

- ***uint16_t Mask***
- ***DMA_HandleTypeDef * hdmatx***
- ***DMA_HandleTypeDef * hdmarx***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_IRDA_StateTypeDef gState***
- ***__IO HAL_IRDA_StateTypeDef RxState***
- ***__IO uint32_t ErrorCode***
- ***void(* TxHalfCpltCallback***
- ***void(* TxCpltCallback***
- ***void(* RxHalfCpltCallback***
- ***void(* RxCpltCallback***
- ***void(* ErrorCallback***
- ***void(* AbortCpltCallback***
- ***void(* AbortTransmitCpltCallback***
- ***void(* AbortReceiveCpltCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***USART_TypeDef* __IRDA_HandleTypeDef::Instance***
  USART registers base address
- ***IRDA_InitTypeDef __IRDA_HandleTypeDef::Init***
  IRDA communication parameters
- ***const uint8_t* __IRDA_HandleTypeDef::pTxBuffPtr***
  Pointer to IRDA Tx transfer Buffer
- ***uint16_t __IRDA_HandleTypeDef::TxXferSize***
  IRDA Tx Transfer size
- ***__IO uint16_t __IRDA_HandleTypeDef::TxXferCount***
  IRDA Tx Transfer Counter
- ***uint8_t* __IRDA_HandleTypeDef::pRxBuffPtr***
  Pointer to IRDA Rx transfer Buffer
- ***uint16_t __IRDA_HandleTypeDef::RxXferSize***
  IRDA Rx Transfer size
- ***__IO uint16_t __IRDA_HandleTypeDef::RxXferCount***
  IRDA Rx Transfer Counter
- ***uint16_t __IRDA_HandleTypeDef::Mask***
  USART RX RDR register mask
- ***DMA_HandleTypeDef* __IRDA_HandleTypeDef::hdmatx***
  IRDA Tx DMA Handle parameters
- ***DMA_HandleTypeDef* __IRDA_HandleTypeDef::hdmarx***
  IRDA Rx DMA Handle parameters
- ***HAL_LockTypeDef __IRDA_HandleTypeDef::Lock***
  Locking object
- ***__IO HAL_IRDA_StateTypeDef __IRDA_HandleTypeDef::gState***
  IRDA state information related to global Handle management and also related to Tx operations. This parameter can be a value of **HAL_IRDA_StateTypeDef**
- ***__IO HAL_IRDA_StateTypeDef __IRDA_HandleTypeDef::RxState***
  IRDA state information related to Rx operations. This parameter can be a value of **HAL_IRDA_StateTypeDef**
- ***__IO uint32_t __IRDA_HandleTypeDef::ErrorCode***
  IRDA Error code
- ***void(* __IRDA_HandleTypeDef::TxHalfCpltCallback)(struct __IRDA_HandleTypeDef *hirda)***
  IRDA Tx Half Complete Callback
- ***void(* __IRDA_HandleTypeDef::TxCpltCallback)(struct __IRDA_HandleTypeDef *hirda)***
  IRDA Tx Complete Callback

- ***void(* __IRDA_HandleTypeDef::RxHalfCpltCallback)(struct __IRDA_HandleTypeDef *hirda)***
  IRDA Rx Half Complete Callback
- ***void(* __IRDA_HandleTypeDef::RxCpltCallback)(struct __IRDA_HandleTypeDef *hirda)***
  IRDA Rx Complete Callback
- ***void(* __IRDA_HandleTypeDef::ErrorCallback)(struct __IRDA_HandleTypeDef *hirda)***
  IRDA Error Callback
- ***void(* __IRDA_HandleTypeDef::AbortCpltCallback)(struct __IRDA_HandleTypeDef *hirda)***
  IRDA Abort Complete Callback
- ***void(* __IRDA_HandleTypeDef::AbortTransmitCpltCallback)(struct __IRDA_HandleTypeDef *hirda)***
  IRDA Abort Transmit Complete Callback
- ***void(* __IRDA_HandleTypeDef::AbortReceiveCpltCallback)(struct __IRDA_HandleTypeDef *hirda)***
  IRDA Abort Receive Complete Callback
- ***void(* __IRDA_HandleTypeDef::MspInitCallback)(struct __IRDA_HandleTypeDef *hirda)***
  IRDA Msp Init callback
- ***void(* __IRDA_HandleTypeDef::MspDeInitCallback)(struct __IRDA_HandleTypeDef *hirda)***
  IRDA Msp DeInit callback

38.2 IRDA Firmware driver API description

The following section lists the various functions of the IRDA library.

38.2.1 How to use this driver

The IRDA HAL driver can be used as follows:

1. Declare a IRDA_HandleTypeDef handle structure (eg. IRDA_HandleTypeDef hirda).
2. Initialize the IRDA low level resources by implementing the HAL_IRDA_MspInit() API in setting the associated USART or UART in IRDA mode:
   - Enable the USARTx/UARTx interface clock.
   - USARTx/UARTx pins configuration:
     - Enable the clock for the USARTx/UARTx GPIOs.
     - Configure these USARTx/UARTx pins (TX as alternate function pull-up, RX as alternate function Input).
   - NVIC configuration if you need to use interrupt process (HAL_IRDA_Transmit_IT() and HAL_IRDA_Receive_IT() APIs):
     - Configure the USARTx/UARTx interrupt priority.
     - Enable the NVIC USARTx/UARTx IRQ handle.
     - The specific IRDA interrupts (Transmission complete interrupt, RXNE interrupt and Error Interrupts) will be managed using the macros __HAL_IRDA_ENABLE_IT() and __HAL_IRDA_DISABLE_IT() inside the transmit and receive process.
   - DMA Configuration if you need to use DMA process (HAL_IRDA_Transmit_DMA() and HAL_IRDA_Receive_DMA() APIs):
     - Declare a DMA handle structure for the Tx/Rx channel.
     - Enable the DMAx interface clock.
     - Configure the declared DMA handle structure with the required Tx/Rx parameters.
     - Configure the DMA Tx/Rx channel.
     - Associate the initialized DMA handle to the IRDA DMA Tx/Rx handle.
     - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx/Rx channel.
3. Program the Baud Rate, Word Length and Parity and Mode(Receiver/Transmitter), the normal or low power mode and the clock prescaler in the hirda handle Init structure.

4. Initialize the IRDA registers by calling the HAL_IRDA_Init() API:
   – This API configures also the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customized HAL_IRDA_MspInit() API.

Note: The specific IRDA interrupts (Transmission complete interrupt, RXNE interrupt and Error Interrupts) will be managed using the macros __HAL_IRDA_ENABLE_IT() and __HAL_IRDA_DISABLE_IT() inside the transmit and receive process.

5. Three operation modes are available within this driver :

Polling mode IO operation

- Send an amount of data in blocking mode using HAL_IRDA_Transmit()
- Receive an amount of data in blocking mode using HAL_IRDA_Receive()

Interrupt mode IO operation

- Send an amount of data in non-blocking mode using HAL_IRDA_Transmit_IT()
- At transmission end of transfer HAL_IRDA_TxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_IRDA_TxCpltCallback()
- Receive an amount of data in non-blocking mode using HAL_IRDA_Receive_IT()
- At reception end of transfer HAL_IRDA_RxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_IRDA_RxCpltCallback()
- In case of transfer Error, HAL_IRDA_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_IRDA_ErrorCallback()

DMA mode IO operation

- Send an amount of data in non-blocking mode (DMA) using HAL_IRDA_Transmit_DMA()
- At transmission half of transfer HAL_IRDA_TxHalfCpltCallback() is executed and user can add his own code by customization of function pointer HAL_IRDA_TxHalfCpltCallback()
- At transmission end of transfer HAL_IRDA_TxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_IRDA_TxCpltCallback()
- Receive an amount of data in non-blocking mode (DMA) using HAL_IRDA_Receive_DMA()
- At reception half of transfer HAL_IRDA_RxHalfCpltCallback() is executed and user can add his own code by customization of function pointer HAL_IRDA_RxHalfCpltCallback()
- At reception end of transfer HAL_IRDA_RxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_IRDA_RxCpltCallback()
- In case of transfer Error, HAL_IRDA_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_IRDA_ErrorCallback()

IRDA HAL driver macros list

Below the list of most used macros in IRDA HAL driver.

- __HAL_IRDA_ENABLE: Enable the IRDA peripheral
- __HAL_IRDA_DISABLE: Disable the IRDA peripheral
- __HAL_IRDA_GET_FLAG : Check whether the specified IRDA flag is set or not
- __HAL_IRDA_CLEAR_FLAG : Clear the specified IRDA pending flag
- __HAL_IRDA_ENABLE_IT: Enable the specified IRDA interrupt
- __HAL_IRDA_DISABLE_IT: Disable the specified IRDA interrupt
- __HAL_IRDA_GET_IT_SOURCE: Check whether or not the specified IRDA interrupt is enabled

Note: You can refer to the IRDA HAL driver header file for more useful macros

38.2.2 Callback registration

The compilation define USE_HAL_IRDA_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks.

Use Function HAL_IRDA_RegisterCallback() to register a user callback. Function HAL_IRDA_RegisterCallback() allows to register following callbacks:

- TxHalfCpltCallback : Tx Half Complete Callback.
- TxCpltCallback : Tx Complete Callback.
- RxHalfCpltCallback : Rx Half Complete Callback.
- RxCpltCallback : Rx Complete Callback.
- ErrorCallback : Error Callback.
- AbortCpltCallback : Abort Complete Callback.
- AbortTransmitCpltCallback : Abort Transmit Complete Callback.
- AbortReceiveCpltCallback : Abort Receive Complete Callback.
- MspInitCallback : IRDA MspInit.
- MspDeInitCallback : IRDA MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

Use function HAL_IRDA_UnRegisterCallback() to reset a callback to the default weak function. HAL_IRDA_UnRegisterCallback() takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks:

- TxHalfCpltCallback : Tx Half Complete Callback.
- TxCpltCallback : Tx Complete Callback.
- RxHalfCpltCallback : Rx Half Complete Callback.
- RxCpltCallback : Rx Complete Callback.
- ErrorCallback : Error Callback.
- AbortCpltCallback : Abort Complete Callback.
- AbortTransmitCpltCallback : Abort Transmit Complete Callback.
- AbortReceiveCpltCallback : Abort Receive Complete Callback.
- MspInitCallback : IRDA MspInit.
- MspDeInitCallback : IRDA MspDeInit.

By default, after the HAL_IRDA_Init() and when the state is HAL_IRDA_STATE_RESET all callbacks are set to the corresponding weak functions: examples HAL_IRDA_TxCpltCallback(), HAL_IRDA_RxHalfCpltCallback(). Exception done for MspInit and MspDeInit functions that are respectively reset to the legacy weak functions in the HAL_IRDA_Init() and HAL_IRDA_DeInit() only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the HAL_IRDA_Init() and HAL_IRDA_DeInit() keep and use the user MspInit/ MspDeInit callbacks (registered beforehand).

Callbacks can be registered/unregistered in HAL_IRDA_STATE_READY state only. Exception done MspInit/ MspDeInit that can be registered/unregistered in HAL_IRDA_STATE_READY or HAL_IRDA_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_IRDA_RegisterCallback() before calling HAL_IRDA_DeInit() or HAL_IRDA_Init() function.

When The compilation define USE_HAL_IRDA_REGISTER_CALLBACKS is set to 0 or not defined, the callback registration feature is not available and weak callbacks are used.

38.2.3 Detailed description of functions

HAL_IRDA_Init

Function name

HAL_StatusTypeDef HAL_IRDA_Init (IRDA_HandleTypeDef * hirda)

Function description

HAL_IRDA_DeInit

Function name

HAL_StatusTypeDef HAL_IRDA_DeInit (IRDA_HandleTypeDef * hirda)

Function description

HAL_IRDA_MspInit

Function name

void HAL_IRDA_MspInit (IRDA_HandleTypeDef * hirda)

Function description

HAL_IRDA_MspDeInit

Function name

void HAL_IRDA_MspDeInit (IRDA_HandleTypeDef * hirda)

Function description

HAL_IRDA_RegisterCallback

Function name

HAL_StatusTypeDef HAL_IRDA_RegisterCallback (IRDA_HandleTypeDef * hirda, HAL_IRDA_CallbackIDTypeDef CallbackID, pIRDA_CallbackTypeDef pCallback)

Function description

HAL_IRDA_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_IRDA_UnRegisterCallback (IRDA_HandleTypeDef * hirda, HAL_IRDA_CallbackIDTypeDef CallbackID)

Function description

HAL_IRDA_Transmit

Function name

HAL_StatusTypeDef HAL_IRDA_Transmit (IRDA_HandleTypeDef * hirda, const uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

HAL_IRDA_Receive

Function name

HAL_StatusTypeDef HAL_IRDA_Receive (IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

HAL_IRDA_Transmit_IT

Function name

HAL_StatusTypeDef HAL_IRDA_Transmit_IT (IRDA_HandleTypeDef * hirda, const uint8_t * pData, uint16_t Size)

Function description

HAL_IRDA_Receive_IT

Function name

HAL_StatusTypeDef HAL_IRDA_Receive_IT (IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size)

Function description

HAL_IRDA_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_IRDA_Transmit_DMA (IRDA_HandleTypeDef * hirda, const uint8_t * pData, uint16_t Size)

Function description

HAL_IRDA_Receive_DMA

Function name

HAL_StatusTypeDef HAL_IRDA_Receive_DMA (IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size)

Function description

HAL_IRDA_DMAPause

Function name

HAL_StatusTypeDef HAL_IRDA_DMAPause (IRDA_HandleTypeDef * hirda)

Function description

HAL_IRDA_DMAResume

Function name

HAL_StatusTypeDef HAL_IRDA_DMAResume (IRDA_HandleTypeDef * hirda)

Function description

HAL_IRDA_DMAStop

Function name

HAL_StatusTypeDef HAL_IRDA_DMAStop (IRDA_HandleTypeDef * hirda)

Function description

HAL_IRDA_Abort

Function name

HAL_StatusTypeDef HAL_IRDA_Abort (IRDA_HandleTypeDef * hirda)

Function description

HAL_IRDA_AbortTransmit

Function name

HAL_StatusTypeDef HAL_IRDA_AbortTransmit (IRDA_HandleTypeDef * hirda)

Function description

HAL_IRDA_AbortReceive

Function name

HAL_StatusTypeDef HAL_IRDA_AbortReceive (IRDA_HandleTypeDef * hirda)

Function description

HAL_IRDA_Abort_IT

Function name

HAL_StatusTypeDef HAL_IRDA_Abort_IT (IRDA_HandleTypeDef * hirda)

Function description

HAL_IRDA_AbortTransmit_IT

Function name

HAL_StatusTypeDef HAL_IRDA_AbortTransmit_IT (IRDA_HandleTypeDef * hirda)

Function description

HAL_IRDA_AbortReceive_IT

Function name

HAL_StatusTypeDef HAL_IRDA_AbortReceive_IT (IRDA_HandleTypeDef * hirda)

Function description

HAL_IRDA_IRQHandler

Function name

void HAL_IRDA_IRQHandler (IRDA_HandleTypeDef * hirda)

Function description

HAL_IRDA_TxCpltCallback

Function name

void HAL_IRDA_TxCpltCallback (IRDA_HandleTypeDef * hirda)

Function description

HAL_IRDA_RxCpltCallback

Function name

void HAL_IRDA_RxCpltCallback (IRDA_HandleTypeDef * hirda)

Function description

HAL_IRDA_TxHalfCpltCallback

Function name

void HAL_IRDA_TxHalfCpltCallback (IRDA_HandleTypeDef * hirda)

Function description

HAL_IRDA_RxHalfCpltCallback

Function name

void HAL_IRDA_RxHalfCpltCallback (IRDA_HandleTypeDef * hirda)

Function description

HAL_IRDA_ErrorCallback

Function name

void HAL_IRDA_ErrorCallback (IRDA_HandleTypeDef * hirda)

Function description

HAL_IRDA_AbortCpltCallback

Function name

void HAL_IRDA_AbortCpltCallback (IRDA_HandleTypeDef * hirda)

Function description

HAL_IRDA_AbortTransmitCpltCallback

Function name

void HAL_IRDA_AbortTransmitCpltCallback (IRDA_HandleTypeDef * hirda)

Function description

HAL_IRDA_AbortReceiveCpltCallback

Function name

void HAL_IRDA_AbortReceiveCpltCallback (IRDA_HandleTypeDef * hirda)

Function description

HAL_IRDA_GetState

Function name

HAL_IRDA_StateTypeDef HAL_IRDA_GetState (const IRDA_HandleTypeDef * hirda)

Function description

HAL_IRDA_GetError

Function name

uint32_t HAL_IRDA_GetError (const IRDA_HandleTypeDef * hirda)

Function description

38.3 IRDA Firmware driver defines

The following section lists the various define and macros of the module.

38.3.1 IRDA

IRDA

IRDA Clock Prescaler

IRDA_PRESCALER_DIV1

fclk_pres = fclk

IRDA_PRESCALER_DIV2

fclk_pres = fclk/2

IRDA_PRESCALER_DIV4

fclk_pres = fclk/4

IRDA_PRESCALER_DIV6

fclk_pres = fclk/6

IRDA_PRESCALER_DIV8

fclk_pres = fclk/8

IRDA_PRESCALER_DIV10

fclk_pres = fclk/10

IRDA_PRESCALER_DIV12

fclk_pres = fclk/12

IRDA_PRESCALER_DIV16

fclk_pres = fclk/16

IRDA_PRESCALER_DIV32

fclk_pres = fclk/32

IRDA_PRESCALER_DIV64

fclk_pres = fclk/64

IRDA_PRESCALER_DIV128

fclk_pres = fclk/128

IRDA_PRESCALER_DIV256

fclk_pres = fclk/256

IRDA DMA Rx

IRDA_DMA_RX_DISABLE

IRDA DMA RX disabled

IRDA_DMA_RX_ENABLE

IRDA DMA RX enabled

IRDA DMA Tx

IRDA_DMA_TX_DISABLE

IRDA DMA TX disabled

IRDA_DMA_TX_ENABLE

IRDA DMA TX enabled

IRDA Error Code Definition

HAL_IRDA_ERROR_NONE

No error

HAL_IRDA_ERROR_PE

Parity error

HAL_IRDA_ERROR_NE

Noise error

HAL_IRDA_ERROR_FE

frame error

HAL_IRDA_ERROR_ORE

Overrun error

HAL_IRDA_ERROR_DMA

DMA transfer error

HAL_IRDA_ERROR_BUSY

Busy Error

HAL_IRDA_ERROR_INVALID_CALLBACK

Invalid Callback error

IRDA Exported Macros

__HAL_IRDA_RESET_HANDLE_STATE

Description:

- Reset IRDA handle state.

Parameters:

- __HANDLE__: IRDA handle.

Return value:

- None

__HAL_IRDA_FLUSH_DRREGISTER

Description:

- Flush the IRDA DR register.

Parameters:

- __HANDLE__: specifies the IRDA Handle.

Return value:

- None

__HAL_IRDA_CLEAR_FLAG

Description:

- Clear the specified IRDA pending flag.

Parameters:

- __HANDLE__: specifies the IRDA Handle.
- __FLAG__: specifies the flag to check. This parameter can be any combination of the following values:
  - IRDA_CLEAR_PEF
  - IRDA_CLEAR_FEF
  - IRDA_CLEAR_NEF
  - IRDA_CLEAR_OREF
  - IRDA_CLEAR_TCF
  - IRDA_CLEAR_IDLEF

Return value:

- None

__HAL_IRDA_CLEAR_PEFLAG

Description:

- Clear the IRDA PE pending flag.

Parameters:

- __HANDLE__: specifies the IRDA Handle.

Return value:

- None

__HAL_IRDA_CLEAR_FEFLAG

Description:

- Clear the IRDA FE pending flag.

Parameters:

- __HANDLE__: specifies the IRDA Handle.

Return value:

- None

__HAL_IRDA_CLEAR_NEFLAG

Description:

- Clear the IRDA NE pending flag.

Parameters:

- __HANDLE__: specifies the IRDA Handle.

Return value:

- None

__HAL_IRDA_CLEAR_OREFLAG

Description:

- Clear the IRDA ORE pending flag.

Parameters:

- __HANDLE__: specifies the IRDA Handle.

Return value:

- None

__HAL_IRDA_CLEAR_IDLEFLAG

Description:

- Clear the IRDA IDLE pending flag.

Parameters:

- __HANDLE__: specifies the IRDA Handle.

Return value:

- None

__HAL_IRDA_GET_FLAG

Description:

- Check whether the specified IRDA flag is set or not.

Parameters:

- __HANDLE__: specifies the IRDA Handle.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - IRDA_FLAG_REACK Receive enable acknowledge flag
  - IRDA_FLAG_TEACK Transmit enable acknowledge flag
  - IRDA_FLAG_BUSY Busy flag
  - IRDA_FLAG_ABRF Auto Baud rate detection flag
  - IRDA_FLAG_ABRE Auto Baud rate detection error flag
  - IRDA_FLAG_TXE Transmit data register empty flag
  - IRDA_FLAG_TC Transmission Complete flag
  - IRDA_FLAG_RXNE Receive data register not empty flag
  - IRDA_FLAG_ORE OverRun Error flag
  - IRDA_FLAG_NE Noise Error flag
  - IRDA_FLAG_FE Framing Error flag
  - IRDA_FLAG_PE Parity Error flag

Return value:

- The: new state of __FLAG__ (TRUE or FALSE).

__HAL_IRDA_ENABLE_IT

Description:

- Enable the specified IRDA interrupt.

Parameters:

- __HANDLE__: specifies the IRDA Handle.
- __INTERRUPT__: specifies the IRDA interrupt source to enable. This parameter can be one of the following values:
  - IRDA_IT_TXE Transmit Data Register empty interrupt
  - IRDA_IT_TC Transmission complete interrupt
  - IRDA_IT_RXNE Receive Data register not empty interrupt
  - IRDA_IT_IDLE Idle line detection interrupt
  - IRDA_IT_PE Parity Error interrupt
  - IRDA_IT_ERR Error interrupt(Frame error, noise error, overrun error)

Return value:

- None

__HAL_IRDA_DISABLE_IT

Description:

- Disable the specified IRDA interrupt.

Parameters:

- __HANDLE__: specifies the IRDA Handle.
- __INTERRUPT__: specifies the IRDA interrupt source to disable. This parameter can be one of the following values:
  - IRDA_IT_TXE Transmit Data Register empty interrupt
  - IRDA_IT_TC Transmission complete interrupt
  - IRDA_IT_RXNE Receive Data register not empty interrupt
  - IRDA_IT_IDLE Idle line detection interrupt
  - IRDA_IT_PE Parity Error interrupt
  - IRDA_IT_ERR Error interrupt(Frame error, noise error, overrun error)

Return value:

- None

__HAL_IRDA_GET_IT

Description:

- Check whether the specified IRDA interrupt has occurred or not.

Parameters:

- __HANDLE__: specifies the IRDA Handle.
- __INTERRUPT__: specifies the IRDA interrupt source to check. This parameter can be one of the following values:
  - IRDA_IT_TXE Transmit Data Register empty interrupt
  - IRDA_IT_TC Transmission complete interrupt
  - IRDA_IT_RXNE Receive Data register not empty interrupt
  - IRDA_IT_IDLE Idle line detection interrupt
  - IRDA_IT_ORE OverRun Error interrupt
  - IRDA_IT_NE Noise Error interrupt
  - IRDA_IT_FE Framing Error interrupt
  - IRDA_IT_PE Parity Error interrupt

Return value:

- The: new state of __IT__ (SET or RESET).

__HAL_IRDA_GET_IT_SOURCE

Description:

- Check whether the specified IRDA interrupt source is enabled or not.

Parameters:

- __HANDLE__: specifies the IRDA Handle.
- __INTERRUPT__: specifies the IRDA interrupt source to check. This parameter can be one of the following values:
  - IRDA_IT_TXE Transmit Data Register empty interrupt
  - IRDA_IT_TC Transmission complete interrupt
  - IRDA_IT_RXNE Receive Data register not empty interrupt
  - IRDA_IT_IDLE Idle line detection interrupt
  - IRDA_IT_ERR Framing, overrun or noise error interrupt
  - IRDA_IT_PE Parity Error interrupt

Return value:

- The: new state of __IT__ (SET or RESET).

__HAL_IRDA_CLEAR_IT

Description:

- Clear the specified IRDA ISR flag, in setting the proper ICR register flag.

Parameters:

- __HANDLE__: specifies the IRDA Handle.
- __IT_CLEAR__: specifies the interrupt clear register flag that needs to be set to clear the corresponding interrupt This parameter can be one of the following values:
  - IRDA_CLEAR_PEF Parity Error Clear Flag
  - IRDA_CLEAR_FEF Framing Error Clear Flag
  - IRDA_CLEAR_NEF Noise detected Clear Flag
  - IRDA_CLEAR_OREF OverRun Error Clear Flag
  - IRDA_CLEAR_TCF Transmission Complete Clear Flag

Return value:

- None

__HAL_IRDA_SEND_REQ

Description:

- Set a specific IRDA request flag.

Parameters:

- __HANDLE__: specifies the IRDA Handle.
- __REQ__: specifies the request flag to set This parameter can be one of the following values:
  - IRDA_AUTOBAUD_REQUEST Auto-Baud Rate Request
  - IRDA_RXDATA_FLUSH_REQUEST Receive Data flush Request
  - IRDA_TXDATA_FLUSH_REQUEST Transmit data flush Request

Return value:

- None

__HAL_IRDA_ONE_BIT_SAMPLE_ENABLE

Description:

- Enable the IRDA one bit sample method.

Parameters:

- __HANDLE__: specifies the IRDA Handle.

Return value:

- None

__HAL_IRDA_ONE_BIT_SAMPLE_DISABLE

Description:

- Disable the IRDA one bit sample method.

Parameters:

- __HANDLE__: specifies the IRDA Handle.

Return value:

- None

__HAL_IRDA_ENABLE

Description:

- Enable UART/USART associated to IRDA Handle.

Parameters:

- __HANDLE__: specifies the IRDA Handle.

Return value:

- None

__HAL_IRDA_DISABLE

Description:

- Disable UART/USART associated to IRDA Handle.

Parameters:

- __HANDLE__: specifies the IRDA Handle.

Return value:

- None

IRDA Flags

IRDA_FLAG_REACK

IRDA receive enable acknowledge flag

IRDA_FLAG_TEACK

IRDA transmit enable acknowledge flag

IRDA_FLAG_BUSY

IRDA busy flag

IRDA_FLAG_ABRF

IRDA auto Baud rate flag

IRDA_FLAG_ABRE

IRDA auto Baud rate error

IRDA_FLAG_TXE

IRDA transmit data register empty

IRDA_FLAG_TC

IRDA transmission complete

IRDA_FLAG_RXNE

IRDA read data register not empty

IRDA_FLAG_ORE

IRDA overrun error

IRDA_FLAG_NE

IRDA noise error

IRDA_FLAG_FE

IRDA frame error

IRDA_FLAG_PE

IRDA parity error

IRDA interruptions flags mask

IRDA_IT_MASK

IRDA Interruptions flags mask

IRDA_CR_MASK

IRDA control register mask

IRDA_CR_POS

IRDA control register position

IRDA_ISR_MASK

IRDA ISR register mask

IRDA_ISR_POS

IRDA ISR register position

IRDA Interrupts Definition

IRDA_IT_PE

IRDA Parity error interruption

IRDA_IT_TXE

IRDA Transmit data register empty interruption

IRDA_IT_TC

IRDA Transmission complete interruption

IRDA_IT_RXNE

IRDA Read data register not empty interruption

IRDA_IT_IDLE

IRDA Idle interruption

IRDA_IT_ERR

IRDA Error interruption

IRDA_IT_ORE

IRDA Overrun error interruption

IRDA_IT_NE

IRDA Noise error interruption

IRDA_IT_FE

IRDA Frame error interruption

IRDA Interruption Clear Flags

IRDA_CLEAR_PEF

Parity Error Clear Flag

IRDA_CLEAR_FEF

Framing Error Clear Flag

IRDA_CLEAR_NEF

Noise Error detected Clear Flag

IRDA_CLEAR_OREF

OverRun Error Clear Flag

IRDA_CLEAR_IDLEF

IDLE line detected Clear Flag

IRDA_CLEAR_TCF

Transmission Complete Clear Flag

IRDA Low Power

IRDA_POWERMODE_NORMAL

IRDA normal power mode

IRDA_POWERMODE_LOWPOWER

IRDA low power mode

IRDA Mode

IRDA_MODE_DISABLE

Associated UART disabled in IRDA mode

IRDA_MODE_ENABLE

Associated UART enabled in IRDA mode

IRDA One Bit Sampling

IRDA_ONE_BIT_SAMPLE_DISABLE

One-bit sampling disabled

IRDA_ONE_BIT_SAMPLE_ENABLE

One-bit sampling enabled

IRDA Parity

IRDA_PARITY_NONE

No parity

IRDA_PARITY_EVEN

Even parity

IRDA_PARITY_ODD

Odd parity

IRDA Request Parameters

IRDA_AUTOBAUD_REQUEST

Auto-Baud Rate Request

IRDA_RXDATA_FLUSH_REQUEST

Receive Data flush Request

IRDA_TXDATA_FLUSH_REQUEST

Transmit data flush Request

IRDA State

IRDA_STATE_DISABLE

IRDA disabled

IRDA_STATE_ENABLE

IRDA enabled

IRDA State Code Definition

HAL_IRDA_STATE_RESET

Peripheral is not initialized Value is allowed for gState and RxState

HAL_IRDA_STATE_READY

Peripheral Initialized and ready for use Value is allowed for gState and RxState

HAL_IRDA_STATE_BUSY

An internal process is ongoing Value is allowed for gState only

HAL_IRDA_STATE_BUSY_TX

Data Transmission process is ongoing Value is allowed for gState only

HAL_IRDA_STATE_BUSY_RX

Data Reception process is ongoing Value is allowed for RxState only

HAL_IRDA_STATE_BUSY_TX_RX

Data Transmission and Reception process is ongoing Not to be used for neither gState nor RxState. Value is result of combination (Or) between gState and RxState values

HAL_IRDA_STATE_TIMEOUT

Timeout state Value is allowed for gState only

HAL_IRDA_STATE_ERROR

Error Value is allowed for gState only

IRDA Transfer Mode

IRDA_MODE_RX

RX mode

IRDA_MODE_TX

TX mode

IRDA_MODE_TX_RX

RX and TX mode

39 HAL IRDA Extension Driver

39.1 IRDAEx Firmware driver defines

The following section lists the various define and macros of the module.

39.1.1 IRDAEx

IRDAEx

IRDAEx Word Length

IRDA_WORDLENGTH_7B

7-bit long frame

IRDA_WORDLENGTH_8B

8-bit long frame

IRDA_WORDLENGTH_9B

9-bit long frame

40 HAL IWDG Generic Driver

40.1 IWDG Firmware driver registers structures

40.1.1 IWDG_InitTypeDef

IWDG_InitTypeDef is defined in the stm32n6xx_hal_iwdg.h

Data Fields

- ***uint32_t Prescaler***
- ***uint32_t Reload***
- ***uint32_t Window***
- ***uint32_t EWI***

Field Documentation

- ***uint32_t IWDG_InitTypeDef::Prescaler***
  Select the prescaler of the IWDG. This parameter can be a value of ***IWDG_Prescaler***
- ***uint32_t IWDG_InitTypeDef::Reload***
  Specifies the IWDG down-counter reload value. This parameter must be a number between Min_Data = 0 and Max_Data = 0x0FFF
- ***uint32_t IWDG_InitTypeDef::Window***
  Specifies the window value to be compared to the down-counter. This parameter must be a number between Min_Data = 0 and Max_Data = 0x0FFF
- ***uint32_t IWDG_InitTypeDef::EWI***
  Specifies if IWDG Early Wakeup Interrupt is enable or not and the comparator value. This parameter must be a number between Min_Data = 0 and Max_Data = 0x0FFF value 0 means that EWI is disabled

40.1.2 __IWDG_HandleTypeDef

__IWDG_HandleTypeDef is defined in the stm32n6xx_hal_iwdg.h

Data Fields

- ***IWDG_TypeDef * Instance***
- ***IWDG_InitTypeDef Init***
- ***void(* EwiCallback***
- ***void(* MspInitCallback***

Field Documentation

- ***IWDG_TypeDef* __IWDG_HandleTypeDef::Instance***
  Register base address
- ***IWDG_InitTypeDef __IWDG_HandleTypeDef::Init***
  IWDG required parameters
- ***void(* __IWDG_HandleTypeDef::EwiCallback)(struct __IWDG_HandleTypeDef *hiwdg)***
  IWDG Early WakeUp Interrupt callback
- ***void(* __IWDG_HandleTypeDef::MspInitCallback)(struct __IWDG_HandleTypeDef *hiwdg)***
  IWDG Msp Init callback

40.2 IWDG Firmware driver API description

The following section lists the various functions of the IWDG library.

40.2.1 IWDG Generic features

- The IWDG can be started by either software or hardware (configurable through option byte).
- The IWDG is clocked by the Low-Speed Internal clock (LSI) and thus stays active even if the main clock fails.

- Once the IWDG is started, the LSI is forced ON and both cannot be disabled. The counter starts counting down from the reset value (0xFFF). When it reaches the end of count value (0x000) a reset signal is generated (IWDG reset).
- Whenever the key value 0x0000 AAAA is written in the IWDG_KR register, the IWDG_RLR value is reloaded into the counter and the watchdog reset is prevented.
- The IWDG is implemented in the VDD voltage domain that is still functional in STOP and STANDBY mode (IWDG reset can wake up the CPU from STANDBY). IWDGRST flag in RCC_CSR register can be used to inform when an IWDG reset occurs.
- Debug mode: When the microcontroller enters debug mode (core halted), the IWDG counter either continues to work normally or stops, depending on DBG_IWDG_STOP configuration bit in DBG module, accessible through __HAL_DBGMCU_FREEZE_IWDG() and __HAL_DBGMCU_UNFREEZE_IWDG() macros.

Min-max timeout value @32KHz (LSI): ~125us / ~131.04s The IWDG timeout may vary due to LSI clock frequency dispersion. STM32N6xx devices provide the capability to measure the LSI clock frequency (LSI clock is internally connected to TIM16 CH1 input capture). The measured value can be used to have an IWDG timeout with an acceptable accuracy.

Default timeout value (necessary for IWDG_SR status register update): Constant LSI_VALUE is defined based on the nominal LSI clock frequency. This frequency being subject to variations as mentioned above, the default timeout value (defined through constant HAL_IWDG_DEFAULT_TIMEOUT below) may become too short or too long. In such cases, this default timeout value can be tuned by redefining the constant LSI_VALUE at user-application level (based, for instance, on the measured LSI clock frequency as explained above).

40.2.2 How to use this driver

1. Register callback to treat Iwdg interrupt and MspInit using HAL_IWDG_RegisterCallback().
   - Provide exiting handle as first parameter.
   - Provide which callback will be registered using one value from HAL_IWDG_CallbackIDTypeDef.
   - Provide callback function pointer.
2. Use IWDG using HAL_IWDG_Init() function to :
   - Enable instance by writing Start keyword in IWDG_KEY register. LSI clock is forced ON and IWDG counter starts counting down.
   - Enable write access to configuration registers: IWDG_PR, IWDG_RLR, IWDG_WINR and EWCR.
   - Configure the IWDG prescaler and counter reload value. This reload value will be loaded in the IWDG counter each time the watchdog is reloaded, then the IWDG will start counting down from this value.
   - Depending on window parameter:
     - If Window Init parameter is same as Window register value, nothing more is done but reload counter value in order to exit function with exact time base.
     - Else modify Window register. This will automatically reload watchdog counter.
   - Depending on Early Wakeup Interrupt parameter:
     - If EWI is set to disable, comparator is set to 0, interrupt is disable & flag is clear.
     - Else modify EWCR register, setting comparator value, enable interrupt & clear flag.
   - Wait for status flags to be reset.
3. Then the application program must refresh the IWDG counter at regular intervals during normal operation to prevent an MCU reset, using HAL_IWDG_Refresh() function.

IWDG HAL driver macros list

Below the list of most used macros in IWDG HAL driver:

- __HAL_IWDG_START: Enable the IWDG peripheral
- __HAL_IWDG_RELOAD_COUNTER: Reloads IWDG counter with value defined in the reload register

40.2.3 Detailed description of functions

HAL_IWDG_Init

Function name

HAL_StatusTypeDef HAL_IWDG_Init (IWDG_HandleTypeDef * hiwdg)

Function description

HAL_IWDG_MspInit

Function name

void HAL_IWDG_MspInit (IWDG_HandleTypeDef * hiwdg)

Function description

HAL_IWDG_RegisterCallback

Function name

HAL_StatusTypeDef HAL_IWDG_RegisterCallback (IWDG_HandleTypeDef * hiwdg, HAL_IWDG_CallbackIDTypeDef CallbackID, pIWDG_CallbackTypeDef pCallback)

Function description

HAL_IWDG_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_IWDG_UnRegisterCallback (IWDG_HandleTypeDef * hiwdg, HAL_IWDG_CallbackIDTypeDef CallbackID)

Function description

HAL_IWDG_Refresh

Function name

HAL_StatusTypeDef HAL_IWDG_Refresh (IWDG_HandleTypeDef * hiwdg)

Function description

HAL_IWDG_GetActiveStatus

Function name

uint32_t HAL_IWDG_GetActiveStatus (const IWDG_HandleTypeDef * hiwdg)

Function description

HAL_IWDG_IRQHandler

Function name

void HAL_IWDG_IRQHandler (IWDG_HandleTypeDef * hiwdg)

Function description

HAL_IWDG_EarlyWakeupCallback

Function name

void HAL_IWDG_EarlyWakeupCallback (IWDG_HandleTypeDef * hiwdg)

Function description

40.3 IWDG Firmware driver defines

The following section lists the various define and macros of the module.

40.3.1 IWDG

IWDG

IWDG Active Status

IWDG_STATUS_DISABLE

IWDG_STATUS_ENABLE

IWDG Early Wakeup Interrupt Mode

IWDG_EWI_DISABLE

EWI Disable

IWDG Exported Macros

__HAL_IWDG_START

Description:

- Enable the IWDG peripheral.

Parameters:

- __HANDLE__: IWDG handle

Return value:

- None

__HAL_IWDG_RELOAD_COUNTER

Description:

- Reload IWDG counter with value defined in the reload register (write access to IWDG_PR, IWDG_RLR, IWDG_WINR and EWCR registers disabled).

Parameters:

- __HANDLE__: IWDG handle

Return value:

- None

IWDG Prescaler

IWDG_PRESCALER_4

IWDG prescaler set to 4

IWDG_PRESCALER_8

IWDG prescaler set to 8

IWDG_PRESCALER_16

IWDG prescaler set to 16

IWDG_PRESCALER_32

IWDG prescaler set to 32

IWDG_PRESCALER_64

IWDG prescaler set to 64

IWDG_PRESCALER_128

IWDG prescaler set to 128

IWDG_PRESCALER_256

IWDG prescaler set to 256

IWDG_PRESCALER_512

IWDG prescaler set to 512

IWDG_PRESCALER_1024

IWDG prescaler set to 1024

IWDG Window option

IWDG_WINDOW_DISABLE

41 HAL JPEG Generic Driver

41.1 JPEG Firmware driver registers structures

41.1.1 JPEG_ConfTypeDef

JPEG_ConfTypeDef is defined in the stm32n6xx_hal_jpeg.h

Data Fields

- ***uint32_t ColorSpace***
- ***uint32_t ChromaSubsampling***
- ***uint32_t ImageHeight***
- ***uint32_t ImageWidth***
- ***uint32_t ImageQuality***

Field Documentation

- ***uint32_t JPEG_ConfTypeDef::ColorSpace***
  Image Color space : gray-scale, YCBCR, RGB or CMYK This parameter can be a value of ***JPEG_ColorSpace***
- ***uint32_t JPEG_ConfTypeDef::ChromaSubsampling***
  Chroma Subsampling in case of YCBCR or CMYK color space, 0-> 4:4:4 , 1-> 4:2:2, 2 -> 4:1:1, 3 -> 4:2:0 This parameter can be a value of ***JPEG_ChromaSubsampling***
- ***uint32_t JPEG_ConfTypeDef::ImageHeight***
  Image height : number of lines
- ***uint32_t JPEG_ConfTypeDef::ImageWidth***
  Image width : number of pixels per line
- ***uint32_t JPEG_ConfTypeDef::ImageQuality***
  Quality of the JPEG encoding : from 1 to 100

41.1.2 JPEG_HandleTypeDef

JPEG_HandleTypeDef is defined in the stm32n6xx_hal_jpeg.h

Data Fields

- ***JPEG_TypeDef * Instance***
- ***JPEG_ConfTypeDef Conf***
- ***uint8_t * pJpegInBuffPtr***
- ***uint8_t * pJpegOutBuffPtr***
- ***__IO uint32_t JpegInCount***
- ***__IO uint32_t JpegOutCount***
- ***uint32_t InDataLength***
- ***uint32_t OutDataLength***
- ***DMA_HandleTypeDef * hdmain***
- ***DMA_HandleTypeDef * hdmaout***
- ***uint8_t CustomQuanTable***
- ***uint8_t * QuantTable0***
- ***uint8_t * QuantTable1***
- ***uint8_t * QuantTable2***
- ***uint8_t * QuantTable3***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_JPEG_STATETypeDef State***
- ***__IO uint32_t ErrorCode***
- ***__IO uint32_t Context***

Field Documentation

- ***JPEG_TypeDef* JPEG_HandleTypeDef::Instance***
  JPEG peripheral register base address
- ***JPEG_ConfTypeDef JPEG_HandleTypeDef::Conf***
  Current JPEG encoding/decoding parameters
- ***uint8_t* JPEG_HandleTypeDef::pJpegInBuffPtr***
  Pointer to JPEG processing (encoding, decoding,...) input buffer
- ***uint8_t* JPEG_HandleTypeDef::pJpegOutBuffPtr***
  Pointer to JPEG processing (encoding, decoding,...) output buffer
- ***__IO uint32_t JPEG_HandleTypeDef::JpegInCount***
  Internal Counter of input data
- ***__IO uint32_t JPEG_HandleTypeDef::JpegOutCount***
  Internal Counter of output data
- ***uint32_t JPEG_HandleTypeDef::InDataLength***
  Input Buffer Length in Bytes
- ***uint32_t JPEG_HandleTypeDef::OutDataLength***
  Output Buffer Length in Bytes
- ***DMA_HandleTypeDef* JPEG_HandleTypeDef::hdmain***
  JPEG In DMA handle parameters
- ***DMA_HandleTypeDef* JPEG_HandleTypeDef::hdmaout***
  JPEG Out DMA handle parameters
- ***uint8_t JPEG_HandleTypeDef::CustomQuanTable***
  If set to 1 specify that user customized quantization tables are used
- ***uint8_t* JPEG_HandleTypeDef::QuantTable0***
  Basic Quantization Table for component 0
- ***uint8_t* JPEG_HandleTypeDef::QuantTable1***
  Basic Quantization Table for component 1
- ***uint8_t* JPEG_HandleTypeDef::QuantTable2***
  Basic Quantization Table for component 2
- ***uint8_t* JPEG_HandleTypeDef::QuantTable3***
  Basic Quantization Table for component 3
- ***HAL_LockTypeDef JPEG_HandleTypeDef::Lock***
  JPEG locking object
- ***__IO HAL_JPEG_STATETypeDef JPEG_HandleTypeDef::State***
  JPEG peripheral state
- ***__IO uint32_t JPEG_HandleTypeDef::ErrorCode***
  JPEG Error code
- ***__IO uint32_t JPEG_HandleTypeDef::Context***
  JPEG Internal context

41.2 JPEG Firmware driver API description

The following section lists the various functions of the JPEG library.

41.2.1 How to use this driver

1. Initialize the JPEG peripheral using HAL_JPEG_Init : No initialization parameters are required. Only the call to HAL_JPEG_Init is necessary to initialize the JPEG peripheral.
2. If operation is JPEG encoding use function HAL_JPEG_ConfigEncoding to set the encoding parameters (mandatory before calling the encoding function). the application can change the encoding parameter ImageQuality from 1 to 100 to obtain a more or less quality (visual quality vs the original row image), and inversely more or less jpg file size.
3. Note that for decoding operation the JPEG peripheral output data are organized in YCbCr blocks called MCU (Minimum Coded Unit) as defioned in the JPEG specification ISO/IEC 10918-1 standard. It is up to the application to transform these YCbCr blocks to RGB data that can be display. Respectively, for Encoding operation the JPEG peripheral input should be organized in YCbCr MCU blocks. It is up to the application to perform the necessary RGB to YCbCr MCU blocks transformation before feeding the JPEG peripheral with data.

4. Use functions HAL_JPEG_Encode and HAL_JPEG_Decode to start respectively a JPEG encoding/decoding operation in polling method (blocking).
5. Use functions HAL_JPEG_Encode_IT and HAL_JPEG_Decode_IT to start respectively a JPEG encoding/ decoding operation with Interrupt method (not blocking).
6. Use functions HAL_JPEG_Encode_DMA and HAL_JPEG_Decode_DMA to start respectively a JPEG encoding/decoding operation with DMA method (not blocking).
7. Callback HAL_JPEG_InfoReadyCallback is asserted if the current operation is a JPEG decoding to provide the application with JPEG image parameters. This callback is asserted when the JPEG peripheral successfully parse the JPEG header.
8. Callback HAL_JPEG_GetDataCallback is asserted for both encoding and decoding operations to inform the application that the input buffer has been consumed by the peripheral and to ask for a new data chunk if the operation (encoding/decoding) has not been complete yet.
   - This CallBack should be implemented in the application side. It should call the function HAL_JPEG_ConfigInputBuffer if new input data are available, or call HAL_JPEG_Pause with parameter XferSelection set to JPEG_PAUSE_RESUME_INPUT to inform the JPEG HAL driver that the ongoing operation shall pause waiting for the application to provide a new input data chunk. Once the application succeed getting new data and if the input has been paused, the application can call the function HAL_JPEG_ConfigInputBuffer to set the new input buffer and size, then resume the JPEG HAL input by calling new function HAL_JPEG_Resume. If the application has ended feeding the HAL JPEG with input data (no more input data), the application Should call the function HAL_JPEG_ConfigInputBuffer (within the callback HAL_JPEG_GetDataCallback) with the parameter InDataLength set to zero.
   - The mechanism of HAL_JPEG_ConfigInputBuffer/HAL_JPEG_Pause/HAL_JPEG_Resume allows to the application to provide the input data (for encoding or decoding) by chunks. If the new input data chunk is not available (because data should be read from an input file for example) the application can pause the JPEG input (using function HAL_JPEG_Pause) Once the new input data chunk is available (read from a file for example), the application can call the function HAL_JPEG_ConfigInputBuffer to provide the HAL with the new chunk then resume the JPEG HAL input by calling function HAL_JPEG_Resume.
   - The application can call functions HAL_JPEG_ConfigInputBuffer then HAL_JPEG_Resume. any time (outside the HAL_JPEG_GetDataCallback) Once the new input chunk data available. However, to keep data coherency, the function HAL_JPEG_Pause must be imperatively called (if necessary) within the callback HAL_JPEG_GetDataCallback, i.e when the HAL JPEG has ended Transferring the previous chunk buffer to the JPEG peripheral.
9. Callback HAL_JPEG_DataReadyCallback is asserted when the HAL JPEG driver has filled the given output buffer with the given size.
   - This CallBack should be implemented in the application side. It should call the function HAL_JPEG_ConfigOutputBuffer to provide the HAL JPEG driver with the new output buffer location and size to be used to store next data chunk. if the application is not ready to provide the output chunk location then it can call the function HAL_JPEG_Pause with parameter XferSelection set to JPEG_PAUSE_RESUME_OUTPUT to inform the JPEG HAL driver that it shall pause output data. Once the application is ready to receive the new data chunk (output buffer location free or available) it should call the function HAL_JPEG_ConfigOutputBuffer to provide the HAL JPEG driver with the new output chunk buffer location and size, then call HAL_JPEG_Resume to inform the HAL that it shall resume outputting data in the given output buffer.
   - The mechanism of HAL_JPEG_ConfigOutputBuffer/HAL_JPEG_Pause/HAL_JPEG_Resume allows the application to receive data from the JPEG peripheral by chunks. when a chunk is received, the application can pause the HAL JPEG output data to be able to process these received data (YCbCr to RGB conversion in case of decoding or data storage in case of encoding).
   - The application can call functions HAL_JPEG_ ConfigOutputBuffer then HAL_JPEG_Resume. any time (outside the HAL_JPEG_DataReadyCallback) Once the output data buffer is free to use. However, to keep data coherency, the function HAL_JPEG_Pause must be imperatively called (if necessary) within the callback HAL_JPEG_ DataReadyCallback, i.e when the HAL JPEG has ended Transferring the previous chunk buffer from the JPEG peripheral to the application.
10. Callback HAL_JPEG_EncodeCpltCallback is asserted when the HAL JPEG driver has ended the current JPEG encoding operation, and all output data has been transmitted to the application.
11. Callback HAL_JPEG_DecodeCpltCallback is asserted when the HAL JPEG driver has ended the current JPEG decoding operation. and all output data has been transmitted to the application.
12. Callback HAL_JPEG_ErrorCallback is asserted when an error occurred during the current operation. the application can call the function HAL_JPEG_GetError() to retrieve the error codes.

13. By default the HAL JPEG driver uses the default quantization tables as provide in the JPEG specification (ISO/IEC 10918-1 standard) for encoding. User can change these default tables if necessary using the function HAL_JPEG_SetUserQuantTables Note that for decoding the quantization tables are automatically extracted from the JPEG header.
14. To control JPEG state you can use the following function: HAL_JPEG_GetState()

JPEG HAL driver macros list

Below the list of most used macros in JPEG HAL driver.

- __HAL_JPEG_RESET_HANDLE_STATE : Reset JPEG handle state.
- __HAL_JPEG_ENABLE : Enable the JPEG peripheral.
- __HAL_JPEG_DISABLE : Disable the JPEG peripheral.
- __HAL_JPEG_GET_FLAG : Check the specified JPEG status flag.
- __HAL_JPEG_CLEAR_FLAG : Clear the specified JPEG status flag.
- __HAL_JPEG_ENABLE_IT : Enable the specified JPEG Interrupt.
- __HAL_JPEG_DISABLE_IT : Disable the specified JPEG Interrupt.
- __HAL_JPEG_GET_IT_SOURCE : returns the state of the specified JPEG Interrupt (Enabled or disabled).

Callback registration

41.2.2 Detailed description of functions

HAL_JPEG_Init

Function name

HAL_StatusTypeDef HAL_JPEG_Init (JPEG_HandleTypeDef * hjpeg)

Function description

HAL_JPEG_DeInit

Function name

HAL_StatusTypeDef HAL_JPEG_DeInit (JPEG_HandleTypeDef * hjpeg)

Function description

HAL_JPEG_MspInit

Function name

void HAL_JPEG_MspInit (JPEG_HandleTypeDef * hjpeg)

Function description

HAL_JPEG_MspDeInit

Function name

void HAL_JPEG_MspDeInit (JPEG_HandleTypeDef * hjpeg)

Function description

HAL_JPEG_ConfigEncoding

Function name

HAL_StatusTypeDef HAL_JPEG_ConfigEncoding (JPEG_HandleTypeDef * hjpeg, const JPEG_ConfTypeDef * pConf)

Function description

HAL_JPEG_GetInfo

Function name

HAL_StatusTypeDef HAL_JPEG_GetInfo (JPEG_HandleTypeDef * hjpeg, JPEG_ConfTypeDef * pInfo)

Function description

HAL_JPEG_EnableHeaderParsing

Function name

HAL_StatusTypeDef HAL_JPEG_EnableHeaderParsing (JPEG_HandleTypeDef * hjpeg)

Function description

HAL_JPEG_DisableHeaderParsing

Function name

HAL_StatusTypeDef HAL_JPEG_DisableHeaderParsing (JPEG_HandleTypeDef * hjpeg)

Function description

HAL_JPEG_SetUserQuantTables

Function name

HAL_StatusTypeDef HAL_JPEG_SetUserQuantTables (JPEG_HandleTypeDef * hjpeg, uint8_t * QTable0, uint8_t * QTable1, uint8_t * QTable2, uint8_t * QTable3)

Function description

HAL_JPEG_Encode

Function name

HAL_StatusTypeDef HAL_JPEG_Encode (JPEG_HandleTypeDef * hjpeg, uint8_t * pDataInMCU, uint32_t InDataLength, uint8_t * pDataOut, uint32_t OutDataLength, uint32_t Timeout)

Function description

HAL_JPEG_Decode

Function name

HAL_StatusTypeDef HAL_JPEG_Decode (JPEG_HandleTypeDef * hjpeg, uint8_t * pDataIn, uint32_t InDataLength, uint8_t * pDataOutMCU, uint32_t OutDataLength, uint32_t Timeout)

Function description

HAL_JPEG_Encode_IT

Function name

HAL_StatusTypeDef HAL_JPEG_Encode_IT (JPEG_HandleTypeDef * hjpeg, uint8_t * pDataInMCU, uint32_t InDataLength, uint8_t * pDataOut, uint32_t OutDataLength)

Function description

HAL_JPEG_Decode_IT

Function name

HAL_StatusTypeDef HAL_JPEG_Decode_IT (JPEG_HandleTypeDef * hjpeg, uint8_t * pDataIn, uint32_t InDataLength, uint8_t * pDataOutMCU, uint32_t OutDataLength)

Function description

HAL_JPEG_Encode_DMA

Function name

HAL_StatusTypeDef HAL_JPEG_Encode_DMA (JPEG_HandleTypeDef * hjpeg, uint8_t * pDataInMCU, uint32_t InDataLength, uint8_t * pDataOut, uint32_t OutDataLength)

Function description

HAL_JPEG_Decode_DMA

Function name

HAL_StatusTypeDef HAL_JPEG_Decode_DMA (JPEG_HandleTypeDef * hjpeg, uint8_t * pDataIn, uint32_t InDataLength, uint8_t * pDataOutMCU, uint32_t OutDataLength)

Function description

HAL_JPEG_Pause

Function name

HAL_StatusTypeDef HAL_JPEG_Pause (JPEG_HandleTypeDef * hjpeg, uint32_t XferSelection)

Function description

HAL_JPEG_Resume

Function name

HAL_StatusTypeDef HAL_JPEG_Resume (JPEG_HandleTypeDef * hjpeg, uint32_t XferSelection)

Function description

HAL_JPEG_ConfigInputBuffer

Function name

void HAL_JPEG_ConfigInputBuffer (JPEG_HandleTypeDef * hjpeg, uint8_t * pNewInputBuffer, uint32_t InDataLength)

Function description

HAL_JPEG_ConfigOutputBuffer

Function name

void HAL_JPEG_ConfigOutputBuffer (JPEG_HandleTypeDef * hjpeg, uint8_t * pNewOutputBuffer, uint32_t OutDataLength)

Function description

HAL_JPEG_Abort

Function name

HAL_StatusTypeDef HAL_JPEG_Abort (JPEG_HandleTypeDef * hjpeg)

Function description

HAL_JPEG_InfoReadyCallback

Function name

void HAL_JPEG_InfoReadyCallback (JPEG_HandleTypeDef * hjpeg, JPEG_ConfTypeDef * pInfo)

Function description

HAL_JPEG_EncodeCpltCallback

Function name

void HAL_JPEG_EncodeCpltCallback (JPEG_HandleTypeDef * hjpeg)

Function description

HAL_JPEG_DecodeCpltCallback

Function name

void HAL_JPEG_DecodeCpltCallback (JPEG_HandleTypeDef * hjpeg)

Function description

HAL_JPEG_ErrorCallback

Function name

void HAL_JPEG_ErrorCallback (JPEG_HandleTypeDef * hjpeg)

Function description

HAL_JPEG_GetDataCallback

Function name

void HAL_JPEG_GetDataCallback (JPEG_HandleTypeDef * hjpeg, uint32_t NbDecodedData)

Function description

HAL_JPEG_DataReadyCallback

Function name

void HAL_JPEG_DataReadyCallback (JPEG_HandleTypeDef * hjpeg, uint8_t * pDataOut, uint32_t OutDataLength)

Function description

HAL_JPEG_IRQHandler

Function name

void HAL_JPEG_IRQHandler (JPEG_HandleTypeDef * hjpeg)

Function description

HAL_JPEG_GetState

Function name

HAL_JPEG_STATETypeDef HAL_JPEG_GetState (const JPEG_HandleTypeDef * hjpeg)

Function description

HAL_JPEG_GetError

Function name

uint32_t HAL_JPEG_GetError (const JPEG_HandleTypeDef * hjpeg)

Function description

41.3 JPEG Firmware driver defines

The following section lists the various define and macros of the module.

41.3.1 JPEG

JPEG

JPEG Chrominance Sampling

JPEG_444_SUBSAMPLING

Chroma Subsampling 4:4:4

JPEG_420_SUBSAMPLING

Chroma Subsampling 4:2:0

JPEG_422_SUBSAMPLING

Chroma Subsampling 4:2:2

JPEG ColorSpace

JPEG_GRAYSCALE_COLORSPACE

JPEG_YCBCR_COLORSPACE

JPEG_CMYK_COLORSPACE

JPEG Error Code definition

HAL_JPEG_ERROR_NONE

No error

HAL_JPEG_ERROR_HUFF_TABLE

HUffman Table programming error

HAL_JPEG_ERROR_QUANT_TABLE

Quantization Table programming error

HAL_JPEG_ERROR_DMA

DMA transfer error

HAL_JPEG_ERROR_TIMEOUT

Timeout error

JPEG Exported Macros

__HAL_JPEG_RESET_HANDLE_STATE

Description:

- Reset JPEG handle state.

Parameters:

- __HANDLE__: specifies the JPEG handle.

Return value:

- None

__HAL_JPEG_ENABLE

Description:

- Enable the JPEG peripheral.

Parameters:

- __HANDLE__: specifies the JPEG handle.

Return value:

- None

__HAL_JPEG_DISABLE

Description:

- Disable the JPEG peripheral.

Parameters:

- __HANDLE__: specifies the JPEG handle.

Return value:

- None

__HAL_JPEG_GET_FLAG

Description:

- Check the specified JPEG status flag.

Parameters:

- __HANDLE__: specifies the JPEG handle.
- __FLAG__: specifies the flag to check This parameter can be one of the following values:
  - JPEG_FLAG_IFTF : The input FIFO is not full and is below its threshold flag
  - JPEG_FLAG_IFNFF : The input FIFO Not Full Flag, a data can be written
  - JPEG_FLAG_OFTF : The output FIFO is not empty and has reach its threshold
  - JPEG_FLAG_OFNEF : The output FIFO is not empty, a data is available
  - JPEG_FLAG_EOCF : JPEG Codec core has finished the encoding or the decoding process and than last data has been sent to the output FIFO
  - JPEG_FLAG_HPDF : JPEG Codec has finished the parsing of the headers and the internal registers have been updated
  - JPEG_FLAG_COF : JPEG Codec operation on going flag

Return value:

- __HAL_JPEG_GET_FLAG: : returns The new state of __FLAG__ (TRUE or FALSE)

__HAL_JPEG_CLEAR_FLAG

Description:

- Clear the specified JPEG status flag.

Parameters:

- __HANDLE__: specifies the JPEG handle.
- __FLAG__: specifies the flag to clear This parameter can be one of the following values:
  - JPEG_FLAG_EOCF : JPEG Codec core has finished the encoding or the decoding process and than last data has been sent to the output FIFO
  - JPEG_FLAG_HPDF : JPEG Codec has finished the parsing of the headers

Return value:

- None

__HAL_JPEG_ENABLE_IT

Description:

- Enable Interrupt.

Parameters:

- __HANDLE__: specifies the JPEG handle.
- __INTERRUPT__: specifies the interrupt to enable This parameter can be one of the following values:
  - JPEG_IT_IFT : Input FIFO Threshold Interrupt
  - JPEG_IT_IFNF : Input FIFO Not Full Interrupt
  - JPEG_IT_OFT : Output FIFO Threshold Interrupt
  - JPEG_IT_OFNE : Output FIFO Not empty Interrupt
  - JPEG_IT_EOC : End of Conversion Interrupt
  - JPEG_IT_HPD : Header Parsing Done Interrupt

Return value:

- No: return

__HAL_JPEG_DISABLE_IT

Description:

- Disable Interrupt.

Parameters:

- __HANDLE__: specifies the JPEG handle.
- __INTERRUPT__: specifies the interrupt to disable This parameter can be one of the following values:
  - JPEG_IT_IFT : Input FIFO Threshold Interrupt
  - JPEG_IT_IFNF : Input FIFO Not Full Interrupt
  - JPEG_IT_OFT : Output FIFO Threshold Interrupt
  - JPEG_IT_OFNE : Output FIFO Not empty Interrupt
  - JPEG_IT_EOC : End of Conversion Interrupt
  - JPEG_IT_HPD : Header Parsing Done Interrupt

Return value:

- No: return

Notes:

- To disable an IT we must use MODIFY_REG macro to avoid writing "1" to the FIFO flush bits located in the same IT enable register (CR register).

__HAL_JPEG_GET_IT_SOURCE

Description:

- Get Interrupt state.

Parameters:

- __HANDLE__: specifies the JPEG handle.
- __INTERRUPT__: specifies the interrupt to check This parameter can be one of the following values:
  - JPEG_IT_IFT : Input FIFO Threshold Interrupt
  - JPEG_IT_IFNF : Input FIFO Not Full Interrupt
  - JPEG_IT_OFT : Output FIFO Threshold Interrupt
  - JPEG_IT_OFNE : Output FIFO Not empty Interrupt
  - JPEG_IT_EOC : End of Conversion Interrupt
  - JPEG_IT_HPD : Header Parsing Done Interrupt

Return value:

- returns: The new state of __INTERRUPT__ (Enabled or disabled)

JPEG Flag definition

JPEG_FLAG_IFTF

Input FIFO is not full and is below its threshold flag

JPEG_FLAG_IFNFF

Input FIFO Not Full Flag, a data can be written

JPEG_FLAG_OFTF

Output FIFO is not empty and has reach its threshold

JPEG_FLAG_OFNEF

Output FIFO is not empty, a data is available

JPEG_FLAG_EOCF

JPEG Codec core has finished the encoding or the decoding process and than last data has been sent to the output FIFO

JPEG_FLAG_HPDF

JPEG Codec has finished the parsing of the headers and the internal registers have been updated

JPEG_FLAG_COF

JPEG Codec operation on going flag

JPEG_FLAG_ALL

JPEG Codec All previous flag

JPEG Image Quality

JPEG_IMAGE_QUALITY_MIN

Minimum JPEG quality

JPEG_IMAGE_QUALITY_MAX

Maximum JPEG quality

JPEG Interrupt configuration definition

JPEG_IT_IFT

Input FIFO Threshold Interrupt

JPEG_IT_IFNF

Input FIFO Not Full Interrupt

JPEG_IT_OFT

Output FIFO Threshold Interrupt

JPEG_IT_OFNE

Output FIFO Not Empty Interrupt

JPEG_IT_EOC

End of Conversion Interrupt

JPEG_IT_HPD

Header Parsing Done Interrupt

JPEG Process Pause Resume definition

JPEG_PAUSE_RESUME_INPUT

Pause/Resume Input FIFO Xfer

JPEG_PAUSE_RESUME_OUTPUT

Pause/Resume Output FIFO Xfer

JPEG_PAUSE_RESUME_INPUT_OUTPUT

Pause/Resume Input and Output FIFO Xfer

JPEG Quantization Table Size

JPEG_QUANT_TABLE_SIZE

JPEG Quantization Table Size in bytes

42 HAL LPTIM Generic Driver

42.1 LPTIM Firmware driver registers structures

42.1.1 LPTIM_ClockConfigTypeDef

LPTIM_ClockConfigTypeDef is defined in the stm32n6xx_hal_lptim.h

Data Fields

- ***uint32_t Source***
- ***uint32_t Prescaler***

Field Documentation

- ***uint32_t LPTIM_ClockConfigTypeDef::Source***
  Selects the clock source. This parameter can be a value of ***LPTIM_Clock_Source***
- ***uint32_t LPTIM_ClockConfigTypeDef::Prescaler***
  Specifies the counter clock Prescaler. This parameter can be a value of ***LPTIM_Clock_Prescaler***

42.1.2 LPTIM_ULPClockConfigTypeDef

LPTIM_ULPClockConfigTypeDef is defined in the stm32n6xx_hal_lptim.h

Data Fields

- ***uint32_t Polarity***
- ***uint32_t SampleTime***

Field Documentation

- ***uint32_t LPTIM_ULPClockConfigTypeDef::Polarity***
  Selects the polarity of the active edge for the counter unit if the ULPTIM input is selected. Note: This parameter is used only when Ultra low power clock source is used. Note: If the polarity is configured on 'both edges', an auxiliary clock (one of the Low power oscillator) must be active. This parameter can be a value of ***LPTIM_Clock_Polarity***
- ***uint32_t LPTIM_ULPClockConfigTypeDef::SampleTime***
  Selects the clock sampling time to configure the clock glitch filter. Note: This parameter is used only when Ultra low power clock source is used. This parameter can be a value of ***LPTIM_Clock_Sample_Time***

42.1.3 LPTIM_TriggerConfigTypeDef

LPTIM_TriggerConfigTypeDef is defined in the stm32n6xx_hal_lptim.h

Data Fields

- ***uint32_t Source***
- ***uint32_t ActiveEdge***
- ***uint32_t SampleTime***

Field Documentation

- ***uint32_t LPTIM_TriggerConfigTypeDef::Source***
  Selects the Trigger source. This parameter can be a value of ***LPTIM_Trigger_Source***
- ***uint32_t LPTIM_TriggerConfigTypeDef::ActiveEdge***
  Selects the Trigger active edge. Note: This parameter is used only when an external trigger is used. This parameter can be a value of ***LPTIM_External_Trigger_Polarity***
- ***uint32_t LPTIM_TriggerConfigTypeDef::SampleTime***
  Selects the trigger sampling time to configure the clock glitch filter. Note: This parameter is used only when an external trigger is used. This parameter can be a value of ***LPTIM_Trigger_Sample_Time***

42.1.4 LPTIM_InitTypeDef

LPTIM_InitTypeDef is defined in the stm32n6xx_hal_lptim.h

Data Fields

- ***LPTIM_ClockConfigTypeDef Clock***
- ***LPTIM_ULPClockConfigTypeDef UltraLowPowerClock***

- ***LPTIM_TriggerConfigTypeDef Trigger***
- ***uint32_t Period***
- ***uint32_t UpdateMode***
- ***uint32_t CounterSource***
- ***uint32_t Input1Source***
- ***uint32_t Input2Source***
- ***uint32_t RepetitionCounter***

Field Documentation

- ***LPTIM_ClockConfigTypeDef LPTIM_InitTypeDef::Clock***
  Specifies the clock parameters
- ***LPTIM_ULPClockConfigTypeDef LPTIM_InitTypeDef::UltraLowPowerClock***
  Specifies the Ultra Low Power clock parameters
- ***LPTIM_TriggerConfigTypeDef LPTIM_InitTypeDef::Trigger***
  Specifies the Trigger parameters
- ***uint32_t LPTIM_InitTypeDef::Period***
  Specifies the period value to be loaded into the active Auto-Reload Register at the next update event. This parameter can be a number between Min_Data = 0x0001 and Max_Data = 0xFFFF.
- ***uint32_t LPTIM_InitTypeDef::UpdateMode***
  Specifies whether the update of the autoreload and the compare values is done immediately or after the end of current period. This parameter can be a value of ***LPTIM_Updating_Mode***
- ***uint32_t LPTIM_InitTypeDef::CounterSource***
  Specifies whether the counter is incremented each internal event or each external event. This parameter can be a value of ***LPTIM_Counter_Source***
- ***uint32_t LPTIM_InitTypeDef::Input1Source***
  Specifies source selected for input1 (GPIO or comparator output). This parameter can be a value of ***LPTIM_Input1_Source***
- ***uint32_t LPTIM_InitTypeDef::Input2Source***
  Specifies source selected for input2 (GPIO or comparator output). Note: This parameter is used only for encoder feature so is used only for LPTIM1 instance. This parameter can be a value of ***LPTIM_Input2_Source***
- ***uint32_t LPTIM_InitTypeDef::RepetitionCounter***
  Specifies the repetition counter value. Each time the RCR downcounter reaches zero, an update event is generated and counting restarts from the RCR value (N). Note: When using repetition counter the UpdateMode field must be set to LPTIM_UPDATE_ENDOFPERIOD otherwise unpredictable behavior may occur. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF.

42.1.5 LPTIM_OC_ConfigTypeDef

LPTIM_OC_ConfigTypeDef is defined in the stm32n6xx_hal_lptim.h

Data Fields

- ***uint32_t Pulse***
- ***uint32_t OCPolarity***

Field Documentation

- ***uint32_t LPTIM_OC_ConfigTypeDef::Pulse***
  Specifies the pulse value to be loaded into the Capture Compare Register. This parameter can be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF
- ***uint32_t LPTIM_OC_ConfigTypeDef::OCPolarity***
  Specifies the output polarity. This parameter can be a value of ***LPTIM_Output_Compare_Polarity***

42.1.6 LPTIM_IC_ConfigTypeDef

LPTIM_IC_ConfigTypeDef is defined in the stm32n6xx_hal_lptim.h

Data Fields

- ***uint32_t ICInputSource***
- ***uint32_t ICPrescaler***
- ***uint32_t ICPolarity***

- ***uint32_t ICFilter***

Field Documentation

- ***uint32_t LPTIM_IC_ConfigTypeDef::ICInputSource***
  Specifies source selected for IC channel. This parameter can be a value of ***LPTIM_Input_Capture_Source***
- ***uint32_t LPTIM_IC_ConfigTypeDef::ICPrescaler***
  Specifies the Input Capture Prescaler. This parameter can be a value of ***LPTIM_Input_Capture_Prescaler***
- ***uint32_t LPTIM_IC_ConfigTypeDef::ICPolarity***
  Specifies the active edge of the input signal. This parameter can be a value of ***LPTIM_Input_Capture_Polarity***
- ***uint32_t LPTIM_IC_ConfigTypeDef::ICFilter***
  Specifies the input capture filter. This parameter can be a value of ***LPTIM_Input_Capture_Filter***

42.1.7 __LPTIM_HandleTypeDef

__LPTIM_HandleTypeDef is defined in the stm32n6xx_hal_lptim.h

Data Fields

- ***LPTIM_TypeDef * Instance***
- ***LPTIM_InitTypeDef Init***
- ***HAL_LPTIM_ActiveChannel Channel***
- ***DMA_HandleTypeDef * hdma***
- ***HAL_StatusTypeDef Status***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_LPTIM_StateTypeDef State***
- ***__IO HAL_LPTIM_ChannelStateTypeDef ChannelState***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***
- ***void(* CompareMatchCallback***
- ***void(* AutoReloadMatchCallback***
- ***void(* TriggerCallback***
- ***void(* CompareWriteCallback***
- ***void(* AutoReloadWriteCallback***
- ***void(* DirectionUpCallback***
- ***void(* DirectionDownCallback***
- ***void(* UpdateEventCallback***
- ***void(* RepCounterWriteCallback***
- ***void(* UpdateEventHalfCpltCallback***
- ***void(* ErrorCallback***
- ***void(* IC_CaptureCallback***
- ***void(* IC_CaptureHalfCpltCallback***
- ***void(* IC_OverCaptureCallback***

Field Documentation

- ***LPTIM_TypeDef* __LPTIM_HandleTypeDef::Instance***
  Register base address
- ***LPTIM_InitTypeDef __LPTIM_HandleTypeDef::Init***
  LPTIM required parameters
- ***HAL_LPTIM_ActiveChannel __LPTIM_HandleTypeDef::Channel***
  Active channel
- ***DMA_HandleTypeDef* __LPTIM_HandleTypeDef::hdma[3]***
  DMA Handlers array, This array is accessed by a ***LPTIM_DMA_Handle_index***
- ***HAL_StatusTypeDef __LPTIM_HandleTypeDef::Status***
  LPTIM peripheral status

- ***HAL_LockTypeDef __LPTIM_HandleTypeDef::Lock***
  LPTIM locking object
- ***__IO HAL_LPTIM_StateTypeDef __LPTIM_HandleTypeDef::State***
  LPTIM peripheral state
- ***__IO HAL_LPTIM_ChannelStateTypeDef __LPTIM_HandleTypeDef::ChannelState[2]***
  LPTIM channel operation state
- ***void(* __LPTIM_HandleTypeDef::MspInitCallback)(struct __LPTIM_HandleTypeDef *hlptim)***
  LPTIM Base Msp Init Callback
- ***void(* __LPTIM_HandleTypeDef::MspDeInitCallback)(struct __LPTIM_HandleTypeDef *hlptim)***
  LPTIM Base Msp DeInit Callback
- ***void(* __LPTIM_HandleTypeDef::CompareMatchCallback)(struct __LPTIM_HandleTypeDef *hlptim)***
  Compare match Callback
- ***void(* __LPTIM_HandleTypeDef::AutoReloadMatchCallback)(struct __LPTIM_HandleTypeDef *hlptim)***
  Auto-reload match Callback
- ***void(* __LPTIM_HandleTypeDef::TriggerCallback)(struct __LPTIM_HandleTypeDef *hlptim)***
  External trigger event detection Callback
- ***void(* __LPTIM_HandleTypeDef::CompareWriteCallback)(struct __LPTIM_HandleTypeDef *hlptim)***
  Compare register write complete Callback
- ***void(* __LPTIM_HandleTypeDef::AutoReloadWriteCallback)(struct __LPTIM_HandleTypeDef *hlptim)***
  Auto-reload register write complete Callback
- ***void(* __LPTIM_HandleTypeDef::DirectionUpCallback)(struct __LPTIM_HandleTypeDef *hlptim)***
  Up-counting direction change Callback
- ***void(* __LPTIM_HandleTypeDef::DirectionDownCallback)(struct __LPTIM_HandleTypeDef *hlptim)***
  Down-counting direction change Callback
- ***void(* __LPTIM_HandleTypeDef::UpdateEventCallback)(struct __LPTIM_HandleTypeDef *hlptim)***
  Update event detection Callback
- ***void(* __LPTIM_HandleTypeDef::RepCounterWriteCallback)(struct __LPTIM_HandleTypeDef *hlptim)***
  Repetition counter register write complete Callback
- ***void(* __LPTIM_HandleTypeDef::UpdateEventHalfCpltCallback)(struct __LPTIM_HandleTypeDef *hlptim)***
  Update event half complete detection Callback
- ***void(* __LPTIM_HandleTypeDef::ErrorCallback)(struct __LPTIM_HandleTypeDef *hlptim)***
  LPTIM Error Callback
- ***void(* __LPTIM_HandleTypeDef::IC_CaptureCallback)(struct __LPTIM_HandleTypeDef *hlptim)***
  Input capture Callback
- ***void(* __LPTIM_HandleTypeDef::IC_CaptureHalfCpltCallback)(struct __LPTIM_HandleTypeDef *htim)***
  Input Capture half complete Callback
- ***void(* __LPTIM_HandleTypeDef::IC_OverCaptureCallback)(struct __LPTIM_HandleTypeDef *hlptim)***
  Over capture Callback

42.2 LPTIM Firmware driver API description

The following section lists the various functions of the LPTIM library.

42.2.1 How to use this driver

The LPTIM HAL driver can be used as follows:

1. Initialize the LPTIM low level resources by implementing the HAL_LPTIM_MspInit():
   - Enable the LPTIM interface clock using __HAL_RCC_LPTIMx_CLK_ENABLE().
   - In case of using interrupts (e.g. HAL_LPTIM_PWM_Start_IT()):
     - Configure the LPTIM interrupt priority using HAL_NVIC_SetPriority().
     - Enable the LPTIM IRQ handler using HAL_NVIC_EnableIRQ().
     - In LPTIM IRQ handler, call HAL_LPTIM_IRQHandler().
2. Initialize the LPTIM HAL using HAL_LPTIM_Init(). This function configures mainly:
   - The instance: LPTIM1, LPTIM2, LPTIM3, LPTIM4 or LPTIM5.
   - Clock: the counter clock.
     - Source : it can be either the ULPTIM input (IN1) or one of the internal clock; (APB, LSE, LSI or MSI).
     - Prescaler: select the clock divider.
   - UltraLowPowerClock : To be used only if the ULPTIM is selected as counter clock source.
     - Polarity: polarity of the active edge for the counter unit if the ULPTIM input is selected.
     - SampleTime: clock sampling time to configure the clock glitch filter.
   - Trigger: How the counter start.
     - Source: trigger can be software or one of the hardware triggers.
     - ActiveEdge : only for hardware trigger.
     - SampleTime : trigger sampling time to configure the trigger glitch filter.
   - OutputPolarity : 2 opposite polarities are possible.
   - UpdateMode: specifies whether the update of the autoreload and the compare values is done immediately or after the end of current period.
   - Input1Source: Source selected for input1 (GPIO or comparator output).
   - Input2Source: Source selected for input2 (GPIO or comparator output). Input2 is used only for encoder feature so is used only for LPTIM1 instance.
3. Six modes are available:
   - PWM Mode: To generate a PWM signal with specified period and pulse, call HAL_LPTIM_PWM_Start() or HAL_LPTIM_PWM_Start_IT() for interruption mode.
   - One Pulse Mode: To generate pulse with specified width in response to a stimulus, call HAL_LPTIM_OnePulse_Start() or HAL_LPTIM_OnePulse_Start_IT() for interruption mode.
   - Set once Mode: In this mode, the output changes the level (from low level to high level if the output polarity is configured high, else the opposite) when a compare match occurs. To start this mode, call HAL_LPTIM_SetOnce_Start() or HAL_LPTIM_SetOnce_Start_IT() for interruption mode.
   - Encoder Mode: To use the encoder interface call HAL_LPTIM_Encoder_Start() or HAL_LPTIM_Encoder_Start_IT() for interruption mode. Only available for LPTIM1 instance.
   - Time out Mode: an active edge on one selected trigger input rests the counter. The first trigger event will start the timer, any successive trigger event will reset the counter and the timer will restart. To start this mode call HAL_LPTIM_TimeOut_Start_IT() or HAL_LPTIM_TimeOut_Start_IT() for interruption mode.
   - Counter Mode: counter can be used to count external events on the LPTIM Input1 or it can be used to count internal clock cycles. To start this mode, call HAL_LPTIM_Counter_Start() or HAL_LPTIM_Counter_Start_IT() for interruption mode.
4. User can stop any process by calling the corresponding API: HAL_LPTIM_Xxx_Stop() or HAL_LPTIM_Xxx_Stop_IT() if the process is already started in interruption mode.
5. De-initialize the LPTIM peripheral using HAL_LPTIM_DeInit().

Callback registration

The compilation define USE_HAL_LPTIM_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks.

Use Function HAL_LPTIM_RegisterCallback() to register a callback. HAL_LPTIM_RegisterCallback() takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

Use function HAL_LPTIM_UnRegisterCallback() to reset a callback to the default weak function. HAL_LPTIM_UnRegisterCallback takes as parameters the HAL peripheral handle, and the Callback ID.

These functions allow to register/unregister following callbacks:

- MspInitCallback : LPTIM Base Msp Init Callback.
- MspDeInitCallback : LPTIM Base Msp DeInit Callback.
- CompareMatchCallback : Compare match Callback.
- AutoReloadMatchCallback : Auto-reload match Callback.
- TriggerCallback : External trigger event detection Callback.
- CompareWriteCallback : Compare register write complete Callback.
- AutoReloadWriteCallback : Auto-reload register write complete Callback.
- DirectionUpCallback : Up-counting direction change Callback.
- DirectionDownCallback : Down-counting direction change Callback.
- UpdateEventCallback : Update event detection Callback.
- RepCounterWriteCallback : Repetition counter register write complete Callback.

By default, after the Init and when the state is HAL_LPTIM_STATE_RESET all interrupt callbacks are set to the corresponding weak functions: examples HAL_LPTIM_TriggerCallback(), HAL_LPTIM_CompareMatchCallback().

Exception done for MspInit and MspDeInit functions that are reset to the legacy weak functionalities in the Init/DeInit only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the Init/DeInit keep and use the user MspInit/MspDeInit callbacks (registered beforehand)

Callbacks can be registered/unregistered in HAL_LPTIM_STATE_READY state only. Exception done MspInit/MspDeInit that can be registered/unregistered in HAL_LPTIM_STATE_READY or HAL_LPTIM_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_LPTIM_RegisterCallback() before calling DeInit or Init function.

When The compilation define USE_HAL_LPTIM_REGISTER_CALLBACKS is set to 0 or not defined, the callback registration feature is not available and all callbacks are set to the corresponding weak functions.

42.2.2 Detailed description of functions

HAL_LPTIM_Init

Function name

HAL_StatusTypeDef HAL_LPTIM_Init (LPTIM_HandleTypeDef * hlptim)

Function description

HAL_LPTIM_DeInit

Function name

HAL_StatusTypeDef HAL_LPTIM_DeInit (LPTIM_HandleTypeDef * hlptim)

Function description

HAL_LPTIM_MspInit

Function name

void HAL_LPTIM_MspInit (LPTIM_HandleTypeDef * hlptim)

Function description

HAL_LPTIM_MspDeInit

Function name

void HAL_LPTIM_MspDeInit (LPTIM_HandleTypeDef * hlptim)

Function description

HAL_LPTIM_OC_ConfigChannel

Function name

HAL_StatusTypeDef HAL_LPTIM_OC_ConfigChannel (LPTIM_HandleTypeDef * hlptim, const LPTIM_OC_ConfigTypeDef * sConfig, uint32_t Channel)

Function description

HAL_LPTIM_PWM_Start

Function name

HAL_StatusTypeDef HAL_LPTIM_PWM_Start (LPTIM_HandleTypeDef * hlptim, uint32_t Channel)

Function description

HAL_LPTIM_PWM_Stop

Function name

HAL_StatusTypeDef HAL_LPTIM_PWM_Stop (LPTIM_HandleTypeDef * hlptim, uint32_t Channel)

Function description

HAL_LPTIM_PWM_Start_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_PWM_Start_IT (LPTIM_HandleTypeDef * hlptim, uint32_t Channel)

Function description

HAL_LPTIM_PWM_Stop_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_PWM_Stop_IT (LPTIM_HandleTypeDef * hlptim, uint32_t Channel)

Function description

HAL_LPTIM_PWM_Start_DMA

Function name

HAL_StatusTypeDef HAL_LPTIM_PWM_Start_DMA (LPTIM_HandleTypeDef * hlptim, uint32_t Channel, const uint32_t * pData, uint32_t Length)

Function description

HAL_LPTIM_PWM_Stop_DMA

Function name

HAL_StatusTypeDef HAL_LPTIM_PWM_Stop_DMA (LPTIM_HandleTypeDef * hlptim, uint32_t Channel)

Function description

HAL_LPTIM_OnePulse_Start

Function name

HAL_StatusTypeDef HAL_LPTIM_OnePulse_Start (LPTIM_HandleTypeDef * hlptim, uint32_t Channel)

Function description

HAL_LPTIM_OnePulse_Stop

Function name

HAL_StatusTypeDef HAL_LPTIM_OnePulse_Stop (LPTIM_HandleTypeDef * hlptim, uint32_t Channel)

Function description

HAL_LPTIM_OnePulse_Start_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_OnePulse_Start_IT (LPTIM_HandleTypeDef * hlptim, uint32_t Channel)

Function description

HAL_LPTIM_OnePulse_Stop_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_OnePulse_Stop_IT (LPTIM_HandleTypeDef * hlptim, uint32_t Channel)

Function description

HAL_LPTIM_SetOnce_Start

Function name

HAL_StatusTypeDef HAL_LPTIM_SetOnce_Start (LPTIM_HandleTypeDef * hlptim, uint32_t Channel)

Function description

HAL_LPTIM_SetOnce_Stop

Function name

HAL_StatusTypeDef HAL_LPTIM_SetOnce_Stop (LPTIM_HandleTypeDef * hlptim, uint32_t Channel)

Function description

HAL_LPTIM_SetOnce_Start_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_SetOnce_Start_IT (LPTIM_HandleTypeDef * hlptim, uint32_t Channel)

Function description

HAL_LPTIM_SetOnce_Stop_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_SetOnce_Stop_IT (LPTIM_HandleTypeDef * hlptim, uint32_t Channel)

Function description

HAL_LPTIM_Encoder_Start

Function name

HAL_StatusTypeDef HAL_LPTIM_Encoder_Start (LPTIM_HandleTypeDef * hlptim)

Function description

HAL_LPTIM_Encoder_Stop

Function name

HAL_StatusTypeDef HAL_LPTIM_Encoder_Stop (LPTIM_HandleTypeDef * hlptim)

Function description

HAL_LPTIM_Encoder_Start_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_Encoder_Start_IT (LPTIM_HandleTypeDef * hlptim)

Function description

HAL_LPTIM_Encoder_Stop_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_Encoder_Stop_IT (LPTIM_HandleTypeDef * hlptim)

Function description

HAL_LPTIM_TimeOut_Start

Function name

HAL_StatusTypeDef HAL_LPTIM_TimeOut_Start (LPTIM_HandleTypeDef * hlptim, uint32_t Timeout)

Function description

HAL_LPTIM_TimeOut_Stop

Function name

HAL_StatusTypeDef HAL_LPTIM_TimeOut_Stop (LPTIM_HandleTypeDef * hlptim)

Function description

HAL_LPTIM_TimeOut_Start_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_TimeOut_Start_IT (LPTIM_HandleTypeDef * hlptim, uint32_t Timeout)

Function description

HAL_LPTIM_TimeOut_Stop_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_TimeOut_Stop_IT (LPTIM_HandleTypeDef * hlptim)

Function description

HAL_LPTIM_Counter_Start

Function name

HAL_StatusTypeDef HAL_LPTIM_Counter_Start (LPTIM_HandleTypeDef * hlptim)

Function description

HAL_LPTIM_Counter_Stop

Function name

HAL_StatusTypeDef HAL_LPTIM_Counter_Stop (LPTIM_HandleTypeDef * hlptim)

Function description

HAL_LPTIM_Counter_Start_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_Counter_Start_IT (LPTIM_HandleTypeDef * hlptim)

Function description

HAL_LPTIM_Counter_Stop_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_Counter_Stop_IT (LPTIM_HandleTypeDef * hlptim)

Function description

HAL_LPTIM_IC_ConfigChannel

Function name

HAL_StatusTypeDef HAL_LPTIM_IC_ConfigChannel (LPTIM_HandleTypeDef * hlptim, const LPTIM_IC_ConfigTypeDef * sConfig, uint32_t Channel)

Function description

HAL_LPTIM_IC_Start

Function name

HAL_StatusTypeDef HAL_LPTIM_IC_Start (LPTIM_HandleTypeDef * hlptim, uint32_t Channel)

Function description

HAL_LPTIM_IC_Stop

Function name

HAL_StatusTypeDef HAL_LPTIM_IC_Stop (LPTIM_HandleTypeDef * hlptim, uint32_t Channel)

Function description

HAL_LPTIM_IC_Start_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_IC_Start_IT (LPTIM_HandleTypeDef * hlptim, uint32_t Channel)

Function description

HAL_LPTIM_IC_Stop_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_IC_Stop_IT (LPTIM_HandleTypeDef * hlptim, uint32_t Channel)

Function description

HAL_LPTIM_IC_Start_DMA

Function name

HAL_StatusTypeDef HAL_LPTIM_IC_Start_DMA (LPTIM_HandleTypeDef * hlptim, uint32_t Channel, uint32_t * pData, uint32_t Length)

Function description

HAL_LPTIM_IC_Stop_DMA

Function name

HAL_StatusTypeDef HAL_LPTIM_IC_Stop_DMA (LPTIM_HandleTypeDef * hlptim, uint32_t Channel)

Function description

HAL_LPTIM_ReadCounter

Function name

uint32_t HAL_LPTIM_ReadCounter (const LPTIM_HandleTypeDef * hlptim)

Function description

HAL_LPTIM_ReadAutoReload

Function name

uint32_t HAL_LPTIM_ReadAutoReload (const LPTIM_HandleTypeDef * hlptim)

Function description

HAL_LPTIM_ReadCapturedValue

Function name

uint32_t HAL_LPTIM_ReadCapturedValue (const LPTIM_HandleTypeDef * hlptim, uint32_t Channel)

Function description

HAL_LPTIM_IC_GetOffset

Function name

uint8_t HAL_LPTIM_IC_GetOffset (const LPTIM_HandleTypeDef * hlptim, uint32_t Channel)

Function description

HAL_LPTIM_IRQHandler

Function name

void HAL_LPTIM_IRQHandler (LPTIM_HandleTypeDef * hlptim)

Function description

HAL_LPTIM_CompareMatchCallback

Function name

void HAL_LPTIM_CompareMatchCallback (LPTIM_HandleTypeDef * hlptim)

Function description

HAL_LPTIM_AutoReloadMatchCallback

Function name

void HAL_LPTIM_AutoReloadMatchCallback (LPTIM_HandleTypeDef * hlptim)

Function description

HAL_LPTIM_TriggerCallback

Function name

void HAL_LPTIM_TriggerCallback (LPTIM_HandleTypeDef * hlptim)

Function description

HAL_LPTIM_CompareWriteCallback

Function name

void HAL_LPTIM_CompareWriteCallback (LPTIM_HandleTypeDef * hlptim)

Function description

HAL_LPTIM_AutoReloadWriteCallback

Function name

void HAL_LPTIM_AutoReloadWriteCallback (LPTIM_HandleTypeDef * hlptim)

Function description

HAL_LPTIM_DirectionUpCallback

Function name

void HAL_LPTIM_DirectionUpCallback (LPTIM_HandleTypeDef * hlptim)

Function description

HAL_LPTIM_DirectionDownCallback

Function name

void HAL_LPTIM_DirectionDownCallback (LPTIM_HandleTypeDef * hlptim)

Function description

HAL_LPTIM_ErrorCallback

Function name

void HAL_LPTIM_ErrorCallback (LPTIM_HandleTypeDef * hlptim)

Function description

HAL_LPTIM_UpdateEventCallback

Function name

void HAL_LPTIM_UpdateEventCallback (LPTIM_HandleTypeDef * hlptim)

Function description

HAL_LPTIM_UpdateEventHalfCpltCallback

Function name

void HAL_LPTIM_UpdateEventHalfCpltCallback (LPTIM_HandleTypeDef * hlptim)

Function description

HAL_LPTIM_RepCounterWriteCallback

Function name

void HAL_LPTIM_RepCounterWriteCallback (LPTIM_HandleTypeDef * hlptim)

Function description

HAL_LPTIM_IC_CaptureCallback

Function name

void HAL_LPTIM_IC_CaptureCallback (LPTIM_HandleTypeDef * hlptim)

Function description

HAL_LPTIM_IC_CaptureHalfCpltCallback

Function name

void HAL_LPTIM_IC_CaptureHalfCpltCallback (LPTIM_HandleTypeDef * hlptim)

Function description

HAL_LPTIM_IC_OverCaptureCallback

Function name

void HAL_LPTIM_IC_OverCaptureCallback (LPTIM_HandleTypeDef * hlptim)

Function description

HAL_LPTIM_RegisterCallback

Function name

HAL_StatusTypeDef HAL_LPTIM_RegisterCallback (LPTIM_HandleTypeDef * lphtim, HAL_LPTIM_CallbackIDTypeDef CallbackID, pLPTIM_CallbackTypeDef pCallback)

Function description

HAL_LPTIM_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_LPTIM_UnRegisterCallback (LPTIM_HandleTypeDef * lphtim, HAL_LPTIM_CallbackIDTypeDef CallbackID)

Function description

HAL_LPTIM_GetState

Function name

HAL_LPTIM_StateTypeDef HAL_LPTIM_GetState (const LPTIM_HandleTypeDef * hlptim)

Function description

42.3 LPTIM Firmware driver defines

The following section lists the various define and macros of the module.

42.3.1 LPTIM

LPTIM

LPTIM Channel

LPTIM_CHANNEL_1

Capture/compare channel 1 identifier

LPTIM_CHANNEL_2

Capture/compare channel 2 identifier

LPTIM Clock Polarity

LPTIM_CLOCKPOLARITY_RISING

LPTIM_CLOCKPOLARITY_FALLING

LPTIM_CLOCKPOLARITY_RISING_FALLING

LPTIM Clock Prescaler

LPTIM_PRESCALER_DIV1

LPTIM_PRESCALER_DIV2

LPTIM_PRESCALER_DIV4

LPTIM_PRESCALER_DIV8

LPTIM_PRESCALER_DIV16

LPTIM_PRESCALER_DIV32

LPTIM_PRESCALER_DIV64

LPTIM_PRESCALER_DIV128

LPTIM Clock Sample Time

LPTIM_CLOCKSAMPLETIME_DIRECTTRANSITION

LPTIM_CLOCKSAMPLETIME_2TRANSITIONS

LPTIM_CLOCKSAMPLETIME_4TRANSITIONS

LPTIM_CLOCKSAMPLETIME_8TRANSITIONS

LPTIM Clock Source

LPTIM_CLOCKSOURCE_APBCLOCK_LPOSC

LPTIM_CLOCKSOURCE_ULPTIM

LPTIM Counter Source

LPTIM_COUNTERSOURCE_INTERNAL

LPTIM_COUNTERSOURCE_EXTERNAL

LPTIM DMA Handle Index

LPTIM_DMA_ID_UPDATE

Index of the DMA handle used for Update DMA requests

LPTIM_DMA_ID_CC1

Index of the DMA handle used for Capture/Update event 1 DMA request

LPTIM_DMA_ID_CC2

Index of the DMA handle used for Capture/Update event 2 DMA request

LPTIM DMA Sources

LPTIM_DMA_UPDATE

DMA request is triggered by the update event

LPTIM_DMA_CC1

DMA request is triggered by the capture 1 event

LPTIM_DMA_CC2

DMA request is triggered by the capture 2 event

LPTIM Exported Macros

__HAL_LPTIM_RESET_HANDLE_STATE

Description:

- Reset LPTIM handle state.

Parameters:

- __HANDLE__: LPTIM handle

Return value:

- None

__HAL_LPTIM_ENABLE

Description:

- Enable the LPTIM peripheral.

Parameters:

- __HANDLE__: LPTIM handle

Return value:

- None

__HAL_LPTIM_DISABLE

Description:

- Disable the LPTIM peripheral.

Parameters:

- __HANDLE__: LPTIM handle

Return value:

- None

__HAL_LPTIM_START_CONTINUOUS

Description:

- Start the LPTIM peripheral in Continuous mode.

Parameters:

- __HANDLE__: LPTIM handle

Return value:

- None

__HAL_LPTIM_START_SINGLE

Description:

- Start the LPTIM peripheral in single mode.

Parameters:

- __HANDLE__: LPTIM handle

Return value:

- None

__HAL_LPTIM_RESET_COUNTER

Description:

- Reset the LPTIM Counter register in synchronous mode.

Parameters:

- __HANDLE__: LPTIM handle

Return value:

- None

__HAL_LPTIM_RESET_COUNTER_AFTERREAD

Description:

- Reset after read of the LPTIM Counter register in asynchronous mode.

Parameters:

- __HANDLE__: LPTIM handle

Return value:

- None

__HAL_LPTIM_AUTORELOAD_SET

Description:

- Write the passed parameter in the Autoreload register.

Parameters:

- __HANDLE__: LPTIM handle
- __VALUE__: Autoreload value

Return value:

- None

Notes:

- The ARR register can only be modified when the LPTIM instance is enabled.

__HAL_LPTIM_COMPARE_SET

Description:

- Write the passed parameter in the Compare register.

Parameters:

- __HANDLE__: LPTIM handle
- __VALUE__: Compare value
- __CHANNEL__: TIM Channel to be configured

Return value:

- None

Notes:

- The CCRx registers can only be modified when the LPTIM instance is enabled.

__HAL_LPTIM_REPETITIONCOUNTER_SET

Description:

- Write the passed parameter in the Repetition register.

Parameters:

- __HANDLE__: LPTIM handle
- __VALUE__: Repetition value

Return value:

- None

__HAL_LPTIM_REPETITIONCOUNTER_GET

Description:

- Return the current Repetition value.

Parameters:

- __HANDLE__: LPTIM handle

Return value:

- Repetition: register value

Notes:

- The RCR register can only be modified when the LPTIM instance is enabled.

__HAL_LPTIM_CAPTURE_COMPARE_ENABLE

Description:

- Enable the LPTIM signal input/output on the corresponding pin.

Parameters:

- __HANDLE__: LPTIM handle
- __CHANNEL__: LPTIM Channels to be enabled. This parameter can be one of the following values:
  - LPTIM_CHANNEL_1: LPTIM Channel 1 selected
  - LPTIM_CHANNEL_2: LPTIM Channel 2 selected

Return value:

- None

__HAL_LPTIM_CAPTURE_COMPARE_DISABLE

Description:

- Disable the LPTIM signal input/output on the corresponding pin.

Parameters:

- __HANDLE__: LPTIM handle
- __CHANNEL__: LPTIM Channels to be disabled. This parameter can be one of the following values:
  - LPTIM_CHANNEL_1: LPTIM Channel 1 selected
  - LPTIM_CHANNEL_2: LPTIM Channel 2 selected

Return value:

- None

__HAL_LPTIM_GET_FLAG

Description:

- Check whether the specified LPTIM flag is set or not.

Parameters:

- __HANDLE__: LPTIM handle
- __FLAG__: LPTIM flag to check This parameter can be a value of:
  - LPTIM_FLAG_REPOK : Repetition register update OK Flag.
  - LPTIM_FLAG_UPDATE : Update event Flag.
  - LPTIM_FLAG_DOWN : Counter direction change up Flag.
  - LPTIM_FLAG_UP : Counter direction change down to up Flag.
  - LPTIM_FLAG_ARROK : Autoreload register update OK Flag.
  - LPTIM_FLAG_CMP1OK : Compare register 1 update OK Flag.
  - LPTIM_FLAG_CMP2OK : Compare register 2 update OK Flag.
  - LPTIM_FLAG_EXTTRIG : External trigger edge event Flag.
  - LPTIM_FLAG_ARRM : Autoreload match Flag.
  - LPTIM_FLAG_CC1 : Capture/Compare 1 interrupt flag.
  - LPTIM_FLAG_CC2 : Capture/Compare 2 interrupt flag.
  - LPTIM_FLAG_CC1O : Capture/Compare 1 over-capture flag.
  - LPTIM_FLAG_CC2O : Capture/Compare 2 over-capture flag.
  - LPTIM_FLAG_DIEROK : DMA & interrupt enable update OK flag.

Return value:

- The: state of the specified flag (SET or RESET).

__HAL_LPTIM_CLEAR_FLAG

Description:

- Clear the specified LPTIM flag.

Parameters:

- __HANDLE__: LPTIM handle.
- __FLAG__: LPTIM flag to clear. This parameter can be a value of:
  - LPTIM_FLAG_REPOK : Repetition register update OK Flag.
  - LPTIM_FLAG_UPDATE : Update event Flag.
  - LPTIM_FLAG_DOWN : Counter direction change up Flag.
  - LPTIM_FLAG_UP : Counter direction change down to up Flag.
  - LPTIM_FLAG_ARROK : Autoreload register update OK Flag.
  - LPTIM_FLAG_CMP1OK : Compare register 1 update OK Flag.
  - LPTIM_FLAG_CMP2OK : Compare register 2 update OK Flag.
  - LPTIM_FLAG_EXTTRIG : External trigger edge event Flag.
  - LPTIM_FLAG_ARRM : Autoreload match Flag.
  - LPTIM_FLAG_CC1 : Capture/Compare 1 interrupt flag.
  - LPTIM_FLAG_CC2 : Capture/Compare 2 interrupt flag.
  - LPTIM_FLAG_CC1O : Capture/Compare 1 over-capture flag.
  - LPTIM_FLAG_CC2O : Capture/Compare 2 over-capture flag.
  - LPTIM_FLAG_DIEROK : DMA & interrupt enable update OK flag.

Return value:

- None.

__HAL_LPTIM_ENABLE_IT

Description:

- Enable the specified LPTIM interrupt.

Parameters:

- __HANDLE__: LPTIM handle.
- __INTERRUPT__: LPTIM interrupt to set. This parameter can be a value of:
  - LPTIM_IT_REPOK : Repetition register update OK Interrupt.
  - LPTIM_IT_UPDATE : Update event register Interrupt.
  - LPTIM_IT_DOWN : Counter direction change up Interrupt.
  - LPTIM_IT_UP : Counter direction change down to up Interrupt.
  - LPTIM_IT_ARROK : Autoreload register update OK Interrupt.
  - LPTIM_IT_CMP1OK : Compare register 1 update OK Interrupt.
  - LPTIM_IT_CMP2OK : Compare register 2 update OK Interrupt.
  - LPTIM_IT_EXTTRIG : External trigger edge event Interrupt.
  - LPTIM_IT_ARRM : Autoreload match Interrupt.
  - LPTIM_IT_CC1 : Capture/Compare 1 interrupt Interrupt.
  - LPTIM_IT_CC2 : Capture/Compare 2 interrupt Interrupt.
  - LPTIM_IT_CC1O : Capture/Compare 1 over-capture Interrupt.
  - LPTIM_IT_CC2O : Capture/Compare 2 over-capture Interrupt.

Return value:

- None.

Notes:

- The LPTIM interrupts can only be enabled when the LPTIM instance is enabled.

__HAL_LPTIM_DISABLE_IT

Description:

- Disable the specified LPTIM interrupt.

Parameters:

- __HANDLE__: LPTIM handle.
- __INTERRUPT__: LPTIM interrupt to set. This parameter can be a value of:
  - LPTIM_IT_REPOK : Repetition register update OK Interrupt.
  - LPTIM_IT_UPDATE : Update event register Interrupt.
  - LPTIM_IT_DOWN : Counter direction change up Interrupt.
  - LPTIM_IT_UP : Counter direction change down to up Interrupt.
  - LPTIM_IT_ARROK : Autoreload register update OK Interrupt.
  - LPTIM_IT_CMP1OK : Compare register 1 update OK Interrupt.
  - LPTIM_IT_CMP2OK : Compare register 2 update OK Interrupt.
  - LPTIM_IT_EXTTRIG : External trigger edge event Interrupt.
  - LPTIM_IT_ARRM : Autoreload match Interrupt.
  - LPTIM_IT_CC1 : Capture/Compare 1 interrupt Interrupt.
  - LPTIM_IT_CC2 : Capture/Compare 2 interrupt Interrupt.
  - LPTIM_IT_CC1O : Capture/Compare 1 over-capture Interrupt.
  - LPTIM_IT_CC2O : Capture/Compare 2 over-capture Interrupt.

Return value:

- None.

Notes:

- The LPTIM interrupts can only be disabled when the LPTIM instance is enabled.

__HAL_LPTIM_ENABLE_DMA

Description:

- Enable the specified DMA request.

Parameters:

- __HANDLE__: specifies the TIM Handle.
- __DMA__: specifies the LPTIM DMA request to enable. This parameter can be one of the following values:
  - LPTIM_DMA_UPDATE: Update DMA request
  - LPTIM_DMA_CC1: Capture/Compare 1 DMA request
  - LPTIM_DMA_CC2: Capture/Compare 2 DMA request

Return value:

- None

__HAL_LPTIM_DISABLE_DMA

Description:

- Disable the specified DMA request.

Parameters:

- __HANDLE__: specifies the LPTIM Handle.
- __DMA__: specifies the LPTIM DMA request to disable. This parameter can be one of the following values:
  - LPTIM_DMA_UPDATE: Update DMA request
  - LPTIM_DMA_CC1: Capture/Compare 1 DMA request
  - LPTIM_DMA_CC2: Capture/Compare 2 DMA request

Return value:

- None

__HAL_LPTIM_GET_IT_SOURCE

Description:

- Check whether the specified LPTIM interrupt source is enabled or not.

Parameters:

- __HANDLE__: LPTIM handle.
- __INTERRUPT__: LPTIM interrupt to check. This parameter can be a value of:
  - LPTIM_IT_REPOK : Repetition register update OK Interrupt.
  - LPTIM_IT_UPDATE : Update event register Interrupt.
  - LPTIM_IT_DOWN : Counter direction change up Interrupt.
  - LPTIM_IT_UP : Counter direction change down to up Interrupt.
  - LPTIM_IT_ARROK : Autoreload register update OK Interrupt.
  - LPTIM_IT_CMP1OK : Compare register 1 update OK Interrupt.
  - LPTIM_IT_CMP2OK : Compare register 2 update OK Interrupt.
  - LPTIM_IT_EXTTRIG : External trigger edge event Interrupt.
  - LPTIM_IT_ARRM : Autoreload match Interrupt.
  - LPTIM_IT_CC1 : Capture/Compare 1 interrupt Interrupt.
  - LPTIM_IT_CC2 : Capture/Compare 2 interrupt Interrupt.
  - LPTIM_IT_CC1O : Capture/Compare 1 over-capture Interrupt.
  - LPTIM_IT_CC2O : Capture/Compare 2 over-capture Interrupt.

Return value:

- Interrupt: status.

__HAL_LPTIM_LPTIM1_EXTI_ENABLE_IT

Description:

- Enable the LPTIM1 EXTI line in interrupt mode.

Return value:

- None

__HAL_LPTIM_LPTIM1_EXTI_DISABLE_IT

Description:

- Disable the LPTIM1 EXTI line in interrupt mode.

Return value:

- None

__HAL_LPTIM_LPTIM2_EXTI_ENABLE_IT

Description:

- Enable the LPTIM2 EXTI line in interrupt mode.

Return value:

- None

__HAL_LPTIM_LPTIM2_EXTI_DISABLE_IT

Description:

- Disable the LPTIM2 EXTI line in interrupt mode.

Return value:

- None

__HAL_LPTIM_LPTIM3_EXTI_ENABLE_IT

Description:

- Enable the LPTIM3 EXTI line in interrupt mode.

Return value:

- None

__HAL_LPTIM_LPTIM3_EXTI_DISABLE_IT

Description:

- Disable the LPTIM3 EXTI line in interrupt mode.

Return value:

- None

__HAL_LPTIM_LPTIM4_EXTI_ENABLE_IT

Description:

- Enable the LPTIM4 EXTI line in interrupt mode.

Return value:

- None

__HAL_LPTIM_LPTIM4_EXTI_DISABLE_IT

Description:

- Disable the LPTIM4 EXTI line in interrupt mode.

Return value:

- None

__HAL_LPTIM_LPTIM5_EXTI_ENABLE_IT

Description:

- Enable the LPTIM5 EXTI line in interrupt mode.

Return value:

- None

__HAL_LPTIM_LPTIM5_EXTI_DISABLE_IT

Description:

- Disable the LPTIM5 EXTI line in interrupt mode.

Return value:

- None

LPTIM Exported Types

LPTIM_EXTI_LINE_LPTIM1

External interrupt line 52 Connected to the LPTIM1 EXTI Line

LPTIM_EXTI_LINE_LPTIM2

External interrupt line 53 Connected to the LPTIM2 EXTI Line

LPTIM_EXTI_LINE_LPTIM3

External interrupt line 55 Connected to the LPTIM3 EXTI Line

LPTIM_EXTI_LINE_LPTIM4

External interrupt line 57 Connected to the LPTIM4 EXTI Line

LPTIM_EXTI_LINE_LPTIM5

External interrupt line 58 Connected to the LPTIM5 EXTI Line

LPTIM External Trigger Polarity

LPTIM_ACTIVEEDGE_RISING

LPTIM_ACTIVEEDGE_FALLING

LPTIM_ACTIVEEDGE_RISING_FALLING

LPTIM Flags Definition

LPTIM_FLAG_CC1O

LPTIM_FLAG_CC2O

LPTIM_FLAG_CC1

LPTIM_FLAG_CC2

LPTIM_FLAG_CMP1OK

LPTIM_FLAG_CMP2OK

LPTIM_FLAG_DIEROK

LPTIM_FLAG_REPOK

LPTIM_FLAG_UPDATE

LPTIM_FLAG_DOWN

LPTIM_FLAG_UP

LPTIM_FLAG_ARROK

LPTIM_FLAG_EXTTRIG

LPTIM_FLAG_ARRM

LPTIM Input1 Source

LPTIM_INPUT1SOURCE_GPIO

For LPTIM1 and LPTIM2

LPTIM Input2 Source

LPTIM_INPUT2SOURCE_GPIO

For LPTIM1 and LPTIM2

LPTIM Input Capture Filter

LPTIM_ICFLT_CLOCK_DIV1

any external input capture signal level change is considered as a valid transition

LPTIM_ICFLT_CLOCK_DIV2

external input capture signal level change must be stable for at least 2 clock periods before it is considered as valid transition

LPTIM_ICFLT_CLOCK_DIV4

external input capture signal level change must be stable for at least 4 clock periods before it is considered as valid transition

LPTIM_ICFLT_CLOCK_DIV8

external input capture signal level change must be stable for at least 8 clock periods before it is considered as valid transition

LPTIM Input Capture Polarity

LPTIM_ICPOLARITY_RISING

Capture/Compare input rising polarity

LPTIM_ICPOLARITY_FALLING

Capture/Compare input falling polarity

LPTIM_ICPOLARITY_RISING_FALLING

Capture/Compare input rising and falling polarities

LPTIM Input Capture Prescaler

LPTIM_ICPSC_DIV1

Capture performed each time an edge is detected on the capture input

LPTIM_ICPSC_DIV2

Capture performed once every 2 events

LPTIM_ICPSC_DIV4

Capture performed once every 4 events

LPTIM_ICPSC_DIV8

Capture performed once every 8 events

LPTIM Input Capture Source

LPTIM_IC1SOURCE_GPIO

For LPTIM1, LPTIM2 and LPTIM3

LPTIM_IC1SOURCE_I3C1_IBIACK

For LPTIM2 and LPTIM3

LPTIM_IC2SOURCE_GPIO

For LPTIM1, LPTIM2 and LPTIM3

LPTIM_IC2SOURCE_LSI

For LPTIM1

LPTIM_IC2SOURCE_LSE

For LPTIM1

LPTIM_IC2SOURCE_HSI_1024

For LPTIM2

LPTIM_IC2SOURCE_MSI_128

For LPTIM2

LPTIM_IC2SOURCE_I3C2_IBIACK

For LPTIM2 and LPTIM3

LPTIM Interrupts Definition

LPTIM_IT_CC1O

LPTIM_IT_CC2O

LPTIM_IT_CC1

LPTIM_IT_CC2

LPTIM_IT_CMP1OK

LPTIM_IT_CMP2OK

LPTIM_IT_REPOK

LPTIM_IT_UPDATE

LPTIM_IT_DOWN

LPTIM_IT_UP

LPTIM_IT_ARROK

LPTIM_IT_EXTTRIG

LPTIM_IT_ARRM

LPTIM Output Compare Polarity

LPTIM_OCPOLARITY_HIGH

Capture/Compare output polarity

LPTIM_OCPOLARITY_LOW

Capture/Compare output polarity

LPTIM Trigger Sample Time

LPTIM_TRIGSAMPLETIME_DIRECTTRANSITION

LPTIM_TRIGSAMPLETIME_2TRANSITIONS

LPTIM_TRIGSAMPLETIME_4TRANSITIONS

LPTIM_TRIGSAMPLETIME_8TRANSITIONS

LPTIM Trigger Source

LPTIM_TRIGSOURCE_SOFTWARE

LPTIM_TRIGSOURCE_0

LPTIM_TRIGSOURCE_1

LPTIM_TRIGSOURCE_2

LPTIM_TRIGSOURCE_3

LPTIM_TRIGSOURCE_4

LPTIM_TRIGSOURCE_5

LPTIM_TRIGSOURCE_6

LPTIM_TRIGSOURCE_7

LPTIM Updating Mode

LPTIM_UPDATE_IMMEDIATE

LPTIM_UPDATE_ENDOFPERIOD

43 HAL LTDC Generic Driver

43.1 LTDC Firmware driver registers structures

43.1.1 LTDC_ColorTypeDef

LTDC_ColorTypeDef is defined in the stm32n6xx_hal_ltdc.h

Data Fields

- ***uint8_t Blue***
- ***uint8_t Green***
- ***uint8_t Red***
- ***uint8_t Reserved***

Field Documentation

- ***uint8_t LTDC_ColorTypeDef::Blue***
  Configures the blue value. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF.
- ***uint8_t LTDC_ColorTypeDef::Green***
  Configures the green value. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF.
- ***uint8_t LTDC_ColorTypeDef::Red***
  Configures the red value. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF.
- ***uint8_t LTDC_ColorTypeDef::Reserved***
  Reserved 0xFF

43.1.2 LTDC_InitTypeDef

LTDC_InitTypeDef is defined in the stm32n6xx_hal_ltdc.h

Data Fields

- ***uint32_t HSPolarity***
- ***uint32_t VSPolarity***
- ***uint32_t DEPolarity***
- ***uint32_t PCPolarity***
- ***uint32_t HorizontalSync***
- ***uint32_t VerticalSync***
- ***uint32_t AccumulatedHBP***
- ***uint32_t AccumulatedVBP***
- ***uint32_t AccumulatedActiveW***
- ***uint32_t AccumulatedActiveH***
- ***uint32_t TotalWidth***
- ***uint32_t TotalHeigh***
- ***LTDC_ColorTypeDef Backcolor***

Field Documentation

- ***uint32_t LTDC_InitTypeDef::HSPolarity***
  configures the horizontal synchronization polarity. This parameter can be one value of ***LTDC_HS_POLARITY***
- ***uint32_t LTDC_InitTypeDef::VSPolarity***
  configures the vertical synchronization polarity. This parameter can be one value of ***LTDC_VS_POLARITY***
- ***uint32_t LTDC_InitTypeDef::DEPolarity***
  configures the data enable polarity. This parameter can be one of value of ***LTDC_DE_POLARITY***
- ***uint32_t LTDC_InitTypeDef::PCPolarity***
  configures the pixel clock polarity. This parameter can be one of value of ***LTDC_PC_POLARITY***

- ***uint32_t LTDC_InitTypeDef::HorizontalSync***
  configures the number of Horizontal synchronization width. This parameter must be a number between Min_Data = 0x000 and Max_Data = 0xFFF.
- ***uint32_t LTDC_InitTypeDef::VerticalSync***
  configures the number of Vertical synchronization height. This parameter must be a number between Min_Data = 0x000 and Max_Data = 0x7FF.
- ***uint32_t LTDC_InitTypeDef::AccumulatedHBP***
  configures the accumulated horizontal back porch width. This parameter must be a number between Min_Data = LTDC_HorizontalSync and Max_Data = 0xFFF.
- ***uint32_t LTDC_InitTypeDef::AccumulatedVBP***
  configures the accumulated vertical back porch height. This parameter must be a number between Min_Data = LTDC_VerticalSync and Max_Data = 0x7FF.
- ***uint32_t LTDC_InitTypeDef::AccumulatedActiveW***
  configures the accumulated active width. This parameter must be a number between Min_Data = LTDC_AccumulatedHBP and Max_Data = 0xFFF.
- ***uint32_t LTDC_InitTypeDef::AccumulatedActiveH***
  configures the accumulated active height. This parameter must be a number between Min_Data = LTDC_AccumulatedVBP and Max_Data = 0x7FF.
- ***uint32_t LTDC_InitTypeDef::TotalWidth***
  configures the total width. This parameter must be a number between Min_Data = LTDC_AccumulatedActiveW and Max_Data = 0xFFF.
- ***uint32_t LTDC_InitTypeDef::TotalHeigh***
  configures the total height. This parameter must be a number between Min_Data = LTDC_AccumulatedActiveH and Max_Data = 0x7FF.
- ***LTDC_ColorTypeDef LTDC_InitTypeDef::Backcolor***
  Configures the background color.

43.1.3 LTDC_LayerCfgTypeDef

LTDC_LayerCfgTypeDef is defined in the stm32n6xx_hal_ltdc.h

Data Fields

- ***uint32_t WindowX0***
- ***uint32_t WindowX1***
- ***uint32_t WindowY0***
- ***uint32_t WindowY1***
- ***uint32_t PixelFormat***
- ***uint32_t Alpha***
- ***uint32_t Alpha0***
- ***uint32_t BlendingFactor1***
- ***uint32_t BlendingFactor2***
- ***uint32_t FBStartAdress***
- ***uint32_t ImageWidth***
- ***uint32_t ImageHeight***
- ***LTDC_ColorTypeDef Backcolor***

Field Documentation

- ***uint32_t LTDC_LayerCfgTypeDef::WindowX0***
  Configures the Window Horizontal Start Position. This parameter must be a number between Min_Data = 0x000 and Max_Data = 0xFFF.
- ***uint32_t LTDC_LayerCfgTypeDef::WindowX1***
  Configures the Window Horizontal Stop Position. This parameter must be a number between Min_Data = 0x000 and Max_Data = 0xFFF.
- ***uint32_t LTDC_LayerCfgTypeDef::WindowY0***
  Configures the Window vertical Start Position. This parameter must be a number between Min_Data = 0x000 and Max_Data = 0x7FF.

- ***uint32_t LTDC_LayerCfgTypeDef::WindowY1***
  Configures the Window vertical Stop Position. This parameter must be a number between Min_Data = 0x0000 and Max_Data = 0x7FF.
- ***uint32_t LTDC_LayerCfgTypeDef::PixelFormat***
  Specifies the pixel format. This parameter can be one of value of ***LTDC_Pixelformat***
- ***uint32_t LTDC_LayerCfgTypeDef::Alpha***
  Specifies the constant alpha used for blending. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF.
- ***uint32_t LTDC_LayerCfgTypeDef::Alpha0***
  Configures the default alpha value. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF.
- ***uint32_t LTDC_LayerCfgTypeDef::BlendingFactor1***
  Select the blending factor 1. This parameter can be one of value of ***LTDC_BlendingFactor1***
- ***uint32_t LTDC_LayerCfgTypeDef::BlendingFactor2***
  Select the blending factor 2. This parameter can be one of value of ***LTDC_BlendingFactor2***
- ***uint32_t LTDC_LayerCfgTypeDef::FBStartAdress***
  Configures the color frame buffer address
- ***uint32_t LTDC_LayerCfgTypeDef::ImageWidth***
  Configures the color frame buffer line length. This parameter must be a number between Min_Data = 0x0000 and Max_Data = 0x1FFF.
- ***uint32_t LTDC_LayerCfgTypeDef::ImageHeight***
  Specifies the number of line in frame buffer. This parameter must be a number between Min_Data = 0x000 and Max_Data = 0x7FF.
- ***LTDC_ColorTypeDef LTDC_LayerCfgTypeDef::Backcolor***
  Configures the layer background color.

43.1.4 LTDC_LayerTypeDef

LTDC_LayerTypeDef is defined in the stm32n6xx_hal_ltdc.h

Data Fields

- ***uint32_t WindowX0***
- ***uint32_t WindowX1***
- ***uint32_t WindowY0***
- ***uint32_t WindowY1***
- ***uint32_t Alpha***
- ***uint32_t Alpha0***
- ***uint32_t BlendingFactor1***
- ***uint32_t BlendingFactor2***
- ***uint32_t ImageWidth***
- ***uint32_t ImageHeight***
- ***LTDC_ColorTypeDef Backcolor***

Field Documentation

- ***uint32_t LTDC_LayerTypeDef::WindowX0***
  Configures the Window Horizontal Start Position. This parameter must be a number between Min_Data = 0x000 and Max_Data = 0xFFF.
- ***uint32_t LTDC_LayerTypeDef::WindowX1***
  Configures the Window Horizontal Stop Position. This parameter must be a number between Min_Data = 0x000 and Max_Data = 0xFFF.
- ***uint32_t LTDC_LayerTypeDef::WindowY0***
  Configures the Window vertical Start Position. This parameter must be a number between Min_Data = 0x000 and Max_Data = 0x7FF.
- ***uint32_t LTDC_LayerTypeDef::WindowY1***
  Configures the Window vertical Stop Position. This parameter must be a number between Min_Data = 0x0000 and Max_Data = 0x7FF.

- ***uint32_t LTDC_LayerTypeDef::Alpha***
  Specifies the constant alpha used for blending. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF.
- ***uint32_t LTDC_LayerTypeDef::Alpha0***
  Configures the default alpha value. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF.
- ***uint32_t LTDC_LayerTypeDef::BlendingFactor1***
  Select the blending factor 1. This parameter can be one of value of ***LTDC_BlendingFactor1***
- ***uint32_t LTDC_LayerTypeDef::BlendingFactor2***
  Select the blending factor 2. This parameter can be one of value of ***LTDC_BlendingFactor2***
- ***uint32_t LTDC_LayerTypeDef::ImageWidth***
  Configures the color frame buffer line length. This parameter must be a number between Min_Data = 0x0000 and Max_Data = 0x1FFF.
- ***uint32_t LTDC_LayerTypeDef::ImageHeight***
  Specifies the number of line in frame buffer. This parameter must be a number between Min_Data = 0x000 and Max_Data = 0x7FF.
- ***LTDC_ColorTypeDef LTDC_LayerTypeDef::Backcolor***
  Configures the layer background color.

43.1.5 LTDC_FlexARGBTypeDef

LTDC_FlexARGBTypeDef is defined in the stm32n6xx_hal_ltdc.h

Data Fields

- ***uint32_t PixelSize***
- ***uint32_t AlphaPos***
- ***uint32_t RedPos***
- ***uint32_t GreenPos***
- ***uint32_t BluePos***
- ***uint32_t AlphaWidth***
- ***uint32_t RedWidth***
- ***uint32_t GreenWidth***
- ***uint32_t BlueWidth***

Field Documentation

- ***uint32_t LTDC_FlexARGBTypeDef::PixelSize***
  Value of ARGB Pixel size. Could be a value of LTDC_ARGB_SIZE This parameter can be one of value of ***LTDC_ARGB_SIZE***
- ***uint32_t LTDC_FlexARGBTypeDef::AlphaPos***
  Position of the Alpha component inside the pixel memory word (in bits) This parameter must be a number between Min_Data = 0x000 and Max_Data = 0x1F.
- ***uint32_t LTDC_FlexARGBTypeDef::RedPos***
  Position of the Red component inside the pixel memory word (in bits) This parameter must be a number between Min_Data = 0x000 and Max_Data = 0x1F.
- ***uint32_t LTDC_FlexARGBTypeDef::GreenPos***
  Position of the Green component inside the pixel memory word (in bits) This parameter must be a number between Min_Data = 0x000 and Max_Data = 0x1F.
- ***uint32_t LTDC_FlexARGBTypeDef::BluePos***
  Position of the Blue component inside the pixel memory word (in bits) This parameter must be a number between Min_Data = 0x0 and Max_Data = 0xF.
- ***uint32_t LTDC_FlexARGBTypeDef::AlphaWidth***
  Width of the Alpha component in bits This parameter must be a number between Min_Data = 0x0 and Max_Data = 0xF.
- ***uint32_t LTDC_FlexARGBTypeDef::RedWidth***
  Width of the Red component in bits This parameter must be a number between Min_Data = 0x0 and Max_Data = 0xF.

- ***uint32_t LTDC_FlexARGBTypeDef::GreenWidth***
  Width of the Green component in bits This parameter must be a number between Min_Data = 0x0 and Max_Data = 0xF.
- ***uint32_t LTDC_FlexARGBTypeDef::BlueWidth***
  Width of the Blue component in bits This parameter must be a number between Min_Data = 0x0 and Max_Data = 0xF.

43.1.6 LTDC_LayerFlexARGBTypeDef

LTDC_LayerFlexARGBTypeDef is defined in the stm32n6xx_hal_ltdc.h

Data Fields

- ***LTDC_LayerTypeDef Layer***
- ***LTDC_FlexARGBTypeDef FlexARGB***
- ***uint32_t ARGBAddress***

Field Documentation

- ***LTDC_LayerTypeDef LTDC_LayerFlexARGBTypeDef::Layer***
  Pointer to ***Section 43.3.1: LTDC_LayerTypeDef*** structure
- ***LTDC_FlexARGBTypeDef LTDC_LayerFlexARGBTypeDef::FlexARGB***
  Pointer to ***Section 43.3.1: LTDC_FlexARGBTypeDef*** structure
- ***uint32_t LTDC_LayerFlexARGBTypeDef::ARGBAddress***
  Configures the color frame buffer address

43.1.7 LTDC_FlexYUVTypeDef

LTDC_FlexYUVTypeDef is defined in the stm32n6xx_hal_ltdc.h

Data Fields

- ***uint32_t YUVOrder***
- ***uint32_t LuminanceOrder***
- ***uint32_t ChrominanceOrder***
- ***uint32_t LuminanceRescale***

Field Documentation

- ***uint32_t LTDC_FlexYUVTypeDef::YUVOrder***
  Y/UV order. This parameter can be one of value of ***LTDC_YUV_ORDER***
- ***uint32_t LTDC_FlexYUVTypeDef::LuminanceOrder***
  Y1/Y2 order. This parameter can be one of value of ***LTDC_LUMINANCE_ORDER***
- ***uint32_t LTDC_FlexYUVTypeDef::ChrominanceOrder***
  U/V order. This parameter can be one of value of ***LTDC_CHROMINANCE_ORDER***
- ***uint32_t LTDC_FlexYUVTypeDef::LuminanceRescale***
  Enable or disable luminance rescale. This parameter can be one of value of ***LTDC_LUMINANCE_RANGE***

43.1.8 LTDC_LayerFlexYUVCoPlanarTypeDef

LTDC_LayerFlexYUVCoPlanarTypeDef is defined in the stm32n6xx_hal_ltdc.h

Data Fields

- ***LTDC_LayerTypeDef Layer***
- ***LTDC_FlexYUVTypeDef FlexYUV***
- ***uint32_t YUVAddress***
- ***uint32_t ColorConverter***

Field Documentation

- ***LTDC_LayerTypeDef LTDC_LayerFlexYUVCoPlanarTypeDef::Layer***
  Pointer to ***Section 43.3.1: LTDC_LayerTypeDef*** structure
- ***LTDC_FlexYUVTypeDef LTDC_LayerFlexYUVCoPlanarTypeDef::FlexYUV***
  Pointer to ***Section 43.3.1: LTDC_FlexYUVTypeDef*** structure
- ***uint32_t LTDC_LayerFlexYUVCoPlanarTypeDef::YUVAddress***
  YUV Frame Buffer address

- ***uint32_t LTDC_LayerFlexYUVCoPlanarTypeDef::ColorConverter***
  Configure the YUV to RGB conversion. This parameter can be one of value of ***LTDC_YUV2RGBCONVERTOR***

43.1.9 LTDC_YUVSemiPlanarAddressTypeDef

LTDC_YUVSemiPlanarAddressTypeDef is defined in the stm32n6xx_hal_ltdc.h

Data Fields

- ***uint32_t YAddress***
- ***uint32_t UVAddress***

Field Documentation

- ***uint32_t LTDC_YUVSemiPlanarAddressTypeDef::YAddress***
  Y Frame Buffer address
- ***uint32_t LTDC_YUVSemiPlanarAddressTypeDef::UVAddress***
  UV Frame Buffer address

43.1.10 LTDC_YUVFullPlanarAddressTypeDef

LTDC_YUVFullPlanarAddressTypeDef is defined in the stm32n6xx_hal_ltdc.h

Data Fields

- ***uint32_t YAddress***
- ***uint32_t UAddress***
- ***uint32_t VAddress***

Field Documentation

- ***uint32_t LTDC_YUVFullPlanarAddressTypeDef::YAddress***
  Y Frame Buffer address
- ***uint32_t LTDC_YUVFullPlanarAddressTypeDef::UAddress***
  U Frame Buffer address
- ***uint32_t LTDC_YUVFullPlanarAddressTypeDef::VAddress***
  V Frame Buffer address

43.1.11 LTDC_LayerFlexYUVSemiPlanarTypeDef

LTDC_LayerFlexYUVSemiPlanarTypeDef is defined in the stm32n6xx_hal_ltdc.h

Data Fields

- ***LTDC_LayerTypeDef Layer***
- ***LTDC_FlexYUVTypeDef FlexYUV***
- ***LTDC_YUVSemiPlanarAddressTypeDef YUVSemiPlanarAddress***
- ***uint32_t ColorConverter***

Field Documentation

- ***LTDC_LayerTypeDef LTDC_LayerFlexYUVSemiPlanarTypeDef::Layer***
  Pointer to ***Section 43.3.1: LTDC_LayerTypeDef***
- ***LTDC_FlexYUVTypeDef LTDC_LayerFlexYUVSemiPlanarTypeDef::FlexYUV***
  Pointer ***Section 43.3.1: LTDC_FlexYUVTypeDef***
- ***LTDC_YUVSemiPlanarAddressTypeDef LTDC_LayerFlexYUVSemiPlanarTypeDef::YUVSemiPlanarAddress***
  Pointer ***Section 43.3.1: LTDC_YUVSemiPlanarAddressTypeDef***
- ***uint32_t LTDC_LayerFlexYUVSemiPlanarTypeDef::ColorConverter***
  Configure the YUV to RGB conversion. This parameter can be one of value of ***LTDC_YUV2RGBCONVERTOR***

43.1.12 LTDC_LayerFlexYUVFullPlanarTypeDef

LTDC_LayerFlexYUVFullPlanarTypeDef is defined in the stm32n6xx_hal_ltdc.h

Data Fields

- ***LTDC_LayerTypeDef Layer***

- ***LTDC_FlexYUVTypeDef FlexYUV***
- ***LTDC_YUVFullPlanarAddressTypeDef YUVFullPlanarAddress***
- ***uint32_t ColorConverter***

Field Documentation

- ***LTDC_LayerTypeDef LTDC_LayerFlexYUVFullPlanarTypeDef::Layer***
  Pointer to ***Section 43.3.1: LTDC_LayerTypeDef***
- ***LTDC_FlexYUVTypeDef LTDC_LayerFlexYUVFullPlanarTypeDef::FlexYUV***
  Pointer ***Section 43.3.1: LTDC_FlexYUVTypeDef***
- ***LTDC_YUVFullPlanarAddressTypeDef LTDC_LayerFlexYUVFullPlanarTypeDef::YUVFullPlanarAddress***
  Pointer ***Section 43.3.1: LTDC_YUVFullPlanarAddressTypeDef***
- ***uint32_t LTDC_LayerFlexYUVFullPlanarTypeDef::ColorConverter***
  Configure the YUV to RGB conversion. This parameter can be one of value of ***LTDC_YUV2RGBCONVERTOR***

43.1.13 __LTDC_HandleTypeDef

__LTDC_HandleTypeDef is defined in the stm32n6xx_hal_ltdc.h

Data Fields

- ***LTDC_TypeDef * Instance***
- ***LTDC_InitTypeDef Init***
- ***LTDC_LayerCfgTypeDef LayerCfg***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_LTDC_StateTypeDef State***
- ***__IO uint32_t ErrorCode***
- ***void(* LineEventCallback***
- ***void(* ReloadEventCallback***
- ***void(* ErrorCallback***
- ***void(* WarningEventCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***LTDC_TypeDef* __LTDC_HandleTypeDef::Instance***
  LTDC Register base address
- ***LTDC_InitTypeDef __LTDC_HandleTypeDef::Init***
  LTDC parameters
- ***LTDC_LayerCfgTypeDef __LTDC_HandleTypeDef::LayerCfg[MAX_LAYER]***
  LTDC Layers parameters
- ***HAL_LockTypeDef __LTDC_HandleTypeDef::Lock***
  LTDC Lock
- ***__IO HAL_LTDC_StateTypeDef __LTDC_HandleTypeDef::State***
  LTDC state
- ***__IO uint32_t __LTDC_HandleTypeDef::ErrorCode***
  LTDC Error code
- ***void(* __LTDC_HandleTypeDef::LineEventCallback)(struct __LTDC_HandleTypeDef *hltdc)***
  LTDC Line Event Callback
- ***void(* __LTDC_HandleTypeDef::ReloadEventCallback)(struct __LTDC_HandleTypeDef *hltdc)***
  LTDC Reload Event Callback
- ***void(* __LTDC_HandleTypeDef::ErrorCallback)(struct __LTDC_HandleTypeDef *hltdc)***
  LTDC Error Callback
- ***void(* __LTDC_HandleTypeDef::WarningEventCallback)(struct __LTDC_HandleTypeDef *hltdc)***
  LTDC Warning Event
- ***void(* __LTDC_HandleTypeDef::MspInitCallback)(struct __LTDC_HandleTypeDef *hltdc)***
  LTDC Msp Init callback

- ***void(* __LTDC_HandleTypeDef::MspDeInitCallback)(struct __LTDC_HandleTypeDef *hltdc)***
  LTDC Msp DeInit callback

43.2 LTDC Firmware driver API description

The following section lists the various functions of the LTDC library.

43.2.1 How to use this driver

The LTDC HAL driver can be used as follows:

1. Declare a LTDC_HandleTypeDef handle structure, for example: LTDC_HandleTypeDef hltdc;
2. Initialize the LTDC low level resources by implementing the HAL_LTDC_MspInit() API:
   a. Enable the LTDC interface clock
   b. NVIC configuration if you need to use interrupt process
      - Configure the LTDC interrupt priority
      - Enable the NVIC LTDC IRQ Channel
3. Initialize the required configuration through the following parameters: the LTDC timing, the horizontal and vertical polarity, the pixel clock polarity, Data Enable polarity and the LTDC background color value using HAL_LTDC_Init() function

Configuration

1. Program the required configuration through the following parameters: the pixel format, the blending factors, input alpha value, the window size and the image size using HAL_LTDC_ConfigLayer() function for foreground or/and background layer.
2. Optionally, configure and enable the CLUT using HAL_LTDC_ConfigCLUT() and HAL_LTDC_EnableCLUT functions.
3. Optionally, enable the Dither using HAL_LTDC_EnableDither().
4. Optionally, configure and enable the Color keying using HAL_LTDC_ConfigColorKeying() and HAL_LTDC_EnableColorKeying functions.
5. Optionally, configure LineInterrupt using HAL_LTDC_ProgramLineEvent() function
6. If needed, reconfigure and change the pixel format value, the alpha value value, the window size, the window position and the layer start address for foreground or/and background layer using respectively the following functions: HAL_LTDC_SetPixelFormat(), HAL_LTDC_SetAlpha(), HAL_LTDC_SetWindowSize(), HAL_LTDC_SetWindowPosition() and HAL_LTDC_SetAddress().
7. Variant functions with _NoReload suffix allows to set the LTDC configuration/settings without immediate reload. This is useful in case when the program requires to modify serval LTDC settings (on one or both layers) then applying(reload) these settings in one shot by calling the function HAL_LTDC_Reload(). After calling the _NoReload functions to set different color/format/layer settings, the program shall call the function HAL_LTDC_Reload() to apply(reload) these settings. Function HAL_LTDC_Reload() can be called with the parameter ReloadType set to LTDC_RELOAD_IMMEDIATE if an immediate reload is required. Function HAL_LTDC_Reload() can be called with the parameter ReloadType set to LTDC_RELOAD_VERTICAL_BLANKING if the reload should be done in the next vertical blanking period, this option allows to avoid display flicker by applying the new settings during the vertical blanking period.
8. To control LTDC state you can use the following function: HAL_LTDC_GetState()

LTDC HAL driver macros list

Below the list of most used macros in LTDC HAL driver.

- __HAL_LTDC_ENABLE: Enable the LTDC.
- __HAL_LTDC_DISABLE: Disable the LTDC.
- __HAL_LTDC_LAYER_ENABLE: Enable an LTDC Layer.
- __HAL_LTDC_LAYER_DISABLE: Disable an LTDC Layer.
- __HAL_LTDC_RELOAD_IMMEDIATE_CONFIG: Reload Layer Configuration.
- __HAL_LTDC_GET_FLAG: Get the LTDC pending flags.
- __HAL_LTDC_CLEAR_FLAG: Clear the LTDC pending flags.
- __HAL_LTDC_ENABLE_IT: Enable the specified LTDC interrupts.

- __HAL_LTDC_DISABLE_IT: Disable the specified LTDC interrupts.
- __HAL_LTDC_GET_IT_SOURCE: Check whether the specified LTDC interrupt has occurred or not.

Note: You can refer to the LTDC HAL driver header file for more useful macros

Callback registration

The compilation define USE_HAL_LTDC_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use function HAL_LTDC_RegisterCallback() to register a callback.

Function HAL_LTDC_RegisterCallback() allows to register following callbacks:

- LineEventCallback : LTDC Line Event Callback.
- ReloadEventCallback : LTDC Reload Event Callback.
- ErrorCallback : LTDC Error Callback
- MspInitCallback : LTDC MspInit.
- MspDeInitCallback : LTDC MspDeInit.

This function takes as parameters the HAL peripheral handle, the callback ID and a pointer to the user callback function.

Use function HAL_LTDC_UnRegisterCallback() to reset a callback to the default weak function. HAL_LTDC_UnRegisterCallback() takes as parameters the HAL peripheral handle and the callback ID.

This function allows to reset following callbacks:

- LineEventCallback : LTDC Line Event Callback
- ReloadEventCallback : LTDC Reload Event Callback
- ErrorCallback : LTDC Error Callback
- MspInitCallback : LTDC MspInit
- MspDeInitCallback : LTDC MspDeInit.

By default, after the HAL_LTDC_Init and when the state is HAL_LTDC_STATE_RESET all callbacks are set to the corresponding weak functions: examples HAL_LTDC_LineEventCallback(), HAL_LTDC_ErrorCallback(). Exception done for MspInit and MspDeInit functions that are reset to the legacy weak (surcharged) functions in the HAL_LTDC_Init() and HAL_LTDC_DeInit() only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the HAL_LTDC_Init() and HAL_LTDC_DeInit() keep and use the user MspInit/MspDeInit callbacks (registered beforehand).

Callbacks can be registered/unregistered in HAL_LTDC_STATE_READY state only. Exception done MspInit/ MspDeInit that can be registered/unregistered in HAL_LTDC_STATE_READY or HAL_LTDC_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_LTDC_RegisterCallback() before calling HAL_LTDC_DeInit() or HAL_LTDC_Init() function.

When the compilation define USE_HAL_LTDC_REGISTER_CALLBACKS is set to 0 or not defined, the callback registration feature is not available and all callbacks are set to the corresponding weak functions.

43.2.2 Detailed description of functions

HAL_LTDC_Init

Function name

HAL_StatusTypeDef HAL_LTDC_Init (LTDC_HandleTypeDef * hltdc)

Function description

HAL_LTDC_DeInit

Function name

HAL_StatusTypeDef HAL_LTDC_DeInit (LTDC_HandleTypeDef * hltdc)

Function description

HAL_LTDC_MspInit

Function name

void HAL_LTDC_MspInit (LTDC_HandleTypeDef * hltdc)

Function description

HAL_LTDC_MspDeInit

Function name

void HAL_LTDC_MspDeInit (LTDC_HandleTypeDef * hltdc)

Function description

HAL_LTDC_ErrorCallback

Function name

void HAL_LTDC_ErrorCallback (LTDC_HandleTypeDef * hltdc)

Function description

HAL_LTDC_LineEventCallback

Function name

void HAL_LTDC_LineEventCallback (LTDC_HandleTypeDef * hltdc)

Function description

HAL_LTDC_ReloadEventCallback

Function name

void HAL_LTDC_ReloadEventCallback (LTDC_HandleTypeDef * hltdc)

Function description

HAL_LTDC_WarningEventCallback

Function name

void HAL_LTDC_WarningEventCallback (LTDC_HandleTypeDef * hltdc)

Function description

HAL_LTDC_ConfigBurstLength

Function name

HAL_StatusTypeDef HAL_LTDC_ConfigBurstLength (LTDC_HandleTypeDef * hltdc, uint32_t BurstLength, uint32_t LayerIdx)

Function description

HAL_LTDC_ConfigUnderrunThreshold

Function name

HAL_StatusTypeDef HAL_LTDC_ConfigUnderrunThreshold (LTDC_HandleTypeDef * hltdc, uint16_t Threshold)

Function description

HAL_LTDC_ConfigMirror

Function name

HAL_StatusTypeDef HAL_LTDC_ConfigMirror (LTDC_HandleTypeDef * hltdc, uint32_t Mirror, uint32_t LayerIdx)

Function description

HAL_LTDC_RegisterCallback

Function name

HAL_StatusTypeDef HAL_LTDC_RegisterCallback (LTDC_HandleTypeDef * hltdc, HAL_LTDC_CallbackIDTypeDef CallbackID, pLTDC_CallbackTypeDef pCallback)

Function description

HAL_LTDC_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_LTDC_UnRegisterCallback (LTDC_HandleTypeDef * hltdc, HAL_LTDC_CallbackIDTypeDef CallbackID)

Function description

HAL_LTDC_IRQHandler

Function name

void HAL_LTDC_IRQHandler (LTDC_HandleTypeDef * hltdc)

Function description

HAL_LTDC_ConfigLayer

Function name

HAL_StatusTypeDef HAL_LTDC_ConfigLayer (LTDC_HandleTypeDef * hltdc, LTDC_LayerCfgTypeDef * pLayerCfg, uint32_t LayerIdx)

Function description

HAL_LTDC_ConfigLayerFlexARGB

Function name

HAL_StatusTypeDef HAL_LTDC_ConfigLayerFlexARGB (LTDC_HandleTypeDef * hltdc, const LTDC_LayerFlexARGBTypeDef * pLayerFlexARGB, uint32_t LayerIdx)

Function description

HAL_LTDC_ConfigLayerFlexYUVSemiPlanar

Function name

HAL_StatusTypeDef HAL_LTDC_ConfigLayerFlexYUVSemiPlanar (LTDC_HandleTypeDef * hltdc, LTDC_LayerFlexYUVSemiPlanarTypeDef * pLayerFlexYUVSemiPlanar, uint32_t LayerIdx)

Function description

HAL_LTDC_ConfigLayerFlexYUVFullPlanar

Function name

HAL_StatusTypeDef HAL_LTDC_ConfigLayerFlexYUVFullPlanar (LTDC_HandleTypeDef * hltdc, LTDC_LayerFlexYUVFullPlanarTypeDef * pLayerFlexYUVFullPlanar, uint32_t LayerIdx)

Function description

HAL_LTDC_ConfigLayerFlexYUVCoPlanar

Function name

HAL_StatusTypeDef HAL_LTDC_ConfigLayerFlexYUVCoPlanar (LTDC_HandleTypeDef * hltdc, const LTDC_LayerFlexYUVCoPlanarTypeDef * pLayerFlexYUVCoPlanar, uint32_t LayerIdx)

Function description

HAL_LTDC_SetWindowSize

Function name

HAL_StatusTypeDef HAL_LTDC_SetWindowSize (LTDC_HandleTypeDef * hltdc, uint32_t XSize, uint32_t YSize, uint32_t LayerIdx)

Function description

HAL_LTDC_SetWindowPosition

Function name

HAL_StatusTypeDef HAL_LTDC_SetWindowPosition (LTDC_HandleTypeDef * hltdc, uint32_t X0, uint32_t Y0, uint32_t LayerIdx)

Function description

HAL_LTDC_SetPixelFormat

Function name

HAL_StatusTypeDef HAL_LTDC_SetPixelFormat (LTDC_HandleTypeDef * hltdc, uint32_t Pixelformat, uint32_t LayerIdx)

Function description

HAL_LTDC_SetAlpha

Function name

HAL_StatusTypeDef HAL_LTDC_SetAlpha (LTDC_HandleTypeDef * hltdc, uint32_t Alpha, uint32_t LayerIdx)

Function description

HAL_LTDC_SetAddress

Function name

HAL_StatusTypeDef HAL_LTDC_SetAddress (LTDC_HandleTypeDef * hltdc, uint32_t Address, uint32_t LayerIdx)

Function description

HAL_LTDC_SetSemiPlanarAddress

Function name

HAL_StatusTypeDef HAL_LTDC_SetSemiPlanarAddress (LTDC_HandleTypeDef * hltdc, LTDC_LayerFlexYUVSemiPlanarTypeDef * pYUVSemiPlanarAddress, uint32_t LayerIdx)

Function description

HAL_LTDC_SetFullPlanarAddress

Function name

HAL_StatusTypeDef HAL_LTDC_SetFullPlanarAddress (LTDC_HandleTypeDef * hltdc, LTDC_LayerFlexYUVFullPlanarTypeDef * pYUVFullPlanarAddress, uint32_t LayerIdx)

Function description

HAL_LTDC_SetSemiPlanarAddress_NoReload

Function name

HAL_StatusTypeDef HAL_LTDC_SetSemiPlanarAddress_NoReload (LTDC_HandleTypeDef * hltdc, LTDC_LayerFlexYUVSemiPlanarTypeDef * pYUVSemiPlanarAddress, uint32_t LayerIdx)

Function description

HAL_LTDC_SetFullPlanarAddress_NoReload

Function name

HAL_StatusTypeDef HAL_LTDC_SetFullPlanarAddress_NoReload (LTDC_HandleTypeDef * hltdc, LTDC_LayerFlexYUVFullPlanarTypeDef * pYUVFullPlanarAddress, uint32_t LayerIdx)

Function description

HAL_LTDC_SetPitch

Function name

HAL_StatusTypeDef HAL_LTDC_SetPitch (LTDC_HandleTypeDef * hltdc, uint32_t LinePitchInPixels, uint32_t LayerIdx)

Function description

HAL_LTDC_ConfigColorKeying

Function name

HAL_StatusTypeDef HAL_LTDC_ConfigColorKeying (LTDC_HandleTypeDef * hltdc, uint32_t RGBValue, uint32_t LayerIdx)

Function description

HAL_LTDC_ConfigCLUT

Function name

HAL_StatusTypeDef HAL_LTDC_ConfigCLUT (LTDC_HandleTypeDef * hltdc, const uint32_t * pCLUT, uint32_t CLUTSize, uint32_t LayerIdx)

Function description

HAL_LTDC_EnableColorKeying

Function name

HAL_StatusTypeDef HAL_LTDC_EnableColorKeying (LTDC_HandleTypeDef * hltdc, uint32_t LayerIdx)

Function description

HAL_LTDC_DisableColorKeying

Function name

HAL_StatusTypeDef HAL_LTDC_DisableColorKeying (LTDC_HandleTypeDef * hltdc, uint32_t LayerIdx)

Function description

HAL_LTDC_ConfigBlendingOrder

Function name

HAL_StatusTypeDef HAL_LTDC_ConfigBlendingOrder (LTDC_HandleTypeDef * hltdc, uint32_t Order, uint32_t LayerIdx)

Function description

HAL_LTDC_EnableDefaultColor

Function name

HAL_StatusTypeDef HAL_LTDC_EnableDefaultColor (LTDC_HandleTypeDef * hltdc, uint32_t LayerIdx)

Function description

HAL_LTDC_DisableDefaultColor

Function name

HAL_StatusTypeDef HAL_LTDC_DisableDefaultColor (LTDC_HandleTypeDef * hltdc, uint32_t LayerIdx)

Function description

HAL_LTDC_EnableGammaCorrection

Function name

HAL_StatusTypeDef HAL_LTDC_EnableGammaCorrection (LTDC_HandleTypeDef * hltdc)

Function description

HAL_LTDC_DisableGammaCorrection

Function name

HAL_StatusTypeDef HAL_LTDC_DisableGammaCorrection (LTDC_HandleTypeDef * hltdc)

Function description

HAL_LTDC_ConfigGammaCorrection

Function name

HAL_StatusTypeDef HAL_LTDC_ConfigGammaCorrection (LTDC_HandleTypeDef * hltdc, uint32_t GammaOnes, uint32_t GammaTenths, uint32_t RGBComponent)

Function description

HAL_LTDC_EnableCLUT

Function name

HAL_StatusTypeDef HAL_LTDC_EnableCLUT (LTDC_HandleTypeDef * hltdc, uint32_t LayerIdx)

Function description

HAL_LTDC_DisableCLUT

Function name

HAL_StatusTypeDef HAL_LTDC_DisableCLUT (LTDC_HandleTypeDef * hltdc, uint32_t LayerIdx)

Function description

HAL_LTDC_ProgramLineEvent

Function name

HAL_StatusTypeDef HAL_LTDC_ProgramLineEvent (LTDC_HandleTypeDef * hltdc, uint32_t Line)

Function description

HAL_LTDC_EnableDither

Function name

HAL_StatusTypeDef HAL_LTDC_EnableDither (LTDC_HandleTypeDef * hltdc)

Function description

HAL_LTDC_DisableDither

Function name

HAL_StatusTypeDef HAL_LTDC_DisableDither (LTDC_HandleTypeDef * hltdc)

Function description

HAL_LTDC_Reload

Function name

HAL_StatusTypeDef HAL_LTDC_Reload (LTDC_HandleTypeDef * hltdc, uint32_t ReloadType)

Function description

HAL_LTDC_ReloadLayer

Function name

HAL_StatusTypeDef HAL_LTDC_ReloadLayer (LTDC_HandleTypeDef * hltdc, uint32_t ReloadType, uint32_t LayerIdx)

Function description

HAL_LTDC_SetOutputDisplay

Function name

HAL_StatusTypeDef HAL_LTDC_SetOutputDisplay (LTDC_HandleTypeDef * hltdc, uint32_t Display)

Function description

HAL_LTDC_SetExpectedCRC

Function name

HAL_StatusTypeDef HAL_LTDC_SetExpectedCRC (LTDC_HandleTypeDef * hltdc, uint16_t ExpectedCRC)

Function description

HAL_LTDC_EnableCRC

Function name

HAL_StatusTypeDef HAL_LTDC_EnableCRC (LTDC_HandleTypeDef * hltdc)

Function description

HAL_LTDC_DisableCRC

Function name

HAL_StatusTypeDef HAL_LTDC_DisableCRC (LTDC_HandleTypeDef * hltdc)

Function description

HAL_LTDC_GetComputedCRC

Function name

HAL_StatusTypeDef HAL_LTDC_GetComputedCRC (LTDC_HandleTypeDef * hltdc, uint16_t * pComputedCRC)

Function description

HAL_LTDC_ConfigLayer_NoReload

Function name

HAL_StatusTypeDef HAL_LTDC_ConfigLayer_NoReload (LTDC_HandleTypeDef * hltdc, LTDC_LayerCfgTypeDef * pLayerCfg, uint32_t LayerIdx)

Function description

HAL_LTDC_SetWindowSize_NoReload

Function name

HAL_StatusTypeDef HAL_LTDC_SetWindowSize_NoReload (LTDC_HandleTypeDef * hltdc, uint32_t XSize, uint32_t YSize, uint32_t LayerIdx)

Function description

HAL_LTDC_SetWindowPosition_NoReload

Function name

HAL_StatusTypeDef HAL_LTDC_SetWindowPosition_NoReload (LTDC_HandleTypeDef * hltdc, uint32_t X0, uint32_t Y0, uint32_t LayerIdx)

Function description

HAL_LTDC_SetPixelFormat_NoReload

Function name

HAL_StatusTypeDef HAL_LTDC_SetPixelFormat_NoReload (LTDC_HandleTypeDef * hltdc, uint32_t Pixelformat, uint32_t LayerIdx)

Function description

HAL_LTDC_SetAlpha_NoReload

Function name

HAL_StatusTypeDef HAL_LTDC_SetAlpha_NoReload (LTDC_HandleTypeDef * hltdc, uint32_t Alpha, uint32_t LayerIdx)

Function description

HAL_LTDC_SetAddress_NoReload

Function name

HAL_StatusTypeDef HAL_LTDC_SetAddress_NoReload (LTDC_HandleTypeDef * hltdc, uint32_t Address, uint32_t LayerIdx)

Function description

HAL_LTDC_SetPitch_NoReload

Function name

HAL_StatusTypeDef HAL_LTDC_SetPitch_NoReload (LTDC_HandleTypeDef * hltdc, uint32_t LinePitchInPixels, uint32_t LayerIdx)

Function description

HAL_LTDC_EnableColorKeying_NoReload

Function name

HAL_StatusTypeDef HAL_LTDC_EnableColorKeying_NoReload (LTDC_HandleTypeDef * hltdc, uint32_t LayerIdx)

Function description

HAL_LTDC_DisableColorKeying_NoReload

Function name

HAL_StatusTypeDef HAL_LTDC_DisableColorKeying_NoReload (LTDC_HandleTypeDef * hltdc, uint32_t LayerIdx)

Function description

HAL_LTDC_EnableCLUT_NoReload

Function name

HAL_StatusTypeDef HAL_LTDC_EnableCLUT_NoReload (LTDC_HandleTypeDef * hltdc, uint32_t LayerIdx)

Function description

HAL_LTDC_DisableCLUT_NoReload

Function name

HAL_StatusTypeDef HAL_LTDC_DisableCLUT_NoReload (LTDC_HandleTypeDef * hltdc, uint32_t LayerIdx)

Function description

HAL_LTDC_GetState

Function name

HAL_LTDC_StateTypeDef HAL_LTDC_GetState (const LTDC_HandleTypeDef * hltdc)

Function description

HAL_LTDC_GetError

Function name

uint32_t HAL_LTDC_GetError (const LTDC_HandleTypeDef * hltdc)

Function description

43.3 LTDC Firmware driver defines

The following section lists the various define and macros of the module.

43.3.1 LTDC

LTDC

LTDC Alpha

LTDC_ALPHA

LTDC Constant Alpha mask

LTDC ARGB pixel size in bytes

LTDC_ARGB_PIXEL_SIZE_1_BYTE

Custom ARGB with 1 byte per pixel

LTDC_ARGB_PIXEL_SIZE_2_BYTES

Custom ARGB with 2 bytes per pixel

LTDC_ARGB_PIXEL_SIZE_3_BYTES

Custom ARGB with 3 bytes per pixel

LTDC_ARGB_PIXEL_SIZE_4_BYTES

Custom ARGB with 4 bytes per pixel

LTDC BACK COLOR

LTDC_COLOR

Color mask

LTDC Blending Factor1

LTDC_BLENDING_FACTOR1_CA

Blending factor : Cte Alpha

LTDC_BLENDING_FACTOR1_PAxCA

Blending factor : Cte Alpha x Pixel Alpha

LTDC Blending Factor2

LTDC_BLENDING_FACTOR2_CA

Blending factor : Cte Alpha

LTDC_BLENDING_FACTOR2_PAxCA

Blending factor : Cte Alpha x Pixel Alpha

LTDC layers blending order

LTDC_BLENDING_ORDER_FOREGROUND

Layer set to Foreground during blending

LTDC_BLENDING_ORDER_BACKGROUND

Layer set to Background during blending

LTDC chrominance order from LSB

LTDC_YUV_CHROMIANCE_ORDER_U_FIRST

Chrominance order U first

LTDC_YUV_CHROMIANCE_ORDER_V_FIRST

Chrominance order V first

LTDC DE POLARITY

LTDC_DEPOLARITY_AL

Data Enable, is active low.

LTDC_DEPOLARITY_AH

Data Enable, is active high.

LTDC Display

LTDC_OUT_RGB

Select RGB as output format (default)

LTDC_OUT_YUV_HDTV

RGB converted to YUV using ITU-R BT.709

LTDC_OUT_YUV_SDTV

RGB converted to YUV using ITU-R BT.601

LTDC_OUT_YVU_HDTV

RGB converted to YVU using ITU-R BT.709

LTDC_OUT_YVU_SDTV

RGB converted to YVU using ITU-R BT.601

LTDC Error Code

HAL_LTDC_ERROR_NONE

LTDC No error

HAL_LTDC_ERROR_TE

LTDC Transfer error

HAL_LTDC_ERROR_FU

LTDC FIFO Underrun

HAL_LTDC_ERROR_CRC

LTDC CRC error

HAL_LTDC_ERROR_TIMEOUT

LTDC Timeout error

HAL_LTDC_ERROR_INVALID_CALLBACK

LTDC Invalid Callback error

LTDC Exported Macros

__HAL_LTDC_RESET_HANDLE_STATE

Description:

- Reset LTDC handle state.

Parameters:

- __HANDLE__: LTDC handle

Return value:

- None

__HAL_LTDC_ENABLE

Description:

- Enable the LTDC.

Parameters:

- __HANDLE__: LTDC handle

Return value:

- None.

__HAL_LTDC_DISABLE

Description:

- Disable the LTDC.

Parameters:

- __HANDLE__: LTDC handle

Return value:

- None.

__HAL_LTDC_LAYER_ENABLE

Description:

- Enable the LTDC Layer.

Parameters:

- __HANDLE__: LTDC handle
- __LAYER__: Specify the layer to be enabled. This parameter can be LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1).

Return value:

- None.

__HAL_LTDC_LAYER_DISABLE

Description:

- Disable the LTDC Layer.

Parameters:

- __HANDLE__: LTDC handle
- __LAYER__: Specify the layer to be disabled. This parameter can be LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1).

Return value:

- None.

__HAL_LTDC_RELOAD_IMMEDIATE_CONFIG

Description:

- Reload immediately all LTDC Layers.

Parameters:

- __HANDLE__: LTDC handle

Return value:

- None.

__HAL_LTDC_VERTICAL_BLANKING_RELOAD_CONFIG

Description:

- Reload during vertical blanking period all LTDC Layers.

Parameters:

- __HANDLE__: LTDC handle

Return value:

- None.

__HAL_LTDC_GET_FLAG

Description:

- Get the LTDC pending flags.

Parameters:

- __HANDLE__: LTDC handle
- __FLAG__: Get the specified flag. This parameter can be any combination of the following values:
  - LTDC_FLAG_LI: Line Interrupt flag
  - LTDC_FLAG_FU: FIFO Underrun Interrupt flag
  - LTDC_FLAG_TE: Transfer Error interrupt flag
  - LTDC_FLAG_RR: Register Reload Interrupt Flag
  - LTDC_FLAG_CRC: LTDC CRC error interrupt flag
  - LTDC_FLAG_FUW: FIFO Warning Underrun interrupt flag

Return value:

- The: state of FLAG (SET or RESET).

__HAL_LTDC_CLEAR_FLAG

Description:

- Clears the LTDC pending flags.

Parameters:

- __HANDLE__: LTDC handle
- __FLAG__: Specify the flag to clear. This parameter can be any combination of the following values:
  - LTDC_FLAG_LI: Line Interrupt flag
  - LTDC_FLAG_FU: FIFO Underrun Interrupt flag
  - LTDC_FLAG_TE: Transfer Error interrupt flag
  - LTDC_FLAG_RR: Register Reload Interrupt Flag
  - LTDC_FLAG_CRC: LTDC CRC error Interrupt flag
  - LTDC_FLAG_FUW: FIFO Warning Underrun Interrupt flag

Return value:

- None

__HAL_LTDC_ENABLE_IT

Description:

- Enables the specified LTDC interrupts.

Parameters:

- __HANDLE__: LTDC handle
- __INTERRUPT__: Specify the LTDC interrupt sources to be enabled. This parameter can be any combination of the following values:
  - LTDC_IT_LI: Line Interrupt flag
  - LTDC_IT_FU: FIFO Underrun Interrupt flag
  - LTDC_IT_TE: Transfer Error interrupt flag
  - LTDC_IT_RR: Register Reload Interrupt Flag
  - LTDC_IT_CRC: LTDC CRC error Interrupt
  - LTDC_IT_FUW: FIFO Warning Underrun Interrupt

Return value:

- None

__HAL_LTDC_DISABLE_IT

Description:

- Disables the specified LTDC interrupts.

Parameters:

- __HANDLE__: LTDC handle
- __INTERRUPT__: Specify the LTDC interrupt sources to be disabled. This parameter can be any combination of the following values:
  - LTDC_IT_LI: Line Interrupt flag
  - LTDC_IT_FU: FIFO Underrun Interrupt flag
  - LTDC_IT_TE: Transfer Error interrupt flag
  - LTDC_IT_RR: Register Reload Interrupt Flag
  - LTDC_IT_CRC: LTDC CRC error Interrupt
  - LTDC_IT_FUW: FIFO Underrun Warning Interrupt

Return value:

- None

__HAL_LTDC_GET_IT_SOURCE

Description:

- Check whether the specified LTDC interrupt has occurred or not.

Parameters:

- __HANDLE__: LTDC handle
- __INTERRUPT__: Specify the LTDC interrupt source to check. This parameter can be one of the following values:
  - LTDC_IT_LI: Line Interrupt flag
  - LTDC_IT_FU: FIFO Underrun Interrupt flag
  - LTDC_IT_TE: Transfer Error interrupt flag
  - LTDC_IT_RR: Register Reload Interrupt Flag
  - LTDC_IT_CRC: LTDC CRC error Interrupt
  - LTDC_IT_FUW: FIFO Underrun Warning Interrupt

Return value:

- The: state of INTERRUPT (SET or RESET).

LTDC Exported Types

MAX_LAYER

LTDC Flags

LTDC_FLAG_LI

LTDC Line Interrupt Flag

LTDC_FLAG_FU

LTDC FIFO Underrun interrupt Flag

LTDC_FLAG_TE

LTDC Transfer Error interrupt Flag

LTDC_FLAG_RR

LTDC Register Reload interrupt Flag

LTDC_FLAG_CRC

LTDC CRC error Interrupt

LTDC_FLAG_FUW

LTDC FIFO Warning Underrun Interrupt

LTDC HS POLARITY

LTDC_HSPOLARITY_AL

Horizontal Synchronization is active low.

LTDC_HSPOLARITY_AH

Horizontal Synchronization is active high.

LTDC Interrupts

LTDC_IT_LI

LTDC Line Interrupt

LTDC_IT_FU

LTDC FIFO Underrun Interrupt

LTDC_IT_TE

LTDC Transfer Error Interrupt

LTDC_IT_RR

LTDC Register Reload Interrupt

LTDC_IT_CRC

LTDC CRC error Interrupt

LTDC_IT_FUW

LTDC FIFO Warning Underrun Interrupt

LTDC Layer

LTDC_LAYER_1

LTDC Layer 1

LTDC_LAYER_2

LTDC Layer 2

LTDC LAYER Config

LTDC_STOPPOSITION

LTDC Layer stop position

LTDC_STARTPOSITION

LTDC Layer start position

LTDC_COLOR_FRAME_BUFFER

LTDC Layer Line length

LTDC_LINE_NUMBER

LTDC Layer Line number

LTDC luminance order from LSB

LTDC_YUV_LUMINANCE_ORDER_ODD_FIRST

Lumiance order Y1 then Y2

LTDC_YUV_LUMINANCE_ORDER_EVEN_FIRST

Luminance order Y2 then Y1

LTDC luminance range

LTDC_YUV_LUMINANCE_RESCALE_ENABLE

Rescale Luminance from [16-235] to [0-255]

LTDC_YUV_LUMINANCE_RESCALE_DISABLE

Luminance provided in range [0-255]

LTDC Mirror Configuration

LTDC_MIRROR_HORIZONTAL

Enable Horizontal Mirroring

LTDC_MIRROR_VERTICAL

Enable Vertical Mirroring

LTDC_MIRROR_HORIZONTAL_VERTICAL

Enable Horizontal and vertical Mirroring

LTDC_MIRROR_NONE

Disable Horizontal and vertical Mirroring

LTDC PC POLARITY

LTDC_PCPOLARITY_IPC

input pixel clock.

LTDC_PCPOLARITY_IIPC

inverted input pixel clock.

LTDC Pixel format

LTDC_PIXEL_FORMAT_ARGB8888

ARGB8888 LTDC pixel format

LTDC_PIXEL_FORMAT_ABGR8888

ABGR888 LTDC pixel format

LTDC_PIXEL_FORMAT_RGBA8888

RGBA888 LTDC pixel format

LTDC_PIXEL_FORMAT_BGRA8888

BGRA8888 LTDC pixel format

LTDC_PIXEL_FORMAT_RGB565

RGB565 LTDC pixel format

LTDC_PIXEL_FORMAT_BGR565

BGR565 LTDC pixel format

LTDC_PIXEL_FORMAT_RGB888

RGB888 LTDC pixel format

LTDC_PIXEL_FORMAT_ARGB1555

ARGB1555 LTDC pixel format

LTDC_PIXEL_FORMAT_ARGB4444

ARGB4444 LTDC pixel format

LTDC_PIXEL_FORMAT_L8

L8 LTDC pixel format

LTDC_PIXEL_FORMAT_AL44

AL44 LTDC pixel format

LTDC_PIXEL_FORMAT_AL88

AL88 LTDC pixel format

LTDC Reload Type

LTDC_RELOAD_IMMEDIATE

Immediate Reload

LTDC_RELOAD_VERTICAL_BLANKING

Vertical Blanking Reload

LTDC RGB component

LTDC_RGB_COMPONENT_RED

Red component

LTDC_RGB_COMPONENT_GREEN

Green component

LTDC_RGB_COMPONENT_BLUE

Blue component

LTDC_RGB_COMPONENT_ALL

Red, Blue and Green components

LTDC SYNC

LTDC_HORIZONTALSYNC

Horizontal synchronization width.

LTDC_VERTICALSYNC

Vertical synchronization height.

LTDC VS POLARITY

LTDC_VSPOLARITY_AL

Vertical Synchronization is active low.

LTDC_VSPOLARITY_AH

Vertical Synchronization is active high.

LTDC YUV to RGB color convertor

LTDC_YUV2RGBCONVERTOR_BT601_REDUCED_RANGE

Select ITU-R BT.601 as output format

LTDC_YUV2RGBCONVERTOR_BT709_REDUCED_RANGE

Select ITU-R BT.709 as output format

LTDC_YUV2RGBCONVERTOR_BT601_FULL_RANGE

Select RGB as output format

LTDC_YUV2RGBCONVERTOR_BT709_FULL_RANGE

Select RGB as output format

LTDC YUV order from LSB

LTDC_YUV_ORDER_LUMINANCE_FIRST

Luminance component first

LTDC_YUV_ORDER_CHROMINANCE_FIRST

Chrominance component (U or V) first

44 HAL MCE Generic Driver

44.1 MCE Firmware driver registers structures

44.1.1 MCE_RegionConfigTypeDef

MCE_RegionConfigTypeDef is defined in the stm32n6xx_hal_mce.h

Data Fields

- ***uint32_t ContextID***
- ***uint32_t StartAddress***
- ***uint32_t EndAddress***

Field Documentation

- ***uint32_t MCE_RegionConfigTypeDef::ContextID***
  MCE region context ID (can be null) This parameter is a value of ***MCE_Context_Index***.
- ***uint32_t MCE_RegionConfigTypeDef::StartAddress***
  MCE region start address
- ***uint32_t MCE_RegionConfigTypeDef::EndAddress***
  MCE region end address

44.1.2 MCE_AESConfigTypeDef

MCE_AESConfigTypeDef is defined in the stm32n6xx_hal_mce.h

Data Fields

- ***uint32_t Nonce***
- ***uint32_t Version***
- ***uint32_t * pKey***
- ***uint32_t KeySize***
- ***uint32_t Cipher_Mode***

Field Documentation

- ***uint32_t MCE_AESConfigTypeDef::Nonce[2]***
  MCE context nonce
- ***uint32_t MCE_AESConfigTypeDef::Version***
  16-bit long MCE context version
- ***uint32_t* MCE_AESConfigTypeDef::pKey***
  Pointer at the key used for encryption/decryption
- ***uint32_t MCE_AESConfigTypeDef::KeySize***
  This parameter can be MCE_AES_128 or MCE_AES_256
- ***uint32_t MCE_AESConfigTypeDef::Cipher_Mode***
  Authorized cipher mode

44.1.3 MCE_NoekeonConfigTypeDef

MCE_NoekeonConfigTypeDef is defined in the stm32n6xx_hal_mce.h

Data Fields

- ***uint32_t KeyType***
- ***uint32_t * pKey***

Field Documentation

- ***uint32_t MCE_NoekeonConfigTypeDef::KeyType***
  This parameter is a value of ***MCE_KeyType***.
- ***uint32_t* MCE_NoekeonConfigTypeDef::pKey***
  Pointer at the key used for encryption/decryption .

44.1.4 __MCE_HandleTypeDef

__MCE_HandleTypeDef is defined in the stm32n6xx_hal_mce.h

Data Fields

- ***MCE_TypeDef * Instance***
- ***HAL_MCE_StateTypeDef State***
- ***HAL_LockTypeDef Lock***
- ***__IO uint32_t ErrorCode***
- ***void(* ErrorCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***MCE_TypeDef* __MCE_HandleTypeDef::Instance***
  MCE registers base address
- ***HAL_MCE_StateTypeDef __MCE_HandleTypeDef::State***
  MCE state
- ***HAL_LockTypeDef __MCE_HandleTypeDef::Lock***
  MCE Locking object
- ***__IO uint32_t __MCE_HandleTypeDef::ErrorCode***
  MCE error code
- ***void(* __MCE_HandleTypeDef::ErrorCallback)(struct __MCE_HandleTypeDef *hmce)***
  MCE error callback
- ***void(* __MCE_HandleTypeDef::MspInitCallback)(struct __MCE_HandleTypeDef *hmce)***
  MCE Msp Init callback
- ***void(* __MCE_HandleTypeDef::MspDeInitCallback)(struct __MCE_HandleTypeDef *hmce)***
  MCE Msp DeInit callback

44.2 MCE Firmware driver defines

The following section lists the various define and macros of the module.

44.2.1 MCE

MCE

MCE Ciphering Algorithm

MCE_NO_CIPHER

MCE no cipher

MCE_STREAM_CIPHER

MCE stream cipher

MCE_BLOCK_CIPHER

MCE block cipher

MCE_FASTBLOCK_CIPHER

MCE fast block cipher

MCE Cipher Selection

MCE_AES_128

AES-128 cipher selected for all encrypted regions

MCE_AES_256

AES-256 cipher selected for all encrypted regions

MCE_NOEKEON

Noekeon cipher selected for all encrypted regions

MCE Configuration Attributes

MCE_ATTRIBUTE_NPRIV

Non-privileged access protection

MCE_ATTRIBUTE_PRIV

Privileged access protection

MCE Context Mode

MCE_CONTEXT_STREAM_CIPHER

Stream cipher is allowed with this cipher context

MCE_CONTEXT_BLOCK_CIPHER

Block cipher is allowed with this cipher context

MCE_CONTEXT_FASTBLOCK_CIPHER

Fast block cipher is allowed with this cipher context

MCE Context Index

MCE_NO_CONTEXT

MCE no context

MCE_CONTEXT1

MCE context 1

MCE_CONTEXT2

MCE context 2

MCE Error Definition

HAL_MCE_ERROR_NONE

No error

HAL_MCE_CONFIGURATION_ACCESS_ERROR

Configuration access error

HAL_MCE_ILLEGAL_ACCESS_READ_PRIV_ERROR

Illegal privileged data read or instruction fetch access error

HAL_MCE_ILLEGAL_ACCESS_READ_NPRIV_ERROR

Illegal unprivileged data read or instruction fetch access error

HAL_MCE_ILLEGAL_ACCESS_WRITE_PRIV_ERROR

Illegal privileged data write access error

HAL_MCE_ILLEGAL_ACCESS_WRITE_NPRIV_ERROR

Illegal un privileged data write access error

HAL_MCE_MASTER_KEY_ERROR

Master key error

HAL_MCE_FASTMASTER_KEY_ERROR

Fast master key error

HAL_MCE_CONTEXT_KEY_ERROR

Context key error

HAL_MCE_ERROR_INVALID_CALLBACK

Invalid Callback error

MCE Exported Macros

__HAL_MCE_RESET_HANDLE_STATE

Description:

- Reset MCE handle state.

Parameters:

- __HANDLE__: pointer to an MCE_HandleTypeDef structure that contains the configuration information for MCE module

Return value:

- None

__HAL_MCE_ENABLE_IT

Description:

- Enable MCE peripheral interrupts combination.

Parameters:

- __HANDLE__: pointer to an MCE_HandleTypeDef structure that contains the configuration information for MCE module
- __INTERRUPT__: mask on enabled interrupts This parameter can be one of the following values:
  - MCE_IT_ILLEGAL_ACCESS_ERROR MCE illegal access error interrupt

Return value:

- None

__HAL_MCE_DISABLE_IT

Description:

- Disable MCE peripheral interrupts combination.

Parameters:

- __HANDLE__: pointer to an MCE_HandleTypeDef structure that contains the configuration information for MCE module
- __INTERRUPT__: mask on disabled interrupts This parameter can be one of the following values:
  - MCE_IT_ILLEGAL_ACCESS_ERROR MCE illegal access error interrupt

Return value:

- None

__HAL_MCE_GET_FLAG

Description:

- Get MCE peripheral access error flag.

Parameters:

- __HANDLE__: pointer to an MCE_HandleTypeDef structure that contains the configuration information for MCE module
- __FLAG__: access error flag to check This parameter can be one of the following values:
  - MCE_ILLEGAL_ACCESS_READ_NPRIV MCE illegal access error flag

Return value:

- 0: (not set) or 1 (set)

__HAL_MCE_CLEAR_FLAG

Description:

- Clear MCE peripheral illegal/configuration access flag.

Parameters:

- __HANDLE__: pointer to an MCE_HandleTypeDef structure that contains the configuration information for MCE module
- __FLAG__: illegal access flag to check This parameter can be one of the following values:
  - MCE_ILLEGAL_ACCESS_READ_NPRIV MCE illegal access error flag

Return value:

- 0: (not set) or 1 (set)

MCE Illegal Access Flags

MCE_ILLEGAL_ACCESS_READ_NPRIV

Illegal unprivileged data read/instruction fetch access flag

MCE Interrupts

MCE_IT_ILLEGAL_ACCESS_ERROR

Illegal access error interrupt

key type used for encryption, Master key or Fast Master key

MCE_USE_MASTERKEYS

Master keys used for encryption

MCE_USE_FASTMASTERKEYS

Fast Master keys used for encryption

MCE Lock values

MCE_LOCK_OFF

No global lock set

MCE_LOCK_ON

Global lock set

MCE_MASTERKEYS_LOCK_OFF

No master keys lock set

MCE_MASTERKEYS_LOCK_ON

Master keys lock set

MCE Regions Index

MCE_REGION1

MCE region 1

MCE_REGION2

MCE region 2

MCE_REGION3

MCE region 3

MCE_REGION4

MCE region 4

MCE Region Privilege

MCE_REGION_READONLY

Writes to region are ignored, reads are allowed

MCE_REGION_READWRITE

Region can be read and written

45 HAL MDF Generic Driver

45.1 MDF Firmware driver registers structures

45.1.1 MDF_ClockTriggerTypeDef

MDF_ClockTriggerTypeDef is defined in the stm32n6xx_hal_mdf.h

Data Fields

- ***FunctionalState Activation***
- ***uint32_t Source***
- ***uint32_t Edge***

Field Documentation

- ***FunctionalState MDF_ClockTriggerTypeDef::Activation***
  Output clock trigger enable/disable
- ***uint32_t MDF_ClockTriggerTypeDef::Source***
  Output clock trigger source. This parameter can be a value of ***MDF_ClockTriggerSource***
- ***uint32_t MDF_ClockTriggerTypeDef::Edge***
  Output clock trigger edge. This parameter can be a value of ***MDF_ClockTriggerEdge***

45.1.2 MDF_OutputClockTypeDef

MDF_OutputClockTypeDef is defined in the stm32n6xx_hal_mdf.h

Data Fields

- ***FunctionalState Activation***
- ***uint32_t Pins***
- ***uint32_t Divider***
- ***MDF_ClockTriggerTypeDef Trigger***

Field Documentation

- ***FunctionalState MDF_OutputClockTypeDef::Activation***
  Output clock enable/disable
- ***uint32_t MDF_OutputClockTypeDef::Pins***
  Output clock pins. This parameter can be a value of ***MDF_OuputClockPins***
- ***uint32_t MDF_OutputClockTypeDef::Divider***
  Output clock divider. This parameter must be a number between Min_Data = 1 and Max_Data = 16
- ***MDF_ClockTriggerTypeDef MDF_OutputClockTypeDef::Trigger***
  Output clock trigger parameters

45.1.3 MDF_CommonParamTypeDef

MDF_CommonParamTypeDef is defined in the stm32n6xx_hal_mdf.h

Data Fields

- ***uint32_t InterleavedFilters***
- ***uint32_t ProcClockDivider***
- ***MDF_OutputClockTypeDef OutputClock***

Field Documentation

- ***uint32_t MDF_CommonParamTypeDef::InterleavedFilters***
  Number of filters in interleaved mode with filter 0. This parameter must be a number between Min_Data = 0 and Max_Data = 5.
  Note:
  – This parameter is not used for ADF instance
- ***uint32_t MDF_CommonParamTypeDef::ProcClockDivider***
  Processing clock divider. This parameter must be a number between Min_Data = 1 and Max_Data = 128

- ***MDF_OutputClockTypeDef MDF_CommonParamTypeDef::OutputClock***
  Output clock parameters

45.1.4 MDF_SerialInterfaceTypeDef

MDF_SerialInterfaceTypeDef is defined in the stm32n6xx_hal_mdf.h

Data Fields

- ***FunctionalState Activation***
- ***uint32_t Mode***
- ***uint32_t ClockSource***
- ***uint32_t Threshold***

Field Documentation

- ***FunctionalState MDF_SerialInterfaceTypeDef::Activation***
  Serial interface enable/disable
- ***uint32_t MDF_SerialInterfaceTypeDef::Mode***
  Serial interface mode. This parameter can be a value of ***MDF_SitfMode***
- ***uint32_t MDF_SerialInterfaceTypeDef::ClockSource***
  Serial interface clock source. This parameter can be a value of ***MDF_SitfClockSource***
- ***uint32_t MDF_SerialInterfaceTypeDef::Threshold***
  SPI threshold for clock absence detection or Manchester symbol threshold. This parameter must be a number between Min_Data = 4 and Max_Data = 31

45.1.5 MDF_InitTypeDef

MDF_InitTypeDef is defined in the stm32n6xx_hal_mdf.h

Data Fields

- ***MDF_CommonParamTypeDef CommonParam***
- ***MDF_SerialInterfaceTypeDef SerialInterface***
- ***uint32_t FilterBistream***

Field Documentation

- ***MDF_CommonParamTypeDef MDF_InitTypeDef::CommonParam***
  MDF common parameters
- ***MDF_SerialInterfaceTypeDef MDF_InitTypeDef::SerialInterface***
  MDF serial interface parameters
- ***uint32_t MDF_InitTypeDef::FilterBistream***
  MDF filter bitstream selection. This parameter can be a value of ***MDF_FilterBitstream***

45.1.6 __MDF_HandleTypeDef

__MDF_HandleTypeDef is defined in the stm32n6xx_hal_mdf.h

Data Fields

- ***MDF_Filter_TypeDef * Instance***
- ***MDF_InitTypeDef Init***
- ***DMA_HandleTypeDef * hdma***
- ***__IO HAL_MDF_StateTypeDef State***
- ***__IO uint32_t ErrorCode***
- ***void(* OldCallback***
- ***void(* AcqCpltCallback***
- ***void(* AcqHalfCpltCallback***
- ***void(* SndLvCallback***
- ***void(* SadCallback***
- ***void(* ErrorCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***MDF_Filter_TypeDef* __MDF_HandleTypeDef::Instance***
  MDF instance
- ***MDF_InitTypeDef __MDF_HandleTypeDef::Init***
  MDF init parameters
- ***DMA_HandleTypeDef* __MDF_HandleTypeDef::hdma***
  Pointer on DMA handler for acquisitions
- ***__IO HAL_MDF_StateTypeDef __MDF_HandleTypeDef::State***
  MDF state
- ***__IO uint32_t __MDF_HandleTypeDef::ErrorCode***
  MDF error code
- ***void(* __MDF_HandleTypeDef::OldCallback)(struct __MDF_HandleTypeDef *hmdf, uint32_t Threshold)***
  MDF out-off limit detector callback.
  Note:
  – Not used for ADF instance
- ***void(* __MDF_HandleTypeDef::AcqCpltCallback)(struct __MDF_HandleTypeDef *hmdf)***
  MDF acquisition complete callback
- ***void(* __MDF_HandleTypeDef::AcqHalfCpltCallback)(struct __MDF_HandleTypeDef *hmdf)***
  MDF acquisition half complete callback
- ***void(* __MDF_HandleTypeDef::SndLvCallback)(struct __MDF_HandleTypeDef *hmdf, uint32_t SoundLevel, uint32_t AmbientNoise)***
  MDF sound level callback.
  Note:
  – Not used for MDF instance
- ***void(* __MDF_HandleTypeDef::SadCallback)(struct __MDF_HandleTypeDef *hmdf)***
  MDF sound activity detector callback.
  Note:
  – Not used for MDF instance
- ***void(* __MDF_HandleTypeDef::ErrorCallback)(struct __MDF_HandleTypeDef *hmdf)***
  MDF error callback
- ***void(* __MDF_HandleTypeDef::MspInitCallback)(struct __MDF_HandleTypeDef *hmdf)***
  MDF MSP init callback
- ***void(* __MDF_HandleTypeDef::MspDeInitCallback)(struct __MDF_HandleTypeDef *hmdf)***
  MDF MSP de-init callback

45.1.7 MDF_ReshapeFilterTypeDef

MDF_ReshapeFilterTypeDef is defined in the stm32n6xx_hal_mdf.h

Data Fields

- ***FunctionalState Activation***
- ***uint32_t DecimationRatio***

Field Documentation

- ***FunctionalState MDF_ReshapeFilterTypeDef::Activation***
  Reshape filter enable/disable
- ***uint32_t MDF_ReshapeFilterTypeDef::DecimationRatio***
  Reshape filter decimation ratio. This parameter can be a value of ***MDF_ReshapeDecimationRatio***

45.1.8 MDF_HighPassFilterTypeDef

MDF_HighPassFilterTypeDef is defined in the stm32n6xx_hal_mdf.h

Data Fields

- ***FunctionalState Activation***
- ***uint32_t CutOffFrequency***

Field Documentation

- ***FunctionalState MDF_HighPassFilterTypeDef::Activation***
  High pass filter enable/disable

- ***uint32_t MDF_HighPassFilterTypeDef::CutOffFrequency***
  High pass filter cut-off frequency. This parameter can be a value of ***MDF_HighPassCutOffFreq***

45.1.9 MDF_IntegratorTypeDef

MDF_IntegratorTypeDef is defined in the stm32n6xx_hal_mdf.h

Data Fields

- ***FunctionalState Activation***
- ***uint32_t Value***
- ***uint32_t OutputDivision***

Field Documentation

- ***FunctionalState MDF_IntegratorTypeDef::Activation***
  Integrator enable/disable
- ***uint32_t MDF_IntegratorTypeDef::Value***
  Integrator value. This parameter must be a number between Min_Data = 2 and Max_Data = 128
- ***uint32_t MDF_IntegratorTypeDef::OutputDivision***
  Integrator output division. This parameter can be a value of ***MDF_IntegratorOutputDiv***

45.1.10 MDF_SoundActivityTypeDef

MDF_SoundActivityTypeDef is defined in the stm32n6xx_hal_mdf.h

Data Fields

- ***FunctionalState Activation***
- ***uint32_t Mode***
- ***uint32_t FrameSize***
- ***FunctionalState Hysteresis***
- ***uint32_t SoundTriggerEvent***
- ***uint32_t DataMemoryTransfer***
- ***uint32_t MinNoiseLevel***
- ***uint32_t HangoverWindow***
- ***uint32_t LearningFrames***
- ***uint32_t AmbientNoiseSlope***
- ***uint32_t SignalNoiseThreshold***
- ***FunctionalState SoundLevelInterrupt***

Field Documentation

- ***FunctionalState MDF_SoundActivityTypeDef::Activation***
  Sound activity detector enable/disable
- ***uint32_t MDF_SoundActivityTypeDef::Mode***
  Sound activity detector mode. This parameter can be a value of ***MDF_SadMode***
- ***uint32_t MDF_SoundActivityTypeDef::FrameSize***
  Size of one frame to compute short-term signal level. This parameter can be a value of ***MDF_SadFrameSize***
- ***FunctionalState MDF_SoundActivityTypeDef::Hysteresis***
  Hysteresis enable/disable.
  Note:
  – This parameter is not used if Mode is set to MDF_SAD_AMBIENT_NOISE_ESTIMATOR
- ***uint32_t MDF_SoundActivityTypeDef::SoundTriggerEvent***
  Sound trigger event configuration. This parameter can be a value of ***MDF_SadSoundTriggerEvent***
- ***uint32_t MDF_SoundActivityTypeDef::DataMemoryTransfer***
  Data memory transfer mode. This parameter can be a value of ***MDF_SadDataMemoryTransfer***
- ***uint32_t MDF_SoundActivityTypeDef::MinNoiseLevel***
  Minimum noise level. This parameter must be a number between Min_Data = 0 and Max_Data = 8191
- ***uint32_t MDF_SoundActivityTypeDef::HangoverWindow***
  Hangover time window in frames. This parameter can be a value of ***MDF_SadHangoverWindow***

- ***uint32_t MDF_SoundActivityTypeDef::LearningFrames***
  Number of learning frames for the first estimation of noise level. This parameter can be a value of ***MDF_SadLearningFrames***
- ***uint32_t MDF_SoundActivityTypeDef::AmbientNoiseSlope***
  Ambient noise slope control. This parameter must be a number between Min_Data = 0 and Max_Data = 7.
  Note:
  – This parameter is not used if Mode is set to MDF_SAD_SOUND_DETECTOR
- ***uint32_t MDF_SoundActivityTypeDef::SignalNoiseThreshold***
  Signal to noise threshold. This parameter can be a value of ***MDF_SadSignalNoiseThreshold***
- ***FunctionalState MDF_SoundActivityTypeDef::SoundLevelInterrupt***
  Sound level interrupt enable/disable.
  Note:
  – This interrupt is mainly used for debug purpose

45.1.11 MDF_FilterTriggerTypeDef

MDF_FilterTriggerTypeDef is defined in the stm32n6xx_hal_mdf.h

Data Fields

- ***uint32_t Source***
- ***uint32_t Edge***

Field Documentation

- ***uint32_t MDF_FilterTriggerTypeDef::Source***
  Filter trigger source. This parameter can be a value of ***MDF_FilterTriggerSource***
- ***uint32_t MDF_FilterTriggerTypeDef::Edge***
  Filter trigger edge. This parameter can be a value of ***MDF_FilterTriggerEdge***

45.1.12 MDF_FilterConfigTypeDef

MDF_FilterConfigTypeDef is defined in the stm32n6xx_hal_mdf.h

Data Fields

- ***uint32_t DataSource***
- ***uint32_t Delay***
- ***uint32_t CicMode***
- ***uint32_t DecimationRatio***
- ***int32_t Offset***
- ***int32_t Gain***
- ***MDF_ReshapeFilterTypeDef ReshapeFilter***
- ***MDF_HighPassFilterTypeDef HighPassFilter***
- ***MDF_IntegratorTypeDef Integrator***
- ***MDF_SoundActivityTypeDef SoundActivity***
- ***uint32_t AcquisitionMode***
- ***uint32_t FifoThreshold***
- ***uint32_t DiscardSamples***
- ***MDF_FilterTriggerTypeDef Trigger***
- ***uint32_t SnapshotFormat***

Field Documentation

- ***uint32_t MDF_FilterConfigTypeDef::DataSource***
  Filter data source. This parameter can be a value of ***MDF_DataSource***
- ***uint32_t MDF_FilterConfigTypeDef::Delay***
  Delay to apply on data source in number of samples. This parameter must be a number between Min_Data = 0 and Max_Data = 127
- ***uint32_t MDF_FilterConfigTypeDef::CicMode***
  CIC filter mode. This parameter can be a value of ***MDF_CicMode***

- ***uint32_t MDF_FilterConfigTypeDef::DecimationRatio***
  Filter decimation ratio. This parameter must be a number between Min_Data = 2 and Max_Data = 512
- ***int32_t MDF_FilterConfigTypeDef::Offset***
  Filter offset error compensation. This parameter must be a number between Min_Data = -33554432 and Max_Data = 33554431.
  Note:
  – This parameter is not used for ADF instance
- ***int32_t MDF_FilterConfigTypeDef::Gain***
  Filter gain in step of around 3db (from -48db to 72dB). This parameter must be a number between Min_Data = -16 and Max_Data = 24
- ***MDF_ReshapeFilterTypeDef MDF_FilterConfigTypeDef::ReshapeFilter***
  Reshape filter configuration
- ***MDF_HighPassFilterTypeDef MDF_FilterConfigTypeDef::HighPassFilter***
  High pass filter configuration
- ***MDF_IntegratorTypeDef MDF_FilterConfigTypeDef::Integrator***
  Integrator configuration.
  Note:
  – This parameter is not used for ADF instance
- ***MDF_SoundActivityTypeDef MDF_FilterConfigTypeDef::SoundActivity***
  Sound activity detector configuration.
  Note:
  – This parameter is not used for MDF instance
- ***uint32_t MDF_FilterConfigTypeDef::AcquisitionMode***
  Filter acquisition mode. This parameter can be a value of ***MDF_AcquisitionMode***
- ***uint32_t MDF_FilterConfigTypeDef::FifoThreshold***
  Filter RXFIFO threshold. This parameter can be a value of ***MDF_FifoThreshold***
- ***uint32_t MDF_FilterConfigTypeDef::DiscardSamples***
  Number of samples to discard after filter enable. This parameter must be a number between Min_Data = 0 and Max_Data = 255
- ***MDF_FilterTriggerTypeDef MDF_FilterConfigTypeDef::Trigger***
  Filter trigger configuration.
  Note:
  – This parameter is not used if AcquisitionMode is set to MDF_MODE_ASYNC_CONT or MDF_MODE_ASYNC_SINGLE
- ***uint32_t MDF_FilterConfigTypeDef::SnapshotFormat***
  Snapshot format. This parameter can be a value of ***MDF_SnapshotFormat***.
  Note:
  – This parameter is used only if AcquisitionMode is set to MDF_MODE_SYNC_SNAPSHOT and for MDF instance

45.1.13 MDF_SnapshotParamTypeDef

MDF_SnapshotParamTypeDef is defined in the stm32n6xx_hal_mdf.h

Data Fields

- ***int32_t Value***
- ***uint32_t DecimationCounter***
- ***uint32_t IntegratorCounter***

Field Documentation

- ***int32_t MDF_SnapshotParamTypeDef::Value***
  Snapshot acquisition value on 16 or 23 MSB depending on snapshot format
- ***uint32_t MDF_SnapshotParamTypeDef::DecimationCounter***
  Decimation counter value when snapshot trigger occurs
- ***uint32_t MDF_SnapshotParamTypeDef::IntegratorCounter***
  Integrator counter value when snapshot trigger occurs. This value is available only if snapshot format is 16 bits resolution

45.1.14 MDF_DmaConfigTypeDef

MDF_DmaConfigTypeDef is defined in the stm32n6xx_hal_mdf.h

Data Fields

- ***uint32_t Address***
- ***uint32_t DataLength***
- ***FunctionalState MsbOnly***

Field Documentation

- ***uint32_t MDF_DmaConfigTypeDef::Address***
  DMA destination address
- ***uint32_t MDF_DmaConfigTypeDef::DataLength***
  Length of data to transfer in bytes
- ***FunctionalState MDF_DmaConfigTypeDef::MsbOnly***
  Transfer only the 16MSB of the acquistion data

45.1.15 MDF_ScdConfigTypeDef

MDF_ScdConfigTypeDef is defined in the stm32n6xx_hal_mdf.h

Data Fields

- ***uint32_t Threshold***
- ***uint32_t BreakSignal***

Field Documentation

- ***uint32_t MDF_ScdConfigTypeDef::Threshold***
  Short-circuit detector threshold. This parameter must be a number between Min_Data = 2 and Max_Data = 256
- ***uint32_t MDF_ScdConfigTypeDef::BreakSignal***
  Break signal assignment for short-circuit detector. This parameter can be a values combination of ***MDF_BreakSignals***

45.1.16 MDF_OldConfigTypeDef

MDF_OldConfigTypeDef is defined in the stm32n6xx_hal_mdf.h

Data Fields

- ***uint32_t OldCicMode***
- ***uint32_t OldDecimationRatio***
- ***int32_t HighThreshold***
- ***int32_t LowThreshold***
- ***uint32_t OldEventConfig***
- ***uint32_t BreakSignal***

Field Documentation

- ***uint32_t MDF_OldConfigTypeDef::OldCicMode***
  Out-off limit detector CIC filter mode. This parameter can be a value of ***MDF_OldCicMode***
- ***uint32_t MDF_OldConfigTypeDef::OldDecimationRatio***
  Out-off limit detector decimation ratio. This parameter must be a number between Min_Data = 1 and Max_Data = 32
- ***int32_t MDF_OldConfigTypeDef::HighThreshold***
  Out-off limit detector high threshold. This parameter must be a number between Min_Data = -33554432 and Max_Data = 33554431
- ***int32_t MDF_OldConfigTypeDef::LowThreshold***
  Out-off limit detector low threshold. This parameter must be a number between Min_Data = -33554432 and Max_Data = 33554431
- ***uint32_t MDF_OldConfigTypeDef::OldEventConfig***
  Out-off limit event configuration. This parameter can be a value of ***MDF_OldEventConfig***
- ***uint32_t MDF_OldConfigTypeDef::BreakSignal***
  Break signal assignment for out-off limit detector. This parameter can be a values combination of ***MDF_BreakSignals***

45.2 MDF Firmware driver API description

The following section lists the various functions of the MDF library.

45.2.1 How to use this driver

Initialization and de-initialization

1. User has first to initialize MDF or ADF instance.
2. As prerequisite, fill in the HAL_MDF_MspInit() :
   – Enable MDFz or ADFz clock interface with __HAL_RCC_MDFz_CLK_ENABLE() or __HAL_RCC_ADFz_CLK_ENABLE().
   – Enable the clocks for the used GPIOS with __HAL_RCC_GPIOx_CLK_ENABLE().
   – Configure these pins in alternate mode using HAL_GPIO_Init().
   – If interrupt mode is used, enable and configure MDFz_FLTx or ADFz interrupt with HAL_NVIC_SetPriority() and HAL_NVIC_EnableIRQ().
   – If DMA mode is used, initialize and configure DMA.
3. Configure the common parameters (only for first MDF or ADF instance init), serial interface parameters and filter bitstream selection by calling HAL_MDF_Init() function.
1. User can de-initialize MDF or ADF instance with HAL_MDF_DeInit() function.

Acquisition

1. Configure filter parameters and start acquisition using HAL_MDF_AcqStart(), HAL_MDF_AcqStart_IT() or HAL_MDF_AcqStart_DMA().
2. In polling mode :
   – Use HAL_MDF_PollForAcq() to detect the end of acquisition. Use HAL_MDF_GetAcqValue to get acquisition value.
   – Only for MDF instance, use HAL_MDF_PollForSnapshotAcq() to detect the end of snapshot acquisition. Use HAL_MDF_GetSnapshotAcqValue to get snapshot acquisition value.
   – Only for ADF instance, use HAL_MDF_PollForSndLvl() to detect and get new sound level value and ambient noise value.
   – Only for ADF instance, use HAL_MDF_PollForSad() to detect sound activity.
3. In interrupt mode :
   – HAL_MDF_AcqCpltCallback() will be called at the end of acquisition. Use HAL_MDF_GetAcqValue to get acquisition value or use HAL_MDF_GetSnapshotAcqValue to get snapshot acquisition value.
   – Only for ADF instance, HAL_MDF_SndLvlCallback() will be called when new sound level and ambient noise values are available.
   – Only for ADF instance, HAL_MDF_SadCallback() will be called when sound activity detection occurs.
   – HAL_MDF_ErrorCallback() will be called if overflow, filter overrun or saturation occurs. Use HAL_MDF_GetErrorCode() to get the corresponding error.
4. In DMA mode :
   – HAL_MDF_AcqHalfCpltCallback() and HAL_MDF_AcqCpltCallback() will be called respectively at the half acquisition and at the acquisition complete.
   – Only for ADF instance, HAL_MDF_SndLvlCallback() will be called when new sound level and ambient noise values are available.
   – Only for ADF instance, HAL_MDF_SadCallback() will be called when sound activity detection occurs.
   – HAL_MDF_ErrorCallback() will be called if overflow, filter overrun, saturation or DMA error occurs. Use HAL_MDF_GetErrorCode() to get the corresponding error.
5. Use HAL_MDF_GenerateTrgo() to generate pulse on TRGO signal.
6. During acquisition, use HAL_MDF_SetDelay() and HAL_MDF_GetDelay() to respectively set and get the delay on data source.
7. During acquisition, use HAL_MDF_SetGain() and HAL_MDF_GetGain() to respectively set and get the filter gain.

8. During acquisition, use HAL_MDF_SetOffset() and HAL_MDF_GetOffset() to respectively set and get the filter offset error compensation.
9. During acquisition, only for MDF instance, use HAL_MDF_SetOffset() and HAL_MDF_GetOffset() to respectively set and get the filter offset error compensation.
10. Stop acquisition using HAL_MDF_AcqStop(), HAL_MDF_AcqStop_IT() or HAL_MDF_AcqStop_DMA().

Clock absence detection

1. Clock absence detection is always enabled so no need to start clock absence detection in polling mode. Use HAL_MDF_CkabStart_IT() to start clock absence detection in interrupt mode.
2. In polling mode, use HAL_MDF_PollForCkab() to detect the clock absence.
3. In interrupt mode, HAL_MDF_ErrorCallback() will be called if clock absence detection occurs. Use HAL_MDF_GetErrorCode() to get the corresponding error.
4. Stop clock absence detection in interrupt mode using HAL_MDF_CkabStop_IT().

Short circuit detection

1. Only for MDF instance, start short circuit detection using HAL_MDF_ScdStart() or HAL_MDF_ScdStart_IT().
2. In polling mode, use HAL_MDF_PollForScd() to detect short circuit.
3. In interrupt mode, HAL_MDF_ErrorCallback() will be called if short circuit detection occurs. Use HAL_MDF_GetErrorCode() to get the corresponding error.
4. Stop short circuit detection using HAL_MDF_ScdStop() or HAL_MDF_ScdStop_IT().

Out-off limit detection

1. Only for MDF instance, start out-off limit detection using HAL_MDF_OldStart() or HAL_MDF_OldStart_IT().
2. In polling mode, use HAL_MDF_PollForOld() to detect out-off limit and to get threshold information.
3. In interrupt mode, HAL_MDF_OldCallback() will be called if out-off limit detection occurs.
4. Stop out-off limit detection using HAL_MDF_OldStop() or HAL_MDF_OldStop_IT().

generic functions

1. HAL_MDF_IRQHandler will be called when MDF or ADF interrupt occurs.
2. HAL_MDF_ErrorCallback will be called when MDF or ADF error occurs.
3. Use HAL_MDF_GetState() to get the current MDF or ADF instance state.
4. Use HAL_MDF_GetErrorCode() to get the current MDF or ADF instance error code.

Callback registration

The compilation define USE_HAL_MDF_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use functions HAL_MDF_RegisterCallback(), HAL_MDF_RegisterOldCallback() or HAL_MDF_RegisterSndLvlCallback() to register a user callback.

Function HAL_MDF_RegisterCallback() allows to register following callbacks :

- AcqCpltCallback : Acquisition complete callback.
- AcqHalfCpltCallback : Acquisition half complete callback.
- SadCallback : Sound activity detection callback (only for ADF instance).
- ErrorCallback : Error callback.
- MspInitCallback : MSP init callback.
- MspDeInitCallback : MSP de-init callback.

This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

For MDF instance and for specific out-off limit detection callback use dedicated register callback : HAL_MDF_RegisterOldCallback().

For ADF instance and for specific sound level callback use dedicated register callback : HAL_MDF_RegisterSndLvlCallback().

Use function HAL_MDF_UnRegisterCallback() to reset a callback to the default weak function.

HAL_MDF_UnRegisterCallback() takes as parameters the HAL peripheral handle and the Callback ID.

This function allows to reset following callbacks :

- AcqCpltCallback : Acquisition complete callback.
- AcqHalfCpltCallback : Acquisition half complete callback.
- SadCallback : Sound activity detection callback (only for ADF instance).
- ErrorCallback : Error callback.
- MspInitCallback : MSP init callback.
- MspDeInitCallback : MSP de-init callback.

For MDF instance and for specific out-off limit detection callback use dedicated unregister callback : HAL_MDF_UnRegisterOldCallback().

For ADF instance and for specific sound level callback use dedicated unregister callback : HAL_MDF_UnRegisterSndLvlCallback().

By default, after the call of init function and if the state is RESET all callbacks are reset to the corresponding legacy weak functions : examples HAL_MDF_AcqCpltCallback(), HAL_MDF_ErrorCallback(). Exception done for MspInit and MspDeInit callbacks that are respectively reset to the legacy weak functions in the init and de-init only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the init and de-init keep and use the user MspInit/MspDeInit callbacks (registered beforehand).

Callbacks can be registered/unregistered in READY state only. Exception done for MspInit/MspDeInit callbacks that can be registered/unregistered in READY or RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the init/de-init. In that case first register the MspInit/MspDeInit user callbacks using HAL_MDF_RegisterCallback() before calling init or de-init function.

When the compilation define USE_HAL_MDF_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak callbacks are used.

45.2.2 Detailed description of functions

HAL_MDF_Init

Function name

HAL_StatusTypeDef HAL_MDF_Init (MDF_HandleTypeDef * hmdf)

Function description

HAL_MDF_DeInit

Function name

HAL_StatusTypeDef HAL_MDF_DeInit (MDF_HandleTypeDef * hmdf)

Function description

HAL_MDF_MspInit

Function name

void HAL_MDF_MspInit (MDF_HandleTypeDef * hmdf)

Function description

HAL_MDF_MspDeInit

Function name

void HAL_MDF_MspDeInit (MDF_HandleTypeDef * hmdf)

Function description

HAL_MDF_RegisterCallback

Function name

HAL_StatusTypeDef HAL_MDF_RegisterCallback (MDF_HandleTypeDef * hmdf, HAL_MDF_CallbackIDTypeDef CallbackID, pMDF_CallbackTypeDef pCallback)

Function description

HAL_MDF_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_MDF_UnRegisterCallback (MDF_HandleTypeDef * hmdf, HAL_MDF_CallbackIDTypeDef CallbackID)

Function description

HAL_MDF_RegisterOldCallback

Function name

HAL_StatusTypeDef HAL_MDF_RegisterOldCallback (MDF_HandleTypeDef * hmdf, pMDF_OldCallbackTypeDef pCallback)

Function description

HAL_MDF_UnRegisterOldCallback

Function name

HAL_StatusTypeDef HAL_MDF_UnRegisterOldCallback (MDF_HandleTypeDef * hmdf)

Function description

HAL_MDF_RegisterSndLvlCallback

Function name

HAL_StatusTypeDef HAL_MDF_RegisterSndLvlCallback (MDF_HandleTypeDef * hmdf, pMDF_SndLvlCallbackTypeDef pCallback)

Function description

HAL_MDF_UnRegisterSndLvlCallback

Function name

HAL_StatusTypeDef HAL_MDF_UnRegisterSndLvlCallback (MDF_HandleTypeDef * hmdf)

Function description

HAL_MDF_AcqStart

Function name

HAL_StatusTypeDef HAL_MDF_AcqStart (MDF_HandleTypeDef * hmdf, const MDF_FilterConfigTypeDef * pFilterConfig)

Function description

HAL_MDF_PollForAcq

Function name

HAL_StatusTypeDef HAL_MDF_PollForAcq (MDF_HandleTypeDef * hmdf, uint32_t Timeout)

Function description

HAL_MDF_PollForSnapshotAcq

Function name

HAL_StatusTypeDef HAL_MDF_PollForSnapshotAcq (MDF_HandleTypeDef * hmdf, uint32_t Timeout)

Function description

HAL_MDF_GetAcqValue

Function name

HAL_StatusTypeDef HAL_MDF_GetAcqValue (const MDF_HandleTypeDef * hmdf, int32_t * pValue)

Function description

HAL_MDF_GetSnapshotAcqValue

Function name

HAL_StatusTypeDef HAL_MDF_GetSnapshotAcqValue (MDF_HandleTypeDef * hmdf, MDF_SnapshotParamTypeDef * pSnapshotParam)

Function description

HAL_MDF_AcqStop

Function name

HAL_StatusTypeDef HAL_MDF_AcqStop (MDF_HandleTypeDef * hmdf)

Function description

HAL_MDF_AcqStart_IT

Function name

HAL_StatusTypeDef HAL_MDF_AcqStart_IT (MDF_HandleTypeDef * hmdf, const MDF_FilterConfigTypeDef * pFilterConfig)

Function description

HAL_MDF_AcqStop_IT

Function name

HAL_StatusTypeDef HAL_MDF_AcqStop_IT (MDF_HandleTypeDef * hmdf)

Function description

HAL_MDF_AcqStart_DMA

Function name

HAL_StatusTypeDef HAL_MDF_AcqStart_DMA (MDF_HandleTypeDef * hmdf, const MDF_FilterConfigTypeDef * pFilterConfig, const MDF_DmaConfigTypeDef * pDmaConfig)

Function description

HAL_MDF_AcqStop_DMA

Function name

HAL_StatusTypeDef HAL_MDF_AcqStop_DMA (MDF_HandleTypeDef * hmdf)

Function description

HAL_MDF_GenerateTrgo

Function name

HAL_StatusTypeDef HAL_MDF_GenerateTrgo (const MDF_HandleTypeDef * hmdf)

Function description

HAL_MDF_SetDelay

Function name

HAL_StatusTypeDef HAL_MDF_SetDelay (MDF_HandleTypeDef * hmdf, uint32_t Delay)

Function description

HAL_MDF_GetDelay

Function name

HAL_StatusTypeDef HAL_MDF_GetDelay (const MDF_HandleTypeDef * hmdf, uint32_t * pDelay)

Function description

HAL_MDF_SetGain

Function name

HAL_StatusTypeDef HAL_MDF_SetGain (MDF_HandleTypeDef * hmdf, int32_t Gain)

Function description

HAL_MDF_GetGain

Function name

HAL_StatusTypeDef HAL_MDF_GetGain (const MDF_HandleTypeDef * hmdf, int32_t * pGain)

Function description

HAL_MDF_SetOffset

Function name

HAL_StatusTypeDef HAL_MDF_SetOffset (MDF_HandleTypeDef * hmdf, int32_t Offset)

Function description

HAL_MDF_GetOffset

Function name

HAL_StatusTypeDef HAL_MDF_GetOffset (const MDF_HandleTypeDef * hmdf, int32_t * pOffset)

Function description

HAL_MDF_PollForSndLvl

Function name

HAL_StatusTypeDef HAL_MDF_PollForSndLvl (MDF_HandleTypeDef * hmdf, uint32_t Timeout, uint32_t * pSoundLevel, uint32_t * pAmbientNoise)

Function description

HAL_MDF_PollForSad

Function name

HAL_StatusTypeDef HAL_MDF_PollForSad (MDF_HandleTypeDef * hmdf, uint32_t Timeout)

Function description

HAL_MDF_AcqCpltCallback

Function name

void HAL_MDF_AcqCpltCallback (MDF_HandleTypeDef * hmdf)

Function description

HAL_MDF_AcqHalfCpltCallback

Function name

void HAL_MDF_AcqHalfCpltCallback (MDF_HandleTypeDef * hmdf)

Function description

HAL_MDF_SndLvlCallback

Function name

void HAL_MDF_SndLvlCallback (MDF_HandleTypeDef * hmdf, uint32_t SoundLevel, uint32_t AmbientNoise)

Function description

HAL_MDF_SadCallback

Function name

void HAL_MDF_SadCallback (MDF_HandleTypeDef * hmdf)

Function description

HAL_MDF_PollForCkab

Function name

HAL_StatusTypeDef HAL_MDF_PollForCkab (MDF_HandleTypeDef * hmdf, uint32_t Timeout)

Function description

HAL_MDF_CkabStart_IT

Function name

HAL_StatusTypeDef HAL_MDF_CkabStart_IT (MDF_HandleTypeDef * hmdf)

Function description

HAL_MDF_CkabStop_IT

Function name

HAL_StatusTypeDef HAL_MDF_CkabStop_IT (MDF_HandleTypeDef * hmdf)

Function description

HAL_MDF_ScdStart

Function name

HAL_StatusTypeDef HAL_MDF_ScdStart (MDF_HandleTypeDef * hmdf, const MDF_ScdConfigTypeDef * pScdConfig)

Function description

HAL_MDF_PollForScd

Function name

HAL_StatusTypeDef HAL_MDF_PollForScd (MDF_HandleTypeDef * hmdf, uint32_t Timeout)

Function description

HAL_MDF_ScdStop

Function name

HAL_StatusTypeDef HAL_MDF_ScdStop (MDF_HandleTypeDef * hmdf)

Function description

HAL_MDF_ScdStart_IT

Function name

HAL_StatusTypeDef HAL_MDF_ScdStart_IT (MDF_HandleTypeDef * hmdf, const MDF_ScdConfigTypeDef * pScdConfig)

Function description

HAL_MDF_ScdStop_IT

Function name

HAL_StatusTypeDef HAL_MDF_ScdStop_IT (MDF_HandleTypeDef * hmdf)

Function description

HAL_MDF_OldStart

Function name

HAL_StatusTypeDef HAL_MDF_OldStart (MDF_HandleTypeDef * hmdf, const MDF_OldConfigTypeDef * pOldConfig)

Function description

HAL_MDF_PollForOld

Function name

HAL_StatusTypeDef HAL_MDF_PollForOld (MDF_HandleTypeDef * hmdf, uint32_t Timeout, uint32_t * pThresholdInfo)

Function description

HAL_MDF_OldStop

Function name

HAL_StatusTypeDef HAL_MDF_OldStop (MDF_HandleTypeDef * hmdf)

Function description

HAL_MDF_OldStart_IT

Function name

HAL_StatusTypeDef HAL_MDF_OldStart_IT (MDF_HandleTypeDef * hmdf, const MDF_OldConfigTypeDef * pOldConfig)

Function description

HAL_MDF_OldStop_IT

Function name

HAL_StatusTypeDef HAL_MDF_OldStop_IT (MDF_HandleTypeDef * hmdf)

Function description

HAL_MDF_OldCallback

Function name

void HAL_MDF_OldCallback (MDF_HandleTypeDef * hmdf, uint32_t ThresholdInfo)

Function description

HAL_MDF_IRQHandler

Function name

void HAL_MDF_IRQHandler (MDF_HandleTypeDef * hmdf)

Function description

HAL_MDF_ErrorCallback

Function name

void HAL_MDF_ErrorCallback (MDF_HandleTypeDef * hmdf)

Function description

HAL_MDF_GetState

Function name

HAL_MDF_StateTypeDef HAL_MDF_GetState (const MDF_HandleTypeDef * hmdf)

Function description

HAL_MDF_GetError

Function name

uint32_t HAL_MDF_GetError (const MDF_HandleTypeDef * hmdf)

Function description

45.3 MDF Firmware driver defines

The following section lists the various define and macros of the module.

45.3.1 MDF

MDF

MDF acquisition mode

MDF_MODE_ASYNC_CONT

Asynchronous, continuous acquisition mode

MDF_MODE_ASYNC_SINGLE

Notes:

- Not available for ADF instance with SAD usage

Asynchronous, single-shot acquisition mode.

MDF_MODE_SYNC_CONT

Synchronous, continuous acquisition mode

MDF_MODE_SYNC_SINGLE

Notes:

- Not available for ADF instance with SAD usage

Synchronous, single-shot acquisition mode.

MDF_MODE_WINDOW_CONT

Notes:

- Not available for ADF instance with SAD usage

Window, continuous acquisition mode.

MDF_MODE_SYNC_SNAPSHOT

Notes:

- Not available for ADF instance

Synchronous, snapshot acquisition mode.

MDF break signals

MDF_NO_BREAK_SIGNAL

No break signal

MDF_TIM1_BREAK_SIGNAL

TIM1_BREAK signal

MDF_TIM1_BREAK2_SIGNAL

TIM1_BREAK2 signal

MDF_TIM8_BREAK_SIGNAL

TIM8_BREAK signal

MDF_TIM8_BREAK2_SIGNAL

TIM8_BREAK2 signal

MDF CIC mode

MDF_TWO_FILTERS_MCIC_FASTSINC

Notes:

- Not available for ADF instance

Two filters, main filter in FastSinc order.

MDF_TWO_FILTERS_MCIC_SINC1

Notes:

- Not available for ADF instance

Two filters, main filter in Sinc1 order.

MDF_TWO_FILTERS_MCIC_SINC2

Notes:

- Not available for ADF instance

Two filters, main filter in Sinc2 order.

MDF_TWO_FILTERS_MCIC_SINC3

Notes:

- Not available for ADF instance

Two filters, main filter in Sinc3 order.

MDF_ONE_FILTER_SINC4

One filter in Sinc4 order

MDF_ONE_FILTER_SINC5

One filter in Sinc5 order

MDF output clock trigger edge

MDF_CLOCK_TRIG_RISING_EDGE

Rising edge

MDF_CLOCK_TRIG_FALLING_EDGE

Falling edge

MDF output clock trigger source

MDF_CLOCK_TRIG_TRGO

MDF_CLOCK_TRIG_EXTI15

MDF_CLOCK_TRIG_EXTI11

Notes:

- Not available for ADF instance

MDF_CLOCK_TRIG_TIM1_TRGO

Notes:

- Not available for ADF instance

MDF_CLOCK_TRIG_TIM1_TRGO2

Notes:

- Not available for ADF instance

MDF_CLOCK_TRIG_TIM3_TRGO

Notes:

- Not available for ADF instance

MDF_CLOCK_TRIG_TIM4_TRGO

Notes:

- Not available for ADF instance

MDF_CLOCK_TRIG_TIM6_TRGO

Notes:

- Not available for ADF instance

MDF_CLOCK_TRIG_TIM7_TRGO

Notes:

- Not available for ADF instance

MDF_CLOCK_TRIG_TIM8_TRGO

Notes:

- Not available for ADF instance

MDF_CLOCK_TRIG_TIM8_TRGO2

Notes:

- Not available for ADF instance

MDF_CLOCK_TRIG_TIM15_TRGO

Notes:

- Not available for ADF instance

MDF_CLOCK_TRIG_LPTIM1_OUT

Notes:

- Not available for ADF instance

MDF_CLOCK_TRIG_SAD_DET

Notes:

- Not available for ADF instance

MDF_CLOCK_TRIG_SAD_TRGO

Notes:

- Not available for ADF instance

MDF data source

MDF_DATA_SOURCE_BSMX

Data from bitstream matrix

MDF_DATA_SOURCE_ADCITF1

Notes:

- Not available for ADF instance

Data from ADC interface 1.

MDF_DATA_SOURCE_ADCITF2

Notes:

- Not available for ADF instance

Data from ADC interface 2.

MDF error code

MDF_ERROR_NONE

No error

MDF_ERROR_ACQUISITION_OVERFLOW

Overflow occurs during acquisition

MDF_ERROR_RSF_OVERRUN

Overrun occurs on reshape filter

MDF_ERROR_CLOCK_ABSENCE

Clock absence detection occurs

MDF_ERROR_SHORT_CIRCUIT

Notes:

- Not used for ADF instance

Short circuit detection occurs.

MDF_ERROR_SATURATION

Saturation detection occurs

MDF_ERROR_OUT_OFF_LIMIT

Notes:

- Not used for ADF instance

Out-off limit detection occurs.

MDF_ERROR_DMA

DMA error occurs

MDF_ERROR_INVALID_CALLBACK

Invalid callback error occurs

MDF Exported Macros

__HAL_MDF_RESET_HANDLE_STATE

Description:

- Reset MDF handle state.

Parameters:

- __HANDLE__: MDF handle.

Return value:

- None

MDF RXFIFO threshold

MDF_FIFO_THRESHOLD_NOT_EMPTY

Event generated when RXFIFO is not empty

MDF_FIFO_THRESHOLD_HALF_FULL

Event generated when RXFIFO is half_full

MDF filter bitstream

MDF_BITSTREAM0_RISING

MDF_BITSTREAM0_FALLING

MDF_BITSTREAM1_RISING

Notes:

- Not available for ADF instance

MDF_BITSTREAM1_FALLING

Notes:

- Not available for ADF instance

MDF_BITSTREAM2_RISING

Notes:

- Not available for ADF instance

MDF_BITSTREAM2_FALLING

Notes:

- Not available for ADF instance

MDF_BITSTREAM3_RISING

Notes:

- Not available for ADF instance

MDF_BITSTREAM3_FALLING

Notes:

- Not available for ADF instance

MDF_BITSTREAM4_RISING

Notes:

- Not available for ADF instance

MDF_BITSTREAM4_FALLING

Notes:

- Not available for ADF instance

MDF_BITSTREAM5_RISING

Notes:

- Not available for ADF instance

MDF_BITSTREAM5_FALLING

Notes:

- Not available for ADF instance

MDF filter trigger edge

MDF_FILTER_TRIG_RISING_EDGE

Rising edge

MDF_FILTER_TRIG_FALLING_EDGE

Falling edge

MDF filter trigger source

MDF_FILTER_TRIG_TRGO

MDF_FILTER_TRIG_OLD_EVENT

Notes:

- Not available for ADF instance

MDF_FILTER_TRIG_EXTI15

MDF_FILTER_TRIG_EXTI11

Notes:

- Not available for ADF instance

MDF_FILTER_TRIG_TIM1_TRGO

Notes:

- Not available for ADF instance

MDF_FILTER_TRIG_TIM1_TRGO2

Notes:

- Not available for ADF instance

MDF_FILTER_TRIG_TIM3_TRGO

Notes:

- Not available for ADF instance

MDF_FILTER_TRIG_TIM4_TRGO

Notes:

- Not available for ADF instance

MDF_FILTER_TRIG_TIM6_TRGO

Notes:

- Not available for ADF instance

MDF_FILTER_TRIG_TIM7_TRGO

Notes:

- Not available for ADF instance

MDF_FILTER_TRIG_TIM8_TRGO

Notes:

- Not available for ADF instance

MDF_FILTER_TRIG_TIM8_TRGO2

Notes:

- Not available for ADF instance

MDF_FILTER_TRIG_TIM15_TRGO

Notes:

- Not available for ADF instance

MDF_FILTER_TRIG_LPTIM1_OUT

Notes:

- Not available for ADF instance

MDF_FILTER_TRIG_ADF_SAD_DET

Notes:

- Not available for ADF instance

MDF_FILTER_TRIG_ADF_TRGO

Notes:

- Not available for ADF instance

MDF high pass filter cut-off frequency

MDF_HPF_CUTOFF_0_000625FPCM

Cut-off frequency of 0.000625xFpcm

MDF_HPF_CUTOFF_0_00125FPCM

Cut-off frequency of 0.00125xFpcm

MDF_HPF_CUTOFF_0_0025FPCM

Cut-off frequency of 0.0025xFpcm

MDF_HPF_CUTOFF_0_0095FPCM

Cut-off frequency of 0.0095xFpcm

MDF integrator output division

MDF_INTEGRATOR_OUTPUT_DIV_128

Integrator data outputs divided by 128

MDF_INTEGRATOR_OUTPUT_DIV_32

Integrator data outputs divided by 32

MDF_INTEGRATOR_OUTPUT_DIV_4

Integrator data outputs divided by 4

MDF_INTEGRATOR_OUTPUT_NO_DIV

Integrator data outputs not divided

MDF out-off limit detector CIC mode

MDF_OLD_FILTER_FASTSINC

Out-off limit detector filter in FastSinc order

MDF_OLD_FILTER_SINC1

Out-off limit detector filter in Sinc1 order

MDF_OLD_FILTER_SINC2

Out-off limit detector filter in Sinc2 order

MDF_OLD_FILTER_SINC3

Out-off limit detector filter in Sinc3 order

MDF out-off limit detector event configuration

MDF_OLD_SIGNAL_OUTSIDE_THRESHOLDS

Out-off limit detector event is generated if signal is outside thresholds

MDF_OLD_SIGNAL_IN_THRESHOLDS

Out-off limit detector event is generated if signal is in thresholds

MDF out-off limit detector threshold information

MDF_OLD_IN_THRESHOLDS

The signal was in thresholds when out-off limit detection occurs

MDF_OLD_HIGH_THRESHOLD

The signal was higher than high threshold when out-off limit detection occurs

MDF_OLD_LOW_THRESHOLD

The signal was lower than low threshold when out-off limit detection occurs

MDF output clock pins

MDF_OUTPUT_CLOCK_0

MDF_CCK0 is used as output clock

MDF_OUTPUT_CLOCK_1

MDF_CCK1 is used as output clock

MDF_OUTPUT_CLOCK_ALL

MDF_CCK0 and MDF_CCK1 are used as output clock

MDF reshape filter decimation ratio

MDF_RSF_DECIMATION_RATIO_4

Reshape filter decimation ratio is 4

MDF_RSF_DECIMATION_RATIO_1

Reshape filter decimation ratio is 1

MDF sound activity detector data memory transfer mode

MDF_SAD_NO_MEMORY_TRANSFER

No memory transfer

MDF_SAD_MEMORY_TRANSFER_IN_DETECT

Memory transfer only in detect state

MDF_SAD_MEMORY_TRANSFER_ALWAYS

Memory transfer always

MDF sound activity detector frame size

MDF_SAD_8_PCM_SAMPLES

Frame size of 8 PCM samples

MDF_SAD_16_PCM_SAMPLES

Frame size of 16 PCM samples

MDF_SAD_32_PCM_SAMPLES

Frame size of 32 PCM samples

MDF_SAD_64_PCM_SAMPLES

Frame size of 64 PCM samples

MDF_SAD_128_PCM_SAMPLES

Frame size of 128 PCM samples

MDF_SAD_256_PCM_SAMPLES

Frame size of 256 PCM samples

MDF_SAD_512_PCM_SAMPLES

Frame size of 512 PCM samples

MDF sound activity detector data hangover time window

MDF_SAD_HANGOVER_4_FRAMES

Hangover window of 4 frames

MDF_SAD_HANGOVER_8_FRAMES

Hangover window of 8 frames

MDF_SAD_HANGOVER_16_FRAMES

Hangover window of 16 frames

MDF_SAD_HANGOVER_32_FRAMES

Hangover window of 32 frames

MDF_SAD_HANGOVER_64_FRAMES

Hangover window of 64 frames

MDF_SAD_HANGOVER_128_FRAMES

Hangover window of 128 frames

MDF_SAD_HANGOVER_256_FRAMES

Hangover window of 256 frames

MDF_SAD_HANGOVER_512_FRAMES

Hangover window of 512 frames

MDF sound activity detector data learning frames

MDF_SAD_LEARNING_2_FRAMES

2 learning frames

MDF_SAD_LEARNING_4_FRAMES

4 learning frames

MDF_SAD_LEARNING_8_FRAMES

8 learning frames

MDF_SAD_LEARNING_16_FRAMES

16 learning frames

MDF_SAD_LEARNING_32_FRAMES

32 learning frames

MDF sound activity detector mode

MDF_SAD_VOICE_ACTIVITY_DETECTOR

Voice activity detector

MDF_SAD_SOUND_DETECTOR

Sound detector

MDF_SAD_AMBIENT_NOISE_DETECTOR

Ambient noise detector

MDF sound activity detector data signal to noise threshold

MDF_SAD_SIGNAL_NOISE_3_5DB

Signal to noise threshold is 3.5dB

MDF_SAD_SIGNAL_NOISE_6DB

Signal to noise threshold is 6dB

MDF_SAD_SIGNAL_NOISE_9_5DB

Signal to noise threshold is 9.5dB

MDF_SAD_SIGNAL_NOISE_12DB

Signal to noise threshold is 12dB

MDF_SAD_SIGNAL_NOISE_15_6DB

Signal to noise threshold is 15.6dB

MDF_SAD_SIGNAL_NOISE_18DB

Signal to noise threshold is 18dB

MDF_SAD_SIGNAL_NOISE_21_6DB

Signal to noise threshold is 21.6dB

MDF_SAD_SIGNAL_NOISE_24_1DB

Signal to noise threshold is 24.1dB

MDF_SAD_SIGNAL_NOISE_27_6DB

Signal to noise threshold is 27.6dB

MDF_SAD_SIGNAL_NOISE_30_1DB

Signal to noise threshold is 30.1dB

MDF sound activity detector trigger event

MDF_SAD_ENTER_DETECT

Event when SAD enters in detect state

MDF_SAD_ENTER_EXIT_DETECT

Event when SAD enters or exits from detect state

MDF serial interface clock source

MDF_SITF_CCK0_SOURCE

Common clock 0 source

MDF_SITF_CCK1_SOURCE

Common clock 1 source

MDF_SITF_CKI_SOURCE

Notes:

- Not available for ADF instance

Dedicated input clock source.

MDF serial interface mode

MDF_SITF_LF_MASTER_SPI_MODE

Low frequency master SPI mode

MDF_SITF_NORMAL_SPI_MODE

Normal SPI mode

MDF_SITF_MANCHESTER_FALLING_MODE

Manchester mode rising edge logic 0 and falling edge logic 1

MDF_SITF_MANCHESTER_RISING_MODE

Manchester mode rising edge logic 1 and falling edge logic 0

MDF snapshot format

MDF_SNAPSHOT_23BITS

Snapshot data resolution of 23 bits

MDF_SNAPSHOT_16BITS

Snapshot data resolution of 16 bits

46 HAL MDIOS Generic Driver

46.1 MDIOS Firmware driver registers structures

46.1.1 MDIOS_InitTypeDef

MDIOS_InitTypeDef is defined in the stm32n6xx_hal_mdios.h

Data Fields

- ***uint32_t PortAddress***
- ***uint32_t PreambleCheck***

Field Documentation

- ***uint32_t MDIOS_InitTypeDef::PortAddress***
  Specifies the MDIOS port address. This parameter can be a value from 0 to 31
- ***uint32_t MDIOS_InitTypeDef::PreambleCheck***
  Specifies whether the preamble check is enabled or disabled. This parameter can be a value of ***MDIOS_Preamble_Check***

46.1.2 MDIOS_HandleTypeDef

MDIOS_HandleTypeDef is defined in the stm32n6xx_hal_mdios.h

Data Fields

- ***MDIOS_TypeDef * Instance***
- ***MDIOS_InitTypeDef Init***
- ***__IO HAL_MDIOS_StateTypeDef State***
- ***__IO uint32_t ErrorCode***
- ***HAL_LockTypeDef Lock***

Field Documentation

- ***MDIOS_TypeDef* MDIOS_HandleTypeDef::Instance***
  Register base address
- ***MDIOS_InitTypeDef MDIOS_HandleTypeDef::Init***
  MDIOS Init Structure
- ***__IO HAL_MDIOS_StateTypeDef MDIOS_HandleTypeDef::State***
  MDIOS communication state This parameter can be a value of of **HAL_MDIOS_StateTypeDef**
- ***__IO uint32_t MDIOS_HandleTypeDef::ErrorCode***
  Holds the global Error code of the MDIOS HAL status machine This parameter can be a value of of ***MDIOS_Error_Code***
- ***HAL_LockTypeDef MDIOS_HandleTypeDef::Lock***
  MDIOS Lock

46.2 MDIOS Firmware driver API description

The following section lists the various functions of the MDIOS library.

46.2.1 How to use this driver

Note: A callback is executed for each generated interrupt, so the driver provide the following HAL_MDIOS_WriteCpltCallback(), HAL_MDIOS_ReadCpltCallback() and HAL_MDIOS_ErrorCallback()

Note: HAL_MDIOS_IRQHandler() must be called from the MDIOS IRQ Handler, to handle the interrupt and execute the previous callbacks (#) Reset the MDIOS peripheral and all related resources by calling the HAL_MDIOS_DeInit() API. (##) HAL_MDIOS_MspDeInit() must be implemented to reset low level resources (GPIO, Clocks, NVIC configuration ...)

Callback registration

46.2.2 Detailed description of functions

HAL_MDIOS_Init

Function name

HAL_StatusTypeDef HAL_MDIOS_Init (MDIOS_HandleTypeDef * hmdios)

Function description

HAL_MDIOS_DeInit

Function name

HAL_StatusTypeDef HAL_MDIOS_DeInit (MDIOS_HandleTypeDef * hmdios)

Function description

HAL_MDIOS_MspInit

Function name

void HAL_MDIOS_MspInit (MDIOS_HandleTypeDef * hmdios)

Function description

HAL_MDIOS_MspDeInit

Function name

void HAL_MDIOS_MspDeInit (MDIOS_HandleTypeDef * hmdios)

Function description

HAL_MDIOS_WriteReg

Function name

HAL_StatusTypeDef HAL_MDIOS_WriteReg (MDIOS_HandleTypeDef * hmdios, uint32_t RegNum, uint16_t Data)

Function description

HAL_MDIOS_ReadReg

Function name

HAL_StatusTypeDef HAL_MDIOS_ReadReg (MDIOS_HandleTypeDef * hmdios, uint32_t RegNum, uint16_t * pData)

Function description

HAL_MDIOS_GetWrittenRegAddress

Function name

uint32_t HAL_MDIOS_GetWrittenRegAddress (const MDIOS_HandleTypeDef * hmdios)

Function description

HAL_MDIOS_GetReadRegAddress

Function name

uint32_t HAL_MDIOS_GetReadRegAddress (const MDIOS_HandleTypeDef * hmdios)

Function description

HAL_MDIOS_ClearWriteRegAddress

Function name

HAL_StatusTypeDef HAL_MDIOS_ClearWriteRegAddress (MDIOS_HandleTypeDef * hmdios, uint32_t RegNum)

Function description

HAL_MDIOS_ClearReadRegAddress

Function name

HAL_StatusTypeDef HAL_MDIOS_ClearReadRegAddress (MDIOS_HandleTypeDef * hmdios, uint32_t RegNum)

Function description

HAL_MDIOS_EnableEvents

Function name

HAL_StatusTypeDef HAL_MDIOS_EnableEvents (MDIOS_HandleTypeDef * hmdios)

Function description

HAL_MDIOS_IRQHandler

Function name

void HAL_MDIOS_IRQHandler (MDIOS_HandleTypeDef * hmdios)

Function description

HAL_MDIOS_WriteCpltCallback

Function name

void HAL_MDIOS_WriteCpltCallback (MDIOS_HandleTypeDef * hmdios)

Function description

HAL_MDIOS_ReadCpltCallback

Function name

void HAL_MDIOS_ReadCpltCallback (MDIOS_HandleTypeDef * hmdios)

Function description

HAL_MDIOS_ErrorCallback

Function name

void HAL_MDIOS_ErrorCallback (MDIOS_HandleTypeDef * hmdios)

Function description

HAL_MDIOS_WakeUpCallback

Function name

void HAL_MDIOS_WakeUpCallback (MDIOS_HandleTypeDef * hmdios)

Function description

HAL_MDIOS_GetError

Function name

uint32_t HAL_MDIOS_GetError (const MDIOS_HandleTypeDef * hmdios)

Function description

HAL_MDIOS_GetState

Function name

HAL_MDIOS_StateTypeDef HAL_MDIOS_GetState (const MDIOS_HandleTypeDef * hmdios)

Function description

46.3 MDIOS Firmware driver defines

The following section lists the various define and macros of the module.

46.3.1 MDIOS

MDIOS

MDIOS Error Code

HAL_MDIOS_ERROR_NONE

No error

HAL_MDIOS_ERROR_PARAM

Busy error

HAL_MDIOS_ERROR_BUSY

Parameter error

HAL_MDIOS_ERROR_TIMEOUT

Timeout error

HAL_MDIOS_ERROR_DATA

Data transfer error

MDIOS Exported Macros

__HAL_MDIOS_RESET_HANDLE_STATE

Description:

- Reset MDIOS handle state.

Parameters:

- __HANDLE__: MDIOS handle.

Return value:

- None

__HAL_MDIOS_ENABLE

Description:

- Enable/Disable the MDIOS peripheral.

Parameters:

- __HANDLE__: specifies the MDIOS handle.

Return value:

- None

__HAL_MDIOS_DISABLE

__HAL_MDIOS_ENABLE_IT

Description:

- Enable the MDIOS device interrupt.

Parameters:

- __HANDLE__: specifies the MDIOS handle.
- __INTERRUPT__: : specifies the MDIOS interrupt sources to be enabled. This parameter can be one or a combination of the following values:
  - MDIOS_IT_WRITE: Register write interrupt
  - MDIOS_IT_READ: Register read interrupt
  - MDIOS_IT_ERROR: Error interrupt

Return value:

- None

__HAL_MDIOS_DISABLE_IT

Description:

- Disable the MDIOS device interrupt.

Parameters:

- __HANDLE__: specifies the MDIOS handle.
- __INTERRUPT__: : specifies the MDIOS interrupt sources to be disabled. This parameter can be one or a combination of the following values:
  - MDIOS_IT_WRITE: Register write interrupt
  - MDIOS_IT_READ: Register read interrupt
  - MDIOS_IT_ERROR: Error interrupt

Return value:

- None

__HAL_MDIOS_GET_WRITE_FLAG

Description:

- Set MDIOS slave get write register flag.

Parameters:

- __HANDLE__: specifies the MDIOS handle.
- __FLAG__: specifies the write register flag

Return value:

- The: state of write flag

__HAL_MDIOS_GET_READ_FLAG

Description:

- MDIOS slave get read register flag.

Parameters:

- __HANDLE__: specifies the MDIOS handle.
- __FLAG__: specifies the read register flag

Return value:

- The: state of read flag

__HAL_MDIOS_GET_ERROR_FLAG

Description:

- MDIOS slave get interrupt.

Parameters:

- __HANDLE__: specifies the MDIOS handle.
- __FLAG__: : specifies the Error flag. This parameter can be one or a combination of the following values:
  - MDIOS_TURNARROUND_ERROR_FLAG: Register write interrupt
  - MDIOS_START_ERROR_FLAG: Register read interrupt
  - MDIOS_PREAMBLE_ERROR_FLAG: Error interrupt

Return value:

- The: state of the error flag

__HAL_MDIOS_CLEAR_ERROR_FLAG

Description:

- MDIOS slave clear interrupt.

Parameters:

- __HANDLE__: specifies the MDIOS handle.
- __FLAG__: : specifies the Error flag. This parameter can be one or a combination of the following values:
  - MDIOS_TURNARROUND_ERROR_FLAG: Register write interrupt
  - MDIOS_START_ERROR_FLAG: Register read interrupt
  - MDIOS_PREAMBLE_ERROR_FLAG: Error interrupt

Return value:

- none

__HAL_MDIOS_GET_IT_SOURCE

Description:

- Checks whether the specified MDIOS interrupt is set or not.

Parameters:

- __HANDLE__: specifies the MDIOS handle.
- __INTERRUPT__: : specifies the MDIOS interrupt sources This parameter can be one or a combination of the following values:
  - MDIOS_IT_WRITE: Register write interrupt
  - MDIOS_IT_READ: Register read interrupt
  - MDIOS_IT_ERROR: Error interrupt

Return value:

- The: state of the interrupt source

__HAL_MDIOS_WAKEUP_EXTI_ENABLE_IT

Description:

- Enable the MDIOS WAKEUP Exti Line.

Parameters:

- __EXTI_LINE__: specifies the MDIOS WAKEUP Exti sources to be enabled. This parameter can be:
  - MDIOS_WAKEUP_EXTI_LINE

Return value:

- None.

__HAL_MDIOS_WAKEUP_EXTI_GET_FLAG

Description:

- checks whether the specified MDIOS WAKEUP Exti interrupt flag is set or not.

Parameters:

- __EXTI_LINE__: specifies the MDIOS WAKEUP Exti sources to be cleared. This parameter can be:
  - MDIOS_WAKEUP_EXTI_LINE

Return value:

- EXTI: MDIOS WAKEUP Line Status.

__HAL_MDIOS_WAKEUP_EXTI_CLEAR_FLAG

Description:

- Clear the MDIOS WAKEUP Exti flag.

Parameters:

- __EXTI_LINE__: specifies the MDIOS WAKEUP Exti sources to be cleared. This parameter can be:
  - MDIOS_WAKEUP_EXTI_LINE

Return value:

- None.

__HAL_MDIOS_WAKEUP_EXTI_ENABLE_RISING_EDGE

Description:

- enable rising edge interrupt on selected EXTI line.

Parameters:

- __EXTI_LINE__: specifies the MDIOS WAKEUP EXTI sources to be disabled. This parameter can be:
  - MDIOS_WAKEUP_EXTI_LINE

Return value:

- None

__HAL_MDIOS_WAKEUP_EXTI_ENABLE_FALLING_EDGE

Description:

- enable falling edge interrupt on selected EXTI line.

Parameters:

- __EXTI_LINE__: specifies the MDIOS WAKEUP EXTI sources to be disabled. This parameter can be:
  - MDIOS_WAKEUP_EXTI_LINE

Return value:

- None

__HAL_MDIOS_WAKEUP_EXTI_ENABLE_RISING_FALLING_EDGE

Description:

- enable falling edge interrupt on selected EXTI line.

Parameters:

- __EXTI_LINE__: specifies the MDIOS WAKEUP EXTI sources to be disabled. This parameter can be:
  - MDIOS_WAKEUP_EXTI_LINE

Return value:

- None

__HAL_MDIOS_WAKEUP_EXTI_GENERATE_SWIT

Description:

- Generates a Software interrupt on selected EXTI line.

Parameters:

- __EXTI_LINE__: specifies the MDIOS WAKEUP EXTI sources to be disabled. This parameter can be:
  - MDIOS_WAKEUP_EXTI_LINE

Return value:

- None

MDIOS Input Output Registers Definitions

MDIOS_REG0

MDIOS_REG1

MDIOS_REG2

MDIOS_REG3

MDIOS_REG4

MDIOS_REG5

MDIOS_REG6

MDIOS_REG7

MDIOS_REG8

MDIOS_REG9

MDIOS_REG10

MDIOS_REG11

MDIOS_REG12

MDIOS_REG13

MDIOS_REG14

MDIOS_REG15

MDIOS_REG16

MDIOS_REG17

MDIOS_REG18

MDIOS_REG19

MDIOS_REG20

MDIOS_REG21

MDIOS_REG22

MDIOS_REG23

MDIOS_REG24

MDIOS_REG25

MDIOS_REG26

MDIOS_REG27

MDIOS_REG28

MDIOS_REG29

MDIOS_REG30

MDIOS_REG31

MDIOS Interrupt Flags

MDIOS_TURNAROUND_ERROR_FLAG

MDIOS_START_ERROR_FLAG

MDIOS_PREAMBLE_ERROR_FLAG

Interrupt Sources

MDIOS_IT_WRITE

MDIOS_IT_READ

MDIOS_IT_ERROR

MDIOS Preamble Check

MDIOS_PREAMBLE_CHECK_ENABLE

MDIOS_PREAMBLE_CHECK_DISABLE

MDIOS Registers Flags

MDIOS_REG0_FLAG

MDIOS_REG1_FLAG

MDIOS_REG2_FLAG

MDIOS_REG3_FLAG

MDIOS_REG4_FLAG

MDIOS_REG5_FLAG

MDIOS_REG6_FLAG

MDIOS_REG7_FLAG

MDIOS_REG8_FLAG

MDIOS_REG9_FLAG

MDIOS_REG10_FLAG

MDIOS_REG11_FLAG

MDIOS_REG12_FLAG

MDIOS_REG13_FLAG

MDIOS_REG14_FLAG

MDIOS_REG15_FLAG

MDIOS_REG16_FLAG

MDIOS_REG17_FLAG

MDIOS_REG18_FLAG

MDIOS_REG19_FLAG

MDIOS_REG20_FLAG

MDIOS_REG21_FLAG

MDIOS_REG22_FLAG

MDIOS_REG23_FLAG

MDIOS_REG24_FLAG

MDIOS_REG25_FLAG

MDIOS_REG26_FLAG

MDIOS_REG27_FLAG

MDIOS_REG28_FLAG

MDIOS_REG29_FLAG

MDIOS_REG30_FLAG

MDIOS_REG31_FLAG

MDIOS_ALLREG_FLAG

MDIOS Wakeup Line

MDIOS_WAKEUP_EXTI_LINE

47 HAL MMC Generic Driver

47.1 MMC Firmware driver registers structures

47.1.1 HAL_MMC_CardInfoTypeDef

HAL_MMC_CardInfoTypeDef is defined in the stm32n6xx_hal_mmc.h

Data Fields

- ***uint32_t CardType***
- ***uint32_t Class***
- ***uint32_t RelCardAdd***
- ***uint32_t BlockNbr***
- ***uint32_t BlockSize***
- ***uint32_t LogBlockNbr***
- ***uint32_t LogBlockSize***

Field Documentation

- ***uint32_t HAL_MMC_CardInfoTypeDef::CardType***
  Specifies the card Type
- ***uint32_t HAL_MMC_CardInfoTypeDef::Class***
  Specifies the class of the card class
- ***uint32_t HAL_MMC_CardInfoTypeDef::RelCardAdd***
  Specifies the Relative Card Address
- ***uint32_t HAL_MMC_CardInfoTypeDef::BlockNbr***
  Specifies the Card Capacity in blocks
- ***uint32_t HAL_MMC_CardInfoTypeDef::BlockSize***
  Specifies one block size in bytes
- ***uint32_t HAL_MMC_CardInfoTypeDef::LogBlockNbr***
  Specifies the Card logical Capacity in blocks
- ***uint32_t HAL_MMC_CardInfoTypeDef::LogBlockSize***
  Specifies logical block size in bytes

47.1.2 __MMC_HandleTypeDef

__MMC_HandleTypeDef is defined in the stm32n6xx_hal_mmc.h

Data Fields

- ***MMC_TypeDef * Instance***
- ***MMC_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***const uint8_t * pTxBuffPtr***
- ***uint32_t TxXferSize***
- ***uint8_t * pRxBuffPtr***
- ***uint32_t RxXferSize***
- ***__IO uint32_t Context***
- ***__IO HAL_MMC_StateTypeDef State***
- ***__IO uint32_t ErrorCode***
- ***__IO uint16_t RPMBErrorCode***
- ***HAL_MMC_CardInfoTypeDef MmcCard***
- ***uint32_t CSD***
- ***uint32_t CID***
- ***uint32_t Ext_CSD***
- ***void(* TxCpltCallback***
- ***void(* RxCpltCallback***

- ***void(* ErrorCallback***
- ***void(* AbortCpltCallback***
- ***void(* Read_DMALnkLstBufCpltCallback***
- ***void(* Write_DMALnkLstBufCpltCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***MMC_TypeDef* __MMC_HandleTypeDef::Instance***
  MMC registers base address
- ***MMC_InitTypeDef __MMC_HandleTypeDef::Init***
  MMC required parameters
- ***HAL_LockTypeDef __MMC_HandleTypeDef::Lock***
  MMC locking object
- ***const uint8_t* __MMC_HandleTypeDef::pTxBuffPtr***
  Pointer to MMC Tx transfer Buffer
- ***uint32_t __MMC_HandleTypeDef::TxXferSize***
  MMC Tx Transfer size
- ***uint8_t* __MMC_HandleTypeDef::pRxBuffPtr***
  Pointer to MMC Rx transfer Buffer
- ***uint32_t __MMC_HandleTypeDef::RxXferSize***
  MMC Rx Transfer size
- ***__IO uint32_t __MMC_HandleTypeDef::Context***
  MMC transfer context
- ***__IO HAL_MMC_StateTypeDef __MMC_HandleTypeDef::State***
  MMC card State
- ***__IO uint32_t __MMC_HandleTypeDef::ErrorCode***
  MMC Card Error codes
- ***__IO uint16_t __MMC_HandleTypeDef::RPMBErrorCode***
  MMC RPMB Area Error codes
- ***HAL_MMC_CardInfoTypeDef __MMC_HandleTypeDef::MmcCard***
  MMC Card information
- ***uint32_t __MMC_HandleTypeDef::CSD[4U]***
  MMC card specific data table
- ***uint32_t __MMC_HandleTypeDef::CID[4U]***
  MMC card identification number table
- ***uint32_t __MMC_HandleTypeDef::Ext_CSD[128]***
- ***void(* __MMC_HandleTypeDef::TxCpltCallback)(struct __MMC_HandleTypeDef *hmmc)***
- ***void(* __MMC_HandleTypeDef::RxCpltCallback)(struct __MMC_HandleTypeDef *hmmc)***
- ***void(* __MMC_HandleTypeDef::ErrorCallback)(struct __MMC_HandleTypeDef *hmmc)***
- ***void(* __MMC_HandleTypeDef::AbortCpltCallback)(struct __MMC_HandleTypeDef *hmmc)***
- ***void(* __MMC_HandleTypeDef::Read_DMALnkLstBufCpltCallback)(struct __MMC_HandleTypeDef *hmmc)***
- ***void(* __MMC_HandleTypeDef::Write_DMALnkLstBufCpltCallback)(struct __MMC_HandleTypeDef *hmmc)***
- ***void(* __MMC_HandleTypeDef::MspInitCallback)(struct __MMC_HandleTypeDef *hmmc)***
- ***void(* __MMC_HandleTypeDef::MspDeInitCallback)(struct __MMC_HandleTypeDef *hmmc)***

47.1.3 HAL_MMC_CardCSDTypeDef

HAL_MMC_CardCSDTypeDef is defined in the stm32n6xx_hal_mmc.h

Data Fields

- ***__IO uint8_t CSDStruct***
- ***__IO uint8_t SysSpecVersion***
- ***__IO uint8_t Reserved1***

- ***__IO uint8_t TAAC***
- ***__IO uint8_t NSAC***
- ***__IO uint8_t MaxBusClkFrec***
- ***__IO uint16_t CardComdClasses***
- ***__IO uint8_t RdBlockLen***
- ***__IO uint8_t PartBlockRead***
- ***__IO uint8_t WrBlockMisalign***
- ***__IO uint8_t RdBlockMisalign***
- ***__IO uint8_t DSRImpl***
- ***__IO uint8_t Reserved2***
- ***__IO uint32_t DeviceSize***
- ***__IO uint8_t MaxRdCurrentVDDMin***
- ***__IO uint8_t MaxRdCurrentVDDMax***
- ***__IO uint8_t MaxWrCurrentVDDMin***
- ***__IO uint8_t MaxWrCurrentVDDMax***
- ***__IO uint8_t DeviceSizeMul***
- ***__IO uint8_t EraseGrSize***
- ***__IO uint8_t EraseGrMul***
- ***__IO uint8_t WrProtectGrSize***
- ***__IO uint8_t WrProtectGrEnable***
- ***__IO uint8_t ManDeflECC***
- ***__IO uint8_t WrSpeedFact***
- ***__IO uint8_t MaxWrBlockLen***
- ***__IO uint8_t WriteBlockPaPartial***
- ***__IO uint8_t Reserved3***
- ***__IO uint8_t ContentProtectAppli***
- ***__IO uint8_t FileFormatGroup***
- ***__IO uint8_t CopyFlag***
- ***__IO uint8_t PermWrProtect***
- ***__IO uint8_t TempWrProtect***
- ***__IO uint8_t FileFormat***
- ***__IO uint8_t ECC***
- ***__IO uint8_t CSD_CRC***
- ***__IO uint8_t Reserved4***

Field Documentation

- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::CSDStruct***
  CSD structure
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::SysSpecVersion***
  System specification version
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::Reserved1***
  Reserved
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::TAAC***
  Data read access time 1
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::NSAC***
  Data read access time 2 in CLK cycles
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::MaxBusClkFrec***
  Max. bus clock frequency
- ***__IO uint16_t HAL_MMC_CardCSDTypeDef::CardComdClasses***
  Card command classes
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::RdBlockLen***
  Max. read data block length

- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::PartBlockRead***
  Partial blocks for read allowed
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::WrBlockMisalign***
  Write block misalignment
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::RdBlockMisalign***
  Read block misalignment
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::DSRImpl***
  DSR implemented
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::Reserved2***
  Reserved
- ***__IO uint32_t HAL_MMC_CardCSDTypeDef::DeviceSize***
  Device Size
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::MaxRdCurrentVDDMin***
  Max. read current @ VDD min
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::MaxRdCurrentVDDMax***
  Max. read current @ VDD max
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::MaxWrCurrentVDDMin***
  Max. write current @ VDD min
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::MaxWrCurrentVDDMax***
  Max. write current @ VDD max
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::DeviceSizeMul***
  Device size multiplier
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::EraseGrSize***
  Erase group size
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::EraseGrMul***
  Erase group size multiplier
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::WrProtectGrSize***
  Write protect group size
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::WrProtectGrEnable***
  Write protect group enable
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::ManDeflECC***
  Manufacturer default ECC
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::WrSpeedFact***
  Write speed factor
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::MaxWrBlockLen***
  Max. write data block length
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::WriteBlockPaPartial***
  Partial blocks for write allowed
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::Reserved3***
  Reserved
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::ContentProtectAppli***
  Content protection application
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::FileFormatGroup***
  File format group
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::CopyFlag***
  Copy flag (OTP)
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::PermWrProtect***
  Permanent write protection
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::TempWrProtect***
  Temporary write protection
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::FileFormat***
  File format
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::ECC***
  ECC code
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::CSD_CRC***
  CSD CRC

- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::Reserved4***
  Always 1

47.1.4 HAL_MMC_CardCIDTypeDef

HAL_MMC_CardCIDTypeDef is defined in the stm32n6xx_hal_mmc.h

Data Fields

- ***__IO uint8_t ManufacturerID***
- ***__IO uint16_t OEM_AppliID***
- ***__IO uint32_t ProdName1***
- ***__IO uint8_t ProdName2***
- ***__IO uint8_t ProdRev***
- ***__IO uint32_t ProdSN***
- ***__IO uint8_t Reserved1***
- ***__IO uint16_t ManufactDate***
- ***__IO uint8_t CID_CRC***
- ***__IO uint8_t Reserved2***

Field Documentation

- ***__IO uint8_t HAL_MMC_CardCIDTypeDef::ManufacturerID***
  Manufacturer ID
- ***__IO uint16_t HAL_MMC_CardCIDTypeDef::OEM_AppliID***
  OEM/Application ID
- ***__IO uint32_t HAL_MMC_CardCIDTypeDef::ProdName1***
  Product Name part1
- ***__IO uint8_t HAL_MMC_CardCIDTypeDef::ProdName2***
  Product Name part2
- ***__IO uint8_t HAL_MMC_CardCIDTypeDef::ProdRev***
  Product Revision
- ***__IO uint32_t HAL_MMC_CardCIDTypeDef::ProdSN***
  Product Serial Number
- ***__IO uint8_t HAL_MMC_CardCIDTypeDef::Reserved1***
  Reserved1
- ***__IO uint16_t HAL_MMC_CardCIDTypeDef::ManufactDate***
  Manufacturing Date
- ***__IO uint8_t HAL_MMC_CardCIDTypeDef::CID_CRC***
  CID CRC
- ***__IO uint8_t HAL_MMC_CardCIDTypeDef::Reserved2***
  Always 1

47.2 MMC Firmware driver API description

The following section lists the various functions of the MMC library.

47.2.1 How to use this driver

This driver implements a high level communication layer for read and write from/to this memory. The needed STM32 hardware resources (SDMMC and GPIO) are performed by the user in HAL_MMC_MspInit() function (MSP layer). Basically, the MSP layer configuration should be the same as we provide in the examples. You can easily tailor this configuration according to hardware resources.

This driver is a generic layered driver for SDMMC memories which uses the HAL SDMMC driver functions to interface with MMC and eMMC cards devices. It is used as follows:

1. Initialize the SDMMC low level resources by implement the HAL_MMC_MspInit() API:
   a. Enable the SDMMC interface clock using __HAL_RCC_SDMMC_CLK_ENABLE();
   b. SDMMC pins configuration for MMC card
      - Enable the clock for the SDMMC GPIOs using the functions __HAL_RCC_GPIOx_CLK_ENABLE();
      - Configure these SDMMC pins as alternate function pull-up using HAL_GPIO_Init() and according to your pin assignment;
   c. NVIC configuration if you need to use interrupt process (HAL_MMC_ReadBlocks_IT() and HAL_MMC_WriteBlocks_IT() APIs).
      - Configure the SDMMC interrupt priorities using function HAL_NVIC_SetPriority();
      - Enable the NVIC SDMMC IRQs using function HAL_NVIC_EnableIRQ()
      - SDMMC interrupts are managed using the macros __HAL_MMC_ENABLE_IT() and __HAL_MMC_DISABLE_IT() inside the communication process.
      - SDMMC interrupts pending bits are managed using the macros __HAL_MMC_GET_IT() and __HAL_MMC_CLEAR_IT()
   d. No general propose DMA Configuration is needed, an Internal DMA for SDMMC Peripheral are used.
2. At this stage, you can perform MMC read/write/erase operations after MMC card initialization

MMC Card Initialization and configuration

To initialize the MMC Card, use the HAL_MMC_Init() function. It Initializes SDMMC Peripheral (STM32 side) and the MMC Card, and put it into StandBy State (Ready for data transfer). This function provide the following operations:

1. Initialize the SDMMC peripheral interface with default configuration. The initialization process is done at 400KHz. You can change or adapt this frequency by adjusting the "ClockDiv" field. The MMC Card frequency (SDMMC_CK) is computed as follows: SDMMC_CK = SDMMCCLK / (2 * ClockDiv) In initialization mode and according to the MMC Card standard, make sure that the SDMMC_CK frequency doesn't exceed 400KHz. This phase of initialization is done through SDMMC_Init() and SDMMC_PowerState_ON() SDMMC low level APIs.
2. Initialize the MMC card. The API used is HAL_MMC_InitCard(). This phase allows the card initialization and identification and check the MMC Card type (Standard Capacity or High Capacity) The initialization flow is compatible with MMC standard. This API (HAL_MMC_InitCard()) could be used also to reinitialize the card in case of plug-off plug-in.
3. Configure the MMC Card Data transfer frequency. By Default, the card transfer frequency by adjusting the "ClockDiv" field. In transfer mode and according to the MMC Card standard, make sure that the SDMMC_CK frequency doesn't exceed 25MHz and 100MHz in High-speed mode switch.
4. Select the corresponding MMC Card according to the address read with the step 2.
5. Configure the MMC Card in wide bus mode: 4-bits data.
6. Select the MMC Card partition using HAL_MMC_SwitchPartition()

MMC Card Read operation

- You can read from MMC card in polling mode by using function HAL_MMC_ReadBlocks(). This function support only 512-bytes block length (the block size should be chosen as 512 bytes). You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter. After this, you have to ensure that the transfer is done correctly. The check is done through HAL_MMC_GetCardState() function for MMC card state.
- You can read from MMC card in DMA mode by using function HAL_MMC_ReadBlocks_DMA(). This function support only 512-bytes block length (the block size should be chosen as 512 bytes). You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter. After this, you have to ensure that the transfer is done correctly. The check is done through HAL_MMC_GetCardState() function for MMC card state. You could also check the DMA transfer process through the MMC Rx interrupt event.
- You can read from MMC card in Interrupt mode by using function HAL_MMC_ReadBlocks_IT(). This function allows the read of 512 bytes blocks. You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter. After this, you have to ensure that the transfer is done correctly. The check is done through HAL_MMC_GetCardState() function for MMC card state. You could also check the IT transfer process through the MMC Rx interrupt event.

MMC Card Write operation

- You can write to MMC card in polling mode by using function HAL_MMC_WriteBlocks(). This function support only 512-bytes block length (the block size should be chosen as 512 bytes). You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter. After this, you have to ensure that the transfer is done correctly. The check is done through HAL_MMC_GetCardState() function for MMC card state.
- You can write to MMC card in DMA mode by using function HAL_MMC_WriteBlocks_DMA(). This function support only 512-bytes block length (the block size should be chosen as 512 byte). You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter. After this, you have to ensure that the transfer is done correctly. The check is done through HAL_MMC_GetCardState() function for MMC card state. You could also check the DMA transfer process through the MMC Tx interrupt event.
- You can write to MMC card in Interrupt mode by using function HAL_MMC_WriteBlocks_IT(). This function allows the read of 512 bytes blocks. You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter. After this, you have to ensure that the transfer is done correctly. The check is done through HAL_MMC_GetCardState() function for MMC card state. You could also check the IT transfer process through the MMC Tx interrupt event.

MMC card information

- To get MMC card information, you can use the function HAL_MMC_GetCardInfo(). It returns useful information about the MMC card such as block size, card type, block number ...

MMC card CSD register

- The HAL_MMC_GetCardCSD() API allows to get the parameters of the CSD register. Some of the CSD parameters are useful for card initialization and identification.

MMC card CID register

- The HAL_MMC_GetCardCID() API allows to get the parameters of the CID register. Some of the CID parameters are useful for card initialization and identification.

MMC Card Reply Protected Memory Block (RPMB) Key Programming operation

- You can program the authentication key of RPMB area in polling mode by using function HAL_MMC_RPMB_ProgramAuthenticationKey(). This function is only used once during the life of an MMC card. After this, you have to ensure that the transfer is done correctly. The check is done through HAL_MMC_GetRPMBError() function for operation state.
- You can program the authentication key of RPMB area in Interrupt mode by using function HAL_MMC_RPMB_ProgramAuthenticationKey_IT(). This function is only used once during the life of an MMC card. After this, you have to ensure that the transfer is done correctly. The check is done through HAL_MMC_GetRPMBError() function for operation state.

MMC Card Reply Protected Memory Block (RPMB) write counter operation

- You can get the write counter value of RPMB area in polling mode by using function HAL_MMC_RPMB_GetWriteCounter().
- You can get the write counter value of RPMB area in Interrupt mode by using function HAL_MMC_RPMB_GetWriteCounter_IT().

MMC Card Reply Protected Memory Block (RPMB) write operation

- You can write to the RPMB area of MMC card in polling mode by using function HAL_MMC_WriteBlocks(). This function supports the one, two, or thirty two blocks write operation (with 512-bytes block length). You can choose the number of blocks at the multiple block read operation by adjusting the "NumberOfBlocks" parameter. After this, you have to ensure that the transfer is done correctly. The check is done through HAL_MMC_GetRPMBError() function for operation state.

- You can write to the RPMB area of MMC card in Interrupt mode by using function HAL_MMC_WriteBlocks_IT(). This function supports the one, two, or thirty two blocks write operation (with 512-bytes block length). You can choose the number of blocks at the multiple block read operation by adjusting the "NumberOfBlocks" parameter. After this, you have to ensure that the transfer is done correctly. The check is done through HAL_MMC_GetRPMBError() function for operation state.

MMC Card Reply Protected Memory Block (RPMB) read operation

- You can read from the RPMB area of MMC card in polling mode by using function HAL_MMC_RPMB_ReadBlocks(). The block size should be chosen as multiple of 512 bytes. You can choose the number of blocks by adjusting the "NumberOfBlocks" parameter. After this, you have to ensure that the transfer is done correctly. The check is done through HAL_MMC_GetRPMBError() function for MMC card state.
- You can read from the RPMB area of MMC card in Interrupt mode by using function HAL_MMC_RPMB_ReadBlocks_IT(). The block size should be chosen as multiple of 512 bytes. You can choose the number of blocks by adjusting the "NumberOfBlocks" parameter. After this, you have to ensure that the transfer is done correctly. The check is done through HAL_MMC_GetRPMBError() function for MMC card state.

MMC HAL driver macros list

Below the list of most used macros in MMC HAL driver.

- __HAL_MMC_ENABLE_IT: Enable the MMC device interrupt
- __HAL_MMC_DISABLE_IT: Disable the MMC device interrupt
- __HAL_MMC_GET_FLAG:Check whether the specified MMC flag is set or not
- __HAL_MMC_CLEAR_FLAG: Clear the MMC's pending flags

Note: You can refer to the MMC HAL driver header file for more useful macros

Callback registration

The compilation define USE_HAL_MMC_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use Functions HAL_MMC_RegisterCallback() to register a user callback, it allows to register following callbacks:

- TxCpltCallback : callback when a transmission transfer is completed.
- RxCpltCallback : callback when a reception transfer is completed.
- ErrorCallback : callback when error occurs.
- AbortCpltCallback : callback when abort is completed.
- Read_DMALnkLstBufCpltCallback : callback when the DMA reception of linked list node buffer is completed.
- Write_DMALnkLstBufCpltCallback : callback when the DMA transmission of linked list node buffer is completed.
- MspInitCallback : MMC MspInit.
- MspDeInitCallback : MMC MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function. Use function HAL_MMC_UnRegisterCallback() to reset a callback to the default weak (overridden) function. It allows to reset following callbacks:
- TxCpltCallback : callback when a transmission transfer is completed.
- RxCpltCallback : callback when a reception transfer is completed.
- ErrorCallback : callback when error occurs.
- AbortCpltCallback : callback when abort is completed.
- Read_DMADblBuf0CpltCallback : callback when the DMA reception of first buffer is completed.
- Read_DMADblBuf1CpltCallback : callback when the DMA reception of second buffer is completed.
- Write_DMADblBuf0CpltCallback : callback when the DMA transmission of first buffer is completed.
- Write_DMADblBuf1CpltCallback : callback when the DMA transmission of second buffer is completed.
- MspInitCallback : MMC MspInit.

- MspDeInitCallback : MMC MspDeInit. This function) takes as parameters the HAL peripheral handle and the Callback ID. By default, after the HAL_MMC_Init and if the state is HAL_MMC_STATE_RESET all callbacks are reset to the corresponding legacy weak (overridden) functions. Exception done for MspInit and MspDeInit callbacks that are respectively reset to the legacy weak (overridden) functions in the HAL_MMC_Init and HAL_MMC_DeInit only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the HAL_MMC_Init and HAL_MMC_DeInit keep and use the user MspInit/MspDeInit callbacks (registered beforehand) Callbacks can be registered/unregistered in READY state only. Exception done for MspInit/MspDeInit callbacks that can be registered/unregistered in READY or RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_MMC_RegisterCallback before calling HAL_MMC_DeInit or HAL_MMC_Init function. When The compilation define USE_HAL_MMC_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak (overridden) callbacks are used.

47.2.2 Detailed description of functions

HAL_MMC_Init

Function name

HAL_StatusTypeDef HAL_MMC_Init (MMC_HandleTypeDef * hmmc)

Function description

HAL_MMC_InitCard

Function name

HAL_StatusTypeDef HAL_MMC_InitCard (MMC_HandleTypeDef * hmmc)

Function description

HAL_MMC_DeInit

Function name

HAL_StatusTypeDef HAL_MMC_DeInit (MMC_HandleTypeDef * hmmc)

Function description

HAL_MMC_MspInit

Function name

void HAL_MMC_MspInit (MMC_HandleTypeDef * hmmc)

Function description

HAL_MMC_MspDeInit

Function name

void HAL_MMC_MspDeInit (MMC_HandleTypeDef * hmmc)

Function description

HAL_MMC_ReadBlocks

Function name

HAL_StatusTypeDef HAL_MMC_ReadBlocks (MMC_HandleTypeDef * hmmc, uint8_t * pData, uint32_t BlockAdd, uint32_t NumberOfBlocks, uint32_t Timeout)

Function description

HAL_MMC_WriteBlocks

Function name

HAL_StatusTypeDef HAL_MMC_WriteBlocks (MMC_HandleTypeDef * hmmc, const uint8_t * pData, uint32_t BlockAdd, uint32_t NumberOfBlocks, uint32_t Timeout)

Function description

HAL_MMC_Erase

Function name

HAL_StatusTypeDef HAL_MMC_Erase (MMC_HandleTypeDef * hmmc, uint32_t BlockStartAdd, uint32_t BlockEndAdd)

Function description

HAL_MMC_ReadBlocks_IT

Function name

HAL_StatusTypeDef HAL_MMC_ReadBlocks_IT (MMC_HandleTypeDef * hmmc, uint8_t * pData, uint32_t BlockAdd, uint32_t NumberOfBlocks)

Function description

HAL_MMC_WriteBlocks_IT

Function name

HAL_StatusTypeDef HAL_MMC_WriteBlocks_IT (MMC_HandleTypeDef * hmmc, const uint8_t * pData, uint32_t BlockAdd, uint32_t NumberOfBlocks)

Function description

HAL_MMC_ReadBlocks_DMA

Function name

HAL_StatusTypeDef HAL_MMC_ReadBlocks_DMA (MMC_HandleTypeDef * hmmc, uint8_t * pData, uint32_t BlockAdd, uint32_t NumberOfBlocks)

Function description

HAL_MMC_WriteBlocks_DMA

Function name

HAL_StatusTypeDef HAL_MMC_WriteBlocks_DMA (MMC_HandleTypeDef * hmmc, const uint8_t * pData, uint32_t BlockAdd, uint32_t NumberOfBlocks)

Function description

HAL_MMC_IRQHandler

Function name

void HAL_MMC_IRQHandler (MMC_HandleTypeDef * hmmc)

Function description

HAL_MMC_TxCpltCallback

Function name

void HAL_MMC_TxCpltCallback (MMC_HandleTypeDef * hmmc)

Function description

HAL_MMC_RxCpltCallback

Function name

void HAL_MMC_RxCpltCallback (MMC_HandleTypeDef * hmmc)

Function description

HAL_MMC_ErrorCallback

Function name

void HAL_MMC_ErrorCallback (MMC_HandleTypeDef * hmmc)

Function description

HAL_MMC_AbortCallback

Function name

void HAL_MMC_AbortCallback (MMC_HandleTypeDef * hmmc)

Function description

HAL_MMC_RegisterCallback

Function name

HAL_StatusTypeDef HAL_MMC_RegisterCallback (MMC_HandleTypeDef * hmmc, HAL_MMC_CallbackIDTypeDef CallbackId, pMMC_CallbackTypeDef pCallback)

Function description

HAL_MMC_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_MMC_UnRegisterCallback (MMC_HandleTypeDef * hmmc, HAL_MMC_CallbackIDTypeDef CallbackId)

Function description

HAL_MMC_ConfigWideBusOperation

Function name

HAL_StatusTypeDef HAL_MMC_ConfigWideBusOperation (MMC_HandleTypeDef * hmmc, uint32_t WideMode)

Function description

HAL_MMC_ConfigSpeedBusOperation

Function name

HAL_StatusTypeDef HAL_MMC_ConfigSpeedBusOperation (MMC_HandleTypeDef * hmmc, uint32_t SpeedMode)

Function description

HAL_MMC_SwitchPartition

Function name

HAL_StatusTypeDef HAL_MMC_SwitchPartition (MMC_HandleTypeDef * hmmc, HAL_MMC_PartitionTypeDef Partition)

Function description

HAL_MMC_GetCardState

Function name

HAL_MMC_CardStateTypeDef HAL_MMC_GetCardState (MMC_HandleTypeDef * hmmc)

Function description

HAL_MMC_GetCardCID

Function name

HAL_StatusTypeDef HAL_MMC_GetCardCID (const MMC_HandleTypeDef * hmmc, HAL_MMC_CardCIDTypeDef * pCID)

Function description

HAL_MMC_GetCardCSD

Function name

HAL_StatusTypeDef HAL_MMC_GetCardCSD (MMC_HandleTypeDef * hmmc, HAL_MMC_CardCSDTypeDef * pCSD)

Function description

HAL_MMC_GetCardInfo

Function name

HAL_StatusTypeDef HAL_MMC_GetCardInfo (const MMC_HandleTypeDef * hmmc, HAL_MMC_CardInfoTypeDef * pCardInfo)

Function description

HAL_MMC_GetCardExtCSD

Function name

HAL_StatusTypeDef HAL_MMC_GetCardExtCSD (MMC_HandleTypeDef * hmmc, uint32_t * pExtCSD, uint32_t Timeout)

Function description

HAL_MMC_GetState

Function name

HAL_MMC_StateTypeDef HAL_MMC_GetState (const MMC_HandleTypeDef * hmmc)

Function description

HAL_MMC_GetError

Function name

uint32_t HAL_MMC_GetError (const MMC_HandleTypeDef * hmmc)

Function description

HAL_MMC_GetRPMBError

Function name

uint32_t HAL_MMC_GetRPMBError (const MMC_HandleTypeDef * hmmc)

Function description

HAL_MMC_Abort

Function name

HAL_StatusTypeDef HAL_MMC_Abort (MMC_HandleTypeDef * hmmc)

Function description

HAL_MMC_Abort_IT

Function name

HAL_StatusTypeDef HAL_MMC_Abort_IT (MMC_HandleTypeDef * hmmc)

Function description

HAL_MMC_EraseSequence

Function name

HAL_StatusTypeDef HAL_MMC_EraseSequence (MMC_HandleTypeDef * hmmc, uint32_t EraseType, uint32_t BlockStartAdd, uint32_t BlockEndAdd)

Function description

HAL_MMC_Sanitize

Function name

HAL_StatusTypeDef HAL_MMC_Sanitize (MMC_HandleTypeDef * hmmc)

Function description

HAL_MMC_ConfigSecRemovalType

Function name

HAL_StatusTypeDef HAL_MMC_ConfigSecRemovalType (MMC_HandleTypeDef * hmmc, uint32_t SRTMode)

Function description

HAL_MMC_GetSupportedSecRemovalType

Function name

HAL_StatusTypeDef HAL_MMC_GetSupportedSecRemovalType (MMC_HandleTypeDef * hmmc, uint32_t * SupportedSRT)

Function description

HAL_MMC_SleepDevice

Function name

HAL_StatusTypeDef HAL_MMC_SleepDevice (MMC_HandleTypeDef * hmmc)

Function description

HAL_MMC_AwakeDevice

Function name

HAL_StatusTypeDef HAL_MMC_AwakeDevice (MMC_HandleTypeDef * hmmc)

Function description

HAL_MMC_RPMB_ProgramAuthenticationKey

Function name

HAL_StatusTypeDef HAL_MMC_RPMB_ProgramAuthenticationKey (MMC_HandleTypeDef * hmmc, const uint8_t * pKey, uint32_t Timeout)

Function description

HAL_MMC_RPMB_ProgramAuthenticationKey_IT

Function name

HAL_StatusTypeDef HAL_MMC_RPMB_ProgramAuthenticationKey_IT (MMC_HandleTypeDef * hmmc, const uint8_t * pKey, uint32_t Timeout)

Function description

HAL_MMC_RPMB_GetWriteCounter

Function name

uint32_t HAL_MMC_RPMB_GetWriteCounter (MMC_HandleTypeDef * hmmc, uint8_t * pNonce, uint32_t Timeout)

Function description

HAL_MMC_RPMB_GetWriteCounter_IT

Function name

uint32_t HAL_MMC_RPMB_GetWriteCounter_IT (MMC_HandleTypeDef * hmmc, uint8_t * pNonce)

Function description

HAL_MMC_RPMB_WriteBlocks

Function name

HAL_StatusTypeDef HAL_MMC_RPMB_WriteBlocks (MMC_HandleTypeDef * hmmc, const uint8_t * pData, uint16_t BlockAdd, uint16_t NumberOfBlocks, const uint8_t * pMAC, uint32_t Timeout)

Function description

HAL_MMC_RPMB_WriteBlocks_IT

Function name

HAL_StatusTypeDef HAL_MMC_RPMB_WriteBlocks_IT (MMC_HandleTypeDef * hmmc, const uint8_t * pData, uint16_t BlockAdd, uint16_t NumberOfBlocks, const uint8_t * pMAC)

Function description

HAL_MMC_RPMB_ReadBlocks

Function name

HAL_StatusTypeDef HAL_MMC_RPMB_ReadBlocks (MMC_HandleTypeDef * hmmc, uint8_t * pData, uint16_t BlockAdd, uint16_t NumberOfBlocks, const uint8_t * pNonce, uint8_t * pMAC, uint32_t Timeout)

Function description

HAL_MMC_RPMB_ReadBlocks_IT

Function name

HAL_StatusTypeDef HAL_MMC_RPMB_ReadBlocks_IT (MMC_HandleTypeDef * hmmc, uint8_t * pData, uint16_t BlockAdd, uint16_t NumberOfBlocks, const uint8_t * pNonce, uint8_t * pMAC)

Function description

47.3 MMC Firmware driver defines

The following section lists the various define and macros of the module.

47.3.1 MMC

MMC

MMC Error status enumeration Structure definition

HAL_MMC_ERROR_NONE

No error

HAL_MMC_ERROR_CMD_CRC_FAIL

Command response received (but CRC check failed)

HAL_MMC_ERROR_DATA_CRC_FAIL

Data block sent/received (CRC check failed)

HAL_MMC_ERROR_CMD_RSP_TIMEOUT

Command response timeout

HAL_MMC_ERROR_DATA_TIMEOUT

Data timeout

HAL_MMC_ERROR_TX_UNDERRUN

Transmit FIFO underrun

HAL_MMC_ERROR_RX_OVERRUN

Receive FIFO overrun

HAL_MMC_ERROR_ADDR_MISALIGNED

Misaligned address

HAL_MMC_ERROR_BLOCK_LEN_ERR

Transferred block length is not allowed for the card or the number of transferred bytes does not match the block length

HAL_MMC_ERROR_ERASE_SEQ_ERR

An error in the sequence of erase command occurs

HAL_MMC_ERROR_BAD_ERASE_PARAM

An invalid selection for erase groups

HAL_MMC_ERROR_WRITE_PROT_VIOLATION

Attempt to program a write protect block

HAL_MMC_ERROR_LOCK_UNLOCK_FAILED

Sequence or password error has been detected in unlock command or if there was an attempt to access a locked card

HAL_MMC_ERROR_COM_CRC_FAILED

CRC check of the previous command failed

HAL_MMC_ERROR_ILLEGAL_CMD

Command is not legal for the card state

HAL_MMC_ERROR_CARD_ECC_FAILED

Card internal ECC was applied but failed to correct the data

HAL_MMC_ERROR_CC_ERR

Internal card controller error

HAL_MMC_ERROR_GENERAL_UNKNOWN_ERR

General or unknown error

HAL_MMC_ERROR_STREAM_READ_UNDERRUN

The card could not sustain data reading in stream rmode

HAL_MMC_ERROR_STREAM_WRITE_OVERRUN

The card could not sustain data programming in stream mode

HAL_MMC_ERROR_CID_CSD_OVERWRITE

CID/CSD overwrite error

HAL_MMC_ERROR_WP_ERASE_SKIP

Only partial address space was erased

HAL_MMC_ERROR_CARD_ECC_DISABLED

Command has been executed without using internal ECC

HAL_MMC_ERROR_ERASE_RESET

Erase sequence was cleared before executing because an out of erase sequence command was received

HAL_MMC_ERROR_AKE_SEQ_ERR

Error in sequence of authentication

HAL_MMC_ERROR_INVALID_VOLTRANGE

Error in case of invalid voltage range

HAL_MMC_ERROR_ADDR_OUT_OF_RANGE

Error when addressed block is out of range

HAL_MMC_ERROR_REQUEST_NOT_APPLICABLE

Error when command request is not applicable

HAL_MMC_ERROR_PARAM

the used parameter is not valid

HAL_MMC_ERROR_UNSUPPORTED_FEATURE

Error when feature is not insupported

HAL_MMC_ERROR_BUSY

Error when transfer process is busy

HAL_MMC_ERROR_DMA

Error while DMA transfer

HAL_MMC_ERROR_TIMEOUT

Timeout error response results after operating with RPMB partition

HAL_MMC_ERROR_RPMB_OPERATION_OK

Operation OK

HAL_MMC_ERROR_RPMB_GENERAL_FAILURE

General failure

HAL_MMC_ERROR_RPMB_AUTHENTICATION_FAILURE

Authentication failure

HAL_MMC_ERROR_RPMB_COUNTER_FAILURE

Counter failure

HAL_MMC_ERROR_RPMB_ADDRESS_FAILURE

Address failure

HAL_MMC_ERROR_RPMB_WRITE_FAILURE

Write failure

HAL_MMC_ERROR_RPMB_READ_FAILURE

Read failure

HAL_MMC_ERROR_RPMB_KEY_NOT_YET_PROG

Authentication Key not yet programmed

HAL_MMC_ERROR_RPMB_COUNTER_EXPIRED

Write Counter has expired i.e. reached its max value

HAL_MMC_ERROR_INVALID_CALLBACK

Invalid callback error

MMC context enumeration

MMC_CONTEXT_NONE

None

MMC_CONTEXT_READ_SINGLE_BLOCK

Read single block operation

MMC_CONTEXT_READ_MULTIPLE_BLOCK

Read multiple blocks operation

MMC_CONTEXT_WRITE_SINGLE_BLOCK

Write single block operation

MMC_CONTEXT_WRITE_MULTIPLE_BLOCK

Write multiple blocks operation

MMC_CONTEXT_IT

Process in Interrupt mode

MMC_CONTEXT_DMA

Process in DMA mode

MMC Voltage mode

MMC_HIGH_VOLTAGE_RANGE

High voltage in byte mode

MMC_DUAL_VOLTAGE_RANGE

Dual voltage in byte mode

MMC_LOW_VOLTAGE_RANGE

Low voltage in byte mode

EMMC_HIGH_VOLTAGE_RANGE

High voltage in sector mode

EMMC_DUAL_VOLTAGE_RANGE

Dual voltage in sector mode

EMMC_LOW_VOLTAGE_RANGE

Low voltage in sector mode

MMC_INVALID_VOLTAGE_RANGE

MMC Memory Cards

MMC_LOW_CAPACITY_CARD

MMC Card Capacity <=2Gbytes

MMC_HIGH_CAPACITY_CARD

MMC Card Capacity >2Gbytes and <2Tbytes

MMC Erase Type

HAL_MMC_ERASE

Erase the erase groups identified by CMD35 & 36

HAL_MMC_TRIM

Erase the write blocks identified by CMD35 & 36

HAL_MMC_DISCARD

Discard the write blocks identified by CMD35 & 36

HAL_MMC_SECURE_ERASE

Perform a secure purge according SRT on the erase groups identified by CMD35 & 36

HAL_MMC_SECURE_TRIM_STEP1

Mark the write blocks identified by CMD35 & 36 for secure erase

HAL_MMC_SECURE_TRIM_STEP2

Perform a secure purge according SRT on the write blocks previously identified

IS_MMC_ERASE_TYPE

MMC Secure Removal Type

HAL_MMC_SRT_ERASE

Information removed by an erase

HAL_MMC_SRT_WRITE_CHAR_ERASE

Information removed by an overwriting with a character followed by an erase

HAL_MMC_SRT_WRITE_CHAR_COMPL_RANDOM

Information removed by an overwriting with a character, its complement then a random character

HAL_MMC_SRT_VENDOR_DEFINED

Information removed using a vendor defined

IS_MMC_SRT_TYPE

MMC Partitions types

HAL_MMC_USER_AREA_PARTITION

User area partition

HAL_MMC_BOOT_PARTITION1

Boot partition 1

HAL_MMC_BOOT_PARTITION2

Boot partition 2

HAL_MMC_RPMB_PARTITION

RPMB partition

Exported Constants

MMC_BLOCKSIZE

Block size is 512 bytes

MMC Exported Macros

__HAL_MMC_RESET_HANDLE_STATE

Description:

- Reset MMC handle state.

Parameters:

- __HANDLE__: MMC Handle.

Return value:

- None

__HAL_MMC_ENABLE_IT

Description:

- Enable the MMC device interrupt.

Parameters:

- __HANDLE__: MMC Handle.
- __INTERRUPT__: specifies the SDMMC interrupt sources to be enabled. This parameter can be one or a combination of the following values:
  - SDMMC_IT_CCRCFAIL: Command response received (CRC check failed) interrupt
  - SDMMC_IT_DCRCFAIL: Data block sent/received (CRC check failed) interrupt
  - SDMMC_IT_CTIMEOUT: Command response timeout interrupt
  - SDMMC_IT_DTIMEOUT: Data timeout interrupt
  - SDMMC_IT_TXUNDERR: Transmit FIFO underrun error interrupt
  - SDMMC_IT_RXOVERR: Received FIFO overrun error interrupt
  - SDMMC_IT_CMDREND: Command response received (CRC check passed) interrupt
  - SDMMC_IT_CMDSENT: Command sent (no response required) interrupt
  - SDMMC_IT_DATAEND: Data end (data counter, DATACOUNT, is zero) interrupt
  - SDMMC_IT_DHOLD: Data transfer Hold interrupt
  - SDMMC_IT_DBCKEND: Data block sent/received (CRC check passed) interrupt
  - SDMMC_IT_DABORT: Data transfer aborted by CMD12 interrupt
  - SDMMC_IT_TXFIFOHE: Transmit FIFO Half Empty interrupt
  - SDMMC_IT_RXFIFOHF: Receive FIFO Half Full interrupt
  - SDMMC_IT_RXFIFOF: Receive FIFO full interrupt
  - SDMMC_IT_TXFIFOE: Transmit FIFO empty interrupt
  - SDMMC_IT_BUSYD0END: End of SDMMC_D0 Busy following a CMD response detected interrupt
  - SDMMC_IT_SDIOIT: SD I/O interrupt received interrupt
  - SDMMC_IT_ACKFAIL: Boot Acknowledgment received interrupt
  - SDMMC_IT_ACKTIMEOUT: Boot Acknowledgment timeout interrupt
  - SDMMC_IT_VSWEND: Voltage switch critical timing section completion interrupt
  - SDMMC_IT_CKSTOP: SDMMC_CK stopped in Voltage switch procedure interrupt
  - SDMMC_IT_IDMABTC: IDMA buffer transfer complete interrupt

Return value:

- None

__HAL_MMC_DISABLE_IT

Description:

- Disable the MMC device interrupt.

Parameters:

- __HANDLE__: MMC Handle.
- __INTERRUPT__: specifies the SDMMC interrupt sources to be disabled. This parameter can be one or a combination of the following values:
  - SDMMC_IT_CCRCFAIL: Command response received (CRC check failed) interrupt
  - SDMMC_IT_DCRCFAIL: Data block sent/received (CRC check failed) interrupt
  - SDMMC_IT_CTIMEOUT: Command response timeout interrupt
  - SDMMC_IT_DTIMEOUT: Data timeout interrupt
  - SDMMC_IT_TXUNDERR: Transmit FIFO underrun error interrupt
  - SDMMC_IT_RXOVERR: Received FIFO overrun error interrupt
  - SDMMC_IT_CMDREND: Command response received (CRC check passed) interrupt
  - SDMMC_IT_CMDSENT: Command sent (no response required) interrupt
  - SDMMC_IT_DATAEND: Data end (data counter, DATACOUNT, is zero) interrupt
  - SDMMC_IT_DHOLD: Data transfer Hold interrupt
  - SDMMC_IT_DBCKEND: Data block sent/received (CRC check passed) interrupt
  - SDMMC_IT_DABORT: Data transfer aborted by CMD12 interrupt
  - SDMMC_IT_TXFIFOHE: Transmit FIFO Half Empty interrupt
  - SDMMC_IT_RXFIFOHF: Receive FIFO Half Full interrupt
  - SDMMC_IT_RXFIFOF: Receive FIFO full interrupt
  - SDMMC_IT_TXFIFOE: Transmit FIFO empty interrupt
  - SDMMC_IT_BUSYD0END: End of SDMMC_D0 Busy following a CMD response detected interrupt
  - SDMMC_IT_SDIOIT: SD I/O interrupt received interrupt
  - SDMMC_IT_ACKFAIL: Boot Acknowledgment received interrupt
  - SDMMC_IT_ACKTIMEOUT: Boot Acknowledgment timeout interrupt
  - SDMMC_IT_VSWEND: Voltage switch critical timing section completion interrupt
  - SDMMC_IT_CKSTOP: SDMMC_CK stopped in Voltage switch procedure interrupt
  - SDMMC_IT_IDMABTC: IDMA buffer transfer complete interrupt

Return value:

- None

__HAL_MMC_GET_FLAG

Description:

- Check whether the specified MMC flag is set or not.

Parameters:

- __HANDLE__: MMC Handle.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - SDMMC_FLAG_CCRCFAIL: Command response received (CRC check failed)
  - SDMMC_FLAG_DCRCFAIL: Data block sent/received (CRC check failed)
  - SDMMC_FLAG_CTIMEOUT: Command response timeout
  - SDMMC_FLAG_DTIMEOUT: Data timeout
  - SDMMC_FLAG_TXUNDERR: Transmit FIFO underrun error
  - SDMMC_FLAG_RXOVERR: Received FIFO overrun error
  - SDMMC_FLAG_CMDREND: Command response received (CRC check passed)
  - SDMMC_FLAG_CMDSENT: Command sent (no response required)
  - SDMMC_FLAG_DATAEND: Data end (data counter, DATACOUNT, is zero)
  - SDMMC_FLAG_DHOLD: Data transfer Hold
  - SDMMC_FLAG_DBCKEND: Data block sent/received (CRC check passed)
  - SDMMC_FLAG_DABORT: Data transfer aborted by CMD12
  - SDMMC_FLAG_DPSMACT: Data path state machine active
  - SDMMC_FLAG_CPSMACT: Command path state machine active
  - SDMMC_FLAG_TXFIFOHE: Transmit FIFO Half Empty
  - SDMMC_FLAG_RXFIFOHF: Receive FIFO Half Full
  - SDMMC_FLAG_TXFIFOF: Transmit FIFO full
  - SDMMC_FLAG_RXFIFOF: Receive FIFO full
  - SDMMC_FLAG_TXFIFOE: Transmit FIFO empty
  - SDMMC_FLAG_RXFIFOE: Receive FIFO empty
  - SDMMC_FLAG_BUSYD0: Inverted value of SDMMC_D0 line (Busy)
  - SDMMC_FLAG_BUSYD0END: End of SDMMC_D0 Busy following a CMD response detected
  - SDMMC_FLAG_SDIOIT: SD I/O interrupt received
  - SDMMC_FLAG_ACKFAIL: Boot Acknowledgment received
  - SDMMC_FLAG_ACKTIMEOUT: Boot Acknowledgment timeout
  - SDMMC_FLAG_VSWEND: Voltage switch critical timing section completion
  - SDMMC_FLAG_CKSTOP: SDMMC_CK stopped in Voltage switch procedure
  - SDMMC_FLAG_IDMATE: IDMA transfer error
  - SDMMC_FLAG_IDMABTC: IDMA buffer transfer complete

Return value:

- The: new state of MMC FLAG (SET or RESET).

__HAL_MMC_CLEAR_FLAG

Description:

- Clear the MMC's pending flags.

Parameters:

- __HANDLE__: MMC Handle.
- __FLAG__: specifies the flag to clear. This parameter can be one or a combination of the following values:
  - SDMMC_FLAG_CCRCFAIL: Command response received (CRC check failed)
  - SDMMC_FLAG_DCRCFAIL: Data block sent/received (CRC check failed)
  - SDMMC_FLAG_CTIMEOUT: Command response timeout
  - SDMMC_FLAG_DTIMEOUT: Data timeout
  - SDMMC_FLAG_TXUNDERR: Transmit FIFO underrun error
  - SDMMC_FLAG_RXOVERR: Received FIFO overrun error
  - SDMMC_FLAG_CMDREND: Command response received (CRC check passed)
  - SDMMC_FLAG_CMDSENT: Command sent (no response required)
  - SDMMC_FLAG_DATAEND: Data end (data counter, DATACOUNT, is zero)
  - SDMMC_FLAG_DHOLD: Data transfer Hold
  - SDMMC_FLAG_DBCKEND: Data block sent/received (CRC check passed)
  - SDMMC_FLAG_DABORT: Data transfer aborted by CMD12
  - SDMMC_FLAG_BUSYD0END: End of SDMMC_D0 Busy following a CMD response detected
  - SDMMC_FLAG_SDIOIT: SD I/O interrupt received
  - SDMMC_FLAG_ACKFAIL: Boot Acknowledgment received
  - SDMMC_FLAG_ACKTIMEOUT: Boot Acknowledgment timeout
  - SDMMC_FLAG_VSWEND: Voltage switch critical timing section completion
  - SDMMC_FLAG_CKSTOP: SDMMC_CK stopped in Voltage switch procedure
  - SDMMC_FLAG_IDMATE: IDMA transfer error
  - SDMMC_FLAG_IDMABTC: IDMA buffer transfer complete

Return value:

- None

__HAL_MMC_GET_IT

Description:

- Check whether the specified MMC interrupt has occurred or not.

Parameters:

- __HANDLE__: MMC Handle.
- __INTERRUPT__: specifies the SDMMC interrupt source to check. This parameter can be one of the following values:
  - SDMMC_IT_CCRCFAIL: Command response received (CRC check failed) interrupt
  - SDMMC_IT_DCRCFAIL: Data block sent/received (CRC check failed) interrupt
  - SDMMC_IT_CTIMEOUT: Command response timeout interrupt
  - SDMMC_IT_DTIMEOUT: Data timeout interrupt
  - SDMMC_IT_TXUNDERR: Transmit FIFO underrun error interrupt
  - SDMMC_IT_RXOVERR: Received FIFO overrun error interrupt
  - SDMMC_IT_CMDREND: Command response received (CRC check passed) interrupt
  - SDMMC_IT_CMDSENT: Command sent (no response required) interrupt
  - SDMMC_IT_DATAEND: Data end (data counter, DATACOUNT, is zero) interrupt
  - SDMMC_IT_DHOLD: Data transfer Hold interrupt
  - SDMMC_IT_DBCKEND: Data block sent/received (CRC check passed) interrupt
  - SDMMC_IT_DABORT: Data transfer aborted by CMD12 interrupt
  - SDMMC_IT_TXFIFOHE: Transmit FIFO Half Empty interrupt
  - SDMMC_IT_RXFIFOHF: Receive FIFO Half Full interrupt
  - SDMMC_IT_RXFIFOF: Receive FIFO full interrupt
  - SDMMC_IT_TXFIFOE: Transmit FIFO empty interrupt
  - SDMMC_IT_BUSYD0END: End of SDMMC_D0 Busy following a CMD response detected interrupt
  - SDMMC_IT_SDIOIT: SD I/O interrupt received interrupt
  - SDMMC_IT_ACKFAIL: Boot Acknowledgment received interrupt
  - SDMMC_IT_ACKTIMEOUT: Boot Acknowledgment timeout interrupt
  - SDMMC_IT_VSWEND: Voltage switch critical timing section completion interrupt
  - SDMMC_IT_CKSTOP: SDMMC_CK stopped in Voltage switch procedure interrupt
  - SDMMC_IT_IDMABTC: IDMA buffer transfer complete interrupt

Return value:

- The: new state of MMC IT (SET or RESET).

__HAL_MMC_CLEAR_IT

Description:

- Clear the MMC's interrupt pending bits.

Parameters:

- __HANDLE__: MMC Handle.
- __INTERRUPT__: specifies the interrupt pending bit to clear. This parameter can be one or a combination of the following values:
  - SDMMC_IT_CCRCFAIL: Command response received (CRC check failed) interrupt
  - SDMMC_IT_DCRCFAIL: Data block sent/received (CRC check failed) interrupt
  - SDMMC_IT_CTIMEOUT: Command response timeout interrupt
  - SDMMC_IT_DTIMEOUT: Data timeout interrupt
  - SDMMC_IT_TXUNDERR: Transmit FIFO underrun error interrupt
  - SDMMC_IT_RXOVERR: Received FIFO overrun error interrupt
  - SDMMC_IT_CMDREND: Command response received (CRC check passed) interrupt
  - SDMMC_IT_CMDSENT: Command sent (no response required) interrupt
  - SDMMC_IT_DATAEND: Data end (data counter, DATACOUNT, is zero) interrupt
  - SDMMC_IT_DHOLD: Data transfer Hold interrupt
  - SDMMC_IT_DBCKEND: Data block sent/received (CRC check passed) interrupt
  - SDMMC_IT_DABORT: Data transfer aborted by CMD12 interrupt
  - SDMMC_IT_TXFIFOHE: Transmit FIFO Half Empty interrupt
  - SDMMC_IT_RXFIFOHF: Receive FIFO Half Full interrupt
  - SDMMC_IT_RXFIFOF: Receive FIFO full interrupt
  - SDMMC_IT_TXFIFOE: Transmit FIFO empty interrupt
  - SDMMC_IT_BUSYD0END: End of SDMMC_D0 Busy following a CMD response detected interrupt
  - SDMMC_IT_SDIOIT: SD I/O interrupt received interrupt
  - SDMMC_IT_ACKFAIL: Boot Acknowledgment received interrupt
  - SDMMC_IT_ACKTIMEOUT: Boot Acknowledgment timeout interrupt
  - SDMMC_IT_VSWEND: Voltage switch critical timing section completion interrupt
  - SDMMC_IT_CKSTOP: SDMMC_CK stopped in Voltage switch procedure interrupt
  - SDMMC_IT_IDMABTC: IDMA buffer transfer complete interrupt

Return value:

- None

MMC Card State enumeration structure

HAL_MMC_CARD_IDLE

Card is in idle state (can't be checked by CMD13)

HAL_MMC_CARD_READY

Card state is ready (can't be checked by CMD13)

HAL_MMC_CARD_IDENTIFICATION

Card is in identification state (can't be checked by CMD13)

HAL_MMC_CARD_STANDBY

Card is in standby state

HAL_MMC_CARD_TRANSFER

Card is in transfer state

HAL_MMC_CARD_SENDING

Card is sending an operation

HAL_MMC_CARD_RECEIVING

Card is receiving operation information

HAL_MMC_CARD_PROGRAMMING

Card is in programming state

HAL_MMC_CARD_DISCONNECTED

Card is disconnected

HAL_MMC_CARD_BUSTEST

Card is in bus test state

HAL_MMC_CARD_SLEEP

Card is in sleep state (can't be checked by CMD13)

HAL_MMC_CARD_ERROR

Card response Error (can't be checked by CMD13)

MMC Handle Structure definition

MMC_InitTypeDef

MMC_TypeDef

48 HAL MMC Extension Driver

48.1 MMCEx Firmware driver API description

The following section lists the various functions of the MMCEx library.

48.1.1 How to use this driver

The MMC Extension HAL driver can be used as follows:

- Configure Buffer0 and Buffer1 start address and Buffer size using HAL_MMCEx_ConfigDMAMultiBuffer() function.
- Start Read and Write for multibuffer mode using HAL_MMCEx_ReadBlocksDMAMultiBuffer() and HAL_MMCEx_WriteBlocksDMAMultiBuffer() functions.

48.1.2 Detailed description of functions

HAL_MMCEx_DMALinkedList_ReadBlocks

Function name

HAL_StatusTypeDef HAL_MMCEx_DMALinkedList_ReadBlocks (MMC_HandleTypeDef * hmmc, const MMC_DMALinkedListTypeDef * pLinkedList, uint32_t BlockAdd, uint32_t NumberOfBlocks)

Function description

HAL_MMCEx_DMALinkedList_WriteBlocks

Function name

HAL_StatusTypeDef HAL_MMCEx_DMALinkedList_WriteBlocks (MMC_HandleTypeDef * hmmc, const MMC_DMALinkedListTypeDef * pLinkedList, uint32_t BlockAdd, uint32_t NumberOfBlocks)

Function description

HAL_MMCEx_DMALinkedList_BuildNode

Function name

HAL_StatusTypeDef HAL_MMCEx_DMALinkedList_BuildNode (MMC_DMALinkNodeTypeDef * pNode, MMC_DMALinkNodeConfTypeDef * pNodeConf)

Function description

HAL_MMCEx_DMALinkedList_InsertNode

Function name

HAL_StatusTypeDef HAL_MMCEx_DMALinkedList_InsertNode (MMC_DMALinkedListTypeDef * pLinkedList, MMC_DMALinkNodeTypeDef * pPrevNode, MMC_DMALinkNodeTypeDef * pNewNode)

Function description

HAL_MMCEx_DMALinkedList_RemoveNode

Function name

HAL_StatusTypeDef HAL_MMCEx_DMALinkedList_RemoveNode (MMC_DMALinkedListTypeDef * pLinkedList, MMC_DMALinkNodeTypeDef * pNode)

Function description

HAL_MMCEx_DMALinkedList_LockNode

Function name

HAL_StatusTypeDef HAL_MMCEx_DMALinkedList_LockNode (MMC_DMALinkNodeTypeDef * pNode)

Function description

HAL_MMCEx_DMALinkedList_UnlockNode

Function name

HAL_StatusTypeDef HAL_MMCEx_DMALinkedList_UnlockNode (MMC_DMALinkNodeTypeDef * pNode)

Function description

HAL_MMCEx_DMALinkedList_EnableCircularMode

Function name

HAL_StatusTypeDef HAL_MMCEx_DMALinkedList_EnableCircularMode (SDMMC_DMALinkedListTypeDef * pLinkedList)

Function description

HAL_MMCEx_DMALinkedList_DisableCircularMode

Function name

HAL_StatusTypeDef HAL_MMCEx_DMALinkedList_DisableCircularMode (SDMMC_DMALinkedListTypeDef * pLinkedList)

Function description

HAL_MMCEx_Read_DMALnkLstBufCpltCallback

Function name

void HAL_MMCEx_Read_DMALnkLstBufCpltCallback (MMC_HandleTypeDef * hmmc)

Function description

HAL_MMCEx_Write_DMALnkLstBufCpltCallback

Function name

void HAL_MMCEx_Write_DMALnkLstBufCpltCallback (MMC_HandleTypeDef * hmmc)

Function description

48.2 MMCEx Firmware driver defines

The following section lists the various define and macros of the module.

48.2.1 MMCEx

MMCEx

Linked List Wrapper

MMC_DMALinkNodeTypeDef

MMC_DMALinkNodeConfTypeDef

MMC_DMALinkedListTypeDef

HAL_MMCx_DMALinkedList_WriteCpltCallback

HAL_MMCx_DMALinkedList_ReadCpltCallback

49 HAL NAND Generic Driver

49.1 NAND Firmware driver registers structures

49.1.1 NAND_IDTypeDef

NAND_IDTypeDef is defined in the stm32n6xx_hal_nand.h

Data Fields

- ***uint8_t Maker_Id***
- ***uint8_t Device_Id***
- ***uint8_t Third_Id***
- ***uint8_t Fourth_Id***

Field Documentation

- ***uint8_t NAND_IDTypeDef::Maker_Id***
- ***uint8_t NAND_IDTypeDef::Device_Id***
- ***uint8_t NAND_IDTypeDef::Third_Id***
- ***uint8_t NAND_IDTypeDef::Fourth_Id***

49.1.2 NAND_AddressTypeDef

NAND_AddressTypeDef is defined in the stm32n6xx_hal_nand.h

Data Fields

- ***uint16_t Page***
- ***uint16_t Plane***
- ***uint16_t Block***

Field Documentation

- ***uint16_t NAND_AddressTypeDef::Page***
  NAND memory Page address
- ***uint16_t NAND_AddressTypeDef::Plane***
  NAND memory Zone address
- ***uint16_t NAND_AddressTypeDef::Block***
  NAND memory Block address

49.1.3 NAND_DeviceConfigTypeDef

NAND_DeviceConfigTypeDef is defined in the stm32n6xx_hal_nand.h

Data Fields

- ***uint32_t PageSize***
- ***uint32_t SpareAreaSize***
- ***uint32_t BlockSize***
- ***uint32_t BlockNbr***
- ***uint32_t PlaneNbr***
- ***uint32_t PlaneSize***
- ***FunctionalState ExtraCommandEnable***

Field Documentation

- ***uint32_t NAND_DeviceConfigTypeDef::PageSize***
  NAND memory page (without spare area) size measured in bytes for 8 bits addressing or words for 16 bits addressing
- ***uint32_t NAND_DeviceConfigTypeDef::SpareAreaSize***
  NAND memory spare area size measured in bytes for 8 bits addressing or words for 16 bits addressing
- ***uint32_t NAND_DeviceConfigTypeDef::BlockSize***
  NAND memory block size measured in number of pages

- ***uint32_t NAND_DeviceConfigTypeDef::BlockNbr***
  NAND memory number of total blocks
- ***uint32_t NAND_DeviceConfigTypeDef::PlaneNbr***
  NAND memory number of planes
- ***uint32_t NAND_DeviceConfigTypeDef::PlaneSize***
  NAND memory zone size measured in number of blocks
- ***FunctionalState NAND_DeviceConfigTypeDef::ExtraCommandEnable***
  NAND extra command needed for Page reading mode. This parameter is mandatory for some NAND parts after the read command (NAND_CMD_AREA_TRUE1) and before DATA reading sequence. This parameter could be ENABLE or DISABLE Please check the Read Mode sequence in the NAND device datasheet

49.1.4 __NAND_HandleTypeDef

__NAND_HandleTypeDef is defined in the stm32n6xx_hal_nand.h

Data Fields

- ***FMC_NAND_TypeDef * Instance***
- ***FMC_NAND_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_NAND_StateTypeDef State***
- ***NAND_DeviceConfigTypeDef Config***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***
- ***void(* ItCallback***

Field Documentation

- ***FMC_NAND_TypeDef* __NAND_HandleTypeDef::Instance***
  Register base address
- ***FMC_NAND_InitTypeDef __NAND_HandleTypeDef::Init***
  NAND device control configuration parameters
- ***HAL_LockTypeDef __NAND_HandleTypeDef::Lock***
  NAND locking object
- ***__IO HAL_NAND_StateTypeDef __NAND_HandleTypeDef::State***
  NAND device access state
- ***NAND_DeviceConfigTypeDef __NAND_HandleTypeDef::Config***
  NAND physical characteristic information structure
- ***void(* __NAND_HandleTypeDef::MspInitCallback)(struct __NAND_HandleTypeDef *hnand)***
  NAND Msp Init callback
- ***void(* __NAND_HandleTypeDef::MspDeInitCallback)(struct __NAND_HandleTypeDef *hnand)***
  NAND Msp DeInit callback
- ***void(* __NAND_HandleTypeDef::ItCallback)(struct __NAND_HandleTypeDef *hnand)***
  NAND IT callback

49.2 NAND Firmware driver API description

The following section lists the various functions of the NAND library.

49.2.1 How to use this driver

This driver is a generic layered driver which contains a set of APIs used to control NAND flash memories. It uses the FMC layer functions to interface with NAND devices. This driver is used as follows:

- NAND flash memory configuration sequence using the function HAL_NAND_Init() with control and timing parameters for both common and attribute spaces.
- Read NAND flash memory maker and device IDs using the function HAL_NAND_Read_ID(). The read information is stored in the NAND_ID_TypeDef structure declared by the function caller.

- Access NAND flash memory by read/write operations using the functions HAL_NAND_Read_Page_8b()/ HAL_NAND_Read_SpareArea_8b(), HAL_NAND_Write_Page_8b()/HAL_NAND_Write_SpareArea_8b(), HAL_NAND_Read_Page_16b()/HAL_NAND_Read_SpareArea_16b(), HAL_NAND_Write_Page_16b()/ HAL_NAND_Write_SpareArea_16b() to read/write page(s)/spare area(s). These functions use specific device information (Block, page size..) predefined by the user in the NAND_DeviceConfigTypeDef structure. The read/write address information is contained by the Nand_Address_Typedef structure passed as parameter.
- Perform NAND flash Reset chip operation using the function HAL_NAND_Reset().
- Perform NAND flash erase block operation using the function HAL_NAND_Erase_Block(). The erase block address information is contained in the Nand_Address_Typedef structure passed as parameter.
- Read the NAND flash status operation using the function HAL_NAND_Read_Status().
- You can also control the NAND device by calling the control APIs HAL_NAND_ECC_Enable()/ HAL_NAND_ECC_Disable() to respectively enable/disable the ECC code correction feature or the function HAL_NAND_GetECC() to get the ECC correction code.
- You can monitor the NAND device HAL state by calling the function HAL_NAND_GetState()

Note: This driver is a set of generic APIs which handle standard NAND flash operations. If a NAND flash device contains different operations and/or implementations, it should be implemented separately.

Callback registration

The compilation define USE_HAL_NAND_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use Functions HAL_NAND_RegisterCallback() to register a user callback, it allows to register following callbacks:

- MspInitCallback : NAND MspInit.
- MspDeInitCallback : NAND MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function. Use function HAL_NAND_UnRegisterCallback() to reset a callback to the default weak (overridden) function. It allows to reset following callbacks:
- MspInitCallback : NAND MspInit.
- MspDeInitCallback : NAND MspDeInit. This function) takes as parameters the HAL peripheral handle and the Callback ID. By default, after the HAL_NAND_Init and if the state is HAL_NAND_STATE_RESET all callbacks are reset to the corresponding legacy weak (overridden) functions. Exception done for MspInit and MspDeInit callbacks that are respectively reset to the legacy weak (overridden) functions in the HAL_NAND_Init and HAL_NAND_DeInit only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the HAL_NAND_Init and HAL_NAND_DeInit keep and use the user MspInit/MspDeInit callbacks (registered beforehand) Callbacks can be registered/unregistered in READY state only. Exception done for MspInit/MspDeInit callbacks that can be registered/unregistered in READY or RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_NAND_RegisterCallback before calling HAL_NAND_DeInit or HAL_NAND_Init function. When The compilation define USE_HAL_NAND_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak (overridden) callbacks are used.

49.2.2 Detailed description of functions

HAL_NAND_Init

Function name

HAL_StatusTypeDef HAL_NAND_Init (NAND_HandleTypeDef * hnand, FMC_NAND_PCC_TimingTypeDef * ComSpace_Timing, FMC_NAND_PCC_TimingTypeDef * AttSpace_Timing)

Function description

HAL_NAND_DeInit

Function name

HAL_StatusTypeDef HAL_NAND_DeInit (NAND_HandleTypeDef * hnand)

Function description

HAL_NAND_ConfigDevice

Function name

HAL_StatusTypeDef HAL_NAND_ConfigDevice (NAND_HandleTypeDef * hnand, const NAND_DeviceConfigTypeDef * pDeviceConfig)

Function description

HAL_NAND_Read_ID

Function name

HAL_StatusTypeDef HAL_NAND_Read_ID (NAND_HandleTypeDef * hnand, NAND_IDTypeDef * pNAND_ID)

Function description

HAL_NAND_MspInit

Function name

void HAL_NAND_MspInit (NAND_HandleTypeDef * hnand)

Function description

HAL_NAND_MspDeInit

Function name

void HAL_NAND_MspDeInit (NAND_HandleTypeDef * hnand)

Function description

HAL_NAND_IRQHandler

Function name

void HAL_NAND_IRQHandler (NAND_HandleTypeDef * hnand)

Function description

HAL_NAND_ITCallback

Function name

void HAL_NAND_ITCallback (NAND_HandleTypeDef * hnand)

Function description

HAL_NAND_Reset

Function name

HAL_StatusTypeDef HAL_NAND_Reset (NAND_HandleTypeDef * hnand)

Function description

HAL_NAND_Read_Page_8b

Function name

HAL_StatusTypeDef HAL_NAND_Read_Page_8b (NAND_HandleTypeDef * hnand, const NAND_AddressTypeDef * pAddress, uint8_t * pBuffer, uint32_t NumPageToRead)

Function description

HAL_NAND_Write_Page_8b

Function name

HAL_StatusTypeDef HAL_NAND_Write_Page_8b (NAND_HandleTypeDef * hnand, const NAND_AddressTypeDef * pAddress, const uint8_t * pBuffer, uint32_t NumPageToWrite)

Function description

HAL_NAND_Read_SpareArea_8b

Function name

HAL_StatusTypeDef HAL_NAND_Read_SpareArea_8b (NAND_HandleTypeDef * hnand, const NAND_AddressTypeDef * pAddress, uint8_t * pBuffer, uint32_t NumSpareAreaToRead)

Function description

HAL_NAND_Write_SpareArea_8b

Function name

HAL_StatusTypeDef HAL_NAND_Write_SpareArea_8b (NAND_HandleTypeDef * hnand, const NAND_AddressTypeDef * pAddress, const uint8_t * pBuffer, uint32_t NumSpareAreaTowrite)

Function description

HAL_NAND_Read_Page_16b

Function name

HAL_StatusTypeDef HAL_NAND_Read_Page_16b (NAND_HandleTypeDef * hnand, const NAND_AddressTypeDef * pAddress, uint16_t * pBuffer, uint32_t NumPageToRead)

Function description

HAL_NAND_Write_Page_16b

Function name

HAL_StatusTypeDef HAL_NAND_Write_Page_16b (NAND_HandleTypeDef * hnand, const NAND_AddressTypeDef * pAddress, const uint16_t * pBuffer, uint32_t NumPageToWrite)

Function description

HAL_NAND_Read_SpareArea_16b

Function name

HAL_StatusTypeDef HAL_NAND_Read_SpareArea_16b (NAND_HandleTypeDef * hnand, const NAND_AddressTypeDef * pAddress, uint16_t * pBuffer, uint32_t NumSpareAreaToRead)

Function description

HAL_NAND_Write_SpareArea_16b

Function name

HAL_StatusTypeDef HAL_NAND_Write_SpareArea_16b (NAND_HandleTypeDef * hnand, const NAND_AddressTypeDef * pAddress, const uint16_t * pBuffer, uint32_t NumSpareAreaTowrite)

Function description

HAL_NAND_Erase_Block

Function name

HAL_StatusTypeDef HAL_NAND_Erase_Block (NAND_HandleTypeDef * hnand, const NAND_AddressTypeDef * pAddress)

Function description

HAL_NAND_Address_Inc

Function name

uint32_t HAL_NAND_Address_Inc (const NAND_HandleTypeDef * hnand, NAND_AddressTypeDef * pAddress)

Function description

HAL_NAND_RegisterCallback

Function name

HAL_StatusTypeDef HAL_NAND_RegisterCallback (NAND_HandleTypeDef * hnand, HAL_NAND_CallbackIDTypeDef CallbackId, pNAND_CallbackTypeDef pCallback)

Function description

HAL_NAND_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_NAND_UnRegisterCallback (NAND_HandleTypeDef * hnand, HAL_NAND_CallbackIDTypeDef CallbackId)

Function description

HAL_NAND_ECC_Enable

Function name

HAL_StatusTypeDef HAL_NAND_ECC_Enable (NAND_HandleTypeDef * hnand)

Function description

HAL_NAND_ECC_Disable

Function name

HAL_StatusTypeDef HAL_NAND_ECC_Disable (NAND_HandleTypeDef * hnand)

Function description

HAL_NAND_GetECC

Function name

HAL_StatusTypeDef HAL_NAND_GetECC (NAND_HandleTypeDef * hnand, uint32_t * ECCval, uint32_t Timeout)

Function description

HAL_NAND_GetState

Function name

HAL_NAND_StateTypeDef HAL_NAND_GetState (const NAND_HandleTypeDef * hnand)

Function description

HAL_NAND_Read_Status

Function name

uint32_t HAL_NAND_Read_Status (const NAND_HandleTypeDef * hnand)

Function description

49.3 NAND Firmware driver defines

The following section lists the various define and macros of the module.

49.3.1 NAND

NAND

NAND Exported Macros

__HAL_NAND_RESET_HANDLE_STATE

Description:

- Reset NAND handle state.

Parameters:

- __HANDLE__: specifies the NAND handle.

Return value:

- None

50 HAL NOR Generic Driver

50.1 NOR Firmware driver registers structures

50.1.1 NOR_IDTypeDef

NOR_IDTypeDef is defined in the stm32n6xx_hal_nor.h

Data Fields

- ***uint16_t Manufacturer_Code***
- ***uint16_t Device_Code1***
- ***uint16_t Device_Code2***
- ***uint16_t Device_Code3***

Field Documentation

- ***uint16_t NOR_IDTypeDef::Manufacturer_Code***
  Defines the device's manufacturer code used to identify the memory
- ***uint16_t NOR_IDTypeDef::Device_Code1***
- ***uint16_t NOR_IDTypeDef::Device_Code2***
- ***uint16_t NOR_IDTypeDef::Device_Code3***
  Defines the device's codes used to identify the memory. These codes can be accessed by performing read operations with specific control signals and addresses set.They can also be accessed by issuing an Auto Select command

50.1.2 NOR_CFITypeDef

NOR_CFITypeDef is defined in the stm32n6xx_hal_nor.h

Data Fields

- ***uint16_t CFI_1***
- ***uint16_t CFI_2***
- ***uint16_t CFI_3***
- ***uint16_t CFI_4***

Field Documentation

- ***uint16_t NOR_CFITypeDef::CFI_1***
  < Defines the information stored in the memory's Common flash interface which contains a description of various electrical and timing parameters, density information and functions supported by the memory
- ***uint16_t NOR_CFITypeDef::CFI_2***
- ***uint16_t NOR_CFITypeDef::CFI_3***
- ***uint16_t NOR_CFITypeDef::CFI_4***

50.1.3 __NOR_HandleTypeDef

__NOR_HandleTypeDef is defined in the stm32n6xx_hal_nor.h

Data Fields

- ***FMC_NORSRAM_TypeDef * Instance***
- ***FMC_NORSRAM_EXTENDED_TypeDef * Extended***
- ***FMC_NORSRAM_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_NOR_StateTypeDef State***
- ***uint32_t CommandSet***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***FMC_NORSRAM_TypeDef* __NOR_HandleTypeDef::Instance***
  Register base address
- ***FMC_NORSRAM_EXTENDED_TypeDef* __NOR_HandleTypeDef::Extended***
  Extended mode register base address
- ***FMC_NORSRAM_InitTypeDef __NOR_HandleTypeDef::Init***
  NOR device control configuration parameters
- ***HAL_LockTypeDef __NOR_HandleTypeDef::Lock***
  NOR locking object
- ***__IO HAL_NOR_StateTypeDef __NOR_HandleTypeDef::State***
  NOR device access state
- ***uint32_t __NOR_HandleTypeDef::CommandSet***
  NOR algorithm command set and control
- ***void(* __NOR_HandleTypeDef::MspInitCallback)(struct __NOR_HandleTypeDef *hnor)***
  NOR Msp Init callback
- ***void(* __NOR_HandleTypeDef::MspDeInitCallback)(struct __NOR_HandleTypeDef *hnor)***
  NOR Msp DeInit callback

50.2 NOR Firmware driver API description

The following section lists the various functions of the NOR library.

50.2.1 How to use this driver

This driver is a generic layered driver which contains a set of APIs used to control NOR flash memories. It uses the FMC layer functions to interface with NOR devices. This driver is used as follows:

- NOR flash memory configuration sequence using the function HAL_NOR_Init() with control and timing parameters for both normal and extended mode.
- Read NOR flash memory manufacturer code and device IDs using the function HAL_NOR_Read_ID(). The read information is stored in the NOR_ID_TypeDef structure declared by the function caller.
- Access NOR flash memory by read/write data unit operations using the functions HAL_NOR_Read(), HAL_NOR_Program().
- Perform NOR flash erase block/chip operations using the functions HAL_NOR_Erase_Block() and HAL_NOR_Erase_Chip().
- Read the NOR flash CFI (common flash interface) IDs using the function HAL_NOR_Read_CFI(). The read information is stored in the NOR_CFI_TypeDef structure declared by the function caller.
- You can also control the NOR device by calling the control APIs HAL_NOR_WriteOperation_Enable()/ HAL_NOR_WriteOperation_Disable() to respectively enable/disable the NOR write operation
- You can monitor the NOR device HAL state by calling the function HAL_NOR_GetState()

Note: This driver is a set of generic APIs which handle standard NOR flash operations. If a NOR flash device contains different operations and/or implementations, it should be implemented separately.

NOR HAL driver macros list

Below the list of most used macros in NOR HAL driver.

- NOR_WRITE : NOR memory write data to specified address

Callback registration

The compilation define USE_HAL_NOR_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use Functions HAL_NOR_RegisterCallback() to register a user callback, it allows to register following callbacks:

- MspInitCallback : NOR MspInit.
- MspDeInitCallback : NOR MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function. Use function HAL_NOR_UnRegisterCallback() to reset a callback to the default weak (overridden) function. It allows to reset following callbacks:
- MspInitCallback : NOR MspInit.

- MspDeInitCallback : NOR MspDeInit. This function) takes as parameters the HAL peripheral handle and the Callback ID. By default, after the HAL_NOR_Init and if the state is HAL_NOR_STATE_RESET all callbacks are reset to the corresponding legacy weak (overridden) functions. Exception done for MspInit and MspDeInit callbacks that are respectively reset to the legacy weak (overridden) functions in the HAL_NOR_Init and HAL_NOR_DeInit only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the HAL_NOR_Init and HAL_NOR_DeInit keep and use the user MspInit/MspDeInit callbacks (registered beforehand) Callbacks can be registered/unregistered in READY state only. Exception done for MspInit/MspDeInit callbacks that can be registered/unregistered in READY or RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_NOR_RegisterCallback before calling HAL_NOR_DeInit or HAL_NOR_Init function. When The compilation define USE_HAL_NOR_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak (overridden) callbacks are used.

50.2.2 Detailed description of functions

HAL_NOR_Init

Function name

HAL_StatusTypeDef HAL_NOR_Init (NOR_HandleTypeDef * hnor, FMC_NORSRAM_TimingTypeDef * Timing, FMC_NORSRAM_TimingTypeDef * ExtTiming)

Function description

HAL_NOR_DeInit

Function name

HAL_StatusTypeDef HAL_NOR_DeInit (NOR_HandleTypeDef * hnor)

Function description

HAL_NOR_MspInit

Function name

void HAL_NOR_MspInit (NOR_HandleTypeDef * hnor)

Function description

HAL_NOR_MspDeInit

Function name

void HAL_NOR_MspDeInit (NOR_HandleTypeDef * hnor)

Function description

HAL_NOR_MspWait

Function name

void HAL_NOR_MspWait (NOR_HandleTypeDef * hnor, uint32_t Timeout)

Function description

HAL_NOR_Read_ID

Function name

HAL_StatusTypeDef HAL_NOR_Read_ID (NOR_HandleTypeDef * hnor, NOR_IDTypeDef * pNOR_ID)

Function description

HAL_NOR_ReturnToReadMode

Function name

HAL_StatusTypeDef HAL_NOR_ReturnToReadMode (NOR_HandleTypeDef * hnor)

Function description

HAL_NOR_Read

Function name

HAL_StatusTypeDef HAL_NOR_Read (NOR_HandleTypeDef * hnor, uint32_t * pAddress, uint16_t * pData)

Function description

HAL_NOR_Program

Function name

HAL_StatusTypeDef HAL_NOR_Program (NOR_HandleTypeDef * hnor, uint32_t * pAddress, uint16_t * pData)

Function description

HAL_NOR_ReadBuffer

Function name

HAL_StatusTypeDef HAL_NOR_ReadBuffer (NOR_HandleTypeDef * hnor, uint32_t uwAddress, uint16_t * pData, uint32_t uwBufferSize)

Function description

HAL_NOR_ProgramBuffer

Function name

HAL_StatusTypeDef HAL_NOR_ProgramBuffer (NOR_HandleTypeDef * hnor, uint32_t uwAddress, uint16_t * pData, uint32_t uwBufferSize)

Function description

HAL_NOR_Erase_Block

Function name

HAL_StatusTypeDef HAL_NOR_Erase_Block (NOR_HandleTypeDef * hnor, uint32_t BlockAddress, uint32_t Address)

Function description

HAL_NOR_Erase_Chip

Function name

HAL_StatusTypeDef HAL_NOR_Erase_Chip (NOR_HandleTypeDef * hnor, uint32_t Address)

Function description

HAL_NOR_Read_CFI

Function name

HAL_StatusTypeDef HAL_NOR_Read_CFI (NOR_HandleTypeDef * hnor, NOR_CFITypeDef * pNOR_CFI)

Function description

HAL_NOR_RegisterCallback

Function name

HAL_StatusTypeDef HAL_NOR_RegisterCallback (NOR_HandleTypeDef * hnor, HAL_NOR_CallbackIDTypeDef CallbackId, pNOR_CallbackTypeDef pCallback)

Function description

HAL_NOR_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_NOR_UnRegisterCallback (NOR_HandleTypeDef * hnor, HAL_NOR_CallbackIDTypeDef CallbackId)

Function description

HAL_NOR_WriteOperation_Enable

Function name

HAL_StatusTypeDef HAL_NOR_WriteOperation_Enable (NOR_HandleTypeDef * hnor)

Function description

HAL_NOR_WriteOperation_Disable

Function name

HAL_StatusTypeDef HAL_NOR_WriteOperation_Disable (NOR_HandleTypeDef * hnor)

Function description

HAL_NOR_GetState

Function name

HAL_NOR_StateTypeDef HAL_NOR_GetState (const NOR_HandleTypeDef * hnor)

Function description

HAL_NOR_GetStatus

Function name

HAL_NOR_StatusTypeDef HAL_NOR_GetStatus (NOR_HandleTypeDef * hnor, uint32_t Address, uint32_t Timeout)

Function description

50.3 NOR Firmware driver defines

The following section lists the various define and macros of the module.

50.3.1 NOR

NOR

NOR Exported Macros

__HAL_NOR_RESET_HANDLE_STATE

Description:

- Reset NOR handle state.

Parameters:

- __HANDLE__: specifies the NOR handle.

Return value:

- None

51 HAL PCD Generic Driver

51.1 PCD Firmware driver registers structures

51.1.1 __PCD_HandleTypeDef

__PCD_HandleTypeDef is defined in the stm32n6xx_hal_pcd.h

Data Fields

- ***PCD_TypeDef * Instance***
- ***PCD_InitTypeDef Init***
- ***__IO uint8_t USB_Address***
- ***PCD_EPTypeDef IN_ep***
- ***PCD_EPTypeDef OUT_ep***
- ***HAL_LockTypeDef Lock***
- ***__IO PCD_StateTypeDef State***
- ***__IO uint32_t ErrorCode***
- ***uint32_t Setup***
- ***PCD_LPM_StateTypeDef LPM_State***
- ***uint32_t BESL***
- ***uint32_t FrameNumber***
- ***uint32_t lpm_active***
- ***uint32_t battery_charging_active***
- ***void * pData***
- ***void(* SOFCallback***
- ***void(* SetupStageCallback***
- ***void(* ResetCallback***
- ***void(* SuspendCallback***
- ***void(* ResumeCallback***
- ***void(* ConnectCallback***
- ***void(* DisconnectCallback***
- ***void(* DataOutStageCallback***
- ***void(* DataInStageCallback***
- ***void(* ISOOUTIncompleteCallback***
- ***void(* ISOINIncompleteCallback***
- ***void(* BCDCallback***
- ***void(* LPMCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***PCD_TypeDef* __PCD_HandleTypeDef::Instance***
  Register base address
- ***PCD_InitTypeDef __PCD_HandleTypeDef::Init***
  PCD required parameters
- ***__IO uint8_t __PCD_HandleTypeDef::USB_Address***
  USB Address
- ***PCD_EPTypeDef __PCD_HandleTypeDef::IN_ep[16]***
  IN endpoint parameters
- ***PCD_EPTypeDef __PCD_HandleTypeDef::OUT_ep[16]***
  OUT endpoint parameters

- ***HAL_LockTypeDef __PCD_HandleTypeDef::Lock***
  PCD peripheral status
- ***__IO PCD_StateTypeDef __PCD_HandleTypeDef::State***
  PCD communication state
- ***__IO uint32_t __PCD_HandleTypeDef::ErrorCode***
  PCD Error code
- ***uint32_t __PCD_HandleTypeDef::Setup[12]***
  Setup packet buffer
- ***PCD_LPM_StateTypeDef __PCD_HandleTypeDef::LPM_State***
  LPM State
- ***uint32_t __PCD_HandleTypeDef::BESL***
- ***uint32_t __PCD_HandleTypeDef::FrameNumber***
  Store Current Frame number
- ***uint32_t __PCD_HandleTypeDef::lpm_active***
  Enable or disable the Link Power Management . This parameter can be set to ENABLE or DISABLE
- ***uint32_t __PCD_HandleTypeDef::battery_charging_active***
  Enable or disable Battery charging. This parameter can be set to ENABLE or DISABLE
- ***void* __PCD_HandleTypeDef::pData***
  Pointer to upper stack Handler
- ***void(* __PCD_HandleTypeDef::SOFCallback)(struct __PCD_HandleTypeDef *hpcd)***
  USB OTG PCD SOF callback
- ***void(* __PCD_HandleTypeDef::SetupStageCallback)(struct __PCD_HandleTypeDef *hpcd)***
  USB OTG PCD Setup Stage callback
- ***void(* __PCD_HandleTypeDef::ResetCallback)(struct __PCD_HandleTypeDef *hpcd)***
  USB OTG PCD Reset callback
- ***void(* __PCD_HandleTypeDef::SuspendCallback)(struct __PCD_HandleTypeDef *hpcd)***
  USB OTG PCD Suspend callback
- ***void(* __PCD_HandleTypeDef::ResumeCallback)(struct __PCD_HandleTypeDef *hpcd)***
  USB OTG PCD Resume callback
- ***void(* __PCD_HandleTypeDef::ConnectCallback)(struct __PCD_HandleTypeDef *hpcd)***
  USB OTG PCD Connect callback
- ***void(* __PCD_HandleTypeDef::DisconnectCallback)(struct __PCD_HandleTypeDef *hpcd)***
  USB OTG PCD Disconnect callback
- ***void(* __PCD_HandleTypeDef::DataOutStageCallback)(struct __PCD_HandleTypeDef *hpcd, uint8_t epnum)***
  USB OTG PCD Data OUT Stage callback
- ***void(* __PCD_HandleTypeDef::DataInStageCallback)(struct __PCD_HandleTypeDef *hpcd, uint8_t epnum)***
  USB OTG PCD Data IN Stage callback
- ***void(* __PCD_HandleTypeDef::ISOOUTIncompleteCallback)(struct __PCD_HandleTypeDef *hpcd, uint8_t epnum)***
  USB OTG PCD ISO OUT Incomplete callback
- ***void(* __PCD_HandleTypeDef::ISOINIncompleteCallback)(struct __PCD_HandleTypeDef *hpcd, uint8_t epnum)***
  USB OTG PCD ISO IN Incomplete callback
- ***void(* __PCD_HandleTypeDef::BCDCallback)(struct __PCD_HandleTypeDef *hpcd, PCD_BCD_MsgTypeDef msg)***
  USB OTG PCD BCD callback
- ***void(* __PCD_HandleTypeDef::LPMCallback)(struct __PCD_HandleTypeDef *hpcd, PCD_LPM_MsgTypeDef msg)***
  USB OTG PCD LPM callback
- ***void(* __PCD_HandleTypeDef::MspInitCallback)(struct __PCD_HandleTypeDef *hpcd)***
  USB OTG PCD Msp Init callback
- ***void(* __PCD_HandleTypeDef::MspDeInitCallback)(struct __PCD_HandleTypeDef *hpcd)***
  USB OTG PCD Msp DeInit callback

51.2 PCD Firmware driver API description

The following section lists the various functions of the PCD library.

51.2.1 How to use this driver

The PCD HAL driver can be used as follows:

1. Declare a PCD_HandleTypeDef handle structure, for example: PCD_HandleTypeDef hpcd;
2. Fill parameters of Init structure in HCD handle
3. Call HAL_PCD_Init() API to initialize the PCD peripheral (Core, Device core, ...)
4. Initialize the PCD low level resources through the HAL_PCD_MspInit() API:
   a. Enable the PCD/USB Low Level interface clock using
      - __HAL_RCC_USB_OTG_FS_CLK_ENABLE();
      - __HAL_RCC_USB_OTG_HS_CLK_ENABLE(); (For High Speed Mode)
   b. Initialize the related GPIO clocks
   c. Configure PCD pin-out
   d. Configure PCD NVIC interrupt
5. Associate the Upper USB device stack to the HAL PCD Driver:
   a. hpcd.pData = pdev;
6. Enable PCD transmission and reception:
   a. HAL_PCD_Start();

51.2.2 Detailed description of functions

HAL_PCD_Init

Function name

HAL_StatusTypeDef HAL_PCD_Init (PCD_HandleTypeDef * hpcd)

Function description

HAL_PCD_DeInit

Function name

HAL_StatusTypeDef HAL_PCD_DeInit (PCD_HandleTypeDef * hpcd)

Function description

HAL_PCD_MspInit

Function name

void HAL_PCD_MspInit (PCD_HandleTypeDef * hpcd)

Function description

HAL_PCD_MspDeInit

Function name

void HAL_PCD_MspDeInit (PCD_HandleTypeDef * hpcd)

Function description

HAL_PCD_RegisterCallback

Function name

HAL_StatusTypeDef HAL_PCD_RegisterCallback (PCD_HandleTypeDef * hpcd, HAL_PCD_CallbackIDTypeDef CallbackID, pPCD_CallbackTypeDef pCallback)

Function description

HAL_PCD_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_PCD_UnRegisterCallback (PCD_HandleTypeDef * hpcd, HAL_PCD_CallbackIDTypeDef CallbackID)

Function description

HAL_PCD_RegisterDataOutStageCallback

Function name

HAL_StatusTypeDef HAL_PCD_RegisterDataOutStageCallback (PCD_HandleTypeDef * hpcd, pPCD_DataOutStageCallbackTypeDef pCallback)

Function description

HAL_PCD_UnRegisterDataOutStageCallback

Function name

HAL_StatusTypeDef HAL_PCD_UnRegisterDataOutStageCallback (PCD_HandleTypeDef * hpcd)

Function description

HAL_PCD_RegisterDataInStageCallback

Function name

HAL_StatusTypeDef HAL_PCD_RegisterDataInStageCallback (PCD_HandleTypeDef * hpcd, pPCD_DataInStageCallbackTypeDef pCallback)

Function description

HAL_PCD_UnRegisterDataInStageCallback

Function name

HAL_StatusTypeDef HAL_PCD_UnRegisterDataInStageCallback (PCD_HandleTypeDef * hpcd)

Function description

HAL_PCD_RegisterIsoOutIncpltCallback

Function name

HAL_StatusTypeDef HAL_PCD_RegisterIsoOutIncpltCallback (PCD_HandleTypeDef * hpcd, pPCD_IsoOutIncpltCallbackTypeDef pCallback)

Function description

HAL_PCD_UnRegisterIsoOutIncpltCallback

Function name

HAL_StatusTypeDef HAL_PCD_UnRegisterIsoOutIncpltCallback (PCD_HandleTypeDef * hpcd)

Function description

HAL_PCD_RegisterIsoInIncpltCallback

Function name

HAL_StatusTypeDef HAL_PCD_RegisterIsoInIncpltCallback (PCD_HandleTypeDef * hpcd, pPCD_IsoInIncpltCallbackTypeDef pCallback)

Function description

HAL_PCD_UnRegisterIsoInIncpltCallback

Function name

HAL_StatusTypeDef HAL_PCD_UnRegisterIsoInIncpltCallback (PCD_HandleTypeDef * hpcd)

Function description

HAL_PCD_RegisterBcdCallback

Function name

HAL_StatusTypeDef HAL_PCD_RegisterBcdCallback (PCD_HandleTypeDef * hpcd, pPCD_BcdCallbackTypeDef pCallback)

Function description

HAL_PCD_UnRegisterBcdCallback

Function name

HAL_StatusTypeDef HAL_PCD_UnRegisterBcdCallback (PCD_HandleTypeDef * hpcd)

Function description

HAL_PCD_RegisterLpmCallback

Function name

HAL_StatusTypeDef HAL_PCD_RegisterLpmCallback (PCD_HandleTypeDef * hpcd, pPCD_LpmCallbackTypeDef pCallback)

Function description

HAL_PCD_UnRegisterLpmCallback

Function name

HAL_StatusTypeDef HAL_PCD_UnRegisterLpmCallback (PCD_HandleTypeDef * hpcd)

Function description

HAL_PCD_Start

Function name

HAL_StatusTypeDef HAL_PCD_Start (PCD_HandleTypeDef * hpcd)

Function description

HAL_PCD_Stop

Function name

HAL_StatusTypeDef HAL_PCD_Stop (PCD_HandleTypeDef * hpcd)

Function description

HAL_PCD_IRQHandler

Function name

void HAL_PCD_IRQHandler (PCD_HandleTypeDef * hpcd)

Function description

HAL_PCD_SOFCallback

Function name

void HAL_PCD_SOFCallback (PCD_HandleTypeDef * hpcd)

Function description

HAL_PCD_SetupStageCallback

Function name

void HAL_PCD_SetupStageCallback (PCD_HandleTypeDef * hpcd)

Function description

HAL_PCD_ResetCallback

Function name

void HAL_PCD_ResetCallback (PCD_HandleTypeDef * hpcd)

Function description

HAL_PCD_SuspendCallback

Function name

void HAL_PCD_SuspendCallback (PCD_HandleTypeDef * hpcd)

Function description

HAL_PCD_ResumeCallback

Function name

void HAL_PCD_ResumeCallback (PCD_HandleTypeDef * hpcd)

Function description

HAL_PCD_ConnectCallback

Function name

void HAL_PCD_ConnectCallback (PCD_HandleTypeDef * hpcd)

Function description

HAL_PCD_DisconnectCallback

Function name

void HAL_PCD_DisconnectCallback (PCD_HandleTypeDef * hpcd)

Function description

HAL_PCD_DataOutStageCallback

Function name

void HAL_PCD_DataOutStageCallback (PCD_HandleTypeDef * hpcd, uint8_t epnum)

Function description

HAL_PCD_DataInStageCallback

Function name

void HAL_PCD_DataInStageCallback (PCD_HandleTypeDef * hpcd, uint8_t epnum)

Function description

HAL_PCD_ISOOUTIncompleteCallback

Function name

void HAL_PCD_ISOOUTIncompleteCallback (PCD_HandleTypeDef * hpcd, uint8_t epnum)

Function description

HAL_PCD_ISOINIncompleteCallback

Function name

void HAL_PCD_ISOINIncompleteCallback (PCD_HandleTypeDef * hpcd, uint8_t epnum)

Function description

HAL_PCD_DevConnect

Function name

HAL_StatusTypeDef HAL_PCD_DevConnect (PCD_HandleTypeDef * hpcd)

Function description

HAL_PCD_DevDisconnect

Function name

HAL_StatusTypeDef HAL_PCD_DevDisconnect (PCD_HandleTypeDef * hpcd)

Function description

HAL_PCD_SetAddress

Function name

HAL_StatusTypeDef HAL_PCD_SetAddress (PCD_HandleTypeDef * hpcd, uint8_t address)

Function description

HAL_PCD_EP_Open

Function name

HAL_StatusTypeDef HAL_PCD_EP_Open (PCD_HandleTypeDef * hpcd, uint8_t ep_addr, uint16_t ep_mps, uint8_t ep_type)

Function description

HAL_PCD_EP_Close

Function name

HAL_StatusTypeDef HAL_PCD_EP_Close (PCD_HandleTypeDef * hpcd, uint8_t ep_addr)

Function description

HAL_PCD_EP_Receive

Function name

HAL_StatusTypeDef HAL_PCD_EP_Receive (PCD_HandleTypeDef * hpcd, uint8_t ep_addr, uint8_t * pBuf, uint32_t len)

Function description

HAL_PCD_EP_Transmit

Function name

HAL_StatusTypeDef HAL_PCD_EP_Transmit (PCD_HandleTypeDef * hpcd, uint8_t ep_addr, uint8_t * pBuf, uint32_t len)

Function description

HAL_PCD_EP_SetStall

Function name

HAL_StatusTypeDef HAL_PCD_EP_SetStall (PCD_HandleTypeDef * hpcd, uint8_t ep_addr)

Function description

HAL_PCD_EP_ClrStall

Function name

HAL_StatusTypeDef HAL_PCD_EP_ClrStall (PCD_HandleTypeDef * hpcd, uint8_t ep_addr)

Function description

HAL_PCD_EP_Flush

Function name

HAL_StatusTypeDef HAL_PCD_EP_Flush (PCD_HandleTypeDef * hpcd, uint8_t ep_addr)

Function description

HAL_PCD_EP_Abort

Function name

HAL_StatusTypeDef HAL_PCD_EP_Abort (PCD_HandleTypeDef * hpcd, uint8_t ep_addr)

Function description

HAL_PCD_ActivateRemoteWakeup

Function name

HAL_StatusTypeDef HAL_PCD_ActivateRemoteWakeup (PCD_HandleTypeDef * hpcd)

Function description

HAL_PCD_DeActivateRemoteWakeup

Function name

HAL_StatusTypeDef HAL_PCD_DeActivateRemoteWakeup (PCD_HandleTypeDef * hpcd)

Function description

HAL_PCD_SetTestMode

Function name

HAL_StatusTypeDef HAL_PCD_SetTestMode (const PCD_HandleTypeDef * hpcd, uint8_t testmode)

Function description

HAL_PCD_EP_GetRxCount

Function name

uint32_t HAL_PCD_EP_GetRxCount (PCD_HandleTypeDef const * hpcd, uint8_t ep_addr)

Function description

HAL_PCD_GetState

Function name

PCD_StateTypeDef HAL_PCD_GetState (PCD_HandleTypeDef const * hpcd)

Function description

51.3 PCD Firmware driver defines

The following section lists the various define and macros of the module.

51.3.1 PCD

PCD

PCD Error Code definition

HAL_PCD_ERROR_INVALID_CALLBACK

Invalid Callback error

PCD Exported Macros

__HAL_PCD_ENABLE

__HAL_PCD_DISABLE

__HAL_PCD_GET_FLAG

__HAL_PCD_CLEAR_FLAG

__HAL_PCD_IS_INVALID_INTERRUPT

__HAL_PCD_UNGATE_PHYCLOCK

__HAL_PCD_GATE_PHYCLOCK

__HAL_PCD_IS_PHY_SUSPENDED

PCD PHY Module

PCD_PHY_ULPI

PCD_PHY_EMBEDDED

PCD_PHY_UTMI

PCD Speed

PCD_SPEED_HIGH

PCD_SPEED_HIGH_IN_FULL

PCD_SPEED_FULL

52 HAL PCD Extension Driver

52.1 PCDEx Firmware driver API description

The following section lists the various functions of the PCDEx library.

52.1.1 Detailed description of functions

HAL_PCDEx_SetTxFiFo

Function name

HAL_StatusTypeDef HAL_PCDEx_SetTxFiFo (PCD_HandleTypeDef * hpcd, uint8_t fifo, uint16_t size)

Function description

HAL_PCDEx_SetRxFiFo

Function name

HAL_StatusTypeDef HAL_PCDEx_SetRxFiFo (PCD_HandleTypeDef * hpcd, uint16_t size)

Function description

HAL_PCDEx_ActivateLPM

Function name

HAL_StatusTypeDef HAL_PCDEx_ActivateLPM (PCD_HandleTypeDef * hpcd)

Function description

HAL_PCDEx_DeActivateLPM

Function name

HAL_StatusTypeDef HAL_PCDEx_DeActivateLPM (PCD_HandleTypeDef * hpcd)

Function description

HAL_PCDEx_ActivateBCD

Function name

HAL_StatusTypeDef HAL_PCDEx_ActivateBCD (PCD_HandleTypeDef * hpcd)

Function description

HAL_PCDEx_DeActivateBCD

Function name

HAL_StatusTypeDef HAL_PCDEx_DeActivateBCD (PCD_HandleTypeDef * hpcd)

Function description

HAL_PCDEx_BCD_VBUSDetect

Function name

void HAL_PCDEx_BCD_VBUSDetect (PCD_HandleTypeDef * hpcd)

Function description

HAL_PCDEx_LPM_Callback

Function name

void HAL_PCDEx_LPM_Callback (PCD_HandleTypeDef * hpcd, PCD_LPM_MsgTypeDef msg)

Function description

HAL_PCDEx_BCD_Callback

Function name

void HAL_PCDEx_BCD_Callback (PCD_HandleTypeDef * hpcd, PCD_BCD_MsgTypeDef msg)

Function description

53 HAL PSSI Generic Driver

53.1 PSSI Firmware driver registers structures

53.1.1 PSSI_InitTypeDef

PSSI_InitTypeDef is defined in the stm32n6xx_hal_pssi.h

Data Fields

- ***uint32_t DataWidth***
- ***uint32_t BusWidth***
- ***uint32_t ControlSignal***
- ***uint32_t ClockPolarity***
- ***uint32_t DataEnablePolarity***
- ***uint32_t ReadyPolarity***

Field Documentation

- ***uint32_t PSSI_InitTypeDef::DataWidth***
- ***uint32_t PSSI_InitTypeDef::BusWidth***
- ***uint32_t PSSI_InitTypeDef::ControlSignal***
- ***uint32_t PSSI_InitTypeDef::ClockPolarity***
- ***uint32_t PSSI_InitTypeDef::DataEnablePolarity***
- ***uint32_t PSSI_InitTypeDef::ReadyPolarity***

53.1.2 __PSSI_HandleTypeDef

__PSSI_HandleTypeDef is defined in the stm32n6xx_hal_pssi.h

Data Fields

- ***PSSI_TypeDef * Instance***
- ***PSSI_InitTypeDef Init***
- ***uint32_t * pBuffPtr***
- ***uint32_t XferCount***
- ***uint32_t XferSize***
- ***DMA_HandleTypeDef * hdmatx***
- ***DMA_HandleTypeDef * hdmarx***
- ***void(* TxCpltCallback***
- ***void(* RxCpltCallback***
- ***void(* ErrorCallback***
- ***void(* AbortCpltCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_PSSI_StateTypeDef State***
- ***__IO uint32_t ErrorCode***

Field Documentation

- ***PSSI_TypeDef* __PSSI_HandleTypeDef::Instance***
  PSSI register base address.
- ***PSSI_InitTypeDef __PSSI_HandleTypeDef::Init***
  PSSI Initialization Structure.
- ***uint32_t* __PSSI_HandleTypeDef::pBuffPtr***
  PSSI Data buffer.
- ***uint32_t __PSSI_HandleTypeDef::XferCount***
  PSSI transfer count

- ***uint32_t __PSSI_HandleTypeDef::XferSize***
  PSSI transfer size
- ***DMA_HandleTypeDef* __PSSI_HandleTypeDef::hdmatx***
  PSSI Tx DMA Handle parameters
- ***DMA_HandleTypeDef* __PSSI_HandleTypeDef::hdmarx***
  PSSI Rx DMA Handle parameters
- ***void(* __PSSI_HandleTypeDef::TxCpltCallback)(struct __PSSI_HandleTypeDef *hpssi)***
  PSSI transfer complete callback.
- ***void(* __PSSI_HandleTypeDef::RxCpltCallback)(struct __PSSI_HandleTypeDef *hpssi)***
  PSSI transfer complete callback.
- ***void(* __PSSI_HandleTypeDef::ErrorCallback)(struct __PSSI_HandleTypeDef *hpssi)***
  PSSI transfer complete callback.
- ***void(* __PSSI_HandleTypeDef::AbortCpltCallback)(struct __PSSI_HandleTypeDef *hpssi)***
  PSSI transfer error callback.
- ***void(* __PSSI_HandleTypeDef::MspInitCallback)(struct __PSSI_HandleTypeDef *hpssi)***
  PSSI Msp Init callback.
- ***void(* __PSSI_HandleTypeDef::MspDeInitCallback)(struct __PSSI_HandleTypeDef *hpssi)***
  PSSI Msp DeInit callback.
- ***HAL_LockTypeDef __PSSI_HandleTypeDef::Lock***
  PSSI lock.
- ***__IO HAL_PSSI_StateTypeDef __PSSI_HandleTypeDef::State***
  PSSI transfer state.
- ***__IO uint32_t __PSSI_HandleTypeDef::ErrorCode***
  PSSI error code.

53.2 PSSI Firmware driver API description

The following section lists the various functions of the PSSI library.

53.2.1 How to use this driver

The PSSI HAL driver can be used as follows:

1. Declare a PSSI_HandleTypeDef handle structure, for example: PSSI_HandleTypeDef hpssi;
2. Initialize the PSSI low level resources by implementing the HAL_PSSI_MspInit() API:
   a. Enable the PSSIx interface clock
   b. PSSI pins configuration
      - Enable the clock for the PSSI GPIOs
      - Configure PSSI pins as alternate function open-drain
   c. NVIC configuration if you need to use interrupt process
      - Configure the PSSIx interrupt priority
      - Enable the NVIC PSSI IRQ Channel
   d. DMA Configuration if you need to use DMA process
      - Declare DMA_HandleTypeDef handles structure for the transmit and receive
      - Enable the DMAx interface clock
      - Configure the DMA handle parameters
      - Configure the DMA Tx and Rx
      - Associate the initialized DMA handle to the hpssi DMA Tx and Rx handle
      - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx and Rx
3. Configure the Communication Bus Width, Control Signals, Input Polarity and Output Polarity in the hpssi Init structure.
4. Initialize the PSSI registers by calling HAL_PSSI_Init(), configure also the low level Hardware (GPIO, CLOCK, NVIC...etc) by calling the customized HAL_PSSI_MspInit(&hpssi) API.
5. Configure the PSSI clock as internal or external by calling HAL_PSSI_ClockConfig().
6. For PSSI IO operations, two operation modes are available within this driver :

Polling mode IO operation

- Transmit an amount of data by byte in blocking mode using HAL_PSSI_Transmit()
- Receive an amount of data by byte in blocking mode using HAL_PSSI_Receive()

DMA mode IO operation

- Transmit an amount of data in non-blocking mode (DMA) using HAL_PSSI_Transmit_DMA()
- At transmission end of transfer, HAL_PSSI_TxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_PSSI_TxCpltCallback()
- Receive an amount of data in non-blocking mode (DMA) using HAL_PSSI_Receive_DMA()
- At reception end of transfer, HAL_PSSI_RxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_PSSI_RxCpltCallback()
- In case of transfer Error, HAL_PSSI_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_PSSI_ErrorCallback()
- Abort a PSSI process communication with Interrupt using HAL_PSSI_Abort_IT()
- End of abort process, HAL_PSSI_AbortCpltCallback() is executed and user can add his own code by customization of function pointer HAL_PSSI_AbortCpltCallback()

PSSI HAL driver macros list

Below the list of most used macros in PSSI HAL driver.

- HAL_PSSI_ENABLE : Enable the PSSI peripheral
- HAL_PSSI_DISABLE : Disable the PSSI peripheral
- HAL_PSSI_GET_FLAG : Check whether the specified PSSI flag is set or not
- HAL_PSSI_CLEAR_FLAG : Clear the specified PSSI pending flag
- HAL_PSSI_ENABLE_IT : Enable the specified PSSI interrupt
- HAL_PSSI_DISABLE_IT : Disable the specified PSSI interrupt

Callback registration

Note: You can refer to the PSSI HAL driver header file for more useful macros

53.2.2 Detailed description of functions

HAL_PSSI_Init

Function name

HAL_StatusTypeDef HAL_PSSI_Init (PSSI_HandleTypeDef * hpssi)

Function description

HAL_PSSI_DeInit

Function name

HAL_StatusTypeDef HAL_PSSI_DeInit (PSSI_HandleTypeDef * hpssi)

Function description

HAL_PSSI_MspInit

Function name

void HAL_PSSI_MspInit (PSSI_HandleTypeDef * hpssi)

Function description

HAL_PSSI_MspDeInit

Function name

void HAL_PSSI_MspDeInit (PSSI_HandleTypeDef * hpssi)

Function description

HAL_PSSI_RegisterCallback

Function name

HAL_StatusTypeDef HAL_PSSI_RegisterCallback (PSSI_HandleTypeDef * hpssi, HAL_PSSI_CallbackIDTypeDef CallbackID, pPSSI_CallbackTypeDef pCallback)

Function description

HAL_PSSI_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_PSSI_UnRegisterCallback (PSSI_HandleTypeDef * hpssi, HAL_PSSI_CallbackIDTypeDef CallbackID)

Function description

HAL_PSSI_Transmit

Function name

HAL_StatusTypeDef HAL_PSSI_Transmit (PSSI_HandleTypeDef * hpssi, uint8_t * pData, uint32_t Size, uint32_t Timeout)

Function description

HAL_PSSI_Receive

Function name

HAL_StatusTypeDef HAL_PSSI_Receive (PSSI_HandleTypeDef * hpssi, uint8_t * pData, uint32_t Size, uint32_t Timeout)

Function description

HAL_PSSI_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_PSSI_Transmit_DMA (PSSI_HandleTypeDef * hpssi, uint32_t * pData, uint32_t Size)

Function description

HAL_PSSI_Receive_DMA

Function name

HAL_StatusTypeDef HAL_PSSI_Receive_DMA (PSSI_HandleTypeDef * hpssi, uint32_t * pData, uint32_t Size)

Function description

HAL_PSSI_Abort_DMA

Function name

HAL_StatusTypeDef HAL_PSSI_Abort_DMA (PSSI_HandleTypeDef * hpssi)

Function description

HAL_PSSI_GetState

Function name

HAL_PSSI_StateTypeDef HAL_PSSI_GetState (const PSSI_HandleTypeDef * hpssi)

Function description

HAL_PSSI_GetError

Function name

uint32_t HAL_PSSI_GetError (const PSSI_HandleTypeDef * hpssi)

Function description

HAL_PSSI_ClockConfig

Function name

HAL_StatusTypeDef HAL_PSSI_ClockConfig (PSSI_HandleTypeDef * hpssi, uint32_t ClockSource)

Function description

HAL_PSSI_IRQHandler

Function name

void HAL_PSSI_IRQHandler (PSSI_HandleTypeDef * hpssi)

Function description

HAL_PSSI_TxCpltCallback

Function name

void HAL_PSSI_TxCpltCallback (PSSI_HandleTypeDef * hpssi)

Function description

HAL_PSSI_RxCpltCallback

Function name

void HAL_PSSI_RxCpltCallback (PSSI_HandleTypeDef * hpssi)

Function description

HAL_PSSI_ErrorCallback

Function name

void HAL_PSSI_ErrorCallback (PSSI_HandleTypeDef * hpssi)

Function description

HAL_PSSI_AbortCpltCallback

Function name

void HAL_PSSI_AbortCpltCallback (PSSI_HandleTypeDef * hpssi)

Function description

53.3 PSSI Firmware driver defines

The following section lists the various define and macros of the module.

53.3.1 PSSI

PSSI

PSSI Bus Width

HAL_PSSI_8LINES

8 data lines

HAL_PSSI_16LINES

16 data lines

PSSI Data Width

HAL_PSSI_8BITS

8 Bits

HAL_PSSI_16BITS

16 Bits

HAL_PSSI_32BITS

32 Bits

PSSI definitions

PSSI_MAX_NBYTE_SIZE

PSSI_TIMEOUT_TRANSMIT

Timeout Value

PSSI_CR_OUTEN_INPUT

Input Mode

PSSI_CR_OUTEN_OUTPUT

Output Mode

PSSI_CR_DMA_ENABLE

DMA Mode Enable

PSSI_CR_DMA_DISABLE

DMA Mode Disable

PSSI_CR_16BITS

16 Lines Mode

PSSI_CR_8BITS

8 Lines Mode

PSSI_FLAG_RTT1B

1 Byte Fifo Flag

PSSI_FLAG_RTT4B

4 Bytes Fifo Flag

PSSI Error Code

HAL_PSSI_ERROR_NONE

No error

HAL_PSSI_ERROR_NOT_SUPPORTED

Not supported operation

HAL_PSSI_ERROR_UNDER_RUN

FIFO Under-run error

HAL_PSSI_ERROR_OVER_RUN

FIFO Over-run error

HAL_PSSI_ERROR_DMA

Dma error

HAL_PSSI_ERROR_TIMEOUT

Timeout error

HAL_PSSI_ERROR_INVALID_CALLBACK

Invalid callback error

PSSI Exported Macros

HAL_PSSI_RESET_HANDLE_STATE

Description:

- Reset PSSI handle state.

Parameters:

- __HANDLE__: specifies the PSSI handle.

Return value:

- None

HAL_PSSI_ENABLE

Description:

- Enable the PSSI.

Parameters:

- __HANDLE__: PSSI handle

Return value:

- None.

HAL_PSSI_DISABLE

Description:

- Disable the PSSI.

Parameters:

- __HANDLE__: PSSI handle

Return value:

- None.

HAL_PSSI_GET_STATUS

Description:

- Get the PSSI pending flags.

Parameters:

- __HANDLE__: PSSI handle
- __FLAG__: flag to check. This parameter can be any combination of the following values:
  - PSSI_FLAG_RTT1B: FIFO is ready to transfer one byte
  - PSSI_FLAG_RTT4B: FIFO is ready to transfer four bytes

Return value:

- The: state of FLAG.

HAL_PSSI_GET_FLAG

Description:

- Get the PSSI pending flags.

Parameters:

- __HANDLE__: PSSI handle
- __FLAG__: flag to check. This parameter can be any combination of the following values:
  - PSSI_FLAG_OVR_RIS: Data Buffer overrun/underrun error flag

Return value:

- The: state of FLAG.

HAL_PSSI_CLEAR_FLAG

Description:

- Clear the PSSI pending flags.

Parameters:

- __HANDLE__: PSSI handle
- __FLAG__: specifies the flag to clear. This parameter can be any combination of the following values:
  - PSSI_FLAG_OVR_RIS: Data Buffer overrun/underrun error flag

Return value:

- None

HAL_PSSI_ENABLE_IT

Description:

- Enable the specified PSSI interrupts.

Parameters:

- __HANDLE__: PSSI handle
- __INTERRUPT__: specifies the PSSI interrupt sources to be enabled. This parameter can be any combination of the following values:
  - PSSI_FLAG_OVR_RIS: Configuration error mask

Return value:

- None

HAL_PSSI_DISABLE_IT

Description:

- Disable the specified PSSI interrupts.

Parameters:

- __HANDLE__: PSSI handle
- __INTERRUPT__: specifies the PSSI interrupt sources to be disabled. This parameter can be any combination of the following values:
  - PSSI_IT_OVR_IE: Configuration error mask

Return value:

- None

HAL_PSSI_GET_IT_SOURCE

Description:

- Check whether the specified PSSI interrupt source is enabled or not.

Parameters:

- __HANDLE__: PSSI handle
- __INTERRUPT__: specifies the PSSI interrupt source to check. This parameter can be one of the following values:
  - PSSI_IT_OVR_IE: Data Buffer overrun/underrun error interrupt mask

Return value:

- The: state of INTERRUPT source.

IS_PSSI_CONTROL_SIGNAL

Description:

- Check whether the PSSI Control signal is valid.

Parameters:

- __CONTROL__: Control signals configuration

Return value:

- Valid: or not.

IS_PSSI_BUSWIDTH

Description:

- Check whether the PSSI Bus Width is valid.

Parameters:

- __BUSWIDTH__: PSSI Bush width

Return value:

- Valid: or not.

IS_PSSI_CLOCK_POLARITY

Description:

- Check whether the PSSI Clock Polarity is valid.

Parameters:

- __CLOCKPOL__: PSSI Clock Polarity

Return value:

- Valid: or not.

IS_PSSI_DE_POLARITY

Description:

- Check whether the PSSI Data Enable Polarity is valid.

Parameters:

- __DEPOL__: PSSI DE Polarity

Return value:

- Valid: or not.

IS_PSSI_RDY_POLARITY

Description:

- Check whether the PSSI Ready Polarity is valid.

Parameters:

- __RDYPOL__: PSSI RDY Polarity

Return value:

- Valid: or not.

IS_PSSI_CLOCK_SOURCE

Description:

- Check whether the PSSI Clock source is valid.

Parameters:

- __CLOCKSRC__: PSSI Clock Source

Return value:

- Valid: or not.

PSSI Interrupts

PSSI_FLAG_OVR_RIS

Overrun, Underrun errors flag

PSSI_FLAG_MASK

Overrun, Underrun errors Mask

PSSI_FLAG_OVR_MIS

Overrun, Underrun masked errors flag

PSSI mode

HAL_PSSI_UNIDIRECTIONAL

Uni-directional mode

HAL_PSSI_BIDIRECTIONAL

Bi-directional mode

54 HAL PWR Generic Driver

54.1 PWR Firmware driver registers structures

54.1.1 PWR_PVDTypeDef

PWR_PVDTypeDef is defined in the stm32n6xx_hal_pwr.h

Data Fields

- ***uint32_t Mode***

Field Documentation

- ***uint32_t PWR_PVDTypeDef::Mode***
  Specifies the operating mode for the selected pins. This parameter can be a value of ***PWR_PVD_Mode***.

54.2 PWR Firmware driver API description

The following section lists the various functions of the PWR library.

54.2.1 PWR peripheral overview

1. The Power control (PWR) provides an overview of the supply architecture for the different power domains and of the supply configuration controller.
2. Several low-power modes are available to save power when the CPU does not need to execute code :
   – Sleep (CPU clock stopped and still in RUN mode)
   – Stop (System clock stopped)
   – Standby (System powered down)

54.2.2 How to use this driver

1. Call HAL_PWR_EnableBkUpAccess() and HAL_PWR_DisableBkUpAccess() functions to enable/disable access to the backup domain (RCC Backup domain control register RCC_BDCR, RTC registers, TAMP registers, backup registers and backup SRAM).
2. Call HAL_PWR_ConfigPVD() after setting parameters to be configured (event mode and voltage threshold) in order to set up the Programmed Voltage Detector, then use HAL_PWR_EnablePVD() and HAL_PWR_DisablePVD() functions to start and stop the PVD detection.
   – PVD level on PVD_IN is compared to the internal VREFINT level. PVDO flag is available in PWR_CR2 to indicate if the voltage level on PVD_IN is higher or lower than the PVD threshold.
3. Call HAL_PWR_EnableWakeUpPin() and HAL_PWR_DisableWakeUpPin() functions with the right parameter to configure the wake up pin polarity (Low or High), the wake up pin selection and to enable and disable it.
4. Call HAL_PWREx_GetWakeupFlag() and HAL_PWREx_ClearWakeupFlag() functions to manage wake-up flag for the selected pin.
5. Call HAL_PWR_EnterSLEEPMode() function to enter the CPU in Sleep mode. Wake-up from Sleep mode could be following to an event or an interrupt according to low power mode intrinsic request called (__WFI() or __WFE()).
6. Call HAL_PWR_EnterSTOPMode() function to enter the whole system to Stop mode. Wake-up from Stop mode could be following to an event or an interrupt according to low power mode intrinsic request called (__WFI() or __WFE()). (Regulator state on STM32N6 devices is managed internally but regulator parameter is kept for product compatibility).
7. Call HAL_PWR_EnterSTANDBYMode() function to enter the whole system in Standby mode. Wake-up from Standby mode can be following only by an interrupt.
8. Call HAL_PWR_EnableSleepOnExit() and HAL_PWR_DisableSleepOnExit() APIs to enable and disable the Cortex-M55 re-entry in Sleep mode after an interruption handling is over.
9. Call HAL_PWR_EnableSEVOnPend() and HAL_PWR_DisableSEVOnPend() functions to configure the Cortex-M55 to wake-up after any pending event / interrupt even if it's disabled or has insufficient priority to cause exception entry.
10. Call HAL_PWR_WAKEUP_PIN_IRQHandler() function to handle all wake-up pins interrupts.

11. Call HAL_PWR_ConfigAttributes() function to configure PWR item secure and privilege attributes and call HAL_PWR_GetConfigAttributes() function to get the attribute configuration for the selected item.

PWR HAL driver macros list

Below the list of most used macros in PWR HAL driver.

- __HAL_PWR_GET_FLAG() : Get the PWR pending flags.
- __HAL_PWR_CLEAR_FLAG() : Clear the PWR pending flags.

54.2.3 Initialization and De-Initialization Functions

This section provides functions allowing to deinitialize power peripheral.

After system reset, the backup domain (RCC Backup domain control register RCC_BDCR, RTC registers, TAMP registers, backup registers and backup SRAM) is protected against possible unwanted write accesses. The HAL_PWR_EnableBkUpAccess() function enables the access to the backup domain. The HAL_PWR_DisableBkUpAccess() function disables the access to the backup domain.

This section contains the following APIs:

- ***HAL_PWR_DeInit()***
- ***HAL_PWR_EnableBkUpAccess()***
- ***HAL_PWR_DisableBkUpAccess()***

54.2.4 Peripheral Control Functions

This section provides functions allowing to control power peripheral.

PVD configuration

- The PVD is used to monitor the VDD power supply by comparing it to the internal VREFINT.
- A PVDO flag is available to indicate if VDD is higher or lower than the PVD threshold. This event is internally connected to the EXTI line 66 to generate an interrupt if enabled. It is configurable through __HAL_PWR_PVD_EXTI_ENABLE_IT() macro.
- The PVD is stopped in STANDBY mode.

Wake-up pin configuration

- Wake-up pin is used to wake up the system from STANDBY mode. The pin pull is configurable through the WKUPEPR register to be in No-pull, Pull-up and Pull-down. The pin polarity is configurable through the WKUPEPR register to be active on rising or falling edges.
- There are up to four Wake-up pin in the STM32N6 devices family.

Low Power modes configuration

The device present 3 principles low-power modes :

- SLEEP mode : Cortex-M55 is stopped and all PWR domains are remaining active (Powered and Clocked).
- STOP mode : Cortex-M55 is stopped, clocks are stopped and the regulator is running. The Main regulator or the LP regulator could be selected.
- STANDBY mode : All PWR domains enter DSTANDBY mode and the VCORE supply regulator is powered off.

SLEEP mode

- Entry: The SLEEP mode is entered by using the HAL_PWR_EnterSLEEPMode(Regulator, SLEEPEntry) function.
  - PWR_SLEEPENTRY_WFI: enter SLEEP mode with WFI instruction.
  - PWR_SLEEPENTRY_WFE: enter SLEEP mode with WFE instruction.
  - PWR_SLEEPENTRY_WFE_NO_EVT_CLEAR: enter SLEEP mode with WFE instruction and no clear of pending event.

Note: The Regulator parameter is not used for the STM32N6 family and is kept as parameter just to maintain compatibility with the lower power families.

- Exit: Any peripheral interrupt acknowledged by the nested vectored interrupt controller (NVIC) can wake up the device from SLEEP mode.

STOP mode

In system STOP mode, the CPU clock is stopped. All CPU subsystem peripheral clocks are stopped too. The voltage regulator can be configured either in normal or low-power mode. To minimize the consumption in STOP mode, FLASH can be powered off before entering the STOP mode using the HAL_PWREx_EnableFlashPowerDown() function. It can be switched on again by software after exiting the STOP mode using the HAL_PWREx_DisableFlashPowerDown() function.

- Entry: The STOP mode is entered using the HAL_PWR_EnterSTOPMode(Regulator, STOPEntry) function with:
  - Regulator:
    - PWR_MAINREGULATOR_ON: Main regulator ON. This parameter is not used for the STM32N6 family and is kept as parameter just to maintain compatibility with the lower power families.
  - STOPEntry:
    - PWR_STOPENTRY_WFI: enter STOP mode with WFI instruction.
    - PWR_STOPENTRY_WFE: enter STOP mode with WFE instruction.
    - PWR_STOPENTRY_WFE_NO_EVT_CLEAR: enter STOP mode with WFE instruction and no clear of pending event.
- Exit: Any EXTI Line (Internal or External) configured in Interrupt/Event mode.

STANDBY mode

- The system STANDBY mode allows to achieve the lowest power consumption. It is based on the Cortex-M55 deep SLEEP mode, with the voltage regulator disabled. The system is consequently powered off. The PLL, the HSI oscillator and the HSE oscillator are also switched off. SRAM and register contents are lost except for the RTC registers, RTC backup registers, backup SRAM and standby circuitry.
- The voltage regulator is OFF.
  - Entry:
    - The STANDBY mode is entered using the HAL_PWR_EnterSTANDBYMode() function.
  - Exit:
    - WKUP pin rising or falling edge, RTC alarm (Alarm A and Alarm B), RTC wakeup, tamper event, time stamp event, external reset in NRST pin, IWDG reset.

Auto-wakeup (AWU) from low-power mode

- The MCU can be woken up from low-power mode by an RTC Alarm event, an RTC Wakeup event, a tamper event or a time-stamp event, without depending on an external interrupt (Auto-wakeup mode).
- RTC auto-wakeup (AWU) from the STOP and STANDBY modes
  - To wake up from the STOP mode with an RTC alarm event, it is necessary to configure the RTC to generate the RTC alarm using the HAL_RTC_SetAlarm_IT() function.
  - To wake up from the STOP mode with an RTC Tamper or time stamp event, it is necessary to configure the RTC to detect the tamper or time stamp event using the HAL_RTCEx_SetTimeStamp_IT() or HAL_RTCEx_SetTamper_IT() functions.
  - To wake up from the STOP mode with an RTC WakeUp event, it is necessary to configure the RTC to generate the RTC WakeUp event using the HAL_RTCEx_SetWakeUpTimer_IT() function.

This section contains the following APIs:

- ***HAL_PWR_ConfigPVD()***
- ***HAL_PWR_EnablePVD()***
- ***HAL_PWR_DisablePVD()***
- ***HAL_PWR_EnableWakeUpPin()***
- ***HAL_PWR_DisableWakeUpPin()***
- ***HAL_PWR_GetWakeupFlag()***
- ***HAL_PWR_ClearWakeupFlag()***

- *HAL_PWR_EnterSLEEPMode()*
- *HAL_PWR_EnterSTOPMode()*
- *HAL_PWR_EnterSTANDBYMode()*
- *HAL_PWR_EnableSleepOnExit()*
- *HAL_PWR_DisableSleepOnExit()*
- *HAL_PWR_EnableSEVOnPend()*
- *HAL_PWR_DisableSEVOnPend()*
- *HAL_PWR_PVD_Rising_Callback()*
- *HAL_PWR_PVD_Falling_Callback()*
- *HAL_PWR_WAKEUP_PIN_IRQHandler()*
- *HAL_PWR_WKUP1_Callback()*
- *HAL_PWR_WKUP2_Callback()*
- *HAL_PWR_WKUP3_Callback()*
- *HAL_PWR_WKUP4_Callback()*

54.2.5 PWR Attributes Functions

- The PWR is able to protect register bits from being modified by non-secure and unprivileged accesses. The protection can be activated for the following features through PWR_SECCFGR and PWR_PRIVCFGR:
  - System supply configuration.
  - Voltage scaling.
  - Low-power mode.
  - Wake-up (WKUP) pins.
  - Voltage detection and monitoring.
  - VBAT mode.
- A non-secure access to a secure-protected register bit is denied :
  - The secured bits are not written (WI) with a non-secure write access.
  - The secured bits are read as 0 (RAZ) with a non-secure read access.
- After any application reset or system reset, the PWR does not filter any access (default configuration: non-secure, any privileged) until the trusted agent has programmed the security and privileged protection.
- Secure/non-secure access filtering
- To enable the filtering access based on this attribute, the authorized master agent must set SECx in PWR_SECCFGR related to the PWR feature. When a register is configured as secure, read and write operations are only allowed by a secure access. Non-secure read or write accesses are denied (RAZ/WI). An illegal secure access event is generated to the IAC (illegal access controller). There is no bus error generated. When a register is configured as non-secure, read and write operations are allowed by both secure and non-secure accesses.
- Privileged/unprivileged access filtering
- To enable the filtering access based on this attribute, the authorized master agent has to set PRIVx in PWR_PRIVCFGR related to the PWR feature. When a register is configured as privileged, read and write operations are only allowed by a privileged access. Unprivileged read or write accesses are denied (RAZ/WI). An illegal privileged access event is generated to the IAC. There is no bus error generated. When a register is configured as unprivileged, read and write operations are allowed by both privileged and unprivileged accesses.

This section contains the following APIs:

- *HAL_PWR_ConfigAttributes()*
- *HAL_PWR_GetConfigAttributes()*

54.2.6 Detailed description of functions

HAL_PWR_DeInit

Function name

void HAL_PWR_DeInit (void)

Function description

Deinitialize the HAL PWR peripheral registers to their default reset values.

Return values

- **None.:**

Notes

- This functionality is not available in this product. The prototype is kept just to maintain compatibility with other products.

HAL_PWR_EnableBkUpAccess

Function name

void HAL_PWR_EnableBkUpAccess (void)

Function description

Enable access to the backup domain (RCC Backup domain control register RCC_BDCR, RTC registers, TAMP registers, backup registers and backup SRAM).

Return values

- **None.:**

Notes

- After a system reset, the backup domain is protected against possible unwanted write accesses.

HAL_PWR_DisableBkUpAccess

Function name

void HAL_PWR_DisableBkUpAccess (void)

Function description

Disable access to the backup domain (RCC Backup domain control register RCC_BDCR, RTC registers, TAMP registers, backup registers and backup SRAM).

Return values

- **None.:**

HAL_PWR_ConfigPVD

Function name

void HAL_PWR_ConfigPVD (const PWR_PVDTypeDef * pConfigPVD)

Function description

Configure the voltage threshold detected by the Programmed Voltage Detector (PVD).

Parameters

- **pConfigPVD:** : Pointer to a PWR_PVDTypeDef structure that contains the PVD configuration information (EventMode).

Return values

- **None.:**

HAL_PWR_EnablePVD

Function name

void HAL_PWR_EnablePVD (void)

Function description

Enable the programmable voltage detector (PVD).

Return values

- **None.:**

HAL_PWR_DisablePVD

Function name

void HAL_PWR_DisablePVD (void)

Function description

Disable the programmable voltage detector (PVD).

Return values

- **None.:**

HAL_PWR_EnableWakeUpPin

Function name

void HAL_PWR_EnableWakeUpPin (uint32_t WakeUpPinPolarity)

Function description

Enable the WakeUp PINx functionality.

Parameters

- **WakeUpPinPolarity:** : Specifies which Wake-Up pin to enable. This parameter can be one of the following legacy values, which sets the default (rising edge):
  - PWR_WAKEUP_PIN1, PWR_WAKEUP_PIN2, PWR_WAKEUP_PIN3, PWR_WAKEUP_PIN4. or one of the following values where the user can explicitly states the enabled pin and the chosen polarity:
  - PWR_WAKEUP_PIN1_HIGH, PWR_WAKEUP_PIN1_LOW, PWR_WAKEUP_PIN2_HIGH, PWR_WAKEUP_PIN2_LOW, PWR_WAKEUP_PIN3_HIGH, PWR_WAKEUP_PIN3_LOW, PWR_WAKEUP_PIN4_HIGH, PWR_WAKEUP_PIN4_LOW.

Return values

- **None.:**

Notes

- PWR_WAKEUP_PINx and PWR_WAKEUP_PINx_HIGH are equivalent.

HAL_PWR_DisableWakeUpPin

Function name

void HAL_PWR_DisableWakeUpPin (uint32_t WakeUpPinx)

Function description

Disable the WakeUp PINx functionality.

Parameters

- **WakeUpPinx:** : Specifies the Power Wake-Up pin to disable. This parameter can be one of the following values:
  - PWR_WAKEUP_PIN1, PWR_WAKEUP_PIN2, PWR_WAKEUP_PIN3, PWR_WAKEUP_PIN4, PWR_WAKEUP_PIN1_HIGH, PWR_WAKEUP_PIN1_LOW, PWR_WAKEUP_PIN2_HIGH, PWR_WAKEUP_PIN2_LOW, PWR_WAKEUP_PIN3_HIGH, PWR_WAKEUP_PIN3_LOW, PWR_WAKEUP_PIN4_HIGH, PWR_WAKEUP_PIN4_LOW.

Return values

- **None.:**

HAL_PWR_GetWakeupFlag

Function name

uint32_t HAL_PWR_GetWakeupFlag (uint32_t WakeUpFlag)

Function description

Get the Wake-Up Pin pending flags.

Parameters

- **WakeUpFlag:** : Specifies the Wake-Up PIN flag to be checked. This parameter can be one of the following values:
  - PWR_WAKEUP_FLAG1 : Get wakeup event received from PA0.
  - PWR_WAKEUP_FLAG2 : Get wakeup event received from PA2.
  - PWR_WAKEUP_FLAG3 : Get wakeup event received from PC13.
  - PWR_WAKEUP_FLAG4 : Get wakeup event received from PD2.
  - PWR_WAKEUP_FLAG_ALL : Get Wakeup event received from all wake up pins.

Return values

- **The:** Wake-Up pin flag.

HAL_PWR_ClearWakeupFlag

Function name

HAL_StatusTypeDef HAL_PWR_ClearWakeupFlag (uint32_t WakeUpFlag)

Function description

Clear the Wake-Up pin pending flag.

Parameters

- **WakeUpFlag:** Specifies the Wake-Up PIN flag to clear. This parameter can be one of the following values:
  - PWR_WAKEUP_FLAG1 : Clear the wakeup event received from PA0.
  - PWR_WAKEUP_FLAG2 : Clear the wakeup event received from PA2.
  - PWR_WAKEUP_FLAG3 : Clear the wakeup event received from PC13.
  - PWR_WAKEUP_FLAG4 : Clear the wakeup event received from PD2.
  - PWR_WAKEUP_FLAG_ALL : Clear the wakeup events received from all wake up pins.

Return values

- **HAL:** status.

HAL_PWR_EnterSTOPMode

Function name

void HAL_PWR_EnterSTOPMode (uint32_t Regulator, uint8_t STOPEntry)

Function description

Enter the whole system to Stop mode.

Parameters

- **Regulator:** : This parameter is not used for this product family. and is kept as parameter just to maintain compatibility with the lower power families.
- **STOPEntry:** : Specifies if STOP mode in entered with WFI or WFE intrinsic instruction. This parameter can be one of the following values:
  - PWR_STOPENTRY_WFI : Enter STOP mode with WFI instruction.
  - PWR_STOPENTRY_WFE : Enter STOP mode with WFE instruction.
  - PWR_STOPENTRY_WFE_NO_EVT_CLEAR : Enter STOP mode with WFE and no clear of pending event before.

Return values

- **None.:**

Notes

- This API will enter the system in STOP mode
- In System STOP mode, all I/O pins keep the same state as in Run mode.
- When exiting System STOP mode by issuing an interrupt or a wakeup event, the HSI RC oscillator is selected as default system wakeup clock.
- In System STOP mode, when the voltage regulator operates in low power mode, an additional startup delay is incurred when the system is waking up. By keeping the internal regulator ON during STOP mode, the consumption is higher although the startup time is reduced.

HAL_PWR_EnterSLEEPMode

Function name

void HAL_PWR_EnterSLEEPMode (uint32_t Regulator, uint8_t SLEEPEntry)

Function description

Enter the current core in SLEEP mode (CSLEEP).

Parameters

- **Regulator:** : NA in this family project Specifies the regulator state in SLEEP mode.
- **SLEEPEntry:** : Specifies if SLEEP mode is entered with WFI or WFE intrinsic instruction. This parameter can be one of the following values:
  - PWR_SLEEPENTRY_WFI : Enter SLEEP mode with WFI instruction.
  - PWR_SLEEPENTRY_WFE : Enter SLEEP mode with WFE instruction.
  - PWR_SLEEPENTRY_WFE_NO_EVT_CLEAR: Enter SLEEP mode with WFE and no clear of pending event before.

Return values

- **None.:**

Notes

- This parameter is not used for the STM32N6 family and is kept as parameter just to maintain compatibility with the lower power families.
- Ensure to clear pending events before calling this API through HAL_PWREx_ClearPendingEvent() when the SLEEP entry is WFE.

HAL_PWR_EnterSTANDBYMode

Function name

void HAL_PWR_EnterSTANDBYMode (void)

Function description

Enter the whole system to Standby mode.

Return values

- **None.:**

Notes

- The Standby mode allows achieving the lowest power consumption.
- When the system enters in Standby mode, the voltage regulator is disabled. The complete VCORE domain is consequently powered off. The PLLs, HSI oscillator, CSI oscillator, HSI48 and the HSE oscillator are also switched off. SRAM and register contents are lost except for backup domain registers (RTC registers, RTC backup register and backup RAM), and Standby circuitry.
- When the System exit STANDBY mode by issuing an interrupt or a wakeup event, the HSI RC oscillator is selected as system clock.

HAL_PWR_EnableSleepOnExit

Function name

void HAL_PWR_EnableSleepOnExit (void)

Function description

Indicate SLEEP-ON-EXIT feature when returning from handler mode to thread mode.

Return values

- **None.:**

Notes

- Set SLEEPONEXIT bit of SCR register. When this bit is set, the processor re-enters Sleep mode when an interruption handling is over. Setting this bit is useful when the processor is expected to run only on interruptions handling.

HAL_PWR_DisableSleepOnExit

Function name

void HAL_PWR_DisableSleepOnExit (void)

Function description

Disable SLEEP-ON-EXIT feature when returning from handler mode to thread mode.

Return values

- **None.:**

Notes

- Clears SLEEPONEXIT bit of SCR register. When this bit is set, the processor re-enters Sleep mode when an interruption handling is over.

HAL_PWR_EnableSEVOnPend

Function name

void HAL_PWR_EnableSEVOnPend (void)

Function description

Enable CORTEX SEV-ON-PEND feature.

Return values

- **None.:**

Notes

- Sets SEVONPEND bit of SCR register. When this bit is set, any pending event / interrupt even if it's disabled or has insufficient priority to cause exception entry wakes up the Cortex-M55.

HAL_PWR_DisableSEVOnPend

Function name

void HAL_PWR_DisableSEVOnPend (void)

Function description

Disable CORTEX SEVONPEND feature.

Return values

- **None.:**

Notes

- Resets SEVONPEND bit of SCR register. When this bit is reset, only enabled pending causes exception entry wakes up the Cortex-M55.

HAL_PWR_PVD_Rising_Callback

Function name

void HAL_PWR_PVD_Rising_Callback (void)

Function description

PWR PVD interrupt Rising callback.

Return values

- **None:**

HAL_PWR_PVD_Falling_Callback

Function name

void HAL_PWR_PVD_Falling_Callback (void)

Function description

PWR PVD interrupt Falling callback.

Return values

- **None:**

HAL_PWR_WAKEUP_PIN_IRQHandler

Function name

void HAL_PWR_WAKEUP_PIN_IRQHandler (void)

Function description

This function handles the PWR WAKEUP PIN interrupt request.

Return values

- **None.:**

Notes

- This API should be called under the WAKEUP_PIN_IRQHandler().

HAL_PWR_WKUP1_Callback

Function name

void HAL_PWR_WKUP1_Callback (void)

Function description

PWR WKUP1 interrupt callback.

Return values

- **None.:**

HAL_PWR_WKUP2_Callback

Function name

void HAL_PWR_WKUP2_Callback (void)

Function description

PWR WKUP2 interrupt callback.

Return values

- **None.:**

HAL_PWR_WKUP3_Callback

Function name

void HAL_PWR_WKUP3_Callback (void)

Function description

PWR WKUP3 interrupt callback.

Return values

- **None.:**

HAL_PWR_WKUP4_Callback

Function name

void HAL_PWR_WKUP4_Callback (void)

Function description

PWR WKUP4 interrupt callback.

Return values

- **None.:**

HAL_PWR_ConfigAttributes

Function name

void HAL_PWR_ConfigAttributes (uint32_t Item, uint32_t Attributes)

Function description

Configure the PWR item attributes.

Parameters

- **Item:** : Specifies the item(s) to set attributes on. This parameter can be a combination of PWR_ITEMS.
- **Attributes:** : Specifies the available attribute(s). This parameter can be one of PWR_ATTRIBUTES.

Return values

- **None.:**

Notes

- Available attributes are security and privilege protection.
- Security attribute can only be set only by secure access.
- Privilege attribute for secure items can be managed only by a secure privileged access.
- Privilege attribute for nsecure items can be managed by a secure privileged access or by a nsecure privileged access.

HAL_PWR_GetConfigAttributes

Function name

HAL_StatusTypeDef HAL_PWR_GetConfigAttributes (uint32_t Item, uint32_t * pAttributes)

Function description

Get attribute(s) of a PWR item.

Parameters

- **Item:** : Specifies the item(s) to get attributes on. This parameter can be one of PWR_ITEMS.
- **pAttributes:** : Pointer to return attribute(s). Returned value could be on of PWR_ATTRIBUTES.

Return values

- **HAL:** Status.

54.3 PWR Firmware driver defines

The following section lists the various define and macros of the module.

54.3.1 PWR

PWR

PWR Attributes

PWR_ITEM_ATTR_NSEC_PRIV_MASK

NSecure Privilege / NPrivilege attribute item mask

PWR_ITEM_ATTR_SEC_PRIV_MASK

Secure Privilege / NPrivilege attribute item mask

PWR_NSEC_PRIV

NSecure and Privileged attribute

PWR_NSEC_NPRIV

NSecure and NPrivileged attribute

PWR_SEC_PRIV

Secure and Privileged attribute

PWR_SEC_NPRIV

Secure and NPrivileged attribute

PWR Enable WUP Mask

PWR_EWUP_MASK

PWR Exported Macro

__HAL_PWR_GET_FLAG

Description:

- Check whether or not a specific PWR flag is set.

Parameters:

- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - PWR_FLAG_PVDO: This flag is valid only if PVD is enabled by the HAL_PWR_EnablePVD()function. The PVD is stopped by STANDBY mode.
  - PWR_FLAG_ACTVOSRDY: This flag indicates that the regulator voltage scaling output selection is ready.
  - PWR_FLAG_ACTVOS: This flag indicates that the regulator voltage scaling output selection is ready.
  - PWR_FLAG_VOSRDY When an internal regulator is used, this bit indicates that all the features allowed by the selected VOS can be used.
  - PWR_FLAG_VCOREH: This flag indicates if the VDDCORE level is above the high monitoring threshold.
  - PWR_FLAG_VCOREL This flag indicates if the VDDCORE level is above the low monitoring threshold.
  - PWR_FLAG_TEMPH: This flag indicates if the temperature level is above the high monitoring threshold.
  - PWR_FLAG_TEMPL This flag indicates if the temperature level is above the low monitoring threshold.
  - PWR_FLAG_VBATH: This flag indicates if the VBAT level is above the high monitoring threshold.
  - PWR_FLAG_VBATL: This flag indicates if the VBAT level is above the low monitoring threshold.
  - PWR_FLAG_SBF: Standby mode flag.
  - PWR_FLAG_STOPF: STOP mode flag.
  - PWR_FLAG_VDDIO5RDY: This flag indicates if the VDDIO5 is ready.
  - PWR_FLAG_VDDIO4RDY: This flag indicates if the VDDIO4 is ready.
  - PWR_FLAG_VDDIO3RDY: This flag indicates if the VDDIO3 is ready.
  - PWR_FLAG_VDDIO2RDY: This flag indicates if the VDDIO2 is ready.
  - PWR_FLAG_ARDY: This flag indicates if the VDDA18ADC is ready.
  - PWR_FLAG_USB33RDY: This flag indicates if the VDD33USB is ready.
  - PWR_FLAG_WUF1: This flag indicates a wakeup event was received from WKUP1 pin.
  - PWR_FLAG_WUF2: This flag indicates a wakeup event was received from WKUP2 pin.
  - PWR_FLAG_WUF3: This flag indicates a wakeup event was received from WKUP3 pin.
  - PWR_FLAG_WUF4: This flag indicates a wakeup event was received from WKUP4 pin.

Return value:

- The: (__FLAG__) state (TRUE or FALSE).

__HAL_PWR_CLEAR_FLAG

Description:

- Clear PWR flags.

Parameters:

- __FLAG__: specifies the flag to clear. This parameter can be one of the following values:
  - PWR_FLAG_STOPF : Stop flag. Indicates that the device was resumed from Stop mode.
  - PWR_FLAG_SBF : Standby flag. Indicates that the device was resumed from Standby mode.

Return value:

- None.

__HAL_PWR_GET_WAKEUPFLAG

Description:

- Check PWR wake up flags are set or not.

Parameters:

- __FLAG__: specifies the wake up flag to check. This parameter can be one of the following values:
  - PWR_FLAG_WKUP1 : This parameter gets Wake up line 1 flag.
  - PWR_FLAG_WKUP2 : This parameter gets Wake up line 2 flag.
  - PWR_FLAG_WKUP3 : This parameter gets Wake up line 3 flag.
  - PWR_FLAG_WKUP4 : This parameter gets Wake up line 4 flag.
  - PWR_FLAG_WKUP : This parameter gets Wake up lines 1 to 4 flags.

Return value:

- The: (__FLAG__) state (TRUE or FALSE).

__HAL_PWR_CLEAR_WAKEUPFLAG

Description:

- Clear CPU PWR wake up flags.

Parameters:

- __FLAG__: : Specifies the wake up flag to be cleared. This parameter can be one of the following values :
  - PWR_FLAG_WKUP1 : This parameter clears Wake up line 1 flag.
  - PWR_FLAG_WKUP2 : This parameter clears Wake up line 2 flag.
  - PWR_FLAG_WKUP3 : This parameter clears Wake up line 3 flag.
  - PWR_FLAG_WKUP4 : This parameter clears Wake up line 4 flag.
  - PWR_FLAG_WKUP : This parameter clears Wake up lines 1 to 4 flags.

Return value:

- None.

__HAL_PWR_PVD_EXTI_ENABLE_IT

Description:

- Enable the PVD Extended Interrupt Line.

Return value:

- None.

__HAL_PWR_PVD_EXTI_DISABLE_IT

Description:

- Disable the PVD Extended Interrupt Line.

Return value:

- None.

__HAL_PWR_PVD_EXTI_ENABLE_EVENT

Description:

- Enable the PVD Event Line.

Return value:

- None.

__HAL_PWR_PVD_EXTI_DISABLE_EVENT

Description:

- Disable the PVD Event Line.

Return value:

- None.

__HAL_PWR_PVD_EXTI_ENABLE_RISING_EDGE

Description:

- Enable the PVD Extended Interrupt Rising Trigger.

Return value:

- None.

__HAL_PWR_PVD_EXTI_DISABLE_RISING_EDGE

Description:

- Disable the PVD Extended Interrupt Rising Trigger.

Return value:

- None.

__HAL_PWR_PVD_EXTI_ENABLE_FALLING_EDGE

Description:

- Enable the PVD Extended Interrupt Falling Trigger.

Return value:

- None.

__HAL_PWR_PVD_EXTI_DISABLE_FALLING_EDGE

Description:

- Disable the PVD Extended Interrupt Falling Trigger.

Return value:

- None.

__HAL_PWR_PVD_EXTI_ENABLE_RISING_FALLING_EDGE

Description:

- Enable the PVD Extended Interrupt Rising & Falling Trigger.

Return value:

- None.

__HAL_PWR_PVD_EXTI_DISABLE_RISING_FALLING_EDGE

Description:

- Disable the PVD Extended Interrupt Rising & Falling Trigger.

Return value:

- None.

__HAL_PWR_PVD_EXTI_GET_RISING_FLAG

Description:

- Check whether or not the PVD EXTI interrupt Rising flag is set.

Return value:

- EXTI: PVD Line Status.

__HAL_PWR_PVD_EXTI_GET_FALLING_FLAG

Description:

- Check whether or not the PVD EXTI interrupt Falling flag is set.

Return value:

- EXTI: PVD Line Status.

__HAL_PWR_PVD_EXTI_CLEAR_RISING_FLAG

Description:

- Clear the PVD EXTI interrupt Rising flag.

Return value:

- None

__HAL_PWR_PVD_EXTI_CLEAR_FALLING_FLAG

Description:

- Clear the PVD EXTI interrupt Falling flag.

Return value:

- None

__HAL_PWR_PVD_EXTI_GENERATE_SWIT

Description:

- Generates a Software interrupt on PVD EXTI line.

Return value:

- None.

PWR Status Flags

PWR_FLAG_PVDO

Programmable voltage detect output

PWR_FLAG_ACTVOSRDY

Voltage levels ready bit

PWR_FLAG_ACTVOS

Programmable voltage detect output

PWR_FLAG_VOSRDY

VOS Ready bit for VCORE voltage scaling output selection

PWR_FLAG_VCOREH

VDDCORE level monitoring versus high threshold

PWR_FLAG_VCOREL

VDDCORE level monitoring versus low threshold

PWR_FLAG_TEMPH

Temperature level monitoring versus high threshold

PWR_FLAG_TEMPL

Temperature level monitoring versus low threshold

PWR_FLAG_VBATH

VBAT level monitoring versus high threshold

PWR_FLAG_VBATL

VBAT level monitoring versus low threshold

PWR_FLAG_SBF

System Standby flag

PWR_FLAG_STOPF

System Stop flag

PWR_FLAG_VDDIO4RDY

VDDIO4 ready

PWR_FLAG_VDDIO5RDY

VDDIO5 ready

PWR_FLAG_ARDY

VDDA18ADC ready

PWR_FLAG_USB33RDY

VDD33USB ready

PWR_FLAG_VDDIO3RDY

VDDIO3 ready

PWR_FLAG_VDDIO2RDY

VDDIO2 ready

PWR_FLAG_WKUPF4

Wakeup event on wakeup pin 4

PWR_FLAG_WKUPF3

Wakeup event on wakeup pin 3

PWR_FLAG_WKUPF2

Wakeup event on wakeup pin 2

PWR_FLAG_WKUPF1

Wakeup event on wakeup pin 1

PWR Flag WakeUp

PWR_FLAG_WKUP1

PWR_FLAG_WKUP2

PWR_FLAG_WKUP3

PWR_FLAG_WKUP4

PWR_FLAG_WKUP

PWR Private macros to check input parameters

IS_PWR_PVD_MODE

IS_PWR_SLEEP_ENTRY

IS_PWR_STOP_ENTRY

IS_PWR_WAKEUP_PIN

IS_PWR_WAKEUP_PIN_POLARITY

IS_PWR_WAKEUP_PIN_PULL

IS_PWR_WAKEUP_FLAG

IS_PWR_ITEMS_ATTRIBUTES

IS_PWR_ATTRIBUTES

PWR Items

PWR_ITEM_0

System supply configuration secure/privilege protection

PWR_ITEM_1

Programmable voltage detector secure/privilege protection

PWR_ITEM_2

VDDCORE monitor secure/privilege protection

PWR_ITEM_3

TCM and FLEXMEM low power control secure/privilege protection

PWR_ITEM_4

Voltage scaling selection secure/privilege protection

PWR_ITEM_5

Backup domain secure/privilege protection

PWR_ITEM_6

CPU power control secure/privilege protection

PWR_ITEM_7

Peripheral voltage monitor secure/privilege protection

PWR_ITEM_WKUP1

WKUP1 pin secure/privilege protection

PWR_ITEM_WKUP2

WKUP2 pin secure/privilege protection

PWR_ITEM_WKUP3

WKUP3 pin secure/privilege protection

PWR_ITEM_WKUP4

WKUP4 pin secure/privilege protection

PWR_ITEM_ALL

PWR Pin Polarity configuration

PWR_PIN_POLARITY_HIGH

PWR_PIN_POLARITY_LOW

PWR Pin Pull configuration

PWR_PIN_NO_PULL

PWR_PIN_PULL_UP

PWR_PIN_PULL_DOWN

PWR PVD EXTI Line

PWR_EXTI_LINE_PVD

External interrupt line 66 connected to the PVD EXTI Line

PWR PVD Mode

PWR_PVD_MODE_NORMAL

Basic mode is used

PWR_PVD_MODE_IT_RISING

Interrupt Mode with Rising edge trigger detection

PWR_PVD_MODE_IT_FALLING

Interrupt Mode with Falling edge trigger detection

PWR_PVD_MODE_IT_RISING_FALLING

Interrupt Mode with Rising/Falling edge trigger detection

PWR_PVD_MODE_EVENT_RISING

Event Mode with Rising edge trigger detection

PWR_PVD_MODE_EVENT_FALLING

Event Mode with Falling edge trigger detection

PWR_PVD_MODE_EVENT_RISING_FALLING

Event Mode with Rising/Falling edge trigger detection

PWR PVD Mode Mask

PVD_MODE_IT

Mask for interruption yielded by PVD threshold crossing

PVD_MODE_EVT

Mask for event yielded by PVD threshold crossing

PVD_RISING_EDGE

Mask for rising edge set as PVD trigger

PVD_FALLING_EDGE

Mask for falling edge set as PVD trigger

PVD_RISING_FALLING_EDGE

Mask for rising & falling edge set as PVD trigger

PWR PVM Mode Mask

PVM_RISING_EDGE

Mask for rising edge set as PVM trigger

PVM_FALLING_EDGE

Mask for falling edge set as PVM trigger

PVM_MODE_IT

Mask for interruption yielded by PVM threshold crossing

PVM_MODE_EVT

Mask for event yielded by PVM threshold crossing

PWR Regulator state in SLEEP/STOP mode

PWR_MAINREGULATOR_ON

PWR Regulator Voltage Scale

PWR_REGULATOR_VOLTAGE_SCALE0

Voltage scaling range 0 (highest frequency)

PWR_REGULATOR_VOLTAGE_SCALE1

Voltage scaling range 1 (lowest power)

PWR SLEEP mode entry

PWR_SLEEPENTRY_WFI

Wait For Interruption instruction to enter Sleep mode

PWR_SLEEPENTRY_WFE

Wait For Event instruction to enter Sleep mode

PWR_SLEEPENTRY_WFE_NO_EVT_CLEAR

Wait For Event instruction to enter Sleep mode with no event clear

PWR STOP mode entry

PWR_STOPENTRY_WFI

Wait For Interruption instruction to enter Stop mode

PWR_STOPENTRY_WFE

Wait For Event instruction to enter Stop mode

PWR_STOPENTRY_WFE_NO_EVT_CLEAR

Wait For Event instruction to enter Stop mode with no event clear

PWR VddCORE VM Mode Mask

VDDCOREVM_RISING_EDGE

Mask for rising edge set as PVM trigger

VDDCOREVM_FALLING_EDGE

Mask for falling edge set as PVM trigger

VDDCOREVM_MODE_IT

Mask for interruption yielded by PVM threshold crossing

VDDCOREVM_MODE_EVT

Mask for event yielded by PVM threshold crossing

PWR Wake-Up Pins

PWR_WAKEUP_PIN4

Wakeup pin 1 (with high level polarity)

PWR_WAKEUP_PIN3

Wakeup pin 2 (with high level polarity)

PWR_WAKEUP_PIN2

Wakeup pin 3 (with high level polarity)

PWR_WAKEUP_PIN1

Wakeup pin 4 (with high level polarity)

PWR_WAKEUP_PIN4_HIGH

Wakeup pin 1 (with high level polarity)

PWR_WAKEUP_PIN3_HIGH

Wakeup pin 2 (with high level polarity)

PWR_WAKEUP_PIN2_HIGH

Wakeup pin 3 (with high level polarity)

PWR_WAKEUP_PIN1_HIGH

Wakeup pin 4 (with high level polarity)

PWR_WAKEUP_PIN4_LOW

Wakeup pin 1 (with low level polarity)

PWR_WAKEUP_PIN3_LOW

Wakeup pin 2 (with low level polarity)

PWR_WAKEUP_PIN2_LOW

Wakeup pin 3 (with low level polarity)

PWR_WAKEUP_PIN1_LOW

Wakeup pin 4 (with low level polarity)

PWR Wakeup Pins Flags.

PWR_WAKEUP_FLAG1

Wakeup flag on PA0

PWR_WAKEUP_FLAG2

Wakeup flag on PA2

PWR_WAKEUP_FLAG3

Wakeup flag on PC13

PWR_WAKEUP_FLAG4

Wakeup flag on PD2

PWR_WAKEUP_FLAG_ALL

55 HAL RAMCFG Generic Driver

55.1 RAMCFG Firmware driver registers structures

55.1.1 __RAMCFG_HandleTypeDef

__RAMCFG_HandleTypeDef is defined in the stm32n6xx_hal_ramcfg.h

Data Fields

- ***RAMCFG_TypeDef * Instance***
- ***__IO HAL_RAMCFG_StateTypeDef State***
- ***__IO uint32_t ErrorCode***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***
- ***void(* DetectSingleErrorCallback***
- ***void(* DetectDoubleErrorCallback***

Field Documentation

- ***RAMCFG_TypeDef* __RAMCFG_HandleTypeDef::Instance***
  RAMCFG Register Base Address
- ***__IO HAL_RAMCFG_StateTypeDef __RAMCFG_HandleTypeDef::State***
  RAMCFG State
- ***__IO uint32_t __RAMCFG_HandleTypeDef::ErrorCode***
  RAMCFG Error Code
- ***void(* __RAMCFG_HandleTypeDef::MspInitCallback)(struct __RAMCFG_HandleTypeDef *hramcfg)***
  RAMCFG MSP Init Callback
- ***void(* __RAMCFG_HandleTypeDef::MspDeInitCallback)(struct __RAMCFG_HandleTypeDef *hramcfg)***
  RAMCFG MSP DeInit Callback
- ***void(* __RAMCFG_HandleTypeDef::DetectSingleErrorCallback)(struct __RAMCFG_HandleTypeDef *hramcfg)***
  RAMCFG Single Error Detect Callback
- ***void(* __RAMCFG_HandleTypeDef::DetectDoubleErrorCallback)(struct __RAMCFG_HandleTypeDef *hramcfg)***
  RAMCFG Double Error Detect Callback

55.2 RAMCFG Firmware driver API description

The following section lists the various functions of the RAMCFG library.

55.2.1 How to use this driver

- Each SRAM is managed by a RAMCFG instance (AHBSRAM1/2, AXISRAM1 to 6, BKPSRAM, FLEXRAM, and VENCRAM).
- Each SRAM can be erased independently through its RAMCFG instance.
- AXISRAM 1 to 6 are the main SRAMs. AHBSRAM1/2 are preferably for DMA-controlled peripheral-to-memory data flow. AXISRAM 2 to 6 can be shut down when the application is in Run mode.
- FLEXMEM can be either allocated as Cortex-M55 TCM, or as system RAM (FLEXRAM). 80 Kbytes of the FLEXMEM are retained in Standby mode, either allocated as extended ITCM (64 +16-Kbyte ECC), or allocated as FLEXRAM.
- The BKPSRAM content is retained in low-power modes, even when VDD is off in VBAT mode. On a tamper event detection, the BKPSRAM content is erased.
- The VENCRAM implements hardware and software erases.

- FLEXRAM and BKPRAM support ECC correction feature:
  - Single error detection and correction with interrupt generation.
  - Double error detection with interrupt generation.
  - Status with failing address
- Call HAL_RAMCFG_Init() to initialize the RAMCFG peripheral before using any feature. Call HAL_RAMCFG_DeInit() to de-initialize the RAMCFG when using this peripheral is no more needed or a hardware issue has occurred.
  - HAL_RAMCFG_Init() and HAL_RAMCFG_DeInit() APIs do not change the activation status of ECC feature. It is managed by HAL_RAMCFG_StartECC(), HAL_RAMCFG_StopECC() or option bytes (When available on the device).

ECC feature

- Call HAL_RAMCFG_StartECC() and HAL_RAMCFG_StopECC() to enable and disable ECC hardware mechanism.
  - When ECC feature is previously enabled (case of option byte activation), calling HAL_RAMCFG_StartECC() is recommended to enable the ECC address latching feature.
- Call HAL_RAMCFG_EnableNotification() and HAL_RAMCFG_DisableNotification() to enable and disable ECC interrupts. Interrupts can be:
  - Single error interrupt.
  - Double error interrupt.
  - Double error interrupt redirected to Non maskable interrupt (NMI).
- Call HAL_RAMCFG_GetSingleErrorAddress() to get the address of the last fail RAM word detected (only for single error) and call HAL_RAMCFG_GetDoubleErrorAddress() to get the address of the last fail RAM word detected (only for double error).
- Call HAL_RAMCFG_IsECCSingleErrorDetected() to check if an ECC single error was detected. Call HAL_RAMCFG_IsECCDoubleErrorDetected() to check if an ECC double error was detected. These APIs are used in silent mode (No ECC interrupt is enabled).

Erase feature

- Call HAL_RAMCFG_Erase() to launch a hardware erase for the given SRAM.
- The erase value is equal to 0 when launching erase hardware through RAMCFG.
- SRAM2 write protected pages are erased when performing an erase through RAMCFG.

RAMCFG HAL driver macros list

Below the list of used macros in RAMCFG HAL driver.

- __HAL_RAMCFG_ENABLE_IT : Enable the specified RAMCFG interrupts.
- __HAL_RAMCFG_DISABLE_IT : Disable the specified RAMCFG interrupts.
- __HAL_RAMCFG_GET_FLAG : Get the RAMCFG pending flags.
- __HAL_RAMCFG_CLEAR_FLAG : Clear the RAMCFG pending flags.
- __HAL_RAMCFG_GET_IT_SOURCE : Check whether the specified RAMCFG interrupt source is enabled or not.

55.2.2 Detailed description of functions

HAL_RAMCFG_Init

Function name

HAL_StatusTypeDef HAL_RAMCFG_Init (RAMCFG_HandleTypeDef * hramcfg)

Function description

HAL_RAMCFG_DeInit

Function name

HAL_StatusTypeDef HAL_RAMCFG_DeInit (RAMCFG_HandleTypeDef * hramcfg)

Function description

HAL_RAMCFG_MspInit

Function name

void HAL_RAMCFG_MspInit (RAMCFG_HandleTypeDef * hramcfg)

Function description

HAL_RAMCFG_MspDeInit

Function name

void HAL_RAMCFG_MspDeInit (RAMCFG_HandleTypeDef * hramcfg)

Function description

HAL_RAMCFG_StartECC

Function name

HAL_StatusTypeDef HAL_RAMCFG_StartECC (RAMCFG_HandleTypeDef * hramcfg)

Function description

HAL_RAMCFG_StopECC

Function name

HAL_StatusTypeDef HAL_RAMCFG_StopECC (RAMCFG_HandleTypeDef * hramcfg)

Function description

HAL_RAMCFG_EnableNotification

Function name

HAL_StatusTypeDef HAL_RAMCFG_EnableNotification (RAMCFG_HandleTypeDef * hramcfg, uint32_t Notifications)

Function description

HAL_RAMCFG_DisableNotification

Function name

HAL_StatusTypeDef HAL_RAMCFG_DisableNotification (RAMCFG_HandleTypeDef * hramcfg, uint32_t Notifications)

Function description

HAL_RAMCFG_IsECCSingleErrorDetected

Function name

uint32_t HAL_RAMCFG_IsECCSingleErrorDetected (const RAMCFG_HandleTypeDef * hramcfg)

Function description

HAL_RAMCFG_IsECCDoubleErrorDetected

Function name

uint32_t HAL_RAMCFG_IsECCDoubleErrorDetected (const RAMCFG_HandleTypeDef * hramcfg)

Function description

HAL_RAMCFG_GetSingleErrorAddress

Function name

uint32_t HAL_RAMCFG_GetSingleErrorAddress (const RAMCFG_HandleTypeDef * hramcfg)

Function description

HAL_RAMCFG_GetDoubleErrorAddress

Function name

uint32_t HAL_RAMCFG_GetDoubleErrorAddress (const RAMCFG_HandleTypeDef * hramcfg)

Function description

HAL_RAMCFG_Erase

Function name

HAL_StatusTypeDef HAL_RAMCFG_Erase (RAMCFG_HandleTypeDef * hramcfg)

Function description

HAL_RAMCFG_IRQHandler

Function name

void HAL_RAMCFG_IRQHandler (RAMCFG_HandleTypeDef * hramcfg)

Function description

HAL_RAMCFG_DetectSingleErrorCallback

Function name

void HAL_RAMCFG_DetectSingleErrorCallback (RAMCFG_HandleTypeDef * hramcfg)

Function description

HAL_RAMCFG_DetectDoubleErrorCallback

Function name

void HAL_RAMCFG_DetectDoubleErrorCallback (RAMCFG_HandleTypeDef * hramcfg)

Function description

HAL_RAMCFG_RegisterCallback

Function name

HAL_StatusTypeDef HAL_RAMCFG_RegisterCallback (RAMCFG_HandleTypeDef * hramcfg,
HAL_RAMCFG_CallbackIDTypeDef CallbackID, void(*)(RAMCFG_HandleTypeDef *_hramcfg) pCallback)

Function description

HAL_RAMCFG_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_RAMCFG_UnRegisterCallback (RAMCFG_HandleTypeDef * hramcfg, HAL_RAMCFG_CallbackIDTypeDef CallbackID)

Function description

HAL_RAMCFG_GetError

Function name

uint32_t HAL_RAMCFG_GetError (const RAMCFG_HandleTypeDef * hramcfg)

Function description

HAL_RAMCFG_GetState

Function name

HAL_RAMCFG_StateTypeDef HAL_RAMCFG_GetState (const RAMCFG_HandleTypeDef * hramcfg)

Function description

HAL_RAMCFG_EnableAXISRAM

Function name

void HAL_RAMCFG_EnableAXISRAM (const RAMCFG_HandleTypeDef * hramcfg)

Function description

HAL_RAMCFG_DisableAXISRAM

Function name

void HAL_RAMCFG_DisableAXISRAM (const RAMCFG_HandleTypeDef * hramcfg)

Function description

55.3 RAMCFG Firmware driver defines

The following section lists the various define and macros of the module.

55.3.1 RAMCFG

RAMCFG

RAMCFG SRAM Power Down

RAMCFG_AXISRAM_POWERDOWN

AXISRAM power down

RAMCFG Error Codes

HAL_RAMCFG_ERROR_NONE

RAMCFG No Error

HAL_RAMCFG_ERROR_TIMEOUT

RAMCFG Timeout Error

HAL_RAMCFG_ERROR_BUSY

RAMCFG Busy Error

HAL_RAMCFG_ERROR_INVALID_CALLBACK

Invalid Callback error

RAMCFG Exported Macros

__HAL_RAMCFG_ENABLE_IT

Description:

- Enable the specified RAMCFG interrupts.

Parameters:

- __HANDLE__: : Specifies RAMCFG handle.
- __INTERRUPT__: Specifies the RAMCFG interrupt sources to be enabled. This parameter can be one of the following values:
  - RAMCFG_IT_SINGLEERR : Single Error Interrupt Mask.
  - RAMCFG_IT_DOUBLEERR : Double Error Interrupt Mask.
  - RAMCFG_IT_NMIERR : Double Error Interrupt redirection to NMI Mask.
  - RAMCFG_IT_ALL : All Interrupt Mask.

Return value:

- None

__HAL_RAMCFG_DISABLE_IT

Description:

- Disable the specified RAMCFG interrupts.

Parameters:

- __HANDLE__: : Specifies RAMCFG handle.
- __INTERRUPT__: Specifies the RAMCFG interrupt sources to be disabled. This parameter can be one of the following values:
  - RAMCFG_IT_SINGLEERR : Single Error Interrupt Mask.
  - RAMCFG_IT_DOUBLEERR : Double Error Interrupt Mask.

Return value:

- None

Notes:

- This macros is used only to disable RAMCFG_IT_SINGLEERR and RAMCFG_IT_DOUBLEERR interrupts. RAMCFG_IT_NMIERR interrupt can only be disabled by global peripheral reset or system reset.

__HAL_RAMCFG_GET_IT_SOURCE

Description:

- Check whether the specified RAMCFG interrupt source is enabled or not.

Parameters:

- __HANDLE__: : Specifies the RAMCFG Handle.
- __INTERRUPT__: : Specifies the RAMCFG interrupt source to check. This parameter can be one of the following values:
  - RAMCFG_IT_SINGLEERR : Single Error Interrupt Mask.
  - RAMCFG_IT_DOUBLEERR : Double Error Interrupt Mask.
  - RAMCFG_IT_NMIERR : Double Error Interrupt Redirection to NMI Mask.

Return value:

- The: new state of __INTERRUPT__ (SET or RESET).

__HAL_RAMCFG_GET_FLAG

Description:

- Get the RAMCFG pending flags.

Parameters:

- __HANDLE__: : Specifies RAMCFG handle.
- __FLAG__: : Specifies the flag to be checked. This parameter can be one of the following values:
  - RAMCFG_FLAG_SINGLEERR : Single Error Detected and Corrected Flag.
  - RAMCFG_FLAG_DOUBLEERR : Double Error Detected Flag.
  - RAMCFG_FLAG_SRAMBUSY : SRAM Busy Flag.

Return value:

- The: state of FLAG (SET or RESET).

__HAL_RAMCFG_CLEAR_FLAG

Description:

- Clear the RAMCFG pending flags.

Parameters:

- __HANDLE__: : Specifies RAMCFG handle.
- __FLAG__: : Specifies the flag to be cleared. This parameter can be any combination of the following values:
  - RAMCFG_FLAG_SINGLEERR : Single Error Detected and Corrected Flag.
  - RAMCFG_FLAG_DOUBLEERR : Double Error Detected Flag.

Return value:

- None.

__HAL_RAMCFG_RESET_HANDLE_STATE

Description:

- Reset the RAMCFG handle state.

Parameters:

- __HANDLE__: : Specifies the RAMCFG Handle.

Return value:

- None.

RAMCFG Monitor Flags

RAMCFG_FLAG_SINGLEERR

RAMCFG Single Error Detected and Corrected Flag

RAMCFG_FLAG_DOUBLEERR

RAMCFG Double Error Detected Flag

RAMCFG_FLAG_SRAMBUSY

RAMCFG SRAM busy Flag

RAMCFG_FLAGS_ALL

RAMCFG All Flags

RAMCFG Interrupts

RAMCFG_IT_SINGLEERR

RAMCFG Single Error Interrupt

RAMCFG_IT_DOUBLEERR

RAMCFG Double Error Interrupt

RAMCFG_IT_ALL

RAMCFG All RAMCFG interrupt

RAMCFG Keys

RAMCFG_ERASE_KEY1

RAMCFG launch Erase Key 1

RAMCFG_ERASE_KEY2

RAMCFG launch Erase Key 2

RAMCFG_ECC_KEY1

RAMCFG launch ECC Key 1

RAMCFG_ECC_KEY2

RAMCFG launch ECC Key 2

56 HAL RCC Generic Driver

56.1 RCC Firmware driver registers structures

56.1.1 RCC_PLLInitTypeDef

RCC_PLLInitTypeDef is defined in the stm32n6xx_hal_rcc.h

Data Fields

- ***uint32_t PLLState***
- ***uint32_t PLLSource***
- ***uint32_t PLLM***
- ***uint32_t PLLFractional***
- ***uint32_t PLLN***
- ***uint32_t PLLP1***
- ***uint32_t PLLP2***

Field Documentation

- ***uint32_t RCC_PLLInitTypeDef::PLLState***
  The new state of the PLL. This parameter can be a value of ***RCC_PLL_Config***
- ***uint32_t RCC_PLLInitTypeDef::PLLSource***
  PLL entry clock source. This parameter must be a value of ***RCC_PLL_Clock_Source***
- ***uint32_t RCC_PLLInitTypeDef::PLLM***
  Division factor M for PLL VCO input clock. This parameter must be a number between Min_Data = 1 and Max_Data = 63
- ***uint32_t RCC_PLLInitTypeDef::PLLFractional***
  Fractional part of he VCO mulliplication factor. This parameter must be a number between Min_Data = 0 and Max_Data = 0xFFFFFF
- ***uint32_t RCC_PLLInitTypeDef::PLLN***
  Multiplication factor N for PLL VCO output clock. In integer mode, this parameter must be a number between Min_Data = 10 and Max_Data = 2500. In fractional mode, this parameter must be a number between Min_Data = 20 and Max_Data = 500.
- ***uint32_t RCC_PLLInitTypeDef::PLLP1***
  Division factor P1 for system clock. This parameter must be a number between Min_Data = 1 and Max_Data = 7
- ***uint32_t RCC_PLLInitTypeDef::PLLP2***
  Division factor P2 for system clock. This parameter must be a number between Min_Data = 1 and Max_Data = 7

56.1.2 RCC_OscInitTypeDef

RCC_OscInitTypeDef is defined in the stm32n6xx_hal_rcc.h

Data Fields

- ***uint32_t OscillatorType***
- ***uint32_t HSEState***
- ***uint32_t LSEState***
- ***uint32_t HSIState***
- ***uint32_t HSIDiv***
- ***uint32_t HSICalibrationValue***
- ***uint32_t LSIState***
- ***uint32_t MSIState***
- ***uint32_t MSIFrequency***
- ***uint32_t MSICalibrationValue***
- ***RCC_PLLInitTypeDef PLL1***
- ***RCC_PLLInitTypeDef PLL2***

- ***RCC_PLLInitTypeDef PLL3***
- ***RCC_PLLInitTypeDef PLL4***

Field Documentation

- ***uint32_t RCC_OscInitTypeDef::OscillatorType***
  The oscillators to be configured. This parameter can be a value of ***RCC_Oscillator_Type***
- ***uint32_t RCC_OscInitTypeDef::HSEState***
  The new state of the HSE. This parameter can be a value of ***RCC_HSE_Config***
- ***uint32_t RCC_OscInitTypeDef::LSEState***
  The new state of the LSE. This parameter can be a value of ***RCC_LSE_Config***
- ***uint32_t RCC_OscInitTypeDef::HSIState***
  The new state of the HSI. This parameter can be a value of ***RCC_HSI_Config***
- ***uint32_t RCC_OscInitTypeDef::HSIDiv***
  The division factor of the HSI. This parameter can be a value of ***RCC_HSI_Div***
- ***uint32_t RCC_OscInitTypeDef::HSICalibrationValue***
  The calibration trimming value. This parameter must be a number between Min_Data = 0 and Max_Data = 127
- ***uint32_t RCC_OscInitTypeDef::LSIState***
  The new state of the LSI. This parameter can be a value of ***RCC_LSI_Config***
- ***uint32_t RCC_OscInitTypeDef::MSIState***
  The new state of the MSI. This parameter can be a value of ***RCC_MSI_Config***
- ***uint32_t RCC_OscInitTypeDef::MSIFrequency***
  The MSI frequency selection. This parameter can be a value of ***RCC_MSI_Frequency***
- ***uint32_t RCC_OscInitTypeDef::MSICalibrationValue***
  The calibration trimming value. This parameter must be a number between Min_Data = 0 and Max_Data = 31
- ***RCC_PLLInitTypeDef RCC_OscInitTypeDef::PLL1***
  PLL1 structure parameters
- ***RCC_PLLInitTypeDef RCC_OscInitTypeDef::PLL2***
  PLL2 structure parameters
- ***RCC_PLLInitTypeDef RCC_OscInitTypeDef::PLL3***
  PLL3 structure parameters
- ***RCC_PLLInitTypeDef RCC_OscInitTypeDef::PLL4***
  PLL4 structure parameters

56.1.3 RCC_ICInitTypeDef

RCC_ICInitTypeDef is defined in the stm32n6xx_hal_rcc.h

Data Fields

- ***uint32_t ClockSelection***
- ***uint32_t ClockDivider***

Field Documentation

- ***uint32_t RCC_ICInitTypeDef::ClockSelection***
  Specifies ICx clock source. This parameter can be a value of ***RCC_IC_Clock_Source***
- ***uint32_t RCC_ICInitTypeDef::ClockDivider***
  Specifies ICx clock divider. This parameter must be a number between Min_Data = 1 and Max_Data = 256

56.1.4 RCC_ClkInitTypeDef

RCC_ClkInitTypeDef is defined in the stm32n6xx_hal_rcc.h

Data Fields

- ***uint32_t ClockType***
- ***uint32_t CPUCLKSource***
- ***uint32_t SYSCLKSource***
- ***uint32_t AHBCLKDivider***
- ***uint32_t APB1CLKDivider***

- ***uint32_t APB2CLKDivider***
- ***uint32_t APB4CLKDivider***
- ***uint32_t APB5CLKDivider***
- ***RCC_ICInitTypeDef IC1Selection***
- ***RCC_ICInitTypeDef IC2Selection***
- ***RCC_ICInitTypeDef IC6Selection***
- ***RCC_ICInitTypeDef IC11Selection***

Field Documentation

- ***uint32_t RCC_ClkInitTypeDef::ClockType***
  The clock to be configured. This parameter can be a value of ***RCC_Clock_Type***
- ***uint32_t RCC_ClkInitTypeDef::CPUCLKSource***
  The clock source used as CPU clock (CPUCLK). This parameter can be a value of ***RCC_CPU_Clock_Source***
- ***uint32_t RCC_ClkInitTypeDef::SYSCLKSource***
  The clock source used as system bus clock (SYSCLK). This parameter can be a value of ***RCC_System_Bus_Clock_Source***
- ***uint32_t RCC_ClkInitTypeDef::AHBCLKDivider***
  The AHB clock divider for HCLK. This clock is derived from the system clock divided by the system clock divider. This parameter can be a value of ***RCC_HCLK_Clock_Source***
- ***uint32_t RCC_ClkInitTypeDef::APB1CLKDivider***
  The APB1 clock divider for PCLK1. This clock is derived from the AHB clock (HCLK). This parameter can be a value of ***RCC_APB1_Clock_Source***
- ***uint32_t RCC_ClkInitTypeDef::APB2CLKDivider***
  The APB2 clock divider for PCLK2. This clock is derived from the AHB clock (HCLK). This parameter can be a value of ***RCC_APB2_Clock_Source***
- ***uint32_t RCC_ClkInitTypeDef::APB4CLKDivider***
  The APB4 clock divider for PCLK4. This clock is derived from the AHB clock (HCLK). This parameter can be a value of ***RCC_APB4_Clock_Source***
- ***uint32_t RCC_ClkInitTypeDef::APB5CLKDivider***
  The APB5 clock divider for PCLK5. This clock is derived from the AHB clock (HCLK). This parameter can be a value of ***RCC_APB5_Clock_Source***
- ***RCC_ICInitTypeDef RCC_ClkInitTypeDef::IC1Selection***
  IC1 parameters. This parameter shall be used when IC1 is selected as CPU clock source (sysa_ck)
- ***RCC_ICInitTypeDef RCC_ClkInitTypeDef::IC2Selection***
  IC2 parameters. This parameter shall be used when IC2 is selected as system bus clock source (sysb_ck)
- ***RCC_ICInitTypeDef RCC_ClkInitTypeDef::IC6Selection***
  IC6 parameters. This parameter shall be used when IC6 is selected as system bus clock source (sysc_ck)
- ***RCC_ICInitTypeDef RCC_ClkInitTypeDef::IC11Selection***
  IC11 parameters. This parameter shall be used when IC11 is selected as system bus clock source (sysd_ck)

56.2 RCC Firmware driver API description

The following section lists the various functions of the RCC library.

56.2.1 RCC specific features

After reset the device is running from Internal High Speed oscillator (HSI 64MHz) and all peripherals are off except internal SRAM, BSEC, PWR, IAC, RIFSC, RISAF and JTAG

- There is no pre-scaler on High speed (AHB) and Low speed (APB) buses; all peripherals mapped on these buses are running at HSI speed.
- All GPIOs are in analog mode , except the JTAG pins which are assigned to be used for debug purpose.

Once the device started from reset, the user application has to:

- Configure the clock source to be used to drive the System clock (if the application needs higher frequency/ performance than HSI)
- Configure the CPU and System bus clock frequencies

- Configure the AHB and APB buses pre-scalers
- Enable the clock for the peripheral(s) to be used
- Configure the clock kernel source(s) for peripherals which clocks are not derived from the System clock

56.2.2 RCC Limitations

A delay between an RCC peripheral clock enable and the effective peripheral enabling should be taken into account in order to manage the peripheral read/write from/to registers.

- This delay depends on the peripheral mapping.
- If peripheral is mapped on AHB: the delay is 2 AHB clock cycle after the clock enable bit is set on the hardware register
- If peripheral is mapped on APB: the delay is 2 APB clock cycle after the clock enable bit is set on the hardware register

Implemented Workaround:

- For AHB & APB peripherals, a dummy read to the peripheral register has been inserted in each __HAL_RCC_PPP_CLK_ENABLE() macro.

Incorrect APB prescaler setting:

- As described in the product errata, after enabling the bus clock to the peripheral and changing the default APB prescaler setting, performing a configuration on the bus peripheral does not update the peripheral registers, resulting in unexpected behavior.

56.2.3 Initialization and de-initialization functions

This section provides functions allowing to configure the internal/external oscillators (HSE, HSI, LSE, MSI, LSI, PLL, CSS and MCO) and the CPU and System buses clocks (SYSCLK, AHB1, AHB2, AHB3, AHB4, AHB5, APB1, APB2, APB4 and APB5).

Internal/external clock and PLL configuration

1. HSI (high-speed internal), 64 MHz factory-trimmed RC used directly or through the PLL as System clock source.
2. MSI is a low-power RC oscillator which can be used directly as system clock, peripheral clock, or PLL input.But even with frequency calibration, is less accurate than an external crystal oscillator or ceramic resonator.
3. LSI (low-speed internal), 32 KHz low consumption RC used as IWDG and/or RTC clock source.
4. HSE (high-speed external), 4 to 48 MHz crystal oscillator used directly or through the PLL as System clock source. Can be used also as RTC clock source.
5. LSE (low-speed external), 32 KHz oscillator used as RTC clock source.
6. PLL , The RCC features three independent PLLs (clocked by HSI , HSE or MSI), featuring three different output clocks and able to work either in integer or Fractional mode.
   – A main PLL, PLL1, which is generally used to provide clocks to the CPU and to some peripherals via the ICx mux.
   – Two dedicated PLLs, PLL2 and PLL3, which are used to generate the kernel clock for peripherals.
7. HSE CSS (Clock security system), once enabled and if a HSE clock failure occurs (HSE used directly or through PLL as System clock source), the System clock is automatically switched to HSI and an interrupt is generated if enabled. The interrupt is linked to the Cortex-M NMI (Non-Maskable Interrupt) exception vector.
8. LSE CSS (Clock security system), once enabled and if a LSE clock failure occurs the LSE is no more provided to RTC, it allows to wake-up from standby mode and protect the backup registers and BKPSRAM via TAMP. The LSE CSS may be re-armed for further attempt(s) to use LSE.
9. MCO1 (micro controller clock output), used to output HSI, LSE, HSE, LSI, MSI, CPU (SYSA), IC5 mux or IC10 mux clock (through a configurable pre-scaler) on PA8 pin.
10. MCO2 (micro controller clock output), used to output HSI, LSE, HSE, LSI, MSI, System Bus (SYSB), IC15 mux or IC20 mux clock (through a configurable pre-scaler) on PC9 pin.

CPU, System bus, AHB and APB buses clocks configuration

1. Several clock sources can be used to drive independently the CPU (CPUCLK) and System bus clock (SYSCLK): HSI, MSI, HSE or ICx mux. The AHB clock (HCLK) is derived from System core clock through configurable pre-scaler and used to clock the CPU, memory and peripherals mapped on AHB and APB bus through configurable pre-scalers and used to clock the peripherals mapped on these buses. You can use "HAL_RCC_GetSysClockFreq()" function to retrieve system clock frequency.

Note: All the peripheral clocks are derived from the System bus clock (SYSCLK).

This section contains the following APIs:

- ***HAL_RCC_DeInit()***
- ***HAL_RCC_OscConfig()***
- ***HAL_RCC_ClockConfig()***

56.2.4 Peripheral Control functions

This subsection provides a set of functions allowing to control the RCC Clocks frequencies.

This section contains the following APIs:

- ***HAL_RCC_MCOConfig()***
- ***HAL_RCC_EnableCSS()***
- ***HAL_RCC_GetCpuClockFreq()***
- ***HAL_RCC_GetSysClockFreq()***
- ***HAL_RCC_GetHCLKFreq()***
- ***HAL_RCC_GetPCLK1Freq()***
- ***HAL_RCC_GetPCLK2Freq()***
- ***HAL_RCC_GetPCLK4Freq()***
- ***HAL_RCC_GetPCLK5Freq()***
- ***HAL_RCC_GetOscConfig()***
- ***HAL_RCC_GetClockConfig()***
- ***HAL_RCC_NMI_IRQHandler()***
- ***HAL_RCC_CSSCallback()***

56.2.5 RCC attributes functions

This subsection provides a set of functions allowing to:

- Configure the RCC item(s) attribute(s).
- Get the attribute of an RCC item.

This section contains the following APIs:

- ***HAL_RCC_ConfigAttributes()***
- ***HAL_RCC_GetConfigAttributes()***

56.2.6 Detailed description of functions

HAL_RCC_DeInit

Function name

HAL_StatusTypeDef HAL_RCC_DeInit (void)

Function description

Resets the RCC clock configuration to the default reset state.

Return values

- **HAL:** status

Notes

- The default reset state of the clock configuration is given below: HSI ON and used as CPU and system clock sourceMSI and HSE OFFPLL1, PLL2, PLL3 and PLL4 bypassedAHB, APB Bus prescalers set to 1CSS, MCO1 and MCO2 OFFAll interrupts disabled
- This function doesn't modify the configuration of the Peripheral clocksMemory clocksMisc clocksLSI, LSE and RTC clocksICx selections and dividers

HAL_RCC_OscConfig

Function name

HAL_StatusTypeDef HAL_RCC_OscConfig (const RCC_OscInitTypeDef * pRCC_OscInitStruct)

Function description

Initializes the RCC Oscillators according to the specified parameters in the RCC_OscInitTypeDef.

Parameters

- **pRCC_OscInitStruct:** Pointer to an RCC_OscInitTypeDef structure that contains the configuration information for the RCC Oscillators.

Return values

- **HAL:** status

Notes

- LSE control requires that the backup domain shall be previously enabled with HAL_PWR_EnableBkUpAccess().
- The PLL is not disabled when used as system clock.
- Transitions LSE Bypass to LSE On and LSE On to LSE Bypass are not supported by this function. User should request a transition to LSE Off first and then LSE On or LSE Bypass.
- Transition HSE Bypass to HSE On and HSE On to HSE Bypass are not supported by this function. User should request a transition to HSE Off first and then HSE On or HSE Bypass.
- This function does not protect the MCOxSEL, the PERSEL and the PPPSEL glitch-free muxes (Mux selection cannot be changed if selected input clock is inactive).
- This function activates HSE but does not wait for the startup time defined in the datasheet. This must be ensured by the application when the HSE is selected as PLL source.

HAL_RCC_ClockConfig

Function name

HAL_StatusTypeDef HAL_RCC_ClockConfig (const RCC_ClkInitTypeDef * pRCC_ClkInitStruct)

Function description

Initializes the CPU, System AXI, AHB, APB buses clocks according to the specified parameters in the RCC_ClkInitStruct.

Parameters

- **pRCC_ClkInitStruct:** Pointer to an RCC_OscInitTypeDef structure that contains the configuration information for the RCC peripheral.

Return values

- **None:**

Notes

- The SystemCoreClock CMSIS variable is used to store System Core Clock Frequency and updated by HAL_InitTick() function called within this function.
- The HSI is used (enabled by hardware) as system clock source after start-up from Reset, wake-up from STOP (unless MSI is selected) and STANDBY mode, or in case of failure of the HSE used directly or indirectly as system clock (if the Clock Security System CSS is enabled).
- A switch from one clock source to another occurs only if the target clock source is ready (clock stable after start-up delay or PLL locked). If a clock source which is not yet ready is selected, the switch will occur when the clock source will be ready. You can use HAL_RCC_GetClockConfig() function to know which clock is currently used as system clock source.

HAL_RCC_MCOConfig

Function name

void HAL_RCC_MCOConfig (uint32_t RCC_MCOx, uint32_t RCC_MCOSource, uint32_t RCC_MCODiv)

Function description

Selects the clock source to output on MCO1 pin(PA8) or on MCO2 pin(PC9).

Parameters

- **RCC_MCOx:** specifies the output direction for the clock source. This parameter can be one of the following values:
  - RCC_MCO1 Clock source to output on MCO1 pin(PA8).
  - RCC_MCO2 Clock source to output on MCO2 pin(PC9).
- **RCC_MCOSource:** specifies the clock source to output. This parameter can be one of the following values:
  - RCC_MCO1SOURCE_LSI LSI clock selected as MCO1 source
  - RCC_MCO1SOURCE_LSE LSE clock selected as MCO1 source
  - RCC_MCO1SOURCE_MSI MSI clock selected as MCO1 source
  - RCC_MCO1SOURCE_HSI HSI clock selected as MCO1 source
  - RCC_MCO1SOURCE_HSE HSE clock selected as MCO1 source
  - RCC_MCO1SOURCE_IC5 IC5 clock selected as MCO1 source
  - RCC_MCO1SOURCE_IC10 IC10 clock selected as MCO1 source
  - RCC_MCO1SOURCE_SYSA CPU clock (SYSA) selected as MCO1 source
  - RCC_MCO2SOURCE_LSI LSI clock selected as MCO2 source
  - RCC_MCO2SOURCE_LSE LSE clock selected as MCO2 source
  - RCC_MCO2SOURCE_MSI MSI clock selected as MCO2 source
  - RCC_MCO2SOURCE_HSI HSI clock selected as MCO2 source
  - RCC_MCO2SOURCE_HSE HSE clock selected as MCO2 source
  - RCC_MCO2SOURCE_IC15 IC15 clock selected as MCO2 source
  - RCC_MCO2SOURCE_IC20 IC20 clock selected as MCO2 source
  - RCC_MCO2SOURCE_SYSB Bus clock (SYSB) selected as MCO2 source
- **RCC_MCODiv:** specifies the MCOx pre-scaler. This parameter can be one of the following values:
  - RCC_MCODIV_1 up to RCC_MCODIV_16 : divider applied to MCOx clock

Return values

- **None:**

Notes

- PWR VDDIO4 Input Independent I/O supply 4 shall be previously set up for PC9 with HAL_PWREx_EnableVddIO4()
- The MCO switch to the new clock source only occurs when the previous clock source is active (dynamic switch).

HAL_RCC_EnableCSS

Function name

void HAL_RCC_EnableCSS (void)

Function description

Enable the HSE Clock Security System.

Return values

- **None:**

Notes

- If a failure is detected on the HSE oscillator clock, this oscillator is automatically disabled and an interrupt is generated to inform the software about the failure (Clock Security System Interrupt, CSSI), allowing the MCU to perform rescue operations. The CSSI is linked to the Cortex-M NMI (Non-Mask-able Interrupt) exception vector.
- The HSE Clock Security System may be enabled even if the HSE is not enabled but will be activated when the HSE is enabled and ready.
- It is not possible to disable the HSE Clock Security System.

HAL_RCC_GetCpuClockFreq

Function name

uint32_t HAL_RCC_GetCpuClockFreq (void)

Function description

Returns the CPU clock (sysa_ck) frequency.

Return values

- **CPUCLK:** frequency

Notes

- The CPU clock frequency computed by this function may be not the real frequency in the chip. It is calculated based on the predefined constant and the selected clock source:
- If CPUCLK source is HSI, function returns values based on HSI_VALUE(**)
- If CPUCLK source is MSI, function returns values based on MSI_VALUE(*)
- If CPUCLK source is HSE, function returns values based on HSE_VALUE(***)
- If CPUCLK source is IC1, function returns values based on HSI_VALUE(*), MSI_VALUE(**) or HSE_VALUE(***) multiplied/divided by the PLL factors.
- (*) HSI_VALUE is a constant defined in stm32n6xx_hal_conf.h file (default value 64 MHz) but the real value may vary depending on the variations in voltage and temperature.
- (**) MSI_VALUE is a constant defined in stm32n6xx_hal_conf.h file (default value 4 MHz) but the real value may vary depending on the variations in voltage and temperature.
- (***) HSE_VALUE is a constant defined in stm32n6xx_hal_conf.h file (default value 25 MHz), user has to ensure that HSE_VALUE is same as the real frequency of the crystal used. Otherwise, this function may have wrong result.
- The result of this function could be not correct when using fractional value for HSE crystal.
- This function can be used by the user application to compute the baud rate for the communication peripherals or configure other parameters.
- Each time CPUCLK changes, this function must be called by the user application to update the CPUCLK value. Otherwise, any configuration based on this function will be incorrect.

HAL_RCC_GetSysClockFreq

Function name

uint32_t HAL_RCC_GetSysClockFreq (void)

Function description

Returns the SYSCLK bus (sysb_ck) frequency.

Return values

- **SYSCLK:** frequency

Notes

- The system bus frequency computed by this function may be not the real frequency in the chip. It is calculated based on the predefined constant and the selected clock source:
- If SYSCLK source is HSI, function returns values based on HSI_VALUE(**)
- If SYSCLK source is MSI, function returns values based on MSI_VALUE(*)
- If SYSCLK source is HSE, function returns values based on HSE_VALUE(***)
- If SYSCLK source is IC2, function returns values based on HSI_VALUE(*), MSI_VALUE(**) or HSE_VALUE(***) multiplied/divided by the PLL factors.
- (*) HSI_VALUE is a constant defined in stm32n6xx_hal_conf.h file (default value 64 MHz) but the real value may vary depending on the variations in voltage and temperature.
- (**) MSI_VALUE is a constant defined in stm32n6xx_hal_conf.h file (default value 4 MHz) but the real value may vary depending on the variations in voltage and temperature.
- (***) HSE_VALUE is a constant defined in stm32n6xx_hal_conf.h file (default value 25 MHz), user has to ensure that HSE_VALUE is same as the real frequency of the crystal used. Otherwise, this function may have wrong result.
- The result of this function could be not correct when using fractional value for HSE crystal.
- This function can be used by the user application to compute the baud rate for the communication peripherals or configure other parameters.
- Each time SYSCLK changes, this function must be called by the user application to update the SYSCLK value. Otherwise, any configuration based on this function will be incorrect.

HAL_RCC_GetHCLKFreq

Function name

uint32_t HAL_RCC_GetHCLKFreq (void)

Function description

Return the HCLK frequency.

Return values

- **HCLK:** frequency in Hz

HAL_RCC_GetPCLK1Freq

Function name

uint32_t HAL_RCC_GetPCLK1Freq (void)

Function description

Return the PCLK1 frequency.

Return values

- **PCLK1:** frequency in Hz

Notes

- Each time PCLK1 changes, this function must be called to update the right PCLK1 value. Otherwise, any configuration based on this function will be incorrect.

HAL_RCC_GetPCLK2Freq

Function name

uint32_t HAL_RCC_GetPCLK2Freq (void)

Function description

Return the PCLK2 frequency.

Return values

- **PCLK2:** frequency in Hz

Notes

- Each time PCLK2 changes, this function must be called to update the right PCLK2 value. Otherwise, any configuration based on this function will be incorrect.

HAL_RCC_GetPCLK4Freq

Function name

uint32_t HAL_RCC_GetPCLK4Freq (void)

Function description

Return the PCLK4 frequency.

Return values

- **PCLK4:** frequency in Hz

Notes

- Each time PCLK4 changes, this function must be called to update the right PCLK4 value. Otherwise, any configuration based on this function will be incorrect.

HAL_RCC_GetPCLK5Freq

Function name

uint32_t HAL_RCC_GetPCLK5Freq (void)

Function description

Return the PCLK5 frequency.

Return values

- **PCLK5:** frequency in Hz

Notes

- Each time PCLK5 changes, this function must be called to update the right PCLK5 value. Otherwise, any configuration based on this function will be incorrect.

HAL_RCC_GetOscConfig

Function name

void HAL_RCC_GetOscConfig (RCC_OscInitTypeDef * pRCC_OscInitStruct)

Function description

Get the oscillators status configuration in RCC_OscInitStruct according to the internal RCC configuration registers.

Parameters

- **pRCC_OscInitStruct:** Pointer to an RCC_OscInitTypeDef structure that will return the configuration.

Return values

- **None:**

HAL_RCC_GetClockConfig

Function name

void HAL_RCC_GetClockConfig (RCC_ClkInitTypeDef * pRCC_ClkInitStruct)

Function description

Get the clocks status configuration in RCC_ClkInitStruct according to the internal RCC configuration registers.

Parameters

- **pRCC_ClkInitStruct:** Pointer to an RCC_ClkInitTypeDef structure that will return the configuration.

Return values

- **None:**

HAL_RCC_NMI_IRQHandler

Function name

void HAL_RCC_NMI_IRQHandler (void)

Function description

This function handles the RCC HSE CSS interrupt request.

Return values

- **None:**

Notes

- This API should be called under the NMI_Handler().

HAL_RCC_CSSCallback

Function name

void HAL_RCC_CSSCallback (void)

Function description

RCC HSE Clock Security System interrupt callback.

Return values

- **none:**

HAL_RCC_ConfigAttributes

Function name

void HAL_RCC_ConfigAttributes (uint32_t Item, uint32_t Attributes)

Function description

Configure the RCC item(s) attribute(s).

Parameters

- **Item:** Item(s) to set attributes on. This parameter can be a one value or a combination of RCC items belonging to the same group of items (eg. RCC_ITEM_GROUP_OSC)
- **Attributes:** specifies the RCC secure/privilege/public attributes. This parameter can be a combination of RCC attributes

Return values

- **None:**

Notes

- Available attributes are to control the secure, privileged, public and lock access rights of items. Default access rights are non-secure, unprivileged, non-public and unlocked.
- Secure/non-secure, privileged/unprivileged and public/non-public attributes can be set and reset from the secure/privileged state only.
- Lock Item attribute can be set from the secure/privileged state only.

HAL_RCC_GetConfigAttributes

Function name

HAL_StatusTypeDef HAL_RCC_GetConfigAttributes (uint32_t Item, uint32_t * pAttributes)

Function description

Get the attributes of an RCC item.

Parameters

- **Item:** A single item to get secure/non-secure, public/non-public, privilege/non-privilege and locked attributes from. This parameter can be a value of RCC_items.
- **pAttributes:** pointer to return the attributes (this is a combination of RCC attributes).

Return values

- **HAL:** Status.

Notes

- Secured items are available from non-secure state when set as public

56.3 RCC Firmware driver defines

The following section lists the various define and macros of the module.

56.3.1 RCC

RCC

AHB1 Peripheral Clock Enabled Status

__HAL_RCC_ADC12_IS_CLK_ENABLED

__HAL_RCC_GPDMA1_IS_CLK_ENABLED

AHB1 Peripheral Clock Sleep Enable Disable

__HAL_RCC_ADC12_CLK_SLEEP_ENABLE

__HAL_RCC_ADC12_CLK_SLEEP_DISABLE

__HAL_RCC_GPDMA1_CLK_SLEEP_ENABLE

__HAL_RCC_GPDMA1_CLK_SLEEP_DISABLE

AHB1 Peripheral Clock Sleep Enabled Status

__HAL_RCC_ADC12_IS_CLK_SLEEP_ENABLED

__HAL_RCC_GPDMA1_IS_CLK_SLEEP_ENABLED

AHB1 Peripheral Force Release Reset

__HAL_RCC_AHB1_FORCE_RESET

__HAL_RCC_AHB1_RELEASE_RESET

__HAL_RCC_GPDMA1_FORCE_RESET

__HAL_RCC_GPDMA1_RELEASE_RESET

__HAL_RCC_ADC12_FORCE_RESET

__HAL_RCC_ADC12_RELEASE_RESET

AHB1 Peripheral Clock Enable Disable

__HAL_RCC_ADC12_CLK_ENABLE

__HAL_RCC_ADC12_CLK_DISABLE

__HAL_RCC_GPDMA1_CLK_ENABLE

__HAL_RCC_GPDMA1_CLK_DISABLE

AHB2 Peripheral Clock Enabled Status

__HAL_RCC_ADF1_IS_CLK_ENABLED

__HAL_RCC_MDF1_IS_CLK_ENABLED

__HAL_RCC_RAMCFG_IS_CLK_ENABLED

AHB2 Peripheral Clock Sleep Enable Disable

__HAL_RCC_ADF1_CLK_SLEEP_ENABLE

__HAL_RCC_ADF1_CLK_SLEEP_DISABLE

__HAL_RCC_MDF1_CLK_SLEEP_ENABLE

__HAL_RCC_MDF1_CLK_SLEEP_DISABLE

__HAL_RCC_RAMCFG_CLK_SLEEP_ENABLE

__HAL_RCC_RAMCFG_CLK_SLEEP_DISABLE

AHB2 Peripheral Clock Sleep Enabled Status

__HAL_RCC_ADF1_IS_CLK_SLEEP_ENABLED

__HAL_RCC_MDF1_IS_CLK_SLEEP_ENABLED

__HAL_RCC_RAMCFG_IS_CLK_SLEEP_ENABLED

AHB2 Peripheral Force Release Reset

__HAL_RCC_AHB2_FORCE_RESET

__HAL_RCC_AHB2_RELEASE_RESET

__HAL_RCC_ADF1_FORCE_RESET

__HAL_RCC_ADF1_RELEASE_RESET

__HAL_RCC_MDF1_FORCE_RESET

__HAL_RCC_MDF1_RELEASE_RESET

__HAL_RCC_RAMCFG_FORCE_RESET

__HAL_RCC_RAMCFG_RELEASE_RESET

AHB2 Peripheral Clock Enable Disable

__HAL_RCC_ADF1_CLK_ENABLE

__HAL_RCC_ADF1_CLK_DISABLE

__HAL_RCC_MDF1_CLK_ENABLE

__HAL_RCC_MDF1_CLK_DISABLE

__HAL_RCC_RAMCFG_CLK_ENABLE

__HAL_RCC_RAMCFG_CLK_DISABLE

AHB3 Peripheral Clock Enabled Status

__HAL_RCC_CRYP_IS_CLK_ENABLED

__HAL_RCC_HASH_IS_CLK_ENABLED

__HAL_RCC_IAC_IS_CLK_ENABLED

__HAL_RCC_PKA_IS_CLK_ENABLED

__HAL_RCC_RIFSC_IS_CLK_ENABLED

__HAL_RCC_RISAF_IS_CLK_ENABLED

__HAL_RCC_RNG_IS_CLK_ENABLED

__HAL_RCC_SAES_IS_CLK_ENABLED

AHB3 Peripheral Clock Sleep Enable Disable

__HAL_RCC_CRYP_CLK_SLEEP_ENABLE

__HAL_RCC_CRYP_CLK_SLEEP_DISABLE

__HAL_RCC_HASH_CLK_SLEEP_ENABLE

__HAL_RCC_HASH_CLK_SLEEP_DISABLE

__HAL_RCC_IAC_CLK_SLEEP_ENABLE

__HAL_RCC_IAC_CLK_SLEEP_DISABLE

__HAL_RCC_PKA_CLK_SLEEP_ENABLE

__HAL_RCC_PKA_CLK_SLEEP_DISABLE

__HAL_RCC_RIFSC_CLK_SLEEP_ENABLE

__HAL_RCC_RIFSC_CLK_SLEEP_DISABLE

__HAL_RCC_RISAF_CLK_SLEEP_ENABLE

__HAL_RCC_RISAF_CLK_SLEEP_DISABLE

__HAL_RCC_RNG_CLK_SLEEP_ENABLE

__HAL_RCC_RNG_CLK_SLEEP_DISABLE

__HAL_RCC_SAES_CLK_SLEEP_ENABLE

__HAL_RCC_SAES_CLK_SLEEP_DISABLE

AHB3 Peripheral Clock Sleep Enabled Status

__HAL_RCC_CRYP_IS_CLK_SLEEP_ENABLED

__HAL_RCC_HASH_IS_CLK_SLEEP_ENABLED

__HAL_RCC_IAC_IS_CLK_SLEEP_ENABLED

__HAL_RCC_PKA_IS_CLK_SLEEP_ENABLED

__HAL_RCC_RIFSC_IS_CLK_SLEEP_ENABLED

__HAL_RCC_RISAF_IS_CLK_SLEEP_ENABLED

__HAL_RCC_RNG_IS_CLK_SLEEP_ENABLED

__HAL_RCC_SAES_IS_CLK_SLEEP_ENABLED

AHB3 Peripheral Force Release Reset

__HAL_RCC_AHB3_FORCE_RESET

__HAL_RCC_AHB3_RELEASE_RESET

__HAL_RCC_IAC_FORCE_RESET

__HAL_RCC_IAC_RELEASE_RESET

__HAL_RCC_CRYP_FORCE_RESET

__HAL_RCC_CRYP_RELEASE_RESET

__HAL_RCC_HASH_FORCE_RESET

__HAL_RCC_HASH_RELEASE_RESET

__HAL_RCC_PKA_FORCE_RESET

__HAL_RCC_PKA_RELEASE_RESET

__HAL_RCC_RNG_FORCE_RESET

__HAL_RCC_RNG_RELEASE_RESET

__HAL_RCC_SAES_FORCE_RESET

__HAL_RCC_SAES_RELEASE_RESET

AHB3 Peripheral Clock Enable Disable

__HAL_RCC_CRYP_CLK_ENABLE

__HAL_RCC_CRYP_CLK_DISABLE

__HAL_RCC_HASH_CLK_ENABLE

__HAL_RCC_HASH_CLK_DISABLE

__HAL_RCC_IAC_CLK_ENABLE

__HAL_RCC_IAC_CLK_DISABLE

__HAL_RCC_PKA_CLK_ENABLE

__HAL_RCC_PKA_CLK_DISABLE

__HAL_RCC_RIFSC_CLK_ENABLE

__HAL_RCC_RIFSC_CLK_DISABLE

__HAL_RCC_RISAF_CLK_ENABLE

__HAL_RCC_RISAF_CLK_DISABLE

__HAL_RCC_RNG_CLK_ENABLE

__HAL_RCC_RNG_CLK_DISABLE

__HAL_RCC_SAES_CLK_ENABLE

__HAL_RCC_SAES_CLK_DISABLE

AHB4 Peripheral Clock Enabled Status

__HAL_RCC_CRC_IS_CLK_ENABLED

__HAL_RCC_GPIOA_IS_CLK_ENABLED

__HAL_RCC_GPIOB_IS_CLK_ENABLED

__HAL_RCC_GPIOC_IS_CLK_ENABLED

__HAL_RCC_GPIOD_IS_CLK_ENABLED

__HAL_RCC_GPIOE_IS_CLK_ENABLED

__HAL_RCC_GPIOF_IS_CLK_ENABLED

__HAL_RCC_GPIOG_IS_CLK_ENABLED

__HAL_RCC_GPIOH_IS_CLK_ENABLED

__HAL_RCC_GPION_IS_CLK_ENABLED

__HAL_RCC_GPIOO_IS_CLK_ENABLED

__HAL_RCC_GPIOP_IS_CLK_ENABLED

__HAL_RCC_GPIOQ_IS_CLK_ENABLED

__HAL_RCC_PWR_IS_CLK_ENABLED

AHB4 Peripheral Clock Sleep Enable Disable

__HAL_RCC_CRC_CLK_SLEEP_ENABLE

__HAL_RCC_CRC_CLK_SLEEP_DISABLE

__HAL_RCC_GPIOA_CLK_SLEEP_ENABLE

__HAL_RCC_GPIOA_CLK_SLEEP_DISABLE

__HAL_RCC_GPIOB_CLK_SLEEP_ENABLE

__HAL_RCC_GPIOB_CLK_SLEEP_DISABLE

__HAL_RCC_GPIOC_CLK_SLEEP_ENABLE

__HAL_RCC_GPIOC_CLK_SLEEP_DISABLE

__HAL_RCC_GPIOD_CLK_SLEEP_ENABLE

__HAL_RCC_GPIOD_CLK_SLEEP_DISABLE

__HAL_RCC_GPIOE_CLK_SLEEP_ENABLE

__HAL_RCC_GPIOE_CLK_SLEEP_DISABLE

__HAL_RCC_GPIOF_CLK_SLEEP_ENABLE

__HAL_RCC_GPIOF_CLK_SLEEP_DISABLE

__HAL_RCC_GPIOG_CLK_SLEEP_ENABLE

__HAL_RCC_GPIOG_CLK_SLEEP_DISABLE

__HAL_RCC_GPIOH_CLK_SLEEP_ENABLE

__HAL_RCC_GPIOH_CLK_SLEEP_DISABLE

__HAL_RCC_GPION_CLK_SLEEP_ENABLE

__HAL_RCC_GPION_CLK_SLEEP_DISABLE

__HAL_RCC_GPIOO_CLK_SLEEP_ENABLE

__HAL_RCC_GPIOO_CLK_SLEEP_DISABLE

__HAL_RCC_GPIOP_CLK_SLEEP_ENABLE

__HAL_RCC_GPIOP_CLK_SLEEP_DISABLE

__HAL_RCC_GPIOQ_CLK_SLEEP_ENABLE

__HAL_RCC_GPIOQ_CLK_SLEEP_DISABLE

__HAL_RCC_PWR_CLK_SLEEP_ENABLE

__HAL_RCC_PWR_CLK_SLEEP_DISABLE

AHB4 Peripheral Clock Sleep Enabled Status

__HAL_RCC_CRC_IS_CLK_SLEEP_ENABLED

__HAL_RCC_GPIOA_IS_CLK_SLEEP_ENABLED

__HAL_RCC_GPIOB_IS_CLK_SLEEP_ENABLED

__HAL_RCC_GPIOC_IS_CLK_SLEEP_ENABLED

__HAL_RCC_GPIOD_IS_CLK_SLEEP_ENABLED

__HAL_RCC_GPIOE_IS_CLK_SLEEP_ENABLED

__HAL_RCC_GPIOF_IS_CLK_SLEEP_ENABLED

__HAL_RCC_GPIOG_IS_CLK_SLEEP_ENABLED

__HAL_RCC_GPIOH_IS_CLK_SLEEP_ENABLED

__HAL_RCC_GPION_IS_CLK_SLEEP_ENABLED

__HAL_RCC_GPIOO_IS_CLK_SLEEP_ENABLED

__HAL_RCC_GPIOP_IS_CLK_SLEEP_ENABLED

__HAL_RCC_GPIOQ_IS_CLK_SLEEP_ENABLED

__HAL_RCC_PWR_IS_CLK_SLEEP_ENABLED

AHB4 Peripheral Force Release Reset

__HAL_RCC_AHB4_FORCE_RESET

__HAL_RCC_AHB4_RELEASE_RESET

__HAL_RCC_CRC_FORCE_RESET

__HAL_RCC_CRC_RELEASE_RESET

__HAL_RCC_GPIOA_FORCE_RESET

__HAL_RCC_GPIOA_RELEASE_RESET

__HAL_RCC_GPIOB_FORCE_RESET

__HAL_RCC_GPIOB_RELEASE_RESET

__HAL_RCC_GPIOC_FORCE_RESET

__HAL_RCC_GPIOC_RELEASE_RESET

__HAL_RCC_GPIOD_FORCE_RESET

__HAL_RCC_GPIOD_RELEASE_RESET

__HAL_RCC_GPIOE_FORCE_RESET

__HAL_RCC_GPIOE_RELEASE_RESET

__HAL_RCC_GPIOF_FORCE_RESET

__HAL_RCC_GPIOF_RELEASE_RESET

__HAL_RCC_GPIOG_FORCE_RESET

__HAL_RCC_GPIOG_RELEASE_RESET

__HAL_RCC_GPIOH_FORCE_RESET

__HAL_RCC_GPIOH_RELEASE_RESET

__HAL_RCC_GPION_FORCE_RESET

__HAL_RCC_GPION_RELEASE_RESET

__HAL_RCC_GPIOO_FORCE_RESET

__HAL_RCC_GPIOO_RELEASE_RESET

__HAL_RCC_GPIOP_FORCE_RESET

__HAL_RCC_GPIOP_RELEASE_RESET

__HAL_RCC_GPIOQ_FORCE_RESET

__HAL_RCC_GPIOQ_RELEASE_RESET

__HAL_RCC_PWR_FORCE_RESET

__HAL_RCC_PWR_RELEASE_RESET

AHB4 Peripheral Clock Enable Disable

__HAL_RCC_CRC_CLK_ENABLE

__HAL_RCC_CRC_CLK_DISABLE

__HAL_RCC_GPIOA_CLK_ENABLE

__HAL_RCC_GPIOA_CLK_DISABLE

__HAL_RCC_GPIOB_CLK_ENABLE

__HAL_RCC_GPIOB_CLK_DISABLE

__HAL_RCC_GPIOC_CLK_ENABLE

__HAL_RCC_GPIOC_CLK_DISABLE

__HAL_RCC_GPIOD_CLK_ENABLE

__HAL_RCC_GPIOD_CLK_DISABLE

__HAL_RCC_GPIOE_CLK_ENABLE

__HAL_RCC_GPIOE_CLK_DISABLE

__HAL_RCC_GPIOF_CLK_ENABLE

__HAL_RCC_GPIOF_CLK_DISABLE

__HAL_RCC_GPIOG_CLK_ENABLE

__HAL_RCC_GPIOG_CLK_DISABLE

__HAL_RCC_GPIOH_CLK_ENABLE

__HAL_RCC_GPIOH_CLK_DISABLE

__HAL_RCC_GPION_CLK_ENABLE

__HAL_RCC_GPION_CLK_DISABLE

__HAL_RCC_GPIOO_CLK_ENABLE

__HAL_RCC_GPIOO_CLK_DISABLE

__HAL_RCC_GPIOP_CLK_ENABLE

__HAL_RCC_GPIOP_CLK_DISABLE

__HAL_RCC_GPIOQ_CLK_ENABLE

__HAL_RCC_GPIOQ_CLK_DISABLE

__HAL_RCC_PWR_CLK_ENABLE

__HAL_RCC_PWR_CLK_DISABLE

AHB5 Peripheral Clock Enabled Status

__HAL_RCC_DMA2D_IS_CLK_ENABLED

__HAL_RCC_ETH1_IS_CLK_ENABLED

__HAL_RCC_ETH1MAC_IS_CLK_ENABLED

__HAL_RCC_ETH1TX_IS_CLK_ENABLED

__HAL_RCC_ETH1RX_IS_CLK_ENABLED

__HAL_RCC_FMC_IS_CLK_ENABLED

__HAL_RCC_GFXMMU_IS_CLK_ENABLED

__HAL_RCC_GPU2D_IS_CLK_ENABLED

__HAL_RCC_HPDMA1_IS_CLK_ENABLED

__HAL_RCC_JPEG_IS_CLK_ENABLED

__HAL_RCC_XSPI1_IS_CLK_ENABLED

__HAL_RCC_XSPI2_IS_CLK_ENABLED

__HAL_RCC_XSPI3_IS_CLK_ENABLED

__HAL_RCC_XSPIM_IS_CLK_ENABLED

__HAL_RCC_MCE1_IS_CLK_ENABLED

__HAL_RCC_MCE2_IS_CLK_ENABLED

__HAL_RCC_MCE3_IS_CLK_ENABLED

__HAL_RCC_MCE4_IS_CLK_ENABLED

__HAL_RCC_CACHEAXI_IS_CLK_ENABLED

__HAL_RCC_NPU_IS_CLK_ENABLED

__HAL_RCC_DCMI_PSSI_IS_CLK_ENABLED

__HAL_RCC_SDMMC1_IS_CLK_ENABLED

__HAL_RCC_SDMMC2_IS_CLK_ENABLED

__HAL_RCC_USB1_OTG_HS_IS_CLK_ENABLED

__HAL_RCC_USB1_OTG_HS_PHY_IS_CLK_ENABLED

__HAL_RCC_USB2_OTG_HS_IS_CLK_ENABLED

__HAL_RCC_USB2_OTG_HS_PHY_IS_CLK_ENABLED

AHB5 Peripheral Clock Sleep Enable Disable

__HAL_RCC_DMA2D_CLK_SLEEP_ENABLE

__HAL_RCC_DMA2D_CLK_SLEEP_DISABLE

__HAL_RCC_ETH1_CLK_SLEEP_ENABLE

__HAL_RCC_ETH1_CLK_SLEEP_DISABLE

__HAL_RCC_ETH1MAC_CLK_SLEEP_ENABLE

__HAL_RCC_ETH1MAC_CLK_SLEEP_DISABLE

__HAL_RCC_ETH1TX_CLK_SLEEP_ENABLE

__HAL_RCC_ETH1TX_CLK_SLEEP_DISABLE

__HAL_RCC_ETH1RX_CLK_SLEEP_ENABLE

__HAL_RCC_ETH1RX_CLK_SLEEP_DISABLE

__HAL_RCC_FMC_CLK_SLEEP_ENABLE

__HAL_RCC_FMC_CLK_SLEEP_DISABLE

__HAL_RCC_GFXMMU_CLK_SLEEP_ENABLE

__HAL_RCC_GFXMMU_CLK_SLEEP_DISABLE

__HAL_RCC_GPU2D_CLK_SLEEP_ENABLE

__HAL_RCC_GPU2D_CLK_SLEEP_DISABLE

__HAL_RCC_HPDMA1_CLK_SLEEP_ENABLE

__HAL_RCC_HPDMA1_CLK_SLEEP_DISABLE

__HAL_RCC_JPEG_CLK_SLEEP_ENABLE

__HAL_RCC_JPEG_CLK_SLEEP_DISABLE

__HAL_RCC_XSPI1_CLK_SLEEP_ENABLE

__HAL_RCC_XSPI1_CLK_SLEEP_DISABLE

__HAL_RCC_XSPI2_CLK_SLEEP_ENABLE

__HAL_RCC_XSPI2_CLK_SLEEP_DISABLE

__HAL_RCC_XSPI3_CLK_SLEEP_ENABLE

__HAL_RCC_XSPI3_CLK_SLEEP_DISABLE

__HAL_RCC_XSPIM_CLK_SLEEP_ENABLE

__HAL_RCC_XSPIM_CLK_SLEEP_DISABLE

__HAL_RCC_MCE1_CLK_SLEEP_ENABLE

__HAL_RCC_MCE1_CLK_SLEEP_DISABLE

__HAL_RCC_MCE2_CLK_SLEEP_ENABLE

__HAL_RCC_MCE2_CLK_SLEEP_DISABLE

__HAL_RCC_MCE3_CLK_SLEEP_ENABLE

__HAL_RCC_MCE3_CLK_SLEEP_DISABLE

__HAL_RCC_MCE4_CLK_SLEEP_ENABLE

__HAL_RCC_MCE4_CLK_SLEEP_DISABLE

__HAL_RCC_CACHEAXI_CLK_SLEEP_ENABLE

__HAL_RCC_CACHEAXI_CLK_SLEEP_DISABLE

__HAL_RCC_NPU_CLK_SLEEP_ENABLE

__HAL_RCC_NPU_CLK_SLEEP_DISABLE

__HAL_RCC_DCMI_PSSI_CLK_SLEEP_ENABLE

__HAL_RCC_DCMI_PSSI_CLK_SLEEP_DISABLE

__HAL_RCC_SDMMC1_CLK_SLEEP_ENABLE

__HAL_RCC_SDMMC1_CLK_SLEEP_DISABLE

__HAL_RCC_SDMMC2_CLK_SLEEP_ENABLE

__HAL_RCC_SDMMC2_CLK_SLEEP_DISABLE

__HAL_RCC_USB1_OTG_HS_CLK_SLEEP_ENABLE

__HAL_RCC_USB1_OTG_HS_CLK_SLEEP_DISABLE

__HAL_RCC_USB1_OTG_HS_PHY_CLK_SLEEP_ENABLE

__HAL_RCC_USB1_OTG_HS_PHY_CLK_SLEEP_DISABLE

__HAL_RCC_USB2_OTG_HS_CLK_SLEEP_ENABLE

__HAL_RCC_USB2_OTG_HS_CLK_SLEEP_DISABLE

__HAL_RCC_USB2_OTG_HS_PHY_CLK_SLEEP_ENABLE

__HAL_RCC_USB2_OTG_HS_PHY_CLK_SLEEP_DISABLE

AHB5 Peripheral Clock Sleep Enabled Status

__HAL_RCC_DMA2D_IS_CLK_SLEEP_ENABLED

__HAL_RCC_ETH1_IS_CLK_SLEEP_ENABLED

__HAL_RCC_ETH1MAC_IS_CLK_SLEEP_ENABLED

__HAL_RCC_ETH1TX_IS_CLK_SLEEP_ENABLED

__HAL_RCC_ETH1RX_IS_CLK_SLEEP_ENABLED

__HAL_RCC_FMC_IS_CLK_SLEEP_ENABLED

__HAL_RCC_GFXMMU_IS_CLK_SLEEP_ENABLED

__HAL_RCC_GPU2D_IS_CLK_SLEEP_ENABLED

__HAL_RCC_HPDMA1_IS_CLK_SLEEP_ENABLED

__HAL_RCC_JPEG_IS_CLK_SLEEP_ENABLED

__HAL_RCC_XSPI1_IS_CLK_SLEEP_ENABLED

__HAL_RCC_XSPI2_IS_CLK_SLEEP_ENABLED

__HAL_RCC_XSPI3_IS_CLK_SLEEP_ENABLED

__HAL_RCC_XSPIM_IS_CLK_SLEEP_ENABLED

__HAL_RCC_MCE1_IS_CLK_SLEEP_ENABLED

__HAL_RCC_MCE2_IS_CLK_SLEEP_ENABLED

__HAL_RCC_MCE3_IS_CLK_SLEEP_ENABLED

__HAL_RCC_MCE4_IS_CLK_SLEEP_ENABLED

__HAL_RCC_CACHEAXI_IS_CLK_SLEEP_ENABLED

__HAL_RCC_NPU_IS_CLK_SLEEP_ENABLED

__HAL_RCC_DCMI_PSSI_IS_CLK_SLEEP_ENABLED

__HAL_RCC_SDMMC1_IS_CLK_SLEEP_ENABLED

__HAL_RCC_SDMMC2_IS_CLK_SLEEP_ENABLED

__HAL_RCC_USB1_OTG_HS_IS_CLK_SLEEP_ENABLED

__HAL_RCC_USB1_OTG_HS_PHY_IS_CLK_SLEEP_ENABLED

__HAL_RCC_USB2_OTG_HS_IS_CLK_SLEEP_ENABLED

__HAL_RCC_USB2_OTG_HS_PHY_IS_CLK_SLEEP_ENABLED

AHB5 Peripheral Force Release Reset

__HAL_RCC_AHB5_FORCE_RESET

__HAL_RCC_AHB5_RELEASE_RESET

__HAL_RCC_DMA2D_FORCE_RESET

__HAL_RCC_DMA2D_RELEASE_RESET

__HAL_RCC_ETH1_FORCE_RESET

__HAL_RCC_ETH1_RELEASE_RESET

__HAL_RCC_FMC_FORCE_RESET

__HAL_RCC_FMC_RELEASE_RESET

__HAL_RCC_GFXMMU_FORCE_RESET

__HAL_RCC_GFXMMU_RELEASE_RESET

__HAL_RCC_GPU2D_FORCE_RESET

__HAL_RCC_GPU2D_RELEASE_RESET

__HAL_RCC_HPDMA1_FORCE_RESET

__HAL_RCC_HPDMA1_RELEASE_RESET

__HAL_RCC_XSPI1_FORCE_RESET

__HAL_RCC_XSPI1_RELEASE_RESET

__HAL_RCC_XSPI2_FORCE_RESET

__HAL_RCC_XSPI2_RELEASE_RESET

__HAL_RCC_XSPI3_FORCE_RESET

__HAL_RCC_XSPI3_RELEASE_RESET

__HAL_RCC_XSPIM_FORCE_RESET

__HAL_RCC_XSPIM_RELEASE_RESET

__HAL_RCC_JPEG_FORCE_RESET

__HAL_RCC_JPEG_RELEASE_RESET

__HAL_RCC_CACHEAXI_FORCE_RESET

__HAL_RCC_CACHEAXI_RELEASE_RESET

__HAL_RCC_NPU_FORCE_RESET

__HAL_RCC_NPU_RELEASE_RESET

__HAL_RCC_DCMI_PSSI_FORCE_RESET

__HAL_RCC_DCMI_PSSI_RELEASE_RESET

__HAL_RCC_SDMMC1_FORCE_RESET

__HAL_RCC_SDMMC1_RELEASE_RESET

__HAL_RCC_SDMMC2_FORCE_RESET

__HAL_RCC_SDMMC2_RELEASE_RESET

__HAL_RCC_USB1_OTG_HS_FORCE_RESET

__HAL_RCC_USB1_OTG_HS_RELEASE_RESET

__HAL_RCC_USB1_OTG_HS_PHY_FORCE_RESET

__HAL_RCC_USB1_OTG_HS_PHY_RELEASE_RESET

__HAL_RCC_USB1_OTG_HS_CTL_FORCE_RESET

__HAL_RCC_USB1_OTG_HS_CTL_RELEASE_RESET

__HAL_RCC_USB2_OTG_HS_FORCE_RESET

__HAL_RCC_USB2_OTG_HS_RELEASE_RESET

__HAL_RCC_USB2_OTG_HS_PHY_FORCE_RESET

__HAL_RCC_USB2_OTG_HS_PHY_RELEASE_RESET

__HAL_RCC_USB2_OTG_HS_CTL_FORCE_RESET

__HAL_RCC_USB2_OTG_HS_CTL_RELEASE_RESET

AHB5 Peripheral Clock Enable Disable

__HAL_RCC_DMA2D_CLK_ENABLE

__HAL_RCC_DMA2D_CLK_DISABLE

__HAL_RCC_ETH1_CLK_ENABLE

__HAL_RCC_ETH1_CLK_DISABLE

__HAL_RCC_ETH1MAC_CLK_ENABLE

__HAL_RCC_ETH1MAC_CLK_DISABLE

__HAL_RCC_ETH1TX_CLK_ENABLE

__HAL_RCC_ETH1TX_CLK_DISABLE

__HAL_RCC_ETH1RX_CLK_ENABLE

__HAL_RCC_ETH1RX_CLK_DISABLE

__HAL_RCC_FMC_CLK_ENABLE

__HAL_RCC_FMC_CLK_DISABLE

__HAL_RCC_GFXMMU_CLK_ENABLE

__HAL_RCC_GFXMMU_CLK_DISABLE

__HAL_RCC_GPU2D_CLK_ENABLE

__HAL_RCC_GPU2D_CLK_DISABLE

__HAL_RCC_HPDMA1_CLK_ENABLE

__HAL_RCC_HPDMA1_CLK_DISABLE

__HAL_RCC_JPEG_CLK_ENABLE

__HAL_RCC_JPEG_CLK_DISABLE

__HAL_RCC_XSPI1_CLK_ENABLE

__HAL_RCC_XSPI1_CLK_DISABLE

__HAL_RCC_XSPI2_CLK_ENABLE

__HAL_RCC_XSPI2_CLK_DISABLE

__HAL_RCC_XSPI3_CLK_ENABLE

__HAL_RCC_XSPI3_CLK_DISABLE

__HAL_RCC_XSPIM_CLK_ENABLE

__HAL_RCC_XSPIM_CLK_DISABLE

__HAL_RCC_MCE1_CLK_ENABLE

__HAL_RCC_MCE1_CLK_DISABLE

__HAL_RCC_MCE2_CLK_ENABLE

__HAL_RCC_MCE2_CLK_DISABLE

__HAL_RCC_MCE3_CLK_ENABLE

__HAL_RCC_MCE3_CLK_DISABLE

__HAL_RCC_MCE4_CLK_ENABLE

__HAL_RCC_MCE4_CLK_DISABLE

__HAL_RCC_CACHEAXI_CLK_ENABLE

__HAL_RCC_CACHEAXI_CLK_DISABLE

__HAL_RCC_NPU_CLK_ENABLE

__HAL_RCC_NPU_CLK_DISABLE

__HAL_RCC_DCMI_PSSI_CLK_ENABLE

__HAL_RCC_DCMI_PSSI_CLK_DISABLE

__HAL_RCC_SDMMC1_CLK_ENABLE

__HAL_RCC_SDMMC1_CLK_DISABLE

__HAL_RCC_SDMMC2_CLK_ENABLE

__HAL_RCC_SDMMC2_CLK_DISABLE

__HAL_RCC_USB1_OTG_HS_CLK_ENABLE

__HAL_RCC_USB1_OTG_HS_CLK_DISABLE

__HAL_RCC_USB1_OTG_HS_PHY_CLK_ENABLE

__HAL_RCC_USB1_OTG_HS_PHY_CLK_DISABLE

__HAL_RCC_USB2_OTG_HS_CLK_ENABLE

__HAL_RCC_USB2_OTG_HS_CLK_DISABLE

__HAL_RCC_USB2_OTG_HS_PHY_CLK_ENABLE

__HAL_RCC_USB2_OTG_HS_PHY_CLK_DISABLE

APB1 Peripheral Clock Enabled Status

__HAL_RCC_FDCAN_IS_CLK_ENABLED

__HAL_RCC_I2C1_IS_CLK_ENABLED

__HAL_RCC_I2C2_IS_CLK_ENABLED

__HAL_RCC_I2C3_IS_CLK_ENABLED

__HAL_RCC_I3C1_IS_CLK_ENABLED

__HAL_RCC_I3C2_IS_CLK_ENABLED

__HAL_RCC_LPTIM1_IS_CLK_ENABLED

__HAL_RCC_MDIOS_IS_CLK_ENABLED

__HAL_RCC_SPDIFRX1_IS_CLK_ENABLED

__HAL_RCC_SPI2_IS_CLK_ENABLED

__HAL_RCC_SPI3_IS_CLK_ENABLED

__HAL_RCC_TIM2_IS_CLK_ENABLED

__HAL_RCC_TIM3_IS_CLK_ENABLED

__HAL_RCC_TIM4_IS_CLK_ENABLED

__HAL_RCC_TIM5_IS_CLK_ENABLED

__HAL_RCC_TIM6_IS_CLK_ENABLED

__HAL_RCC_TIM7_IS_CLK_ENABLED

__HAL_RCC_TIM10_IS_CLK_ENABLED

__HAL_RCC_TIM11_IS_CLK_ENABLED

__HAL_RCC_TIM12_IS_CLK_ENABLED

__HAL_RCC_TIM13_IS_CLK_ENABLED

__HAL_RCC_TIM14_IS_CLK_ENABLED

__HAL_RCC_USART2_IS_CLK_ENABLED

__HAL_RCC_USART3_IS_CLK_ENABLED

__HAL_RCC_UART4_IS_CLK_ENABLED

__HAL_RCC_UART5_IS_CLK_ENABLED

__HAL_RCC_UART7_IS_CLK_ENABLED

__HAL_RCC_UART8_IS_CLK_ENABLED

__HAL_RCC_UCPD1_IS_CLK_ENABLED

__HAL_RCC_WWDG_IS_CLK_ENABLED

APB1 Peripheral Clock Sleep Enable Disable

__HAL_RCC_FDCAN_CLK_SLEEP_ENABLE

__HAL_RCC_FDCAN_CLK_SLEEP_DISABLE

__HAL_RCC_I2C1_CLK_SLEEP_ENABLE

__HAL_RCC_I2C1_CLK_SLEEP_DISABLE

__HAL_RCC_I2C2_CLK_SLEEP_ENABLE

__HAL_RCC_I2C2_CLK_SLEEP_DISABLE

__HAL_RCC_I2C3_CLK_SLEEP_ENABLE

__HAL_RCC_I2C3_CLK_SLEEP_DISABLE

__HAL_RCC_I3C1_CLK_SLEEP_ENABLE

__HAL_RCC_I3C1_CLK_SLEEP_DISABLE

__HAL_RCC_I3C2_CLK_SLEEP_ENABLE

__HAL_RCC_I3C2_CLK_SLEEP_DISABLE

__HAL_RCC_LPTIM1_CLK_SLEEP_ENABLE

__HAL_RCC_LPTIM1_CLK_SLEEP_DISABLE

__HAL_RCC_MDIOS_CLK_SLEEP_ENABLE

__HAL_RCC_MDIOS_CLK_SLEEP_DISABLE

__HAL_RCC_SPDIFRX1_CLK_SLEEP_ENABLE

__HAL_RCC_SPDIFRX1_CLK_SLEEP_DISABLE

__HAL_RCC_SPI2_CLK_SLEEP_ENABLE

__HAL_RCC_SPI2_CLK_SLEEP_DISABLE

__HAL_RCC_SPI3_CLK_SLEEP_ENABLE

__HAL_RCC_SPI3_CLK_SLEEP_DISABLE

__HAL_RCC_TIM2_CLK_SLEEP_ENABLE

__HAL_RCC_TIM2_CLK_SLEEP_DISABLE

__HAL_RCC_TIM3_CLK_SLEEP_ENABLE

__HAL_RCC_TIM3_CLK_SLEEP_DISABLE

__HAL_RCC_TIM4_CLK_SLEEP_ENABLE

__HAL_RCC_TIM4_CLK_SLEEP_DISABLE

__HAL_RCC_TIM5_CLK_SLEEP_ENABLE

__HAL_RCC_TIM5_CLK_SLEEP_DISABLE

__HAL_RCC_TIM6_CLK_SLEEP_ENABLE

__HAL_RCC_TIM6_CLK_SLEEP_DISABLE

__HAL_RCC_TIM7_CLK_SLEEP_ENABLE

__HAL_RCC_TIM7_CLK_SLEEP_DISABLE

__HAL_RCC_TIM10_CLK_SLEEP_ENABLE

__HAL_RCC_TIM10_CLK_SLEEP_DISABLE

__HAL_RCC_TIM11_CLK_SLEEP_ENABLE

__HAL_RCC_TIM11_CLK_SLEEP_DISABLE

__HAL_RCC_TIM12_CLK_SLEEP_ENABLE

__HAL_RCC_TIM12_CLK_SLEEP_DISABLE

__HAL_RCC_TIM13_CLK_SLEEP_ENABLE

__HAL_RCC_TIM13_CLK_SLEEP_DISABLE

__HAL_RCC_TIM14_CLK_SLEEP_ENABLE

__HAL_RCC_TIM14_CLK_SLEEP_DISABLE

__HAL_RCC_USART2_CLK_SLEEP_ENABLE

__HAL_RCC_USART2_CLK_SLEEP_DISABLE

__HAL_RCC_USART3_CLK_SLEEP_ENABLE

__HAL_RCC_USART3_CLK_SLEEP_DISABLE

__HAL_RCC_UART4_CLK_SLEEP_ENABLE

__HAL_RCC_UART4_CLK_SLEEP_DISABLE

__HAL_RCC_UART5_CLK_SLEEP_ENABLE

__HAL_RCC_UART5_CLK_SLEEP_DISABLE

__HAL_RCC_UART7_CLK_SLEEP_ENABLE

__HAL_RCC_UART7_CLK_SLEEP_DISABLE

__HAL_RCC_UART8_CLK_SLEEP_ENABLE

__HAL_RCC_UART8_CLK_SLEEP_DISABLE

__HAL_RCC_UCPD1_CLK_SLEEP_ENABLE

__HAL_RCC_UCPD1_CLK_SLEEP_DISABLE

__HAL_RCC_WWDG_CLK_SLEEP_ENABLE

__HAL_RCC_WWDG_CLK_SLEEP_DISABLE

APB1 Peripheral Clock Sleep Enabled Status

__HAL_RCC_FDCAN_IS_CLK_SLEEP_ENABLED

__HAL_RCC_I2C1_IS_CLK_SLEEP_ENABLED

__HAL_RCC_I2C2_IS_CLK_SLEEP_ENABLED

__HAL_RCC_I2C3_IS_CLK_SLEEP_ENABLED

__HAL_RCC_I3C1_IS_CLK_SLEEP_ENABLED

__HAL_RCC_I3C2_IS_CLK_SLEEP_ENABLED

__HAL_RCC_LPTIM1_IS_CLK_SLEEP_ENABLED

__HAL_RCC_MDIOS_IS_CLK_SLEEP_ENABLED

__HAL_RCC_SPDIFRX1_IS_CLK_SLEEP_ENABLED

__HAL_RCC_SPI2_IS_CLK_SLEEP_ENABLED

__HAL_RCC_SPI3_IS_CLK_SLEEP_ENABLED

__HAL_RCC_TIM2_IS_CLK_SLEEP_ENABLED

__HAL_RCC_TIM3_IS_CLK_SLEEP_ENABLED

__HAL_RCC_TIM4_IS_CLK_SLEEP_ENABLED

__HAL_RCC_TIM5_IS_CLK_SLEEP_ENABLED

__HAL_RCC_TIM6_IS_CLK_SLEEP_ENABLED

__HAL_RCC_TIM7_IS_CLK_SLEEP_ENABLED

__HAL_RCC_TIM10_IS_CLK_SLEEP_ENABLED

__HAL_RCC_TIM11_IS_CLK_SLEEP_ENABLED

__HAL_RCC_TIM12_IS_CLK_SLEEP_ENABLED

__HAL_RCC_TIM13_IS_CLK_SLEEP_ENABLED

__HAL_RCC_TIM14_IS_CLK_SLEEP_ENABLED

__HAL_RCC_USART2_IS_CLK_SLEEP_ENABLED

__HAL_RCC_USART3_IS_CLK_SLEEP_ENABLED

__HAL_RCC_UART4_IS_CLK_SLEEP_ENABLED

__HAL_RCC_UART5_IS_CLK_SLEEP_ENABLED

__HAL_RCC_UART7_IS_CLK_SLEEP_ENABLED

__HAL_RCC_UART8_IS_CLK_SLEEP_ENABLED

__HAL_RCC_UCPD1_IS_CLK_SLEEP_ENABLED

__HAL_RCC_WWDG_IS_CLK_SLEEP_ENABLED

APB1 Clock Source

RCC_APB1_DIV1

APB1 not divided

RCC_APB1_DIV2

APB1 divided by 2

RCC_APB1_DIV4

APB1 divided by 4

RCC_APB1_DIV8

APB1 divided by 8

RCC_APB1_DIV16

APB1 divided by 16

RCC_APB1_DIV32

APB1 divided by 32

RCC_APB1_DIV64

APB1 divided by 64

RCC_APB1_DIV128

APB1 divided by 128

APB1 Peripheral Force Release Reset

__HAL_RCC_APB1_FORCE_RESET

__HAL_RCC_APB1_RELEASE_RESET

__HAL_RCC_FDCAN_FORCE_RESET

__HAL_RCC_FDCAN_RELEASE_RESET

__HAL_RCC_I2C1_FORCE_RESET

__HAL_RCC_I2C1_RELEASE_RESET

__HAL_RCC_I2C2_FORCE_RESET

__HAL_RCC_I2C2_RELEASE_RESET

__HAL_RCC_I2C3_FORCE_RESET

__HAL_RCC_I2C3_RELEASE_RESET

__HAL_RCC_I3C1_FORCE_RESET

__HAL_RCC_I3C1_RELEASE_RESET

__HAL_RCC_I3C2_FORCE_RESET

__HAL_RCC_I3C2_RELEASE_RESET

__HAL_RCC_LPTIM1_FORCE_RESET

__HAL_RCC_LPTIM1_RELEASE_RESET

__HAL_RCC_MDIOS_FORCE_RESET

__HAL_RCC_MDIOS_RELEASE_RESET

__HAL_RCC_SPDIFRX1_FORCE_RESET

__HAL_RCC_SPDIFRX1_RELEASE_RESET

__HAL_RCC_SPI2_FORCE_RESET

__HAL_RCC_SPI2_RELEASE_RESET

__HAL_RCC_SPI3_FORCE_RESET

__HAL_RCC_SPI3_RELEASE_RESET

__HAL_RCC_TIM2_FORCE_RESET

__HAL_RCC_TIM2_RELEASE_RESET

__HAL_RCC_TIM3_FORCE_RESET

__HAL_RCC_TIM3_RELEASE_RESET

__HAL_RCC_TIM4_FORCE_RESET

__HAL_RCC_TIM4_RELEASE_RESET

__HAL_RCC_TIM5_FORCE_RESET

__HAL_RCC_TIM5_RELEASE_RESET

__HAL_RCC_TIM6_FORCE_RESET

__HAL_RCC_TIM6_RELEASE_RESET

__HAL_RCC_TIM7_FORCE_RESET

__HAL_RCC_TIM7_RELEASE_RESET

__HAL_RCC_TIM10_FORCE_RESET

__HAL_RCC_TIM10_RELEASE_RESET

__HAL_RCC_TIM11_FORCE_RESET

__HAL_RCC_TIM11_RELEASE_RESET

__HAL_RCC_TIM12_FORCE_RESET

__HAL_RCC_TIM12_RELEASE_RESET

__HAL_RCC_TIM13_FORCE_RESET

__HAL_RCC_TIM13_RELEASE_RESET

__HAL_RCC_TIM14_FORCE_RESET

__HAL_RCC_TIM14_RELEASE_RESET

__HAL_RCC_UCPD1_FORCE_RESET

__HAL_RCC_UCPD1_RELEASE_RESET

__HAL_RCC_USART2_FORCE_RESET

__HAL_RCC_USART2_RELEASE_RESET

__HAL_RCC_USART3_FORCE_RESET

__HAL_RCC_USART3_RELEASE_RESET

__HAL_RCC_UART4_FORCE_RESET

__HAL_RCC_UART4_RELEASE_RESET

__HAL_RCC_UART5_FORCE_RESET

__HAL_RCC_UART5_RELEASE_RESET

__HAL_RCC_UART7_FORCE_RESET

__HAL_RCC_UART7_RELEASE_RESET

__HAL_RCC_UART8_FORCE_RESET

__HAL_RCC_UART8_RELEASE_RESET

APB1 Peripheral Clock Enable Disable

__HAL_RCC_FDCAN_CLK_ENABLE

__HAL_RCC_FDCAN_CLK_DISABLE

__HAL_RCC_I2C1_CLK_ENABLE

__HAL_RCC_I2C1_CLK_DISABLE

__HAL_RCC_I2C2_CLK_ENABLE

__HAL_RCC_I2C2_CLK_DISABLE

__HAL_RCC_I2C3_CLK_ENABLE

__HAL_RCC_I2C3_CLK_DISABLE

__HAL_RCC_I3C1_CLK_ENABLE

__HAL_RCC_I3C1_CLK_DISABLE

__HAL_RCC_I3C2_CLK_ENABLE

__HAL_RCC_I3C2_CLK_DISABLE

__HAL_RCC_LPTIM1_CLK_ENABLE

__HAL_RCC_LPTIM1_CLK_DISABLE

__HAL_RCC_MDIOS_CLK_ENABLE

__HAL_RCC_MDIOS_CLK_DISABLE

__HAL_RCC_SPDIFRX1_CLK_ENABLE

__HAL_RCC_SPDIFRX1_CLK_DISABLE

__HAL_RCC_SPI2_CLK_ENABLE

__HAL_RCC_SPI2_CLK_DISABLE

__HAL_RCC_SPI3_CLK_ENABLE

__HAL_RCC_SPI3_CLK_DISABLE

__HAL_RCC_TIM2_CLK_ENABLE

__HAL_RCC_TIM2_CLK_DISABLE

__HAL_RCC_TIM3_CLK_ENABLE

__HAL_RCC_TIM3_CLK_DISABLE

__HAL_RCC_TIM4_CLK_ENABLE

__HAL_RCC_TIM4_CLK_DISABLE

__HAL_RCC_TIM5_CLK_ENABLE

__HAL_RCC_TIM5_CLK_DISABLE

__HAL_RCC_TIM6_CLK_ENABLE

__HAL_RCC_TIM6_CLK_DISABLE

__HAL_RCC_TIM7_CLK_ENABLE

__HAL_RCC_TIM7_CLK_DISABLE

__HAL_RCC_TIM10_CLK_ENABLE

__HAL_RCC_TIM10_CLK_DISABLE

__HAL_RCC_TIM11_CLK_ENABLE

__HAL_RCC_TIM11_CLK_DISABLE

__HAL_RCC_TIM12_CLK_ENABLE

__HAL_RCC_TIM12_CLK_DISABLE

__HAL_RCC_TIM13_CLK_ENABLE

__HAL_RCC_TIM13_CLK_DISABLE

__HAL_RCC_TIM14_CLK_ENABLE

__HAL_RCC_TIM14_CLK_DISABLE

__HAL_RCC_USART2_CLK_ENABLE

__HAL_RCC_USART2_CLK_DISABLE

__HAL_RCC_USART3_CLK_ENABLE

__HAL_RCC_USART3_CLK_DISABLE

__HAL_RCC_UART4_CLK_ENABLE

__HAL_RCC_UART4_CLK_DISABLE

__HAL_RCC_UART5_CLK_ENABLE

__HAL_RCC_UART5_CLK_DISABLE

__HAL_RCC_UART7_CLK_ENABLE

__HAL_RCC_UART7_CLK_DISABLE

__HAL_RCC_UART8_CLK_ENABLE

__HAL_RCC_UART8_CLK_DISABLE

__HAL_RCC_UCPD1_CLK_ENABLE

__HAL_RCC_UCPD1_CLK_DISABLE

__HAL_RCC_WWDG_CLK_ENABLE

APB2 Peripheral Clock Enabled Status

__HAL_RCC_SAI1_IS_CLK_ENABLED

__HAL_RCC_SAI2_IS_CLK_ENABLED

__HAL_RCC_SPI1_IS_CLK_ENABLED

__HAL_RCC_SPI4_IS_CLK_ENABLED

__HAL_RCC_SPI5_IS_CLK_ENABLED

__HAL_RCC_TIM1_IS_CLK_ENABLED

__HAL_RCC_TIM8_IS_CLK_ENABLED

__HAL_RCC_TIM9_IS_CLK_ENABLED

__HAL_RCC_TIM15_IS_CLK_ENABLED

__HAL_RCC_TIM16_IS_CLK_ENABLED

__HAL_RCC_TIM17_IS_CLK_ENABLED

__HAL_RCC_TIM18_IS_CLK_ENABLED

__HAL_RCC_USART1_IS_CLK_ENABLED

__HAL_RCC_USART6_IS_CLK_ENABLED

__HAL_RCC_UART9_IS_CLK_ENABLED

__HAL_RCC_USART10_IS_CLK_ENABLED

APB2 Peripheral Clock Sleep Enable Disable

__HAL_RCC_SAI1_CLK_SLEEP_ENABLE

__HAL_RCC_SAI1_CLK_SLEEP_DISABLE

__HAL_RCC_SAI2_CLK_SLEEP_ENABLE

__HAL_RCC_SAI2_CLK_SLEEP_DISABLE

__HAL_RCC_SPI1_CLK_SLEEP_ENABLE

__HAL_RCC_SPI1_CLK_SLEEP_DISABLE

__HAL_RCC_SPI4_CLK_SLEEP_ENABLE

__HAL_RCC_SPI4_CLK_SLEEP_DISABLE

__HAL_RCC_SPI5_CLK_SLEEP_ENABLE

__HAL_RCC_SPI5_CLK_SLEEP_DISABLE

__HAL_RCC_TIM1_CLK_SLEEP_ENABLE

__HAL_RCC_TIM1_CLK_SLEEP_DISABLE

__HAL_RCC_TIM8_CLK_SLEEP_ENABLE

__HAL_RCC_TIM8_CLK_SLEEP_DISABLE

__HAL_RCC_TIM9_CLK_SLEEP_ENABLE

__HAL_RCC_TIM9_CLK_SLEEP_DISABLE

__HAL_RCC_TIM15_CLK_SLEEP_ENABLE

__HAL_RCC_TIM15_CLK_SLEEP_DISABLE

__HAL_RCC_TIM16_CLK_SLEEP_ENABLE

__HAL_RCC_TIM16_CLK_SLEEP_DISABLE

__HAL_RCC_TIM17_CLK_SLEEP_ENABLE

__HAL_RCC_TIM17_CLK_SLEEP_DISABLE

__HAL_RCC_TIM18_CLK_SLEEP_ENABLE

__HAL_RCC_TIM18_CLK_SLEEP_DISABLE

__HAL_RCC_USART1_CLK_SLEEP_ENABLE

__HAL_RCC_USART1_CLK_SLEEP_DISABLE

__HAL_RCC_USART6_CLK_SLEEP_ENABLE

__HAL_RCC_USART6_CLK_SLEEP_DISABLE

__HAL_RCC_UART9_CLK_SLEEP_ENABLE

__HAL_RCC_UART9_CLK_SLEEP_DISABLE

__HAL_RCC_USART10_CLK_SLEEP_ENABLE

__HAL_RCC_USART10_CLK_SLEEP_DISABLE

APB2 Peripheral Clock Sleep Enabled Status

__HAL_RCC_SAI1_IS_CLK_SLEEP_ENABLED

__HAL_RCC_SAI2_IS_CLK_SLEEP_ENABLED

__HAL_RCC_SPI1_IS_CLK_SLEEP_ENABLED

__HAL_RCC_SPI4_IS_CLK_SLEEP_ENABLED

__HAL_RCC_SPI5_IS_CLK_SLEEP_ENABLED

__HAL_RCC_TIM1_IS_CLK_SLEEP_ENABLED

__HAL_RCC_TIM8_IS_CLK_SLEEP_ENABLED

__HAL_RCC_TIM9_IS_CLK_SLEEP_ENABLED

__HAL_RCC_TIM15_IS_CLK_SLEEP_ENABLED

__HAL_RCC_TIM16_IS_CLK_SLEEP_ENABLED

__HAL_RCC_TIM17_IS_CLK_SLEEP_ENABLED

__HAL_RCC_TIM18_IS_CLK_SLEEP_ENABLED

__HAL_RCC_USART1_IS_CLK_SLEEP_ENABLED

__HAL_RCC_USART6_IS_CLK_SLEEP_ENABLED

__HAL_RCC_UART9_IS_CLK_SLEEP_ENABLED

__HAL_RCC_USART10_IS_CLK_SLEEP_ENABLED

APB2 Clock Source

RCC_APB2_DIV1

APB2 not divided

RCC_APB2_DIV2

APB2 divided by 2

RCC_APB2_DIV4

APB2 divided by 4

RCC_APB2_DIV8

APB2 divided by 8

RCC_APB2_DIV16

APB2 divided by 16

RCC_APB2_DIV32

APB2 divided by 32

RCC_APB2_DIV64

APB2 divided by 64

RCC_APB2_DIV128

APB2 divided by 128

APB2 Peripheral Force Release Reset

__HAL_RCC_APB2_FORCE_RESET

__HAL_RCC_APB2_RELEASE_RESET

__HAL_RCC_SAI1_FORCE_RESET

__HAL_RCC_SAI1_RELEASE_RESET

__HAL_RCC_SAI2_FORCE_RESET

__HAL_RCC_SAI2_RELEASE_RESET

__HAL_RCC_SPI1_FORCE_RESET

__HAL_RCC_SPI1_RELEASE_RESET

__HAL_RCC_SPI4_FORCE_RESET

__HAL_RCC_SPI4_RELEASE_RESET

__HAL_RCC_SPI5_FORCE_RESET

__HAL_RCC_SPI5_RELEASE_RESET

__HAL_RCC_TIM1_FORCE_RESET

__HAL_RCC_TIM1_RELEASE_RESET

__HAL_RCC_TIM8_FORCE_RESET

__HAL_RCC_TIM8_RELEASE_RESET

__HAL_RCC_TIM9_FORCE_RESET

__HAL_RCC_TIM9_RELEASE_RESET

__HAL_RCC_TIM15_FORCE_RESET

__HAL_RCC_TIM15_RELEASE_RESET

__HAL_RCC_TIM16_FORCE_RESET

__HAL_RCC_TIM16_RELEASE_RESET

__HAL_RCC_TIM17_FORCE_RESET

__HAL_RCC_TIM17_RELEASE_RESET

__HAL_RCC_TIM18_FORCE_RESET

__HAL_RCC_TIM18_RELEASE_RESET

__HAL_RCC_USART1_FORCE_RESET

__HAL_RCC_USART1_RELEASE_RESET

__HAL_RCC_USART6_FORCE_RESET

__HAL_RCC_USART6_RELEASE_RESET

__HAL_RCC_UART9_FORCE_RESET

__HAL_RCC_UART9_RELEASE_RESET

__HAL_RCC_USART10_FORCE_RESET

__HAL_RCC_USART10_RELEASE_RESET

APB2 Peripheral Clock Enable Disable

__HAL_RCC_SAI1_CLK_ENABLE

__HAL_RCC_SAI1_CLK_DISABLE

__HAL_RCC_SAI2_CLK_ENABLE

__HAL_RCC_SAI2_CLK_DISABLE

__HAL_RCC_SPI1_CLK_ENABLE

__HAL_RCC_SPI1_CLK_DISABLE

__HAL_RCC_SPI4_CLK_ENABLE

__HAL_RCC_SPI4_CLK_DISABLE

__HAL_RCC_SPI5_CLK_ENABLE

__HAL_RCC_SPI5_CLK_DISABLE

__HAL_RCC_TIM1_CLK_ENABLE

__HAL_RCC_TIM1_CLK_DISABLE

__HAL_RCC_TIM8_CLK_ENABLE

__HAL_RCC_TIM8_CLK_DISABLE

__HAL_RCC_TIM9_CLK_ENABLE

__HAL_RCC_TIM9_CLK_DISABLE

__HAL_RCC_TIM15_CLK_ENABLE

__HAL_RCC_TIM15_CLK_DISABLE

__HAL_RCC_TIM16_CLK_ENABLE

__HAL_RCC_TIM16_CLK_DISABLE

__HAL_RCC_TIM17_CLK_ENABLE

__HAL_RCC_TIM17_CLK_DISABLE

__HAL_RCC_TIM18_CLK_ENABLE

__HAL_RCC_TIM18_CLK_DISABLE

__HAL_RCC_USART1_CLK_ENABLE

__HAL_RCC_USART1_CLK_DISABLE

__HAL_RCC_USART6_CLK_ENABLE

__HAL_RCC_USART6_CLK_DISABLE

__HAL_RCC_UART9_CLK_ENABLE

__HAL_RCC_UART9_CLK_DISABLE

__HAL_RCC_USART10_CLK_ENABLE

__HAL_RCC_USART10_CLK_DISABLE

APB3 Peripheral Clock Enabled Status

__HAL_RCC_DBGMCU_IS_CLK_ENABLED

APB3 Peripheral Clock Enable Disable

__HAL_RCC_DBGMCU_CLK_ENABLE

__HAL_RCC_DBGMCU_CLK_DISABLE

APB4 Peripheral Clock Enabled Status

__HAL_RCC_BSEC_IS_CLK_ENABLED

__HAL_RCC_DTS_IS_CLK_ENABLED

__HAL_RCC_HDP_IS_CLK_ENABLED

__HAL_RCC_I2C4_IS_CLK_ENABLED

__HAL_RCC_LPTIM2_IS_CLK_ENABLED

__HAL_RCC_LPTIM3_IS_CLK_ENABLED

__HAL_RCC_LPTIM4_IS_CLK_ENABLED

__HAL_RCC_LPTIM5_IS_CLK_ENABLED

__HAL_RCC_LPUART1_IS_CLK_ENABLED

__HAL_RCC_RTC_IS_CLK_ENABLED

__HAL_RCC_RTCAPB_IS_CLK_ENABLED

__HAL_RCC_SPI6_IS_CLK_ENABLED

__HAL_RCC_SYSCFG_IS_CLK_ENABLED

__HAL_RCC_VREFBUF_IS_CLK_ENABLED

APB4 Peripheral Clock Sleep Enable Disable

__HAL_RCC_BSEC_CLK_SLEEP_ENABLE

__HAL_RCC_BSEC_CLK_SLEEP_DISABLE

__HAL_RCC_DTS_CLK_SLEEP_ENABLE

__HAL_RCC_DTS_CLK_SLEEP_DISABLE

__HAL_RCC_HDP_CLK_SLEEP_ENABLE

__HAL_RCC_HDP_CLK_SLEEP_DISABLE

__HAL_RCC_I2C4_CLK_SLEEP_ENABLE

__HAL_RCC_I2C4_CLK_SLEEP_DISABLE

__HAL_RCC_LPTIM2_CLK_SLEEP_ENABLE

__HAL_RCC_LPTIM2_CLK_SLEEP_DISABLE

__HAL_RCC_LPTIM3_CLK_SLEEP_ENABLE

__HAL_RCC_LPTIM3_CLK_SLEEP_DISABLE

__HAL_RCC_LPTIM4_CLK_SLEEP_ENABLE

__HAL_RCC_LPTIM4_CLK_SLEEP_DISABLE

__HAL_RCC_LPTIM5_CLK_SLEEP_ENABLE

__HAL_RCC_LPTIM5_CLK_SLEEP_DISABLE

__HAL_RCC_LPUART1_CLK_SLEEP_ENABLE

__HAL_RCC_LPUART1_CLK_SLEEP_DISABLE

__HAL_RCC_RTC_CLK_SLEEP_ENABLE

__HAL_RCC_RTC_CLK_SLEEP_DISABLE

__HAL_RCC_RTCAPB_CLK_SLEEP_ENABLE

__HAL_RCC_RTCAPB_CLK_SLEEP_DISABLE

__HAL_RCC_SPI6_CLK_SLEEP_ENABLE

__HAL_RCC_SPI6_CLK_SLEEP_DISABLE

__HAL_RCC_SYSCFG_CLK_SLEEP_ENABLE

__HAL_RCC_SYSCFG_CLK_SLEEP_DISABLE

__HAL_RCC_VREFBUF_CLK_SLEEP_ENABLE

__HAL_RCC_VREFBUF_CLK_SLEEP_DISABLE

APB4 Peripheral Clock Sleep Enabled Status

__HAL_RCC_BSEC_IS_CLK_SLEEP_ENABLED

__HAL_RCC_DTS_IS_CLK_SLEEP_ENABLED

__HAL_RCC_HDP_IS_CLK_SLEEP_ENABLED

__HAL_RCC_I2C4_IS_CLK_SLEEP_ENABLED

__HAL_RCC_LPTIM2_IS_CLK_SLEEP_ENABLED

__HAL_RCC_LPTIM3_IS_CLK_SLEEP_ENABLED

__HAL_RCC_LPTIM4_IS_CLK_SLEEP_ENABLED

__HAL_RCC_LPTIM5_IS_CLK_SLEEP_ENABLED

__HAL_RCC_LPUART1_IS_CLK_SLEEP_ENABLED

__HAL_RCC_RTC_IS_CLK_SLEEP_ENABLED

__HAL_RCC_RTCAPB_IS_CLK_SLEEP_ENABLED

__HAL_RCC_SPI6_IS_CLK_SLEEP_ENABLED

__HAL_RCC_SYSCFG_IS_CLK_SLEEP_ENABLED

__HAL_RCC_VREFBUF_IS_CLK_SLEEP_ENABLED

APB4 Clock Source

RCC_APB4_DIV1

APB4 not divided

RCC_APB4_DIV2

APB4 divided by 2

RCC_APB4_DIV4

APB4 divided by 4

RCC_APB4_DIV8

APB4 divided by 8

RCC_APB4_DIV16

APB4 divided by 16

RCC_APB4_DIV32

APB4 divided by 32

RCC_APB4_DIV64

APB4 divided by 64

RCC_APB4_DIV128

APB4 divided by 128

APB4 Peripheral Force Release Reset

__HAL_RCC_APB4_FORCE_RESET

__HAL_RCC_APB4_RELEASE_RESET

__HAL_RCC_DTS_FORCE_RESET

__HAL_RCC_DTS_RELEASE_RESET

__HAL_RCC_HDP_FORCE_RESET

__HAL_RCC_HDP_RELEASE_RESET

__HAL_RCC_I2C4_FORCE_RESET

__HAL_RCC_I2C4_RELEASE_RESET

__HAL_RCC_LPTIM2_FORCE_RESET

__HAL_RCC_LPTIM2_RELEASE_RESET

__HAL_RCC_LPTIM3_FORCE_RESET

__HAL_RCC_LPTIM3_RELEASE_RESET

__HAL_RCC_LPTIM4_FORCE_RESET

__HAL_RCC_LPTIM4_RELEASE_RESET

__HAL_RCC_LPTIM5_FORCE_RESET

__HAL_RCC_LPTIM5_RELEASE_RESET

__HAL_RCC_LPUART1_FORCE_RESET

__HAL_RCC_LPUART1_RELEASE_RESET

__HAL_RCC_RTC_FORCE_RESET

__HAL_RCC_RTC_RELEASE_RESET

__HAL_RCC_SPI6_FORCE_RESET

__HAL_RCC_SPI6_RELEASE_RESET

__HAL_RCC_SYSCFG_FORCE_RESET

__HAL_RCC_SYSCFG_RELEASE_RESET

__HAL_RCC_VREFBUF_FORCE_RESET

__HAL_RCC_VREFBUF_RELEASE_RESET

APB4 Peripheral Clock Enable Disable

__HAL_RCC_BSEC_CLK_ENABLE

__HAL_RCC_BSEC_CLK_DISABLE

__HAL_RCC_DTS_CLK_ENABLE

__HAL_RCC_DTS_CLK_DISABLE

__HAL_RCC_HDP_CLK_ENABLE

__HAL_RCC_HDP_CLK_DISABLE

__HAL_RCC_I2C4_CLK_ENABLE

__HAL_RCC_I2C4_CLK_DISABLE

__HAL_RCC_LPTIM2_CLK_ENABLE

__HAL_RCC_LPTIM2_CLK_DISABLE

__HAL_RCC_LPTIM3_CLK_ENABLE

__HAL_RCC_LPTIM3_CLK_DISABLE

__HAL_RCC_LPTIM4_CLK_ENABLE

__HAL_RCC_LPTIM4_CLK_DISABLE

__HAL_RCC_LPTIM5_CLK_ENABLE

__HAL_RCC_LPTIM5_CLK_DISABLE

__HAL_RCC_LPUART1_CLK_ENABLE

__HAL_RCC_LPUART1_CLK_DISABLE

__HAL_RCC_RTC_CLK_ENABLE

__HAL_RCC_RTC_CLK_DISABLE

__HAL_RCC_RTCAPB_CLK_ENABLE

__HAL_RCC_RTCAPB_CLK_DISABLE

__HAL_RCC_SPI6_CLK_ENABLE

__HAL_RCC_SPI6_CLK_DISABLE

__HAL_RCC_SYSCFG_CLK_ENABLE

__HAL_RCC_SYSCFG_CLK_DISABLE

__HAL_RCC_VREFBUF_CLK_ENABLE

__HAL_RCC_VREFBUF_CLK_DISABLE

APB5 Peripheral Clock Enabled Status

__HAL_RCC_CSI_IS_CLK_ENABLED

__HAL_RCC_DCMIPP_IS_CLK_ENABLED

__HAL_RCC_GFXTIM_IS_CLK_ENABLED

__HAL_RCC_LTDC_IS_CLK_ENABLED

__HAL_RCC_VENC_IS_CLK_ENABLED

APB5 Peripheral Clock Sleep Enable Disable

__HAL_RCC_CSI_CLK_SLEEP_ENABLE

__HAL_RCC_CSI_CLK_SLEEP_DISABLE

__HAL_RCC_DCMIPP_CLK_SLEEP_ENABLE

__HAL_RCC_DCMIPP_CLK_SLEEP_DISABLE

__HAL_RCC_GFXTIM_CLK_SLEEP_ENABLE

__HAL_RCC_GFXTIM_CLK_SLEEP_DISABLE

__HAL_RCC_LTDC_CLK_SLEEP_ENABLE

__HAL_RCC_LTDC_CLK_SLEEP_DISABLE

__HAL_RCC_VENC_CLK_SLEEP_ENABLE

__HAL_RCC_VENC_CLK_SLEEP_DISABLE

APB5 Peripheral Clock Sleep Enabled Status

__HAL_RCC_CSI_IS_CLK_SLEEP_ENABLED

__HAL_RCC_DCMIPP_IS_CLK_SLEEP_ENABLED

__HAL_RCC_GFXTIM_IS_CLK_SLEEP_ENABLED

__HAL_RCC_LTDC_IS_CLK_SLEEP_ENABLED

__HAL_RCC_VENC_IS_CLK_SLEEP_ENABLED

APB5 Clock Source

RCC_APB5_DIV1
: APB5 not divided

RCC_APB5_DIV2
: APB5 divided by 2

RCC_APB5_DIV4
: APB5 divided by 4

RCC_APB5_DIV8
: APB5 divided by 8

RCC_APB5_DIV16
: APB5 divided by 16

RCC_APB5_DIV32
: APB5 divided by 32

RCC_APB5_DIV64
: APB5 divided by 64

RCC_APB5_DIV128

APB5 divided by 128

APB5 Peripheral Force Release Reset

__HAL_RCC_APB5_FORCE_RESET

__HAL_RCC_APB5_RELEASE_RESET

__HAL_RCC_CSI_FORCE_RESET

__HAL_RCC_CSI_RELEASE_RESET

__HAL_RCC_DCMIPP_FORCE_RESET

__HAL_RCC_DCMIPP_RELEASE_RESET

__HAL_RCC_GFXTIM_FORCE_RESET

__HAL_RCC_GFXTIM_RELEASE_RESET

__HAL_RCC_LTDC_FORCE_RESET

__HAL_RCC_LTDC_RELEASE_RESET

__HAL_RCC_VENC_FORCE_RESET

__HAL_RCC_VENC_RELEASE_RESET

APB5 Peripheral Clock Enable Disable

__HAL_RCC_CSI_CLK_ENABLE

__HAL_RCC_CSI_CLK_DISABLE

__HAL_RCC_DCMIPP_CLK_ENABLE

__HAL_RCC_DCMIPP_CLK_DISABLE

__HAL_RCC_GFXTIM_CLK_ENABLE

__HAL_RCC_GFXTIM_CLK_DISABLE

__HAL_RCC_LTDC_CLK_ENABLE

__HAL_RCC_LTDC_CLK_DISABLE

__HAL_RCC_VENC_CLK_ENABLE

__HAL_RCC_VENC_CLK_DISABLE

RCC attributes

RCC_ATTR_PRIV

attribute is privileged

RCC_ATTR_NPRIV

attribute is unprivileged

RCC_ATTR_SEC

attribute is secure

RCC_ATTR_NSEC

attribute is non-secure

RCC_ATTR_PUB

attribute is public

RCC_ATTR_NPUB

attribute is not public

RCC_ATTR_LOCK

attribute is locked

RCC_ATTR_NLOCK

attribute is not locked

Attribute Management

HAL_RCC_ATTRIBUTES_IS_SEC

Description:

- Check whether an item attribute is secured.

Parameters:

- __ATTRIBUTES__: specifies the item attributes This parameter is a combination of

Return value:

- 1: if the item attribute is secured, 0 otherwise.

HAL_RCC_ATTRIBUTES_IS_PRIV

Description:

- Check whether an item attribute is privileged.

Parameters:

- __ATTRIBUTES__: specifies the item attributes This parameter is a combination of

Return value:

- 1: if the item attribute is privileged, 0 otherwise.

HAL_RCC_ATTRIBUTES_IS_PUB

Description:

- Check whether an item attribute is public.

Parameters:

- __ATTRIBUTES__: specifies the item attributes This parameter is a combination of

Return value:

- 1: if the item attribute is public, 0 otherwise.

HAL_RCC_ATTRIBUTES_IS_LOCK

Description:

- Check whether an item attribute is locked.

Parameters:

- __ATTRIBUTES__: specifies the item attributes This parameter is a combination of

Return value:

- 1: if the item attribute is locked, 0 otherwise.

Clock Type

RCC_CLOCKTYPE_CPUCLK

CPU clock to configure

RCC_CLOCKTYPE_SYSCLK

System bus clock to configure for AXI

RCC_CLOCKTYPE_HCLK

HCLK to configure for AHB

RCC_CLOCKTYPE_PCLK1

PCLK1 to configure for APB1

RCC_CLOCKTYPE_PCLK2

PCLK2 to configure for APB2

RCC_CLOCKTYPE_PCLK4

PCLK4 to configure for APB4

RCC_CLOCKTYPE_PCLK5

PCLK5 to configure for APB5

CPU Clock Source

RCC_CPUCLKSOURCE_HSI

HSI selection as CPU clock

RCC_CPUCLKSOURCE_MSI

MSI selection as CPU clock

RCC_CPUCLKSOURCE_HSE

HSE selection as CPU clock

RCC_CPUCLKSOURCE_IC1

IC1 selection as CPU clock

CPU Clock Source Status

RCC_CPUCLKSOURCE_STATUS_HSI

HSI used as CPU clock

RCC_CPUCLKSOURCE_STATUS_MSI

MSI used as CPU clock

RCC_CPUCLKSOURCE_STATUS_HSE

HSE used as CPU clock

RCC_CPUCLKSOURCE_STATUS_IC1

IC1 used as CPU clock

Embedded Memory Clock Enable Disable

__HAL_RCC_AXISRAM1_MEM_CLK_ENABLE

__HAL_RCC_AXISRAM1_MEM_CLK_DISABLE

__HAL_RCC_AXISRAM2_MEM_CLK_ENABLE

__HAL_RCC_AXISRAM2_MEM_CLK_DISABLE

__HAL_RCC_AXISRAM3_MEM_CLK_ENABLE

__HAL_RCC_AXISRAM3_MEM_CLK_DISABLE

__HAL_RCC_AXISRAM4_MEM_CLK_ENABLE

__HAL_RCC_AXISRAM4_MEM_CLK_DISABLE

__HAL_RCC_AXISRAM5_MEM_CLK_ENABLE

__HAL_RCC_AXISRAM5_MEM_CLK_DISABLE

__HAL_RCC_AXISRAM6_MEM_CLK_ENABLE

__HAL_RCC_AXISRAM6_MEM_CLK_DISABLE

__HAL_RCC_AHBSRAM1_MEM_CLK_ENABLE

__HAL_RCC_AHBSRAM1_MEM_CLK_DISABLE

__HAL_RCC_AHBSRAM2_MEM_CLK_ENABLE

__HAL_RCC_AHBSRAM2_MEM_CLK_DISABLE

__HAL_RCC_BKPSRAM_MEM_CLK_ENABLE

__HAL_RCC_BKPSRAM_MEM_CLK_DISABLE

__HAL_RCC_FLEXRAM_MEM_CLK_ENABLE

__HAL_RCC_FLEXRAM_MEM_CLK_DISABLE

__HAL_RCC_CACHEAXIRAM_MEM_CLK_ENABLE

__HAL_RCC_CACHEAXIRAM_MEM_CLK_DISABLE

__HAL_RCC_VENCRAM_MEM_CLK_ENABLE

__HAL_RCC_VENCRAM_MEM_CLK_DISABLE

Embedded memory Clock Sleep Enable Disable

__HAL_RCC_AXISRAM1_MEM_CLK_SLEEP_ENABLE

__HAL_RCC_AXISRAM1_MEM_CLK_SLEEP_DISABLE

__HAL_RCC_AXISRAM2_MEM_CLK_SLEEP_ENABLE

__HAL_RCC_AXISRAM2_MEM_CLK_SLEEP_DISABLE

__HAL_RCC_AXISRAM3_MEM_CLK_SLEEP_ENABLE

__HAL_RCC_AXISRAM3_MEM_CLK_SLEEP_DISABLE

__HAL_RCC_AXISRAM4_MEM_CLK_SLEEP_ENABLE

__HAL_RCC_AXISRAM4_MEM_CLK_SLEEP_DISABLE

__HAL_RCC_AXISRAM5_MEM_CLK_SLEEP_ENABLE

__HAL_RCC_AXISRAM5_MEM_CLK_SLEEP_DISABLE

__HAL_RCC_AXISRAM6_MEM_CLK_SLEEP_ENABLE

__HAL_RCC_AXISRAM6_MEM_CLK_SLEEP_DISABLE

__HAL_RCC_AHBSRAM1_MEM_CLK_SLEEP_ENABLE

__HAL_RCC_AHBSRAM1_MEM_CLK_SLEEP_DISABLE

__HAL_RCC_AHBSRAM2_MEM_CLK_SLEEP_ENABLE

__HAL_RCC_AHBSRAM2_MEM_CLK_SLEEP_DISABLE

__HAL_RCC_BKPSRAM_MEM_CLK_SLEEP_ENABLE

__HAL_RCC_BKPSRAM_MEM_CLK_SLEEP_DISABLE

__HAL_RCC_FLEXRAM_MEM_CLK_SLEEP_ENABLE

__HAL_RCC_FLEXRAM_MEM_CLK_SLEEP_DISABLE

__HAL_RCC_CACHEAXIRAM_MEM_CLK_SLEEP_ENABLE

__HAL_RCC_CACHEAXIRAM_MEM_CLK_SLEEP_DISABLE

__HAL_RCC_VENCRAM_MEM_CLK_SLEEP_ENABLE

__HAL_RCC_VENCRAM_MEM_CLK_SLEEP_DISABLE

Embedded Memory Clock Sleep Enabled Status

__HAL_RCC_AXISRAM1_MEM_IS_CLK_SLEEP_ENABLED

__HAL_RCC_AXISRAM2_MEM_IS_CLK_SLEEP_ENABLED

__HAL_RCC_AXISRAM3_MEM_IS_CLK_SLEEP_ENABLED

__HAL_RCC_AXISRAM4_MEM_IS_CLK_SLEEP_ENABLED

__HAL_RCC_AXISRAM5_MEM_IS_CLK_SLEEP_ENABLED

__HAL_RCC_AXISRAM6_MEM_IS_CLK_SLEEP_ENABLED

__HAL_RCC_AHBSRAM1_MEM_IS_CLK_SLEEP_ENABLED

__HAL_RCC_AHBSRAM2_MEM_IS_CLK_SLEEP_ENABLED

__HAL_RCC_BKPSRAM_MEM_IS_CLK_SLEEP_ENABLED

__HAL_RCC_FLEXRAM_MEM_IS_CLK_SLEEP_ENABLED

__HAL_RCC_CACHEAXIRAM_MEM_IS_CLK_SLEEP_ENABLED

__HAL_RCC_VENCRAM_MEM_IS_CLK_SLEEP_ENABLED

Embedded Memory Clock Enabled Status

__HAL_RCC_AXISRAM1_MEM_IS_CLK_ENABLED

__HAL_RCC_AXISRAM2_MEM_IS_CLK_ENABLED

__HAL_RCC_AXISRAM3_MEM_IS_CLK_ENABLED

__HAL_RCC_AXISRAM4_MEM_IS_CLK_ENABLED

__HAL_RCC_AXISRAM5_MEM_IS_CLK_ENABLED

__HAL_RCC_AXISRAM6_MEM_IS_CLK_ENABLED

__HAL_RCC_AHBSRAM1_MEM_IS_CLK_ENABLED

__HAL_RCC_AHBSRAM2_MEM_IS_CLK_ENABLED

__HAL_RCC_BKPSRAM_MEM_IS_CLK_ENABLED

__HAL_RCC_FLEXRAM_MEM_IS_CLK_ENABLED

__HAL_RCC_CACHEAXIRAM_MEM_IS_CLK_ENABLED

__HAL_RCC_VENCRAM_MEM_IS_CLK_ENABLED

RCC Exported Macros

__HAL_RCC_HSI_CONFIG

Description:

- Macro to enable or disable the Internal High Speed oscillator (HSI).

Parameters:

- __STATE__: specifies the new state of the HSI. This parameter can be one of the following values:
  - RCC_HSI_OFF turn OFF the HSI oscillator
  - RCC_HSI_ON turn ON the HSI oscillator

Notes:

- After enabling the HSI, the application software should wait on HSIRDY flag to be set indicating that the HSI clock is stable and that HSI clock can be used to clock the PLL and/or system clock. HSI can not be stopped if it is used directly or through the PLL as system clock. In this case, you have to select another source of the system clock then stop the HSI. The HSI is stopped by hardware when entering STOP and STANDBY modes.
- When the HSI is stopped, HSIRDY flag goes low after 6 HSI oscillator clock cycles.

__HAL_RCC_HSI_DIVIDER_CONFIG

Description:

- Macro to configure the Internal High Speed oscillator (HSI) clock divider.

Parameters:

- __HSIDIV__: specifies the HSI division factor. This parameter can be one of the following values:
  - RCC_HSI_DIV1 Divide the HSI oscillator clock by 1 (default after reset)
  - RCC_HSI_DIV2 Divide the HSI oscillator clock by 2
  - RCC_HSI_DIV4 Divide the HSI oscillator clock by 4
  - RCC_HSI_DIV8 Divide the HSI oscillator clock by 8

__HAL_RCC_GET_HSI_DIVIDER

Description:

- Macro to get the HSI divider.

Return value:

- The: HSI divider. The returned value can be one of the following values: of the following:
  - RCC_HSI_DIV1 Divide the HSI oscillator clock by 1 (default after reset)
  - RCC_HSI_DIV2 Divide the HSI oscillator clock by 2
  - RCC_HSI_DIV4 Divide the HSI oscillator clock by 4
  - RCC_HSI_DIV8 Divide the HSI oscillator clock by 8

__HAL_RCC_HSI_ENABLE

Notes:

- The HSI is stopped by hardware when entering STOP and STANDBY modes. It is used (enabled by hardware) as system clock source after start-up from Reset, wakeup from STOP and STANDBY mode, or in case of failure of the HSE used directly or indirectly as system clock (if the Clock Security System CSS is enabled). HSI can not be stopped if it is used as system clock source. In this case, you have to select another source of the system clock then stop the HSI. After enabling the HSI, the application software should wait on HSIRDY flag to be set indicating that HSI clock is stable and can be used as system clock source. When the HSI is stopped, HSIRDY flag goes low after 6 HSI oscillator clock cycles.

__HAL_RCC_HSI_DISABLE

__HAL_RCC_HSI_CALIBRATIONVALUE_ADJUST

Description:

- Macro to adjust the Internal High Speed 64MHz oscillator (HSI) calibration value.

Parameters:

- __HSICALIBRATIONVALUE__: specifies the calibration trimming value (default is RCC_HSICALIBRATION_DEFAULT). This parameter must be a number between 0 and 127.

Return value:

- None

Notes:

- The calibration is used to compensate for the variations in voltage and temperature that influence the frequency of the internal HSI RC.

__HAL_RCC_HSISTOP_ENABLE

Description:

- Macros to enable or disable the force of the Internal High Speed oscillator (HSI) in STOP mode to be quickly available as kernel clock for some peripherals.

Return value:

- None

Notes:

- Keeping the HSI ON in STOP mode allows to avoid slowing down the communication speed because of the HSI start-up time. The enable of this function has not effect on the HSION bit. This parameter can be: ENABLE or DISABLE.

__HAL_RCC_HSISTOP_DISABLE

__HAL_RCC_MSI_ENABLE

Notes:

- The MSI is stopped by hardware when entering STOP and STANDBY modes. It is used (enabled by hardware) as system clock source after start-up from Reset, wakeup from STOP and STANDBY mode, or in case of failure of the HSE used directly or indirectly as system clock (if the Clock Security System CSS is enabled). MSI can not be stopped if it is used as system clock source. In this case, you have to select another source of the system clock then stop the MSI. After enabling the MSI, the application software should wait on MSIRDY flag to be set indicating that MSI clock is stable and can be used as system clock source. When the MSI is stopped, MSIRDY flag goes low after 6 MSI oscillator clock cycles.

__HAL_RCC_MSI_DISABLE

__HAL_RCC_MSI_FREQUENCY_CONFIG

Description:

- Macro to configure the Internal oscillator (MSI) frequency.

Parameters:

- __FREQ__: specifies the MSI frequency. This parameter can be one of the following values:
  - RCC_MSI_FREQ_4MHZ 4MHz selection (default after reset)
  - RCC_MSI_FREQ_16MHZ 16MHz selection

__HAL_RCC_MSI_CALIBRATIONVALUE_ADJUST

Description:

- Macro to adjust the Internal oscillator (MSI) calibration value.

Parameters:

- __MSICALIBRATIONVALUE__: specifies the calibration trimming value (default is RCC_MSICALIBRATION_DEFAULT). This parameter must be a number between 0 and 0x1F.

Return value:

- None

Notes:

- The calibration is used to compensate for the variations in voltage and temperature that influence the frequency of the internal MSI RC.

__HAL_RCC_MSISTOP_ENABLE

Description:

- Macros to enable or disable the force of the Low-power Internal oscillator (MSI) in STOP mode to be quickly available as kernel clock for USARTs and I2Cs.

Return value:

- None

Notes:

- Keeping the MSI ON in STOP mode allows to avoid slowing down the communication speed because of the MSI start-up time. The enable of this function has not effect on the MSION bit. This parameter can be: ENABLE or DISABLE.

__HAL_RCC_MSISTOP_DISABLE

__HAL_RCC_LSI_ENABLE

Notes:

- After enabling the LSI, the application software should wait on LSIRDY flag to be set indicating that LSI clock is stable and can be used to clock the IWDG and/or the RTC. LSI can not be disabled if the IWDG is running. When the LSI is stopped, LSIRDY flag goes low after 6 LSI oscillator clock cycles.

__HAL_RCC_LSI_DISABLE

__HAL_RCC_HSE_CONFIG

Description:

- Macros to configure the External High Speed oscillator (HSE).

Parameters:

- __STATE__: specifies the new state of the HSE. This parameter can be one of the following values:
  - RCC_HSE_OFF turn OFF the HSE oscillator, HSERDY flag goes low after 6 HSE oscillator clock cycles.
  - RCC_HSE_ON turn ON the HSE oscillator.
  - RCC_HSE_BYPASS HSE oscillator bypassed with external clock (analog).
  - RCC_HSE_BYPASS_DIGITAL HSE oscillator bypassed with external digital clock.

Notes:

- After enabling the HSE (RCC_HSE_ON or RCC_HSE_BYPASS or RCC_HSE_BYPASS_DIGITAL), the application software should wait on HSERDY flag to be set indicating that HSE clock is stable and can be used to clock the PLL and/or system clock and peripheral kernel clock source. HSE state can not be changed if it is used directly or through the PLL as system clock. In this case, you have to select another source of the system clock then change the HSE state (ex. disable it). The HSE is stopped by hardware when entering STOP and STANDBY modes.

__HAL_RCC_LSE_CONFIG

Description:

- Macros to configure the External Low Speed oscillator (LSE).

Parameters:

- __STATE__: specifies the new state of the LSE. This parameter can be one of the following values:
  - RCC_LSE_OFF turn OFF the LSE oscillator, LSERDY flag goes low after 6 LSE oscillator clock cycles.
  - RCC_LSE_ON turn ON the LSE oscillator.
  - RCC_LSE_BYPASS LSE oscillator bypassed with external clock (analog).
  - RCC_LSE_BYPASS_DIGITAL LSE oscillator bypassed with external digital clock.

Notes:

- Transition LSE Bypass to LSE On and LSE On to LSE Bypass are not supported by this macro. User should request a transition to LSE Off first and then LSE On or LSE Bypass. The external input clock can have a frequency up to 1 MHz and be low swing (analog) or digital(*). A duty cycle close to 50% is recommended. As the LSE is in the Backup domain and write access is denied to this domain after reset, you have to enable write access using HAL_PWR_EnableBkUpAccess() function before to configure the LSE (to be done once after reset). After enabling the LSE (RCC_LSE_ON, RCC_LSE_BYPASS or RCC_LSE_BYPASS_DIGITAL), the application software should wait on LSERDY flag to be set indicating that LSE clock is stable and can be used to clock the RTC. If the RTC is used, the LSE bypass must not be configured in digital mode but in low swing analog mode (*)

__HAL_RCC_RTC_ENABLE

Notes:

- These macros must be used only after the RTC clock source was selected.

__HAL_RCC_RTC_DISABLE

__HAL_RCC_RTC_CONFIG

Description:

- Macro to configure the RTC clock (RTCCLK).

Parameters:

- __RTC_CLKSOURCE__: specifies the RTC clock source. This parameter can be one of the following values: RCC_RTCCLKSOURCE_LSE LSE selected as RTC clock. RCC_RTCCLKSOURCE_LSI LSI selected as RTC clock. RCC_RTCCLKSOURCE_HSE_DIVx HSE clock divided by x selected as RTC clock, where x:[1,64]

Notes:

- As the RTC clock configuration bits are in the Backup domain and write access is denied to this domain after reset, you have to enable write access using the Power Backup Access macro before to configure the RTC clock source (to be done once after reset). Once the RTC clock is configured it cannot be changed unless the Backup domain is reset using __HAL_RCC_BACKUPRESET_FORCE() and __HAL_RCC_BACKUPRESET_RELEASE() macros, or by a Power On Reset (POR).
- If the LSE or LSI is used as RTC clock source, the RTC continues to work in STOP and STANDBY modes, and can be used as wakeup source. However, when the HSE clock is used as RTC clock source, the RTC cannot be used in STOP and STANDBY modes. The maximum input clock frequency for RTC is 1MHz (when using HSE as RTC clock source).

__HAL_RCC_RTC_CLKPRESCALER

Description:

- Macro to configure the RTC clock prescaler for HSE for RTCCLK.

Parameters:

- __RTC_CLKSOURCE__: specifies the RTC clock prescaler for HSE. This parameter can be one of the following values: RCC_RTCCLKSOURCE_HSE_DIVx HSE clock divided by x selected as RTC clock, where x:[1,64]

__HAL_RCC_GET_RTC_SOURCE

Description:

- Macros to get the RTC clock (RTCCLK).

Return value:

- The: returned value can be one of the following values: RCC_RTCCLKSOURCE_DISABLE No clock selected as RTC clock. RCC_RTCCLKSOURCE_LSE LSE selected as RTC clock. RCC_RTCCLKSOURCE_LSI LSI selected as RTC clock. RCC_RTCCLKSOURCE_HSE_DIVx HSE clock divided by x selected as RTC clock, where x:[1,64]

__HAL_RCC_BACKUPRESET_FORCE

Notes:

- This function resets the RTC peripheral (including the backup registers) and the RTC clock source selection in RCC_BDCR register. The BKPSRAM is not affected by this reset.

__HAL_RCC_BACKUPRESET_RELEASE

__HAL_RCC_PLL1_ENABLE

Notes:

- After enabling the PLL1, the application software should wait on PLL1RDY flag to be set indicating that PLL1 clock is stable and can be used as CPU and/or system bus and/or kernel clock source thru ICx. The PLL1 can not be disabled if it is used as system clock source The PLL1 is disabled by hardware when entering STOP and STANDBY modes.

__HAL_RCC_PLL1_DISABLE

__HAL_RCC_PLL1CLKOUT_ENABLE

__HAL_RCC_PLL1CLKOUT_DISABLE

__HAL_RCC_PLL1_CONFIG

Description:

- Macro to configure the PLL1 in integer mode with clock source, multiplication and division factors.

Parameters:

- __PLLSOURCE__: specifies the PLL entry clock source. This parameter can be one of the following values:
  - RCC_PLLSOURCE_HSI HSI oscillator clock selected as PLL clock entry
  - RCC_PLLSOURCE_MSI MSI oscillator clock selected as PLL clock entry
  - RCC_PLLSOURCE_HSE HSE oscillator clock selected as PLL clock entry
  - RCC_PLLSOURCE_PIN I2S_CKIN clock selected as PLL clock entry
- __PLLM__: specifies the division factor for PLL VCO input clock. This parameter must be a number between 1 and 63.
- __PLLN__: specifies the multiplication factor for PLL VCO output clock. This parameter must be a number between 16 and 2500.
- __PLLP1__: specifies the post division factor 1 for system clock. This parameter must be a number between 1 and 7.
- __PLLP2__: specifies the post division factor 2 for system clock. This parameter must be a number between 1 and 7.

Return value:

- None

Notes:

- This macro must be used only when the PLL1 is disabled or the PLL1 output is bypassed and driven by the PLL reference clock. This macro switches off the bypass mode.
- This clock source cannot be changed on any PLL if another PLL is already enabled.
- You have to set the PLLM parameter correctly to ensure that the VCO input frequency ranges from 1 to 50 MHz.

__HAL_RCC_PLL1_PLLSOURCE_CONFIG

Description:

- Macro to configure the PLL1 clock source.

Parameters:

- __PLLSOURCE__: specifies the PLL entry clock source. This parameter can be one of the following values:
  - RCC_PLLSOURCE_HSI HSI oscillator clock selected as PLL clock entry
  - RCC_PLLSOURCE_MSI MSI oscillator clock selected as PLL clock entry
  - RCC_PLLSOURCE_HSE HSE oscillator clock selected as PLL clock entry
  - RCC_PLLSOURCE_PIN External I2S_CKIN clock selected as PLL clock entry

Notes:

- This macro must be used only when the PLL1 is disabled or the PLL1 output is bypassed and driven by the PLL reference clock. This macro switches off the bypass mode.

__HAL_RCC_GET_PLL1_OSCSOURCE

Description:

- Macro to get the clock source used as PLL1 clock source.

Return value:

- The: oscillator used as PLL clock source. The returned value can be one of the following values:
  - RCC_PLLSOURCE_HSI HSI oscillator is used as PLL clock source.
  - RCC_PLLSOURCE_MSI MSI oscillator is used as PLL clock source.
  - RCC_PLLSOURCE_HSE HSE oscillator is used as PLL clock source.
  - RCC_PLLSOURCE_PIN External I2S_CKIN is used as PLL clock source.

__HAL_RCC_PLL1_DIVNFRAC_CONFIG

Description:

- Macro to configure the PLL1 clock Fractional Part Of The Multiplication Factor.

Parameters:

- __PLLDIVNFRAC__: specifies Fractional Part Of The Multiplication Factor for PLL1 VCO. It should be a value between 0 and 2^24.

Return value:

- None

Notes:

- This configuration cannot be requested when the PLL1 has been enabled.

__HAL_RCC_PLL1_FRACN_ENABLE

Description:

- Macro to enable the PLL1 clock Fractional mode.

Return value:

- None

Notes:

- This configuration cannot be requested when the PLL1 has been enabled.

__HAL_RCC_PLL1_FRACN_DISABLE

Description:

- Macro to disable the PLL1 clock Fractional mode.

Return value:

- None

Notes:

- This configuration cannot be requested when the PLL1 has been enabled.

__HAL_RCC_PLL2_ENABLE

Notes:

- After enabling PLL2, the application software should wait on PLL2RDY flag to be set indicating that PLL2 clock is stable and can be used as CPU and/or system bus and/or kernel clock source thru ICx. PLL2 is disabled by hardware when entering Stop and Standby modes.

__HAL_RCC_PLL2_DISABLE

__HAL_RCC_PLL2CLKOUT_ENABLE

__HAL_RCC_PLL2CLKOUT_DISABLE

__HAL_RCC_PLL2_CONFIG

Description:

- Macro to configure the PLL2 in integer mode with clock source, multiplication and division factors.

Parameters:

- __PLLSOURCE__: specifies the PLL entry clock source. This parameter can be one of the following values:
  - RCC_PLLSOURCE_HSI HSI oscillator clock selected as PLL clock entry
  - RCC_PLLSOURCE_MSI MSI oscillator clock selected as PLL clock entry
  - RCC_PLLSOURCE_HSE HSE oscillator clock selected as PLL clock entry
  - RCC_PLLSOURCE_PIN I2S_CKIN clock selected as PLL clock entry
- __PLLM__: specifies the division factor for PLL VCO input clock This parameter must be a number between 1 and 63.
- __PLLN__: specifies the multiplication factor for PLL VCO output clock This parameter must be a number between 16 and 2500.
- __PLLP1__: specifies the post division factor 1 for system clock. This parameter must be a number between 1 and 7.
- __PLLP2__: specifies the post division factor 2 for system clock. This parameter must be a number between 1 and 7.

Return value:

- None

Notes:

- This macro must be used only when the PLL2 is disabled or the PLL2 output is bypassed and driven by the PLL reference clock. This macro switches off the bypass mode.
- This clock source cannot be changed on any PLL if another PLL is already enabled.
- You have to set the PLLM parameter correctly to ensure that the VCO input frequency ranges from 1 to 50 MHz.

__HAL_RCC_PLL2_PLLSOURCE_CONFIG

Description:

- Macro to configure the PLL2 clock source.

Parameters:

- __PLLSOURCE__: specifies the PLL entry clock source. This parameter can be one of the following values:
  - RCC_PLLSOURCE_HSI HSI oscillator clock selected as PLL clock entry
  - RCC_PLLSOURCE_MSI MSI oscillator clock selected as PLL clock entry
  - RCC_PLLSOURCE_HSE HSE oscillator clock selected as PLL clock entry
  - RCC_PLLSOURCE_PIN External I2S_CKIN clock selected as PLL clock entry

Notes:

- This macro must be used only when the PLL2 is disabled or the PLL2 output is bypassed and driven by the PLL reference clock. This macro switches off the bypass mode.

__HAL_RCC_GET_PLL2_OSCSOURCE

Description:

- Macro to get the clock source used as PLL2 clock source.

Return value:

- The: oscillator used as PLL clock source. The returned value can be one of the following values:
  - RCC_PLLSOURCE_HSI HSI oscillator is used as PLL clock source.
  - RCC_PLLSOURCE_MSI MSI oscillator is used as PLL clock source.
  - RCC_PLLSOURCE_HSE HSE oscillator is used as PLL clock source.
  - RCC_PLLSOURCE_PIN External I2S_CKIN is used as PLL clock source.

__HAL_RCC_PLL2_DIVNFRAC_CONFIG

Description:

- Macro to configure the PLL2 clock Fractional Part Of The Multiplication Factor.

Parameters:

- __PLLDIVNFRAC__: specifies Fractional Part Of The Multiplication Factor for PLL2 VCO. It should be a value between 0 and 2^24.

Return value:

- None

Notes:

- This configuration cannot be requested when the PLL2 has been enabled.

__HAL_RCC_PLL2_FRACN_ENABLE

Description:

- Macro to enable the PLL2 clock Fractional mode.

Return value:

- None

Notes:

- This configuration cannot be requested when the PLL2 has been enabled.

__HAL_RCC_PLL2_FRACN_DISABLE

Description:

- Macro to disable the PLL2 clock Fractional mode.

Return value:

- None

Notes:

- This configuration cannot be requested when the PLL2 has been enabled.

__HAL_RCC_PLL3_ENABLE

Notes:

- After enabling PLL3, the application software should wait on PLL3RDY flag to be set indicating that PLL3 clock is stable and can be used as CPU and/or system bus and/or kernel clock source thru ICx. PLL3 is disabled by hardware when entering Stop and Standby modes.

__HAL_RCC_PLL3_DISABLE

__HAL_RCC_PLL3CLKOUT_ENABLE

__HAL_RCC_PLL3CLKOUT_DISABLE

__HAL_RCC_PLL3_CONFIG

Description:

- Macro to configure the PLL3 in integer mode with clock source, multiplication and division factors.

Parameters:

- __PLLSOURCE__: specifies the PLL entry clock source. This parameter can be one of the following values:
  - RCC_PLLSOURCE_HSI HSI oscillator clock selected as PLL clock entry
  - RCC_PLLSOURCE_MSI MSI oscillator clock selected as PLL clock entry
  - RCC_PLLSOURCE_HSE HSE oscillator clock selected as PLL clock entry
  - RCC_PLLSOURCE_PIN I2S_CKIN clock selected as PLL clock entry
- __PLLM__: specifies the division factor for PLL VCO input clock This parameter must be a number between 1 and 63.
- __PLLN__: specifies the multiplication factor for PLL VCO output clock This parameter must be a number between 16 and 2500.
- __PLLP1__: specifies the post division factor 1 for system clock. This parameter must be a number between 1 and 7.
- __PLLP2__: specifies the post division factor 2 for system clock. This parameter must be a number between 1 and 7.

Return value:

- None

Notes:

- This macro must be used only when the PLL3 is disabled or the PLL3 output is bypassed and driven by the PLL reference clock. This macro switches off the bypass mode.
- This clock source cannot be changed on any PLL if another PLL is already enabled.
- You have to set the PLLM parameter correctly to ensure that the VCO input frequency ranges from 1 to 50 MHz.

__HAL_RCC_PLL3_PLLSOURCE_CONFIG

Description:

- Macro to configure the PLL3 clock source.

Parameters:

- __PLLSOURCE__: specifies the PLL entry clock source. This parameter can be one of the following values:
  - RCC_PLLSOURCE_HSI HSI oscillator clock selected as PLL clock entry
  - RCC_PLLSOURCE_MSI MSI oscillator clock selected as PLL clock entry
  - RCC_PLLSOURCE_HSE HSE oscillator clock selected as PLL clock entry
  - RCC_PLLSOURCE_PIN External I2S_CKIN clock selected as PLL clock entry

Notes:

- This macro must be used only when the PLL3 is disabled or the PLL3 output is bypassed and driven by the PLL reference clock. This macro switches off the bypass mode.

__HAL_RCC_GET_PLL3_OSCSOURCE

Description:

- Macro to get the clock source used as PLL3 clock source.

Return value:

- The: oscillator used as PLL clock source. The returned value can be one of the following values:
  - RCC_PLLSOURCE_HSI HSI oscillator is used as PLL clock source.
  - RCC_PLLSOURCE_MSI MSI oscillator is used as PLL clock source.
  - RCC_PLLSOURCE_HSE HSE oscillator is used as PLL clock source.
  - RCC_PLLSOURCE_PIN External I2S_CKIN is used as PLL clock source.

__HAL_RCC_PLL3_DIVNFRAC_CONFIG

Description:

- Macro to configure the PLL3 clock Fractional Part Of The Multiplication Factor.

Parameters:

- __PLLDIVNFRAC__: specifies Fractional Part Of The Multiplication Factor for PLL3 VCO. It should be a value between 0 and 2^24.

Return value:

- None

Notes:

- This configuration cannot be requested when the PLL3 has been enabled.

__HAL_RCC_PLL3_FRACN_ENABLE

Description:

- Macro to enable the PLL3 clock Fractional mode.

Return value:

- None

Notes:

- This configuration cannot be requested when the PLL3 has been enabled.

__HAL_RCC_PLL3_FRACN_DISABLE

Description:

- Macro to disable the PLL3 clock Fractional mode.

Return value:

- None

Notes:

- This configuration cannot be requested when the PLL3 has been enabled.

__HAL_RCC_PLL4_ENABLE

Notes:

- After enabling PLL4, the application software should wait on PLL4RDY flag to be set indicating that PLL4 clock is stable and can be used as CPU and/or system bus and/or kernel clock source thru ICx. PLL4 is disabled by hardware when entering Stop and Standby modes.

__HAL_RCC_PLL4_DISABLE

__HAL_RCC_PLL4CLKOUT_ENABLE

__HAL_RCC_PLL4CLKOUT_DISABLE

__HAL_RCC_PLL4_CONFIG

Description:

- Macro to configure the PLL4 in integer mode with clock source, multiplication and division factors.

Parameters:

- __PLLSOURCE__: specifies the PLL entry clock source. This parameter can be one of the following values:
  - RCC_PLLSOURCE_HSI HSI oscillator clock selected as PLL clock entry
  - RCC_PLLSOURCE_MSI MSI oscillator clock selected as PLL clock entry
  - RCC_PLLSOURCE_HSE HSE oscillator clock selected as PLL clock entry
  - RCC_PLLSOURCE_PIN I2S_CKIN clock selected as PLL clock entry
- __PLLM__: specifies the division factor for PLL VCO input clock This parameter must be a number between 1 and 63.
- __PLLN__: specifies the multiplication factor for PLL VCO output clock This parameter must be a number between 16 and 2500.
- __PLLP1__: specifies the post division factor 1 for system clock. This parameter must be a number between 1 and 7.
- __PLLP2__: specifies the post division factor 2 for system clock. This parameter must be a number between 1 and 7.

Return value:

- None

Notes:

- This macro must be used only when the PLL4 is disabled or the PLL4 output is bypassed and driven by the PLL reference clock. This macro switches off the bypass mode.
- This clock source cannot be changed on any PLL if another PLL is already enabled.
- You have to set the PLLM parameter correctly to ensure that the VCO input frequency ranges from 1 to 50 MHz.

__HAL_RCC_PLL4_PLLSOURCE_CONFIG

Description:

- Macro to configure the PLL4 clock source.

Parameters:

- __PLLSOURCE__: specifies the PLL entry clock source. This parameter can be one of the following values:
  - RCC_PLLSOURCE_HSI HSI oscillator clock selected as PLL clock entry
  - RCC_PLLSOURCE_MSI MSI oscillator clock selected as PLL clock entry
  - RCC_PLLSOURCE_HSE HSE oscillator clock selected as PLL clock entry
  - RCC_PLLSOURCE_PIN External I2S_CKIN clock selected as PLL clock entry

Notes:

- This macro must be used only when the PLL4 is disabled or the PLL4 output is bypassed and driven by the PLL reference clock. This macro switches off the bypass mode.

__HAL_RCC_GET_PLL4_OSCSOURCE

Description:

- Macro to get the clock source used as PLL4 clock source.

Return value:

- The: oscillator used as PLL clock source. The returned value can be one of the following values:
  - RCC_PLLSOURCE_HSI HSI oscillator is used as PLL clock source.
  - RCC_PLLSOURCE_MSI MSI oscillator is used as PLL clock source.
  - RCC_PLLSOURCE_HSE HSE oscillator is used as PLL clock source.
  - RCC_PLLSOURCE_PIN External I2S_CKIN is used as PLL clock source.

__HAL_RCC_PLL4_DIVNFRAC_CONFIG

Description:

- Macro to configure the PLL4 clock Fractional Part Of The Multiplication Factor.

Parameters:

- __PLLDIVNFRAC__: specifies Fractional Part Of The Multiplication Factor for PLL4 VCO. It should be a value between 0 and 2^24.

Return value:

- None

Notes:

- This configuration cannot be requested when the PLL4 has been enabled.

__HAL_RCC_PLL4_FRACN_ENABLE

Description:

- Macro to enable the PLL4 clock Fractional mode.

Return value:

- None

Notes:

- This configuration cannot be requested when the PLL4 has been enabled.

__HAL_RCC_PLL4_FRACN_DISABLE

Description:

- Macro to disable the PLL4 clock Fractional mode.

Return value:

- None

Notes:

- This configuration cannot be requested when the PLL4 has been enabled.

__HAL_RCC_CPUCLK_CONFIG

Description:

- Macro to configure the CPU clock source.

Parameters:

- __CPUCLKSOURCE__: specifies the CPU clock source. This parameter can be one of the following values:
  - RCC_CPUCLKSOURCE_HSI HSI oscillator is used as CPU clock source.
  - RCC_CPUCLKSOURCE_MSI MSI oscillator is used as CPU clock source.
  - RCC_CPUCLKSOURCE_HSE HSE oscillator is used as CPU clock source.
  - RCC_CPUCLKSOURCE_IC1 IC1 output is used as CPU clock source.

Return value:

- None

__HAL_RCC_GET_CPUCLK_SOURCE

Description:

- Macro to get the clock source used as CPU clock.

Return value:

- The: clock source used as CPU clock. The returned value can be one of the following values:
  - RCC_CPUCLKSOURCE_STATUS_HSI HSI used as CPU clock.
  - RCC_CPUCLKSOURCE_STATUS_MSI MSI used as CPU clock.
  - RCC_CPUCLKSOURCE_STATUS_HSE HSE used as CPU clock.
  - RCC_CPUCLKSOURCE_STATUS_IC1 IC1 used as CPU clock.

__HAL_RCC_SYSCLK_CONFIG

Description:

- Macro to configure the system bus clock source.

Parameters:

- __SYSCLKSOURCE__: specifies the system clock source. This parameter can be one of the following values:
  - RCC_SYSCLKSOURCE_HSI HSI oscillator is used as system bus clock source.
  - RCC_SYSCLKSOURCE_MSI MSI oscillator is used as system bus clock source.
  - RCC_SYSCLKSOURCE_HSE HSE oscillator is used as system bus clock source.
  - RCC_SYSCLKSOURCE_IC2_IC6_IC11 IC2, IC6 and IC11 outputs are used as system bus clock source.

Return value:

- None

__HAL_RCC_GET_SYSCLK_SOURCE

Description:

- Macro to get the clock source used as system bus clock.

Return value:

- The: clock source used as system clock. The returned value can be one of the following values:
  - RCC_SYSCLKSOURCE_STATUS_HSI HSI used as system bus clock.
  - RCC_SYSCLKSOURCE_STATUS_MSI MSI used as system bus clock.
  - RCC_SYSCLKSOURCE_STATUS_HSE HSE used as system bus clock.
  - RCC_SYSCLKSOURCE_STATUS_IC2_IC6_IC11 IC2, IC6 and IC11 outputs used as system bus clock.

__HAL_RCC_LSEDRIVE_CONFIG

Description:

- Macro to configure the External Low Speed oscillator (LSE) drive capability.

Parameters:

- __LSEDRIVE__: specifies the new state of the LSE drive capability. This parameter can be one of the following values:
  - RCC_LSEDRIVE_LOW LSE oscillator low drive capability.
  - RCC_LSEDRIVE_MEDIUMLOW LSE oscillator medium low drive capability.
  - RCC_LSEDRIVE_MEDIUMHIGH LSE oscillator medium high drive capability.
  - RCC_LSEDRIVE_HIGH LSE oscillator high drive capability.

Return value:

- None

Notes:

- As the LSE is in the Backup domain and write access is denied to this domain after reset, you have to enable write access using HAL_PWR_EnableBkUpAccess() function before to configure the LSE (to be done once after reset). This parameter cannot be updated while LSE is ON.

__HAL_RCC_WAKEUPSTOP_CLK_CONFIG

Description:

- Macro to configure the wake up from stop clock.

Parameters:

- __STOPWUCLK__: specifies the clock source used after wake up from stop This parameter can be one of the following values:
  - RCC_STOP_WAKEUPCLOCK_HSI HSI selected as system clock source
  - RCC_STOP_WAKEUPCLOCK_MSI MSI selected as system clock source

Return value:

- None

Flag

RCC_FLAG_LSIRDY

LSI ready flag

RCC_FLAG_LSERDY

LSE ready flag

RCC_FLAG_HSIRDY

HSI ready flag

RCC_FLAG_MSIRDY

MSI ready flag

RCC_FLAG_HSERDY

HSE ready flag

RCC_FLAG_PLL1RDY

PLL1 ready flag

RCC_FLAG_PLL2RDY

PLL2 ready flag

RCC_FLAG_PLL3RDY

PLL3 ready flag

RCC_FLAG_PLL4RDY

PLL4 ready flag

RCC_FLAG_LSECSSD

LSE Clock Security System failure detection flag

RCC_FLAG_HSECSSD

HSE Clock Security System failure detection flag

RCC_FLAG_LCKRST

CPU Lockup reset flag

RCC_FLAG_BORRST

BOR reset flag

RCC_FLAG_PINRST

PIN reset flag

RCC_FLAG_PORRST

Power-on reset flag

RCC_FLAG_SFTRST

Software Reset flag

RCC_FLAG_IWDGRST

Independent Watchdog reset flag

RCC_FLAG_WWDGRST

Window watchdog reset flag

RCC_FLAG_LPWRRST

Low-Power reset flag

Flags Interrupts Management

__HAL_RCC_ENABLE_IT

Description:

- Enable RCC interrupt.

Parameters:

- __INTERRUPT__: specifies the RCC interrupt source(s) to be enabled. This parameter can be any combination of the following values:
  - RCC_IT_LSIRDY LSI ready interrupt
  - RCC_IT_LSERDY LSE ready interrupt
  - RCC_IT_HSIRDY HSI ready interrupt
  - RCC_IT_MSIRDY MSI ready interrupt
  - RCC_IT_HSERDY HSE ready interrupt
  - RCC_IT_PLL1RDY PLL1 ready interrupt
  - RCC_IT_PLL2RDY PLL2 ready interrupt
  - RCC_IT_PLL3RDY PLL3 ready interrupt
  - RCC_IT_PLL4RDY PLL4 ready interrupt
  - RCC_IT_LSECSS LSE clock security system interrupt
  - RCC_IT_HSECSS HSE clock security system interrupt
  - RCC_IT_WKUP CPU wakeup interrupt

__HAL_RCC_DISABLE_IT

Description:

- Disable RCC interrupt.

Parameters:

- __INTERRUPT__: specifies the RCC interrupt source(s) to be disabled. This parameter can be any combination of the following values:
  - RCC_IT_LSIRDY LSI ready interrupt
  - RCC_IT_LSERDY LSE ready interrupt
  - RCC_IT_HSIRDY HSI ready interrupt
  - RCC_IT_MSIRDY MSI ready interrupt
  - RCC_IT_HSERDY HSE ready interrupt
  - RCC_IT_PLL1RDY PLL1 ready interrupt
  - RCC_IT_PLL2RDY PLL2 ready interrupt
  - RCC_IT_PLL3RDY PLL3 ready interrupt
  - RCC_IT_PLL4RDY PLL4 ready interrupt
  - RCC_IT_LSECSS LSE clock security system interrupt
  - RCC_IT_HSECSS HSE clock security system interrupt
  - RCC_IT_WKUP CPU wakeup interrupt

__HAL_RCC_CLEAR_IT

Description:

- Clear the RCC's interrupt pending bits.

Parameters:

- __INTERRUPT__: specifies the interrupt pending bit to clear. This parameter can be any combination of the following values:
  - RCC_IT_LSIRDY LSI ready interrupt
  - RCC_IT_LSERDY LSE ready interrupt
  - RCC_IT_HSIRDY HSI ready interrupt
  - RCC_IT_MSIRDY MSI ready interrupt
  - RCC_IT_HSERDY HSE ready interrupt
  - RCC_IT_PLL1RDY PLL1 ready interrupt
  - RCC_IT_PLL2RDY PLL2 ready interrupt
  - RCC_IT_PLL3RDY PLL3 ready interrupt
  - RCC_IT_PLL4RDY PLL4 ready interrupt
  - RCC_IT_LSECSS LSE clock security system interrupt
  - RCC_IT_HSECSS HSE clock security interrupt
  - RCC_IT_WKUP CPU wakeup interrupt

__HAL_RCC_GET_IT

Description:

- Check whether the RCC interrupt has occurred or not.

Parameters:

- __INTERRUPT__: specifies the RCC interrupt source to check. This parameter can be one of the following values:
  - RCC_IT_LSIRDY LSI ready interrupt
  - RCC_IT_LSERDY LSE ready interrupt
  - RCC_IT_HSIRDY HSI ready interrupt
  - RCC_IT_MSIRDY MSI ready interrupt
  - RCC_IT_HSERDY HSE ready interrupt
  - RCC_IT_PLL1RDY PLL1 ready interrupt
  - RCC_IT_PLL2RDY PLL2 ready interrupt
  - RCC_IT_PLL3RDY PLL3 ready interrupt
  - RCC_IT_PLL4RDY PLL4 ready interrupt
  - RCC_IT_LSECSS LSE clock security system interrupt
  - RCC_IT_HSECSS HSE clock security interrupt
  - RCC_IT_WKUP CPU wakeup interrupt

Return value:

- The: pending state of __INTERRUPT__ (TRUE or FALSE).

__HAL_RCC_CLEAR_RESET_FLAGS

The reset flags are: RCC_FLAG_LCKRST, RCC_FLAG_BORRST, RCC_FLAG_PINRST, RCC_FLAG_PORRST, RCC_FLAG_SFTRST, RCC_FLAG_IWDGRST, RCC_FLAG_WWDGRST and RCC_FLAG_LPWRRST.

__HAL_RCC_GET_FLAG

Description:

- Check whether the selected RCC flag is set or not.

Parameters:

- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - RCC_FLAG_LCKRST CPU lockup reset flag
  - RCC_FLAG_BORRST BOR reset flag
  - RCC_FLAG_PINRST Pin reset flag
  - RCC_FLAG_PORRST Power-on reset flag
  - RCC_FLAG_SFTRST Software reset flag
  - RCC_FLAG_IWDGRST Independent Watchdog reset flag
  - RCC_FLAG_WWDGRST Window Watchdog reset flag
  - RCC_FLAG_LPWRRST Low Power reset flag
  - RCC_FLAG_LSECSSD Clock security system failure on LSE oscillator detection flag
  - RCC_FLAG_HSECSSD Clock security system failure on HSE oscillator detection flag
  - RCC_FLAG_LSIRDY LSI oscillator clock ready flag
  - RCC_FLAG_LSERDY LSE oscillator clock ready flag
  - RCC_FLAG_HSIRDY HSI oscillator clock ready flag
  - RCC_FLAG_MSIRDY MSI oscillator clock ready flag
  - RCC_FLAG_HSERDY HSE oscillator clock ready flag
  - RCC_FLAG_PLL1RDY PLL1 ready flag
  - RCC_FLAG_PLL2RDY PLL2 ready flag
  - RCC_FLAG_PLL3RDY PLL3 ready flag
  - RCC_FLAG_PLL4RDY PLL4 ready flag

Return value:

- The: new state of __FLAG__ (TRUE or FALSE).

HCLK Clock Source

RCC_HCLK_DIV1

HCLK not divided

RCC_HCLK_DIV2

HCLK divided by 2

RCC_HCLK_DIV4

HCLK divided by 4

RCC_HCLK_DIV8

HCLK divided by 8

RCC_HCLK_DIV16

HCLK divided by 16

RCC_HCLK_DIV32

HCLK divided by 32

RCC_HCLK_DIV64

HCLK divided by 64

RCC_HCLK_DIV128

HCLK divided by 128

HSE Config

RCC_HSE_OFF

HSE clock deactivation

RCC_HSE_ON

HSE clock activation

RCC_HSE_BYPASS

HSE bypass analog clock activation

RCC_HSE_BYPASS_DIGITAL

HSE bypass digital clock activation

HSI Calibration default

RCC_HSICALIBRATION_DEFAULT

Default HSI calibration trimming value

HSI Config

RCC_HSI_OFF

HSI clock deactivation

RCC_HSI_ON

HSI clock activation

HSI Clock Divider

RCC_HSI_DIV1

HSI clock is not divided

RCC_HSI_DIV2

HSI clock is divided by 2

RCC_HSI_DIV4

HSI clock is divided by 4

RCC_HSI_DIV8

HSI clock is divided by 8

IC Clock Source

RCC_ICCLKSOURCE_PLL1

ICx clock source selection is PLL1 output

RCC_ICCLKSOURCE_PLL2

ICx clock source selection is PLL2 output

RCC_ICCLKSOURCE_PLL3

ICx clock source selection is PLL3 output

RCC_ICCLKSOURCE_PLL4

ICx clock source selection is PLL4 output

Interrupt

RCC_IT_LSIRDY

LSI Ready Interrupt

RCC_IT_LSERDY

LSE Ready Interrupt

RCC_IT_MSIRDY

MSI Ready Interrupt

RCC_IT_HSIRDY

HSI Ready Interrupt

RCC_IT_HSERDY

HSE Ready Interrupt

RCC_IT_PLL1RDY

PLL1 Ready Interrupt

RCC_IT_PLL2RDY

PLL2 Ready Interrupt

RCC_IT_PLL3RDY

PLL3 Ready Interrupt

RCC_IT_PLL4RDY

PLL4 Ready Interrupt

RCC_IT_LSECSS

LSE Clock Security System Interrupt

RCC_IT_HSECSS

HSE Clock Security System Interrupt

RCC_IT_WKUP

CPU Wakeup Interrupt

RCC Private macros to check input parameters

IS_RCC_OSCILLATORTYPE

IS_RCC_HSE

IS_RCC_LSE

IS_RCC_HSI

IS_RCC_HSI_DIV

IS_RCC_LSI

IS_RCC_MSI

IS_RCC_MSI_FREQUENCY

IS_RCC_PLL

IS_RCC_PLLSOURCE

IS_RCC_PLLM_VALUE

IS_RCC_PLLN_VALUE

IS_RCC_PLLP_VALUE

IS_RCC_PLLFRACN_VALUE

IS_RCC_CLOCKTYPE

IS_RCC_CPUCLKSOURCE

IS_RCC_SYSCLKSOURCE

IS_RCC_HCLK

IS_RCC_PCLK1

IS_RCC_PCLK2

IS_RCC_PCLK4

IS_RCC_PCLK5

IS_RCC_RTCCLKSOURCE

IS_RCC_MCO

IS_RCC_MCO1SOURCE

IS_RCC_MCO2SOURCE

IS_RCC_MCODIV

IS_RCC_HSI_CALIBRATION_VALUE

IS_RCC_MSI_CALIBRATION_VALUE

IS_RCC_STOP_WAKEUPCLOCK

IS_RCC_ICCLKSOURCE

IS_RCC_ICCLKDIVIDER

IS_RCC_ITEM_ATTRIBUTES

IS_RCC_SINGLE_ITEM_ATTRIBUTES

IS_RCC_ATTRIBUTES

RCC items

RCC_ITEM_LSI

RCC_ITEM_LSE

RCC_ITEM_MSI

RCC_ITEM_HSI

RCC_ITEM_HSE

RCC_ITEM_PLL1

RCC_ITEM_PLL2

RCC_ITEM_PLL3

RCC_ITEM_PLL4

RCC_ITEM_IC1

RCC_ITEM_IC2

RCC_ITEM_IC3

RCC_ITEM_IC4

RCC_ITEM_IC5

RCC_ITEM_IC6

RCC_ITEM_IC7

RCC_ITEM_IC8

RCC_ITEM_IC9

RCC_ITEM_IC10

RCC_ITEM_IC11

RCC_ITEM_IC12

RCC_ITEM_IC13

RCC_ITEM_IC14

RCC_ITEM_IC15

RCC_ITEM_IC16

RCC_ITEM_IC17

RCC_ITEM_IC18

RCC_ITEM_IC19

RCC_ITEM_IC20

RCC_ITEM_MOD

RCC_ITEM_SYS

RCC_ITEM_BUS

RCC_ITEM_PER

RCC_ITEM_INT

RCC_ITEM_RST

RCC_ITEM_ACLKN

RCC_ITEM_ACLKNC

RCC_ITEM_AHBM

RCC_ITEM_AHB1

RCC_ITEM_AHB2

RCC_ITEM_AHB3

RCC_ITEM_AHB4

RCC_ITEM_AHB5

RCC_ITEM_APB1

RCC_ITEM_APB2

RCC_ITEM_APB3

RCC_ITEM_APB4

RCC_ITEM_APB5

RCC_ITEM_NOC

RCC_ITEM_AXISRAM3

RCC_ITEM_AXISRAM4

RCC_ITEM_AXISRAM5

RCC_ITEM_AXISRAM6

RCC_ITEM_AHBSRAM1

RCC_ITEM_AHBSRAM2

RCC_ITEM_BKPSRAM

RCC_ITEM_AXISRAM1

RCC_ITEM_AXISRAM2

RCC_ITEM_FLEXRAM

RCC_ITEM_CACHEAXIRAM

RCC_ITEM_VENCRAM

RCC_ITEM_ALL

RCC_ITEM_ALL_CLK

RCC_ITEM_ALL_PLL

RCC_ITEM_ALL_ICx

RCC_ITEM_ALL_SYSCFG

RCC_ITEM_ALL_BUS

RCC_ITEM_ALL_MEM

LSE Drive Config

RCC_LSEDRIVE_LOW

LSE low drive capability

RCC_LSEDRIVE_MEDIUMLOW

LSE medium low drive capability

RCC_LSEDRIVE_MEDIUMHIGH

LSE medium high drive capability

RCC_LSEDRIVE_HIGH

LSE high drive capability

LSE Config

RCC_LSE_OFF

LSE clock deactivation

RCC_LSE_ON

LSE clock activation

RCC_LSE_BYPASS

LSE bypass analog clock activation

RCC_LSE_BYPASS_DIGITAL

LSE bypass digital clock activation

LSI Config

RCC_LSI_OFF

LSI clock deactivation

RCC_LSI_ON

LSI clock activation

MCO1 Clock Source

RCC_MCO1SOURCE_HSI

HSI clock selected as MCO1 source (reset)

RCC_MCO1SOURCE_LSE

LSE clock selected as MCO1 source

RCC_MCO1SOURCE_MSI

MSI clock selected as MCO1 source

RCC_MCO1SOURCE_LSI

HSI clock selected as MCO1 source

RCC_MCO1SOURCE_HSE

HSE clock selected as MCO1 source

RCC_MCO1SOURCE_IC5

IC5 clock selected as MCO1 source

RCC_MCO1SOURCE_IC10

IC10 clock selected as MCO1 source

RCC_MCO1SOURCE_SYSA

SYSA CPU clock selected as MCO1 source

MCO2 Clock Source

RCC_MCO2SOURCE_HSI

HSI clock selected as MCO2 source (reset)

RCC_MCO2SOURCE_LSE

LSE clock selected as MCO2 source

RCC_MCO2SOURCE_MSI

MSI clock selected as MCO2 source

RCC_MCO2SOURCE_LSI

LSI clock selected as MCO2 source

RCC_MCO2SOURCE_HSE

HSE clock selected as MCO2 source

RCC_MCO2SOURCE_IC15

IC15 clock selected as MCO2 source

RCC_MCO2SOURCE_IC20

IC20 clock selected as MCO2 source

RCC_MCO2SOURCE_SYSB

SYSB bus clock selected as MCO2 source

MCOx Clock Prescaler

RCC_MCODIV_1

MCO divided by 1

RCC_MCODIV_2

MCO divided by 2

RCC_MCODIV_3

MCO divided by 3

RCC_MCODIV_4

MCO divided by 4

RCC_MCODIV_5

MCO divided by 5

RCC_MCODIV_6

MCO divided by 6

RCC_MCODIV_7

MCO divided by 7

RCC_MCODIV_8

MCO divided by 8

RCC_MCODIV_9

MCO divided by 9

RCC_MCODIV_10

MCO divided by 10

RCC_MCODIV_11

MCO divided by 11

RCC_MCODIV_12

MCO divided by 12

RCC_MCODIV_13

MCO divided by 13

RCC_MCODIV_14

MCO divided by 14

RCC_MCODIV_15

MCO divided by 15

RCC_MCODIV_16

MCO divided by 16 (reset)

MCOx Index

RCC_MCO1

Microcontroller Clock Output 1

RCC_MCO2

Microcontroller Clock Output 2

Misc Configuration Clock Enable Disable

__HAL_RCC_DBG_CLK_ENABLE

__HAL_RCC_DBG_CLK_DISABLE

__HAL_RCC_XSPIPHYCOMP_CLK_ENABLE

__HAL_RCC_XSPIPHYCOMP_CLK_DISABLE

__HAL_RCC_PER_CLK_ENABLE

__HAL_RCC_PER_CLK_DISABLE

Misc Configuration Clock Enabled Status

__HAL_RCC_DBG_IS_CLK_ENABLED

__HAL_RCC_MCO1_IS_CLK_ENABLED

__HAL_RCC_MCO2_IS_CLK_ENABLED

__HAL_RCC_XSPIPHYCOMP_IS_CLK_ENABLED

__HAL_RCC_PER_IS_CLK_ENABLED

Misc Configuration Clock Sleep Enable Disable

__HAL_RCC_DBG_CLK_SLEEP_ENABLE

__HAL_RCC_DBG_CLK_SLEEP_DISABLE

__HAL_RCC_XSPIPHYCOMP_CLK_SLEEP_ENABLE

__HAL_RCC_XSPIPHYCOMP_CLK_SLEEP_DISABLE

__HAL_RCC_PER_CLK_SLEEP_ENABLE

__HAL_RCC_PER_CLK_SLEEP_DISABLE

Misc Configuration Clock Sleep Enabled Status

__HAL_RCC_DBG_IS_CLK_SLEEP_ENABLED

__HAL_RCC_XSPIPHYCOMP_IS_CLK_SLEEP_ENABLED

__HAL_RCC_PER_IS_CLK_SLEEP_ENABLED

Misc Configuration Force Release Reset

__HAL_RCC_DBG_FORCE_RESET

__HAL_RCC_DBG_RELEASE_RESET

__HAL_RCC_XSPIPHY1_FORCE_RESET

__HAL_RCC_XSPIPHY1_RELEASE_RESET

__HAL_RCC_XSPIPHY2_FORCE_RESET

__HAL_RCC_XSPIPHY2_RELEASE_RESET

__HAL_RCC_SDMMC1DLL_FORCE_RESET

__HAL_RCC_SDMMC1DLL_RELEASE_RESET

__HAL_RCC_SDMMC2DLL_FORCE_RESET

__HAL_RCC_SDMMC2DLL_RELEASE_RESET

MSI Calibration default

RCC_MSICALIBRATION_DEFAULT

Default MSI calibration trimming value

MSI Config

RCC_MSI_OFF

MSI clock deactivation

RCC_MSI_ON

MSI clock activation

MSI Frequency

RCC_MSI_FREQ_4MHZ

MSI 4MHz selection

RCC_MSI_FREQ_16MHZ

MSI 16MHz selection

Oscillator Type

RCC_OSCILLATORTYPE_NONE

Oscillator configuration unchanged

RCC_OSCILLATORTYPE_HSE

HSE selected

RCC_OSCILLATORTYPE_HSI

HSI selected

RCC_OSCILLATORTYPE_LSE

LSE selected

RCC_OSCILLATORTYPE_LSI

LSI selected

RCC_OSCILLATORTYPE_MSI

MSI selected

PLL Clock Source

RCC_PLLSOURCE_HSI

HSI clock selected as PLL entry clock source

RCC_PLLSOURCE_MSI

MSI clock selected as PLL entry clock source

RCC_PLLSOURCE_HSE

HSE clock selected as PLL entry clock source

RCC_PLLSOURCE_PIN

External clock I2S_CKIN selected as PLL entry clock source

PLL Config

RCC_PLL_NONE

PLL configuration unchanged

RCC_PLL_OFF

PLL deactivation

RCC_PLL_ON

PLL activation

RCC_PLL_BYPASS

PLL activation in bypass mode with FREF set as source

PLL selection

RCC_PLL1

RCC_PLL2

RCC_PLL3

RCC_PLL4

PLL SSCG spread mode selection

RCC_PLL_SSCG_CENTER

PLL SSCG center spread modulation

RCC_PLL_SSCG_DOWN

PLL SSCG down spread modulation

RTC Clock Source

RCC_RTCCLKSOURCE_DISABLE

No clock used as RTC clock

RCC_RTCCLKSOURCE_LSE

LSE oscillator clock used as RTC clock

RCC_RTCCLKSOURCE_LSI

LSI oscillator clock used as RTC clock

RCC_RTCCLKSOURCE_HSE_DIV1

HSE oscillator clock divided by 1 used as RTC clock

RCC_RTCCLKSOURCE_HSE_DIV2

HSE oscillator clock divided by 2 used as RTC clock

RCC_RTCCLKSOURCE_HSE_DIV3

HSE oscillator clock divided by 3 used as RTC clock

RCC_RTCCLKSOURCE_HSE_DIV4

HSE oscillator clock divided by 4 used as RTC clock

RCC_RTCCLKSOURCE_HSE_DIV5

HSE oscillator clock divided by 5 used as RTC clock

RCC_RTCCLKSOURCE_HSE_DIV6

HSE oscillator clock divided by 6 used as RTC clock

RCC_RTCCLKSOURCE_HSE_DIV7

HSE oscillator clock divided by 7 used as RTC clock

RCC_RTCCLKSOURCE_HSE_DIV8

HSE oscillator clock divided by 8 used as RTC clock

RCC_RTCCLKSOURCE_HSE_DIV9

HSE oscillator clock divided by 9 used as RTC clock

RCC_RTCCLKSOURCE_HSE_DIV10

HSE oscillator clock divided by 10 used as RTC clock

RCC_RTCCLKSOURCE_HSE_DIV11

HSE oscillator clock divided by 11 used as RTC clock

RCC_RTCCLKSOURCE_HSE_DIV12

HSE oscillator clock divided by 12 used as RTC clock

RCC_RTCCLKSOURCE_HSE_DIV13

HSE oscillator clock divided by 13 used as RTC clock

RCC_RTCCLKSOURCE_HSE_DIV14

HSE oscillator clock divided by 14 used as RTC clock

RCC_RTCCLKSOURCE_HSE_DIV15

HSE oscillator clock divided by 15 used as RTC clock

RCC_RTCCLKSOURCE_HSE_DIV16

HSE oscillator clock divided by 16 used as RTC clock

RCC_RTCCLKSOURCE_HSE_DIV17

HSE oscillator clock divided by 17 used as RTC clock

RCC_RTCCLKSOURCE_HSE_DIV18

HSE oscillator clock divided by 18 used as RTC clock

RCC_RTCCLKSOURCE_HSE_DIV19

HSE oscillator clock divided by 19 used as RTC clock

RCC_RTCCLKSOURCE_HSE_DIV20

HSE oscillator clock divided by 20 used as RTC clock

RCC_RTCCLKSOURCE_HSE_DIV21

HSE oscillator clock divided by 21 used as RTC clock

RCC_RTCCLKSOURCE_HSE_DIV22

HSE oscillator clock divided by 22 used as RTC clock

RCC_RTCCLKSOURCE_HSE_DIV23

HSE oscillator clock divided by 23 used as RTC clock

RCC_RTCCLKSOURCE_HSE_DIV24

HSE oscillator clock divided by 24 used as RTC clock

RCC_RTCCLKSOURCE_HSE_DIV25

HSE oscillator clock divided by 25 used as RTC clock

RCC_RTCCLKSOURCE_HSE_DIV26

HSE oscillator clock divided by 26 used as RTC clock

RCC_RTCCLKSOURCE_HSE_DIV27

HSE oscillator clock divided by 27 used as RTC clock

RCC_RTCCLKSOURCE_HSE_DIV28

HSE oscillator clock divided by 28 used as RTC clock

RCC_RTCCLKSOURCE_HSE_DIV29

HSE oscillator clock divided by 29 used as RTC clock

RCC_RTCCLKSOURCE_HSE_DIV30

HSE oscillator clock divided by 30 used as RTC clock

RCC_RTCCLKSOURCE_HSE_DIV31

HSE oscillator clock divided by 31 used as RTC clock

RCC_RTCCLKSOURCE_HSE_DIV32

HSE oscillator clock divided by 32 used as RTC clock

RCC_RTCCLKSOURCE_HSE_DIV33

HSE oscillator clock divided by 33 used as RTC clock

RCC_RTCCLKSOURCE_HSE_DIV34

HSE oscillator clock divided by 34 used as RTC clock

RCC_RTCCLKSOURCE_HSE_DIV35

HSE oscillator clock divided by 35 used as RTC clock

RCC_RTCCLKSOURCE_HSE_DIV36

HSE oscillator clock divided by 36 used as RTC clock

RCC_RTCCLKSOURCE_HSE_DIV37

HSE oscillator clock divided by 37 used as RTC clock

RCC_RTCCLKSOURCE_HSE_DIV38

HSE oscillator clock divided by 38 used as RTC clock

RCC_RTCCLKSOURCE_HSE_DIV39

HSE oscillator clock divided by 39 used as RTC clock

RCC_RTCCLKSOURCE_HSE_DIV40

HSE oscillator clock divided by 40 used as RTC clock

RCC_RTCCLKSOURCE_HSE_DIV41

HSE oscillator clock divided by 41 used as RTC clock

RCC_RTCCLKSOURCE_HSE_DIV42

HSE oscillator clock divided by 42 used as RTC clock

RCC_RTCCLKSOURCE_HSE_DIV43

HSE oscillator clock divided by 43 used as RTC clock

RCC_RTCCLKSOURCE_HSE_DIV44

HSE oscillator clock divided by 44 used as RTC clock

RCC_RTCCLKSOURCE_HSE_DIV45

HSE oscillator clock divided by 45 used as RTC clock

RCC_RTCCLKSOURCE_HSE_DIV46

HSE oscillator clock divided by 46 used as RTC clock

RCC_RTCCLKSOURCE_HSE_DIV47

HSE oscillator clock divided by 47 used as RTC clock

RCC_RTCCLKSOURCE_HSE_DIV48

HSE oscillator clock divided by 48 used as RTC clock

RCC_RTCCLKSOURCE_HSE_DIV49

HSE oscillator clock divided by 49 used as RTC clock

RCC_RTCCLKSOURCE_HSE_DIV50

HSE oscillator clock divided by 50 used as RTC clock

RCC_RTCCLKSOURCE_HSE_DIV51

HSE oscillator clock divided by 51 used as RTC clock

RCC_RTCCLKSOURCE_HSE_DIV52

HSE oscillator clock divided by 52 used as RTC clock

RCC_RTCCLKSOURCE_HSE_DIV53

HSE oscillator clock divided by 53 used as RTC clock

RCC_RTCCLKSOURCE_HSE_DIV54

HSE oscillator clock divided by 54 used as RTC clock

RCC_RTCCLKSOURCE_HSE_DIV55

HSE oscillator clock divided by 55 used as RTC clock

RCC_RTCCLKSOURCE_HSE_DIV56

HSE oscillator clock divided by 56 used as RTC clock

RCC_RTCCLKSOURCE_HSE_DIV57

HSE oscillator clock divided by 57 used as RTC clock

RCC_RTCCLKSOURCE_HSE_DIV58

HSE oscillator clock divided by 58 used as RTC clock

RCC_RTCCLKSOURCE_HSE_DIV59

HSE oscillator clock divided by 59 used as RTC clock

RCC_RTCCLKSOURCE_HSE_DIV60

HSE oscillator clock divided by 60 used as RTC clock

RCC_RTCCLKSOURCE_HSE_DIV61

HSE oscillator clock divided by 61 used as RTC clock

RCC_RTCCLKSOURCE_HSE_DIV62

HSE oscillator clock divided by 62 used as RTC clock

RCC_RTCCLKSOURCE_HSE_DIV63

HSE oscillator clock divided by 63 used as RTC clock

RCC_RTCCLKSOURCE_HSE_DIV64

HSE oscillator clock divided by 64 used as RTC clock

Stop WakeUp Clock

RCC_STOP_WAKEUPCLOCK_HSI

HSI selected as wake up system clock from system Stop (default after reset)

RCC_STOP_WAKEUPCLOCK_MSI

MSI selected as wake up system clock from system Stop

System Bus Clock Source

RCC_SYSCLKSOURCE_HSI

HSI selection as system bus clocks

RCC_SYSCLKSOURCE_MSI

MSI selection as system bus clocks

RCC_SYSCLKSOURCE_HSE

HSE selection as system bus clocks

RCC_SYSCLKSOURCE_IC2_IC6_IC11

IC2/IC6/IC11 selection as system bus clocks

System Bus Clock Source Status

RCC_SYSCLKSOURCE_STATUS_HSI

HSI used as system bus clocks

RCC_SYSCLKSOURCE_STATUS_MSI

MSI used as system bus clocks

RCC_SYSCLKSOURCE_STATUS_HSE

HSE used as system bus clocks

RCC_SYSCLKSOURCE_STATUS_IC2_IC6_IC11

IC2/IC6/IC11 used as system bus clocks

57 HAL RCC Extension Driver

57.1 RCCEx Firmware driver registers structures

57.1.1 RCC_PeriphCLKInitTypeDef

RCC_PeriphCLKInitTypeDef is defined in the stm32n6xx_hal_rcc_ex.h

Data Fields

- ***uint64_t PeriphClockSelection***
- ***RCC_ICInitTypeDef ICSelection***
- ***uint32_t FmcClockSelection***
- ***uint32_t Xspi1ClockSelection***
- ***uint32_t Xspi2ClockSelection***
- ***uint32_t Xspi3ClockSelection***
- ***uint32_t CkperClockSelection***
- ***uint32_t AdcClockSelection***
- ***uint32_t AdcDivider***
- ***uint32_t Adf1ClockSelection***
- ***uint32_t DcmippClockSelection***
- ***uint32_t Eth1ClockSelection***
- ***uint32_t Eth1PhyInterfaceSelection***
- ***uint32_t Eth1RxClockSelection***
- ***uint32_t Eth1TxClockSelection***
- ***uint32_t Eth1PtpClockSelection***
- ***uint32_t Eth1PtpDivider***
- ***uint32_t FdcanClockSelection***
- ***uint32_t I2c1ClockSelection***
- ***uint32_t I2c2ClockSelection***
- ***uint32_t I2c3ClockSelection***
- ***uint32_t I2c4ClockSelection***
- ***uint32_t I3c1ClockSelection***
- ***uint32_t I3c2ClockSelection***
- ***uint32_t Lptim1ClockSelection***
- ***uint32_t Lptim2ClockSelection***
- ***uint32_t Lptim3ClockSelection***
- ***uint32_t Lptim4ClockSelection***
- ***uint32_t Lptim5ClockSelection***
- ***uint32_t Lpuart1ClockSelection***
- ***uint32_t LtdcClockSelection***
- ***uint32_t Mdf1ClockSelection***
- ***uint32_t PssiClockSelection***
- ***uint32_t Sai1ClockSelection***
- ***uint32_t Sai2ClockSelection***
- ***uint32_t Sdmmc1ClockSelection***
- ***uint32_t Sdmmc2ClockSelection***
- ***uint32_t Spi1ClockSelection***
- ***uint32_t Spi2ClockSelection***
- ***uint32_t Spi3ClockSelection***
- ***uint32_t Spi4ClockSelection***

- ***uint32_t Spi5ClockSelection***
- ***uint32_t Spi6ClockSelection***
- ***uint32_t Spdifrx1ClockSelection***
- ***uint32_t Usart1ClockSelection***
- ***uint32_t Usart2ClockSelection***
- ***uint32_t Usart3ClockSelection***
- ***uint32_t Uart4ClockSelection***
- ***uint32_t Uart5ClockSelection***
- ***uint32_t Usart6ClockSelection***
- ***uint32_t Uart7ClockSelection***
- ***uint32_t Uart8ClockSelection***
- ***uint32_t Uart9ClockSelection***
- ***uint32_t Usart10ClockSelection***
- ***uint32_t UsbPhy1ClockSelection***
- ***uint32_t UsbOtgHs1ClockSelection***
- ***uint32_t UsbPhy2ClockSelection***
- ***uint32_t UsbOtgHs2ClockSelection***
- ***uint32_t RTCClockSelection***
- ***uint32_t TIMPresSelection***

Field Documentation

- ***uint64_t RCC_PeriphCLKInitTypeDef::PeriphClockSelection***
  The Extended Clock to be configured. This parameter can be a value of ***RCCEx_Periph_Clock_Selection***
- ***RCC_ICInitTypeDef RCC_PeriphCLKInitTypeDef::ICSelection[RCC_IC_MAX_NUMBER]***
  ICx structure parameters. This parameter shall be used when ICx is selected as kernel clock source for some peripherals
- ***uint32_t RCC_PeriphCLKInitTypeDef::FmcClockSelection***
  Specifies FMC clock source. This parameter can be a value of ***RCCEx_FMC_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Xspi1ClockSelection***
  Specifies XSPI1 clock source. This parameter can be a value of ***RCCEx_XSPI1_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Xspi2ClockSelection***
  Specifies XSPI2 clock source. This parameter can be a value of ***RCCEx_XSPI2_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Xspi3ClockSelection***
  Specifies XSPI3 clock source. This parameter can be a value of ***RCCEx_XSPI3_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::CkperClockSelection***
  Specifies CKPER clock source. This parameter can be a value of ***RCCEx_CLKP_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::AdcClockSelection***
  Specifies ADC interface clock source. This parameter can be a value of ***RCCEx_ADC_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::AdcDivider***
  Specifies ADC clock divider. This parameter can be a value between 1 to 256
- ***uint32_t RCC_PeriphCLKInitTypeDef::Adf1ClockSelection***
  Specifies ADF1 Clock clock source. This parameter can be a value of ***RCCEx_ADF1_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::DcmippClockSelection***
  Specifies DCMIPP clock source. This parameter can be a value of ***RCCEx_DCMIPP_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Eth1ClockSelection***
  Specifies ETH1 clock source. This parameter can be a value of ***RCCEx_ETH1_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Eth1PhyInterfaceSelection***
  Specifies ETH1 PHY interface. This parameter can be a value of ***RCCEx_ETH1_PHY_Interface***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Eth1RxClockSelection***
  Specifies ETH1 RX clock source. This parameter can be a value of ***RCCEx_ETH1_RX_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Eth1TxClockSelection***
  Specifies ETH1 TX clock source. This parameter can be a value of ***RCCEx_ETH1_TX_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Eth1PtpClockSelection***
  Specifies ETH1 PTP clock source. This parameter can be a value of ***RCCEx_ETH1_PTP_Clock_Source***

- ***uint32_t RCC_PeriphCLKInitTypeDef::Eth1PtpDivider***
  Specifies ETH1 PTP clock divider. This parameter can be a value between 1 to 16
- ***uint32_t RCC_PeriphCLKInitTypeDef::FdcanClockSelection***
  Specifies FDCAN kernel clock source. This parameter can be a value of ***RCCEx_FDCAN_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::I2c1ClockSelection***
  Specifies I2C1 clock source. This parameter can be a value of ***RCCEx_I2C1_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::I2c2ClockSelection***
  Specifies I2C2 clock source. This parameter can be a value of ***RCCEx_I2C2_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::I2c3ClockSelection***
  Specifies I2C3 clock source. This parameter can be a value of ***RCCEx_I2C3_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::I2c4ClockSelection***
  Specifies I2C4 clock source. This parameter can be a value of ***RCCEx_I2C4_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::I3c1ClockSelection***
  Specifies I3C1 clock source. This parameter can be a value of ***RCCEx_I3C1_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::I3c2ClockSelection***
  Specifies I3C2 clock source. This parameter can be a value of ***RCCEx_I3C2_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Lptim1ClockSelection***
  Specifies LPTIM1 clock source. This parameter can be a value of ***RCCEx_LPTIM1_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Lptim2ClockSelection***
  Specifies LPTIM2 clock source. This parameter can be a value of ***RCCEx_LPTIM2_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Lptim3ClockSelection***
  Specifies LPTIM3 clock source. This parameter can be a value of ***RCCEx_LPTIM3_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Lptim4ClockSelection***
  Specifies LPTIM4 clock source. This parameter can be a value of ***RCCEx_LPTIM4_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Lptim5ClockSelection***
  Specifies LPTIM5 clock source. This parameter can be a value of ***RCCEx_LPTIM5_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Lpuart1ClockSelection***
  Specifies LPUART1 clock source. This parameter can be a value of ***RCCEx_LPUART1_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::LtdcClockSelection***
  Specifies LPUART1 clock source. This parameter can be a value of ***RCCEx_LTDC_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Mdf1ClockSelection***
  Specifies MDF1 Clock clock source. This parameter can be a value of ***RCCEx_MDF1_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::PssiClockSelection***
  Specifies PSSI clock source. This parameter can be a value of ***RCCEx_PSSI_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Sai1ClockSelection***
  Specifies SAI1 clock source. This parameter can be a value of ***RCCEx_SAI1_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Sai2ClockSelection***
  Specifies SAI2 clock source. This parameter can be a value of ***RCCEx_SAI2_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Sdmmc1ClockSelection***
  Specifies SDMMC1 clock source. This parameter can be a value of ***RCCEx_SDMMC1_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Sdmmc2ClockSelection***
  Specifies SDMMC2 clock source. This parameter can be a value of ***RCCEx_SDMMC2_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Spi1ClockSelection***
  Specifies SPI1 clock source. This parameter can be a value of ***RCCEx_SPI1_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Spi2ClockSelection***
  Specifies SPI2 clock source. This parameter can be a value of ***RCCEx_SPI2_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Spi3ClockSelection***
  Specifies SPI3 clock source. This parameter can be a value of ***RCCEx_SPI3_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Spi4ClockSelection***
  Specifies SPI4 clock source. This parameter can be a value of ***RCCEx_SPI4_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Spi5ClockSelection***
  Specifies SPI5 clock source. This parameter can be a value of ***RCCEx_SPI5_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Spi6ClockSelection***
  Specifies SPI6 clock source. This parameter can be a value of ***RCCEx_SPI6_Clock_Source***

- ***uint32_t RCC_PeriphCLKInitTypeDef::Spdifrx1ClockSelection***
  Specifies SPDIFRX1 Clock clock source. This parameter can be a value of ***RCCEx_SPDIFRX1_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Usart1ClockSelection***
  Specifies USART1 clock source. This parameter can be a value of ***RCCEx_USART1_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Usart2ClockSelection***
  Specifies USART2 clock source. This parameter can be a value of ***RCCEx_USART2_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Usart3ClockSelection***
  Specifies USART3 clock source. This parameter can be a value of ***RCCEx_USART3_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Uart4ClockSelection***
  Specifies UART4 clock source. This parameter can be a value of ***RCCEx_UART4_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Uart5ClockSelection***
  Specifies UART5 clock source. This parameter can be a value of ***RCCEx_UART5_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Usart6ClockSelection***
  Specifies USART6 clock source. This parameter can be a value of ***RCCEx_USART6_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Uart7ClockSelection***
  Specifies UART7 clock source. This parameter can be a value of ***RCCEx_UART7_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Uart8ClockSelection***
  Specifies UART8 clock source. This parameter can be a value of ***RCCEx_UART8_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Uart9ClockSelection***
  Specifies UART9 clock source. This parameter can be a value of ***RCCEx_UART9_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Usart10ClockSelection***
  Specifies USART10 clock source. This parameter can be a value of ***RCCEx_USART10_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::UsbPhy1ClockSelection***
  Specifies USBPHY1 clock source. This parameter can be a value of ***RCCEx_USBPHY1_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::UsbOtgHs1ClockSelection***
  Specifies USB OTG HS1 clock source. This parameter can be a value of ***RCCEx_USB_OTGHS1_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::UsbPhy2ClockSelection***
  Specifies USBPHY2 clock source. This parameter can be a value of ***RCCEx_USBPHY2_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::UsbOtgHs2ClockSelection***
  Specifies USB OTG HS2 clock source. This parameter can be a value of ***RCCEx_USB_OTGHS2_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::RTCClockSelection***
  Specifies RTC Clock clock source. This parameter can be a value of ***RCC_RTC_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::TIMPresSelection***
  Specifies TIM Clock Prescalers Selection. This parameter can be a value of ***RCCEx_TIM_Prescaler_Selection***

57.1.2 RCC_PLLSSCGInitTypeDef

RCC_PLLSSCGInitTypeDef is defined in the stm32n6xx_hal_rcc_ex.h

Data Fields

- ***uint32_t PLLModDiv***
- ***uint32_t PLLModSpreadDepth***
- ***uint32_t PLLModSpreadMode***

Field Documentation

- ***uint32_t RCC_PLLSSCGInitTypeDef::PLLModDiv***
  Modulation division frequency. This parameter must be a number between Min_Data = 0 and Max_Data = 0xF
- ***uint32_t RCC_PLLSSCGInitTypeDef::PLLModSpreadDepth***
  Modulation spread spectrum depth. This parameter must be a number between Min_Data = 0 and Max_Data = 0x1F
- ***uint32_t RCC_PLLSSCGInitTypeDef::PLLModSpreadMode***
  Modulation spread spectrum mode. This parameter must be a value of ***RCC_PLL_SSCGSpread_Mode***

57.2 RCCEx Firmware driver API description

The following section lists the various functions of the RCCEx library.

57.2.1 Extended Peripheral Control functions

This subsection provides a set of functions allowing to control the RCC Clocks frequencies.

Note: Important note: Care must be taken when HAL_RCCEx_PeriphCLKConfig() is used to select the RTC clock source; in this case the Backup domain will be reset in order to modify the RTC Clock source, as consequence RTC registers (including the backup registers) and RCC_BDCR register are set to their reset values.

This section contains the following APIs:

- *HAL_RCCEx_PeriphCLKConfig()*
- *HAL_RCCEx_GetPeriphCLKConfig()*
- *HAL_RCCEx_GetPeriphCLKFreq()*
- *HAL_RCCEx_GetPLL1CLKFreq()*
- *HAL_RCCEx_GetPLL2CLKFreq()*
- *HAL_RCCEx_GetPLL3CLKFreq()*
- *HAL_RCCEx_GetPLL4CLKFreq()*

57.2.2 Detailed description of functions

HAL_RCCEx_PeriphCLKConfig

Function name

HAL_StatusTypeDef HAL_RCCEx_PeriphCLKConfig (const RCC_PeriphCLKInitTypeDef * PeriphClkInit)

Function description

Initializes the RCC extended peripherals clocks according to the specified parameters in the RCC_PeriphCLKInitTypeDef.

Parameters

- **PeriphClkInit:** pointer to an RCC_PeriphCLKInitTypeDef structure that contains the configuration information for the Extended Peripherals clocks (ADC, ADF1, CKPER, DCMI, DCMIPP, FDCAN, FMC, I2C1, I2C2, I2C3, I2C4, I3C1, I3C2, LPTIM1, LPTIM2, LPTIM3, LPTIM4, LPTIM5, LTDC, LPUART1, MDF1, PSSI, RTC, SAI1, SAI2, SDMMC1, SDMMC2, SPDIFRX1, SPI1, SPI2, SPI3, SPI4, SPI5, SPI6, TIM, USART1, USART2, USART3, UART4, UART5, USART6, UART7, UART8, UART9, USART10, USBPHY1, USBPHY2, USB OTGHS1, USB OTGHS2, XSPI1, XSPI2, XSPI3).

Return values

- **HAL:** status

Notes

- Care must be taken when HAL_RCCEx_PeriphCLKConfig() is used to select the RTC clock source; in this case the Backup domain will be reset in order to modify the RTC Clock source, as consequence RTC registers (including the backup registers) are set to their reset values.
- Dynamic switches are used for peripheral input clock selection meaning the switch setting can only be changed if both input clocks are present during transition time.
- Care must be taken when RCC_PPPCLKSOURCE_ICx is selected. The ICx ClockSelection and ClockDivider fields modification indeed impacts all peripherals using this ICx as clock source.

HAL_RCCEx_GetPeriphCLKConfig

Function name

void HAL_RCCEx_GetPeriphCLKConfig (RCC_PeriphCLKInitTypeDef * PeriphClkInit)

Function description

Get the RCC_ClkInitStruct according to the internal RCC configuration registers.

Parameters

- **PeriphClkInit:** pointer to an RCC_PeriphCLKInitTypeDef structure that returns the configuration information for the Extended Peripherals clocks : (ADC, ADF1, CKPER, DCMI, DCMIPP, FDCAN, FMC, I2C1, I2C2, I2C3, I2C4, I3C1, I3C2, LPTIM1, LPTIM2, LPTIM3, LPTIM4, LPTIM5, LTDC, LPUART1, MDF1, PSSI, RTC, SAI1, SAI2, SDMMC1, SDMMC2, SPDIFRX1, SPI1, SPI2, SPI3, SPI4, SPI5, SPI6, TIM, USART1, USART2, USART3, UART4, UART5, USART6, UART7, UART8, UART9, USART10, USBPHYC, USB OTGFS, XSPI1, XSPI2, XSPI3).

Return values

- **None:**

HAL_RCCEx_GetPeriphCLKFreq

Function name

uint32_t HAL_RCCEx_GetPeriphCLKFreq (uint64_t PeriphClk)

Function description

Return the peripheral clock frequency for a given peripheral (SAI..)

Parameters

- **PeriphClk:** Peripheral clock identifier This parameter can be one of the following values:
  - RCC_PERIPHCLK_ADC : ADC peripheral clock
  - RCC_PERIPHCLK_ADF1 : ADF1 peripheral clock
  - RCC_PERIPHCLK_CSI : CSI peripheral clock
  - RCC_PERIPHCLK_DCMIPP : DCMIPP peripheral clock
  - RCC_PERIPHCLK_I2C1 : I2C1 peripheral clock
  - RCC_PERIPHCLK_I2C2 : I2C2 peripheral clock
  - RCC_PERIPHCLK_I2C3 : I2C3 peripheral clock
  - RCC_PERIPHCLK_I2C4 : I2C4 peripheral clock
  - RCC_PERIPHCLK_I3C1 : I3C1 peripheral clock
  - RCC_PERIPHCLK_I3C2 : I3C2 peripheral clock
  - RCC_PERIPHCLK_FDCAN : FDCAN peripheral clock
  - RCC_PERIPHCLK_FMC : FMC peripheral clock
  - RCC_PERIPHCLK_LPTIM1 : LPTIM1 peripheral clock
  - RCC_PERIPHCLK_LPTIM2 : LPTIM2 peripheral clock
  - RCC_PERIPHCLK_LPTIM3 : LPTIM3 peripheral clock
  - RCC_PERIPHCLK_LPTIM4 : LPTIM4 peripheral clock
  - RCC_PERIPHCLK_LPTIM5 : LPTIM5 peripheral clock
  - RCC_PERIPHCLK_LPUART1 : LPUART1 peripheral clock
  - RCC_PERIPHCLK_LTDC : LTDC peripheral clock
  - RCC_PERIPHCLK_MDF1 : MDF1 peripheral clock
  - RCC_PERIPHCLK_PSSI : PSSI peripheral clock
  - RCC_PERIPHCLK_SAI1 : SAI1 peripheral clock
  - RCC_PERIPHCLK_SAI2 : SAI2 peripheral clock
  - RCC_PERIPHCLK_SDMMC1 : SDMMC1 peripheral clock
  - RCC_PERIPHCLK_SDMMC2 : SDMMC2 peripheral clock
  - RCC_PERIPHCLK_SPDIFRX1 : SPDIFRX1 peripheral clock
  - RCC_PERIPHCLK_SPI1 : SPI1 peripheral clock
  - RCC_PERIPHCLK_SPI2 : SPI2 peripheral clock
  - RCC_PERIPHCLK_SPI3 : SPI3 peripheral clock
  - RCC_PERIPHCLK_SPI4 : SPI4 peripheral clock
  - RCC_PERIPHCLK_SPI5 : SPI5 peripheral clock
  - RCC_PERIPHCLK_SPI6 : SPI6 peripheral clock
  - RCC_PERIPHCLK_USART1 : USART1 peripheral clock
  - RCC_PERIPHCLK_USART2 : USART2 peripheral clock
  - RCC_PERIPHCLK_USART3 : USART3 peripheral clock
  - RCC_PERIPHCLK_UART4 : UART4 peripheral clock
  - RCC_PERIPHCLK_UART5 : UART5 peripheral clock
  - RCC_PERIPHCLK_USART6 : USART6 peripheral clock
  - RCC_PERIPHCLK_UART7 : UART7 peripheral clock
  - RCC_PERIPHCLK_UART8 : UART8 peripheral clock
  - RCC_PERIPHCLK_UART9 : UART9 peripheral clock
  - RCC_PERIPHCLK_USART10 : USART10 peripheral clock
  - RCC_PERIPHCLK_XSPI1 : XSPI1 peripheral clock
  - RCC_PERIPHCLK_XSPI2 : XSPI2 peripheral clock
  - RCC_PERIPHCLK_XSPI3 : XSPI3 peripheral clock

Return values

- **Frequency:** in KHz

Notes

- Return 0 if peripheral clock identifier not managed by this API or if the selected clock source is not enabled (HSI, PLLs clock output..)

HAL_RCCEx_GetPLL1CLKFreq

Function name

uint32_t HAL_RCCEx_GetPLL1CLKFreq (void)

Function description

Return PLL1 clock frequency.

Return values

- **PLL1:** output clock frequency

Notes

- RCC_PERIPH_FREQUENCY_NO returned for non activated output or oscillator not ready

HAL_RCCEx_GetPLL2CLKFreq

Function name

uint32_t HAL_RCCEx_GetPLL2CLKFreq (void)

Function description

Return PLL2 clock frequency.

Return values

- **PLL2:** output clock frequency

Notes

- RCC_PERIPH_FREQUENCY_NO returned for non activated output or oscillator not ready

HAL_RCCEx_GetPLL3CLKFreq

Function name

uint32_t HAL_RCCEx_GetPLL3CLKFreq (void)

Function description

Return PLL3 clock frequency.

Return values

- **PLL3:** output clock frequency

Notes

- RCC_PERIPH_FREQUENCY_NO returned for non activated output or oscillator not ready

HAL_RCCEx_GetPLL4CLKFreq

Function name

uint32_t HAL_RCCEx_GetPLL4CLKFreq (void)

Function description

Return PLL4 clock frequency.

Return values

- **PLL4:** output clock frequency

Notes

- RCC_PERIPH_FREQUENCY_NO returned for non activated output or oscillator not ready

HAL_RCCEx_WakeUpStopCLKConfig

Function name

void HAL_RCCEx_WakeUpStopCLKConfig (uint32_t WakeUpClk)

Function description

Configure the oscillator clock source for wakeup from Stop and CSS backup clock.

Parameters

- **WakeUpClk:** Wakeup clock This parameter can be one of the following values:
  - RCC_STOP_WAKEUPCLOCK_MSI: MSI oscillator selection
  - RCC_STOP_WAKEUPCLOCK_HSI: HSI oscillator selection

Return values

- **None:**

Notes

- This function shall not be called after the Clock Security System on HSE has been enabled and the system clock is HSE or a switch on HSE is requested.

HAL_RCCEx_EnableLSECSS

Function name

void HAL_RCCEx_EnableLSECSS (void)

Function description

Enable the LSE Clock Security System.

Return values

- **None:**

Notes

- Prior to enable the LSE Clock Security System, LSE oscillator is to be enabled with HAL_RCC_OscConfig() and the LSE oscillator clock is to be selected as RTC clock with HAL_RCCEx_PeriphCLKConfig().
- Backup domain access should be enabled

HAL_RCCEx_DisableLSECSS

Function name

void HAL_RCCEx_DisableLSECSS (void)

Function description

Disable the LSE Clock Security System.

Return values

- **None:**

Notes

- LSE Clock Security System can only be disabled after a LSE failure detection.
- Backup domain access should be enabled

HAL_RCCEx_EnableLSECSS_IT

Function name

void HAL_RCCEx_EnableLSECSS_IT (void)

Function description

Enable the LSE Clock Security System Interrupt.

Return values

- **None:**

Notes

- LSE Clock Security System Interrupt is mapped on EXTI line 65 which must be enabled at application level to wake up from low power modes

HAL_RCCEx_DisableLSECSS_IT

Function name

void HAL_RCCEx_DisableLSECSS_IT (void)

Function description

Disable the LSE Clock Security System Interrupt.

Return values

- **None:**

HAL_RCCEx_ReArmLSECSS

Function name

void HAL_RCCEx_ReArmLSECSS (void)

Function description

Rearm the LSE Clock Security System.

Return values

- **None:**

Notes

- Allow to re-arm the LSE Clock Security System after a LSE failure detection

HAL_RCCEx_LSECSS_IRQHandler

Function name

void HAL_RCCEx_LSECSS_IRQHandler (void)

Function description

Handle the RCC LSE Clock Security System interrupt request.

Return values

- **None:**

HAL_RCCEx_LSECSS_Callback

Function name

void HAL_RCCEx_LSECSS_Callback (void)

Function description

RCCEx LSE Clock Security System interrupt callback.

Return values

- **none:**

HAL_RCCEx_PLLSSCGConfig

Function name

HAL_StatusTypeDef HAL_RCCEx_PLLSSCGConfig (uint32_t PLLnumber, const RCC_PLLInitTypeDef * pPLLInit, const RCC_PLLSSCGInitTypeDef * pPLLSSCGInit)

Function description

Configure the requested PLL to SSCG Mode.

Parameters

- **PLLnumber:** PLL number to configure. This parameter can be one of the following values: PLL selection
- **pPLLInit:** Pointer to an RCC_PLLInitTypeDef structure that contains the configuration parameters.
- **pPLLSSCGInit:** Pointer to an RCC_PLLSSCGInitTypeDef structure that contains the configuration parameters specific to SSCG.

Return values

- **HAL:** status

Notes

- PLL is temporary disabled to apply new parameters

57.3 RCCEx Firmware driver defines

The following section lists the various define and macros of the module.

57.3.1 RCCEx

RCCEx

ADC Clock Source

RCC_ADCCLKSOURCE_HCLK

RCC_ADCCLKSOURCE_CLKP

RCC_ADCCLKSOURCE_IC7

RCC_ADCCLKSOURCE_IC8

RCC_ADCCLKSOURCE_MSI

RCC_ADCCLKSOURCE_HSI

RCC_ADCCLKSOURCE_PIN

RCC_ADCCLKSOURCE_TIMG

ADF1 Clock Source

RCC_ADF1CLKSOURCE_HCLK

RCC_ADF1CLKSOURCE_CLKP

RCC_ADF1CLKSOURCE_IC7

RCC_ADF1CLKSOURCE_IC8

RCC_ADF1CLKSOURCE_MSI

RCC_ADF1CLKSOURCE_HSI

RCC_ADF1CLKSOURCE_PIN

RCC_ADF1CLKSOURCE_TIMG

CLKP Clock Source

RCC_CLKPCLKSOURCE_HSI

RCC_CLKPCLKSOURCE_MSI

RCC_CLKPCLKSOURCE_HSE

RCC_CLKPCLKSOURCE_IC5

RCC_CLKPCLKSOURCE_IC10

RCC_CLKPCLKSOURCE_IC15

RCC_CLKPCLKSOURCE_IC19

RCC_CLKPCLKSOURCE_IC20

DCMIPP Clock Source

RCC_DCMIPPCLKSOURCE_PCLK5

RCC_DCMIPPCLKSOURCE_CLKP

RCC_DCMIPPCLKSOURCE_IC17

RCC_DCMIPPCLKSOURCE_HSI

ETH1 Clock Source

RCC_ETH1CLKSOURCE_HCLK

RCC_ETH1CLKSOURCE_CLKP

RCC_ETH1CLKSOURCE_IC12

RCC_ETH1CLKSOURCE_HSE

ETH1 PHY Interface

RCC_ETH1PHYIF_MII

RCC_ETH1PHYIF_RGMII

RCC_ETH1PHYIF_RMII

ETH1 PTP Clock Source

RCC_ETH1PTPCLKSOURCE_HCLK

RCC_ETH1PTPCLKSOURCE_CLKP

RCC_ETH1PTPCLKSOURCE_IC13

RCC_ETH1PTPCLKSOURCE_HSE

ETH1 RX Clock Source

RCC_ETH1RXCLKSOURCE_EXT

RCC_ETH1RXCLKSOURCE_INT

ETH1 TX Clock Source

RCC_ETH1TXCLKSOURCE_EXT

RCC_ETH1TXCLKSOURCE_INT

RCCEx Exported Macros

__HAL_RCC_PLL1_MODSPR_CONFIG

Description:

- Macro to configure the PLL1 SSCG Spread Depth.

Parameters:

- __PLLMODSPR__: specifies the SSCG modulation spread depth. It should be a value between 0 and 0x1F.

Return value:

- None

Notes:

- This configuration cannot be requested when the PLL1 has been enabled.

__HAL_RCC_PLL1_MODSPRDW_CONFIG

Description:

- Macro to configure the PLL1 SSCG Spread Mode.

Parameters:

- __PLLMODSPRDW__: specifies the SSCG modulation spread mode. It should be a value of

Return value:

- None

Notes:

- This configuration cannot be requested when the PLL1 has been enabled.

__HAL_RCC_PLL2_MODSPR_CONFIG

Description:

- Macro to configure the PLL2 SSCG Spread Depth.

Parameters:

- __PLLMODSPR__: specifies the SSCG modulation spread depth. It should be a value between 0 and 0x1F.

Return value:

- None

Notes:

- This configuration cannot be requested when the PLL2 has been enabled.

__HAL_RCC_PLL2_MODSPRDW_CONFIG

Description:

- Macro to configure the PLL2 SSCG Spread Mode.

Parameters:

- __PLLMODSPRDW__: specifies the SSCG modulation spread mode. It should be a value of

Return value:

- None

Notes:

- This configuration cannot be requested when the PLL2 has been enabled.

__HAL_RCC_PLL3_MODSPR_CONFIG

Description:

- Macro to configure the PLL3 SSCG Spread Depth.

Parameters:

- __PLLMODSPR__: specifies the SSCG modulation spread depth. It should be a value between 0 and 0x1F.

Return value:

- None

Notes:

- This configuration cannot be requested when the PLL3 has been enabled.

__HAL_RCC_PLL3_MODSPRDW_CONFIG

Description:

- Macro to configure the PLL3 SSCG Spread Mode.

Parameters:

- __PLLMODSPRDW__: specifies the SSCG modulation spread mode. It should be a value of

Return value:

- None

Notes:

- This configuration cannot be requested when the PLL3 has been enabled.

__HAL_RCC_PLL4_MODSPR_CONFIG

Description:

- Macro to configure the PLL4 SSCG Spread Depth.

Parameters:

- __PLLMODSPR__: specifies the SSCG modulation spread depth. It should be a value between 0 and 0x1F.

Return value:

- None

Notes:

- This configuration cannot be requested when the PLL4 has been enabled.

__HAL_RCC_PLL4_MODSPRDW_CONFIG

Description:

- Macro to configure the PLL4 SSCG Spread Mode.

Parameters:

- __PLLMODSPRDW__: specifies the SSCG modulation spread mode. It should be a value of

Return value:

- None

Notes:

- This configuration cannot be requested when the PLL4 has been enabled.

__HAL_RCC_ADC_CONFIG

Description:

- Macro to configure the ADC clock.

Parameters:

- __ADC_CLKSOURCE__: specifies the ADC clock source. This parameter can be one of the following values:
  - RCC_ADCCLKSOURCE_HCLK HCLK Clock selected as ADC clock
  - RCC_ADCCLKSOURCE_CLKP CLKP selected as ADC clock
  - RCC_ADCCLKSOURCE_IC7 IC7 selected as ADC clock
  - RCC_ADCCLKSOURCE_IC8 IC8 selected as ADC clock
  - RCC_ADCCLKSOURCE_MSI MSI selected as ADC clock
  - RCC_ADCCLKSOURCE_HSI HSI selected as ADC clock
  - RCC_ADCCLKSOURCE_PIN External I2S_CKIN selected as ADC clock
  - RCC_ADCCLKSOURCE_TIMG TIMG selected as ADC clock

__HAL_RCC_GET_ADC_SOURCE

Description:

- Macro to get the ADC clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_ADCCLKSOURCE_HCLK HCLK Clock selected as ADC clock
  - RCC_ADCCLKSOURCE_CLKP CLKP selected as ADC clock
  - RCC_ADCCLKSOURCE_IC7 IC7 selected as ADC clock
  - RCC_ADCCLKSOURCE_IC8 IC8 selected as ADC clock
  - RCC_ADCCLKSOURCE_MSI MSI selected as ADC clock
  - RCC_ADCCLKSOURCE_HSI HSI selected as ADC clock
  - RCC_ADCCLKSOURCE_PIN External I2S_CKIN selected as ADC clock
  - RCC_ADCCLKSOURCE_TIMG TIMG selected as ADC clock

__HAL_RCC_ADC_DIVIDER_CONFIG

Description:

- Macro to configure the ADC clock divider.

Parameters:

- __ADC_DIVIDER__: specifies clock divider for ADC. This parameter can be a value between 1 and 256.

__HAL_RCC_GET_ADC_DIVIDER

Description:

- Macro to get the ADC clock divider.

Return value:

- The: divider can be a value between 1 and 256.

__HAL_RCC_ADF1_CONFIG

Description:

- Macro to configure the ADF1 clock.

Parameters:

- __ADF1_CLKSOURCE__: specifies the ADF1 clock source. This parameter can be one of the following values:
  - RCC_ADF1CLKSOURCE_HCLK HCLK Clock selected as ADF1 clock
  - RCC_ADF1CLKSOURCE_CLKP CLKP selected as ADF1 clock
  - RCC_ADF1CLKSOURCE_IC7 IC7 selected as ADF1 clock
  - RCC_ADF1CLKSOURCE_IC8 IC8 selected as ADF1 clock
  - RCC_ADF1CLKSOURCE_MSI MSI selected as ADF1 clock
  - RCC_ADF1CLKSOURCE_HSI HSI selected as ADF1 clock
  - RCC_ADF1CLKSOURCE_PIN External I2S_CKIN selected as ADF1 clock
  - RCC_ADF1CLKSOURCE_TIMG TIMG selected as ADF1 clock

__HAL_RCC_GET_ADF1_SOURCE

Description:

- Macro to get the ADF1 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_ADF1CLKSOURCE_HCLK HCLK Clock selected as ADF1 clock
  - RCC_ADF1CLKSOURCE_CLKP CLKP selected as ADF1 clock
  - RCC_ADF1CLKSOURCE_IC7 IC7 selected as ADF1 clock
  - RCC_ADF1CLKSOURCE_IC8 IC8 selected as ADF1 clock
  - RCC_ADF1CLKSOURCE_MSI MSI selected as ADF1 clock
  - RCC_ADF1CLKSOURCE_HSI HSI selected as ADF1 clock
  - RCC_ADF1CLKSOURCE_PIN External I2S_CKIN selected as ADF1 clock
  - RCC_ADF1CLKSOURCE_TIMG TIMG selected as ADF1 clock

__HAL_RCC_CLKP_CONFIG

Description:

- Macro to configure the CLKP clock for peripheral.

Parameters:

- __CLKP_CLKSOURCE__: specifies the clock for peripheral This parameter can be one of the following values:
  - RCC_CLKPCLKSOURCE_HSI HSI selected as CLKP clock
  - RCC_CLKPCLKSOURCE_MSI MSI selected as CLKP clock
  - RCC_CLKPCLKSOURCE_HSE HSE selected as CLKP clock
  - RCC_CLKPCLKSOURCE_IC5 IC5 selected as CLKP clock
  - RCC_CLKPCLKSOURCE_IC10 IC10 selected as CLKP clock
  - RCC_CLKPCLKSOURCE_IC15 IC15 selected as CLKP clock
  - RCC_CLKPCLKSOURCE_IC19 IC19 selected as CLKP clock
  - RCC_CLKPCLKSOURCE_IC20 IC20 selected as CLKP clock

__HAL_RCC_GET_CLKP_SOURCE

Description:

- Macro to get the Oscillator clock for peripheral source.

Return value:

- The: clock source can be one of the following values:
  - RCC_CLKPCLKSOURCE_HSI HSI selected as CLKP clock
  - RCC_CLKPCLKSOURCE_MSI MSI selected as CLKP clock
  - RCC_CLKPCLKSOURCE_HSE HSE selected as CLKP clock
  - RCC_CLKPCLKSOURCE_IC5 IC5 selected as CLKP clock
  - RCC_CLKPCLKSOURCE_IC10 IC10 selected as CLKP clock
  - RCC_CLKPCLKSOURCE_IC15 IC15 selected as CLKP clock
  - RCC_CLKPCLKSOURCE_IC19 IC19 selected as CLKP clock
  - RCC_CLKPCLKSOURCE_IC20 IC20 selected as CLKP clock

__HAL_RCC_DCMIPP_CONFIG

Description:

- Macro to configure the DCMIPP clock source.

Parameters:

- __DCMIPP_CLKSOURCE__: specifies the DCMIPP clock source.
  - RCC_DCMIPPCLKSOURCE_PCLK5 PCLK5 selected as DCMIPP clock
  - RCC_DCMIPPCLKSOURCE_CLKP CLKP selected as DCMIPP clock
  - RCC_DCMIPPCLKSOURCE_IC17 IC17 selected as DCMIPP clock
  - RCC_DCMIPPCLKSOURCE_HSI HSI selected as DCMIPP clock

__HAL_RCC_GET_DCMIPP_SOURCE

Description:

- Macro to get the DCMIPP clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_DCMIPPCLKSOURCE_PCLK5 PCLK5 selected as DCMIPP clock
  - RCC_DCMIPPCLKSOURCE_CLKP CLKP selected as DCMIPP clock
  - RCC_DCMIPPCLKSOURCE_IC17 IC17 elected as DCMIPP clock
  - RCC_DCMIPPCLKSOURCE_HSI HSI selected as DCMIPP clock

__HAL_RCC_ETH1_CONFIG

Description:

- Macro to configure the ETH1 kernel clock source.

Parameters:

- __ETH1_CLKSOURCE__: specifies clock source for ETH1 This parameter can be one of the following values:
  - RCC_ETH1CLKSOURCE_HCLK HCLK selected as ETH1 clock
  - RCC_ETH1CLKSOURCE_CLKP CLKP selected as ETH1 clock
  - RCC_ETH1CLKSOURCE_IC12 IC12 selected as ETH1 clock
  - RCC_ETH1CLKSOURCE_HSE HSE selected as ETH1 clock

__HAL_RCC_GET_ETH1_SOURCE

Description:

- Macro to get the ETH1 kernel clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_ETH1CLKSOURCE_HCLK HCLK selected as ETH1 clock
  - RCC_ETH1CLKSOURCE_CLKP CLKP selected as ETH1 clock
  - RCC_ETH1CLKSOURCE_IC12 IC12 selected as ETH1 clock
  - RCC_ETH1CLKSOURCE_HSE HSE selected as ETH1 clock

__HAL_RCC_ETH1PHY_CONFIG

Description:

- Macro to configure the ETH1 PHY interface.

Parameters:

- __ETH1PHY_IF__: specifies interface for ETH1 PHY This parameter can be one of the following values:
  - RCC_ETH1PHYIF_MII MII selected as ETH1 PHY interface
  - RCC_ETH1PHYIF_RGMII RGMII selected as ETH1 PHY interface
  - RCC_ETH1PHYIF_RMII RMII selected as ETH1 PHY interface

Notes:

- This configuration must be done while ETH1 is under reset, and before enabling the ETH1 clocks

__HAL_RCC_GET_ETH1PHY_INTERFACE

Description:

- Macro to get the ETH PHY clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_ETH1PHYIF_MII MII selected as ETH1 PHY interface
  - RCC_ETH1PHYIF_RGMII RGMII selected as ETH1 PHY interface
  - RCC_ETH1PHYIF_RMII RMII selected as ETH1 PHY interface

__HAL_RCC_ETH1RX_CONFIG

Description:

- Macro to configure the ETH1 reference Rx clock source.

Parameters:

- __ETH1RX_CLKSOURCE__: specifies clock source for ETH1 reference Rx This parameter can be one of the following values:
  - RCC_ETH1RXCLKSOURCE_EXT External clock selected as ETH1 reference Rx clock
  - RCC_ETH1RXCLKSOURCE_INT Internal kernel selected as ETH1 reference Rx clock

__HAL_RCC_GET_ETH1RX_SOURCE

Description:

- Macro to get the ETH1 reference Rx clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_ETH1RXCLKSOURCE_EXT External clock selected as ETH1 reference Rx clock
  - RCC_ETH1RXCLKSOURCE_INT Internal kernel selected as ETH1 reference Rx clock

__HAL_RCC_ETH1TX_CONFIG

Description:

- Macro to configure the ETH1 reference Tx RGMII 125MHz clock source.

Parameters:

- __ETH1TX_CLKSOURCE__: specifies clock source for ETH1 reference Tx This parameter can be one of the following values:
  - RCC_ETH1TXCLKSOURCE_EXT External clock selected as ETH1 reference Tx clock
  - RCC_ETH1TXCLKSOURCE_INT Internal kernel selected as ETH1 reference Tx clock

__HAL_RCC_GET_ETH1TX_SOURCE

Description:

- Macro to get the ETH1 reference Tx RGMII 125MHz clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_ETH1TXCLKSOURCE_EXT External clock selected as ETH1 reference Tx clock
  - RCC_ETH1TXCLKSOURCE_INT Internal kernel selected as ETH1 reference Tx clock

__HAL_RCC_ETH1PTP_CONFIG

Description:

- Macro to configure the ETH1 PTP clock source.

Parameters:

- __ETH1PTP_CLKSOURCE__: specifies clock source for ETH1 PTP This parameter can be one of the following values:
  - RCC_ETH1PTPCLKSOURCE_HCLK HCLK selected as ETH1 PTP clock
  - RCC_ETH1PTPCLKSOURCE_CLKP CLKP selected as ETH1 PTP clock
  - RCC_ETH1PTPCLKSOURCE_IC13 IC13 selected as ETH1 PTP clock
  - RCC_ETH1PTPCLKSOURCE_HSE HSE selected as ETH1 PTP clock

__HAL_RCC_GET_ETH1PTP_SOURCE

Description:

- Macro to get the ETH1 PTP clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_ETH1PTPCLKSOURCE_HCLK HCLK selected as ETH1 PTP clock
  - RCC_ETH1PTPCLKSOURCE_CLKP CLKP selected as ETH1 PTP clock
  - RCC_ETH1PTPCLKSOURCE_IC13 IC13 selected as ETH1 PTP clock
  - RCC_ETH1PTPCLKSOURCE_HSE HSE selected as ETH1 PTP clock

__HAL_RCC_ETH1PTP_DIVIDER_CONFIG

Description:

- Macro to configure the ETH1 PTP divider.

Parameters:

- __ETH1PTP_DIVIDER__: specifies clock divider for ETH1 PTP This parameter can be a value between 1 and 16.

__HAL_RCC_GET_ETH1PTP_DIVIDER

Description:

- Macro to get the ETH1 PTP divider.

Return value:

- The: divider can be a value between 1 and 16.

__HAL_RCC_FDCAN_CONFIG

Description:

- Macro to configure the FDCAN kernel clock source.

Parameters:

- __FDCAN_CLKSOURCE__: specifies clock source for FDCAN kernel This parameter can be one of the following values:
  - RCC_FDCANCLKSOURCE_PCLK1 PCLK1 selected as FDCAN kernel clock (default)
  - RCC_FDCANCLKSOURCE_CLKP CLKP selected as FDCAN kernel clock
  - RCC_FDCANCLKSOURCE_IC19 IC19 selected as FDCAN kernel clock
  - RCC_FDCANCLKSOURCE_HSE HSE selected as FDCAN kernel clock

__HAL_RCC_GET_FDCAN_SOURCE

Description:

- Macro to get the FDCAN kernel clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_FDCANCLKSOURCE_PCLK1 PCLK1 selected as FDCAN kernel clock (default)
  - RCC_FDCANCLKSOURCE_CLKP CLKP selected as FDCAN kernel clock
  - RCC_FDCANCLKSOURCE_IC19 IC19 selected as FDCAN kernel clock
  - RCC_FDCANCLKSOURCE_HSE HSE selected as FDCAN kernel clock

__HAL_RCC_FMC_CONFIG

Description:

- Macro to configure the FMC clock source.

Parameters:

- __FMC_CLKSOURCE__: specifies the FMC clock source.
  - RCC_FMCCLKSOURCE_HCLK HCLK Clock selected as FMC clock
  - RCC_FMCCLKSOURCE_CLKP CLKP Clock selected as FMC clock
  - RCC_FMCCLKSOURCE_IC3 IC3 Clock selected as FMC clock
  - RCC_FMCCLKSOURCE_IC4 IC4 selected as FMC clock

__HAL_RCC_GET_FMC_SOURCE

Description:

- Macro to get the FMC clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_FMCCLKSOURCE_HCLK HCLK Clock selected as FMC clock
  - RCC_FMCCLKSOURCE_CLKP CLKP Clock selected as FMC clock
  - RCC_FMCCLKSOURCE_IC3 IC3 Clock selected as FMC clock
  - RCC_FMCCLKSOURCE_IC4 IC4 selected as FMC clock

__HAL_RCC_I2C1_CONFIG

Description:

- Macro to configure the I2C1 clock source.

Parameters:

- __I2C1_CLKSOURCE__: specifies the I2C1 clock source. This parameter can be one of the following values:
  - RCC_I2C1CLKSOURCE_PCLK1 PCLK1 selected as I2C1 clock
  - RCC_I2C1CLKSOURCE_CLKP CKLP selected as I2C1 clock
  - RCC_I2C1CLKSOURCE_IC10 IC10 selected as I2C1 clock
  - RCC_I2C1CLKSOURCE_IC15 IC15 selected as I2C1 clock
  - RCC_I2C1CLKSOURCE_MSI MSI selected as I2C1 clock
  - RCC_I2C1CLKSOURCE_HSI HSI selected as I2C1 clock

__HAL_RCC_GET_I2C1_SOURCE

Description:

- Macro to get the I2C1 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_I2C1CLKSOURCE_PCLK1 PCLK1 selected as I2C1 clock
  - RCC_I2C1CLKSOURCE_CLKP CKLP selected as I2C1 clock
  - RCC_I2C1CLKSOURCE_IC10 IC10 selected as I2C1 clock
  - RCC_I2C1CLKSOURCE_IC15 IC15 selected as I2C1 clock
  - RCC_I2C1CLKSOURCE_MSI MSI selected as I2C1 clock
  - RCC_I2C1CLKSOURCE_HSI HSI selected as I2C1 clock

__HAL_RCC_I2C2_CONFIG

Description:

- Macro to configure the I2C2 clock source.

Parameters:

- __I2C2_CLKSOURCE__: specifies the I2C2 clock source. This parameter can be one of the following values:
  - RCC_I2C2CLKSOURCE_PCLK1 PCLK1 selected as I2C2 clock
  - RCC_I2C2CLKSOURCE_CLKP CKLP selected as I2C2 clock
  - RCC_I2C2CLKSOURCE_IC10 IC10 selected as I2C2 clock
  - RCC_I2C2CLKSOURCE_IC15 IC15 selected as I2C2 clock
  - RCC_I2C2CLKSOURCE_MSI MSI selected as I2C2 clock
  - RCC_I2C2CLKSOURCE_HSI HSI selected as I2C2 clock

__HAL_RCC_GET_I2C2_SOURCE

Description:

- Macro to get the I2C2 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_I2C2CLKSOURCE_PCLK1 PCLK1 selected as I2C2 clock
  - RCC_I2C2CLKSOURCE_CLKP CKLP selected as I2C2 clock
  - RCC_I2C2CLKSOURCE_IC10 IC10 selected as I2C2 clock
  - RCC_I2C2CLKSOURCE_IC15 IC15 selected as I2C2 clock
  - RCC_I2C2CLKSOURCE_MSI MSI selected as I2C2 clock
  - RCC_I2C2CLKSOURCE_HSI HSI selected as I2C2 clock

__HAL_RCC_I2C3_CONFIG

Description:

- Macro to configure the I2C3 clock source.

Parameters:

- __I2C3_CLKSOURCE__: specifies the I2C3 clock source. This parameter can be one of the following values:
  - RCC_I2C3CLKSOURCE_PCLK1 PCLK1 selected as I2C3 clock
  - RCC_I2C3CLKSOURCE_CLKP CKLP selected as I2C3 clock
  - RCC_I2C3CLKSOURCE_IC10 IC10 selected as I2C3 clock
  - RCC_I2C3CLKSOURCE_IC15 IC15 selected as I2C3 clock
  - RCC_I2C3CLKSOURCE_MSI MSI selected as I2C3 clock
  - RCC_I2C3CLKSOURCE_HSI HSI selected as I2C3 clock

__HAL_RCC_GET_I2C3_SOURCE

Description:

- Macro to get the I2C3 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_I2C3CLKSOURCE_PCLK1 PCLK1 selected as I2C3 clock
  - RCC_I2C3CLKSOURCE_CLKP CKLP selected as I2C3 clock
  - RCC_I2C3CLKSOURCE_IC10 IC10 selected as I2C3 clock
  - RCC_I2C3CLKSOURCE_IC15 IC15 selected as I2C3 clock
  - RCC_I2C3CLKSOURCE_MSI MSI selected as I2C3 clock
  - RCC_I2C3CLKSOURCE_HSI HSI selected as I2C3 clock

__HAL_RCC_I2C4_CONFIG

Description:

- Macro to configure the I2C4 clock source.

Parameters:

- __I2C4_CLKSOURCE__: specifies the I2C4 clock source. This parameter can be one of the following values:
  - RCC_I2C4CLKSOURCE_PCLK1 PCLK1 selected as I2C4 clock
  - RCC_I2C4CLKSOURCE_CLKP CKLP selected as I2C4 clock
  - RCC_I2C4CLKSOURCE_IC10 IC10 selected as I2C4 clock
  - RCC_I2C4CLKSOURCE_IC15 IC15 selected as I2C4 clock
  - RCC_I2C4CLKSOURCE_MSI MSI selected as I2C4 clock
  - RCC_I2C4CLKSOURCE_HSI HSI selected as I2C4 clock

__HAL_RCC_GET_I2C4_SOURCE

Description:

- Macro to get the I2C4 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_I2C4CLKSOURCE_PCLK1 PCLK1 selected as I2C4 clock
  - RCC_I2C4CLKSOURCE_CLKP CKLP selected as I2C4 clock
  - RCC_I2C4CLKSOURCE_IC10 IC10 selected as I2C4 clock
  - RCC_I2C4CLKSOURCE_IC15 IC15 selected as I2C4 clock
  - RCC_I2C4CLKSOURCE_MSI MSI selected as I2C4 clock
  - RCC_I2C4CLKSOURCE_HSI HSI selected as I2C4 clock

__HAL_RCC_I3C1_CONFIG

Description:

- Macro to configure the I3C1 clock source.

Parameters:

- __I3C1_CLKSOURCE__: specifies the I3C1 clock source. This parameter can be one of the following values:
  - RCC_I3C1CLKSOURCE_PCLK1 PCLK1 selected as I3C1 clock
  - RCC_I3C1CLKSOURCE_CLKP CKLP selected as I3C1 clock
  - RCC_I3C1CLKSOURCE_IC10 IC10 selected as I3C1 clock
  - RCC_I3C1CLKSOURCE_IC15 IC15 selected as I3C1 clock
  - RCC_I3C1CLKSOURCE_MSI MSI selected as I3C1 clock
  - RCC_I3C1CLKSOURCE_HSI HSI selected as I3C1 clock

__HAL_RCC_GET_I3C1_SOURCE

Description:

- Macro to get the I3C1 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_I3C1CLKSOURCE_PCLK1 PCLK1 selected as I3C1 clock
  - RCC_I3C1CLKSOURCE_CLKP CKLP selected as I3C1 clock
  - RCC_I3C1CLKSOURCE_IC10 IC10 selected as I3C1 clock
  - RCC_I3C1CLKSOURCE_IC15 IC15 selected as I3C1 clock
  - RCC_I3C1CLKSOURCE_MSI MSI selected as I3C1 clock
  - RCC_I3C1CLKSOURCE_HSI HSI selected as I3C1 clock

__HAL_RCC_I3C2_CONFIG

Description:

- Macro to configure the I3C2 clock source.

Parameters:

- __I3C2_CLKSOURCE__: specifies the I3C2 clock source. This parameter can be one of the following values:
  - RCC_I3C2CLKSOURCE_PCLK1 PCLK1 selected as I3C2 clock
  - RCC_I3C2CLKSOURCE_CLKP CKLP selected as I3C2 clock
  - RCC_I3C2CLKSOURCE_IC10 IC10 selected as I3C2 clock
  - RCC_I3C2CLKSOURCE_IC15 IC15 selected as I3C2 clock
  - RCC_I3C2CLKSOURCE_MSI MSI selected as I3C2 clock
  - RCC_I3C2CLKSOURCE_HSI HSI selected as I3C2 clock

__HAL_RCC_GET_I3C2_SOURCE

Description:

- Macro to get the I3C2 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_I3C2CLKSOURCE_PCLK1 PCLK1 selected as I3C2 clock
  - RCC_I3C2CLKSOURCE_CLKP CKLP selected as I3C2 clock
  - RCC_I3C2CLKSOURCE_IC10 IC10 selected as I3C2 clock
  - RCC_I3C2CLKSOURCE_IC15 IC15 selected as I3C2 clock
  - RCC_I3C2CLKSOURCE_MSI MSI selected as I3C2 clock
  - RCC_I3C2CLKSOURCE_HSI HSI selected as I3C2 clock

__HAL_RCC_LPTIM1_CONFIG

Description:

- Macro to configure the LPTIM1 clock source.

Parameters:

- __LPTIM1_CLKSOURCE__: specifies the LPTIM1 clock source. This parameter can be one of the following values:
  - RCC_LPTIM1CLKSOURCE_PCLK1 PCLK1 clock selected as LPTIM1 clock
  - RCC_LPTIM1CLKSOURCE_CLKP CLKP clock selected as LPTIM1 clock
  - RCC_LPTIM1CLKSOURCE_IC15 IC15 clock selected as LPTIM1 clock
  - RCC_LPTIM1CLKSOURCE_LSE LSE clock selected as LPTIM1 clock
  - RCC_LPTIM1CLKSOURCE_LSI LSI clock selected as LPTIM1 clock
  - RCC_LPTIM1CLKSOURCE_TIMG TIMG clock selected as LPTIM1 clock

__HAL_RCC_GET_LPTIM1_SOURCE

Description:

- Macro to get the LPTIM1 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_LPTIM1CLKSOURCE_PCLK1 PCLK1 clock selected as LPTIM1 clock
  - RCC_LPTIM1CLKSOURCE_CLKP CLKP clock selected as LPTIM1 clock
  - RCC_LPTIM1CLKSOURCE_IC15 IC15 clock selected as LPTIM1 clock
  - RCC_LPTIM1CLKSOURCE_LSE LSE clock selected as LPTIM1 clock
  - RCC_LPTIM1CLKSOURCE_LSI LSI clock selected as LPTIM1 clock
  - RCC_LPTIM1CLKSOURCE_TIMG TIMG clock selected as LPTIM1 clock

__HAL_RCC_LPTIM2_CONFIG

Description:

- Macro to configure the LPTIM2 clock source.

Parameters:

- __LPTIM2_CLKSOURCE__: specifies the LPTIM2 clock source. This parameter can be one of the following values:
  - RCC_LPTIM2CLKSOURCE_PCLK4 PCLK4 clock selected as LPTIM2 clock
  - RCC_LPTIM2CLKSOURCE_CLKP CLKP clock selected as LPTIM2 clock
  - RCC_LPTIM2CLKSOURCE_IC15 IC15 clock selected as LPTIM2 clock
  - RCC_LPTIM2CLKSOURCE_LSE LSE clock selected as LPTIM2 clock
  - RCC_LPTIM2CLKSOURCE_LSI LSI clock selected as LPTIM2 clock
  - RCC_LPTIM2CLKSOURCE_TIMG TIMG clock selected as LPTIM2 clock

__HAL_RCC_GET_LPTIM2_SOURCE

Description:

- Macro to get the LPTIM2 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_LPTIM2CLKSOURCE_PCLK4 PCLK4 clock selected as LPTIM2 clock
  - RCC_LPTIM2CLKSOURCE_CLKP CLKP clock selected as LPTIM2 clock
  - RCC_LPTIM2CLKSOURCE_IC15 IC15 clock selected as LPTIM2 clock
  - RCC_LPTIM2CLKSOURCE_LSE LSE clock selected as LPTIM2 clock
  - RCC_LPTIM2CLKSOURCE_LSI LSI clock selected as LPTIM2 clock
  - RCC_LPTIM2CLKSOURCE_TIMG TIMG clock selected as LPTIM2 clock

__HAL_RCC_LPTIM3_CONFIG

Description:

- Macro to configure the LPTIM3 clock source.

Parameters:

- __LPTIM3_CLKSOURCE__: specifies the LPTIM3 clock source. This parameter can be one of the following values:
  - RCC_LPTIM3CLKSOURCE_PCLK4 PCLK4 clock selected as LPTIM3 clock
  - RCC_LPTIM3CLKSOURCE_CLKP CLKP clock selected as LPTIM3 clock
  - RCC_LPTIM3CLKSOURCE_IC15 IC15 clock selected as LPTIM3 clock
  - RCC_LPTIM3CLKSOURCE_LSE LSE clock selected as LPTIM3 clock
  - RCC_LPTIM3CLKSOURCE_LSI LSI clock selected as LPTIM3 clock
  - RCC_LPTIM3CLKSOURCE_TIMG TIMG clock selected as LPTIM3 clock

__HAL_RCC_GET_LPTIM3_SOURCE

Description:

- Macro to get the LPTIM3 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_LPTIM3CLKSOURCE_PCLK4 PCLK4 clock selected as LPTIM3 clock
  - RCC_LPTIM3CLKSOURCE_CLKP CLKP clock selected as LPTIM3 clock
  - RCC_LPTIM3CLKSOURCE_IC15 IC15 clock selected as LPTIM3 clock
  - RCC_LPTIM3CLKSOURCE_LSE LSE clock selected as LPTIM3 clock
  - RCC_LPTIM3CLKSOURCE_LSI LSI clock selected as LPTIM3 clock
  - RCC_LPTIM3CLKSOURCE_TIMG TIMG clock selected as LPTIM3 clock

__HAL_RCC_LPTIM4_CONFIG

Description:

- Macro to configure the LPTIM4 clock source.

Parameters:

- __LPTIM4_CLKSOURCE__: specifies the LPTIM4 clock source. This parameter can be one of the following values:
  - RCC_LPTIM4CLKSOURCE_PCLK4 PCLK4 clock selected as LPTIM4 clock
  - RCC_LPTIM4CLKSOURCE_CLKP CLKP clock selected as LPTIM4 clock
  - RCC_LPTIM4CLKSOURCE_IC15 IC15 clock selected as LPTIM4 clock
  - RCC_LPTIM4CLKSOURCE_LSE LSE clock selected as LPTIM4 clock
  - RCC_LPTIM4CLKSOURCE_LSI LSI clock selected as LPTIM4 clock
  - RCC_LPTIM4CLKSOURCE_TIMG TIMG clock selected as LPTIM4 clock

__HAL_RCC_GET_LPTIM4_SOURCE

Description:

- Macro to get the LPTIM4 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_LPTIM4CLKSOURCE_PCLK4 PCLK4 clock selected as LPTIM4 clock
  - RCC_LPTIM4CLKSOURCE_CLKP CLKP clock selected as LPTIM4 clock
  - RCC_LPTIM4CLKSOURCE_IC15 IC15 clock selected as LPTIM4 clock
  - RCC_LPTIM4CLKSOURCE_LSE LSE clock selected as LPTIM4 clock
  - RCC_LPTIM4CLKSOURCE_LSI LSI clock selected as LPTIM4 clock
  - RCC_LPTIM4CLKSOURCE_TIMG TIMG clock selected as LPTIM4 clock

__HAL_RCC_LPTIM5_CONFIG

Description:

- Macro to configure the LPTIM5 clock source.

Parameters:

- __LPTIM5_CLKSOURCE__: specifies the LPTIM5 clock source. This parameter can be one of the following values:
  - RCC_LPTIM5CLKSOURCE_PCLK4 PCLK4 clock selected as LPTIM5 clock
  - RCC_LPTIM5CLKSOURCE_CLKP CLKP clock selected as LPTIM5 clock
  - RCC_LPTIM5CLKSOURCE_IC15 IC15 clock selected as LPTIM5 clock
  - RCC_LPTIM5CLKSOURCE_LSE LSE clock selected as LPTIM5 clock
  - RCC_LPTIM5CLKSOURCE_LSI LSI clock selected as LPTIM5 clock
  - RCC_LPTIM5CLKSOURCE_TIMG TIMG clock selected as LPTIM5 clock

__HAL_RCC_GET_LPTIM5_SOURCE

Description:

- Macro to get the LPTIM5 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_LPTIM5CLKSOURCE_PCLK4 PCLK4 clock selected as LPTIM5 clock
  - RCC_LPTIM5CLKSOURCE_CLKP CLKP clock selected as LPTIM5 clock
  - RCC_LPTIM5CLKSOURCE_IC15 IC15 clock selected as LPTIM5 clock
  - RCC_LPTIM5CLKSOURCE_LSE LSE clock selected as LPTIM5 clock
  - RCC_LPTIM5CLKSOURCE_LSI LSI clock selected as LPTIM5 clock
  - RCC_LPTIM5CLKSOURCE_TIMG TIMG clock selected as LPTIM5 clock

__HAL_RCC_LPUART1_CONFIG

Description:

- Macro to configure the LPUART1 clock (LPUART1CLK).

Parameters:

- __LPUART1_CLKSOURCE__: specifies the LPUART1 clock source. This parameter can be one of the following values:
  - RCC_LPUART1CLKSOURCE_PCLK4 PCLK4 Clock selected as LPUART1 clock
  - RCC_LPUART1CLKSOURCE_CLKP CLKP selected as LPUART1 clock
  - RCC_LPUART1CLKSOURCE_IC9 IC9 selected as LPUART1 clock
  - RCC_LPUART1CLKSOURCE_IC14 IC14 selected as LPUART1 clock
  - RCC_LPUART1CLKSOURCE_LSE LSE selected as LPUART1 clock
  - RCC_LPUART1CLKSOURCE_MSI MSI selected as LPUART1 clock
  - RCC_LPUART1CLKSOURCE_HSI HSI selected as LPUART1 clock

__HAL_RCC_GET_LPUART1_SOURCE

Description:

- Macro to get the LPUART1 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_LPUART1CLKSOURCE_PCLK4 PCLK4 Clock selected as LPUART1 clock
  - RCC_LPUART1CLKSOURCE_CLKP CLKP selected as LPUART1 clock
  - RCC_LPUART1CLKSOURCE_IC9 IC9 selected as LPUART1 clock
  - RCC_LPUART1CLKSOURCE_IC14 IC14 selected as LPUART1 clock
  - RCC_LPUART1CLKSOURCE_LSE LSE selected as LPUART1 clock
  - RCC_LPUART1CLKSOURCE_MSI MSI selected as LPUART1 clock
  - RCC_LPUART1CLKSOURCE_HSI HSI selected as LPUART1 clock

__HAL_RCC_LTDC_CONFIG

Description:

- Macro to configure the LTDC clock (LTDCCLK).

Parameters:

- __LTDC_CLKSOURCE__: specifies the LTDC clock source. This parameter can be one of the following values:
  - RCC_LTDCCLKSOURCE_PCLK5 PCLK5 Clock selected as LTDC clock
  - RCC_LTDCCLKSOURCE_CLKP CLKP selected as LTDC clock
  - RCC_LTDCCLKSOURCE_IC16 IC14 selected as LTDC clock
  - RCC_LTDCCLKSOURCE_HSI HSI selected as LTDC clock

__HAL_RCC_GET_LTDC_SOURCE

Description:

- Macro to get the LTDC clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_LTDCCLKSOURCE_PCLK5 PCLK5 Clock selected as LTDC clock
  - RCC_LTDCCLKSOURCE_CLKP CLKP selected as LTDC clock
  - RCC_LTDCCLKSOURCE_IC16 IC14 selected as LTDC clock
  - RCC_LTDCCLKSOURCE_HSI HSI selected as LTDC clock

__HAL_RCC_MDF1_CONFIG

Description:

- Macro to configure the MDF1 clock.

Parameters:

- __MDF1_CLKSOURCE__: specifies the MDF1 clock source. This parameter can be one of the following values:
  - RCC_MDF1CLKSOURCE_HCLK HCLK Clock selected as MDF1 clock
  - RCC_MDF1CLKSOURCE_CLKP CLKP selected as MDF1 clock
  - RCC_MDF1CLKSOURCE_IC7 IC7 selected as MDF1 clock
  - RCC_MDF1CLKSOURCE_IC8 IC8 selected as MDF1 clock
  - RCC_MDF1CLKSOURCE_MSI MSI selected as MDF1 clock
  - RCC_MDF1CLKSOURCE_HSI HSI selected as MDF1 clock
  - RCC_MDF1CLKSOURCE_PIN External I2S_CKIN selected as MDF1 clock
  - RCC_MDF1CLKSOURCE_TIMG TIMG selected as MDF1 clock

__HAL_RCC_GET_MDF1_SOURCE

Description:

- Macro to get the MDF1 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_MDF1CLKSOURCE_HCLK HCLK Clock selected as MDF1 clock
  - RCC_MDF1CLKSOURCE_CLKP CLKP selected as MDF1 clock
  - RCC_MDF1CLKSOURCE_IC7 IC7 selected as MDF1 clock
  - RCC_MDF1CLKSOURCE_IC8 IC8 selected as MDF1 clock
  - RCC_MDF1CLKSOURCE_MSI MSI selected as MDF1 clock
  - RCC_MDF1CLKSOURCE_HSI HSI selected as MDF1 clock
  - RCC_MDF1CLKSOURCE_PIN External I2S_CKIN selected as MDF1 clock
  - RCC_MDF1CLKSOURCE_TIMG TIMG selected as MDF1 clock

__HAL_RCC_PSSI_CONFIG

Description:

- Macro to configure the PSSI clock source.

Parameters:

- __PSSI_CLKSOURCE__: defines the PSSI clock source. This parameter can be one of the following values:
  - RCC_PSSICLKSOURCE_HCLK HCLK selected as PSSI clock
  - RCC_PSSICLKSOURCE_CLKP Peripheral clock CLKP selected as PSSI clock
  - RCC_PSSICLKSOURCE_IC20 IC20 selected as PSSI clock
  - RCC_PSSICLKSOURCE_HSI HSI selected as PSSI clock

Return value:

- None

__HAL_RCC_GET_PSSI_SOURCE

Description:

- Macro to get the PSSI clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_PSSICLKSOURCE_HCLK HCLK selected as PSSI clock
  - RCC_PSSICLKSOURCE_CLKP Peripheral clock CLKP selected as PSSI clock
  - RCC_PSSICLKSOURCE_IC20 IC20 selected as PSSI clock
  - RCC_PSSICLKSOURCE_HSI HSI selected as PSSI clock

__HAL_RCC_SAI1_CONFIG

Description:

- Macro to configure the SAI1 clock source.

Parameters:

- __SAI1_CLKSOURCE__: defines the SAI1 clock source. This parameter can be one of the following values:
  - RCC_SAI1CLKSOURCE_PCLK2 PCLK2 Clock selected as SAI1 clock
  - RCC_SAI1CLKSOURCE_CLKP Peripheral clock CLKP selected as SAI1 clock
  - RCC_SAI1CLKSOURCE_IC7 IC7 selected as SAI1 clock
  - RCC_SAI1CLKSOURCE_IC8 IC8 selected as SAI1 clock
  - RCC_SAI1CLKSOURCE_MSI MSI selected as SAI1 clock
  - RCC_SAI1CLKSOURCE_HSI HSI selected as SAI1 clock
  - RCC_SAI1CLKSOURCE_PIN External I2S_CKIN selected as SAI1 clock
  - RCC_SAI1CLKSOURCE_SPDIFRX1 SPDIFRX1 selected as SAI1 clock

Return value:

- None

__HAL_RCC_GET_SAI1_SOURCE

Description:

- Macro to get the SAI1 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_SAI1CLKSOURCE_PCLK2 PCLK2 Clock selected as SAI1 clock
  - RCC_SAI1CLKSOURCE_CLKP Peripheral clock CLKP selected as SAI1 clock
  - RCC_SAI1CLKSOURCE_IC7 IC7 selected as SAI1 clock
  - RCC_SAI1CLKSOURCE_IC8 IC8 selected as SAI1 clock
  - RCC_SAI1CLKSOURCE_MSI MSI selected as SAI1 clock
  - RCC_SAI1CLKSOURCE_HSI HSI selected as SAI1 clock
  - RCC_SAI1CLKSOURCE_PIN External I2S_CKIN selected as SAI1 clock
  - RCC_SAI1CLKSOURCE_SPDIFRX1 SPDIFRX1 selected as SAI1 clock

__HAL_RCC_SAI2_CONFIG

Description:

- Macro to configure the SAI2 clock source.

Parameters:

- __SAI2_CLKSOURCE__: defines the SAI2 clock source. This parameter can be one of the following values:
  - RCC_SAI2CLKSOURCE_PCLK2 PCLK2 Clock selected as SAI2 clock
  - RCC_SAI2CLKSOURCE_CLKP Peripheral clock CLKP selected as SAI2 clock
  - RCC_SAI2CLKSOURCE_IC7 IC7 selected as SAI2 clock
  - RCC_SAI2CLKSOURCE_IC8 IC8 selected as SAI2 clock
  - RCC_SAI2CLKSOURCE_MSI MSI selected as SAI2 clock
  - RCC_SAI2CLKSOURCE_HSI HSI selected as SAI2 clock
  - RCC_SAI2CLKSOURCE_PIN External I2S_CKIN selected as SAI2 clock
  - RCC_SAI2CLKSOURCE_SPDIFRX1 SPDIFRX1 selected as SAI2 clock

Return value:

- None

__HAL_RCC_GET_SAI2_SOURCE

Description:

- Macro to get the SAI2 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_SAI2CLKSOURCE_PCLK2 PCLK2 Clock selected as SAI2 clock
  - RCC_SAI2CLKSOURCE_CLKP Peripheral clock CLKP selected as SAI2 clock
  - RCC_SAI2CLKSOURCE_IC7 IC7 selected as SAI2 clock
  - RCC_SAI2CLKSOURCE_IC8 IC8 selected as SAI2 clock
  - RCC_SAI2CLKSOURCE_MSI MSI selected as SAI2 clock
  - RCC_SAI2CLKSOURCE_HSI HSI selected as SAI2 clock
  - RCC_SAI2CLKSOURCE_PIN External I2S_CKIN selected as SAI2 clock
  - RCC_SAI2CLKSOURCE_SPDIFRX1 SPDIFRX1 selected as SAI2 clock

__HAL_RCC_SDMMC1_CONFIG

Description:

- Macro to configure the SDMMC1 kernel clock source.

Parameters:

- __SDMMC1_CLKSOURCE__: specifies clock source for SDMMC1 This parameter can be one of the following values:
  - RCC_SDMMC1CLKSOURCE_HCLK HCLK Clock selected as SDMMC1 kernel clock
  - RCC_SDMMC1CLKSOURCE_CLKP Peripheral clock CLKP selected as SDMMC1 kernel clock
  - RCC_SDMMC1CLKSOURCE_IC4 IC4 selected as SDMMC1 kernel clock
  - RCC_SDMMC1CLKSOURCE_IC5 IC5 selected as SDMMC1 kernel clock

__HAL_RCC_GET_SDMMC1_SOURCE

Description:

- Macro to get the SDMMC1 kernel clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_SDMMC1CLKSOURCE_HCLK HCLK Clock selected as SDMMC1 kernel clock
  - RCC_SDMMC1CLKSOURCE_CLKP Peripheral clock CLKP selected as SDMMC1 kernel clock
  - RCC_SDMMC1CLKSOURCE_IC4 IC4 selected as SDMMC1 kernel clock
  - RCC_SDMMC1CLKSOURCE_IC5 IC5 selected as SDMMC1 kernel clock

__HAL_RCC_SDMMC2_CONFIG

Description:

- Macro to configure the SDMMC2 kernel clock source.

Parameters:

- __SDMMC2_CLKSOURCE__: specifies clock source for SDMMC2 This parameter can be one of the following values:
  - RCC_SDMMC2CLKSOURCE_HCLK HCLK Clock selected as SDMMC2 kernel clock
  - RCC_SDMMC2CLKSOURCE_CLKP Peripheral clock CLKP selected as SDMMC2 kernel clock
  - RCC_SDMMC2CLKSOURCE_IC4 IC4 selected as SDMMC2 kernel clock
  - RCC_SDMMC2CLKSOURCE_IC5 IC5 selected as SDMMC2 kernel clock

__HAL_RCC_GET_SDMMC2_SOURCE

Description:

- Macro to get the SDMMC2 kernel clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_SDMMC2CLKSOURCE_HCLK HCLK Clock selected as SDMMC2 kernel clock
  - RCC_SDMMC2CLKSOURCE_CLKP Peripheral clock CLKP selected as SDMMC2 kernel clock
  - RCC_SDMMC2CLKSOURCE_IC4 IC4 selected as SDMMC2 kernel clock
  - RCC_SDMMC2CLKSOURCE_IC5 IC5 selected as SDMMC2 kernel clock

__HAL_RCC_SPDIFRX1_CONFIG

Description:

- Macro to Configure the SPDIFRX1 clock source.

Parameters:

- __SPDIFRX_CLKSOURCE__: defines the SPDIFRX1 clock source. This parameter can be one of the following values:
  - RCC_SPDIFRX1CLKSOURCE_PCLK1 PCLK1 Clock selected as SPDIFRX1 clock
  - RCC_SPDIFRX1CLKSOURCE_CLKP Peripheral clock CLKP selected as SPDIFRX1 clock
  - RCC_SPDIFRX1CLKSOURCE_IC7 IC7 selected as SPDIFRX1 clock
  - RCC_SPDIFRX1CLKSOURCE_IC8 IC8 selected as SPDIFRX1 clock
  - RCC_SPDIFRX1CLKSOURCE_MSI MSI selected as SPDIFRX1 clock
  - RCC_SPDIFRX1CLKSOURCE_HSI HSI selected as SPDIFRX1 clock
  - RCC_SPDIFRX1CLKSOURCE_PIN External I2S_CKIN selected as SPDIFRX1 clock

Return value:

- None

__HAL_RCC_GET_SPDIFRX1_SOURCE

Description:

- Macro to get the SPDIFRX1 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_SPDIFRX1CLKSOURCE_PCLK1 PCLK1 Clock selected as SPDIFRX1 clock
  - RCC_SPDIFRX1CLKSOURCE_CLKP Peripheral clock CLKP selected as SPDIFRX1 clock
  - RCC_SPDIFRX1CLKSOURCE_IC7 IC7 selected as SPDIFRX1 clock
  - RCC_SPDIFRX1CLKSOURCE_IC8 IC8 selected as SPDIFRX1 clock
  - RCC_SPDIFRX1CLKSOURCE_MSI MSI selected as SPDIFRX1 clock
  - RCC_SPDIFRX1CLKSOURCE_HSI HSI selected as SPDIFRX1 clock
  - RCC_SPDIFRX1CLKSOURCE_PIN External I2S_CKIN selected as SPDIFRX1 clock
- None

__HAL_RCC_SPI1_CONFIG

Description:

- Macro to configure the SPI1 clock source.

Parameters:

- __SPI1_CLKSOURCE__: defines the SPI1 clock source. This parameter can be one of the following values:
  - RCC_SPI1CLKSOURCE_PCLK2 PCLK2 selected as SPI1 clock
  - RCC_SPI1CLKSOURCE_CLKP Peripheral clock CKLP selected as SPI1 clock
  - RCC_SPI1CLKSOURCE_IC8 IC8 selected as SPI1 clock
  - RCC_SPI1CLKSOURCE_IC9 IC9 selected as SPI1 clock
  - RCC_SPI1CLKSOURCE_MSI MSI selected as SPI1 clock
  - RCC_SPI1CLKSOURCE_HSI HSI selected as SPI1 clock
  - RCC_SPI1CLKSOURCE_PIN External I2S_CKIN selected as SPI1 clock

Return value:

- None

__HAL_RCC_GET_SPI1_SOURCE

Description:

- Macro to get the SPI1 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_SPI1CLKSOURCE_PCLK2 PCLK2 selected as SPI1 clock
  - RCC_SPI1CLKSOURCE_CLKP Peripheral clock CKLP selected as SPI1 clock
  - RCC_SPI1CLKSOURCE_IC8 IC8 selected as SPI1 clock
  - RCC_SPI1CLKSOURCE_IC9 IC9 selected as SPI1 clock
  - RCC_SPI1CLKSOURCE_MSI MSI selected as SPI1 clock
  - RCC_SPI1CLKSOURCE_HSI HSI selected as SPI1 clock
  - RCC_SPI1CLKSOURCE_PIN External I2S_CKIN selected as SPI1 clock

__HAL_RCC_SPI2_CONFIG

Description:

- Macro to configure the SPI2 clock source.

Parameters:

- __SPI2_CLKSOURCE__: defines the SPI2 clock source. This parameter can be one of the following values:
  - RCC_SPI2CLKSOURCE_PCLK1 PCLK1 selected as SPI2 clock
  - RCC_SPI2CLKSOURCE_CLKP Peripheral clock CKLP selected as SPI2 clock
  - RCC_SPI2CLKSOURCE_IC8 IC8 selected as SPI2 clock
  - RCC_SPI2CLKSOURCE_IC9 IC9 selected as SPI2 clock
  - RCC_SPI2CLKSOURCE_MSI MSI selected as SPI2 clock
  - RCC_SPI2CLKSOURCE_HSI HSI selected as SPI2 clock
  - RCC_SPI2CLKSOURCE_PIN External I2S_CKIN selected as SPI2 clock

Return value:

- None

__HAL_RCC_GET_SPI2_SOURCE

Description:

- Macro to get the SPI2 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_SPI2CLKSOURCE_PCLK1 PCLK1 selected as SPI2 clock
  - RCC_SPI2CLKSOURCE_CLKP Peripheral clock CKLP selected as SPI2 clock
  - RCC_SPI2CLKSOURCE_IC8 IC8 selected as SPI2 clock
  - RCC_SPI2CLKSOURCE_IC9 IC9 selected as SPI2 clock
  - RCC_SPI2CLKSOURCE_MSI MSI selected as SPI2 clock
  - RCC_SPI2CLKSOURCE_HSI HSI selected as SPI2 clock
  - RCC_SPI2CLKSOURCE_PIN External I2S_CKIN selected as SPI2 clock

__HAL_RCC_SPI3_CONFIG

Description:

- Macro to configure the SPI3 clock source.

Parameters:

- __SPI3_CLKSOURCE__: defines the SPI3 clock source. This parameter can be one of the following values:
  - RCC_SPI3CLKSOURCE_PCLK1 PCLK1 selected as SPI3 clock
  - RCC_SPI3CLKSOURCE_CLKP Peripheral clock CKLP selected as SPI3 clock
  - RCC_SPI3CLKSOURCE_IC8 IC8 selected as SPI3 clock
  - RCC_SPI3CLKSOURCE_IC9 IC9 selected as SPI3 clock
  - RCC_SPI3CLKSOURCE_MSI MSI selected as SPI3 clock
  - RCC_SPI3CLKSOURCE_HSI HSI selected as SPI3 clock
  - RCC_SPI3CLKSOURCE_PIN External I2S_CKIN selected as SPI3 clock

Return value:

- None

__HAL_RCC_GET_SPI3_SOURCE

Description:

- Macro to get the SPI3 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_SPI3CLKSOURCE_PCLK1 PCLK1 selected as SPI3 clock
  - RCC_SPI3CLKSOURCE_CLKP Peripheral clock CKLP selected as SPI3 clock
  - RCC_SPI3CLKSOURCE_IC8 IC8 selected as SPI3 clock
  - RCC_SPI3CLKSOURCE_IC9 IC9 selected as SPI3 clock
  - RCC_SPI3CLKSOURCE_MSI MSI selected as SPI3 clock
  - RCC_SPI3CLKSOURCE_HSI HSI selected as SPI3 clock
  - RCC_SPI3CLKSOURCE_PIN External I2S_CKIN selected as SPI3 clock

__HAL_RCC_SPI4_CONFIG

Description:

- Macro to configure the SPI4 clock source.

Parameters:

- __SPI4_CLKSOURCE__: defines the SPI4 clock source. This parameter can be one of the following values:
  - RCC_SPI4CLKSOURCE_PCLK2 PCLK2 selected as SPI4 clock
  - RCC_SPI4CLKSOURCE_CLKP Peripheral clock CKLP selected as SPI4 clock
  - RCC_SPI4CLKSOURCE_IC8 IC8 selected as SPI4 clock
  - RCC_SPI4CLKSOURCE_IC9 IC9 selected as SPI4 clock
  - RCC_SPI4CLKSOURCE_MSI MSI selected as SPI4 clock
  - RCC_SPI4CLKSOURCE_HSI HSI selected as SPI4 clock
  - RCC_SPI4CLKSOURCE_PIN External I2S_CKIN selected as SPI4 clock

Return value:

- None

__HAL_RCC_GET_SPI4_SOURCE

Description:

- Macro to get the SPI4 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_SPI4CLKSOURCE_PCLK2 PCLK2 selected as SPI4 clock
  - RCC_SPI4CLKSOURCE_CLKP Peripheral clock CKLP selected as SPI4 clock
  - RCC_SPI4CLKSOURCE_IC8 IC8 selected as SPI4 clock
  - RCC_SPI4CLKSOURCE_IC9 IC9 selected as SPI4 clock
  - RCC_SPI4CLKSOURCE_MSI MSI selected as SPI4 clock
  - RCC_SPI4CLKSOURCE_HSI HSI selected as SPI4 clock
  - RCC_SPI4CLKSOURCE_PIN External I2S_CKIN selected as SPI4 clock

__HAL_RCC_SPI5_CONFIG

Description:

- Macro to configure the SPI5 clock source.

Parameters:

- __SPI5_CLKSOURCE__: defines the SPI5 clock source. This parameter can be one of the following values:
  - RCC_SPI5CLKSOURCE_PCLK2 PCLK2 selected as SPI5 clock
  - RCC_SPI5CLKSOURCE_CLKP Peripheral clock CKLP selected as SPI5 clock
  - RCC_SPI5CLKSOURCE_IC8 IC8 selected as SPI5 clock
  - RCC_SPI5CLKSOURCE_IC9 IC9 selected as SPI5 clock
  - RCC_SPI5CLKSOURCE_MSI MSI selected as SPI5 clock
  - RCC_SPI5CLKSOURCE_HSI HSI selected as SPI5 clock
  - RCC_SPI5CLKSOURCE_PIN External I2S_CKIN selected as SPI5 clock

Return value:

- None

__HAL_RCC_GET_SPI5_SOURCE

Description:

- Macro to get the SPI5 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_SPI5CLKSOURCE_PCLK2 PCLK2 selected as SPI5 clock
  - RCC_SPI5CLKSOURCE_CLKP Peripheral clock CKLP selected as SPI5 clock
  - RCC_SPI5CLKSOURCE_IC8 IC8 selected as SPI5 clock
  - RCC_SPI5CLKSOURCE_IC9 IC9 selected as SPI5 clock
  - RCC_SPI5CLKSOURCE_MSI MSI selected as SPI5 clock
  - RCC_SPI5CLKSOURCE_HSI HSI selected as SPI5 clock
  - RCC_SPI5CLKSOURCE_PIN External I2S_CKIN selected as SPI5 clock

__HAL_RCC_SPI6_CONFIG

Description:

- Macro to configure the SPI6 clock source.

Parameters:

- __SPI6_CLKSOURCE__: defines the SPI6 clock source. This parameter can be one of the following values:
  - RCC_SPI6CLKSOURCE_PCLK4 PCLK4 selected as SPI6 clock
  - RCC_SPI6CLKSOURCE_CLKP Peripheral clock CKLP selected as SPI6 clock
  - RCC_SPI6CLKSOURCE_IC8 IC8 selected as SPI6 clock
  - RCC_SPI6CLKSOURCE_IC9 IC9 selected as SPI6 clock
  - RCC_SPI6CLKSOURCE_MSI MSI selected as SPI6 clock
  - RCC_SPI6CLKSOURCE_HSI HSI selected as SPI6 clock
  - RCC_SPI6CLKSOURCE_PIN External I2S_CKIN selected as SPI6 clock

Return value:

- None

__HAL_RCC_GET_SPI6_SOURCE

Description:

- Macro to get the SPI6 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_SPI6CLKSOURCE_PCLK4 PCLK4 selected as SPI6 clock
  - RCC_SPI6CLKSOURCE_CLKP Peripheral clock CKLP selected as SPI6 clock
  - RCC_SPI6CLKSOURCE_IC8 IC8 selected as SPI6 clock
  - RCC_SPI6CLKSOURCE_IC9 IC9 selected as SPI6 clock
  - RCC_SPI6CLKSOURCE_MSI MSI selected as SPI6 clock
  - RCC_SPI6CLKSOURCE_HSI HSI selected as SPI6 clock
  - RCC_SPI6CLKSOURCE_PIN External I2S_CKIN selected as SPI6 clock

__HAL_RCC_USART1_CONFIG

Description:

- Macro to configure the USART1 clock source.

Parameters:

- __USART1_CLKSOURCE__: specifies the USART1 clock source. This parameter can be one of the following values:
  - RCC_USART1CLKSOURCE_PCLK2 PCLK2 selected as USART1 clock
  - RCC_USART1CLKSOURCE_CLKP Peripheral clock CKLP selected as USART1 clock
  - RCC_USART1CLKSOURCE_IC9 IC9 selected as USART1 clock
  - RCC_USART1CLKSOURCE_IC14 IC14 selected as USART1 clock
  - RCC_USART1CLKSOURCE_LSE LSE selected as USART1 clock
  - RCC_USART1CLKSOURCE_MSI MSI selected as USART1 clock
  - RCC_USART1CLKSOURCE_HSI HSI selected as USART1 clock

__HAL_RCC_GET_USART1_SOURCE

Description:

- Macro to get the USART1 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_USART1CLKSOURCE_PCLK2 PCLK2 selected as USART1 clock
  - RCC_USART1CLKSOURCE_CLKP Peripheral clock CKLP selected as USART1 clock
  - RCC_USART1CLKSOURCE_IC9 IC9 selected as USART1 clock
  - RCC_USART1CLKSOURCE_IC14 IC14 selected as USART1 clock
  - RCC_USART1CLKSOURCE_LSE LSE selected as USART1 clock
  - RCC_USART1CLKSOURCE_MSI MSI selected as USART1 clock
  - RCC_USART1CLKSOURCE_HSI HSI selected as USART1 clock

__HAL_RCC_USART2_CONFIG

Description:

- Macro to configure the USART2 clock source.

Parameters:

- __USART2_CLKSOURCE__: specifies the USART2 clock source. This parameter can be one of the following values:
  - RCC_USART2CLKSOURCE_PCLK1 PCLK1 selected as USART2 clock
  - RCC_USART2CLKSOURCE_CLKP Peripheral clock CKLP selected as USART2 clock
  - RCC_USART2CLKSOURCE_IC9 IC9 selected as USART2 clock
  - RCC_USART2CLKSOURCE_IC14 IC14 selected as USART2 clock
  - RCC_USART2CLKSOURCE_LSE LSE selected as USART2 clock
  - RCC_USART2CLKSOURCE_MSI MSI selected as USART2 clock
  - RCC_USART2CLKSOURCE_HSI HSI selected as USART2 clock

__HAL_RCC_GET_USART2_SOURCE

Description:

- Macro to get the USART2 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_USART2CLKSOURCE_PCLK1 PCLK1 selected as USART2 clock
  - RCC_USART2CLKSOURCE_CLKP Peripheral clock CKLP selected as USART2 clock
  - RCC_USART2CLKSOURCE_IC9 IC9 selected as USART2 clock
  - RCC_USART2CLKSOURCE_IC14 IC14 selected as USART2 clock
  - RCC_USART2CLKSOURCE_LSE LSE selected as USART2 clock
  - RCC_USART2CLKSOURCE_MSI MSI selected as USART2 clock
  - RCC_USART2CLKSOURCE_HSI HSI selected as USART2 clock

__HAL_RCC_USART3_CONFIG

Description:

- Macro to configure the USART3 clock source.

Parameters:

- __USART3_CLKSOURCE__: specifies the USART3 clock source. This parameter can be one of the following values:
  - RCC_USART3CLKSOURCE_PCLK1 PCLK1 selected as USART3 clock
  - RCC_USART3CLKSOURCE_CLKP Peripheral clock CKLP selected as USART3 clock
  - RCC_USART3CLKSOURCE_IC9 IC9 selected as USART3 clock
  - RCC_USART3CLKSOURCE_IC14 IC14 selected as USART3 clock
  - RCC_USART3CLKSOURCE_LSE LSE selected as USART3 clock
  - RCC_USART3CLKSOURCE_MSI MSI selected as USART3 clock
  - RCC_USART3CLKSOURCE_HSI HSI selected as USART3 clock

__HAL_RCC_GET_USART3_SOURCE

Description:

- Macro to get the USART3 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_USART3CLKSOURCE_PCLK1 PCLK1 selected as USART3 clock
  - RCC_USART3CLKSOURCE_CLKP Peripheral clock CKLP selected as USART3 clock
  - RCC_USART3CLKSOURCE_IC9 IC9 selected as USART3 clock
  - RCC_USART3CLKSOURCE_IC14 IC14 selected as USART3 clock
  - RCC_USART3CLKSOURCE_LSE LSE selected as USART3 clock
  - RCC_USART3CLKSOURCE_MSI MSI selected as USART3 clock
  - RCC_USART3CLKSOURCE_HSI HSI selected as USART3 clock

__HAL_RCC_UART4_CONFIG

Description:

- Macro to configure the UART4 clock source.

Parameters:

- __UART4_CLKSOURCE__: specifies the UART4 clock source. This parameter can be one of the following values:
  - RCC_UART4CLKSOURCE_PCLK1 PCLK1 selected as UART4 clock
  - RCC_UART4CLKSOURCE_CLKP Peripheral clock CKLP selected as UART4 clock
  - RCC_UART4CLKSOURCE_IC9 IC9 selected as UART4 clock
  - RCC_UART4CLKSOURCE_IC14 IC14 selected as UART4 clock
  - RCC_UART4CLKSOURCE_LSE LSE selected as UART4 clock
  - RCC_UART4CLKSOURCE_MSI MSI selected as UART4 clock
  - RCC_UART4CLKSOURCE_HSI HSI selected as UART4 clock

__HAL_RCC_GET_UART4_SOURCE

Description:

- Macro to get the UART4 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_UART4CLKSOURCE_PCLK1 PCLK1 selected as UART4 clock
  - RCC_UART4CLKSOURCE_CLKP Peripheral clock CKLP selected as UART4 clock
  - RCC_UART4CLKSOURCE_IC9 IC9 selected as UART4 clock
  - RCC_UART4CLKSOURCE_IC14 IC14 selected as UART4 clock
  - RCC_UART4CLKSOURCE_LSE LSE selected as UART4 clock
  - RCC_UART4CLKSOURCE_MSI MSI selected as UART4 clock
  - RCC_UART4CLKSOURCE_HSI HSI selected as UART4 clock

__HAL_RCC_UART5_CONFIG

Description:

- Macro to configure the UART5 clock source.

Parameters:

- __UART5_CLKSOURCE__: specifies the UART5 clock source. This parameter can be one of the following values:
  - RCC_UART5CLKSOURCE_PCLK1 PCLK1 selected as UART5 clock
  - RCC_UART5CLKSOURCE_CLKP Peripheral clock CKLP selected as UART5 clock
  - RCC_UART5CLKSOURCE_IC9 IC9 selected as UART5 clock
  - RCC_UART5CLKSOURCE_IC14 IC14 selected as UART5 clock
  - RCC_UART5CLKSOURCE_LSE LSE selected as UART5 clock
  - RCC_UART5CLKSOURCE_MSI MSI selected as UART5 clock
  - RCC_UART5CLKSOURCE_HSI HSI selected as UART5 clock

__HAL_RCC_GET_UART5_SOURCE

Description:

- Macro to get the UART5 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_UART5CLKSOURCE_PCLK1 PCLK1 selected as UART5 clock
  - RCC_UART5CLKSOURCE_CLKP Peripheral clock CKLP selected as UART5 clock
  - RCC_UART5CLKSOURCE_IC9 IC9 selected as UART5 clock
  - RCC_UART5CLKSOURCE_IC14 IC14 selected as UART5 clock
  - RCC_UART5CLKSOURCE_LSE LSE selected as UART5 clock
  - RCC_UART5CLKSOURCE_MSI MSI selected as UART5 clock
  - RCC_UART5CLKSOURCE_HSI HSI selected as UART5 clock

__HAL_RCC_USART6_CONFIG

Description:

- Macro to configure the USART6 clock source.

Parameters:

- __USART6_CLKSOURCE__: specifies the USART6 clock source. This parameter can be one of the following values:
  - RCC_USART6CLKSOURCE_PCLK2 PCLK2 selected as USART6 clock
  - RCC_USART6CLKSOURCE_CLKP Peripheral clock CKLP selected as USART6 clock
  - RCC_USART6CLKSOURCE_IC9 IC9 selected as USART6 clock
  - RCC_USART6CLKSOURCE_IC14 IC14 selected as USART6 clock
  - RCC_USART6CLKSOURCE_LSE LSE selected as USART6 clock
  - RCC_USART6CLKSOURCE_MSI MSI selected as USART6 clock
  - RCC_USART6CLKSOURCE_HSI HSI selected as USART6 clock

__HAL_RCC_GET_USART6_SOURCE

Description:

- Macro to get the USART6 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_USART6CLKSOURCE_PCLK2 PCLK2 selected as USART6 clock
  - RCC_USART6CLKSOURCE_CLKP Peripheral clock CKLP selected as USART6 clock
  - RCC_USART6CLKSOURCE_IC9 IC9 selected as USART6 clock
  - RCC_USART6CLKSOURCE_IC14 IC14 selected as USART6 clock
  - RCC_USART6CLKSOURCE_LSE LSE selected as USART6 clock
  - RCC_USART6CLKSOURCE_MSI MSI selected as USART6 clock
  - RCC_USART6CLKSOURCE_HSI HSI selected as USART6 clock

__HAL_RCC_UART7_CONFIG

Description:

- Macro to configure the UART7 clock source.

Parameters:

- __UART7_CLKSOURCE__: specifies the UART7 clock source. This parameter can be one of the following values:
  - RCC_UART7CLKSOURCE_PCLK1 PCLK1 selected as UART7 clock
  - RCC_UART7CLKSOURCE_CLKP Peripheral clock CKLP selected as UART7 clock
  - RCC_UART7CLKSOURCE_IC9 IC9 selected as UART7 clock
  - RCC_UART7CLKSOURCE_IC14 IC14 selected as UART7 clock
  - RCC_UART7CLKSOURCE_LSE LSE selected as UART7 clock
  - RCC_UART7CLKSOURCE_MSI MSI selected as UART7 clock
  - RCC_UART7CLKSOURCE_HSI HSI selected as UART7 clock

__HAL_RCC_GET_UART7_SOURCE

Description:

- Macro to get the UART7 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_UART7CLKSOURCE_PCLK1 PCLK1 selected as UART7 clock
  - RCC_UART7CLKSOURCE_CLKP Peripheral clock CKLP selected as UART7 clock
  - RCC_UART7CLKSOURCE_IC9 IC9 selected as UART7 clock
  - RCC_UART7CLKSOURCE_IC14 IC14 selected as UART7 clock
  - RCC_UART7CLKSOURCE_LSE LSE selected as UART7 clock
  - RCC_UART7CLKSOURCE_MSI MSI selected as UART7 clock
  - RCC_UART7CLKSOURCE_HSI HSI selected as UART7 clock

__HAL_RCC_UART8_CONFIG

Description:

- Macro to configure the UART8 clock source.

Parameters:

- __UART8_CLKSOURCE__: specifies the UART8 clock source. This parameter can be one of the following values:
  - RCC_UART8CLKSOURCE_PCLK1 PCLK1 selected as UART8 clock
  - RCC_UART8CLKSOURCE_CLKP Peripheral clock CKLP selected as UART8 clock
  - RCC_UART8CLKSOURCE_IC9 IC9 selected as UART8 clock
  - RCC_UART8CLKSOURCE_IC14 IC14 selected as UART8 clock
  - RCC_UART8CLKSOURCE_LSE LSE selected as UART8 clock
  - RCC_UART8CLKSOURCE_MSI MSI selected as UART8 clock
  - RCC_UART8CLKSOURCE_HSI HSI selected as UART8 clock

__HAL_RCC_GET_UART8_SOURCE

Description:

- Macro to get the UART8 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_UART8CLKSOURCE_PCLK1 PCLK1 selected as UART8 clock
  - RCC_UART8CLKSOURCE_CLKP Peripheral clock CKLP selected as UART8 clock
  - RCC_UART8CLKSOURCE_IC9 IC9 selected as UART8 clock
  - RCC_UART8CLKSOURCE_IC14 IC14 selected as UART8 clock
  - RCC_UART8CLKSOURCE_LSE LSE selected as UART8 clock
  - RCC_UART8CLKSOURCE_MSI MSI selected as UART8 clock
  - RCC_UART8CLKSOURCE_HSI HSI selected as UART8 clock

__HAL_RCC_UART9_CONFIG

Description:

- Macro to configure the UART9 clock source.

Parameters:

- __UART9_CLKSOURCE__: specifies the UART9 clock source. This parameter can be one of the following values:
  - RCC_UART9CLKSOURCE_PCLK2 PCLK2 selected as UART9 clock
  - RCC_UART9CLKSOURCE_CLKP Peripheral clock CKLP selected as UART9 clock
  - RCC_UART9CLKSOURCE_IC9 IC9 selected as UART9 clock
  - RCC_UART9CLKSOURCE_IC14 IC14 selected as UART9 clock
  - RCC_UART9CLKSOURCE_LSE LSE selected as UART9 clock
  - RCC_UART9CLKSOURCE_MSI MSI selected as UART9 clock
  - RCC_UART9CLKSOURCE_HSI HSI selected as UART9 clock

__HAL_RCC_GET_UART9_SOURCE

Description:

- Macro to get the UART9 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_UART9CLKSOURCE_PCLK2 PCLK2 selected as UART9 clock
  - RCC_UART9CLKSOURCE_CLKP Peripheral clock CKLP selected as UART9 clock
  - RCC_UART9CLKSOURCE_IC9 IC9 selected as UART9 clock
  - RCC_UART9CLKSOURCE_IC14 IC14 selected as UART9 clock
  - RCC_UART9CLKSOURCE_LSE LSE selected as UART9 clock
  - RCC_UART9CLKSOURCE_MSI MSI selected as UART9 clock
  - RCC_UART9CLKSOURCE_HSI HSI selected as UART9 clock

__HAL_RCC_USART10_CONFIG

Description:

- Macro to configure the USART10 clock source.

Parameters:

- __USART10_CLKSOURCE__: specifies the USART10 clock source. This parameter can be one of the following values:
  - RCC_USART10CLKSOURCE_PCLK2 PCLK2 selected as USART10 clock
  - RCC_USART10CLKSOURCE_CLKP Peripheral clock CKLP selected as USART10 clock
  - RCC_USART10CLKSOURCE_IC9 IC9 selected as USART10 clock
  - RCC_USART10CLKSOURCE_IC14 IC14 selected as USART10 clock
  - RCC_USART10CLKSOURCE_LSE LSE selected as USART10 clock
  - RCC_USART10CLKSOURCE_MSI MSI selected as USART10 clock
  - RCC_USART10CLKSOURCE_HSI HSI selected as USART10 clock

__HAL_RCC_GET_USART10_SOURCE

Description:

- Macro to get the USART10 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_USART10CLKSOURCE_PCLK2 PCLK2 selected as USART10 clock
  - RCC_USART10CLKSOURCE_CLKP Peripheral clock CKLP selected as USART10 clock
  - RCC_USART10CLKSOURCE_IC9 IC9 selected as USART10 clock
  - RCC_USART10CLKSOURCE_IC14 IC14 selected as USART10 clock
  - RCC_USART10CLKSOURCE_LSE LSE selected as USART10 clock
  - RCC_USART10CLKSOURCE_MSI MSI selected as USART10 clock
  - RCC_USART10CLKSOURCE_HSI HSI selected as USART10 clock

__HAL_RCC_XSPI1_CONFIG

Description:

- Macro to configure the XSPI1 clock source.

Parameters:

- __XSPI1_CLKSOURCE__: specifies the XSPI1 clock source.
  - RCC_XSPI1CLKSOURCE_HCLK HCLK selected as XSPI1 clock
  - RCC_XSPI1CLKSOURCE_CLKP Peripheral clock CKLP selected as XSPI1 clock
  - RCC_XSPI1CLKSOURCE_IC3 IC3 selected as XSPI1 clock
  - RCC_XSPI1CLKSOURCE_IC4 IC4 selected as XSPI1 clock

__HAL_RCC_GET_XSPI1_SOURCE

Description:

- Macro to get the XSPI1 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_XSPI1CLKSOURCE_HCLK HCLK selected as XSPI1 clock
  - RCC_XSPI1CLKSOURCE_CLKP Peripheral clock CKLP selected as XSPI1 clock
  - RCC_XSPI1CLKSOURCE_IC3 IC3 selected as XSPI1 clock
  - RCC_XSPI1CLKSOURCE_IC4 IC4 selected as XSPI1 clock

__HAL_RCC_XSPI2_CONFIG

Description:

- Macro to configure the XSPI2 clock source.

Parameters:

- __XSPI2_CLKSOURCE__: specifies the XSPI2 clock source.
  - RCC_XSPI2CLKSOURCE_HCLK HCLK selected as XSPI2 clock
  - RCC_XSPI2CLKSOURCE_CLKP Peripheral clock CKLP selected as XSPI2 clock
  - RCC_XSPI2CLKSOURCE_IC3 IC3 selected as XSPI2 clock
  - RCC_XSPI2CLKSOURCE_IC4 IC4 selected as XSPI2 clock

__HAL_RCC_GET_XSPI2_SOURCE

Description:

- Macro to get the XSPI2 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_XSPI2CLKSOURCE_HCLK HCLK selected as XSPI2 clock
  - RCC_XSPI2CLKSOURCE_CLKP Peripheral clock CKLP selected as XSPI2 clock
  - RCC_XSPI2CLKSOURCE_IC3 IC3 selected as XSPI2 clock
  - RCC_XSPI2CLKSOURCE_IC4 IC4 selected as XSPI2 clock

__HAL_RCC_XSPI3_CONFIG

Description:

- Macro to configure the XSPI3 clock source.

Parameters:

- __XSPI3_CLKSOURCE__: specifies the XSPI3 clock source.
  - RCC_XSPI3CLKSOURCE_HCLK HCLK selected as XSPI3 clock
  - RCC_XSPI3CLKSOURCE_CLKP Peripheral clock CKLP selected as XSPI3 clock
  - RCC_XSPI3CLKSOURCE_IC3 IC3 selected as XSPI3 clock
  - RCC_XSPI3CLKSOURCE_IC4 IC4 selected as XSPI3 clock

__HAL_RCC_GET_XSPI3_SOURCE

Description:

- Macro to get the XSPI3 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_XSPI3CLKSOURCE_HCLK HCLK selected as XSPI3 clock
  - RCC_XSPI3CLKSOURCE_CLKP Peripheral clock CKLP selected as XSPI3 clock
  - RCC_XSPI3CLKSOURCE_IC3 IC3 selected as XSPI3 clock
  - RCC_XSPI3CLKSOURCE_IC4 IC4 selected as XSPI3 clock

__HAL_RCC_TIMCLKPRESCALER_CONFIG

Description:

- Macro to configure the Timers clocks prescaler.

Parameters:

- __PRESC__: specifies the Timers clocks prescaler selection This parameter can be one of the following values:
  - RCC_TIMPRES_DIV1 The Timers kernels clocks prescaler is 1
  - RCC_TIMPRES_DIV2 The Timers kernels clocks prescaler is 2
  - RCC_TIMPRES_DIV4 The Timers kernels clocks prescaler is 4
  - RCC_TIMPRES_DIV8 The Timers kernels clocks prescaler is 8

__HAL_RCC_GET_TIMCLKPRESCALER

Description:

- Macro to get the Timers clocks prescaler.

Return value:

- Timers: clocks prescaler can be one of the following values:
  - RCC_TIMPRES_DIV1 The Timers kernels clocks prescaler is 1
  - RCC_TIMPRES_DIV2 The Timers kernels clocks prescaler is 2
  - RCC_TIMPRES_DIV4 The Timers kernels clocks prescaler is 4
  - RCC_TIMPRES_DIV8 The Timers kernels clocks prescaler is 8

__HAL_RCC_USBPHY1_CONFIG

Description:

- Macro to configure the USBPHY1 clock.

Parameters:

- __USBPHY1_CLKSOURCE__: specifies the USBPHY1 clock source. This parameter can be one of the following values:
  - RCC_USBPHY1REFCLKSOURCE_OTGPHY1 USB OTGPHY1 kernel clock selected as USBPHY1 clock
  - RCC_USBPHY1REFCLKSOURCE_HSE_DIRECT HSE from oscillator selected as USBPHY1 clock
  - RCC_USBPHY1REFCLKSOURCE_HSE_DIRECT_DIV2 HSE from oscillator divided by 2 selected as USBPHY1 clock

__HAL_RCC_GET_USBPHY1_SOURCE

Description:

- Macro to get the USBPHY1 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_USBPHY1REFCLKSOURCE_OTGPHY1 USB OTGPHY1 kernel clock selected as USBPHY1 clock
  - RCC_USBPHY1REFCLKSOURCE_HSE_DIRECT HSE from oscillator selected as USBPHY1 clock
  - RCC_USBPHY1REFCLKSOURCE_HSE_DIRECT_DIV2 HSE from oscillator divided by 2 selected as USBPHY1 clock

__HAL_RCC_USBPHY2_CONFIG

Description:

- Macro to configure the USBPHY2 clock.

Parameters:

- __USBPHY2_CLKSOURCE__: specifies the USBPHY2 clock source. This parameter can be one of the following values:
  - RCC_USBPHY2REFCLKSOURCE_OTGPHY2 USB OTGPHY2 kernel clock selected as USBPHY2 clock
  - RCC_USBPHY2REFCLKSOURCE_HSE_DIRECT HSE from oscillator selected as USBPHY2 clock
  - RCC_USBPHY2REFCLKSOURCE_HSE_DIRECT_DIV2 HSE from oscillator divided by 2 selected as USBPHY2 clock

__HAL_RCC_GET_USBPHY2_SOURCE

Description:

- Macro to get the USBPHY2 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_USBPHY2REFCLKSOURCE_OTGPHY2 USB OTGPHY2 kernel clock selected as USBPHY2 clock
  - RCC_USBPHY2REFCLKSOURCE_HSE_DIRECT HSE from oscillator selected as USBPHY2 clock
  - RCC_USBPHY2REFCLKSOURCE_HSE_DIRECT_DIV2 HSE from oscillator divided by 2 selected as USBPHY2 clock

__HAL_RCC_USBOTGHS1_CONFIG

Description:

- Macro to configure the USB OTGHS1 clock.

Parameters:

- __USBOTGHS1_CLKSOURCE__: specifies the USB OTGHS1 clock source. This parameter can be one of the following values:
  - RCC_USBOTGHS1CLKSOURCE_HSE_DIV2 HSE divided by 2 selected as USBOTGHS1 clock
  - RCC_USBOTGHS1CLKSOURCE_CLKP Peripheral clock CKLP selected as USBOTGHS1 clock
  - RCC_USBOTGHS1CLKSOURCE_IC15 IC15 selected as USBOTGHS1 clock
  - RCC_USBOTGHS1CLKSOURCE_HSE_DIV_2_OSC HSE divided by 2 from oscillator selected as USBOTGHS1 clock

__HAL_RCC_GET_USBOTGHS1_SOURCE

Description:

- Macro to get the USB OTGHS1 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_USBOTGHS1CLKSOURCE_HSE_DIV2 HSE divided by 2 selected as USBOTGHS1 clock
  - RCC_USBOTGHS1CLKSOURCE_CLKP Peripheral clock CKLP selected as USBOTGHS1 clock
  - RCC_USBOTGHS1CLKSOURCE_IC15 IC15 selected as USBOTGHS1 clock
  - RCC_USBOTGHS1CLKSOURCE_HSE_DIV_2_OSC HSE divided by 2 from oscillator selected as USBOTGHS1 clock

__HAL_RCC_USBOTGHS2_CONFIG

Description:

- Macro to configure the USB OTGHS2 clock.

Parameters:

- __USBOTGHS2_CLKSOURCE__: specifies the USB OTGHS2 clock source. This parameter can be one of the following values:
  - RCC_USBOTGHS2CLKSOURCE_HSE_DIV2 HSE divided by 2 selected as USBOTGHS2 clock
  - RCC_USBOTGHS2CLKSOURCE_CLKP Peripheral clock CKLP selected as USBOTGHS2 clock
  - RCC_USBOTGHS2CLKSOURCE_IC15 IC15 selected as USBOTGHS2 clock
  - RCC_USBOTGHS2CLKSOURCE_HSE_DIV_2_OSC HSE divided by 2 from oscillator selected as USBOTGHS2 clock

__HAL_RCC_GET_USBOTGHS2_SOURCE

Description:

- Macro to get the USB OTGHS2 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_USBOTGHS2CLKSOURCE_HSE_DIV2 HSE divided by 2 selected as USBOTGHS2 clock
  - RCC_USBOTGHS2CLKSOURCE_CLKP Peripheral clock CKLP selected as USBOTGHS2 clock
  - RCC_USBOTGHS2C LKSOURCE_IC15 IC15 selected as USBOTGHS2 clock
  - RCC_USBOTGHS2CLKSOURCE_HSE_DIV_2_OSC HSE divided by 2 from oscillator selected as USBOTGHS2 clock

__HAL_RCC_LSECSS_EXTI_ENABLE_IT

Description:

- Enable the RCC LSE CSS Extended Interrupt Line.

Return value:

- None

__HAL_RCC_LSECSS_EXTI_DISABLE_IT

Description:

- Disable the RCC LSE CSS Extended Interrupt Line.

Return value:

- None

__HAL_RCC_LSECSS_EXTI_ENABLE_EVENT

Description:

- Enable the RCC LSE CSS Event Line.

Return value:

- None.

__HAL_RCC_LSECSS_EXTI_DISABLE_EVENT

Description:

- Disable the RCC LSE CSS Event Line.

Return value:

- None.

LSE CSS external interrupt line

RCC_EXTI_LINE_LSECSS

LSE CSS failure interrupt connected to direct EXTI line 65 (Tamper)

FDCAN Kernel Clock Source

RCC_FDCANCLKSOURCE_PCLK1

RCC_FDCANCLKSOURCE_CLKP

RCC_FDCANCLKSOURCE_IC19

RCC_FDCANCLKSOURCE_HSE

FMC Clock Source

RCC_FMCCLKSOURCE_HCLK

RCC_FMCCLKSOURCE_CLKP

RCC_FMCCLKSOURCE_IC3

RCC_FMCCLKSOURCE_IC4

I2C1 Clock Source

RCC_I2C1CLKSOURCE_PCLK1

RCC_I2C1CLKSOURCE_CLKP

RCC_I2C1CLKSOURCE_IC10

RCC_I2C1CLKSOURCE_IC15

RCC_I2C1CLKSOURCE_MSI

RCC_I2C1CLKSOURCE_HSI

I2C2 Clock Source

RCC_I2C2CLKSOURCE_PCLK1

RCC_I2C2CLKSOURCE_CLKP

RCC_I2C2CLKSOURCE_IC10

RCC_I2C2CLKSOURCE_IC15

RCC_I2C2CLKSOURCE_MSI

RCC_I2C2CLKSOURCE_HSI

I2C3 Clock Source

RCC_I2C3CLKSOURCE_PCLK1

RCC_I2C3CLKSOURCE_CLKP

RCC_I2C3CLKSOURCE_IC10

RCC_I2C3CLKSOURCE_IC15

RCC_I2C3CLKSOURCE_MSI

RCC_I2C3CLKSOURCE_HSI

I2C4 Clock Source

RCC_I2C4CLKSOURCE_PCLK1

RCC_I2C4CLKSOURCE_CLKP

RCC_I2C4CLKSOURCE_IC10

RCC_I2C4CLKSOURCE_IC15

RCC_I2C4CLKSOURCE_MSI

RCC_I2C4CLKSOURCE_HSI

I3C1 Clock Source

RCC_I3C1CLKSOURCE_PCLK1

RCC_I3C1CLKSOURCE_CLKP

RCC_I3C1CLKSOURCE_IC10

RCC_I3C1CLKSOURCE_IC15

RCC_I3C1CLKSOURCE_MSI

RCC_I3C1CLKSOURCE_HSI

I3C2 Clock Source

RCC_I3C2CLKSOURCE_PCLK1

RCC_I3C2CLKSOURCE_CLKP

RCC_I3C2CLKSOURCE_IC10

RCC_I3C2CLKSOURCE_IC15

RCC_I3C2CLKSOURCE_MSI

RCC_I3C2CLKSOURCE_HSI

IC Selection

RCC_IC1

IC1 source is PLL1 output by default

RCC_IC2

IC2 source is PLL1 output by default

RCC_IC3

IC3 source is PLL1 output by default

RCC_IC4

IC4 source is PLL1 output by default

RCC_IC5

IC5 source is PLL1 output by default

RCC_IC6

IC6 source is PLL1 output by default

RCC_IC7

IC7 source is PLL2 output by default

RCC_IC8

IC8 source is PLL2 output by default

RCC_IC9

IC9 source is PLL2 output by default

RCC_IC10

IC10 source is PLL2 output by default

RCC_IC11

IC11 source is PLL1 output by default

RCC_IC12

IC12 source is PLL3 output by default

RCC_IC13

IC13 source is PLL3 output by default

RCC_IC14

IC14 source is PLL3 output by default

RCC_IC15

IC15 source is PLL3 output by default

RCC_IC16

IC16 source is PLL4 output by default

RCC_IC17

IC17 source is PLL4 output by default

RCC_IC18

IC18 source is PLL4 output by default

RCC_IC19

IC19 source is PLL4 output by default

RCC_IC20

IC20 source is PLL4 output by default

Maximum IC Selection

RCC_IC_MAX_NUMBER

Private macros to check input parameters

IS_RCC_PERIPHCLOCK

IS_RCC_ADCCLKSOURCE

IS_RCC_ADCDIVIDER

IS_RCC_ADF1CLKSOURCE

IS_RCC_CKPERCLKSOURCE

IS_RCC_DCMIPPCLKSOURCE

IS_RCC_ETH1CLKSOURCE

IS_RCC_ETH1PHYIF

IS_RCC_ETH1RXCLKSOURCE

IS_RCC_ETH1TXCLKSOURCE

IS_RCC_ETH1PTPCLKSOURCE

IS_RCC_ETH1PTPDIVIDER

IS_RCC_FDCANCLKSOURCE

IS_RCC_FMCCLKSOURCE

IS_RCC_I2C1CLKSOURCE

IS_RCC_I2C2CLKSOURCE

IS_RCC_I2C3CLKSOURCE

IS_RCC_I2C4CLKSOURCE

IS_RCC_I3C1CLKSOURCE

IS_RCC_I3C2CLKSOURCE

IS_RCC_LPTIM1CLKSOURCE

IS_RCC_LPTIM2CLKSOURCE

IS_RCC_LPTIM3CLKSOURCE

IS_RCC_LPTIM4CLKSOURCE

IS_RCC_LPTIM5CLKSOURCE

IS_RCC_LPUART1CLKSOURCE

IS_RCC_LTDCCLKSOURCE

IS_RCC_MDF1CLKSOURCE

IS_RCC_PSSICLKSOURCE

IS_RCC_SAI1CLKSOURCE

IS_RCC_SAI2CLKSOURCE

IS_RCC_SDMMC1CLKSOURCE

IS_RCC_SDMMC2CLKSOURCE

IS_RCC_SPDIFRX1CLKSOURCE

IS_RCC_SPI1CLKSOURCE

IS_RCC_SPI2CLKSOURCE

IS_RCC_SPI3CLKSOURCE

IS_RCC_SPI4CLKSOURCE

IS_RCC_SPI5CLKSOURCE

IS_RCC_SPI6CLKSOURCE

IS_RCC_TIMPRES

IS_RCC_USART1CLKSOURCE

IS_RCC_USART2CLKSOURCE

IS_RCC_USART3CLKSOURCE

IS_RCC_UART4CLKSOURCE

IS_RCC_UART5CLKSOURCE

IS_RCC_USART6CLKSOURCE

IS_RCC_UART7CLKSOURCE

IS_RCC_UART8CLKSOURCE

IS_RCC_UART9CLKSOURCE

IS_RCC_USART10CLKSOURCE

IS_RCC_USBPHY1CLKSOURCE

IS_RCC_USBPHY2CLKSOURCE

IS_RCC_USBOTGHS1CLKSOURCE

IS_RCC_USBOTGHS2CLKSOURCE

IS_RCC_XSPI1CLKSOURCE

IS_RCC_XSPI2CLKSOURCE

IS_RCC_XSPI3CLKSOURCE

LPTIM1 Clock Source

RCC_LPTIM1CLKSOURCE_PCLK1

RCC_LPTIM1CLKSOURCE_CLKP

RCC_LPTIM1CLKSOURCE_IC15

RCC_LPTIM1CLKSOURCE_LSE

RCC_LPTIM1CLKSOURCE_LSI

RCC_LPTIM1CLKSOURCE_TIMG

LPTIM2 Clock Source

RCC_LPTIM2CLKSOURCE_PCLK4

RCC_LPTIM2CLKSOURCE_CLKP

RCC_LPTIM2CLKSOURCE_IC15

RCC_LPTIM2CLKSOURCE_LSE

RCC_LPTIM2CLKSOURCE_LSI

RCC_LPTIM2CLKSOURCE_TIMG

LPTIM3 Clock Source

RCC_LPTIM3CLKSOURCE_PCLK4

RCC_LPTIM3CLKSOURCE_CLKP

RCC_LPTIM3CLKSOURCE_IC15

RCC_LPTIM3CLKSOURCE_LSE

RCC_LPTIM3CLKSOURCE_LSI

RCC_LPTIM3CLKSOURCE_TIMG

LPTIM4 Clock Source

RCC_LPTIM4CLKSOURCE_PCLK4

RCC_LPTIM4CLKSOURCE_CLKP

RCC_LPTIM4CLKSOURCE_IC15

RCC_LPTIM4CLKSOURCE_LSE

RCC_LPTIM4CLKSOURCE_LSI

RCC_LPTIM4CLKSOURCE_TIMG

LPTIM5 Clock Source

RCC_LPTIM5CLKSOURCE_PCLK4

RCC_LPTIM5CLKSOURCE_CLKP

RCC_LPTIM5CLKSOURCE_IC15

RCC_LPTIM5CLKSOURCE_LSE

RCC_LPTIM5CLKSOURCE_LSI

RCC_LPTIM5CLKSOURCE_TIMG

LPUART1 Clock Source

RCC_LPUART1CLKSOURCE_PCLK4

RCC_LPUART1CLKSOURCE_CLKP

RCC_LPUART1CLKSOURCE_IC9

RCC_LPUART1CLKSOURCE_IC14

RCC_LPUART1CLKSOURCE_LSE

RCC_LPUART1CLKSOURCE_MSI

RCC_LPUART1CLKSOURCE_HSI

LTDC Clock Source

RCC_LTDCCLKSOURCE_PCLK5

RCC_LTDCCLKSOURCE_CLKP

RCC_LTDCCLKSOURCE_IC16

RCC_LTDCCLKSOURCE_HSI

RCC Extended MCOx Clock Config

__HAL_RCC_MCO1_ENABLE

Description:

- Macros to enable or disable the MCO1 clock output.

Return value:

- None

__HAL_RCC_MCO1_DISABLE

__HAL_RCC_MCO2_ENABLE

Description:

- Macros to enable or disable the MCO2 clock output.

Return value:

- None

__HAL_RCC_MCO2_DISABLE

__HAL_RCC_MCO1_CONFIG

Description:

- Macro to configure the MCO1 clock.

Parameters:

- __MCOCLKSOURCE__: specifies the MCO clock source. This parameter can be one of the following values:
  - RCC_MCO1SOURCE_HSI HSI clock selected as MCO1 source
  - RCC_MCO1SOURCE_LSE LSE clock selected as MCO1 source
  - RCC_MCO1SOURCE_MSI MSI clock selected as MCO1 source
  - RCC_MCO1SOURCE_LSI LSI clock selected as MCO1 source
  - RCC_MCO1SOURCE_HSE HSE clock selected as MCO1 source
  - RCC_MCO1SOURCE_IC5 IC5 clock selected as MCO1 source
  - RCC_MCO1SOURCE_IC10 IC10 clock selected as MCO1 source
  - RCC_MCO1SOURCE_SYSA SYSA CPU clock selected as MCO1 source
- __MCODIV__: specifies the MCO clock prescaler. This parameter can be one of the following values:
  - RCC_MCODIV_1 up to RCC_MCODIV_16 : divider applied to MCO1 clock

Notes:

- MCO1 clock output shall be enabled with __HAL_RCC_MCO1_ENABLE() The MCO switch to the new clock source only occurs when the previous clock source is active (dynamic switch).

__HAL_RCC_MCO2_CONFIG

Description:

- Macro to configure the MCO2 clock.

Parameters:

- __MCOCLKSOURCE__: specifies the MCO clock source. This parameter can be one of the following values:
  - RCC_MCO2SOURCE_HSI HSI clock selected as MCO2 source
  - RCC_MCO2SOURCE_LSE LSE clock selected as MCO2 source
  - RCC_MCO2SOURCE_MSI MSI clock selected as MCO2 source
  - RCC_MCO2SOURCE_LSI LSI clock selected as MCO2 source
  - RCC_MCO2SOURCE_HSE HSE clock selected as MCO2 source
  - RCC_MCO2SOURCE_IC15 IC15 clock selected as MCO2 source
  - RCC_MCO2SOURCE_IC20 IC20 clock selected as MCO2 source
  - RCC_MCO2SOURCE_SYSB SYSB bus clock selected as MCO2 source
- __MCODIV__: specifies the MCO clock prescaler. This parameter can be one of the following values:
  - RCC_MCODIV_1 up to RCC_MCODIV_16 : divider applied to MCO2 clock

Notes:

- MCO2 clock output shall be enabled with __HAL_RCC_MCO2_ENABLE() The MCO switch to the new clock source only occurs when the previous clock source is active (dynamic switch).

MDF1 Clock Source

RCC_MDF1CLKSOURCE_HCLK

RCC_MDF1CLKSOURCE_CLKP

RCC_MDF1CLKSOURCE_IC7

RCC_MDF1CLKSOURCE_IC8

RCC_MDF1CLKSOURCE_MSI

RCC_MDF1CLKSOURCE_HSI

RCC_MDF1CLKSOURCE_PIN

RCC_MDF1CLKSOURCE_TIMG

Periph Clock Selection

RCC_PERIPHCLK_ADC

RCC_PERIPHCLK_ADF1

RCC_PERIPHCLK_CKPER

RCC_PERIPHCLK_CSI

RCC_PERIPHCLK_DCMIPP

RCC_PERIPHCLK_ETH1

RCC_PERIPHCLK_ETH1PHY

RCC_PERIPHCLK_ETH1RX

RCC_PERIPHCLK_ETH1TX

RCC_PERIPHCLK_ETH1PTP

RCC_PERIPHCLK_FDCAN

RCC_PERIPHCLK_FMC

RCC_PERIPHCLK_I2C1

RCC_PERIPHCLK_I2C2

RCC_PERIPHCLK_I2C3

RCC_PERIPHCLK_I2C4

RCC_PERIPHCLK_I3C1

RCC_PERIPHCLK_I3C2

RCC_PERIPHCLK_LPTIM1

RCC_PERIPHCLK_LPTIM2

RCC_PERIPHCLK_LPTIM3

RCC_PERIPHCLK_LPTIM4

RCC_PERIPHCLK_LPTIM5

RCC_PERIPHCLK_LPUART1

RCC_PERIPHCLK_LTDC

RCC_PERIPHCLK_MDF1

RCC_PERIPHCLK_PSSI

RCC_PERIPHCLK_RTC

RCC_PERIPHCLK_SAI1

RCC_PERIPHCLK_SAI2

RCC_PERIPHCLK_SDMMC1

RCC_PERIPHCLK_SDMMC2

RCC_PERIPHCLK_SPDIFRX1

RCC_PERIPHCLK_SPI1

RCC_PERIPHCLK_SPI2

RCC_PERIPHCLK_SPI3

RCC_PERIPHCLK_SPI4

RCC_PERIPHCLK_SPI5

RCC_PERIPHCLK_SPI6

RCC_PERIPHCLK_TIM

RCC_PERIPHCLK_USART1

RCC_PERIPHCLK_USART2

RCC_PERIPHCLK_USART3

RCC_PERIPHCLK_UART4

RCC_PERIPHCLK_UART5

RCC_PERIPHCLK_USART6

RCC_PERIPHCLK_UART7

RCC_PERIPHCLK_UART8

RCC_PERIPHCLK_UART9

RCC_PERIPHCLK_USART10

RCC_PERIPHCLK_USBPHY1

RCC_PERIPHCLK_USBOTGHS1

RCC_PERIPHCLK_USBPHY2

RCC_PERIPHCLK_USBOTGHS2

RCC_PERIPHCLK_XSPI1

RCC_PERIPHCLK_XSPI2

RCC_PERIPHCLK_XSPI3

Peripheral clock frequency

RCC_PERIPH_FREQUENCY_NO

No clock enabled for the peripheral

RCC_PERIPH_FREQUENCY_NA

Frequency cannot be provided as external clock

PSSI Clock Source

RCC_PSSICLKSOURCE_HCLK

RCC_PSSICLKSOURCE_CLKP

RCC_PSSICLKSOURCE_IC20

RCC_PSSICLKSOURCE_HSI

SAI1 Clock Source

RCC_SAI1CLKSOURCE_PCLK2

RCC_SAI1CLKSOURCE_CLKP

RCC_SAI1CLKSOURCE_IC7

RCC_SAI1CLKSOURCE_IC8

RCC_SAI1CLKSOURCE_MSI

RCC_SAI1CLKSOURCE_HSI

RCC_SAI1CLKSOURCE_PIN

RCC_SAI1CLKSOURCE_SPDIFRX1

SAI2 Clock Source

RCC_SAI2CLKSOURCE_PCLK2

RCC_SAI2CLKSOURCE_CLKP

RCC_SAI2CLKSOURCE_IC7

RCC_SAI2CLKSOURCE_IC8

RCC_SAI2CLKSOURCE_MSI

RCC_SAI2CLKSOURCE_HSI

RCC_SAI2CLKSOURCE_PIN

RCC_SAI2CLKSOURCE_SPDIFRX1

SDMMC1 Clock Source

RCC_SDMMC1CLKSOURCE_HCLK

RCC_SDMMC1CLKSOURCE_CLKP

RCC_SDMMC1CLKSOURCE_IC4

RCC_SDMMC1CLKSOURCE_IC5

SDMMC2 Clock Source

RCC_SDMMC2CLKSOURCE_HCLK

RCC_SDMMC2CLKSOURCE_CLKP

RCC_SDMMC2CLKSOURCE_IC4

RCC_SDMMC2CLKSOURCE_IC5

SPDIFRX1 Clock Source

RCC_SPDIFRX1CLKSOURCE_PCLK1

RCC_SPDIFRX1CLKSOURCE_CLKP

RCC_SPDIFRX1CLKSOURCE_IC7

RCC_SPDIFRX1CLKSOURCE_IC8

RCC_SPDIFRX1CLKSOURCE_MSI

RCC_SPDIFRX1CLKSOURCE_HSI

RCC_SPDIFRX1CLKSOURCE_PIN

SPI1 Clock Source

RCC_SPI1CLKSOURCE_PCLK2

RCC_SPI1CLKSOURCE_CLKP

RCC_SPI1CLKSOURCE_IC8

RCC_SPI1CLKSOURCE_IC9

RCC_SPI1CLKSOURCE_MSI

RCC_SPI1CLKSOURCE_HSI

RCC_SPI1CLKSOURCE_PIN

SPI2 Clock Source

RCC_SPI2CLKSOURCE_PCLK1

RCC_SPI2CLKSOURCE_CLKP

RCC_SPI2CLKSOURCE_IC8

RCC_SPI2CLKSOURCE_IC9

RCC_SPI2CLKSOURCE_MSI

RCC_SPI2CLKSOURCE_HSI

RCC_SPI2CLKSOURCE_PIN

SPI3 Clock Source

RCC_SPI3CLKSOURCE_PCLK1

RCC_SPI3CLKSOURCE_CLKP

RCC_SPI3CLKSOURCE_IC8

RCC_SPI3CLKSOURCE_IC9

RCC_SPI3CLKSOURCE_MSI

RCC_SPI3CLKSOURCE_HSI

RCC_SPI3CLKSOURCE_PIN

SPI4 Clock Source

RCC_SPI4CLKSOURCE_PCLK2

RCC_SPI4CLKSOURCE_CLKP

RCC_SPI4CLKSOURCE_IC9

RCC_SPI4CLKSOURCE_IC14

RCC_SPI4CLKSOURCE_MSI

RCC_SPI4CLKSOURCE_HSI

RCC_SPI4CLKSOURCE_HSE

SPI5 Clock Source

RCC_SPI5CLKSOURCE_PCLK2

RCC_SPI5CLKSOURCE_CLKP

RCC_SPI5CLKSOURCE_IC9

RCC_SPI5CLKSOURCE_IC14

RCC_SPI5CLKSOURCE_MSI

RCC_SPI5CLKSOURCE_HSI

RCC_SPI5CLKSOURCE_HSE

SPI6 Clock Source

RCC_SPI6CLKSOURCE_PCLK4

RCC_SPI6CLKSOURCE_CLKP

RCC_SPI6CLKSOURCE_IC8

RCC_SPI6CLKSOURCE_IC9

RCC_SPI6CLKSOURCE_MSI

RCC_SPI6CLKSOURCE_HSI

RCC_SPI6CLKSOURCE_PIN

TIM Prescaler Selection

RCC_TIMPRES_DIV1
: Timers clocks prescaler divide by 1

RCC_TIMPRES_DIV2
: Timers clocks prescaler divide by 2

RCC_TIMPRES_DIV4
: Timers clocks prescaler divide by 4

RCC_TIMPRES_DIV8
: Timers clocks prescaler divide by 8

UART4 Clock Source

RCC_UART4CLKSOURCE_PCLK1

RCC_UART4CLKSOURCE_CLKP

RCC_UART4CLKSOURCE_IC9

RCC_UART4CLKSOURCE_IC14

RCC_UART4CLKSOURCE_LSE

RCC_UART4CLKSOURCE_MSI

RCC_UART4CLKSOURCE_HSI

UART5 Clock Source

RCC_UART5CLKSOURCE_PCLK1

RCC_UART5CLKSOURCE_CLKP

RCC_UART5CLKSOURCE_IC9

RCC_UART5CLKSOURCE_IC14

RCC_UART5CLKSOURCE_LSE

RCC_UART5CLKSOURCE_MSI

RCC_UART5CLKSOURCE_HSI

UART7 Clock Source

RCC_UART7CLKSOURCE_PCLK1

RCC_UART7CLKSOURCE_CLKP

RCC_UART7CLKSOURCE_IC9

RCC_UART7CLKSOURCE_IC14

RCC_UART7CLKSOURCE_LSE

RCC_UART7CLKSOURCE_MSI

RCC_UART7CLKSOURCE_HSI

UART8 Clock Source

RCC_UART8CLKSOURCE_PCLK1

RCC_UART8CLKSOURCE_CLKP

RCC_UART8CLKSOURCE_IC9

RCC_UART8CLKSOURCE_IC14

RCC_UART8CLKSOURCE_LSE

RCC_UART8CLKSOURCE_MSI

RCC_UART8CLKSOURCE_HSI

UART9 Clock Source

RCC_UART9CLKSOURCE_PCLK2

RCC_UART9CLKSOURCE_CLKP

RCC_UART9CLKSOURCE_IC9

RCC_UART9CLKSOURCE_IC14

RCC_UART9CLKSOURCE_LSE

RCC_UART9CLKSOURCE_MSI

RCC_UART9CLKSOURCE_HSI

USART10 Clock Source

RCC_USART10CLKSOURCE_PCLK2

RCC_USART10CLKSOURCE_CLKP

RCC_USART10CLKSOURCE_IC9

RCC_USART10CLKSOURCE_IC14

RCC_USART10CLKSOURCE_LSE

RCC_USART10CLKSOURCE_MSI

RCC_USART10CLKSOURCE_HSI

USART1 Clock Source

RCC_USART1CLKSOURCE_PCLK2

RCC_USART1CLKSOURCE_CLKP

RCC_USART1CLKSOURCE_IC9

RCC_USART1CLKSOURCE_IC14

RCC_USART1CLKSOURCE_LSE

RCC_USART1CLKSOURCE_MSI

RCC_USART1CLKSOURCE_HSI

USART2 Clock Source

RCC_USART2CLKSOURCE_PCLK1

RCC_USART2CLKSOURCE_CLKP

RCC_USART2CLKSOURCE_IC9

RCC_USART2CLKSOURCE_IC14

RCC_USART2CLKSOURCE_LSE

RCC_USART2CLKSOURCE_MSI

RCC_USART2CLKSOURCE_HSI

USART3 Clock Source

RCC_USART3CLKSOURCE_PCLK1

RCC_USART3CLKSOURCE_CLKP

RCC_USART3CLKSOURCE_IC9

RCC_USART3CLKSOURCE_IC14

RCC_USART3CLKSOURCE_LSE

RCC_USART3CLKSOURCE_MSI

RCC_USART3CLKSOURCE_HSI

USART6 Clock Source

RCC_USART6CLKSOURCE_PCLK2

RCC_USART6CLKSOURCE_CLKP

RCC_USART6CLKSOURCE_IC9

RCC_USART6CLKSOURCE_IC14

RCC_USART6CLKSOURCE_LSE

RCC_USART6CLKSOURCE_MSI

RCC_USART6CLKSOURCE_HSI

USBPHY1 Clock Source

RCC_USBPHY1REFCLKSOURCE_OTGPHY1

RCC_USBPHY1REFCLKSOURCE_HSE_DIRECT

RCC_USBPHY1REFCLKSOURCE_HSE_DIRECT_DIV2

USBPHY2 Clock Source

RCC_USBPHY2REFCLKSOURCE_OTGPHY2

RCC_USBPHY2REFCLKSOURCE_HSE_DIRECT

RCC_USBPHY2REFCLKSOURCE_HSE_DIRECT_DIV2

USB OTGHS1 Clock Source

RCC_USBOTGHS1CLKSOURCE_HSE_DIV2

RCC_USBOTGHS1CLKSOURCE_CLKP

RCC_USBOTGHS1CLKSOURCE_IC15

RCC_USBOTGHS1CLKSOURCE_HSE_DIRECT

RCC_USBOTGHS1CLKSOURCE_HSE_DIRECT_DIV2

USB OTGHS2 Clock Source

RCC_USBOTGHS2CLKSOURCE_HSE_DIV2

RCC_USBOTGHS2CLKSOURCE_CLKP

RCC_USBOTGHS2CLKSOURCE_IC15

RCC_USBOTGHS2CLKSOURCE_HSE_DIRECT

RCC_USBOTGHS2CLKSOURCE_HSE_DIRECT_DIV2

XSPI1 Clock Source

RCC_XSPI1CLKSOURCE_HCLK

RCC_XSPI1CLKSOURCE_CLKP

RCC_XSPI1CLKSOURCE_IC3

RCC_XSPI1CLKSOURCE_IC4

XSPI2 Clock Source

RCC_XSPI2CLKSOURCE_HCLK

RCC_XSPI2CLKSOURCE_CLKP

RCC_XSPI2CLKSOURCE_IC3

RCC_XSPI2CLKSOURCE_IC4

XSPI3 Clock Source

RCC_XSPI3CLKSOURCE_HCLK

RCC_XSPI3CLKSOURCE_CLKP

RCC_XSPI3CLKSOURCE_IC3

RCC_XSPI3CLKSOURCE_IC4

58 HAL RIF Generic Driver

58.1 RIF Firmware driver registers structures

58.1.1 RIMC_MasterConfig_t

RIMC_MasterConfig_t is defined in the stm32n6xx_hal_rif.h

Data Fields

- ***uint32_t MasterCID***
- ***uint32_t SecPriv***

Field Documentation

- ***uint32_t RIMC_MasterConfig_t::MasterCID***
  One of ***RIF_COMPARTMENT_ID***
- ***uint32_t RIMC_MasterConfig_t::SecPriv***
  A combination of ***RIF_SEC_PRIV***

58.1.2 RISAF_BaseRegionConfig_t

RISAF_BaseRegionConfig_t is defined in the stm32n6xx_hal_rif.h

Data Fields

- ***uint32_t Filtering***
- ***uint32_t Secure***
- ***uint32_t PrivWhitelist***
- ***uint32_t ReadWhitelist***
- ***uint32_t WriteWhitelist***
- ***uint32_t StartAddress***
- ***uint32_t EndAddress***

Field Documentation

- ***uint32_t RISAF_BaseRegionConfig_t::Filtering***
  One of ***RIF_RISAF_FILTERING_MODE***
- ***uint32_t RISAF_BaseRegionConfig_t::Secure***
  One of ***RIF_SEC_PRIV***
- ***uint32_t RISAF_BaseRegionConfig_t::PrivWhitelist***
  A combination of ***RIF_COMPARTMENT_ID***
- ***uint32_t RISAF_BaseRegionConfig_t::ReadWhitelist***
  A combination of ***RIF_COMPARTMENT_ID***
- ***uint32_t RISAF_BaseRegionConfig_t::WriteWhitelist***
  A combination of ***RIF_COMPARTMENT_ID***
- ***uint32_t RISAF_BaseRegionConfig_t::StartAddress***
  Base region start, address offset to the base address of the protected address space
- ***uint32_t RISAF_BaseRegionConfig_t::EndAddress***
  Base region end, address offset to the base address of the protected address space

58.1.3 RISAF_SubRegionConfig_t

RISAF_SubRegionConfig_t is defined in the stm32n6xx_hal_rif.h

Data Fields

- ***uint32_t Filtering***
- ***uint32_t CID***
- ***uint32_t SecPriv***
- ***uint32_t ReadWrite***
- ***uint32_t Lock***
- ***uint32_t StartAddress***

- ***uint32_t EndAddress***

Field Documentation

- ***uint32_t RISAF_SubRegionConfig_t::Filtering***
  One of ***RIF_RISAF_FILTERING_MODE***
- ***uint32_t RISAF_SubRegionConfig_t::CID***
  One of ***RIF_COMPARTMENT_ID***
- ***uint32_t RISAF_SubRegionConfig_t::SecPriv***
  A combination of ***RIF_SEC_PRIV***
- ***uint32_t RISAF_SubRegionConfig_t::ReadWrite***
  A combination of ***RIF_RISAF_READ_WRITE***
- ***uint32_t RISAF_SubRegionConfig_t::Lock***
  One of ***RIF_LOCK_STATE***
- ***uint32_t RISAF_SubRegionConfig_t::StartAddress***
  Subregion start, address offset to the base address of the protected address space
- ***uint32_t RISAF_SubRegionConfig_t::EndAddress***
  Subregion end, address offset to the base address of the protected address space

58.1.4 RISAF_DelegationConfig_t

RISAF_DelegationConfig_t is defined in the stm32n6xx_hal_rif.h

Data Fields

- ***uint32_t Delegation***
- ***uint32_t DelegatedCID***

Field Documentation

- ***uint32_t RISAF_DelegationConfig_t::Delegation***
  Enable or Disable
- ***uint32_t RISAF_DelegationConfig_t::DelegatedCID***
  One of ***RIF_COMPARTMENT_ID***

58.1.5 RISAF_IllegalAccessData_t

RISAF_IllegalAccessData_t is defined in the stm32n6xx_hal_rif.h

Data Fields

- ***uint32_t CID***
- ***uint32_t SecPriv***
- ***uint32_t AccessType***
- ***uint32_t Address***

Field Documentation

- ***uint32_t RISAF_IllegalAccessData_t::CID***
  One of ***RIF_COMPARTMENT_ID***
- ***uint32_t RISAF_IllegalAccessData_t::SecPriv***
  A combination of ***RIF_SEC_PRIV***
- ***uint32_t RISAF_IllegalAccessData_t::AccessType***
  One of ***RIF_ACCESS_TYPE***
- ***uint32_t RISAF_IllegalAccessData_t::Address***

58.1.6 RISAF_IllegalAccess_t

RISAF_IllegalAccess_t is defined in the stm32n6xx_hal_rif.h

Data Fields

- ***uint32_t ErrorType***
- ***RISAF_IllegalAccessData_t Data***

Field Documentation

- ***uint32_t RISAF_IllegalAccess_t::ErrorType***
  One of ***RIF_RISAF_ILLEGAL_ACCESS***

- ***RISAF_IllegalAccessData_t RISAF_IllegalAccess_t::Data***

58.2 RIF Firmware driver API description

The following section lists the various functions of the RIF library.

58.2.1 How to use this driver

- Resource Isolation framework (RIF) is a comprehensive set of hardware blocks designed to enforce and manage isolation of STM32 hardware resources like memory and peripherals.
- Resource Isolation Framework (RIF) composed of the following sub-blocks:
  - RIMC: Resource Isolation Master Controller This sub-block assigns all non RIF-aware bus master to one security domain by setting secure, privileged and compartment information on the system bus. Note that if non-secure software is permitted to configure one of these peripherals, then this setting is ignored and all accesses initiated by the peripheral are forced to be non-secure.
  - RISC: Resource Isolation Slave Controller This sub-block assigns all non-RIF aware peripherals to zero, one or any security domains (secure, privilege, compartment).
  - RISAF: Resource Isolation Slave unit for Address space (Full version) This sub-block assigns memory regions and subregions to one or more security domains (secure, privilege, compartment).
  - IAC: Illegal Access Controller This sub-block centralizes detection of RIF-related accesses, managed by a secure application. Supported illegal accesses are: secure, privileged.
- All the setter functions are protected under the CPU_AS_TRUSTED_DOMAIN directive. It ensures that only the trusted domain can manage the critical configurations of the RIF. This directive is defined in the CMSIS file.

Locking RIF components configuration

1. Lock and get the lock configuration of RIMC registers until next reset with HAL_RIF_RIMC_Lock / HAL_RIF_RIMC_GetLock.
2. Lock and get the lock configuration of RISC registers until next reset with HAL_RIF_RISC_Lock / HAL_RIF_RISC_GetLock.
3. Lock and get the lock configuration of slave resources registers until next reset with HAL_RIF_RISC_SlaveConfigLock / HAL_RIF_RISC_GetSlaveConfigLock.
4. Lock and get the lock configuration of RISAF registers until next reset with HAL_RIF_RISAF_Lock / HAL_RIF_RISAF_GetLock.

Context-related functions

1. Set or get Debugger Access Port Compartment ID with HAL_RIF_RIMC_SetDebugAccessPortCID / HAL_RIF_RIMC_GetDebugAccessPortCID functions.
2. Check the Debug Domain and Global Debug Domain status with HAL_RIF_RIMC_GetDebugStatus. #if defined(GENERATOR_RIF_FEATURE_ENCRYPTION_IMPLEMENTED)

Encryption-related functions

1. Read of write RISAF encryption keys and related flags. #endif

Configuration of master-related attributes

1. Set or get the secure, privilege and compartment configuration of a bus master thanks to the HAL_RIF_RIMC_ConfigMasterAttributes / HAL_RIF_RIMC_GetConfigMasterAttributes functions.
2. Lock and get lock status the bus master attributes configuration until next system reset using HAL_RIF_RIMC_Lock.

Configuration of slave-related attributes

1. Set or get the secure and privilege configuration of a bus slave thanks to the HAL_RIF_RISC_SetSlaveSecureAttributes / HAL_RIF_RISC_GetSlaveSecureAttributes functions.
2. Lock and get lock status each bus slave attributes configuration using HAL_RIF_RISC_SlaveConfigLock or lock the whole RISC configuration with HAL_RIF_RISC_Lock until next system reset.

Configuration of RISAF attributes

1. Set or get a base region configuration with HAL_RIF_RISAF_ConfigBaseRegion / HAL_RIF_RISAF_GetConfigBaseRegion.
2. Set or get a subregion configuration with HAL_RIF_RISAF_ConfigSubRegion / HAL_RIF_RISAF_GetConfigSubRegion.
3. Delegate a subregion configuration to another CID with HAL_RIF_RISAF_ConfigSubRegionDelegation.
4. Get the data from an illegal access to a protected memory region with HAL_RIF_RISAF_GetIllegalAccess.

IAC related functions

1. Enable or disable peripheral illegal access detection with HAL_RIF_IAC_EnableIT / HAL_RIF_IAC_DisableIT.
2. Retrieve an illegal access flag from a designated peripheral with HAL_RIF_IAC_GetFlag or clear it with HAL_RIF_IAC_ClearFlag.
3. Illegal access interrupt service routines are served by HAL_RIF_IRQHandler() and user can add his own code using HAL_RIF_ILA_Callback().

58.2.2 Detailed description of functions

HAL_RIF_RIMC_Lock

Function name

void HAL_RIF_RIMC_Lock (void)

Function description

HAL_RIF_RIMC_GetLock

Function name

uint32_t HAL_RIF_RIMC_GetLock (void)

Function description

HAL_RIF_RIMC_SetDebugAccessPortCID

Function name

void HAL_RIF_RIMC_SetDebugAccessPortCID (uint32_t CID)

Function description

HAL_RIF_RIMC_GetDebugAccessPortCID

Function name

uint32_t HAL_RIF_RIMC_GetDebugAccessPortCID (void)

Function description

HAL_RIF_RIMC_ConfigMasterAttributes

Function name

void HAL_RIF_RIMC_ConfigMasterAttributes (uint32_t MasterId, const RIMC_MasterConfig_t * pConfig)

Function description

HAL_RIF_RIMC_GetConfigMasterAttributes

Function name

void HAL_RIF_RIMC_GetConfigMasterAttributes (uint32_t MasterId, RIMC_MasterConfig_t * pConfig)

Function description

HAL_RIF_RISC_Lock

Function name

void HAL_RIF_RISC_Lock (void)

Function description

HAL_RIF_RISC_GetLock

Function name

uint32_t HAL_RIF_RISC_GetLock (void)

Function description

HAL_RIF_RISC_SetSlaveSecureAttributes

Function name

void HAL_RIF_RISC_SetSlaveSecureAttributes (uint32_t PeriphId, uint32_t SecPriv)

Function description

HAL_RIF_RISC_GetSlaveSecureAttributes

Function name

uint32_t HAL_RIF_RISC_GetSlaveSecureAttributes (uint32_t PeriphId)

Function description

HAL_RIF_RISC_SlaveConfigLock

Function name

void HAL_RIF_RISC_SlaveConfigLock (uint32_t PeriphId)

Function description

HAL_RIF_RISC_GetSlaveConfigLock

Function name

uint32_t HAL_RIF_RISC_GetSlaveConfigLock (uint32_t PeriphId)

Function description

HAL_RIF_RISAF_Lock

Function name

void HAL_RIF_RISAF_Lock (RISAF_TypeDef * RISAFx)

Function description

HAL_RIF_RISAF_GetLock

Function name

uint32_t HAL_RIF_RISAF_GetLock (const RISAF_TypeDef * RISAFx)

Function description

HAL_RIF_RISAF_ConfigBaseRegion

Function name

void HAL_RIF_RISAF_ConfigBaseRegion (RISAF_TypeDef * RISAFx, uint32_t Region, const RISAF_BaseRegionConfig_t * pConfig)

Function description

HAL_RIF_RISAF_GetConfigBaseRegion

Function name

void HAL_RIF_RISAF_GetConfigBaseRegion (const RISAF_TypeDef * RISAFx, uint32_t Region, RISAF_BaseRegionConfig_t * pConfig)

Function description

HAL_RIF_RISAF_ConfigSubRegionDelegation

Function name

void HAL_RIF_RISAF_ConfigSubRegionDelegation (RISAF_TypeDef * RISAFx, uint32_t Region, uint32_t SubRegion, const RISAF_DelegationConfig_t * pConfig)

Function description

HAL_RIF_RISAF_GetConfigSubRegionDelegation

Function name

void HAL_RIF_RISAF_GetConfigSubRegionDelegation (const RISAF_TypeDef * RISAFx, uint32_t Region, uint32_t SubRegion, RISAF_DelegationConfig_t * pConfig)

Function description

HAL_RIF_RISAF_ConfigSubRegion

Function name

void HAL_RIF_RISAF_ConfigSubRegion (RISAF_TypeDef * RISAFx, uint32_t Region, uint32_t SubRegion, const RISAF_SubRegionConfig_t * pConfig)

Function description

HAL_RIF_RISAF_GetConfigSubRegion

Function name

void HAL_RIF_RISAF_GetConfigSubRegion (const RISAF_TypeDef * RISAFx, uint32_t Region, uint32_t SubRegion, RISAF_SubRegionConfig_t * pConfig)

Function description

HAL_RIF_RISAF_GetIllegalAccess

Function name

void HAL_RIF_RISAF_GetIllegalAccess (RISAF_TypeDef * RISAFx, RISAF_IllegalAccess_t * IllegalAccess)

Function description

HAL_RIF_IAC_EnableIT

Function name

void HAL_RIF_IAC_EnableIT (uint32_t PeriphId)

Function description

HAL_RIF_IAC_DisableIT

Function name

void HAL_RIF_IAC_DisableIT (uint32_t PeriphId)

Function description

HAL_RIF_IAC_GetFlag

Function name

uint32_t HAL_RIF_IAC_GetFlag (uint32_t PeriphId)

Function description

HAL_RIF_IAC_ClearFlag

Function name

void HAL_RIF_IAC_ClearFlag (uint32_t PeriphId)

Function description

HAL_RIF_IRQHandler

Function name

void HAL_RIF_IRQHandler (void)

Function description

HAL_RIF_ILA_Callback

Function name

void HAL_RIF_ILA_Callback (uint32_t PeriphId)

Function description

58.3 RIF Firmware driver defines

The following section lists the various define and macros of the module.

58.3.1 RIF

RIF

RIF Access type definitions

RIF_ACCTYPE_READ_FETCH

RIF_ACCTYPE_WRITE

RIF-aware Peripheral index

RIF_AWARE_PERIPH_INDEX_CM55

RIF_AWARE_PERIPH_INDEX_EXTI

RIF_AWARE_PERIPH_INDEX_GPDMA1

RIF_AWARE_PERIPH_INDEX_HPDMA1

RIF_AWARE_PERIPH_INDEX_RTC

RIF_AWARE_PERIPH_INDEX_TAMP

RIF_AWARE_PERIPH_INDEX_BSEC

RIF_AWARE_PERIPH_INDEX_RCC

RIF_AWARE_PERIPH_INDEX_PWR

RIF_AWARE_PERIPH_INDEX_IAC

RIF_AWARE_PERIPH_INDEX_RISAF1

RIF_AWARE_PERIPH_INDEX_RISAF2

RIF_AWARE_PERIPH_INDEX_RISAF3

RIF_AWARE_PERIPH_INDEX_RISAF4

RIF_AWARE_PERIPH_INDEX_RISAF5

RIF_AWARE_PERIPH_INDEX_RISAF6

RIF_AWARE_PERIPH_INDEX_RISAF7

RIF_AWARE_PERIPH_INDEX_RISAF8

RIF_AWARE_PERIPH_INDEX_RISAF9

RIF_AWARE_PERIPH_INDEX_RISAF11

RIF_AWARE_PERIPH_INDEX_RISAF12

RIF_AWARE_PERIPH_INDEX_RISAF13

RIF_AWARE_PERIPH_INDEX_RISAF14

RIF_AWARE_PERIPH_INDEX_RISAF15

RIF_AWARE_PERIPH_INDEX_RISAF21

RIF_AWARE_PERIPH_INDEX_RISAF22

RIF_AWARE_PERIPH_INDEX_RISAF23

RIF_AWARE_PERIPH_INDEX_RIFSC

RIF CID number definition

RIF_CID_NONE

RIF_CID_0

RIF_CID_1

RIF_CID_2

RIF_CID_3

RIF_CID_4

RIF_CID_5

RIF_CID_6

RIF_CID_7

RIF Lock states definitions

RIF_LOCK_DISABLE

RIF_LOCK_ENABLE

RIF register masks

RIF_CID_MASK

RIF_ATTRIBUTE_MASK

RISAF_READ_WRITE_MASK

RIF Master index

RIF_MASTER_INDEX_ETR

RIF_MASTER_INDEX_NPU

RIF_MASTER_INDEX_SDMMC1

RIF_MASTER_INDEX_SDMMC2

RIF_MASTER_INDEX_OTG1

RIF_MASTER_INDEX_OTG2

RIF_MASTER_INDEX_ETH1

RIF_MASTER_INDEX_GPU2D

RIF_MASTER_INDEX_DMA2D

RIF_MASTER_INDEX_DCMIPP

RIF_MASTER_INDEX_LTDC1

RIF_MASTER_INDEX_LTDC2

RIF_MASTER_INDEX_VENC

RIF RISUP Peripheral index

RIF_RISC_PERIPH_INDEX_SPI1

RIF_RISC_PERIPH_INDEX_SPI2

RIF_RISC_PERIPH_INDEX_SPI3

RIF_RISC_PERIPH_INDEX_SPI4

RIF_RISC_PERIPH_INDEX_SPI5

RIF_RISC_PERIPH_INDEX_SPI6

RIF_RISC_PERIPH_INDEX_SAI1

RIF_RISC_PERIPH_INDEX_SAI2

RIF_RISC_PERIPH_INDEX_I2C1

RIF_RISC_PERIPH_INDEX_I2C2

RIF_RISC_PERIPH_INDEX_I2C3

RIF_RISC_PERIPH_INDEX_I2C4

RIF_RISC_PERIPH_INDEX_I3C1

RIF_RISC_PERIPH_INDEX_I3C2

RIF_RISC_PERIPH_INDEX_USART1

RIF_RISC_PERIPH_INDEX_USART2

RIF_RISC_PERIPH_INDEX_USART3

RIF_RISC_PERIPH_INDEX_UART4

RIF_RISC_PERIPH_INDEX_UART5

RIF_RISC_PERIPH_INDEX_USART6

RIF_RISC_PERIPH_INDEX_UART7

RIF_RISC_PERIPH_INDEX_UART8

RIF_RISC_PERIPH_INDEX_UART9

RIF_RISC_PERIPH_INDEX_USART10

RIF_RISC_PERIPH_INDEX_LPUART1

RIF_RISC_PERIPH_INDEX_FDCAN1

RIF_RISC_PERIPH_INDEX_TIM1

RIF_RISC_PERIPH_INDEX_TIM2

RIF_RISC_PERIPH_INDEX_TIM3

RIF_RISC_PERIPH_INDEX_TIM4

RIF_RISC_PERIPH_INDEX_TIM5

RIF_RISC_PERIPH_INDEX_TIM6

RIF_RISC_PERIPH_INDEX_TIM7

RIF_RISC_PERIPH_INDEX_TIM8

RIF_RISC_PERIPH_INDEX_TIM9

RIF_RISC_PERIPH_INDEX_TIM10

RIF_RISC_PERIPH_INDEX_TIM11

RIF_RISC_PERIPH_INDEX_TIM12

RIF_RISC_PERIPH_INDEX_TIM13

RIF_RISC_PERIPH_INDEX_TIM14

RIF_RISC_PERIPH_INDEX_TIM15

RIF_RISC_PERIPH_INDEX_TIM16

RIF_RISC_PERIPH_INDEX_TIM17

RIF_RISC_PERIPH_INDEX_TIM18

RIF_RISC_PERIPH_INDEX_GFXTIM

RIF_RISC_PERIPH_INDEX_LPTIM1

RIF_RISC_PERIPH_INDEX_LPTIM2

RIF_RISC_PERIPH_INDEX_LPTIM3

RIF_RISC_PERIPH_INDEX_LPTIM4

RIF_RISC_PERIPH_INDEX_LPTIM5

RIF_RISC_PERIPH_INDEX_ADF1

RIF_RISC_PERIPH_INDEX_MDF1

RIF_RISC_PERIPH_INDEX_SDMMC1

RIF_RISC_PERIPH_INDEX_SDMMC2

RIF_RISC_PERIPH_INDEX_MDIOS

RIF_RISC_PERIPH_INDEX_OTG1HS

RIF_RISC_PERIPH_INDEX_OTG2HS

RIF_RISC_PERIPH_INDEX_UCPD1

RIF_RISC_PERIPH_INDEX_ETH1

RIF_RISC_PERIPH_INDEX_SPDIFRX

RIF_RISC_PERIPH_INDEX_SYSCFG

RIF_RISC_PERIPH_INDEX_ADC12

RIF_RISC_PERIPH_INDEX_VREFBUF

RIF_RISC_PERIPH_INDEX_CRC

RIF_RISC_PERIPH_INDEX_IWDG

RIF_RISC_PERIPH_INDEX_WWDG

RIF_RISC_PERIPH_INDEX_RNG

RIF_RISC_PERIPH_INDEX_PKA

RIF_RISC_PERIPH_INDEX_SAES

RIF_RISC_PERIPH_INDEX_HASH

RIF_RISC_PERIPH_INDEX_CRYP

RIF_RISC_PERIPH_INDEX_MCE1

RIF_RISC_PERIPH_INDEX_MCE2

RIF_RISC_PERIPH_INDEX_MCE3

RIF_RISC_PERIPH_INDEX_MCE4

RIF_RISC_PERIPH_INDEX_XSPI1

RIF_RISC_PERIPH_INDEX_XSPI2

RIF_RISC_PERIPH_INDEX_XSPI3

RIF_RISC_PERIPH_INDEX_XSPIM

RIF_RISC_PERIPH_INDEX_FMC

RIF_RISC_PERIPH_INDEX_CSI

RIF_RISC_PERIPH_INDEX_DCMIPP

RIF_RISC_PERIPH_INDEX_DCMI

RIF_RISC_PERIPH_INDEX_JPEG

RIF_RISC_PERIPH_INDEX_VENC

RIF_RISC_PERIPH_INDEX_ICACHE

RIF_RISC_PERIPH_INDEX_GPU2D

RIF_RISC_PERIPH_INDEX_GFXMMU

RIF_RISC_PERIPH_INDEX_DMA2D

RIF_RISC_PERIPH_INDEX_LTDC

RIF_RISC_PERIPH_INDEX_LTDCL1

RIF_RISC_PERIPH_INDEX_LTDCL2

RIF_RISC_PERIPH_INDEX_NPU

RIF RCC Security Control Peripheral index

RIF_RCC_PERIPH_INDEX_GPDMA1

RIF_RCC_PERIPH_INDEX_HPDMA1

RIF_RCC_PERIPH_INDEX_RTC

RIF_RCC_PERIPH_INDEX_AXISRAM1

RIF_RCC_PERIPH_INDEX_AXISRAM2

RIF_RCC_PERIPH_INDEX_FLEXRAM

RIF_RCC_PERIPH_INDEX_CACHEAXIRAM

RIF_RCC_PERIPH_INDEX_VENCRAM

RIF_RCC_PERIPH_INDEX_CACHECONFIG

RIF_RCC_PERIPH_INDEX_AHBRAM1

RIF_RCC_PERIPH_INDEX_AHBRAM2

RIF_RCC_PERIPH_INDEX_BKPRAM

RIF_RCC_PERIPH_INDEX_GPIOA

RIF_RCC_PERIPH_INDEX_GPIOB

RIF_RCC_PERIPH_INDEX_GPIOC

RIF_RCC_PERIPH_INDEX_GPIOD

RIF_RCC_PERIPH_INDEX_GPIOE

RIF_RCC_PERIPH_INDEX_GPIOF

RIF_RCC_PERIPH_INDEX_GPIOG

RIF_RCC_PERIPH_INDEX_GPIOH

RIF_RCC_PERIPH_INDEX_GPION

RIF_RCC_PERIPH_INDEX_GPIOO

RIF_RCC_PERIPH_INDEX_GPIOP

RIF_RCC_PERIPH_INDEX_GPIOQ

RIF_RCC_PERIPH_INDEX_DTS

RIF_RCC_PERIPH_INDEX_MCO1

RIF_RCC_PERIPH_INDEX_MCO2

RIF_RCC_PERIPH_INDEX_NPURAM0

RIF_RCC_PERIPH_INDEX_NPURAM1

RIF_RCC_PERIPH_INDEX_NPURAM2

RIF_RCC_PERIPH_INDEX_NPURAM3

RIF_RCC_PERIPH_INDEX_XSPIPHYCOMP

RIF_RCC_PERIPH_INDEX_XSPIPHY1

RIF_RCC_PERIPH_INDEX_XSPIPHY2

RIF_RCC_PERIPH_INDEX_HDP

RIF_RCC_PERIPH_INDEX_RAMCFG

RIF register index definition

RIF_PERIPH_REG_SHIFT

RIF_PERIPH_REG

RIF_PERIPH_REG0

RIF_PERIPH_REG1

RIF_PERIPH_REG2

RIF_PERIPH_REG3

RIF_PERIPH_REG4

RIF_PERIPH_REG5

RIF_PERIPH_BIT_POSITION

RISAF address space sizes definitions

RISAF1_LIMIT_ADDRESS_SPACE_SIZE

RISAF2_LIMIT_ADDRESS_SPACE_SIZE

RISAF3_LIMIT_ADDRESS_SPACE_SIZE

RISAF4_LIMIT_ADDRESS_SPACE_SIZE

RISAF5_LIMIT_ADDRESS_SPACE_SIZE

RISAF6_LIMIT_ADDRESS_SPACE_SIZE

RISAF7_LIMIT_ADDRESS_SPACE_SIZE

RISAF8_LIMIT_ADDRESS_SPACE_SIZE

RISAF9_LIMIT_ADDRESS_SPACE_SIZE

RISAF11_LIMIT_ADDRESS_SPACE_SIZE

RISAF12_LIMIT_ADDRESS_SPACE_SIZE

RISAF13_LIMIT_ADDRESS_SPACE_SIZE

RISAF14_LIMIT_ADDRESS_SPACE_SIZE

RISAF15_LIMIT_ADDRESS_SPACE_SIZE

RISAF21_LIMIT_ADDRESS_SPACE_SIZE

RISAF22_LIMIT_ADDRESS_SPACE_SIZE

RISAF23_LIMIT_ADDRESS_SPACE_SIZE

RISAF Delegation mode definitions

RISAF_DELEGATION_DISABLE

RISAF_DELEGATION_ENABLE

RISAF Filtering mode definitions

RISAF_FILTER_DISABLE

RISAF_FILTER_ENABLE

RISAF granularity definitions

RISAF1_GRANULARITY

RISAF2_GRANULARITY

RISAF3_GRANULARITY

RISAF4_GRANULARITY

RISAF5_GRANULARITY

RISAF6_GRANULARITY

RISAF7_GRANULARITY

RISAF8_GRANULARITY

RISAF9_GRANULARITY

RISAF11_GRANULARITY

RISAF12_GRANULARITY

RISAF13_GRANULARITY

RISAF14_GRANULARITY

RISAF15_GRANULARITY

RISAF21_GRANULARITY

RISAF22_GRANULARITY

RISAF23_GRANULARITY

RISAF Illegal Access definitions

RISAF_ILLEGAL_ACCESS_NONE

RISAF_ILLEGAL_CONFIGURATION_ACCESS

RISAF_ILLEGAL_ACCESS

RISAF Read/Write definitions

RISAF_READ_DISABLE

RISAF_READ_ENABLE

RISAF_WRITE_DISABLE

RISAF_WRITE_ENABLE

RISAF Base region definitions

RISAF_REGION_1

RISAF_REGION_2

RISAF_REGION_3

RISAF_REGION_4

RISAF_REGION_5

RISAF_REGION_6

RISAF_REGION_7

RISAF_REGION_8

RISAF_REGION_9

RISAF_REGION_10

RISAF_REGION_11

RISAF_REGION_12

RISAF_REGION_13

RISAF_REGION_14

RISAF_REGION_15

RIF subregion definitions

RISAF_SUBREGION_A

RISAF_SUBREGION_B

RIF Secure Privileged attributes definitions

RIF_ATTRIBUTE_NSEC

RIF_ATTRIBUTE_SEC

RIF_ATTRIBUTE_NPRIV

RIF_ATTRIBUTE_PRIV

59 HAL RNG Generic Driver

59.1 RNG Firmware driver registers structures

59.1.1 RNG_InitTypeDef

RNG_InitTypeDef is defined in the stm32n6xx_hal_rng.h

Data Fields

- ***uint32_t ClockErrorDetection***

Field Documentation

- ***uint32_t RNG_InitTypeDef::ClockErrorDetection***
  CED Clock error detection

59.1.2 __RNG_HandleTypeDef

__RNG_HandleTypeDef is defined in the stm32n6xx_hal_rng.h

Data Fields

- ***RNG_TypeDef * Instance***
- ***RNG_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_RNG_StateTypeDef State***
- ***__IO uint32_t ErrorCode***
- ***uint32_t RandomNumber***
- ***void(* ReadyDataCallback***
- ***void(* ErrorCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***RNG_TypeDef* __RNG_HandleTypeDef::Instance***
  Register base address
- ***RNG_InitTypeDef __RNG_HandleTypeDef::Init***
  RNG configuration parameters
- ***HAL_LockTypeDef __RNG_HandleTypeDef::Lock***
  RNG locking object
- ***__IO HAL_RNG_StateTypeDef __RNG_HandleTypeDef::State***
  RNG communication state
- ***__IO uint32_t __RNG_HandleTypeDef::ErrorCode***
  RNG Error code
- ***uint32_t __RNG_HandleTypeDef::RandomNumber***
  Last Generated RNG Data
- ***void(* __RNG_HandleTypeDef::ReadyDataCallback)(struct __RNG_HandleTypeDef *hrng, uint32_t random32bit)***
  RNG Data Ready Callback
- ***void(* __RNG_HandleTypeDef::ErrorCallback)(struct __RNG_HandleTypeDef *hrng)***
  RNG Error Callback
- ***void(* __RNG_HandleTypeDef::MspInitCallback)(struct __RNG_HandleTypeDef *hrng)***
  RNG Msp Init callback
- ***void(* __RNG_HandleTypeDef::MspDeInitCallback)(struct __RNG_HandleTypeDef *hrng)***
  RNG Msp DeInit callback

59.2 RNG Firmware driver API description

The following section lists the various functions of the RNG library.

59.2.1 How to use this driver

The RNG HAL driver can be used as follows:

1. Enable the RNG controller clock using __HAL_RCC_RNG_CLK_ENABLE() macro in HAL_RNG_MspInit().
2. Activate the RNG peripheral using HAL_RNG_Init() function.
3. Wait until the 32 bit Random Number Generator contains a valid random data using (polling/interrupt) mode.
4. Get the 32 bit random number using HAL_RNG_GenerateRandomNumber() function.

59.2.2 Callback registration

The compilation define USE_HAL_RNG_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks.

Use Function HAL_RNG_RegisterCallback() to register a user callback. Function HAL_RNG_RegisterCallback() allows to register following callbacks:

- ErrorCallback : RNG Error Callback.
- MspInitCallback : RNG MspInit.
- MspDeInitCallback : RNG MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

Use function HAL_RNG_UnRegisterCallback() to reset a callback to the default weak (overridden) function. HAL_RNG_UnRegisterCallback() takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks:

- ErrorCallback : RNG Error Callback.
- MspInitCallback : RNG MspInit.
- MspDeInitCallback : RNG MspDeInit.

For specific callback ReadyDataCallback, use dedicated register callbacks: respectively HAL_RNG_RegisterReadyDataCallback() , HAL_RNG_UnRegisterReadyDataCallback().

By default, after the HAL_RNG_Init() and when the state is HAL_RNG_STATE_RESET all callbacks are set to the corresponding weak (overridden) functions: example HAL_RNG_ErrorCallback(). Exception done for MspInit and MspDeInit functions that are respectively reset to the legacy weak (overridden) functions in the HAL_RNG_Init() and HAL_RNG_DeInit() only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the HAL_RNG_Init() and HAL_RNG_DeInit() keep and use the user MspInit/MspDeInit callbacks (registered beforehand).

Callbacks can be registered/unregistered in HAL_RNG_STATE_READY state only. Exception done MspInit/MspDeInit that can be registered/unregistered in HAL_RNG_STATE_READY or HAL_RNG_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_RNG_RegisterCallback() before calling HAL_RNG_DeInit() or HAL_RNG_Init() function.

When The compilation define USE_HAL_RNG_REGISTER_CALLBACKS is set to 0 or not defined, the callback registration feature is not available and weak (overridden) callbacks are used.

59.2.3 Detailed description of functions

HAL_RNG_Init

Function name

HAL_StatusTypeDef HAL_RNG_Init (RNG_HandleTypeDef * hrng)

Function description

HAL_RNG_DeInit

Function name

HAL_StatusTypeDef HAL_RNG_DeInit (RNG_HandleTypeDef * hrng)

Function description

HAL_RNG_MspInit

Function name

void HAL_RNG_MspInit (RNG_HandleTypeDef * hrng)

Function description

HAL_RNG_MspDeInit

Function name

void HAL_RNG_MspDeInit (RNG_HandleTypeDef * hrng)

Function description

HAL_RNG_RegisterCallback

Function name

HAL_StatusTypeDef HAL_RNG_RegisterCallback (RNG_HandleTypeDef * hrng, HAL_RNG_CallbackIDTypeDef CallbackID, pRNG_CallbackTypeDef pCallback)

Function description

HAL_RNG_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_RNG_UnRegisterCallback (RNG_HandleTypeDef * hrng, HAL_RNG_CallbackIDTypeDef CallbackID)

Function description

HAL_RNG_RegisterReadyDataCallback

Function name

HAL_StatusTypeDef HAL_RNG_RegisterReadyDataCallback (RNG_HandleTypeDef * hrng, pRNG_ReadyDataCallbackTypeDef pCallback)

Function description

HAL_RNG_UnRegisterReadyDataCallback

Function name

HAL_StatusTypeDef HAL_RNG_UnRegisterReadyDataCallback (RNG_HandleTypeDef * hrng)

Function description

HAL_RNG_GenerateRandomNumber

Function name

HAL_StatusTypeDef HAL_RNG_GenerateRandomNumber (RNG_HandleTypeDef * hrng, uint32_t * random32bit)

Function description

HAL_RNG_GenerateRandomNumber_IT

Function name

HAL_StatusTypeDef HAL_RNG_GenerateRandomNumber_IT (RNG_HandleTypeDef * hrng)

Function description

HAL_RNG_ReadLastRandomNumber

Function name

uint32_t HAL_RNG_ReadLastRandomNumber (const RNG_HandleTypeDef * hrng)

Function description

HAL_RNG_IRQHandler

Function name

void HAL_RNG_IRQHandler (RNG_HandleTypeDef * hrng)

Function description

HAL_RNG_ErrorCallback

Function name

void HAL_RNG_ErrorCallback (RNG_HandleTypeDef * hrng)

Function description

HAL_RNG_ReadyDataCallback

Function name

void HAL_RNG_ReadyDataCallback (RNG_HandleTypeDef * hrng, uint32_t random32bit)

Function description

HAL_RNG_GetState

Function name

HAL_RNG_StateTypeDef HAL_RNG_GetState (const RNG_HandleTypeDef * hrng)

Function description

HAL_RNG_GetError

Function name

uint32_t HAL_RNG_GetError (const RNG_HandleTypeDef * hrng)

Function description

RNG_RecoverSeedError

Function name

HAL_StatusTypeDef RNG_RecoverSeedError (RNG_HandleTypeDef * hrng)

Function description

59.3 RNG Firmware driver defines

The following section lists the various define and macros of the module.

59.3.1 RNG

RNG

RNG Error Definition

HAL_RNG_ERROR_NONE

No error

HAL_RNG_ERROR_INVALID_CALLBACK

Invalid Callback error

HAL_RNG_ERROR_TIMEOUT

Timeout error

HAL_RNG_ERROR_BUSY

Busy error

HAL_RNG_ERROR_SEED

Seed error

HAL_RNG_ERROR_CLOCK

Clock error

RNG Interrupt definition

RNG_IT_DRDY

Data Ready interrupt

RNG_IT_CEI

Clock error interrupt

RNG_IT_SEI

Seed error interrupt

RNG Flag definition

RNG_FLAG_DRDY

Data ready

RNG_FLAG_CECS

Clock error current status

RNG_FLAG_SECS

Seed error current status

RNG Clock Error Detection

RNG_CED_ENABLE

Clock error detection Enabled

RNG_CED_DISABLE

Clock error detection Disabled

RNG Exported Macros

__HAL_RNG_RESET_HANDLE_STATE

Description:

- Reset RNG handle state.

Parameters:

- __HANDLE__: RNG Handle

Return value:

- None

__HAL_RNG_ENABLE

Description:

- Enables the RNG peripheral.

Parameters:

- __HANDLE__: RNG Handle

Return value:

- None

__HAL_RNG_DISABLE

Description:

- Disables the RNG peripheral.

Parameters:

- __HANDLE__: RNG Handle

Return value:

- None

__HAL_RNG_GET_FLAG

Description:

- Check the selected RNG flag status.

Parameters:

- __HANDLE__: RNG Handle
- __FLAG__: RNG flag This parameter can be one of the following values:
  - RNG_FLAG_DRDY: Data ready
  - RNG_FLAG_CECS: Clock error current status
  - RNG_FLAG_SECS: Seed error current status

Return value:

- The: new state of __FLAG__ (SET or RESET).

__HAL_RNG_CLEAR_FLAG

Description:

- Clears the selected RNG flag status.

Parameters:

- __HANDLE__: RNG handle
- __FLAG__: RNG flag to clear

Return value:

- None

Notes:

- WARNING: This is a dummy macro for HAL code alignment, flags RNG_FLAG_DRDY, RNG_FLAG_CECS and RNG_FLAG_SECS are read-only.

__HAL_RNG_ENABLE_IT

Description:

- Enables the RNG interrupts.

Parameters:

- __HANDLE__: RNG Handle

Return value:

- None

__HAL_RNG_DISABLE_IT

Description:

- Disables the RNG interrupts.

Parameters:

- __HANDLE__: RNG Handle

Return value:

- None

__HAL_RNG_GET_IT

Description:

- Checks whether the specified RNG interrupt has occurred or not.

Parameters:

- __HANDLE__: RNG Handle
- __INTERRUPT__: specifies the RNG interrupt status flag to check. This parameter can be one of the following values:
  - RNG_IT_DRDY: Data ready interrupt
  - RNG_IT_CEI: Clock error interrupt
  - RNG_IT_SEI: Seed error interrupt

Return value:

- The: new state of __INTERRUPT__ (SET or RESET).

__HAL_RNG_CLEAR_IT

Description:

- Clear the RNG interrupt status flags.

Parameters:

- __HANDLE__: RNG Handle
- __INTERRUPT__: specifies the RNG interrupt status flag to clear. This parameter can be one of the following values:
  - RNG_IT_CEI: Clock error interrupt
  - RNG_IT_SEI: Seed error interrupt

Return value:

- None

Notes:

- RNG_IT_DRDY flag is read-only, reading RNG_DR register automatically clears RNG_IT_DRDY.

60 HAL RNG Extension Driver

60.1 RNGEx Firmware driver registers structures

60.1.1 RNG_ConfigTypeDef

RNG_ConfigTypeDef is defined in the stm32n6xx_hal_rng_ex.h

Data Fields

- ***uint32_t Config1***
- ***uint32_t Config2***
- ***uint32_t Config3***
- ***uint32_t ClockDivider***
- ***uint32_t NistCompliance***
- ***uint32_t AutoReset***
- ***uint32_t HealthTest***

Field Documentation

- ***uint32_t RNG_ConfigTypeDef::Config1***
  Config1 must be a value between 0 and 0x3F
- ***uint32_t RNG_ConfigTypeDef::Config2***
  Config2 must be a value between 0 and 0x7
- ***uint32_t RNG_ConfigTypeDef::Config3***
  Config3 must be a value between 0 and 0xF
- ***uint32_t RNG_ConfigTypeDef::ClockDivider***
  Clock Divider factor.This parameter can be a value of ***RNGEx_Clock_Divider_Factor***
- ***uint32_t RNG_ConfigTypeDef::NistCompliance***
  NIST compliance.This parameter can be a value of ***RNGEx_NIST_Compliance***
- ***uint32_t RNG_ConfigTypeDef::AutoReset***
  automatic reset When a noise source error occurs value of ***RNGEx_Auto_Reset***
- ***uint32_t RNG_ConfigTypeDef::HealthTest***
  RNG health test control must be a value between 0x0FFCABFF and 0x00005200

60.2 RNGEx Firmware driver API description

The following section lists the various functions of the RNGEx library.

60.2.1 Detailed description of functions

HAL_RNGEx_SetConfig

Function name

HAL_StatusTypeDef HAL_RNGEx_SetConfig (RNG_HandleTypeDef * hrng, const RNG_ConfigTypeDef * pConf)

Function description

HAL_RNGEx_GetConfig

Function name

HAL_StatusTypeDef HAL_RNGEx_GetConfig (RNG_HandleTypeDef * hrng, RNG_ConfigTypeDef * pConf)

Function description

HAL_RNGEx_LockConfig

Function name

HAL_StatusTypeDef HAL_RNGEx_LockConfig (RNG_HandleTypeDef * hrng)

Function description

HAL_RNGEx_RecoverSeedError

Function name

HAL_StatusTypeDef HAL_RNGEx_RecoverSeedError (RNG_HandleTypeDef * hrng)

Function description

60.3 RNGEx Firmware driver defines

The following section lists the various define and macros of the module.

60.3.1 RNGEx

RNGEx

Auto Reset configuration

RNG_ARDIS_ENABLE

automatic reset after seed error

RNG_ARDIS_DISABLE

Disable automatic reset after seed error

Value used to configure an internal

RNG_CLKDIV_BY_1

No clock division

RNG_CLKDIV_BY_2

2 RNG clock cycles per internal RNG clock

RNG_CLKDIV_BY_4

4 RNG clock cycles per internal RNG clock

RNG_CLKDIV_BY_8

8 RNG clock cycles per internal RNG clock

RNG_CLKDIV_BY_16

16 RNG clock cycles per internal RNG clock

RNG_CLKDIV_BY_32

32 RNG clock cycles per internal RNG clock

RNG_CLKDIV_BY_64

64 RNG clock cycles per internal RNG clock

RNG_CLKDIV_BY_128

128 RNG clock cycles per internal RNG clock

RNG_CLKDIV_BY_256

256 RNG clock cycles per internal RNG clock

RNG_CLKDIV_BY_512

512 RNG clock cycles per internal RNG clock

RNG_CLKDIV_BY_1024

1024 RNG clock cycles per internal RNG clock

RNG_CLKDIV_BY_2048

2048 RNG clock cycles per internal RNG clock

RNG_CLKDIV_BY_4096

4096 RNG clock cycles per internal RNG clock

RNG_CLKDIV_BY_8192

8192 RNG clock cycles per internal RNG clock

RNG_CLKDIV_BY_16384

16384 RNG clock cycles per internal RNG clock

RNG_CLKDIV_BY_32768

32768 RNG clock cycles per internal RNG clock

NIST Compliance configuration

RNG_NIST_COMPLIANT

NIST compliant configuration

RNG_CUSTOM_NIST

Custom NIST configuration

61 HAL RTC Generic Driver

61.1 RTC Firmware driver registers structures

61.1.1 RTC_InitTypeDef

RTC_InitTypeDef is defined in the stm32n6xx_hal_rtc.h

Data Fields

- ***uint32_t HourFormat***
- ***uint32_t AsynchPrediv***
- ***uint32_t SynchPrediv***
- ***uint32_t OutPut***
- ***uint32_t OutPutRemap***
- ***uint32_t OutPutPolarity***
- ***uint32_t OutPutType***
- ***uint32_t OutPutPullUp***
- ***uint32_t BinMode***
- ***uint32_t BinMixBcdU***

Field Documentation

- ***uint32_t RTC_InitTypeDef::HourFormat***
  Specifies the RTC Hour Format. This parameter can be a value of ***RTC_Hour_Formats***
- ***uint32_t RTC_InitTypeDef::AsynchPrediv***
  Specifies the RTC Asynchronous Predivider value. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0x7F
- ***uint32_t RTC_InitTypeDef::SynchPrediv***
  Specifies the RTC Synchronous Predivider value. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0x7FFF
- ***uint32_t RTC_InitTypeDef::OutPut***
  Specifies which signal will be routed to the RTC output. This parameter can be a value of ***RTCEx_Output_selection_Definitions***
- ***uint32_t RTC_InitTypeDef::OutPutRemap***
  Specifies the remap for RTC output. This parameter can be a value of ***RTC_Output_ALARM_OUT_Remap***
- ***uint32_t RTC_InitTypeDef::OutPutPolarity***
  Specifies the polarity of the output signal. This parameter can be a value of ***RTC_Output_Polarity_Definitions***
- ***uint32_t RTC_InitTypeDef::OutPutType***
  Specifies the RTC Output Pin mode. This parameter can be a value of ***RTC_Output_Type_ALARM_OUT***
- ***uint32_t RTC_InitTypeDef::OutPutPullUp***
  Specifies the RTC Output Pull-Up mode. This parameter can be a value of ***RTC_Output_PullUp_ALARM_OUT***
- ***uint32_t RTC_InitTypeDef::BinMode***
  Specifies the RTC binary mode. This parameter can be a value of ***RTCEx_Binary_Mode***
- ***uint32_t RTC_InitTypeDef::BinMixBcdU***
  Specifies the BCD calendar update if and only if BinMode = RTC_BINARY_MIX. This parameter can be a value of ***RTCEx_Binary_mix_BCDU***

61.1.2 RTC_TimeTypeDef

RTC_TimeTypeDef is defined in the stm32n6xx_hal_rtc.h

Data Fields

- ***uint8_t Hours***
- ***uint8_t Minutes***
- ***uint8_t Seconds***

- ***uint8_t TimeFormat***
- ***uint32_t SubSeconds***
- ***uint32_t SecondFraction***
- ***uint32_t DayLightSaving***
- ***uint32_t StoreOperation***

Field Documentation

- ***uint8_t RTC_TimeTypeDef::Hours***
  Specifies the RTC Time Hour. This parameter must be a number between: Min_Data = 0 and Max_Data = 12 if the RTC_HOURFORMAT_12 is selected. This parameter must be a number between: Min_Data = 0 and Max_Data = 23 if the RTC_HOURFORMAT_24 is selected
- ***uint8_t RTC_TimeTypeDef::Minutes***
  Specifies the RTC Time Minutes. This parameter must be a number between Min_Data = 0 and Max_Data = 59
- ***uint8_t RTC_TimeTypeDef::Seconds***
  Specifies the RTC Time Seconds. This parameter must be a number between Min_Data = 0 and Max_Data = 59
- ***uint8_t RTC_TimeTypeDef::TimeFormat***
  Specifies the RTC AM/PM Time. This parameter can be a value of ***RTC_AM_PM_Definitions***
- ***uint32_t RTC_TimeTypeDef::SubSeconds***
  Specifies the RTC_SSR RTC Sub Second register content. This field is not used by HAL_RTC_SetTime. If the free running 32 bit counter is not activated (mode binary none)
  - This parameter corresponds to a time unit range between [0-1] Second with [1 Sec / SecondFraction +1] granularity else
  - This parameter corresponds to the free running 32 bit counter.
- ***uint32_t RTC_TimeTypeDef::SecondFraction***
  Specifies the range or granularity of Sub Second register content corresponding to Synchronous pre-scaler factor value (PREDIV_S) This parameter corresponds to a time unit range between [0-1] Second with [1 Sec / SecondFraction +1] granularity. This field will be used only by HAL_RTC_GetTime function
- ***uint32_t RTC_TimeTypeDef::DayLightSaving***
  This interface is deprecated. To manage Daylight Saving Time, please use HAL_RTC_DST_xxx functions
- ***uint32_t RTC_TimeTypeDef::StoreOperation***
  This interface is deprecated. To manage Daylight Saving Time, please use HAL_RTC_DST_xxx functions

61.1.3 RTC_DateTypeDef

RTC_DateTypeDef is defined in the stm32n6xx_hal_rtc.h

Data Fields

- ***uint8_t WeekDay***
- ***uint8_t Month***
- ***uint8_t Date***
- ***uint8_t Year***

Field Documentation

- ***uint8_t RTC_DateTypeDef::WeekDay***
  Specifies the RTC Date WeekDay. This parameter can be a value of ***RTC_WeekDay_Definitions***
- ***uint8_t RTC_DateTypeDef::Month***
  Specifies the RTC Date Month (in BCD format). This parameter can be a value of ***RTC_Month_Date_Definitions***
- ***uint8_t RTC_DateTypeDef::Date***
  Specifies the RTC Date. This parameter must be a number between Min_Data = 1 and Max_Data = 31
- ***uint8_t RTC_DateTypeDef::Year***
  Specifies the RTC Date Year. This parameter must be a number between Min_Data = 0 and Max_Data = 99

61.1.4 RTC_AlarmTypeDef

RTC_AlarmTypeDef is defined in the stm32n6xx_hal_rtc.h

Data Fields

- ***RTC_TimeTypeDef AlarmTime***
- ***uint32_t AlarmMask***
- ***uint32_t AlarmSubSecondMask***
- ***uint32_t BinaryAutoClr***
- ***uint32_t AlarmDateWeekDaySel***
- ***uint8_t AlarmDateWeekDay***
- ***uint32_t FlagAutoClr***
- ***uint32_t Alarm***

Field Documentation

- ***RTC_TimeTypeDef RTC_AlarmTypeDef::AlarmTime***
  Specifies the RTC Alarm Time members
- ***uint32_t RTC_AlarmTypeDef::AlarmMask***
  Specifies the RTC Alarm Masks. This parameter can be a value of ***RTC_AlarmMask_Definitions***
- ***uint32_t RTC_AlarmTypeDef::AlarmSubSecondMask***
  Specifies the RTC Alarm SubSeconds Masks. if Binary mode is RTC_BINARY_ONLY or is RTC_BINARY_MIX This parameter can be a value of ***RTCEx_Alarm_Sub_Seconds_binary_Masks_Definitions*** else if Binary mode is RTC_BINARY_NONE This parameter can be a value of ***RTC_Alarm_Sub_Seconds_BCD_Masks_Definitions***
- ***uint32_t RTC_AlarmTypeDef::BinaryAutoClr***
  Clear synchronously counter (RTC_SSR) on binary alarm. RTC_ALARMSUBSECONDBIN_AUTOCLR_YES must only be used if Binary mode is RTC_BINARY_ONLY This parameter can be a value of ***RTCEx_Alarm_Sub_Seconds_binary_Clear_Definitions***
- ***uint32_t RTC_AlarmTypeDef::AlarmDateWeekDaySel***
  Specifies the RTC Alarm is on Date or WeekDay. This parameter can be a value of ***RTC_AlarmDateWeekDay_Definitions***
- ***uint8_t RTC_AlarmTypeDef::AlarmDateWeekDay***
  Specifies the RTC Alarm Date/WeekDay. If the Alarm Date is selected, this parameter must be set to a value in the 1-31 range. If the Alarm WeekDay is selected, this parameter can be a value of ***RTC_WeekDay_Definitions***
- ***uint32_t RTC_AlarmTypeDef::FlagAutoClr***
  Specifies the alarm trigger generation. This feature is meaningful to avoid any RTC software execution after configuration. This parameter can be a value of ***RTC_ALARM_Flag_AutoClear_Definitions***
- ***uint32_t RTC_AlarmTypeDef::Alarm***
  Specifies the alarm. This parameter can be a value of ***RTC_Alarms_Definitions***

61.1.5 __RTC_HandleTypeDef

__RTC_HandleTypeDef is defined in the stm32n6xx_hal_rtc.h

Data Fields

- ***RTC_TypeDef * Instance***
- ***RTC_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_RTCStateTypeDef State***
- ***void(* AlarmAEventCallback***
- ***void(* AlarmBEventCallback***
- ***void(* TimeStampEventCallback***
- ***void(* WakeUpTimerEventCallback***
- ***void(* SSRUEventCallback***
- ***void(* Tamper1EventCallback***
- ***void(* Tamper2EventCallback***
- ***void(* Tamper3EventCallback***
- ***void(* Tamper4EventCallback***

- ***void(* Tamper5EventCallback***
- ***void(* Tamper6EventCallback***
- ***void(* Tamper7EventCallback***
- ***void(* Tamper8EventCallback***
- ***void(* InternalTamper1EventCallback***
- ***void(* InternalTamper2EventCallback***
- ***void(* InternalTamper3EventCallback***
- ***void(* InternalTamper4EventCallback***
- ***void(* InternalTamper5EventCallback***
- ***void(* InternalTamper6EventCallback***
- ***void(* InternalTamper7EventCallback***
- ***void(* InternalTamper8EventCallback***
- ***void(* InternalTamper9EventCallback***
- ***void(* InternalTamper11EventCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***RTC_TypeDef* __RTC_HandleTypeDef::Instance***
  Legacy register base address. Not used anymore, the driver directly uses cmsis base address
- ***RTC_InitTypeDef __RTC_HandleTypeDef::Init***
  RTC required parameters
- ***HAL_LockTypeDef __RTC_HandleTypeDef::Lock***
  RTC locking object
- ***__IO HAL_RTCStateTypeDef __RTC_HandleTypeDef::State***
  Time communication state
- ***void(* __RTC_HandleTypeDef::AlarmAEventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Alarm A Event callback
- ***void(* __RTC_HandleTypeDef::AlarmBEventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Alarm B Event callback
- ***void(* __RTC_HandleTypeDef::TimeStampEventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC TimeStamp Event callback
- ***void(* __RTC_HandleTypeDef::WakeUpTimerEventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC WakeUpTimer Event callback
- ***void(* __RTC_HandleTypeDef::SSRUEventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC SSRU Event callback
- ***void(* __RTC_HandleTypeDef::Tamper1EventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Tamper 1 Event callback
- ***void(* __RTC_HandleTypeDef::Tamper2EventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Tamper 2 Event callback
- ***void(* __RTC_HandleTypeDef::Tamper3EventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Tamper 3 Event callback
- ***void(* __RTC_HandleTypeDef::Tamper4EventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Tamper 4 Event callback
- ***void(* __RTC_HandleTypeDef::Tamper5EventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Tamper 5 Event callback
- ***void(* __RTC_HandleTypeDef::Tamper6EventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Tamper 6 Event callback
- ***void(* __RTC_HandleTypeDef::Tamper7EventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Tamper 7 Event callback
- ***void(* __RTC_HandleTypeDef::Tamper8EventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Tamper 8 Event callback
- ***void(* __RTC_HandleTypeDef::InternalTamper1EventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Internal Tamper 1 Event callback

- ***void(* __RTC_HandleTypeDef::InternalTamper2EventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Internal Tamper 2 Event callback
- ***void(* __RTC_HandleTypeDef::InternalTamper3EventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Internal Tamper 3 Event callback
- ***void(* __RTC_HandleTypeDef::InternalTamper4EventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Internal Tamper 4 Event callback
- ***void(* __RTC_HandleTypeDef::InternalTamper5EventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Internal Tamper 5 Event callback
- ***void(* __RTC_HandleTypeDef::InternalTamper6EventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Internal Tamper 6 Event callback
- ***void(* __RTC_HandleTypeDef::InternalTamper7EventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Internal Tamper 7 Event callback
- ***void(* __RTC_HandleTypeDef::InternalTamper8EventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Internal Tamper 8 Event callback
- ***void(* __RTC_HandleTypeDef::InternalTamper9EventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Internal Tamper 9 Event callback
- ***void(* __RTC_HandleTypeDef::InternalTamper11EventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Internal Tamper 11 Event callback
- ***void(* __RTC_HandleTypeDef::MspInitCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Msp Init callback
- ***void(* __RTC_HandleTypeDef::MspDeInitCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Msp DeInit callback

61.2 RTC Firmware driver API description

The following section lists the various functions of the RTC library.

61.2.1 RTC Operating Condition

The real-time clock (RTC) and the RTC backup registers can be powered from the VBAT voltage when the main VDD supply is powered off. To retain the content of the RTC backup registers and supply the RTC when VDD is turned off, VBAT pin can be connected to an optional standby voltage supplied by a battery or by another source.

61.2.2 Backup Domain Reset

The backup domain reset sets all RTC registers and the RCC_BDCR register to their reset values. A backup domain reset is generated when one of the following events occurs:

1. Software reset, triggered by setting the BDRST bit in the RCC Backup domain control register (RCC_BDCR).
2. VDD or VBAT power on, if both supplies have previously been powered off.
3. Tamper detection event resets all data backup registers.

61.2.3 Backup Domain Access

After reset, the backup domain (RTC registers and RTC backup data registers) is protected against possible unwanted write accesses.

To enable access to the RTC Domain and RTC registers, proceed as follows:

- Enable access to RTC domain using the HAL_PWR_EnableBkUpAccess() function.
- Call the function HAL_RCCEx_PeriphCLKConfig with RCC_PERIPHCLK_RTC for PeriphClockSelection and select RTCClockSelection (LSE, LSI or HSEdiv32)
- Enable RTC Clock using the __HAL_RCC_RTC_ENABLE() function.

61.2.4 How to use RTC Driver

- Enable the RTC domain access (see description in the section above).
- Configure the RTC Prescaler (Asynchronous and Synchronous) and RTC hour format using the HAL_RTC_Init() function.

Time and Date configuration

- To configure the RTC Calendar (Time and Date) use the HAL_RTC_SetTime() and HAL_RTC_SetDate() functions.
- To read the RTC Calendar, use the HAL_RTC_GetTime() and HAL_RTC_GetDate() functions.

Alarm configuration

- To configure the RTC Alarm use the HAL_RTC_SetAlarm() function. You can also configure the RTC Alarm with interrupt mode using the HAL_RTC_SetAlarm_IT() function.
- To read the RTC Alarm, use the HAL_RTC_GetAlarm() function.

61.2.5 RTC and low power modes

The MCU can be woken up from a low power mode by an RTC alternate function.

The RTC alternate functions are the RTC alarms (Alarm A and Alarm B), RTC wakeup, RTC tamper event detection and RTC time stamp event detection. These RTC alternate functions can wake up the system from the Stop and Standby low power modes.

The system can also wake up from low power modes without depending on an external interrupt (Auto-wakeup mode), by using the RTC alarm or the RTC wakeup events.

The RTC provides a programmable time base for waking up from the Stop or Standby mode at regular intervals. Wakeup from STOP and STANDBY modes is possible only when the RTC clock source is LSE or LSI.

Callback registration

61.2.6 Detailed description of functions

HAL_RTC_Init

Function name

HAL_StatusTypeDef HAL_RTC_Init (RTC_HandleTypeDef * hrtc)

Function description

HAL_RTC_DeInit

Function name

HAL_StatusTypeDef HAL_RTC_DeInit (RTC_HandleTypeDef * hrtc)

Function description

HAL_RTC_MspInit

Function name

void HAL_RTC_MspInit (RTC_HandleTypeDef * hrtc)

Function description

HAL_RTC_MspDeInit

Function name

void HAL_RTC_MspDeInit (RTC_HandleTypeDef * hrtc)

Function description

HAL_RTC_RegisterCallback

Function name

HAL_StatusTypeDef HAL_RTC_RegisterCallback (RTC_HandleTypeDef * hrtc, HAL_RTC_CallbackIDTypeDef CallbackID, pRTC_CallbackTypeDef pCallback)

Function description

HAL_RTC_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_RTC_UnRegisterCallback (RTC_HandleTypeDef * hrtc, HAL_RTC_CallbackIDTypeDef CallbackID)

Function description

HAL_RTC_SetTime

Function name

HAL_StatusTypeDef HAL_RTC_SetTime (RTC_HandleTypeDef * hrtc, RTC_TimeTypeDef * sTime, uint32_t Format)

Function description

HAL_RTC_GetTime

Function name

HAL_StatusTypeDef HAL_RTC_GetTime (const RTC_HandleTypeDef * hrtc, RTC_TimeTypeDef * sTime, uint32_t Format)

Function description

HAL_RTC_SetDate

Function name

HAL_StatusTypeDef HAL_RTC_SetDate (RTC_HandleTypeDef * hrtc, RTC_DateTypeDef * sDate, uint32_t Format)

Function description

HAL_RTC_GetDate

Function name

HAL_StatusTypeDef HAL_RTC_GetDate (const RTC_HandleTypeDef * hrtc, RTC_DateTypeDef * sDate, uint32_t Format)

Function description

HAL_RTC_DST_Add1Hour

Function name

void HAL_RTC_DST_Add1Hour (const RTC_HandleTypeDef * hrtc)

Function description

HAL_RTC_DST_Sub1Hour

Function name

void HAL_RTC_DST_Sub1Hour (const RTC_HandleTypeDef * hrtc)

Function description

HAL_RTC_DST_SetStoreOperation

Function name

void HAL_RTC_DST_SetStoreOperation (const RTC_HandleTypeDef * hrtc)

Function description

HAL_RTC_DST_ClearStoreOperation

Function name

void HAL_RTC_DST_ClearStoreOperation (const RTC_HandleTypeDef * hrtc)

Function description

HAL_RTC_DST_ReadStoreOperation

Function name

uint32_t HAL_RTC_DST_ReadStoreOperation (const RTC_HandleTypeDef * hrtc)

Function description

HAL_RTC_SetAlarm

Function name

HAL_StatusTypeDef HAL_RTC_SetAlarm (RTC_HandleTypeDef * hrtc, RTC_AlarmTypeDef * sAlarm, uint32_t Format)

Function description

HAL_RTC_SetAlarm_IT

Function name

HAL_StatusTypeDef HAL_RTC_SetAlarm_IT (RTC_HandleTypeDef * hrtc, RTC_AlarmTypeDef * sAlarm, uint32_t Format)

Function description

HAL_RTC_DeactivateAlarm

Function name

HAL_StatusTypeDef HAL_RTC_DeactivateAlarm (RTC_HandleTypeDef * hrtc, uint32_t Alarm)

Function description

HAL_RTC_GetAlarm

Function name

HAL_StatusTypeDef HAL_RTC_GetAlarm (const RTC_HandleTypeDef * hrtc, RTC_AlarmTypeDef * sAlarm, uint32_t Alarm, uint32_t Format)

Function description

HAL_RTC_AlarmIRQHandler

Function name

void HAL_RTC_AlarmIRQHandler (RTC_HandleTypeDef * hrtc)

Function description

HAL_RTC_PollForAlarmAEvent

Function name

HAL_StatusTypeDef HAL_RTC_PollForAlarmAEvent (const RTC_HandleTypeDef * hrtc, uint32_t Timeout)

Function description

HAL_RTC_AlarmAEventCallback

Function name

void HAL_RTC_AlarmAEventCallback (RTC_HandleTypeDef * hrtc)

Function description

HAL_RTC_WaitForSynchro

Function name

HAL_StatusTypeDef HAL_RTC_WaitForSynchro (RTC_HandleTypeDef * hrtc)

Function description

HAL_RTC_GetState

Function name

HAL_RTCStateTypeDef HAL_RTC_GetState (const RTC_HandleTypeDef * hrtc)

Function description

RTC_EnterInitMode

Function name

HAL_StatusTypeDef RTC_EnterInitMode (RTC_HandleTypeDef * hrtc)

Function description

RTC_ExitInitMode

Function name

HAL_StatusTypeDef RTC_ExitInitMode (RTC_HandleTypeDef * hrtc)

Function description

RTC_ByteToBcd2

Function name

uint8_t RTC_ByteToBcd2 (uint8_t Value)

Function description

RTC_Bcd2ToByte

Function name

uint8_t RTC_Bcd2ToByte (uint8_t Value)

Function description

61.3 RTC Firmware driver defines

The following section lists the various define and macros of the module.

61.3.1 RTC

RTC

RTC AlarmDateWeekDay Definitions

RTC_ALARMDATEWEEKDAYSEL_DATE

RTC_ALARMDATEWEEKDAYSEL_WEEKDAY

RTC AlarmMask Definitions

RTC_ALARMMASK_NONE

RTC_ALARMMASK_DATEWEEKDAY

RTC_ALARMMASK_HOURS

RTC_ALARMMASK_MINUTES

RTC_ALARMMASK_SECONDS

RTC_ALARMMASK_ALL

RTC Alarms Definitions

RTC_ALARM_A

RTC_ALARM_B

RTC Alarms Flag Auto Clear Definitions

ALARM_FLAG_AUTOCLR_ENABLE

ALARM_FLAG_AUTOCLR_DISABLE

RTC Alarm Sub Seconds BCD Masks Definitions

RTC_ALARMSUBSECONDMASK_ALL

All Alarm SS fields are masked. There is no comparison on sub seconds for Alarm

RTC_ALARMSUBSECONDMASK_SS14_1

SS[14:1] not used in Alarmcomparison. Only SS[0] is compared

RTC_ALARMSUBSECONDMASK_SS14_2

SS[14:2] not used in Alarm comparison. Only SS[1:0] are compared

RTC_ALARMSUBSECONDMASK_SS14_3

SS[14:3] not used in Alarm comparison. Only SS[2:0] are compared

RTC_ALARMSUBSECONDMASK_SS14_4

SS[14:4] not used in Alarm comparison. Only SS[3:0] are compared

RTC_ALARMSUBSECONDMASK_SS14_5

SS[14:5] not used in Alarm comparison. Only SS[4:0] are compared

RTC_ALARMSUBSECONDMASK_SS14_6

SS[14:6] not used in Alarm comparison. Only SS[5:0] are compared

RTC_ALARMSUBSECONDMASK_SS14_7

SS[14:7] not used in Alarm comparison. Only SS[6:0] are compared

RTC_ALARMSUBSECONDMASK_SS14_8

SS[14:8] not used in Alarm comparison. Only SS[7:0] are compared

RTC_ALARMSUBSECONDMASK_SS14_9

SS[14:9] not used in Alarm comparison. Only SS[8:0] are compared

RTC_ALARMSUBSECONDMASK_SS14_10

SS[14:10] not used in Alarm comparison. Only SS[9:0] are compared

RTC_ALARMSUBSECONDMASK_SS14_11

SS[14:11] not used in Alarm comparison. Only SS[10:0] are compared

RTC_ALARMSUBSECONDMASK_SS14_12

SS[14:12] not used in Alarm comparison.Only SS[11:0] are compared

RTC_ALARMSUBSECONDMASK_SS14_13

SS[14:13] not used in Alarm comparison. Only SS[12:0] are compared

RTC_ALARMSUBSECONDMASK_SS14

SS[14] not used in Alarm comparison. Only SS[13:0] are compared

RTC_ALARMSUBSECONDMASK_NONE

SS[14:0] are compared and must match to activate alarm

RTC AM PM Definitions

RTC_HOURFORMAT12_AM

RTC_HOURFORMAT12_PM

RTC Clear Flags Definitions

RTC_CLEAR_SSRUF

Clear SSR underflow flag

RTC_CLEAR_ITSF

Clear Internal Time-stamp flag

RTC_CLEAR_TSOVF

Clear Time-stamp overflow flag

RTC_CLEAR_TSF

Clear Time-stamp flag

RTC_CLEAR_WUTF

Clear Wakeup timer flag

RTC_CLEAR_ALRBF

Clear Alarm B flag

RTC_CLEAR_ALRAF

Clear Alarm A flag

RTC DayLightSaving Definitions

RTC_DAYLIGHTSAVING_SUB1H

RTC_DAYLIGHTSAVING_ADD1H

RTC_DAYLIGHTSAVING_NONE

RTC Exported Macros

__HAL_RTC_RESET_HANDLE_STATE

Description:

- Reset RTC handle state.

Parameters:

- __HANDLE__: RTC handle.

Return value:

- None

__HAL_RTC_WRITEPROTECTION_DISABLE

Description:

- Disable the write protection for RTC registers.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_WRITEPROTECTION_ENABLE

Description:

- Enable the write protection for RTC registers.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_DAYLIGHT_SAVING_TIME_ADD1H

Description:

- Add 1 hour (summer time change).

Parameters:

- __HANDLE__: specifies the RTC handle.
- __BKP__: Backup This parameter can be:
  - RTC_STOREOPERATION_RESET
  - RTC_STOREOPERATION_SET

Return value:

- None

Notes:

- This interface is deprecated. To manage Daylight Saving Time, please use HAL_RTC_DST_xxx functions

__HAL_RTC_DAYLIGHT_SAVING_TIME_SUB1H

Description:

- Subtract 1 hour (winter time change).

Parameters:

- __HANDLE__: specifies the RTC handle.
- __BKP__: Backup This parameter can be:
  - RTC_STOREOPERATION_RESET
  - RTC_STOREOPERATION_SET

Return value:

- None

Notes:

- This interface is deprecated. To manage Daylight Saving Time, please use HAL_RTC_DST_xxx functions

__HAL_RTC_ALARMA_ENABLE

Description:

- Enable the RTC ALARMA peripheral.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_ALARMA_DISABLE

Description:

- Disable the RTC ALARMA peripheral.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_ALARMB_ENABLE

Description:

- Enable the RTC ALARMB peripheral.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_ALARMB_DISABLE

Description:

- Disable the RTC ALARMB peripheral.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_ALARM_ENABLE_IT

Description:

- Enable the RTC Alarm interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Alarm interrupt sources to be enabled or disabled. This parameter can be any combination of the following values:
  - RTC_IT_ALRA Alarm A interrupt
  - RTC_IT_ALRB Alarm B interrupt

Return value:

- None

__HAL_RTC_ALARM_DISABLE_IT

Description:

- Disable the RTC Alarm interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Alarm interrupt sources to be enabled or disabled. This parameter can be any combination of the following values:
  - RTC_IT_ALRA Alarm A interrupt
  - RTC_IT_ALRB Alarm B interrupt

Return value:

- None

__HAL_RTC_ALARM_GET_IT

Description:

- Check whether the specified RTC Alarm interrupt has occurred or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Alarm interrupt sources to check. This parameter can be:
  - RTC_IT_ALRA Alarm A interrupt
  - RTC_IT_ALRB Alarm B interrupt

Return value:

- The: state of __INTERRUPT__ (TRUE or FALSE).

__HAL_RTC_ALARM_GET_IT_SOURCE

Description:

- Check whether the specified RTC Alarm interrupt has been enabled or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Alarm interrupt sources to check. This parameter can be:
  - RTC_IT_ALRA Alarm A interrupt
  - RTC_IT_ALRB Alarm B interrupt

Return value:

- The: state of __INTERRUPT__ (TRUE or FALSE).

__HAL_RTC_ALARM_GET_FLAG

Description:

- Get the selected RTC Alarms flag status.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC Alarm Flag sources to check. This parameter can be:
  - RTC_FLAG_ALRAF
  - RTC_FLAG_ALRBF

Return value:

- The: state of __FLAG__ (TRUE or FALSE).

__HAL_RTC_ALARM_CLEAR_FLAG

Description:

- Clear the RTC Alarms pending flags.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC Alarm Flag sources to clear. This parameter can be:
  - RTC_FLAG_ALRAF
  - RTC_FLAG_ALRBF

Return value:

- None

__HAL_RTC_IS_CALENDAR_INITIALIZED

Description:

- Check whether if the RTC Calendar is initialized.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- The: state of RTC Calendar initialization (TRUE or FALSE).

RTC Flags Definitions

RTC_FLAG_RECALPF

Recalibration pending flag

RTC_FLAG_INITF

Initialization flag

RTC_FLAG_RSF

Registers synchronization flag

RTC_FLAG_INITS

Initialization status flag

RTC_FLAG_SHPF

Shift operation pending flag

RTC_FLAG_WUTWF

Wakeup timer write flag

RTC_FLAG_SSRUF

SSR underflow flag

RTC_FLAG_ITSF

Internal Time-stamp flag

RTC_FLAG_TSOVF

Time-stamp overflow flag

RTC_FLAG_TSF

Time-stamp flag

RTC_FLAG_WUTF

Wakeup timer flag

RTC_FLAG_ALRBF

Alarm B flag

RTC_FLAG_ALRAF

Alarm A flag

RTC Hour Formats

RTC_HOURFORMAT_24

RTC_HOURFORMAT_12

RTC Input Parameter Format Definitions

RTC_FORMAT_BIN

RTC_FORMAT_BCD

RTC Interrupts Definitions

RTC_IT_TS

Enable Timestamp Interrupt

RTC_IT_WUT

Enable Wakeup timer Interrupt

RTC_IT_SSRU

Enable SSR Underflow Interrupt

RTC_IT_ALRA

Enable Alarm A Interrupt

RTC_IT_ALRB

Enable Alarm B Interrupt

RTC Private macros to check input parameters

IS_RTC_OUTPUT

IS_RTC_HOUR_FORMAT

IS_RTC_OUTPUT_POL

IS_RTC_OUTPUT_TYPE

IS_RTC_OUTPUT_PULLUP

IS_RTC_OUTPUT_REMAP

IS_RTC_HOURFORMAT12

IS_RTC_DAYLIGHT_SAVING

IS_RTC_STORE_OPERATION

IS_RTC_FORMAT

IS_RTC_YEAR

IS_RTC_MONTH

IS_RTC_DATE

IS_RTC_WEEKDAY

IS_RTC_ALARM_DATE_WEEKDAY_DATE

IS_RTC_ALARM_DATE_WEEKDAY_WEEKDAY

IS_RTC_ALARM_DATE_WEEKDAY_SEL

IS_RTC_ALARM_MASK

IS_RTC_ALARM

IS_RTC_ALARM_SUB_SECOND_VALUE

IS_RTC_ALARM_SUB_SECOND_MASK

IS_RTC_ASYNCH_PREDIV

IS_RTC_SYNCH_PREDIV

IS_RTC_HOUR12

IS_RTC_HOUR24

IS_RTC_MINUTES

IS_RTC_SECONDS

RTC Month Date Definitions

RTC_MONTH_JANUARY

RTC_MONTH_FEBRUARY

RTC_MONTH_MARCH

RTC_MONTH_APRIL

RTC_MONTH_MAY

RTC_MONTH_JUNE

RTC_MONTH_JULY

RTC_MONTH_AUGUST

RTC_MONTH_SEPTEMBER

RTC_MONTH_OCTOBER

RTC_MONTH_NOVEMBER

RTC_MONTH_DECEMBER

RTC Output ALARM OUT Remap

RTC_OUTPUT_REMAP_NONE

RTC_OUTPUT_REMAP_POS1

RTC Output Polarity Definitions

RTC_OUTPUT_POLARITY_HIGH

RTC_OUTPUT_POLARITY_LOW

RTC Output Pull-Up ALARM OUT

RTC_OUTPUT_PULLUP_NONE

RTC_OUTPUT_PULLUP_ON

RTC Output Type ALARM OUT

RTC_OUTPUT_TYPE_PUSHPULL

RTC_OUTPUT_TYPE_OPENDRAIN

RTC StoreOperation Definitions

RTC_STOREOPERATION_RESET

RTC_STOREOPERATION_SET

RTC WeekDay Definitions

RTC_WEEKDAY_MONDAY

RTC_WEEKDAY_TUESDAY

RTC_WEEKDAY_WEDNESDAY

RTC_WEEKDAY_THURSDAY

RTC_WEEKDAY_FRIDAY

RTC_WEEKDAY_SATURDAY

RTC_WEEKDAY_SUNDAY

62 HAL RTC Extension Driver

62.1 RTCEx Firmware driver registers structures

62.1.1 RTC_TamperTypeDef

RTC_TamperTypeDef is defined in the stm32n6xx_hal_rtc_ex.h

Data Fields

- ***uint32_t Tamper***
- ***uint32_t Trigger***
- ***uint32_t NoErase***
- ***uint32_t MaskFlag***
- ***uint32_t Filter***
- ***uint32_t SamplingFrequency***
- ***uint32_t PrechargeDuration***
- ***uint32_t TamperPullUp***
- ***uint32_t TimeStampOnTamperDetection***

Field Documentation

- ***uint32_t RTC_TamperTypeDef::Tamper***
  Specifies the Tamper Pin. This parameter can be a value of ***RTCEx_Tamper_Pins***
- ***uint32_t RTC_TamperTypeDef::Trigger***
  Specifies the Tamper Trigger. This parameter can be a value of ***RTCEx_Tamper_Trigger***
- ***uint32_t RTC_TamperTypeDef::NoErase***
  Specifies the Tamper no erase mode. This parameter can be a value of ***RTCEx_Tamper_EraseBackUp***
- ***uint32_t RTC_TamperTypeDef::MaskFlag***
  Specifies the Tamper Flag masking. This parameter can be a value of ***RTCEx_Tamper_MaskFlag***
- ***uint32_t RTC_TamperTypeDef::Filter***
  Specifies the TAMP Filter Tamper. This parameter can be a value of ***RTCEx_Tamper_Filter***
- ***uint32_t RTC_TamperTypeDef::SamplingFrequency***
  Specifies the sampling frequency. This parameter can be a value of ***RTCEx_Tamper_Sampling_Frequencies***
- ***uint32_t RTC_TamperTypeDef::PrechargeDuration***
  Specifies the Precharge Duration. This parameter can be a value of ***RTCEx_Tamper_Pin_Precharge_Duration***
- ***uint32_t RTC_TamperTypeDef::TamperPullUp***
  Specifies the Tamper PullUp. This parameter can be a value of ***RTCEx_Tamper_Pull_UP***
- ***uint32_t RTC_TamperTypeDef::TimeStampOnTamperDetection***
  Specifies the TimeStampOnTamperDetection. This parameter can be a value of ***RTCEx_Tamper_TimeStampOnTamperDetection***

62.1.2 RTC_ATampInputTypeDef

RTC_ATampInputTypeDef is defined in the stm32n6xx_hal_rtc_ex.h

Data Fields

- ***uint32_t Enable***
- ***uint32_t Interrupt***
- ***uint32_t Output***
- ***uint32_t NoErase***
- ***uint32_t MaskFlag***

Field Documentation

- ***uint32_t RTC_ATampInputTypeDef::Enable***
  Specifies the Tamper input is active. This parameter can be a value of ***RTCEx_ActiveTamper_Enable***

- ***uint32_t RTC_ATampInputTypeDef::Interrupt***
  Specifies the interrupt mode. This parameter can be a value of ***RTCEx_ActiveTamper_Interrupt***
- ***uint32_t RTC_ATampInputTypeDef::Output***
  Specifies the TAMP output to be compared with. The same output can be used for several tamper inputs. This parameter can be a value of ***RTCEx_ActiveTamper_Sel***
- ***uint32_t RTC_ATampInputTypeDef::NoErase***
  Specifies the Tamper no erase mode. This parameter can be a value of ***RTCEx_Tamper_EraseBackUp***
- ***uint32_t RTC_ATampInputTypeDef::MaskFlag***
  Specifies the Tamper Flag masking. This parameter can be a value of ***RTCEx_Tamper_MaskFlag***

62.1.3 RTC_ActiveTampersTypeDef

RTC_ActiveTampersTypeDef is defined in the stm32n6xx_hal_rtc_ex.h

Data Fields

- ***uint32_t ActiveFilter***
- ***uint32_t ActiveAsyncPrescaler***
- ***uint32_t TimeStampOnTamperDetection***
- ***uint32_t ActiveOutputChangePeriod***
- ***uint32_t Seed***
- ***RTC_ATampInputTypeDef TampInput***

Field Documentation

- ***uint32_t RTC_ActiveTampersTypeDef::ActiveFilter***
  Specifies the Active tamper filter enable. This parameter can be a value of ***RTCEx_ActiveTamper_Filter***
- ***uint32_t RTC_ActiveTampersTypeDef::ActiveAsyncPrescaler***
  Specifies the Active Tamper asynchronous Prescaler clock. This parameter can be a value of ***RTCEx_ActiveTamper_Async_prescaler***
- ***uint32_t RTC_ActiveTampersTypeDef::TimeStampOnTamperDetection***
  Specifies the timeStamp on tamper detection. This parameter can be a value of ***RTCEx_Tamper_TimeStampOnTamperDetection***
- ***uint32_t RTC_ActiveTampersTypeDef::ActiveOutputChangePeriod***
  Specifies the Active Tamper output change period. This parameter can be a value from 0 to 7
- ***uint32_t RTC_ActiveTampersTypeDef::Seed[RTC_ATAMP_SEED_NB_UINT32]***
  Specifies the RNG Seed value. This parameter is an array of value from 0 to 0xFFFFFFFF
- ***RTC_ATampInputTypeDef RTC_ActiveTampersTypeDef::TampInput[RTC_TAMP_NB]***
  Specifies configuration of all active tampers. The index of TampInput[RTC_TAMP_NB] can be a value of RTCEx_ActiveTamper_Sel

62.1.4 RTC_InternalTamperTypeDef

RTC_InternalTamperTypeDef is defined in the stm32n6xx_hal_rtc_ex.h

Data Fields

- ***uint32_t IntTamper***
- ***uint32_t TimeStampOnTamperDetection***
- ***uint32_t NoErase***

Field Documentation

- ***uint32_t RTC_InternalTamperTypeDef::IntTamper***
  Specifies the Internal Tamper Pin. This parameter can be a value of ***RTCEx_Internal_Tamper_Pins***
- ***uint32_t RTC_InternalTamperTypeDef::TimeStampOnTamperDetection***
  Specifies the TimeStampOnTamperDetection. This parameter can be a value of ***RTCEx_Tamper_TimeStampOnTamperDetection***
- ***uint32_t RTC_InternalTamperTypeDef::NoErase***
  Specifies the internal Tamper no erase mode. This parameter can be a value of ***RTCEx_Tamper_EraseBackUp***

62.1.5 RTC_SecureStateTypeDef

RTC_SecureStateTypeDef is defined in the stm32n6xx_hal_rtc_ex.h

Data Fields

- ***uint32_t rtcSecureFull***
- ***uint32_t rtcNonSecureFeatures***
- ***uint32_t tampSecureFull***
- ***uint32_t backupRegisterStartZone2***
- ***uint32_t backupRegisterStartZone3***
- ***uint32_t MonotonicCounterSecure***

Field Documentation

- ***uint32_t RTC_SecureStateTypeDef::rtcSecureFull***
  Specifies If the RTC is fully secure or not. This parameter can be a value of ***RTCEx_RTC_Secure_Full***
- ***uint32_t RTC_SecureStateTypeDef::rtcNonSecureFeatures***
  Specifies the non-secure features. This parameter is only relevant if RTC is not fully secure (rtcSecureFull == RTC_SECURE_FULL_NO). This parameter can be a combination of ***RTCEx_RTC_NonSecure_Features***
- ***uint32_t RTC_SecureStateTypeDef::tampSecureFull***
  Specifies If the TAMP is fully secure or not execpt monotonic counters and BackUp registers. This parameter can be a value of ***RTCEx_TAMP_Secure_Full***
- ***uint32_t RTC_SecureStateTypeDef::backupRegisterStartZone2***
  Specifies the backup register start zone 2. Zone 1 : read secure write secure. Zone 2 : read non-secure write secure. This parameter can be RTC_BKP_DRx where x can be from 0 to 31 to specify the register. Warning : this parameter is shared with ***Section 61.3.1: RTC_PrivilegeStateTypeDef***
- ***uint32_t RTC_SecureStateTypeDef::backupRegisterStartZone3***
  Specifies the backup register start zone 3. Zone 3 : read non-secure write non-secure. This parameter can be RTC_BKP_DRx where x can be from 0 to 31 to specify the register. Warning : this parameter is shared with ***Section 61.3.1: RTC_PrivilegeStateTypeDef***
- ***uint32_t RTC_SecureStateTypeDef::MonotonicCounterSecure***
  Specifies If the monotonic counter is secure or not. This parameter can be a value of ***RTCEx_TAMP_Monotonic_Counter_Secure***

62.1.6 RTC_PrivilegeStateTypeDef

RTC_PrivilegeStateTypeDef is defined in the stm32n6xx_hal_rtc_ex.h

Data Fields

- ***uint32_t rtcPrivilegeFull***
- ***uint32_t rtcPrivilegeFeatures***
- ***uint32_t tampPrivilegeFull***
- ***uint32_t backupRegisterPrivZone***
- ***uint32_t backupRegisterStartZone2***
- ***uint32_t backupRegisterStartZone3***
- ***uint32_t MonotonicCounterPrivilege***

Field Documentation

- ***uint32_t RTC_PrivilegeStateTypeDef::rtcPrivilegeFull***
  Specifies If the RTC is fully privileged or not. This parameter can be a value of ***RTCEx_RTC_Privilege_Full***
- ***uint32_t RTC_PrivilegeStateTypeDef::rtcPrivilegeFeatures***
  Specifies the privileged features. This parameter is only relevant if RTC is not fully privileged (rtcPrivilegeFull == RTC_PRIVILEGE_FULL_NO). This parameter can be a combination of ***RTCEx_RTC_Privilege_Features***
- ***uint32_t RTC_PrivilegeStateTypeDef::tampPrivilegeFull***
  Specifies If the TAMP is fully privileged or not execpt monotonic counters and BackUp registers. This parameter can be a value of ***RTCEx_TAMP_Privilege_Full***
- ***uint32_t RTC_PrivilegeStateTypeDef::backupRegisterPrivZone***
  Specifies backup register zone to be privileged. This parameter can be a combination of ***RTCEx_Backup_Reg_Privilege_zone***. Warning : this parameter is writable in secure mode or if trustzone is disabled

- ***uint32_t RTC_PrivilegeStateTypeDef::backupRegisterStartZone2***
  Specifies the backup register start zone 2. Zone 1 : read secure, write secure. Zone 2 : read non-secure, write secure. This parameter can be RTC_BKP_DRx where x can be from 0 to 31 to specify the register . Warning : this parameter is writable in secure mode or if trustzone is disabled. Warning : this parameter is shared with ***Section 61.3.1: RTC_SecureStateTypeDef***
- ***uint32_t RTC_PrivilegeStateTypeDef::backupRegisterStartZone3***
  Specifies the backup register start zone 3. Zone 3 : read non-secure, write non-secure. This parameter can be RTC_BKP_DRx where x can be from 0 to 31 to specify the register. Warning : this parameter is writable in secure mode or if trustzone is disabled. Warning : this parameter is shared with ***Section 61.3.1: RTC_SecureStateTypeDef***
- ***uint32_t RTC_PrivilegeStateTypeDef::MonotonicCounterPrivilege***
  Specifies If the monotonic counter is privileged or not. This parameter can be a value of ***RTCEx_TAMP_Monotonic_Counter_Privilege***

62.2 RTCEx Firmware driver API description

The following section lists the various functions of the RTCEx library.

62.2.1 How to use this driver

- Enable the RTC domain access.
- Configure the RTC Prescaler (Asynchronous and Synchronous) and RTC hour format using the HAL_RTC_Init() function.

RTC Wakeup configuration

- To configure the RTC Wakeup Clock source and Counter use the HAL_RTCEx_SetWakeUpTimer() function. You can also configure the RTC Wakeup timer with interrupt mode using the HAL_RTCEx_SetWakeUpTimer_IT() function.
- To read the RTC WakeUp Counter register, use the HAL_RTCEx_GetWakeUpTimer() function.

Outputs configuration

The RTC has 2 outputs pins (RTC_OUT1 & RTC_OUT2). To configure the outputs, use the HAL_RTC_Init() function.

- RTC_OUT1 and RTC_OUT2 which select one of the following two outputs:
- CALIB: 512Hz or 1Hz clock output (with an LSE frequency of 32.768kHz). To enable the CALIB, use the HAL_RTCEx_SetCalibrationOutPut() function.
- TAMPALRM: This output is the OR between rtc_tamp_evt and ALARM signals. ALARM is enabled by configuring the OSEL[1:0] bits in the RTC_CR register which select the alarm A, alarm B or wakeup outputs. rtc_tamp_evt is enabled by setting the TAMPOE bit in the RTC_CR register which selects the tamper event outputs.

Smooth digital Calibration configuration

- Configure the RTC Original Digital Calibration Value and the corresponding calibration cycle period (32s,16s and 8s) using the HAL_RTCEx_SetSmoothCalib() function.

RTC synchronization

- The RTC can be finely adjusted using HAL_RTCEx_SetSynchroShift() function. Writing to RTC_SHIFTR can shift (either delay or advance) the clock with a resolution of 1 ck_apre period. The shift operation consists in adding the SUBFS[14:0] value to the synchronous prescaler counter SS[15:0]: this delays the clock.

Bypass shadow registers

- Enable bypass shadow registers using the HAL_RTCEx_EnableBypassShadow(). When the Bypass Shadow is enabled the calendar value are taken directly from the Calendar counter. Thus eliminating the need to wait for the RSF bit to be set. This is especially useful after exiting from low-power modes (Stop or Standby), since the shadow registers are not updated during these modes.

RTC ultra-low-power mode

- Configure the RTC ultra-low-power mode using HAL_RTCEx_SetLowPowerCalib() function. In this case, the calibration mechanism is applied on ck_apre instead of RTCCLK. The resulting accuracy is the same, but the calibration is performed during a calibration cycle of about 220 x PREDIV_A x RTCCLK pulses instead of 220 RTCCLK pulses.

RTC subsecond register underflow interruption

- Enable the RTC SSRU interruption mode using HAL_RTCEx_SetSSRU_IT() function. In this case, when the SSR rolls under 0, an SSRU interruption is triggered. Disable the RTC SSRU interruption mode using HAL_RTCEx_DeactivateSSRU() function.

TimeStamp configuration

- Enable the RTC TimeStamp using the HAL_RTCEx_SetTimeStamp() function. You can also configure the RTC TimeStamp with interrupt mode using the HAL_RTCEx_SetTimeStamp_IT() function.
- To read the RTC TimeStamp Time and Date register, use the HAL_RTCEx_GetTimeStamp() function.

Internal TimeStamp configuration

- Enable the RTC internal TimeStamp using the HAL_RTCEx_SetInternalTimeStamp() function. User has to check internal timestamp occurrence using __HAL_RTC_INTERNAL_TIMESTAMP_GET_FLAG.
- To read the RTC TimeStamp Time and Date register, use the HAL_RTCEx_GetTimeStamp() function.

Tamper configuration

- Enable the RTC Tamper and configure the Tamper filter count, trigger Edge or Level according to the Tamper filter (if equal to 0 Edge else Level) value, sampling frequency, NoErase, MaskFlag, precharge or discharge and Pull-UP using the HAL_RTCEx_SetTamper() function. You can configure RTC Tamper with interrupt mode using HAL_RTCEx_SetTamper_IT() function.
- The default configuration of the Tamper erases the backup registers. To avoid erase, enable the TAMPxPOM field on the TAMP_CR2 register.
- With new RTC tamper configuration, you have to call HAL_RTC_Init() in order to perform TAMP base address offset calculation.
- Enable Internal tamper using HAL_RTCEx_SetInternalTamper. IT mode can be chosen using setting Interrupt field.

Backup Data Registers and Device Secrets configuration

- To write to the RTC Backup Data registers, use the HAL_RTCEx_BKUPWrite() function.
- To read the RTC Backup registers, use the HAL_RTCEx_BKUPRead() function.
- To reset the RTC Backup registers and erase the device secrets, use HAL_RTCEx_BKUPErase() function.
- Enable the lock of the Boot hardware Key using the HAL_RTCEx_LockBootHardwareKey() function. The backup registers from TAMP_BKP0R to TAMP_BKP7R cannot be accessed neither in read nor in write (they are read as 0 and write ignore).
- Configure the erase of the Device Secrets using HAL_RTCEx_ConfigEraseDeviceSecrets() function.
- Block the access to the RTC Backup registers and all the device secrets using HAL_RTCEx_BKUPBlock() function.

Monotonic counter

- To increment the Monotonic counter, use the HAL_RTCEx_MonotonicCounterIncrement() function.
- To get the current value of the Monotonic counter, use the HAL_RTCEx_MonotonicCounterGet() function.

RTC & TAMP secure protection modes

- Set the security level of the RTC/TAMP/Backup registers using HAL_RTCEx_SecureModeSet() function. +) Get the security level of the RTC/TAMP/Backup registers using HAL_RTCEx_SecureModeGet() function.

RTC & TAMP privilege protection modes

- Set the privilege level of the RTC/TAMP/Backup registers using HAL_RTCEx_PrivilegeModeSet() function.
  +) Get the privilege level of the RTC/TAMP/Backup registers using HAL_RTCEx_PrivilegeModeGet() function.

62.2.2 Detailed description of functions

HAL_RTCEx_SetTimeStamp

Function name

HAL_StatusTypeDef HAL_RTCEx_SetTimeStamp (RTC_HandleTypeDef * hrtc, uint32_t TimeStampEdge, uint32_t RTC_TimeStampPin)

Function description

HAL_RTCEx_SetTimeStamp_IT

Function name

HAL_StatusTypeDef HAL_RTCEx_SetTimeStamp_IT (RTC_HandleTypeDef * hrtc, uint32_t TimeStampEdge, uint32_t RTC_TimeStampPin)

Function description

HAL_RTCEx_DeactivateTimeStamp

Function name

HAL_StatusTypeDef HAL_RTCEx_DeactivateTimeStamp (RTC_HandleTypeDef * hrtc)

Function description

HAL_RTCEx_SetInternalTimeStamp

Function name

HAL_StatusTypeDef HAL_RTCEx_SetInternalTimeStamp (RTC_HandleTypeDef * hrtc)

Function description

HAL_RTCEx_DeactivateInternalTimeStamp

Function name

HAL_StatusTypeDef HAL_RTCEx_DeactivateInternalTimeStamp (RTC_HandleTypeDef * hrtc)

Function description

HAL_RTCEx_GetTimeStamp

Function name

HAL_StatusTypeDef HAL_RTCEx_GetTimeStamp (const RTC_HandleTypeDef * hrtc, RTC_TimeTypeDef * sTimeStamp, RTC_DateTypeDef * sTimeStampDate, uint32_t Format)

Function description

HAL_RTCEx_TimeStampIRQHandler

Function name

void HAL_RTCEx_TimeStampIRQHandler (RTC_HandleTypeDef * hrtc)

Function description

HAL_RTCEx_PollForTimeStampEvent

Function name

HAL_StatusTypeDef HAL_RTCEx_PollForTimeStampEvent (const RTC_HandleTypeDef * hrtc, uint32_t Timeout)

Function description

HAL_RTCEx_TimeStampEventCallback

Function name

void HAL_RTCEx_TimeStampEventCallback (RTC_HandleTypeDef * hrtc)

Function description

HAL_RTCEx_SetWakeUpTimer

Function name

HAL_StatusTypeDef HAL_RTCEx_SetWakeUpTimer (RTC_HandleTypeDef * hrtc, uint32_t WakeUpCounter, uint32_t WakeUpClock)

Function description

HAL_RTCEx_SetWakeUpTimer_IT

Function name

HAL_StatusTypeDef HAL_RTCEx_SetWakeUpTimer_IT (RTC_HandleTypeDef * hrtc, uint32_t WakeUpCounter, uint32_t WakeUpClock, uint32_t WakeUpAutoClr)

Function description

HAL_RTCEx_DeactivateWakeUpTimer

Function name

HAL_StatusTypeDef HAL_RTCEx_DeactivateWakeUpTimer (RTC_HandleTypeDef * hrtc)

Function description

HAL_RTCEx_GetWakeUpTimer

Function name

uint32_t HAL_RTCEx_GetWakeUpTimer (const RTC_HandleTypeDef * hrtc)

Function description

HAL_RTCEx_WakeUpTimerIRQHandler

Function name

void HAL_RTCEx_WakeUpTimerIRQHandler (RTC_HandleTypeDef * hrtc)

Function description

HAL_RTCEx_WakeUpTimerEventCallback

Function name

void HAL_RTCEx_WakeUpTimerEventCallback (RTC_HandleTypeDef * hrtc)

Function description

HAL_RTCEx_PollForWakeUpTimerEvent

Function name

HAL_StatusTypeDef HAL_RTCEx_PollForWakeUpTimerEvent (const RTC_HandleTypeDef * hrtc, uint32_t Timeout)

Function description

HAL_RTCEx_SetSmoothCalib

Function name

HAL_StatusTypeDef HAL_RTCEx_SetSmoothCalib (RTC_HandleTypeDef * hrtc, uint32_t SmoothCalibPeriod, uint32_t SmoothCalibPlusPulses, uint32_t SmoothCalibMinusPulsesValue)

Function description

HAL_RTCEx_SetLowPowerCalib

Function name

HAL_StatusTypeDef HAL_RTCEx_SetLowPowerCalib (RTC_HandleTypeDef * hrtc, uint32_t LowPowerCalib)

Function description

HAL_RTCEx_SetSynchroShift

Function name

HAL_StatusTypeDef HAL_RTCEx_SetSynchroShift (RTC_HandleTypeDef * hrtc, uint32_t ShiftAdd1S, uint32_t ShiftSubFS)

Function description

HAL_RTCEx_SetCalibrationOutPut

Function name

HAL_StatusTypeDef HAL_RTCEx_SetCalibrationOutPut (RTC_HandleTypeDef * hrtc, uint32_t CalibOutput)

Function description

HAL_RTCEx_DeactivateCalibrationOutPut

Function name

HAL_StatusTypeDef HAL_RTCEx_DeactivateCalibrationOutPut (RTC_HandleTypeDef * hrtc)

Function description

HAL_RTCEx_SetRefClock

Function name

HAL_StatusTypeDef HAL_RTCEx_SetRefClock (RTC_HandleTypeDef * hrtc)

Function description

HAL_RTCEx_DeactivateRefClock

Function name

HAL_StatusTypeDef HAL_RTCEx_DeactivateRefClock (RTC_HandleTypeDef * hrtc)

Function description

HAL_RTCEx_EnableBypassShadow

Function name

HAL_StatusTypeDef HAL_RTCEx_EnableBypassShadow (RTC_HandleTypeDef * hrtc)

Function description

HAL_RTCEx_DisableBypassShadow

Function name

HAL_StatusTypeDef HAL_RTCEx_DisableBypassShadow (RTC_HandleTypeDef * hrtc)

Function description

HAL_RTCEx_MonotonicCounterIncrement

Function name

HAL_StatusTypeDef HAL_RTCEx_MonotonicCounterIncrement (const RTC_HandleTypeDef * hrtc, uint32_t Instance)

Function description

HAL_RTCEx_MonotonicCounterGet

Function name

HAL_StatusTypeDef HAL_RTCEx_MonotonicCounterGet (const RTC_HandleTypeDef * hrtc, uint32_t Instance, uint32_t * pValue)

Function description

HAL_RTCEx_SetSSRU_IT

Function name

HAL_StatusTypeDef HAL_RTCEx_SetSSRU_IT (RTC_HandleTypeDef * hrtc)

Function description

HAL_RTCEx_DeactivateSSRU

Function name

HAL_StatusTypeDef HAL_RTCEx_DeactivateSSRU (RTC_HandleTypeDef * hrtc)

Function description

HAL_RTCEx_SSRUIRQHandler

Function name

void HAL_RTCEx_SSRUIRQHandler (RTC_HandleTypeDef * hrtc)

Function description

HAL_RTCEx_SSRUEventCallback

Function name

void HAL_RTCEx_SSRUEventCallback (RTC_HandleTypeDef * hrtc)

Function description

HAL_RTCEx_AlarmBEventCallback

Function name

void HAL_RTCEx_AlarmBEventCallback (RTC_HandleTypeDef * hrtc)

Function description

HAL_RTCEx_PollForAlarmBEvent

Function name

HAL_StatusTypeDef HAL_RTCEx_PollForAlarmBEvent (const RTC_HandleTypeDef * hrtc, uint32_t Timeout)

Function description

HAL_RTCEx_SetTamper

Function name

HAL_StatusTypeDef HAL_RTCEx_SetTamper (const RTC_HandleTypeDef * hrtc, const RTC_TamperTypeDef * sTamper)

Function description

HAL_RTCEx_SetActiveTampers

Function name

HAL_StatusTypeDef HAL_RTCEx_SetActiveTampers (RTC_HandleTypeDef * hrtc, const RTC_ActiveTampersTypeDef * sAllTamper)

Function description

HAL_RTCEx_SetActiveSeed

Function name

HAL_StatusTypeDef HAL_RTCEx_SetActiveSeed (RTC_HandleTypeDef * hrtc, const uint32_t * pSeed)

Function description

HAL_RTCEx_SetTamper_IT

Function name

HAL_StatusTypeDef HAL_RTCEx_SetTamper_IT (const RTC_HandleTypeDef * hrtc, const RTC_TamperTypeDef * sTamper)

Function description

HAL_RTCEx_DeactivateTamper

Function name

HAL_StatusTypeDef HAL_RTCEx_DeactivateTamper (const RTC_HandleTypeDef * hrtc, uint32_t Tamper)

Function description

HAL_RTCEx_DeactivateActiveTampers

Function name

HAL_StatusTypeDef HAL_RTCEx_DeactivateActiveTampers (const RTC_HandleTypeDef * hrtc)

Function description

HAL_RTCEx_PollForTamperEvent

Function name

HAL_StatusTypeDef HAL_RTCEx_PollForTamperEvent (const RTC_HandleTypeDef * hrtc, uint32_t Tamper, uint32_t Timeout)

Function description

HAL_RTCEx_SetInternalTamper

Function name

HAL_StatusTypeDef HAL_RTCEx_SetInternalTamper (const RTC_HandleTypeDef * hrtc, const RTC_InternalTamperTypeDef * sIntTamper)

Function description

HAL_RTCEx_SetInternalTamper_IT

Function name

HAL_StatusTypeDef HAL_RTCEx_SetInternalTamper_IT (const RTC_HandleTypeDef * hrtc, const RTC_InternalTamperTypeDef * sIntTamper)

Function description

HAL_RTCEx_DeactivateInternalTamper

Function name

HAL_StatusTypeDef HAL_RTCEx_DeactivateInternalTamper (const RTC_HandleTypeDef * hrtc, uint32_t IntTamper)

Function description

HAL_RTCEx_PollForInternalTamperEvent

Function name

HAL_StatusTypeDef HAL_RTCEx_PollForInternalTamperEvent (const RTC_HandleTypeDef * hrtc, uint32_t IntTamper, uint32_t Timeout)

Function description

HAL_RTCEx_LockBootHardwareKey

Function name

HAL_StatusTypeDef HAL_RTCEx_LockBootHardwareKey (const RTC_HandleTypeDef * hrtc)

Function description

HAL_RTCEx_TamperIRQHandler

Function name

void HAL_RTCEx_TamperIRQHandler (RTC_HandleTypeDef * hrtc)

Function description

HAL_RTCEx_Tamper1EventCallback

Function name

void HAL_RTCEx_Tamper1EventCallback (RTC_HandleTypeDef * hrtc)

Function description

HAL_RTCEx_Tamper2EventCallback

Function name

void HAL_RTCEx_Tamper2EventCallback (RTC_HandleTypeDef * hrtc)

Function description

HAL_RTCEx_Tamper3EventCallback

Function name

void HAL_RTCEx_Tamper3EventCallback (RTC_HandleTypeDef * hrtc)

Function description

HAL_RTCEx_Tamper4EventCallback

Function name

void HAL_RTCEx_Tamper4EventCallback (RTC_HandleTypeDef * hrtc)

Function description

HAL_RTCEx_Tamper5EventCallback

Function name

void HAL_RTCEx_Tamper5EventCallback (RTC_HandleTypeDef * hrtc)

Function description

HAL_RTCEx_Tamper6EventCallback

Function name

void HAL_RTCEx_Tamper6EventCallback (RTC_HandleTypeDef * hrtc)

Function description

HAL_RTCEx_Tamper7EventCallback

Function name

void HAL_RTCEx_Tamper7EventCallback (RTC_HandleTypeDef * hrtc)

Function description

HAL_RTCEx_Tamper8EventCallback

Function name

void HAL_RTCEx_Tamper8EventCallback (RTC_HandleTypeDef * hrtc)

Function description

HAL_RTCEx_InternalTamper1EventCallback

Function name

void HAL_RTCEx_InternalTamper1EventCallback (RTC_HandleTypeDef * hrtc)

Function description

HAL_RTCEx_InternalTamper2EventCallback

Function name

void HAL_RTCEx_InternalTamper2EventCallback (RTC_HandleTypeDef * hrtc)

Function description

HAL_RTCEx_InternalTamper3EventCallback

Function name

void HAL_RTCEx_InternalTamper3EventCallback (RTC_HandleTypeDef * hrtc)

Function description

HAL_RTCEx_InternalTamper4EventCallback

Function name

void HAL_RTCEx_InternalTamper4EventCallback (RTC_HandleTypeDef * hrtc)

Function description

HAL_RTCEx_InternalTamper5EventCallback

Function name

void HAL_RTCEx_InternalTamper5EventCallback (RTC_HandleTypeDef * hrtc)

Function description

HAL_RTCEx_InternalTamper6EventCallback

Function name

void HAL_RTCEx_InternalTamper6EventCallback (RTC_HandleTypeDef * hrtc)

Function description

HAL_RTCEx_InternalTamper7EventCallback

Function name

void HAL_RTCEx_InternalTamper7EventCallback (RTC_HandleTypeDef * hrtc)

Function description

HAL_RTCEx_InternalTamper8EventCallback

Function name

void HAL_RTCEx_InternalTamper8EventCallback (RTC_HandleTypeDef * hrtc)

Function description

HAL_RTCEx_InternalTamper9EventCallback

Function name

void HAL_RTCEx_InternalTamper9EventCallback (RTC_HandleTypeDef * hrtc)

Function description

HAL_RTCEx_InternalTamper11EventCallback

Function name

void HAL_RTCEx_InternalTamper11EventCallback (RTC_HandleTypeDef * hrtc)

Function description

HAL_RTCEx_BKUPWrite

Function name

void HAL_RTCEx_BKUPWrite (const RTC_HandleTypeDef * hrtc, uint32_t BackupRegister, uint32_t Data)

Function description

HAL_RTCEx_BKUPRead

Function name

uint32_t HAL_RTCEx_BKUPRead (const RTC_HandleTypeDef * hrtc, uint32_t BackupRegister)

Function description

HAL_RTCEx_BKUPErase

Function name

void HAL_RTCEx_BKUPErase (const RTC_HandleTypeDef * hrtc)

Function description

HAL_RTCEx_BKUPBlock

Function name

void HAL_RTCEx_BKUPBlock (const RTC_HandleTypeDef * hrtc)

Function description

HAL_RTCEx_BKUPUnblock

Function name

void HAL_RTCEx_BKUPUnblock (const RTC_HandleTypeDef * hrtc)

Function description

HAL_RTCEx_ConfigEraseDeviceSecrets

Function name

void HAL_RTCEx_ConfigEraseDeviceSecrets (const RTC_HandleTypeDef * hrtc, uint32_t DeviceSecretConf)

Function description

HAL_RTCEx_SecureModeGet

Function name

HAL_StatusTypeDef HAL_RTCEx_SecureModeGet (const RTC_HandleTypeDef * hrtc, RTC_SecureStateTypeDef * secureState)

Function description

HAL_RTCEx_SecureModeSet

Function name

HAL_StatusTypeDef HAL_RTCEx_SecureModeSet (const RTC_HandleTypeDef * hrtc, const RTC_SecureStateTypeDef * secureState)

Function description

HAL_RTCEx_PrivilegeModeSet

Function name

HAL_StatusTypeDef HAL_RTCEx_PrivilegeModeSet (const RTC_HandleTypeDef * hrtc, const RTC_PrivilegeStateTypeDef * privilegeState)

Function description

HAL_RTCEx_PrivilegeModeGet

Function name

HAL_StatusTypeDef HAL_RTCEx_PrivilegeModeGet (const RTC_HandleTypeDef * hrtc, RTC_PrivilegeStateTypeDef * privilegeState)

Function description

62.3 RTCEx Firmware driver defines

The following section lists the various define and macros of the module.

62.3.1 RTCEx

RTCEx

RTCEx Active_Tamper_Asynchronous_Prescaler clock Definitions

RTC_ATAMP_ASYNCPRES_RTCCLK

RTCCLK

RTC_ATAMP_ASYNCPRES_RTCCLK_2
RTCCLK/2
RTC_ATAMP_ASYNCPRES_RTCCLK_4
RTCCLK/4
RTC_ATAMP_ASYNCPRES_RTCCLK_8
RTCCLK/8
RTC_ATAMP_ASYNCPRES_RTCCLK_16
RTCCLK/16
RTC_ATAMP_ASYNCPRES_RTCCLK_32
RTCCLK/32
RTC_ATAMP_ASYNCPRES_RTCCLK_64
RTCCLK/64
RTC_ATAMP_ASYNCPRES_RTCCLK_128
RTCCLK/128
RTC_ATAMP_ASYNCPRES_RTCCLK_256
RTCCLK/256
RTC_ATAMP_ASYNCPRES_RTCCLK_512
RTCCLK/512
RTC_ATAMP_ASYNCPRES_RTCCLK_1024
RTCCLK/1024
RTC_ATAMP_ASYNCPRES_RTCCLK_2048
RTCCLK/2048
RTC_ATAMP_ASYNCPRES_RTCCLK_4096
RTCCLK/4096
RTC_ATAMP_ASYNCPRES_RTCCLK_8192
RTCCLK/8192
RTC_ATAMP_ASYNCPRES_RTCCLK_16384
RTCCLK/16384
RTC_ATAMP_ASYNCPRES_RTCCLK_32768
RTCCLK/32768

RTCEx_ActiveTamper_Enable Definitions

RTC_ATAMP_ENABLE

RTC_ATAMP_DISABLE

RTCEx_ActiveTamper_Filter Definitions

RTC_ATAMP_FILTER_ENABLE

RTC_ATAMP_FILTER_DISABLE

RTCEx_ActiveTamper_Interrupt Definitions

RTC_ATAMP_INTERRUPT_ENABLE

RTC_ATAMP_INTERRUPT_DISABLE

RTCEx Active Tamper selection Definition

RTC_ATAMP_1

Tamper 1

RTC_ATAMP_2

Tamper 2

RTC_ATAMP_3

Tamper 3

RTC_ATAMP_4

Tamper 4

RTC_ATAMP_5

Tamper 5

RTC_ATAMP_6

Tamper 6

RTC_ATAMP_7

Tamper 7

RTC_ATAMP_8

Tamper 8

Seed size Definitions

RTC_ATAMP_SEED_NB_UINT32

RTCEx Add 1 Second Parameter Definitions

RTC_SHIFTADD1S_RESET

RTC_SHIFTADD1S_SET

RTC Alarm Sub Seconds

RTC_ALARMSUBSECONDBIN_AUTOCLR_NO

The synchronous Binary counter(SS[31:0] in RTC_SSR) is free-running

RTC_ALARMSUBSECONDBIN_AUTOCLR_YES

The synchronous Binary counter (SS[31:0] in RTC_SSR) is running from 0xFFFF FFFF to RTC_ALRMABINR -> SS[31:0] value and is automatically reloaded with 0xFFFF FFFF whenreaching RTC_ALRMABINR -> SS[31:0].

RTC Alarm Sub Seconds with binary or mix mode

RTC_ALARMSUBSECONDBINMASK_ALL

All Alarm SS fields are masked.There is no comparison on sub seconds for Alarm

RTC_ALARMSUBSECONDBINMASK_SS31_1

SS[31:1] are don't care in Alarm comparison. Only SS[0] is compared

RTC_ALARMSUBSECONDBINMASK_SS31_2

SS[31:2] are don't care in Alarm comparison. Only SS[1:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_3

SS[31:3] are don't care in Alarm comparison. Only SS[2:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_4

SS[31:4] are don't care in Alarm comparison. Only SS[3:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_5

SS[31:5] are don't care in Alarm comparison. Only SS[4:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_6

SS[31:6] are don't care in Alarm comparison. Only SS[5:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_7

SS[31:7] are don't care in Alarm comparison. Only SS[6:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_8

SS[31:8] are don't care in Alarm comparison. Only SS[7:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_9

SS[31:9] are don't care in Alarm comparison. Only SS[8:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_10

SS[31:10] are don't care in Alarm comparison. Only SS[9:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_11

SS[31:11] are don't care in Alarm comparison. Only SS[10:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_12

SS[31:12] are don't care in Alarm comparison.Only SS[11:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_13

SS[31:13] are don't care in Alarm comparison. Only SS[12:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_14

SS[31:14] are don't care in Alarm comparison. Only SS[13:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_15

SS[31:15] are don't care in Alarm comparison. Only SS[14:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_16

SS[31:16] are don't care in Alarm comparison. Only SS[15:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_17

SS[31:17] are don't care in Alarm comparison. Only SS[16:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_18

SS[31:18] are don't care in Alarm comparison. Only SS[17:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_19

SS[31:19] are don't care in Alarm comparison. Only SS[18:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_20

SS[31:20] are don't care in Alarm comparison. Only SS[19:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_21

SS[31:21] are don't care in Alarm comparison. Only SS[20:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_22

SS[31:22] are don't care in Alarm comparison. Only SS[21:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_23

SS[31:23] are don't care in Alarm comparison. Only SS[22:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_24

SS[31:24] are don't care in Alarm comparison. Only SS[23:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_25

SS[31:25] are don't care in Alarm comparison. Only SS[24:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_26

SS[31:26] are don't care in Alarm comparison. Only SS[25:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_27

SS[31:27] are don't care in Alarm comparison. Only SS[26:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_28

SS[31:28] are don't care in Alarm comparison. Only SS[27:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_29

SS[31:29] are don't care in Alarm comparison. Only SS[28:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_30

SS[31:30] are don't care in Alarm comparison. Only SS[29:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31

SS[31] is don't care in Alarm comparison. Only SS[30:0] are compared

RTC_ALARMSUBSECONDBINMASK_NONE

SS[31:0] are compared and must match to activate alarm

RTCEx Backup Registers Definition

RTC_BKP_NUMBER

RTC_BKP_DR0

RTC_BKP_DR1

RTC_BKP_DR2

RTC_BKP_DR3

RTC_BKP_DR4

RTC_BKP_DR5

RTC_BKP_DR6

RTC_BKP_DR7

RTC_BKP_DR8

RTC_BKP_DR9

RTC_BKP_DR10

RTC_BKP_DR11

RTC_BKP_DR12

RTC_BKP_DR13

RTC_BKP_DR14

RTC_BKP_DR15

RTC_BKP_DR16

RTC_BKP_DR17

RTC_BKP_DR18

RTC_BKP_DR19

RTC_BKP_DR20

RTC_BKP_DR21

RTC_BKP_DR22

RTC_BKP_DR23

RTC_BKP_DR24

RTC_BKP_DR25

RTC_BKP_DR26

RTC_BKP_DR27

RTC_BKP_DR28

RTC_BKP_DR29

RTC_BKP_DR30

RTC_BKP_DR31

RTCEx Privilege Backup register privilege zone Definition

RTC_PRIVILEGE_BKUP_ZONE_NONE

RTC_PRIVILEGE_BKUP_ZONE_1

RTC_PRIVILEGE_BKUP_ZONE_2

RTC_PRIVILEGE_BKUP_ZONE_ALL

If Binary mode is RTC_BINARY_MIX, the BCD calendar second is incremented

RTC_BINARY_MIX_BCDU_0

The 1s BCD calendar increment is generated each time SS[7:0] = 0

RTC_BINARY_MIX_BCDU_1

The 1s BCD calendar increment is generated each time SS[8:0] = 0

RTC_BINARY_MIX_BCDU_2

The 1s BCD calendar increment is generated each time SS[9:0] = 0

RTC_BINARY_MIX_BCDU_3

The 1s BCD calendar increment is generated each time SS[10:0] = 0

RTC_BINARY_MIX_BCDU_4

The 1s BCD calendar increment is generated each time SS[11:0] = 0

RTC_BINARY_MIX_BCDU_5

The 1s BCD calendar increment is generated each time SS[12:0] = 0

RTC_BINARY_MIX_BCDU_6

The 1s BCD calendar increment is generated each time SS[13:0] = 0

RTC_BINARY_MIX_BCDU_7

The 1s BCD calendar increment is generated each time SS[14:0] = 0

RTC Binary Mode (32-bit free-running counter configuration).

RTC_BINARY_NONE

Free running BCD calendar mode (Binary mode disabled)

RTC_BINARY_ONLY

Free running Binary mode (BCD mode disabled)

RTC_BINARY_MIX

Free running BCD calendar and Binary modes

RTC Calibration

__HAL_RTC_CALIBRATION_OUTPUT_ENABLE

Description:

- Enable the RTC calibration output.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_CALIBRATION_OUTPUT_DISABLE

Description:

- Disable the calibration output.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_CLOCKREF_DETECTION_ENABLE

Description:

- Enable the clock reference detection.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_CLOCKREF_DETECTION_DISABLE

Description:

- Disable the clock reference detection.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_SHIFT_GET_FLAG

Description:

- Get the selected RTC shift operations flag status.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC shift operation Flag is pending or not. This parameter can be:
  - RTC_FLAG_SHPF

Return value:

- The: state of __FLAG__ (TRUE or FALSE)

RTCEx Calib Output selection Definitions

RTC_CALIBOUTPUT_512HZ

RTC_CALIBOUTPUT_1HZ

RTCEx Exported Macros

__HAL_RTC_CLEAR_FLAG

Description:

- Clear the specified RTC pending flag.

Parameters:

- __HANDLE__: specifies the RTC Handle.
- __FLAG__: specifies the flag to check. This parameter can be any combination of the following values:
  - RTC_CLEAR_SSRUF Clear SSR underflow flag
  - RTC_CLEAR_ITSF Clear Internal Time-stamp flag
  - RTC_CLEAR_TSOVF Clear Time-stamp overflow flag
  - RTC_CLEAR_TSF Clear Time-stamp flag
  - RTC_CLEAR_WUTF Clear Wakeup timer flag
  - RTC_CLEAR_ALRBF Clear Alarm B flag
  - RTC_CLEAR_ALRAF Clear Alarm A flag

Return value:

- None

__HAL_RTC_GET_FLAG

Description:

- Check whether the specified RTC flag is set or not.

Parameters:

- __HANDLE__: specifies the RTC Handle.
- __FLAG__: specifies the flag to check. This parameter can be any combination of the following values:
  - RTC_FLAG_RECALPF Recalibration pending Flag
  - RTC_FLAG_INITF Initialization flag
  - RTC_FLAG_RSF Registers synchronization flag
  - RTC_FLAG_INITS Initialization status flag
  - RTC_FLAG_SHPF Shift operation pending flag
  - RTC_FLAG_WUTWF Wakeup timer write flag
  - RTC_FLAG_ITSF Internal Time-stamp flag
  - RTC_FLAG_TSOVF Time-stamp overflow flag
  - RTC_FLAG_TSF Time-stamp flag
  - RTC_FLAG_WUTF Wakeup timer flag
  - RTC_FLAG_ALRBF Alarm B flag
  - RTC_FLAG_ALRAF Alarm A flag

Return value:

- The: state of __FLAG__ (TRUE or FALSE).

RTCEx Flags

RTC_FLAG_TAMP_1

RTC_FLAG_TAMP_2

RTC_FLAG_TAMP_3

RTC_FLAG_TAMP_4

RTC_FLAG_TAMP_5

RTC_FLAG_TAMP_6

RTC_FLAG_TAMP_7

RTC_FLAG_TAMP_ALL

RTC_FLAG_INT_TAMP_1

RTC_FLAG_INT_TAMP_2

RTC_FLAG_INT_TAMP_3

RTC_FLAG_INT_TAMP_4

RTC_FLAG_INT_TAMP_5

RTC_FLAG_INT_TAMP_6

RTC_FLAG_INT_TAMP_7

RTC_FLAG_INT_TAMP_8

RTC_FLAG_INT_TAMP_9

RTC_FLAG_INT_TAMP_11

RTC_FLAG_INT_TAMP_ALL

RTCEx Internal Tamper Interrupt

RTC_IT_INT_TAMP_1

Tamper 1 internal Interrupt

RTC_IT_INT_TAMP_2

Tamper 2 internal Interrupt

RTC_IT_INT_TAMP_3

Tamper 3 internal Interrupt

RTC_IT_INT_TAMP_4

Tamper 4 internal Interrupt

RTC_IT_INT_TAMP_5

Tamper 5 internal Interrupt

RTC_IT_INT_TAMP_6

Tamper 6 internal Interrupt

RTC_IT_INT_TAMP_7

Tamper 7 internal Interrupt

RTC_IT_INT_TAMP_8

Tamper 8 internal Interrupt

RTC_IT_INT_TAMP_9

Tamper 9 internal Interrupt

RTC_IT_INT_TAMP_11

Tamper 11 internal Interrupt

RTC_IT_INT_TAMP_ALL

RTCEx Internal Tamper Pins Definition

RTC_INT_TAMPER_1

RTC_INT_TAMPER_2

RTC_INT_TAMPER_3

RTC_INT_TAMPER_4

RTC_INT_TAMPER_5

RTC_INT_TAMPER_6

RTC_INT_TAMPER_7

RTC_INT_TAMPER_8

RTC_INT_TAMPER_9

RTC_INT_TAMPER_11

RTC_INT_TAMPER_ALL

Private macros to check input parameters

IS_TIMESTAMP_EDGE

IS_RTC_TIMESTAMP_PIN

IS_RTC_TIMESTAMPONTAMPER_DETECTION

IS_RTC_TAMPER_TAMPERDETECTIONOUTPUT

IS_RTC_WAKEUP_CLOCK

IS_RTC_WAKEUP_COUNTER

IS_RTC_SMOOTH_CALIB_PERIOD

IS_RTC_SMOOTH_CALIB_PLUS

IS_RTC_SMOOTH_CALIB_MINUS

IS_RTC_LOW_POWER_CALIB

IS_RTC_TAMPER

IS_RTC_INTERNAL_TAMPER

IS_RTC_TAMPER_TRIGGER

IS_RTC_TAMPER_ERASE_MODE

IS_RTC_TAMPER_MASKFLAG_STATE

IS_RTC_TAMPER_FILTER

IS_RTC_TAMPER_SAMPLING_FREQ

IS_RTC_TAMPER_PRECHARGE_DURATION

IS_RTC_TAMPER_PULLUP_STATE

IS_RTC_TAMPER_TIMESTAMPONTAMPER_DETECTION

IS_RTC_ATAMPER_FILTER

IS_RTC_ATAMPER_OUTPUT_CHANGE_PERIOD

IS_RTC_ATAMPER_ASYNCPRES_RTCCLK

IS_RTC_BKP

IS_RTC_SHIFT_ADD1S

IS_RTC_SHIFT_SUBFS

IS_RTC_CALIB_OUTPUT

IS_RTC_SECURE_FULL

IS_RTC_NONSECURE_FEATURES

IS_TAMP_SECURE_FULL

IS_TAMP_MONOTONIC_CNT_SECURE

IS_RTC_PRIVILEGE_FULL

IS_RTC_PRIVILEGE_FEATURES

IS_TAMP_PRIVILEGE_FULL

IS_TAMP_MONOTONIC_CNT_PRIVILEGE

IS_RTC_PRIVILEGE_BKUP_ZONE

IS_RTC_BINARY_MODE

IS_RTC_BINARY_MIX_BCDU

IS_RTC_ALARM_SUB_SECOND_BINARY_MASK

IS_RTC_ALARMSUBSECONDBIN_AUTOCLR

RTCEx Monotonic Counter Instance Definition

RTC_MONOTONIC_COUNTER_1

Monotonic counter 1

RTCEx Output Selection Definition

RTC_OUTPUT_DISABLE

RTC_OUTPUT_ALARMA

RTC_OUTPUT_ALARMB

RTC_OUTPUT_WAKEUP

RTC_OUTPUT_TAMPER

RTCEx Secure Features Definition

RTC_NONSECURE_FEATURE_NONE

RTC_NONSECURE_FEATURE_INIT
: Initialization

RTC_NONSECURE_FEATURE_CAL
: Calibration

RTC_NONSECURE_FEATURE_TS
: Time stamp

RTC_NONSECURE_FEATURE_WUT
: Wake up timer

RTC_NONSECURE_FEATURE_ALRA
: Alarm A

RTC_NONSECURE_FEATURE_ALRB
: Alarm B

RTC_NONSECURE_FEATURE_ALL

RTCEx Privilege Features Definition

RTC_PRIVILEGE_FEATURE_NONE

RTC_PRIVILEGE_FEATURE_INIT
: Initialization

RTC_PRIVILEGE_FEATURE_CAL
: Calibration

RTC_PRIVILEGE_FEATURE_TS
: Time stamp

RTC_PRIVILEGE_FEATURE_WUT
: Wake up timer

RTC_PRIVILEGE_FEATURE_ALRA
: Alarm A

RTC_PRIVILEGE_FEATURE_ALRB
: Alarm B

RTC_PRIVILEGE_FEATURE_ALL

RTCEx Privilege Full Definition

RTC_PRIVILEGE_FULL_YES

RTC_PRIVILEGE_FULL_NO

RTCEx Secure Definition

RTC_SECURE_FULL_YES
: RTC full secure

RTC_SECURE_FULL_NO
: RTC is not full secure, features can be unsecure. See RTCEx_RTC_NonSecure_Features

RTCEx Smooth calib Low Power Definitions

RTC_LPCAL_SET

Calibration window is 2exp20 ck_apre, which is the required configuration for ultra-low consumption mode.

RTC_LPCAL_RESET

Calibration window is 2exp20 RTCCLK, which is a high-consumption mode. This mode should be set only when less than 32s calibration window is required.

RTCEx Smooth calib period Definitions

RTC_SMOOTHCALIB_PERIOD_32SEC

If RTCCLK = 32768 Hz, Smooth calibration period is 32s, else 2exp20 RTCCLK pulses

RTC_SMOOTHCALIB_PERIOD_16SEC

If RTCCLK = 32768 Hz, Smooth calibration period is 16s, else 2exp19 RTCCLK pulses

RTC_SMOOTHCALIB_PERIOD_8SEC

If RTCCLK = 32768 Hz, Smooth calibration period is 8s, else 2exp18 RTCCLK pulses

RTCEx Smooth calib Plus pulses Definitions

RTC_SMOOTHCALIB_PLUSPULSES_SET

The number of RTCCLK pulses added during a X -second window = Y - CALM[8:0] with Y = 512, 256, 128 when X = 32, 16, 8

RTC_SMOOTHCALIB_PLUSPULSES_RESET

The number of RTCCLK pulses subbstited during a 32-second window = CALM[8:0]

RTC SSR Underflow

__HAL_RTC_SSRU_ENABLE_IT

Description:

- Enable the RTC SSRU interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC SSRU interrupt sources to be enabled. This parameter can be:
  - RTC_IT_SSRU SSRU interrupt

Return value:

- None

__HAL_RTC_SSRU_DISABLE_IT

Description:

- Disable the RTC SSRU interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC SSRU interrupt sources to be disabled. This parameter can be:
  - RTC_IT_SSRU SSRU interrupt

Return value:

- None

__HAL_RTC_SSRU_GET_IT

Description:

- Check whether the specified RTC SSRU interrupt has occurred or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC SSRU interrupt to check. This parameter can be:
  - RTC_IT_SSRU SSRU interrupt

Return value:

- The: state of __INTERRUPT__ (TRUE or FALSE)

__HAL_RTC_SSRU_GET_IT_SOURCE

Description:

- Check whether the specified RTC SSRU interrupt has been enabled or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC SSRU interrupt sources to check. This parameter can be:
  - RTC_IT_SSRU SSRU interrupt

Return value:

- The: state of __INTERRUPT__ (TRUE or FALSE)

__HAL_RTC_SSRU_GET_FLAG

Description:

- Get the selected RTC SSRU's flag status.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC SSRU Flag is pending or not. This parameter can be:
  - RTC_FLAG_SSRUF

Return value:

- The: state of __FLAG__ (TRUE or FALSE)

__HAL_RTC_SSRU_CLEAR_FLAG

Description:

- Clear the RTC SSRU's pending flags.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC SSRU Flag to clear. This parameter can be:
  - RTC_FLAG_SSRUF

Return value:

- None

RTCEx tamper

__HAL_RTC_TAMPER_ENABLE

Description:

- Enable the TAMP Tamper input detection.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __TAMPER__: specifies the RTC Tamper source to be enabled. This parameter can be any combination of the following values:
  - RTC_TAMPER_ALL: All tampers
  - RTC_TAMPER_1: Tamper1
  - RTC_TAMPER_2: Tamper2
  - RTC_TAMPER_3: Tamper3
  - RTC_TAMPER_4: Tamper4
  - RTC_TAMPER_5: Tamper5
  - RTC_TAMPER_6: Tamper6
  - RTC_TAMPER_7: Tamper7
  - RTC_TAMPER_8: Tamper8

Return value:

- None

__HAL_RTC_TAMPER_DISABLE

Description:

- Disable the TAMP Tamper input detection.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __TAMPER__: specifies the RTC Tamper sources to be enabled. This parameter can be any combination of the following values:
  - RTC_TAMPER_ALL: All tampers
  - RTC_TAMPER_1: Tamper1
  - RTC_TAMPER_2: Tamper2
  - RTC_TAMPER_3: Tamper3
  - RTC_TAMPER_4: Tamper4
  - RTC_TAMPER_5: Tamper5
  - RTC_TAMPER_6: Tamper6
  - RTC_TAMPER_7: Tamper7
  - RTC_TAMPER_8: Tamper8

__HAL_RTC_TAMPER_ENABLE_IT

Description:

- Enable the TAMP Tamper interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Tamper interrupt sources to be enabled. This parameter can be any combination of the following values:
  - RTC_IT_TAMP_ALL: All tampers interrupts
  - RTC_IT_TAMP_1: Tamper1 interrupt
  - RTC_IT_TAMP_2: Tamper2 interrupt
  - RTC_IT_TAMP_3: Tamper3 interrupt
  - RTC_IT_TAMP_4: Tamper4 interrupt
  - RTC_IT_TAMP_5: Tamper5 interrupt
  - RTC_IT_TAMP_6: Tamper6 interrupt
  - RTC_IT_TAMP_7: Tamper7 interrupt
  - RTC_IT_TAMP_8: Tamper8 interrupt

Return value:

- None

__HAL_RTC_TAMPER_DISABLE_IT

Description:

- Disable the TAMP Tamper interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Tamper interrupt sources to be disabled. This parameter can be any combination of the following values:
  - RTC_IT_TAMP_ALL: All tampers interrupts
  - RTC_IT_TAMP_1: Tamper1 interrupt
  - RTC_IT_TAMP_2: Tamper2 interrupt
  - RTC_IT_TAMP_3: Tamper3 interrupt
  - RTC_IT_TAMP_4: Tamper4 interrupt
  - RTC_IT_TAMP_5: Tamper5 interrupt
  - RTC_IT_TAMP_6: Tamper6 interrupt
  - RTC_IT_TAMP_7: Tamper7 interrupt
  - RTC_IT_TAMP_8: Tamper8 interrupt

Return value:

- None

__HAL_RTC_TAMPER_GET_IT

Description:

- Check whether the specified RTC Tamper interrupt has occurred or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Tamper interrupt to check. This parameter can be:
  - RTC_IT_TAMP_ALL: All tampers interrupts
  - RTC_IT_TAMP_1: Tamper1 interrupt
  - RTC_IT_TAMP_2: Tamper2 interrupt
  - RTC_IT_TAMP_3: Tamper3 interrupt
  - RTC_IT_TAMP_4: Tamper4 interrupt
  - RTC_IT_TAMP_5: Tamper5 interrupt
  - RTC_IT_TAMP_6: Tamper6 interrupt
  - RTC_IT_TAMP_7: Tamper7 interrupt
  - RTC_IT_TAMP_8: Tamper8 interrupt
  - RTC_IT_INT_TAMP_ALL: All Internal Tamper interrupts
  - RTC_IT_INT_TAMP_1: Internal Tamper1 interrupt
  - RTC_IT_INT_TAMP_2: Internal Tamper2 interrupt
  - RTC_IT_INT_TAMP_3: Internal Tamper3 interrupt
  - RTC_IT_INT_TAMP_4: Internal Tamper4 interrupt
  - RTC_IT_INT_TAMP_5: Internal Tamper5 interrupt
  - RTC_IT_INT_TAMP_6: Internal Tamper6 interrupt
  - RTC_IT_INT_TAMP_7: Internal Tamper7 interrupt
  - RTC_IT_INT_TAMP_8: Internal Tamper8 interrupt
  - RTC_IT_INT_TAMP_9: Internal Tamper9 interrupt
  - RTC_IT_INT_TAMP_11: Internal Tamper11 interrupt

Return value:

- The: state of __INTERRUPT__ (TRUE or FALSE)

__HAL_RTC_TAMPER_GET_IT_SOURCE

Description:

- Check whether the specified RTC Tamper interrupt has been enabled or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Tamper interrupt source to check. This parameter can be:
  - RTC_IT_TAMP_ALL: All tampers interrupts
  - RTC_IT_TAMP_1: Tamper1 interrupt
  - RTC_IT_TAMP_2: Tamper2 interrupt
  - RTC_IT_TAMP_3: Tamper3 interrupt
  - RTC_IT_TAMP_4: Tamper4 interrupt
  - RTC_IT_TAMP_5: Tamper5 interrupt
  - RTC_IT_TAMP_6: Tamper6 interrupt
  - RTC_IT_TAMP_7: Tamper7 interrupt
  - RTC_IT_TAMP_8: Tamper8 interrupt
  - RTC_IT_INT_TAMP_ALL: All internal tampers interrupts
  - RTC_IT_INT_TAMP_1: Internal Tamper1 interrupt
  - RTC_IT_INT_TAMP_2: Internal Tamper2 interrupt
  - RTC_IT_INT_TAMP_3: Internal Tamper3 interrupt
  - RTC_IT_INT_TAMP_4: Internal Tamper4 interrupt
  - RTC_IT_INT_TAMP_5: Internal Tamper5 interrupt
  - RTC_IT_INT_TAMP_6: Internal Tamper6 interrupt
  - RTC_IT_INT_TAMP_7: Internal Tamper7 interrupt
  - RTC_IT_INT_TAMP_8: Internal Tamper8 interrupt
  - RTC_IT_INT_TAMP_9: Internal Tamper9 interrupt
  - RTC_IT_INT_TAMP_11: Internal Tamper11 interrupt

Return value:

- The: state of __INTERRUPT__ (TRUE or FALSE)

__HAL_RTC_TAMPER_GET_FLAG

Description:

- Get the selected RTC Tampers flag status.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC Tamper Flag is pending or not. This parameter can be:
  - RTC_FLAG_TAMP_ALL: All tampers flag
  - RTC_FLAG_TAMP_1: Tamper1 flag
  - RTC_FLAG_TAMP_2: Tamper2 flag
  - RTC_FLAG_TAMP_3: Tamper3 flag
  - RTC_FLAG_TAMP_4: Tamper4 flag
  - RTC_FLAG_TAMP_5: Tamper5 flag
  - RTC_FLAG_TAMP_6: Tamper6 flag
  - RTC_FLAG_TAMP_7: Tamper7 flag
  - RTC_FLAG_TAMP_8: Tamper8 flag
  - RTC_FLAG_INT_TAMP_1: Internal Tamper1 flag
  - RTC_FLAG_INT_TAMP_2: Internal Tamper2 flag
  - RTC_FLAG_INT_TAMP_3: Internal Tamper3 flag
  - RTC_FLAG_INT_TAMP_4: Internal Tamper4 flag
  - RTC_FLAG_INT_TAMP_5: Internal Tamper5 flag
  - RTC_FLAG_INT_TAMP_6: Internal Tamper6 flag
  - RTC_FLAG_INT_TAMP_7: Internal Tamper7 flag
  - RTC_FLAG_INT_TAMP_8: Internal Tamper8 flag
  - RTC_FLAG_INT_TAMP_9: Internal Tamper9 flag
  - RTC_FLAG_INT_TAMP_11: Internal Tamper11 flag

Return value:

- The: state of __FLAG__ (TRUE or FALSE)

__HAL_RTC_TAMPER_CLEAR_FLAG

Description:

- Clear the RTC Tamper's pending flags.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC Tamper Flag to clear. This parameter can be:
  - RTC_FLAG_TAMP_ALL: All tampers flag
  - RTC_FLAG_TAMP_1: Tamper1 flag
  - RTC_FLAG_TAMP_2: Tamper2 flag
  - RTC_FLAG_TAMP_3: Tamper3 flag
  - RTC_FLAG_TAMP_4: Tamper4 flag
  - RTC_FLAG_TAMP_5: Tamper5 flag
  - RTC_FLAG_TAMP_6: Tamper6 flag
  - RTC_FLAG_TAMP_7: Tamper7 flag
  - RTC_FLAG_TAMP_8: Tamper8 flag
  - RTC_FLAG_INT_TAMP_ALL: All Internal Tamper flags
  - RTC_FLAG_INT_TAMP_1: Internal Tamper1 flag
  - RTC_FLAG_INT_TAMP_2: Internal Tamper2 flag
  - RTC_FLAG_INT_TAMP_3: Internal Tamper3 flag
  - RTC_FLAG_INT_TAMP_4: Internal Tamper4 flag
  - RTC_FLAG_INT_TAMP_5: Internal Tamper5 flag
  - RTC_FLAG_INT_TAMP_6: Internal Tamper6 flag
  - RTC_FLAG_INT_TAMP_7: Internal Tamper7 flag
  - RTC_FLAG_INT_TAMP_8: Internal Tamper8 flag
  - RTC_FLAG_INT_TAMP_9: Internal Tamper9 flag
  - RTC_FLAG_INT_TAMP_11: Internal Tamper11 flag

Return value:

- None

RTCEx Tamper detection output Definitions

RTC_TAMPERDETECTIONOUTPUT_DISABLE

Tamper detection output disable on TAMPALRM

RTC_TAMPERDETECTIONOUTPUT_ENABLE

Tamper detection output enable on TAMPALRM

RTCEx Tamper EraseBackUp

RTC_TAMPER_ERASE_BACKUP_ENABLE

RTC_TAMPER_ERASE_BACKUP_DISABLE

RTCEx Tamper Filter

RTC_TAMPERFILTER_DISABLE

Tamper filter is disabled

RTC_TAMPERFILTER_2SAMPLE

Tamper is activated after 2 consecutive samples at the active level

RTC_TAMPERFILTER_4SAMPLE

Tamper is activated after 4 consecutive samples at the active level

RTC_TAMPERFILTER_8SAMPLE

Tamper is activated after 8 consecutive samples at the active level

RTCEx Tamper Interrupt

RTC_IT_TAMP_1

Tamper 1 Interrupt

RTC_IT_TAMP_2

Tamper 2 Interrupt

RTC_IT_TAMP_3

Tamper 3 Interrupt

RTC_IT_TAMP_4

Tamper 4 Interrupt

RTC_IT_TAMP_5

Tamper 5 Interrupt

RTC_IT_TAMP_6

Tamper 6 Interrupt

RTC_IT_TAMP_7

Tamper 7 Interrupt

RTC_IT_TAMP_ALL

RTCEx Tampers maskable number

RTC_TAMPER_MASKABLE_NB

RTCEx Tamper MaskFlag

RTC_TAMPERMASK_FLAG_DISABLE

RTC_TAMPERMASK_FLAG_ENABLE

RTCEx Tamper Pins Definition

RTC_TAMPER_1

RTC_TAMPER_2

RTC_TAMPER_3

RTC_TAMPER_4

RTC_TAMPER_5

RTC_TAMPER_6

RTC_TAMPER_7

RTC_TAMPER_ALL

RTCEx Tamper Pin Precharge Duration

RTC_TAMPERPRECHARGEDURATION_1RTCCLK

Tamper pins are pre-charged before sampling during 1 RTCCLK cycle

RTC_TAMPERPRECHARGEDURATION_2RTCCLK

Tamper pins are pre-charged before sampling during 2 RTCCLK cycles

RTC_TAMPERPRECHARGEDURATION_4RTCCLK

Tamper pins are pre-charged before sampling during 4 RTCCLK cycles

RTC_TAMPERPRECHARGEDURATION_8RTCCLK

Tamper pins are pre-charged before sampling during 8 RTCCLK cycles

RTCEx Tamper Pull UP

RTC_TAMPER_PULLUP_ENABLE

Tamper pins are pre-charged before sampling

RTC_TAMPER_PULLUP_DISABLE

Tamper pins pre-charge is disabled

RTCEx Tamper Sampling Frequencies

RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV32768

Each of the tamper inputs are sampled with a frequency = RTCCLK / 32768

RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV16384

Each of the tamper inputs are sampled with a frequency = RTCCLK / 16384

RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV8192

Each of the tamper inputs are sampled with a frequency = RTCCLK / 8192

RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV4096

Each of the tamper inputs are sampled with a frequency = RTCCLK / 4096

RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV2048

Each of the tamper inputs are sampled with a frequency = RTCCLK / 2048

RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV1024

Each of the tamper inputs are sampled with a frequency = RTCCLK / 1024

RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV512

Each of the tamper inputs are sampled with a frequency = RTCCLK / 512

RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV256

Each of the tamper inputs are sampled with a frequency = RTCCLK / 256

RTCEx Tamper TimeStamp On Tamper Detection Definitions

RTC_TIMESTAMPONTAMPERDETECTION_DISABLE

TimeStamp on Tamper Detection event is not saved

RTC_TIMESTAMPONTAMPERDETECTION_ENABLE

TimeStamp on Tamper Detection event saved

RTCEx Tamper Trigger

RTC_TAMPERTRIGGER_RISINGEDGE

Warning : Filter must be RTC_TAMPERFILTER_DISABLE

RTC_TAMPERTRIGGER_FALLINGEDGE

Warning : Filter must be RTC_TAMPERFILTER_DISABLE

RTC_TAMPERTRIGGER_LOWLEVEL

Warning : Filter must not be RTC_TAMPERFILTER_DISABLE

RTC_TAMPERTRIGGER_HIGHLEVEL

Warning : Filter must not be RTC_TAMPERFILTER_DISABLE

RTCEx TAMP Device Secrets Erase Configuration Definition

TAMP_DEVICESECRETS_ERASE_NONE

No Erase

TAMP_DEVICESECRETS_ERASE_BKPSRAM

Backup SRAM

RTCEx TAMP Monotonic Counter Privilege Definition

TAMP_MONOTONIC_CNT_PRIVILEGE_YES

TAMP_MONOTONIC_CNT_PRIVILEGE_NO

RTCEx TAMP Monotonic Counter Secure Definition

TAMP_MONOTONIC_CNT_SECURE_YES

TAMPER Monotonic Counter secure

TAMP_MONOTONIC_CNT_SECURE_NO

TAMPER Monotonic Counter is not secure

RTCEx TAMP security Definition

TAMP_PRIVILEGE_FULL_YES

TAMP_PRIVILEGE_FULL_NO

RTCEx TAMP Secure Definition

TAMP_SECURE_FULL_YES

TAMPER full secure

TAMP_SECURE_FULL_NO

TAMPER is not secure

RTC Timestamp

__HAL_RTC_TIMESTAMP_ENABLE

Description:

- Enable the RTC TimeStamp peripheral.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_TIMESTAMP_DISABLE

Description:

- Disable the RTC TimeStamp peripheral.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_TIMESTAMP_ENABLE_IT

Description:

- Enable the RTC TimeStamp interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC TimeStamp interrupt source to be enabled. This parameter can be:
  - RTC_IT_TS TimeStamp interrupt

Return value:

- None

__HAL_RTC_TIMESTAMP_DISABLE_IT

Description:

- Disable the RTC TimeStamp interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC TimeStamp interrupt source to be disabled. This parameter can be:
  - RTC_IT_TS TimeStamp interrupt

Return value:

- None

__HAL_RTC_TIMESTAMP_GET_IT

Description:

- Check whether the specified RTC TimeStamp interrupt has occurred or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC TimeStamp interrupt to check. This parameter can be:
  - RTC_IT_TS TimeStamp interrupt

Return value:

- The: state of __INTERRUPT__ (TRUE or FALSE).

__HAL_RTC_TIMESTAMP_GET_IT_SOURCE

Description:

- Check whether the specified RTC Time Stamp interrupt has been enabled or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Time Stamp interrupt source to check. This parameter can be:
  - RTC_IT_TS TimeStamp interrupt

Return value:

- The: state of __INTERRUPT__ (TRUE or FALSE).

__HAL_RTC_TIMESTAMP_GET_FLAG

Description:

- Get the selected RTC TimeStamps flag status.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC TimeStamp Flag is pending or not. This parameter can be:
  - RTC_FLAG_TSF
  - RTC_FLAG_TSOVF

Return value:

- The: state of __FLAG__ (TRUE or FALSE) or 255 if invalid parameter.

__HAL_RTC_TIMESTAMP_CLEAR_FLAG

Description:

- Clear the RTC Time Stamps pending flags.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC TimeStamp Flag to clear. This parameter can be:
  - RTC_FLAG_TSF
  - RTC_FLAG_TSOVF

Return value:

- None

__HAL_RTC_INTERNAL_TIMESTAMP_ENABLE

Description:

- Enable the RTC internal TimeStamp peripheral.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_INTERNAL_TIMESTAMP_DISABLE

Description:

- Disable the RTC internal TimeStamp peripheral.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_INTERNAL_TIMESTAMP_GET_FLAG

Description:

- Get the selected RTC Internal Time Stamps flag status.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC Internal Time Stamp Flag is pending or not. This parameter can be:
  - RTC_FLAG_ITSF

Return value:

- None

__HAL_RTC_INTERNAL_TIMESTAMP_CLEAR_FLAG

Description:

- Clear the RTC Internal Time Stamps pending flags.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC Internal Time Stamp Flag source to clear. This parameter can be:
  - RTC_FLAG_ITSF

Return value:

- None

__HAL_RTC_TAMPTS_ENABLE

Description:

- Enable the RTC TimeStamp on Tamper detection.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_TAMPTS_DISABLE

Description:

- Disable the RTC TimeStamp on Tamper detection.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_TAMPOE_ENABLE

Description:

- Enable the RTC Tamper detection output.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_TAMPOE_DISABLE

Description:

- Disable the RTC Tamper detection output.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

RTCEx TimeStamp Pin Selection

RTC_TIMESTAMPPIN_DEFAULT

RTCEx Time Stamp Edges definition

RTC_TIMESTAMPEDGE_RISING

RTC_TIMESTAMPEDGE_FALLING

RTC WakeUp Timer

__HAL_RTC_WAKEUPTIMER_ENABLE

Description:

- Enable the RTC WakeUp Timer peripheral.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_WAKEUPTIMER_DISABLE

Description:

- Disable the RTC WakeUp Timer peripheral.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_WAKEUPTIMER_ENABLE_IT

Description:

- Enable the RTC WakeUpTimer interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC WakeUpTimer interrupt sources to be enabled. This parameter can be:
  - RTC_IT_WUT WakeUpTimer interrupt

Return value:

- None

__HAL_RTC_WAKEUPTIMER_DISABLE_IT

Description:

- Disable the RTC WakeUpTimer interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC WakeUpTimer interrupt sources to be disabled. This parameter can be:
  - RTC_IT_WUT WakeUpTimer interrupt

Return value:

- None

__HAL_RTC_WAKEUPTIMER_GET_IT

Description:

- Check whether the specified RTC WakeUpTimer interrupt has occurred or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC WakeUpTimer interrupt to check. This parameter can be:
  - RTC_IT_WUT WakeUpTimer interrupt

Return value:

- The: state of __INTERRUPT__ (TRUE or FALSE).

__HAL_RTC_WAKEUPTIMER_GET_IT_SOURCE

Description:

- Check whether the specified RTC Wake Up timer interrupt has been enabled or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Wake Up timer interrupt sources to check. This parameter can be:
  - RTC_IT_WUT WakeUpTimer interrupt

Return value:

- The: state of __INTERRUPT__ (TRUE or FALSE).

__HAL_RTC_WAKEUPTIMER_GET_FLAG

Description:

- Get the selected RTC WakeUpTimers flag status.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC WakeUpTimer Flag is pending or not. This parameter can be:
  - RTC_FLAG_WUTF
  - RTC_FLAG_WUTWF

Return value:

- The: state of __FLAG__ (TRUE or FALSE).

__HAL_RTC_WAKEUPTIMER_CLEAR_FLAG

Description:

- Clear the RTC Wake Up timers pending flags.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC WakeUpTimer Flag to clear. This parameter can be:
  - RTC_FLAG_WUTF

Return value:

- None

RTCEx Wakeup Timer Definitions

RTC_WAKEUPCLOCK_RTCCLK_DIV16

RTC_WAKEUPCLOCK_RTCCLK_DIV8

RTC_WAKEUPCLOCK_RTCCLK_DIV4

RTC_WAKEUPCLOCK_RTCCLK_DIV2

RTC_WAKEUPCLOCK_CK_SPRE_16BITS

RTC_WAKEUPCLOCK_CK_SPRE_17BITS

63 HAL SAI Generic Driver

63.1 SAI Firmware driver registers structures

63.1.1 SAI_PdmInitTypeDef

SAI_PdmInitTypeDef is defined in the stm32n6xx_hal_sai.h

Data Fields

- ***FunctionalState Activation***
- ***uint32_t MicPairsNbr***
- ***uint32_t ClockEnable***

Field Documentation

- ***FunctionalState SAI_PdmInitTypeDef::Activation***
  Enable/disable PDM interface
- ***uint32_t SAI_PdmInitTypeDef::MicPairsNbr***
  Specifies the number of microphone pairs used. This parameter must be a number between Min_Data = 1 and Max_Data = 3.
- ***uint32_t SAI_PdmInitTypeDef::ClockEnable***
  Specifies which clock must be enabled. This parameter can be a values combination of ***SAI_PDM_ClockEnable***

63.1.2 SAI_InitTypeDef

SAI_InitTypeDef is defined in the stm32n6xx_hal_sai.h

Data Fields

- ***uint32_t AudioMode***
- ***uint32_t Synchro***
- ***uint32_t SynchroExt***
- ***uint32_t MckOutput***
- ***uint32_t OutputDrive***
- ***uint32_t NoDivider***
- ***uint32_t FIFOThreshold***
- ***uint32_t AudioFrequency***
- ***uint32_t Mckdiv***
- ***uint32_t MckOverSampling***
- ***uint32_t MonoStereoMode***
- ***uint32_t CompandingMode***
- ***uint32_t TriState***
- ***SAI_PdmInitTypeDef PdmInit***
- ***uint32_t Protocol***
- ***uint32_t DataSize***
- ***uint32_t FirstBit***
- ***uint32_t ClockStrobing***

Field Documentation

- ***uint32_t SAI_InitTypeDef::AudioMode***
  Specifies the SAI Block audio Mode. This parameter can be a value of ***SAI_Block_Mode***
- ***uint32_t SAI_InitTypeDef::Synchro***
  Specifies SAI Block synchronization This parameter can be a value of ***SAI_Block_Synchronization***

- ***uint32_t SAI_InitTypeDef::SynchroExt***
  Specifies SAI external output synchronization, this setup is common for BlockA and BlockB This parameter can be a value of ***SAI_Block_SyncExt***
  Note:
  – If both audio blocks of same SAI are used, this parameter has to be set to the same value for each audio block
- ***uint32_t SAI_InitTypeDef::MckOutput***
  Specifies whether master clock output will be generated or not. This parameter can be a value of ***SAI_Block_MckOutput***
- ***uint32_t SAI_InitTypeDef::OutputDrive***
  Specifies when SAI Block outputs are driven. This parameter can be a value of ***SAI_Block_Output_Drive***
  Note:
  – This value has to be set before enabling the audio block but after the audio block configuration.
- ***uint32_t SAI_InitTypeDef::NoDivider***
  Specifies whether master clock will be divided or not. This parameter can be a value of ***SAI_Block_NoDivider***
  Note:
  – If bit NODIV in the SAI_xCR1 register is cleared, the frame length should be aligned to a number equal to a power of 2, from 8 to 256. If bit NODIV in the SAI_xCR1 register is set, the frame length can take any of the values from 8 to 256.
- ***uint32_t SAI_InitTypeDef::FIFOThreshold***
  Specifies SAI Block FIFO threshold. This parameter can be a value of ***SAI_Block_Fifo_Threshold***
- ***uint32_t SAI_InitTypeDef::AudioFrequency***
  Specifies the audio frequency sampling. This parameter can be a value of ***SAI_Audio_Frequency***
- ***uint32_t SAI_InitTypeDef::Mckdiv***
  Specifies the master clock divider. This parameter must be a number between Min_Data = 0 and Max_Data = 63.
  Note:
  – This parameter is used only if AudioFrequency is set to SAI_AUDIO_FREQUENCY_MCKDIV otherwise it is internally computed.
- ***uint32_t SAI_InitTypeDef::MckOverSampling***
  Specifies the master clock oversampling. This parameter can be a value of ***SAI_Block_Mck_OverSampling***
- ***uint32_t SAI_InitTypeDef::MonoStereoMode***
  Specifies if the mono or stereo mode is selected. This parameter can be a value of ***SAI_Mono_Stereo_Mode***
- ***uint32_t SAI_InitTypeDef::CompandingMode***
  Specifies the companding mode type. This parameter can be a value of ***SAI_Block_Companding_Mode***
- ***uint32_t SAI_InitTypeDef::TriState***
  Specifies the companding mode type. This parameter can be a value of ***SAI_TRIState_Management***
- ***SAI_PdmInitTypeDef SAI_InitTypeDef::PdmInit***
  Specifies the PDM configuration.
- ***uint32_t SAI_InitTypeDef::Protocol***
  Specifies the SAI Block protocol. This parameter can be a value of ***SAI_Block_Protocol***
- ***uint32_t SAI_InitTypeDef::DataSize***
  Specifies the SAI Block data size. This parameter can be a value of ***SAI_Block_Data_Size***
- ***uint32_t SAI_InitTypeDef::FirstBit***
  Specifies whether data transfers start from MSB or LSB bit. This parameter can be a value of ***SAI_Block_MSB_LSB_transmission***
- ***uint32_t SAI_InitTypeDef::ClockStrobing***
  Specifies the SAI Block clock strobing edge sensitivity. This parameter can be a value of ***SAI_Block_Clock_Strobing***

63.1.3 SAI_FrameInitTypeDef

SAI_FrameInitTypeDef is defined in the stm32n6xx_hal_sai.h

Data Fields

- ***uint32_t FrameLength***
- ***uint32_t ActiveFrameLength***
- ***uint32_t FSDefinition***
- ***uint32_t FSPolarity***
- ***uint32_t FSOffset***

Field Documentation

- ***uint32_t SAI_FrameInitTypeDef::FrameLength***
  Specifies the Frame length, the number of SCK clocks for each audio frame. This parameter must be a number between Min_Data = 8 and Max_Data = 256.
  Note:
  – If master clock MCLK_x pin is declared as an output, the frame length should be aligned to a number equal to power of 2 in order to keep in an audio frame, an integer number of MCLK pulses by bit Clock.
- ***uint32_t SAI_FrameInitTypeDef::ActiveFrameLength***
  Specifies the Frame synchronization active level length. This Parameter specifies the length in number of bit clock (SCK + 1) of the active level of FS signal in audio frame. This parameter must be a number between Min_Data = 1 and Max_Data = 128
- ***uint32_t SAI_FrameInitTypeDef::FSDefinition***
  Specifies the Frame synchronization definition. This parameter can be a value of ***SAI_Block_FS_Definition***
- ***uint32_t SAI_FrameInitTypeDef::FSPolarity***
  Specifies the Frame synchronization Polarity. This parameter can be a value of ***SAI_Block_FS_Polarity***
- ***uint32_t SAI_FrameInitTypeDef::FSOffset***
  Specifies the Frame synchronization Offset. This parameter can be a value of ***SAI_Block_FS_Offset***

63.1.4 SAI_SlotInitTypeDef

SAI_SlotInitTypeDef is defined in the stm32n6xx_hal_sai.h

Data Fields

- ***uint32_t FirstBitOffset***
- ***uint32_t SlotSize***
- ***uint32_t SlotNumber***
- ***uint32_t SlotActive***

Field Documentation

- ***uint32_t SAI_SlotInitTypeDef::FirstBitOffset***
  Specifies the position of first data transfer bit in the slot. This parameter must be a number between Min_Data = 0 and Max_Data = 24
- ***uint32_t SAI_SlotInitTypeDef::SlotSize***
  Specifies the Slot Size. This parameter can be a value of ***SAI_Block_Slot_Size***
- ***uint32_t SAI_SlotInitTypeDef::SlotNumber***
  Specifies the number of slot in the audio frame. This parameter must be a number between Min_Data = 1 and Max_Data = 16
- ***uint32_t SAI_SlotInitTypeDef::SlotActive***
  Specifies the slots in audio frame that will be activated. This parameter can be a value of ***SAI_Block_Slot_Active***

63.1.5 __SAI_HandleTypeDef

__SAI_HandleTypeDef is defined in the stm32n6xx_hal_sai.h

Data Fields

- ***SAI_Block_TypeDef * Instance***
- ***SAI_InitTypeDef Init***
- ***SAI_FrameInitTypeDef FrameInit***
- ***SAI_SlotInitTypeDef SlotInit***
- ***uint8_t * pBuffPtr***
- ***uint16_t XferSize***

- ***uint16_t XferCount***
- ***DMA_HandleTypeDef * hdmatx***
- ***DMA_HandleTypeDef * hdmarx***
- ***SAIcallback mutecallback***
- ***void(* InterruptServiceRoutine***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_SAI_StateTypeDef State***
- ***__IO uint32_t ErrorCode***
- ***void(* RxCpltCallback***
- ***void(* RxHalfCpltCallback***
- ***void(* TxCpltCallback***
- ***void(* TxHalfCpltCallback***
- ***void(* ErrorCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***SAI_Block_TypeDef* __SAI_HandleTypeDef::Instance***
  SAI Blockx registers base address
- ***SAI_InitTypeDef __SAI_HandleTypeDef::Init***
  SAI communication parameters
- ***SAI_FrameInitTypeDef __SAI_HandleTypeDef::FrameInit***
  SAI Frame configuration parameters
- ***SAI_SlotInitTypeDef __SAI_HandleTypeDef::SlotInit***
  SAI Slot configuration parameters
- ***uint8_t* __SAI_HandleTypeDef::pBuffPtr***
  Pointer to SAI transfer Buffer
- ***uint16_t __SAI_HandleTypeDef::XferSize***
  SAI transfer size
- ***uint16_t __SAI_HandleTypeDef::XferCount***
  SAI transfer counter
- ***DMA_HandleTypeDef* __SAI_HandleTypeDef::hdmatx***
  SAI Tx DMA handle parameters
- ***DMA_HandleTypeDef* __SAI_HandleTypeDef::hdmarx***
  SAI Rx DMA handle parameters
- ***SAIcallback __SAI_HandleTypeDef::mutecallback***
  SAI mute callback
- ***void(* __SAI_HandleTypeDef::InterruptServiceRoutine)(struct __SAI_HandleTypeDef *hsai)***
- ***HAL_LockTypeDef __SAI_HandleTypeDef::Lock***
  SAI locking object
- ***__IO HAL_SAI_StateTypeDef __SAI_HandleTypeDef::State***
  SAI communication state
- ***__IO uint32_t __SAI_HandleTypeDef::ErrorCode***
  SAI Error code
- ***void(* __SAI_HandleTypeDef::RxCpltCallback)(struct __SAI_HandleTypeDef *hsai)***
  SAI receive complete callback
- ***void(* __SAI_HandleTypeDef::RxHalfCpltCallback)(struct __SAI_HandleTypeDef *hsai)***
  SAI receive half complete callback
- ***void(* __SAI_HandleTypeDef::TxCpltCallback)(struct __SAI_HandleTypeDef *hsai)***
  SAI transmit complete callback
- ***void(* __SAI_HandleTypeDef::TxHalfCpltCallback)(struct __SAI_HandleTypeDef *hsai)***
  SAI transmit half complete callback
- ***void(* __SAI_HandleTypeDef::ErrorCallback)(struct __SAI_HandleTypeDef *hsai)***
  SAI error callback

- ***void(* __SAI_HandleTypeDef::MspInitCallback)(struct __SAI_HandleTypeDef *hsai)***
  SAI MSP init callback
- ***void(* __SAI_HandleTypeDef::MspDeInitCallback)(struct __SAI_HandleTypeDef *hsai)***
  SAI MSP de-init callback

63.2 SAI Firmware driver API description

The following section lists the various functions of the SAI library.

63.2.1 How to use this driver

The SAI HAL driver can be used as follows:

1. Declare a SAI_HandleTypeDef handle structure (eg. SAI_HandleTypeDef hsai).
2. Initialize the SAI low level resources by implementing the HAL_SAI_MspInit() API:
   a. Enable the SAI interface clock.
   b. SAI pins configuration:
      - Enable the clock for the SAI GPIOs.
      - Configure these SAI pins as alternate function pull-up.
   c. NVIC configuration if you need to use interrupt process (HAL_SAI_Transmit_IT() and HAL_SAI_Receive_IT() APIs):
      - Configure the SAI interrupt priority.
      - Enable the NVIC SAI IRQ handle.
   d. DMA Configuration if you need to use DMA process (HAL_SAI_Transmit_DMA() and HAL_SAI_Receive_DMA() APIs):
      - Declare a DMA handle structure for the Tx/Rx stream.
      - Enable the DMAx interface clock.
      - Configure the declared DMA handle structure with the required Tx/Rx parameters.
      - Configure the DMA Tx/Rx Stream.
      - Associate the initialized DMA handle to the SAI DMA Tx/Rx handle.
      - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx/Rx Stream.
3. The initialization can be done by two ways
   a. Expert mode : Initialize the structures Init, FrameInit and SlotInit and call HAL_SAI_Init().
   b. Simplified mode : Initialize the high part of Init Structure and call HAL_SAI_InitProtocol().

Note: The specific SAI interrupts (FIFO request and Overrun underrun interrupt) will be managed using the macros __HAL_SAI_ENABLE_IT() and __HAL_SAI_DISABLE_IT() inside the transmit and receive process.

Note: Make sure that either:

- *PLLSAI1CLK output is configured or*
- *PLLSAI2CLK output is configured or*
- *PLLSAI3CLK output is configured or*
- *External clock source is configured after setting correctly the define constant EXTERNAL_SAI1_CLOCK_VALUE or EXTERNAL_SAI2_CLOCK_VALUE in the stm32n6xx_hal_conf.h file.*

Note: In master Tx mode: enabling the audio block immediately generates the bit clock for the external slaves even if there is no data in the FIFO, However FS signal generation is conditioned by the presence of data in the FIFO.

Note: In master Rx mode: enabling the audio block immediately generates the bit clock and FS signal for the external slaves.

Note: It is mandatory to respect the following conditions in order to avoid bad SAI behavior:

- *First bit Offset <= (SLOT size - Data size)*
- *Data size <= SLOT size*
- *Number of SLOT x SLOT size = Frame length*
- *The number of slots should be even when SAI_FS_CHANNEL_IDENTIFICATION is selected.*

Note: PDM interface can be activated through HAL_SAI_Init function. Please note that PDM interface is only available for SAI1 sub-block A. PDM microphone delays can be tuned with HAL_SAIEx_ConfigPdmMicDelay function.

Three operation modes are available within this driver :

Polling mode IO operation

- Send an amount of data in blocking mode using HAL_SAI_Transmit()
- Receive an amount of data in blocking mode using HAL_SAI_Receive()

Interrupt mode IO operation

- Send an amount of data in non-blocking mode using HAL_SAI_Transmit_IT()
- At transmission end of transfer HAL_SAI_TxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SAI_TxCpltCallback()
- Receive an amount of data in non-blocking mode using HAL_SAI_Receive_IT()
- At reception end of transfer HAL_SAI_RxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SAI_RxCpltCallback()
- In case of flag error, HAL_SAI_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_SAI_ErrorCallback()

DMA mode IO operation

- Send an amount of data in non-blocking mode (DMA) using HAL_SAI_Transmit_DMA()
- At transmission end of transfer HAL_SAI_TxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SAI_TxCpltCallback()
- Receive an amount of data in non-blocking mode (DMA) using HAL_SAI_Receive_DMA()
- At reception end of transfer HAL_SAI_RxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SAI_RxCpltCallback()
- In case of flag error, HAL_SAI_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_SAI_ErrorCallback()
- Pause the DMA Transfer using HAL_SAI_DMAPause()
- Resume the DMA Transfer using HAL_SAI_DMAResume()
- Stop the DMA Transfer using HAL_SAI_DMAStop()

SAI HAL driver additional function list

Below the list the others API available SAI HAL driver :

- HAL_SAI_EnableTxMuteMode(): Enable the mute in tx mode
- HAL_SAI_DisableTxMuteMode(): Disable the mute in tx mode
- HAL_SAI_EnableRxMuteMode(): Enable the mute in Rx mode
- HAL_SAI_DisableRxMuteMode(): Disable the mute in Rx mode
- HAL_SAI_FlushRxFifo(): Flush the rx fifo.
- HAL_SAI_Abort(): Abort the current transfer

SAI HAL driver macros list

Below the list of most used macros in SAI HAL driver :

- __HAL_SAI_ENABLE(): Enable the SAI peripheral
- __HAL_SAI_DISABLE(): Disable the SAI peripheral
- __HAL_SAI_ENABLE_IT(): Enable the specified SAI interrupts
- __HAL_SAI_DISABLE_IT(): Disable the specified SAI interrupts
- __HAL_SAI_GET_IT_SOURCE(): Check if the specified SAI interrupt source is enabled or disabled
- __HAL_SAI_GET_FLAG(): Check whether the specified SAI flag is set or not

Callback registration

The compilation define USE_HAL_SAI_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use functions HAL_SAI_RegisterCallback() to register a user callback.

Function HAL_SAI_RegisterCallback() allows to register following callbacks:

- RxCpltCallback : SAI receive complete.
- RxHalfCpltCallback : SAI receive half complete.
- TxCpltCallback : SAI transmit complete.
- TxHalfCpltCallback : SAI transmit half complete.
- ErrorCallback : SAI error.
- MspInitCallback : SAI MspInit.
- MspDeInitCallback : SAI MspDeInit.

This function takes as parameters the HAL peripheral handle, the callback ID and a pointer to the user callback function.

Use function HAL_SAI_UnRegisterCallback() to reset a callback to the default weak function. HAL_SAI_UnRegisterCallback() takes as parameters the HAL peripheral handle, and the callback ID.

This function allows to reset following callbacks:

- RxCpltCallback : SAI receive complete.
- RxHalfCpltCallback : SAI receive half complete.
- TxCpltCallback : SAI transmit complete.
- TxHalfCpltCallback : SAI transmit half complete.
- ErrorCallback : SAI error.
- MspInitCallback : SAI MspInit.
- MspDeInitCallback : SAI MspDeInit.

By default, after the HAL_SAI_Init and if the state is HAL_SAI_STATE_RESET all callbacks are reset to the corresponding legacy weak functions: examples HAL_SAI_RxCpltCallback(), HAL_SAI_ErrorCallback(). Exception done for MspInit and MspDeInit callbacks that are respectively reset to the legacy weak functions in the HAL_SAI_Init and HAL_SAI_DeInit only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the HAL_SAI_Init and HAL_SAI_DeInit keep and use the user MspInit/MspDeInit callbacks (registered beforehand).

Callbacks can be registered/unregistered in READY state only. Exception done for MspInit/MspDeInit callbacks that can be registered/unregistered in READY or RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_SAI_RegisterCallback before calling HAL_SAI_DeInit or HAL_SAI_Init function.

When the compilation define USE_HAL_SAI_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak callbacks are used.

63.2.2 Detailed description of functions

HAL_SAI_InitProtocol

Function name

HAL_StatusTypeDef HAL_SAI_InitProtocol (SAI_HandleTypeDef * hsai, uint32_t protocol, uint32_t datasize, uint32_t nbslot)

Function description

HAL_SAI_Init

Function name

HAL_StatusTypeDef HAL_SAI_Init (SAI_HandleTypeDef * hsai)

Function description

HAL_SAI_DeInit

Function name

HAL_StatusTypeDef HAL_SAI_DeInit (SAI_HandleTypeDef * hsai)

Function description

HAL_SAI_MspInit

Function name

void HAL_SAI_MspInit (SAI_HandleTypeDef * hsai)

Function description

HAL_SAI_MspDeInit

Function name

void HAL_SAI_MspDeInit (SAI_HandleTypeDef * hsai)

Function description

HAL_SAI_RegisterCallback

Function name

HAL_StatusTypeDef HAL_SAI_RegisterCallback (SAI_HandleTypeDef * hsai, HAL_SAI_CallbackIDTypeDef CallbackID, pSAI_CallbackTypeDef pCallback)

Function description

HAL_SAI_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_SAI_UnRegisterCallback (SAI_HandleTypeDef * hsai, HAL_SAI_CallbackIDTypeDef CallbackID)

Function description

HAL_SAI_Transmit

Function name

HAL_StatusTypeDef HAL_SAI_Transmit (SAI_HandleTypeDef * hsai, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

HAL_SAI_Receive

Function name

HAL_StatusTypeDef HAL_SAI_Receive (SAI_HandleTypeDef * hsai, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

HAL_SAI_Transmit_IT

Function name

HAL_StatusTypeDef HAL_SAI_Transmit_IT (SAI_HandleTypeDef * hsai, uint8_t * pData, uint16_t Size)

Function description

HAL_SAI_Receive_IT

Function name

HAL_StatusTypeDef HAL_SAI_Receive_IT (SAI_HandleTypeDef * hsai, uint8_t * pData, uint16_t Size)

Function description

HAL_SAI_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_SAI_Transmit_DMA (SAI_HandleTypeDef * hsai, uint8_t * pData, uint16_t Size)

Function description

HAL_SAI_Receive_DMA

Function name

HAL_StatusTypeDef HAL_SAI_Receive_DMA (SAI_HandleTypeDef * hsai, uint8_t * pData, uint16_t Size)

Function description

HAL_SAI_DMAPause

Function name

HAL_StatusTypeDef HAL_SAI_DMAPause (SAI_HandleTypeDef * hsai)

Function description

HAL_SAI_DMAResume

Function name

HAL_StatusTypeDef HAL_SAI_DMAResume (SAI_HandleTypeDef * hsai)

Function description

HAL_SAI_DMAStop

Function name

HAL_StatusTypeDef HAL_SAI_DMAStop (SAI_HandleTypeDef * hsai)

Function description

HAL_SAI_Abort

Function name

HAL_StatusTypeDef HAL_SAI_Abort (SAI_HandleTypeDef * hsai)

Function description

HAL_SAI_EnableTxMuteMode

Function name

HAL_StatusTypeDef HAL_SAI_EnableTxMuteMode (SAI_HandleTypeDef * hsai, uint16_t val)

Function description

HAL_SAI_DisableTxMuteMode

Function name

HAL_StatusTypeDef HAL_SAI_DisableTxMuteMode (SAI_HandleTypeDef * hsai)

Function description

HAL_SAI_EnableRxMuteMode

Function name

HAL_StatusTypeDef HAL_SAI_EnableRxMuteMode (SAI_HandleTypeDef * hsai, SAIcallback callback, uint16_t counter)

Function description

HAL_SAI_DisableRxMuteMode

Function name

HAL_StatusTypeDef HAL_SAI_DisableRxMuteMode (SAI_HandleTypeDef * hsai)

Function description

HAL_SAI_IRQHandler

Function name

void HAL_SAI_IRQHandler (SAI_HandleTypeDef * hsai)

Function description

HAL_SAI_TxHalfCpltCallback

Function name

void HAL_SAI_TxHalfCpltCallback (SAI_HandleTypeDef * hsai)

Function description

HAL_SAI_TxCpltCallback

Function name

void HAL_SAI_TxCpltCallback (SAI_HandleTypeDef * hsai)

Function description

HAL_SAI_RxHalfCpltCallback

Function name

void HAL_SAI_RxHalfCpltCallback (SAI_HandleTypeDef * hsai)

Function description

HAL_SAI_RxCpltCallback

Function name

void HAL_SAI_RxCpltCallback (SAI_HandleTypeDef * hsai)

Function description

HAL_SAI_ErrorCallback

Function name

void HAL_SAI_ErrorCallback (SAI_HandleTypeDef * hsai)

Function description

HAL_SAI_GetState

Function name

HAL_SAI_StateTypeDef HAL_SAI_GetState (const SAI_HandleTypeDef * hsai)

Function description

HAL_SAI_GetError

Function name

uint32_t HAL_SAI_GetError (const SAI_HandleTypeDef * hsai)

Function description

63.3 SAI Firmware driver defines

The following section lists the various define and macros of the module.

63.3.1 SAI

SAI

SAI Audio Frequency

SAI_AUDIO_FREQUENCY_192K

SAI_AUDIO_FREQUENCY_96K

SAI_AUDIO_FREQUENCY_48K

SAI_AUDIO_FREQUENCY_44K

SAI_AUDIO_FREQUENCY_32K

SAI_AUDIO_FREQUENCY_22K

SAI_AUDIO_FREQUENCY_16K

SAI_AUDIO_FREQUENCY_11K

SAI_AUDIO_FREQUENCY_8K

SAI_AUDIO_FREQUENCY_MCKDIV

SAI Block Clock Strobing

SAI_CLOCKSTROBING_FALLINGEDGE

SAI_CLOCKSTROBING_RISINGEDGE

SAI Block Companding Mode

SAI_NOCOMPANDING

SAI_ULAW_1CPL_COMPANDING

SAI_ALAW_1CPL_COMPANDING

SAI_ULAW_2CPL_COMPANDING

SAI_ALAW_2CPL_COMPANDING

SAI Block Data Size

SAI_DATASIZE_8

SAI_DATASIZE_10

SAI_DATASIZE_16

SAI_DATASIZE_20

SAI_DATASIZE_24

SAI_DATASIZE_32

SAI Block Fifo Status Level

SAI_FIFOSTATUS_EMPTY

SAI_FIFOSTATUS_LESS1QUARTERFULL

SAI_FIFOSTATUS_1QUARTERFULL

SAI_FIFOSTATUS_HALFFULL

SAI_FIFOSTATUS_3QUARTERFULL

SAI_FIFOSTATUS_FULL

SAI Block Fifo Threshold

SAI_FIFOTHRESHOLD_EMPTY

SAI_FIFOTHRESHOLD_1QF

SAI_FIFOTHRESHOLD_HF

SAI_FIFOTHRESHOLD_3QF

SAI_FIFOTHRESHOLD_FULL

SAI Block Flags Definition

SAI_FLAG_OVRUDR

SAI_FLAG_MUTEDET

SAI_FLAG_WCKCFG

SAI_FLAG_FREQ

SAI_FLAG_CNRDY

SAI_FLAG_AFSDET

SAI_FLAG_LFSDET

SAI Block FS Definition

SAI_FS_STARTFRAME

SAI_FS_CHANNEL_IDENTIFICATION

SAI Block FS Offset

SAI_FS_FIRSTBIT

SAI_FS_BEFOREFIRSTBIT

SAI Block FS Polarity

SAI_FS_ACTIVE_LOW

SAI_FS_ACTIVE_HIGH

SAI Block Interrupts Definition

SAI_IT_OVRUDR

SAI_IT_MUTEDET

SAI_IT_WCKCFG

SAI_IT_FREQ

SAI_IT_CNRDY

SAI_IT_AFSDET

SAI_IT_LFSDET

SAI Block Master Clock Output

SAI_MCK_OUTPUT_DISABLE

SAI_MCK_OUTPUT_ENABLE

SAI Block Master Clock OverSampling

SAI_MCK_OVERSAMPLING_DISABLE

SAI_MCK_OVERSAMPLING_ENABLE

SAI Block Mode

SAI_MODEMASTER_TX

SAI_MODEMASTER_RX

SAI_MODESLAVE_TX

SAI_MODESLAVE_RX

SAI Block MSB LSB transmission

SAI_FIRSTBIT_MSB

SAI_FIRSTBIT_LSB

SAI Block Mute Value

SAI_ZERO_VALUE

SAI_LAST_SENT_VALUE

SAI Block NoDivider

SAI_MASTERDIVIDER_ENABLE

SAI_MASTERDIVIDER_DISABLE

SAI Block Output Drive

SAI_OUTPUTDRIVE_DISABLE

SAI_OUTPUTDRIVE_ENABLE

SAI Block Protocol

SAI_FREE_PROTOCOL

SAI_SPDIF_PROTOCOL

SAI_AC97_PROTOCOL

SAI Block Slot Active

SAI_SLOT_NOTACTIVE

SAI_SLOTACTIVE_0

SAI_SLOTACTIVE_1

SAI_SLOTACTIVE_2

SAI_SLOTACTIVE_3

SAI_SLOTACTIVE_4

SAI_SLOTACTIVE_5

SAI_SLOTACTIVE_6

SAI_SLOTACTIVE_7

SAI_SLOTACTIVE_8

SAI_SLOTACTIVE_9

SAI_SLOTACTIVE_10

SAI_SLOTACTIVE_11

SAI_SLOTACTIVE_12

SAI_SLOTACTIVE_13

SAI_SLOTACTIVE_14

SAI_SLOTACTIVE_15

SAI_SLOTACTIVE_ALL

SAI Block Slot Size

SAI_SLOTSIZE_DATASIZE

SAI_SLOTSIZE_16B

SAI_SLOTSIZE_32B

SAI External synchronisation

SAI_SYNCEXT_DISABLE

SAI_SYNCEXT_OUTBLOCKA_ENABLE

SAI_SYNCEXT_OUTBLOCKB_ENABLE

SAI Block Synchronization

SAI_ASYNCHRONOUS

Asynchronous

SAI_SYNCHRONOUS

Synchronous with other block of same SAI

SAI_SYNCHRONOUS_EXT_SAI1

Synchronous with other SAI, SAI1

SAI_SYNCHRONOUS_EXT_SAI2

Synchronous with other SAI, SAI2

SAI Error Code

HAL_SAI_ERROR_NONE

No error

HAL_SAI_ERROR_OVR

Overrun Error

HAL_SAI_ERROR_UDR

Underrun error

HAL_SAI_ERROR_AFSDET

Anticipated Frame synchronisation detection

HAL_SAI_ERROR_LFSDET

Late Frame synchronisation detection

HAL_SAI_ERROR_CNREADY

codec not ready

HAL_SAI_ERROR_WCKCFG

Wrong clock configuration

HAL_SAI_ERROR_TIMEOUT

Timeout error

HAL_SAI_ERROR_DMA

DMA error

HAL_SAI_ERROR_INVALID_CALLBACK

Invalid callback error

SAI Exported Macros

__HAL_SAI_RESET_HANDLE_STATE

Description:

- Reset SAI handle state.

Parameters:

- __HANDLE__: specifies the SAI Handle.

Return value:

- None

__HAL_SAI_ENABLE_IT

Description:

- Enable the specified SAI interrupts.

Parameters:

- __HANDLE__: specifies the SAI Handle.
- __INTERRUPT__: specifies the interrupt source to enable or disable. This parameter can be one of the following values:
  - SAI_IT_OVRUDR: Overrun underrun interrupt enable
  - SAI_IT_MUTEDET: Mute detection interrupt enable
  - SAI_IT_WCKCFG: Wrong Clock Configuration interrupt enable
  - SAI_IT_FREQ: FIFO request interrupt enable
  - SAI_IT_CNRDY: Codec not ready interrupt enable
  - SAI_IT_AFSDET: Anticipated frame synchronization detection interrupt enable
  - SAI_IT_LFSDET: Late frame synchronization detection interrupt enable

Return value:

- None

__HAL_SAI_DISABLE_IT

Description:

- Disable the specified SAI interrupts.

Parameters:

- __HANDLE__: specifies the SAI Handle.
- __INTERRUPT__: specifies the interrupt source to enable or disable. This parameter can be one of the following values:
  - SAI_IT_OVRUDR: Overrun underrun interrupt enable
  - SAI_IT_MUTEDET: Mute detection interrupt enable
  - SAI_IT_WCKCFG: Wrong Clock Configuration interrupt enable
  - SAI_IT_FREQ: FIFO request interrupt enable
  - SAI_IT_CNRDY: Codec not ready interrupt enable
  - SAI_IT_AFSDET: Anticipated frame synchronization detection interrupt enable
  - SAI_IT_LFSDET: Late frame synchronization detection interrupt enable

Return value:

- None

__HAL_SAI_GET_IT_SOURCE

Description:

- Check whether the specified SAI interrupt source is enabled or not.

Parameters:

- __HANDLE__: specifies the SAI Handle.
- __INTERRUPT__: specifies the SAI interrupt source to check. This parameter can be one of the following values:
  - SAI_IT_OVRUDR: Overrun underrun interrupt enable
  - SAI_IT_MUTEDET: Mute detection interrupt enable
  - SAI_IT_WCKCFG: Wrong Clock Configuration interrupt enable
  - SAI_IT_FREQ: FIFO request interrupt enable
  - SAI_IT_CNRDY: Codec not ready interrupt enable
  - SAI_IT_AFSDET: Anticipated frame synchronization detection interrupt enable
  - SAI_IT_LFSDET: Late frame synchronization detection interrupt enable

Return value:

- The: new state of __INTERRUPT__ (TRUE or FALSE).

__HAL_SAI_GET_FLAG

Description:

- Check whether the specified SAI flag is set or not.

Parameters:

- __HANDLE__: specifies the SAI Handle.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - SAI_FLAG_OVRUDR: Overrun underrun flag.
  - SAI_FLAG_MUTEDET: Mute detection flag.
  - SAI_FLAG_WCKCFG: Wrong Clock Configuration flag.
  - SAI_FLAG_FREQ: FIFO request flag.
  - SAI_FLAG_CNRDY: Codec not ready flag.
  - SAI_FLAG_AFSDET: Anticipated frame synchronization detection flag.
  - SAI_FLAG_LFSDET: Late frame synchronization detection flag.

Return value:

- The: new state of __FLAG__ (TRUE or FALSE).

__HAL_SAI_CLEAR_FLAG

Description:

- Clear the specified SAI pending flag.

Parameters:

- __HANDLE__: specifies the SAI Handle.
- __FLAG__: specifies the flag to check. This parameter can be any combination of the following values:
  - SAI_FLAG_OVRUDR: Clear Overrun underrun
  - SAI_FLAG_MUTEDET: Clear Mute detection
  - SAI_FLAG_WCKCFG: Clear Wrong Clock Configuration
  - SAI_FLAG_FREQ: Clear FIFO request
  - SAI_FLAG_CNRDY: Clear Codec not ready
  - SAI_FLAG_AFSDET: Clear Anticipated frame synchronization detection
  - SAI_FLAG_LFSDET: Clear Late frame synchronization detection

Return value:

- None

__HAL_SAI_ENABLE

Description:

- Enable SAI.

Parameters:

- __HANDLE__: specifies the SAI Handle.

Return value:

- None

__HAL_SAI_DISABLE

Description:

- Disable SAI.

Parameters:

- __HANDLE__: specifies the SAI Handle.

Return value:

- None

SAI Mono Stereo Mode

SAI_STEREOMODE

SAI_MONOMODE

SAI PDM Clock Enable

SAI_PDM_CLOCK1_ENABLE

SAI_PDM_CLOCK2_ENABLE

SAI Supported protocol

SAI_I2S_STANDARD

SAI_I2S_MSBJUSTIFIED

SAI_I2S_LSBJUSTIFIED

SAI_PCM_LONG

SAI_PCM_SHORT

SAI protocol data size

SAI_PROTOCOL_DATASIZE_16BIT

SAI_PROTOCOL_DATASIZE_16BITEXTENDED

SAI_PROTOCOL_DATASIZE_24BIT

SAI_PROTOCOL_DATASIZE_32BIT

SAI TRIState Management

SAI_OUTPUT_NOTRELEASED

SAI_OUTPUT_RELEASED

64 HAL SAI Extension Driver

64.1 SAIEx Firmware driver registers structures

64.1.1 SAIEx_PdmMicDelayParamTypeDef

SAIEx_PdmMicDelayParamTypeDef is defined in the stm32n6xx_hal_sai_ex.h

Data Fields

- ***uint32_t MicPair***
- ***uint32_t LeftDelay***
- ***uint32_t RightDelay***

Field Documentation

- ***uint32_t SAIEx_PdmMicDelayParamTypeDef::MicPair***
  Specifies which pair of microphones is selected. This parameter must be a number between Min_Data = 1 and Max_Data = 3.
- ***uint32_t SAIEx_PdmMicDelayParamTypeDef::LeftDelay***
  Specifies the delay in PDM clock unit to apply on left microphone. This parameter must be a number between Min_Data = 0 and Max_Data = 7.
- ***uint32_t SAIEx_PdmMicDelayParamTypeDef::RightDelay***
  Specifies the delay in PDM clock unit to apply on right microphone. This parameter must be a number between Min_Data = 0 and Max_Data = 7.

64.2 SAIEx Firmware driver API description

The following section lists the various functions of the SAIEx library.

64.2.1 Detailed description of functions

HAL_SAIEx_ConfigPdmMicDelay

Function name

HAL_StatusTypeDef HAL_SAIEx_ConfigPdmMicDelay (const SAI_HandleTypeDef * hsai, const SAIEx_PdmMicDelayParamTypeDef * pdmMicDelay)

Function description

65 HAL SDIO Generic Driver

65.1 SDIO Firmware driver registers structures

65.1.1 HAL_SDIO_CCCR_TypeDef

HAL_SDIO_CCCR_TypeDef is defined in the stm32n6xx_hal_sdio.h

Data Fields

- ***uint8_t sdio_revision***
- ***uint8_t cccr_revision***
- ***uint8_t sd_spec_revision***
- ***uint8_t bus_width_8Bit***
- ***uint32_t card_capability***
- ***uint32_t commonCISPointer***

Field Documentation

- ***uint8_t HAL_SDIO_CCCR_TypeDef::sdio_revision***
  SDIO revision
- ***uint8_t HAL_SDIO_CCCR_TypeDef::cccr_revision***
  CCCR version
- ***uint8_t HAL_SDIO_CCCR_TypeDef::sd_spec_revision***
  SD revision
- ***uint8_t HAL_SDIO_CCCR_TypeDef::bus_width_8Bit***
  SDIO bus width 8 bit support
- ***uint32_t HAL_SDIO_CCCR_TypeDef::card_capability***
  SDIO card capability
- ***uint32_t HAL_SDIO_CCCR_TypeDef::commonCISPointer***
  point to common CIS

65.1.2 HAL_SDIO_FBR_t

HAL_SDIO_FBR_t is defined in the stm32n6xx_hal_sdio.h

Data Fields

- ***uint8_t flags***
- ***uint8_t ioStdFunctionCode***
- ***uint8_t ioExtFunctionCode***
- ***uint32_t ioPointerToCIS***
- ***uint32_t ioPointerToCSA***

Field Documentation

- ***uint8_t HAL_SDIO_FBR_t::flags***
  SDIO current IO flags
- ***uint8_t HAL_SDIO_FBR_t::ioStdFunctionCode***
  SDIO current IO standard function code
- ***uint8_t HAL_SDIO_FBR_t::ioExtFunctionCode***
  SDIO current IO extended function code
- ***uint32_t HAL_SDIO_FBR_t::ioPointerToCIS***
  SDIO current IO pointer to CIS
- ***uint32_t HAL_SDIO_FBR_t::ioPointerToCSA***
  SDIO current IO pointer to CSA

65.1.3 HAL_SDIO_DirectCmd_TypeDef

HAL_SDIO_DirectCmd_TypeDef is defined in the stm32n6xx_hal_sdio.h

Data Fields

- ***uint32_t Reg_Addr***
- ***uint8_t ReadAfterWrite***
- ***uint8_t IOFunctionNbr***

Field Documentation

- ***uint32_t HAL_SDIO_DirectCmd_TypeDef::Reg_Addr***
  This is the address of the byte of data inside of the selected function to read or write
- ***uint8_t HAL_SDIO_DirectCmd_TypeDef::ReadAfterWrite***
  This is the read after write flag, it is used for write access only.
- ***uint8_t HAL_SDIO_DirectCmd_TypeDef::IOFunctionNbr***
  The number of the function within the IO card you wish to read or write

65.1.4 HAL_SDIO_ExtendedCmd_TypeDef

HAL_SDIO_ExtendedCmd_TypeDef is defined in the stm32n6xx_hal_sdio.h

Data Fields

- ***uint32_t Reg_Addr***
- ***uint32_t OpCode***
- ***uint32_t Block_Mode***
- ***uint32_t IOFunctionNbr***

Field Documentation

- ***uint32_t HAL_SDIO_ExtendedCmd_TypeDef::Reg_Addr***
  This is the address of the byte of data inside of the selected function to read or write
- ***uint32_t HAL_SDIO_ExtendedCmd_TypeDef::OpCode***
  Read/Write operation mode
- ***uint32_t HAL_SDIO_ExtendedCmd_TypeDef::Block_Mode***
  Bytes or Blocks mode
- ***uint32_t HAL_SDIO_ExtendedCmd_TypeDef::IOFunctionNbr***
  The number of the function within the IO card you wish to read or write

65.1.5 __SDIO_HandleTypeDef

__SDIO_HandleTypeDef is defined in the stm32n6xx_hal_sdio.h

Data Fields

- ***SDIO_TypeDef * Instance***
- ***SDIO_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***uint8_t * pTxBuffPtr***
- ***uint32_t TxXferSize***
- ***uint8_t * pRxBuffPtr***
- ***uint32_t RxXferSize***
- ***uint32_t remaining_data***
- ***uint32_t next_data_addr***
- ***__IO uint32_t next_reg_addr***
- ***uint16_t block_size***
- ***__IO uint32_t Context***
- ***__IO HAL_SDIO_StateTypeDef State***
- ***__IO uint32_t ErrorCode***
- ***uint8_t IOFunctionMask***
- ***uint8_t IOInterruptNbr***
- ***void(* SDIO_IOFunction_Callback***
- ***void(* TxCpltCallback***
- ***void(* RxCpltCallback***
- ***void(* ErrorCallback***

- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***
- ***void(* DriveTransceiver_1_8V_Callback***
- ***HAL_StatusTypeDef(* SDIO_IdentifyCard***

Field Documentation

- ***SDIO_TypeDef* __SDIO_HandleTypeDef::Instance***
  SDIO registers base address
- ***SDIO_InitTypeDef __SDIO_HandleTypeDef::Init***
  SDIO required parameters
- ***HAL_LockTypeDef __SDIO_HandleTypeDef::Lock***
  SDIO locking object
- ***uint8_t* __SDIO_HandleTypeDef::pTxBuffPtr***
  Pointer to SDIO Tx transfer Buffer
- ***uint32_t __SDIO_HandleTypeDef::TxXferSize***
  SDIO Tx Transfer size
- ***uint8_t* __SDIO_HandleTypeDef::pRxBuffPtr***
  Pointer to SDIO Rx transfer Buffer
- ***uint32_t __SDIO_HandleTypeDef::RxXferSize***
  SDIO Rx Transfer size
- ***uint32_t __SDIO_HandleTypeDef::remaining_data***
  Remaining data to transfer
- ***uint32_t __SDIO_HandleTypeDef::next_data_addr***
  SDIO Next data address
- ***__IO uint32_t __SDIO_HandleTypeDef::next_reg_addr***
  SDIO Next register address
- ***uint16_t __SDIO_HandleTypeDef::block_size***
  SDIO Block size
- ***__IO uint32_t __SDIO_HandleTypeDef::Context***
  SDIO transfer context
- ***__IO HAL_SDIO_StateTypeDef __SDIO_HandleTypeDef::State***
  SDIO card State
- ***__IO uint32_t __SDIO_HandleTypeDef::ErrorCode***
  SDIO Card Error codes
- ***uint8_t __SDIO_HandleTypeDef::IOFunctionMask***
  SDIO used to record current enabled io interrupt
- ***volatile uint8_t __SDIO_HandleTypeDef::IOInterruptNbr***
  SDIO used to record total enabled io interrupt numbers
- ***void(* __SDIO_HandleTypeDef::SDIO_IOFunction_Callback[SDIO_MAX_IO_NUMBER])(struct __SDIO_HandleTypeDef *hsdio, uint32_t func)***
- ***void(* __SDIO_HandleTypeDef::TxCpltCallback)(struct __SDIO_HandleTypeDef *hsdio)***
- ***void(* __SDIO_HandleTypeDef::RxCpltCallback)(struct __SDIO_HandleTypeDef *hsdio)***
- ***void(* __SDIO_HandleTypeDef::ErrorCallback)(struct __SDIO_HandleTypeDef *hsdio)***
- ***void(* __SDIO_HandleTypeDef::MspInitCallback)(struct __SDIO_HandleTypeDef *hsdio)***
- ***void(* __SDIO_HandleTypeDef::MspDeInitCallback)(struct __SDIO_HandleTypeDef *hsdio)***
- ***void(* __SDIO_HandleTypeDef::DriveTransceiver_1_8V_Callback)(struct __SDIO_HandleTypeDef *hsdio, FlagStatus status)***
- ***HAL_StatusTypeDef(* __SDIO_HandleTypeDef::SDIO_IdentifyCard)(struct __SDIO_HandleTypeDef *hsdio)***

65.2 SDIO Firmware driver API description

The following section lists the various functions of the SDIO library.

65.2.1 How to use this driver

This driver implements a high level communication layer for read and write from/to this SDIO card. The needed STM32 hardware resources (SDMMC and GPIO) are performed by the user in HAL_SDIO_MspInit() function (MSP layer). Basically, the MSP layer configuration should be the same as we provide in the examples. You can easily tailor this configuration according to hardware resources.

This driver is a generic layered driver for SDMMC memories which uses the HAL SDMMC driver functions to interface with SDIO cards devices. It is used as follows:

1. Initialize the SDMMC low level resources by implementing the HAL_SDIO_MspInit() API:
   a. Enable the SDMMC interface clock using __HAL_RCC_SDMMC_CLK_ENABLE();
   b. SDMMC pins configuration for SDIO card
      - Enable the clock for the SDMMC GPIOs using the functions __HAL_RCC_GPIOx_CLK_ENABLE();
      - Configure these SDMMC pins as alternate function pull-up using HAL_GPIO_Init() and according to your pin assignment;
   c. NVIC configuration if you need to use interrupt process (HAL_SDIO_ReadExtended_DMA() and HAL_SDIO_WriteExtended_DMA() APIs).
      - Configure the SDMMC interrupt priorities using function HAL_NVIC_SetPriority();
      - Enable the NVIC SDMMC IRQs using function HAL_NVIC_EnableIRQ()
      - SDMMC interrupts are managed using the macros __HAL_SDIO_ENABLE_IT() and __HAL_SDIO_DISABLE_IT() inside the communication process.
      - SDMMC interrupts pending bits are managed using the macros __HAL_SDIO_GET_IT().
   d. No general propose DMA Configuration is needed, an Internal DMA for SDMMC Peripheral are used.
2. At this stage, you can perform SDIO read/write/erase operations after SDIO card initialization.

SDIO Card Initialization and configuration

To initialize the SDIO Card, use the HAL_SDIO_Init() function. It Initializes SDMMC Peripheral(STM32 side) and the SDIO Card, and put it into StandBy State (Ready for data transfer). This function provide the following operations:

1. Apply the SDIO Card initialization process at 400KHz. You can change or adapt this frequency by adjusting the "ClockDiv" field. The SDIO Card frequency (SDMMC_CK) is computed as follows: SDMMC_CK = SDMMCCLK / (2 * ClockDiv) In initialization mode and according to the SDIO Card standard, make sure that the SDMMC_CK frequency doesn't exceed 400KHz. This phase of initialization is done through SDMMC_Init() and SDMMC_PowerState_ON() SDMMC low level APIs.
2. Initialize the SDIO card. The API used is HAL_SDIO_Init(). This phase allows the card initialization and identification.
3. Configure the SDIO Card Data transfer frequency. You can change or adapt this frequency by adjusting the "ClockDiv" field by the API HAL_SDIO_ConfigFrequency().
4. Configure the SDIO Card in wide bus mode: 4-bits data by the API HAL_SDIO_SetDataBusWidth().
5. Configure the SDIO Card data block size by the API : HAL_SDIO_SetBlockSize().
6. Configure the SDIO Card speed mode by the API : HAL_SDIO_SetSpeedMode().
7. To custumize the SDIO Init card function for the enumeration card sequence, you can register a user callback function by calling the HAL_SDIO_RegisterIdentifyCardCallback before the HAL_SDIO_Init() function.

SDIO Card Read operation

- You can read from SDIO card in polling mode by using function HAL_SDIO_ReadExtended(). This function support only 2048-bytes block length (the block size should be chosen by using the API HAL_SDIO_SetBlockSize).
- You can read from SDIO card in DMA mode by using function HAL_SDIO_ReadExtended_DMA(). This function support only 2048-bytes block length (the block size should be chosen by using the API HAL_SDIO_SetBlockSize). After this, you have to ensure that the transfer is done correctly. You could also check the DMA transfer process through the SDIO Rx interrupt event.

SDIO Card Write operation

- You can write to SDIO card in polling mode by using function HAL_SDIO_WriteExtended(). This function support only 2048-bytes block length (the block size should be chosen by using the API HAL_SDIO_SetBlockSize).

- You can write to SDIO card in DMA mode by using function HAL_SDIO_WriteExtended_DMA(). This function support only 2048-bytes block length (the block size should be chosen by using the API HAL_SDIO_SetBlockSize). You could also check the DMA transfer process through the SDIO Tx interrupt event.

SDIO card common control register (CCCR)

- The SDIO CCCR allow for quick host checking and control of an IO card's enable and interrupts on a per card and per function basis. To get the Card common control registers field, you can use the API HAL_SDIO_GetCardCommonControlRegister().

SDIO card Function basic register (FBR)

- The SDIO card function basic register are used to allow the host to quickly determine the abilities and requirements of each function.
- To get the SDIO function basic register information, you can use the API HAL_SDIO_GetCardFBRRegister().

SDIO HAL driver macros list

Note: You can refer to the SDIO HAL driver header file for more useful macros

Callback registration

The compilation define USE_HAL_SDIO_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use Functions HAL_SDIO_RegisterCallback() to register a user callback, it allows to register following callbacks:

- TxCpltCallback : callback when a transmission transfer is completed.
- RxCpltCallback : callback when a reception transfer is completed.
- ErrorCallback : callback when error occurs.
- MspInitCallback : SDIO MspInit.
- MspDeInitCallback : SDIO MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function. For specific callbacks TransceiverCallback use dedicated register callbacks: respectively HAL_SDIO_RegisterTransceiverCallback(). Use function HAL_SDIO_UnRegisterCallback() to reset a callback to the default weak (overridden) function. It allows to reset following callbacks:
- TxCpltCallback : callback when a transmission transfer is completed.
- RxCpltCallback : callback when a reception transfer is completed.
- ErrorCallback : callback when error occurs.
- MspInitCallback : SDIO MspInit.
- MspDeInitCallback : SDIO MspDeInit. This function) takes as parameters the HAL peripheral handle and the Callback ID. For specific callbacks TransceiverCallback use dedicated unregister callbacks: respectively HAL_SDIO_UnRegisterTransceiverCallback(). By default, after the HAL_SDIO_Init and if the state is HAL_SDIO_STATE_RESET all callbacks are reset to the corresponding legacy weak (overridden) functions. Exception done for MspInit and MspDeInit callbacks that are respectively reset to the legacy weak (overridden) functions in the HAL_SDIO_Init and HAL_SDIO_DeInit only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the HAL_SDIO_Init and HAL_SDIO_DeInit keep and use the user MspInit/MspDeInit callbacks (registered beforehand) Callbacks can be registered/unregistered in READY state only. Exception done for MspInit/MspDeInit callbacks that can be registered/unregistered in READY or RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_SDIO_RegisterCallback before calling HAL_SDIO_DeInit or HAL_SDIO_Init function. When The compilation define USE_HAL_SDIO_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak (overridden) callbacks are used.

SDIO peripheral IO interrupt

- Below the list of most used SDIO function to check and control the IO card's enable and interrupts on a per functions basis.
- HAL_SDIO_EnableIOFunctionInterrupt: Enable SDIO IO interrupt.

- HAL_SDIO_DisableIOFunctionInterrupt: Disable SDIO IO interrupt.
- HAL_SDIO_EnableIOFunction: Enable Function number(0-7)
- HAL_SDIO_DisableIOFunction: Disable Function number(0-7)
- HAL_SDIO_SelectIOFunction: Select a function number(0-7)
- HAL_SDIO_AbortIOFunction: Abort an IO read or write operation and free the SDIO bus.
- HAL_SDIO_EnableIOAsynInterrupt: Enable Bit of asynchronous interrupt
- HAL_SDIO_DisableIOAsynInterrupt: Disable Bit of asynchronous interrupt

65.2.2 Detailed description of functions

HAL_SDIO_Init

Function name

HAL_StatusTypeDef HAL_SDIO_Init (SDIO_HandleTypeDef * hsdio)

Function description

HAL_SDIO_DeInit

Function name

HAL_StatusTypeDef HAL_SDIO_DeInit (SDIO_HandleTypeDef * hsdio)

Function description

HAL_SDIO_MspInit

Function name

void HAL_SDIO_MspInit (SDIO_HandleTypeDef * hsdio)

Function description

HAL_SDIO_MspDeInit

Function name

void HAL_SDIO_MspDeInit (SDIO_HandleTypeDef * hsdio)

Function description

HAL_SDIO_SetDataBusWidth

Function name

HAL_StatusTypeDef HAL_SDIO_SetDataBusWidth (SDIO_HandleTypeDef * hsdio, uint32_t BusWide)

Function description

HAL_SDIO_ConfigFrequency

Function name

HAL_StatusTypeDef HAL_SDIO_ConfigFrequency (SDIO_HandleTypeDef * hsdio, uint32_t ClockSpeed)

Function description

HAL_SDIO_SetBlockSize

Function name

HAL_StatusTypeDef HAL_SDIO_SetBlockSize (SDIO_HandleTypeDef * hsdio, uint8_t function_nbr, uint16_t BlockSize)

Function description

HAL_SDIO_SetSpeedMode

Function name

HAL_StatusTypeDef HAL_SDIO_SetSpeedMode (SDIO_HandleTypeDef * hsdio, uint32_t DataRate)

Function description

HAL_SDIO_CardReset

Function name

HAL_StatusTypeDef HAL_SDIO_CardReset (SDIO_HandleTypeDef * hsdio)

Function description

HAL_SDIO_GetCardCommonControlRegister

Function name

HAL_StatusTypeDef HAL_SDIO_GetCardCommonControlRegister (SDIO_HandleTypeDef * hsdio, HAL_SDIO_CCCR_TypeDef * pCccr)

Function description

HAL_SDIO_GetCardFBRRegister

Function name

HAL_StatusTypeDef HAL_SDIO_GetCardFBRRegister (SDIO_HandleTypeDef * hsdio, HAL_SDIO_FBR_t * pFbr)

Function description

HAL_SDIO_ReadDirect

Function name

HAL_StatusTypeDef HAL_SDIO_ReadDirect (SDIO_HandleTypeDef * hsdio, const HAL_SDIO_DirectCmd_TypeDef * Argument, uint8_t * pData)

Function description

HAL_SDIO_WriteDirect

Function name

HAL_StatusTypeDef HAL_SDIO_WriteDirect (SDIO_HandleTypeDef * hsdio, const HAL_SDIO_DirectCmd_TypeDef * Argument, uint8_t Data)

Function description

HAL_SDIO_ReadExtended

Function name

HAL_StatusTypeDef HAL_SDIO_ReadExtended (SDIO_HandleTypeDef * hsdio, const HAL_SDIO_ExtendedCmd_TypeDef * Argument, uint8_t * pData, uint32_t Size_byte, uint32_t Timeout_Ms)

Function description

HAL_SDIO_WriteExtended

Function name

HAL_StatusTypeDef HAL_SDIO_WriteExtended (SDIO_HandleTypeDef * hsdio, const HAL_SDIO_ExtendedCmd_TypeDef * Argument, uint8_t * pData, uint32_t Size_byte, uint32_t Timeout_Ms)

Function description

HAL_SDIO_ReadExtended_DMA

Function name

HAL_StatusTypeDef HAL_SDIO_ReadExtended_DMA (SDIO_HandleTypeDef * hsdio, const HAL_SDIO_ExtendedCmd_TypeDef * Argument, uint8_t * pData, uint32_t Size_byte)

Function description

HAL_SDIO_WriteExtended_DMA

Function name

HAL_StatusTypeDef HAL_SDIO_WriteExtended_DMA (SDIO_HandleTypeDef * hsdio, const HAL_SDIO_ExtendedCmd_TypeDef * Argument, uint8_t * pData, uint32_t Size_byte)

Function description

HAL_SDIO_IRQHandler

Function name

void HAL_SDIO_IRQHandler (SDIO_HandleTypeDef * hsdio)

Function description

HAL_SDIO_TxCpltCallback

Function name

void HAL_SDIO_TxCpltCallback (SDIO_HandleTypeDef * hsdio)

Function description

HAL_SDIO_RxCpltCallback

Function name

void HAL_SDIO_RxCpltCallback (SDIO_HandleTypeDef * hsdio)

Function description

HAL_SDIO_ErrorCallback

Function name

void HAL_SDIO_ErrorCallback (SDIO_HandleTypeDef * hsdio)

Function description

HAL_SDIO_IOFunctionCallback

Function name

void HAL_SDIO_IOFunctionCallback (SDIO_HandleTypeDef * hsdio, uint32_t func)

Function description

HAL_SDIO_DriveTransceiver_1_8V_Callback

Function name

void HAL_SDIO_DriveTransceiver_1_8V_Callback (SDIO_HandleTypeDef * hsdio, FlagStatus status)

Function description

HAL_SDIO_RegisterCallback

Function name

HAL_StatusTypeDef HAL_SDIO_RegisterCallback (SDIO_HandleTypeDef * hsdio, HAL_SDIO_CallbackIDTypeDef CallbackID, pSDIO_CallbackTypeDef pCallback)

Function description

HAL_SDIO_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_SDIO_UnRegisterCallback (SDIO_HandleTypeDef * hsdio, HAL_SDIO_CallbackIDTypeDef CallbackID)

Function description

HAL_SDIO_RegisterTransceiverCallback

Function name

HAL_StatusTypeDef HAL_SDIO_RegisterTransceiverCallback (SDIO_HandleTypeDef * hsdio, pSDIO_TransceiverCallbackTypeDef pCallback)

Function description

HAL_SDIO_UnRegisterTransceiverCallback

Function name

HAL_StatusTypeDef HAL_SDIO_UnRegisterTransceiverCallback (SDIO_HandleTypeDef * hsdio)

Function description

HAL_SDIO_RegisterIOFunctionCallback

Function name

HAL_StatusTypeDef HAL_SDIO_RegisterIOFunctionCallback (SDIO_HandleTypeDef * hsdio, uint32_t IOFunction, HAL_SDIO_IOFunction_CallbackTypeDef pCallback)

Function description

HAL_SDIO_RegisterIdentifyCardCallback

Function name

HAL_StatusTypeDef HAL_SDIO_RegisterIdentifyCardCallback (SDIO_HandleTypeDef * hsdio, pSDIO_IdentifyCardCallbackTypeDef pCallback)

Function description

HAL_SDIO_GetState

Function name

HAL_SDIO_StateTypeDef HAL_SDIO_GetState (const SDIO_HandleTypeDef * hsdio)

Function description

HAL_SDIO_GetError

Function name

uint32_t HAL_SDIO_GetError (const SDIO_HandleTypeDef * hsdio)

Function description

HAL_SDIO_EnableIOFunctionInterrupt

Function name

HAL_StatusTypeDef HAL_SDIO_EnableIOFunctionInterrupt (SDIO_HandleTypeDef * hsdio, uint32_t IOFunction)

Function description

HAL_SDIO_DisableIOFunctionInterrupt

Function name

HAL_StatusTypeDef HAL_SDIO_DisableIOFunctionInterrupt (SDIO_HandleTypeDef * hsdio, uint32_t IOFunction)

Function description

HAL_SDIO_EnableIOFunction

Function name

HAL_StatusTypeDef HAL_SDIO_EnableIOFunction (SDIO_HandleTypeDef * hsdio, uint32_t IOFunction)

Function description

HAL_SDIO_DisableIOFunction

Function name

HAL_StatusTypeDef HAL_SDIO_DisableIOFunction (SDIO_HandleTypeDef * hsdio, uint32_t IOFunction)

Function description

HAL_SDIO_SelectIOFunction

Function name

HAL_StatusTypeDef HAL_SDIO_SelectIOFunction (SDIO_HandleTypeDef * hsdio, uint32_t IOFunction)

Function description

HAL_SDIO_AbortIOFunction

Function name

HAL_StatusTypeDef HAL_SDIO_AbortIOFunction (SDIO_HandleTypeDef * hsdio, uint32_t IOFunction)

Function description

HAL_SDIO_EnableIOAsynInterrupt

Function name

HAL_StatusTypeDef HAL_SDIO_EnableIOAsynInterrupt (SDIO_HandleTypeDef * hsdio)

Function description

HAL_SDIO_DisableIOAsynInterrupt

Function name

HAL_StatusTypeDef HAL_SDIO_DisableIOAsynInterrupt (SDIO_HandleTypeDef * hsdio)

Function description

65.3 SDIO Firmware driver defines

The following section lists the various define and macros of the module.

65.3.1 SDIO

SDIO

SDIO Error status Structure definition

HAL_SDIO_ERROR_NONE

No error

HAL_SDIO_ERROR_DATA_CRC_FAIL

Data block sent/received (CRC check failed)

HAL_SDIO_ERROR_DATA_TIMEOUT

Data timeout

HAL_SDIO_ERROR_TX_UNDERRUN

Transmit FIFO underrun

HAL_SDIO_ERROR_RX_OVERRUN

Receive FIFO overrun

HAL_SDIO_ERROR_TIMEOUT

Timeout error

HAL_SDIO_ERROR_INVALID_CALLBACK

Invalid callback error

SDIO Functions definitions

HAL_SDIO_FUNCTION_0

SDIO function 0

HAL_SDIO_FUNCTION_1

SDIO function 1

HAL_SDIO_FUNCTION_2

SDIO function 2

HAL_SDIO_FUNCTION_3

SDIO function 3

HAL_SDIO_FUNCTION_4

SDIO function 4

HAL_SDIO_FUNCTION_5

SDIO function 5

HAL_SDIO_FUNCTION_6

SDIO function 6

HAL_SDIO_FUNCTION_7

SDIO function 7

SDIO FBR definitions

HAL_SDIO_FBR_SUPPORT_CSA

SDIO function support CSA

HAL_SDIO_FBR_SUPPORT_POWER_SELECTION

SDIO function support power selection

SDIO context enumeration

SDIO_CONTEXT_NONE

None

SDIO_CONTEXT_READ_SINGLE_BLOCK

Read single block operation

SDIO_CONTEXT_READ_MULTIPLE_BLOCK

Read multiple blocks operation

SDIO_CONTEXT_WRITE_SINGLE_BLOCK

Write single block operation

SDIO_CONTEXT_WRITE_MULTIPLE_BLOCK

Write multiple blocks operation

SDIO_CONTEXT_IT

Process in Interrupt mode

SDIO_CONTEXT_DMA

Process in DMA mode

SDIO Block mode enumeration

HAL_SDIO_MODE_BYTE

HAL_SDIO_MODE_BLOCK

SDIO operation code enumeration

HAL_SDIO_OP_CODE_NO_INC

HAL_SDIO_OP_CODE_AUTO_INC

SDIO Read After Write(RAW) enumeration

HAL_SDIO_WRITE_ONLY

SDIO Write only

HAL_SDIO_READ_AFTER_WRITE

SDIO Read after write

SDIO wire mode enumeration

HAL_SDIO_1_WIRE_MODE

SDIO wire support 1 wire

HAL_SDIO_4_WIRES_MODE

SDIO wire support 4 wires

SDIO Data block size enumeration

HAL_SDIO_DATA_BLOCK_SIZE_1BYTE

SDIO data block size 1 byte

HAL_SDIO_DATA_BLOCK_SIZE_2BYTE

SDIO data block size 2 byte

HAL_SDIO_DATA_BLOCK_SIZE_4BYTE

SDIO data block size 4 byte

HAL_SDIO_DATA_BLOCK_SIZE_8BYTE

SDIO data block size 8 byte

HAL_SDIO_DATA_BLOCK_SIZE_16BYTE

SDIO data block size 16 byte

HAL_SDIO_DATA_BLOCK_SIZE_32BYTE

SDIO data block size 32 byte

HAL_SDIO_DATA_BLOCK_SIZE_64BYTE

SDIO data block size 64 byte

HAL_SDIO_DATA_BLOCK_SIZE_128BYTE

SDIO data block size 128 byte

HAL_SDIO_DATA_BLOCK_SIZE_256BYTE

SDIO data block size 256 byte

HAL_SDIO_DATA_BLOCK_SIZE_512BYTE

SDIO data block size 512 byte

HAL_SDIO_DATA_BLOCK_SIZE_1024BYTE

SDIO data block size 1024 byte

HAL_SDIO_DATA_BLOCK_SIZE_2048BYTE

SDIO data block size 2048 byte

SDIO Bus Width enumeration

HAL_SDIO_BUS_WIDTH_8BIT_NOT_SUPPORTED

SDIO bus width 8 bit is not supported

HAL_SDIO_BUS_WIDTH_8BIT_SUPPORTED

SDIO bus width 8 bit is supported

SDIO Data rate definitions

HAL_SDIOS_DATA_RATE_SDR12

SDIO Data rate SDR12

HAL_SDIOS_DATA_RATE_SDR25

SDIO Data rate SDR25

HAL_SDIOS_DATA_RATE_SDR50

SDIO Data rate SDR50

HAL_SDIOS_DATA_RATE_DDR50

SDIO Data rate DDR50

SDIO Exported Macros

__HAL_SDIO_ENABLE_IT

Description:

- Enable the SDIO device interrupt.

Parameters:

- __HANDLE__: SDIO Handle.
- __INTERRUPT__: specifies the SDMMC interrupt sources to be enabled.

Return value:

- None

__HAL_SDIO_DISABLE_IT

Description:

- Disable the SDIO device interrupt.

Parameters:

- __HANDLE__: SDIO Handle.
- __INTERRUPT__: specifies the SDMMC interrupt sources to be disabled.

Return value:

- None

__HAL_SDIO_GET_FLAG

Description:

- Check whether the specified SDIO flag is set or not.

Parameters:

- __HANDLE__: SDIO Handle.
- __FLAG__: specifies the flag to check.

Return value:

- The: new state of SDIO FLAG (SET or RESET).

__HAL_SDIO_CLEAR_FLAG

Description:

- Clear the SDIO's pending flags.

Parameters:

- __HANDLE__: SDIO Handle.
- __FLAG__: specifies the flag to clear.

Return value:

- None

__HAL_SDIO_GET_IT

Description:

- Check whether the specified SDIO interrupt has occurred or not.

Parameters:

- __HANDLE__: SDIO Handle.
- __INTERRUPT__: specifies the SDMMC interrupt source to check.

Return value:

- The: new state of SDIO IT (SET or RESET).

__HAL_SDIO_GET_IT_SOURCE

Description:

- Checks whether the specified SDIO interrupt is enabled or not.

Parameters:

- __HANDLE__: : SDIO handle.
- __INTERRUPT__: : specifies the SDMMC interrupt source to check.

Return value:

- The: state of SDIO IT (SET or RESET).

SDIO Handle and Structure definition

SDIO_InitTypeDef

SDIO_TypeDef

66 HAL SDRAM Generic Driver

66.1 SDRAM Firmware driver registers structures

66.1.1 __SDRAM_HandleTypeDef

__SDRAM_HandleTypeDef is defined in the stm32n6xx_hal_sdram.h

Data Fields

- ***FMC_SDRAM_TypeDef * Instance***
- ***FMC_SDRAM_InitTypeDef Init***
- ***__IO HAL_SDRAM_StateTypeDef State***
- ***HAL_LockTypeDef Lock***
- ***DMA_HandleTypeDef * hdma***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***
- ***void(* RefreshErrorCallback***
- ***void(* DmaXferCpltCallback***
- ***void(* DmaXferErrorCallback***

Field Documentation

- ***FMC_SDRAM_TypeDef* __SDRAM_HandleTypeDef::Instance***
  Register base address
- ***FMC_SDRAM_InitTypeDef __SDRAM_HandleTypeDef::Init***
  SDRAM device configuration parameters
- ***__IO HAL_SDRAM_StateTypeDef __SDRAM_HandleTypeDef::State***
  SDRAM access state
- ***HAL_LockTypeDef __SDRAM_HandleTypeDef::Lock***
  SDRAM locking object
- ***DMA_HandleTypeDef* __SDRAM_HandleTypeDef::hdma***
  Pointer DMA handler
- ***void(* __SDRAM_HandleTypeDef::MspInitCallback)(struct __SDRAM_HandleTypeDef *hsdram)***
  SDRAM Msp Init callback
- ***void(* __SDRAM_HandleTypeDef::MspDeInitCallback)(struct __SDRAM_HandleTypeDef *hsdram)***
  SDRAM Msp DeInit callback
- ***void(* __SDRAM_HandleTypeDef::RefreshErrorCallback)(struct __SDRAM_HandleTypeDef *hsdram)***
  SDRAM Refresh Error callback
- ***void(* __SDRAM_HandleTypeDef::DmaXferCpltCallback)(DMA_HandleTypeDef *hdma)***
  SDRAM DMA Xfer Complete callback
- ***void(* __SDRAM_HandleTypeDef::DmaXferErrorCallback)(DMA_HandleTypeDef *hdma)***
  SDRAM DMA Xfer Error callback

66.2 SDRAM Firmware driver API description

The following section lists the various functions of the SDRAM library.

66.2.1 How to use this driver

This driver is a generic layered driver which contains a set of APIs used to control SDRAM memories. It uses the FMC layer functions to interface with SDRAM devices. The following sequence should be followed to configure the FMC to interface with SDRAM memories:

1. Declare a SDRAM_HandleTypeDef handle structure, for example: SDRAM_HandleTypeDef hsdram
   – Fill the SDRAM_HandleTypeDef handle "Init" field with the allowed values of the structure member.
   – Fill the SDRAM_HandleTypeDef handle "Instance" field with a predefined base register instance for NOR or SDRAM device

2. Declare a FMC_SDRAM_TimingTypeDef structure; for example: FMC_SDRAM_TimingTypeDef Timing; and fill its fields with the allowed values of the structure member.
3. Initialize the SDRAM Controller by calling the function HAL_SDRAM_Init(). This function performs the following sequence:
   a. MSP hardware layer configuration using the function HAL_SDRAM_MspInit()
   b. Control register configuration using the FMC SDRAM interface function FMC_SDRAM_Init()
   c. Timing register configuration using the FMC SDRAM interface function FMC_SDRAM_Timing_Init()
   d. Program the SDRAM external device by applying its initialization sequence according to the device plugged in your hardware. This step is mandatory for accessing the SDRAM device.
4. At this stage you can perform read/write accesses from/to the memory connected to the SDRAM Bank. You can perform either polling or DMA transfer using the following APIs:
   – HAL_SDRAM_Read()/HAL_SDRAM_Write() for polling read/write access
   – HAL_SDRAM_Read_DMA()/HAL_SDRAM_Write_DMA() for DMA read/write transfer
5. You can also control the SDRAM device by calling the control APIs HAL_SDRAM_WriteOperation_Enable()/ HAL_SDRAM_WriteOperation_Disable() to respectively enable/disable the SDRAM write operation or the function HAL_SDRAM_SendCommand() to send a specified command to the SDRAM device. The command to be sent must be configured with the FMC_SDRAM_CommandTypeDef structure.
6. You can continuously monitor the SDRAM device HAL state by calling the function HAL_SDRAM_GetState()

Callback registration

The compilation define USE_HAL_SDRAM_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use Functions HAL_SDRAM_RegisterCallback() to register a user callback, it allows to register following callbacks:

- MspInitCallback : SDRAM MspInit.
- MspDeInitCallback : SDRAM MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function. Use function HAL_SDRAM_UnRegisterCallback() to reset a callback to the default weak (overridden) function. It allows to reset following callbacks:
- MspInitCallback : SDRAM MspInit.
- MspDeInitCallback : SDRAM MspDeInit. This function) takes as parameters the HAL peripheral handle and the Callback ID. By default, after the HAL_SDRAM_Init and if the state is HAL_SDRAM_STATE_RESET all callbacks are reset to the corresponding legacy weak (overridden) functions. Exception done for MspInit and MspDeInit callbacks that are respectively reset to the legacy weak (overridden) functions in the HAL_SDRAM_Init and HAL_SDRAM_DeInit only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the HAL_SDRAM_Init and HAL_SDRAM_DeInit keep and use the user MspInit/MspDeInit callbacks (registered beforehand) Callbacks can be registered/ unregistered in READY state only. Exception done for MspInit/MspDeInit callbacks that can be registered/ unregistered in READY or RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_SDRAM_RegisterCallback before calling HAL_SDRAM_DeInit or HAL_SDRAM_Init function. When The compilation define USE_HAL_SDRAM_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak (overridden) callbacks are used.

66.2.2 Detailed description of functions

HAL_SDRAM_Init

Function name

HAL_StatusTypeDef HAL_SDRAM_Init (SDRAM_HandleTypeDef * hsdram, FMC_SDRAM_TimingTypeDef * Timing)

Function description

HAL_SDRAM_DeInit

Function name

HAL_StatusTypeDef HAL_SDRAM_DeInit (SDRAM_HandleTypeDef * hsdram)

Function description

HAL_SDRAM_MspInit

Function name

void HAL_SDRAM_MspInit (SDRAM_HandleTypeDef * hsdram)

Function description

HAL_SDRAM_MspDeInit

Function name

void HAL_SDRAM_MspDeInit (SDRAM_HandleTypeDef * hsdram)

Function description

HAL_SDRAM_IRQHandler

Function name

void HAL_SDRAM_IRQHandler (SDRAM_HandleTypeDef * hsdram)

Function description

HAL_SDRAM_RefreshErrorCallback

Function name

void HAL_SDRAM_RefreshErrorCallback (SDRAM_HandleTypeDef * hsdram)

Function description

HAL_SDRAM_DMA_XferCpltCallback

Function name

void HAL_SDRAM_DMA_XferCpltCallback (DMA_HandleTypeDef * hdma)

Function description

HAL_SDRAM_DMA_XferErrorCallback

Function name

void HAL_SDRAM_DMA_XferErrorCallback (DMA_HandleTypeDef * hdma)

Function description

HAL_SDRAM_Read_8b

Function name

HAL_StatusTypeDef HAL_SDRAM_Read_8b (SDRAM_HandleTypeDef * hsdram, uint32_t * pAddress, uint8_t * pDstBuffer, uint32_t BufferSize)

Function description

HAL_SDRAM_Write_8b

Function name

HAL_StatusTypeDef HAL_SDRAM_Write_8b (SDRAM_HandleTypeDef * hsdram, uint32_t * pAddress, uint8_t * pSrcBuffer, uint32_t BufferSize)

Function description

HAL_SDRAM_Read_16b

Function name

HAL_StatusTypeDef HAL_SDRAM_Read_16b (SDRAM_HandleTypeDef * hsdram, uint32_t * pAddress, uint16_t * pDstBuffer, uint32_t BufferSize)

Function description

HAL_SDRAM_Write_16b

Function name

HAL_StatusTypeDef HAL_SDRAM_Write_16b (SDRAM_HandleTypeDef * hsdram, uint32_t * pAddress, uint16_t * pSrcBuffer, uint32_t BufferSize)

Function description

HAL_SDRAM_Read_32b

Function name

HAL_StatusTypeDef HAL_SDRAM_Read_32b (SDRAM_HandleTypeDef * hsdram, uint32_t * pAddress, uint32_t * pDstBuffer, uint32_t BufferSize)

Function description

HAL_SDRAM_Write_32b

Function name

HAL_StatusTypeDef HAL_SDRAM_Write_32b (SDRAM_HandleTypeDef * hsdram, uint32_t * pAddress, uint32_t * pSrcBuffer, uint32_t BufferSize)

Function description

HAL_SDRAM_Read_DMA

Function name

HAL_StatusTypeDef HAL_SDRAM_Read_DMA (SDRAM_HandleTypeDef * hsdram, uint32_t * pAddress, uint32_t * pDstBuffer, uint32_t BufferSize)

Function description

HAL_SDRAM_Write_DMA

Function name

HAL_StatusTypeDef HAL_SDRAM_Write_DMA (SDRAM_HandleTypeDef * hsdram, uint32_t * pAddress, uint32_t * pSrcBuffer, uint32_t BufferSize)

Function description

HAL_SDRAM_RegisterCallback

Function name

HAL_StatusTypeDef HAL_SDRAM_RegisterCallback (SDRAM_HandleTypeDef * hsdram, HAL_SDRAM_CallbackIDTypeDef CallbackId, pSDRAM_CallbackTypeDef pCallback)

Function description

HAL_SDRAM_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_SDRAM_UnRegisterCallback (SDRAM_HandleTypeDef * hsdram, HAL_SDRAM_CallbackIDTypeDef CallbackId)

Function description

HAL_SDRAM_RegisterDmaCallback

Function name

HAL_StatusTypeDef HAL_SDRAM_RegisterDmaCallback (SDRAM_HandleTypeDef * hsdram, HAL_SDRAM_CallbackIDTypeDef CallbackId, pSDRAM_DmaCallbackTypeDef pCallback)

Function description

HAL_SDRAM_WriteProtection_Enable

Function name

HAL_StatusTypeDef HAL_SDRAM_WriteProtection_Enable (SDRAM_HandleTypeDef * hsdram)

Function description

HAL_SDRAM_WriteProtection_Disable

Function name

HAL_StatusTypeDef HAL_SDRAM_WriteProtection_Disable (SDRAM_HandleTypeDef * hsdram)

Function description

HAL_SDRAM_SendCommand

Function name

HAL_StatusTypeDef HAL_SDRAM_SendCommand (SDRAM_HandleTypeDef * hsdram, FMC_SDRAM_CommandTypeDef * Command, uint32_t Timeout)

Function description

HAL_SDRAM_ProgramRefreshRate

Function name

HAL_StatusTypeDef HAL_SDRAM_ProgramRefreshRate (SDRAM_HandleTypeDef * hsdram, uint32_t RefreshRate)

Function description

HAL_SDRAM_SetAutoRefreshNumber

Function name

HAL_StatusTypeDef HAL_SDRAM_SetAutoRefreshNumber (SDRAM_HandleTypeDef * hsdram, uint32_t AutoRefreshNumber)

Function description

HAL_SDRAM_GetModeStatus

Function name

uint32_t HAL_SDRAM_GetModeStatus (SDRAM_HandleTypeDef * hsdram)

Function description

HAL_SDRAM_GetState

Function name

HAL_SDRAM_StateTypeDef HAL_SDRAM_GetState (const SDRAM_HandleTypeDef * hsdram)

Function description

66.3 SDRAM Firmware driver defines

The following section lists the various define and macros of the module.

66.3.1 SDRAM

SDRAM

SDRAM Exported Macros

__HAL_SDRAM_RESET_HANDLE_STATE

Description:

- Reset SDRAM handle state.

Parameters:

- __HANDLE__: specifies the SDRAM handle.

Return value:

- None

67 HAL SD Generic Driver

67.1 SD Firmware driver registers structures

67.1.1 HAL_SD_CardInfoTypeDef

HAL_SD_CardInfoTypeDef is defined in the stm32n6xx_hal_sd.h

Data Fields

- ***uint32_t CardType***
- ***uint32_t CardVersion***
- ***uint32_t Class***
- ***uint32_t RelCardAdd***
- ***uint32_t BlockNbr***
- ***uint32_t BlockSize***
- ***uint32_t LogBlockNbr***
- ***uint32_t LogBlockSize***
- ***uint32_t CardSpeed***

Field Documentation

- ***uint32_t HAL_SD_CardInfoTypeDef::CardType***
  Specifies the card Type
- ***uint32_t HAL_SD_CardInfoTypeDef::CardVersion***
  Specifies the card version
- ***uint32_t HAL_SD_CardInfoTypeDef::Class***
  Specifies the class of the card class
- ***uint32_t HAL_SD_CardInfoTypeDef::RelCardAdd***
  Specifies the Relative Card Address
- ***uint32_t HAL_SD_CardInfoTypeDef::BlockNbr***
  Specifies the Card Capacity in blocks
- ***uint32_t HAL_SD_CardInfoTypeDef::BlockSize***
  Specifies one block size in bytes
- ***uint32_t HAL_SD_CardInfoTypeDef::LogBlockNbr***
  Specifies the Card logical Capacity in blocks
- ***uint32_t HAL_SD_CardInfoTypeDef::LogBlockSize***
  Specifies logical block size in bytes
- ***uint32_t HAL_SD_CardInfoTypeDef::CardSpeed***
  Specifies the card Speed

67.1.2 __SD_HandleTypeDef

__SD_HandleTypeDef is defined in the stm32n6xx_hal_sd.h

Data Fields

- ***SD_TypeDef * Instance***
- ***SD_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***const uint8_t * pTxBuffPtr***
- ***uint32_t TxXferSize***
- ***uint8_t * pRxBuffPtr***
- ***uint32_t RxXferSize***
- ***__IO uint32_t Context***
- ***__IO HAL_SD_StateTypeDef State***
- ***__IO uint32_t ErrorCode***
- ***HAL_SD_CardInfoTypeDef SdCard***

- ***uint32_t CSD***
- ***uint32_t CID***
- ***void(* TxCpltCallback***
- ***void(* RxCpltCallback***
- ***void(* ErrorCallback***
- ***void(* AbortCpltCallback***
- ***void(* Read_DMALnkLstBufCpltCallback***
- ***void(* Write_DMALnkLstBufCpltCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***SD_TypeDef* __SD_HandleTypeDef::Instance***
  SD registers base address
- ***SD_InitTypeDef __SD_HandleTypeDef::Init***
  SD required parameters
- ***HAL_LockTypeDef __SD_HandleTypeDef::Lock***
  SD locking object
- ***const uint8_t* __SD_HandleTypeDef::pTxBuffPtr***
  Pointer to SD Tx transfer Buffer
- ***uint32_t __SD_HandleTypeDef::TxXferSize***
  SD Tx Transfer size
- ***uint8_t* __SD_HandleTypeDef::pRxBuffPtr***
  Pointer to SD Rx transfer Buffer
- ***uint32_t __SD_HandleTypeDef::RxXferSize***
  SD Rx Transfer size
- ***__IO uint32_t __SD_HandleTypeDef::Context***
  SD transfer context
- ***__IO HAL_SD_StateTypeDef __SD_HandleTypeDef::State***
  SD card State
- ***__IO uint32_t __SD_HandleTypeDef::ErrorCode***
  SD Card Error codes
- ***HAL_SD_CardInfoTypeDef __SD_HandleTypeDef::SdCard***
  SD Card information
- ***uint32_t __SD_HandleTypeDef::CSD[4]***
  SD card specific data table
- ***uint32_t __SD_HandleTypeDef::CID[4]***
  SD card identification number table
- ***void(* __SD_HandleTypeDef::TxCpltCallback)(struct __SD_HandleTypeDef *hsd)***
- ***void(* __SD_HandleTypeDef::RxCpltCallback)(struct __SD_HandleTypeDef *hsd)***
- ***void(* __SD_HandleTypeDef::ErrorCallback)(struct __SD_HandleTypeDef *hsd)***
- ***void(* __SD_HandleTypeDef::AbortCpltCallback)(struct __SD_HandleTypeDef *hsd)***
- ***void(* __SD_HandleTypeDef::Read_DMALnkLstBufCpltCallback)(struct __SD_HandleTypeDef *hsd)***
- ***void(* __SD_HandleTypeDef::Write_DMALnkLstBufCpltCallback)(struct __SD_HandleTypeDef *hsd)***
- ***void(* __SD_HandleTypeDef::MspInitCallback)(struct __SD_HandleTypeDef *hsd)***
- ***void(* __SD_HandleTypeDef::MspDeInitCallback)(struct __SD_HandleTypeDef *hsd)***

67.1.3 HAL_SD_CardCSDTypeDef

HAL_SD_CardCSDTypeDef is defined in the stm32n6xx_hal_sd.h

Data Fields

- ***__IO uint8_t CSDStruct***
- ***__IO uint8_t SysSpecVersion***
- ***__IO uint8_t Reserved1***
- ***__IO uint8_t TAAC***

- ***__IO uint8_t NSAC***
- ***__IO uint8_t MaxBusClkFrec***
- ***__IO uint16_t CardComdClasses***
- ***__IO uint8_t RdBlockLen***
- ***__IO uint8_t PartBlockRead***
- ***__IO uint8_t WrBlockMisalign***
- ***__IO uint8_t RdBlockMisalign***
- ***__IO uint8_t DSRImpl***
- ***__IO uint8_t Reserved2***
- ***__IO uint32_t DeviceSize***
- ***__IO uint8_t MaxRdCurrentVDDMin***
- ***__IO uint8_t MaxRdCurrentVDDMax***
- ***__IO uint8_t MaxWrCurrentVDDMin***
- ***__IO uint8_t MaxWrCurrentVDDMax***
- ***__IO uint8_t DeviceSizeMul***
- ***__IO uint8_t EraseGrSize***
- ***__IO uint8_t EraseGrMul***
- ***__IO uint8_t WrProtectGrSize***
- ***__IO uint8_t WrProtectGrEnable***
- ***__IO uint8_t ManDeflECC***
- ***__IO uint8_t WrSpeedFact***
- ***__IO uint8_t MaxWrBlockLen***
- ***__IO uint8_t WriteBlockPaPartial***
- ***__IO uint8_t Reserved3***
- ***__IO uint8_t ContentProtectAppli***
- ***__IO uint8_t FileFormatGroup***
- ***__IO uint8_t CopyFlag***
- ***__IO uint8_t PermWrProtect***
- ***__IO uint8_t TempWrProtect***
- ***__IO uint8_t FileFormat***
- ***__IO uint8_t ECC***
- ***__IO uint8_t CSD_CRC***
- ***__IO uint8_t Reserved4***

Field Documentation

- ***__IO uint8_t HAL_SD_CardCSDTypeDef::CSDStruct***
  CSD structure
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::SysSpecVersion***
  System specification version
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::Reserved1***
  Reserved
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::TAAC***
  Data read access time 1
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::NSAC***
  Data read access time 2 in CLK cycles
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::MaxBusClkFrec***
  Max. bus clock frequency
- ***__IO uint16_t HAL_SD_CardCSDTypeDef::CardComdClasses***
  Card command classes
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::RdBlockLen***
  Max. read data block length
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::PartBlockRead***
  Partial blocks for read allowed

- ***__IO uint8_t HAL_SD_CardCSDTypeDef::WrBlockMisalign***
  Write block misalignment
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::RdBlockMisalign***
  Read block misalignment
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::DSRImpl***
  DSR implemented
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::Reserved2***
  Reserved
- ***__IO uint32_t HAL_SD_CardCSDTypeDef::DeviceSize***
  Device Size
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::MaxRdCurrentVDDMin***
  Max. read current @ VDD min
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::MaxRdCurrentVDDMax***
  Max. read current @ VDD max
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::MaxWrCurrentVDDMin***
  Max. write current @ VDD min
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::MaxWrCurrentVDDMax***
  Max. write current @ VDD max
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::DeviceSizeMul***
  Device size multiplier
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::EraseGrSize***
  Erase group size
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::EraseGrMul***
  Erase group size multiplier
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::WrProtectGrSize***
  Write protect group size
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::WrProtectGrEnable***
  Write protect group enable
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::ManDeflECC***
  Manufacturer default ECC
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::WrSpeedFact***
  Write speed factor
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::MaxWrBlockLen***
  Max. write data block length
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::WriteBlockPaPartial***
  Partial blocks for write allowed
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::Reserved3***
  Reserved
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::ContentProtectAppli***
  Content protection application
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::FileFormatGroup***
  File format group
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::CopyFlag***
  Copy flag (OTP)
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::PermWrProtect***
  Permanent write protection
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::TempWrProtect***
  Temporary write protection
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::FileFormat***
  File format
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::ECC***
  ECC code
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::CSD_CRC***
  CSD CRC
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::Reserved4***
  Always 1

67.1.4 HAL_SD_CardCIDTypeDef

HAL_SD_CardCIDTypeDef is defined in the stm32n6xx_hal_sd.h

Data Fields

- ***__IO uint8_t ManufacturerID***
- ***__IO uint16_t OEM_AppliID***
- ***__IO uint32_t ProdName1***
- ***__IO uint8_t ProdName2***
- ***__IO uint8_t ProdRev***
- ***__IO uint32_t ProdSN***
- ***__IO uint8_t Reserved1***
- ***__IO uint16_t ManufactDate***
- ***__IO uint8_t CID_CRC***
- ***__IO uint8_t Reserved2***

Field Documentation

- ***__IO uint8_t HAL_SD_CardCIDTypeDef::ManufacturerID***
  Manufacturer ID
- ***__IO uint16_t HAL_SD_CardCIDTypeDef::OEM_AppliID***
  OEM/Application ID
- ***__IO uint32_t HAL_SD_CardCIDTypeDef::ProdName1***
  Product Name part1
- ***__IO uint8_t HAL_SD_CardCIDTypeDef::ProdName2***
  Product Name part2
- ***__IO uint8_t HAL_SD_CardCIDTypeDef::ProdRev***
  Product Revision
- ***__IO uint32_t HAL_SD_CardCIDTypeDef::ProdSN***
  Product Serial Number
- ***__IO uint8_t HAL_SD_CardCIDTypeDef::Reserved1***
  Reserved1
- ***__IO uint16_t HAL_SD_CardCIDTypeDef::ManufactDate***
  Manufacturing Date
- ***__IO uint8_t HAL_SD_CardCIDTypeDef::CID_CRC***
  CID CRC
- ***__IO uint8_t HAL_SD_CardCIDTypeDef::Reserved2***
  Always 1

67.1.5 HAL_SD_CardStatusTypeDef

HAL_SD_CardStatusTypeDef is defined in the stm32n6xx_hal_sd.h

Data Fields

- ***__IO uint8_t DataBusWidth***
- ***__IO uint8_t SecuredMode***
- ***__IO uint16_t CardType***
- ***__IO uint32_t ProtectedAreaSize***
- ***__IO uint8_t SpeedClass***
- ***__IO uint8_t PerformanceMove***
- ***__IO uint8_t AllocationUnitSize***
- ***__IO uint16_t EraseSize***
- ***__IO uint8_t EraseTimeout***
- ***__IO uint8_t EraseOffset***
- ***__IO uint8_t UhsSpeedGrade***
- ***__IO uint8_t UhsAllocationUnitSize***
- ***__IO uint8_t VideoSpeedClass***

Field Documentation

- ***__IO uint8_t HAL_SD_CardStatusTypeDef::DataBusWidth***
  Shows the currently defined data bus width
- ***__IO uint8_t HAL_SD_CardStatusTypeDef::SecuredMode***
  Card is in secured mode of operation
- ***__IO uint16_t HAL_SD_CardStatusTypeDef::CardType***
  Carries information about card type
- ***__IO uint32_t HAL_SD_CardStatusTypeDef::ProtectedAreaSize***
  Carries information about the capacity of protected area
- ***__IO uint8_t HAL_SD_CardStatusTypeDef::SpeedClass***
  Carries information about the speed class of the card
- ***__IO uint8_t HAL_SD_CardStatusTypeDef::PerformanceMove***
  Carries information about the card's performance move
- ***__IO uint8_t HAL_SD_CardStatusTypeDef::AllocationUnitSize***
  Carries information about the card's allocation unit size
- ***__IO uint16_t HAL_SD_CardStatusTypeDef::EraseSize***
  Determines the number of AUs to be erased in one operation
- ***__IO uint8_t HAL_SD_CardStatusTypeDef::EraseTimeout***
  Determines the timeout for any number of AU erase
- ***__IO uint8_t HAL_SD_CardStatusTypeDef::EraseOffset***
  Carries information about the erase offset
- ***__IO uint8_t HAL_SD_CardStatusTypeDef::UhsSpeedGrade***
  Carries information about the speed grade of UHS card
- ***__IO uint8_t HAL_SD_CardStatusTypeDef::UhsAllocationUnitSize***
  Carries information about the UHS card's allocation unit size
- ***__IO uint8_t HAL_SD_CardStatusTypeDef::VideoSpeedClass***
  Carries information about the Video Speed Class of UHS card

67.2 SD Firmware driver API description

The following section lists the various functions of the SD library.

67.2.1 How to use this driver

This driver implements a high level communication layer for read and write from/to this memory. The needed STM32 hardware resources (SDMMC and GPIO) are performed by the user in HAL_SD_MspInit() function (MSP layer). Basically, the MSP layer configuration should be the same as we provide in the examples. You can easily tailor this configuration according to hardware resources.

This driver is a generic layered driver for SDMMC memories which uses the HAL SDMMC driver functions to interface with SD and uSD cards devices. It is used as follows:

1. Initialize the SDMMC low level resources by implementing the HAL_SD_MspInit() API:
   a. Enable the SDMMC interface clock using __HAL_RCC_SDMMC_CLK_ENABLE();
   b. SDMMC pins configuration for SD card
      - Enable the clock for the SDMMC GPIOs using the functions __HAL_RCC_GPIOx_CLK_ENABLE();
      - Configure these SDMMC pins as alternate function pull-up using HAL_GPIO_Init() and according to your pin assignment;
   c. NVIC configuration if you need to use interrupt process (HAL_SD_ReadBlocks_IT() and HAL_SD_WriteBlocks_IT() APIs).
      - Configure the SDMMC interrupt priorities using function HAL_NVIC_SetPriority();
      - Enable the NVIC SDMMC IRQs using function HAL_NVIC_EnableIRQ()
      - SDMMC interrupts are managed using the macros __HAL_SD_ENABLE_IT() and __HAL_SD_DISABLE_IT() inside the communication process.
      - SDMMC interrupts pending bits are managed using the macros __HAL_SD_GET_IT() and __HAL_SD_CLEAR_IT()
   d. No general propose DMA Configuration is needed, an Internal DMA for SDMMC Peripheral are used.
2. At this stage, you can perform SD read/write/erase operations after SD card initialization

SD Card Initialization and configuration

To initialize the SD Card, use the HAL_SD_Init() function. It Initializes SDMMC Peripheral(STM32 side) and the SD Card, and put it into StandBy State (Ready for data transfer). This function provide the following operations:

1. Apply the SD Card initialization process at 400KHz and check the SD Card type (Standard Capacity or High Capacity). You can change or adapt this frequency by adjusting the "ClockDiv" field. The SD Card frequency (SDMMC_CK) is computed as follows: SDMMC_CK = SDMMCCLK / (2 * ClockDiv) In initialization mode and according to the SD Card standard, make sure that the SDMMC_CK frequency doesn't exceed 400KHz. This phase of initialization is done through SDMMC_Init() and SDMMC_PowerState_ON() SDMMC low level APIs.
2. Initialize the SD card. The API used is HAL_SD_InitCard(). This phase allows the card initialization and identification and check the SD Card type (Standard Capacity or High Capacity) The initialization flow is compatible with SD standard. This API (HAL_SD_InitCard()) could be used also to reinitialize the card in case of plug-off plug-in.
3. Configure the SD Card Data transfer frequency. You can change or adapt this frequency by adjusting the "ClockDiv" field. In transfer mode and according to the SD Card standard, make sure that the SDMMC_CK frequency doesn't exceed 25MHz and 100MHz in High-speed mode switch.
4. Select the corresponding SD Card according to the address read with the step 2.
5. Configure the SD Card in wide bus mode: 4-bits data.

SD Card Read operation

- You can read from SD card in polling mode by using function HAL_SD_ReadBlocks(). This function support only 512-bytes block length (the block size should be chosen as 512 bytes). You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter. After this, you have to ensure that the transfer is done correctly. The check is done through HAL_SD_GetCardState() function for SD card state.
- You can read from SD card in DMA mode by using function HAL_SD_ReadBlocks_DMA(). This function support only 512-bytes block length (the block size should be chosen as 512 bytes). You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter. After this, you have to ensure that the transfer is done correctly. The check is done through HAL_SD_GetCardState() function for SD card state. You could also check the DMA transfer process through the SD Rx interrupt event.
- You can read from SD card in Interrupt mode by using function HAL_SD_ReadBlocks_IT(). This function support only 512-bytes block length (the block size should be chosen as 512 bytes). You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter. After this, you have to ensure that the transfer is done correctly. The check is done through HAL_SD_GetCardState() function for SD card state. You could also check the IT transfer process through the SD Rx interrupt event.

SD Card Write operation

- You can write to SD card in polling mode by using function HAL_SD_WriteBlocks(). This function support only 512-bytes block length (the block size should be chosen as 512 bytes). You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter. After this, you have to ensure that the transfer is done correctly. The check is done through HAL_SD_GetCardState() function for SD card state.
- You can write to SD card in DMA mode by using function HAL_SD_WriteBlocks_DMA(). This function support only 512-bytes block length (the block size should be chosen as 512 bytes). You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter. After this, you have to ensure that the transfer is done correctly. The check is done through HAL_SD_GetCardState() function for SD card state. You could also check the DMA transfer process through the SD Tx interrupt event.
- You can write to SD card in Interrupt mode by using function HAL_SD_WriteBlocks_IT(). This function support only 512-bytes block length (the block size should be chosen as 512 bytes). You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter. After this, you have to ensure that the transfer is done correctly. The check is done through HAL_SD_GetCardState() function for SD card state. You could also check the IT transfer process through the SD Tx interrupt event.

SD card status

- The SD Status contains status bits that are related to the SD Memory Card proprietary features. To get SD card status use the HAL_SD_GetCardStatus().

SD card information

- To get SD card information, you can use the function HAL_SD_GetCardInfo(). It returns useful information about the SD card such as block size, card type, block number ...

SD card CSD register

SD card CID register

SD HAL driver macros list

Note: You can refer to the SD HAL driver header file for more useful macros

Callback registration

The compilation define USE_HAL_SD_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use Functions HAL_SD_RegisterCallback() to register a user callback, it allows to register following callbacks:

- TxCpltCallback : callback when a transmission transfer is completed.
- RxCpltCallback : callback when a reception transfer is completed.
- ErrorCallback : callback when error occurs.
- AbortCpltCallback : callback when abort is completed.
- Read_DMADblBuf0CpltCallback : callback when the DMA reception of first buffer is completed.
- Read_DMADblBuf1CpltCallback : callback when the DMA reception of second buffer is completed.
- Write_DMADblBuf0CpltCallback : callback when the DMA transmission of first buffer is completed.
- Write_DMADblBuf1CpltCallback : callback when the DMA transmission of second buffer is completed.
- MspInitCallback : SD MspInit.
- MspDeInitCallback : SD MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function. For specific callbacks TransceiverCallback use dedicated register callbacks: respectively HAL_SD_RegisterTransceiverCallback(). Use function HAL_SD_UnRegisterCallback() to reset a callback to the default weak (overridden) function. It allows to reset following callbacks:
- TxCpltCallback : callback when a transmission transfer is completed.
- RxCpltCallback : callback when a reception transfer is completed.
- ErrorCallback : callback when error occurs.
- AbortCpltCallback : callback when abort is completed.
- Read_DMALnkLstBufCpltCallback : callback when the DMA reception of linked list node buffer is completed.
- Write_DMALnkLstBufCpltCallback : callback when the DMA transmission of linked list node buffer is completed.
- MspInitCallback : SD MspInit.

- MspDeInitCallback : SD MspDeInit. This function) takes as parameters the HAL peripheral handle and the Callback ID. For specific callbacks TransceiverCallback use dedicated unregister callbacks: respectively HAL_SD_UnRegisterTransceiverCallback(). By default, after the HAL_SD_Init and if the state is HAL_SD_STATE_RESET all callbacks are reset to the corresponding legacy weak (overridden) functions. Exception done for MspInit and MspDeInit callbacks that are respectively reset to the legacy weak (overridden) functions in the HAL_SD_Init and HAL_SD_DeInit only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the HAL_SD_Init and HAL_SD_DeInit keep and use the user MspInit/MspDeInit callbacks (registered beforehand) Callbacks can be registered/ unregistered in READY state only. Exception done for MspInit/MspDeInit callbacks that can be registered/ unregistered in READY or RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_SD_RegisterCallback before calling HAL_SD_DeInit or HAL_SD_Init function. When The compilation define USE_HAL_SD_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak (overridden) callbacks are used.

67.2.2 Detailed description of functions

HAL_SD_Init

Function name

HAL_StatusTypeDef HAL_SD_Init (SD_HandleTypeDef * hsd)

Function description

HAL_SD_InitCard

Function name

HAL_StatusTypeDef HAL_SD_InitCard (SD_HandleTypeDef * hsd)

Function description

HAL_SD_DeInit

Function name

HAL_StatusTypeDef HAL_SD_DeInit (SD_HandleTypeDef * hsd)

Function description

HAL_SD_MspInit

Function name

void HAL_SD_MspInit (SD_HandleTypeDef * hsd)

Function description

HAL_SD_MspDeInit

Function name

void HAL_SD_MspDeInit (SD_HandleTypeDef * hsd)

Function description

HAL_SD_ReadBlocks

Function name

HAL_StatusTypeDef HAL_SD_ReadBlocks (SD_HandleTypeDef * hsd, uint8_t * pData, uint32_t BlockAdd, uint32_t NumberOfBlocks, uint32_t Timeout)

Function description

HAL_SD_WriteBlocks

Function name

HAL_StatusTypeDef HAL_SD_WriteBlocks (SD_HandleTypeDef * hsd, const uint8_t * pData, uint32_t BlockAdd, uint32_t NumberOfBlocks, uint32_t Timeout)

Function description

HAL_SD_Erase

Function name

HAL_StatusTypeDef HAL_SD_Erase (SD_HandleTypeDef * hsd, uint32_t BlockStartAdd, uint32_t BlockEndAdd)

Function description

HAL_SD_ReadBlocks_IT

Function name

HAL_StatusTypeDef HAL_SD_ReadBlocks_IT (SD_HandleTypeDef * hsd, uint8_t * pData, uint32_t BlockAdd, uint32_t NumberOfBlocks)

Function description

HAL_SD_WriteBlocks_IT

Function name

HAL_StatusTypeDef HAL_SD_WriteBlocks_IT (SD_HandleTypeDef * hsd, const uint8_t * pData, uint32_t BlockAdd, uint32_t NumberOfBlocks)

Function description

HAL_SD_ReadBlocks_DMA

Function name

HAL_StatusTypeDef HAL_SD_ReadBlocks_DMA (SD_HandleTypeDef * hsd, uint8_t * pData, uint32_t BlockAdd, uint32_t NumberOfBlocks)

Function description

HAL_SD_WriteBlocks_DMA

Function name

HAL_StatusTypeDef HAL_SD_WriteBlocks_DMA (SD_HandleTypeDef * hsd, const uint8_t * pData, uint32_t BlockAdd, uint32_t NumberOfBlocks)

Function description

HAL_SD_IRQHandler

Function name

void HAL_SD_IRQHandler (SD_HandleTypeDef * hsd)

Function description

HAL_SD_TxCpltCallback

Function name

void HAL_SD_TxCpltCallback (SD_HandleTypeDef * hsd)

Function description

HAL_SD_RxCpltCallback

Function name

void HAL_SD_RxCpltCallback (SD_HandleTypeDef * hsd)

Function description

HAL_SD_ErrorCallback

Function name

void HAL_SD_ErrorCallback (SD_HandleTypeDef * hsd)

Function description

HAL_SD_AbortCallback

Function name

void HAL_SD_AbortCallback (SD_HandleTypeDef * hsd)

Function description

HAL_SD_RegisterCallback

Function name

HAL_StatusTypeDef HAL_SD_RegisterCallback (SD_HandleTypeDef * hsd, HAL_SD_CallbackIDTypeDef CallbackID, pSD_CallbackTypeDef pCallback)

Function description

HAL_SD_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_SD_UnRegisterCallback (SD_HandleTypeDef * hsd, HAL_SD_CallbackIDTypeDef CallbackID)

Function description

HAL_SD_ConfigWideBusOperation

Function name

HAL_StatusTypeDef HAL_SD_ConfigWideBusOperation (SD_HandleTypeDef * hsd, uint32_t WideMode)

Function description

HAL_SD_ConfigSpeedBusOperation

Function name

HAL_StatusTypeDef HAL_SD_ConfigSpeedBusOperation (SD_HandleTypeDef * hsd, uint32_t SpeedMode)

Function description

HAL_SD_GetCardState

Function name

HAL_SD_CardStateTypeDef HAL_SD_GetCardState (SD_HandleTypeDef * hsd)

Function description

HAL_SD_GetCardCID

Function name

HAL_StatusTypeDef HAL_SD_GetCardCID (const SD_HandleTypeDef * hsd, HAL_SD_CardCIDTypeDef * pCID)

Function description

HAL_SD_GetCardCSD

Function name

HAL_StatusTypeDef HAL_SD_GetCardCSD (SD_HandleTypeDef * hsd, HAL_SD_CardCSDTypeDef * pCSD)

Function description

HAL_SD_GetCardStatus

Function name

HAL_StatusTypeDef HAL_SD_GetCardStatus (SD_HandleTypeDef * hsd, HAL_SD_CardStatusTypeDef * pStatus)

Function description

HAL_SD_GetCardInfo

Function name

HAL_StatusTypeDef HAL_SD_GetCardInfo (const SD_HandleTypeDef * hsd, HAL_SD_CardInfoTypeDef * pCardInfo)

Function description

HAL_SD_GetState

Function name

HAL_SD_StateTypeDef HAL_SD_GetState (const SD_HandleTypeDef * hsd)

Function description

HAL_SD_GetError

Function name

uint32_t HAL_SD_GetError (const SD_HandleTypeDef * hsd)

Function description

HAL_SD_Abort

Function name

HAL_StatusTypeDef HAL_SD_Abort (SD_HandleTypeDef * hsd)

Function description

HAL_SD_Abort_IT

Function name

HAL_StatusTypeDef HAL_SD_Abort_IT (SD_HandleTypeDef * hsd)

Function description

67.3 SD Firmware driver defines

The following section lists the various define and macros of the module.

67.3.1 SD

SD

SD Error status enumeration Structure definition

HAL_SD_ERROR_NONE

No error

HAL_SD_ERROR_CMD_CRC_FAIL

Command response received (but CRC check failed)

HAL_SD_ERROR_DATA_CRC_FAIL

Data block sent/received (CRC check failed)

HAL_SD_ERROR_CMD_RSP_TIMEOUT

Command response timeout

HAL_SD_ERROR_DATA_TIMEOUT

Data timeout

HAL_SD_ERROR_TX_UNDERRUN

Transmit FIFO underrun

HAL_SD_ERROR_RX_OVERRUN

Receive FIFO overrun

HAL_SD_ERROR_ADDR_MISALIGNED

Misaligned address

HAL_SD_ERROR_BLOCK_LEN_ERR

Transferred block length is not allowed for the card or the number of transferred bytes does not match the block length

HAL_SD_ERROR_ERASE_SEQ_ERR

An error in the sequence of erase command occurs

HAL_SD_ERROR_BAD_ERASE_PARAM

An invalid selection for erase groups

HAL_SD_ERROR_WRITE_PROT_VIOLATION

Attempt to program a write protect block

HAL_SD_ERROR_LOCK_UNLOCK_FAILED

Sequence or password error has been detected in unlock command or if there was an attempt to access a locked card

HAL_SD_ERROR_COM_CRC_FAILED

CRC check of the previous command failed

HAL_SD_ERROR_ILLEGAL_CMD

Command is not legal for the card state

HAL_SD_ERROR_CARD_ECC_FAILED

Card internal ECC was applied but failed to correct the data

HAL_SD_ERROR_CC_ERR

Internal card controller error

HAL_SD_ERROR_GENERAL_UNKNOWN_ERR

General or unknown error

HAL_SD_ERROR_STREAM_READ_UNDERRUN

The card could not sustain data reading in stream rmode

HAL_SD_ERROR_STREAM_WRITE_OVERRUN

The card could not sustain data programming in stream mode

HAL_SD_ERROR_CID_CSD_OVERWRITE

CID/CSD overwrite error

HAL_SD_ERROR_WP_ERASE_SKIP

Only partial address space was erased

HAL_SD_ERROR_CARD_ECC_DISABLED

Command has been executed without using internal ECC

HAL_SD_ERROR_ERASE_RESET

Erase sequence was cleared before executing because an out of erase sequence command was received

HAL_SD_ERROR_AKE_SEQ_ERR

Error in sequence of authentication

HAL_SD_ERROR_INVALID_VOLTRANGE

Error in case of invalid voltage range

HAL_SD_ERROR_ADDR_OUT_OF_RANGE

Error when addressed block is out of range

HAL_SD_ERROR_REQUEST_NOT_APPLICABLE

Error when command request is not applicable

HAL_SD_ERROR_PARAM

the used parameter is not valid

HAL_SD_ERROR_UNSUPPORTED_FEATURE

Error when feature is not insupported

HAL_SD_ERROR_BUSY

Error when transfer process is busy

HAL_SD_ERROR_DMA

Error while DMA transfer

HAL_SD_ERROR_TIMEOUT

Timeout error

HAL_SD_ERROR_INVALID_CALLBACK

Invalid callback error

SD context enumeration

SD_CONTEXT_NONE

None

SD_CONTEXT_READ_SINGLE_BLOCK

Read single block operation

SD_CONTEXT_READ_MULTIPLE_BLOCK

Read multiple blocks operation

SD_CONTEXT_WRITE_SINGLE_BLOCK

Write single block operation

SD_CONTEXT_WRITE_MULTIPLE_BLOCK

Write multiple blocks operation

SD_CONTEXT_IT

Process in Interrupt mode

SD_CONTEXT_DMA

Process in DMA mode

SD Supported Memory Cards

CARD_NORMAL_SPEED

Normal Speed Card <12.5Mo/s , Spec Version 1.01

CARD_HIGH_SPEED

High Speed Card <25Mo/s , Spec version 2.00

CARD_ULTRA_HIGH_SPEED

UHS-I SD Card <50Mo/s for SDR50, DDR5 Cards and <104Mo/s for SDR104, Spec version 3.01

CARD_SDSC

SD Standard Capacity <2Go

CARD_SDHC_SDXC

SD High Capacity <32Go, SD Extended Capacity <2To

CARD_SECURED

SD Supported Version

CARD_V1_X

CARD_V2_X

SD Exported Constants

BLOCKSIZE

Block size is 512 bytes

SD Exported Macros

__HAL_SD_RESET_HANDLE_STATE

Description:

- Reset SD handle state.

Parameters:

- __HANDLE__: SD Handle.

Return value:

- None

__HAL_SD_ENABLE_IT

Description:

- Enable the SD device interrupt.

Parameters:

- __HANDLE__: SD Handle.
- __INTERRUPT__: specifies the SDMMC interrupt sources to be enabled. This parameter can be one or a combination of the following values:
  - SDMMC_IT_CCRCFAIL: Command response received (CRC check failed) interrupt
  - SDMMC_IT_DCRCFAIL: Data block sent/received (CRC check failed) interrupt
  - SDMMC_IT_CTIMEOUT: Command response timeout interrupt
  - SDMMC_IT_DTIMEOUT: Data timeout interrupt
  - SDMMC_IT_TXUNDERR: Transmit FIFO underrun error interrupt
  - SDMMC_IT_RXOVERR: Received FIFO overrun error interrupt
  - SDMMC_IT_CMDREND: Command response received (CRC check passed) interrupt
  - SDMMC_IT_CMDSENT: Command sent (no response required) interrupt
  - SDMMC_IT_DATAEND: Data end (data counter, DATACOUNT, is zero) interrupt
  - SDMMC_IT_DHOLD: Data transfer Hold interrupt
  - SDMMC_IT_DBCKEND: Data block sent/received (CRC check passed) interrupt
  - SDMMC_IT_DABORT: Data transfer aborted by CMD12 interrupt
  - SDMMC_IT_TXFIFOHE: Transmit FIFO Half Empty interrupt
  - SDMMC_IT_RXFIFOHF: Receive FIFO Half Full interrupt
  - SDMMC_IT_RXFIFOF: Receive FIFO full interrupt
  - SDMMC_IT_TXFIFOE: Transmit FIFO empty interrupt
  - SDMMC_IT_BUSYD0END: End of SDMMC_D0 Busy following a CMD response detected interrupt
  - SDMMC_IT_SDIOIT: SDIO interrupt received interrupt
  - SDMMC_IT_ACKFAIL: Boot Acknowledgment received interrupt
  - SDMMC_IT_ACKTIMEOUT: Boot Acknowledgment timeout interrupt
  - SDMMC_IT_VSWEND: Voltage switch critical timing section completion interrupt
  - SDMMC_IT_CKSTOP: SDMMC_CK stopped in Voltage switch procedure interrupt
  - SDMMC_IT_IDMABTC: IDMA buffer transfer complete interrupt

Return value:

- None

__HAL_SD_DISABLE_IT

Description:

- Disable the SD device interrupt.

Parameters:

- __HANDLE__: SD Handle.
- __INTERRUPT__: specifies the SDMMC interrupt sources to be disabled. This parameter can be one or a combination of the following values:
  - SDMMC_IT_CCRCFAIL: Command response received (CRC check failed) interrupt
  - SDMMC_IT_DCRCFAIL: Data block sent/received (CRC check failed) interrupt
  - SDMMC_IT_CTIMEOUT: Command response timeout interrupt
  - SDMMC_IT_DTIMEOUT: Data timeout interrupt
  - SDMMC_IT_TXUNDERR: Transmit FIFO underrun error interrupt
  - SDMMC_IT_RXOVERR: Received FIFO overrun error interrupt
  - SDMMC_IT_CMDREND: Command response received (CRC check passed) interrupt
  - SDMMC_IT_CMDSENT: Command sent (no response required) interrupt
  - SDMMC_IT_DATAEND: Data end (data counter, DATACOUNT, is zero) interrupt
  - SDMMC_IT_DHOLD: Data transfer Hold interrupt
  - SDMMC_IT_DBCKEND: Data block sent/received (CRC check passed) interrupt
  - SDMMC_IT_DABORT: Data transfer aborted by CMD12 interrupt
  - SDMMC_IT_TXFIFOHE: Transmit FIFO Half Empty interrupt
  - SDMMC_IT_RXFIFOHF: Receive FIFO Half Full interrupt
  - SDMMC_IT_RXFIFOF: Receive FIFO full interrupt
  - SDMMC_IT_TXFIFOE: Transmit FIFO empty interrupt
  - SDMMC_IT_BUSYD0END: End of SDMMC_D0 Busy following a CMD response detected interrupt
  - SDMMC_IT_SDIOIT: SDIO interrupt received interrupt
  - SDMMC_IT_ACKFAIL: Boot Acknowledgment received interrupt
  - SDMMC_IT_ACKTIMEOUT: Boot Acknowledgment timeout interrupt
  - SDMMC_IT_VSWEND: Voltage switch critical timing section completion interrupt
  - SDMMC_IT_CKSTOP: SDMMC_CK stopped in Voltage switch procedure interrupt
  - SDMMC_IT_IDMABTC: IDMA buffer transfer complete interrupt

Return value:

- None

__HAL_SD_GET_FLAG

Description:

- Check whether the specified SD flag is set or not.

Parameters:

- __HANDLE__: SD Handle.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - SDMMC_FLAG_CCRCFAIL: Command response received (CRC check failed)
  - SDMMC_FLAG_DCRCFAIL: Data block sent/received (CRC check failed)
  - SDMMC_FLAG_CTIMEOUT: Command response timeout
  - SDMMC_FLAG_DTIMEOUT: Data timeout
  - SDMMC_FLAG_TXUNDERR: Transmit FIFO underrun error
  - SDMMC_FLAG_RXOVERR: Received FIFO overrun error
  - SDMMC_FLAG_CMDREND: Command response received (CRC check passed)
  - SDMMC_FLAG_CMDSENT: Command sent (no response required)
  - SDMMC_FLAG_DATAEND: Data end (data counter, DATACOUNT, is zero)
  - SDMMC_FLAG_DHOLD: Data transfer Hold
  - SDMMC_FLAG_DBCKEND: Data block sent/received (CRC check passed)
  - SDMMC_FLAG_DABORT: Data transfer aborted by CMD12
  - SDMMC_FLAG_DPSMACT: Data path state machine active
  - SDMMC_FLAG_CPSMACT: Command path state machine active
  - SDMMC_FLAG_TXFIFOHE: Transmit FIFO Half Empty
  - SDMMC_FLAG_RXFIFOHF: Receive FIFO Half Full
  - SDMMC_FLAG_TXFIFOF: Transmit FIFO full
  - SDMMC_FLAG_RXFIFOF: Receive FIFO full
  - SDMMC_FLAG_TXFIFOE: Transmit FIFO empty
  - SDMMC_FLAG_RXFIFOE: Receive FIFO empty
  - SDMMC_FLAG_BUSYD0: Inverted value of SDMMC_D0 line (Busy)
  - SDMMC_FLAG_BUSYD0END: End of SDMMC_D0 Busy following a CMD response detected
  - SDMMC_FLAG_SDIOIT: SDIO interrupt received
  - SDMMC_FLAG_ACKFAIL: Boot Acknowledgment received
  - SDMMC_FLAG_ACKTIMEOUT: Boot Acknowledgment timeout
  - SDMMC_FLAG_VSWEND: Voltage switch critical timing section completion
  - SDMMC_FLAG_CKSTOP: SDMMC_CK stopped in Voltage switch procedure
  - SDMMC_FLAG_IDMATE: IDMA transfer error
  - SDMMC_FLAG_IDMABTC: IDMA buffer transfer complete

Return value:

- The: new state of SD FLAG (SET or RESET).

__HAL_SD_CLEAR_FLAG

Description:

- Clear the SD's pending flags.

Parameters:

- __HANDLE__: SD Handle.
- __FLAG__: specifies the flag to clear. This parameter can be one or a combination of the following values:
  - SDMMC_FLAG_CCRCFAIL: Command response received (CRC check failed)
  - SDMMC_FLAG_DCRCFAIL: Data block sent/received (CRC check failed)
  - SDMMC_FLAG_CTIMEOUT: Command response timeout
  - SDMMC_FLAG_DTIMEOUT: Data timeout
  - SDMMC_FLAG_TXUNDERR: Transmit FIFO underrun error
  - SDMMC_FLAG_RXOVERR: Received FIFO overrun error
  - SDMMC_FLAG_CMDREND: Command response received (CRC check passed)
  - SDMMC_FLAG_CMDSENT: Command sent (no response required)
  - SDMMC_FLAG_DATAEND: Data end (data counter, DATACOUNT, is zero)
  - SDMMC_FLAG_DHOLD: Data transfer Hold
  - SDMMC_FLAG_DBCKEND: Data block sent/received (CRC check passed)
  - SDMMC_FLAG_DABORT: Data transfer aborted by CMD12
  - SDMMC_FLAG_BUSYD0END: End of SDMMC_D0 Busy following a CMD response detected
  - SDMMC_FLAG_SDIOIT: SDIO interrupt received
  - SDMMC_FLAG_ACKFAIL: Boot Acknowledgment received
  - SDMMC_FLAG_ACKTIMEOUT: Boot Acknowledgment timeout
  - SDMMC_FLAG_VSWEND: Voltage switch critical timing section completion
  - SDMMC_FLAG_CKSTOP: SDMMC_CK stopped in Voltage switch procedure
  - SDMMC_FLAG_IDMATE: IDMA transfer error
  - SDMMC_FLAG_IDMABTC: IDMA buffer transfer complete

Return value:

- None

__HAL_SD_GET_IT

Description:

- Check whether the specified SD interrupt has occurred or not.

Parameters:

- __HANDLE__: SD Handle.
- __INTERRUPT__: specifies the SDMMC interrupt source to check. This parameter can be one of the following values:
  - SDMMC_IT_CCRCFAIL: Command response received (CRC check failed) interrupt
  - SDMMC_IT_DCRCFAIL: Data block sent/received (CRC check failed) interrupt
  - SDMMC_IT_CTIMEOUT: Command response timeout interrupt
  - SDMMC_IT_DTIMEOUT: Data timeout interrupt
  - SDMMC_IT_TXUNDERR: Transmit FIFO underrun error interrupt
  - SDMMC_IT_RXOVERR: Received FIFO overrun error interrupt
  - SDMMC_IT_CMDREND: Command response received (CRC check passed) interrupt
  - SDMMC_IT_CMDSENT: Command sent (no response required) interrupt
  - SDMMC_IT_DATAEND: Data end (data counter, DATACOUNT, is zero) interrupt
  - SDMMC_IT_DHOLD: Data transfer Hold interrupt
  - SDMMC_IT_DBCKEND: Data block sent/received (CRC check passed) interrupt
  - SDMMC_IT_DABORT: Data transfer aborted by CMD12 interrupt
  - SDMMC_IT_TXFIFOHE: Transmit FIFO Half Empty interrupt
  - SDMMC_IT_RXFIFOHF: Receive FIFO Half Full interrupt
  - SDMMC_IT_RXFIFOF: Receive FIFO full interrupt
  - SDMMC_IT_TXFIFOE: Transmit FIFO empty interrupt
  - SDMMC_IT_BUSYD0END: End of SDMMC_D0 Busy following a CMD response detected interrupt
  - SDMMC_IT_SDIOIT: SDIO interrupt received interrupt
  - SDMMC_IT_ACKFAIL: Boot Acknowledgment received interrupt
  - SDMMC_IT_ACKTIMEOUT: Boot Acknowledgment timeout interrupt
  - SDMMC_IT_VSWEND: Voltage switch critical timing section completion interrupt
  - SDMMC_IT_CKSTOP: SDMMC_CK stopped in Voltage switch procedure interrupt
  - SDMMC_IT_IDMABTC: IDMA buffer transfer complete interrupt

Return value:

- The: new state of SD IT (SET or RESET).

__HAL_SD_CLEAR_IT

Description:

- Clear the SD's interrupt pending bits.

Parameters:

- __HANDLE__: SD Handle.
- __INTERRUPT__: specifies the interrupt pending bit to clear. This parameter can be one or a combination of the following values:
  - SDMMC_IT_CCRCFAIL: Command response received (CRC check failed) interrupt
  - SDMMC_IT_DCRCFAIL: Data block sent/received (CRC check failed) interrupt
  - SDMMC_IT_CTIMEOUT: Command response timeout interrupt
  - SDMMC_IT_DTIMEOUT: Data timeout interrupt
  - SDMMC_IT_TXUNDERR: Transmit FIFO underrun error interrupt
  - SDMMC_IT_RXOVERR: Received FIFO overrun error interrupt
  - SDMMC_IT_CMDREND: Command response received (CRC check passed) interrupt
  - SDMMC_IT_CMDSENT: Command sent (no response required) interrupt
  - SDMMC_IT_DATAEND: Data end (data counter, DATACOUNT, is zero) interrupt
  - SDMMC_IT_DHOLD: Data transfer Hold interrupt
  - SDMMC_IT_DBCKEND: Data block sent/received (CRC check passed) interrupt
  - SDMMC_IT_DABORT: Data transfer aborted by CMD12 interrupt
  - SDMMC_IT_BUSYD0END: End of SDMMC_D0 Busy following a CMD response detected interrupt
  - SDMMC_IT_SDIOIT: SDIO interrupt received interrupt
  - SDMMC_IT_ACKFAIL: Boot Acknowledgment received interrupt
  - SDMMC_IT_ACKTIMEOUT: Boot Acknowledgment timeout interrupt
  - SDMMC_IT_VSWEND: Voltage switch critical timing section completion interrupt
  - SDMMC_IT_CKSTOP: SDMMC_CK stopped in Voltage switch procedure interrupt
  - SDMMC_IT_IDMABTC: IDMA buffer transfer complete interrupt

Return value:

- None

SD Card State enumeration structure

HAL_SD_CARD_READY

Card state is ready

HAL_SD_CARD_IDENTIFICATION

Card is in identification state

HAL_SD_CARD_STANDBY

Card is in standby state

HAL_SD_CARD_TRANSFER

Card is in transfer state

HAL_SD_CARD_SENDING

Card is sending an operation

HAL_SD_CARD_RECEIVING

Card is receiving operation information

HAL_SD_CARD_PROGRAMMING

Card is in programming state

HAL_SD_CARD_DISCONNECTED

Card is disconnected

HAL_SD_CARD_ERROR

Card response Error

SD Handle Structure definition

SD_InitTypeDef

SD_TypeDef

68 HAL SD Extension Driver

68.1 SDEx Firmware driver API description

The following section lists the various functions of the SDEx library.

68.1.1 How to use this driver

The SD Extension HAL driver can be used as follows:

- Configure Buffer0 and Buffer1 start address and Buffer size using HAL_SDEx_ConfigDMAMultiBuffer() function.
- Start Read and Write for multibuffer mode using HAL_SDEx_ReadBlocksDMAMultiBuffer() and HAL_SDEx_WriteBlocksDMAMultiBuffer() functions.

68.1.2 Detailed description of functions

HAL_SDEx_DMALinkedList_ReadBlocks

Function name

HAL_StatusTypeDef HAL_SDEx_DMALinkedList_ReadBlocks (SD_HandleTypeDef * hsd, const SD_DMALinkedListTypeDef * pLinkedList, uint32_t BlockAdd, uint32_t NumberOfBlocks)

Function description

HAL_SDEx_DMALinkedList_WriteBlocks

Function name

HAL_StatusTypeDef HAL_SDEx_DMALinkedList_WriteBlocks (SD_HandleTypeDef * hsd, const SD_DMALinkedListTypeDef * pLinkedList, uint32_t BlockAdd, uint32_t NumberOfBlocks)

Function description

HAL_SDEx_DMALinkedList_BuildNode

Function name

HAL_StatusTypeDef HAL_SDEx_DMALinkedList_BuildNode (SD_DMALinkNodeTypeDef * pNode, SD_DMALinkNodeConfTypeDef * pNodeConf)

Function description

HAL_SDEx_DMALinkedList_InsertNode

Function name

HAL_StatusTypeDef HAL_SDEx_DMALinkedList_InsertNode (SD_DMALinkedListTypeDef * pLinkedList, SD_DMALinkNodeTypeDef * pPrevNode, SD_DMALinkNodeTypeDef * pNewNode)

Function description

HAL_SDEx_DMALinkedList_RemoveNode

Function name

HAL_StatusTypeDef HAL_SDEx_DMALinkedList_RemoveNode (SD_DMALinkedListTypeDef * pLinkedList, SD_DMALinkNodeTypeDef * pNode)

Function description

HAL_SDEx_DMALinkedList_LockNode

Function name

HAL_StatusTypeDef HAL_SDEx_DMALinkedList_LockNode (SD_DMALinkNodeTypeDef * pNode)

Function description

HAL_SDEx_DMALinkedList_UnlockNode

Function name

HAL_StatusTypeDef HAL_SDEx_DMALinkedList_UnlockNode (SD_DMALinkNodeTypeDef * pNode)

Function description

HAL_SDEx_DMALinkedList_EnableCircularMode

Function name

HAL_StatusTypeDef HAL_SDEx_DMALinkedList_EnableCircularMode (SD_DMALinkedListTypeDef * pLinkedList)

Function description

HAL_SDEx_DMALinkedList_DisableCircularMode

Function name

HAL_StatusTypeDef HAL_SDEx_DMALinkedList_DisableCircularMode (SD_DMALinkedListTypeDef * pLinkedList)

Function description

HAL_SDEx_Read_DMALnkLstBufCpltCallback

Function name

void HAL_SDEx_Read_DMALnkLstBufCpltCallback (SD_HandleTypeDef * hsd)

Function description

HAL_SDEx_Write_DMALnkLstBufCpltCallback

Function name

void HAL_SDEx_Write_DMALnkLstBufCpltCallback (SD_HandleTypeDef * hsd)

Function description

68.2 SDEx Firmware driver defines

The following section lists the various define and macros of the module.

68.2.1 SDEx

SDEx

Linked List Wrapper

SD_DMALinkNodeTypeDef

SD_DMALinkNodeConfTypeDef

SD_DMALinkedListTypeDef

HAL_SDEx_DMALinkedList_WriteCpltCallback

HAL_SDEx_DMALinkedList_ReadCpltCallback

69 HAL SMARTCARD Generic Driver

69.1 SMARTCARD Firmware driver registers structures

69.1.1 SMARTCARD_InitTypeDef

SMARTCARD_InitTypeDef is defined in the stm32n6xx_hal_smartcard.h

Data Fields

- ***uint32_t BaudRate***
- ***uint32_t WordLength***
- ***uint32_t StopBits***
- ***uint16_t Parity***
- ***uint16_t Mode***
- ***uint16_t CLKPolarity***
- ***uint16_t CLKPhase***
- ***uint16_t CLKLastBit***
- ***uint16_t OneBitSampling***
- ***uint8_t Prescaler***
- ***uint8_t GuardTime***
- ***uint16_t NACKEnable***
- ***uint32_t TimeOutEnable***
- ***uint32_t TimeOutValue***
- ***uint8_t BlockLength***
- ***uint8_t AutoRetryCount***
- ***uint32_t ClockPrescaler***

Field Documentation

- ***uint32_t SMARTCARD_InitTypeDef::BaudRate***
  Configures the SmartCard communication baud rate. The baud rate register is computed using the following formula: Baud Rate Register = ((usart_ker_ckpres) / ((hsmartcard->Init.BaudRate))) where usart_ker_ckpres is the USART input clock divided by a prescaler
- ***uint32_t SMARTCARD_InitTypeDef::WordLength***
  Specifies the number of data bits transmitted or received in a frame. This parameter ***SMARTCARD_Word_Length*** can only be set to 9 (8 data + 1 parity bits).
- ***uint32_t SMARTCARD_InitTypeDef::StopBits***
  Specifies the number of stop bits. This parameter can be a value of ***SMARTCARD_Stop_Bits***.
- ***uint16_t SMARTCARD_InitTypeDef::Parity***
  Specifies the parity mode. This parameter can be a value of ***SMARTCARD_Parity***
  Note:
  – The parity is enabled by default (PCE is forced to 1). Since the WordLength is forced to 8 bits + parity, M is forced to 1 and the parity bit is the 9th bit.
- ***uint16_t SMARTCARD_InitTypeDef::Mode***
  Specifies whether the Receive or Transmit mode is enabled or disabled. This parameter can be a value of ***SMARTCARD_Mode***
- ***uint16_t SMARTCARD_InitTypeDef::CLKPolarity***
  Specifies the steady state of the serial clock. This parameter can be a value of ***SMARTCARD_Clock_Polarity***
- ***uint16_t SMARTCARD_InitTypeDef::CLKPhase***
  Specifies the clock transition on which the bit capture is made. This parameter can be a value of ***SMARTCARD_Clock_Phase***
- ***uint16_t SMARTCARD_InitTypeDef::CLKLastBit***
  Specifies whether the clock pulse corresponding to the last transmitted data bit (MSB) has to be output on the SCLK pin in synchronous mode. This parameter can be a value of ***SMARTCARD_Last_Bit***

- ***uint16_t SMARTCARD_InitTypeDef::OneBitSampling***
  Specifies whether a single sample or three samples' majority vote is selected. Selecting the single sample method increases the receiver tolerance to clock deviations. This parameter can be a value of ***SMARTCARD_OneBit_Sampling***.
- ***uint8_t SMARTCARD_InitTypeDef::Prescaler***
  Specifies the SmartCard Prescaler. This parameter can be any value from 0x01 to 0x1F. Prescaler value is multiplied by 2 to give the division factor of the source clock frequency
- ***uint8_t SMARTCARD_InitTypeDef::GuardTime***
  Specifies the SmartCard Guard Time applied after stop bits.
- ***uint16_t SMARTCARD_InitTypeDef::NACKEnable***
  Specifies whether the SmartCard NACK transmission is enabled in case of parity error. This parameter can be a value of ***SMARTCARD_NACK_Enable***
- ***uint32_t SMARTCARD_InitTypeDef::TimeOutEnable***
  Specifies whether the receiver timeout is enabled. This parameter can be a value of ***SMARTCARD_Timeout_Enable***
- ***uint32_t SMARTCARD_InitTypeDef::TimeOutValue***
  Specifies the receiver time out value in number of baud blocks: it is used to implement the Character Wait Time (CWT) and Block Wait Time (BWT). It is coded over 24 bits.
- ***uint8_t SMARTCARD_InitTypeDef::BlockLength***
  Specifies the SmartCard Block Length in T=1 Reception mode. This parameter can be any value from 0x0 to 0xFF
- ***uint8_t SMARTCARD_InitTypeDef::AutoRetryCount***
  Specifies the SmartCard auto-retry count (number of retries in receive and transmit mode). When set to 0, retransmission is disabled. Otherwise, its maximum value is 7 (before signalling an error)
- ***uint32_t SMARTCARD_InitTypeDef::ClockPrescaler***
  Specifies the prescaler value used to divide the USART clock source. This parameter can be a value of ***SMARTCARD_ClockPrescaler***.

69.1.2 SMARTCARD_AdvFeatureInitTypeDef

SMARTCARD_AdvFeatureInitTypeDef is defined in the stm32n6xx_hal_smartcard.h

Data Fields

- ***uint32_t AdvFeatureInit***
- ***uint32_t TxPinLevelInvert***
- ***uint32_t RxPinLevelInvert***
- ***uint32_t DataInvert***
- ***uint32_t Swap***
- ***uint32_t OverrunDisable***
- ***uint32_t DMADisableonRxError***
- ***uint32_t MSBFirst***
- ***uint16_t TxCompletionIndication***

Field Documentation

- ***uint32_t SMARTCARD_AdvFeatureInitTypeDef::AdvFeatureInit***
  Specifies which advanced SMARTCARD features is initialized. Several advanced features may be initialized at the same time. This parameter can be a value of ***SMARTCARDEx_Advanced_Features_Initialization_Type***
- ***uint32_t SMARTCARD_AdvFeatureInitTypeDef::TxPinLevelInvert***
  Specifies whether the TX pin active level is inverted. This parameter can be a value of ***SMARTCARD_Tx_Inv***
- ***uint32_t SMARTCARD_AdvFeatureInitTypeDef::RxPinLevelInvert***
  Specifies whether the RX pin active level is inverted. This parameter can be a value of ***SMARTCARD_Rx_Inv***
- ***uint32_t SMARTCARD_AdvFeatureInitTypeDef::DataInvert***
  Specifies whether data are inverted (positive/direct logic vs negative/inverted logic). This parameter can be a value of ***SMARTCARD_Data_Inv***

- ***uint32_t SMARTCARD_AdvFeatureInitTypeDef::Swap***
  Specifies whether TX and RX pins are swapped. This parameter can be a value of ***SMARTCARD_Rx_Tx_Swap***
- ***uint32_t SMARTCARD_AdvFeatureInitTypeDef::OverrunDisable***
  Specifies whether the reception overrun detection is disabled. This parameter can be a value of ***SMARTCARD_Overrun_Disable***
- ***uint32_t SMARTCARD_AdvFeatureInitTypeDef::DMADisableonRxError***
  Specifies whether the DMA is disabled in case of reception error. This parameter can be a value of ***SMARTCARD_DMA_Disable_on_Rx_Error***
- ***uint32_t SMARTCARD_AdvFeatureInitTypeDef::MSBFirst***
  Specifies whether MSB is sent first on UART line. This parameter can be a value of ***SMARTCARD_MSB_First***
- ***uint16_t SMARTCARD_AdvFeatureInitTypeDef::TxCompletionIndication***
  Specifies which transmission completion indication is used: before (when relevant flag is available) or once guard time period has elapsed. This parameter can be a value of ***SMARTCARDEx_Transmission_Completion_Indication***.

69.1.3 __SMARTCARD_HandleTypeDef

__SMARTCARD_HandleTypeDef is defined in the stm32n6xx_hal_smartcard.h

Data Fields

- ***USART_TypeDef * Instance***
- ***SMARTCARD_InitTypeDef Init***
- ***SMARTCARD_AdvFeatureInitTypeDef AdvancedInit***
- ***const uint8_t * pTxBuffPtr***
- ***uint16_t TxXferSize***
- ***__IO uint16_t TxXferCount***
- ***uint8_t * pRxBuffPtr***
- ***uint16_t RxXferSize***
- ***__IO uint16_t RxXferCount***
- ***uint16_t NbRxDataToProcess***
- ***uint16_t NbTxDataToProcess***
- ***uint32_t FifoMode***
- ***void(* RxISR***
- ***void(* TxISR***
- ***DMA_HandleTypeDef * hdmatx***
- ***DMA_HandleTypeDef * hdmarx***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_SMARTCARD_StateTypeDef gState***
- ***__IO HAL_SMARTCARD_StateTypeDef RxState***
- ***__IO uint32_t ErrorCode***
- ***void(* TxCpltCallback***
- ***void(* RxCpltCallback***
- ***void(* ErrorCallback***
- ***void(* AbortCpltCallback***
- ***void(* AbortTransmitCpltCallback***
- ***void(* AbortReceiveCpltCallback***
- ***void(* RxFifoFullCallback***
- ***void(* TxFifoEmptyCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***USART_TypeDef* __SMARTCARD_HandleTypeDef::Instance***
  USART registers base address
- ***SMARTCARD_InitTypeDef __SMARTCARD_HandleTypeDef::Init***
  SmartCard communication parameters
- ***SMARTCARD_AdvFeatureInitTypeDef __SMARTCARD_HandleTypeDef::AdvancedInit***
  SmartCard advanced features initialization parameters
- ***const uint8_t* __SMARTCARD_HandleTypeDef::pTxBuffPtr***
  Pointer to SmartCard Tx transfer Buffer
- ***uint16_t __SMARTCARD_HandleTypeDef::TxXferSize***
  SmartCard Tx Transfer size
- ***__IO uint16_t __SMARTCARD_HandleTypeDef::TxXferCount***
  SmartCard Tx Transfer Counter
- ***uint8_t* __SMARTCARD_HandleTypeDef::pRxBuffPtr***
  Pointer to SmartCard Rx transfer Buffer
- ***uint16_t __SMARTCARD_HandleTypeDef::RxXferSize***
  SmartCard Rx Transfer size
- ***__IO uint16_t __SMARTCARD_HandleTypeDef::RxXferCount***
  SmartCard Rx Transfer Counter
- ***uint16_t __SMARTCARD_HandleTypeDef::NbRxDataToProcess***
  Number of data to process during RX ISR execution
- ***uint16_t __SMARTCARD_HandleTypeDef::NbTxDataToProcess***
  Number of data to process during TX ISR execution
- ***uint32_t __SMARTCARD_HandleTypeDef::FifoMode***
  Specifies if the FIFO mode will be used. This parameter can be a value of ***SMARTCARDEx_FIFO_mode***.
- ***void(* __SMARTCARD_HandleTypeDef::RxISR)(struct __SMARTCARD_HandleTypeDef *huart)***
  Function pointer on Rx IRQ handler
- ***void(* __SMARTCARD_HandleTypeDef::TxISR)(struct __SMARTCARD_HandleTypeDef *huart)***
  Function pointer on Tx IRQ handler
- ***DMA_HandleTypeDef* __SMARTCARD_HandleTypeDef::hdmatx***
  SmartCard Tx DMA Handle parameters
- ***DMA_HandleTypeDef* __SMARTCARD_HandleTypeDef::hdmarx***
  SmartCard Rx DMA Handle parameters
- ***HAL_LockTypeDef __SMARTCARD_HandleTypeDef::Lock***
  Locking object
- ***__IO HAL_SMARTCARD_StateTypeDef __SMARTCARD_HandleTypeDef::gState***
  SmartCard state information related to global Handle management and also related to Tx operations. This parameter can be a value of **HAL_SMARTCARD_StateTypeDef**
- ***__IO HAL_SMARTCARD_StateTypeDef __SMARTCARD_HandleTypeDef::RxState***
  SmartCard state information related to Rx operations. This parameter can be a value of **HAL_SMARTCARD_StateTypeDef**
- ***__IO uint32_t __SMARTCARD_HandleTypeDef::ErrorCode***
  SmartCard Error code
- ***void(* __SMARTCARD_HandleTypeDef::TxCpltCallback)(struct __SMARTCARD_HandleTypeDef *hsmartcard)***
  SMARTCARD Tx Complete Callback
- ***void(* __SMARTCARD_HandleTypeDef::RxCpltCallback)(struct __SMARTCARD_HandleTypeDef *hsmartcard)***
  SMARTCARD Rx Complete Callback
- ***void(* __SMARTCARD_HandleTypeDef::ErrorCallback)(struct __SMARTCARD_HandleTypeDef *hsmartcard)***
  SMARTCARD Error Callback
- ***void(* __SMARTCARD_HandleTypeDef::AbortCpltCallback)(struct __SMARTCARD_HandleTypeDef *hsmartcard)***
  SMARTCARD Abort Complete Callback

- ***void(* __SMARTCARD_HandleTypeDef::AbortTransmitCpltCallback)(struct __SMARTCARD_HandleTypeDef *hsmartcard)***
  SMARTCARD Abort Transmit Complete Callback
- ***void(* __SMARTCARD_HandleTypeDef::AbortReceiveCpltCallback)(struct __SMARTCARD_HandleTypeDef *hsmartcard)***
  SMARTCARD Abort Receive Complete Callback
- ***void(* __SMARTCARD_HandleTypeDef::RxFifoFullCallback)(struct __SMARTCARD_HandleTypeDef *hsmartcard)***
  SMARTCARD Rx Fifo Full Callback
- ***void(* __SMARTCARD_HandleTypeDef::TxFifoEmptyCallback)(struct __SMARTCARD_HandleTypeDef *hsmartcard)***
  SMARTCARD Tx Fifo Empty Callback
- ***void(* __SMARTCARD_HandleTypeDef::MspInitCallback)(struct __SMARTCARD_HandleTypeDef *hsmartcard)***
  SMARTCARD Msp Init callback
- ***void(* __SMARTCARD_HandleTypeDef::MspDeInitCallback)(struct __SMARTCARD_HandleTypeDef *hsmartcard)***
  SMARTCARD Msp DeInit callback

69.2 SMARTCARD Firmware driver API description

The following section lists the various functions of the SMARTCARD library.

69.2.1 How to use this driver

The SMARTCARD HAL driver can be used as follows:

1. Declare a SMARTCARD_HandleTypeDef handle structure (eg. SMARTCARD_HandleTypeDef hsmartcard).
2. Associate a USART to the SMARTCARD handle hsmartcard.
3. Initialize the SMARTCARD low level resources by implementing the HAL_SMARTCARD_MspInit() API:
   - Enable the USARTx interface clock.
   - USART pins configuration:
     - Enable the clock for the USART GPIOs.
     - Configure the USART pins (TX as alternate function pull-up, RX as alternate function Input).
   - NVIC configuration if you need to use interrupt process (HAL_SMARTCARD_Transmit_IT() and HAL_SMARTCARD_Receive_IT() APIs):
     - Configure the USARTx interrupt priority.
     - Enable the NVIC USART IRQ handle.
   - DMA Configuration if you need to use DMA process (HAL_SMARTCARD_Transmit_DMA() and HAL_SMARTCARD_Receive_DMA() APIs):
     - Declare a DMA handle structure for the Tx/Rx channel.
     - Enable the DMAx interface clock.
     - Configure the declared DMA handle structure with the required Tx/Rx parameters.
     - Configure the DMA Tx/Rx channel.
     - Associate the initialized DMA handle to the SMARTCARD DMA Tx/Rx handle.
     - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx/Rx channel.
4. Program the Baud Rate, Parity, Mode(Receiver/Transmitter), clock enabling/disabling and accordingly, the clock parameters (parity, phase, last bit), prescaler value, guard time and NACK on transmission error enabling or disabling in the hsmartcard handle Init structure.
5. If required, program SMARTCARD advanced features (TX/RX pins swap, TimeOut, auto-retry counter,...) in the hsmartcard handle AdvancedInit structure.
6. Initialize the SMARTCARD registers by calling the HAL_SMARTCARD_Init() API:
   - This API configures also the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customized HAL_SMARTCARD_MspInit() API.

Note: The specific SMARTCARD interrupts (Transmission complete interrupt, RXNE interrupt and Error Interrupts) will be managed using the macros __HAL_SMARTCARD_ENABLE_IT() and __HAL_SMARTCARD_DISABLE_IT() inside the transmit and receive process.

Three operation modes are available within this driver :

Polling mode IO operation

- Send an amount of data in blocking mode using HAL_SMARTCARD_Transmit()
- Receive an amount of data in blocking mode using HAL_SMARTCARD_Receive()

Interrupt mode IO operation

- Send an amount of data in non-blocking mode using HAL_SMARTCARD_Transmit_IT()
- At transmission end of transfer HAL_SMARTCARD_TxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SMARTCARD_TxCpltCallback()
- Receive an amount of data in non-blocking mode using HAL_SMARTCARD_Receive_IT()
- At reception end of transfer HAL_SMARTCARD_RxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SMARTCARD_RxCpltCallback()
- In case of transfer Error, HAL_SMARTCARD_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_SMARTCARD_ErrorCallback()

DMA mode IO operation

- Send an amount of data in non-blocking mode (DMA) using HAL_SMARTCARD_Transmit_DMA()
- At transmission end of transfer HAL_SMARTCARD_TxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SMARTCARD_TxCpltCallback()
- Receive an amount of data in non-blocking mode (DMA) using HAL_SMARTCARD_Receive_DMA()
- At reception end of transfer HAL_SMARTCARD_RxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SMARTCARD_RxCpltCallback()
- In case of transfer Error, HAL_SMARTCARD_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_SMARTCARD_ErrorCallback()

SMARTCARD HAL driver macros list

Below the list of most used macros in SMARTCARD HAL driver.

- __HAL_SMARTCARD_GET_FLAG : Check whether or not the specified SMARTCARD flag is set
- __HAL_SMARTCARD_CLEAR_FLAG : Clear the specified SMARTCARD pending flag
- __HAL_SMARTCARD_ENABLE_IT: Enable the specified SMARTCARD interrupt
- __HAL_SMARTCARD_DISABLE_IT: Disable the specified SMARTCARD interrupt
- __HAL_SMARTCARD_GET_IT_SOURCE: Check whether or not the specified SMARTCARD interrupt is enabled

Note: You can refer to the SMARTCARD HAL driver header file for more useful macros

69.2.2 Callback registration

The compilation define USE_HAL_SMARTCARD_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks.

Use Function HAL_SMARTCARD_RegisterCallback() to register a user callback. Function HAL_SMARTCARD_RegisterCallback() allows to register following callbacks:

- TxCpltCallback : Tx Complete Callback.
- RxCpltCallback : Rx Complete Callback.
- ErrorCallback : Error Callback.
- AbortCpltCallback : Abort Complete Callback.
- AbortTransmitCpltCallback : Abort Transmit Complete Callback.
- AbortReceiveCpltCallback : Abort Receive Complete Callback.
- RxFifoFullCallback : Rx Fifo Full Callback.

- TxFifoEmptyCallback : Tx Fifo Empty Callback.
- MspInitCallback : SMARTCARD MspInit.
- MspDeInitCallback : SMARTCARD MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

Use function HAL_SMARTCARD_UnRegisterCallback() to reset a callback to the default weak function. HAL_SMARTCARD_UnRegisterCallback() takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks:

- TxCpltCallback : Tx Complete Callback.
- RxCpltCallback : Rx Complete Callback.
- ErrorCallback : Error Callback.
- AbortCpltCallback : Abort Complete Callback.
- AbortTransmitCpltCallback : Abort Transmit Complete Callback.
- AbortReceiveCpltCallback : Abort Receive Complete Callback.
- RxFifoFullCallback : Rx Fifo Full Callback.
- TxFifoEmptyCallback : Tx Fifo Empty Callback.
- MspInitCallback : SMARTCARD MspInit.
- MspDeInitCallback : SMARTCARD MspDeInit.

By default, after the HAL_SMARTCARD_Init() and when the state is HAL_SMARTCARD_STATE_RESET all callbacks are set to the corresponding weak functions: examples HAL_SMARTCARD_TxCpltCallback(), HAL_SMARTCARD_RxCpltCallback(). Exception done for MspInit and MspDeInit functions that are respectively reset to the legacy weak functions in the HAL_SMARTCARD_Init() and HAL_SMARTCARD_DeInit() only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the HAL_SMARTCARD_Init() and HAL_SMARTCARD_DeInit() keep and use the user MspInit/MspDeInit callbacks (registered beforehand).

Callbacks can be registered/unregistered in HAL_SMARTCARD_STATE_READY state only. Exception done MspInit/MspDeInit that can be registered/unregistered in HAL_SMARTCARD_STATE_READY or HAL_SMARTCARD_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_SMARTCARD_RegisterCallback() before calling HAL_SMARTCARD_DeInit() or HAL_SMARTCARD_Init() function.

When The compilation define USE_HAL_SMARTCARD_REGISTER_CALLBACKS is set to 0 or not defined, the callback registration feature is not available and weak callbacks are used.

69.2.3 Detailed description of functions

HAL_SMARTCARD_Init

Function name

HAL_StatusTypeDef HAL_SMARTCARD_Init (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

HAL_SMARTCARD_DeInit

Function name

HAL_StatusTypeDef HAL_SMARTCARD_DeInit (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

HAL_SMARTCARD_MspInit

Function name

void HAL_SMARTCARD_MspInit (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

HAL_SMARTCARD_MspDeInit

Function name

void HAL_SMARTCARD_MspDeInit (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

HAL_SMARTCARD_RegisterCallback

Function name

HAL_StatusTypeDef HAL_SMARTCARD_RegisterCallback (SMARTCARD_HandleTypeDef * hsmartcard, HAL_SMARTCARD_CallbackIDTypeDef CallbackID, pSMARTCARD_CallbackTypeDef pCallback)

Function description

HAL_SMARTCARD_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_SMARTCARD_UnRegisterCallback (SMARTCARD_HandleTypeDef * hsmartcard, HAL_SMARTCARD_CallbackIDTypeDef CallbackID)

Function description

HAL_SMARTCARD_Transmit

Function name

HAL_StatusTypeDef HAL_SMARTCARD_Transmit (SMARTCARD_HandleTypeDef * hsmartcard, const uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

HAL_SMARTCARD_Receive

Function name

HAL_StatusTypeDef HAL_SMARTCARD_Receive (SMARTCARD_HandleTypeDef * hsmartcard, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

HAL_SMARTCARD_Transmit_IT

Function name

HAL_StatusTypeDef HAL_SMARTCARD_Transmit_IT (SMARTCARD_HandleTypeDef * hsmartcard, const uint8_t * pData, uint16_t Size)

Function description

HAL_SMARTCARD_Receive_IT

Function name

HAL_StatusTypeDef HAL_SMARTCARD_Receive_IT (SMARTCARD_HandleTypeDef * hsmartcard, uint8_t * pData, uint16_t Size)

Function description

HAL_SMARTCARD_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_SMARTCARD_Transmit_DMA (SMARTCARD_HandleTypeDef * hsmartcard, const uint8_t * pData, uint16_t Size)

Function description

HAL_SMARTCARD_Receive_DMA

Function name

HAL_StatusTypeDef HAL_SMARTCARD_Receive_DMA (SMARTCARD_HandleTypeDef * hsmartcard, uint8_t * pData, uint16_t Size)

Function description

HAL_SMARTCARD_Abort

Function name

HAL_StatusTypeDef HAL_SMARTCARD_Abort (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

HAL_SMARTCARD_AbortTransmit

Function name

HAL_StatusTypeDef HAL_SMARTCARD_AbortTransmit (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

HAL_SMARTCARD_AbortReceive

Function name

HAL_StatusTypeDef HAL_SMARTCARD_AbortReceive (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

HAL_SMARTCARD_Abort_IT

Function name

HAL_StatusTypeDef HAL_SMARTCARD_Abort_IT (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

HAL_SMARTCARD_AbortTransmit_IT

Function name

HAL_StatusTypeDef HAL_SMARTCARD_AbortTransmit_IT (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

HAL_SMARTCARD_AbortReceive_IT

Function name

HAL_StatusTypeDef HAL_SMARTCARD_AbortReceive_IT (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

HAL_SMARTCARD_IRQHandler

Function name

void HAL_SMARTCARD_IRQHandler (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

HAL_SMARTCARD_TxCpltCallback

Function name

void HAL_SMARTCARD_TxCpltCallback (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

HAL_SMARTCARD_RxCpltCallback

Function name

void HAL_SMARTCARD_RxCpltCallback (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

HAL_SMARTCARD_ErrorCallback

Function name

void HAL_SMARTCARD_ErrorCallback (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

HAL_SMARTCARD_AbortCpltCallback

Function name

void HAL_SMARTCARD_AbortCpltCallback (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

HAL_SMARTCARD_AbortTransmitCpltCallback

Function name

void HAL_SMARTCARD_AbortTransmitCpltCallback (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

HAL_SMARTCARD_AbortReceiveCpltCallback

Function name

void HAL_SMARTCARD_AbortReceiveCpltCallback (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

HAL_SMARTCARD_GetState

Function name

HAL_SMARTCARD_StateTypeDef HAL_SMARTCARD_GetState (const SMARTCARD_HandleTypeDef * hsmartcard)

Function description

HAL_SMARTCARD_GetError

Function name

uint32_t HAL_SMARTCARD_GetError (const SMARTCARD_HandleTypeDef * hsmartcard)

Function description

69.3 SMARTCARD Firmware driver defines

The following section lists the various define and macros of the module.

69.3.1 SMARTCARD

SMARTCARD

SMARTCARD Clock Prescaler

SMARTCARD_PRESCALER_DIV1

fclk_pres = fclk

SMARTCARD_PRESCALER_DIV2

fclk_pres = fclk/2

SMARTCARD_PRESCALER_DIV4

fclk_pres = fclk/4

SMARTCARD_PRESCALER_DIV6

fclk_pres = fclk/6

SMARTCARD_PRESCALER_DIV8

fclk_pres = fclk/8

SMARTCARD_PRESCALER_DIV10

fclk_pres = fclk/10

SMARTCARD_PRESCALER_DIV12

fclk_pres = fclk/12

SMARTCARD_PRESCALER_DIV16

fclk_pres = fclk/16

SMARTCARD_PRESCALER_DIV32

fclk_pres = fclk/32

SMARTCARD_PRESCALER_DIV64

fclk_pres = fclk/64

SMARTCARD_PRESCALER_DIV128

fclk_pres = fclk/128

SMARTCARD_PRESCALER_DIV256

fclk_pres = fclk/256

SMARTCARD Clock Phase

SMARTCARD_PHASE_1EDGE

SMARTCARD frame phase on first clock transition

SMARTCARD_PHASE_2EDGE

SMARTCARD frame phase on second clock transition

SMARTCARD Clock Polarity

SMARTCARD_POLARITY_LOW

SMARTCARD frame low polarity

SMARTCARD_POLARITY_HIGH

SMARTCARD frame high polarity

SMARTCARD advanced feature Binary Data inversion

SMARTCARD_ADVFEATURE_DATAINV_DISABLE

Binary data inversion disable

SMARTCARD_ADVFEATURE_DATAINV_ENABLE

Binary data inversion enable

SMARTCARD advanced feature DMA Disable on Rx Error

SMARTCARD_ADVFEATURE_DMA_ENABLEONRXERROR

DMA enable on Reception Error

SMARTCARD_ADVFEATURE_DMA_DISABLEONRXERROR

DMA disable on Reception Error

SMARTCARD Error Code Definition

HAL_SMARTCARD_ERROR_NONE

No error

HAL_SMARTCARD_ERROR_PE

Parity error

HAL_SMARTCARD_ERROR_NE

Noise error

HAL_SMARTCARD_ERROR_FE

frame error

HAL_SMARTCARD_ERROR_ORE

Overrun error

HAL_SMARTCARD_ERROR_DMA

DMA transfer error

HAL_SMARTCARD_ERROR_RTO

Receiver TimeOut error

HAL_SMARTCARD_ERROR_INVALID_CALLBACK

Invalid Callback error

SMARTCARD Exported Macros

__HAL_SMARTCARD_RESET_HANDLE_STATE

Description:

- Reset SMARTCARD handle states.

Parameters:

- __HANDLE__: SMARTCARD handle.

Return value:

- None

__HAL_SMARTCARD_FLUSH_DRREGISTER

Description:

- Flush the Smartcard Data registers.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.

Return value:

- None

__HAL_SMARTCARD_CLEAR_FLAG

Description:

- Clear the specified SMARTCARD pending flag.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.
- __FLAG__: specifies the flag to check. This parameter can be any combination of the following values:
  - SMARTCARD_CLEAR_PEF Parity error clear flag
  - SMARTCARD_CLEAR_FEF Framing error clear flag
  - SMARTCARD_CLEAR_NEF Noise detected clear flag
  - SMARTCARD_CLEAR_OREF OverRun error clear flag
  - SMARTCARD_CLEAR_IDLEF Idle line detected clear flag
  - SMARTCARD_CLEAR_TCF Transmission complete clear flag
  - SMARTCARD_CLEAR_TCBGTF Transmission complete before guard time clear flag
  - SMARTCARD_CLEAR_RTOF Receiver timeout clear flag
  - SMARTCARD_CLEAR_EOBF End of block clear flag
  - SMARTCARD_CLEAR_TXFECF TXFIFO empty Clear flag

Return value:

- None

__HAL_SMARTCARD_CLEAR_PEFLAG

Description:

- Clear the SMARTCARD PE pending flag.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.

Return value:

- None

__HAL_SMARTCARD_CLEAR_FEFLAG

Description:

- Clear the SMARTCARD FE pending flag.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.

Return value:

- None

__HAL_SMARTCARD_CLEAR_NEFLAG

Description:

- Clear the SMARTCARD NE pending flag.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.

Return value:

- None

__HAL_SMARTCARD_CLEAR_OREFLAG

Description:

- Clear the SMARTCARD ORE pending flag.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.

Return value:

- None

__HAL_SMARTCARD_CLEAR_IDLEFLAG

Description:

- Clear the SMARTCARD IDLE pending flag.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.

Return value:

- None

__HAL_SMARTCARD_GET_FLAG

Description:

- Check whether the specified Smartcard flag is set or not.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - SMARTCARD_FLAG_TCBGT Transmission complete before guard time flag (when flag available)
  - SMARTCARD_FLAG_REACK Receive enable acknowledge flag
  - SMARTCARD_FLAG_TEACK Transmit enable acknowledge flag
  - SMARTCARD_FLAG_BUSY Busy flag
  - SMARTCARD_FLAG_EOBF End of block flag
  - SMARTCARD_FLAG_RTOF Receiver timeout flag
  - SMARTCARD_FLAG_TXE Transmit data register empty flag
  - SMARTCARD_FLAG_TC Transmission complete flag
  - SMARTCARD_FLAG_RXNE Receive data register not empty flag
  - SMARTCARD_FLAG_IDLE Idle line detection flag
  - SMARTCARD_FLAG_ORE Overrun error flag
  - SMARTCARD_FLAG_NE Noise error flag
  - SMARTCARD_FLAG_FE Framing error flag
  - SMARTCARD_FLAG_PE Parity error flag
  - SMARTCARD_FLAG_TXFNF TXFIFO not full flag
  - SMARTCARD_FLAG_RXFNE RXFIFO not empty flag
  - SMARTCARD_FLAG_TXFE TXFIFO Empty flag
  - SMARTCARD_FLAG_RXFF RXFIFO Full flag
  - SMARTCARD_FLAG_RXFT SMARTCARD RXFIFO threshold flag
  - SMARTCARD_FLAG_TXFT SMARTCARD TXFIFO threshold flag

Return value:

- The: new state of __FLAG__ (TRUE or FALSE).

__HAL_SMARTCARD_ENABLE_IT

Description:

- Enable the specified SmartCard interrupt.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.
- __INTERRUPT__: specifies the SMARTCARD interrupt to enable. This parameter can be one of the following values:
  - SMARTCARD_IT_EOB End of block interrupt
  - SMARTCARD_IT_RTO Receive timeout interrupt
  - SMARTCARD_IT_TXE Transmit data register empty interrupt
  - SMARTCARD_IT_TC Transmission complete interrupt
  - SMARTCARD_IT_TCBGT Transmission complete before guard time interrupt (when interruption available)
  - SMARTCARD_IT_RXNE Receive data register not empty interrupt
  - SMARTCARD_IT_IDLE Idle line detection interrupt
  - SMARTCARD_IT_PE Parity error interrupt
  - SMARTCARD_IT_ERR Error interrupt(frame error, noise error, overrun error)
  - SMARTCARD_IT_TXFNF TX FIFO not full interruption
  - SMARTCARD_IT_RXFNE RXFIFO not empty interruption
  - SMARTCARD_IT_RXFF RXFIFO full interruption
  - SMARTCARD_IT_TXFE TXFIFO empty interruption
  - SMARTCARD_IT_RXFT RXFIFO threshold reached interruption
  - SMARTCARD_IT_TXFT TXFIFO threshold reached interruption

Return value:

- None

__HAL_SMARTCARD_DISABLE_IT

Description:

- Disable the specified SmartCard interrupt.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.
- __INTERRUPT__: specifies the SMARTCARD interrupt to disable. This parameter can be one of the following values:
  - SMARTCARD_IT_EOB End of block interrupt
  - SMARTCARD_IT_RTO Receive timeout interrupt
  - SMARTCARD_IT_TXE Transmit data register empty interrupt
  - SMARTCARD_IT_TC Transmission complete interrupt
  - SMARTCARD_IT_TCBGT Transmission complete before guard time interrupt (when interruption available)
  - SMARTCARD_IT_RXNE Receive data register not empty interrupt
  - SMARTCARD_IT_IDLE Idle line detection interrupt
  - SMARTCARD_IT_PE Parity error interrupt
  - SMARTCARD_IT_ERR Error interrupt(frame error, noise error, overrun error)
  - SMARTCARD_IT_TXFNF TX FIFO not full interruption
  - SMARTCARD_IT_RXFNE RXFIFO not empty interruption
  - SMARTCARD_IT_RXFF RXFIFO full interruption
  - SMARTCARD_IT_TXFE TXFIFO empty interruption
  - SMARTCARD_IT_RXFT RXFIFO threshold reached interruption
  - SMARTCARD_IT_TXFT TXFIFO threshold reached interruption

Return value:

- None

__HAL_SMARTCARD_GET_IT

Description:

- Check whether the specified SmartCard interrupt has occurred or not.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.
- __INTERRUPT__: specifies the SMARTCARD interrupt to check. This parameter can be one of the following values:
  - SMARTCARD_IT_EOB End of block interrupt
  - SMARTCARD_IT_RTO Receive timeout interrupt
  - SMARTCARD_IT_TXE Transmit data register empty interrupt
  - SMARTCARD_IT_TC Transmission complete interrupt
  - SMARTCARD_IT_TCBGT Transmission complete before guard time interrupt (when interruption available)
  - SMARTCARD_IT_RXNE Receive data register not empty interrupt
  - SMARTCARD_IT_IDLE Idle line detection interrupt
  - SMARTCARD_IT_PE Parity error interrupt
  - SMARTCARD_IT_ERR Error interrupt(frame error, noise error, overrun error)
  - SMARTCARD_IT_TXFNF TX FIFO not full interruption
  - SMARTCARD_IT_RXFNE RXFIFO not empty interruption
  - SMARTCARD_IT_RXFF RXFIFO full interruption
  - SMARTCARD_IT_TXFE TXFIFO empty interruption
  - SMARTCARD_IT_RXFT RXFIFO threshold reached interruption
  - SMARTCARD_IT_TXFT TXFIFO threshold reached interruption

Return value:

- The: new state of __INTERRUPT__ (SET or RESET).

__HAL_SMARTCARD_GET_IT_SOURCE

Description:

- Check whether the specified SmartCard interrupt source is enabled or not.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.
- __INTERRUPT__: specifies the SMARTCARD interrupt source to check. This parameter can be one of the following values:
  - SMARTCARD_IT_EOB End of block interrupt
  - SMARTCARD_IT_RTO Receive timeout interrupt
  - SMARTCARD_IT_TXE Transmit data register empty interrupt
  - SMARTCARD_IT_TC Transmission complete interrupt
  - SMARTCARD_IT_TCBGT Transmission complete before guard time interrupt (when interruption available)
  - SMARTCARD_IT_RXNE Receive data register not empty interrupt
  - SMARTCARD_IT_IDLE Idle line detection interrupt
  - SMARTCARD_IT_PE Parity error interrupt
  - SMARTCARD_IT_ERR Error interrupt(frame error, noise error, overrun error)
  - SMARTCARD_IT_TXFNF TX FIFO not full interruption
  - SMARTCARD_IT_RXFNE RXFIFO not empty interruption
  - SMARTCARD_IT_RXFF RXFIFO full interruption
  - SMARTCARD_IT_TXFE TXFIFO empty interruption
  - SMARTCARD_IT_RXFT RXFIFO threshold reached interruption
  - SMARTCARD_IT_TXFT TXFIFO threshold reached interruption

Return value:

- The: new state of __INTERRUPT__ (SET or RESET).

__HAL_SMARTCARD_CLEAR_IT

Description:

- Clear the specified SMARTCARD ISR flag, in setting the proper ICR register flag.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.
- __IT_CLEAR__: specifies the interrupt clear register flag that needs to be set to clear the corresponding interrupt. This parameter can be one of the following values:
  - SMARTCARD_CLEAR_PEF Parity error clear flag
  - SMARTCARD_CLEAR_FEF Framing error clear flag
  - SMARTCARD_CLEAR_NEF Noise detected clear flag
  - SMARTCARD_CLEAR_OREF OverRun error clear flag
  - SMARTCARD_CLEAR_IDLEF Idle line detection clear flag
  - SMARTCARD_CLEAR_TXFECF TXFIFO empty Clear Flag
  - SMARTCARD_CLEAR_TCF Transmission complete clear flag
  - SMARTCARD_CLEAR_TCBGTF Transmission complete before guard time clear flag (when flag available)
  - SMARTCARD_CLEAR_RTOF Receiver timeout clear flag
  - SMARTCARD_CLEAR_EOBF End of block clear flag

Return value:

- None

__HAL_SMARTCARD_SEND_REQ

Description:

- Set a specific SMARTCARD request flag.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.
- __REQ__: specifies the request flag to set This parameter can be one of the following values:
  - SMARTCARD_RXDATA_FLUSH_REQUEST Receive data flush Request
  - SMARTCARD_TXDATA_FLUSH_REQUEST Transmit data flush Request

Return value:

- None

__HAL_SMARTCARD_ONE_BIT_SAMPLE_ENABLE

Description:

- Enable the SMARTCARD one bit sample method.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.

Return value:

- None

__HAL_SMARTCARD_ONE_BIT_SAMPLE_DISABLE

Description:

- Disable the SMARTCARD one bit sample method.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.

Return value:

- None

__HAL_SMARTCARD_ENABLE

Description:

- Enable the USART associated to the SMARTCARD Handle.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.

Return value:

- None

__HAL_SMARTCARD_DISABLE

Description:

- Disable the USART associated to the SMARTCARD Handle.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.

Return value:

- None

SMARTCARD interruptions flags mask

SMARTCARD_IT_MASK

SMARTCARD interruptions flags mask

SMARTCARD_CR_MASK

SMARTCARD control register mask

SMARTCARD_CR_POS

SMARTCARD control register position

SMARTCARD_ISR_MASK

SMARTCARD ISR register mask

SMARTCARD_ISR_POS

SMARTCARD ISR register position

SMARTCARD Last Bit

SMARTCARD_LASTBIT_DISABLE

SMARTCARD frame last data bit clock pulse not output to SCLK pin

SMARTCARD_LASTBIT_ENABLE

SMARTCARD frame last data bit clock pulse output to SCLK pin

SMARTCARD Transfer Mode

SMARTCARD_MODE_RX

SMARTCARD RX mode

SMARTCARD_MODE_TX

SMARTCARD TX mode

SMARTCARD_MODE_TX_RX

SMARTCARD RX and TX mode

SMARTCARD advanced feature MSB first

SMARTCARD_ADVFEATURE_MSBFIRST_DISABLE

Most significant bit sent/received first disable

SMARTCARD_ADVFEATURE_MSBFIRST_ENABLE

Most significant bit sent/received first enable

SMARTCARD NACK Enable

SMARTCARD_NACK_DISABLE

SMARTCARD NACK transmission disabled

SMARTCARD_NACK_ENABLE

SMARTCARD NACK transmission enabled

SMARTCARD One Bit Sampling Method

SMARTCARD_ONE_BIT_SAMPLE_DISABLE

SMARTCARD frame one-bit sample disabled

SMARTCARD_ONE_BIT_SAMPLE_ENABLE

SMARTCARD frame one-bit sample enabled

SMARTCARD advanced feature Overrun Disable

SMARTCARD_ADVFEATURE_OVERRUN_ENABLE

RX overrun enable

SMARTCARD_ADVFEATURE_OVERRUN_DISABLE

RX overrun disable

SMARTCARD Parity

SMARTCARD_PARITY_EVEN

SMARTCARD frame even parity

SMARTCARD_PARITY_ODD

SMARTCARD frame odd parity

SMARTCARD Request Parameters

SMARTCARD_RXDATA_FLUSH_REQUEST

Receive data flush request

SMARTCARD_TXDATA_FLUSH_REQUEST

Transmit data flush request

SMARTCARD advanced feature RX pin active level inversion

SMARTCARD_ADVFEATURE_RXINV_DISABLE

RX pin active level inversion disable

SMARTCARD_ADVFEATURE_RXINV_ENABLE

RX pin active level inversion enable

SMARTCARD advanced feature RX TX pins swap

SMARTCARD_ADVFEATURE_SWAP_DISABLE

TX/RX pins swap disable

SMARTCARD_ADVFEATURE_SWAP_ENABLE

TX/RX pins swap enable

SMARTCARD State Code Definition

HAL_SMARTCARD_STATE_RESET

Peripheral is not initialized. Value is allowed for gState and RxState

HAL_SMARTCARD_STATE_READY

Peripheral Initialized and ready for use. Value is allowed for gState and RxState

HAL_SMARTCARD_STATE_BUSY

an internal process is ongoing Value is allowed for gState only

HAL_SMARTCARD_STATE_BUSY_TX

Data Transmission process is ongoing Value is allowed for gState only

HAL_SMARTCARD_STATE_BUSY_RX

Data Reception process is ongoing Value is allowed for RxState only

HAL_SMARTCARD_STATE_BUSY_TX_RX

Data Transmission and Reception process is ongoing Not to be used for neither gState nor RxState. Value is result of combination (Or) between gState and RxState values

HAL_SMARTCARD_STATE_TIMEOUT

Timeout state Value is allowed for gState only

HAL_SMARTCARD_STATE_ERROR

Error Value is allowed for gState only

SMARTCARD Number of Stop Bits

SMARTCARD_STOPBITS_0_5

SMARTCARD frame with 0.5 stop bit

SMARTCARD_STOPBITS_1_5

SMARTCARD frame with 1.5 stop bits

SMARTCARD Timeout Enable

SMARTCARD_TIMEOUT_DISABLE

SMARTCARD receiver timeout disabled

SMARTCARD_TIMEOUT_ENABLE

SMARTCARD receiver timeout enabled

SMARTCARD advanced feature TX pin active level inversion

SMARTCARD_ADVFEATURE_TXINV_DISABLE

TX pin active level inversion disable

SMARTCARD_ADVFEATURE_TXINV_ENABLE

TX pin active level inversion enable

SMARTCARD Word Length

SMARTCARD_WORDLENGTH_9B

SMARTCARD frame length

70 HAL SMARTCARD Extension Driver

70.1 SMARTCARDEx Firmware driver API description

The following section lists the various functions of the SMARTCARDEx library.

70.1.1 SMARTCARD peripheral extended features

The Extended SMARTCARD HAL driver can be used as follows:

1. After having configured the SMARTCARD basic features with HAL_SMARTCARD_Init(), then program SMARTCARD advanced features if required (TX/RX pins swap, TimeOut, auto-retry counter,...) in the hsmartcard AdvancedInit structure.
2. FIFO mode enabling/disabling and RX/TX FIFO threshold programming.

Note: When SMARTCARD operates in FIFO mode, FIFO mode must be enabled prior starting RX/TX transfers. Also RX/TX FIFO thresholds must be configured prior starting RX/TX transfers.

70.1.2 Detailed description of functions

HAL_SMARTCARDEx_BlockLength_Config

Function name

void HAL_SMARTCARDEx_BlockLength_Config (SMARTCARD_HandleTypeDef * hsmartcard, uint8_t BlockLength)

Function description

HAL_SMARTCARDEx_TimeOut_Config

Function name

void HAL_SMARTCARDEx_TimeOut_Config (SMARTCARD_HandleTypeDef * hsmartcard, uint32_t TimeOutValue)

Function description

HAL_SMARTCARDEx_EnableReceiverTimeOut

Function name

HAL_StatusTypeDef HAL_SMARTCARDEx_EnableReceiverTimeOut (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

HAL_SMARTCARDEx_DisableReceiverTimeOut

Function name

HAL_StatusTypeDef HAL_SMARTCARDEx_DisableReceiverTimeOut (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

HAL_SMARTCARDEx_RxFifoFullCallback

Function name

void HAL_SMARTCARDEx_RxFifoFullCallback (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

HAL_SMARTCARDEx_TxFifoEmptyCallback

Function name

void HAL_SMARTCARDEx_TxFifoEmptyCallback (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

HAL_SMARTCARDEx_EnableFifoMode

Function name

HAL_StatusTypeDef HAL_SMARTCARDEx_EnableFifoMode (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

HAL_SMARTCARDEx_DisableFifoMode

Function name

HAL_StatusTypeDef HAL_SMARTCARDEx_DisableFifoMode (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

HAL_SMARTCARDEx_SetTxFifoThreshold

Function name

HAL_StatusTypeDef HAL_SMARTCARDEx_SetTxFifoThreshold (SMARTCARD_HandleTypeDef * hsmartcard, uint32_t Threshold)

Function description

HAL_SMARTCARDEx_SetRxFifoThreshold

Function name

HAL_StatusTypeDef HAL_SMARTCARDEx_SetRxFifoThreshold (SMARTCARD_HandleTypeDef * hsmartcard, uint32_t Threshold)

Function description

70.2 SMARTCARDEx Firmware driver defines

The following section lists the various define and macros of the module.

70.2.1 SMARTCARDEx

SMARTCARDEx

SMARTCARD advanced feature initialization type

SMARTCARD_ADVFEATURE_NO_INIT

No advanced feature initialization

SMARTCARD_ADVFEATURE_TXINVERT_INIT

TX pin active level inversion

SMARTCARD_ADVFEATURE_RXINVERT_INIT

RX pin active level inversion

SMARTCARD_ADVFEATURE_DATAINVERT_INIT

Binary data inversion

SMARTCARD_ADVFEATURE_SWAP_INIT

TX/RX pins swap

SMARTCARD_ADVFEATURE_RXOVERRUNDISABLE_INIT

RX overrun disable

SMARTCARD_ADVFEATURE_DMADISABLEONERROR_INIT

DMA disable on Reception Error

SMARTCARD_ADVFEATURE_MSBFIRST_INIT

Most significant bit sent/received first

SMARTCARD_ADVFEATURE_TXCOMPLETION

TX completion indication before of after guard time

SMARTCARD FIFO mode

SMARTCARD_FIFOMODE_DISABLE

FIFO mode disable

SMARTCARD_FIFOMODE_ENABLE

FIFO mode enable

SMARTCARD Flags

SMARTCARD_FLAG_TCBGT

SMARTCARD transmission complete before guard time completion

SMARTCARD_FLAG_REACK

SMARTCARD receive enable acknowledge flag

SMARTCARD_FLAG_TEACK

SMARTCARD transmit enable acknowledge flag

SMARTCARD_FLAG_BUSY

SMARTCARD busy flag

SMARTCARD_FLAG_EOBF

SMARTCARD end of block flag

SMARTCARD_FLAG_RTOF

SMARTCARD receiver timeout flag

SMARTCARD_FLAG_TXE

SMARTCARD transmit data register empty

SMARTCARD_FLAG_TXFNF

SMARTCARD TXFIFO not full

SMARTCARD_FLAG_TC

SMARTCARD transmission complete

SMARTCARD_FLAG_RXNE

SMARTCARD read data register not empty

SMARTCARD_FLAG_RXFNE

SMARTCARD RXFIFO not empty

SMARTCARD_FLAG_IDLE

SMARTCARD idle line detection

SMARTCARD_FLAG_ORE

SMARTCARD overrun error

SMARTCARD_FLAG_NE

SMARTCARD noise error

SMARTCARD_FLAG_FE

SMARTCARD frame error

SMARTCARD_FLAG_PE

SMARTCARD parity error

SMARTCARD_FLAG_TXFE

SMARTCARD TXFIFO Empty flag

SMARTCARD_FLAG_RXFF

SMARTCARD RXFIFO Full flag

SMARTCARD_FLAG_RXFT

SMARTCARD RXFIFO threshold flag

SMARTCARD_FLAG_TXFT

SMARTCARD TXFIFO threshold flag

SMARTCARD Interrupts Definition

SMARTCARD_IT_PE

SMARTCARD parity error interruption

SMARTCARD_IT_TXE

SMARTCARD transmit data register empty interruption

SMARTCARD_IT_TXFNF

SMARTCARD TX FIFO not full interruption

SMARTCARD_IT_TC

SMARTCARD transmission complete interruption

SMARTCARD_IT_RXNE

SMARTCARD read data register not empty interruption

SMARTCARD_IT_RXFNE

SMARTCARD RXFIFO not empty interruption

SMARTCARD_IT_IDLE

SMARTCARD idle line detection interruption

SMARTCARD_IT_ERR

SMARTCARD error interruption

SMARTCARD_IT_ORE

SMARTCARD overrun error interruption

SMARTCARD_IT_NE

SMARTCARD noise error interruption

SMARTCARD_IT_FE

SMARTCARD frame error interruption

SMARTCARD_IT_EOB

SMARTCARD end of block interruption

SMARTCARD_IT_RTO

SMARTCARD receiver timeout interruption

SMARTCARD_IT_TCBGT

SMARTCARD transmission complete before guard time completion interruption

SMARTCARD_IT_RXFF

SMARTCARD RXFIFO full interruption

SMARTCARD_IT_TXFE

SMARTCARD TXFIFO empty interruption

SMARTCARD_IT_RXFT

SMARTCARD RXFIFO threshold reached interruption

SMARTCARD_IT_TXFT

SMARTCARD TXFIFO threshold reached interruption

SMARTCARD Interruption Clear Flags

SMARTCARD_CLEAR_PEF

SMARTCARD parity error clear flag

SMARTCARD_CLEAR_FEF

SMARTCARD framing error clear flag

SMARTCARD_CLEAR_NEF

SMARTCARD noise error detected clear flag

SMARTCARD_CLEAR_OREF

SMARTCARD overrun error clear flag

SMARTCARD_CLEAR_IDLEF

SMARTCARD idle line detected clear flag

SMARTCARD_CLEAR_TXFECF

TXFIFO empty Clear Flag

SMARTCARD_CLEAR_TCF

SMARTCARD transmission complete clear flag

SMARTCARD_CLEAR_TCBGTF

SMARTCARD transmission complete before guard time completion clear flag

SMARTCARD_CLEAR_RTOF

SMARTCARD receiver time out clear flag

SMARTCARD_CLEAR_EOBF

SMARTCARD end of block clear flag

SMARTCARD RXFIFO threshold level

SMARTCARD_RXFIFO_THRESHOLD_1_8

RXFIFO FIFO reaches 1/8 of its depth

SMARTCARD_RXFIFO_THRESHOLD_1_4

RXFIFO FIFO reaches 1/4 of its depth

SMARTCARD_RXFIFO_THRESHOLD_1_2

RXFIFO FIFO reaches 1/2 of its depth

SMARTCARD_RXFIFO_THRESHOLD_3_4

RXFIFO FIFO reaches 3/4 of its depth

SMARTCARD_RXFIFO_THRESHOLD_7_8

RXFIFO FIFO reaches 7/8 of its depth

SMARTCARD_RXFIFO_THRESHOLD_8_8

RXFIFO FIFO becomes full

SMARTCARD Transmission Completion Indication

SMARTCARD_TCBGT

SMARTCARD transmission complete before guard time

SMARTCARD_TC

SMARTCARD transmission complete (flag raised when guard time has elapsed)

SMARTCARD TXFIFO threshold level

SMARTCARD_TXFIFO_THRESHOLD_1_8

TXFIFO reaches 1/8 of its depth

SMARTCARD_TXFIFO_THRESHOLD_1_4

TXFIFO reaches 1/4 of its depth

SMARTCARD_TXFIFO_THRESHOLD_1_2

TXFIFO reaches 1/2 of its depth

SMARTCARD_TXFIFO_THRESHOLD_3_4

TXFIFO reaches 3/4 of its depth

SMARTCARD_TXFIFO_THRESHOLD_7_8

TXFIFO reaches 7/8 of its depth

SMARTCARD_TXFIFO_THRESHOLD_8_8

TXFIFO becomes empty

71 HAL SMBUS Generic Driver

71.1 SMBUS Firmware driver registers structures

71.1.1 SMBUS_InitTypeDef

SMBUS_InitTypeDef is defined in the stm32n6xx_hal_smbus.h

Data Fields

- ***uint32_t Timing***
- ***uint32_t AnalogFilter***
- ***uint32_t OwnAddress1***
- ***uint32_t AddressingMode***
- ***uint32_t DualAddressMode***
- ***uint32_t OwnAddress2***
- ***uint32_t OwnAddress2Masks***
- ***uint32_t GeneralCallMode***
- ***uint32_t NoStretchMode***
- ***uint32_t PacketErrorCheckMode***
- ***uint32_t PeripheralMode***
- ***uint32_t SMBusTimeout***

Field Documentation

- ***uint32_t SMBUS_InitTypeDef::Timing***
  Specifies the SMBUS_TIMINGR_register value. This parameter calculated by referring to SMBUS initialization section in Reference manual
- ***uint32_t SMBUS_InitTypeDef::AnalogFilter***
  Specifies if Analog Filter is enable or not. This parameter can be a value of ***SMBUS_Analog_Filter***
- ***uint32_t SMBUS_InitTypeDef::OwnAddress1***
  Specifies the first device own address. This parameter can be a 7-bit address.
- ***uint32_t SMBUS_InitTypeDef::AddressingMode***
  Specifies addressing mode selected. This parameter can be a value of ***SMBUS_addressing_mode***
- ***uint32_t SMBUS_InitTypeDef::DualAddressMode***
  Specifies if dual addressing mode is selected. This parameter can be a value of ***SMBUS_dual_addressing_mode***
- ***uint32_t SMBUS_InitTypeDef::OwnAddress2***
  Specifies the second device own address if dual addressing mode is selected This parameter can be a 7-bit address.
- ***uint32_t SMBUS_InitTypeDef::OwnAddress2Masks***
  Specifies the acknowledge mask address second device own address if dual addressing mode is selected This parameter can be a value of ***SMBUS_own_address2_masks***.
- ***uint32_t SMBUS_InitTypeDef::GeneralCallMode***
  Specifies if general call mode is selected. This parameter can be a value of ***SMBUS_general_call_addressing_mode***.
- ***uint32_t SMBUS_InitTypeDef::NoStretchMode***
  Specifies if nostretch mode is selected. This parameter can be a value of ***SMBUS_nostretch_mode***
- ***uint32_t SMBUS_InitTypeDef::PacketErrorCheckMode***
  Specifies if Packet Error Check mode is selected. This parameter can be a value of ***SMBUS_packet_error_check_mode***
- ***uint32_t SMBUS_InitTypeDef::PeripheralMode***
  Specifies which mode of Periphal is selected. This parameter can be a value of ***SMBUS_peripheral_mode***
- ***uint32_t SMBUS_InitTypeDef::SMBusTimeout***
  Specifies the content of the 32 Bits SMBUS_TIMEOUT_register value. (Enable bits and different timeout values) This parameter calculated by referring to SMBUS initialization section in Reference manual

71.1.2 __SMBUS_HandleTypeDef

__SMBUS_HandleTypeDef is defined in the stm32n6xx_hal_smbus.h

Data Fields

- ***I2C_TypeDef * Instance***
- ***SMBUS_InitTypeDef Init***
- ***uint8_t * pBuffPtr***
- ***uint16_t XferSize***
- ***__IO uint16_t XferCount***
- ***__IO uint32_t XferOptions***
- ***__IO uint32_t PreviousState***
- ***HAL_LockTypeDef Lock***
- ***__IO uint32_t State***
- ***__IO uint32_t ErrorCode***
- ***void(* MasterTxCpltCallback***
- ***void(* MasterRxCpltCallback***
- ***void(* SlaveTxCpltCallback***
- ***void(* SlaveRxCpltCallback***
- ***void(* ListenCpltCallback***
- ***void(* ErrorCallback***
- ***void(* AddrCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***I2C_TypeDef* __SMBUS_HandleTypeDef::Instance***
  SMBUS registers base address
- ***SMBUS_InitTypeDef __SMBUS_HandleTypeDef::Init***
  SMBUS communication parameters
- ***uint8_t* __SMBUS_HandleTypeDef::pBuffPtr***
  Pointer to SMBUS transfer buffer
- ***uint16_t __SMBUS_HandleTypeDef::XferSize***
  SMBUS transfer size
- ***__IO uint16_t __SMBUS_HandleTypeDef::XferCount***
  SMBUS transfer counter
- ***__IO uint32_t __SMBUS_HandleTypeDef::XferOptions***
  SMBUS transfer options
- ***__IO uint32_t __SMBUS_HandleTypeDef::PreviousState***
  SMBUS communication Previous state
- ***HAL_LockTypeDef __SMBUS_HandleTypeDef::Lock***
  SMBUS locking object
- ***__IO uint32_t __SMBUS_HandleTypeDef::State***
  SMBUS communication state
- ***__IO uint32_t __SMBUS_HandleTypeDef::ErrorCode***
  SMBUS Error code
- ***void(* __SMBUS_HandleTypeDef::MasterTxCpltCallback)(struct __SMBUS_HandleTypeDef *hsmbus)***
  SMBUS Master Tx Transfer completed callback
- ***void(* __SMBUS_HandleTypeDef::MasterRxCpltCallback)(struct __SMBUS_HandleTypeDef *hsmbus)***
  SMBUS Master Rx Transfer completed callback
- ***void(* __SMBUS_HandleTypeDef::SlaveTxCpltCallback)(struct __SMBUS_HandleTypeDef *hsmbus)***
  SMBUS Slave Tx Transfer completed callback
- ***void(* __SMBUS_HandleTypeDef::SlaveRxCpltCallback)(struct __SMBUS_HandleTypeDef *hsmbus)***
  SMBUS Slave Rx Transfer completed callback

- ***void(* __SMBUS_HandleTypeDef::ListenCpltCallback)(struct __SMBUS_HandleTypeDef *hsmbus)***
  SMBUS Listen Complete callback
- ***void(* __SMBUS_HandleTypeDef::ErrorCallback)(struct __SMBUS_HandleTypeDef *hsmbus)***
  SMBUS Error callback
- ***void(* __SMBUS_HandleTypeDef::AddrCallback)(struct __SMBUS_HandleTypeDef *hsmbus, uint8_t TransferDirection, uint16_t AddrMatchCode)***
  SMBUS Slave Address Match callback
- ***void(* __SMBUS_HandleTypeDef::MspInitCallback)(struct __SMBUS_HandleTypeDef *hsmbus)***
  SMBUS Msp Init callback
- ***void(* __SMBUS_HandleTypeDef::MspDeInitCallback)(struct __SMBUS_HandleTypeDef *hsmbus)***
  SMBUS Msp DeInit callback

71.2 SMBUS Firmware driver API description

The following section lists the various functions of the SMBUS library.

71.2.1 How to use this driver

The SMBUS HAL driver can be used as follows:

1. Declare a SMBUS_HandleTypeDef handle structure, for example: SMBUS_HandleTypeDef hsmbus;
2. Initialize the SMBUS low level resources by implementing the HAL_SMBUS_MspInit() API:
   a. Enable the SMBUSx interface clock
   b. SMBUS pins configuration
      - Enable the clock for the SMBUS GPIOs
      - Configure SMBUS pins as alternate function open-drain
   c. NVIC configuration if you need to use interrupt process
      - Configure the SMBUSx interrupt priority
      - Enable the NVIC SMBUS IRQ Channel
3. Configure the Communication Clock Timing, Bus Timeout, Own Address1, Master Addressing mode, Dual Addressing mode, Own Address2, Own Address2 Mask, General call, Nostretch mode, Peripheral mode and Packet Error Check mode in the hsmbus Init structure.
4. Initialize the SMBUS registers by calling the HAL_SMBUS_Init() API:
   – These API's configures also the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customized HAL_SMBUS_MspInit(&hsmbus) API.
5. To check if target device is ready for communication, use the function HAL_SMBUS_IsDeviceReady()
6. For SMBUS IO operations, only one mode of operations is available within this driver

Interrupt mode IO operation

- Transmit in master/host SMBUS mode an amount of data in non-blocking mode using HAL_SMBUS_Master_Transmit_IT()
  – At transmission end of transfer HAL_SMBUS_MasterTxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_SMBUS_MasterTxCpltCallback()
- Receive in master/host SMBUS mode an amount of data in non-blocking mode using HAL_SMBUS_Master_Receive_IT()
  – At reception end of transfer HAL_SMBUS_MasterRxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_SMBUS_MasterRxCpltCallback()
- Abort a master/host SMBUS process communication with Interrupt using HAL_SMBUS_Master_Abort_IT()
  – The associated previous transfer callback is called at the end of abort process
  – mean HAL_SMBUS_MasterTxCpltCallback() in case of previous state was master transmit
  – mean HAL_SMBUS_MasterRxCpltCallback() in case of previous state was master receive

- Enable/disable the Address listen mode in slave/device or host/slave SMBUS mode using HAL_SMBUS_EnableListen_IT() HAL_SMBUS_DisableListen_IT()
  - When address slave/device SMBUS match, HAL_SMBUS_AddrCallback() is executed and users can add their own code to check the Address Match Code and the transmission direction request by master/host (Write/Read).
  - At Listen mode end HAL_SMBUS_ListenCpltCallback() is executed and users can add their own code by customization of function pointer HAL_SMBUS_ListenCpltCallback()
- Transmit in slave/device SMBUS mode an amount of data in non-blocking mode using HAL_SMBUS_Slave_Transmit_IT()
  - At transmission end of transfer HAL_SMBUS_SlaveTxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_SMBUS_SlaveTxCpltCallback()
- Receive in slave/device SMBUS mode an amount of data in non-blocking mode using HAL_SMBUS_Slave_Receive_IT()
  - At reception end of transfer HAL_SMBUS_SlaveRxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_SMBUS_SlaveRxCpltCallback()
- Enable/Disable the SMBUS alert mode using HAL_SMBUS_EnableAlert_IT() or HAL_SMBUS_DisableAlert_IT()
  - When SMBUS Alert is generated HAL_SMBUS_ErrorCallback() is executed and users can add their own code by customization of function pointer HAL_SMBUS_ErrorCallback() to check the Alert Error Code using function HAL_SMBUS_GetError()
- Get HAL state machine or error values using HAL_SMBUS_GetState() or HAL_SMBUS_GetError()
- In case of transfer Error, HAL_SMBUS_ErrorCallback() function is executed and users can add their own code by customization of function pointer HAL_SMBUS_ErrorCallback() to check the Error Code using function HAL_SMBUS_GetError()

SMBUS HAL driver macros list

Below the list of most used macros in SMBUS HAL driver.

- __HAL_SMBUS_ENABLE: Enable the SMBUS peripheral
- __HAL_SMBUS_DISABLE: Disable the SMBUS peripheral
- __HAL_SMBUS_GET_FLAG: Check whether the specified SMBUS flag is set or not
- __HAL_SMBUS_CLEAR_FLAG: Clear the specified SMBUS pending flag
- __HAL_SMBUS_ENABLE_IT: Enable the specified SMBUS interrupt
- __HAL_SMBUS_DISABLE_IT: Disable the specified SMBUS interrupt

Callback registration

The compilation flag USE_HAL_SMBUS_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use Functions HAL_SMBUS_RegisterCallback() or HAL_SMBUS_RegisterAddrCallback() to register an interrupt callback.

Function HAL_SMBUS_RegisterCallback() allows to register following callbacks:

- MasterTxCpltCallback : callback for Master transmission end of transfer.
- MasterRxCpltCallback : callback for Master reception end of transfer.
- SlaveTxCpltCallback : callback for Slave transmission end of transfer.
- SlaveRxCpltCallback : callback for Slave reception end of transfer.
- ListenCpltCallback : callback for end of listen mode.
- ErrorCallback : callback for error detection.
- MspInitCallback : callback for Msp Init.
- MspDeInitCallback : callback for Msp DeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

For specific callback AddrCallback use dedicated register callbacks : HAL_SMBUS_RegisterAddrCallback.

Use function HAL_SMBUS_UnRegisterCallback to reset a callback to the default weak function. HAL_SMBUS_UnRegisterCallback takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks:

- MasterTxCpltCallback : callback for Master transmission end of transfer.

- MasterRxCpltCallback : callback for Master reception end of transfer.
- SlaveTxCpltCallback : callback for Slave transmission end of transfer.
- SlaveRxCpltCallback : callback for Slave reception end of transfer.
- ListenCpltCallback : callback for end of listen mode.
- ErrorCallback : callback for error detection.
- MspInitCallback : callback for Msp Init.
- MspDeInitCallback : callback for Msp DeInit.

For callback AddrCallback use dedicated register callbacks : HAL_SMBUS_UnRegisterAddrCallback.

By default, after the HAL_SMBUS_Init() and when the state is HAL_I2C_STATE_RESET all callbacks are set to the corresponding weak functions: examples HAL_SMBUS_MasterTxCpltCallback(), HAL_SMBUS_MasterRxCpltCallback(). Exception done for MspInit and MspDeInit functions that are reset to the legacy weak functions in the HAL_SMBUS_Init()/ HAL_SMBUS_DeInit() only when these callbacks are null (not registered beforehand). If MspInit or MspDeInit are not null, the HAL_SMBUS_Init()/ HAL_SMBUS_DeInit() keep and use the user MspInit/MspDeInit callbacks (registered beforehand) whatever the state.

Callbacks can be registered/unregistered in HAL_I2C_STATE_READY state only. Exception done MspInit/ MspDeInit functions that can be registered/unregistered in HAL_I2C_STATE_READY or HAL_I2C_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. Then, the user first registers the MspInit/MspDeInit user callbacks using HAL_SMBUS_RegisterCallback() before calling HAL_SMBUS_DeInit() or HAL_SMBUS_Init() function.

When the compilation flag USE_HAL_SMBUS_REGISTER_CALLBACKS is set to 0 or not defined, the callback registration feature is not available and all callbacks are set to the corresponding weak functions.

Note: You can refer to the SMBUS HAL driver header file for more useful macros

71.2.2 Detailed description of functions

HAL_SMBUS_Init

Function name

HAL_StatusTypeDef HAL_SMBUS_Init (SMBUS_HandleTypeDef * hsmbus)

Function description

HAL_SMBUS_DeInit

Function name

HAL_StatusTypeDef HAL_SMBUS_DeInit (SMBUS_HandleTypeDef * hsmbus)

Function description

HAL_SMBUS_MspInit

Function name

void HAL_SMBUS_MspInit (SMBUS_HandleTypeDef * hsmbus)

Function description

HAL_SMBUS_MspDeInit

Function name

void HAL_SMBUS_MspDeInit (SMBUS_HandleTypeDef * hsmbus)

Function description

HAL_SMBUS_ConfigAnalogFilter

Function name

HAL_StatusTypeDef HAL_SMBUS_ConfigAnalogFilter (SMBUS_HandleTypeDef * hsmbus, uint32_t AnalogFilter)

Function description

HAL_SMBUS_ConfigDigitalFilter

Function name

HAL_StatusTypeDef HAL_SMBUS_ConfigDigitalFilter (SMBUS_HandleTypeDef * hsmbus, uint32_t DigitalFilter)

Function description

HAL_SMBUS_RegisterCallback

Function name

HAL_StatusTypeDef HAL_SMBUS_RegisterCallback (SMBUS_HandleTypeDef * hsmbus, HAL_SMBUS_CallbackIDTypeDef CallbackID, pSMBUS_CallbackTypeDef pCallback)

Function description

HAL_SMBUS_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_SMBUS_UnRegisterCallback (SMBUS_HandleTypeDef * hsmbus, HAL_SMBUS_CallbackIDTypeDef CallbackID)

Function description

HAL_SMBUS_RegisterAddrCallback

Function name

HAL_StatusTypeDef HAL_SMBUS_RegisterAddrCallback (SMBUS_HandleTypeDef * hsmbus, pSMBUS_AddrCallbackTypeDef pCallback)

Function description

HAL_SMBUS_UnRegisterAddrCallback

Function name

HAL_StatusTypeDef HAL_SMBUS_UnRegisterAddrCallback (SMBUS_HandleTypeDef * hsmbus)

Function description

HAL_SMBUS_IsDeviceReady

Function name

HAL_StatusTypeDef HAL_SMBUS_IsDeviceReady (SMBUS_HandleTypeDef * hsmbus, uint16_t DevAddress, uint32_t Trials, uint32_t Timeout)

Function description

HAL_SMBUS_Master_Transmit_IT

Function name

HAL_StatusTypeDef HAL_SMBUS_Master_Transmit_IT (SMBUS_HandleTypeDef * hsmbus, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t XferOptions)

Function description

HAL_SMBUS_Master_Receive_IT

Function name

HAL_StatusTypeDef HAL_SMBUS_Master_Receive_IT (SMBUS_HandleTypeDef * hsmbus, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t XferOptions)

Function description

HAL_SMBUS_Master_Abort_IT

Function name

HAL_StatusTypeDef HAL_SMBUS_Master_Abort_IT (SMBUS_HandleTypeDef * hsmbus, uint16_t DevAddress)

Function description

HAL_SMBUS_Slave_Transmit_IT

Function name

HAL_StatusTypeDef HAL_SMBUS_Slave_Transmit_IT (SMBUS_HandleTypeDef * hsmbus, uint8_t * pData, uint16_t Size, uint32_t XferOptions)

Function description

HAL_SMBUS_Slave_Receive_IT

Function name

HAL_StatusTypeDef HAL_SMBUS_Slave_Receive_IT (SMBUS_HandleTypeDef * hsmbus, uint8_t * pData, uint16_t Size, uint32_t XferOptions)

Function description

HAL_SMBUS_EnableAlert_IT

Function name

HAL_StatusTypeDef HAL_SMBUS_EnableAlert_IT (SMBUS_HandleTypeDef * hsmbus)

Function description

HAL_SMBUS_DisableAlert_IT

Function name

HAL_StatusTypeDef HAL_SMBUS_DisableAlert_IT (SMBUS_HandleTypeDef * hsmbus)

Function description

HAL_SMBUS_EnableListen_IT

Function name

HAL_StatusTypeDef HAL_SMBUS_EnableListen_IT (SMBUS_HandleTypeDef * hsmbus)

Function description

HAL_SMBUS_DisableListen_IT

Function name

HAL_StatusTypeDef HAL_SMBUS_DisableListen_IT (SMBUS_HandleTypeDef * hsmbus)

Function description

HAL_SMBUS_EV_IRQHandler

Function name

void HAL_SMBUS_EV_IRQHandler (SMBUS_HandleTypeDef * hsmbus)

Function description

HAL_SMBUS_ER_IRQHandler

Function name

void HAL_SMBUS_ER_IRQHandler (SMBUS_HandleTypeDef * hsmbus)

Function description

HAL_SMBUS_MasterTxCpltCallback

Function name

void HAL_SMBUS_MasterTxCpltCallback (SMBUS_HandleTypeDef * hsmbus)

Function description

HAL_SMBUS_MasterRxCpltCallback

Function name

void HAL_SMBUS_MasterRxCpltCallback (SMBUS_HandleTypeDef * hsmbus)

Function description

HAL_SMBUS_SlaveTxCpltCallback

Function name

void HAL_SMBUS_SlaveTxCpltCallback (SMBUS_HandleTypeDef * hsmbus)

Function description

HAL_SMBUS_SlaveRxCpltCallback

Function name

void HAL_SMBUS_SlaveRxCpltCallback (SMBUS_HandleTypeDef * hsmbus)

Function description

HAL_SMBUS_AddrCallback

Function name

void HAL_SMBUS_AddrCallback (SMBUS_HandleTypeDef * hsmbus, uint8_t TransferDirection, uint16_t AddrMatchCode)

Function description

HAL_SMBUS_ListenCpltCallback

Function name

void HAL_SMBUS_ListenCpltCallback (SMBUS_HandleTypeDef * hsmbus)

Function description

HAL_SMBUS_ErrorCallback

Function name

void HAL_SMBUS_ErrorCallback (SMBUS_HandleTypeDef * hsmbus)

Function description

HAL_SMBUS_GetState

Function name

uint32_t HAL_SMBUS_GetState (const SMBUS_HandleTypeDef * hsmbus)

Function description

HAL_SMBUS_GetError

Function name

uint32_t HAL_SMBUS_GetError (const SMBUS_HandleTypeDef * hsmbus)

Function description

71.3 SMBUS Firmware driver defines

The following section lists the various define and macros of the module.

71.3.1 SMBUS

SMBUS

SMBUS addressing mode

SMBUS_ADDRESSINGMODE_7BIT

SMBUS Analog Filter

SMBUS_ANALOGFILTER_ENABLE

SMBUS_ANALOGFILTER_DISABLE

SMBUS dual addressing mode

SMBUS_DUALADDRESS_DISABLE

SMBUS_DUALADDRESS_ENABLE

SMBUS Error Code definition

HAL_SMBUS_ERROR_NONE

No error

HAL_SMBUS_ERROR_BERR

BERR error

HAL_SMBUS_ERROR_ARLO

ARLO error

HAL_SMBUS_ERROR_ACKF

ACKF error

HAL_SMBUS_ERROR_OVR

OVR error

HAL_SMBUS_ERROR_HALTIMEOUT

Timeout error

HAL_SMBUS_ERROR_BUSTIMEOUT

Bus Timeout error

HAL_SMBUS_ERROR_ALERT

Alert error

HAL_SMBUS_ERROR_PECERR

PEC error

HAL_SMBUS_ERROR_INVALID_CALLBACK

Invalid Callback error

HAL_SMBUS_ERROR_INVALID_PARAM

Invalid Parameters error

SMBUS Exported Macros

__HAL_SMBUS_RESET_HANDLE_STATE

Description:

- Reset SMBUS handle state.

Parameters:

- __HANDLE__: specifies the SMBUS Handle.

Return value:

- None

__HAL_SMBUS_ENABLE_IT

Description:

- Enable the specified SMBUS interrupts.

Parameters:

- __HANDLE__: specifies the SMBUS Handle.
- __INTERRUPT__: specifies the interrupt source to enable. This parameter can be one of the following values:
  - SMBUS_IT_ERRI Errors interrupt enable
  - SMBUS_IT_TCI Transfer complete interrupt enable
  - SMBUS_IT_STOPI STOP detection interrupt enable
  - SMBUS_IT_NACKI NACK received interrupt enable
  - SMBUS_IT_ADDRI Address match interrupt enable
  - SMBUS_IT_RXI RX interrupt enable
  - SMBUS_IT_TXI TX interrupt enable

Return value:

- None

__HAL_SMBUS_DISABLE_IT

Description:

- Disable the specified SMBUS interrupts.

Parameters:

- __HANDLE__: specifies the SMBUS Handle.
- __INTERRUPT__: specifies the interrupt source to disable. This parameter can be one of the following values:
  - SMBUS_IT_ERRI Errors interrupt enable
  - SMBUS_IT_TCI Transfer complete interrupt enable
  - SMBUS_IT_STOPI STOP detection interrupt enable
  - SMBUS_IT_NACKI NACK received interrupt enable
  - SMBUS_IT_ADDRI Address match interrupt enable
  - SMBUS_IT_RXI RX interrupt enable
  - SMBUS_IT_TXI TX interrupt enable

Return value:

- None

__HAL_SMBUS_GET_IT_SOURCE

Description:

- Check whether the specified SMBUS interrupt source is enabled or not.

Parameters:

- __HANDLE__: specifies the SMBUS Handle.
- __INTERRUPT__: specifies the SMBUS interrupt source to check. This parameter can be one of the following values:
  - SMBUS_IT_ERRI Errors interrupt enable
  - SMBUS_IT_TCI Transfer complete interrupt enable
  - SMBUS_IT_STOPI STOP detection interrupt enable
  - SMBUS_IT_NACKI NACK received interrupt enable
  - SMBUS_IT_ADDRI Address match interrupt enable
  - SMBUS_IT_RXI RX interrupt enable
  - SMBUS_IT_TXI TX interrupt enable

Return value:

- The: new state of __IT__ (SET or RESET).

SMBUS_FLAG_MASK

Description:

- Check whether the specified SMBUS flag is set or not.

Parameters:

- __HANDLE__: specifies the SMBUS Handle.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - SMBUS_FLAG_TXE Transmit data register empty
  - SMBUS_FLAG_TXIS Transmit interrupt status
  - SMBUS_FLAG_RXNE Receive data register not empty
  - SMBUS_FLAG_ADDR Address matched (slave mode)
  - SMBUS_FLAG_AF NACK received flag
  - SMBUS_FLAG_STOPF STOP detection flag
  - SMBUS_FLAG_TC Transfer complete (master mode)
  - SMBUS_FLAG_TCR Transfer complete reload
  - SMBUS_FLAG_BERR Bus error
  - SMBUS_FLAG_ARLO Arbitration lost
  - SMBUS_FLAG_OVR Overrun/Underrun
  - SMBUS_FLAG_PECERR PEC error in reception
  - SMBUS_FLAG_TIMEOUT Timeout or Tlow detection flag
  - SMBUS_FLAG_ALERT SMBus alert
  - SMBUS_FLAG_BUSY Bus busy
  - SMBUS_FLAG_DIR Transfer direction (slave mode)

Return value:

- The: new state of __FLAG__ (SET or RESET).

__HAL_SMBUS_GET_FLAG

__HAL_SMBUS_CLEAR_FLAG

Description:

- Clear the SMBUS pending flags which are cleared by writing 1 in a specific bit.

Parameters:

- __HANDLE__: specifies the SMBUS Handle.
- __FLAG__: specifies the flag to clear. This parameter can be any combination of the following values:
  - SMBUS_FLAG_TXE Transmit data register empty
  - SMBUS_FLAG_ADDR Address matched (slave mode)
  - SMBUS_FLAG_AF NACK received flag
  - SMBUS_FLAG_STOPF STOP detection flag
  - SMBUS_FLAG_BERR Bus error
  - SMBUS_FLAG_ARLO Arbitration lost
  - SMBUS_FLAG_OVR Overrun/Underrun
  - SMBUS_FLAG_PECERR PEC error in reception
  - SMBUS_FLAG_TIMEOUT Timeout or Tlow detection flag
  - SMBUS_FLAG_ALERT SMBus alert

Return value:

- None

__HAL_SMBUS_ENABLE

Description:

- Enable the specified SMBUS peripheral.

Parameters:

- __HANDLE__: specifies the SMBUS Handle.

Return value:

- None

__HAL_SMBUS_DISABLE

Description:

- Disable the specified SMBUS peripheral.

Parameters:

- __HANDLE__: specifies the SMBUS Handle.

Return value:

- None

__HAL_SMBUS_GENERATE_NACK

Description:

- Generate a Non-Acknowledge SMBUS peripheral in Slave mode.

Parameters:

- __HANDLE__: specifies the SMBUS Handle.

Return value:

- None

SMBUS Flag definition

SMBUS_FLAG_TXE

SMBUS_FLAG_TXIS

SMBUS_FLAG_RXNE

SMBUS_FLAG_ADDR

SMBUS_FLAG_AF

SMBUS_FLAG_STOPF

SMBUS_FLAG_TC

SMBUS_FLAG_TCR

SMBUS_FLAG_BERR

SMBUS_FLAG_ARLO

SMBUS_FLAG_OVR

SMBUS_FLAG_PECERR

SMBUS_FLAG_TIMEOUT

SMBUS_FLAG_ALERT

SMBUS_FLAG_BUSY

SMBUS_FLAG_DIR

SMBUS general call addressing mode

SMBUS_GENERALCALL_DISABLE

SMBUS_GENERALCALL_ENABLE

SMBUS Interrupt configuration definition

SMBUS_IT_ERRI

SMBUS_IT_TCI

SMBUS_IT_STOPI

SMBUS_IT_NACKI

SMBUS_IT_ADDRI

SMBUS_IT_RXI

SMBUS_IT_TXI

SMBUS_IT_TX

SMBUS_IT_RX

SMBUS_IT_ALERT

SMBUS_IT_ADDR

SMBUS nostretch mode

SMBUS_NOSTRETCH_DISABLE

SMBUS_NOSTRETCH_ENABLE

SMBUS ownaddress2 masks

SMBUS_OA2_NOMASK

SMBUS_OA2_MASK01

SMBUS_OA2_MASK02

SMBUS_OA2_MASK03

SMBUS_OA2_MASK04

SMBUS_OA2_MASK05

SMBUS_OA2_MASK06

SMBUS_OA2_MASK07

SMBUS packet error check mode

SMBUS_PEC_DISABLE

SMBUS_PEC_ENABLE

SMBUS peripheral mode

SMBUS_PERIPHERAL_MODE_SMBUS_HOST

SMBUS_PERIPHERAL_MODE_SMBUS_SLAVE

SMBUS_PERIPHERAL_MODE_SMBUS_SLAVE_ARP

SMBUS ReloadEndMode definition

SMBUS_SOFTEND_MODE

SMBUS_RELOAD_MODE

SMBUS_AUTOEND_MODE

SMBUS_SENDPEC_MODE

SMBUS StartStopMode definition

SMBUS_NO_STARTSTOP

SMBUS_GENERATE_STOP

SMBUS_GENERATE_START_READ

SMBUS_GENERATE_START_WRITE

SMBUS XferOptions definition

SMBUS_FIRST_FRAME

SMBUS_NEXT_FRAME

SMBUS_FIRST_AND_LAST_FRAME_NO_PEC

SMBUS_LAST_FRAME_NO_PEC

SMBUS_FIRST_FRAME_WITH_PEC

SMBUS_FIRST_AND_LAST_FRAME_WITH_PEC

SMBUS_LAST_FRAME_WITH_PEC

SMBUS_OTHER_FRAME_NO_PEC

SMBUS_OTHER_FRAME_WITH_PEC

SMBUS_OTHER_AND_LAST_FRAME_NO_PEC

SMBUS_OTHER_AND_LAST_FRAME_WITH_PEC

72 HAL SMBUS Extension Driver

72.1 SMBUSEx Firmware driver API description

The following section lists the various functions of the SMBUSEx library.

72.1.1 SMBUS peripheral Extended features

Comparing to other previous devices, the SMBUS interface for STM32N6xx devices contains the following additional features

- Disable or enable wakeup from Stop mode(s)

72.1.2 How to use this driver

72.1.3 Detailed description of functions

HAL_SMBUSEx_EnableWakeUp

Function name

HAL_StatusTypeDef HAL_SMBUSEx_EnableWakeUp (SMBUS_HandleTypeDef * hsmbus)

Function description

HAL_SMBUSEx_DisableWakeUp

Function name

HAL_StatusTypeDef HAL_SMBUSEx_DisableWakeUp (SMBUS_HandleTypeDef * hsmbus)

Function description

HAL_SMBUSEx_ConfigFastModePlus

Function name

HAL_StatusTypeDef HAL_SMBUSEx_ConfigFastModePlus (SMBUS_HandleTypeDef * hsmbus, uint32_t FastModePlus)

Function description

72.2 SMBUSEx Firmware driver defines

The following section lists the various define and macros of the module.

72.2.1 SMBUSEx

SMBUSEx

SMBUS Extended Fast Mode Plus

SMBUS_FASTMODEPLUS_ENABLE

Enable Fast Mode Plus

SMBUS_FASTMODEPLUS_DISABLE

Disable Fast Mode Plus

73 HAL SPDIFRX Generic Driver

73.1 SPDIFRX Firmware driver registers structures

73.1.1 SPDIFRX_InitTypeDef

SPDIFRX_InitTypeDef is defined in the stm32n6xx_hal_spdifrx.h

Data Fields

- ***uint32_t InputSelection***
- ***uint32_t Retries***
- ***uint32_t WaitForActivity***
- ***uint32_t ChannelSelection***
- ***uint32_t DataFormat***
- ***uint32_t StereoMode***
- ***uint32_t PreambleTypeMask***
- ***uint32_t ChannelStatusMask***
- ***uint32_t ValidityBitMask***
- ***uint32_t ParityErrorMask***
- ***FunctionalState SymbolClockGen***
- ***FunctionalState BackupSymbolClockGen***

Field Documentation

- ***uint32_t SPDIFRX_InitTypeDef::InputSelection***
  Specifies the SPDIF input selection. This parameter can be a value of ***SPDIFRX_Input_Selection***
- ***uint32_t SPDIFRX_InitTypeDef::Retries***
  Specifies the Maximum allowed re-tries during synchronization phase. This parameter can be a value of ***SPDIFRX_Max_Retries***
- ***uint32_t SPDIFRX_InitTypeDef::WaitForActivity***
  Specifies the wait for activity on SPDIF selected input. This parameter can be a value of ***SPDIFRX_Wait_For_Activity***.
- ***uint32_t SPDIFRX_InitTypeDef::ChannelSelection***
  Specifies whether the control flow will take the channel status from channel A or B. This parameter can be a value of ***SPDIFRX_Channel_Selection***
- ***uint32_t SPDIFRX_InitTypeDef::DataFormat***
  Specifies the Data samples format (LSB, MSB, ...). This parameter can be a value of ***SPDIFRX_Data_Format***
- ***uint32_t SPDIFRX_InitTypeDef::StereoMode***
  Specifies whether the peripheral is in stereo or mono mode. This parameter can be a value of ***SPDIFRX_Stereo_Mode***
- ***uint32_t SPDIFRX_InitTypeDef::PreambleTypeMask***
  Specifies whether The preamble type bits are copied or not into the received frame. This parameter can be a value of ***SPDIFRX_PT_Mask***
- ***uint32_t SPDIFRX_InitTypeDef::ChannelStatusMask***
  Specifies whether the channel status and user bits are copied or not into the received frame. This parameter can be a value of ***SPDIFRX_ChannelStatus_Mask***
- ***uint32_t SPDIFRX_InitTypeDef::ValidityBitMask***
  Specifies whether the validity bit is copied or not into the received frame. This parameter can be a value of ***SPDIFRX_V_Mask***
- ***uint32_t SPDIFRX_InitTypeDef::ParityErrorMask***
  Specifies whether the parity error bit is copied or not into the received frame. This parameter can be a value of ***SPDIFRX_PE_Mask***
- ***FunctionalState SPDIFRX_InitTypeDef::SymbolClockGen***
  Enable/Disable the SPDIFRX Symbol Clock generation. This parameter can be set to Enable or Disable

- ***FunctionalState SPDIFRX_InitTypeDef::BackupSymbolClockGen***
  Enable/Disable the SPDIFRX Backup Symbol Clock generation. This parameter can be set to Enable or Disable

73.1.2 SPDIFRX_SetDataFormatTypeDef

SPDIFRX_SetDataFormatTypeDef is defined in the stm32n6xx_hal_spdifrx.h

Data Fields

- ***uint32_t DataFormat***
- ***uint32_t StereoMode***
- ***uint32_t PreambleTypeMask***
- ***uint32_t ChannelStatusMask***
- ***uint32_t ValidityBitMask***
- ***uint32_t ParityErrorMask***

Field Documentation

- ***uint32_t SPDIFRX_SetDataFormatTypeDef::DataFormat***
  Specifies the Data samples format (LSB, MSB, ...). This parameter can be a value of ***SPDIFRX_Data_Format***
- ***uint32_t SPDIFRX_SetDataFormatTypeDef::StereoMode***
  Specifies whether the peripheral is in stereo or mono mode. This parameter can be a value of ***SPDIFRX_Stereo_Mode***
- ***uint32_t SPDIFRX_SetDataFormatTypeDef::PreambleTypeMask***
  Specifies whether The preamble type bits are copied or not into the received frame. This parameter can be a value of ***SPDIFRX_PT_Mask***
- ***uint32_t SPDIFRX_SetDataFormatTypeDef::ChannelStatusMask***
  Specifies whether the channel status and user bits are copied or not into the received frame. This parameter can be a value of ***SPDIFRX_ChannelStatus_Mask***
- ***uint32_t SPDIFRX_SetDataFormatTypeDef::ValidityBitMask***
  Specifies whether the validity bit is copied or not into the received frame. This parameter can be a value of ***SPDIFRX_V_Mask***
- ***uint32_t SPDIFRX_SetDataFormatTypeDef::ParityErrorMask***
  Specifies whether the parity error bit is copied or not into the received frame. This parameter can be a value of ***SPDIFRX_PE_Mask***

73.1.3 __SPDIFRX_HandleTypeDef

__SPDIFRX_HandleTypeDef is defined in the stm32n6xx_hal_spdifrx.h

Data Fields

- ***SPDIFRX_TypeDef * Instance***
- ***SPDIFRX_InitTypeDef Init***
- ***uint32_t * pRxBuffPtr***
- ***uint32_t * pCsBuffPtr***
- ***__IO uint16_t RxXferSize***
- ***__IO uint16_t RxXferCount***
- ***__IO uint16_t CsXferSize***
- ***__IO uint16_t CsXferCount***
- ***DMA_HandleTypeDef * hdmaCsRx***
- ***DMA_HandleTypeDef * hdmaDrRx***
- ***__IO HAL_LockTypeDef Lock***
- ***__IO HAL_SPDIFRX_StateTypeDef State***
- ***__IO uint32_t ErrorCode***
- ***void(* RxHalfCpltCallback***
- ***void(* RxCpltCallback***
- ***void(* CxHalfCpltCallback***
- ***void(* CxCpltCallback***

- *void(* ErrorCallback*
- *void(* MspInitCallback*
- *void(* MspDeInitCallback*

Field Documentation

- ***SPDIFRX_TypeDef* __SPDIFRX_HandleTypeDef::Instance***
- ***SPDIFRX_InitTypeDef __SPDIFRX_HandleTypeDef::Init***
- ***uint32_t* __SPDIFRX_HandleTypeDef::pRxBuffPtr***
- ***uint32_t* __SPDIFRX_HandleTypeDef::pCsBuffPtr***
- ***__IO uint16_t __SPDIFRX_HandleTypeDef::RxXferSize***
- ***__IO uint16_t __SPDIFRX_HandleTypeDef::RxXferCount***
- ***__IO uint16_t __SPDIFRX_HandleTypeDef::CsXferSize***
- ***__IO uint16_t __SPDIFRX_HandleTypeDef::CsXferCount***
- ***DMA_HandleTypeDef* __SPDIFRX_HandleTypeDef::hdmaCsRx***
- ***DMA_HandleTypeDef* __SPDIFRX_HandleTypeDef::hdmaDrRx***
- ***__IO HAL_LockTypeDef __SPDIFRX_HandleTypeDef::Lock***
- ***__IO HAL_SPDIFRX_StateTypeDef __SPDIFRX_HandleTypeDef::State***
- ***__IO uint32_t __SPDIFRX_HandleTypeDef::ErrorCode***
- ***void(* __SPDIFRX_HandleTypeDef::RxHalfCpltCallback)(struct __SPDIFRX_HandleTypeDef *hspdif)***
  SPDIFRX Data flow half completed callback
- ***void(* __SPDIFRX_HandleTypeDef::RxCpltCallback)(struct __SPDIFRX_HandleTypeDef *hspdif)***
  SPDIFRX Data flow completed callback
- ***void(* __SPDIFRX_HandleTypeDef::CxHalfCpltCallback)(struct __SPDIFRX_HandleTypeDef *hspdif)***
  SPDIFRX Control flow half completed callback
- ***void(* __SPDIFRX_HandleTypeDef::CxCpltCallback)(struct __SPDIFRX_HandleTypeDef *hspdif)***
  SPDIFRX Control flow completed callback
- ***void(* __SPDIFRX_HandleTypeDef::ErrorCallback)(struct __SPDIFRX_HandleTypeDef *hspdif)***
  SPDIFRX error callback
- ***void(* __SPDIFRX_HandleTypeDef::MspInitCallback)(struct __SPDIFRX_HandleTypeDef *hspdif)***
  SPDIFRX Msp Init callback
- ***void(* __SPDIFRX_HandleTypeDef::MspDeInitCallback)(struct __SPDIFRX_HandleTypeDef *hspdif)***
  SPDIFRX Msp DeInit callback

73.2 SPDIFRX Firmware driver API description

The following section lists the various functions of the SPDIFRX library.

73.2.1 How to use this driver

The SPDIFRX HAL driver can be used as follow:

1. Declare SPDIFRX_HandleTypeDef handle structure.

2. Initialize the SPDIFRX low level resources by implement the HAL_SPDIFRX_MspInit() API:
   a. Enable the SPDIFRX interface clock.
   b. SPDIFRX pins configuration:
      - Enable the clock for the SPDIFRX GPIOs.
      - Configure these SPDIFRX pins as alternate function pull-up.
   c. NVIC configuration if you need to use interrupt process (HAL_SPDIFRX_ReceiveCtrlFlow_IT() and HAL_SPDIFRX_ReceiveDataFlow_IT() API's).
      - Configure the SPDIFRX interrupt priority.
      - Enable the NVIC SPDIFRX IRQ handle.
   d. DMA Configuration if you need to use DMA process (HAL_SPDIFRX_ReceiveDataFlow_DMA() and HAL_SPDIFRX_ReceiveCtrlFlow_DMA() API's).
      - Declare a DMA handle structure for the reception of the Data Flow channel.
      - Declare a DMA handle structure for the reception of the Control Flow channel.
      - Enable the DMAx interface clock.
      - Configure the declared DMA handle structure CtrlRx/DataRx with the required parameters.
      - Configure the DMA Channel.
      - Associate the initialized DMA handle to the SPDIFRX DMA CtrlRx/DataRx handle.
      - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA CtrlRx/DataRx channel.
3. Program the input selection, re-tries number, wait for activity, channel status selection, data format, stereo mode and masking of user bits using HAL_SPDIFRX_Init() function.

Note: The specific SPDIFRX interrupts (RXNE/CSRNE and Error Interrupts) will be managed using the macros __SPDIFRX_ENABLE_IT() and __SPDIFRX_DISABLE_IT() inside the receive process.

Note: Make sure that ck_spdif clock is configured.

4. Three operation modes are available within this driver :

Polling mode for reception operation (for debug purpose)

- Receive data flow in blocking mode using HAL_SPDIFRX_ReceiveDataFlow()
- Receive control flow of data in blocking mode using HAL_SPDIFRX_ReceiveCtrlFlow()

Interrupt mode for reception operation

- Receive an amount of data (Data Flow) in non blocking mode using HAL_SPDIFRX_ReceiveDataFlow_IT()
- Receive an amount of data (Control Flow) in non blocking mode using HAL_SPDIFRX_ReceiveCtrlFlow_IT()
- At reception end of half transfer HAL_SPDIFRX_RxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL_SPDIFRX_RxHalfCpltCallback
- At reception end of transfer HAL_SPDIFRX_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL_SPDIFRX_RxCpltCallback
- In case of transfer Error, HAL_SPDIFRX_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_SPDIFRX_ErrorCallback

DMA mode for reception operation

- Receive an amount of data (Data Flow) in non blocking mode (DMA) using HAL_SPDIFRX_ReceiveDataFlow_DMA()
- Receive an amount of data (Control Flow) in non blocking mode (DMA) using HAL_SPDIFRX_ReceiveCtrlFlow_DMA()
- At reception end of half transfer HAL_SPDIFRX_RxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL_SPDIFRX_RxHalfCpltCallback
- At reception end of transfer HAL_SPDIFRX_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL_SPDIFRX_RxCpltCallback
- In case of transfer Error, HAL_SPDIFRX_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_SPDIFRX_ErrorCallback

- Stop the DMA Transfer using HAL_SPDIFRX_DMAStop()

SPDIFRX HAL driver macros list

Below the list of most used macros in SPDIFRX HAL driver.

- __HAL_SPDIFRX_IDLE: Disable the specified SPDIFRX peripheral (IDLE State)
- __HAL_SPDIFRX_SYNC: Enable the synchronization state of the specified SPDIFRX peripheral (SYNC State)
- __HAL_SPDIFRX_RCV: Enable the receive state of the specified SPDIFRX peripheral (RCV State)
- __HAL_SPDIFRX_ENABLE_IT : Enable the specified SPDIFRX interrupts
- __HAL_SPDIFRX_DISABLE_IT : Disable the specified SPDIFRX interrupts
- __HAL_SPDIFRX_GET_FLAG: Check whether the specified SPDIFRX flag is set or not.

Note: You can refer to the SPDIFRX HAL driver header file for more useful macros

Callback registration

73.2.2 Detailed description of functions

HAL_SPDIFRX_Init

Function name

HAL_StatusTypeDef HAL_SPDIFRX_Init (SPDIFRX_HandleTypeDef * hspdif)

Function description

HAL_SPDIFRX_DeInit

Function name

HAL_StatusTypeDef HAL_SPDIFRX_DeInit (SPDIFRX_HandleTypeDef * hspdif)

Function description

HAL_SPDIFRX_MspInit

Function name

void HAL_SPDIFRX_MspInit (SPDIFRX_HandleTypeDef * hspdif)

Function description

HAL_SPDIFRX_MspDeInit

Function name

void HAL_SPDIFRX_MspDeInit (SPDIFRX_HandleTypeDef * hspdif)

Function description

HAL_SPDIFRX_SetDataFormat

Function name

HAL_StatusTypeDef HAL_SPDIFRX_SetDataFormat (SPDIFRX_HandleTypeDef * hspdif, SPDIFRX_SetDataFormatTypeDef sDataFormat)

Function description

HAL_SPDIFRX_RegisterCallback

Function name

HAL_StatusTypeDef HAL_SPDIFRX_RegisterCallback (SPDIFRX_HandleTypeDef * hspdif, HAL_SPDIFRX_CallbackIDTypeDef CallbackID, pSPDIFRX_CallbackTypeDef pCallback)

Function description

HAL_SPDIFRX_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_SPDIFRX_UnRegisterCallback (SPDIFRX_HandleTypeDef * hspdif, HAL_SPDIFRX_CallbackIDTypeDef CallbackID)

Function description

HAL_SPDIFRX_ReceiveDataFlow

Function name

HAL_StatusTypeDef HAL_SPDIFRX_ReceiveDataFlow (SPDIFRX_HandleTypeDef * hspdif, uint32_t * pData, uint16_t Size, uint32_t Timeout)

Function description

HAL_SPDIFRX_ReceiveCtrlFlow

Function name

HAL_StatusTypeDef HAL_SPDIFRX_ReceiveCtrlFlow (SPDIFRX_HandleTypeDef * hspdif, uint32_t * pData, uint16_t Size, uint32_t Timeout)

Function description

HAL_SPDIFRX_ReceiveCtrlFlow_IT

Function name

HAL_StatusTypeDef HAL_SPDIFRX_ReceiveCtrlFlow_IT (SPDIFRX_HandleTypeDef * hspdif, uint32_t * pData, uint16_t Size)

Function description

HAL_SPDIFRX_ReceiveDataFlow_IT

Function name

HAL_StatusTypeDef HAL_SPDIFRX_ReceiveDataFlow_IT (SPDIFRX_HandleTypeDef * hspdif, uint32_t * pData, uint16_t Size)

Function description

HAL_SPDIFRX_IRQHandler

Function name

void HAL_SPDIFRX_IRQHandler (SPDIFRX_HandleTypeDef * hspdif)

Function description

HAL_SPDIFRX_ReceiveCtrlFlow_DMA

Function name

HAL_StatusTypeDef HAL_SPDIFRX_ReceiveCtrlFlow_DMA (SPDIFRX_HandleTypeDef * hspdif, uint32_t * pData, uint16_t Size)

Function description

HAL_SPDIFRX_ReceiveDataFlow_DMA

Function name

HAL_StatusTypeDef HAL_SPDIFRX_ReceiveDataFlow_DMA (SPDIFRX_HandleTypeDef * hspdif, uint32_t * pData, uint16_t Size)

Function description

HAL_SPDIFRX_DMAStop

Function name

HAL_StatusTypeDef HAL_SPDIFRX_DMAStop (SPDIFRX_HandleTypeDef * hspdif)

Function description

HAL_SPDIFRX_RxHalfCpltCallback

Function name

void HAL_SPDIFRX_RxHalfCpltCallback (SPDIFRX_HandleTypeDef * hspdif)

Function description

HAL_SPDIFRX_RxCpltCallback

Function name

void HAL_SPDIFRX_RxCpltCallback (SPDIFRX_HandleTypeDef * hspdif)

Function description

HAL_SPDIFRX_ErrorCallback

Function name

void HAL_SPDIFRX_ErrorCallback (SPDIFRX_HandleTypeDef * hspdif)

Function description

HAL_SPDIFRX_CxHalfCpltCallback

Function name

void HAL_SPDIFRX_CxHalfCpltCallback (SPDIFRX_HandleTypeDef * hspdif)

Function description

HAL_SPDIFRX_CxCpltCallback

Function name

void HAL_SPDIFRX_CxCpltCallback (SPDIFRX_HandleTypeDef * hspdif)

Function description

HAL_SPDIFRX_GetState

Function name

HAL_SPDIFRX_StateTypeDef HAL_SPDIFRX_GetState (SPDIFRX_HandleTypeDef const *const hspdif)

Function description

HAL_SPDIFRX_GetError

Function name

uint32_t HAL_SPDIFRX_GetError (SPDIFRX_HandleTypeDef const *const hspdif)

Function description

73.3 SPDIFRX Firmware driver defines

The following section lists the various define and macros of the module.

73.3.1 SPDIFRX

SPDIFRX

SPDIFRX Channel Status Mask

SPDIFRX_CHANNELSTATUS_OFF

SPDIFRX_CHANNELSTATUS_ON

SPDIFRX Channel Selection

SPDIFRX_CHANNEL_A

SPDIFRX_CHANNEL_B

SPDIFRX Data Format

SPDIFRX_DATAFORMAT_LSB

SPDIFRX_DATAFORMAT_MSB

SPDIFRX_DATAFORMAT_32BITS

SPDIFRX Error Code

HAL_SPDIFRX_ERROR_NONE

No error

HAL_SPDIFRX_ERROR_TIMEOUT

Timeout error

HAL_SPDIFRX_ERROR_OVR

OVR error

HAL_SPDIFRX_ERROR_PE

Parity error

HAL_SPDIFRX_ERROR_DMA

DMA transfer error

HAL_SPDIFRX_ERROR_UNKNOWN

Unknown Error error

HAL_SPDIFRX_ERROR_INVALID_CALLBACK

Invalid Callback error

SPDIFRX Exported Macros

__HAL_SPDIFRX_RESET_HANDLE_STATE

Description:

- Reset SPDIFRX handle state.

Parameters:

- __HANDLE__: SPDIFRX handle.

Return value:

- None

__HAL_SPDIFRX_IDLE

Description:

- Disable the specified SPDIFRX peripheral (IDLE State).

Parameters:

- __HANDLE__: specifies the SPDIFRX Handle.

Return value:

- None

__HAL_SPDIFRX_SYNC

Description:

- Enable the specified SPDIFRX peripheral (SYNC State).

Parameters:

- __HANDLE__: specifies the SPDIFRX Handle.

Return value:

- None

__HAL_SPDIFRX_RCV

Description:

- Enable the specified SPDIFRX peripheral (RCV State).

Parameters:

- __HANDLE__: specifies the SPDIFRX Handle.

Return value:

- None

__HAL_SPDIFRX_ENABLE_IT

Description:

- Enable or disable the specified SPDIFRX interrupts.

Parameters:

- __HANDLE__: specifies the SPDIFRX Handle.
- __INTERRUPT__: specifies the interrupt source to enable or disable. This parameter can be one of the following values:
  - SPDIFRX_IT_RXNE
  - SPDIFRX_IT_CSRNE
  - SPDIFRX_IT_PERRIE
  - SPDIFRX_IT_OVRIE
  - SPDIFRX_IT_SBLKIE
  - SPDIFRX_IT_SYNCDIE
  - SPDIFRX_IT_IFEIE

Return value:

- None

__HAL_SPDIFRX_DISABLE_IT

__HAL_SPDIFRX_GET_IT_SOURCE

Description:

- Checks if the specified SPDIFRX interrupt source is enabled or disabled.

Parameters:

- __HANDLE__: specifies the SPDIFRX Handle.
- __INTERRUPT__: specifies the SPDIFRX interrupt source to check. This parameter can be one of the following values:
  - SPDIFRX_IT_RXNE
  - SPDIFRX_IT_CSRNE
  - SPDIFRX_IT_PERRIE
  - SPDIFRX_IT_OVRIE
  - SPDIFRX_IT_SBLKIE
  - SPDIFRX_IT_SYNCDIE
  - SPDIFRX_IT_IFEIE

Return value:

- The: new state of __IT__ (TRUE or FALSE).

__HAL_SPDIFRX_GET_FLAG

Description:

- Checks whether the specified SPDIFRX flag is set or not.

Parameters:

- __HANDLE__: specifies the SPDIFRX Handle.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - SPDIFRX_FLAG_RXNE
  - SPDIFRX_FLAG_CSRNE
  - SPDIFRX_FLAG_PERR
  - SPDIFRX_FLAG_OVR
  - SPDIFRX_FLAG_SBD
  - SPDIFRX_FLAG_SYNCD
  - SPDIFRX_FLAG_FERR
  - SPDIFRX_FLAG_SERR
  - SPDIFRX_FLAG_TERR

Return value:

- The: new state of __FLAG__ (TRUE or FALSE).

__HAL_SPDIFRX_CLEAR_IT

Description:

- Clears the specified SPDIFRX SR flag, in setting the proper IFCR register bit.

Parameters:

- __HANDLE__: specifies the USART Handle.
- __IT_CLEAR__: specifies the interrupt clear register flag that needs to be set to clear the corresponding interrupt This parameter can be one of the following values:
  - SPDIFRX_FLAG_PERR
  - SPDIFRX_FLAG_OVR
  - SPDIFRX_SR_SBD
  - SPDIFRX_SR_SYNCD

Return value:

- None

SPDIFRX Flags Definition

SPDIFRX_FLAG_RXNE

SPDIFRX_FLAG_CSRNE

SPDIFRX_FLAG_PERR

SPDIFRX_FLAG_OVR

SPDIFRX_FLAG_SBD

SPDIFRX_FLAG_SYNCD

SPDIFRX_FLAG_FERR

SPDIFRX_FLAG_SERR

SPDIFRX_FLAG_TERR

SPDIFRX Input Selection

SPDIFRX_INPUT_IN0

SPDIFRX_INPUT_IN1

SPDIFRX_INPUT_IN2

SPDIFRX_INPUT_IN3

SPDIFRX Interrupts Definition

SPDIFRX_IT_RXNE

SPDIFRX_IT_CSRNE

SPDIFRX_IT_PERRIE

SPDIFRX_IT_OVRIE

SPDIFRX_IT_SBLKIE

SPDIFRX_IT_SYNCDIE

SPDIFRX_IT_IFEIE

SPDIFRX Maximum Retries

SPDIFRX_MAXRETRIES_NONE

SPDIFRX_MAXRETRIES_3

SPDIFRX_MAXRETRIES_15

SPDIFRX_MAXRETRIES_63

SPDIFRX Parity Error Mask

SPDIFRX_PARITYERRORMASK_OFF

SPDIFRX_PARITYERRORMASK_ON

SPDIFRX Preamble Type Mask

SPDIFRX_PREAMBLETYPEMASK_OFF

SPDIFRX_PREAMBLETYPEMASK_ON

SPDIFRX State

SPDIFRX_STATE_IDLE

SPDIFRX_STATE_SYNC

SPDIFRX_STATE_RCV

SPDIFRX Stereo Mode

SPDIFRX_STEREOMODE_DISABLE

SPDIFRX_STEREOMODE_ENABLE

SPDIFRX Validity Mask

SPDIFRX_VALIDITYMASK_OFF

SPDIFRX_VALIDITYMASK_ON

SPDIFRX Wait For Activity

SPDIFRX_WAITFORACTIVITY_OFF

SPDIFRX_WAITFORACTIVITY_ON

74 HAL SPI Generic Driver

74.1 SPI Firmware driver registers structures

74.1.1 SPI_InitTypeDef

SPI_InitTypeDef is defined in the stm32n6xx_hal_spi.h

Data Fields

- ***uint32_t Mode***
- ***uint32_t Direction***
- ***uint32_t DataSize***
- ***uint32_t CLKPolarity***
- ***uint32_t CLKPhase***
- ***uint32_t NSS***
- ***uint32_t BaudRatePrescaler***
- ***uint32_t FirstBit***
- ***uint32_t TIMode***
- ***uint32_t CRCCalculation***
- ***uint32_t CRCPolynomial***
- ***uint32_t CRCLength***
- ***uint32_t NSSPMode***
- ***uint32_t NSSPolarity***
- ***uint32_t FifoThreshold***
- ***uint32_t TxCRCInitializationPattern***
- ***uint32_t RxCRCInitializationPattern***
- ***uint32_t MasterSSIdleness***
- ***uint32_t MasterInterDataIdleness***
- ***uint32_t MasterReceiverAutoSusp***
- ***uint32_t MasterKeepIOState***
- ***uint32_t IOSwap***
- ***uint32_t ReadyMasterManagement***
- ***uint32_t ReadyPolarity***

Field Documentation

- ***uint32_t SPI_InitTypeDef::Mode***
  Specifies the SPI operating mode. This parameter can be a value of ***SPI_Mode***
- ***uint32_t SPI_InitTypeDef::Direction***
  Specifies the SPI bidirectional mode state. This parameter can be a value of ***SPI_Direction***
- ***uint32_t SPI_InitTypeDef::DataSize***
  Specifies the SPI data size. This parameter can be a value of ***SPI_Data_Size***
- ***uint32_t SPI_InitTypeDef::CLKPolarity***
  Specifies the serial clock steady state. This parameter can be a value of ***SPI_Clock_Polarity***
- ***uint32_t SPI_InitTypeDef::CLKPhase***
  Specifies the clock active edge for the bit capture. This parameter can be a value of ***SPI_Clock_Phase***
- ***uint32_t SPI_InitTypeDef::NSS***
  Specifies whether the NSS signal is managed by hardware (NSS pin) or by software using the SSI bit. This parameter can be a value of ***SPI_Slave_Select_Management***
- ***uint32_t SPI_InitTypeDef::BaudRatePrescaler***
  Specifies the Baud Rate prescaler value which will be used to configure the transmit and receive SCK clock. This parameter can be a value of ***SPI_BaudRate_Prescaler***
  Note:
  – The communication clock is derived from the master clock. The slave clock does not need to be set.

- ***uint32_t SPI_InitTypeDef::FirstBit***
  Specifies whether data transfers start from MSB or LSB bit. This parameter can be a value of ***SPI_MSB_LSB_Transmission***
- ***uint32_t SPI_InitTypeDef::TIMode***
  Specifies if the TI mode is enabled or not. This parameter can be a value of ***SPI_TI_Mode***
- ***uint32_t SPI_InitTypeDef::CRCCalculation***
  Specifies if the CRC calculation is enabled or not. This parameter can be a value of ***SPI_CRC_Calculation***
- ***uint32_t SPI_InitTypeDef::CRCPolynomial***
  Specifies the polynomial used for the CRC calculation. This parameter must be an odd number between Min_Data = 0 and Max_Data = 65535
- ***uint32_t SPI_InitTypeDef::CRCLength***
  Specifies the CRC Length used for the CRC calculation. This parameter can be a value of ***SPI_CRC_length***
- ***uint32_t SPI_InitTypeDef::NSSPMode***
  Specifies whether the NSSP signal is enabled or not . This parameter can be a value of ***SPI_NSSP_Mode***
  This mode is activated by the SSOM bit in the SPIx_CR2 register and it takes effect only if the SPI interface is configured as Motorola SPI master (FRF=0).
- ***uint32_t SPI_InitTypeDef::NSSPolarity***
  Specifies which level of SS input/output external signal (present on SS pin) is considered as active one. This parameter can be a value of ***SPI_NSS_Polarity***
- ***uint32_t SPI_InitTypeDef::FifoThreshold***
  Specifies the FIFO threshold level. This parameter can be a value of ***SPI_Fifo_Threshold***
- ***uint32_t SPI_InitTypeDef::TxCRCInitializationPattern***
  Specifies the transmitter CRC initialization Pattern used for the CRC calculation. This parameter can be a value of ***SPI_CRC_Calculation_Initialization_Pattern***
- ***uint32_t SPI_InitTypeDef::RxCRCInitializationPattern***
  Specifies the receiver CRC initialization Pattern used for the CRC calculation. This parameter can be a value of ***SPI_CRC_Calculation_Initialization_Pattern***
- ***uint32_t SPI_InitTypeDef::MasterSSIdleness***
  Specifies an extra delay, expressed in number of SPI clock cycle periods, inserted additionally between active edge of SS and first data transaction start in master mode. This parameter can be a value of ***SPI_Master_SS_Idleness***
- ***uint32_t SPI_InitTypeDef::MasterInterDataIdleness***
  Specifies minimum time delay (expressed in SPI clock cycles periods) inserted between two consecutive data frames in master mode. This parameter can be a value of ***SPI_Master_InterData_Idleness***
- ***uint32_t SPI_InitTypeDef::MasterReceiverAutoSusp***
  Control continuous SPI transfer in master receiver mode and automatic management in order to avoid overrun condition. This parameter can be a value of ***SPI_Master_RX_AutoSuspend***
- ***uint32_t SPI_InitTypeDef::MasterKeepIOState***
  Control of Alternate function GPIOs state This parameter can be a value of ***SPI_Master_Keep_IO_State***
- ***uint32_t SPI_InitTypeDef::IOSwap***
  Invert MISO/MOSI alternate functions This parameter can be a value of ***SPI_IO_Swap***
- ***uint32_t SPI_InitTypeDef::ReadyMasterManagement***
  Specifies if RDY Signal is managed internally or not. This parameter can be a value of ***SPI_RDY_Master_Management***
- ***uint32_t SPI_InitTypeDef::ReadyPolarity***
  Specifies which level of RDY Signal input (present on RDY pin) is considered as active one. This parameter can be a value of ***SPI_RDY_Polarity***

74.1.2 __SPI_HandleTypeDef

__SPI_HandleTypeDef is defined in the stm32n6xx_hal_spi.h

Data Fields

- ***SPI_TypeDef * Instance***
- ***SPI_InitTypeDef Init***
- ***const uint8_t * pTxBuffPtr***
- ***uint16_t TxXferSize***

- ***__IO uint16_t TxXferCount***
- ***uint8_t * pRxBuffPtr***
- ***uint16_t RxXferSize***
- ***__IO uint16_t RxXferCount***
- ***uint32_t CRCSize***
- ***void(* RxISR***
- ***void(* TxISR***
- ***DMA_HandleTypeDef * hdmatx***
- ***DMA_HandleTypeDef * hdmarx***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_SPI_StateTypeDef State***
- ***__IO uint32_t ErrorCode***
- ***void(* TxCpltCallback***
- ***void(* RxCpltCallback***
- ***void(* TxRxCpltCallback***
- ***void(* TxHalfCpltCallback***
- ***void(* RxHalfCpltCallback***
- ***void(* TxRxHalfCpltCallback***
- ***void(* ErrorCallback***
- ***void(* AbortCpltCallback***
- ***void(* SuspendCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***SPI_TypeDef* __SPI_HandleTypeDef::Instance***
  SPI registers base address
- ***SPI_InitTypeDef __SPI_HandleTypeDef::Init***
  SPI communication parameters
- ***const uint8_t* __SPI_HandleTypeDef::pTxBuffPtr***
  Pointer to SPI Tx transfer Buffer
- ***uint16_t __SPI_HandleTypeDef::TxXferSize***
  SPI Tx Transfer size
- ***__IO uint16_t __SPI_HandleTypeDef::TxXferCount***
  SPI Tx Transfer Counter
- ***uint8_t* __SPI_HandleTypeDef::pRxBuffPtr***
  Pointer to SPI Rx transfer Buffer
- ***uint16_t __SPI_HandleTypeDef::RxXferSize***
  SPI Rx Transfer size
- ***__IO uint16_t __SPI_HandleTypeDef::RxXferCount***
  SPI Rx Transfer Counter
- ***uint32_t __SPI_HandleTypeDef::CRCSize***
  SPI CRC size used for the transfer
- ***void(* __SPI_HandleTypeDef::RxISR)(struct __SPI_HandleTypeDef *hspi)***
  function pointer on Rx ISR
- ***void(* __SPI_HandleTypeDef::TxISR)(struct __SPI_HandleTypeDef *hspi)***
  function pointer on Tx ISR
- ***DMA_HandleTypeDef* __SPI_HandleTypeDef::hdmatx***
  SPI Tx DMA Handle parameters
- ***DMA_HandleTypeDef* __SPI_HandleTypeDef::hdmarx***
  SPI Rx DMA Handle parameters
- ***HAL_LockTypeDef __SPI_HandleTypeDef::Lock***
  Locking object

- ***__IO HAL_SPI_StateTypeDef __SPI_HandleTypeDef::State***
  SPI communication state
- ***__IO uint32_t __SPI_HandleTypeDef::ErrorCode***
  SPI Error code
- ***void(* __SPI_HandleTypeDef::TxCpltCallback)(struct __SPI_HandleTypeDef *hspi)***
  SPI Tx Completed callback
- ***void(* __SPI_HandleTypeDef::RxCpltCallback)(struct __SPI_HandleTypeDef *hspi)***
  SPI Rx Completed callback
- ***void(* __SPI_HandleTypeDef::TxRxCpltCallback)(struct __SPI_HandleTypeDef *hspi)***
  SPI TxRx Completed callback
- ***void(* __SPI_HandleTypeDef::TxHalfCpltCallback)(struct __SPI_HandleTypeDef *hspi)***
  SPI Tx Half Completed callback
- ***void(* __SPI_HandleTypeDef::RxHalfCpltCallback)(struct __SPI_HandleTypeDef *hspi)***
  SPI Rx Half Completed callback
- ***void(* __SPI_HandleTypeDef::TxRxHalfCpltCallback)(struct __SPI_HandleTypeDef *hspi)***
  SPI TxRx Half Completed callback
- ***void(* __SPI_HandleTypeDef::ErrorCallback)(struct __SPI_HandleTypeDef *hspi)***
  SPI Error callback
- ***void(* __SPI_HandleTypeDef::AbortCpltCallback)(struct __SPI_HandleTypeDef *hspi)***
  SPI Abort callback
- ***void(* __SPI_HandleTypeDef::SuspendCallback)(struct __SPI_HandleTypeDef *hspi)***
  SPI Suspend callback
- ***void(* __SPI_HandleTypeDef::MspInitCallback)(struct __SPI_HandleTypeDef *hspi)***
  SPI Msp Init callback
- ***void(* __SPI_HandleTypeDef::MspDeInitCallback)(struct __SPI_HandleTypeDef *hspi)***
  SPI Msp DeInit callback

74.2 SPI Firmware driver API description

The following section lists the various functions of the SPI library.

74.2.1 How to use this driver

The SPI HAL driver can be used as follows:

1. Declare a SPI_HandleTypeDef handle structure, for example: SPI_HandleTypeDef hspi;
2. Initialize the SPI low level resources by implementing the HAL_SPI_MspInit() API:
   a. Enable the SPIx interface clock
   b. SPI pins configuration
      - Enable the clock for the SPI GPIOs
      - Configure these SPI pins as alternate function push-pull
   c. NVIC configuration if you need to use interrupt process or DMA process
      - Configure the SPIx interrupt priority
      - Enable the NVIC SPI IRQ handle
   d. DMA Configuration if you need to use DMA process
      - Declare a DMA_HandleTypeDef handle structure for the transmit or receive Stream/Channel
      - Enable the DMAx clock
      - Configure the DMA handle parameters
      - Configure the DMA Tx or Rx Stream/Channel
      - Associate the initialized hdma_tx handle to the hspi DMA Tx or Rx handle
      - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx or Rx Stream/Channel
3. Program the Mode, BidirectionalMode , Data size, Baudrate Prescaler, NSS management, Clock polarity and phase, FirstBit and CRC configuration in the hspi Init structure.

4. Initialize the SPI registers by calling the HAL_SPI_Init() API:
   – This API configures also the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customized HAL_SPI_MspInit() API.

Callback registration:

1. The compilation flag USE_HAL_SPI_REGISTER_CALLBACKS when set to 1UL allows the user to configure dynamically the driver callbacks. Use Functions HAL_SPI_RegisterCallback() to register an interrupt callback. Function HAL_SPI_RegisterCallback() allows to register following callbacks:
   – TxCpltCallback : SPI Tx Completed callback
   – RxCpltCallback : SPI Rx Completed callback
   – TxRxCpltCallback : SPI TxRx Completed callback
   – TxHalfCpltCallback : SPI Tx Half Completed callback
   – RxHalfCpltCallback : SPI Rx Half Completed callback
   – TxRxHalfCpltCallback : SPI TxRx Half Completed callback
   – ErrorCallback : SPI Error callback
   – AbortCpltCallback : SPI Abort callback
   – SuspendCallback : SPI Suspend callback
   – MspInitCallback : SPI Msp Init callback
   – MspDeInitCallback : SPI Msp DeInit callback This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.
2. Use function HAL_SPI_UnRegisterCallback to reset a callback to the default weak function. HAL_SPI_UnRegisterCallback takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks:
   – TxCpltCallback : SPI Tx Completed callback
   – RxCpltCallback : SPI Rx Completed callback
   – TxRxCpltCallback : SPI TxRx Completed callback
   – TxHalfCpltCallback : SPI Tx Half Completed callback
   – RxHalfCpltCallback : SPI Rx Half Completed callback
   – TxRxHalfCpltCallback : SPI TxRx Half Completed callback
   – ErrorCallback : SPI Error callback
   – AbortCpltCallback : SPI Abort callback
   – SuspendCallback : SPI Suspend callback
   – MspInitCallback : SPI Msp Init callback
   – MspDeInitCallback : SPI Msp DeInit callback
3. By default, after the HAL_SPI_Init() and when the state is HAL_SPI_STATE_RESET all callbacks are set to the corresponding weak functions: examples HAL_SPI_MasterTxCpltCallback(), HAL_SPI_MasterRxCpltCallback(). Exception done for MspInit and MspDeInit functions that are reset to the legacy weak functions in the HAL_SPI_Init()/ HAL_SPI_DeInit() only when these callbacks are null (not registered beforehand). If MspInit or MspDeInit are not null, the HAL_SPI_Init()/ HAL_SPI_DeInit() keep and use the user MspInit/MspDeInit callbacks (registered beforehand) whatever the state.
4. Callbacks can be registered/unregistered in HAL_SPI_STATE_READY state only. Exception done MspInit/ MspDeInit functions that can be registered/unregistered in HAL_SPI_STATE_READY or HAL_SPI_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/ DeInit. Then, the user first registers the MspInit/MspDeInit user callbacks using HAL_SPI_RegisterCallback() before calling HAL_SPI_DeInit() or HAL_SPI_Init() function.
5. When The compilation define USE_HAL_PPP_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak callbacks are used.
6. SuspendCallback restriction: SuspendCallback is called only when MasterReceiverAutoSusp is enabled and EOT interrupt is activated. SuspendCallback is used in relation with functions HAL_SPI_Transmit_IT, HAL_SPI_Receive_IT and HAL_SPI_TransmitReceive_IT.

Circular mode restriction:

- The DMA circular mode cannot be used when the SPI is configured in these modes:
  – Master 2Lines RxOnly
  – Master 1Line Rx

- The CRC feature is not managed when the DMA circular mode is enabled
- The functions HAL_SPI_DMAPause()/ HAL_SPI_DMAResume() are not supported. Return always HAL_ERROR with ErrorCode set to HAL_SPI_ERROR_NOT_SUPPORTED. Those functions are maintained for backward compatibility reasons.

74.2.2 Detailed description of functions

HAL_SPI_Init

Function name

HAL_StatusTypeDef HAL_SPI_Init (SPI_HandleTypeDef * hspi)

Function description

HAL_SPI_DeInit

Function name

HAL_StatusTypeDef HAL_SPI_DeInit (SPI_HandleTypeDef * hspi)

Function description

HAL_SPI_MspInit

Function name

void HAL_SPI_MspInit (SPI_HandleTypeDef * hspi)

Function description

HAL_SPI_MspDeInit

Function name

void HAL_SPI_MspDeInit (SPI_HandleTypeDef * hspi)

Function description

HAL_SPI_RegisterCallback

Function name

HAL_StatusTypeDef HAL_SPI_RegisterCallback (SPI_HandleTypeDef * hspi, HAL_SPI_CallbackIDTypeDef CallbackID, pSPI_CallbackTypeDef pCallback)

Function description

HAL_SPI_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_SPI_UnRegisterCallback (SPI_HandleTypeDef * hspi, HAL_SPI_CallbackIDTypeDef CallbackID)

Function description

HAL_SPI_Transmit

Function name

HAL_StatusTypeDef HAL_SPI_Transmit (SPI_HandleTypeDef * hspi, const uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

HAL_SPI_Receive

Function name

HAL_StatusTypeDef HAL_SPI_Receive (SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

HAL_SPI_TransmitReceive

Function name

HAL_StatusTypeDef HAL_SPI_TransmitReceive (SPI_HandleTypeDef * hspi, const uint8_t * pTxData, uint8_t * pRxData, uint16_t Size, uint32_t Timeout)

Function description

HAL_SPI_Transmit_IT

Function name

HAL_StatusTypeDef HAL_SPI_Transmit_IT (SPI_HandleTypeDef * hspi, const uint8_t * pData, uint16_t Size)

Function description

HAL_SPI_Receive_IT

Function name

HAL_StatusTypeDef HAL_SPI_Receive_IT (SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size)

Function description

HAL_SPI_TransmitReceive_IT

Function name

HAL_StatusTypeDef HAL_SPI_TransmitReceive_IT (SPI_HandleTypeDef * hspi, const uint8_t * pTxData, uint8_t * pRxData, uint16_t Size)

Function description

HAL_SPI_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_SPI_Transmit_DMA (SPI_HandleTypeDef * hspi, const uint8_t * pData, uint16_t Size)

Function description

HAL_SPI_Receive_DMA

Function name

HAL_StatusTypeDef HAL_SPI_Receive_DMA (SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size)

Function description

HAL_SPI_TransmitReceive_DMA

Function name

HAL_StatusTypeDef HAL_SPI_TransmitReceive_DMA (SPI_HandleTypeDef * hspi, const uint8_t * pTxData, uint8_t * pRxData, uint16_t Size)

Function description

HAL_SPI_DMAPause

Function name

HAL_StatusTypeDef HAL_SPI_DMAPause (SPI_HandleTypeDef * hspi)

Function description

HAL_SPI_DMAResume

Function name

HAL_StatusTypeDef HAL_SPI_DMAResume (SPI_HandleTypeDef * hspi)

Function description

HAL_SPI_DMAStop

Function name

HAL_StatusTypeDef HAL_SPI_DMAStop (SPI_HandleTypeDef * hspi)

Function description

HAL_SPI_Abort

Function name

HAL_StatusTypeDef HAL_SPI_Abort (SPI_HandleTypeDef * hspi)

Function description

HAL_SPI_Abort_IT

Function name

HAL_StatusTypeDef HAL_SPI_Abort_IT (SPI_HandleTypeDef * hspi)

Function description

HAL_SPI_IRQHandler

Function name

void HAL_SPI_IRQHandler (SPI_HandleTypeDef * hspi)

Function description

HAL_SPI_TxCpltCallback

Function name

void HAL_SPI_TxCpltCallback (SPI_HandleTypeDef * hspi)

Function description

HAL_SPI_RxCpltCallback

Function name

void HAL_SPI_RxCpltCallback (SPI_HandleTypeDef * hspi)

Function description

HAL_SPI_TxRxCpltCallback

Function name

void HAL_SPI_TxRxCpltCallback (SPI_HandleTypeDef * hspi)

Function description

HAL_SPI_TxHalfCpltCallback

Function name

void HAL_SPI_TxHalfCpltCallback (SPI_HandleTypeDef * hspi)

Function description

HAL_SPI_RxHalfCpltCallback

Function name

void HAL_SPI_RxHalfCpltCallback (SPI_HandleTypeDef * hspi)

Function description

HAL_SPI_TxRxHalfCpltCallback

Function name

void HAL_SPI_TxRxHalfCpltCallback (SPI_HandleTypeDef * hspi)

Function description

HAL_SPI_ErrorCallback

Function name

void HAL_SPI_ErrorCallback (SPI_HandleTypeDef * hspi)

Function description

HAL_SPI_AbortCpltCallback

Function name

void HAL_SPI_AbortCpltCallback (SPI_HandleTypeDef * hspi)

Function description

HAL_SPI_SuspendCallback

Function name

void HAL_SPI_SuspendCallback (SPI_HandleTypeDef * hspi)

Function description

HAL_SPI_GetState

Function name

HAL_SPI_StateTypeDef HAL_SPI_GetState (const SPI_HandleTypeDef * hspi)

Function description

HAL_SPI_GetError

Function name

uint32_t HAL_SPI_GetError (const SPI_HandleTypeDef * hspi)

Function description

74.3 SPI Firmware driver defines

The following section lists the various define and macros of the module.

74.3.1 SPI

SPI

SPI BaudRate Prescaler

SPI_BAUDRATEPRESCALER_BYPASS

SPI_BAUDRATEPRESCALER_2

SPI_BAUDRATEPRESCALER_4

SPI_BAUDRATEPRESCALER_8

SPI_BAUDRATEPRESCALER_16

SPI_BAUDRATEPRESCALER_32

SPI_BAUDRATEPRESCALER_64

SPI_BAUDRATEPRESCALER_128

SPI_BAUDRATEPRESCALER_256

SPI Clock Phase

SPI_PHASE_1EDGE

SPI_PHASE_2EDGE

SPI Clock Polarity

SPI_POLARITY_LOW

SPI_POLARITY_HIGH

SPI CRC Calculation

SPI_CRCCALCULATION_DISABLE

SPI_CRCCALCULATION_ENABLE

SPI CRC Calculation Initialization Pattern

SPI_CRC_INITIALIZATION_ALL_ZERO_PATTERN

SPI_CRC_INITIALIZATION_ALL_ONE_PATTERN

SPI CRC Length

SPI_CRC_LENGTH_DATASIZE

SPI_CRC_LENGTH_4BIT

SPI_CRC_LENGTH_5BIT

SPI_CRC_LENGTH_6BIT

SPI_CRC_LENGTH_7BIT

SPI_CRC_LENGTH_8BIT

SPI_CRC_LENGTH_9BIT

SPI_CRC_LENGTH_10BIT

SPI_CRC_LENGTH_11BIT

SPI_CRC_LENGTH_12BIT

SPI_CRC_LENGTH_13BIT

SPI_CRC_LENGTH_14BIT

SPI_CRC_LENGTH_15BIT

SPI_CRC_LENGTH_16BIT

SPI_CRC_LENGTH_17BIT

SPI_CRC_LENGTH_18BIT

SPI_CRC_LENGTH_19BIT

SPI_CRC_LENGTH_20BIT

SPI_CRC_LENGTH_21BIT

SPI_CRC_LENGTH_22BIT

SPI_CRC_LENGTH_23BIT

SPI_CRC_LENGTH_24BIT

SPI_CRC_LENGTH_25BIT

SPI_CRC_LENGTH_26BIT

SPI_CRC_LENGTH_27BIT

SPI_CRC_LENGTH_28BIT

SPI_CRC_LENGTH_29BIT

SPI_CRC_LENGTH_30BIT

SPI_CRC_LENGTH_31BIT

SPI_CRC_LENGTH_32BIT

SPI Data Size

SPI_DATASIZE_4BIT

SPI_DATASIZE_5BIT

SPI_DATASIZE_6BIT

SPI_DATASIZE_7BIT

SPI_DATASIZE_8BIT

SPI_DATASIZE_9BIT

SPI_DATASIZE_10BIT

SPI_DATASIZE_11BIT

SPI_DATASIZE_12BIT

SPI_DATASIZE_13BIT

SPI_DATASIZE_14BIT

SPI_DATASIZE_15BIT

SPI_DATASIZE_16BIT

SPI_DATASIZE_17BIT

SPI_DATASIZE_18BIT

SPI_DATASIZE_19BIT

SPI_DATASIZE_20BIT

SPI_DATASIZE_21BIT

SPI_DATASIZE_22BIT

SPI_DATASIZE_23BIT

SPI_DATASIZE_24BIT

SPI_DATASIZE_25BIT

SPI_DATASIZE_26BIT

SPI_DATASIZE_27BIT

SPI_DATASIZE_28BIT

SPI_DATASIZE_29BIT

SPI_DATASIZE_30BIT

SPI_DATASIZE_31BIT

SPI_DATASIZE_32BIT

SPI Direction Mode

SPI_DIRECTION_2LINES

SPI_DIRECTION_2LINES_TXONLY

SPI_DIRECTION_2LINES_RXONLY

SPI_DIRECTION_1LINE

SPI Error Codes

HAL_SPI_ERROR_NONE
: No error

HAL_SPI_ERROR_MODF
: MODF error

HAL_SPI_ERROR_CRC
: CRC error

HAL_SPI_ERROR_OVR
: OVR error

HAL_SPI_ERROR_FRE
: FRE error

HAL_SPI_ERROR_DMA
: DMA transfer error

HAL_SPI_ERROR_FLAG
: Error on RXP/TXP/DXP/FTLVL/FRLVL Flag

HAL_SPI_ERROR_ABORT
: Error during SPI Abort procedure

HAL_SPI_ERROR_UDR
: Underrun error

HAL_SPI_ERROR_TIMEOUT

Timeout error

HAL_SPI_ERROR_UNKNOW

Unknown error

HAL_SPI_ERROR_NOT_SUPPORTED

Requested operation not supported

HAL_SPI_ERROR_INVALID_CALLBACK

Invalid Callback error

SPI Exported Macros

__HAL_SPI_RESET_HANDLE_STATE

Description:

- Reset SPI handle state.

Parameters:

- __HANDLE__: specifies the SPI Handle. This parameter can be SPI where x: 1, 2, 3, 4, 5 or 6 to select the SPI peripheral.

Return value:

- None

__HAL_SPI_ENABLE_IT

Description:

- Enable the specified SPI interrupts.

Parameters:

- __HANDLE__: specifies the SPI Handle. This parameter can be SPI where x: 1, 2, 3, 4, 5 or 6 to select the SPI peripheral.
- __INTERRUPT__: specifies the interrupt source to enable or disable. This parameter can be one of the following values:
  - SPI_IT_RXP : Rx-Packet available interrupt
  - SPI_IT_TXP : Tx-Packet space available interrupt
  - SPI_IT_DXP : Duplex Packet interrupt
  - SPI_IT_EOT : End of transfer interrupt
  - SPI_IT_TXTF : Transmission Transfer Filled interrupt
  - SPI_IT_UDR : Underrun interrupt
  - SPI_IT_OVR : Overrun interrupt
  - SPI_IT_CRCERR : CRC error interrupt
  - SPI_IT_FRE : TI mode frame format error interrupt
  - SPI_IT_MODF : Mode fault interrupt
  - SPI_IT_ERR : Error interrupt

Return value:

- None

__HAL_SPI_DISABLE_IT

Description:

- Disable the specified SPI interrupts.

Parameters:

- __HANDLE__: specifies the SPI Handle. This parameter can be SPI where x: 1, 2, 3, 4, 5 or 6 to select the SPI peripheral.
- __INTERRUPT__: specifies the interrupt source to enable or disable. This parameter can be one of the following values:
  - SPI_IT_RXP : Rx-Packet available interrupt
  - SPI_IT_TXP : Tx-Packet space available interrupt
  - SPI_IT_DXP : Duplex Packet interrupt
  - SPI_IT_EOT : End of transfer interrupt
  - SPI_IT_TXTF : Transmission Transfer Filled interrupt
  - SPI_IT_UDR : Underrun interrupt
  - SPI_IT_OVR : Overrun interrupt
  - SPI_IT_CRCERR : CRC error interrupt
  - SPI_IT_FRE : TI mode frame format error interrupt
  - SPI_IT_MODF : Mode fault interrupt
  - SPI_IT_ERR : Error interrupt

Return value:

- None

__HAL_SPI_GET_IT_SOURCE

Description:

- Check whether the specified SPI interrupt source is enabled or not.

Parameters:

- __HANDLE__: specifies the SPI Handle. This parameter can be SPI where x: 1, 2, 3, 4, 5 or 6 to select the SPI peripheral.
- __INTERRUPT__: specifies the SPI interrupt source to check. This parameter can be one of the following values:
  - SPI_IT_RXP : Rx-Packet available interrupt
  - SPI_IT_TXP : Tx-Packet space available interrupt
  - SPI_IT_DXP : Duplex Packet interrupt
  - SPI_IT_EOT : End of transfer interrupt
  - SPI_IT_TXTF : Transmission Transfer Filled interrupt
  - SPI_IT_UDR : Underrun interrupt
  - SPI_IT_OVR : Overrun interrupt
  - SPI_IT_CRCERR : CRC error interrupt
  - SPI_IT_FRE : TI mode frame format error interrupt
  - SPI_IT_MODF : Mode fault interrupt
  - SPI_IT_ERR : Error interrupt

Return value:

- The: new state of __IT__ (TRUE or FALSE).

__HAL_SPI_GET_FLAG

Description:

- Check whether the specified SPI flag is set or not.

Parameters:

- __HANDLE__: specifies the SPI Handle. This parameter can be SPI where x: 1, 2, 3, 4, 5 or 6 to select the SPI peripheral.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - SPI_FLAG_RXP : Rx-Packet available flag
  - SPI_FLAG_TXP : Tx-Packet space available flag
  - SPI_FLAG_DXP : Duplex Packet flag
  - SPI_FLAG_EOT : End of transfer flag
  - SPI_FLAG_TXTF : Transmission Transfer Filled flag
  - SPI_FLAG_UDR : Underrun flag
  - SPI_FLAG_OVR : Overrun flag
  - SPI_FLAG_CRCERR : CRC error flag
  - SPI_FLAG_FRE : TI mode frame format error flag
  - SPI_FLAG_MODF : Mode fault flag
  - SPI_FLAG_SUSP : Transfer suspend complete flag
  - SPI_FLAG_TXC : TxFIFO transmission complete flag
  - SPI_FLAG_FRLVL : Fifo reception level flag
  - SPI_FLAG_RXWNE : RxFIFO word not empty flag

Return value:

- The: new state of __FLAG__ (TRUE or FALSE).

__HAL_SPI_CLEAR_CRCERRFLAG

Description:

- Clear the SPI CRCERR pending flag.

Parameters:

- __HANDLE__: specifies the SPI Handle.

Return value:

- None

__HAL_SPI_CLEAR_MODFFLAG

Description:

- Clear the SPI MODF pending flag.

Parameters:

- __HANDLE__: specifies the SPI Handle.

Return value:

- None

__HAL_SPI_CLEAR_OVRFLAG

Description:

- Clear the SPI OVR pending flag.

Parameters:

- __HANDLE__: specifies the SPI Handle.

Return value:

- None

__HAL_SPI_CLEAR_FREFLAG

Description:

- Clear the SPI FRE pending flag.

Parameters:

- __HANDLE__: specifies the SPI Handle.

Return value:

- None

__HAL_SPI_CLEAR_UDRFLAG

Description:

- Clear the SPI UDR pending flag.

Parameters:

- __HANDLE__: specifies the SPI Handle.

Return value:

- None

__HAL_SPI_CLEAR_EOTFLAG

Description:

- Clear the SPI EOT pending flag.

Parameters:

- __HANDLE__: specifies the SPI Handle.

Return value:

- None

__HAL_SPI_CLEAR_TXTFFLAG

Description:

- Clear the SPI UDR pending flag.

Parameters:

- __HANDLE__: specifies the SPI Handle.

Return value:

- None

__HAL_SPI_CLEAR_SUSPFLAG

Description:

- Clear the SPI SUSP pending flag.

Parameters:

- __HANDLE__: specifies the SPI Handle.

Return value:

- None

__HAL_SPI_ENABLE

Description:

- Enable the SPI peripheral.

Parameters:

- __HANDLE__: specifies the SPI Handle.

Return value:

- None

__HAL_SPI_DISABLE

Description:

- Disable the SPI peripheral.

Parameters:

- __HANDLE__: specifies the SPI Handle.

Return value:

- None

SPI Fifo Threshold

SPI_FIFO_THRESHOLD_01DATA

SPI_FIFO_THRESHOLD_02DATA

SPI_FIFO_THRESHOLD_03DATA

SPI_FIFO_THRESHOLD_04DATA

SPI_FIFO_THRESHOLD_05DATA

SPI_FIFO_THRESHOLD_06DATA

SPI_FIFO_THRESHOLD_07DATA

SPI_FIFO_THRESHOLD_08DATA

SPI_FIFO_THRESHOLD_09DATA

SPI_FIFO_THRESHOLD_10DATA

SPI_FIFO_THRESHOLD_11DATA

SPI_FIFO_THRESHOLD_12DATA

SPI_FIFO_THRESHOLD_13DATA

SPI_FIFO_THRESHOLD_14DATA

SPI_FIFO_THRESHOLD_15DATA

SPI_FIFO_THRESHOLD_16DATA

SPI FIFO Type

SPI_LOWEND_FIFO_SIZE

SPI_HIGHEND_FIFO_SIZE

SPI Flags Definition

SPI_FLAG_RXP

SPI_FLAG_TXP

SPI_FLAG_DXP

SPI_FLAG_EOT

SPI_FLAG_TXTF

SPI_FLAG_UDR

SPI_FLAG_OVR

SPI_FLAG_CRCERR

SPI_FLAG_FRE

SPI_FLAG_MODF

SPI_FLAG_SUSP

SPI_FLAG_TXC

SPI_FLAG_FRLVL

SPI_FLAG_RXWNE

SPI Interrupt Definition

SPI_IT_RXP

SPI_IT_TXP

SPI_IT_DXP

SPI_IT_EOT

SPI_IT_TXTF

SPI_IT_UDR

SPI_IT_OVR

SPI_IT_CRCERR

SPI_IT_FRE

SPI_IT_MODF

SPI_IT_ERR

Control SPI IO Swap

SPI_IO_SWAP_DISABLE

SPI_IO_SWAP_ENABLE

SPI Master Inter-Data Idleness

SPI_MASTER_INTERDATA_IDLENESS_00CYCLE

SPI_MASTER_INTERDATA_IDLENESS_01CYCLE

SPI_MASTER_INTERDATA_IDLENESS_02CYCLE

SPI_MASTER_INTERDATA_IDLENESS_03CYCLE

SPI_MASTER_INTERDATA_IDLENESS_04CYCLE

SPI_MASTER_INTERDATA_IDLENESS_05CYCLE

SPI_MASTER_INTERDATA_IDLENESS_06CYCLE

SPI_MASTER_INTERDATA_IDLENESS_07CYCLE

SPI_MASTER_INTERDATA_IDLENESS_08CYCLE

SPI_MASTER_INTERDATA_IDLENESS_09CYCLE

SPI_MASTER_INTERDATA_IDLENESS_10CYCLE

SPI_MASTER_INTERDATA_IDLENESS_11CYCLE

SPI_MASTER_INTERDATA_IDLENESS_12CYCLE

SPI_MASTER_INTERDATA_IDLENESS_13CYCLE

SPI_MASTER_INTERDATA_IDLENESS_14CYCLE

SPI_MASTER_INTERDATA_IDLENESS_15CYCLE

Keep IO State

SPI_MASTER_KEEP_IO_STATE_DISABLE

SPI_MASTER_KEEP_IO_STATE_ENABLE

SPI Master Receiver AutoSuspend

SPI_MASTER_RX_AUTOSUSP_DISABLE

SPI_MASTER_RX_AUTOSUSP_ENABLE

SPI Master SS Idleness

SPI_MASTER_SS_IDLENESS_00CYCLE

SPI_MASTER_SS_IDLENESS_01CYCLE

SPI_MASTER_SS_IDLENESS_02CYCLE

SPI_MASTER_SS_IDLENESS_03CYCLE

SPI_MASTER_SS_IDLENESS_04CYCLE

SPI_MASTER_SS_IDLENESS_05CYCLE

SPI_MASTER_SS_IDLENESS_06CYCLE

SPI_MASTER_SS_IDLENESS_07CYCLE

SPI_MASTER_SS_IDLENESS_08CYCLE

SPI_MASTER_SS_IDLENESS_09CYCLE

SPI_MASTER_SS_IDLENESS_10CYCLE

SPI_MASTER_SS_IDLENESS_11CYCLE

SPI_MASTER_SS_IDLENESS_12CYCLE

SPI_MASTER_SS_IDLENESS_13CYCLE

SPI_MASTER_SS_IDLENESS_14CYCLE

SPI_MASTER_SS_IDLENESS_15CYCLE

SPI Mode

SPI_MODE_SLAVE

SPI_MODE_MASTER

SPI MSB LSB Transmission

SPI_FIRSTBIT_MSB

SPI_FIRSTBIT_LSB

SPI NSS Pulse Mode

SPI_NSS_PULSE_DISABLE

SPI_NSS_PULSE_ENABLE

SPI NSS Polarity

SPI_NSS_POLARITY_LOW

SPI_NSS_POLARITY_HIGH

SPI RDY Signal Input Master Management

SPI_RDY_MASTER_MANAGEMENT_INTERNALLY

SPI_RDY_MASTER_MANAGEMENT_EXTERNALLY

SPI RDY Signal Input/Output Polarity

SPI_RDY_POLARITY_HIGH

SPI_RDY_POLARITY_LOW

SPI Reception FIFO Status Level

SPI_RX_FIFO_0PACKET

SPI_RX_FIFO_1PACKET

SPI_RX_FIFO_2PACKET

SPI_RX_FIFO_3PACKET

SPI Slave Select Management

SPI_NSS_SOFT

SPI_NSS_HARD_INPUT

SPI_NSS_HARD_OUTPUT

SPI TI Mode

SPI_TIMODE_DISABLE

SPI_TIMODE_ENABLE

SPI Underrun Behavior

SPI_UNDERRUN_BEHAV_REGISTER_PATTERN

SPI_UNDERRUN_BEHAV_LAST_RECEIVED

75 HAL SPI Extension Driver

75.1 SPIEx Firmware driver API description

The following section lists the various functions of the SPIEx library.

75.1.1 Detailed description of functions

HAL_SPIEx_FlushRxFifo

Function name

HAL_StatusTypeDef HAL_SPIEx_FlushRxFifo (const SPI_HandleTypeDef * hspi)

Function description

HAL_SPIEx_EnableLockConfiguration

Function name

HAL_StatusTypeDef HAL_SPIEx_EnableLockConfiguration (SPI_HandleTypeDef * hspi)

Function description

HAL_SPIEx_ConfigureUnderrun

Function name

HAL_StatusTypeDef HAL_SPIEx_ConfigureUnderrun (SPI_HandleTypeDef * hspi, uint32_t UnderrunDetection, uint32_t UnderrunBehaviour)

Function description

76 HAL SRAM Generic Driver

76.1 SRAM Firmware driver registers structures

76.1.1 __SRAM_HandleTypeDef

__SRAM_HandleTypeDef is defined in the stm32n6xx_hal_sram.h

Data Fields

- ***FMC_NORSRAM_TypeDef * Instance***
- ***FMC_NORSRAM_EXTENDED_TypeDef * Extended***
- ***FMC_NORSRAM_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_SRAM_StateTypeDef State***
- ***DMA_HandleTypeDef * hdma***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***
- ***void(* DmaXferCpltCallback***
- ***void(* DmaXferErrorCallback***

Field Documentation

- ***FMC_NORSRAM_TypeDef* __SRAM_HandleTypeDef::Instance***
  Register base address
- ***FMC_NORSRAM_EXTENDED_TypeDef* __SRAM_HandleTypeDef::Extended***
  Extended mode register base address
- ***FMC_NORSRAM_InitTypeDef __SRAM_HandleTypeDef::Init***
  SRAM device control configuration parameters
- ***HAL_LockTypeDef __SRAM_HandleTypeDef::Lock***
  SRAM locking object
- ***__IO HAL_SRAM_StateTypeDef __SRAM_HandleTypeDef::State***
  SRAM device access state
- ***DMA_HandleTypeDef* __SRAM_HandleTypeDef::hdma***
  Pointer DMA handler
- ***void(* __SRAM_HandleTypeDef::MspInitCallback)(struct __SRAM_HandleTypeDef *hsram)***
  SRAM Msp Init callback
- ***void(* __SRAM_HandleTypeDef::MspDeInitCallback)(struct __SRAM_HandleTypeDef *hsram)***
  SRAM Msp DeInit callback
- ***void(* __SRAM_HandleTypeDef::DmaXferCpltCallback)(DMA_HandleTypeDef *hdma)***
  SRAM DMA Xfer Complete callback
- ***void(* __SRAM_HandleTypeDef::DmaXferErrorCallback)(DMA_HandleTypeDef *hdma)***
  SRAM DMA Xfer Error callback

76.2 SRAM Firmware driver API description

The following section lists the various functions of the SRAM library.

76.2.1 How to use this driver

This driver is a generic layered driver which contains a set of APIs used to control SRAM memories. It uses the FMC layer functions to interface with SRAM devices. The following sequence should be followed to configure the FMC to interface with SRAM/PSRAM memories:

1. Declare a SRAM_HandleTypeDef handle structure, for example: SRAM_HandleTypeDef hsram; and:
   – Fill the SRAM_HandleTypeDef handle "Init" field with the allowed values of the structure member.
   – Fill the SRAM_HandleTypeDef handle "Instance" field with a predefined base register instance for NOR or SRAM device
   – Fill the SRAM_HandleTypeDef handle "Extended" field with a predefined base register instance for NOR or SRAM extended mode
2. Declare two FMC_NORSRAM_TimingTypeDef structures, for both normal and extended mode timings; for example: FMC_NORSRAM_TimingTypeDef Timing and FMC_NORSRAM_TimingTypeDef ExTiming; and fill its fields with the allowed values of the structure member.
3. Initialize the SRAM Controller by calling the function HAL_SRAM_Init(). This function performs the following sequence:
   a. MSP hardware layer configuration using the function HAL_SRAM_MspInit()
   b. Control register configuration using the FMC NORSRAM interface function FMC_NORSRAM_Init()
   c. Timing register configuration using the FMC NORSRAM interface function FMC_NORSRAM_Timing_Init()
   d. Extended mode Timing register configuration using the FMC NORSRAM interface function FMC_NORSRAM_Extended_Timing_Init()
   e. Enable the SRAM device using the macro __FMC_NORSRAM_ENABLE()
4. At this stage you can perform read/write accesses from/to the memory connected to the NOR/SRAM Bank. You can perform either polling or DMA transfer using the following APIs:
   – HAL_SRAM_Read()/HAL_SRAM_Write() for polling read/write access
   – HAL_SRAM_Read_DMA()/HAL_SRAM_Write_DMA() for DMA read/write transfer
5. You can also control the SRAM device by calling the control APIs HAL_SRAM_WriteOperation_Enable()/ HAL_SRAM_WriteOperation_Disable() to respectively enable/disable the SRAM write operation
6. You can continuously monitor the SRAM device HAL state by calling the function HAL_SRAM_GetState()

Callback registration

The compilation define USE_HAL_SRAM_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use Functions HAL_SRAM_RegisterCallback() to register a user callback, it allows to register following callbacks:

- MspInitCallback : SRAM MspInit.
- MspDeInitCallback : SRAM MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function. Use function HAL_SRAM_UnRegisterCallback() to reset a callback to the default weak (overridden) function. It allows to reset following callbacks:
- MspInitCallback : SRAM MspInit.
- MspDeInitCallback : SRAM MspDeInit. This function) takes as parameters the HAL peripheral handle and the Callback ID. By default, after the HAL_SRAM_Init and if the state is HAL_SRAM_STATE_RESET all callbacks are reset to the corresponding legacy weak (overridden) functions. Exception done for MspInit and MspDeInit callbacks that are respectively reset to the legacy weak (overridden) functions in the HAL_SRAM_Init and HAL_SRAM_DeInit only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the HAL_SRAM_Init and HAL_SRAM_DeInit keep and use the user MspInit/MspDeInit callbacks (registered beforehand) Callbacks can be registered/unregistered in READY state only. Exception done for MspInit/MspDeInit callbacks that can be registered/unregistered in READY or RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_SRAM_RegisterCallback before calling HAL_SRAM_DeInit or HAL_SRAM_Init function. When The compilation define USE_HAL_SRAM_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak (overridden) callbacks are used.

76.2.2 Detailed description of functions

HAL_SRAM_Init

Function name

HAL_StatusTypeDef HAL_SRAM_Init (SRAM_HandleTypeDef * hsram, FMC_NORSRAM_TimingTypeDef * Timing, FMC_NORSRAM_TimingTypeDef * ExtTiming)

Function description

HAL_SRAM_DeInit

Function name

HAL_StatusTypeDef HAL_SRAM_DeInit (SRAM_HandleTypeDef * hsram)

Function description

HAL_SRAM_MspInit

Function name

void HAL_SRAM_MspInit (SRAM_HandleTypeDef * hsram)

Function description

HAL_SRAM_MspDeInit

Function name

void HAL_SRAM_MspDeInit (SRAM_HandleTypeDef * hsram)

Function description

HAL_SRAM_Read_8b

Function name

HAL_StatusTypeDef HAL_SRAM_Read_8b (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint8_t * pDstBuffer, uint32_t BufferSize)

Function description

HAL_SRAM_Write_8b

Function name

HAL_StatusTypeDef HAL_SRAM_Write_8b (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint8_t * pSrcBuffer, uint32_t BufferSize)

Function description

HAL_SRAM_Read_16b

Function name

HAL_StatusTypeDef HAL_SRAM_Read_16b (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint16_t * pDstBuffer, uint32_t BufferSize)

Function description

HAL_SRAM_Write_16b

Function name

HAL_StatusTypeDef HAL_SRAM_Write_16b (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint16_t * pSrcBuffer, uint32_t BufferSize)

Function description

HAL_SRAM_Read_32b

Function name

HAL_StatusTypeDef HAL_SRAM_Read_32b (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint32_t * pDstBuffer, uint32_t BufferSize)

Function description

HAL_SRAM_Write_32b

Function name

HAL_StatusTypeDef HAL_SRAM_Write_32b (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint32_t * pSrcBuffer, uint32_t BufferSize)

Function description

HAL_SRAM_Read_DMA

Function name

HAL_StatusTypeDef HAL_SRAM_Read_DMA (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint32_t * pDstBuffer, uint32_t BufferSize)

Function description

HAL_SRAM_Write_DMA

Function name

HAL_StatusTypeDef HAL_SRAM_Write_DMA (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint32_t * pSrcBuffer, uint32_t BufferSize)

Function description

HAL_SRAM_DMA_XferCpltCallback

Function name

void HAL_SRAM_DMA_XferCpltCallback (DMA_HandleTypeDef * hdma)

Function description

HAL_SRAM_DMA_XferErrorCallback

Function name

void HAL_SRAM_DMA_XferErrorCallback (DMA_HandleTypeDef * hdma)

Function description

HAL_SRAM_RegisterCallback

Function name

HAL_StatusTypeDef HAL_SRAM_RegisterCallback (SRAM_HandleTypeDef * hsram, HAL_SRAM_CallbackIDTypeDef CallbackId, pSRAM_CallbackTypeDef pCallback)

Function description

HAL_SRAM_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_SRAM_UnRegisterCallback (SRAM_HandleTypeDef * hsram, HAL_SRAM_CallbackIDTypeDef CallbackId)

Function description

HAL_SRAM_RegisterDmaCallback

Function name

HAL_StatusTypeDef HAL_SRAM_RegisterDmaCallback (SRAM_HandleTypeDef * hsram, HAL_SRAM_CallbackIDTypeDef CallbackId, pSRAM_DmaCallbackTypeDef pCallback)

Function description

HAL_SRAM_WriteOperation_Enable

Function name

HAL_StatusTypeDef HAL_SRAM_WriteOperation_Enable (SRAM_HandleTypeDef * hsram)

Function description

HAL_SRAM_WriteOperation_Disable

Function name

HAL_StatusTypeDef HAL_SRAM_WriteOperation_Disable (SRAM_HandleTypeDef * hsram)

Function description

HAL_SRAM_GetState

Function name

HAL_SRAM_StateTypeDef HAL_SRAM_GetState (const SRAM_HandleTypeDef * hsram)

Function description

76.3 SRAM Firmware driver defines

The following section lists the various define and macros of the module.

76.3.1 SRAM

SRAM

SRAM Exported Macros

__HAL_SRAM_RESET_HANDLE_STATE

Description:

- Reset SRAM handle state.

Parameters:

- __HANDLE__: SRAM handle

Return value:

- None

77 HAL TIM Generic Driver

77.1 TIM Firmware driver registers structures

77.1.1 TIM_Base_InitTypeDef

TIM_Base_InitTypeDef is defined in the stm32n6xx_hal_tim.h

Data Fields

- ***uint32_t Prescaler***
- ***uint32_t CounterMode***
- ***uint32_t Period***
- ***uint32_t ClockDivision***
- ***uint32_t RepetitionCounter***
- ***uint32_t AutoReloadPreload***

Field Documentation

- ***uint32_t TIM_Base_InitTypeDef::Prescaler***
  Specifies the prescaler value used to divide the TIM clock. This parameter can be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF Macro **__HAL_TIM_CALC_PSC()** can be used to calculate prescaler value
- ***uint32_t TIM_Base_InitTypeDef::CounterMode***
  Specifies the counter mode. This parameter can be a value of ***TIM_Counter_Mode***
- ***uint32_t TIM_Base_InitTypeDef::Period***
  Specifies the period value to be loaded into the active Auto-Reload Register at the next update event. This parameter can be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF (or 0xFFEF if dithering is activated)Macros **__HAL_TIM_CALC_PERIOD()**, **__HAL_TIM_CALC_PERIOD_DITHER()**,**__HAL_TIM_CALC_PERIOD_BY_DELAY()**, **__HAL_TIM_CALC_PERIOD_DITHER_BY_DELAY()**can be used to calculate Period value
- ***uint32_t TIM_Base_InitTypeDef::ClockDivision***
  Specifies the clock division. This parameter can be a value of ***TIM_ClockDivision***
- ***uint32_t TIM_Base_InitTypeDef::RepetitionCounter***
  Specifies the repetition counter value. Each time the RCR downcounter reaches zero, an update event is generated and counting restarts from the RCR value (N). This means in PWM mode that (N+1) corresponds to:
  - the number of PWM periods in edge-aligned mode
  - the number of half PWM period in center-aligned mode GP timers: this parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF. Advanced timers: this parameter must be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF.
- ***uint32_t TIM_Base_InitTypeDef::AutoReloadPreload***
  Specifies the auto-reload preload. This parameter can be a value of ***TIM_AutoReloadPreload***

77.1.2 TIM_OC_InitTypeDef

TIM_OC_InitTypeDef is defined in the stm32n6xx_hal_tim.h

Data Fields

- ***uint32_t OCMode***
- ***uint32_t Pulse***
- ***uint32_t OCPolarity***
- ***uint32_t OCNPolarity***
- ***uint32_t OCFastMode***
- ***uint32_t OCIdleState***
- ***uint32_t OCNIdleState***

Field Documentation

- ***uint32_t TIM_OC_InitTypeDef::OCMode***
  Specifies the TIM mode. This parameter can be a value of ***TIM_Output_Compare_and_PWM_modes***
- ***uint32_t TIM_OC_InitTypeDef::Pulse***
  Specifies the pulse value to be loaded into the Capture Compare Register. This parameter can be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF (or 0xFFEF if dithering is activated) Macros **__HAL_TIM_CALC_PULSE()**, **__HAL_TIM_CALC_PULSE_DITHER()** can be used to calculate Pulse value
- ***uint32_t TIM_OC_InitTypeDef::OCPolarity***
  Specifies the output polarity. This parameter can be a value of ***TIM_Output_Compare_Polarity***
- ***uint32_t TIM_OC_InitTypeDef::OCNPolarity***
  Specifies the complementary output polarity. This parameter can be a value of ***TIM_Output_Compare_N_Polarity***
  Note:
  – This parameter is valid only for timer instances supporting break feature.
- ***uint32_t TIM_OC_InitTypeDef::OCFastMode***
  Specifies the Fast mode state. This parameter can be a value of ***TIM_Output_Fast_State***
  Note:
  – This parameter is valid only in PWM1 and PWM2 mode.
- ***uint32_t TIM_OC_InitTypeDef::OCIdleState***
  Specifies the TIM Output Compare pin state during Idle state. This parameter can be a value of ***TIM_Output_Compare_Idle_State***
  Note:
  – This parameter is valid only for timer instances supporting break feature.
- ***uint32_t TIM_OC_InitTypeDef::OCNIdleState***
  Specifies the TIM Output Compare pin state during Idle state. This parameter can be a value of ***TIM_Output_Compare_N_Idle_State***
  Note:
  – This parameter is valid only for timer instances supporting break feature.

77.1.3 TIM_OnePulse_InitTypeDef

TIM_OnePulse_InitTypeDef is defined in the stm32n6xx_hal_tim.h

Data Fields

- ***uint32_t OCMode***
- ***uint32_t Pulse***
- ***uint32_t OCPolarity***
- ***uint32_t OCNPolarity***
- ***uint32_t OCIdleState***
- ***uint32_t OCNIdleState***
- ***uint32_t ICPolarity***
- ***uint32_t ICSelection***
- ***uint32_t ICFilter***

Field Documentation

- ***uint32_t TIM_OnePulse_InitTypeDef::OCMode***
  Specifies the TIM mode. This parameter can be a value of ***TIM_Output_Compare_and_PWM_modes***
- ***uint32_t TIM_OnePulse_InitTypeDef::Pulse***
  Specifies the pulse value to be loaded into the Capture Compare Register. This parameter can be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF (or 0xFFEF if dithering is activated) Macros **__HAL_TIM_CALC_PULSE()**, **__HAL_TIM_CALC_PULSE_DITHER()** can be used to calculate Pulse value
- ***uint32_t TIM_OnePulse_InitTypeDef::OCPolarity***
  Specifies the output polarity. This parameter can be a value of ***TIM_Output_Compare_Polarity***

- ***uint32_t TIM_OnePulse_InitTypeDef::OCNPolarity***
  Specifies the complementary output polarity. This parameter can be a value of ***TIM_Output_Compare_N_Polarity***
  Note:
  – This parameter is valid only for timer instances supporting break feature.
- ***uint32_t TIM_OnePulse_InitTypeDef::OCIdleState***
  Specifies the TIM Output Compare pin state during Idle state. This parameter can be a value of ***TIM_Output_Compare_Idle_State***
  Note:
  – This parameter is valid only for timer instances supporting break feature.
- ***uint32_t TIM_OnePulse_InitTypeDef::OCNIdleState***
  Specifies the TIM Output Compare pin state during Idle state. This parameter can be a value of ***TIM_Output_Compare_N_Idle_State***
  Note:
  – This parameter is valid only for timer instances supporting break feature.
- ***uint32_t TIM_OnePulse_InitTypeDef::ICPolarity***
  Specifies the active edge of the input signal. This parameter can be a value of ***TIM_Input_Capture_Polarity***
- ***uint32_t TIM_OnePulse_InitTypeDef::ICSelection***
  Specifies the input. This parameter can be a value of ***TIM_Input_Capture_Selection***
- ***uint32_t TIM_OnePulse_InitTypeDef::ICFilter***
  Specifies the input capture filter. This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF

77.1.4 TIM_IC_InitTypeDef

TIM_IC_InitTypeDef is defined in the stm32n6xx_hal_tim.h

Data Fields

- ***uint32_t ICPolarity***
- ***uint32_t ICSelection***
- ***uint32_t ICPrescaler***
- ***uint32_t ICFilter***

Field Documentation

- ***uint32_t TIM_IC_InitTypeDef::ICPolarity***
  Specifies the active edge of the input signal. This parameter can be a value of ***TIM_Input_Capture_Polarity***
- ***uint32_t TIM_IC_InitTypeDef::ICSelection***
  Specifies the input. This parameter can be a value of ***TIM_Input_Capture_Selection***
- ***uint32_t TIM_IC_InitTypeDef::ICPrescaler***
  Specifies the Input Capture Prescaler. This parameter can be a value of ***TIM_Input_Capture_Prescaler***
- ***uint32_t TIM_IC_InitTypeDef::ICFilter***
  Specifies the input capture filter. This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF

77.1.5 TIM_Encoder_InitTypeDef

TIM_Encoder_InitTypeDef is defined in the stm32n6xx_hal_tim.h

Data Fields

- ***uint32_t EncoderMode***
- ***uint32_t IC1Polarity***
- ***uint32_t IC1Selection***
- ***uint32_t IC1Prescaler***
- ***uint32_t IC1Filter***
- ***uint32_t IC2Polarity***
- ***uint32_t IC2Selection***
- ***uint32_t IC2Prescaler***

- ***uint32_t IC2Filter***

Field Documentation

- ***uint32_t TIM_Encoder_InitTypeDef::EncoderMode***
  Specifies the active edge of the input signal. This parameter can be a value of ***TIM_Encoder_Mode***
- ***uint32_t TIM_Encoder_InitTypeDef::IC1Polarity***
  Specifies the active edge of the input signal. This parameter can be a value of ***TIM_Encoder_Input_Polarity***
- ***uint32_t TIM_Encoder_InitTypeDef::IC1Selection***
  Specifies the input. This parameter can be a value of ***TIM_Input_Capture_Selection***
- ***uint32_t TIM_Encoder_InitTypeDef::IC1Prescaler***
  Specifies the Input Capture Prescaler. This parameter can be a value of ***TIM_Input_Capture_Prescaler***
- ***uint32_t TIM_Encoder_InitTypeDef::IC1Filter***
  Specifies the input capture filter. This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF
- ***uint32_t TIM_Encoder_InitTypeDef::IC2Polarity***
  Specifies the active edge of the input signal. This parameter can be a value of ***TIM_Encoder_Input_Polarity***
- ***uint32_t TIM_Encoder_InitTypeDef::IC2Selection***
  Specifies the input. This parameter can be a value of ***TIM_Input_Capture_Selection***
- ***uint32_t TIM_Encoder_InitTypeDef::IC2Prescaler***
  Specifies the Input Capture Prescaler. This parameter can be a value of ***TIM_Input_Capture_Prescaler***
- ***uint32_t TIM_Encoder_InitTypeDef::IC2Filter***
  Specifies the input capture filter. This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF

77.1.6 TIM_ClockConfigTypeDef

TIM_ClockConfigTypeDef is defined in the stm32n6xx_hal_tim.h

Data Fields

- ***uint32_t ClockSource***
- ***uint32_t ClockPolarity***
- ***uint32_t ClockPrescaler***
- ***uint32_t ClockFilter***

Field Documentation

- ***uint32_t TIM_ClockConfigTypeDef::ClockSource***
  TIM clock sources This parameter can be a value of ***TIM_Clock_Source***
- ***uint32_t TIM_ClockConfigTypeDef::ClockPolarity***
  TIM clock polarity This parameter can be a value of ***TIM_Clock_Polarity***
- ***uint32_t TIM_ClockConfigTypeDef::ClockPrescaler***
  TIM clock prescaler This parameter can be a value of ***TIM_Clock_Prescaler***
- ***uint32_t TIM_ClockConfigTypeDef::ClockFilter***
  TIM clock filter This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF

77.1.7 TIM_ClearInputConfigTypeDef

TIM_ClearInputConfigTypeDef is defined in the stm32n6xx_hal_tim.h

Data Fields

- ***uint32_t ClearInputState***
- ***uint32_t ClearInputSource***
- ***uint32_t ClearInputPolarity***
- ***uint32_t ClearInputPrescaler***
- ***uint32_t ClearInputFilter***

Field Documentation

- ***uint32_t TIM_ClearInputConfigTypeDef::ClearInputState***
  TIM clear Input state This parameter can be ENABLE or DISABLE

- ***uint32_t TIM_ClearInputConfigTypeDef::ClearInputSource***
  TIM clear Input sources This parameter can be a value of ***TIM_ClearInput_Source***
- ***uint32_t TIM_ClearInputConfigTypeDef::ClearInputPolarity***
  TIM Clear Input polarity This parameter can be a value of ***TIM_ClearInput_Polarity***
- ***uint32_t TIM_ClearInputConfigTypeDef::ClearInputPrescaler***
  TIM Clear Input prescaler This parameter must be 0: When OCRef clear feature is used with ETR source, ETR prescaler must be off
- ***uint32_t TIM_ClearInputConfigTypeDef::ClearInputFilter***
  TIM Clear Input filter This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF

77.1.8 TIM_MasterConfigTypeDef

TIM_MasterConfigTypeDef is defined in the stm32n6xx_hal_tim.h

Data Fields

- ***uint32_t MasterOutputTrigger***
- ***uint32_t MasterOutputTrigger2***
- ***uint32_t MasterSlaveMode***

Field Documentation

- ***uint32_t TIM_MasterConfigTypeDef::MasterOutputTrigger***
  Trigger output (TRGO) selection This parameter can be a value of ***TIM_Master_Mode_Selection***
- ***uint32_t TIM_MasterConfigTypeDef::MasterOutputTrigger2***
  Trigger output2 (TRGO2) selection This parameter can be a value of ***TIM_Master_Mode_Selection_2***
- ***uint32_t TIM_MasterConfigTypeDef::MasterSlaveMode***
  Master/slave mode selection This parameter can be a value of ***TIM_Master_Slave_Mode***
  Note:
  – When the Master/slave mode is enabled, the effect of an event on the trigger input (TRGI) is delayed to allow a perfect synchronization between the current timer and its slaves (through TRGO). It is not mandatory in case of timer synchronization mode.

77.1.9 TIM_SlaveConfigTypeDef

TIM_SlaveConfigTypeDef is defined in the stm32n6xx_hal_tim.h

Data Fields

- ***uint32_t SlaveMode***
- ***uint32_t InputTrigger***
- ***uint32_t TriggerPolarity***
- ***uint32_t TriggerPrescaler***
- ***uint32_t TriggerFilter***

Field Documentation

- ***uint32_t TIM_SlaveConfigTypeDef::SlaveMode***
  Slave mode selection This parameter can be a value of ***TIM_Slave_Mode***
- ***uint32_t TIM_SlaveConfigTypeDef::InputTrigger***
  Input Trigger source This parameter can be a value of ***TIM_Trigger_Selection***
- ***uint32_t TIM_SlaveConfigTypeDef::TriggerPolarity***
  Input Trigger polarity This parameter can be a value of ***TIM_Trigger_Polarity***
- ***uint32_t TIM_SlaveConfigTypeDef::TriggerPrescaler***
  Input trigger prescaler This parameter can be a value of ***TIM_Trigger_Prescaler***
- ***uint32_t TIM_SlaveConfigTypeDef::TriggerFilter***
  Input trigger filter This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF

77.1.10 TIM_BreakDeadTimeConfigTypeDef

TIM_BreakDeadTimeConfigTypeDef is defined in the stm32n6xx_hal_tim.h

Data Fields

- ***uint32_t OffStateRunMode***
- ***uint32_t OffStateIDLEMode***

- ***uint32_t LockLevel***
- ***uint32_t DeadTime***
- ***uint32_t BreakState***
- ***uint32_t BreakPolarity***
- ***uint32_t BreakFilter***
- ***uint32_t BreakAFMode***
- ***uint32_t Break2State***
- ***uint32_t Break2Polarity***
- ***uint32_t Break2Filter***
- ***uint32_t Break2AFMode***
- ***uint32_t AutomaticOutput***

Field Documentation

- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::OffStateRunMode***
  TIM off state in run mode, This parameter can be a value of ***TIM_OSSR_Off_State_Selection_for_Run_mode_state***
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::OffStateIDLEMode***
  TIM off state in IDLE mode, This parameter can be a value of ***TIM_OSSI_Off_State_Selection_for_Idle_mode_state***
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::LockLevel***
  TIM Lock level, This parameter can be a value of ***TIM_Lock_level***
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::DeadTime***
  TIM dead Time, This parameter can be a number between Min_Data = 0x00 and Max_Data = 0xFF
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::BreakState***
  TIM Break State, This parameter can be a value of ***TIM_Break_Input_enable_disable***
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::BreakPolarity***
  TIM Break input polarity, This parameter can be a value of ***TIM_Break_Polarity***
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::BreakFilter***
  Specifies the break input filter.This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::BreakAFMode***
  Specifies the alternate function mode of the break input.This parameter can be a value of ***TIM_Break_Input_AF_Mode***
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::Break2State***
  TIM Break2 State, This parameter can be a value of ***TIM_Break2_Input_enable_disable***
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::Break2Polarity***
  TIM Break2 input polarity, This parameter can be a value of ***TIM_Break2_Polarity***
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::Break2Filter***
  TIM break2 input filter.This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::Break2AFMode***
  Specifies the alternate function mode of the break2 input.This parameter can be a value of ***TIM_Break2_Input_AF_Mode***
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::AutomaticOutput***
  TIM Automatic Output Enable state, This parameter can be a value of ***TIM_AOE_Bit_Set_Reset***

77.1.11 __TIM_HandleTypeDef

__TIM_HandleTypeDef is defined in the stm32n6xx_hal_tim.h

Data Fields

- ***TIM_TypeDef * Instance***
- ***TIM_Base_InitTypeDef Init***
- ***HAL_TIM_ActiveChannel Channel***
- ***DMA_HandleTypeDef * hdma***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_TIM_StateTypeDef State***
- ***__IO HAL_TIM_ChannelStateTypeDef ChannelState***

- *__IO HAL_TIM_ChannelStateTypeDef ChannelNState*
- *__IO HAL_TIM_DMABurstStateTypeDef DMABurstState*
- *void(* Base_MspInitCallback*
- *void(* Base_MspDeInitCallback*
- *void(* IC_MspInitCallback*
- *void(* IC_MspDeInitCallback*
- *void(* OC_MspInitCallback*
- *void(* OC_MspDeInitCallback*
- *void(* PWM_MspInitCallback*
- *void(* PWM_MspDeInitCallback*
- *void(* OnePulse_MspInitCallback*
- *void(* OnePulse_MspDeInitCallback*
- *void(* Encoder_MspInitCallback*
- *void(* Encoder_MspDeInitCallback*
- *void(* HallSensor_MspInitCallback*
- *void(* HallSensor_MspDeInitCallback*
- *void(* PeriodElapsedCallback*
- *void(* PeriodElapsedHalfCpltCallback*
- *void(* TriggerCallback*
- *void(* TriggerHalfCpltCallback*
- *void(* IC_CaptureCallback*
- *void(* IC_CaptureHalfCpltCallback*
- *void(* OC_DelayElapsedCallback*
- *void(* PWM_PulseFinishedCallback*
- *void(* PWM_PulseFinishedHalfCpltCallback*
- *void(* ErrorCallback*
- *void(* CommutationCallback*
- *void(* CommutationHalfCpltCallback*
- *void(* BreakCallback*
- *void(* Break2Callback*
- *void(* EncoderIndexCallback*
- *void(* DirectionChangeCallback*
- *void(* IndexErrorCallback*
- *void(* TransitionErrorCallback*

Field Documentation

- ***TIM_TypeDef* __TIM_HandleTypeDef::Instance***
  Register base address
- ***TIM_Base_InitTypeDef __TIM_HandleTypeDef::Init***
  TIM Time Base required parameters
- ***HAL_TIM_ActiveChannel __TIM_HandleTypeDef::Channel***
  Active channel
- ***DMA_HandleTypeDef* __TIM_HandleTypeDef::hdma[7]***
  DMA Handlers array This array is accessed by a *DMA_Handle_index*
- ***HAL_LockTypeDef __TIM_HandleTypeDef::Lock***
  Locking object
- ***__IO HAL_TIM_StateTypeDef __TIM_HandleTypeDef::State***
  TIM operation state
- ***__IO HAL_TIM_ChannelStateTypeDef __TIM_HandleTypeDef::ChannelState[6]***
  TIM channel operation state
- ***__IO HAL_TIM_ChannelStateTypeDef __TIM_HandleTypeDef::ChannelNState[4]***
  TIM complementary channel operation state

- ***__IO HAL_TIM_DMABurstStateTypeDef __TIM_HandleTypeDef::DMABurstState***
  DMA burst operation state
- ***void(* __TIM_HandleTypeDef::Base_MspInitCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Base Msp Init Callback
- ***void(* __TIM_HandleTypeDef::Base_MspDeInitCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Base Msp DeInit Callback
- ***void(* __TIM_HandleTypeDef::IC_MspInitCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM IC Msp Init Callback
- ***void(* __TIM_HandleTypeDef::IC_MspDeInitCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM IC Msp DeInit Callback
- ***void(* __TIM_HandleTypeDef::OC_MspInitCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM OC Msp Init Callback
- ***void(* __TIM_HandleTypeDef::OC_MspDeInitCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM OC Msp DeInit Callback
- ***void(* __TIM_HandleTypeDef::PWM_MspInitCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM PWM Msp Init Callback
- ***void(* __TIM_HandleTypeDef::PWM_MspDeInitCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM PWM Msp DeInit Callback
- ***void(* __TIM_HandleTypeDef::OnePulse_MspInitCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM One Pulse Msp Init Callback
- ***void(* __TIM_HandleTypeDef::OnePulse_MspDeInitCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM One Pulse Msp DeInit Callback
- ***void(* __TIM_HandleTypeDef::Encoder_MspInitCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Encoder Msp Init Callback
- ***void(* __TIM_HandleTypeDef::Encoder_MspDeInitCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Encoder Msp DeInit Callback
- ***void(* __TIM_HandleTypeDef::HallSensor_MspInitCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Hall Sensor Msp Init Callback
- ***void(* __TIM_HandleTypeDef::HallSensor_MspDeInitCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Hall Sensor Msp DeInit Callback
- ***void(* __TIM_HandleTypeDef::PeriodElapsedCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Period Elapsed Callback
- ***void(* __TIM_HandleTypeDef::PeriodElapsedHalfCpltCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Period Elapsed half complete Callback
- ***void(* __TIM_HandleTypeDef::TriggerCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Trigger Callback
- ***void(* __TIM_HandleTypeDef::TriggerHalfCpltCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Trigger half complete Callback
- ***void(* __TIM_HandleTypeDef::IC_CaptureCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Input Capture Callback
- ***void(* __TIM_HandleTypeDef::IC_CaptureHalfCpltCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Input Capture half complete Callback
- ***void(* __TIM_HandleTypeDef::OC_DelayElapsedCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Output Compare Delay Elapsed Callback
- ***void(* __TIM_HandleTypeDef::PWM_PulseFinishedCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM PWM Pulse Finished Callback
- ***void(* __TIM_HandleTypeDef::PWM_PulseFinishedHalfCpltCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM PWM Pulse Finished half complete Callback
- ***void(* __TIM_HandleTypeDef::ErrorCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Error Callback
- ***void(* __TIM_HandleTypeDef::CommutationCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Commutation Callback
- ***void(* __TIM_HandleTypeDef::CommutationHalfCpltCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Commutation half complete Callback

- ***void(* __TIM_HandleTypeDef::BreakCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Break Callback
- ***void(* __TIM_HandleTypeDef::Break2Callback)(struct __TIM_HandleTypeDef *htim)***
  TIM Break2 Callback
- ***void(* __TIM_HandleTypeDef::EncoderIndexCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Encoder Index Callback
- ***void(* __TIM_HandleTypeDef::DirectionChangeCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Direction Change Callback
- ***void(* __TIM_HandleTypeDef::IndexErrorCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Index Error Callback
- ***void(* __TIM_HandleTypeDef::TransitionErrorCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Transition Error Callback

77.2 TIM Firmware driver API description

The following section lists the various functions of the TIM library.

77.2.1 TIMER Generic features

The Timer features include:

1. 16-bit up, down, up/down auto-reload counter.
2. 16-bit programmable prescaler allowing dividing (also on the fly) the counter clock frequency either by any factor between 1 and 65536.
3. Up to 4 independent channels for:
   – Input Capture
   – Output Compare
   – PWM generation (Edge and Center-aligned Mode)
   – One-pulse mode output
4. Synchronization circuit to control the timer with external signals and to interconnect several timers together.
5. Supports incremental encoder for positioning purposes

77.2.2 How to use this driver

1. Initialize the TIM low level resources by implementing the following functions depending on the selected feature:
   – Time Base : HAL_TIM_Base_MspInit()
   – Input Capture : HAL_TIM_IC_MspInit()
   – Output Compare : HAL_TIM_OC_MspInit()
   – PWM generation : HAL_TIM_PWM_MspInit()
   – One-pulse mode output : HAL_TIM_OnePulse_MspInit()
   – Encoder mode output : HAL_TIM_Encoder_MspInit()
2. Initialize the TIM low level resources :
   a. Enable the TIM interface clock using __HAL_RCC_TIMx_CLK_ENABLE();
   b. TIM pins configuration
      ◦ Enable the clock for the TIM GPIOs using the following function: __HAL_RCC_GPIOx_CLK_ENABLE();
      ◦ Configure these TIM pins in Alternate function mode using HAL_GPIO_Init();
3. The external Clock can be configured, if needed (the default clock is the internal clock from the APBx), using the following function: HAL_TIM_ConfigClockSource, the clock configuration should be done before any start function.

4. Configure the TIM in the desired functioning mode using one of the Initialization function of this driver:
   - HAL_TIM_Base_Init: to use the Timer to generate a simple time base
   - HAL_TIM_OC_Init, HAL_TIM_OC_ConfigChannel and optionally HAL_TIMEx_OC_ConfigPulseOnCompare: to use the Timer to generate an Output Compare signal.
   - HAL_TIM_PWM_Init and HAL_TIM_PWM_ConfigChannel: to use the Timer to generate a PWM signal.
   - HAL_TIM_IC_Init and HAL_TIM_IC_ConfigChannel: to use the Timer to measure an external signal.
   - HAL_TIM_OnePulse_Init and HAL_TIM_OnePulse_ConfigChannel: to use the Timer in One Pulse Mode.
   - HAL_TIM_Encoder_Init: to use the Timer Encoder Interface.
5. Activate the TIM peripheral using one of the start functions depending from the feature used:
   - Time Base : HAL_TIM_Base_Start(), HAL_TIM_Base_Start_DMA(), HAL_TIM_Base_Start_IT()
   - Input Capture : HAL_TIM_IC_Start(), HAL_TIM_IC_Start_DMA(), HAL_TIM_IC_Start_IT()
   - Output Compare : HAL_TIM_OC_Start(), HAL_TIM_OC_Start_DMA(), HAL_TIM_OC_Start_IT()
   - PWM generation : HAL_TIM_PWM_Start(), HAL_TIM_PWM_Start_DMA(), HAL_TIM_PWM_Start_IT()
   - One-pulse mode output : HAL_TIM_OnePulse_Start(), HAL_TIM_OnePulse_Start_IT()
   - Encoder mode output : HAL_TIM_Encoder_Start(), HAL_TIM_Encoder_Start_DMA(), HAL_TIM_Encoder_Start_IT().
6. The DMA Burst is managed with the two following functions: HAL_TIM_DMABurst_WriteStart() HAL_TIM_DMABurst_ReadStart()

Callback registration

The compilation define USE_HAL_TIM_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks.

Use Function HAL_TIM_RegisterCallback() to register a callback. HAL_TIM_RegisterCallback() takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

Use function HAL_TIM_UnRegisterCallback() to reset a callback to the default weak function. HAL_TIM_UnRegisterCallback takes as parameters the HAL peripheral handle, and the Callback ID.

These functions allow to register/unregister following callbacks:

- Base_MspInitCallback : TIM Base Msp Init Callback.
- Base_MspDeInitCallback : TIM Base Msp DeInit Callback.
- IC_MspInitCallback : TIM IC Msp Init Callback.
- IC_MspDeInitCallback : TIM IC Msp DeInit Callback.
- OC_MspInitCallback : TIM OC Msp Init Callback.
- OC_MspDeInitCallback : TIM OC Msp DeInit Callback.
- PWM_MspInitCallback : TIM PWM Msp Init Callback.
- PWM_MspDeInitCallback : TIM PWM Msp DeInit Callback.
- OnePulse_MspInitCallback : TIM One Pulse Msp Init Callback.
- OnePulse_MspDeInitCallback : TIM One Pulse Msp DeInit Callback.
- Encoder_MspInitCallback : TIM Encoder Msp Init Callback.
- Encoder_MspDeInitCallback : TIM Encoder Msp DeInit Callback.
- HallSensor_MspInitCallback : TIM Hall Sensor Msp Init Callback.
- HallSensor_MspDeInitCallback : TIM Hall Sensor Msp DeInit Callback.
- PeriodElapsedCallback : TIM Period Elapsed Callback.
- PeriodElapsedHalfCpltCallback : TIM Period Elapsed half complete Callback.
- TriggerCallback : TIM Trigger Callback.
- TriggerHalfCpltCallback : TIM Trigger half complete Callback.
- IC_CaptureCallback : TIM Input Capture Callback.
- IC_CaptureHalfCpltCallback : TIM Input Capture half complete Callback.
- OC_DelayElapsedCallback : TIM Output Compare Delay Elapsed Callback.
- PWM_PulseFinishedCallback : TIM PWM Pulse Finished Callback.
- PWM_PulseFinishedHalfCpltCallback : TIM PWM Pulse Finished half complete Callback.

- ErrorCallback : TIM Error Callback.
- CommutationCallback : TIM Commutation Callback.
- CommutationHalfCpltCallback : TIM Commutation half complete Callback.
- BreakCallback : TIM Break Callback.
- Break2Callback : TIM Break2 Callback.
- EncoderIndexCallback : TIM Encoder Index Callback.
- DirectionChangeCallback : TIM Direction Change Callback
- IndexErrorCallback : TIM Index Error Callback.
- TransitionErrorCallback : TIM Transition Error Callback

By default, after the Init and when the state is HAL_TIM_STATE_RESET all interrupt callbacks are set to the corresponding weak functions: examples HAL_TIM_TriggerCallback(), HAL_TIM_ErrorCallback().

Exception done for MspInit and MspDeInit functions that are reset to the legacy weak functionalities in the Init / DeInit only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the Init / DeInit keep and use the user MspInit / MspDeInit callbacks(registered beforehand)

Callbacks can be registered / unregistered in HAL_TIM_STATE_READY state only. Exception done MspInit / MspDeInit that can be registered / unregistered in HAL_TIM_STATE_READY or HAL_TIM_STATE_RESET state, thus registered(user) MspInit / DeInit callbacks can be used during the Init / DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_TIM_RegisterCallback() before calling DeInit or Init function.

When The compilation define USE_HAL_TIM_REGISTER_CALLBACKS is set to 0 or not defined, the callback registration feature is not available and all callbacks are set to the corresponding weak functions.

77.2.3 Detailed description of functions

HAL_TIM_Base_Init

Function name

HAL_StatusTypeDef HAL_TIM_Base_Init (TIM_HandleTypeDef * htim)

Function description

HAL_TIM_Base_DeInit

Function name

HAL_StatusTypeDef HAL_TIM_Base_DeInit (TIM_HandleTypeDef * htim)

Function description

HAL_TIM_Base_MspInit

Function name

void HAL_TIM_Base_MspInit (TIM_HandleTypeDef * htim)

Function description

HAL_TIM_Base_MspDeInit

Function name

void HAL_TIM_Base_MspDeInit (TIM_HandleTypeDef * htim)

Function description

HAL_TIM_Base_Start

Function name

HAL_StatusTypeDef HAL_TIM_Base_Start (TIM_HandleTypeDef * htim)

Function description

HAL_TIM_Base_Stop

Function name

HAL_StatusTypeDef HAL_TIM_Base_Stop (TIM_HandleTypeDef * htim)

Function description

HAL_TIM_Base_Start_IT

Function name

HAL_StatusTypeDef HAL_TIM_Base_Start_IT (TIM_HandleTypeDef * htim)

Function description

HAL_TIM_Base_Stop_IT

Function name

HAL_StatusTypeDef HAL_TIM_Base_Stop_IT (TIM_HandleTypeDef * htim)

Function description

HAL_TIM_Base_Start_DMA

Function name

HAL_StatusTypeDef HAL_TIM_Base_Start_DMA (TIM_HandleTypeDef * htim, const uint32_t * pData, uint16_t Length)

Function description

HAL_TIM_Base_Stop_DMA

Function name

HAL_StatusTypeDef HAL_TIM_Base_Stop_DMA (TIM_HandleTypeDef * htim)

Function description

HAL_TIM_OC_Init

Function name

HAL_StatusTypeDef HAL_TIM_OC_Init (TIM_HandleTypeDef * htim)

Function description

HAL_TIM_OC_DeInit

Function name

HAL_StatusTypeDef HAL_TIM_OC_DeInit (TIM_HandleTypeDef * htim)

Function description

HAL_TIM_OC_MspInit

Function name

void HAL_TIM_OC_MspInit (TIM_HandleTypeDef * htim)

Function description

HAL_TIM_OC_MspDeInit

Function name

void HAL_TIM_OC_MspDeInit (TIM_HandleTypeDef * htim)

Function description

HAL_TIM_OC_Start

Function name

HAL_StatusTypeDef HAL_TIM_OC_Start (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

HAL_TIM_OC_Stop

Function name

HAL_StatusTypeDef HAL_TIM_OC_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

HAL_TIM_OC_Start_IT

Function name

HAL_StatusTypeDef HAL_TIM_OC_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

HAL_TIM_OC_Stop_IT

Function name

HAL_StatusTypeDef HAL_TIM_OC_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

HAL_TIM_OC_Start_DMA

Function name

HAL_StatusTypeDef HAL_TIM_OC_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, const uint32_t * pData, uint16_t Length)

Function description

HAL_TIM_OC_Stop_DMA

Function name

HAL_StatusTypeDef HAL_TIM_OC_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

HAL_TIM_PWM_Init

Function name

HAL_StatusTypeDef HAL_TIM_PWM_Init (TIM_HandleTypeDef * htim)

Function description

HAL_TIM_PWM_DeInit

Function name

HAL_StatusTypeDef HAL_TIM_PWM_DeInit (TIM_HandleTypeDef * htim)

Function description

HAL_TIM_PWM_MspInit

Function name

void HAL_TIM_PWM_MspInit (TIM_HandleTypeDef * htim)

Function description

HAL_TIM_PWM_MspDeInit

Function name

void HAL_TIM_PWM_MspDeInit (TIM_HandleTypeDef * htim)

Function description

HAL_TIM_PWM_Start

Function name

HAL_StatusTypeDef HAL_TIM_PWM_Start (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

HAL_TIM_PWM_Stop

Function name

HAL_StatusTypeDef HAL_TIM_PWM_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

HAL_TIM_PWM_Start_IT

Function name

HAL_StatusTypeDef HAL_TIM_PWM_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

HAL_TIM_PWM_Stop_IT

Function name

HAL_StatusTypeDef HAL_TIM_PWM_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

HAL_TIM_PWM_Start_DMA

Function name

HAL_StatusTypeDef HAL_TIM_PWM_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, const uint32_t * pData, uint16_t Length)

Function description

HAL_TIM_PWM_Stop_DMA

Function name

HAL_StatusTypeDef HAL_TIM_PWM_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

HAL_TIM_IC_Init

Function name

HAL_StatusTypeDef HAL_TIM_IC_Init (TIM_HandleTypeDef * htim)

Function description

HAL_TIM_IC_DeInit

Function name

HAL_StatusTypeDef HAL_TIM_IC_DeInit (TIM_HandleTypeDef * htim)

Function description

HAL_TIM_IC_MspInit

Function name

void HAL_TIM_IC_MspInit (TIM_HandleTypeDef * htim)

Function description

HAL_TIM_IC_MspDeInit

Function name

void HAL_TIM_IC_MspDeInit (TIM_HandleTypeDef * htim)

Function description

HAL_TIM_IC_Start

Function name

HAL_StatusTypeDef HAL_TIM_IC_Start (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

HAL_TIM_IC_Stop

Function name

HAL_StatusTypeDef HAL_TIM_IC_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

HAL_TIM_IC_Start_IT

Function name

HAL_StatusTypeDef HAL_TIM_IC_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

HAL_TIM_IC_Stop_IT

Function name

HAL_StatusTypeDef HAL_TIM_IC_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

HAL_TIM_IC_Start_DMA

Function name

HAL_StatusTypeDef HAL_TIM_IC_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t * pData, uint16_t Length)

Function description

HAL_TIM_IC_Stop_DMA

Function name

HAL_StatusTypeDef HAL_TIM_IC_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

HAL_TIM_OnePulse_Init

Function name

HAL_StatusTypeDef HAL_TIM_OnePulse_Init (TIM_HandleTypeDef * htim, uint32_t OnePulseMode)

Function description

HAL_TIM_OnePulse_DeInit

Function name

HAL_StatusTypeDef HAL_TIM_OnePulse_DeInit (TIM_HandleTypeDef * htim)

Function description

HAL_TIM_OnePulse_MspInit

Function name

void HAL_TIM_OnePulse_MspInit (TIM_HandleTypeDef * htim)

Function description

HAL_TIM_OnePulse_MspDeInit

Function name

void HAL_TIM_OnePulse_MspDeInit (TIM_HandleTypeDef * htim)

Function description

HAL_TIM_OnePulse_Start

Function name

HAL_StatusTypeDef HAL_TIM_OnePulse_Start (TIM_HandleTypeDef * htim, uint32_t OutputChannel)

Function description

HAL_TIM_OnePulse_Stop

Function name

HAL_StatusTypeDef HAL_TIM_OnePulse_Stop (TIM_HandleTypeDef * htim, uint32_t OutputChannel)

Function description

HAL_TIM_OnePulse_Start_IT

Function name

HAL_StatusTypeDef HAL_TIM_OnePulse_Start_IT (TIM_HandleTypeDef * htim, uint32_t OutputChannel)

Function description

HAL_TIM_OnePulse_Stop_IT

Function name

HAL_StatusTypeDef HAL_TIM_OnePulse_Stop_IT (TIM_HandleTypeDef * htim, uint32_t OutputChannel)

Function description

HAL_TIM_Encoder_Init

Function name

HAL_StatusTypeDef HAL_TIM_Encoder_Init (TIM_HandleTypeDef * htim, const TIM_Encoder_InitTypeDef * sConfig)

Function description

HAL_TIM_Encoder_DeInit

Function name

HAL_StatusTypeDef HAL_TIM_Encoder_DeInit (TIM_HandleTypeDef * htim)

Function description

HAL_TIM_Encoder_MspInit

Function name

void HAL_TIM_Encoder_MspInit (TIM_HandleTypeDef * htim)

Function description

HAL_TIM_Encoder_MspDeInit

Function name

void HAL_TIM_Encoder_MspDeInit (TIM_HandleTypeDef * htim)

Function description

HAL_TIM_Encoder_Start

Function name

HAL_StatusTypeDef HAL_TIM_Encoder_Start (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

HAL_TIM_Encoder_Stop

Function name

HAL_StatusTypeDef HAL_TIM_Encoder_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

HAL_TIM_Encoder_Start_IT

Function name

HAL_StatusTypeDef HAL_TIM_Encoder_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

HAL_TIM_Encoder_Stop_IT

Function name

HAL_StatusTypeDef HAL_TIM_Encoder_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

HAL_TIM_Encoder_Start_DMA

Function name

HAL_StatusTypeDef HAL_TIM_Encoder_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t * pData1, uint32_t * pData2, uint16_t Length)

Function description

HAL_TIM_Encoder_Stop_DMA

Function name

HAL_StatusTypeDef HAL_TIM_Encoder_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

HAL_TIM_IRQHandler

Function name

void HAL_TIM_IRQHandler (TIM_HandleTypeDef * htim)

Function description

HAL_TIM_OC_ConfigChannel

Function name

HAL_StatusTypeDef HAL_TIM_OC_ConfigChannel (TIM_HandleTypeDef * htim, const TIM_OC_InitTypeDef * sConfig, uint32_t Channel)

Function description

HAL_TIM_PWM_ConfigChannel

Function name

HAL_StatusTypeDef HAL_TIM_PWM_ConfigChannel (TIM_HandleTypeDef * htim, const TIM_OC_InitTypeDef * sConfig, uint32_t Channel)

Function description

HAL_TIM_IC_ConfigChannel

Function name

HAL_StatusTypeDef HAL_TIM_IC_ConfigChannel (TIM_HandleTypeDef * htim, const TIM_IC_InitTypeDef * sConfig, uint32_t Channel)

Function description

HAL_TIM_OnePulse_ConfigChannel

Function name

HAL_StatusTypeDef HAL_TIM_OnePulse_ConfigChannel (TIM_HandleTypeDef * htim, TIM_OnePulse_InitTypeDef * sConfig, uint32_t OutputChannel, uint32_t InputChannel)

Function description

HAL_TIM_ConfigOCrefClear

Function name

HAL_StatusTypeDef HAL_TIM_ConfigOCrefClear (TIM_HandleTypeDef * htim, const TIM_ClearInputConfigTypeDef * sClearInputConfig, uint32_t Channel)

Function description

HAL_TIM_ConfigClockSource

Function name

HAL_StatusTypeDef HAL_TIM_ConfigClockSource (TIM_HandleTypeDef * htim, const TIM_ClockConfigTypeDef * sClockSourceConfig)

Function description

HAL_TIM_ConfigTI1Input

Function name

HAL_StatusTypeDef HAL_TIM_ConfigTI1Input (TIM_HandleTypeDef * htim, uint32_t TI1_Selection)

Function description

HAL_TIM_SlaveConfigSynchro

Function name

HAL_StatusTypeDef HAL_TIM_SlaveConfigSynchro (TIM_HandleTypeDef * htim, const TIM_SlaveConfigTypeDef * sSlaveConfig)

Function description

HAL_TIM_SlaveConfigSynchro_IT

Function name

HAL_StatusTypeDef HAL_TIM_SlaveConfigSynchro_IT (TIM_HandleTypeDef * htim, const TIM_SlaveConfigTypeDef * sSlaveConfig)

Function description

HAL_TIM_DMABurst_WriteStart

Function name

HAL_StatusTypeDef HAL_TIM_DMABurst_WriteStart (TIM_HandleTypeDef * htim, uint32_t BurstBaseAddress, uint32_t BurstRequestSrc, const uint32_t * BurstBuffer, uint32_t BurstLength)

Function description

HAL_TIM_DMABurst_MultiWriteStart

Function name

HAL_StatusTypeDef HAL_TIM_DMABurst_MultiWriteStart (TIM_HandleTypeDef * htim, uint32_t BurstBaseAddress, uint32_t BurstRequestSrc, const uint32_t * BurstBuffer, uint32_t BurstLength, uint32_t DataLength)

Function description

HAL_TIM_DMABurst_WriteStop

Function name

HAL_StatusTypeDef HAL_TIM_DMABurst_WriteStop (TIM_HandleTypeDef * htim, uint32_t BurstRequestSrc)

Function description

HAL_TIM_DMABurst_ReadStart

Function name

HAL_StatusTypeDef HAL_TIM_DMABurst_ReadStart (TIM_HandleTypeDef * htim, uint32_t BurstBaseAddress, uint32_t BurstRequestSrc, uint32_t * BurstBuffer, uint32_t BurstLength)

Function description

HAL_TIM_DMABurst_MultiReadStart

Function name

HAL_StatusTypeDef HAL_TIM_DMABurst_MultiReadStart (TIM_HandleTypeDef * htim, uint32_t BurstBaseAddress, uint32_t BurstRequestSrc, uint32_t * BurstBuffer, uint32_t BurstLength, uint32_t DataLength)

Function description

HAL_TIM_DMABurst_ReadStop

Function name

HAL_StatusTypeDef HAL_TIM_DMABurst_ReadStop (TIM_HandleTypeDef * htim, uint32_t BurstRequestSrc)

Function description

HAL_TIM_GenerateEvent

Function name

HAL_StatusTypeDef HAL_TIM_GenerateEvent (TIM_HandleTypeDef * htim, uint32_t EventSource)

Function description

HAL_TIM_ReadCapturedValue

Function name

uint32_t HAL_TIM_ReadCapturedValue (const TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

HAL_TIM_PeriodElapsedCallback

Function name

void HAL_TIM_PeriodElapsedCallback (TIM_HandleTypeDef * htim)

Function description

HAL_TIM_PeriodElapsedHalfCpltCallback

Function name

void HAL_TIM_PeriodElapsedHalfCpltCallback (TIM_HandleTypeDef * htim)

Function description

HAL_TIM_OC_DelayElapsedCallback

Function name

void HAL_TIM_OC_DelayElapsedCallback (TIM_HandleTypeDef * htim)

Function description

HAL_TIM_IC_CaptureCallback

Function name

void HAL_TIM_IC_CaptureCallback (TIM_HandleTypeDef * htim)

Function description

HAL_TIM_IC_CaptureHalfCpltCallback

Function name

void HAL_TIM_IC_CaptureHalfCpltCallback (TIM_HandleTypeDef * htim)

Function description

HAL_TIM_PWM_PulseFinishedCallback

Function name

void HAL_TIM_PWM_PulseFinishedCallback (TIM_HandleTypeDef * htim)

Function description

HAL_TIM_PWM_PulseFinishedHalfCpltCallback

Function name

void HAL_TIM_PWM_PulseFinishedHalfCpltCallback (TIM_HandleTypeDef * htim)

Function description

HAL_TIM_TriggerCallback

Function name

void HAL_TIM_TriggerCallback (TIM_HandleTypeDef * htim)

Function description

HAL_TIM_TriggerHalfCpltCallback

Function name

void HAL_TIM_TriggerHalfCpltCallback (TIM_HandleTypeDef * htim)

Function description

HAL_TIM_ErrorCallback

Function name

void HAL_TIM_ErrorCallback (TIM_HandleTypeDef * htim)

Function description

HAL_TIM_RegisterCallback

Function name

HAL_StatusTypeDef HAL_TIM_RegisterCallback (TIM_HandleTypeDef * htim, HAL_TIM_CallbackIDTypeDef CallbackID, pTIM_CallbackTypeDef pCallback)

Function description

HAL_TIM_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_TIM_UnRegisterCallback (TIM_HandleTypeDef * htim, HAL_TIM_CallbackIDTypeDef CallbackID)

Function description

HAL_TIM_Base_GetState

Function name

HAL_TIM_StateTypeDef HAL_TIM_Base_GetState (const TIM_HandleTypeDef * htim)

Function description

HAL_TIM_OC_GetState

Function name

HAL_TIM_StateTypeDef HAL_TIM_OC_GetState (const TIM_HandleTypeDef * htim)

Function description

HAL_TIM_PWM_GetState

Function name

HAL_TIM_StateTypeDef HAL_TIM_PWM_GetState (const TIM_HandleTypeDef * htim)

Function description

HAL_TIM_IC_GetState

Function name

HAL_TIM_StateTypeDef HAL_TIM_IC_GetState (const TIM_HandleTypeDef * htim)

Function description

HAL_TIM_OnePulse_GetState

Function name

HAL_TIM_StateTypeDef HAL_TIM_OnePulse_GetState (const TIM_HandleTypeDef * htim)

Function description

HAL_TIM_Encoder_GetState

Function name

HAL_TIM_StateTypeDef HAL_TIM_Encoder_GetState (const TIM_HandleTypeDef * htim)

Function description

HAL_TIM_GetActiveChannel

Function name

HAL_TIM_ActiveChannel HAL_TIM_GetActiveChannel (const TIM_HandleTypeDef * htim)

Function description

HAL_TIM_GetChannelState

Function name

HAL_TIM_ChannelStateTypeDef HAL_TIM_GetChannelState (const TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

HAL_TIM_DMABurstState

Function name

HAL_TIM_DMABurstStateTypeDef HAL_TIM_DMABurstState (const TIM_HandleTypeDef * htim)

Function description

TIM_Base_SetConfig

Function name

void TIM_Base_SetConfig (TIM_TypeDef * TIMx, const TIM_Base_InitTypeDef * Structure)

Function description

TIM_TI1_SetConfig

Function name

void TIM_TI1_SetConfig (TIM_TypeDef * TIMx, uint32_t TIM_ICPolarity, uint32_t TIM_ICSelection, uint32_t TIM_ICFilter)

Function description

TIM_OC2_SetConfig

Function name

void TIM_OC2_SetConfig (TIM_TypeDef * TIMx, const TIM_OC_InitTypeDef * OC_Config)

Function description

TIM_ETR_SetConfig

Function name

void TIM_ETR_SetConfig (TIM_TypeDef * TIMx, uint32_t TIM_ExtTRGPrescaler, uint32_t TIM_ExtTRGPolarity, uint32_t ExtTRGFilter)

Function description

TIM_DMADelayPulseHalfCplt

Function name

void TIM_DMADelayPulseHalfCplt (DMA_HandleTypeDef * hdma)

Function description

TIM_DMAError

Function name

void TIM_DMAError (DMA_HandleTypeDef * hdma)

Function description

TIM_DMACaptureCplt

Function name

void TIM_DMACaptureCplt (DMA_HandleTypeDef * hdma)

Function description

TIM_DMACaptureHalfCplt

Function name

void TIM_DMACaptureHalfCplt (DMA_HandleTypeDef * hdma)

Function description

TIM_CCxChannelCmd

Function name

void TIM_CCxChannelCmd (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t ChannelState)

Function description

TIM_DMA_Start_IT

Function name

HAL_StatusTypeDef TIM_DMA_Start_IT (DMA_HandleTypeDef * hdma, uint32_t src, uint32_t dst, uint32_t length)

Function description

TIM_ResetCallback

Function name

void TIM_ResetCallback (TIM_HandleTypeDef * htim)

Function description

77.3 TIM Firmware driver defines

The following section lists the various define and macros of the module.

77.3.1 TIM

TIM

TIM Automatic Output Enable

TIM_AUTOMATICOUTPUT_DISABLE

MOE can be set only by software

TIM_AUTOMATICOUTPUT_ENABLE

MOE can be set by software or automatically at the next update event (if none of the break inputs BRK and BRK2 is active)

TIM Auto-Reload Preload

TIM_AUTORELOAD_PRELOAD_DISABLE

TIMx_ARR register is not buffered

TIM_AUTORELOAD_PRELOAD_ENABLE

TIMx_ARR register is buffered

TIM Break2 Input Alternate Function Mode

TIM_BREAK2_AFMODE_INPUT

Break2 input BRK2 in input mode

TIM_BREAK2_AFMODE_BIDIRECTIONAL

Break2 input BRK2 in bidirectional mode

TIM Break input 2 Enable

TIM_BREAK2_DISABLE

Break input BRK2 is disabled

TIM_BREAK2_ENABLE

Break input BRK2 is enabled

TIM Break Input 2 Polarity

TIM_BREAK2POLARITY_LOW

Break input BRK2 is active low

TIM_BREAK2POLARITY_HIGH

Break input BRK2 is active high

TIM Break Input Alternate Function Mode

TIM_BREAK_AFMODE_INPUT

Break input BRK in input mode

TIM_BREAK_AFMODE_BIDIRECTIONAL

Break input BRK in bidirectional mode

TIM Break Input Enable

TIM_BREAK_ENABLE

Break input BRK is enabled

TIM_BREAK_DISABLE

Break input BRK is disabled

TIM Break Input Polarity

TIM_BREAKPOLARITY_LOW

Break input BRK is active low

TIM_BREAKPOLARITY_HIGH

Break input BRK is active high

TIM Break System

TIM_BREAK_SYSTEM_ECC_TCM

Enables and locks the Cortex-M55 TCM double ECC error signal connection to TIM1/8/15/16/17 break inputs

TIM_BREAK_SYSTEM_ECC_CACHE

Enables and locks the Cortex-M55 cache double ECC error signa connection to TIM1/8/15/16/17 break inputs

TIM_BREAK_SYSTEM_ECC_BACKUP_RAM

Enables and locks the Backup RAM ECC double error detection flag connection to TIM1/8/15/16/17 break inputs

TIM_BREAK_SYSTEM_PVD

Enables and locks the PVD signal connection to TIM1/8/15/16/17 break inputs

TIM_BREAK_SYSTEM_LOCKUP

Enables and locks the Cortex-M55 lockup signal connection to TIM1/8/15/16/17 break inputs

CCx DMA request selection

TIM_CCDMAREQUEST_CC

CCx DMA request sent when capture or compare match event occurs

TIM_CCDMAREQUEST_UPDATE

CCx DMA requests sent when update event occurs

TIM Channel

TIM_CHANNEL_1

Capture/compare channel 1 identifier

TIM_CHANNEL_2

Capture/compare channel 2 identifier

TIM_CHANNEL_3

Capture/compare channel 3 identifier

TIM_CHANNEL_4

Capture/compare channel 4 identifier

TIM_CHANNEL_5

Compare channel 5 identifier

TIM_CHANNEL_6

Compare channel 6 identifier

TIM_CHANNEL_ALL

Global Capture/compare channel identifier

TIM Clear Input Polarity

TIM_CLEARINPUTPOLARITY_INVERTED

Polarity for ETRx pin

TIM_CLEARINPUTPOLARITY_NONINVERTED

Polarity for ETRx pin

TIM Clear Input Prescaler

TIM_CLEARINPUTPRESCALER_DIV1

No prescaler is used

TIM_CLEARINPUTPRESCALER_DIV2

Prescaler for External ETR pin: Capture performed once every 2 events.

TIM_CLEARINPUTPRESCALER_DIV4

Prescaler for External ETR pin: Capture performed once every 4 events.

TIM_CLEARINPUTPRESCALER_DIV8

Prescaler for External ETR pin: Capture performed once every 8 events.

TIM Clear Input Source

TIM_CLEARINPUTSOURCE_NONE

OCREF_CLR is disabled

TIM_CLEARINPUTSOURCE_ETR

OCREF_CLR is connected to ETRF input

TIM Clock Division

TIM_CLOCKDIVISION_DIV1

Clock division: tDTS=tCK_INT

TIM_CLOCKDIVISION_DIV2

Clock division: tDTS=2*tCK_INT

TIM_CLOCKDIVISION_DIV4

Clock division: tDTS=4*tCK_INT

TIM Clock Polarity

TIM_CLOCKPOLARITY_INVERTED

Polarity for ETRx clock sources

TIM_CLOCKPOLARITY_NONINVERTED

Polarity for ETRx clock sources

TIM_CLOCKPOLARITY_RISING

Polarity for TIx clock sources

TIM_CLOCKPOLARITY_FALLING

Polarity for TIx clock sources

TIM_CLOCKPOLARITY_BOTHEDGE

Polarity for TIx clock sources

TIM Clock Prescaler

TIM_CLOCKPRESCALER_DIV1

No prescaler is used

TIM_CLOCKPRESCALER_DIV2

Prescaler for External ETR Clock: Capture performed once every 2 events.

TIM_CLOCKPRESCALER_DIV4

Prescaler for External ETR Clock: Capture performed once every 4 events.

TIM_CLOCKPRESCALER_DIV8

Prescaler for External ETR Clock: Capture performed once every 8 events.

TIM Clock Source

TIM_CLOCKSOURCE_INTERNAL

Internal clock source

TIM_CLOCKSOURCE_ETRMODE1

External clock source mode 1 (ETRF)

TIM_CLOCKSOURCE_ETRMODE2

External clock source mode 2

TIM_CLOCKSOURCE_TI1ED

External clock source mode 1 (TTI1FP1 + edge detect.)

TIM_CLOCKSOURCE_TI1

External clock source mode 1 (TTI1FP1)

TIM_CLOCKSOURCE_TI2

External clock source mode 1 (TTI2FP2)

TIM_CLOCKSOURCE_ITR0

External clock source mode 1 (ITR0)

TIM_CLOCKSOURCE_ITR1

External clock source mode 1 (ITR1)

TIM_CLOCKSOURCE_ITR2

External clock source mode 1 (ITR2)

TIM_CLOCKSOURCE_ITR3

External clock source mode 1 (ITR3)

TIM_CLOCKSOURCE_ITR4

External clock source mode 1 (ITR4)

TIM_CLOCKSOURCE_ITR5

External clock source mode 1 (ITR5)

TIM_CLOCKSOURCE_ITR6

External clock source mode 1 (ITR6)

TIM_CLOCKSOURCE_ITR7

External clock source mode 1 (ITR7)

TIM_CLOCKSOURCE_ITR8

External clock source mode 1 (ITR8)

TIM_CLOCKSOURCE_ITR9

External clock source mode 1 (ITR9)

TIM_CLOCKSOURCE_ITR10

External clock source mode 1 (ITR10)

TIM_CLOCKSOURCE_ITR11

External clock source mode 1 (ITR11)

TIM_CLOCKSOURCE_ITR12

External clock source mode 1 (ITR12)

TIM_CLOCKSOURCE_ITR13

External clock source mode 1 (ITR13)

TIM Commutation Source

TIM_COMMUTATION_TRGI

When Capture/compare control bits are preloaded, they are updated by setting the COMG bit or when an rising edge occurs on trigger input

TIM_COMMUTATION_SOFTWARE

When Capture/compare control bits are preloaded, they are updated by setting the COMG bit

TIM Counter Mode

TIM_COUNTERMODE_UP

Counter used as up-counter

TIM_COUNTERMODE_DOWN

Counter used as down-counter

TIM_COUNTERMODE_CENTERALIGNED1

Center-aligned mode 1

TIM_COUNTERMODE_CENTERALIGNED2

Center-aligned mode 2

TIM_COUNTERMODE_CENTERALIGNED3

Center-aligned mode 3

TIM DMA Base Address

TIM_DMABASE_CR1

TIM_DMABASE_CR2

TIM_DMABASE_SMCR

TIM_DMABASE_DIER

TIM_DMABASE_SR

TIM_DMABASE_EGR

TIM_DMABASE_CCMR1

TIM_DMABASE_CCMR2

TIM_DMABASE_CCER

TIM_DMABASE_CNT

TIM_DMABASE_PSC

TIM_DMABASE_ARR

TIM_DMABASE_RCR

TIM_DMABASE_CCR1

TIM_DMABASE_CCR2

TIM_DMABASE_CCR3

TIM_DMABASE_CCR4

TIM_DMABASE_BDTR

TIM_DMABASE_CCR5

TIM_DMABASE_CCR6

TIM_DMABASE_CCMR3

TIM_DMABASE_DTR2

TIM_DMABASE_ECR

TIM_DMABASE_TISEL

TIM_DMABASE_AF1

TIM_DMABASE_AF2

TIM DMA Burst Length

TIM_DMABURSTLENGTH_1TRANSFER

The transfer is done to 1 register starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_2TRANSFERS

The transfer is done to 2 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_3TRANSFERS

The transfer is done to 3 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_4TRANSFERS

The transfer is done to 4 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_5TRANSFERS

The transfer is done to 5 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_6TRANSFERS

The transfer is done to 6 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_7TRANSFERS

The transfer is done to 7 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_8TRANSFERS

The transfer is done to 8 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_9TRANSFERS

The transfer is done to 9 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_10TRANSFERS

The transfer is done to 10 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_11TRANSFERS

The transfer is done to 11 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_12TRANSFERS

The transfer is done to 12 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_13TRANSFERS

The transfer is done to 13 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_14TRANSFERS

The transfer is done to 14 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_15TRANSFERS

The transfer is done to 15 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_16TRANSFERS

The transfer is done to 16 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_17TRANSFERS

The transfer is done to 17 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_18TRANSFERS

The transfer is done to 18 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_19TRANSFERS

The transfer is done to 19 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_20TRANSFERS

The transfer is done to 20 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_21TRANSFERS

The transfer is done to 21 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_22TRANSFERS

The transfer is done to 22 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_23TRANSFERS

The transfer is done to 23 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_24TRANSFERS

The transfer is done to 24 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_25TRANSFERS

The transfer is done to 25 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_26TRANSFERS

The transfer is done to 26 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM DMA Sources

TIM_DMA_UPDATE

DMA request is triggered by the update event

TIM_DMA_CC1

DMA request is triggered by the capture/compare macth 1 event

TIM_DMA_CC2

DMA request is triggered by the capture/compare macth 2 event event

TIM_DMA_CC3

DMA request is triggered by the capture/compare macth 3 event event

TIM_DMA_CC4

DMA request is triggered by the capture/compare macth 4 event event

TIM_DMA_COM

DMA request is triggered by the commutation event

TIM_DMA_TRIGGER

DMA request is triggered by the trigger event

TIM Encoder Input Polarity

TIM_ENCODERINPUTPOLARITY_RISING

Encoder input with rising edge polarity

TIM_ENCODERINPUTPOLARITY_FALLING

Encoder input with falling edge polarity

TIM Encoder Mode

TIM_ENCODERMODE_TI1

Quadrature encoder mode 1, x2 mode, counts up/down on TI1FP1 edge depending on TI2FP2 level

TIM_ENCODERMODE_TI2

Quadrature encoder mode 2, x2 mode, counts up/down on TI2FP2 edge depending on TI1FP1 level.

TIM_ENCODERMODE_TI12

Quadrature encoder mode 3, x4 mode, counts up/down on both TI1FP1 and TI2FP2 edges depending on the level of the other input.

TIM_ENCODERMODE_CLOCKPLUSDIRECTION_X2

Encoder mode: Clock plus direction, x2 mode

TIM_ENCODERMODE_CLOCKPLUSDIRECTION_X1

Encoder mode: Clock plus direction, x1 mode, TI2FP2 edge sensitivity is set by CC2P

TIM_ENCODERMODE_DIRECTIONALCLOCK_X2

Encoder mode: Directional Clock, x2 mode

TIM_ENCODERMODE_DIRECTIONALCLOCK_X1_TI12

Encoder mode: Directional Clock, x1 mode, TI1FP1 and TI2FP2 edge sensitivity is set by CC1P and CC2P

TIM_ENCODERMODE_X1_TI1

Quadrature encoder mode: x1 mode, counting on TI1FP1 edges only, edge sensitivity is set by CC1P

TIM_ENCODERMODE_X1_TI2

Quadrature encoder mode: x1 mode, counting on TI2FP2 edges only, edge sensitivity is set by CC1P

TIM ETR Polarity

TIM_ETRPOLARITY_INVERTED

Polarity for ETR source

TIM_ETRPOLARITY_NONINVERTED

Polarity for ETR source

TIM ETR Prescaler

TIM_ETRPRESCALER_DIV1

No prescaler is used

TIM_ETRPRESCALER_DIV2

ETR input source is divided by 2

TIM_ETRPRESCALER_DIV4

ETR input source is divided by 4

TIM_ETRPRESCALER_DIV8

ETR input source is divided by 8

TIM Event Source

TIM_EVENTSOURCE_UPDATE

Reinitialize the counter and generates an update of the registers

TIM_EVENTSOURCE_CC1

A capture/compare event is generated on channel 1

TIM_EVENTSOURCE_CC2

A capture/compare event is generated on channel 2

TIM_EVENTSOURCE_CC3

A capture/compare event is generated on channel 3

TIM_EVENTSOURCE_CC4

A capture/compare event is generated on channel 4

TIM_EVENTSOURCE_COM

A commutation event is generated

TIM_EVENTSOURCE_TRIGGER

A trigger event is generated

TIM_EVENTSOURCE_BREAK

A break event is generated

TIM_EVENTSOURCE_BREAK2

A break 2 event is generated

TIM Exported Macros

__HAL_TIM_RESET_HANDLE_STATE

Description:

- Reset TIM handle state.

Parameters:

- __HANDLE__: TIM handle.

Return value:

- None

__HAL_TIM_ENABLE

Description:

- Enable the TIM peripheral.

Parameters:

- __HANDLE__: TIM handle

Return value:

- None

__HAL_TIM_MOE_ENABLE

Description:

- Enable the TIM main Output.

Parameters:

- __HANDLE__: TIM handle

Return value:

- None

__HAL_TIM_DISABLE

Description:

- Disable the TIM peripheral.

Parameters:

- __HANDLE__: TIM handle

Return value:

- None

__HAL_TIM_MOE_DISABLE

Description:

- Disable the TIM main Output.

Parameters:

- __HANDLE__: TIM handle

Return value:

- None

Notes:

- The Main Output Enable of a timer instance is disabled only if all the CCx and CCxN channels have been disabled

__HAL_TIM_MOE_DISABLE_UNCONDITIONALLY

Description:

- Disable the TIM main Output.

Parameters:

- __HANDLE__: TIM handle

Return value:

- None

Notes:

- The Main Output Enable of a timer instance is disabled unconditionally

__HAL_TIM_ENABLE_IT

Description:

- Enable the specified TIM interrupt.

Parameters:

- __HANDLE__: specifies the TIM Handle.
- __INTERRUPT__: specifies the TIM interrupt source to enable. This parameter can be one of the following values:
  - TIM_IT_UPDATE: Update interrupt
  - TIM_IT_CC1: Capture/Compare 1 interrupt
  - TIM_IT_CC2: Capture/Compare 2 interrupt
  - TIM_IT_CC3: Capture/Compare 3 interrupt
  - TIM_IT_CC4: Capture/Compare 4 interrupt
  - TIM_IT_COM: Commutation interrupt
  - TIM_IT_TRIGGER: Trigger interrupt
  - TIM_IT_BREAK: Break interrupt
  - TIM_IT_IDX: Index interrupt
  - TIM_IT_DIR: Direction change interrupt
  - TIM_IT_IERR: Index error interrupt
  - TIM_IT_TERR: Transition error interrupt

Return value:

- None

__HAL_TIM_DISABLE_IT

Description:

- Disable the specified TIM interrupt.

Parameters:

- __HANDLE__: specifies the TIM Handle.
- __INTERRUPT__: specifies the TIM interrupt source to disable. This parameter can be one of the following values:
  - TIM_IT_UPDATE: Update interrupt
  - TIM_IT_CC1: Capture/Compare 1 interrupt
  - TIM_IT_CC2: Capture/Compare 2 interrupt
  - TIM_IT_CC3: Capture/Compare 3 interrupt
  - TIM_IT_CC4: Capture/Compare 4 interrupt
  - TIM_IT_COM: Commutation interrupt
  - TIM_IT_TRIGGER: Trigger interrupt
  - TIM_IT_BREAK: Break interrupt
  - TIM_IT_IDX: Index interrupt
  - TIM_IT_DIR: Direction change interrupt
  - TIM_IT_IERR: Index error interrupt
  - TIM_IT_TERR: Transition error interrupt

Return value:

- None

__HAL_TIM_ENABLE_DMA

Description:

- Enable the specified DMA request.

Parameters:

- __HANDLE__: specifies the TIM Handle.
- __DMA__: specifies the TIM DMA request to enable. This parameter can be one of the following values:
  - TIM_DMA_UPDATE: Update DMA request
  - TIM_DMA_CC1: Capture/Compare 1 DMA request
  - TIM_DMA_CC2: Capture/Compare 2 DMA request
  - TIM_DMA_CC3: Capture/Compare 3 DMA request
  - TIM_DMA_CC4: Capture/Compare 4 DMA request
  - TIM_DMA_COM: Commutation DMA request
  - TIM_DMA_TRIGGER: Trigger DMA request

Return value:

- None

__HAL_TIM_DISABLE_DMA

Description:

- Disable the specified DMA request.

Parameters:

- __HANDLE__: specifies the TIM Handle.
- __DMA__: specifies the TIM DMA request to disable. This parameter can be one of the following values:
  - TIM_DMA_UPDATE: Update DMA request
  - TIM_DMA_CC1: Capture/Compare 1 DMA request
  - TIM_DMA_CC2: Capture/Compare 2 DMA request
  - TIM_DMA_CC3: Capture/Compare 3 DMA request
  - TIM_DMA_CC4: Capture/Compare 4 DMA request
  - TIM_DMA_COM: Commutation DMA request
  - TIM_DMA_TRIGGER: Trigger DMA request

Return value:

- None

__HAL_TIM_GET_FLAG

Description:

- Check whether the specified TIM interrupt flag is set or not.

Parameters:

- __HANDLE__: specifies the TIM Handle.
- __FLAG__: specifies the TIM interrupt flag to check. This parameter can be one of the following values:
  - TIM_FLAG_UPDATE: Update interrupt flag
  - TIM_FLAG_CC1: Capture/Compare 1 interrupt flag
  - TIM_FLAG_CC2: Capture/Compare 2 interrupt flag
  - TIM_FLAG_CC3: Capture/Compare 3 interrupt flag
  - TIM_FLAG_CC4: Capture/Compare 4 interrupt flag
  - TIM_FLAG_CC5: Compare 5 interrupt flag
  - TIM_FLAG_CC6: Compare 6 interrupt flag
  - TIM_FLAG_COM: Commutation interrupt flag
  - TIM_FLAG_TRIGGER: Trigger interrupt flag
  - TIM_FLAG_BREAK: Break interrupt flag
  - TIM_FLAG_BREAK2: Break 2 interrupt flag
  - TIM_FLAG_SYSTEM_BREAK: System Break interrupt flag
  - TIM_FLAG_CC1OF: Capture/Compare 1 overcapture flag
  - TIM_FLAG_CC2OF: Capture/Compare 2 overcapture flag
  - TIM_FLAG_CC3OF: Capture/Compare 3 overcapture flag
  - TIM_FLAG_CC4OF: Capture/Compare 4 overcapture flag
  - TIM_FLAG_IDX: Index interrupt flag
  - TIM_FLAG_DIR: Direction change interrupt flag
  - TIM_FLAG_IERR: Index error interrupt flag
  - TIM_FLAG_TERR: Transition error interrupt flag

Return value:

- The: new state of __FLAG__ (TRUE or FALSE).

__HAL_TIM_CLEAR_FLAG

Description:

- Clear the specified TIM interrupt flag.

Parameters:

- __HANDLE__: specifies the TIM Handle.
- __FLAG__: specifies the TIM interrupt flag to clear. This parameter can be one of the following values:
  - TIM_FLAG_UPDATE: Update interrupt flag
  - TIM_FLAG_CC1: Capture/Compare 1 interrupt flag
  - TIM_FLAG_CC2: Capture/Compare 2 interrupt flag
  - TIM_FLAG_CC3: Capture/Compare 3 interrupt flag
  - TIM_FLAG_CC4: Capture/Compare 4 interrupt flag
  - TIM_FLAG_CC5: Compare 5 interrupt flag
  - TIM_FLAG_CC6: Compare 6 interrupt flag
  - TIM_FLAG_COM: Commutation interrupt flag
  - TIM_FLAG_TRIGGER: Trigger interrupt flag
  - TIM_FLAG_BREAK: Break interrupt flag
  - TIM_FLAG_BREAK2: Break 2 interrupt flag
  - TIM_FLAG_SYSTEM_BREAK: System Break interrupt flag
  - TIM_FLAG_CC1OF: Capture/Compare 1 overcapture flag
  - TIM_FLAG_CC2OF: Capture/Compare 2 overcapture flag
  - TIM_FLAG_CC3OF: Capture/Compare 3 overcapture flag
  - TIM_FLAG_CC4OF: Capture/Compare 4 overcapture flag
  - TIM_FLAG_IDX: Index interrupt flag
  - TIM_FLAG_DIR: Direction change interrupt flag
  - TIM_FLAG_IERR: Index error interrupt flag
  - TIM_FLAG_TERR: Transition error interrupt flag

Return value:

- The: new state of __FLAG__ (TRUE or FALSE).

__HAL_TIM_GET_IT_SOURCE

Description:

- Check whether the specified TIM interrupt source is enabled or not.

Parameters:

- __HANDLE__: TIM handle
- __INTERRUPT__: specifies the TIM interrupt source to check. This parameter can be one of the following values:
  - TIM_IT_UPDATE: Update interrupt
  - TIM_IT_CC1: Capture/Compare 1 interrupt
  - TIM_IT_CC2: Capture/Compare 2 interrupt
  - TIM_IT_CC3: Capture/Compare 3 interrupt
  - TIM_IT_CC4: Capture/Compare 4 interrupt
  - TIM_IT_COM: Commutation interrupt
  - TIM_IT_TRIGGER: Trigger interrupt
  - TIM_IT_BREAK: Break interrupt
  - TIM_IT_IDX: Index interrupt
  - TIM_IT_DIR: Direction change interrupt
  - TIM_IT_IERR: Index error interrupt
  - TIM_IT_TERR: Transition error interrupt

Return value:

- The: state of TIM_IT (SET or RESET).

__HAL_TIM_CLEAR_IT

Description:

- Clear the TIM interrupt pending bits.

Parameters:

- __HANDLE__: TIM handle
- __INTERRUPT__: specifies the interrupt pending bit to clear. This parameter can be one of the following values:
  - TIM_IT_UPDATE: Update interrupt
  - TIM_IT_CC1: Capture/Compare 1 interrupt
  - TIM_IT_CC2: Capture/Compare 2 interrupt
  - TIM_IT_CC3: Capture/Compare 3 interrupt
  - TIM_IT_CC4: Capture/Compare 4 interrupt
  - TIM_IT_COM: Commutation interrupt
  - TIM_IT_TRIGGER: Trigger interrupt
  - TIM_IT_BREAK: Break interrupt
  - TIM_IT_IDX: Index interrupt
  - TIM_IT_DIR: Direction change interrupt
  - TIM_IT_IERR: Index error interrupt
  - TIM_IT_TERR: Transition error interrupt

Return value:

- None

__HAL_TIM_UIFREMAP_ENABLE

Description:

- Force a continuous copy of the update interrupt flag (UIF) into the timer counter register (bit 31).

Parameters:

- __HANDLE__: TIM handle.

Return value:

- None: mode.

Notes:

- This allows both the counter value and a potential roll-over condition signalled by the UIFCPY flag to be read in an atomic way.

__HAL_TIM_UIFREMAP_DISABLE

Description:

- Disable update interrupt flag (UIF) remapping.

Parameters:

- __HANDLE__: TIM handle.

Return value:

- None: mode.

__HAL_TIM_GET_UIFCPY

Description:

- Get update interrupt flag (UIF) copy status.

Parameters:

- __COUNTER__: Counter value.

Return value:

- The: state of UIFCPY (TRUE or FALSE). mode.

__HAL_TIM_IS_TIM_COUNTING_DOWN

Description:

- Indicates whether or not the TIM Counter is used as downcounter.

Parameters:

- __HANDLE__: TIM handle.

Return value:

- False: (Counter used as upcounter) or True (Counter used as downcounter)

Notes:

- This macro is particularly useful to get the counting mode when the timer operates in Center-aligned mode or Encoder mode.

__HAL_TIM_SET_PRESCALER

Description:

- Set the TIM Prescaler on runtime.

Parameters:

- __HANDLE__: TIM handle.
- __PRESC__: specifies the Prescaler new value.

Return value:

- None

__HAL_TIM_SET_COUNTER

Description:

- Set the TIM Counter Register value on runtime.

Parameters:

- __HANDLE__: TIM handle.
- __COUNTER__: specifies the Counter register new value.

Return value:

- None

__HAL_TIM_GET_COUNTER

Description:

- Get the TIM Counter Register value on runtime.

Parameters:

- __HANDLE__: TIM handle.

Return value:

- 16-bit: or 32-bit value of the timer counter register (TIMx_CNT)

__HAL_TIM_SET_AUTORELOAD

Description:

- Set the TIM Autoreload Register value on runtime without calling another time any Init function.

Parameters:

- __HANDLE__: TIM handle.
- __AUTORELOAD__: specifies the Counter register new value.

Return value:

- None

__HAL_TIM_GET_AUTORELOAD

Description:

- Get the TIM Autoreload Register value on runtime.

Parameters:

- __HANDLE__: TIM handle.

Return value:

- 16-bit: or 32-bit value of the timer auto-reload register(TIMx_ARR)

__HAL_TIM_SET_CLOCKDIVISION

Description:

- Set the TIM Clock Division value on runtime without calling another time any Init function.

Parameters:

- __HANDLE__: TIM handle.
- __CKD__: specifies the clock division value. This parameter can be one of the following value:
  - TIM_CLOCKDIVISION_DIV1: tDTS=tCK_INT
  - TIM_CLOCKDIVISION_DIV2: tDTS=2*tCK_INT
  - TIM_CLOCKDIVISION_DIV4: tDTS=4*tCK_INT

Return value:

- None

__HAL_TIM_GET_CLOCKDIVISION

Description:

- Get the TIM Clock Division value on runtime.

Parameters:

- __HANDLE__: TIM handle.

Return value:

- The: clock division can be one of the following values:
  - TIM_CLOCKDIVISION_DIV1: tDTS=tCK_INT
  - TIM_CLOCKDIVISION_DIV2: tDTS=2*tCK_INT
  - TIM_CLOCKDIVISION_DIV4: tDTS=4*tCK_INT

__HAL_TIM_SET_ICPRESCALER

Description:

- Set the TIM Input Capture prescaler on runtime without calling another time

Parameters:

- __HANDLE__: TIM handle.
- __CHANNEL__: TIM Channels to be configured. This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
- __ICPSC__: specifies the Input Capture4 prescaler new value. This parameter can be one of the following values:
  - TIM_ICPSC_DIV1: no prescaler
  - TIM_ICPSC_DIV2: capture is done once every 2 events
  - TIM_ICPSC_DIV4: capture is done once every 4 events
  - TIM_ICPSC_DIV8: capture is done once every 8 events

Return value:

- None

__HAL_TIM_GET_ICPRESCALER

Description:

- Get the TIM Input Capture prescaler on runtime.

Parameters:

- __HANDLE__: TIM handle.
- __CHANNEL__: TIM Channels to be configured. This parameter can be one of the following values:
  - TIM_CHANNEL_1: get input capture 1 prescaler value
  - TIM_CHANNEL_2: get input capture 2 prescaler value
  - TIM_CHANNEL_3: get input capture 3 prescaler value
  - TIM_CHANNEL_4: get input capture 4 prescaler value

Return value:

- The: input capture prescaler can be one of the following values:
  - TIM_ICPSC_DIV1: no prescaler
  - TIM_ICPSC_DIV2: capture is done once every 2 events
  - TIM_ICPSC_DIV4: capture is done once every 4 events
  - TIM_ICPSC_DIV8: capture is done once every 8 events

__HAL_TIM_SET_COMPARE

Description:

- Set the TIM Capture Compare Register value on runtime without calling another time ConfigChannel function.

Parameters:

- __HANDLE__: TIM handle.
- __CHANNEL__: TIM Channels to be configured. This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
  - TIM_CHANNEL_5: TIM Channel 5 selected
  - TIM_CHANNEL_6: TIM Channel 6 selected
- __COMPARE__: specifies the Capture Compare register new value.

Return value:

- None

__HAL_TIM_GET_COMPARE

Description:

- Get the TIM Capture Compare Register value on runtime.

Parameters:

- __HANDLE__: TIM handle.
- __CHANNEL__: TIM Channel associated with the capture compare register This parameter can be one of the following values:
  - TIM_CHANNEL_1: get capture/compare 1 register value
  - TIM_CHANNEL_2: get capture/compare 2 register value
  - TIM_CHANNEL_3: get capture/compare 3 register value
  - TIM_CHANNEL_4: get capture/compare 4 register value
  - TIM_CHANNEL_5: get capture/compare 5 register value
  - TIM_CHANNEL_6: get capture/compare 6 register value

Return value:

- 16-bit: or 32-bit value of the capture/compare register (TIMx_CCRy)

__HAL_TIM_ENABLE_OCxPRELOAD

Description:

- Set the TIM Output compare preload.

Parameters:

- __HANDLE__: TIM handle.
- __CHANNEL__: TIM Channels to be configured. This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
  - TIM_CHANNEL_5: TIM Channel 5 selected
  - TIM_CHANNEL_6: TIM Channel 6 selected

Return value:

- None

__HAL_TIM_DISABLE_OCxPRELOAD

Description:

- Reset the TIM Output compare preload.

Parameters:

- __HANDLE__: TIM handle.
- __CHANNEL__: TIM Channels to be configured. This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
  - TIM_CHANNEL_5: TIM Channel 5 selected
  - TIM_CHANNEL_6: TIM Channel 6 selected

Return value:

- None

__HAL_TIM_ENABLE_OCxFAST

Description:

- Enable fast mode for a given channel.

Parameters:

- __HANDLE__: TIM handle.
- __CHANNEL__: TIM Channels to be configured. This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
  - TIM_CHANNEL_5: TIM Channel 5 selected
  - TIM_CHANNEL_6: TIM Channel 6 selected

Return value:

- None

Notes:

- When fast mode is enabled an active edge on the trigger input acts like a compare match on CCx output. Delay to sample the trigger input and to activate CCx output is reduced to 3 clock cycles. Fast mode acts only if the channel is configured in PWM1 or PWM2 mode.

__HAL_TIM_DISABLE_OCxFAST

Description:

- Disable fast mode for a given channel.

Parameters:

- __HANDLE__: TIM handle.
- __CHANNEL__: TIM Channels to be configured. This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
  - TIM_CHANNEL_5: TIM Channel 5 selected
  - TIM_CHANNEL_6: TIM Channel 6 selected

Return value:

- None

Notes:

- When fast mode is disabled CCx output behaves normally depending on counter and CCRx values even when the trigger is ON. The minimum delay to activate CCx output when an active edge occurs on the trigger input is 5 clock cycles.

__HAL_TIM_URS_ENABLE

Description:

- Set the Update Request Source (URS) bit of the TIMx_CR1 register.

Parameters:

- __HANDLE__: TIM handle.

Return value:

- None

Notes:

- When the URS bit of the TIMx_CR1 register is set, only counter overflow/underflow generates an update interrupt or DMA request (if enabled)

__HAL_TIM_URS_DISABLE

Description:

- Reset the Update Request Source (URS) bit of the TIMx_CR1 register.

Parameters:

- __HANDLE__: TIM handle.

Return value:

- None

Notes:

- When the URS bit of the TIMx_CR1 register is reset, any of the following events generate an update interrupt or DMA request (if enabled): _ Counter overflow underflow _ Setting the UG bit _ Update generation through the slave mode controller

__HAL_TIM_SET_CAPTUREPOLARITY

Description:

- Set the TIM Capture x input polarity on runtime.

Parameters:

- __HANDLE__: TIM handle.
- __CHANNEL__: TIM Channels to be configured. This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
- __POLARITY__: Polarity for TIx source
  - TIM_INPUTCHANNELPOLARITY_RISING: Rising Edge
  - TIM_INPUTCHANNELPOLARITY_FALLING: Falling Edge
  - TIM_INPUTCHANNELPOLARITY_BOTHEDGE: Rising and Falling Edge

Return value:

- None

__HAL_TIM_SELECT_CCDMAREQUEST

Description:

- Select the Capture/compare DMA request source.

Parameters:

- __HANDLE__: specifies the TIM Handle.
- __CCDMA__: specifies Capture/compare DMA request source This parameter can be one of the following values:
  - TIM_CCDMAREQUEST_CC: CCx DMA request generated on Capture/Compare event
  - TIM_CCDMAREQUEST_UPDATE: CCx DMA request generated on Update event

Return value:

- None

TIM Flag Definition

TIM_FLAG_UPDATE

Update interrupt flag

TIM_FLAG_CC1

Capture/Compare 1 interrupt flag

TIM_FLAG_CC2

Capture/Compare 2 interrupt flag

TIM_FLAG_CC3

Capture/Compare 3 interrupt flag

TIM_FLAG_CC4

Capture/Compare 4 interrupt flag

TIM_FLAG_CC5

Capture/Compare 5 interrupt flag

TIM_FLAG_CC6

Capture/Compare 6 interrupt flag

TIM_FLAG_COM

Commutation interrupt flag

TIM_FLAG_TRIGGER

Trigger interrupt flag

TIM_FLAG_BREAK

Break interrupt flag

TIM_FLAG_BREAK2

Break 2 interrupt flag

TIM_FLAG_SYSTEM_BREAK

System Break interrupt flag

TIM_FLAG_CC1OF

Capture 1 overcapture flag

TIM_FLAG_CC2OF

Capture 2 overcapture flag

TIM_FLAG_CC3OF

Capture 3 overcapture flag

TIM_FLAG_CC4OF

Capture 4 overcapture flag

TIM_FLAG_IDX

Encoder index flag

TIM_FLAG_DIR

Direction change flag

TIM_FLAG_IERR

Index error flag

TIM_FLAG_TERR

Transition error flag

TIM Group Channel 5 and Channel 1, 2 or 3

TIM_GROUPCH5_NONE

No effect of OC5REF on OC1REFC, OC2REFC and OC3REFC

TIM_GROUPCH5_OC1REFC

OC1REFC is the logical AND of OC1REFC and OC5REF

TIM_GROUPCH5_OC2REFC

OC2REFC is the logical AND of OC2REFC and OC5REF

TIM_GROUPCH5_OC3REFC

OC3REFC is the logical AND of OC3REFC and OC5REF

TIM Input Capture Polarity

TIM_ICPOLARITY_RISING

Capture triggered by rising edge on timer input

TIM_ICPOLARITY_FALLING

Capture triggered by falling edge on timer input

TIM_ICPOLARITY_BOTHEDGE

Capture triggered by both rising and falling edges on timer input

TIM Input Capture Prescaler

TIM_ICPSC_DIV1

Capture performed each time an edge is detected on the capture input

TIM_ICPSC_DIV2

Capture performed once every 2 events

TIM_ICPSC_DIV4

Capture performed once every 4 events

TIM_ICPSC_DIV8

Capture performed once every 8 events

TIM Input Capture Selection

TIM_ICSELECTION_DIRECTTI

TIM Input 1, 2, 3 or 4 is selected to be connected to IC1, IC2, IC3 or IC4, respectively

TIM_ICSELECTION_INDIRECTTI

TIM Input 1, 2, 3 or 4 is selected to be connected to IC2, IC1, IC4 or IC3, respectively

TIM_ICSELECTION_TRC

TIM Input 1, 2, 3 or 4 is selected to be connected to TRC

TIM Input Channel polarity

TIM_INPUTCHANNELPOLARITY_RISING

Polarity for TIx source

TIM_INPUTCHANNELPOLARITY_FALLING

Polarity for TIx source

TIM_INPUTCHANNELPOLARITY_BOTHEDGE

Polarity for TIx source

TIM interrupt Definition

TIM_IT_UPDATE

Update interrupt

TIM_IT_CC1

Capture/Compare 1 interrupt

TIM_IT_CC2

Capture/Compare 2 interrupt

TIM_IT_CC3

Capture/Compare 3 interrupt

TIM_IT_CC4

Capture/Compare 4 interrupt

TIM_IT_COM

Commutation interrupt

TIM_IT_TRIGGER

Trigger interrupt

TIM_IT_BREAK

Break interrupt

TIM_IT_IDX

Index interrupt

TIM_IT_DIR

Direction change interrupt

TIM_IT_IERR

Index error interrupt

TIM_IT_TERR

Transition error interrupt

TIM Lock level

TIM_LOCKLEVEL_OFF

LOCK OFF

TIM_LOCKLEVEL_1

LOCK Level 1

TIM_LOCKLEVEL_2

LOCK Level 2

TIM_LOCKLEVEL_3

LOCK Level 3

TIM Master Mode Selection

TIM_TRGO_RESET

TIMx_EGR.UG bit is used as trigger output (TRGO)

TIM_TRGO_ENABLE

TIMx_CR1.CEN bit is used as trigger output (TRGO)

TIM_TRGO_UPDATE

Update event is used as trigger output (TRGO)

TIM_TRGO_OC1

Capture or a compare match 1 is used as trigger output (TRGO)

TIM_TRGO_OC1REF

OC1REF signal is used as trigger output (TRGO)

TIM_TRGO_OC2REF

OC2REF signal is used as trigger output(TRGO)

TIM_TRGO_OC3REF

OC3REF signal is used as trigger output(TRGO)

TIM_TRGO_OC4REF

OC4REF signal is used as trigger output(TRGO)

TIM_TRGO_ENCODER_CLK

Encoder clock is used as trigger output(TRGO)

TIM Master Mode Selection 2 (TRGO2)

TIM_TRGO2_RESET

TIMx_EGR.UG bit is used as trigger output (TRGO2)

TIM_TRGO2_ENABLE

TIMx_CR1.CEN bit is used as trigger output (TRGO2)

TIM_TRGO2_UPDATE

Update event is used as trigger output (TRGO2)

TIM_TRGO2_OC1

Capture or a compare match 1 is used as trigger output (TRGO2)

TIM_TRGO2_OC1REF

OC1REF signal is used as trigger output (TRGO2)

TIM_TRGO2_OC2REF

OC2REF signal is used as trigger output (TRGO2)

TIM_TRGO2_OC3REF

OC3REF signal is used as trigger output (TRGO2)

TIM_TRGO2_OC4REF

OC4REF signal is used as trigger output (TRGO2)

TIM_TRGO2_OC5REF

OC5REF signal is used as trigger output (TRGO2)

TIM_TRGO2_OC6REF

OC6REF signal is used as trigger output (TRGO2)

TIM_TRGO2_OC4REF_RISINGFALLING

OC4REF rising or falling edges generate pulses on TRGO2

TIM_TRGO2_OC6REF_RISINGFALLING

OC6REF rising or falling edges generate pulses on TRGO2

TIM_TRGO2_OC4REF_RISING_OC6REF_RISING

OC4REF or OC6REF rising edges generate pulses on TRGO2

TIM_TRGO2_OC4REF_RISING_OC6REF_FALLING

OC4REF rising or OC6REF falling edges generate pulses on TRGO2

TIM_TRGO2_OC5REF_RISING_OC6REF_RISING

OC5REF or OC6REF rising edges generate pulses on TRGO2

TIM_TRGO2_OC5REF_RISING_OC6REF_FALLING

OC5REF or OC6REF rising edges generate pulses on TRGO2

TIM Master/Slave Mode

TIM_MASTERSLAVEMODE_ENABLE

No action

TIM_MASTERSLAVEMODE_DISABLE

Master/slave mode is selected

TIM One Pulse Mode

TIM_OPMODE_SINGLE

Counter stops counting at the next update event

TIM_OPMODE_REPETITIVE

Counter is not stopped at update event

TIM OSSI OffState Selection for Idle mode state

TIM_OSSI_ENABLE

When inactive, OC/OCN outputs are enabled (still controlled by the timer)

TIM_OSSI_DISABLE

When inactive, OC/OCN outputs are disabled (not controlled any longer by the timer)

TIM OSSR OffState Selection for Run mode state

TIM_OSSR_ENABLE

When inactive, OC/OCN outputs are enabled (still controlled by the timer)

TIM_OSSR_DISABLE

When inactive, OC/OCN outputs are disabled (not controlled any longer by the timer)

TIM Output Compare and PWM Modes

TIM_OCMODE_TIMING

Frozen

TIM_OCMODE_ACTIVE

Set channel to active level on match

TIM_OCMODE_INACTIVE

Set channel to inactive level on match

TIM_OCMODE_TOGGLE

Toggle

TIM_OCMODE_PWM1

PWM mode 1

TIM_OCMODE_PWM2

PWM mode 2

TIM_OCMODE_FORCED_ACTIVE

Force active level

TIM_OCMODE_FORCED_INACTIVE

Force inactive level

TIM_OCMODE_RETRIGERRABLE_OPM1

Retrigerrable OPM mode 1

TIM_OCMODE_RETRIGERRABLE_OPM2

Retrigerrable OPM mode 2

TIM_OCMODE_COMBINED_PWM1

Combined PWM mode 1

TIM_OCMODE_COMBINED_PWM2

Combined PWM mode 2

TIM_OCMODE_ASYMMETRIC_PWM1

Asymmetric PWM mode 1

TIM_OCMODE_ASYMMETRIC_PWM2

Asymmetric PWM mode 2

TIM_OCMODE_PULSE_ON_COMPARE

Pulse on compare (CH3&CH4 only)

TIM_OCMODE_DIRECTION_OUTPUT

Direction output (CH3&CH4 only)

TIM Output Compare Idle State

TIM_OCIDLESTATE_SET

Output Idle state: OCx=1 when MOE=0

TIM_OCIDLESTATE_RESET

Output Idle state: OCx=0 when MOE=0

TIM Complementary Output Compare Idle State

TIM_OCNIDLESTATE_SET

Complementary output Idle state: OCxN=1 when MOE=0

TIM_OCNIDLESTATE_RESET

Complementary output Idle state: OCxN=0 when MOE=0

TIM Complementary Output Compare Polarity

TIM_OCNPOLARITY_HIGH

Capture/Compare complementary output polarity

TIM_OCNPOLARITY_LOW

Capture/Compare complementary output polarity

TIM Complementary Output Compare State

TIM_OUTPUTNSTATE_DISABLE

OCxN is disabled

TIM_OUTPUTNSTATE_ENABLE

OCxN is enabled

TIM Output Compare Polarity

TIM_OCPOLARITY_HIGH

Capture/Compare output polarity

TIM_OCPOLARITY_LOW

Capture/Compare output polarity

TIM Output Compare State

TIM_OUTPUTSTATE_DISABLE

Capture/Compare 1 output disabled

TIM_OUTPUTSTATE_ENABLE

Capture/Compare 1 output enabled

TIM Output Fast State

TIM_OCFAST_DISABLE

Output Compare fast disable

TIM_OCFAST_ENABLE

Output Compare fast enable

TIM Slave mode

TIM_SLAVEMODE_DISABLE

Slave mode disabled

TIM_SLAVEMODE_RESET

Reset Mode

TIM_SLAVEMODE_GATED

Gated Mode

TIM_SLAVEMODE_TRIGGER

Trigger Mode

TIM_SLAVEMODE_EXTERNAL1

External Clock Mode 1

TIM_SLAVEMODE_COMBINED_RESETTRIGGER

Combined reset + trigger mode

TIM_SLAVEMODE_COMBINED_GATEDRESET

Combined gated + reset mode

TIM TI1 Input Selection

TIM_TI1SELECTION_CH1

The TIMx_CH1 pin is connected to TI1 input

TIM_TI1SELECTION_XORCOMBINATION

The TIMx_CH1, CH2 and CH3 pins are connected to the TI1 input (XOR combination)

TIM Trigger Polarity

TIM_TRIGGERPOLARITY_INVERTED

Polarity for ETRx trigger sources

TIM_TRIGGERPOLARITY_NONINVERTED

Polarity for ETRx trigger sources

TIM_TRIGGERPOLARITY_RISING

Polarity for TIxFPx or TI1_ED trigger sources

TIM_TRIGGERPOLARITY_FALLING

Polarity for TIxFPx or TI1_ED trigger sources

TIM_TRIGGERPOLARITY_BOTHEDGE

Polarity for TIxFPx or TI1_ED trigger sources

TIM Trigger Prescaler

TIM_TRIGGERPRESCALER_DIV1

No prescaler is used

TIM_TRIGGERPRESCALER_DIV2

Prescaler for External ETR Trigger: Capture performed once every 2 events.

TIM_TRIGGERPRESCALER_DIV4

Prescaler for External ETR Trigger: Capture performed once every 4 events.

TIM_TRIGGERPRESCALER_DIV8

Prescaler for External ETR Trigger: Capture performed once every 8 events.

TIM Trigger Selection

TIM_TS_ITR0

Internal Trigger 0 (ITR0)

TIM_TS_ITR1

Internal Trigger 1 (ITR1)

TIM_TS_ITR2

Internal Trigger 2 (ITR2)

TIM_TS_ITR3

Internal Trigger 3 (ITR3)

TIM_TS_ITR4

Internal Trigger 4 (ITR4)

TIM_TS_ITR5

Internal Trigger 5 (ITR5)

TIM_TS_ITR6

Internal Trigger 6 (ITR6)

TIM_TS_ITR7

Internal Trigger 7 (ITR7)

TIM_TS_ITR8

Internal Trigger 8 (ITR8)

TIM_TS_ITR9

Internal Trigger 9 (ITR9)

TIM_TS_ITR10

Internal Trigger 10 (ITR10)

TIM_TS_ITR11

Internal Trigger 11 (ITR11)

TIM_TS_ITR12

Internal Trigger 12 (ITR12)

TIM_TS_ITR13

Internal Trigger 13 (ITR13)

TIM_TS_TI1F_ED

TI1 Edge Detector (TI1F_ED)

TIM_TS_TI1FP1

Filtered Timer Input 1 (TI1FP1)

TIM_TS_TI2FP2

Filtered Timer Input 2 (TI2FP2)

TIM_TS_ETRF

Filtered External Trigger input (ETRF)

TIM_TS_NONE

No trigger selected

TIM Update Interrupt Flag Remap

TIM_UIFREMAP_DISABLE

Update interrupt flag remap disabled

TIM_UIFREMAP_ENABLE

Update interrupt flag remap enabled

78 HAL TIM Extension Driver

78.1 TIMEx Firmware driver registers structures

78.1.1 TIM_HallSensor_InitTypeDef

TIM_HallSensor_InitTypeDef is defined in the stm32n6xx_hal_tim_ex.h

Data Fields

- ***uint32_t IC1Polarity***
- ***uint32_t IC1Prescaler***
- ***uint32_t IC1Filter***
- ***uint32_t Commutation_Delay***

Field Documentation

- ***uint32_t TIM_HallSensor_InitTypeDef::IC1Polarity***
  Specifies the active edge of the input signal. This parameter can be a value of ***TIM_Input_Capture_Polarity***
- ***uint32_t TIM_HallSensor_InitTypeDef::IC1Prescaler***
  Specifies the Input Capture Prescaler. This parameter can be a value of ***TIM_Input_Capture_Prescaler***
- ***uint32_t TIM_HallSensor_InitTypeDef::IC1Filter***
  Specifies the input capture filter. This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF
- ***uint32_t TIM_HallSensor_InitTypeDef::Commutation_Delay***
  Specifies the pulse value to be loaded into the Capture Compare Register. This parameter can be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF

78.1.2 TIMEx_BreakInputConfigTypeDef

TIMEx_BreakInputConfigTypeDef is defined in the stm32n6xx_hal_tim_ex.h

Data Fields

- ***uint32_t Source***
- ***uint32_t Enable***
- ***uint32_t Polarity***

Field Documentation

- ***uint32_t TIMEx_BreakInputConfigTypeDef::Source***
  Specifies the source of the timer break input. This parameter can be a value of ***TIMEx_Break_Input_Source***
- ***uint32_t TIMEx_BreakInputConfigTypeDef::Enable***
  Specifies whether or not the break input source is enabled. This parameter can be a value of ***TIMEx_Break_Input_Source_Enable***
- ***uint32_t TIMEx_BreakInputConfigTypeDef::Polarity***
  Specifies the break input source polarity. This parameter can be a value of ***TIMEx_Break_Input_Source_Polarity***

78.1.3 TIMEx_EncoderIndexConfigTypeDef

TIMEx_EncoderIndexConfigTypeDef is defined in the stm32n6xx_hal_tim_ex.h

Data Fields

- ***uint32_t Polarity***
- ***uint32_t Prescaler***
- ***uint32_t Filter***
- ***uint32_t Blanking***
- ***FunctionalState FirstIndexEnable***
- ***uint32_t Position***
- ***uint32_t Direction***

Field Documentation

- ***uint32_t TIMEx_EncoderIndexConfigTypeDef::Polarity***
  TIM Encoder index polarity.This parameter can be a value of ***TIMEx_Encoder_Index_Polarity***
- ***uint32_t TIMEx_EncoderIndexConfigTypeDef::Prescaler***
  TIM Encoder index prescaler.This parameter can be a value of ***TIMEx_Encoder_Index_Prescaler***
- ***uint32_t TIMEx_EncoderIndexConfigTypeDef::Filter***
  TIM Encoder index filter.This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF
- ***uint32_t TIMEx_EncoderIndexConfigTypeDef::Blanking***
  Specifies whether or not the encoder index event is conditioned by TI3 or TI4 input.This parameter can be a value of ***TIMEx_Encoder_Index_Blanking***
- ***FunctionalState TIMEx_EncoderIndexConfigTypeDef::FirstIndexEnable***
  Specifies whether or not the encoder first index is enabled.This parameter value can be ENABLE or DISABLE.
- ***uint32_t TIMEx_EncoderIndexConfigTypeDef::Position***
  Specifies in which AB input configuration the index event resets the counter.This parameter can be a value of ***TIMEx_Encoder_Index_Position***
- ***uint32_t TIMEx_EncoderIndexConfigTypeDef::Direction***
  Specifies in which counter direction the index event resets the counter.This parameter can be a value of ***TIMEx_Encoder_Index_Direction***

78.2 TIMEx Firmware driver API description

The following section lists the various functions of the TIMEx library.

78.2.1 TIMER Extended features

The Timer Extended features include:

1. Complementary outputs with programmable dead-time for :
   – Output Compare
   – PWM generation (Edge and Center-aligned Mode)
   – One-pulse mode output
2. Synchronization circuit to control the timer with external signals and to interconnect several timers together.
3. Break input to put the timer output signals in reset state or in a known state.
4. Supports incremental (quadrature) encoder and hall-sensor circuitry for positioning purposes
5. In case of Pulse on compare, configure pulse length and delay
6. Encoder index configuration

78.2.2 How to use this driver

1. Initialize the TIM low level resources by implementing the following functions depending on the selected feature:
   – Hall Sensor output : HAL_TIMEx_HallSensor_MspInit()
2. Initialize the TIM low level resources :
   a. Enable the TIM interface clock using __HAL_RCC_TIMx_CLK_ENABLE();
   b. TIM pins configuration
      ◦ Enable the clock for the TIM GPIOs using the following function: __HAL_RCC_GPIOx_CLK_ENABLE();
      ◦ Configure these TIM pins in Alternate function mode using HAL_GPIO_Init();
3. The external Clock can be configured, if needed (the default clock is the internal clock from the APBx), using the following function: HAL_TIM_ConfigClockSource, the clock configuration should be done before any start function.
4. Configure the TIM in the desired functioning mode using one of the initialization function of this driver:
   – HAL_TIMEx_HallSensor_Init() and HAL_TIMEx_ConfigCommutEvent(): to use the Timer Hall Sensor Interface and the commutation event with the corresponding Interrupt and DMA request if needed (Note that One Timer is used to interface with the Hall sensor Interface and another Timer should be used to use the commutation event).

5. In case of Pulse On Compare:
   – HAL_TIMEx_OC_ConfigPulseOnCompare(): to configure pulse width and prescaler
6. Activate the TIM peripheral using one of the start functions:
   – Complementary Output Compare : HAL_TIMEx_OCN_Start(), HAL_TIMEx_OCN_Start_DMA(), HAL_TIMEx_OCN_Start_IT()
   – Complementary PWM generation : HAL_TIMEx_PWMN_Start(), HAL_TIMEx_PWMN_Start_DMA(), HAL_TIMEx_PWMN_Start_IT()
   – Complementary One-pulse mode output : HAL_TIMEx_OnePulseN_Start(), HAL_TIMEx_OnePulseN_Start_IT()
   – Hall Sensor output : HAL_TIMEx_HallSensor_Start(), HAL_TIMEx_HallSensor_Start_DMA(), HAL_TIMEx_HallSensor_Start_IT().

78.2.3 Detailed description of functions

HAL_TIMEx_HallSensor_Init

Function name

HAL_StatusTypeDef HAL_TIMEx_HallSensor_Init (TIM_HandleTypeDef * htim, const TIM_HallSensor_InitTypeDef * sConfig)

Function description

HAL_TIMEx_HallSensor_DeInit

Function name

HAL_StatusTypeDef HAL_TIMEx_HallSensor_DeInit (TIM_HandleTypeDef * htim)

Function description

HAL_TIMEx_HallSensor_MspInit

Function name

void HAL_TIMEx_HallSensor_MspInit (TIM_HandleTypeDef * htim)

Function description

HAL_TIMEx_HallSensor_MspDeInit

Function name

void HAL_TIMEx_HallSensor_MspDeInit (TIM_HandleTypeDef * htim)

Function description

HAL_TIMEx_HallSensor_Start

Function name

HAL_StatusTypeDef HAL_TIMEx_HallSensor_Start (TIM_HandleTypeDef * htim)

Function description

HAL_TIMEx_HallSensor_Stop

Function name

HAL_StatusTypeDef HAL_TIMEx_HallSensor_Stop (TIM_HandleTypeDef * htim)

Function description

HAL_TIMEx_HallSensor_Start_IT

Function name

HAL_StatusTypeDef HAL_TIMEx_HallSensor_Start_IT (TIM_HandleTypeDef * htim)

Function description

HAL_TIMEx_HallSensor_Stop_IT

Function name

HAL_StatusTypeDef HAL_TIMEx_HallSensor_Stop_IT (TIM_HandleTypeDef * htim)

Function description

HAL_TIMEx_HallSensor_Start_DMA

Function name

HAL_StatusTypeDef HAL_TIMEx_HallSensor_Start_DMA (TIM_HandleTypeDef * htim, uint32_t * pData, uint16_t Length)

Function description

HAL_TIMEx_HallSensor_Stop_DMA

Function name

HAL_StatusTypeDef HAL_TIMEx_HallSensor_Stop_DMA (TIM_HandleTypeDef * htim)

Function description

HAL_TIMEx_OCN_Start

Function name

HAL_StatusTypeDef HAL_TIMEx_OCN_Start (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

HAL_TIMEx_OCN_Stop

Function name

HAL_StatusTypeDef HAL_TIMEx_OCN_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

HAL_TIMEx_OCN_Start_IT

Function name

HAL_StatusTypeDef HAL_TIMEx_OCN_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

HAL_TIMEx_OCN_Stop_IT

Function name

HAL_StatusTypeDef HAL_TIMEx_OCN_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

HAL_TIMEx_OCN_Start_DMA

Function name

HAL_StatusTypeDef HAL_TIMEx_OCN_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, const uint32_t * pData, uint16_t Length)

Function description

HAL_TIMEx_OCN_Stop_DMA

Function name

HAL_StatusTypeDef HAL_TIMEx_OCN_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

HAL_TIMEx_PWMN_Start

Function name

HAL_StatusTypeDef HAL_TIMEx_PWMN_Start (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

HAL_TIMEx_PWMN_Stop

Function name

HAL_StatusTypeDef HAL_TIMEx_PWMN_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

HAL_TIMEx_PWMN_Start_IT

Function name

HAL_StatusTypeDef HAL_TIMEx_PWMN_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

HAL_TIMEx_PWMN_Stop_IT

Function name

HAL_StatusTypeDef HAL_TIMEx_PWMN_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

HAL_TIMEx_PWMN_Start_DMA

Function name

HAL_StatusTypeDef HAL_TIMEx_PWMN_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, const uint32_t * pData, uint16_t Length)

Function description

HAL_TIMEx_PWMN_Stop_DMA

Function name

HAL_StatusTypeDef HAL_TIMEx_PWMN_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

HAL_TIMEx_OnePulseN_Start

Function name

HAL_StatusTypeDef HAL_TIMEx_OnePulseN_Start (TIM_HandleTypeDef * htim, uint32_t OutputChannel)

Function description

HAL_TIMEx_OnePulseN_Stop

Function name

HAL_StatusTypeDef HAL_TIMEx_OnePulseN_Stop (TIM_HandleTypeDef * htim, uint32_t OutputChannel)

Function description

HAL_TIMEx_OnePulseN_Start_IT

Function name

HAL_StatusTypeDef HAL_TIMEx_OnePulseN_Start_IT (TIM_HandleTypeDef * htim, uint32_t OutputChannel)

Function description

HAL_TIMEx_OnePulseN_Stop_IT

Function name

HAL_StatusTypeDef HAL_TIMEx_OnePulseN_Stop_IT (TIM_HandleTypeDef * htim, uint32_t OutputChannel)

Function description

HAL_TIMEx_ConfigCommutEvent

Function name

HAL_StatusTypeDef HAL_TIMEx_ConfigCommutEvent (TIM_HandleTypeDef * htim, uint32_t InputTrigger, uint32_t CommutationSource)

Function description

HAL_TIMEx_ConfigCommutEvent_IT

Function name

HAL_StatusTypeDef HAL_TIMEx_ConfigCommutEvent_IT (TIM_HandleTypeDef * htim, uint32_t InputTrigger, uint32_t CommutationSource)

Function description

HAL_TIMEx_ConfigCommutEvent_DMA

Function name

HAL_StatusTypeDef HAL_TIMEx_ConfigCommutEvent_DMA (TIM_HandleTypeDef * htim, uint32_t InputTrigger, uint32_t CommutationSource)

Function description

HAL_TIMEx_MasterConfigSynchronization

Function name

HAL_StatusTypeDef HAL_TIMEx_MasterConfigSynchronization (TIM_HandleTypeDef * htim, const TIM_MasterConfigTypeDef * sMasterConfig)

Function description

HAL_TIMEx_ConfigBreakDeadTime

Function name

HAL_StatusTypeDef HAL_TIMEx_ConfigBreakDeadTime (TIM_HandleTypeDef * htim, const TIM_BreakDeadTimeConfigTypeDef * sBreakDeadTimeConfig)

Function description

HAL_TIMEx_ConfigBreakInput

Function name

HAL_StatusTypeDef HAL_TIMEx_ConfigBreakInput (TIM_HandleTypeDef * htim, uint32_t BreakInput, const TIMEx_BreakInputConfigTypeDef * sBreakInputConfig)

Function description

HAL_TIMEx_GroupChannel5

Function name

HAL_StatusTypeDef HAL_TIMEx_GroupChannel5 (TIM_HandleTypeDef * htim, uint32_t Channels)

Function description

HAL_TIMEx_RemapConfig

Function name

HAL_StatusTypeDef HAL_TIMEx_RemapConfig (TIM_HandleTypeDef * htim, uint32_t Remap)

Function description

HAL_TIMEx_TISelection

Function name

HAL_StatusTypeDef HAL_TIMEx_TISelection (TIM_HandleTypeDef * htim, uint32_t TISelection, uint32_t Channel)

Function description

HAL_TIMEx_DisarmBreakInput

Function name

HAL_StatusTypeDef HAL_TIMEx_DisarmBreakInput (TIM_HandleTypeDef * htim, uint32_t BreakInput)

Function description

HAL_TIMEx_ReArmBreakInput

Function name

HAL_StatusTypeDef HAL_TIMEx_ReArmBreakInput (const TIM_HandleTypeDef * htim, uint32_t BreakInput)

Function description

HAL_TIMEx_DitheringEnable

Function name

HAL_StatusTypeDef HAL_TIMEx_DitheringEnable (TIM_HandleTypeDef * htim)

Function description

HAL_TIMEx_DitheringDisable

Function name

HAL_StatusTypeDef HAL_TIMEx_DitheringDisable (TIM_HandleTypeDef * htim)

Function description

HAL_TIMEx_OC_ConfigPulseOnCompare

Function name

HAL_StatusTypeDef HAL_TIMEx_OC_ConfigPulseOnCompare (TIM_HandleTypeDef * htim, uint32_t PulseWidthPrescaler, uint32_t PulseWidth)

Function description

HAL_TIMEx_ConfigSlaveModePreload

Function name

HAL_StatusTypeDef HAL_TIMEx_ConfigSlaveModePreload (TIM_HandleTypeDef * htim, uint32_t Source)

Function description

HAL_TIMEx_EnableSlaveModePreload

Function name

HAL_StatusTypeDef HAL_TIMEx_EnableSlaveModePreload (TIM_HandleTypeDef * htim)

Function description

HAL_TIMEx_DisableSlaveModePreload

Function name

HAL_StatusTypeDef HAL_TIMEx_DisableSlaveModePreload (TIM_HandleTypeDef * htim)

Function description

HAL_TIMEx_EnableDeadTimePreload

Function name

HAL_StatusTypeDef HAL_TIMEx_EnableDeadTimePreload (TIM_HandleTypeDef * htim)

Function description

HAL_TIMEx_DisableDeadTimePreload

Function name

HAL_StatusTypeDef HAL_TIMEx_DisableDeadTimePreload (TIM_HandleTypeDef * htim)

Function description

HAL_TIMEx_ConfigDeadTime

Function name

HAL_StatusTypeDef HAL_TIMEx_ConfigDeadTime (TIM_HandleTypeDef * htim, uint32_t Deadtime)

Function description

HAL_TIMEx_ConfigAsymmetricalDeadTime

Function name

HAL_StatusTypeDef HAL_TIMEx_ConfigAsymmetricalDeadTime (TIM_HandleTypeDef * htim, uint32_t FallingDeadtime)

Function description

HAL_TIMEx_EnableAsymmetricalDeadTime

Function name

HAL_StatusTypeDef HAL_TIMEx_EnableAsymmetricalDeadTime (TIM_HandleTypeDef * htim)

Function description

HAL_TIMEx_DisableAsymmetricalDeadTime

Function name

HAL_StatusTypeDef HAL_TIMEx_DisableAsymmetricalDeadTime (TIM_HandleTypeDef * htim)

Function description

HAL_TIMEx_ConfigEncoderIndex

Function name

HAL_StatusTypeDef HAL_TIMEx_ConfigEncoderIndex (TIM_HandleTypeDef * htim, TIMEx_EncoderIndexConfigTypeDef * sEncoderIndexConfig)

Function description

HAL_TIMEx_EnableEncoderIndex

Function name

HAL_StatusTypeDef HAL_TIMEx_EnableEncoderIndex (TIM_HandleTypeDef * htim)

Function description

HAL_TIMEx_DisableEncoderIndex

Function name

HAL_StatusTypeDef HAL_TIMEx_DisableEncoderIndex (TIM_HandleTypeDef * htim)

Function description

HAL_TIMEx_EnableEncoderFirstIndex

Function name

HAL_StatusTypeDef HAL_TIMEx_EnableEncoderFirstIndex (TIM_HandleTypeDef * htim)

Function description

HAL_TIMEx_DisableEncoderFirstIndex

Function name

HAL_StatusTypeDef HAL_TIMEx_DisableEncoderFirstIndex (TIM_HandleTypeDef * htim)

Function description

HAL_TIMEx_EnableADCSynchronization

Function name

HAL_StatusTypeDef HAL_TIMEx_EnableADCSynchronization (TIM_HandleTypeDef * htim)

Function description

HAL_TIMEx_DisableADCSynchronization

Function name

HAL_StatusTypeDef HAL_TIMEx_DisableADCSynchronization (TIM_HandleTypeDef * htim)

Function description

HAL_TIMEx_CommutCallback

Function name

void HAL_TIMEx_CommutCallback (TIM_HandleTypeDef * htim)

Function description

HAL_TIMEx_CommutHalfCpltCallback

Function name

void HAL_TIMEx_CommutHalfCpltCallback (TIM_HandleTypeDef * htim)

Function description

HAL_TIMEx_BreakCallback

Function name

void HAL_TIMEx_BreakCallback (TIM_HandleTypeDef * htim)

Function description

HAL_TIMEx_Break2Callback

Function name

void HAL_TIMEx_Break2Callback (TIM_HandleTypeDef * htim)

Function description

HAL_TIMEx_EncoderIndexCallback

Function name

void HAL_TIMEx_EncoderIndexCallback (TIM_HandleTypeDef * htim)

Function description

HAL_TIMEx_DirectionChangeCallback

Function name

void HAL_TIMEx_DirectionChangeCallback (TIM_HandleTypeDef * htim)

Function description

HAL_TIMEx_IndexErrorCallback

Function name

void HAL_TIMEx_IndexErrorCallback (TIM_HandleTypeDef * htim)

Function description

HAL_TIMEx_TransitionErrorCallback

Function name

void HAL_TIMEx_TransitionErrorCallback (TIM_HandleTypeDef * htim)

Function description

HAL_TIMEx_HallSensor_GetState

Function name

HAL_TIM_StateTypeDef HAL_TIMEx_HallSensor_GetState (const TIM_HandleTypeDef * htim)

Function description

HAL_TIMEx_GetChannelNState

Function name

HAL_TIM_ChannelStateTypeDef HAL_TIMEx_GetChannelNState (const TIM_HandleTypeDef * htim, uint32_t ChannelN)

Function description

TIMEx_DMACommutationCplt

Function name

void TIMEx_DMACommutationCplt (DMA_HandleTypeDef * hdma)

Function description

TIMEx_DMACommutationHalfCplt

Function name

void TIMEx_DMACommutationHalfCplt (DMA_HandleTypeDef * hdma)

Function description

78.3 TIMEx Firmware driver defines

The following section lists the various define and macros of the module.

78.3.1 TIMEx

TIMEx

TIM Extended Break input

TIM_BREAKINPUT_BRK

Timer break input

TIM_BREAKINPUT_BRK2

Timer break2 input

TIM Extended Break input source

TIM_BREAKINPUTSOURCE_BKIN

An external source (GPIO) is connected to the BKIN pin

TIM_BREAKINPUTSOURCE_MDF1

The analog watchdog output of the MDF1 peripheral is connected to the break input

TIM Extended Break input source enabling

TIM_BREAKINPUTSOURCE_DISABLE

Break input source is disabled

TIM_BREAKINPUTSOURCE_ENABLE

Break input source is enabled

TIM Extended Break input polarity

TIM_BREAKINPUTSOURCE_POLARITY_LOW

Break input source is active low

TIM_BREAKINPUTSOURCE_POLARITY_HIGH

Break input source is active_high

TIM Extended Encoder index blanking

TIM_ENCODERINDEX_BLANKING_DISABLE

Encoder index blanking is disabled

TIM_ENCODERINDEX_BLANKING_TI3

Encoder index blanking is enabled on TI3

TIM_ENCODERINDEX_BLANKING_TI4

Encoder index blanking is enabled on TI4

TIM Extended Encoder index direction

TIM_ENCODERINDEX_DIRECTION_UP_DOWN

Index resets the counter whatever the direction

TIM_ENCODERINDEX_DIRECTION_UP

Index resets the counter when up-counting only

TIM_ENCODERINDEX_DIRECTION_DOWN

Index resets the counter when down-counting only

TIM Extended Encoder index polarity

TIM_ENCODERINDEX_POLARITY_INVERTED

Polarity for ETRx pin

TIM_ENCODERINDEX_POLARITY_NONINVERTED

Polarity for ETRx pin

TIM Extended Encoder index position

TIM_ENCODERINDEX_POSITION_00

Encoder index position is AB=00

TIM_ENCODERINDEX_POSITION_01

Encoder index position is AB=01

TIM_ENCODERINDEX_POSITION_10

Encoder index position is AB=10

TIM_ENCODERINDEX_POSITION_11

Encoder index position is AB=11

TIM_ENCODERINDEX_POSITION_0

In directional clock mode or clock plus direction mode, index resets the counter when clock is 0

TIM_ENCODERINDEX_POSITION_1

In directional clock mode or clock plus direction mode, index resets the counter when clock is 1

TIM Extended Encodder index prescaler

TIM_ENCODERINDEX_PRESCALER_DIV1

No prescaler is used

TIM_ENCODERINDEX_PRESCALER_DIV2

Prescaler for External ETR pin: Capture performed once every 2 events.

TIM_ENCODERINDEX_PRESCALER_DIV4

Prescaler for External ETR pin: Capture performed once every 4 events.

TIM_ENCODERINDEX_PRESCALER_DIV8

Prescaler for External ETR pin: Capture performed once every 8 events.

TIM Extended Exported Macros

__HAL_TIM_CALC_PSC

Description:

- HELPER macro calculating the prescaler value to achieve the required counter clock frequency.

Parameters:

- __TIMCLK__: timer input clock frequency (in Hz)
- __CNTCLK__: counter clock frequency (in Hz)

Return value:

- Prescaler: value (between Min_Data=0 and Max_Data=65535)

Notes:

- ex: __HAL_TIM_CALC_PSC(80000000, 1000000);

__HAL_TIM_CALC_PERIOD

Description:

- HELPER macro calculating the auto-reload value to achieve the required output signal frequency.

Parameters:

- __TIMCLK__: timer input clock frequency (in Hz)
- __PSC__: prescaler
- __FREQ__: output signal frequency (in Hz)

Return value:

- Auto-reload: value (between Min_Data=0 and Max_Data=65535)

Notes:

- ex: __HAL_TIM_CALC_PERIOD(1000000, 0, 10000);

__HAL_TIM_CALC_PERIOD_DITHER

Description:

- HELPER macro calculating the auto-reload value, with dithering feature enabled, to achieve the required output signal frequency.

Parameters:

- __TIMCLK__: timer input clock frequency (in Hz)
- __PSC__: prescaler
- __FREQ__: output signal frequency (in Hz)

Return value:

- Auto-reload: value (between Min_Data=0 and Max_Data=65519)

Notes:

- ex: __HAL_TIM_CALC_PERIOD_DITHER(1000000, 0, 10000); This macro should be used only if dithering is already enabled

__HAL_TIM_CALC_PULSE

Description:

- HELPER macro calculating the compare value required to achieve the required timer output compare active/inactive delay.

Parameters:

- __TIMCLK__: timer input clock frequency (in Hz)
- __PSC__: prescaler
- __DELAY__: timer output compare active/inactive delay (in us)

Return value:

- Compare: value (between Min_Data=0 and Max_Data=65535)

Notes:

- ex: __HAL_TIM_CALC_PULSE(1000000, 0, 10);

__HAL_TIM_CALC_PULSE_DITHER

Description:

- HELPER macro calculating the compare value, with dithering feature enabled, to achieve the required timer output compare active/inactive delay.

Parameters:

- __TIMCLK__: timer input clock frequency (in Hz)
- __PSC__: prescaler
- __DELAY__: timer output compare active/inactive delay (in us)

Return value:

- Compare: value (between Min_Data=0 and Max_Data=65519)

Notes:

- ex: __HAL_TIM_CALC_PULSE_DITHER(1000000, 0, 10); This macro should be used only if dithering is already enabled

__HAL_TIM_CALC_PERIOD_BY_DELAY

Description:

- HELPER macro calculating the auto-reload value to achieve the required pulse duration (when the timer operates in one pulse mode).

Parameters:

- __TIMCLK__: timer input clock frequency (in Hz)
- __PSC__: prescaler
- __DELAY__: timer output compare active/inactive delay (in us)
- __PULSE__: pulse duration (in us)

Return value:

- Auto-reload: value (between Min_Data=0 and Max_Data=65535)

Notes:

- ex: __HAL_TIM_CALC_PERIOD_BY_DELAY(1000000, 0, 10, 20);

__HAL_TIM_CALC_PERIOD_DITHER_BY_DELAY

Description:

- HELPER macro calculating the auto-reload value, with dithering feature enabled, to achieve the required pulse duration (when the timer operates in one pulse mode).

Parameters:

- __TIMCLK__: timer input clock frequency (in Hz)
- __PSC__: prescaler
- __DELAY__: timer output compare active/inactive delay (in us)
- __PULSE__: pulse duration (in us)

Return value:

- Auto-reload: value (between Min_Data=0 and Max_Data=65519)

Notes:

- ex: __HAL_TIM_CALC_PERIOD_DITHER_BY_DELAY(1000000, 0, 10, 20); This macro should be used only if dithering is already enabled

TIM Extended Remapping

TIM_TIM1_ETR_GPIO

TIM1_ETR is connected to I/O

TIM_TIM1_ETR_ADC1_AWD1

TIM1 ETR is connected to ADC1 AWD1

TIM_TIM1_ETR_ADC1_AWD2

TIM1 ETR is connected to ADC1 AWD2

TIM_TIM1_ETR_ADC1_AWD3

TIM1 ETR is connected to ADC1 AWD3

TIM_TIM1_ETR_ADC2_AWD1

TIM1 ETR is connected to ADC2 AWD1

TIM_TIM1_ETR_ADC2_AWD2

TIM1 ETR is connected to ADC2 AWD2

TIM_TIM1_ETR_ADC2_AWD3

TIM1 ETR is connected to ADC2 AWD3

TIM_TIM2_ETR_GPIO

TIM2 ETR is connected to GPIO

TIM_TIM2_ETR_DCMIPP_HSYNC

TIM2_ETR is connected to DCMIPP HSYNC

TIM_TIM2_ETR_LCD_HSYNC

TIM2_ETR is connected to LCD HSYNC

TIM_TIM2_ETR_SAI1_FSA

TIM2_ETR is connected to SAI1 FS_A

TIM_TIM2_ETR_SAI1_FSB

TIM2_ETR is connected to SAI1 FS_B

TIM_TIM2_ETR_GFXTIM_TE

TIM2_ETR is connected to GFXTIM TE

TIM_TIM2_ETR_DCMIPP_VSYNC

TIM2_ETR is connected to DCMIPP VSYNC

TIM_TIM2_ETR_LCD_VSYNC

TIM2_ETR is connected to LCD VSYNC

TIM_TIM2_ETR_TIM3_ETR

TIM2_ETR is connected to TIM3_ETR

TIM_TIM2_ETR_TIM4_ETR

TIM2_ETR is connected to TIM4_ETR

TIM_TIM2_ETR_TIM5_ETR

TIM2_ETR is connected to TIM5_ETR

TIM_TIM2_ETR_ETH1_PPS

TIM2_ETR is connected to ETH1 PPS

TIM_TIM2_ETR_USB1_SOF

TIM2_ETR is connected to USB1 OTG SOF

TIM_TIM2_ETR_USB2_SOF

TIM2_ETR is connected to USB2 OTG SOF

TIM_TIM3_ETR_GPIO

TIM3 ETR is connected to GPIO

TIM_TIM3_ETR_DCMIPP_HSYNC

TIM3 ETR is connected to DCMIPP HSYNC

TIM_TIM3_ETR_LCD_HSYNC

TIM3 ETR is connected to LCD HSYNC

TIM_TIM3_ETR_GFXTIM_TE

TIM3 ETR is connected to GFXTIM TE

TIM_TIM3_ETR_DCMIPP_VSYNC

TIM3 ETR is connected to DCMIPP VSYNC

TIM_TIM3_ETR_LCD_VSYNC

TIM3 ETR is connected to LCD VSYNC

TIM_TIM3_ETR_TIM2_ETR

TIM3 ETR is connected to TIM2 ETR

TIM_TIM3_ETR_TIM4_ETR

TIM3 ETR is connected to TIM4_ETR

TIM_TIM3_ETR_TIM5_ETR

TIM3 ETR is connected to TIM5_ETR

TIM_TIM3_ETR_ETH1_PPS

TIM3_ETR is connected to ETH1 PPS

TIM_TIM4_ETR_GPIO

TIM4 ETR is connected to GPIO

TIM_TIM4_ETR_DCMIPP_HSYNC

TIM4 ETR is connected to DCMIPP HSYNC

TIM_TIM4_ETR_LCD_HSYNC

TIM4 ETR is connected to LCD HSYNC

TIM_TIM4_ETR_GFXTIM_TE

TIM4 ETR is connected to GFXTIM TE

TIM_TIM4_ETR_DCMIPP_VSYNC

TIM4 ETR is connected to DCMIPP VSYNC

TIM_TIM4_ETR_LCD_VSYNC

TIM4 ETR is connected to LCD VSYNC

TIM_TIM4_ETR_TIM2_ETR

TIM4 ETR is connected to TIM2 ETR

TIM_TIM4_ETR_TIM3_ETR

TIM4 ETR is connected to TIM3_ETR

TIM_TIM4_ETR_TIM5_ETR

TIM4 ETR is connected to TIM5_ETR

TIM_TIM5_ETR_GPIO

TIM5 ETR is connected to GPIO

TIM_TIM5_ETR_SAI2_FSA

TIM5_ETR is connected to SAI2 FS_A

TIM_TIM5_ETR_SAI2_FSB

TIM5_ETR is connected to SAI2 FS_B

TIM_TIM5_ETR_DCMIPP_HSYNC

TIM5 ETR is connected to DCMIPP HSYNC

TIM_TIM5_ETR_LCD_HSYNC

TIM5 ETR is connected to LCD HSYNC

TIM_TIM5_ETR_GFXTIM_TE

TIM5 ETR is connected to GFXTIM TE

TIM_TIM5_ETR_DCMIPP_VSYNC

TIM5 ETR is connected to DCMIPP VSYNC

TIM_TIM5_ETR_LCD_VSYNC

TIM5 ETR is connected to LCD VSYNC

TIM_TIM5_ETR_TIM2_ETR

TIM5 ETR is connected to TIM2_ETR

TIM_TIM5_ETR_TIM3_ETR

TIM5 ETR is connected to TIM3_ETR

TIM_TIM5_ETR_TIM4_ETR

TIM5 ETR is connected to TIM4_ETR

TIM_TIM5_ETR_USB1_SOF

TIM5_ETR is connected to USB1 OTG SOF

TIM_TIM5_ETR_USB2_SOF

TIM5_ETR is connected to USB2 OTG SOF

TIM_TIM8_ETR_GPIO

TIM8 ETR is connected to GPIO

TIM_TIM8_ETR_ADC2_AWD1

TIM8 ETR is connected to ADC2 AWD1

TIM_TIM8_ETR_ADC2_AWD2

TIM8 ETR is connected to ADC2 AWD2

TIM_TIM8_ETR_ADC2_AWD3

TIM8 ETR is connected to ADC2 AWD3

TIM Extended Bitfield SMS preload enabling

TIM_SMS_PRELOAD_SOURCE_UPDATE

Prelaod of SMS bitfield is disabled

TIM_SMS_PRELOAD_SOURCE_INDEX

Preload of SMS bitfield is enabled

TIM Extended Timer input selection

TIM_TIM2_TI1_GPIO

TIM2 TI1 is connected to GPIO

TIM_TIM2_TI1_ETH1_PPS

TIM2 TI1 is connected to ETH1 PPS

TIM_TIM3_TI1_GPIO

TIM3 TI1 is connected to GPIO

TIM_TIM3_TI1_ETH1_PPS

TIM3 TI1 is connected to ETH1 PPS

TIM_TIM3_TI1_FDCAN_RTP

TIM3_TI1 is connected to FDCAN RTP

TIM_TIM3_TI1_FDCAN_TMP

TIM3_TI1 is connected to FDCAN TMP

TIM_TIM3_TI1_FDCAN_SOC

TIM3_TI1 is connected to FDCAN SOC

TIM_TIM5_TI1_GPIO

TIM5 TI1 is connected to GPIO

TIM_TIM5_TI1_FDCAN_RTP

TIM5_TI1 is connected to FDCAN RTP

TIM_TIM5_TI1_FDCAN_TMP

TIM5_TI1 is connected to FDCAN TMP

TIM_TIM9_TI1_GPIO

TIM9 TI1 is connected to GPIO

TIM_TIM9_TI1_MCO1

TIM9 TI1 is connected to MCO1

TIM_TIM9_TI1_MCO2

TIM9 TI1 is connected to MCO2

TIM_TIM10_TI1_GPIO

TIM10_TI1 is connected to GPIO

TIM_TIM10_TI1_I3C1_IBIACK

TIM10_TI1 is connected to I3C1 IBI ACK

TIM_TIM11_TI1_GPIO

TIM11_TI1 is connected to GPIO

TIM_TIM11_TI1_I3C2_IBIACK

TIM11_TI1 is connected to I3C2 IBI ACK

TIM_TIM12_TI1_GPIO

TIM12 TI1 is connected to GPIO

TIM_TIM12_TI1_SPDIF_FS

TIM12 TI1 is connected to SPDIF FS

TIM_TIM12_TI1_HSI_1024

TIM12 TI1 is connected to HSI/1024

TIM_TIM12_TI1_MSI_128

TIM12_TI1 is connected to MSI/128

TIM_TIM12_TI1_MCO1

TIM12 TI1 is connected to MCO1

TIM_TIM12_TI1_MCO2

TIM12 TI1 is connected to MCO2

TIM_TIM13_TI1_GPIO

TIM13_TI1 is connected to GPIO

TIM_TIM13_TI1_I3C1_IBIACK

TIM13_TI1 is connected to I3C1 IBI ACK

TIM_TIM14_TI1_GPIO

TIM14_TI1 is connected to GPIO

TIM_TIM14_TI1_I3C2_IBIACK

TIM14_TI1 is connected to I3C2 IBI ACK

TIM_TIM15_TI1_GPIO

TIM15 TI1 is connected to GPIO

TIM_TIM15_TI1_TIM2_CH1

TIM15 TI1 is connected to TIM2 CH1

TIM_TIM15_TI1_TIM3_CH1

TIM15 TI1 is connected to TIM3 CH1

TIM_TIM15_TI1_TIM4_CH1

TIM15 TI1 is connected to TIM4 CH1

TIM_TIM15_TI1_MCO1

TIM15 TI1 is connected to MCO1

TIM_TIM15_TI1_MCO2

TIM15 TI1 is connected to MCO2

TIM_TIM15_TI2_GPIO

TIM15 TI2 is connected to GPIO

TIM_TIM15_TI2_TIM2_CH2

TIM15 TI2 is connected to TIM2 CH2

TIM_TIM15_TI2_TIM3_CH2

TIM15 TI2 is connected to TIM3 CH2

TIM_TIM15_TI2_TIM4_CH2

TIM15 TI2 is connected to TIM4 CH2

TIM_TIM16_TI1_GPIO

TIM16 TI1 is connected to GPIO

TIM_TIM16_TI1_LSI

TIM16 TI1 is connected to LSI

TIM_TIM16_TI1_LSE

TIM16 TI1 is connected to LSE

TIM_TIM16_TI1_RTC_WKUP

TIM16 TI1 is connected to RTC wakeup interrupt

TIM_TIM17_TI1_GPIO

TIM17 TI1 is connected to GPIO

TIM_TIM17_TI1_SPDIF_FS

TIM17 TI1 is connected to SPDIF FS

TIM_TIM17_TI1_HSE_1024

TIM17 TI1 is connected to HSE/1024

79 HAL UART Generic Driver

79.1 UART Firmware driver registers structures

79.1.1 UART_InitTypeDef

UART_InitTypeDef is defined in the stm32n6xx_hal_uart.h

Data Fields

- ***uint32_t BaudRate***
- ***uint32_t WordLength***
- ***uint32_t StopBits***
- ***uint32_t Parity***
- ***uint32_t Mode***
- ***uint32_t HwFlowCtl***
- ***uint32_t OverSampling***
- ***uint32_t OneBitSampling***
- ***uint32_t ClockPrescaler***

Field Documentation

- ***uint32_t UART_InitTypeDef::BaudRate***
  This member configures the UART communication baud rate. The baud rate register is computed using the following formula:
  Note:
  – For LPUART : Baud Rate Register = ((256 * lpuart_ker_ckpres) / ((huart->Init.BaudRate))) where lpuart_ker_ck_pres is the UART input clock divided by a prescaler.
  – For UART :
    ◦ If oversampling is 16 or in LIN mode, Baud Rate Register = ((uart_ker_ckpres) / ((huart->Init.BaudRate)))
    ◦ If oversampling is 8, Baud Rate Register[15:4] = ((2 * uart_ker_ckpres) / ((huart->Init.BaudRate)))[15:4] Baud Rate Register[3] = 0 Baud Rate Register[2:0] = (((2 * uart_ker_ckpres) / ((huart->Init.BaudRate)))[3:0]) >> 1 where uart_ker_ck_pres is the UART input clock divided by a prescaler
- ***uint32_t UART_InitTypeDef::WordLength***
  Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of ***UARTEx_Word_Length***.
- ***uint32_t UART_InitTypeDef::StopBits***
  Specifies the number of stop bits transmitted. This parameter can be a value of ***UART_Stop_Bits***.
- ***uint32_t UART_InitTypeDef::Parity***
  Specifies the parity mode. This parameter can be a value of ***UART_Parity***
  Note:
  – When parity is enabled, the computed parity is inserted at the MSB position of the transmitted data (9th bit when the word length is set to 9 data bits; 8th bit when the word length is set to 8 data bits).
- ***uint32_t UART_InitTypeDef::Mode***
  Specifies whether the Receive or Transmit mode is enabled or disabled. This parameter can be a value of ***UART_Mode***.
- ***uint32_t UART_InitTypeDef::HwFlowCtl***
  Specifies whether the hardware flow control mode is enabled or disabled. This parameter can be a value of ***UART_Hardware_Flow_Control***.
- ***uint32_t UART_InitTypeDef::OverSampling***
  Specifies whether the Over sampling 8 is enabled or disabled, to achieve higher speed (up to f_PCLK/8). This parameter can be a value of ***UART_Over_Sampling***.
- ***uint32_t UART_InitTypeDef::OneBitSampling***
  Specifies whether a single sample or three samples' majority vote is selected. Selecting the single sample method increases the receiver tolerance to clock deviations. This parameter can be a value of ***UART_OneBit_Sampling***.

- ***uint32_t UART_InitTypeDef::ClockPrescaler***
  Specifies the prescaler value used to divide the UART clock source. This parameter can be a value of ***UART_ClockPrescaler***.

79.1.2 UART_AdvFeatureInitTypeDef

UART_AdvFeatureInitTypeDef is defined in the stm32n6xx_hal_uart.h

Data Fields

- ***uint32_t AdvFeatureInit***
- ***uint32_t TxPinLevelInvert***
- ***uint32_t RxPinLevelInvert***
- ***uint32_t DataInvert***
- ***uint32_t Swap***
- ***uint32_t OverrunDisable***
- ***uint32_t DMADisableonRxError***
- ***uint32_t AutoBaudRateEnable***
- ***uint32_t AutoBaudRateMode***
- ***uint32_t MSBFirst***

Field Documentation

- ***uint32_t UART_AdvFeatureInitTypeDef::AdvFeatureInit***
  Specifies which advanced UART features is initialized. Several Advanced Features may be initialized at the same time . This parameter can be a value of ***UART_Advanced_Features_Initialization_Type***.
- ***uint32_t UART_AdvFeatureInitTypeDef::TxPinLevelInvert***
  Specifies whether the TX pin active level is inverted. This parameter can be a value of ***UART_Tx_Inv***.
- ***uint32_t UART_AdvFeatureInitTypeDef::RxPinLevelInvert***
  Specifies whether the RX pin active level is inverted. This parameter can be a value of ***UART_Rx_Inv***.
- ***uint32_t UART_AdvFeatureInitTypeDef::DataInvert***
  Specifies whether data are inverted (positive/direct logic vs negative/inverted logic). This parameter can be a value of ***UART_Data_Inv***.
- ***uint32_t UART_AdvFeatureInitTypeDef::Swap***
  Specifies whether TX and RX pins are swapped. This parameter can be a value of ***UART_Rx_Tx_Swap***.
- ***uint32_t UART_AdvFeatureInitTypeDef::OverrunDisable***
  Specifies whether the reception overrun detection is disabled. This parameter can be a value of ***UART_Overrun_Disable***.
- ***uint32_t UART_AdvFeatureInitTypeDef::DMADisableonRxError***
  Specifies whether the DMA is disabled in case of reception error. This parameter can be a value of ***UART_DMA_Disable_on_Rx_Error***.
- ***uint32_t UART_AdvFeatureInitTypeDef::AutoBaudRateEnable***
  Specifies whether auto Baud rate detection is enabled. This parameter can be a value of ***UART_AutoBaudRate_Enable***.
- ***uint32_t UART_AdvFeatureInitTypeDef::AutoBaudRateMode***
  If auto Baud rate detection is enabled, specifies how the rate detection is carried out. This parameter can be a value of ***UART_AutoBaud_Rate_Mode***.
- ***uint32_t UART_AdvFeatureInitTypeDef::MSBFirst***
  Specifies whether MSB is sent first on UART line. This parameter can be a value of ***UART_MSB_First***.

79.1.3 __UART_HandleTypeDef

__UART_HandleTypeDef is defined in the stm32n6xx_hal_uart.h

Data Fields

- ***USART_TypeDef * Instance***
- ***UART_InitTypeDef Init***
- ***UART_AdvFeatureInitTypeDef AdvancedInit***
- ***const uint8_t * pTxBuffPtr***
- ***uint16_t TxXferSize***
- ***__IO uint16_t TxXferCount***

- *uint8_t * pRxBuffPtr*
- *uint16_t RxXferSize*
- *__IO uint16_t RxXferCount*
- *uint16_t Mask*
- *uint32_t FifoMode*
- *uint16_t NbRxDataToProcess*
- *uint16_t NbTxDataToProcess*
- *__IO HAL_UART_RxTypeTypeDef ReceptionType*
- *__IO HAL_UART_RxEventTypeTypeDef RxEventType*
- *void(* RxISR*
- *void(* TxISR*
- *DMA_HandleTypeDef * hdmatx*
- *DMA_HandleTypeDef * hdmarx*
- *HAL_LockTypeDef Lock*
- *__IO HAL_UART_StateTypeDef gState*
- *__IO HAL_UART_StateTypeDef RxState*
- *__IO uint32_t ErrorCode*
- *void(* TxHalfCpltCallback*
- *void(* TxCpltCallback*
- *void(* RxHalfCpltCallback*
- *void(* RxCpltCallback*
- *void(* ErrorCallback*
- *void(* AbortCpltCallback*
- *void(* AbortTransmitCpltCallback*
- *void(* AbortReceiveCpltCallback*
- *void(* WakeupCallback*
- *void(* RxFifoFullCallback*
- *void(* TxFifoEmptyCallback*
- *void(* RxEventCallback*
- *void(* MspInitCallback*
- *void(* MspDeInitCallback*

Field Documentation

- ***USART_TypeDef* __UART_HandleTypeDef::Instance***
  UART registers base address
- ***UART_InitTypeDef __UART_HandleTypeDef::Init***
  UART communication parameters
- ***UART_AdvFeatureInitTypeDef __UART_HandleTypeDef::AdvancedInit***
  UART Advanced Features initialization parameters
- ***const uint8_t* __UART_HandleTypeDef::pTxBuffPtr***
  Pointer to UART Tx transfer Buffer
- ***uint16_t __UART_HandleTypeDef::TxXferSize***
  UART Tx Transfer size
- ***__IO uint16_t __UART_HandleTypeDef::TxXferCount***
  UART Tx Transfer Counter
- ***uint8_t* __UART_HandleTypeDef::pRxBuffPtr***
  Pointer to UART Rx transfer Buffer
- ***uint16_t __UART_HandleTypeDef::RxXferSize***
  UART Rx Transfer size
- ***__IO uint16_t __UART_HandleTypeDef::RxXferCount***
  UART Rx Transfer Counter
- ***uint16_t __UART_HandleTypeDef::Mask***
  UART Rx RDR register mask

- ***uint32_t __UART_HandleTypeDef::FifoMode***
  Specifies if the FIFO mode is being used. This parameter can be a value of ***UARTEx_FIFO_mode***.
- ***uint16_t __UART_HandleTypeDef::NbRxDataToProcess***
  Number of data to process during RX ISR execution
- ***uint16_t __UART_HandleTypeDef::NbTxDataToProcess***
  Number of data to process during TX ISR execution
- ***__IO HAL_UART_RxTypeTypeDef __UART_HandleTypeDef::ReceptionType***
  Type of ongoing reception
- ***__IO HAL_UART_RxEventTypeTypeDef __UART_HandleTypeDef::RxEventType***
  Type of Rx Event
- ***void(* __UART_HandleTypeDef::RxISR)(struct __UART_HandleTypeDef *huart)***
  Function pointer on Rx IRQ handler
- ***void(* __UART_HandleTypeDef::TxISR)(struct __UART_HandleTypeDef *huart)***
  Function pointer on Tx IRQ handler
- ***DMA_HandleTypeDef* __UART_HandleTypeDef::hdmatx***
  UART Tx DMA Handle parameters
- ***DMA_HandleTypeDef* __UART_HandleTypeDef::hdmarx***
  UART Rx DMA Handle parameters
- ***HAL_LockTypeDef __UART_HandleTypeDef::Lock***
  Locking object
- ***__IO HAL_UART_StateTypeDef __UART_HandleTypeDef::gState***
  UART state information related to global Handle management and also related to Tx operations. This parameter can be a value of **HAL_UART_StateTypeDef**
- ***__IO HAL_UART_StateTypeDef __UART_HandleTypeDef::RxState***
  UART state information related to Rx operations. This parameter can be a value of **HAL_UART_StateTypeDef**
- ***__IO uint32_t __UART_HandleTypeDef::ErrorCode***
  UART Error code
- ***void(* __UART_HandleTypeDef::TxHalfCpltCallback)(struct __UART_HandleTypeDef *huart)***
  UART Tx Half Complete Callback
- ***void(* __UART_HandleTypeDef::TxCpltCallback)(struct __UART_HandleTypeDef *huart)***
  UART Tx Complete Callback
- ***void(* __UART_HandleTypeDef::RxHalfCpltCallback)(struct __UART_HandleTypeDef *huart)***
  UART Rx Half Complete Callback
- ***void(* __UART_HandleTypeDef::RxCpltCallback)(struct __UART_HandleTypeDef *huart)***
  UART Rx Complete Callback
- ***void(* __UART_HandleTypeDef::ErrorCallback)(struct __UART_HandleTypeDef *huart)***
  UART Error Callback
- ***void(* __UART_HandleTypeDef::AbortCpltCallback)(struct __UART_HandleTypeDef *huart)***
  UART Abort Complete Callback
- ***void(* __UART_HandleTypeDef::AbortTransmitCpltCallback)(struct __UART_HandleTypeDef *huart)***
  UART Abort Transmit Complete Callback
- ***void(* __UART_HandleTypeDef::AbortReceiveCpltCallback)(struct __UART_HandleTypeDef *huart)***
  UART Abort Receive Complete Callback
- ***void(* __UART_HandleTypeDef::WakeupCallback)(struct __UART_HandleTypeDef *huart)***
  UART Wakeup Callback
- ***void(* __UART_HandleTypeDef::RxFifoFullCallback)(struct __UART_HandleTypeDef *huart)***
  UART Rx Fifo Full Callback
- ***void(* __UART_HandleTypeDef::TxFifoEmptyCallback)(struct __UART_HandleTypeDef *huart)***
  UART Tx Fifo Empty Callback
- ***void(* __UART_HandleTypeDef::RxEventCallback)(struct __UART_HandleTypeDef *huart, uint16_t Pos)***
  UART Reception Event Callback
- ***void(* __UART_HandleTypeDef::MspInitCallback)(struct __UART_HandleTypeDef *huart)***
  UART Msp Init callback

- ***void(* __UART_HandleTypeDef::MspDeInitCallback)(struct __UART_HandleTypeDef *huart)***
  UART Msp DeInit callback

79.2 UART Firmware driver API description

The following section lists the various functions of the UART library.

79.2.1 How to use this driver

The UART HAL driver can be used as follows:

1. Declare a UART_HandleTypeDef handle structure (eg. UART_HandleTypeDef huart).
2. Initialize the UART low level resources by implementing the HAL_UART_MspInit() API:
   - Enable the USARTx interface clock.
   - UART pins configuration:
     - Enable the clock for the UART GPIOs.
     - Configure these UART pins as alternate function pull-up.
   - NVIC configuration if you need to use interrupt process (HAL_UART_Transmit_IT() and HAL_UART_Receive_IT() APIs):
     - Configure the USARTx interrupt priority.
     - Enable the NVIC USART IRQ handle.
   - UART interrupts handling:

   Note: The specific UART interrupts (Transmission complete interrupt, RXNE interrupt, RX/TX FIFOs related interrupts and Error Interrupts) are managed using the macros __HAL_UART_ENABLE_IT() and __HAL_UART_DISABLE_IT() inside the transmit and receive processes.

   - DMA Configuration if you need to use DMA process (HAL_UART_Transmit_DMA() and HAL_UART_Receive_DMA() APIs):
     - Declare a DMA handle structure for the Tx/Rx channel.
     - Enable the DMAx interface clock.
     - Configure the declared DMA handle structure with the required Tx/Rx parameters.
     - Configure the DMA Tx/Rx channel.
     - Associate the initialized DMA handle to the UART DMA Tx/Rx handle.
     - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx/Rx channel.
3. Program the Baud Rate, Word Length, Stop Bit, Parity, Prescaler value , Hardware flow control and Mode (Receiver/Transmitter) in the huart handle Init structure.
4. If required, program UART advanced features (TX/RX pins swap, auto Baud rate detection,...) in the huart handle AdvancedInit structure.
5. For the UART asynchronous mode, initialize the UART registers by calling the HAL_UART_Init() API.
6. For the UART Half duplex mode, initialize the UART registers by calling the HAL_HalfDuplex_Init() API.
7. For the UART LIN (Local Interconnection Network) mode, initialize the UART registers by calling the HAL_LIN_Init() API.
8. For the UART Multiprocessor mode, initialize the UART registers by calling the HAL_MultiProcessor_Init() API.
9. For the UART RS485 Driver Enabled mode, initialize the UART registers by calling the HAL_RS485Ex_Init() API.

Note: These API's (HAL_UART_Init(), HAL_HalfDuplex_Init(), HAL_LIN_Init(), HAL_MultiProcessor_Init(), also configure the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customized HAL_UART_MspInit() API.

79.2.2 Callback registration

The compilation define USE_HAL_UART_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks.

Use Function HAL_UART_RegisterCallback() to register a user callback. Function HAL_UART_RegisterCallback() allows to register following callbacks:

- TxHalfCpltCallback : Tx Half Complete Callback.

- TxCpltCallback : Tx Complete Callback.
- RxHalfCpltCallback : Rx Half Complete Callback.
- RxCpltCallback : Rx Complete Callback.
- ErrorCallback : Error Callback.
- AbortCpltCallback : Abort Complete Callback.
- AbortTransmitCpltCallback : Abort Transmit Complete Callback.
- AbortReceiveCpltCallback : Abort Receive Complete Callback.
- WakeupCallback : Wakeup Callback.
- RxFifoFullCallback : Rx Fifo Full Callback.
- TxFifoEmptyCallback : Tx Fifo Empty Callback.
- MspInitCallback : UART MspInit.
- MspDeInitCallback : UART MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

Use function HAL_UART_UnRegisterCallback() to reset a callback to the default weak function. HAL_UART_UnRegisterCallback() takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks:

- TxHalfCpltCallback : Tx Half Complete Callback.
- TxCpltCallback : Tx Complete Callback.
- RxHalfCpltCallback : Rx Half Complete Callback.
- RxCpltCallback : Rx Complete Callback.
- ErrorCallback : Error Callback.
- AbortCpltCallback : Abort Complete Callback.
- AbortTransmitCpltCallback : Abort Transmit Complete Callback.
- AbortReceiveCpltCallback : Abort Receive Complete Callback.
- WakeupCallback : Wakeup Callback.
- RxFifoFullCallback : Rx Fifo Full Callback.
- TxFifoEmptyCallback : Tx Fifo Empty Callback.
- MspInitCallback : UART MspInit.
- MspDeInitCallback : UART MspDeInit.

For specific callback RxEventCallback, use dedicated registration/reset functions: respectively HAL_UART_RegisterRxEventCallback() , HAL_UART_UnRegisterRxEventCallback().

By default, after the HAL_UART_Init() and when the state is HAL_UART_STATE_RESET all callbacks are set to the corresponding weak functions: examples HAL_UART_TxCpltCallback(), HAL_UART_RxHalfCpltCallback(). Exception done for MspInit and MspDeInit functions that are respectively reset to the legacy weak functions in the HAL_UART_Init() and HAL_UART_DeInit() only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the HAL_UART_Init() and HAL_UART_DeInit() keep and use the user MspInit/MspDeInit callbacks (registered beforehand).

Callbacks can be registered/unregistered in HAL_UART_STATE_READY state only. Exception done MspInit/MspDeInit that can be registered/unregistered in HAL_UART_STATE_READY or HAL_UART_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_UART_RegisterCallback() before calling HAL_UART_DeInit() or HAL_UART_Init() function.

When The compilation define USE_HAL_UART_REGISTER_CALLBACKS is set to 0 or not defined, the callback registration feature is not available and weak callbacks are used.

79.2.3 Detailed description of functions

HAL_UART_Init

Function name

HAL_StatusTypeDef HAL_UART_Init (UART_HandleTypeDef * huart)

Function description

HAL_HalfDuplex_Init

Function name

HAL_StatusTypeDef HAL_HalfDuplex_Init (UART_HandleTypeDef * huart)

Function description

HAL_LIN_Init

Function name

HAL_StatusTypeDef HAL_LIN_Init (UART_HandleTypeDef * huart, uint32_t BreakDetectLength)

Function description

HAL_MultiProcessor_Init

Function name

HAL_StatusTypeDef HAL_MultiProcessor_Init (UART_HandleTypeDef * huart, uint8_t Address, uint32_t WakeUpMethod)

Function description

HAL_UART_DeInit

Function name

HAL_StatusTypeDef HAL_UART_DeInit (UART_HandleTypeDef * huart)

Function description

HAL_UART_MspInit

Function name

void HAL_UART_MspInit (UART_HandleTypeDef * huart)

Function description

HAL_UART_MspDeInit

Function name

void HAL_UART_MspDeInit (UART_HandleTypeDef * huart)

Function description

HAL_UART_RegisterCallback

Function name

HAL_StatusTypeDef HAL_UART_RegisterCallback (UART_HandleTypeDef * huart, HAL_UART_CallbackIDTypeDef CallbackID, pUART_CallbackTypeDef pCallback)

Function description

HAL_UART_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_UART_UnRegisterCallback (UART_HandleTypeDef * huart, HAL_UART_CallbackIDTypeDef CallbackID)

Function description

HAL_UART_RegisterRxEventCallback

Function name

HAL_StatusTypeDef HAL_UART_RegisterRxEventCallback (UART_HandleTypeDef * huart, pUART_RxEventCallbackTypeDef pCallback)

Function description

HAL_UART_UnRegisterRxEventCallback

Function name

HAL_StatusTypeDef HAL_UART_UnRegisterRxEventCallback (UART_HandleTypeDef * huart)

Function description

HAL_UART_Transmit

Function name

HAL_StatusTypeDef HAL_UART_Transmit (UART_HandleTypeDef * huart, const uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

HAL_UART_Receive

Function name

HAL_StatusTypeDef HAL_UART_Receive (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

HAL_UART_Transmit_IT

Function name

HAL_StatusTypeDef HAL_UART_Transmit_IT (UART_HandleTypeDef * huart, const uint8_t * pData, uint16_t Size)

Function description

HAL_UART_Receive_IT

Function name

HAL_StatusTypeDef HAL_UART_Receive_IT (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)

Function description

HAL_UART_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_UART_Transmit_DMA (UART_HandleTypeDef * huart, const uint8_t * pData, uint16_t Size)

Function description

HAL_UART_Receive_DMA

Function name

HAL_StatusTypeDef HAL_UART_Receive_DMA (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)

Function description

HAL_UART_DMAPause

Function name

HAL_StatusTypeDef HAL_UART_DMAPause (UART_HandleTypeDef * huart)

Function description

HAL_UART_DMAResume

Function name

HAL_StatusTypeDef HAL_UART_DMAResume (UART_HandleTypeDef * huart)

Function description

HAL_UART_DMAStop

Function name

HAL_StatusTypeDef HAL_UART_DMAStop (UART_HandleTypeDef * huart)

Function description

HAL_UART_Abort

Function name

HAL_StatusTypeDef HAL_UART_Abort (UART_HandleTypeDef * huart)

Function description

HAL_UART_AbortTransmit

Function name

HAL_StatusTypeDef HAL_UART_AbortTransmit (UART_HandleTypeDef * huart)

Function description

HAL_UART_AbortReceive

Function name

HAL_StatusTypeDef HAL_UART_AbortReceive (UART_HandleTypeDef * huart)

Function description

HAL_UART_Abort_IT

Function name

HAL_StatusTypeDef HAL_UART_Abort_IT (UART_HandleTypeDef * huart)

Function description

HAL_UART_AbortTransmit_IT

Function name

HAL_StatusTypeDef HAL_UART_AbortTransmit_IT (UART_HandleTypeDef * huart)

Function description

HAL_UART_AbortReceive_IT

Function name

HAL_StatusTypeDef HAL_UART_AbortReceive_IT (UART_HandleTypeDef * huart)

Function description

HAL_UART_IRQHandler

Function name

void HAL_UART_IRQHandler (UART_HandleTypeDef * huart)

Function description

HAL_UART_TxHalfCpltCallback

Function name

void HAL_UART_TxHalfCpltCallback (UART_HandleTypeDef * huart)

Function description

HAL_UART_TxCpltCallback

Function name

void HAL_UART_TxCpltCallback (UART_HandleTypeDef * huart)

Function description

HAL_UART_RxHalfCpltCallback

Function name

void HAL_UART_RxHalfCpltCallback (UART_HandleTypeDef * huart)

Function description

HAL_UART_RxCpltCallback

Function name

void HAL_UART_RxCpltCallback (UART_HandleTypeDef * huart)

Function description

HAL_UART_ErrorCallback

Function name

void HAL_UART_ErrorCallback (UART_HandleTypeDef * huart)

Function description

HAL_UART_AbortCpltCallback

Function name

void HAL_UART_AbortCpltCallback (UART_HandleTypeDef * huart)

Function description

HAL_UART_AbortTransmitCpltCallback

Function name

void HAL_UART_AbortTransmitCpltCallback (UART_HandleTypeDef * huart)

Function description

HAL_UART_AbortReceiveCpltCallback

Function name

void HAL_UART_AbortReceiveCpltCallback (UART_HandleTypeDef * huart)

Function description

HAL_UARTEx_RxEventCallback

Function name

void HAL_UARTEx_RxEventCallback (UART_HandleTypeDef * huart, uint16_t Size)

Function description

HAL_UART_ReceiverTimeout_Config

Function name

void HAL_UART_ReceiverTimeout_Config (UART_HandleTypeDef * huart, uint32_t TimeoutValue)

Function description

HAL_UART_EnableReceiverTimeout

Function name

HAL_StatusTypeDef HAL_UART_EnableReceiverTimeout (UART_HandleTypeDef * huart)

Function description

HAL_UART_DisableReceiverTimeout

Function name

HAL_StatusTypeDef HAL_UART_DisableReceiverTimeout (UART_HandleTypeDef * huart)

Function description

HAL_LIN_SendBreak

Function name

HAL_StatusTypeDef HAL_LIN_SendBreak (UART_HandleTypeDef * huart)

Function description

HAL_MultiProcessor_EnableMuteMode

Function name

HAL_StatusTypeDef HAL_MultiProcessor_EnableMuteMode (UART_HandleTypeDef * huart)

Function description

HAL_MultiProcessor_DisableMuteMode

Function name

HAL_StatusTypeDef HAL_MultiProcessor_DisableMuteMode (UART_HandleTypeDef * huart)

Function description

HAL_MultiProcessor_EnterMuteMode

Function name

void HAL_MultiProcessor_EnterMuteMode (UART_HandleTypeDef * huart)

Function description

HAL_HalfDuplex_EnableTransmitter

Function name

HAL_StatusTypeDef HAL_HalfDuplex_EnableTransmitter (UART_HandleTypeDef * huart)

Function description

HAL_HalfDuplex_EnableReceiver

Function name

HAL_StatusTypeDef HAL_HalfDuplex_EnableReceiver (UART_HandleTypeDef * huart)

Function description

HAL_UART_GetState

Function name

HAL_UART_StateTypeDef HAL_UART_GetState (const UART_HandleTypeDef * huart)

Function description

HAL_UART_GetError

Function name

uint32_t HAL_UART_GetError (const UART_HandleTypeDef * huart)

Function description

UART_InitCallbacksToDefault

Function name

void UART_InitCallbacksToDefault (UART_HandleTypeDef * huart)

Function description

UART_SetConfig

Function name

HAL_StatusTypeDef UART_SetConfig (UART_HandleTypeDef * huart)

Function description

UART_CheckIdleState

Function name

HAL_StatusTypeDef UART_CheckIdleState (UART_HandleTypeDef * huart)

Function description

UART_WaitOnFlagUntilTimeout

Function name

HAL_StatusTypeDef UART_WaitOnFlagUntilTimeout (UART_HandleTypeDef * huart, uint32_t Flag, FlagStatus Status, uint32_t Tickstart, uint32_t Timeout)

Function description

UART_AdvFeatureConfig

Function name

void UART_AdvFeatureConfig (UART_HandleTypeDef * huart)

Function description

UART_Start_Receive_IT

Function name

HAL_StatusTypeDef UART_Start_Receive_IT (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)

Function description

UART_Start_Receive_DMA

Function name

HAL_StatusTypeDef UART_Start_Receive_DMA (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)

Function description

79.3 UART Firmware driver defines

The following section lists the various define and macros of the module.

79.3.1 UART

UART

UART Advanced Feature Initialization Type

UART_ADVFEATURE_NO_INIT

No advanced feature initialization

UART_ADVFEATURE_TXINVERT_INIT

TX pin active level inversion

UART_ADVFEATURE_RXINVERT_INIT

RX pin active level inversion

UART_ADVFEATURE_DATAINVERT_INIT

Binary data inversion

UART_ADVFEATURE_SWAP_INIT

TX/RX pins swap

UART_ADVFEATURE_RXOVERRUNDISABLE_INIT

RX overrun disable

UART_ADVFEATURE_DMADISABLEONERROR_INIT

DMA disable on Reception Error

UART_ADVFEATURE_AUTOBAUDRATE_INIT

Auto Baud rate detection initialization

UART_ADVFEATURE_MSBFIRST_INIT

Most significant bit sent/received first

UART Advanced Feature Auto BaudRate Enable

UART_ADVFEATURE_AUTOBAUDRATE_DISABLE

RX Auto Baud rate detection enable

UART_ADVFEATURE_AUTOBAUDRATE_ENABLE

RX Auto Baud rate detection disable

UART Advanced Feature AutoBaud Rate Mode

UART_ADVFEATURE_AUTOBAUDRATE_ONSTARTBIT

Auto Baud rate detection on start bit

UART_ADVFEATURE_AUTOBAUDRATE_ONFALLINGEDGE

Auto Baud rate detection on falling edge

UART_ADVFEATURE_AUTOBAUDRATE_ON0X7FFRAME

Auto Baud rate detection on 0x7F frame detection

UART_ADVFEATURE_AUTOBAUDRATE_ON0X55FRAME

Auto Baud rate detection on 0x55 frame detection

UART Clock Prescaler

UART_PRESCALER_DIV1

fclk_pres = fclk

UART_PRESCALER_DIV2

fclk_pres = fclk/2

UART_PRESCALER_DIV4

fclk_pres = fclk/4

UART_PRESCALER_DIV6

fclk_pres = fclk/6

UART_PRESCALER_DIV8

fclk_pres = fclk/8

UART_PRESCALER_DIV10

fclk_pres = fclk/10

UART_PRESCALER_DIV12

fclk_pres = fclk/12

UART_PRESCALER_DIV16

fclk_pres = fclk/16

UART_PRESCALER_DIV32

fclk_pres = fclk/32

UART_PRESCALER_DIV64

fclk_pres = fclk/64

UART_PRESCALER_DIV128

fclk_pres = fclk/128

UART_PRESCALER_DIV256

fclk_pres = fclk/256

UART Driver Enable Assertion Time LSB Position In CR1 Register

UART_CR1_DEAT_ADDRESS_LSB_POS

UART Driver Enable assertion time LSB position in CR1 register

UART Driver Enable DeAssertion Time LSB Position In CR1 Register

UART_CR1_DEDT_ADDRESS_LSB_POS

UART Driver Enable de-assertion time LSB position in CR1 register

UART Address-matching LSB Position In CR2 Register

UART_CR2_ADDRESS_LSB_POS

UART address-matching LSB position in CR2 register

UART Advanced Feature Binary Data Inversion

UART_ADVFEATURE_DATAINV_DISABLE

Binary data inversion disable

UART_ADVFEATURE_DATAINV_ENABLE

Binary data inversion enable

UART Advanced Feature DMA Disable On Rx Error

UART_ADVFEATURE_DMA_ENABLEONRXERROR

DMA enable on Reception Error

UART_ADVFEATURE_DMA_DISABLEONRXERROR

DMA disable on Reception Error

UART DMA Rx

UART_DMA_RX_DISABLE

UART DMA RX disabled

UART_DMA_RX_ENABLE

UART DMA RX enabled

UART DMA Tx

UART_DMA_TX_DISABLE

UART DMA TX disabled

UART_DMA_TX_ENABLE

UART DMA TX enabled

UART DriverEnable Polarity

UART_DE_POLARITY_HIGH

Driver enable signal is active high

UART_DE_POLARITY_LOW

Driver enable signal is active low

UART Error Definition

HAL_UART_ERROR_NONE

No error

HAL_UART_ERROR_PE

Parity error

HAL_UART_ERROR_NE

Noise error

HAL_UART_ERROR_FE

Frame error

HAL_UART_ERROR_ORE

Overrun error

HAL_UART_ERROR_DMA

DMA transfer error

HAL_UART_ERROR_RTO

Receiver Timeout error

HAL_UART_ERROR_INVALID_CALLBACK

Invalid Callback error

UART Exported Macros

__HAL_UART_RESET_HANDLE_STATE

Description:

- Reset UART handle states.

Parameters:

- __HANDLE__: UART handle.

Return value:

- None

__HAL_UART_FLUSH_DRREGISTER

Description:

- Flush the UART Data registers.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

__HAL_UART_CLEAR_FLAG

Description:

- Clear the specified UART pending flag.

Parameters:

- __HANDLE__: specifies the UART Handle.
- __FLAG__: specifies the flag to check. This parameter can be any combination of the following values:
  - UART_CLEAR_PEF Parity Error Clear Flag
  - UART_CLEAR_FEF Framing Error Clear Flag
  - UART_CLEAR_NEF Noise detected Clear Flag
  - UART_CLEAR_OREF Overrun Error Clear Flag
  - UART_CLEAR_IDLEF IDLE line detected Clear Flag
  - UART_CLEAR_TXFECF TXFIFO empty clear Flag
  - UART_CLEAR_TCF Transmission Complete Clear Flag
  - UART_CLEAR_RTOF Receiver Timeout clear flag
  - UART_CLEAR_LBDF LIN Break Detection Clear Flag
  - UART_CLEAR_CTSF CTS Interrupt Clear Flag
  - UART_CLEAR_CMF Character Match Clear Flag
  - UART_CLEAR_WUF Wake Up from stop mode Clear Flag

Return value:

- None

__HAL_UART_CLEAR_PEFLAG

Description:

- Clear the UART PE pending flag.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

__HAL_UART_CLEAR_FEFLAG

Description:

- Clear the UART FE pending flag.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

__HAL_UART_CLEAR_NEFLAG

Description:

- Clear the UART NE pending flag.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

__HAL_UART_CLEAR_OREFLAG

Description:

- Clear the UART ORE pending flag.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

__HAL_UART_CLEAR_IDLEFLAG

Description:

- Clear the UART IDLE pending flag.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

__HAL_UART_CLEAR_TXFECF

Description:

- Clear the UART TX FIFO empty clear flag.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

__HAL_UART_GET_FLAG

Description:

- Check whether the specified UART flag is set or not.

Parameters:

- __HANDLE__: specifies the UART Handle.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - UART_FLAG_TXFT TXFIFO threshold flag
  - UART_FLAG_RXFT RXFIFO threshold flag
  - UART_FLAG_RXFF RXFIFO Full flag
  - UART_FLAG_TXFE TXFIFO Empty flag
  - UART_FLAG_REACK Receive enable acknowledge flag
  - UART_FLAG_TEACK Transmit enable acknowledge flag
  - UART_FLAG_WUF Wake up from stop mode flag
  - UART_FLAG_RWU Receiver wake up flag (if the UART in mute mode)
  - UART_FLAG_SBKF Send Break flag
  - UART_FLAG_CMF Character match flag
  - UART_FLAG_BUSY Busy flag
  - UART_FLAG_ABRF Auto Baud rate detection flag
  - UART_FLAG_ABRE Auto Baud rate detection error flag
  - UART_FLAG_CTS CTS Change flag
  - UART_FLAG_LBDF LIN Break detection flag
  - UART_FLAG_TXE Transmit data register empty flag
  - UART_FLAG_TXFNF UART TXFIFO not full flag
  - UART_FLAG_TC Transmission Complete flag
  - UART_FLAG_RXNE Receive data register not empty flag
  - UART_FLAG_RXFNE UART RXFIFO not empty flag
  - UART_FLAG_RTOF Receiver Timeout flag
  - UART_FLAG_IDLE Idle Line detection flag
  - UART_FLAG_ORE Overrun Error flag
  - UART_FLAG_NE Noise Error flag
  - UART_FLAG_FE Framing Error flag
  - UART_FLAG_PE Parity Error flag

Return value:

- The: new state of __FLAG__ (TRUE or FALSE).

__HAL_UART_ENABLE_IT

Description:

- Enable the specified UART interrupt.

Parameters:

- __HANDLE__: specifies the UART Handle.
- __INTERRUPT__: specifies the UART interrupt source to enable. This parameter can be one of the following values:
  - UART_IT_RXFF RXFIFO Full interrupt
  - UART_IT_TXFE TXFIFO Empty interrupt
  - UART_IT_RXFT RXFIFO threshold interrupt
  - UART_IT_TXFT TXFIFO threshold interrupt
  - UART_IT_WUF Wakeup from stop mode interrupt
  - UART_IT_CM Character match interrupt
  - UART_IT_CTS CTS change interrupt
  - UART_IT_LBD LIN Break detection interrupt
  - UART_IT_TXE Transmit Data Register empty interrupt
  - UART_IT_TXFNF TX FIFO not full interrupt
  - UART_IT_TC Transmission complete interrupt
  - UART_IT_RXNE Receive Data register not empty interrupt
  - UART_IT_RXFNE RXFIFO not empty interrupt
  - UART_IT_RTO Receive Timeout interrupt
  - UART_IT_IDLE Idle line detection interrupt
  - UART_IT_PE Parity Error interrupt
  - UART_IT_ERR Error interrupt (frame error, noise error, overrun error)

Return value:

- None

__HAL_UART_DISABLE_IT

Description:

- Disable the specified UART interrupt.

Parameters:

- __HANDLE__: specifies the UART Handle.
- __INTERRUPT__: specifies the UART interrupt source to disable. This parameter can be one of the following values:
  - UART_IT_RXFF RXFIFO Full interrupt
  - UART_IT_TXFE TXFIFO Empty interrupt
  - UART_IT_RXFT RXFIFO threshold interrupt
  - UART_IT_TXFT TXFIFO threshold interrupt
  - UART_IT_WUF Wakeup from stop mode interrupt
  - UART_IT_CM Character match interrupt
  - UART_IT_CTS CTS change interrupt
  - UART_IT_LBD LIN Break detection interrupt
  - UART_IT_TXE Transmit Data Register empty interrupt
  - UART_IT_TXFNF TX FIFO not full interrupt
  - UART_IT_TC Transmission complete interrupt
  - UART_IT_RXNE Receive Data register not empty interrupt
  - UART_IT_RXFNE RXFIFO not empty interrupt
  - UART_IT_RTO Receive Timeout interrupt
  - UART_IT_IDLE Idle line detection interrupt
  - UART_IT_PE Parity Error interrupt
  - UART_IT_ERR Error interrupt (Frame error, noise error, overrun error)

Return value:

- None

__HAL_UART_GET_IT

Description:

- Check whether the specified UART interrupt has occurred or not.

Parameters:

- __HANDLE__: specifies the UART Handle.
- __INTERRUPT__: specifies the UART interrupt to check. This parameter can be one of the following values:
  - UART_IT_RXFF RXFIFO Full interrupt
  - UART_IT_TXFE TXFIFO Empty interrupt
  - UART_IT_RXFT RXFIFO threshold interrupt
  - UART_IT_TXFT TXFIFO threshold interrupt
  - UART_IT_WUF Wakeup from stop mode interrupt
  - UART_IT_CM Character match interrupt
  - UART_IT_CTS CTS change interrupt
  - UART_IT_LBD LIN Break detection interrupt
  - UART_IT_TXE Transmit Data Register empty interrupt
  - UART_IT_TXFNF TX FIFO not full interrupt
  - UART_IT_TC Transmission complete interrupt
  - UART_IT_RXNE Receive Data register not empty interrupt
  - UART_IT_RXFNE RXFIFO not empty interrupt
  - UART_IT_RTO Receive Timeout interrupt
  - UART_IT_IDLE Idle line detection interrupt
  - UART_IT_PE Parity Error interrupt
  - UART_IT_ERR Error interrupt (Frame error, noise error, overrun error)

Return value:

- The: new state of __INTERRUPT__ (SET or RESET).

__HAL_UART_GET_IT_SOURCE

Description:

- Check whether the specified UART interrupt source is enabled or not.

Parameters:

- __HANDLE__: specifies the UART Handle.
- __INTERRUPT__: specifies the UART interrupt source to check. This parameter can be one of the following values:
  - UART_IT_RXFF RXFIFO Full interrupt
  - UART_IT_TXFE TXFIFO Empty interrupt
  - UART_IT_RXFT RXFIFO threshold interrupt
  - UART_IT_TXFT TXFIFO threshold interrupt
  - UART_IT_WUF Wakeup from stop mode interrupt
  - UART_IT_CM Character match interrupt
  - UART_IT_CTS CTS change interrupt
  - UART_IT_LBD LIN Break detection interrupt
  - UART_IT_TXE Transmit Data Register empty interrupt
  - UART_IT_TXFNF TX FIFO not full interrupt
  - UART_IT_TC Transmission complete interrupt
  - UART_IT_RXNE Receive Data register not empty interrupt
  - UART_IT_RXFNE RXFIFO not empty interrupt
  - UART_IT_RTO Receive Timeout interrupt
  - UART_IT_IDLE Idle line detection interrupt
  - UART_IT_PE Parity Error interrupt
  - UART_IT_ERR Error interrupt (Frame error, noise error, overrun error)

Return value:

- The: new state of __INTERRUPT__ (SET or RESET).

__HAL_UART_CLEAR_IT

Description:

- Clear the specified UART ISR flag, in setting the proper ICR register flag.

Parameters:

- __HANDLE__: specifies the UART Handle.
- __IT_CLEAR__: specifies the interrupt clear register flag that needs to be set to clear the corresponding interrupt This parameter can be one of the following values:
  - UART_CLEAR_PEF Parity Error Clear Flag
  - UART_CLEAR_FEF Framing Error Clear Flag
  - UART_CLEAR_NEF Noise detected Clear Flag
  - UART_CLEAR_OREF Overrun Error Clear Flag
  - UART_CLEAR_IDLEF IDLE line detected Clear Flag
  - UART_CLEAR_RTOF Receiver timeout clear flag
  - UART_CLEAR_TXFECF TXFIFO empty Clear Flag
  - UART_CLEAR_TCF Transmission Complete Clear Flag
  - UART_CLEAR_LBDF LIN Break Detection Clear Flag
  - UART_CLEAR_CTSF CTS Interrupt Clear Flag
  - UART_CLEAR_CMF Character Match Clear Flag
  - UART_CLEAR_WUF Wake Up from stop mode Clear Flag

Return value:

- None

__HAL_UART_SEND_REQ

Description:

- Set a specific UART request flag.

Parameters:

- __HANDLE__: specifies the UART Handle.
- __REQ__: specifies the request flag to set This parameter can be one of the following values:
  - UART_AUTOBAUD_REQUEST Auto-Baud Rate Request
  - UART_SENDBREAK_REQUEST Send Break Request
  - UART_MUTE_MODE_REQUEST Mute Mode Request
  - UART_RXDATA_FLUSH_REQUEST Receive Data flush Request
  - UART_TXDATA_FLUSH_REQUEST Transmit data flush Request

Return value:

- None

__HAL_UART_ONE_BIT_SAMPLE_ENABLE

Description:

- Enable the UART one bit sample method.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

__HAL_UART_ONE_BIT_SAMPLE_DISABLE

Description:

- Disable the UART one bit sample method.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

__HAL_UART_ENABLE

Description:

- Enable UART.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

__HAL_UART_DISABLE

Description:

- Disable UART.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

__HAL_UART_HWCONTROL_CTS_ENABLE

Description:

- Enable CTS flow control.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

Notes:

- This macro allows to enable CTS hardware flow control for a given UART instance, without need to call HAL_UART_Init() function. As involving direct access to UART registers, usage of this macro should be fully endorsed by user. As macro is expected to be used for modifying CTS Hw flow control feature activation, without need for USART instance Deinit/Init, following conditions for macro call should be fulfilled : UART instance should have already been initialised (through call of HAL_UART_Init())macro could only be called when corresponding UART instance is disabled (i.e. __HAL_UART_DISABLE(__HANDLE__)) and should be followed by an Enable macro (i.e. __HAL_UART_ENABLE(__HANDLE__)).

__HAL_UART_HWCONTROL_CTS_DISABLE

Description:

- Disable CTS flow control.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

Notes:

- This macro allows to disable CTS hardware flow control for a given UART instance, without need to call HAL_UART_Init() function. As involving direct access to UART registers, usage of this macro should be fully endorsed by user. As macro is expected to be used for modifying CTS Hw flow control feature activation, without need for USART instance Deinit/Init, following conditions for macro call should be fulfilled : UART instance should have already been initialised (through call of HAL_UART_Init())macro could only be called when corresponding UART instance is disabled (i.e. __HAL_UART_DISABLE(__HANDLE__)) and should be followed by an Enable macro (i.e. __HAL_UART_ENABLE(__HANDLE__)).

__HAL_UART_HWCONTROL_RTS_ENABLE

Description:

- Enable RTS flow control.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

Notes:

- This macro allows to enable RTS hardware flow control for a given UART instance, without need to call HAL_UART_Init() function. As involving direct access to UART registers, usage of this macro should be fully endorsed by user. As macro is expected to be used for modifying RTS Hw flow control feature activation, without need for USART instance Deinit/Init, following conditions for macro call should be fulfilled : UART instance should have already been initialised (through call of HAL_UART_Init())macro could only be called when corresponding UART instance is disabled (i.e. __HAL_UART_DISABLE(__HANDLE__)) and should be followed by an Enable macro (i.e. __HAL_UART_ENABLE(__HANDLE__)).

__HAL_UART_HWCONTROL_RTS_DISABLE

Description:

- Disable RTS flow control.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

Notes:

- This macro allows to disable RTS hardware flow control for a given UART instance, without need to call HAL_UART_Init() function. As involving direct access to UART registers, usage of this macro should be fully endorsed by user. As macro is expected to be used for modifying RTS Hw flow control feature activation, without need for USART instance Deinit/Init, following conditions for macro call should be fulfilled : UART instance should have already been initialised (through call of HAL_UART_Init())macro could only be called when corresponding UART instance is disabled (i.e. __HAL_UART_DISABLE(__HANDLE__)) and should be followed by an Enable macro (i.e. __HAL_UART_ENABLE(__HANDLE__)).

UART Status Flags

UART_FLAG_TXFT

UART TXFIFO threshold flag

UART_FLAG_RXFT

UART RXFIFO threshold flag

UART_FLAG_RXFF

UART RXFIFO Full flag

UART_FLAG_TXFE

UART TXFIFO Empty flag

UART_FLAG_REACK

UART receive enable acknowledge flag

UART_FLAG_TEACK

UART transmit enable acknowledge flag

UART_FLAG_WUF

UART wake-up from stop mode flag

UART_FLAG_RWU

UART receiver wake-up from mute mode flag

UART_FLAG_SBKF

UART send break flag

UART_FLAG_CMF

UART character match flag

UART_FLAG_BUSY

UART busy flag

UART_FLAG_ABRF

UART auto Baud rate flag

UART_FLAG_ABRE

UART auto Baud rate error

UART_FLAG_RTOF

UART receiver timeout flag

UART_FLAG_CTS

UART clear to send flag

UART_FLAG_CTSIF

UART clear to send interrupt flag

UART_FLAG_LBDF

UART LIN break detection flag

UART_FLAG_TXE

UART transmit data register empty

UART_FLAG_TXFNF

UART TXFIFO not full

UART_FLAG_TC

UART transmission complete

UART_FLAG_RXNE

UART read data register not empty

UART_FLAG_RXFNE

UART RXFIFO not empty

UART_FLAG_IDLE

UART idle flag

UART_FLAG_ORE

UART overrun error

UART_FLAG_NE

UART noise error

UART_FLAG_FE

UART frame error

UART_FLAG_PE

UART parity error

UART Half Duplex Selection

UART_HALF_DUPLEX_DISABLE

UART half-duplex disabled

UART_HALF_DUPLEX_ENABLE

UART half-duplex enabled

UART Hardware Flow Control

UART_HWCONTROL_NONE

No hardware control

UART_HWCONTROL_RTS

Request To Send

UART_HWCONTROL_CTS

Clear To Send

UART_HWCONTROL_RTS_CTS

Request and Clear To Send

UART Interruptions Flag Mask

UART_IT_MASK

UART interruptions flags mask

UART Interrupts Definition

UART_IT_PE

UART parity error interruption

UART_IT_TXE

UART transmit data register empty interruption

UART_IT_TXFNF

UART TX FIFO not full interruption

UART_IT_TC

UART transmission complete interruption

UART_IT_RXNE

UART read data register not empty interruption

UART_IT_RXFNE

UART RXFIFO not empty interruption

UART_IT_IDLE

UART idle interruption

UART_IT_LBD

UART LIN break detection interruption

UART_IT_CTS

UART CTS interruption

UART_IT_CM

UART character match interruption

UART_IT_WUF

UART wake-up from stop mode interruption

UART_IT_RXFF

UART RXFIFO full interruption

UART_IT_TXFE

UART TXFIFO empty interruption

UART_IT_RXFT

UART RXFIFO threshold reached interruption

UART_IT_TXFT

UART TXFIFO threshold reached interruption

UART_IT_RTO

UART receiver timeout interruption

UART_IT_ERR

UART error interruption

UART_IT_ORE

UART overrun error interruption

UART_IT_NE

UART noise error interruption

UART_IT_FE

UART frame error interruption

UART Interruption Clear Flags

UART_CLEAR_PEF

Parity Error Clear Flag

UART_CLEAR_FEF

Framing Error Clear Flag

UART_CLEAR_NEF

Noise Error detected Clear Flag

UART_CLEAR_OREF

Overrun Error Clear Flag

UART_CLEAR_IDLEF

IDLE line detected Clear Flag

UART_CLEAR_TXFECF

TXFIFO empty clear flag

UART_CLEAR_TCF

Transmission Complete Clear Flag

UART_CLEAR_LBDF

LIN Break Detection Clear Flag

UART_CLEAR_CTSF

CTS Interrupt Clear Flag

UART_CLEAR_CMF

Character Match Clear Flag

UART_CLEAR_WUF

Wake Up from stop mode Clear Flag

UART_CLEAR_RTOF

UART receiver timeout clear flag

UART Local Interconnection Network mode

UART_LIN_DISABLE

Local Interconnect Network disable

UART_LIN_ENABLE

Local Interconnect Network enable

UART LIN Break Detection

UART_LINBREAKDETECTLENGTH_10B

LIN 10-bit break detection length

UART_LINBREAKDETECTLENGTH_11B

LIN 11-bit break detection length

UART Transfer Mode

UART_MODE_RX

RX mode

UART_MODE_TX

TX mode

UART_MODE_TX_RX

RX and TX mode

UART Advanced Feature MSB First

UART_ADVFEATURE_MSBFIRST_DISABLE

Most significant bit sent/received first disable

UART_ADVFEATURE_MSBFIRST_ENABLE

Most significant bit sent/received first enable

UART Advanced Feature Mute Mode Enable

UART_ADVFEATURE_MUTEMODE_DISABLE

UART mute mode disable

UART_ADVFEATURE_MUTEMODE_ENABLE

UART mute mode enable

UART One Bit Sampling Method

UART_ONE_BIT_SAMPLE_DISABLE

One-bit sampling disable

UART_ONE_BIT_SAMPLE_ENABLE

One-bit sampling enable

UART Advanced Feature Overrun Disable

UART_ADVFEATURE_OVERRUN_ENABLE

RX overrun enable

UART_ADVFEATURE_OVERRUN_DISABLE

RX overrun disable

UART Over Sampling

UART_OVERSAMPLING_16

Oversampling by 16

UART_OVERSAMPLING_8

Oversampling by 8

UART Parity

UART_PARITY_NONE

No parity

UART_PARITY_EVEN

Even parity

UART_PARITY_ODD

Odd parity

UART Receiver Timeout

UART_RECEIVER_TIMEOUT_DISABLE

UART Receiver Timeout disable

UART_RECEIVER_TIMEOUT_ENABLE

UART Receiver Timeout enable

UART Reception type values

HAL_UART_RECEPTION_STANDARD

Standard reception

HAL_UART_RECEPTION_TOIDLE

Reception till completion or IDLE event

HAL_UART_RECEPTION_TORTO

Reception till completion or RTO event

HAL_UART_RECEPTION_TOCHARMATCH

Reception till completion or CM event

UART Request Parameters

UART_AUTOBAUD_REQUEST

Auto-Baud Rate Request

UART_SENDBREAK_REQUEST

Send Break Request

UART_MUTE_MODE_REQUEST

Mute Mode Request

UART_RXDATA_FLUSH_REQUEST

Receive Data flush Request

UART_TXDATA_FLUSH_REQUEST

Transmit data flush Request

UART RxEvent type values

HAL_UART_RXEVENT_TC

RxEvent linked to Transfer Complete event

HAL_UART_RXEVENT_HT

RxEvent linked to Half Transfer event

HAL_UART_RXEVENT_IDLE

RxEvent linked to IDLE event

UART Advanced Feature RX Pin Active Level Inversion

UART_ADVFEATURE_RXINV_DISABLE

RX pin active level inversion disable

UART_ADVFEATURE_RXINV_ENABLE

RX pin active level inversion enable

UART Advanced Feature RX TX Pins Swap

UART_ADVFEATURE_SWAP_DISABLE

TX/RX pins swap disable

UART_ADVFEATURE_SWAP_ENABLE

TX/RX pins swap enable

UART State

UART_STATE_DISABLE

UART disabled

UART_STATE_ENABLE

UART enabled

UART State Code Definition

HAL_UART_STATE_RESET

Peripheral is not initialized Value is allowed for gState and RxState

HAL_UART_STATE_READY

Peripheral Initialized and ready for use Value is allowed for gState and RxState

HAL_UART_STATE_BUSY

an internal process is ongoing Value is allowed for gState only

HAL_UART_STATE_BUSY_TX

Data Transmission process is ongoing Value is allowed for gState only

HAL_UART_STATE_BUSY_RX

Data Reception process is ongoing Value is allowed for RxState only

HAL_UART_STATE_BUSY_TX_RX

Data Transmission and Reception process is ongoing Not to be used for neither gState nor RxState.Value is result of combination (Or) between gState and RxState values

HAL_UART_STATE_TIMEOUT

Timeout state Value is allowed for gState only

HAL_UART_STATE_ERROR

Error Value is allowed for gState only

UART Number of Stop Bits

UART_STOPBITS_0_5

UART frame with 0.5 stop bit

UART_STOPBITS_1

UART frame with 1 stop bit

UART_STOPBITS_1_5

UART frame with 1.5 stop bits

UART_STOPBITS_2

UART frame with 2 stop bits

UART Advanced Feature Stop Mode Enable

UART_ADVFEATURE_STOPMODE_DISABLE

UART stop mode disable

UART_ADVFEATURE_STOPMODE_ENABLE

UART stop mode enable

UART polling-based communications time-out value

HAL_UART_TIMEOUT_VALUE

UART polling-based communications time-out value

UART Advanced Feature TX Pin Active Level Inversion

UART_ADVFEATURE_TXINV_DISABLE

TX pin active level inversion disable

UART_ADVFEATURE_TXINV_ENABLE

TX pin active level inversion enable

UART WakeUp From Stop Selection

UART_WAKEUP_ON_ADDRESS

UART wake-up on address

UART_WAKEUP_ON_STARTBIT

UART wake-up on start bit

UART_WAKEUP_ON_READDATA_NONEMPTY

UART wake-up on receive data register not empty or RXFIFO is not empty

UART WakeUp Methods

UART_WAKEUPMETHOD_IDLELINE

UART wake-up on idle line

UART_WAKEUPMETHOD_ADDRESSMARK

UART wake-up on address mark

80 HAL UART Extension Driver

80.1 UARTEx Firmware driver registers structures

80.1.1 UART_WakeUpTypeDef

UART_WakeUpTypeDef is defined in the stm32n6xx_hal_uart_ex.h

Data Fields

- ***uint32_t WakeUpEvent***
- ***uint16_t AddressLength***
- ***uint8_t Address***

Field Documentation

- ***uint32_t UART_WakeUpTypeDef::WakeUpEvent***
  Specifies which event will activate the Wakeup from Stop mode flag (WUF). This parameter can be a value of ***UART_WakeUp_from_Stop_Selection***. If set to UART_WAKEUP_ON_ADDRESS, the two other fields below must be filled up.
- ***uint16_t UART_WakeUpTypeDef::AddressLength***
  Specifies whether the address is 4 or 7-bit long. This parameter can be a value of ***UARTEx_WakeUp_Address_Length***.
- ***uint8_t UART_WakeUpTypeDef::Address***
  UART/USART node address (7-bit long max).

80.2 UARTEx Firmware driver API description

The following section lists the various functions of the UARTEx library.

80.2.1 UART peripheral extended features

1. Declare a UART_HandleTypeDef handle structure.
2. For the UART RS485 Driver Enable mode, initialize the UART registers by calling the HAL_RS485Ex_Init() API.
3. FIFO mode enabling/disabling and RX/TX FIFO threshold programming.

Note: When UART operates in FIFO mode, FIFO mode must be enabled prior starting RX/TX transfers. Also RX/TX FIFO thresholds must be configured prior starting RX/TX transfers.

80.2.2 Detailed description of functions

HAL_RS485Ex_Init

Function name

HAL_StatusTypeDef HAL_RS485Ex_Init (UART_HandleTypeDef * huart, uint32_t Polarity, uint32_t AssertionTime, uint32_t DeassertionTime)

Function description

HAL_UARTEx_WakeupCallback

Function name

void HAL_UARTEx_WakeupCallback (UART_HandleTypeDef * huart)

Function description

HAL_UARTEx_RxFifoFullCallback

Function name

void HAL_UARTEx_RxFifoFullCallback (UART_HandleTypeDef * huart)

Function description

HAL_UARTEx_TxFifoEmptyCallback

Function name

void HAL_UARTEx_TxFifoEmptyCallback (UART_HandleTypeDef * huart)

Function description

HAL_UARTEx_StopModeWakeUpSourceConfig

Function name

HAL_StatusTypeDef HAL_UARTEx_StopModeWakeUpSourceConfig (UART_HandleTypeDef * huart, UART_WakeUpTypeDef WakeUpSelection)

Function description

HAL_UARTEx_EnableStopMode

Function name

HAL_StatusTypeDef HAL_UARTEx_EnableStopMode (UART_HandleTypeDef * huart)

Function description

HAL_UARTEx_DisableStopMode

Function name

HAL_StatusTypeDef HAL_UARTEx_DisableStopMode (UART_HandleTypeDef * huart)

Function description

HAL_MultiProcessorEx_AddressLength_Set

Function name

HAL_StatusTypeDef HAL_MultiProcessorEx_AddressLength_Set (UART_HandleTypeDef * huart, uint32_t AddressLength)

Function description

HAL_UARTEx_EnableFifoMode

Function name

HAL_StatusTypeDef HAL_UARTEx_EnableFifoMode (UART_HandleTypeDef * huart)

Function description

HAL_UARTEx_DisableFifoMode

Function name

HAL_StatusTypeDef HAL_UARTEx_DisableFifoMode (UART_HandleTypeDef * huart)

Function description

HAL_UARTEx_SetTxFifoThreshold

Function name

HAL_StatusTypeDef HAL_UARTEx_SetTxFifoThreshold (UART_HandleTypeDef * huart, uint32_t Threshold)

Function description

HAL_UARTEx_SetRxFifoThreshold

Function name

HAL_StatusTypeDef HAL_UARTEx_SetRxFifoThreshold (UART_HandleTypeDef * huart, uint32_t Threshold)

Function description

HAL_UARTEx_ReceiveToIdle

Function name

HAL_StatusTypeDef HAL_UARTEx_ReceiveToIdle (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size, uint16_t * RxLen, uint32_t Timeout)

Function description

HAL_UARTEx_ReceiveToIdle_IT

Function name

HAL_StatusTypeDef HAL_UARTEx_ReceiveToIdle_IT (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)

Function description

HAL_UARTEx_ReceiveToIdle_DMA

Function name

HAL_StatusTypeDef HAL_UARTEx_ReceiveToIdle_DMA (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)

Function description

HAL_UARTEx_GetRxEventType

Function name

HAL_UART_RxEventTypeTypeDef HAL_UARTEx_GetRxEventType (const UART_HandleTypeDef * huart)

Function description

80.3 UARTEx Firmware driver defines

The following section lists the various define and macros of the module.

80.3.1 UARTEx

UARTEx

UARTEx FIFO mode

UART_FIFOMODE_DISABLE

FIFO mode disable

UART_FIFOMODE_ENABLE

FIFO mode enable

UARTEx RXFIFO threshold level

UART_RXFIFO_THRESHOLD_1_8

RX FIFO reaches 1/8 of its depth

UART_RXFIFO_THRESHOLD_1_4

RX FIFO reaches 1/4 of its depth

UART_RXFIFO_THRESHOLD_1_2

RX FIFO reaches 1/2 of its depth

UART_RXFIFO_THRESHOLD_3_4

RX FIFO reaches 3/4 of its depth

UART_RXFIFO_THRESHOLD_7_8

RX FIFO reaches 7/8 of its depth

UART_RXFIFO_THRESHOLD_8_8

RX FIFO becomes full

UARTEx TXFIFO threshold level

UART_TXFIFO_THRESHOLD_1_8

TX FIFO reaches 1/8 of its depth

UART_TXFIFO_THRESHOLD_1_4

TX FIFO reaches 1/4 of its depth

UART_TXFIFO_THRESHOLD_1_2

TX FIFO reaches 1/2 of its depth

UART_TXFIFO_THRESHOLD_3_4

TX FIFO reaches 3/4 of its depth

UART_TXFIFO_THRESHOLD_7_8

TX FIFO reaches 7/8 of its depth

UART_TXFIFO_THRESHOLD_8_8

TX FIFO becomes empty

UARTEx WakeUp Address Length

UART_ADDRESS_DETECT_4B

4-bit long wake-up address

UART_ADDRESS_DETECT_7B

7-bit long wake-up address

UARTEx Word Length

UART_WORDLENGTH_7B

7-bit long UART frame

UART_WORDLENGTH_8B

8-bit long UART frame

UART_WORDLENGTH_9B

9-bit long UART frame

81 HAL USART Generic Driver

81.1 USART Firmware driver registers structures

81.1.1 USART_InitTypeDef

USART_InitTypeDef is defined in the stm32n6xx_hal_usart.h

Data Fields

- ***uint32_t BaudRate***
- ***uint32_t WordLength***
- ***uint32_t StopBits***
- ***uint32_t Parity***
- ***uint32_t Mode***
- ***uint32_t CLKPolarity***
- ***uint32_t CLKPhase***
- ***uint32_t CLKLastBit***
- ***uint32_t ClockPrescaler***

Field Documentation

- ***uint32_t USART_InitTypeDef::BaudRate***
  This member configures the Usart communication baud rate. The baud rate is computed using the following formula: Baud Rate Register[15:4] = ((2 * fclk_pres) / ((huart->Init.BaudRate)))[15:4] Baud Rate Register[3] = 0 Baud Rate Register[2:0] = (((2 * fclk_pres) / ((huart->Init.BaudRate)))[3:0]) >> 1 where fclk_pres is the USART input clock frequency (fclk) divided by a prescaler.
  Note:
  – Oversampling by 8 is systematically applied to achieve high baud rates.
- ***uint32_t USART_InitTypeDef::WordLength***
  Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of ***USARTEx_Word_Length***.
- ***uint32_t USART_InitTypeDef::StopBits***
  Specifies the number of stop bits transmitted. This parameter can be a value of ***USART_Stop_Bits***.
- ***uint32_t USART_InitTypeDef::Parity***
  Specifies the parity mode. This parameter can be a value of ***USART_Parity***
  Note:
  – When parity is enabled, the computed parity is inserted at the MSB position of the transmitted data (9th bit when the word length is set to 9 data bits; 8th bit when the word length is set to 8 data bits).
- ***uint32_t USART_InitTypeDef::Mode***
  Specifies whether the Receive or Transmit mode is enabled or disabled. This parameter can be a value of ***USART_Mode***.
- ***uint32_t USART_InitTypeDef::CLKPolarity***
  Specifies the steady state of the serial clock. This parameter can be a value of ***USART_Clock_Polarity***.
- ***uint32_t USART_InitTypeDef::CLKPhase***
  Specifies the clock transition on which the bit capture is made. This parameter can be a value of ***USART_Clock_Phase***.
- ***uint32_t USART_InitTypeDef::CLKLastBit***
  Specifies whether the clock pulse corresponding to the last transmitted data bit (MSB) has to be output on the SCLK pin in synchronous mode. This parameter can be a value of ***USART_Last_Bit***.
- ***uint32_t USART_InitTypeDef::ClockPrescaler***
  Specifies the prescaler value used to divide the USART clock source. This parameter can be a value of ***USART_ClockPrescaler***.

81.1.2 __USART_HandleTypeDef

__USART_HandleTypeDef is defined in the stm32n6xx_hal_usart.h

Data Fields

- ***USART_TypeDef * Instance***
- ***USART_InitTypeDef Init***
- ***const uint8_t * pTxBuffPtr***
- ***uint16_t TxXferSize***
- ***__IO uint16_t TxXferCount***
- ***uint8_t * pRxBuffPtr***
- ***uint16_t RxXferSize***
- ***__IO uint16_t RxXferCount***
- ***uint16_t Mask***
- ***uint16_t NbRxDataToProcess***
- ***uint16_t NbTxDataToProcess***
- ***uint32_t SlaveMode***
- ***uint32_t FifoMode***
- ***void(* RxISR***
- ***void(* TxISR***
- ***DMA_HandleTypeDef * hdmatx***
- ***DMA_HandleTypeDef * hdmarx***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_USART_StateTypeDef State***
- ***__IO uint32_t ErrorCode***
- ***void(* TxHalfCpltCallback***
- ***void(* TxCpltCallback***
- ***void(* RxHalfCpltCallback***
- ***void(* RxCpltCallback***
- ***void(* TxRxCpltCallback***
- ***void(* ErrorCallback***
- ***void(* AbortCpltCallback***
- ***void(* RxFifoFullCallback***
- ***void(* TxFifoEmptyCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***USART_TypeDef* __USART_HandleTypeDef::Instance***
  USART registers base address
- ***USART_InitTypeDef __USART_HandleTypeDef::Init***
  USART communication parameters
- ***const uint8_t* __USART_HandleTypeDef::pTxBuffPtr***
  Pointer to USART Tx transfer Buffer
- ***uint16_t __USART_HandleTypeDef::TxXferSize***
  USART Tx Transfer size
- ***__IO uint16_t __USART_HandleTypeDef::TxXferCount***
  USART Tx Transfer Counter
- ***uint8_t* __USART_HandleTypeDef::pRxBuffPtr***
  Pointer to USART Rx transfer Buffer
- ***uint16_t __USART_HandleTypeDef::RxXferSize***
  USART Rx Transfer size
- ***__IO uint16_t __USART_HandleTypeDef::RxXferCount***
  USART Rx Transfer Counter
- ***uint16_t __USART_HandleTypeDef::Mask***
  USART Rx RDR register mask
- ***uint16_t __USART_HandleTypeDef::NbRxDataToProcess***
  Number of data to process during RX ISR execution

- ***uint16_t __USART_HandleTypeDef::NbTxDataToProcess***
  Number of data to process during TX ISR execution
- ***uint32_t __USART_HandleTypeDef::SlaveMode***
  Enable/Disable USART SPI Slave Mode. This parameter can be a value of ***USARTEx_Slave_Mode***
- ***uint32_t __USART_HandleTypeDef::FifoMode***
  Specifies if the FIFO mode will be used. This parameter can be a value of ***USARTEx_FIFO_mode***.
- ***void(* __USART_HandleTypeDef::RxISR)(struct __USART_HandleTypeDef *husart)***
  Function pointer on Rx IRQ handler
- ***void(* __USART_HandleTypeDef::TxISR)(struct __USART_HandleTypeDef *husart)***
  Function pointer on Tx IRQ handler
- ***DMA_HandleTypeDef* __USART_HandleTypeDef::hdmatx***
  USART Tx DMA Handle parameters
- ***DMA_HandleTypeDef* __USART_HandleTypeDef::hdmarx***
  USART Rx DMA Handle parameters
- ***HAL_LockTypeDef __USART_HandleTypeDef::Lock***
  Locking object
- ***__IO HAL_USART_StateTypeDef __USART_HandleTypeDef::State***
  USART communication state
- ***__IO uint32_t __USART_HandleTypeDef::ErrorCode***
  USART Error code
- ***void(* __USART_HandleTypeDef::TxHalfCpltCallback)(struct __USART_HandleTypeDef *husart)***
  USART Tx Half Complete Callback
- ***void(* __USART_HandleTypeDef::TxCpltCallback)(struct __USART_HandleTypeDef *husart)***
  USART Tx Complete Callback
- ***void(* __USART_HandleTypeDef::RxHalfCpltCallback)(struct __USART_HandleTypeDef *husart)***
  USART Rx Half Complete Callback
- ***void(* __USART_HandleTypeDef::RxCpltCallback)(struct __USART_HandleTypeDef *husart)***
  USART Rx Complete Callback
- ***void(* __USART_HandleTypeDef::TxRxCpltCallback)(struct __USART_HandleTypeDef *husart)***
  USART Tx Rx Complete Callback
- ***void(* __USART_HandleTypeDef::ErrorCallback)(struct __USART_HandleTypeDef *husart)***
  USART Error Callback
- ***void(* __USART_HandleTypeDef::AbortCpltCallback)(struct __USART_HandleTypeDef *husart)***
  USART Abort Complete Callback
- ***void(* __USART_HandleTypeDef::RxFifoFullCallback)(struct __USART_HandleTypeDef *husart)***
  USART Rx Fifo Full Callback
- ***void(* __USART_HandleTypeDef::TxFifoEmptyCallback)(struct __USART_HandleTypeDef *husart)***
  USART Tx Fifo Empty Callback
- ***void(* __USART_HandleTypeDef::MspInitCallback)(struct __USART_HandleTypeDef *husart)***
  USART Msp Init callback
- ***void(* __USART_HandleTypeDef::MspDeInitCallback)(struct __USART_HandleTypeDef *husart)***
  USART Msp DeInit callback

81.2 USART Firmware driver API description

The following section lists the various functions of the USART library.

81.2.1 How to use this driver

The USART HAL driver can be used as follows:

1. Declare a USART_HandleTypeDef handle structure (eg. USART_HandleTypeDef husart).

2. Initialize the USART low level resources by implementing the HAL_USART_MspInit() API:
   - Enable the USARTx interface clock.
   - USART pins configuration:
     - Enable the clock for the USART GPIOs.
     - Configure these USART pins as alternate function pull-up.
   - NVIC configuration if you need to use interrupt process (HAL_USART_Transmit_IT(), HAL_USART_Receive_IT() and HAL_USART_TransmitReceive_IT() APIs):
     - Configure the USARTx interrupt priority.
     - Enable the NVIC USART IRQ handle.
   - USART interrupts handling:

Note: The specific USART interrupts (Transmission complete interrupt, RXNE interrupt and Error Interrupts) will be managed using the macros __HAL_USART_ENABLE_IT() and __HAL_USART_DISABLE_IT() inside the transmit and receive process.

   - DMA Configuration if you need to use DMA process (HAL_USART_Transmit_DMA() HAL_USART_Receive_DMA() and HAL_USART_TransmitReceive_DMA() APIs):
     - Declare a DMA handle structure for the Tx/Rx channel.
     - Enable the DMAx interface clock.
     - Configure the declared DMA handle structure with the required Tx/Rx parameters.
     - Configure the DMA Tx/Rx channel.
     - Associate the initialized DMA handle to the USART DMA Tx/Rx handle.
     - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx/Rx channel.
3. Program the Baud Rate, Word Length, Stop Bit, Parity, and Mode (Receiver/Transmitter) in the husart handle Init structure.
4. Initialize the USART registers by calling the HAL_USART_Init() API:
   - This API configures also the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customized HAL_USART_MspInit(&husart) API.

Note: To configure and enable/disable the USART to wake up the MCU from stop mode, resort to UART API's HAL_UARTEx_StopModeWakeUpSourceConfig(), HAL_UARTEx_EnableStopMode() and HAL_UARTEx_DisableStopMode() in casting the USART handle to UART type UART_HandleTypeDef.

81.2.2 Callback registration

The compilation define USE_HAL_USART_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks.

Use Function HAL_USART_RegisterCallback() to register a user callback. Function HAL_USART_RegisterCallback() allows to register following callbacks:

- TxHalfCpltCallback : Tx Half Complete Callback.
- TxCpltCallback : Tx Complete Callback.
- RxHalfCpltCallback : Rx Half Complete Callback.
- RxCpltCallback : Rx Complete Callback.
- TxRxCpltCallback : Tx Rx Complete Callback.
- ErrorCallback : Error Callback.
- AbortCpltCallback : Abort Complete Callback.
- RxFifoFullCallback : Rx Fifo Full Callback.
- TxFifoEmptyCallback : Tx Fifo Empty Callback.
- MspInitCallback : USART MspInit.
- MspDeInitCallback : USART MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

Use function HAL_USART_UnRegisterCallback() to reset a callback to the default weak function. HAL_USART_UnRegisterCallback() takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks:

- TxHalfCpltCallback : Tx Half Complete Callback.

- TxCpltCallback : Tx Complete Callback.
- RxHalfCpltCallback : Rx Half Complete Callback.
- RxCpltCallback : Rx Complete Callback.
- TxRxCpltCallback : Tx Rx Complete Callback.
- ErrorCallback : Error Callback.
- AbortCpltCallback : Abort Complete Callback.
- RxFifoFullCallback : Rx Fifo Full Callback.
- TxFifoEmptyCallback : Tx Fifo Empty Callback.
- MspInitCallback : USART MspInit.
- MspDeInitCallback : USART MspDeInit.

By default, after the HAL_USART_Init() and when the state is HAL_USART_STATE_RESET all callbacks are set to the corresponding weak functions: examples HAL_USART_TxCpltCallback(), HAL_USART_RxHalfCpltCallback(). Exception done for MspInit and MspDeInit functions that are respectively reset to the legacy weak functions in the HAL_USART_Init() and HAL_USART_DeInit() only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the HAL_USART_Init() and HAL_USART_DeInit() keep and use the user MspInit/MspDeInit callbacks (registered beforehand).

Callbacks can be registered/unregistered in HAL_USART_STATE_READY state only. Exception done MspInit/MspDeInit that can be registered/unregistered in HAL_USART_STATE_READY or HAL_USART_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_USART_RegisterCallback() before calling HAL_USART_DeInit() or HAL_USART_Init() function.

When The compilation define USE_HAL_USART_REGISTER_CALLBACKS is set to 0 or not defined, the callback registration feature is not available and weak callbacks are used.

81.2.3 Detailed description of functions

HAL_USART_Init

Function name

HAL_StatusTypeDef HAL_USART_Init (USART_HandleTypeDef * husart)

Function description

HAL_USART_DeInit

Function name

HAL_StatusTypeDef HAL_USART_DeInit (USART_HandleTypeDef * husart)

Function description

HAL_USART_MspInit

Function name

void HAL_USART_MspInit (USART_HandleTypeDef * husart)

Function description

HAL_USART_MspDeInit

Function name

void HAL_USART_MspDeInit (USART_HandleTypeDef * husart)

Function description

HAL_USART_RegisterCallback

Function name

HAL_StatusTypeDef HAL_USART_RegisterCallback (USART_HandleTypeDef * husart, HAL_USART_CallbackIDTypeDef CallbackID, pUSART_CallbackTypeDef pCallback)

Function description

HAL_USART_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_USART_UnRegisterCallback (USART_HandleTypeDef * husart, HAL_USART_CallbackIDTypeDef CallbackID)

Function description

HAL_USART_Transmit

Function name

HAL_StatusTypeDef HAL_USART_Transmit (USART_HandleTypeDef * husart, const uint8_t * pTxData, uint16_t Size, uint32_t Timeout)

Function description

HAL_USART_Receive

Function name

HAL_StatusTypeDef HAL_USART_Receive (USART_HandleTypeDef * husart, uint8_t * pRxData, uint16_t Size, uint32_t Timeout)

Function description

HAL_USART_TransmitReceive

Function name

HAL_StatusTypeDef HAL_USART_TransmitReceive (USART_HandleTypeDef * husart, const uint8_t * pTxData, uint8_t * pRxData, uint16_t Size, uint32_t Timeout)

Function description

HAL_USART_Transmit_IT

Function name

HAL_StatusTypeDef HAL_USART_Transmit_IT (USART_HandleTypeDef * husart, const uint8_t * pTxData, uint16_t Size)

Function description

HAL_USART_Receive_IT

Function name

HAL_StatusTypeDef HAL_USART_Receive_IT (USART_HandleTypeDef * husart, uint8_t * pRxData, uint16_t Size)

Function description

HAL_USART_TransmitReceive_IT

Function name

HAL_StatusTypeDef HAL_USART_TransmitReceive_IT (USART_HandleTypeDef * husart, const uint8_t * pTxData, uint8_t * pRxData, uint16_t Size)

Function description

HAL_USART_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_USART_Transmit_DMA (USART_HandleTypeDef * husart, const uint8_t * pTxData, uint16_t Size)

Function description

HAL_USART_Receive_DMA

Function name

HAL_StatusTypeDef HAL_USART_Receive_DMA (USART_HandleTypeDef * husart, uint8_t * pRxData, uint16_t Size)

Function description

HAL_USART_TransmitReceive_DMA

Function name

HAL_StatusTypeDef HAL_USART_TransmitReceive_DMA (USART_HandleTypeDef * husart, const uint8_t * pTxData, uint8_t * pRxData, uint16_t Size)

Function description

HAL_USART_DMAPause

Function name

HAL_StatusTypeDef HAL_USART_DMAPause (USART_HandleTypeDef * husart)

Function description

HAL_USART_DMAResume

Function name

HAL_StatusTypeDef HAL_USART_DMAResume (USART_HandleTypeDef * husart)

Function description

HAL_USART_DMAStop

Function name

HAL_StatusTypeDef HAL_USART_DMAStop (USART_HandleTypeDef * husart)

Function description

HAL_USART_Abort

Function name

HAL_StatusTypeDef HAL_USART_Abort (USART_HandleTypeDef * husart)

Function description

HAL_USART_Abort_IT

Function name

HAL_StatusTypeDef HAL_USART_Abort_IT (USART_HandleTypeDef * husart)

Function description

HAL_USART_IRQHandler

Function name

void HAL_USART_IRQHandler (USART_HandleTypeDef * husart)

Function description

HAL_USART_TxHalfCpltCallback

Function name

void HAL_USART_TxHalfCpltCallback (USART_HandleTypeDef * husart)

Function description

HAL_USART_TxCpltCallback

Function name

void HAL_USART_TxCpltCallback (USART_HandleTypeDef * husart)

Function description

HAL_USART_RxCpltCallback

Function name

void HAL_USART_RxCpltCallback (USART_HandleTypeDef * husart)

Function description

HAL_USART_RxHalfCpltCallback

Function name

void HAL_USART_RxHalfCpltCallback (USART_HandleTypeDef * husart)

Function description

HAL_USART_TxRxCpltCallback

Function name

void HAL_USART_TxRxCpltCallback (USART_HandleTypeDef * husart)

Function description

HAL_USART_ErrorCallback

Function name

void HAL_USART_ErrorCallback (USART_HandleTypeDef * husart)

Function description

HAL_USART_AbortCpltCallback

Function name

void HAL_USART_AbortCpltCallback (USART_HandleTypeDef * husart)

Function description

HAL_USART_GetState

Function name

HAL_USART_StateTypeDef HAL_USART_GetState (const USART_HandleTypeDef * husart)

Function description

HAL_USART_GetError

Function name

uint32_t HAL_USART_GetError (const USART_HandleTypeDef * husart)

Function description

81.3 USART Firmware driver defines

The following section lists the various define and macros of the module.

81.3.1 USART

USART

USART Clock

USART_CLOCK_DISABLE

USART clock disable

USART_CLOCK_ENABLE

USART clock enable

USART Clock Prescaler

USART_PRESCALER_DIV1

fclk_pres = fclk

USART_PRESCALER_DIV2

fclk_pres = fclk/2

USART_PRESCALER_DIV4

fclk_pres = fclk/4

USART_PRESCALER_DIV6

fclk_pres = fclk/6

USART_PRESCALER_DIV8

fclk_pres = fclk/8

USART_PRESCALER_DIV10

fclk_pres = fclk/10

USART_PRESCALER_DIV12

fclk_pres = fclk/12

USART_PRESCALER_DIV16

fclk_pres = fclk/16

USART_PRESCALER_DIV32

fclk_pres = fclk/32

USART_PRESCALER_DIV64

fclk_pres = fclk/64

USART_PRESCALER_DIV128

fclk_pres = fclk/128

USART_PRESCALER_DIV256

fclk_pres = fclk/256

USART Clock Phase

USART_PHASE_1EDGE

USART frame phase on first clock transition

USART_PHASE_2EDGE

USART frame phase on second clock transition

USART Clock Polarity

USART_POLARITY_LOW

Driver enable signal is active high

USART_POLARITY_HIGH

Driver enable signal is active low

USART Error Definition

HAL_USART_ERROR_NONE

No error

HAL_USART_ERROR_PE

Parity error

HAL_USART_ERROR_NE

Noise error

HAL_USART_ERROR_FE

Frame error

HAL_USART_ERROR_ORE

Overrun error

HAL_USART_ERROR_DMA

DMA transfer error

HAL_USART_ERROR_UDR

SPI slave underrun error

HAL_USART_ERROR_INVALID_CALLBACK

Invalid Callback error

HAL_USART_ERROR_RTO

Receiver Timeout error

USART Exported Macros

__HAL_USART_RESET_HANDLE_STATE

Description:

- Reset USART handle state.

Parameters:

- __HANDLE__: USART handle.

Return value:

- None

__HAL_USART_GET_FLAG

Description:

- Check whether the specified USART flag is set or not.

Parameters:

- __HANDLE__: specifies the USART Handle
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - USART_FLAG_TXFT TXFIFO threshold flag
  - USART_FLAG_RXFT RXFIFO threshold flag
  - USART_FLAG_RXFF RXFIFO Full flag
  - USART_FLAG_TXFE TXFIFO Empty flag
  - USART_FLAG_REACK Receive enable acknowledge flag
  - USART_FLAG_TEACK Transmit enable acknowledge flag
  - USART_FLAG_BUSY Busy flag
  - USART_FLAG_UDR SPI slave underrun error flag
  - USART_FLAG_TXE Transmit data register empty flag
  - USART_FLAG_TXFNF TXFIFO not full flag
  - USART_FLAG_TC Transmission Complete flag
  - USART_FLAG_RXNE Receive data register not empty flag
  - USART_FLAG_RXFNE RXFIFO not empty flag
  - USART_FLAG_RTOF Receiver Timeout flag
  - USART_FLAG_IDLE Idle Line detection flag
  - USART_FLAG_ORE OverRun Error flag
  - USART_FLAG_NE Noise Error flag
  - USART_FLAG_FE Framing Error flag
  - USART_FLAG_PE Parity Error flag

Return value:

- The: new state of __FLAG__ (TRUE or FALSE).

__HAL_USART_CLEAR_FLAG

Description:

- Clear the specified USART pending flag.

Parameters:

- __HANDLE__: specifies the USART Handle.
- __FLAG__: specifies the flag to check. This parameter can be any combination of the following values:
  - USART_CLEAR_PEF Parity Error Clear Flag
  - USART_CLEAR_FEF Framing Error Clear Flag
  - USART_CLEAR_NEF Noise detected Clear Flag
  - USART_CLEAR_OREF Overrun Error Clear Flag
  - USART_CLEAR_IDLEF IDLE line detected Clear Flag
  - USART_CLEAR_TXFECF TXFIFO empty clear Flag
  - USART_CLEAR_TCF Transmission Complete Clear Flag
  - USART_CLEAR_RTOF Receiver Timeout clear flag
  - USART_CLEAR_UDRF SPI slave underrun error Clear Flag

Return value:

- None

__HAL_USART_CLEAR_PEFLAG

Description:

- Clear the USART PE pending flag.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

__HAL_USART_CLEAR_FEFLAG

Description:

- Clear the USART FE pending flag.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

__HAL_USART_CLEAR_NEFLAG

Description:

- Clear the USART NE pending flag.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

__HAL_USART_CLEAR_OREFLAG

Description:

- Clear the USART ORE pending flag.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

__HAL_USART_CLEAR_IDLEFLAG

Description:

- Clear the USART IDLE pending flag.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

__HAL_USART_CLEAR_TXFECF

Description:

- Clear the USART TX FIFO empty clear flag.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

__HAL_USART_CLEAR_UDRFLAG

Description:

- Clear SPI slave underrun error flag.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

__HAL_USART_ENABLE_IT

Description:

- Enable the specified USART interrupt.

Parameters:

- __HANDLE__: specifies the USART Handle.
- __INTERRUPT__: specifies the USART interrupt source to enable. This parameter can be one of the following values:
  - USART_IT_RXFF RXFIFO Full interrupt
  - USART_IT_TXFE TXFIFO Empty interrupt
  - USART_IT_RXFT RXFIFO threshold interrupt
  - USART_IT_TXFT TXFIFO threshold interrupt
  - USART_IT_TXE Transmit Data Register empty interrupt
  - USART_IT_TXFNF TX FIFO not full interrupt
  - USART_IT_TC Transmission complete interrupt
  - USART_IT_RXNE Receive Data register not empty interrupt
  - USART_IT_RXFNE RXFIFO not empty interrupt
  - USART_IT_IDLE Idle line detection interrupt
  - USART_IT_PE Parity Error interrupt
  - USART_IT_ERR Error interrupt(Frame error, noise error, overrun error)

Return value:

- None

__HAL_USART_DISABLE_IT

Description:

- Disable the specified USART interrupt.

Parameters:

- __HANDLE__: specifies the USART Handle.
- __INTERRUPT__: specifies the USART interrupt source to disable. This parameter can be one of the following values:
  - USART_IT_RXFF RXFIFO Full interrupt
  - USART_IT_TXFE TXFIFO Empty interrupt
  - USART_IT_RXFT RXFIFO threshold interrupt
  - USART_IT_TXFT TXFIFO threshold interrupt
  - USART_IT_TXE Transmit Data Register empty interrupt
  - USART_IT_TXFNF TX FIFO not full interrupt
  - USART_IT_TC Transmission complete interrupt
  - USART_IT_RXNE Receive Data register not empty interrupt
  - USART_IT_RXFNE RXFIFO not empty interrupt
  - USART_IT_IDLE Idle line detection interrupt
  - USART_IT_PE Parity Error interrupt
  - USART_IT_ERR Error interrupt(Frame error, noise error, overrun error)

Return value:

- None

__HAL_USART_GET_IT

Description:

- Check whether the specified USART interrupt has occurred or not.

Parameters:

- __HANDLE__: specifies the USART Handle.
- __INTERRUPT__: specifies the USART interrupt source to check. This parameter can be one of the following values:
  - USART_IT_RXFF RXFIFO Full interrupt
  - USART_IT_TXFE TXFIFO Empty interrupt
  - USART_IT_RXFT RXFIFO threshold interrupt
  - USART_IT_TXFT TXFIFO threshold interrupt
  - USART_IT_TXE Transmit Data Register empty interrupt
  - USART_IT_TXFNF TX FIFO not full interrupt
  - USART_IT_TC Transmission complete interrupt
  - USART_IT_RXNE Receive Data register not empty interrupt
  - USART_IT_RXFNE RXFIFO not empty interrupt
  - USART_IT_IDLE Idle line detection interrupt
  - USART_IT_ORE OverRun Error interrupt
  - USART_IT_NE Noise Error interrupt
  - USART_IT_FE Framing Error interrupt
  - USART_IT_PE Parity Error interrupt

Return value:

- The: new state of __INTERRUPT__ (SET or RESET).

__HAL_USART_GET_IT_SOURCE

Description:

- Check whether the specified USART interrupt source is enabled or not.

Parameters:

- __HANDLE__: specifies the USART Handle.
- __INTERRUPT__: specifies the USART interrupt source to check. This parameter can be one of the following values:
  - USART_IT_RXFF RXFIFO Full interrupt
  - USART_IT_TXFE TXFIFO Empty interrupt
  - USART_IT_RXFT RXFIFO threshold interrupt
  - USART_IT_TXFT TXFIFO threshold interrupt
  - USART_IT_TXE Transmit Data Register empty interrupt
  - USART_IT_TXFNF TX FIFO not full interrupt
  - USART_IT_TC Transmission complete interrupt
  - USART_IT_RXNE Receive Data register not empty interrupt
  - USART_IT_RXFNE RXFIFO not empty interrupt
  - USART_IT_IDLE Idle line detection interrupt
  - USART_IT_ORE OverRun Error interrupt
  - USART_IT_NE Noise Error interrupt
  - USART_IT_FE Framing Error interrupt
  - USART_IT_PE Parity Error interrupt

Return value:

- The: new state of __INTERRUPT__ (SET or RESET).

__HAL_USART_CLEAR_IT

Description:

- Clear the specified USART ISR flag, in setting the proper ICR register flag.

Parameters:

- __HANDLE__: specifies the USART Handle.
- __IT_CLEAR__: specifies the interrupt clear register flag that needs to be set to clear the corresponding interrupt. This parameter can be one of the following values:
  - USART_CLEAR_PEF Parity Error Clear Flag
  - USART_CLEAR_FEF Framing Error Clear Flag
  - USART_CLEAR_NEF Noise detected Clear Flag
  - USART_CLEAR_OREF Overrun Error Clear Flag
  - USART_CLEAR_IDLEF IDLE line detected Clear Flag
  - USART_CLEAR_RTOF Receiver timeout clear flag
  - USART_CLEAR_TXFECF TXFIFO empty clear Flag
  - USART_CLEAR_TCF Transmission Complete Clear Flag

Return value:

- None

__HAL_USART_SEND_REQ

Description:

- Set a specific USART request flag.

Parameters:

- __HANDLE__: specifies the USART Handle.
- __REQ__: specifies the request flag to set. This parameter can be one of the following values:
  - USART_RXDATA_FLUSH_REQUEST Receive Data flush Request
  - USART_TXDATA_FLUSH_REQUEST Transmit data flush Request

Return value:

- None

__HAL_USART_ONE_BIT_SAMPLE_ENABLE

Description:

- Enable the USART one bit sample method.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

__HAL_USART_ONE_BIT_SAMPLE_DISABLE

Description:

- Disable the USART one bit sample method.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

__HAL_USART_ENABLE

Description:

- Enable USART.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

__HAL_USART_DISABLE

Description:

- Disable USART.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

USART Flags

USART_FLAG_TXFT

USART TXFIFO threshold flag

USART_FLAG_RXFT

USART RXFIFO threshold flag

USART_FLAG_RXFF

USART RXFIFO Full flag

USART_FLAG_TXFE

USART TXFIFO Empty flag

USART_FLAG_REACK

USART receive enable acknowledge flag

USART_FLAG_TEACK

USART transmit enable acknowledge flag

USART_FLAG_BUSY

USART busy flag

USART_FLAG_UDR

SPI slave underrun error flag

USART_FLAG_TXE

USART transmit data register empty

USART_FLAG_TXFNF

USART TXFIFO not full

USART_FLAG_RTOF

USART receiver timeout flag

USART_FLAG_TC

USART transmission complete

USART_FLAG_RXNE

USART read data register not empty

USART_FLAG_RXFNE

USART RXFIFO not empty

USART_FLAG_IDLE

USART idle flag

USART_FLAG_ORE

USART overrun error

USART_FLAG_NE

USART noise error

USART_FLAG_FE

USART frame error

USART_FLAG_PE

USART parity error

USART Interruption Flags Mask

USART_IT_MASK

USART interruptions flags mask

USART_CR_MASK

USART control register mask

USART_CR_POS

USART control register position

USART_ISR_MASK

USART ISR register mask

USART_ISR_POS

USART ISR register position

USART Interrupts Definition

USART_IT_PE

USART parity error interruption

USART_IT_TXE

USART transmit data register empty interruption

USART_IT_TXFNF

USART TX FIFO not full interruption

USART_IT_TC

USART transmission complete interruption

USART_IT_RXNE

USART read data register not empty interruption

USART_IT_RXFNE

USART RXFIFO not empty interruption

USART_IT_IDLE

USART idle interruption

USART_IT_ERR

USART error interruption

USART_IT_ORE

USART overrun error interruption

USART_IT_NE

USART noise error interruption

USART_IT_FE

USART frame error interruption

USART_IT_RXFF

USART RXFIFO full interruption

USART_IT_TXFE

USART TXFIFO empty interruption

USART_IT_RXFT

USART RXFIFO threshold reached interruption

USART_IT_TXFT

USART TXFIFO threshold reached interruption

USART Interruption Clear Flags

USART_CLEAR_PEF

Parity Error Clear Flag

USART_CLEAR_FEF

Framing Error Clear Flag

USART_CLEAR_NEF

Noise Error detected Clear Flag

USART_CLEAR_OREF

OverRun Error Clear Flag

USART_CLEAR_IDLEF

IDLE line detected Clear Flag

USART_CLEAR_TCF

Transmission Complete Clear Flag

USART_CLEAR_UDRF

SPI slave underrun error Clear Flag

USART_CLEAR_TXFECF

TXFIFO Empty Clear Flag

USART_CLEAR_RTOF

USART receiver timeout clear flag

USART Last Bit

USART_LASTBIT_DISABLE

USART frame last data bit clock pulse not output to SCLK pin

USART_LASTBIT_ENABLE

USART frame last data bit clock pulse output to SCLK pin

USART Mode

USART_MODE_RX

RX mode

USART_MODE_TX

TX mode

USART_MODE_TX_RX

RX and TX mode

USART Parity

USART_PARITY_NONE

No parity

USART_PARITY_EVEN

Even parity

USART_PARITY_ODD

Odd parity

USART Request Parameters

USART_RXDATA_FLUSH_REQUEST

Receive Data flush Request

USART_TXDATA_FLUSH_REQUEST

Transmit data flush Request

USART Number of Stop Bits

USART_STOPBITS_0_5

USART frame with 0.5 stop bit

USART_STOPBITS_1

USART frame with 1 stop bit

USART_STOPBITS_1_5

USART frame with 1.5 stop bits

USART_STOPBITS_2

USART frame with 2 stop bits

82 HAL USART Extension Driver

82.1 USARTEx Firmware driver API description

The following section lists the various functions of the USARTEx library.

82.1.1 USART peripheral extended features

82.1.2 Detailed description of functions

HAL_USARTEx_RxFifoFullCallback

Function name

void HAL_USARTEx_RxFifoFullCallback (USART_HandleTypeDef * husart)

Function description

HAL_USARTEx_TxFifoEmptyCallback

Function name

void HAL_USARTEx_TxFifoEmptyCallback (USART_HandleTypeDef * husart)

Function description

HAL_USARTEx_EnableSlaveMode

Function name

HAL_StatusTypeDef HAL_USARTEx_EnableSlaveMode (USART_HandleTypeDef * husart)

Function description

HAL_USARTEx_DisableSlaveMode

Function name

HAL_StatusTypeDef HAL_USARTEx_DisableSlaveMode (USART_HandleTypeDef * husart)

Function description

HAL_USARTEx_ConfigNSS

Function name

HAL_StatusTypeDef HAL_USARTEx_ConfigNSS (USART_HandleTypeDef * husart, uint32_t NSSConfig)

Function description

HAL_USARTEx_EnableFifoMode

Function name

HAL_StatusTypeDef HAL_USARTEx_EnableFifoMode (USART_HandleTypeDef * husart)

Function description

HAL_USARTEx_DisableFifoMode

Function name

HAL_StatusTypeDef HAL_USARTEx_DisableFifoMode (USART_HandleTypeDef * husart)

Function description

HAL_USARTEx_SetTxFifoThreshold

Function name

HAL_StatusTypeDef HAL_USARTEx_SetTxFifoThreshold (USART_HandleTypeDef * husart, uint32_t Threshold)

Function description

HAL_USARTEx_SetRxFifoThreshold

Function name

HAL_StatusTypeDef HAL_USARTEx_SetRxFifoThreshold (USART_HandleTypeDef * husart, uint32_t Threshold)

Function description

82.2 USARTEx Firmware driver defines

The following section lists the various define and macros of the module.

82.2.1 USARTEx

USARTEx

USARTEx FIFO mode

USART_FIFOMODE_DISABLE

FIFO mode disable

USART_FIFOMODE_ENABLE

FIFO mode enable

USARTEx RXFIFO threshold level

USART_RXFIFO_THRESHOLD_1_8

RXFIFO FIFO reaches 1/8 of its depth

USART_RXFIFO_THRESHOLD_1_4

RXFIFO FIFO reaches 1/4 of its depth

USART_RXFIFO_THRESHOLD_1_2

RXFIFO FIFO reaches 1/2 of its depth

USART_RXFIFO_THRESHOLD_3_4

RXFIFO FIFO reaches 3/4 of its depth

USART_RXFIFO_THRESHOLD_7_8

RXFIFO FIFO reaches 7/8 of its depth

USART_RXFIFO_THRESHOLD_8_8

RXFIFO FIFO becomes full

USARTEx Synchronous Slave mode enable

USART_SLAVEMODE_DISABLE

USART SPI Slave Mode Enable

USART_SLAVEMODE_ENABLE

USART SPI Slave Mode Disable

USARTEx Slave Select Management

USART_NSS_HARD

SPI slave selection depends on NSS input pin

USART_NSS_SOFT

SPI slave is always selected and NSS input pin is ignored

USARTEx TXFIFO threshold level

USART_TXFIFO_THRESHOLD_1_8

TXFIFO reaches 1/8 of its depth

USART_TXFIFO_THRESHOLD_1_4

TXFIFO reaches 1/4 of its depth

USART_TXFIFO_THRESHOLD_1_2

TXFIFO reaches 1/2 of its depth

USART_TXFIFO_THRESHOLD_3_4

TXFIFO reaches 3/4 of its depth

USART_TXFIFO_THRESHOLD_7_8

TXFIFO reaches 7/8 of its depth

USART_TXFIFO_THRESHOLD_8_8

TXFIFO becomes empty

USARTEx Word Length

USART_WORDLENGTH_7B

7-bit long USART frame

USART_WORDLENGTH_8B

8-bit long USART frame

USART_WORDLENGTH_9B

9-bit long USART frame

83 HAL WWDG Generic Driver

83.1 WWDG Firmware driver registers structures

83.1.1 WWDG_InitTypeDef

WWDG_InitTypeDef is defined in the stm32n6xx_hal_wwdg.h

Data Fields

- ***uint32_t Prescaler***
- ***uint32_t Window***
- ***uint32_t Counter***
- ***uint32_t EWIMode***

Field Documentation

- ***uint32_t WWDG_InitTypeDef::Prescaler***
  Specifies the prescaler value of the WWDG. This parameter can be a value of ***WWDG_Prescaler***
- ***uint32_t WWDG_InitTypeDef::Window***
  Specifies the WWDG window value to be compared to the downcounter. This parameter must be a number Min_Data = 0x40 and Max_Data = 0x7F
- ***uint32_t WWDG_InitTypeDef::Counter***
  Specifies the WWDG free-running downcounter value. This parameter must be a number between Min_Data = 0x40 and Max_Data = 0x7F
- ***uint32_t WWDG_InitTypeDef::EWIMode***
  Specifies if WWDG Early Wakeup Interrupt is enable or not. This parameter can be a value of ***WWDG_EWI_Mode***

83.1.2 __WWDG_HandleTypeDef

__WWDG_HandleTypeDef is defined in the stm32n6xx_hal_wwdg.h

Data Fields

- ***WWDG_TypeDef * Instance***
- ***WWDG_InitTypeDef Init***
- ***void(* EwiCallback***
- ***void(* MspInitCallback***

Field Documentation

- ***WWDG_TypeDef* __WWDG_HandleTypeDef::Instance***
  Register base address
- ***WWDG_InitTypeDef __WWDG_HandleTypeDef::Init***
  WWDG required parameters
- ***void(* __WWDG_HandleTypeDef::EwiCallback)(struct __WWDG_HandleTypeDef *hwwdg)***
  WWDG Early WakeUp Interrupt callback
- ***void(* __WWDG_HandleTypeDef::MspInitCallback)(struct __WWDG_HandleTypeDef *hwwdg)***
  WWDG Msp Init callback

83.2 WWDG Firmware driver API description

The following section lists the various functions of the WWDG library.

83.2.1 WWDG Specific features

Once enabled the WWDG generates a system reset on expiry of a programmed time period, unless the program refreshes the counter (T[6;0] downcounter) before reaching 0x3F value (i.e. a reset is generated when the counter value rolls down from 0x40 to 0x3F).

- An MCU reset is also generated if the counter value is refreshed before the counter has reached the refresh window value. This implies that the counter must be refreshed in a limited window.

- Once enabled the WWDG cannot be disabled except by a system reset.
- If required by application, an Early Wakeup Interrupt can be triggered in order to be warned before WWDG expiration. The Early Wakeup Interrupt (EWI) can be used if specific safety operations or data logging must be performed before the actual reset is generated. When the downcounter reaches 0x40, interrupt occurs. This mechanism requires WWDG interrupt line to be enabled in NVIC. Once enabled, EWI interrupt cannot be disabled except by a system reset.
- WWDGRST flag in RCC CSR register can be used to inform when a WWDG reset occurs.
- The WWDG counter input clock is derived from the APB clock divided by a programmable prescaler.
- WWDG clock (Hz) = PCLK1 / (4096 * Prescaler)
- WWDG timeout (mS) = 1000 * (T[5;0] + 1) / WWDG clock (Hz) where T[5;0] are the lowest 6 bits of Counter.
- WWDG Counter refresh is allowed between the following limits :
  - min time (mS) = 1000 * (Counter - Window) / WWDG clock
  - max time (mS) = 1000 * (Counter - 0x40) / WWDG clock
- Typical values:
  - Counter min (T[5;0] = 0x00) at 64 MHz (PCLK1) with zero prescaler: max timeout before reset: approximately 64us
  - Counter max (T[5;0] = 0x3F) at 64 MHz (PCLK1) with prescaler dividing by 128: max timeout before reset: approximately 524.28ms

83.2.2 How to use this driver

Common driver usage

- Enable WWDG APB1 clock using __HAL_RCC_WWDG_CLK_ENABLE().
- Configure the WWDG prescaler, refresh window value, counter value and early interrupt status using HAL_WWDG_Init() function. This will automatically enable WWDG and start its downcounter. Time reference can be taken from function exit. Care must be taken to provide a counter value greater than 0x40 to prevent generation of immediate reset.
- If the Early Wakeup Interrupt (EWI) feature is enabled, an interrupt is generated when the counter reaches 0x40. When HAL_WWDG_IRQHandler is triggered by the interrupt service routine, flag will be automatically cleared and HAL_WWDG_WakeupCallback user callback will be executed. User can add his own code by customization of callback HAL_WWDG_WakeupCallback.
- Then the application program must refresh the WWDG counter at regular intervals during normal operation to prevent an MCU reset, using HAL_WWDG_Refresh() function. This operation must occur only when the counter is lower than the refresh window value already programmed.

Callback registration

The compilation define USE_HAL_WWDG_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use Functions HAL_WWDG_RegisterCallback() to register a user callback.

- Function HAL_WWDG_RegisterCallback() allows to register following callbacks:
  - EwiCallback : callback for Early WakeUp Interrupt.
  - MspInitCallback : WWDG MspInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.
- Use function HAL_WWDG_UnRegisterCallback() to reset a callback to the default weak (surcharged) function. HAL_WWDG_UnRegisterCallback() takes as parameters the HAL peripheral handle and the Callback ID. This function allows to reset following callbacks:
  - EwiCallback : callback for Early WakeUp Interrupt.
  - MspInitCallback : WWDG MspInit.

When calling HAL_WWDG_Init function, callbacks are reset to the corresponding legacy weak (surcharged) functions: HAL_WWDG_EarlyWakeupCallback() and HAL_WWDG_MspInit() only if they have not been registered before.

When compilation define USE_HAL_WWDG_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak (surcharged) callbacks are used.

WWDG HAL driver macros list

Below the list of available macros in WWDG HAL driver.

- __HAL_WWDG_ENABLE: Enable the WWDG peripheral
- __HAL_WWDG_GET_FLAG: Get the selected WWDG's flag status
- __HAL_WWDG_CLEAR_FLAG: Clear the WWDG's pending flags
- __HAL_WWDG_ENABLE_IT: Enable the WWDG early wakeup interrupt

83.2.3 Detailed description of functions

HAL_WWDG_Init

Function name

HAL_StatusTypeDef HAL_WWDG_Init (WWDG_HandleTypeDef * hwwdg)

Function description

HAL_WWDG_MspInit

Function name

void HAL_WWDG_MspInit (WWDG_HandleTypeDef * hwwdg)

Function description

HAL_WWDG_RegisterCallback

Function name

HAL_StatusTypeDef HAL_WWDG_RegisterCallback (WWDG_HandleTypeDef * hwwdg, HAL_WWDG_CallbackIDTypeDef CallbackID, pWWDG_CallbackTypeDef pCallback)

Function description

HAL_WWDG_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_WWDG_UnRegisterCallback (WWDG_HandleTypeDef * hwwdg, HAL_WWDG_CallbackIDTypeDef CallbackID)

Function description

HAL_WWDG_Refresh

Function name

HAL_StatusTypeDef HAL_WWDG_Refresh (WWDG_HandleTypeDef * hwwdg)

Function description

HAL_WWDG_IRQHandler

Function name

void HAL_WWDG_IRQHandler (WWDG_HandleTypeDef * hwwdg)

Function description

HAL_WWDG_EarlyWakeupCallback

Function name

void HAL_WWDG_EarlyWakeupCallback (WWDG_HandleTypeDef * hwwdg)

Function description

83.3 WWDG Firmware driver defines

The following section lists the various define and macros of the module.

83.3.1 WWDG

WWDG

WWDG Early Wakeup Interrupt Mode

WWDG_EWI_DISABLE

EWI Disable

WWDG_EWI_ENABLE

EWI Enable

WWDG Exported Macros

__HAL_WWDG_ENABLE

Description:

- Enable the WWDG peripheral.

Parameters:

- __HANDLE__: WWDG handle

Return value:

- None

__HAL_WWDG_ENABLE_IT

Description:

- Enable the WWDG early wakeup interrupt.

Parameters:

- __HANDLE__: WWDG handle
- __INTERRUPT__: specifies the interrupt to enable. This parameter can be one of the following values:
  - WWDG_IT_EWI: Early wakeup interrupt

Return value:

- None

Notes:

- Once enabled this interrupt cannot be disabled except by a system reset.

__HAL_WWDG_GET_IT

Description:

- Check whether the selected WWDG interrupt has occurred or not.

Parameters:

- __HANDLE__: WWDG handle
- __INTERRUPT__: specifies the it to check. This parameter can be one of the following values:
  - WWDG_FLAG_EWIF: Early wakeup interrupt IT

Return value:

- The: new state of WWDG_FLAG (SET or RESET).

__HAL_WWDG_CLEAR_IT

Description:

- Clear the WWDG interrupt pending bits.

Parameters:

- __HANDLE__: WWDG handle
- __INTERRUPT__: specifies the interrupt pending bit to clear. This parameter can be one of the following values:
  - WWDG_FLAG_EWIF: Early wakeup interrupt flag

__HAL_WWDG_GET_FLAG

Description:

- Check whether the specified WWDG flag is set or not.

Parameters:

- __HANDLE__: WWDG handle
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - WWDG_FLAG_EWIF: Early wakeup interrupt flag

Return value:

- The: new state of WWDG_FLAG (SET or RESET).

__HAL_WWDG_CLEAR_FLAG

Description:

- Clear the WWDG's pending flags.

Parameters:

- __HANDLE__: WWDG handle
- __FLAG__: specifies the flag to clear. This parameter can be one of the following values:
  - WWDG_FLAG_EWIF: Early wakeup interrupt flag

Return value:

- None

__HAL_WWDG_GET_IT_SOURCE

Description:

- Check whether the specified WWDG interrupt source is enabled or not.

Parameters:

- __HANDLE__: WWDG Handle.
- __INTERRUPT__: specifies the WWDG interrupt source to check. This parameter can be one of the following values:
  - WWDG_IT_EWI: Early Wakeup Interrupt

Return value:

- state: of __INTERRUPT__ (TRUE or FALSE).

WWDG Flag definition

WWDG_FLAG_EWIF

Early wakeup interrupt flag

WWDG Interrupt definition

WWDG_IT_EWI

Early wakeup interrupt

WWDG Prescaler

WWDG_PRESCALER_1

WWDG counter clock = (PCLK1/4096)/1

WWDG_PRESCALER_2

WWDG counter clock = (PCLK1/4096)/2

WWDG_PRESCALER_4

WWDG counter clock = (PCLK1/4096)/4

WWDG_PRESCALER_8

WWDG counter clock = (PCLK1/4096)/8

WWDG_PRESCALER_16

WWDG counter clock = (PCLK1/4096)/16

WWDG_PRESCALER_32

WWDG counter clock = (PCLK1/4096)/32

WWDG_PRESCALER_64

WWDG counter clock = (PCLK1/4096)/64

WWDG_PRESCALER_128

WWDG counter clock = (PCLK1/4096)/128

84 HAL XSPI Generic Driver

84.1 XSPI Firmware driver registers structures

84.1.1 XSPI_InitTypeDef

XSPI_InitTypeDef is defined in the stm32n6xx_hal_xspi.h

Data Fields

- ***uint32_t FifoThresholdByte***
- ***uint32_t MemoryMode***
- ***uint32_t MemoryType***
- ***uint32_t MemorySize***
- ***uint32_t ChipSelectHighTimeCycle***
- ***uint32_t FreeRunningClock***
- ***uint32_t ClockMode***
- ***uint32_t WrapSize***
- ***uint32_t ClockPrescaler***
- ***uint32_t SampleShifting***
- ***uint32_t DelayHoldQuarterCycle***
- ***uint32_t ChipSelectBoundary***
- ***uint32_t MaxTran***
- ***uint32_t Refresh***
- ***uint32_t MemorySelect***
- ***uint32_t MemoryExtended***

Field Documentation

- ***uint32_t XSPI_InitTypeDef::FifoThresholdByte***
  This is the threshold used by the Peripheral to generate the interrupt indicating that data are available in reception or free place is available in transmission. For XSPI, this parameter can be a value between 1 and 64
- ***uint32_t XSPI_InitTypeDef::MemoryMode***
  It Specifies the memory mode. This parameter can be a value of ***XSPI_MemoryMode***
- ***uint32_t XSPI_InitTypeDef::MemoryType***
  It indicates the external device type connected to the XSPI. This parameter can be a value of ***XSPI_MemoryType***
- ***uint32_t XSPI_InitTypeDef::MemorySize***
  It defines the size of the external device connected to the XSPI, it corresponds to the number of address bits required to access the external device. This parameter can be a value of ***XSPI_MemorySize***
- ***uint32_t XSPI_InitTypeDef::ChipSelectHighTimeCycle***
  It defines the minimum number of clocks which the chip select must remain high between commands. This parameter can be a value between 1 and 64U
- ***uint32_t XSPI_InitTypeDef::FreeRunningClock***
  It enables or not the free running clock. This parameter can be a value of ***XSPI_FreeRunningClock***
- ***uint32_t XSPI_InitTypeDef::ClockMode***
  It indicates the level of clock when the chip select is released. This parameter can be a value of ***XSPI_ClockMode***
- ***uint32_t XSPI_InitTypeDef::WrapSize***
  It indicates the wrap-size corresponding the external device configuration. This parameter can be a value of ***XSPI_WrapSize***
- ***uint32_t XSPI_InitTypeDef::ClockPrescaler***
  It specifies the prescaler factor used for generating the external clock based on the AHB clock. This parameter can be a value between 0 and 255U

- ***uint32_t XSPI_InitTypeDef::SampleShifting***
  It allows to delay to 1/2 cycle the data sampling in order to take in account external signal delays. This parameter can be a value of ***XSPI_SampleShifting***
- ***uint32_t XSPI_InitTypeDef::DelayHoldQuarterCycle***
  It allows to hold to 1/4 cycle the data. This parameter can be a value of ***XSPI_DelayHoldQuarterCycle***
- ***uint32_t XSPI_InitTypeDef::ChipSelectBoundary***
  It enables the transaction boundary feature and defines the boundary of bytes to release the chip select. This parameter can be a value of ***XSPI_ChipSelectBoundary***
- ***uint32_t XSPI_InitTypeDef::MaxTran***
  It enables the communication regulation feature. The chip select is released every MaxTran+1 bytes when the other XSPI request the access to the bus. This parameter can be a value between 0 and 255U
- ***uint32_t XSPI_InitTypeDef::Refresh***
  It enables the refresh rate feature. The chip select is released every Refresh+1 clock cycles. This parameter can be a value between 0 and 0xFFFFFFFF
- ***uint32_t XSPI_InitTypeDef::MemorySelect***
  It indicates if the output of nCS. This parameter can be a value of ***XSPI_MemorySelect***
- ***uint32_t XSPI_InitTypeDef::MemoryExtended***
  If available, It indicates if NCS1 and NCS2 are software or hardware controlled when one XSPI drives two same size external memories located in contiguous places in the memory map. This parameter can be a value of ***XSPI_MemoryExtended***

84.1.2 __XSPI_HandleTypeDef

__XSPI_HandleTypeDef is defined in the stm32n6xx_hal_xspi.h

Data Fields

- ***XSPI_TypeDef * Instance***
- ***XSPI_InitTypeDef Init***
- ***uint8_t * pBuffPtr***
- ***__IO uint32_t XferSize***
- ***__IO uint32_t XferCount***
- ***DMA_HandleTypeDef * hdmatx***
- ***DMA_HandleTypeDef * hdmarx***
- ***__IO uint32_t State***
- ***__IO uint32_t ErrorCode***
- ***uint32_t Timeout***
- ***void(* ErrorCallback***
- ***void(* AbortCpltCallback***
- ***void(* FifoThresholdCallback***
- ***void(* CmdCpltCallback***
- ***void(* RxCpltCallback***
- ***void(* TxCpltCallback***
- ***void(* RxHalfCpltCallback***
- ***void(* TxHalfCpltCallback***
- ***void(* StatusMatchCallback***
- ***void(* TimeOutCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***XSPI_TypeDef* __XSPI_HandleTypeDef::Instance***
  XSPI registers base address
- ***XSPI_InitTypeDef __XSPI_HandleTypeDef::Init***
  XSPI initialization parameters
- ***uint8_t* __XSPI_HandleTypeDef::pBuffPtr***
  Address of the XSPI buffer for transfer

- ***__IO uint32_t __XSPI_HandleTypeDef::XferSize***
  Number of data to transfer
- ***__IO uint32_t __XSPI_HandleTypeDef::XferCount***
  Counter of data transferred
- ***DMA_HandleTypeDef* __XSPI_HandleTypeDef::hdmatx***
  Handle of the DMA channel used for transmit
- ***DMA_HandleTypeDef* __XSPI_HandleTypeDef::hdmarx***
  Handle of the DMA channel used for receive
- ***__IO uint32_t __XSPI_HandleTypeDef::State***
  Internal state of the XSPI HAL driver
- ***__IO uint32_t __XSPI_HandleTypeDef::ErrorCode***
  Error code in case of HAL driver internal error
- ***uint32_t __XSPI_HandleTypeDef::Timeout***
  Timeout used for the XSPI external device access
- ***void(* __XSPI_HandleTypeDef::ErrorCallback)(struct __XSPI_HandleTypeDef *hxspi)***
- ***void(* __XSPI_HandleTypeDef::AbortCpltCallback)(struct __XSPI_HandleTypeDef *hxspi)***
- ***void(* __XSPI_HandleTypeDef::FifoThresholdCallback)(struct __XSPI_HandleTypeDef *hxspi)***
- ***void(* __XSPI_HandleTypeDef::CmdCpltCallback)(struct __XSPI_HandleTypeDef *hxspi)***
- ***void(* __XSPI_HandleTypeDef::RxCpltCallback)(struct __XSPI_HandleTypeDef *hxspi)***
- ***void(* __XSPI_HandleTypeDef::TxCpltCallback)(struct __XSPI_HandleTypeDef *hxspi)***
- ***void(* __XSPI_HandleTypeDef::RxHalfCpltCallback)(struct __XSPI_HandleTypeDef *hxspi)***
- ***void(* __XSPI_HandleTypeDef::TxHalfCpltCallback)(struct __XSPI_HandleTypeDef *hxspi)***
- ***void(* __XSPI_HandleTypeDef::StatusMatchCallback)(struct __XSPI_HandleTypeDef *hxspi)***
- ***void(* __XSPI_HandleTypeDef::TimeOutCallback)(struct __XSPI_HandleTypeDef *hxspi)***
- ***void(* __XSPI_HandleTypeDef::MspInitCallback)(struct __XSPI_HandleTypeDef *hxspi)***
- ***void(* __XSPI_HandleTypeDef::MspDeInitCallback)(struct __XSPI_HandleTypeDef *hxspi)***

84.1.3 XSPI_RegularCmdTypeDef

XSPI_RegularCmdTypeDef is defined in the stm32n6xx_hal_xspi.h

Data Fields

- ***uint32_t OperationType***
- ***uint32_t IOSelect***
- ***uint32_t Instruction***
- ***uint32_t InstructionMode***
- ***uint32_t InstructionWidth***
- ***uint32_t InstructionDTRMode***
- ***uint32_t Address***
- ***uint32_t AddressMode***
- ***uint32_t AddressWidth***
- ***uint32_t AddressDTRMode***
- ***uint32_t AlternateBytes***
- ***uint32_t AlternateBytesMode***
- ***uint32_t AlternateBytesWidth***
- ***uint32_t AlternateBytesDTRMode***
- ***uint32_t DataMode***
- ***uint32_t DataLength***
- ***uint32_t DataDTRMode***
- ***uint32_t DummyCycles***
- ***uint32_t DQSMode***

Field Documentation

- ***uint32_t XSPI_RegularCmdTypeDef::OperationType***
  It indicates if the configuration applies to the common registers or to the registers for the write operation (these registers are only used for memory-mapped mode). This parameter can be a value of ***XSPI_OperationType***
- ***uint32_t XSPI_RegularCmdTypeDef::IOSelect***
  It indicates the IOs used to exchange data with external memory. This parameter can be a value of ***XSPI_IOSelect***
- ***uint32_t XSPI_RegularCmdTypeDef::Instruction***
  It contains the instruction to be sent to the device. This parameter can be a value between 0 and 0xFFFFFFFFU
- ***uint32_t XSPI_RegularCmdTypeDef::InstructionMode***
  It indicates the mode of the instruction. This parameter can be a value of ***XSPI_InstructionMode***
- ***uint32_t XSPI_RegularCmdTypeDef::InstructionWidth***
  It indicates the width of the instruction. This parameter can be a value of ***XSPI_InstructionWidth***
- ***uint32_t XSPI_RegularCmdTypeDef::InstructionDTRMode***
  It enables or not the DTR mode for the instruction phase. This parameter can be a value of ***XSPI_InstructionDTRMode***
- ***uint32_t XSPI_RegularCmdTypeDef::Address***
  It contains the address to be sent to the device. This parameter can be a value between 0 and 0xFFFFFFFFU
- ***uint32_t XSPI_RegularCmdTypeDef::AddressMode***
  It indicates the address mode. Address mode precises number of lines for address (except no address). This parameter can be a value of ***XSPI_AddressMode***
- ***uint32_t XSPI_RegularCmdTypeDef::AddressWidth***
  It indicates the width of the address. This parameter can be a value of ***XSPI_AddressWidth***
- ***uint32_t XSPI_RegularCmdTypeDef::AddressDTRMode***
  It enables or not the DTR mode for the address phase. This parameter can be a value of ***XSPI_AddressDTRMode***
- ***uint32_t XSPI_RegularCmdTypeDef::AlternateBytes***
  It contains the alternate bytes to be sent to the device. This parameter can be a value between 0 and 0xFFFFFFFFU
- ***uint32_t XSPI_RegularCmdTypeDef::AlternateBytesMode***
  It indicates the mode of the alternate bytes. This parameter can be a value of ***XSPI_AlternateBytesMode***
- ***uint32_t XSPI_RegularCmdTypeDef::AlternateBytesWidth***
  It indicates the width of the alternate bytes. This parameter can be a value of ***XSPI_AlternateBytesWidth***
- ***uint32_t XSPI_RegularCmdTypeDef::AlternateBytesDTRMode***
  It enables or not the DTR mode for the alternate bytes phase. This parameter can be a value of ***XSPI_AlternateBytesDTRMode***
- ***uint32_t XSPI_RegularCmdTypeDef::DataMode***
  It indicates the data mode. Data mode precises number of lines for data exchange (except no data). This parameter can be a value of ***XSPI_DataMode***
- ***uint32_t XSPI_RegularCmdTypeDef::DataLength***
  It indicates the number of data transferred with this command. This field is only used for indirect mode. This parameter can be a value between 1 and 0xFFFFFFFFU
- ***uint32_t XSPI_RegularCmdTypeDef::DataDTRMode***
  It enables or not the DTR mode for the data phase. This parameter can be a value of ***XSPI_DataDTRMode***
- ***uint32_t XSPI_RegularCmdTypeDef::DummyCycles***
  It indicates the number of dummy cycles inserted before data phase. This parameter can be a value between 0 and 31U
- ***uint32_t XSPI_RegularCmdTypeDef::DQSMode***
  It enables or not the data strobe management. This parameter can be a value of ***XSPI_DQSMode***

84.1.4 XSPI_HyperbusCfgTypeDef

XSPI_HyperbusCfgTypeDef is defined in the stm32n6xx_hal_xspi.h

Data Fields

- ***uint32_t RWRecoveryTimeCycle***

- ***uint32_t AccessTimeCycle***
- ***uint32_t WriteZeroLatency***
- ***uint32_t LatencyMode***

Field Documentation

- ***uint32_t XSPI_HyperbusCfgTypeDef::RWRecoveryTimeCycle***
  It indicates the number of cycles for the device read write recovery time. This parameter can be a value between 0 and 255U
- ***uint32_t XSPI_HyperbusCfgTypeDef::AccessTimeCycle***
  It indicates the number of cycles for the device access time. This parameter can be a value between 0 and 255U
- ***uint32_t XSPI_HyperbusCfgTypeDef::WriteZeroLatency***
  It enables or not the latency for the write access. This parameter can be a value of ***XSPI_WriteZeroLatency***
- ***uint32_t XSPI_HyperbusCfgTypeDef::LatencyMode***
  It configures the latency mode. This parameter can be a value of ***XSPI_LatencyMode***

84.1.5 XSPI_HyperbusCmdTypeDef

XSPI_HyperbusCmdTypeDef is defined in the stm32n6xx_hal_xspi.h

Data Fields

- ***uint32_t AddressSpace***
- ***uint32_t Address***
- ***uint32_t AddressWidth***
- ***uint32_t DataLength***
- ***uint32_t DQSMode***
- ***uint32_t DataMode***

Field Documentation

- ***uint32_t XSPI_HyperbusCmdTypeDef::AddressSpace***
  It indicates the address space accessed by the command. This parameter can be a value of ***XSPI_AddressSpace***
- ***uint32_t XSPI_HyperbusCmdTypeDef::Address***
  It contains the address to be sent to the device. This parameter can be a value between 0 and 0xFFFFFFFFU
- ***uint32_t XSPI_HyperbusCmdTypeDef::AddressWidth***
  It indicates the width of the address. This parameter can be a value of ***XSPI_AddressWidth***
- ***uint32_t XSPI_HyperbusCmdTypeDef::DataLength***
  It indicates the number of data transferred with this command. This field is only used for indirect mode. This parameter can be a value between 1 and 0xFFFFFFFF In case of autopolling mode, this parameter can be any value between 1 and 4
- ***uint32_t XSPI_HyperbusCmdTypeDef::DQSMode***
  It enables or not the data strobe management. This parameter can be a value of ***XSPI_DQSMode***
- ***uint32_t XSPI_HyperbusCmdTypeDef::DataMode***
  It indicates the data mode. Data mode precises number of lines for data exchange (except no data). This parameter can be a value of ***XSPI_DataMode***

84.1.6 XSPI_AutoPollingTypeDef

XSPI_AutoPollingTypeDef is defined in the stm32n6xx_hal_xspi.h

Data Fields

- ***uint32_t MatchValue***
- ***uint32_t MatchMask***
- ***uint32_t MatchMode***
- ***uint32_t AutomaticStop***
- ***uint32_t IntervalTime***

Field Documentation

- ***uint32_t XSPI_AutoPollingTypeDef::MatchValue***
  Specifies the value to be compared with the masked status register to get a match. This parameter can be any value between 0 and 0xFFFFFFFFU
- ***uint32_t XSPI_AutoPollingTypeDef::MatchMask***
  Specifies the mask to be applied to the status bytes received. This parameter can be any value between 0 and 0xFFFFFFFFU
- ***uint32_t XSPI_AutoPollingTypeDef::MatchMode***
  Specifies the method used for determining a match. This parameter can be a value of ***XSPI_MatchMode***
- ***uint32_t XSPI_AutoPollingTypeDef::AutomaticStop***
  Specifies if automatic polling is stopped after a match. This parameter can be a value of ***XSPI_AutomaticStop***
- ***uint32_t XSPI_AutoPollingTypeDef::IntervalTime***
  Specifies the number of clock cycles between two read during automatic polling phases. This parameter can be any value between 0 and 0xFFFFU

84.1.7 XSPI_MemoryMappedTypeDef

XSPI_MemoryMappedTypeDef is defined in the stm32n6xx_hal_xspi.h

Data Fields

- ***uint32_t TimeOutActivation***
- ***uint32_t TimeoutPeriodClock***
- ***uint32_t NoPrefetchData***
- ***uint32_t NoPrefetchAXI***

Field Documentation

- ***uint32_t XSPI_MemoryMappedTypeDef::TimeOutActivation***
  Specifies if the timeout counter is enabled to release the chip select. This parameter can be a value of ***XSPI_TimeOutActivation***
- ***uint32_t XSPI_MemoryMappedTypeDef::TimeoutPeriodClock***
  Specifies the number of clock to wait when the FIFO is full before to release the chip select. This parameter can be any value between 0 and 0xFFFFU
- ***uint32_t XSPI_MemoryMappedTypeDef::NoPrefetchData***
  Specifies if the automatic prefetch in the external memory is enabled or not. This parameter can be a value of ***XSPI_NoPrefetchData***
- ***uint32_t XSPI_MemoryMappedTypeDef::NoPrefetchAXI***
  Specifies if the automatic prefetch in the external memory when the corresponding AXI transaction is signaled as not-prefetchable, is enabled or not. This parameter can be a value of ***XSPI_NoPrefetchAXI***

84.1.8 XSPIM_CfgTypeDef

XSPIM_CfgTypeDef is defined in the stm32n6xx_hal_xspi.h

Data Fields

- ***uint32_t nCSOverride***
- ***uint32_t IOPort***
- ***uint32_t Req2AckTime***

Field Documentation

- ***uint32_t XSPIM_CfgTypeDef::nCSOverride***
  It indicates Chip select selector override setting for XSPI. This parameter can be a value ***XSPIM_MemorySelect_Override***
- ***uint32_t XSPIM_CfgTypeDef::IOPort***
  It indicates which port of the XSPI IO Manager is used for the instance. This parameter can be a value of ***XSPI_IO_Manger_IOPort***
- ***uint32_t XSPIM_CfgTypeDef::Req2AckTime***
  It indicates the minimum switching duration (in number of clock cycles) expected if some signals are multiplexed in the XSPI IO Manager with the other XSPI. This parameter can be a value between 1 and 256

84.1.9 XSPI_HSCalTypeDef

XSPI_HSCalTypeDef is defined in the stm32n6xx_hal_xspi.h

Data Fields

- ***uint32_t DelayValueType***
- ***uint32_t FineCalibrationUnit***
- ***uint32_t CoarseCalibrationUnit***
- ***uint32_t MaxCalibration***

Field Documentation

- ***uint32_t XSPI_HSCalTypeDef::DelayValueType***
  It indicates which calibration is concerned by the configuration. This parameter can be a value of ***XSPI_DelayType***
- ***uint32_t XSPI_HSCalTypeDef::FineCalibrationUnit***
  It indicates the fine calibration value of the delay. This parameter can be a value between 0 and 0x7FU
- ***uint32_t XSPI_HSCalTypeDef::CoarseCalibrationUnit***
  It indicates the coarse calibration value of the delay. This parameter can be a value between 0 and 0x1FU
- ***uint32_t XSPI_HSCalTypeDef::MaxCalibration***
  It indicates that the calibration is outside the range of DLL master. It applies only when the DelayValueType is HAL_XSPI_CAL_FULL_CYCLE_DELAY. This parameter can be a value of ***XSPI_MaxCal***

84.2 XSPI Firmware driver API description

The following section lists the various functions of the XSPI library.

84.2.1 How to use this driver

Initialization

As prerequisite, fill in the HAL_XSPI_MspInit() :

- Enable XSPI clocks interface with __HAL_RCC_XSPI_CLK_ENABLE().
- Reset XSPI Peripheral with __HAL_RCC_XSPI_FORCE_RESET() and __HAL_RCC_XSPI_RELEASE_RESET().
- Enable the clocks for the XSPI GPIOS with __HAL_RCC_GPIOx_CLK_ENABLE().
- Configure these XSPI pins in alternate mode using HAL_GPIO_Init().
- If interrupt or DMA mode is used, enable and configure XSPI global interrupt with HAL_NVIC_SetPriority() and HAL_NVIC_EnableIRQ().
- If DMA mode is used, enable the clocks for the XSPI DMA channel with __HAL_RCC_DMAx_CLK_ENABLE(), configure DMA with HAL_DMA_Init(), link it with XSPI handle using __HAL_LINKDMA(), enable and configure DMA channel global interrupt with HAL_NVIC_SetPriority() and HAL_NVIC_EnableIRQ().

Configure the fifo threshold, the memory mode, the memory type, the device size, the CS high time, the free running clock, the clock mode, the wrap size, the clock prescaler, the sample shifting, the hold delay and the CS boundary using the HAL_XSPI_Init() function.

When using Hyperbus, configure the RW recovery time, the access time, the write latency and the latency mode using the HAL_XSPI_HyperbusCfg() function.

Indirect functional mode

In regular mode, configure the command sequence using the HAL_XSPI_Command() or HAL_XSPI_Command_IT() functions :

- Instruction phase : the mode used and if present the size, the instruction opcode and the DTR mode.
- Address phase : the mode used and if present the size, the address value and the DTR mode.
- Alternate-bytes phase : the mode used and if present the size, the alternate bytes values and the DTR mode.
- Dummy-cycles phase : the number of dummy cycles (mode used is same as data phase).
- Data phase : the mode used and if present the number of bytes and the DTR mode.

- Data strobe (DQS) mode : the activation (or not) of this mode
- IO selection : to access external memory.
- Operation type : always common configuration.

In Hyperbus mode, configure the command sequence using the HAL_XSPI_HyperbusCmd() function :

- Address space : indicate if the access will be done in register or memory
- Address size
- Number of data
- Data strobe (DQS) mode : the activation (or not) of this mode

If no data is required for the command (only for regular mode, not for Hyperbus mode), it is sent directly to the memory :

- In polling mode, the output of the function is done when the transfer is complete.
- In interrupt mode, HAL_XSPI_CmdCpltCallback() will be called when the transfer is complete.

For the indirect write mode, use HAL_XSPI_Transmit(), HAL_XSPI_Transmit_DMA() or HAL_XSPI_Transmit_IT() after the command configuration :

- In polling mode, the output of the function is done when the transfer is complete.
- In interrupt mode, HAL_XSPI_FifoThresholdCallback() will be called when the fifo threshold is reached and HAL_XSPI_TxCpltCallback() will be called when the transfer is complete.
- In DMA mode, HAL_XSPI_TxHalfCpltCallback() will be called at the half transfer and HAL_XSPI_TxCpltCallback() will be called when the transfer is complete.

For the indirect read mode, use HAL_XSPI_Receive(), HAL_XSPI_Receive_DMA() or HAL_XSPI_Receive_IT() after the command configuration :

- In polling mode, the output of the function is done when the transfer is complete.
- In interrupt mode, HAL_XSPI_FifoThresholdCallback() will be called when the fifo threshold is reached and HAL_XSPI_RxCpltCallback() will be called when the transfer is complete.
- In DMA mode, HAL_XSPI_RxHalfCpltCallback() will be called at the half transfer and HAL_XSPI_RxCpltCallback() will be called when the transfer is complete.

Auto-polling functional mode

Configure the command sequence by the same way than the indirect mode

Configure the auto-polling functional mode using the HAL_XSPI_AutoPolling() or HAL_XSPI_AutoPolling_IT() functions :

- The size of the status bytes, the match value, the mask used, the match mode (OR/AND), the polling interval and the automatic stop activation.

After the configuration :

- In polling mode, the output of the function is done when the status match is reached. The automatic stop is activated to avoid an infinite loop.
- In interrupt mode, HAL_XSPI_StatusMatchCallback() will be called each time the status match is reached.

Memory-mapped functional mode

Configure the command sequence by the same way than the indirect mode except for the operation type in regular mode :

- Operation type equals to read configuration : the command configuration applies to read access in memory-mapped mode
- Operation type equals to write configuration : the command configuration applies to write access in memory-mapped mode
- Both read and write configuration should be performed before activating memory-mapped mode

Configure the memory-mapped functional mode using the HAL_XSPI_MemoryMapped() functions :

- The timeout activation and the timeout period.

After the configuration, the XSPI will be used as soon as an access on the AHB is done on the address range. HAL_XSPI_TimeOutCallback() will be called when the timeout expires.

Errors management and abort functionality

HAL_XSPI_GetError() function gives the error raised during the last operation.

HAL_XSPI_Abort() and HAL_XSPI_AbortIT() functions aborts any on-going operation and flushes the fifo :

- In polling mode, the output of the function is done when the transfer complete bit is set and the busy bit cleared.
- In interrupt mode, HAL_XSPI_AbortCpltCallback() will be called when the transfer complete bit is set.

Control functions

HAL_XSPI_GetState() function gives the current state of the HAL XSPI driver.

HAL_XSPI_SetTimeout() function configures the timeout value used in the driver.

HAL_XSPI_SetFifoThreshold() function configures the threshold on the Fifo of the XSPI Peripheral.

HAL_XSPI_SetMemoryType() function configures the type of the external memory.

HAL_XSPI_SetDeviceSize() function configures the size of the external memory.

HAL_XSPI_SetClockPrescaler() function configures the clock prescaler of the XSPI Peripheral.

HAL_XSPI_GetFifoThreshold() function gives the current of the Fifo's threshold

IO manager configuration functions

HAL_XSPIM_Config() function configures the IO manager for the XSPI instance.

High-speed interface and calibration functions

The purpose of High-speed interface is primary to shift data or data strobe by one quarter of octal bus clock period, with a correct timing accuracy. DLL must be calibrated versus this clock period. The calibration process is automatically enabled when one of the three conditions below is met:

- The XSPI exits Reset state.
- The Prescaler is set.
- The configuration of communication is set.

HAL_XSPI_GetDelayValue() function Get the delay values of the high-speed interface DLLs..

HAL_XSPI_SetDelayValue() function Set the delay values of the high-speed interface DLLs..

Callback registration

The compilation define USE_HAL_XSPI_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks.

Use function HAL_XSPI_RegisterCallback() to register a user callback, it allows to register following callbacks:

- ErrorCallback : callback when error occurs.
- AbortCpltCallback : callback when abort is completed.
- FifoThresholdCallback : callback when the fifo threshold is reached.
- CmdCpltCallback : callback when a command without data is completed.
- RxCpltCallback : callback when a reception transfer is completed.
- TxCpltCallback : callback when a transmission transfer is completed.
- RxHalfCpltCallback : callback when half of the reception transfer is completed.
- TxHalfCpltCallback : callback when half of the transmission transfer is completed.
- StatusMatchCallback : callback when a status match occurs.
- TimeOutCallback : callback when the timeout perioed expires.
- MspInitCallback : XSPI MspInit.
- MspDeInitCallback : XSPI MspDeInit.

This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

Use function HAL_XSPI_UnRegisterCallback() to reset a callback to the default weak (overridden) function. It allows to reset following callbacks:

- ErrorCallback : callback when error occurs.

- AbortCpltCallback : callback when abort is completed.
- FifoThresholdCallback : callback when the fifo threshold is reached.
- CmdCpltCallback : callback when a command without data is completed.
- RxCpltCallback : callback when a reception transfer is completed.
- TxCpltCallback : callback when a transmission transfer is completed.
- RxHalfCpltCallback : callback when half of the reception transfer is completed.
- TxHalfCpltCallback : callback when half of the transmission transfer is completed.
- StatusMatchCallback : callback when a status match occurs.
- TimeOutCallback : callback when the timeout perioed expires.
- MspInitCallback : XSPI MspInit.
- MspDeInitCallback : XSPI MspDeInit.

This function) takes as parameters the HAL peripheral handle and the Callback ID.

By default, after the HAL_XSPI_Init() and if the state is HAL_XSPI_STATE_RESET all callbacks are reset to the corresponding legacy weak (overridden) functions. Exception done for MspInit and MspDeInit callbacks that are respectively reset to the legacy weak (overridden) functions in the HAL_XSPI_Init() and HAL_XSPI_DeInit() only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the HAL_XSPI_Init() and HAL_XSPI_DeInit() keep and use the user MspInit/MspDeInit callbacks (registered beforehand)

Callbacks can be registered/unregistered in READY state only. Exception done for MspInit/MspDeInit callbacks that can be registered/unregistered in READY or RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_XSPI_RegisterCallback() before calling HAL_XSPI_DeInit() or HAL_XSPI_Init() function.

When The compilation define USE_HAL_XSPI_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak (overridden) callbacks are used.

84.2.2 Detailed description of functions

HAL_XSPI_Init

Function name

HAL_StatusTypeDef HAL_XSPI_Init (XSPI_HandleTypeDef * hxspi)

Function description

HAL_XSPI_MspInit

Function name

void HAL_XSPI_MspInit (XSPI_HandleTypeDef * hxspi)

Function description

HAL_XSPI_DeInit

Function name

HAL_StatusTypeDef HAL_XSPI_DeInit (XSPI_HandleTypeDef * hxspi)

Function description

HAL_XSPI_MspDeInit

Function name

void HAL_XSPI_MspDeInit (XSPI_HandleTypeDef * hxspi)

Function description

HAL_XSPI_IRQHandler

Function name

void HAL_XSPI_IRQHandler (XSPI_HandleTypeDef * hxspi)

Function description

HAL_XSPI_Command

Function name

HAL_StatusTypeDef HAL_XSPI_Command (XSPI_HandleTypeDef * hxspi, const XSPI_RegularCmdTypeDef * pCmd, uint32_t Timeout)

Function description

HAL_XSPI_Command_IT

Function name

HAL_StatusTypeDef HAL_XSPI_Command_IT (XSPI_HandleTypeDef * hxspi, const XSPI_RegularCmdTypeDef * pCmd)

Function description

HAL_XSPI_HyperbusCfg

Function name

HAL_StatusTypeDef HAL_XSPI_HyperbusCfg (XSPI_HandleTypeDef * hxspi, const XSPI_HyperbusCfgTypeDef * pCfg, uint32_t Timeout)

Function description

HAL_XSPI_HyperbusCmd

Function name

HAL_StatusTypeDef HAL_XSPI_HyperbusCmd (XSPI_HandleTypeDef * hxspi, const XSPI_HyperbusCmdTypeDef * pCmd, uint32_t Timeout)

Function description

HAL_XSPI_Transmit

Function name

HAL_StatusTypeDef HAL_XSPI_Transmit (XSPI_HandleTypeDef * hxspi, const uint8_t * pData, uint32_t Timeout)

Function description

HAL_XSPI_Receive

Function name

HAL_StatusTypeDef HAL_XSPI_Receive (XSPI_HandleTypeDef * hxspi, uint8_t * pData, uint32_t Timeout)

Function description

HAL_XSPI_Transmit_IT

Function name

HAL_StatusTypeDef HAL_XSPI_Transmit_IT (XSPI_HandleTypeDef * hxspi, const uint8_t * pData)

Function description

HAL_XSPI_Receive_IT

Function name

HAL_StatusTypeDef HAL_XSPI_Receive_IT (XSPI_HandleTypeDef * hxspi, uint8_t * pData)

Function description

HAL_XSPI_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_XSPI_Transmit_DMA (XSPI_HandleTypeDef * hxspi, const uint8_t * pData)

Function description

HAL_XSPI_Receive_DMA

Function name

HAL_StatusTypeDef HAL_XSPI_Receive_DMA (XSPI_HandleTypeDef * hxspi, uint8_t * pData)

Function description

HAL_XSPI_AutoPolling

Function name

HAL_StatusTypeDef HAL_XSPI_AutoPolling (XSPI_HandleTypeDef * hxspi, const XSPI_AutoPollingTypeDef * pCfg, uint32_t Timeout)

Function description

HAL_XSPI_AutoPolling_IT

Function name

HAL_StatusTypeDef HAL_XSPI_AutoPolling_IT (XSPI_HandleTypeDef * hxspi, const XSPI_AutoPollingTypeDef * pCfg)

Function description

HAL_XSPI_MemoryMapped

Function name

HAL_StatusTypeDef HAL_XSPI_MemoryMapped (XSPI_HandleTypeDef * hxspi, const XSPI_MemoryMappedTypeDef * pCfg)

Function description

HAL_XSPI_ErrorCallback

Function name

void HAL_XSPI_ErrorCallback (XSPI_HandleTypeDef * hxspi)

Function description

HAL_XSPI_AbortCpltCallback

Function name

void HAL_XSPI_AbortCpltCallback (XSPI_HandleTypeDef * hxspi)

Function description

HAL_XSPI_FifoThresholdCallback

Function name

void HAL_XSPI_FifoThresholdCallback (XSPI_HandleTypeDef * hxspi)

Function description

HAL_XSPI_CmdCpltCallback

Function name

void HAL_XSPI_CmdCpltCallback (XSPI_HandleTypeDef * hxspi)

Function description

HAL_XSPI_RxCpltCallback

Function name

void HAL_XSPI_RxCpltCallback (XSPI_HandleTypeDef * hxspi)

Function description

HAL_XSPI_TxCpltCallback

Function name

void HAL_XSPI_TxCpltCallback (XSPI_HandleTypeDef * hxspi)

Function description

HAL_XSPI_RxHalfCpltCallback

Function name

void HAL_XSPI_RxHalfCpltCallback (XSPI_HandleTypeDef * hxspi)

Function description

HAL_XSPI_TxHalfCpltCallback

Function name

void HAL_XSPI_TxHalfCpltCallback (XSPI_HandleTypeDef * hxspi)

Function description

HAL_XSPI_StatusMatchCallback

Function name

void HAL_XSPI_StatusMatchCallback (XSPI_HandleTypeDef * hxspi)

Function description

HAL_XSPI_TimeOutCallback

Function name

void HAL_XSPI_TimeOutCallback (XSPI_HandleTypeDef * hxspi)

Function description

HAL_XSPI_RegisterCallback

Function name

HAL_StatusTypeDef HAL_XSPI_RegisterCallback (XSPI_HandleTypeDef * hxspi, HAL_XSPI_CallbackIDTypeDef CallbackID, pXSPI_CallbackTypeDef pCallback)

Function description

HAL_XSPI_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_XSPI_UnRegisterCallback (XSPI_HandleTypeDef * hxspi, HAL_XSPI_CallbackIDTypeDef CallbackID)

Function description

HAL_XSPI_Abort

Function name

HAL_StatusTypeDef HAL_XSPI_Abort (XSPI_HandleTypeDef * hxspi)

Function description

HAL_XSPI_Abort_IT

Function name

HAL_StatusTypeDef HAL_XSPI_Abort_IT (XSPI_HandleTypeDef * hxspi)

Function description

HAL_XSPI_SetFifoThreshold

Function name

HAL_StatusTypeDef HAL_XSPI_SetFifoThreshold (XSPI_HandleTypeDef * hxspi, uint32_t Threshold)

Function description

HAL_XSPI_GetFifoThreshold

Function name

uint32_t HAL_XSPI_GetFifoThreshold (const XSPI_HandleTypeDef * hxspi)

Function description

HAL_XSPI_SetMemoryType

Function name

HAL_StatusTypeDef HAL_XSPI_SetMemoryType (XSPI_HandleTypeDef * hxspi, uint32_t Type)

Function description

HAL_XSPI_SetDeviceSize

Function name

HAL_StatusTypeDef HAL_XSPI_SetDeviceSize (XSPI_HandleTypeDef * hxspi, uint32_t Size)

Function description

HAL_XSPI_SetClockPrescaler

Function name

HAL_StatusTypeDef HAL_XSPI_SetClockPrescaler (XSPI_HandleTypeDef * hxspi, uint32_t Prescaler)

Function description

HAL_XSPI_SetTimeout

Function name

HAL_StatusTypeDef HAL_XSPI_SetTimeout (XSPI_HandleTypeDef * hxspi, uint32_t Timeout)

Function description

HAL_XSPI_GetError

Function name

uint32_t HAL_XSPI_GetError (const XSPI_HandleTypeDef * hxspi)

Function description

HAL_XSPI_GetState

Function name

uint32_t HAL_XSPI_GetState (const XSPI_HandleTypeDef * hxspi)

Function description

HAL_XSPIM_Config

Function name

HAL_StatusTypeDef HAL_XSPIM_Config (XSPI_HandleTypeDef * hxspi, const XSPIM_CfgTypeDef * pCfg, uint32_t Timeout)

Function description

HAL_XSPI_GetDelayValue

Function name

HAL_StatusTypeDef HAL_XSPI_GetDelayValue (XSPI_HandleTypeDef * hxspi, XSPI_HSCalTypeDef * pCfg)

Function description

HAL_XSPI_SetDelayValue

Function name

HAL_StatusTypeDef HAL_XSPI_SetDelayValue (XSPI_HandleTypeDef * hxspi, const XSPI_HSCalTypeDef * pCfg)

Function description

84.3 XSPI Firmware driver defines

The following section lists the various define and macros of the module.

84.3.1 XSPI

XSPI

XSPI Address DTR Mode

HAL_XSPI_ADDRESS_DTR_DISABLE

DTR mode disabled for address phase

HAL_XSPI_ADDRESS_DTR_ENABLE

DTR mode enabled for address phase

XSPI Address Mode

HAL_XSPI_ADDRESS_NONE

No address

HAL_XSPI_ADDRESS_1_LINE

Address on a single line

HAL_XSPI_ADDRESS_2_LINES

Address on two lines

HAL_XSPI_ADDRESS_4_LINES

Address on four lines

HAL_XSPI_ADDRESS_8_LINES

Address on eight lines

XSPI Hyperbus Address Space

HAL_XSPI_MEMORY_ADDRESS_SPACE

HyperBus memory mode

HAL_XSPI_REGISTER_ADDRESS_SPACE

HyperBus register mode

XSPI Address width

HAL_XSPI_ADDRESS_8_BITS

8-bit address

HAL_XSPI_ADDRESS_16_BITS

16-bit address

HAL_XSPI_ADDRESS_24_BITS

24-bit address

HAL_XSPI_ADDRESS_32_BITS

32-bit address

XSPI Alternate Bytes DTR Mode

HAL_XSPI_ALT_BYTES_DTR_DISABLE

DTR mode disabled for alternate bytes phase

HAL_XSPI_ALT_BYTES_DTR_ENABLE

DTR mode enabled for alternate bytes phase

XSPI Alternate Bytes Mode

HAL_XSPI_ALT_BYTES_NONE

No alternate bytes

HAL_XSPI_ALT_BYTES_1_LINE

Alternate bytes on a single line

HAL_XSPI_ALT_BYTES_2_LINES

Alternate bytes on two lines

HAL_XSPI_ALT_BYTES_4_LINES

Alternate bytes on four lines

HAL_XSPI_ALT_BYTES_8_LINES

Alternate bytes on eight lines

XSPI Alternate Bytes Width

HAL_XSPI_ALT_BYTES_8_BITS

8-bit alternate bytes

HAL_XSPI_ALT_BYTES_16_BITS

16-bit alternate bytes

HAL_XSPI_ALT_BYTES_24_BITS

24-bit alternate bytes

HAL_XSPI_ALT_BYTES_32_BITS

32-bit alternate bytes

XSPI Automatic Stop

HAL_XSPI_AUTOMATIC_STOP_DISABLE

AutoPolling stops only with abort or XSPI disabling

HAL_XSPI_AUTOMATIC_STOP_ENABLE

AutoPolling stops as soon as there is a match

XSPI Chip Select Boundary

HAL_XSPI_BONDARYOF_NONE

HAL_XSPI_BONDARYOF_16B

16 bits (2 Bytes = 2^(1))

HAL_XSPI_BONDARYOF_32B

32 bits (4 Bytes = 2^(2))

HAL_XSPI_BONDARYOF_64B

64 bits (8 Bytes = 2^(3))

HAL_XSPI_BONDARYOF_128B

128 bits (16 Bytes = 2^(4))

HAL_XSPI_BONDARYOF_256B

256 bits (32 Bytes = 2^(5))

HAL_XSPI_BONDARYOF_512B

512 bits (64 Bytes = 2^(6))

HAL_XSPI_BONDARYOF_1KB

1 Kbits (128 Bytes = 2^(7))

HAL_XSPI_BONDARYOF_2KB

2 Kbits (256 Bytes = 2^(8))

HAL_XSPI_BONDARYOF_4KB

4 Kbits (512 Bytes = 2^(9))

HAL_XSPI_BONDARYOF_8KB

8 Kbits (1 KBytes = 2^(10))

HAL_XSPI_BONDARYOF_16KB

16 Kbits (2 KBytes = 2^(11))

HAL_XSPI_BONDARYOF_32KB

32 Kbits (4 KBytes = 2^(12))

HAL_XSPI_BONDARYOF_64KB

64 Kbits (8 KBytes = 2^(13))

HAL_XSPI_BONDARYOF_128KB

128 Kbits (16 KBytes = 2^(14))

HAL_XSPI_BONDARYOF_256KB

256 Kbits (32 KBytes = 2^(15))

HAL_XSPI_BONDARYOF_512KB

512 Kbits (64 KBytes = 2^(16))

HAL_XSPI_BONDARYOF_1MB

1 Mbits (128 KBytes = 2^(17))

HAL_XSPI_BONDARYOF_2MB

2 Mbits (256 KBytes = 2^(18))

HAL_XSPI_BONDARYOF_4MB

4 Mbits (512 KBytes = 2^(19))

HAL_XSPI_BONDARYOF_8MB

8 Mbits (1 MBytes = 2^(20))

HAL_XSPI_BONDARYOF_16MB

16 Mbits (2 MBytes = 2^(21))

HAL_XSPI_BONDARYOF_32MB

32 Mbits (4 MBytes = 2^(22))

HAL_XSPI_BONDARYOF_64MB

64 Mbits (8 MBytes = 2^(23))

HAL_XSPI_BONDARYOF_128MB

128 Mbits (16 MBytes = 2^(24))

HAL_XSPI_BONDARYOF_256MB

256 Mbits (32 MBytes = 2^(25))

HAL_XSPI_BONDARYOF_512MB

512 Mbits (64 MBytes = 2^(26))

HAL_XSPI_BONDARYOF_1GB

1 Gbits (128 MBytes = 2^(27))

HAL_XSPI_BONDARYOF_2GB

2 Gbits (256 MBytes = 2^(28))

HAL_XSPI_BONDARYOF_4GB

4 Gbits (512 MBytes = 2^(29))

HAL_XSPI_BONDARYOF_8GB

8 Gbits (1 GBytes = 2^(30))

HAL_XSPI_BONDARYOF_16GB

16 Gbits (2 GBytes = 2^(31))

XSPI Clock Mode

HAL_XSPI_CLOCK_MODE_0

CLK must stay low while nCS is high

HAL_XSPI_CLOCK_MODE_3

CLK must stay high while nCS is high

XSPI Data DTR Mode

HAL_XSPI_DATA_DTR_DISABLE

DTR mode disabled for data phase

HAL_XSPI_DATA_DTR_ENABLE

DTR mode enabled for data phase

XSPI Data Mode

HAL_XSPI_DATA_NONE

No data

HAL_XSPI_DATA_1_LINE

Data on a single line

HAL_XSPI_DATA_2_LINES

Data on two lines

HAL_XSPI_DATA_4_LINES

Data on four lines

HAL_XSPI_DATA_8_LINES

Data on eight lines

HAL_XSPI_DATA_16_LINES

Data on sixteen lines valid for HSPI only

XSPI Delay Hold Quarter Cycle

HAL_XSPI_DHQC_DISABLE

No Delay

HAL_XSPI_DHQC_ENABLE

Delay Hold 1/4 cycle

XSPI Calibration Delay Type

HAL_XSPI_CAL_FULL_CYCLE_DELAY

Delay value equivalent to full memory-clock cycle

HAL_XSPI_CAL_FEEDBACK_CLK_DELAY

Delay value for the feedback clock when reading without DQS

HAL_XSPI_CAL_DATA_OUTPUT_DELAY

Delay value for output data in DDR mode for write operations

HAL_XSPI_CAL_DQS_INPUT_DELAY

Delay value for DQS input when sampling data for read operations

XSPI DQS Mode

HAL_XSPI_DQS_DISABLE

DQS disabled

HAL_XSPI_DQS_ENABLE

DQS enabled

XSPI Error Code

HAL_XSPI_ERROR_NONE

No error

HAL_XSPI_ERROR_TIMEOUT

Timeout error

HAL_XSPI_ERROR_TRANSFER

Transfer error

HAL_XSPI_ERROR_DMA

DMA transfer error

HAL_XSPI_ERROR_INVALID_PARAM

Invalid parameters error

HAL_XSPI_ERROR_INVALID_SEQUENCE

Sequence is incorrect

HAL_XSPI_ERROR_INVALID_CALLBACK

Invalid callback error

XSPI Exported Macros

HAL_XSPI_RESET_HANDLE_STATE

Description:

- Reset XSPI handle state.

Parameters:

- __HANDLE__: specifies the XSPI Handle.

Return value:

- None

HAL_XSPI_ENABLE

Description:

- Enable the XSPI peripheral.

Parameters:

- __HANDLE__: specifies the XSPI Handle.

Return value:

- None

HAL_XSPI_DISABLE

Description:

- Disable the XSPI peripheral.

Parameters:

- __HANDLE__: specifies the XSPI Handle.

Return value:

- None

HAL_XSPI_ENABLE_IT

Description:

- Enable the specified XSPI interrupt.

Parameters:

- __HANDLE__: specifies the XSPI Handle.
- __INTERRUPT__: specifies the XSPI interrupt source to enable. This parameter can be one of the following values:
  - HAL_XSPI_IT_TO: XSPI Timeout interrupt
  - HAL_XSPI_IT_SM: XSPI Status match interrupt
  - HAL_XSPI_IT_FT: XSPI FIFO threshold interrupt
  - HAL_XSPI_IT_TC: XSPI Transfer complete interrupt
  - HAL_XSPI_IT_TE: XSPI Transfer error interrupt

Return value:

- None

HAL_XSPI_DISABLE_IT

Description:

- Disable the specified XSPI interrupt.

Parameters:

- __HANDLE__: specifies the XSPI Handle.
- __INTERRUPT__: specifies the XSPI interrupt source to disable. This parameter can be one of the following values:
  - HAL_XSPI_IT_TO: XSPI Timeout interrupt
  - HAL_XSPI_IT_SM: XSPI Status match interrupt
  - HAL_XSPI_IT_FT: XSPI FIFO threshold interrupt
  - HAL_XSPI_IT_TC: XSPI Transfer complete interrupt
  - HAL_XSPI_IT_TE: XSPI Transfer error interrupt

Return value:

- None

HAL_XSPI_GET_IT_SOURCE

Description:

- Check whether the specified XSPI interrupt source is enabled or not.

Parameters:

- __HANDLE__: specifies the XSPI Handle.
- __INTERRUPT__: specifies the XSPI interrupt source to check. This parameter can be one of the following values:
  - HAL_XSPI_IT_TO: XSPI Timeout interrupt
  - HAL_XSPI_IT_SM: XSPI Status match interrupt
  - HAL_XSPI_IT_FT: XSPI FIFO threshold interrupt
  - HAL_XSPI_IT_TC: XSPI Transfer complete interrupt
  - HAL_XSPI_IT_TE: XSPI Transfer error interrupt

Return value:

- The: new state of __INTERRUPT__ (TRUE or FALSE).

HAL_XSPI_GET_FLAG

Description:

- Check whether the selected XSPI flag is set or not.

Parameters:

- __HANDLE__: specifies the XSPI Handle.
- __FLAG__: specifies the XSPI flag to check. This parameter can be one of the following values:
  - HAL_XSPI_FLAG_BUSY: XSPI Busy flag
  - HAL_XSPI_FLAG_TO: XSPI Timeout flag
  - HAL_XSPI_FLAG_SM: XSPI Status match flag
  - HAL_XSPI_FLAG_FT: XSPI FIFO threshold flag
  - HAL_XSPI_FLAG_TC: XSPI Transfer complete flag
  - HAL_XSPI_FLAG_TE: XSPI Transfer error flag

Return value:

- None

HAL_XSPI_CLEAR_FLAG

Description:

- Clears the specified XSPI's flag status.

Parameters:

- __HANDLE__: specifies the XSPI Handle.
- __FLAG__: specifies the XSPI clear register flag that needs to be set This parameter can be one of the following values:
  - HAL_XSPI_FLAG_TO: XSPI Timeout flag
  - HAL_XSPI_FLAG_SM: XSPI Status match flag
  - HAL_XSPI_FLAG_TC: XSPI Transfer complete flag
  - HAL_XSPI_FLAG_TE: XSPI Transfer error flag

Return value:

- None

XSPI Flags

HAL_XSPI_FLAG_BUSY

Busy flag: operation is ongoing

HAL_XSPI_FLAG_TO

Timeout flag: timeout occurs in memory-mapped mode

HAL_XSPI_FLAG_SM

Status match flag: received data matches in autopolling mode

HAL_XSPI_FLAG_FT

Fifo threshold flag: Fifo threshold reached or data left after read from memory is complete

HAL_XSPI_FLAG_TC

Transfer complete flag: programmed number of data have been transferred or the transfer has been aborted

HAL_XSPI_FLAG_TE

Transfer error flag: invalid address is being accessed

XSPI Free Running Clock

HAL_XSPI_FREERUNCLK_DISABLE

CLK is not free running

HAL_XSPI_FREERUNCLK_ENABLE

CLK is always provided (running)

XSPI Instruction DTR Mode

HAL_XSPI_INSTRUCTION_DTR_DISABLE

DTR mode disabled for instruction phase

HAL_XSPI_INSTRUCTION_DTR_ENABLE

DTR mode enabled for instruction phase

XSPI Instruction Mode

HAL_XSPI_INSTRUCTION_NONE

No instruction

HAL_XSPI_INSTRUCTION_1_LINE

Instruction on a single line

HAL_XSPI_INSTRUCTION_2_LINES

Instruction on two lines

HAL_XSPI_INSTRUCTION_4_LINES

Instruction on four lines

HAL_XSPI_INSTRUCTION_8_LINES

Instruction on eight lines

XSPI Instruction Width

HAL_XSPI_INSTRUCTION_8_BITS

8-bit instruction

HAL_XSPI_INSTRUCTION_16_BITS

16-bit instruction

HAL_XSPI_INSTRUCTION_24_BITS

24-bit instruction

HAL_XSPI_INSTRUCTION_32_BITS

32-bit instruction

XSPI Interrupts

HAL_XSPI_IT_TO

Interrupt on the timeout flag

HAL_XSPI_IT_SM

Interrupt on the status match flag

HAL_XSPI_IT_FT

Interrupt on the fifo threshold flag

HAL_XSPI_IT_TC

Interrupt on the transfer complete flag

HAL_XSPI_IT_TE

Interrupt on the transfer error flag

XSPI IO Select

HAL_XSPI_SELECT_IO_3_0

Data exchanged over IO[3:0]

HAL_XSPI_SELECT_IO_7_4

Data exchanged over IO[7:4]

HAL_XSPI_SELECT_IO_11_8

Data exchanged over IO[11:8]

HAL_XSPI_SELECT_IO_15_12

Data exchanged over IO[15:12]

HAL_XSPI_SELECT_IO_7_0

Data exchanged over IO[7:0]

HAL_XSPI_SELECT_IO_15_8

Data exchanged over IO[15:8]

XSPI IO Port

HAL_XSPIM_IOPORT_1

Port 1

HAL_XSPIM_IOPORT_2

Port 2

XSPI Hyperbus Latency Mode

HAL_XSPI_VARIABLE_LATENCY

Variable initial latency

HAL_XSPI_FIXED_LATENCY

Fixed latency

XSPI Match Mode

HAL_XSPI_MATCH_MODE_AND

AND match mode between unmasked bits

HAL_XSPI_MATCH_MODE_OR

OR match mode between unmasked bits

XSPI Calibration Maximal Value

HAL_XSPI_MAXCAL_NOT_REACHED

Memory-clock perido inside the range of DLL master

HAL_XSPI_MAXCAL_REACHED

Memory-clock period outside the range of DLL master (max delay values used)

XSPI Memory Extended

HAL_XSPI_CSSEL_SW

NCS1 and NCS2 are software controlled.

HAL_XSPI_CSSEL_HW

NCS1 and NCS2 are hardware controlled.

XSPI Memory Mode

HAL_XSPI_SINGLE_MEM

Dual-memory mode disabled

HAL_XSPI_DUAL_MEM

Dual mode enabled

XSPI Memory Select

HAL_XSPI_CSSEL_NCS1

The output of nCS is nCS1

HAL_XSPI_CSSEL_NCS2

The output of nCS is nCS2

XSPI Memory Size

HAL_XSPI_SIZE_16B

16 bits (2 Bytes = 2^(0+1))

HAL_XSPI_SIZE_32B

32 bits (4 Bytes = 2^(1+1))

HAL_XSPI_SIZE_64B

64 bits (8 Bytes = 2^(2+1))

HAL_XSPI_SIZE_128B

128 bits (16 Bytes = 2^(3+1))

HAL_XSPI_SIZE_256B

256 bits (32 Bytes = 2^(4+1))

HAL_XSPI_SIZE_512B

512 bits (64 Bytes = 2^(5+1))

HAL_XSPI_SIZE_1KB

1 Kbits (128 Bytes = 2^(6+1))

HAL_XSPI_SIZE_2KB

2 Kbits (256 Bytes = 2^(7+1))

HAL_XSPI_SIZE_4KB

4 Kbits (512 Bytes = 2^(8+1))

HAL_XSPI_SIZE_8KB

8 Kbits (1 KBytes = 2^(9+1))

HAL_XSPI_SIZE_16KB

16 Kbits (2 KBytes = 2^(10+1))

HAL_XSPI_SIZE_32KB

32 Kbits (4 KBytes = 2^(11+1))

HAL_XSPI_SIZE_64KB

64 Kbits (8 KBytes = 2^(12+1))

HAL_XSPI_SIZE_128KB

128 Kbits (16 KBytes = 2^(13+1))

HAL_XSPI_SIZE_256KB

256 Kbits (32 KBytes = 2^(14+1))

HAL_XSPI_SIZE_512KB

512 Kbits (64 KBytes = 2^(15+1))

HAL_XSPI_SIZE_1MB

1 Mbits (128 KBytes = 2^(16+1))

HAL_XSPI_SIZE_2MB

2 Mbits (256 KBytes = 2^(17+1))

HAL_XSPI_SIZE_4MB

4 Mbits (512 KBytes = 2^(18+1))

HAL_XSPI_SIZE_8MB

8 Mbits (1 MBytes = 2^(19+1))

HAL_XSPI_SIZE_16MB

16 Mbits (2 MBytes = 2^(20+1))

HAL_XSPI_SIZE_32MB

32 Mbits (4 MBytes = 2^(21+1))

HAL_XSPI_SIZE_64MB

64 Mbits (8 MBytes = 2^(22+1))

HAL_XSPI_SIZE_128MB

128 Mbits (16 MBytes = 2^(23+1))

HAL_XSPI_SIZE_256MB

256 Mbits (32 MBytes = 2^(24+1))

HAL_XSPI_SIZE_512MB

512 Mbits (64 MBytes = 2^(25+1))

HAL_XSPI_SIZE_1GB

1 Gbits (128 MBytes = 2^(26+1))

HAL_XSPI_SIZE_2GB

2 Gbits (256 MBytes = 2^(27+1))

HAL_XSPI_SIZE_4GB

4 Gbits (512 MBytes = 2^(28+1))

HAL_XSPI_SIZE_8GB

8 Gbits (1 GBytes = 2^(29+1))

HAL_XSPI_SIZE_16GB

16 Gbits (2 GBytes = 2^(30+1))

HAL_XSPI_SIZE_32GB

32 Gbits (4 GBytes = 2^(31+1))

XSPI Memory Type

HAL_XSPI_MEMTYPE_MICRON

Micron mode

HAL_XSPI_MEMTYPE_MACRONIX

Macronix mode

HAL_XSPI_MEMTYPE_APMEM

AP Memory mode

HAL_XSPI_MEMTYPE_MACRONIX_RAM

Macronix RAM mode

HAL_XSPI_MEMTYPE_HYPERBUS

Hyperbus mode

HAL_XSPI_MEMTYPE_APMEM_16BITS

AP Memory mode

XSPI No Prefetch AXI

HAL_XSPI_AXI_PREFETCH_ENABLE

Prefetch is enabled for AXI signaled transactions

HAL_XSPI_AXI_PREFETCH_DISABLE

Prefetch is disable for AXI signaled transactions

XSPI No Prefetch Data

HAL_XSPI_AUTOMATIC_PREFETCH_ENABLE

Automatic prefetch enabled

HAL_XSPI_AUTOMATIC_PREFETCH_DISABLE

Automatic prefetch disabled

XSPI Operation Type

HAL_XSPI_OPTYPE_COMMON_CFG

Common configuration (indirect or auto-polling mode)

HAL_XSPI_OPTYPE_READ_CFG

Read configuration (memory-mapped mode)

HAL_XSPI_OPTYPE_WRITE_CFG

Write configuration (memory-mapped mode)

HAL_XSPI_OPTYPE_WRAP_CFG

Wrap configuration (memory-mapped mode)

XSPI Sample Shifting

HAL_XSPI_SAMPLE_SHIFT_NONE

No shift

HAL_XSPI_SAMPLE_SHIFT_HALFCYCLE

1/2 cycle shift

XSPI State

HAL_XSPI_STATE_RESET

Initial state

HAL_XSPI_STATE_READY

Driver ready to be used

HAL_XSPI_STATE_HYPERBUS_INIT

Initialization done in hyperbus mode but timing configuration not done

HAL_XSPI_STATE_CMD_CFG

Command (regular or hyperbus) configured, ready for an action

HAL_XSPI_STATE_READ_CMD_CFG

Read command configuration done, not the write command configuration

HAL_XSPI_STATE_WRITE_CMD_CFG

Write command configuration done, not the read command configuration

HAL_XSPI_STATE_BUSY_CMD

Command without data on-going

HAL_XSPI_STATE_BUSY_TX

Indirect Tx on-going

HAL_XSPI_STATE_BUSY_RX

Indirect Rx on-going

HAL_XSPI_STATE_BUSY_AUTO_POLLING

Auto-polling on-going

HAL_XSPI_STATE_BUSY_MEM_MAPPED

Memory-mapped on-going

HAL_XSPI_STATE_ABORT

Abort on-going

HAL_XSPI_STATE_ERROR

Blocking error, driver should be re-initialized

XSPI Timeout Activation

HAL_XSPI_TIMEOUT_COUNTER_DISABLE

Timeout counter disabled, nCS remains active

HAL_XSPI_TIMEOUT_COUNTER_ENABLE

Timeout counter enabled, nCS released when timeout expires

XSPI Timeout definition

HAL_XSPI_TIMEOUT_DEFAULT_VALUE

XSPI Wrap-Size

HAL_XSPI_WRAP_NOT_SUPPORTED

wrapped reads are not supported by the memory

HAL_XSPI_WRAP_16_BYTES

external memory supports wrap size of 16 bytes

HAL_XSPI_WRAP_32_BYTES

external memory supports wrap size of 32 bytes

HAL_XSPI_WRAP_64_BYTES

external memory supports wrap size of 64 bytes

HAL_XSPI_WRAP_128_BYTES

external memory supports wrap size of 128 bytes

XSPI Hyperbus Write Zero Latency Activation

HAL_XSPI_LATENCY_ON_WRITE

Latency on write accesses

HAL_XSPI_NO_LATENCY_ON_WRITE

No latency on write accesses

85 LL ADC Generic Driver

85.1 ADC Firmware driver registers structures

85.1.1 LL_ADC_CommonInitTypeDef

LL_ADC_CommonInitTypeDef is defined in the stm32n6xx_ll_adc.h

Data Fields

- ***uint32_t Multimode***
- ***uint32_t MultiDataFormat***
- ***uint32_t MultiTwoSamplingDelay***

Field Documentation

- ***uint32_t LL_ADC_CommonInitTypeDef::Multimode***
  Set ADC multimode configuration to operate in independent mode or multimode (for devices with several ADC instances). This parameter can be a value of ***ADC_LL_EC_MULTI_MODE*** This feature can be modified afterwards using unitary function **LL_ADC_SetMultimode()**.
- ***uint32_t LL_ADC_CommonInitTypeDef::MultiDataFormat***
  Set ADC multimode conversion data format of ADC group regular: conversion data in data register of each ADC instance or ADC common instance. This parameter can be a value of ***ADC_LL_EC_MULTI_DATA_FORMAT*** This feature can be modified afterwards using unitary function **LL_ADC_SetMultiDataFormat()**.
- ***uint32_t LL_ADC_CommonInitTypeDef::MultiTwoSamplingDelay***
  Set ADC multimode delay between 2 sampling phases. Relevant only for multimode modes: interleaved based modes. This parameter can be a value of ***ADC_LL_EC_MULTI_TWOSMP_DELAY*** This feature can be modified afterwards using unitary function **LL_ADC_SetMultiTwoSamplingDelay()**.

85.1.2 LL_ADC_InitTypeDef

LL_ADC_InitTypeDef is defined in the stm32n6xx_ll_adc.h

Data Fields

- ***uint32_t Resolution***
- ***uint32_t LeftBitShift***
- ***uint32_t LowPowerMode***

Field Documentation

- ***uint32_t LL_ADC_InitTypeDef::Resolution***
  Set ADC resolution. This parameter can be a value of ***ADC_LL_EC_RESOLUTION*** This feature can be modified afterwards using unitary function **LL_ADC_SetResolution()**.
- ***uint32_t LL_ADC_InitTypeDef::LeftBitShift***
  Configures the left shifting applied to the final result with or without oversampling. This parameter can be a value of ***ADC_LL_EC_LEFT_BIT_SHIFT***.
- ***uint32_t LL_ADC_InitTypeDef::LowPowerMode***
  Set ADC low power mode. This parameter can be a value of ***ADC_LL_EC_LP_MODE*** This feature can be modified afterwards using unitary function **LL_ADC_SetLPModeAutoWait()**.

85.1.3 LL_ADC_REG_InitTypeDef

LL_ADC_REG_InitTypeDef is defined in the stm32n6xx_ll_adc.h

Data Fields

- ***uint32_t TriggerSource***
- ***uint32_t SequencerLength***
- ***uint32_t SequencerDiscont***
- ***uint32_t ContinuousMode***
- ***uint32_t DataTransferMode***
- ***uint32_t Overrun***

Field Documentation

- ***uint32_t LL_ADC_REG_InitTypeDef::TriggerSource***
  Set ADC group regular conversion trigger source: internal (SW start) or from external peripheral (timer event, external interrupt line). This parameter can be a value of ***ADC_LL_EC_REG_TRIGGER_SOURCE***
  Note:
  – On this STM32 series, setting trigger source to external trigger also set trigger polarity to rising edge(default setting for compatibility with some ADC on other STM32 series having this setting set by HW default value). In case of need to modify trigger edge, use function **LL_ADC_REG_SetTriggerEdge()**. This feature can be modified afterwards using unitary function **LL_ADC_REG_SetTriggerSource()**.
- ***uint32_t LL_ADC_REG_InitTypeDef::SequencerLength***
  Set ADC group regular sequencer length. This parameter can be a value of ***ADC_LL_EC_REG_SEQ_SCAN_LENGTH*** This feature can be modified afterwards using unitary function **LL_ADC_REG_SetSequencerLength()**.
- ***uint32_t LL_ADC_REG_InitTypeDef::SequencerDiscont***
  Set ADC group regular sequencer discontinuous mode: sequence subdivided and scan conversions interrupted every selected number of ranks. This parameter can be a value of ***ADC_LL_EC_REG_SEQ_DISCONT_MODE***
  Note:
  – This parameter has an effect only if group regular sequencer is enabled (scan length of 2 ranks or more). This feature can be modified afterwards using unitary function **LL_ADC_REG_SetSequencerDiscont()**.
- ***uint32_t LL_ADC_REG_InitTypeDef::ContinuousMode***
  Set ADC continuous conversion mode on ADC group regular, whether ADC conversions are performed in single mode (one conversion per trigger) or in continuous mode (after the first trigger, following conversions launched successively automatically). This parameter can be a value of ***ADC_LL_EC_REG_CONTINUOUS_MODE*** Note: It is not possible to enable both ADC group regular continuous mode and discontinuous mode. This feature can be modified afterwards using unitary function **LL_ADC_REG_SetContinuousMode()**.
- ***uint32_t LL_ADC_REG_InitTypeDef::DataTransferMode***
  Set ADC group regular conversion data transfer: no transfer, transfer by DMA or other peripherals. This parameter can be a value of ***ADC_LL_EC_REG_DATA_TRANSFER*** This feature can be modified afterwards using unitary function **LL_ADC_REG_SetDataTransferMode()**.
- ***uint32_t LL_ADC_REG_InitTypeDef::Overrun***
  Set ADC group regular behavior in case of overrun: data preserved or overwritten. This parameter can be a value of ***ADC_LL_EC_REG_OVR_DATA_BEHAVIOR*** This feature can be modified afterwards using unitary function **LL_ADC_REG_SetOverrun()**.

85.1.4 LL_ADC_INJ_InitTypeDef

LL_ADC_INJ_InitTypeDef is defined in the stm32n6xx_ll_adc.h

Data Fields

- ***uint32_t TriggerSource***
- ***uint32_t SequencerLength***
- ***uint32_t SequencerDiscont***
- ***uint32_t TrigAuto***

Field Documentation

- ***uint32_t LL_ADC_INJ_InitTypeDef::TriggerSource***
  Set ADC group injected conversion trigger source: internal (SW start) or from external peripheral (timer event, external interrupt line). This parameter can be a value of ***ADC_LL_EC_INJ_TRIGGER_SOURCE***
  Note:
  – On this STM32 series, setting trigger source to external trigger also set trigger polarity to rising edge (default setting for compatibility with some ADC on other STM32 series having this setting set by HW default value). In case of need to modify trigger edge, use function **LL_ADC_INJ_SetTriggerEdge()**. This feature can be modified afterwards using unitary function **LL_ADC_INJ_SetTriggerSource()**.

- ***uint32_t LL_ADC_INJ_InitTypeDef::SequencerLength***
  Set ADC group injected sequencer length. This parameter can be a value of ***ADC_LL_EC_INJ_SEQ_SCAN_LENGTH*** This feature can be modified afterwards using unitary function **LL_ADC_INJ_SetSequencerLength()**.
- ***uint32_t LL_ADC_INJ_InitTypeDef::SequencerDiscont***
  Set ADC group injected sequencer discontinuous mode: sequence subdivided and scan conversions interrupted every selected number of ranks. This parameter can be a value of ***ADC_LL_EC_INJ_SEQ_DISCONT_MODE***
  Note:
  – This parameter has an effect only if group injected sequencer is enabled (scan length of 2 ranks or more). This feature can be modified afterwards using unitary function **LL_ADC_INJ_SetSequencerDiscont()**.
- ***uint32_t LL_ADC_INJ_InitTypeDef::TrigAuto***
  Set ADC group injected conversion trigger: independent or from ADC group regular. This parameter can be a value of ***ADC_LL_EC_INJ_TRIG_AUTO*** Note: This parameter must be set to set to independent trigger if injected trigger source is set to an external trigger. This feature can be modified afterwards using unitary function **LL_ADC_INJ_SetTrigAuto()**.

85.2 ADC Firmware driver API description

The following section lists the various functions of the ADC library.

85.2.1 Detailed description of functions

LL_ADC_DMA_GetRegAddr

Function name

__STATIC_INLINE uint32_t LL_ADC_DMA_GetRegAddr (const ADC_TypeDef * ADCx, uint32_t Register)

Function description

Function to help to configure DMA transfer from ADC: retrieve the ADC register address from ADC instance and a list of ADC registers intended to be used (most commonly) with DMA transfer.

Parameters

- **ADCx:** ADC instance
- **Register:** This parameter can be one of the following values:
  – LL_ADC_DMA_REG_REGULAR_DATA
  – LL_ADC_DMA_REG_REGULAR_MULTI_NO_PACKING (1)
  – LL_ADC_DMA_REG_REGULAR_MULTI_PACKING (1)

  (1) Available on devices with several ADC instances.

Return values

- **ADC:** register address

Notes

- These ADC registers are data registers: when ADC conversion data is available in ADC data registers, ADC generates a DMA transfer request.
- This macro is intended to be used with LL DMA driver, refer to function "LL_DMA_ConfigAddresses()". Example: LL_DMA_ConfigAddresses(DMA1, LL_DMA_CHANNEL_1, LL_ADC_DMA_GetRegAddr(ADC1, LL_ADC_DMA_REG_REGULAR_DATA), (uint32_t)&< array or variable >, LL_DMA_DIRECTION_PERIPH_TO_MEMORY);
- For devices with several ADC: in multimode, some devices use a different data register outside of ADC instance scope (common data register). This macro manages this register difference, only ADC instance has to be set as parameter.

Reference Manual to LL API cross reference:

- DR RDATA LL_ADC_DMA_GetRegAddr
- CDR RDATA_MST LL_ADC_DMA_GetRegAddr
- CDR RDATA_SLV LL_ADC_DMA_GetRegAddr

LL_ADC_SetCommonPathInternalCh

Function name

__STATIC_INLINE void LL_ADC_SetCommonPathInternalCh (ADC_Common_TypeDef * ADCxy_COMMON, uint32_t PathInternal)

Function description

Set parameter common to several ADC: measurement path to internal channels (VrefInt, temperature sensor, ...).

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())
- **PathInternal:** This parameter can be a combination of the following values:
  - LL_ADC_PATH_INTERNAL_NONE
  - LL_ADC_PATH_INTERNAL_VREFINT
  - LL_ADC_PATH_INTERNAL_VBAT

Return values

- **None:**

Notes

- One or several values can be selected. Example: (LL_ADC_PATH_INTERNAL_VREFINT | LL_ADC_PATH_INTERNAL_TEMPSENSOR) The values not selected are removed from configuration.
- Stabilization time of measurement path to internal channel: After enabling internal paths, before starting ADC conversion, a delay is required for internal voltage reference and temperature sensor stabilization time. Refer to device datasheet. Refer to literal LL_ADC_DELAY_VREFINT_STAB_US.
- ADC internal channel sampling time constraint: For ADC conversion of internal channels, a sampling time minimum value is required. Refer to device datasheet.
- On this STM32 series, setting of this feature is conditioned to ADC state: All ADC instances of the ADC common group must be disabled. This check can be done with function LL_ADC_IsEnabled() for each ADC instance or by using helper macro helper macro __LL_ADC_IS_ENABLED_ALL_COMMON_INSTANCE().

Reference Manual to LL API cross reference:

- CCR VREFEN LL_ADC_SetCommonPathInternalCh
- CCR VBATEN LL_ADC_SetCommonPathInternalCh

LL_ADC_SetCommonPathInternalChAdd

Function name

__STATIC_INLINE void LL_ADC_SetCommonPathInternalChAdd (ADC_Common_TypeDef * ADCxy_COMMON, uint32_t PathInternal)

Function description

Set parameter common to several ADC: measurement path to internal channels (VrefInt, temperature sensor, ...).

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())
- **PathInternal:** This parameter can be a combination of the following values:
  - LL_ADC_PATH_INTERNAL_NONE
  - LL_ADC_PATH_INTERNAL_VREFINT
  - LL_ADC_PATH_INTERNAL_VBAT

Return values

- **None:**

Notes

- One or several values can be selected. Example: (LL_ADC_PATH_INTERNAL_VREFINT | LL_ADC_PATH_INTERNAL_TEMPSENSOR)
- Stabilization time of measurement path to internal channel: After enabling internal paths, before starting ADC conversion, a delay is required for internal voltage reference and temperature sensor stabilization time. Refer to device datasheet. Refer to literal LL_ADC_DELAY_VREFINT_STAB_US.
- ADC internal channel sampling time constraint: For ADC conversion of internal channels, a sampling time minimum value is required. Refer to device datasheet.
- On this STM32 series, setting of this feature is conditioned to ADC state: All ADC instances of the ADC common group must be disabled. This check can be done with function LL_ADC_IsEnabled() for each ADC instance or by using helper macro helper macro __LL_ADC_IS_ENABLED_ALL_COMMON_INSTANCE().

Reference Manual to LL API cross reference:

- CCR VREFEN LL_ADC_SetCommonPathInternalChAdd
- CCR VBATEN LL_ADC_SetCommonPathInternalChAdd

LL_ADC_SetCommonPathInternalChRem

Function name

__STATIC_INLINE void LL_ADC_SetCommonPathInternalChRem (ADC_Common_TypeDef * ADCxy_COMMON, uint32_t PathInternal)

Function description

Set parameter common to several ADC: measurement path to internal channels (VrefInt, temperature sensor, ...).

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())
- **PathInternal:** This parameter can be a combination of the following values:
  - LL_ADC_PATH_INTERNAL_NONE
  - LL_ADC_PATH_INTERNAL_VREFINT
  - LL_ADC_PATH_INTERNAL_VBAT

Return values

- **None:**

Notes

- One or several values can be selected. Example: (LL_ADC_PATH_INTERNAL_VREFINT | LL_ADC_PATH_INTERNAL_TEMPSENSOR)
- On this STM32 series, setting of this feature is conditioned to ADC state: All ADC instances of the ADC common group must be disabled. This check can be done with function LL_ADC_IsEnabled() for each ADC instance or by using helper macro helper macro __LL_ADC_IS_ENABLED_ALL_COMMON_INSTANCE().

Reference Manual to LL API cross reference:

- CCR VREFEN LL_ADC_SetCommonPathInternalChRem
- CCR VBATEN LL_ADC_SetCommonPathInternalChRem

LL_ADC_GetCommonPathInternalCh

Function name

__STATIC_INLINE uint32_t LL_ADC_GetCommonPathInternalCh (const ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get parameter common to several ADC: measurement path to internal channels (VrefInt, temperature sensor, ...).

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **Returned:** value can be a combination of the following values:
  - LL_ADC_PATH_INTERNAL_NONE
  - LL_ADC_PATH_INTERNAL_VREFINT
  - LL_ADC_PATH_INTERNAL_VBAT

Notes

- One or several values can be selected. Example: (LL_ADC_PATH_INTERNAL_VREFINT | LL_ADC_PATH_INTERNAL_TEMPSENSOR)

Reference Manual to LL API cross reference:

- CCR VREFEN LL_ADC_GetCommonPathInternalCh
- CCR VBATEN LL_ADC_GetCommonPathInternalCh

LL_ADC_SetPathInternalCh

Function name

__STATIC_INLINE void LL_ADC_SetPathInternalCh (ADC_TypeDef * ADCx, uint32_t PathInternal)

Function description

Set parameter on ADC instance scope: measurement path to internal channels (Vcore, Vcpu...).

Parameters

- **ADCx:** ADC instance
- **PathInternal:** This parameter can be a combination of the following values:
  - LL_ADC_PATH_INTERNAL_NONE
  - LL_ADC_PATH_INTERNAL_VDDCORE

Return values

- **None:**

Notes

- One or several values can be selected. Example: (LL_ADC_PATH_INTERNAL_VDDCORE | LL_ADC_PATH_INTERNAL_...) The values not selected are removed from configuration.
- Stabilization time of measurement path to internal channel: After enabling internal paths, before starting ADC conversion, a delay may be required required for analog stabilization. Refer to device datasheet.
- ADC internal channel sampling time constraint: For ADC conversion of internal channels, a sampling time minimum value is required. Refer to device datasheet.

Reference Manual to LL API cross reference:

- OR OP2 LL_ADC_SetPathInternalCh

LL_ADC_SetPathInternalChAdd

Function name

__STATIC_INLINE void LL_ADC_SetPathInternalChAdd (ADC_TypeDef * ADCx, uint32_t PathInternal)

Function description

Set parameter on ADC instance scope: measurement path to internal channels (Vcore, Vcpu...).

Parameters

- **ADCx:** ADC instance
- **PathInternal:** This parameter can be a combination of the following values:
  - LL_ADC_PATH_INTERNAL_NONE
  - LL_ADC_PATH_INTERNAL_VDDCORE

Return values

- **None:**

Notes

- One or several values can be selected. Example: (LL_ADC_PATH_INTERNAL_VDDCORE | LL_ADC_PATH_INTERNAL_...) The values not selected are removed from configuration.
- Stabilization time of measurement path to internal channel: After enabling internal paths, before starting ADC conversion, a delay may be required required for analog stabilization. Refer to device datasheet.
- ADC internal channel sampling time constraint: For ADC conversion of internal channels, a sampling time minimum value is required. Refer to device datasheet.

Reference Manual to LL API cross reference:

- OR OP2 LL_ADC_SetPathInternalChAdd

LL_ADC_SetPathInternalChRem

Function name

__STATIC_INLINE void LL_ADC_SetPathInternalChRem (ADC_TypeDef * ADCx, uint32_t PathInternal)

Function description

Set parameter on ADC instance scope: measurement path to internal channels (Vcore, Vcpu...).

Parameters

- **ADCx:** ADC instance
- **PathInternal:** This parameter can be a combination of the following values:
  - LL_ADC_PATH_INTERNAL_NONE
  - LL_ADC_PATH_INTERNAL_VDDCORE

Return values

- **None:**

Notes

- One or several values can be selected. Example: (LL_ADC_PATH_INTERNAL_VDDCORE | LL_ADC_PATH_INTERNAL_...) The values not selected are removed from configuration.
- Stabilization time of measurement path to internal channel: After enabling internal paths, before starting ADC conversion, a delay may be required required for analog stabilization. Refer to device datasheet.
- ADC internal channel sampling time constraint: For ADC conversion of internal channels, a sampling time minimum value is required. Refer to device datasheet.

Reference Manual to LL API cross reference:

- OR OP2 LL_ADC_SetPathInternalChRem

LL_ADC_GetPathInternalCh

Function name

__STATIC_INLINE uint32_t LL_ADC_GetPathInternalCh (const ADC_TypeDef * ADCx)

Function description

Get parameter on ADC instance scope: measurement path to internal channels (Vcore, Vcpu...).

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_PATH_INTERNAL_NONE
  - LL_ADC_PATH_INTERNAL_VDDCORE

Notes

- One or several values can be selected. Example: (LL_ADC_PATH_INTERNAL_VDDCORE | LL_ADC_PATH_INTERNAL_...)

Reference Manual to LL API cross reference:

- OR OP2 LL_ADC_GetPathInternalCh

LL_ADC_SetCalibrationFactor

Function name

__STATIC_INLINE void LL_ADC_SetCalibrationFactor (ADC_TypeDef * ADCx, uint32_t SingleDiff, uint32_t CalibrationFactor)

Function description

Set ADC calibration factor in the mode single-ended or differential (for devices with differential mode available).

Parameters

- **ADCx:** ADC instance
- **SingleDiff:** This parameter can be one of the following values:
  - LL_ADC_SINGLE_ENDED
  - LL_ADC_DIFFERENTIAL_ENDED
  - LL_ADC_BOTH_SINGLE_DIFF_ENDED
- **CalibrationFactor:** Value between Min_Data=0x00 and Max_Data=0x7F

Return values

- **None:**

Notes

- This function is intended to set calibration parameters without having to perform a new calibration using LL_ADC_StartCalibration().
- In case of setting calibration factors of both modes single ended and differential (parameter LL_ADC_BOTH_SINGLE_DIFF_ENDED): both calibration factors must be concatenated. To perform this processing, use helper macro __LL_ADC_CALIB_FACTOR_SINGLE_DIFF().
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be enabled, without calibration on going, without conversion on going on group regular and group injected.

Reference Manual to LL API cross reference:

- CALFACT CALFACT_S LL_ADC_SetCalibrationFactor
- CALFACT CALFACT_D LL_ADC_SetCalibrationFactor

LL_ADC_GetCalibrationFactor

Function name

__STATIC_INLINE uint32_t LL_ADC_GetCalibrationFactor (const ADC_TypeDef * ADCx, uint32_t SingleDiff)

Function description

Get ADC calibration factor in the mode single-ended or differential (for devices with differential mode available).

Parameters

- **ADCx:** ADC instance
- **SingleDiff:** This parameter can be one of the following values:
  - LL_ADC_SINGLE_ENDED
  - LL_ADC_DIFFERENTIAL_ENDED

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x1FF

Notes

- Calibration factors are set by hardware after performing a calibration run using function LL_ADC_StartCalibration().
- For devices with differential mode available: Calibration of offset is specific to each of single-ended and differential modes

Reference Manual to LL API cross reference:

- CALFACT CALFACT_S LL_ADC_GetCalibrationFactor
- CALFACT CALFACT_D LL_ADC_GetCalibrationFactor

LL_ADC_SetResolution

Function name

__STATIC_INLINE void LL_ADC_SetResolution (ADC_TypeDef * ADCx, uint32_t Resolution)

Function description

Set ADC resolution.

Parameters

- **ADCx:** ADC instance
- **Resolution:** This parameter can be one of the following values:
  - LL_ADC_RESOLUTION_12B
  - LL_ADC_RESOLUTION_10B
  - LL_ADC_RESOLUTION_8B
  - LL_ADC_RESOLUTION_6B

Return values

- **None:**

Notes

- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- CFGR1 RES LL_ADC_SetResolution

LL_ADC_GetResolution

Function name

__STATIC_INLINE uint32_t LL_ADC_GetResolution (const ADC_TypeDef * ADCx)

Function description

Get ADC resolution.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_RESOLUTION_12B
  - LL_ADC_RESOLUTION_10B
  - LL_ADC_RESOLUTION_8B
  - LL_ADC_RESOLUTION_6B

Reference Manual to LL API cross reference:

- CFGR1 RES LL_ADC_GetResolution

LL_ADC_SetLPModeAutoWait

Function name

__STATIC_INLINE void LL_ADC_SetLPModeAutoWait (ADC_TypeDef * ADCx, uint32_t LowPowerMode)

Function description

Set ADC low power mode.

Parameters

- **ADCx:** ADC instance
- **LowPowerMode:** This parameter can be one of the following values:
  - LL_ADC_LP_AUTOWAIT_DISABLE
  - LL_ADC_LP_AUTOWAIT_ENABLE

Return values

- **None:**

Notes

- Description of ADC low power modes: ADC low power mode "auto wait": Dynamic low power mode, ADC conversions occurrences are limited to the minimum necessary in order to reduce power consumption. New ADC conversion starts only when the previous unitary conversion data (for ADC group regular) or previous sequence conversions data (for ADC group injected) has been retrieved by user software. In the meantime, ADC remains idle: does not performs any other conversion. This mode allows to automatically adapt the ADC conversions triggers to the speed of the software that reads the data. Moreover, this avoids risk of overrun for low frequency applications. How to use this low power mode: It is not recommended to use with interruption or DMA since these modes have to clear immediately the EOC flag (by CPU to free the IRQ pending event or by DMA). Auto wait will work but fort a very short time, discarding its intended benefit (except specific case of high load of CPU or DMA transfers which can justify usage of auto wait).Do use with polling: 1. Start conversion, 2. Later on, when conversion data is needed: poll for end of conversion to ensure that conversion is completed and retrieve ADC conversion data. This will trig another ADC conversion start. ADC low power mode "auto power-off" (feature available on this device if parameter LL_ADC_LP_AUTOPOWEROFF is available): the ADC automatically powers-off after a conversion and automatically wakes up when a new conversion is triggered (with startup time between trigger and start of sampling). This feature can be combined with low power mode "auto wait".
- With ADC low power mode "auto wait", the ADC conversion data read is corresponding to previous ADC conversion start, independently of delay during which ADC was idle. Therefore, the ADC conversion data may be outdated: does not correspond to the current voltage level on the selected ADC channel.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- CFGR1 AUTDLY LL_ADC_SetLPModeAutoWait

LL_ADC_GetLPModeAutoWait

Function name

__STATIC_INLINE uint32_t LL_ADC_GetLPModeAutoWait (const ADC_TypeDef * ADCx)

Function description

Get ADC low power mode:

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_LP_AUTOWAIT_DISABLE
  - LL_ADC_LP_AUTOWAIT_ENABLE

Notes

- Description of ADC low power modes: ADC low power mode "auto wait": Dynamic low power mode, ADC conversions occurrences are limited to the minimum necessary in order to reduce power consumption. New ADC conversion starts only when the previous unitary conversion data (for ADC group regular) or previous sequence conversions data (for ADC group injected) has been retrieved by user software. In the meantime, ADC remains idle: does not performs any other conversion. This mode allows to automatically adapt the ADC conversions triggers to the speed of the software that reads the data. Moreover, this avoids risk of overrun for low frequency applications. How to use this low power mode: It is not recommended to use with interruption or DMA since these modes have to clear immediately the EOC flag (by CPU to free the IRQ pending event or by DMA). Auto wait will work but fort a very short time, discarding its intended benefit (except specific case of high load of CPU or DMA transfers which can justify usage of auto wait).Do use with polling: 1. Start conversion, 2. Later on, when conversion data is needed: poll for end of conversion to ensure that conversion is completed and retrieve ADC conversion data. This will trig another ADC conversion start. ADC low power mode "auto power-off" (feature available on this device if parameter LL_ADC_LP_AUTOPOWEROFF is available): the ADC automatically powers-off after a conversion and automatically wakes up when a new conversion is triggered (with startup time between trigger and start of sampling). This feature can be combined with low power mode "auto wait".
- With ADC low power mode "auto wait", the ADC conversion data read is corresponding to previous ADC conversion start, independently of delay during which ADC was idle. Therefore, the ADC conversion data may be outdated: does not correspond to the current voltage level on the selected ADC channel.

Reference Manual to LL API cross reference:

- CFGR1 AUTDLY LL_ADC_GetLPModeAutoWait

LL_ADC_SetOffsetChannel

Function name

__STATIC_INLINE void LL_ADC_SetOffsetChannel (ADC_TypeDef * ADCx, uint32_t Offsety, uint32_t Channel)

Function description

Set for the ADC selected offset number 1, 2, 3 or 4: Channel to which the offset programmed will be applied (independently of channel mapped on ADC group regular or group injected)

Parameters

- **ADCx:** ADC instance
- **Offsety:** This parameter can be one of the following values:
  - LL_ADC_OFFSET_1
  - LL_ADC_OFFSET_2
  - LL_ADC_OFFSET_3
  - LL_ADC_OFFSET_4
- **Channel:** This parameter can be one of the following values:
  - LL_ADC_CHANNEL_0
  - LL_ADC_CHANNEL_1
  - LL_ADC_CHANNEL_2
  - LL_ADC_CHANNEL_3
  - LL_ADC_CHANNEL_4
  - LL_ADC_CHANNEL_5
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_19
  - LL_ADC_CHANNEL_VREFINT (1)
  - LL_ADC_CHANNEL_VBAT (2)
  - LL_ADC_CHANNEL_VDDCORE (2)

  (1) On this STM32 series, parameter available only on ADC instance: ADC1.
- (2) On this STM32 series, parameter available only on ADC instance: ADC2.
-

Return values

- **None:**

Notes

- If a channel is mapped on several offsets numbers, only the offset with the lowest value is considered for the subtraction.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- OFR1 OFFSET_CH LL_ADC_SetOffsetChannel
- OFR2 OFFSET_CH LL_ADC_SetOffsetChannel
- OFR3 OFFSET_CH LL_ADC_SetOffsetChannel
- OFR4 OFFSET_CH LL_ADC_SetOffsetChannel

LL_ADC_GetOffsetChannel

Function name

__STATIC_INLINE uint32_t LL_ADC_GetOffsetChannel (const ADC_TypeDef * ADCx, uint32_t Offsety)

Function description

Get for the ADC selected offset instance 1, 2, 3 or 4: Channel to which the offset programmed will be applied (independently of channel mapped on ADC group regular or group injected)

Parameters

- **ADCx:** ADC instance
- **Offsety:** This parameter can be one of the following values:
  - LL_ADC_OFFSET_1
  - LL_ADC_OFFSET_2
  - LL_ADC_OFFSET_3
  - LL_ADC_OFFSET_4

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_CHANNEL_0
  - LL_ADC_CHANNEL_1
  - LL_ADC_CHANNEL_2
  - LL_ADC_CHANNEL_3
  - LL_ADC_CHANNEL_4
  - LL_ADC_CHANNEL_5
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_19
  - LL_ADC_CHANNEL_VREFINT (1)(3)
  - LL_ADC_CHANNEL_VBAT (2)(3)
  - LL_ADC_CHANNEL_VDDCORE (2)(3)

  (1) On this STM32 series, parameter available only on ADC instance: ADC1.
- (2) On this STM32 series, parameter available only on ADC instance: ADC2.
- (3) For ADC channel read back from ADC register, comparison with internal channel parameter to be done using helper macro __LL_ADC_CHANNEL_INTERNAL_TO_EXTERNAL().

Notes

- Usage of the returned channel number: To reinject this channel into another function LL_ADC_xxx: the returned channel number is only partly formatted on definition of literals LL_ADC_CHANNEL_x. Therefore, it has to be compared with parts of literals LL_ADC_CHANNEL_x or using helper macro __LL_ADC_CHANNEL_TO_DECIMAL_NB(). Then the selected literal LL_ADC_CHANNEL_x can be used as parameter for another function.To get the channel number in decimal format: process the returned value with the helper macro __LL_ADC_CHANNEL_TO_DECIMAL_NB().

Reference Manual to LL API cross reference:

- OFR1 OFFSET_CH LL_ADC_SetOffsetChannel
- OFR2 OFFSET_CH LL_ADC_SetOffsetChannel
- OFR3 OFFSET_CH LL_ADC_SetOffsetChannel
- OFR4 OFFSET_CH LL_ADC_SetOffsetChannel

LL_ADC_SetOffsetLevel

Function name

__STATIC_INLINE void LL_ADC_SetOffsetLevel (ADC_TypeDef * ADCx, uint32_t Offsety, uint32_t OffsetLevel)

Function description

Set for the ADC selected offset number 1, 2, 3 or 4: Offset level (offset to be subtracted from the raw converted data).

Parameters

- **ADCx:** ADC instance
- **Offsety:** This parameter can be one of the following values:
  - LL_ADC_OFFSET_1
  - LL_ADC_OFFSET_2
  - LL_ADC_OFFSET_3
  - LL_ADC_OFFSET_4
- **OffsetLevel:** Value between Min_Data=0x000 and Max_Data=0x00FFFFFF

Return values

- **None:**

Notes

- Caution: Offset format is dependent to ADC resolution: offset has to be left-aligned on bit 11, the LSB (right bits) are set to 0.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- OFR1 OFFSET1 LL_ADC_SetOffsetLevel
- OFR2 OFFSET2 LL_ADC_SetOffsetLevel
- OFR3 OFFSET3 LL_ADC_SetOffsetLevel
- OFR4 OFFSET4 LL_ADC_SetOffsetLevel

LL_ADC_GetOffsetLevel

Function name

__STATIC_INLINE uint32_t LL_ADC_GetOffsetLevel (const ADC_TypeDef * ADCx, uint32_t Offsety)

Function description

Get for the ADC selected offset instance 1, 2, 3 or 4: Offset level (offset to be subtracted from the raw converted data).

Parameters

- **ADCx:** ADC instance
- **Offsety:** This parameter can be one of the following values:
  - LL_ADC_OFFSET_1
  - LL_ADC_OFFSET_2
  - LL_ADC_OFFSET_3
  - LL_ADC_OFFSET_4

Return values

- **Value:** between Min_Data=0x000 and Max_Data=0x00FFFFFF

Notes

- Caution: Offset format is dependent to ADC resolution: offset has to be left-aligned on bit 11, the LSB (right bits) are set to 0.

Reference Manual to LL API cross reference:

- OFR1 OFFSET1 LL_ADC_GetOffsetLevel
- OFR2 OFFSET2 LL_ADC_GetOffsetLevel
- OFR3 OFFSET3 LL_ADC_GetOffsetLevel
- OFR4 OFFSET4 LL_ADC_GetOffsetLevel

LL_ADC_SetOffsetSign

Function name

__STATIC_INLINE void LL_ADC_SetOffsetSign (ADC_TypeDef * ADCx, uint32_t Offsety, uint32_t OffsetSign)

Function description

Set for the ADC selected offset instance 1, 2, 3 or 4: force offset state disable or enable without modifying offset channel or offset value.

Parameters

- **ADCx:** ADC instance
- **Offsety:** This parameter can be one of the following values:
  - LL_ADC_OFFSET_1
  - LL_ADC_OFFSET_2
  - LL_ADC_OFFSET_3
  - LL_ADC_OFFSET_4
- **OffsetSign:** This parameter can be one of the following values:
  - LL_ADC_OFFSET_SIGN_NEGATIVE
  - LL_ADC_OFFSET_SIGN_POSITIVE

Return values

- **None:**

Notes

- This function should be needed only in case of offset to be enabled-disabled dynamically, and should not be needed in other cases: function LL_ADC_SetOffset() automatically enables the offset.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- OFCFGR1 OFFSETPOS LL_ADC_SetOffsetSign
- OFCFGR2 OFFSETPOS LL_ADC_SetOffsetSign
- OFCFGR3 OFFSETPOS LL_ADC_SetOffsetSign
- OFCFGR4 OFFSETPOS LL_ADC_SetOffsetSign

LL_ADC_GetOffsetSign

Function name

__STATIC_INLINE uint32_t LL_ADC_GetOffsetSign (const ADC_TypeDef * ADCx, uint32_t Offsety)

Function description

Get for the ADC selected offset instance 1, 2, 3 or 4: offset sign if positive or negative.

Parameters

- **ADCx:** ADC instance
- **Offsety:** This parameter can be one of the following values:
  - LL_ADC_OFFSET_1
  - LL_ADC_OFFSET_2
  - LL_ADC_OFFSET_3
  - LL_ADC_OFFSET_4

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_OFFSET_SIGN_NEGATIVE
  - LL_ADC_OFFSET_SIGN_POSITIVE

Reference Manual to LL API cross reference:

- OFCFGR1 OFFSETPOS LL_ADC_GetOffsetSign
- OFCFGR2 OFFSETPOS LL_ADC_GetOffsetSign
- OFCFGR3 OFFSETPOS LL_ADC_GetOffsetSign
- OFCFGR4 OFFSETPOS LL_ADC_GetOffsetSign

LL_ADC_SetOffsetSignedSaturation

Function name

__STATIC_INLINE void LL_ADC_SetOffsetSignedSaturation (ADC_TypeDef * ADCx, uint32_t Offsety, uint32_t OffsetSignedSaturation)

Function description

Set Signed saturation for the ADC selected offset instance 1, 2, 3 or 4: signed offset saturation if enabled or disabled.

Parameters

- **ADCx:** ADC instance
- **Offsety:** This parameter can be one of the following values:
  - LL_ADC_OFFSET_1
  - LL_ADC_OFFSET_2
  - LL_ADC_OFFSET_3
  - LL_ADC_OFFSET_4
- **OffsetSignedSaturation:** This parameter can be one of the following values:
  - LL_ADC_OFFSET_SIGNED_SAT_ENABLE
  - LL_ADC_OFFSET_SIGNED_SAT_DISABLE

Return values

- **None:**

Reference Manual to LL API cross reference:

- OFCFGR1 SSAT LL_ADC_SetOffsetSignedSaturation
- OFCFGR2 SSAT LL_ADC_SetOffsetSignedSaturation
- OFCFGR3 SSAT LL_ADC_SetOffsetSignedSaturation
- OFCFGR4 SSAT LL_ADC_SetOffsetSignedSaturation

LL_ADC_GetOffsetSignedSaturation

Function name

__STATIC_INLINE uint32_t LL_ADC_GetOffsetSignedSaturation (const ADC_TypeDef * ADCx, uint32_t Offsety)

Function description

Get Signed saturation for the ADC selected offset number 1, 2, 3 or 4: signed offset saturation if enabled or disabled.

Parameters

- **ADCx:** ADC instance
- **Offsety:** This parameter can be one of the following values:
  - LL_ADC_OFFSET_1
  - LL_ADC_OFFSET_2
  - LL_ADC_OFFSET_3
  - LL_ADC_OFFSET_4

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_OFFSET_SIGNED_SAT_ENABLE
  - LL_ADC_OFFSET_SIGNED_SAT_DISABLE

Reference Manual to LL API cross reference:

- OFCFGR1 SSAT LL_ADC_GetOffsetSignedSaturation
- OFCFGR2 SSAT LL_ADC_GetOffsetSignedSaturation
- OFCFGR3 SSAT LL_ADC_GetOffsetSignedSaturation
- OFCFGR4 SSAT LL_ADC_GetOffsetSignedSaturation

LL_ADC_SetOffsetUnsignedSaturation

Function name

__STATIC_INLINE void LL_ADC_SetOffsetUnsignedSaturation (ADC_TypeDef * ADCx, uint32_t Offsety, uint32_t OffsetUnsignedSaturation)

Function description

Set Unsigned saturation for the ADC selected offset instance 1, 2, 3 or 4: signed offset saturation if enabled or disabled.

Parameters

- **ADCx:** ADC instance
- **Offsety:** This parameter can be one of the following values:
  - LL_ADC_OFFSET_1
  - LL_ADC_OFFSET_2
  - LL_ADC_OFFSET_3
  - LL_ADC_OFFSET_4
- **OffsetUnsignedSaturation:** This parameter can be one of the following values:
  - LL_ADC_OFFSET_UNSIGNED_SAT_ENABLE
  - LL_ADC_OFFSET_UNSIGNED_SAT_DISABLE

Return values

- **None:**

Reference Manual to LL API cross reference:

- OFCFGR1 USAT LL_ADC_SetOffsetUnsignedSaturation
- OFCFGR2 USAT LL_ADC_SetOffsetUnsignedSaturation
- OFCFGR3 USAT LL_ADC_SetOffsetUnsignedSaturation
- OFCFGR4 USAT LL_ADC_SetOffsetUnsignedSaturation

LL_ADC_GetOffsetUnsignedSaturation

Function name

__STATIC_INLINE uint32_t LL_ADC_GetOffsetUnsignedSaturation (const ADC_TypeDef * ADCx, uint32_t Offsety)

Function description

Get Unsigned saturation for the ADC selected offset instance 1, 2, 3 or 4: signed offset saturation if enabled or disabled.

Parameters

- **ADCx:** ADC instance
- **Offsety:** This parameter can be one of the following values:
  - LL_ADC_OFFSET_1
  - LL_ADC_OFFSET_2
  - LL_ADC_OFFSET_3
  - LL_ADC_OFFSET_4

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_OFFSET_UNSIGNED_SAT_ENABLE
  - LL_ADC_OFFSET_UNSIGNED_SAT_DISABLE

Reference Manual to LL API cross reference:

- OFCFGR1 USAT LL_ADC_GetOffsetUnsignedSaturation
- OFCFGR2 USAT LL_ADC_GetOffsetUnsignedSaturation
- OFCFGR3 USAT LL_ADC_GetOffsetUnsignedSaturation
- OFCFGR4 USAT LL_ADC_GetOffsetUnsignedSaturation

LL_ADC_SetGainCompensation

Function name

__STATIC_INLINE void LL_ADC_SetGainCompensation (ADC_TypeDef * ADCx, uint32_t GainCompensation)

Function description

Set ADC gain compensation.

Parameters

- **ADCx:** ADC instance
- **GainCompensation:** This parameter can be: 0 Gain compensation will be disabled and value set to 0 1 -> 16393 Gain compensation will be enabled with specified value

Return values

- **None:**

Notes

- This function set the gain compensation coefficient that is applied to raw converted data using the formula: DATA = DATA(raw) * (gain compensation coef) / 4096
- This function enables the gain compensation if given coefficient is above 0, otherwise it disables it.
- Gain compensation when enabled is applied to all channels.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- GCOMP GCOMPCOEFF LL_ADC_SetGainCompensation
- CFGR2 GCOMP LL_ADC_SetGainCompensation

LL_ADC_GetGainCompensation

Function name

__STATIC_INLINE uint32_t LL_ADC_GetGainCompensation (const ADC_TypeDef * ADCx)

Function description

Get the ADC gain compensation value.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be: 0 Gain compensation is disabled 1 -> 16393 Gain compensation is enabled with returned value

Reference Manual to LL API cross reference:

- GCOMP GCOMPCOEFF LL_ADC_GetGainCompensation
- CFGR2 GCOMP LL_ADC_GetGainCompensation

LL_ADC_REG_SetTriggerSource

Function name

__STATIC_INLINE void LL_ADC_REG_SetTriggerSource (ADC_TypeDef * ADCx, uint32_t TriggerSource)

Function description

Set ADC group regular conversion trigger source: internal (SW start) or from external peripheral (timer event, external interrupt line).

Parameters

- **ADCx:** ADC instance
- **TriggerSource:** This parameter can be one of the following values:
  - LL_ADC_REG_TRIG_SOFTWARE
  - LL_ADC_REG_TRIG_EXT_EXTI_LINE11
  - LL_ADC_REG_TRIG_EXT_TIM1_CH1
  - LL_ADC_REG_TRIG_EXT_TIM1_CH2
  - LL_ADC_REG_TRIG_EXT_TIM1_CH3
  - LL_ADC_REG_TRIG_EXT_TIM1_TRGO
  - LL_ADC_REG_TRIG_EXT_TIM1_TRGO2
  - LL_ADC_REG_TRIG_EXT_TIM2_CH2
  - LL_ADC_REG_TRIG_EXT_TIM2_TRGO
  - LL_ADC_REG_TRIG_EXT_TIM3_CH4
  - LL_ADC_REG_TRIG_EXT_TIM3_TRGO
  - LL_ADC_REG_TRIG_EXT_TIM4_CH4
  - LL_ADC_REG_TRIG_EXT_TIM4_TRGO
  - LL_ADC_REG_TRIG_EXT_TIM5_TRGO
  - LL_ADC_REG_TRIG_EXT_TIM6_TRGO
  - LL_ADC_REG_TRIG_EXT_TIM7_TRGO
  - LL_ADC_REG_TRIG_EXT_TIM8_TRGO
  - LL_ADC_REG_TRIG_EXT_TIM8_TRGO2
  - LL_ADC_REG_TRIG_EXT_TIM9_CH1
  - LL_ADC_REG_TRIG_EXT_TIM9_TRGO
  - LL_ADC_REG_TRIG_EXT_TIM12_TRGO
  - LL_ADC_REG_TRIG_EXT_TIM15_TRGO
  - LL_ADC_REG_TRIG_EXT_TIM18_TRGO
  - LL_ADC_REG_TRIG_EXT_LPTIM1_CH1
  - LL_ADC_REG_TRIG_EXT_LPTIM2_CH1
  - LL_ADC_REG_TRIG_EXT_LPTIM3_CH1

Return values

- **None:**

Notes

- On this STM32 series, setting trigger source to external trigger also set trigger polarity to rising edge (default setting for compatibility with some ADC on other STM32 series having this setting set by HW default value). In case of need to modify trigger edge, use function LL_ADC_REG_SetTriggerEdge().
- Availability of parameters of trigger sources from timer depends on timers availability on the selected device.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.

Reference Manual to LL API cross reference:

- CFGR1 EXTSEL LL_ADC_REG_SetTriggerSource
- CFGR1 EXTEN LL_ADC_REG_SetTriggerSource

LL_ADC_REG_GetTriggerSource

Function name

__STATIC_INLINE uint32_t LL_ADC_REG_GetTriggerSource (const ADC_TypeDef * ADCx)

Function description

Get ADC group regular conversion trigger source: internal (SW start) or from external peripheral (timer event, external interrupt line).

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_REG_TRIG_SOFTWARE
  - LL_ADC_REG_TRIG_EXT_EXTI_LINE11
  - LL_ADC_REG_TRIG_EXT_TIM1_CH1
  - LL_ADC_REG_TRIG_EXT_TIM1_CH2
  - LL_ADC_REG_TRIG_EXT_TIM1_CH3
  - LL_ADC_REG_TRIG_EXT_TIM1_TRGO
  - LL_ADC_REG_TRIG_EXT_TIM1_TRGO2
  - LL_ADC_REG_TRIG_EXT_TIM2_CH2
  - LL_ADC_REG_TRIG_EXT_TIM2_TRGO
  - LL_ADC_REG_TRIG_EXT_TIM3_CH4
  - LL_ADC_REG_TRIG_EXT_TIM3_TRGO
  - LL_ADC_REG_TRIG_EXT_TIM4_CH4
  - LL_ADC_REG_TRIG_EXT_TIM4_TRGO
  - LL_ADC_REG_TRIG_EXT_TIM5_TRGO
  - LL_ADC_REG_TRIG_EXT_TIM6_TRGO
  - LL_ADC_REG_TRIG_EXT_TIM7_TRGO
  - LL_ADC_REG_TRIG_EXT_TIM8_TRGO
  - LL_ADC_REG_TRIG_EXT_TIM8_TRGO2
  - LL_ADC_REG_TRIG_EXT_TIM9_CH1
  - LL_ADC_REG_TRIG_EXT_TIM9_TRGO
  - LL_ADC_REG_TRIG_EXT_TIM12_TRGO
  - LL_ADC_REG_TRIG_EXT_TIM15_TRGO
  - LL_ADC_REG_TRIG_EXT_TIM18_TRGO
  - LL_ADC_REG_TRIG_EXT_LPTIM1_CH1
  - LL_ADC_REG_TRIG_EXT_LPTIM2_CH1
  - LL_ADC_REG_TRIG_EXT_LPTIM3_CH1

Notes

- To determine whether group regular trigger source is internal (SW start) or external, without detail of which peripheral is selected as external trigger, (equivalent to "if(LL_ADC_REG_GetTriggerSource(ADC1) == LL_ADC_REG_TRIG_SOFTWARE)") use function LL_ADC_REG_IsTriggerSourceSWStart.
- Availability of parameters of trigger sources from timer depends on timers availability on the selected device.

Reference Manual to LL API cross reference:

- CFGR1 EXTSEL LL_ADC_REG_GetTriggerSource
- CFGR1 EXTEN LL_ADC_REG_GetTriggerSource

LL_ADC_REG_IsTriggerSourceSWStart

Function name

__STATIC_INLINE uint32_t LL_ADC_REG_IsTriggerSourceSWStart (const ADC_TypeDef * ADCx)

Function description

Get ADC group regular conversion trigger source internal (SW start) or external.

Parameters

- **ADCx:** ADC instance

Return values

- **Value:** "0" if trigger source external trigger Value "1" if trigger source SW start.

Notes

- In case of group regular trigger source set to external trigger, to determine which peripheral is selected as external trigger, use function LL_ADC_REG_GetTriggerSource().

Reference Manual to LL API cross reference:

- CFGR1 EXTEN LL_ADC_REG_IsTriggerSourceSWStart

LL_ADC_REG_SetTriggerEdge

Function name

__STATIC_INLINE void LL_ADC_REG_SetTriggerEdge (ADC_TypeDef * ADCx, uint32_t ExternalTriggerEdge)

Function description

Set ADC group regular conversion trigger polarity.

Parameters

- **ADCx:** ADC instance
- **ExternalTriggerEdge:** This parameter can be one of the following values:
  - LL_ADC_REG_TRIG_EXT_RISING
  - LL_ADC_REG_TRIG_EXT_FALLING
  - LL_ADC_REG_TRIG_EXT_RISINGFALLING

Return values

- **None:**

Notes

- Applicable only for trigger source set to external trigger.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.

Reference Manual to LL API cross reference:

- CFGR1 EXTEN LL_ADC_REG_SetTriggerEdge

LL_ADC_REG_GetTriggerEdge

Function name

__STATIC_INLINE uint32_t LL_ADC_REG_GetTriggerEdge (const ADC_TypeDef * ADCx)

Function description

Get ADC group regular conversion trigger polarity.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_REG_TRIG_EXT_RISING
  - LL_ADC_REG_TRIG_EXT_FALLING
  - LL_ADC_REG_TRIG_EXT_RISINGFALLING

Notes

- Applicable only for trigger source set to external trigger.

Reference Manual to LL API cross reference:

- CFGR1 EXTEN LL_ADC_REG_GetTriggerEdge

LL_ADC_REG_SetSamplingMode

Function name

__STATIC_INLINE void LL_ADC_REG_SetSamplingMode (ADC_TypeDef * ADCx, uint32_t SamplingMode)

Function description

Set ADC sampling mode.

Parameters

- **ADCx:** ADC instance
- **SamplingMode:** This parameter can be one of the following values:
  - LL_ADC_REG_SAMPLING_MODE_NORMAL
  - LL_ADC_REG_SAMPLING_MODE_BULB
  - LL_ADC_REG_SAMPLING_MODE_TRIGGER_CONTROLED

Return values

- **None:**

Notes

- This function set the ADC conversion sampling mode
- This mode applies to regular group only.
- Set sampling mode is applied to all conversion of regular group.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.

Reference Manual to LL API cross reference:

- CFGR2 BULB LL_ADC_REG_SetSamplingMode
- CFGR2 SMPTRIG LL_ADC_REG_SetSamplingMode

LL_ADC_REG_GetSamplingMode

Function name

__STATIC_INLINE uint32_t LL_ADC_REG_GetSamplingMode (const ADC_TypeDef * ADCx)

Function description

Get the ADC sampling mode.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_REG_SAMPLING_MODE_NORMAL
  - LL_ADC_REG_SAMPLING_MODE_BULB
  - LL_ADC_REG_SAMPLING_MODE_TRIGGER_CONTROLED

Reference Manual to LL API cross reference:

- CFGR2 BULB LL_ADC_REG_GetSamplingMode
- CFGR2 SMPTRIG LL_ADC_REG_GetSamplingMode

LL_ADC_REG_StartSamplingPhase

Function name

__STATIC_INLINE void LL_ADC_REG_StartSamplingPhase (ADC_TypeDef * ADCx)

Function description

Start ADC sampling phase for sampling time trigger mode.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Notes

- This function is relevant only when LL_ADC_REG_SAMPLING_MODE_TRIGGER_CONTROLED has been set using LL_ADC_REG_SetSamplingModeLL_ADC_REG_TRIG_SOFTWARE is used as trigger source
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be enabled without conversion on going on group regular, without conversion stop command on going on group regular, without ADC disable command on going.

Reference Manual to LL API cross reference:

- CFGR2 SWTRIG LL_ADC_REG_StartSamplingPhase

LL_ADC_REG_StopSamplingPhase

Function name

__STATIC_INLINE void LL_ADC_REG_StopSamplingPhase (ADC_TypeDef * ADCx)

Function description

Stop ADC sampling phase for sampling time trigger mode and start conversion.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Notes

- This function is relevant only when LL_ADC_REG_SAMPLING_MODE_TRIGGER_CONTROLED has been set using LL_ADC_REG_SetSamplingModeLL_ADC_REG_TRIG_SOFTWARE is used as trigger sourceLL_ADC_REG_StartSamplingPhase has been called to start the sampling phase
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be enabled without conversion on going on group regular, without conversion stop command on going on group regular, without ADC disable command on going.

Reference Manual to LL API cross reference:

- CFGR2 SWTRIG LL_ADC_REG_StopSamplingPhase

LL_ADC_REG_SetSequencerLength

Function name

__STATIC_INLINE void LL_ADC_REG_SetSequencerLength (ADC_TypeDef * ADCx, uint32_t SequencerNbRanks)

Function description

Set ADC group regular sequencer length and scan direction.

Parameters

- **ADCx:** ADC instance
- **SequencerNbRanks:** This parameter can be one of the following values:
  - LL_ADC_REG_SEQ_SCAN_DISABLE
  - LL_ADC_REG_SEQ_SCAN_ENABLE_2RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_3RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_4RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_5RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_6RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_7RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_8RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_9RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_10RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_11RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_12RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_13RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_14RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_15RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_16RANKS

Return values

- **None:**

Notes

- Description of ADC group regular sequencer features: For devices with sequencer fully configurable (function "LL_ADC_REG_SetSequencerRanks()" available): sequencer length and each rank affectation to a channel are configurable. This function performs configuration of: Sequence length: Number of ranks in the scan sequence.Sequence direction: Unless specified in parameters, sequencer scan direction is forward (from rank 1 to rank n). Sequencer ranks are selected using function "LL_ADC_REG_SetSequencerRanks()". For devices with sequencer not fully configurable (function "LL_ADC_REG_SetSequencerChannels()" available): sequencer length and each rank affectation to a channel are defined by channel number. This function performs configuration of: Sequence length: Number of ranks in the scan sequence is defined by number of channels set in the sequence, rank of each channel is fixed by channel HW number. (channel 0 fixed on rank 0, channel 1 fixed on rank1, ...).Sequence direction: Unless specified in parameters, sequencer scan direction is forward (from lowest channel number to highest channel number). Sequencer ranks are selected using function "LL_ADC_REG_SetSequencerChannels()".
- Sequencer disabled is equivalent to sequencer of 1 rank: ADC conversion on only 1 channel.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.

Reference Manual to LL API cross reference:

- SQR1 L LL_ADC_REG_SetSequencerLength

LL_ADC_REG_GetSequencerLength

Function name

__STATIC_INLINE uint32_t LL_ADC_REG_GetSequencerLength (const ADC_TypeDef * ADCx)

Function description

Get ADC group regular sequencer length and scan direction.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_REG_SEQ_SCAN_DISABLE
  - LL_ADC_REG_SEQ_SCAN_ENABLE_2RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_3RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_4RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_5RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_6RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_7RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_8RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_9RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_10RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_11RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_12RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_13RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_14RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_15RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_16RANKS

Notes

- Description of ADC group regular sequencer features: For devices with sequencer fully configurable (function "LL_ADC_REG_SetSequencerRanks()" available): sequencer length and each rank affectation to a channel are configurable. This function retrieves: Sequence length: Number of ranks in the scan sequence.Sequence direction: Unless specified in parameters, sequencer scan direction is forward (from rank 1 to rank n). Sequencer ranks are selected using function "LL_ADC_REG_SetSequencerRanks()". For devices with sequencer not fully configurable (function "LL_ADC_REG_SetSequencerChannels()" available): sequencer length and each rank affectation to a channel are defined by channel number. This function retrieves: Sequence length: Number of ranks in the scan sequence is defined by number of channels set in the sequence, rank of each channel is fixed by channel HW number. (channel 0 fixed on rank 0, channel 1 fixed on rank1, ...).Sequence direction: Unless specified in parameters, sequencer scan direction is forward (from lowest channel number to highest channel number). Sequencer ranks are selected using function "LL_ADC_REG_SetSequencerChannels()".
- Sequencer disabled is equivalent to sequencer of 1 rank: ADC conversion on only 1 channel.

Reference Manual to LL API cross reference:

- SQR1 L LL_ADC_REG_GetSequencerLength

LL_ADC_REG_SetSequencerDiscont

Function name

__STATIC_INLINE void LL_ADC_REG_SetSequencerDiscont (ADC_TypeDef * ADCx, uint32_t SeqDiscont)

Function description

Set ADC group regular sequencer discontinuous mode: sequence subdivided and scan conversions interrupted every selected number of ranks.

Parameters

- **ADCx:** ADC instance
- **SeqDiscont:** This parameter can be one of the following values:
  - LL_ADC_REG_SEQ_DISCONT_DISABLE
  - LL_ADC_REG_SEQ_DISCONT_1RANK
  - LL_ADC_REG_SEQ_DISCONT_2RANKS
  - LL_ADC_REG_SEQ_DISCONT_3RANKS
  - LL_ADC_REG_SEQ_DISCONT_4RANKS
  - LL_ADC_REG_SEQ_DISCONT_5RANKS
  - LL_ADC_REG_SEQ_DISCONT_6RANKS
  - LL_ADC_REG_SEQ_DISCONT_7RANKS
  - LL_ADC_REG_SEQ_DISCONT_8RANKS

Return values

- **None:**

Notes

- It is not possible to enable both ADC group regular continuous mode and sequencer discontinuous mode.
- It is not possible to enable both ADC auto-injected mode and ADC group regular sequencer discontinuous mode.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.

Reference Manual to LL API cross reference:

- CFGR1 DISCEN LL_ADC_REG_SetSequencerDiscont
- CFGR1 DISCNUM LL_ADC_REG_SetSequencerDiscont

LL_ADC_REG_GetSequencerDiscont

Function name

__STATIC_INLINE uint32_t LL_ADC_REG_GetSequencerDiscont (const ADC_TypeDef * ADCx)

Function description

Get ADC group regular sequencer discontinuous mode: sequence subdivided and scan conversions interrupted every selected number of ranks.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_REG_SEQ_DISCONT_DISABLE
  - LL_ADC_REG_SEQ_DISCONT_1RANK
  - LL_ADC_REG_SEQ_DISCONT_2RANKS
  - LL_ADC_REG_SEQ_DISCONT_3RANKS
  - LL_ADC_REG_SEQ_DISCONT_4RANKS
  - LL_ADC_REG_SEQ_DISCONT_5RANKS
  - LL_ADC_REG_SEQ_DISCONT_6RANKS
  - LL_ADC_REG_SEQ_DISCONT_7RANKS
  - LL_ADC_REG_SEQ_DISCONT_8RANKS

Reference Manual to LL API cross reference:

- CFGR1 DISCEN LL_ADC_REG_GetSequencerDiscont
- CFGR1 DISCNUM LL_ADC_REG_GetSequencerDiscont

LL_ADC_REG_SetSequencerRanks

Function name

__STATIC_INLINE void LL_ADC_REG_SetSequencerRanks (ADC_TypeDef * ADCx, uint32_t Rank, uint32_t Channel)

Function description

Set ADC group regular sequence: channel on the selected scan sequence rank.

Parameters

- **ADCx:** ADC instance
- **Rank:** This parameter can be one of the following values:
  - LL_ADC_REG_RANK_1
  - LL_ADC_REG_RANK_2
  - LL_ADC_REG_RANK_3
  - LL_ADC_REG_RANK_4
  - LL_ADC_REG_RANK_5
  - LL_ADC_REG_RANK_6
  - LL_ADC_REG_RANK_7
  - LL_ADC_REG_RANK_8
  - LL_ADC_REG_RANK_9
  - LL_ADC_REG_RANK_10
  - LL_ADC_REG_RANK_11
  - LL_ADC_REG_RANK_12
  - LL_ADC_REG_RANK_13
  - LL_ADC_REG_RANK_14
  - LL_ADC_REG_RANK_15
  - LL_ADC_REG_RANK_16
- **Channel:** This parameter can be one of the following values:
  - LL_ADC_CHANNEL_0
  - LL_ADC_CHANNEL_1
  - LL_ADC_CHANNEL_2
  - LL_ADC_CHANNEL_3
  - LL_ADC_CHANNEL_4
  - LL_ADC_CHANNEL_5
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_19
  - LL_ADC_CHANNEL_VREFINT (1)
  - LL_ADC_CHANNEL_VBAT (2)
  - LL_ADC_CHANNEL_VDDCORE (2)

  (1) On this STM32 series, parameter available only on ADC instance: ADC1.
- (2) On this STM32 series, parameter available only on ADC instance: ADC2.
-

Return values

- **None:**

Notes

- This function performs configuration of: Channels ordering into each rank of scan sequence: whatever channel can be placed into whatever rank.
- On this STM32 series, ADC group regular sequencer is fully configurable: sequencer length and each rank affectation to a channel are configurable. Refer to description of function LL_ADC_REG_SetSequencerLength().
- Depending on devices and packages, some channels may not be available. Refer to device datasheet for channels availability.
- On this STM32 series, to measure internal channels (VrefInt, TempSensor, ...), measurement paths to internal channels must be enabled separately. This can be done using function LL_ADC_SetCommonPathInternalCh().
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.

Reference Manual to LL API cross reference:

- SQR1 SQ1 LL_ADC_REG_SetSequencerRanks
- SQR1 SQ2 LL_ADC_REG_SetSequencerRanks
- SQR1 SQ3 LL_ADC_REG_SetSequencerRanks
- SQR1 SQ4 LL_ADC_REG_SetSequencerRanks
- SQR2 SQ5 LL_ADC_REG_SetSequencerRanks
- SQR2 SQ6 LL_ADC_REG_SetSequencerRanks
- SQR2 SQ7 LL_ADC_REG_SetSequencerRanks
- SQR2 SQ8 LL_ADC_REG_SetSequencerRanks
- SQR2 SQ9 LL_ADC_REG_SetSequencerRanks
- SQR3 SQ10 LL_ADC_REG_SetSequencerRanks
- SQR3 SQ11 LL_ADC_REG_SetSequencerRanks
- SQR3 SQ12 LL_ADC_REG_SetSequencerRanks
- SQR3 SQ13 LL_ADC_REG_SetSequencerRanks
- SQR3 SQ14 LL_ADC_REG_SetSequencerRanks
- SQR4 SQ15 LL_ADC_REG_SetSequencerRanks
- SQR4 SQ16 LL_ADC_REG_SetSequencerRanks

LL_ADC_REG_GetSequencerRanks

Function name

__STATIC_INLINE uint32_t LL_ADC_REG_GetSequencerRanks (const ADC_TypeDef * ADCx, uint32_t Rank)

Function description

Get ADC group regular sequence: channel on the selected scan sequence rank.

Parameters

- **ADCx:** ADC instance
- **Rank:** This parameter can be one of the following values:
  - LL_ADC_REG_RANK_1
  - LL_ADC_REG_RANK_2
  - LL_ADC_REG_RANK_3
  - LL_ADC_REG_RANK_4
  - LL_ADC_REG_RANK_5
  - LL_ADC_REG_RANK_6
  - LL_ADC_REG_RANK_7
  - LL_ADC_REG_RANK_8
  - LL_ADC_REG_RANK_9
  - LL_ADC_REG_RANK_10
  - LL_ADC_REG_RANK_11
  - LL_ADC_REG_RANK_12
  - LL_ADC_REG_RANK_13
  - LL_ADC_REG_RANK_14
  - LL_ADC_REG_RANK_15
  - LL_ADC_REG_RANK_16

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_CHANNEL_0
  - LL_ADC_CHANNEL_1
  - LL_ADC_CHANNEL_2
  - LL_ADC_CHANNEL_3
  - LL_ADC_CHANNEL_4
  - LL_ADC_CHANNEL_5
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_19
  - LL_ADC_CHANNEL_VREFINT (1)(3)
  - LL_ADC_CHANNEL_VBAT (2)(3)
  - LL_ADC_CHANNEL_VDDCORE (2)(3)

  (1) On this STM32 series, parameter available only on ADC instance: ADC1.
- (2) On this STM32 series, parameter available only on ADC instance: ADC2.
- (3) For ADC channel read back from ADC register, comparison with internal channel parameter to be done using helper macro __LL_ADC_CHANNEL_INTERNAL_TO_EXTERNAL().

Notes

- On this STM32 series, ADC group regular sequencer is fully configurable: sequencer length and each rank affectation to a channel are configurable. Refer to description of function LL_ADC_REG_SetSequencerLength().
- Depending on devices and packages, some channels may not be available. Refer to device datasheet for channels availability.
- Usage of the returned channel number: To reinject this channel into another function LL_ADC_xxx: the returned channel number is only partly formatted on definition of literals LL_ADC_CHANNEL_x. Therefore, it has to be compared with parts of literals LL_ADC_CHANNEL_x or using helper macro __LL_ADC_CHANNEL_TO_DECIMAL_NB(). Then the selected literal LL_ADC_CHANNEL_x can be used as parameter for another function.To get the channel number in decimal format: process the returned value with the helper macro __LL_ADC_CHANNEL_TO_DECIMAL_NB().

Reference Manual to LL API cross reference:

- SQR1 SQ1 LL_ADC_REG_GetSequencerRanks
- SQR1 SQ2 LL_ADC_REG_GetSequencerRanks
- SQR1 SQ3 LL_ADC_REG_GetSequencerRanks
- SQR1 SQ4 LL_ADC_REG_GetSequencerRanks
- SQR2 SQ5 LL_ADC_REG_GetSequencerRanks
- SQR2 SQ6 LL_ADC_REG_GetSequencerRanks
- SQR2 SQ7 LL_ADC_REG_GetSequencerRanks
- SQR2 SQ8 LL_ADC_REG_GetSequencerRanks
- SQR2 SQ9 LL_ADC_REG_GetSequencerRanks
- SQR3 SQ10 LL_ADC_REG_GetSequencerRanks
- SQR3 SQ11 LL_ADC_REG_GetSequencerRanks
- SQR3 SQ12 LL_ADC_REG_GetSequencerRanks
- SQR3 SQ13 LL_ADC_REG_GetSequencerRanks
- SQR3 SQ14 LL_ADC_REG_GetSequencerRanks
- SQR4 SQ15 LL_ADC_REG_GetSequencerRanks
- SQR4 SQ16 LL_ADC_REG_GetSequencerRanks

LL_ADC_SetChannelPreselection

Function name

__STATIC_INLINE void LL_ADC_SetChannelPreselection (ADC_TypeDef * ADCx, uint32_t Channel)

Function description

Set ADC Channel Preselection to LL_ADC_CHANNEL_x, x = 0 to 19.

Parameters

- **ADCx:** ADC instance.
- **Channel:** This parameter can be one of the following values:
  - LL_ADC_CHANNEL_0
  - LL_ADC_CHANNEL_1
  - LL_ADC_CHANNEL_2
  - LL_ADC_CHANNEL_3
  - LL_ADC_CHANNEL_4
  - LL_ADC_CHANNEL_5
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_19
  - LL_ADC_CHANNEL_VREFINT (1)
  - LL_ADC_CHANNEL_VBAT (2)
  - LL_ADC_CHANNEL_VDDCORE (2)

  (1) On this STM32 series, parameter available only on ADC instance: ADC1.
- (2) On this STM32 series, parameter available only on ADC instance: ADC2.
-

Return values

- **None:**

Notes

- This function set the the value for the channel preselection register corresponding to ADC channel to be selected.

Reference Manual to LL API cross reference:

- PCSEL PCSEL LL_ADC_SetChannelPreselection

LL_ADC_GetChannelPreselection

Function name

__STATIC_INLINE uint32_t LL_ADC_GetChannelPreselection (const ADC_TypeDef * ADCx)

Function description

Get ADC Channel Preselection register value.

Parameters

- **ADCx:** ADC instance.

Return values

- **Returned:** decimal value that can correspond to one or multiple channels:

Notes

- This function set the the value for the channel preselection register corresponding to ADC channel to be selected.

Reference Manual to LL API cross reference:

- PCSEL PCSEL LL_ADC_GetChannelPreselection

LL_ADC_REG_SetContinuousMode

Function name

__STATIC_INLINE void LL_ADC_REG_SetContinuousMode (ADC_TypeDef * ADCx, uint32_t Continuous)

Function description

Set ADC continuous conversion mode on ADC group regular.

Parameters

- **ADCx:** ADC instance
- **Continuous:** This parameter can be one of the following values:
  - LL_ADC_REG_CONV_SINGLE
  - LL_ADC_REG_CONV_CONTINUOUS

Return values

- **None:**

Notes

- Description of ADC continuous conversion mode: single mode: one conversion per triggercontinuous mode: after the first trigger, following conversions launched successively automatically.
- It is not possible to enable both ADC group regular continuous mode and sequencer discontinuous mode.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.

Reference Manual to LL API cross reference:

- CFGR1 CONT LL_ADC_REG_SetContinuousMode

LL_ADC_REG_GetContinuousMode

Function name

__STATIC_INLINE uint32_t LL_ADC_REG_GetContinuousMode (const ADC_TypeDef * ADCx)

Function description

Get ADC continuous conversion mode on ADC group regular.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_REG_CONV_SINGLE
  - LL_ADC_REG_CONV_CONTINUOUS

Notes

- Description of ADC continuous conversion mode: single mode: one conversion per triggercontinuous mode: after the first trigger, following conversions launched successively automatically.

Reference Manual to LL API cross reference:

- CFGR1 CONT LL_ADC_REG_GetContinuousMode

LL_ADC_REG_SetDataTransferMode

Function name

__STATIC_INLINE void LL_ADC_REG_SetDataTransferMode (ADC_TypeDef * ADCx, uint32_t DataTransferMode)

Function description

Set ADC data transfer mode.

Parameters

- **ADCx:** ADC instance
- **DataTransferMode:** Select Data Management configuration
  - LL_ADC_REG_DR_TRANSFER
  - LL_ADC_REG_DMA_TRANSFER_LIMITED
  - LL_ADC_REG_DMA_TRANSFER_UNLIMITED
  - LL_ADC_REG_MDF_TRANSFER

Return values

- **None:**

Notes

- Conversion data can be either: Available in Data RegisterTransferred by DMA in one shot modeTransferred by DMA in circular modeTransferred to MDF data register

Reference Manual to LL API cross reference:

- CFGR1 DMNGT LL_ADC_REG_SetDataTransferMode

LL_ADC_REG_GetDataTransferMode

Function name

__STATIC_INLINE uint32_t LL_ADC_REG_GetDataTransferMode (const ADC_TypeDef * ADCx)

Function description

Get ADC data transfer mode.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_REG_DR_TRANSFER
  - LL_ADC_REG_DMA_TRANSFER_LIMITED
  - LL_ADC_REG_DMA_TRANSFER_UNLIMITED
  - LL_ADC_REG_MDF_TRANSFER

Notes

- Conversion data can be either: Available in Data RegisterTransferred by DMA in one shot modeTransferred by DMA in circular modeTransferred to DFSDM data register

Reference Manual to LL API cross reference:

- CFGR1 DMNGT LL_ADC_REG_GetDataTransferMode

LL_ADC_REG_SetOverrun

Function name

__STATIC_INLINE void LL_ADC_REG_SetOverrun (ADC_TypeDef * ADCx, uint32_t Overrun)

Function description

Set ADC group regular behavior in case of overrun: data preserved or overwritten.

Parameters

- **ADCx:** ADC instance
- **Overrun:** This parameter can be one of the following values:
  - LL_ADC_REG_OVR_DATA_PRESERVED
  - LL_ADC_REG_OVR_DATA_OVERWRITTEN

Return values

- **None:**

Notes

- Compatibility with devices without feature overrun: other devices without this feature have a behavior equivalent to data overwritten. The default setting of overrun is data preserved. Therefore, for compatibility with all devices, parameter overrun should be set to data overwritten.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.

Reference Manual to LL API cross reference:

- CFGR1 OVRMOD LL_ADC_REG_SetOverrun

LL_ADC_REG_GetOverrun

Function name

__STATIC_INLINE uint32_t LL_ADC_REG_GetOverrun (const ADC_TypeDef * ADCx)

Function description

Get ADC group regular behavior in case of overrun: data preserved or overwritten.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_REG_OVR_DATA_PRESERVED
  - LL_ADC_REG_OVR_DATA_OVERWRITTEN

Reference Manual to LL API cross reference:

- CFGR1 OVRMOD LL_ADC_REG_GetOverrun

LL_ADC_INJ_SetTriggerSource

Function name

__STATIC_INLINE void LL_ADC_INJ_SetTriggerSource (ADC_TypeDef * ADCx, uint32_t TriggerSource)

Function description

Set ADC group injected conversion trigger source: internal (SW start) or from external peripheral (timer event, external interrupt line).

Parameters

- **ADCx:** ADC instance
- **TriggerSource:** This parameter can be one of the following values:
  - LL_ADC_INJ_TRIG_SOFTWARE
  - LL_ADC_INJ_TRIG_EXT_EXTI_LINE15
  - LL_ADC_INJ_TRIG_EXT_TIM1_CH4
  - LL_ADC_INJ_TRIG_EXT_TIM1_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM1_TRGO2
  - LL_ADC_INJ_TRIG_EXT_TIM2_CH1
  - LL_ADC_INJ_TRIG_EXT_TIM2_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM3_CH1
  - LL_ADC_INJ_TRIG_EXT_TIM3_CH3
  - LL_ADC_INJ_TRIG_EXT_TIM3_CH4
  - LL_ADC_INJ_TRIG_EXT_TIM3_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM4_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM5_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM6_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM7_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM8_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM8_TRGO2
  - LL_ADC_INJ_TRIG_EXT_TIM9_CH2
  - LL_ADC_INJ_TRIG_EXT_TIM9_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM12_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM15_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM18_TRGO
  - LL_ADC_INJ_TRIG_EXT_LPTIM1_CH2
  - LL_ADC_INJ_TRIG_EXT_LPTIM2_CH2
  - LL_ADC_INJ_TRIG_EXT_LPTIM3_CH2

Return values

- **None:**

Notes

- On this STM32 series, setting trigger source to external trigger also set trigger polarity to rising edge (default setting for compatibility with some ADC on other STM32 series having this setting set by HW default value). In case of need to modify trigger edge, use function LL_ADC_INJ_SetTriggerEdge().
- Availability of parameters of trigger sources from timer depends on timers availability on the selected device.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must not be disabled. Can be enabled with or without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- JSQR JEXTSEL LL_ADC_INJ_SetTriggerSource
- JSQR JEXTEN LL_ADC_INJ_SetTriggerSource

LL_ADC_INJ_GetTriggerSource

Function name

__STATIC_INLINE uint32_t LL_ADC_INJ_GetTriggerSource (const ADC_TypeDef * ADCx)

Function description

Get ADC group injected conversion trigger source: internal (SW start) or from external peripheral (timer event, external interrupt line).

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_INJ_TRIG_SOFTWARE
  - LL_ADC_INJ_TRIG_EXT_EXTI_LINE15
  - LL_ADC_INJ_TRIG_EXT_TIM1_CH4
  - LL_ADC_INJ_TRIG_EXT_TIM1_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM1_TRGO2
  - LL_ADC_INJ_TRIG_EXT_TIM2_CH1
  - LL_ADC_INJ_TRIG_EXT_TIM2_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM3_CH1
  - LL_ADC_INJ_TRIG_EXT_TIM3_CH3
  - LL_ADC_INJ_TRIG_EXT_TIM3_CH4
  - LL_ADC_INJ_TRIG_EXT_TIM3_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM4_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM5_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM6_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM7_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM8_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM8_TRGO2
  - LL_ADC_INJ_TRIG_EXT_TIM9_CH2
  - LL_ADC_INJ_TRIG_EXT_TIM9_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM12_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM15_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM18_TRGO
  - LL_ADC_INJ_TRIG_EXT_LPTIM1_CH2
  - LL_ADC_INJ_TRIG_EXT_LPTIM2_CH2
  - LL_ADC_INJ_TRIG_EXT_LPTIM3_CH2

Notes

- To determine whether group injected trigger source is internal (SW start) or external, without detail of which peripheral is selected as external trigger, (equivalent to "if(LL_ADC_INJ_GetTriggerSource(ADC1) == LL_ADC_INJ_TRIG_SOFTWARE)") use function LL_ADC_INJ_IsTriggerSourceSWStart.
- Availability of parameters of trigger sources from timer depends on timers availability on the selected device.

Reference Manual to LL API cross reference:

- JSQR JEXTSEL LL_ADC_INJ_GetTriggerSource
- JSQR JEXTEN LL_ADC_INJ_GetTriggerSource

LL_ADC_INJ_IsTriggerSourceSWStart

Function name

__STATIC_INLINE uint32_t LL_ADC_INJ_IsTriggerSourceSWStart (const ADC_TypeDef * ADCx)

Function description

Get ADC group injected conversion trigger source internal (SW start) or external.

Parameters

- **ADCx:** ADC instance

Return values

- **Value:** "0" if trigger source external trigger Value "1" if trigger source SW start.

Notes

- In case of group injected trigger source set to external trigger, to determine which peripheral is selected as external trigger, use function LL_ADC_INJ_GetTriggerSource.

Reference Manual to LL API cross reference:

- JSQR JEXTEN LL_ADC_INJ_IsTriggerSourceSWStart

LL_ADC_INJ_SetTriggerEdge

Function name

__STATIC_INLINE void LL_ADC_INJ_SetTriggerEdge (ADC_TypeDef * ADCx, uint32_t ExternalTriggerEdge)

Function description

Set ADC group injected conversion trigger polarity.

Parameters

- **ADCx:** ADC instance
- **ExternalTriggerEdge:** This parameter can be one of the following values:
  - LL_ADC_INJ_TRIG_EXT_RISING
  - LL_ADC_INJ_TRIG_EXT_FALLING
  - LL_ADC_INJ_TRIG_EXT_RISINGFALLING

Return values

- **None:**

Notes

- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must not be disabled. Can be enabled with or without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- JSQR JEXTEN LL_ADC_INJ_SetTriggerEdge

LL_ADC_INJ_GetTriggerEdge

Function name

__STATIC_INLINE uint32_t LL_ADC_INJ_GetTriggerEdge (const ADC_TypeDef * ADCx)

Function description

Get ADC group injected conversion trigger polarity.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_INJ_TRIG_EXT_RISING
  - LL_ADC_INJ_TRIG_EXT_FALLING
  - LL_ADC_INJ_TRIG_EXT_RISINGFALLING

Reference Manual to LL API cross reference:

- JSQR JEXTEN LL_ADC_INJ_GetTriggerEdge

LL_ADC_INJ_SetSequencerLength

Function name

__STATIC_INLINE void LL_ADC_INJ_SetSequencerLength (ADC_TypeDef * ADCx, uint32_t SequencerNbRanks)

Function description

Set ADC group injected sequencer length and scan direction.

Parameters

- **ADCx:** ADC instance
- **SequencerNbRanks:** This parameter can be one of the following values:
  - LL_ADC_INJ_SEQ_SCAN_DISABLE
  - LL_ADC_INJ_SEQ_SCAN_ENABLE_2RANKS
  - LL_ADC_INJ_SEQ_SCAN_ENABLE_3RANKS
  - LL_ADC_INJ_SEQ_SCAN_ENABLE_4RANKS

Return values

- **None:**

Notes

- This function performs configuration of: Sequence length: Number of ranks in the scan sequence.Sequence direction: Unless specified in parameters, sequencer scan direction is forward (from rank 1 to rank n).
- Sequencer disabled is equivalent to sequencer of 1 rank: ADC conversion on only 1 channel.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must not be disabled. Can be enabled with or without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- JSQR JL LL_ADC_INJ_SetSequencerLength

LL_ADC_INJ_GetSequencerLength

Function name

__STATIC_INLINE uint32_t LL_ADC_INJ_GetSequencerLength (const ADC_TypeDef * ADCx)

Function description

Get ADC group injected sequencer length and scan direction.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_INJ_SEQ_SCAN_DISABLE
  - LL_ADC_INJ_SEQ_SCAN_ENABLE_2RANKS
  - LL_ADC_INJ_SEQ_SCAN_ENABLE_3RANKS
  - LL_ADC_INJ_SEQ_SCAN_ENABLE_4RANKS

Notes

- This function retrieves: Sequence length: Number of ranks in the scan sequence.Sequence direction: Unless specified in parameters, sequencer scan direction is forward (from rank 1 to rank n).
- Sequencer disabled is equivalent to sequencer of 1 rank: ADC conversion on only 1 channel.

Reference Manual to LL API cross reference:

- JSQR JL LL_ADC_INJ_GetSequencerLength

LL_ADC_INJ_SetSequencerDiscont

Function name

__STATIC_INLINE void LL_ADC_INJ_SetSequencerDiscont (ADC_TypeDef * ADCx, uint32_t SeqDiscont)

Function description

Set ADC group injected sequencer discontinuous mode: sequence subdivided and scan conversions interrupted every selected number of ranks.

Parameters

- **ADCx:** ADC instance
- **SeqDiscont:** This parameter can be one of the following values:
  - LL_ADC_INJ_SEQ_DISCONT_DISABLE
  - LL_ADC_INJ_SEQ_DISCONT_1RANK

Return values

- **None:**

Notes

- It is not possible to enable both ADC group injected auto-injected mode and sequencer discontinuous mode.

Reference Manual to LL API cross reference:

- CFGR1 JDISCEN LL_ADC_INJ_SetSequencerDiscont

LL_ADC_INJ_GetSequencerDiscont

Function name

__STATIC_INLINE uint32_t LL_ADC_INJ_GetSequencerDiscont (const ADC_TypeDef * ADCx)

Function description

Get ADC group injected sequencer discontinuous mode: sequence subdivided and scan conversions interrupted every selected number of ranks.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_INJ_SEQ_DISCONT_DISABLE
  - LL_ADC_INJ_SEQ_DISCONT_1RANK

Reference Manual to LL API cross reference:

- CFGR1 JDISCEN LL_ADC_INJ_GetSequencerDiscont

LL_ADC_INJ_SetSequencerRanks

Function name

__STATIC_INLINE void LL_ADC_INJ_SetSequencerRanks (ADC_TypeDef * ADCx, uint32_t Rank, uint32_t Channel)

Function description

Set ADC group injected sequence: channel on the selected sequence rank.

Parameters

- **ADCx:** ADC instance
- **Rank:** This parameter can be one of the following values:
  - LL_ADC_INJ_RANK_1
  - LL_ADC_INJ_RANK_2
  - LL_ADC_INJ_RANK_3
  - LL_ADC_INJ_RANK_4
- **Channel:** This parameter can be one of the following values:
  - LL_ADC_CHANNEL_0
  - LL_ADC_CHANNEL_1
  - LL_ADC_CHANNEL_2
  - LL_ADC_CHANNEL_3
  - LL_ADC_CHANNEL_4
  - LL_ADC_CHANNEL_5
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_19
  - LL_ADC_CHANNEL_VREFINT (1)
  - LL_ADC_CHANNEL_VBAT (2)
  - LL_ADC_CHANNEL_VDDCORE (2)

  (1) On this STM32 series, parameter available only on ADC instance: ADC1.
- (2) On this STM32 series, parameter available only on ADC instance: ADC2.
-

Return values

- **None:**

Notes

- Depending on devices and packages, some channels may not be available. Refer to device datasheet for channels availability.
- On this STM32 series, to measure internal channels (VrefInt, TempSensor, ...), measurement paths to internal channels must be enabled separately. This can be done using function LL_ADC_SetCommonPathInternalCh().
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must not be disabled. Can be enabled with or without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- JSQR JSQ1 LL_ADC_INJ_SetSequencerRanks
- JSQR JSQ2 LL_ADC_INJ_SetSequencerRanks
- JSQR JSQ3 LL_ADC_INJ_SetSequencerRanks
- JSQR JSQ4 LL_ADC_INJ_SetSequencerRanks

LL_ADC_INJ_GetSequencerRanks

Function name

__STATIC_INLINE uint32_t LL_ADC_INJ_GetSequencerRanks (const ADC_TypeDef * ADCx, uint32_t Rank)

Function description

Get ADC group injected sequence: channel on the selected sequence rank.

Parameters

- **ADCx:** ADC instance
- **Rank:** This parameter can be one of the following values:
  - LL_ADC_INJ_RANK_1
  - LL_ADC_INJ_RANK_2
  - LL_ADC_INJ_RANK_3
  - LL_ADC_INJ_RANK_4

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_CHANNEL_0
  - LL_ADC_CHANNEL_1
  - LL_ADC_CHANNEL_2
  - LL_ADC_CHANNEL_3
  - LL_ADC_CHANNEL_4
  - LL_ADC_CHANNEL_5
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_19
  - LL_ADC_CHANNEL_VREFINT (1)(3)
  - LL_ADC_CHANNEL_VBAT (2)(3)
  - LL_ADC_CHANNEL_VDDCORE (2)(3)

  (1) On this STM32 series, parameter available only on ADC instance: ADC1.
- (2) On this STM32 series, parameter available only on ADC instance: ADC2.
- (3) For ADC channel read back from ADC register, comparison with internal channel parameter to be done using helper macro __LL_ADC_CHANNEL_INTERNAL_TO_EXTERNAL().

Notes

- Depending on devices and packages, some channels may not be available. Refer to device datasheet for channels availability.
- Usage of the returned channel number: To reinject this channel into another function LL_ADC_xxx: the returned channel number is only partly formatted on definition of literals LL_ADC_CHANNEL_x. Therefore, it has to be compared with parts of literals LL_ADC_CHANNEL_x or using helper macro __LL_ADC_CHANNEL_TO_DECIMAL_NB(). Then the selected literal LL_ADC_CHANNEL_x can be used as parameter for another function.To get the channel number in decimal format: process the returned value with the helper macro __LL_ADC_CHANNEL_TO_DECIMAL_NB().

Reference Manual to LL API cross reference:

- JSQR JSQ1 LL_ADC_INJ_GetSequencerRanks
- JSQR JSQ2 LL_ADC_INJ_GetSequencerRanks
- JSQR JSQ3 LL_ADC_INJ_GetSequencerRanks
- JSQR JSQ4 LL_ADC_INJ_GetSequencerRanks

LL_ADC_INJ_SetTrigAuto

Function name

__STATIC_INLINE void LL_ADC_INJ_SetTrigAuto (ADC_TypeDef * ADCx, uint32_t TrigAuto)

Function description

Set ADC group injected conversion trigger: independent or from ADC group regular.

Parameters

- **ADCx:** ADC instance
- **TrigAuto:** This parameter can be one of the following values:
  - LL_ADC_INJ_TRIG_INDEPENDENT
  - LL_ADC_INJ_TRIG_FROM_GRP_REGULAR

Return values

- **None:**

Notes

- This mode can be used to extend number of data registers updated after one ADC conversion trigger and with data permanently kept (not erased by successive conversions of scan of ADC sequencer ranks), up to 5 data registers: 1 data register on ADC group regular, 4 data registers on ADC group injected.
- If ADC group injected injected trigger source is set to an external trigger, this feature must be must be set to independent trigger. ADC group injected automatic trigger is compliant only with group injected trigger source set to SW start, without any further action on ADC group injected conversion start or stop: in this case, ADC group injected is controlled only from ADC group regular.
- It is not possible to enable both ADC group injected auto-injected mode and sequencer discontinuous mode.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- CFGR1 JAUTO LL_ADC_INJ_SetTrigAuto

LL_ADC_INJ_GetTrigAuto

Function name

__STATIC_INLINE uint32_t LL_ADC_INJ_GetTrigAuto (const ADC_TypeDef * ADCx)

Function description

Get ADC group injected conversion trigger: independent or from ADC group regular.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_INJ_TRIG_INDEPENDENT
  - LL_ADC_INJ_TRIG_FROM_GRP_REGULAR

Reference Manual to LL API cross reference:

- CFGR1 JAUTO LL_ADC_INJ_GetTrigAuto

LL_ADC_SetChannelSamplingTime

Function name

__STATIC_INLINE void LL_ADC_SetChannelSamplingTime (ADC_TypeDef * ADCx, uint32_t Channel, uint32_t SamplingTime)

Function description

Set sampling time of the selected ADC channel Unit: ADC clock cycles.

Parameters

- **ADCx:** ADC instance
- **Channel:** This parameter can be one of the following values:
  - LL_ADC_CHANNEL_0
  - LL_ADC_CHANNEL_1
  - LL_ADC_CHANNEL_2
  - LL_ADC_CHANNEL_3
  - LL_ADC_CHANNEL_4
  - LL_ADC_CHANNEL_5
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_19
  - LL_ADC_CHANNEL_VREFINT (1)
  - LL_ADC_CHANNEL_VBAT (2)
  - LL_ADC_CHANNEL_VDDCORE (2)

  (1) On this STM32 series, parameter available only on ADC instance: ADC1.
- (2) On this STM32 series, parameter available only on ADC instance: ADC2.
-
- **SamplingTime:** This parameter can be one of the following values:
  - LL_ADC_SAMPLINGTIME_1CYCLE_5
  - LL_ADC_SAMPLINGTIME_2CYCLES_5
  - LL_ADC_SAMPLINGTIME_6CYCLES_5
  - LL_ADC_SAMPLINGTIME_11CYCLES_5
  - LL_ADC_SAMPLINGTIME_23CYCLES_5
  - LL_ADC_SAMPLINGTIME_46CYCLES_5
  - LL_ADC_SAMPLINGTIME_246CYCLES_5
  - LL_ADC_SAMPLINGTIME_1499CYCLES_5

Return values

- **None:**

Notes

- On this device, sampling time is on channel scope: independently of channel mapped on ADC group regular or injected.
- In case of internal channel (VrefInt, TempSensor, ...) to be converted: sampling time constraints must be respected (sampling time can be adjusted in function of ADC clock frequency and sampling time setting). Refer to device datasheet for timings values (parameters TS_vrefint, TS_temp, ...).
- Conversion time is the addition of sampling time and processing time. On this STM32 series, ADC processing time is: 13.5 ADC clock cycles at ADC resolution 12 bits11.5 ADC clock cycles at ADC resolution 10 bits8.5 ADC clock cycles at ADC resolution 8 bits6.5 ADC clock cycles at ADC resolution 6 bits
- In case of ADC conversion of internal channel (VrefInt, temperature sensor, ...), a sampling time minimum value is required. Refer to device datasheet.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- SMPR1 SMP0 LL_ADC_SetChannelSamplingTime
- SMPR1 SMP1 LL_ADC_SetChannelSamplingTime
- SMPR1 SMP2 LL_ADC_SetChannelSamplingTime
- SMPR1 SMP3 LL_ADC_SetChannelSamplingTime
- SMPR1 SMP4 LL_ADC_SetChannelSamplingTime
- SMPR1 SMP5 LL_ADC_SetChannelSamplingTime
- SMPR1 SMP6 LL_ADC_SetChannelSamplingTime
- SMPR1 SMP7 LL_ADC_SetChannelSamplingTime
- SMPR1 SMP8 LL_ADC_SetChannelSamplingTime
- SMPR1 SMP9 LL_ADC_SetChannelSamplingTime
- SMPR2 SMP10 LL_ADC_SetChannelSamplingTime
- SMPR2 SMP11 LL_ADC_SetChannelSamplingTime
- SMPR2 SMP12 LL_ADC_SetChannelSamplingTime
- SMPR2 SMP13 LL_ADC_SetChannelSamplingTime
- SMPR2 SMP14 LL_ADC_SetChannelSamplingTime
- SMPR2 SMP15 LL_ADC_SetChannelSamplingTime
- SMPR2 SMP16 LL_ADC_SetChannelSamplingTime
- SMPR2 SMP17 LL_ADC_SetChannelSamplingTime
- SMPR2 SMP18 LL_ADC_SetChannelSamplingTime

LL_ADC_GetChannelSamplingTime

Function name

__STATIC_INLINE uint32_t LL_ADC_GetChannelSamplingTime (const ADC_TypeDef * ADCx, uint32_t Channel)

Function description

Get sampling time of the selected ADC channel Unit: ADC clock cycles.

Parameters

- **ADCx:** ADC instance
- **Channel:** This parameter can be one of the following values:
  - LL_ADC_CHANNEL_0
  - LL_ADC_CHANNEL_1
  - LL_ADC_CHANNEL_2
  - LL_ADC_CHANNEL_3
  - LL_ADC_CHANNEL_4
  - LL_ADC_CHANNEL_5
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_19
  - LL_ADC_CHANNEL_VREFINT (1)(3)
  - LL_ADC_CHANNEL_VBAT (2)(3)
  - LL_ADC_CHANNEL_VDDCORE (2)(3)

  (1) On this STM32 series, parameter available only on ADC instance: ADC1.
- (2) On this STM32 series, parameter available only on ADC instance: ADC2.
- (3) For ADC channel read back from ADC register, comparison with internal channel parameter to be done using helper macro __LL_ADC_CHANNEL_INTERNAL_TO_EXTERNAL().

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_SAMPLINGTIME_1CYCLE_5
  - LL_ADC_SAMPLINGTIME_2CYCLES_5
  - LL_ADC_SAMPLINGTIME_6CYCLES_5
  - LL_ADC_SAMPLINGTIME_11CYCLES_5
  - LL_ADC_SAMPLINGTIME_23CYCLES_5
  - LL_ADC_SAMPLINGTIME_46CYCLES_5
  - LL_ADC_SAMPLINGTIME_246CYCLES_5
  - LL_ADC_SAMPLINGTIME_1499CYCLES_5

Notes

- On this device, sampling time is on channel scope: independently of channel mapped on ADC group regular or injected.
- Conversion time is the addition of sampling time and processing time. On this STM32 series, ADC processing time is: 13.5 ADC clock cycles at ADC resolution 12 bits11.5 ADC clock cycles at ADC resolution 10 bits8.5 ADC clock cycles at ADC resolution 8 bits6.5 ADC clock cycles at ADC resolution 6 bits

Reference Manual to LL API cross reference:

- SMPR1 SMP0 LL_ADC_GetChannelSamplingTime
- SMPR1 SMP1 LL_ADC_GetChannelSamplingTime
- SMPR1 SMP2 LL_ADC_GetChannelSamplingTime
- SMPR1 SMP3 LL_ADC_GetChannelSamplingTime
- SMPR1 SMP4 LL_ADC_GetChannelSamplingTime
- SMPR1 SMP5 LL_ADC_GetChannelSamplingTime
- SMPR1 SMP6 LL_ADC_GetChannelSamplingTime
- SMPR1 SMP7 LL_ADC_GetChannelSamplingTime
- SMPR1 SMP8 LL_ADC_GetChannelSamplingTime
- SMPR1 SMP9 LL_ADC_GetChannelSamplingTime
- SMPR2 SMP10 LL_ADC_GetChannelSamplingTime
- SMPR2 SMP11 LL_ADC_GetChannelSamplingTime
- SMPR2 SMP12 LL_ADC_GetChannelSamplingTime
- SMPR2 SMP13 LL_ADC_GetChannelSamplingTime
- SMPR2 SMP14 LL_ADC_GetChannelSamplingTime
- SMPR2 SMP15 LL_ADC_GetChannelSamplingTime
- SMPR2 SMP16 LL_ADC_GetChannelSamplingTime
- SMPR2 SMP17 LL_ADC_GetChannelSamplingTime
- SMPR2 SMP18 LL_ADC_GetChannelSamplingTime

LL_ADC_SetChannelSingleDiff

Function name

__STATIC_INLINE void LL_ADC_SetChannelSingleDiff (ADC_TypeDef * ADCx, uint32_t Channel, uint32_t SingleDiff)

Function description

Set mode single-ended or differential input of the selected ADC channel.

Parameters

- **ADCx:** ADC instance
- **Channel:** This parameter can be one of the following values:
  - LL_ADC_CHANNEL_1
  - LL_ADC_CHANNEL_2
  - LL_ADC_CHANNEL_3
  - LL_ADC_CHANNEL_4
  - LL_ADC_CHANNEL_5
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_19
- **SingleDiff:** This parameter can be a combination of the following values:
  - LL_ADC_SINGLE_ENDED
  - LL_ADC_DIFFERENTIAL_ENDED

Return values

- **None:**

Notes

- Channel ending is on channel scope: independently of channel mapped on ADC group regular or injected. In differential mode: Differential measurement is carried out between the selected channel (positive input) and another channel (negative input). Only selected channel has to be configured, the other channel is configured automatically.
- The selected channel and the other other channel have not necessarily contiguous channel number. To get the other channel number, refer to reference manual in section of ADC instance connectivity.
- Refer to Reference Manual to ensure the selected channel is available in differential mode. For example, internal channels (VrefInt, TempSensor, ...) are not available in differential mode.
- When configuring a channel 'inp' in differential mode, the channel 'inn' is not usable separately.
- Some channels are internally fixed to single-ended inputs configuration, refer to device datasheet for more details.
- For ADC channels configured in differential mode, both inputs should be biased at (Vref+)/2 +/-200mV. (Vref+ is the analog voltage reference)
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be ADC disabled.

Reference Manual to LL API cross reference:

- DIFSEL DIFSEL LL_ADC_SetChannelSingleDiff

LL_ADC_GetChannelSingleDiff

Function name

__STATIC_INLINE uint32_t LL_ADC_GetChannelSingleDiff (const ADC_TypeDef * ADCx, uint32_t Channel)

Function description

Get mode single-ended or differential input of the selected ADC channel.

Parameters

- **ADCx:** ADC instance
- **Channel:** This parameter can be a combination of the following values:
  - LL_ADC_CHANNEL_1
  - LL_ADC_CHANNEL_2
  - LL_ADC_CHANNEL_3
  - LL_ADC_CHANNEL_4
  - LL_ADC_CHANNEL_5
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_19

Return values

- **0:** channel in single-ended mode, else: channel in differential mode

Notes

- Channel ending is on channel scope: independently of channel mapped on ADC group regular or injected. In differential mode: Differential measurement is carried out between the selected channel (positive input) and another channel (negative input). Only selected channel has to be configured, the other channel is configured automatically.
- The selected channel and the other other channel have not necessarily contiguous channel number. To get the other channel number, refer to reference manual in section of ADC instance connectivity.
- Refer to Reference Manual to ensure the selected channel is available in differential mode. For example, internal channels (VrefInt, TempSensor, ...) are not available in differential mode.
- When configuring a channel 'i' in differential mode, the channel 'i+1' is not usable separately.
- Some channels are internally fixed to single-ended inputs configuration, refer to device datasheet for more details.

Reference Manual to LL API cross reference:

- DIFSEL DIFSEL LL_ADC_GetChannelSingleDiff

LL_ADC_SetAnalogWDMonitChannels

Function name

__STATIC_INLINE void LL_ADC_SetAnalogWDMonitChannels (ADC_TypeDef * ADCx, uint32_t AWDy, uint32_t AWDChannelGroup)

Function description

Set ADC analog watchdog monitored channels: a single channel, multiple channels or all channels, on ADC groups regular and-or injected.

Parameters

- **ADCx:** ADC instance
- **AWDy:** This parameter can be one of the following values:
  - LL_ADC_AWD1
  - LL_ADC_AWD2
  - LL_ADC_AWD3

- **AWDChannelGroup:** This parameter can be one of the following values:
  - LL_ADC_AWD_DISABLE
  - LL_ADC_AWD_ALL_CHANNELS_REG (0)
  - LL_ADC_AWD_ALL_CHANNELS_INJ (0)
  - LL_ADC_AWD_ALL_CHANNELS_REG_INJ
  - LL_ADC_AWD_CHANNEL_0_REG (0)
  - LL_ADC_AWD_CHANNEL_0_INJ (0)
  - LL_ADC_AWD_CHANNEL_0_REG_INJ
  - LL_ADC_AWD_CHANNEL_1_REG (0)
  - LL_ADC_AWD_CHANNEL_1_INJ (0)
  - LL_ADC_AWD_CHANNEL_1_REG_INJ
  - LL_ADC_AWD_CHANNEL_2_REG (0)
  - LL_ADC_AWD_CHANNEL_2_INJ (0)
  - LL_ADC_AWD_CHANNEL_2_REG_INJ
  - LL_ADC_AWD_CHANNEL_3_REG (0)
  - LL_ADC_AWD_CHANNEL_3_INJ (0)
  - LL_ADC_AWD_CHANNEL_3_REG_INJ
  - LL_ADC_AWD_CHANNEL_4_REG (0)
  - LL_ADC_AWD_CHANNEL_4_INJ (0)
  - LL_ADC_AWD_CHANNEL_4_REG_INJ
  - LL_ADC_AWD_CHANNEL_5_REG (0)
  - LL_ADC_AWD_CHANNEL_5_INJ (0)
  - LL_ADC_AWD_CHANNEL_5_REG_INJ
  - LL_ADC_AWD_CHANNEL_6_REG (0)
  - LL_ADC_AWD_CHANNEL_6_INJ (0)
  - LL_ADC_AWD_CHANNEL_6_REG_INJ
  - LL_ADC_AWD_CHANNEL_7_REG (0)
  - LL_ADC_AWD_CHANNEL_7_INJ (0)
  - LL_ADC_AWD_CHANNEL_7_REG_INJ
  - LL_ADC_AWD_CHANNEL_8_REG (0)
  - LL_ADC_AWD_CHANNEL_8_INJ (0)
  - LL_ADC_AWD_CHANNEL_8_REG_INJ
  - LL_ADC_AWD_CHANNEL_9_REG (0)
  - LL_ADC_AWD_CHANNEL_9_INJ (0)
  - LL_ADC_AWD_CHANNEL_9_REG_INJ
  - LL_ADC_AWD_CHANNEL_10_REG (0)
  - LL_ADC_AWD_CHANNEL_10_INJ (0)
  - LL_ADC_AWD_CHANNEL_10_REG_INJ
  - LL_ADC_AWD_CHANNEL_11_REG (0)
  - LL_ADC_AWD_CHANNEL_11_INJ (0)
  - LL_ADC_AWD_CHANNEL_11_REG_INJ
  - LL_ADC_AWD_CHANNEL_12_REG (0)
  - LL_ADC_AWD_CHANNEL_12_INJ (0)
  - LL_ADC_AWD_CHANNEL_12_REG_INJ
  - LL_ADC_AWD_CHANNEL_13_REG (0)
  - LL_ADC_AWD_CHANNEL_13_INJ (0)
  - LL_ADC_AWD_CHANNEL_13_REG_INJ
  - LL_ADC_AWD_CHANNEL_14_REG (0)
  - LL_ADC_AWD_CHANNEL_14_INJ (0)
  - LL_ADC_AWD_CHANNEL_14_REG_INJ

  - LL_ADC_AWD_CHANNEL_15_REG (0)
  - LL_ADC_AWD_CHANNEL_15_INJ (0)
  - LL_ADC_AWD_CHANNEL_15_REG_INJ
  - LL_ADC_AWD_CHANNEL_16_REG (0)
  - LL_ADC_AWD_CHANNEL_16_INJ (0)
  - LL_ADC_AWD_CHANNEL_16_REG_INJ
  - LL_ADC_AWD_CHANNEL_17_REG (0)
  - LL_ADC_AWD_CHANNEL_17_INJ (0)
  - LL_ADC_AWD_CHANNEL_17_REG_INJ
  - LL_ADC_AWD_CHANNEL_18_REG (0)
  - LL_ADC_AWD_CHANNEL_18_INJ (0)
  - LL_ADC_AWD_CHANNEL_18_REG_INJ
  - LL_ADC_AWD_CHANNEL_19_REG (0)
  - LL_ADC_AWD_CHANNEL_19_INJ (0)
  - LL_ADC_AWD_CHANNEL_19_REG_INJ
  - LL_ADC_AWD_CH_VREFINT_REG (0)
  - LL_ADC_AWD_CH_VREFINT_INJ (0)
  - LL_ADC_AWD_CH_VREFINT_REG_INJ
  - LL_ADC_AWD_CH_VBAT_REG (0)(1)
  - LL_ADC_AWD_CH_VBAT_INJ (0)(1)
  - LL_ADC_AWD_CH_VBAT_REG_INJ (1)
  - LL_ADC_AWD_CH_VDDCORE_REG (0)(1)
  - LL_ADC_AWD_CH_VDDCORE_INJ (0)(1)
  - LL_ADC_AWD_CH_VDDCORE_REG_INJ (1)

  (0) On this STM32 series, parameter available only on analog watchdog instance: AWD1.
- (1) On this STM32 series, parameter available only on ADC instance: ADC2, ADC3.

Return values

- **None:**

Notes

- Once monitored channels are selected, analog watchdog is enabled.
- In case of need to define a single channel to monitor with analog watchdog from sequencer channel definition, use helper macro __LL_ADC_ANALOGWD_CHANNEL_GROUP().
- On this STM32 series, there are 2 kinds of analog watchdog instance: AWD standard (instance AWD1): channels monitored: can monitor 1 channel or all channels.groups monitored: ADC groups regular and-or injected. AWD flexible (instances AWD2, AWD3): channels monitored: flexible on channels monitored, selection is channel wise, from from 1 to all channels. Specificity of this analog watchdog: Multiple channels can be selected. For example: (LL_ADC_AWD_CHANNEL4_REG_INJ | LL_ADC_AWD_CHANNEL5_REG_INJ | ...)groups monitored: not selection possible (monitoring on both groups regular and injected). Channels selected are monitored on groups regular and injected: LL_ADC_AWD_CHANNELxx_REG_INJ (do not use parameters LL_ADC_AWD_CHANNELxx_REG and LL_ADC_AWD_CHANNELxx_INJ)
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- CFGR1 AWD1CH LL_ADC_SetAnalogWDMonitChannels
- CFGR1 AWD1SGL LL_ADC_SetAnalogWDMonitChannels
- CFGR1 AWD1EN LL_ADC_SetAnalogWDMonitChannels
- CFGR1 JAWD1EN LL_ADC_SetAnalogWDMonitChannels
- AWD2CR AWD2CH LL_ADC_SetAnalogWDMonitChannels
- AWD3CR AWD3CH LL_ADC_SetAnalogWDMonitChannels

LL_ADC_GetAnalogWDMonitChannels

Function name

__STATIC_INLINE uint32_t LL_ADC_GetAnalogWDMonitChannels (const ADC_TypeDef * ADCx, uint32_t AWDy)

Function description

Get ADC analog watchdog monitored channel.

Parameters

- **ADCx:** ADC instance
- **AWDy:** This parameter can be one of the following values:
  - LL_ADC_AWD1
  - LL_ADC_AWD2 (1)
  - LL_ADC_AWD3 (1)

  (1) On this AWD number, monitored channel can be retrieved if only 1 channel is programmed (or none or all channels). This function cannot retrieve monitored channel if multiple channels are programmed simultaneously by bitfield.

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_AWD_DISABLE
  - LL_ADC_AWD_ALL_CHANNELS_REG (0)
  - LL_ADC_AWD_ALL_CHANNELS_INJ (0)
  - LL_ADC_AWD_ALL_CHANNELS_REG_INJ
  - LL_ADC_AWD_CHANNEL_0_REG (0)
  - LL_ADC_AWD_CHANNEL_0_INJ (0)
  - LL_ADC_AWD_CHANNEL_0_REG_INJ
  - LL_ADC_AWD_CHANNEL_1_REG (0)
  - LL_ADC_AWD_CHANNEL_1_INJ (0)
  - LL_ADC_AWD_CHANNEL_1_REG_INJ
  - LL_ADC_AWD_CHANNEL_2_REG (0)
  - LL_ADC_AWD_CHANNEL_2_INJ (0)
  - LL_ADC_AWD_CHANNEL_2_REG_INJ
  - LL_ADC_AWD_CHANNEL_3_REG (0)
  - LL_ADC_AWD_CHANNEL_3_INJ (0)
  - LL_ADC_AWD_CHANNEL_3_REG_INJ
  - LL_ADC_AWD_CHANNEL_4_REG (0)
  - LL_ADC_AWD_CHANNEL_4_INJ (0)
  - LL_ADC_AWD_CHANNEL_4_REG_INJ
  - LL_ADC_AWD_CHANNEL_5_REG (0)
  - LL_ADC_AWD_CHANNEL_5_INJ (0)
  - LL_ADC_AWD_CHANNEL_5_REG_INJ
  - LL_ADC_AWD_CHANNEL_6_REG (0)
  - LL_ADC_AWD_CHANNEL_6_INJ (0)
  - LL_ADC_AWD_CHANNEL_6_REG_INJ
  - LL_ADC_AWD_CHANNEL_7_REG (0)
  - LL_ADC_AWD_CHANNEL_7_INJ (0)
  - LL_ADC_AWD_CHANNEL_7_REG_INJ
  - LL_ADC_AWD_CHANNEL_8_REG (0)
  - LL_ADC_AWD_CHANNEL_8_INJ (0)
  - LL_ADC_AWD_CHANNEL_8_REG_INJ
  - LL_ADC_AWD_CHANNEL_9_REG (0)
  - LL_ADC_AWD_CHANNEL_9_INJ (0)
  - LL_ADC_AWD_CHANNEL_9_REG_INJ
  - LL_ADC_AWD_CHANNEL_10_REG (0)
  - LL_ADC_AWD_CHANNEL_10_INJ (0)
  - LL_ADC_AWD_CHANNEL_10_REG_INJ
  - LL_ADC_AWD_CHANNEL_11_REG (0)
  - LL_ADC_AWD_CHANNEL_11_INJ (0)
  - LL_ADC_AWD_CHANNEL_11_REG_INJ
  - LL_ADC_AWD_CHANNEL_12_REG (0)
  - LL_ADC_AWD_CHANNEL_12_INJ (0)
  - LL_ADC_AWD_CHANNEL_12_REG_INJ
  - LL_ADC_AWD_CHANNEL_13_REG (0)
  - LL_ADC_AWD_CHANNEL_13_INJ (0)
  - LL_ADC_AWD_CHANNEL_13_REG_INJ
  - LL_ADC_AWD_CHANNEL_14_REG (0)
  - LL_ADC_AWD_CHANNEL_14_INJ (0)

- LL_ADC_AWD_CHANNEL_14_REG_INJ
- LL_ADC_AWD_CHANNEL_15_REG (0)
- LL_ADC_AWD_CHANNEL_15_INJ (0)
- LL_ADC_AWD_CHANNEL_15_REG_INJ
- LL_ADC_AWD_CHANNEL_16_REG (0)
- LL_ADC_AWD_CHANNEL_16_INJ (0)
- LL_ADC_AWD_CHANNEL_16_REG_INJ
- LL_ADC_AWD_CHANNEL_17_REG (0)
- LL_ADC_AWD_CHANNEL_17_INJ (0)
- LL_ADC_AWD_CHANNEL_17_REG_INJ
- LL_ADC_AWD_CHANNEL_18_REG (0)
- LL_ADC_AWD_CHANNEL_18_INJ (0)
- LL_ADC_AWD_CHANNEL_18_REG_INJ
- LL_ADC_AWD_CHANNEL_19_REG (0)
- LL_ADC_AWD_CHANNEL_19_INJ (0)
- LL_ADC_AWD_CHANNEL_19_REG_INJ

(0) On this STM32 series, parameter available only on analog watchdog number: AWD1.

Notes

- Usage of the returned channel number: To reinject this channel into another function LL_ADC_xxx: the returned channel number is only partly formatted on definition of literals LL_ADC_CHANNEL_x. Therefore, it has to be compared with parts of literals LL_ADC_CHANNEL_x or using helper macro __LL_ADC_CHANNEL_TO_DECIMAL_NB(). Then the selected literal LL_ADC_CHANNEL_x can be used as parameter for another function.To get the channel number in decimal format: process the returned value with the helper macro __LL_ADC_CHANNEL_TO_DECIMAL_NB(). Applicable only when the analog watchdog is set to monitor one channel.
- On this STM32 series, there are 2 kinds of analog watchdog instance: AWD standard (instance AWD1): channels monitored: can monitor 1 channel or all channels.groups monitored: ADC groups regular and-or injected. AWD flexible (instances AWD2, AWD3): channels monitored: flexible on channels monitored, selection is channel wise, from from 1 to all channels. Specificity of this analog watchdog: Multiple channels can be selected. For example: (LL_ADC_AWD_CHANNEL4_REG_INJ | LL_ADC_AWD_CHANNEL5_REG_INJ | ...)groups monitored: not selection possible (monitoring on both groups regular and injected). Channels selected are monitored on groups regular and injected: LL_ADC_AWD_CHANNELxx_REG_INJ (do not use parameters LL_ADC_AWD_CHANNELxx_REG and LL_ADC_AWD_CHANNELxx_INJ)
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- CFGR1 AWD1CH LL_ADC_GetAnalogWDMonitChannels
- CFGR1 AWD1SGL LL_ADC_GetAnalogWDMonitChannels
- CFGR1 AWD1EN LL_ADC_GetAnalogWDMonitChannels
- CFGR1 JAWD1EN LL_ADC_GetAnalogWDMonitChannels
- AWD2CR AWD2CH LL_ADC_GetAnalogWDMonitChannels
- AWD3CR AWD3CH LL_ADC_GetAnalogWDMonitChannels

LL_ADC_SetAnalogWDThresholds

Function name

__STATIC_INLINE void LL_ADC_SetAnalogWDThresholds (ADC_TypeDef * ADCx, uint32_t AWDy, uint32_t AWDThresholdsHighLow, uint32_t AWDThresholdValue)

Function description

Set ADC analog watchdog threshold value of threshold high or low.

Parameters

- **ADCx:** ADC instance
- **AWDy:** This parameter can be one of the following values:
  - LL_ADC_AWD1
  - LL_ADC_AWD2
  - LL_ADC_AWD3
- **AWDThresholdsHighLow:** This parameter can be one of the following values:
  - LL_ADC_AWD_THRESHOLD_HIGH
  - LL_ADC_AWD_THRESHOLD_LOW
- **AWDThresholdValue:** Value between Min_Data=0x000 and Max_Data=0xFFF

Return values

- **None:**

Notes

- In case of ADC resolution different of 12 bits, specific threshold data formatting is required: analog watchdog thresholds data must be aligned to left side (bit 11).bits out of resolution range (LSB) must be set to value "0". To perform this formatting automatically, use helper macro __LL_ADC_ANALOGWD_SET_THRESHOLD_RESOLUTION().
- On this STM32 series, there are 2 kinds of analog watchdog instance: AWD standard (instance AWD1): channels monitored: can monitor 1 channel or all channels.groups monitored: ADC groups regular and-or injected.resolution: resolution is not limited (corresponds to ADC resolution configured). AWD flexible (instances AWD2, AWD3): channels monitored: flexible on channels monitored, selection is channel wise, from from 1 to all channels. Specificity of this analog watchdog: Multiple channels can be selected. For example: (LL_ADC_AWD_CHANNEL4_REG_INJ | LL_ADC_AWD_CHANNEL5_REG_INJ | ...)groups monitored: not selection possible (monitoring on both groups regular and injected). Channels selected are monitored on groups regular and injected: LL_ADC_AWD_CHANNELxx_REG_INJ (do not use parameters LL_ADC_AWD_CHANNELxx_REG and LL_ADC_AWD_CHANNELxx_INJ)
- If ADC oversampling is enabled, ADC analog watchdog thresholds are impacted: the comparison of analog watchdog thresholds is done on oversampling intermediate computation (after ratio, before shift application): intermediate register bitfield [32:7] (26 most significant bits).
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC can be disabled, enabled with or without conversion on going on either ADC groups regular or injected.

Reference Manual to LL API cross reference:

- AWD1LTR LTR LL_ADC_SetAnalogWDThresholds
- AWD1HTR HTR LL_ADC_SetAnalogWDThresholds
- AWD2LTR LTR LL_ADC_SetAnalogWDThresholds
- AWD2HTR HTR LL_ADC_SetAnalogWDThresholds
- AWD3LTR LTR LL_ADC_SetAnalogWDThresholds
- AWD3HTR HTR LL_ADC_SetAnalogWDThresholds

LL_ADC_GetAnalogWDThresholds

Function name

__STATIC_INLINE uint32_t LL_ADC_GetAnalogWDThresholds (const ADC_TypeDef * ADCx, uint32_t AWDy, uint32_t AWDThresholdsHighLow)

Function description

Get ADC analog watchdog threshold value of threshold high, threshold low or raw data with ADC thresholds high and low concatenated.

Parameters

- **ADCx:** ADC instance
- **AWDy:** This parameter can be one of the following values:
  - LL_ADC_AWD1
  - LL_ADC_AWD2
  - LL_ADC_AWD3
- **AWDThresholdsHighLow:** This parameter can be one of the following values:
  - LL_ADC_AWD_THRESHOLD_HIGH
  - LL_ADC_AWD_THRESHOLD_LOW

Return values

- **Value:** between Min_Data=0x000 and Max_Data=0xFFF

Notes

- In case of ADC resolution different of 12 bits, specific threshold data formatting is required: analog watchdog thresholds data must be aligned to left side (bit 11).bits out of resolution range (LSB) must be set to value "0". To perform this formatting automatically, use helper macro __LL_ADC_ANALOGWD_SET_THRESHOLD_RESOLUTION().

Reference Manual to LL API cross reference:

- AWD1LTR LTR LL_ADC_GetAnalogWDThresholds
- AWD1HTR HTR LL_ADC_GetAnalogWDThresholds
- AWD2LTR LTR LL_ADC_GetAnalogWDThresholds
- AWD2HTR HTR LL_ADC_GetAnalogWDThresholds
- AWD3LTR LTR LL_ADC_GetAnalogWDThresholds
- AWD3HTR HTR LL_ADC_GetAnalogWDThresholds

LL_ADC_SetAWDFilteringConfiguration

Function name

__STATIC_INLINE void LL_ADC_SetAWDFilteringConfiguration (ADC_TypeDef * ADCx, uint32_t AWDy, uint32_t FilteringConfig)

Function description

Set ADC analog watchdog filtering configuration.

Parameters

- **ADCx:** ADC instance
- **AWDy:** This parameter can be one of the following values:
  - LL_ADC_AWD1
- **FilteringConfig:** This parameter can be one of the following values:
  - LL_ADC_AWD_FILTERING_NONE
  - LL_ADC_AWD_FILTERING_2SAMPLES
  - LL_ADC_AWD_FILTERING_3SAMPLES
  - LL_ADC_AWD_FILTERING_4SAMPLES
  - LL_ADC_AWD_FILTERING_5SAMPLES
  - LL_ADC_AWD_FILTERING_6SAMPLES
  - LL_ADC_AWD_FILTERING_7SAMPLES
  - LL_ADC_AWD_FILTERING_8SAMPLES

Return values

- **None:**

Notes

- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.
- On this STM32 series, this feature is only available on first analog watchdog (AWD1)

Reference Manual to LL API cross reference:

- AWD1HTR AWDFILT LL_ADC_SetAWDFilteringConfiguration

LL_ADC_GetAWDFilteringConfiguration

Function name

__STATIC_INLINE uint32_t LL_ADC_GetAWDFilteringConfiguration (const ADC_TypeDef * ADCx, uint32_t AWDy)

Function description

Get ADC analog watchdog filtering configuration.

Parameters

- **ADCx:** ADC instance
- **AWDy:** This parameter can be one of the following values:
  - LL_ADC_AWD1

Return values

- **Returned:** value can be:
  - LL_ADC_AWD_FILTERING_NONE
  - LL_ADC_AWD_FILTERING_2SAMPLES
  - LL_ADC_AWD_FILTERING_3SAMPLES
  - LL_ADC_AWD_FILTERING_4SAMPLES
  - LL_ADC_AWD_FILTERING_5SAMPLES
  - LL_ADC_AWD_FILTERING_6SAMPLES
  - LL_ADC_AWD_FILTERING_7SAMPLES
  - LL_ADC_AWD_FILTERING_8SAMPLES

Notes

- On this STM32 series, this feature is only available on first analog watchdog (AWD1)

Reference Manual to LL API cross reference:

- AWD1HTR AWDFILT LL_ADC_GetAWDFilteringConfiguration

LL_ADC_SetOverSamplingScope

Function name

__STATIC_INLINE void LL_ADC_SetOverSamplingScope (ADC_TypeDef * ADCx, uint32_t OvsScope)

Function description

Set ADC oversampling scope: ADC groups regular and-or injected (availability of ADC group injected depends on STM32 series).

Parameters

- **ADCx:** ADC instance
- **OvsScope:** This parameter can be one of the following values:
  - LL_ADC_OVS_DISABLE
  - LL_ADC_OVS_GRP_REGULAR_CONTINUED
  - LL_ADC_OVS_GRP_REGULAR_RESUMED
  - LL_ADC_OVS_GRP_INJECTED
  - LL_ADC_OVS_GRP_INJ_REG_RESUMED

Return values

- **None:**

Notes

- If both groups regular and injected are selected, specify behavior of ADC group injected interrupting group regular: when ADC group injected is triggered, the oversampling on ADC group regular is either temporary stopped and continued, or resumed from start (oversampler buffer reset).
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- CFGR2 ROVSE LL_ADC_SetOverSamplingScope
- CFGR2 JOVSE LL_ADC_SetOverSamplingScope
- CFGR2 ROVSM LL_ADC_SetOverSamplingScope

LL_ADC_GetOverSamplingScope

Function name

__STATIC_INLINE uint32_t LL_ADC_GetOverSamplingScope (const ADC_TypeDef * ADCx)

Function description

Get ADC oversampling scope: ADC groups regular and-or injected (availability of ADC group injected depends on STM32 series).

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_OVS_DISABLE
  - LL_ADC_OVS_GRP_REGULAR_CONTINUED
  - LL_ADC_OVS_GRP_REGULAR_RESUMED
  - LL_ADC_OVS_GRP_INJECTED
  - LL_ADC_OVS_GRP_INJ_REG_RESUMED

Notes

- If both groups regular and injected are selected, specify behavior of ADC group injected interrupting group regular: when ADC group injected is triggered, the oversampling on ADC group regular is either temporary stopped and continued, or resumed from start (oversampler buffer reset).

Reference Manual to LL API cross reference:

- CFGR2 ROVSE LL_ADC_GetOverSamplingScope
- CFGR2 JOVSE LL_ADC_GetOverSamplingScope
- CFGR2 ROVSM LL_ADC_GetOverSamplingScope

LL_ADC_SetOverSamplingDiscont

Function name

__STATIC_INLINE void LL_ADC_SetOverSamplingDiscont (ADC_TypeDef * ADCx, uint32_t OverSamplingDiscont)

Function description

Set ADC oversampling discontinuous mode (triggered mode) on the selected ADC group.

Parameters

- **ADCx:** ADC instance
- **OverSamplingDiscont:** This parameter can be one of the following values:
  - LL_ADC_OVS_REG_CONT
  - LL_ADC_OVS_REG_DISCONT

Return values

- **None:**

Notes

- Number of oversampled conversions are done either in: continuous mode (all conversions of oversampling ratio are done from 1 trigger)discontinuous mode (each conversion of oversampling ratio needs a trigger)
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.
- On this STM32 series, oversampling discontinuous mode (triggered mode) can be used only when oversampling is set on group regular only and in resumed mode.

Reference Manual to LL API cross reference:

- CFGR2 TROVS LL_ADC_SetOverSamplingDiscont

LL_ADC_GetOverSamplingDiscont

Function name

__STATIC_INLINE uint32_t LL_ADC_GetOverSamplingDiscont (const ADC_TypeDef * ADCx)

Function description

Get ADC oversampling discontinuous mode (triggered mode) on the selected ADC group.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_OVS_REG_CONT
  - LL_ADC_OVS_REG_DISCONT

Notes

- Number of oversampled conversions are done either in: continuous mode (all conversions of oversampling ratio are done from 1 trigger)discontinuous mode (each conversion of oversampling ratio needs a trigger)

Reference Manual to LL API cross reference:

- CFGR2 TROVS LL_ADC_GetOverSamplingDiscont

LL_ADC_ConfigOverSamplingRatioShift

Function name

__STATIC_INLINE void LL_ADC_ConfigOverSamplingRatioShift (ADC_TypeDef * ADCx, uint32_t Ratio, uint32_t Shift)

Function description

Set ADC oversampling (impacting both ADC groups regular and injected)

Parameters

- **ADCx:** ADC instance
- **Ratio:** This parameter can be in the range from 1 to 1024
- **Shift:** This parameter can be one of the following values:
  - LL_ADC_OVS_SHIFT_NONE
  - LL_ADC_OVS_SHIFT_RIGHT_1
  - LL_ADC_OVS_SHIFT_RIGHT_2
  - LL_ADC_OVS_SHIFT_RIGHT_3
  - LL_ADC_OVS_SHIFT_RIGHT_4
  - LL_ADC_OVS_SHIFT_RIGHT_5
  - LL_ADC_OVS_SHIFT_RIGHT_6
  - LL_ADC_OVS_SHIFT_RIGHT_7
  - LL_ADC_OVS_SHIFT_RIGHT_8
  - LL_ADC_OVS_SHIFT_RIGHT_9
  - LL_ADC_OVS_SHIFT_RIGHT_10

Return values

- **None:**

Notes

- This function set the 2 items of oversampling configuration: ratioshift
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- CFGR2 OVSS LL_ADC_ConfigOverSamplingRatioShift
- CFGR2 OVSR LL_ADC_ConfigOverSamplingRatioShift

LL_ADC_GetOverSamplingRatio

Function name

__STATIC_INLINE uint32_t LL_ADC_GetOverSamplingRatio (const ADC_TypeDef * ADCx)

Function description

Get ADC oversampling ratio (impacting both ADC groups regular and injected)

Parameters

- **ADCx:** ADC instance

Return values

- **Ratio:** This parameter can be in the from 1 to 1024.

Reference Manual to LL API cross reference:

- CFGR2 OVSR LL_ADC_GetOverSamplingRatio

LL_ADC_GetOverSamplingShift

Function name

__STATIC_INLINE uint32_t LL_ADC_GetOverSamplingShift (const ADC_TypeDef * ADCx)

Function description

Get ADC oversampling shift (impacting both ADC groups regular and injected)

Parameters

- **ADCx:** ADC instance

Return values

- **Shift:** This parameter can be one of the following values:
  - LL_ADC_OVS_SHIFT_NONE
  - LL_ADC_OVS_SHIFT_RIGHT_1
  - LL_ADC_OVS_SHIFT_RIGHT_2
  - LL_ADC_OVS_SHIFT_RIGHT_3
  - LL_ADC_OVS_SHIFT_RIGHT_4
  - LL_ADC_OVS_SHIFT_RIGHT_5
  - LL_ADC_OVS_SHIFT_RIGHT_6
  - LL_ADC_OVS_SHIFT_RIGHT_7
  - LL_ADC_OVS_SHIFT_RIGHT_8
  - LL_ADC_OVS_SHIFT_RIGHT_9
  - LL_ADC_OVS_SHIFT_RIGHT_10

Reference Manual to LL API cross reference:

- CFGR2 OVSS LL_ADC_GetOverSamplingShift

LL_ADC_SetMultimode

Function name

__STATIC_INLINE void LL_ADC_SetMultimode (ADC_Common_TypeDef * ADCxy_COMMON, uint32_t Multimode)

Function description

Set ADC multimode configuration to operate in independent mode or multimode (for devices with several ADC instances).

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())
- **Multimode:** This parameter can be one of the following values:
  - LL_ADC_MULTI_INDEPENDENT
  - LL_ADC_MULTI_DUAL_REG_SIMULT
  - LL_ADC_MULTI_DUAL_REG_INTERL
  - LL_ADC_MULTI_DUAL_INJ_SIMULT
  - LL_ADC_MULTI_DUAL_INJ_ALTERN
  - LL_ADC_MULTI_DUAL_REG_SIM_INJ_SIM
  - LL_ADC_MULTI_DUAL_REG_SIM_INJ_ALT
  - LL_ADC_MULTI_DUAL_REG_INT_INJ_SIM

Return values

- **None:**

Notes

- If multimode configuration: the selected ADC instance is either master or slave depending on hardware. Refer to reference manual.
- On this STM32 series, setting of this feature is conditioned to ADC state: All ADC instances of the ADC common group must be disabled. This check can be done with function LL_ADC_IsEnabled() for each ADC instance or by using helper macro __LL_ADC_IS_ENABLED_ALL_COMMON_INSTANCE().

Reference Manual to LL API cross reference:

- CCR DUAL LL_ADC_SetMultimode

LL_ADC_GetMultimode

Function name

__STATIC_INLINE uint32_t LL_ADC_GetMultimode (const ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get ADC multimode configuration to operate in independent mode or multimode (for devices with several ADC instances).

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_MULTI_INDEPENDENT
  - LL_ADC_MULTI_DUAL_REG_SIMULT
  - LL_ADC_MULTI_DUAL_REG_INTERL
  - LL_ADC_MULTI_DUAL_INJ_SIMULT
  - LL_ADC_MULTI_DUAL_INJ_ALTERN
  - LL_ADC_MULTI_DUAL_REG_SIM_INJ_SIM
  - LL_ADC_MULTI_DUAL_REG_SIM_INJ_ALT
  - LL_ADC_MULTI_DUAL_REG_INT_INJ_SIM

Notes

- If multimode configuration: the selected ADC instance is either master or slave depending on hardware. Refer to reference manual.

Reference Manual to LL API cross reference:

- CCR DUAL LL_ADC_GetMultimode

LL_ADC_SetMultiDataFormat

Function name

__STATIC_INLINE void LL_ADC_SetMultiDataFormat (ADC_Common_TypeDef * ADCxy_COMMON, uint32_t MultiDataFormat)

Function description

Set ADC multimode conversion data format: conversion data in data register of each ADC instance or ADC common instance.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())
- **MultiDataFormat:** This parameter can be one of the following values:
  - LL_ADC_MULTI_REG_DATA_EACH_ADC
  - LL_ADC_MULTI_REG_DATA_COMMON_32B
  - LL_ADC_MULTI_REG_DATA_COMMON_16B

Return values

- **None:**

Notes

- If ADC data transfer by DMA is used: data register as DMA source and number of DMA requests is impacted, refer to description of parameters of this function.
- How to retrieve multimode conversion data: For multimode data transfer setting 16 bit resolution or less, using functions LL_ADC_REG_ReadMultiConvPacking() or LL_ADC_REG_ReadMultiConvNoPacking(). Conversion data is a raw data with ADC master and slave concatenated. A macro is available to get the conversion data of ADC master or ADC slave: see helper macro __LL_ADC_MULTI_CONV_DATA_MASTER_SLAVE().
- On this STM32 series, setting of this feature is conditioned to ADC state: All ADC instances of the ADC common group must be disabled or enabled without conversion on going on group regular.

Reference Manual to LL API cross reference:

- CCR DAMDF LL_ADC_SetMultiDataFormat

LL_ADC_GetMultiDataFormat

Function name

__STATIC_INLINE uint32_t LL_ADC_GetMultiDataFormat (const ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get ADC multimode conversion data format: conversion data in data register of each ADC instance or ADC common instance.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_MULTI_REG_DATA_EACH_ADC
  - LL_ADC_MULTI_REG_DATA_COMMON_32B
  - LL_ADC_MULTI_REG_DATA_COMMON_16B

Notes

- If ADC data transfer by DMA is used: data register as DMA source and number of DMA requests is impacted, refer to description of parameters of this function.
- How to retrieve multimode conversion data: For multimode data transfer setting 16 bit resolution or less, using functions LL_ADC_REG_ReadMultiConvPacking() or LL_ADC_REG_ReadMultiConvNoPacking(). Conversion data is a raw data with ADC master and slave concatenated. A macro is available to get the conversion data of ADC master or ADC slave: see helper macro __LL_ADC_MULTI_CONV_DATA_MASTER_SLAVE().

Reference Manual to LL API cross reference:

- CCR DAMDF LL_ADC_GetMultiDataFormat

LL_ADC_SetMultiTwoSamplingDelay

Function name

__STATIC_INLINE void LL_ADC_SetMultiTwoSamplingDelay (ADC_Common_TypeDef * ADCxy_COMMON, uint32_t MultiTwoSamplingDelay)

Function description

Set ADC multimode delay between 2 sampling phases.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())
- **MultiTwoSamplingDelay:** This parameter can be one of the following values:
  - LL_ADC_MULTI_TWOSMP_DELAY_1CYCLE
  - LL_ADC_MULTI_TWOSMP_DELAY_2CYCLES
  - LL_ADC_MULTI_TWOSMP_DELAY_3CYCLES
  - LL_ADC_MULTI_TWOSMP_DELAY_4CYCLES
  - LL_ADC_MULTI_TWOSMP_DELAY_5CYCLES
  - LL_ADC_MULTI_TWOSMP_DELAY_6CYCLES
  - LL_ADC_MULTI_TWOSMP_DELAY_7CYCLES
  - LL_ADC_MULTI_TWOSMP_DELAY_8CYCLES
  - LL_ADC_MULTI_TWOSMP_DELAY_9CYCLES
  - LL_ADC_MULTI_TWOSMP_DELAY_10CYCLES
  - LL_ADC_MULTI_TWOSMP_DELAY_11CYCLES
  - LL_ADC_MULTI_TWOSMP_DELAY_12CYCLES
  - LL_ADC_MULTI_TWOSMP_DELAY_13CYCLES

Return values

- **None:**

Notes

- The sampling delay range depends on ADC resolution: ADC resolution 12 bits can have maximum delay of 12 cycles.ADC resolution 10 bits can have maximum delay of 10 cycles.ADC resolution 8 bits can have maximum delay of 8 cycles.ADC resolution 6 bits can have maximum delay of 6 cycles.
- On this STM32 series, setting of this feature is conditioned to ADC state: All ADC instances of the ADC common group must be disabled. This check can be done with function LL_ADC_IsEnabled() for each ADC instance or by using helper macro helper macro __LL_ADC_IS_ENABLED_ALL_COMMON_INSTANCE().

Reference Manual to LL API cross reference:

- CCR DELAY LL_ADC_SetMultiTwoSamplingDelay

LL_ADC_GetMultiTwoSamplingDelay

Function name

__STATIC_INLINE uint32_t LL_ADC_GetMultiTwoSamplingDelay (const ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get ADC multimode delay between 2 sampling phases.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_MULTI_TWOSMP_DELAY_1CYCLE
  - LL_ADC_MULTI_TWOSMP_DELAY_2CYCLES
  - LL_ADC_MULTI_TWOSMP_DELAY_3CYCLES
  - LL_ADC_MULTI_TWOSMP_DELAY_4CYCLES
  - LL_ADC_MULTI_TWOSMP_DELAY_5CYCLES
  - LL_ADC_MULTI_TWOSMP_DELAY_6CYCLES
  - LL_ADC_MULTI_TWOSMP_DELAY_7CYCLES
  - LL_ADC_MULTI_TWOSMP_DELAY_8CYCLES
  - LL_ADC_MULTI_TWOSMP_DELAY_9CYCLES
  - LL_ADC_MULTI_TWOSMP_DELAY_10CYCLES
  - LL_ADC_MULTI_TWOSMP_DELAY_11CYCLES
  - LL_ADC_MULTI_TWOSMP_DELAY_12CYCLES
  - LL_ADC_MULTI_TWOSMP_DELAY_13CYCLES

Reference Manual to LL API cross reference:

- CCR DELAY LL_ADC_GetMultiTwoSamplingDelay

LL_ADC_EnableDeepPowerDown

Function name

__STATIC_INLINE void LL_ADC_EnableDeepPowerDown (ADC_TypeDef * ADCx)

Function description

Put ADC instance in deep power down state.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Notes

- In case of ADC calibration necessary: When ADC is in deep-power-down state, the internal analog calibration is lost. After exiting from deep power down, calibration must be relaunched or calibration factor (preliminarily saved) must be set back into calibration register.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be ADC disabled.

Reference Manual to LL API cross reference:

- CR DEEPPWD LL_ADC_EnableDeepPowerDown

LL_ADC_DisableDeepPowerDown

Function name

__STATIC_INLINE void LL_ADC_DisableDeepPowerDown (ADC_TypeDef * ADCx)

Function description

Disable ADC deep power down mode.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Notes

- In case of ADC calibration necessary: When ADC is in deep-power-down state, the internal analog calibration is lost. After exiting from deep power down, calibration must be relaunched or calibration factor (preliminarily saved) must be set back into calibration register.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be ADC disabled.

Reference Manual to LL API cross reference:

- CR DEEPPWD LL_ADC_DisableDeepPowerDown

LL_ADC_IsDeepPowerDownEnabled

Function name

__STATIC_INLINE uint32_t LL_ADC_IsDeepPowerDownEnabled (const ADC_TypeDef * ADCx)

Function description

Get the selected ADC instance deep power down state.

Parameters

- **ADCx:** ADC instance

Return values

- **0:** deep power down is disabled, 1: deep power down is enabled.

Reference Manual to LL API cross reference:

- CR DEEPPWD LL_ADC_IsDeepPowerDownEnabled

LL_ADC_Enable

Function name

__STATIC_INLINE void LL_ADC_Enable (ADC_TypeDef * ADCx)

Function description

Enable the selected ADC instance.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Notes

- On this STM32 series, after ADC enable, a delay for ADC internal analog stabilization is required before performing a ADC conversion start. Refer to device datasheet, parameter tSTAB.
- On this STM32 series, flag LL_ADC_FLAG_ADRDY is raised when the ADC is enabled and when conversion clock is active. (not only core clock: this ADC has a dual clock domain)
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be ADC disabled and ADC internal voltage regulator enabled.

Reference Manual to LL API cross reference:

- CR ADEN LL_ADC_Enable

LL_ADC_Disable

Function name

__STATIC_INLINE void LL_ADC_Disable (ADC_TypeDef * ADCx)

Function description

Disable the selected ADC instance.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Notes

- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be not disabled. Must be enabled without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- CR ADDIS LL_ADC_Disable

LL_ADC_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_ADC_IsEnabled (const ADC_TypeDef * ADCx)

Function description

Get the selected ADC instance enable state.

Parameters

- **ADCx:** ADC instance

Return values

- **0:** ADC is disabled, 1: ADC is enabled.

Notes

- On this STM32 series, flag LL_ADC_FLAG_ADRDY is raised when the ADC is enabled and when conversion clock is active. (not only core clock: this ADC has a dual clock domain)

Reference Manual to LL API cross reference:

- CR ADEN LL_ADC_IsEnabled

LL_ADC_IsDisableOngoing

Function name

__STATIC_INLINE uint32_t LL_ADC_IsDisableOngoing (const ADC_TypeDef * ADCx)

Function description

Get the selected ADC instance disable state.

Parameters

- **ADCx:** ADC instance

Return values

- **0:** no ADC disable command on going.

Reference Manual to LL API cross reference:

- CR ADDIS LL_ADC_IsDisableOngoing

LL_ADC_StartCalibration

Function name

__STATIC_INLINE void LL_ADC_StartCalibration (ADC_TypeDef * ADCx, uint32_t SingleDiff)

Function description

Start ADC calibration in the mode single-ended or differential (for devices with differential mode available).

Parameters

- **ADCx:** ADC instance
- **SingleDiff:** This parameter can be one of the following values:
  - LL_ADC_SINGLE_ENDED
  - LL_ADC_DIFFERENTIAL_ENDED

Return values

- **None:**

Notes

- On this STM32 series, a minimum number of ADC clock cycles are required between ADC end of calibration and ADC enable. Refer to literal LL_ADC_DELAY_CALIB_ENABLE_ADC_CYCLES.
- For devices with differential mode available: Calibration of offset is specific to each of single-ended and differential modes (calibration run must be performed for each of these differential modes, if used afterwards and if the application requires their calibration).
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be ADC disabled.

Reference Manual to LL API cross reference:

- CR ADCAL LL_ADC_StartCalibration
- CR ADCALDIF LL_ADC_StartCalibration

LL_ADC_IsCalibrationOnGoing

Function name

__STATIC_INLINE uint32_t LL_ADC_IsCalibrationOnGoing (const ADC_TypeDef * ADCx)

Function description

Get ADC calibration state.

Parameters

- **ADCx:** ADC instance

Return values

- **0:** calibration complete, 1: calibration in progress.

Reference Manual to LL API cross reference:

- CR ADCAL LL_ADC_IsCalibrationOnGoing

LL_ADC_StopCalibration

Function name

__STATIC_INLINE void LL_ADC_StopCalibration (ADC_TypeDef * ADCx)

Function description

Stop ADC calibration.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Notes

- On this STM32 series, a minimum number of ADC clock cycles are required between ADC end of calibration and ADC enable. Refer to literal LL_ADC_DELAY_CALIB_ENABLE_ADC_CYCLES.

Reference Manual to LL API cross reference:

- CR ADCAL LL_ADC_StopCalibration
- CR ADCALDIF LL_ADC_StopCalibration

LL_ADC_EnableCalibrationOffset

Function name

__STATIC_INLINE void LL_ADC_EnableCalibrationOffset (ADC_TypeDef * ADCx)

Function description

Enable calibration additional offset.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CALFACT CALADDOS LL_ADC_EnableCalibrationOffset

LL_ADC_DisableCalibrationOffset

Function name

__STATIC_INLINE void LL_ADC_DisableCalibrationOffset (ADC_TypeDef * ADCx)

Function description

Disable calibration additional offset.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CALFACT CALADDOS LL_ADC_DisableCalibrationOffset

LL_ADC_IsCalibrationOffsetEnabled

Function name

__STATIC_INLINE uint32_t LL_ADC_IsCalibrationOffsetEnabled (const ADC_TypeDef * ADCx)

Function description

Get calibration additional offset state.

Parameters

- **ADCx:** ADC instance

Return values

- **0:** calibration offset disabled, 1: calibration offset enabled.

Reference Manual to LL API cross reference:

- CALFACT CALADDOS LL_ADC_IsCalibrationOnGoing

LL_ADC_REG_StartConversion

Function name

__STATIC_INLINE void LL_ADC_REG_StartConversion (ADC_TypeDef * ADCx)

Function description

Start ADC group regular conversion.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Notes

- On this STM32 series, this function is relevant for both internal trigger (SW start) and external trigger: If ADC trigger has been set to software start, ADC conversion starts immediately.If ADC trigger has been set to external trigger, ADC conversion will start at next trigger event (on the selected trigger edge) following the ADC start conversion command.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be enabled without conversion on going on group regular, without conversion stop command on going on group regular, without ADC disable command on going.

Reference Manual to LL API cross reference:

- CR ADSTART LL_ADC_REG_StartConversion

LL_ADC_REG_StopConversion

Function name

__STATIC_INLINE void LL_ADC_REG_StopConversion (ADC_TypeDef * ADCx)

Function description

Stop ADC group regular conversion.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Notes

- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be enabled with conversion on going on group regular, without ADC disable command on going.

Reference Manual to LL API cross reference:

- CR ADSTP LL_ADC_REG_StopConversion

LL_ADC_REG_IsConversionOngoing

Function name

__STATIC_INLINE uint32_t LL_ADC_REG_IsConversionOngoing (const ADC_TypeDef * ADCx)

Function description

Get ADC group regular conversion state.

Parameters

- **ADCx:** ADC instance

Return values

- **0:** no conversion is on going on ADC group regular.

Reference Manual to LL API cross reference:

- CR ADSTART LL_ADC_REG_IsConversionOngoing

LL_ADC_REG_IsStopConversionOngoing

Function name

__STATIC_INLINE uint32_t LL_ADC_REG_IsStopConversionOngoing (const ADC_TypeDef * ADCx)

Function description

Get ADC group regular command of conversion stop state.

Parameters

- **ADCx:** ADC instance

Return values

- **0:** no command of conversion stop is on going on ADC group regular.

Reference Manual to LL API cross reference:

- CR ADSTP LL_ADC_REG_IsStopConversionOngoing

LL_ADC_REG_ReadConversionData32

Function name

__STATIC_INLINE uint32_t LL_ADC_REG_ReadConversionData32 (const ADC_TypeDef * ADCx)

Function description

Get ADC group regular conversion data, range fit for all ADC configurations: all ADC resolutions and all oversampling increased data width (for devices with feature oversampling).

Parameters

- **ADCx:** ADC instance

Return values

- **Value:** between Min_Data=0x00000000 and Max_Data=0xFFFFFFFF

Reference Manual to LL API cross reference:

- DR RDATA LL_ADC_REG_ReadConversionData32

LL_ADC_REG_ReadConversionData12

Function name

__STATIC_INLINE uint16_t LL_ADC_REG_ReadConversionData12 (const ADC_TypeDef * ADCx)

Function description

Get ADC group regular conversion data, range fit for ADC resolution 12 bits.

Parameters

- **ADCx:** ADC instance

Return values

- **Value:** between Min_Data=0x000 and Max_Data=0xFFF

Notes

- For devices with feature oversampling: Oversampling can increase data width, function for extended range may be needed: LL_ADC_REG_ReadConversionData32.

Reference Manual to LL API cross reference:

- DR RDATA LL_ADC_REG_ReadConversionData12

LL_ADC_REG_ReadConversionData10

Function name

__STATIC_INLINE uint16_t LL_ADC_REG_ReadConversionData10 (const ADC_TypeDef * ADCx)

Function description

Get ADC group regular conversion data, range fit for ADC resolution 10 bits.

Parameters

- **ADCx:** ADC instance

Return values

- **Value:** between Min_Data=0x000 and Max_Data=0x3FF

Notes

- For devices with feature oversampling: Oversampling can increase data width, function for extended range may be needed: LL_ADC_REG_ReadConversionData32.

Reference Manual to LL API cross reference:

- DR RDATA LL_ADC_REG_ReadConversionData10

LL_ADC_REG_ReadConversionData8

Function name

__STATIC_INLINE uint8_t LL_ADC_REG_ReadConversionData8 (const ADC_TypeDef * ADCx)

Function description

Get ADC group regular conversion data, range fit for ADC resolution 8 bits.

Parameters

- **ADCx:** ADC instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0xFF

Notes

- For devices with feature oversampling: Oversampling can increase data width, function for extended range may be needed: LL_ADC_REG_ReadConversionData32.

Reference Manual to LL API cross reference:

- DR RDATA LL_ADC_REG_ReadConversionData8

LL_ADC_REG_ReadConversionData6

Function name

__STATIC_INLINE uint8_t LL_ADC_REG_ReadConversionData6 (const ADC_TypeDef * ADCx)

Function description

Get ADC group regular conversion data, range fit for ADC resolution 6 bits.

Parameters

- **ADCx:** ADC instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x3F

Notes

- For devices with feature oversampling: Oversampling can increase data width, function for extended range may be needed: LL_ADC_REG_ReadConversionData32.

Reference Manual to LL API cross reference:

- DR RDATA LL_ADC_REG_ReadConversionData6

LL_ADC_REG_ReadMultiConvPacking

Function name

__STATIC_INLINE uint32_t LL_ADC_REG_ReadMultiConvPacking (const ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get ADC multimode conversion data of ADC master and ADC slave concatenated.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **Value:** between Min_Data=0x00000000 and Max_Data=0xFFFFFFFF

Notes

- Multimode data concatenation is usable under conditions of ADC multimode data format selected and data width, refer to description of literals in Multimode - Data format.
- From raw data with ADC master and slave concatenated, a macro is available to get the conversion data of ADC master or ADC slave: see helper macro __LL_ADC_MULTI_CONV_DATA_MASTER_SLAVE().

Reference Manual to LL API cross reference:

- CDR RDATA_MST LL_ADC_REG_ReadMultiConvPacking
- CDR RDATA_SLV LL_ADC_REG_ReadMultiConvPacking

LL_ADC_REG_ReadMultiConvNoPacking

Function name

__STATIC_INLINE uint32_t LL_ADC_REG_ReadMultiConvNoPacking (const ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get ADC multimode conversion data of ADC master or ADC slave (data set alternatively in specific data register).

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **Value:** between Min_Data=0x00000000 and Max_Data=0xFFFFFFFF

Notes

- Multimode data without concatenation is compliant with all ADC multimode data format and data width, refer to description of literals in Multimode - Data format.
- Multimode data without packing is using a unique data register (CDR2) for data of ADC master and slave alternatively. Therefore, data can be overwritten very quickly, recommended method to retrieve data without risk to miss some occurrences is to use DMA transfer.

Reference Manual to LL API cross reference:

- CDR2 RDATA_ALT LL_ADC_REG_ReadMultiConvNoPacking

LL_ADC_INJ_StartConversion

Function name

__STATIC_INLINE void LL_ADC_INJ_StartConversion (ADC_TypeDef * ADCx)

Function description

Start ADC group injected conversion.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Notes

- On this STM32 series, this function is relevant for both internal trigger (SW start) and external trigger: If ADC trigger has been set to software start, ADC conversion starts immediately.If ADC trigger has been set to external trigger, ADC conversion will start at next trigger event (on the selected trigger edge) following the ADC start conversion command.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be enabled without conversion on going on group injected, without conversion stop command on going on group injected, without ADC disable command on going.

Reference Manual to LL API cross reference:

- CR JADSTART LL_ADC_INJ_StartConversion

LL_ADC_INJ_StopConversion

Function name

__STATIC_INLINE void LL_ADC_INJ_StopConversion (ADC_TypeDef * ADCx)

Function description

Stop ADC group injected conversion.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Notes

- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be enabled with conversion on going on group injected, without ADC disable command on going.

Reference Manual to LL API cross reference:

- CR JADSTP LL_ADC_INJ_StopConversion

LL_ADC_INJ_IsConversionOngoing

Function name

__STATIC_INLINE uint32_t LL_ADC_INJ_IsConversionOngoing (const ADC_TypeDef * ADCx)

Function description

Get ADC group injected conversion state.

Parameters

- **ADCx:** ADC instance

Return values

- **0:** no conversion is on going on ADC group injected.

Reference Manual to LL API cross reference:

- CR JADSTART LL_ADC_INJ_IsConversionOngoing

LL_ADC_INJ_IsStopConversionOngoing

Function name

__STATIC_INLINE uint32_t LL_ADC_INJ_IsStopConversionOngoing (const ADC_TypeDef * ADCx)

Function description

Get ADC group injected command of conversion stop state.

Parameters

- **ADCx:** ADC instance

Return values

- **0:** no command of conversion stop is on going on ADC group injected.

Reference Manual to LL API cross reference:

- CR JADSTP LL_ADC_INJ_IsStopConversionOngoing

LL_ADC_INJ_ReadConversionData32

Function name

__STATIC_INLINE uint32_t LL_ADC_INJ_ReadConversionData32 (const ADC_TypeDef * ADCx, uint32_t Rank)

Function description

Get ADC group injected conversion data, range fit for all ADC configurations: all ADC resolutions and all oversampling increased data width (for devices with feature oversampling).

Parameters

- **ADCx:** ADC instance
- **Rank:** This parameter can be one of the following values:
  - LL_ADC_INJ_RANK_1
  - LL_ADC_INJ_RANK_2
  - LL_ADC_INJ_RANK_3
  - LL_ADC_INJ_RANK_4

Return values

- **Value:** between Min_Data=0x00000000 and Max_Data=0xFFFFFFFF

Reference Manual to LL API cross reference:

- JDR1 JDATA LL_ADC_INJ_ReadConversionData32
- JDR2 JDATA LL_ADC_INJ_ReadConversionData32
- JDR3 JDATA LL_ADC_INJ_ReadConversionData32
- JDR4 JDATA LL_ADC_INJ_ReadConversionData32

LL_ADC_INJ_ReadConversionData12

Function name

__STATIC_INLINE uint16_t LL_ADC_INJ_ReadConversionData12 (const ADC_TypeDef * ADCx, uint32_t Rank)

Function description

Get ADC group injected conversion data, range fit for ADC resolution 12 bits.

Parameters

- **ADCx:** ADC instance
- **Rank:** This parameter can be one of the following values:
  - LL_ADC_INJ_RANK_1
  - LL_ADC_INJ_RANK_2
  - LL_ADC_INJ_RANK_3
  - LL_ADC_INJ_RANK_4

Return values

- **Value:** between Min_Data=0x000 and Max_Data=0xFFF

Notes

- For devices with feature oversampling: Oversampling can increase data width, function for extended range may be needed: LL_ADC_INJ_ReadConversionData32.

Reference Manual to LL API cross reference:

- JDR1 JDATA LL_ADC_INJ_ReadConversionData12
- JDR2 JDATA LL_ADC_INJ_ReadConversionData12
- JDR3 JDATA LL_ADC_INJ_ReadConversionData12
- JDR4 JDATA LL_ADC_INJ_ReadConversionData12

LL_ADC_INJ_ReadConversionData10

Function name

__STATIC_INLINE uint16_t LL_ADC_INJ_ReadConversionData10 (const ADC_TypeDef * ADCx, uint32_t Rank)

Function description

Get ADC group injected conversion data, range fit for ADC resolution 10 bits.

Parameters

- **ADCx:** ADC instance
- **Rank:** This parameter can be one of the following values:
  - LL_ADC_INJ_RANK_1
  - LL_ADC_INJ_RANK_2
  - LL_ADC_INJ_RANK_3
  - LL_ADC_INJ_RANK_4

Return values

- **Value:** between Min_Data=0x000 and Max_Data=0x3FF

Notes

- For devices with feature oversampling: Oversampling can increase data width, function for extended range may be needed: LL_ADC_INJ_ReadConversionData32.

Reference Manual to LL API cross reference:

- JDR1 JDATA LL_ADC_INJ_ReadConversionData10
- JDR2 JDATA LL_ADC_INJ_ReadConversionData10
- JDR3 JDATA LL_ADC_INJ_ReadConversionData10
- JDR4 JDATA LL_ADC_INJ_ReadConversionData10

LL_ADC_INJ_ReadConversionData8

Function name

__STATIC_INLINE uint8_t LL_ADC_INJ_ReadConversionData8 (const ADC_TypeDef * ADCx, uint32_t Rank)

Function description

Get ADC group injected conversion data, range fit for ADC resolution 8 bits.

Parameters

- **ADCx:** ADC instance
- **Rank:** This parameter can be one of the following values:
  - LL_ADC_INJ_RANK_1
  - LL_ADC_INJ_RANK_2
  - LL_ADC_INJ_RANK_3
  - LL_ADC_INJ_RANK_4

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0xFF

Notes

- For devices with feature oversampling: Oversampling can increase data width, function for extended range may be needed: LL_ADC_INJ_ReadConversionData32.

Reference Manual to LL API cross reference:

- JDR1 JDATA LL_ADC_INJ_ReadConversionData8
- JDR2 JDATA LL_ADC_INJ_ReadConversionData8
- JDR3 JDATA LL_ADC_INJ_ReadConversionData8
- JDR4 JDATA LL_ADC_INJ_ReadConversionData8

LL_ADC_INJ_ReadConversionData6

Function name

__STATIC_INLINE uint8_t LL_ADC_INJ_ReadConversionData6 (const ADC_TypeDef * ADCx, uint32_t Rank)

Function description

Get ADC group injected conversion data, range fit for ADC resolution 6 bits.

Parameters

- **ADCx:** ADC instance
- **Rank:** This parameter can be one of the following values:
  - LL_ADC_INJ_RANK_1
  - LL_ADC_INJ_RANK_2
  - LL_ADC_INJ_RANK_3
  - LL_ADC_INJ_RANK_4

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x3F

Notes

- For devices with feature oversampling: Oversampling can increase data width, function for extended range may be needed: LL_ADC_INJ_ReadConversionData32.

Reference Manual to LL API cross reference:

- JDR1 JDATA LL_ADC_INJ_ReadConversionData6
- JDR2 JDATA LL_ADC_INJ_ReadConversionData6
- JDR3 JDATA LL_ADC_INJ_ReadConversionData6
- JDR4 JDATA LL_ADC_INJ_ReadConversionData6

LL_ADC_IsActiveFlag_ADRDY

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_ADRDY (const ADC_TypeDef * ADCx)

Function description

Get flag ADC ready.

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Notes

- On this STM32 series, flag LL_ADC_FLAG_ADRDY is raised when the ADC is enabled and when conversion clock is active. (not only core clock: this ADC has a dual clock domain)

Reference Manual to LL API cross reference:

- ISR ADRDY LL_ADC_IsActiveFlag_ADRDY

LL_ADC_IsActiveFlag_EOC

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_EOC (const ADC_TypeDef * ADCx)

Function description

Get flag ADC group regular end of unitary conversion.

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR EOC LL_ADC_IsActiveFlag_EOC

LL_ADC_IsActiveFlag_EOS

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_EOS (const ADC_TypeDef * ADCx)

Function description

Get flag ADC group regular end of sequence conversions.

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR EOS LL_ADC_IsActiveFlag_EOS

LL_ADC_IsActiveFlag_OVR

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_OVR (const ADC_TypeDef * ADCx)

Function description

Get flag ADC group regular overrun.

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR OVR LL_ADC_IsActiveFlag_OVR

LL_ADC_IsActiveFlag_EOSMP

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_EOSMP (const ADC_TypeDef * ADCx)

Function description

Get flag ADC group regular end of sampling phase.

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR EOSMP LL_ADC_IsActiveFlag_EOSMP

LL_ADC_IsActiveFlag_JEOC

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_JEOC (const ADC_TypeDef * ADCx)

Function description

Get flag ADC group injected end of unitary conversion.

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR JEOC LL_ADC_IsActiveFlag_JEOC

LL_ADC_IsActiveFlag_JEOS

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_JEOS (const ADC_TypeDef * ADCx)

Function description

Get flag ADC group injected end of sequence conversions.

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR JEOS LL_ADC_IsActiveFlag_JEOS

LL_ADC_IsActiveFlag_AWD1

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_AWD1 (const ADC_TypeDef * ADCx)

Function description

Get flag ADC analog watchdog 1 flag.

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR AWD1 LL_ADC_IsActiveFlag_AWD1

LL_ADC_IsActiveFlag_AWD2

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_AWD2 (const ADC_TypeDef * ADCx)

Function description

Get flag ADC analog watchdog 2.

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR AWD2 LL_ADC_IsActiveFlag_AWD2

LL_ADC_IsActiveFlag_AWD3

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_AWD3 (const ADC_TypeDef * ADCx)

Function description

Get flag ADC analog watchdog 3.

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR AWD3 LL_ADC_IsActiveFlag_AWD3

LL_ADC_ClearFlag_ADRDY

Function name

__STATIC_INLINE void LL_ADC_ClearFlag_ADRDY (ADC_TypeDef * ADCx)

Function description

Clear flag ADC ready.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Notes

- On this STM32 series, flag LL_ADC_FLAG_ADRDY is raised when the ADC is enabled and when conversion clock is active. (not only core clock: this ADC has a dual clock domain)

Reference Manual to LL API cross reference:

- ISR ADRDY LL_ADC_ClearFlag_ADRDY

LL_ADC_ClearFlag_EOC

Function name

__STATIC_INLINE void LL_ADC_ClearFlag_EOC (ADC_TypeDef * ADCx)

Function description

Clear flag ADC group regular end of unitary conversion.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ISR EOC LL_ADC_ClearFlag_EOC

LL_ADC_ClearFlag_EOS

Function name

__STATIC_INLINE void LL_ADC_ClearFlag_EOS (ADC_TypeDef * ADCx)

Function description

Clear flag ADC group regular end of sequence conversions.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ISR EOS LL_ADC_ClearFlag_EOS

LL_ADC_ClearFlag_OVR

Function name

__STATIC_INLINE void LL_ADC_ClearFlag_OVR (ADC_TypeDef * ADCx)

Function description

Clear flag ADC group regular overrun.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ISR OVR LL_ADC_ClearFlag_OVR

LL_ADC_ClearFlag_EOSMP

Function name

__STATIC_INLINE void LL_ADC_ClearFlag_EOSMP (ADC_TypeDef * ADCx)

Function description

Clear flag ADC group regular end of sampling phase.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ISR EOSMP LL_ADC_ClearFlag_EOSMP

LL_ADC_ClearFlag_JEOC

Function name

__STATIC_INLINE void LL_ADC_ClearFlag_JEOC (ADC_TypeDef * ADCx)

Function description

Clear flag ADC group injected end of unitary conversion.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ISR JEOC LL_ADC_ClearFlag_JEOC

LL_ADC_ClearFlag_JEOS

Function name

__STATIC_INLINE void LL_ADC_ClearFlag_JEOS (ADC_TypeDef * ADCx)

Function description

Clear flag ADC group injected end of sequence conversions.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ISR JEOS LL_ADC_ClearFlag_JEOS

LL_ADC_ClearFlag_AWD1

Function name

__STATIC_INLINE void LL_ADC_ClearFlag_AWD1 (ADC_TypeDef * ADCx)

Function description

Clear flag ADC analog watchdog 1.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ISR AWD1 LL_ADC_ClearFlag_AWD1

LL_ADC_ClearFlag_AWD2

Function name

__STATIC_INLINE void LL_ADC_ClearFlag_AWD2 (ADC_TypeDef * ADCx)

Function description

Clear flag ADC analog watchdog 2.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ISR AWD2 LL_ADC_ClearFlag_AWD2

LL_ADC_ClearFlag_AWD3

Function name

__STATIC_INLINE void LL_ADC_ClearFlag_AWD3 (ADC_TypeDef * ADCx)

Function description

Clear flag ADC analog watchdog 3.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ISR AWD3 LL_ADC_ClearFlag_AWD3

LL_ADC_IsActiveFlag_MST_ADRDY

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_MST_ADRDY (const ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC ready of the ADC master.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR ADRDY_MST LL_ADC_IsActiveFlag_MST_ADRDY

LL_ADC_IsActiveFlag_SLV_ADRDY

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_SLV_ADRDY (const ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC ready of the ADC slave.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR ADRDY_SLV LL_ADC_IsActiveFlag_SLV_ADRDY

LL_ADC_IsActiveFlag_MST_EOC

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_MST_EOC (const ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC group regular end of unitary conversion of the ADC master.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR EOC_MST LL_ADC_IsActiveFlag_MST_EOC

LL_ADC_IsActiveFlag_SLV_EOC

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_SLV_EOC (const ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC group regular end of unitary conversion of the ADC slave.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR EOC_SLV LL_ADC_IsActiveFlag_SLV_EOC

LL_ADC_IsActiveFlag_MST_EOS

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_MST_EOS (const ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC group regular end of sequence conversions of the ADC master.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR EOS_MST LL_ADC_IsActiveFlag_MST_EOS

LL_ADC_IsActiveFlag_SLV_EOS

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_SLV_EOS (const ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC group regular end of sequence conversions of the ADC slave.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR EOS_SLV LL_ADC_IsActiveFlag_SLV_EOS

LL_ADC_IsActiveFlag_MST_OVR

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_MST_OVR (const ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC group regular overrun of the ADC master.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR OVR_MST LL_ADC_IsActiveFlag_MST_OVR

LL_ADC_IsActiveFlag_SLV_OVR

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_SLV_OVR (const ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC group regular overrun of the ADC slave.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR OVR_SLV LL_ADC_IsActiveFlag_SLV_OVR

LL_ADC_IsActiveFlag_MST_EOSMP

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_MST_EOSMP (const ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC group regular end of sampling of the ADC master.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR EOSMP_MST LL_ADC_IsActiveFlag_MST_EOSMP

LL_ADC_IsActiveFlag_SLV_EOSMP

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_SLV_EOSMP (const ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC group regular end of sampling of the ADC slave.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR EOSMP_SLV LL_ADC_IsActiveFlag_SLV_EOSMP

LL_ADC_IsActiveFlag_MST_JEOC

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_MST_JEOC (const ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC group injected end of unitary conversion of the ADC master.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR JEOC_MST LL_ADC_IsActiveFlag_MST_JEOC

LL_ADC_IsActiveFlag_SLV_JEOC

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_SLV_JEOC (const ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC group injected end of unitary conversion of the ADC slave.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR JEOC_SLV LL_ADC_IsActiveFlag_SLV_JEOC

LL_ADC_IsActiveFlag_MST_JEOS

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_MST_JEOS (const ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC group injected end of sequence conversions of the ADC master.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR JEOS_MST LL_ADC_IsActiveFlag_MST_JEOS

LL_ADC_IsActiveFlag_SLV_JEOS

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_SLV_JEOS (const ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC group injected end of sequence conversions of the ADC slave.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR JEOS_SLV LL_ADC_IsActiveFlag_SLV_JEOS

LL_ADC_IsActiveFlag_MST_AWD1

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_MST_AWD1 (const ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC analog watchdog 1 of the ADC master.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR AWD1_MST LL_ADC_IsActiveFlag_MST_AWD1

LL_ADC_IsActiveFlag_SLV_AWD1

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_SLV_AWD1 (const ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode analog watchdog 1 of the ADC slave.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR AWD1_SLV LL_ADC_IsActiveFlag_SLV_AWD1

LL_ADC_IsActiveFlag_MST_AWD2

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_MST_AWD2 (const ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC analog watchdog 2 of the ADC master.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR AWD2_MST LL_ADC_IsActiveFlag_MST_AWD2

LL_ADC_IsActiveFlag_SLV_AWD2

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_SLV_AWD2 (const ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC analog watchdog 2 of the ADC slave.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR AWD2_SLV LL_ADC_IsActiveFlag_SLV_AWD2

LL_ADC_IsActiveFlag_MST_AWD3

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_MST_AWD3 (const ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC analog watchdog 3 of the ADC master.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR AWD3_MST LL_ADC_IsActiveFlag_MST_AWD3

LL_ADC_IsActiveFlag_SLV_AWD3

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_SLV_AWD3 (const ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC analog watchdog 3 of the ADC slave.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR AWD3_SLV LL_ADC_IsActiveFlag_SLV_AWD3

LL_ADC_EnableIT_ADRDY

Function name

__STATIC_INLINE void LL_ADC_EnableIT_ADRDY (ADC_TypeDef * ADCx)

Function description

Enable ADC ready.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER ADRDYIE LL_ADC_EnableIT_ADRDY

LL_ADC_EnableIT_EOC

Function name

__STATIC_INLINE void LL_ADC_EnableIT_EOC (ADC_TypeDef * ADCx)

Function description

Enable interruption ADC group regular end of unitary conversion.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER EOCIE LL_ADC_EnableIT_EOC

LL_ADC_EnableIT_EOS

Function name

__STATIC_INLINE void LL_ADC_EnableIT_EOS (ADC_TypeDef * ADCx)

Function description

Enable interruption ADC group regular end of sequence conversions.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER EOSIE LL_ADC_EnableIT_EOS

LL_ADC_EnableIT_OVR

Function name

__STATIC_INLINE void LL_ADC_EnableIT_OVR (ADC_TypeDef * ADCx)

Function description

Enable ADC group regular interruption overrun.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER OVRIE LL_ADC_EnableIT_OVR

LL_ADC_EnableIT_EOSMP

Function name

__STATIC_INLINE void LL_ADC_EnableIT_EOSMP (ADC_TypeDef * ADCx)

Function description

Enable interruption ADC group regular end of sampling.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER EOSMPIE LL_ADC_EnableIT_EOSMP

LL_ADC_EnableIT_JEOC

Function name

__STATIC_INLINE void LL_ADC_EnableIT_JEOC (ADC_TypeDef * ADCx)

Function description

Enable interruption ADC group injected end of unitary conversion.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER JEOCIE LL_ADC_EnableIT_JEOC

LL_ADC_EnableIT_JEOS

Function name

__STATIC_INLINE void LL_ADC_EnableIT_JEOS (ADC_TypeDef * ADCx)

Function description

Enable interruption ADC group injected end of sequence conversions.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER JEOSIE LL_ADC_EnableIT_JEOS

LL_ADC_EnableIT_AWD1

Function name

__STATIC_INLINE void LL_ADC_EnableIT_AWD1 (ADC_TypeDef * ADCx)

Function description

Enable interruption ADC analog watchdog 1.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER AWD1IE LL_ADC_EnableIT_AWD1

LL_ADC_EnableIT_AWD2

Function name

__STATIC_INLINE void LL_ADC_EnableIT_AWD2 (ADC_TypeDef * ADCx)

Function description

Enable interruption ADC analog watchdog 2.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER AWD2IE LL_ADC_EnableIT_AWD2

LL_ADC_EnableIT_AWD3

Function name

__STATIC_INLINE void LL_ADC_EnableIT_AWD3 (ADC_TypeDef * ADCx)

Function description

Enable interruption ADC analog watchdog 3.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER AWD3IE LL_ADC_EnableIT_AWD3

LL_ADC_DisableIT_ADRDY

Function name

__STATIC_INLINE void LL_ADC_DisableIT_ADRDY (ADC_TypeDef * ADCx)

Function description

Disable interruption ADC ready.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER ADRDYIE LL_ADC_DisableIT_ADRDY

LL_ADC_DisableIT_EOC

Function name

__STATIC_INLINE void LL_ADC_DisableIT_EOC (ADC_TypeDef * ADCx)

Function description

Disable interruption ADC group regular end of unitary conversion.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER EOCIE LL_ADC_DisableIT_EOC

LL_ADC_DisableIT_EOS

Function name

__STATIC_INLINE void LL_ADC_DisableIT_EOS (ADC_TypeDef * ADCx)

Function description

Disable interruption ADC group regular end of sequence conversions.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER EOSIE LL_ADC_DisableIT_EOS

LL_ADC_DisableIT_OVR

Function name

__STATIC_INLINE void LL_ADC_DisableIT_OVR (ADC_TypeDef * ADCx)

Function description

Disable interruption ADC group regular overrun.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER OVRIE LL_ADC_DisableIT_OVR

LL_ADC_DisableIT_EOSMP

Function name

__STATIC_INLINE void LL_ADC_DisableIT_EOSMP (ADC_TypeDef * ADCx)

Function description

Disable interruption ADC group regular end of sampling.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER EOSMPIE LL_ADC_DisableIT_EOSMP

LL_ADC_DisableIT_JEOC

Function name

__STATIC_INLINE void LL_ADC_DisableIT_JEOC (ADC_TypeDef * ADCx)

Function description

Disable interruption ADC group regular end of unitary conversion.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER JEOCIE LL_ADC_DisableIT_JEOC

LL_ADC_DisableIT_JEOS

Function name

__STATIC_INLINE void LL_ADC_DisableIT_JEOS (ADC_TypeDef * ADCx)

Function description

Disable interruption ADC group injected end of sequence conversions.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER JEOSIE LL_ADC_DisableIT_JEOS

LL_ADC_DisableIT_AWD1

Function name

__STATIC_INLINE void LL_ADC_DisableIT_AWD1 (ADC_TypeDef * ADCx)

Function description

Disable interruption ADC analog watchdog 1.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER AWD1IE LL_ADC_DisableIT_AWD1

LL_ADC_DisableIT_AWD2

Function name

__STATIC_INLINE void LL_ADC_DisableIT_AWD2 (ADC_TypeDef * ADCx)

Function description

Disable interruption ADC analog watchdog 2.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER AWD2IE LL_ADC_DisableIT_AWD2

LL_ADC_DisableIT_AWD3

Function name

__STATIC_INLINE void LL_ADC_DisableIT_AWD3 (ADC_TypeDef * ADCx)

Function description

Disable interruption ADC analog watchdog 3.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER AWD3IE LL_ADC_DisableIT_AWD3

LL_ADC_IsEnabledIT_ADRDY

Function name

__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_ADRDY (const ADC_TypeDef * ADCx)

Function description

Get state of interruption ADC ready (0: interrupt disabled, 1: interrupt enabled).

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER ADRDYIE LL_ADC_IsEnabledIT_ADRDY

LL_ADC_IsEnabledIT_EOC

Function name

__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_EOC (const ADC_TypeDef * ADCx)

Function description

Get state of interruption ADC group regular end of unitary conversion (0: interrupt disabled, 1: interrupt enabled).

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER EOCIE LL_ADC_IsEnabledIT_EOC

LL_ADC_IsEnabledIT_EOS

Function name

__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_EOS (const ADC_TypeDef * ADCx)

Function description

Get state of interruption ADC group regular end of sequence conversions (0: interrupt disabled, 1: interrupt enabled).

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER EOSIE LL_ADC_IsEnabledIT_EOS

LL_ADC_IsEnabledIT_OVR

Function name

__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_OVR (const ADC_TypeDef * ADCx)

Function description

Get state of interruption ADC group regular overrun (0: interrupt disabled, 1: interrupt enabled).

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER OVRIE LL_ADC_IsEnabledIT_OVR

LL_ADC_IsEnabledIT_EOSMP

Function name

__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_EOSMP (const ADC_TypeDef * ADCx)

Function description

Get state of interruption ADC group regular end of sampling (0: interrupt disabled, 1: interrupt enabled).

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER EOSMPIE LL_ADC_IsEnabledIT_EOSMP

LL_ADC_IsEnabledIT_JEOC

Function name

__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_JEOC (const ADC_TypeDef * ADCx)

Function description

Get state of interruption ADC group injected end of unitary conversion (0: interrupt disabled, 1: interrupt enabled).

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER JEOCIE LL_ADC_IsEnabledIT_JEOC

LL_ADC_IsEnabledIT_JEOS

Function name

__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_JEOS (const ADC_TypeDef * ADCx)

Function description

Get state of interruption ADC group injected end of sequence conversions (0: interrupt disabled, 1: interrupt enabled).

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER JEOSIE LL_ADC_IsEnabledIT_JEOS

LL_ADC_IsEnabledIT_AWD1

Function name

__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_AWD1 (const ADC_TypeDef * ADCx)

Function description

Get state of interruption ADC analog watchdog 1 (0: interrupt disabled, 1: interrupt enabled).

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER AWD1IE LL_ADC_IsEnabledIT_AWD1

LL_ADC_IsEnabledIT_AWD2

Function name

__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_AWD2 (const ADC_TypeDef * ADCx)

Function description

Get state of interruption Get ADC analog watchdog 2 (0: interrupt disabled, 1: interrupt enabled).

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER AWD2IE LL_ADC_IsEnabledIT_AWD2

LL_ADC_IsEnabledIT_AWD3

Function name

__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_AWD3 (const ADC_TypeDef * ADCx)

Function description

Get state of interruption Get ADC analog watchdog 3 (0: interrupt disabled, 1: interrupt enabled).

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER AWD3IE LL_ADC_IsEnabledIT_AWD3

LL_ADC_CommonDeInit

Function name

ErrorStatus LL_ADC_CommonDeInit (const ADC_Common_TypeDef * ADCxy_COMMON)

Function description

De-initialize registers of all ADC instances belonging to the same ADC common instance to their default reset values.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: ADC common registers are de-initialized
  - ERROR: not applicable

Notes

- This function is performing a hard reset, using high level clock source RCC ADC reset. Caution: On this STM32 series, if several ADC instances are available on the selected device, RCC ADC reset will reset all ADC instances belonging to the common ADC instance. To de-initialize only 1 ADC instance, use function LL_ADC_DeInit().

LL_ADC_CommonInit

Function name

ErrorStatus LL_ADC_CommonInit (ADC_Common_TypeDef * ADCxy_COMMON, const LL_ADC_CommonInitTypeDef * pADC_CommonInitStruct)

Function description

Initialize some features of ADC common parameters (all ADC instances belonging to the same ADC common instance) and multimode (for devices with several ADC instances available).

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())
- **pADC_CommonInitStruct:** Pointer to a LL_ADC_CommonInitTypeDef structure

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: ADC common registers are initialized
  - ERROR: ADC common registers are not initialized

Notes

- The setting of ADC common parameters is conditioned to ADC instances state: All ADC instances belonging to the same ADC common instance must be disabled.

LL_ADC_CommonStructInit

Function name

void LL_ADC_CommonStructInit (LL_ADC_CommonInitTypeDef * pADC_CommonInitStruct)

Function description

Set each LL_ADC_CommonInitTypeDef field to default value.

Parameters

- **pADC_CommonInitStruct:** Pointer to a LL_ADC_CommonInitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

LL_ADC_DeInit

Function name

ErrorStatus LL_ADC_DeInit (ADC_TypeDef * ADCx)

Function description

De-initialize registers of the selected ADC instance to their default reset values.

Parameters

- **ADCx:** ADC instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: ADC registers are de-initialized
  - ERROR: ADC registers are not de-initialized

Notes

- To reset all ADC instances quickly (perform a hard reset), use function LL_ADC_CommonDeInit().
- If this functions returns error status, it means that ADC instance is in an unknown state. In this case, perform a hard reset using high level clock source RCC ADC reset. Caution: On this STM32 series, if several ADC instances are available on the selected device, RCC ADC reset will reset all ADC instances belonging to the common ADC instance. Refer to function LL_ADC_CommonDeInit().

LL_ADC_Init

Function name

ErrorStatus LL_ADC_Init (ADC_TypeDef * ADCx, const LL_ADC_InitTypeDef * pADC_InitStruct)

Function description

Initialize some features of ADC instance.

Parameters

- **ADCx:** ADC instance
- **pADC_InitStruct:** Pointer to a LL_ADC_REG_InitTypeDef structure

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: ADC registers are initialized
  - ERROR: ADC registers are not initialized

Notes

- These parameters have an impact on ADC scope: ADC instance. Affects both group regular and group injected (availability of ADC group injected depends on STM32 families). Refer to corresponding unitary functions into Configuration of ADC hierarchical scope: ADC instance .
- The setting of these parameters by function LL_ADC_Init() is conditioned to ADC state: ADC instance must be disabled. This condition is applied to all ADC features, for efficiency and compatibility over all STM32 families. However, the different features can be set under different ADC state conditions (setting possible with ADC enabled without conversion on going, ADC enabled with conversion on going, ...) Each feature can be updated afterwards with a unitary function and potentially with ADC in a different state than disabled, refer to description of each function for setting conditioned to ADC state.
- After using this function, some other features must be configured using LL unitary functions. The minimum configuration remaining to be done is: Set ADC group regular or group injected sequencer: map channel on the selected sequencer rank. Refer to function LL_ADC_REG_SetSequencerRanks().Set ADC channel sampling time Refer to function LL_ADC_SetChannelSamplingTime();

LL_ADC_StructInit

Function name

void LL_ADC_StructInit (LL_ADC_InitTypeDef * pADC_InitStruct)

Function description

Set each LL_ADC_InitTypeDef field to default value.

Parameters

- **pADC_InitStruct:** Pointer to a LL_ADC_InitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

LL_ADC_REG_Init

Function name

ErrorStatus LL_ADC_REG_Init (ADC_TypeDef * ADCx, const LL_ADC_REG_InitTypeDef * pADC_RegInitStruct)

Function description

Initialize some features of ADC group regular.

Parameters

- **ADCx:** ADC instance
- **pADC_RegInitStruct:** Pointer to a LL_ADC_REG_InitTypeDef structure

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: ADC registers are initialized
  - ERROR: ADC registers are not initialized

Notes

- These parameters have an impact on ADC scope: ADC group regular. Refer to corresponding unitary functions into Configuration of ADC hierarchical scope: group regular (functions with prefix "REG").
- The setting of these parameters by function LL_ADC_Init() is conditioned to ADC state: ADC instance must be disabled. This condition is applied to all ADC features, for efficiency and compatibility over all STM32 families. However, the different features can be set under different ADC state conditions (setting possible with ADC enabled without conversion on going, ADC enabled with conversion on going, ...) Each feature can be updated afterwards with a unitary function and potentially with ADC in a different state than disabled, refer to description of each function for setting conditioned to ADC state.
- After using this function, other features must be configured using LL unitary functions. The minimum configuration remaining to be done is: Set ADC group regular or group injected sequencer: map channel on the selected sequencer rank. Refer to function LL_ADC_REG_SetSequencerRanks().Set ADC channel sampling time Refer to function LL_ADC_SetChannelSamplingTime();

LL_ADC_REG_StructInit

Function name

void LL_ADC_REG_StructInit (LL_ADC_REG_InitTypeDef * pADC_RegInitStruct)

Function description

Set each LL_ADC_REG_InitTypeDef field to default value.

Parameters

- **pADC_RegInitStruct:** Pointer to a LL_ADC_REG_InitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

LL_ADC_INJ_Init

Function name

ErrorStatus LL_ADC_INJ_Init (ADC_TypeDef * ADCx, const LL_ADC_INJ_InitTypeDef * pADC_InjInitStruct)

Function description

Initialize some features of ADC group injected.

Parameters

- **ADCx:** ADC instance
- **pADC_InjInitStruct:** Pointer to a LL_ADC_INJ_InitTypeDef structure

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: ADC registers are initialized
  - ERROR: ADC registers are not initialized

Notes

- These parameters have an impact on ADC scope: ADC group injected. Refer to corresponding unitary functions into Configuration of ADC hierarchical scope: group regular (functions with prefix "INJ").
- The setting of these parameters by function LL_ADC_Init() is conditioned to ADC state: ADC instance must be disabled. This condition is applied to all ADC features, for efficiency and compatibility over all STM32 families. However, the different features can be set under different ADC state conditions (setting possible with ADC enabled without conversion on going, ADC enabled with conversion on going, ...) Each feature can be updated afterwards with a unitary function and potentially with ADC in a different state than disabled, refer to description of each function for setting conditioned to ADC state.
- After using this function, other features must be configured using LL unitary functions. The minimum configuration remaining to be done is: Set ADC group injected sequencer: map channel on the selected sequencer rank. Refer to function LL_ADC_INJ_SetSequencerRanks().Set ADC channel sampling time Refer to function LL_ADC_SetChannelSamplingTime();

LL_ADC_INJ_StructInit

Function name

void LL_ADC_INJ_StructInit (LL_ADC_INJ_InitTypeDef * pADC_InjInitStruct)

Function description

Set each LL_ADC_INJ_InitTypeDef field to default value.

Parameters

- **pADC_InjInitStruct:** Pointer to a LL_ADC_INJ_InitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

85.3 ADC Firmware driver defines

The following section lists the various define and macros of the module.

85.3.1 ADC

ADC

Analog watchdog - Monitored channels

LL_ADC_AWD_DISABLE

ADC analog watchdog monitoring disabled

LL_ADC_AWD_ALL_CHANNELS_REG

ADC analog watchdog monitoring of all channels, converted by group regular only

LL_ADC_AWD_ALL_CHANNELS_INJ

ADC analog watchdog monitoring of all channels, converted by group injected only

LL_ADC_AWD_ALL_CHANNELS_REG_INJ

ADC analog watchdog monitoring of all channels, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_0_REG

ADC analog watchdog monitoring of ADC channel ADCx_IN0, converted by group regular only

LL_ADC_AWD_CHANNEL_0_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN0, converted by group injected only

LL_ADC_AWD_CHANNEL_0_REG_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN0, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_1_REG

ADC analog watchdog monitoring of ADC channel ADCx_IN1, converted by group regular only

LL_ADC_AWD_CHANNEL_1_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN1, converted by group injected only

LL_ADC_AWD_CHANNEL_1_REG_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN1, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_2_REG

ADC analog watchdog monitoring of ADC channel ADCx_IN2, converted by group regular only

LL_ADC_AWD_CHANNEL_2_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN2, converted by group injected only

LL_ADC_AWD_CHANNEL_2_REG_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN2, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_3_REG

ADC analog watchdog monitoring of ADC channel ADCx_IN3, converted by group regular only

LL_ADC_AWD_CHANNEL_3_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN3, converted by group injected only

LL_ADC_AWD_CHANNEL_3_REG_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN3, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_4_REG

ADC analog watchdog monitoring of ADC channel ADCx_IN4, converted by group regular only

LL_ADC_AWD_CHANNEL_4_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN4, converted by group injected only

LL_ADC_AWD_CHANNEL_4_REG_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN4, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_5_REG

ADC analog watchdog monitoring of ADC channel ADCx_IN5, converted by group regular only

LL_ADC_AWD_CHANNEL_5_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN5, converted by group injected only

LL_ADC_AWD_CHANNEL_5_REG_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN5, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_6_REG

ADC analog watchdog monitoring of ADC channel ADCx_IN6, converted by group regular only

LL_ADC_AWD_CHANNEL_6_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN6, converted by group injected only

LL_ADC_AWD_CHANNEL_6_REG_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN6, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_7_REG

ADC analog watchdog monitoring of ADC channel ADCx_IN7, converted by group regular only

LL_ADC_AWD_CHANNEL_7_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN7, converted by group injected only

LL_ADC_AWD_CHANNEL_7_REG_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN7, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_8_REG

ADC analog watchdog monitoring of ADC channel ADCx_IN8, converted by group regular only

LL_ADC_AWD_CHANNEL_8_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN8, converted by group injected only

LL_ADC_AWD_CHANNEL_8_REG_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN8, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_9_REG

ADC analog watchdog monitoring of ADC channel ADCx_IN9, converted by group regular only

LL_ADC_AWD_CHANNEL_9_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN9, converted by group injected only

LL_ADC_AWD_CHANNEL_9_REG_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN9, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_10_REG

ADC analog watchdog monitoring of ADC channel ADCx_IN10, converted by group regular only

LL_ADC_AWD_CHANNEL_10_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN10, converted by group injected only

LL_ADC_AWD_CHANNEL_10_REG_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN10, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_11_REG

ADC analog watchdog monitoring of ADC channel ADCx_IN11, converted by group regular only

LL_ADC_AWD_CHANNEL_11_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN11, converted by group injected only

LL_ADC_AWD_CHANNEL_11_REG_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN11, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_12_REG

ADC analog watchdog monitoring of ADC channel ADCx_IN12, converted by group regular only

LL_ADC_AWD_CHANNEL_12_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN12, converted by group injected only

LL_ADC_AWD_CHANNEL_12_REG_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN12, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_13_REG

ADC analog watchdog monitoring of ADC channel ADCx_IN13, converted by group regular only

LL_ADC_AWD_CHANNEL_13_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN13, converted by group injected only

LL_ADC_AWD_CHANNEL_13_REG_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN13, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_14_REG

ADC analog watchdog monitoring of ADC channel ADCx_IN14, converted by group regular only

LL_ADC_AWD_CHANNEL_14_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN14, converted by group only

LL_ADC_AWD_CHANNEL_14_REG_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN14, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_15_REG

ADC analog watchdog monitoring monitoring of ADC channel ADCx_IN15, converted by group regular only

LL_ADC_AWD_CHANNEL_15_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN15, converted by group injected only

LL_ADC_AWD_CHANNEL_15_REG_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN15, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_16_REG

ADC analog watchdog monitoring of ADC channel ADCx_IN16, converted by group regular only

LL_ADC_AWD_CHANNEL_16_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN16, converted by group injected only

LL_ADC_AWD_CHANNEL_16_REG_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN16, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_17_REG

ADC analog watchdog monitoring of ADC channel ADCx_IN17, converted by group regular only

LL_ADC_AWD_CHANNEL_17_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN17, converted by group injected only

LL_ADC_AWD_CHANNEL_17_REG_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN17, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_18_REG

ADC analog watchdog monitoring of ADC channel ADCx_IN18, converted by group regular only

LL_ADC_AWD_CHANNEL_18_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN18, converted by group injected only

LL_ADC_AWD_CHANNEL_18_REG_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN18, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_19_REG

ADC analog watchdog monitoring of ADC channel ADCx_IN19, converted by group regular only

LL_ADC_AWD_CHANNEL_19_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN19, converted by group injected only

LL_ADC_AWD_CHANNEL_19_REG_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN19, converted by either group regular or injected

LL_ADC_AWD_CH_VREFINT_REG

ADC analog watchdog monitoring of ADC internal channel connected to VrefInt: Internal voltage reference, converted by group regular only

LL_ADC_AWD_CH_VREFINT_INJ

ADC analog watchdog monitoring of ADC internal channel connected to VrefInt: Internal voltage reference, converted by group injected only

LL_ADC_AWD_CH_VREFINT_REG_INJ

ADC analog watchdog monitoring of ADC internal channel connected to VrefInt: Internal voltage reference, converted by either group regular or injected

LL_ADC_AWD_CH_VBAT_REG

ADC analog watchdog monitoring of ADC internal channel connected to Vbat/4: Vbat voltage through a divider ladder of factor 1/4 to have Vbat always below Vdda, converted by group regular only

LL_ADC_AWD_CH_VBAT_INJ

ADC analog watchdog monitoring of ADC internal channel connected to Vbat/4: Vbat voltage through a divider ladder of factor 1/4 to have Vbat always below Vdda, converted by group injected only

LL_ADC_AWD_CH_VBAT_REG_INJ

ADC analog watchdog monitoring of ADC internal channel connected to Vbat/4: Vbat voltage through a divider ladder of factor 1/4 to have Vbat always below Vdda

LL_ADC_AWD_CH_VDDCORE_REG

ADC analog watchdog monitoring of ADC internal channel connected to VddCore, converted by group regular only

LL_ADC_AWD_CH_VDDCORE_INJ

ADC analog watchdog monitoring of ADC internal channel connected to VddCore, converted by group injected only

LL_ADC_AWD_CH_VDDCORE_REG_INJ

ADC analog watchdog monitoring of ADC internal channel connected to VddCore, converted by either group regular or injected

Analog watchdog - filtering config

LL_ADC_AWD_FILTERING_NONE

ADC analog watchdog filtering disabled: one out-of-window sample is needed to raise flag or interrupt

LL_ADC_AWD_FILTERING_2SAMPLES

ADC analog watchdog filtering enabled: 2 out-of-window samples are needed to raise flag or interrupt

LL_ADC_AWD_FILTERING_3SAMPLES

ADC analog watchdog filtering enabled: 3 out-of-window samples are needed to raise flag or interrupt

LL_ADC_AWD_FILTERING_4SAMPLES

ADC analog watchdog filtering enabled: 4 out-of-window samples are needed to raise flag or interrupt

LL_ADC_AWD_FILTERING_5SAMPLES

ADC analog watchdog filtering enabled: 5 out-of-window samples are needed to raise flag or interrupt

LL_ADC_AWD_FILTERING_6SAMPLES

ADC analog watchdog filtering enabled: 6 out-of-window samples are needed to raise flag or interrupt

LL_ADC_AWD_FILTERING_7SAMPLES

ADC analog watchdog filtering enabled: 7 out-of-window samples are needed to raise flag or interrupt

LL_ADC_AWD_FILTERING_8SAMPLES

ADC analog watchdog filtering enabled: 8 out-of-window samples are needed to raise flag or interrupt

Analog watchdog - Analog watchdog number

LL_ADC_AWD1

ADC analog watchdog number 1

LL_ADC_AWD2

ADC analog watchdog number 2

LL_ADC_AWD3

ADC analog watchdog number 3

Analog watchdog - Thresholds

LL_ADC_AWD_THRESHOLD_HIGH

ADC analog watchdog threshold high

LL_ADC_AWD_THRESHOLD_LOW

ADC analog watchdog threshold low

ADC instance - Channel number

LL_ADC_CHANNEL_0

Channel ADCx_IN0

LL_ADC_CHANNEL_1

Channel ADCx_IN1

LL_ADC_CHANNEL_2

Channel ADCx_IN2

LL_ADC_CHANNEL_3

Channel ADCx_IN3

LL_ADC_CHANNEL_4

Channel ADCx_IN4

LL_ADC_CHANNEL_5

Channel ADCx_IN5

LL_ADC_CHANNEL_6

Channel ADCx_IN6

LL_ADC_CHANNEL_7

Channel ADCx_IN7

LL_ADC_CHANNEL_8

Channel ADCx_IN8

LL_ADC_CHANNEL_9

Channel ADCx_IN9

LL_ADC_CHANNEL_10

Channel ADCx_IN10

LL_ADC_CHANNEL_11

Channel ADCx_IN11

LL_ADC_CHANNEL_12

Channel ADCx_IN12

LL_ADC_CHANNEL_13

Channel ADCx_IN13

LL_ADC_CHANNEL_14

Channel ADCx_IN14

LL_ADC_CHANNEL_15

Channel ADCx_IN15

LL_ADC_CHANNEL_16

Channel ADCx_IN16

LL_ADC_CHANNEL_17

Channel ADCx_IN17

LL_ADC_CHANNEL_18

Channel ADCx_IN18

LL_ADC_CHANNEL_19

Channel ADCx_IN19

LL_ADC_CHANNEL_VREFINT

ADC internal channel connected to VrefInt: Internal voltage reference. On this STM32 series, ADC channel available only on ADC instance: ADC1.

LL_ADC_CHANNEL_VBAT

ADC internal channel connected to Vbat/4: Vbat voltage through a divider ladder of factor 1/4 to have channel voltage always below Vdda. On this STM32 series, ADC channel available only on ADC instance: ADC2.

LL_ADC_CHANNEL_VDDCORE

ADC internal channel connected to VddCore. On this STM32 series, ADC channel available only on ADC instance: ADC2.

Channel - Sampling time

LL_ADC_SAMPLINGTIME_1CYCLE_5

Sampling time 1.5 ADC clock cycle

LL_ADC_SAMPLINGTIME_2CYCLES_5

Sampling time 2.5 ADC clock cycles

LL_ADC_SAMPLINGTIME_6CYCLES_5

Sampling time 6.5 ADC clock cycles

LL_ADC_SAMPLINGTIME_11CYCLES_5

Sampling time 11.5 ADC clock cycles

LL_ADC_SAMPLINGTIME_23CYCLES_5

Sampling time 23.5 ADC clock cycles

LL_ADC_SAMPLINGTIME_46CYCLES_5

Sampling time 46.5 ADC clock cycles

LL_ADC_SAMPLINGTIME_246CYCLES_5

Sampling time 246.5 ADC clock cycles

LL_ADC_SAMPLINGTIME_1499CYCLES_5

Sampling time 1499.5 ADC clock cycles

Channel - Single or differential ending

LL_ADC_SINGLE_ENDED

ADC channel ending set to single ended (literal also used to set calibration mode)

LL_ADC_DIFFERENTIAL_ENDED

ADC channel ending set to differential (literal also used to set calibration mode)

LL_ADC_BOTH_SINGLE_DIFF_ENDED

ADC channel ending set to both single ended and differential (literal used only to set calibration factors)

ADC common - Measurement path to internal channels

LL_ADC_PATH_INTERNAL_NONE

ADC measurement paths all disabled

LL_ADC_PATH_INTERNAL_VREFINT

ADC measurement path to internal channel VrefInt

LL_ADC_PATH_INTERNAL_VBAT

ADC measurement path to internal channel Vbat

ADC flags

LL_ADC_FLAG_ADRDY

ADC flag ADC instance ready

LL_ADC_FLAG_EOC

ADC flag ADC group regular end of unitary conversion

LL_ADC_FLAG_EOS

ADC flag ADC group regular end of sequence conversions

LL_ADC_FLAG_OVR

ADC flag ADC group regular overrun

LL_ADC_FLAG_EOSMP

ADC flag ADC group regular end of sampling phase

LL_ADC_FLAG_JEOC

ADC flag ADC group injected end of unitary conversion

LL_ADC_FLAG_JEOS

ADC flag ADC group injected end of sequence conversions

LL_ADC_FLAG_AWD1

ADC flag ADC analog watchdog 1

LL_ADC_FLAG_AWD2

ADC flag ADC analog watchdog 2

LL_ADC_FLAG_AWD3

ADC flag ADC analog watchdog 3

LL_ADC_FLAG_ADRDY_MST

ADC flag ADC multimode master instance ready

LL_ADC_FLAG_ADRDY_SLV

ADC flag ADC multimode slave instance ready

LL_ADC_FLAG_EOC_MST

ADC flag ADC multimode master group regular end of unitary conversion

LL_ADC_FLAG_EOC_SLV

ADC flag ADC multimode slave group regular end of unitary conversion

LL_ADC_FLAG_EOS_MST

ADC flag ADC multimode master group regular end of sequence conversions

LL_ADC_FLAG_EOS_SLV

ADC flag ADC multimode slave group regular end of sequence conversions

LL_ADC_FLAG_OVR_MST

ADC flag ADC multimode master group regular overrun

LL_ADC_FLAG_OVR_SLV

ADC flag ADC multimode slave group regular overrun

LL_ADC_FLAG_EOSMP_MST

ADC flag ADC multimode master group regular end of sampling phase

LL_ADC_FLAG_EOSMP_SLV

ADC flag ADC multimode slave group regular end of sampling phase

LL_ADC_FLAG_JEOC_MST

ADC flag ADC multimode master group injected end of unitary conversion

LL_ADC_FLAG_JEOC_SLV

ADC flag ADC multimode slave group injected end of unitary conversion

LL_ADC_FLAG_JEOS_MST

ADC flag ADC multimode master group injected end of sequence conversions

LL_ADC_FLAG_JEOS_SLV

ADC flag ADC multimode slave group injected end of sequence conversions

LL_ADC_FLAG_AWD1_MST

ADC flag ADC multimode master analog watchdog 1 of the ADC master

LL_ADC_FLAG_AWD1_SLV

ADC flag ADC multimode slave analog watchdog 1 of the ADC slave

LL_ADC_FLAG_AWD2_MST

ADC flag ADC multimode master analog watchdog 2 of the ADC master

LL_ADC_FLAG_AWD2_SLV

ADC flag ADC multimode slave analog watchdog 2 of the ADC slave

LL_ADC_FLAG_AWD3_MST

ADC flag ADC multimode master analog watchdog 3 of the ADC master

LL_ADC_FLAG_AWD3_SLV

ADC flag ADC multimode slave analog watchdog 3 of the ADC slave

ADC instance - Groups

LL_ADC_GROUP_REGULAR

ADC group regular (available on all STM32 devices)

LL_ADC_GROUP_INJECTED

ADC group injected (not available on all STM32 devices)

LL_ADC_GROUP_REGULAR_INJECTED

ADC both groups regular and injected

Definitions of ADC hardware constraints delays

LL_ADC_DELAY_INTERNAL_REGUL_STAB_US

Delay for ADC stabilization time (ADC voltage regulator start-up time)

LL_ADC_DELAY_VREFINT_STAB_US

Delay for internal voltage reference stabilization time

LL_ADC_DELAY_CALIB_ENABLE_ADC_CYCLES

Delay required between ADC end of calibration and ADC enable

ADC group injected - Sequencer discontinuous mode

LL_ADC_INJ_SEQ_DISCONT_DISABLE

ADC group injected sequencer discontinuous mode disable

LL_ADC_INJ_SEQ_DISCONT_1RANK

ADC group injected sequencer discontinuous mode enable with sequence interruption every rank

ADC group injected - Sequencer ranks

LL_ADC_INJ_RANK_1

ADC group injected sequencer rank 1

LL_ADC_INJ_RANK_2

ADC group injected sequencer rank 2

LL_ADC_INJ_RANK_3

ADC group injected sequencer rank 3

LL_ADC_INJ_RANK_4

ADC group injected sequencer rank 4

ADC group injected - Sequencer scan length

LL_ADC_INJ_SEQ_SCAN_DISABLE

ADC group injected sequencer disable (equivalent to sequencer of 1 rank: ADC conversion on only 1 channel)

LL_ADC_INJ_SEQ_SCAN_ENABLE_2RANKS

ADC group injected sequencer enable with 2 ranks in the sequence

LL_ADC_INJ_SEQ_SCAN_ENABLE_3RANKS

ADC group injected sequencer enable with 3 ranks in the sequence

LL_ADC_INJ_SEQ_SCAN_ENABLE_4RANKS

ADC group injected sequencer enable with 4 ranks in the sequence

ADC group injected - Trigger edge

LL_ADC_INJ_TRIG_EXT_RISING

ADC group injected conversion trigger polarity set to rising edge

LL_ADC_INJ_TRIG_EXT_FALLING

ADC group injected conversion trigger polarity set to falling edge

LL_ADC_INJ_TRIG_EXT_RISINGFALLING

ADC group injected conversion trigger polarity set to both rising and falling edges

ADC group injected - Trigger source

LL_ADC_INJ_TRIG_SOFTWARE

ADC group inject. conversion trigger internal: SW start.

LL_ADC_INJ_TRIG_EXT_EXTI_LINE15

ADC group inject. conversion trigger from external peripheral: external interrupt line 15. Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_TIM1_CH4

ADC group inject. conversion trigger from external peripheral: TIM1 channel 4 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_TIM1_TRGO

ADC group inject. conversion trigger from external peripheral: TIM1 TRGO event. Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_TIM1_TRGO2

ADC group inject. conversion trigger from external peripheral: TIM1 TRGO2 event. Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_TIM2_CH1

ADC group inject. conversion trigger from external peripheral: TIM2 channel 1 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_TIM2_TRGO

ADC group inject. conversion trigger from external peripheral: TIM2 TRGO event. Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_TIM3_CH1

ADC group inject. conversion trigger from external peripheral: TIM3 channel 1 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_TIM3_CH3

ADC group inject. conversion trigger from external peripheral: TIM3 channel 3 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_TIM3_CH4

ADC group inject. conversion trigger from external peripheral: TIM3 channel 4 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_TIM3_TRGO

ADC group inject. conversion trigger from external peripheral: TIM3 TRGO event. Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_TIM4_TRGO

ADC group inject. conversion trigger from external peripheral: TIM4 TRGO event. Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_TIM5_TRGO

ADC group inject. conversion trigger from external peripheral: TIM5 TRGO event. Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_TIM6_TRGO

ADC group inject. conversion trigger from external peripheral: TIM6 TRGO event. Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_TIM7_TRGO

ADC group inject. conversion trigger from external peripheral: TIM7 TRGO event. Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_TIM8_TRGO

ADC group inject. conversion trigger from external peripheral: TIM8 TRGO event. Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_TIM8_TRGO2

ADC group inject. conversion trigger from external peripheral: TIM8 TRGO2 event. Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_TIM9_CH2

ADC group inject. conversion trigger from external peripheral: TIM9 channel 2 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_TIM9_TRGO

ADC group inject. conversion trigger from external peripheral: TIM9 TRGO event. Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_TIM12_TRGO

ADC group inject. conversion trigger from external peripheral: TIM12 TRGO event. Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_TIM15_TRGO

ADC group inject. conversion trigger from external peripheral: TIM12 TRGO event. Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_TIM18_TRGO

ADC group inject. conversion trigger from external peripheral: TIM12 TRGO event. Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_LPTIM1_CH2

ADC group inject. conversion trigger from external peripheral: LPTIM1 channel 2 event. Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_LPTIM2_CH2

ADC group inject. conversion trigger from external peripheral: LPTIM2 channel 2 event. Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_LPTIM3_CH2

ADC group inject. conversion trigger from external peripheral: LPTIM3 channel 2 event. Trigger edge set to rising edge (default setting).

ADC group injected - Automatic trigger mode

LL_ADC_INJ_TRIG_INDEPENDENT

ADC group injected conversion trigger independent. Setting mandatory if ADC group injected injected trigger source is set to an external trigger.

LL_ADC_INJ_TRIG_FROM_GRP_REGULAR

ADC group injected conversion trigger from ADC group regular. Setting compliant only with group injected trigger source set to SW start, without any further action on ADC group injected conversion start or stop: in this case, ADC group injected is controlled only from ADC group regular.

ADC interruptions for configuration (interruption enable or disable)

LL_ADC_IT_ADRDY

ADC interruption ADC instance ready

LL_ADC_IT_EOC

ADC interruption ADC group regular end of unitary conversion

LL_ADC_IT_EOS

ADC interruption ADC group regular end of sequence conversions

LL_ADC_IT_OVR

ADC interruption ADC group regular overrun

LL_ADC_IT_EOSMP

ADC interruption ADC group regular end of sampling phase

LL_ADC_IT_JEOC

ADC interruption ADC group injected end of unitary conversion

LL_ADC_IT_JEOS

ADC interruption ADC group injected end of sequence conversions

LL_ADC_IT_AWD1

ADC interruption ADC analog watchdog 1

LL_ADC_IT_AWD2

ADC interruption ADC analog watchdog 2

LL_ADC_IT_AWD3

ADC interruption ADC analog watchdog 3

ADC left Shift

LL_ADC_LEFT_BIT_SHIFT_NONE

ADC no bit shift left applied on the final ADC conversion data

LL_ADC_LEFT_BIT_SHIFT_1

ADC 1 bit shift left applied on the final ADC conversion data

LL_ADC_LEFT_BIT_SHIFT_2

ADC 2 bits shift left applied on the final ADC conversion data

LL_ADC_LEFT_BIT_SHIFT_3

ADC 3 bits shift left applied on the final ADC conversion data

LL_ADC_LEFT_BIT_SHIFT_4

ADC 4 bits shift left applied on the final ADC conversion data

LL_ADC_LEFT_BIT_SHIFT_5

ADC 5 bits shift left applied on the final ADC conversion data

LL_ADC_LEFT_BIT_SHIFT_6

ADC 6 bits shift left applied on the final ADC conversion data

LL_ADC_LEFT_BIT_SHIFT_7

ADC 7 bits shift left applied on the final ADC conversion data

LL_ADC_LEFT_BIT_SHIFT_8

ADC 8 bits shift left applied on the final ADC conversion data

LL_ADC_LEFT_BIT_SHIFT_9

ADC 9 bits shift left applied on the final ADC conversion data

LL_ADC_LEFT_BIT_SHIFT_10

ADC 10 bits shift left applied on the final ADC conversion data

LL_ADC_LEFT_BIT_SHIFT_11

ADC 11 bits shift left applied on the final ADC conversion data

LL_ADC_LEFT_BIT_SHIFT_12

ADC 12 bits shift left applied on the final ADC conversion data

LL_ADC_LEFT_BIT_SHIFT_13

ADC 13 bits shift left applied on the final ADC conversion data

LL_ADC_LEFT_BIT_SHIFT_14

ADC 14 bits shift left applied on the final ADC conversion data

LL_ADC_LEFT_BIT_SHIFT_15

ADC 15 bits shift left applied on the final ADC conversion data

ADC instance - Low power mode

LL_ADC_LP_AUTOWAIT_DISABLE

ADC low power mode auto delay disabled.

LL_ADC_LP_AUTOWAIT_ENABLE

ADC low power mode auto delay enabled: dynamic low power mode, ADC conversions are performed only when necessary (when previous ADC conversion data is read). See description with function

Multimode - Data format

LL_ADC_MULTI_REG_DATA_EACH_ADC

ADC multimode group regular data format: conversion data in data register of each ADC instance. If ADC data transfer by DMA is used: each ADC uses its own DMA channel, with its individual DMA transfer settings.

LL_ADC_MULTI_REG_DATA_COMMON_32B

ADC multimode group regular data format: conversion data in two ADC common instance data registers (CDR, CDR2) with packing option on 32 bit. In register CDR, data packing on 32 bit: ADC master and slave data are concatenated (data master in [15; 0], data slave in [31; 16]), therefore data width must be lower than 16 bit (even with ADC resolution 12 bit, higher width reachable by post processing: oversampling, offset, ...). In register CDR2, data of master and slave are alternatively set in full register width 32 bit, therefore no constraint on data width. In case of usage with DMA, CDR generate ones transfer request and CDR2 two transfer requests per conversion.

LL_ADC_MULTI_REG_DATA_COMMON_16B

ADC multimode group regular data format: conversion data in two ADC common instance data registers (CDR, CDR2) with packing option on 16 bit. In register CDR, data packing on 16 bit: ADC master and slave data are concatenated (data master in [7; 0], data slave in [15; 8]), therefore data width must be lower than 8 bit (even with ADC resolution 8 bit, higher width reachable by post processing: oversampling, offset, ...). In register CDR2, data of master and slave are alternatively set in full register width 32 bit, therefore no constraint on data width. In case of usage with DMA, CDR generate ones transfer request and CDR2 two transfer requests per conversion.

LL_ADC_MULTI_REG_DMA_EACH_ADC

LL_ADC_MULTI_REG_DMA_RES_32B

LL_ADC_MULTI_REG_DMA_RES_16B

Multimode - ADC master or slave

LL_ADC_MULTI_MASTER

In multimode, selection among several ADC instances: ADC master

LL_ADC_MULTI_SLAVE

In multimode, selection among several ADC instances: ADC slave

LL_ADC_MULTI_MASTER_SLAVE

In multimode, selection among several ADC instances: both ADC master and ADC slave

Multimode - Mode

LL_ADC_MULTI_INDEPENDENT

ADC dual mode disabled (ADC independent mode)

LL_ADC_MULTI_DUAL_REG_SIMULT

ADC dual mode enabled: group regular simultaneous

LL_ADC_MULTI_DUAL_REG_INTERL

ADC dual mode enabled: Combined group regular interleaved

LL_ADC_MULTI_DUAL_INJ_SIMULT

ADC dual mode enabled: group injected simultaneous

LL_ADC_MULTI_DUAL_INJ_ALTERN

ADC dual mode enabled: group injected alternate trigger. Works only with external triggers (not SW start)

LL_ADC_MULTI_DUAL_REG_SIM_INJ_SIM

ADC dual mode enabled: Combined group regular simultaneous + group injected simultaneous

LL_ADC_MULTI_DUAL_REG_SIM_INJ_ALT

ADC dual mode enabled: Combined group regular simultaneous + group injected alternate trigger

LL_ADC_MULTI_DUAL_REG_INT_INJ_SIM

ADC dual mode enabled: Combined group regular interleaved + group injected simultaneous

Multimode - Delay between two sampling phases

LL_ADC_MULTI_TWOSMP_DELAY_1CYCLE

ADC multimode interleaved delay between two sampling phases: 1 ADC clock cycle

LL_ADC_MULTI_TWOSMP_DELAY_2CYCLES

ADC multimode interleaved delay between two sampling phases: 2 ADC clock cycles

LL_ADC_MULTI_TWOSMP_DELAY_3CYCLES

ADC multimode interleaved delay between two sampling phases: 3 ADC clock cycles

LL_ADC_MULTI_TWOSMP_DELAY_4CYCLES

ADC multimode interleaved delay between two sampling phases: 4 ADC clock cycles

LL_ADC_MULTI_TWOSMP_DELAY_5CYCLES

ADC multimode interleaved delay between two sampling phases: 5 ADC clock cycles

LL_ADC_MULTI_TWOSMP_DELAY_6CYCLES

ADC multimode interleaved delay between two sampling phases: 6 ADC clock cycles

LL_ADC_MULTI_TWOSMP_DELAY_7CYCLES

ADC multimode interleaved delay between two sampling phases: 7 ADC clock cycles

LL_ADC_MULTI_TWOSMP_DELAY_8CYCLES

ADC multimode interleaved delay between two sampling phases: 8 ADC clock cycles

LL_ADC_MULTI_TWOSMP_DELAY_9CYCLES

ADC multimode interleaved delay between two sampling phases: 9 ADC clock cycles

LL_ADC_MULTI_TWOSMP_DELAY_10CYCLES

ADC multimode interleaved delay between two sampling phases: 10 ADC clock cycles

LL_ADC_MULTI_TWOSMP_DELAY_11CYCLES

ADC multimode interleaved delay between two sampling phases: 11 ADC clock cycles

LL_ADC_MULTI_TWOSMP_DELAY_12CYCLES

ADC multimode interleaved delay between two sampling phases: 12 ADC clock cycles

LL_ADC_MULTI_TWOSMP_DELAY_13CYCLES

ADC multimode interleaved delay between two sampling phases: 13 ADC clock cycles

ADC instance - Offset instance

LL_ADC_OFFSET_1

ADC offset instance 1: ADC channel and offset level to which the offset programmed will be applied (independently of channel mapped on ADC group regular or injected)

LL_ADC_OFFSET_2

ADC offset instance 2: ADC channel and offset level to which the offset programmed will be applied (independently of channel mapped on ADC group regular or injected)

LL_ADC_OFFSET_3

ADC offset instance 3: ADC channel and offset level to which the offset programmed will be applied (independently of channel mapped on ADC group regular or injected)

LL_ADC_OFFSET_4

ADC offset instance 4: ADC channel and offset level to which the offset programmed will be applied (independently of channel mapped on ADC group regular or injected)

ADC instance - Offset sign

LL_ADC_OFFSET_SIGN_NEGATIVE

ADC offset is negative

LL_ADC_OFFSET_SIGN_POSITIVE

ADC offset is positive

ADC instance - Offset signed saturation mode

LL_ADC_OFFSET_SIGNED_SAT_DISABLE

ADC offset signed saturation is disabled

LL_ADC_OFFSET_SIGNED_SAT_ENABLE

ADC offset signed saturation is enabled

ADC instance - Offset unsigned saturation mode

LL_ADC_OFFSET_UNSIGNED_SAT_DISABLE

ADC offset unsigned saturation is disabled

LL_ADC_OFFSET_UNSIGNED_SAT_ENABLE

ADC offset unsigned saturation is enabled

Oversampling - Discontinuous mode

LL_ADC_OVS_REG_CONT

ADC oversampling discontinuous mode: continuous mode (all conversions of oversampling ratio are done from 1 trigger)

LL_ADC_OVS_REG_DISCONT

ADC oversampling discontinuous mode: discontinuous mode (each conversion of oversampling ratio needs a trigger)

Oversampling - Oversampling scope

LL_ADC_OVS_DISABLE

ADC oversampling disabled.

LL_ADC_OVS_GRP_REGULAR_CONTINUED

ADC oversampling on conversions of ADC group regular. If group injected interrupts group regular: when ADC group injected is triggered, the oversampling on ADC group regular is temporary stopped and continued afterwards.

LL_ADC_OVS_GRP_REGULAR_RESUMED

ADC oversampling on conversions of ADC group regular. If group injected interrupts group regular: when ADC group injected is triggered, the oversampling on ADC group regular is resumed from start (oversampler buffer reset).

LL_ADC_OVS_GRP_INJECTED

ADC oversampling on conversions of ADC group injected.

LL_ADC_OVS_GRP_INJ_REG_RESUMED

ADC oversampling on conversions of both ADC groups regular and injected. If group injected interrupting group regular: when ADC group injected is triggered, the oversampling on ADC group regular is resumed from start (oversampler buffer reset).

Oversampling - Data right shift

LL_ADC_OVS_SHIFT_NONE

ADC oversampling no shift (sum of the ADC conversions data is not divided to result as oversampling conversion data)

LL_ADC_OVS_SHIFT_RIGHT_1

ADC oversampling right shift of 1 (sum of the ADC conversions data (after OVS ratio) is divided by 2 to result as oversampling conversion data)

LL_ADC_OVS_SHIFT_RIGHT_2

ADC oversampling right shift of 2 (sum of the ADC conversions data (after OVS ratio) is divided by 4 to result as oversampling conversion data)

LL_ADC_OVS_SHIFT_RIGHT_3

ADC oversampling right shift of 3 (sum of the ADC conversions data (after OVS ratio) is divided by 8 to result as oversampling conversion data)

LL_ADC_OVS_SHIFT_RIGHT_4

ADC oversampling right shift of 4 (sum of the ADC conversions data (after OVS ratio) is divided by 16 to result as oversampling conversion data)

LL_ADC_OVS_SHIFT_RIGHT_5

ADC oversampling right shift of 5 (sum of the ADC conversions data (after OVS ratio) is divided by 32 to result as oversampling conversion data)

LL_ADC_OVS_SHIFT_RIGHT_6

ADC oversampling right shift of 6 (sum of the ADC conversions data (after OVS ratio) is divided by 64 to result as oversampling conversion data)

LL_ADC_OVS_SHIFT_RIGHT_7

ADC oversampling right shift of 7 (sum of the ADC conversions data (after OVS ratio) is divided by 128 to result as oversampling conversion data)

LL_ADC_OVS_SHIFT_RIGHT_8

ADC oversampling right shift of 8 (sum of the ADC conversions data (after OVS ratio) is divided by 256 to result as oversampling conversion data)

LL_ADC_OVS_SHIFT_RIGHT_9

ADC oversampling right shift of 8 (sum of the ADC conversions data (after OVS ratio) is divided by 512 to result as oversampling conversion data)

LL_ADC_OVS_SHIFT_RIGHT_10

ADC oversampling right shift of 8 (sum of the ADC conversions data (after OVS ratio) is divided by 1024 to result as oversampling conversion data)

ADC instance - Measurement path to internal channels

LL_ADC_PATH_INTERNAL_VDDCORE

ADC measurement path to internal channel VddCore

ADC registers compliant with specific purpose

LL_ADC_DMA_REG_REGULAR_DATA

ADC group regular conversion data register (corresponding to register DR) to be used with ADC configured in independent mode. Without DMA transfer, register accessed by LL function

LL_ADC_DMA_REG_REGULAR_MULTI_NO_PACKING

ADC group regular conversion data register to be used with ADC configured in multimode (available on STM32 devices with several ADC instances), without data packing. Register used is CDR2, compliant with all ADC multimode data format and data width, refer to description of literals in

LL_ADC_DMA_REG_REGULAR_MULTI_PACKING

ADC group regular conversion data register to be used with ADC configured in multimode (available on STM32 devices with several ADC instances), with data packing. Register used is CDR, usable under conditions of ADC multimode data format selected and data width, refer to description of literals in

ADC group regular - Continuous mode

LL_ADC_REG_CONV_SINGLE

ADC conversions performed in single mode: one conversion per trigger

LL_ADC_REG_CONV_CONTINUOUS

ADC conversions performed in continuous mode: after the first trigger, following conversions launched successively automatically

ADC group regular - Data transfer mode of ADC conversion data

LL_ADC_REG_DR_TRANSFER

ADC conversions data are available in ADC data register only

LL_ADC_REG_DMA_TRANSFER_LIMITED

ADC conversion data are transferred by DMA, in limited mode (one shot mode): DMA transfer requests are stopped when number of DMA data transfers (number of ADC conversions) is reached. This ADC mode is intended to be used with DMA mode non-circular.

LL_ADC_REG_DMA_TRANSFER_UNLIMITED

ADC conversion data are transferred by DMA, in unlimited mode: DMA transfer requests are unlimited, whatever number of DMA data transferred (number of ADC conversions). This ADC mode is intended to be used with DMA mode circular.

LL_ADC_REG_MDF_TRANSFER

ADC conversion data are transferred to MDF

ADC group regular - Overrun behavior on conversion data

LL_ADC_REG_OVR_DATA_PRESERVED

ADC group regular behavior in case of overrun: data preserved

LL_ADC_REG_OVR_DATA_OVERWRITTEN

ADC group regular behavior in case of overrun: data overwritten

ADC group regular - Sampling mode

LL_ADC_REG_SAMPLING_MODE_NORMAL

ADC conversions sampling phase duration is defined using

LL_ADC_REG_SAMPLING_MODE_BULB

ADC conversions sampling phase starts immediately after end of conversion, and stops upon trigger event. Note: First conversion is using minimal sampling time (see

LL_ADC_REG_SAMPLING_MODE_TRIGGER_CONTROLED

ADC conversions sampling phase is controlled by trigger events: Trigger rising edge = start sampling, Trigger falling edge = stop sampling and start conversion.

ADC group regular - Sequencer discontinuous mode

LL_ADC_REG_SEQ_DISCONT_DISABLE

ADC group regular sequencer discontinuous mode disable

LL_ADC_REG_SEQ_DISCONT_1RANK

ADC group regular sequencer discontinuous mode enable with sequence interruption every rank

LL_ADC_REG_SEQ_DISCONT_2RANKS

ADC group regular sequencer discontinuous mode enabled with sequence interruption every 2 ranks

LL_ADC_REG_SEQ_DISCONT_3RANKS

ADC group regular sequencer discontinuous mode enable with sequence interruption every 3 ranks

LL_ADC_REG_SEQ_DISCONT_4RANKS

ADC group regular sequencer discontinuous mode enable with sequence interruption every 4 ranks

LL_ADC_REG_SEQ_DISCONT_5RANKS

ADC group regular sequencer discontinuous mode enable with sequence interruption every 5 ranks

LL_ADC_REG_SEQ_DISCONT_6RANKS

ADC group regular sequencer discontinuous mode enable with sequence interruption every 6 ranks

LL_ADC_REG_SEQ_DISCONT_7RANKS

ADC group regular sequencer discontinuous mode enable with sequence interruption every 7 ranks

LL_ADC_REG_SEQ_DISCONT_8RANKS

ADC group regular sequencer discontinuous mode enable with sequence interruption every 8 ranks

ADC group regular - Sequencer ranks

LL_ADC_REG_RANK_1

ADC group regular sequencer rank 1

LL_ADC_REG_RANK_2

ADC group regular sequencer rank 2

LL_ADC_REG_RANK_3

ADC group regular sequencer rank 3

LL_ADC_REG_RANK_4

ADC group regular sequencer rank 4

LL_ADC_REG_RANK_5

ADC group regular sequencer rank 5

LL_ADC_REG_RANK_6

ADC group regular sequencer rank 6

LL_ADC_REG_RANK_7

ADC group regular sequencer rank 7

LL_ADC_REG_RANK_8

ADC group regular sequencer rank 8

LL_ADC_REG_RANK_9

ADC group regular sequencer rank 9

LL_ADC_REG_RANK_10

ADC group regular sequencer rank 10

LL_ADC_REG_RANK_11

ADC group regular sequencer rank 11

LL_ADC_REG_RANK_12

ADC group regular sequencer rank 12

LL_ADC_REG_RANK_13

ADC group regular sequencer rank 13

LL_ADC_REG_RANK_14

ADC group regular sequencer rank 14

LL_ADC_REG_RANK_15

ADC group regular sequencer rank 15

LL_ADC_REG_RANK_16

ADC group regular sequencer rank 16

ADC group regular - Sequencer scan length

LL_ADC_REG_SEQ_SCAN_DISABLE

ADC group regular sequencer disable (equivalent to sequencer of 1 rank: ADC conversion on only 1 channel)

LL_ADC_REG_SEQ_SCAN_ENABLE_2RANKS

ADC group regular sequencer enable with 2 ranks in the sequence

LL_ADC_REG_SEQ_SCAN_ENABLE_3RANKS

ADC group regular sequencer enable with 3 ranks in the sequence

LL_ADC_REG_SEQ_SCAN_ENABLE_4RANKS

ADC group regular sequencer enable with 4 ranks in the sequence

LL_ADC_REG_SEQ_SCAN_ENABLE_5RANKS

ADC group regular sequencer enable with 5 ranks in the sequence

LL_ADC_REG_SEQ_SCAN_ENABLE_6RANKS

ADC group regular sequencer enable with 6 ranks in the sequence

LL_ADC_REG_SEQ_SCAN_ENABLE_7RANKS

ADC group regular sequencer enable with 7 ranks in the sequence

LL_ADC_REG_SEQ_SCAN_ENABLE_8RANKS

ADC group regular sequencer enable with 8 ranks in the sequence

LL_ADC_REG_SEQ_SCAN_ENABLE_9RANKS

ADC group regular sequencer enable with 9 ranks in the sequence

LL_ADC_REG_SEQ_SCAN_ENABLE_10RANKS

ADC group regular sequencer enable with 10 ranks in the sequence

LL_ADC_REG_SEQ_SCAN_ENABLE_11RANKS

ADC group regular sequencer enable with 11 ranks in the sequence

LL_ADC_REG_SEQ_SCAN_ENABLE_12RANKS

ADC group regular sequencer enable with 12 ranks in the sequence

LL_ADC_REG_SEQ_SCAN_ENABLE_13RANKS

ADC group regular sequencer enable with 13 ranks in the sequence

LL_ADC_REG_SEQ_SCAN_ENABLE_14RANKS

ADC group regular sequencer enable with 14 ranks in the sequence

LL_ADC_REG_SEQ_SCAN_ENABLE_15RANKS

ADC group regular sequencerenable with 15 ranks in the sequence

LL_ADC_REG_SEQ_SCAN_ENABLE_16RANKS

ADC group regular sequencer enable with 16 ranks in the sequence

ADC group regular - Trigger edge

LL_ADC_REG_TRIG_EXT_RISING

ADC group regular conversion trigger polarity set to rising edge

LL_ADC_REG_TRIG_EXT_FALLING

ADC group regular conversion trigger polarity set to falling edge

LL_ADC_REG_TRIG_EXT_RISINGFALLING

ADC group regular conversion trigger polarity set to both rising and falling edges

ADC group regular - Trigger source

LL_ADC_REG_TRIG_SOFTWARE

ADC group regular conversion trigger internal: SW start.

LL_ADC_REG_TRIG_EXT_EXTI_LINE11

ADC group regular conversion trigger from external peripheral: external interrupt line 11. Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM1_CH1

ADC group regular conversion trigger from external peripheral: TIM1 channel 1 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM1_CH2

ADC group regular conversion trigger from external peripheral: TIM1 channel 2 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM1_CH3

ADC group regular conversion trigger from external peripheral: TIM1 channel 3 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM1_TRGO

ADC group regular conversion trigger from external peripheral: TIM1 TRGO event. Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM1_TRGO2

ADC group regular conversion trigger from external peripheral: TIM1 TRGO2 event. Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM2_CH2

ADC group regular conversion trigger from external peripheral: TIM2 channel 2 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM2_TRGO

ADC group regular conversion trigger from external peripheral: TIM2 TRGO event. Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM3_CH4

ADC group regular conversion trigger from external peripheral: TIM3 channel 4 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM3_TRGO

ADC group regular conversion trigger from external peripheral: TIM3 TRGO event. Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM4_CH4

ADC group regular conversion trigger from external peripheral: TIM4 channel 4 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM4_TRGO

ADC group regular conversion trigger from external peripheral: TIM4 TRGO event. Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM5_TRGO

ADC group regular conversion trigger from external peripheral: TIM5 TRGO event. Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM6_TRGO

ADC group regular conversion trigger from external peripheral: TIM6 TRGO event. Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM7_TRGO

ADC group regular conversion trigger from external peripheral: TIM7 TRGO event. Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM8_TRGO

ADC group regular conversion trigger from external peripheral: TIM8 TRGO event. Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM8_TRGO2

ADC group regular conversion trigger from external peripheral: TIM8 TRGO2 event. Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM9_CH1

ADC group regular conversion trigger from external peripheral: TIM9 channel 1 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM9_TRGO

ADC group regular conversion trigger from external peripheral: TIM9 TRGO event. Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM12_TRGO

ADC group regular conversion trigger from external peripheral: TIM12 TRGO event. Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM15_TRGO

ADC group regular conversion trigger from external peripheral: TIM15 TRGO event. Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM18_TRGO

ADC group regular conversion trigger from external peripheral: TIM18 TRGO event. Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_LPTIM1_CH1

ADC group regular conversion trigger from external peripheral: LPTIM1 channel 1 event. Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_LPTIM2_CH1

ADC group regular conversion trigger from external peripheral: LPTIM2 channel 2 event. Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_LPTIM3_CH1

ADC group regular conversion trigger from external peripheral: LPTIM3 channel 3 event. Trigger edge set to rising edge (default setting).

ADC instance - Resolution

LL_ADC_RESOLUTION_12B

ADC resolution 12 bits

LL_ADC_RESOLUTION_10B

ADC resolution 10 bits

LL_ADC_RESOLUTION_8B

ADC resolution 8 bits

LL_ADC_RESOLUTION_6B

ADC resolution 6 bits

ADC helper macro

__LL_ADC_CHANNEL_TO_DECIMAL_NB

Description:

- Helper macro to get ADC channel number in decimal format from literals LL_ADC_CHANNEL_x.

Parameters:

- __CHANNEL__: This parameter can be one of the following values:
  - LL_ADC_CHANNEL_0
  - LL_ADC_CHANNEL_1
  - LL_ADC_CHANNEL_2
  - LL_ADC_CHANNEL_3
  - LL_ADC_CHANNEL_4
  - LL_ADC_CHANNEL_5
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_19
  - LL_ADC_CHANNEL_VREFINT (1)
  - LL_ADC_CHANNEL_VBAT (2)
  - LL_ADC_CHANNEL_VDDCORE (2)

Return value:

- Value: between Min_Data=0 and Max_Data=18

Notes:

- Example: __LL_ADC_CHANNEL_TO_DECIMAL_NB(LL_ADC_CHANNEL_4) will return decimal number "4". The input can be a value from functions where a channel number is returned, either defined with number or with bitfield (only one bit must be set).

__LL_ADC_DECIMAL_NB_TO_CHANNEL

Description:

- Helper macro to get ADC channel in literal format LL_ADC_CHANNEL_x from number in decimal format.

Parameters:

- __DECIMAL_NB__: Value between Min_Data=0 and Max_Data=18

Return value:

- Returned: value can be one of the following values:
  - LL_ADC_CHANNEL_0
  - LL_ADC_CHANNEL_1
  - LL_ADC_CHANNEL_2
  - LL_ADC_CHANNEL_3
  - LL_ADC_CHANNEL_4
  - LL_ADC_CHANNEL_5
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_19
  - LL_ADC_CHANNEL_VREFINT (1)(3)
  - LL_ADC_CHANNEL_VBAT (2)(3)
  - LL_ADC_CHANNEL_VDDCORE (2)(3)

Notes:

- Example: __LL_ADC_DECIMAL_NB_TO_CHANNEL(4) will return a data equivalent to "LL_ADC_CHANNEL_4".

__LL_ADC_IS_CHANNEL_INTERNAL

Description:

- Helper macro to determine whether the selected channel corresponds to literal definitions of driver.

Parameters:

- __CHANNEL__: This parameter can be one of the following values:
  - LL_ADC_CHANNEL_0
  - LL_ADC_CHANNEL_1
  - LL_ADC_CHANNEL_2
  - LL_ADC_CHANNEL_3
  - LL_ADC_CHANNEL_4
  - LL_ADC_CHANNEL_5
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_19
  - LL_ADC_CHANNEL_VREFINT (1)
  - LL_ADC_CHANNEL_VBAT (2)
  - LL_ADC_CHANNEL_VDDCORE (2)

Return value:

- Value: "0" if the channel corresponds to a parameter definition of a ADC external channel (channel connected to a GPIO pin). Value "1" if the channel corresponds to a parameter definition of a ADC internal channel.

Notes:

- The different literal definitions of ADC channels are: ADC internal channel: LL_ADC_CHANNEL_VREFINT, ...ADC external channel (channel connected to a GPIO pin): LL_ADC_CHANNEL_1, LL_ADC_CHANNEL_2, ... The channel parameter must be a value defined from literal definition of a ADC internal channel (LL_ADC_CHANNEL_VREFINT, ...), ADC external channel (LL_ADC_CHANNEL_1, LL_ADC_CHANNEL_2, ...), must not be a value from functions where a channel number is returned from ADC registers, because internal and external channels share the same channel number in ADC registers. The differentiation is made only with parameters definitions of driver.

__LL_ADC_CHANNEL_INTERNAL_TO_EXTERNAL

Description:

- Helper macro to convert a channel defined from parameter definition of a ADC internal channel (LL_ADC_CHANNEL_VREFINT, ...), to its equivalent parameter definition of a ADC external channel (LL_ADC_CHANNEL_1, LL_ADC_CHANNEL_2, ...).

Parameters:

- __CHANNEL__: This parameter can be one of the following values:
  - LL_ADC_CHANNEL_0
  - LL_ADC_CHANNEL_1
  - LL_ADC_CHANNEL_2
  - LL_ADC_CHANNEL_3
  - LL_ADC_CHANNEL_4
  - LL_ADC_CHANNEL_5
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_19
  - LL_ADC_CHANNEL_VREFINT (1)
  - LL_ADC_CHANNEL_VBAT (2)
  - LL_ADC_CHANNEL_VDDCORE (2)

Return value:

- Returned: value can be one of the following values:
  - LL_ADC_CHANNEL_0
  - LL_ADC_CHANNEL_1
  - LL_ADC_CHANNEL_2
  - LL_ADC_CHANNEL_3
  - LL_ADC_CHANNEL_4
  - LL_ADC_CHANNEL_5
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_19

Notes:

- The channel parameter can be, additionally to a value defined from parameter definition of a ADC internal channel (LL_ADC_CHANNEL_VREFINT, ...), a value defined from parameter definition of ADC external channel (LL_ADC_CHANNEL_1, LL_ADC_CHANNEL_2, ...) or a value from functions where a channel number is returned from ADC registers.

__LL_ADC_IS_CHANNEL_INTERNAL_AVAILABLE

Description:

- Helper macro to determine whether the internal channel selected is available on the ADC instance selected.

Parameters:

- __ADC_INSTANCE__: ADC instance
- __CHANNEL__: This parameter can be one of the following values:
  - LL_ADC_CHANNEL_VREFINT (1)
  - LL_ADC_CHANNEL_VBAT (2)
  - LL_ADC_CHANNEL_VDDCORE (2)

Return value:

- Value: "0" if the internal channel selected is not available on the ADC instance selected. Value "1" if the internal channel selected is available on the ADC instance selected.

Notes:

- The channel parameter must be a value defined from parameter definition of a ADC internal channel (LL_ADC_CHANNEL_VREFINT, ...), must not be a value defined from parameter definition of ADC external channel (LL_ADC_CHANNEL_1, LL_ADC_CHANNEL_2, ...) or a value from functions where a channel number is returned from ADC registers, because internal and external channels share the same channel number in ADC registers. The differentiation is made only with parameters definitions of driver.

__LL_ADC_CHANNEL_DIFF_NEG_INPUT

Description:

- Helper macro to determine the selected channel corresponding negative input on the ADC instance selected.

Parameters:

- __ADC_INSTANCE__: ADC instance
- __CHANNEL__: This parameter can be one of the following values:
  - LL_ADC_CHANNEL_0
  - LL_ADC_CHANNEL_1
  - LL_ADC_CHANNEL_2
  - LL_ADC_CHANNEL_3
  - LL_ADC_CHANNEL_4
  - LL_ADC_CHANNEL_5
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_19
  - LL_ADC_CHANNEL_VREFINT (1)
  - LL_ADC_CHANNEL_VBAT (2)
  - LL_ADC_CHANNEL_VDDCORE (2)

Return value:

- Returned: value can be one of the following values:
  - LL_ADC_CHANNEL_0
  - LL_ADC_CHANNEL_1
  - LL_ADC_CHANNEL_2
  - LL_ADC_CHANNEL_3
  - LL_ADC_CHANNEL_4
  - LL_ADC_CHANNEL_5
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_19
  - LL_ADC_CHANNEL_VREFINT (1)(3)
  - LL_ADC_CHANNEL_VBAT (2)(3)
  - LL_ADC_CHANNEL_VDDCORE (2)(3)

__LL_ADC_ANALOGWD_CHANNEL_GROUP

Description:

- Helper macro to define ADC analog watchdog parameter: define a single channel to monitor with analog watchdog from sequencer channel and groups definition.

Parameters:

- __CHANNEL__: This parameter can be one of the following values:
  - LL_ADC_CHANNEL_0
  - LL_ADC_CHANNEL_1
  - LL_ADC_CHANNEL_2
  - LL_ADC_CHANNEL_3
  - LL_ADC_CHANNEL_4
  - LL_ADC_CHANNEL_5
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_19
  - LL_ADC_CHANNEL_VREFINT (1)(3)
  - LL_ADC_CHANNEL_VBAT (2)(3)
  - LL_ADC_CHANNEL_VDDCORE (2)(3)
- __GROUP__: This parameter can be one of the following values:
  - LL_ADC_GROUP_REGULAR
  - LL_ADC_GROUP_INJECTED
  - LL_ADC_GROUP_REGULAR_INJECTED

Return value:

- Returned: value can be one of the following values:
  - LL_ADC_AWD_DISABLE
  - LL_ADC_AWD_ALL_CHANNELS_REG (0)
  - LL_ADC_AWD_ALL_CHANNELS_INJ (0)
  - LL_ADC_AWD_ALL_CHANNELS_REG_INJ
  - LL_ADC_AWD_CHANNEL_0_REG (0)
  - LL_ADC_AWD_CHANNEL_0_INJ (0)
  - LL_ADC_AWD_CHANNEL_0_REG_INJ
  - LL_ADC_AWD_CHANNEL_1_REG (0)
  - LL_ADC_AWD_CHANNEL_1_INJ (0)
  - LL_ADC_AWD_CHANNEL_1_REG_INJ
  - LL_ADC_AWD_CHANNEL_2_REG (0)
  - LL_ADC_AWD_CHANNEL_2_INJ (0)
  - LL_ADC_AWD_CHANNEL_2_REG_INJ
  - LL_ADC_AWD_CHANNEL_3_REG (0)
  - LL_ADC_AWD_CHANNEL_3_INJ (0)
  - LL_ADC_AWD_CHANNEL_3_REG_INJ
  - LL_ADC_AWD_CHANNEL_4_REG (0)
  - LL_ADC_AWD_CHANNEL_4_INJ (0)
  - LL_ADC_AWD_CHANNEL_4_REG_INJ
  - LL_ADC_AWD_CHANNEL_5_REG (0)
  - LL_ADC_AWD_CHANNEL_5_INJ (0)
  - LL_ADC_AWD_CHANNEL_5_REG_INJ
  - LL_ADC_AWD_CHANNEL_6_REG (0)
  - LL_ADC_AWD_CHANNEL_6_INJ (0)
  - LL_ADC_AWD_CHANNEL_6_REG_INJ
  - LL_ADC_AWD_CHANNEL_7_REG (0)
  - LL_ADC_AWD_CHANNEL_7_INJ (0)
  - LL_ADC_AWD_CHANNEL_7_REG_INJ
  - LL_ADC_AWD_CHANNEL_8_REG (0)
  - LL_ADC_AWD_CHANNEL_8_INJ (0)
  - LL_ADC_AWD_CHANNEL_8_REG_INJ
  - LL_ADC_AWD_CHANNEL_9_REG (0)
  - LL_ADC_AWD_CHANNEL_9_INJ (0)
  - LL_ADC_AWD_CHANNEL_9_REG_INJ
  - LL_ADC_AWD_CHANNEL_10_REG (0)
  - LL_ADC_AWD_CHANNEL_10_INJ (0)
  - LL_ADC_AWD_CHANNEL_10_REG_INJ
  - LL_ADC_AWD_CHANNEL_11_REG (0)
  - LL_ADC_AWD_CHANNEL_11_INJ (0)
  - LL_ADC_AWD_CHANNEL_11_REG_INJ
  - LL_ADC_AWD_CHANNEL_12_REG (0)
  - LL_ADC_AWD_CHANNEL_12_INJ (0)
  - LL_ADC_AWD_CHANNEL_12_REG_INJ
  - LL_ADC_AWD_CHANNEL_13_REG (0)
  - LL_ADC_AWD_CHANNEL_13_INJ (0)
  - LL_ADC_AWD_CHANNEL_13_REG_INJ
  - LL_ADC_AWD_CHANNEL_14_REG (0)
  - LL_ADC_AWD_CHANNEL_14_INJ (0)

- LL_ADC_AWD_CHANNEL_14_REG_INJ
- LL_ADC_AWD_CHANNEL_15_REG (0)
- LL_ADC_AWD_CHANNEL_15_INJ (0)
- LL_ADC_AWD_CHANNEL_15_REG_INJ
- LL_ADC_AWD_CHANNEL_16_REG (0)
- LL_ADC_AWD_CHANNEL_16_INJ (0)
- LL_ADC_AWD_CHANNEL_16_REG_INJ
- LL_ADC_AWD_CHANNEL_17_REG (0)
- LL_ADC_AWD_CHANNEL_17_INJ (0)
- LL_ADC_AWD_CHANNEL_17_REG_INJ
- LL_ADC_AWD_CHANNEL_18_REG (0)
- LL_ADC_AWD_CHANNEL_18_INJ (0)
- LL_ADC_AWD_CHANNEL_18_REG_INJ
- LL_ADC_AWD_CHANNEL_19_REG (0)
- LL_ADC_AWD_CHANNEL_19_INJ (0)
- LL_ADC_AWD_CHANNEL_19_REG_INJ
- LL_ADC_AWD_CH_VREFINT_REG (0)
- LL_ADC_AWD_CH_VREFINT_INJ (0)
- LL_ADC_AWD_CH_VREFINT_REG_INJ
- LL_ADC_AWD_CH_VBAT_REG (0)(1)
- LL_ADC_AWD_CH_VBAT_INJ (0)(1)
- LL_ADC_AWD_CH_VBAT_REG_INJ (1)
- LL_ADC_AWD_CH_VDDCORE_REG (0)(1)
- LL_ADC_AWD_CH_VDDCORE_INJ (0)(1)
- LL_ADC_AWD_CH_VDDCORE_REG_INJ (1)

Notes:

- To be used with function LL_ADC_SetAnalogWDMonitChannels(). Example: LL_ADC_SetAnalogWDMonitChannels(ADC1, LL_ADC_AWD1, __LL_ADC_ANALOGWD_CHANNEL_GROUP(LL_ADC_CHANNEL4, LL_ADC_GROUP_REGULAR))

__LL_ADC_ANALOGWD_SET_THRESHOLD_RESOLUTION

Description:

- Helper macro to set the value of ADC analog watchdog threshold high or low in function of ADC resolution when ADC resolution is different of 12 bit.

Parameters:

- __ADC_RESOLUTION__: This parameter can be one of the following values:
  - LL_ADC_RESOLUTION_12B
  - LL_ADC_RESOLUTION_10B
  - LL_ADC_RESOLUTION_8B
  - LL_ADC_RESOLUTION_6B
- __AWD_THRESHOLD_DATA__: Value between Min_Data=0x000 and Max_Data=0xFFF (or tigher range for data corresponding to lower ADC resolution)

Return value:

- Value: between Min_Data=0x000 and Max_Data=0xFFF

Notes:

- In case of ADC resolution different of 12 bits, this macro performs the required data formatting: analog watchdog thresholds data aligned to left side (bit 11).bits out of resolution range (LSB) set to value "0". To be used with function LL_ADC_SetAnalogWDThresholds(). Example, with a ADC resolution of 8 bits, to set the value of analog watchdog threshold high (on 8 bits): LL_ADC_SetAnalogWDThresholds (< ADCx param >, __LL_ADC_ANALOGWD_SET_THRESHOLD_RESOLUTION(LL_ADC_RESOLUTION_8B, <threshold_data_8_bits>));

__LL_ADC_ANALOGWD_GET_THRESHOLD_RESOLUTION

Description:

- Helper macro to get the value of ADC analog watchdog threshold high or low in function of ADC resolution, when ADC resolution is different of 12 bits.

Parameters:

- __ADC_RESOLUTION__: This parameter can be one of the following values:
  - LL_ADC_RESOLUTION_12B
  - LL_ADC_RESOLUTION_10B
  - LL_ADC_RESOLUTION_8B
  - LL_ADC_RESOLUTION_6B
- __AWD_THRESHOLD_12_BITS__: Value between Min_Data=0x000 and Max_Data=0xFFF

Return value:

- Value: between Min_Data=0x000 and Max_Data=0xFFF

Notes:

- To be used with function LL_ADC_GetAnalogWDThresholds(). Example, with a ADC resolution of 8 bits, to get the value of analog watchdog threshold high (on 8 bits): < threshold_value_6_bits > = __LL_ADC_ANALOGWD_GET_THRESHOLD_RESOLUTION (LL_ADC_RESOLUTION_8B, LL_ADC_GetAnalogWDThresholds(<ADCx param>, LL_ADC_AWD_THRESHOLD_HIGH));

__LL_ADC_CALIB_FACTOR_SINGLE_DIFF

Description:

- Helper macro to set the ADC calibration value with both single ended and differential modes calibration factors concatenated.

Parameters:

- __CALIB_FACTOR_SINGLE_ENDED__: Value between Min_Data=0x00 and Max_Data=0x1FF
- __CALIB_FACTOR_DIFFERENTIAL__: Value between Min_Data=0x00 and Max_Data=0x1FF

Return value:

- Value: between Min_Data=0x00000000 and Max_Data=0xFFFFFFFF

Notes:

- To be used with function LL_ADC_SetCalibrationFactor(). Example, to set calibration factors single ended to 0x55 and differential ended to 0x2A: LL_ADC_SetCalibrationFactor(ADC1, __LL_ADC_CALIB_FACTOR_SINGLE_DIFF(0x55, 0x2A))

__LL_ADC_MULTI_CONV_DATA_MASTER_SLAVE

Description:

- Helper macro to get the ADC multimode conversion data of ADC master or ADC slave from raw value with both ADC conversion data concatenated.

Parameters:

- __ADC_MULTI_MASTER_SLAVE__: This parameter can be one of the following values:
  - LL_ADC_MULTI_MASTER
  - LL_ADC_MULTI_SLAVE
- __ADC_MULTI_CONV_DATA__: Value between Min_Data=0x000 and Max_Data=0xFFF

Return value:

- Value: between Min_Data=0x000 and Max_Data=0xFFF

Notes:

- This macro is intended to be used when multimode transfer by DMA is enabled: refer to function LL_ADC_SetMultiDataFormat(). In this case the transferred data need to processed with this macro to separate the conversion data of ADC master and ADC slave.

__LL_ADC_MULTI_INSTANCE_MASTER

Description:

- Helper macro to select, from a ADC instance, to which ADC instance it has a dependence in multimode (ADC master of the corresponding ADC common instance).

Parameters:

- __ADCx__: ADC instance

Return value:

- __ADCx__: ADC instance master of the corresponding ADC common instance

Notes:

- In case of device with multimode available and a mix of ADC instances compliant and not compliant with multimode feature, ADC instances not compliant with multimode feature are considered as master instances (do not depend to any other ADC instance).

__LL_ADC_COMMON_INSTANCE

Description:

- Helper macro to select the ADC common instance to which is belonging the selected ADC instance.

Parameters:

- __ADCx__: ADC instance

Return value:

- ADC: common register instance

Notes:

- ADC common register instance can be used for: Set parameters common to several ADC instancesMultimode (for devices with several ADC instances) Refer to functions having argument "ADCxy_COMMON" as parameter.

__LL_ADC_IS_ENABLED_ALL_COMMON_INSTANCE

Description:

- Helper macro to check if all ADC instances sharing the same ADC common instance are disabled.

Parameters:

- __ADCXY_COMMON__: ADC common instance (can be set directly from CMSIS definition or by using helper macro

Return value:

- Value: "0" if all ADC instances sharing the same ADC common instance are disabled. Value "1" if at least one ADC instance sharing the same ADC common instance is enabled.

Notes:

- This check is required by functions with setting conditioned to ADC state: All ADC instances of the ADC common group must be disabled. Refer to functions having argument "ADCxy_COMMON" as parameter. On devices with only 1 ADC common instance, parameter of this macro is useless and can be ignored (parameter kept for compatibility with devices featuring several ADC common instances).

__LL_ADC_DIGITAL_SCALE

Description:

- Helper macro to define the ADC conversion data full-scale digital value corresponding to the selected ADC resolution.

Parameters:

- __ADC_RESOLUTION__: This parameter can be one of the following values:
  - LL_ADC_RESOLUTION_12B
  - LL_ADC_RESOLUTION_10B
  - LL_ADC_RESOLUTION_8B
  - LL_ADC_RESOLUTION_6B

Return value:

- ADC: conversion data full-scale digital value (unit: digital value of ADC conversion data)

Notes:

- ADC conversion data full-scale corresponds to voltage range determined by analog voltage references Vref+ and Vref- (refer to reference manual).

__LL_ADC_CONVERT_DATA_RESOLUTION

Description:

- Helper macro to convert the ADC conversion data from a resolution to another resolution.

Parameters:

- __DATA__: ADC conversion data to be converted
- __ADC_RESOLUTION_CURRENT__: Resolution of the data to be converted This parameter can be one of the following values:
  - LL_ADC_RESOLUTION_12B
  - LL_ADC_RESOLUTION_10B
  - LL_ADC_RESOLUTION_8B
  - LL_ADC_RESOLUTION_6B
- __ADC_RESOLUTION_TARGET__: Resolution of the data after conversion This parameter can be one of the following values:
  - LL_ADC_RESOLUTION_12B
  - LL_ADC_RESOLUTION_10B
  - LL_ADC_RESOLUTION_8B
  - LL_ADC_RESOLUTION_6B

Return value:

- ADC: conversion data to the requested resolution

__LL_ADC_CALC_DATA_TO_VOLTAGE

Description:

- Helper macro to calculate the voltage (unit: mVolt) corresponding to a ADC conversion data (unit: digital value).

Parameters:

- __VREFANALOG_VOLTAGE__: Analog reference voltage (unit: mV)
- __ADC_DATA__: ADC conversion data (resolution 12 bits) (unit: digital value).
- __ADC_RESOLUTION__: This parameter can be one of the following values:
  - LL_ADC_RESOLUTION_12B
  - LL_ADC_RESOLUTION_10B
  - LL_ADC_RESOLUTION_8B
  - LL_ADC_RESOLUTION_6B

Return value:

- ADC: conversion data equivalent voltage value (unit: mVolt)

Notes:

- Analog reference voltage (Vref+) must be either known from user board environment.

__LL_ADC_CALC_DIFF_DATA_TO_VOLTAGE

Description:

- Helper macro to calculate the voltage (unit: mVolt) corresponding to a ADC conversion data (unit: digital value) in differential ended mode.

Parameters:

- __VREFANALOG_VOLTAGE__: Analog reference voltage (unit: mV)
- __ADC_DATA__: ADC conversion data (unit: digital value).
- __ADC_RESOLUTION__: This parameter can be one of the following values:
  - LL_ADC_RESOLUTION_12B
  - LL_ADC_RESOLUTION_10B
  - LL_ADC_RESOLUTION_8B
  - LL_ADC_RESOLUTION_6B

Return value:

- ADC: conversion data equivalent voltage value (unit: mVolt)

Notes:

- ADC data from ADC data register is unsigned and centered around middle code in. Converted voltage can be positive or negative depending on differential input voltages. Analog reference voltage (Vref+) must be either known from user board environment.

Common write and read registers Macros

LL_ADC_WriteReg

Description:

- Write a value in ADC register.

Parameters:

- __INSTANCE__: ADC Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_ADC_ReadReg

Description:

- Read a value in ADC register.

Parameters:

- __INSTANCE__: ADC Instance
- __REG__: Register to be read

Return value:

- Register: value

ADC Exported Constants

LL_ADC_LP_MODE_NONE

LL_ADC_LP_AUTOWAIT

86 LL BUS Generic Driver

86.1 BUS Firmware driver API description

The following section lists the various functions of the BUS library.

86.1.1 Detailed description of functions

LL_BUS_EnableClock

Function name

__STATIC_INLINE void LL_BUS_EnableClock (uint32_t Bus)

Function description

Enable AXI bus clock.

Parameters

- **Bus:** This parameter can be a combination of the following values:
  - LL_ACLKN
  - LL_ACLKNC
  - LL_AHBM
  - LL_AHB1
  - LL_AHB2
  - LL_AHB3
  - LL_AHB4
  - LL_AHB5
  - LL_APB1
  - LL_APB2
  - LL_APB3
  - LL_APB4
  - LL_APB5

Return values

- **None:**

Reference Manual to LL API cross reference:

- BUSENSR ACLKNENS LL_BUS_EnableClock
- BUSENSR ACLKNCENS LL_BUS_EnableClock
- BUSENSR AHBMENS LL_BUS_EnableClock
- BUSENSR AHB1ENS LL_BUS_EnableClock
- BUSENSR AHB2ENS LL_BUS_EnableClock
- BUSENSR AHB3ENS LL_BUS_EnableClock
- BUSENSR AHB4ENS LL_BUS_EnableClock
- BUSENSR AHB5ENS LL_BUS_EnableClock
- BUSENSR APB1ENS LL_BUS_EnableClock
- BUSENSR APB2ENS LL_BUS_EnableClock
- BUSENSR APB3ENS LL_BUS_EnableClock
- BUSENSR APB4ENS LL_BUS_EnableClock
- BUSENSR APB5ENS LL_BUS_EnableClock

LL_BUS_IsEnabledClock

Function name

__STATIC_INLINE uint32_t LL_BUS_IsEnabledClock (uint32_t Bus)

Function description

Check if AXI bus clock is enabled or not.

Parameters

- **Bus:** This parameter can be a combination of the following values:
  - LL_ACLKN
  - LL_ACLKNC
  - LL_AHBM
  - LL_AHB1
  - LL_AHB2
  - LL_AHB3
  - LL_AHB4
  - LL_AHB5
  - LL_APB1
  - LL_APB2
  - LL_APB3
  - LL_APB4
  - LL_APB5

Return values

- **uint32_t:**

Reference Manual to LL API cross reference:

- BUSENR ACLKNEN LL_BUS_IsEnabledClock
- BUSENR ACLKNCEN LL_BUS_IsEnabledClock
- BUSENR AHBMEN LL_BUS_IsEnabledClock
- BUSENR AHB1EN LL_BUS_IsEnabledClock
- BUSENR AHB2EN LL_BUS_IsEnabledClock
- BUSENR AHB3EN LL_BUS_IsEnabledClock
- BUSENR AHB4EN LL_BUS_IsEnabledClock
- BUSENR AHB5EN LL_BUS_IsEnabledClock
- BUSENR APB1EN LL_BUS_IsEnabledClock
- BUSENR APB2EN LL_BUS_IsEnabledClock
- BUSENR APB3EN LL_BUS_IsEnabledClock
- BUSENR APB4EN LL_BUS_IsEnabledClock
- BUSENR APB5EN LL_BUS_IsEnabledClock

LL_BUS_DisableClock

Function name

__STATIC_INLINE void LL_BUS_DisableClock (uint32_t Bus)

Function description

Disable AXI bus clock.

Parameters

- **Bus:** This parameter can be a combination of the following values:
  - LL_ACLKN
  - LL_ACLKNC
  - LL_AHBM
  - LL_AHB1
  - LL_AHB2
  - LL_AHB3
  - LL_AHB4
  - LL_AHB5
  - LL_APB1
  - LL_APB2
  - LL_APB3
  - LL_APB4
  - LL_APB5

Return values

- **None:**

Reference Manual to LL API cross reference:

- BUSENCR ACLKNENC LL_BUS_DisableClock
- BUSENCR ACLKNCENC LL_BUS_DisableClock
- BUSENCR AHBMENC LL_BUS_DisableClock
- BUSENCR AHB1ENC LL_BUS_DisableClock
- BUSENCR AHB2ENC LL_BUS_DisableClock
- BUSENCR AHB3ENC LL_BUS_DisableClock
- BUSENCR AHB4ENC LL_BUS_DisableClock
- BUSENCR AHB5ENC LL_BUS_DisableClock
- BUSENCR APB1ENC LL_BUS_DisableClock
- BUSENCR APB2ENC LL_BUS_DisableClock
- BUSENCR APB3ENC LL_BUS_DisableClock
- BUSENCR APB4ENC LL_BUS_DisableClock
- BUSENCR APB5ENC LL_BUS_DisableClock

LL_BUS_EnableClockLowPower

Function name

__STATIC_INLINE void LL_BUS_EnableClockLowPower (uint32_t Bus)

Function description

Enable AXI bus clock during Low Power mode.

Parameters

- **Bus:** This parameter can be a combination of the following values:
  - LL_ACLKN
  - LL_ACLKNC
  - LL_AHBM
  - LL_AHB1
  - LL_AHB2
  - LL_AHB3
  - LL_AHB4
  - LL_AHB5
  - LL_APB1
  - LL_APB2
  - LL_APB3
  - LL_APB4
  - LL_APB5

Return values

- **None:**

Reference Manual to LL API cross reference:

- BUSLPENSR ACLKNLPENS LL_BUS_EnableClockLowPower
- BUSLPENSR ACLKNCLPENS LL_BUS_EnableClockLowPower
- BUSLPENSR AHBMLPENS LL_BUS_EnableClockLowPower
- BUSLPENSR AHB1LPENS LL_BUS_EnableClockLowPower
- BUSLPENSR AHB2LPENS LL_BUS_EnableClockLowPower
- BUSLPENSR AHB3LPENS LL_BUS_EnableClockLowPower
- BUSLPENSR AHB4LPENS LL_BUS_EnableClockLowPower
- BUSLPENSR AHB5LPENS LL_BUS_EnableClockLowPower
- BUSLPENSR APB1LPENS LL_BUS_EnableClockLowPower
- BUSLPENSR APB2LPENS LL_BUS_EnableClockLowPower
- BUSLPENSR APB3LPENS LL_BUS_EnableClockLowPower
- BUSLPENSR APB4LPENS LL_BUS_EnableClockLowPower
- BUSLPENSR APB5LPENS LL_BUS_EnableClockLowPower

LL_BUS_IsEnabledClockLowPower

Function name

__STATIC_INLINE uint32_t LL_BUS_IsEnabledClockLowPower (uint32_t Bus)

Function description

Check if AXI bus clock during Low Power mode is enabled or not.

Parameters

- **Bus:** This parameter can be a combination of the following values:
  - LL_ACLKN
  - LL_ACLKNC
  - LL_AHBM
  - LL_AHB1
  - LL_AHB2
  - LL_AHB3
  - LL_AHB4
  - LL_AHB5
  - LL_APB1
  - LL_APB2
  - LL_APB3
  - LL_APB4
  - LL_APB5

Return values

- **uint32_t:**

Reference Manual to LL API cross reference:

- BUSLPENR ACLKNLPEN LL_BUS_IsEnabledClockLowPower
- BUSLPENR ACLKNCLPEN LL_BUS_IsEnabledClockLowPower
- BUSLPENR AHBMLPEN LL_BUS_IsEnabledClockLowPower
- BUSLPENR AHB1LPEN LL_BUS_IsEnabledClockLowPower
- BUSLPENR AHB2LPEN LL_BUS_IsEnabledClockLowPower
- BUSLPENR AHB3LPEN LL_BUS_IsEnabledClockLowPower
- BUSLPENR AHB4LPEN LL_BUS_IsEnabledClockLowPower
- BUSLPENR AHB5LPEN LL_BUS_IsEnabledClockLowPower
- BUSLPENR APB1LPEN LL_BUS_IsEnabledClockLowPower
- BUSLPENR APB2LPEN LL_BUS_IsEnabledClockLowPower
- BUSLPENR APB3LPEN LL_BUS_IsEnabledClockLowPower
- BUSLPENR APB4LPEN LL_BUS_IsEnabledClockLowPower
- BUSLPENR APB5LPEN LL_BUS_IsEnabledClockLowPower

LL_BUS_DisableClockLowPower

Function name

__STATIC_INLINE void LL_BUS_DisableClockLowPower (uint32_t Bus)

Function description

Disable AXI bus clock during Low Power mode.

Parameters

- **Bus:** This parameter can be a combination of the following values:
  - LL_ACLKN
  - LL_ACLKNC
  - LL_AHBM
  - LL_AHB1
  - LL_AHB2
  - LL_AHB3
  - LL_AHB4
  - LL_AHB5
  - LL_APB1
  - LL_APB2
  - LL_APB3
  - LL_APB4
  - LL_APB5

Return values

- **None:**

Reference Manual to LL API cross reference:

- BUSLPENCR ACLKNLPENC LL_BUS_DisableClockLowPower
- BUSLPENCR ACLKNCLPENC LL_BUS_DisableClockLowPower
- BUSLPENCR AHBMLPENC LL_BUS_DisableClockLowPower
- BUSLPENCR AHB1LPENC LL_BUS_DisableClockLowPower
- BUSLPENCR AHB2LPENC LL_BUS_DisableClockLowPower
- BUSLPENCR AHB3LPENC LL_BUS_DisableClockLowPower
- BUSLPENCR AHB4LPENC LL_BUS_DisableClockLowPower
- BUSLPENCR AHB5LPENC LL_BUS_DisableClockLowPower
- BUSLPENCR APB1LPENC LL_BUS_DisableClockLowPower
- BUSLPENCR APB2LPENC LL_BUS_DisableClockLowPower
- BUSLPENCR APB3LPENC LL_BUS_DisableClockLowPower
- BUSLPENCR APB4LPENC LL_BUS_DisableClockLowPower
- BUSLPENCR APB5LPENC LL_BUS_DisableClockLowPower

LL_MEM_EnableClock

Function name

__STATIC_INLINE void LL_MEM_EnableClock (uint32_t Memories)

Function description

Enable memories clock.

Parameters

- **Memories:** This parameter can be a combination of the following values:
  - LL_MEM_AXISRAM1
  - LL_MEM_AXISRAM2
  - LL_MEM_AXISRAM3
  - LL_MEM_AXISRAM4
  - LL_MEM_AXISRAM5
  - LL_MEM_AXISRAM6
  - LL_MEM_AHBSRAM1
  - LL_MEM_AHBSRAM2
  - LL_MEM_BKPSRAM
  - LL_MEM_FLEXRAM
  - LL_MEM_CACHEAXIRAM
  - LL_MEM_VENCRAM

Return values

- **None:**

Reference Manual to LL API cross reference:

- MEMENSR AXISRAM1ENS LL_MEM_EnableClock
- MEMENSR AXISRAM2ENS LL_MEM_EnableClock
- MEMENSR AXISRAM3ENS LL_MEM_EnableClock
- MEMENSR AXISRAM4ENS LL_MEM_EnableClock
- MEMENSR AXISRAM5ENS LL_MEM_EnableClock
- MEMENSR AXISRAM6ENS LL_MEM_EnableClock
- MEMENSR AHBSRAM1ENS LL_MEM_EnableClock
- MEMENSR AHBSRAM2ENS LL_MEM_EnableClock
- MEMENSR BKPSRAMENS LL_MEM_EnableClock
- MEMENSR FLEXRAMENS LL_MEM_EnableClock
- MEMENSR CACHEAXIRAMENS LL_MEM_EnableClock
- MEMENSR VENCRAMENS LL_MEM_EnableClock

LL_MEM_IsEnabledClock

Function name

__STATIC_INLINE uint32_t LL_MEM_IsEnabledClock (uint32_t Memories)

Function description

Check if memory clock is enabled or not.

Parameters

- **Memories:** This parameter can be a combination of the following values:
  - LL_MEM_AXISRAM1
  - LL_MEM_AXISRAM2
  - LL_MEM_AXISRAM3
  - LL_MEM_AXISRAM4
  - LL_MEM_AXISRAM5
  - LL_MEM_AXISRAM6
  - LL_MEM_AHBSRAM1
  - LL_MEM_AHBSRAM2
  - LL_MEM_BKPSRAM
  - LL_MEM_FLEXRAM
  - LL_MEM_CACHEAXIRAM
  - LL_MEM_VENCRAM

Return values

- **uint32_t:**

Reference Manual to LL API cross reference:

- MEMENR AXISRAM1EN LL_MEM_IsEnabledClock
- MEMENR AXISRAM2EN LL_MEM_IsEnabledClock
- MEMENR AXISRAM3EN LL_MEM_IsEnabledClock
- MEMENR AXISRAM4EN LL_MEM_IsEnabledClock
- MEMENR AXISRAM5EN LL_MEM_IsEnabledClock
- MEMENR AXISRAM6EN LL_MEM_IsEnabledClock
- MEMENR AHBSRAM1EN LL_MEM_IsEnabledClock
- MEMENR AHBSRAM2EN LL_MEM_IsEnabledClock
- MEMENR BKPSRAMEN LL_MEM_IsEnabledClock
- MEMENR FLEXRAMEN LL_MEM_IsEnabledClock
- MEMENR CACHEAXIRAMEN LL_MEM_IsEnabledClock
- MEMENR VENCRAMEN LL_MEM_IsEnabledClock

LL_MEM_DisableClock

Function name

__STATIC_INLINE void LL_MEM_DisableClock (uint32_t Memories)

Function description

Disable memories clock.

Parameters

- **Memories:** This parameter can be a combination of the following values:
  - LL_MEM_AXISRAM1
  - LL_MEM_AXISRAM2
  - LL_MEM_AXISRAM3
  - LL_MEM_AXISRAM4
  - LL_MEM_AXISRAM5
  - LL_MEM_AXISRAM6
  - LL_MEM_AHBSRAM1
  - LL_MEM_AHBSRAM2
  - LL_MEM_BKPSRAM
  - LL_MEM_FLEXRAM
  - LL_MEM_CACHEAXIRAM
  - LL_MEM_VENCRAM

Return values

- **None:**

Reference Manual to LL API cross reference:

- MEMENCR AXISRAM1ENC LL_MEM_DisableClock
- MEMENCR AXISRAM2ENC LL_MEM_DisableClock
- MEMENCR AXISRAM3ENC LL_MEM_DisableClock
- MEMENCR AXISRAM4ENC LL_MEM_DisableClock
- MEMENCR AXISRAM5ENC LL_MEM_DisableClock
- MEMENCR AXISRAM6ENC LL_MEM_DisableClock
- MEMENCR AHBSRAM1ENC LL_MEM_DisableClock
- MEMENCR AHBSRAM2ENC LL_MEM_DisableClock
- MEMENCR BKPSRAMENC LL_MEM_DisableClock
- MEMENCR FLEXRAMENC LL_MEM_DisableClock
- MEMENCR CACHEAXIRAMENC LL_MEM_DisableClock
- MEMENCR VENCRAMENC LL_MEM_DisableClock

LL_MEM_EnableClockLowPower

Function name

__STATIC_INLINE void LL_MEM_EnableClockLowPower (uint32_t Memories)

Function description

Enable memories clock during Low Power mode.

Parameters

- **Memories:** This parameter can be a combination of the following values:
  - LL_MEM_AXISRAM1
  - LL_MEM_AXISRAM2
  - LL_MEM_AXISRAM3
  - LL_MEM_AXISRAM4
  - LL_MEM_AXISRAM5
  - LL_MEM_AXISRAM6
  - LL_MEM_AHBSRAM1
  - LL_MEM_AHBSRAM2
  - LL_MEM_BKPSRAM
  - LL_MEM_FLEXRAM
  - LL_MEM_CACHEAXIRAM
  - LL_MEM_VENCRAM

Return values

- **None:**

Reference Manual to LL API cross reference:

- MEMLPENSR AXISRAM1LPENS LL_MEM_EnableClockLowPower
- MEMLPENSR AXISRAM2LPENS LL_MEM_EnableClockLowPower
- MEMLPENSR AXISRAM3LPENS LL_MEM_EnableClockLowPower
- MEMLPENSR AXISRAM4LPENS LL_MEM_EnableClockLowPower
- MEMLPENSR AXISRAM5LPENS LL_MEM_EnableClockLowPower
- MEMLPENSR AXISRAM6LPENS LL_MEM_EnableClockLowPower
- MEMLPENSR AHBSRAM1LPENS LL_MEM_EnableClockLowPower
- MEMLPENSR AHBSRAM2LPENS LL_MEM_EnableClockLowPower
- MEMLPENSR BKPSRAMLPENS LL_MEM_EnableClockLowPower
- MEMLPENSR FLEXRAMLPENS LL_MEM_EnableClockLowPower
- MEMLPENSR CACHEAXIRAMLPENS LL_MEM_EnableClockLowPower
- MEMLPENSR VENCRAMLPENS LL_MEM_EnableClockLowPower

LL_MEM_IsEnabledClockLowPower

Function name

__STATIC_INLINE uint32_t LL_MEM_IsEnabledClockLowPower (uint32_t Memories)

Function description

Check if memories clock during Low Power mode is enabled or not .

Parameters

- **Memories:** This parameter can be a combination of the following values:
  - LL_MEM_AXISRAM1
  - LL_MEM_AXISRAM2
  - LL_MEM_AXISRAM3
  - LL_MEM_AXISRAM4
  - LL_MEM_AXISRAM5
  - LL_MEM_AXISRAM6
  - LL_MEM_AHBSRAM1
  - LL_MEM_AHBSRAM2
  - LL_MEM_BKPSRAM
  - LL_MEM_FLEXRAM
  - LL_MEM_CACHEAXIRAM
  - LL_MEM_VENCRAM

Return values

- **None:**

Reference Manual to LL API cross reference:

- MEMLPENR AXISRAM1LPEN LL_MEM_IsEnabledClockLowPower
- MEMLPENR AXISRAM2LPEN LL_MEM_IsEnabledClockLowPower
- MEMLPENR AXISRAM3LPEN LL_MEM_IsEnabledClockLowPower
- MEMLPENR AXISRAM4LPEN LL_MEM_IsEnabledClockLowPower
- MEMLPENR AXISRAM5LPEN LL_MEM_IsEnabledClockLowPower
- MEMLPENR AXISRAM6LPEN LL_MEM_IsEnabledClockLowPower
- MEMLPENR AHBSRAM1LPEN LL_MEM_IsEnabledClockLowPower
- MEMLPENR AHBSRAM2LPEN LL_MEM_IsEnabledClockLowPower
- MEMLPENR BKPSRAMLPEN LL_MEM_IsEnabledClockLowPower
- MEMLPENR FLEXRAMLPEN LL_MEM_IsEnabledClockLowPower
- MEMLPENR CACHEAXIRAMLPEN LL_MEM_IsEnabledClockLowPower
- MEMLPENR VENCRAMLPEN LL_MEM_IsEnabledClockLowPower

LL_MEM_DisableClockLowPower

Function name

__STATIC_INLINE void LL_MEM_DisableClockLowPower (uint32_t Memories)

Function description

Disable memories clock during Low Power mode.

Parameters

- **Memories:** This parameter can be a combination of the following values:
  - LL_MEM_AXISRAM1
  - LL_MEM_AXISRAM2
  - LL_MEM_AXISRAM3
  - LL_MEM_AXISRAM4
  - LL_MEM_AXISRAM5
  - LL_MEM_AXISRAM6
  - LL_MEM_AHBSRAM1
  - LL_MEM_AHBSRAM2
  - LL_MEM_BKPSRAM
  - LL_MEM_FLEXRAM
  - LL_MEM_CACHEAXIRAM
  - LL_MEM_VENCRAM

Return values

- **None:**

Reference Manual to LL API cross reference:

- MEMLPENCR AXISRAM1LPENC LL_MEM_DisableClockLowPower
- MEMLPENCR AXISRAM2LPENC LL_MEM_DisableClockLowPower
- MEMLPENCR AXISRAM3LPENC LL_MEM_DisableClockLowPower
- MEMLPENCR AXISRAM4LPENC LL_MEM_DisableClockLowPower
- MEMLPENCR AXISRAM5LPENC LL_MEM_DisableClockLowPower
- MEMLPENCR AXISRAM6LPENC LL_MEM_DisableClockLowPower
- MEMLPENCR AHBSRAM1LPENC LL_MEM_DisableClockLowPower
- MEMLPENCR AHBSRAM2LPENC LL_MEM_DisableClockLowPower
- MEMLPENCR BKPSRAMLPENC LL_MEM_DisableClockLowPower
- MEMLPENCR FLEXRAMLPENC LL_MEM_DisableClockLowPower
- MEMLPENCR CACHEAXIRAMLPENC LL_MEM_DisableClockLowPower
- MEMLPENCR VENCRAMLPENC LL_MEM_DisableClockLowPower

LL_AHB1_GRP1_EnableClock

Function name

__STATIC_INLINE void LL_AHB1_GRP1_EnableClock (uint32_t Periphs)

Function description

Enable AHB1 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB1_GRP1_PERIPH_GPDMA1
  - LL_AHB1_GRP1_PERIPH_ADC12

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB1ENSR GPDMA1ENS LL_AHB1_GRP1_EnableClock
- AHB1ENSR ADC12ENS LL_AHB1_GRP1_EnableClock

LL_AHB1_GRP1_IsEnabledClock

Function name

__STATIC_INLINE uint32_t LL_AHB1_GRP1_IsEnabledClock (uint32_t Periphs)

Function description

Check if AHB1 peripheral clock is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB1_GRP1_PERIPH_GPDMA1
  - LL_AHB1_GRP1_PERIPH_ADC12

Return values

- **uint32_t:**

Reference Manual to LL API cross reference:

- AHB1ENR GPDMA1EN LL_AHB1_GRP1_IsEnabledClock
- AHB1ENR ADC12EN LL_AHB1_GRP1_IsEnabledClock

LL_AHB1_GRP1_DisableClock

Function name

__STATIC_INLINE void LL_AHB1_GRP1_DisableClock (uint32_t Periphs)

Function description

Disable AHB1 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB1_GRP1_PERIPH_GPDMA1
  - LL_AHB1_GRP1_PERIPH_ADC12

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB1ENCR GPDMA1ENC LL_AHB1_GRP1_DisableClock
- AHB1ENCR ADC12ENC LL_AHB1_GRP1_DisableClock

LL_AHB1_GRP1_ForceReset

Function name

__STATIC_INLINE void LL_AHB1_GRP1_ForceReset (uint32_t Periphs)

Function description

Force AHB1 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB1_GRP1_PERIPH_GPDMA1
  - LL_AHB1_GRP1_PERIPH_ADC12

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB1RSTSR GPDMA1RSTS LL_AHB1_GRP1_ForceReset
- AHB1RSTSR ADC12RSTS LL_AHB1_GRP1_ForceReset

LL_AHB1_GRP1_ReleaseReset

Function name

__STATIC_INLINE void LL_AHB1_GRP1_ReleaseReset (uint32_t Periphs)

Function description

Release AHB1 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB1_GRP1_PERIPH_GPDMA1
  - LL_AHB1_GRP1_PERIPH_ADC12

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB1RSTCR GPDMA1RSTC LL_AHB1_GRP1_ReleaseReset
- AHB1RSTCR ADC12RSTC LL_AHB1_GRP1_ReleaseReset

LL_AHB1_GRP1_EnableClockLowPower

Function name

__STATIC_INLINE void LL_AHB1_GRP1_EnableClockLowPower (uint32_t Periphs)

Function description

Enable AHB1 peripherals clock during Low Power mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB1_GRP1_PERIPH_GPDMA1
  - LL_AHB1_GRP1_PERIPH_ADC12

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB1LPENSR GPDMA1LPENS LL_AHB1_GRP1_EnableClockLowPower
- AHB1LPENSR ADC12LPENS LL_AHB1_GRP1_EnableClockLowPower

LL_AHB1_GRP1_IsEnabledClockLowPower

Function name

__STATIC_INLINE uint32_t LL_AHB1_GRP1_IsEnabledClockLowPower (uint32_t Periphs)

Function description

Check if AHB1 peripheral clock during Low Power mode is enabled or not .

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB1_GRP1_PERIPH_GPDMA1
  - LL_AHB1_GRP1_PERIPH_ADC12

Return values

- **uint32_t:**

Reference Manual to LL API cross reference:

- AHB1LPENR GPDMA1LPEN LL_AHB1_GRP1_IsEnabledClockLowPower
- AHB1LPENR ADC12LPEN LL_AHB1_GRP1_IsEnabledClockLowPower

LL_AHB1_GRP1_DisableClockLowPower

Function name

__STATIC_INLINE void LL_AHB1_GRP1_DisableClockLowPower (uint32_t Periphs)

Function description

Disable AHB1 peripherals clock during Low Power mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB1_GRP1_PERIPH_GPDMA1
  - LL_AHB1_GRP1_PERIPH_ADC12

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB1LPENCR GPDMA1LPENC LL_AHB1_GRP1_DisableClockLowPower
- AHB1LPENCR ADC12LPENC LL_AHB1_GRP1_DisableClockLowPower

LL_AHB2_GRP1_EnableClock

Function name

__STATIC_INLINE void LL_AHB2_GRP1_EnableClock (uint32_t Periphs)

Function description

Enable AHB2 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB2_GRP1_PERIPH_RAMCFG
  - LL_AHB2_GRP1_PERIPH_MDF1
  - LL_AHB2_GRP1_PERIPH_ADF1

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB2ENSR RAMCFGENS LL_AHB2_GRP1_EnableClock
- AHB2ENSR MDF1ENS LL_AHB2_GRP1_EnableClock
- AHB2ENSR ADF1ENS LL_AHB2_GRP1_EnableClock

LL_AHB2_GRP1_IsEnabledClock

Function name

__STATIC_INLINE uint32_t LL_AHB2_GRP1_IsEnabledClock (uint32_t Periphs)

Function description

Check if AHB2 peripheral clock is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB2_GRP1_PERIPH_RAMCFG
  - LL_AHB2_GRP1_PERIPH_MDF1
  - LL_AHB2_GRP1_PERIPH_ADF1

Return values

- **uint32_t:**

Reference Manual to LL API cross reference:

- AHB2ENR RAMCFGEN LL_AHB2_GRP1_IsEnabledClock
- AHB2ENR MDF1EN LL_AHB2_GRP1_IsEnabledClock
- AHB2ENR ADF1EN LL_AHB2_GRP1_IsEnabledClock

LL_AHB2_GRP1_DisableClock

Function name

__STATIC_INLINE void LL_AHB2_GRP1_DisableClock (uint32_t Periphs)

Function description

Disable AHB2 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB2_GRP1_PERIPH_RAMCFG
  - LL_AHB2_GRP1_PERIPH_MDF1
  - LL_AHB2_GRP1_PERIPH_ADF1

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB2ENCR RAMCFGENC LL_AHB2_GRP1_DisableClock
- AHB2ENCR MDF1ENC LL_AHB2_GRP1_DisableClock
- AHB2ENCR ADF1ENC LL_AHB2_GRP1_DisableClock

LL_AHB2_GRP1_ForceReset

Function name

__STATIC_INLINE void LL_AHB2_GRP1_ForceReset (uint32_t Periphs)

Function description

Force AHB2 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB2_GRP1_PERIPH_RAMCFG
  - LL_AHB2_GRP1_PERIPH_MDF1
  - LL_AHB2_GRP1_PERIPH_ADF1

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB2RSTSR RAMCFGRSTS LL_AHB2_GRP1_ForceReset
- AHB2RSTSR MDF1RSTS LL_AHB2_GRP1_ForceReset
- AHB2RSTSR ADF1RSTS LL_AHB2_GRP1_ForceReset

LL_AHB2_GRP1_ReleaseReset

Function name

__STATIC_INLINE void LL_AHB2_GRP1_ReleaseReset (uint32_t Periphs)

Function description

Release AHB2 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB2_GRP1_PERIPH_RAMCFG
  - LL_AHB2_GRP1_PERIPH_MDF1
  - LL_AHB2_GRP1_PERIPH_ADF1

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB2RSTCR RAMCFGRSTC LL_AHB2_GRP1_ReleaseReset
- AHB2RSTCR MDF1RSTC LL_AHB2_GRP1_ReleaseReset
- AHB2RSTCR ADF1RSTC LL_AHB2_GRP1_ReleaseReset

LL_AHB2_GRP1_EnableClockLowPower

Function name

__STATIC_INLINE void LL_AHB2_GRP1_EnableClockLowPower (uint32_t Periphs)

Function description

Enable AHB2 peripherals clock during Low Power mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB2_GRP1_PERIPH_RAMCFG
  - LL_AHB2_GRP1_PERIPH_MDF1
  - LL_AHB2_GRP1_PERIPH_ADF1

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB2LPENSR RAMCFGLPENC LL_AHB2_GRP1_EnableClockLowPower
- AHB2LPENSR MDF1LPENC LL_AHB2_GRP1_EnableClockLowPower
- AHB2LPENSR ADF1LPENC LL_AHB2_GRP1_EnableClockLowPower

LL_AHB2_GRP1_IsEnabledClockLowPower

Function name

__STATIC_INLINE uint32_t LL_AHB2_GRP1_IsEnabledClockLowPower (uint32_t Periphs)

Function description

Check if AHB2 peripheral clock during Low Power mode is enabled or not .

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB2_GRP1_PERIPH_RAMCFG
  - LL_AHB2_GRP1_PERIPH_MDF1
  - LL_AHB2_GRP1_PERIPH_ADF1

Return values

- **uint32_t:**

Reference Manual to LL API cross reference:

- AHB2LPENR RAMCFGLPEN LL_AHB2_GRP1_IsEnabledClockLowPower
- AHB2LPENR MDF1LPEN LL_AHB2_GRP1_IsEnabledClockLowPower
- AHB2LPENR ADF1LPEN LL_AHB2_GRP1_IsEnabledClockLowPower

LL_AHB2_GRP1_DisableClockLowPower

Function name

__STATIC_INLINE void LL_AHB2_GRP1_DisableClockLowPower (uint32_t Periphs)

Function description

Disable AHB2 peripherals clock during Low Power mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB2_GRP1_PERIPH_RAMCFG
  - LL_AHB2_GRP1_PERIPH_MDF1
  - LL_AHB2_GRP1_PERIPH_ADF1

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB2LPENCR RAMCFGLPENC LL_AHB2_GRP1_DisableClockLowPower
- AHB2LPENCR MDF1LPENC LL_AHB2_GRP1_DisableClockLowPower
- AHB2LPENCR ADF1LPENC LL_AHB2_GRP1_DisableClockLowPower

LL_AHB3_GRP1_EnableClock

Function name

__STATIC_INLINE void LL_AHB3_GRP1_EnableClock (uint32_t Periphs)

Function description

Enable AHB3 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB3_GRP1_PERIPH_RNG
  - LL_AHB3_GRP1_PERIPH_HASH
  - LL_AHB3_GRP1_PERIPH_CRYP (*)
  - LL_AHB3_GRP1_PERIPH_SAES (*)
  - LL_AHB3_GRP1_PERIPH_PKA
  - LL_AHB3_GRP1_PERIPH_RIFSC
  - LL_AHB3_GRP1_PERIPH_RISAF
  - LL_AHB3_GRP1_PERIPH_IAC

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB3ENSR RNGENS LL_AHB3_GRP1_EnableClock
- AHB3ENSR HASHENS LL_AHB3_GRP1_EnableClock
- AHB3ENSR CRYPENS LL_AHB3_GRP1_EnableClock
- (*) AHB3ENSR SAESENS LL_AHB3_GRP1_EnableClock
- (*) AHB3ENSR PKAENS LL_AHB3_GRP1_EnableClock
- AHB3ENSR RIFSCENS LL_AHB3_GRP1_EnableClock
- AHB3ENSR RISAFENS LL_AHB3_GRP1_EnableClock
- AHB3ENSR IACENS LL_AHB3_GRP1_EnableClock

LL_AHB3_GRP1_IsEnabledClock

Function name

__STATIC_INLINE uint32_t LL_AHB3_GRP1_IsEnabledClock (uint32_t Periphs)

Function description

Check if AHB3 peripheral clock is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB3_GRP1_PERIPH_RNG
  - LL_AHB3_GRP1_PERIPH_HASH
  - LL_AHB3_GRP1_PERIPH_CRYP (*)
  - LL_AHB3_GRP1_PERIPH_SAES (*)
  - LL_AHB3_GRP1_PERIPH_PKA
  - LL_AHB3_GRP1_PERIPH_RIFSC
  - LL_AHB3_GRP1_PERIPH_RISAF
  - LL_AHB3_GRP1_PERIPH_IAC

  (*) value not defined in all devices.

Return values

- **uint32_t:**

Reference Manual to LL API cross reference:

- AHB3ENR RNGEN LL_AHB3_GRP1_IsEnabledClock
- AHB3ENR HASHEN LL_AHB3_GRP1_IsEnabledClock
- AHB3ENR CRYPEN LL_AHB3_GRP1_IsEnabledClock
- (*) AHB3ENR SAESEN LL_AHB3_GRP1_IsEnabledClock
- (*) AHB3ENR PKAEN LL_AHB3_GRP1_IsEnabledClock
- AHB3ENR RIFSCEN LL_AHB3_GRP1_IsEnabledClock
- AHB3ENR RISAFEN LL_AHB3_GRP1_IsEnabledClock
- AHB3ENR IACEN LL_AHB3_GRP1_IsEnabledClock

LL_AHB3_GRP1_DisableClock

Function name

__STATIC_INLINE void LL_AHB3_GRP1_DisableClock (uint32_t Periphs)

Function description

Disable AHB3 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB3_GRP1_PERIPH_RNG
  - LL_AHB3_GRP1_PERIPH_HASH
  - LL_AHB3_GRP1_PERIPH_CRYP (*)
  - LL_AHB3_GRP1_PERIPH_SAES (*)
  - LL_AHB3_GRP1_PERIPH_PKA
  - LL_AHB3_GRP1_PERIPH_RIFSC
  - LL_AHB3_GRP1_PERIPH_RISAF
  - LL_AHB3_GRP1_PERIPH_IAC

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB3ENCR RNGENC LL_AHB3_GRP1_DisableClock
- AHB3ENCR HASHENC LL_AHB3_GRP1_DisableClock
- AHB3ENCR CRYPENC LL_AHB3_GRP1_DisableClock
- (*) AHB3ENCR SAESENC LL_AHB3_GRP1_DisableClock
- (*) AHB3ENCR PKAENC LL_AHB3_GRP1_DisableClock
- AHB3ENCR RIFSCENC LL_AHB3_GRP1_DisableClock
- AHB3ENCR RISAFENC LL_AHB3_GRP1_DisableClock
- AHB3ENCR IACENC LL_AHB3_GRP1_DisableClock

LL_AHB3_GRP1_ForceReset

Function name

__STATIC_INLINE void LL_AHB3_GRP1_ForceReset (uint32_t Periphs)

Function description

Force AHB3 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB3_GRP1_PERIPH_RNG
  - LL_AHB3_GRP1_PERIPH_HASH
  - LL_AHB3_GRP1_PERIPH_CRYP (*)
  - LL_AHB3_GRP1_PERIPH_SAES (*)
  - LL_AHB3_GRP1_PERIPH_PKA
  - LL_AHB3_GRP1_PERIPH_IAC

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB3RSTSR RNGRSTS LL_AHB3_GRP1_ForceReset
- AHB3RSTSR HASHRSTS LL_AHB3_GRP1_ForceReset
- AHB3RSTSR CRYPRSTS LL_AHB3_GRP1_ForceReset
- (*) AHB3RSTSR SAESRSTS LL_AHB3_GRP1_ForceReset
- (*) AHB3RSTSR PKARSTS LL_AHB3_GRP1_ForceReset
- AHB3RSTSR IACRSTS LL_AHB3_GRP1_ForceReset

LL_AHB3_GRP1_ReleaseReset

Function name

__STATIC_INLINE void LL_AHB3_GRP1_ReleaseReset (uint32_t Periphs)

Function description

Release AHB3 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB3_GRP1_PERIPH_RNG
  - LL_AHB3_GRP1_PERIPH_HASH
  - LL_AHB3_GRP1_PERIPH_CRYP (*)
  - LL_AHB3_GRP1_PERIPH_SAES (*)
  - LL_AHB3_GRP1_PERIPH_PKA
  - LL_AHB3_GRP1_PERIPH_IAC

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB3RSTCR RNGRSTC LL_AHB3_GRP1_ReleaseReset
- AHB3RSTCR HASHRSTC LL_AHB3_GRP1_ReleaseReset
- AHB3RSTCR CRYPRSTC LL_AHB3_GRP1_ReleaseReset
- (*) AHB3RSTCR SAESRSTC LL_AHB3_GRP1_ReleaseReset
- (*) AHB3RSTCR PKARSTC LL_AHB3_GRP1_ReleaseReset
- AHB3RSTCR IACRSTC LL_AHB3_GRP1_ReleaseReset

LL_AHB3_GRP1_EnableClockLowPower

Function name

__STATIC_INLINE void LL_AHB3_GRP1_EnableClockLowPower (uint32_t Periphs)

Function description

Enable AHB3 peripherals clock during Low Power mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB3_GRP1_PERIPH_RNG
  - LL_AHB3_GRP1_PERIPH_HASH
  - LL_AHB3_GRP1_PERIPH_CRYP (*)
  - LL_AHB3_GRP1_PERIPH_SAES (*)
  - LL_AHB3_GRP1_PERIPH_PKA
  - LL_AHB3_GRP1_PERIPH_RIFSC
  - LL_AHB3_GRP1_PERIPH_RISAF
  - LL_AHB3_GRP1_PERIPH_IAC

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB3LPENSR RNGLPENS LL_AHB3_GRP1_EnableClockLowPower
- AHB3LPENSR HASHLPENS LL_AHB3_GRP1_EnableClockLowPower
- AHB3LPENSR CRYPLPENS LL_AHB3_GRP1_EnableClockLowPower
- (*) AHB3LPENSR SAESLPENS LL_AHB3_GRP1_EnableClockLowPower
- (*) AHB3LPENSR PKALPENS LL_AHB3_GRP1_EnableClockLowPower
- AHB3LPENSR RIFSCLPENS LL_AHB3_GRP1_EnableClockLowPower
- AHB3LPENSR RISAFLPENS LL_AHB3_GRP1_EnableClockLowPower
- AHB3LPENSR IACLPENS LL_AHB3_GRP1_EnableClockLowPower

LL_AHB3_GRP1_IsEnabledClockLowPower

Function name

__STATIC_INLINE uint32_t LL_AHB3_GRP1_IsEnabledClockLowPower (uint32_t Periphs)

Function description

Check if AHB3 peripheral clock during Low Power mode is enabled or not .

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB3_GRP1_PERIPH_RNG
  - LL_AHB3_GRP1_PERIPH_HASH
  - LL_AHB3_GRP1_PERIPH_CRYP (*)
  - LL_AHB3_GRP1_PERIPH_SAES (*)
  - LL_AHB3_GRP1_PERIPH_PKA
  - LL_AHB3_GRP1_PERIPH_RIFSC
  - LL_AHB3_GRP1_PERIPH_RISAF
  - LL_AHB3_GRP1_PERIPH_IAC

  (*) value not defined in all devices.

Return values

- **uint32_t:**

Reference Manual to LL API cross reference:

- AHB3LPENR RNGLPEN LL_AHB3_GRP1_IsEnabledClockLowPower
- AHB3LPENR HASHLPEN LL_AHB3_GRP1_IsEnabledClockLowPower
- AHB3LPENR CRYPLPEN LL_AHB3_GRP1_IsEnabledClockLowPower
- (*) AHB3LPENR SAESLPEN LL_AHB3_GRP1_IsEnabledClockLowPower
- (*) AHB3LPENR PKALPEN LL_AHB3_GRP1_IsEnabledClockLowPower
- AHB3LPENR RIFSCLPEN LL_AHB3_GRP1_IsEnabledClockLowPower
- AHB3LPENR RISAFLPEN LL_AHB3_GRP1_IsEnabledClockLowPower
- AHB3LPENR IACLPEN LL_AHB3_GRP1_IsEnabledClockLowPower

LL_AHB3_GRP1_DisableClockLowPower

Function name

__STATIC_INLINE void LL_AHB3_GRP1_DisableClockLowPower (uint32_t Periphs)

Function description

Disable AHB3 peripherals clock during Low Power mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB3_GRP1_PERIPH_RNG
  - LL_AHB3_GRP1_PERIPH_HASH
  - LL_AHB3_GRP1_PERIPH_CRYP (*)
  - LL_AHB3_GRP1_PERIPH_SAES (*)
  - LL_AHB3_GRP1_PERIPH_PKA
  - LL_AHB3_GRP1_PERIPH_RIFSC
  - LL_AHB3_GRP1_PERIPH_RISAF
  - LL_AHB3_GRP1_PERIPH_IAC

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB3LPENCR RNGLPENC LL_AHB3_GRP1_DisableClockLowPower
- AHB3LPENCR HASHLPENC LL_AHB3_GRP1_DisableClockLowPower
- AHB3LPENCR CRYPLPENC LL_AHB3_GRP1_DisableClockLowPower
- (*) AHB3LPENCR SAESLPENC LL_AHB3_GRP1_DisableClockLowPower
- (*) AHB3LPENCR PKALPENC LL_AHB3_GRP1_DisableClockLowPower
- AHB3LPENCR RIFSCLPENC LL_AHB3_GRP1_DisableClockLowPower
- AHB3LPENCR RISAFLPENC LL_AHB3_GRP1_DisableClockLowPower
- AHB3LPENCR IACLPENC LL_AHB3_GRP1_DisableClockLowPower

LL_AHB4_GRP1_EnableClock

Function name

__STATIC_INLINE void LL_AHB4_GRP1_EnableClock (uint32_t Periphs)

Function description

Enable AHB4 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB4_GRP1_PERIPH_GPIOA
  - LL_AHB4_GRP1_PERIPH_GPIOB
  - LL_AHB4_GRP1_PERIPH_GPIOC
  - LL_AHB4_GRP1_PERIPH_GPIOD
  - LL_AHB4_GRP1_PERIPH_GPIOE
  - LL_AHB4_GRP1_PERIPH_GPIOF
  - LL_AHB4_GRP1_PERIPH_GPIOG
  - LL_AHB4_GRP1_PERIPH_GPIOH
  - LL_AHB4_GRP1_PERIPH_GPION
  - LL_AHB4_GRP1_PERIPH_GPIOO
  - LL_AHB4_GRP1_PERIPH_GPIOP
  - LL_AHB4_GRP1_PERIPH_GPIOQ
  - LL_AHB4_GRP1_PERIPH_PWR
  - LL_AHB4_GRP1_PERIPH_CRC

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB4ENSR GPIOAENS LL_AHB4_GRP1_EnableClock
- AHB4ENSR GPIOBENS LL_AHB4_GRP1_EnableClock
- AHB4ENSR GPIOCENS LL_AHB4_GRP1_EnableClock
- AHB4ENSR GPIODENS LL_AHB4_GRP1_EnableClock
- AHB4ENSR GPIOEENS LL_AHB4_GRP1_EnableClock
- AHB4ENSR GPIOFENS LL_AHB4_GRP1_EnableClock
- AHB4ENSR GPIOGENS LL_AHB4_GRP1_EnableClock
- AHB4ENSR GPIOHENS LL_AHB4_GRP1_EnableClock
- AHB4ENSR GPIONENS LL_AHB4_GRP1_EnableClock
- AHB4ENSR GPIOOENS LL_AHB4_GRP1_EnableClock
- AHB4ENSR GPIOPENS LL_AHB4_GRP1_EnableClock
- AHB4ENSR GPIOQENS LL_AHB4_GRP1_EnableClock
- AHB4ENSR PWRENS LL_AHB4_GRP1_EnableClock
- AHB4ENSR CRCENS LL_AHB4_GRP1_EnableClock

LL_AHB4_GRP1_IsEnabledClock

Function name

__STATIC_INLINE uint32_t LL_AHB4_GRP1_IsEnabledClock (uint32_t Periphs)

Function description

Check if AHB4 peripheral clock is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB4_GRP1_PERIPH_GPIOA
  - LL_AHB4_GRP1_PERIPH_GPIOB
  - LL_AHB4_GRP1_PERIPH_GPIOC
  - LL_AHB4_GRP1_PERIPH_GPIOD
  - LL_AHB4_GRP1_PERIPH_GPIOE
  - LL_AHB4_GRP1_PERIPH_GPIOF
  - LL_AHB4_GRP1_PERIPH_GPIOG
  - LL_AHB4_GRP1_PERIPH_GPIOH
  - LL_AHB4_GRP1_PERIPH_GPION
  - LL_AHB4_GRP1_PERIPH_GPIOO
  - LL_AHB4_GRP1_PERIPH_GPIOP
  - LL_AHB4_GRP1_PERIPH_GPIOQ
  - LL_AHB4_GRP1_PERIPH_PWR
  - LL_AHB4_GRP1_PERIPH_CRC

Return values

- **uint32_t:**

Reference Manual to LL API cross reference:

- AHB4ENR GPIOAEN LL_AHB4_GRP1_IsEnabledClock
- AHB4ENR GPIOBEN LL_AHB4_GRP1_IsEnabledClock
- AHB4ENR GPIOCEN LL_AHB4_GRP1_IsEnabledClock
- AHB4ENR GPIODEN LL_AHB4_GRP1_IsEnabledClock
- AHB4ENR GPIOEEN LL_AHB4_GRP1_IsEnabledClock
- AHB4ENR GPIOFEN LL_AHB4_GRP1_IsEnabledClock
- AHB4ENR GPIOGEN LL_AHB4_GRP1_IsEnabledClock
- AHB4ENR GPIOHEN LL_AHB4_GRP1_IsEnabledClock
- AHB4ENR GPIONEN LL_AHB4_GRP1_IsEnabledClock
- AHB4ENR GPIOOEN LL_AHB4_GRP1_IsEnabledClock
- AHB4ENR GPIOPEN LL_AHB4_GRP1_IsEnabledClock
- AHB4ENR GPIOQEN LL_AHB4_GRP1_IsEnabledClock
- AHB4ENR PWREN LL_AHB4_GRP1_IsEnabledClock
- AHB4ENR CRCEN LL_AHB4_GRP1_IsEnabledClock

LL_AHB4_GRP1_DisableClock

Function name

__STATIC_INLINE void LL_AHB4_GRP1_DisableClock (uint32_t Periphs)

Function description

Disable AHB4 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB4_GRP1_PERIPH_GPIOA
  - LL_AHB4_GRP1_PERIPH_GPIOB
  - LL_AHB4_GRP1_PERIPH_GPIOC
  - LL_AHB4_GRP1_PERIPH_GPIOD
  - LL_AHB4_GRP1_PERIPH_GPIOE
  - LL_AHB4_GRP1_PERIPH_GPIOF
  - LL_AHB4_GRP1_PERIPH_GPIOG
  - LL_AHB4_GRP1_PERIPH_GPIOH
  - LL_AHB4_GRP1_PERIPH_GPION
  - LL_AHB4_GRP1_PERIPH_GPIOO
  - LL_AHB4_GRP1_PERIPH_GPIOP
  - LL_AHB4_GRP1_PERIPH_GPIOQ
  - LL_AHB4_GRP1_PERIPH_PWR
  - LL_AHB4_GRP1_PERIPH_CRC

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB4ENCR GPIOAENC LL_AHB4_GRP1_DisableClock
- AHB4ENCR GPIOBENC LL_AHB4_GRP1_DisableClock
- AHB4ENCR GPIOCENC LL_AHB4_GRP1_DisableClock
- AHB4ENCR GPIODENC LL_AHB4_GRP1_DisableClock
- AHB4ENCR GPIOEENC LL_AHB4_GRP1_DisableClock
- AHB4ENCR GPIOFENC LL_AHB4_GRP1_DisableClock
- AHB4ENCR GPIOGENC LL_AHB4_GRP1_DisableClock
- AHB4ENCR GPIOHENC LL_AHB4_GRP1_DisableClock
- AHB4ENCR GPIONENC LL_AHB4_GRP1_DisableClock
- AHB4ENCR GPIOOENC LL_AHB4_GRP1_DisableClock
- AHB4ENCR GPIOPENC LL_AHB4_GRP1_DisableClock
- AHB4ENCR GPIOQENC LL_AHB4_GRP1_DisableClock
- AHB4ENCR PWRENC LL_AHB4_GRP1_DisableClock
- AHB4ENCR CRCENC LL_AHB4_GRP1_DisableClock

LL_AHB4_GRP1_ForceReset

Function name

__STATIC_INLINE void LL_AHB4_GRP1_ForceReset (uint32_t Periphs)

Function description

Force AHB4 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB4_GRP1_PERIPH_GPIOA
  - LL_AHB4_GRP1_PERIPH_GPIOB
  - LL_AHB4_GRP1_PERIPH_GPIOC
  - LL_AHB4_GRP1_PERIPH_GPIOD
  - LL_AHB4_GRP1_PERIPH_GPIOE
  - LL_AHB4_GRP1_PERIPH_GPIOF
  - LL_AHB4_GRP1_PERIPH_GPIOG
  - LL_AHB4_GRP1_PERIPH_GPIOH
  - LL_AHB4_GRP1_PERIPH_GPION
  - LL_AHB4_GRP1_PERIPH_GPIOO
  - LL_AHB4_GRP1_PERIPH_GPIOP
  - LL_AHB4_GRP1_PERIPH_GPIOQ
  - LL_AHB4_GRP1_PERIPH_PWR
  - LL_AHB4_GRP1_PERIPH_CRC

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB4RSTSR GPIOARSTS LL_AHB4_GRP1_ForceReset
- AHB4RSTSR GPIOBRSTS LL_AHB4_GRP1_ForceReset
- AHB4RSTSR GPIOCRSTS LL_AHB4_GRP1_ForceReset
- AHB4RSTSR GPIODRSTS LL_AHB4_GRP1_ForceReset
- AHB4RSTSR GPIOERSTS LL_AHB4_GRP1_ForceReset
- AHB4RSTSR GPIOFRSTS LL_AHB4_GRP1_ForceReset
- AHB4RSTSR GPIOGRSTS LL_AHB4_GRP1_ForceReset
- AHB4RSTSR GPIOHRSTS LL_AHB4_GRP1_ForceReset
- AHB4RSTSR GPIONRSTS LL_AHB4_GRP1_ForceReset
- AHB4RSTSR GPIOORSTS LL_AHB4_GRP1_ForceReset
- AHB4RSTSR GPIOPRSTS LL_AHB4_GRP1_ForceReset
- AHB4RSTSR GPIOQRSTS LL_AHB4_GRP1_ForceReset
- AHB4RSTSR CRCRSTS LL_AHB4_GRP1_ForceReset
- AHB4RSTSR BKPRAMRSTS LL_AHB4_GRP1_ForceReset

LL_AHB4_GRP1_ReleaseReset

Function name

__STATIC_INLINE void LL_AHB4_GRP1_ReleaseReset (uint32_t Periphs)

Function description

Release AHB4 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB4_GRP1_PERIPH_GPIOA
  - LL_AHB4_GRP1_PERIPH_GPIOB
  - LL_AHB4_GRP1_PERIPH_GPIOC
  - LL_AHB4_GRP1_PERIPH_GPIOD
  - LL_AHB4_GRP1_PERIPH_GPIOE
  - LL_AHB4_GRP1_PERIPH_GPIOF
  - LL_AHB4_GRP1_PERIPH_GPIOG
  - LL_AHB4_GRP1_PERIPH_GPIOH
  - LL_AHB4_GRP1_PERIPH_GPION
  - LL_AHB4_GRP1_PERIPH_GPIOO
  - LL_AHB4_GRP1_PERIPH_GPIOP
  - LL_AHB4_GRP1_PERIPH_GPIOQ
  - LL_AHB4_GRP1_PERIPH_PWR
  - LL_AHB4_GRP1_PERIPH_CRC

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB4RSTCR GPIOARSTC LL_AHB4_GRP1_ReleaseReset
- AHB4RSTCR GPIOBRSTC LL_AHB4_GRP1_ReleaseReset
- AHB4RSTCR GPIOCRSTC LL_AHB4_GRP1_ReleaseReset
- AHB4RSTCR GPIODRSTC LL_AHB4_GRP1_ReleaseReset
- AHB4RSTCR GPIOERSTC LL_AHB4_GRP1_ReleaseReset
- AHB4RSTCR GPIOFRSTC LL_AHB4_GRP1_ReleaseReset
- AHB4RSTCR GPIOGRSTC LL_AHB4_GRP1_ReleaseReset
- AHB4RSTCR GPIOHRSTC LL_AHB4_GRP1_ReleaseReset
- AHB4RSTCR GPIONRSTC LL_AHB4_GRP1_ReleaseReset
- AHB4RSTCR GPIOORSTC LL_AHB4_GRP1_ReleaseReset
- AHB4RSTCR GPIOPRSTC LL_AHB4_GRP1_ReleaseReset
- AHB4RSTCR GPIOQRSTC LL_AHB4_GRP1_ReleaseReset
- AHB4RSTCR CRCRSTC LL_AHB4_GRP1_ReleaseReset
- AHB4RSTCR BKPRAMRSTC LL_AHB4_GRP1_ReleaseReset

LL_AHB4_GRP1_EnableClockLowPower

Function name

__STATIC_INLINE void LL_AHB4_GRP1_EnableClockLowPower (uint32_t Periphs)

Function description

Enable AHB4 peripherals clock during Low Power mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB4_GRP1_PERIPH_GPIOA
  - LL_AHB4_GRP1_PERIPH_GPIOB
  - LL_AHB4_GRP1_PERIPH_GPIOC
  - LL_AHB4_GRP1_PERIPH_GPIOD
  - LL_AHB4_GRP1_PERIPH_GPIOE
  - LL_AHB4_GRP1_PERIPH_GPIOF
  - LL_AHB4_GRP1_PERIPH_GPIOG
  - LL_AHB4_GRP1_PERIPH_GPIOH
  - LL_AHB4_GRP1_PERIPH_GPION
  - LL_AHB4_GRP1_PERIPH_GPIOO
  - LL_AHB4_GRP1_PERIPH_GPIOP
  - LL_AHB4_GRP1_PERIPH_GPIOQ
  - LL_AHB4_GRP1_PERIPH_PWR
  - LL_AHB4_GRP1_PERIPH_CRC

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB4LPENSR GPIOALPENS LL_AHB4_GRP1_EnableClockLowPower
- AHB4LPENSR GPIOBLPENS LL_AHB4_GRP1_EnableClockLowPower
- AHB4LPENSR GPIOCLPENS LL_AHB4_GRP1_EnableClockLowPower
- AHB4LPENSR GPIODLPENS LL_AHB4_GRP1_EnableClockLowPower
- AHB4LPENSR GPIOELPENS LL_AHB4_GRP1_EnableClockLowPower
- AHB4LPENSR GPIOFLPENS LL_AHB4_GRP1_EnableClockLowPower
- AHB4LPENSR GPIOGLPENS LL_AHB4_GRP1_EnableClockLowPower
- AHB4LPENSR GPIOHLPENS LL_AHB4_GRP1_EnableClockLowPower
- AHB4LPENSR GPIONLPENS LL_AHB4_GRP1_EnableClockLowPower
- AHB4LPENSR GPIOOLPENS LL_AHB4_GRP1_EnableClockLowPower
- AHB4LPENSR GPIOPLPENS LL_AHB4_GRP1_EnableClockLowPower
- AHB4LPENSR GPIOQLPENS LL_AHB4_GRP1_EnableClockLowPower
- AHB4LPENSR CRCLPENS LL_AHB4_GRP1_EnableClockLowPower
- AHB4LPENSR BKPRAMLPENS LL_AHB4_GRP1_EnableClockLowPower

LL_AHB4_GRP1_IsEnabledClockLowPower

Function name

__STATIC_INLINE uint32_t LL_AHB4_GRP1_IsEnabledClockLowPower (uint32_t Periphs)

Function description

Check if AHB4 peripheral clock during Low Power mode is enabled or not .

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB4_GRP1_PERIPH_GPIOA
  - LL_AHB4_GRP1_PERIPH_GPIOB
  - LL_AHB4_GRP1_PERIPH_GPIOC
  - LL_AHB4_GRP1_PERIPH_GPIOD
  - LL_AHB4_GRP1_PERIPH_GPIOE
  - LL_AHB4_GRP1_PERIPH_GPIOF
  - LL_AHB4_GRP1_PERIPH_GPIOG
  - LL_AHB4_GRP1_PERIPH_GPIOH
  - LL_AHB4_GRP1_PERIPH_GPION
  - LL_AHB4_GRP1_PERIPH_GPIOO
  - LL_AHB4_GRP1_PERIPH_GPIOP
  - LL_AHB4_GRP1_PERIPH_GPIOQ
  - LL_AHB4_GRP1_PERIPH_PWR
  - LL_AHB4_GRP1_PERIPH_CRC

Return values

- **uint32_t:**

Reference Manual to LL API cross reference:

- AHB4LPENR GPIOALPEN LL_AHB4_GRP1_IsEnabledClockLowPower
- AHB4LPENR GPIOBLPEN LL_AHB4_GRP1_IsEnabledClockLowPower
- AHB4LPENR GPIOCLPEN LL_AHB4_GRP1_IsEnabledClockLowPower
- AHB4LPENR GPIODLPEN LL_AHB4_GRP1_IsEnabledClockLowPower
- AHB4LPENR GPIOELPEN LL_AHB4_GRP1_IsEnabledClockLowPower
- AHB4LPENR GPIOFLPEN LL_AHB4_GRP1_IsEnabledClockLowPower
- AHB4LPENR GPIOGLPEN LL_AHB4_GRP1_IsEnabledClockLowPower
- AHB4LPENR GPIOHLPEN LL_AHB4_GRP1_IsEnabledClockLowPower
- AHB4LPENR GPIONLPEN LL_AHB4_GRP1_IsEnabledClockLowPower
- AHB4LPENR GPIOOLPEN LL_AHB4_GRP1_IsEnabledClockLowPower
- AHB4LPENR GPIOPLPEN LL_AHB4_GRP1_IsEnabledClockLowPower
- AHB4LPENR GPIOQLPEN LL_AHB4_GRP1_IsEnabledClockLowPower
- AHB4LPENR PWRLPEN LL_AHB4_GRP1_IsEnabledClockLowPower
- AHB4LPENR CRCLPEN LL_AHB4_GRP1_IsEnabledClockLowPower

LL_AHB4_GRP1_DisableClockLowPower

Function name

__STATIC_INLINE void LL_AHB4_GRP1_DisableClockLowPower (uint32_t Periphs)

Function description

Disable AHB4 peripherals clock during Low Power mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB4_GRP1_PERIPH_GPIOA
  - LL_AHB4_GRP1_PERIPH_GPIOB
  - LL_AHB4_GRP1_PERIPH_GPIOC
  - LL_AHB4_GRP1_PERIPH_GPIOD
  - LL_AHB4_GRP1_PERIPH_GPIOE
  - LL_AHB4_GRP1_PERIPH_GPIOF
  - LL_AHB4_GRP1_PERIPH_GPIOG
  - LL_AHB4_GRP1_PERIPH_GPIOH
  - LL_AHB4_GRP1_PERIPH_GPION
  - LL_AHB4_GRP1_PERIPH_GPIOO
  - LL_AHB4_GRP1_PERIPH_GPIOP
  - LL_AHB4_GRP1_PERIPH_GPIOQ
  - LL_AHB4_GRP1_PERIPH_PWR
  - LL_AHB4_GRP1_PERIPH_CRC

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB4LPENCR GPIOALPENC LL_AHB4_GRP1_DisableClockLowPower
- AHB4LPENCR GPIOBLPENC LL_AHB4_GRP1_DisableClockLowPower
- AHB4LPENCR GPIOCLPENC LL_AHB4_GRP1_DisableClockLowPower
- AHB4LPENCR GPIODLPENC LL_AHB4_GRP1_DisableClockLowPower
- AHB4LPENCR GPIOELPENC LL_AHB4_GRP1_DisableClockLowPower
- AHB4LPENCR GPIOFLPENC LL_AHB4_GRP1_DisableClockLowPower
- AHB4LPENCR GPIOGLPENC LL_AHB4_GRP1_DisableClockLowPower
- AHB4LPENCR GPIOHLPENC LL_AHB4_GRP1_DisableClockLowPower
- AHB4LPENCR GPIONLPENC LL_AHB4_GRP1_DisableClockLowPower
- AHB4LPENCR GPIOOLPENC LL_AHB4_GRP1_DisableClockLowPower
- AHB4LPENCR GPIOPLPENC LL_AHB4_GRP1_DisableClockLowPower
- AHB4LPENCR GPIOQLPENC LL_AHB4_GRP1_DisableClockLowPower
- AHB4LPENCR PWRLPENC LL_AHB4_GRP1_DisableClockLowPower
- AHB4LPENCR CRCLPENC LL_AHB4_GRP1_DisableClockLowPower

LL_AHB5_GRP1_EnableClock

Function name

__STATIC_INLINE void LL_AHB5_GRP1_EnableClock (uint32_t Periphs)

Function description

Enable AHB5 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB5_GRP1_PERIPH_HPDMA1
  - LL_AHB5_GRP1_PERIPH_DMA2D
  - LL_AHB5_GRP1_PERIPH_JPEG
  - LL_AHB5_GRP1_PERIPH_FMC
  - LL_AHB5_GRP1_PERIPH_XSPI1
  - LL_AHB5_GRP1_PERIPH_XSPI2
  - LL_AHB5_GRP1_PERIPH_XSPI3
  - LL_AHB5_GRP1_PERIPH_XSPIM
  - LL_AHB5_GRP1_PERIPH_PSSI
  - LL_AHB5_GRP1_PERIPH_SDMMC1
  - LL_AHB5_GRP1_PERIPH_SDMMC2
  - LL_AHB5_GRP1_PERIPH_MCE1 (*)
  - LL_AHB5_GRP1_PERIPH_MCE2 (*)
  - LL_AHB5_GRP1_PERIPH_MCE3 (*)
  - LL_AHB5_GRP1_PERIPH_MCE4 (*)
  - LL_AHB5_GRP1_PERIPH_GFXMMU
  - LL_AHB5_GRP1_PERIPH_GPU2D
  - LL_AHB5_GRP1_PERIPH_ETH1
  - LL_AHB5_GRP1_PERIPH_ETH1MAC
  - LL_AHB5_GRP1_PERIPH_ETH1TX
  - LL_AHB5_GRP1_PERIPH_ETH1RX
  - LL_AHB5_GRP1_PERIPH_OTG1
  - LL_AHB5_GRP1_PERIPH_OTG2
  - LL_AHB5_GRP1_PERIPH_OTGPHY1
  - LL_AHB5_GRP1_PERIPH_OTGPHY2
  - LL_AHB5_GRP1_PERIPH_CACHEAXI
  - LL_AHB5_GRP1_PERIPH_NPU

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB5ENSR HPDMA1ENS LL_AHB5_GRP1_EnableClock
- AHB5ENSR DMA2DENS LL_AHB5_GRP1_EnableClock
- AHB5ENSR JPEGENS LL_AHB5_GRP1_EnableClock
- AHB5ENSR FMCENS LL_AHB5_GRP1_EnableClock
- AHB5ENSR XSPI1ENS LL_AHB5_GRP1_EnableClock
- AHB5ENSR PSSIENS LL_AHB5_GRP1_EnableClock
- AHB5ENSR SDMMC2ENS LL_AHB5_GRP1_EnableClock
- AHB5ENSR SDMMC1ENS LL_AHB5_GRP1_EnableClock
- AHB5ENSR XSPI2ENS LL_AHB5_GRP1_EnableClock
- AHB5ENSR XSPIMENS LL_AHB5_GRP1_EnableClock
- AHB5ENSR MCE1ENS LL_AHB5_GRP1_EnableClock
- (*) AHB5ENSR MCE2ENS LL_AHB5_GRP1_EnableClock
- (*) AHB5ENSR MCE3ENS LL_AHB5_GRP1_EnableClock
- (*) AHB5ENSR XSPI3ENS LL_AHB5_GRP1_EnableClock
- AHB5ENSR MCE4ENS LL_AHB5_GRP1_EnableClock
- (*) AHB5ENSR GFXMMUENS LL_AHB5_GRP1_EnableClock
- AHB5ENSR GPU2DENS LL_AHB5_GRP1_EnableClock
- AHB5ENSR ETH1MACENS LL_AHB5_GRP1_EnableClock
- AHB5ENSR ETH1TXENS LL_AHB5_GRP1_EnableClock
- AHB5ENSR ETH1RXENS LL_AHB5_GRP1_EnableClock
- AHB5ENSR ETH1ENS LL_AHB5_GRP1_EnableClock
- AHB5ENSR OTG1ENS LL_AHB5_GRP1_EnableClock
- AHB5ENSR OTGPHY1ENS LL_AHB5_GRP1_EnableClock
- AHB5ENSR OTGPHY2ENS LL_AHB5_GRP1_EnableClock
- AHB5ENSR OTG2ENS LL_AHB5_GRP1_EnableClock
- AHB5ENSR CACHEAXIENS LL_AHB5_GRP1_EnableClock
- AHB5ENSR NPUENS LL_AHB5_GRP1_EnableClock

LL_AHB5_GRP1_IsEnabledClock

Function name

__STATIC_INLINE uint32_t LL_AHB5_GRP1_IsEnabledClock (uint32_t Periphs)

Function description

Check if AHB5 peripheral clock is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB5_GRP1_PERIPH_HPDMA1
  - LL_AHB5_GRP1_PERIPH_DMA2D
  - LL_AHB5_GRP1_PERIPH_JPEG
  - LL_AHB5_GRP1_PERIPH_FMC
  - LL_AHB5_GRP1_PERIPH_XSPI1
  - LL_AHB5_GRP1_PERIPH_XSPI2
  - LL_AHB5_GRP1_PERIPH_XSPI3
  - LL_AHB5_GRP1_PERIPH_XSPIM
  - LL_AHB5_GRP1_PERIPH_PSSI
  - LL_AHB5_GRP1_PERIPH_SDMMC1
  - LL_AHB5_GRP1_PERIPH_SDMMC2
  - LL_AHB5_GRP1_PERIPH_MCE1 (*)
  - LL_AHB5_GRP1_PERIPH_MCE2 (*)
  - LL_AHB5_GRP1_PERIPH_MCE3 (*)
  - LL_AHB5_GRP1_PERIPH_MCE4 (*)
  - LL_AHB5_GRP1_PERIPH_GFXMMU
  - LL_AHB5_GRP1_PERIPH_GPU2D
  - LL_AHB5_GRP1_PERIPH_ETH1
  - LL_AHB5_GRP1_PERIPH_ETH1MAC
  - LL_AHB5_GRP1_PERIPH_ETH1TX
  - LL_AHB5_GRP1_PERIPH_ETH1RX
  - LL_AHB5_GRP1_PERIPH_OTG1
  - LL_AHB5_GRP1_PERIPH_OTG2
  - LL_AHB5_GRP1_PERIPH_OTGPHY1
  - LL_AHB5_GRP1_PERIPH_OTGPHY2
  - LL_AHB5_GRP1_PERIPH_CACHEAXI
  - LL_AHB5_GRP1_PERIPH_NPU

  (*) value not defined in all devices.

Return values

- **uint32_t:**

Reference Manual to LL API cross reference:

- AHB5ENR HPDMA1EN LL_AHB5_GRP1_IsEnabledClock
- AHB5ENR DMA2DEN LL_AHB5_GRP1_IsEnabledClock
- AHB5ENR JPEGEN LL_AHB5_GRP1_IsEnabledClock
- AHB5ENR FMCEN LL_AHB5_GRP1_IsEnabledClock
- AHB5ENR XSPI1EN LL_AHB5_GRP1_IsEnabledClock
- AHB5ENR PSSIEN LL_AHB5_GRP1_IsEnabledClock
- AHB5ENR SDMMC2EN LL_AHB5_GRP1_IsEnabledClock
- AHB5ENR SDMMC1EN LL_AHB5_GRP1_IsEnabledClock
- AHB5ENR XSPI2EN LL_AHB5_GRP1_IsEnabledClock
- AHB5ENR XSPIMEN LL_AHB5_GRP1_IsEnabledClock
- AHB5ENR MCE1EN LL_AHB5_GRP1_IsEnabledClock
- (*) AHB5ENR MCE2EN LL_AHB5_GRP1_IsEnabledClock
- (*) AHB5ENR MCE3EN LL_AHB5_GRP1_IsEnabledClock
- (*) AHB5ENR XSPI3EN LL_AHB5_GRP1_IsEnabledClock
- AHB5ENR MCE4EN LL_AHB5_GRP1_IsEnabledClock
- (*) AHB5ENR GFXMMUEN LL_AHB5_GRP1_IsEnabledClock
- AHB5ENR GPU2DEN LL_AHB5_GRP1_IsEnabledClock
- AHB5ENR ETH1MACEN LL_AHB5_GRP1_IsEnabledClock
- AHB5ENR ETH1TXEN LL_AHB5_GRP1_IsEnabledClock
- AHB5ENR ETH1RXEN LL_AHB5_GRP1_IsEnabledClock
- AHB5ENR ETH1EN LL_AHB5_GRP1_IsEnabledClock
- AHB5ENR OTG1EN LL_AHB5_GRP1_IsEnabledClock
- AHB5ENR OTGPHY1EN LL_AHB5_GRP1_IsEnabledClock
- AHB5ENR OTGPHY2EN LL_AHB5_GRP1_IsEnabledClock
- AHB5ENR OTG2EN LL_AHB5_GRP1_IsEnabledClock
- AHB5ENR CACHEAXIEN LL_AHB5_GRP1_IsEnabledClock
- AHB5ENR NPUEN LL_AHB5_GRP1_IsEnabledClock

LL_AHB5_GRP1_DisableClock

Function name

__STATIC_INLINE void LL_AHB5_GRP1_DisableClock (uint32_t Periphs)

Function description

Disable AHB5 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB5_GRP1_PERIPH_HPDMA1
  - LL_AHB5_GRP1_PERIPH_DMA2D
  - LL_AHB5_GRP1_PERIPH_JPEG
  - LL_AHB5_GRP1_PERIPH_FMC
  - LL_AHB5_GRP1_PERIPH_XSPI1
  - LL_AHB5_GRP1_PERIPH_XSPI2
  - LL_AHB5_GRP1_PERIPH_XSPI3
  - LL_AHB5_GRP1_PERIPH_XSPIM
  - LL_AHB5_GRP1_PERIPH_PSSI
  - LL_AHB5_GRP1_PERIPH_SDMMC1
  - LL_AHB5_GRP1_PERIPH_SDMMC2
  - LL_AHB5_GRP1_PERIPH_MCE1 (*)
  - LL_AHB5_GRP1_PERIPH_MCE2 (*)
  - LL_AHB5_GRP1_PERIPH_MCE3 (*)
  - LL_AHB5_GRP1_PERIPH_MCE4 (*)
  - LL_AHB5_GRP1_PERIPH_GFXMMU
  - LL_AHB5_GRP1_PERIPH_GPU2D
  - LL_AHB5_GRP1_PERIPH_ETH1
  - LL_AHB5_GRP1_PERIPH_ETH1MAC
  - LL_AHB5_GRP1_PERIPH_ETH1TX
  - LL_AHB5_GRP1_PERIPH_ETH1RX
  - LL_AHB5_GRP1_PERIPH_OTG1
  - LL_AHB5_GRP1_PERIPH_OTG2
  - LL_AHB5_GRP1_PERIPH_OTGPHY1
  - LL_AHB5_GRP1_PERIPH_OTGPHY2
  - LL_AHB5_GRP1_PERIPH_CACHEAXI
  - LL_AHB5_GRP1_PERIPH_NPU

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB5ENCR HPDMA1ENC LL_AHB5_GRP1_DisableClock
- AHB5ENCR DMA2DENC LL_AHB5_GRP1_DisableClock
- AHB5ENCR JPEGENC LL_AHB5_GRP1_DisableClock
- AHB5ENCR FMCENC LL_AHB5_GRP1_DisableClock
- AHB5ENCR XSPI1ENC LL_AHB5_GRP1_DisableClock
- AHB5ENCR PSSIENC LL_AHB5_GRP1_DisableClock
- AHB5ENCR SDMMC2ENC LL_AHB5_GRP1_DisableClock
- AHB5ENCR SDMMC1ENC LL_AHB5_GRP1_DisableClock
- AHB5ENCR XSPI2ENC LL_AHB5_GRP1_DisableClock
- AHB5ENCR XSPIMENC LL_AHB5_GRP1_DisableClock
- AHB5ENCR MCE1ENC LL_AHB5_GRP1_DisableClock
- (*) AHB5ENCR MCE2ENC LL_AHB5_GRP1_DisableClock
- (*) AHB5ENCR MCE3ENC LL_AHB5_GRP1_DisableClock
- (*) AHB5ENCR XSPI3ENC LL_AHB5_GRP1_DisableClock
- AHB5ENCR MCE4ENC LL_AHB5_GRP1_DisableClock
- (*) AHB5ENCR GFXMMUENC LL_AHB5_GRP1_DisableClock
- AHB5ENCR GPU2DENC LL_AHB5_GRP1_DisableClock
- AHB5ENCR ETH1MACENC LL_AHB5_GRP1_DisableClock
- AHB5ENCR ETH1TXENC LL_AHB5_GRP1_DisableClock
- AHB5ENCR ETH1RXENC LL_AHB5_GRP1_DisableClock
- AHB5ENCR ETH1ENC LL_AHB5_GRP1_DisableClock
- AHB5ENCR OTG1ENC LL_AHB5_GRP1_DisableClock
- AHB5ENCR OTGPHY1ENC LL_AHB5_GRP1_DisableClock
- AHB5ENCR OTGPHY2ENC LL_AHB5_GRP1_DisableClock
- AHB5ENCR OTG2ENC LL_AHB5_GRP1_DisableClock
- AHB5ENCR CACHEAXIENC LL_AHB5_GRP1_DisableClock
- AHB5ENCR NPUENC LL_AHB5_GRP1_DisableClock

LL_AHB5_GRP1_ForceReset

Function name

__STATIC_INLINE void LL_AHB5_GRP1_ForceReset (uint32_t Periphs)

Function description

Force AHB5 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB5_GRP1_PERIPH_HPDMA1
  - LL_AHB5_GRP1_PERIPH_DMA2D
  - LL_AHB5_GRP1_PERIPH_JPEG
  - LL_AHB5_GRP1_PERIPH_FMC
  - LL_AHB5_GRP1_PERIPH_XSPI1
  - LL_AHB5_GRP1_PERIPH_XSPI2
  - LL_AHB5_GRP1_PERIPH_XSPI3
  - LL_AHB5_GRP1_PERIPH_XSPIM
  - LL_AHB5_GRP1_PERIPH_PSSI
  - LL_AHB5_GRP1_PERIPH_SDMMC1
  - LL_AHB5_GRP1_PERIPH_SDMMC2
  - LL_AHB5_GRP1_PERIPH_GFXMMU
  - LL_AHB5_GRP1_PERIPH_GPU2D
  - LL_AHB5_GRP1_PERIPH_OTG1PHYCTL
  - LL_AHB5_GRP1_PERIPH_OTG2PHYCTL
  - LL_AHB5_GRP1_PERIPH_ETH1
  - LL_AHB5_GRP1_PERIPH_OTG1
  - LL_AHB5_GRP1_PERIPH_OTG2
  - LL_AHB5_GRP1_PERIPH_OTGPHY1
  - LL_AHB5_GRP1_PERIPH_OTGPHY2
  - LL_AHB5_GRP1_PERIPH_CACHEAXI
  - LL_AHB5_GRP1_PERIPH_NPU

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB5RSTSR HPDMA1RSTS LL_AHB5_GRP1_ForceReset
- AHB5RSTSR DMA2DRSTS LL_AHB5_GRP1_ForceReset
- AHB5RSTSR JPEGRSTS LL_AHB5_GRP1_ForceReset
- AHB5RSTSR FMCRSTS LL_AHB5_GRP1_ForceReset
- AHB5RSTSR XSPI1RSTS LL_AHB5_GRP1_ForceReset
- AHB5RSTSR PSSIRSTS LL_AHB5_GRP1_ForceReset
- AHB5RSTSR SDMMC2RSTS LL_AHB5_GRP1_ForceReset
- AHB5RSTSR SDMMC1RSTS LL_AHB5_GRP1_ForceReset
- AHB5RSTSR XSPI2RSTS LL_AHB5_GRP1_ForceReset
- AHB5RSTSR XSPIMRSTS LL_AHB5_GRP1_ForceReset
- AHB5RSTSR XSPI3RSTS LL_AHB5_GRP1_ForceReset
- AHB5RSTSR GFXMMURSTS LL_AHB5_GRP1_ForceReset
- AHB5RSTSR GPU2DRSTS LL_AHB5_GRP1_ForceReset
- AHB5RSTSR OTG1PHYCTLRSTS LL_AHB5_GRP1_ForceReset
- AHB5RSTSR OTG2PHYCTLRSTS LL_AHB5_GRP1_ForceReset
- AHB5RSTSR ETH1RSTS LL_AHB5_GRP1_ForceReset
- AHB5RSTSR OTG1RSTS LL_AHB5_GRP1_ForceReset
- AHB5RSTSR OTGPHY1RSTS LL_AHB5_GRP1_ForceReset
- AHB5RSTSR OTGPHY2RSTS LL_AHB5_GRP1_ForceReset
- AHB5RSTSR OTG2RSTS LL_AHB5_GRP1_ForceReset
- AHB5RSTSR CACHEAXIRSTS LL_AHB5_GRP1_ForceReset
- AHB5RSTSR NPURSTS LL_AHB5_GRP1_ForceReset

LL_AHB5_GRP1_ReleaseReset

Function name

__STATIC_INLINE void LL_AHB5_GRP1_ReleaseReset (uint32_t Periphs)

Function description

Release AHB5 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB5_GRP1_PERIPH_HPDMA1
  - LL_AHB5_GRP1_PERIPH_DMA2D
  - LL_AHB5_GRP1_PERIPH_JPEG
  - LL_AHB5_GRP1_PERIPH_FMC
  - LL_AHB5_GRP1_PERIPH_XSPI1
  - LL_AHB5_GRP1_PERIPH_XSPI2
  - LL_AHB5_GRP1_PERIPH_XSPI3
  - LL_AHB5_GRP1_PERIPH_XSPIM
  - LL_AHB5_GRP1_PERIPH_PSSI
  - LL_AHB5_GRP1_PERIPH_SDMMC1
  - LL_AHB5_GRP1_PERIPH_SDMMC2
  - LL_AHB5_GRP1_PERIPH_GFXMMU
  - LL_AHB5_GRP1_PERIPH_GPU2D
  - LL_AHB5_GRP1_PERIPH_OTG1PHYCTL
  - LL_AHB5_GRP1_PERIPH_OTG2PHYCTL
  - LL_AHB5_GRP1_PERIPH_ETH1
  - LL_AHB5_GRP1_PERIPH_OTG1
  - LL_AHB5_GRP1_PERIPH_OTG2
  - LL_AHB5_GRP1_PERIPH_OTGPHY1
  - LL_AHB5_GRP1_PERIPH_OTGPHY2
  - LL_AHB5_GRP1_PERIPH_CACHEAXI
  - LL_AHB5_GRP1_PERIPH_NPU

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB5RSTCR HPDMA1RSTC LL_AHB5_GRP1_ReleaseReset
- AHB5RSTCR DMA2DRSTC LL_AHB5_GRP1_ReleaseReset
- AHB5RSTCR JPEGRSTC LL_AHB5_GRP1_ReleaseReset
- AHB5RSTCR FMCRSTC LL_AHB5_GRP1_ReleaseReset
- AHB5RSTCR XSPI1RSTC LL_AHB5_GRP1_ReleaseReset
- AHB5RSTCR PSSIRSTC LL_AHB5_GRP1_ReleaseReset
- AHB5RSTCR SDMMC2RSTC LL_AHB5_GRP1_ReleaseReset
- AHB5RSTCR SDMMC1RSTC LL_AHB5_GRP1_ReleaseReset
- AHB5RSTCR XSPI2RSTC LL_AHB5_GRP1_ReleaseReset
- AHB5RSTCR XSPIMRSTC LL_AHB5_GRP1_ReleaseReset
- AHB5RSTCR XSPI3RSTC LL_AHB5_GRP1_ReleaseReset
- AHB5RSTCR GFXMMURSTC LL_AHB5_GRP1_ReleaseReset
- AHB5RSTCR GPU2DRSTC LL_AHB5_GRP1_ReleaseReset
- AHB5RSTCR OTG1PHYCTLRSTC LL_AHB5_GRP1_ReleaseReset
- AHB5RSTCR OTG2PHYCTLRSTC LL_AHB5_GRP1_ReleaseReset
- AHB5RSTCR ETH1RSTC LL_AHB5_GRP1_ReleaseReset
- AHB5RSTCR OTG1RSTC LL_AHB5_GRP1_ReleaseReset
- AHB5RSTCR OTGPHY1RSTC LL_AHB5_GRP1_ReleaseReset
- AHB5RSTCR OTGPHY2RSTC LL_AHB5_GRP1_ReleaseReset
- AHB5RSTCR OTG2RSTC LL_AHB5_GRP1_ReleaseReset
- AHB5RSTCR CACHEAXIRSTC LL_AHB5_GRP1_ReleaseReset
- AHB5RSTCR NPURSTS LL_AHB5_GRP1_ReleaseReset

LL_AHB5_GRP1_EnableClockLowPower

Function name

__STATIC_INLINE void LL_AHB5_GRP1_EnableClockLowPower (uint32_t Periphs)

Function description

Enable AHB5 peripherals clock during Low Power mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB5_GRP1_PERIPH_HPDMA1
  - LL_AHB5_GRP1_PERIPH_DMA2D
  - LL_AHB5_GRP1_PERIPH_JPEG
  - LL_AHB5_GRP1_PERIPH_FMC
  - LL_AHB5_GRP1_PERIPH_XSPI1
  - LL_AHB5_GRP1_PERIPH_XSPI2
  - LL_AHB5_GRP1_PERIPH_XSPI3
  - LL_AHB5_GRP1_PERIPH_XSPIM
  - LL_AHB5_GRP1_PERIPH_PSSI
  - LL_AHB5_GRP1_PERIPH_SDMMC1
  - LL_AHB5_GRP1_PERIPH_SDMMC2
  - LL_AHB5_GRP1_PERIPH_MCE1 (*)
  - LL_AHB5_GRP1_PERIPH_MCE2 (*)
  - LL_AHB5_GRP1_PERIPH_MCE3 (*)
  - LL_AHB5_GRP1_PERIPH_MCE4 (*)
  - LL_AHB5_GRP1_PERIPH_GFXMMU
  - LL_AHB5_GRP1_PERIPH_GPU2D
  - LL_AHB5_GRP1_PERIPH_ETH1
  - LL_AHB5_GRP1_PERIPH_ETH1MAC
  - LL_AHB5_GRP1_PERIPH_ETH1TX
  - LL_AHB5_GRP1_PERIPH_ETH1RX
  - LL_AHB5_GRP1_PERIPH_OTG1
  - LL_AHB5_GRP1_PERIPH_OTG2
  - LL_AHB5_GRP1_PERIPH_OTGPHY1
  - LL_AHB5_GRP1_PERIPH_OTGPHY2
  - LL_AHB5_GRP1_PERIPH_CACHEAXI
  - LL_AHB5_GRP1_PERIPH_NPU

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB5LPENSR HPDMA1LPENS LL_AHB5_GRP1_EnableClockLowPower
- AHB5LPENSR DMA2DLPENS LL_AHB5_GRP1_EnableClockLowPower
- AHB5LPENSR JPEGLPENS LL_AHB5_GRP1_EnableClockLowPower
- AHB5LPENSR FMCLPENS LL_AHB5_GRP1_EnableClockLowPower
- AHB5LPENSR XSPI1LPENS LL_AHB5_GRP1_EnableClockLowPower
- AHB5LPENSR PSSILPENS LL_AHB5_GRP1_EnableClockLowPower
- AHB5LPENSR SDMMC2LPENS LL_AHB5_GRP1_EnableClockLowPower
- AHB5LPENSR SDMMC1LPENS LL_AHB5_GRP1_EnableClockLowPower
- AHB5LPENSR XSPI2LPENS LL_AHB5_GRP1_EnableClockLowPower
- AHB5LPENSR XSPIMLPENS LL_AHB5_GRP1_EnableClockLowPower
- AHB5LPENSR MCE1LPENS LL_AHB5_GRP1_EnableClockLowPower
- (*) AHB5LPENSR MCE2LPENS LL_AHB5_GRP1_EnableClockLowPower
- (*) AHB5LPENSR MCE3LPENS LL_AHB5_GRP1_EnableClockLowPower
- (*) AHB5LPENSR XSPI3LPENS LL_AHB5_GRP1_EnableClockLowPower
- AHB5LPENSR MCE4LPENS LL_AHB5_GRP1_EnableClockLowPower
- (*) AHB5LPENSR GFXMMULPENS LL_AHB5_GRP1_EnableClockLowPower
- AHB5LPENSR GPU2DLPENS LL_AHB5_GRP1_EnableClockLowPower
- AHB5LPENSR ETH1MACLPENS LL_AHB5_GRP1_EnableClockLowPower
- AHB5LPENSR ETH1TXLPENS LL_AHB5_GRP1_EnableClockLowPower
- AHB5LPENSR ETH1RXLPENS LL_AHB5_GRP1_EnableClockLowPower
- AHB5LPENSR ETH1LPENS LL_AHB5_GRP1_EnableClockLowPower
- AHB5LPENSR OTG1LPENS LL_AHB5_GRP1_EnableClockLowPower
- AHB5LPENSR OTGPHY1LPENS LL_AHB5_GRP1_EnableClockLowPower
- AHB5LPENSR OTGPHY2LPENS LL_AHB5_GRP1_EnableClockLowPower
- AHB5LPENSR OTG2LPENS LL_AHB5_GRP1_EnableClockLowPower
- AHB5LPENSR CACHEAXILPENS LL_AHB5_GRP1_EnableClockLowPower
- AHB5LPENSR NPULPENS LL_AHB5_GRP1_EnableClockLowPower

LL_AHB5_GRP1_IsEnabledClockLowPower

Function name

__STATIC_INLINE uint32_t LL_AHB5_GRP1_IsEnabledClockLowPower (uint32_t Periphs)

Function description

Check if AHB5 peripheral clock during Low Power mode is enabled or not .

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB5_GRP1_PERIPH_HPDMA1
  - LL_AHB5_GRP1_PERIPH_DMA2D
  - LL_AHB5_GRP1_PERIPH_JPEG
  - LL_AHB5_GRP1_PERIPH_FMC
  - LL_AHB5_GRP1_PERIPH_XSPI1
  - LL_AHB5_GRP1_PERIPH_XSPI2
  - LL_AHB5_GRP1_PERIPH_XSPI3
  - LL_AHB5_GRP1_PERIPH_XSPIM
  - LL_AHB5_GRP1_PERIPH_PSSI
  - LL_AHB5_GRP1_PERIPH_SDMMC1
  - LL_AHB5_GRP1_PERIPH_SDMMC2
  - LL_AHB5_GRP1_PERIPH_MCE1 (*)
  - LL_AHB5_GRP1_PERIPH_MCE2 (*)
  - LL_AHB5_GRP1_PERIPH_MCE3 (*)
  - LL_AHB5_GRP1_PERIPH_MCE4 (*)
  - LL_AHB5_GRP1_PERIPH_GFXMMU
  - LL_AHB5_GRP1_PERIPH_GPU2D
  - LL_AHB5_GRP1_PERIPH_ETH1
  - LL_AHB5_GRP1_PERIPH_ETH1MAC
  - LL_AHB5_GRP1_PERIPH_ETH1TX
  - LL_AHB5_GRP1_PERIPH_ETH1RX
  - LL_AHB5_GRP1_PERIPH_OTG1
  - LL_AHB5_GRP1_PERIPH_OTG2
  - LL_AHB5_GRP1_PERIPH_OTGPHY1
  - LL_AHB5_GRP1_PERIPH_OTGPHY2
  - LL_AHB5_GRP1_PERIPH_CACHEAXI
  - LL_AHB5_GRP1_PERIPH_NPU

  (*) value not defined in all devices.

Return values

- **uint32_t:**

Reference Manual to LL API cross reference:

- AHB5LPENR HPDMA1LPEN LL_AHB5_GRP1_IsEnabledClockLowPower
- AHB5LPENR DMA2DLPEN LL_AHB5_GRP1_IsEnabledClockLowPower
- AHB5LPENR JPEGLPEN LL_AHB5_GRP1_IsEnabledClockLowPower
- AHB5LPENR FMCLPEN LL_AHB5_GRP1_IsEnabledClockLowPower
- AHB5LPENR XSPI1LPEN LL_AHB5_GRP1_IsEnabledClockLowPower
- AHB5LPENR PSSILPEN LL_AHB5_GRP1_IsEnabledClockLowPower
- AHB5LPENR SDMMC2LPEN LL_AHB5_GRP1_IsEnabledClockLowPower
- AHB5LPENR SDMMC1LPEN LL_AHB5_GRP1_IsEnabledClockLowPower
- AHB5LPENR XSPI2LPEN LL_AHB5_GRP1_IsEnabledClockLowPower
- AHB5LPENR XSPIMLPEN LL_AHB5_GRP1_IsEnabledClockLowPower
- AHB5LPENR MCE1LPEN LL_AHB5_GRP1_IsEnabledClockLowPower
- (*) AHB5LPENR MCE2LPEN LL_AHB5_GRP1_IsEnabledClockLowPower
- (*) AHB5LPENR MCE3LPEN LL_AHB5_GRP1_IsEnabledClockLowPower
- (*) AHB5LPENR XSPI3LPEN LL_AHB5_GRP1_IsEnabledClockLowPower
- AHB5LPENR MCE4LPEN LL_AHB5_GRP1_IsEnabledClockLowPower
- (*) AHB5LPENR GFXMMULPEN LL_AHB5_GRP1_IsEnabledClockLowPower
- AHB5LPENR GPU2DLPEN LL_AHB5_GRP1_IsEnabledClockLowPower
- AHB5LPENR ETH1MACLPEN LL_AHB5_GRP1_IsEnabledClockLowPower
- AHB5LPENR ETH1TXLPEN LL_AHB5_GRP1_IsEnabledClockLowPower
- AHB5LPENR ETH1RXLPEN LL_AHB5_GRP1_IsEnabledClockLowPower
- AHB5LPENR ETH1LPEN LL_AHB5_GRP1_IsEnabledClockLowPower
- AHB5LPENR OTG1LPEN LL_AHB5_GRP1_IsEnabledClockLowPower
- AHB5LPENR OTGPHY1LPEN LL_AHB5_GRP1_IsEnabledClockLowPower
- AHB5LPENR OTGPHY2LPEN LL_AHB5_GRP1_IsEnabledClockLowPower
- AHB5LPENR OTG2LPEN LL_AHB5_GRP1_IsEnabledClockLowPower
- AHB5LPENR CACHEAXILPEN LL_AHB5_GRP1_IsEnabledClockLowPower
- AHB5LPENR NPULPEN LL_AHB5_GRP1_IsEnabledClockLowPower

LL_AHB5_GRP1_DisableClockLowPower

Function name

__STATIC_INLINE void LL_AHB5_GRP1_DisableClockLowPower (uint32_t Periphs)

Function description

Disable AHB5 peripherals clock during Low Power mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB5_GRP1_PERIPH_HPDMA1
  - LL_AHB5_GRP1_PERIPH_DMA2D
  - LL_AHB5_GRP1_PERIPH_JPEG
  - LL_AHB5_GRP1_PERIPH_FMC
  - LL_AHB5_GRP1_PERIPH_XSPI1
  - LL_AHB5_GRP1_PERIPH_XSPI2
  - LL_AHB5_GRP1_PERIPH_XSPI3
  - LL_AHB5_GRP1_PERIPH_XSPIM
  - LL_AHB5_GRP1_PERIPH_PSSI
  - LL_AHB5_GRP1_PERIPH_SDMMC1
  - LL_AHB5_GRP1_PERIPH_SDMMC2
  - LL_AHB5_GRP1_PERIPH_MCE1 (*)
  - LL_AHB5_GRP1_PERIPH_MCE2 (*)
  - LL_AHB5_GRP1_PERIPH_MCE3 (*)
  - LL_AHB5_GRP1_PERIPH_MCE4 (*)
  - LL_AHB5_GRP1_PERIPH_GFXMMU
  - LL_AHB5_GRP1_PERIPH_GPU2D
  - LL_AHB5_GRP1_PERIPH_ETH1
  - LL_AHB5_GRP1_PERIPH_ETH1MAC
  - LL_AHB5_GRP1_PERIPH_ETH1TX
  - LL_AHB5_GRP1_PERIPH_ETH1RX
  - LL_AHB5_GRP1_PERIPH_OTG1
  - LL_AHB5_GRP1_PERIPH_OTG2
  - LL_AHB5_GRP1_PERIPH_OTGPHY1
  - LL_AHB5_GRP1_PERIPH_OTGPHY2
  - LL_AHB5_GRP1_PERIPH_CACHEAXI
  - LL_AHB5_GRP1_PERIPH_NPU

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB5LPENCR HPDMA1LPENC LL_AHB5_GRP1_DisableClockLowPower
- AHB5LPENCR DMA2DLPENC LL_AHB5_GRP1_DisableClockLowPower
- AHB5LPENCR JPEGLPENC LL_AHB5_GRP1_DisableClockLowPower
- AHB5LPENCR FMCLPENC LL_AHB5_GRP1_DisableClockLowPower
- AHB5LPENCR XSPI1LPENC LL_AHB5_GRP1_DisableClockLowPower
- AHB5LPENCR PSSILPENC LL_AHB5_GRP1_DisableClockLowPower
- AHB5LPENCR SDMMC2LPENC LL_AHB5_GRP1_DisableClockLowPower
- AHB5LPENCR SDMMC1LPENC LL_AHB5_GRP1_DisableClockLowPower
- AHB5LPENCR XSPI2LPENC LL_AHB5_GRP1_DisableClockLowPower
- AHB5LPENCR XSPIMLPENC LL_AHB5_GRP1_DisableClockLowPower
- AHB5LPENCR MCE1LPENC LL_AHB5_GRP1_DisableClockLowPower
- (*) AHB5LPENCR MCE2LPENC LL_AHB5_GRP1_DisableClockLowPower
- (*) AHB5LPENCR MCE3LPENC LL_AHB5_GRP1_DisableClockLowPower
- (*) AHB5LPENCR XSPI3LPENC LL_AHB5_GRP1_DisableClockLowPower
- AHB5LPENCR MCE4LPENC LL_AHB5_GRP1_DisableClockLowPower
- (*) AHB5LPENCR GFXMMULPENC LL_AHB5_GRP1_DisableClockLowPower
- AHB5LPENCR GPU2DLPENC LL_AHB5_GRP1_DisableClockLowPower
- AHB5LPENCR ETH1MACLPENC LL_AHB5_GRP1_DisableClockLowPower
- AHB5LPENCR ETH1TXLPENC LL_AHB5_GRP1_DisableClockLowPower
- AHB5LPENCR ETH1RXLPENC LL_AHB5_GRP1_DisableClockLowPower
- AHB5LPENCR ETH1LPENC LL_AHB5_GRP1_DisableClockLowPower
- AHB5LPENCR OTG1LPENC LL_AHB5_GRP1_DisableClockLowPower
- AHB5LPENCR OTGPHY1LPENC LL_AHB5_GRP1_DisableClockLowPower
- AHB5LPENCR OTGPHY2LPENC LL_AHB5_GRP1_DisableClockLowPower
- AHB5LPENCR OTG2LPENC LL_AHB5_GRP1_DisableClockLowPower
- AHB5LPENCR CACHEAXILPENC LL_AHB5_GRP1_DisableClockLowPower
- AHB5LPENCR NPULPENC LL_AHB5_GRP1_DisableClockLowPower

LL_APB1_GRP1_EnableClock

Function name

__STATIC_INLINE void LL_APB1_GRP1_EnableClock (uint32_t Periphs)

Function description

Enable APB1 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP1_PERIPH_I2C1
  - LL_APB1_GRP1_PERIPH_I2C2
  - LL_APB1_GRP1_PERIPH_I2C3
  - LL_APB1_GRP1_PERIPH_I3C1
  - LL_APB1_GRP1_PERIPH_I3C2
  - LL_APB1_GRP1_PERIPH_LPTIM1
  - LL_APB1_GRP1_PERIPH_SPDIFRX1
  - LL_APB1_GRP1_PERIPH_SPI2
  - LL_APB1_GRP1_PERIPH_SPI3
  - LL_APB1_GRP1_PERIPH_TIM2
  - LL_APB1_GRP1_PERIPH_TIM3
  - LL_APB1_GRP1_PERIPH_TIM4
  - LL_APB1_GRP1_PERIPH_TIM5
  - LL_APB1_GRP1_PERIPH_TIM6
  - LL_APB1_GRP1_PERIPH_TIM7
  - LL_APB1_GRP1_PERIPH_TIM10
  - LL_APB1_GRP1_PERIPH_TIM11
  - LL_APB1_GRP1_PERIPH_TIM12
  - LL_APB1_GRP1_PERIPH_TIM13
  - LL_APB1_GRP1_PERIPH_TIM14
  - LL_APB1_GRP1_PERIPH_USART2
  - LL_APB1_GRP1_PERIPH_USART3
  - LL_APB1_GRP1_PERIPH_UART4
  - LL_APB1_GRP1_PERIPH_UART5
  - LL_APB1_GRP1_PERIPH_UART7
  - LL_APB1_GRP1_PERIPH_UART8
  - LL_APB1_GRP1_PERIPH_WWDG

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1ENSR1 TIM2ENS LL_APB1_GRP1_EnableClock
- APB1ENSR1 TIM3ENS LL_APB1_GRP1_EnableClock
- APB1ENSR1 TIM4ENS LL_APB1_GRP1_EnableClock
- APB1ENSR1 TIM5ENS LL_APB1_GRP1_EnableClock
- APB1ENSR1 TIM6ENS LL_APB1_GRP1_EnableClock
- APB1ENSR1 TIM7ENS LL_APB1_GRP1_EnableClock
- APB1ENSR1 TIM12ENS LL_APB1_GRP1_EnableClock
- APB1ENSR1 TIM13ENS LL_APB1_GRP1_EnableClock
- APB1ENSR1 TIM14ENS LL_APB1_GRP1_EnableClock
- APB1ENSR1 LPTIM1ENS LL_APB1_GRP1_EnableClock
- APB1ENSR1 WWDGENS LL_APB1_GRP1_EnableClock
- APB1ENSR1 TIM10ENS LL_APB1_GRP1_EnableClock
- APB1ENSR1 TIM11ENS LL_APB1_GRP1_EnableClock
- APB1ENSR1 SPI2ENS LL_APB1_GRP1_EnableClock
- APB1ENSR1 SPI3ENS LL_APB1_GRP1_EnableClock
- APB1ENSR1 SPDIFRX1ENS LL_APB1_GRP1_EnableClock
- APB1ENSR1 USART2ENS LL_APB1_GRP1_EnableClock
- APB1ENSR1 USART3ENS LL_APB1_GRP1_EnableClock
- APB1ENSR1 UART4ENS LL_APB1_GRP1_EnableClock
- APB1ENSR1 UART5ENS LL_APB1_GRP1_EnableClock
- APB1ENSR1 I2C1ENS LL_APB1_GRP1_EnableClock
- APB1ENSR1 I2C2ENS LL_APB1_GRP1_EnableClock
- APB1ENSR1 I2C3ENS LL_APB1_GRP1_EnableClock
- APB1ENSR1 I3C1ENS LL_APB1_GRP1_EnableClock
- APB1ENSR1 I3C2ENS LL_APB1_GRP1_EnableClock
- APB1ENSR1 UART7ENS LL_APB1_GRP1_EnableClock
- APB1ENSR1 UART8ENS LL_APB1_GRP1_EnableClock

LL_APB1_GRP1_IsEnabledClock

Function name

__STATIC_INLINE uint32_t LL_APB1_GRP1_IsEnabledClock (uint32_t Periphs)

Function description

Check if APB1 peripheral clock is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP1_PERIPH_I2C1
  - LL_APB1_GRP1_PERIPH_I2C2
  - LL_APB1_GRP1_PERIPH_I2C3
  - LL_APB1_GRP1_PERIPH_I3C1
  - LL_APB1_GRP1_PERIPH_I3C2
  - LL_APB1_GRP1_PERIPH_LPTIM1
  - LL_APB1_GRP1_PERIPH_SPDIFRX1
  - LL_APB1_GRP1_PERIPH_SPI2
  - LL_APB1_GRP1_PERIPH_SPI3
  - LL_APB1_GRP1_PERIPH_TIM2
  - LL_APB1_GRP1_PERIPH_TIM3
  - LL_APB1_GRP1_PERIPH_TIM4
  - LL_APB1_GRP1_PERIPH_TIM5
  - LL_APB1_GRP1_PERIPH_TIM6
  - LL_APB1_GRP1_PERIPH_TIM7
  - LL_APB1_GRP1_PERIPH_TIM10
  - LL_APB1_GRP1_PERIPH_TIM11
  - LL_APB1_GRP1_PERIPH_TIM12
  - LL_APB1_GRP1_PERIPH_TIM13
  - LL_APB1_GRP1_PERIPH_TIM14
  - LL_APB1_GRP1_PERIPH_USART2
  - LL_APB1_GRP1_PERIPH_USART3
  - LL_APB1_GRP1_PERIPH_UART4
  - LL_APB1_GRP1_PERIPH_UART5
  - LL_APB1_GRP1_PERIPH_UART7
  - LL_APB1_GRP1_PERIPH_UART8
  - LL_APB1_GRP1_PERIPH_WWDG

Return values

- **uint32_t:**

Reference Manual to LL API cross reference:

- APB1ENR1 TIM2EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR1 TIM3EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR1 TIM4EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR1 TIM5EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR1 TIM6EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR1 TIM7EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR1 TIM12EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR1 TIM13EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR1 TIM14EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR1 LPTIM1EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR1 WWDGEN LL_APB1_GRP1_IsEnabledClock
- APB1ENR1 TIM10EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR1 TIM11EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR1 SPI2EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR1 SPI3EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR1 SPDIFRX1EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR1 USART2EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR1 USART3EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR1 UART4EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR1 UART5EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR1 I2C1EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR1 I2C2EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR1 I2C3EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR1 I3C1EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR1 I3C2EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR1 UART7EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR1 UART8EN LL_APB1_GRP1_IsEnabledClock

LL_APB1_GRP1_DisableClock

Function name

__STATIC_INLINE void LL_APB1_GRP1_DisableClock (uint32_t Periphs)

Function description

Disable APB1 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP1_PERIPH_I2C1
  - LL_APB1_GRP1_PERIPH_I2C2
  - LL_APB1_GRP1_PERIPH_I2C3
  - LL_APB1_GRP1_PERIPH_I3C1
  - LL_APB1_GRP1_PERIPH_I3C2
  - LL_APB1_GRP1_PERIPH_LPTIM1
  - LL_APB1_GRP1_PERIPH_SPDIFRX1
  - LL_APB1_GRP1_PERIPH_SPI2
  - LL_APB1_GRP1_PERIPH_SPI3
  - LL_APB1_GRP1_PERIPH_TIM2
  - LL_APB1_GRP1_PERIPH_TIM3
  - LL_APB1_GRP1_PERIPH_TIM4
  - LL_APB1_GRP1_PERIPH_TIM5
  - LL_APB1_GRP1_PERIPH_TIM6
  - LL_APB1_GRP1_PERIPH_TIM7
  - LL_APB1_GRP1_PERIPH_TIM10
  - LL_APB1_GRP1_PERIPH_TIM11
  - LL_APB1_GRP1_PERIPH_TIM12
  - LL_APB1_GRP1_PERIPH_TIM13
  - LL_APB1_GRP1_PERIPH_TIM14
  - LL_APB1_GRP1_PERIPH_USART2
  - LL_APB1_GRP1_PERIPH_USART3
  - LL_APB1_GRP1_PERIPH_UART4
  - LL_APB1_GRP1_PERIPH_UART5
  - LL_APB1_GRP1_PERIPH_UART7
  - LL_APB1_GRP1_PERIPH_UART8

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1ENCR1 TIM2ENC LL_APB1_GRP1_DisableClock
- APB1ENCR1 TIM3ENC LL_APB1_GRP1_DisableClock
- APB1ENCR1 TIM4ENC LL_APB1_GRP1_DisableClock
- APB1ENCR1 TIM5ENC LL_APB1_GRP1_DisableClock
- APB1ENCR1 TIM6ENC LL_APB1_GRP1_DisableClock
- APB1ENCR1 TIM7ENC LL_APB1_GRP1_DisableClock
- APB1ENCR1 TIM12ENC LL_APB1_GRP1_DisableClock
- APB1ENCR1 TIM13ENC LL_APB1_GRP1_DisableClock
- APB1ENCR1 TIM14ENC LL_APB1_GRP1_DisableClock
- APB1ENCR1 LPTIM1ENC LL_APB1_GRP1_DisableClock
- APB1ENCR1 TIM10ENC LL_APB1_GRP1_DisableClock
- APB1ENCR1 TIM11ENC LL_APB1_GRP1_DisableClock
- APB1ENCR1 SPI2ENC LL_APB1_GRP1_DisableClock
- APB1ENCR1 SPI3ENC LL_APB1_GRP1_DisableClock
- APB1ENCR1 SPDIFRX1ENC LL_APB1_GRP1_DisableClock
- APB1ENCR1 USART2ENC LL_APB1_GRP1_DisableClock
- APB1ENCR1 USART3ENC LL_APB1_GRP1_DisableClock
- APB1ENCR1 UART4ENC LL_APB1_GRP1_DisableClock
- APB1ENCR1 UART5ENC LL_APB1_GRP1_DisableClock
- APB1ENCR1 I2C1ENC LL_APB1_GRP1_DisableClock
- APB1ENCR1 I2C2ENC LL_APB1_GRP1_DisableClock
- APB1ENCR1 I2C3ENC LL_APB1_GRP1_DisableClock
- APB1ENCR1 I3C1ENC LL_APB1_GRP1_DisableClock
- APB1ENCR1 I3C2ENC LL_APB1_GRP1_DisableClock
- APB1ENCR1 UART7ENC LL_APB1_GRP1_DisableClock
- APB1ENCR1 UART8ENC LL_APB1_GRP1_DisableClock

LL_APB1_GRP1_ForceReset

Function name

__STATIC_INLINE void LL_APB1_GRP1_ForceReset (uint32_t Periphs)

Function description

Force APB1 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP1_PERIPH_I2C1
  - LL_APB1_GRP1_PERIPH_I2C2
  - LL_APB1_GRP1_PERIPH_I2C3
  - LL_APB1_GRP1_PERIPH_I3C1
  - LL_APB1_GRP1_PERIPH_I3C2
  - LL_APB1_GRP1_PERIPH_LPTIM1
  - LL_APB1_GRP1_PERIPH_SPDIFRX1
  - LL_APB1_GRP1_PERIPH_SPI2
  - LL_APB1_GRP1_PERIPH_SPI3
  - LL_APB1_GRP1_PERIPH_TIM2
  - LL_APB1_GRP1_PERIPH_TIM3
  - LL_APB1_GRP1_PERIPH_TIM4
  - LL_APB1_GRP1_PERIPH_TIM5
  - LL_APB1_GRP1_PERIPH_TIM6
  - LL_APB1_GRP1_PERIPH_TIM7
  - LL_APB1_GRP1_PERIPH_TIM10
  - LL_APB1_GRP1_PERIPH_TIM11
  - LL_APB1_GRP1_PERIPH_TIM12
  - LL_APB1_GRP1_PERIPH_TIM13
  - LL_APB1_GRP1_PERIPH_TIM14
  - LL_APB1_GRP1_PERIPH_USART2
  - LL_APB1_GRP1_PERIPH_USART3
  - LL_APB1_GRP1_PERIPH_UART4
  - LL_APB1_GRP1_PERIPH_UART5
  - LL_APB1_GRP1_PERIPH_UART7
  - LL_APB1_GRP1_PERIPH_UART8

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1RSTSR1 TIM2RSTS LL_APB1_GRP1_ForceReset
- APB1RSTSR1 TIM3RSTS LL_APB1_GRP1_ForceReset
- APB1RSTSR1 TIM4RSTS LL_APB1_GRP1_ForceReset
- APB1RSTSR1 TIM5RSTS LL_APB1_GRP1_ForceReset
- APB1RSTSR1 TIM6RSTS LL_APB1_GRP1_ForceReset
- APB1RSTSR1 TIM7RSTS LL_APB1_GRP1_ForceReset
- APB1RSTSR1 TIM12RSTS LL_APB1_GRP1_ForceReset
- APB1RSTSR1 TIM13RSTS LL_APB1_GRP1_ForceReset
- APB1RSTSR1 TIM14RSTS LL_APB1_GRP1_ForceReset
- APB1RSTSR1 LPTIM1RSTS LL_APB1_GRP1_ForceReset
- APB1RSTSR1 TIM10RSTS LL_APB1_GRP1_ForceReset
- APB1RSTSR1 TIM11RSTS LL_APB1_GRP1_ForceReset
- APB1RSTSR1 SPI2RSTS LL_APB1_GRP1_ForceReset
- APB1RSTSR1 SPI3RSTS LL_APB1_GRP1_ForceReset
- APB1RSTSR1 SPDIFRX1RSTS LL_APB1_GRP1_ForceReset
- APB1RSTSR1 USART2RSTS LL_APB1_GRP1_ForceReset
- APB1RSTSR1 USART3RSTS LL_APB1_GRP1_ForceReset
- APB1RSTSR1 UART4RSTS LL_APB1_GRP1_ForceReset
- APB1RSTSR1 UART5RSTS LL_APB1_GRP1_ForceReset
- APB1RSTSR1 I2C1RSTS LL_APB1_GRP1_ForceReset
- APB1RSTSR1 I2C2RSTS LL_APB1_GRP1_ForceReset
- APB1RSTSR1 I2C3RSTS LL_APB1_GRP1_ForceReset
- APB1RSTSR1 I3C1RSTS LL_APB1_GRP1_ForceReset
- APB1RSTSR1 I3C2RSTS LL_APB1_GRP1_ForceReset
- APB1RSTSR1 UART7RSTS LL_APB1_GRP1_ForceReset
- APB1RSTSR1 UART8RSTS LL_APB1_GRP1_ForceReset

LL_APB1_GRP1_ReleaseReset

Function name

__STATIC_INLINE void LL_APB1_GRP1_ReleaseReset (uint32_t Periphs)

Function description

Release APB1 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP1_PERIPH_I2C1
  - LL_APB1_GRP1_PERIPH_I2C2
  - LL_APB1_GRP1_PERIPH_I2C3
  - LL_APB1_GRP1_PERIPH_I3C1
  - LL_APB1_GRP1_PERIPH_I3C2
  - LL_APB1_GRP1_PERIPH_LPTIM1
  - LL_APB1_GRP1_PERIPH_SPDIFRX1
  - LL_APB1_GRP1_PERIPH_SPI2
  - LL_APB1_GRP1_PERIPH_SPI3
  - LL_APB1_GRP1_PERIPH_TIM2
  - LL_APB1_GRP1_PERIPH_TIM3
  - LL_APB1_GRP1_PERIPH_TIM4
  - LL_APB1_GRP1_PERIPH_TIM5
  - LL_APB1_GRP1_PERIPH_TIM6
  - LL_APB1_GRP1_PERIPH_TIM7
  - LL_APB1_GRP1_PERIPH_TIM10
  - LL_APB1_GRP1_PERIPH_TIM11
  - LL_APB1_GRP1_PERIPH_TIM12
  - LL_APB1_GRP1_PERIPH_TIM13
  - LL_APB1_GRP1_PERIPH_TIM14
  - LL_APB1_GRP1_PERIPH_USART2
  - LL_APB1_GRP1_PERIPH_USART3
  - LL_APB1_GRP1_PERIPH_UART4
  - LL_APB1_GRP1_PERIPH_UART5
  - LL_APB1_GRP1_PERIPH_UART7
  - LL_APB1_GRP1_PERIPH_UART8

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1RSTCR1 TIM2RSTC LL_APB1_GRP1_ReleaseReset
- APB1RSTCR1 TIM3RSTC LL_APB1_GRP1_ReleaseReset
- APB1RSTCR1 TIM4RSTC LL_APB1_GRP1_ReleaseReset
- APB1RSTCR1 TIM5RSTC LL_APB1_GRP1_ReleaseReset
- APB1RSTCR1 TIM6RSTC LL_APB1_GRP1_ReleaseReset
- APB1RSTCR1 TIM7RSTC LL_APB1_GRP1_ReleaseReset
- APB1RSTCR1 TIM12RSTC LL_APB1_GRP1_ReleaseReset
- APB1RSTCR1 TIM13RSTC LL_APB1_GRP1_ReleaseReset
- APB1RSTCR1 TIM14RSTC LL_APB1_GRP1_ReleaseReset
- APB1RSTCR1 LPTIM1RSTC LL_APB1_GRP1_ReleaseReset
- APB1RSTCR1 TIM10RSTC LL_APB1_GRP1_ReleaseReset
- APB1RSTCR1 TIM11RSTC LL_APB1_GRP1_ReleaseReset
- APB1RSTCR1 SPI2RSTC LL_APB1_GRP1_ReleaseReset
- APB1RSTCR1 SPI3RSTC LL_APB1_GRP1_ReleaseReset
- APB1RSTCR1 SPDIFRX1RSTC LL_APB1_GRP1_ReleaseReset
- APB1RSTCR1 USART2RSTC LL_APB1_GRP1_ReleaseReset
- APB1RSTCR1 USART3RSTC LL_APB1_GRP1_ReleaseReset
- APB1RSTCR1 UART4RSTC LL_APB1_GRP1_ReleaseReset
- APB1RSTCR1 UART5RSTC LL_APB1_GRP1_ReleaseReset
- APB1RSTCR1 I2C1RSTC LL_APB1_GRP1_ReleaseReset
- APB1RSTCR1 I2C2RSTC LL_APB1_GRP1_ReleaseReset
- APB1RSTCR1 I2C3RSTC LL_APB1_GRP1_ReleaseReset
- APB1RSTCR1 I3C1RSTC LL_APB1_GRP1_ReleaseReset
- APB1RSTCR1 I3C2RSTC LL_APB1_GRP1_ReleaseReset
- APB1RSTCR1 UART7RSTC LL_APB1_GRP1_ReleaseReset
- APB1RSTCR1 UART8RSTC LL_APB1_GRP1_ReleaseReset

LL_APB1_GRP1_EnableClockLowPower

Function name

__STATIC_INLINE void LL_APB1_GRP1_EnableClockLowPower (uint32_t Periphs)

Function description

Enable APB1 peripherals clock during Low Power mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP1_PERIPH_I2C1
  - LL_APB1_GRP1_PERIPH_I2C2
  - LL_APB1_GRP1_PERIPH_I2C3
  - LL_APB1_GRP1_PERIPH_I3C1
  - LL_APB1_GRP1_PERIPH_I3C2
  - LL_APB1_GRP1_PERIPH_LPTIM1
  - LL_APB1_GRP1_PERIPH_SPDIFRX1
  - LL_APB1_GRP1_PERIPH_SPI2
  - LL_APB1_GRP1_PERIPH_SPI3
  - LL_APB1_GRP1_PERIPH_TIM2
  - LL_APB1_GRP1_PERIPH_TIM3
  - LL_APB1_GRP1_PERIPH_TIM4
  - LL_APB1_GRP1_PERIPH_TIM5
  - LL_APB1_GRP1_PERIPH_TIM6
  - LL_APB1_GRP1_PERIPH_TIM7
  - LL_APB1_GRP1_PERIPH_TIM10
  - LL_APB1_GRP1_PERIPH_TIM11
  - LL_APB1_GRP1_PERIPH_TIM12
  - LL_APB1_GRP1_PERIPH_TIM13
  - LL_APB1_GRP1_PERIPH_TIM14
  - LL_APB1_GRP1_PERIPH_USART2
  - LL_APB1_GRP1_PERIPH_USART3
  - LL_APB1_GRP1_PERIPH_UART4
  - LL_APB1_GRP1_PERIPH_UART5
  - LL_APB1_GRP1_PERIPH_UART7
  - LL_APB1_GRP1_PERIPH_UART8
  - LL_APB1_GRP1_PERIPH_WWDG

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1LPENSR1 TIM2LPENS LL_APB1_GRP1_EnableClockLowPower
- APB1LPENSR1 TIM3LPENS LL_APB1_GRP1_EnableClockLowPower
- APB1LPENSR1 TIM4LPENS LL_APB1_GRP1_EnableClockLowPower
- APB1LPENSR1 TIM5LPENS LL_APB1_GRP1_EnableClockLowPower
- APB1LPENSR1 TIM6LPENS LL_APB1_GRP1_EnableClockLowPower
- APB1LPENSR1 TIM7LPENS LL_APB1_GRP1_EnableClockLowPower
- APB1LPENSR1 TIM12LPENS LL_APB1_GRP1_EnableClockLowPower
- APB1LPENSR1 TIM13LPENS LL_APB1_GRP1_EnableClockLowPower
- APB1LPENSR1 TIM14LPENS LL_APB1_GRP1_EnableClockLowPower
- APB1LPENSR1 LPTIM1LPENS LL_APB1_GRP1_EnableClockLowPower
- APB1LPENSR1 WWDGLPENS LL_APB1_GRP1_EnableClockLowPower
- APB1LPENSR1 TIM10LPENS LL_APB1_GRP1_EnableClockLowPower
- APB1LPENSR1 TIM11LPENS LL_APB1_GRP1_EnableClockLowPower
- APB1LPENSR1 SPI2LPENS LL_APB1_GRP1_EnableClockLowPower
- APB1LPENSR1 SPI3LPENS LL_APB1_GRP1_EnableClockLowPower
- APB1LPENSR1 SPDIFRX1LPENS LL_APB1_GRP1_EnableClockLowPower
- APB1LPENSR1 USART2LPENS LL_APB1_GRP1_EnableClockLowPower
- APB1LPENSR1 USART3LPENS LL_APB1_GRP1_EnableClockLowPower
- APB1LPENSR1 UART4LPENS LL_APB1_GRP1_EnableClockLowPower
- APB1LPENSR1 UART5LPENS LL_APB1_GRP1_EnableClockLowPower
- APB1LPENSR1 I2C1LPENS LL_APB1_GRP1_EnableClockLowPower
- APB1LPENSR1 I2C2LPENS LL_APB1_GRP1_EnableClockLowPower
- APB1LPENSR1 I2C3LPENS LL_APB1_GRP1_EnableClockLowPower
- APB1LPENSR1 I3C1LPENS LL_APB1_GRP1_EnableClockLowPower
- APB1LPENSR1 I3C2LPENS LL_APB1_GRP1_EnableClockLowPower
- APB1LPENSR1 UART7LPENS LL_APB1_GRP1_EnableClockLowPower
- APB1LPENSR1 UART8LPENS LL_APB1_GRP1_EnableClockLowPower

LL_APB1_GRP1_IsEnabledClockLowPower

Function name

__STATIC_INLINE uint32_t LL_APB1_GRP1_IsEnabledClockLowPower (uint32_t Periphs)

Function description

Check if APB1 peripheral clock during Low Power mode is enabled or not .

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP1_PERIPH_I2C1
  - LL_APB1_GRP1_PERIPH_I2C2
  - LL_APB1_GRP1_PERIPH_I2C3
  - LL_APB1_GRP1_PERIPH_I3C1
  - LL_APB1_GRP1_PERIPH_I3C2
  - LL_APB1_GRP1_PERIPH_LPTIM1
  - LL_APB1_GRP1_PERIPH_SPDIFRX1
  - LL_APB1_GRP1_PERIPH_SPI2
  - LL_APB1_GRP1_PERIPH_SPI3
  - LL_APB1_GRP1_PERIPH_TIM2
  - LL_APB1_GRP1_PERIPH_TIM3
  - LL_APB1_GRP1_PERIPH_TIM4
  - LL_APB1_GRP1_PERIPH_TIM5
  - LL_APB1_GRP1_PERIPH_TIM6
  - LL_APB1_GRP1_PERIPH_TIM7
  - LL_APB1_GRP1_PERIPH_TIM10
  - LL_APB1_GRP1_PERIPH_TIM11
  - LL_APB1_GRP1_PERIPH_TIM12
  - LL_APB1_GRP1_PERIPH_TIM13
  - LL_APB1_GRP1_PERIPH_TIM14
  - LL_APB1_GRP1_PERIPH_USART2
  - LL_APB1_GRP1_PERIPH_USART3
  - LL_APB1_GRP1_PERIPH_UART4
  - LL_APB1_GRP1_PERIPH_UART5
  - LL_APB1_GRP1_PERIPH_UART7
  - LL_APB1_GRP1_PERIPH_UART8
  - LL_APB1_GRP1_PERIPH_WWDG

Return values

- **uint32_t:**

Reference Manual to LL API cross reference:

- APB1LPENR1 TIM2LPEN LL_APB1_GRP1_IsEnabledClockLowPower
- APB1LPENR1 TIM3LPEN LL_APB1_GRP1_IsEnabledClockLowPower
- APB1LPENR1 TIM4LPEN LL_APB1_GRP1_IsEnabledClockLowPower
- APB1LPENR1 TIM5LPEN LL_APB1_GRP1_IsEnabledClockLowPower
- APB1LPENR1 TIM6LPEN LL_APB1_GRP1_IsEnabledClockLowPower
- APB1LPENR1 TIM7LPEN LL_APB1_GRP1_IsEnabledClockLowPower
- APB1LPENR1 TIM12LPEN LL_APB1_GRP1_IsEnabledClockLowPower
- APB1LPENR1 TIM13LPEN LL_APB1_GRP1_IsEnabledClockLowPower
- APB1LPENR1 TIM14LPEN LL_APB1_GRP1_IsEnabledClockLowPower
- APB1LPENR1 LPTIM1LPEN LL_APB1_GRP1_IsEnabledClockLowPower
- APB1LPENR1 WWDGLPEN LL_APB1_GRP1_IsEnabledClockLowPower
- APB1LPENR1 TIM10LPEN LL_APB1_GRP1_IsEnabledClockLowPower
- APB1LPENR1 TIM11LPEN LL_APB1_GRP1_IsEnabledClockLowPower
- APB1LPENR1 SPI2LPEN LL_APB1_GRP1_IsEnabledClockLowPower
- APB1LPENR1 SPI3LPEN LL_APB1_GRP1_IsEnabledClockLowPower
- APB1LPENR1 SPDIFRX1LPEN LL_APB1_GRP1_IsEnabledClockLowPower
- APB1LPENR1 USART2LPEN LL_APB1_GRP1_IsEnabledClockLowPower
- APB1LPENR1 USART3LPEN LL_APB1_GRP1_IsEnabledClockLowPower
- APB1LPENR1 UART4LPEN LL_APB1_GRP1_IsEnabledClockLowPower
- APB1LPENR1 UART5LPEN LL_APB1_GRP1_IsEnabledClockLowPower
- APB1LPENR1 I2C1LPEN LL_APB1_GRP1_IsEnabledClockLowPower
- APB1LPENR1 I2C2LPEN LL_APB1_GRP1_IsEnabledClockLowPower
- APB1LPENR1 I2C3LPEN LL_APB1_GRP1_IsEnabledClockLowPower
- APB1LPENR1 I3C1LPEN LL_APB1_GRP1_IsEnabledClockLowPower
- APB1LPENR1 I3C2LPEN LL_APB1_GRP1_IsEnabledClockLowPower
- APB1LPENR1 UART7LPEN LL_APB1_GRP1_IsEnabledClockLowPower
- APB1LPENR1 UART8LPEN LL_APB1_GRP1_IsEnabledClockLowPower

LL_APB1_GRP1_DisableClockLowPower

Function name

__STATIC_INLINE void LL_APB1_GRP1_DisableClockLowPower (uint32_t Periphs)

Function description

Disable APB1 peripherals clock during Low Power mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP1_PERIPH_I2C1
  - LL_APB1_GRP1_PERIPH_I2C2
  - LL_APB1_GRP1_PERIPH_I2C3
  - LL_APB1_GRP1_PERIPH_I3C1
  - LL_APB1_GRP1_PERIPH_I3C2
  - LL_APB1_GRP1_PERIPH_LPTIM1
  - LL_APB1_GRP1_PERIPH_SPDIFRX1
  - LL_APB1_GRP1_PERIPH_SPI2
  - LL_APB1_GRP1_PERIPH_SPI3
  - LL_APB1_GRP1_PERIPH_TIM2
  - LL_APB1_GRP1_PERIPH_TIM3
  - LL_APB1_GRP1_PERIPH_TIM4
  - LL_APB1_GRP1_PERIPH_TIM5
  - LL_APB1_GRP1_PERIPH_TIM6
  - LL_APB1_GRP1_PERIPH_TIM7
  - LL_APB1_GRP1_PERIPH_TIM10
  - LL_APB1_GRP1_PERIPH_TIM11
  - LL_APB1_GRP1_PERIPH_TIM12
  - LL_APB1_GRP1_PERIPH_TIM13
  - LL_APB1_GRP1_PERIPH_TIM14
  - LL_APB1_GRP1_PERIPH_USART2
  - LL_APB1_GRP1_PERIPH_USART3
  - LL_APB1_GRP1_PERIPH_UART4
  - LL_APB1_GRP1_PERIPH_UART5
  - LL_APB1_GRP1_PERIPH_UART7
  - LL_APB1_GRP1_PERIPH_UART8
  - LL_APB1_GRP1_PERIPH_WWDG

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1LPENCR1 TIM2LPENC LL_APB1_GRP1_DisableClockLowPower
- APB1LPENCR1 TIM3LPENC LL_APB1_GRP1_DisableClockLowPower
- APB1LPENCR1 TIM4LPENC LL_APB1_GRP1_DisableClockLowPower
- APB1LPENCR1 TIM5LPENC LL_APB1_GRP1_DisableClockLowPower
- APB1LPENCR1 TIM6LPENC LL_APB1_GRP1_DisableClockLowPower
- APB1LPENCR1 TIM7LPENC LL_APB1_GRP1_DisableClockLowPower
- APB1LPENCR1 TIM12LPENC LL_APB1_GRP1_DisableClockLowPower
- APB1LPENCR1 TIM13LPENC LL_APB1_GRP1_DisableClockLowPower
- APB1LPENCR1 TIM14LPENC LL_APB1_GRP1_DisableClockLowPower
- APB1LPENCR1 LPTIM1LPENC LL_APB1_GRP1_DisableClockLowPower
- APB1LPENCR1 WWDGLPENC LL_APB1_GRP1_DisableClockLowPower
- APB1LPENCR1 TIM10LPENC LL_APB1_GRP1_DisableClockLowPower
- APB1LPENCR1 TIM11LPENC LL_APB1_GRP1_DisableClockLowPower
- APB1LPENCR1 SPI2LPENC LL_APB1_GRP1_DisableClockLowPower
- APB1LPENCR1 SPI3LPENC LL_APB1_GRP1_DisableClockLowPower
- APB1LPENCR1 SPDIFRX1LPENC LL_APB1_GRP1_DisableClockLowPower
- APB1LPENCR1 USART2LPENC LL_APB1_GRP1_DisableClockLowPower
- APB1LPENCR1 USART3LPENC LL_APB1_GRP1_DisableClockLowPower
- APB1LPENCR1 UART4LPENC LL_APB1_GRP1_DisableClockLowPower
- APB1LPENCR1 UART5LPENC LL_APB1_GRP1_DisableClockLowPower
- APB1LPENCR1 I2C1LPENC LL_APB1_GRP1_DisableClockLowPower
- APB1LPENCR1 I2C2LPENC LL_APB1_GRP1_DisableClockLowPower
- APB1LPENCR1 I2C3LPENC LL_APB1_GRP1_DisableClockLowPower
- APB1LPENCR1 I3C1LPENC LL_APB1_GRP1_DisableClockLowPower
- APB1LPENCR1 I3C2LPENC LL_APB1_GRP1_DisableClockLowPower
- APB1LPENCR1 UART7LPENC LL_APB1_GRP1_DisableClockLowPower
- APB1LPENCR1 UART8LPENC LL_APB1_GRP1_DisableClockLowPower

LL_APB1_GRP2_EnableClock

Function name

__STATIC_INLINE void LL_APB1_GRP2_EnableClock (uint32_t Periphs)

Function description

Enable APB1 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP2_PERIPH_FDCAN
  - LL_APB1_GRP2_PERIPH_MDIOS
  - LL_APB1_GRP2_PERIPH_UCPD1

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1ENSR2 MDIOSENS LL_APB1_GRP2_EnableClock
- APB1ENSR2 FDCANENS LL_APB1_GRP2_EnableClock
- APB1ENSR2 UCPD1ENS LL_APB1_GRP2_EnableClock

LL_APB1_GRP2_IsEnabledClock

Function name

__STATIC_INLINE uint32_t LL_APB1_GRP2_IsEnabledClock (uint32_t Periphs)

Function description

Check if APB1 peripheral clock is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP2_PERIPH_FDCAN
  - LL_APB1_GRP2_PERIPH_MDIOS
  - LL_APB1_GRP2_PERIPH_UCPD1

Return values

- **uint32_t:**

Reference Manual to LL API cross reference:

- APB1ENR2 MDIOSEN LL_APB1_GRP2_IsEnabledClock
- APB1ENR2 FDCANEN LL_APB1_GRP2_IsEnabledClock
- APB1ENR2 UCPD1EN LL_APB1_GRP2_IsEnabledClock

LL_APB1_GRP2_DisableClock

Function name

__STATIC_INLINE void LL_APB1_GRP2_DisableClock (uint32_t Periphs)

Function description

Disable APB1 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP2_PERIPH_FDCAN
  - LL_APB1_GRP2_PERIPH_MDIOS
  - LL_APB1_GRP2_PERIPH_UCPD1

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1ENCR2 MDIOSENC LL_APB1_GRP2_DisableClock
- APB1ENCR2 FDCANENC LL_APB1_GRP2_DisableClock
- APB1ENCR2 UCPD1ENC LL_APB1_GRP2_DisableClock

LL_APB1_GRP2_ForceReset

Function name

__STATIC_INLINE void LL_APB1_GRP2_ForceReset (uint32_t Periphs)

Function description

Force APB1 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP2_PERIPH_FDCAN
  - LL_APB1_GRP2_PERIPH_MDIOS
  - LL_APB1_GRP2_PERIPH_UCPD1

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1RSTSR2 MDIOSRSTS LL_APB1_GRP2_ForceReset
- APB1RSTSR2 FDCANRSTS LL_APB1_GRP2_ForceReset
- APB1RSTSR2 UCPD1RSTS LL_APB1_GRP2_ForceReset

LL_APB1_GRP2_ReleaseReset

Function name

__STATIC_INLINE void LL_APB1_GRP2_ReleaseReset (uint32_t Periphs)

Function description

Release APB1 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP2_PERIPH_FDCAN
  - LL_APB1_GRP2_PERIPH_MDIOS
  - LL_APB1_GRP2_PERIPH_UCPD1

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1RSTCR2 MDIOSRSTC LL_APB1_GRP2_ReleaseReset
- APB1RSTCR2 FDCANRSTC LL_APB1_GRP2_ReleaseReset
- APB1RSTCR2 UCPD1RSTC LL_APB1_GRP2_ReleaseReset

LL_APB1_GRP2_EnableClockLowPower

Function name

__STATIC_INLINE void LL_APB1_GRP2_EnableClockLowPower (uint32_t Periphs)

Function description

Enable APB1 peripherals clock during Low Power mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP2_PERIPH_FDCAN
  - LL_APB1_GRP2_PERIPH_MDIOS
  - LL_APB1_GRP2_PERIPH_UCPD1

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1LPENSR2 MDIOSLPENS LL_APB1_GRP2_EnableClockLowPower
- APB1LPENSR2 FDCANLPENS LL_APB1_GRP2_EnableClockLowPower
- APB1LPENSR2 UCPD1LPENS LL_APB1_GRP2_EnableClockLowPower

LL_APB1_GRP2_IsEnabledClockLowPower

Function name

__STATIC_INLINE uint32_t LL_APB1_GRP2_IsEnabledClockLowPower (uint32_t Periphs)

Function description

Check if APB1 peripheral clock during Low Power mode is enabled or not .

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP2_PERIPH_FDCAN
  - LL_APB1_GRP2_PERIPH_MDIOS
  - LL_APB1_GRP2_PERIPH_UCPD1

Return values

- **uint32_t:**

Reference Manual to LL API cross reference:

- APB1LPENR2 MDIOSLPEN LL_APB1_GRP2_IsEnabledClockLowPower
- APB1LPENR2 FDCANLPEN LL_APB1_GRP2_IsEnabledClockLowPower
- APB1LPENR2 UCPD1LPEN LL_APB1_GRP2_IsEnabledClockLowPower

LL_APB1_GRP2_DisableClockLowPower

Function name

__STATIC_INLINE void LL_APB1_GRP2_DisableClockLowPower (uint32_t Periphs)

Function description

Disable APB1 peripherals clock during Low Power mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP2_PERIPH_FDCAN
  - LL_APB1_GRP2_PERIPH_MDIOS
  - LL_APB1_GRP2_PERIPH_UCPD1

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1LPENCR2 MDIOSLPENC LL_APB1_GRP2_DisableClockLowPower
- APB1LPENCR2 FDCANLPENC LL_APB1_GRP2_DisableClockLowPower
- APB1LPENCR2 UCPD1LPENC LL_APB1_GRP2_DisableClockLowPower

LL_APB2_GRP1_EnableClock

Function name

__STATIC_INLINE void LL_APB2_GRP1_EnableClock (uint32_t Periphs)

Function description

Enable APB2 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB2_GRP1_PERIPH_SAI1
  - LL_APB2_GRP1_PERIPH_SAI2
  - LL_APB2_GRP1_PERIPH_SPI1
  - LL_APB2_GRP1_PERIPH_SPI4
  - LL_APB2_GRP1_PERIPH_SPI5
  - LL_APB2_GRP1_PERIPH_TIM1
  - LL_APB2_GRP1_PERIPH_TIM8
  - LL_APB2_GRP1_PERIPH_TIM9
  - LL_APB2_GRP1_PERIPH_TIM15
  - LL_APB2_GRP1_PERIPH_TIM16
  - LL_APB2_GRP1_PERIPH_TIM17
  - LL_APB2_GRP1_PERIPH_TIM18
  - LL_APB2_GRP1_PERIPH_USART1
  - LL_APB2_GRP1_PERIPH_USART6
  - LL_APB2_GRP1_PERIPH_UART9
  - LL_APB2_GRP1_PERIPH_USART10

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB2ENSR TIM1ENS LL_APB2_GRP1_EnableClock
- APB2ENSR TIM8ENS LL_APB2_GRP1_EnableClock
- APB2ENSR USART1ENS LL_APB2_GRP1_EnableClock
- APB2ENSR USART6ENS LL_APB2_GRP1_EnableClock
- APB2ENSR UART9ENS LL_APB2_GRP1_EnableClock
- APB2ENSR USART10ENS LL_APB2_GRP1_EnableClock
- APB2ENSR SPI1ENS LL_APB2_GRP1_EnableClock
- APB2ENSR SPI4ENS LL_APB2_GRP1_EnableClock
- APB2ENSR TIM18ENS LL_APB2_GRP1_EnableClock
- APB2ENSR TIM15ENS LL_APB2_GRP1_EnableClock
- APB2ENSR TIM16ENS LL_APB2_GRP1_EnableClock
- APB2ENSR TIM17ENS LL_APB2_GRP1_EnableClock
- APB2ENSR TIM9ENS LL_APB2_GRP1_EnableClock
- APB2ENSR SPI5ENS LL_APB2_GRP1_EnableClock
- APB2ENSR SAI1ENS LL_APB2_GRP1_EnableClock
- APB2ENSR SAI2ENS LL_APB2_GRP1_EnableClock

LL_APB2_GRP1_IsEnabledClock

Function name

__STATIC_INLINE uint32_t LL_APB2_GRP1_IsEnabledClock (uint32_t Periphs)

Function description

Check if APB2 peripheral clock is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB2_GRP1_PERIPH_SAI1
  - LL_APB2_GRP1_PERIPH_SAI2
  - LL_APB2_GRP1_PERIPH_SPI1
  - LL_APB2_GRP1_PERIPH_SPI4
  - LL_APB2_GRP1_PERIPH_SPI5
  - LL_APB2_GRP1_PERIPH_TIM1
  - LL_APB2_GRP1_PERIPH_TIM8
  - LL_APB2_GRP1_PERIPH_TIM9
  - LL_APB2_GRP1_PERIPH_TIM15
  - LL_APB2_GRP1_PERIPH_TIM16
  - LL_APB2_GRP1_PERIPH_TIM17
  - LL_APB2_GRP1_PERIPH_TIM18
  - LL_APB2_GRP1_PERIPH_USART1
  - LL_APB2_GRP1_PERIPH_USART6
  - LL_APB2_GRP1_PERIPH_UART9
  - LL_APB2_GRP1_PERIPH_USART10

Return values

- **uint32_t:**

Reference Manual to LL API cross reference:

- APB2ENR TIM1EN LL_APB2_GRP1_IsEnabledClock
- APB2ENR TIM8EN LL_APB2_GRP1_IsEnabledClock
- APB2ENR USART1EN LL_APB2_GRP1_IsEnabledClock
- APB2ENR USART6EN LL_APB2_GRP1_IsEnabledClock
- APB2ENR UART9EN LL_APB2_GRP1_IsEnabledClock
- APB2ENR USART10EN LL_APB2_GRP1_IsEnabledClock
- APB2ENR SPI1EN LL_APB2_GRP1_IsEnabledClock
- APB2ENR SPI4EN LL_APB2_GRP1_IsEnabledClock
- APB2ENR TIM18EN LL_APB2_GRP1_IsEnabledClock
- APB2ENR TIM15EN LL_APB2_GRP1_IsEnabledClock
- APB2ENR TIM16EN LL_APB2_GRP1_IsEnabledClock
- APB2ENR TIM17EN LL_APB2_GRP1_IsEnabledClock
- APB2ENR TIM9EN LL_APB2_GRP1_IsEnabledClock
- APB2ENR SPI5EN LL_APB2_GRP1_IsEnabledClock
- APB2ENR SAI1EN LL_APB2_GRP1_IsEnabledClock
- APB2ENR SAI2EN LL_APB2_GRP1_IsEnabledClock

LL_APB2_GRP1_DisableClock

Function name

__STATIC_INLINE void LL_APB2_GRP1_DisableClock (uint32_t Periphs)

Function description

Disable APB2 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB2_GRP1_PERIPH_SAI1
  - LL_APB2_GRP1_PERIPH_SAI2
  - LL_APB2_GRP1_PERIPH_SPI1
  - LL_APB2_GRP1_PERIPH_SPI4
  - LL_APB2_GRP1_PERIPH_SPI5
  - LL_APB2_GRP1_PERIPH_TIM1
  - LL_APB2_GRP1_PERIPH_TIM8
  - LL_APB2_GRP1_PERIPH_TIM9
  - LL_APB2_GRP1_PERIPH_TIM15
  - LL_APB2_GRP1_PERIPH_TIM16
  - LL_APB2_GRP1_PERIPH_TIM17
  - LL_APB2_GRP1_PERIPH_TIM18
  - LL_APB2_GRP1_PERIPH_USART1
  - LL_APB2_GRP1_PERIPH_USART6
  - LL_APB2_GRP1_PERIPH_UART9
  - LL_APB2_GRP1_PERIPH_USART10

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB2ENCR TIM1ENC LL_APB2_GRP1_DisableClock
- APB2ENCR TIM8ENC LL_APB2_GRP1_DisableClock
- APB2ENCR USART1ENC LL_APB2_GRP1_DisableClock
- APB2ENCR USART6ENC LL_APB2_GRP1_DisableClock
- APB2ENCR UART9ENC LL_APB2_GRP1_DisableClock
- APB2ENCR USART10ENC LL_APB2_GRP1_DisableClock
- APB2ENCR SPI1ENC LL_APB2_GRP1_DisableClock
- APB2ENCR SPI4ENC LL_APB2_GRP1_DisableClock
- APB2ENCR TIM18ENC LL_APB2_GRP1_DisableClock
- APB2ENCR TIM15ENC LL_APB2_GRP1_DisableClock
- APB2ENCR TIM16ENC LL_APB2_GRP1_DisableClock
- APB2ENCR TIM17ENC LL_APB2_GRP1_DisableClock
- APB2ENCR TIM9ENC LL_APB2_GRP1_DisableClock
- APB2ENCR SPI5ENC LL_APB2_GRP1_DisableClock
- APB2ENCR SAI1ENC LL_APB2_GRP1_DisableClock
- APB2ENCR SAI2ENC LL_APB2_GRP1_DisableClock

LL_APB2_GRP1_ForceReset

Function name

__STATIC_INLINE void LL_APB2_GRP1_ForceReset (uint32_t Periphs)

Function description

Force APB2 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB2_GRP1_PERIPH_SAI1
  - LL_APB2_GRP1_PERIPH_SAI2
  - LL_APB2_GRP1_PERIPH_SPI1
  - LL_APB2_GRP1_PERIPH_SPI4
  - LL_APB2_GRP1_PERIPH_SPI5
  - LL_APB2_GRP1_PERIPH_TIM1
  - LL_APB2_GRP1_PERIPH_TIM8
  - LL_APB2_GRP1_PERIPH_TIM9
  - LL_APB2_GRP1_PERIPH_TIM15
  - LL_APB2_GRP1_PERIPH_TIM16
  - LL_APB2_GRP1_PERIPH_TIM17
  - LL_APB2_GRP1_PERIPH_TIM18
  - LL_APB2_GRP1_PERIPH_USART1
  - LL_APB2_GRP1_PERIPH_USART6
  - LL_APB2_GRP1_PERIPH_UART9
  - LL_APB2_GRP1_PERIPH_USART10

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB2RSTSR TIM1RSTS LL_APB2_GRP1_ForceReset
- APB2RSTSR TIM8RSTS LL_APB2_GRP1_ForceReset
- APB2RSTSR USART1RSTS LL_APB2_GRP1_ForceReset
- APB2RSTSR USART6RSTS LL_APB2_GRP1_ForceReset
- APB2RSTSR UART9RSTS LL_APB2_GRP1_ForceReset
- APB2RSTSR USART10RSTS LL_APB2_GRP1_ForceReset
- APB2RSTSR SPI1RSTS LL_APB2_GRP1_ForceReset
- APB2RSTSR SPI4RSTS LL_APB2_GRP1_ForceReset
- APB2RSTSR TIM18RSTS LL_APB2_GRP1_ForceReset
- APB2RSTSR TIM15RSTS LL_APB2_GRP1_ForceReset
- APB2RSTSR TIM16RSTS LL_APB2_GRP1_ForceReset
- APB2RSTSR TIM17RSTS LL_APB2_GRP1_ForceReset
- APB2RSTSR TIM9RSTS LL_APB2_GRP1_ForceReset
- APB2RSTSR SPI5RSTS LL_APB2_GRP1_ForceReset
- APB2RSTSR SAI1RSTS LL_APB2_GRP1_ForceReset
- APB2RSTSR SAI2RSTS LL_APB2_GRP1_ForceReset

LL_APB2_GRP1_ReleaseReset

Function name

__STATIC_INLINE void LL_APB2_GRP1_ReleaseReset (uint32_t Periphs)

Function description

Release APB2 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB2_GRP1_PERIPH_SAI1
  - LL_APB2_GRP1_PERIPH_SAI2
  - LL_APB2_GRP1_PERIPH_SPI1
  - LL_APB2_GRP1_PERIPH_SPI4
  - LL_APB2_GRP1_PERIPH_SPI5
  - LL_APB2_GRP1_PERIPH_TIM1
  - LL_APB2_GRP1_PERIPH_TIM8
  - LL_APB2_GRP1_PERIPH_TIM9
  - LL_APB2_GRP1_PERIPH_TIM15
  - LL_APB2_GRP1_PERIPH_TIM16
  - LL_APB2_GRP1_PERIPH_TIM17
  - LL_APB2_GRP1_PERIPH_TIM18
  - LL_APB2_GRP1_PERIPH_USART1
  - LL_APB2_GRP1_PERIPH_USART6
  - LL_APB2_GRP1_PERIPH_UART9
  - LL_APB2_GRP1_PERIPH_USART10

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB2RSTCR TIM1RSTC LL_APB2_GRP1_ReleaseReset
- APB2RSTCR TIM8RSTC LL_APB2_GRP1_ReleaseReset
- APB2RSTCR USART1RSTC LL_APB2_GRP1_ReleaseReset
- APB2RSTCR USART6RSTC LL_APB2_GRP1_ReleaseReset
- APB2RSTCR UART9RSTC LL_APB2_GRP1_ReleaseReset
- APB2RSTCR USART10RSTC LL_APB2_GRP1_ReleaseReset
- APB2RSTCR SPI1RSTC LL_APB2_GRP1_ReleaseReset
- APB2RSTCR SPI4RSTC LL_APB2_GRP1_ReleaseReset
- APB2RSTCR TIM18RSTC LL_APB2_GRP1_ReleaseReset
- APB2RSTCR TIM15RSTC LL_APB2_GRP1_ReleaseReset
- APB2RSTCR TIM16RSTC LL_APB2_GRP1_ReleaseReset
- APB2RSTCR TIM17RSTC LL_APB2_GRP1_ReleaseReset
- APB2RSTCR TIM9RSTC LL_APB2_GRP1_ReleaseReset
- APB2RSTCR SPI5RSTC LL_APB2_GRP1_ReleaseReset
- APB2RSTCR SAI1RSTC LL_APB2_GRP1_ReleaseReset
- APB2RSTCR SAI2RSTC LL_APB2_GRP1_ReleaseReset

LL_APB2_GRP1_EnableClockLowPower

Function name

__STATIC_INLINE void LL_APB2_GRP1_EnableClockLowPower (uint32_t Periphs)

Function description

Enable APB2 peripherals clock during Low Power mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB2_GRP1_PERIPH_SAI1
  - LL_APB2_GRP1_PERIPH_SAI2
  - LL_APB2_GRP1_PERIPH_SPI1
  - LL_APB2_GRP1_PERIPH_SPI4
  - LL_APB2_GRP1_PERIPH_SPI5
  - LL_APB2_GRP1_PERIPH_TIM1
  - LL_APB2_GRP1_PERIPH_TIM8
  - LL_APB2_GRP1_PERIPH_TIM9
  - LL_APB2_GRP1_PERIPH_TIM15
  - LL_APB2_GRP1_PERIPH_TIM16
  - LL_APB2_GRP1_PERIPH_TIM17
  - LL_APB2_GRP1_PERIPH_TIM18
  - LL_APB2_GRP1_PERIPH_USART1
  - LL_APB2_GRP1_PERIPH_USART6
  - LL_APB2_GRP1_PERIPH_UART9
  - LL_APB2_GRP1_PERIPH_USART10

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB2LPENSR TIM1LPENS LL_APB2_GRP1_EnableClockLowPower
- APB2LPENSR TIM8LPENS LL_APB2_GRP1_EnableClockLowPower
- APB2LPENSR USART1LPENS LL_APB2_GRP1_EnableClockLowPower
- APB2LPENSR USART6LPENS LL_APB2_GRP1_EnableClockLowPower
- APB2LPENSR UART9LPENS LL_APB2_GRP1_EnableClockLowPower
- APB2LPENSR USART10LPENS LL_APB2_GRP1_EnableClockLowPower
- APB2LPENSR SPI1LPENS LL_APB2_GRP1_EnableClockLowPower
- APB2LPENSR SPI4LPENS LL_APB2_GRP1_EnableClockLowPower
- APB2LPENSR TIM18LPENS LL_APB2_GRP1_EnableClockLowPower
- APB2LPENSR TIM15LPENS LL_APB2_GRP1_EnableClockLowPower
- APB2LPENSR TIM16LPENS LL_APB2_GRP1_EnableClockLowPower
- APB2LPENSR TIM17LPENS LL_APB2_GRP1_EnableClockLowPower
- APB2LPENSR TIM9LPENS LL_APB2_GRP1_EnableClockLowPower
- APB2LPENSR SPI5LPENS LL_APB2_GRP1_EnableClockLowPower
- APB2LPENSR SAI1LPENS LL_APB2_GRP1_EnableClockLowPower
- APB2LPENSR SAI2LPENS LL_APB2_GRP1_EnableClockLowPower

LL_APB2_GRP1_IsEnabledClockLowPower

Function name

__STATIC_INLINE uint32_t LL_APB2_GRP1_IsEnabledClockLowPower (uint32_t Periphs)

Function description

Check if APB2 peripheral clock during Low Power mode is enabled or not .

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB2_GRP1_PERIPH_SAI1
  - LL_APB2_GRP1_PERIPH_SAI2
  - LL_APB2_GRP1_PERIPH_SPI1
  - LL_APB2_GRP1_PERIPH_SPI4
  - LL_APB2_GRP1_PERIPH_SPI5
  - LL_APB2_GRP1_PERIPH_TIM1
  - LL_APB2_GRP1_PERIPH_TIM8
  - LL_APB2_GRP1_PERIPH_TIM9
  - LL_APB2_GRP1_PERIPH_TIM15
  - LL_APB2_GRP1_PERIPH_TIM16
  - LL_APB2_GRP1_PERIPH_TIM17
  - LL_APB2_GRP1_PERIPH_TIM18
  - LL_APB2_GRP1_PERIPH_USART1
  - LL_APB2_GRP1_PERIPH_USART6
  - LL_APB2_GRP1_PERIPH_UART9
  - LL_APB2_GRP1_PERIPH_USART10

Return values

- **uint32_t:**

Reference Manual to LL API cross reference:

- APB2LPENR TIM1LPEN LL_APB2_GRP1_IsEnabledClockLowPower
- APB2LPENR TIM8LPEN LL_APB2_GRP1_IsEnabledClockLowPower
- APB2LPENR USART1LPEN LL_APB2_GRP1_IsEnabledClockLowPower
- APB2LPENR USART6LPEN LL_APB2_GRP1_IsEnabledClockLowPower
- APB2LPENR UART9LPEN LL_APB2_GRP1_IsEnabledClockLowPower
- APB2LPENR USART10LPEN LL_APB2_GRP1_IsEnabledClockLowPower
- APB2LPENR SPI1LPEN LL_APB2_GRP1_IsEnabledClockLowPower
- APB2LPENR SPI4LPEN LL_APB2_GRP1_IsEnabledClockLowPower
- APB2LPENR TIM18LPEN LL_APB2_GRP1_IsEnabledClockLowPower
- APB2LPENR TIM15LPEN LL_APB2_GRP1_IsEnabledClockLowPower
- APB2LPENR TIM16LPEN LL_APB2_GRP1_IsEnabledClockLowPower
- APB2LPENR TIM17LPEN LL_APB2_GRP1_IsEnabledClockLowPower
- APB2LPENR TIM9LPEN LL_APB2_GRP1_IsEnabledClockLowPower
- APB2LPENR SPI5LPEN LL_APB2_GRP1_IsEnabledClockLowPower
- APB2LPENR SAI1LPEN LL_APB2_GRP1_IsEnabledClockLowPower
- APB2LPENR SAI2LPEN LL_APB2_GRP1_IsEnabledClockLowPower

LL_APB2_GRP1_DisableClockLowPower

Function name

__STATIC_INLINE void LL_APB2_GRP1_DisableClockLowPower (uint32_t Periphs)

Function description

Disable APB2 peripherals clock during Low Power mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB2_GRP1_PERIPH_SAI1
  - LL_APB2_GRP1_PERIPH_SAI2
  - LL_APB2_GRP1_PERIPH_SPI1
  - LL_APB2_GRP1_PERIPH_SPI4
  - LL_APB2_GRP1_PERIPH_SPI5
  - LL_APB2_GRP1_PERIPH_TIM1
  - LL_APB2_GRP1_PERIPH_TIM8
  - LL_APB2_GRP1_PERIPH_TIM9
  - LL_APB2_GRP1_PERIPH_TIM15
  - LL_APB2_GRP1_PERIPH_TIM16
  - LL_APB2_GRP1_PERIPH_TIM17
  - LL_APB2_GRP1_PERIPH_TIM18
  - LL_APB2_GRP1_PERIPH_USART1
  - LL_APB2_GRP1_PERIPH_USART6
  - LL_APB2_GRP1_PERIPH_UART9
  - LL_APB2_GRP1_PERIPH_USART10

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB2LPENCR TIM1LPENC LL_APB2_GRP1_DisableClockLowPower
- APB2LPENCR TIM8LPENC LL_APB2_GRP1_DisableClockLowPower
- APB2LPENCR USART1LPENC LL_APB2_GRP1_DisableClockLowPower
- APB2LPENCR USART6LPENC LL_APB2_GRP1_DisableClockLowPower
- APB2LPENCR UART9LPENC LL_APB2_GRP1_DisableClockLowPower
- APB2LPENCR USART10LPENC LL_APB2_GRP1_DisableClockLowPower
- APB2LPENCR SPI1LPENC LL_APB2_GRP1_DisableClockLowPower
- APB2LPENCR SPI4LPENC LL_APB2_GRP1_DisableClockLowPower
- APB2LPENCR TIM18LPENC LL_APB2_GRP1_DisableClockLowPower
- APB2LPENCR TIM15LPENC LL_APB2_GRP1_DisableClockLowPower
- APB2LPENCR TIM16LPENC LL_APB2_GRP1_DisableClockLowPower
- APB2LPENCR TIM17LPENC LL_APB2_GRP1_DisableClockLowPower
- APB2LPENCR TIM9LPENC LL_APB2_GRP1_DisableClockLowPower
- APB2LPENCR SPI5LPENC LL_APB2_GRP1_DisableClockLowPower
- APB2LPENCR SAI1LPENC LL_APB2_GRP1_DisableClockLowPower
- APB2LPENCR SAI2LPENC LL_APB2_GRP1_DisableClockLowPower

LL_APB4_GRP1_EnableClock

Function name

__STATIC_INLINE void LL_APB4_GRP1_EnableClock (uint32_t Periphs)

Function description

Enable APB4 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB4_GRP1_PERIPH_HDP
  - LL_APB4_GRP1_PERIPH_I2C4
  - LL_APB4_GRP1_PERIPH_LPTIM2
  - LL_APB4_GRP1_PERIPH_LPTIM3
  - LL_APB4_GRP1_PERIPH_LPTIM4
  - LL_APB4_GRP1_PERIPH_LPTIM5
  - LL_APB4_GRP1_PERIPH_LPUART1
  - LL_APB4_GRP1_PERIPH_RTC
  - LL_APB4_GRP1_PERIPH_RTCAPB
  - LL_APB4_GRP1_PERIPH_SPI6
  - LL_APB4_GRP1_PERIPH_VREFBUF

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB4ENSR1 HDPENS LL_APB4_GRP1_EnableClock
- APB4ENSR1 LPUART1ENS LL_APB4_GRP1_EnableClock
- APB4ENSR1 SPI6ENS LL_APB4_GRP1_EnableClock
- APB4ENSR1 I2C4ENS LL_APB4_GRP1_EnableClock
- APB4ENSR1 LPTIM2ENS LL_APB4_GRP1_EnableClock
- APB4ENSR1 LPTIM3ENS LL_APB4_GRP1_EnableClock
- APB4ENSR1 LPTIM4ENS LL_APB4_GRP1_EnableClock
- APB4ENSR1 LPTIM5ENS LL_APB4_GRP1_EnableClock
- APB4ENSR1 VREFBUFENS LL_APB4_GRP1_EnableClock
- APB4ENSR1 RTCENS LL_APB4_GRP1_EnableClock
- APB4ENSR1 RTCAPBENS LL_APB4_GRP1_EnableClock

LL_APB4_GRP1_IsEnabledClock

Function name

__STATIC_INLINE uint32_t LL_APB4_GRP1_IsEnabledClock (uint32_t Periphs)

Function description

Check if APB4 peripheral clock is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB4_GRP1_PERIPH_HDP
  - LL_APB4_GRP1_PERIPH_I2C4
  - LL_APB4_GRP1_PERIPH_LPTIM2
  - LL_APB4_GRP1_PERIPH_LPTIM3
  - LL_APB4_GRP1_PERIPH_LPTIM4
  - LL_APB4_GRP1_PERIPH_LPTIM5
  - LL_APB4_GRP1_PERIPH_LPUART1
  - LL_APB4_GRP1_PERIPH_RTC
  - LL_APB4_GRP1_PERIPH_RTCAPB
  - LL_APB4_GRP1_PERIPH_SPI6
  - LL_APB4_GRP1_PERIPH_VREFBUF

Return values

- **uint32_t:**

Reference Manual to LL API cross reference:

- APB4ENR1 HDPEN LL_APB4_GRP1_IsEnabledClock
- APB4ENR1 LPUART1EN LL_APB4_GRP1_IsEnabledClock
- APB4ENR1 SPI6EN LL_APB4_GRP1_IsEnabledClock
- APB4ENR1 I2C4EN LL_APB4_GRP1_IsEnabledClock
- APB4ENR1 LPTIM2EN LL_APB4_GRP1_IsEnabledClock
- APB4ENR1 LPTIM3EN LL_APB4_GRP1_IsEnabledClock
- APB4ENR1 LPTIM4EN LL_APB4_GRP1_IsEnabledClock
- APB4ENR1 LPTIM5EN LL_APB4_GRP1_IsEnabledClock
- APB4ENR1 VREFBUFEN LL_APB4_GRP1_IsEnabledClock
- APB4ENR1 RTCEN LL_APB4_GRP1_IsEnabledClock
- APB4ENR1 RTCAPBEN LL_APB4_GRP1_IsEnabledClock

LL_APB4_GRP1_DisableClock

Function name

__STATIC_INLINE void LL_APB4_GRP1_DisableClock (uint32_t Periphs)

Function description

Disable APB4 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB4_GRP1_PERIPH_HDP
  - LL_APB4_GRP1_PERIPH_I2C4
  - LL_APB4_GRP1_PERIPH_LPTIM2
  - LL_APB4_GRP1_PERIPH_LPTIM3
  - LL_APB4_GRP1_PERIPH_LPTIM4
  - LL_APB4_GRP1_PERIPH_LPTIM5
  - LL_APB4_GRP1_PERIPH_LPUART1
  - LL_APB4_GRP1_PERIPH_RTC
  - LL_APB4_GRP1_PERIPH_RTCAPB
  - LL_APB4_GRP1_PERIPH_SPI6
  - LL_APB4_GRP1_PERIPH_VREFBUF

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB4ENCR1 HDPENC LL_APB4_GRP1_DisableClock
- APB4ENCR1 LPUART1ENC LL_APB4_GRP1_DisableClock
- APB4ENCR1 SPI6ENC LL_APB4_GRP1_DisableClock
- APB4ENCR1 I2C4ENC LL_APB4_GRP1_DisableClock
- APB4ENCR1 LPTIM2ENC LL_APB4_GRP1_DisableClock
- APB4ENCR1 LPTIM3ENC LL_APB4_GRP1_DisableClock
- APB4ENCR1 LPTIM4ENC LL_APB4_GRP1_DisableClock
- APB4ENCR1 LPTIM5ENC LL_APB4_GRP1_DisableClock
- APB4ENCR1 VREFBUFENC LL_APB4_GRP1_DisableClock
- APB4ENCR1 RTCENC LL_APB4_GRP1_DisableClock
- APB4ENCR1 RTCAPBENC LL_APB4_GRP1_DisableClock

LL_APB4_GRP1_ForceReset

Function name

__STATIC_INLINE void LL_APB4_GRP1_ForceReset (uint32_t Periphs)

Function description

Force APB4 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB4_GRP1_PERIPH_HDP
  - LL_APB4_GRP1_PERIPH_I2C4
  - LL_APB4_GRP1_PERIPH_LPTIM2
  - LL_APB4_GRP1_PERIPH_LPTIM3
  - LL_APB4_GRP1_PERIPH_LPTIM4
  - LL_APB4_GRP1_PERIPH_LPTIM5
  - LL_APB4_GRP1_PERIPH_LPUART1
  - LL_APB4_GRP1_PERIPH_RTCAPB
  - LL_APB4_GRP1_PERIPH_SPI6
  - LL_APB4_GRP1_PERIPH_VREFBUF

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB4RSTSR1 HDPRSTS LL_APB4_GRP1_ForceReset
- APB4RSTSR1 LPUART1RSTS LL_APB4_GRP1_ForceReset
- APB4RSTSR1 SPI6RSTS LL_APB4_GRP1_ForceReset
- APB4RSTSR1 I2C4RSTS LL_APB4_GRP1_ForceReset
- APB4RSTSR1 LPTIM2RSTS LL_APB4_GRP1_ForceReset
- APB4RSTSR1 LPTIM3RSTS LL_APB4_GRP1_ForceReset
- APB4RSTSR1 LPTIM4RSTS LL_APB4_GRP1_ForceReset
- APB4RSTSR1 LPTIM5RSTS LL_APB4_GRP1_ForceReset
- APB4RSTSR1 VREFBUFRSTS LL_APB4_GRP1_ForceReset
- APB4RSTSR1 RTCAPBRSTS LL_APB4_GRP1_ForceReset

LL_APB4_GRP1_ReleaseReset

Function name

__STATIC_INLINE void LL_APB4_GRP1_ReleaseReset (uint32_t Periphs)

Function description

Release APB4 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB4_GRP1_PERIPH_HDP
  - LL_APB4_GRP1_PERIPH_I2C4
  - LL_APB4_GRP1_PERIPH_LPTIM2
  - LL_APB4_GRP1_PERIPH_LPTIM3
  - LL_APB4_GRP1_PERIPH_LPTIM4
  - LL_APB4_GRP1_PERIPH_LPTIM5
  - LL_APB4_GRP1_PERIPH_LPUART1
  - LL_APB4_GRP1_PERIPH_RTCAPB
  - LL_APB4_GRP1_PERIPH_SPI6
  - LL_APB4_GRP1_PERIPH_VREFBUF

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB4RSTCR1 HDPRSTC LL_APB4_GRP1_ReleaseReset
- APB4RSTCR1 LPUART1RSTC LL_APB4_GRP1_ReleaseReset
- APB4RSTCR1 SPI6RSTC LL_APB4_GRP1_ReleaseReset
- APB4RSTCR1 I2C4RSTC LL_APB4_GRP1_ReleaseReset
- APB4RSTCR1 LPTIM2RSTC LL_APB4_GRP1_ReleaseReset
- APB4RSTCR1 LPTIM3RSTC LL_APB4_GRP1_ReleaseReset
- APB4RSTCR1 LPTIM4RSTC LL_APB4_GRP1_ReleaseReset
- APB4RSTCR1 LPTIM5RSTC LL_APB4_GRP1_ReleaseReset
- APB4RSTCR1 VREFBUFRSTC LL_APB4_GRP1_ReleaseReset
- APB4RSTCR1 RTCAPBRSTC LL_APB4_GRP1_ReleaseReset

LL_APB4_GRP1_EnableClockLowPower

Function name

__STATIC_INLINE void LL_APB4_GRP1_EnableClockLowPower (uint32_t Periphs)

Function description

Enable APB4 peripherals clock during Low Power mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB4_GRP1_PERIPH_HDP
  - LL_APB4_GRP1_PERIPH_I2C4
  - LL_APB4_GRP1_PERIPH_LPTIM2
  - LL_APB4_GRP1_PERIPH_LPTIM3
  - LL_APB4_GRP1_PERIPH_LPTIM4
  - LL_APB4_GRP1_PERIPH_LPTIM5
  - LL_APB4_GRP1_PERIPH_LPUART1
  - LL_APB4_GRP1_PERIPH_RTC
  - LL_APB4_GRP1_PERIPH_RTCAPB
  - LL_APB4_GRP1_PERIPH_SPI6
  - LL_APB4_GRP1_PERIPH_VREFBUF

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB4LPENSR1 HDPLPENS LL_APB4_GRP1_EnableClockLowPower
- APB4LPENSR1 LPUART1LPENS LL_APB4_GRP1_EnableClockLowPower
- APB4LPENSR1 SPI6LPENS LL_APB4_GRP1_EnableClockLowPower
- APB4LPENSR1 I2C4LPENS LL_APB4_GRP1_EnableClockLowPower
- APB4LPENSR1 LPTIM2LPENS LL_APB4_GRP1_EnableClockLowPower
- APB4LPENSR1 LPTIM3LPENS LL_APB4_GRP1_EnableClockLowPower
- APB4LPENSR1 LPTIM4LPENS LL_APB4_GRP1_EnableClockLowPower
- APB4LPENSR1 LPTIM5LPENS LL_APB4_GRP1_EnableClockLowPower
- APB4LPENSR1 VREFBUFLPENS LL_APB4_GRP1_EnableClockLowPower
- APB4LPENSR1 RTCLPENS LL_APB4_GRP1_EnableClockLowPower
- APB4LPENSR1 RTCAPBLPENS LL_APB4_GRP1_EnableClockLowPower

LL_APB4_GRP1_IsEnabledClockLowPower

Function name

__STATIC_INLINE uint32_t LL_APB4_GRP1_IsEnabledClockLowPower (uint32_t Periphs)

Function description

Check if APB4 peripheral clock during Low Power mode is enabled or not .

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB4_GRP1_PERIPH_HDP
  - LL_APB4_GRP1_PERIPH_I2C4
  - LL_APB4_GRP1_PERIPH_LPTIM2
  - LL_APB4_GRP1_PERIPH_LPTIM3
  - LL_APB4_GRP1_PERIPH_LPTIM4
  - LL_APB4_GRP1_PERIPH_LPTIM5
  - LL_APB4_GRP1_PERIPH_LPUART1
  - LL_APB4_GRP1_PERIPH_RTC
  - LL_APB4_GRP1_PERIPH_RTCAPB
  - LL_APB4_GRP1_PERIPH_SPI6
  - LL_APB4_GRP1_PERIPH_VREFBUF

Return values

- **uint32_t:**

Reference Manual to LL API cross reference:

- APB4LPENR1 HDPLPEN LL_APB4_GRP1_IsEnabledClockLowPower
- APB4LPENR1 LPUART1LPEN LL_APB4_GRP1_IsEnabledClockLowPower
- APB4LPENR1 SPI6LPEN LL_APB4_GRP1_IsEnabledClockLowPower
- APB4LPENR1 I2C4LPEN LL_APB4_GRP1_IsEnabledClockLowPower
- APB4LPENR1 LPTIM2LPEN LL_APB4_GRP1_IsEnabledClockLowPower
- APB4LPENR1 LPTIM3LPEN LL_APB4_GRP1_IsEnabledClockLowPower
- APB4LPENR1 LPTIM4LPEN LL_APB4_GRP1_IsEnabledClockLowPower
- APB4LPENR1 LPTIM5LPEN LL_APB4_GRP1_IsEnabledClockLowPower
- APB4LPENR1 VREFBUFLPEN LL_APB4_GRP1_IsEnabledClockLowPower
- APB4LPENR1 RTCLPEN LL_APB4_GRP1_IsEnabledClockLowPower
- APB4LPENR1 RTCAPBLPEN LL_APB4_GRP1_IsEnabledClockLowPower

LL_APB4_GRP1_DisableClockLowPower

Function name

__STATIC_INLINE void LL_APB4_GRP1_DisableClockLowPower (uint32_t Periphs)

Function description

Disable APB4 peripherals clock during Low Power mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB4_GRP1_PERIPH_HDP
  - LL_APB4_GRP1_PERIPH_I2C4
  - LL_APB4_GRP1_PERIPH_LPTIM2
  - LL_APB4_GRP1_PERIPH_LPTIM3
  - LL_APB4_GRP1_PERIPH_LPTIM4
  - LL_APB4_GRP1_PERIPH_LPTIM5
  - LL_APB4_GRP1_PERIPH_LPUART1
  - LL_APB4_GRP1_PERIPH_RTC
  - LL_APB4_GRP1_PERIPH_RTCAPB
  - LL_APB4_GRP1_PERIPH_SPI6
  - LL_APB4_GRP1_PERIPH_VREFBUF

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB4LPENCR1 HDPLPENC LL_APB4_GRP1_DisableClockLowPower
- APB4LPENCR1 LPUART1LPENC LL_APB4_GRP1_DisableClockLowPower
- APB4LPENCR1 SPI6LPENC LL_APB4_GRP1_DisableClockLowPower
- APB4LPENCR1 I2C4LPENC LL_APB4_GRP1_DisableClockLowPower
- APB4LPENCR1 LPTIM2LPENC LL_APB4_GRP1_DisableClockLowPower
- APB4LPENCR1 LPTIM3LPENC LL_APB4_GRP1_DisableClockLowPower
- APB4LPENCR1 LPTIM4LPENC LL_APB4_GRP1_DisableClockLowPower
- APB4LPENCR1 LPTIM5LPENC LL_APB4_GRP1_DisableClockLowPower
- APB4LPENCR1 VREFBUFLPENC LL_APB4_GRP1_DisableClockLowPower
- APB4LPENCR1 RTCLPENC LL_APB4_GRP1_DisableClockLowPower
- APB4LPENCR1 RTCAPBLPENC LL_APB4_GRP1_DisableClockLowPower

LL_APB4_GRP2_EnableClock

Function name

__STATIC_INLINE void LL_APB4_GRP2_EnableClock (uint32_t Periphs)

Function description

Enable APB4 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB4_GRP2_PERIPH_BSEC
  - LL_APB4_GRP2_PERIPH_SYSCFG
  - LL_APB4_GRP2_PERIPH_DTS

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB4ENSR2 SYSCFGENS LL_APB4_GRP2_EnableClock
- APB4ENSR2 BSECENS LL_APB4_GRP2_EnableClock
- APB4ENSR2 DTSENS LL_APB4_GRP2_EnableClock

LL_APB4_GRP2_IsEnabledClock

Function name

__STATIC_INLINE uint32_t LL_APB4_GRP2_IsEnabledClock (uint32_t Periphs)

Function description

Check if APB4 peripheral clock is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB4_GRP2_PERIPH_BSEC
  - LL_APB4_GRP2_PERIPH_SYSCFG
  - LL_APB4_GRP2_PERIPH_DTS

Return values

- **uint32_t:**

Reference Manual to LL API cross reference:

- APB4ENR2 SYSCFGEN LL_APB4_GRP2_IsEnabledClock
- APB4ENR2 BSECEN LL_APB4_GRP2_IsEnabledClock
- APB4ENR2 DTSEN LL_APB4_GRP2_IsEnabledClock

LL_APB4_GRP2_DisableClock

Function name

__STATIC_INLINE void LL_APB4_GRP2_DisableClock (uint32_t Periphs)

Function description

Disable APB4 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB4_GRP2_PERIPH_BSEC
  - LL_APB4_GRP2_PERIPH_SYSCFG
  - LL_APB4_GRP2_PERIPH_DTS

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB4ENCR2 SYSCFGENC LL_APB4_GRP2_DisableClock
- APB4ENCR2 BSECENC LL_APB4_GRP2_DisableClock
- APB4ENCR2 DTSENC LL_APB4_GRP2_DisableClock

LL_APB4_GRP2_ForceReset

Function name

__STATIC_INLINE void LL_APB4_GRP2_ForceReset (uint32_t Periphs)

Function description

Force APB4 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB4_GRP2_PERIPH_SYSCFG
  - LL_APB4_GRP2_PERIPH_DTS

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB4RSTR2 SYSCFGRSTS LL_APB4_GRP2_ForceReset
- APB4RSTR2 DTSRSTS LL_APB4_GRP2_ForceReset

LL_APB4_GRP2_ReleaseReset

Function name

__STATIC_INLINE void LL_APB4_GRP2_ReleaseReset (uint32_t Periphs)

Function description

Release APB4 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB4_GRP2_PERIPH_SYSCFG
  - LL_APB4_GRP2_PERIPH_DTS

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB4RSTCR2 SYSCFGRSTC LL_APB4_GRP2_ReleaseReset
- APB4RSTCR2 DTSRSTC LL_APB4_GRP2_ReleaseReset

LL_APB4_GRP2_EnableClockLowPower

Function name

__STATIC_INLINE void LL_APB4_GRP2_EnableClockLowPower (uint32_t Periphs)

Function description

Enable APB4 peripherals clock during Low Power mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB4_GRP2_PERIPH_BSEC
  - LL_APB4_GRP2_PERIPH_SYSCFG
  - LL_APB4_GRP2_PERIPH_DTS

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB4LPENSR2 SYSCFGLPENS LL_APB4_GRP2_EnableClockLowPower
- APB4LPENSR2 BSECLPENS LL_APB4_GRP2_EnableClockLowPower
- APB4LPENSR2 DTSLPENS LL_APB4_GRP2_EnableClockLowPower

LL_APB4_GRP2_IsEnabledClockLowPower

Function name

__STATIC_INLINE uint32_t LL_APB4_GRP2_IsEnabledClockLowPower (uint32_t Periphs)

Function description

Check if APB4 peripheral clock during Low Power mode is enabled or not .

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB4_GRP2_PERIPH_BSEC
  - LL_APB4_GRP2_PERIPH_SYSCFG
  - LL_APB4_GRP2_PERIPH_DTS

Return values

- **uint32_t:**

Reference Manual to LL API cross reference:

- APB4LPENR2 SYSCFGLPEN LL_APB4_GRP2_IsEnabledClockLowPower
- APB4LPENR2 BSECLPEN LL_APB4_GRP2_IsEnabledClockLowPower
- APB4LPENR2 DTSLPEN LL_APB4_GRP2_IsEnabledClockLowPower

LL_APB4_GRP2_DisableClockLowPower

Function name

__STATIC_INLINE void LL_APB4_GRP2_DisableClockLowPower (uint32_t Periphs)

Function description

Disable APB4 peripherals clock during Low Power mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB4_GRP2_PERIPH_BSEC
  - LL_APB4_GRP2_PERIPH_SYSCFG
  - LL_APB4_GRP2_PERIPH_DTS

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB4LPENCR2 SYSCFGLPENC LL_APB4_GRP2_DisableClockLowPower
- APB4LPENCR2 BSECLPENC LL_APB4_GRP2_DisableClockLowPower
- APB4LPENCR2 DTSLPENC LL_APB4_GRP2_DisableClockLowPower

LL_APB5_GRP1_EnableClock

Function name

__STATIC_INLINE void LL_APB5_GRP1_EnableClock (uint32_t Periphs)

Function description

Enable APB5 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB5_GRP1_PERIPH_CSI
  - LL_APB5_GRP1_PERIPH_DCMIPP
  - LL_APB5_GRP1_PERIPH_GFXTIM
  - LL_APB5_GRP1_PERIPH_LTDC
  - LL_APB5_GRP1_PERIPH_VENC

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB5ENSR LTDCENS LL_APB5_GRP1_EnableClock
- APB5ENSR DCMIPPENS LL_APB5_GRP1_EnableClock
- APB5ENSR GFXTIMENS LL_APB5_GRP1_EnableClock
- APB5ENSR VENCENS LL_APB5_GRP1_EnableClock
- (*) APB5ENSR CSIENS LL_APB5_GRP1_EnableClock

LL_APB5_GRP1_IsEnabledClock

Function name

__STATIC_INLINE uint32_t LL_APB5_GRP1_IsEnabledClock (uint32_t Periphs)

Function description

Check if APB5 peripheral clock is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB5_GRP1_PERIPH_CSI
  - LL_APB5_GRP1_PERIPH_DCMIPP
  - LL_APB5_GRP1_PERIPH_GFXTIM
  - LL_APB5_GRP1_PERIPH_LTDC
  - LL_APB5_GRP1_PERIPH_VENC

Return values

- **uint32_t:**

Reference Manual to LL API cross reference:

- APB5ENR LTDCEN LL_APB5_GRP1_IsEnabledClock
- APB5ENR DCMIPPEN LL_APB5_GRP1_IsEnabledClock
- APB5ENR GFXTIMEN LL_APB5_GRP1_IsEnabledClock
- APB5ENR VENCEN LL_APB5_GRP1_IsEnabledClock
- (*) APB5ENR CSIEN LL_APB5_GRP1_IsEnabledClock

LL_APB5_GRP1_DisableClock

Function name

__STATIC_INLINE void LL_APB5_GRP1_DisableClock (uint32_t Periphs)

Function description

Disable APB5 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB5_GRP1_PERIPH_CSI
  - LL_APB5_GRP1_PERIPH_DCMIPP
  - LL_APB5_GRP1_PERIPH_GFXTIM
  - LL_APB5_GRP1_PERIPH_LTDC
  - LL_APB5_GRP1_PERIPH_VENC

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB5ENCR LTDCENC LL_APB5_GRP1_DisableClock
- APB5ENCR DCMIPPENC LL_APB5_GRP1_DisableClock
- APB5ENCR GFXTIMENC LL_APB5_GRP1_DisableClock
- APB5ENCR VENCENC LL_APB5_GRP1_DisableClock
- (*) APB5ENCR CSIENC LL_APB5_GRP1_DisableClock

LL_APB5_GRP1_ForceReset

Function name

__STATIC_INLINE void LL_APB5_GRP1_ForceReset (uint32_t Periphs)

Function description

Force APB5 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB5_GRP1_PERIPH_CSI
  - LL_APB5_GRP1_PERIPH_DCMIPP
  - LL_APB5_GRP1_PERIPH_GFXTIM
  - LL_APB5_GRP1_PERIPH_LTDC
  - LL_APB5_GRP1_PERIPH_VENC

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB5RSTSR LTDCRSTS LL_APB5_GRP1_ForceReset
- APB5RSTSR DCMIPPRSTS LL_APB5_GRP1_ForceReset
- APB5RSTSR GFXTIMRSTS LL_APB5_GRP1_ForceReset
- APB5RSTSR VENCRSTS LL_APB5_GRP1_ForceReset
- (*) APB5RSTSR CSIRSTS LL_APB5_GRP1_ForceReset

LL_APB5_GRP1_ReleaseReset

Function name

__STATIC_INLINE void LL_APB5_GRP1_ReleaseReset (uint32_t Periphs)

Function description

Release APB5 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB5_GRP1_PERIPH_CSI
  - LL_APB5_GRP1_PERIPH_DCMIPP
  - LL_APB5_GRP1_PERIPH_GFXTIM
  - LL_APB5_GRP1_PERIPH_LTDC
  - LL_APB5_GRP1_PERIPH_VENC

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB5RSTCR LTDCRSTC LL_APB5_GRP1_ReleaseReset
- APB5RSTCR DCMIPPRSTC LL_APB5_GRP1_ReleaseReset
- APB5RSTCR GFXTIMRSTC LL_APB5_GRP1_ReleaseReset
- APB5RSTCR VENCRSTC LL_APB5_GRP1_ReleaseReset
- (*) APB5RSTCR CSIRSTC LL_APB5_GRP1_ReleaseReset

LL_APB5_GRP1_EnableClockLowPower

Function name

__STATIC_INLINE void LL_APB5_GRP1_EnableClockLowPower (uint32_t Periphs)

Function description

Enable APB5 peripherals clock during Low Power mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB5_GRP1_PERIPH_CSI
  - LL_APB5_GRP1_PERIPH_DCMIPP
  - LL_APB5_GRP1_PERIPH_GFXTIM
  - LL_APB5_GRP1_PERIPH_LTDC
  - LL_APB5_GRP1_PERIPH_VENC

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB5LPENSR LTDCLPENS LL_APB5_GRP1_EnableClockLowPower
- APB5LPENSR DCMIPPLPENS LL_APB5_GRP1_EnableClockLowPower
- APB5LPENSR GFXTIMLPENS LL_APB5_GRP1_EnableClockLowPower
- APB5LPENSR VENCLPENS LL_APB5_GRP1_EnableClockLowPower
- (*) APB5LPENSR CSILPENS LL_APB5_GRP1_EnableClockLowPower

LL_APB5_GRP1_IsEnabledClockLowPower

Function name

__STATIC_INLINE uint32_t LL_APB5_GRP1_IsEnabledClockLowPower (uint32_t Periphs)

Function description

Check if APB5 peripheral clock during Low Power mode is enabled or not .

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB5_GRP1_PERIPH_CSI
  - LL_APB5_GRP1_PERIPH_DCMIPP
  - LL_APB5_GRP1_PERIPH_GFXTIM
  - LL_APB5_GRP1_PERIPH_LTDC
  - LL_APB5_GRP1_PERIPH_VENC

Return values

- **uint32_t:**

Reference Manual to LL API cross reference:

- APB5LPENR LTDCLPEN LL_APB5_GRP1_IsEnabledClockLowPower
- APB5LPENR DCMIPPLPEN LL_APB5_GRP1_IsEnabledClockLowPower
- APB5LPENR GFXTIMLPEN LL_APB5_GRP1_IsEnabledClockLowPower
- APB5LPENR VENCLPEN LL_APB5_GRP1_IsEnabledClockLowPower
- (*) APB5LPENR CSILPEN LL_APB5_GRP1_IsEnabledClockLowPower

LL_APB5_GRP1_DisableClockLowPower

Function name

__STATIC_INLINE void LL_APB5_GRP1_DisableClockLowPower (uint32_t Periphs)

Function description

Disable APB5 peripherals clock during Low Power mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB5_GRP1_PERIPH_CSI
  - LL_APB5_GRP1_PERIPH_DCMIPP
  - LL_APB5_GRP1_PERIPH_GFXTIM
  - LL_APB5_GRP1_PERIPH_LTDC
  - LL_APB5_GRP1_PERIPH_VENC

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB5LPENCR LTDCLPENC LL_APB5_GRP1_DisableClockLowPower
- APB5LPENCR DCMIPPLPENC LL_APB5_GRP1_DisableClockLowPower
- APB5LPENCR GFXTIMLPENC LL_APB5_GRP1_DisableClockLowPower
- APB5LPENCR VENCLPENC LL_APB5_GRP1_DisableClockLowPower
- (*) APB5LPENCR CSILPENC LL_APB5_GRP1_DisableClockLowPower

LL_MISC_EnableClock

Function name

__STATIC_INLINE void LL_MISC_EnableClock (uint32_t Misc)

Function description

Enable miscellaneous clock.

Parameters

- **Misc:** This parameter can be a combination of the following values:
  - LL_DBG
  - LL_MCO1
  - LL_MCO2
  - LL_XSPIPHYCOMP
  - LL_PER

Return values

- **None:**

Reference Manual to LL API cross reference:

- MISCENSR DBGENS LL_MISC_EnableClock
- MISCENSR MCO1ENS LL_MISC_EnableClock
- MISCENSR MCO2ENS LL_MISC_EnableClock
- MISCENSR XSPIPHYCOMPENS LL_MISC_EnableClock
- MISCENSR PERENS LL_MISC_EnableClock

LL_MISC_IsEnabledClock

Function name

__STATIC_INLINE uint32_t LL_MISC_IsEnabledClock (uint32_t Misc)

Function description

Check if miscellaneous clock is enabled or not.

Parameters

- **Misc:** This parameter can be a combination of the following values:
  - LL_DBG
  - LL_MCO1
  - LL_MCO2
  - LL_XSPIPHYCOMP
  - LL_PER

Return values

- **uint32_t:**

Reference Manual to LL API cross reference:

- MISCENR DBGEN LL_MISC_IsEnabledClock
- MISCENR MCO1EN LL_MISC_IsEnabledClock
- MISCENR MCO2EN LL_MISC_IsEnabledClock
- MISCENR XSPIPHYCOMPEN LL_MISC_IsEnabledClock
- MISCENR PEREN LL_MISC_IsEnabledClock

LL_MISC_DisableClock

Function name

__STATIC_INLINE void LL_MISC_DisableClock (uint32_t Misc)

Function description

Disable miscellaneous clock.

Parameters

- **Misc:** This parameter can be a combination of the following values:
  - LL_DBG
  - LL_MCO1
  - LL_MCO2
  - LL_XSPIPHYCOMP
  - LL_PER

Return values

- **None:**

Reference Manual to LL API cross reference:

- MISCENCR DBGENC LL_MISC_DisableClock
- MISCENCR MCO1ENC LL_MISC_DisableClock
- MISCENCR MCO2ENC LL_MISC_DisableClock
- MISCENCR XSPIPHYCOMPENC LL_MISC_DisableClock
- MISCENCR PERENC LL_MISC_DisableClock

LL_MISC_ForceReset

Function name

__STATIC_INLINE void LL_MISC_ForceReset (uint32_t Misc)

Function description

Force miscellaneous reset.

Parameters

- **Misc:** This parameter can be a combination of the following values:
  - LL_DBG
  - LL_XSPIPHY1
  - LL_XSPIPHY2
  - LL_SDMMC1DLL
  - LL_SDMMC2DLL

Return values

- **None:**

Reference Manual to LL API cross reference:

- MISCRSTSR DBGRSTS LL_MISC_ForceReset
- MISCRSTSR XSPIPHY1RSTS LL_MISC_ForceReset
- MISCRSTSR XSPIPHY2RSTS LL_MISC_ForceReset
- MISCRSTSR SDMMC1DLLRSTS LL_MISC_ForceReset
- MISCRSTSR SDMMC2DLLRSTS LL_MISC_ForceReset

LL_MISC_ReleaseReset

Function name

__STATIC_INLINE void LL_MISC_ReleaseReset (uint32_t Misc)

Function description

Release miscellaneous reset.

Parameters

- **Misc:** This parameter can be a combination of the following values:
  - LL_DBG
  - LL_XSPIPHY1
  - LL_XSPIPHY2
  - LL_SDMMC1DLL
  - LL_SDMMC2DLL

Return values

- **None:**

Reference Manual to LL API cross reference:

- MISCRSTCR DBGRSTC LL_MISC_ReleaseReset
- MISCRSTCR XSPIPHY1RSTC LL_MISC_ReleaseReset
- MISCRSTCR XSPIPHY2RSTC LL_MISC_ReleaseReset
- MISCRSTCR SDMMC1DLLRSTC LL_MISC_ReleaseReset
- MISCRSTCR SDMMC2DLLRSTC LL_MISC_ReleaseReset

LL_MISC_EnableClockLowPower

Function name

__STATIC_INLINE void LL_MISC_EnableClockLowPower (uint32_t Misc)

Function description

Enable bus clock during Low Power mode.

Parameters

- **Misc:** This parameter can be a combination of the following values:
  - LL_DBG
  - LL_XSPIPHYCOMP
  - LL_PER

Return values

- **None:**

Reference Manual to LL API cross reference:

- MISCLPENSR DBGLPENS LL_MISC_EnableClockLowPower
- MISCLPENSR XSPIPHYCOMPLPENS LL_MISC_EnableClockLowPower
- MISCLPENSR PERLPENS LL_MISC_EnableClockLowPower

LL_MISC_DisableClockLowPower

Function name

__STATIC_INLINE void LL_MISC_DisableClockLowPower (uint32_t Misc)

Function description

Disable bus clock during Low Power mode.

Parameters

- **Misc:** This parameter can be a combination of the following values:
  - LL_DBG
  - LL_XSPIPHYCOMP
  - LL_PER

Return values

- **None:**

Reference Manual to LL API cross reference:

- MISCLPENCR DBGLPENC LL_MISC_DisableClockLowPower
- MISCLPENCR XSPIPHYCOMPLPENC LL_MISC_DisableClockLowPower
- MISCLPENCR PERLPENC LL_MISC_DisableClockLowPower

86.2 BUS Firmware driver defines

The following section lists the various define and macros of the module.

86.2.1 BUS

BUS

AHB1 GRP1 PERIPH

LL_AHB1_GRP1_PERIPH_ADC12

LL_AHB1_GRP1_PERIPH_GPDMA1

LL_AHB1_GRP1_PERIPH_ALL

AHB2 GRP1 PERIPH

LL_AHB2_GRP1_PERIPH_ADF1

LL_AHB2_GRP1_PERIPH_MDF1

LL_AHB2_GRP1_PERIPH_RAMCFG

LL_AHB2_GRP1_PERIPH_ALL

AHB3 GRP1 PERIPH

LL_AHB3_GRP1_PERIPH_CRYP

LL_AHB3_GRP1_PERIPH_HASH

LL_AHB3_GRP1_PERIPH_IAC

LL_AHB3_GRP1_PERIPH_PKA

LL_AHB3_GRP1_PERIPH_RIFSC

LL_AHB3_GRP1_PERIPH_RISAF

LL_AHB3_GRP1_PERIPH_RNG

LL_AHB3_GRP1_PERIPH_SAES

LL_AHB3_GRP1_PERIPH_ALL

AHB4 GRP1 PERIPH

LL_AHB4_GRP1_PERIPH_CRC

LL_AHB4_GRP1_PERIPH_GPIOA

LL_AHB4_GRP1_PERIPH_GPIOB

LL_AHB4_GRP1_PERIPH_GPIOC

LL_AHB4_GRP1_PERIPH_GPIOD

LL_AHB4_GRP1_PERIPH_GPIOE

LL_AHB4_GRP1_PERIPH_GPIOF

LL_AHB4_GRP1_PERIPH_GPIOG

LL_AHB4_GRP1_PERIPH_GPIOH

LL_AHB4_GRP1_PERIPH_GPION

LL_AHB4_GRP1_PERIPH_GPIOO

LL_AHB4_GRP1_PERIPH_GPIOP

LL_AHB4_GRP1_PERIPH_GPIOQ

LL_AHB4_GRP1_PERIPH_PWR

LL_AHB4_GRP1_PERIPH_ALL

AHB5 GRP1 PERIPH

LL_AHB5_GRP1_PERIPH_DMA2D

LL_AHB5_GRP1_PERIPH_ETH1

LL_AHB5_GRP1_PERIPH_ETH1MAC

LL_AHB5_GRP1_PERIPH_ETH1TX

LL_AHB5_GRP1_PERIPH_ETH1RX

LL_AHB5_GRP1_PERIPH_FMC

LL_AHB5_GRP1_PERIPH_GFXMMU

LL_AHB5_GRP1_PERIPH_GPU2D

LL_AHB5_GRP1_PERIPH_HPDMA1

LL_AHB5_GRP1_PERIPH_XSPI1

LL_AHB5_GRP1_PERIPH_XSPI2

LL_AHB5_GRP1_PERIPH_XSPI3

LL_AHB5_GRP1_PERIPH_XSPIM

LL_AHB5_GRP1_PERIPH_JPEG

LL_AHB5_GRP1_PERIPH_MCE1

LL_AHB5_GRP1_PERIPH_MCE2

LL_AHB5_GRP1_PERIPH_MCE3

LL_AHB5_GRP1_PERIPH_MCE4

LL_AHB5_GRP1_PERIPH_CACHEAXI

LL_AHB5_GRP1_PERIPH_NPU

LL_AHB5_GRP1_PERIPH_OTG1

LL_AHB5_GRP1_PERIPH_OTG2

LL_AHB5_GRP1_PERIPH_OTGPHY1

LL_AHB5_GRP1_PERIPH_OTGPHY2

LL_AHB5_GRP1_PERIPH_OTG1PHYCTL

LL_AHB5_GRP1_PERIPH_OTG2PHYCTL

LL_AHB5_GRP1_PERIPH_PSSI

LL_AHB5_GRP1_PERIPH_SDMMC1

LL_AHB5_GRP1_PERIPH_SDMMC2

LL_AHB5_GRP1_PERIPH_ALL

APB1 GRP1 PERIPH

LL_APB1_GRP1_PERIPH_I2C1

LL_APB1_GRP1_PERIPH_I2C2

LL_APB1_GRP1_PERIPH_I2C3

LL_APB1_GRP1_PERIPH_I3C1

LL_APB1_GRP1_PERIPH_I3C2

LL_APB1_GRP1_PERIPH_LPTIM1

LL_APB1_GRP1_PERIPH_SPDIFRX1

LL_APB1_GRP1_PERIPH_SPI2

LL_APB1_GRP1_PERIPH_SPI3

LL_APB1_GRP1_PERIPH_TIM2

LL_APB1_GRP1_PERIPH_TIM3

LL_APB1_GRP1_PERIPH_TIM4

LL_APB1_GRP1_PERIPH_TIM5

LL_APB1_GRP1_PERIPH_TIM6

LL_APB1_GRP1_PERIPH_TIM7

LL_APB1_GRP1_PERIPH_TIM10

LL_APB1_GRP1_PERIPH_TIM11

LL_APB1_GRP1_PERIPH_TIM12

LL_APB1_GRP1_PERIPH_TIM13

LL_APB1_GRP1_PERIPH_TIM14

LL_APB1_GRP1_PERIPH_USART2

LL_APB1_GRP1_PERIPH_USART3

LL_APB1_GRP1_PERIPH_UART4

LL_APB1_GRP1_PERIPH_UART5

LL_APB1_GRP1_PERIPH_UART7

LL_APB1_GRP1_PERIPH_UART8

LL_APB1_GRP1_PERIPH_WWDG

LL_APB1_GRP1_PERIPH_ALL

APB1 GRP2 PERIPH

LL_APB1_GRP2_PERIPH_FDCAN

LL_APB1_GRP2_PERIPH_MDIOS

LL_APB1_GRP2_PERIPH_UCPD1

LL_APB1_GRP2_PERIPH_ALL

APB2 GRP1 PERIPH

LL_APB2_GRP1_PERIPH_SAI1

LL_APB2_GRP1_PERIPH_SAI2

LL_APB2_GRP1_PERIPH_SPI1

LL_APB2_GRP1_PERIPH_SPI4

LL_APB2_GRP1_PERIPH_SPI5

LL_APB2_GRP1_PERIPH_TIM1

LL_APB2_GRP1_PERIPH_TIM8

LL_APB2_GRP1_PERIPH_TIM9

LL_APB2_GRP1_PERIPH_TIM15

LL_APB2_GRP1_PERIPH_TIM16

LL_APB2_GRP1_PERIPH_TIM17

LL_APB2_GRP1_PERIPH_TIM18

LL_APB2_GRP1_PERIPH_USART1

LL_APB2_GRP1_PERIPH_USART6

LL_APB2_GRP1_PERIPH_UART9

LL_APB2_GRP1_PERIPH_USART10

LL_APB2_GRP1_PERIPH_ALL

APB4 GRP1 PERIPH

LL_APB4_GRP1_PERIPH_HDP

LL_APB4_GRP1_PERIPH_I2C4

LL_APB4_GRP1_PERIPH_LPTIM2

LL_APB4_GRP1_PERIPH_LPTIM3

LL_APB4_GRP1_PERIPH_LPTIM4

LL_APB4_GRP1_PERIPH_LPTIM5

LL_APB4_GRP1_PERIPH_LPUART1

LL_APB4_GRP1_PERIPH_RTC

LL_APB4_GRP1_PERIPH_RTCAPB

LL_APB4_GRP1_PERIPH_SPI6

LL_APB4_GRP1_PERIPH_VREFBUF

LL_APB4_GRP1_PERIPH_ALL

APB4 GRP2 PERIPH

LL_APB4_GRP2_PERIPH_BSEC

LL_APB4_GRP2_PERIPH_DTS

LL_APB4_GRP2_PERIPH_SYSCFG

LL_APB4_GRP2_PERIPH_ALL

APB5 GRP1 PERIPH

LL_APB5_GRP1_PERIPH_CSI

LL_APB5_GRP1_PERIPH_DCMIPP

LL_APB5_GRP1_PERIPH_GFXTIM

LL_APB5_GRP1_PERIPH_LTDC

LL_APB5_GRP1_PERIPH_VENC

LL_APB5_GRP1_PERIPH_ALL

AXI

LL_ACLKN

LL_ACLKNC

LL_AHBM

LL_AHB1

LL_AHB2

LL_AHB3

LL_AHB4

LL_AHB5

LL_APB1

LL_APB2

LL_APB3

LL_APB4

LL_APB5

MEM

LL_MEM_AXISRAM1

LL_MEM_AXISRAM2

LL_MEM_AXISRAM3

LL_MEM_AXISRAM4

LL_MEM_AXISRAM5

LL_MEM_AXISRAM6

LL_MEM_AHBSRAM1

LL_MEM_AHBSRAM2

LL_MEM_BKPSRAM

LL_MEM_FLEXRAM

LL_MEM_CACHEAXIRAM

LL_MEM_VENCRAM

LL_MEM_BOOTROM

MISC

LL_DBG

LL_MCO1

LL_MCO2

LL_XSPIPHYCOMP

LL_XSPIPHY1

LL_XSPIPHY2

LL_PER

LL_SDMMC1DLL

LL_SDMMC2DLL

87 LL CACHEAXI Generic Driver

87.1 CACHEAXI Firmware driver API description

The following section lists the various functions of the CACHEAXI library.

87.1.1 Detailed description of functions

LL_CACHEAXI_Enable

Function name

__STATIC_INLINE void LL_CACHEAXI_Enable (CACHEAXI_TypeDef * CACHEAXIx)

Function description

Enable the selected CACHEAXI instance.

Parameters

- **CACHEAXIx:** CACHEAXI instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 EN LL_CACHEAXI_Enable

LL_CACHEAXI_Disable

Function name

__STATIC_INLINE void LL_CACHEAXI_Disable (CACHEAXI_TypeDef * CACHEAXIx)

Function description

Disable the selected CACHEAXI instance.

Parameters

- **CACHEAXIx:** CACHEAXI instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 EN LL_CACHEAXI_Disable

LL_CACHEAXI_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_CACHEAXI_IsEnabled (const CACHEAXI_TypeDef * CACHEAXIx)

Function description

Get the selected CACHEAXI instance enable state.

Parameters

- **CACHEAXIx:** CACHEAXI instance

Return values

- **0:** CACHEAXI is disabled, 1: CACHEAXI is enabled.

Reference Manual to LL API cross reference:

- CR1 EN LL_CACHEAXI_IsEnabled

LL_CACHEAXI_SetStartAddress

Function name

__STATIC_INLINE void LL_CACHEAXI_SetStartAddress (CACHEAXI_TypeDef * CACHEAXIx, uint32_t addr)

Function description

Set the cacheaxi instance start command address.

Parameters

- **addr:** cacheaxi command start address(Clean, Invalidate or Clean and Invalidate).
- **CACHEAXIx:** CACHEAXI instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 CMDRSADDRR LL_CACHEAXI_SetStartAddress

LL_CACHEAXI_GetStartAddress

Function name

__STATIC_INLINE uint32_t LL_CACHEAXI_GetStartAddress (const CACHEAXI_TypeDef * CACHEAXIx)

Function description

Get the cacheaxi command start address.

Parameters

- **CACHEAXIx:** CACHEAXI instance

Return values

- **Start:** address of cacheaxi command

Reference Manual to LL API cross reference:

- CR2 CMDRSADDRR LL_CACHEAXI_GetStartAddress

LL_CACHEAXI_SetEndAddress

Function name

__STATIC_INLINE void LL_CACHEAXI_SetEndAddress (CACHEAXI_TypeDef * CACHEAXIx, uint32_t addr)

Function description

Set the cacheaxi instance End command address.

Parameters

- **CACHEAXIx:** CACHEAXI instance
- **addr:** cacheaxi command end address(Clean, Invalidate or Clean and Invalidate).

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 CMDREADDRR LL_CACHEAXI_SetEndAddress

LL_CACHEAXI_GetEndAddress

Function name

__STATIC_INLINE uint32_t LL_CACHEAXI_GetEndAddress (const CACHEAXI_TypeDef * CACHEAXIx)

Function description

Get the cacheaxi command End address.

Parameters

- **CACHEAXIx:** CACHEAXI instance

Return values

- **End:** address of cacheaxi command

Reference Manual to LL API cross reference:

- CR2 CMDREADDRR LL_CACHEAXI_GetEndAddress

LL_CACHEAXI_SetCommand

Function name

__STATIC_INLINE void LL_CACHEAXI_SetCommand (CACHEAXI_TypeDef * CACHEAXIx, uint32_t Command)

Function description

Set Dcache command.

Parameters

- **CACHEAXIx:** CACHEAXI instance
- **Command:** command to be applied for the cacheaxi Command can be one of the following values:
  - LL_CACHEAXI_COMMAND_CLEAN_BY_ADDR
  - LL_CACHEAXI_COMMAND_CLEAN_INVALIDATE_BY_ADDR
  - LL_CACHEAXI_COMMAND_NO_OPERATION

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 CACHECMD LL_CACHEAXI_SetCommand

LL_CACHEAXI_GetCommand

Function name

__STATIC_INLINE uint32_t LL_CACHEAXI_GetCommand (const CACHEAXI_TypeDef * CACHEAXIx)

Function description

Set Dcache command.

Parameters

- **CACHEAXIx:** CACHEAXI instance

Return values

- **Returned:** value can be one of the following values:
  - LL_CACHEAXI_COMMAND_NO_OPERATION
  - LL_CACHEAXI_COMMAND_CLEAN_BY_ADDR
  - LL_CACHEAXI_COMMAND_CLEAN_INVALIDATE_BY_ADDR

Reference Manual to LL API cross reference:

- CR2 CACHECMD LL_CACHEAXI_GetCommand

LL_CACHEAXI_StartCommand

Function name

__STATIC_INLINE void LL_CACHEAXI_StartCommand (CACHEAXI_TypeDef * CACHEAXIx)

Function description

Launch Dcache Command.

Parameters

- **CACHEAXIx:** CACHEAXI instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 CACHECMD LL_CACHEAXI_StartCommand

LL_CACHEAXI_Invalidate

Function name

__STATIC_INLINE void LL_CACHEAXI_Invalidate (CACHEAXI_TypeDef * CACHEAXIx)

Function description

Invalidate the Data cache.

Parameters

- **CACHEAXIx:** CACHEAXI instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 CACHEINV LL_CACHEAXI_Invalidate

LL_CACHEAXI_EnableMonitors

Function name

__STATIC_INLINE void LL_CACHEAXI_EnableMonitors (CACHEAXI_TypeDef * CACHEAXIx, uint32_t Monitors)

Function description

Enable the hit/miss monitor(s).

Parameters

- **CACHEAXIx:** CACHEAXI instance
- **Monitors:** This parameter can be one or a combination of the following values:
  - LL_CACHEAXI_MONITOR_READ_HIT
  - LL_CACHEAXI_MONITOR_READ_MISS
  - LL_CACHEAXI_MONITOR_WRITE_HIT
  - LL_CACHEAXI_MONITOR_WRITE_MISS
  - LL_CACHEAXI_MONITOR_READALLOC_MISS
  - LL_CACHEAXI_MONITOR_WRITEALLOC_MISS
  - LL_CACHEAXI_MONITOR_WRITETHROUGH
  - LL_CACHEAXI_MONITOR_EVICTION
  - LL_CACHEAXI_MONITOR_ALL

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 (RMISSMEN/RHITMEN/WMISSMEN/WHITMEN/RAMMEN/WAMMEN/WTMEN/EVIMEN) LL_CACHEAXI_EnableMonitors

LL_CACHEAXI_DisableMonitors

Function name

__STATIC_INLINE void LL_CACHEAXI_DisableMonitors (CACHEAXI_TypeDef * CACHEAXIx, uint32_t Monitors)

Function description

Disable the hit/miss monitor(s).

Parameters

- **CACHEAXIx:** CACHEAXI instance
- **Monitors:** This parameter can be one or a combination of the following values:
  - LL_CACHEAXI_MONITOR_READ_HIT
  - LL_CACHEAXI_MONITOR_READ_MISS
  - LL_CACHEAXI_MONITOR_WRITE_HIT
  - LL_CACHEAXI_MONITOR_WRITE_MISS
  - LL_CACHEAXI_MONITOR_READALLOC_MISS
  - LL_CACHEAXI_MONITOR_WRITEALLOC_MISS
  - LL_CACHEAXI_MONITOR_WRITETHROUGH
  - LL_CACHEAXI_MONITOR_EVICTION
  - LL_CACHEAXI_MONITOR_ALL

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 (RMISSMEN/RHITMEN/WMISSMEN/WHITMEN/RAMMEN/WAMMEN/WTMEN/EVIMEN) LL_CACHEAXI_DisableMonitors

LL_CACHEAXI_IsEnabledMonitors

Function name

__STATIC_INLINE uint32_t LL_CACHEAXI_IsEnabledMonitors (const CACHEAXI_TypeDef * CACHEAXIx, uint32_t Monitors)

Function description

Return the hit/miss monitor(s) enable state.

Parameters

- **CACHEAXIx:** CACHEAXI instance
- **Monitors:** This parameter can be one or a combination of the following values:
  - LL_CACHEAXI_MONITOR_READ_HIT
  - LL_CACHEAXI_MONITOR_READ_MISS
  - LL_CACHEAXI_MONITOR_WRITE_HIT
  - LL_CACHEAXI_MONITOR_WRITE_MISS
  - LL_CACHEAXI_MONITOR_READALLOC_MISS
  - LL_CACHEAXI_MONITOR_WRITEALLOC_MISS
  - LL_CACHEAXI_MONITOR_WRITETHROUGH
  - LL_CACHEAXI_MONITOR_EVICTION
  - LL_CACHEAXI_MONITOR_ALL

Return values

- **State:** of parameter value (1 or 0).

Reference Manual to LL API cross reference:

- CR1 (RMISSMEN/RHITMEN/WMISSMEN/WHITMEN/RAMMEN/WAMMEN/WTMEN/EVIMEN) LL_CACHEAXI_IsEnabledMonitors

LL_CACHEAXI_ResetMonitors

Function name

__STATIC_INLINE void LL_CACHEAXI_ResetMonitors (CACHEAXI_TypeDef * CACHEAXIx, uint32_t Monitors)

Function description

Reset the Data Cache performance monitoring.

Parameters

- **CACHEAXIx:** CACHEAXI instance
- **Monitors:** Monitoring type This parameter can be a combination of the following values:
  - LL_CACHEAXI_MONITOR_READ_HIT
  - LL_CACHEAXI_MONITOR_READ_MISS
  - LL_CACHEAXI_MONITOR_WRITE_HIT
  - LL_CACHEAXI_MONITOR_WRITE_MISS
  - LL_CACHEAXI_MONITOR_READALLOC_MISS
  - LL_CACHEAXI_MONITOR_WRITEALLOC_MISS
  - LL_CACHEAXI_MONITOR_WRITETHROUGH
  - LL_CACHEAXI_MONITOR_EVICTION
  - LL_CACHEAXI_MONITOR_ALL

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 (RHITMRST/RMISSMRST/WHITMRST/WMISSMRST/RAMMRST/WAMMRST/WTMRST/EVIMRST) LL_CACHEAXI_ResetMonitors

LL_CACHEAXI_Monitor_GetReadHitValue

Function name

__STATIC_INLINE uint32_t LL_CACHEAXI_Monitor_GetReadHitValue (CACHEAXI_TypeDef * CACHEAXIx)

Function description

Get the Read Hit monitor Value.

Parameters

- **CACHEAXIx:** CACHEAXI instance

Return values

- **Value:** between Min_Data=0 and Max_Data=0xFFFFFFFF

Reference Manual to LL API cross reference:

- RHMONR LL_CACHEAXI_Monitor_GetReadHitValue

LL_CACHEAXI_Monitor_GetReadMissValue

Function name

__STATIC_INLINE uint32_t LL_CACHEAXI_Monitor_GetReadMissValue (CACHEAXI_TypeDef * CACHEAXIx)

Function description

Get the Read Miss monitor Value.

Parameters

- **CACHEAXIx:** CACHEAXI instance

Return values

- **Value:** between Min_Data=0 and Max_Data=0xFFFF

Reference Manual to LL API cross reference:

- RMMONR LL_CACHEAXI_Monitor_GetReadMissValue

LL_CACHEAXI_Monitor_GetWriteHitValue

Function name

__STATIC_INLINE uint32_t LL_CACHEAXI_Monitor_GetWriteHitValue (CACHEAXI_TypeDef * CACHEAXIx)

Function description

Get the Write Hit monitor Value.

Parameters

- **CACHEAXIx:** CACHEAXI instance

Return values

- **Value:** between Min_Data=0 and Max_Data=0xFFFFFFFF

Reference Manual to LL API cross reference:

- WHMONR LL_CACHEAXI_Monitor_GetWriteHitValue

LL_CACHEAXI_Monitor_GetWriteMissValue

Function name

__STATIC_INLINE uint32_t LL_CACHEAXI_Monitor_GetWriteMissValue (CACHEAXI_TypeDef * CACHEAXIx)

Function description

Get the Write Miss monitor Value.

Parameters

- **CACHEAXIx:** CACHEAXI instance

Return values

- **Value:** between Min_Data=0 and Max_Data=0xFFFF

Reference Manual to LL API cross reference:

- WMMONR LL_CACHEAXI_Monitor_GetWriteMissValue

LL_CACHEAXI_Monitor_GetReadAllocMissValue

Function name

__STATIC_INLINE uint32_t LL_CACHEAXI_Monitor_GetReadAllocMissValue (CACHEAXI_TypeDef * CACHEAXIx)

Function description

Get the Read-allocate Miss monitor Value.

Parameters

- **CACHEAXIx:** CACHEAXI instance

Return values

- **Value:** between Min_Data=0 and Max_Data=0xFFFF

Reference Manual to LL API cross reference:

- WMMONR LL_CACHEAXI_Monitor_GetReadAllocMissValue

LL_CACHEAXI_Monitor_GetWriteAllocMissValue

Function name

__STATIC_INLINE uint32_t LL_CACHEAXI_Monitor_GetWriteAllocMissValue (CACHEAXI_TypeDef * CACHEAXIx)

Function description

Get the Write-allocate Miss monitor Value.

Parameters

- **CACHEAXIx:** CACHEAXI instance

Return values

- **Value:** between Min_Data=0 and Max_Data=0xFFFF

Reference Manual to LL API cross reference:

- WMMONR LL_CACHEAXI_Monitor_GetWriteAllocMissValue

LL_CACHEAXI_Monitor_GetWriteThroughValue

Function name

__STATIC_INLINE uint32_t LL_CACHEAXI_Monitor_GetWriteThroughValue (CACHEAXI_TypeDef * CACHEAXIx)

Function description

Get the Write-through monitor Value.

Parameters

- **CACHEAXIx:** CACHEAXI instance

Return values

- **Value:** between Min_Data=0 and Max_Data=0xFFFF

Reference Manual to LL API cross reference:

- WMMONR LL_CACHEAXI_Monitor_GetWriteThroughValue

LL_CACHEAXI_Monitor_GetEvictionValue

Function name

__STATIC_INLINE uint32_t LL_CACHEAXI_Monitor_GetEvictionValue (CACHEAXI_TypeDef * CACHEAXIx)

Function description

Get the Eviction monitor Value.

Parameters

- **CACHEAXIx:** CACHEAXI instance

Return values

- **Value:** between Min_Data=0 and Max_Data=0xFFFF

Reference Manual to LL API cross reference:

- WMMONR LL_CACHEAXI_Monitor_GetEvictionValue

LL_CACHEAXI_EnableIT_BSYEND

Function name

__STATIC_INLINE void LL_CACHEAXI_EnableIT_BSYEND (CACHEAXI_TypeDef * CACHEAXIx)

Function description

Enable BusyEnd interrupt.

Parameters

- **CACHEAXIx:** CACHEAXI instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER BSYENDIE LL_CACHEAXI_EnableIT_BSYEND

LL_CACHEAXI_DisableIT_BSYEND

Function name

__STATIC_INLINE void LL_CACHEAXI_DisableIT_BSYEND (CACHEAXI_TypeDef * CACHEAXIx)

Function description

Disable BusyEnd interrupt.

Parameters

- **CACHEAXIx:** CACHEAXI instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER BSYENDIE LL_CACHEAXI_DisableIT_BSYEND

LL_CACHEAXI_IsEnabledIT_BSYEND

Function name

__STATIC_INLINE uint32_t LL_CACHEAXI_IsEnabledIT_BSYEND (const CACHEAXI_TypeDef * CACHEAXIx)

Function description

Indicates whether the Busyend interrupt is enabled.

Parameters

- **CACHEAXIx:** CACHEAXI instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER BSYENDIE LL_CACHEAXI_IsEnabledIT_BSYEND

LL_CACHEAXI_EnableIT_ERR

Function name

__STATIC_INLINE void LL_CACHEAXI_EnableIT_ERR (CACHEAXI_TypeDef * CACHEAXIx)

Function description

Enable Error interrupt.

Parameters

- **CACHEAXIx:** CACHEAXI instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER ERRIE LL_CACHEAXI_EnableIT_ERR

LL_CACHEAXI_DisableIT_ERR

Function name

__STATIC_INLINE void LL_CACHEAXI_DisableIT_ERR (CACHEAXI_TypeDef * CACHEAXIx)

Function description

Disable Error interrupt.

Parameters

- **CACHEAXIx:** CACHEAXI instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER ERRIE LL_CACHEAXI_DisableIT_ERR

LL_CACHEAXI_IsEnabledIT_ERR

Function name

__STATIC_INLINE uint32_t LL_CACHEAXI_IsEnabledIT_ERR (const CACHEAXI_TypeDef * CACHEAXIx)

Function description

Indicates whether the Error interrupt is enabled.

Parameters

- **CACHEAXIx:** CACHEAXI instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER ERRIE LL_CACHEAXI_IsEnabledIT_ERR

LL_CACHEAXI_EnableIT_CMDEND

Function name

__STATIC_INLINE void LL_CACHEAXI_EnableIT_CMDEND (CACHEAXI_TypeDef * CACHEAXIx)

Function description

Enable command end interrupt.

Parameters

- **CACHEAXIx:** CACHEAXI instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER CMDENDIE LL_CACHEAXI_EnableIT_CMDEND

LL_CACHEAXI_DisableIT_CMDEND

Function name

__STATIC_INLINE void LL_CACHEAXI_DisableIT_CMDEND (CACHEAXI_TypeDef * CACHEAXIx)

Function description

Disable command end interrupt.

Parameters

- **CACHEAXIx:** CACHEAXI instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER CMDENDIE LL_CACHEAXI_DisableIT_CMDEND

LL_CACHEAXI_IsEnabledIT_CMDEND

Function name

__STATIC_INLINE uint32_t LL_CACHEAXI_IsEnabledIT_CMDEND (const CACHEAXI_TypeDef * CACHEAXIx)

Function description

Indicates whether the command end interrupt is enabled.

Parameters

- **CACHEAXIx:** CACHEAXI instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER CMDENDIE LL_CACHEAXI_IsEnabledIT_CMDEND

LL_CACHEAXI_ClearFlag_BSYEND

Function name

__STATIC_INLINE void LL_CACHEAXI_ClearFlag_BSYEND (CACHEAXI_TypeDef * CACHEAXIx)

Function description

Clear full invalidate busy end flag.

Parameters

- **CACHEAXIx:** CACHEAXI instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- FCR CBSYENDF LL_CACHEAXI_ClearFlag_BSYEND

LL_CACHEAXI_ClearFlag_ERR

Function name

__STATIC_INLINE void LL_CACHEAXI_ClearFlag_ERR (CACHEAXI_TypeDef * CACHEAXIx)

Function description

Clear cache error flag.

Parameters

- **CACHEAXIx:** CACHEAXI instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- FCR CERRF LL_CACHEAXI_ClearFlag_ERR

LL_CACHEAXI_ClearFlag_CMDEND

Function name

__STATIC_INLINE void LL_CACHEAXI_ClearFlag_CMDEND (CACHEAXI_TypeDef * CACHEAXIx)

Function description

Clear command end flag.

Parameters

- **CACHEAXIx:** CACHEAXI instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- FCR CCMDENDF LL_CACHEAXI_ClearFlag_CMDEND

LL_CACHEAXI_IsActiveFlag_BUSY

Function name

__STATIC_INLINE uint32_t LL_CACHEAXI_IsActiveFlag_BUSY (const CACHEAXI_TypeDef * CACHEAXIx)

Function description

Get flag Dcache BUSY.

Parameters

- **CACHEAXIx:** CACHEAXI instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR BUSYF LL_CACHEAXI_IsActiveFlag_BUSY

LL_CACHEAXI_IsActiveFlag_BSYEND

Function name

__STATIC_INLINE uint32_t LL_CACHEAXI_IsActiveFlag_BSYEND (const CACHEAXI_TypeDef * CACHEAXIx)

Function description

Get flag Dcache Busyend.

Parameters

- **CACHEAXIx:** CACHEAXI instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR BSYENDF LL_CACHEAXI_IsActiveFlag_BSYEND

LL_CACHEAXI_IsActiveFlag_ERR

Function name

__STATIC_INLINE uint32_t LL_CACHEAXI_IsActiveFlag_ERR (const CACHEAXI_TypeDef * CACHEAXIx)

Function description

Get flag Dcache Error.

Parameters

- **CACHEAXIx:** CACHEAXI instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR ERRF LL_CACHEAXI_IsActiveFlag_ERR

LL_CACHEAXI_IsActiveFlag_BUSYCMD

Function name

__STATIC_INLINE uint32_t LL_CACHEAXI_IsActiveFlag_BUSYCMD (const CACHEAXI_TypeDef * CACHEAXIx)

Function description

Get flag Dcache Busy command.

Parameters

- **CACHEAXIx:** CACHEAXI instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR BUSYCMDF LL_CACHEAXI_IsActiveFlag_BUSYCMD

LL_CACHEAXI_IsActiveFlag_CMDEND

Function name

__STATIC_INLINE uint32_t LL_CACHEAXI_IsActiveFlag_CMDEND (const CACHEAXI_TypeDef * CACHEAXIx)

Function description

Get flag Dcache command end.

Parameters

- **CACHEAXIx:** CACHEAXI instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR CMDENDF LL_CACHEAXI_IsActiveFlag_CMDEND

87.2 CACHEAXI Firmware driver defines

The following section lists the various define and macros of the module.

87.2.1 CACHEAXI

CACHEAXI

Clear Flags Defines

LL_CACHEAXI_FCR_CERRF

Cache error flag

LL_CACHEAXI_FCR_CBSYENDF

Full invalidate busy end flag

LL_CACHEAXI_FCR_CCMDENDF

Command end flag

Get Flags Defines

LL_CACHEAXI_SR_ERRF

Cache error flag

LL_CACHEAXI_SR_BUSYF

Busy flag

LL_CACHEAXI_SR_CMDENDF

Command end flag

LL_CACHEAXI_SR_BSYENDF

Full invalidate busy end flag

LL_CACHEAXI_SR_BUSYCMDF

Command busy flag

IT Defines

LL_CACHEAXI_IER_BSYENDIE

Busy end interrupt

LL_CACHEAXI_IER_ERRIE

Cache error interrupt

LL_CACHEAXI_IER_CMDENDIE

Command end interrupt

Common write and read registers Macros

LL_CACHEAXI_WriteReg

Description:

- Write a value in CACHEAXI register.

Parameters:

- __INSTANCE__: CACHEAXI Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_CACHEAXI_ReadReg

Description:

- Read a value in CACHEAXI register.

Parameters:

- __INSTANCE__: CACHEAXI Instance
- __REG__: Register to be read

Return value:

- Register: value

88 LL CORTEX Generic Driver

88.1 CORTEX Firmware driver API description

The following section lists the various functions of the CORTEX library.

88.1.1 Detailed description of functions

LL_SYSTICK_IsActiveCounterFlag

Function name

__STATIC_INLINE uint32_t LL_SYSTICK_IsActiveCounterFlag (void)

Function description

This function checks if the Systick counter flag is active or not.

Return values

- **State:** of bit (1 or 0).

Notes

- It can be used in timeout function on application side.

Reference Manual to LL API cross reference:

- STK_CTRL COUNTFLAG LL_SYSTICK_IsActiveCounterFlag

LL_SYSTICK_SetClkSource

Function name

__STATIC_INLINE void LL_SYSTICK_SetClkSource (uint32_t Source)

Function description

Configures the SysTick clock source.

Parameters

- **Source:** This parameter can be one of the following values:
  - LL_SYSTICK_CLKSOURCE_HCLK_DIV8
  - LL_SYSTICK_CLKSOURCE_HCLK

Return values

- **None:**

Reference Manual to LL API cross reference:

- STK_CTRL CLKSOURCE LL_SYSTICK_SetClkSource

LL_SYSTICK_GetClkSource

Function name

__STATIC_INLINE uint32_t LL_SYSTICK_GetClkSource (void)

Function description

Get the SysTick clock source.

Return values

- **Returned:** value can be one of the following values:
  - LL_SYSTICK_CLKSOURCE_HCLK_DIV8
  - LL_SYSTICK_CLKSOURCE_HCLK

Reference Manual to LL API cross reference:

- STK_CTRL CLKSOURCE LL_SYSTICK_GetClkSource

LL_SYSTICK_EnableIT

Function name

__STATIC_INLINE void LL_SYSTICK_EnableIT (void)

Function description

Enable SysTick exception request.

Return values

- **None:**

Reference Manual to LL API cross reference:

- STK_CTRL TICKINT LL_SYSTICK_EnableIT

LL_SYSTICK_DisableIT

Function name

__STATIC_INLINE void LL_SYSTICK_DisableIT (void)

Function description

Disable SysTick exception request.

Return values

- **None:**

Reference Manual to LL API cross reference:

- STK_CTRL TICKINT LL_SYSTICK_DisableIT

LL_SYSTICK_IsEnabledIT

Function name

__STATIC_INLINE uint32_t LL_SYSTICK_IsEnabledIT (void)

Function description

Checks if the SYSTICK interrupt is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- STK_CTRL TICKINT LL_SYSTICK_IsEnabledIT

LL_LPM_EnableSleep

Function name

__STATIC_INLINE void LL_LPM_EnableSleep (void)

Function description

Processor uses sleep as its low power mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SCB_SCR SLEEPDEEP LL_LPM_EnableSleep

LL_LPM_EnableDeepSleep

Function name

__STATIC_INLINE void LL_LPM_EnableDeepSleep (void)

Function description

Processor uses deep sleep as its low power mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SCB_SCR SLEEPDEEP LL_LPM_EnableDeepSleep

LL_LPM_EnableSleepOnExit

Function name

__STATIC_INLINE void LL_LPM_EnableSleepOnExit (void)

Function description

Configures sleep-on-exit when returning from Handler mode to Thread mode.

Return values

- **None:**

Notes

- Setting this bit to 1 enables an interrupt-driven application to avoid returning to an empty main application.

Reference Manual to LL API cross reference:

- SCB_SCR SLEEPONEXIT LL_LPM_EnableSleepOnExit

LL_LPM_DisableSleepOnExit

Function name

__STATIC_INLINE void LL_LPM_DisableSleepOnExit (void)

Function description

Do not sleep when returning to Thread mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SCB_SCR SLEEPONEXIT LL_LPM_DisableSleepOnExit

LL_LPM_EnableEventOnPend

Function name

__STATIC_INLINE void LL_LPM_EnableEventOnPend (void)

Function description

Enabled events and all interrupts, including disabled interrupts, can wakeup the processor.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SCB_SCR SEVEONPEND LL_LPM_EnableEventOnPend

LL_LPM_DisableEventOnPend

Function name

__STATIC_INLINE void LL_LPM_DisableEventOnPend (void)

Function description

Only enabled interrupts or events can wakeup the processor, disabled interrupts are excluded.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SCB_SCR SEVEONPEND LL_LPM_DisableEventOnPend

LL_HANDLER_EnableFault

Function name

__STATIC_INLINE void LL_HANDLER_EnableFault (uint32_t Fault)

Function description

Enable a fault in System handler control register (SHCSR)

Parameters

- **Fault:** This parameter can be a combination of the following values:
  - LL_HANDLER_FAULT_USG
  - LL_HANDLER_FAULT_BUS
  - LL_HANDLER_FAULT_MEM
  - LL_HANDLER_FAULT_SECURE (*)

  (*) value applicable in secure when the system implements the security.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SCB_SHCSR USGFAULTENA LL_HANDLER_EnableFault
- SCB_SHCSR BUSFAULTENA LL_HANDLER_EnableFault
- SCB_SHCSR MEMFAULTENA LL_HANDLER_EnableFault
- SCB_SHCSR SECUREFAULTENA LL_HANDLER_EnableFault

LL_HANDLER_DisableFault

Function name

__STATIC_INLINE void LL_HANDLER_DisableFault (uint32_t Fault)

Function description

Disable a fault in System handler control register (SHCSR)

Parameters

- **Fault:** This parameter can be a combination of the following values:
  - LL_HANDLER_FAULT_USG
  - LL_HANDLER_FAULT_BUS
  - LL_HANDLER_FAULT_MEM
  - LL_HANDLER_FAULT_SECURE (*)

  (*) value applicable in secure when the system implements the security.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SCB_SHCSR USGFAULTENA LL_HANDLER_DisableFault
- SCB_SHCSR BUSFAULTENA LL_HANDLER_DisableFault
- SCB_SHCSR MEMFAULTENA LL_HANDLER_DisableFault
- SCB_SHCSR SECUREFAULTENA LL_HANDLER_DisableFault

LL_CPUID_GetImplementer

Function name

__STATIC_INLINE uint32_t LL_CPUID_GetImplementer (void)

Function description

Get Implementer code.

Return values

- **Value:** should be equal to 0x41 for ARM

Reference Manual to LL API cross reference:

- SCB_CPUID IMPLEMENTER LL_CPUID_GetImplementer

LL_CPUID_GetVariant

Function name

__STATIC_INLINE uint32_t LL_CPUID_GetVariant (void)

Function description

Get Variant number (The r value in the rnpn product revision identifier)

Return values

- **Value:** between 0 and 255 (0x0: revision 0)

Reference Manual to LL API cross reference:

- SCB_CPUID VARIANT LL_CPUID_GetVariant

LL_CPUID_GetArchitecture

Function name

__STATIC_INLINE uint32_t LL_CPUID_GetArchitecture (void)

Function description

Get Architecture version.

Return values

- **Value:** should be equal to 0xF for Cortex-M55 ("ARMv8-M with Main Extension")

Reference Manual to LL API cross reference:

- SCB_CPUID ARCHITECTURE LL_CPUID_GetArchitecture

LL_CPUID_GetParNo

Function name

__STATIC_INLINE uint32_t LL_CPUID_GetParNo (void)

Function description

Get Part number.

Return values

- **Value:** should be equal to 0xD22 for Cortex-M55

Reference Manual to LL API cross reference:

- SCB_CPUID PARTNO LL_CPUID_GetParNo

LL_CPUID_GetRevision

Function name

__STATIC_INLINE uint32_t LL_CPUID_GetRevision (void)

Function description

Get Revision number (The p value in the rnpn product revision identifier, indicates patch release)

Return values

- **Value:** between 0 and 255 (0x1: patch 1)

Reference Manual to LL API cross reference:

- SCB_CPUID REVISION LL_CPUID_GetRevision

LL_MPU_Enable

Function name

__STATIC_INLINE void LL_MPU_Enable (uint32_t MPU_Control)

Function description

Enable MPU with input options.

Parameters

- **MPU_Control:** This parameter can be one of the following values:
  - LL_MPU_CTRL_HFNMI_PRIVDEF_NONE
  - LL_MPU_CTRL_HARDFAULT_NMI
  - LL_MPU_CTRL_PRIVILEGED_DEFAULT
  - LL_MPU_CTRL_HFNMI_PRIVDEF

Return values

- **None:**

Reference Manual to LL API cross reference:

- MPU_CTRL ENABLE LL_MPU_Enable

LL_MPU_Disable

Function name

__STATIC_INLINE void LL_MPU_Disable (void)

Function description

Disable MPU.

Return values

- **None:**

Reference Manual to LL API cross reference:

- MPU_CTRL ENABLE LL_MPU_Disable

LL_MPU_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_MPU_IsEnabled (void)

Function description

Check if MPU is enabled or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- MPU_CTRL ENABLE LL_MPU_IsEnabled

LL_MPU_EnableRegion

Function name

__STATIC_INLINE void LL_MPU_EnableRegion (uint32_t Region)

Function description

Enable a MPU region.

Parameters

- **Region:** This parameter can be one of the following values:
  - LL_MPU_REGION_NUMBER0
  - LL_MPU_REGION_NUMBER1
  - LL_MPU_REGION_NUMBER2
  - LL_MPU_REGION_NUMBER3
  - LL_MPU_REGION_NUMBER4
  - LL_MPU_REGION_NUMBER5
  - LL_MPU_REGION_NUMBER6
  - LL_MPU_REGION_NUMBER7
  - LL_MPU_REGION_NUMBER8
  - LL_MPU_REGION_NUMBER9
  - LL_MPU_REGION_NUMBER10
  - LL_MPU_REGION_NUMBER11
  - LL_MPU_REGION_NUMBER12
  - LL_MPU_REGION_NUMBER13
  - LL_MPU_REGION_NUMBER14
  - LL_MPU_REGION_NUMBER15

Return values

- **None:**

Notes

- Cortex-M55 supports 16 secure and 16 non secure regions.

Reference Manual to LL API cross reference:

- MPU_RLAR EN LL_MPU_EnableRegion

LL_MPU_IsEnabledRegion

Function name

__STATIC_INLINE uint32_t LL_MPU_IsEnabledRegion (uint32_t Region)

Function description

Check if MPU region is enabled or not.

Parameters

- **Region:** This parameter can be one of the following values:
  - LL_MPU_REGION_NUMBER0
  - LL_MPU_REGION_NUMBER1
  - LL_MPU_REGION_NUMBER2
  - LL_MPU_REGION_NUMBER3
  - LL_MPU_REGION_NUMBER4
  - LL_MPU_REGION_NUMBER5
  - LL_MPU_REGION_NUMBER6
  - LL_MPU_REGION_NUMBER7
  - LL_MPU_REGION_NUMBER8
  - LL_MPU_REGION_NUMBER9
  - LL_MPU_REGION_NUMBER10
  - LL_MPU_REGION_NUMBER11
  - LL_MPU_REGION_NUMBER12
  - LL_MPU_REGION_NUMBER13
  - LL_MPU_REGION_NUMBER14
  - LL_MPU_REGION_NUMBER15

Return values

- **State:** of bit (1 or 0).

Notes

- Cortex-M55 supports 16 secure and 16 non secure regions.

Reference Manual to LL API cross reference:

- MPU_RNR EN LL_MPU_IsEnabledRegion

LL_MPU_DisableRegion

Function name

__STATIC_INLINE void LL_MPU_DisableRegion (uint32_t Region)

Function description

Disable a MPU region.

Parameters

- **Region:** This parameter can be one of the following values:
  - LL_MPU_REGION_NUMBER0
  - LL_MPU_REGION_NUMBER1
  - LL_MPU_REGION_NUMBER2
  - LL_MPU_REGION_NUMBER3
  - LL_MPU_REGION_NUMBER4
  - LL_MPU_REGION_NUMBER5
  - LL_MPU_REGION_NUMBER6
  - LL_MPU_REGION_NUMBER7
  - LL_MPU_REGION_NUMBER8
  - LL_MPU_REGION_NUMBER9
  - LL_MPU_REGION_NUMBER10
  - LL_MPU_REGION_NUMBER11
  - LL_MPU_REGION_NUMBER12
  - LL_MPU_REGION_NUMBER13
  - LL_MPU_REGION_NUMBER14
  - LL_MPU_REGION_NUMBER15

Return values

- **None:**

Notes

- Cortex-M55 supports 16 secure and 16 non secure regions.

Reference Manual to LL API cross reference:

- MPU_RNR REGION LL_MPU_DisableRegion
- MPU_RLAR EN LL_MPU_DisableRegion

LL_MPU_ConfigRegion

Function name

__STATIC_INLINE void LL_MPU_ConfigRegion (uint32_t Region, uint32_t Attributes, uint32_t AttrIndx, uint32_t BaseAddress, uint32_t LimitAddress)

Function description

Configure and enable a MPU region.

Parameters

- **Region:** This parameter can be one of the following values:
  - LL_MPU_REGION_NUMBER0
  - LL_MPU_REGION_NUMBER1
  - LL_MPU_REGION_NUMBER2
  - LL_MPU_REGION_NUMBER3
  - LL_MPU_REGION_NUMBER4
  - LL_MPU_REGION_NUMBER5
  - LL_MPU_REGION_NUMBER6
  - LL_MPU_REGION_NUMBER7
  - LL_MPU_REGION_NUMBER8
  - LL_MPU_REGION_NUMBER9
  - LL_MPU_REGION_NUMBER10
  - LL_MPU_REGION_NUMBER11
  - LL_MPU_REGION_NUMBER12
  - LL_MPU_REGION_NUMBER13
  - LL_MPU_REGION_NUMBER14
  - LL_MPU_REGION_NUMBER15
- **Attributes:** This parameter can be a combination of the following values:
  - LL_MPU_INSTRUCTION_ACCESS_ENABLE or LL_MPU_INSTRUCTION_ACCESS_DISABLE
  - LL_MPU_ACCESS_NOT_SHAREABLE or LL_MPU_ACCESS_OUTER_SHAREABLE or LL_MPU_ACCESS_INNER_SHAREABLE
  - LL_MPU_REGION_PRIV_RW or LL_MPU_REGION_ALL_RW or LL_MPU_REGION_PRIV_RO or LL_MPU_REGION_ALL_RO
- **AttrIndx:** This parameter can be one of the following values:
  - LL_MPU_ATTRIBUTES_NUMBER0
  - LL_MPU_ATTRIBUTES_NUMBER1
  - LL_MPU_ATTRIBUTES_NUMBER2
  - LL_MPU_ATTRIBUTES_NUMBER3
  - LL_MPU_ATTRIBUTES_NUMBER4
  - LL_MPU_ATTRIBUTES_NUMBER5
  - LL_MPU_ATTRIBUTES_NUMBER6
  - LL_MPU_ATTRIBUTES_NUMBER7
- **BaseAddress:** Value of region base address
- **LimitAddress:** Value of region limit address

Return values

- **None:**

Notes

- Cortex-M55 supports 16 secure and 16 non secure regions.

Reference Manual to LL API cross reference:

- MPU_RNR REGION LL_MPU_ConfigRegion
- MPU_RBAR BASE LL_MPU_ConfigRegion
- MPU_RLAR LIMIT LL_MPU_ConfigRegion
- MPU_RBAR XN LL_MPU_ConfigRegion
- MPU_RBAR AP LL_MPU_ConfigRegion
- MPU_RBAR SH LL_MPU_ConfigRegion
- MPU_RLAR EN LL_MPU_ConfigRegion
- MPU_RLAR AttrX LL_MPU_ConfigRegion

LL_MPU_ConfigRegionAddress

Function name

__STATIC_INLINE void LL_MPU_ConfigRegionAddress (uint32_t Region, uint32_t BaseAddress, uint32_t LimitAddress)

Function description

Configure a MPU region address range.

Parameters

- **Region:** This parameter can be one of the following values:
  - LL_MPU_REGION_NUMBER0
  - LL_MPU_REGION_NUMBER1
  - LL_MPU_REGION_NUMBER2
  - LL_MPU_REGION_NUMBER3
  - LL_MPU_REGION_NUMBER4
  - LL_MPU_REGION_NUMBER5
  - LL_MPU_REGION_NUMBER6
  - LL_MPU_REGION_NUMBER7
  - LL_MPU_REGION_NUMBER8
  - LL_MPU_REGION_NUMBER9
  - LL_MPU_REGION_NUMBER10
  - LL_MPU_REGION_NUMBER11
  - LL_MPU_REGION_NUMBER12
  - LL_MPU_REGION_NUMBER13
  - LL_MPU_REGION_NUMBER14
  - LL_MPU_REGION_NUMBER15
- **BaseAddress:** Value of region base address
- **LimitAddress:** Value of region limit address

Return values

- **None:**

Notes

- Cortex-M55 supports 16 secure and 16 non secure regions.

Reference Manual to LL API cross reference:

- MPU_RNR REGION LL_MPU_ConfigRegionAddress
- MPU_RBAR BASE LL_MPU_ConfigRegionAddress
- MPU_RLAR LIMIT LL_MPU_ConfigRegionAddress

LL_MPU_ConfigAttributes

Function name

__STATIC_INLINE void LL_MPU_ConfigAttributes (uint32_t AttIndex, uint32_t Attributes)

Function description

Configure a MPU attributes index.

Parameters

- **AttIndex:** This parameter can be one of the following values:
  - LL_MPU_ATTRIBUTES_NUMBER0
  - LL_MPU_ATTRIBUTES_NUMBER1
  - LL_MPU_ATTRIBUTES_NUMBER2
  - LL_MPU_ATTRIBUTES_NUMBER3
  - LL_MPU_ATTRIBUTES_NUMBER4
  - LL_MPU_ATTRIBUTES_NUMBER5
  - LL_MPU_ATTRIBUTES_NUMBER6
  - LL_MPU_ATTRIBUTES_NUMBER7
- **Attributes:** This parameter can be a combination of CORTEX LL MPU Attributes

Return values

- **None:**

Reference Manual to LL API cross reference:

- MPU_MAIR0 AttrX LL_MPU_ConfigAttributes
- MPU_MAIR1 AttrX LL_MPU_ConfigAttributes

LL_MPU_SetRegionLimitAddress

Function name

__STATIC_INLINE void LL_MPU_SetRegionLimitAddress (uint32_t Region, uint32_t LimitAddress)

Function description

Configure a MPU region limit address.

Parameters

- **Region:** This parameter can be one of the following values:
  - LL_MPU_REGION_NUMBER0
  - LL_MPU_REGION_NUMBER1
  - LL_MPU_REGION_NUMBER2
  - LL_MPU_REGION_NUMBER3
  - LL_MPU_REGION_NUMBER4
  - LL_MPU_REGION_NUMBER5
  - LL_MPU_REGION_NUMBER6
  - LL_MPU_REGION_NUMBER7
  - LL_MPU_REGION_NUMBER8
  - LL_MPU_REGION_NUMBER9
  - LL_MPU_REGION_NUMBER10
  - LL_MPU_REGION_NUMBER11
  - LL_MPU_REGION_NUMBER12
  - LL_MPU_REGION_NUMBER13
  - LL_MPU_REGION_NUMBER14
  - LL_MPU_REGION_NUMBER15
- **LimitAddress:** Value of region limit address

Return values

- **None:**

Notes

- Cortex-M55 supports 16 secure and 16 non secure regions.

Reference Manual to LL API cross reference:

- MPU_RNR REGION LL_MPU_SetRegionLimitAddress
- MPU_RLAR LIMIT LL_MPU_SetRegionLimitAddress

LL_MPU_GetRegionLimitAddress

Function name

__STATIC_INLINE uint32_t LL_MPU_GetRegionLimitAddress (uint32_t Region)

Function description

Get a MPU region limit address.

Parameters

- **Region:** This parameter can be one of the following values:
  - LL_MPU_REGION_NUMBER0
  - LL_MPU_REGION_NUMBER1
  - LL_MPU_REGION_NUMBER2
  - LL_MPU_REGION_NUMBER3
  - LL_MPU_REGION_NUMBER4
  - LL_MPU_REGION_NUMBER5
  - LL_MPU_REGION_NUMBER6
  - LL_MPU_REGION_NUMBER7
  - LL_MPU_REGION_NUMBER8
  - LL_MPU_REGION_NUMBER9
  - LL_MPU_REGION_NUMBER10
  - LL_MPU_REGION_NUMBER11
  - LL_MPU_REGION_NUMBER12
  - LL_MPU_REGION_NUMBER13
  - LL_MPU_REGION_NUMBER14
  - LL_MPU_REGION_NUMBER15

Return values

- **MPU:** region limit address

Reference Manual to LL API cross reference:

- MPU_RNR REGION LL_MPU_GetRegionLimitAddress
- MPU_RLAR LIMIT LL_MPU_GetRegionLimitAddress

LL_MPU_SetRegionBaseAddress

Function name

__STATIC_INLINE void LL_MPU_SetRegionBaseAddress (uint32_t Region, uint32_t BaseAddress)

Function description

Configure a MPU region base address.

Parameters

- **Region:** This parameter can be one of the following values:
  - LL_MPU_REGION_NUMBER0
  - LL_MPU_REGION_NUMBER1
  - LL_MPU_REGION_NUMBER2
  - LL_MPU_REGION_NUMBER3
  - LL_MPU_REGION_NUMBER4
  - LL_MPU_REGION_NUMBER5
  - LL_MPU_REGION_NUMBER6
  - LL_MPU_REGION_NUMBER7
  - LL_MPU_REGION_NUMBER8
  - LL_MPU_REGION_NUMBER9
  - LL_MPU_REGION_NUMBER10
  - LL_MPU_REGION_NUMBER11
  - LL_MPU_REGION_NUMBER12
  - LL_MPU_REGION_NUMBER13
  - LL_MPU_REGION_NUMBER14
  - LL_MPU_REGION_NUMBER15
- **BaseAddress:** Value of region base address

Return values

- **None:**

Notes

- Cortex-M55 supports 16 secure and 16 non secure regions.

Reference Manual to LL API cross reference:

- MPU_RNR REGION LL_MPU_SetRegionBaseAddress
- MPU_RBAR BASE LL_MPU_SetRegionBaseAddress

LL_MPU_GetRegionBaseAddress

Function name

__STATIC_INLINE uint32_t LL_MPU_GetRegionBaseAddress (uint32_t Region)

Function description

Get a MPU region base address.

Parameters

- **Region:** This parameter can be one of the following values:
  - LL_MPU_REGION_NUMBER0
  - LL_MPU_REGION_NUMBER1
  - LL_MPU_REGION_NUMBER2
  - LL_MPU_REGION_NUMBER3
  - LL_MPU_REGION_NUMBER4
  - LL_MPU_REGION_NUMBER5
  - LL_MPU_REGION_NUMBER6
  - LL_MPU_REGION_NUMBER7
  - LL_MPU_REGION_NUMBER8
  - LL_MPU_REGION_NUMBER9
  - LL_MPU_REGION_NUMBER10
  - LL_MPU_REGION_NUMBER11
  - LL_MPU_REGION_NUMBER12
  - LL_MPU_REGION_NUMBER13
  - LL_MPU_REGION_NUMBER14
  - LL_MPU_REGION_NUMBER15

Return values

- **MPU:** region base address

Reference Manual to LL API cross reference:

- MPU_RNR REGION LL_MPU_GetRegionBaseAddress
- MPU_RBAR BASE LL_MPU_GetRegionBaseAddress

LL_MPU_SetRegionAccess

Function name

__STATIC_INLINE void LL_MPU_SetRegionAccess (uint32_t Region, uint32_t Attributes)

Function description

Configure a MPU region access attributes and enable a region.

Parameters

- **Region:** This parameter can be one of the following values:
  - LL_MPU_REGION_NUMBER0
  - LL_MPU_REGION_NUMBER1
  - LL_MPU_REGION_NUMBER2
  - LL_MPU_REGION_NUMBER3
  - LL_MPU_REGION_NUMBER4
  - LL_MPU_REGION_NUMBER5
  - LL_MPU_REGION_NUMBER6
  - LL_MPU_REGION_NUMBER7
  - LL_MPU_REGION_NUMBER8
  - LL_MPU_REGION_NUMBER9
  - LL_MPU_REGION_NUMBER10
  - LL_MPU_REGION_NUMBER11
  - LL_MPU_REGION_NUMBER12
  - LL_MPU_REGION_NUMBER13
  - LL_MPU_REGION_NUMBER14
  - LL_MPU_REGION_NUMBER15
- **Attributes:** This parameter can be a combination of the following values:
  - LL_MPU_INSTRUCTION_ACCESS_ENABLE or LL_MPU_INSTRUCTION_ACCESS_DISABLE
  - LL_MPU_ACCESS_NOT_SHAREABLE or LL_MPU_ACCESS_OUTER_SHAREABLE or LL_MPU_ACCESS_INNER_SHAREABLE
  - LL_MPU_REGION_PRIV_RW or LL_MPU_REGION_ALL_RW or LL_MPU_REGION_PRIV_RO or LL_MPU_REGION_ALL_RO

Return values

- **None:**

Notes

- Cortex-M55 supports 16 secure and 16 non secure regions.

Reference Manual to LL API cross reference:

- MPU_RNR REGION LL_MPU_SetRegionAccess
- MPU_RBAR XN LL_MPU_SetRegionAccess
- MPU_RBAR AP LL_MPU_SetRegionAccess
- MPU_RBAR SH LL_MPU_SetRegionAccess

LL_MPU_GetRegionAccess

Function name

__STATIC_INLINE uint32_t LL_MPU_GetRegionAccess (uint32_t Region)

Function description

Get a MPU region access attributes.

Parameters

- **Region:** This parameter can be one of the following values:
  - LL_MPU_REGION_NUMBER0
  - LL_MPU_REGION_NUMBER1
  - LL_MPU_REGION_NUMBER2
  - LL_MPU_REGION_NUMBER3
  - LL_MPU_REGION_NUMBER4
  - LL_MPU_REGION_NUMBER5
  - LL_MPU_REGION_NUMBER6
  - LL_MPU_REGION_NUMBER7
  - LL_MPU_REGION_NUMBER8
  - LL_MPU_REGION_NUMBER9
  - LL_MPU_REGION_NUMBER10
  - LL_MPU_REGION_NUMBER11
  - LL_MPU_REGION_NUMBER12
  - LL_MPU_REGION_NUMBER13
  - LL_MPU_REGION_NUMBER14
  - LL_MPU_REGION_NUMBER15

Return values

- **MPU:** region access attributes

Reference Manual to LL API cross reference:

- MPU_RNR REGION LL_MPU_GetRegionAccess
- MPU_RBAR XN LL_MPU_GetRegionAccess
- MPU_RBAR AP LL_MPU_GetRegionAccess
- MPU_RBAR SH LL_MPU_GetRegionAccess

88.2 CORTEX Firmware driver defines

The following section lists the various define and macros of the module.

88.2.1 CORTEX

CORTEX

CORTEX LL SYSTICK Clock Source

LL_SYSTICK_CLKSOURCE_HCLK_DIV8

AHB clock divided by 8 selected as SysTick clock source

LL_SYSTICK_CLKSOURCE_HCLK

AHB clock selected as SysTick clock source

CORTEX LL Handler Fault type

LL_HANDLER_FAULT_USG

Usage fault

LL_HANDLER_FAULT_BUS

Bus fault

LL_HANDLER_FAULT_MEM

Memory management fault

LL_HANDLER_FAULT_SECURE

Secure fault

CORTEX LL MPU Region Access Attributes

MPU_ACCESS_MSK

CORTEX LL MPU Instruction Access Shareable

LL_MPU_ACCESS_NOT_SHAREABLE

LL_MPU_ACCESS_OUTER_SHAREABLE

LL_MPU_ACCESS_INNER_SHAREABLE

CORTEX LL MPU Attributes

LL_MPU_DEVICE_NGNRNE

LL_MPU_DEVICE_NGNRE

LL_MPU_DEVICE_NGRE

LL_MPU_DEVICE_GRE

LL_MPU_WRITE_THROUGH

LL_MPU_NOT_CACHEABLE

LL_MPU_WRITE_BACK

LL_MPU_TRANSIENT

LL_MPU_NON_TRANSIENT

LL_MPU_NO_ALLOCATE

LL_MPU_W_ALLOCATE

LL_MPU_R_ALLOCATE

LL_MPU_RW_ALLOCATE

CORTEX LL MPU Memory Attributes Index

LL_MPU_ATTRIBUTES_NUMBER0

LL_MPU_ATTRIBUTES_NUMBER1

LL_MPU_ATTRIBUTES_NUMBER2

LL_MPU_ATTRIBUTES_NUMBER3

LL_MPU_ATTRIBUTES_NUMBER4

LL_MPU_ATTRIBUTES_NUMBER5

LL_MPU_ATTRIBUTES_NUMBER6

LL_MPU_ATTRIBUTES_NUMBER7

CORTEX LL MPU HFNMI and PRIVILEGED Access control

LL_MPU_CTRL_HFNMI_PRIVDEF_NONE

Disable NMI and privileged SW access

LL_MPU_CTRL_HARDFAULT_NMI

Enables the operation of MPU during hard fault, NMI, and FAULTMASK handlers

LL_MPU_CTRL_PRIVILEGED_DEFAULT

Enable privileged software access to default memory map

LL_MPU_CTRL_HFNMI_PRIVDEF

Enable NMI and privileged SW access

CORTEX LL MPU Instruction Access

LL_MPU_INSTRUCTION_ACCESS_ENABLE

LL_MPU_INSTRUCTION_ACCESS_DISABLE

CORTEX LL MPU Region Enable

LL_MPU_REGION_ENABLE

LL_MPU_REGION_DISABLE

CORTEX LL MPU Region Index

LL_MPU_REGION_NUMBER0

LL_MPU_REGION_NUMBER1

LL_MPU_REGION_NUMBER2

LL_MPU_REGION_NUMBER3

LL_MPU_REGION_NUMBER4

LL_MPU_REGION_NUMBER5

LL_MPU_REGION_NUMBER6

LL_MPU_REGION_NUMBER7

LL_MPU_REGION_NUMBER8

LL_MPU_REGION_NUMBER9

LL_MPU_REGION_NUMBER10

LL_MPU_REGION_NUMBER11

LL_MPU_REGION_NUMBER12

LL_MPU_REGION_NUMBER13

LL_MPU_REGION_NUMBER14

LL_MPU_REGION_NUMBER15

CORTEX LL MPU Region Permission Attributes

LL_MPU_REGION_PRIV_RW

LL_MPU_REGION_ALL_RW

LL_MPU_REGION_PRIV_RO

LL_MPU_REGION_ALL_RO

89 LL CRC Generic Driver

89.1 CRC Firmware driver API description

The following section lists the various functions of the CRC library.

89.1.1 Detailed description of functions

LL_CRC_ResetCRCCalculationUnit

Function name

__STATIC_INLINE void LL_CRC_ResetCRCCalculationUnit (CRC_TypeDef * CRCx)

Function description

Reset the CRC calculation unit.

Parameters

- **CRCx:** CRC Instance

Return values

- **None:**

Notes

- If Programmable Initial CRC value feature is available, also set the Data Register to the value stored in the CRC_INIT register, otherwise, reset Data Register to its default value.

Reference Manual to LL API cross reference:

- CR RESET LL_CRC_ResetCRCCalculationUnit

LL_CRC_SetPolynomialSize

Function name

__STATIC_INLINE void LL_CRC_SetPolynomialSize (CRC_TypeDef * CRCx, uint32_t PolySize)

Function description

Configure size of the polynomial.

Parameters

- **CRCx:** CRC Instance
- **PolySize:** This parameter can be one of the following values:
  - LL_CRC_POLYLENGTH_32B
  - LL_CRC_POLYLENGTH_16B
  - LL_CRC_POLYLENGTH_8B
  - LL_CRC_POLYLENGTH_7B

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR POLYSIZE LL_CRC_SetPolynomialSize

LL_CRC_GetPolynomialSize

Function name

__STATIC_INLINE uint32_t LL_CRC_GetPolynomialSize (const CRC_TypeDef * CRCx)

Function description

Return size of the polynomial.

Parameters

- **CRCx:** CRC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_CRC_POLYLENGTH_32B
  - LL_CRC_POLYLENGTH_16B
  - LL_CRC_POLYLENGTH_8B
  - LL_CRC_POLYLENGTH_7B

Reference Manual to LL API cross reference:

- CR POLYSIZE LL_CRC_GetPolynomialSize

LL_CRC_SetInputDataReverseType

Function name

__STATIC_INLINE void LL_CRC_SetInputDataReverseType (CRC_TypeDef * CRCx, uint32_t ReverseType)

Function description

Configure the reversal type of the input data.

Parameters

- **CRCx:** CRC Instance
- **ReverseType:** This parameter can be one of the following values:
  - LL_CRC_INDATA_REVERSETYPE_BIT
  - LL_CRC_INDATA_REVERSETYPE_BYTE_HALFWORD

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR RTYPE_IN LL_CRC_SetInputDataReverseType

LL_CRC_GetInputDataReverseType

Function name

__STATIC_INLINE uint32_t LL_CRC_GetInputDataReverseType (CRC_TypeDef * CRCx)

Function description

Return input data type of reversal.

Parameters

- **CRCx:** CRC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_CRC_INDATA_REVERSETYPE_BIT
  - LL_CRC_INDATA_REVERSETYPE_BYTE_HALFWORD

Reference Manual to LL API cross reference:

- CR RTYPE_IN LL_CRC_GetInputDataReverseType

LL_CRC_SetOutputDataReverseType

Function name

__STATIC_INLINE void LL_CRC_SetOutputDataReverseType (CRC_TypeDef * CRCx, uint32_t ReverseType)

Function description

Configure the reversal type of the output data.

Parameters

- **CRCx:** CRC Instance
- **ReverseType:** This parameter can be one of the following values:
  - LL_CRC_OUTDATA_REVERSETYPE_BIT
  - LL_CRC_OUTDATA_REVERSETYPE_BYTE_HALFWORD

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR RTYPE_OUT LL_CRC_SetOutputDataReverseType

LL_CRC_GetOutputDataReverseType

Function name

__STATIC_INLINE uint32_t LL_CRC_GetOutputDataReverseType (CRC_TypeDef * CRCx)

Function description

Return output data type of reversal.

Parameters

- **CRCx:** CRC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_CRC_OUTDATA_REVERSETYPE_BIT
  - LL_CRC_OUTDATA_REVERSETYPE_BYTE_HALFWORD

Reference Manual to LL API cross reference:

- CR RTYPE_OUT LL_CRC_GetOutputDataReverseType

LL_CRC_SetInputDataReverseMode

Function name

__STATIC_INLINE void LL_CRC_SetInputDataReverseMode (CRC_TypeDef * CRCx, uint32_t ReverseMode)

Function description

Configure the reversal of the bit order of the input data.

Parameters

- **CRCx:** CRC Instance
- **ReverseMode:** This parameter can be one of the following values:
  - LL_CRC_INDATA_REVERSE_NONE
  - LL_CRC_INDATA_REVERSE_BIT_BYBYTE
  - LL_CRC_INDATA_REVERSE_HALFWORD_BYWORD
  - LL_CRC_INDATA_REVERSE_BIT_BYHALFWORD
  - LL_CRC_INDATA_REVERSE_BYTE_BYWORD
  - LL_CRC_INDATA_REVERSE_BIT_BYWORD

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR REV_IN LL_CRC_SetInputDataReverseMode

LL_CRC_GetInputDataReverseMode

Function name

__STATIC_INLINE uint32_t LL_CRC_GetInputDataReverseMode (const CRC_TypeDef * CRCx)

Function description

Return mode of reversal for input data bit order.

Parameters

- **CRCx:** CRC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_CRC_INDATA_REVERSE_NONE
  - LL_CRC_INDATA_REVERSE_BIT_BYBYTE
  - LL_CRC_INDATA_REVERSE_HALFWORD_BYWORD
  - LL_CRC_INDATA_REVERSE_BIT_BYHALFWORD
  - LL_CRC_INDATA_REVERSE_BYTE_BYWORD
  - LL_CRC_INDATA_REVERSE_BIT_BYWORD

Reference Manual to LL API cross reference:

- CR REV_IN LL_CRC_GetInputDataReverseMode

LL_CRC_SetOutputDataReverseMode

Function name

__STATIC_INLINE void LL_CRC_SetOutputDataReverseMode (CRC_TypeDef * CRCx, uint32_t ReverseMode)

Function description

Configure the reversal of the bit order of the Output data.

Parameters

- **CRCx:** CRC Instance
- **ReverseMode:** This parameter can be one of the following values:
  - LL_CRC_OUTDATA_REVERSE_NONE
  - LL_CRC_OUTDATA_REVERSE_BIT
  - LL_CRC_OUTDATA_REVERSE_HALFWORD
  - LL_CRC_OUTDATA_REVERSE_BYTE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR REV_OUT LL_CRC_SetOutputDataReverseMode

LL_CRC_GetOutputDataReverseMode

Function name

__STATIC_INLINE uint32_t LL_CRC_GetOutputDataReverseMode (const CRC_TypeDef * CRCx)

Function description

Return mode of reversal of the bit order of the Output data.

Parameters

- **CRCx:** CRC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_CRC_OUTDATA_REVERSE_NONE
  - LL_CRC_OUTDATA_REVERSE_BIT
  - LL_CRC_OUTDATA_REVERSE_HALFWORD
  - LL_CRC_OUTDATA_REVERSE_BYTE

Reference Manual to LL API cross reference:

- CR REV_OUT LL_CRC_GetOutputDataReverseMode

LL_CRC_SetInitialData

Function name

__STATIC_INLINE void LL_CRC_SetInitialData (CRC_TypeDef * CRCx, uint32_t InitCrc)

Function description

Initialize the Programmable initial CRC value.

Parameters

- **CRCx:** CRC Instance
- **InitCrc:** Value to be programmed in Programmable initial CRC value register

Return values

- **None:**

Notes

- If the CRC size is less than 32 bits, the least significant bits are used to write the correct value
- LL_CRC_DEFAULT_CRC_INITVALUE could be used as value for InitCrc parameter.

Reference Manual to LL API cross reference:

- INIT INIT LL_CRC_SetInitialData

LL_CRC_GetInitialData

Function name

__STATIC_INLINE uint32_t LL_CRC_GetInitialData (const CRC_TypeDef * CRCx)

Function description

Return current Initial CRC value.

Parameters

- **CRCx:** CRC Instance

Return values

- **Value:** programmed in Programmable initial CRC value register

Notes

- If the CRC size is less than 32 bits, the least significant bits are used to read the correct value

Reference Manual to LL API cross reference:

- INIT INIT LL_CRC_GetInitialData

LL_CRC_SetPolynomialCoef

Function name

__STATIC_INLINE void LL_CRC_SetPolynomialCoef (CRC_TypeDef * CRCx, uint32_t PolynomCoef)

Function description

Initialize the Programmable polynomial value (coefficients of the polynomial to be used for CRC calculation).

Parameters

- **CRCx:** CRC Instance
- **PolynomCoef:** Value to be programmed in Programmable Polynomial value register

Return values

- **None:**

Notes

- LL_CRC_DEFAULT_CRC32_POLY could be used as value for PolynomCoef parameter.
- Please check Reference Manual and existing Errata Sheets, regarding possible limitations for Polynomial values usage. For example, for a polynomial of degree 7, X^7 + X^6 + X^5 + X^2 + 1 is written 0x65

Reference Manual to LL API cross reference:

- POL POL LL_CRC_SetPolynomialCoef

LL_CRC_GetPolynomialCoef

Function name

__STATIC_INLINE uint32_t LL_CRC_GetPolynomialCoef (const CRC_TypeDef * CRCx)

Function description

Return current Programmable polynomial value.

Parameters

- **CRCx:** CRC Instance

Return values

- **Value:** programmed in Programmable Polynomial value register

Notes

- Please check Reference Manual and existing Errata Sheets, regarding possible limitations for Polynomial values usage. For example, for a polynomial of degree 7, X^7 + X^6 + X^5 + X^2 + 1 is written 0x65

Reference Manual to LL API cross reference:

- POL POL LL_CRC_GetPolynomialCoef

LL_CRC_FeedData32

Function name

__STATIC_INLINE void LL_CRC_FeedData32 (CRC_TypeDef * CRCx, uint32_t InData)

Function description

Write given 32-bit data to the CRC calculator.

Parameters

- **CRCx:** CRC Instance
- **InData:** value to be provided to CRC calculator between between Min_Data=0 and Max_Data=0xFFFFFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- DR DR LL_CRC_FeedData32

LL_CRC_FeedData16

Function name

__STATIC_INLINE void LL_CRC_FeedData16 (CRC_TypeDef * CRCx, uint16_t InData)

Function description

Write given 16-bit data to the CRC calculator.

Parameters

- **CRCx:** CRC Instance
- **InData:** 16 bit value to be provided to CRC calculator between between Min_Data=0 and Max_Data=0xFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- DR DR LL_CRC_FeedData16

LL_CRC_FeedData8

Function name

__STATIC_INLINE void LL_CRC_FeedData8 (CRC_TypeDef * CRCx, uint8_t InData)

Function description

Write given 8-bit data to the CRC calculator.

Parameters

- **CRCx:** CRC Instance
- **InData:** 8 bit value to be provided to CRC calculator between between Min_Data=0 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- DR DR LL_CRC_FeedData8

LL_CRC_ReadData32

Function name

__STATIC_INLINE uint32_t LL_CRC_ReadData32 (const CRC_TypeDef * CRCx)

Function description

Return current CRC calculation result.

Parameters

- **CRCx:** CRC Instance

Return values

- **Current:** CRC calculation result as stored in CRC_DR register (32 bits).

Reference Manual to LL API cross reference:

- DR DR LL_CRC_ReadData32

LL_CRC_ReadData16

Function name

__STATIC_INLINE uint16_t LL_CRC_ReadData16 (const CRC_TypeDef * CRCx)

Function description

Return current CRC calculation result.

Parameters

- **CRCx:** CRC Instance

Return values

- **Current:** CRC calculation result as stored in CRC_DR register (16 bits).

Notes

- This function is expected to be used in a 16 bits CRC polynomial size context.

Reference Manual to LL API cross reference:

- DR DR LL_CRC_ReadData16

LL_CRC_ReadData8

Function name

__STATIC_INLINE uint8_t LL_CRC_ReadData8 (const CRC_TypeDef * CRCx)

Function description

Return current CRC calculation result.

Parameters

- **CRCx:** CRC Instance

Return values

- **Current:** CRC calculation result as stored in CRC_DR register (8 bits).

Notes

- This function is expected to be used in a 8 bits CRC polynomial size context.

Reference Manual to LL API cross reference:

- DR DR LL_CRC_ReadData8

LL_CRC_ReadData7

Function name

__STATIC_INLINE uint8_t LL_CRC_ReadData7 (const CRC_TypeDef * CRCx)

Function description

Return current CRC calculation result.

Parameters

- **CRCx:** CRC Instance

Return values

- **Current:** CRC calculation result as stored in CRC_DR register (7 bits).

Notes

- This function is expected to be used in a 7 bits CRC polynomial size context.

Reference Manual to LL API cross reference:

- DR DR LL_CRC_ReadData7

LL_CRC_Read_IDR

Function name

__STATIC_INLINE uint32_t LL_CRC_Read_IDR (const CRC_TypeDef * CRCx)

Function description

Return data stored in the Independent Data(IDR) register.

Parameters

- **CRCx:** CRC Instance

Return values

- **Value:** stored in CRC_IDR register (General-purpose 32-bit data register).

Notes

- This register can be used as a temporary storage location for one 32-bit long data.

Reference Manual to LL API cross reference:

- IDR IDR LL_CRC_Read_IDR

LL_CRC_Write_IDR

Function name

__STATIC_INLINE void LL_CRC_Write_IDR (CRC_TypeDef * CRCx, uint32_t InData)

Function description

Store data in the Independent Data(IDR) register.

Parameters

- **CRCx:** CRC Instance
- **InData:** value to be stored in CRC_IDR register (32-bit) between Min_Data=0 and Max_Data=0xFFFFFFFF

Return values

- **None:**

Notes

- This register can be used as a temporary storage location for one 32-bit long data.

Reference Manual to LL API cross reference:

- IDR IDR LL_CRC_Write_IDR

LL_CRC_DeInit

Function name

ErrorStatus LL_CRC_DeInit (const CRC_TypeDef * CRCx)

Function description

De-initialize CRC registers (Registers restored to their default values).

Parameters

- **CRCx:** CRC Instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: CRC registers are de-initialized
  - ERROR: CRC registers are not de-initialized

89.2 CRC Firmware driver defines

The following section lists the various define and macros of the module.

89.2.1 CRC

CRC

Default CRC computation initialization value

LL_CRC_DEFAULT_CRC_INITVALUE

Default CRC computation initialization value

Default CRC generating polynomial value

LL_CRC_DEFAULT_CRC32_POLY

Default CRC generating polynomial value

Input Data Reverse

LL_CRC_INDATA_REVERSE_NONE

Input Data bit order not affected

LL_CRC_INDATA_REVERSE_BIT_BYBYTE

Input Data bit reversal done by byte

LL_CRC_INDATA_REVERSE_BYTE

Definition for compatibility with legacy code

LL_CRC_INDATA_REVERSE_HALFWORD_BYWORD

Input Data half-word reversal done by word

LL_CRC_INDATA_REVERSE_BIT_BYHALFWORD

Input Data bit reversal done by half-word

LL_CRC_INDATA_REVERSE_HALFWORD

Definition for compatibility with legacy code

LL_CRC_INDATA_REVERSE_BYTE_BYWORD

Input Data byte reversal done by word

LL_CRC_INDATA_REVERSE_BIT_BYWORD

Input Data bit reversal done by word

LL_CRC_INDATA_REVERSE_WORD

Definition for compatibility with legacy code

Input Reverse Type

LL_CRC_INDATA_REVERSETYPE_BIT

Input Data reverse type at bit level granularity

LL_CRC_INDATA_REVERSETYPE_BYTE_HALFWORD

Input Data reverse type at byte or half-word level granularity

Output Data Reverse

LL_CRC_OUTDATA_REVERSE_NONE

Output Data bit order not affected

LL_CRC_OUTDATA_REVERSE_BIT

Output Data bit reversal done by bit

LL_CRC_OUTDATA_REVERSE_HALFWORD

Output Data half-word reversal done by word

LL_CRC_OUTDATA_REVERSE_BYTE

Output Data byte reversal done by word

Output Reverse Type

LL_CRC_OUTDATA_REVERSETYPE_BIT

Output Data reverse type at bit level granularity

LL_CRC_OUTDATA_REVERSETYPE_BYTE_HALFWORD

Output Data reverse type at byte or half-word level granularity

Polynomial length

LL_CRC_POLYLENGTH_32B

32 bits Polynomial size

LL_CRC_POLYLENGTH_16B

16 bits Polynomial size

LL_CRC_POLYLENGTH_8B

8 bits Polynomial size

LL_CRC_POLYLENGTH_7B

7 bits Polynomial size

Common Write and read registers Macros

LL_CRC_WriteReg

Description:

- Write a value in CRC register.

Parameters:

- __INSTANCE__: CRC Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_CRC_ReadReg

Description:

- Read a value in CRC register.

Parameters:

- __INSTANCE__: CRC Instance
- __REG__: Register to be read

Return value:

- Register: value

90 LL DMA2D Generic Driver

90.1 DMA2D Firmware driver registers structures

90.1.1 LL_DMA2D_InitTypeDef

LL_DMA2D_InitTypeDef is defined in the stm32n6xx_ll_dma2d.h

Data Fields

- ***uint32_t Mode***
- ***uint32_t ColorMode***
- ***uint32_t OutputBlue***
- ***uint32_t OutputGreen***
- ***uint32_t OutputRed***
- ***uint32_t OutputAlpha***
- ***uint32_t OutputMemoryAddress***
- ***uint32_t OutputSwapMode***
- ***uint32_t LineOffsetMode***
- ***uint32_t LineOffset***
- ***uint32_t NbrOfLines***
- ***uint32_t NbrOfPixelsPerLines***
- ***uint32_t AlphaInversionMode***
- ***uint32_t RBSwapMode***

Field Documentation

- ***uint32_t LL_DMA2D_InitTypeDef::Mode***
  Specifies the DMA2D transfer mode.
  - This parameter can be one value of ***DMA2D_LL_EC_MODE***.

  This parameter can be modified afterwards, using unitary function **LL_DMA2D_SetMode()**.
- ***uint32_t LL_DMA2D_InitTypeDef::ColorMode***
  Specifies the color format of the output image.
  - This parameter can be one value of ***DMA2D_LL_EC_OUTPUT_COLOR_MODE***.

  This parameter can be modified afterwards using, unitary function **LL_DMA2D_SetOutputColorMode()**.
- ***uint32_t LL_DMA2D_InitTypeDef::OutputBlue***
  Specifies the Blue value of the output image.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0xFF if ARGB8888 color mode is selected.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0xFF if RGB888 color mode is selected.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0x1F if RGB565 color mode is selected.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0x1F if ARGB1555 color mode is selected.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0x0F if ARGB4444 color mode is selected.

  This parameter can be modified afterwards, using unitary function **LL_DMA2D_SetOutputColor()** or configuration function **LL_DMA2D_ConfigOutputColor()**.

- ***uint32_t LL_DMA2D_InitTypeDef::OutputGreen***
  Specifies the Green value of the output image.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0xFF if ARGB8888 color mode is selected.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0xFF if RGB888 color mode is selected.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0x3F if RGB565 color mode is selected.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0x1F if ARGB1555 color mode is selected.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0x0F if ARGB4444 color mode is selected.

  This parameter can be modified afterwards using unitary function **LL_DMA2D_SetOutputColor()** or configuration function **LL_DMA2D_ConfigOutputColor()**.
- ***uint32_t LL_DMA2D_InitTypeDef::OutputRed***
  Specifies the Red value of the output image.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0xFF if ARGB8888 color mode is selected.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0xFF if RGB888 color mode is selected.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0x1F if RGB565 color mode is selected.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0x1F if ARGB1555 color mode is selected.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0x0F if ARGB4444 color mode is selected.

  This parameter can be modified afterwards using unitary function **LL_DMA2D_SetOutputColor()** or configuration function **LL_DMA2D_ConfigOutputColor()**.
- ***uint32_t LL_DMA2D_InitTypeDef::OutputAlpha***
  Specifies the Alpha channel of the output image.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0xFF if ARGB8888 color mode is selected.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0x01 if ARGB1555 color mode is selected.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0x0F if ARGB4444 color mode is selected.
  - This parameter is not considered if RGB888 or RGB565 color mode is selected.

  This parameter can be modified afterwards using, unitary function **LL_DMA2D_SetOutputColor()** or configuration function **LL_DMA2D_ConfigOutputColor()**.
- ***uint32_t LL_DMA2D_InitTypeDef::OutputMemoryAddress***
  Specifies the memory address.
  - This parameter must be a number between: Min_Data = 0x0000 and Max_Data = 0xFFFFFFFF.

  This parameter can be modified afterwards, using unitary function **LL_DMA2D_SetOutputMemAddr()**.
- ***uint32_t LL_DMA2D_InitTypeDef::OutputSwapMode***
  Specifies the output swap mode color format of the output image.
  - This parameter can be one value of ***DMA2D_LL_EC_OUTPUT_SWAP_MODE***.

  This parameter can be modified afterwards, using unitary function **LL_DMA2D_SetOutputSwapMode()**.
- ***uint32_t LL_DMA2D_InitTypeDef::LineOffsetMode***
  Specifies the output line offset mode.
  - This parameter can be one value of ***DMA2D_LL_EC_LINE_OFFSET_MODE***.

  This parameter can be modified afterwards, using unitary function **LL_DMA2D_SetLineOffsetMode()**.

- ***uint32_t LL_DMA2D_InitTypeDef::LineOffset***
  Specifies the output line offset value.
  - This parameter must be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF.

  This parameter can be modified afterwards, using unitary function **LL_DMA2D_SetLineOffset()**.
- ***uint32_t LL_DMA2D_InitTypeDef::NbrOfLines***
  Specifies the number of lines of the area to be transferred.
  - This parameter must be a number between: Min_Data = 0x0000 and Max_Data = 0xFFFF.

  This parameter can be modified afterwards, using unitary function **LL_DMA2D_SetNbrOfLines()**.
- ***uint32_t LL_DMA2D_InitTypeDef::NbrOfPixelsPerLines***
  Specifies the number of pixels per lines of the area to be transferred.
  - This parameter must be a number between Min_Data = 0x0000 and Max_Data = 0x3FFF.

  This parameter can be modified afterwards using, unitary function **LL_DMA2D_SetNbrOfPixelsPerLines()**.
- ***uint32_t LL_DMA2D_InitTypeDef::AlphaInversionMode***
  Specifies the output alpha inversion mode.
  - This parameter can be one value of ***DMA2D_LL_EC_ALPHA_INVERSION***.

  This parameter can be modified afterwards, using unitary function **LL_DMA2D_SetOutputAlphaInvMode()**.
- ***uint32_t LL_DMA2D_InitTypeDef::RBSwapMode***
  Specifies the output Red Blue swap mode.
  - This parameter can be one value of ***DMA2D_LL_EC_RED_BLUE_SWAP***.

  This parameter can be modified afterwards, using unitary function **LL_DMA2D_SetOutputRBSwapMode()**.

90.1.2 LL_DMA2D_LayerCfgTypeDef

LL_DMA2D_LayerCfgTypeDef is defined in the stm32n6xx_ll_dma2d.h

Data Fields

- ***uint32_t MemoryAddress***
- ***uint32_t LineOffset***
- ***uint32_t ColorMode***
- ***uint32_t CLUTColorMode***
- ***uint32_t CLUTSize***
- ***uint32_t AlphaMode***
- ***uint32_t Alpha***
- ***uint32_t Blue***
- ***uint32_t Green***
- ***uint32_t Red***
- ***uint32_t CLUTMemoryAddress***
- ***uint32_t AlphaInversionMode***
- ***uint32_t RBSwapMode***
- ***uint32_t ChromaSubSampling***

Field Documentation

- ***uint32_t LL_DMA2D_LayerCfgTypeDef::MemoryAddress***
  Specifies the foreground or background memory address.
  - This parameter must be a number between: Min_Data = 0x0000 and Max_Data = 0xFFFFFFFF.

  This parameter can be modified afterwards using unitary functions
  - **LL_DMA2D_FGND_SetMemAddr()** for foreground layer,
  - **LL_DMA2D_BGND_SetMemAddr()** for background layer.

- ***uint32_t LL_DMA2D_LayerCfgTypeDef::LineOffset***
  Specifies the foreground or background line offset value.
  – This parameter must be a number between Min_Data = 0x0000 and Max_Data = 0x3FFF.
  This parameter can be modified afterwards using unitary functions
  – **LL_DMA2D_FGND_SetLineOffset()** for foreground layer,
  – **LL_DMA2D_BGND_SetLineOffset()** for background layer.
- ***uint32_t LL_DMA2D_LayerCfgTypeDef::ColorMode***
  Specifies the foreground or background color mode.
  – This parameter can be one value of ***DMA2D_LL_EC_INPUT_COLOR_MODE***.
  This parameter can be modified afterwards using unitary functions
  – **LL_DMA2D_FGND_SetColorMode()** for foreground layer,
  – **LL_DMA2D_BGND_SetColorMode()** for background layer.
- ***uint32_t LL_DMA2D_LayerCfgTypeDef::CLUTColorMode***
  Specifies the foreground or background CLUT color mode.
  – This parameter can be one value of ***DMA2D_LL_EC_CLUT_COLOR_MODE***.
  This parameter can be modified afterwards using unitary functions
  – **LL_DMA2D_FGND_SetCLUTColorMode()** for foreground layer,
  – **LL_DMA2D_BGND_SetCLUTColorMode()** for background layer.
- ***uint32_t LL_DMA2D_LayerCfgTypeDef::CLUTSize***
  Specifies the foreground or background CLUT size.
  – This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF.
  This parameter can be modified afterwards using unitary functions
  – **LL_DMA2D_FGND_SetCLUTSize()** for foreground layer,
  – **LL_DMA2D_BGND_SetCLUTSize()** for background layer.
- ***uint32_t LL_DMA2D_LayerCfgTypeDef::AlphaMode***
  Specifies the foreground or background alpha mode.
  – This parameter can be one value of ***DMA2D_LL_EC_ALPHA_MODE***.
  This parameter can be modified afterwards using unitary functions
  – **LL_DMA2D_FGND_SetAlphaMode()** for foreground layer,
  – **LL_DMA2D_BGND_SetAlphaMode()** for background layer.
- ***uint32_t LL_DMA2D_LayerCfgTypeDef::Alpha***
  Specifies the foreground or background Alpha value.
  – This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF.
  This parameter can be modified afterwards using unitary functions
  – **LL_DMA2D_FGND_SetAlpha()** for foreground layer,
  – **LL_DMA2D_BGND_SetAlpha()** for background layer.
- ***uint32_t LL_DMA2D_LayerCfgTypeDef::Blue***
  Specifies the foreground or background Blue color value.
  – This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF.
  This parameter can be modified afterwards using unitary functions
  – **LL_DMA2D_FGND_SetBlueColor()** for foreground layer,
  – **LL_DMA2D_BGND_SetBlueColor()** for background layer.
- ***uint32_t LL_DMA2D_LayerCfgTypeDef::Green***
  Specifies the foreground or background Green color value.
  – This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF.
  This parameter can be modified afterwards using unitary functions
  – **LL_DMA2D_FGND_SetGreenColor()** for foreground layer,
  – **LL_DMA2D_BGND_SetGreenColor()** for background layer.

- ***uint32_t LL_DMA2D_LayerCfgTypeDef::Red***
  Specifies the foreground or background Red color value.
  – This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF.
  This parameter can be modified afterwards using unitary functions
  – **LL_DMA2D_FGND_SetRedColor()** for foreground layer,
  – **LL_DMA2D_BGND_SetRedColor()** for background layer.
- ***uint32_t LL_DMA2D_LayerCfgTypeDef::CLUTMemoryAddress***
  Specifies the foreground or background CLUT memory address.
  – This parameter must be a number between: Min_Data = 0x0000 and Max_Data = 0xFFFFFFFF.
  This parameter can be modified afterwards using unitary functions
  – **LL_DMA2D_FGND_SetCLUTMemAddr()** for foreground layer,
  – **LL_DMA2D_BGND_SetCLUTMemAddr()** for background layer.
- ***uint32_t LL_DMA2D_LayerCfgTypeDef::AlphaInversionMode***
  Specifies the foreground or background alpha inversion mode.
  – This parameter can be one value of ***DMA2D_LL_EC_ALPHA_INVERSION***.
  This parameter can be modified afterwards using unitary functions
  – **LL_DMA2D_FGND_SetAlphaInvMode()** for foreground layer,
  – **LL_DMA2D_BGND_SetAlphaInvMode()** for background layer.
- ***uint32_t LL_DMA2D_LayerCfgTypeDef::RBSwapMode***
  Specifies the foreground or background Red Blue swap mode. This parameter can be one value of ***DMA2D_LL_EC_RED_BLUE_SWAP*** .This parameter can be modified afterwards using unitary functions
  – **LL_DMA2D_FGND_SetRBSwapMode()** for foreground layer,
  – **LL_DMA2D_BGND_SetRBSwapMode()** for background layer.
- ***uint32_t LL_DMA2D_LayerCfgTypeDef::ChromaSubSampling***
  Configure the chroma sub-sampling mode for the YCbCr color mode This parameter is applicable for foreground layer only. This parameter can be one value of ***DMA2D_LL_CHROMA_SUB_SAMPLING***This parameter can be modified afterwards using unitary functions
  – **LL_DMA2D_FGND_SetChrSubSampling()** for foreground layer.

90.1.3 LL_DMA2D_ColorTypeDef

LL_DMA2D_ColorTypeDef is defined in the stm32n6xx_ll_dma2d.h

Data Fields

- ***uint32_t ColorMode***
- ***uint32_t OutputBlue***
- ***uint32_t OutputGreen***
- ***uint32_t OutputRed***
- ***uint32_t OutputAlpha***

Field Documentation

- ***uint32_t LL_DMA2D_ColorTypeDef::ColorMode***
  Specifies the color format of the output image.
  – This parameter can be one value of ***DMA2D_LL_EC_OUTPUT_COLOR_MODE***.
  This parameter can be modified afterwards using unitary function **LL_DMA2D_SetOutputColorMode()**.

- ***uint32_t LL_DMA2D_ColorTypeDef::OutputBlue***
  Specifies the Blue value of the output image.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0xFF if ARGB8888 color mode is selected.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0xFF if RGB888 color mode is selected.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0x1F if RGB565 color mode is selected.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0x1F if ARGB1555 color mode is selected.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0x0F if ARGB4444 color mode is selected.

  This parameter can be modified afterwards using, unitary function **LL_DMA2D_SetOutputColor()** or configuration function **LL_DMA2D_ConfigOutputColor()**.
- ***uint32_t LL_DMA2D_ColorTypeDef::OutputGreen***
  Specifies the Green value of the output image.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0xFF if ARGB8888 color mode is selected.
  - This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF if RGB888 color mode is selected.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0x3F if RGB565 color mode is selected.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0x1F if ARGB1555 color mode is selected.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0x0F if ARGB4444 color mode is selected.

  This parameter can be modified afterwards, using unitary function **LL_DMA2D_SetOutputColor()** or configuration function **LL_DMA2D_ConfigOutputColor()**.
- ***uint32_t LL_DMA2D_ColorTypeDef::OutputRed***
  Specifies the Red value of the output image.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0xFF if ARGB8888 color mode is selected.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0xFF if RGB888 color mode is selected.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0x1F if RGB565 color mode is selected.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0x1F if ARGB1555 color mode is selected.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0x0F if ARGB4444 color mode is selected.

  This parameter can be modified afterwards, using unitary function **LL_DMA2D_SetOutputColor()** or configuration function **LL_DMA2D_ConfigOutputColor()**.
- ***uint32_t LL_DMA2D_ColorTypeDef::OutputAlpha***
  Specifies the Alpha channel of the output image.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0xFF if ARGB8888 color mode is selected.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0x01 if ARGB1555 color mode is selected.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0x0F if ARGB4444 color mode is selected.
  - This parameter is not considered if RGB888 or RGB565 color mode is selected.

  This parameter can be modified afterwards, using unitary function **LL_DMA2D_SetOutputColor()** or configuration function **LL_DMA2D_ConfigOutputColor()**.

90.2 DMA2D Firmware driver API description

The following section lists the various functions of the DMA2D library.

90.2.1 Detailed description of functions

LL_DMA2D_Start

Function name

__STATIC_INLINE void LL_DMA2D_Start (DMA2D_TypeDef * DMA2Dx)

Function description

Start a DMA2D transfer.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR START LL_DMA2D_Start

LL_DMA2D_IsTransferOngoing

Function name

__STATIC_INLINE uint32_t LL_DMA2D_IsTransferOngoing (const DMA2D_TypeDef * DMA2Dx)

Function description

Indicate if a DMA2D transfer is ongoing.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR START LL_DMA2D_IsTransferOngoing

LL_DMA2D_Suspend

Function name

__STATIC_INLINE void LL_DMA2D_Suspend (DMA2D_TypeDef * DMA2Dx)

Function description

Suspend DMA2D transfer.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **None:**

Notes

- This API can be used to suspend automatic foreground or background CLUT loading.

Reference Manual to LL API cross reference:

- CR SUSP LL_DMA2D_Suspend

LL_DMA2D_Resume

Function name

__STATIC_INLINE void LL_DMA2D_Resume (DMA2D_TypeDef * DMA2Dx)

Function description

Resume DMA2D transfer.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **None:**

Notes

- This API can be used to resume automatic foreground or background CLUT loading.

Reference Manual to LL API cross reference:

- CR SUSP LL_DMA2D_Resume

LL_DMA2D_IsSuspended

Function name

__STATIC_INLINE uint32_t LL_DMA2D_IsSuspended (const DMA2D_TypeDef * DMA2Dx)

Function description

Indicate if DMA2D transfer is suspended.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **State:** of bit (1 or 0).

Notes

- This API can be used to indicate whether or not automatic foreground or background CLUT loading is suspended.

Reference Manual to LL API cross reference:

- CR SUSP LL_DMA2D_IsSuspended

LL_DMA2D_Abort

Function name

__STATIC_INLINE void LL_DMA2D_Abort (DMA2D_TypeDef * DMA2Dx)

Function description

Abort DMA2D transfer.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **None:**

Notes

- This API can be used to abort automatic foreground or background CLUT loading.

Reference Manual to LL API cross reference:

- CR ABORT LL_DMA2D_Abort

LL_DMA2D_IsAborted

Function name

__STATIC_INLINE uint32_t LL_DMA2D_IsAborted (const DMA2D_TypeDef * DMA2Dx)

Function description

Indicate if DMA2D transfer is aborted.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **State:** of bit (1 or 0).

Notes

- This API can be used to indicate whether or not automatic foreground or background CLUT loading is aborted.

Reference Manual to LL API cross reference:

- CR ABORT LL_DMA2D_IsAborted

LL_DMA2D_SetMode

Function name

__STATIC_INLINE void LL_DMA2D_SetMode (DMA2D_TypeDef * DMA2Dx, uint32_t Mode)

Function description

Set DMA2D mode.

Parameters

- **DMA2Dx:** DMA2D Instance
- **Mode:** This parameter can be one of the following values:
  - LL_DMA2D_MODE_M2M
  - LL_DMA2D_MODE_M2M_PFC
  - LL_DMA2D_MODE_M2M_BLEND
  - LL_DMA2D_MODE_R2M
  - LL_DMA2D_MODE_M2M_BLEND_FIXED_COLOR_FG
  - LL_DMA2D_MODE_M2M_BLEND_FIXED_COLOR_BG

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR MODE LL_DMA2D_SetMode

LL_DMA2D_GetMode

Function name

__STATIC_INLINE uint32_t LL_DMA2D_GetMode (const DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D mode.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA2D_MODE_M2M
  - LL_DMA2D_MODE_M2M_PFC
  - LL_DMA2D_MODE_M2M_BLEND
  - LL_DMA2D_MODE_R2M
  - LL_DMA2D_MODE_M2M_BLEND_FIXED_COLOR_FG
  - LL_DMA2D_MODE_M2M_BLEND_FIXED_COLOR_BG

Reference Manual to LL API cross reference:

- CR MODE LL_DMA2D_GetMode

LL_DMA2D_SetOutputColorMode

Function name

__STATIC_INLINE void LL_DMA2D_SetOutputColorMode (DMA2D_TypeDef * DMA2Dx, uint32_t ColorMode)

Function description

Set DMA2D output color mode.

Parameters

- **DMA2Dx:** DMA2D Instance
- **ColorMode:** This parameter can be one of the following values:
  - LL_DMA2D_OUTPUT_MODE_ARGB8888
  - LL_DMA2D_OUTPUT_MODE_RGB888
  - LL_DMA2D_OUTPUT_MODE_RGB565
  - LL_DMA2D_OUTPUT_MODE_ARGB1555
  - LL_DMA2D_OUTPUT_MODE_ARGB4444

Return values

- **None:**

Reference Manual to LL API cross reference:

- OPFCCR CM LL_DMA2D_SetOutputColorMode

LL_DMA2D_GetOutputColorMode

Function name

__STATIC_INLINE uint32_t LL_DMA2D_GetOutputColorMode (const DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D output color mode.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA2D_OUTPUT_MODE_ARGB8888
  - LL_DMA2D_OUTPUT_MODE_RGB888
  - LL_DMA2D_OUTPUT_MODE_RGB565
  - LL_DMA2D_OUTPUT_MODE_ARGB1555
  - LL_DMA2D_OUTPUT_MODE_ARGB4444

Reference Manual to LL API cross reference:

- OPFCCR CM LL_DMA2D_GetOutputColorMode

LL_DMA2D_SetOutputRBSwapMode

Function name

__STATIC_INLINE void LL_DMA2D_SetOutputRBSwapMode (DMA2D_TypeDef * DMA2Dx, uint32_t RBSwapMode)

Function description

Set DMA2D output Red Blue swap mode.

Parameters

- **DMA2Dx:** DMA2D Instance
- **RBSwapMode:** This parameter can be one of the following values:
  - LL_DMA2D_RB_MODE_REGULAR
  - LL_DMA2D_RB_MODE_SWAP

Return values

- **None:**

Reference Manual to LL API cross reference:

- OPFCCR RBS LL_DMA2D_SetOutputRBSwapMode

LL_DMA2D_GetOutputRBSwapMode

Function name

__STATIC_INLINE uint32_t LL_DMA2D_GetOutputRBSwapMode (const DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D output Red Blue swap mode.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA2D_RB_MODE_REGULAR
  - LL_DMA2D_RB_MODE_SWAP

Reference Manual to LL API cross reference:

- OPFCCR RBS LL_DMA2D_GetOutputRBSwapMode

LL_DMA2D_SetOutputAlphaInvMode

Function name

__STATIC_INLINE void LL_DMA2D_SetOutputAlphaInvMode (DMA2D_TypeDef * DMA2Dx, uint32_t AlphaInversionMode)

Function description

Set DMA2D output alpha inversion mode.

Parameters

- **DMA2Dx:** DMA2D Instance
- **AlphaInversionMode:** This parameter can be one of the following values:
  - LL_DMA2D_ALPHA_REGULAR
  - LL_DMA2D_ALPHA_INVERTED

Return values

- **None:**

Reference Manual to LL API cross reference:

- OPFCCR AI LL_DMA2D_SetOutputAlphaInvMode

LL_DMA2D_GetOutputAlphaInvMode

Function name

__STATIC_INLINE uint32_t LL_DMA2D_GetOutputAlphaInvMode (const DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D output alpha inversion mode.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA2D_ALPHA_REGULAR
  - LL_DMA2D_ALPHA_INVERTED

Reference Manual to LL API cross reference:

- OPFCCR AI LL_DMA2D_GetOutputAlphaInvMode

LL_DMA2D_SetOutputSwapMode

Function name

__STATIC_INLINE void LL_DMA2D_SetOutputSwapMode (DMA2D_TypeDef * DMA2Dx, uint32_t OutputSwapMode)

Function description

Set DMA2D output swap mode.

Parameters

- **DMA2Dx:** DMA2D Instance
- **OutputSwapMode:** This parameter can be one of the following values:
  - LL_DMA2D_SWAP_MODE_REGULAR
  - LL_DMA2D_SWAP_MODE_TWO_BY_TWO

Return values

- **None:**

Reference Manual to LL API cross reference:

- OPFCCR SB LL_DMA2D_SetOutputSwapMode

LL_DMA2D_GetOutputSwapMode

Function name

__STATIC_INLINE uint32_t LL_DMA2D_GetOutputSwapMode (const DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D output swap mode.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA2D_SWAP_MODE_REGULAR
  - LL_DMA2D_SWAP_MODE_TWO_BY_TWO

Reference Manual to LL API cross reference:

- OPFCCR SB LL_DMA2D_GetOutputSwapMode

LL_DMA2D_SetLineOffsetMode

Function name

__STATIC_INLINE void LL_DMA2D_SetLineOffsetMode (DMA2D_TypeDef * DMA2Dx, uint32_t LineOffsetMode)

Function description

Set DMA2D line offset mode.

Parameters

- **DMA2Dx:** DMA2D Instance
- **LineOffsetMode:** This parameter can be one of the following values:
  - LL_DMA2D_LINE_OFFSET_PIXELS
  - LL_DMA2D_LINE_OFFSET_BYTES

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR LOM LL_DMA2D_SetLineOffsetMode

LL_DMA2D_GetLineOffsetMode

Function name

__STATIC_INLINE uint32_t LL_DMA2D_GetLineOffsetMode (const DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D line offset mode.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA2D_LINE_OFFSET_PIXELS
  - LL_DMA2D_LINE_OFFSET_BYTES

Reference Manual to LL API cross reference:

- CR LOM LL_DMA2D_GetLineOffsetMode

LL_DMA2D_SetLineOffset

Function name

__STATIC_INLINE void LL_DMA2D_SetLineOffset (DMA2D_TypeDef * DMA2Dx, uint32_t LineOffset)

Function description

Set DMA2D line offset, expressed on 14 bits ([13:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance
- **LineOffset:** Value between Min_Data=0 and Max_Data=0xFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- OOR LO LL_DMA2D_SetLineOffset

LL_DMA2D_GetLineOffset

Function name

__STATIC_INLINE uint32_t LL_DMA2D_GetLineOffset (const DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D line offset, expressed on 14 bits ([13:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Line:** offset value between Min_Data=0 and Max_Data=0xFFFF

Reference Manual to LL API cross reference:

- OOR LO LL_DMA2D_GetLineOffset

LL_DMA2D_SetNbrOfPixelsPerLines

Function name

__STATIC_INLINE void LL_DMA2D_SetNbrOfPixelsPerLines (DMA2D_TypeDef * DMA2Dx, uint32_t NbrOfPixelsPerLines)

Function description

Set DMA2D number of pixels per lines, expressed on 14 bits ([13:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance
- **NbrOfPixelsPerLines:** Value between Min_Data=0 and Max_Data=0x3FFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- NLR PL LL_DMA2D_SetNbrOfPixelsPerLines

LL_DMA2D_GetNbrOfPixelsPerLines

Function name

__STATIC_INLINE uint32_t LL_DMA2D_GetNbrOfPixelsPerLines (const DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D number of pixels per lines, expressed on 14 bits ([13:0] bits)

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Number:** of pixels per lines value between Min_Data=0 and Max_Data=0x3FFF

Reference Manual to LL API cross reference:

- NLR PL LL_DMA2D_GetNbrOfPixelsPerLines

LL_DMA2D_SetNbrOfLines

Function name

__STATIC_INLINE void LL_DMA2D_SetNbrOfLines (DMA2D_TypeDef * DMA2Dx, uint32_t NbrOfLines)

Function description

Set DMA2D number of lines, expressed on 16 bits ([15:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance
- **NbrOfLines:** Value between Min_Data=0 and Max_Data=0xFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- NLR NL LL_DMA2D_SetNbrOfLines

LL_DMA2D_GetNbrOfLines

Function name

__STATIC_INLINE uint32_t LL_DMA2D_GetNbrOfLines (const DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D number of lines, expressed on 16 bits ([15:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Number:** of lines value between Min_Data=0 and Max_Data=0xFFFF

Reference Manual to LL API cross reference:

- NLR NL LL_DMA2D_GetNbrOfLines

LL_DMA2D_SetOutputMemAddr

Function name

__STATIC_INLINE void LL_DMA2D_SetOutputMemAddr (DMA2D_TypeDef * DMA2Dx, uint32_t OutputMemoryAddress)

Function description

Set DMA2D output memory address, expressed on 32 bits ([31:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance
- **OutputMemoryAddress:** Value between Min_Data=0 and Max_Data=0xFFFFFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- OMAR MA LL_DMA2D_SetOutputMemAddr

LL_DMA2D_GetOutputMemAddr

Function name

__STATIC_INLINE uint32_t LL_DMA2D_GetOutputMemAddr (const DMA2D_TypeDef * DMA2Dx)

Function description

Get DMA2D output memory address, expressed on 32 bits ([31:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Output:** memory address value between Min_Data=0 and Max_Data=0xFFFFFFFF

Reference Manual to LL API cross reference:

- OMAR MA LL_DMA2D_GetOutputMemAddr

LL_DMA2D_SetOutputColor

Function name

__STATIC_INLINE void LL_DMA2D_SetOutputColor (DMA2D_TypeDef * DMA2Dx, uint32_t OutputColor)

Function description

Set DMA2D output color, expressed on 32 bits ([31:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance
- **OutputColor:** Value between Min_Data=0 and Max_Data=0xFFFFFFFF

Return values

- **None:**

Notes

- Output color format depends on output color mode, ARGB8888, RGB888, RGB565, ARGB1555 or ARGB4444.
- LL_DMA2D_ConfigOutputColor() API may be used instead if colors values formatting with respect to color mode is not done by the user code.

Reference Manual to LL API cross reference:

- OCOLR BLUE LL_DMA2D_SetOutputColor
- OCOLR GREEN LL_DMA2D_SetOutputColor
- OCOLR RED LL_DMA2D_SetOutputColor
- OCOLR ALPHA LL_DMA2D_SetOutputColor

LL_DMA2D_GetOutputColor

Function name

__STATIC_INLINE uint32_t LL_DMA2D_GetOutputColor (const DMA2D_TypeDef * DMA2Dx)

Function description

Get DMA2D output color, expressed on 32 bits ([31:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Output:** color value between Min_Data=0 and Max_Data=0xFFFFFFFF

Notes

- Alpha channel and red, green, blue color values must be retrieved from the returned value based on the output color mode (ARGB8888, RGB888, RGB565, ARGB1555 or ARGB4444) as set by LL_DMA2D_SetOutputColorMode.

Reference Manual to LL API cross reference:

- OCOLR BLUE LL_DMA2D_GetOutputColor
- OCOLR GREEN LL_DMA2D_GetOutputColor
- OCOLR RED LL_DMA2D_GetOutputColor
- OCOLR ALPHA LL_DMA2D_GetOutputColor

LL_DMA2D_SetLineWatermark

Function name

__STATIC_INLINE void LL_DMA2D_SetLineWatermark (DMA2D_TypeDef * DMA2Dx, uint32_t LineWatermark)

Function description

Set DMA2D line watermark, expressed on 16 bits ([15:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance
- **LineWatermark:** Value between Min_Data=0 and Max_Data=0xFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- LWR LW LL_DMA2D_SetLineWatermark

LL_DMA2D_GetLineWatermark

Function name

__STATIC_INLINE uint32_t LL_DMA2D_GetLineWatermark (const DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D line watermark, expressed on 16 bits ([15:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Line:** watermark value between Min_Data=0 and Max_Data=0xFFFF

Reference Manual to LL API cross reference:

- LWR LW LL_DMA2D_GetLineWatermark

LL_DMA2D_SetDeadTime

Function name

__STATIC_INLINE void LL_DMA2D_SetDeadTime (DMA2D_TypeDef * DMA2Dx, uint32_t DeadTime)

Function description

Set DMA2D dead time, expressed on 8 bits ([7:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance
- **DeadTime:** Value between Min_Data=0 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- AMTCR DT LL_DMA2D_SetDeadTime

LL_DMA2D_GetDeadTime

Function name

__STATIC_INLINE uint32_t LL_DMA2D_GetDeadTime (const DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D dead time, expressed on 8 bits ([7:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Dead:** time value between Min_Data=0 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- AMTCR DT LL_DMA2D_GetDeadTime

LL_DMA2D_EnableDeadTime

Function name

__STATIC_INLINE void LL_DMA2D_EnableDeadTime (DMA2D_TypeDef * DMA2Dx)

Function description

Enable DMA2D dead time functionality.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- AMTCR EN LL_DMA2D_EnableDeadTime

LL_DMA2D_DisableDeadTime

Function name

__STATIC_INLINE void LL_DMA2D_DisableDeadTime (DMA2D_TypeDef * DMA2Dx)

Function description

Disable DMA2D dead time functionality.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- AMTCR EN LL_DMA2D_DisableDeadTime

LL_DMA2D_IsEnabledDeadTime

Function name

__STATIC_INLINE uint32_t LL_DMA2D_IsEnabledDeadTime (const DMA2D_TypeDef * DMA2Dx)

Function description

Indicate if DMA2D dead time functionality is enabled.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- AMTCR EN LL_DMA2D_IsEnabledDeadTime

LL_DMA2D_FGND_SetMemAddr

Function name

__STATIC_INLINE void LL_DMA2D_FGND_SetMemAddr (DMA2D_TypeDef * DMA2Dx, uint32_t MemoryAddress)

Function description

Set DMA2D foreground memory address, expressed on 32 bits ([31:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance
- **MemoryAddress:** Value between Min_Data=0 and Max_Data=0xFFFFFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- FGMAR MA LL_DMA2D_FGND_SetMemAddr

LL_DMA2D_FGND_GetMemAddr

Function name

__STATIC_INLINE uint32_t LL_DMA2D_FGND_GetMemAddr (const DMA2D_TypeDef * DMA2Dx)

Function description

Get DMA2D foreground memory address, expressed on 32 bits ([31:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Foreground:** memory address value between Min_Data=0 and Max_Data=0xFFFFFFFF

Reference Manual to LL API cross reference:

- FGMAR MA LL_DMA2D_FGND_GetMemAddr

LL_DMA2D_FGND_EnableCLUTLoad

Function name

__STATIC_INLINE void LL_DMA2D_FGND_EnableCLUTLoad (DMA2D_TypeDef * DMA2Dx)

Function description

Enable DMA2D foreground CLUT loading.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- FGPFCCR START LL_DMA2D_FGND_EnableCLUTLoad

LL_DMA2D_FGND_IsEnabledCLUTLoad

Function name

__STATIC_INLINE uint32_t LL_DMA2D_FGND_IsEnabledCLUTLoad (const DMA2D_TypeDef * DMA2Dx)

Function description

Indicate if DMA2D foreground CLUT loading is enabled.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- FGPFCCR START LL_DMA2D_FGND_IsEnabledCLUTLoad

LL_DMA2D_FGND_SetColorMode

Function name

__STATIC_INLINE void LL_DMA2D_FGND_SetColorMode (DMA2D_TypeDef * DMA2Dx, uint32_t ColorMode)

Function description

Set DMA2D foreground color mode.

Parameters

- **DMA2Dx:** DMA2D Instance
- **ColorMode:** This parameter can be one of the following values:
  - LL_DMA2D_INPUT_MODE_ARGB8888
  - LL_DMA2D_INPUT_MODE_RGB888
  - LL_DMA2D_INPUT_MODE_RGB565
  - LL_DMA2D_INPUT_MODE_ARGB1555
  - LL_DMA2D_INPUT_MODE_ARGB4444
  - LL_DMA2D_INPUT_MODE_L8
  - LL_DMA2D_INPUT_MODE_AL44
  - LL_DMA2D_INPUT_MODE_AL88
  - LL_DMA2D_INPUT_MODE_L4
  - LL_DMA2D_INPUT_MODE_A8
  - LL_DMA2D_INPUT_MODE_A4

Return values

- **None:**

Reference Manual to LL API cross reference:

- FGPFCCR CM LL_DMA2D_FGND_SetColorMode

LL_DMA2D_FGND_GetColorMode

Function name

__STATIC_INLINE uint32_t LL_DMA2D_FGND_GetColorMode (const DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D foreground color mode.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA2D_INPUT_MODE_ARGB8888
  - LL_DMA2D_INPUT_MODE_RGB888
  - LL_DMA2D_INPUT_MODE_RGB565
  - LL_DMA2D_INPUT_MODE_ARGB1555
  - LL_DMA2D_INPUT_MODE_ARGB4444
  - LL_DMA2D_INPUT_MODE_L8
  - LL_DMA2D_INPUT_MODE_AL44
  - LL_DMA2D_INPUT_MODE_AL88
  - LL_DMA2D_INPUT_MODE_L4
  - LL_DMA2D_INPUT_MODE_A8
  - LL_DMA2D_INPUT_MODE_A4

Reference Manual to LL API cross reference:

- FGPFCCR CM LL_DMA2D_FGND_GetColorMode

LL_DMA2D_FGND_SetAlphaMode

Function name

__STATIC_INLINE void LL_DMA2D_FGND_SetAlphaMode (DMA2D_TypeDef * DMA2Dx, uint32_t AphaMode)

Function description

Set DMA2D foreground alpha mode.

Parameters

- **DMA2Dx:** DMA2D Instance
- **AphaMode:** This parameter can be one of the following values:
  - LL_DMA2D_ALPHA_MODE_NO_MODIF
  - LL_DMA2D_ALPHA_MODE_REPLACE
  - LL_DMA2D_ALPHA_MODE_COMBINE

Return values

- **None:**

Reference Manual to LL API cross reference:

- FGPFCCR AM LL_DMA2D_FGND_SetAlphaMode

LL_DMA2D_FGND_GetAlphaMode

Function name

__STATIC_INLINE uint32_t LL_DMA2D_FGND_GetAlphaMode (const DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D foreground alpha mode.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA2D_ALPHA_MODE_NO_MODIF
  - LL_DMA2D_ALPHA_MODE_REPLACE
  - LL_DMA2D_ALPHA_MODE_COMBINE

Reference Manual to LL API cross reference:

- FGPFCCR AM LL_DMA2D_FGND_GetAlphaMode

LL_DMA2D_FGND_SetAlpha

Function name

__STATIC_INLINE void LL_DMA2D_FGND_SetAlpha (DMA2D_TypeDef * DMA2Dx, uint32_t Alpha)

Function description

Set DMA2D foreground alpha value, expressed on 8 bits ([7:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance
- **Alpha:** Value between Min_Data=0 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- FGPFCCR ALPHA LL_DMA2D_FGND_SetAlpha

LL_DMA2D_FGND_GetAlpha

Function name

__STATIC_INLINE uint32_t LL_DMA2D_FGND_GetAlpha (const DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D foreground alpha value, expressed on 8 bits ([7:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Alpha:** value between Min_Data=0 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- FGPFCCR ALPHA LL_DMA2D_FGND_GetAlpha

LL_DMA2D_FGND_SetRBSwapMode

Function name

__STATIC_INLINE void LL_DMA2D_FGND_SetRBSwapMode (DMA2D_TypeDef * DMA2Dx, uint32_t RBSwapMode)

Function description

Set DMA2D foreground Red Blue swap mode.

Parameters

- **DMA2Dx:** DMA2D Instance
- **RBSwapMode:** This parameter can be one of the following values:
  - LL_DMA2D_RB_MODE_REGULAR
  - LL_DMA2D_RB_MODE_SWAP

Return values

- **None:**

Reference Manual to LL API cross reference:

- FGPFCCR RBS LL_DMA2D_FGND_SetRBSwapMode

LL_DMA2D_FGND_GetRBSwapMode

Function name

__STATIC_INLINE uint32_t LL_DMA2D_FGND_GetRBSwapMode (const DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D foreground Red Blue swap mode.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA2D_RB_MODE_REGULAR
  - LL_DMA2D_RB_MODE_SWAP

Reference Manual to LL API cross reference:

- FGPFCCR RBS LL_DMA2D_FGND_GetRBSwapMode

LL_DMA2D_FGND_SetAlphaInvMode

Function name

__STATIC_INLINE void LL_DMA2D_FGND_SetAlphaInvMode (DMA2D_TypeDef * DMA2Dx, uint32_t AlphaInversionMode)

Function description

Set DMA2D foreground alpha inversion mode.

Parameters

- **DMA2Dx:** DMA2D Instance
- **AlphaInversionMode:** This parameter can be one of the following values:
  - LL_DMA2D_ALPHA_REGULAR
  - LL_DMA2D_ALPHA_INVERTED

Return values

- **None:**

Reference Manual to LL API cross reference:

- FGPFCCR AI LL_DMA2D_FGND_SetAlphaInvMode

LL_DMA2D_FGND_GetAlphaInvMode

Function name

__STATIC_INLINE uint32_t LL_DMA2D_FGND_GetAlphaInvMode (const DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D foreground alpha inversion mode.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA2D_ALPHA_REGULAR
  - LL_DMA2D_ALPHA_INVERTED

Reference Manual to LL API cross reference:

- FGPFCCR AI LL_DMA2D_FGND_GetAlphaInvMode

LL_DMA2D_FGND_SetLineOffset

Function name

__STATIC_INLINE void LL_DMA2D_FGND_SetLineOffset (DMA2D_TypeDef * DMA2Dx, uint32_t LineOffset)

Function description

Set DMA2D foreground line offset, expressed on 14 bits ([13:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance
- **LineOffset:** Value between Min_Data=0 and Max_Data=0x3FF

Return values

- **None:**

Reference Manual to LL API cross reference:

- FGOR LO LL_DMA2D_FGND_SetLineOffset

LL_DMA2D_FGND_GetLineOffset

Function name

__STATIC_INLINE uint32_t LL_DMA2D_FGND_GetLineOffset (const DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D foreground line offset, expressed on 14 bits ([13:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Foreground:** line offset value between Min_Data=0 and Max_Data=0x3FF

Reference Manual to LL API cross reference:

- FGOR LO LL_DMA2D_FGND_GetLineOffset

LL_DMA2D_FGND_SetColor

Function name

__STATIC_INLINE void LL_DMA2D_FGND_SetColor (DMA2D_TypeDef * DMA2Dx, uint32_t Red, uint32_t Green, uint32_t Blue)

Function description

Set DMA2D foreground color values, expressed on 24 bits ([23:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance
- **Red:** Value between Min_Data=0 and Max_Data=0xFF
- **Green:** Value between Min_Data=0 and Max_Data=0xFF
- **Blue:** Value between Min_Data=0 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- FGCOLR RED LL_DMA2D_FGND_SetColor
- FGCOLR GREEN LL_DMA2D_FGND_SetColor
- FGCOLR BLUE LL_DMA2D_FGND_SetColor

LL_DMA2D_FGND_SetRedColor

Function name

__STATIC_INLINE void LL_DMA2D_FGND_SetRedColor (DMA2D_TypeDef * DMA2Dx, uint32_t Red)

Function description

Set DMA2D foreground red color value, expressed on 8 bits ([7:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance
- **Red:** Value between Min_Data=0 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- FGCOLR RED LL_DMA2D_FGND_SetRedColor

LL_DMA2D_FGND_GetRedColor

Function name

__STATIC_INLINE uint32_t LL_DMA2D_FGND_GetRedColor (const DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D foreground red color value, expressed on 8 bits ([7:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Red:** color value between Min_Data=0 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- FGCOLR RED LL_DMA2D_FGND_GetRedColor

LL_DMA2D_FGND_SetGreenColor

Function name

__STATIC_INLINE void LL_DMA2D_FGND_SetGreenColor (DMA2D_TypeDef * DMA2Dx, uint32_t Green)

Function description

Set DMA2D foreground green color value, expressed on 8 bits ([7:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance
- **Green:** Value between Min_Data=0 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- FGCOLR GREEN LL_DMA2D_FGND_SetGreenColor

LL_DMA2D_FGND_GetGreenColor

Function name

__STATIC_INLINE uint32_t LL_DMA2D_FGND_GetGreenColor (const DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D foreground green color value, expressed on 8 bits ([7:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Green:** color value between Min_Data=0 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- FGCOLR GREEN LL_DMA2D_FGND_GetGreenColor

LL_DMA2D_FGND_SetBlueColor

Function name

__STATIC_INLINE void LL_DMA2D_FGND_SetBlueColor (DMA2D_TypeDef * DMA2Dx, uint32_t Blue)

Function description

Set DMA2D foreground blue color value, expressed on 8 bits ([7:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance
- **Blue:** Value between Min_Data=0 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- FGCOLR BLUE LL_DMA2D_FGND_SetBlueColor

LL_DMA2D_FGND_GetBlueColor

Function name

__STATIC_INLINE uint32_t LL_DMA2D_FGND_GetBlueColor (const DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D foreground blue color value, expressed on 8 bits ([7:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Blue:** color value between Min_Data=0 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- FGCOLR BLUE LL_DMA2D_FGND_GetBlueColor

LL_DMA2D_FGND_SetCLUTMemAddr

Function name

__STATIC_INLINE void LL_DMA2D_FGND_SetCLUTMemAddr (DMA2D_TypeDef * DMA2Dx, uint32_t CLUTMemoryAddress)

Function description

Set DMA2D foreground CLUT memory address, expressed on 32 bits ([31:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance
- **CLUTMemoryAddress:** Value between Min_Data=0 and Max_Data=0xFFFFFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- FGCMAR MA LL_DMA2D_FGND_SetCLUTMemAddr

LL_DMA2D_FGND_GetCLUTMemAddr

Function name

__STATIC_INLINE uint32_t LL_DMA2D_FGND_GetCLUTMemAddr (const DMA2D_TypeDef * DMA2Dx)

Function description

Get DMA2D foreground CLUT memory address, expressed on 32 bits ([31:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Foreground:** CLUT memory address value between Min_Data=0 and Max_Data=0xFFFFFFFF

Reference Manual to LL API cross reference:

- FGCMAR MA LL_DMA2D_FGND_GetCLUTMemAddr

LL_DMA2D_FGND_SetCLUTSize

Function name

__STATIC_INLINE void LL_DMA2D_FGND_SetCLUTSize (DMA2D_TypeDef * DMA2Dx, uint32_t CLUTSize)

Function description

Set DMA2D foreground CLUT size, expressed on 8 bits ([7:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance
- **CLUTSize:** Value between Min_Data=0 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- FGPFCCR CS LL_DMA2D_FGND_SetCLUTSize

LL_DMA2D_FGND_GetCLUTSize

Function name

__STATIC_INLINE uint32_t LL_DMA2D_FGND_GetCLUTSize (const DMA2D_TypeDef * DMA2Dx)

Function description

Get DMA2D foreground CLUT size, expressed on 8 bits ([7:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Foreground:** CLUT size value between Min_Data=0 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- FGPFCCR CS LL_DMA2D_FGND_GetCLUTSize

LL_DMA2D_FGND_SetCLUTColorMode

Function name

__STATIC_INLINE void LL_DMA2D_FGND_SetCLUTColorMode (DMA2D_TypeDef * DMA2Dx, uint32_t CLUTColorMode)

Function description

Set DMA2D foreground CLUT color mode.

Parameters

- **DMA2Dx:** DMA2D Instance
- **CLUTColorMode:** This parameter can be one of the following values:
  - LL_DMA2D_CLUT_COLOR_MODE_ARGB8888
  - LL_DMA2D_CLUT_COLOR_MODE_RGB888

Return values

- **None:**

Reference Manual to LL API cross reference:

- FGPFCCR CCM LL_DMA2D_FGND_SetCLUTColorMode

LL_DMA2D_FGND_GetCLUTColorMode

Function name

__STATIC_INLINE uint32_t LL_DMA2D_FGND_GetCLUTColorMode (const DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D foreground CLUT color mode.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA2D_CLUT_COLOR_MODE_ARGB8888
  - LL_DMA2D_CLUT_COLOR_MODE_RGB888

Reference Manual to LL API cross reference:

- FGPFCCR CCM LL_DMA2D_FGND_GetCLUTColorMode

LL_DMA2D_FGND_SetChrSubSampling

Function name

__STATIC_INLINE void LL_DMA2D_FGND_SetChrSubSampling (DMA2D_TypeDef * DMA2Dx, uint32_t ChromaSubSampling)

Function description

Set DMA2D foreground Chroma Sub Sampling (for YCbCr input color mode).

Parameters

- **DMA2Dx:** DMA2D Instance
- **ChromaSubSampling:** This parameter can be one of the following values:
  - LL_DMA2D_CSS_444
  - LL_DMA2D_CSS_422
  - LL_DMA2D_CSS_420

Return values

- **None:**

Reference Manual to LL API cross reference:

- FGPFCCR CSS LL_DMA2D_FGND_SetChrSubSampling

LL_DMA2D_FGND_GetChrSubSampling

Function name

__STATIC_INLINE uint32_t LL_DMA2D_FGND_GetChrSubSampling (const DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D foreground Chroma Sub Sampling (for YCbCr input color mode).

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA2D_CSS_444
  - LL_DMA2D_CSS_422
  - LL_DMA2D_CSS_420

Reference Manual to LL API cross reference:

- FGPFCCR CSS LL_DMA2D_FGND_GetChrSubSampling

LL_DMA2D_BGND_SetMemAddr

Function name

__STATIC_INLINE void LL_DMA2D_BGND_SetMemAddr (DMA2D_TypeDef * DMA2Dx, uint32_t MemoryAddress)

Function description

Set DMA2D background memory address, expressed on 32 bits ([31:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance
- **MemoryAddress:** Value between Min_Data=0 and Max_Data=0xFFFFFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- BGMAR MA LL_DMA2D_BGND_SetMemAddr

LL_DMA2D_BGND_GetMemAddr

Function name

__STATIC_INLINE uint32_t LL_DMA2D_BGND_GetMemAddr (const DMA2D_TypeDef * DMA2Dx)

Function description

Get DMA2D background memory address, expressed on 32 bits ([31:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Background:** memory address value between Min_Data=0 and Max_Data=0xFFFFFFFF

Reference Manual to LL API cross reference:

- BGMAR MA LL_DMA2D_BGND_GetMemAddr

LL_DMA2D_BGND_EnableCLUTLoad

Function name

__STATIC_INLINE void LL_DMA2D_BGND_EnableCLUTLoad (DMA2D_TypeDef * DMA2Dx)

Function description

Enable DMA2D background CLUT loading.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- BGPFCCR START LL_DMA2D_BGND_EnableCLUTLoad

LL_DMA2D_BGND_IsEnabledCLUTLoad

Function name

__STATIC_INLINE uint32_t LL_DMA2D_BGND_IsEnabledCLUTLoad (const DMA2D_TypeDef * DMA2Dx)

Function description

Indicate if DMA2D background CLUT loading is enabled.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- BGPFCCR START LL_DMA2D_BGND_IsEnabledCLUTLoad

LL_DMA2D_BGND_SetColorMode

Function name

__STATIC_INLINE void LL_DMA2D_BGND_SetColorMode (DMA2D_TypeDef * DMA2Dx, uint32_t ColorMode)

Function description

Set DMA2D background color mode.

Parameters

- **DMA2Dx:** DMA2D Instance
- **ColorMode:** This parameter can be one of the following values:
  - LL_DMA2D_INPUT_MODE_ARGB8888
  - LL_DMA2D_INPUT_MODE_RGB888
  - LL_DMA2D_INPUT_MODE_RGB565
  - LL_DMA2D_INPUT_MODE_ARGB1555
  - LL_DMA2D_INPUT_MODE_ARGB4444
  - LL_DMA2D_INPUT_MODE_L8
  - LL_DMA2D_INPUT_MODE_AL44
  - LL_DMA2D_INPUT_MODE_AL88
  - LL_DMA2D_INPUT_MODE_L4
  - LL_DMA2D_INPUT_MODE_A8
  - LL_DMA2D_INPUT_MODE_A4

Return values

- **None:**

Reference Manual to LL API cross reference:

- BGPFCCR CM LL_DMA2D_BGND_SetColorMode

LL_DMA2D_BGND_GetColorMode

Function name

__STATIC_INLINE uint32_t LL_DMA2D_BGND_GetColorMode (const DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D background color mode.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA2D_INPUT_MODE_ARGB8888
  - LL_DMA2D_INPUT_MODE_RGB888
  - LL_DMA2D_INPUT_MODE_RGB565
  - LL_DMA2D_INPUT_MODE_ARGB1555
  - LL_DMA2D_INPUT_MODE_ARGB4444
  - LL_DMA2D_INPUT_MODE_L8
  - LL_DMA2D_INPUT_MODE_AL44
  - LL_DMA2D_INPUT_MODE_AL88
  - LL_DMA2D_INPUT_MODE_L4
  - LL_DMA2D_INPUT_MODE_A8
  - LL_DMA2D_INPUT_MODE_A4

Reference Manual to LL API cross reference:

- BGPFCCR CM LL_DMA2D_BGND_GetColorMode

LL_DMA2D_BGND_SetAlphaMode

Function name

__STATIC_INLINE void LL_DMA2D_BGND_SetAlphaMode (DMA2D_TypeDef * DMA2Dx, uint32_t AphaMode)

Function description

Set DMA2D background alpha mode.

Parameters

- **DMA2Dx:** DMA2D Instance
- **AphaMode:** This parameter can be one of the following values:
  - LL_DMA2D_ALPHA_MODE_NO_MODIF
  - LL_DMA2D_ALPHA_MODE_REPLACE
  - LL_DMA2D_ALPHA_MODE_COMBINE

Return values

- **None:**

Reference Manual to LL API cross reference:

- BGPFCCR AM LL_DMA2D_BGND_SetAlphaMode

LL_DMA2D_BGND_GetAlphaMode

Function name

__STATIC_INLINE uint32_t LL_DMA2D_BGND_GetAlphaMode (const DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D background alpha mode.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA2D_ALPHA_MODE_NO_MODIF
  - LL_DMA2D_ALPHA_MODE_REPLACE
  - LL_DMA2D_ALPHA_MODE_COMBINE

Reference Manual to LL API cross reference:

- BGPFCCR AM LL_DMA2D_BGND_GetAlphaMode

LL_DMA2D_BGND_SetAlpha

Function name

__STATIC_INLINE void LL_DMA2D_BGND_SetAlpha (DMA2D_TypeDef * DMA2Dx, uint32_t Alpha)

Function description

Set DMA2D background alpha value, expressed on 8 bits ([7:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance
- **Alpha:** Value between Min_Data=0 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- BGPFCCR ALPHA LL_DMA2D_BGND_SetAlpha

LL_DMA2D_BGND_GetAlpha

Function name

__STATIC_INLINE uint32_t LL_DMA2D_BGND_GetAlpha (const DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D background alpha value, expressed on 8 bits ([7:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Alpha:** value between Min_Data=0 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- BGPFCCR ALPHA LL_DMA2D_BGND_GetAlpha

LL_DMA2D_BGND_SetRBSwapMode

Function name

__STATIC_INLINE void LL_DMA2D_BGND_SetRBSwapMode (DMA2D_TypeDef * DMA2Dx, uint32_t RBSwapMode)

Function description

Set DMA2D background Red Blue swap mode.

Parameters

- **DMA2Dx:** DMA2D Instance
- **RBSwapMode:** This parameter can be one of the following values:
  - LL_DMA2D_RB_MODE_REGULAR
  - LL_DMA2D_RB_MODE_SWAP

Return values

- **None:**

Reference Manual to LL API cross reference:

- BGPFCCR RBS LL_DMA2D_BGND_SetRBSwapMode

LL_DMA2D_BGND_GetRBSwapMode

Function name

__STATIC_INLINE uint32_t LL_DMA2D_BGND_GetRBSwapMode (const DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D background Red Blue swap mode.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA2D_RB_MODE_REGULAR
  - LL_DMA2D_RB_MODE_SWAP

Reference Manual to LL API cross reference:

- BGPFCCR RBS LL_DMA2D_BGND_GetRBSwapMode

LL_DMA2D_BGND_SetAlphaInvMode

Function name

__STATIC_INLINE void LL_DMA2D_BGND_SetAlphaInvMode (DMA2D_TypeDef * DMA2Dx, uint32_t AlphaInversionMode)

Function description

Set DMA2D background alpha inversion mode.

Parameters

- **DMA2Dx:** DMA2D Instance
- **AlphaInversionMode:** This parameter can be one of the following values:
  - LL_DMA2D_ALPHA_REGULAR
  - LL_DMA2D_ALPHA_INVERTED

Return values

- **None:**

Reference Manual to LL API cross reference:

- BGPFCCR AI LL_DMA2D_BGND_SetAlphaInvMode

LL_DMA2D_BGND_GetAlphaInvMode

Function name

__STATIC_INLINE uint32_t LL_DMA2D_BGND_GetAlphaInvMode (const DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D background alpha inversion mode.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA2D_ALPHA_REGULAR
  - LL_DMA2D_ALPHA_INVERTED

Reference Manual to LL API cross reference:

- BGPFCCR AI LL_DMA2D_BGND_GetAlphaInvMode

LL_DMA2D_BGND_SetLineOffset

Function name

__STATIC_INLINE void LL_DMA2D_BGND_SetLineOffset (DMA2D_TypeDef * DMA2Dx, uint32_t LineOffset)

Function description

Set DMA2D background line offset, expressed on 14 bits ([13:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance
- **LineOffset:** Value between Min_Data=0 and Max_Data=0x3FF

Return values

- **None:**

Reference Manual to LL API cross reference:

- BGOR LO LL_DMA2D_BGND_SetLineOffset

LL_DMA2D_BGND_GetLineOffset

Function name

__STATIC_INLINE uint32_t LL_DMA2D_BGND_GetLineOffset (const DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D background line offset, expressed on 14 bits ([13:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Background:** line offset value between Min_Data=0 and Max_Data=0x3FF

Reference Manual to LL API cross reference:

- BGOR LO LL_DMA2D_BGND_GetLineOffset

LL_DMA2D_BGND_SetColor

Function name

__STATIC_INLINE void LL_DMA2D_BGND_SetColor (DMA2D_TypeDef * DMA2Dx, uint32_t Red, uint32_t Green, uint32_t Blue)

Function description

Set DMA2D background color values, expressed on 24 bits ([23:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance
- **Red:** Value between Min_Data=0 and Max_Data=0xFF
- **Green:** Value between Min_Data=0 and Max_Data=0xFF
- **Blue:** Value between Min_Data=0 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- BGCOLR RED LL_DMA2D_BGND_SetColor
- BGCOLR GREEN LL_DMA2D_BGND_SetColor
- BGCOLR BLUE LL_DMA2D_BGND_SetColor

LL_DMA2D_BGND_SetRedColor

Function name

__STATIC_INLINE void LL_DMA2D_BGND_SetRedColor (DMA2D_TypeDef * DMA2Dx, uint32_t Red)

Function description

Set DMA2D background red color value, expressed on 8 bits ([7:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance
- **Red:** Value between Min_Data=0 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- BGCOLR RED LL_DMA2D_BGND_SetRedColor

LL_DMA2D_BGND_GetRedColor

Function name

__STATIC_INLINE uint32_t LL_DMA2D_BGND_GetRedColor (const DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D background red color value, expressed on 8 bits ([7:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Red:** color value between Min_Data=0 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- BGCOLR RED LL_DMA2D_BGND_GetRedColor

LL_DMA2D_BGND_SetGreenColor

Function name

__STATIC_INLINE void LL_DMA2D_BGND_SetGreenColor (DMA2D_TypeDef * DMA2Dx, uint32_t Green)

Function description

Set DMA2D background green color value, expressed on 8 bits ([7:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance
- **Green:** Value between Min_Data=0 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- BGCOLR GREEN LL_DMA2D_BGND_SetGreenColor

LL_DMA2D_BGND_GetGreenColor

Function name

__STATIC_INLINE uint32_t LL_DMA2D_BGND_GetGreenColor (const DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D background green color value, expressed on 8 bits ([7:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Green:** color value between Min_Data=0 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- BGCOLR GREEN LL_DMA2D_BGND_GetGreenColor

LL_DMA2D_BGND_SetBlueColor

Function name

__STATIC_INLINE void LL_DMA2D_BGND_SetBlueColor (DMA2D_TypeDef * DMA2Dx, uint32_t Blue)

Function description

Set DMA2D background blue color value, expressed on 8 bits ([7:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance
- **Blue:** Value between Min_Data=0 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- BGCOLR BLUE LL_DMA2D_BGND_SetBlueColor

LL_DMA2D_BGND_GetBlueColor

Function name

__STATIC_INLINE uint32_t LL_DMA2D_BGND_GetBlueColor (const DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D background blue color value, expressed on 8 bits ([7:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Blue:** color value between Min_Data=0 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- BGCOLR BLUE LL_DMA2D_BGND_GetBlueColor

LL_DMA2D_BGND_SetCLUTMemAddr

Function name

__STATIC_INLINE void LL_DMA2D_BGND_SetCLUTMemAddr (DMA2D_TypeDef * DMA2Dx, uint32_t CLUTMemoryAddress)

Function description

Set DMA2D background CLUT memory address, expressed on 32 bits ([31:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance
- **CLUTMemoryAddress:** Value between Min_Data=0 and Max_Data=0xFFFFFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- BGCMAR MA LL_DMA2D_BGND_SetCLUTMemAddr

LL_DMA2D_BGND_GetCLUTMemAddr

Function name

__STATIC_INLINE uint32_t LL_DMA2D_BGND_GetCLUTMemAddr (const DMA2D_TypeDef * DMA2Dx)

Function description

Get DMA2D background CLUT memory address, expressed on 32 bits ([31:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Background:** CLUT memory address value between Min_Data=0 and Max_Data=0xFFFFFFFF

Reference Manual to LL API cross reference:

- BGCMAR MA LL_DMA2D_BGND_GetCLUTMemAddr

LL_DMA2D_BGND_SetCLUTSize

Function name

__STATIC_INLINE void LL_DMA2D_BGND_SetCLUTSize (DMA2D_TypeDef * DMA2Dx, uint32_t CLUTSize)

Function description

Set DMA2D background CLUT size, expressed on 8 bits ([7:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance
- **CLUTSize:** Value between Min_Data=0 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- BGPFCCR CS LL_DMA2D_BGND_SetCLUTSize

LL_DMA2D_BGND_GetCLUTSize

Function name

__STATIC_INLINE uint32_t LL_DMA2D_BGND_GetCLUTSize (const DMA2D_TypeDef * DMA2Dx)

Function description

Get DMA2D background CLUT size, expressed on 8 bits ([7:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Background:** CLUT size value between Min_Data=0 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- BGPFCCR CS LL_DMA2D_BGND_GetCLUTSize

LL_DMA2D_BGND_SetCLUTColorMode

Function name

__STATIC_INLINE void LL_DMA2D_BGND_SetCLUTColorMode (DMA2D_TypeDef * DMA2Dx, uint32_t CLUTColorMode)

Function description

Set DMA2D background CLUT color mode.

Parameters

- **DMA2Dx:** DMA2D Instance
- **CLUTColorMode:** This parameter can be one of the following values:
  - LL_DMA2D_CLUT_COLOR_MODE_ARGB8888
  - LL_DMA2D_CLUT_COLOR_MODE_RGB888

Return values

- **None:**

Reference Manual to LL API cross reference:

- BGPFCCR CCM LL_DMA2D_BGND_SetCLUTColorMode

LL_DMA2D_BGND_GetCLUTColorMode

Function name

__STATIC_INLINE uint32_t LL_DMA2D_BGND_GetCLUTColorMode (const DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D background CLUT color mode.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA2D_CLUT_COLOR_MODE_ARGB8888
  - LL_DMA2D_CLUT_COLOR_MODE_RGB888

Reference Manual to LL API cross reference:

- BGPFCCR CCM LL_DMA2D_BGND_GetCLUTColorMode

LL_DMA2D_IsActiveFlag_CE

Function name

__STATIC_INLINE uint32_t LL_DMA2D_IsActiveFlag_CE (const DMA2D_TypeDef * DMA2Dx)

Function description

Check if the DMA2D Configuration Error Interrupt Flag is set or not.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR CEIF LL_DMA2D_IsActiveFlag_CE

LL_DMA2D_IsActiveFlag_CTC

Function name

__STATIC_INLINE uint32_t LL_DMA2D_IsActiveFlag_CTC (const DMA2D_TypeDef * DMA2Dx)

Function description

Check if the DMA2D CLUT Transfer Complete Interrupt Flag is set or not.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR CTCIF LL_DMA2D_IsActiveFlag_CTC

LL_DMA2D_IsActiveFlag_CAE

Function name

__STATIC_INLINE uint32_t LL_DMA2D_IsActiveFlag_CAE (const DMA2D_TypeDef * DMA2Dx)

Function description

Check if the DMA2D CLUT Access Error Interrupt Flag is set or not.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR CAEIF LL_DMA2D_IsActiveFlag_CAE

LL_DMA2D_IsActiveFlag_TW

Function name

__STATIC_INLINE uint32_t LL_DMA2D_IsActiveFlag_TW (const DMA2D_TypeDef * DMA2Dx)

Function description

Check if the DMA2D Transfer Watermark Interrupt Flag is set or not.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TWIF LL_DMA2D_IsActiveFlag_TW

LL_DMA2D_IsActiveFlag_TC

Function name

__STATIC_INLINE uint32_t LL_DMA2D_IsActiveFlag_TC (const DMA2D_TypeDef * DMA2Dx)

Function description

Check if the DMA2D Transfer Complete Interrupt Flag is set or not.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TCIF LL_DMA2D_IsActiveFlag_TC

LL_DMA2D_IsActiveFlag_TE

Function name

__STATIC_INLINE uint32_t LL_DMA2D_IsActiveFlag_TE (const DMA2D_TypeDef * DMA2Dx)

Function description

Check if the DMA2D Transfer Error Interrupt Flag is set or not.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TEIF LL_DMA2D_IsActiveFlag_TE

LL_DMA2D_ClearFlag_CE

Function name

__STATIC_INLINE void LL_DMA2D_ClearFlag_CE (DMA2D_TypeDef * DMA2Dx)

Function description

Clear DMA2D Configuration Error Interrupt Flag.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CCEIF LL_DMA2D_ClearFlag_CE

LL_DMA2D_ClearFlag_CTC

Function name

__STATIC_INLINE void LL_DMA2D_ClearFlag_CTC (DMA2D_TypeDef * DMA2Dx)

Function description

Clear DMA2D CLUT Transfer Complete Interrupt Flag.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CCTCIF LL_DMA2D_ClearFlag_CTC

LL_DMA2D_ClearFlag_CAE

Function name

__STATIC_INLINE void LL_DMA2D_ClearFlag_CAE (DMA2D_TypeDef * DMA2Dx)

Function description

Clear DMA2D CLUT Access Error Interrupt Flag.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CAECIF LL_DMA2D_ClearFlag_CAE

LL_DMA2D_ClearFlag_TW

Function name

__STATIC_INLINE void LL_DMA2D_ClearFlag_TW (DMA2D_TypeDef * DMA2Dx)

Function description

Clear DMA2D Transfer Watermark Interrupt Flag.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CTWIF LL_DMA2D_ClearFlag_TW

LL_DMA2D_ClearFlag_TC

Function name

__STATIC_INLINE void LL_DMA2D_ClearFlag_TC (DMA2D_TypeDef * DMA2Dx)

Function description

Clear DMA2D Transfer Complete Interrupt Flag.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CTCIF LL_DMA2D_ClearFlag_TC

LL_DMA2D_ClearFlag_TE

Function name

__STATIC_INLINE void LL_DMA2D_ClearFlag_TE (DMA2D_TypeDef * DMA2Dx)

Function description

Clear DMA2D Transfer Error Interrupt Flag.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CTEIF LL_DMA2D_ClearFlag_TE

LL_DMA2D_EnableIT_CE

Function name

__STATIC_INLINE void LL_DMA2D_EnableIT_CE (DMA2D_TypeDef * DMA2Dx)

Function description

Enable Configuration Error Interrupt.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CEIE LL_DMA2D_EnableIT_CE

LL_DMA2D_EnableIT_CTC

Function name

__STATIC_INLINE void LL_DMA2D_EnableIT_CTC (DMA2D_TypeDef * DMA2Dx)

Function description

Enable CLUT Transfer Complete Interrupt.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CTCIE LL_DMA2D_EnableIT_CTC

LL_DMA2D_EnableIT_CAE

Function name

__STATIC_INLINE void LL_DMA2D_EnableIT_CAE (DMA2D_TypeDef * DMA2Dx)

Function description

Enable CLUT Access Error Interrupt.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CAEIE LL_DMA2D_EnableIT_CAE

LL_DMA2D_EnableIT_TW

Function name

__STATIC_INLINE void LL_DMA2D_EnableIT_TW (DMA2D_TypeDef * DMA2Dx)

Function description

Enable Transfer Watermark Interrupt.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR TWIE LL_DMA2D_EnableIT_TW

LL_DMA2D_EnableIT_TC

Function name

__STATIC_INLINE void LL_DMA2D_EnableIT_TC (DMA2D_TypeDef * DMA2Dx)

Function description

Enable Transfer Complete Interrupt.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR TCIE LL_DMA2D_EnableIT_TC

LL_DMA2D_EnableIT_TE

Function name

__STATIC_INLINE void LL_DMA2D_EnableIT_TE (DMA2D_TypeDef * DMA2Dx)

Function description

Enable Transfer Error Interrupt.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR TEIE LL_DMA2D_EnableIT_TE

LL_DMA2D_DisableIT_CE

Function name

__STATIC_INLINE void LL_DMA2D_DisableIT_CE (DMA2D_TypeDef * DMA2Dx)

Function description

Disable Configuration Error Interrupt.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CEIE LL_DMA2D_DisableIT_CE

LL_DMA2D_DisableIT_CTC

Function name

__STATIC_INLINE void LL_DMA2D_DisableIT_CTC (DMA2D_TypeDef * DMA2Dx)

Function description

Disable CLUT Transfer Complete Interrupt.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CTCIE LL_DMA2D_DisableIT_CTC

LL_DMA2D_DisableIT_CAE

Function name

__STATIC_INLINE void LL_DMA2D_DisableIT_CAE (DMA2D_TypeDef * DMA2Dx)

Function description

Disable CLUT Access Error Interrupt.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CAEIE LL_DMA2D_DisableIT_CAE

LL_DMA2D_DisableIT_TW

Function name

__STATIC_INLINE void LL_DMA2D_DisableIT_TW (DMA2D_TypeDef * DMA2Dx)

Function description

Disable Transfer Watermark Interrupt.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR TWIE LL_DMA2D_DisableIT_TW

LL_DMA2D_DisableIT_TC

Function name

__STATIC_INLINE void LL_DMA2D_DisableIT_TC (DMA2D_TypeDef * DMA2Dx)

Function description

Disable Transfer Complete Interrupt.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR TCIE LL_DMA2D_DisableIT_TC

LL_DMA2D_DisableIT_TE

Function name

__STATIC_INLINE void LL_DMA2D_DisableIT_TE (DMA2D_TypeDef * DMA2Dx)

Function description

Disable Transfer Error Interrupt.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR TEIE LL_DMA2D_DisableIT_TE

LL_DMA2D_IsEnabledIT_CE

Function name

__STATIC_INLINE uint32_t LL_DMA2D_IsEnabledIT_CE (const DMA2D_TypeDef * DMA2Dx)

Function description

Check if the DMA2D Configuration Error interrupt source is enabled or disabled.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR CEIE LL_DMA2D_IsEnabledIT_CE

LL_DMA2D_IsEnabledIT_CTC

Function name

__STATIC_INLINE uint32_t LL_DMA2D_IsEnabledIT_CTC (const DMA2D_TypeDef * DMA2Dx)

Function description

Check if the DMA2D CLUT Transfer Complete interrupt source is enabled or disabled.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR CTCIE LL_DMA2D_IsEnabledIT_CTC

LL_DMA2D_IsEnabledIT_CAE

Function name

__STATIC_INLINE uint32_t LL_DMA2D_IsEnabledIT_CAE (const DMA2D_TypeDef * DMA2Dx)

Function description

Check if the DMA2D CLUT Access Error interrupt source is enabled or disabled.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR CAEIE LL_DMA2D_IsEnabledIT_CAE

LL_DMA2D_IsEnabledIT_TW

Function name

__STATIC_INLINE uint32_t LL_DMA2D_IsEnabledIT_TW (const DMA2D_TypeDef * DMA2Dx)

Function description

Check if the DMA2D Transfer Watermark interrupt source is enabled or disabled.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR TWIE LL_DMA2D_IsEnabledIT_TW

LL_DMA2D_IsEnabledIT_TC

Function name

__STATIC_INLINE uint32_t LL_DMA2D_IsEnabledIT_TC (const DMA2D_TypeDef * DMA2Dx)

Function description

Check if the DMA2D Transfer Complete interrupt source is enabled or disabled.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR TCIE LL_DMA2D_IsEnabledIT_TC

LL_DMA2D_IsEnabledIT_TE

Function name

__STATIC_INLINE uint32_t LL_DMA2D_IsEnabledIT_TE (const DMA2D_TypeDef * DMA2Dx)

Function description

Check if the DMA2D Transfer Error interrupt source is enabled or disabled.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR TEIE LL_DMA2D_IsEnabledIT_TE

LL_DMA2D_DeInit

Function name

ErrorStatus LL_DMA2D_DeInit (const DMA2D_TypeDef * DMA2Dx)

Function description

De-initialize DMA2D registers (registers restored to their default values).

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: DMA2D registers are de-initialized
  - ERROR: DMA2D registers are not de-initialized

LL_DMA2D_Init

Function name

ErrorStatus LL_DMA2D_Init (DMA2D_TypeDef * DMA2Dx, LL_DMA2D_InitTypeDef * DMA2D_InitStruct)

Function description

Initialize DMA2D registers according to the specified parameters in DMA2D_InitStruct.

Parameters

- **DMA2Dx:** DMA2D Instance
- **DMA2D_InitStruct:** pointer to a LL_DMA2D_InitTypeDef structure that contains the configuration information for the specified DMA2D peripheral.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: DMA2D registers are initialized according to DMA2D_InitStruct content
  - ERROR: Issue occurred during DMA2D registers initialization

Notes

- DMA2D transfers must be disabled to set initialization bits in configuration registers, otherwise ERROR result is returned.

LL_DMA2D_StructInit

Function name

void LL_DMA2D_StructInit (LL_DMA2D_InitTypeDef * DMA2D_InitStruct)

Function description

Set each LL_DMA2D_InitTypeDef field to default value.

Parameters

- **DMA2D_InitStruct:** pointer to a LL_DMA2D_InitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

LL_DMA2D_ConfigLayer

Function name

void LL_DMA2D_ConfigLayer (DMA2D_TypeDef * DMA2Dx, LL_DMA2D_LayerCfgTypeDef * DMA2D_LayerCfg, uint32_t LayerIdx)

Function description

Configure the foreground or background according to the specified parameters in the LL_DMA2D_LayerCfgTypeDef structure.

Parameters

- **DMA2Dx:** DMA2D Instance
- **DMA2D_LayerCfg:** pointer to a LL_DMA2D_LayerCfgTypeDef structure that contains the configuration information for the specified layer.
- **LayerIdx:** DMA2D Layer index. This parameter can be one of the following values: 0(background) / 1(foreground)

Return values

- **None:**

LL_DMA2D_LayerCfgStructInit

Function name

void LL_DMA2D_LayerCfgStructInit (LL_DMA2D_LayerCfgTypeDef * DMA2D_LayerCfg)

Function description

Set each LL_DMA2D_LayerCfgTypeDef field to default value.

Parameters

- **DMA2D_LayerCfg:** pointer to a LL_DMA2D_LayerCfgTypeDef structure whose fields will be set to default values.

Return values

- **None:**

LL_DMA2D_ConfigOutputColor

Function name

void LL_DMA2D_ConfigOutputColor (DMA2D_TypeDef * DMA2Dx, LL_DMA2D_ColorTypeDef * DMA2D_ColorStruct)

Function description

Initialize DMA2D output color register according to the specified parameters in DMA2D_ColorStruct.

Parameters

- **DMA2Dx:** DMA2D Instance
- **DMA2D_ColorStruct:** pointer to a LL_DMA2D_ColorTypeDef structure that contains the color configuration information for the specified DMA2D peripheral.

Return values

- **None:**

LL_DMA2D_GetOutputBlueColor

Function name

uint32_t LL_DMA2D_GetOutputBlueColor (const DMA2D_TypeDef * DMA2Dx, uint32_t ColorMode)

Function description

Return DMA2D output Blue color.

Parameters

- **DMA2Dx:** DMA2D Instance.
- **ColorMode:** This parameter can be one of the following values:
  - LL_DMA2D_OUTPUT_MODE_ARGB8888
  - LL_DMA2D_OUTPUT_MODE_RGB888
  - LL_DMA2D_OUTPUT_MODE_RGB565
  - LL_DMA2D_OUTPUT_MODE_ARGB1555
  - LL_DMA2D_OUTPUT_MODE_ARGB4444

Return values

- **Output:** Blue color value between Min_Data=0 and Max_Data=0xFF

LL_DMA2D_GetOutputGreenColor

Function name

uint32_t LL_DMA2D_GetOutputGreenColor (const DMA2D_TypeDef * DMA2Dx, uint32_t ColorMode)

Function description

Return DMA2D output Green color.

Parameters

- **DMA2Dx:** DMA2D Instance.
- **ColorMode:** This parameter can be one of the following values:
  - LL_DMA2D_OUTPUT_MODE_ARGB8888
  - LL_DMA2D_OUTPUT_MODE_RGB888
  - LL_DMA2D_OUTPUT_MODE_RGB565
  - LL_DMA2D_OUTPUT_MODE_ARGB1555
  - LL_DMA2D_OUTPUT_MODE_ARGB4444

Return values

- **Output:** Green color value between Min_Data=0 and Max_Data=0xFF

LL_DMA2D_GetOutputRedColor

Function name

uint32_t LL_DMA2D_GetOutputRedColor (const DMA2D_TypeDef * DMA2Dx, uint32_t ColorMode)

Function description

Return DMA2D output Red color.

Parameters

- **DMA2Dx:** DMA2D Instance.
- **ColorMode:** This parameter can be one of the following values:
  - LL_DMA2D_OUTPUT_MODE_ARGB8888
  - LL_DMA2D_OUTPUT_MODE_RGB888
  - LL_DMA2D_OUTPUT_MODE_RGB565
  - LL_DMA2D_OUTPUT_MODE_ARGB1555
  - LL_DMA2D_OUTPUT_MODE_ARGB4444

Return values

- **Output:** Red color value between Min_Data=0 and Max_Data=0xFF

LL_DMA2D_GetOutputAlphaColor

Function name

uint32_t LL_DMA2D_GetOutputAlphaColor (const DMA2D_TypeDef * DMA2Dx, uint32_t ColorMode)

Function description

Return DMA2D output Alpha color.

Parameters

- **DMA2Dx:** DMA2D Instance.
- **ColorMode:** This parameter can be one of the following values:
  - LL_DMA2D_OUTPUT_MODE_ARGB8888
  - LL_DMA2D_OUTPUT_MODE_RGB888
  - LL_DMA2D_OUTPUT_MODE_RGB565
  - LL_DMA2D_OUTPUT_MODE_ARGB1555
  - LL_DMA2D_OUTPUT_MODE_ARGB4444

Return values

- **Output:** Alpha color value between Min_Data=0 and Max_Data=0xFF

LL_DMA2D_ConfigSize

Function name

void LL_DMA2D_ConfigSize (DMA2D_TypeDef * DMA2Dx, uint32_t NbrOfLines, uint32_t NbrOfPixelsPerLines)

Function description

Configure DMA2D transfer size.

Parameters

- **DMA2Dx:** DMA2D Instance
- **NbrOfLines:** Value between Min_Data=0 and Max_Data=0xFFFF
- **NbrOfPixelsPerLines:** Value between Min_Data=0 and Max_Data=0x3FFF

Return values

- **None:**

90.3 DMA2D Firmware driver defines

The following section lists the various define and macros of the module.

90.3.1 DMA2D

DMA2D

Chroma Sub Sampling

LL_DMA2D_CSS_444

No chroma sub-sampling 4:4:4

LL_DMA2D_CSS_422

chroma sub-sampling 4:2:2

LL_DMA2D_CSS_420

chroma sub-sampling 4:2:0

Alpha Inversion

LL_DMA2D_ALPHA_REGULAR

Regular alpha

LL_DMA2D_ALPHA_INVERTED

Inverted alpha

Alpha Mode

LL_DMA2D_ALPHA_MODE_NO_MODIF

No modification of the alpha channel value

LL_DMA2D_ALPHA_MODE_REPLACE

Replace original alpha channel value by programmed alpha value

LL_DMA2D_ALPHA_MODE_COMBINE

Replace original alpha channel value by programmed alpha value with, original alpha channel value

CLUT Color Mode

LL_DMA2D_CLUT_COLOR_MODE_ARGB8888

ARGB8888

LL_DMA2D_CLUT_COLOR_MODE_RGB888

RGB888

Get Flags Defines

LL_DMA2D_FLAG_CEIF

Configuration Error Interrupt Flag

LL_DMA2D_FLAG_CTCIF

CLUT Transfer Complete Interrupt Flag

LL_DMA2D_FLAG_CAEIF

CLUT Access Error Interrupt Flag

LL_DMA2D_FLAG_TWIF

Transfer Watermark Interrupt Flag

LL_DMA2D_FLAG_TCIF

Transfer Complete Interrupt Flag

LL_DMA2D_FLAG_TEIF

Transfer Error Interrupt Flag

Input Color Mode

LL_DMA2D_INPUT_MODE_ARGB8888

ARGB8888

LL_DMA2D_INPUT_MODE_RGB888

RGB888

LL_DMA2D_INPUT_MODE_RGB565

RGB565

LL_DMA2D_INPUT_MODE_ARGB1555

ARGB1555

LL_DMA2D_INPUT_MODE_ARGB4444

ARGB4444

LL_DMA2D_INPUT_MODE_L8

L8

LL_DMA2D_INPUT_MODE_AL44

AL44

LL_DMA2D_INPUT_MODE_AL88

AL88

LL_DMA2D_INPUT_MODE_L4

L4

LL_DMA2D_INPUT_MODE_A8

A8

LL_DMA2D_INPUT_MODE_A4

A4

LL_DMA2D_INPUT_MODE_YCBCR

YCbCr

IT Defines

LL_DMA2D_IT_CEIE

Configuration Error Interrupt

LL_DMA2D_IT_CTCIE

CLUT Transfer Complete Interrupt

LL_DMA2D_IT_CAEIE

CLUT Access Error Interrupt

LL_DMA2D_IT_TWIE

Transfer Watermark Interrupt

LL_DMA2D_IT_TCIE

Transfer Complete Interrupt

LL_DMA2D_IT_TEIE

Transfer Error Interrupt

Line Offset Mode

LL_DMA2D_LINE_OFFSET_PIXELS

Line offsets are expressed in pixels

LL_DMA2D_LINE_OFFSET_BYTES

Line offsets are expressed in bytes

Mode

LL_DMA2D_MODE_M2M

DMA2D memory to memory transfer mode

LL_DMA2D_MODE_M2M_PFC

DMA2D memory to memory with pixel format conversion transfer mode

LL_DMA2D_MODE_M2M_BLEND

DMA2D memory to memory with blending transfer mode

LL_DMA2D_MODE_R2M

DMA2D register to memory transfer mode

LL_DMA2D_MODE_M2M_BLEND_FIXED_COLOR_FG

DMA2D memory to memory with blending transfer mode and fixed color foreground

LL_DMA2D_MODE_M2M_BLEND_FIXED_COLOR_BG

DMA2D memory to memory with blending transfer mode and fixed color background

Output Color Mode

LL_DMA2D_OUTPUT_MODE_ARGB8888

ARGB8888

LL_DMA2D_OUTPUT_MODE_RGB888

RGB888

LL_DMA2D_OUTPUT_MODE_RGB565

RGB565

LL_DMA2D_OUTPUT_MODE_ARGB1555

ARGB1555

LL_DMA2D_OUTPUT_MODE_ARGB4444

ARGB4444

Swap Mode

LL_DMA2D_SWAP_MODE_REGULAR

Regular order

LL_DMA2D_SWAP_MODE_TWO_BY_TWO

Bytes swapped two by two

Red Blue Swap

LL_DMA2D_RB_MODE_REGULAR

RGB or ARGB

LL_DMA2D_RB_MODE_SWAP

BGR or ABGR

Common Write and read registers Macros

LL_DMA2D_WriteReg

Description:

- Write a value in DMA2D register.

Parameters:

- __INSTANCE__: DMA2D Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_DMA2D_ReadReg

Description:

- Read a value in DMA2D register.

Parameters:

- __INSTANCE__: DMA2D Instance
- __REG__: Register to be read

Return value:

- Register: value

91 LL DMA Generic Driver

91.1 DMA Firmware driver registers structures

91.1.1 LL_DMA_InitTypeDef

LL_DMA_InitTypeDef is defined in the stm32n6xx_ll_dma.h

Data Fields

- ***uint32_t SrcAddress***
- ***uint32_t DestAddress***
- ***uint32_t Direction***
- ***uint32_t BlkHWRequest***
- ***uint32_t DataAlignment***
- ***uint32_t SrcBurstLength***
- ***uint32_t DestBurstLength***
- ***uint32_t SrcDataWidth***
- ***uint32_t DestDataWidth***
- ***uint32_t SrcIncMode***
- ***uint32_t DestIncMode***
- ***uint32_t Priority***
- ***uint32_t BlkDataLength***
- ***uint32_t BlkRptCount***
- ***uint32_t TriggerMode***
- ***uint32_t TriggerPolarity***
- ***uint32_t TriggerSelection***
- ***uint32_t Request***
- ***uint32_t TransferEventMode***
- ***uint32_t DestWordExchange***
- ***uint32_t DestHWordExchange***
- ***uint32_t DestByteExchange***
- ***uint32_t SrcByteExchange***
- ***uint32_t SrcAllocatedPort***
- ***uint32_t DestAllocatedPort***
- ***uint32_t LinkAllocatedPort***
- ***uint32_t LinkStepMode***
- ***uint32_t SrcAddrUpdateMode***
- ***uint32_t DestAddrUpdateMode***
- ***uint32_t SrcAddrOffset***
- ***uint32_t DestAddrOffset***
- ***uint32_t BlkRptSrcAddrUpdateMode***
- ***uint32_t BlkRptDestAddrUpdateMode***
- ***uint32_t BlkRptSrcAddrOffset***
- ***uint32_t BlkRptDestAddrOffset***
- ***uint32_t LinkedListBaseAddr***
- ***uint32_t LinkedListAddrOffset***
- ***uint32_t Mode***

Field Documentation

- ***uint32_t LL_DMA_InitTypeDef::SrcAddress***
  This field specify the data transfer source address. Programming this field is mandatory for all available DMA channels. This parameter must be a value between Min_Data = 0 and Max_Data = 0xFFFFFFFF. This feature can be modified afterwards using unitary function **LL_DMA_SetSrcAddress()**.
- ***uint32_t LL_DMA_InitTypeDef::DestAddress***
  This field specify the data transfer destination address. Programming this field is mandatory for all available DMA channels. This parameter must be a value between Min_Data = 0 and Max_Data = 0xFFFFFFFF. This feature can be modified afterwards using unitary function **LL_DMA_SetDestAddress()**.
- ***uint32_t LL_DMA_InitTypeDef::Direction***
  This field specify the data transfer direction. Programming this field is mandatory for all available DMA channels. This parameter can be a value of ***DMA_LL_EC_TRANSFER_DIRECTION***. This feature can be modified afterwards using unitary function **LL_DMA_SetDataTransferDirection()**.
- ***uint32_t LL_DMA_InitTypeDef::BlkHWRequest***
  This field specify the hardware request unity. Programming this field is mandatory for all available DMA channels. This parameter can be a value of ***DMA_LL_EC_BLKHW_REQUEST***. This feature can be modified afterwards using unitary function **LL_DMA_SetBlkHWRequest()**.
- ***uint32_t LL_DMA_InitTypeDef::DataAlignment***
  This field specify the transfer data alignment. Programming this field is mandatory for all available DMA channels. This parameter can be a value of ***DMA_LL_EC_DATA_ALIGNMENT***. This feature can be modified afterwards using unitary function **LL_DMA_SetDataAlignment()**.
- ***uint32_t LL_DMA_InitTypeDef::SrcBurstLength***
  This field specify the source burst length of transfer in bytes. Programming this field is mandatory for all available DMA channels. This parameter must be a value between Min_Data = 1 and Max_Data = 64. This feature can be modified afterwards using unitary function **LL_DMA_SetSrcBurstLength()**.
- ***uint32_t LL_DMA_InitTypeDef::DestBurstLength***
  This field specify the destination burst length of transfer in bytes. Programming this field is mandatory for all available DMA channels. This parameter must be a value between Min_Data = 1 and Max_Data = 64. This feature can be modified afterwards using unitary function **LL_DMA_SetDestBurstLength()**.
- ***uint32_t LL_DMA_InitTypeDef::SrcDataWidth***
  This field specify the source data width. Programming this field is mandatory for all available DMA channels. This parameter can be a value of ***DMA_LL_EC_SOURCE_DATA_WIDTH***. This feature can be modified afterwards using unitary function **LL_DMA_SetSrcDataWidth()**.
- ***uint32_t LL_DMA_InitTypeDef::DestDataWidth***
  This field specify the destination data width. Programming this field is mandatory for all available DMA channels. This parameter can be a value of ***DMA_LL_EC_DESTINATION_DATA_WIDTH***. This feature can be modified afterwards using unitary function **LL_DMA_SetDestDataWidth()**.
- ***uint32_t LL_DMA_InitTypeDef::SrcIncMode***
  This field specify the source burst increment mode. Programming this field is mandatory for all available DMA channels. This parameter can be a value of ***DMA_LL_EC_SOURCE_INCREMENT_MODE***. This feature can be modified afterwards using unitary function **LL_DMA_SetSrcIncMode()**.
- ***uint32_t LL_DMA_InitTypeDef::DestIncMode***
  This field specify the destination burst increment mode. Programming this field is mandatory for all available DMA channels. This parameter can be a value of ***DMA_LL_EC_DESTINATION_INCREMENT_MODE***. This feature can be modified afterwards using unitary function **LL_DMA_SetDestIncMode()**.
- ***uint32_t LL_DMA_InitTypeDef::Priority***
  This field specify the channel priority level. Programming this field is mandatory for all available DMA channels. This parameter can be a value of ***DMA_LL_EC_PRIORITY_LEVEL***. This feature can be modified afterwards using unitary function **LL_DMA_SetChannelPriorityLevel()**.
- ***uint32_t LL_DMA_InitTypeDef::BlkDataLength***
  This field specify the length of a block transfer in bytes. Programming this field is mandatory for all available DMA channels. This parameter must be a value between Min_Data = 0 and Max_Data = 0x0000FFFF. This feature can be modified afterwards using unitary function **LL_DMA_SetBlkDataLength()**.
- ***uint32_t LL_DMA_InitTypeDef::BlkRptCount***
  This field specify the number of repetitions of the current block. Programming this field is mandatory only for 2D addressing channels. This parameter can be a value between 1 and 2048 Min_Data = 0 and Max_Data = 0x000007FF. This feature can be modified afterwards using unitary function **LL_DMA_SetBlkRptCount()**.

- ***uint32_t LL_DMA_InitTypeDef::TriggerMode***
  This field specify the trigger mode. Programming this field is mandatory for all available DMA channels. This parameter can be a value of ***DMA_LL_EC_TRIGGER_MODE***. This feature can be modified afterwards using unitary function **LL_DMA_SetTriggerMode()**.
- ***uint32_t LL_DMA_InitTypeDef::TriggerPolarity***
  This field specify the trigger event polarity. Programming this field is mandatory for all available DMA channels. This parameter can be a value of ***DMA_LL_EC_TRIGGER_POLARITY***. This feature can be modified afterwards using unitary function **LL_DMA_SetTriggerPolarity()**.
- ***uint32_t LL_DMA_InitTypeDef::TriggerSelection***
  This field specify the trigger event selection. Programming this field is mandatory for all available DMA channels. This parameter can be a value of ***DMA_LL_EC_TRIGGER_SELECTION***. This feature can be modified afterwards using unitary function **LL_DMA_SetHWTrigger()**.
- ***uint32_t LL_DMA_InitTypeDef::Request***
  This field specify the peripheral request selection. Programming this field is mandatory for all available DMA channels. This parameter can be a value of ***DMA_LL_EC_REQUEST_SELECTION***. This feature can be modified afterwards using unitary function **LL_DMA_SetPeriphRequest()**.
- ***uint32_t LL_DMA_InitTypeDef::TransferEventMode***
  This field specify the transfer event mode. Programming this field is mandatory for all available DMA channels. This parameter can be a value of ***DMA_LL_EC_TRANSFER_EVENT_MODE***. This feature can be modified afterwards using unitary function **LL_DMA_SetTransferEventMode()**.
- ***uint32_t LL_DMA_InitTypeDef::DestWordExchange***
  This field specify the destination word exchange. Programming this field is mandatory for all available HPDMA channels. This parameter can be a value of ***DMA_LL_EC_DEST_WORD_EXCHANGE***. This feature can be modified afterwards using unitary function **LL_DMA_SetDestWordExchange()**.
- ***uint32_t LL_DMA_InitTypeDef::DestHWordExchange***
  This field specify the destination half word exchange. Programming this field is mandatory for all available DMA channels. This parameter can be a value of ***DMA_LL_EC_DEST_HALFWORD_EXCHANGE***. This feature can be modified afterwards using unitary function **LL_DMA_SetDestHWordExchange()**.
- ***uint32_t LL_DMA_InitTypeDef::DestByteExchange***
  This field specify the destination byte exchange. Programming this field is mandatory for all available DMA channels. This parameter can be a value of ***DMA_LL_EC_DEST_BYTE_EXCHANGE***. This feature can be modified afterwards using unitary function **LL_DMA_SetDestByteExchange()**.
- ***uint32_t LL_DMA_InitTypeDef::SrcByteExchange***
  This field specify the source byte exchange. Programming this field is mandatory for all available DMA channels. This parameter can be a value of ***DMA_LL_EC_SRC_BYTE_EXCHANGE***. This feature can be modified afterwards using unitary function **LL_DMA_SetSrcByteExchange()**.
- ***uint32_t LL_DMA_InitTypeDef::SrcAllocatedPort***
  This field specify the source allocated port. Programming this field is mandatory for all available DMA channels. This parameter can be a value of ***DMA_LL_EC_SOURCE_ALLOCATED_PORT***. This feature can be modified afterwards using unitary function **LL_DMA_SetSrcAllocatedPort()**.
- ***uint32_t LL_DMA_InitTypeDef::DestAllocatedPort***
  This field specify the destination allocated port. Programming this field is mandatory for all available DMA channels. This parameter can be a value of ***DMA_LL_EC_DESTINATION_ALLOCATED_PORT***. This feature can be modified afterwards using unitary function **LL_DMA_SetDestAllocatedPort()**.
- ***uint32_t LL_DMA_InitTypeDef::LinkAllocatedPort***
  This field specify the linked-list allocated port. Programming this field is mandatory for all available DMA channels. This parameter can be a value of ***DMA_LL_EC_LINKED_LIST_ALLOCATED_PORT***. This feature can be modified afterwards using unitary function **LL_DMA_SetLinkAllocatedPort()**.
- ***uint32_t LL_DMA_InitTypeDef::LinkStepMode***
  This field specify the link step mode. Programming this field is mandatory for all available DMA channels. This parameter can be a value of ***DMA_LL_EC_LINK_STEP_MODE***. This feature can be modified afterwards using unitary function **LL_DMA_SetLinkStepMode()**.
- ***uint32_t LL_DMA_InitTypeDef::SrcAddrUpdateMode***
  This field specify the source address update mode. Programming this field is mandatory only for 2D addressing channels. This parameter can be a value of ***DMA_LL_EC_SRC_ADDR_UPDATE_MODE***. This feature can be modified afterwards using unitary function **LL_DMA_SetSrcAddrUpdate()**.

- ***uint32_t LL_DMA_InitTypeDef::DestAddrUpdateMode***
  This field specify the destination address update mode. Programming this field is mandatory only for 2D addressing channels. This parameter can be a value of ***DMA_LL_EC_DEST_ADDR_UPDATE_MODE***. This feature can be modified afterwards using unitary function **LL_DMA_SetDestAddrUpdate()**.
- ***uint32_t LL_DMA_InitTypeDef::SrcAddrOffset***
  This field specifies the source address offset. Programming this field is mandatory only for 2D addressing channels. This parameter can be a value Between 0 to 0x00001FFF. This feature can be modified afterwards using unitary function **LL_DMA_SetSrcAddrUpdateValue()**.
- ***uint32_t LL_DMA_InitTypeDef::DestAddrOffset***
  This field specifies the destination address offset. Programming this field is mandatory only for 2D addressing channels. This parameter can be a value Between 0 to 0x00001FFF. This feature can be modified afterwards using unitary function **LL_DMA_SetDestAddrUpdateValue()**.
- ***uint32_t LL_DMA_InitTypeDef::BlkRptSrcAddrUpdateMode***
  This field specifies the block repeat source address update mode. Programming this field is mandatory only for 2D addressing channels. This parameter can be a value of ***DMA_LL_EC_BLK_RPT_SRC_ADDR_UPDATE_MODE***. This feature can be modified afterwards using unitary function **LL_DMA_SetBlkRptSrcAddrUpdate()**.
- ***uint32_t LL_DMA_InitTypeDef::BlkRptDestAddrUpdateMode***
  This field specifies the block repeat destination address update mode. Programming this field is mandatory only for 2D addressing channels. This parameter can be a value of ***DMA_LL_EC_BLK_RPT_DEST_ADDR_UPDATE_MODE***. This feature can be modified afterwards using unitary function **LL_DMA_SetBlkRptDestAddrUpdate()**.
- ***uint32_t LL_DMA_InitTypeDef::BlkRptSrcAddrOffset***
  This field specifies the block repeat source address offset. Programming this field is mandatory only for 2D addressing channels. This parameter can be a value Between 0 to 0x0000FFFF. This feature can be modified afterwards using unitary function **LL_DMA_SetBlkRptSrcAddrUpdateValue()**.
- ***uint32_t LL_DMA_InitTypeDef::BlkRptDestAddrOffset***
  This field specifies the block repeat destination address offset. Programming this field is mandatory only for 2D addressing channels. This parameter can be a value Between 0 to 0x0000FFFF. This feature can be modified afterwards using unitary function **LL_DMA_SetBlkRptDestAddrUpdateValue()**.
- ***uint32_t LL_DMA_InitTypeDef::LinkedListBaseAddr***
  This field specify the linked list base address. Programming this field is mandatory for all available DMA channels. This parameter can be a value Between 0 to 0xFFFF0000 (where the 4 first bytes are always forced to 0). This feature can be modified afterwards using unitary function **LL_DMA_SetLinkedListBaseAddr()**.
- ***uint32_t LL_DMA_InitTypeDef::LinkedListAddrOffset***
  Specifies the linked list address offset. Programming this field is mandatory for all available DMA channels. This parameter can be a value Between 0 to 0x0000FFFC. This feature can be modified afterwards using unitary function **LL_DMA_SetLinkedListAddrOffset()**.
- ***uint32_t LL_DMA_InitTypeDef::Mode***
  Specifies the transfer mode for the DMA channel. This parameter can be a value of ***DMA_LL_TRANSFER_MODE***

91.1.2 LL_DMA_InitLinkedListTypeDef

LL_DMA_InitLinkedListTypeDef is defined in the stm32n6xx_ll_dma.h

Data Fields

- ***uint32_t Priority***
- ***uint32_t LinkStepMode***
- ***uint32_t LinkAllocatedPort***
- ***uint32_t TransferEventMode***

Field Documentation

- ***uint32_t LL_DMA_InitLinkedListTypeDef::Priority***
  This field specify the channel priority level. Programming this field is mandatory for all available DMA channels. This parameter can be a value of ***DMA_LL_EC_PRIORITY_LEVEL***. This feature can be modified afterwards using unitary function **LL_DMA_SetChannelPriorityLevel()**.

- ***uint32_t LL_DMA_InitLinkedListTypeDef::LinkStepMode***
  This field specify the link step mode. Programming this field is mandatory for all available DMA channels. This parameter can be a value of ***DMA_LL_EC_LINK_STEP_MODE***. This feature can be modified afterwards using unitary function **LL_DMA_SetLinkStepMode()**.
- ***uint32_t LL_DMA_InitLinkedListTypeDef::LinkAllocatedPort***
  This field specify the linked-list allocated port. Programming this field is mandatory for all available DMA channels. This parameter can be a value of ***DMA_LL_EC_LINKED_LIST_ALLOCATED_PORT***. This feature can be modified afterwards using unitary function **LL_DMA_SetLinkAllocatedPort()**.
- ***uint32_t LL_DMA_InitLinkedListTypeDef::TransferEventMode***
  This field specify the transfer event mode. Programming this field is mandatory for all available DMA channels. This parameter can be a value of ***DMA_LL_EC_TRANSFER_EVENT_MODE***. This feature can be modified afterwards using unitary function **LL_DMA_SetTransferEventMode()**.

91.1.3 LL_DMA_InitNodeTypeDef

LL_DMA_InitNodeTypeDef is defined in the stm32n6xx_ll_dma.h

Data Fields

- ***uint32_t DestSecure***
- ***uint32_t DestAllocatedPort***
- ***uint32_t DestWordExchange***
- ***uint32_t DestHWordExchange***
- ***uint32_t DestByteExchange***
- ***uint32_t DestBurstLength***
- ***uint32_t DestIncMode***
- ***uint32_t DestDataWidth***
- ***uint32_t SrcSecure***
- ***uint32_t SrcAllocatedPort***
- ***uint32_t SrcByteExchange***
- ***uint32_t DataAlignment***
- ***uint32_t SrcBurstLength***
- ***uint32_t SrcIncMode***
- ***uint32_t SrcDataWidth***
- ***uint32_t TransferEventMode***
- ***uint32_t TriggerPolarity***
- ***uint32_t TriggerSelection***
- ***uint32_t TriggerMode***
- ***uint32_t BlkHWRequest***
- ***uint32_t Direction***
- ***uint32_t Request***
- ***uint32_t Mode***
- ***uint32_t BlkRptDestAddrUpdateMode***
- ***uint32_t BlkRptSrcAddrUpdateMode***
- ***uint32_t DestAddrUpdateMode***
- ***uint32_t SrcAddrUpdateMode***
- ***uint32_t BlkRptCount***
- ***uint32_t BlkDataLength***
- ***uint32_t SrcAddress***
- ***uint32_t DestAddress***
- ***uint32_t DestAddrOffset***
- ***uint32_t SrcAddrOffset***
- ***uint32_t BlkRptDestAddrOffset***
- ***uint32_t BlkRptSrcAddrOffset***
- ***uint32_t UpdateRegisters***

- ***uint32_t NodeType***

Field Documentation

- ***uint32_t LL_DMA_InitNodeTypeDef::DestSecure***
  This field specify the destination secure. This parameter can be a value of ***DMA_LL_EC_DESTINATION_SECURITY_ATTRIBUTE***.
- ***uint32_t LL_DMA_InitNodeTypeDef::DestAllocatedPort***
  This field specify the destination allocated port. This parameter can be a value of ***DMA_LL_EC_DESTINATION_ALLOCATED_PORT***.
- ***uint32_t LL_DMA_InitNodeTypeDef::DestWordExchange***
  This field specify the destination word exchange. This parameter can be a value of ***DMA_LL_EC_DEST_WORD_EXCHANGE***.
- ***uint32_t LL_DMA_InitNodeTypeDef::DestHWordExchange***
  This field specify the destination half word exchange. This parameter can be a value of ***DMA_LL_EC_DEST_HALFWORD_EXCHANGE***.
- ***uint32_t LL_DMA_InitNodeTypeDef::DestByteExchange***
  This field specify the destination byte exchange. This parameter can be a value of ***DMA_LL_EC_DEST_BYTE_EXCHANGE***.
- ***uint32_t LL_DMA_InitNodeTypeDef::DestBurstLength***
  This field specify the destination burst length of transfer in bytes. This parameter must be a value between Min_Data = 1 and Max_Data = 64.
- ***uint32_t LL_DMA_InitNodeTypeDef::DestIncMode***
  This field specify the destination increment mode. This parameter can be a value of ***DMA_LL_EC_DESTINATION_INCREMENT_MODE***.
- ***uint32_t LL_DMA_InitNodeTypeDef::DestDataWidth***
  This field specify the destination data width. This parameter can be a value of ***DMA_LL_EC_DESTINATION_DATA_WIDTH***.
- ***uint32_t LL_DMA_InitNodeTypeDef::SrcSecure***
  This field specify the source secure. This parameter can be a value of ***DMA_LL_EC_SOURCE_SECURITY_ATTRIBUTE***.
- ***uint32_t LL_DMA_InitNodeTypeDef::SrcAllocatedPort***
  This field specify the source allocated port. This parameter can be a value of ***DMA_LL_EC_SOURCE_ALLOCATED_PORT***.
- ***uint32_t LL_DMA_InitNodeTypeDef::SrcByteExchange***
  This field specify the source byte exchange. This parameter can be a value of ***DMA_LL_EC_SRC_BYTE_EXCHANGE***.
- ***uint32_t LL_DMA_InitNodeTypeDef::DataAlignment***
  This field specify the transfer data alignment. This parameter can be a value of ***DMA_LL_EC_DATA_ALIGNMENT***.
- ***uint32_t LL_DMA_InitNodeTypeDef::SrcBurstLength***
  This field specify the source burst length of transfer in bytes. This parameter must be a value between Min_Data = 1 and Max_Data = 64.
- ***uint32_t LL_DMA_InitNodeTypeDef::SrcIncMode***
  This field specify the source increment mode. This parameter can be a value of ***DMA_LL_EC_SOURCE_INCREMENT_MODE***.
- ***uint32_t LL_DMA_InitNodeTypeDef::SrcDataWidth***
  This field specify the source data width. This parameter can be a value of ***DMA_LL_EC_SOURCE_DATA_WIDTH***.
- ***uint32_t LL_DMA_InitNodeTypeDef::TransferEventMode***
  This field specify the transfer event mode. This parameter can be a value of ***DMA_LL_EC_TRANSFER_EVENT_MODE***.
- ***uint32_t LL_DMA_InitNodeTypeDef::TriggerPolarity***
  This field specify the trigger event polarity. This parameter can be a value of ***DMA_LL_EC_TRIGGER_POLARITY***.
- ***uint32_t LL_DMA_InitNodeTypeDef::TriggerSelection***
  This field specify the trigger event selection. This parameter can be a value of ***DMA_LL_EC_TRIGGER_SELECTION***.

- ***uint32_t LL_DMA_InitNodeTypeDef::TriggerMode***
  This field specify the trigger mode. This parameter can be a value of ***DMA_LL_EC_TRIGGER_MODE***.
- ***uint32_t LL_DMA_InitNodeTypeDef::BlkHWRequest***
  This field specify the hardware request unity. This parameter can be a value of ***DMA_LL_EC_BLKHW_REQUEST***.
- ***uint32_t LL_DMA_InitNodeTypeDef::Direction***
  This field specify the transfer direction. This parameter can be a value of ***DMA_LL_EC_TRANSFER_DIRECTION***.
- ***uint32_t LL_DMA_InitNodeTypeDef::Request***
  This field specify the peripheral request selection. This parameter can be a value of ***DMA_LL_EC_REQUEST_SELECTION***.
- ***uint32_t LL_DMA_InitNodeTypeDef::Mode***
  This field DMA Transfer Mode. This parameter can be a value of ***DMA_LL_TRANSFER_MODE***.
- ***uint32_t LL_DMA_InitNodeTypeDef::BlkRptDestAddrUpdateMode***
  This field specifies the block repeat destination address update mode. This parameter can be a value of ***DMA_LL_EC_BLK_RPT_DEST_ADDR_UPDATE_MODE***.
- ***uint32_t LL_DMA_InitNodeTypeDef::BlkRptSrcAddrUpdateMode***
  This field specifies the block repeat source address update mode. This parameter can be a value of ***DMA_LL_EC_BLK_RPT_SRC_ADDR_UPDATE_MODE***.
- ***uint32_t LL_DMA_InitNodeTypeDef::DestAddrUpdateMode***
  This field specify the Destination address update mode. This parameter can be a value of ***DMA_LL_EC_DEST_ADDR_UPDATE_MODE***.
- ***uint32_t LL_DMA_InitNodeTypeDef::SrcAddrUpdateMode***
  This field specify the Source address update mode. This parameter can be a value of ***DMA_LL_EC_SRC_ADDR_UPDATE_MODE***.
- ***uint32_t LL_DMA_InitNodeTypeDef::BlkRptCount***
  This field specify the number of repetitions of the current block. This parameter can be a value between 1 and 2048 Min_Data = 0 and Max_Data = 0x000007FF.
- ***uint32_t LL_DMA_InitNodeTypeDef::BlkDataLength***
  This field specify the length of a block transfer in bytes. This parameter must be a value between Min_Data = 0 and Max_Data = 0x0000FFFF.
- ***uint32_t LL_DMA_InitNodeTypeDef::SrcAddress***
  This field specify the transfer source address. This parameter must be a value between Min_Data = 0 and Max_Data = 0xFFFFFFFF.
- ***uint32_t LL_DMA_InitNodeTypeDef::DestAddress***
  This field specify the transfer destination address. This parameter must be a value between Min_Data = 0 and Max_Data = 0xFFFFFFFF.
- ***uint32_t LL_DMA_InitNodeTypeDef::DestAddrOffset***
  This field specifies the destination address offset. This parameter can be a value Between 0 to 0x00001FFF.
- ***uint32_t LL_DMA_InitNodeTypeDef::SrcAddrOffset***
  This field specifies the source address offset. This parameter can be a value Between 0 to 0x00001FFF.
- ***uint32_t LL_DMA_InitNodeTypeDef::BlkRptDestAddrOffset***
  This field specifies the block repeat destination address offset. This parameter can be a value Between 0 to 0x0000FFFF.
- ***uint32_t LL_DMA_InitNodeTypeDef::BlkRptSrcAddrOffset***
  This field specifies the block repeat source address offset. This parameter can be a value Between 0 to 0x0000FFFF.
- ***uint32_t LL_DMA_InitNodeTypeDef::UpdateRegisters***
  Specifies the linked list register update. This parameter can be a value of ***DMA_LL_EC_LINKEDLIST_REGISTER_UPDATE***.
- ***uint32_t LL_DMA_InitNodeTypeDef::NodeType***
  Specifies the node type to be created. This parameter can be a value of ***DMA_LL_EC_LINKEDLIST_NODE_TYPE***.

91.1.4 LL_DMA_LinkNodeTypeDef

LL_DMA_LinkNodeTypeDef is defined in the stm32n6xx_ll_dma.h

Data Fields

- ***__IO uint32_t LinkRegisters***

Field Documentation

- ***__IO uint32_t LL_DMA_LinkNodeTypeDef::LinkRegisters[8U]***

91.2 DMA Firmware driver API description

The following section lists the various functions of the DMA library.

91.2.1 LL DMA driver acronyms

Acronyms table :

- SRC Source
- DEST Destination
- ADDR Address
- ADDRS Addresses
- INC Increment / Incremented
- DEC Decrement / Decremented
- BLK Block
- RPT Repeat / Repeated
- TRIG Trigger

91.2.2 Detailed description of functions

LL_DMA_EnableChannel

Function name

__STATIC_INLINE void LL_DMA_EnableChannel (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Enable channel.

Parameters

- **DMAx:** DMAx Instance.
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR EN LL_DMA_EnableChannel

LL_DMA_DisableChannel

Function name

__STATIC_INLINE void LL_DMA_DisableChannel (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Disable channel.

Parameters

- **DMAx:** DMAx Instance.
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR EN LL_DMA_DisableChannel

LL_DMA_IsEnabledChannel

Function name

__STATIC_INLINE uint32_t LL_DMA_IsEnabledChannel (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Check if channel is enabled or disabled.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR EN LL_DMA_IsEnabledChannel

LL_DMA_ResetChannel

Function name

__STATIC_INLINE void LL_DMA_ResetChannel (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Reset channel.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR RESET LL_DMA_ResetChannel

LL_DMA_SuspendChannel

Function name

__STATIC_INLINE void LL_DMA_SuspendChannel (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Suspend channel.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR SUSP LL_DMA_SuspendChannel

LL_DMA_ResumeChannel

Function name

__STATIC_INLINE void LL_DMA_ResumeChannel (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Resume channel.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR SUSP LL_DMA_ResumeChannel

LL_DMA_IsSuspendedChannel

Function name

__STATIC_INLINE uint32_t LL_DMA_IsSuspendedChannel (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Check if channel is suspended.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR SUSP LL_DMA_IsSuspendedChannel

LL_DMA_SetLinkedListBaseAddr

Function name

__STATIC_INLINE void LL_DMA_SetLinkedListBaseAddr (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t LinkedListBaseAddr)

Function description

Set linked-list base address.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **LinkedListBaseAddr:** Between 0 to 0xFFFF0000 (where the 4 LSB bytes are always 0)

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CLBAR LBA LL_DMA_SetLinkedListBaseAddr

LL_DMA_GetLinkedListBaseAddr

Function name

__STATIC_INLINE uint32_t LL_DMA_GetLinkedListBaseAddr (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get linked-list base address.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **Value:** between 0 to 0xFFFF0000 (where the 4 LSB bytes are always 0)

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CLBAR LBA LL_DMA_GetLinkedListBaseAddr

LL_DMA_ConfigControl

Function name

__STATIC_INLINE void LL_DMA_ConfigControl (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t Configuration)

Function description

Configure all parameters linked to channel control.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **Configuration:** This parameter must be a combination of all the following values:
  - LL_DMA_LOW_PRIORITY_LOW_WEIGHT or LL_DMA_LOW_PRIORITY_MID_WEIGHT or LL_DMA_LOW_PRIORITY_HIGH_WEIGHT or LL_DMA_HIGH_PRIORITY
  - LL_DMA_LINK_ALLOCATED_PORT0 or LL_DMA_LINK_ALLOCATED_PORT1
  - LL_DMA_LSM_FULL_EXECUTION or LL_DMA_LSM_1LINK_EXECUTION

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR PRIO LL_DMA_ConfigControl
- CCR LAP LL_DMA_ConfigControl
- CCR LSM LL_DMA_ConfigControl

LL_DMA_SetChannelPriorityLevel

Function name

__STATIC_INLINE void LL_DMA_SetChannelPriorityLevel (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t Priority)

Function description

Set priority level.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **Priority:** This parameter can be one of the following values:
  - LL_DMA_LOW_PRIORITY_LOW_WEIGHT
  - LL_DMA_LOW_PRIORITY_MID_WEIGHT
  - LL_DMA_LOW_PRIORITY_HIGH_WEIGHT
  - LL_DMA_HIGH_PRIORITY

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR PRIO LL_DMA_SetChannelPriorityLevel

LL_DMA_GetChannelPriorityLevel

Function name

__STATIC_INLINE uint32_t LL_DMA_GetChannelPriorityLevel (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get Channel priority level.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_LOW_PRIORITY_LOW_WEIGHT
  - LL_DMA_LOW_PRIORITY_MID_WEIGHT
  - LL_DMA_LOW_PRIORITY_HIGH_WEIGHT
  - LL_DMA_HIGH_PRIORITY

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR PRIO LL_DMA_GetChannelPriorityLevel

LL_DMA_SetLinkAllocatedPort

Function name

__STATIC_INLINE void LL_DMA_SetLinkAllocatedPort (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t LinkAllocatedPort)

Function description

Set linked-list allocated port.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **LinkAllocatedPort:** This parameter can be one of the following values:
  - LL_DMA_LINK_ALLOCATED_PORT0
  - LL_DMA_LINK_ALLOCATED_PORT1

Return values

- **None.:**

Reference Manual to LL API cross reference:

- CCR LAP LL_DMA_SetLinkAllocatedPort

LL_DMA_GetLinkAllocatedPort

Function name

__STATIC_INLINE uint32_t LL_DMA_GetLinkAllocatedPort (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get linked-list allocated port.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_LINK_ALLOCATED_PORT0
  - LL_DMA_LINK_ALLOCATED_PORT1

Reference Manual to LL API cross reference:

- CCR LAP LL_DMA_GetLinkAllocatedPort

LL_DMA_SetLinkStepMode

Function name

__STATIC_INLINE void LL_DMA_SetLinkStepMode (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t LinkStepMode)

Function description

Set link step mode.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **LinkStepMode:** This parameter can be one of the following values:
  - LL_DMA_LSM_FULL_EXECUTION
  - LL_DMA_LSM_1LINK_EXECUTION

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR LSM LL_DMA_SetLinkStepMode

LL_DMA_GetLinkStepMode

Function name

__STATIC_INLINE uint32_t LL_DMA_GetLinkStepMode (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get Link step mode.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_LSM_FULL_EXECUTION
  - LL_DMA_LSM_1LINK_EXECUTION

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR LSM LL_DMA_GetLinkStepMode

LL_DMA_ConfigTransfer

Function name

__STATIC_INLINE void LL_DMA_ConfigTransfer (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t Configuration)

Function description

Configure data transfer.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **Configuration:** This parameter must be a combination of all the following values:
  - LL_DMA_DEST_ALLOCATED_PORT0 or LL_DMA_DEST_ALLOCATED_PORT1
  - LL_DMA_DEST_WORD_PRESERVE or LL_DMA_DEST_WORD_EXCHANGE
  - LL_DMA_DEST_HALFWORD_PRESERVE or LL_DMA_DEST_HALFWORD_EXCHANGE
  - LL_DMA_DEST_BYTE_PRESERVE or LL_DMA_DEST_BYTE_EXCHANGE
  - LL_DMA_SRC_BYTE_PRESERVE or LL_DMA_SRC_BYTE_EXCHANGE
  - LL_DMA_DEST_FIXED or LL_DMA_DEST_INCREMENT
  - LL_DMA_DEST_DATAWIDTH_BYTE or LL_DMA_DEST_DATAWIDTH_HALFWORD or LL_DMA_DEST_DATAWIDTH_WORD or LL_DMA_DEST_DATAWIDTH_DOUBLEWORD
  - LL_DMA_SRC_ALLOCATED_PORT0 or LL_DMA_SRC_ALLOCATED_PORT1
  - LL_DMA_DATA_ALIGN_ZEROPADD or LL_DMA_DATA_ALIGN_SIGNEXTPADD or LL_DMA_DATA_PACK_UNPACK
  - LL_DMA_SRC_FIXED or LL_DMA_SRC_INCREMENT
  - LL_DMA_SRC_DATAWIDTH_BYTE or LL_DMA_SRC_DATAWIDTH_HALFWORD or LL_DMA_SRC_DATAWIDTH_WORD or LL_DMA_SRC_DATAWIDTH_DOUBLEWORD

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR1 DAP LL_DMA_ConfigTransfer
- CTR1 DWX LL_DMA_ConfigTransfer
- CTR1 DHX LL_DMA_ConfigTransfer
- CTR1 DBX LL_DMA_ConfigTransfer
- CTR1 DINC LL_DMA_ConfigTransfer
- CTR1 SAP LL_DMA_ConfigTransfer
- CTR1 SBX LL_DMA_ConfigTransfer
- CTR1 PAM LL_DMA_ConfigTransfer
- CTR1 SINC LL_DMA_ConfigTransfer

LL_DMA_ConfigBurstLength

Function name

__STATIC_INLINE void LL_DMA_ConfigBurstLength (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t SrcBurstLength, uint32_t DestBurstLength)

Function description

Configure source and destination burst length.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **SrcBurstLength:** Between 1 to 64
- **DestBurstLength:** Between 1 to 64

Return values

- **None.:**

Reference Manual to LL API cross reference:

- CTR1 DBL_1 LL_DMA_SetDestBurstLength
-
- CTR1 SBL_1 LL_DMA_SetDestBurstLength

LL_DMA_SetStaticIsolation

Function name

__STATIC_INLINE void LL_DMA_SetStaticIsolation (DMA_TypeDef * DMAx, uint32_t Channel, uint32_t Cid)

Function description

Set the static isolation attribute (CID) of the DMA channel.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **Cid:** This parameter must be a combination of all the following values:
  - LL_DMA_CHANNEL_STATIC_CID_0
  - LL_DMA_CHANNEL_STATIC_CID_1
  - LL_DMA_CHANNEL_STATIC_CID_2
  - LL_DMA_CHANNEL_STATIC_CID_3
  - LL_DMA_CHANNEL_STATIC_CID_4
  - LL_DMA_CHANNEL_STATIC_CID_5
  - LL_DMA_CHANNEL_STATIC_CID_6

Return values

- **None.:**

Notes

- This API is used for HPDMA channels.

Reference Manual to LL API cross reference:

- CCIDCFGR SCID LL_DMA_SetStaticIsolation
-

LL_DMA_GetStaticIsolation

Function name

__STATIC_INLINE uint32_t LL_DMA_GetStaticIsolation (DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get the static isolation attribute (CID) of the DMA channel.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_CHANNEL_STATIC_CID_0
  - LL_DMA_CHANNEL_STATIC_CID_1
  - LL_DMA_CHANNEL_STATIC_CID_2
  - LL_DMA_CHANNEL_STATIC_CID_3
  - LL_DMA_CHANNEL_STATIC_CID_4
  - LL_DMA_CHANNEL_STATIC_CID_5
  - LL_DMA_CHANNEL_STATIC_CID_6

Notes

- This API is used for HPDMA channels.

Reference Manual to LL API cross reference:

- CCIDCFGR SCID LL_DMA_GetStaticIsolation
-

LL_DMA_EnableIsolation

Function name

__STATIC_INLINE void LL_DMA_EnableIsolation (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Enable the isolation (CID filtering) of the DMA channel.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for HPDMA channels.

Reference Manual to LL API cross reference:

- CCIDCFGR CFEN LL_DMA_EnableIsolation

LL_DMA_DisableIsolation

Function name

__STATIC_INLINE void LL_DMA_DisableIsolation (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Disable the isolation (CID filtering) of the DMA channel.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for HPDMA channels.

Reference Manual to LL API cross reference:

- CCIDCFGR CFEN LL_DMA_DisableIsolation

LL_DMA_IsEnabledIsolation

Function name

__STATIC_INLINE uint32_t LL_DMA_IsEnabledIsolation (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Check isolation (CID filtering) of the DMA channel.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for HPDMA channels.

Reference Manual to LL API cross reference:

- CCIDCFGR CFEN LL_DMA_IsEnabledIsolation

LL_DMA_ConfigChannelSecure

Function name

__STATIC_INLINE void LL_DMA_ConfigChannelSecure (DMA_TypeDef * DMAx, uint32_t Channel, uint32_t Configuration)

Function description

Configure all secure parameters linked to DMA channel.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **Configuration:** This parameter must be a combination of all the following values:
  - LL_DMA_CHANNEL_NSEC or LL_DMA_CHANNEL_SEC
  - LL_DMA_CHANNEL_SRC_NSEC or LL_DMA_CHANNEL_SRC_SEC
  - LL_DMA_CHANNEL_DEST_NSEC or LL_DMA_CHANNEL_DEST_SEC

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- SECCFGR SEC LL_DMA_ConfigChannelSecure
-
- CTR1 SSEC LL_DMA_ConfigChannelSecure
-
- CTR1 DSEC LL_DMA_ConfigChannelSecure

LL_DMA_EnableChannelDestSecure

Function name

__STATIC_INLINE void LL_DMA_EnableChannelDestSecure (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Enable security attribute of the DMA transfer to the destination.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR1 DSEC LL_DMA_EnableChannelDestSecure

LL_DMA_DisableChannelDestSecure

Function name

__STATIC_INLINE void LL_DMA_DisableChannelDestSecure (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Disable security attribute of the DMA transfer to the destination.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR1 DSEC LL_DMA_DisableChannelDestSecure

LL_DMA_IsEnabledChannelDestSecure

Function name

__STATIC_INLINE uint32_t LL_DMA_IsEnabledChannelDestSecure (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Check security attribute of the DMA transfer to the destination.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR1 DSEC LL_DMA_IsEnabledChannelDestSecure

LL_DMA_EnableChannelSrcSecure

Function name

__STATIC_INLINE void LL_DMA_EnableChannelSrcSecure (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Enable security attribute of the DMA transfer from the source.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR1 SSEC LL_DMA_EnableChannelSrcSecure

LL_DMA_DisableChannelSrcSecure

Function name

__STATIC_INLINE void LL_DMA_DisableChannelSrcSecure (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Disable security attribute of the DMA transfer from the source.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR1 SSEC LL_DMA_DisableChannelSrcSecure

LL_DMA_IsEnabledChannelSrcSecure

Function name

__STATIC_INLINE uint32_t LL_DMA_IsEnabledChannelSrcSecure (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Check security attribute of the DMA transfer from the source.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR1 SSEC LL_DMA_IsEnabledChannelSrcSecure

LL_DMA_SetDestAllocatedPort

Function name

__STATIC_INLINE void LL_DMA_SetDestAllocatedPort (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t DestAllocatedPort)

Function description

Set destination allocated port.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **DestAllocatedPort:** This parameter can be one of the following values:
  - LL_DMA_DEST_ALLOCATED_PORT0
  - LL_DMA_DEST_ALLOCATED_PORT1

Return values

- **None.:**

Reference Manual to LL API cross reference:

- CTR1 DAP LL_DMA_SetDestAllocatedPort

LL_DMA_GetDestAllocatedPort

Function name

__STATIC_INLINE uint32_t LL_DMA_GetDestAllocatedPort (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get destination allocated port.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_DEST_ALLOCATED_PORT0
  - LL_DMA_DEST_ALLOCATED_PORT1

Reference Manual to LL API cross reference:

- CTR1 DAP LL_DMA_GetDestAllocatedPort

LL_DMA_SetDestWordExchange

Function name

__STATIC_INLINE void LL_DMA_SetDestWordExchange (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t DestWordExchange)

Function description

Set destination word exchange.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **DestWordExchange:** This parameter can be one of the following values:
  - LL_DMA_DEST_WORD_PRESERVE
  - LL_DMA_DEST_WORD_EXCHANGE

Return values

- **None.:**

Reference Manual to LL API cross reference:

- CTR1 DWX LL_DMA_SetDestWordExchange

LL_DMA_GetDestWordExchange

Function name

__STATIC_INLINE uint32_t LL_DMA_GetDestWordExchange (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get destination word exchange.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_DEST_WORD_PRESERVE
  - LL_DMA_DEST_WORD_EXCHANGE

Reference Manual to LL API cross reference:

- CTR1 DWX LL_DMA_GetDestWordExchange

LL_DMA_SetDestHWordExchange

Function name

__STATIC_INLINE void LL_DMA_SetDestHWordExchange (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t DestHWordExchange)

Function description

Set destination half-word exchange.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **DestHWordExchange:** This parameter can be one of the following values:
  - LL_DMA_DEST_HALFWORD_PRESERVE
  - LL_DMA_DEST_HALFWORD_EXCHANGE

Return values

- **None.:**

Reference Manual to LL API cross reference:

- CTR1 DHX LL_DMA_SetDestHWordExchange

LL_DMA_GetDestHWordExchange

Function name

__STATIC_INLINE uint32_t LL_DMA_GetDestHWordExchange (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get destination half-word exchange.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_DEST_HALFWORD_PRESERVE
  - LL_DMA_DEST_HALFWORD_EXCHANGE

Reference Manual to LL API cross reference:

- CTR1 DHX LL_DMA_GetDestHWordExchange

LL_DMA_SetDestByteExchange

Function name

__STATIC_INLINE void LL_DMA_SetDestByteExchange (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t DestByteExchange)

Function description

Set destination byte exchange.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **DestByteExchange:** This parameter can be one of the following values:
  - LL_DMA_DEST_BYTE_PRESERVE
  - LL_DMA_DEST_BYTE_EXCHANGE

Return values

- **None.:**

Reference Manual to LL API cross reference:

- CTR1 DBX LL_DMA_SetDestByteExchange

LL_DMA_GetDestByteExchange

Function name

__STATIC_INLINE uint32_t LL_DMA_GetDestByteExchange (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get destination byte exchange.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_DEST_BYTE_PRESERVE
  - LL_DMA_DEST_BYTE_EXCHANGE

Reference Manual to LL API cross reference:

- CTR1 DBX LL_DMA_GetDestByteExchange

LL_DMA_SetSrcByteExchange

Function name

__STATIC_INLINE void LL_DMA_SetSrcByteExchange (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t SrcByteExchange)

Function description

Set source byte exchange.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **SrcByteExchange:** This parameter can be one of the following values:
  - LL_DMA_SRC_BYTE_PRESERVE
  - LL_DMA_SRC_BYTE_EXCHANGE

Return values

- **None.:**

Reference Manual to LL API cross reference:

- CTR1 SBX LL_DMA_SetSrcByteExchange

LL_DMA_GetSrcByteExchange

Function name

__STATIC_INLINE uint32_t LL_DMA_GetSrcByteExchange (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get source byte exchange.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_SRC_BYTE_PRESERVE
  - LL_DMA_SRC_BYTE_EXCHANGE

Reference Manual to LL API cross reference:

- CTR1 SBX LL_DMA_GetSrcByteExchange

LL_DMA_SetDestBurstLength

Function name

__STATIC_INLINE void LL_DMA_SetDestBurstLength (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t DestBurstLength)

Function description

Set destination burst length.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **DestBurstLength:** Between 1 to 64

Return values

- **None.:**

Reference Manual to LL API cross reference:

- CTR1 DBL_1 LL_DMA_SetDestBurstLength

LL_DMA_GetDestBurstLength

Function name

__STATIC_INLINE uint32_t LL_DMA_GetDestBurstLength (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get destination burst length.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **Between:** 1 to 64.

Reference Manual to LL API cross reference:

- CTR1 DBL_1 LL_DMA_GetDestBurstLength

LL_DMA_SetDestIncMode

Function name

__STATIC_INLINE void LL_DMA_SetDestIncMode (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t DestInc)

Function description

Set destination increment mode.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **DestInc:** This parameter can be one of the following values:
  - LL_DMA_DEST_FIXED
  - LL_DMA_DEST_INCREMENT

Return values

- **None.:**

Reference Manual to LL API cross reference:

- CTR1 DINC LL_DMA_SetDestIncMode

LL_DMA_GetDestIncMode

Function name

__STATIC_INLINE uint32_t LL_DMA_GetDestIncMode (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get destination increment mode.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_DEST_FIXED
  - LL_DMA_DEST_INCREMENT

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR1 DINC LL_DMA_GetDestIncMode

LL_DMA_SetDestDataWidth

Function name

__STATIC_INLINE void LL_DMA_SetDestDataWidth (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t DestDataWidth)

Function description

Set destination data width.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **DestDataWidth:** This parameter can be one of the following values:
  - LL_DMA_DEST_DATAWIDTH_BYTE
  - LL_DMA_DEST_DATAWIDTH_HALFWORD
  - LL_DMA_DEST_DATAWIDTH_WORD
  - LL_DMA_DEST_DATAWIDTH_DOUBLEWORD

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR1 DDW_LOG2 LL_DMA_SetDestDataWidth

LL_DMA_GetDestDataWidth

Function name

__STATIC_INLINE uint32_t LL_DMA_GetDestDataWidth (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get destination data width.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_DEST_DATAWIDTH_BYTE
  - LL_DMA_DEST_DATAWIDTH_HALFWORD
  - LL_DMA_DEST_DATAWIDTH_WORD
  - LL_DMA_DEST_DATAWIDTH_DOUBLEWORD

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR1 DDW_LOG2 LL_DMA_GetDestDataWidth

LL_DMA_SetSrcAllocatedPort

Function name

__STATIC_INLINE void LL_DMA_SetSrcAllocatedPort (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t SrcAllocatedPort)

Function description

Set source allocated port.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **SrcAllocatedPort:** This parameter can be one of the following values:
  - LL_DMA_SRC_ALLOCATED_PORT0
  - LL_DMA_SRC_ALLOCATED_PORT1

Return values

- **None.:**

Reference Manual to LL API cross reference:

- CTR1 SAP LL_DMA_SetSrcAllocatedPort

LL_DMA_GetSrcAllocatedPort

Function name

__STATIC_INLINE uint32_t LL_DMA_GetSrcAllocatedPort (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get source allocated port.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_SRC_ALLOCATED_PORT0
  - LL_DMA_SRC_ALLOCATED_PORT1

Reference Manual to LL API cross reference:

- CTR1 SAP LL_DMA_GetSrcAllocatedPort

LL_DMA_SetDataAlignment

Function name

__STATIC_INLINE void LL_DMA_SetDataAlignment (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t DataAlignment)

Function description

Set data alignment mode.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **DataAlignment:** This parameter can be one of the following values:
  - LL_DMA_DATA_ALIGN_ZEROPADD
  - LL_DMA_DATA_ALIGN_SIGNEXTPADD
  - LL_DMA_DATA_PACK_UNPACK

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR1 PAM LL_DMA_SetDataAlignment

LL_DMA_GetDataAlignment

Function name

__STATIC_INLINE uint32_t LL_DMA_GetDataAlignment (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get data alignment mode.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_DATA_ALIGN_ZEROPADD
  - LL_DMA_DATA_ALIGN_SIGNEXTPADD
  - LL_DMA_DATA_PACK_UNPACK

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR1 PAM LL_DMA_GetDataAlignment

LL_DMA_SetSrcBurstLength

Function name

__STATIC_INLINE void LL_DMA_SetSrcBurstLength (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t SrcBurstLength)

Function description

Set source burst length.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **SrcBurstLength:** Between 1 to 64

Return values

- **None.:**

Reference Manual to LL API cross reference:

- CTR1 SBL_1 LL_DMA_SetSrcBurstLength

LL_DMA_GetSrcBurstLength

Function name

__STATIC_INLINE uint32_t LL_DMA_GetSrcBurstLength (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get source burst length.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **Between:** 1 to 64
- **None.:**

Reference Manual to LL API cross reference:

- CTR1 SBL_1 LL_DMA_GetSrcBurstLength

LL_DMA_SetSrcIncMode

Function name

__STATIC_INLINE void LL_DMA_SetSrcIncMode (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t SrcInc)

Function description

Set source increment mode.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **SrcInc:** This parameter can be one of the following values:
  - LL_DMA_SRC_FIXED
  - LL_DMA_SRC_INCREMENT

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR1 SINC LL_DMA_SetSrcIncMode

LL_DMA_GetSrcIncMode

Function name

__STATIC_INLINE uint32_t LL_DMA_GetSrcIncMode (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get source increment mode.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_SRC_FIXED
  - LL_DMA_SRC_INCREMENT

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR1 SINC LL_DMA_GetSrcIncMode

LL_DMA_SetSrcDataWidth

Function name

__STATIC_INLINE void LL_DMA_SetSrcDataWidth (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t SrcDataWidth)

Function description

Set source data width.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **SrcDataWidth:** This parameter can be one of the following values:
  - LL_DMA_SRC_DATAWIDTH_BYTE
  - LL_DMA_SRC_DATAWIDTH_HALFWORD
  - LL_DMA_SRC_DATAWIDTH_WORD
  - LL_DMA_SRC_DATAWIDTH_DOUBLEWORD

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR1 SDW_LOG2 LL_DMA_SetSrcDataWidth

LL_DMA_GetSrcDataWidth

Function name

__STATIC_INLINE uint32_t LL_DMA_GetSrcDataWidth (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get Source Data width.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_SRC_DATAWIDTH_BYTE
  - LL_DMA_SRC_DATAWIDTH_HALFWORD
  - LL_DMA_SRC_DATAWIDTH_WORD
  - LL_DMA_SRC_DATAWIDTH_DOUBLEWORD

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR1 SDW_LOG2 LL_DMA_GetSrcDataWidth

LL_DMA_ConfigChannelTransfer

Function name

__STATIC_INLINE void LL_DMA_ConfigChannelTransfer (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t Configuration)

Function description

Configure channel transfer.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **Configuration:** This parameter must be a combination of all the following values:
  - LL_DMA_TCEM_BLK_TRANSFER or LL_DMA_TCEM_RPT_BLK_TRANSFER or LL_DMA_TCEM_EACH_LLITEM_TRANSFER or LL_DMA_TCEM_LAST_LLITEM_TRANSFER
  - LL_DMA_HWREQUEST_SINGLEBURST or LL_DMA_HWREQUEST_BLK
  - LL_DMA_TRIG_POLARITY_MASKED or LL_DMA_TRIG_POLARITY_RISING or LL_DMA_TRIG_POLARITY_FALLING
  - LL_DMA_TRIGM_BLK_TRANSFER or LL_DMA_TRIGM_RPT_BLK_TRANSFER or LL_DMA_TRIGM_LLI_LINK_TRANSFER or LL_DMA_TRIGM_SINGLBURST_TRANSFER
  - LL_DMA_DIRECTION_PERIPH_TO_MEMORY or LL_DMA_DIRECTION_MEMORY_TO_PERIPH or LL_DMA_DIRECTION_MEMORY_TO_MEMORY
  - LL_DMA_NORMAL or LL_DMA_PFCTRL

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR2 TCEM LL_DMA_ConfigChannelTransfer
- CTR2 TRIGPOL LL_DMA_ConfigChannelTransfer
- CTR2 TRIGM LL_DMA_ConfigChannelTransfer
- CTR2 BREQ LL_DMA_ConfigChannelTransfer
- CTR2 DREQ LL_DMA_ConfigChannelTransfer
- CTR2 SWREQ LL_DMA_ConfigChannelTransfer
- CTR2 PFREQ LL_DMA_ConfigChannelTransfer

LL_DMA_SetTransferEventMode

Function name

__STATIC_INLINE void LL_DMA_SetTransferEventMode (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t TransferEventMode)

Function description

Set transfer event mode.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **TransferEventMode:** This parameter can be one of the following values:
  - LL_DMA_TCEM_BLK_TRANSFER
  - LL_DMA_TCEM_RPT_BLK_TRANSFER
  - LL_DMA_TCEM_EACH_LLITEM_TRANSFER
  - LL_DMA_TCEM_LAST_LLITEM_TRANSFER

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR2 TCEM LL_DMA_SetTransferEventMode

LL_DMA_GetTransferEventMode

Function name

__STATIC_INLINE uint32_t LL_DMA_GetTransferEventMode (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get transfer event mode.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_TCEM_BLK_TRANSFER
  - LL_DMA_TCEM_RPT_BLK_TRANSFER
  - LL_DMA_TCEM_EACH_LLITEM_TRANSFER
  - LL_DMA_TCEM_LAST_LLITEM_TRANSFER

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR2 TCEM LL_DMA_GetTransferEventMode

LL_DMA_SetTriggerPolarity

Function name

__STATIC_INLINE void LL_DMA_SetTriggerPolarity (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t TriggerPolarity)

Function description

Set trigger polarity.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **TriggerPolarity:** This parameter can be one of the following values:
  - LL_DMA_TRIG_POLARITY_MASKED
  - LL_DMA_TRIG_POLARITY_RISING
  - LL_DMA_TRIG_POLARITY_FALLING

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR2 TRIGPOL LL_DMA_SetTriggerPolarity

LL_DMA_GetTriggerPolarity

Function name

__STATIC_INLINE uint32_t LL_DMA_GetTriggerPolarity (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get trigger polarity.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_TRIG_POLARITY_MASKED
  - LL_DMA_TRIG_POLARITY_RISING
  - LL_DMA_TRIG_POLARITY_FALLING

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR2 TRIGPOL LL_DMA_GetTriggerPolarity

LL_DMA_SetTriggerMode

Function name

__STATIC_INLINE void LL_DMA_SetTriggerMode (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t TriggerMode)

Function description

Set trigger Mode.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **TriggerMode:** This parameter can be one of the following values:
  - LL_DMA_TRIGM_BLK_TRANSFER
  - LL_DMA_TRIGM_RPT_BLK_TRANSFER (This value is allowed only for 2D addressing channels)
  - LL_DMA_TRIGM_LLI_LINK_TRANSFER
  - LL_DMA_TRIGM_SINGLBURST_TRANSFER

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR2 TRIGM LL_DMA_SetTriggerMode

LL_DMA_GetTriggerMode

Function name

__STATIC_INLINE uint32_t LL_DMA_GetTriggerMode (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get trigger Mode.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_TRIGM_BLK_TRANSFER
  - LL_DMA_TRIGM_RPT_BLK_TRANSFER (This value is allowed only for 2D addressing channels)
  - LL_DMA_TRIGM_LLI_LINK_TRANSFER
  - LL_DMA_TRIGM_SINGLBURST_TRANSFER

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR2 TRIGM LL_DMA_GetTriggerMode

LL_DMA_SetDataTransferDirection

Function name

__STATIC_INLINE void LL_DMA_SetDataTransferDirection (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t Direction)

Function description

Set destination hardware and software transfer request.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **Direction:** This parameter can be one of the following values:
  - LL_DMA_DIRECTION_PERIPH_TO_MEMORY
  - LL_DMA_DIRECTION_MEMORY_TO_PERIPH
  - LL_DMA_DIRECTION_MEMORY_TO_MEMORY

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR2 DREQ LL_DMA_SetDataTransferDirection
-
- CTR2 SWREQ LL_DMA_SetDataTransferDirection

LL_DMA_GetDataTransferDirection

Function name

__STATIC_INLINE uint32_t LL_DMA_GetDataTransferDirection (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get destination hardware and software transfer request.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_DIRECTION_PERIPH_TO_MEMORY
  - LL_DMA_DIRECTION_MEMORY_TO_PERIPH
  - LL_DMA_DIRECTION_MEMORY_TO_MEMORY

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR2 DREQ LL_DMA_GetDataTransferDirection
-
- CTR2 SWREQ LL_DMA_GetDataTransferDirection

LL_DMA_SetBlkHWRequest

Function name

__STATIC_INLINE void LL_DMA_SetBlkHWRequest (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t BlkHWRequest)

Function description

Set block hardware request.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **BlkHWRequest:** This parameter can be one of the following values:
  - LL_DMA_HWREQUEST_SINGLEBURST
  - LL_DMA_HWREQUEST_BLK

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR2 BREQ LL_DMA_SetBlkHWRequest
-

LL_DMA_GetBlkHWRequest

Function name

__STATIC_INLINE uint32_t LL_DMA_GetBlkHWRequest (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get block hardware request.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_HWREQUEST_SINGLEBURST
  - LL_DMA_HWREQUEST_BLK

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR2 BREQ LL_DMA_GetBlkHWRequest
-

LL_DMA_SetPeriphRequest

Function name

__STATIC_INLINE void LL_DMA_SetPeriphRequest (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t Request)

Function description

Set hardware request.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

- **Request:** This parameter can be one of the following values:
  - LL_HPDMA1_REQUEST_JPEG_RX
  - LL_HPDMA1_REQUEST_JPEG_TX
  - LL_HPDMA1_REQUEST_OCTOSPI1
  - LL_HPDMA1_REQUEST_OCTOSPI2
  - LL_HPDMA1_REQUEST_OCTOSPI3
  - LL_HPDMA1_REQUEST_FMC2_TXRX
  - LL_HPDMA1_REQUEST_FMC2_BCH
  - LL_HPDMA1_REQUEST_ADC1
  - LL_HPDMA1_REQUEST_ADC2
  - LL_HPDMA1_REQUEST_CRYP_IN
  - LL_HPDMA1_REQUEST_CRYP_OUT
  - LL_HPDMA1_REQUEST_SAES_OUT
  - LL_HPDMA1_REQUEST_SAES_IN
  - LL_HPDMA1_REQUEST_HASH_IN
  - LL_HPDMA1_REQUEST_TIM1_CH1
  - LL_HPDMA1_REQUEST_TIM1_CH2
  - LL_HPDMA1_REQUEST_TIM1_CH3
  - LL_HPDMA1_REQUEST_TIM1_CH4
  - LL_HPDMA1_REQUEST_TIM1_UP
  - LL_HPDMA1_REQUEST_TIM1_TRIG
  - LL_HPDMA1_REQUEST_TIM1_COM
  - LL_HPDMA1_REQUEST_TIM2_CH1
  - LL_HPDMA1_REQUEST_TIM2_CH2
  - LL_HPDMA1_REQUEST_TIM2_CH3
  - LL_HPDMA1_REQUEST_TIM2_CH4
  - LL_HPDMA1_REQUEST_TIM2_UP
  - LL_HPDMA1_REQUEST_TIM2_TRIG
  - LL_HPDMA1_REQUEST_TIM3_CH1
  - LL_HPDMA1_REQUEST_TIM3_CH2
  - LL_HPDMA1_REQUEST_TIM3_CH3
  - LL_HPDMA1_REQUEST_TIM3_CH4
  - LL_HPDMA1_REQUEST_TIM3_UP
  - LL_HPDMA1_REQUEST_TIM3_TRIG
  - LL_HPDMA1_REQUEST_TIM4_CH1
  - LL_HPDMA1_REQUEST_TIM4_CH2
  - LL_HPDMA1_REQUEST_TIM4_CH3
  - LL_HPDMA1_REQUEST_TIM4_CH4
  - LL_HPDMA1_REQUEST_TIM4_UP
  - LL_HPDMA1_REQUEST_TIM4_TRIG
  - LL_HPDMA1_REQUEST_TIM5_CH1
  - LL_HPDMA1_REQUEST_TIM5_CH2
  - LL_HPDMA1_REQUEST_TIM5_CH3
  - LL_HPDMA1_REQUEST_TIM5_CH4
  - LL_HPDMA1_REQUEST_TIM5_UP
  - LL_HPDMA1_REQUEST_TIM5_TRIG
  - LL_HPDMA1_REQUEST_TIM6_UP
  - LL_HPDMA1_REQUEST_TIM7_UP
  - LL_HPDMA1_REQUEST_TIM8_CH1
  - LL_HPDMA1_REQUEST_TIM8_CH2

- LL_HPDMA1_REQUEST_TIM8_CH3
- LL_HPDMA1_REQUEST_TIM8_CH4
- LL_HPDMA1_REQUEST_TIM8_UP
- LL_HPDMA1_REQUEST_TIM8_TRIG
- LL_HPDMA1_REQUEST_TIM8_COM
- LL_HPDMA1_REQUEST_TIM15_CH1
- LL_HPDMA1_REQUEST_TIM15_CH2
- LL_HPDMA1_REQUEST_TIM15_UP
- LL_HPDMA1_REQUEST_TIM15_TRIG
- LL_HPDMA1_REQUEST_TIM15_COM
- LL_HPDMA1_REQUEST_TIM16_CH1
- LL_HPDMA1_REQUEST_TIM16_UP
- LL_HPDMA1_REQUEST_TIM16_COM
- LL_HPDMA1_REQUEST_TIM17_CH1
- LL_HPDMA1_REQUEST_TIM17_UP
- LL_HPDMA1_REQUEST_TIM17_COM
- LL_HPDMA1_REQUEST_TIM18_CH1
- LL_HPDMA1_REQUEST_TIM18_UP
- LL_HPDMA1_REQUEST_TIM18_COM
- LL_HPDMA1_REQUEST_LPTIM1_IC1
- LL_HPDMA1_REQUEST_LPTIM1_IC2
- LL_HPDMA1_REQUEST_LPTIM1_UE
- LL_HPDMA1_REQUEST_LPTIM2_IC1
- LL_HPDMA1_REQUEST_LPTIM2_IC2
- LL_HPDMA1_REQUEST_LPTIM2_UE
- LL_HPDMA1_REQUEST_LPTIM3_IC1
- LL_HPDMA1_REQUEST_LPTIM3_IC2
- LL_HPDMA1_REQUEST_LPTIM3_UE
- LL_HPDMA1_REQUEST_SPI1_RX
- LL_HPDMA1_REQUEST_SPI1_TX
- LL_HPDMA1_REQUEST_SPI2_RX
- LL_HPDMA1_REQUEST_SPI2_TX
- LL_HPDMA1_REQUEST_SPI3_RX
- LL_HPDMA1_REQUEST_SPI3_TX
- LL_HPDMA1_REQUEST_SPI4_RX
- LL_HPDMA1_REQUEST_SPI4_TX
- LL_HPDMA1_REQUEST_SPI5_RX
- LL_HPDMA1_REQUEST_SPI5_TX
- LL_HPDMA1_REQUEST_SPI6_RX
- LL_HPDMA1_REQUEST_SPI6_TX
- LL_HPDMA1_REQUEST_SAI1_A
- LL_HPDMA1_REQUEST_SAI1_B
- LL_HPDMA1_REQUEST_SAI1_A
- LL_HPDMA1_REQUEST_SAI1_B
- LL_HPDMA1_REQUEST_I2C1_RX
- LL_HPDMA1_REQUEST_I2C1_TX
- LL_HPDMA1_REQUEST_I2C2_RX
- LL_HPDMA1_REQUEST_I2C2_TX
- LL_HPDMA1_REQUEST_I2C3_RX
- LL_HPDMA1_REQUEST_I2C3_TX

- LL_HPDMA1_REQUEST_I2C4_RX
- LL_HPDMA1_REQUEST_I2C4_TX
- LL_HPDMA1_REQUEST_I3C1_RX
- LL_HPDMA1_REQUEST_I3C1_TX
- LL_HPDMA1_REQUEST_I3C2_RX
- LL_HPDMA1_REQUEST_I3C2_TX
- LL_HPDMA1_REQUEST_USART1_RX
- LL_HPDMA1_REQUEST_USART1_TX
- LL_HPDMA1_REQUEST_USART2_RX
- LL_HPDMA1_REQUEST_USART2_TX
- LL_HPDMA1_REQUEST_USART3_RX
- LL_HPDMA1_REQUEST_USART3_TX
- LL_HPDMA1_REQUEST_UART4_RX
- LL_HPDMA1_REQUEST_UART4_TX
- LL_HPDMA1_REQUEST_UART5_RX
- LL_HPDMA1_REQUEST_UART5_TX
- LL_HPDMA1_REQUEST_USART6_RX
- LL_HPDMA1_REQUEST_USART6_TX
- LL_HPDMA1_REQUEST_UART7_RX
- LL_HPDMA1_REQUEST_UART7_TX
- LL_HPDMA1_REQUEST_UART8_RX
- LL_HPDMA1_REQUEST_UART8_TX
- LL_HPDMA1_REQUEST_UART9_RX
- LL_HPDMA1_REQUEST_UART9_TX
- LL_HPDMA1_REQUEST_USART10_RX
- LL_HPDMA1_REQUEST_USART10_TX
- LL_HPDMA1_REQUEST_LPUART1_RX
- LL_HPDMA1_REQUEST_LPUART1_TX
- LL_HPDMA1_REQUEST_SPDIFRX_CS
- LL_HPDMA1_REQUEST_SPDIFRX_DT
- LL_HPDMA1_REQUEST_ADF1_FLT0
- LL_HPDMA1_REQUEST_MDF1_FLT0
- LL_HPDMA1_REQUEST_MDF1_FLT1
- LL_HPDMA1_REQUEST_MDF1_FLT2
- LL_HPDMA1_REQUEST_MDF1_FLT3
- LL_HPDMA1_REQUEST_MDF1_FLT4
- LL_HPDMA1_REQUEST_MDF1_FLT5
- LL_HPDMA1_REQUEST_UCPD1_TX
- LL_HPDMA1_REQUEST_UCPD1_RX
- LL_HPDMA1_REQUEST_DCMI_PSSI
- LL_HPDMA1_REQUEST_I3C1_TC
- LL_HPDMA1_REQUEST_I3C1_RS
- LL_HPDMA1_REQUEST_I3C2_TC
- LL_HPDMA1_REQUEST_I3C2_RS
- LL_GPDMA1_REQUEST_JPEG_RX
- LL_GPDMA1_REQUEST_JPEG_TX
- LL_GPDMA1_REQUEST_OCTOSPI1
- LL_GPDMA1_REQUEST_OCTOSPI2
- LL_GPDMA1_REQUEST_OCTOSPI3
- LL_GPDMA1_REQUEST_FMC2_TXRX

- LL_GPDMA1_REQUEST_FMC2_BCH
- LL_GPDMA1_REQUEST_ADC1
- LL_GPDMA1_REQUEST_ADC2
- LL_GPDMA1_REQUEST_CRYP_IN
- LL_GPDMA1_REQUEST_CRYP_OUT
- LL_GPDMA1_REQUEST_SAES_OUT
- LL_GPDMA1_REQUEST_SAES_IN
- LL_GPDMA1_REQUEST_HASH_IN
- LL_GPDMA1_REQUEST_TIM1_CH1
- LL_GPDMA1_REQUEST_TIM1_CH2
- LL_GPDMA1_REQUEST_TIM1_CH3
- LL_GPDMA1_REQUEST_TIM1_CH4
- LL_GPDMA1_REQUEST_TIM1_UP
- LL_GPDMA1_REQUEST_TIM1_TRIG
- LL_GPDMA1_REQUEST_TIM1_COM
- LL_GPDMA1_REQUEST_TIM2_CH1
- LL_GPDMA1_REQUEST_TIM2_CH2
- LL_GPDMA1_REQUEST_TIM2_CH3
- LL_GPDMA1_REQUEST_TIM2_CH4
- LL_GPDMA1_REQUEST_TIM2_UP
- LL_GPDMA1_REQUEST_TIM2_TRIG
- LL_GPDMA1_REQUEST_TIM3_CH1
- LL_GPDMA1_REQUEST_TIM3_CH2
- LL_GPDMA1_REQUEST_TIM3_CH3
- LL_GPDMA1_REQUEST_TIM3_CH4
- LL_GPDMA1_REQUEST_TIM3_UP
- LL_GPDMA1_REQUEST_TIM3_TRIG
- LL_GPDMA1_REQUEST_TIM4_CH1
- LL_GPDMA1_REQUEST_TIM4_CH2
- LL_GPDMA1_REQUEST_TIM4_CH3
- LL_GPDMA1_REQUEST_TIM4_CH4
- LL_GPDMA1_REQUEST_TIM4_UP
- LL_GPDMA1_REQUEST_TIM4_TRIG
- LL_GPDMA1_REQUEST_TIM5_CH1
- LL_GPDMA1_REQUEST_TIM5_CH2
- LL_GPDMA1_REQUEST_TIM5_CH3
- LL_GPDMA1_REQUEST_TIM5_CH4
- LL_GPDMA1_REQUEST_TIM5_UP
- LL_GPDMA1_REQUEST_TIM5_TRIG
- LL_GPDMA1_REQUEST_TIM6_UP
- LL_GPDMA1_REQUEST_TIM7_UP
- LL_GPDMA1_REQUEST_TIM8_CH1
- LL_GPDMA1_REQUEST_TIM8_CH2
- LL_GPDMA1_REQUEST_TIM8_CH3
- LL_GPDMA1_REQUEST_TIM8_CH4
- LL_GPDMA1_REQUEST_TIM8_UP
- LL_GPDMA1_REQUEST_TIM8_TRIG
- LL_GPDMA1_REQUEST_TIM8_COM
- LL_GPDMA1_REQUEST_TIM15_CH1
- LL_GPDMA1_REQUEST_TIM15_CH2

- LL_GPDMA1_REQUEST_TIM15_UP
- LL_GPDMA1_REQUEST_TIM15_TRIG
- LL_GPDMA1_REQUEST_TIM15_COM
- LL_GPDMA1_REQUEST_TIM16_CH1
- LL_GPDMA1_REQUEST_TIM16_UP
- LL_GPDMA1_REQUEST_TIM16_COM
- LL_GPDMA1_REQUEST_TIM17_CH1
- LL_GPDMA1_REQUEST_TIM17_UP
- LL_GPDMA1_REQUEST_TIM17_COM
- LL_GPDMA1_REQUEST_TIM18_CH1
- LL_GPDMA1_REQUEST_TIM18_UP
- LL_GPDMA1_REQUEST_TIM18_COM
- LL_GPDMA1_REQUEST_LPTIM1_IC1
- LL_GPDMA1_REQUEST_LPTIM1_IC2
- LL_GPDMA1_REQUEST_LPTIM1_UE
- LL_GPDMA1_REQUEST_LPTIM2_IC1
- LL_GPDMA1_REQUEST_LPTIM2_IC2
- LL_GPDMA1_REQUEST_LPTIM2_UE
- LL_GPDMA1_REQUEST_LPTIM3_IC1
- LL_GPDMA1_REQUEST_LPTIM3_IC2
- LL_GPDMA1_REQUEST_LPTIM3_UE
- LL_GPDMA1_REQUEST_SPI1_RX
- LL_GPDMA1_REQUEST_SPI1_TX
- LL_GPDMA1_REQUEST_SPI2_RX
- LL_GPDMA1_REQUEST_SPI2_TX
- LL_GPDMA1_REQUEST_SPI3_RX
- LL_GPDMA1_REQUEST_SPI3_TX
- LL_GPDMA1_REQUEST_SPI4_RX
- LL_GPDMA1_REQUEST_SPI4_TX
- LL_GPDMA1_REQUEST_SPI5_RX
- LL_GPDMA1_REQUEST_SPI5_TX
- LL_GPDMA1_REQUEST_SPI6_RX
- LL_GPDMA1_REQUEST_SPI6_TX
- LL_GPDMA1_REQUEST_SAI1_A
- LL_GPDMA1_REQUEST_SAI1_B
- LL_GPDMA1_REQUEST_SAI1_A
- LL_GPDMA1_REQUEST_SAI1_B
- LL_GPDMA1_REQUEST_I2C1_RX
- LL_GPDMA1_REQUEST_I2C1_TX
- LL_GPDMA1_REQUEST_I2C2_RX
- LL_GPDMA1_REQUEST_I2C2_TX
- LL_GPDMA1_REQUEST_I2C3_RX
- LL_GPDMA1_REQUEST_I2C3_TX
- LL_GPDMA1_REQUEST_I2C4_RX
- LL_GPDMA1_REQUEST_I2C4_TX
- LL_GPDMA1_REQUEST_I3C1_RX
- LL_GPDMA1_REQUEST_I3C1_TX
- LL_GPDMA1_REQUEST_I3C2_RX
- LL_GPDMA1_REQUEST_I3C2_TX
- LL_GPDMA1_REQUEST_USART1_RX

- LL_GPDMA1_REQUEST_USART1_TX
- LL_GPDMA1_REQUEST_USART2_RX
- LL_GPDMA1_REQUEST_USART2_TX
- LL_GPDMA1_REQUEST_USART3_RX
- LL_GPDMA1_REQUEST_USART3_TX
- LL_GPDMA1_REQUEST_UART4_RX
- LL_GPDMA1_REQUEST_UART4_TX
- LL_GPDMA1_REQUEST_UART5_RX
- LL_GPDMA1_REQUEST_UART5_TX
- LL_GPDMA1_REQUEST_USART6_RX
- LL_GPDMA1_REQUEST_USART6_TX
- LL_GPDMA1_REQUEST_UART7_RX
- LL_GPDMA1_REQUEST_UART7_TX
- LL_GPDMA1_REQUEST_UART8_RX
- LL_GPDMA1_REQUEST_UART8_TX
- LL_GPDMA1_REQUEST_UART9_RX
- LL_GPDMA1_REQUEST_UART9_TX
- LL_GPDMA1_REQUEST_USART10_RX
- LL_GPDMA1_REQUEST_USART10_TX
- LL_GPDMA1_REQUEST_LPUART1_RX
- LL_GPDMA1_REQUEST_LPUART1_TX
- LL_GPDMA1_REQUEST_SPDIFRX_DT
- LL_GPDMA1_REQUEST_SPDIFRX_CS
- LL_GPDMA1_REQUEST_ADF1_FLT0
- LL_GPDMA1_REQUEST_MDF1_FLT0
- LL_GPDMA1_REQUEST_MDF1_FLT1
- LL_GPDMA1_REQUEST_MDF1_FLT2
- LL_GPDMA1_REQUEST_MDF1_FLT3
- LL_GPDMA1_REQUEST_MDF1_FLT4
- LL_GPDMA1_REQUEST_MDF1_FLT5
- LL_GPDMA1_REQUEST_UCPD1_TX
- LL_GPDMA1_REQUEST_UCPD1_RX
- LL_GPDMA1_REQUEST_DCMI_PSSI
- LL_GPDMA1_REQUEST_I3C1_TC
- LL_GPDMA1_REQUEST_I3C1_RS
- LL_GPDMA1_REQUEST_I3C2_TC
- LL_GPDMA1_REQUEST_I3C2_RS

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR2 REQSEL LL_DMA_SetPeriphRequest

LL_DMA_GetPeriphRequest

Function name

__STATIC_INLINE uint32_t LL_DMA_GetPeriphRequest (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get hardware request.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **Returned:** value can be one of the following values:
  - LL_HPDMA1_REQUEST_JPEG_RX
  - LL_HPDMA1_REQUEST_JPEG_TX
  - LL_HPDMA1_REQUEST_OCTOSPI1
  - LL_HPDMA1_REQUEST_OCTOSPI2
  - LL_HPDMA1_REQUEST_OCTOSPI3
  - LL_HPDMA1_REQUEST_FMC2_TXRX
  - LL_HPDMA1_REQUEST_FMC2_BCH
  - LL_HPDMA1_REQUEST_ADC1
  - LL_HPDMA1_REQUEST_ADC2
  - LL_HPDMA1_REQUEST_CRYP_IN
  - LL_HPDMA1_REQUEST_CRYP_OUT
  - LL_HPDMA1_REQUEST_SAES_OUT
  - LL_HPDMA1_REQUEST_SAES_IN
  - LL_HPDMA1_REQUEST_HASH_IN
  - LL_HPDMA1_REQUEST_TIM1_CH1
  - LL_HPDMA1_REQUEST_TIM1_CH2
  - LL_HPDMA1_REQUEST_TIM1_CH3
  - LL_HPDMA1_REQUEST_TIM1_CH4
  - LL_HPDMA1_REQUEST_TIM1_UP
  - LL_HPDMA1_REQUEST_TIM1_TRIG
  - LL_HPDMA1_REQUEST_TIM1_COM
  - LL_HPDMA1_REQUEST_TIM2_CH1
  - LL_HPDMA1_REQUEST_TIM2_CH2
  - LL_HPDMA1_REQUEST_TIM2_CH3
  - LL_HPDMA1_REQUEST_TIM2_CH4
  - LL_HPDMA1_REQUEST_TIM2_UP
  - LL_HPDMA1_REQUEST_TIM2_TRIG
  - LL_HPDMA1_REQUEST_TIM3_CH1
  - LL_HPDMA1_REQUEST_TIM3_CH2
  - LL_HPDMA1_REQUEST_TIM3_CH3
  - LL_HPDMA1_REQUEST_TIM3_CH4
  - LL_HPDMA1_REQUEST_TIM3_UP
  - LL_HPDMA1_REQUEST_TIM3_TRIG
  - LL_HPDMA1_REQUEST_TIM4_CH1
  - LL_HPDMA1_REQUEST_TIM4_CH2
  - LL_HPDMA1_REQUEST_TIM4_CH3
  - LL_HPDMA1_REQUEST_TIM4_CH4
  - LL_HPDMA1_REQUEST_TIM4_UP
  - LL_HPDMA1_REQUEST_TIM4_TRIG
  - LL_HPDMA1_REQUEST_TIM5_CH1
  - LL_HPDMA1_REQUEST_TIM5_CH2
  - LL_HPDMA1_REQUEST_TIM5_CH3
  - LL_HPDMA1_REQUEST_TIM5_CH4
  - LL_HPDMA1_REQUEST_TIM5_UP
  - LL_HPDMA1_REQUEST_TIM5_TRIG
  - LL_HPDMA1_REQUEST_TIM6_UP
  - LL_HPDMA1_REQUEST_TIM7_UP
  - LL_HPDMA1_REQUEST_TIM8_CH1

- LL_HPDMA1_REQUEST_TIM8_CH2
- LL_HPDMA1_REQUEST_TIM8_CH3
- LL_HPDMA1_REQUEST_TIM8_CH4
- LL_HPDMA1_REQUEST_TIM8_UP
- LL_HPDMA1_REQUEST_TIM8_TRIG
- LL_HPDMA1_REQUEST_TIM8_COM
- LL_HPDMA1_REQUEST_TIM15_CH1
- LL_HPDMA1_REQUEST_TIM15_CH2
- LL_HPDMA1_REQUEST_TIM15_UP
- LL_HPDMA1_REQUEST_TIM15_TRIG
- LL_HPDMA1_REQUEST_TIM15_COM
- LL_HPDMA1_REQUEST_TIM16_CH1
- LL_HPDMA1_REQUEST_TIM16_UP
- LL_HPDMA1_REQUEST_TIM16_COM
- LL_HPDMA1_REQUEST_TIM17_CH1
- LL_HPDMA1_REQUEST_TIM17_UP
- LL_HPDMA1_REQUEST_TIM17_COM
- LL_HPDMA1_REQUEST_TIM18_CH1
- LL_HPDMA1_REQUEST_TIM18_UP
- LL_HPDMA1_REQUEST_TIM18_COM
- LL_HPDMA1_REQUEST_LPTIM1_IC1
- LL_HPDMA1_REQUEST_LPTIM1_IC2
- LL_HPDMA1_REQUEST_LPTIM1_UE
- LL_HPDMA1_REQUEST_LPTIM2_IC1
- LL_HPDMA1_REQUEST_LPTIM2_IC2
- LL_HPDMA1_REQUEST_LPTIM2_UE
- LL_HPDMA1_REQUEST_LPTIM3_IC1
- LL_HPDMA1_REQUEST_LPTIM3_IC2
- LL_HPDMA1_REQUEST_LPTIM3_UE
- LL_HPDMA1_REQUEST_SPI1_RX
- LL_HPDMA1_REQUEST_SPI1_TX
- LL_HPDMA1_REQUEST_SPI2_RX
- LL_HPDMA1_REQUEST_SPI2_TX
- LL_HPDMA1_REQUEST_SPI3_RX
- LL_HPDMA1_REQUEST_SPI3_TX
- LL_HPDMA1_REQUEST_SPI4_RX
- LL_HPDMA1_REQUEST_SPI4_TX
- LL_HPDMA1_REQUEST_SPI5_RX
- LL_HPDMA1_REQUEST_SPI5_TX
- LL_HPDMA1_REQUEST_SPI6_RX
- LL_HPDMA1_REQUEST_SPI6_TX
- LL_HPDMA1_REQUEST_SAI1_A
- LL_HPDMA1_REQUEST_SAI1_B
- LL_HPDMA1_REQUEST_SAI1_A
- LL_HPDMA1_REQUEST_SAI1_B
- LL_HPDMA1_REQUEST_I2C1_RX
- LL_HPDMA1_REQUEST_I2C1_TX
- LL_HPDMA1_REQUEST_I2C2_RX
- LL_HPDMA1_REQUEST_I2C2_TX
- LL_HPDMA1_REQUEST_I2C3_RX

- LL_HPDMA1_REQUEST_I2C3_TX
- LL_HPDMA1_REQUEST_I2C4_RX
- LL_HPDMA1_REQUEST_I2C4_TX
- LL_HPDMA1_REQUEST_I3C1_RX
- LL_HPDMA1_REQUEST_I3C1_TX
- LL_HPDMA1_REQUEST_I3C2_RX
- LL_HPDMA1_REQUEST_I3C2_TX
- LL_HPDMA1_REQUEST_USART1_RX
- LL_HPDMA1_REQUEST_USART1_TX
- LL_HPDMA1_REQUEST_USART2_RX
- LL_HPDMA1_REQUEST_USART2_TX
- LL_HPDMA1_REQUEST_USART3_RX
- LL_HPDMA1_REQUEST_USART3_TX
- LL_HPDMA1_REQUEST_UART4_RX
- LL_HPDMA1_REQUEST_UART4_TX
- LL_HPDMA1_REQUEST_UART5_RX
- LL_HPDMA1_REQUEST_UART5_TX
- LL_HPDMA1_REQUEST_USART6_RX
- LL_HPDMA1_REQUEST_USART6_TX
- LL_HPDMA1_REQUEST_UART7_RX
- LL_HPDMA1_REQUEST_UART7_TX
- LL_HPDMA1_REQUEST_UART8_RX
- LL_HPDMA1_REQUEST_UART8_TX
- LL_HPDMA1_REQUEST_UART9_RX
- LL_HPDMA1_REQUEST_UART9_TX
- LL_HPDMA1_REQUEST_USART10_RX
- LL_HPDMA1_REQUEST_USART10_TX
- LL_HPDMA1_REQUEST_LPUART1_RX
- LL_HPDMA1_REQUEST_LPUART1_TX
- LL_HPDMA1_REQUEST_SPDIFRX_CS
- LL_HPDMA1_REQUEST_SPDIFRX_DT
- LL_HPDMA1_REQUEST_ADF1_FLT0
- LL_HPDMA1_REQUEST_MDF1_FLT0
- LL_HPDMA1_REQUEST_MDF1_FLT1
- LL_HPDMA1_REQUEST_MDF1_FLT2
- LL_HPDMA1_REQUEST_MDF1_FLT3
- LL_HPDMA1_REQUEST_MDF1_FLT4
- LL_HPDMA1_REQUEST_MDF1_FLT5
- LL_HPDMA1_REQUEST_UCPD1_TX
- LL_HPDMA1_REQUEST_UCPD1_RX
- LL_HPDMA1_REQUEST_DCMI_PSSI
- LL_HPDMA1_REQUEST_I3C1_TC
- LL_HPDMA1_REQUEST_I3C1_RS
- LL_HPDMA1_REQUEST_I3C2_TC
- LL_HPDMA1_REQUEST_I3C2_RS
- LL_GPDMA1_REQUEST_JPEG_RX
- LL_GPDMA1_REQUEST_JPEG_TX
- LL_GPDMA1_REQUEST_OCTOSPI1
- LL_GPDMA1_REQUEST_OCTOSPI2
- LL_GPDMA1_REQUEST_OCTOSPI3

- LL_GPDMA1_REQUEST_FMC2_TXRX
- LL_GPDMA1_REQUEST_FMC2_BCH
- LL_GPDMA1_REQUEST_ADC1
- LL_GPDMA1_REQUEST_ADC2
- LL_GPDMA1_REQUEST_CRYP_IN
- LL_GPDMA1_REQUEST_CRYP_OUT
- LL_GPDMA1_REQUEST_SAES_OUT
- LL_GPDMA1_REQUEST_SAES_IN
- LL_GPDMA1_REQUEST_HASH_IN
- LL_GPDMA1_REQUEST_TIM1_CH1
- LL_GPDMA1_REQUEST_TIM1_CH2
- LL_GPDMA1_REQUEST_TIM1_CH3
- LL_GPDMA1_REQUEST_TIM1_CH4
- LL_GPDMA1_REQUEST_TIM1_UP
- LL_GPDMA1_REQUEST_TIM1_TRIG
- LL_GPDMA1_REQUEST_TIM1_COM
- LL_GPDMA1_REQUEST_TIM2_CH1
- LL_GPDMA1_REQUEST_TIM2_CH2
- LL_GPDMA1_REQUEST_TIM2_CH3
- LL_GPDMA1_REQUEST_TIM2_CH4
- LL_GPDMA1_REQUEST_TIM2_UP
- LL_GPDMA1_REQUEST_TIM2_TRIG
- LL_GPDMA1_REQUEST_TIM3_CH1
- LL_GPDMA1_REQUEST_TIM3_CH2
- LL_GPDMA1_REQUEST_TIM3_CH3
- LL_GPDMA1_REQUEST_TIM3_CH4
- LL_GPDMA1_REQUEST_TIM3_UP
- LL_GPDMA1_REQUEST_TIM3_TRIG
- LL_GPDMA1_REQUEST_TIM4_CH1
- LL_GPDMA1_REQUEST_TIM4_CH2
- LL_GPDMA1_REQUEST_TIM4_CH3
- LL_GPDMA1_REQUEST_TIM4_CH4
- LL_GPDMA1_REQUEST_TIM4_UP
- LL_GPDMA1_REQUEST_TIM4_TRIG
- LL_GPDMA1_REQUEST_TIM5_CH1
- LL_GPDMA1_REQUEST_TIM5_CH2
- LL_GPDMA1_REQUEST_TIM5_CH3
- LL_GPDMA1_REQUEST_TIM5_CH4
- LL_GPDMA1_REQUEST_TIM5_UP
- LL_GPDMA1_REQUEST_TIM5_TRIG
- LL_GPDMA1_REQUEST_TIM6_UP
- LL_GPDMA1_REQUEST_TIM7_UP
- LL_GPDMA1_REQUEST_TIM8_CH1
- LL_GPDMA1_REQUEST_TIM8_CH2
- LL_GPDMA1_REQUEST_TIM8_CH3
- LL_GPDMA1_REQUEST_TIM8_CH4
- LL_GPDMA1_REQUEST_TIM8_UP
- LL_GPDMA1_REQUEST_TIM8_TRIG
- LL_GPDMA1_REQUEST_TIM8_COM
- LL_GPDMA1_REQUEST_TIM15_CH1

- LL_GPDMA1_REQUEST_TIM15_CH2
- LL_GPDMA1_REQUEST_TIM15_UP
- LL_GPDMA1_REQUEST_TIM15_TRIG
- LL_GPDMA1_REQUEST_TIM15_COM
- LL_GPDMA1_REQUEST_TIM16_CH1
- LL_GPDMA1_REQUEST_TIM16_UP
- LL_GPDMA1_REQUEST_TIM16_COM
- LL_GPDMA1_REQUEST_TIM17_CH1
- LL_GPDMA1_REQUEST_TIM17_UP
- LL_GPDMA1_REQUEST_TIM17_COM
- LL_GPDMA1_REQUEST_TIM18_CH1
- LL_GPDMA1_REQUEST_TIM18_UP
- LL_GPDMA1_REQUEST_TIM18_COM
- LL_GPDMA1_REQUEST_LPTIM1_IC1
- LL_GPDMA1_REQUEST_LPTIM1_IC2
- LL_GPDMA1_REQUEST_LPTIM1_UE
- LL_GPDMA1_REQUEST_LPTIM2_IC1
- LL_GPDMA1_REQUEST_LPTIM2_IC2
- LL_GPDMA1_REQUEST_LPTIM2_UE
- LL_GPDMA1_REQUEST_LPTIM3_IC1
- LL_GPDMA1_REQUEST_LPTIM3_IC2
- LL_GPDMA1_REQUEST_LPTIM3_UE
- LL_GPDMA1_REQUEST_SPI1_RX
- LL_GPDMA1_REQUEST_SPI1_TX
- LL_GPDMA1_REQUEST_SPI2_RX
- LL_GPDMA1_REQUEST_SPI2_TX
- LL_GPDMA1_REQUEST_SPI3_RX
- LL_GPDMA1_REQUEST_SPI3_TX
- LL_GPDMA1_REQUEST_SPI4_RX
- LL_GPDMA1_REQUEST_SPI4_TX
- LL_GPDMA1_REQUEST_SPI5_RX
- LL_GPDMA1_REQUEST_SPI5_TX
- LL_GPDMA1_REQUEST_SPI6_RX
- LL_GPDMA1_REQUEST_SPI6_TX
- LL_GPDMA1_REQUEST_SAI1_A
- LL_GPDMA1_REQUEST_SAI1_B
- LL_GPDMA1_REQUEST_SAI1_A
- LL_GPDMA1_REQUEST_SAI1_B
- LL_GPDMA1_REQUEST_I2C1_RX
- LL_GPDMA1_REQUEST_I2C1_TX
- LL_GPDMA1_REQUEST_I2C2_RX
- LL_GPDMA1_REQUEST_I2C2_TX
- LL_GPDMA1_REQUEST_I2C3_RX
- LL_GPDMA1_REQUEST_I2C3_TX
- LL_GPDMA1_REQUEST_I2C4_RX
- LL_GPDMA1_REQUEST_I2C4_TX
- LL_GPDMA1_REQUEST_I3C1_RX
- LL_GPDMA1_REQUEST_I3C1_TX
- LL_GPDMA1_REQUEST_I3C2_RX
- LL_GPDMA1_REQUEST_I3C2_TX

- LL_GPDMA1_REQUEST_USART1_RX
- LL_GPDMA1_REQUEST_USART1_TX
- LL_GPDMA1_REQUEST_USART2_RX
- LL_GPDMA1_REQUEST_USART2_TX
- LL_GPDMA1_REQUEST_USART3_RX
- LL_GPDMA1_REQUEST_USART3_TX
- LL_GPDMA1_REQUEST_UART4_RX
- LL_GPDMA1_REQUEST_UART4_TX
- LL_GPDMA1_REQUEST_UART5_RX
- LL_GPDMA1_REQUEST_UART5_TX
- LL_GPDMA1_REQUEST_USART6_RX
- LL_GPDMA1_REQUEST_USART6_TX
- LL_GPDMA1_REQUEST_UART7_RX
- LL_GPDMA1_REQUEST_UART7_TX
- LL_GPDMA1_REQUEST_UART8_RX
- LL_GPDMA1_REQUEST_UART8_TX
- LL_GPDMA1_REQUEST_UART9_RX
- LL_GPDMA1_REQUEST_UART9_TX
- LL_GPDMA1_REQUEST_USART10_RX
- LL_GPDMA1_REQUEST_USART10_TX
- LL_GPDMA1_REQUEST_LPUART1_RX
- LL_GPDMA1_REQUEST_LPUART1_TX
- LL_GPDMA1_REQUEST_SPDIFRX_DT
- LL_GPDMA1_REQUEST_SPDIFRX_CS
- LL_GPDMA1_REQUEST_ADF1_FLT0
- LL_GPDMA1_REQUEST_MDF1_FLT0
- LL_GPDMA1_REQUEST_MDF1_FLT1
- LL_GPDMA1_REQUEST_MDF1_FLT2
- LL_GPDMA1_REQUEST_MDF1_FLT3
- LL_GPDMA1_REQUEST_MDF1_FLT4
- LL_GPDMA1_REQUEST_MDF1_FLT5
- LL_GPDMA1_REQUEST_UCPD1_TX
- LL_GPDMA1_REQUEST_UCPD1_RX
- LL_GPDMA1_REQUEST_DCMI_PSSI
- LL_GPDMA1_REQUEST_I3C1_TC
- LL_GPDMA1_REQUEST_I3C1_RS
- LL_GPDMA1_REQUEST_I3C2_TC
- LL_GPDMA1_REQUEST_I3C2_RS

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR2 REQSEL LL_DMA_GetPeriphRequest

LL_DMA_SetHWTrigger

Function name

__STATIC_INLINE void LL_DMA_SetHWTrigger (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t Trigger)

Function description

Set hardware trigger.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

- **Trigger:** This parameter can be one of the following values:
  - LL_HPDMA1_TRIGGER_DCMIPP_P0_FEND
  - LL_HPDMA1_TRIGGER_DCMIPP_P0_LEND
  - LL_HPDMA1_TRIGGER_DCMIPP_P0_HSYNC
  - LL_HPDMA1_TRIGGER_DCMIPP_P0_VSYNC
  - LL_HPDMA1_TRIGGER_DCMIPP_P1_FEND
  - LL_HPDMA1_TRIGGER_DCMIPP_P1_LEND
  - LL_HPDMA1_TRIGGER_DCMIPP_P1_HSYNC
  - LL_HPDMA1_TRIGGER_DCMIPP_P1_VSYNC
  - LL_HPDMA1_TRIGGER_DCMIPP_P2_FEND
  - LL_HPDMA1_TRIGGER_DCMIPP_P2_LEND
  - LL_HPDMA1_TRIGGER_DCMIPP_P2_HSYNC
  - LL_HPDMA1_TRIGGER_DCMIPP_P2_VSYNC
  - LL_HPDMA1_TRIGGER_DMA2D_CTC
  - LL_HPDMA1_TRIGGER_DMA2D_TC
  - LL_HPDMA1_TRIGGER_DMA2D_TW
  - LL_HPDMA1_TRIGGER_JPEG_EOC
  - LL_HPDMA1_TRIGGER_JPEG_IFNF
  - LL_HPDMA1_TRIGGER_JPEG_IFT
  - LL_HPDMA1_TRIGGER_JPEG_OFNE
  - LL_HPDMA1_TRIGGER_JPEG_OFT
  - LL_HPDMA1_TRIGGER_LCD_LI
  - LL_HPDMA1_TRIGGER_GPU2D1_GP_0
  - LL_HPDMA1_TRIGGER_GPU2D1_GP_1
  - LL_HPDMA1_TRIGGER_GPU2D1_GP_2
  - LL_HPDMA1_TRIGGER_GPU2D1_GP_3
  - LL_HPDMA1_TRIGGER_GFXTIM_3
  - LL_HPDMA1_TRIGGER_GFXTIM_2
  - LL_HPDMA1_TRIGGER_GFXTIM_1
  - LL_HPDMA1_TRIGGER_GFXTIM_0
  - LL_HPDMA1_TRIGGER_LPTIM1_CH1
  - LL_HPDMA1_TRIGGER_LPTIM1_CH2
  - LL_HPDMA1_TRIGGER_LPTIM2_CH1
  - LL_HPDMA1_TRIGGER_LPTIM2_CH2
  - LL_HPDMA1_TRIGGER_LPTIM3_CH1
  - LL_HPDMA1_TRIGGER_LPTIM3_CH2
  - LL_HPDMA1_TRIGGER_LPTIM4_OUT
  - LL_HPDMA1_TRIGGER_LPTIM5_OUT
  - LL_HPDMA1_TRIGGER_RTC_WKUP
  - LL_HPDMA1_TRIGGER_LPUART1_IT_R_WUP_ASYNC
  - LL_HPDMA1_TRIGGER_LPUART1_IT_T_WUP_ASYNC
  - LL_HPDMA1_TRIGGER_SPI6_IT_OR_SPI6_AIT_SYNC
  - LL_HPDMA1_TRIGGER_TIM1_TRGO
  - LL_HPDMA1_TRIGGER_TIM1_TRGO2
  - LL_HPDMA1_TRIGGER_TIM2_TRGO
  - LL_HPDMA1_TRIGGER_TIM3_TRGO
  - LL_HPDMA1_TRIGGER_TIM4_TRGO
  - LL_HPDMA1_TRIGGER_TIM5_TRGO
  - LL_HPDMA1_TRIGGER_TIM6_TRGO
  - LL_HPDMA1_TRIGGER_TIM7_TRGO

- LL_HPDMA1_TRIGGER_TIM8_TRGO
- LL_HPDMA1_TRIGGER_TIM8_TRGO2
- LL_HPDMA1_TRIGGER_TIM12_TRGO
- LL_HPDMA1_TRIGGER_TIM15_TRGO
- LL_HPDMA1_TRIGGER_HPDMA1_CH0_TCF
- LL_HPDMA1_TRIGGER_HPDMA1_CH1_TCF
- LL_HPDMA1_TRIGGER_HPDMA1_CH2_TCF
- LL_HPDMA1_TRIGGER_HPDMA1_CH3_TCF
- LL_HPDMA1_TRIGGER_HPDMA1_CH4_TCF
- LL_HPDMA1_TRIGGER_HPDMA1_CH5_TCF
- LL_HPDMA1_TRIGGER_HPDMA1_CH6_TCF
- LL_HPDMA1_TRIGGER_HPDMA1_CH7_TCF
- LL_HPDMA1_TRIGGER_HPDMA1_CH8_TCF
- LL_HPDMA1_TRIGGER_HPDMA1_CH9_TCF
- LL_HPDMA1_TRIGGER_HPDMA1_CH10_TCF
- LL_HPDMA1_TRIGGER_HPDMA1_CH11_TCF
- LL_HPDMA1_TRIGGER_HPDMA1_CH12_TCF
- LL_HPDMA1_TRIGGER_HPDMA1_CH13_TCF
- LL_HPDMA1_TRIGGER_HPDMA1_CH14_TCF
- LL_HPDMA1_TRIGGER_HPDMA1_CH15_TCF
- LL_HPDMA1_TRIGGER_GPDMA1_CH0_TCF
- LL_HPDMA1_TRIGGER_GPDMA1_CH1_TCF
- LL_HPDMA1_TRIGGER_GPDMA1_CH2_TCF
- LL_HPDMA1_TRIGGER_GPDMA1_CH3_TCF
- LL_HPDMA1_TRIGGER_GPDMA1_CH4_TCF
- LL_HPDMA1_TRIGGER_GPDMA1_CH5_TCF
- LL_HPDMA1_TRIGGER_GPDMA1_CH6_TCF
- LL_HPDMA1_TRIGGER_GPDMA1_CH7_TCF
- LL_HPDMA1_TRIGGER_GPDMA1_CH8_TCF
- LL_HPDMA1_TRIGGER_GPDMA1_CH9_TCF
- LL_HPDMA1_TRIGGER_GPDMA1_CH10_TCF
- LL_HPDMA1_TRIGGER_GPDMA1_CH11_TCF
- LL_HPDMA1_TRIGGER_GPDMA1_CH12_TCF
- LL_HPDMA1_TRIGGER_GPDMA1_CH13_TCF
- LL_HPDMA1_TRIGGER_GPDMA1_CH14_TCF
- LL_HPDMA1_TRIGGER_GPDMA1_CH15_TCF
- LL_HPDMA1_TRIGGER_EXTIT0_SYNC
- LL_HPDMA1_TRIGGER_EXTIT1_SYNC
- LL_HPDMA1_TRIGGER_EXTIT2_SYNC
- LL_HPDMA1_TRIGGER_EXTIT3_SYNC
- LL_HPDMA1_TRIGGER_EXTIT4_SYNC
- LL_HPDMA1_TRIGGER_EXTIT5_SYNC
- LL_HPDMA1_TRIGGER_EXTIT6_SYNC
- LL_HPDMA1_TRIGGER_EXTIT7_SYNC
- LL_HPDMA1_TRIGGER_EXTIT8_SYNC
- LL_HPDMA1_TRIGGER_EXTIT9_SYNC
- LL_HPDMA1_TRIGGER_EXTIT10_SYNC
- LL_HPDMA1_TRIGGER_EXTIT11_SYNC
- LL_HPDMA1_TRIGGER_EXTIT12_SYNC
- LL_HPDMA1_TRIGGER_EXTIT13_SYNC

- LL_HPDMA1_TRIGGER_EXTIT14_SYNC
- LL_HPDMA1_TRIGGER_EXTIT15_SYNC
- LL_GPDMA1_TRIGGER_DCMIPP_P0_FEND
- LL_GPDMA1_TRIGGER_DCMIPP_P0_LEND
- LL_GPDMA1_TRIGGER_DCMIPP_P0_HSYNC
- LL_GPDMA1_TRIGGER_DCMIPP_P0_VSYNC
- LL_GPDMA1_TRIGGER_DCMIPP_P1_FEND
- LL_GPDMA1_TRIGGER_DCMIPP_P1_LEND
- LL_GPDMA1_TRIGGER_DCMIPP_P1_HSYNC
- LL_GPDMA1_TRIGGER_DCMIPP_P1_VSYNC
- LL_GPDMA1_TRIGGER_DCMIPP_P2_FEND
- LL_GPDMA1_TRIGGER_DCMIPP_P2_LEND
- LL_GPDMA1_TRIGGER_DCMIPP_P2_HSYNC
- LL_GPDMA1_TRIGGER_DCMIPP_P2_VSYNC
- LL_GPDMA1_TRIGGER_DMA2D_CTC
- LL_GPDMA1_TRIGGER_DMA2D_TC
- LL_GPDMA1_TRIGGER_DMA2D_TW
- LL_GPDMA1_TRIGGER_JPEG_EOC
- LL_GPDMA1_TRIGGER_JPEG_IFNF
- LL_GPDMA1_TRIGGER_JPEG_IFT
- LL_GPDMA1_TRIGGER_JPEG_OFNE
- LL_GPDMA1_TRIGGER_JPEG_OFT
- LL_GPDMA1_TRIGGER_LCD_LI
- LL_GPDMA1_TRIGGER_GPU2D1_GP_0
- LL_GPDMA1_TRIGGER_GPU2D1_GP_1
- LL_GPDMA1_TRIGGER_GPU2D1_GP_2
- LL_GPDMA1_TRIGGER_GPU2D1_GP_3
- LL_GPDMA1_TRIGGER_GFXTIM_3
- LL_GPDMA1_TRIGGER_GFXTIM_2
- LL_GPDMA1_TRIGGER_GFXTIM_1
- LL_GPDMA1_TRIGGER_GFXTIM_0
- LL_GPDMA1_TRIGGER_LPTIM1_CH1
- LL_GPDMA1_TRIGGER_LPTIM1_CH2
- LL_GPDMA1_TRIGGER_LPTIM2_CH1
- LL_GPDMA1_TRIGGER_LPTIM2_CH2
- LL_GPDMA1_TRIGGER_LPTIM3_CH1
- LL_GPDMA1_TRIGGER_LPTIM3_CH2
- LL_GPDMA1_TRIGGER_LPTIM4_OUT
- LL_GPDMA1_TRIGGER_LPTIM5_OUT
- LL_GPDMA1_TRIGGER_RTC_WKUP
- LL_GPDMA1_TRIGGER_LPUART1_IT_R_WUP_ASYNC
- LL_GPDMA1_TRIGGER_LPUART1_IT_T_WUP_ASYNC
- LL_GPDMA1_TRIGGER_SPI6_IT_OR_SPI6_AIT_SYNC
- LL_GPDMA1_TRIGGER_TIM1_TRGO
- LL_GPDMA1_TRIGGER_TIM1_TRGO2
- LL_GPDMA1_TRIGGER_TIM2_TRGO
- LL_GPDMA1_TRIGGER_TIM3_TRGO
- LL_GPDMA1_TRIGGER_TIM4_TRGO
- LL_GPDMA1_TRIGGER_TIM5_TRGO
- LL_GPDMA1_TRIGGER_TIM6_TRGO

- LL_GPDMA1_TRIGGER_TIM7_TRGO
- LL_GPDMA1_TRIGGER_TIM8_TRGO
- LL_GPDMA1_TRIGGER_TIM8_TRGO2
- LL_GPDMA1_TRIGGER_TIM12_TRGO
- LL_GPDMA1_TRIGGER_TIM15_TRGO
- LL_GPDMA1_TRIGGER_HPDMA1_CH0_TCF
- LL_GPDMA1_TRIGGER_HPDMA1_CH1_TCF
- LL_GPDMA1_TRIGGER_HPDMA1_CH2_TCF
- LL_GPDMA1_TRIGGER_HPDMA1_CH3_TCF
- LL_GPDMA1_TRIGGER_HPDMA1_CH4_TCF
- LL_GPDMA1_TRIGGER_HPDMA1_CH5_TCF
- LL_GPDMA1_TRIGGER_HPDMA1_CH6_TCF
- LL_GPDMA1_TRIGGER_HPDMA1_CH7_TCF
- LL_GPDMA1_TRIGGER_HPDMA1_CH8_TCF
- LL_GPDMA1_TRIGGER_HPDMA1_CH9_TCF
- LL_GPDMA1_TRIGGER_HPDMA1_CH10_TCF
- LL_GPDMA1_TRIGGER_HPDMA1_CH11_TCF
- LL_GPDMA1_TRIGGER_HPDMA1_CH12_TCF
- LL_GPDMA1_TRIGGER_HPDMA1_CH13_TCF
- LL_GPDMA1_TRIGGER_HPDMA1_CH14_TCF
- LL_GPDMA1_TRIGGER_HPDMA1_CH15_TCF
- LL_GPDMA1_TRIGGER_GPDMA1_CH0_TCF
- LL_GPDMA1_TRIGGER_GPDMA1_CH1_TCF
- LL_GPDMA1_TRIGGER_GPDMA1_CH2_TCF
- LL_GPDMA1_TRIGGER_GPDMA1_CH3_TCF
- LL_GPDMA1_TRIGGER_GPDMA1_CH4_TCF
- LL_GPDMA1_TRIGGER_GPDMA1_CH5_TCF
- LL_GPDMA1_TRIGGER_GPDMA1_CH6_TCF
- LL_GPDMA1_TRIGGER_GPDMA1_CH7_TCF
- LL_GPDMA1_TRIGGER_GPDMA1_CH8_TCF
- LL_GPDMA1_TRIGGER_GPDMA1_CH9_TCF
- LL_GPDMA1_TRIGGER_GPDMA1_CH10_TCF
- LL_GPDMA1_TRIGGER_GPDMA1_CH11_TCF
- LL_GPDMA1_TRIGGER_GPDMA1_CH12_TCF
- LL_GPDMA1_TRIGGER_GPDMA1_CH13_TCF
- LL_GPDMA1_TRIGGER_GPDMA1_CH14_TCF
- LL_GPDMA1_TRIGGER_GPDMA1_CH15_TCF
- LL_GPDMA1_TRIGGER_EXTIT0_SYNC
- LL_GPDMA1_TRIGGER_EXTIT1_SYNC
- LL_GPDMA1_TRIGGER_EXTIT2_SYNC
- LL_GPDMA1_TRIGGER_EXTIT3_SYNC
- LL_GPDMA1_TRIGGER_EXTIT4_SYNC
- LL_GPDMA1_TRIGGER_EXTIT5_SYNC
- LL_GPDMA1_TRIGGER_EXTIT6_SYNC
- LL_GPDMA1_TRIGGER_EXTIT7_SYNC
- LL_GPDMA1_TRIGGER_EXTIT8_SYNC
- LL_GPDMA1_TRIGGER_EXTIT9_SYNC
- LL_GPDMA1_TRIGGER_EXTIT10_SYNC
- LL_GPDMA1_TRIGGER_EXTIT11_SYNC
- LL_GPDMA1_TRIGGER_EXTIT12_SYNC

- LL_GPDMA1_TRIGGER_EXTIT13_SYNC
- LL_GPDMA1_TRIGGER_EXTIT14_SYNC
- LL_GPDMA1_TRIGGER_EXTIT15_SYNC

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR2 TRIGSEL LL_DMA_SetHWTrigger

LL_DMA_GetHWTrigger

Function name

__STATIC_INLINE uint32_t LL_DMA_GetHWTrigger (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get hardware triggers.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **Returned:** value can be one of the following values:
  - LL_HPDMA1_TRIGGER_DCMIPP_P0_FEND
  - LL_HPDMA1_TRIGGER_DCMIPP_P0_LEND
  - LL_HPDMA1_TRIGGER_DCMIPP_P0_HSYNC
  - LL_HPDMA1_TRIGGER_DCMIPP_P0_VSYNC
  - LL_HPDMA1_TRIGGER_DCMIPP_P1_FEND
  - LL_HPDMA1_TRIGGER_DCMIPP_P1_LEND
  - LL_HPDMA1_TRIGGER_DCMIPP_P1_HSYNC
  - LL_HPDMA1_TRIGGER_DCMIPP_P1_VSYNC
  - LL_HPDMA1_TRIGGER_DCMIPP_P2_FEND
  - LL_HPDMA1_TRIGGER_DCMIPP_P2_LEND
  - LL_HPDMA1_TRIGGER_DCMIPP_P2_HSYNC
  - LL_HPDMA1_TRIGGER_DCMIPP_P2_VSYNC
  - LL_HPDMA1_TRIGGER_DMA2D_CTC
  - LL_HPDMA1_TRIGGER_DMA2D_TC
  - LL_HPDMA1_TRIGGER_DMA2D_TW
  - LL_HPDMA1_TRIGGER_JPEG_EOC
  - LL_HPDMA1_TRIGGER_JPEG_IFNF
  - LL_HPDMA1_TRIGGER_JPEG_IFT
  - LL_HPDMA1_TRIGGER_JPEG_OFNE
  - LL_HPDMA1_TRIGGER_JPEG_OFT
  - LL_HPDMA1_TRIGGER_LCD_LI
  - LL_HPDMA1_TRIGGER_GPU2D1_GP_0
  - LL_HPDMA1_TRIGGER_GPU2D1_GP_1
  - LL_HPDMA1_TRIGGER_GPU2D1_GP_2
  - LL_HPDMA1_TRIGGER_GPU2D1_GP_3
  - LL_HPDMA1_TRIGGER_GFXTIM_3
  - LL_HPDMA1_TRIGGER_GFXTIM_2
  - LL_HPDMA1_TRIGGER_GFXTIM_1
  - LL_HPDMA1_TRIGGER_GFXTIM_0
  - LL_HPDMA1_TRIGGER_LPTIM1_CH1
  - LL_HPDMA1_TRIGGER_LPTIM1_CH2
  - LL_HPDMA1_TRIGGER_LPTIM2_CH1
  - LL_HPDMA1_TRIGGER_LPTIM2_CH2
  - LL_HPDMA1_TRIGGER_LPTIM3_CH1
  - LL_HPDMA1_TRIGGER_LPTIM3_CH2
  - LL_HPDMA1_TRIGGER_LPTIM4_OUT
  - LL_HPDMA1_TRIGGER_LPTIM5_OUT
  - LL_HPDMA1_TRIGGER_RTC_WKUP
  - LL_HPDMA1_TRIGGER_LPUART1_IT_R_WUP_ASYNC
  - LL_HPDMA1_TRIGGER_LPUART1_IT_T_WUP_ASYNC
  - LL_HPDMA1_TRIGGER_SPI6_IT_OR_SPI6_AIT_SYNC
  - LL_HPDMA1_TRIGGER_TIM1_TRGO
  - LL_HPDMA1_TRIGGER_TIM1_TRGO2
  - LL_HPDMA1_TRIGGER_TIM2_TRGO
  - LL_HPDMA1_TRIGGER_TIM3_TRGO
  - LL_HPDMA1_TRIGGER_TIM4_TRGO
  - LL_HPDMA1_TRIGGER_TIM5_TRGO
  - LL_HPDMA1_TRIGGER_TIM6_TRGO

- LL_HPDMA1_TRIGGER_TIM7_TRGO
- LL_HPDMA1_TRIGGER_TIM8_TRGO
- LL_HPDMA1_TRIGGER_TIM8_TRGO2
- LL_HPDMA1_TRIGGER_TIM12_TRGO
- LL_HPDMA1_TRIGGER_TIM15_TRGO
- LL_HPDMA1_TRIGGER_HPDMA1_CH0_TCF
- LL_HPDMA1_TRIGGER_HPDMA1_CH1_TCF
- LL_HPDMA1_TRIGGER_HPDMA1_CH2_TCF
- LL_HPDMA1_TRIGGER_HPDMA1_CH3_TCF
- LL_HPDMA1_TRIGGER_HPDMA1_CH4_TCF
- LL_HPDMA1_TRIGGER_HPDMA1_CH5_TCF
- LL_HPDMA1_TRIGGER_HPDMA1_CH6_TCF
- LL_HPDMA1_TRIGGER_HPDMA1_CH7_TCF
- LL_HPDMA1_TRIGGER_HPDMA1_CH8_TCF
- LL_HPDMA1_TRIGGER_HPDMA1_CH9_TCF
- LL_HPDMA1_TRIGGER_HPDMA1_CH10_TCF
- LL_HPDMA1_TRIGGER_HPDMA1_CH11_TCF
- LL_HPDMA1_TRIGGER_HPDMA1_CH12_TCF
- LL_HPDMA1_TRIGGER_HPDMA1_CH13_TCF
- LL_HPDMA1_TRIGGER_HPDMA1_CH14_TCF
- LL_HPDMA1_TRIGGER_HPDMA1_CH15_TCF
- LL_HPDMA1_TRIGGER_GPDMA1_CH0_TCF
- LL_HPDMA1_TRIGGER_GPDMA1_CH1_TCF
- LL_HPDMA1_TRIGGER_GPDMA1_CH2_TCF
- LL_HPDMA1_TRIGGER_GPDMA1_CH3_TCF
- LL_HPDMA1_TRIGGER_GPDMA1_CH4_TCF
- LL_HPDMA1_TRIGGER_GPDMA1_CH5_TCF
- LL_HPDMA1_TRIGGER_GPDMA1_CH6_TCF
- LL_HPDMA1_TRIGGER_GPDMA1_CH7_TCF
- LL_HPDMA1_TRIGGER_GPDMA1_CH8_TCF
- LL_HPDMA1_TRIGGER_GPDMA1_CH9_TCF
- LL_HPDMA1_TRIGGER_GPDMA1_CH10_TCF
- LL_HPDMA1_TRIGGER_GPDMA1_CH11_TCF
- LL_HPDMA1_TRIGGER_GPDMA1_CH12_TCF
- LL_HPDMA1_TRIGGER_GPDMA1_CH13_TCF
- LL_HPDMA1_TRIGGER_GPDMA1_CH14_TCF
- LL_HPDMA1_TRIGGER_GPDMA1_CH15_TCF
- LL_HPDMA1_TRIGGER_EXTIT0_SYNC
- LL_HPDMA1_TRIGGER_EXTIT1_SYNC
- LL_HPDMA1_TRIGGER_EXTIT2_SYNC
- LL_HPDMA1_TRIGGER_EXTIT3_SYNC
- LL_HPDMA1_TRIGGER_EXTIT4_SYNC
- LL_HPDMA1_TRIGGER_EXTIT5_SYNC
- LL_HPDMA1_TRIGGER_EXTIT6_SYNC
- LL_HPDMA1_TRIGGER_EXTIT7_SYNC
- LL_HPDMA1_TRIGGER_EXTIT8_SYNC
- LL_HPDMA1_TRIGGER_EXTIT9_SYNC
- LL_HPDMA1_TRIGGER_EXTIT10_SYNC
- LL_HPDMA1_TRIGGER_EXTIT11_SYNC
- LL_HPDMA1_TRIGGER_EXTIT12_SYNC

- LL_HPDMA1_TRIGGER_EXTIT13_SYNC
- LL_HPDMA1_TRIGGER_EXTIT14_SYNC
- LL_HPDMA1_TRIGGER_EXTIT15_SYNC
- LL_GPDMA1_TRIGGER_DCMIPP_P0_FEND
- LL_GPDMA1_TRIGGER_DCMIPP_P0_LEND
- LL_GPDMA1_TRIGGER_DCMIPP_P0_HSYNC
- LL_GPDMA1_TRIGGER_DCMIPP_P0_VSYNC
- LL_GPDMA1_TRIGGER_DCMIPP_P1_FEND
- LL_GPDMA1_TRIGGER_DCMIPP_P1_LEND
- LL_GPDMA1_TRIGGER_DCMIPP_P1_HSYNC
- LL_GPDMA1_TRIGGER_DCMIPP_P1_VSYNC
- LL_GPDMA1_TRIGGER_DCMIPP_P2_FEND
- LL_GPDMA1_TRIGGER_DCMIPP_P2_LEND
- LL_GPDMA1_TRIGGER_DCMIPP_P2_HSYNC
- LL_GPDMA1_TRIGGER_DCMIPP_P2_VSYNC
- LL_GPDMA1_TRIGGER_DMA2D_CTC
- LL_GPDMA1_TRIGGER_DMA2D_TC
- LL_GPDMA1_TRIGGER_DMA2D_TW
- LL_GPDMA1_TRIGGER_JPEG_EOC
- LL_GPDMA1_TRIGGER_JPEG_IFNF
- LL_GPDMA1_TRIGGER_JPEG_IFT
- LL_GPDMA1_TRIGGER_JPEG_OFNE
- LL_GPDMA1_TRIGGER_JPEG_OFT
- LL_GPDMA1_TRIGGER_LCD_LI
- LL_GPDMA1_TRIGGER_GPU2D1_GP_0
- LL_GPDMA1_TRIGGER_GPU2D1_GP_1
- LL_GPDMA1_TRIGGER_GPU2D1_GP_2
- LL_GPDMA1_TRIGGER_GPU2D1_GP_3
- LL_GPDMA1_TRIGGER_GFXTIM_3
- LL_GPDMA1_TRIGGER_GFXTIM_2
- LL_GPDMA1_TRIGGER_GFXTIM_1
- LL_GPDMA1_TRIGGER_GFXTIM_0
- LL_GPDMA1_TRIGGER_LPTIM1_CH1
- LL_GPDMA1_TRIGGER_LPTIM1_CH2
- LL_GPDMA1_TRIGGER_LPTIM2_CH1
- LL_GPDMA1_TRIGGER_LPTIM2_CH2
- LL_GPDMA1_TRIGGER_LPTIM3_CH1
- LL_GPDMA1_TRIGGER_LPTIM3_CH2
- LL_GPDMA1_TRIGGER_LPTIM4_OUT
- LL_GPDMA1_TRIGGER_LPTIM5_OUT
- LL_GPDMA1_TRIGGER_RTC_WKUP
- LL_GPDMA1_TRIGGER_LPUART1_IT_R_WUP_ASYNC
- LL_GPDMA1_TRIGGER_LPUART1_IT_T_WUP_ASYNC
- LL_GPDMA1_TRIGGER_SPI6_IT_OR_SPI6_AIT_SYNC
- LL_GPDMA1_TRIGGER_TIM1_TRGO
- LL_GPDMA1_TRIGGER_TIM1_TRGO2
- LL_GPDMA1_TRIGGER_TIM2_TRGO
- LL_GPDMA1_TRIGGER_TIM3_TRGO
- LL_GPDMA1_TRIGGER_TIM4_TRGO
- LL_GPDMA1_TRIGGER_TIM5_TRGO

- LL_GPDMA1_TRIGGER_TIM6_TRGO
- LL_GPDMA1_TRIGGER_TIM7_TRGO
- LL_GPDMA1_TRIGGER_TIM8_TRGO
- LL_GPDMA1_TRIGGER_TIM8_TRGO2
- LL_GPDMA1_TRIGGER_TIM12_TRGO
- LL_GPDMA1_TRIGGER_TIM15_TRGO
- LL_GPDMA1_TRIGGER_HPDMA1_CH0_TCF
- LL_GPDMA1_TRIGGER_HPDMA1_CH1_TCF
- LL_GPDMA1_TRIGGER_HPDMA1_CH2_TCF
- LL_GPDMA1_TRIGGER_HPDMA1_CH3_TCF
- LL_GPDMA1_TRIGGER_HPDMA1_CH4_TCF
- LL_GPDMA1_TRIGGER_HPDMA1_CH5_TCF
- LL_GPDMA1_TRIGGER_HPDMA1_CH6_TCF
- LL_GPDMA1_TRIGGER_HPDMA1_CH7_TCF
- LL_GPDMA1_TRIGGER_HPDMA1_CH8_TCF
- LL_GPDMA1_TRIGGER_HPDMA1_CH9_TCF
- LL_GPDMA1_TRIGGER_HPDMA1_CH10_TCF
- LL_GPDMA1_TRIGGER_HPDMA1_CH11_TCF
- LL_GPDMA1_TRIGGER_HPDMA1_CH12_TCF
- LL_GPDMA1_TRIGGER_HPDMA1_CH13_TCF
- LL_GPDMA1_TRIGGER_HPDMA1_CH14_TCF
- LL_GPDMA1_TRIGGER_HPDMA1_CH15_TCF
- LL_GPDMA1_TRIGGER_GPDMA1_CH0_TCF
- LL_GPDMA1_TRIGGER_GPDMA1_CH1_TCF
- LL_GPDMA1_TRIGGER_GPDMA1_CH2_TCF
- LL_GPDMA1_TRIGGER_GPDMA1_CH3_TCF
- LL_GPDMA1_TRIGGER_GPDMA1_CH4_TCF
- LL_GPDMA1_TRIGGER_GPDMA1_CH5_TCF
- LL_GPDMA1_TRIGGER_GPDMA1_CH6_TCF
- LL_GPDMA1_TRIGGER_GPDMA1_CH7_TCF
- LL_GPDMA1_TRIGGER_GPDMA1_CH8_TCF
- LL_GPDMA1_TRIGGER_GPDMA1_CH9_TCF
- LL_GPDMA1_TRIGGER_GPDMA1_CH10_TCF
- LL_GPDMA1_TRIGGER_GPDMA1_CH11_TCF
- LL_GPDMA1_TRIGGER_GPDMA1_CH12_TCF
- LL_GPDMA1_TRIGGER_GPDMA1_CH13_TCF
- LL_GPDMA1_TRIGGER_GPDMA1_CH14_TCF
- LL_GPDMA1_TRIGGER_GPDMA1_CH15_TCF
- LL_GPDMA1_TRIGGER_EXTIT0_SYNC
- LL_GPDMA1_TRIGGER_EXTIT1_SYNC
- LL_GPDMA1_TRIGGER_EXTIT2_SYNC
- LL_GPDMA1_TRIGGER_EXTIT3_SYNC
- LL_GPDMA1_TRIGGER_EXTIT4_SYNC
- LL_GPDMA1_TRIGGER_EXTIT5_SYNC
- LL_GPDMA1_TRIGGER_EXTIT6_SYNC
- LL_GPDMA1_TRIGGER_EXTIT7_SYNC
- LL_GPDMA1_TRIGGER_EXTIT8_SYNC
- LL_GPDMA1_TRIGGER_EXTIT9_SYNC
- LL_GPDMA1_TRIGGER_EXTIT10_SYNC
- LL_GPDMA1_TRIGGER_EXTIT11_SYNC

- LL_GPDMA1_TRIGGER_EXTIT12_SYNC
- LL_GPDMA1_TRIGGER_EXTIT13_SYNC
- LL_GPDMA1_TRIGGER_EXTIT14_SYNC
- LL_GPDMA1_TRIGGER_EXTIT15_SYNC

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR2 TRIGSEL LL_DMA_GetHWTrigger

LL_DMA_SetTransferMode

Function name

__STATIC_INLINE void LL_DMA_SetTransferMode (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t Mode)

Function description

Set DMA transfer mode.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_15
- **Mode:** This parameter can be one of the following values:
  - LL_DMA_NORMAL
  - LL_DMA_PFCTRL

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR2 PFREQ LL_DMA_SetTransferMode

LL_DMA_GetTransferMode

Function name

__STATIC_INLINE uint32_t LL_DMA_GetTransferMode (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get DMA transfer mode.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_15

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_NORMAL
  - LL_DMA_PFCTRL

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR2 TRIGSEL LL_DMA_GetTransferMode

LL_DMA_ConfigBlkRptAddrUpdate

Function name

__STATIC_INLINE void LL_DMA_ConfigBlkRptAddrUpdate (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t Configuration)

Function description

Configure addresses update.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **Configuration:** This parameter must be a combination of all the following values:
  - LL_DMA_BLKRPT_DEST_ADDR_INCREMENT or LL_DMA_BLKRPT_DEST_ADDR_DECREMENT
  - LL_DMA_BLKRPT_SRC_ADDR_INCREMENT or LL_DMA_BLKRPT_SRC_ADDR_DECREMENT
  - LL_DMA_BURST_DEST_ADDR_INCREMENT or LL_DMA_BURST_DEST_ADDR_DECREMENT
  - LL_DMA_BURST_SRC_ADDR_INCREMENT or LL_DMA_BURST_SRC_ADDR_DECREMENT

Return values

- **None.:**

Notes

- This API is used only for 2D addressing channels.

Reference Manual to LL API cross reference:

- CBR1 BRDDEC LL_DMA_ConfigBlkRptAddrUpdate
- CBR1 BRSDEC LL_DMA_ConfigBlkRptAddrUpdate
- CBR1 DDEC LL_DMA_ConfigBlkRptAddrUpdate
- CBR1 SDEC LL_DMA_ConfigBlkRptAddrUpdate

LL_DMA_ConfigBlkCounters

Function name

__STATIC_INLINE void LL_DMA_ConfigBlkCounters (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t BlkDataLength, uint32_t BlkRptCount)

Function description

Configure DMA Block number of data and repeat Count.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **BlkDataLength:** Block transfer length Value between 0 to 0x0000FFFF
- **BlkRptCount:** Block repeat counter Value between 0 to 0x000007FF

Return values

- **None.:**

Notes

- This API is used only for 2D addressing channels.

Reference Manual to LL API cross reference:

- CBR1 BNDT LL_DMA_ConfigBlkCounters
- CBR1 BRC LL_DMA_ConfigBlkCounters

LL_DMA_SetBlkRptDestAddrUpdate

Function name

__STATIC_INLINE void LL_DMA_SetBlkRptDestAddrUpdate (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t BlkRptDestAddrUpdate)

Function description

Set block repeat destination address update.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **BlkRptDestAddrUpdate:** This parameter can be one of the following values:
  - LL_DMA_BLKRPT_DEST_ADDR_INCREMENT
  - LL_DMA_BLKRPT_DEST_ADDR_DECREMENT

Return values

- **None.:**

Notes

- This API is used only for 2D addressing channels.

Reference Manual to LL API cross reference:

- CBR1 BRDDEC LL_DMA_SetBlkRptDestAddrUpdate

LL_DMA_GetBlkRptDestAddrUpdate

Function name

__STATIC_INLINE uint32_t LL_DMA_GetBlkRptDestAddrUpdate (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get block repeat destination address update.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_BLKRPT_DEST_ADDR_INCREMENT
  - LL_DMA_BLKRPT_DEST_ADDR_DECREMENT

Notes

- This API is used only for 2D addressing channels.

Reference Manual to LL API cross reference:

- CBR1 BRDDEC LL_DMA_GetBlkRptDestAddrUpdate

LL_DMA_SetBlkRptSrcAddrUpdate

Function name

__STATIC_INLINE void LL_DMA_SetBlkRptSrcAddrUpdate (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t BlkRptSrcAddrUpdate)

Function description

Set block repeat source address update.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **BlkRptSrcAddrUpdate:** This parameter can be one of the following values:
  - LL_DMA_BLKRPT_SRC_ADDR_INCREMENT
  - LL_DMA_BLKRPT_SRC_ADDR_DECREMENT

Return values

- **None.:**

Notes

- This API is used only for 2D addressing channels.

Reference Manual to LL API cross reference:

- CBR1 BRSDEC LL_DMA_SetBlkRptSrcAddrUpdate

LL_DMA_GetBlkRptSrcAddrUpdate

Function name

__STATIC_INLINE uint32_t LL_DMA_GetBlkRptSrcAddrUpdate (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get block repeat source address update.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_BLKRPT_SRC_ADDR_INCREMENT
  - LL_DMA_BLKRPT_SRC_ADDR_DECREMENT

Notes

- This API is used only for 2D addressing channels.

Reference Manual to LL API cross reference:

- CBR1 BRSDEC LL_DMA_GetBlkRptSrcAddrUpdate

LL_DMA_SetDestAddrUpdate

Function name

__STATIC_INLINE void LL_DMA_SetDestAddrUpdate (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t DestAddrUpdate)

Function description

Set destination address update.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **DestAddrUpdate:** This parameter can be one of the following values:
  - LL_DMA_BURST_DEST_ADDR_INCREMENT
  - LL_DMA_BURST_DEST_ADDR_DECREMENT

Return values

- **None.:**

Notes

- This API is used only for 2D addressing channels.

Reference Manual to LL API cross reference:

- CBR1 DDEC LL_DMA_SetDestAddrUpdate

LL_DMA_GetDestAddrUpdate

Function name

__STATIC_INLINE uint32_t LL_DMA_GetDestAddrUpdate (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get destination address update.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_BURST_DEST_ADDR_INCREMENT
  - LL_DMA_BURST_DEST_ADDR_DECREMENT

Notes

- This API is used only for 2D addressing channels.

Reference Manual to LL API cross reference:

- CBR1 DDEC LL_DMA_GetDestAddrUpdate

LL_DMA_SetSrcAddrUpdate

Function name

__STATIC_INLINE void LL_DMA_SetSrcAddrUpdate (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t SrcAddrUpdate)

Function description

Set source address update.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **SrcAddrUpdate:** This parameter can be one of the following values:
  - LL_DMA_BURST_SRC_ADDR_INCREMENT
  - LL_DMA_BURST_SRC_ADDR_DECREMENT

Return values

- **None.:**

Notes

- This API is used only for 2D addressing channels.

Reference Manual to LL API cross reference:

- CBR1 SDEC LL_DMA_SetSrcAddrUpdate

LL_DMA_GetSrcAddrUpdate

Function name

__STATIC_INLINE uint32_t LL_DMA_GetSrcAddrUpdate (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get source address update.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_BURST_SRC_ADDR_INCREMENT
  - LL_DMA_BURST_SRC_ADDR_DECREMENT

Notes

- This API is used only for 2D addressing channels.

Reference Manual to LL API cross reference:

- CBR1 SDEC LL_DMA_GetSrcAddrUpdate

LL_DMA_SetBlkRptCount

Function name

__STATIC_INLINE void LL_DMA_SetBlkRptCount (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t BlkRptCount)

Function description

Set block repeat count.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **BlkRptCount:** Block repeat counter Value between 0 to 0x000007FF

Return values

- **None.:**

Notes

- This API is used only for 2D addressing channels.

Reference Manual to LL API cross reference:

- CBR1 BRC LL_DMA_SetBlkRptCount

LL_DMA_GetBlkRptCount

Function name

__STATIC_INLINE uint32_t LL_DMA_GetBlkRptCount (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get block repeat count.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **Between:** 0 to 0x000007FF

Notes

- This API is used only for 2D addressing channels.

Reference Manual to LL API cross reference:

- CBR1 BRC LL_DMA_GetBlkRptCount

LL_DMA_SetBlkDataLength

Function name

__STATIC_INLINE void LL_DMA_SetBlkDataLength (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t BlkDataLength)

Function description

Set block data length in bytes to transfer.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **BlkDataLength:** Between 0 to 0x0000FFFF

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CBR1 BNDT LL_DMA_SetBlkDataLength

LL_DMA_GetBlkDataLength

Function name

__STATIC_INLINE uint32_t LL_DMA_GetBlkDataLength (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get block data length in bytes to transfer.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **Between:** 0 to 0x0000FFFF

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CBR1 BNDT LL_DMA_GetBlkDataLength

LL_DMA_ConfigAddresses

Function name

__STATIC_INLINE void LL_DMA_ConfigAddresses (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t SrcAddress, uint32_t DestAddress)

Function description

Configure the source and destination addresses.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **SrcAddress:** Between 0 to 0xFFFFFFFF
- **DestAddress:** Between 0 to 0xFFFFFFFF

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.
- This API must not be called when the DMA Channel is enabled.

Reference Manual to LL API cross reference:

- CSAR SA LL_DMA_ConfigAddresses
- CDAR DA LL_DMA_ConfigAddresses

LL_DMA_SetSrcAddress

Function name

__STATIC_INLINE void LL_DMA_SetSrcAddress (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t SrcAddress)

Function description

Set source address.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **SrcAddress:** Between 0 to 0xFFFFFFFF

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CSAR SA LL_DMA_SetSrcAddress

LL_DMA_GetSrcAddress

Function name

__STATIC_INLINE uint32_t LL_DMA_GetSrcAddress (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get source address.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **Between:** 0 to 0xFFFFFFFF

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CSAR SA LL_DMA_GetSrcAddress

LL_DMA_SetDestAddress

Function name

__STATIC_INLINE void LL_DMA_SetDestAddress (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t DestAddress)

Function description

Set destination address.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **DestAddress:** Between 0 to 0xFFFFFFFF

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CDAR DA LL_DMA_SetDestAddress

LL_DMA_GetDestAddress

Function name

__STATIC_INLINE uint32_t LL_DMA_GetDestAddress (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get destination address.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **Between:** 0 to 0xFFFFFFFF

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CDAR DA LL_DMA_GetDestAddress

LL_DMA_ConfigAddrUpdateValue

Function name

__STATIC_INLINE void LL_DMA_ConfigAddrUpdateValue (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t SrcAddrOffset, uint32_t DestAddrOffset)

Function description

Configure source and destination addresses offset.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **DestAddrOffset:** Between 0 to 0x00001FFF
- **SrcAddrOffset:** Between 0 to 0x00001FFF

Return values

- **None.:**

Notes

- This API is used only for 2D addressing channels.
- This API must not be called when the DMA Channel is enabled.

Reference Manual to LL API cross reference:

- CTR3 DAO LL_DMA_ConfigAddrUpdateValue
- CTR3 SAO LL_DMA_ConfigAddrUpdateValue

LL_DMA_SetDestAddrUpdateValue

Function name

__STATIC_INLINE void LL_DMA_SetDestAddrUpdateValue (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t DestAddrOffset)

Function description

Set destination address offset.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **DestAddrOffset:** Between 0 to 0x00001FFF

Return values

- **None.:**

Notes

- This API is used only for 2D addressing channels.

Reference Manual to LL API cross reference:

- CTR3 DAO LL_DMA_SetDestAddrUpdateValue

LL_DMA_GetDestAddrUpdateValue

Function name

__STATIC_INLINE uint32_t LL_DMA_GetDestAddrUpdateValue (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get destination address offset.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **Between:** 0 to 0x00001FFF

Notes

- This API is used only for 2D addressing channels.

Reference Manual to LL API cross reference:

- CDAR DAO LL_DMA_GetDestAddrUpdateValue

LL_DMA_SetSrcAddrUpdateValue

Function name

__STATIC_INLINE void LL_DMA_SetSrcAddrUpdateValue (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t SrcAddrOffset)

Function description

Set source address offset.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **SrcAddrOffset:** Between 0 to 0x00001FFF

Return values

- **None.:**

Notes

- This API is used only for 2D addressing channels.

Reference Manual to LL API cross reference:

- CTR3 SAO LL_DMA_SetSrcAddrUpdateValue

LL_DMA_GetSrcAddrUpdateValue

Function name

__STATIC_INLINE uint32_t LL_DMA_GetSrcAddrUpdateValue (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get source address offset.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **Between:** 0 to 0x00001FFF

Notes

- This API is used only for 2D addressing channels.

Reference Manual to LL API cross reference:

- CTR3 SAO LL_DMA_GetSrcAddrUpdateValue

LL_DMA_ConfigBlkRptAddrUpdateValue

Function name

__STATIC_INLINE void LL_DMA_ConfigBlkRptAddrUpdateValue (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t BlkRptSrcAddrOffset, uint32_t BlkRptDestAddrOffset)

Function description

Configure the block repeated source and destination addresses offset.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **BlkRptDestAddrOffset:** Between 0 to 0x0000FFFF
- **BlkRptSrcAddrOffset:** Between 0 to 0x0000FFFF

Return values

- **None.:**

Notes

- This API is used only for 2D addressing channels.
- This API must not be called when the DMA Channel is enabled.

Reference Manual to LL API cross reference:

- CBR2 BRDAO LL_DMA_ConfigBlkRptAddrUpdateValue
- CBR2 BRSAO LL_DMA_ConfigBlkRptAddrUpdateValue

LL_DMA_SetBlkRptDestAddrUpdateValue

Function name

__STATIC_INLINE void LL_DMA_SetBlkRptDestAddrUpdateValue (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t BlkRptDestAddrOffset)

Function description

Set block repeated destination address offset.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **BlkRptDestAddrOffset:** Between 0 to 0x0000FFFF

Return values

- **None.:**

Notes

- This API is used only for 2D addressing channels.

Reference Manual to LL API cross reference:

- CBR2 BRDAO LL_DMA_SetBlkRptDestAddrUpdateValue

LL_DMA_GetBlkRptDestAddrUpdateValue

Function name

__STATIC_INLINE uint32_t LL_DMA_GetBlkRptDestAddrUpdateValue (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get block repeated destination address offset.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **Between:** 0 to 0x0000FFFF.

Notes

- This API is used only for 2D addressing channels.

Reference Manual to LL API cross reference:

- CBR2 BRDAO LL_DMA_GetBlkRptDestAddrUpdateValue

LL_DMA_SetBlkRptSrcAddrUpdateValue

Function name

__STATIC_INLINE void LL_DMA_SetBlkRptSrcAddrUpdateValue (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t BlkRptSrcAddrOffset)

Function description

Set block repeated source address offset.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **BlkRptSrcAddrOffset:** Between 0 to 0x0000FFFF

Return values

- **None.:**

Notes

- This API is used only for 2D addressing channels.

Reference Manual to LL API cross reference:

- CBR2 BRSAO LL_DMA_SetBlkRptSrcAddrUpdateValue

LL_DMA_GetBlkRptSrcAddrUpdateValue

Function name

__STATIC_INLINE uint32_t LL_DMA_GetBlkRptSrcAddrUpdateValue (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get block repeated source address offset.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **Between:** 0 to 0x0000FFFF

Notes

- This API is used only for 2D addressing channels.

Reference Manual to LL API cross reference:

- CBR2 BRSAO LL_DMA_GetBlkRptSrcAddrUpdateValue

LL_DMA_ConfigLinkUpdate

Function name

__STATIC_INLINE void LL_DMA_ConfigLinkUpdate (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t RegistersUpdate, uint32_t LinkedListAddrOffset)

Function description

Configure registers update and node address offset during the link transfer.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **RegistersUpdate:** This parameter must be a combination of all the following values:
  - LL_DMA_UPDATE_CTR1
  - LL_DMA_UPDATE_CTR2
  - LL_DMA_UPDATE_CBR1
  - LL_DMA_UPDATE_CSAR
  - LL_DMA_UPDATE_CDAR
  - LL_DMA_UPDATE_CTR3 (This value is allowed only for 2D addressing channels)
  - LL_DMA_UPDATE_CBR2 (This value is allowed only for 2D addressing channels)
  - LL_DMA_UPDATE_CLLR
- **LinkedListAddrOffset:** Between 0 to 0x0000FFFC by increment of 4 Bytes.

Return values

- **None.:**

Notes

- This API is used for all available DMA channels. For linear addressing channels, UT3 and UB2 fields are discarded.

Reference Manual to LL API cross reference:

- CLLR UT1 LL_DMA_ConfigLinkUpdate
- CLLR UT2 LL_DMA_ConfigLinkUpdate
- CLLR UB1 LL_DMA_ConfigLinkUpdate
- CLLR USA LL_DMA_ConfigLinkUpdate
- CLLR UDA LL_DMA_ConfigLinkUpdate
- CLLR UT3 LL_DMA_ConfigLinkUpdate
- CLLR UB2 LL_DMA_ConfigLinkUpdate
- CLLR ULL LL_DMA_ConfigLinkUpdate

LL_DMA_EnableCTR1Update

Function name

__STATIC_INLINE void LL_DMA_EnableCTR1Update (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Enable CTR1 update during the link transfer.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CLLR UT1 LL_DMA_EnableCTR1Update

LL_DMA_DisableCTR1Update

Function name

__STATIC_INLINE void LL_DMA_DisableCTR1Update (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Disable CTR1 update during the link transfer.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CLLR UT1 LL_DMA_DisableCTR1Update

LL_DMA_IsEnabledCTR1Update

Function name

__STATIC_INLINE uint32_t LL_DMA_IsEnabledCTR1Update (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Check if CTR1 update during the link transfer is enabled.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CLLR UT1 LL_DMA_IsEnabledCTR1Update

LL_DMA_EnableCTR2Update

Function name

__STATIC_INLINE void LL_DMA_EnableCTR2Update (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Enable CTR2 update during the link transfer.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CLLR UT2 LL_DMA_EnableCTR2Update

LL_DMA_DisableCTR2Update

Function name

__STATIC_INLINE void LL_DMA_DisableCTR2Update (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Disable CTR2 update during the link transfer.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CLLR UT2 LL_DMA_DisableCTR2Update

LL_DMA_IsEnabledCTR2Update

Function name

__STATIC_INLINE uint32_t LL_DMA_IsEnabledCTR2Update (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Check if CTR2 update during the link transfer is enabled.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CLLR UT2 LL_DMA_IsEnabledCTR2Update

LL_DMA_EnableCBR1Update

Function name

__STATIC_INLINE void LL_DMA_EnableCBR1Update (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Enable CBR1 update during the link transfer.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CLLR UB1 LL_DMA_EnableCBR1Update

LL_DMA_DisableCBR1Update

Function name

__STATIC_INLINE void LL_DMA_DisableCBR1Update (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Disable CBR1 update during the link transfer.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CLLR UB1 LL_DMA_DisableCBR1Update

LL_DMA_IsEnabledCBR1Update

Function name

__STATIC_INLINE uint32_t LL_DMA_IsEnabledCBR1Update (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Check if CBR1 update during the link transfer is enabled.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CLLR UB1 LL_DMA_IsEnabledCBR1Update

LL_DMA_EnableCSARUpdate

Function name

__STATIC_INLINE void LL_DMA_EnableCSARUpdate (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Enable CSAR update during the link transfer.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CLLR USA LL_DMA_EnableCSARUpdate

LL_DMA_DisableCSARUpdate

Function name

__STATIC_INLINE void LL_DMA_DisableCSARUpdate (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Disable CSAR update during the link transfer.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CLLR USA LL_DMA_DisableCSARUpdate

LL_DMA_IsEnabledCSARUpdate

Function name

__STATIC_INLINE uint32_t LL_DMA_IsEnabledCSARUpdate (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Check if CSAR update during the link transfer is enabled.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CLLR USA LL_DMA_IsEnabledCSARUpdate

LL_DMA_EnableCDARUpdate

Function name

__STATIC_INLINE void LL_DMA_EnableCDARUpdate (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Enable CDAR update during the link transfer.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CLLR UDA LL_DMA_EnableCDARUpdate

LL_DMA_DisableCDARUpdate

Function name

__STATIC_INLINE void LL_DMA_DisableCDARUpdate (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Disable CDAR update during the link transfer.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CLLR UDA LL_DMA_DisableCDARUpdate

LL_DMA_IsEnabledCDARUpdate

Function name

__STATIC_INLINE uint32_t LL_DMA_IsEnabledCDARUpdate (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Check if CDAR update during the link transfer is enabled.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CLLR UDA LL_DMA_IsEnabledCDARUpdate

LL_DMA_EnableCTR3Update

Function name

__STATIC_INLINE void LL_DMA_EnableCTR3Update (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Enable CTR3 update during the link transfer.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used only for 2D addressing channels.

Reference Manual to LL API cross reference:

- CLLR UT3 LL_DMA_EnableCTR3Update

LL_DMA_DisableCTR3Update

Function name

__STATIC_INLINE void LL_DMA_DisableCTR3Update (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Disable CTR3 update during the link transfer.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used only for 2D addressing channels.

Reference Manual to LL API cross reference:

- CLLR UT3 LL_DMA_DisableCTR3Update

LL_DMA_IsEnabledCTR3Update

Function name

__STATIC_INLINE uint32_t LL_DMA_IsEnabledCTR3Update (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Check if CTR3 update during the link transfer is enabled.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used only for 2D addressing channels.

Reference Manual to LL API cross reference:

- CLLR UT3 LL_DMA_IsEnabledCTR3Update

LL_DMA_EnableCBR2Update

Function name

__STATIC_INLINE void LL_DMA_EnableCBR2Update (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Enable CBR2 update during the link transfer.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used only for 2D addressing channels.

Reference Manual to LL API cross reference:

- CLLR UB2 LL_DMA_EnableCBR2Update

LL_DMA_DisableCBR2Update

Function name

__STATIC_INLINE void LL_DMA_DisableCBR2Update (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Disable CBR2 update during the link transfer.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used only for 2D addressing channels.

Reference Manual to LL API cross reference:

- CLLR UB2 LL_DMA_DisableCBR2Update

LL_DMA_IsEnabledCBR2Update

Function name

__STATIC_INLINE uint32_t LL_DMA_IsEnabledCBR2Update (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Check if CBR2 update during the link transfer is enabled.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used only for 2D addressing channels.

Reference Manual to LL API cross reference:

- CLLR UB2 LL_DMA_IsEnabledCBR2Update

LL_DMA_EnableCLLRUpdate

Function name

__STATIC_INLINE void LL_DMA_EnableCLLRUpdate (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Enable CLLR update during the link transfer.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CLLR ULL LL_DMA_EnableCLLRUpdate

LL_DMA_DisableCLLRUpdate

Function name

__STATIC_INLINE void LL_DMA_DisableCLLRUpdate (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Disable CLLR update during the link transfer.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CLLR ULL LL_DMA_DisableCLLRUpdate

LL_DMA_IsEnabledCLLRUpdate

Function name

__STATIC_INLINE uint32_t LL_DMA_IsEnabledCLLRUpdate (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Check if CLLR update during the link transfer is enabled.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CLLR ULL LL_DMA_IsEnabledCLLRUpdate

LL_DMA_SetLinkedListAddrOffset

Function name

__STATIC_INLINE void LL_DMA_SetLinkedListAddrOffset (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t LinkedListAddrOffset)

Function description

Set linked list address offset.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **LinkedListAddrOffset:** Between 0 to 0x0000FFFC by increment of 4 Bytes.

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CLLR LA LL_DMA_SetLinkedListAddrOffset

LL_DMA_GetLinkedListAddrOffset

Function name

__STATIC_INLINE uint32_t LL_DMA_GetLinkedListAddrOffset (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get linked list address offset.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **Between:** 0 to 0x0000FFFC.

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CLLR LA LL_DMA_GetLinkedListAddrOffset

LL_DMA_GetFIFOLevel

Function name

__STATIC_INLINE uint32_t LL_DMA_GetFIFOLevel (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get FIFO level.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **Between:** 0 to 0x000000FF.

Reference Manual to LL API cross reference:

- CSR FIFOL LL_DMA_GetFIFOLevel

LL_DMA_EnableChannelSecure

Function name

__STATIC_INLINE void LL_DMA_EnableChannelSecure (DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Enable the DMA channel secure attribute.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- SECCFGR SECx LL_DMA_EnableChannelSecure

LL_DMA_DisableChannelSecure

Function name

__STATIC_INLINE void LL_DMA_DisableChannelSecure (DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Disable the DMA channel secure attribute.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- SECCFGR SECx LL_DMA_DisableChannelSecure

LL_DMA_IsEnabledChannelSecure

Function name

__STATIC_INLINE uint32_t LL_DMA_IsEnabledChannelSecure (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Check if DMA channel secure is enabled.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- SECCFGR SECx LL_DMA_IsEnabledChannelSecure

LL_DMA_EnableChannelPrivilege

Function name

__STATIC_INLINE void LL_DMA_EnableChannelPrivilege (DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Enable the DMA channel privilege attribute.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- PRIVCFGR PRIVx LL_DMA_EnableChannelPrivilege

LL_DMA_DisableChannelPrivilege

Function name

__STATIC_INLINE void LL_DMA_DisableChannelPrivilege (DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Disable the DMA channel privilege attribute.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- PRIVCFGR PRIVx LL_DMA_DisableChannelPrivilege

LL_DMA_IsEnabledChannelPrivilege

Function name

__STATIC_INLINE uint32_t LL_DMA_IsEnabledChannelPrivilege (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Check if DMA Channel privilege is enabled.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- PRIVCFGR PRIVx LL_DMA_IsEnabledChannelPrivilege

LL_DMA_EnableChannelLockAttribute

Function name

__STATIC_INLINE void LL_DMA_EnableChannelLockAttribute (DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Enable the DMA channel lock attributes.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- RCFGLOCKR LOCKx LL_DMA_EnableChannelLockAttribute

LL_DMA_IsEnabledChannelLockAttribute

Function name

__STATIC_INLINE uint32_t LL_DMA_IsEnabledChannelLockAttribute (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Check if DMA channel attributes are locked.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- SECCFGR LOCKx LL_DMA_IsEnabledChannelLockAttribute

LL_DMA_ClearFlag_TO

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_TO (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Clear trigger overrun flag.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CFCR TOF LL_DMA_ClearFlag_TO

LL_DMA_ClearFlag_SUSP

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_SUSP (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Clear suspension flag.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CFCR SUSPF LL_DMA_ClearFlag_SUSP

LL_DMA_ClearFlag_USE

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_USE (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Clear user setting error flag.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CFCR USEF LL_DMA_ClearFlag_USE

LL_DMA_ClearFlag_ULE

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_ULE (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Clear link transfer error flag.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CFCR ULEF LL_DMA_ClearFlag_ULE

LL_DMA_ClearFlag_DTE

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_DTE (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Clear data transfer error flag.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CFCR DTEF LL_DMA_ClearFlag_DTE

LL_DMA_ClearFlag_HT

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_HT (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Clear half transfer flag.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CFCR HTF LL_DMA_ClearFlag_HT

LL_DMA_ClearFlag_TC

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_TC (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Clear transfer complete flag.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CFCR TCF LL_DMA_ClearFlag_TC

LL_DMA_IsActiveFlag_TO

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TO (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get trigger overrun flag.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CSR TOF LL_DMA_IsActiveFlag_TO

LL_DMA_IsActiveFlag_SUSP

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_SUSP (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get suspension flag.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CSR SUSPF LL_DMA_IsActiveFlag_SUSP

LL_DMA_IsActiveFlag_USE

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_USE (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get user setting error flag.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CSR USEF LL_DMA_IsActiveFlag_USE

LL_DMA_IsActiveFlag_ULE

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_ULE (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get user setting error flag.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CSR ULEF LL_DMA_IsActiveFlag_ULE

LL_DMA_IsActiveFlag_DTE

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_DTE (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get data transfer error flag.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CSR DTEF LL_DMA_IsActiveFlag_DTE

LL_DMA_IsActiveFlag_HT

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_HT (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get half transfer flag.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CSR HTF LL_DMA_IsActiveFlag_HT

LL_DMA_IsActiveFlag_TC

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TC (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get transfer complete flag.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CSR TCF LL_DMA_IsActiveFlag_TC

LL_DMA_IsActiveFlag_IDLE

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_IDLE (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get idle flag.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CSR IDLEF LL_DMA_IsActiveFlag_IDLE

LL_DMA_IsActiveFlag_MIS

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_MIS (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Check if nsecure masked interrupt is active.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- MISR MISx LL_DMA_IsActiveFlag_MIS

LL_DMA_IsActiveFlag_SMIS

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_SMIS (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Check if secure masked interrupt is active.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- SMISR MISx LL_DMA_IsActiveFlag_SMIS

LL_DMA_EnableIT_TO

Function name

__STATIC_INLINE void LL_DMA_EnableIT_TO (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Enable trigger overrun interrupt.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR TOIE LL_DMA_EnableIT_TO

LL_DMA_EnableIT_SUSP

Function name

__STATIC_INLINE void LL_DMA_EnableIT_SUSP (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Enable suspension interrupt.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR SUSPIE LL_DMA_EnableIT_SUSP

LL_DMA_EnableIT_USE

Function name

__STATIC_INLINE void LL_DMA_EnableIT_USE (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Enable user setting error interrupt.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR USEIE LL_DMA_EnableIT_USE

LL_DMA_EnableIT_ULE

Function name

__STATIC_INLINE void LL_DMA_EnableIT_ULE (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Enable update link transfer error interrupt.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR ULEIE LL_DMA_EnableIT_ULE

LL_DMA_EnableIT_DTE

Function name

__STATIC_INLINE void LL_DMA_EnableIT_DTE (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Enable data transfer error interrupt.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR DTEIE LL_DMA_EnableIT_DTE

LL_DMA_EnableIT_HT

Function name

__STATIC_INLINE void LL_DMA_EnableIT_HT (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Enable half transfer complete interrupt.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR HTIE LL_DMA_EnableIT_HT

LL_DMA_EnableIT_TC

Function name

__STATIC_INLINE void LL_DMA_EnableIT_TC (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Enable transfer complete interrupt.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR TCIE LL_DMA_EnableIT_TC

LL_DMA_DisableIT_TO

Function name

__STATIC_INLINE void LL_DMA_DisableIT_TO (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Disable trigger overrun interrupt.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR TOIE LL_DMA_DisableIT_TO

LL_DMA_DisableIT_SUSP

Function name

__STATIC_INLINE void LL_DMA_DisableIT_SUSP (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Disable suspension interrupt.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR SUSPIE LL_DMA_DisableIT_SUSP

LL_DMA_DisableIT_USE

Function name

__STATIC_INLINE void LL_DMA_DisableIT_USE (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Disable user setting error interrupt.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR USEIE LL_DMA_DisableIT_USE

LL_DMA_DisableIT_ULE

Function name

__STATIC_INLINE void LL_DMA_DisableIT_ULE (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Disable update link transfer error interrupt.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR ULEIE LL_DMA_DisableIT_ULE

LL_DMA_DisableIT_DTE

Function name

__STATIC_INLINE void LL_DMA_DisableIT_DTE (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Disable data transfer error interrupt.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR DTEIE LL_DMA_DisableIT_DTE

LL_DMA_DisableIT_HT

Function name

__STATIC_INLINE void LL_DMA_DisableIT_HT (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Disable half transfer complete interrupt.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR HTIE LL_DMA_DisableIT_HT

LL_DMA_DisableIT_TC

Function name

__STATIC_INLINE void LL_DMA_DisableIT_TC (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Disable transfer complete interrupt.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR TCIE LL_DMA_DisableIT_TC

LL_DMA_IsEnabledIT_TO

Function name

__STATIC_INLINE uint32_t LL_DMA_IsEnabledIT_TO (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Check if trigger overrun interrupt is enabled.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR TOIE LL_DMA_IsEnabledIT_TO

LL_DMA_IsEnabledIT_SUSP

Function name

__STATIC_INLINE uint32_t LL_DMA_IsEnabledIT_SUSP (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Check if suspension interrupt is enabled.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR SUSPIE LL_DMA_IsEnabledIT_SUSP

LL_DMA_IsEnabledIT_USE

Function name

__STATIC_INLINE uint32_t LL_DMA_IsEnabledIT_USE (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Check if user setting error interrupt is enabled.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR USEIE LL_DMA_IsEnabledIT_USE

LL_DMA_IsEnabledIT_ULE

Function name

__STATIC_INLINE uint32_t LL_DMA_IsEnabledIT_ULE (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Check if update link transfer error interrupt is enabled.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR ULEIE LL_DMA_IsEnabledIT_ULE

LL_DMA_IsEnabledIT_DTE

Function name

__STATIC_INLINE uint32_t LL_DMA_IsEnabledIT_DTE (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Check if data transfer error interrupt is enabled.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR DTEIE LL_DMA_IsEnabledIT_DTE

LL_DMA_IsEnabledIT_HT

Function name

__STATIC_INLINE uint32_t LL_DMA_IsEnabledIT_HT (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Check if half transfer complete interrupt is enabled.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR HTIE LL_DMA_IsEnabledIT_HT

LL_DMA_IsEnabledIT_TC

Function name

__STATIC_INLINE uint32_t LL_DMA_IsEnabledIT_TC (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Check if transfer complete interrupt is enabled.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR TCIE LL_DMA_IsEnabledIT_TC

LL_DMA_Init

Function name

uint32_t LL_DMA_Init (DMA_TypeDef * DMAx, uint32_t Channel, LL_DMA_InitTypeDef * DMA_InitStruct)

Function description

Initialize the DMA registers according to the specified parameters in DMA_InitStruct.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **DMA_InitStruct:** pointer to a LL_DMA_InitTypeDef structure.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS : DMA registers are initialized.
  - ERROR : Not applicable.

Notes

- This API is used for all available DMA channels.
- A software request transfer can be done once programming the direction field in memory to memory value.
- To convert DMAx_Channely Instance to DMAx Instance and Channely, use helper macros : LL_DMA_GET_INSTANCE LL_DMA_GET_CHANNEL
- Warning if AXI port is selected with HPDMA, the maximum source (and destination) length should be less than 17. Otherwise, an error will be returned and no initialization performed.

LL_DMA_DeInit

Function name

uint32_t LL_DMA_DeInit (DMA_TypeDef * DMAx, uint32_t Channel)

Function description

De-initialize the DMA registers to their default reset values.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS : DMA registers are de-initialized.
  - ERROR : DMA registers are not de-initialized.

Notes

- This API is used for all available DMA channels.
- To convert DMAx_Channely Instance to DMAx Instance and Channely, use helper macros : LL_DMA_GET_INSTANCE LL_DMA_GET_CHANNEL

LL_DMA_StructInit

Function name

void LL_DMA_StructInit (LL_DMA_InitTypeDef * DMA_InitStruct)

Function description

Set each LL_DMA_InitTypeDef field to default value.

Parameters

- **DMA_InitStruct:** Pointer to a LL_DMA_InitTypeDef structure.

Return values

- **None.:**

LL_DMA_ListStructInit

Function name

void LL_DMA_ListStructInit (LL_DMA_InitLinkedListTypeDef * DMA_InitLinkedListStruct)

Function description

Set each LL_DMA_InitLinkedListTypeDef field to default value.

Parameters

- **DMA_InitLinkedListStruct:** Pointer to a LL_DMA_InitLinkedListTypeDef structure.

Return values

- **None.:**

LL_DMA_NodeStructInit

Function name

void LL_DMA_NodeStructInit (LL_DMA_InitNodeTypeDef * DMA_InitNodeStruct)

Function description

Set each LL_DMA_InitNodeTypeDef field to default value.

Parameters

- **DMA_InitNodeStruct:** Pointer to a LL_DMA_InitNodeTypeDef structure.

Return values

- **None.:**

LL_DMA_List_Init

Function name

uint32_t LL_DMA_List_Init (DMA_TypeDef * DMAx, uint32_t Channel, LL_DMA_InitLinkedListTypeDef * DMA_InitLinkedListStruct)

Function description

Initialize the DMA linked list according to the specified parameters in LL_DMA_InitLinkedListTypeDef.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **DMA_InitLinkedListStruct:** pointer to a LL_DMA_InitLinkedListTypeDef structure.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS : DMA registers are initialized.
  - ERROR : Not applicable.

Notes

- This API is used for all available DMA channels.
- To convert DMAx_Channely Instance to DMAx Instance and Channely, use helper macros : LL_DMA_GET_INSTANCE LL_DMA_GET_CHANNEL

LL_DMA_List_DeInit

Function name

uint32_t LL_DMA_List_DeInit (DMA_TypeDef * DMAx, uint32_t Channel)

Function description

De-initialize the DMA linked list.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS : DMA registers are de-initialized.
  - ERROR : DMA registers are not de-initialized.

Notes

- This API is used for all available DMA channels.
- To convert DMAx_Channely Instance to DMAx Instance and Channely, use helper macros : LL_DMA_GET_INSTANCE LL_DMA_GET_CHANNEL

LL_DMA_CreateLinkNode

Function name

uint32_t LL_DMA_CreateLinkNode (const LL_DMA_InitNodeTypeDef * DMA_InitNodeStruct, LL_DMA_LinkNodeTypeDef * pNode)

Function description

Initializes DMA linked list node according to the specified parameters in the DMA_InitNodeStruct.

Parameters

- **DMA_InitNodeStruct:** Pointer to a LL_DMA_InitNodeTypeDef structure that contains linked list node registers configurations.
- **pNode:** Pointer to linked list node to fill according to LL_DMA_LinkNodeTypeDef parameters.

Return values

- **None:**

Notes

- Warning: the maximum allowed AXI burst length shall not exceed 16. Otherwise, an error will be returned and no initialization performed.

LL_DMA_ConnectLinkNode

Function name

void LL_DMA_ConnectLinkNode (LL_DMA_LinkNodeTypeDef * pPrevLinkNode, uint32_t PrevNodeCLLRIdx, LL_DMA_LinkNodeTypeDef * pNewLinkNode, uint32_t NewNodeCLLRIdx)

Function description

Connect Linked list Nodes.

Parameters

- **pPrevLinkNode:** Pointer to previous linked list node to be connected to new Linked list node.
- **PrevNodeCLLRIdx:** Offset of Previous Node CLLR register. This parameter can be a value of CLLR offset.
- **pNewLinkNode:** Pointer to new Linked list.
- **NewNodeCLLRIdx:** Offset of New Node CLLR register. This parameter can be a value of CLLR offset.

Return values

- **None:**

LL_DMA_DisconnectNextLinkNode

Function name

void LL_DMA_DisconnectNextLinkNode (LL_DMA_LinkNodeTypeDef * pLinkNode, uint32_t LinkNodeCLLRIdx)

Function description

Disconnect the next linked list node.

Parameters

- **pLinkNode:** Pointer to linked list node to be disconnected from the next one.
- **LinkNodeCLLRIdx:** Offset of Link Node CLLR register.

Return values

- **None.:**

91.3 DMA Firmware driver defines

The following section lists the various define and macros of the module.

91.3.1 DMA

DMA

Block Hardware Request

LL_DMA_HWREQUEST_SINGLEBURST

Hardware request is driven by a peripheral with a hardware request/acknowledge protocol at a burst level

LL_DMA_HWREQUEST_BLK

Hardware request is driven by a peripheral with a hardware request/acknowledge protocol at a block level

Block Repeat Destination Address Update Mode

LL_DMA_BLKRPT_DEST_ADDR_INCREMENT

Destination address is incremented after each block transfer by destination update value

LL_DMA_BLKRPT_DEST_ADDR_DECREMENT

Destination address is decremented after each block transfer by destination update value

Block Repeat Source Address Update Mode

LL_DMA_BLKRPT_SRC_ADDR_INCREMENT

Source address pointer is incremented after each block transfer by source update value

LL_DMA_BLKRPT_SRC_ADDR_DECREMENT

Source address pointer is decremented after each block transfer by source update value

Channel

LL_DMA_CHANNEL_0

LL_DMA_CHANNEL_1

LL_DMA_CHANNEL_2

LL_DMA_CHANNEL_3

LL_DMA_CHANNEL_4

LL_DMA_CHANNEL_5

LL_DMA_CHANNEL_6

LL_DMA_CHANNEL_7

LL_DMA_CHANNEL_8

LL_DMA_CHANNEL_9

LL_DMA_CHANNEL_10

LL_DMA_CHANNEL_11

LL_DMA_CHANNEL_12

LL_DMA_CHANNEL_13

LL_DMA_CHANNEL_14

LL_DMA_CHANNEL_15

LL_DMA_CHANNEL_ALL

Static Isolation CID

LL_DMA_CHANNEL_STATIC_CID_0

Channel is assigned to CID0

LL_DMA_CHANNEL_STATIC_CID_1

Channel is assigned to CID1

LL_DMA_CHANNEL_STATIC_CID_2

Channel is assigned to CID2

LL_DMA_CHANNEL_STATIC_CID_3

Channel is assigned to CID3

LL_DMA_CHANNEL_STATIC_CID_4

Channel is assigned to CID4

LL_DMA_CHANNEL_STATIC_CID_5

Channel is assigned to CID5

LL_DMA_CHANNEL_STATIC_CID_6

Channel is assigned to CID6

CLLR offset

LL_DMA_CLLR_OFFSET0

LL_DMA_CLLR_OFFSET1

LL_DMA_CLLR_OFFSET2

LL_DMA_CLLR_OFFSET3

LL_DMA_CLLR_OFFSET4

LL_DMA_CLLR_OFFSET5

LL_DMA_CLLR_OFFSET6

LL_DMA_CLLR_OFFSET7

Data Alignment

LL_DMA_DATA_ALIGN_ZEROPADD

If src data width < dest data width : => Right Aligned padded with 0 up to destination data width. If src data width > dest data width : => Right Aligned Left Truncated down to destination data width.

LL_DMA_DATA_ALIGN_SIGNEXTPADD

If src data width < dest data width : => Right Aligned padded with sign extended up to destination data width. If src data width > dest data width : => Left Aligned Right Truncated down to the destination data width

LL_DMA_DATA_PACK_UNPACK

If src data width < dest data width : => Packed at the destination data width If src data width > dest data width : => Unpacked at the destination data width

Destination Allocated Port

LL_DMA_DEST_ALLOCATED_PORT0

Destination Allocated Port 0

LL_DMA_DEST_ALLOCATED_PORT1

Destination Allocated Port 1

Destination Data Width

LL_DMA_DEST_DATAWIDTH_BYTE

Destination Data Width : Byte

LL_DMA_DEST_DATAWIDTH_HALFWORD

Destination Data Width : HalfWord

LL_DMA_DEST_DATAWIDTH_WORD

Destination Data Width : Word

LL_DMA_DEST_DATAWIDTH_DOUBLEWORD

Destination Data Width : DoubleWord

Destination Increment Mode

LL_DMA_DEST_FIXED

Destination fixed single/burst

LL_DMA_DEST_INCREMENT

Destination incremented single/burst

Destination Security Attribute

LL_DMA_CHANNEL_DEST_NSEC

NSecure transfer from the destination

LL_DMA_CHANNEL_DEST_SEC

Secure transfer from the destination

Burst Destination Address Update Mode

LL_DMA_BURST_DEST_ADDR_INCREMENT

Destination address pointer is incremented after each burst transfer by destination update value

LL_DMA_BURST_DEST_ADDR_DECREMENT

Destination address pointer is decremented after each burst transfer by destination update value

Destination Byte Exchange

LL_DMA_DEST_BYTE_PRESERVE

No destination Byte exchange when destination data width > Byte

LL_DMA_DEST_BYTE_EXCHANGE

Destination Byte exchange when destination data width > Byte

Destination Half-Word Exchange

LL_DMA_DEST_HALFWORD_PRESERVE

No destination Half-Word exchange when destination data width is word

LL_DMA_DEST_HALFWORD_EXCHANGE

Destination Half-Word exchange when destination data width is word

Destination Word Exchange

LL_DMA_DEST_WORD_PRESERVE

No destination Word exchange when destination data width is double-word

LL_DMA_DEST_WORD_EXCHANGE

Destination Word exchange when destination data width is double-word

Linked list node type

LL_DMA_GPDMA_LINEAR_NODE

GPDMA node : linear addressing node

LL_DMA_GPDMA_2D_NODE

GPDMA node : 2 dimension addressing node

LL_DMA_HPDMA_LINEAR_NODE

HPDMA node : linear addressing node

LL_DMA_HPDMA_2D_NODE

HPDMA node : 2 dimension addressing node

Linked list register update

LL_DMA_UPDATE_CTR1

Update CTR1 register from memory : available for all DMA channels

LL_DMA_UPDATE_CTR2

Update CTR2 register from memory : available for all DMA channels

LL_DMA_UPDATE_CBR1

Update CBR1 register from memory : available for all DMA channels

LL_DMA_UPDATE_CSAR

Update CSAR register from memory : available for all DMA channels

LL_DMA_UPDATE_CDAR

Update CDAR register from memory : available for all DMA channels

LL_DMA_UPDATE_CTR3

Update CTR3 register from memory : available only for 2D addressing DMA channels

LL_DMA_UPDATE_CBR2

Update CBR2 register from memory : available only for 2D addressing DMA channels

LL_DMA_UPDATE_CLLR

Update CLLR register from memory : available for all DMA channels

Linked List Allocated Port

LL_DMA_LINK_ALLOCATED_PORT0

Linked List Allocated Port 0

LL_DMA_LINK_ALLOCATED_PORT1

Linked List Allocated Port 1

Link Step Mode

LL_DMA_LSM_FULL_EXECUTION

Channel execute the full linked list

LL_DMA_LSM_1LINK_EXECUTION

Channel execute one node of the linked list

Priority Level

LL_DMA_LOW_PRIORITY_LOW_WEIGHT

Priority level : Low Priority, Low Weight

LL_DMA_LOW_PRIORITY_MID_WEIGHT

Priority level : Low Priority, Mid Weight

LL_DMA_LOW_PRIORITY_HIGH_WEIGHT

Priority level : Low Priority, High Weight

LL_DMA_HIGH_PRIORITY

Priority level : High Priority

Request Selection

LL_HPDMA1_REQUEST_JPEG_RX

HPDMA1 HW request is JPEG_RX

LL_HPDMA1_REQUEST_JPEG_TX

HPDMA1 HW request is JPEG_TX

LL_HPDMA1_REQUEST_OCTOSPI1

HPDMA1 HW request is OCTOSPI1

LL_HPDMA1_REQUEST_OCTOSPI2

HPDMA1 HW request is OCTOSPI2

LL_HPDMA1_REQUEST_OCTOSPI3

HPDMA1 HW request is OCTOSPI3

LL_HPDMA1_REQUEST_FMC2_TXRX

HPDMA1 HW request is FMC2_TXRX

LL_HPDMA1_REQUEST_FMC2_BCH

HPDMA1 HW request is FMC2_BCH

LL_HPDMA1_REQUEST_ADC1

HPDMA1 HW request is ADC1

LL_HPDMA1_REQUEST_ADC2

HPDMA1 HW request is ADC2

LL_HPDMA1_REQUEST_CRYP_IN

HPDMA1 HW request is CRYP_IN

LL_HPDMA1_REQUEST_CRYP_OUT

HPDMA1 HW request is CRYP_OUT

LL_HPDMA1_REQUEST_SAES_OUT

HPDMA1 HW request is SAES_OUT

LL_HPDMA1_REQUEST_SAES_IN

HPDMA1 HW request is SAES_IN

LL_HPDMA1_REQUEST_HASH_IN

HPDMA1 HW request is HASH_IN

LL_HPDMA1_REQUEST_TIM1_CH1

HPDMA1 HW request is TIM1_CH1

LL_HPDMA1_REQUEST_TIM1_CH2

HPDMA1 HW request is TIM1_CH2

LL_HPDMA1_REQUEST_TIM1_CH3

HPDMA1 HW request is TIM1_CH3

LL_HPDMA1_REQUEST_TIM1_CH4

HPDMA1 HW request is TIM1_CH4

LL_HPDMA1_REQUEST_TIM1_UP

HPDMA1 HW request is TIM1_UP

LL_HPDMA1_REQUEST_TIM1_TRIG

HPDMA1 HW request is TIM1_TRIG

LL_HPDMA1_REQUEST_TIM1_COM

HPDMA1 HW request is TIM1_COM

LL_HPDMA1_REQUEST_TIM2_CH1

HPDMA1 HW request is TIM2_CH1

LL_HPDMA1_REQUEST_TIM2_CH2

HPDMA1 HW request is TIM2_CH2

LL_HPDMA1_REQUEST_TIM2_CH3

HPDMA1 HW request is TIM2_CH3

LL_HPDMA1_REQUEST_TIM2_CH4

HPDMA1 HW request is TIM2_CH4

LL_HPDMA1_REQUEST_TIM2_UP

HPDMA1 HW request is TIM2_UP

LL_HPDMA1_REQUEST_TIM2_TRIG

HPDMA1 HW request is TIM2_TRIG

LL_HPDMA1_REQUEST_TIM3_CH1

HPDMA1 HW request is TIM3_CH1

LL_HPDMA1_REQUEST_TIM3_CH2

HPDMA1 HW request is TIM3_CH2

LL_HPDMA1_REQUEST_TIM3_CH3

HPDMA1 HW request is TIM3_CH3

LL_HPDMA1_REQUEST_TIM3_CH4

HPDMA1 HW request is TIM3_CH4

LL_HPDMA1_REQUEST_TIM3_UP

HPDMA1 HW request is TIM3_UP

LL_HPDMA1_REQUEST_TIM3_TRIG

HPDMA1 HW request is TIM3_TRIG

LL_HPDMA1_REQUEST_TIM4_CH1

HPDMA1 HW request is TIM4_CH1

LL_HPDMA1_REQUEST_TIM4_CH2

HPDMA1 HW request is TIM4_CH2

LL_HPDMA1_REQUEST_TIM4_CH3

HPDMA1 HW request is TIM4_CH3

LL_HPDMA1_REQUEST_TIM4_CH4

HPDMA1 HW request is TIM4_CH4

LL_HPDMA1_REQUEST_TIM4_UP

HPDMA1 HW request is TIM4_UP

LL_HPDMA1_REQUEST_TIM4_TRIG

HPDMA1 HW request is TIM4_TRIG

LL_HPDMA1_REQUEST_TIM5_CH1

HPDMA1 HW request is TIM5_CH1

LL_HPDMA1_REQUEST_TIM5_CH2

HPDMA1 HW request is TIM5_CH2

LL_HPDMA1_REQUEST_TIM5_CH3

HPDMA1 HW request is TIM5_CH3

LL_HPDMA1_REQUEST_TIM5_CH4

HPDMA1 HW request is TIM5_CH4

LL_HPDMA1_REQUEST_TIM5_UP

HPDMA1 HW request is TIM5_UP

LL_HPDMA1_REQUEST_TIM5_TRIG

HPDMA1 HW request is TIM5_TRIG

LL_HPDMA1_REQUEST_TIM6_UP

HPDMA1 HW request is TIM6_UP

LL_HPDMA1_REQUEST_TIM7_UP

HPDMA1 HW request is TIM7_UP

LL_HPDMA1_REQUEST_TIM8_CH1

HPDMA1 HW request is TIM8_CH1

LL_HPDMA1_REQUEST_TIM8_CH2

HPDMA1 HW request is TIM8_CH2

LL_HPDMA1_REQUEST_TIM8_CH3

HPDMA1 HW request is TIM8_CH3

LL_HPDMA1_REQUEST_TIM8_CH4

HPDMA1 HW request is TIM8_CH4

LL_HPDMA1_REQUEST_TIM8_UP

HPDMA1 HW request is TIM8_UP

LL_HPDMA1_REQUEST_TIM8_TRIG

HPDMA1 HW request is TIM8_TRIG

LL_HPDMA1_REQUEST_TIM8_COM

HPDMA1 HW request is TIM8_COM

LL_HPDMA1_REQUEST_TIM15_CH1

HPDMA1 HW request is TIM15_CH1

LL_HPDMA1_REQUEST_TIM15_CH2

HPDMA1 HW request is TIM15_CH2

LL_HPDMA1_REQUEST_TIM15_UP

HPDMA1 HW request is TIM15_UP

LL_HPDMA1_REQUEST_TIM15_TRIG

HPDMA1 HW request is TIM15_TRIG

LL_HPDMA1_REQUEST_TIM15_COM

HPDMA1 HW request is TIM15_COM

LL_HPDMA1_REQUEST_TIM16_CH1

HPDMA1 HW request is TIM16_CH1

LL_HPDMA1_REQUEST_TIM16_UP

HPDMA1 HW request is TIM16_UP

LL_HPDMA1_REQUEST_TIM16_COM

HPDMA1 HW request is TIM16_COM

LL_HPDMA1_REQUEST_TIM17_CH1

HPDMA1 HW request is TIM17_CH1

LL_HPDMA1_REQUEST_TIM17_UP

HPDMA1 HW request is TIM17_UP

LL_HPDMA1_REQUEST_TIM17_COM

HPDMA1 HW request is TIM17_COM

LL_HPDMA1_REQUEST_TIM18_CH1

HPDMA1 HW request is TIM18_CH1

LL_HPDMA1_REQUEST_TIM18_UP

HPDMA1 HW request is TIM18_UP

LL_HPDMA1_REQUEST_TIM18_COM

HPDMA1 HW request is TIM18_COM

LL_HPDMA1_REQUEST_LPTIM1_IC1

HPDMA1 HW request is LPTIM1_IC1

LL_HPDMA1_REQUEST_LPTIM1_IC2

HPDMA1 HW request is LPTIM1_IC2

LL_HPDMA1_REQUEST_LPTIM1_UE

HPDMA1 HW request is LPTIM1_UE

LL_HPDMA1_REQUEST_LPTIM2_IC1

HPDMA1 HW request is LPTIM2_IC1

LL_HPDMA1_REQUEST_LPTIM2_IC2

HPDMA1 HW request is LPTIM2_IC2

LL_HPDMA1_REQUEST_LPTIM2_UE

HPDMA1 HW request is LPTIM2_UE

LL_HPDMA1_REQUEST_LPTIM3_IC1

HPDMA1 HW request is LPTIM3_IC1

LL_HPDMA1_REQUEST_LPTIM3_IC2

HPDMA1 HW request is LPTIM3_IC2

LL_HPDMA1_REQUEST_LPTIM3_UE

HPDMA1 HW request is LPTIM3_UE

LL_HPDMA1_REQUEST_SPI1_RX

HPDMA1 HW request is SPI1_RX

LL_HPDMA1_REQUEST_SPI1_TX

HPDMA1 HW request is SPI1_TX

LL_HPDMA1_REQUEST_SPI2_RX

HPDMA1 HW request is SPI2_RX

LL_HPDMA1_REQUEST_SPI2_TX

HPDMA1 HW request is SPI2_TX

LL_HPDMA1_REQUEST_SPI3_RX

HPDMA1 HW request is SPI3_RX

LL_HPDMA1_REQUEST_SPI3_TX

HPDMA1 HW request is SPI3_TX

LL_HPDMA1_REQUEST_SPI4_RX

HPDMA1 HW request is SPI4_RX

LL_HPDMA1_REQUEST_SPI4_TX

HPDMA1 HW request is SPI4_TX

LL_HPDMA1_REQUEST_SPI5_RX

HPDMA1 HW request is SPI5_RX

LL_HPDMA1_REQUEST_SPI5_TX

HPDMA1 HW request is SPI5_TX

LL_HPDMA1_REQUEST_SPI6_RX

HPDMA1 HW request is SPI6_RX

LL_HPDMA1_REQUEST_SPI6_TX

HPDMA1 HW request is SPI6_TX

LL_HPDMA1_REQUEST_SAI1_A

HPDMA1 HW request is SAI1_A

LL_HPDMA1_REQUEST_SAI1_B

HPDMA1 HW request is SAI1_B

LL_HPDMA1_REQUEST_SAI2_A

HPDMA1 HW request is SAI2_A

LL_HPDMA1_REQUEST_SAI2_B

HPDMA1 HW request is SAI2_B

LL_HPDMA1_REQUEST_I2C1_RX

HPDMA1 HW request is I2C1_RX

LL_HPDMA1_REQUEST_I2C1_TX

HPDMA1 HW request is I2C1_TX

LL_HPDMA1_REQUEST_I2C2_RX

HPDMA1 HW request is I2C2_RX

LL_HPDMA1_REQUEST_I2C2_TX

HPDMA1 HW request is I2C2_TX

LL_HPDMA1_REQUEST_I2C3_RX

HPDMA1 HW request is I2C3_RX

LL_HPDMA1_REQUEST_I2C3_TX

HPDMA1 HW request is I2C3_TX

LL_HPDMA1_REQUEST_I2C4_RX

HPDMA1 HW request is I2C4_RX

LL_HPDMA1_REQUEST_I2C4_TX

HPDMA1 HW request is I2C4_TX

LL_HPDMA1_REQUEST_I3C1_RX

HPDMA1 HW request is I3C1_RX

LL_HPDMA1_REQUEST_I3C1_TX

HPDMA1 HW request is I3C1_TX

LL_HPDMA1_REQUEST_I3C2_RX

HPDMA1 HW request is I3C2_RX

LL_HPDMA1_REQUEST_I3C2_TX

HPDMA1 HW request is I3C2_TX

LL_HPDMA1_REQUEST_USART1_RX

HPDMA1 HW request is USART1_RX

LL_HPDMA1_REQUEST_USART1_TX

HPDMA1 HW request is USART1_TX

LL_HPDMA1_REQUEST_USART2_RX

HPDMA1 HW request is USART2_RX

LL_HPDMA1_REQUEST_USART2_TX

HPDMA1 HW request is USART2_TX

LL_HPDMA1_REQUEST_USART3_RX

HPDMA1 HW request is USART3_RX

LL_HPDMA1_REQUEST_USART3_TX

HPDMA1 HW request is USART3_TX

LL_HPDMA1_REQUEST_UART4_RX

HPDMA1 HW request is UART4_RX

LL_HPDMA1_REQUEST_UART4_TX

HPDMA1 HW request is UART4_TX

LL_HPDMA1_REQUEST_UART5_RX

HPDMA1 HW request is UART5_RX

LL_HPDMA1_REQUEST_UART5_TX

HPDMA1 HW request is UART5_TX

LL_HPDMA1_REQUEST_USART6_RX

HPDMA1 HW request is USART6_RX

LL_HPDMA1_REQUEST_USART6_TX

HPDMA1 HW request is USART6_TX

LL_HPDMA1_REQUEST_UART7_RX

HPDMA1 HW request is UART7_RX

LL_HPDMA1_REQUEST_UART7_TX

HPDMA1 HW request is UART7_TX

LL_HPDMA1_REQUEST_UART8_RX

HPDMA1 HW request is UART8_RX

LL_HPDMA1_REQUEST_UART8_TX

HPDMA1 HW request is UART8_TX

LL_HPDMA1_REQUEST_UART9_RX

HPDMA1 HW request is UART9_RX

LL_HPDMA1_REQUEST_UART9_TX

HPDMA1 HW request is UART9_TX

LL_HPDMA1_REQUEST_USART10_RX

HPDMA1 HW request is USART10_RX

LL_HPDMA1_REQUEST_USART10_TX

HPDMA1 HW request is USART10_TX

LL_HPDMA1_REQUEST_LPUART1_RX

HPDMA1 HW request is LPUART1_RX

LL_HPDMA1_REQUEST_LPUART1_TX

HPDMA1 HW request is LPUART1_TX

LL_HPDMA1_REQUEST_SPDIFRX_CS

HPDMA1 HW request is SPDIFRX_CS

LL_HPDMA1_REQUEST_SPDIFRX_DT

HPDMA1 HW request is SPDIFRX_DT

LL_HPDMA1_REQUEST_ADF1_FLT0

HPDMA1 HW request is ADF1_FLT0

LL_HPDMA1_REQUEST_MDF1_FLT0

HPDMA1 HW request is MDF1_FLT0

LL_HPDMA1_REQUEST_MDF1_FLT1

HPDMA1 HW request is MDF1_FLT1

LL_HPDMA1_REQUEST_MDF1_FLT2

HPDMA1 HW request is MDF1_FLT2

LL_HPDMA1_REQUEST_MDF1_FLT3

HPDMA1 HW request is MDF1_FLT3

LL_HPDMA1_REQUEST_MDF1_FLT4

HPDMA1 HW request is MDF1_FLT4

LL_HPDMA1_REQUEST_MDF1_FLT5

HPDMA1 HW request is MDF1_FLT5

LL_HPDMA1_REQUEST_UCPD1_TX

HPDMA1 HW request is UCPD1_TX

LL_HPDMA1_REQUEST_UCPD1_RX

HPDMA1 HW request is UCPD1_RX

LL_HPDMA1_REQUEST_DCMI_PSSI

HPDMA1 HW request is DCMI_PSSI

LL_HPDMA1_REQUEST_I3C1_TC

HPDMA1 HW request is I3C1_TC

LL_HPDMA1_REQUEST_I3C1_RS

HPDMA1 HW request is I3C1_RS

LL_HPDMA1_REQUEST_I3C2_TC

HPDMA1 HW request is I3C2_TC

LL_HPDMA1_REQUEST_I3C2_RS

HPDMA1 HW request is I3C2_RS

LL_GPDMA1_REQUEST_JPEG_RX

GPDMA1 HW request is JPEG_DMA_RX

LL_GPDMA1_REQUEST_JPEG_TX

GPDMA1 HW request is JPEG_DMA_TX

LL_GPDMA1_REQUEST_OCTOSPI1

GPDMA1 HW request is OCTOSPI1

LL_GPDMA1_REQUEST_OCTOSPI2

GPDMA1 HW request is OCTOSPI2

LL_GPDMA1_REQUEST_OCTOSPI3

GPDMA1 HW request is OCTOSPI3

LL_GPDMA1_REQUEST_FMC2_TXRX

GPDMA1 HW request is FMC2_TXRX

LL_GPDMA1_REQUEST_FMC2_BCH

GPDMA1 HW request is FMC2_BCH

LL_GPDMA1_REQUEST_ADC1

GPDMA1 HW request is ADC1

LL_GPDMA1_REQUEST_ADC2

GPDMA1 HW request is ADC2

LL_GPDMA1_REQUEST_CRYP_IN

GPDMA1 HW request is CRYP_IN

LL_GPDMA1_REQUEST_CRYP_OUT

GPDMA1 HW request is CRYP_OUT

LL_GPDMA1_REQUEST_SAES_OUT

GPDMA1 HW request is SAES_OUT

LL_GPDMA1_REQUEST_SAES_IN

GPDMA1 HW request is SAES_IN

LL_GPDMA1_REQUEST_HASH_IN

GPDMA1 HW request is HASH_IN

LL_GPDMA1_REQUEST_TIM1_CH1

GPDMA1 HW request is TIM1_CH1

LL_GPDMA1_REQUEST_TIM1_CH2

GPDMA1 HW request is TIM1_CH2

LL_GPDMA1_REQUEST_TIM1_CH3

GPDMA1 HW request is TIM1_CH3

LL_GPDMA1_REQUEST_TIM1_CH4

GPDMA1 HW request is TIM1_CH4

LL_GPDMA1_REQUEST_TIM1_UP

GPDMA1 HW request is TIM1_UP

LL_GPDMA1_REQUEST_TIM1_TRIG

GPDMA1 HW request is TIM1_TRIG

LL_GPDMA1_REQUEST_TIM1_COM

GPDMA1 HW request is TIM1_COM

LL_GPDMA1_REQUEST_TIM2_CH1

GPDMA1 HW request is TIM2_CH1

LL_GPDMA1_REQUEST_TIM2_CH2

GPDMA1 HW request is TIM2_CH2

LL_GPDMA1_REQUEST_TIM2_CH3

GPDMA1 HW request is TIM2_CH3

LL_GPDMA1_REQUEST_TIM2_CH4

GPDMA1 HW request is TIM2_CH4

LL_GPDMA1_REQUEST_TIM2_UP

GPDMA1 HW request is TIM2_UP

LL_GPDMA1_REQUEST_TIM2_TRIG

GPDMA1 HW request is TIM2_TRIG

LL_GPDMA1_REQUEST_TIM3_CH1

GPDMA1 HW request is TIM3_CH1

LL_GPDMA1_REQUEST_TIM3_CH2

GPDMA1 HW request is TIM3_CH2

LL_GPDMA1_REQUEST_TIM3_CH3

GPDMA1 HW request is TIM3_CH3

LL_GPDMA1_REQUEST_TIM3_CH4

GPDMA1 HW request is TIM3_CH4

LL_GPDMA1_REQUEST_TIM3_UP

GPDMA1 HW request is TIM3_UP

LL_GPDMA1_REQUEST_TIM3_TRIG

GPDMA1 HW request is TIM3_TRIG

LL_GPDMA1_REQUEST_TIM4_CH1

GPDMA1 HW request is TIM4_CH1

LL_GPDMA1_REQUEST_TIM4_CH2

GPDMA1 HW request is TIM4_CH2

LL_GPDMA1_REQUEST_TIM4_CH3

GPDMA1 HW request is TIM4_CH3

LL_GPDMA1_REQUEST_TIM4_CH4

GPDMA1 HW request is TIM4_CH4

LL_GPDMA1_REQUEST_TIM4_UP

GPDMA1 HW request is TIM4_UP

LL_GPDMA1_REQUEST_TIM4_TRIG

GPDMA1 HW request is TIM4_TRIG

LL_GPDMA1_REQUEST_TIM5_CH1

GPDMA1 HW request is TIM5_CH1

LL_GPDMA1_REQUEST_TIM5_CH2

GPDMA1 HW request is TIM5_CH2

LL_GPDMA1_REQUEST_TIM5_CH3

GPDMA1 HW request is TIM5_CH3

LL_GPDMA1_REQUEST_TIM5_CH4

GPDMA1 HW request is TIM5_CH4

LL_GPDMA1_REQUEST_TIM5_UP

GPDMA1 HW request is TIM5_UP

LL_GPDMA1_REQUEST_TIM5_TRIG

GPDMA1 HW request is TIM5_TRIG

LL_GPDMA1_REQUEST_TIM6_UP

GPDMA1 HW request is TIM6_UP

LL_GPDMA1_REQUEST_TIM7_UP

GPDMA1 HW request is TIM6_UP

LL_GPDMA1_REQUEST_TIM8_CH1

GPDMA1 HW request is TIM8_CH1

LL_GPDMA1_REQUEST_TIM8_CH2

GPDMA1 HW request is TIM8_CH2

LL_GPDMA1_REQUEST_TIM8_CH3

GPDMA1 HW request is TIM8_CH3

LL_GPDMA1_REQUEST_TIM8_CH4

GPDMA1 HW request is TIM8_CH4

LL_GPDMA1_REQUEST_TIM8_UP

GPDMA1 HW request is TIM8_UP

LL_GPDMA1_REQUEST_TIM8_TRIG

GPDMA1 HW request is TIM8_TRIG

LL_GPDMA1_REQUEST_TIM8_COM

GPDMA1 HW request is TIM8_COM

LL_GPDMA1_REQUEST_TIM15_CH1

GPDMA1 HW request is TIM15_CH1

LL_GPDMA1_REQUEST_TIM15_CH2

GPDMA1 HW request is TIM15_CH2

LL_GPDMA1_REQUEST_TIM15_UP

GPDMA1 HW request is TIM15_UP

LL_GPDMA1_REQUEST_TIM15_TRIG

GPDMA1 HW request is TIM15_TRIG

LL_GPDMA1_REQUEST_TIM15_COM

GPDMA1 HW request is TIM15_COM

LL_GPDMA1_REQUEST_TIM16_CH1

GPDMA1 HW request is TIM16_CH1

LL_GPDMA1_REQUEST_TIM16_UP

GPDMA1 HW request is TIM16_UP

LL_GPDMA1_REQUEST_TIM16_COM

GPDMA1 HW request is TIM16_COM

LL_GPDMA1_REQUEST_TIM17_CH1

GPDMA1 HW request is TIM17_CH1

LL_GPDMA1_REQUEST_TIM17_UP

GPDMA1 HW request is TIM17_UP

LL_GPDMA1_REQUEST_TIM17_COM

GPDMA1 HW request is TIM17_COM

LL_GPDMA1_REQUEST_TIM18_CH1

GPDMA1 HW request is TIM18_CH1

LL_GPDMA1_REQUEST_TIM18_UP

GPDMA1 HW request is TIM18_UP

LL_GPDMA1_REQUEST_TIM18_COM

GPDMA1 HW request is TIM18_COM

LL_GPDMA1_REQUEST_LPTIM1_IC1

GPDMA1 HW request is LPTIM1_IC1

LL_GPDMA1_REQUEST_LPTIM1_IC2

GPDMA1 HW request is LPTIM1_IC2

LL_GPDMA1_REQUEST_LPTIM1_UE

GPDMA1 HW request is LPTIM1_UE

LL_GPDMA1_REQUEST_LPTIM2_IC1

GPDMA1 HW request is LPTIM2_IC1

LL_GPDMA1_REQUEST_LPTIM2_IC2

GPDMA1 HW request is LPTIM2_IC2

LL_GPDMA1_REQUEST_LPTIM2_UE

GPDMA1 HW request is LPTIM2_UE

LL_GPDMA1_REQUEST_LPTIM3_IC1

GPDMA1 HW request is LPTIM3_IC1

LL_GPDMA1_REQUEST_LPTIM3_IC2

GPDMA1 HW request is LPTIM3_IC2

LL_GPDMA1_REQUEST_LPTIM3_UE

GPDMA1 HW request is LPTIM3_UE

LL_GPDMA1_REQUEST_SPI1_RX

GPDMA1 HW request is SPI1_RX

LL_GPDMA1_REQUEST_SPI1_TX

GPDMA1 HW request is SPI1_TX

LL_GPDMA1_REQUEST_SPI2_RX

GPDMA1 HW request is SPI2_RX

LL_GPDMA1_REQUEST_SPI2_TX

GPDMA1 HW request is SPI2_TX

LL_GPDMA1_REQUEST_SPI3_RX

GPDMA1 HW request is SPI3_RX

LL_GPDMA1_REQUEST_SPI3_TX

GPDMA1 HW request is SPI3_TX

LL_GPDMA1_REQUEST_SPI4_RX

GPDMA1 HW request is SPI4_RX

LL_GPDMA1_REQUEST_SPI4_TX

GPDMA1 HW request is SPI4_TX

LL_GPDMA1_REQUEST_SPI5_RX

GPDMA1 HW request is SPI5_RX

LL_GPDMA1_REQUEST_SPI5_TX

GPDMA1 HW request is SPI5_TX

LL_GPDMA1_REQUEST_SPI6_RX

GPDMA1 HW request is SPI6_RX

LL_GPDMA1_REQUEST_SPI6_TX

GPDMA1 HW request is SPI6_TX

LL_GPDMA1_REQUEST_SAI1_A

GPDMA1 HW request is SAI1_A

LL_GPDMA1_REQUEST_SAI1_B

GPDMA1 HW request is SAI1_B

LL_GPDMA1_REQUEST_SAI2_A

GPDMA1 HW request is SAI2_A

LL_GPDMA1_REQUEST_SAI2_B

GPDMA1 HW request is SAI2_B

LL_GPDMA1_REQUEST_I2C1_RX

GPDMA1 HW request is I2C1_RX

LL_GPDMA1_REQUEST_I2C1_TX

GPDMA1 HW request is I2C1_TX

LL_GPDMA1_REQUEST_I2C2_RX

GPDMA1 HW request is I2C2_RX

LL_GPDMA1_REQUEST_I2C2_TX

GPDMA1 HW request is I2C2_TX

LL_GPDMA1_REQUEST_I2C3_RX

GPDMA1 HW request is I2C3_RX

LL_GPDMA1_REQUEST_I2C3_TX

GPDMA1 HW request is I2C3_TX

LL_GPDMA1_REQUEST_I2C4_RX

GPDMA1 HW request is I2C4_RX

LL_GPDMA1_REQUEST_I2C4_TX

GPDMA1 HW request is I2C4_TX

LL_GPDMA1_REQUEST_I3C1_RX

GPDMA1 HW request is I3C1_RX

LL_GPDMA1_REQUEST_I3C1_TX

GPDMA1 HW request is I3C1_TX

LL_GPDMA1_REQUEST_I3C2_RX

GPDMA1 HW request is I3C2_RX

LL_GPDMA1_REQUEST_I3C2_TX

GPDMA1 HW request is I3C2_TX

LL_GPDMA1_REQUEST_USART1_RX

GPDMA1 HW request is USART1_RX

LL_GPDMA1_REQUEST_USART1_TX

GPDMA1 HW request is USART1_TX

LL_GPDMA1_REQUEST_USART2_RX

GPDMA1 HW request is USART2_RX

LL_GPDMA1_REQUEST_USART2_TX

GPDMA1 HW request is USART2_TX

LL_GPDMA1_REQUEST_USART3_RX

GPDMA1 HW request is USART3_RX

LL_GPDMA1_REQUEST_USART3_TX

GPDMA1 HW request is USART3_TX

LL_GPDMA1_REQUEST_UART4_RX

GPDMA1 HW request is UART4_RX

LL_GPDMA1_REQUEST_UART4_TX

GPDMA1 HW request is UART4_TX

LL_GPDMA1_REQUEST_UART5_RX

GPDMA1 HW request is UART5_RX

LL_GPDMA1_REQUEST_UART5_TX

GPDMA1 HW request is UART5_TX

LL_GPDMA1_REQUEST_USART6_RX

GPDMA1 HW request is USART6_RX

LL_GPDMA1_REQUEST_USART6_TX

GPDMA1 HW request is USART6_TX

LL_GPDMA1_REQUEST_UART7_RX

GPDMA1 HW request is UART7_RX

LL_GPDMA1_REQUEST_UART7_TX

GPDMA1 HW request is UART7_TX

LL_GPDMA1_REQUEST_UART8_RX

GPDMA1 HW request is UART8_RX

LL_GPDMA1_REQUEST_UART8_TX

GPDMA1 HW request is UART8_TX

LL_GPDMA1_REQUEST_UART9_RX

GPDMA1 HW request is UART9_RX

LL_GPDMA1_REQUEST_UART9_TX

GPDMA1 HW request is UART9_TX

LL_GPDMA1_REQUEST_USART10_RX

GPDMA1 HW request is USART10_RX

LL_GPDMA1_REQUEST_USART10_TX

GPDMA1 HW request is USART10_TX

LL_GPDMA1_REQUEST_LPUART1_RX

GPDMA1 HW request is LPUART1_RX

LL_GPDMA1_REQUEST_LPUART1_TX

GPDMA1 HW request is LPUART1_TX

LL_GPDMA1_REQUEST_SPDIFRX_CS

GPDMA1 HW request is SPDIFRX_CS

LL_GPDMA1_REQUEST_SPDIFRX_DT

GPDMA1 HW request is SPDIFRX_DT

LL_GPDMA1_REQUEST_ADF1_FLT0

GPDMA1 HW request is ADF1_FLT0

LL_GPDMA1_REQUEST_MDF1_FLT0

GPDMA1 HW request is MDF1_FLT0

LL_GPDMA1_REQUEST_MDF1_FLT1

GPDMA1 HW request is MDF1_FLT1

LL_GPDMA1_REQUEST_MDF1_FLT2

GPDMA1 HW request is MDF1_FLT2

LL_GPDMA1_REQUEST_MDF1_FLT3

GPDMA1 HW request is MDF1_FLT3

LL_GPDMA1_REQUEST_MDF1_FLT4

GPDMA1 HW request is MDF1_FLT4

LL_GPDMA1_REQUEST_MDF1_FLT5

GPDMA1 HW request is MDF1_FLT5

LL_GPDMA1_REQUEST_UCPD1_TX

GPDMA1 HW request is UCPD1_TX

LL_GPDMA1_REQUEST_UCPD1_RX

GPDMA1 HW request is UCPD1_RX

LL_GPDMA1_REQUEST_DCMI_PSSI

GPDMA1 HW request is DCMI_PSSI

LL_GPDMA1_REQUEST_I3C1_TC

GPDMA1 HW request is I3C1_TC

LL_GPDMA1_REQUEST_I3C1_RS

GPDMA1 HW request is I3C1_RS

LL_GPDMA1_REQUEST_I3C2_TC

GPDMA1 HW request is I3C2_TC

LL_GPDMA1_REQUEST_I3C2_RS

GPDMA1 HW request is I3C2_RS

Source Allocated Port

LL_DMA_SRC_ALLOCATED_PORT0

Source Allocated Port 0

LL_DMA_SRC_ALLOCATED_PORT1

Source Allocated Port 1

Source Data Width

LL_DMA_SRC_DATAWIDTH_BYTE

Source Data Width : Byte

LL_DMA_SRC_DATAWIDTH_HALFWORD

Source Data Width : HalfWord

LL_DMA_SRC_DATAWIDTH_WORD

Source Data Width : Word

LL_DMA_SRC_DATAWIDTH_DOUBLEWORD

Source Data Width : DoubleWord

Source Increment Mode

LL_DMA_SRC_FIXED

Source fixed single/burst

LL_DMA_SRC_INCREMENT

Source incremented single/burst

Source Security Attribute

LL_DMA_CHANNEL_NSEC

NSecure channel

LL_DMA_CHANNEL_SEC

Secure channel

LL_DMA_CHANNEL_SRC_NSEC

NSecure transfer from the source

LL_DMA_CHANNEL_SRC_SEC

Secure transfer from the source

Burst Source Address Update Mode

LL_DMA_BURST_SRC_ADDR_INCREMENT

Source address pointer is incremented after each burst transfer by source update value

LL_DMA_BURST_SRC_ADDR_DECREMENT

Source address pointer is decremented after each burst transfer by source update value

Source Byte Exchange

LL_DMA_SRC_BYTE_PRESERVE

No source Byte exchange when source data width is word

LL_DMA_SRC_BYTE_EXCHANGE

Source Byte exchange when source data width is word

Transfer Direction

LL_DMA_DIRECTION_MEMORY_TO_MEMORY

Memory to memory direction

LL_DMA_DIRECTION_PERIPH_TO_MEMORY

Peripheral to memory direction

LL_DMA_DIRECTION_MEMORY_TO_PERIPH

Memory to peripheral direction

Transfer Event Mode

LL_DMA_TCEM_BLK_TRANSFER

The TC (and the HT) event is generated at the (respectively half) end of each block

LL_DMA_TCEM_RPT_BLK_TRANSFER

The TC (and the HT) event is generated at the (respectively half) end of the repeated block

LL_DMA_TCEM_EACH_LLITEM_TRANSFER

The TC (and the HT) event is generated at the (respectively half) end of each linked-list item

LL_DMA_TCEM_LAST_LLITEM_TRANSFER

The TC (and the HT) event is generated at the (respectively half) end of the last linked-list item

Transfer Trigger Mode

LL_DMA_TRIGM_BLK_TRANSFER

A block transfer is conditioned by (at least) one hit trigger

LL_DMA_TRIGM_RPT_BLK_TRANSFER

A repeated block transfer is conditioned by (at least) one hit trigger

LL_DMA_TRIGM_LLI_LINK_TRANSFER

A LLI link transfer is conditioned by (at least) one hit trigger

LL_DMA_TRIGM_SINGLBURST_TRANSFER

A Single/Burst transfer is conditioned by (at least) one hit trigger

Trigger Polarity

LL_DMA_TRIG_POLARITY_MASKED

No trigger of the selected DMA request. Masked trigger event

LL_DMA_TRIG_POLARITY_RISING

Trigger of the selected DMA request on the rising edge of the selected trigger event input

LL_DMA_TRIG_POLARITY_FALLING

Trigger of the selected DMA request on the falling edge of the selected trigger event input

Trigger Selection

LL_HPDMA1_TRIGGER_DCMIPP_P0_FEND

HPDMA1 HW Trigger signal is DCMIPP_P0_FEND

LL_HPDMA1_TRIGGER_DCMIPP_P0_LEND

HPDMA1 HW Trigger signal is DCMIPP_P0_LEND

LL_HPDMA1_TRIGGER_DCMIPP_P0_HSYNC

HPDMA1 HW Trigger signal is DCMIPP_P0_HSYNC

LL_HPDMA1_TRIGGER_DCMIPP_P0_VSYNC

HPDMA1 HW Trigger signal is DCMIPP_P0_VSYNC

LL_HPDMA1_TRIGGER_DCMIPP_P1_FEND

HPDMA1 HW Trigger signal is DCMIPP_P1_FEND

LL_HPDMA1_TRIGGER_DCMIPP_P1_LEND

HPDMA1 HW Trigger signal is DCMIPP_P1_LEND

LL_HPDMA1_TRIGGER_DCMIPP_P1_HSYNC

HPDMA1 HW Trigger signal is DCMIPP_P1_HSYNC

LL_HPDMA1_TRIGGER_DCMIPP_P1_VSYNC

HPDMA1 HW Trigger signal is DCMIPP_P1_VSYNC

LL_HPDMA1_TRIGGER_DCMIPP_P2_FEND

HPDMA1 HW Trigger signal is DCMIPP_P2_FEND

LL_HPDMA1_TRIGGER_DCMIPP_P2_LEND

HPDMA1 HW Trigger signal is DCMIPP_P2_LEND

LL_HPDMA1_TRIGGER_DCMIPP_P2_HSYNC

HPDMA1 HW Trigger signal is DCMIPP_P2_HSYNC

LL_HPDMA1_TRIGGER_DCMIPP_P2_VSYNC

HPDMA1 HW Trigger signal is DCMIPP_P2_VSYNC

LL_HPDMA1_TRIGGER_DMA2D_CTC

HPDMA1 HW Trigger signal is DMA2D_CTC

LL_HPDMA1_TRIGGER_DMA2D_TC

HPDMA1 HW Trigger signal is DMA2D_TC

LL_HPDMA1_TRIGGER_DMA2D_TW

HPDMA1 HW Trigger signal is DMA2D_TW

LL_HPDMA1_TRIGGER_JPEG_EOC

HPDMA1 HW Trigger signal is JPEG_EOC

LL_HPDMA1_TRIGGER_JPEG_IFNF

HPDMA1 HW Trigger signal is JPEG_IFNF

LL_HPDMA1_TRIGGER_JPEG_IFT

HPDMA1 HW Trigger signal is JPEG_IFT

LL_HPDMA1_TRIGGER_JPEG_OFNE

HPDMA1 HW Trigger signal is JPEG_OFNE

LL_HPDMA1_TRIGGER_JPEG_OFT

HPDMA1 HW Trigger signal is JPEG_OFT

LL_HPDMA1_TRIGGER_LCD_LI

HPDMA1 HW Trigger signal is LCD_LI

LL_HPDMA1_TRIGGER_GPU2D1_GP_0

HPDMA1 HW Trigger signal is GPU2D1_GP_0

LL_HPDMA1_TRIGGER_GPU2D1_GP_1

HPDMA1 HW Trigger signal is GPU2D1_GP_1

LL_HPDMA1_TRIGGER_GPU2D1_GP_2

HPDMA1 HW Trigger signal is GPU2D1_GP_2

LL_HPDMA1_TRIGGER_GPU2D1_GP_3

HPDMA1 HW Trigger signal is GPU2D1_GP_3

LL_HPDMA1_TRIGGER_GFXTIM_3

HPDMA1 HW Trigger signal is GFXTIM_3

LL_HPDMA1_TRIGGER_GFXTIM_2

HPDMA1 HW Trigger signal is GFXTIM_2

LL_HPDMA1_TRIGGER_GFXTIM_1

HPDMA1 HW Trigger signal is GFXTIM_1

LL_HPDMA1_TRIGGER_GFXTIM_0

HPDMA1 HW Trigger signal is GFXTIM_0

LL_HPDMA1_TRIGGER_LPTIM1_CH1

HPDMA1 HW Trigger signal is LPTIM1_CH1

LL_HPDMA1_TRIGGER_LPTIM1_CH2

HPDMA1 HW Trigger signal is LPTIM1_CH2

LL_HPDMA1_TRIGGER_LPTIM2_CH1

HPDMA1 HW Trigger signal is LPTIM2_CH1

LL_HPDMA1_TRIGGER_LPTIM2_CH2

HPDMA1 HW Trigger signal is LPTIM2_CH2

LL_HPDMA1_TRIGGER_LPTIM3_CH1

HPDMA1 HW Trigger signal is LPTIM3_CH1

LL_HPDMA1_TRIGGER_LPTIM3_CH2

HPDMA1 HW Trigger signal is LPTIM3_CH2

LL_HPDMA1_TRIGGER_LPTIM4_OUT

HPDMA1 HW Trigger signal is LPTIM4_OUT

LL_HPDMA1_TRIGGER_LPTIM5_OUT

HPDMA1 HW Trigger signal is LPTIM5_OUT

LL_HPDMA1_TRIGGER_RTC_WKUP

HPDMA1 HW Trigger signal is RTC_WKUP

LL_HPDMA1_TRIGGER_LPUART1_IT_R_WUP_ASYNC

HPDMA1 HW Trigger signal is IT_R_WUP_ASYNC

LL_HPDMA1_TRIGGER_LPUART1_IT_T_WUP_ASYNC

HPDMA1 HW Trigger signal is IT_T_WUP_ASYNC

LL_HPDMA1_TRIGGER_SPI6_IT_OR_SPI6_AIT_SYNC

HPDMA1 HW Trigger signal is SPI6_IT_OR_SPI6_AIT_SYNC

LL_HPDMA1_TRIGGER_TIM1_TRGO

HPDMA1 HW Trigger signal is TIM1_TRGO

LL_HPDMA1_TRIGGER_TIM1_TRGO2

HPDMA1 HW Trigger signal is TIM1_TRGO2

LL_HPDMA1_TRIGGER_TIM2_TRGO

HPDMA1 HW Trigger signal is TIM2_TRGO

LL_HPDMA1_TRIGGER_TIM3_TRGO

HPDMA1 HW Trigger signal is TIM3_TRGO

LL_HPDMA1_TRIGGER_TIM4_TRGO

HPDMA1 HW Trigger signal is TIM4_TRGO

LL_HPDMA1_TRIGGER_TIM5_TRGO

HPDMA1 HW Trigger signal is TIM5_TRGO

LL_HPDMA1_TRIGGER_TIM6_TRGO

HPDMA1 HW Trigger signal is TIM6_TRGO

LL_HPDMA1_TRIGGER_TIM7_TRGO

HPDMA1 HW Trigger signal is TIM7_TRGO

LL_HPDMA1_TRIGGER_TIM8_TRGO

HPDMA1 HW Trigger signal is TIM8_TRGO

LL_HPDMA1_TRIGGER_TIM8_TRGO2

HPDMA1 HW Trigger signal is TIM8_TRGO2

LL_HPDMA1_TRIGGER_TIM12_TRGO

HPDMA1 HW Trigger signal is TIM12_TRGO

LL_HPDMA1_TRIGGER_TIM15_TRGO

HPDMA1 HW Trigger signal is TIM15_TRGO

LL_HPDMA1_TRIGGER_HPDMA1_CH0_TCF

HPDMA1 HW Trigger signal is HPDMA1_CH0_TCF

LL_HPDMA1_TRIGGER_HPDMA1_CH1_TCF

HPDMA1 HW Trigger signal is HPDMA1_CH1_TCF

LL_HPDMA1_TRIGGER_HPDMA1_CH2_TCF

HPDMA1 HW Trigger signal is HPDMA1_CH2_TCF

LL_HPDMA1_TRIGGER_HPDMA1_CH3_TCF

HPDMA1 HW Trigger signal is HPDMA1_CH3_TCF

LL_HPDMA1_TRIGGER_HPDMA1_CH4_TCF

HPDMA1 HW Trigger signal is HPDMA1_CH4_TCF

LL_HPDMA1_TRIGGER_HPDMA1_CH5_TCF

HPDMA1 HW Trigger signal is HPDMA1_CH5_TCF

LL_HPDMA1_TRIGGER_HPDMA1_CH6_TCF

HPDMA1 HW Trigger signal is HPDMA1_CH6_TCF

LL_HPDMA1_TRIGGER_HPDMA1_CH7_TCF

HPDMA1 HW Trigger signal is HPDMA1_CH7_TCF

LL_HPDMA1_TRIGGER_HPDMA1_CH8_TCF

HPDMA1 HW Trigger signal is HPDMA1_CH8_TCF

LL_HPDMA1_TRIGGER_HPDMA1_CH9_TCF

HPDMA1 HW Trigger signal is HPDMA1_CH9_TCF

LL_HPDMA1_TRIGGER_HPDMA1_CH10_TCF

HPDMA1 HW Trigger signal is HPDMA1_CH10_TCF

LL_HPDMA1_TRIGGER_HPDMA1_CH11_TCF

HPDMA1 HW Trigger signal is HPDMA1_CH11_TCF

LL_HPDMA1_TRIGGER_HPDMA1_CH12_TCF

HPDMA1 HW Trigger signal is HPDMA1_CH12_TCF

LL_HPDMA1_TRIGGER_HPDMA1_CH13_TCF

HPDMA1 HW Trigger signal is HPDMA1_CH13_TCF

LL_HPDMA1_TRIGGER_HPDMA1_CH14_TCF

HPDMA1 HW Trigger signal is HPDMA1_CH14_TCF

LL_HPDMA1_TRIGGER_HPDMA1_CH15_TCF

HPDMA1 HW Trigger signal is HPDMA1_CH15_TCF

LL_HPDMA1_TRIGGER_GPDMA1_CH0_TCF

HPDMA1 HW Trigger signal is GPDMA1_CH0_TCF

LL_HPDMA1_TRIGGER_GPDMA1_CH1_TCF

HPDMA1 HW Trigger signal is GPDMA1_CH1_TCF

LL_HPDMA1_TRIGGER_GPDMA1_CH2_TCF

HPDMA1 HW Trigger signal is GPDMA1_CH2_TCF

LL_HPDMA1_TRIGGER_GPDMA1_CH3_TCF

HPDMA1 HW Trigger signal is GPDMA1_CH3_TCF

LL_HPDMA1_TRIGGER_GPDMA1_CH4_TCF

HPDMA1 HW Trigger signal is GPDMA1_CH4_TCF

LL_HPDMA1_TRIGGER_GPDMA1_CH5_TCF

HPDMA1 HW Trigger signal is GPDMA1_CH5_TCF

LL_HPDMA1_TRIGGER_GPDMA1_CH6_TCF

HPDMA1 HW Trigger signal is GPDMA1_CH6_TCF

LL_HPDMA1_TRIGGER_GPDMA1_CH7_TCF

HPDMA1 HW Trigger signal is GPDMA1_CH7_TCF

LL_HPDMA1_TRIGGER_GPDMA1_CH8_TCF

HPDMA1 HW Trigger signal is GPDMA1_CH8_TCF

LL_HPDMA1_TRIGGER_GPDMA1_CH9_TCF

HPDMA1 HW Trigger signal is GPDMA1_CH9_TCF

LL_HPDMA1_TRIGGER_GPDMA1_CH10_TCF

HPDMA1 HW Trigger signal is GPDMA1_CH10_TCF

LL_HPDMA1_TRIGGER_GPDMA1_CH11_TCF

HPDMA1 HW Trigger signal is GPDMA1_CH11_TCF

LL_HPDMA1_TRIGGER_GPDMA1_CH12_TCF

HPDMA1 HW Trigger signal is GPDMA1_CH12_TCF

LL_HPDMA1_TRIGGER_GPDMA1_CH13_TCF

HPDMA1 HW Trigger signal is GPDMA1_CH13_TCF

LL_HPDMA1_TRIGGER_GPDMA1_CH14_TCF

HPDMA1 HW Trigger signal is GPDMA1_CH14_TCF

LL_HPDMA1_TRIGGER_GPDMA1_CH15_TCF

HPDMA1 HW Trigger signal is GPDMA1_CH15_TCF

LL_HPDMA1_TRIGGER_EXTIT0_SYNC

HPDMA1 HW Trigger signal is EXTIT0_SYNC

LL_HPDMA1_TRIGGER_EXTIT1_SYNC

HPDMA1 HW Trigger signal is EXTIT1_SYNC

LL_HPDMA1_TRIGGER_EXTIT2_SYNC

HPDMA1 HW Trigger signal is EXTIT2_SYNC

LL_HPDMA1_TRIGGER_EXTIT3_SYNC

HPDMA1 HW Trigger signal is EXTIT3_SYNC

LL_HPDMA1_TRIGGER_EXTIT4_SYNC

HPDMA1 HW Trigger signal is EXTIT4_SYNC

LL_HPDMA1_TRIGGER_EXTIT5_SYNC

HPDMA1 HW Trigger signal is EXTIT5_SYNC

LL_HPDMA1_TRIGGER_EXTIT6_SYNC

HPDMA1 HW Trigger signal is EXTIT6_SYNC

LL_HPDMA1_TRIGGER_EXTIT7_SYNC

HPDMA1 HW Trigger signal is EXTIT7_SYNC

LL_HPDMA1_TRIGGER_EXTIT8_SYNC

HPDMA1 HW Trigger signal is EXTIT8_SYNC

LL_HPDMA1_TRIGGER_EXTIT9_SYNC

HPDMA1 HW Trigger signal is EXTIT9_SYNC

LL_HPDMA1_TRIGGER_EXTIT10_SYNC

HPDMA1 HW Trigger signal is EXTIT10_SYNC

LL_HPDMA1_TRIGGER_EXTIT11_SYNC

HPDMA1 HW Trigger signal is EXTIT11_SYNC

LL_HPDMA1_TRIGGER_EXTIT12_SYNC

HPDMA1 HW Trigger signal is EXTIT12_SYNC

LL_HPDMA1_TRIGGER_EXTIT13_SYNC

HPDMA1 HW Trigger signal is EXTIT13_SYNC

LL_HPDMA1_TRIGGER_EXTIT14_SYNC

HPDMA1 HW Trigger signal is EXTIT14_SYNC

LL_HPDMA1_TRIGGER_EXTIT15_SYNC

HPDMA1 HW Trigger signal is EXTIT15_SYNC

LL_GPDMA1_TRIGGER_DCMIPP_P0_FEND

GPDMA1 HW Trigger signal is DCMIPP_P0_FEND

LL_GPDMA1_TRIGGER_DCMIPP_P0_LEND

GPDMA1 HW Trigger signal is DCMIPP_P0_LEND

LL_GPDMA1_TRIGGER_DCMIPP_P0_HSYNC

GPDMA1 HW Trigger signal is DCMIPP_P0_HSYNC

LL_GPDMA1_TRIGGER_DCMIPP_P0_VSYNC

GPDMA1 HW Trigger signal is DCMIPP_P0_VSYNC

LL_GPDMA1_TRIGGER_DCMIPP_P1_FEND

GPDMA1 HW Trigger signal is DCMIPP_P1_FEND

LL_GPDMA1_TRIGGER_DCMIPP_P1_LEND

GPDMA1 HW Trigger signal is DCMIPP_P1_LEND

LL_GPDMA1_TRIGGER_DCMIPP_P1_HSYNC

GPDMA1 HW Trigger signal is DCMIPP_P1_HSYNC

LL_GPDMA1_TRIGGER_DCMIPP_P1_VSYNC

GPDMA1 HW Trigger signal is DCMIPP_P1_VSYNC

LL_GPDMA1_TRIGGER_DCMIPP_P2_FEND

GPDMA1 HW Trigger signal is DCMIPP_P2_FEND

LL_GPDMA1_TRIGGER_DCMIPP_P2_LEND

GPDMA1 HW Trigger signal is DCMIPP_P2_LEND

LL_GPDMA1_TRIGGER_DCMIPP_P2_HSYNC

GPDMA1 HW Trigger signal is DCMIPP_P2_HSYNC

LL_GPDMA1_TRIGGER_DCMIPP_P2_VSYNC

GPDMA1 HW Trigger signal is DCMIPP_P2_VSYNC

LL_GPDMA1_TRIGGER_DMA2D_CTC

GPDMA1 HW Trigger signal is DMA2D_CTC

LL_GPDMA1_TRIGGER_DMA2D_TC

GPDMA1 HW Trigger signal is DMA2D_TC

LL_GPDMA1_TRIGGER_DMA2D_TW

GPDMA1 HW Trigger signal is DMA2D_TW

LL_GPDMA1_TRIGGER_JPEG_EOC

GPDMA1 HW Trigger signal is JPEG_EOC

LL_GPDMA1_TRIGGER_JPEG_IFNF

GPDMA1 HW Trigger signal is JPEG_IFNF

LL_GPDMA1_TRIGGER_JPEG_IFT

GPDMA1 HW Trigger signal is JPEG_IFT

LL_GPDMA1_TRIGGER_JPEG_OFNE

GPDMA1 HW Trigger signal is JPEG_OFNE

LL_GPDMA1_TRIGGER_JPEG_OFT

GPDMA1 HW Trigger signal is JPEG_OFT

LL_GPDMA1_TRIGGER_LCD_LI

GPDMA1 HW Trigger signal is LCD_LI

LL_GPDMA1_TRIGGER_GPU2D1_GP_0

GPDMA1 HW Trigger signal is GPU2D1_GP_0

LL_GPDMA1_TRIGGER_GPU2D1_GP_1

GPDMA1 HW Trigger signal is GPU2D1_GP_1

LL_GPDMA1_TRIGGER_GPU2D1_GP_2

GPDMA1 HW Trigger signal is GPU2D1_GP_2

LL_GPDMA1_TRIGGER_GPU2D1_GP_3

GPDMA1 HW Trigger signal is GPU2D1_GP_3

LL_GPDMA1_TRIGGER_GFXTIM_3

GPDMA1 HW Trigger signal is GFXTIM_3

LL_GPDMA1_TRIGGER_GFXTIM_2

GPDMA1 HW Trigger signal is GFXTIM_2

LL_GPDMA1_TRIGGER_GFXTIM_1

GPDMA1 HW Trigger signal is GFXTIM_1

LL_GPDMA1_TRIGGER_GFXTIM_0

GPDMA1 HW Trigger signal is GFXTIM_0

LL_GPDMA1_TRIGGER_LPTIM1_CH1

GPDMA1 HW Trigger signal is LPTIM1_CH1

LL_GPDMA1_TRIGGER_LPTIM1_CH2

GPDMA1 HW Trigger signal is LPTIM1_CH2

LL_GPDMA1_TRIGGER_LPTIM2_CH1

GPDMA1 HW Trigger signal is LPTIM2_CH1

LL_GPDMA1_TRIGGER_LPTIM2_CH2

GPDMA1 HW Trigger signal is LPTIM2_CH2

LL_GPDMA1_TRIGGER_LPTIM3_CH1

GPDMA1 HW Trigger signal is LPTIM3_CH1

LL_GPDMA1_TRIGGER_LPTIM3_CH2

GPDMA1 HW Trigger signal is LPTIM3_CH2

LL_GPDMA1_TRIGGER_LPTIM4_OUT

GPDMA1 HW Trigger signal is LPTIM4_OUT

LL_GPDMA1_TRIGGER_LPTIM5_OUT

GPDMA1 HW Trigger signal is LPTIM5_OUT

LL_GPDMA1_TRIGGER_RTC_WKUP

GPDMA1 HW Trigger signal is RTC_WKUP

LL_GPDMA1_TRIGGER_LPUART1_IT_R_WUP_ASYNC

GPDMA1 HW Trigger signal is IT_R_WUP_ASYNC

LL_GPDMA1_TRIGGER_LPUART1_IT_T_WUP_ASYNC

GPDMA1 HW Trigger signal is IT_T_WUP_ASYNC

LL_GPDMA1_TRIGGER_SPI6_IT_OR_SPI6_AIT_SYNC

GPDMA1 HW Trigger signal is SPI6_IT_OR_SPI6_AIT_SYNC

LL_GPDMA1_TRIGGER_TIM1_TRGO

GPDMA1 HW Trigger signal is TIM1_TRGO

LL_GPDMA1_TRIGGER_TIM1_TRGO2

GPDMA1 HW Trigger signal is TIM1_TRGO2

LL_GPDMA1_TRIGGER_TIM2_TRGO

GPDMA1 HW Trigger signal is TIM2_TRGO

LL_GPDMA1_TRIGGER_TIM3_TRGO

GPDMA1 HW Trigger signal is TIM3_TRGO

LL_GPDMA1_TRIGGER_TIM4_TRGO

GPDMA1 HW Trigger signal is TIM4_TRGO

LL_GPDMA1_TRIGGER_TIM5_TRGO

GPDMA1 HW Trigger signal is TIM5_TRGO

LL_GPDMA1_TRIGGER_TIM6_TRGO

GPDMA1 HW Trigger signal is TIM6_TRGO

LL_GPDMA1_TRIGGER_TIM7_TRGO

GPDMA1 HW Trigger signal is TIM7_TRGO

LL_GPDMA1_TRIGGER_TIM8_TRGO

GPDMA1 HW Trigger signal is TIM8_TRGO

LL_GPDMA1_TRIGGER_TIM8_TRGO2

GPDMA1 HW Trigger signal is TIM8_TRGO2

LL_GPDMA1_TRIGGER_TIM12_TRGO

GPDMA1 HW Trigger signal is TIM12_TRGO

LL_GPDMA1_TRIGGER_TIM15_TRGO

GPDMA1 HW Trigger signal is TIM15_TRGO

LL_GPDMA1_TRIGGER_HPDMA1_CH0_TCF

GPDMA1 HW Trigger signal is HPDMA1_CH0_TCF

LL_GPDMA1_TRIGGER_HPDMA1_CH1_TCF

GPDMA1 HW Trigger signal is HPDMA1_CH1_TCF

LL_GPDMA1_TRIGGER_HPDMA1_CH2_TCF

GPDMA1 HW Trigger signal is HPDMA1_CH2_TCF

LL_GPDMA1_TRIGGER_HPDMA1_CH3_TCF

GPDMA1 HW Trigger signal is HPDMA1_CH3_TCF

LL_GPDMA1_TRIGGER_HPDMA1_CH4_TCF

GPDMA1 HW Trigger signal is HPDMA1_CH4_TCF

LL_GPDMA1_TRIGGER_HPDMA1_CH5_TCF

GPDMA1 HW Trigger signal is HPDMA1_CH5_TCF

LL_GPDMA1_TRIGGER_HPDMA1_CH6_TCF

GPDMA1 HW Trigger signal is HPDMA1_CH6_TCF

LL_GPDMA1_TRIGGER_HPDMA1_CH7_TCF

GPDMA1 HW Trigger signal is HPDMA1_CH7_TCF

LL_GPDMA1_TRIGGER_HPDMA1_CH8_TCF

GPDMA1 HW Trigger signal is HPDMA1_CH8_TCF

LL_GPDMA1_TRIGGER_HPDMA1_CH9_TCF

GPDMA1 HW Trigger signal is HPDMA1_CH9_TCF

LL_GPDMA1_TRIGGER_HPDMA1_CH10_TCF

GPDMA1 HW Trigger signal is HPDMA1_CH10_TCF

LL_GPDMA1_TRIGGER_HPDMA1_CH11_TCF

GPDMA1 HW Trigger signal is HPDMA1_CH11_TCF

LL_GPDMA1_TRIGGER_HPDMA1_CH12_TCF

GPDMA1 HW Trigger signal is HPDMA1_CH12_TCF

LL_GPDMA1_TRIGGER_HPDMA1_CH13_TCF

GPDMA1 HW Trigger signal is HPDMA1_CH13_TCF

LL_GPDMA1_TRIGGER_HPDMA1_CH14_TCF

GPDMA1 HW Trigger signal is HPDMA1_CH14_TCF

LL_GPDMA1_TRIGGER_HPDMA1_CH15_TCF

GPDMA1 HW Trigger signal is HPDMA1_CH15_TCF

LL_GPDMA1_TRIGGER_GPDMA1_CH0_TCF

GPDMA1 HW Trigger signal is GPDMA1_CH0_TCF

LL_GPDMA1_TRIGGER_GPDMA1_CH1_TCF

GPDMA1 HW Trigger signal is GPDMA1_CH1_TCF

LL_GPDMA1_TRIGGER_GPDMA1_CH2_TCF

GPDMA1 HW Trigger signal is GPDMA1_CH2_TCF

LL_GPDMA1_TRIGGER_GPDMA1_CH3_TCF

GPDMA1 HW Trigger signal is GPDMA1_CH3_TCF

LL_GPDMA1_TRIGGER_GPDMA1_CH4_TCF

GPDMA1 HW Trigger signal is GPDMA1_CH4_TCF

LL_GPDMA1_TRIGGER_GPDMA1_CH5_TCF

GPDMA1 HW Trigger signal is GPDMA1_CH5_TCF

LL_GPDMA1_TRIGGER_GPDMA1_CH6_TCF

GPDMA1 HW Trigger signal is GPDMA1_CH6_TCF

LL_GPDMA1_TRIGGER_GPDMA1_CH7_TCF

GPDMA1 HW Trigger signal is GPDMA1_CH7_TCF

LL_GPDMA1_TRIGGER_GPDMA1_CH8_TCF

GPDMA1 HW Trigger signal is GPDMA1_CH8_TCF

LL_GPDMA1_TRIGGER_GPDMA1_CH9_TCF

GPDMA1 HW Trigger signal is GPDMA1_CH9_TCF

LL_GPDMA1_TRIGGER_GPDMA1_CH10_TCF

GPDMA1 HW Trigger signal is GPDMA1_CH10_TCF

LL_GPDMA1_TRIGGER_GPDMA1_CH11_TCF

GPDMA1 HW Trigger signal is GPDMA1_CH11_TCF

LL_GPDMA1_TRIGGER_GPDMA1_CH12_TCF

GPDMA1 HW Trigger signal is GPDMA1_CH12_TCF

LL_GPDMA1_TRIGGER_GPDMA1_CH13_TCF

GPDMA1 HW Trigger signal is GPDMA1_CH13_TCF

LL_GPDMA1_TRIGGER_GPDMA1_CH14_TCF

GPDMA1 HW Trigger signal is GPDMA1_CH14_TCF

LL_GPDMA1_TRIGGER_GPDMA1_CH15_TCF

GPDMA1 HW Trigger signal is GPDMA1_CH15_TCF

LL_GPDMA1_TRIGGER_EXTIT0_SYNC

GPDMA1 HW Trigger signal is EXTIT0_SYNC

LL_GPDMA1_TRIGGER_EXTIT1_SYNC

GPDMA1 HW Trigger signal is EXTIT1_SYNC

LL_GPDMA1_TRIGGER_EXTIT2_SYNC

GPDMA1 HW Trigger signal is EXTIT2_SYNC

LL_GPDMA1_TRIGGER_EXTIT3_SYNC

GPDMA1 HW Trigger signal is EXTIT3_SYNC

LL_GPDMA1_TRIGGER_EXTIT4_SYNC

GPDMA1 HW Trigger signal is EXTIT4_SYNC

LL_GPDMA1_TRIGGER_EXTIT5_SYNC

GPDMA1 HW Trigger signal is EXTIT5_SYNC

LL_GPDMA1_TRIGGER_EXTIT6_SYNC

GPDMA1 HW Trigger signal is EXTIT6_SYNC

LL_GPDMA1_TRIGGER_EXTIT7_SYNC

GPDMA1 HW Trigger signal is EXTIT7_SYNC

LL_GPDMA1_TRIGGER_EXTIT8_SYNC

GPDMA1 HW Trigger signal is EXTIT8_SYNC

LL_GPDMA1_TRIGGER_EXTIT9_SYNC

GPDMA1 HW Trigger signal is EXTIT9_SYNC

LL_GPDMA1_TRIGGER_EXTIT10_SYNC

GPDMA1 HW Trigger signal is EXTIT10_SYNC

LL_GPDMA1_TRIGGER_EXTIT11_SYNC

GPDMA1 HW Trigger signal is EXTIT11_SYNC

LL_GPDMA1_TRIGGER_EXTIT12_SYNC

GPDMA1 HW Trigger signal is EXTIT12_SYNC

LL_GPDMA1_TRIGGER_EXTIT13_SYNC

GPDMA1 HW Trigger signal is EXTIT13_SYNC

LL_GPDMA1_TRIGGER_EXTIT14_SYNC

GPDMA1 HW Trigger signal is EXTIT14_SYNC

LL_GPDMA1_TRIGGER_EXTIT15_SYNC

GPDMA1 HW Trigger signal is EXTIT15_SYNC

Common Write and Read Registers macros

LL_DMA_WriteReg

Description:

- Write a value in DMA register.

Parameters:

- __INSTANCE__: DMA Instance.
- __REG__: Register to be written.
- __VALUE__: Value to be written in the register.

Return value:

- None.

LL_DMA_ReadReg

Description:

- Read a value in DMA register.

Parameters:

- __INSTANCE__: DMA Instance.
- __REG__: Register to be read.

Return value:

- Register: value.

Convert DMAxChannely

LL_DMA_GET_INSTANCE

Description:

- Convert DMAx_Channely into DMAx.

Parameters:

- __CHANNEL_INSTANCE__: DMAx_Channely.

Return value:

- DMAx.

LL_DMA_GET_CHANNEL

Description:

- Convert DMAx_Channely into LL_DMA_CHANNEL_y.

Parameters:

- __CHANNEL_INSTANCE__: DMAx_Channely.

Return value:

- LL_DMA_CHANNEL_y.

LL_DMA_GET_CHANNEL_INSTANCE

Description:

- Convert DMA Instance DMAx and LL_DMA_CHANNEL_y into DMAx_Channely.

Parameters:

- __DMA_INSTANCE__: DMAx.
- __CHANNEL__: LL_DMA_CHANNEL_y.

Return value:

- DMAx_Channely.

Transfer Mode

LL_DMA_NORMAL

Normal DMA transfer

LL_DMA_PFCTRL

HW request peripheral flow control mode

92 LL EXTI Generic Driver

92.1 EXTI Firmware driver registers structures

92.1.1 LL_EXTI_InitTypeDef

LL_EXTI_InitTypeDef is defined in the stm32n6xx_ll_exti.h

Data Fields

- ***uint32_t Line_0_31***
- ***uint32_t Line_32_63***
- ***uint32_t Line_64_95***
- ***FunctionalState LineCommand***
- ***uint8_t Mode***
- ***uint8_t Trigger***

Field Documentation

- ***uint32_t LL_EXTI_InitTypeDef::Line_0_31***
  Specifies the EXTI lines to be enabled or disabled for Lines in range 0 to 31 This parameter can be any combination of ***EXTI_LL_EC_LINE***
- ***uint32_t LL_EXTI_InitTypeDef::Line_32_63***
  Specifies the EXTI lines to be enabled or disabled for Lines in range 32 to 63 This parameter can be any combination of ***EXTI_LL_EC_LINE***
- ***uint32_t LL_EXTI_InitTypeDef::Line_64_95***
  Specifies the EXTI lines to be enabled or disabled for Lines in range 64 to 95 This parameter can be any combination of ***EXTI_LL_EC_LINE***
- ***FunctionalState LL_EXTI_InitTypeDef::LineCommand***
  Specifies the new state of the selected EXTI lines. This parameter can be set either to ENABLE or DISABLE
- ***uint8_t LL_EXTI_InitTypeDef::Mode***
  Specifies the mode for the EXTI lines. This parameter can be a value of ***EXTI_LL_EC_MODE***.
- ***uint8_t LL_EXTI_InitTypeDef::Trigger***
  Specifies the trigger signal active edge for the EXTI lines. This parameter can be a value of ***EXTI_LL_EC_TRIGGER***.

92.2 EXTI Firmware driver API description

The following section lists the various functions of the EXTI library.

92.2.1 Detailed description of functions

LL_EXTI_EnableIT_0_31

Function name

__STATIC_INLINE void LL_EXTI_EnableIT_0_31 (uint32_t ExtiLine)

Function description

Enable ExtiLine Interrupt request for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_17
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21
  - LL_EXTI_LINE_22
  - LL_EXTI_LINE_23
  - LL_EXTI_LINE_24
  - LL_EXTI_LINE_25
  - LL_EXTI_LINE_26
  - LL_EXTI_LINE_27
  - LL_EXTI_LINE_28
  - LL_EXTI_LINE_29
  - LL_EXTI_LINE_30
  - LL_EXTI_LINE_31
  - LL_EXTI_LINE_ALL_0_31

Return values

- **None:**

Notes

- The reset value for the direct or internal lines (see RM) is set to 1 in order to enable the interrupt by default. Bits are set automatically at Power on.
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- IMR1 IMx LL_EXTI_EnableIT_0_31

LL_EXTI_EnableIT_32_63

Function name

__STATIC_INLINE void LL_EXTI_EnableIT_32_63 (uint32_t ExtiLine)

Function description

Enable ExtiLine Interrupt request for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_32
  - LL_EXTI_LINE_33
  - LL_EXTI_LINE_34
  - LL_EXTI_LINE_35
  - LL_EXTI_LINE_36
  - LL_EXTI_LINE_37
  - LL_EXTI_LINE_38
  - LL_EXTI_LINE_39
  - LL_EXTI_LINE_40
  - LL_EXTI_LINE_41
  - LL_EXTI_LINE_42
  - LL_EXTI_LINE_43
  - LL_EXTI_LINE_44
  - LL_EXTI_LINE_45
  - LL_EXTI_LINE_46
  - LL_EXTI_LINE_47
  - LL_EXTI_LINE_48
  - LL_EXTI_LINE_49
  - LL_EXTI_LINE_50
  - LL_EXTI_LINE_51
  - LL_EXTI_LINE_52
  - LL_EXTI_LINE_53
  - LL_EXTI_LINE_54
  - LL_EXTI_LINE_55
  - LL_EXTI_LINE_56
  - LL_EXTI_LINE_57
  - LL_EXTI_LINE_58
  - LL_EXTI_LINE_62
  - LL_EXTI_LINE_61
  - LL_EXTI_LINE_62
  - LL_EXTI_LINE_63
  - LL_EXTI_LINE_ALL_32_63

Return values

- **None:**

Notes

- The reset value for the direct lines (see RM) is set to 1 in order to enable the interrupt by default. Bits are set automatically at Power on.

Reference Manual to LL API cross reference:

- IMR2 IMx LL_EXTI_EnableIT_32_63

LL_EXTI_EnableIT_64_95

Function name

__STATIC_INLINE void LL_EXTI_EnableIT_64_95 (uint32_t ExtiLine)

Function description

Enable ExtiLine Interrupt request for Lines in range 64 to 95.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_64
  - LL_EXTI_LINE_65
  - LL_EXTI_LINE_66
  - LL_EXTI_LINE_68
  - LL_EXTI_LINE_69
  - LL_EXTI_LINE_70
  - LL_EXTI_LINE_71
  - LL_EXTI_LINE_72
  - LL_EXTI_LINE_73
  - LL_EXTI_LINE_74
  - LL_EXTI_LINE_77
  - LL_EXTI_LINE_ALL_64_95

Return values

- **None:**

Notes

- The reset value for the direct lines (see RM) is set to 1 in order to enable the interrupt by default. Bits are set automatically at Power on.

Reference Manual to LL API cross reference:

- IMR3 IMx LL_EXTI_EnableIT_64_95

LL_EXTI_DisableIT_0_31

Function name

__STATIC_INLINE void LL_EXTI_DisableIT_0_31 (uint32_t ExtiLine)

Function description

Disable ExtiLine Interrupt request for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_17
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21
  - LL_EXTI_LINE_22
  - LL_EXTI_LINE_23
  - LL_EXTI_LINE_24
  - LL_EXTI_LINE_25
  - LL_EXTI_LINE_26
  - LL_EXTI_LINE_27
  - LL_EXTI_LINE_28
  - LL_EXTI_LINE_29
  - LL_EXTI_LINE_30
  - LL_EXTI_LINE_31
  - LL_EXTI_LINE_ALL_0_31

Return values

- **None:**

Notes

- The reset value for the direct or internal lines (see RM) is set to 1 in order to enable the interrupt by default. Bits are set automatically at Power on.
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- IMR1 IMx LL_EXTI_DisableIT_0_31

LL_EXTI_DisableIT_32_63

Function name

__STATIC_INLINE void LL_EXTI_DisableIT_32_63 (uint32_t ExtiLine)

Function description

Disable ExtiLine Interrupt request for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_32
  - LL_EXTI_LINE_33
  - LL_EXTI_LINE_34
  - LL_EXTI_LINE_35
  - LL_EXTI_LINE_36
  - LL_EXTI_LINE_37
  - LL_EXTI_LINE_38
  - LL_EXTI_LINE_39
  - LL_EXTI_LINE_40
  - LL_EXTI_LINE_41
  - LL_EXTI_LINE_42
  - LL_EXTI_LINE_43
  - LL_EXTI_LINE_44
  - LL_EXTI_LINE_45
  - LL_EXTI_LINE_46
  - LL_EXTI_LINE_47
  - LL_EXTI_LINE_48
  - LL_EXTI_LINE_49
  - LL_EXTI_LINE_50
  - LL_EXTI_LINE_51
  - LL_EXTI_LINE_52
  - LL_EXTI_LINE_53
  - LL_EXTI_LINE_54
  - LL_EXTI_LINE_55
  - LL_EXTI_LINE_56
  - LL_EXTI_LINE_57
  - LL_EXTI_LINE_58
  - LL_EXTI_LINE_62
  - LL_EXTI_LINE_61
  - LL_EXTI_LINE_62
  - LL_EXTI_LINE_63
  - LL_EXTI_LINE_ALL_32_63

Return values

- **None:**

Notes

- The reset value for the direct lines (see RM) is set to 1 in order to enable the interrupt by default. Bits are set automatically at Power on.

Reference Manual to LL API cross reference:

- IMR2 IMx LL_EXTI_DisableIT_32_63

LL_EXTI_DisableIT_64_95

Function name

__STATIC_INLINE void LL_EXTI_DisableIT_64_95 (uint32_t ExtiLine)

Function description

Disable ExtiLine Interrupt request for Lines in range 64 to 95.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_64
  - LL_EXTI_LINE_65
  - LL_EXTI_LINE_66
  - LL_EXTI_LINE_68
  - LL_EXTI_LINE_69
  - LL_EXTI_LINE_70
  - LL_EXTI_LINE_71
  - LL_EXTI_LINE_72
  - LL_EXTI_LINE_73
  - LL_EXTI_LINE_74
  - LL_EXTI_LINE_77
  - LL_EXTI_LINE_ALL_64_95

Return values

- **None:**

Notes

- The reset value for the direct lines (see RM) is set to 1 in order to enable the interrupt by default. Bits are set automatically at Power on.

Reference Manual to LL API cross reference:

- IMR3 IMx LL_EXTI_DisableIT_64_95

LL_EXTI_IsEnabledIT_0_31

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsEnabledIT_0_31 (uint32_t ExtiLine)

Function description

Indicate if ExtiLine Interrupt request is enabled for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_17
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21
  - LL_EXTI_LINE_22
  - LL_EXTI_LINE_23
  - LL_EXTI_LINE_24
  - LL_EXTI_LINE_25
  - LL_EXTI_LINE_26
  - LL_EXTI_LINE_27
  - LL_EXTI_LINE_28
  - LL_EXTI_LINE_29
  - LL_EXTI_LINE_30
  - LL_EXTI_LINE_31
  - LL_EXTI_LINE_ALL_0_31

Return values

- **State:** of bit (1 or 0).

Notes

- The reset value for the direct or internal lines (see RM) is set to 1 in order to enable the interrupt by default. Bits are set automatically at Power on.
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- IMR1 IMx LL_EXTI_IsEnabledIT_0_31

LL_EXTI_IsEnabledIT_32_63

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsEnabledIT_32_63 (uint32_t ExtiLine)

Function description

Indicate if ExtiLine Interrupt request is enabled for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_32
  - LL_EXTI_LINE_33
  - LL_EXTI_LINE_34
  - LL_EXTI_LINE_35
  - LL_EXTI_LINE_36
  - LL_EXTI_LINE_37
  - LL_EXTI_LINE_38
  - LL_EXTI_LINE_39
  - LL_EXTI_LINE_40
  - LL_EXTI_LINE_41
  - LL_EXTI_LINE_42
  - LL_EXTI_LINE_43
  - LL_EXTI_LINE_44
  - LL_EXTI_LINE_45
  - LL_EXTI_LINE_46
  - LL_EXTI_LINE_47
  - LL_EXTI_LINE_48
  - LL_EXTI_LINE_49
  - LL_EXTI_LINE_50
  - LL_EXTI_LINE_51
  - LL_EXTI_LINE_52
  - LL_EXTI_LINE_53
  - LL_EXTI_LINE_54
  - LL_EXTI_LINE_55
  - LL_EXTI_LINE_56
  - LL_EXTI_LINE_57
  - LL_EXTI_LINE_58
  - LL_EXTI_LINE_62
  - LL_EXTI_LINE_61
  - LL_EXTI_LINE_62
  - LL_EXTI_LINE_63
  - LL_EXTI_LINE_ALL_32_63

Return values

- **State:** of bit (1 or 0).

Notes

- The reset value for the direct lines (see RM) is set to 1 in order to enable the interrupt by default. Bits are set automatically at Power on.

Reference Manual to LL API cross reference:

- IMR2 IMx LL_EXTI_IsEnabledIT_32_63

LL_EXTI_IsEnabledIT_64_95

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsEnabledIT_64_95 (uint32_t ExtiLine)

Function description

Indicate if ExtiLine Interrupt request is enabled for Lines in range 64 to 95.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_64
  - LL_EXTI_LINE_65
  - LL_EXTI_LINE_66
  - LL_EXTI_LINE_68
  - LL_EXTI_LINE_69
  - LL_EXTI_LINE_70
  - LL_EXTI_LINE_71
  - LL_EXTI_LINE_72
  - LL_EXTI_LINE_73
  - LL_EXTI_LINE_74
  - LL_EXTI_LINE_77
  - LL_EXTI_LINE_ALL_64_95

Return values

- **State:** of bit (1 or 0).

Notes

- The reset value for the direct lines (see RM) is set to 1 in order to enable the interrupt by default. Bits are set automatically at Power on.

Reference Manual to LL API cross reference:

- IMR3 IMx LL_EXTI_IsEnabledIT_64_95

LL_EXTI_EnableEvent_0_31

Function name

__STATIC_INLINE void LL_EXTI_EnableEvent_0_31 (uint32_t ExtiLine)

Function description

Enable ExtiLine Event request for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_17
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21
  - LL_EXTI_LINE_22
  - LL_EXTI_LINE_23
  - LL_EXTI_LINE_24
  - LL_EXTI_LINE_25
  - LL_EXTI_LINE_26
  - LL_EXTI_LINE_27
  - LL_EXTI_LINE_28
  - LL_EXTI_LINE_29
  - LL_EXTI_LINE_30
  - LL_EXTI_LINE_31
  - LL_EXTI_LINE_ALL_0_31

Return values

- **None:**

Notes

- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- EMR1 EMx LL_EXTI_EnableEvent_0_31

LL_EXTI_EnableEvent_32_63

Function name

__STATIC_INLINE void LL_EXTI_EnableEvent_32_63 (uint32_t ExtiLine)

Function description

Enable ExtiLine Event request for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_32
  - LL_EXTI_LINE_33
  - LL_EXTI_LINE_34
  - LL_EXTI_LINE_35
  - LL_EXTI_LINE_36
  - LL_EXTI_LINE_37
  - LL_EXTI_LINE_38
  - LL_EXTI_LINE_39
  - LL_EXTI_LINE_40
  - LL_EXTI_LINE_41
  - LL_EXTI_LINE_42
  - LL_EXTI_LINE_43
  - LL_EXTI_LINE_44
  - LL_EXTI_LINE_45
  - LL_EXTI_LINE_46
  - LL_EXTI_LINE_47
  - LL_EXTI_LINE_48
  - LL_EXTI_LINE_49
  - LL_EXTI_LINE_50
  - LL_EXTI_LINE_51
  - LL_EXTI_LINE_52
  - LL_EXTI_LINE_53
  - LL_EXTI_LINE_54
  - LL_EXTI_LINE_55
  - LL_EXTI_LINE_56
  - LL_EXTI_LINE_57
  - LL_EXTI_LINE_58
  - LL_EXTI_LINE_62
  - LL_EXTI_LINE_61
  - LL_EXTI_LINE_62
  - LL_EXTI_LINE_63
  - LL_EXTI_LINE_ALL_32_63

Return values

- **None:**

Reference Manual to LL API cross reference:

- EMR2 EMx LL_EXTI_EnableEvent_32_63

LL_EXTI_EnableEvent_64_95

Function name

__STATIC_INLINE void LL_EXTI_EnableEvent_64_95 (uint32_t ExtiLine)

Function description

Enable ExtiLine Event request for Lines in range 64 to 95.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_64
  - LL_EXTI_LINE_65
  - LL_EXTI_LINE_66
  - LL_EXTI_LINE_68
  - LL_EXTI_LINE_69
  - LL_EXTI_LINE_70
  - LL_EXTI_LINE_71
  - LL_EXTI_LINE_72
  - LL_EXTI_LINE_73
  - LL_EXTI_LINE_74
  - LL_EXTI_LINE_77
  - LL_EXTI_LINE_ALL_64_95

Return values

- **None:**

Reference Manual to LL API cross reference:

- EMR3 EMx LL_EXTI_EnableEvent_64_95

LL_EXTI_DisableEvent_0_31

Function name

__STATIC_INLINE void LL_EXTI_DisableEvent_0_31 (uint32_t ExtiLine)

Function description

Disable ExtiLine Event request for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_17
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21
  - LL_EXTI_LINE_22
  - LL_EXTI_LINE_23
  - LL_EXTI_LINE_24
  - LL_EXTI_LINE_25
  - LL_EXTI_LINE_26
  - LL_EXTI_LINE_27
  - LL_EXTI_LINE_28
  - LL_EXTI_LINE_29
  - LL_EXTI_LINE_30
  - LL_EXTI_LINE_31
  - LL_EXTI_LINE_ALL_0_31

Return values

- **None:**

Notes

- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- EMR1 EMx LL_EXTI_DisableEvent_0_31

LL_EXTI_DisableEvent_32_63

Function name

__STATIC_INLINE void LL_EXTI_DisableEvent_32_63 (uint32_t ExtiLine)

Function description

Disable ExtiLine Event request for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_32
  - LL_EXTI_LINE_33
  - LL_EXTI_LINE_34
  - LL_EXTI_LINE_35
  - LL_EXTI_LINE_36
  - LL_EXTI_LINE_37
  - LL_EXTI_LINE_38
  - LL_EXTI_LINE_39
  - LL_EXTI_LINE_40
  - LL_EXTI_LINE_41
  - LL_EXTI_LINE_42
  - LL_EXTI_LINE_43
  - LL_EXTI_LINE_44
  - LL_EXTI_LINE_45
  - LL_EXTI_LINE_46
  - LL_EXTI_LINE_47
  - LL_EXTI_LINE_48
  - LL_EXTI_LINE_49
  - LL_EXTI_LINE_50
  - LL_EXTI_LINE_51
  - LL_EXTI_LINE_52
  - LL_EXTI_LINE_53
  - LL_EXTI_LINE_54
  - LL_EXTI_LINE_55
  - LL_EXTI_LINE_56
  - LL_EXTI_LINE_57
  - LL_EXTI_LINE_58
  - LL_EXTI_LINE_62
  - LL_EXTI_LINE_61
  - LL_EXTI_LINE_62
  - LL_EXTI_LINE_63
  - LL_EXTI_LINE_ALL_32_63

Return values

- **None:**

Reference Manual to LL API cross reference:

- EMR2 EMx LL_EXTI_DisableEvent_32_63

LL_EXTI_DisableEvent_64_95

Function name

__STATIC_INLINE void LL_EXTI_DisableEvent_64_95 (uint32_t ExtiLine)

Function description

Disable ExtiLine Event request for Lines in range 64 to 95.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_64
  - LL_EXTI_LINE_65
  - LL_EXTI_LINE_66
  - LL_EXTI_LINE_68
  - LL_EXTI_LINE_69
  - LL_EXTI_LINE_70
  - LL_EXTI_LINE_71
  - LL_EXTI_LINE_72
  - LL_EXTI_LINE_73
  - LL_EXTI_LINE_74
  - LL_EXTI_LINE_77
  - LL_EXTI_LINE_ALL_64_95

Return values

- **None:**

Reference Manual to LL API cross reference:

- EMR3 EMx LL_EXTI_DisableEvent_64_95

LL_EXTI_IsEnabledEvent_0_31

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsEnabledEvent_0_31 (uint32_t ExtiLine)

Function description

Indicate if ExtiLine Event request is enabled for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_17
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21
  - LL_EXTI_LINE_22
  - LL_EXTI_LINE_23
  - LL_EXTI_LINE_24
  - LL_EXTI_LINE_25
  - LL_EXTI_LINE_26
  - LL_EXTI_LINE_27
  - LL_EXTI_LINE_28
  - LL_EXTI_LINE_29
  - LL_EXTI_LINE_30
  - LL_EXTI_LINE_31
  - LL_EXTI_LINE_ALL_0_31

Return values

- **State:** of bit (1 or 0).

Notes

- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- EMR1 EMx LL_EXTI_IsEnabledEvent_0_31

LL_EXTI_IsEnabledEvent_32_63

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsEnabledEvent_32_63 (uint32_t ExtiLine)

Function description

Indicate if ExtiLine Event request is enabled for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_32
  - LL_EXTI_LINE_33
  - LL_EXTI_LINE_34
  - LL_EXTI_LINE_35
  - LL_EXTI_LINE_36
  - LL_EXTI_LINE_37
  - LL_EXTI_LINE_38
  - LL_EXTI_LINE_39
  - LL_EXTI_LINE_40
  - LL_EXTI_LINE_41
  - LL_EXTI_LINE_42
  - LL_EXTI_LINE_43
  - LL_EXTI_LINE_44
  - LL_EXTI_LINE_45
  - LL_EXTI_LINE_46
  - LL_EXTI_LINE_47
  - LL_EXTI_LINE_48
  - LL_EXTI_LINE_49
  - LL_EXTI_LINE_50
  - LL_EXTI_LINE_51
  - LL_EXTI_LINE_52
  - LL_EXTI_LINE_53
  - LL_EXTI_LINE_54
  - LL_EXTI_LINE_55
  - LL_EXTI_LINE_56
  - LL_EXTI_LINE_57
  - LL_EXTI_LINE_58
  - LL_EXTI_LINE_62
  - LL_EXTI_LINE_61
  - LL_EXTI_LINE_62
  - LL_EXTI_LINE_63
  - LL_EXTI_LINE_ALL_32_63

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- EMR2 EMx LL_EXTI_IsEnabledEvent_32_63

LL_EXTI_IsEnabledEvent_64_95

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsEnabledEvent_64_95 (uint32_t ExtiLine)

Function description

Indicate if ExtiLine Event request is enabled for Lines in range 64 to 95.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_64
  - LL_EXTI_LINE_65
  - LL_EXTI_LINE_66
  - LL_EXTI_LINE_68
  - LL_EXTI_LINE_69
  - LL_EXTI_LINE_70
  - LL_EXTI_LINE_71
  - LL_EXTI_LINE_72
  - LL_EXTI_LINE_73
  - LL_EXTI_LINE_74
  - LL_EXTI_LINE_77
  - LL_EXTI_LINE_ALL_64_95

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- EMR3 EMx LL_EXTI_IsEnabledEvent_64_95

LL_EXTI_EnableRisingTrig_0_31

Function name

__STATIC_INLINE void LL_EXTI_EnableRisingTrig_0_31 (uint32_t ExtiLine)

Function description

Enable ExtiLine Rising Edge Trigger for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21

Return values

- **None:**

Notes

- The configurable wakeup lines are edge-triggered. No glitch must be generated on these lines. If a rising edge on a configurable interrupt line occurs during a write operation in the EXTI_RTSR register, the pending bit is not set. Rising and falling edge triggers can be set for the same interrupt line. In this case, both generate a trigger condition.
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- RTSR1 RTx LL_EXTI_EnableRisingTrig_0_31

LL_EXTI_EnableRisingTrig_32_63

Function name

__STATIC_INLINE void LL_EXTI_EnableRisingTrig_32_63 (uint32_t ExtiLine)

Function description

Enable ExtiLine Rising Edge Trigger for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_39
  - LL_EXTI_LINE_40
  - LL_EXTI_LINE_51
  - LL_EXTI_LINE_54
  - LL_EXTI_LINE_56

Return values

- **None:**

Notes

- The configurable wakeup lines are edge-triggered. No glitch must be generated on these lines. If a rising edge on a configurable interrupt line occurs during a write operation in the EXTI_RTSR register, the pending bit is not set.Rising and falling edge triggers can be set for the same interrupt line. In this case, both generate a trigger condition.

Reference Manual to LL API cross reference:

- RTSR2 RTx LL_EXTI_EnableRisingTrig_32_63

LL_EXTI_EnableRisingTrig_64_95

Function name

__STATIC_INLINE void LL_EXTI_EnableRisingTrig_64_95 (uint32_t ExtiLine)

Function description

Enable ExtiLine Rising Edge Trigger for Lines in range 64 to 95.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_66
  - LL_EXTI_LINE_68
  - LL_EXTI_LINE_69
  - LL_EXTI_LINE_70
  - LL_EXTI_LINE_71
  - LL_EXTI_LINE_72
  - LL_EXTI_LINE_73
  - LL_EXTI_LINE_74

Return values

- **None:**

Notes

- The configurable wakeup lines are edge-triggered. No glitch must be generated on these lines. If a rising edge on a configurable interrupt line occurs during a write operation in the EXTI_RTSR register, the pending bit is not set.Rising and falling edge triggers can be set for the same interrupt line. In this case, both generate a trigger condition.

Reference Manual to LL API cross reference:

- RTSR3 RTx LL_EXTI_EnableRisingTrig_64_95

LL_EXTI_DisableRisingTrig_0_31

Function name

__STATIC_INLINE void LL_EXTI_DisableRisingTrig_0_31 (uint32_t ExtiLine)

Function description

Disable ExtiLine Rising Edge Trigger for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21

Return values

- **None:**

Notes

- The configurable wakeup lines are edge-triggered. No glitch must be generated on these lines. If a rising edge on a configurable interrupt line occurs during a write operation in the EXTI_RTSR register, the pending bit is not set. Rising and falling edge triggers can be set for the same interrupt line. In this case, both generate a trigger condition.
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- RTSR1 RTx LL_EXTI_DisableRisingTrig_0_31

LL_EXTI_DisableRisingTrig_32_63

Function name

__STATIC_INLINE void LL_EXTI_DisableRisingTrig_32_63 (uint32_t ExtiLine)

Function description

Disable ExtiLine Rising Edge Trigger for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_39
  - LL_EXTI_LINE_40
  - LL_EXTI_LINE_51
  - LL_EXTI_LINE_54
  - LL_EXTI_LINE_56

Return values

- **None:**

Notes

- The configurable wakeup lines are edge-triggered. No glitch must be generated on these lines. If a rising edge on a configurable interrupt line occurs during a write operation in the EXTI_RTSR register, the pending bit is not set. Rising and falling edge triggers can be set for the same interrupt line. In this case, both generate a trigger condition.

Reference Manual to LL API cross reference:

- RTSR2 RTx LL_EXTI_DisableRisingTrig_32_63

LL_EXTI_DisableRisingTrig_64_95

Function name

__STATIC_INLINE void LL_EXTI_DisableRisingTrig_64_95 (uint32_t ExtiLine)

Function description

Disable ExtiLine Rising Edge Trigger for Lines in range 64 to 95.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_66
  - LL_EXTI_LINE_68
  - LL_EXTI_LINE_69
  - LL_EXTI_LINE_70
  - LL_EXTI_LINE_71
  - LL_EXTI_LINE_72
  - LL_EXTI_LINE_73
  - LL_EXTI_LINE_74

Return values

- **None:**

Notes

- The configurable wakeup lines are edge-triggered. No glitch must be generated on these lines. If a rising edge on a configurable interrupt line occurs during a write operation in the EXTI_RTSR register, the pending bit is not set. Rising and falling edge triggers can be set for the same interrupt line. In this case, both generate a trigger condition.

Reference Manual to LL API cross reference:

- RTSR3 RTx LL_EXTI_DisableRisingTrig_64_95

LL_EXTI_IsEnabledRisingTrig_0_31

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsEnabledRisingTrig_0_31 (uint32_t ExtiLine)

Function description

Check if rising edge trigger is enabled for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21

Return values

- **State:** of bit (1 or 0).

Notes

- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- RTSR1 RTx LL_EXTI_IsEnabledRisingTrig_0_31

LL_EXTI_IsEnabledRisingTrig_32_63

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsEnabledRisingTrig_32_63 (uint32_t ExtiLine)

Function description

Check if rising edge trigger is enabled for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_39
  - LL_EXTI_LINE_40
  - LL_EXTI_LINE_51
  - LL_EXTI_LINE_54
  - LL_EXTI_LINE_56

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTSR2 RTx LL_EXTI_IsEnabledRisingTrig_32_63

LL_EXTI_IsEnabledRisingTrig_64_95

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsEnabledRisingTrig_64_95 (uint32_t ExtiLine)

Function description

Check if rising edge trigger is enabled for Lines in range 64 to 95.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_66
  - LL_EXTI_LINE_68
  - LL_EXTI_LINE_69
  - LL_EXTI_LINE_70
  - LL_EXTI_LINE_71
  - LL_EXTI_LINE_72
  - LL_EXTI_LINE_73
  - LL_EXTI_LINE_74

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTSR3 RTx LL_EXTI_IsEnabledRisingTrig_64_95

LL_EXTI_EnableFallingTrig_0_31

Function name

__STATIC_INLINE void LL_EXTI_EnableFallingTrig_0_31 (uint32_t ExtiLine)

Function description

Enable ExtiLine Falling Edge Trigger for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21

Return values

- **None:**

Notes

- The configurable wakeup lines are edge-triggered. No glitch must be generated on these lines. If a falling edge on a configurable interrupt line occurs during a write operation in the EXTI_FTSR register, the pending bit is not set. Rising and falling edge triggers can be set for the same interrupt line. In this case, both generate a trigger condition.
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- FTSR1 FTx LL_EXTI_EnableFallingTrig_0_31

LL_EXTI_EnableFallingTrig_32_63

Function name

__STATIC_INLINE void LL_EXTI_EnableFallingTrig_32_63 (uint32_t ExtiLine)

Function description

Enable ExtiLine Falling Edge Trigger for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_39
  - LL_EXTI_LINE_40
  - LL_EXTI_LINE_51
  - LL_EXTI_LINE_54
  - LL_EXTI_LINE_56

Return values

- **None:**

Notes

- The configurable wakeup lines are edge-triggered. No glitch must be generated on these lines. If a Falling edge on a configurable interrupt line occurs during a write operation in the EXTI_FTSR register, the pending bit is not set. Rising and falling edge triggers can be set for the same interrupt line. In this case, both generate a trigger condition.

Reference Manual to LL API cross reference:

- FTSR2 FTx LL_EXTI_EnableFallingTrig_32_63

LL_EXTI_EnableFallingTrig_64_95

Function name

__STATIC_INLINE void LL_EXTI_EnableFallingTrig_64_95 (uint32_t ExtiLine)

Function description

Enable ExtiLine Falling Edge Trigger for Lines in range 64 to 95.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_66
  - LL_EXTI_LINE_68
  - LL_EXTI_LINE_69
  - LL_EXTI_LINE_70
  - LL_EXTI_LINE_71
  - LL_EXTI_LINE_72
  - LL_EXTI_LINE_73
  - LL_EXTI_LINE_74

Return values

- **None:**

Notes

- The configurable wakeup lines are edge-triggered. No glitch must be generated on these lines. If a Falling edge on a configurable interrupt line occurs during a write operation in the EXTI_FTSR register, the pending bit is not set. Rising and falling edge triggers can be set for the same interrupt line. In this case, both generate a trigger condition.

Reference Manual to LL API cross reference:

- FTSR3 FTx LL_EXTI_EnableFallingTrig_64_95

LL_EXTI_DisableFallingTrig_0_31

Function name

__STATIC_INLINE void LL_EXTI_DisableFallingTrig_0_31 (uint32_t ExtiLine)

Function description

Disable ExtiLine Falling Edge Trigger for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21

Return values

- **None:**

Notes

- The configurable wakeup lines are edge-triggered. No glitch must be generated on these lines. If a Falling edge on a configurable interrupt line occurs during a write operation in the EXTI_FTSR register, the pending bit is not set. Rising and falling edge triggers can be set for the same interrupt line. In this case, both generate a trigger condition.
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- FTSR1 FTx LL_EXTI_DisableFallingTrig_0_31

LL_EXTI_DisableFallingTrig_32_63

Function name

__STATIC_INLINE void LL_EXTI_DisableFallingTrig_32_63 (uint32_t ExtiLine)

Function description

Disable ExtiLine Falling Edge Trigger for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_39
  - LL_EXTI_LINE_40
  - LL_EXTI_LINE_51
  - LL_EXTI_LINE_54
  - LL_EXTI_LINE_56

Return values

- **None:**

Notes

- The configurable wakeup lines are edge-triggered. No glitch must be generated on these lines. If a Falling edge on a configurable interrupt line occurs during a write operation in the EXTI_FTSR register, the pending bit is not set. Rising and falling edge triggers can be set for the same interrupt line. In this case, both generate a trigger condition.

Reference Manual to LL API cross reference:

- FTSR2 FTx LL_EXTI_DisableFallingTrig_32_63

LL_EXTI_DisableFallingTrig_64_95

Function name

__STATIC_INLINE void LL_EXTI_DisableFallingTrig_64_95 (uint32_t ExtiLine)

Function description

Disable ExtiLine Falling Edge Trigger for Lines in range 64 to 95.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_66
  - LL_EXTI_LINE_68
  - LL_EXTI_LINE_69
  - LL_EXTI_LINE_70
  - LL_EXTI_LINE_71
  - LL_EXTI_LINE_72
  - LL_EXTI_LINE_73
  - LL_EXTI_LINE_74

Return values

- **None:**

Notes

- The configurable wakeup lines are edge-triggered. No glitch must be generated on these lines. If a Falling edge on a configurable interrupt line occurs during a write operation in the EXTI_FTSR register, the pending bit is not set. Rising and falling edge triggers can be set for the same interrupt line. In this case, both generate a trigger condition.

Reference Manual to LL API cross reference:

- FTSR3 FTx LL_EXTI_DisableFallingTrig_64_95

LL_EXTI_IsEnabledFallingTrig_0_31

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsEnabledFallingTrig_0_31 (uint32_t ExtiLine)

Function description

Check if falling edge trigger is enabled for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21

Return values

- **State:** of bit (1 or 0).

Notes

- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- FTSR1 FTx LL_EXTI_IsEnabledFallingTrig_0_31

LL_EXTI_IsEnabledFallingTrig_32_63

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsEnabledFallingTrig_32_63 (uint32_t ExtiLine)

Function description

Check if falling edge trigger is enabled for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_39
  - LL_EXTI_LINE_40
  - LL_EXTI_LINE_51
  - LL_EXTI_LINE_54
  - LL_EXTI_LINE_56

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- FTSR2 FTx LL_EXTI_IsEnabledFallingTrig_32_63

LL_EXTI_IsEnabledFallingTrig_64_95

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsEnabledFallingTrig_64_95 (uint32_t ExtiLine)

Function description

Check if falling edge trigger is enabled for Lines in range 64 to 95.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_66
  - LL_EXTI_LINE_68
  - LL_EXTI_LINE_69
  - LL_EXTI_LINE_70
  - LL_EXTI_LINE_71
  - LL_EXTI_LINE_72
  - LL_EXTI_LINE_73
  - LL_EXTI_LINE_74

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- FTSR3 FTx LL_EXTI_IsEnabledFallingTrig_64_95

LL_EXTI_GenerateSWI_0_31

Function name

__STATIC_INLINE void LL_EXTI_GenerateSWI_0_31 (uint32_t ExtiLine)

Function description

Generate a software Interrupt Event for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21

Return values

- **None:**

Notes

- If the interrupt is enabled on this line in the EXTI_IMR, writing a 1 to this bit when it is at '0' sets the corresponding pending bit in EXTI_PR resulting in an interrupt request generation. This bit is cleared by clearing the corresponding bit in the EXTI_PR register (by writing a 1 into the bit)
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- SWIER1 SWIx LL_EXTI_GenerateSWI_0_31

LL_EXTI_GenerateSWI_32_63

Function name

__STATIC_INLINE void LL_EXTI_GenerateSWI_32_63 (uint32_t ExtiLine)

Function description

Generate a software Interrupt Event for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_39
  - LL_EXTI_LINE_40
  - LL_EXTI_LINE_51
  - LL_EXTI_LINE_54
  - LL_EXTI_LINE_56

Return values

- **None:**

Notes

- If the interrupt is enabled on this line in the EXTI_IMR2, writing a 1 to this bit when it is at '0' sets the corresponding pending bit in EXTI_PR2 resulting in an interrupt request generation. This bit is cleared by clearing the corresponding bit in the EXTI_PR2 register (by writing a 1 into the bit)

Reference Manual to LL API cross reference:

- SWIER2 SWIx LL_EXTI_GenerateSWI_32_63

LL_EXTI_GenerateSWI_64_95

Function name

__STATIC_INLINE void LL_EXTI_GenerateSWI_64_95 (uint32_t ExtiLine)

Function description

Generate a software Interrupt Event for Lines in range 64 to 95.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_66
  - LL_EXTI_LINE_68
  - LL_EXTI_LINE_69
  - LL_EXTI_LINE_70
  - LL_EXTI_LINE_71
  - LL_EXTI_LINE_72
  - LL_EXTI_LINE_73
  - LL_EXTI_LINE_74

Return values

- **None:**

Notes

- If the interrupt is enabled on this line in the EXTI_IMR3, writing a 1 to this bit when it is at '0' sets the corresponding pending bit in EXTI_PR3 resulting in an interrupt request generation. This bit is cleared by clearing the corresponding bit in the EXTI_PR3 register (by writing a 1 into the bit)

Reference Manual to LL API cross reference:

- SWIER2 SWIx LL_EXTI_GenerateSWI_64_95

LL_EXTI_IsActiveFallingFlag_0_31

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsActiveFallingFlag_0_31 (uint32_t ExtiLine)

Function description

Check if the ExtLine Falling Flag is set or not for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21

Return values

- **State:** of bit (1 or 0).

Notes

- This bit is set when the falling edge event arrives on the interrupt line. This bit is cleared by writing a 1 to the bit.
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- FPR1 FPIFx LL_EXTI_IsActiveFallingFlag_0_31

LL_EXTI_IsActiveFallingFlag_32_63

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsActiveFallingFlag_32_63 (uint32_t ExtiLine)

Function description

Check if the ExtLine Flag is set or not for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_39
  - LL_EXTI_LINE_40
  - LL_EXTI_LINE_51
  - LL_EXTI_LINE_54
  - LL_EXTI_LINE_56

Return values

- **State:** of bit (1 or 0).

Notes

- This bit is set when the selected edge event arrives on the interrupt line. This bit is cleared by writing a 1 to the bit.

Reference Manual to LL API cross reference:

- FPR2 FPIFx LL_EXTI_IsActiveFallingFlag_32_63

LL_EXTI_IsActiveFallingFlag_64_95

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsActiveFallingFlag_64_95 (uint32_t ExtiLine)

Function description

Check if the ExtLine Flag is set or not for Lines in range 64 to 95.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_66
  - LL_EXTI_LINE_68
  - LL_EXTI_LINE_69
  - LL_EXTI_LINE_70
  - LL_EXTI_LINE_71
  - LL_EXTI_LINE_72
  - LL_EXTI_LINE_73
  - LL_EXTI_LINE_74

Return values

- **State:** of bit (1 or 0).

Notes

- This bit is set when the selected edge event arrives on the interrupt line. This bit is cleared by writing a 1 to the bit.

Reference Manual to LL API cross reference:

- FPR3 FPIFx LL_EXTI_IsActiveFallingFlag_64_95

LL_EXTI_ReadFallingFlag_0_31

Function name

__STATIC_INLINE uint32_t LL_EXTI_ReadFallingFlag_0_31 (uint32_t ExtiLine)

Function description

Read ExtLine Combination Falling Flag for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21

Return values

- **@note:** This bit is set when the selected edge event arrives on the interrupt

Notes

- This bit is set when the falling edge event arrives on the interrupt line. This bit is cleared by writing a 1 to the bit.
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- FPR1 FPIFx LL_EXTI_ReadFallingFlag_0_31

LL_EXTI_ReadFallingFlag_32_63

Function name

__STATIC_INLINE uint32_t LL_EXTI_ReadFallingFlag_32_63 (uint32_t ExtiLine)

Function description

Read ExtLine Combination Flag for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_39
  - LL_EXTI_LINE_40
  - LL_EXTI_LINE_51
  - LL_EXTI_LINE_54
  - LL_EXTI_LINE_56

Return values

- **@note:** This bit is set when the selected edge event arrives on the interrupt

Notes

- This bit is set when the selected edge event arrives on the interrupt line. This bit is cleared by writing a 1 to the bit.

Reference Manual to LL API cross reference:

- FPR2 FPIFx LL_EXTI_ReadFallingFlag_32_63

LL_EXTI_ReadFallingFlag_64_95

Function name

__STATIC_INLINE uint32_t LL_EXTI_ReadFallingFlag_64_95 (uint32_t ExtiLine)

Function description

Read ExtLine Combination Flag for Lines in range 64 to 95.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_66
  - LL_EXTI_LINE_68
  - LL_EXTI_LINE_69
  - LL_EXTI_LINE_70
  - LL_EXTI_LINE_71
  - LL_EXTI_LINE_72
  - LL_EXTI_LINE_73
  - LL_EXTI_LINE_74

Return values

- **@note:** This bit is set when the selected edge event arrives on the interrupt

Notes

- This bit is set when the selected edge event arrives on the interrupt line. This bit is cleared by writing a 1 to the bit.

Reference Manual to LL API cross reference:

- FPR3 FPIFx LL_EXTI_ReadFallingFlag_64_95

LL_EXTI_ClearFallingFlag_0_31

Function name

__STATIC_INLINE void LL_EXTI_ClearFallingFlag_0_31 (uint32_t ExtiLine)

Function description

Clear ExtLine Falling Flags for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21

Return values

- **None:**

Notes

- This bit is set when the falling edge event arrives on the interrupt line. This bit is cleared by writing a 1 to the bit.
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- FPR1 FPIFx LL_EXTI_ClearFallingFlag_0_31

LL_EXTI_ClearFallingFlag_32_63

Function name

__STATIC_INLINE void LL_EXTI_ClearFallingFlag_32_63 (uint32_t ExtiLine)

Function description

Clear ExtLine Flags for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_39
  - LL_EXTI_LINE_40
  - LL_EXTI_LINE_51
  - LL_EXTI_LINE_54
  - LL_EXTI_LINE_56

Return values

- **None:**

Notes

- This bit is set when the selected edge event arrives on the interrupt line. This bit is cleared by writing a 1 to the bit.

Reference Manual to LL API cross reference:

- FPR2 FPIFx LL_EXTI_ClearFallingFlag_32_63

LL_EXTI_ClearFallingFlag_64_95

Function name

__STATIC_INLINE void LL_EXTI_ClearFallingFlag_64_95 (uint32_t ExtiLine)

Function description

Clear ExtLine Flags for Lines in range 64 to 95.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_66
  - LL_EXTI_LINE_68
  - LL_EXTI_LINE_69
  - LL_EXTI_LINE_70
  - LL_EXTI_LINE_71
  - LL_EXTI_LINE_72
  - LL_EXTI_LINE_73
  - LL_EXTI_LINE_74

Return values

- **None:**

Notes

- This bit is set when the selected edge event arrives on the interrupt line. This bit is cleared by writing a 1 to the bit.

Reference Manual to LL API cross reference:

- FPR3 FPIFx LL_EXTI_ClearFallingFlag_64_95

LL_EXTI_IsActiveRisingFlag_0_31

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsActiveRisingFlag_0_31 (uint32_t ExtiLine)

Function description

Check if the ExtLine Rising Flag is set or not for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21

Return values

- **State:** of bit (1 or 0).

Notes

- This bit is set when the Rising edge event arrives on the interrupt line. This bit is cleared by writing a 1 to the bit.
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- RPR1 RPIFx LL_EXTI_IsActiveRisingFlag_0_31

LL_EXTI_IsActiveRisingFlag_32_63

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsActiveRisingFlag_32_63 (uint32_t ExtiLine)

Function description

Check if the ExtLine Rising Flag is set or not for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_39
  - LL_EXTI_LINE_40
  - LL_EXTI_LINE_51
  - LL_EXTI_LINE_54
  - LL_EXTI_LINE_56

Return values

- **State:** of bit (1 or 0).

Notes

- This bit is set when the Rising edge event arrives on the interrupt line. This bit is cleared by writing a 1 to the bit.
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- RPR2 RPIFx LL_EXTI_IsActiveRisingFlag_32_63

LL_EXTI_IsActiveRisingFlag_64_95

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsActiveRisingFlag_64_95 (uint32_t ExtiLine)

Function description

Check if the ExtLine Rising Flag is set or not for Lines in range 64 to 95.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_66
  - LL_EXTI_LINE_68
  - LL_EXTI_LINE_69
  - LL_EXTI_LINE_70
  - LL_EXTI_LINE_71
  - LL_EXTI_LINE_72
  - LL_EXTI_LINE_73
  - LL_EXTI_LINE_74

Return values

- **State:** of bit (1 or 0).

Notes

- This bit is set when the Rising edge event arrives on the interrupt line. This bit is cleared by writing a 1 to the bit.
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- RPR3 RPIFx LL_EXTI_IsActiveRisingFlag_64_95

LL_EXTI_ReadRisingFlag_0_31

Function name

__STATIC_INLINE uint32_t LL_EXTI_ReadRisingFlag_0_31 (uint32_t ExtiLine)

Function description

Read ExtLine Combination Rising Flag for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21

Return values

- **@note:** This bit is set when the selected edge event arrives on the interrupt

Notes

- This bit is set when the Rising edge event arrives on the interrupt line. This bit is cleared by writing a 1 to the bit.
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- RPR1 RPIFx LL_EXTI_ReadRisingFlag_0_31

LL_EXTI_ReadRisingFlag_32_63

Function name

__STATIC_INLINE uint32_t LL_EXTI_ReadRisingFlag_32_63 (uint32_t ExtiLine)

Function description

Read ExtLine Combination Rising Flag for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_39
  - LL_EXTI_LINE_40
  - LL_EXTI_LINE_51
  - LL_EXTI_LINE_54
  - LL_EXTI_LINE_56

Return values

- **@note:** This bit is set when the selected edge event arrives on the interrupt

Notes

- This bit is set when the Rising edge event arrives on the interrupt line. This bit is cleared by writing a 1 to the bit.
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- RPR2 RPIFx LL_EXTI_ReadRisingFlag_32_63

LL_EXTI_ReadRisingFlag_64_95

Function name

__STATIC_INLINE uint32_t LL_EXTI_ReadRisingFlag_64_95 (uint32_t ExtiLine)

Function description

Read ExtLine Combination Rising Flag for Lines in range 64 to 95.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_66
  - LL_EXTI_LINE_68
  - LL_EXTI_LINE_69
  - LL_EXTI_LINE_70
  - LL_EXTI_LINE_71
  - LL_EXTI_LINE_72
  - LL_EXTI_LINE_73
  - LL_EXTI_LINE_74

Return values

- **@note:** This bit is set when the selected edge event arrives on the interrupt

Notes

- This bit is set when the Rising edge event arrives on the interrupt line. This bit is cleared by writing a 1 to the bit.
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- RPR3 RPIFx LL_EXTI_ReadRisingFlag_64_95

LL_EXTI_ClearRisingFlag_0_31

Function name

__STATIC_INLINE void LL_EXTI_ClearRisingFlag_0_31 (uint32_t ExtiLine)

Function description

Clear ExtLine Rising Flags for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21

Return values

- **None:**

Notes

- This bit is set when the Rising edge event arrives on the interrupt line. This bit is cleared by writing a 1 to the bit.
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- RPR1 RPIFx LL_EXTI_ClearRisingFlag_0_31

LL_EXTI_ClearRisingFlag_32_63

Function name

__STATIC_INLINE void LL_EXTI_ClearRisingFlag_32_63 (uint32_t ExtiLine)

Function description

Clear ExtLine Rising Flags for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_39
  - LL_EXTI_LINE_40
  - LL_EXTI_LINE_51
  - LL_EXTI_LINE_54
  - LL_EXTI_LINE_56

Return values

- **None:**

Notes

- This bit is set when the Rising edge event arrives on the interrupt line. This bit is cleared by writing a 1 to the bit.
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- RPR2 RPIFx LL_EXTI_ClearRisingFlag_32_63

LL_EXTI_ClearRisingFlag_64_95

Function name

__STATIC_INLINE void LL_EXTI_ClearRisingFlag_64_95 (uint32_t ExtiLine)

Function description

Clear ExtLine Rising Flags for Lines in range 64 to 95.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_66
  - LL_EXTI_LINE_68
  - LL_EXTI_LINE_69
  - LL_EXTI_LINE_70
  - LL_EXTI_LINE_71
  - LL_EXTI_LINE_72
  - LL_EXTI_LINE_73
  - LL_EXTI_LINE_74

Return values

- **None:**

Notes

- This bit is set when the Rising edge event arrives on the interrupt line. This bit is cleared by writing a 1 to the bit.
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- RPR3 RPIFx LL_EXTI_ClearRisingFlag_64_95

LL_EXTI_SetEXTISource

Function name

__STATIC_INLINE void LL_EXTI_SetEXTISource (uint32_t Port, uint32_t Line)

Function description

Configure source input for the EXTI external interrupt.

Parameters

- **Port:** This parameter can be one of the following values:
  - LL_EXTI_EXTI_PORTA
  - LL_EXTI_EXTI_PORTB
  - LL_EXTI_EXTI_PORTC
  - LL_EXTI_EXTI_PORTD
  - LL_EXTI_EXTI_PORTE
  - LL_EXTI_EXTI_PORTF
  - LL_EXTI_EXTI_PORTG
  - LL_EXTI_EXTI_PORTH
  - LL_EXTI_EXTI_PORTN
  - LL_EXTI_EXTI_PORTO
  - LL_EXTI_EXTI_PORTP
  - LL_EXTI_EXTI_PORTQ
- **Line:** This parameter can be one of the following values:
  - LL_EXTI_EXTI_LINE0
  - LL_EXTI_EXTI_LINE1
  - LL_EXTI_EXTI_LINE2
  - LL_EXTI_EXTI_LINE3
  - LL_EXTI_EXTI_LINE4
  - LL_EXTI_EXTI_LINE5
  - LL_EXTI_EXTI_LINE6
  - LL_EXTI_EXTI_LINE7
  - LL_EXTI_EXTI_LINE8
  - LL_EXTI_EXTI_LINE9
  - LL_EXTI_EXTI_LINE10
  - LL_EXTI_EXTI_LINE11
  - LL_EXTI_EXTI_LINE12
  - LL_EXTI_EXTI_LINE13
  - LL_EXTI_EXTI_LINE14
  - LL_EXTI_EXTI_LINE15

Return values

- **None:**

Reference Manual to LL API cross reference:

- EXTI_EXTICR1 EXTI0 LL_EXTI_SetEXTISource
- EXTI_EXTICR1 EXTI1 LL_EXTI_SetEXTISource
- EXTI_EXTICR1 EXTI2 LL_EXTI_SetEXTISource
- EXTI_EXTICR1 EXTI3 LL_EXTI_SetEXTISource
- EXTI_EXTICR2 EXTI4 LL_EXTI_SetEXTISource
- EXTI_EXTICR2 EXTI5 LL_EXTI_SetEXTISource
- EXTI_EXTICR2 EXTI6 LL_EXTI_SetEXTISource
- EXTI_EXTICR2 EXTI7 LL_EXTI_SetEXTISource
- EXTI_EXTICR3 EXTI8 LL_EXTI_SetEXTISource
- EXTI_EXTICR3 EXTI9 LL_EXTI_SetEXTISource
- EXTI_EXTICR3 EXTI10 LL_EXTI_SetEXTISource
- EXTI_EXTICR3 EXTI11 LL_EXTI_SetEXTISource
- EXTI_EXTICR4 EXTI12 LL_EXTI_SetEXTISource
- EXTI_EXTICR4 EXTI13 LL_EXTI_SetEXTISource
- EXTI_EXTICR4 EXTI14 LL_EXTI_SetEXTISource
- EXTI_EXTICR4 EXTI15 LL_EXTI_SetEXTISource

LL_EXTI_GetEXTISource

Function name

__STATIC_INLINE uint32_t LL_EXTI_GetEXTISource (uint32_t Line)

Function description

Get the configured defined for specific EXTI Line.

Parameters

- **Line:** This parameter can be one of the following values:
  - LL_EXTI_EXTI_LINE0
  - LL_EXTI_EXTI_LINE1
  - LL_EXTI_EXTI_LINE2
  - LL_EXTI_EXTI_LINE3
  - LL_EXTI_EXTI_LINE4
  - LL_EXTI_EXTI_LINE5
  - LL_EXTI_EXTI_LINE6
  - LL_EXTI_EXTI_LINE7
  - LL_EXTI_EXTI_LINE8
  - LL_EXTI_EXTI_LINE9
  - LL_EXTI_EXTI_LINE10
  - LL_EXTI_EXTI_LINE11
  - LL_EXTI_EXTI_LINE12
  - LL_EXTI_EXTI_LINE13
  - LL_EXTI_EXTI_LINE14
  - LL_EXTI_EXTI_LINE15

Return values

- **Returned:** value can be one of the following values:
  - LL_EXTI_EXTI_PORTA
  - LL_EXTI_EXTI_PORTB
  - LL_EXTI_EXTI_PORTC
  - LL_EXTI_EXTI_PORTD
  - LL_EXTI_EXTI_PORTE
  - LL_EXTI_EXTI_PORTF
  - LL_EXTI_EXTI_PORTG
  - LL_EXTI_EXTI_PORTH
  - LL_EXTI_EXTI_PORTN
  - LL_EXTI_EXTI_PORTO
  - LL_EXTI_EXTI_PORTP
  - LL_EXTI_EXTI_PORTQ

Reference Manual to LL API cross reference:

- EXTI_EXTICR1 EXTI0 LL_EXTI_GetEXTISource
- EXTI_EXTICR1 EXTI1 LL_EXTI_GetEXTISource
- EXTI_EXTICR1 EXTI2 LL_EXTI_GetEXTISource
- EXTI_EXTICR1 EXTI3 LL_EXTI_GetEXTISource
- EXTI_EXTICR2 EXTI4 LL_EXTI_GetEXTISource
- EXTI_EXTICR2 EXTI5 LL_EXTI_GetEXTISource
- EXTI_EXTICR2 EXTI6 LL_EXTI_GetEXTISource
- EXTI_EXTICR2 EXTI7 LL_EXTI_GetEXTISource
- EXTI_EXTICR3 EXTI8 LL_EXTI_GetEXTISource
- EXTI_EXTICR3 EXTI9 LL_EXTI_GetEXTISource
- EXTI_EXTICR3 EXTI10 LL_EXTI_GetEXTISource
- EXTI_EXTICR3 EXTI11 LL_EXTI_GetEXTISource
- EXTI_EXTICR4 EXTI12 LL_EXTI_GetEXTISource
- EXTI_EXTICR4 EXTI13 LL_EXTI_GetEXTISource
- EXTI_EXTICR4 EXTI14 LL_EXTI_GetEXTISource
- EXTI_EXTICR4 EXTI15 LL_EXTI_GetEXTISource

LL_EXTI_EnableSecure_0_31

Function name

__STATIC_INLINE void LL_EXTI_EnableSecure_0_31 (uint32_t ExtiLine)

Function description

Enable ExtiLine Secure attribute for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_17
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21
  - LL_EXTI_LINE_22
  - LL_EXTI_LINE_23
  - LL_EXTI_LINE_24
  - LL_EXTI_LINE_25
  - LL_EXTI_LINE_26
  - LL_EXTI_LINE_27
  - LL_EXTI_LINE_28
  - LL_EXTI_LINE_29
  - LL_EXTI_LINE_30
  - LL_EXTI_LINE_31
  - LL_EXTI_LINE_ALL_0_31

Return values

- **None:**

Notes

- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- SECCFGR1 SECx LL_EXTI_EnableSecure_0_31

LL_EXTI_EnableSecure_32_63

Function name

__STATIC_INLINE void LL_EXTI_EnableSecure_32_63 (uint32_t ExtiLine)

Function description

Enable ExtiLine Secure attribute for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_32
  - LL_EXTI_LINE_33
  - LL_EXTI_LINE_34
  - LL_EXTI_LINE_35
  - LL_EXTI_LINE_36
  - LL_EXTI_LINE_37
  - LL_EXTI_LINE_38
  - LL_EXTI_LINE_39
  - LL_EXTI_LINE_40
  - LL_EXTI_LINE_41
  - LL_EXTI_LINE_42
  - LL_EXTI_LINE_43
  - LL_EXTI_LINE_44
  - LL_EXTI_LINE_45
  - LL_EXTI_LINE_46
  - LL_EXTI_LINE_47
  - LL_EXTI_LINE_48
  - LL_EXTI_LINE_49
  - LL_EXTI_LINE_50
  - LL_EXTI_LINE_51
  - LL_EXTI_LINE_52
  - LL_EXTI_LINE_53
  - LL_EXTI_LINE_54
  - LL_EXTI_LINE_55
  - LL_EXTI_LINE_56
  - LL_EXTI_LINE_57
  - LL_EXTI_LINE_58
  - LL_EXTI_LINE_62
  - LL_EXTI_LINE_61
  - LL_EXTI_LINE_62
  - LL_EXTI_LINE_63
  - LL_EXTI_LINE_ALL_32_63

Return values

- **None:**

Notes

- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- SECCFGR2 SECx LL_EXTI_EnableSecure_32_63

LL_EXTI_EnableSecure_64_95

Function name

__STATIC_INLINE void LL_EXTI_EnableSecure_64_95 (uint32_t ExtiLine)

Function description

Enable ExtiLine Secure attribute for Lines in range 64 to 95.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_64
  - LL_EXTI_LINE_65
  - LL_EXTI_LINE_66
  - LL_EXTI_LINE_68
  - LL_EXTI_LINE_69
  - LL_EXTI_LINE_70
  - LL_EXTI_LINE_71
  - LL_EXTI_LINE_72
  - LL_EXTI_LINE_73
  - LL_EXTI_LINE_74
  - LL_EXTI_LINE_77
  - LL_EXTI_LINE_ALL_64_95

Return values

- **None:**

Notes

- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- SECCFGR3 SECx LL_EXTI_EnableSecure_64_95

LL_EXTI_DisableSecure_0_31

Function name

__STATIC_INLINE void LL_EXTI_DisableSecure_0_31 (uint32_t ExtiLine)

Function description

Disable ExtiLine Secure attribute for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_17
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21
  - LL_EXTI_LINE_22
  - LL_EXTI_LINE_23
  - LL_EXTI_LINE_24
  - LL_EXTI_LINE_25
  - LL_EXTI_LINE_26
  - LL_EXTI_LINE_27
  - LL_EXTI_LINE_28
  - LL_EXTI_LINE_29
  - LL_EXTI_LINE_30
  - LL_EXTI_LINE_31
  - LL_EXTI_LINE_ALL_0_31

Return values

- **None:**

Notes

- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- SECCFGR1 SECx LL_EXTI_DisableSecure_0_31

LL_EXTI_DisableSecure_32_63

Function name

__STATIC_INLINE void LL_EXTI_DisableSecure_32_63 (uint32_t ExtiLine)

Function description

Disable ExtiLine Secure attribute for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_32
  - LL_EXTI_LINE_33
  - LL_EXTI_LINE_34
  - LL_EXTI_LINE_35
  - LL_EXTI_LINE_36
  - LL_EXTI_LINE_37
  - LL_EXTI_LINE_38
  - LL_EXTI_LINE_39
  - LL_EXTI_LINE_40
  - LL_EXTI_LINE_41
  - LL_EXTI_LINE_42
  - LL_EXTI_LINE_43
  - LL_EXTI_LINE_44
  - LL_EXTI_LINE_45
  - LL_EXTI_LINE_46
  - LL_EXTI_LINE_47
  - LL_EXTI_LINE_48
  - LL_EXTI_LINE_49
  - LL_EXTI_LINE_50
  - LL_EXTI_LINE_51
  - LL_EXTI_LINE_52
  - LL_EXTI_LINE_53
  - LL_EXTI_LINE_54
  - LL_EXTI_LINE_55
  - LL_EXTI_LINE_56
  - LL_EXTI_LINE_57
  - LL_EXTI_LINE_58
  - LL_EXTI_LINE_62
  - LL_EXTI_LINE_61
  - LL_EXTI_LINE_62
  - LL_EXTI_LINE_63
  - LL_EXTI_LINE_ALL_32_63

Return values

- **None:**

Notes

- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- SECCFGR2 SECx LL_EXTI_DisableSecure_32_63

LL_EXTI_DisableSecure_64_95

Function name

__STATIC_INLINE void LL_EXTI_DisableSecure_64_95 (uint32_t ExtiLine)

Function description

Disable ExtiLine Secure attribute for Lines in range 64 to 95.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_64
  - LL_EXTI_LINE_65
  - LL_EXTI_LINE_66
  - LL_EXTI_LINE_68
  - LL_EXTI_LINE_69
  - LL_EXTI_LINE_70
  - LL_EXTI_LINE_71
  - LL_EXTI_LINE_72
  - LL_EXTI_LINE_73
  - LL_EXTI_LINE_74
  - LL_EXTI_LINE_77
  - LL_EXTI_LINE_ALL_64_95

Return values

- **None:**

Notes

- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- SECCFGR3 SECx LL_EXTI_DisableSecure_64_95

LL_EXTI_IsEnabledSecure_0_31

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsEnabledSecure_0_31 (uint32_t ExtiLine)

Function description

Indicate if ExtiLine Secure attribute is enabled for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_17
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21
  - LL_EXTI_LINE_22
  - LL_EXTI_LINE_23
  - LL_EXTI_LINE_24
  - LL_EXTI_LINE_25
  - LL_EXTI_LINE_26
  - LL_EXTI_LINE_27
  - LL_EXTI_LINE_28
  - LL_EXTI_LINE_29
  - LL_EXTI_LINE_30
  - LL_EXTI_LINE_31
  - LL_EXTI_LINE_ALL_0_31

Return values

- **State:** of bit (1 or 0).

Notes

- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- SECCFGR1 SECx LL_EXTI_IsEnabledSecure_0_31

LL_EXTI_IsEnabledSecure_32_63

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsEnabledSecure_32_63 (uint32_t ExtiLine)

Function description

Indicate if ExtiLine Secure attribute is enabled for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_32
  - LL_EXTI_LINE_33
  - LL_EXTI_LINE_34
  - LL_EXTI_LINE_35
  - LL_EXTI_LINE_36
  - LL_EXTI_LINE_37
  - LL_EXTI_LINE_38
  - LL_EXTI_LINE_39
  - LL_EXTI_LINE_40
  - LL_EXTI_LINE_41
  - LL_EXTI_LINE_42
  - LL_EXTI_LINE_43
  - LL_EXTI_LINE_44
  - LL_EXTI_LINE_45
  - LL_EXTI_LINE_46
  - LL_EXTI_LINE_47
  - LL_EXTI_LINE_48
  - LL_EXTI_LINE_49
  - LL_EXTI_LINE_50
  - LL_EXTI_LINE_51
  - LL_EXTI_LINE_52
  - LL_EXTI_LINE_53
  - LL_EXTI_LINE_54
  - LL_EXTI_LINE_55
  - LL_EXTI_LINE_56
  - LL_EXTI_LINE_57
  - LL_EXTI_LINE_58
  - LL_EXTI_LINE_62
  - LL_EXTI_LINE_61
  - LL_EXTI_LINE_62
  - LL_EXTI_LINE_63
  - LL_EXTI_LINE_ALL_32_63

Return values

- **State:** of bit (1 or 0).

Notes

- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- SECCFGR2 SECx LL_EXTI_IsEnabledSecure_32_63

LL_EXTI_IsEnabledSecure_64_95

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsEnabledSecure_64_95 (uint32_t ExtiLine)

Function description

Indicate if ExtiLine Secure attribute is enabled for Lines in range 64 to 95.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_64
  - LL_EXTI_LINE_65
  - LL_EXTI_LINE_66
  - LL_EXTI_LINE_68
  - LL_EXTI_LINE_69
  - LL_EXTI_LINE_70
  - LL_EXTI_LINE_71
  - LL_EXTI_LINE_72
  - LL_EXTI_LINE_73
  - LL_EXTI_LINE_74
  - LL_EXTI_LINE_77
  - LL_EXTI_LINE_ALL_64_95

Return values

- **State:** of bit (1 or 0).

Notes

- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- SECCFGR3 SECx LL_EXTI_IsEnabledSecure_64_95

LL_EXTI_EnablePrivilege_0_31

Function name

__STATIC_INLINE void LL_EXTI_EnablePrivilege_0_31 (uint32_t ExtiLine)

Function description

Enable ExtiLine Privilege attribute for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_17
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21
  - LL_EXTI_LINE_22
  - LL_EXTI_LINE_23
  - LL_EXTI_LINE_24
  - LL_EXTI_LINE_25
  - LL_EXTI_LINE_26
  - LL_EXTI_LINE_27
  - LL_EXTI_LINE_28
  - LL_EXTI_LINE_29
  - LL_EXTI_LINE_30
  - LL_EXTI_LINE_31
  - LL_EXTI_LINE_ALL_0_31

Return values

- **None:**

Notes

- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- PRIVCFGR1 PRIVx LL_EXTI_EnablePrivilege_0_31

LL_EXTI_EnablePrivilege_32_63

Function name

__STATIC_INLINE void LL_EXTI_EnablePrivilege_32_63 (uint32_t ExtiLine)

Function description

Enable ExtiLine Privilege attribute for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_32
  - LL_EXTI_LINE_33
  - LL_EXTI_LINE_34
  - LL_EXTI_LINE_35
  - LL_EXTI_LINE_36
  - LL_EXTI_LINE_37
  - LL_EXTI_LINE_38
  - LL_EXTI_LINE_39
  - LL_EXTI_LINE_40
  - LL_EXTI_LINE_41
  - LL_EXTI_LINE_42
  - LL_EXTI_LINE_43
  - LL_EXTI_LINE_44
  - LL_EXTI_LINE_45
  - LL_EXTI_LINE_46
  - LL_EXTI_LINE_47
  - LL_EXTI_LINE_48
  - LL_EXTI_LINE_49
  - LL_EXTI_LINE_50
  - LL_EXTI_LINE_51
  - LL_EXTI_LINE_52
  - LL_EXTI_LINE_53
  - LL_EXTI_LINE_54
  - LL_EXTI_LINE_55
  - LL_EXTI_LINE_56
  - LL_EXTI_LINE_57
  - LL_EXTI_LINE_58
  - LL_EXTI_LINE_62
  - LL_EXTI_LINE_61
  - LL_EXTI_LINE_62
  - LL_EXTI_LINE_63
  - LL_EXTI_LINE_ALL_32_63

Return values

- **None:**

Notes

- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- PRIVCFGR2 PRIVx LL_EXTI_EnablePrivilege_32_63

LL_EXTI_EnablePrivilege_64_95

Function name

__STATIC_INLINE void LL_EXTI_EnablePrivilege_64_95 (uint32_t ExtiLine)

Function description

Enable ExtiLine Privilege attribute for Lines in range 64 to 95.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_64
  - LL_EXTI_LINE_65
  - LL_EXTI_LINE_66
  - LL_EXTI_LINE_68
  - LL_EXTI_LINE_69
  - LL_EXTI_LINE_70
  - LL_EXTI_LINE_71
  - LL_EXTI_LINE_72
  - LL_EXTI_LINE_73
  - LL_EXTI_LINE_74
  - LL_EXTI_LINE_77
  - LL_EXTI_LINE_ALL_64_95

Return values

- **None:**

Notes

- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- PRIVCFGR3 PRIVx LL_EXTI_EnablePrivilege_64_95

LL_EXTI_DisablePrivilege_0_31

Function name

__STATIC_INLINE void LL_EXTI_DisablePrivilege_0_31 (uint32_t ExtiLine)

Function description

Disable ExtiLine Privilege attribute for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_17
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21
  - LL_EXTI_LINE_22
  - LL_EXTI_LINE_23
  - LL_EXTI_LINE_24
  - LL_EXTI_LINE_25
  - LL_EXTI_LINE_26
  - LL_EXTI_LINE_27
  - LL_EXTI_LINE_28
  - LL_EXTI_LINE_29
  - LL_EXTI_LINE_30
  - LL_EXTI_LINE_31
  - LL_EXTI_LINE_ALL_0_31

Return values

- **None:**

Notes

- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- PRIVCFGR1 PRIVx LL_EXTI_DisablePrivilege_0_31

LL_EXTI_DisablePrivilege_32_63

Function name

__STATIC_INLINE void LL_EXTI_DisablePrivilege_32_63 (uint32_t ExtiLine)

Function description

Disable ExtiLine Privilege attribute for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_32
  - LL_EXTI_LINE_33
  - LL_EXTI_LINE_34
  - LL_EXTI_LINE_35
  - LL_EXTI_LINE_36
  - LL_EXTI_LINE_37
  - LL_EXTI_LINE_38
  - LL_EXTI_LINE_39
  - LL_EXTI_LINE_40
  - LL_EXTI_LINE_41
  - LL_EXTI_LINE_42
  - LL_EXTI_LINE_43
  - LL_EXTI_LINE_44
  - LL_EXTI_LINE_45
  - LL_EXTI_LINE_46
  - LL_EXTI_LINE_47
  - LL_EXTI_LINE_48
  - LL_EXTI_LINE_49
  - LL_EXTI_LINE_50
  - LL_EXTI_LINE_51
  - LL_EXTI_LINE_52
  - LL_EXTI_LINE_53
  - LL_EXTI_LINE_54
  - LL_EXTI_LINE_55
  - LL_EXTI_LINE_56
  - LL_EXTI_LINE_57
  - LL_EXTI_LINE_58
  - LL_EXTI_LINE_62
  - LL_EXTI_LINE_61
  - LL_EXTI_LINE_62
  - LL_EXTI_LINE_63
  - LL_EXTI_LINE_ALL_32_63

Return values

- **None:**

Notes

- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- PRIVCFGR2 PRIVx LL_EXTI_DisablePrivilege_32_63

LL_EXTI_DisablePrivilege_64_95

Function name

__STATIC_INLINE void LL_EXTI_DisablePrivilege_64_95 (uint32_t ExtiLine)

Function description

Disable ExtiLine Privilege attribute for Lines in range 64 to 95.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_64
  - LL_EXTI_LINE_65
  - LL_EXTI_LINE_66
  - LL_EXTI_LINE_68
  - LL_EXTI_LINE_69
  - LL_EXTI_LINE_70
  - LL_EXTI_LINE_71
  - LL_EXTI_LINE_72
  - LL_EXTI_LINE_73
  - LL_EXTI_LINE_74
  - LL_EXTI_LINE_77
  - LL_EXTI_LINE_ALL_64_95

Return values

- **None:**

Notes

- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- PRIVCFGR3 PRIVx LL_EXTI_DisablePrivilege_64_95

LL_EXTI_IsEnabledPrivilege_0_31

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsEnabledPrivilege_0_31 (uint32_t ExtiLine)

Function description

Indicate if ExtiLine Privilege attribute is enabled for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_17
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21
  - LL_EXTI_LINE_22
  - LL_EXTI_LINE_23
  - LL_EXTI_LINE_24
  - LL_EXTI_LINE_25
  - LL_EXTI_LINE_26
  - LL_EXTI_LINE_27
  - LL_EXTI_LINE_28
  - LL_EXTI_LINE_29
  - LL_EXTI_LINE_30
  - LL_EXTI_LINE_31
  - LL_EXTI_LINE_ALL_0_31

Return values

- **State:** of bit (1 or 0).

Notes

- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- PRIVCFGR1 PRIVx LL_EXTI_IsEnabledPrivilege_0_31

LL_EXTI_IsEnabledPrivilege_32_63

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsEnabledPrivilege_32_63 (uint32_t ExtiLine)

Function description

Indicate if ExtiLine Privilege attribute is enabled for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_32
  - LL_EXTI_LINE_33
  - LL_EXTI_LINE_34
  - LL_EXTI_LINE_35
  - LL_EXTI_LINE_36
  - LL_EXTI_LINE_37
  - LL_EXTI_LINE_38
  - LL_EXTI_LINE_39
  - LL_EXTI_LINE_40
  - LL_EXTI_LINE_41
  - LL_EXTI_LINE_42
  - LL_EXTI_LINE_43
  - LL_EXTI_LINE_44
  - LL_EXTI_LINE_45
  - LL_EXTI_LINE_46
  - LL_EXTI_LINE_47
  - LL_EXTI_LINE_48
  - LL_EXTI_LINE_49
  - LL_EXTI_LINE_50
  - LL_EXTI_LINE_51
  - LL_EXTI_LINE_52
  - LL_EXTI_LINE_53
  - LL_EXTI_LINE_54
  - LL_EXTI_LINE_55
  - LL_EXTI_LINE_56
  - LL_EXTI_LINE_57
  - LL_EXTI_LINE_58
  - LL_EXTI_LINE_62
  - LL_EXTI_LINE_61
  - LL_EXTI_LINE_62
  - LL_EXTI_LINE_63
  - LL_EXTI_LINE_ALL_32_63

Return values

- **State:** of bit (1 or 0).

Notes

- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- PRIVCFGR2 PRIVx LL_EXTI_IsEnabledPrivilege_32_63

LL_EXTI_IsEnabledPrivilege_64_95

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsEnabledPrivilege_64_95 (uint32_t ExtiLine)

Function description

Indicate if ExtiLine Privilege attribute is enabled for Lines in range 64 to 95.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_17
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21
  - LL_EXTI_LINE_22
  - LL_EXTI_LINE_23
  - LL_EXTI_LINE_24
  - LL_EXTI_LINE_25
  - LL_EXTI_LINE_26
  - LL_EXTI_LINE_27
  - LL_EXTI_LINE_28
  - LL_EXTI_LINE_29
  - LL_EXTI_LINE_30
  - LL_EXTI_LINE_31
  - LL_EXTI_LINE_ALL_0_31

Return values

- **State:** of bit (1 or 0).

Notes

- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- PRIVCFGR3 PRIVx LL_EXTI_IsEnabledPrivilege_64_95

LL_EXTI_LockAttributes

Function name

__STATIC_INLINE void LL_EXTI_LockAttributes (void)

Function description

Lock the secure and privilege configuration registers.

Return values

- **None:**

Reference Manual to LL API cross reference:

- LOCKR LOCK LL_EXTI_LockAttributes

LL_EXTI_GetLockAttributes

Function name

__STATIC_INLINE uint32_t LL_EXTI_GetLockAttributes (void)

Function description

Return the secure and privilege configuration registers LOCK status.

Return values

- **1:** if the secure and privilege configuration registers have been locked else 0.

Reference Manual to LL API cross reference:

- LOCKR LOCK LL_EXTI_GetLockAttributes

LL_EXTI_Init

Function name

uint32_t LL_EXTI_Init (LL_EXTI_InitTypeDef * EXTI_InitStruct)

Function description

Initialize the EXTI registers according to the specified parameters in EXTI_InitStruct.

Parameters

- **EXTI_InitStruct:** pointer to a LL_EXTI_InitTypeDef structure.

Return values

- **An:** ErrorStatus enumeration value:
  - 0x00: EXTI registers are initialized
  - any other value : wrong configuration

LL_EXTI_DeInit

Function name

uint32_t LL_EXTI_DeInit (void)

Function description

De-initialize the EXTI registers to their default reset values.

Return values

- **An:** ErrorStatus enumeration value:
  - 0x00: EXTI registers are de-initialized

LL_EXTI_StructInit

Function name

void LL_EXTI_StructInit (LL_EXTI_InitTypeDef * EXTI_InitStruct)

Function description

Set each LL_EXTI_InitTypeDef field to default value.

Parameters

- **EXTI_InitStruct:** Pointer to a LL_EXTI_InitTypeDef structure.

Return values

- **None:**

92.3 EXTI Firmware driver defines

The following section lists the various define and macros of the module.

92.3.1 EXTI

EXTI

LINE

LL_EXTI_LINE_0

Extended line 0

LL_EXTI_LINE_1

Extended line 1

LL_EXTI_LINE_2

Extended line 2

LL_EXTI_LINE_3

Extended line 3

LL_EXTI_LINE_4

Extended line 4

LL_EXTI_LINE_5

Extended line 5

LL_EXTI_LINE_6

Extended line 6

LL_EXTI_LINE_7

Extended line 7

LL_EXTI_LINE_8

Extended line 8

LL_EXTI_LINE_9

Extended line 9

LL_EXTI_LINE_10

Extended line 10

LL_EXTI_LINE_11

Extended line 11

LL_EXTI_LINE_12

Extended line 12

LL_EXTI_LINE_13

Extended line 13

LL_EXTI_LINE_14

Extended line 14

LL_EXTI_LINE_15

Extended line 15

LL_EXTI_LINE_17

Extended line 17

LL_EXTI_LINE_18

Extended line 18

LL_EXTI_LINE_19

Extended line 19

LL_EXTI_LINE_20

Extended line 20

LL_EXTI_LINE_21

Extended line 21

LL_EXTI_LINE_22

Extended line 22

LL_EXTI_LINE_23

Extended line 23

LL_EXTI_LINE_24

Extended line 24

LL_EXTI_LINE_25

Extended line 25

LL_EXTI_LINE_26

Extended line 26

LL_EXTI_LINE_27

Extended line 27

LL_EXTI_LINE_28

Extended line 28

LL_EXTI_LINE_29

Extended line 29

LL_EXTI_LINE_30

Extended line 30

LL_EXTI_LINE_31

Extended line 31

LL_EXTI_LINE_ALL_0_31

ALL Extended line

LL_EXTI_LINE_32

Extended line 32

LL_EXTI_LINE_33

Extended line 33

LL_EXTI_LINE_34

Extended line 34

LL_EXTI_LINE_35

Extended line 35

LL_EXTI_LINE_36

Extended line 36

LL_EXTI_LINE_37

Extended line 37

LL_EXTI_LINE_38

Extended line 38

LL_EXTI_LINE_39

Extended line 39

LL_EXTI_LINE_40

Extended line 40

LL_EXTI_LINE_41

Extended line 41

LL_EXTI_LINE_42

Extended line 42

LL_EXTI_LINE_43

Extended line 43

LL_EXTI_LINE_44

Extended line 44

LL_EXTI_LINE_45

Extended line 45

LL_EXTI_LINE_46

Extended line 46

LL_EXTI_LINE_47

Extended line 47

LL_EXTI_LINE_48

Extended line 48

LL_EXTI_LINE_49

Extended line 49

LL_EXTI_LINE_50

Extended line 50

LL_EXTI_LINE_51

Extended line 51

LL_EXTI_LINE_52

Extended line 52

LL_EXTI_LINE_53

Extended line 53

LL_EXTI_LINE_54

Extended line 54

LL_EXTI_LINE_55

Extended line 55

LL_EXTI_LINE_56

Extended line 56

LL_EXTI_LINE_57

Extended line 57

LL_EXTI_LINE_58

Extended line 58

LL_EXTI_LINE_60

Extended line 60

LL_EXTI_LINE_61

Extended line 61

LL_EXTI_LINE_62

Extended line 62

LL_EXTI_LINE_63

Extended line 63

LL_EXTI_LINE_ALL_32_63

All Extended line not reserved

LL_EXTI_LINE_64

Extended line 64

LL_EXTI_LINE_65

Extended line 65

LL_EXTI_LINE_66

Extended line 66

LL_EXTI_LINE_68

Extended line 68

LL_EXTI_LINE_69

Extended line 69

LL_EXTI_LINE_70

Extended line 70

LL_EXTI_LINE_71

Extended line 71

LL_EXTI_LINE_72

Extended line 72

LL_EXTI_LINE_73

Extended line 73

LL_EXTI_LINE_74

Extended line 74

LL_EXTI_LINE_77

Extended line 77

LL_EXTI_LINE_ALL_64_95

All Extended line not reserved

LL_EXTI_LINE_NONE

None Extended line

Mode

LL_EXTI_MODE_IT

Interrupt Mode

LL_EXTI_MODE_EVENT

Event Mode

LL_EXTI_MODE_IT_EVENT

Interrupt & Event Mode

Edge Trigger

LL_EXTI_TRIGGER_NONE

No Trigger Mode

LL_EXTI_TRIGGER_RISING

Trigger Rising Mode

LL_EXTI_TRIGGER_FALLING

Trigger Falling Mode

LL_EXTI_TRIGGER_RISING_FALLING

Trigger Rising & Falling Mode

Common Write and read registers Macros

LL_EXTI_WriteReg

Description:

- Write a value in EXTI register.

Parameters:

- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_EXTI_ReadReg

Description:

- Read a value in EXTI register.

Parameters:

- __REG__: Register to be read

Return value:

- Register: value

93 LL GPIO Generic Driver

93.1 GPIO Firmware driver registers structures

93.1.1 LL_GPIO_InitTypeDef

LL_GPIO_InitTypeDef is defined in the stm32n6xx_ll_gpio.h

Data Fields

- ***uint32_t Pin***
- ***uint32_t Mode***
- ***uint32_t Speed***
- ***uint32_t OutputType***
- ***uint32_t Pull***
- ***uint32_t Alternate***

Field Documentation

- ***uint32_t LL_GPIO_InitTypeDef::Pin***
  Specifies the GPIO pins to be configured. This parameter can be any value of ***GPIO_LL_EC_PIN***
- ***uint32_t LL_GPIO_InitTypeDef::Mode***
  Specifies the operating mode for the selected pins. This parameter can be a value of ***GPIO_LL_EC_MODE***.GPIO HW configuration can be modified afterwards using unitary function **LL_GPIO_SetPinMode()**.
- ***uint32_t LL_GPIO_InitTypeDef::Speed***
  Specifies the speed for the selected pins. This parameter can be a value of ***GPIO_LL_EC_SPEED***.GPIO HW configuration can be modified afterwards using unitary function **LL_GPIO_SetPinSpeed()**.
- ***uint32_t LL_GPIO_InitTypeDef::OutputType***
  Specifies the operating output type for the selected pins. This parameter can be a value of ***GPIO_LL_EC_OUTPUT***.GPIO HW configuration can be modified afterwards using unitary function **LL_GPIO_SetPinOutputType()**.
- ***uint32_t LL_GPIO_InitTypeDef::Pull***
  Specifies the operating Pull-up/Pull down for the selected pins. This parameter can be a value of ***GPIO_LL_EC_PULL***.GPIO HW configuration can be modified afterwards using unitary function **LL_GPIO_SetPinPull()**.
- ***uint32_t LL_GPIO_InitTypeDef::Alternate***
  Specifies the Peripheral to be connected to the selected pins. This parameter can be a value of ***GPIO_LL_EC_AF***.GPIO HW configuration can be modified afterwards using unitary function **LL_GPIO_SetAFPin_0_7()** and **LL_GPIO_SetAFPin_8_15()**.

93.2 GPIO Firmware driver API description

The following section lists the various functions of the GPIO library.

93.2.1 Detailed description of functions

LL_GPIO_SetPinMode

Function name

__STATIC_INLINE void LL_GPIO_SetPinMode (GPIO_TypeDef * GPIOx, uint32_t Pin, uint32_t Mode)

Function description

Configure gpio mode for a dedicated pin on dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **Pin:** This parameter can be one of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
- **Mode:** This parameter can be one of the following values:
  - LL_GPIO_MODE_INPUT
  - LL_GPIO_MODE_OUTPUT
  - LL_GPIO_MODE_ALTERNATE
  - LL_GPIO_MODE_ANALOG

Return values

- **None:**

Notes

- I/O mode can be Input mode, General purpose output, Alternate function mode or Analog.
- Warning: only one pin can be passed as parameter.

Reference Manual to LL API cross reference:

- MODER MODEy LL_GPIO_SetPinMode

LL_GPIO_GetPinMode

Function name

__STATIC_INLINE uint32_t LL_GPIO_GetPinMode (const GPIO_TypeDef * GPIOx, uint32_t Pin)

Function description

Return gpio mode for a dedicated pin on dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **Pin:** This parameter can be one of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15

Return values

- **Returned:** value can be one of the following values:
  - LL_GPIO_MODE_INPUT
  - LL_GPIO_MODE_OUTPUT
  - LL_GPIO_MODE_ALTERNATE
  - LL_GPIO_MODE_ANALOG

Notes

- I/O mode can be Input mode, General purpose output, Alternate function mode or Analog.
- Warning: only one pin can be passed as parameter.

Reference Manual to LL API cross reference:

- MODER MODEy LL_GPIO_GetPinMode

LL_GPIO_SetPinOutputType

Function name

__STATIC_INLINE void LL_GPIO_SetPinOutputType (GPIO_TypeDef * GPIOx, uint32_t PinMask, uint32_t OutputType)

Function description

Configure gpio output type for several pins on dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **PinMask:** This parameter can be a combination of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
  - LL_GPIO_PIN_ALL
- **OutputType:** This parameter can be one of the following values:
  - LL_GPIO_OUTPUT_PUSHPULL
  - LL_GPIO_OUTPUT_OPENDRAIN

Return values

- **None:**

Notes

- Output type as to be set when gpio pin is in output or alternate modes. Possible type are Push-pull or Open-drain.

Reference Manual to LL API cross reference:

- OTYPER OTy LL_GPIO_SetPinOutputType

LL_GPIO_GetPinOutputType

Function name

__STATIC_INLINE uint32_t LL_GPIO_GetPinOutputType (const GPIO_TypeDef * GPIOx, uint32_t Pin)

Function description

Return gpio output type for several pins on dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **Pin:** This parameter can be one of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
  - LL_GPIO_PIN_ALL

Return values

- **Returned:** value can be one of the following values:
  - LL_GPIO_OUTPUT_PUSHPULL
  - LL_GPIO_OUTPUT_OPENDRAIN

Notes

- Output type as to be set when gpio pin is in output or alternate modes. Possible type are Push-pull or Open-drain.
- Warning: only one pin can be passed as parameter.

Reference Manual to LL API cross reference:

- OTYPER OTy LL_GPIO_GetPinOutputType

LL_GPIO_SetPinSpeed

Function name

__STATIC_INLINE void LL_GPIO_SetPinSpeed (GPIO_TypeDef * GPIOx, uint32_t Pin, uint32_t Speed)

Function description

Configure gpio speed for a dedicated pin on dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **Pin:** This parameter can be one of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
- **Speed:** This parameter can be one of the following values:
  - LL_GPIO_SPEED_FREQ_LOW
  - LL_GPIO_SPEED_FREQ_MEDIUM
  - LL_GPIO_SPEED_FREQ_HIGH
  - LL_GPIO_SPEED_FREQ_VERY_HIGH

Return values

- **None:**

Notes

- I/O speed can be Low, Medium, Fast or High speed.
- Warning: only one pin can be passed as parameter.
- Refer to datasheet for frequency specifications and the power supply and load conditions for each speed.

Reference Manual to LL API cross reference:

- OSPEEDR OSPEEDy LL_GPIO_SetPinSpeed

LL_GPIO_GetPinSpeed

Function name

__STATIC_INLINE uint32_t LL_GPIO_GetPinSpeed (const GPIO_TypeDef * GPIOx, uint32_t Pin)

Function description

Return gpio speed for a dedicated pin on dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **Pin:** This parameter can be one of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15

Return values

- **Returned:** value can be one of the following values:
  - LL_GPIO_SPEED_FREQ_LOW
  - LL_GPIO_SPEED_FREQ_MEDIUM
  - LL_GPIO_SPEED_FREQ_HIGH
  - LL_GPIO_SPEED_FREQ_VERY_HIGH

Notes

- I/O speed can be Low, Medium, Fast or High speed.
- Warning: only one pin can be passed as parameter.
- Refer to datasheet for frequency specifications and the power supply and load conditions for each speed.

Reference Manual to LL API cross reference:

- OSPEEDR OSPEEDy LL_GPIO_GetPinSpeed

LL_GPIO_SetPinPull

Function name

__STATIC_INLINE void LL_GPIO_SetPinPull (GPIO_TypeDef * GPIOx, uint32_t Pin, uint32_t Pull)

Function description

Configure gpio pull-up or pull-down for a dedicated pin on a dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **Pin:** This parameter can be one of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
- **Pull:** This parameter can be one of the following values:
  - LL_GPIO_PULL_NO
  - LL_GPIO_PULL_UP
  - LL_GPIO_PULL_DOWN

Return values

- **None:**

Notes

- Warning: only one pin can be passed as parameter.

Reference Manual to LL API cross reference:

- PUPDR PUPDy LL_GPIO_SetPinPull

LL_GPIO_GetPinPull

Function name

__STATIC_INLINE uint32_t LL_GPIO_GetPinPull (const GPIO_TypeDef * GPIOx, uint32_t Pin)

Function description

Return gpio pull-up or pull-down for a dedicated pin on a dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **Pin:** This parameter can be one of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15

Return values

- **Returned:** value can be one of the following values:
  - LL_GPIO_PULL_NO
  - LL_GPIO_PULL_UP
  - LL_GPIO_PULL_DOWN

Notes

- Warning: only one pin can be passed as parameter.

Reference Manual to LL API cross reference:

- PUPDR PUPDy LL_GPIO_GetPinPull

LL_GPIO_SetAFPin_0_7

Function name

__STATIC_INLINE void LL_GPIO_SetAFPin_0_7 (GPIO_TypeDef * GPIOx, uint32_t Pin, uint32_t Alternate)

Function description

Configure gpio alternate function of a dedicated pin from 0 to 7 for a dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **Pin:** This parameter can be one of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
- **Alternate:** This parameter can be one of the following values:
  - LL_GPIO_AF_0
  - LL_GPIO_AF_1
  - LL_GPIO_AF_2
  - LL_GPIO_AF_3
  - LL_GPIO_AF_4
  - LL_GPIO_AF_5
  - LL_GPIO_AF_6
  - LL_GPIO_AF_7
  - LL_GPIO_AF_8
  - LL_GPIO_AF_9
  - LL_GPIO_AF_10
  - LL_GPIO_AF_11
  - LL_GPIO_AF_12
  - LL_GPIO_AF_13
  - LL_GPIO_AF_14
  - LL_GPIO_AF_15

Return values

- **None:**

Notes

- Possible values are from AF0 to AF15 depending on target.
- Warning: only one pin can be passed as parameter.

Reference Manual to LL API cross reference:

- AFRL AFSELy LL_GPIO_SetAFPin_0_7

LL_GPIO_GetAFPin_0_7

Function name

__STATIC_INLINE uint32_t LL_GPIO_GetAFPin_0_7 (const GPIO_TypeDef * GPIOx, uint32_t Pin)

Function description

Return gpio alternate function of a dedicated pin from 0 to 7 for a dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **Pin:** This parameter can be one of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7

Return values

- **Returned:** value can be one of the following values:
  - LL_GPIO_AF_0
  - LL_GPIO_AF_1
  - LL_GPIO_AF_2
  - LL_GPIO_AF_3
  - LL_GPIO_AF_4
  - LL_GPIO_AF_5
  - LL_GPIO_AF_6
  - LL_GPIO_AF_7
  - LL_GPIO_AF_8
  - LL_GPIO_AF_9
  - LL_GPIO_AF_10
  - LL_GPIO_AF_11
  - LL_GPIO_AF_12
  - LL_GPIO_AF_13
  - LL_GPIO_AF_14
  - LL_GPIO_AF_15

Reference Manual to LL API cross reference:

- AFRL AFSELy LL_GPIO_GetAFPin_0_7

LL_GPIO_SetAFPin_8_15

Function name

__STATIC_INLINE void LL_GPIO_SetAFPin_8_15 (GPIO_TypeDef * GPIOx, uint32_t Pin, uint32_t Alternate)

Function description

Configure gpio alternate function of a dedicated pin from 8 to 15 for a dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **Pin:** This parameter can be one of the following values:
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
- **Alternate:** This parameter can be one of the following values:
  - LL_GPIO_AF_0
  - LL_GPIO_AF_1
  - LL_GPIO_AF_2
  - LL_GPIO_AF_3
  - LL_GPIO_AF_4
  - LL_GPIO_AF_5
  - LL_GPIO_AF_6
  - LL_GPIO_AF_7
  - LL_GPIO_AF_8
  - LL_GPIO_AF_9
  - LL_GPIO_AF_10
  - LL_GPIO_AF_11
  - LL_GPIO_AF_12
  - LL_GPIO_AF_13
  - LL_GPIO_AF_14
  - LL_GPIO_AF_15

Return values

- **None:**

Notes

- Possible values are from AF0 to AF15 depending on target.
- Warning: only one pin can be passed as parameter.

Reference Manual to LL API cross reference:

- AFRH AFSELy LL_GPIO_SetAFPin_8_15

LL_GPIO_GetAFPin_8_15

Function name

__STATIC_INLINE uint32_t LL_GPIO_GetAFPin_8_15 (const GPIO_TypeDef * GPIOx, uint32_t Pin)

Function description

Return gpio alternate function of a dedicated pin from 8 to 15 for a dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **Pin:** This parameter can be one of the following values:
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15

Return values

- **Returned:** value can be one of the following values:
  - LL_GPIO_AF_0
  - LL_GPIO_AF_1
  - LL_GPIO_AF_2
  - LL_GPIO_AF_3
  - LL_GPIO_AF_4
  - LL_GPIO_AF_5
  - LL_GPIO_AF_6
  - LL_GPIO_AF_7
  - LL_GPIO_AF_8
  - LL_GPIO_AF_9
  - LL_GPIO_AF_10
  - LL_GPIO_AF_11
  - LL_GPIO_AF_12
  - LL_GPIO_AF_13
  - LL_GPIO_AF_14
  - LL_GPIO_AF_15

Notes

- Possible values are from AF0 to AF15 depending on target.

Reference Manual to LL API cross reference:

- AFRH AFSELy LL_GPIO_GetAFPin_8_15

LL_GPIO_LockPin

Function name

__STATIC_INLINE void LL_GPIO_LockPin (GPIO_TypeDef * GPIOx, uint32_t PinMask)

Function description

Lock configuration of several pins for a dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **PinMask:** This parameter can be a combination of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
  - LL_GPIO_PIN_ALL

Return values

- **None:**

Notes

- When the lock sequence has been applied on a port bit, the value of this port bit can no longer be modified until the next reset.
- Each lock bit freezes a specific configuration register (control and alternate function registers).

Reference Manual to LL API cross reference:

- LCKR LCKK LL_GPIO_LockPin

LL_GPIO_IsPinLocked

Function name

__STATIC_INLINE uint32_t LL_GPIO_IsPinLocked (const GPIO_TypeDef * GPIOx, uint32_t PinMask)

Function description

Return 1 if all pins passed as parameter, of a dedicated port, are locked.

Parameters

- **GPIOx:** GPIO Port
- **PinMask:** This parameter can be a combination of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
  - LL_GPIO_PIN_ALL

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- LCKR LCKy LL_GPIO_IsPinLocked

LL_GPIO_IsAnyPinLocked

Function name

__STATIC_INLINE uint32_t LL_GPIO_IsAnyPinLocked (const GPIO_TypeDef * GPIOx)

Function description

Return 1 if one of the pin of a dedicated port is locked.

Parameters

- **GPIOx:** GPIO Port

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- LCKR LCKK LL_GPIO_IsAnyPinLocked

LL_GPIO_LockResourcePin

Function name

__STATIC_INLINE void LL_GPIO_LockResourcePin (GPIO_TypeDef * GPIOx, uint32_t PinMask)

Function description

Lock resource configuration of several pins for a dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **PinMask:** This parameter can be a combination of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
  - LL_GPIO_PIN_ALL

Return values

- **None:**

Notes

- When the lock sequence has been applied on a port bit, the value of this port bit can no longer be modified until the next reset.
- Each lock bit freezes a security configuration register (control and alternate function registers).

Reference Manual to LL API cross reference:

- LCKR LCKK LL_GPIO_LockPin

LL_GPIO_IsResourcePinLocked

Function name

__STATIC_INLINE uint32_t LL_GPIO_IsResourcePinLocked (const GPIO_TypeDef * GPIOx, uint32_t PinMask)

Function description

Return 1 if all resource pins passed as parameter, of a dedicated port, are locked.

Parameters

- **GPIOx:** GPIO Port
- **PinMask:** This parameter can be a combination of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
  - LL_GPIO_PIN_ALL

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- LCKR LCKy LL_GPIO_IsPinLocked

LL_GPIO_IsAnyResourcePinLocked

Function name

__STATIC_INLINE uint32_t LL_GPIO_IsAnyResourcePinLocked (const GPIO_TypeDef * GPIOx)

Function description

Return 1 if one of the resource pin of a dedicated port is locked.

Parameters

- **GPIOx:** GPIO Port

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- LCKR LCKK LL_GPIO_IsAnyPinLocked

LL_GPIO_SetDelayPin_0_7

Function name

__STATIC_INLINE void LL_GPIO_SetDelayPin_0_7 (GPIO_TypeDef * GPIOx, uint32_t Pin, uint32_t Delay)

Function description

Configure gpio delay of a dedicated pin from 0 to 7 for a dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **Pin:** This parameter can be one of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
- **Delay:** This parameter can be one of the following values:
  - LL_GPIO_DELAY_0
  - LL_GPIO_DELAY_300
  - LL_GPIO_DELAY_500
  - LL_GPIO_DELAY_750
  - LL_GPIO_DELAY_1000
  - LL_GPIO_DELAY_1250
  - LL_GPIO_DELAY_1500
  - LL_GPIO_DELAY_1750
  - LL_GPIO_DELAY_2000
  - LL_GPIO_DELAY_2250
  - LL_GPIO_DELAY_2500
  - LL_GPIO_DELAY_2750
  - LL_GPIO_DELAY_3000
  - LL_GPIO_DELAY_3250

Return values

- **None:**

Notes

- Warning: only one pin can be passed as parameter.

LL_GPIO_GetDelayPin_0_7

Function name

__STATIC_INLINE uint32_t LL_GPIO_GetDelayPin_0_7 (const GPIO_TypeDef * GPIOx, uint32_t Pin)

Function description

Return gpio delay of a dedicated pin from 0 to 7 for a dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **Pin:** This parameter can be one of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7

Return values

- **Returned:** value can be one of the following values:
  - LL_GPIO_DELAY_0
  - LL_GPIO_DELAY_300
  - LL_GPIO_DELAY_500
  - LL_GPIO_DELAY_750
  - LL_GPIO_DELAY_1000
  - LL_GPIO_DELAY_1250
  - LL_GPIO_DELAY_1500
  - LL_GPIO_DELAY_1750
  - LL_GPIO_DELAY_2000
  - LL_GPIO_DELAY_2250
  - LL_GPIO_DELAY_2500
  - LL_GPIO_DELAY_2750
  - LL_GPIO_DELAY_3000
  - LL_GPIO_DELAY_3250

LL_GPIO_SetDelayPin_8_15

Function name

__STATIC_INLINE void LL_GPIO_SetDelayPin_8_15 (GPIO_TypeDef * GPIOx, uint32_t Pin, uint32_t Delay)

Function description

Configure gpio delay of a dedicated pin from 8 to 15 for a dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **Pin:** This parameter can be one of the following values:
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
- **Delay:** This parameter can be one of the following values:
  - LL_GPIO_DELAY_0
  - LL_GPIO_DELAY_300
  - LL_GPIO_DELAY_500
  - LL_GPIO_DELAY_750
  - LL_GPIO_DELAY_1000
  - LL_GPIO_DELAY_1250
  - LL_GPIO_DELAY_1500
  - LL_GPIO_DELAY_1750
  - LL_GPIO_DELAY_2000
  - LL_GPIO_DELAY_2250
  - LL_GPIO_DELAY_2500
  - LL_GPIO_DELAY_2750
  - LL_GPIO_DELAY_3000
  - LL_GPIO_DELAY_3250

Return values

- **None:**

Notes

- Warning: only one pin can be passed as parameter.

LL_GPIO_GetDelayPin_8_15

Function name

__STATIC_INLINE uint32_t LL_GPIO_GetDelayPin_8_15 (const GPIO_TypeDef * GPIOx, uint32_t Pin)

Function description

Return gpio delay of a dedicated pin from 8 to 15 for a dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **Pin:** This parameter can be one of the following values:
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15

Return values

- **Returned:** value can be one of the following values:
  - LL_GPIO_DELAY_0
  - LL_GPIO_DELAY_300
  - LL_GPIO_DELAY_500
  - LL_GPIO_DELAY_750
  - LL_GPIO_DELAY_1000
  - LL_GPIO_DELAY_1250
  - LL_GPIO_DELAY_1500
  - LL_GPIO_DELAY_1750
  - LL_GPIO_DELAY_2000
  - LL_GPIO_DELAY_2250
  - LL_GPIO_DELAY_2500
  - LL_GPIO_DELAY_2750
  - LL_GPIO_DELAY_3000
  - LL_GPIO_DELAY_3250

LL_GPIO_SetPIOControlPin_0_7

Function name

__STATIC_INLINE void LL_GPIO_SetPIOControlPin_0_7 (GPIO_TypeDef * GPIOx, uint32_t Pin, uint32_t CfgMask)

Function description

Configure gpio delay of a dedicated pin from 0 to 7 for a dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **Pin:** This parameter can be one of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
- **CfgMask:** ADVCFG[0] : Delay on ouptput path ADVCFG[1] : input and ouptput data are double edge ADVCFG[2] : I/O clocks are inverted ADVCFG[3] : input and ouptput data are retimed

Return values

- **None:**

Notes

- Warning: only one pin can be passed as parameter.

LL_GPIO_GetPIOControlPin_0_7

Function name

__STATIC_INLINE uint32_t LL_GPIO_GetPIOControlPin_0_7 (const GPIO_TypeDef * GPIOx, uint32_t Pin)

Function description

Return gpio delay of a dedicated pin from 0 to 7 for a dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **Pin:** This parameter can be one of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7

Return values

- **Returned:** value can be one of the following values: ADVCFG[0] : Delay on ouptput path ADVCFG[1] : input and ouptput data are double edge ADVCFG[2] : I/O clocks are inverted ADVCFG[3] : input and ouptput data are retimed

LL_GPIO_SetPIOControlPin_8_15

Function name

__STATIC_INLINE void LL_GPIO_SetPIOControlPin_8_15 (GPIO_TypeDef * GPIOx, uint32_t Pin, uint32_t CfgMask)

Function description

Configure gpio delay of a dedicated pin from 8 to 15 for a dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **Pin:** This parameter can be one of the following values:
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
- **CfgMask:** ADVCFG[0] : Delay on ouptput path ADVCFG[1] : input and ouptput data are double edge ADVCFG[2] : I/O clocks are inverted ADVCFG[3] : input and ouptput data are retimed

Return values

- **None:**

Notes

- Warning: only one pin can be passed as parameter.

LL_GPIO_GetPIOControlPin_8_15

Function name

__STATIC_INLINE uint32_t LL_GPIO_GetPIOControlPin_8_15 (const GPIO_TypeDef * GPIOx, uint32_t Pin)

Function description

Return gpio delay of a dedicated pin from 8 to 15 for a dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **Pin:** This parameter can be one of the following values:
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15

Return values

- **Returned:** value can be one of the following values: ADVCFG[0] : Delay on ouptput path ADVCFG[1] : input and ouptput data are double edge ADVCFG[2] : I/O clocks are inverted ADVCFG[3] : input and ouptput data are retimed

LL_GPIO_ReadInputPort

Function name

__STATIC_INLINE uint32_t LL_GPIO_ReadInputPort (const GPIO_TypeDef * GPIOx)

Function description

Return full input data register value for a dedicated port.

Parameters

- **GPIOx:** GPIO Port

Return values

- **Input:** data register value of port

Reference Manual to LL API cross reference:

- IDR IDy LL_GPIO_ReadInputPort

LL_GPIO_IsInputPinSet

Function name

__STATIC_INLINE uint32_t LL_GPIO_IsInputPinSet (const GPIO_TypeDef * GPIOx, uint32_t PinMask)

Function description

Return if input data level for several pins of dedicated port is high or low.

Parameters

- **GPIOx:** GPIO Port
- **PinMask:** This parameter can be a combination of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
  - LL_GPIO_PIN_ALL

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IDR IDy LL_GPIO_IsInputPinSet

LL_GPIO_WriteOutputPort

Function name

__STATIC_INLINE void LL_GPIO_WriteOutputPort (GPIO_TypeDef * GPIOx, uint32_t PortValue)

Function description

Write output data register for the port.

Parameters

- **GPIOx:** GPIO Port
- **PortValue:** Level value for each pin of the port

Return values

- **None:**

Reference Manual to LL API cross reference:

- ODR ODy LL_GPIO_WriteOutputPort

LL_GPIO_ReadOutputPort

Function name

__STATIC_INLINE uint32_t LL_GPIO_ReadOutputPort (const GPIO_TypeDef * GPIOx)

Function description

Return full output data register value for a dedicated port.

Parameters

- **GPIOx:** GPIO Port

Return values

- **Output:** data register value of port

Reference Manual to LL API cross reference:

- ODR ODy LL_GPIO_ReadOutputPort

LL_GPIO_IsOutputPinSet

Function name

__STATIC_INLINE uint32_t LL_GPIO_IsOutputPinSet (const GPIO_TypeDef * GPIOx, uint32_t PinMask)

Function description

Return if input data level for several pins of dedicated port is high or low.

Parameters

- **GPIOx:** GPIO Port
- **PinMask:** This parameter can be a combination of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
  - LL_GPIO_PIN_ALL

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ODR ODy LL_GPIO_IsOutputPinSet

LL_GPIO_SetOutputPin

Function name

__STATIC_INLINE void LL_GPIO_SetOutputPin (GPIO_TypeDef * GPIOx, uint32_t PinMask)

Function description

Set several pins to high level on dedicated gpio port.

Parameters

- **GPIOx:** GPIO Port
- **PinMask:** This parameter can be a combination of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
  - LL_GPIO_PIN_ALL

Return values

- **None:**

Reference Manual to LL API cross reference:

- BSRR BSy LL_GPIO_SetOutputPin

LL_GPIO_ResetOutputPin

Function name

__STATIC_INLINE void LL_GPIO_ResetOutputPin (GPIO_TypeDef * GPIOx, uint32_t PinMask)

Function description

Set several pins to low level on dedicated gpio port.

Parameters

- **GPIOx:** GPIO Port
- **PinMask:** This parameter can be a combination of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
  - LL_GPIO_PIN_ALL

Return values

- **None:**

Reference Manual to LL API cross reference:

- BRR BRy LL_GPIO_ResetOutputPin

LL_GPIO_TogglePin

Function name

__STATIC_INLINE void LL_GPIO_TogglePin (GPIO_TypeDef * GPIOx, uint32_t PinMask)

Function description

Toggle data value for several pin of dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **PinMask:** This parameter can be a combination of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
  - LL_GPIO_PIN_ALL

Return values

- **None:**

Reference Manual to LL API cross reference:

- ODR ODy LL_GPIO_TogglePin

LL_GPIO_EnablePinSecure

Function name

__STATIC_INLINE void LL_GPIO_EnablePinSecure (GPIO_TypeDef * GPIOx, uint32_t PinMask)

Function description

Enable secure write only access for several pin of dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **PinMask:** This parameter can be a combination of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
  - LL_GPIO_PIN_ALL

Return values

- **None:**

Reference Manual to LL API cross reference:

- SECCFGR SECy LL_GPIO_EnablePinSecure

LL_GPIO_DisablePinSecure

Function name

__STATIC_INLINE void LL_GPIO_DisablePinSecure (GPIO_TypeDef * GPIOx, uint32_t PinMask)

Function description

Disable secure write only access for several pin of dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **PinMask:** This parameter can be a combination of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
  - LL_GPIO_PIN_ALL

Return values

- **None:**

Reference Manual to LL API cross reference:

- SECCFGR SECy LL_GPIO_DisablePinSecure

LL_GPIO_IsEnabledPinSecure

Function name

__STATIC_INLINE uint32_t LL_GPIO_IsEnabledPinSecure (const GPIO_TypeDef * GPIOx, uint32_t PinMask)

Function description

Return if secure write only access for several pin of dedicated port is enabled or not.

Parameters

- **GPIOx:** GPIO Port
- **PinMask:** This parameter can be a combination of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
  - LL_GPIO_PIN_ALL

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SECCFGR SECy LL_GPIO_IsEnabledPinSecure

LL_GPIO_EnablePinPrivilege

Function name

__STATIC_INLINE void LL_GPIO_EnablePinPrivilege (GPIO_TypeDef * GPIOx, uint32_t PinMask)

Function description

Enable secure write only access for several pin of dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **PinMask:** This parameter can be a combination of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
  - LL_GPIO_PIN_ALL

Return values

- **None:**

Reference Manual to LL API cross reference:

- SECCFGR SECy LL_GPIO_EnablePinPrivilege

LL_GPIO_DisablePinPrivilege

Function name

__STATIC_INLINE void LL_GPIO_DisablePinPrivilege (GPIO_TypeDef * GPIOx, uint32_t PinMask)

Function description

Disable secure write only access for several pin of dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **PinMask:** This parameter can be a combination of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
  - LL_GPIO_PIN_ALL

Return values

- **None:**

Reference Manual to LL API cross reference:

- SECCFGR SECy LL_GPIO_DisablePinPrivilege

LL_GPIO_IsEnabledPinPrivilege

Function name

__STATIC_INLINE uint32_t LL_GPIO_IsEnabledPinPrivilege (const GPIO_TypeDef * GPIOx, uint32_t PinMask)

Function description

Return if privilege write only access for several pin of dedicated port is enabled or not.

Parameters

- **GPIOx:** GPIO Port
- **PinMask:** This parameter can be a combination of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
  - LL_GPIO_PIN_ALL

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SECCFGR SECy LL_GPIO_IsEnabledPinSecure

LL_GPIO_DeInit

Function name

ErrorStatus LL_GPIO_DeInit (const GPIO_TypeDef * GPIOx)

Function description

De-initialize GPIO registers (Registers restored to their default values).

Parameters

- **GPIOx:** GPIO Port

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: GPIO registers are de-initialized
  - ERROR: Wrong GPIO Port

LL_GPIO_Init

Function name

ErrorStatus LL_GPIO_Init (GPIO_TypeDef * GPIOx, const LL_GPIO_InitTypeDef * GPIO_InitStruct)

Function description

Initialize GPIO registers according to the specified parameters in GPIO_InitStruct.

Parameters

- **GPIOx:** GPIO Port
- **GPIO_InitStruct:** pointer to a LL_GPIO_InitTypeDef structure that contains the configuration information for the specified GPIO peripheral.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: GPIO registers are initialized according to GPIO_InitStruct content
  - ERROR: Not applicable

LL_GPIO_StructInit

Function name

void LL_GPIO_StructInit (LL_GPIO_InitTypeDef * GPIO_InitStruct)

Function description

Set each LL_GPIO_InitTypeDef field to default value.

Parameters

- **GPIO_InitStruct:** pointer to a LL_GPIO_InitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

93.3 GPIO Firmware driver defines

The following section lists the various define and macros of the module.

93.3.1 GPIO

GPIO

Alternate Function

LL_GPIO_AF_0

Select alternate function 0

LL_GPIO_AF_1

Select alternate function 1

LL_GPIO_AF_2

Select alternate function 2

LL_GPIO_AF_3

Select alternate function 3

LL_GPIO_AF_4

Select alternate function 4

LL_GPIO_AF_5

Select alternate function 5

LL_GPIO_AF_6

Select alternate function 6

LL_GPIO_AF_7

Select alternate function 7

LL_GPIO_AF_8

Select alternate function 8

LL_GPIO_AF_9

Select alternate function 9

LL_GPIO_AF_10

Select alternate function 10

LL_GPIO_AF_11

Select alternate function 11

LL_GPIO_AF_12

Select alternate function 12

LL_GPIO_AF_13

Select alternate function 13

LL_GPIO_AF_14

Select alternate function 14

LL_GPIO_AF_15

Select alternate function 15

Delay

LL_GPIO_DELAY_0

Select no delay

LL_GPIO_DELAY_300

Select 0.3 ns

LL_GPIO_DELAY_500

Select 0.5 ns

LL_GPIO_DELAY_750

Select 0.75 ns

LL_GPIO_DELAY_1000

Select 1 ns

LL_GPIO_DELAY_1250

Select 1.25 ns

LL_GPIO_DELAY_1500

Select 1.5 ns

LL_GPIO_DELAY_1750

Select 1.75 ns

LL_GPIO_DELAY_2000

Select 2 ns

LL_GPIO_DELAY_2250

Select 2.25 ns

LL_GPIO_DELAY_2500

Select 2.5 ns

LL_GPIO_DELAY_2750

Select 2.75 ns

LL_GPIO_DELAY_3000

Select 3 ns

LL_GPIO_DELAY_3250

Select 3.25 ns

Mode

LL_GPIO_MODE_INPUT

Select input mode

LL_GPIO_MODE_OUTPUT

Select output mode

LL_GPIO_MODE_ALTERNATE

Select alternate function mode

LL_GPIO_MODE_ANALOG

Select analog mode

Output Type

LL_GPIO_OUTPUT_PUSHPULL

Select push-pull as output type

LL_GPIO_OUTPUT_OPENDRAIN

Select open-drain as output type

PIN

LL_GPIO_PIN_0

Select pin 0

LL_GPIO_PIN_1

Select pin 1

LL_GPIO_PIN_2

Select pin 2

LL_GPIO_PIN_3

Select pin 3

LL_GPIO_PIN_4

Select pin 4

LL_GPIO_PIN_5

Select pin 5

LL_GPIO_PIN_6

Select pin 6

LL_GPIO_PIN_7

Select pin 7

LL_GPIO_PIN_8

Select pin 8

LL_GPIO_PIN_9

Select pin 9

LL_GPIO_PIN_10

Select pin 10

LL_GPIO_PIN_11

Select pin 11

LL_GPIO_PIN_12

Select pin 12

LL_GPIO_PIN_13

Select pin 13

LL_GPIO_PIN_14

Select pin 14

LL_GPIO_PIN_15

Select pin 15

LL_GPIO_PIN_ALL

Select all pins

Pull Up Pull Down

LL_GPIO_PULL_NO

Select I/O no pull

LL_GPIO_PULL_UP

Select I/O pull up

LL_GPIO_PULL_DOWN

Select I/O pull down

Output Speed

LL_GPIO_SPEED_FREQ_LOW

Select I/O low output speed

LL_GPIO_SPEED_FREQ_MEDIUM

Select I/O medium output speed

LL_GPIO_SPEED_FREQ_HIGH

Select I/O fast output speed

LL_GPIO_SPEED_FREQ_VERY_HIGH

Select I/O high output speed

Common Write and read registers Macros

LL_GPIO_WriteReg

Description:

- Write a value in GPIO register.

Parameters:

- __INSTANCE__: GPIO Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_GPIO_ReadReg

Description:

- Read a value in GPIO register.

Parameters:

- __INSTANCE__: GPIO Instance
- __REG__: Register to be read

Return value:

- Register: value

GPIO Exported Constants

LL_GPIO_SPEED_LOW

LL_GPIO_SPEED_MEDIUM

LL_GPIO_SPEED_FAST

LL_GPIO_SPEED_HIGH

94 LL I2C Generic Driver

94.1 I2C Firmware driver registers structures

94.1.1 LL_I2C_InitTypeDef

LL_I2C_InitTypeDef is defined in the stm32n6xx_ll_i2c.h

Data Fields

- ***uint32_t PeripheralMode***
- ***uint32_t Timing***
- ***uint32_t AnalogFilter***
- ***uint32_t DigitalFilter***
- ***uint32_t OwnAddress1***
- ***uint32_t TypeAcknowledge***
- ***uint32_t OwnAddrSize***

Field Documentation

- ***uint32_t LL_I2C_InitTypeDef::PeripheralMode***
  Specifies the peripheral mode. This parameter can be a value of ***I2C_LL_EC_PERIPHERAL_MODE***.This feature can be modified afterwards using unitary function **LL_I2C_SetMode()**.
- ***uint32_t LL_I2C_InitTypeDef::Timing***
  Specifies the SDA setup, hold time and the SCL high, low period values. This parameter must be set by referring to the STM32CubeMX Tool and the helper macro **__LL_I2C_CONVERT_TIMINGS()**.This feature can be modified afterwards using unitary function **LL_I2C_SetTiming()**.
- ***uint32_t LL_I2C_InitTypeDef::AnalogFilter***
  Enables or disables analog noise filter. This parameter can be a value of ***I2C_LL_EC_ANALOGFILTER_SELECTION***.This feature can be modified afterwards using unitary functions **LL_I2C_EnableAnalogFilter()** or **LL_I2C_DisableAnalogFilter()**.
- ***uint32_t LL_I2C_InitTypeDef::DigitalFilter***
  Configures the digital noise filter. This parameter can be a number between Min_Data = 0x00 and Max_Data = 0x0F.This feature can be modified afterwards using unitary function **LL_I2C_SetDigitalFilter()**.
- ***uint32_t LL_I2C_InitTypeDef::OwnAddress1***
  Specifies the device own address 1. This parameter must be a value between Min_Data = 0x00 and Max_Data = 0x3FF.This feature can be modified afterwards using unitary function **LL_I2C_SetOwnAddress1()**.
- ***uint32_t LL_I2C_InitTypeDef::TypeAcknowledge***
  Specifies the ACKnowledge or Non ACKnowledge condition after the address receive match code or next received byte. This parameter can be a value of ***I2C_LL_EC_I2C_ACKNOWLEDGE***.This feature can be modified afterwards using unitary function **LL_I2C_AcknowledgeNextData()**.
- ***uint32_t LL_I2C_InitTypeDef::OwnAddrSize***
  Specifies the device own address 1 size (7-bit or 10-bit). This parameter can be a value of ***I2C_LL_EC_OWNADDRESS1***.This feature can be modified afterwards using unitary function **LL_I2C_SetOwnAddress1()**.

94.2 I2C Firmware driver API description

The following section lists the various functions of the I2C library.

94.2.1 Detailed description of functions

LL_I2C_Enable

Function name

__STATIC_INLINE void LL_I2C_Enable (I2C_TypeDef * I2Cx)

Function description

Enable I2C peripheral (PE = 1).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 PE LL_I2C_Enable

LL_I2C_Disable

Function name

__STATIC_INLINE void LL_I2C_Disable (I2C_TypeDef * I2Cx)

Function description

Disable I2C peripheral (PE = 0).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- When PE = 0, the I2C SCL and SDA lines are released. Internal state machines and status bits are put back to their reset value. When cleared, PE must be kept low for at least 3 APB clock cycles.

Reference Manual to LL API cross reference:

- CR1 PE LL_I2C_Disable

LL_I2C_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabled (const I2C_TypeDef * I2Cx)

Function description

Check if the I2C peripheral is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 PE LL_I2C_IsEnabled

LL_I2C_ConfigFilters

Function name

__STATIC_INLINE void LL_I2C_ConfigFilters (I2C_TypeDef * I2Cx, uint32_t AnalogFilter, uint32_t DigitalFilter)

Function description

Configure Noise Filters (Analog and Digital).

Parameters

- **I2Cx:** I2C Instance.
- **AnalogFilter:** This parameter can be one of the following values:
  - LL_I2C_ANALOGFILTER_ENABLE
  - LL_I2C_ANALOGFILTER_DISABLE
- **DigitalFilter:** This parameter must be a value between Min_Data=0x00 (Digital filter disabled) and Max_Data=0x0F (Digital filter enabled and filtering capability up to 15*ti2cclk). This parameter is used to configure the digital noise filter on SDA and SCL input. The digital filter will filter spikes with a length of up to DNF[3:0]*ti2cclk.

Return values

- **None:**

Notes

- If the analog filter is also enabled, the digital filter is added to analog filter. The filters can only be programmed when the I2C is disabled (PE = 0).

Reference Manual to LL API cross reference:

- CR1 ANFOFF LL_I2C_ConfigFilters
- CR1 DNF LL_I2C_ConfigFilters

LL_I2C_SetDigitalFilter

Function name

__STATIC_INLINE void LL_I2C_SetDigitalFilter (I2C_TypeDef * I2Cx, uint32_t DigitalFilter)

Function description

Configure Digital Noise Filter.

Parameters

- **I2Cx:** I2C Instance.
- **DigitalFilter:** This parameter must be a value between Min_Data=0x00 (Digital filter disabled) and Max_Data=0x0F (Digital filter enabled and filtering capability up to 15*ti2cclk). This parameter is used to configure the digital noise filter on SDA and SCL input. The digital filter will filter spikes with a length of up to DNF[3:0]*ti2cclk.

Return values

- **None:**

Notes

- If the analog filter is also enabled, the digital filter is added to analog filter. This filter can only be programmed when the I2C is disabled (PE = 0).

Reference Manual to LL API cross reference:

- CR1 DNF LL_I2C_SetDigitalFilter

LL_I2C_GetDigitalFilter

Function name

__STATIC_INLINE uint32_t LL_I2C_GetDigitalFilter (const I2C_TypeDef * I2Cx)

Function description

Get the current Digital Noise Filter configuration.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Value:** between Min_Data=0x0 and Max_Data=0xF

Reference Manual to LL API cross reference:

- CR1 DNF LL_I2C_GetDigitalFilter

LL_I2C_EnableAnalogFilter

Function name

__STATIC_INLINE void LL_I2C_EnableAnalogFilter (I2C_TypeDef * I2Cx)

Function description

Enable Analog Noise Filter.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- This filter can only be programmed when the I2C is disabled (PE = 0).

Reference Manual to LL API cross reference:

- CR1 ANFOFF LL_I2C_EnableAnalogFilter

LL_I2C_DisableAnalogFilter

Function name

__STATIC_INLINE void LL_I2C_DisableAnalogFilter (I2C_TypeDef * I2Cx)

Function description

Disable Analog Noise Filter.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- This filter can only be programmed when the I2C is disabled (PE = 0).

Reference Manual to LL API cross reference:

- CR1 ANFOFF LL_I2C_DisableAnalogFilter

LL_I2C_IsEnabledAnalogFilter

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledAnalogFilter (const I2C_TypeDef * I2Cx)

Function description

Check if Analog Noise Filter is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 ANFOFF LL_I2C_IsEnabledAnalogFilter

LL_I2C_EnableDMAReq_TX

Function name

__STATIC_INLINE void LL_I2C_EnableDMAReq_TX (I2C_TypeDef * I2Cx)

Function description

Enable DMA transmission requests.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TXDMAEN LL_I2C_EnableDMAReq_TX

LL_I2C_DisableDMAReq_TX

Function name

__STATIC_INLINE void LL_I2C_DisableDMAReq_TX (I2C_TypeDef * I2Cx)

Function description

Disable DMA transmission requests.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TXDMAEN LL_I2C_DisableDMAReq_TX

LL_I2C_IsEnabledDMAReq_TX

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledDMAReq_TX (const I2C_TypeDef * I2Cx)

Function description

Check if DMA transmission requests are enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 TXDMAEN LL_I2C_IsEnabledDMAReq_TX

LL_I2C_EnableDMAReq_RX

Function name

__STATIC_INLINE void LL_I2C_EnableDMAReq_RX (I2C_TypeDef * I2Cx)

Function description

Enable DMA reception requests.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RXDMAEN LL_I2C_EnableDMAReq_RX

LL_I2C_DisableDMAReq_RX

Function name

__STATIC_INLINE void LL_I2C_DisableDMAReq_RX (I2C_TypeDef * I2Cx)

Function description

Disable DMA reception requests.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RXDMAEN LL_I2C_DisableDMAReq_RX

LL_I2C_IsEnabledDMAReq_RX

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledDMAReq_RX (const I2C_TypeDef * I2Cx)

Function description

Check if DMA reception requests are enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 RXDMAEN LL_I2C_IsEnabledDMAReq_RX

LL_I2C_DMA_GetRegAddr

Function name

__STATIC_INLINE uint32_t LL_I2C_DMA_GetRegAddr (const I2C_TypeDef * I2Cx, uint32_t Direction)

Function description

Get the data register address used for DMA transfer.

Parameters

- **I2Cx:** I2C Instance
- **Direction:** This parameter can be one of the following values:
  - LL_I2C_DMA_REG_DATA_TRANSMIT
  - LL_I2C_DMA_REG_DATA_RECEIVE

Return values

- **Address:** of data register

Reference Manual to LL API cross reference:

- TXDR TXDATA LL_I2C_DMA_GetRegAddr
- RXDR RXDATA LL_I2C_DMA_GetRegAddr

LL_I2C_EnableClockStretching

Function name

__STATIC_INLINE void LL_I2C_EnableClockStretching (I2C_TypeDef * I2Cx)

Function description

Enable Clock stretching.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- This bit can only be programmed when the I2C is disabled (PE = 0).

Reference Manual to LL API cross reference:

- CR1 NOSTRETCH LL_I2C_EnableClockStretching

LL_I2C_DisableClockStretching

Function name

__STATIC_INLINE void LL_I2C_DisableClockStretching (I2C_TypeDef * I2Cx)

Function description

Disable Clock stretching.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- This bit can only be programmed when the I2C is disabled (PE = 0).

Reference Manual to LL API cross reference:

- CR1 NOSTRETCH LL_I2C_DisableClockStretching

LL_I2C_IsEnabledClockStretching

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledClockStretching (const I2C_TypeDef * I2Cx)

Function description

Check if Clock stretching is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 NOSTRETCH LL_I2C_IsEnabledClockStretching

LL_I2C_EnableSlaveByteControl

Function name

__STATIC_INLINE void LL_I2C_EnableSlaveByteControl (I2C_TypeDef * I2Cx)

Function description

Enable hardware byte control in slave mode.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 SBC LL_I2C_EnableSlaveByteControl

LL_I2C_DisableSlaveByteControl

Function name

__STATIC_INLINE void LL_I2C_DisableSlaveByteControl (I2C_TypeDef * I2Cx)

Function description

Disable hardware byte control in slave mode.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 SBC LL_I2C_DisableSlaveByteControl

LL_I2C_IsEnabledSlaveByteControl

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledSlaveByteControl (const I2C_TypeDef * I2Cx)

Function description

Check if hardware byte control in slave mode is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 SBC LL_I2C_IsEnabledSlaveByteControl

LL_I2C_EnableWakeUpFromStop

Function name

__STATIC_INLINE void LL_I2C_EnableWakeUpFromStop (I2C_TypeDef * I2Cx)

Function description

Enable Wakeup from STOP.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- The macro IS_I2C_WAKEUP_FROMSTOP_INSTANCE(I2Cx) can be used to check whether or not WakeUpFromStop feature is supported by the I2Cx Instance.
- This bit can only be programmed when Digital Filter is disabled.

Reference Manual to LL API cross reference:

- CR1 WUPEN LL_I2C_EnableWakeUpFromStop

LL_I2C_DisableWakeUpFromStop

Function name

__STATIC_INLINE void LL_I2C_DisableWakeUpFromStop (I2C_TypeDef * I2Cx)

Function description

Disable Wakeup from STOP.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- The macro IS_I2C_WAKEUP_FROMSTOP_INSTANCE(I2Cx) can be used to check whether or not WakeUpFromStop feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- CR1 WUPEN LL_I2C_DisableWakeUpFromStop

LL_I2C_IsEnabledWakeUpFromStop

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledWakeUpFromStop (const I2C_TypeDef * I2Cx)

Function description

Check if Wakeup from STOP is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- The macro IS_I2C_WAKEUP_FROMSTOP_INSTANCE(I2Cx) can be used to check whether or not WakeUpFromStop feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- CR1 WUPEN LL_I2C_IsEnabledWakeUpFromStop

LL_I2C_EnableGeneralCall

Function name

__STATIC_INLINE void LL_I2C_EnableGeneralCall (I2C_TypeDef * I2Cx)

Function description

Enable General Call.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- When enabled the Address 0x00 is ACKed.

Reference Manual to LL API cross reference:

- CR1 GCEN LL_I2C_EnableGeneralCall

LL_I2C_DisableGeneralCall

Function name

__STATIC_INLINE void LL_I2C_DisableGeneralCall (I2C_TypeDef * I2Cx)

Function description

Disable General Call.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- When disabled the Address 0x00 is NACKed.

Reference Manual to LL API cross reference:

- CR1 GCEN LL_I2C_DisableGeneralCall

LL_I2C_IsEnabledGeneralCall

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledGeneralCall (const I2C_TypeDef * I2Cx)

Function description

Check if General Call is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 GCEN LL_I2C_IsEnabledGeneralCall

LL_I2C_EnableFastModePlus

Function name

__STATIC_INLINE void LL_I2C_EnableFastModePlus (I2C_TypeDef * I2Cx)

Function description

Enable I2C Fast Mode Plus (FMP = 1).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- 20mA I/O drive enable

Reference Manual to LL API cross reference:

- CR1 FMP LL_I2C_EnableFastModePlus

LL_I2C_DisableFastModePlus

Function name

__STATIC_INLINE void LL_I2C_DisableFastModePlus (I2C_TypeDef * I2Cx)

Function description

Disable I2C Fast Mode Plus (FMP = 0).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- 20mA I/O drive disable

Reference Manual to LL API cross reference:

- CR1 FMP LL_I2C_DisableFastModePlus

LL_I2C_IsEnabledFastModePlus

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledFastModePlus (const I2C_TypeDef * I2Cx)

Function description

Check if the I2C Fast Mode Plus is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 FMP LL_I2C_IsEnabledFastModePlus

LL_I2C_EnableAutoClearFlag_ADDR

Function name

__STATIC_INLINE void LL_I2C_EnableAutoClearFlag_ADDR (I2C_TypeDef * I2Cx)

Function description

Enable automatic clear of ADDR flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 ADDRACLR LL_I2C_EnableAutoClearFlag_ADDR

LL_I2C_DisableAutoClearFlag_ADDR

Function name

__STATIC_INLINE void LL_I2C_DisableAutoClearFlag_ADDR (I2C_TypeDef * I2Cx)

Function description

Disable automatic clear of ADDR flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 ADDRACLR LL_I2C_DisableAutoClearFlag_ADDR

LL_I2C_IsEnabledAutoClearFlag_ADDR

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledAutoClearFlag_ADDR (const I2C_TypeDef * I2Cx)

Function description

Check if the automatic clear of ADDR flag is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 ADDRACLR LL_I2C_IsEnabledAutoClearFlag_ADDR

LL_I2C_EnableAutoClearFlag_STOP

Function name

__STATIC_INLINE void LL_I2C_EnableAutoClearFlag_STOP (I2C_TypeDef * I2Cx)

Function description

Enable automatic clear of STOP flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 STOPFACLR LL_I2C_EnableAutoClearFlag_STOP

LL_I2C_DisableAutoClearFlag_STOP

Function name

__STATIC_INLINE void LL_I2C_DisableAutoClearFlag_STOP (I2C_TypeDef * I2Cx)

Function description

Disable automatic clear of STOP flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 STOPFACLR LL_I2C_DisableAutoClearFlag_STOP

LL_I2C_IsEnabledAutoClearFlag_STOP

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledAutoClearFlag_STOP (const I2C_TypeDef * I2Cx)

Function description

Check if the automatic clear of STOP flag is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 STOPFACLR LL_I2C_IsEnabledAutoClearFlag_STOP

LL_I2C_SetMasterAddressingMode

Function name

__STATIC_INLINE void LL_I2C_SetMasterAddressingMode (I2C_TypeDef * I2Cx, uint32_t AddressingMode)

Function description

Configure the Master to operate in 7-bit or 10-bit addressing mode.

Parameters

- **I2Cx:** I2C Instance.
- **AddressingMode:** This parameter can be one of the following values:
  - LL_I2C_ADDRESSING_MODE_7BIT
  - LL_I2C_ADDRESSING_MODE_10BIT

Return values

- **None:**

Notes

- Changing this bit is not allowed, when the START bit is set.

Reference Manual to LL API cross reference:

- CR2 ADD10 LL_I2C_SetMasterAddressingMode

LL_I2C_GetMasterAddressingMode

Function name

__STATIC_INLINE uint32_t LL_I2C_GetMasterAddressingMode (const I2C_TypeDef * I2Cx)

Function description

Get the Master addressing mode.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Returned:** value can be one of the following values:
  - LL_I2C_ADDRESSING_MODE_7BIT
  - LL_I2C_ADDRESSING_MODE_10BIT

Reference Manual to LL API cross reference:

- CR2 ADD10 LL_I2C_GetMasterAddressingMode

LL_I2C_SetOwnAddress1

Function name

__STATIC_INLINE void LL_I2C_SetOwnAddress1 (I2C_TypeDef * I2Cx, uint32_t OwnAddress1, uint32_t OwnAddrSize)

Function description

Set the Own Address1.

Parameters

- **I2Cx:** I2C Instance.
- **OwnAddress1:** This parameter must be a value between Min_Data=0 and Max_Data=0x3FF.
- **OwnAddrSize:** This parameter can be one of the following values:
  - LL_I2C_OWNADDRESS1_7BIT
  - LL_I2C_OWNADDRESS1_10BIT

Return values

- **None:**

Reference Manual to LL API cross reference:

- OAR1 OA1 LL_I2C_SetOwnAddress1
- OAR1 OA1MODE LL_I2C_SetOwnAddress1

LL_I2C_EnableOwnAddress1

Function name

__STATIC_INLINE void LL_I2C_EnableOwnAddress1 (I2C_TypeDef * I2Cx)

Function description

Enable acknowledge on Own Address1 match address.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- OAR1 OA1EN LL_I2C_EnableOwnAddress1

LL_I2C_DisableOwnAddress1

Function name

__STATIC_INLINE void LL_I2C_DisableOwnAddress1 (I2C_TypeDef * I2Cx)

Function description

Disable acknowledge on Own Address1 match address.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- OAR1 OA1EN LL_I2C_DisableOwnAddress1

LL_I2C_IsEnabledOwnAddress1

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledOwnAddress1 (const I2C_TypeDef * I2Cx)

Function description

Check if Own Address1 acknowledge is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- OAR1 OA1EN LL_I2C_IsEnabledOwnAddress1

LL_I2C_SetOwnAddress2

Function name

__STATIC_INLINE void LL_I2C_SetOwnAddress2 (I2C_TypeDef * I2Cx, uint32_t OwnAddress2, uint32_t OwnAddrMask)

Function description

Set the 7bits Own Address2.

Parameters

- **I2Cx:** I2C Instance.
- **OwnAddress2:** Value between Min_Data=0 and Max_Data=0x7F.
- **OwnAddrMask:** This parameter can be one of the following values:
  - LL_I2C_OWNADDRESS2_NOMASK
  - LL_I2C_OWNADDRESS2_MASK01
  - LL_I2C_OWNADDRESS2_MASK02
  - LL_I2C_OWNADDRESS2_MASK03
  - LL_I2C_OWNADDRESS2_MASK04
  - LL_I2C_OWNADDRESS2_MASK05
  - LL_I2C_OWNADDRESS2_MASK06
  - LL_I2C_OWNADDRESS2_MASK07

Return values

- **None:**

Notes

- This action has no effect if own address2 is enabled.

Reference Manual to LL API cross reference:

- OAR2 OA2 LL_I2C_SetOwnAddress2
- OAR2 OA2MSK LL_I2C_SetOwnAddress2

LL_I2C_EnableOwnAddress2

Function name

__STATIC_INLINE void LL_I2C_EnableOwnAddress2 (I2C_TypeDef * I2Cx)

Function description

Enable acknowledge on Own Address2 match address.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- OAR2 OA2EN LL_I2C_EnableOwnAddress2

LL_I2C_DisableOwnAddress2

Function name

__STATIC_INLINE void LL_I2C_DisableOwnAddress2 (I2C_TypeDef * I2Cx)

Function description

Disable acknowledge on Own Address2 match address.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- OAR2 OA2EN LL_I2C_DisableOwnAddress2

LL_I2C_IsEnabledOwnAddress2

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledOwnAddress2 (const I2C_TypeDef * I2Cx)

Function description

Check if Own Address1 acknowledge is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- OAR2 OA2EN LL_I2C_IsEnabledOwnAddress2

LL_I2C_SetTiming

Function name

__STATIC_INLINE void LL_I2C_SetTiming (I2C_TypeDef * I2Cx, uint32_t Timing)

Function description

Configure the SDA setup, hold time and the SCL high, low period.

Parameters

- **I2Cx:** I2C Instance.
- **Timing:** This parameter must be a value between Min_Data=0 and Max_Data=0xFFFFFFFF.

Return values

- **None:**

Notes

- This bit can only be programmed when the I2C is disabled (PE = 0).
- This parameter is computed with the STM32CubeMX Tool.

Reference Manual to LL API cross reference:

- TIMINGR TIMINGR LL_I2C_SetTiming

LL_I2C_GetTimingPrescaler

Function name

__STATIC_INLINE uint32_t LL_I2C_GetTimingPrescaler (const I2C_TypeDef * I2Cx)

Function description

Get the Timing Prescaler setting.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Value:** between Min_Data=0x0 and Max_Data=0xF

Reference Manual to LL API cross reference:

- TIMINGR PRESC LL_I2C_GetTimingPrescaler

LL_I2C_GetClockLowPeriod

Function name

__STATIC_INLINE uint32_t LL_I2C_GetClockLowPeriod (const I2C_TypeDef * I2Cx)

Function description

Get the SCL low period setting.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- TIMINGR SCLL LL_I2C_GetClockLowPeriod

LL_I2C_GetClockHighPeriod

Function name

__STATIC_INLINE uint32_t LL_I2C_GetClockHighPeriod (const I2C_TypeDef * I2Cx)

Function description

Get the SCL high period setting.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- TIMINGR SCLH LL_I2C_GetClockHighPeriod

LL_I2C_GetDataHoldTime

Function name

__STATIC_INLINE uint32_t LL_I2C_GetDataHoldTime (const I2C_TypeDef * I2Cx)

Function description

Get the SDA hold time.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Value:** between Min_Data=0x0 and Max_Data=0xF

Reference Manual to LL API cross reference:

- TIMINGR SDADEL LL_I2C_GetDataHoldTime

LL_I2C_GetDataSetupTime

Function name

__STATIC_INLINE uint32_t LL_I2C_GetDataSetupTime (const I2C_TypeDef * I2Cx)

Function description

Get the SDA setup time.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Value:** between Min_Data=0x0 and Max_Data=0xF

Reference Manual to LL API cross reference:

- TIMINGR SCLDEL LL_I2C_GetDataSetupTime

LL_I2C_SetMode

Function name

__STATIC_INLINE void LL_I2C_SetMode (I2C_TypeDef * I2Cx, uint32_t PeripheralMode)

Function description

Configure peripheral mode.

Parameters

- **I2Cx:** I2C Instance.
- **PeripheralMode:** This parameter can be one of the following values:
  - LL_I2C_MODE_I2C
  - LL_I2C_MODE_SMBUS_HOST
  - LL_I2C_MODE_SMBUS_DEVICE
  - LL_I2C_MODE_SMBUS_DEVICE_ARP

Return values

- **None:**

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- CR1 SMBHEN LL_I2C_SetMode
- CR1 SMBDEN LL_I2C_SetMode

LL_I2C_GetMode

Function name

__STATIC_INLINE uint32_t LL_I2C_GetMode (const I2C_TypeDef * I2Cx)

Function description

Get peripheral mode.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Returned:** value can be one of the following values:
  - LL_I2C_MODE_I2C
  - LL_I2C_MODE_SMBUS_HOST
  - LL_I2C_MODE_SMBUS_DEVICE
  - LL_I2C_MODE_SMBUS_DEVICE_ARP

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- CR1 SMBHEN LL_I2C_GetMode
- CR1 SMBDEN LL_I2C_GetMode

LL_I2C_EnableSMBusAlert

Function name

__STATIC_INLINE void LL_I2C_EnableSMBusAlert (I2C_TypeDef * I2Cx)

Function description

Enable SMBus alert (Host or Device mode)

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
- SMBus Device mode: SMBus Alert pin is drived low and Alert Response Address Header acknowledge is enabled. SMBus Host mode:SMBus Alert pin management is supported.

Reference Manual to LL API cross reference:

- CR1 ALERTEN LL_I2C_EnableSMBusAlert

LL_I2C_DisableSMBusAlert

Function name

__STATIC_INLINE void LL_I2C_DisableSMBusAlert (I2C_TypeDef * I2Cx)

Function description

Disable SMBus alert (Host or Device mode)

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
- SMBus Device mode: SMBus Alert pin is not drived (can be used as a standard GPIO) and Alert Response Address Header acknowledge is disabled. SMBus Host mode:SMBus Alert pin management is not supported.

Reference Manual to LL API cross reference:

- CR1 ALERTEN LL_I2C_DisableSMBusAlert

LL_I2C_IsEnabledSMBusAlert

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledSMBusAlert (const I2C_TypeDef * I2Cx)

Function description

Check if SMBus alert (Host or Device mode) is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- CR1 ALERTEN LL_I2C_IsEnabledSMBusAlert

LL_I2C_EnableSMBusPEC

Function name

__STATIC_INLINE void LL_I2C_EnableSMBusPEC (I2C_TypeDef * I2Cx)

Function description

Enable SMBus Packet Error Calculation (PEC).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- CR1 PECEN LL_I2C_EnableSMBusPEC

LL_I2C_DisableSMBusPEC

Function name

__STATIC_INLINE void LL_I2C_DisableSMBusPEC (I2C_TypeDef * I2Cx)

Function description

Disable SMBus Packet Error Calculation (PEC).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- CR1 PECEN LL_I2C_DisableSMBusPEC

LL_I2C_IsEnabledSMBusPEC

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledSMBusPEC (const I2C_TypeDef * I2Cx)

Function description

Check if SMBus Packet Error Calculation (PEC) is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- CR1 PECEN LL_I2C_IsEnabledSMBusPEC

LL_I2C_ConfigSMBusTimeout

Function name

__STATIC_INLINE void LL_I2C_ConfigSMBusTimeout (I2C_TypeDef * I2Cx, uint32_t TimeoutA, uint32_t TimeoutAMode, uint32_t TimeoutB)

Function description

Configure the SMBus Clock Timeout.

Parameters

- **I2Cx:** I2C Instance.
- **TimeoutA:** This parameter must be a value between Min_Data=0 and Max_Data=0xFFF.
- **TimeoutAMode:** This parameter can be one of the following values:
  - LL_I2C_SMBUS_TIMEOUTA_MODE_SCL_LOW
  - LL_I2C_SMBUS_TIMEOUTA_MODE_SDA_SCL_HIGH
- **TimeoutB:**

Return values

- **None:**

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
- This configuration can only be programmed when associated Timeout is disabled (TimeoutA and/ orTimeoutB).

Reference Manual to LL API cross reference:

- TIMEOUTR TIMEOUTA LL_I2C_ConfigSMBusTimeout
- TIMEOUTR TIDLE LL_I2C_ConfigSMBusTimeout
- TIMEOUTR TIMEOUTB LL_I2C_ConfigSMBusTimeout

LL_I2C_SetSMBusTimeoutA

Function name

__STATIC_INLINE void LL_I2C_SetSMBusTimeoutA (I2C_TypeDef * I2Cx, uint32_t TimeoutA)

Function description

Configure the SMBus Clock TimeoutA (SCL low timeout or SCL and SDA high timeout depends on TimeoutA mode).

Parameters

- **I2Cx:** I2C Instance.
- **TimeoutA:** This parameter must be a value between Min_Data=0 and Max_Data=0xFFF.

Return values

- **None:**

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
- These bits can only be programmed when TimeoutA is disabled.

Reference Manual to LL API cross reference:

- TIMEOUTR TIMEOUTA LL_I2C_SetSMBusTimeoutA

LL_I2C_GetSMBusTimeoutA

Function name

__STATIC_INLINE uint32_t LL_I2C_GetSMBusTimeoutA (const I2C_TypeDef * I2Cx)

Function description

Get the SMBus Clock TimeoutA setting.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Value:** between Min_Data=0 and Max_Data=0xFFF

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- TIMEOUTR TIMEOUTA LL_I2C_GetSMBusTimeoutA

LL_I2C_SetSMBusTimeoutAMode

Function name

__STATIC_INLINE void LL_I2C_SetSMBusTimeoutAMode (I2C_TypeDef * I2Cx, uint32_t TimeoutAMode)

Function description

Set the SMBus Clock TimeoutA mode.

Parameters

- **I2Cx:** I2C Instance.
- **TimeoutAMode:** This parameter can be one of the following values:
  - LL_I2C_SMBUS_TIMEOUTA_MODE_SCL_LOW
  - LL_I2C_SMBUS_TIMEOUTA_MODE_SDA_SCL_HIGH

Return values

- **None:**

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
- This bit can only be programmed when TimeoutA is disabled.

Reference Manual to LL API cross reference:

- TIMEOUTR TIDLE LL_I2C_SetSMBusTimeoutAMode

LL_I2C_GetSMBusTimeoutAMode

Function name

__STATIC_INLINE uint32_t LL_I2C_GetSMBusTimeoutAMode (const I2C_TypeDef * I2Cx)

Function description

Get the SMBus Clock TimeoutA mode.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Returned:** value can be one of the following values:
  - LL_I2C_SMBUS_TIMEOUTA_MODE_SCL_LOW
  - LL_I2C_SMBUS_TIMEOUTA_MODE_SDA_SCL_HIGH

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- TIMEOUTR TIDLE LL_I2C_GetSMBusTimeoutAMode

LL_I2C_SetSMBusTimeoutB

Function name

__STATIC_INLINE void LL_I2C_SetSMBusTimeoutB (I2C_TypeDef * I2Cx, uint32_t TimeoutB)

Function description

Configure the SMBus Extended Cumulative Clock TimeoutB (Master or Slave mode).

Parameters

- **I2Cx:** I2C Instance.
- **TimeoutB:** This parameter must be a value between Min_Data=0 and Max_Data=0xFFF.

Return values

- **None:**

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
- These bits can only be programmed when TimeoutB is disabled.

Reference Manual to LL API cross reference:

- TIMEOUTR TIMEOUTB LL_I2C_SetSMBusTimeoutB

LL_I2C_GetSMBusTimeoutB

Function name

__STATIC_INLINE uint32_t LL_I2C_GetSMBusTimeoutB (const I2C_TypeDef * I2Cx)

Function description

Get the SMBus Extended Cumulative Clock TimeoutB setting.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Value:** between Min_Data=0 and Max_Data=0xFFF

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- TIMEOUTR TIMEOUTB LL_I2C_GetSMBusTimeoutB

LL_I2C_EnableSMBusTimeout

Function name

__STATIC_INLINE void LL_I2C_EnableSMBusTimeout (I2C_TypeDef * I2Cx, uint32_t ClockTimeout)

Function description

Enable the SMBus Clock Timeout.

Parameters

- **I2Cx:** I2C Instance.
- **ClockTimeout:** This parameter can be one of the following values:
  - LL_I2C_SMBUS_TIMEOUTA
  - LL_I2C_SMBUS_TIMEOUTB
  - LL_I2C_SMBUS_ALL_TIMEOUT

Return values

- **None:**

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- TIMEOUTR TIMOUTEN LL_I2C_EnableSMBusTimeout
- TIMEOUTR TEXTEN LL_I2C_EnableSMBusTimeout

LL_I2C_DisableSMBusTimeout

Function name

__STATIC_INLINE void LL_I2C_DisableSMBusTimeout (I2C_TypeDef * I2Cx, uint32_t ClockTimeout)

Function description

Disable the SMBus Clock Timeout.

Parameters

- **I2Cx:** I2C Instance.
- **ClockTimeout:** This parameter can be one of the following values:
  - LL_I2C_SMBUS_TIMEOUTA
  - LL_I2C_SMBUS_TIMEOUTB
  - LL_I2C_SMBUS_ALL_TIMEOUT

Return values

- **None:**

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- TIMEOUTR TIMOUTEN LL_I2C_DisableSMBusTimeout
- TIMEOUTR TEXTEN LL_I2C_DisableSMBusTimeout

LL_I2C_IsEnabledSMBusTimeout

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledSMBusTimeout (const I2C_TypeDef * I2Cx, uint32_t ClockTimeout)

Function description

Check if the SMBus Clock Timeout is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.
- **ClockTimeout:** This parameter can be one of the following values:
  - LL_I2C_SMBUS_TIMEOUTA
  - LL_I2C_SMBUS_TIMEOUTB
  - LL_I2C_SMBUS_ALL_TIMEOUT

Return values

- **State:** of bit (1 or 0).

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- TIMEOUTR TIMOUTEN LL_I2C_IsEnabledSMBusTimeout
- TIMEOUTR TEXTEN LL_I2C_IsEnabledSMBusTimeout

LL_I2C_EnableIT_TX

Function name

__STATIC_INLINE void LL_I2C_EnableIT_TX (I2C_TypeDef * I2Cx)

Function description

Enable TXIS interrupt.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TXIE LL_I2C_EnableIT_TX

LL_I2C_DisableIT_TX

Function name

__STATIC_INLINE void LL_I2C_DisableIT_TX (I2C_TypeDef * I2Cx)

Function description

Disable TXIS interrupt.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TXIE LL_I2C_DisableIT_TX

LL_I2C_IsEnabledIT_TX

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledIT_TX (const I2C_TypeDef * I2Cx)

Function description

Check if the TXIS Interrupt is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 TXIE LL_I2C_IsEnabledIT_TX

LL_I2C_EnableIT_RX

Function name

__STATIC_INLINE void LL_I2C_EnableIT_RX (I2C_TypeDef * I2Cx)

Function description

Enable RXNE interrupt.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RXIE LL_I2C_EnableIT_RX

LL_I2C_DisableIT_RX

Function name

__STATIC_INLINE void LL_I2C_DisableIT_RX (I2C_TypeDef * I2Cx)

Function description

Disable RXNE interrupt.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RXIE LL_I2C_DisableIT_RX

LL_I2C_IsEnabledIT_RX

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledIT_RX (const I2C_TypeDef * I2Cx)

Function description

Check if the RXNE Interrupt is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 RXIE LL_I2C_IsEnabledIT_RX

LL_I2C_EnableIT_ADDR

Function name

__STATIC_INLINE void LL_I2C_EnableIT_ADDR (I2C_TypeDef * I2Cx)

Function description

Enable Address match interrupt (slave mode only).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 ADDRIE LL_I2C_EnableIT_ADDR

LL_I2C_DisableIT_ADDR

Function name

__STATIC_INLINE void LL_I2C_DisableIT_ADDR (I2C_TypeDef * I2Cx)

Function description

Disable Address match interrupt (slave mode only).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 ADDRIE LL_I2C_DisableIT_ADDR

LL_I2C_IsEnabledIT_ADDR

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledIT_ADDR (const I2C_TypeDef * I2Cx)

Function description

Check if Address match interrupt is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 ADDRIE LL_I2C_IsEnabledIT_ADDR

LL_I2C_EnableIT_NACK

Function name

__STATIC_INLINE void LL_I2C_EnableIT_NACK (I2C_TypeDef * I2Cx)

Function description

Enable Not acknowledge received interrupt.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 NACKIE LL_I2C_EnableIT_NACK

LL_I2C_DisableIT_NACK

Function name

__STATIC_INLINE void LL_I2C_DisableIT_NACK (I2C_TypeDef * I2Cx)

Function description

Disable Not acknowledge received interrupt.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 NACKIE LL_I2C_DisableIT_NACK

LL_I2C_IsEnabledIT_NACK

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledIT_NACK (const I2C_TypeDef * I2Cx)

Function description

Check if Not acknowledge received interrupt is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 NACKIE LL_I2C_IsEnabledIT_NACK

LL_I2C_EnableIT_STOP

Function name

__STATIC_INLINE void LL_I2C_EnableIT_STOP (I2C_TypeDef * I2Cx)

Function description

Enable STOP detection interrupt.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 STOPIE LL_I2C_EnableIT_STOP

LL_I2C_DisableIT_STOP

Function name

__STATIC_INLINE void LL_I2C_DisableIT_STOP (I2C_TypeDef * I2Cx)

Function description

Disable STOP detection interrupt.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 STOPIE LL_I2C_DisableIT_STOP

LL_I2C_IsEnabledIT_STOP

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledIT_STOP (const I2C_TypeDef * I2Cx)

Function description

Check if STOP detection interrupt is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 STOPIE LL_I2C_IsEnabledIT_STOP

LL_I2C_EnableIT_TC

Function name

__STATIC_INLINE void LL_I2C_EnableIT_TC (I2C_TypeDef * I2Cx)

Function description

Enable Transfer Complete interrupt.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- Any of these events will generate interrupt : Transfer Complete (TC) Transfer Complete Reload (TCR)

Reference Manual to LL API cross reference:

- CR1 TCIE LL_I2C_EnableIT_TC

LL_I2C_DisableIT_TC

Function name

__STATIC_INLINE void LL_I2C_DisableIT_TC (I2C_TypeDef * I2Cx)

Function description

Disable Transfer Complete interrupt.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- Any of these events will generate interrupt : Transfer Complete (TC) Transfer Complete Reload (TCR)

Reference Manual to LL API cross reference:

- CR1 TCIE LL_I2C_DisableIT_TC

LL_I2C_IsEnabledIT_TC

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledIT_TC (const I2C_TypeDef * I2Cx)

Function description

Check if Transfer Complete interrupt is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 TCIE LL_I2C_IsEnabledIT_TC

LL_I2C_EnableIT_ERR

Function name

__STATIC_INLINE void LL_I2C_EnableIT_ERR (I2C_TypeDef * I2Cx)

Function description

Enable Error interrupts.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
- Any of these errors will generate interrupt : Arbitration Loss (ARLO) Bus Error detection (BERR) Overrun/ Underrun (OVR) SMBus Timeout detection (TIMEOUT) SMBus PEC error detection (PECERR) SMBus Alert pin event detection (ALERT)

Reference Manual to LL API cross reference:

- CR1 ERRIE LL_I2C_EnableIT_ERR

LL_I2C_DisableIT_ERR

Function name

__STATIC_INLINE void LL_I2C_DisableIT_ERR (I2C_TypeDef * I2Cx)

Function description

Disable Error interrupts.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
- Any of these errors will generate interrupt : Arbitration Loss (ARLO) Bus Error detection (BERR) Overrun/ Underrun (OVR) SMBus Timeout detection (TIMEOUT) SMBus PEC error detection (PECERR) SMBus Alert pin event detection (ALERT)

Reference Manual to LL API cross reference:

- CR1 ERRIE LL_I2C_DisableIT_ERR

LL_I2C_IsEnabledIT_ERR

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledIT_ERR (const I2C_TypeDef * I2Cx)

Function description

Check if Error interrupts are enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 ERRIE LL_I2C_IsEnabledIT_ERR

LL_I2C_IsActiveFlag_TXE

Function name

__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_TXE (const I2C_TypeDef * I2Cx)

Function description

Indicate the status of Transmit data register empty flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- RESET: When next data is written in Transmit data register. SET: When Transmit data register is empty.

Reference Manual to LL API cross reference:

- ISR TXE LL_I2C_IsActiveFlag_TXE

LL_I2C_IsActiveFlag_TXIS

Function name

__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_TXIS (const I2C_TypeDef * I2Cx)

Function description

Indicate the status of Transmit interrupt flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- RESET: When next data is written in Transmit data register. SET: When Transmit data register is empty.

Reference Manual to LL API cross reference:

- ISR TXIS LL_I2C_IsActiveFlag_TXIS

LL_I2C_IsActiveFlag_RXNE

Function name

__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_RXNE (const I2C_TypeDef * I2Cx)

Function description

Indicate the status of Receive data register not empty flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- RESET: When Receive data register is read. SET: When the received data is copied in Receive data register.

Reference Manual to LL API cross reference:

- ISR RXNE LL_I2C_IsActiveFlag_RXNE

LL_I2C_IsActiveFlag_ADDR

Function name

__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_ADDR (const I2C_TypeDef * I2Cx)

Function description

Indicate the status of Address matched flag (slave mode).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- RESET: Clear default value. SET: When the received slave address matched with one of the enabled slave address.

Reference Manual to LL API cross reference:

- ISR ADDR LL_I2C_IsActiveFlag_ADDR

LL_I2C_IsActiveFlag_NACK

Function name

__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_NACK (const I2C_TypeDef * I2Cx)

Function description

Indicate the status of Not Acknowledge received flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- RESET: Clear default value. SET: When a NACK is received after a byte transmission.

Reference Manual to LL API cross reference:

- ISR NACKF LL_I2C_IsActiveFlag_NACK

LL_I2C_IsActiveFlag_STOP

Function name

__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_STOP (const I2C_TypeDef * I2Cx)

Function description

Indicate the status of Stop detection flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- RESET: Clear default value. SET: When a Stop condition is detected.

Reference Manual to LL API cross reference:

- ISR STOPF LL_I2C_IsActiveFlag_STOP

LL_I2C_IsActiveFlag_TC

Function name

__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_TC (const I2C_TypeDef * I2Cx)

Function description

Indicate the status of Transfer complete flag (master mode).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- RESET: Clear default value. SET: When RELOAD=0, AUTOEND=0 and NBYTES date have been transferred.

Reference Manual to LL API cross reference:

- ISR TC LL_I2C_IsActiveFlag_TC

LL_I2C_IsActiveFlag_TCR

Function name

__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_TCR (const I2C_TypeDef * I2Cx)

Function description

Indicate the status of Transfer complete flag (master mode).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- RESET: Clear default value. SET: When RELOAD=1 and NBYTES date have been transferred.

Reference Manual to LL API cross reference:

- ISR TCR LL_I2C_IsActiveFlag_TCR

LL_I2C_IsActiveFlag_BERR

Function name

__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_BERR (const I2C_TypeDef * I2Cx)

Function description

Indicate the status of Bus error flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- RESET: Clear default value. SET: When a misplaced Start or Stop condition is detected.

Reference Manual to LL API cross reference:

- ISR BERR LL_I2C_IsActiveFlag_BERR

LL_I2C_IsActiveFlag_ARLO

Function name

__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_ARLO (const I2C_TypeDef * I2Cx)

Function description

Indicate the status of Arbitration lost flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- RESET: Clear default value. SET: When arbitration lost.

Reference Manual to LL API cross reference:

- ISR ARLO LL_I2C_IsActiveFlag_ARLO

LL_I2C_IsActiveFlag_OVR

Function name

__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_OVR (const I2C_TypeDef * I2Cx)

Function description

Indicate the status of Overrun/Underrun flag (slave mode).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- RESET: Clear default value. SET: When an overrun/underrun error occurs (Clock Stretching Disabled).

Reference Manual to LL API cross reference:

- ISR OVR LL_I2C_IsActiveFlag_OVR

LL_I2C_IsActiveSMBusFlag_PECERR

Function name

__STATIC_INLINE uint32_t LL_I2C_IsActiveSMBusFlag_PECERR (const I2C_TypeDef * I2Cx)

Function description

Indicate the status of SMBus PEC error flag in reception.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
- RESET: Clear default value. SET: When the received PEC does not match with the PEC register content.

Reference Manual to LL API cross reference:

- ISR PECERR LL_I2C_IsActiveSMBusFlag_PECERR

LL_I2C_IsActiveSMBusFlag_TIMEOUT

Function name

__STATIC_INLINE uint32_t LL_I2C_IsActiveSMBusFlag_TIMEOUT (const I2C_TypeDef * I2Cx)

Function description

Indicate the status of SMBus Timeout detection flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
- RESET: Clear default value. SET: When a timeout or extended clock timeout occurs.

Reference Manual to LL API cross reference:

- ISR TIMEOUT LL_I2C_IsActiveSMBusFlag_TIMEOUT

LL_I2C_IsActiveSMBusFlag_ALERT

Function name

__STATIC_INLINE uint32_t LL_I2C_IsActiveSMBusFlag_ALERT (const I2C_TypeDef * I2Cx)

Function description

Indicate the status of SMBus alert flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
- RESET: Clear default value. SET: When SMBus host configuration, SMBus alert enabled and a falling edge event occurs on SMBA pin.

Reference Manual to LL API cross reference:

- ISR ALERT LL_I2C_IsActiveSMBusFlag_ALERT

LL_I2C_IsActiveFlag_BUSY

Function name

__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_BUSY (const I2C_TypeDef * I2Cx)

Function description

Indicate the status of Bus Busy flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- RESET: Clear default value. SET: When a Start condition is detected.

Reference Manual to LL API cross reference:

- ISR BUSY LL_I2C_IsActiveFlag_BUSY

LL_I2C_ClearFlag_ADDR

Function name

__STATIC_INLINE void LL_I2C_ClearFlag_ADDR (I2C_TypeDef * I2Cx)

Function description

Clear Address Matched flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR ADDRCF LL_I2C_ClearFlag_ADDR

LL_I2C_ClearFlag_NACK

Function name

__STATIC_INLINE void LL_I2C_ClearFlag_NACK (I2C_TypeDef * I2Cx)

Function description

Clear Not Acknowledge flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR NACKCF LL_I2C_ClearFlag_NACK

LL_I2C_ClearFlag_STOP

Function name

__STATIC_INLINE void LL_I2C_ClearFlag_STOP (I2C_TypeDef * I2Cx)

Function description

Clear Stop detection flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR STOPCF LL_I2C_ClearFlag_STOP

LL_I2C_ClearFlag_TXE

Function name

__STATIC_INLINE void LL_I2C_ClearFlag_TXE (I2C_TypeDef * I2Cx)

Function description

Clear Transmit data register empty flag (TXE).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- This bit can be clear by software in order to flush the transmit data register (TXDR).

Reference Manual to LL API cross reference:

- ISR TXE LL_I2C_ClearFlag_TXE

LL_I2C_ClearFlag_BERR

Function name

__STATIC_INLINE void LL_I2C_ClearFlag_BERR (I2C_TypeDef * I2Cx)

Function description

Clear Bus error flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR BERRCF LL_I2C_ClearFlag_BERR

LL_I2C_ClearFlag_ARLO

Function name

__STATIC_INLINE void LL_I2C_ClearFlag_ARLO (I2C_TypeDef * I2Cx)

Function description

Clear Arbitration lost flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR ARLOCF LL_I2C_ClearFlag_ARLO

LL_I2C_ClearFlag_OVR

Function name

__STATIC_INLINE void LL_I2C_ClearFlag_OVR (I2C_TypeDef * I2Cx)

Function description

Clear Overrun/Underrun flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR OVRCF LL_I2C_ClearFlag_OVR

LL_I2C_ClearSMBusFlag_PECERR

Function name

__STATIC_INLINE void LL_I2C_ClearSMBusFlag_PECERR (I2C_TypeDef * I2Cx)

Function description

Clear SMBus PEC error flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- ICR PECCF LL_I2C_ClearSMBusFlag_PECERR

LL_I2C_ClearSMBusFlag_TIMEOUT

Function name

__STATIC_INLINE void LL_I2C_ClearSMBusFlag_TIMEOUT (I2C_TypeDef * I2Cx)

Function description

Clear SMBus Timeout detection flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- ICR TIMOUTCF LL_I2C_ClearSMBusFlag_TIMEOUT

LL_I2C_ClearSMBusFlag_ALERT

Function name

__STATIC_INLINE void LL_I2C_ClearSMBusFlag_ALERT (I2C_TypeDef * I2Cx)

Function description

Clear SMBus Alert flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- ICR ALERTCF LL_I2C_ClearSMBusFlag_ALERT

LL_I2C_EnableAutoEndMode

Function name

__STATIC_INLINE void LL_I2C_EnableAutoEndMode (I2C_TypeDef * I2Cx)

Function description

Enable automatic STOP condition generation (master mode).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- Automatic end mode : a STOP condition is automatically sent when NBYTES data are transferred. This bit has no effect in slave mode or when RELOAD bit is set.

Reference Manual to LL API cross reference:

- CR2 AUTOEND LL_I2C_EnableAutoEndMode

LL_I2C_DisableAutoEndMode

Function name

__STATIC_INLINE void LL_I2C_DisableAutoEndMode (I2C_TypeDef * I2Cx)

Function description

Disable automatic STOP condition generation (master mode).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- Software end mode : TC flag is set when NBYTES data are transferre, stretching SCL low.

Reference Manual to LL API cross reference:

- CR2 AUTOEND LL_I2C_DisableAutoEndMode

LL_I2C_IsEnabledAutoEndMode

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledAutoEndMode (const I2C_TypeDef * I2Cx)

Function description

Check if automatic STOP condition is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR2 AUTOEND LL_I2C_IsEnabledAutoEndMode

LL_I2C_EnableReloadMode

Function name

__STATIC_INLINE void LL_I2C_EnableReloadMode (I2C_TypeDef * I2Cx)

Function description

Enable reload mode (master mode).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- The transfer is not completed after the NBYTES data transfer, NBYTES will be reloaded when TCR flag is set.

Reference Manual to LL API cross reference:

- CR2 RELOAD LL_I2C_EnableReloadMode

LL_I2C_DisableReloadMode

Function name

__STATIC_INLINE void LL_I2C_DisableReloadMode (I2C_TypeDef * I2Cx)

Function description

Disable reload mode (master mode).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- The transfer is completed after the NBYTES data transfer(STOP or RESTART will follow).

Reference Manual to LL API cross reference:

- CR2 RELOAD LL_I2C_DisableReloadMode

LL_I2C_IsEnabledReloadMode

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledReloadMode (const I2C_TypeDef * I2Cx)

Function description

Check if reload mode is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR2 RELOAD LL_I2C_IsEnabledReloadMode

LL_I2C_SetTransferSize

Function name

__STATIC_INLINE void LL_I2C_SetTransferSize (I2C_TypeDef * I2Cx, uint32_t TransferSize)

Function description

Configure the number of bytes for transfer.

Parameters

- **I2Cx:** I2C Instance.
- **TransferSize:** This parameter must be a value between Min_Data=0x00 and Max_Data=0xFF.

Return values

- **None:**

Notes

- Changing these bits when START bit is set is not allowed.

Reference Manual to LL API cross reference:

- CR2 NBYTES LL_I2C_SetTransferSize

LL_I2C_GetTransferSize

Function name

__STATIC_INLINE uint32_t LL_I2C_GetTransferSize (const I2C_TypeDef * I2Cx)

Function description

Get the number of bytes configured for transfer.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Value:** between Min_Data=0x0 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- CR2 NBYTES LL_I2C_GetTransferSize

LL_I2C_AcknowledgeNextData

Function name

__STATIC_INLINE void LL_I2C_AcknowledgeNextData (I2C_TypeDef * I2Cx, uint32_t TypeAcknowledge)

Function description

Prepare the generation of a ACKnowledge or Non ACKnowledge condition after the address receive match code or next received byte.

Parameters

- **I2Cx:** I2C Instance.
- **TypeAcknowledge:** This parameter can be one of the following values:
  - LL_I2C_ACK
  - LL_I2C_NACK

Return values

- **None:**

Notes

- Usage in Slave mode only.

Reference Manual to LL API cross reference:

- CR2 NACK LL_I2C_AcknowledgeNextData

LL_I2C_GenerateStartCondition

Function name

__STATIC_INLINE void LL_I2C_GenerateStartCondition (I2C_TypeDef * I2Cx)

Function description

Generate a START or RESTART condition.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- The START bit can be set even if bus is BUSY or I2C is in slave mode. This action has no effect when RELOAD is set.

Reference Manual to LL API cross reference:

- CR2 START LL_I2C_GenerateStartCondition

LL_I2C_GenerateStopCondition

Function name

__STATIC_INLINE void LL_I2C_GenerateStopCondition (I2C_TypeDef * I2Cx)

Function description

Generate a STOP condition after the current byte transfer (master mode).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 STOP LL_I2C_GenerateStopCondition

LL_I2C_EnableAuto10BitRead

Function name

__STATIC_INLINE void LL_I2C_EnableAuto10BitRead (I2C_TypeDef * I2Cx)

Function description

Enable automatic RESTART Read request condition for 10bit address header (master mode).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- The master sends the complete 10bit slave address read sequence : Start + 2 bytes 10bit address in Write direction + Restart + first 7 bits of 10bit address in Read direction.

Reference Manual to LL API cross reference:

- CR2 HEAD10R LL_I2C_EnableAuto10BitRead

LL_I2C_DisableAuto10BitRead

Function name

__STATIC_INLINE void LL_I2C_DisableAuto10BitRead (I2C_TypeDef * I2Cx)

Function description

Disable automatic RESTART Read request condition for 10bit address header (master mode).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- The master only sends the first 7 bits of 10bit address in Read direction.

Reference Manual to LL API cross reference:

- CR2 HEAD10R LL_I2C_DisableAuto10BitRead

LL_I2C_IsEnabledAuto10BitRead

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledAuto10BitRead (const I2C_TypeDef * I2Cx)

Function description

Check if automatic RESTART Read request condition for 10bit address header is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR2 HEAD10R LL_I2C_IsEnabledAuto10BitRead

LL_I2C_SetTransferRequest

Function name

__STATIC_INLINE void LL_I2C_SetTransferRequest (I2C_TypeDef * I2Cx, uint32_t TransferRequest)

Function description

Configure the transfer direction (master mode).

Parameters

- **I2Cx:** I2C Instance.
- **TransferRequest:** This parameter can be one of the following values:
  - LL_I2C_REQUEST_WRITE
  - LL_I2C_REQUEST_READ

Return values

- **None:**

Notes

- Changing these bits when START bit is set is not allowed.

Reference Manual to LL API cross reference:

- CR2 RD_WRN LL_I2C_SetTransferRequest

LL_I2C_GetTransferRequest

Function name

__STATIC_INLINE uint32_t LL_I2C_GetTransferRequest (const I2C_TypeDef * I2Cx)

Function description

Get the transfer direction requested (master mode).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Returned:** value can be one of the following values:
  - LL_I2C_REQUEST_WRITE
  - LL_I2C_REQUEST_READ

Reference Manual to LL API cross reference:

- CR2 RD_WRN LL_I2C_GetTransferRequest

LL_I2C_SetSlaveAddr

Function name

__STATIC_INLINE void LL_I2C_SetSlaveAddr (I2C_TypeDef * I2Cx, uint32_t SlaveAddr)

Function description

Configure the slave address for transfer (master mode).

Parameters

- **I2Cx:** I2C Instance.
- **SlaveAddr:** This parameter must be a value between Min_Data=0x00 and Max_Data=0x3F.

Return values

- **None:**

Notes

- Changing these bits when START bit is set is not allowed.

Reference Manual to LL API cross reference:

- CR2 SADD LL_I2C_SetSlaveAddr

LL_I2C_GetSlaveAddr

Function name

__STATIC_INLINE uint32_t LL_I2C_GetSlaveAddr (const I2C_TypeDef * I2Cx)

Function description

Get the slave address programmed for transfer.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Value:** between Min_Data=0x0 and Max_Data=0x3F

Reference Manual to LL API cross reference:

- CR2 SADD LL_I2C_GetSlaveAddr

LL_I2C_HandleTransfer

Function name

__STATIC_INLINE void LL_I2C_HandleTransfer (I2C_TypeDef * I2Cx, uint32_t SlaveAddr, uint32_t SlaveAddrSize, uint32_t TransferSize, uint32_t EndMode, uint32_t Request)

Function description

Handles I2Cx communication when starting transfer or during transfer (TC or TCR flag are set).

Parameters

- **I2Cx:** I2C Instance.
- **SlaveAddr:** Specifies the slave address to be programmed.
- **SlaveAddrSize:** This parameter can be one of the following values:
  - LL_I2C_ADDRSLAVE_7BIT
  - LL_I2C_ADDRSLAVE_10BIT
- **TransferSize:** Specifies the number of bytes to be programmed. This parameter must be a value between Min_Data=0 and Max_Data=255.
- **EndMode:** This parameter can be one of the following values:
  - LL_I2C_MODE_RELOAD
  - LL_I2C_MODE_AUTOEND
  - LL_I2C_MODE_SOFTEND
  - LL_I2C_MODE_SMBUS_RELOAD
  - LL_I2C_MODE_SMBUS_AUTOEND_NO_PEC
  - LL_I2C_MODE_SMBUS_SOFTEND_NO_PEC
  - LL_I2C_MODE_SMBUS_AUTOEND_WITH_PEC
  - LL_I2C_MODE_SMBUS_SOFTEND_WITH_PEC
- **Request:** This parameter can be one of the following values:
  - LL_I2C_GENERATE_NOSTARTSTOP
  - LL_I2C_GENERATE_STOP
  - LL_I2C_GENERATE_START_READ
  - LL_I2C_GENERATE_START_WRITE
  - LL_I2C_GENERATE_RESTART_7BIT_READ
  - LL_I2C_GENERATE_RESTART_7BIT_WRITE
  - LL_I2C_GENERATE_RESTART_10BIT_READ
  - LL_I2C_GENERATE_RESTART_10BIT_WRITE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 SADD LL_I2C_HandleTransfer
- CR2 ADD10 LL_I2C_HandleTransfer
- CR2 RD_WRN LL_I2C_HandleTransfer
- CR2 START LL_I2C_HandleTransfer
- CR2 STOP LL_I2C_HandleTransfer
- CR2 RELOAD LL_I2C_HandleTransfer
- CR2 NBYTES LL_I2C_HandleTransfer
- CR2 AUTOEND LL_I2C_HandleTransfer
- CR2 HEAD10R LL_I2C_HandleTransfer

LL_I2C_GetTransferDirection

Function name

__STATIC_INLINE uint32_t LL_I2C_GetTransferDirection (const I2C_TypeDef * I2Cx)

Function description

Indicate the value of transfer direction (slave mode).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Returned:** value can be one of the following values:
  - LL_I2C_DIRECTION_WRITE
  - LL_I2C_DIRECTION_READ

Notes

- RESET: Write transfer, Slave enters in receiver mode. SET: Read transfer, Slave enters in transmitter mode.

Reference Manual to LL API cross reference:

- ISR DIR LL_I2C_GetTransferDirection

LL_I2C_GetAddressMatchCode

Function name

__STATIC_INLINE uint32_t LL_I2C_GetAddressMatchCode (const I2C_TypeDef * I2Cx)

Function description

Return the slave matched address.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x3F

Reference Manual to LL API cross reference:

- ISR ADDCODE LL_I2C_GetAddressMatchCode

LL_I2C_EnableSMBusPECCompare

Function name

__STATIC_INLINE void LL_I2C_EnableSMBusPECCompare (I2C_TypeDef * I2Cx)

Function description

Enable internal comparison of the SMBus Packet Error byte (transmission or reception mode).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
- This feature is cleared by hardware when the PEC byte is transferred, or when a STOP condition or an Address Matched is received. This bit has no effect when RELOAD bit is set. This bit has no effect in device mode when SBC bit is not set.

Reference Manual to LL API cross reference:

- CR2 PECBYTE LL_I2C_EnableSMBusPECCompare

LL_I2C_IsEnabledSMBusPECCompare

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledSMBusPECCompare (const I2C_TypeDef * I2Cx)

Function description

Check if the SMBus Packet Error byte internal comparison is requested or not.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- CR2 PECBYTE LL_I2C_IsEnabledSMBusPECCompare

LL_I2C_GetSMBusPEC

Function name

__STATIC_INLINE uint32_t LL_I2C_GetSMBusPEC (const I2C_TypeDef * I2Cx)

Function description

Get the SMBus Packet Error byte calculated.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0xFF

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- PECR PEC LL_I2C_GetSMBusPEC

LL_I2C_ReceiveData8

Function name

__STATIC_INLINE uint8_t LL_I2C_ReceiveData8 (const I2C_TypeDef * I2Cx)

Function description

Read Receive Data register.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- RXDR RXDATA LL_I2C_ReceiveData8

LL_I2C_TransmitData8

Function name

__STATIC_INLINE void LL_I2C_TransmitData8 (I2C_TypeDef * I2Cx, uint8_t Data)

Function description

Write in Transmit Data Register .

Parameters

- **I2Cx:** I2C Instance.
- **Data:** Value between Min_Data=0x00 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- TXDR TXDATA LL_I2C_TransmitData8

LL_I2C_Init

Function name

ErrorStatus LL_I2C_Init (I2C_TypeDef * I2Cx, const LL_I2C_InitTypeDef * I2C_InitStruct)

Function description

Initialize the I2C registers according to the specified parameters in I2C_InitStruct.

Parameters

- **I2Cx:** I2C Instance.
- **I2C_InitStruct:** pointer to a LL_I2C_InitTypeDef structure.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: I2C registers are initialized
  - ERROR: Not applicable

LL_I2C_DeInit

Function name

ErrorStatus LL_I2C_DeInit (const I2C_TypeDef * I2Cx)

Function description

De-initialize the I2C registers to their default reset values.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: I2C registers are de-initialized
  - ERROR: I2C registers are not de-initialized

LL_I2C_StructInit

Function name

void LL_I2C_StructInit (LL_I2C_InitTypeDef * I2C_InitStruct)

Function description

Set each LL_I2C_InitTypeDef field to default value.

Parameters

- **I2C_InitStruct:** Pointer to a LL_I2C_InitTypeDef structure.

Return values

- **None:**

94.3 I2C Firmware driver defines

The following section lists the various define and macros of the module.

94.3.1 I2C

I2C

Master Addressing Mode

LL_I2C_ADDRESSING_MODE_7BIT

Master operates in 7-bit addressing mode.

LL_I2C_ADDRESSING_MODE_10BIT

Master operates in 10-bit addressing mode.

Slave Address Length

LL_I2C_ADDRSLAVE_7BIT

Slave Address in 7-bit.

LL_I2C_ADDRSLAVE_10BIT

Slave Address in 10-bit.

Analog Filter Selection

LL_I2C_ANALOGFILTER_ENABLE

Analog filter is enabled.

LL_I2C_ANALOGFILTER_DISABLE

Analog filter is disabled.

Clear Flags Defines

LL_I2C_ICR_ADDRCF

Address Matched flag

LL_I2C_ICR_NACKCF

Not Acknowledge flag

LL_I2C_ICR_STOPCF

Stop detection flag

LL_I2C_ICR_BERRCF

Bus error flag

LL_I2C_ICR_ARLOCF

Arbitration Lost flag

LL_I2C_ICR_OVRCF

Overrun/Underrun flag

LL_I2C_ICR_PECCF

PEC error flag

LL_I2C_ICR_TIMOUTCF

Timeout detection flag

LL_I2C_ICR_ALERTCF

Alert flag

Read Write Direction

LL_I2C_DIRECTION_WRITE

Write transfer request by master, slave enters receiver mode.

LL_I2C_DIRECTION_READ

Read transfer request by master, slave enters transmitter mode.

DMA Register Data

LL_I2C_DMA_REG_DATA_TRANSMIT

Get address of data register used for transmission

LL_I2C_DMA_REG_DATA_RECEIVE

Get address of data register used for reception

Start And Stop Generation

LL_I2C_GENERATE_NOSTARTSTOP

Don't Generate Stop and Start condition.

LL_I2C_GENERATE_STOP

Generate Stop condition (Size should be set to 0).

LL_I2C_GENERATE_START_READ

Generate Start for read request.

LL_I2C_GENERATE_START_WRITE

Generate Start for write request.

LL_I2C_GENERATE_RESTART_7BIT_READ

Generate Restart for read request, slave 7Bit address.

LL_I2C_GENERATE_RESTART_7BIT_WRITE

Generate Restart for write request, slave 7Bit address.

LL_I2C_GENERATE_RESTART_10BIT_READ

Generate Restart for read request, slave 10Bit address.

LL_I2C_GENERATE_RESTART_10BIT_WRITE

Generate Restart for write request, slave 10Bit address.

Get Flags Defines

LL_I2C_ISR_TXE

Transmit data register empty

LL_I2C_ISR_TXIS

Transmit interrupt status

LL_I2C_ISR_RXNE

Receive data register not empty

LL_I2C_ISR_ADDR

Address matched (slave mode)

LL_I2C_ISR_NACKF

Not Acknowledge received flag

LL_I2C_ISR_STOPF

Stop detection flag

LL_I2C_ISR_TC

Transfer Complete (master mode)

LL_I2C_ISR_TCR

Transfer Complete Reload

LL_I2C_ISR_BERR

Bus error

LL_I2C_ISR_ARLO

Arbitration lost

LL_I2C_ISR_OVR

Overrun/Underrun (slave mode)

LL_I2C_ISR_PECERR

PEC Error in reception (SMBus mode)

LL_I2C_ISR_TIMEOUT

Timeout detection flag (SMBus mode)

LL_I2C_ISR_ALERT

SMBus alert (SMBus mode)

LL_I2C_ISR_BUSY

Bus busy

Acknowledge Generation

LL_I2C_ACK

ACK is sent after current received byte.

LL_I2C_NACK

NACK is sent after current received byte.

IT Defines

LL_I2C_CR1_TXIE

TX Interrupt enable

LL_I2C_CR1_RXIE

RX Interrupt enable

LL_I2C_CR1_ADDRIE

Address match Interrupt enable (slave only)

LL_I2C_CR1_NACKIE

Not acknowledge received Interrupt enable

LL_I2C_CR1_STOPIE

STOP detection Interrupt enable

LL_I2C_CR1_TCIE

Transfer Complete interrupt enable

LL_I2C_CR1_ERRIE

Error interrupts enable

Transfer End Mode

LL_I2C_MODE_RELOAD

Enable I2C Reload mode.

LL_I2C_MODE_AUTOEND

Enable I2C Automatic end mode with no HW PEC comparison.

LL_I2C_MODE_SOFTEND

Enable I2C Software end mode with no HW PEC comparison.

LL_I2C_MODE_SMBUS_RELOAD

Enable SMBUS Automatic end mode with HW PEC comparison.

LL_I2C_MODE_SMBUS_AUTOEND_NO_PEC

Enable SMBUS Automatic end mode with HW PEC comparison.

LL_I2C_MODE_SMBUS_SOFTEND_NO_PEC

Enable SMBUS Software end mode with HW PEC comparison.

LL_I2C_MODE_SMBUS_AUTOEND_WITH_PEC

Enable SMBUS Automatic end mode with HW PEC comparison.

LL_I2C_MODE_SMBUS_SOFTEND_WITH_PEC

Enable SMBUS Software end mode with HW PEC comparison.

Own Address 1 Length

LL_I2C_OWNADDRESS1_7BIT

Own address 1 is a 7-bit address.

LL_I2C_OWNADDRESS1_10BIT

Own address 1 is a 10-bit address.

Own Address 2 Masks

LL_I2C_OWNADDRESS2_NOMASK

Own Address2 No mask.

LL_I2C_OWNADDRESS2_MASK01

Only Address2 bits[7:2] are compared.

LL_I2C_OWNADDRESS2_MASK02

Only Address2 bits[7:3] are compared.

LL_I2C_OWNADDRESS2_MASK03

Only Address2 bits[7:4] are compared.

LL_I2C_OWNADDRESS2_MASK04

Only Address2 bits[7:5] are compared.

LL_I2C_OWNADDRESS2_MASK05

Only Address2 bits[7:6] are compared.

LL_I2C_OWNADDRESS2_MASK06

Only Address2 bits[7] are compared.

LL_I2C_OWNADDRESS2_MASK07

No comparison is done. All Address2 are acknowledged.

Peripheral Mode

LL_I2C_MODE_I2C

I2C Master or Slave mode

LL_I2C_MODE_SMBUS_HOST

SMBus Host address acknowledge

LL_I2C_MODE_SMBUS_DEVICE

SMBus Device default mode (Default address not acknowledge)

LL_I2C_MODE_SMBUS_DEVICE_ARP

SMBus Device Default address acknowledge

Transfer Request Direction

LL_I2C_REQUEST_WRITE

Master request a write transfer.

LL_I2C_REQUEST_READ

Master request a read transfer.

SMBus TimeoutA Mode SCL SDA Timeout

LL_I2C_SMBUS_TIMEOUTA_MODE_SCL_LOW

TimeoutA is used to detect SCL low level timeout.

LL_I2C_SMBUS_TIMEOUTA_MODE_SDA_SCL_HIGH

TimeoutA is used to detect both SCL and SDA high level timeout.

SMBus Timeout Selection

LL_I2C_SMBUS_TIMEOUTA

TimeoutA enable bit

LL_I2C_SMBUS_TIMEOUTB

TimeoutB (extended clock) enable bit

LL_I2C_SMBUS_ALL_TIMEOUT

TimeoutA and TimeoutB (extended clock) enable bits

Convert SDA SCL timings

__LL_I2C_CONVERT_TIMINGS

Description:

- Configure the SDA setup, hold time and the SCL high, low period.

Parameters:

- __PRESCALER__: This parameter must be a value between Min_Data=0 and Max_Data=0xF.
- __SETUP_TIME__: This parameter must be a value between Min_Data=0 and Max_Data=0xF. (tscldel = (SCLDEL+1)xtpresc)
- __HOLD_TIME__: This parameter must be a value between Min_Data=0 and Max_Data=0xF. (tsdadel = SDADELxtpresc)
- __SCLH_PERIOD__: This parameter must be a value between Min_Data=0 and Max_Data=0xFF. (tsclh = (SCLH+1)xtpresc)
- __SCLL_PERIOD__: This parameter must be a value between Min_Data=0 and Max_Data=0xFF. (tscll = (SCLL+1)xtpresc)

Return value:

- Value: between Min_Data=0 and Max_Data=0xFFFFFFFF

Common Write and read registers Macros

LL_I2C_WriteReg

Description:

- Write a value in I2C register.

Parameters:

- __INSTANCE__: I2C Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_I2C_ReadReg

Description:

- Read a value in I2C register.

Parameters:

- __INSTANCE__: I2C Instance
- __REG__: Register to be read

Return value:

- Register: value

95 LL I3C Generic Driver

95.1 I3C Firmware driver registers structures

95.1.1 LL_I3C_CtrlBusConfTypeDef

LL_I3C_CtrlBusConfTypeDef is defined in the stm32n6xx_ll_i3c.h

Data Fields

- ***uint32_t SDAHoldTime***
- ***uint32_t WaitTime***
- ***uint8_t SCLPPLowDuration***
- ***uint8_t SCLI3CHighDuration***
- ***uint8_t SCLODLowDuration***
- ***uint8_t SCLI2CHighDuration***
- ***uint8_t BusFreeDuration***
- ***uint8_t BusIdleDuration***

Field Documentation

- ***uint32_t LL_I3C_CtrlBusConfTypeDef::SDAHoldTime***
  Specifies the I3C SDA hold time. This parameter must be a value of ***I3C_LL_EC_SDA_HOLD_TIME***
- ***uint32_t LL_I3C_CtrlBusConfTypeDef::WaitTime***
  Specifies the time that the main and the new controllers should wait before issuing a start. This parameter must be a value of ***I3C_LL_EC_OWN_ACTIVITY_STATE***
- ***uint8_t LL_I3C_CtrlBusConfTypeDef::SCLPPLowDuration***
  Specifies the I3C SCL low duration in number of kernel clock cycles in I3C push-pull phases. This parameter must be a number between Min_Data=0 and Max_Data=0xFF.
- ***uint8_t LL_I3C_CtrlBusConfTypeDef::SCLI3CHighDuration***
  Specifies the I3C SCL high duration in number of kernel clock cycles, used for I3C messages for I3C open-drain and push pull phases. This parameter must be a number between Min_Data=0 and Max_Data=0xFF.
- ***uint8_t LL_I3C_CtrlBusConfTypeDef::SCLODLowDuration***
  Specifies the I3C SCL low duration in number of kernel clock cycles in open-drain phases, used for legacy I2C commands and for I3C open-drain phases. This parameter must be a number between Min_Data=0 and Max_Data=0xFF.
- ***uint8_t LL_I3C_CtrlBusConfTypeDef::SCLI2CHighDuration***
  Specifies the I3C SCL high duration in number of kernel clock cycles, used for legacy I2C commands. This parameter must be a number between Min_Data=0 and Max_Data=0xFF.
- ***uint8_t LL_I3C_CtrlBusConfTypeDef::BusFreeDuration***
  Specifies the I3C controller duration in number of kernel clock cycles, after a stop and before a start. This parameter must be a number between Min_Data=0 and Max_Data=0xFF.
- ***uint8_t LL_I3C_CtrlBusConfTypeDef::BusIdleDuration***
  Specifies the I3C controller duration in number of kernel clock cycles to be elapsed, after that both SDA and SCL are continuously high and stable before issuing a hot-join event. This parameter must be a number between Min_Data=0 and Max_Data=0xFF.

95.1.2 LL_I3C_TgtBusConfTypeDef

LL_I3C_TgtBusConfTypeDef is defined in the stm32n6xx_ll_i3c.h

Data Fields

- ***uint8_t BusAvailableDuration***

Field Documentation

- ***uint8_t LL_I3C_TgtBusConfTypeDef::BusAvailableDuration***
  Specifies the I3C target duration in number of kernel clock cycles, when the SDA and the SCL are high for at least taval. This parameter must be a number between Min_Data=0 and Max_Data=0xFF.

95.1.3 LL_I3C_InitTypeDef

LL_I3C_InitTypeDef is defined in the stm32n6xx_ll_i3c.h

Data Fields

- ***LL_I3C_CtrlBusConfTypeDef CtrlBusCharacteristic***
- ***LL_I3C_TgtBusConfTypeDef TgtBusCharacteristic***

Field Documentation

- ***LL_I3C_CtrlBusConfTypeDef LL_I3C_InitTypeDef::CtrlBusCharacteristic***
  Specifies the I3C controller bus characteristic configuration when Controller mode
- ***LL_I3C_TgtBusConfTypeDef LL_I3C_InitTypeDef::TgtBusCharacteristic***
  Specifies the I3C target bus characteristic configuration when Target mode

95.2 I3C Firmware driver API description

The following section lists the various functions of the I3C library.

95.2.1 Detailed description of functions

LL_I3C_Enable

Function name

__STATIC_INLINE void LL_I3C_Enable (I3C_TypeDef * I3Cx)

Function description

Enable I3C peripheral (EN = 1).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR EN LL_I3C_Enable

LL_I3C_Disable

Function name

__STATIC_INLINE void LL_I3C_Disable (I3C_TypeDef * I3Cx)

Function description

Disable I3C peripheral (EN = 0).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Notes

- Controller mode: before clearing EN, all possible target requests must be disabled using DISEC CCC.
  Target mode: software is not expected clearing EN unless a partial reset of the IP is needed

Reference Manual to LL API cross reference:

- CFGR EN LL_I3C_Disable

LL_I3C_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabled (const I3C_TypeDef * I3Cx)

Function description

Check if the I3C peripheral is enabled or disabled.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CFGR EN LL_I3C_IsEnabled

LL_I3C_IsEnabledReset

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledReset (const I3C_TypeDef * I3Cx)

Function description

Check if Reset action is required or not required.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- This bit indicates if Reset Action field has been updated by HW upon reception of RSTACT during current frame.

Reference Manual to LL API cross reference:

- DEVR0 RSTVAL LL_I3C_IsEnabledReset

LL_I3C_SetMode

Function name

__STATIC_INLINE void LL_I3C_SetMode (I3C_TypeDef * I3Cx, uint32_t PeripheralMode)

Function description

Configure peripheral mode.

Parameters

- **I3Cx:** I3C Instance.
- **PeripheralMode:** This parameter can be one of the following values:
  - LL_I3C_MODE_CONTROLLER
  - LL_I3C_MODE_TARGET

Return values

- **None:**

Notes

- This bit can only be programmed when the I3C is disabled (EN = 0).

Reference Manual to LL API cross reference:

- CFGR CRINIT LL_I3C_SetMode

LL_I3C_GetMode

Function name

__STATIC_INLINE uint32_t LL_I3C_GetMode (const I3C_TypeDef * I3Cx)

Function description

Get peripheral mode.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Returned:** value can be one of the following values:
  - LL_I3C_MODE_CONTROLLER
  - LL_I3C_MODE_TARGET

Reference Manual to LL API cross reference:

- CFGR CRINIT LL_I3C_GetMode

LL_I3C_EnableArbitrationHeader

Function name

__STATIC_INLINE void LL_I3C_EnableArbitrationHeader (I3C_TypeDef * I3Cx)

Function description

An arbitration header (7'h7E) is sent after Start in case of legacy I2C or I3C private transfers.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Notes

- This bit can be modified only when there is no frame ongoing

Reference Manual to LL API cross reference:

- CFGR NOARBH LL_I3C_EnableArbitrationHeader

LL_I3C_DisableArbitrationHeader

Function name

__STATIC_INLINE void LL_I3C_DisableArbitrationHeader (I3C_TypeDef * I3Cx)

Function description

Target address is sent directly after a Start in case of legacy I2C or I3C private transfers.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Notes

- This bit can be modified only when there is no frame ongoing

Reference Manual to LL API cross reference:

- CFGR NOARBH LL_I3C_DisableArbitrationHeader

LL_I3C_IsEnabledArbitrationHeader

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledArbitrationHeader (const I3C_TypeDef * I3Cx)

Function description

Check if the arbitration header is enabled of disabled.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CFGR NOARBH LL_I3C_IsEnabledArbitrationHeader

LL_I3C_EnableResetPattern

Function name

__STATIC_INLINE void LL_I3C_EnableResetPattern (I3C_TypeDef * I3Cx)

Function description

A Reset Pattern is inserted before the STOP at the end of a frame when the last CCC of the frame was RSTACT CCC.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Notes

- This bit can be modified only when there is no frame ongoing

Reference Manual to LL API cross reference:

- CFGR RSTPTRN LL_I3C_EnableResetPattern

LL_I3C_DisableResetPattern

Function name

__STATIC_INLINE void LL_I3C_DisableResetPattern (I3C_TypeDef * I3Cx)

Function description

A single STOP is emitted at the end of a frame.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Notes

- This bit can be modified only when there is no frame ongoing

Reference Manual to LL API cross reference:

- CFGR RSTPTRN LL_I3C_DisableResetPattern

LL_I3C_IsEnabledResetPattern

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledResetPattern (const I3C_TypeDef * I3Cx)

Function description

Check if Reset Pattern is enabled of disabled.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CFGR RSTPTRN LL_I3C_IsEnabledResetPattern

LL_I3C_EnableExitPattern

Function name

__STATIC_INLINE void LL_I3C_EnableExitPattern (I3C_TypeDef * I3Cx)

Function description

An Exit Pattern is sent after header (MTYPE = header) to program an escalation fault.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Notes

- This bit can be modified only when there is no frame ongoing

Reference Manual to LL API cross reference:

- CFGR EXITPTRN LL_I3C_EnableExitPattern

LL_I3C_DisableExitPattern

Function name

__STATIC_INLINE void LL_I3C_DisableExitPattern (I3C_TypeDef * I3Cx)

Function description

An Exit Pattern is not sent after header (MTYPE = header).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Notes

- This bit can be modified only when there is no frame ongoing

Reference Manual to LL API cross reference:

- CFGR EXITPTRN LL_I3C_DisableExitPattern

LL_I3C_IsEnabledExitPattern

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledExitPattern (const I3C_TypeDef * I3Cx)

Function description

Check if Exit Pattern is enabled or disabled.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CFGR EXITPTRN LL_I3C_IsEnabledExitPattern

LL_I3C_EnableHighKeeperSDA

Function name

__STATIC_INLINE void LL_I3C_EnableHighKeeperSDA (I3C_TypeDef * I3Cx)

Function description

High Keeper is enabled and will be used in place of standard Open drain Pull Up device during handoff procedures.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Notes

- This bit can only be programmed when the I3C is disabled (EN = 0).

Reference Manual to LL API cross reference:

- CFGR HKSDAEN LL_I3C_EnableHighKeeperSDA

LL_I3C_DisableHighKeeperSDA

Function name

__STATIC_INLINE void LL_I3C_DisableHighKeeperSDA (I3C_TypeDef * I3Cx)

Function description

High Keeper is disabled.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Notes

- This bit can only be programmed when the I3C is disabled (EN = 0).

Reference Manual to LL API cross reference:

- CFGR HKSDAEN LL_I3C_DisableHighKeeperSDA

LL_I3C_IsEnabledHighKeeperSDA

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledHighKeeperSDA (const I3C_TypeDef * I3Cx)

Function description

Check if High Keeper is enabled or disabled.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CFGR HKSDAEN LL_I3C_IsEnabledHighKeeperSDA

LL_I3C_EnableHJAck

Function name

__STATIC_INLINE void LL_I3C_EnableHJAck (I3C_TypeDef * I3Cx)

Function description

Hot Join Request is Acked.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Notes

- This bit can be used when I3C is acting as a Controller.

Reference Manual to LL API cross reference:

- CFGR HJACK LL_I3C_EnableHJAck

LL_I3C_DisableHJAck

Function name

__STATIC_INLINE void LL_I3C_DisableHJAck (I3C_TypeDef * I3Cx)

Function description

Hot Join Request is Nacked.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Notes

- This bit can be used when I3C is acting as a Controller.

Reference Manual to LL API cross reference:

- CFGR HJACK LL_I3C_DisableHJAck

LL_I3C_IsEnabledHJAck

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledHJAck (const I3C_TypeDef * I3Cx)

Function description

Check if Hot Join Request Acknowledgement is enabled or disabled.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CFGR HJACK LL_I3C_IsEnabledHJAck

LL_I3C_DMA_GetRegAddr

Function name

__STATIC_INLINE uint32_t LL_I3C_DMA_GetRegAddr (const I3C_TypeDef * I3Cx, uint32_t Direction)

Function description

Get the data register address used for DMA transfer.

Parameters

- **I3Cx:** I3C Instance
- **Direction:** This parameter can be one of the following values:
  - LL_I3C_DMA_REG_DATA_TRANSMIT_BYTE
  - LL_I3C_DMA_REG_DATA_RECEIVE_BYTE
  - LL_I3C_DMA_REG_DATA_TRANSMIT_WORD
  - LL_I3C_DMA_REG_DATA_RECEIVE_WORD
  - LL_I3C_DMA_REG_STATUS
  - LL_I3C_DMA_REG_CONTROL

Return values

- **Address:** of data register

Reference Manual to LL API cross reference:

- TDR TDB0 LL_I3C_DMA_GetRegAddr
- TDWR TDWR LL_I3C_DMA_GetRegAddr
- RDR RXRB0 LL_I3C_DMA_GetRegAddr
- RDWR RDWR LL_I3C_DMA_GetRegAddr
- SR SR LL_I3C_DMA_GetRegAddr
- CR CR LL_I3C_DMA_GetRegAddr

LL_I3C_EnableDMAReq_RX

Function name

__STATIC_INLINE void LL_I3C_EnableDMAReq_RX (I3C_TypeDef * I3Cx)

Function description

Enable DMA FIFO reception requests.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR RXDMAEN LL_I3C_EnableDMAReq_RX

LL_I3C_DisableDMAReq_RX

Function name

__STATIC_INLINE void LL_I3C_DisableDMAReq_RX (I3C_TypeDef * I3Cx)

Function description

Disable DMA FIFO reception requests.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR RXDMAEN LL_I3C_DisableDMAReq_RX

LL_I3C_IsEnabledDMAReq_RX

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledDMAReq_RX (const I3C_TypeDef * I3Cx)

Function description

Check if DMA FIFO reception requests are enabled or disabled.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CFGR RXDMAEN LL_I3C_IsEnabledDMAReq_RX

LL_I3C_SetRxFIFOThreshold

Function name

__STATIC_INLINE void LL_I3C_SetRxFIFOThreshold (I3C_TypeDef * I3Cx, uint32_t RxFIFOThreshold)

Function description

Set the Receive FIFO Threshold level.

Parameters

- **I3Cx:** I3C Instance.
- **RxFIFOThreshold:** This parameter can be one of the following values:
  - LL_I3C_RXFIFO_THRESHOLD_1_4
  - LL_I3C_RXFIFO_THRESHOLD_4_4

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR RXTHRES LL_I3C_SetRxFIFOThreshold

LL_I3C_GetRxFIFOThreshold

Function name

__STATIC_INLINE uint32_t LL_I3C_GetRxFIFOThreshold (const I3C_TypeDef * I3Cx)

Function description

Get the Receive FIFO Threshold level.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Returned:** value can be one of the following values:
  - LL_I3C_RXFIFO_THRESHOLD_1_4
  - LL_I3C_RXFIFO_THRESHOLD_4_4

Reference Manual to LL API cross reference:

- CFGR RXTHRES LL_I3C_GetRxFIFOThreshold

LL_I3C_EnableDMAReq_TX

Function name

__STATIC_INLINE void LL_I3C_EnableDMAReq_TX (I3C_TypeDef * I3Cx)

Function description

Enable DMA FIFO transmission requests.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR TXDMAEN LL_I3C_EnableDMAReq_TX

LL_I3C_DisableDMAReq_TX

Function name

__STATIC_INLINE void LL_I3C_DisableDMAReq_TX (I3C_TypeDef * I3Cx)

Function description

Disable DMA FIFO transmission requests.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR TXDMAEN LL_I3C_DisableDMAReq_TX

LL_I3C_IsEnabledDMAReq_TX

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledDMAReq_TX (const I3C_TypeDef * I3Cx)

Function description

Check if DMA FIFO transmission requests are enabled or disabled.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CFGR TXDMAEN LL_I3C_IsEnabledDMAReq_TX

LL_I3C_SetTxFIFOThreshold

Function name

__STATIC_INLINE void LL_I3C_SetTxFIFOThreshold (I3C_TypeDef * I3Cx, uint32_t TxFIFOThreshold)

Function description

Set the Transmit FIFO Threshold level.

Parameters

- **I3Cx:** I3C Instance.
- **TxFIFOThreshold:** This parameter can be one of the following values:
  - LL_I3C_TXFIFO_THRESHOLD_1_4
  - LL_I3C_TXFIFO_THRESHOLD_4_4

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR TXTHRES LL_I3C_SetTxFIFOThreshold

LL_I3C_GetTxFIFOThreshold

Function name

__STATIC_INLINE uint32_t LL_I3C_GetTxFIFOThreshold (const I3C_TypeDef * I3Cx)

Function description

Get the Transmit FIFO Threshold level.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Returned:** value can be one of the following values:
  - LL_I3C_TXFIFO_THRESHOLD_1_4
  - LL_I3C_TXFIFO_THRESHOLD_4_4
- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CFGR TXTHRES LL_I3C_GetTxFIFOThreshold

LL_I3C_EnableDMAReq_Status

Function name

__STATIC_INLINE void LL_I3C_EnableDMAReq_Status (I3C_TypeDef * I3Cx)

Function description

Enable DMA FIFO Status requests.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR SDMAEN LL_I3C_EnableDMAReq_Status

LL_I3C_DisableDMAReq_Status

Function name

__STATIC_INLINE void LL_I3C_DisableDMAReq_Status (I3C_TypeDef * I3Cx)

Function description

Disable DMA FIFO Status requests.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR SDMAEN LL_I3C_DisableDMAReq_Status

LL_I3C_IsEnabledDMAReq_Status

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledDMAReq_Status (const I3C_TypeDef * I3Cx)

Function description

Check if DMA FIFO Status requests are enabled or disabled.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CFGR SDMAEN LL_I3C_IsEnabledDMAReq_Status

LL_I3C_EnableStatusFIFO

Function name

__STATIC_INLINE void LL_I3C_EnableStatusFIFO (I3C_TypeDef * I3Cx)

Function description

Enable the Status FIFO.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Notes

- Not applicable in target mode. Status FIFO always disabled in target mode.

Reference Manual to LL API cross reference:

- CFGR SMODE LL_I3C_EnableStatusFIFO

LL_I3C_DisableStatusFIFO

Function name

__STATIC_INLINE void LL_I3C_DisableStatusFIFO (I3C_TypeDef * I3Cx)

Function description

Disable the Status FIFO Threshold.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR SMODE LL_I3C_DisableStatusFIFO

LL_I3C_IsEnabledStatusFIFO

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledStatusFIFO (const I3C_TypeDef * I3Cx)

Function description

Check if the Status FIFO Threshold is enabled or disabled.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CFGR SMODE LL_I3C_IsEnabledStatusFIFO

LL_I3C_EnableControlFIFO

Function name

__STATIC_INLINE void LL_I3C_EnableControlFIFO (I3C_TypeDef * I3Cx)

Function description

Enable the Control and Transmit FIFO preloaded before starting a transfer on I3C bus.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Notes

- Not applicable in target mode. Control FIFO always disabled in target mode.

Reference Manual to LL API cross reference:

- CFGR TMODE LL_I3C_EnableControlFIFO

LL_I3C_DisableControlFIFO

Function name

__STATIC_INLINE void LL_I3C_DisableControlFIFO (I3C_TypeDef * I3Cx)

Function description

Disable the Control and Transmit FIFO preloaded before starting a transfer on I3C bus.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR TMODE LL_I3C_DisableControlFIFO

LL_I3C_IsEnabledControlFIFO

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledControlFIFO (const I3C_TypeDef * I3Cx)

Function description

Check if the Control and Transmit FIFO preloaded is enabled or disabled.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CFGR TMODE LL_I3C_IsEnabledControlFIFO

LL_I3C_EnableDMAReq_Control

Function name

__STATIC_INLINE void LL_I3C_EnableDMAReq_Control (I3C_TypeDef * I3Cx)

Function description

Enable DMA FIFO Control word transfer requests.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR CDMAEN LL_I3C_EnableDMAReq_Control

LL_I3C_DisableDMAReq_Control

Function name

__STATIC_INLINE void LL_I3C_DisableDMAReq_Control (I3C_TypeDef * I3Cx)

Function description

Disable DMA FIFO Control word transfer requests.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR CDMAEN LL_I3C_DisableDMAReq_Control

LL_I3C_IsEnabledDMAReq_Control

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledDMAReq_Control (const I3C_TypeDef * I3Cx)

Function description

Check if DMA FIFO Control word transfer requests are enabled or disabled.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CFGR CDMAEN LL_I3C_IsEnabledDMAReq_Control

LL_I3C_EnableOwnDynAddress

Function name

__STATIC_INLINE void LL_I3C_EnableOwnDynAddress (I3C_TypeDef * I3Cx)

Function description

Set Own Dynamic Address as Valid.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DEVR0 DAVAL LL_I3C_EnableOwnDynAddress

LL_I3C_DisableOwnDynAddress

Function name

__STATIC_INLINE void LL_I3C_DisableOwnDynAddress (I3C_TypeDef * I3Cx)

Function description

Set Own Dynamic Address as Not-Valid.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DEVR0 DAVAL LL_I3C_DisableOwnDynAddress

LL_I3C_IsEnabledOwnDynAddress

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledOwnDynAddress (const I3C_TypeDef * I3Cx)

Function description

Check if Own Dynamic address is Valid or Not-Valid.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DEVR0 DAVAL LL_I3C_IsEnabledOwnDynAddress

LL_I3C_SetOwnDynamicAddress

Function name

__STATIC_INLINE void LL_I3C_SetOwnDynamicAddress (I3C_TypeDef * I3Cx, uint32_t OwnDynamicAddress)

Function description

Configure Own Dynamic Address.

Parameters

- **I3Cx:** I3C Instance.
- **OwnDynamicAddress:** This parameter must be a value between Min_Data=0 and Max_Data=0x7F

Return values

- **None:**

Notes

- This bit can be programmed in controller mode or during Dynamic Address procedure from current controller.

Reference Manual to LL API cross reference:

- DEVR0 DA LL_I3C_SetOwnDynamicAddress

LL_I3C_GetOwnDynamicAddress

Function name

__STATIC_INLINE uint8_t LL_I3C_GetOwnDynamicAddress (const I3C_TypeDef * I3Cx)

Function description

Get Own Dynamic Address.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Value:** between Min_Data=0 and Max_Data=0x7F

Reference Manual to LL API cross reference:

- DEVR0 DA LL_I3C_GetOwnDynamicAddress

LL_I3C_EnableIBI

Function name

__STATIC_INLINE void LL_I3C_EnableIBI (I3C_TypeDef * I3Cx)

Function description

Set IBI procedure allowed (when the I3C is acting as target).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Notes

- This bit can be programmed when the I3C is disabled (EN = 0) or updated by HW upon reception of DISEC CCC.

Reference Manual to LL API cross reference:

- DEVR0 IBIEN LL_I3C_EnableIBI

LL_I3C_DisableIBI

Function name

__STATIC_INLINE void LL_I3C_DisableIBI (I3C_TypeDef * I3Cx)

Function description

Set IBI procedure not-allowed (when the I3C is acting as target).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Notes

- This bit can be programmed when the I3C is disabled (EN = 0) or updated by HW upon reception of DISEC CCC.

Reference Manual to LL API cross reference:

- DEVR0 IBIEN LL_I3C_DisableIBI

LL_I3C_IsEnabledIBI

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledIBI (const I3C_TypeDef * I3Cx)

Function description

Check if IBI procedure is allowed or not allowed.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DEVR0 IBIEN LL_I3C_IsEnabledIBI

LL_I3C_EnableControllerRoleReq

Function name

__STATIC_INLINE void LL_I3C_EnableControllerRoleReq (I3C_TypeDef * I3Cx)

Function description

Set Controller-role Request allowed (when the I3C is acting as target).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Notes

- This bit can be programmed when the I3C is disabled (EN = 0) or updated by HW upon reception of DISEC CCC.

Reference Manual to LL API cross reference:

- DEVR0 CREN LL_I3C_EnableControllerRoleReq

LL_I3C_DisableControllerRoleReq

Function name

__STATIC_INLINE void LL_I3C_DisableControllerRoleReq (I3C_TypeDef * I3Cx)

Function description

Set Controller-role Request as not-allowed (when the I3C is acting as target).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Notes

- This bit can be programmed when the I3C is disabled (EN = 0) or updated by HW upon reception of DISEC CCC.

Reference Manual to LL API cross reference:

- DEVR0 CREN LL_I3C_DisableControllerRoleReq

LL_I3C_IsEnabledControllerRoleReq

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledControllerRoleReq (const I3C_TypeDef * I3Cx)

Function description

Check if Controller-role Request is allowed or not-allowed.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DEVR0 CREN LL_I3C_IsEnabledControllerRoleReq

LL_I3C_EnableHotJoin

Function name

__STATIC_INLINE void LL_I3C_EnableHotJoin (I3C_TypeDef * I3Cx)

Function description

Set Hot Join allowed (when the I3C is acting as target).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Notes

- This bit can be programmed when the I3C is disabled (EN = 0) or updated by HW upon reception of DISEC CCC.

Reference Manual to LL API cross reference:

- DEVR0 HJEN LL_I3C_EnableHotJoin

LL_I3C_DisableHotJoin

Function name

__STATIC_INLINE void LL_I3C_DisableHotJoin (I3C_TypeDef * I3Cx)

Function description

Set Hot Join as not-allowed (when the I3C is acting as target).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Notes

- This bit can be programmed when the I3C is disabled (EN = 0) or updated by HW upon reception of DISEC CCC.

Reference Manual to LL API cross reference:

- DEVR0 HJEN LL_I3C_DisableHotJoin

LL_I3C_IsEnabledHotJoin

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledHotJoin (const I3C_TypeDef * I3Cx)

Function description

Check if Hot Join is allowed or not-allowed.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DEVR0 HJEN LL_I3C_IsEnabledHotJoin

LL_I3C_SetMaxReadLength

Function name

__STATIC_INLINE void LL_I3C_SetMaxReadLength (I3C_TypeDef * I3Cx, uint16_t MaxReadLength)

Function description

Configure Maximum Read Length (target mode).

Parameters

- **I3Cx:** I3C Instance.
- **MaxReadLength:** This parameter must be a value between Min_Data=0x0 and Max_Data=0xFFFF

Return values

- **None:**

Notes

- Those bits can be updated by HW upon reception of GETMRL CCC.

Reference Manual to LL API cross reference:

- MAXRLR MRL LL_I3C_SetMaxReadLength

LL_I3C_GetMaxReadLength

Function name

__STATIC_INLINE uint32_t LL_I3C_GetMaxReadLength (const I3C_TypeDef * I3Cx)

Function description

Return Maximum Read Length (target mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Value:** between Min_Data=0x0 and Max_Data=0xFFFFF

Reference Manual to LL API cross reference:

- MAXRLR MRL LL_I3C_GetMaxReadLength

LL_I3C_ConfigNbIBIAddData

Function name

__STATIC_INLINE void LL_I3C_ConfigNbIBIAddData (I3C_TypeDef * I3Cx, uint32_t NbIBIAddData)

Function description

Configure the number of additional Mandatory Data Byte (MDB) sent to the controller after an acknowledge of the IBI (target mode).

Parameters

- **I3Cx:** I3C Instance.
- **NbIBIAddData:** This parameter can be one of the following values:
  - LL_I3C_PAYLOAD_EMPTY
  - LL_I3C_PAYLOAD_1_BYTE
  - LL_I3C_PAYLOAD_2_BYTES
  - LL_I3C_PAYLOAD_3_BYTES
  - LL_I3C_PAYLOAD_4_BYTES

Return values

- **None:**

Reference Manual to LL API cross reference:

- MAXRLR IBIP LL_I3C_ConfigNbIBIAddData

LL_I3C_GetConfigNbIBIAddData

Function name

__STATIC_INLINE uint32_t LL_I3C_GetConfigNbIBIAddData (const I3C_TypeDef * I3Cx)

Function description

Return the number of additional Mandatory Data Byte (MDB) sent to the controller after an acknowledge of the IBI (target mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Returned:** value can be one of the following values:
  - LL_I3C_PAYLOAD_EMPTY
  - LL_I3C_PAYLOAD_1_BYTE
  - LL_I3C_PAYLOAD_2_BYTES
  - LL_I3C_PAYLOAD_3_BYTES
  - LL_I3C_PAYLOAD_4_BYTES

Reference Manual to LL API cross reference:

- MAXRLR IBIP LL_I3C_GetConfigNbIBIAddData

LL_I3C_SetMaxWriteLength

Function name

__STATIC_INLINE void LL_I3C_SetMaxWriteLength (I3C_TypeDef * I3Cx, uint16_t MaxWriteLength)

Function description

Configure Maximum Write Length (target mode).

Parameters

- **I3Cx:** I3C Instance.
- **MaxWriteLength:** This parameter must be a value between Min_Data=0x0 and Max_Data=0xFFFF

Return values

- **None:**

Notes

- Those bits can be updated by HW upon reception of GETMWL CCC.

Reference Manual to LL API cross reference:

- MAXWLR MWL LL_I3C_SetMaxWriteLength

LL_I3C_GetMaxWriteLength

Function name

__STATIC_INLINE uint32_t LL_I3C_GetMaxWriteLength (const I3C_TypeDef * I3Cx)

Function description

Return Maximum Write Length (target mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Value:** between Min_Data=0x0 and Max_Data=0xFFFFF

Reference Manual to LL API cross reference:

- MAXWLR MWL LL_I3C_GetMaxWriteLength

LL_I3C_ConfigClockWaveForm

Function name

__STATIC_INLINE void LL_I3C_ConfigClockWaveForm (I3C_TypeDef * I3Cx, uint32_t ClockWaveForm)

Function description

Configure the SCL clock signal waveform.

Parameters

- **I3Cx:** I3C Instance.
- **ClockWaveForm:** This parameter must be a value between Min_Data=0 and Max_Data=0xFFFFFFFF.

Return values

- **None:**

Notes

- This bit can only be programmed when the I3C is disabled (EN = 0).
- This parameter is computed with the STM32CubeMX Tool.

Reference Manual to LL API cross reference:

- TIMINGR0 TIMINGR0 LL_I3C_ConfigClockWaveForm

LL_I3C_GetClockWaveForm

Function name

__STATIC_INLINE uint32_t LL_I3C_GetClockWaveForm (const I3C_TypeDef * I3Cx)

Function description

Get the SCL clock signal waveform.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Value:** between Min_Data=0 and Max_Data=0xFFFFFFFF.

Reference Manual to LL API cross reference:

- TIMINGR0 TIMINGR0 LL_I3C_GetClockWaveForm

LL_I3C_SetPeriodClockLowPP

Function name

__STATIC_INLINE void LL_I3C_SetPeriodClockLowPP (I3C_TypeDef * I3Cx, uint32_t PeriodClockLowPP)

Function description

Configure the SCL clock low period during I3C push-pull phases.

Parameters

- **I3Cx:** I3C Instance.
- **PeriodClockLowPP:** This parameter must be a value between Min_Data=0 and Max_Data=0xFF.

Return values

- **None:**

Notes

- This bit can only be programmed when the I3C is disabled (EN = 0).
- This parameter is computed with the STM32CubeMX Tool.

Reference Manual to LL API cross reference:

- TIMINGR0 SCLL_PP LL_I3C_SetPeriodClockLowPP

LL_I3C_GetPeriodClockLowPP

Function name

__STATIC_INLINE uint32_t LL_I3C_GetPeriodClockLowPP (const I3C_TypeDef * I3Cx)

Function description

Get the SCL clock low period during I3C push-pull phases.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Value:** between Min_Data=0 and Max_Data=0xFF.

Reference Manual to LL API cross reference:

- TIMINGR0 SCLL_PP LL_I3C_GetPeriodClockLowPP

LL_I3C_SetPeriodClockHighI3C

Function name

__STATIC_INLINE void LL_I3C_SetPeriodClockHighI3C (I3C_TypeDef * I3Cx, uint32_t PeriodClockHighI3C)

Function description

Configure the SCL clock High period during I3C open drain and push-pull phases.

Parameters

- **I3Cx:** I3C Instance.
- **PeriodClockHighI3C:** This parameter must be a value between Min_Data=0 and Max_Data=0xFF.

Return values

- **None:**

Notes

- This bit can only be programmed when the I3C is disabled (EN = 0).
- This parameter is computed with the STM32CubeMX Tool.

Reference Manual to LL API cross reference:

- TIMINGR0 SCLH_I3C LL_I3C_SetPeriodClockHighI3C

LL_I3C_GetPeriodClockHighI3C

Function name

__STATIC_INLINE uint32_t LL_I3C_GetPeriodClockHighI3C (const I3C_TypeDef * I3Cx)

Function description

Get the SCL clock high period during I3C open drain and push-pull phases.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Value:** between Min_Data=0 and Max_Data=0xFF.

Reference Manual to LL API cross reference:

- TIMINGR0 SCLH_I3C LL_I3C_GetPeriodClockHighI3C

LL_I3C_SetPeriodClockLowOD

Function name

__STATIC_INLINE void LL_I3C_SetPeriodClockLowOD (I3C_TypeDef * I3Cx, uint32_t PeriodClockLowOD)

Function description

Configure the SCL clock low period during I3C open drain phases.

Parameters

- **I3Cx:** I3C Instance.
- **PeriodClockLowOD:** This parameter must be a value between Min_Data=0 and Max_Data=0xFF.

Return values

- **None:**

Notes

- This bit can only be programmed when the I3C is disabled (EN = 0).
- This parameter is computed with the STM32CubeMX Tool.

Reference Manual to LL API cross reference:

- TIMINGR0 SCLL_OD LL_I3C_SetPeriodClockLowOD

LL_I3C_GetPeriodClockLowOD

Function name

__STATIC_INLINE uint32_t LL_I3C_GetPeriodClockLowOD (const I3C_TypeDef * I3Cx)

Function description

Get the SCL clock low period during I3C open phases.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Value:** between Min_Data=0 and Max_Data=0xFF.

Reference Manual to LL API cross reference:

- TIMINGR0 SCLL_OD LL_I3C_GetPeriodClockLowOD

LL_I3C_SetPeriodClockHighI2C

Function name

__STATIC_INLINE void LL_I3C_SetPeriodClockHighI2C (I3C_TypeDef * I3Cx, uint32_t PeriodClockHighI2C)

Function description

Configure the SCL clock High period during I2C open drain phases.

Parameters

- **I3Cx:** I3C Instance.
- **PeriodClockHighI2C:** This parameter must be a value between Min_Data=0 and Max_Data=0xFF.

Return values

- **None:**

Notes

- This bit can only be programmed when the I3C is disabled (EN = 0).
- This parameter is computed with the STM32CubeMX Tool.

Reference Manual to LL API cross reference:

- TIMINGR0 SCLH_I2C LL_I3C_SetPeriodClockHighI2C

LL_I3C_GetPeriodClockHighI2C

Function name

__STATIC_INLINE uint32_t LL_I3C_GetPeriodClockHighI2C (const I3C_TypeDef * I3Cx)

Function description

Get the SCL clock high period during I2C open drain phases.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Value:** between Min_Data=0 and Max_Data=0xFF.

Reference Manual to LL API cross reference:

- TIMINGR0 SCLH_I2C LL_I3C_GetPeriodClockHighI2C

LL_I3C_SetDataHoldTime

Function name

__STATIC_INLINE void LL_I3C_SetDataHoldTime (I3C_TypeDef * I3Cx, uint32_t DataHoldTime)

Function description

Configure the Controller additional hold time on SDA line.

Parameters

- **I3Cx:** I3C Instance.
- **DataHoldTime:** This parameter can be one of the following values:
  - LL_I3C_SDA_HOLD_TIME_0_5
  - LL_I3C_SDA_HOLD_TIME_1_5

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMINGR1 SDA_HD LL_I3C_SetDataHoldTime

LL_I3C_GetDataHoldTime

Function name

__STATIC_INLINE uint32_t LL_I3C_GetDataHoldTime (const I3C_TypeDef * I3Cx)

Function description

Get the Controller additional hold time on SDA line.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Returned:** value can be one of the following values:
  - LL_I3C_SDA_HOLD_TIME_0_5
  - LL_I3C_SDA_HOLD_TIME_1_5

Reference Manual to LL API cross reference:

- TIMINGR1 SDA_HD LL_I3C_GetDataHoldTime

LL_I3C_SetAvalTiming

Function name

__STATIC_INLINE void LL_I3C_SetAvalTiming (I3C_TypeDef * I3Cx, uint32_t AvalTiming)

Function description

Configure the Idle, Available state.

Parameters

- **I3Cx:** I3C Instance.
- **AvalTiming:** This parameter must be a value between Min_Data=0 and Max_Data=0xFF.

Return values

- **None:**

Notes

- This bit can only be programmed when the I3C is disabled (EN = 0).
- This parameter is computed with the STM32CubeMX Tool.

Reference Manual to LL API cross reference:

- TIMINGR1 AVAL LL_I3C_SetAvalTiming

LL_I3C_GetAvalTiming

Function name

__STATIC_INLINE uint32_t LL_I3C_GetAvalTiming (const I3C_TypeDef * I3Cx)

Function description

Get the Idle, Available integer value state.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Value:** between Min_Data=0 and Max_Data=0xFF.

Reference Manual to LL API cross reference:

- TIMINGR1 AVAL LL_I3C_GetAvalTiming

LL_I3C_SetFreeTiming

Function name

__STATIC_INLINE void LL_I3C_SetFreeTiming (I3C_TypeDef * I3Cx, uint32_t FreeTiming)

Function description

Configure the Free state.

Parameters

- **I3Cx:** I3C Instance.
- **FreeTiming:** This parameter must be a value between Min_Data=0 and Max_Data=0x3F.

Return values

- **None:**

Notes

- This bit can only be programmed when the I3C is disabled (EN = 0).
- This parameter is computed with the STM32CubeMX Tool.

Reference Manual to LL API cross reference:

- TIMINGR1 FREE LL_I3C_SetFreeTiming

LL_I3C_GetFreeTiming

Function name

__STATIC_INLINE uint32_t LL_I3C_GetFreeTiming (const I3C_TypeDef * I3Cx)

Function description

Get the Free integeter value state.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Value:** between Min_Data=0 and Max_Data=0x3F.

Reference Manual to LL API cross reference:

- TIMINGR1 FREE LL_I3C_GetFreeTiming

LL_I3C_SetControllerActivityState

Function name

__STATIC_INLINE void LL_I3C_SetControllerActivityState (I3C_TypeDef * I3Cx, uint32_t ControllerActivityState)

Function description

Configure the activity state of the new controller.

Parameters

- **I3Cx:** I3C Instance.
- **ControllerActivityState:** This parameter can be one of the following values:
  - LL_I3C_OWN_ACTIVITY_STATE_0
  - LL_I3C_OWN_ACTIVITY_STATE_1
  - LL_I3C_OWN_ACTIVITY_STATE_2
  - LL_I3C_OWN_ACTIVITY_STATE_3

Return values

- **None:**

Notes

- Refer to MIPI I3C specification (https:__www.mipi.org_specifications) for more details related to Activity State.

Reference Manual to LL API cross reference:

- TIMINGR1 ASNCR LL_I3C_SetControllerActivityState

LL_I3C_GetControllerActivityState

Function name

__STATIC_INLINE uint32_t LL_I3C_GetControllerActivityState (const I3C_TypeDef * I3Cx)

Function description

Get the activity state of the new controller.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Returned:** value can be one of the following values:
  - LL_I3C_OWN_ACTIVITY_STATE_0
  - LL_I3C_OWN_ACTIVITY_STATE_1
  - LL_I3C_OWN_ACTIVITY_STATE_2
  - LL_I3C_OWN_ACTIVITY_STATE_3

Notes

- Refer to MIPI I3C specification (https:__www.mipi.org_specifications) for more details related to Activity State.

Reference Manual to LL API cross reference:

- TIMINGR1 ASNCR LL_I3C_GetControllerActivityState

LL_I3C_SetCtrlBusCharacteristic

Function name

__STATIC_INLINE void LL_I3C_SetCtrlBusCharacteristic (I3C_TypeDef * I3Cx, uint32_t CtrlBusCharacteristic)

Function description

Configure the Controller SDA Hold time, Bus Free, Activity state, Idle state.

Parameters

- **I3Cx:** I3C Instance.
- **CtrlBusCharacteristic:** This parameter must be a value between Min_Data=0 and Max_Data=0x107F03FF.

Return values

- **None:**

Notes

- This bit can only be programmed when the I3C is disabled (EN = 0).
- This parameter is computed with the STM32CubeMX Tool.

Reference Manual to LL API cross reference:

- TIMINGR1 SDA_HD LL_I3C_SetCtrlBusCharacteristic
- TIMINGR1 FREE LL_I3C_SetCtrlBusCharacteristic
- TIMINGR1 ASNCR LL_I3C_SetCtrlBusCharacteristic
- TIMINGR1 IDLE LL_I3C_SetCtrlBusCharacteristic

LL_I3C_GetCtrlBusCharacteristic

Function name

__STATIC_INLINE uint32_t LL_I3C_GetCtrlBusCharacteristic (const I3C_TypeDef * I3Cx)

Function description

Get the Controller SDA Hold time, Bus Free, Activity state, Idle state.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Value:** between Min_Data=0 and Max_Data=0x107F03FF.

Reference Manual to LL API cross reference:

- TIMINGR1 SDA_HD LL_I3C_GetCtrlBusCharacteristic
- TIMINGR1 FREE LL_I3C_GetCtrlBusCharacteristic
- TIMINGR1 ASNCR LL_I3C_GetCtrlBusCharacteristic
- TIMINGR1 IDLE LL_I3C_GetCtrlBusCharacteristic

LL_I3C_SetTgtBusCharacteristic

Function name

__STATIC_INLINE void LL_I3C_SetTgtBusCharacteristic (I3C_TypeDef * I3Cx, uint32_t TgtBusCharacteristic)

Function description

Configure the Target Available state.

Parameters

- **I3Cx:** I3C Instance.
- **TgtBusCharacteristic:** This parameter must be a value between Min_Data=0 and Max_Data=0xFF.

Return values

- **None:**

Notes

- This bit can only be programmed when the I3C is disabled (EN = 0).
- This parameter is computed with the STM32CubeMX Tool.

Reference Manual to LL API cross reference:

- TIMINGR1 IDLE LL_I3C_SetTgtBusCharacteristic

LL_I3C_GetTgtBusCharacteristic

Function name

__STATIC_INLINE uint32_t LL_I3C_GetTgtBusCharacteristic (const I3C_TypeDef * I3Cx)

Function description

Get the Target Available state.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Value:** between Min_Data=0 and Max_Data=0xFF.

Reference Manual to LL API cross reference:

- TIMINGR1 IDLE LL_I3C_GetTgtBusCharacteristic

LL_I3C_SetStallTime

Function name

__STATIC_INLINE void LL_I3C_SetStallTime (I3C_TypeDef * I3Cx, uint32_t ControllerStallTime)

Function description

Configure the SCL clock stalling time on I3C Bus (controller mode).

Parameters

- **I3Cx:** I3C Instance.
- **ControllerStallTime:** This parameter must be a value between Min_Data=0 and Max_Data=0xFF.

Return values

- **None:**

Notes

- This bit can only be programmed when the I3C is disabled (EN = 0).
- This parameter is computed with the STM32CubeMX Tool.

Reference Manual to LL API cross reference:

- TIMINGR2 STALL LL_I3C_SetStallTime

LL_I3C_GetStallTime

Function name

__STATIC_INLINE uint32_t LL_I3C_GetStallTime (const I3C_TypeDef * I3Cx)

Function description

Get the SCL clock stalling time on I3C Bus (controller mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Value:** between Min_Data=0 and Max_Data=0xFF.

Reference Manual to LL API cross reference:

- TIMINGR2 STALL LL_I3C_GetStallTime

LL_I3C_EnableStallACK

Function name

__STATIC_INLINE void LL_I3C_EnableStallACK (I3C_TypeDef * I3Cx)

Function description

Set stall on ACK bit (controller mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Notes

- This bit can be programmed when the I3C is disabled (EN = 0).

Reference Manual to LL API cross reference:

- TIMINGR2 STALLA LL_I3C_EnableStallACK

LL_I3C_DisableStallACK

Function name

__STATIC_INLINE void LL_I3C_DisableStallACK (I3C_TypeDef * I3Cx)

Function description

Disable stall on ACK bit (controller mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Notes

- This bit can be programmed when the I3C is disabled (EN = 0).

Reference Manual to LL API cross reference:

- TIMINGR2 STALLA LL_I3C_DisableStallACK

LL_I3C_IsEnabledStallACK

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledStallACK (const I3C_TypeDef * I3Cx)

Function description

Check if stall on ACK bit is enabled or disabled (controller mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TIMINGR2 STALLA LL_I3C_IsEnabledStallACK

LL_I3C_EnableStallParityCCC

Function name

__STATIC_INLINE void LL_I3C_EnableStallParityCCC (I3C_TypeDef * I3Cx)

Function description

Set stall on Parity bit of Command Code byte (controller mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Notes

- This bit can be programmed when the I3C is disabled (EN = 0).

Reference Manual to LL API cross reference:

- TIMINGR2 STALLC LL_I3C_EnableStallParityCCC

LL_I3C_DisableStallParityCCC

Function name

__STATIC_INLINE void LL_I3C_DisableStallParityCCC (I3C_TypeDef * I3Cx)

Function description

Disable stall on Parity bit of Command Code byte (controller mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Notes

- This bit can be programmed when the I3C is disabled (EN = 0).

Reference Manual to LL API cross reference:

- TIMINGR2 STALLC LL_I3C_DisableStallParityCCC

LL_I3C_IsEnabledStallParityCCC

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledStallParityCCC (const I3C_TypeDef * I3Cx)

Function description

Check if stall on Parity bit of Command Code byte is enabled or disabled (controller mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TIMINGR2 STALLC LL_I3C_IsEnabledStallParityCCC

LL_I3C_EnableStallParityData

Function name

__STATIC_INLINE void LL_I3C_EnableStallParityData (I3C_TypeDef * I3Cx)

Function description

Set stall on Parity bit of Data bytes (controller mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Notes

- This bit can be programmed when the I3C is disabled (EN = 0).

Reference Manual to LL API cross reference:

- TIMINGR2 STALLD LL_I3C_EnableStallParityData

LL_I3C_DisableStallParityData

Function name

__STATIC_INLINE void LL_I3C_DisableStallParityData (I3C_TypeDef * I3Cx)

Function description

Disable stall on Parity bit of Data bytes (controller mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Notes

- This bit can be programmed when the I3C is disabled (EN = 0).

Reference Manual to LL API cross reference:

- TIMINGR2 STALLD LL_I3C_DisableStallParityData

LL_I3C_IsEnabledStallParityData

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledStallParityData (const I3C_TypeDef * I3Cx)

Function description

Check if stall on Parity bit of Data bytes is enabled or disabled (controller mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TIMINGR2 STALLD LL_I3C_IsEnabledStallParityData

LL_I3C_EnableStallTbit

Function name

__STATIC_INLINE void LL_I3C_EnableStallTbit (I3C_TypeDef * I3Cx)

Function description

Set stall on T bit (controller mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Notes

- This bit can be programmed when the I3C is disabled (EN = 0).

Reference Manual to LL API cross reference:

- TIMINGR2 STALLT LL_I3C_EnableStallTbit

LL_I3C_DisableStallTbit

Function name

__STATIC_INLINE void LL_I3C_DisableStallTbit (I3C_TypeDef * I3Cx)

Function description

Disable stall on T bit (controller mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Notes

- This bit can be programmed when the I3C is disabled (EN = 0).

Reference Manual to LL API cross reference:

- TIMINGR2 STALLT LL_I3C_DisableStallTbit

LL_I3C_IsEnabledStallTbit

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledStallTbit (const I3C_TypeDef * I3Cx)

Function description

Check if stall on T bit is enabled or disabled (controller mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TIMINGR2 STALLT LL_I3C_IsEnabledStallTbit

LL_I3C_SetDeviceCapabilityOnBus

Function name

__STATIC_INLINE void LL_I3C_SetDeviceCapabilityOnBus (I3C_TypeDef * I3Cx, uint32_t DeviceCapabilityOnBus)

Function description

Configure the Device Capability on Bus as Target or Controller (MIPI Bus Characteristics Register BCR6).

Parameters

- **I3Cx:** I3C Instance.
- **DeviceCapabilityOnBus:** This parameter can be one of the following values:
  - LL_I3C_DEVICE_ROLE_AS_TARGET
  - LL_I3C_DEVICE_ROLE_AS_CONTROLLER

Return values

- **None:**

Notes

- Those bits can be programmed when the I3C is disabled (EN = 0).

Reference Manual to LL API cross reference:

- BCR BCR6 LL_I3C_SetDeviceCapabilityOnBus

LL_I3C_GetDeviceCapabilityOnBus

Function name

__STATIC_INLINE uint32_t LL_I3C_GetDeviceCapabilityOnBus (const I3C_TypeDef * I3Cx)

Function description

Get the Device Capability on Bus as Target or Controller (MIPI Bus Characteristics Register BCR6).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Returned:** value can be one of the following values:
  - LL_I3C_DEVICE_ROLE_AS_TARGET
  - LL_I3C_DEVICE_ROLE_AS_CONTROLLER

Reference Manual to LL API cross reference:

- BCR BCR6 LL_I3C_GetDeviceCapabilityOnBus

LL_I3C_SetDeviceIBIPayload

Function name

__STATIC_INLINE void LL_I3C_SetDeviceIBIPayload (I3C_TypeDef * I3Cx, uint32_t DeviceIBIPayload)

Function description

Configure the Device IBI Payload (MIPI Bus Characteristics Register BCR2).

Parameters

- **I3Cx:** I3C Instance.
- **DeviceIBIPayload:** This parameter can be one of the following values:
  - LL_I3C_IBI_NO_ADDITIONAL_DATA
  - LL_I3C_IBI_ADDITIONAL_DATA

Return values

- **None:**

Notes

- Those bits can be programmed when the I3C is disabled (EN = 0).

Reference Manual to LL API cross reference:

- BCR BCR2 LL_I3C_SetDeviceIBIPayload

LL_I3C_GetDeviceIBIPayload

Function name

__STATIC_INLINE uint32_t LL_I3C_GetDeviceIBIPayload (const I3C_TypeDef * I3Cx)

Function description

Get the Device IBI Payload (MIPI Bus Characteristics Register BCR2).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Returned:** value can be one of the following values:
  - LL_I3C_IBI_NO_ADDITIONAL_DATA
  - LL_I3C_IBI_ADDITIONAL_DATA

Reference Manual to LL API cross reference:

- BCR BCR2 LL_I3C_GetDeviceIBIPayload

LL_I3C_SetDataSpeedLimitation

Function name

__STATIC_INLINE void LL_I3C_SetDataSpeedLimitation (I3C_TypeDef * I3Cx, uint32_t DataSpeedLimitation)

Function description

Configure the Data Speed Limitation (limitation, as described by I3C_GETMXDSR).

Parameters

- **I3Cx:** I3C Instance.
- **DataSpeedLimitation:** This parameter can be one of the following values:
  - LL_I3C_NO_DATA_SPEED_LIMITATION
  - LL_I3C_MAX_DATA_SPEED_LIMITATION

Return values

- **None:**

Notes

- Those bits can be programmed when the I3C is disabled (EN = 0).

Reference Manual to LL API cross reference:

- BCR BCR0 LL_I3C_SetDataSpeedLimitation

LL_I3C_GetDataSpeedLimitation

Function name

__STATIC_INLINE uint32_t LL_I3C_GetDataSpeedLimitation (const I3C_TypeDef * I3Cx)

Function description

Get the Data Speed Limitation (limitation, as described by I3C_GETMXDSR).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Returned:** value can be one of the following values:
  - LL_I3C_NO_DATA_SPEED_LIMITATION
  - LL_I3C_MAX_DATA_SPEED_LIMITATION

Reference Manual to LL API cross reference:

- BCR BCR0 LL_I3C_GetDataSpeedLimitation

LL_I3C_SetDeviceCharacteristics

Function name

__STATIC_INLINE void LL_I3C_SetDeviceCharacteristics (I3C_TypeDef * I3Cx, uint32_t DeviceCharacteristics)

Function description

Configure the Device Characteristics Register (DCR).

Parameters

- **I3Cx:** I3C Instance.
- **DeviceCharacteristics:** This parameter must be a value between Min_Data=0 and Max_Data=0xFF.

Return values

- **None:**

Notes

- This bit can only be programmed when the I3C is disabled (EN = 0).
- Refer MIPI web site for the list of device code available.

Reference Manual to LL API cross reference:

- DCR DC LL_I3C_SetDeviceCharacteristics

LL_I3C_GetDeviceCharacteristics

Function name

__STATIC_INLINE uint32_t LL_I3C_GetDeviceCharacteristics (const I3C_TypeDef * I3Cx)

Function description

Get the Device Characteristics Register (DCR).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Value:** between Min_Data=0 and Max_Data=0xFF.

Notes

- Refer MIPI web site to associated value with the list of device code available.

Reference Manual to LL API cross reference:

- DCR DCR LL_I3C_GetDeviceCharacteristics

LL_I3C_SetPendingReadMDB

Function name

__STATIC_INLINE void LL_I3C_SetPendingReadMDB (I3C_TypeDef * I3Cx, uint32_t PendingReadMDB)

Function description

Configure IBI MDB support for pending read notification.

Parameters

- **I3Cx:** I3C Instance.
- **PendingReadMDB:** This parameter can be one of the following values:
  - LL_I3C_MDB_NO_PENDING_READ_NOTIFICATION
  - LL_I3C_MDB_PENDING_READ_NOTIFICATION

Return values

- **None:**

Notes

- Those bits can be programmed when the I3C is disabled (EN = 0).

Reference Manual to LL API cross reference:

- GETCAPR CAPPEND LL_I3C_SetPendingReadMDB

LL_I3C_GetPendingReadMDB

Function name

__STATIC_INLINE uint32_t LL_I3C_GetPendingReadMDB (const I3C_TypeDef * I3Cx)

Function description

Get IBI MDB support for pending read notification value.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Returned:** value can be one of the following values:
  - LL_I3C_MDB_NO_PENDING_READ_NOTIFICATION
  - LL_I3C_MDB_PENDING_READ_NOTIFICATION

Reference Manual to LL API cross reference:

- GETCAPR CAPPEND LL_I3C_GetPendingReadMDB

LL_I3C_SetGrpAddrHandoffSupport

Function name

__STATIC_INLINE void LL_I3C_SetGrpAddrHandoffSupport (I3C_TypeDef * I3Cx, uint32_t GrpAddrHandoffSupport)

Function description

Configure the Group Management Support bit of MSTCAP1.

Parameters

- **I3Cx:** I3C Instance.
- **GrpAddrHandoffSupport:** This parameter can be one of the following values:
  - LL_I3C_HANDOFF_GRP_ADDR_NOT_SUPPORTED
  - LL_I3C_HANDOFF_GRP_ADDR_SUPPORTED

Return values

- **None:**

Notes

- Those bits can be programmed when the I3C is disabled (EN = 0).

Reference Manual to LL API cross reference:

- CRCAPR CAPGRP LL_I3C_SetGrpAddrHandoffSupport

LL_I3C_GetGrpAddrHandoffSupport

Function name

__STATIC_INLINE uint32_t LL_I3C_GetGrpAddrHandoffSupport (const I3C_TypeDef * I3Cx)

Function description

Get the Group Management Support bit of MSTCAP1.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Returned:** value can be one of the following values:
  - LL_I3C_HANDOFF_GRP_ADDR_NOT_SUPPORTED
  - LL_I3C_HANDOFF_GRP_ADDR_SUPPORTED

Reference Manual to LL API cross reference:

- CRCAPR CAPGRP LL_I3C_GetGrpAddrHandoffSupport

LL_I3C_SetControllerHandoffDelayed

Function name

__STATIC_INLINE void LL_I3C_SetControllerHandoffDelayed (I3C_TypeDef * I3Cx, uint32_t ControllerHandoffDelayed)

Function description

Configure the Delayed Controller Handoff bit in MSTCAP2.

Parameters

- **I3Cx:** I3C Instance.
- **ControllerHandoffDelayed:** This parameter can be one of the following values:
  - LL_I3C_HANDOFF_NOT_DELAYED
  - LL_I3C_HANDOFF_DELAYED

Return values

- **None:**

Notes

- Those bits can be programmed when the I3C is disabled (EN = 0).

Reference Manual to LL API cross reference:

- CRCAPR CAPDHOFF LL_I3C_SetControllerHandoffDelayed

LL_I3C_GetControllerHandoffDelayed

Function name

__STATIC_INLINE uint32_t LL_I3C_GetControllerHandoffDelayed (const I3C_TypeDef * I3Cx)

Function description

Get the Delayed Controller Handoff bit in MSTCAP2.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Returned:** value can be one of the following values:
  - LL_I3C_HANDOFF_NOT_DELAYED
  - LL_I3C_HANDOFF_DELAYED

Reference Manual to LL API cross reference:

- CRCAPR CAPDHOFF LL_I3C_GetControllerHandoffDelayed

LL_I3C_SetHandoffActivityState

Function name

__STATIC_INLINE void LL_I3C_SetHandoffActivityState (I3C_TypeDef * I3Cx, uint32_t HandoffActivityState)

Function description

Configure the Activity State after controllership handoff.

Parameters

- **I3Cx:** I3C Instance.
- **HandoffActivityState:** This parameter can be one of the following values:
  - LL_I3C_HANDOFF_ACTIVITY_STATE_0
  - LL_I3C_HANDOFF_ACTIVITY_STATE_1
  - LL_I3C_HANDOFF_ACTIVITY_STATE_2
  - LL_I3C_HANDOFF_ACTIVITY_STATE_3

Return values

- **None:**

Notes

- Those bits can be programmed when the I3C is disabled (EN = 0).

Reference Manual to LL API cross reference:

- GETMXDSR HOFFAS LL_I3C_SetHandoffActivityState

LL_I3C_GetHandoffActivityState

Function name

__STATIC_INLINE uint32_t LL_I3C_GetHandoffActivityState (const I3C_TypeDef * I3Cx)

Function description

Get the Activity State after controllership handoff.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Returned:** value can be one of the following values:
  - LL_I3C_HANDOFF_ACTIVITY_STATE_0
  - LL_I3C_HANDOFF_ACTIVITY_STATE_1
  - LL_I3C_HANDOFF_ACTIVITY_STATE_2
  - LL_I3C_HANDOFF_ACTIVITY_STATE_3

Reference Manual to LL API cross reference:

- GETMXDSR HOFFAS LL_I3C_GetHandoffActivityState

LL_I3C_SetMaxDataSpeedFormat

Function name

__STATIC_INLINE void LL_I3C_SetMaxDataSpeedFormat (I3C_TypeDef * I3Cx, uint32_t MaxDataSpeedFormat)

Function description

Configure the Max Data Speed Format response for GETMXDS CCC.

Parameters

- **I3Cx:** I3C Instance.
- **MaxDataSpeedFormat:** This parameter can be one of the following values:
  - LL_I3C_GETMXDS_FORMAT_1
  - LL_I3C_GETMXDS_FORMAT_2_LSB
  - LL_I3C_GETMXDS_FORMAT_2_MID
  - LL_I3C_GETMXDS_FORMAT_2_MSB

Return values

- **None:**

Notes

- Those bits can be programmed when the I3C is disabled (EN = 0).

Reference Manual to LL API cross reference:

- GETMXDSR FMT LL_I3C_SetMaxDataSpeedFormat

LL_I3C_GetMaxDataSpeedFormat

Function name

__STATIC_INLINE uint32_t LL_I3C_GetMaxDataSpeedFormat (const I3C_TypeDef * I3Cx)

Function description

Get the Max Data Speed Format response for GETMXDS CCC.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Returned:** value can be one of the following values:
  - LL_I3C_GETMXDS_FORMAT_1
  - LL_I3C_GETMXDS_FORMAT_2_LSB
  - LL_I3C_GETMXDS_FORMAT_2_MID
  - LL_I3C_GETMXDS_FORMAT_2_MSB

Reference Manual to LL API cross reference:

- GETMXDSR FMT LL_I3C_GetMaxDataSpeedFormat

LL_I3C_SetMiddleByteTurnAround

Function name

__STATIC_INLINE void LL_I3C_SetMiddleByteTurnAround (I3C_TypeDef * I3Cx, uint32_t MiddleByteTurnAround)

Function description

Configure the Middle byte of MaxRdTurn field of GETMXDS CCC Format 2 with turnaround.

Parameters

- **I3Cx:** I3C Instance.
- **MiddleByteTurnAround:** This parameter must be a value between Min_Data=0 and Max_Data=0xF.

Return values

- **None:**

Notes

- Those bits can be programmed when the I3C is disabled (EN = 0).

Reference Manual to LL API cross reference:

- GETMXDSR RDTURN LL_I3C_SetMiddleByteTurnAround

LL_I3C_GetMiddleByteTurnAround

Function name

__STATIC_INLINE uint32_t LL_I3C_GetMiddleByteTurnAround (const I3C_TypeDef * I3Cx)

Function description

Get the value of Middle byte of MaxRdTurn field of GETMXDS CCC Format 2 with turnaround.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Value:** between Min_Data=0 and Max_Data=0xF.

Reference Manual to LL API cross reference:

- GETMXDSR RDTURN LL_I3C_GetMiddleByteTurnAround

LL_I3C_SetDataTurnAroundTime

Function name

__STATIC_INLINE void LL_I3C_SetDataTurnAroundTime (I3C_TypeDef * I3Cx, uint32_t DataTurnAroundTime)

Function description

Configure clock-to-data turnaround time.

Parameters

- **I3Cx:** I3C Instance.
- **DataTurnAroundTime:** This parameter can be one of the following values:
  - LL_I3C_TURNAROUND_TIME_TSCO_LESS_12NS
  - LL_I3C_TURNAROUND_TIME_TSCO_GREATER_12NS

Return values

- **None:**

Notes

- Those bits can be programmed when the I3C is disabled (EN = 0).

Reference Manual to LL API cross reference:

- GETMXDSR TSCO LL_I3C_SetDataTurnAroundTime

LL_I3C_GetDataTurnAroundTime

Function name

__STATIC_INLINE uint32_t LL_I3C_GetDataTurnAroundTime (const I3C_TypeDef * I3Cx)

Function description

Get clock-to-data turnaround time.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Returned:** value can be one of the following values:
  - LL_I3C_TURNAROUND_TIME_TSCO_LESS_12NS
  - LL_I3C_TURNAROUND_TIME_TSCO_GREATER_12NS

Reference Manual to LL API cross reference:

- GETMXDSR TSCO LL_I3C_GetDataTurnAroundTime

LL_I3C_SetMIPIInstanceID

Function name

__STATIC_INLINE void LL_I3C_SetMIPIInstanceID (I3C_TypeDef * I3Cx, uint32_t MIPIInstanceID)

Function description

Configure the MIPI Instance ID.

Parameters

- **I3Cx:** I3C Instance.
- **MIPIInstanceID:** This parameter must be a value between Min_Data=0 and Max_Data=0xF.

Return values

- **None:**

Notes

- Those bits can be programmed when the I3C is disabled (EN = 0).

Reference Manual to LL API cross reference:

- EPIDR MIPIID LL_I3C_SetMIPIInstanceID

LL_I3C_GetMIPIInstanceID

Function name

__STATIC_INLINE uint32_t LL_I3C_GetMIPIInstanceID (const I3C_TypeDef * I3Cx)

Function description

Get the MIPI Instance ID.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Value:** between Min_Data=0 and Max_Data=0xF.

Reference Manual to LL API cross reference:

- EPIDR MIPIID LL_I3C_GetMIPIInstanceID

LL_I3C_GetIDTypeSelector

Function name

__STATIC_INLINE uint32_t LL_I3C_GetIDTypeSelector (const I3C_TypeDef * I3Cx)

Function description

Get the ID type selector.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Value:** between Min_Data=0x0 and Max_Data=0x1

Reference Manual to LL API cross reference:

- EPIDR IDTSEL LL_I3C_GetIDTypeSelector

LL_I3C_GetMIPIManufacturerID

Function name

__STATIC_INLINE uint32_t LL_I3C_GetMIPIManufacturerID (const I3C_TypeDef * I3Cx)

Function description

Get the MIPI Manufacturer ID.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Value:** between Min_Data=0 and Max_Data=0x7FFF.

Reference Manual to LL API cross reference:

- EPIDR MIPIMID LL_I3C_GetMIPIManufacturerID

LL_I3C_RequestRxFIFOFlush

Function name

__STATIC_INLINE void LL_I3C_RequestRxFIFOFlush (I3C_TypeDef * I3Cx)

Function description

Request a reception Data FIFO Flush.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR RXFLUSH LL_I3C_RequestRxFIFOFlush

LL_I3C_RequestTxFIFOFlush

Function name

__STATIC_INLINE void LL_I3C_RequestTxFIFOFlush (I3C_TypeDef * I3Cx)

Function description

Request a transmission Data FIFO Flush.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR TXFLUSH LL_I3C_RequestTxFIFOFlush

LL_I3C_RequestStatusFIFOFlush

Function name

__STATIC_INLINE void LL_I3C_RequestStatusFIFOFlush (I3C_TypeDef * I3Cx)

Function description

Request a Status Data FIFO Flush.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR SFLUSH LL_I3C_RequestStatusFIFOFlush

LL_I3C_GetActivityState

Function name

__STATIC_INLINE uint32_t LL_I3C_GetActivityState (const I3C_TypeDef * I3Cx)

Function description

Get Activity state of Controller on the I3C Bus (Target only).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Returned:** value can be one of the following values:
  - LL_I3C_BUS_ACTIVITY_STATE_0
  - LL_I3C_BUS_ACTIVITY_STATE_1
  - LL_I3C_BUS_ACTIVITY_STATE_2
  - LL_I3C_BUS_ACTIVITY_STATE_3

Reference Manual to LL API cross reference:

- DEVR0 AS LL_I3C_GetActivityState

LL_I3C_GetResetAction

Function name

__STATIC_INLINE uint32_t LL_I3C_GetResetAction (const I3C_TypeDef * I3Cx)

Function description

Get Reset Action (Target only).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Returned:** value can be one of the following values:
  - LL_I3C_RESET_ACTION_NONE
  - LL_I3C_RESET_ACTION_PARTIAL
  - LL_I3C_RESET_ACTION_FULL

Reference Manual to LL API cross reference:

- DEVR0 RSTACT LL_I3C_GetResetAction

LL_I3C_RequestControlFIFOFlush

Function name

__STATIC_INLINE void LL_I3C_RequestControlFIFOFlush (I3C_TypeDef * I3Cx)

Function description

Request a Control word FIFO Flush.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR CFLUSH LL_I3C_RequestControlFIFOFlush

LL_I3C_RequestTransfer

Function name

__STATIC_INLINE void LL_I3C_RequestTransfer (I3C_TypeDef * I3Cx)

Function description

Request a Transfer start.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Notes

- After request, the current instruction in Control Register is executed on I3C Bus.

Reference Manual to LL API cross reference:

- CFGR TSFSET LL_I3C_RequestTransfer

LL_I3C_ControllerHandleMessage

Function name

__STATIC_INLINE void LL_I3C_ControllerHandleMessage (I3C_TypeDef * I3Cx, uint32_t TargetAddr, uint32_t TransferSize, uint32_t Direction, uint32_t MessageType, uint32_t EndMode)

Function description

Handles I3C Message content on the I3C Bus as Controller.

Parameters

- **I3Cx:** I3C Instance.
- **TargetAddr:** Specifies the target address to be programmed. This parameter must be a value between Min_Data=0 and Max_Data=0xFF.
- **TransferSize:** Specifies the number of bytes to be programmed. This parameter must be a value between Min_Data=0 and Max_Data=65535.
- **Direction:** This parameter can be one of the following values:
  - LL_I3C_DIRECTION_WRITE
  - LL_I3C_DIRECTION_READ
- **MessageType:** This parameter can be one of the following values:
  - LL_I3C_CONTROLLER_MTYPE_RELEASE
  - LL_I3C_CONTROLLER_MTYPE_HEADER
  - LL_I3C_CONTROLLER_MTYPE_PRIVATE
  - LL_I3C_CONTROLLER_MTYPE_DIRECT
  - LL_I3C_CONTROLLER_MTYPE_LEGACY_I2C
- **EndMode:** This parameter can be one of the following values:
  - LL_I3C_GENERATE_STOP
  - LL_I3C_GENERATE_RESTART

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR ADD LL_I3C_ControllerHandleMessage
- CR DCNT LL_I3C_ControllerHandleMessage
- CR RNW LL_I3C_ControllerHandleMessage
- CR MTYPE LL_I3C_ControllerHandleMessage
- CR MEND LL_I3C_ControllerHandleMessage

LL_I3C_ControllerHandleCCC

Function name

__STATIC_INLINE void LL_I3C_ControllerHandleCCC (I3C_TypeDef * I3Cx, uint32_t CCCValue, uint32_t AddByteSize, uint32_t EndMode)

Function description

Handles I3C Common Command Code content on the I3C Bus as Controller.

Parameters

- **I3Cx:** I3C Instance.
- **CCCValue:** Specifies the Command Code to be programmed. This parameter must be a value between Min_Data=0 and Max_Data=0x1FF.
- **AddByteSize:** Specifies the number of CCC additional bytes to be programmed. This parameter must be a value between Min_Data=0 and Max_Data=65535.
- **EndMode:** This parameter can be one of the following values:
  - LL_I3C_GENERATE_STOP
  - LL_I3C_GENERATE_RESTART

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CCC LL_I3C_ControllerHandleCCC
- CR DCNT LL_I3C_ControllerHandleCCC
- CR MTYPE LL_I3C_ControllerHandleCCC
- CR MEND LL_I3C_ControllerHandleCCC

LL_I3C_TargetHandleMessage

Function name

__STATIC_INLINE void LL_I3C_TargetHandleMessage (I3C_TypeDef * I3Cx, uint32_t MessageType, uint32_t IBISize)

Function description

Handles I3C Message content on the I3C Bus as Target.

Parameters

- **I3Cx:** I3C Instance.
- **MessageType:** This parameter can be one of the following values:
  - LL_I3C_TARGET_MTYPE_HOT_JOIN
  - LL_I3C_TARGET_MTYPE_CONTROLLER_ROLE_REQ
  - LL_I3C_TARGET_MTYPE_IBI
- **IBISize:** Specifies the number of IBI bytes. This parameter must be a value between Min_Data=0 and Max_Data=65535.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR MTYPE LL_I3C_TargetHandleMessage
- CR DCNT LL_I3C_TargetHandleMessage

LL_I3C_ReceiveData8

Function name

__STATIC_INLINE uint8_t LL_I3C_ReceiveData8 (const I3C_TypeDef * I3Cx)

Function description

Read Receive Data Byte register.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Value:** between Min_Data=0 to Max_Data=0xFF

Reference Manual to LL API cross reference:

- RDR RDB0 LL_I3C_ReceiveData8

LL_I3C_TransmitData8

Function name

__STATIC_INLINE void LL_I3C_TransmitData8 (I3C_TypeDef * I3Cx, uint8_t Data)

Function description

Write in Transmit Data Byte Register.

Parameters

- **I3Cx:** I3C Instance.
- **Data:** This parameter must be a value between Min_Data=0 and Max_Data=0xFF.

Return values

- **None:**

Reference Manual to LL API cross reference:

- TDR TDB0 LL_I3C_TransmitData8

LL_I3C_ReceiveData32

Function name

__STATIC_INLINE uint32_t LL_I3C_ReceiveData32 (const I3C_TypeDef * I3Cx)

Function description

Read Receive Data Word register.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Value:** between Min_Data=0 to Max_Data=0xFFFFFFFF

Notes

- Content of register is filled in Little Endian. Mean MSB correspond to last data byte received, LSB correspond to first data byte received.

Reference Manual to LL API cross reference:

- RDWR RDWR LL_I3C_ReceiveData32

LL_I3C_TransmitData32

Function name

__STATIC_INLINE void LL_I3C_TransmitData32 (I3C_TypeDef * I3Cx, uint32_t Data)

Function description

Write in Transmit Data Word Register.

Parameters

- **I3Cx:** I3C Instance.
- **Data:** This parameter must be a value between Min_Data=0 and Max_Data=0xFFFFFFFF.

Return values

- **None:**

Notes

- Content of register is filled in Little Endian. Mean MSB correspond to last data byte transmitted, LSB correspond to first data byte transmitted.

Reference Manual to LL API cross reference:

- TDWR TDWR LL_I3C_TransmitData32

LL_I3C_SetIBIPayload

Function name

__STATIC_INLINE void LL_I3C_SetIBIPayload (I3C_TypeDef * I3Cx, uint32_t OwnIBIPayload)

Function description

Configure the IBI data payload to be sent during IBI (target mode).

Parameters

- **I3Cx:** I3C Instance.
- **OwnIBIPayload:** This parameter must be a value between Min_Data=0 and Max_Data=0xFFFFFFFF

Return values

- **None:**

Notes

- Content of register is filled in Little Endian. Mean MSB correspond to last IBI data byte, LSB correspond to first IBI data byte.

Reference Manual to LL API cross reference:

- IBIDR IBIDR LL_I3C_SetIBIPayload

LL_I3C_GetIBIPayload

Function name

__STATIC_INLINE uint32_t LL_I3C_GetIBIPayload (const I3C_TypeDef * I3Cx)

Function description

Get the own IBI data payload (target mode), or get the Target IBI received (controller mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Value:** between Min_Data=0 to Max_Data=0xFFFFFFFF

Notes

- Content of register is filled in Little Endian. Mean MSB correspond to last IBI data byte, LSB correspond to first IBI data byte.

Reference Manual to LL API cross reference:

- IBIDR IBIDR LL_I3C_GetIBIPayload

LL_I3C_GetNbIBIAddData

Function name

__STATIC_INLINE uint32_t LL_I3C_GetNbIBIAddData (const I3C_TypeDef * I3Cx)

Function description

Get the number of data bytes received when reading IBI data (controller mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Value:** between Min_Data=0 to Max_Data=0x7

Reference Manual to LL API cross reference:

- RMR IBIRDCNT LL_I3C_GetNbIBIAddData

LL_I3C_GetIBITargetAddr

Function name

__STATIC_INLINE uint32_t LL_I3C_GetIBITargetAddr (const I3C_TypeDef * I3Cx)

Function description

Get the target address received during accepted IBI or Controller-role request.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Value:** between Min_Data=0 to Max_Data=0x3F

Reference Manual to LL API cross reference:

- RMR RADD LL_I3C_GetIBITargetAddr

LL_I3C_ConfigTxPreload

Function name

__STATIC_INLINE void LL_I3C_ConfigTxPreload (I3C_TypeDef * I3Cx, uint16_t TxDataCount)

Function description

Set TX FIFO Preload (target mode).

Parameters

- **I3Cx:** I3C Instance.
- **TxDataCount:** This parameter must be a value between Min_Data=0 and Max_Data=0xFFFF

Return values

- **None:**

Notes

- Set high by Software, cleared by hardware when all the bytes to transmit have been loaded to TX FIFO.

Reference Manual to LL API cross reference:

- TGTTDR PRELOAD LL_I3C_ConfigTxPreload
- TGTTDR TDCNT LL_I3C_ConfigTxPreload

LL_I3C_IsActiveTxPreload

Function name

__STATIC_INLINE uint32_t LL_I3C_IsActiveTxPreload (const I3C_TypeDef * I3Cx)

Function description

Indicates the status of TX FIFO preload (target mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- Set high by Software, cleared by hardware when all the bytes to transmit have been loaded to TX FIFO.

Reference Manual to LL API cross reference:

- TGTTDR PRELOAD LL_I3C_IsActiveTxPreload

LL_I3C_GetTxPreloadDataCount

Function name

__STATIC_INLINE uint16_t LL_I3C_GetTxPreloadDataCount (const I3C_TypeDef * I3Cx)

Function description

Get the number of bytes to transmit (target mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Value:** between Min_Data=0 to Max_Data=0xFFFF

Notes

- The return value correspond to the remaining number of bytes to load in TX FIFO.

Reference Manual to LL API cross reference:

- TGTTDR TDCNT LL_I3C_GetTxPreloadDataCount

LL_I3C_GetXferDataCount

Function name

__STATIC_INLINE uint32_t LL_I3C_GetXferDataCount (const I3C_TypeDef * I3Cx)

Function description

Get the number of data during a Transfer.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Value:** between Min_Data=0 to Max_Data=0xFFFF

Notes

- The return value correspond to number of transmitted bytes reported during Address Assignment process in Target mode. The return value correspond to number of target detected during Address Assignment process in Controller mode. The return value correspond to number of data bytes read from or sent to the I3C bus during the message link to MID current value.

Reference Manual to LL API cross reference:

- SR XDCNT LL_I3C_GetXferDataCount

LL_I3C_IsTargetAbortPrivateRead

Function name

__STATIC_INLINE uint32_t LL_I3C_IsTargetAbortPrivateRead (const I3C_TypeDef * I3Cx)

Function description

Indicates if a Target abort a private read command.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR ABT LL_I3C_IsTargetAbortPrivateRead

LL_I3C_GetMessageDirection

Function name

__STATIC_INLINE uint32_t LL_I3C_GetMessageDirection (const I3C_TypeDef * I3Cx)

Function description

Get Direction of the Message.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Returned:** value can be one of the following values:
  - LL_I3C_MESSAGE_DIRECTION_WRITE
  - LL_I3C_MESSAGE_DIRECTION_READ

Reference Manual to LL API cross reference:

- SR DIR LL_I3C_GetMessageDirection

LL_I3C_GetMessageIdentifier

Function name

__STATIC_INLINE uint32_t LL_I3C_GetMessageIdentifier (const I3C_TypeDef * I3Cx)

Function description

Get Message identifier.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Value:** between Min_Data=0 to Max_Data=0xFF, representing the internal hardware counter value.

Reference Manual to LL API cross reference:

- SR MID LL_I3C_GetMessageIdentifier

LL_I3C_GetMessageErrorCode

Function name

__STATIC_INLINE uint32_t LL_I3C_GetMessageErrorCode (const I3C_TypeDef * I3Cx)

Function description

Get Message error code.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Returned:** value can be one of the following values:
  - LL_I3C_CONTROLLER_ERROR_CE0
  - LL_I3C_CONTROLLER_ERROR_CE1
  - LL_I3C_CONTROLLER_ERROR_CE2
  - LL_I3C_CONTROLLER_ERROR_CE3
  - LL_I3C_TARGET_ERROR_TE0
  - LL_I3C_TARGET_ERROR_TE1
  - LL_I3C_TARGET_ERROR_TE2
  - LL_I3C_TARGET_ERROR_TE3
  - LL_I3C_TARGET_ERROR_TE4
  - LL_I3C_TARGET_ERROR_TE5
  - LL_I3C_TARGET_ERROR_TE6

Reference Manual to LL API cross reference:

- SER CODERR LL_I3C_GetMessageErrorCode

LL_I3C_GetReceiveCommandCode

Function name

__STATIC_INLINE uint32_t LL_I3C_GetReceiveCommandCode (const I3C_TypeDef * I3Cx)

Function description

Get CCC code of received command.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Value:** between Min_Data=0 to Max_Data=0xFF.

Reference Manual to LL API cross reference:

- RMR RCODE LL_I3C_GetReceiveCommandCode

LL_I3C_SetTargetDynamicAddress

Function name

__STATIC_INLINE void LL_I3C_SetTargetDynamicAddress (I3C_TypeDef * I3Cx, uint32_t TargetId, uint32_t DynamicAddr)

Function description

Set Dynamic Address assigned to target x.

Parameters

- **I3Cx:** I3C Instance.
- **TargetId:** This parameter must be a value between Min_Data=1 and Max_Data=4
- **DynamicAddr:** Value between Min_Data=0 to Max_Data=0x7F

Return values

- **None:**

Reference Manual to LL API cross reference:

- DEVRX DA LL_I3C_SetTargetDynamicAddress

LL_I3C_GetTargetDynamicAddress

Function name

__STATIC_INLINE uint32_t LL_I3C_GetTargetDynamicAddress (const I3C_TypeDef * I3Cx, uint32_t TargetId)

Function description

Get Dynamic Address assigned to target x.

Parameters

- **I3Cx:** I3C Instance.
- **TargetId:** This parameter must be a value between Min_Data=1 and Max_Data=4

Return values

- **Value:** between Min_Data=0 to Max_Data=0x7F

Reference Manual to LL API cross reference:

- DEVRX DA LL_I3C_GetTargetDynamicAddress

LL_I3C_EnableTargetIBIAck

Function name

__STATIC_INLINE void LL_I3C_EnableTargetIBIAck (I3C_TypeDef * I3Cx, uint32_t TargetId)

Function description

Enable IBI Acknowledgement from target x(controller mode).

Parameters

- **I3Cx:** I3C Instance.
- **TargetId:** This parameter must be a value between Min_Data=1 and Max_Data=4

Return values

- **None:**

Notes

- The bit DIS is automatically set when CRACK or IBIACK are set. This mean DEVRX register access is not allowed. Reset CRACK and IBIACK will reset DIS bit.

Reference Manual to LL API cross reference:

- DEVRX IBIACK LL_I3C_EnableTargetIBIAck

LL_I3C_DisableTargetIBIAck

Function name

__STATIC_INLINE void LL_I3C_DisableTargetIBIAck (I3C_TypeDef * I3Cx, uint32_t TargetId)

Function description

Disable IBI Acknowledgement from target x (controller mode).

Parameters

- **I3Cx:** I3C Instance.
- **TargetId:** This parameter must be a value between Min_Data=1 and Max_Data=4

Return values

- **None:**

Reference Manual to LL API cross reference:

- DEVRX IBIACK LL_I3C_DisableTargetIBIAck

LL_I3C_IsEnabledTargetIBIAck

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledTargetIBIAck (const I3C_TypeDef * I3Cx, uint32_t TargetId)

Function description

Indicates if IBI from target x will be Acknowledged or Not Acknowledged (controller mode).

Parameters

- **I3Cx:** I3C Instance.
- **TargetId:** This parameter must be a value between Min_Data=1 and Max_Data=4

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DEVRX IBIACK LL_I3C_IsEnabledTargetIBIAck

LL_I3C_EnableTargetCRAck

Function name

__STATIC_INLINE void LL_I3C_EnableTargetCRAck (I3C_TypeDef * I3Cx, uint32_t TargetId)

Function description

Enable Controller-role Request Acknowledgement from target x(controller mode).

Parameters

- **I3Cx:** I3C Instance.
- **TargetId:** This parameter must be a value between Min_Data=1 and Max_Data=4

Return values

- **None:**

Notes

- The bit DIS is automatically set when CRACK or IBIACK are set. This mean DEVRX register access is not allowed. Reset CRACK and IBIACK will reset DIS bit.

Reference Manual to LL API cross reference:

- DEVRX CRACK LL_I3C_EnableTargetCRAck

LL_I3C_DisableTargetCRAck

Function name

__STATIC_INLINE void LL_I3C_DisableTargetCRAck (I3C_TypeDef * I3Cx, uint32_t TargetId)

Function description

Disable Controller-role Request Acknowledgement from target x (controller mode).

Parameters

- **I3Cx:** I3C Instance.
- **TargetId:** This parameter must be a value between Min_Data=1 and Max_Data=4

Return values

- **None:**

Reference Manual to LL API cross reference:

- DEVRX CRACK LL_I3C_DisableTargetCRAck

LL_I3C_IsEnabledTargetCRAck

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledTargetCRAck (const I3C_TypeDef * I3Cx, uint32_t TargetId)

Function description

Indicates if Controller-role Request from target x will be Acknowledged or Not Acknowledged (controller mode).

Parameters

- **I3Cx:** I3C Instance.
- **TargetId:** This parameter must be a value between Min_Data=1 and Max_Data=4

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DEVRX CRACK LL_I3C_IsEnabledTargetCRAck

LL_I3C_EnableIBIAddData

Function name

__STATIC_INLINE void LL_I3C_EnableIBIAddData (I3C_TypeDef * I3Cx, uint32_t TargetId)

Function description

Enable additional Mandatory Data Byte (MDB) follows the accepted IBI from target x.

Parameters

- **I3Cx:** I3C Instance.
- **TargetId:** This parameter must be a value between Min_Data=1 and Max_Data=4

Return values

- **None:**

Reference Manual to LL API cross reference:

- DEVRX IBIDEN LL_I3C_EnableIBIAddData

LL_I3C_DisableIBIAddData

Function name

__STATIC_INLINE void LL_I3C_DisableIBIAddData (I3C_TypeDef * I3Cx, uint32_t TargetId)

Function description

Disable additional Mandatory Data Byte (MDB) follows the accepted IBI from target x.

Parameters

- **I3Cx:** I3C Instance.
- **TargetId:** This parameter must be a value between Min_Data=1 and Max_Data=4

Return values

- **None:**

Reference Manual to LL API cross reference:

- DEVRX IBIDEN LL_I3C_DisableIBIAddData

LL_I3C_IsEnabledIBIAddData

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledIBIAddData (const I3C_TypeDef * I3Cx, uint32_t TargetId)

Function description

Indicates if additional Mandatory Data Byte (MDB) follows the accepted IBI from target x.

Parameters

- **I3Cx:** I3C Instance.
- **TargetId:** This parameter must be a value between Min_Data=1 and Max_Data=4

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DEVRX IBIDEN LL_I3C_IsEnabledIBIAddData

LL_I3C_EnableFrameSuspend

Function name

__STATIC_INLINE void LL_I3C_EnableFrameSuspend (I3C_TypeDef * I3Cx, uint32_t TargetId)

Function description

Enable Suspension of Current transfer during IBI treatment.

Parameters

- **I3Cx:** I3C Instance.
- **TargetId:** This parameter must be a value between Min_Data=1 and Max_Data=4

Return values

- **None:**

Notes

- When set, this feature will allow controller to send a Stop condition and CR FIFO is flushed after IBI treatment. Software has to rewrite instructions in Control Register to start a new transfer.

Reference Manual to LL API cross reference:

- DEVRX SUSP LL_I3C_EnableFrameSuspend

LL_I3C_DisableFrameSuspend

Function name

__STATIC_INLINE void LL_I3C_DisableFrameSuspend (I3C_TypeDef * I3Cx, uint32_t TargetId)

Function description

Disable Suspension of Current transfer during IBI treatment.

Parameters

- **I3Cx:** I3C Instance.
- **TargetId:** This parameter must be a value between Min_Data=1 and Max_Data=4

Return values

- **None:**

Notes

- When set, this feature will allow controller to continue CR FIFO treatment after IBI treatment.

Reference Manual to LL API cross reference:

- DEVRX SUSP LL_I3C_DisableFrameSuspend

LL_I3C_IsFrameMustBeSuspended

Function name

__STATIC_INLINE uint32_t LL_I3C_IsFrameMustBeSuspended (const I3C_TypeDef * I3Cx, uint32_t TargetId)

Function description

Indicates if I3C transfer must be Suspended or not Suspended during IBI treatment from target x.

Parameters

- **I3Cx:** I3C Instance.
- **TargetId:** This parameter must be a value between Min_Data=1 and Max_Data=4

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DEVRX SUSP LL_I3C_IsFrameMustBeSuspended

LL_I3C_IsAllowedPayloadUpdate

Function name

__STATIC_INLINE uint32_t LL_I3C_IsAllowedPayloadUpdate (const I3C_TypeDef * I3Cx, uint32_t TargetId)

Function description

Indicates if update of the Device Characteristics Register is Allowed or Not Allowed.

Parameters

- **I3Cx:** I3C Instance.
- **TargetId:** This parameter must be a value between Min_Data=1 and Max_Data=4

Return values

- **State:** of bit (1 or 0).

Notes

- Used to prevent software writing during reception of an IBI or Controller-role Request from target x.

Reference Manual to LL API cross reference:

- DEVRX DIS LL_I3C_IsAllowedPayloadUpdate

LL_I3C_ConfigDeviceCapabilities

Function name

__STATIC_INLINE void LL_I3C_ConfigDeviceCapabilities (I3C_TypeDef * I3Cx, uint32_t TargetId, uint32_t DynamicAddr, uint32_t IBIAck, uint32_t IBIAddData, uint32_t CRAck)

Function description

Set I3C bus devices configuration.

Parameters

- **I3Cx:** I3C Instance.
- **TargetId:** This parameter must be a value between Min_Data=1 and Max_Data=4
- **DynamicAddr:** Value between Min_Data=0 to Max_Data=0x7F
- **IBIAck:** Value This parameter can be one of the following values:
  - LL_I3C_IBI_CAPABILITY
  - LL_I3C_IBI_NO_CAPABILITY
- **IBIAddData:** This parameter can be one of the following values:
  - LL_I3C_IBI_DATA_ENABLE
  - LL_I3C_IBI_DATA_DISABLE
- **CRAck:** This parameter can be one of the following values:
  - LL_I3C_CR_CAPABILITY
  - LL_I3C_CR_NO_CAPABILITY

Return values

- **None:**

Notes

- This function is called only when the I3C instance is initialized as controller. This function can be called by the controller application to help the automatic treatment when target have capability of IBI and/or Control-Role.

Reference Manual to LL API cross reference:

- DEVRX DA LL_I3C_ConfigDeviceCapabilities
- DEVRX IBIACK LL_I3C_ConfigDeviceCapabilities
- DEVRX IBIDEN LL_I3C_ConfigDeviceCapabilities
- DEVRX CRACK LL_I3C_ConfigDeviceCapabilities

LL_I3C_IsActiveFlag_CFE

Function name

__STATIC_INLINE uint32_t LL_I3C_IsActiveFlag_CFE (const I3C_TypeDef * I3Cx)

Function description

Indicates the status of Control FIFO Empty flag.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- EVR CFEF LL_I3C_IsActiveFlag_CFE

LL_I3C_IsActiveFlag_TXFE

Function name

__STATIC_INLINE uint32_t LL_I3C_IsActiveFlag_TXFE (const I3C_TypeDef * I3Cx)

Function description

Indicates the status of Transmit FIFO Empty flag.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- EVR TXFEF LL_I3C_IsActiveFlag_TXFE

LL_I3C_IsActiveFlag_CFNF

Function name

__STATIC_INLINE uint32_t LL_I3C_IsActiveFlag_CFNF (const I3C_TypeDef * I3Cx)

Function description

Indicates the status of Control FIFO Not Full flag.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- When a transfer is ongoing, the Control FIFO shall not be written unless this flag is set.

Reference Manual to LL API cross reference:

- EVR CFNFF LL_I3C_IsActiveFlag_CFNF

LL_I3C_IsActiveFlag_SFNE

Function name

__STATIC_INLINE uint32_t LL_I3C_IsActiveFlag_SFNE (const I3C_TypeDef * I3Cx)

Function description

Indicates the status of Status FIFO Not Empty flag.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- This flag is updated only when the FIFO is used, mean SMODE = 1.

Reference Manual to LL API cross reference:

- EVR SFNEF LL_I3C_IsActiveFlag_SFNE

LL_I3C_IsActiveFlag_TXFNF

Function name

__STATIC_INLINE uint32_t LL_I3C_IsActiveFlag_TXFNF (const I3C_TypeDef * I3Cx)

Function description

Indicates the status of Transmit FIFO Not Full flag.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- When a transfer is ongoing, the Transmit FIFO shall not be written unless this flag is set.

Reference Manual to LL API cross reference:

- EVR TXFNFF LL_I3C_IsActiveFlag_TXFNF

LL_I3C_IsActiveFlag_RXFNE

Function name

__STATIC_INLINE uint32_t LL_I3C_IsActiveFlag_RXFNE (const I3C_TypeDef * I3Cx)

Function description

Indicates the status of Receive FIFO Not Full flag.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- EVR RXFNEF LL_I3C_IsActiveFlag_RXFNE

LL_I3C_IsActiveFlag_RXLAST

Function name

__STATIC_INLINE uint32_t LL_I3C_IsActiveFlag_RXLAST (const I3C_TypeDef * I3Cx)

Function description

Indicates that the last Receive byte is available.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- EVR RXLASTF LL_I3C_IsActiveFlag_RXLAST

LL_I3C_IsActiveFlag_TXLAST

Function name

__STATIC_INLINE uint32_t LL_I3C_IsActiveFlag_TXLAST (const I3C_TypeDef * I3Cx)

Function description

Indicates that the last Transmit byte is written in FIFO.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- EVR TXLASTF LL_I3C_IsActiveFlag_TXLAST

LL_I3C_IsActiveFlag_FC

Function name

__STATIC_INLINE uint32_t LL_I3C_IsActiveFlag_FC (const I3C_TypeDef * I3Cx)

Function description

Indicates the status of Frame Complete flag (controller and target mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- EVR FCF LL_I3C_IsActiveFlag_FC

LL_I3C_IsActiveFlag_RXTGTEND

Function name

__STATIC_INLINE uint32_t LL_I3C_IsActiveFlag_RXTGTEND (const I3C_TypeDef * I3Cx)

Function description

Indicates the status of Reception Target End flag (controller mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- This flag is set only when status FIFO is not used, mean SMODE = 0.

Reference Manual to LL API cross reference:

- EVR RXTGTENDF LL_I3C_IsActiveFlag_RXTGTEND

LL_I3C_IsActiveFlag_ERR

Function name

__STATIC_INLINE uint32_t LL_I3C_IsActiveFlag_ERR (const I3C_TypeDef * I3Cx)

Function description

Indicates the status of Error flag (controller and target mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- EVR ERRF LL_I3C_IsActiveFlag_ERR

LL_I3C_IsActiveFlag_IBI

Function name

__STATIC_INLINE uint32_t LL_I3C_IsActiveFlag_IBI (const I3C_TypeDef * I3Cx)

Function description

Indicates the status of IBI flag (controller mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- EVR IBIF LL_I3C_IsActiveFlag_IBI

LL_I3C_IsActiveFlag_IBIEND

Function name

__STATIC_INLINE uint32_t LL_I3C_IsActiveFlag_IBIEND (const I3C_TypeDef * I3Cx)

Function description

Indicates the status of IBI End flag (target mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- EVR IBIENDF LL_I3C_IsActiveFlag_IBIEND

LL_I3C_IsActiveFlag_CR

Function name

__STATIC_INLINE uint32_t LL_I3C_IsActiveFlag_CR (const I3C_TypeDef * I3Cx)

Function description

Indicates the status of Controller-role Request flag (controller mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- EVR CRF LL_I3C_IsActiveFlag_CR

LL_I3C_IsActiveFlag_CRUPD

Function name

__STATIC_INLINE uint32_t LL_I3C_IsActiveFlag_CRUPD (const I3C_TypeDef * I3Cx)

Function description

Indicates the status of Controller-role Request Update flag (target mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- EVR BCUPDF LL_I3C_IsActiveFlag_CRUPD

LL_I3C_IsActiveFlag_HJ

Function name

__STATIC_INLINE uint32_t LL_I3C_IsActiveFlag_HJ (const I3C_TypeDef * I3Cx)

Function description

Indicates the status of Hot Join flag (controller mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- EVR HJF LL_I3C_IsActiveFlag_HJ

LL_I3C_IsActiveFlag_WKP

Function name

__STATIC_INLINE uint32_t LL_I3C_IsActiveFlag_WKP (const I3C_TypeDef * I3Cx)

Function description

Indicates the status of Wake Up flag (target mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- EVR WKPF LL_I3C_IsActiveFlag_WKP

LL_I3C_IsActiveFlag_GET

Function name

__STATIC_INLINE uint32_t LL_I3C_IsActiveFlag_GET (const I3C_TypeDef * I3Cx)

Function description

Indicates the status of Get flag (target mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- EVR GETF LL_I3C_IsActiveFlag_GET

LL_I3C_IsActiveFlag_STA

Function name

__STATIC_INLINE uint32_t LL_I3C_IsActiveFlag_STA (const I3C_TypeDef * I3Cx)

Function description

Indicates the status of Get Status flag (target mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- EVR STAF LL_I3C_IsActiveFlag_STA

LL_I3C_IsActiveFlag_DAUPD

Function name

__STATIC_INLINE uint32_t LL_I3C_IsActiveFlag_DAUPD (const I3C_TypeDef * I3Cx)

Function description

Indicates the status of Dynamic Address Update flag (target mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- EVR DAUPDF LL_I3C_IsActiveFlag_DAUPD

LL_I3C_IsActiveFlag_MWLUPD

Function name

__STATIC_INLINE uint32_t LL_I3C_IsActiveFlag_MWLUPD (const I3C_TypeDef * I3Cx)

Function description

Indicates the status of Max Write Length flag (target mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- EVR MWLUPDF LL_I3C_IsActiveFlag_MWLUPD

LL_I3C_IsActiveFlag_MRLUPD

Function name

__STATIC_INLINE uint32_t LL_I3C_IsActiveFlag_MRLUPD (const I3C_TypeDef * I3Cx)

Function description

Indicates the status of Max Read Length flag (target mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- EVR MRLUPDF LL_I3C_IsActiveFlag_MRLUPD

LL_I3C_IsActiveFlag_RST

Function name

__STATIC_INLINE uint32_t LL_I3C_IsActiveFlag_RST (const I3C_TypeDef * I3Cx)

Function description

Indicates the status of Reset flag (target mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- EVR RSTF LL_I3C_IsActiveFlag_RST

LL_I3C_IsActiveFlag_ASUPD

Function name

__STATIC_INLINE uint32_t LL_I3C_IsActiveFlag_ASUPD (const I3C_TypeDef * I3Cx)

Function description

Indicates the status of Active State flag (target mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- EVR ASUPDF LL_I3C_IsActiveFlag_ASUPD

LL_I3C_IsActiveFlag_INTUPD

Function name

__STATIC_INLINE uint32_t LL_I3C_IsActiveFlag_INTUPD (const I3C_TypeDef * I3Cx)

Function description

Indicates the status of Interrupt Update flag (target mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- EVR INTUPDF LL_I3C_IsActiveFlag_INTUPD

LL_I3C_IsActiveFlag_DEF

Function name

__STATIC_INLINE uint32_t LL_I3C_IsActiveFlag_DEF (const I3C_TypeDef * I3Cx)

Function description

Indicates the status of Define List Targets flag (target mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- EVR DEFF LL_I3C_IsActiveFlag_DEF

LL_I3C_IsActiveFlag_GRP

Function name

__STATIC_INLINE uint32_t LL_I3C_IsActiveFlag_GRP (const I3C_TypeDef * I3Cx)

Function description

Indicates the status of Define List Group Addresses flag.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- EVR GRPF LL_I3C_IsActiveFlag_GRP

LL_I3C_IsActiveFlag_PERR

Function name

__STATIC_INLINE uint32_t LL_I3C_IsActiveFlag_PERR (const I3C_TypeDef * I3Cx)

Function description

Indicates the status of Protocol Error flag.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SER PERR LL_I3C_IsActiveFlag_PERR

LL_I3C_IsActiveFlag_STALL

Function name

__STATIC_INLINE uint32_t LL_I3C_IsActiveFlag_STALL (const I3C_TypeDef * I3Cx)

Function description

Indicates the status of SCL Stall Error flag (target mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SER STALL LL_I3C_IsActiveFlag_STALL

LL_I3C_IsActiveFlag_DOVR

Function name

__STATIC_INLINE uint32_t LL_I3C_IsActiveFlag_DOVR (const I3C_TypeDef * I3Cx)

Function description

Indicates the status of RX or TX FIFO Overrun flag.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SER DOVR LL_I3C_IsActiveFlag_DOVR

LL_I3C_IsActiveFlag_COVR

Function name

__STATIC_INLINE uint32_t LL_I3C_IsActiveFlag_COVR (const I3C_TypeDef * I3Cx)

Function description

Indicates the status of Control or Status FIFO Overrun flag (controller mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SER COVR LL_I3C_IsActiveFlag_COVR

LL_I3C_IsActiveFlag_ANACK

Function name

__STATIC_INLINE uint32_t LL_I3C_IsActiveFlag_ANACK (const I3C_TypeDef * I3Cx)

Function description

Indicates the status of Address not acknowledged flag (controller mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SER ANACK LL_I3C_IsActiveFlag_ANACK

LL_I3C_IsActiveFlag_DNACK

Function name

__STATIC_INLINE uint32_t LL_I3C_IsActiveFlag_DNACK (const I3C_TypeDef * I3Cx)

Function description

Indicates the status of Data not acknowledged flag (controller mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SER DNACK LL_I3C_IsActiveFlag_DNACK

LL_I3C_IsActiveFlag_DERR

Function name

__STATIC_INLINE uint32_t LL_I3C_IsActiveFlag_DERR (const I3C_TypeDef * I3Cx)

Function description

Indicates the status of Data error flag (controller mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SER DERR LL_I3C_IsActiveFlag_DERR

LL_I3C_EnableIT_CFNF

Function name

__STATIC_INLINE void LL_I3C_EnableIT_CFNF (I3C_TypeDef * I3Cx)

Function description

Enable Control FIFO Not Full interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER CFNFIE LL_I3C_EnableIT_CFNF

LL_I3C_DisableIT_CFNF

Function name

__STATIC_INLINE void LL_I3C_DisableIT_CFNF (I3C_TypeDef * I3Cx)

Function description

Disable Control FIFO Not Full interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER CFNFIE LL_I3C_DisableIT_CFNF

LL_I3C_IsEnabledIT_CFNF

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledIT_CFNF (const I3C_TypeDef * I3Cx)

Function description

Check if Control FIFO Not Full interrupt is enabled or disabled.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER CFNFIE LL_I3C_IsEnabledIT_CFNF

LL_I3C_EnableIT_SFNE

Function name

__STATIC_INLINE void LL_I3C_EnableIT_SFNE (I3C_TypeDef * I3Cx)

Function description

Enable Status FIFO Not Empty interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER SFNEIE LL_I3C_EnableIT_SFNE

LL_I3C_DisableIT_SFNE

Function name

__STATIC_INLINE void LL_I3C_DisableIT_SFNE (I3C_TypeDef * I3Cx)

Function description

Disable Status FIFO Not Empty interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER SFNEIE LL_I3C_DisableIT_SFNE

LL_I3C_IsEnabledIT_SFNE

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledIT_SFNE (const I3C_TypeDef * I3Cx)

Function description

Check if Status FIFO Not Empty interrupt is enabled or disabled.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER SFNEIE LL_I3C_IsEnabledIT_SFNE

LL_I3C_EnableIT_TXFNF

Function name

__STATIC_INLINE void LL_I3C_EnableIT_TXFNF (I3C_TypeDef * I3Cx)

Function description

Enable Transmit FIFO Not Full interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER TXFNFIE LL_I3C_EnableIT_TXFNF

LL_I3C_DisableIT_TXFNF

Function name

__STATIC_INLINE void LL_I3C_DisableIT_TXFNF (I3C_TypeDef * I3Cx)

Function description

Disable Transmit FIFO Not Full interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER TXFNFIE LL_I3C_DisableIT_TXFNF

LL_I3C_IsEnabledIT_TXFNF

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledIT_TXFNF (const I3C_TypeDef * I3Cx)

Function description

Check if Transmit FIFO Not Full interrupt is enabled or disabled.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER TXFNFIE LL_I3C_IsEnabledIT_TXFNF

LL_I3C_EnableIT_RXFNE

Function name

__STATIC_INLINE void LL_I3C_EnableIT_RXFNE (I3C_TypeDef * I3Cx)

Function description

Enable Receive FIFO Not Empty interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER RXFNEIE LL_I3C_EnableIT_RXFNE

LL_I3C_DisableIT_RXFNE

Function name

__STATIC_INLINE void LL_I3C_DisableIT_RXFNE (I3C_TypeDef * I3Cx)

Function description

Disable Receive FIFO Not Empty interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER RXFNEIE LL_I3C_DisableIT_RXFNE

LL_I3C_IsEnabledIT_RXFNE

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledIT_RXFNE (const I3C_TypeDef * I3Cx)

Function description

Check if Receive FIFO Not Empty interrupt is enabled or disabled.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER RXFNEIE LL_I3C_IsEnabledIT_RXFNE

LL_I3C_EnableIT_FC

Function name

__STATIC_INLINE void LL_I3C_EnableIT_FC (I3C_TypeDef * I3Cx)

Function description

Enable Frame Complete interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER FCIE LL_I3C_EnableIT_FC

LL_I3C_DisableIT_FC

Function name

__STATIC_INLINE void LL_I3C_DisableIT_FC (I3C_TypeDef * I3Cx)

Function description

Disable Frame Complete interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER FCIE LL_I3C_DisableIT_FC

LL_I3C_IsEnabledIT_FC

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledIT_FC (const I3C_TypeDef * I3Cx)

Function description

Check if Frame Complete interrupt is enabled or disabled.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER FCIE LL_I3C_IsEnabledIT_FC

LL_I3C_EnableIT_RXTGTEND

Function name

__STATIC_INLINE void LL_I3C_EnableIT_RXTGTEND (I3C_TypeDef * I3Cx)

Function description

Enable Reception Target End interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER RXTGTENDIE LL_I3C_EnableIT_RXTGTEND

LL_I3C_DisableIT_RXTGTEND

Function name

__STATIC_INLINE void LL_I3C_DisableIT_RXTGTEND (I3C_TypeDef * I3Cx)

Function description

Disable Reception Target End interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER RXTGTENDIE LL_I3C_DisableIT_RXTGTEND

LL_I3C_IsEnabledIT_RXTGTEND

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledIT_RXTGTEND (const I3C_TypeDef * I3Cx)

Function description

Check if Reception Target End interrupt is enabled or disabled.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER RXTGTENDIE LL_I3C_IsEnabledIT_RXTGTEND

LL_I3C_EnableIT_ERR

Function name

__STATIC_INLINE void LL_I3C_EnableIT_ERR (I3C_TypeDef * I3Cx)

Function description

Enable Error interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER ERRIE LL_I3C_EnableIT_ERR

LL_I3C_DisableIT_ERR

Function name

__STATIC_INLINE void LL_I3C_DisableIT_ERR (I3C_TypeDef * I3Cx)

Function description

Disable Error interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER ERRIE LL_I3C_DisableIT_ERR

LL_I3C_IsEnabledIT_ERR

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledIT_ERR (const I3C_TypeDef * I3Cx)

Function description

Check if Error interrupt is enabled or disabled.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER ERRIE LL_I3C_IsEnabledIT_ERR

LL_I3C_EnableIT_IBI

Function name

__STATIC_INLINE void LL_I3C_EnableIT_IBI (I3C_TypeDef * I3Cx)

Function description

Enable IBI interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER IBIIE LL_I3C_EnableIT_IBI

LL_I3C_DisableIT_IBI

Function name

__STATIC_INLINE void LL_I3C_DisableIT_IBI (I3C_TypeDef * I3Cx)

Function description

Disable IBI interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER IBIIE LL_I3C_DisableIT_IBI

LL_I3C_IsEnabledIT_IBI

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledIT_IBI (const I3C_TypeDef * I3Cx)

Function description

Check if IBI interrupt is enabled or disabled.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER IBIIE LL_I3C_IsEnabledIT_IBI

LL_I3C_EnableIT_IBIEND

Function name

__STATIC_INLINE void LL_I3C_EnableIT_IBIEND (I3C_TypeDef * I3Cx)

Function description

Enable IBI End interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER IBIENDIE LL_I3C_EnableIT_IBIEND

LL_I3C_DisableIT_IBIEND

Function name

__STATIC_INLINE void LL_I3C_DisableIT_IBIEND (I3C_TypeDef * I3Cx)

Function description

Disable IBI End interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER IBIENDIE LL_I3C_DisableIT_IBIEND

LL_I3C_IsEnabledIT_IBIEND

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledIT_IBIEND (const I3C_TypeDef * I3Cx)

Function description

Check if IBI End interrupt is enabled or disabled.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER IBIENDIE LL_I3C_IsEnabledIT_IBIEND

LL_I3C_EnableIT_CR

Function name

__STATIC_INLINE void LL_I3C_EnableIT_CR (I3C_TypeDef * I3Cx)

Function description

Enable Controller-role interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER CRIE LL_I3C_EnableIT_CR

LL_I3C_DisableIT_CR

Function name

__STATIC_INLINE void LL_I3C_DisableIT_CR (I3C_TypeDef * I3Cx)

Function description

Disable Controller-role interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER CRIE LL_I3C_DisableIT_CR

LL_I3C_IsEnabledIT_CR

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledIT_CR (const I3C_TypeDef * I3Cx)

Function description

Check if Controller-role interrupt is enabled or disabled.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER CRIE LL_I3C_IsEnabledIT_CR

LL_I3C_EnableIT_CRUPD

Function name

__STATIC_INLINE void LL_I3C_EnableIT_CRUPD (I3C_TypeDef * I3Cx)

Function description

Enable Controller-role Update interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER CRUPDIE LL_I3C_EnableIT_CRUPD

LL_I3C_DisableIT_CRUPD

Function name

__STATIC_INLINE void LL_I3C_DisableIT_CRUPD (I3C_TypeDef * I3Cx)

Function description

Disable Controller-role Update interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER CRUPDIE LL_I3C_DisableIT_CRUPD

LL_I3C_IsEnabledIT_CRUPD

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledIT_CRUPD (const I3C_TypeDef * I3Cx)

Function description

Check if Controller-role Update interrupt is enabled or disabled.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER CRUPDIE LL_I3C_IsEnabledIT_CRUPD

LL_I3C_EnableIT_HJ

Function name

__STATIC_INLINE void LL_I3C_EnableIT_HJ (I3C_TypeDef * I3Cx)

Function description

Enable Hot Join interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER HJIE LL_I3C_EnableIT_HJ

LL_I3C_DisableIT_HJ

Function name

__STATIC_INLINE void LL_I3C_DisableIT_HJ (I3C_TypeDef * I3Cx)

Function description

Disable Hot Join interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER HJIE LL_I3C_DisableIT_HJ

LL_I3C_IsEnabledIT_HJ

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledIT_HJ (const I3C_TypeDef * I3Cx)

Function description

Check if Hot Join interrupt is enabled or disabled.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER HJIE LL_I3C_IsEnabledIT_HJ

LL_I3C_EnableIT_WKP

Function name

__STATIC_INLINE void LL_I3C_EnableIT_WKP (I3C_TypeDef * I3Cx)

Function description

Enable Wake Up interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER WKPIE LL_I3C_EnableIT_WKP

LL_I3C_DisableIT_WKP

Function name

__STATIC_INLINE void LL_I3C_DisableIT_WKP (I3C_TypeDef * I3Cx)

Function description

Disable Wake Up interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER WKPIE LL_I3C_DisableIT_WKP

LL_I3C_IsEnabledIT_WKP

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledIT_WKP (const I3C_TypeDef * I3Cx)

Function description

Check if Wake Up is enabled or disabled.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER WKPIE LL_I3C_IsEnabledIT_WKP

LL_I3C_EnableIT_GET

Function name

__STATIC_INLINE void LL_I3C_EnableIT_GET (I3C_TypeDef * I3Cx)

Function description

Enable Get Command interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER GETIE LL_I3C_EnableIT_GET

LL_I3C_DisableIT_GET

Function name

__STATIC_INLINE void LL_I3C_DisableIT_GET (I3C_TypeDef * I3Cx)

Function description

Disable Get Command interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER GETIE LL_I3C_DisableIT_GET

LL_I3C_IsEnabledIT_GET

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledIT_GET (const I3C_TypeDef * I3Cx)

Function description

Check if Get Command is enabled or disabled.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER GETIE LL_I3C_IsEnabledIT_GET

LL_I3C_EnableIT_STA

Function name

__STATIC_INLINE void LL_I3C_EnableIT_STA (I3C_TypeDef * I3Cx)

Function description

Enable Get Status interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER STAIE LL_I3C_EnableIT_STA

LL_I3C_DisableIT_STA

Function name

__STATIC_INLINE void LL_I3C_DisableIT_STA (I3C_TypeDef * I3Cx)

Function description

Disable Get Status interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER STAIE LL_I3C_DisableIT_STA

LL_I3C_IsEnabledIT_STA

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledIT_STA (const I3C_TypeDef * I3Cx)

Function description

Check if Get Status interrupt is enabled or disabled.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER STAIE LL_I3C_IsEnabledIT_STA

LL_I3C_EnableIT_DAUPD

Function name

__STATIC_INLINE void LL_I3C_EnableIT_DAUPD (I3C_TypeDef * I3Cx)

Function description

Enable Dynamic Address Update interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER DAUPDIE LL_I3C_EnableIT_DAUPD

LL_I3C_DisableIT_DAUPD

Function name

__STATIC_INLINE void LL_I3C_DisableIT_DAUPD (I3C_TypeDef * I3Cx)

Function description

Disable Dynamic Address Update interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER DAUPDIE LL_I3C_DisableIT_DAUPD

LL_I3C_IsEnabledIT_DAUPD

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledIT_DAUPD (const I3C_TypeDef * I3Cx)

Function description

Check if Dynamic Address Update interrupt is enabled or disabled.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER DAUPDIE LL_I3C_IsEnabledIT_DAUPD

LL_I3C_EnableIT_MWLUPD

Function name

__STATIC_INLINE void LL_I3C_EnableIT_MWLUPD (I3C_TypeDef * I3Cx)

Function description

Enable Max Write Length Update interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER MWLUPDIE LL_I3C_EnableIT_MWLUPD

LL_I3C_DisableIT_MWLUPD

Function name

__STATIC_INLINE void LL_I3C_DisableIT_MWLUPD (I3C_TypeDef * I3Cx)

Function description

Disable Max Write Length Update interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER MWLUPDIE LL_I3C_DisableIT_MWLUPD

LL_I3C_IsEnabledIT_MWLUPD

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledIT_MWLUPD (const I3C_TypeDef * I3Cx)

Function description

Check if Max Write Length Update interrupt is enabled or disabled.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER MWLUPDIE LL_I3C_IsEnabledIT_MWLUPD

LL_I3C_EnableIT_MRLUPD

Function name

__STATIC_INLINE void LL_I3C_EnableIT_MRLUPD (I3C_TypeDef * I3Cx)

Function description

Enable Max Read Length Update interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER MRLUPDIE LL_I3C_EnableIT_MRLUPD

LL_I3C_DisableIT_MRLUPD

Function name

__STATIC_INLINE void LL_I3C_DisableIT_MRLUPD (I3C_TypeDef * I3Cx)

Function description

Disable Max Read Length Update interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER MRLUPDIE LL_I3C_DisableIT_MRLUPD

LL_I3C_IsEnabledIT_MRLUPD

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledIT_MRLUPD (const I3C_TypeDef * I3Cx)

Function description

Check if Max Read Length Update interrupt is enabled or disabled.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER MRLUPDIE LL_I3C_IsEnabledIT_MRLUPD

LL_I3C_EnableIT_RST

Function name

__STATIC_INLINE void LL_I3C_EnableIT_RST (I3C_TypeDef * I3Cx)

Function description

Enable Reset interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER RSTIE LL_I3C_EnableIT_RST

LL_I3C_DisableIT_RST

Function name

__STATIC_INLINE void LL_I3C_DisableIT_RST (I3C_TypeDef * I3Cx)

Function description

Disable Reset interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER RSTIE LL_I3C_DisableIT_RST

LL_I3C_IsEnabledIT_RST

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledIT_RST (const I3C_TypeDef * I3Cx)

Function description

Check if Reset interrupt is enabled or disabled.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER RSTIE LL_I3C_IsEnabledIT_RST

LL_I3C_EnableIT_ASUPD

Function name

__STATIC_INLINE void LL_I3C_EnableIT_ASUPD (I3C_TypeDef * I3Cx)

Function description

Enable Activity State Update interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER ASUPDIE LL_I3C_EnableIT_ASUPD

LL_I3C_DisableIT_ASUPD

Function name

__STATIC_INLINE void LL_I3C_DisableIT_ASUPD (I3C_TypeDef * I3Cx)

Function description

Disable Activity State Update interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER ASUPDIE LL_I3C_DisableIT_ASUPD

LL_I3C_IsEnabledIT_ASUPD

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledIT_ASUPD (const I3C_TypeDef * I3Cx)

Function description

Check if Activity State Update interrupt is enabled or disabled.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER ASUPDIE LL_I3C_IsEnabledIT_ASUPD

LL_I3C_EnableIT_INTUPD

Function name

__STATIC_INLINE void LL_I3C_EnableIT_INTUPD (I3C_TypeDef * I3Cx)

Function description

Enable Interrupt Update interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER INTUPDIE LL_I3C_EnableIT_INTUPD

LL_I3C_DisableIT_INTUPD

Function name

__STATIC_INLINE void LL_I3C_DisableIT_INTUPD (I3C_TypeDef * I3Cx)

Function description

Disable Interrupt Update interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER INTUPDIE LL_I3C_DisableIT_INTUPD

LL_I3C_IsEnabledIT_INTUPD

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledIT_INTUPD (const I3C_TypeDef * I3Cx)

Function description

Check if Interrupt Update interrupt is enabled or disabled.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER INTUPDIE LL_I3C_IsEnabledIT_INTUPD

LL_I3C_EnableIT_DEF

Function name

__STATIC_INLINE void LL_I3C_EnableIT_DEF (I3C_TypeDef * I3Cx)

Function description

Enable Define List Target interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER DEFIE LL_I3C_EnableIT_DEF

LL_I3C_DisableIT_DEF

Function name

__STATIC_INLINE void LL_I3C_DisableIT_DEF (I3C_TypeDef * I3Cx)

Function description

Disable Define List Target interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER DEFIE LL_I3C_DisableIT_DEF

LL_I3C_IsEnabledIT_DEF

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledIT_DEF (const I3C_TypeDef * I3Cx)

Function description

Check if Define List Target interrupt is enabled or disabled.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER DEFIE LL_I3C_IsEnabledIT_DEF

LL_I3C_EnableIT_GRP

Function name

__STATIC_INLINE void LL_I3C_EnableIT_GRP (I3C_TypeDef * I3Cx)

Function description

Enable Define List Group Addresses interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER GRPIE LL_I3C_EnableIT_GRP

LL_I3C_DisableIT_GRP

Function name

__STATIC_INLINE void LL_I3C_DisableIT_GRP (I3C_TypeDef * I3Cx)

Function description

Disable Define List Group Addresses interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER GRPIE LL_I3C_DisableIT_GRP

LL_I3C_IsEnabledIT_GRP

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledIT_GRP (const I3C_TypeDef * I3Cx)

Function description

Check if Define List Group Addresses interrupt is enabled or disabled.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER GRPIE LL_I3C_IsEnabledIT_GRP

LL_I3C_ClearFlag_FC

Function name

__STATIC_INLINE void LL_I3C_ClearFlag_FC (I3C_TypeDef * I3Cx)

Function description

Clear Frame Complete flag (controller and target mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CEVR CFCF LL_I3C_ClearFlag_FC

LL_I3C_ClearFlag_RXTGTEND

Function name

__STATIC_INLINE void LL_I3C_ClearFlag_RXTGTEND (I3C_TypeDef * I3Cx)

Function description

Clear Reception Target End flag (controller mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CEVR CRXTGTENDF LL_I3C_ClearFlag_RXTGTEND

LL_I3C_ClearFlag_ERR

Function name

__STATIC_INLINE void LL_I3C_ClearFlag_ERR (I3C_TypeDef * I3Cx)

Function description

Clear Error flag (controller and target mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CEVR CERRF LL_I3C_ClearFlag_ERR

LL_I3C_ClearFlag_IBI

Function name

__STATIC_INLINE void LL_I3C_ClearFlag_IBI (I3C_TypeDef * I3Cx)

Function description

Clear IBI flag (controller mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CEVR CIBIF LL_I3C_ClearFlag_IBI

LL_I3C_ClearFlag_IBIEND

Function name

__STATIC_INLINE void LL_I3C_ClearFlag_IBIEND (I3C_TypeDef * I3Cx)

Function description

Clear IBI End flag (target mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CEVR CIBIENDF LL_I3C_ClearFlag_IBIEND

LL_I3C_ClearFlag_CR

Function name

__STATIC_INLINE void LL_I3C_ClearFlag_CR (I3C_TypeDef * I3Cx)

Function description

Clear Controller-role Request flag (controller mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CEVR CCRF LL_I3C_ClearFlag_CR

LL_I3C_ClearFlag_CRUPD

Function name

__STATIC_INLINE void LL_I3C_ClearFlag_CRUPD (I3C_TypeDef * I3Cx)

Function description

Clear Controller-role Request Update flag (target mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CEVR CCRUPDF LL_I3C_ClearFlag_CRUPD

LL_I3C_ClearFlag_HJ

Function name

__STATIC_INLINE void LL_I3C_ClearFlag_HJ (I3C_TypeDef * I3Cx)

Function description

Clear Hot Join flag (controller mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CEVR CHJF LL_I3C_ClearFlag_HJ

LL_I3C_ClearFlag_WKP

Function name

__STATIC_INLINE void LL_I3C_ClearFlag_WKP (I3C_TypeDef * I3Cx)

Function description

Clear Wake Up flag (target mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CEVR CWKPF LL_I3C_ClearFlag_WKP

LL_I3C_ClearFlag_GET

Function name

__STATIC_INLINE void LL_I3C_ClearFlag_GET (I3C_TypeDef * I3Cx)

Function description

Clear Get flag (target mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CEVR CGETF LL_I3C_ClearFlag_GET

LL_I3C_ClearFlag_STA

Function name

__STATIC_INLINE void LL_I3C_ClearFlag_STA (I3C_TypeDef * I3Cx)

Function description

Clear Get Status flag (target mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CEVR CSTAF LL_I3C_ClearFlag_STA

LL_I3C_ClearFlag_DAUPD

Function name

__STATIC_INLINE void LL_I3C_ClearFlag_DAUPD (I3C_TypeDef * I3Cx)

Function description

Clear Dynamic Address Update flag (target mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CEVR CDAUPDF LL_I3C_ClearFlag_DAUPD

LL_I3C_ClearFlag_MWLUPD

Function name

__STATIC_INLINE void LL_I3C_ClearFlag_MWLUPD (I3C_TypeDef * I3Cx)

Function description

Clear Max Write Length flag (target mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CEVR CMWLUPDF LL_I3C_ClearFlag_MWLUPD

LL_I3C_ClearFlag_MRLUPD

Function name

__STATIC_INLINE void LL_I3C_ClearFlag_MRLUPD (I3C_TypeDef * I3Cx)

Function description

Clear Max Read Length flag (target mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CEVR CMRLUPDF LL_I3C_ClearFlag_MRLUPD

LL_I3C_ClearFlag_RST

Function name

__STATIC_INLINE void LL_I3C_ClearFlag_RST (I3C_TypeDef * I3Cx)

Function description

Clear Reset flag (target mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CEVR CRSTF LL_I3C_ClearFlag_RST

LL_I3C_ClearFlag_ASUPD

Function name

__STATIC_INLINE void LL_I3C_ClearFlag_ASUPD (I3C_TypeDef * I3Cx)

Function description

Clear Active State flag (target mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CEVR CASUPDF LL_I3C_ClearFlag_ASUPD

LL_I3C_ClearFlag_INTUPD

Function name

__STATIC_INLINE void LL_I3C_ClearFlag_INTUPD (I3C_TypeDef * I3Cx)

Function description

Clear Interrupt Update flag (target mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CEVR CINTUPDF LL_I3C_ClearFlag_INTUPD

LL_I3C_ClearFlag_DEF

Function name

__STATIC_INLINE void LL_I3C_ClearFlag_DEF (I3C_TypeDef * I3Cx)

Function description

Clear Define List Targets flag (target mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CEVR CDEFF LL_I3C_ClearFlag_DEF

LL_I3C_ClearFlag_GRP

Function name

__STATIC_INLINE void LL_I3C_ClearFlag_GRP (I3C_TypeDef * I3Cx)

Function description

Clear Define List Group Addresses flag.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CEVR CGRPF LL_I3C_ClearFlag_GRP

LL_I3C_Init

Function name

ErrorStatus LL_I3C_Init (I3C_TypeDef * I3Cx, LL_I3C_InitTypeDef * I3C_InitStruct, uint32_t Mode)

Function description

Initialize the I3C registers according to the specified parameters in I3C_InitStruct.

Parameters

- **I3Cx:** I3C Instance.
- **I3C_InitStruct:** pointer to a LL_I3C_InitTypeDef structure.
- **Mode:** I3C peripheral mode. This parameter can be a value of MODE.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: I3C registers are initialized
  - ERROR: Not applicable

LL_I3C_DeInit

Function name

ErrorStatus LL_I3C_DeInit (const I3C_TypeDef * I3Cx)

Function description

De-initialize the I3C registers to their default reset values.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: I3C registers are de-initialized
  - ERROR: I3C registers are not de-initialized

LL_I3C_StructInit

Function name

void LL_I3C_StructInit (LL_I3C_InitTypeDef * I3C_InitStruct)

Function description

Set each LL_I3C_InitTypeDef field to default value.

Parameters

- **I3C_InitStruct:** Pointer to a LL_I3C_InitTypeDef structure.

Return values

- **None:**

95.3 I3C Firmware driver defines

The following section lists the various define and macros of the module.

95.3.1 I3C

I3C

BUS ACTIVITY STATE

LL_I3C_BUS_ACTIVITY_STATE_0

Controller on the Bus Activity State 0

LL_I3C_BUS_ACTIVITY_STATE_1

Controller on the Bus Activity State 1

LL_I3C_BUS_ACTIVITY_STATE_2

Controller on the Bus Activity State 2

LL_I3C_BUS_ACTIVITY_STATE_3

Controller on the Bus Activity State 3

CONTROLLER ERROR

LL_I3C_CONTROLLER_ERROR_CE0

Controller detected an illegally formatted CCC

LL_I3C_CONTROLLER_ERROR_CE1

Controller detected that transmitted data on the bus is different than expected

LL_I3C_CONTROLLER_ERROR_CE2

Controller detected that broadcast address 7'h7E has been nacked

LL_I3C_CONTROLLER_ERROR_CE3

Controller detected that new Controller did not drive the bus after Controller-role handoff

CONTROLLER MTYPE

LL_I3C_CONTROLLER_MTYPE_RELEASE

SCL output clock stops running until next instruction executed

LL_I3C_CONTROLLER_MTYPE_HEADER

Header Message

LL_I3C_CONTROLLER_MTYPE_PRIVATE

Private Message Type

LL_I3C_CONTROLLER_MTYPE_DIRECT

Direct Message Type

LL_I3C_CONTROLLER_MTYPE_LEGACY_I2C

Legacy I2C Message Type

LL_I3C_CONTROLLER_MTYPE_CCC

Common Command Code

CR CAPABILITY

LL_I3C_CR_CAPABILITY

Controller acknowledge Target Controller Role capable

LL_I3C_CR_NO_CAPABILITY

Controller no acknowledge Target Controller Role capable

DEVICE ROLE AS

LL_I3C_DEVICE_ROLE_AS_TARGET

I3C Target

LL_I3C_DEVICE_ROLE_AS_CONTROLLER

I3C Controller

DIRECTION

LL_I3C_DIRECTION_WRITE

Write transfer

LL_I3C_DIRECTION_READ

Read transfer

DMA Register Data

LL_I3C_DMA_REG_DATA_TRANSMIT_BYTE

Get address of data register used for transmission in Byte

LL_I3C_DMA_REG_DATA_RECEIVE_BYTE

Get address of data register used for reception in Byte

LL_I3C_DMA_REG_DATA_TRANSMIT_WORD

Get address of data register used for transmission in Word

LL_I3C_DMA_REG_DATA_RECEIVE_WORD

Get address of data register used for reception in Word

LL_I3C_DMA_REG_STATUS

Get address of status register used for transfer status in Word

LL_I3C_DMA_REG_CONTROL

Get address of control register used for transfer control in Word

GENERATE

LL_I3C_GENERATE_STOP

Generate Stop condition after sending a message

LL_I3C_GENERATE_RESTART

Generate Restart condition after sending a message

GETMXDS FORMAT

LL_I3C_GETMXDS_FORMAT_1

GETMXDS CCC Format 1 is used, no MaxRdTurn field in response

LL_I3C_GETMXDS_FORMAT_2_LSB

GETMXDS CCC Format 2 is used, MaxRdTurn field in response, LSB = RDTURN[7:0]

LL_I3C_GETMXDS_FORMAT_2_MID

GETMXDS CCC Format 2 is used, MaxRdTurn field in response, Middle byte = RDTURN[7:0]

LL_I3C_GETMXDS_FORMAT_2_MSB

GETMXDS CCC Format 2 is used, MaxRdTurn field in response, MSB = RDTURN[7:0]

GETMXDS TSCO

LL_I3C_TURNAROUND_TIME_TSCO_LESS_12NS

clock-to-data turnaround time tSCO <= 12ns

LL_I3C_TURNAROUND_TIME_TSCO_GREATER_12NS

clock-to-data turnaround time tSCO > 12ns

Get Flags Defines

LL_I3C_EVR_CFEF

LL_I3C_EVR_TXFEF

LL_I3C_EVR_CFNFF

LL_I3C_EVR_SFNEF

LL_I3C_EVR_TXFNFF

LL_I3C_EVR_RXFNEF

LL_I3C_EVR_RXLASTF

LL_I3C_EVR_TXLASTF

LL_I3C_EVR_FCF

LL_I3C_EVR_RXTGTENDF

LL_I3C_EVR_ERRF

LL_I3C_EVR_IBIF

LL_I3C_EVR_IBIENDF

LL_I3C_EVR_CRF

LL_I3C_EVR_CRUPDF

LL_I3C_EVR_HJF

LL_I3C_EVR_WKPF

LL_I3C_EVR_GETF

LL_I3C_EVR_STAF

LL_I3C_EVR_DAUPDF

LL_I3C_EVR_MWLUPDF

LL_I3C_EVR_MRLUPDF

LL_I3C_EVR_RSTF

LL_I3C_EVR_ASUPDF

LL_I3C_EVR_INTUPDF

LL_I3C_EVR_DEFF

LL_I3C_EVR_GRPF

LL_I3C_SER_PERR

LL_I3C_SER_STALL

LL_I3C_SER_DOVR

LL_I3C_SER_COVR

LL_I3C_SER_ANACK

LL_I3C_SER_DNACK

LL_I3C_SER_DERR

HANDOFF

LL_I3C_HANDOFF_NOT_DELAYED

Additional time to process controllership handoff is not needed

LL_I3C_HANDOFF_DELAYED

Additional time to process controllership handoff is needed

HANDOFF ACTIVITY STATE

LL_I3C_HANDOFF_ACTIVITY_STATE_0

Indicates that will act according to Activity State 0 after controllership handoff

LL_I3C_HANDOFF_ACTIVITY_STATE_1

Indicates that will act according to Activity State 1 after controllership handoff

LL_I3C_HANDOFF_ACTIVITY_STATE_2

Indicates that will act according to Activity State 2 after controllership handoff

LL_I3C_HANDOFF_ACTIVITY_STATE_3

Indicates that will act according to Activity State 3 after controllership handoff

HANDOFF GRP ADDR NOT

LL_I3C_HANDOFF_GRP_ADDR_NOT_SUPPORTED

Group Address Handoff is not supported

LL_I3C_HANDOFF_GRP_ADDR_SUPPORTED

Group Address Handoff is supported

IBI ADDITIONAL DATA

LL_I3C_IBI_DATA_ENABLE

A mandatory data byte follows the IBI acknowledgement

LL_I3C_IBI_DATA_DISABLE

No mandatory data byte follows the IBI acknowledgement

IBI CAPABILITY

LL_I3C_IBI_CAPABILITY

Controller acknowledge Target In Band Interrupt capable

LL_I3C_IBI_NO_CAPABILITY

Controller no acknowledge Target In Band Interrupt capable

IBI MDB READ NOTIFICATION

LL_I3C_MDB_NO_PENDING_READ_NOTIFICATION

No support of pending read notification via the IBI MDB[7:0] value

LL_I3C_MDB_PENDING_READ_NOTIFICATION

Support of pending read notification via the IBI MDB[7:0] value

IBI NO ADDITIONAL

LL_I3C_IBI_NO_ADDITIONAL_DATA

No data byte follows the accepted IBI

LL_I3C_IBI_ADDITIONAL_DATA

A Mandatory Data Byte (MDB) follows the accepted IBI

IT Defines

LL_I3C_IER_CFNFIE

LL_I3C_IER_SFNEIE

LL_I3C_IER_TXFNFIE

LL_I3C_IER_RXFNEIE

LL_I3C_IER_FCIE

LL_I3C_IER_RXTGTENDIE

LL_I3C_IER_ERRIE

LL_I3C_IER_IBIIE

LL_I3C_IER_IBIENDIE

LL_I3C_IER_CRIE

LL_I3C_IER_CRUPDIE

LL_I3C_IER_HJIE

LL_I3C_IER_WKPIE

LL_I3C_IER_GETIE

LL_I3C_IER_STAIE

LL_I3C_IER_DAUPDIE

LL_I3C_IER_MWLUPDIE

LL_I3C_IER_MRLUPDIE

LL_I3C_IER_RSTIE

LL_I3C_IER_ASUPDIE

LL_I3C_IER_INTUPDIE

LL_I3C_IER_DEFIE

LL_I3C_IER_GRPIE

MAX DATA SPEED LIMITATION

LL_I3C_NO_DATA_SPEED_LIMITATION

No max data speed limitation

LL_I3C_MAX_DATA_SPEED_LIMITATION

Max data speed limitation

MESSAGE

LL_I3C_MESSAGE_ERROR

An error has been detected in the message

LL_I3C_MESSAGE_SUCCESS

The message ended with success

MESSAGE DIRECTION

LL_I3C_MESSAGE_DIRECTION_WRITE

Write data or command

LL_I3C_MESSAGE_DIRECTION_READ

Read data

MODE

LL_I3C_MODE_CONTROLLER

I3C Controller mode

LL_I3C_MODE_TARGET

I3C Target (Controller capable) mode

OWN ACTIVITY STATE

LL_I3C_OWN_ACTIVITY_STATE_0

Own Controller Activity state 0

LL_I3C_OWN_ACTIVITY_STATE_1

Own Controller Activity state 1

LL_I3C_OWN_ACTIVITY_STATE_2

Own Controller Activity state 2

LL_I3C_OWN_ACTIVITY_STATE_3

Own Controller Activity state 3

PAYLOAD

LL_I3C_PAYLOAD_EMPTY

Empty payload, no additional data after IBI acknowledge

LL_I3C_PAYLOAD_1_BYTE

One additional data byte after IBI acknowledge

LL_I3C_PAYLOAD_2_BYTES

Two additional data bytes after IBI acknowledge

LL_I3C_PAYLOAD_3_BYTES

Three additional data bytes after IBI acknowledge

LL_I3C_PAYLOAD_4_BYTES

Four additional data bytes after IBI acknowledge

RESET ACTION

LL_I3C_RESET_ACTION_NONE

No Reset Action Required

LL_I3C_RESET_ACTION_PARTIAL

Reset of some internal registers of the peripheral

LL_I3C_RESET_ACTION_FULL

Reset all internal registers of the peripheral

RX THRESHOLD

LL_I3C_RXFIFO_THRESHOLD_1_4

Rx Fifo Threshold is 1 byte in a Fifo depth of 4 bytes

LL_I3C_RXFIFO_THRESHOLD_4_4

Rx Fifo Threshold is 4 bytes in a Fifo depth of 4 bytes

SDA HOLD TIME 0

LL_I3C_SDA_HOLD_TIME_0_5

SDA hold time is 0.5 x ti3cclk

LL_I3C_SDA_HOLD_TIME_1_5

SDA hold time is 1.5 x ti3cclk

TARGET ERROR

LL_I3C_TARGET_ERROR_TE0

Target detected an invalid broadcast address

LL_I3C_TARGET_ERROR_TE1

Target detected an invalid CCC Code

LL_I3C_TARGET_ERROR_TE2

Target detected an invalid write data

LL_I3C_TARGET_ERROR_TE3

Target detected an invalid assigned address during Dynamic Address Assignment procedure

LL_I3C_TARGET_ERROR_TE4

Target detected 7'h7E missing after Restart during Dynamic Address Assignment procedure

LL_I3C_TARGET_ERROR_TE5

Target detected an illegally formatted CCC

LL_I3C_TARGET_ERROR_TE6

Target detected that transmitted data on the bus is different than expected

TARGET MTYPE HOT

LL_I3C_TARGET_MTYPE_HOT_JOIN

Hot Join

LL_I3C_TARGET_MTYPE_CONTROLLER_ROLE_REQ

Controller-role Request

LL_I3C_TARGET_MTYPE_IBI

In Band Interrupt (IBI)

TX THRESHOLD

LL_I3C_TXFIFO_THRESHOLD_1_4

Tx Fifo Threshold is 1 byte in a Fifo depth of 4 bytes

LL_I3C_TXFIFO_THRESHOLD_4_4

Tx Fifo Threshold is 4 bytes in a Fifo depth of 4 bytes

Common Write and read registers Macros

LL_I3C_GET_BCR

Description:

- Get Bus Characterics in payload (64bits) receive during ENTDAA procedure.

Parameters:

- __PAYLOAD__: specifies the Bus Characteristics capabilities retrieve during ENTDAA procedure. This parameter must be a number between Min_Data=0x00(uint64_t) and Max_Data=0xFFFFFFFFFFFFFFFF.

Return value:

- The: value of BCR Return value between Min_Data=0x00 and Max_Data=0xFF.

LL_I3C_GET_IBI_CAPABLE

Description:

- Check IBI request capabilities.

Parameters:

- __BCR__: specifies the Bus Characteristics capabilities retrieve during ENTDAA procedure. This parameter must be a number between Min_Data=0x00 and Max_Data=0xFF.

Return value:

- Value: of

LL_I3C_GET_IBI_PAYLOAD

Description:

- Check IBI additional data byte capabilities.

Parameters:

- __BCR__: specifies the Bus Characteristics capabilities retrieve during ENTDAA procedure. This parameter must be a number between Min_Data=0x00 and Max_Data=0xFF.

Return value:

- Value: of

LL_I3C_GET_CR_CAPABLE

Description:

- Check Controller role request capabilities.

Parameters:

- __BCR__: specifies the Bus Characteristics capabilities retrieve during ENTDAA procedure. This parameter must be a number between Min_Data=0x00 and Max_Data=0xFF.

Return value:

- Value: of

LL_I3C_WriteReg

Description:

- Write a value in I3C register.

Parameters:

- __INSTANCE__: I3C Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_I3C_ReadReg

Description:

- Read a value in I3C register.

Parameters:

- __INSTANCE__: I3C Instance
- __REG__: Register to be read

Return value:

- Register: value

96 LL ICACHE Generic Driver

96.1 ICACHE Firmware driver API description

The following section lists the various functions of the ICACHE library.

96.1.1 Detailed description of functions

LL_ICACHE_Enable

Function name

__STATIC_INLINE void LL_ICACHE_Enable (void)

Function description

Enable the ICACHE.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR EN LL_ICACHE_Enable

LL_ICACHE_Disable

Function name

__STATIC_INLINE void LL_ICACHE_Disable (void)

Function description

Disable the ICACHE.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR EN LL_ICACHE_Disable

LL_ICACHE_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_ICACHE_IsEnabled (void)

Function description

Return if ICACHE is enabled or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR EN LL_ICACHE_IsEnabled

LL_ICACHE_SetMode

Function name

__STATIC_INLINE void LL_ICACHE_SetMode (uint32_t Mode)

Function description

Select the ICACHE operating mode.

Parameters

- **Mode:** This parameter can be one of the following values:
  - LL_ICACHE_1WAY
  - LL_ICACHE_4WAYS

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR WAYSEL LL_ICACHE_SetMode

LL_ICACHE_GetMode

Function name

__STATIC_INLINE uint32_t LL_ICACHE_GetMode (void)

Function description

Get the selected ICACHE operating mode.

Return values

- **Returned:** value can be one of the following values:
  - LL_ICACHE_1WAY
  - LL_ICACHE_4WAYS

Reference Manual to LL API cross reference:

- CR WAYSEL LL_ICACHE_GetMode

LL_ICACHE_Invalidate

Function name

__STATIC_INLINE void LL_ICACHE_Invalidate (void)

Function description

Invalidate the ICACHE.

Return values

- **None:**

Notes

- Until the BSYEND flag is set, the cache is bypassed.

Reference Manual to LL API cross reference:

- CR CACHEINV LL_ICACHE_Invalidate

LL_ICACHE_EnableMonitors

Function name

__STATIC_INLINE void LL_ICACHE_EnableMonitors (uint32_t Monitors)

Function description

Enable the hit/miss monitor(s).

Parameters

- **Monitors:** This parameter can be one or a combination of the following values:
  - LL_ICACHE_MONITOR_HIT
  - LL_ICACHE_MONITOR_MISS
  - LL_ICACHE_MONITOR_ALL

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR HITMEN LL_ICACHE_EnableMonitors
- CR MISSMEN LL_ICACHE_EnableMonitors

LL_ICACHE_DisableMonitors

Function name

__STATIC_INLINE void LL_ICACHE_DisableMonitors (uint32_t Monitors)

Function description

Disable the hit/miss monitor(s).

Parameters

- **Monitors:** This parameter can be one or a combination of the following values:
  - LL_ICACHE_MONITOR_HIT
  - LL_ICACHE_MONITOR_MISS
  - LL_ICACHE_MONITOR_ALL

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR HITMEN LL_ICACHE_DisableMonitors
- CR MISSMEN LL_ICACHE_DisableMonitors

LL_ICACHE_IsEnabledMonitors

Function name

__STATIC_INLINE uint32_t LL_ICACHE_IsEnabledMonitors (uint32_t Monitors)

Function description

Check if the monitor(s) is(are) enabled or disabled.

Parameters

- **Monitors:** This parameter can be one or a combination of the following values:
  - LL_ICACHE_MONITOR_HIT
  - LL_ICACHE_MONITOR_MISS
  - LL_ICACHE_MONITOR_ALL

Return values

- **State:** of parameter value (1 or 0).

Reference Manual to LL API cross reference:

- CR HITMEN LL_ICACHE_IsEnabledMonitors
- CR MISSMEN LL_ICACHE_IsEnabledMonitors

LL_ICACHE_ResetMonitors

Function name

__STATIC_INLINE void LL_ICACHE_ResetMonitors (uint32_t Monitors)

Function description

Reset the hit/miss monitor(s).

Parameters

- **Monitors:** This parameter can be one or a combination of the following values:
  - LL_ICACHE_MONITOR_HIT
  - LL_ICACHE_MONITOR_MISS
  - LL_ICACHE_MONITOR_ALL

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR HITMRST LL_ICACHE_ResetMonitors
- CR MISSMRST LL_ICACHE_ResetMonitors

LL_ICACHE_GetHitMonitor

Function name

__STATIC_INLINE uint32_t LL_ICACHE_GetHitMonitor (void)

Function description

Get the Hit monitor.

Return values

- **Value:** between Min_Data=0 and Max_Data=0xFFFFFFFF

Notes

- Upon reaching the 32-bit maximum value, hit monitor does not wrap.

Reference Manual to LL API cross reference:

- HMONR HITMON LL_ICACHE_GetHitMonitor

LL_ICACHE_GetMissMonitor

Function name

__STATIC_INLINE uint32_t LL_ICACHE_GetMissMonitor (void)

Function description

Get the Miss monitor.

Return values

- **Value:** between Min_Data=0 and Max_Data=0xFFFF

Notes

- Upon reaching the 16-bit maximum value, miss monitor does not wrap.

Reference Manual to LL API cross reference:

- MMONR MISSMON LL_ICACHE_GetMissMonitor

LL_ICACHE_EnableIT_BSYEND

Function name

__STATIC_INLINE void LL_ICACHE_EnableIT_BSYEND (void)

Function description

Enable BSYEND interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER BSYENDIE LL_ICACHE_EnableIT_BSYEND

LL_ICACHE_DisableIT_BSYEND

Function name

__STATIC_INLINE void LL_ICACHE_DisableIT_BSYEND (void)

Function description

Disable BSYEND interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER BSYENDIE LL_ICACHE_DisableIT_BSYEND

LL_ICACHE_IsEnabledIT_BSYEND

Function name

__STATIC_INLINE uint32_t LL_ICACHE_IsEnabledIT_BSYEND (void)

Function description

Check if the BSYEND Interrupt is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER BSYENDIE LL_ICACHE_IsEnabledIT_BSYEND

LL_ICACHE_EnableIT_ERR

Function name

__STATIC_INLINE void LL_ICACHE_EnableIT_ERR (void)

Function description

Enable ERR interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER ERRIE LL_ICACHE_EnableIT_ERR

LL_ICACHE_DisableIT_ERR

Function name

__STATIC_INLINE void LL_ICACHE_DisableIT_ERR (void)

Function description

Disable ERR interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER ERRIE LL_ICACHE_DisableIT_ERR

LL_ICACHE_IsEnabledIT_ERR

Function name

__STATIC_INLINE uint32_t LL_ICACHE_IsEnabledIT_ERR (void)

Function description

Check if the ERR Interrupt is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER ERRIE LL_ICACHE_IsEnabledIT_ERR

LL_ICACHE_IsActiveFlag_BUSY

Function name

__STATIC_INLINE uint32_t LL_ICACHE_IsActiveFlag_BUSY (void)

Function description

Indicate the status of an ongoing operation flag.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR BUSYF LL_ICACHE_IsActiveFlag_BUSY

LL_ICACHE_IsActiveFlag_BSYEND

Function name

__STATIC_INLINE uint32_t LL_ICACHE_IsActiveFlag_BSYEND (void)

Function description

Indicate the status of an operation end flag.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR BSYEND LL_ICACHE_IsActiveFlag_BSYEND

LL_ICACHE_IsActiveFlag_ERR

Function name

__STATIC_INLINE uint32_t LL_ICACHE_IsActiveFlag_ERR (void)

Function description

Indicate the status of an error flag.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR ERRF LL_ICACHE_IsActiveFlag_ERR

LL_ICACHE_ClearFlag_BSYEND

Function name

__STATIC_INLINE void LL_ICACHE_ClearFlag_BSYEND (void)

Function description

Clear busy end of operation flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- FCR CBSYENDF LL_ICACHE_ClearFlag_BSYEND

LL_ICACHE_ClearFlag_ERR

Function name

__STATIC_INLINE void LL_ICACHE_ClearFlag_ERR (void)

Function description

Clear error flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- FCR ERRF LL_ICACHE_ClearFlag_ERR

96.2 ICACHE Firmware driver defines

The following section lists the various define and macros of the module.

96.2.1 ICACHE

ICACHE

Clear Flags Defines

LL_ICACHE_FCR_CBSYENDF

Busy end flag

LL_ICACHE_FCR_CERRF

Cache error flag

Get Flags Defines

LL_ICACHE_SR_BUSYF

Busy flag

LL_ICACHE_SR_BSYENDF

Busy end flag

LL_ICACHE_SR_ERRF

Cache error flag

IT Defines

LL_ICACHE_IER_BSYENDIE

Busy end interrupt

LL_ICACHE_IER_ERRIE

Cache error interrupt

Monitor type

LL_ICACHE_MONITOR_HIT

Hit monitor counter

LL_ICACHE_MONITOR_MISS

Miss monitor counter

LL_ICACHE_MONITOR_ALL

All monitors counters

Ways selection

LL_ICACHE_1WAY

1-way cache (direct mapped cache)

LL_ICACHE_4WAYS

4-ways set associative cache (default)

Common write and read registers Macros

LL_ICACHE_WriteReg

Description:

- Write a value in ICACHE register.

Parameters:

- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_ICACHE_ReadReg

Description:

- Read a value in ICACHE register.

Parameters:

- __REG__: Register to be read

Return value:

- Register: value

97 LL IWDG Generic Driver

97.1 IWDG Firmware driver API description

The following section lists the various functions of the IWDG library.

97.1.1 Detailed description of functions

LL_IWDG_Enable

Function name

__STATIC_INLINE void LL_IWDG_Enable (IWDG_TypeDef * IWDGx)

Function description

Start the Independent Watchdog.

Parameters

- **IWDGx:** IWDG Instance

Return values

- **None:**

Notes

- Except if the hardware watchdog option is selected

Reference Manual to LL API cross reference:

- KR KEY LL_IWDG_Enable

LL_IWDG_ReloadCounter

Function name

__STATIC_INLINE void LL_IWDG_ReloadCounter (IWDG_TypeDef * IWDGx)

Function description

Reloads IWDG counter with value defined in the reload register.

Parameters

- **IWDGx:** IWDG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- KR KEY LL_IWDG_ReloadCounter

LL_IWDG_EnableWriteAccess

Function name

__STATIC_INLINE void LL_IWDG_EnableWriteAccess (IWDG_TypeDef * IWDGx)

Function description

Enable write access to IWDG_PR, IWDG_RLR and IWDG_WINR registers.

Parameters

- **IWDGx:** IWDG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- KR KEY LL_IWDG_EnableWriteAccess

LL_IWDG_DisableWriteAccess

Function name

__STATIC_INLINE void LL_IWDG_DisableWriteAccess (IWDG_TypeDef * IWDGx)

Function description

Disable write access to IWDG_PR, IWDG_RLR and IWDG_WINR registers.

Parameters

- **IWDGx:** IWDG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- KR KEY LL_IWDG_DisableWriteAccess

LL_IWDG_SetPrescaler

Function name

__STATIC_INLINE void LL_IWDG_SetPrescaler (IWDG_TypeDef * IWDGx, uint32_t Prescaler)

Function description

Select the prescaler of the IWDG.

Parameters

- **IWDGx:** IWDG Instance
- **Prescaler:** This parameter can be one of the following values:
  - LL_IWDG_PRESCALER_4
  - LL_IWDG_PRESCALER_8
  - LL_IWDG_PRESCALER_16
  - LL_IWDG_PRESCALER_32
  - LL_IWDG_PRESCALER_64
  - LL_IWDG_PRESCALER_128
  - LL_IWDG_PRESCALER_256
  - LL_IWDG_PRESCALER_512
  - LL_IWDG_PRESCALER_1024

Return values

- **None:**

Reference Manual to LL API cross reference:

- PR PR LL_IWDG_SetPrescaler

LL_IWDG_GetPrescaler

Function name

__STATIC_INLINE uint32_t LL_IWDG_GetPrescaler (const IWDG_TypeDef * IWDGx)

Function description

Get the selected prescaler of the IWDG.

Parameters

- **IWDGx:** IWDG Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_IWDG_PRESCALER_4
  - LL_IWDG_PRESCALER_8
  - LL_IWDG_PRESCALER_16
  - LL_IWDG_PRESCALER_32
  - LL_IWDG_PRESCALER_64
  - LL_IWDG_PRESCALER_128
  - LL_IWDG_PRESCALER_256
  - LL_IWDG_PRESCALER_512
  - LL_IWDG_PRESCALER_1024

Reference Manual to LL API cross reference:

- PR PR LL_IWDG_GetPrescaler

LL_IWDG_SetReloadCounter

Function name

__STATIC_INLINE void LL_IWDG_SetReloadCounter (IWDG_TypeDef * IWDGx, uint32_t Counter)

Function description

Specify the IWDG down-counter reload value.

Parameters

- **IWDGx:** IWDG Instance
- **Counter:** Value between Min_Data=0 and Max_Data=0x0FFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- RLR RL LL_IWDG_SetReloadCounter

LL_IWDG_GetReloadCounter

Function name

__STATIC_INLINE uint32_t LL_IWDG_GetReloadCounter (const IWDG_TypeDef * IWDGx)

Function description

Get the specified IWDG down-counter reload value.

Parameters

- **IWDGx:** IWDG Instance

Return values

- **Value:** between Min_Data=0 and Max_Data=0x0FFF

Reference Manual to LL API cross reference:

- RLR RL LL_IWDG_GetReloadCounter

LL_IWDG_SetWindow

Function name

__STATIC_INLINE void LL_IWDG_SetWindow (IWDG_TypeDef * IWDGx, uint32_t Window)

Function description

Specify high limit of the window value to be compared to the down-counter.

Parameters

- **IWDGx:** IWDG Instance
- **Window:** Value between Min_Data=0 and Max_Data=0x0FFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- WINR WIN LL_IWDG_SetWindow

LL_IWDG_GetWindow

Function name

__STATIC_INLINE uint32_t LL_IWDG_GetWindow (const IWDG_TypeDef * IWDGx)

Function description

Get the high limit of the window value specified.

Parameters

- **IWDGx:** IWDG Instance

Return values

- **Value:** between Min_Data=0 and Max_Data=0x0FFF

Reference Manual to LL API cross reference:

- WINR WIN LL_IWDG_GetWindow

LL_IWDG_SetEwiTime

Function name

__STATIC_INLINE void LL_IWDG_SetEwiTime (IWDG_TypeDef * IWDGx, uint32_t Time)

Function description

Specify comparator value that will be used to trig Early Wakeup interrupt.

Parameters

- **IWDGx:** IWDG Instance
- **Time:** Value between Min_Data=0 and Max_Data=0x0FFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- EWCR EWIT LL_IWDG_SetEwiTime

LL_IWDG_GetEwiTime

Function name

__STATIC_INLINE uint32_t LL_IWDG_GetEwiTime (const IWDG_TypeDef * IWDGx)

Function description

Get the Early Wakeup interrupt comparator value.

Parameters

- **IWDGx:** IWDG Instance

Return values

- **Value:** between Min_Data=0 and Max_Data=0x0FFF

Reference Manual to LL API cross reference:

- EWCR EWIT LL_IWDG_GetEwiTime

LL_IWDG_EnableIT_EWI

Function name

__STATIC_INLINE void LL_IWDG_EnableIT_EWI (IWDG_TypeDef * IWDGx)

Function description

Enable Early wakeup interrupt.

Parameters

- **IWDGx:** IWDG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- EWCR EWIE LL_IWDG_EnableIT_EWI

LL_IWDG_DisableIT_EWI

Function name

__STATIC_INLINE void LL_IWDG_DisableIT_EWI (IWDG_TypeDef * IWDGx)

Function description

Disable Early wakeup interrupt.

Parameters

- **IWDGx:** IWDG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- EWCR EWIE LL_IWDG_DisableIT_EWI

LL_IWDG_IsEnabledIT_EWI

Function name

__STATIC_INLINE uint32_t LL_IWDG_IsEnabledIT_EWI (const IWDG_TypeDef * IWDGx)

Function description

Indicates whether Early wakeup interrupt is enable.

Parameters

- **IWDGx:** IWDG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- EWCR EWIE LL_IWDG_IsEnabledIT_EWI

LL_IWDG_IsActiveFlag_PVU

Function name

__STATIC_INLINE uint32_t LL_IWDG_IsActiveFlag_PVU (const IWDG_TypeDef * IWDGx)

Function description

Check if flag Prescaler Value Update is set or not.

Parameters

- **IWDGx:** IWDG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR PVU LL_IWDG_IsActiveFlag_PVU

LL_IWDG_IsActiveFlag_RVU

Function name

__STATIC_INLINE uint32_t LL_IWDG_IsActiveFlag_RVU (const IWDG_TypeDef * IWDGx)

Function description

Check if flag Reload Value Update is set or not.

Parameters

- **IWDGx:** IWDG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR RVU LL_IWDG_IsActiveFlag_RVU

LL_IWDG_IsActiveFlag_WVU

Function name

__STATIC_INLINE uint32_t LL_IWDG_IsActiveFlag_WVU (const IWDG_TypeDef * IWDGx)

Function description

Check if flag Window Value Update is set or not.

Parameters

- **IWDGx:** IWDG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR WVU LL_IWDG_IsActiveFlag_WVU

LL_IWDG_IsActiveFlag_EWU

Function name

__STATIC_INLINE uint32_t LL_IWDG_IsActiveFlag_EWU (const IWDG_TypeDef * IWDGx)

Function description

Check if flag EWI Value Update is set or not.

Parameters

- **IWDGx:** IWDG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR EVU LL_IWDG_IsActiveFlag_EWU

LL_IWDG_IsReady

Function name

__STATIC_INLINE uint32_t LL_IWDG_IsReady (const IWDG_TypeDef * IWDGx)

Function description

Check if all flags Prescaler, Reload, Window & Early Interrupt Value Update are reset or not.

Parameters

- **IWDGx:** IWDG Instance

Return values

- **State:** of bits (1 or 0).

Reference Manual to LL API cross reference:

- SR PVU LL_IWDG_IsReady
- SR RVU LL_IWDG_IsReady
- SR WVU LL_IWDG_IsReady
- SR EWU LL_IWDG_IsReady

LL_IWDG_IsActiveFlag_ONF

Function name

__STATIC_INLINE uint32_t LL_IWDG_IsActiveFlag_ONF (const IWDG_TypeDef * IWDGx)

Function description

Check if IWDG has been started or not.

Parameters

- **IWDGx:** IWDG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR ONF LL_IWDG_IsActiveFlag_ONF

LL_IWDG_IsActiveFlag_EWIF

Function name

__STATIC_INLINE uint32_t LL_IWDG_IsActiveFlag_EWIF (const IWDG_TypeDef * IWDGx)

Function description

Check if Early Wakeup interrupt flag is set or not.

Parameters

- **IWDGx:** IWDG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR EWIF LL_IWDG_IsActiveFlag_EWIF

LL_IWDG_ClearFlag_EWIF

Function name

__STATIC_INLINE void LL_IWDG_ClearFlag_EWIF (IWDG_TypeDef * IWDGx)

Function description

Clear the Early Wakeup interrupt flag.

Parameters

- **IWDGx:** IWDG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- EWCR ICR LL_IWDG_ClearFlag_EWIF

97.2 IWDG Firmware driver defines

The following section lists the various define and macros of the module.

97.2.1 IWDG

IWDG

Get Flags Defines

LL_IWDG_SR_PVU

Watchdog prescaler value update

LL_IWDG_SR_RVU

Watchdog counter reload value update

LL_IWDG_SR_WVU

Watchdog counter window value update

Prescaler Divider

LL_IWDG_PRESCALER_4

Divider by 4

LL_IWDG_PRESCALER_8

Divider by 8

LL_IWDG_PRESCALER_16

Divider by 16

LL_IWDG_PRESCALER_32

Divider by 32

LL_IWDG_PRESCALER_64

Divider by 64

LL_IWDG_PRESCALER_128

Divider by 128

LL_IWDG_PRESCALER_256

Divider by 256

LL_IWDG_PRESCALER_512

Divider by 512

LL_IWDG_PRESCALER_1024

Divider by 1024

Common Write and read registers Macros

LL_IWDG_WriteReg

Description:

- Write a value in IWDG register.

Parameters:

- __INSTANCE__: IWDG Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_IWDG_ReadReg

Description:

- Read a value in IWDG register.

Parameters:

- __INSTANCE__: IWDG Instance
- __REG__: Register to be read

Return value:

- Register: value

98 LL LPTIM Generic Driver

98.1 LPTIM Firmware driver registers structures

98.1.1 LL_LPTIM_InitTypeDef

LL_LPTIM_InitTypeDef is defined in the stm32n6xx_ll_lptim.h

Data Fields

- ***uint32_t ClockSource***
- ***uint32_t Prescaler***
- ***uint32_t Waveform***

Field Documentation

- ***uint32_t LL_LPTIM_InitTypeDef::ClockSource***
  Specifies the source of the clock used by the LPTIM instance. This parameter can be a value of ***LPTIM_LL_EC_CLK_SOURCE***.This feature can be modified afterwards using unitary function **LL_LPTIM_SetClockSource()**.
- ***uint32_t LL_LPTIM_InitTypeDef::Prescaler***
  Specifies the prescaler division ratio. This parameter can be a value of ***LPTIM_LL_EC_PRESCALER***.This feature can be modified afterwards using using unitary function **LL_LPTIM_SetPrescaler()**.
- ***uint32_t LL_LPTIM_InitTypeDef::Waveform***
  Specifies the waveform shape. This parameter can be a value of ***LPTIM_LL_EC_OUTPUT_WAVEFORM***.This feature can be modified afterwards using unitary function **LL_LPTIM_SetWaveform()**.

98.2 LPTIM Firmware driver API description

The following section lists the various functions of the LPTIM library.

98.2.1 Detailed description of functions

LL_LPTIM_DeInit

Function name

ErrorStatus LL_LPTIM_DeInit (const LPTIM_TypeDef * LPTIMx)

Function description

Set LPTIMx registers to their reset values.

Parameters

- **LPTIMx:** LP Timer instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: LPTIMx registers are de-initialized
  - ERROR: invalid LPTIMx instance

LL_LPTIM_StructInit

Function name

void LL_LPTIM_StructInit (LL_LPTIM_InitTypeDef * LPTIM_InitStruct)

Function description

Set each fields of the LPTIM_InitStruct structure to its default value.

Parameters

- **LPTIM_InitStruct:** pointer to a LL_LPTIM_InitTypeDef structure

Return values

- **None:**

LL_LPTIM_Init

Function name

ErrorStatus LL_LPTIM_Init (LPTIM_TypeDef * LPTIMx, const LL_LPTIM_InitTypeDef * LPTIM_InitStruct)

Function description

Configure the LPTIMx peripheral according to the specified parameters.

Parameters

- **LPTIMx:** LP Timer Instance
- **LPTIM_InitStruct:** pointer to a LL_LPTIM_InitTypeDef structure

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: LPTIMx instance has been initialized
  - ERROR: LPTIMx instance hasn't been initialized

Notes

- LL_LPTIM_Init can only be called when the LPTIM instance is disabled.
- LPTIMx can be disabled using unitary function LL_LPTIM_Disable().

LL_LPTIM_Enable

Function name

__STATIC_INLINE void LL_LPTIM_Enable (LPTIM_TypeDef * LPTIMx)

Function description

Enable the LPTIM instance.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Notes

- After setting the ENABLE bit, a delay of two counter clock is needed before the LPTIM instance is actually enabled.

Reference Manual to LL API cross reference:

- CR ENABLE LL_LPTIM_Enable

LL_LPTIM_Disable

Function name

__STATIC_INLINE void LL_LPTIM_Disable (LPTIM_TypeDef * LPTIMx)

Function description

Disable the LPTIM instance.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR ENABLE LL_LPTIM_Disable

LL_LPTIM_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabled (const LPTIM_TypeDef * LPTIMx)

Function description

Indicates whether the LPTIM instance is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR ENABLE LL_LPTIM_IsEnabled

LL_LPTIM_StartCounter

Function name

__STATIC_INLINE void LL_LPTIM_StartCounter (LPTIM_TypeDef * LPTIMx, uint32_t OperatingMode)

Function description

Starts the LPTIM counter in the desired mode.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **OperatingMode:** This parameter can be one of the following values:
  - LL_LPTIM_OPERATING_MODE_CONTINUOUS
  - LL_LPTIM_OPERATING_MODE_ONESHOT

Return values

- **None:**

Notes

- LPTIM instance must be enabled before starting the counter.
- It is possible to change on the fly from One Shot mode to Continuous mode.

Reference Manual to LL API cross reference:

- CR CNTSTRT LL_LPTIM_StartCounter
- CR SNGSTRT LL_LPTIM_StartCounter

LL_LPTIM_EnableResetAfterRead

Function name

__STATIC_INLINE void LL_LPTIM_EnableResetAfterRead (LPTIM_TypeDef * LPTIMx)

Function description

Enable reset after read.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Notes

- After calling this function any read access to LPTIM_CNT register will asynchronously reset the LPTIM_CNT register content.

Reference Manual to LL API cross reference:

- CR RSTARE LL_LPTIM_EnableResetAfterRead

LL_LPTIM_DisableResetAfterRead

Function name

__STATIC_INLINE void LL_LPTIM_DisableResetAfterRead (LPTIM_TypeDef * LPTIMx)

Function description

Disable reset after read.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR RSTARE LL_LPTIM_DisableResetAfterRead

LL_LPTIM_IsEnabledResetAfterRead

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledResetAfterRead (const LPTIM_TypeDef * LPTIMx)

Function description

Indicate whether the reset after read feature is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR RSTARE LL_LPTIM_IsEnabledResetAfterRead

LL_LPTIM_ResetCounter

Function name

__STATIC_INLINE void LL_LPTIM_ResetCounter (LPTIM_TypeDef * LPTIMx)

Function description

Reset of the LPTIM_CNT counter register (synchronous).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Notes

- Due to the synchronous nature of this reset, it only takes place after a synchronization delay of 3 LPTIM core clock cycles (LPTIM core clock may be different from APB clock).
- COUNTRST is automatically cleared by hardware

Reference Manual to LL API cross reference:

- CR COUNTRST LL_LPTIM_ResetCounter
-

LL_LPTIM_SetUpdateMode

Function name

__STATIC_INLINE void LL_LPTIM_SetUpdateMode (LPTIM_TypeDef * LPTIMx, uint32_t UpdateMode)

Function description

Set the LPTIM registers update mode (enable/disable register preload)

Parameters

- **LPTIMx:** Low-Power Timer instance
- **UpdateMode:** This parameter can be one of the following values:
  - LL_LPTIM_UPDATE_MODE_IMMEDIATE
  - LL_LPTIM_UPDATE_MODE_ENDOFPERIOD

Return values

- **None:**

Notes

- This function must be called when the LPTIM instance is disabled.

Reference Manual to LL API cross reference:

- CFGR PRELOAD LL_LPTIM_SetUpdateMode

LL_LPTIM_GetUpdateMode

Function name

__STATIC_INLINE uint32_t LL_LPTIM_GetUpdateMode (const LPTIM_TypeDef * LPTIMx)

Function description

Get the LPTIM registers update mode.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPTIM_UPDATE_MODE_IMMEDIATE
  - LL_LPTIM_UPDATE_MODE_ENDOFPERIOD

Reference Manual to LL API cross reference:

- CFGR PRELOAD LL_LPTIM_GetUpdateMode

LL_LPTIM_SetAutoReload

Function name

__STATIC_INLINE void LL_LPTIM_SetAutoReload (LPTIM_TypeDef * LPTIMx, uint32_t AutoReload)

Function description

Set the auto reload value.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **AutoReload:** Value between Min_Data=0x0001 and Max_Data=0xFFFF

Return values

- **None:**

Notes

- The LPTIMx_ARR register content must only be modified when the LPTIM is enabled
- After a write to the LPTIMx_ARR register a new write operation to the same register can only be performed when the previous write operation is completed. Any successive write before the ARROK flag is set, will lead to unpredictable results.
- autoreload value be strictly greater than the compare value.

Reference Manual to LL API cross reference:

- ARR ARR LL_LPTIM_SetAutoReload

LL_LPTIM_GetAutoReload

Function name

__STATIC_INLINE uint32_t LL_LPTIM_GetAutoReload (const LPTIM_TypeDef * LPTIMx)

Function description

Get actual auto reload value.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **AutoReload:** Value between Min_Data=0x0001 and Max_Data=0xFFFF

Reference Manual to LL API cross reference:

- ARR ARR LL_LPTIM_GetAutoReload

LL_LPTIM_SetRepetition

Function name

__STATIC_INLINE void LL_LPTIM_SetRepetition (LPTIM_TypeDef * LPTIMx, uint32_t Repetition)

Function description

Set the repetition value.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **Repetition:** Value between Min_Data=0x00 and Max_Data=0xFF

Return values

- **None:**

Notes

- The LPTIMx_RCR register content must only be modified when the LPTIM is enabled

Reference Manual to LL API cross reference:

- RCR REP LL_LPTIM_SetRepetition

LL_LPTIM_GetRepetition

Function name

__STATIC_INLINE uint32_t LL_LPTIM_GetRepetition (const LPTIM_TypeDef * LPTIMx)

Function description

Get the repetition value.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **Repetition:** Value between Min_Data=0x00 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- RCR REP LL_LPTIM_GetRepetition

LL_LPTIM_CC_EnableChannel

Function name

__STATIC_INLINE void LL_LPTIM_CC_EnableChannel (LPTIM_TypeDef * LPTIMx, uint32_t Channel)

Function description

Enable capture/compare channel.

Parameters

- **LPTIMx:** LPTimer instance
- **Channel:** This parameter can be one of the following values:
  - LL_LPTIM_CHANNEL_CH1
  - LL_LPTIM_CHANNEL_CH2

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCMR1 CC1E LL_LPTIM_CC_EnableChannel
- CCMR1 CC2E LL_LPTIM_CC_EnableChannel

LL_LPTIM_CC_DisableChannel

Function name

__STATIC_INLINE void LL_LPTIM_CC_DisableChannel (LPTIM_TypeDef * LPTIMx, uint32_t Channel)

Function description

Disable capture/compare channel.

Parameters

- **LPTIMx:** LPTimer instance
- **Channel:** This parameter can be one of the following values:
  - LL_LPTIM_CHANNEL_CH1
  - LL_LPTIM_CHANNEL_CH2

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCMR1 CC1E LL_LPTIM_CC_DisableChannel
- CCMR1 CC2E LL_LPTIM_CC_DisableChannel

LL_LPTIM_CC_IsEnabledChannel

Function name

__STATIC_INLINE uint32_t LL_LPTIM_CC_IsEnabledChannel (const LPTIM_TypeDef * LPTIMx, uint32_t Channel)

Function description

Indicate whether channel is enabled.

Parameters

- **LPTIMx:** LPTimer instance
- **Channel:** This parameter can be one of the following values:
  - LL_LPTIM_CHANNEL_CH1
  - LL_LPTIM_CHANNEL_CH2

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CCMR1 CC1E LL_LPTIM_CC_IsEnabledChannel
- CCMR1 CC2E LL_LPTIM_CC_IsEnabledChannel

LL_LPTIM_OC_SetCompareCH1

Function name

__STATIC_INLINE void LL_LPTIM_OC_SetCompareCH1 (LPTIM_TypeDef * LPTIMx, uint32_t CompareValue)

Function description

Set the compare value.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **CompareValue:** Value between Min_Data=0x00 and Max_Data=0xFFFF

Return values

- **None:**

Notes

- After a write to the LPTIMx_CCR1 register a new write operation to the same register can only be performed when the previous write operation is completed. Any successive write before the CMP1OK flag is set, will lead to unpredictable results.

Reference Manual to LL API cross reference:

- CCR1 CCR1 LL_LPTIM_OC_SetCompareCH1

LL_LPTIM_OC_GetCompareCH1

Function name

__STATIC_INLINE uint32_t LL_LPTIM_OC_GetCompareCH1 (const LPTIM_TypeDef * LPTIMx)

Function description

Get actual compare value.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **CompareValue:** Value between Min_Data=0x00 and Max_Data=0xFFFF

Reference Manual to LL API cross reference:

- CCR1 CCR1 LL_LPTIM_OC_GetCompareCH1

LL_LPTIM_OC_SetCompareCH2

Function name

__STATIC_INLINE void LL_LPTIM_OC_SetCompareCH2 (LPTIM_TypeDef * LPTIMx, uint32_t CompareValue)

Function description

Set the compare value.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **CompareValue:** Value between Min_Data=0x00 and Max_Data=0xFFFF

Return values

- **None:**

Notes

- After a write to the LPTIMx_CCR2 register a new write operation to the same register can only be performed when the previous write operation is completed. Any successive write before the CMP2OK flag is set, will lead to unpredictable results.

Reference Manual to LL API cross reference:

- CCR2 CCR2 LL_LPTIM_OC_SetCompareCH2

LL_LPTIM_OC_GetCompareCH2

Function name

__STATIC_INLINE uint32_t LL_LPTIM_OC_GetCompareCH2 (const LPTIM_TypeDef * LPTIMx)

Function description

Get actual compare value.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **CompareValue:** Value between Min_Data=0x00 and Max_Data=0xFFFF

Reference Manual to LL API cross reference:

- CCR2 CCR2 LL_LPTIM_OC_GetCompareCH2

LL_LPTIM_GetCounter

Function name

__STATIC_INLINE uint32_t LL_LPTIM_GetCounter (const LPTIM_TypeDef * LPTIMx)

Function description

Get actual counter value.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **Counter:** value

Notes

- When the LPTIM instance is running with an asynchronous clock, reading the LPTIMx_CNT register may return unreliable values. So in this case it is necessary to perform two consecutive read accesses and verify that the two returned values are identical.

Reference Manual to LL API cross reference:

- CNT CNT LL_LPTIM_GetCounter

LL_LPTIM_SetCounterMode

Function name

__STATIC_INLINE void LL_LPTIM_SetCounterMode (LPTIM_TypeDef * LPTIMx, uint32_t CounterMode)

Function description

Set the counter mode (selection of the LPTIM counter clock source).

Parameters

- **LPTIMx:** Low-Power Timer instance
- **CounterMode:** This parameter can be one of the following values:
  - LL_LPTIM_COUNTER_MODE_INTERNAL
  - LL_LPTIM_COUNTER_MODE_EXTERNAL

Return values

- **None:**

Notes

- The counter mode can be set only when the LPTIM instance is disabled.

Reference Manual to LL API cross reference:

- CFGR COUNTMODE LL_LPTIM_SetCounterMode

LL_LPTIM_GetCounterMode

Function name

__STATIC_INLINE uint32_t LL_LPTIM_GetCounterMode (const LPTIM_TypeDef * LPTIMx)

Function description

Get the counter mode.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPTIM_COUNTER_MODE_INTERNAL
  - LL_LPTIM_COUNTER_MODE_EXTERNAL

Reference Manual to LL API cross reference:

- CFGR COUNTMODE LL_LPTIM_GetCounterMode

LL_LPTIM_SetWaveform

Function name

__STATIC_INLINE void LL_LPTIM_SetWaveform (LPTIM_TypeDef * LPTIMx, uint32_t Waveform)

Function description

Set waveform shape.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **Waveform:** This parameter can be one of the following values:
  - LL_LPTIM_OUTPUT_WAVEFORM_PWM
  - LL_LPTIM_OUTPUT_WAVEFORM_SETONCE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR WAVE LL_LPTIM_SetWaveform

LL_LPTIM_GetWaveform

Function name

__STATIC_INLINE uint32_t LL_LPTIM_GetWaveform (const LPTIM_TypeDef * LPTIMx)

Function description

Get actual waveform shape.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPTIM_OUTPUT_WAVEFORM_PWM
  - LL_LPTIM_OUTPUT_WAVEFORM_SETONCE

Reference Manual to LL API cross reference:

- CFGR WAVE LL_LPTIM_GetWaveform

LL_LPTIM_OC_SetPolarity

Function name

__STATIC_INLINE void LL_LPTIM_OC_SetPolarity (LPTIM_TypeDef * LPTIMx, uint32_t Channel, uint32_t Polarity)

Function description

Set the polarity of an output channel.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_LPTIM_CHANNEL_CH1
  - LL_LPTIM_CHANNEL_CH2
- **Polarity:** This parameter can be one of the following values:
  - LL_LPTIM_OUTPUT_POLARITY_REGULAR
  - LL_LPTIM_OUTPUT_POLARITY_INVERSE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCMR1 CC1P LL_LPTIM_OC_SetPolarity
-
- CCMR1 CC2P LL_LPTIM_OC_SetPolarity
-

LL_LPTIM_OC_GetPolarity

Function name

__STATIC_INLINE uint32_t LL_LPTIM_OC_GetPolarity (const LPTIM_TypeDef * LPTIMx, uint32_t Channel)

Function description

Get the polarity of an output channel.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_LPTIM_CHANNEL_CH1
  - LL_LPTIM_CHANNEL_CH2

Return values

- **Returned:** value can be one of the following values:
  - LL_LPTIM_OUTPUT_POLARITY_REGULAR
  - LL_LPTIM_OUTPUT_POLARITY_INVERSE

Reference Manual to LL API cross reference:

- CCMR1 CC1P LL_LPTIM_OC_GetPolarity
-
- CCMR1 CC2P LL_LPTIM_OC_GetPolarity
-

LL_LPTIM_SetPrescaler

Function name

__STATIC_INLINE void LL_LPTIM_SetPrescaler (LPTIM_TypeDef * LPTIMx, uint32_t Prescaler)

Function description

Set actual prescaler division ratio.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **Prescaler:** This parameter can be one of the following values:
  - LL_LPTIM_PRESCALER_DIV1
  - LL_LPTIM_PRESCALER_DIV2
  - LL_LPTIM_PRESCALER_DIV4
  - LL_LPTIM_PRESCALER_DIV8
  - LL_LPTIM_PRESCALER_DIV16
  - LL_LPTIM_PRESCALER_DIV32
  - LL_LPTIM_PRESCALER_DIV64
  - LL_LPTIM_PRESCALER_DIV128

Return values

- **None:**

Notes

- This function must be called when the LPTIM instance is disabled.
- When the LPTIM is configured to be clocked by an internal clock source and the LPTIM counter is configured to be updated by active edges detected on the LPTIM external Input1, the internal clock provided to the LPTIM must be not be prescaled.

Reference Manual to LL API cross reference:

- CFGR PRESC LL_LPTIM_SetPrescaler

LL_LPTIM_GetPrescaler

Function name

__STATIC_INLINE uint32_t LL_LPTIM_GetPrescaler (const LPTIM_TypeDef * LPTIMx)

Function description

Get actual prescaler division ratio.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPTIM_PRESCALER_DIV1
  - LL_LPTIM_PRESCALER_DIV2
  - LL_LPTIM_PRESCALER_DIV4
  - LL_LPTIM_PRESCALER_DIV8
  - LL_LPTIM_PRESCALER_DIV16
  - LL_LPTIM_PRESCALER_DIV32
  - LL_LPTIM_PRESCALER_DIV64
  - LL_LPTIM_PRESCALER_DIV128

Reference Manual to LL API cross reference:

- CFGR PRESC LL_LPTIM_GetPrescaler

LL_LPTIM_SetInput1Src

Function name

__STATIC_INLINE void LL_LPTIM_SetInput1Src (LPTIM_TypeDef * LPTIMx, uint32_t Src)

Function description

Set LPTIM input 1 source (default GPIO).

Parameters

- **LPTIMx:** Low-Power Timer instance
- **Src:** This parameter can be one of the following values:
  - LL_LPTIM_INPUT1_SRC_GPIO

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR2 IN1SEL LL_LPTIM_SetInput1Src

LL_LPTIM_SetInput2Src

Function name

__STATIC_INLINE void LL_LPTIM_SetInput2Src (LPTIM_TypeDef * LPTIMx, uint32_t Src)

Function description

Set LPTIM input 2 source (default GPIO).

Parameters

- **LPTIMx:** Low-Power Timer instance
- **Src:** This parameter can be one of the following values:
  - LL_LPTIM_INPUT2_SRC_GPIO

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR2 IN2SEL LL_LPTIM_SetInput2Src

LL_LPTIM_SetRemap

Function name

__STATIC_INLINE void LL_LPTIM_SetRemap (LPTIM_TypeDef * LPTIMx, uint32_t Src)

Function description

Set LPTIM input source (default GPIO).

Parameters

- **LPTIMx:** Low-Power Timer instance
- **Src:** This parameter can be one of the following values:
  - LL_LPTIM_LPTIM1_IC1_RMP_GPIO
  - LL_LPTIM_LPTIM1_IC2_RMP_GPIO
  - LL_LPTIM_LPTIM1_IC2_RMP_LSI
  - LL_LPTIM_LPTIM1_IC2_RMP_LSE
  - LL_LPTIM_LPTIM2_IC1_RMP_GPIO
  - LL_LPTIM_LPTIM2_IC1_RMP_I3C1_IBIACK
  - LL_LPTIM_LPTIM2_IC2_RMP_GPIO
  - LL_LPTIM_LPTIM2_IC2_RMP_HSI_1024
  - LL_LPTIM_LPTIM2_IC2_RMP_MSI_128
  - LL_LPTIM_LPTIM2_IC2_RMP_I3C2_IBIACK
  - LL_LPTIM_LPTIM3_IC1_RMP_GPIO
  - LL_LPTIM_LPTIM3_IC1_RMP_I3C1_IBIACK
  - LL_LPTIM_LPTIM3_IC2_RMP_GPIO
  - LL_LPTIM_LPTIM3_IC2_RMP_I3C2_IBIACK

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR2 IC1SEL LL_LPTIM_SetRemap
- CFGR2 IC2SEL LL_LPTIM_SetRemap

LL_LPTIM_IC_SetPolarity

Function name

__STATIC_INLINE void LL_LPTIM_IC_SetPolarity (LPTIM_TypeDef * LPTIMx, uint32_t Channel, uint32_t Polarity)

Function description

Set the polarity of IC channel 1.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_LPTIM_CHANNEL_CH1
  - LL_LPTIM_CHANNEL_CH2
- **Polarity:** This parameter can be one of the following values:
  - LL_LPTIM_ICPOLARITY_RISING
  - LL_LPTIM_ICPOLARITY_FALLING
  - LL_LPTIM_ICPOLARITY_RISING_FALLING

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCMR1 CC1P LL_LPTIM_IC_SetPolarity
-
- CCMR1 CC2P LL_LPTIM_IC_SetPolarity
-

LL_LPTIM_IC_GetPolarity

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IC_GetPolarity (const LPTIM_TypeDef * LPTIMx, uint32_t Channel)

Function description

Get the polarity of IC channels.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_LPTIM_CHANNEL_CH1
  - LL_LPTIM_CHANNEL_CH2

Return values

- **Returned:** value can be one of the following values:
  - LL_LPTIM_ICPOLARITY_RISING
  - LL_LPTIM_ICPOLARITY_FALLING
  - LL_LPTIM_ICPOLARITY_RISING_FALLING

Reference Manual to LL API cross reference:

- CCMR1 CC1P LL_LPTIM_IC_GetPolarity
-
- CCMR1 CC2P LL_LPTIM_IC_GetPolarity
-

LL_LPTIM_IC_SetFilter

Function name

__STATIC_INLINE void LL_LPTIM_IC_SetFilter (LPTIM_TypeDef * LPTIMx, uint32_t Channel, uint32_t Filter)

Function description

Set the filter of IC channels.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_LPTIM_CHANNEL_CH1
  - LL_LPTIM_CHANNEL_CH2
- **Filter:** This parameter can be one of the following values:
  - LL_LPTIM_ICFLT_CLOCK_DIV1
  - LL_LPTIM_ICFLT_CLOCK_DIV2
  - LL_LPTIM_ICFLT_CLOCK_DIV4
  - LL_LPTIM_ICFLT_CLOCK_DIV8

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCMR1 IC1F LL_LPTIM_IC_SetFilter
-
- CCMR1 IC2F LL_LPTIM_IC_SetFilter
-

LL_LPTIM_IC_GetFilter

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IC_GetFilter (const LPTIM_TypeDef * LPTIMx, uint32_t Channel)

Function description

Get the filter of IC channels.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_LPTIM_CHANNEL_CH1
  - LL_LPTIM_CHANNEL_CH2

Return values

- **Returned:** value can be one of the following values:
  - LL_LPTIM_ICFLT_CLOCK_DIV1
  - LL_LPTIM_ICFLT_CLOCK_DIV2
  - LL_LPTIM_ICFLT_CLOCK_DIV4
  - LL_LPTIM_ICFLT_CLOCK_DIV8

Reference Manual to LL API cross reference:

- CCMR1 IC1F LL_LPTIM_IC_GetFilter
-
- CCMR1 IC2F LL_LPTIM_IC_GetFilter
-

LL_LPTIM_IC_SetPrescaler

Function name

__STATIC_INLINE void LL_LPTIM_IC_SetPrescaler (LPTIM_TypeDef * LPTIMx, uint32_t Channel, uint32_t Prescaler)

Function description

Set the prescaler of IC channels.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_LPTIM_CHANNEL_CH1
  - LL_LPTIM_CHANNEL_CH2
- **Prescaler:** This parameter can be one of the following values:
  - LL_LPTIM_ICPSC_DIV1
  - LL_LPTIM_ICPSC_DIV2
  - LL_LPTIM_ICPSC_DIV4
  - LL_LPTIM_ICPSC_DIV8

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCMR1 IC1PSC LL_LPTIM_IC_SetPrescaler
-
- CCMR1 IC2PSC LL_LPTIM_IC_SetPrescaler
-

LL_LPTIM_IC_GetPrescaler

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IC_GetPrescaler (const LPTIM_TypeDef * LPTIMx, uint32_t Channel)

Function description

Get the prescaler of IC channels.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_LPTIM_CHANNEL_CH1
  - LL_LPTIM_CHANNEL_CH2

Return values

- **Returned:** value can be one of the following values:
  - LL_LPTIM_ICPSC_DIV1
  - LL_LPTIM_ICPSC_DIV2
  - LL_LPTIM_ICPSC_DIV4
  - LL_LPTIM_ICPSC_DIV8

Reference Manual to LL API cross reference:

- CCMR1 IC1PSC LL_LPTIM_IC_GetPrescaler
-
- CCMR1 IC2PSC LL_LPTIM_IC_GetPrescaler
-

LL_LPTIM_CC_SetChannelMode

Function name

__STATIC_INLINE void LL_LPTIM_CC_SetChannelMode (LPTIM_TypeDef * LPTIMx, uint32_t Channel, uint32_t CCMode)

Function description

Set the Channel Mode.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_LPTIM_CHANNEL_CH1
  - LL_LPTIM_CHANNEL_CH2
- **CCMode:** This parameter can be one of the following values:
  - LL_LPTIM_CCMODE_OUTPUT_PWM
  - LL_LPTIM_CCMODE_INPUTCAPTURE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCMR1 CC1SEL LL_LPTIM_CC_SetChannelMode
- CCMR1 CC2SEL LL_LPTIM_CC_SetChannelMode

LL_LPTIM_CC_GetChannelMode

Function name

__STATIC_INLINE uint32_t LL_LPTIM_CC_GetChannelMode (const LPTIM_TypeDef * LPTIMx, uint32_t Channel)

Function description

Get the Channel Mode.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_LPTIM_CHANNEL_CH1
  - LL_LPTIM_CHANNEL_CH2

Return values

- **Returned:** value can be one of the following values:
  - LL_LPTIM_CCMODE_OUTPUT_PWM
  - LL_LPTIM_CCMODE_INPUTCAPTURE

Reference Manual to LL API cross reference:

- CCMR1 CC1SEL LL_LPTIM_CC_GetChannelMode
- CCMR1 CC2SEL LL_LPTIM_CC_GetChannelMode

LL_LPTIM_IC_GetCaptureCH1

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IC_GetCaptureCH1 (const LPTIM_TypeDef * LPTIMx)

Function description

Get captured value for input channel 1.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **CapturedValue:** (between Min_Data=0 and Max_Data=65535)

Notes

- The real capture value corresponding to the input capture trigger can be calculated using the formula hereafter : Real capture value = captured(LPTIM_CCRx) - offset where offset can be retrieved by calling LL_LPTIM_IC_GET_OFFSET

Reference Manual to LL API cross reference:

- CCR1 CCR1 LL_LPTIM_IC_GetCaptureCH1

LL_LPTIM_IC_GetCaptureCH2

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IC_GetCaptureCH2 (const LPTIM_TypeDef * LPTIMx)

Function description

Get captured value for input channel 2.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **CapturedValue:** (between Min_Data=0 and Max_Data=65535)

Notes

- The real capture value corresponding to the input capture trigger can be calculated using the formula hereafter : Real capture value = captured(LPTIM_CCRx) - offset where offset can be retrieved by calling LL_LPTIM_IC_GET_OFFSET

Reference Manual to LL API cross reference:

- CCR2 CCR2 LL_LPTIM_IC_GetCaptureCH2

LL_LPTIM_EnableTimeout

Function name

__STATIC_INLINE void LL_LPTIM_EnableTimeout (LPTIM_TypeDef * LPTIMx)

Function description

Enable the timeout function.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Notes

- This function must be called when the LPTIM instance is disabled.
- The first trigger event will start the timer, any successive trigger event will reset the counter and the timer will restart.
- The timeout value corresponds to the compare value; if no trigger occurs within the expected time frame, the MCU is waked-up by the compare match event.

Reference Manual to LL API cross reference:

- CFGR TIMOUT LL_LPTIM_EnableTimeout

LL_LPTIM_DisableTimeout

Function name

__STATIC_INLINE void LL_LPTIM_DisableTimeout (LPTIM_TypeDef * LPTIMx)

Function description

Disable the timeout function.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Notes

- This function must be called when the LPTIM instance is disabled.
- A trigger event arriving when the timer is already started will be ignored.

Reference Manual to LL API cross reference:

- CFGR TIMOUT LL_LPTIM_DisableTimeout

LL_LPTIM_IsEnabledTimeout

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledTimeout (const LPTIM_TypeDef * LPTIMx)

Function description

Indicate whether the timeout function is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CFGR TIMOUT LL_LPTIM_IsEnabledTimeout

LL_LPTIM_TrigSw

Function name

__STATIC_INLINE void LL_LPTIM_TrigSw (LPTIM_TypeDef * LPTIMx)

Function description

Start the LPTIM counter.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Notes

- This function must be called when the LPTIM instance is disabled.

Reference Manual to LL API cross reference:

- CFGR TRIGEN LL_LPTIM_TrigSw

LL_LPTIM_ConfigTrigger

Function name

__STATIC_INLINE void LL_LPTIM_ConfigTrigger (LPTIM_TypeDef * LPTIMx, uint32_t Source, uint32_t Filter, uint32_t Polarity)

Function description

Configure the external trigger used as a trigger event for the LPTIM.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **Source:** This parameter can be one of the following values:
  - LL_LPTIM_TRIG_SOURCE_GPIO
  - LL_LPTIM_TRIG_SOURCE_RTCALARMA
  - LL_LPTIM_TRIG_SOURCE_RTCALARMB
  - LL_LPTIM_TRIG_SOURCE_RTCTAMP1
  - LL_LPTIM_TRIG_SOURCE_RTCTAMP2
  - LL_LPTIM_TRIG_SOURCE_RTCTAMP3
  - LL_LPTIM_TRIG_SOURCE_SAI2_FS_A
  - LL_LPTIM_TRIG_SOURCE_SAI2_FS_B
  - LL_LPTIM_TRIG_SOURCE_LPTIM2_CH1
  - LL_LPTIM_TRIG_SOURCE_LPTIM3_CH1
  - LL_LPTIM_TRIG_SOURCE_LPTIM4_OUT
  - LL_LPTIM_TRIG_SOURCE_LPTIM5_OUT
- **Filter:** This parameter can be one of the following values:
  - LL_LPTIM_TRIG_FILTER_NONE
  - LL_LPTIM_TRIG_FILTER_2
  - LL_LPTIM_TRIG_FILTER_4
  - LL_LPTIM_TRIG_FILTER_8
- **Polarity:** This parameter can be one of the following values:
  - LL_LPTIM_TRIG_POLARITY_RISING
  - LL_LPTIM_TRIG_POLARITY_FALLING
  - LL_LPTIM_TRIG_POLARITY_RISING_FALLING

Return values

- **None:**

Notes

- This function must be called when the LPTIM instance is disabled.
- An internal clock source must be present when a digital filter is required for the trigger.

Reference Manual to LL API cross reference:

- CFGR TRIGSEL LL_LPTIM_ConfigTrigger
- CFGR TRGFLT LL_LPTIM_ConfigTrigger
- CFGR TRIGEN LL_LPTIM_ConfigTrigger

LL_LPTIM_GetTriggerSource

Function name

__STATIC_INLINE uint32_t LL_LPTIM_GetTriggerSource (const LPTIM_TypeDef * LPTIMx)

Function description

Get actual external trigger source.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPTIM_TRIG_SOURCE_GPIO
  - LL_LPTIM_TRIG_SOURCE_RTCALARMA
  - LL_LPTIM_TRIG_SOURCE_RTCALARMB
  - LL_LPTIM_TRIG_SOURCE_RTCTAMP1
  - LL_LPTIM_TRIG_SOURCE_RTCTAMP2
  - LL_LPTIM_TRIG_SOURCE_RTCTAMP3
  - LL_LPTIM_TRIG_SOURCE_SAI2_FS_A
  - LL_LPTIM_TRIG_SOURCE_SAI2_FS_B
  - LL_LPTIM_TRIG_SOURCE_LPTIM2_CH1
  - LL_LPTIM_TRIG_SOURCE_LPTIM3_CH1
  - LL_LPTIM_TRIG_SOURCE_LPTIM4_OUT
  - LL_LPTIM_TRIG_SOURCE_LPTIM5_OUT

Reference Manual to LL API cross reference:

- CFGR TRIGSEL LL_LPTIM_GetTriggerSource

LL_LPTIM_GetTriggerFilter

Function name

__STATIC_INLINE uint32_t LL_LPTIM_GetTriggerFilter (const LPTIM_TypeDef * LPTIMx)

Function description

Get actual external trigger filter.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPTIM_TRIG_FILTER_NONE
  - LL_LPTIM_TRIG_FILTER_2
  - LL_LPTIM_TRIG_FILTER_4
  - LL_LPTIM_TRIG_FILTER_8

Reference Manual to LL API cross reference:

- CFGR TRGFLT LL_LPTIM_GetTriggerFilter

LL_LPTIM_GetTriggerPolarity

Function name

__STATIC_INLINE uint32_t LL_LPTIM_GetTriggerPolarity (const LPTIM_TypeDef * LPTIMx)

Function description

Get actual external trigger polarity.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPTIM_TRIG_POLARITY_RISING
  - LL_LPTIM_TRIG_POLARITY_FALLING
  - LL_LPTIM_TRIG_POLARITY_RISING_FALLING

Reference Manual to LL API cross reference:

- CFGR TRIGEN LL_LPTIM_GetTriggerPolarity

LL_LPTIM_SetClockSource

Function name

__STATIC_INLINE void LL_LPTIM_SetClockSource (LPTIM_TypeDef * LPTIMx, uint32_t ClockSource)

Function description

Set the source of the clock used by the LPTIM instance.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **ClockSource:** This parameter can be one of the following values:
  - LL_LPTIM_CLK_SOURCE_INTERNAL
  - LL_LPTIM_CLK_SOURCE_EXTERNAL

Return values

- **None:**

Notes

- This function must be called when the LPTIM instance is disabled.

Reference Manual to LL API cross reference:

- CFGR CKSEL LL_LPTIM_SetClockSource

LL_LPTIM_GetClockSource

Function name

__STATIC_INLINE uint32_t LL_LPTIM_GetClockSource (const LPTIM_TypeDef * LPTIMx)

Function description

Get actual LPTIM instance clock source.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPTIM_CLK_SOURCE_INTERNAL
  - LL_LPTIM_CLK_SOURCE_EXTERNAL

Reference Manual to LL API cross reference:

- CFGR CKSEL LL_LPTIM_GetClockSource

LL_LPTIM_ConfigClock

Function name

__STATIC_INLINE void LL_LPTIM_ConfigClock (LPTIM_TypeDef * LPTIMx, uint32_t ClockFilter, uint32_t ClockPolarity)

Function description

Configure the active edge or edges used by the counter when the LPTIM is clocked by an external clock source.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **ClockFilter:** This parameter can be one of the following values:
  - LL_LPTIM_CLK_FILTER_NONE
  - LL_LPTIM_CLK_FILTER_2
  - LL_LPTIM_CLK_FILTER_4
  - LL_LPTIM_CLK_FILTER_8
- **ClockPolarity:** This parameter can be one of the following values:
  - LL_LPTIM_CLK_POLARITY_RISING
  - LL_LPTIM_CLK_POLARITY_FALLING
  - LL_LPTIM_CLK_POLARITY_RISING_FALLING

Return values

- **None:**

Notes

- This function must be called when the LPTIM instance is disabled.
- When both external clock signal edges are considered active ones, the LPTIM must also be clocked by an internal clock source with a frequency equal to at least four times the external clock frequency.
- An internal clock source must be present when a digital filter is required for external clock.

Reference Manual to LL API cross reference:

- CFGR CKFLT LL_LPTIM_ConfigClock
- CFGR CKPOL LL_LPTIM_ConfigClock

LL_LPTIM_GetClockPolarity

Function name

__STATIC_INLINE uint32_t LL_LPTIM_GetClockPolarity (const LPTIM_TypeDef * LPTIMx)

Function description

Get actual clock polarity.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPTIM_CLK_POLARITY_RISING
  - LL_LPTIM_CLK_POLARITY_FALLING
  - LL_LPTIM_CLK_POLARITY_RISING_FALLING

Reference Manual to LL API cross reference:

- CFGR CKPOL LL_LPTIM_GetClockPolarity

LL_LPTIM_GetClockFilter

Function name

__STATIC_INLINE uint32_t LL_LPTIM_GetClockFilter (const LPTIM_TypeDef * LPTIMx)

Function description

Get actual clock digital filter.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPTIM_CLK_FILTER_NONE
  - LL_LPTIM_CLK_FILTER_2
  - LL_LPTIM_CLK_FILTER_4
  - LL_LPTIM_CLK_FILTER_8

Reference Manual to LL API cross reference:

- CFGR CKFLT LL_LPTIM_GetClockFilter

LL_LPTIM_SetEncoderMode

Function name

__STATIC_INLINE void LL_LPTIM_SetEncoderMode (LPTIM_TypeDef * LPTIMx, uint32_t EncoderMode)

Function description

Configure the encoder mode.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **EncoderMode:** This parameter can be one of the following values:
  - LL_LPTIM_ENCODER_MODE_RISING
  - LL_LPTIM_ENCODER_MODE_FALLING
  - LL_LPTIM_ENCODER_MODE_RISING_FALLING

Return values

- **None:**

Notes

- This function must be called when the LPTIM instance is disabled.

Reference Manual to LL API cross reference:

- CFGR CKPOL LL_LPTIM_SetEncoderMode

LL_LPTIM_GetEncoderMode

Function name

__STATIC_INLINE uint32_t LL_LPTIM_GetEncoderMode (const LPTIM_TypeDef * LPTIMx)

Function description

Get actual encoder mode.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPTIM_ENCODER_MODE_RISING
  - LL_LPTIM_ENCODER_MODE_FALLING
  - LL_LPTIM_ENCODER_MODE_RISING_FALLING

Reference Manual to LL API cross reference:

- CFGR CKPOL LL_LPTIM_GetEncoderMode

LL_LPTIM_EnableEncoderMode

Function name

__STATIC_INLINE void LL_LPTIM_EnableEncoderMode (LPTIM_TypeDef * LPTIMx)

Function description

Enable the encoder mode.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Notes

- This function must be called when the LPTIM instance is disabled.
- In this mode the LPTIM instance must be clocked by an internal clock source. Also, the prescaler division ratio must be equal to 1.
- LPTIM instance must be configured in continuous mode prior enabling the encoder mode.

Reference Manual to LL API cross reference:

- CFGR ENC LL_LPTIM_EnableEncoderMode

LL_LPTIM_DisableEncoderMode

Function name

__STATIC_INLINE void LL_LPTIM_DisableEncoderMode (LPTIM_TypeDef * LPTIMx)

Function description

Disable the encoder mode.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Notes

- This function must be called when the LPTIM instance is disabled.

Reference Manual to LL API cross reference:

- CFGR ENC LL_LPTIM_DisableEncoderMode

LL_LPTIM_IsEnabledEncoderMode

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledEncoderMode (const LPTIM_TypeDef * LPTIMx)

Function description

Indicates whether the LPTIM operates in encoder mode.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CFGR ENC LL_LPTIM_IsEnabledEncoderMode

LL_LPTIM_ClearFlag_CC1

Function name

__STATIC_INLINE void LL_LPTIM_ClearFlag_CC1 (LPTIM_TypeDef * LPTIMx)

Function description

Clear the compare match flag for channel 1 (CC1CF)

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR CC1CF LL_LPTIM_ClearFlag_CC1

LL_LPTIM_IsActiveFlag_CC1

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsActiveFlag_CC1 (const LPTIM_TypeDef * LPTIMx)

Function description

Inform application whether a capture/compare interrupt has occurred for channel 1.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR CC1IF LL_LPTIM_IsActiveFlag_CC1

LL_LPTIM_ClearFlag_CC2

Function name

__STATIC_INLINE void LL_LPTIM_ClearFlag_CC2 (LPTIM_TypeDef * LPTIMx)

Function description

Clear the compare match flag for channel 2 (CC2CF)

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR CC2CF LL_LPTIM_ClearFlag_CC2

LL_LPTIM_IsActiveFlag_CC2

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsActiveFlag_CC2 (const LPTIM_TypeDef * LPTIMx)

Function description

Inform application whether a capture/compare interrupt has occurred for channel 2.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR CC2IF LL_LPTIM_IsActiveFlag_CC2

LL_LPTIM_ClearFlag_CC1O

Function name

__STATIC_INLINE void LL_LPTIM_ClearFlag_CC1O (LPTIM_TypeDef * LPTIMx)

Function description

Clear the Capture/Compare 1 over-capture flag for channel 1 (CC1OCF)

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR CC1OCF LL_LPTIM_ClearFlag_CC1O

LL_LPTIM_IsActiveFlag_CC1O

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsActiveFlag_CC1O (const LPTIM_TypeDef * LPTIMx)

Function description

Inform application whether a Capture/Compare 1 over-capture has occurred for channel 1.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR CC1OF LL_LPTIM_IsActiveFlag_CC1O

LL_LPTIM_ClearFlag_CC2O

Function name

__STATIC_INLINE void LL_LPTIM_ClearFlag_CC2O (LPTIM_TypeDef * LPTIMx)

Function description

Clear the Capture/Compare 2 over-capture flag for channel 2 (CC2OCF)

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR CC2OCF LL_LPTIM_ClearFlag_CC2O

LL_LPTIM_IsActiveFlag_CC2O

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsActiveFlag_CC2O (const LPTIM_TypeDef * LPTIMx)

Function description

Inform application whether a Capture/Compare 2 over-capture has occurred for channel 2.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR CC2OF LL_LPTIM_IsActiveFlag_CC2O

LL_LPTIM_ClearFlag_ARRM

Function name

__STATIC_INLINE void LL_LPTIM_ClearFlag_ARRM (LPTIM_TypeDef * LPTIMx)

Function description

Clear the autoreload match flag (ARRMCF)

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR ARRMCF LL_LPTIM_ClearFlag_ARRM

LL_LPTIM_IsActiveFlag_ARRM

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsActiveFlag_ARRM (const LPTIM_TypeDef * LPTIMx)

Function description

Inform application whether a autoreload match interrupt has occurred.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR ARRM LL_LPTIM_IsActiveFlag_ARRM

LL_LPTIM_ClearFlag_EXTTRIG

Function name

__STATIC_INLINE void LL_LPTIM_ClearFlag_EXTTRIG (LPTIM_TypeDef * LPTIMx)

Function description

Clear the external trigger valid edge flag(EXTTRIGCF).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR EXTTRIGCF LL_LPTIM_ClearFlag_EXTTRIG

LL_LPTIM_IsActiveFlag_EXTTRIG

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsActiveFlag_EXTTRIG (const LPTIM_TypeDef * LPTIMx)

Function description

Inform application whether a valid edge on the selected external trigger input has occurred.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR EXTTRIG LL_LPTIM_IsActiveFlag_EXTTRIG

LL_LPTIM_ClearFlag_CMP1OK

Function name

__STATIC_INLINE void LL_LPTIM_ClearFlag_CMP1OK (LPTIM_TypeDef * LPTIMx)

Function description

Clear the compare register update interrupt flag (CMP1OKCF).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR CMP1OKCF LL_LPTIM_ClearFlag_CMP1OK

LL_LPTIM_IsActiveFlag_CMP1OK

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsActiveFlag_CMP1OK (const LPTIM_TypeDef * LPTIMx)

Function description

Informs application whether the APB bus write operation to the LPTIMx_CCR1 register has been successfully completed.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR CMP1OK LL_LPTIM_IsActiveFlag_CMP1OK

LL_LPTIM_ClearFlag_CMP2OK

Function name

__STATIC_INLINE void LL_LPTIM_ClearFlag_CMP2OK (LPTIM_TypeDef * LPTIMx)

Function description

Clear the compare register update interrupt flag (CMP2OKCF).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR CMP2OKCF LL_LPTIM_ClearFlag_CMP2OK

LL_LPTIM_IsActiveFlag_CMP2OK

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsActiveFlag_CMP2OK (const LPTIM_TypeDef * LPTIMx)

Function description

Informs application whether the APB bus write operation to the LPTIMx_CCR2 register has been successfully completed.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR CMP2OK LL_LPTIM_IsActiveFlag_CMP2OK

LL_LPTIM_ClearFlag_DIEROK

Function name

__STATIC_INLINE void LL_LPTIM_ClearFlag_DIEROK (LPTIM_TypeDef * LPTIMx)

Function description

Clear the interrupt register update interrupt flag (DIEROKCF).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR DIEROKCF LL_LPTIM_ClearFlag_DIEROK

LL_LPTIM_IsActiveFlag_DIEROK

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsActiveFlag_DIEROK (const LPTIM_TypeDef * LPTIMx)

Function description

Informs application whether the APB bus write operation to the LPTIMx_DIER register has been successfully completed.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR DIEROK LL_LPTIM_IsActiveFlag_DIEROK

LL_LPTIM_ClearFlag_ARROK

Function name

__STATIC_INLINE void LL_LPTIM_ClearFlag_ARROK (LPTIM_TypeDef * LPTIMx)

Function description

Clear the autoreload register update interrupt flag (ARROKCF).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR ARROKCF LL_LPTIM_ClearFlag_ARROK

LL_LPTIM_IsActiveFlag_ARROK

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsActiveFlag_ARROK (const LPTIM_TypeDef * LPTIMx)

Function description

Informs application whether the APB bus write operation to the LPTIMx_ARR register has been successfully completed.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR ARROK LL_LPTIM_IsActiveFlag_ARROK

LL_LPTIM_ClearFlag_UP

Function name

__STATIC_INLINE void LL_LPTIM_ClearFlag_UP (LPTIM_TypeDef * LPTIMx)

Function description

Clear the counter direction change to up interrupt flag (UPCF).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR UPCF LL_LPTIM_ClearFlag_UP

LL_LPTIM_IsActiveFlag_UP

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsActiveFlag_UP (const LPTIM_TypeDef * LPTIMx)

Function description

Informs the application whether the counter direction has changed from down to up (when the LPTIM instance operates in encoder mode).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR UP LL_LPTIM_IsActiveFlag_UP

LL_LPTIM_ClearFlag_DOWN

Function name

__STATIC_INLINE void LL_LPTIM_ClearFlag_DOWN (LPTIM_TypeDef * LPTIMx)

Function description

Clear the counter direction change to down interrupt flag (DOWNCF).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR DOWNCF LL_LPTIM_ClearFlag_DOWN

LL_LPTIM_IsActiveFlag_DOWN

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsActiveFlag_DOWN (const LPTIM_TypeDef * LPTIMx)

Function description

Informs the application whether the counter direction has changed from up to down (when the LPTIM instance operates in encoder mode).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR DOWN LL_LPTIM_IsActiveFlag_DOWN

LL_LPTIM_ClearFlag_REPOK

Function name

__STATIC_INLINE void LL_LPTIM_ClearFlag_REPOK (LPTIM_TypeDef * LPTIMx)

Function description

Clear the repetition register update interrupt flag (REPOKCF).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR REPOKCF LL_LPTIM_ClearFlag_REPOK

LL_LPTIM_IsActiveFlag_REPOK

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsActiveFlag_REPOK (const LPTIM_TypeDef * LPTIMx)

Function description

Informs application whether the APB bus write operation to the LPTIMx_RCR register has been successfully completed; If so, a new one can be initiated.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR REPOK LL_LPTIM_IsActiveFlag_REPOK

LL_LPTIM_ClearFlag_UE

Function name

__STATIC_INLINE void LL_LPTIM_ClearFlag_UE (LPTIM_TypeDef * LPTIMx)

Function description

Clear the update event flag (UECF).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR UECF LL_LPTIM_ClearFlag_UE

LL_LPTIM_IsActiveFlag_UE

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsActiveFlag_UE (const LPTIM_TypeDef * LPTIMx)

Function description

Informs application whether the LPTIMx update event has occurred.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR UE LL_LPTIM_IsActiveFlag_UE

LL_LPTIM_EnableIT_CC1

Function name

__STATIC_INLINE void LL_LPTIM_EnableIT_CC1 (LPTIM_TypeDef * LPTIMx)

Function description

Enable capture/compare 1 interrupt (CC1IE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC1IE LL_LPTIM_EnableIT_CC1

LL_LPTIM_DisableIT_CC1

Function name

__STATIC_INLINE void LL_LPTIM_DisableIT_CC1 (LPTIM_TypeDef * LPTIMx)

Function description

Disable capture/compare 1 interrupt (CC1IE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC1IE LL_LPTIM_DisableIT_CC1

LL_LPTIM_IsEnabledIT_CC1

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_CC1 (const LPTIM_TypeDef * LPTIMx)

Function description

Indicates whether the capture/compare 1 interrupt (CC1IE) is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER CC1IE LL_LPTIM_IsEnabledIT_CC1

LL_LPTIM_EnableIT_CC2

Function name

__STATIC_INLINE void LL_LPTIM_EnableIT_CC2 (LPTIM_TypeDef * LPTIMx)

Function description

Enable capture/compare 1 interrupt (CC2IE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC2IE LL_LPTIM_EnableIT_CC2

LL_LPTIM_DisableIT_CC2

Function name

__STATIC_INLINE void LL_LPTIM_DisableIT_CC2 (LPTIM_TypeDef * LPTIMx)

Function description

Disable capture/compare 2 interrupt (CC2IE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC2IE LL_LPTIM_DisableIT_CC2

LL_LPTIM_IsEnabledIT_CC2

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_CC2 (const LPTIM_TypeDef * LPTIMx)

Function description

Indicates whether the capture/compare 2 interrupt (CC2IE) is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER CC2IE LL_LPTIM_IsEnabledIT_CC2

LL_LPTIM_EnableIT_CC1O

Function name

__STATIC_INLINE void LL_LPTIM_EnableIT_CC1O (LPTIM_TypeDef * LPTIMx)

Function description

Enable capture/compare 1 over-capture interrupt (CC1OIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC1OIE LL_LPTIM_EnableIT_CC1O

LL_LPTIM_DisableIT_CC1O

Function name

__STATIC_INLINE void LL_LPTIM_DisableIT_CC1O (LPTIM_TypeDef * LPTIMx)

Function description

Disable capture/compare 1 over-capture interrupt (CC1OIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC1OIE LL_LPTIM_DisableIT_CC1O

LL_LPTIM_IsEnabledIT_CC1O

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_CC1O (const LPTIM_TypeDef * LPTIMx)

Function description

Indicates whether the capture/compare 1 over-capture interrupt (CC1OIE) is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER CC1OIE LL_LPTIM_IsEnabledIT_CC1O

LL_LPTIM_EnableIT_CC2O

Function name

__STATIC_INLINE void LL_LPTIM_EnableIT_CC2O (LPTIM_TypeDef * LPTIMx)

Function description

Enable capture/compare 1 over-capture interrupt (CC2OIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC2OIE LL_LPTIM_EnableIT_CC2O

LL_LPTIM_DisableIT_CC2O

Function name

__STATIC_INLINE void LL_LPTIM_DisableIT_CC2O (LPTIM_TypeDef * LPTIMx)

Function description

Disable capture/compare 1 over-capture interrupt (CC2OIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC2OIE LL_LPTIM_DisableIT_CC2O

LL_LPTIM_IsEnabledIT_CC2O

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_CC2O (const LPTIM_TypeDef * LPTIMx)

Function description

Indicates whether the capture/compare 2 over-capture interrupt (CC2OIE) is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER CC2OIE LL_LPTIM_IsEnabledIT_CC2O

LL_LPTIM_EnableIT_ARRM

Function name

__STATIC_INLINE void LL_LPTIM_EnableIT_ARRM (LPTIM_TypeDef * LPTIMx)

Function description

Enable autoreload match interrupt (ARRMIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER ARRMIE LL_LPTIM_EnableIT_ARRM

LL_LPTIM_DisableIT_ARRM

Function name

__STATIC_INLINE void LL_LPTIM_DisableIT_ARRM (LPTIM_TypeDef * LPTIMx)

Function description

Disable autoreload match interrupt (ARRMIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER ARRMIE LL_LPTIM_DisableIT_ARRM

LL_LPTIM_IsEnabledIT_ARRM

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_ARRM (const LPTIM_TypeDef * LPTIMx)

Function description

Indicates whether the autoreload match interrupt (ARRMIE) is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER ARRMIE LL_LPTIM_IsEnabledIT_ARRM

LL_LPTIM_EnableIT_EXTTRIG

Function name

__STATIC_INLINE void LL_LPTIM_EnableIT_EXTTRIG (LPTIM_TypeDef * LPTIMx)

Function description

Enable external trigger valid edge interrupt (EXTTRIGIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER EXTTRIGIE LL_LPTIM_EnableIT_EXTTRIG

LL_LPTIM_DisableIT_EXTTRIG

Function name

__STATIC_INLINE void LL_LPTIM_DisableIT_EXTTRIG (LPTIM_TypeDef * LPTIMx)

Function description

Disable external trigger valid edge interrupt (EXTTRIGIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER EXTTRIGIE LL_LPTIM_DisableIT_EXTTRIG

LL_LPTIM_IsEnabledIT_EXTTRIG

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_EXTTRIG (const LPTIM_TypeDef * LPTIMx)

Function description

Indicates external trigger valid edge interrupt (EXTTRIGIE) is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER EXTTRIGIE LL_LPTIM_IsEnabledIT_EXTTRIG

LL_LPTIM_EnableIT_CMP1OK

Function name

__STATIC_INLINE void LL_LPTIM_EnableIT_CMP1OK (LPTIM_TypeDef * LPTIMx)

Function description

Enable compare register write completed interrupt (CMP1OKIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER CMP1OKIE LL_LPTIM_EnableIT_CMP1OK

LL_LPTIM_DisableIT_CMP1OK

Function name

__STATIC_INLINE void LL_LPTIM_DisableIT_CMP1OK (LPTIM_TypeDef * LPTIMx)

Function description

Disable compare register write completed interrupt (CMP1OKIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER CMPO1KIE LL_LPTIM_DisableIT_CMP1OK

LL_LPTIM_IsEnabledIT_CMP1OK

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_CMP1OK (const LPTIM_TypeDef * LPTIMx)

Function description

Indicates whether the compare register write completed interrupt (CMP1OKIE) is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER CMP1OKIE LL_LPTIM_IsEnabledIT_CMP1OK

LL_LPTIM_EnableIT_CMP2OK

Function name

__STATIC_INLINE void LL_LPTIM_EnableIT_CMP2OK (LPTIM_TypeDef * LPTIMx)

Function description

Enable compare register write completed interrupt (CMP2OKIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER CMP2OKIE LL_LPTIM_EnableIT_CMP2OK

LL_LPTIM_DisableIT_CMP2OK

Function name

__STATIC_INLINE void LL_LPTIM_DisableIT_CMP2OK (LPTIM_TypeDef * LPTIMx)

Function description

Disable compare register write completed interrupt (CMP2OKIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER CMP2OKIE LL_LPTIM_DisableIT_CMP2OK

LL_LPTIM_IsEnabledIT_CMP2OK

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_CMP2OK (const LPTIM_TypeDef * LPTIMx)

Function description

Indicates whether the compare register write completed interrupt (CMP2OKIE) is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER CMP2OKIE LL_LPTIM_IsEnabledIT_CMP2OK

LL_LPTIM_EnableIT_ARROK

Function name

__STATIC_INLINE void LL_LPTIM_EnableIT_ARROK (LPTIM_TypeDef * LPTIMx)

Function description

Enable autoreload register write completed interrupt (ARROKIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER ARROKIE LL_LPTIM_EnableIT_ARROK

LL_LPTIM_DisableIT_ARROK

Function name

__STATIC_INLINE void LL_LPTIM_DisableIT_ARROK (LPTIM_TypeDef * LPTIMx)

Function description

Disable autoreload register write completed interrupt (ARROKIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER ARROKIE LL_LPTIM_DisableIT_ARROK

LL_LPTIM_IsEnabledIT_ARROK

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_ARROK (const LPTIM_TypeDef * LPTIMx)

Function description

Indicates whether the autoreload register write completed interrupt (ARROKIE) is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit(1 or 0).

Reference Manual to LL API cross reference:

- DIER ARROKIE LL_LPTIM_IsEnabledIT_ARROK

LL_LPTIM_EnableIT_UP

Function name

__STATIC_INLINE void LL_LPTIM_EnableIT_UP (LPTIM_TypeDef * LPTIMx)

Function description

Enable direction change to up interrupt (UPIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER UPIE LL_LPTIM_EnableIT_UP

LL_LPTIM_DisableIT_UP

Function name

__STATIC_INLINE void LL_LPTIM_DisableIT_UP (LPTIM_TypeDef * LPTIMx)

Function description

Disable direction change to up interrupt (UPIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER UPIE LL_LPTIM_DisableIT_UP

LL_LPTIM_IsEnabledIT_UP

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_UP (const LPTIM_TypeDef * LPTIMx)

Function description

Indicates whether the direction change to up interrupt (UPIE) is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit(1 or 0).

Reference Manual to LL API cross reference:

- DIER UPIE LL_LPTIM_IsEnabledIT_UP

LL_LPTIM_EnableIT_DOWN

Function name

__STATIC_INLINE void LL_LPTIM_EnableIT_DOWN (LPTIM_TypeDef * LPTIMx)

Function description

Enable direction change to down interrupt (DOWNIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER DOWNIE LL_LPTIM_EnableIT_DOWN

LL_LPTIM_DisableIT_DOWN

Function name

__STATIC_INLINE void LL_LPTIM_DisableIT_DOWN (LPTIM_TypeDef * LPTIMx)

Function description

Disable direction change to down interrupt (DOWNIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER DOWNIE LL_LPTIM_DisableIT_DOWN

LL_LPTIM_IsEnabledIT_DOWN

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_DOWN (const LPTIM_TypeDef * LPTIMx)

Function description

Indicates whether the direction change to down interrupt (DOWNIE) is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit(1 or 0).

Reference Manual to LL API cross reference:

- DIER DOWNIE LL_LPTIM_IsEnabledIT_DOWN

LL_LPTIM_EnableIT_REPOK

Function name

__STATIC_INLINE void LL_LPTIM_EnableIT_REPOK (LPTIM_TypeDef * LPTIMx)

Function description

Enable repetition register update successfully completed interrupt (REPOKIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER REPOKIE LL_LPTIM_EnableIT_REPOK

LL_LPTIM_DisableIT_REPOK

Function name

__STATIC_INLINE void LL_LPTIM_DisableIT_REPOK (LPTIM_TypeDef * LPTIMx)

Function description

Disable repetition register update successfully completed interrupt (REPOKIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER REPOKIE LL_LPTIM_DisableIT_REPOK

LL_LPTIM_IsEnabledIT_REPOK

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_REPOK (const LPTIM_TypeDef * LPTIMx)

Function description

Indicates whether the repetition register update successfully completed interrupt (REPOKIE) is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit(1 or 0).

Reference Manual to LL API cross reference:

- DIER REPOKIE LL_LPTIM_IsEnabledIT_REPOK

LL_LPTIM_EnableIT_UE

Function name

__STATIC_INLINE void LL_LPTIM_EnableIT_UE (LPTIM_TypeDef * LPTIMx)

Function description

Enable update event interrupt (UEIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER UEIE LL_LPTIM_EnableIT_UE

LL_LPTIM_DisableIT_UE

Function name

__STATIC_INLINE void LL_LPTIM_DisableIT_UE (LPTIM_TypeDef * LPTIMx)

Function description

Disable update event interrupt (UEIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER UEIE LL_LPTIM_DisableIT_UE

LL_LPTIM_IsEnabledIT_UE

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_UE (const LPTIM_TypeDef * LPTIMx)

Function description

Indicates whether the update event interrupt (UEIE) is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance @ retval State of bit(1 or 0).

Reference Manual to LL API cross reference:

- DIER UEIE LL_LPTIM_IsEnabledIT_UE

LL_LPTIM_EnableDMAReq_UPDATE

Function name

__STATIC_INLINE void LL_LPTIM_EnableDMAReq_UPDATE (LPTIM_TypeDef * LPTIMx)

Function description

Enable update DMA request.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER UEDE LL_LPTIM_EnableDMAReq_UPDATE

LL_LPTIM_DisableDMAReq_UPDATE

Function name

__STATIC_INLINE void LL_LPTIM_DisableDMAReq_UPDATE (LPTIM_TypeDef * LPTIMx)

Function description

Disable update DMA request.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER UEDE LL_LPTIM_DisableDMAReq_UPDATE

LL_LPTIM_IsEnabledDMAReq_UPDATE

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledDMAReq_UPDATE (const LPTIM_TypeDef * LPTIMx)

Function description

Indicates whether the update DMA request is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER UEDE LL_LPTIM_IsEnabledDMAReq_UPDATE

LL_LPTIM_EnableDMAReq_CC1

Function name

__STATIC_INLINE void LL_LPTIM_EnableDMAReq_CC1 (LPTIM_TypeDef * LPTIMx)

Function description

Enable capture/compare 1 DMA request (CC1DE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC1DE LL_LPTIM_EnableDMAReq_CC1

LL_LPTIM_DisableDMAReq_CC1

Function name

__STATIC_INLINE void LL_LPTIM_DisableDMAReq_CC1 (LPTIM_TypeDef * LPTIMx)

Function description

Disable capture/compare 1 DMA request (CC1DE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC1DE LL_LPTIM_DisableDMAReq_CC1

LL_LPTIM_IsEnabledDMAReq_CC1

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledDMAReq_CC1 (const LPTIM_TypeDef * LPTIMx)

Function description

Indicates whether the capture/compare 1 DMA request (CC1DE) is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER CC1DE LL_LPTIM_IsEnabledDMAReq_CC1

LL_LPTIM_EnableDMAReq_CC2

Function name

__STATIC_INLINE void LL_LPTIM_EnableDMAReq_CC2 (LPTIM_TypeDef * LPTIMx)

Function description

Enable capture/compare 2 DMA request (CC2DE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC2DE LL_LPTIM_EnableDMAReq_CC2

LL_LPTIM_DisableDMAReq_CC2

Function name

__STATIC_INLINE void LL_LPTIM_DisableDMAReq_CC2 (LPTIM_TypeDef * LPTIMx)

Function description

Disable capture/compare 2 DMA request (CC2DE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC2DE LL_LPTIM_DisableDMAReq_CC2

LL_LPTIM_IsEnabledDMAReq_CC2

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledDMAReq_CC2 (const LPTIM_TypeDef * LPTIMx)

Function description

Indicates whether the capture/compare 2 DMA request (CC2DE) is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER CC2DE LL_LPTIM_IsEnabledDMAReq_CC2

98.3 LPTIM Firmware driver defines

The following section lists the various define and macros of the module.

98.3.1 LPTIM

LPTIM

Input1 Source

LL_LPTIM_INPUT1_SRC_GPIO

For LPTIM1, LPTIM2 and LPTIM3

Input2 Source

LL_LPTIM_INPUT2_SRC_GPIO

For LPTIM1, LPTIM2 and LPTIM3

LPTIM1 Input Ch1 Remap

LL_LPTIM_LPTIM1_IC1_RMP_GPIO

IC1 connected to GPIO

LPTIM1 Input Ch2 Remap

LL_LPTIM_LPTIM1_IC2_RMP_GPIO

IC2 connected to GPIO

LL_LPTIM_LPTIM1_IC2_RMP_LSI

IC2 connected to LSI

LL_LPTIM_LPTIM1_IC2_RMP_LSE

IC2 connected to LSE

LPTIM2 Input Ch1 Remap

LL_LPTIM_LPTIM2_IC1_RMP_GPIO

IC1 connected to GPIO

LL_LPTIM_LPTIM2_IC1_RMP_I3C1_IBIACK

IC1 connected to I3C1 IBI ACK

LPTIM2 Input Ch2 Remap

LL_LPTIM_LPTIM2_IC2_RMP_GPIO

IC2 connected to GPIO

LL_LPTIM_LPTIM2_IC2_RMP_HSI_1024

IC2 connected to HSI/1024

LL_LPTIM_LPTIM2_IC2_RMP_MSI_128

IC2 connected to MSI/128

LL_LPTIM_LPTIM2_IC2_RMP_I3C2_IBIACK

IC2 connected to I3C2 IBI ACK

LPTIM3 Input Ch1 Remap

LL_LPTIM_LPTIM3_IC1_RMP_GPIO

IC1 connected to GPIO

LL_LPTIM_LPTIM3_IC1_RMP_I3C1_IBIACK

IC1 connected to I3C1 IBI ACK

LPTIM3 Input Ch2 Remap

LL_LPTIM_LPTIM3_IC2_RMP_GPIO

IC2 connected to GPIO

LL_LPTIM_LPTIM3_IC2_RMP_I3C2_IBIACK

IC2 connected to I3C2 IBI ACK

Clock Filter

LL_LPTIM_CLK_FILTER_NONE

Any external clock signal level change is considered as a valid transition

LL_LPTIM_CLK_FILTER_2

External clock signal level change must be stable for at least 2 clock periods before it is considered as valid transition

LL_LPTIM_CLK_FILTER_4

External clock signal level change must be stable for at least 4 clock periods before it is considered as valid transition

LL_LPTIM_CLK_FILTER_8

External clock signal level change must be stable for at least 8 clock periods before it is considered as valid transition

Clock Polarity

LL_LPTIM_CLK_POLARITY_RISING

The rising edge is the active edge used for counting

LL_LPTIM_CLK_POLARITY_FALLING

The falling edge is the active edge used for counting

LL_LPTIM_CLK_POLARITY_RISING_FALLING

Both edges are active edges

Clock Source

LL_LPTIM_CLK_SOURCE_INTERNAL

LPTIM is clocked by internal clock source (APB clock or any of the embedded oscillators)

LL_LPTIM_CLK_SOURCE_EXTERNAL

LPTIM is clocked by an external clock source through the LPTIM external Input1

Counter Mode

LL_LPTIM_COUNTER_MODE_INTERNAL

The counter is incremented following each internal clock pulse

LL_LPTIM_COUNTER_MODE_EXTERNAL

The counter is incremented following each valid clock pulse on the LPTIM external Input1

Encoder Mode

LL_LPTIM_ENCODER_MODE_RISING

The rising edge is the active edge used for counting

LL_LPTIM_ENCODER_MODE_FALLING

The falling edge is the active edge used for counting

LL_LPTIM_ENCODER_MODE_RISING_FALLING

Both edges are active edges

Get Flags Defines

LL_LPTIM_ISR_CMP1OK

Compare register 1 update OK

LL_LPTIM_ISR_CMP2OK

Compare register 2 update OK

LL_LPTIM_ISR_CC1IF

Capture/Compare 1 interrupt flag

LL_LPTIM_ISR_CC2IF

Capture/Compare 2 interrupt flag

LL_LPTIM_ISR_CC1OF

Capture/Compare 1 over-capture flag

LL_LPTIM_ISR_CC2OF

Capture/Compare 2 over-capture flag

LL_LPTIM_ISR_DIEROK

Interrupt enable register update OK

LL_LPTIM_ISR_ARRM

Autoreload match

LL_LPTIM_ISR_EXTTRIG

External trigger edge event

LL_LPTIM_ISR_ARROK

Autoreload register update OK

LL_LPTIM_ISR_UP

Counter direction change down to up

LL_LPTIM_ISR_DOWN

Counter direction change up to down

LL_LPTIM_ISR_UE

Update event

LL_LPTIM_ISR_REPOK

Repetition register update OK

IT Defines

LL_LPTIM_DIER_CMP1OKIE

Compare register 1 update OK

LL_LPTIM_DIER_CMP2OKIE

Compare register 2 update OK

LL_LPTIM_DIER_CC1IFIE

Capture/Compare 1 interrupt flag

LL_LPTIM_DIER_CC2IFIE

Capture/Compare 2 interrupt flag

LL_LPTIM_DIER_CC1OFIE

Capture/Compare 1 over-capture flag

LL_LPTIM_DIER_CC2OFIE

Capture/Compare 2 over-capture flag

LL_LPTIM_DIER_ARRMIE

Autoreload match

LL_LPTIM_DIER_EXTTRIGIE

External trigger edge event

LL_LPTIM_DIER_ARROKIE

Autoreload register update OK

LL_LPTIM_DIER_UPIE

Counter direction change down to up

LL_LPTIM_DIER_DOWNIE

Counter direction change up to down

LL_LPTIM_DIER_UEIE

Update event

LL_LPTIM_DIER_REPOKIE

Repetition register update OK

Input Capture Filter

LL_LPTIM_ICFLT_CLOCK_DIV1

any external input capture signal level change is considered as a valid transition

LL_LPTIM_ICFLT_CLOCK_DIV2

external input capture signal level change must be stable for at least 2 clock periods before it is considered as valid transition

LL_LPTIM_ICFLT_CLOCK_DIV4

external input capture signal level change must be stable for at least 4 clock periods before it is considered as valid transition

LL_LPTIM_ICFLT_CLOCK_DIV8

external input capture signal level change must be stable for at least 8 clock periods before it is considered as valid transition

Input Capture Polarity

LL_LPTIM_ICPOLARITY_RISING

Capture/Compare input rising polarity

LL_LPTIM_ICPOLARITY_FALLING

Capture/Compare input falling polarity

LL_LPTIM_ICPOLARITY_RISING_FALLING

Capture/Compare input rising and falling polarities

Input Capture Prescaler

LL_LPTIM_ICPSC_DIV1

Capture performed each time an edge is detected on the capture input

LL_LPTIM_ICPSC_DIV2

Capture performed once every 2 events

LL_LPTIM_ICPSC_DIV4

Capture performed once every 4 events

LL_LPTIM_ICPSC_DIV8

Capture performed once every 8 events

Input Capture selection

LL_LPTIM_CCMODE_OUTPUT_PWM

Select PWM mode

LL_LPTIM_CCMODE_INPUTCAPTURE

Select Input Capture mode

Operating Mode

LL_LPTIM_OPERATING_MODE_CONTINUOUS

LP Timer starts in continuous mode

LL_LPTIM_OPERATING_MODE_ONESHOT

LP Tilmer starts in single mode

Output Polarity

LL_LPTIM_OUTPUT_POLARITY_REGULAR

The LPTIM output reflects the compare results between LPTIMx_ARR and LPTIMx_CCRx registers

LL_LPTIM_OUTPUT_POLARITY_INVERSE

The LPTIM output reflects the inverse of the compare results between LPTIMx_ARR and LPTIMx_CCx registers

Output Waveform Type

LL_LPTIM_OUTPUT_WAVEFORM_PWM

LPTIM generates either a PWM waveform or a One pulse waveform depending on chosen operating mode CONTINUOUS or SINGLE

LL_LPTIM_OUTPUT_WAVEFORM_SETONCE

LPTIM generates a Set Once waveform

Prescaler Value

LL_LPTIM_PRESCALER_DIV1

Prescaler division factor is set to 1

LL_LPTIM_PRESCALER_DIV2

Prescaler division factor is set to 2

LL_LPTIM_PRESCALER_DIV4

Prescaler division factor is set to 4

LL_LPTIM_PRESCALER_DIV8

Prescaler division factor is set to 8

LL_LPTIM_PRESCALER_DIV16

Prescaler division factor is set to 16

LL_LPTIM_PRESCALER_DIV32

Prescaler division factor is set to 32

LL_LPTIM_PRESCALER_DIV64

Prescaler division factor is set to 64

LL_LPTIM_PRESCALER_DIV128

Prescaler division factor is set to 128

Trigger Filter

LL_LPTIM_TRIG_FILTER_NONE

Any trigger active level change is considered as a valid trigger

LL_LPTIM_TRIG_FILTER_2

Trigger active level change must be stable for at least 2 clock periods before it is considered as valid trigger

LL_LPTIM_TRIG_FILTER_4

Trigger active level change must be stable for at least 4 clock periods before it is considered as valid trigger

LL_LPTIM_TRIG_FILTER_8

Trigger active level change must be stable for at least 8 clock periods before it is considered as valid trigger

Trigger Polarity

LL_LPTIM_TRIG_POLARITY_RISING

LPTIM counter starts when a rising edge is detected

LL_LPTIM_TRIG_POLARITY_FALLING

LPTIM counter starts when a falling edge is detected

LL_LPTIM_TRIG_POLARITY_RISING_FALLING

LPTIM counter starts when a rising or a falling edge is detected

Trigger Source

LL_LPTIM_TRIG_SOURCE_GPIO

External input trigger is connected to TIMx_ETR input

LL_LPTIM_TRIG_SOURCE_RTCALARMA

External input trigger is connected to RTC Alarm A

LL_LPTIM_TRIG_SOURCE_RTCALARMB

External input trigger is connected to RTC Alarm B

LL_LPTIM_TRIG_SOURCE_RTCTAMP1

External input trigger is connected to RTC Tamper 1

LL_LPTIM_TRIG_SOURCE_RTCTAMP2

External input trigger is connected to RTC Tamper 2

LL_LPTIM_TRIG_SOURCE_RTCTAMP3

External input trigger is connected to RTC Tamper 3

LL_LPTIM_TRIG_SOURCE_SAI2_FS_A

External input trigger is connected to SAI2 FS A output

LL_LPTIM_TRIG_SOURCE_SAI2_FS_B

External input trigger is connected to SAI2 FS B output

LL_LPTIM_TRIG_SOURCE_LPTIM2_CH1

External input trigger is connected to LPTIM2 Channel 1

LL_LPTIM_TRIG_SOURCE_LPTIM3_CH1

External input trigger is connected to LPTIM3 Channel 1

LL_LPTIM_TRIG_SOURCE_LPTIM4_OUT

External input trigger is connected to LPTIM4 Output

LL_LPTIM_TRIG_SOURCE_LPTIM5_OUT

External input trigger is connected to LPTIM5 Output

Update Mode

LL_LPTIM_UPDATE_MODE_IMMEDIATE

Preload is disabled: registers are updated after each APB bus write access

LL_LPTIM_UPDATE_MODE_ENDOFPERIOD

preload is enabled: registers are updated at the end of the current LPTIM period

Common Write and read registers Macros

LL_LPTIM_WriteReg

Description:

- Write a value in LPTIM register.

Parameters:

- __INSTANCE__: LPTIM Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_LPTIM_ReadReg

Description:

- Read a value in LPTIM register.

Parameters:

- __INSTANCE__: LPTIM Instance
- __REG__: Register to be read

Return value:

- Register: value

LL_LPTIM_IC_GET_OFFSET

Description:

- LPTimer Input Capture Get Offset(in counter step unit)

Parameters:

- __PSC__: This parameter can be one of the following values:
  - LL_LPTIM_PRESCALER_DIV1
  - LL_LPTIM_PRESCALER_DIV2
  - LL_LPTIM_PRESCALER_DIV4
  - LL_LPTIM_PRESCALER_DIV8
  - LL_LPTIM_PRESCALER_DIV16
  - LL_LPTIM_PRESCALER_DIV32
  - LL_LPTIM_PRESCALER_DIV64
  - LL_LPTIM_PRESCALER_DIV128
- __FLT__: This parameter can be one of the following values:
  - LL_LPTIM_ICFLT_CLOCK_DIV1
  - LL_LPTIM_ICFLT_CLOCK_DIV2
  - LL_LPTIM_ICFLT_CLOCK_DIV4
  - LL_LPTIM_ICFLT_CLOCK_DIV8

Return value:

- offset: value

Notes:

- The real capture value corresponding to the input capture trigger can be calculated using the formula hereafter : Real capture value = captured(LPTIM_CCRx) - offset The Offset value is depending on the glitch filter value for the channel and the value of the prescaler for the kernel clock. Please check Errata Sheet V1_8 for more details under "variable latency on input capture channel" section.

99 LL LPUART Generic Driver

99.1 LPUART Firmware driver registers structures

99.1.1 LL_LPUART_InitTypeDef

LL_LPUART_InitTypeDef is defined in the stm32n6xx_ll_lpuart.h

Data Fields

- ***uint32_t PrescalerValue***
- ***uint32_t BaudRate***
- ***uint32_t DataWidth***
- ***uint32_t StopBits***
- ***uint32_t Parity***
- ***uint32_t TransferDirection***
- ***uint32_t HardwareFlowControl***

Field Documentation

- ***uint32_t LL_LPUART_InitTypeDef::PrescalerValue***
  Specifies the Prescaler to compute the communication baud rate. This parameter can be a value of ***LPUART_LL_EC_PRESCALER***.This feature can be modified afterwards using unitary function **LL_LPUART_SetPrescaler()**.
- ***uint32_t LL_LPUART_InitTypeDef::BaudRate***
  This field defines expected LPUART communication baud rate.This feature can be modified afterwards using unitary function **LL_LPUART_SetBaudRate()**.
- ***uint32_t LL_LPUART_InitTypeDef::DataWidth***
  Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of ***LPUART_LL_EC_DATAWIDTH***.This feature can be modified afterwards using unitary function **LL_LPUART_SetDataWidth()**.
- ***uint32_t LL_LPUART_InitTypeDef::StopBits***
  Specifies the number of stop bits transmitted. This parameter can be a value of ***LPUART_LL_EC_STOPBITS***.This feature can be modified afterwards using unitary function **LL_LPUART_SetStopBitsLength()**.
- ***uint32_t LL_LPUART_InitTypeDef::Parity***
  Specifies the parity mode. This parameter can be a value of ***LPUART_LL_EC_PARITY***.This feature can be modified afterwards using unitary function **LL_LPUART_SetParity()**.
- ***uint32_t LL_LPUART_InitTypeDef::TransferDirection***
  Specifies whether the Receive and/or Transmit mode is enabled or disabled. This parameter can be a value of ***LPUART_LL_EC_DIRECTION***.This feature can be modified afterwards using unitary function **LL_LPUART_SetTransferDirection()**.
- ***uint32_t LL_LPUART_InitTypeDef::HardwareFlowControl***
  Specifies whether the hardware flow control mode is enabled or disabled. This parameter can be a value of ***LPUART_LL_EC_HWCONTROL***.This feature can be modified afterwards using unitary function **LL_LPUART_SetHWFlowCtrl()**.

99.2 LPUART Firmware driver API description

The following section lists the various functions of the LPUART library.

99.2.1 Detailed description of functions

LL_LPUART_Enable

Function name

__STATIC_INLINE void LL_LPUART_Enable (USART_TypeDef * LPUARTx)

Function description

LPUART Enable.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 UE LL_LPUART_Enable

LL_LPUART_Disable

Function name

__STATIC_INLINE void LL_LPUART_Disable (USART_TypeDef * LPUARTx)

Function description

LPUART Disable.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Notes

- When LPUART is disabled, LPUART prescalers and outputs are stopped immediately, and current operations are discarded. The configuration of the LPUART is kept, but all the status flags, in the LPUARTx_ISR are set to their default values.
- In order to go into low-power mode without generating errors on the line, the TE bit must be reset before and the software must wait for the TC bit in the LPUART_ISR to be set before resetting the UE bit. The DMA requests are also reset when UE = 0 so the DMA channel must be disabled before resetting the UE bit.

Reference Manual to LL API cross reference:

- CR1 UE LL_LPUART_Disable

LL_LPUART_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabled (const USART_TypeDef * LPUARTx)

Function description

Indicate if LPUART is enabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 UE LL_LPUART_IsEnabled

LL_LPUART_EnableFIFO

Function name

__STATIC_INLINE void LL_LPUART_EnableFIFO (USART_TypeDef * LPUARTx)

Function description

FIFO Mode Enable.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 FIFOEN LL_LPUART_EnableFIFO

LL_LPUART_DisableFIFO

Function name

__STATIC_INLINE void LL_LPUART_DisableFIFO (USART_TypeDef * LPUARTx)

Function description

FIFO Mode Disable.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 FIFOEN LL_LPUART_DisableFIFO

LL_LPUART_IsEnabledFIFO

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledFIFO (const USART_TypeDef * LPUARTx)

Function description

Indicate if FIFO Mode is enabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 FIFOEN LL_LPUART_IsEnabledFIFO

LL_LPUART_SetTXFIFOThreshold

Function name

__STATIC_INLINE void LL_LPUART_SetTXFIFOThreshold (USART_TypeDef * LPUARTx, uint32_t Threshold)

Function description

Configure TX FIFO Threshold.

Parameters

- **LPUARTx:** LPUART Instance
- **Threshold:** This parameter can be one of the following values:
  - LL_LPUART_FIFOTHRESHOLD_1_8
  - LL_LPUART_FIFOTHRESHOLD_1_4
  - LL_LPUART_FIFOTHRESHOLD_1_2
  - LL_LPUART_FIFOTHRESHOLD_3_4
  - LL_LPUART_FIFOTHRESHOLD_7_8
  - LL_LPUART_FIFOTHRESHOLD_8_8

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 TXFTCFG LL_LPUART_SetTXFIFOThreshold

LL_LPUART_GetTXFIFOThreshold

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetTXFIFOThreshold (const USART_TypeDef * LPUARTx)

Function description

Return TX FIFO Threshold Configuration.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_FIFOTHRESHOLD_1_8
  - LL_LPUART_FIFOTHRESHOLD_1_4
  - LL_LPUART_FIFOTHRESHOLD_1_2
  - LL_LPUART_FIFOTHRESHOLD_3_4
  - LL_LPUART_FIFOTHRESHOLD_7_8
  - LL_LPUART_FIFOTHRESHOLD_8_8

Reference Manual to LL API cross reference:

- CR3 TXFTCFG LL_LPUART_GetTXFIFOThreshold

LL_LPUART_SetRXFIFOThreshold

Function name

__STATIC_INLINE void LL_LPUART_SetRXFIFOThreshold (USART_TypeDef * LPUARTx, uint32_t Threshold)

Function description

Configure RX FIFO Threshold.

Parameters

- **LPUARTx:** LPUART Instance
- **Threshold:** This parameter can be one of the following values:
  - LL_LPUART_FIFOTHRESHOLD_1_8
  - LL_LPUART_FIFOTHRESHOLD_1_4
  - LL_LPUART_FIFOTHRESHOLD_1_2
  - LL_LPUART_FIFOTHRESHOLD_3_4
  - LL_LPUART_FIFOTHRESHOLD_7_8
  - LL_LPUART_FIFOTHRESHOLD_8_8

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 RXFTCFG LL_LPUART_SetRXFIFOThreshold

LL_LPUART_GetRXFIFOThreshold

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetRXFIFOThreshold (const USART_TypeDef * LPUARTx)

Function description

Return RX FIFO Threshold Configuration.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_FIFOTHRESHOLD_1_8
  - LL_LPUART_FIFOTHRESHOLD_1_4
  - LL_LPUART_FIFOTHRESHOLD_1_2
  - LL_LPUART_FIFOTHRESHOLD_3_4
  - LL_LPUART_FIFOTHRESHOLD_7_8
  - LL_LPUART_FIFOTHRESHOLD_8_8

Reference Manual to LL API cross reference:

- CR3 RXFTCFG LL_LPUART_GetRXFIFOThreshold

LL_LPUART_ConfigFIFOsThreshold

Function name

__STATIC_INLINE void LL_LPUART_ConfigFIFOsThreshold (USART_TypeDef * LPUARTx, uint32_t TXThreshold, uint32_t RXThreshold)

Function description

Configure TX and RX FIFOs Threshold.

Parameters

- **LPUARTx:** LPUART Instance
- **TXThreshold:** This parameter can be one of the following values:
  - LL_LPUART_FIFOTHRESHOLD_1_8
  - LL_LPUART_FIFOTHRESHOLD_1_4
  - LL_LPUART_FIFOTHRESHOLD_1_2
  - LL_LPUART_FIFOTHRESHOLD_3_4
  - LL_LPUART_FIFOTHRESHOLD_7_8
  - LL_LPUART_FIFOTHRESHOLD_8_8
- **RXThreshold:** This parameter can be one of the following values:
  - LL_LPUART_FIFOTHRESHOLD_1_8
  - LL_LPUART_FIFOTHRESHOLD_1_4
  - LL_LPUART_FIFOTHRESHOLD_1_2
  - LL_LPUART_FIFOTHRESHOLD_3_4
  - LL_LPUART_FIFOTHRESHOLD_7_8
  - LL_LPUART_FIFOTHRESHOLD_8_8

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 TXFTCFG LL_LPUART_ConfigFIFOsThreshold
- CR3 RXFTCFG LL_LPUART_ConfigFIFOsThreshold

LL_LPUART_EnableInStopMode

Function name

__STATIC_INLINE void LL_LPUART_EnableInStopMode (USART_TypeDef * LPUARTx)

Function description

LPUART enabled in STOP Mode.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Notes

- When this function is enabled, LPUART is able to wake up the MCU from Stop mode, provided that LPUART clock selection is HSI or LSE in RCC.

Reference Manual to LL API cross reference:

- CR1 UESM LL_LPUART_EnableInStopMode

LL_LPUART_DisableInStopMode

Function name

__STATIC_INLINE void LL_LPUART_DisableInStopMode (USART_TypeDef * LPUARTx)

Function description

LPUART disabled in STOP Mode.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Notes

- When this function is disabled, LPUART is not able to wake up the MCU from Stop mode

Reference Manual to LL API cross reference:

- CR1 UESM LL_LPUART_DisableInStopMode

LL_LPUART_IsEnabledInStopMode

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledInStopMode (const USART_TypeDef * LPUARTx)

Function description

Indicate if LPUART is enabled in STOP Mode (able to wake up MCU from Stop mode or not)

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 UESM LL_LPUART_IsEnabledInStopMode

LL_LPUART_EnableDirectionRx

Function name

__STATIC_INLINE void LL_LPUART_EnableDirectionRx (USART_TypeDef * LPUARTx)

Function description

Receiver Enable (Receiver is enabled and begins searching for a start bit)

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RE LL_LPUART_EnableDirectionRx

LL_LPUART_DisableDirectionRx

Function name

__STATIC_INLINE void LL_LPUART_DisableDirectionRx (USART_TypeDef * LPUARTx)

Function description

Receiver Disable.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RE LL_LPUART_DisableDirectionRx

LL_LPUART_EnableDirectionTx

Function name

__STATIC_INLINE void LL_LPUART_EnableDirectionTx (USART_TypeDef * LPUARTx)

Function description

Transmitter Enable.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TE LL_LPUART_EnableDirectionTx

LL_LPUART_DisableDirectionTx

Function name

__STATIC_INLINE void LL_LPUART_DisableDirectionTx (USART_TypeDef * LPUARTx)

Function description

Transmitter Disable.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TE LL_LPUART_DisableDirectionTx

LL_LPUART_SetTransferDirection

Function name

__STATIC_INLINE void LL_LPUART_SetTransferDirection (USART_TypeDef * LPUARTx, uint32_t TransferDirection)

Function description

Configure simultaneously enabled/disabled states of Transmitter and Receiver.

Parameters

- **LPUARTx:** LPUART Instance
- **TransferDirection:** This parameter can be one of the following values:
  - LL_LPUART_DIRECTION_NONE
  - LL_LPUART_DIRECTION_RX
  - LL_LPUART_DIRECTION_TX
  - LL_LPUART_DIRECTION_TX_RX

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RE LL_LPUART_SetTransferDirection
- CR1 TE LL_LPUART_SetTransferDirection

LL_LPUART_GetTransferDirection

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetTransferDirection (const USART_TypeDef * LPUARTx)

Function description

Return enabled/disabled states of Transmitter and Receiver.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_DIRECTION_NONE
  - LL_LPUART_DIRECTION_RX
  - LL_LPUART_DIRECTION_TX
  - LL_LPUART_DIRECTION_TX_RX

Reference Manual to LL API cross reference:

- CR1 RE LL_LPUART_GetTransferDirection
- CR1 TE LL_LPUART_GetTransferDirection

LL_LPUART_SetParity

Function name

__STATIC_INLINE void LL_LPUART_SetParity (USART_TypeDef * LPUARTx, uint32_t Parity)

Function description

Configure Parity (enabled/disabled and parity mode if enabled)

Parameters

- **LPUARTx:** LPUART Instance
- **Parity:** This parameter can be one of the following values:
  - LL_LPUART_PARITY_NONE
  - LL_LPUART_PARITY_EVEN
  - LL_LPUART_PARITY_ODD

Return values

- **None:**

Notes

- This function selects if hardware parity control (generation and detection) is enabled or disabled. When the parity control is enabled (Odd or Even), computed parity bit is inserted at the MSB position (depending on data width) and parity is checked on the received data.

Reference Manual to LL API cross reference:

- CR1 PS LL_LPUART_SetParity
- CR1 PCE LL_LPUART_SetParity

LL_LPUART_GetParity

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetParity (const USART_TypeDef * LPUARTx)

Function description

Return Parity configuration (enabled/disabled and parity mode if enabled)

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_PARITY_NONE
  - LL_LPUART_PARITY_EVEN
  - LL_LPUART_PARITY_ODD

Reference Manual to LL API cross reference:

- CR1 PS LL_LPUART_GetParity
- CR1 PCE LL_LPUART_GetParity

LL_LPUART_SetWakeUpMethod

Function name

__STATIC_INLINE void LL_LPUART_SetWakeUpMethod (USART_TypeDef * LPUARTx, uint32_t Method)

Function description

Set Receiver Wake Up method from Mute mode.

Parameters

- **LPUARTx:** LPUART Instance
- **Method:** This parameter can be one of the following values:
  - LL_LPUART_WAKEUP_IDLELINE
  - LL_LPUART_WAKEUP_ADDRESSMARK

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 WAKE LL_LPUART_SetWakeUpMethod

LL_LPUART_GetWakeUpMethod

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetWakeUpMethod (const USART_TypeDef * LPUARTx)

Function description

Return Receiver Wake Up method from Mute mode.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_WAKEUP_IDLELINE
  - LL_LPUART_WAKEUP_ADDRESSMARK

Reference Manual to LL API cross reference:

- CR1 WAKE LL_LPUART_GetWakeUpMethod

LL_LPUART_SetDataWidth

Function name

__STATIC_INLINE void LL_LPUART_SetDataWidth (USART_TypeDef * LPUARTx, uint32_t DataWidth)

Function description

Set Word length (nb of data bits, excluding start and stop bits)

Parameters

- **LPUARTx:** LPUART Instance
- **DataWidth:** This parameter can be one of the following values:
  - LL_LPUART_DATAWIDTH_7B
  - LL_LPUART_DATAWIDTH_8B
  - LL_LPUART_DATAWIDTH_9B

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 M LL_LPUART_SetDataWidth

LL_LPUART_GetDataWidth

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetDataWidth (const USART_TypeDef * LPUARTx)

Function description

Return Word length (i.e.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_DATAWIDTH_7B
  - LL_LPUART_DATAWIDTH_8B
  - LL_LPUART_DATAWIDTH_9B

Reference Manual to LL API cross reference:

- CR1 M LL_LPUART_GetDataWidth

LL_LPUART_EnableMuteMode

Function name

__STATIC_INLINE void LL_LPUART_EnableMuteMode (USART_TypeDef * LPUARTx)

Function description

Allow switch between Mute Mode and Active mode.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 MME LL_LPUART_EnableMuteMode

LL_LPUART_DisableMuteMode

Function name

__STATIC_INLINE void LL_LPUART_DisableMuteMode (USART_TypeDef * LPUARTx)

Function description

Prevent Mute Mode use.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 MME LL_LPUART_DisableMuteMode

LL_LPUART_IsEnabledMuteMode

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledMuteMode (const USART_TypeDef * LPUARTx)

Function description

Indicate if switch between Mute Mode and Active mode is allowed.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 MME LL_LPUART_IsEnabledMuteMode

LL_LPUART_SetPrescaler

Function name

__STATIC_INLINE void LL_LPUART_SetPrescaler (USART_TypeDef * LPUARTx, uint32_t PrescalerValue)

Function description

Configure Clock source prescaler for baudrate generator and oversampling.

Parameters

- **LPUARTx:** LPUART Instance
- **PrescalerValue:** This parameter can be one of the following values:
  - LL_LPUART_PRESCALER_DIV1
  - LL_LPUART_PRESCALER_DIV2
  - LL_LPUART_PRESCALER_DIV4
  - LL_LPUART_PRESCALER_DIV6
  - LL_LPUART_PRESCALER_DIV8
  - LL_LPUART_PRESCALER_DIV10
  - LL_LPUART_PRESCALER_DIV12
  - LL_LPUART_PRESCALER_DIV16
  - LL_LPUART_PRESCALER_DIV32
  - LL_LPUART_PRESCALER_DIV64
  - LL_LPUART_PRESCALER_DIV128
  - LL_LPUART_PRESCALER_DIV256

Return values

- **None:**

Reference Manual to LL API cross reference:

- PRESC PRESCALER LL_LPUART_SetPrescaler

LL_LPUART_GetPrescaler

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetPrescaler (const USART_TypeDef * LPUARTx)

Function description

Retrieve the Clock source prescaler for baudrate generator and oversampling.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_PRESCALER_DIV1
  - LL_LPUART_PRESCALER_DIV2
  - LL_LPUART_PRESCALER_DIV4
  - LL_LPUART_PRESCALER_DIV6
  - LL_LPUART_PRESCALER_DIV8
  - LL_LPUART_PRESCALER_DIV10
  - LL_LPUART_PRESCALER_DIV12
  - LL_LPUART_PRESCALER_DIV16
  - LL_LPUART_PRESCALER_DIV32
  - LL_LPUART_PRESCALER_DIV64
  - LL_LPUART_PRESCALER_DIV128
  - LL_LPUART_PRESCALER_DIV256

Reference Manual to LL API cross reference:

- PRESC PRESCALER LL_LPUART_GetPrescaler

LL_LPUART_SetStopBitsLength

Function name

__STATIC_INLINE void LL_LPUART_SetStopBitsLength (USART_TypeDef * LPUARTx, uint32_t StopBits)

Function description

Set the length of the stop bits.

Parameters

- **LPUARTx:** LPUART Instance
- **StopBits:** This parameter can be one of the following values:
  - LL_LPUART_STOPBITS_1
  - LL_LPUART_STOPBITS_2

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 STOP LL_LPUART_SetStopBitsLength

LL_LPUART_GetStopBitsLength

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetStopBitsLength (const USART_TypeDef * LPUARTx)

Function description

Retrieve the length of the stop bits.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_STOPBITS_1
  - LL_LPUART_STOPBITS_2

Reference Manual to LL API cross reference:

- CR2 STOP LL_LPUART_GetStopBitsLength

LL_LPUART_ConfigCharacter

Function name

__STATIC_INLINE void LL_LPUART_ConfigCharacter (USART_TypeDef * LPUARTx, uint32_t DataWidth, uint32_t Parity, uint32_t StopBits)

Function description

Configure Character frame format (Datawidth, Parity control, Stop Bits)

Parameters

- **LPUARTx:** LPUART Instance
- **DataWidth:** This parameter can be one of the following values:
  - LL_LPUART_DATAWIDTH_7B
  - LL_LPUART_DATAWIDTH_8B
  - LL_LPUART_DATAWIDTH_9B
- **Parity:** This parameter can be one of the following values:
  - LL_LPUART_PARITY_NONE
  - LL_LPUART_PARITY_EVEN
  - LL_LPUART_PARITY_ODD
- **StopBits:** This parameter can be one of the following values:
  - LL_LPUART_STOPBITS_1
  - LL_LPUART_STOPBITS_2

Return values

- **None:**

Notes

- Call of this function is equivalent to following function call sequence : Data Width configuration using LL_LPUART_SetDataWidth() functionParity Control and mode configuration using LL_LPUART_SetParity() functionStop bits configuration using LL_LPUART_SetStopBitsLength() function

Reference Manual to LL API cross reference:

- CR1 PS LL_LPUART_ConfigCharacter
- CR1 PCE LL_LPUART_ConfigCharacter
- CR1 M LL_LPUART_ConfigCharacter
- CR2 STOP LL_LPUART_ConfigCharacter

LL_LPUART_SetTXRXSwap

Function name

__STATIC_INLINE void LL_LPUART_SetTXRXSwap (USART_TypeDef * LPUARTx, uint32_t SwapConfig)

Function description

Configure TX/RX pins swapping setting.

Parameters

- **LPUARTx:** LPUART Instance
- **SwapConfig:** This parameter can be one of the following values:
  - LL_LPUART_TXRX_STANDARD
  - LL_LPUART_TXRX_SWAPPED

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 SWAP LL_LPUART_SetTXRXSwap

LL_LPUART_GetTXRXSwap

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetTXRXSwap (const USART_TypeDef * LPUARTx)

Function description

Retrieve TX/RX pins swapping configuration.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_TXRX_STANDARD
  - LL_LPUART_TXRX_SWAPPED

Reference Manual to LL API cross reference:

- CR2 SWAP LL_LPUART_GetTXRXSwap

LL_LPUART_SetRXPinLevel

Function name

__STATIC_INLINE void LL_LPUART_SetRXPinLevel (USART_TypeDef * LPUARTx, uint32_t PinInvMethod)

Function description

Configure RX pin active level logic.

Parameters

- **LPUARTx:** LPUART Instance
- **PinInvMethod:** This parameter can be one of the following values:
  - LL_LPUART_RXPIN_LEVEL_STANDARD
  - LL_LPUART_RXPIN_LEVEL_INVERTED

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 RXINV LL_LPUART_SetRXPinLevel

LL_LPUART_GetRXPinLevel

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetRXPinLevel (const USART_TypeDef * LPUARTx)

Function description

Retrieve RX pin active level logic configuration.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_RXPIN_LEVEL_STANDARD
  - LL_LPUART_RXPIN_LEVEL_INVERTED

Reference Manual to LL API cross reference:

- CR2 RXINV LL_LPUART_GetRXPinLevel

LL_LPUART_SetTXPinLevel

Function name

__STATIC_INLINE void LL_LPUART_SetTXPinLevel (USART_TypeDef * LPUARTx, uint32_t PinInvMethod)

Function description

Configure TX pin active level logic.

Parameters

- **LPUARTx:** LPUART Instance
- **PinInvMethod:** This parameter can be one of the following values:
  - LL_LPUART_TXPIN_LEVEL_STANDARD
  - LL_LPUART_TXPIN_LEVEL_INVERTED

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 TXINV LL_LPUART_SetTXPinLevel

LL_LPUART_GetTXPinLevel

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetTXPinLevel (const USART_TypeDef * LPUARTx)

Function description

Retrieve TX pin active level logic configuration.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_TXPIN_LEVEL_STANDARD
  - LL_LPUART_TXPIN_LEVEL_INVERTED

Reference Manual to LL API cross reference:

- CR2 TXINV LL_LPUART_GetTXPinLevel

LL_LPUART_SetBinaryDataLogic

Function name

__STATIC_INLINE void LL_LPUART_SetBinaryDataLogic (USART_TypeDef * LPUARTx, uint32_t DataLogic)

Function description

Configure Binary data logic.

Parameters

- **LPUARTx:** LPUART Instance
- **DataLogic:** This parameter can be one of the following values:
  - LL_LPUART_BINARY_LOGIC_POSITIVE
  - LL_LPUART_BINARY_LOGIC_NEGATIVE

Return values

- **None:**

Notes

- Allow to define how Logical data from the data register are send/received : either in positive/direct logic (1=H, 0=L) or in negative/inverse logic (1=L, 0=H)

Reference Manual to LL API cross reference:

- CR2 DATAINV LL_LPUART_SetBinaryDataLogic

LL_LPUART_GetBinaryDataLogic

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetBinaryDataLogic (const USART_TypeDef * LPUARTx)

Function description

Retrieve Binary data configuration.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_BINARY_LOGIC_POSITIVE
  - LL_LPUART_BINARY_LOGIC_NEGATIVE

Reference Manual to LL API cross reference:

- CR2 DATAINV LL_LPUART_GetBinaryDataLogic

LL_LPUART_SetTransferBitOrder

Function name

__STATIC_INLINE void LL_LPUART_SetTransferBitOrder (USART_TypeDef * LPUARTx, uint32_t BitOrder)

Function description

Configure transfer bit order (either Less or Most Significant Bit First)

Parameters

- **LPUARTx:** LPUART Instance
- **BitOrder:** This parameter can be one of the following values:
  - LL_LPUART_BITORDER_LSBFIRST
  - LL_LPUART_BITORDER_MSBFIRST

Return values

- **None:**

Notes

- MSB First means data is transmitted/received with the MSB first, following the start bit. LSB First means data is transmitted/received with data bit 0 first, following the start bit.

Reference Manual to LL API cross reference:

- CR2 MSBFIRST LL_LPUART_SetTransferBitOrder

LL_LPUART_GetTransferBitOrder

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetTransferBitOrder (const USART_TypeDef * LPUARTx)

Function description

Return transfer bit order (either Less or Most Significant Bit First)

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_BITORDER_LSBFIRST
  - LL_LPUART_BITORDER_MSBFIRST

Notes

- MSB First means data is transmitted/received with the MSB first, following the start bit. LSB First means data is transmitted/received with data bit 0 first, following the start bit.

Reference Manual to LL API cross reference:

- CR2 MSBFIRST LL_LPUART_GetTransferBitOrder

LL_LPUART_ConfigNodeAddress

Function name

__STATIC_INLINE void LL_LPUART_ConfigNodeAddress (USART_TypeDef * LPUARTx, uint32_t AddressLen, uint32_t NodeAddress)

Function description

Set Address of the LPUART node.

Parameters

- **LPUARTx:** LPUART Instance
- **AddressLen:** This parameter can be one of the following values:
  - LL_LPUART_ADDRESS_DETECT_4B
  - LL_LPUART_ADDRESS_DETECT_7B
- **NodeAddress:** 4 or 7 bit Address of the LPUART node.

Return values

- **None:**

Notes

- This is used in multiprocessor communication during Mute mode or Stop mode, for wake up with address mark detection.
- 4bits address node is used when 4-bit Address Detection is selected in ADDM7. (b7-b4 should be set to 0) 8bits address node is used when 7-bit Address Detection is selected in ADDM7. (This is used in multiprocessor communication during Mute mode or Stop mode, for wake up with 7-bit address mark detection. The MSB of the character sent by the transmitter should be equal to 1. It may also be used for character detection during normal reception, Mute mode inactive (for example, end of block detection in ModBus protocol). In this case, the whole received character (8-bit) is compared to the ADD[7:0] value and CMF flag is set on match)

Reference Manual to LL API cross reference:

- CR2 ADD LL_LPUART_ConfigNodeAddress
- CR2 ADDM7 LL_LPUART_ConfigNodeAddress

LL_LPUART_GetNodeAddress

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetNodeAddress (const USART_TypeDef * LPUARTx)

Function description

Return 8 bit Address of the LPUART node as set in ADD field of CR2.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Address:** of the LPUART node (Value between Min_Data=0 and Max_Data=255)

Notes

- If 4-bit Address Detection is selected in ADDM7, only 4bits (b3-b0) of returned value are relevant (b31-b4 are not relevant) If 7-bit Address Detection is selected in ADDM7, only 8bits (b7-b0) of returned value are relevant (b31-b8 are not relevant)

Reference Manual to LL API cross reference:

- CR2 ADD LL_LPUART_GetNodeAddress

LL_LPUART_GetNodeAddressLen

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetNodeAddressLen (const USART_TypeDef * LPUARTx)

Function description

Return Length of Node Address used in Address Detection mode (7-bit or 4-bit)

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_ADDRESS_DETECT_4B
  - LL_LPUART_ADDRESS_DETECT_7B

Reference Manual to LL API cross reference:

- CR2 ADDM7 LL_LPUART_GetNodeAddressLen

LL_LPUART_EnableRTSHWFlowCtrl

Function name

__STATIC_INLINE void LL_LPUART_EnableRTSHWFlowCtrl (USART_TypeDef * LPUARTx)

Function description

Enable RTS HW Flow Control.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 RTSE LL_LPUART_EnableRTSHWFlowCtrl

LL_LPUART_DisableRTSHWFlowCtrl

Function name

__STATIC_INLINE void LL_LPUART_DisableRTSHWFlowCtrl (USART_TypeDef * LPUARTx)

Function description

Disable RTS HW Flow Control.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 RTSE LL_LPUART_DisableRTSHWFlowCtrl

LL_LPUART_EnableCTSHWFlowCtrl

Function name

__STATIC_INLINE void LL_LPUART_EnableCTSHWFlowCtrl (USART_TypeDef * LPUARTx)

Function description

Enable CTS HW Flow Control.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 CTSE LL_LPUART_EnableCTSHWFlowCtrl

LL_LPUART_DisableCTSHWFlowCtrl

Function name

__STATIC_INLINE void LL_LPUART_DisableCTSHWFlowCtrl (USART_TypeDef * LPUARTx)

Function description

Disable CTS HW Flow Control.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 CTSE LL_LPUART_DisableCTSHWFlowCtrl

LL_LPUART_SetHWFlowCtrl

Function name

__STATIC_INLINE void LL_LPUART_SetHWFlowCtrl (USART_TypeDef * LPUARTx, uint32_t HardwareFlowControl)

Function description

Configure HW Flow Control mode (both CTS and RTS)

Parameters

- **LPUARTx:** LPUART Instance
- **HardwareFlowControl:** This parameter can be one of the following values:
  - LL_LPUART_HWCONTROL_NONE
  - LL_LPUART_HWCONTROL_RTS
  - LL_LPUART_HWCONTROL_CTS
  - LL_LPUART_HWCONTROL_RTS_CTS

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 RTSE LL_LPUART_SetHWFlowCtrl
- CR3 CTSE LL_LPUART_SetHWFlowCtrl

LL_LPUART_GetHWFlowCtrl

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetHWFlowCtrl (const USART_TypeDef * LPUARTx)

Function description

Return HW Flow Control configuration (both CTS and RTS)

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_HWCONTROL_NONE
  - LL_LPUART_HWCONTROL_RTS
  - LL_LPUART_HWCONTROL_CTS
  - LL_LPUART_HWCONTROL_RTS_CTS

Reference Manual to LL API cross reference:

- CR3 RTSE LL_LPUART_GetHWFlowCtrl
- CR3 CTSE LL_LPUART_GetHWFlowCtrl

LL_LPUART_EnableOverrunDetect

Function name

__STATIC_INLINE void LL_LPUART_EnableOverrunDetect (USART_TypeDef * LPUARTx)

Function description

Enable Overrun detection.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 OVRDIS LL_LPUART_EnableOverrunDetect

LL_LPUART_DisableOverrunDetect

Function name

__STATIC_INLINE void LL_LPUART_DisableOverrunDetect (USART_TypeDef * LPUARTx)

Function description

Disable Overrun detection.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 OVRDIS LL_LPUART_DisableOverrunDetect

LL_LPUART_IsEnabledOverrunDetect

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledOverrunDetect (const USART_TypeDef * LPUARTx)

Function description

Indicate if Overrun detection is enabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 OVRDIS LL_LPUART_IsEnabledOverrunDetect

LL_LPUART_SetWKUPType

Function name

__STATIC_INLINE void LL_LPUART_SetWKUPType (USART_TypeDef * LPUARTx, uint32_t Type)

Function description

Select event type for Wake UP Interrupt Flag (WUS[1:0] bits)

Parameters

- **LPUARTx:** LPUART Instance
- **Type:** This parameter can be one of the following values:
  - LL_LPUART_WAKEUP_ON_ADDRESS
  - LL_LPUART_WAKEUP_ON_STARTBIT
  - LL_LPUART_WAKEUP_ON_RXNE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 WUS LL_LPUART_SetWKUPType

LL_LPUART_GetWKUPType

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetWKUPType (const USART_TypeDef * LPUARTx)

Function description

Return event type for Wake UP Interrupt Flag (WUS[1:0] bits)

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_WAKEUP_ON_ADDRESS
  - LL_LPUART_WAKEUP_ON_STARTBIT
  - LL_LPUART_WAKEUP_ON_RXNE

Reference Manual to LL API cross reference:

- CR3 WUS LL_LPUART_GetWKUPType

LL_LPUART_SetBaudRate

Function name

__STATIC_INLINE void LL_LPUART_SetBaudRate (USART_TypeDef * LPUARTx, uint32_t PeriphClk, uint32_t PrescalerValue, uint32_t BaudRate)

Function description

Configure LPUART BRR register for achieving expected Baud Rate value.

Parameters

- **LPUARTx:** LPUART Instance
- **PeriphClk:** Peripheral Clock
- **PrescalerValue:** This parameter can be one of the following values:
  - LL_LPUART_PRESCALER_DIV1
  - LL_LPUART_PRESCALER_DIV2
  - LL_LPUART_PRESCALER_DIV4
  - LL_LPUART_PRESCALER_DIV6
  - LL_LPUART_PRESCALER_DIV8
  - LL_LPUART_PRESCALER_DIV10
  - LL_LPUART_PRESCALER_DIV12
  - LL_LPUART_PRESCALER_DIV16
  - LL_LPUART_PRESCALER_DIV32
  - LL_LPUART_PRESCALER_DIV64
  - LL_LPUART_PRESCALER_DIV128
  - LL_LPUART_PRESCALER_DIV256
- **BaudRate:** Baud Rate

Return values

- **None:**

Notes

- Compute and set LPUARTDIV value in BRR Register (full BRR content) according to used Peripheral Clock and expected Baud Rate values
- Peripheral clock and Baud Rate values provided as function parameters should be valid (Baud rate value ! = 0).
- Provided that LPUARTx_BRR must be > = 0x300 and LPUART_BRR is 20-bit, a care should be taken when generating high baud rates using high PeriphClk values. PeriphClk must be in the range [3 x BaudRate, 4096 x BaudRate].

Reference Manual to LL API cross reference:

- BRR BRR LL_LPUART_SetBaudRate

LL_LPUART_GetBaudRate

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetBaudRate (const USART_TypeDef * LPUARTx, uint32_t PeriphClk, uint32_t PrescalerValue)

Function description

Return current Baud Rate value, according to LPUARTDIV present in BRR register (full BRR content), and to used Peripheral Clock values.

Parameters

- **LPUARTx:** LPUART Instance
- **PeriphClk:** Peripheral Clock
- **PrescalerValue:** This parameter can be one of the following values:
  - LL_LPUART_PRESCALER_DIV1
  - LL_LPUART_PRESCALER_DIV2
  - LL_LPUART_PRESCALER_DIV4
  - LL_LPUART_PRESCALER_DIV6
  - LL_LPUART_PRESCALER_DIV8
  - LL_LPUART_PRESCALER_DIV10
  - LL_LPUART_PRESCALER_DIV12
  - LL_LPUART_PRESCALER_DIV16
  - LL_LPUART_PRESCALER_DIV32
  - LL_LPUART_PRESCALER_DIV64
  - LL_LPUART_PRESCALER_DIV128
  - LL_LPUART_PRESCALER_DIV256

Return values

- **Baud:** Rate

Notes

- In case of non-initialized or invalid value stored in BRR register, value 0 will be returned.

Reference Manual to LL API cross reference:

- BRR BRR LL_LPUART_GetBaudRate

LL_LPUART_EnableHalfDuplex

Function name

__STATIC_INLINE void LL_LPUART_EnableHalfDuplex (USART_TypeDef * LPUARTx)

Function description

Enable Single Wire Half-Duplex mode.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 HDSEL LL_LPUART_EnableHalfDuplex

LL_LPUART_DisableHalfDuplex

Function name

__STATIC_INLINE void LL_LPUART_DisableHalfDuplex (USART_TypeDef * LPUARTx)

Function description

Disable Single Wire Half-Duplex mode.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 HDSEL LL_LPUART_DisableHalfDuplex

LL_LPUART_IsEnabledHalfDuplex

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledHalfDuplex (const USART_TypeDef * LPUARTx)

Function description

Indicate if Single Wire Half-Duplex mode is enabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 HDSEL LL_LPUART_IsEnabledHalfDuplex

LL_LPUART_SetDEDeassertionTime

Function name

__STATIC_INLINE void LL_LPUART_SetDEDeassertionTime (USART_TypeDef * LPUARTx, uint32_t Time)

Function description

Set DEDT (Driver Enable De-Assertion Time), Time value expressed on 5 bits ([4:0] bits).

Parameters

- **LPUARTx:** LPUART Instance
- **Time:** Value between Min_Data=0 and Max_Data=31

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 DEDT LL_LPUART_SetDEDeassertionTime

LL_LPUART_GetDEDeassertionTime

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetDEDeassertionTime (const USART_TypeDef * LPUARTx)

Function description

Return DEDT (Driver Enable De-Assertion Time)

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Time:** value expressed on 5 bits ([4:0] bits) : c

Reference Manual to LL API cross reference:

- CR1 DEDT LL_LPUART_GetDEDeassertionTime

LL_LPUART_SetDEAssertionTime

Function name

__STATIC_INLINE void LL_LPUART_SetDEAssertionTime (USART_TypeDef * LPUARTx, uint32_t Time)

Function description

Set DEAT (Driver Enable Assertion Time), Time value expressed on 5 bits ([4:0] bits).

Parameters

- **LPUARTx:** LPUART Instance
- **Time:** Value between Min_Data=0 and Max_Data=31

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 DEAT LL_LPUART_SetDEAssertionTime

LL_LPUART_GetDEAssertionTime

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetDEAssertionTime (const USART_TypeDef * LPUARTx)

Function description

Return DEAT (Driver Enable Assertion Time)

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Time:** value expressed on 5 bits ([4:0] bits) : Time Value between Min_Data=0 and Max_Data=31

Reference Manual to LL API cross reference:

- CR1 DEAT LL_LPUART_GetDEAssertionTime

LL_LPUART_EnableDEMode

Function name

__STATIC_INLINE void LL_LPUART_EnableDEMode (USART_TypeDef * LPUARTx)

Function description

Enable Driver Enable (DE) Mode.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DEM LL_LPUART_EnableDEMode

LL_LPUART_DisableDEMode

Function name

__STATIC_INLINE void LL_LPUART_DisableDEMode (USART_TypeDef * LPUARTx)

Function description

Disable Driver Enable (DE) Mode.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DEM LL_LPUART_DisableDEMode

LL_LPUART_IsEnabledDEMode

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledDEMode (const USART_TypeDef * LPUARTx)

Function description

Indicate if Driver Enable (DE) Mode is enabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 DEM LL_LPUART_IsEnabledDEMode

LL_LPUART_SetDESignalPolarity

Function name

__STATIC_INLINE void LL_LPUART_SetDESignalPolarity (USART_TypeDef * LPUARTx, uint32_t Polarity)

Function description

Select Driver Enable Polarity.

Parameters

- **LPUARTx:** LPUART Instance
- **Polarity:** This parameter can be one of the following values:
  - LL_LPUART_DE_POLARITY_HIGH
  - LL_LPUART_DE_POLARITY_LOW

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DEP LL_LPUART_SetDESignalPolarity

LL_LPUART_GetDESignalPolarity

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetDESignalPolarity (const USART_TypeDef * LPUARTx)

Function description

Return Driver Enable Polarity.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_DE_POLARITY_HIGH
  - LL_LPUART_DE_POLARITY_LOW

Reference Manual to LL API cross reference:

- CR3 DEP LL_LPUART_GetDESignalPolarity

LL_LPUART_IsActiveFlag_PE

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_PE (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Parity Error Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR PE LL_LPUART_IsActiveFlag_PE

LL_LPUART_IsActiveFlag_FE

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_FE (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Framing Error Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR FE LL_LPUART_IsActiveFlag_FE

LL_LPUART_IsActiveFlag_NE

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_NE (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Noise error detected Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR NE LL_LPUART_IsActiveFlag_NE

LL_LPUART_IsActiveFlag_ORE

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_ORE (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART OverRun Error Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR ORE LL_LPUART_IsActiveFlag_ORE

LL_LPUART_IsActiveFlag_IDLE

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_IDLE (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART IDLE line detected Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR IDLE LL_LPUART_IsActiveFlag_IDLE

LL_LPUART_IsActiveFlag_RXNE_RXFNE

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_RXNE_RXFNE (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Read Data Register or LPUART RX FIFO Not Empty Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR RXNE_RXFNE LL_LPUART_IsActiveFlag_RXNE_RXFNE

LL_LPUART_IsActiveFlag_TC

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_TC (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Transmission Complete Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TC LL_LPUART_IsActiveFlag_TC

LL_LPUART_IsActiveFlag_TXE_TXFNF

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_TXE_TXFNF (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Transmit Data Register Empty or LPUART TX FIFO Not Full Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TXE_TXFNF LL_LPUART_IsActiveFlag_TXE_TXFNF

LL_LPUART_IsActiveFlag_nCTS

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_nCTS (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART CTS interrupt Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR CTSIF LL_LPUART_IsActiveFlag_nCTS

LL_LPUART_IsActiveFlag_CTS

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_CTS (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART CTS Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR CTS LL_LPUART_IsActiveFlag_CTS

LL_LPUART_IsActiveFlag_BUSY

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_BUSY (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Busy Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR BUSY LL_LPUART_IsActiveFlag_BUSY

LL_LPUART_IsActiveFlag_CM

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_CM (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Character Match Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR CMF LL_LPUART_IsActiveFlag_CM

LL_LPUART_IsActiveFlag_SBK

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_SBK (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Send Break Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR SBKF LL_LPUART_IsActiveFlag_SBK

LL_LPUART_IsActiveFlag_RWU

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_RWU (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Receive Wake Up from mute mode Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR RWU LL_LPUART_IsActiveFlag_RWU

LL_LPUART_IsActiveFlag_WKUP

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_WKUP (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Wake Up from stop mode Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR WUF LL_LPUART_IsActiveFlag_WKUP

LL_LPUART_IsActiveFlag_TEACK

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_TEACK (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Transmit Enable Acknowledge Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TEACK LL_LPUART_IsActiveFlag_TEACK

LL_LPUART_IsActiveFlag_REACK

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_REACK (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Receive Enable Acknowledge Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR REACK LL_LPUART_IsActiveFlag_REACK

LL_LPUART_IsActiveFlag_TXFE

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_TXFE (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART TX FIFO Empty Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TXFE LL_LPUART_IsActiveFlag_TXFE

LL_LPUART_IsActiveFlag_RXFF

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_RXFF (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART RX FIFO Full Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR RXFF LL_LPUART_IsActiveFlag_RXFF

LL_LPUART_IsActiveFlag_TXFT

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_TXFT (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART TX FIFO Threshold Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TXFT LL_LPUART_IsActiveFlag_TXFT

LL_LPUART_IsActiveFlag_RXFT

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_RXFT (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART RX FIFO Threshold Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR RXFT LL_LPUART_IsActiveFlag_RXFT

LL_LPUART_ClearFlag_PE

Function name

__STATIC_INLINE void LL_LPUART_ClearFlag_PE (USART_TypeDef * LPUARTx)

Function description

Clear Parity Error Flag.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR PECF LL_LPUART_ClearFlag_PE

LL_LPUART_ClearFlag_FE

Function name

__STATIC_INLINE void LL_LPUART_ClearFlag_FE (USART_TypeDef * LPUARTx)

Function description

Clear Framing Error Flag.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR FECF LL_LPUART_ClearFlag_FE

LL_LPUART_ClearFlag_NE

Function name

__STATIC_INLINE void LL_LPUART_ClearFlag_NE (USART_TypeDef * LPUARTx)

Function description

Clear Noise detected Flag.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR NECF LL_LPUART_ClearFlag_NE

LL_LPUART_ClearFlag_ORE

Function name

__STATIC_INLINE void LL_LPUART_ClearFlag_ORE (USART_TypeDef * LPUARTx)

Function description

Clear OverRun Error Flag.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR ORECF LL_LPUART_ClearFlag_ORE

LL_LPUART_ClearFlag_IDLE

Function name

__STATIC_INLINE void LL_LPUART_ClearFlag_IDLE (USART_TypeDef * LPUARTx)

Function description

Clear IDLE line detected Flag.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR IDLECF LL_LPUART_ClearFlag_IDLE

LL_LPUART_ClearFlag_TC

Function name

__STATIC_INLINE void LL_LPUART_ClearFlag_TC (USART_TypeDef * LPUARTx)

Function description

Clear Transmission Complete Flag.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR TCCF LL_LPUART_ClearFlag_TC

LL_LPUART_ClearFlag_nCTS

Function name

__STATIC_INLINE void LL_LPUART_ClearFlag_nCTS (USART_TypeDef * LPUARTx)

Function description

Clear CTS Interrupt Flag.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR CTSCF LL_LPUART_ClearFlag_nCTS

LL_LPUART_ClearFlag_CM

Function name

__STATIC_INLINE void LL_LPUART_ClearFlag_CM (USART_TypeDef * LPUARTx)

Function description

Clear Character Match Flag.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR CMCF LL_LPUART_ClearFlag_CM

LL_LPUART_ClearFlag_WKUP

Function name

__STATIC_INLINE void LL_LPUART_ClearFlag_WKUP (USART_TypeDef * LPUARTx)

Function description

Clear Wake Up from stop mode Flag.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR WUCF LL_LPUART_ClearFlag_WKUP

LL_LPUART_EnableIT_IDLE

Function name

__STATIC_INLINE void LL_LPUART_EnableIT_IDLE (USART_TypeDef * LPUARTx)

Function description

Enable IDLE Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 IDLEIE LL_LPUART_EnableIT_IDLE

LL_LPUART_EnableIT_RXNE_RXFNE

Function name

__STATIC_INLINE void LL_LPUART_EnableIT_RXNE_RXFNE (USART_TypeDef * LPUARTx)

Function description

Enable RX Not Empty and RX FIFO Not Empty Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RXNEIE_RXFNEIE LL_LPUART_EnableIT_RXNE_RXFNE

LL_LPUART_EnableIT_TC

Function name

__STATIC_INLINE void LL_LPUART_EnableIT_TC (USART_TypeDef * LPUARTx)

Function description

Enable Transmission Complete Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TCIE LL_LPUART_EnableIT_TC

LL_LPUART_EnableIT_TXE_TXFNF

Function name

__STATIC_INLINE void LL_LPUART_EnableIT_TXE_TXFNF (USART_TypeDef * LPUARTx)

Function description

Enable TX Empty and TX FIFO Not Full Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TXEIE_TXFNFIE LL_LPUART_EnableIT_TXE_TXFNF

LL_LPUART_EnableIT_PE

Function name

__STATIC_INLINE void LL_LPUART_EnableIT_PE (USART_TypeDef * LPUARTx)

Function description

Enable Parity Error Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 PEIE LL_LPUART_EnableIT_PE

LL_LPUART_EnableIT_CM

Function name

__STATIC_INLINE void LL_LPUART_EnableIT_CM (USART_TypeDef * LPUARTx)

Function description

Enable Character Match Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 CMIE LL_LPUART_EnableIT_CM

LL_LPUART_EnableIT_TXFE

Function name

__STATIC_INLINE void LL_LPUART_EnableIT_TXFE (USART_TypeDef * LPUARTx)

Function description

Enable TX FIFO Empty Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TXFEIE LL_LPUART_EnableIT_TXFE

LL_LPUART_EnableIT_RXFF

Function name

__STATIC_INLINE void LL_LPUART_EnableIT_RXFF (USART_TypeDef * LPUARTx)

Function description

Enable RX FIFO Full Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RXFFIE LL_LPUART_EnableIT_RXFF

LL_LPUART_EnableIT_ERROR

Function name

__STATIC_INLINE void LL_LPUART_EnableIT_ERROR (USART_TypeDef * LPUARTx)

Function description

Enable Error Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Notes

- When set, Error Interrupt Enable Bit is enabling interrupt generation in case of a framing error, overrun error or noise flag (FE=1 or ORE=1 or NF=1 in the LPUARTx_ISR register). 0: Interrupt is inhibited1: An interrupt is generated when FE=1 or ORE=1 or NF=1 in the LPUARTx_ISR register.

Reference Manual to LL API cross reference:

- CR3 EIE LL_LPUART_EnableIT_ERROR

LL_LPUART_EnableIT_CTS

Function name

__STATIC_INLINE void LL_LPUART_EnableIT_CTS (USART_TypeDef * LPUARTx)

Function description

Enable CTS Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 CTSIE LL_LPUART_EnableIT_CTS

LL_LPUART_EnableIT_WKUP

Function name

__STATIC_INLINE void LL_LPUART_EnableIT_WKUP (USART_TypeDef * LPUARTx)

Function description

Enable Wake Up from Stop Mode Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 WUFIE LL_LPUART_EnableIT_WKUP

LL_LPUART_EnableIT_TXFT

Function name

__STATIC_INLINE void LL_LPUART_EnableIT_TXFT (USART_TypeDef * LPUARTx)

Function description

Enable TX FIFO Threshold Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 TXFTIE LL_LPUART_EnableIT_TXFT

LL_LPUART_EnableIT_RXFT

Function name

__STATIC_INLINE void LL_LPUART_EnableIT_RXFT (USART_TypeDef * LPUARTx)

Function description

Enable RX FIFO Threshold Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 RXFTIE LL_LPUART_EnableIT_RXFT

LL_LPUART_DisableIT_IDLE

Function name

__STATIC_INLINE void LL_LPUART_DisableIT_IDLE (USART_TypeDef * LPUARTx)

Function description

Disable IDLE Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 IDLEIE LL_LPUART_DisableIT_IDLE

LL_LPUART_DisableIT_RXNE_RXFNE

Function name

__STATIC_INLINE void LL_LPUART_DisableIT_RXNE_RXFNE (USART_TypeDef * LPUARTx)

Function description

Disable RX Not Empty and RX FIFO Not Empty Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RXNEIE_RXFNEIE LL_LPUART_DisableIT_RXNE_RXFNE

LL_LPUART_DisableIT_TC

Function name

__STATIC_INLINE void LL_LPUART_DisableIT_TC (USART_TypeDef * LPUARTx)

Function description

Disable Transmission Complete Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TCIE LL_LPUART_DisableIT_TC

LL_LPUART_DisableIT_TXE_TXFNF

Function name

__STATIC_INLINE void LL_LPUART_DisableIT_TXE_TXFNF (USART_TypeDef * LPUARTx)

Function description

Disable TX Empty and TX FIFO Not Full Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TXEIE_TXFNFIE LL_LPUART_DisableIT_TXE_TXFNF

LL_LPUART_DisableIT_PE

Function name

__STATIC_INLINE void LL_LPUART_DisableIT_PE (USART_TypeDef * LPUARTx)

Function description

Disable Parity Error Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 PEIE LL_LPUART_DisableIT_PE

LL_LPUART_DisableIT_CM

Function name

__STATIC_INLINE void LL_LPUART_DisableIT_CM (USART_TypeDef * LPUARTx)

Function description

Disable Character Match Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 CMIE LL_LPUART_DisableIT_CM

LL_LPUART_DisableIT_TXFE

Function name

__STATIC_INLINE void LL_LPUART_DisableIT_TXFE (USART_TypeDef * LPUARTx)

Function description

Disable TX FIFO Empty Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TXFEIE LL_LPUART_DisableIT_TXFE

LL_LPUART_DisableIT_RXFF

Function name

__STATIC_INLINE void LL_LPUART_DisableIT_RXFF (USART_TypeDef * LPUARTx)

Function description

Disable RX FIFO Full Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RXFFIE LL_LPUART_DisableIT_RXFF

LL_LPUART_DisableIT_ERROR

Function name

__STATIC_INLINE void LL_LPUART_DisableIT_ERROR (USART_TypeDef * LPUARTx)

Function description

Disable Error Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Notes

- When set, Error Interrupt Enable Bit is enabling interrupt generation in case of a framing error, overrun error or noise flag (FE=1 or ORE=1 or NF=1 in the LPUARTx_ISR register). 0: Interrupt is inhibited1: An interrupt is generated when FE=1 or ORE=1 or NF=1 in the LPUARTx_ISR register.

Reference Manual to LL API cross reference:

- CR3 EIE LL_LPUART_DisableIT_ERROR

LL_LPUART_DisableIT_CTS

Function name

__STATIC_INLINE void LL_LPUART_DisableIT_CTS (USART_TypeDef * LPUARTx)

Function description

Disable CTS Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 CTSIE LL_LPUART_DisableIT_CTS

LL_LPUART_DisableIT_WKUP

Function name

__STATIC_INLINE void LL_LPUART_DisableIT_WKUP (USART_TypeDef * LPUARTx)

Function description

Disable Wake Up from Stop Mode Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 WUFIE LL_LPUART_DisableIT_WKUP

LL_LPUART_DisableIT_TXFT

Function name

__STATIC_INLINE void LL_LPUART_DisableIT_TXFT (USART_TypeDef * LPUARTx)

Function description

Disable TX FIFO Threshold Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 TXFTIE LL_LPUART_DisableIT_TXFT

LL_LPUART_DisableIT_RXFT

Function name

__STATIC_INLINE void LL_LPUART_DisableIT_RXFT (USART_TypeDef * LPUARTx)

Function description

Disable RX FIFO Threshold Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 RXFTIE LL_LPUART_DisableIT_RXFT

LL_LPUART_IsEnabledIT_IDLE

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_IDLE (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART IDLE Interrupt source is enabled or disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 IDLEIE LL_LPUART_IsEnabledIT_IDLE

LL_LPUART_IsEnabledIT_RXNE_RXFNE

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_RXNE_RXFNE (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART RX Not Empty and LPUART RX FIFO Not Empty Interrupt is enabled or disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 RXNEIE_RXFNEIE LL_LPUART_IsEnabledIT_RXNE_RXFNE

LL_LPUART_IsEnabledIT_TC

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_TC (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Transmission Complete Interrupt is enabled or disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 TCIE LL_LPUART_IsEnabledIT_TC

LL_LPUART_IsEnabledIT_TXE_TXFNF

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_TXE_TXFNF (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART TX Empty and LPUART TX FIFO Not Full Interrupt is enabled or disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 TXEIE_TXFNFIE LL_LPUART_IsEnabledIT_TXE_TXFNF

LL_LPUART_IsEnabledIT_PE

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_PE (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Parity Error Interrupt is enabled or disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 PEIE LL_LPUART_IsEnabledIT_PE

LL_LPUART_IsEnabledIT_CM

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_CM (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Character Match Interrupt is enabled or disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 CMIE LL_LPUART_IsEnabledIT_CM

LL_LPUART_IsEnabledIT_TXFE

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_TXFE (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART TX FIFO Empty Interrupt is enabled or disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 TXFEIE LL_LPUART_IsEnabledIT_TXFE

LL_LPUART_IsEnabledIT_RXFF

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_RXFF (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART RX FIFO Full Interrupt is enabled or disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 RXFFIE LL_LPUART_IsEnabledIT_RXFF

LL_LPUART_IsEnabledIT_ERROR

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_ERROR (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Error Interrupt is enabled or disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 EIE LL_LPUART_IsEnabledIT_ERROR

LL_LPUART_IsEnabledIT_CTS

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_CTS (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART CTS Interrupt is enabled or disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 CTSIE LL_LPUART_IsEnabledIT_CTS

LL_LPUART_IsEnabledIT_WKUP

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_WKUP (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Wake Up from Stop Mode Interrupt is enabled or disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 WUFIE LL_LPUART_IsEnabledIT_WKUP

LL_LPUART_IsEnabledIT_TXFT

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_TXFT (const USART_TypeDef * LPUARTx)

Function description

Check if LPUART TX FIFO Threshold Interrupt is enabled or disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 TXFTIE LL_LPUART_IsEnabledIT_TXFT

LL_LPUART_IsEnabledIT_RXFT

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_RXFT (const USART_TypeDef * LPUARTx)

Function description

Check if LPUART RX FIFO Threshold Interrupt is enabled or disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 RXFTIE LL_LPUART_IsEnabledIT_RXFT

LL_LPUART_EnableDMAReq_RX

Function name

__STATIC_INLINE void LL_LPUART_EnableDMAReq_RX (USART_TypeDef * LPUARTx)

Function description

Enable DMA Mode for reception.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DMAR LL_LPUART_EnableDMAReq_RX

LL_LPUART_DisableDMAReq_RX

Function name

__STATIC_INLINE void LL_LPUART_DisableDMAReq_RX (USART_TypeDef * LPUARTx)

Function description

Disable DMA Mode for reception.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DMAR LL_LPUART_DisableDMAReq_RX

LL_LPUART_IsEnabledDMAReq_RX

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledDMAReq_RX (const USART_TypeDef * LPUARTx)

Function description

Check if DMA Mode is enabled for reception.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 DMAR LL_LPUART_IsEnabledDMAReq_RX

LL_LPUART_EnableDMAReq_TX

Function name

__STATIC_INLINE void LL_LPUART_EnableDMAReq_TX (USART_TypeDef * LPUARTx)

Function description

Enable DMA Mode for transmission.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DMAT LL_LPUART_EnableDMAReq_TX

LL_LPUART_DisableDMAReq_TX

Function name

__STATIC_INLINE void LL_LPUART_DisableDMAReq_TX (USART_TypeDef * LPUARTx)

Function description

Disable DMA Mode for transmission.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DMAT LL_LPUART_DisableDMAReq_TX

LL_LPUART_IsEnabledDMAReq_TX

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledDMAReq_TX (const USART_TypeDef * LPUARTx)

Function description

Check if DMA Mode is enabled for transmission.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 DMAT LL_LPUART_IsEnabledDMAReq_TX

LL_LPUART_EnableDMADeactOnRxErr

Function name

__STATIC_INLINE void LL_LPUART_EnableDMADeactOnRxErr (USART_TypeDef * LPUARTx)

Function description

Enable DMA Disabling on Reception Error.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DDRE LL_LPUART_EnableDMADeactOnRxErr

LL_LPUART_DisableDMADeactOnRxErr

Function name

__STATIC_INLINE void LL_LPUART_DisableDMADeactOnRxErr (USART_TypeDef * LPUARTx)

Function description

Disable DMA Disabling on Reception Error.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DDRE LL_LPUART_DisableDMADeactOnRxErr

LL_LPUART_IsEnabledDMADeactOnRxErr

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledDMADeactOnRxErr (const USART_TypeDef * LPUARTx)

Function description

Indicate if DMA Disabling on Reception Error is disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 DDRE LL_LPUART_IsEnabledDMADeactOnRxErr

LL_LPUART_DMA_GetRegAddr

Function name

__STATIC_INLINE uint32_t LL_LPUART_DMA_GetRegAddr (const USART_TypeDef * LPUARTx, uint32_t Direction)

Function description

Get the LPUART data register address used for DMA transfer.

Parameters

- **LPUARTx:** LPUART Instance
- **Direction:** This parameter can be one of the following values:
  - LL_LPUART_DMA_REG_DATA_TRANSMIT
  - LL_LPUART_DMA_REG_DATA_RECEIVE

Return values

- **Address:** of data register

Reference Manual to LL API cross reference:

- RDR RDR LL_LPUART_DMA_GetRegAddr
-
- TDR TDR LL_LPUART_DMA_GetRegAddr

LL_LPUART_ReceiveData8

Function name

__STATIC_INLINE uint8_t LL_LPUART_ReceiveData8 (const USART_TypeDef * LPUARTx)

Function description

Read Receiver Data register (Receive Data value, 8 bits)

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Time:** Value between Min_Data=0x00 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- RDR RDR LL_LPUART_ReceiveData8

LL_LPUART_ReceiveData9

Function name

__STATIC_INLINE uint16_t LL_LPUART_ReceiveData9 (const USART_TypeDef * LPUARTx)

Function description

Read Receiver Data register (Receive Data value, 9 bits)

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Time:** Value between Min_Data=0x00 and Max_Data=0x1FF

Reference Manual to LL API cross reference:

- RDR RDR LL_LPUART_ReceiveData9

LL_LPUART_TransmitData8

Function name

__STATIC_INLINE void LL_LPUART_TransmitData8 (USART_TypeDef * LPUARTx, uint8_t Value)

Function description

Write in Transmitter Data Register (Transmit Data value, 8 bits)

Parameters

- **LPUARTx:** LPUART Instance
- **Value:** between Min_Data=0x00 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- TDR TDR LL_LPUART_TransmitData8

LL_LPUART_TransmitData9

Function name

__STATIC_INLINE void LL_LPUART_TransmitData9 (USART_TypeDef * LPUARTx, uint16_t Value)

Function description

Write in Transmitter Data Register (Transmit Data value, 9 bits)

Parameters

- **LPUARTx:** LPUART Instance
- **Value:** between Min_Data=0x00 and Max_Data=0x1FF

Return values

- **None:**

Reference Manual to LL API cross reference:

- TDR TDR LL_LPUART_TransmitData9

LL_LPUART_RequestBreakSending

Function name

__STATIC_INLINE void LL_LPUART_RequestBreakSending (USART_TypeDef * LPUARTx)

Function description

Request Break sending.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RQR SBKRQ LL_LPUART_RequestBreakSending

LL_LPUART_RequestEnterMuteMode

Function name

__STATIC_INLINE void LL_LPUART_RequestEnterMuteMode (USART_TypeDef * LPUARTx)

Function description

Put LPUART in mute mode and set the RWU flag.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RQR MMRQ LL_LPUART_RequestEnterMuteMode

LL_LPUART_RequestRxDataFlush

Function name

__STATIC_INLINE void LL_LPUART_RequestRxDataFlush (USART_TypeDef * LPUARTx)

Function description

Request a Receive Data and FIFO flush.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Notes

- Allows to discard the received data without reading them, and avoid an overrun condition.

Reference Manual to LL API cross reference:

- RQR RXFRQ LL_LPUART_RequestRxDataFlush

LL_LPUART_RequestTxDataFlush

Function name

__STATIC_INLINE void LL_LPUART_RequestTxDataFlush (USART_TypeDef * LPUARTx)

Function description

Request a Transmit data FIFO flush.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Notes

- TXFRQ bit is set to flush the whole FIFO when FIFO mode is enabled. This also sets the flag TXFE (TXFIFO empty bit in the LPUART_ISR register).
- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- RQR TXFRQ LL_LPUART_RequestTxDataFlush

LL_LPUART_DeInit

Function name

ErrorStatus LL_LPUART_DeInit (const USART_TypeDef * LPUARTx)

Function description

De-initialize LPUART registers (Registers restored to their default values).

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: LPUART registers are de-initialized
  - ERROR: not applicable

LL_LPUART_Init

Function name

ErrorStatus LL_LPUART_Init (USART_TypeDef * LPUARTx, const LL_LPUART_InitTypeDef * LPUART_InitStruct)

Function description

Initialize LPUART registers according to the specified parameters in LPUART_InitStruct.

Parameters

- **LPUARTx:** LPUART Instance
- **LPUART_InitStruct:** pointer to a LL_LPUART_InitTypeDef structure that contains the configuration information for the specified LPUART peripheral.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: LPUART registers are initialized according to LPUART_InitStruct content
  - ERROR: Problem occurred during LPUART Registers initialization

Notes

- As some bits in LPUART configuration registers can only be written when the LPUART is disabled (USART_CR1_UE bit =0), LPUART Peripheral should be in disabled state prior calling this function. Otherwise, ERROR result will be returned.
- Baud rate value stored in LPUART_InitStruct BaudRate field, should be valid (different from 0).

LL_LPUART_StructInit

Function name

void LL_LPUART_StructInit (LL_LPUART_InitTypeDef * LPUART_InitStruct)

Function description

Set each LL_LPUART_InitTypeDef field to default value.

Parameters

- **LPUART_InitStruct:** pointer to a LL_LPUART_InitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

99.3 LPUART Firmware driver defines

The following section lists the various define and macros of the module.

99.3.1 LPUART

LPUART

Address Length Detection

LL_LPUART_ADDRESS_DETECT_4B

4-bit address detection method selected

LL_LPUART_ADDRESS_DETECT_7B

7-bit address detection (in 8-bit data mode) method selected

Binary Data Inversion

LL_LPUART_BINARY_LOGIC_POSITIVE

Logical data from the data register are send/received in positive/direct logic. (1=H, 0=L)

LL_LPUART_BINARY_LOGIC_NEGATIVE

Logical data from the data register are send/received in negative/inverse logic. (1=L, 0=H). The parity bit is also inverted.

Bit Order

LL_LPUART_BITORDER_LSBFIRST

data is transmitted/received with data bit 0 first, following the start bit

LL_LPUART_BITORDER_MSBFIRST

data is transmitted/received with the MSB first, following the start bit

Clear Flags Defines

LL_LPUART_ICR_PECF

Parity error clear flag

LL_LPUART_ICR_FECF

Framing error clear flag

LL_LPUART_ICR_NCF

Noise error detected clear flag

LL_LPUART_ICR_ORECF

Overrun error clear flag

LL_LPUART_ICR_IDLECF

Idle line detected clear flag

LL_LPUART_ICR_TCCF

Transmission complete clear flag

LL_LPUART_ICR_CTSCF

CTS clear flag

LL_LPUART_ICR_CMCF

Character match clear flag

LL_LPUART_ICR_WUCF

Wakeup from Stop mode clear flag

Datawidth

LL_LPUART_DATAWIDTH_7B

7 bits word length : Start bit, 7 data bits, n stop bits

LL_LPUART_DATAWIDTH_8B

8 bits word length : Start bit, 8 data bits, n stop bits

LL_LPUART_DATAWIDTH_9B

9 bits word length : Start bit, 9 data bits, n stop bits

Driver Enable Polarity

LL_LPUART_DE_POLARITY_HIGH

DE signal is active high

LL_LPUART_DE_POLARITY_LOW

DE signal is active low

Direction

LL_LPUART_DIRECTION_NONE

Transmitter and Receiver are disabled

LL_LPUART_DIRECTION_RX

Transmitter is disabled and Receiver is enabled

LL_LPUART_DIRECTION_TX

Transmitter is enabled and Receiver is disabled

LL_LPUART_DIRECTION_TX_RX

Transmitter and Receiver are enabled

DMA Register Data

LL_LPUART_DMA_REG_DATA_TRANSMIT

Get address of data register used for transmission

LL_LPUART_DMA_REG_DATA_RECEIVE

Get address of data register used for reception

FIFO Threshold

LL_LPUART_FIFOTHRESHOLD_1_8

FIFO reaches 1/8 of its depth

LL_LPUART_FIFOTHRESHOLD_1_4

FIFO reaches 1/4 of its depth

LL_LPUART_FIFOTHRESHOLD_1_2

FIFO reaches 1/2 of its depth

LL_LPUART_FIFOTHRESHOLD_3_4

FIFO reaches 3/4 of its depth

LL_LPUART_FIFOTHRESHOLD_7_8

FIFO reaches 7/8 of its depth

LL_LPUART_FIFOTHRESHOLD_8_8

FIFO becomes empty for TX and full for RX

Get Flags Defines

LL_LPUART_ISR_PE

Parity error flag

LL_LPUART_ISR_FE

Framing error flag

LL_LPUART_ISR_NE

Noise detected flag

LL_LPUART_ISR_ORE

Overrun error flag

LL_LPUART_ISR_IDLE

Idle line detected flag

LL_LPUART_ISR_RXNE_RXFNE

Read data register or RX FIFO not empty flag

LL_LPUART_ISR_TC

Transmission complete flag

LL_LPUART_ISR_TXE_TXFNF

Transmit data register empty or TX FIFO Not Full flag

LL_LPUART_ISR_CTSIF

CTS interrupt flag

LL_LPUART_ISR_CTS

CTS flag

LL_LPUART_ISR_BUSY

Busy flag

LL_LPUART_ISR_CMF

Character match flag

LL_LPUART_ISR_SBKF

Send break flag

LL_LPUART_ISR_RWU

Receiver wakeup from Mute mode flag

LL_LPUART_ISR_WUF

Wakeup from Stop mode flag

LL_LPUART_ISR_TEACK

Transmit enable acknowledge flag

LL_LPUART_ISR_REACK

Receive enable acknowledge flag

LL_LPUART_ISR_TXFE

TX FIFO empty flag

LL_LPUART_ISR_RXFF

RX FIFO full flag

LL_LPUART_ISR_RXFT

RX FIFO threshold flag

LL_LPUART_ISR_TXFT

TX FIFO threshold flag

Hardware Control

LL_LPUART_HWCONTROL_NONE

CTS and RTS hardware flow control disabled

LL_LPUART_HWCONTROL_RTS

RTS output enabled, data is only requested when there is space in the receive buffer

LL_LPUART_HWCONTROL_CTS

CTS mode enabled, data is only transmitted when the nCTS input is asserted (tied to 0)

LL_LPUART_HWCONTROL_RTS_CTS

CTS and RTS hardware flow control enabled

IT Defines

LL_LPUART_CR1_IDLEIE

IDLE interrupt enable

LL_LPUART_CR1_RXNEIE_RXFNEIE

Read data register and RXFIFO not empty interrupt enable

LL_LPUART_CR1_TCIE

Transmission complete interrupt enable

LL_LPUART_CR1_TXEIE_TXFNFIE

Transmit data register empty and TX FIFO not full interrupt enable

LL_LPUART_CR1_PEIE

Parity error

LL_LPUART_CR1_CMIE

Character match interrupt enable

LL_LPUART_CR1_TXFEIE

TX FIFO empty interrupt enable

LL_LPUART_CR1_RXFFIE

RX FIFO full interrupt enable

LL_LPUART_CR3_EIE

Error interrupt enable

LL_LPUART_CR3_CTSIE

CTS interrupt enable

LL_LPUART_CR3_WUFIE

Wakeup from Stop mode interrupt enable

LL_LPUART_CR3_TXFTIE

TX FIFO threshold interrupt enable

LL_LPUART_CR3_RXFTIE

RX FIFO threshold interrupt enable

Parity Control

LL_LPUART_PARITY_NONE

Parity control disabled

LL_LPUART_PARITY_EVEN

Parity control enabled and Even Parity is selected

LL_LPUART_PARITY_ODD

Parity control enabled and Odd Parity is selected

Clock Source Prescaler

LL_LPUART_PRESCALER_DIV1

Input clock not divided

LL_LPUART_PRESCALER_DIV2

Input clock divided by 2

LL_LPUART_PRESCALER_DIV4

Input clock divided by 4

LL_LPUART_PRESCALER_DIV6

Input clock divided by 6

LL_LPUART_PRESCALER_DIV8

Input clock divided by 8

LL_LPUART_PRESCALER_DIV10

Input clock divided by 10

LL_LPUART_PRESCALER_DIV12

Input clock divided by 12

LL_LPUART_PRESCALER_DIV16

Input clock divided by 16

LL_LPUART_PRESCALER_DIV32

Input clock divided by 32

LL_LPUART_PRESCALER_DIV64

Input clock divided by 64

LL_LPUART_PRESCALER_DIV128

Input clock divided by 128

LL_LPUART_PRESCALER_DIV256

Input clock divided by 256

RX Pin Active Level Inversion

LL_LPUART_RXPIN_LEVEL_STANDARD

RX pin signal works using the standard logic levels

LL_LPUART_RXPIN_LEVEL_INVERTED

RX pin signal values are inverted.

Stop Bits

LL_LPUART_STOPBITS_1

1 stop bit

LL_LPUART_STOPBITS_2

2 stop bits

TX Pin Active Level Inversion

LL_LPUART_TXPIN_LEVEL_STANDARD

TX pin signal works using the standard logic levels

LL_LPUART_TXPIN_LEVEL_INVERTED

TX pin signal values are inverted.

TX RX Pins Swap

LL_LPUART_TXRX_STANDARD

TX/RX pins are used as defined in standard pinout

LL_LPUART_TXRX_SWAPPED

TX and RX pins functions are swapped.

Wakeup

LL_LPUART_WAKEUP_IDLELINE

LPUART wake up from Mute mode on Idle Line

LL_LPUART_WAKEUP_ADDRESSMARK

LPUART wake up from Mute mode on Address Mark

Wakeup Activation

LL_LPUART_WAKEUP_ON_ADDRESS

Wake up active on address match

LL_LPUART_WAKEUP_ON_STARTBIT

Wake up active on Start bit detection

LL_LPUART_WAKEUP_ON_RXNE

Wake up active on RXNE

FLAG_Management

LL_LPUART_IsActiveFlag_RXNE

LL_LPUART_IsActiveFlag_TXE

IT_Management

LL_LPUART_EnableIT_RXNE

LL_LPUART_EnableIT_TXE

LL_LPUART_DisableIT_RXNE

LL_LPUART_DisableIT_TXE

LL_LPUART_IsEnabledIT_RXNE

LL_LPUART_IsEnabledIT_TXE

Helper Macros

__LL_LPUART_DIV

Description:

- Compute LPUARTDIV value according to Peripheral Clock and expected Baud Rate (20-bit value of LPUARTDIV is returned)

Parameters:

- __PERIPHCLK__: Peripheral Clock frequency used for LPUART Instance
- __PRESCALER__: This parameter can be one of the following values:
  - LL_LPUART_PRESCALER_DIV1
  - LL_LPUART_PRESCALER_DIV2
  - LL_LPUART_PRESCALER_DIV4
  - LL_LPUART_PRESCALER_DIV6
  - LL_LPUART_PRESCALER_DIV8
  - LL_LPUART_PRESCALER_DIV10
  - LL_LPUART_PRESCALER_DIV12
  - LL_LPUART_PRESCALER_DIV16
  - LL_LPUART_PRESCALER_DIV32
  - LL_LPUART_PRESCALER_DIV64
  - LL_LPUART_PRESCALER_DIV128
  - LL_LPUART_PRESCALER_DIV256
- __BAUDRATE__: Baud Rate value to achieve

Return value:

- LPUARTDIV: value to be used for BRR register filling

Common Write and read registers Macros

LL_LPUART_WriteReg

Description:

- Write a value in LPUART register.

Parameters:

- __INSTANCE__: LPUART Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_LPUART_ReadReg

Description:

- Read a value in LPUART register.

Parameters:

- __INSTANCE__: LPUART Instance
- __REG__: Register to be read

Return value:

- Register: value

100 LL PKA Generic Driver

100.1 PKA Firmware driver registers structures

100.1.1 LL_PKA_InitTypeDef

LL_PKA_InitTypeDef is defined in the stm32n6xx_ll_pka.h

Data Fields

- ***uint32_t Mode***

Field Documentation

- ***uint32_t LL_PKA_InitTypeDef::Mode***
  Specifies the PKA operation mode. This parameter can be a value of ***PKA_LL_EC_MODE***.This feature can be modified afterwards using unitary function **LL_PKA_SetMode()**.

100.2 PKA Firmware driver API description

The following section lists the various functions of the PKA library.

100.2.1 Detailed description of functions

LL_PKA_Config

Function name

__STATIC_INLINE void LL_PKA_Config (PKA_TypeDef * PKAx, uint32_t Mode)

Function description

Configure PKA peripheral.

Parameters

- **PKAx:** PKA Instance.
- **Mode:** This parameter can be one of the following values:
  - LL_PKA_MODE_MONTGOMERY_PARAM
  - LL_PKA_MODE_MODULAR_EXP
  - LL_PKA_MODE_ECDSA_SIGNATURE
  - LL_PKA_MODE_ECDSA_VERIFICATION
  - LL_PKA_MODE_POINT_CHECK
  - LL_PKA_MODE_RSA_CRT_EXP
  - LL_PKA_MODE_MODULAR_INV
  - LL_PKA_MODE_ARITHMETIC_ADD
  - LL_PKA_MODE_ARITHMETIC_SUB
  - LL_PKA_MODE_ARITHMETIC_MUL
  - LL_PKA_MODE_COMPARISON
  - LL_PKA_MODE_MODULAR_REDUC
  - LL_PKA_MODE_MODULAR_ADD
  - LL_PKA_MODE_MODULAR_SUB
  - LL_PKA_MODE_MONTGOMERY_MUL
  - LL_PKA_MODE_MODULAR_EXP_PROTECT
  - LL_PKA_MODE_DOUBLE_BASE_LADDER
  - LL_PKA_MODE_ECC_PROJECTIVE_AFF
  - LL_PKA_MODE_ECC_COMPLETE_ADD
  - LL_PKA_MODE_ECC_MUL
  - LL_PKA_MODE_MODULAR_EXP_FAST

Reference Manual to LL API cross reference:

- CR MODE LL_PKA_Config

LL_PKA_Enable

Function name

__STATIC_INLINE void LL_PKA_Enable (PKA_TypeDef * PKAx)

Function description

Enable PKA peripheral.

Parameters

- **PKAx:** PKA Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR EN LL_PKA_Enable

LL_PKA_Disable

Function name

__STATIC_INLINE void LL_PKA_Disable (PKA_TypeDef * PKAx)

Function description

Disable PKA peripheral.

Parameters

- **PKAx:** PKA Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR EN LL_PKA_Disable

LL_PKA_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_PKA_IsEnabled (const PKA_TypeDef * PKAx)

Function description

Check if the PKA peripheral is enabled or disabled.

Parameters

- **PKAx:** PKA Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR EN LL_PKA_IsEnabled

LL_PKA_SetMode

Function name

__STATIC_INLINE void LL_PKA_SetMode (PKA_TypeDef * PKAx, uint32_t Mode)

Function description

Set PKA operating mode.

Parameters

- **PKAx:** PKA Instance.
- **Mode:** This parameter can be one of the following values:
  - LL_PKA_MODE_MONTGOMERY_PARAM
  - LL_PKA_MODE_MODULAR_EXP
  - LL_PKA_MODE_ECDSA_SIGNATURE
  - LL_PKA_MODE_ECDSA_VERIFICATION
  - LL_PKA_MODE_POINT_CHECK
  - LL_PKA_MODE_RSA_CRT_EXP
  - LL_PKA_MODE_MODULAR_INV
  - LL_PKA_MODE_ARITHMETIC_ADD
  - LL_PKA_MODE_ARITHMETIC_SUB
  - LL_PKA_MODE_ARITHMETIC_MUL
  - LL_PKA_MODE_COMPARISON
  - LL_PKA_MODE_MODULAR_REDUC
  - LL_PKA_MODE_MODULAR_ADD
  - LL_PKA_MODE_MODULAR_SUB
  - LL_PKA_MODE_MONTGOMERY_MUL
  - LL_PKA_MODE_MODULAR_EXP_PROTECT
  - LL_PKA_MODE_DOUBLE_BASE_LADDER
  - LL_PKA_MODE_ECC_PROJECTIVE_AFF
  - LL_PKA_MODE_ECC_COMPLETE_ADD
  - LL_PKA_MODE_ECC_MUL
  - LL_PKA_MODE_MODULAR_EXP_FAST

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR MODE LL_PKA_SetMode

LL_PKA_GetMode

Function name

__STATIC_INLINE uint32_t LL_PKA_GetMode (const PKA_TypeDef * PKAx)

Function description

Get PKA operating mode.

Parameters

- **PKAx:** PKA Instance.

Return values

- **Returned:** value can be one of the following values:
  - LL_PKA_MODE_MONTGOMERY_PARAM
  - LL_PKA_MODE_MODULAR_EXP
  - LL_PKA_MODE_ECDSA_SIGNATURE
  - LL_PKA_MODE_ECDSA_VERIFICATION
  - LL_PKA_MODE_POINT_CHECK
  - LL_PKA_MODE_RSA_CRT_EXP
  - LL_PKA_MODE_MODULAR_INV
  - LL_PKA_MODE_ARITHMETIC_ADD
  - LL_PKA_MODE_ARITHMETIC_SUB
  - LL_PKA_MODE_ARITHMETIC_MUL
  - LL_PKA_MODE_COMPARISON
  - LL_PKA_MODE_MODULAR_REDUC
  - LL_PKA_MODE_MODULAR_ADD
  - LL_PKA_MODE_MODULAR_SUB
  - LL_PKA_MODE_MONTGOMERY_MUL
  - LL_PKA_MODE_MODULAR_EXP_PROTECT
  - LL_PKA_MODE_DOUBLE_BASE_LADDER
  - LL_PKA_MODE_ECC_PROJECTIVE_AFF
  - LL_PKA_MODE_ECC_COMPLETE_ADD
  - LL_PKA_MODE_ECC_MUL
  - LL_PKA_MODE_MODULAR_EXP_FAST

Reference Manual to LL API cross reference:

- CR MODE LL_PKA_GetMode

LL_PKA_Start

Function name

__STATIC_INLINE void LL_PKA_Start (PKA_TypeDef * PKAx)

Function description

Start the operation selected using LL_PKA_SetMode.

Parameters

- **PKAx:** PKA Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR START LL_PKA_Start

LL_PKA_EnableIT_ADDRERR

Function name

__STATIC_INLINE void LL_PKA_EnableIT_ADDRERR (PKA_TypeDef * PKAx)

Function description

Enable address error interrupt.

Parameters

- **PKAx:** PKA Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR ADDRERRIE LL_PKA_EnableIT_ADDRERR

LL_PKA_EnableIT_RAMERR

Function name

__STATIC_INLINE void LL_PKA_EnableIT_RAMERR (PKA_TypeDef * PKAx)

Function description

Enable RAM error interrupt.

Parameters

- **PKAx:** PKA Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR RAMERRIE LL_PKA_EnableIT_RAMERR

LL_PKA_EnableIT_OPERR

Function name

__STATIC_INLINE void LL_PKA_EnableIT_OPERR (PKA_TypeDef * PKAx)

Function description

Enable OPERATION error interrupt.

Parameters

- **PKAx:** PKA Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR OPERRIE LL_PKA_EnableIT_OPERR

LL_PKA_EnableIT_PROCEND

Function name

__STATIC_INLINE void LL_PKA_EnableIT_PROCEND (PKA_TypeDef * PKAx)

Function description

Enable end of operation interrupt.

Parameters

- **PKAx:** PKA Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR PROCENDIE LL_PKA_EnableIT_PROCEND

LL_PKA_DisableIT_ADDERR

Function name

__STATIC_INLINE void LL_PKA_DisableIT_ADDERR (PKA_TypeDef * PKAx)

Function description

Disable address error interrupt.

Parameters

- **PKAx:** PKA Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR ADDRERRIE LL_PKA_DisableIT_ADDERR

LL_PKA_DisableIT_RAMERR

Function name

__STATIC_INLINE void LL_PKA_DisableIT_RAMERR (PKA_TypeDef * PKAx)

Function description

Disable RAM error interrupt.

Parameters

- **PKAx:** PKA Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR RAMERRIE LL_PKA_DisableIT_RAMERR

LL_PKA_DisableIT_PROCEND

Function name

__STATIC_INLINE void LL_PKA_DisableIT_PROCEND (PKA_TypeDef * PKAx)

Function description

Disable End of operation interrupt.

Parameters

- **PKAx:** PKA Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR PROCENDIE LL_PKA_DisableIT_PROCEND

LL_PKA_DisableIT_OPERR

Function name

__STATIC_INLINE void LL_PKA_DisableIT_OPERR (PKA_TypeDef * PKAx)

Function description

Disable OPERATION error interrupt.

Parameters

- **PKAx:** PKA Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR OPERRIE LL_PKA_EnableIT_OPERR

LL_PKA_IsEnabledIT_ADDRERR

Function name

__STATIC_INLINE uint32_t LL_PKA_IsEnabledIT_ADDRERR (const PKA_TypeDef * PKAx)

Function description

Check if address error interrupt is enabled.

Parameters

- **PKAx:** PKA Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR ADDRERRIE LL_PKA_IsEnabledIT_ADDRERR

LL_PKA_IsEnabledIT_RAMERR

Function name

__STATIC_INLINE uint32_t LL_PKA_IsEnabledIT_RAMERR (const PKA_TypeDef * PKAx)

Function description

Check if RAM error interrupt is enabled.

Parameters

- **PKAx:** PKA Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR RAMERRIE LL_PKA_IsEnabledIT_RAMERR

LL_PKA_IsEnabledIT_OPERR

Function name

__STATIC_INLINE uint32_t LL_PKA_IsEnabledIT_OPERR (const PKA_TypeDef * PKAx)

Function description

Check if OPERATION error interrupt is enabled.

Parameters

- **PKAx:** PKA Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR OPERRIE LL_PKA_IsEnabledIT_OPERR

LL_PKA_IsEnabledIT_PROCEND

Function name

__STATIC_INLINE uint32_t LL_PKA_IsEnabledIT_PROCEND (const PKA_TypeDef * PKAx)

Function description

Check if end of operation interrupt is enabled.

Parameters

- **PKAx:** PKA Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR PROCENDIE LL_PKA_IsEnabledIT_PROCEND

LL_PKA_IsActiveFlag_ADDRERR

Function name

__STATIC_INLINE uint32_t LL_PKA_IsActiveFlag_ADDRERR (const PKA_TypeDef * PKAx)

Function description

Get PKA address error flag.

Parameters

- **PKAx:** PKA Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR ADDRERRF LL_PKA_IsActiveFlag_ADDRERR

LL_PKA_IsActiveFlag_RAMERR

Function name

__STATIC_INLINE uint32_t LL_PKA_IsActiveFlag_RAMERR (const PKA_TypeDef * PKAx)

Function description

Get PKA RAM error flag.

Parameters

- **PKAx:** PKA Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR RAMERRF LL_PKA_IsActiveFlag_RAMERR

LL_PKA_IsActiveFlag_OPERR

Function name

__STATIC_INLINE uint32_t LL_PKA_IsActiveFlag_OPERR (const PKA_TypeDef * PKAx)

Function description

Get PKA OPERATION error flag.

Parameters

- **PKAx:** PKA Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR OPERRF LL_PKA_IsActiveFlag_OPERR

LL_PKA_IsActiveFlag_PROCEND

Function name

__STATIC_INLINE uint32_t LL_PKA_IsActiveFlag_PROCEND (const PKA_TypeDef * PKAx)

Function description

Get PKA end of operation flag.

Parameters

- **PKAx:** PKA Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR PROCENDF LL_PKA_IsActiveFlag_PROCEND

LL_PKA_IsActiveFlag_BUSY

Function name

__STATIC_INLINE uint32_t LL_PKA_IsActiveFlag_BUSY (const PKA_TypeDef * PKAx)

Function description

Get PKA busy flag.

Parameters

- **PKAx:** PKA Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR BUSY LL_PKA_IsActiveFlag_BUSY

LL_PKA_ClearFlag_ADDERR

Function name

__STATIC_INLINE void LL_PKA_ClearFlag_ADDERR (PKA_TypeDef * PKAx)

Function description

Clear PKA address error flag.

Parameters

- **PKAx:** PKA Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CLRFR ADDRERRFC LL_PKA_ClearFlag_ADDERR

LL_PKA_ClearFlag_RAMERR

Function name

__STATIC_INLINE void LL_PKA_ClearFlag_RAMERR (PKA_TypeDef * PKAx)

Function description

Clear PKA RAM error flag.

Parameters

- **PKAx:** PKA Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CLRFR RAMERRFC LL_PKA_ClearFlag_RAMERR

LL_PKA_ClearFlag_OPERR

Function name

__STATIC_INLINE void LL_PKA_ClearFlag_OPERR (PKA_TypeDef * PKAx)

Function description

Clear PKA OPERATION error flag.

Parameters

- **PKAx:** PKA Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CLRFR OPERRFC LL_PKA_ClearFlag_OPERR

LL_PKA_ClearFlag_PROCEND

Function name

__STATIC_INLINE void LL_PKA_ClearFlag_PROCEND (PKA_TypeDef * PKAx)

Function description

Clear PKA end of operation flag.

Parameters

- **PKAx:** PKA Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CLRFR PROCENDFC LL_PKA_ClearFlag_PROCEND

LL_PKA_DeInit

Function name

ErrorStatus LL_PKA_DeInit (const PKA_TypeDef * PKAx)

Function description

De-initialize PKA registers (Registers restored to their default values).

Parameters

- **PKAx:** PKA Instance.

Return values

- **ErrorStatus:**
  - SUCCESS: PKA registers are de-initialized
  - ERROR: PKA registers are not de-initialized

LL_PKA_Init

Function name

ErrorStatus LL_PKA_Init (PKA_TypeDef * PKAx, LL_PKA_InitTypeDef * PKA_InitStruct)

Function description

Initialize PKA registers according to the specified parameters in PKA_InitStruct.

Parameters

- **PKAx:** PKA Instance.
- **PKA_InitStruct:** pointer to a LL_PKA_InitTypeDef structure that contains the configuration information for the specified PKA peripheral.

Return values

- **ErrorStatus:**
  - SUCCESS: PKA registers are initialized according to PKA_InitStruct content
  - ERROR: Not applicable

LL_PKA_StructInit

Function name

void LL_PKA_StructInit (LL_PKA_InitTypeDef * PKA_InitStruct)

Function description

Set each LL_PKA_InitTypeDef field to default value.

Parameters

- **PKA_InitStruct:** pointer to a LL_PKA_InitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

100.3 PKA Firmware driver defines

The following section lists the various define and macros of the module.

100.3.1 PKA

PKA

Get Flags Defines

LL_PKA_SR_ADDRERRF

LL_PKA_SR_RAMERRF

LL_PKA_SR_PROCENDF

LL_PKA_SR_BUSY

LL_PKA_SR_INITOK

LL_PKA_SR_OPERRF

IT Defines

LL_PKA_CR_ADDRERRIE

LL_PKA_CR_RAMERRIE

LL_PKA_CR_PROCENDIE

LL_PKA_CLRFR_PROCENDFC

LL_PKA_CLRFR_RAMERRFC

LL_PKA_CLRFR_ADDRERRFC

LL_PKA_CR_OPERRIE

LL_PKA_CLRFR_OPERRFC

Operation Mode

LL_PKA_MODE_MODULAR_EXP
: modular exponentiation

LL_PKA_MODE_MONTGOMERY_PARAM
: Compute Montgomery parameter only

LL_PKA_MODE_MODULAR_EXP_FAST
: modular exponentiation fast mode

LL_PKA_MODE_MODULAR_EXP_PROTECT
: modular exponentiation protect mode

LL_PKA_MODE_ECC_MUL
: compute ECC kP operation

LL_PKA_MODE_ECC_COMPLETE_ADD
: ECC complete addition

LL_PKA_MODE_ECDSA_SIGNATURE
: ECDSA signature

LL_PKA_MODE_ECDSA_VERIFICATION

ECDSA verification

LL_PKA_MODE_POINT_CHECK

Point check

LL_PKA_MODE_RSA_CRT_EXP

RSA CRT exponentiation

LL_PKA_MODE_MODULAR_INV

Modular inversion

LL_PKA_MODE_ARITHMETIC_ADD

Arithmetic addition

LL_PKA_MODE_ARITHMETIC_SUB

Arithmetic subtraction

LL_PKA_MODE_ARITHMETIC_MUL

Arithmetic multiplication

LL_PKA_MODE_COMPARISON

Comparison

LL_PKA_MODE_MODULAR_REDUC

Modular reduction

LL_PKA_MODE_MODULAR_ADD

Modular addition

LL_PKA_MODE_MODULAR_SUB

Modular subtraction

LL_PKA_MODE_MONTGOMERY_MUL

Montgomery multiplication

LL_PKA_MODE_DOUBLE_BASE_LADDER

Double base ladder

LL_PKA_MODE_ECC_PROJECTIVE_AFF

ECC projective to affine

Common Write and read registers Macros

LL_PKA_WriteReg

Description:

- Write a value in PKA register.

Parameters:

- __INSTANCE__: PKA Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_PKA_ReadReg

Description:

- Read a value in PKA register.

Parameters:

- __INSTANCE__: PKA Instance
- __REG__: Register to be read

Return value:

- Register: value

101 LL RCC Generic Driver

101.1 RCC Firmware driver registers structures

101.1.1 LL_RCC_ClocksTypeDef

LL_RCC_ClocksTypeDef is defined in the stm32n6xx_ll_rcc.h

Data Fields

- ***uint32_t SYSCLK_Frequency***
- ***uint32_t CPUCLK_Frequency***
- ***uint32_t HCLK_Frequency***
- ***uint32_t PCLK1_Frequency***
- ***uint32_t PCLK2_Frequency***
- ***uint32_t PCLK4_Frequency***
- ***uint32_t PCLK5_Frequency***

Field Documentation

- ***uint32_t LL_RCC_ClocksTypeDef::SYSCLK_Frequency***
- ***uint32_t LL_RCC_ClocksTypeDef::CPUCLK_Frequency***
- ***uint32_t LL_RCC_ClocksTypeDef::HCLK_Frequency***
- ***uint32_t LL_RCC_ClocksTypeDef::PCLK1_Frequency***
- ***uint32_t LL_RCC_ClocksTypeDef::PCLK2_Frequency***
- ***uint32_t LL_RCC_ClocksTypeDef::PCLK4_Frequency***
- ***uint32_t LL_RCC_ClocksTypeDef::PCLK5_Frequency***

101.2 RCC Firmware driver API description

The following section lists the various functions of the RCC library.

101.2.1 Detailed description of functions

LL_RCC_HSE_EnableCSS

Function name

__STATIC_INLINE void LL_RCC_HSE_EnableCSS (void)

Function description

Enable the HSE Clock Security System.

Return values

- **None:**

Notes

- Once HSE Clock Security System is enabled it cannot be changed anymore unless a reset occurs or system enter in standby mode.

Reference Manual to LL API cross reference:

- HSECFGR HSECSSON LL_RCC_HSE_EnableCSS

LL_RCC_HSE_IsFailureDetected

Function name

__STATIC_INLINE uint32_t LL_RCC_HSE_IsFailureDetected (void)

Function description

Check if HSE failure is detected by Clock Security System.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- HSECFGR HSECSSD LL_RCC_HSE_IsFailureDetected

LL_RCC_HSE_EnableCSSBypass

Function name

__STATIC_INLINE void LL_RCC_HSE_EnableCSSBypass (void)

Function description

Enable the HSE Clock Security System bypass.

Return values

- **None:**

Notes

- Bypass the HSE oscillator when a failure is detected and get the clock from the HSI oscillator (HSI injection)

Reference Manual to LL API cross reference:

- HSECFGR HSECSSBYP LL_RCC_HSE_EnableCSSBypass

LL_RCC_HSE_DisableCSSBypass

Function name

__STATIC_INLINE void LL_RCC_HSE_DisableCSSBypass (void)

Function description

Disable the HSE Clock Security System bypass.

Return values

- **None:**

Reference Manual to LL API cross reference:

- HSECFGR HSECSSBYP LL_RCC_HSE_DisableCSSBypass

LL_RCC_HSE_SetCSSBypassDivider

Function name

__STATIC_INLINE void LL_RCC_HSE_SetCSSBypassDivider (uint32_t Divider)

Function description

Set HSE Clock Security System bypass divider.

Parameters

- **Divider:** This parameter can be a value from RCC_LL_EC_HSECSSBYP_DIV.

Return values

- **None:**

Notes

- To divide the replacement internal HSI oscillator that bypasses the HSE oscillator when a failure is detected

Reference Manual to LL API cross reference:

- HSECFGR HSECSSBPRE LL_RCC_HSE_SetCSSBypassDivider

LL_RCC_HSE_GetCSSBypassDivider

Function name

__STATIC_INLINE uint32_t LL_RCC_HSE_GetCSSBypassDivider (void)

Function description

Get HSE Clock Security System bypass divider.

Return values

- **can:** be a value from RCC_LL_EC_HSECSSBYP_DIV.

Notes

- To divide the replacement internal HSI oscillator that bypasses the HSE oscillator when a failure is detected

Reference Manual to LL API cross reference:

- HSECFGR HSECSSBPRE LL_RCC_HSE_GetCSSBypassDivider

LL_RCC_HSE_EnableBypass

Function name

__STATIC_INLINE void LL_RCC_HSE_EnableBypass (void)

Function description

Enable HSE external oscillator (HSE Bypass)

Return values

- **None:**

Reference Manual to LL API cross reference:

- HSECFGR HSEBYP LL_RCC_HSE_EnableBypass

LL_RCC_HSE_DisableBypass

Function name

__STATIC_INLINE void LL_RCC_HSE_DisableBypass (void)

Function description

Disable HSE external oscillator (HSE Bypass)

Return values

- **None:**

Reference Manual to LL API cross reference:

- HSECFGR HSEBYP LL_RCC_HSE_DisableBypass

LL_RCC_HSE_SelectAnalogClock

Function name

__STATIC_INLINE void LL_RCC_HSE_SelectAnalogClock (void)

Function description

Select the Analog HSE external clock type in Bypass mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- HSECFGR HSEEXT LL_RCC_HSE_SelectAnalogClock

LL_RCC_HSE_SelectDigitalClock

Function name

__STATIC_INLINE void LL_RCC_HSE_SelectDigitalClock (void)

Function description

Select the Digital HSE external clock type in Bypass mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- HSECFGR HSEEXT LL_RCC_HSE_SelectDigitalClock

LL_RCC_HSE_SelectHSEAsDiv2Clock

Function name

__STATIC_INLINE void LL_RCC_HSE_SelectHSEAsDiv2Clock (void)

Function description

Select the HSE as hse_div2_osc_ck output clock.

Return values

- **None:**

Reference Manual to LL API cross reference:

- HSECFGR HSEDIV2SEL LL_RCC_HSE_SelectHSEAsDiv2Clock

LL_RCC_HSE_SelectHSEDiv2AsDiv2Clock

Function name

__STATIC_INLINE void LL_RCC_HSE_SelectHSEDiv2AsDiv2Clock (void)

Function description

Select the HSE divided by 2 as hse_div2_osc_ck output clock.

Return values

- **None:**

Reference Manual to LL API cross reference:

- HSECFGR HSEDIV2SEL LL_RCC_HSE_SelectHSEDiv2AsDiv2Clock

LL_RCC_HSE_IsSelectedHSEDiv2AsDiv2Clock

Function name

__STATIC_INLINE uint32_t LL_RCC_HSE_IsSelectedHSEDiv2AsDiv2Clock (void)

Function description

Check if hse_div2_osc_ck output clock is divided by 2.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- HSECFGR HSEDIV2SEL LL_RCC_HSE_IsSelectedHSEDiv2AsDiv2Clock

LL_RCC_HSE_Enable

Function name

__STATIC_INLINE void LL_RCC_HSE_Enable (void)

Function description

Enable HSE crystal oscillator (HSE ON)

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR HSEONS LL_RCC_HSE_Enable

LL_RCC_HSE_Disable

Function name

__STATIC_INLINE void LL_RCC_HSE_Disable (void)

Function description

Disable HSE crystal oscillator (HSE ON)

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCR HSEONC LL_RCC_HSE_Disable

LL_RCC_HSE_IsReady

Function name

__STATIC_INLINE uint32_t LL_RCC_HSE_IsReady (void)

Function description

Check if HSE oscillator Ready.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR HSERDY LL_RCC_HSE_IsReady

LL_RCC_HSI_Enable

Function name

__STATIC_INLINE void LL_RCC_HSI_Enable (void)

Function description

Enable HSI oscillator.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR HSIONS LL_RCC_HSI_Enable

LL_RCC_HSI_Disable

Function name

__STATIC_INLINE void LL_RCC_HSI_Disable (void)

Function description

Disable HSI oscillator.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCR HSIONC LL_RCC_HSI_Disable

LL_RCC_HSI_IsReady

Function name

__STATIC_INLINE uint32_t LL_RCC_HSI_IsReady (void)

Function description

Check if HSI clock is ready.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR HSIRDY LL_RCC_HSI_IsReady

LL_RCC_HSI_SetDivider

Function name

__STATIC_INLINE void LL_RCC_HSI_SetDivider (uint32_t Divider)

Function description

Set HSI divider.

Parameters

- **Divider:** This parameter can be one of the following values:
  - LL_RCC_HSI_DIV_1
  - LL_RCC_HSI_DIV_2
  - LL_RCC_HSI_DIV_4
  - LL_RCC_HSI_DIV_8

Return values

- **None.:**

Reference Manual to LL API cross reference:

- HSICFGR HSIDIV LL_RCC_HSI_SetDivider

LL_RCC_HSI_GetDivider

Function name

__STATIC_INLINE uint32_t LL_RCC_HSI_GetDivider (void)

Function description

Get HSI divider.

Return values

- **can:** be one of the following values:
  - LL_RCC_HSI_DIV_1
  - LL_RCC_HSI_DIV_2
  - LL_RCC_HSI_DIV_4
  - LL_RCC_HSI_DIV_8

Reference Manual to LL API cross reference:

- HSICFGR HSIDIV LL_RCC_HSI_GetDivider

LL_RCC_HSI_GetCalibration

Function name

__STATIC_INLINE uint32_t LL_RCC_HSI_GetCalibration (void)

Function description

Get HSI Calibration value.

Return values

- **A:** value between 0 and 511 (0x1FF)

Notes

- When HSITRIM is written, HSICAL is updated with the sum of HSITRIM and the factory trim value

Reference Manual to LL API cross reference:

- HSICFGR HSICAL LL_RCC_HSI_GetCalibration

LL_RCC_HSI_SetCalibTrimming

Function name

__STATIC_INLINE void LL_RCC_HSI_SetCalibTrimming (uint32_t Value)

Function description

Set HSI Calibration trimming.

Parameters

- **Value:** This parameter can be a value between 0 and 63

Return values

- **None:**

Notes

- user-programmable trimming value that is added to the HSICAL
- Default value is 32, which, when added to the HSICAL value, should trim the HSI to 32 MHz +/- 1 %

Reference Manual to LL API cross reference:

- HSICFGR HSITRIM LL_RCC_HSI_SetCalibTrimming

LL_RCC_HSI_GetCalibTrimming

Function name

__STATIC_INLINE uint32_t LL_RCC_HSI_GetCalibTrimming (void)

Function description

Get HSI Calibration trimming.

Return values

- **A:** value between 0 and 63

Reference Manual to LL API cross reference:

- HSICFGR HSITRIM LL_RCC_HSI_GetCalibTrimming

LL_RCC_HSI_EnableInStopMode

Function name

__STATIC_INLINE void LL_RCC_HSI_EnableInStopMode (void)

Function description

Enable HSI even in stop mode.

Return values

- **None:**

Notes

- HSI oscillator is forced ON even in Stop mode

Reference Manual to LL API cross reference:

- STOPCR HSISTOPEN LL_RCC_HSI_EnableInStopMode

LL_RCC_HSI_DisableInStopMode

Function name

__STATIC_INLINE void LL_RCC_HSI_DisableInStopMode (void)

Function description

Disable HSI in stop mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- STOPCR HSISTOPEN LL_RCC_HSI_DisableInStopMode

LL_RCC_HSI_IsEnabledInStopMode

Function name

__STATIC_INLINE uint32_t LL_RCC_HSI_IsEnabledInStopMode (void)

Function description

Check if HSI is enabled in stop mode.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- STOPCR HSISTOPEN LL_RCC_HSI_IsEnabledInStopMode

LL_RCC_MSI_Enable

Function name

__STATIC_INLINE void LL_RCC_MSI_Enable (void)

Function description

Enable MSI oscillator.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR MSIONS LL_RCC_MSI_Enable

LL_RCC_MSI_Disable

Function name

__STATIC_INLINE void LL_RCC_MSI_Disable (void)

Function description

Disable MSI oscillator.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCR MSIONC LL_RCC_MSI_Disable

LL_RCC_MSI_IsReady

Function name

__STATIC_INLINE uint32_t LL_RCC_MSI_IsReady (void)

Function description

Check if MSI clock is ready.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR MSIRDY LL_RCC_MSI_IsReady

LL_RCC_MSI_SetFrequency

Function name

__STATIC_INLINE void LL_RCC_MSI_SetFrequency (uint32_t Value)

Function description

Set MSI frequency.

Parameters

- **Value:** This parameter can be one of the following values:
  - LL_RCC_MSI_FREQ_4MHZ
  - LL_RCC_MSI_FREQ_16MHZ

Return values

- **None.:**

Reference Manual to LL API cross reference:

- MSICFGR MSIFREQSEL LL_RCC_MSI_SetFrequency

LL_RCC_MSI_GetFrequency

Function name

__STATIC_INLINE uint32_t LL_RCC_MSI_GetFrequency (void)

Function description

Get HSI divider.

Return values

- **can:** be one of the following values:
  - LL_RCC_MSI_FREQ_4MHZ
  - LL_RCC_MSI_FREQ_16MHZ

Reference Manual to LL API cross reference:

- MSICFGR MSIFREQSEL LL_RCC_MSI_GetFrequency

LL_RCC_MSI_GetCalibration

Function name

__STATIC_INLINE uint32_t LL_RCC_MSI_GetCalibration (void)

Function description

Get MSI Calibration value.

Return values

- **A:** value between 0 and 255 (0xFF)

Notes

- When MSITRIM is written, MSICAL is updated with the sum of MSITRIM and the factory trim value

Reference Manual to LL API cross reference:

- MSICFGR MSICAL LL_RCC_MSI_GetCalibration

LL_RCC_MSI_SetCalibTrimming

Function name

__STATIC_INLINE void LL_RCC_MSI_SetCalibTrimming (uint32_t Value)

Function description

Set MSI Calibration trimming.

Parameters

- **Value:** can be a value between 0 and 31

Return values

- **None:**

Notes

- user-programmable trimming value that is added to the MSICAL
- Default value is 16, which, when added to the MSICAL value, should trim the MSI to 4 MHz +/- 1 %

Reference Manual to LL API cross reference:

- MSICFGR MSITRIM LL_RCC_MSI_SetCalibTrimming

LL_RCC_MSI_GetCalibTrimming

Function name

__STATIC_INLINE uint32_t LL_RCC_MSI_GetCalibTrimming (void)

Function description

Get MSI Calibration trimming.

Return values

- **A:** value between 0 and 31

Reference Manual to LL API cross reference:

- MSICFGR MSITRIM LL_RCC_MSI_GetCalibTrimming

LL_RCC_MSI_EnableInStopMode

Function name

__STATIC_INLINE void LL_RCC_MSI_EnableInStopMode (void)

Function description

Enable MSI even in stop mode.

Return values

- **None:**

Notes

- MSI oscillator is forced ON even in Stop mode

Reference Manual to LL API cross reference:

- STOPCR MSISTOPEN LL_RCC_MSI_EnableInStopMode

LL_RCC_MSI_DisableInStopMode

Function name

__STATIC_INLINE void LL_RCC_MSI_DisableInStopMode (void)

Function description

Disable MSI in stop mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- STOPCR MSISTOPEN LL_RCC_MSI_DisableInStopMode

LL_RCC_MSI_IsEnabledInStopMode

Function name

__STATIC_INLINE uint32_t LL_RCC_MSI_IsEnabledInStopMode (void)

Function description

Check if MSI is enabled in stop mode.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- STOPCR MSISTOPEN LL_RCC_MSI_IsEnabledInStopMode

LL_RCC_LSE_EnableCSS

Function name

__STATIC_INLINE void LL_RCC_LSE_EnableCSS (void)

Function description

Enable the Clock Security System on LSE.

Return values

- **None:**

Notes

- Once LSE Clock Security System is enabled it cannot be changed anymore unless a clock failure is detected.

Reference Manual to LL API cross reference:

- LSECFGR LSECSSON LL_RCC_LSE_EnableCSS

LL_RCC_LSE_DisableCSS

Function name

__STATIC_INLINE void LL_RCC_LSE_DisableCSS (void)

Function description

Disable the Clock Security System on LSE.

Return values

- **None:**

Notes

- Once LSE Clock Security System is enabled it cannot be changed anymore unless a clock failure is detected.

Reference Manual to LL API cross reference:

- LSECFGR LSECSSON LL_RCC_LSE_DisableCSS

LL_RCC_LSE_IsFailureDetected

Function name

__STATIC_INLINE uint32_t LL_RCC_LSE_IsFailureDetected (void)

Function description

Check if LSE failure is detected by Clock Security System.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- LSECFGR LSECSSD LL_RCC_LSE_IsFailureDetected

LL_RCC_LSE_ReArmCSS

Function name

__STATIC_INLINE void LL_RCC_LSE_ReArmCSS (void)

Function description

Re-arm the Clock Security System on LSE.

Return values

- **None:**

Notes

- Once a clock failure is detected, the LSE Clock Security System can be re-armed providing that LSECSSON is disabled.

Reference Manual to LL API cross reference:

- LSECFGR LSECSSRA LL_RCC_LSE_ReArmCSS

LL_RCC_LSE_EnableBypass

Function name

__STATIC_INLINE void LL_RCC_LSE_EnableBypass (void)

Function description

Enable external clock source (LSE bypass)

Return values

- **None:**

Reference Manual to LL API cross reference:

- LSECFGR LSEBYP LL_RCC_LSE_EnableBypass

LL_RCC_LSE_DisableBypass

Function name

__STATIC_INLINE void LL_RCC_LSE_DisableBypass (void)

Function description

Disable external clock source (LSE bypass)

Return values

- **None:**

Reference Manual to LL API cross reference:

- LSECFGR LSEBYP LL_RCC_LSE_DisableBypass

LL_RCC_LSE_SelectDigitalClock

Function name

__STATIC_INLINE void LL_RCC_LSE_SelectDigitalClock (void)

Function description

Enable Low-speed external DIGITAL clock type in Bypass mode (not to be used if RTC is active).

Return values

- **None:**

Notes

- The external clock must be enabled with the LSEON bit, to be used by the device. The LSEEXT bit can be written only if the LSE oscillator is disabled.

Reference Manual to LL API cross reference:

- LSECFGR LSEEXT LL_RCC_LSE_SelectDigitalClock

LL_RCC_LSE_SelectAnalogClock

Function name

__STATIC_INLINE void LL_RCC_LSE_SelectAnalogClock (void)

Function description

Enable Low-speed external ANALOG clock type in Bypass mode (default after Backup domain reset).

Return values

- **None:**

Notes

- The external clock must be enabled with the LSEON bit, to be used by the device. The LSEEXT bit can be written only if the LSE oscillator is disabled.

Reference Manual to LL API cross reference:

- LSECFGR LSEEXT LL_RCC_LSE_SelectAnalogClock

LL_RCC_LSE_SetDriveCapability

Function name

__STATIC_INLINE void LL_RCC_LSE_SetDriveCapability (uint32_t LSEDrive)

Function description

Set LSE oscillator drive capability.

Parameters

- **LSEDrive:** This parameter can be one of the following values:
  - LL_RCC_LSEDRIVE_LOW
  - LL_RCC_LSEDRIVE_MEDIUMLOW
  - LL_RCC_LSEDRIVE_MEDIUMHIGH
  - LL_RCC_LSEDRIVE_HIGH

Return values

- **None:**

Notes

- The oscillator is in Xtal mode when it is not in bypass mode.

Reference Manual to LL API cross reference:

- LSECFGR LSEDRV LL_RCC_LSE_SetDriveCapability

LL_RCC_LSE_GetDriveCapability

Function name

__STATIC_INLINE uint32_t LL_RCC_LSE_GetDriveCapability (void)

Function description

Get LSE oscillator drive capability.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_LSEDRIVE_LOW
  - LL_RCC_LSEDRIVE_MEDIUMLOW
  - LL_RCC_LSEDRIVE_MEDIUMHIGH
  - LL_RCC_LSEDRIVE_HIGH

Reference Manual to LL API cross reference:

- LSECFGR LSEDRV LL_RCC_LSE_GetDriveCapability

LL_RCC_LSE_Enable

Function name

__STATIC_INLINE void LL_RCC_LSE_Enable (void)

Function description

Enable Low Speed External (LSE) crystal.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR LSEONS LL_RCC_LSE_Enable

LL_RCC_LSE_Disable

Function name

__STATIC_INLINE void LL_RCC_LSE_Disable (void)

Function description

Disable Low Speed External (LSE) crystal.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCR LSEONC LL_RCC_LSE_Disable

LL_RCC_LSE_IsReady

Function name

__STATIC_INLINE uint32_t LL_RCC_LSE_IsReady (void)

Function description

Check if LSE oscillator Ready.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR LSERDY LL_RCC_LSE_IsReady

LL_RCC_LSE_EnableGlitchFilter

Function name

__STATIC_INLINE void LL_RCC_LSE_EnableGlitchFilter (void)

Function description

Enable the LSE clock glitch filter.

Return values

- **None:**

Notes

- This API shall be called only when LSE is disabled.

Reference Manual to LL API cross reference:

- LSECFGR LSEGFON LL_RCC_LSE_EnableGlitchFilter

LL_RCC_LSE_DisableGlitchFilter

Function name

__STATIC_INLINE void LL_RCC_LSE_DisableGlitchFilter (void)

Function description

Disable the LSE clock glitch filter.

Return values

- **None:**

Notes

- This API shall be called only when LSE is disabled.

Reference Manual to LL API cross reference:

- LSECFGR LSEGFON LL_RCC_LSE_DisableGlitchFilter

LL_RCC_LSE_IsEnabledGlitchFilter

Function name

__STATIC_INLINE uint32_t LL_RCC_LSE_IsEnabledGlitchFilter (void)

Function description

Check if LSE clock glitch filter is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- LSECFGR LSEGFON LL_RCC_LSE_IsEnabledGlitchFilter

LL_RCC_LSI_Enable

Function name

__STATIC_INLINE void LL_RCC_LSI_Enable (void)

Function description

Enable LSI Oscillator.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR LSIONS LL_RCC_LSI_Enable

LL_RCC_LSI_Disable

Function name

__STATIC_INLINE void LL_RCC_LSI_Disable (void)

Function description

Disable LSI Oscillator.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCR LSIONC LL_RCC_LSI_Disable

LL_RCC_LSI_IsReady

Function name

__STATIC_INLINE uint32_t LL_RCC_LSI_IsReady (void)

Function description

Check if LSI is Ready.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR LSIRDY LL_RCC_LSI_IsReady

LL_RCC_SetCpuClkSource

Function name

__STATIC_INLINE void LL_RCC_SetCpuClkSource (uint32_t Source)

Function description

Configure the CPU clock source.

Parameters

- **Source:** This parameter can be one of the following values:
  - LL_RCC_CPU_CLKSOURCE_HSI
  - LL_RCC_CPU_CLKSOURCE_MSI
  - LL_RCC_CPU_CLKSOURCE_HSE
  - LL_RCC_CPU_CLKSOURCE_IC1

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR1 CPUSW LL_RCC_SetCpuClkSource

LL_RCC_GetCpuClkSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetCpuClkSource (void)

Function description

Get the CPU clock source.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_CPU_CLKSOURCE_STATUS_HSI
  - LL_RCC_CPU_CLKSOURCE_STATUS_MSI
  - LL_RCC_CPU_CLKSOURCE_STATUS_HSE
  - LL_RCC_CPU_CLKSOURCE_STATUS_IC1

Reference Manual to LL API cross reference:

- CFGR1 CPUSWS LL_RCC_GetCpuClkSource

LL_RCC_SetSysClkSource

Function name

__STATIC_INLINE void LL_RCC_SetSysClkSource (uint32_t Source)

Function description

Configure the system clock source (bus clock)

Parameters

- **Source:** This parameter can be one of the following values:
  - LL_RCC_SYS_CLKSOURCE_HSI
  - LL_RCC_SYS_CLKSOURCE_MSI
  - LL_RCC_SYS_CLKSOURCE_HSE
  - LL_RCC_SYS_CLKSOURCE_IC2_IC6_IC11

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR1 SYSSW LL_RCC_SetSysClkSource

LL_RCC_GetSysClkSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetSysClkSource (void)

Function description

Get the system clock source (bus clock)

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_SYS_CLKSOURCE_STATUS_HSI
  - LL_RCC_SYS_CLKSOURCE_STATUS_MSI
  - LL_RCC_SYS_CLKSOURCE_STATUS_HSE
  - LL_RCC_SYS_CLKSOURCE_STATUS_IC2_IC6_IC11

Reference Manual to LL API cross reference:

- CFGR1 SYSSWS LL_RCC_GetSysClkSource

LL_RCC_SetSysWakeUpClkSource

Function name

__STATIC_INLINE void LL_RCC_SetSysWakeUpClkSource (uint32_t Source)

Function description

Configure the system wakeup clock source.

Parameters

- **Source:** This parameter can be one of the following values:
  - LL_RCC_SYSWAKEUP_CLKSOURCE_HSI
  - LL_RCC_SYSWAKEUP_CLKSOURCE_MSI

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR1 STOPWUCK LL_RCC_SetSysWakeUpClkSource

LL_RCC_GetSysWakeUpClkSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetSysWakeUpClkSource (void)

Function description

Get the system wakeup clock source.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_SYSWAKEUP_CLKSOURCE_HSI
  - LL_RCC_SYSWAKEUP_CLKSOURCE_MSI

Reference Manual to LL API cross reference:

- CFGR1 STOPWUCK LL_RCC_GetSysWakeUpClkSource

LL_RCC_SetAHBPrescaler

Function name

__STATIC_INLINE void LL_RCC_SetAHBPrescaler (uint32_t Prescaler)

Function description

Set AHB prescaler.

Parameters

- **Prescaler:** This parameter can be one of the following values:
  - LL_RCC_AHB_DIV_1
  - LL_RCC_AHB_DIV_2
  - LL_RCC_AHB_DIV_4
  - LL_RCC_AHB_DIV_8
  - LL_RCC_AHB_DIV_16
  - LL_RCC_AHB_DIV_32
  - LL_RCC_AHB_DIV_64
  - LL_RCC_AHB_DIV_128

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR2 HPRE LL_RCC_SetAHBPrescaler

LL_RCC_GetAHBPrescaler

Function name

__STATIC_INLINE uint32_t LL_RCC_GetAHBPrescaler (void)

Function description

Get AHB prescaler.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_AHB_DIV_1
  - LL_RCC_AHB_DIV_2
  - LL_RCC_AHB_DIV_4
  - LL_RCC_AHB_DIV_8
  - LL_RCC_AHB_DIV_16
  - LL_RCC_AHB_DIV_32
  - LL_RCC_AHB_DIV_64
  - LL_RCC_AHB_DIV_128

Reference Manual to LL API cross reference:

- CFGR2 HPRE LL_RCC_GetAHBPrescaler

LL_RCC_SetAPB1Prescaler

Function name

__STATIC_INLINE void LL_RCC_SetAPB1Prescaler (uint32_t Prescaler)

Function description

Set APB1 prescaler.

Parameters

- **Prescaler:** This parameter can be one of the following values:
  - LL_RCC_APB1_DIV_1
  - LL_RCC_APB1_DIV_2
  - LL_RCC_APB1_DIV_4
  - LL_RCC_APB1_DIV_8
  - LL_RCC_APB1_DIV_16
  - LL_RCC_APB1_DIV_32
  - LL_RCC_APB1_DIV_64
  - LL_RCC_APB1_DIV_128

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR2 PPRE1 LL_RCC_SetAPB1Prescaler

LL_RCC_GetAPB1Prescaler

Function name

__STATIC_INLINE uint32_t LL_RCC_GetAPB1Prescaler (void)

Function description

Get APB1 prescaler.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_APB1_DIV_1
  - LL_RCC_APB1_DIV_2
  - LL_RCC_APB1_DIV_4
  - LL_RCC_APB1_DIV_8
  - LL_RCC_APB1_DIV_16
  - LL_RCC_APB1_DIV_32
  - LL_RCC_APB1_DIV_64
  - LL_RCC_APB1_DIV_128

Reference Manual to LL API cross reference:

- CFGR2 PPRE1 LL_RCC_GetAPB1Prescaler

LL_RCC_SetAPB2Prescaler

Function name

__STATIC_INLINE void LL_RCC_SetAPB2Prescaler (uint32_t Prescaler)

Function description

Set APB2 prescaler.

Parameters

- **Prescaler:** This parameter can be one of the following values:
  - LL_RCC_APB2_DIV_1
  - LL_RCC_APB2_DIV_2
  - LL_RCC_APB2_DIV_4
  - LL_RCC_APB2_DIV_8
  - LL_RCC_APB2_DIV_16
  - LL_RCC_APB2_DIV_32
  - LL_RCC_APB2_DIV_64
  - LL_RCC_APB2_DIV_128

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR2 PPRE2 LL_RCC_SetAPB2Prescaler

LL_RCC_GetAPB2Prescaler

Function name

__STATIC_INLINE uint32_t LL_RCC_GetAPB2Prescaler (void)

Function description

Get APB2 prescaler.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_APB2_DIV_1
  - LL_RCC_APB2_DIV_2
  - LL_RCC_APB2_DIV_4
  - LL_RCC_APB2_DIV_8
  - LL_RCC_APB2_DIV_16
  - LL_RCC_APB2_DIV_32
  - LL_RCC_APB2_DIV_64
  - LL_RCC_APB2_DIV_128

Reference Manual to LL API cross reference:

- CFGR2 PPRE2 LL_RCC_GetAPB2Prescaler

LL_RCC_SetAPB4Prescaler

Function name

__STATIC_INLINE void LL_RCC_SetAPB4Prescaler (uint32_t Prescaler)

Function description

Set APB4 prescaler.

Parameters

- **Prescaler:** This parameter can be one of the following values:
  - LL_RCC_APB4_DIV_1
  - LL_RCC_APB4_DIV_2
  - LL_RCC_APB4_DIV_4
  - LL_RCC_APB4_DIV_8
  - LL_RCC_APB4_DIV_16
  - LL_RCC_APB4_DIV_32
  - LL_RCC_APB4_DIV_64
  - LL_RCC_APB4_DIV_128

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR2 PPRE4 LL_RCC_SetAPB4Prescaler

LL_RCC_GetAPB4Prescaler

Function name

__STATIC_INLINE uint32_t LL_RCC_GetAPB4Prescaler (void)

Function description

Get APB4 prescaler.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_APB4_DIV_1
  - LL_RCC_APB4_DIV_2
  - LL_RCC_APB4_DIV_4
  - LL_RCC_APB4_DIV_8
  - LL_RCC_APB4_DIV_16
  - LL_RCC_APB4_DIV_32
  - LL_RCC_APB4_DIV_64
  - LL_RCC_APB4_DIV_128

Reference Manual to LL API cross reference:

- CFGR2 PPRE4 LL_RCC_GetAPB4Prescaler

LL_RCC_SetAPB5Prescaler

Function name

__STATIC_INLINE void LL_RCC_SetAPB5Prescaler (uint32_t Prescaler)

Function description

Set APB5 prescaler.

Parameters

- **Prescaler:** This parameter can be one of the following values:
  - LL_RCC_APB5_DIV_1
  - LL_RCC_APB5_DIV_2
  - LL_RCC_APB5_DIV_4
  - LL_RCC_APB5_DIV_8
  - LL_RCC_APB5_DIV_16
  - LL_RCC_APB5_DIV_32
  - LL_RCC_APB5_DIV_64
  - LL_RCC_APB5_DIV_128

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR2 PPRE5 LL_RCC_SetAPB5Prescaler

LL_RCC_GetAPB5Prescaler

Function name

__STATIC_INLINE uint32_t LL_RCC_GetAPB5Prescaler (void)

Function description

Get APB5 prescaler.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_APB5_DIV_1
  - LL_RCC_APB5_DIV_2
  - LL_RCC_APB5_DIV_4
  - LL_RCC_APB5_DIV_8
  - LL_RCC_APB5_DIV_16
  - LL_RCC_APB5_DIV_32
  - LL_RCC_APB5_DIV_64
  - LL_RCC_APB5_DIV_128

Reference Manual to LL API cross reference:

- CFGR2 PPRE5 LL_RCC_GetAPB5Prescaler

LL_RCC_EnableMCO

Function name

__STATIC_INLINE void LL_RCC_EnableMCO (uint32_t MCOx)

Function description

Enable MCOx output.

Parameters

- **MCOx:** This parameter can be one or a combination of the following values:
  - LL_RCC_MCO1
  - LL_RCC_MCO2

Return values

- **None:**

Notes

- The clock provided to the MCOx outputs must not exceed the maximum IO speed, refer to the product datasheet for information about the supported IO speed.

Reference Manual to LL API cross reference:

- MISCENSR MCO1ENS LL_RCC_EnableMCO
- MISCENSR MCO2ENS LL_RCC_EnableMCO

LL_RCC_DisableMCO

Function name

__STATIC_INLINE void LL_RCC_DisableMCO (uint32_t MCOx)

Function description

Disable MCOx output.

Parameters

- **MCOx:** This parameter can be one or a combination of the following values:
  - LL_RCC_MCO1
  - LL_RCC_MCO2

Return values

- **None:**

Reference Manual to LL API cross reference:

- MISCENCR MCO1ENC LL_RCC_DisableMCO
- MISCENCR MCO2ENC LL_RCC_DisableMCO

LL_RCC_ConfigMCO

Function name

__STATIC_INLINE void LL_RCC_ConfigMCO (uint32_t MCOxSource, uint32_t MCOxPrescaler)

Function description

Configure MCOx.

Parameters

- **MCOxSource:** This parameter can be one of the following values:
  - LL_RCC_MCO1SOURCE_LSI
  - LL_RCC_MCO1SOURCE_LSE
  - LL_RCC_MCO1SOURCE_MSI
  - LL_RCC_MCO1SOURCE_HSI
  - LL_RCC_MCO1SOURCE_HSE
  - LL_RCC_MCO1SOURCE_IC5
  - LL_RCC_MCO1SOURCE_IC10
  - LL_RCC_MCO1SOURCE_SYSA
  - LL_RCC_MCO2SOURCE_LSI
  - LL_RCC_MCO2SOURCE_LSE
  - LL_RCC_MCO2SOURCE_MSI
  - LL_RCC_MCO2SOURCE_HSI
  - LL_RCC_MCO2SOURCE_HSE
  - LL_RCC_MCO2SOURCE_IC15
  - LL_RCC_MCO2SOURCE_IC20
  - LL_RCC_MCO2SOURCE_SYSB
- **MCOxPrescaler:** This parameter can be one of the following values:
  - LL_RCC_MCO1_DIV_1
  - LL_RCC_MCO1_DIV_2
  - LL_RCC_MCO1_DIV_3
  - LL_RCC_MCO1_DIV_4
  - LL_RCC_MCO1_DIV_5
  - LL_RCC_MCO1_DIV_6
  - LL_RCC_MCO1_DIV_7
  - LL_RCC_MCO1_DIV_8
  - LL_RCC_MCO1_DIV_9
  - LL_RCC_MCO1_DIV_10
  - LL_RCC_MCO1_DIV_11
  - LL_RCC_MCO1_DIV_12
  - LL_RCC_MCO1_DIV_13
  - LL_RCC_MCO1_DIV_14
  - LL_RCC_MCO1_DIV_15
  - LL_RCC_MCO2_DIV_1
  - LL_RCC_MCO2_DIV_2
  - LL_RCC_MCO2_DIV_3
  - LL_RCC_MCO2_DIV_4
  - LL_RCC_MCO2_DIV_5
  - LL_RCC_MCO2_DIV_6
  - LL_RCC_MCO2_DIV_7
  - LL_RCC_MCO2_DIV_8
  - LL_RCC_MCO2_DIV_9
  - LL_RCC_MCO2_DIV_10
  - LL_RCC_MCO2_DIV_11
  - LL_RCC_MCO2_DIV_12
  - LL_RCC_MCO2_DIV_13
  - LL_RCC_MCO2_DIV_14
  - LL_RCC_MCO2_DIV_15
  - LL_RCC_MCO2_DIV_16

Return values

- **None:**

Notes

- The clock provided to the MCOx outputs must not exceed the maximum IO speed, refer to the product datasheet for information about the supported IO speed.
- The MCO switch to the new clock source only occurs when the previous clock source is active (dynamic switch).

Reference Manual to LL API cross reference:

- CCIPR5 MCO1SEL LL_RCC_ConfigMCO
- CCIPR5 MCO1PRE LL_RCC_ConfigMCO
- CCIPR5 MCO2SEL LL_RCC_ConfigMCO
- CCIPR5 MCO2PRE LL_RCC_ConfigMCO

LL_RCC_SetClockSource

Function name

__STATIC_INLINE void LL_RCC_SetClockSource (uint32_t ClkSource)

Function description

Configure periph clock source.

Parameters

- **ClkSource:** This parameter can be one of the following values:
  - LL_RCC_I2C1_CLKSOURCE_PCLK1
  - LL_RCC_I2C1_CLKSOURCE_CLKP
  - LL_RCC_I2C1_CLKSOURCE_IC10
  - LL_RCC_I2C1_CLKSOURCE_IC15
  - LL_RCC_I2C1_CLKSOURCE_MSI
  - LL_RCC_I2C1_CLKSOURCE_HSI
  - LL_RCC_I2C2_CLKSOURCE_PCLK1
  - LL_RCC_I2C2_CLKSOURCE_CLKP
  - LL_RCC_I2C2_CLKSOURCE_IC10
  - LL_RCC_I2C2_CLKSOURCE_IC15
  - LL_RCC_I2C2_CLKSOURCE_MSI
  - LL_RCC_I2C2_CLKSOURCE_HSI
  - LL_RCC_I2C3_CLKSOURCE_PCLK1
  - LL_RCC_I2C3_CLKSOURCE_CLKP
  - LL_RCC_I2C3_CLKSOURCE_IC10
  - LL_RCC_I2C3_CLKSOURCE_IC15
  - LL_RCC_I2C3_CLKSOURCE_MSI
  - LL_RCC_I2C3_CLKSOURCE_HSI
  - LL_RCC_I2C4_CLKSOURCE_PCLK1
  - LL_RCC_I2C4_CLKSOURCE_CLKP
  - LL_RCC_I2C4_CLKSOURCE_IC10
  - LL_RCC_I2C4_CLKSOURCE_IC15
  - LL_RCC_I2C4_CLKSOURCE_MSI
  - LL_RCC_I2C4_CLKSOURCE_HSI
  - LL_RCC_I3C1_CLKSOURCE_PCLK1
  - LL_RCC_I3C1_CLKSOURCE_CLKP
  - LL_RCC_I3C1_CLKSOURCE_IC10
  - LL_RCC_I3C1_CLKSOURCE_IC15
  - LL_RCC_I3C1_CLKSOURCE_MSI
  - LL_RCC_I3C1_CLKSOURCE_HSI
  - LL_RCC_I3C2_CLKSOURCE_PCLK1
  - LL_RCC_I3C2_CLKSOURCE_CLKP
  - LL_RCC_I3C2_CLKSOURCE_IC10
  - LL_RCC_I3C2_CLKSOURCE_IC15
  - LL_RCC_I3C2_CLKSOURCE_MSI
  - LL_RCC_I3C2_CLKSOURCE_HSI
  - LL_RCC_LPTIM1_CLKSOURCE_PCLK1
  - LL_RCC_LPTIM1_CLKSOURCE_CLKP
  - LL_RCC_LPTIM1_CLKSOURCE_IC15
  - LL_RCC_LPTIM1_CLKSOURCE_LSE
  - LL_RCC_LPTIM1_CLKSOURCE_LSI
  - LL_RCC_LPTIM1_CLKSOURCE_TIMG
  - LL_RCC_LPTIM2_CLKSOURCE_PCLK4
  - LL_RCC_LPTIM2_CLKSOURCE_CLKP
  - LL_RCC_LPTIM2_CLKSOURCE_IC15
  - LL_RCC_LPTIM2_CLKSOURCE_LSE
  - LL_RCC_LPTIM2_CLKSOURCE_LSI
  - LL_RCC_LPTIM2_CLKSOURCE_TIMG

- LL_RCC_LPTIM3_CLKSOURCE_PCLK4
- LL_RCC_LPTIM3_CLKSOURCE_CLKP
- LL_RCC_LPTIM3_CLKSOURCE_IC15
- LL_RCC_LPTIM3_CLKSOURCE_LSE
- LL_RCC_LPTIM3_CLKSOURCE_LSI
- LL_RCC_LPTIM3_CLKSOURCE_TIMG
- LL_RCC_LPTIM4_CLKSOURCE_PCLK4
- LL_RCC_LPTIM4_CLKSOURCE_CLKP
- LL_RCC_LPTIM4_CLKSOURCE_IC15
- LL_RCC_LPTIM4_CLKSOURCE_LSE
- LL_RCC_LPTIM4_CLKSOURCE_LSI
- LL_RCC_LPTIM4_CLKSOURCE_TIMG
- LL_RCC_LPTIM5_CLKSOURCE_PCLK4
- LL_RCC_LPTIM5_CLKSOURCE_CLKP
- LL_RCC_LPTIM5_CLKSOURCE_IC15
- LL_RCC_LPTIM5_CLKSOURCE_LSE
- LL_RCC_LPTIM5_CLKSOURCE_LSI
- LL_RCC_LPTIM5_CLKSOURCE_TIMG
- LL_RCC_OTGPHY1_CLKSOURCE_HSE_DIV_2
- LL_RCC_OTGPHY1_CLKSOURCE_CLKP
- LL_RCC_OTGPHY1_CLKSOURCE_IC15
- LL_RCC_OTGPHY1_CLKSOURCE_HSE_DIV_2_OSC
- LL_RCC_OTGPHY1CKREF_CLKSOURCE_OTGPHY1
- LL_RCC_OTGPHY1CKREF_CLKSOURCE_HSE_DIV_2_OSC
- LL_RCC_OTGPHY2_CLKSOURCE_HSE_DIV_2
- LL_RCC_OTGPHY2_CLKSOURCE_CLKP
- LL_RCC_OTGPHY2_CLKSOURCE_IC15
- LL_RCC_OTGPHY2_CLKSOURCE_HSE_DIV_2_OSC
- LL_RCC_OTGPHY2CKREF_CLKSOURCE_OTGPHY2
- LL_RCC_OTGPHY2CKREF_CLKSOURCE_HSE_DIV_2_OSC
- LL_RCC_SAI1_CLKSOURCE_PCLK2
- LL_RCC_SAI1_CLKSOURCE_CLKP
- LL_RCC_SAI1_CLKSOURCE_IC7
- LL_RCC_SAI1_CLKSOURCE_IC8
- LL_RCC_SAI1_CLKSOURCE_MSI
- LL_RCC_SAI1_CLKSOURCE_HSI
- LL_RCC_SAI1_CLKSOURCE_I2S_CKIN
- LL_RCC_SAI1_CLKSOURCE_SPDIFRX1
- LL_RCC_SAI2_CLKSOURCE_PCLK2
- LL_RCC_SAI2_CLKSOURCE_CLKP
- LL_RCC_SAI2_CLKSOURCE_IC7
- LL_RCC_SAI2_CLKSOURCE_IC8
- LL_RCC_SAI2_CLKSOURCE_MSI
- LL_RCC_SAI2_CLKSOURCE_HSI
- LL_RCC_SAI2_CLKSOURCE_I2S_CKIN
- LL_RCC_SAI2_CLKSOURCE_SPDIFRX1
- LL_RCC_SDMMC1_CLKSOURCE_HCLK
- LL_RCC_SDMMC1_CLKSOURCE_CLKP
- LL_RCC_SDMMC1_CLKSOURCE_IC4
- LL_RCC_SDMMC1_CLKSOURCE_IC5

- LL_RCC_SDMMC2_CLKSOURCE_HCLK
- LL_RCC_SDMMC2_CLKSOURCE_CLKP
- LL_RCC_SDMMC2_CLKSOURCE_IC4
- LL_RCC_SDMMC2_CLKSOURCE_IC5
- LL_RCC_SPI1_CLKSOURCE_PCLK2
- LL_RCC_SPI1_CLKSOURCE_CLKP
- LL_RCC_SPI1_CLKSOURCE_IC8
- LL_RCC_SPI1_CLKSOURCE_IC9
- LL_RCC_SPI1_CLKSOURCE_MSI
- LL_RCC_SPI1_CLKSOURCE_HSI
- LL_RCC_SPI1_CLKSOURCE_I2S_CKIN
- LL_RCC_SPI2_CLKSOURCE_PCLK1
- LL_RCC_SPI2_CLKSOURCE_CLKP
- LL_RCC_SPI2_CLKSOURCE_IC8
- LL_RCC_SPI2_CLKSOURCE_IC9
- LL_RCC_SPI2_CLKSOURCE_MSI
- LL_RCC_SPI2_CLKSOURCE_HSI
- LL_RCC_SPI2_CLKSOURCE_I2S_CKIN
- LL_RCC_SPI3_CLKSOURCE_PCLK1
- LL_RCC_SPI3_CLKSOURCE_CLKP
- LL_RCC_SPI3_CLKSOURCE_IC8
- LL_RCC_SPI3_CLKSOURCE_IC9
- LL_RCC_SPI3_CLKSOURCE_MSI
- LL_RCC_SPI3_CLKSOURCE_HSI
- LL_RCC_SPI3_CLKSOURCE_I2S_CKIN
- LL_RCC_SPI4_CLKSOURCE_PCLK2
- LL_RCC_SPI4_CLKSOURCE_CLKP
- LL_RCC_SPI4_CLKSOURCE_IC9
- LL_RCC_SPI4_CLKSOURCE_IC14
- LL_RCC_SPI4_CLKSOURCE_MSI
- LL_RCC_SPI4_CLKSOURCE_HSI
- LL_RCC_SPI4_CLKSOURCE_HSE
- LL_RCC_SPI5_CLKSOURCE_PCLK2
- LL_RCC_SPI5_CLKSOURCE_CLKP
- LL_RCC_SPI5_CLKSOURCE_IC9
- LL_RCC_SPI5_CLKSOURCE_IC14
- LL_RCC_SPI5_CLKSOURCE_MSI
- LL_RCC_SPI5_CLKSOURCE_HSI
- LL_RCC_SPI5_CLKSOURCE_HSE
- LL_RCC_SPI6_CLKSOURCE_PCLK4
- LL_RCC_SPI6_CLKSOURCE_CLKP
- LL_RCC_SPI6_CLKSOURCE_IC8
- LL_RCC_SPI6_CLKSOURCE_IC9
- LL_RCC_SPI6_CLKSOURCE_MSI
- LL_RCC_SPI6_CLKSOURCE_HSI
- LL_RCC_SPI6_CLKSOURCE_I2S_CKIN
- LL_RCC_UART4_CLKSOURCE_PCLK1
- LL_RCC_UART4_CLKSOURCE_CLKP
- LL_RCC_UART4_CLKSOURCE_IC9
- LL_RCC_UART4_CLKSOURCE_IC14

- LL_RCC_UART4_CLKSOURCE_LSE
- LL_RCC_UART4_CLKSOURCE_MSI
- LL_RCC_UART4_CLKSOURCE_HSI
- LL_RCC_UART5_CLKSOURCE_PCLK1
- LL_RCC_UART5_CLKSOURCE_CLKP
- LL_RCC_UART5_CLKSOURCE_IC9
- LL_RCC_UART5_CLKSOURCE_IC14
- LL_RCC_UART5_CLKSOURCE_LSE
- LL_RCC_UART5_CLKSOURCE_MSI
- LL_RCC_UART5_CLKSOURCE_HSI
- LL_RCC_UART7_CLKSOURCE_PCLK1
- LL_RCC_UART7_CLKSOURCE_CLKP
- LL_RCC_UART7_CLKSOURCE_IC9
- LL_RCC_UART7_CLKSOURCE_IC14
- LL_RCC_UART7_CLKSOURCE_LSE
- LL_RCC_UART7_CLKSOURCE_MSI
- LL_RCC_UART7_CLKSOURCE_HSI
- LL_RCC_UART8_CLKSOURCE_PCLK1
- LL_RCC_UART8_CLKSOURCE_CLKP
- LL_RCC_UART8_CLKSOURCE_IC9
- LL_RCC_UART8_CLKSOURCE_IC14
- LL_RCC_UART8_CLKSOURCE_LSE
- LL_RCC_UART8_CLKSOURCE_MSI
- LL_RCC_UART8_CLKSOURCE_HSI
- LL_RCC_UART9_CLKSOURCE_PCLK2
- LL_RCC_UART9_CLKSOURCE_CLKP
- LL_RCC_UART9_CLKSOURCE_IC9
- LL_RCC_UART9_CLKSOURCE_IC14
- LL_RCC_UART9_CLKSOURCE_LSE
- LL_RCC_UART9_CLKSOURCE_MSI
- LL_RCC_UART9_CLKSOURCE_HSI
- LL_RCC_USART1_CLKSOURCE_PCLK2
- LL_RCC_USART1_CLKSOURCE_CLKP
- LL_RCC_USART1_CLKSOURCE_IC9
- LL_RCC_USART1_CLKSOURCE_IC14
- LL_RCC_USART1_CLKSOURCE_LSE
- LL_RCC_USART1_CLKSOURCE_MSI
- LL_RCC_USART1_CLKSOURCE_HSI
- LL_RCC_USART2_CLKSOURCE_PCLK1
- LL_RCC_USART2_CLKSOURCE_CLKP
- LL_RCC_USART2_CLKSOURCE_IC9
- LL_RCC_USART2_CLKSOURCE_IC14
- LL_RCC_USART2_CLKSOURCE_LSE
- LL_RCC_USART2_CLKSOURCE_MSI
- LL_RCC_USART2_CLKSOURCE_HSI
- LL_RCC_USART3_CLKSOURCE_PCLK1
- LL_RCC_USART3_CLKSOURCE_CLKP
- LL_RCC_USART3_CLKSOURCE_IC9
- LL_RCC_USART3_CLKSOURCE_IC14
- LL_RCC_USART3_CLKSOURCE_LSE

- LL_RCC_USART3_CLKSOURCE_MSI
- LL_RCC_USART3_CLKSOURCE_HSI
- LL_RCC_USART6_CLKSOURCE_PCLK2
- LL_RCC_USART6_CLKSOURCE_CLKP
- LL_RCC_USART6_CLKSOURCE_IC9
- LL_RCC_USART6_CLKSOURCE_IC14
- LL_RCC_USART6_CLKSOURCE_LSE
- LL_RCC_USART6_CLKSOURCE_MSI
- LL_RCC_USART6_CLKSOURCE_HSI
- LL_RCC_USART10_CLKSOURCE_PCLK2
- LL_RCC_USART10_CLKSOURCE_CLKP
- LL_RCC_USART10_CLKSOURCE_IC9
- LL_RCC_USART10_CLKSOURCE_IC14
- LL_RCC_USART10_CLKSOURCE_LSE
- LL_RCC_USART10_CLKSOURCE_MSI
- LL_RCC_USART10_CLKSOURCE_HSI
- LL_RCC_XSPI1_CLKSOURCE_HCLK
- LL_RCC_XSPI1_CLKSOURCE_CLKP
- LL_RCC_XSPI1_CLKSOURCE_IC3
- LL_RCC_XSPI1_CLKSOURCE_IC4
- LL_RCC_XSPI2_CLKSOURCE_HCLK
- LL_RCC_XSPI2_CLKSOURCE_CLKP
- LL_RCC_XSPI2_CLKSOURCE_IC3
- LL_RCC_XSPI2_CLKSOURCE_IC4
- LL_RCC_XSPI3_CLKSOURCE_HCLK
- LL_RCC_XSPI3_CLKSOURCE_CLKP
- LL_RCC_XSPI3_CLKSOURCE_IC3
- LL_RCC_XSPI3_CLKSOURCE_IC4

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR4 * LL_RCC_SetClockSource
- CCIPR6 * LL_RCC_SetClockSource
- CCIPR8 * LL_RCC_SetClockSource
- CCIPR9 * LL_RCC_SetClockSource
- CCIPR12 * LL_RCC_SetClockSource
- CCIPR13 * LL_RCC_SetClockSource
- CCIPR14 * LL_RCC_SetClockSource

LL_RCC_SetADCPrescaler

Function name

__STATIC_INLINE void LL_RCC_SetADCPrescaler (uint32_t Prescaler)

Function description

Set ADC prescaler.

Parameters

- **Prescaler:** This parameter must be a number between Min_Data = 0 and Max_Data = 255 for prescaler 1 to 256

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR1 ADCPRE LL_RCC_SetADCPrescaler

LL_RCC_GetADCPrescaler

Function name

__STATIC_INLINE uint32_t LL_RCC_GetADCPrescaler (void)

Function description

Get ADC prescaler.

Return values

- **Returned:** value between Min_Data = 0 and Max_Data = 255 for prescaler 1 to 256

Reference Manual to LL API cross reference:

- CCIPR1 ADCPRE LL_RCC_GetADCPrescaler

LL_RCC_SetADCClockSource

Function name

__STATIC_INLINE void LL_RCC_SetADCClockSource (uint32_t ClkSource)

Function description

Configure ADCx Kernel clock source.

Parameters

- **ClkSource:** This parameter can be one of the following values:
  - LL_RCC_ADC_CLKSOURCE_HCLK
  - LL_RCC_ADC_CLKSOURCE_CLKP
  - LL_RCC_ADC_CLKSOURCE_IC7
  - LL_RCC_ADC_CLKSOURCE_IC8
  - LL_RCC_ADC_CLKSOURCE_MSI
  - LL_RCC_ADC_CLKSOURCE_HSI
  - LL_RCC_ADC_CLKSOURCE_I2S_CKIN
  - LL_RCC_ADC_CLKSOURCE_TIMG

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR1 ADC12SEL LL_RCC_SetADCClockSource

LL_RCC_SetADFClockSource

Function name

__STATIC_INLINE void LL_RCC_SetADFClockSource (uint32_t ClkSource)

Function description

Configure ADFx Kernel clock source.

Parameters

- **ClkSource:** This parameter can be one of the following values:
  - LL_RCC_ADF1_CLKSOURCE_HCLK
  - LL_RCC_ADF1_CLKSOURCE_CLKP
  - LL_RCC_ADF1_CLKSOURCE_IC7
  - LL_RCC_ADF1_CLKSOURCE_IC8
  - LL_RCC_ADF1_CLKSOURCE_MSI
  - LL_RCC_ADF1_CLKSOURCE_HSI
  - LL_RCC_ADF1_CLKSOURCE_I2S_CKIN
  - LL_RCC_ADF1_CLKSOURCE_TIMG

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR1 ADF1SEL LL_RCC_SetADFClockSource

LL_RCC_SetCLKPClockSource

Function name

__STATIC_INLINE void LL_RCC_SetCLKPClockSource (uint32_t ClkSource)

Function description

Configure CLKP Kernel clock source.

Parameters

- **ClkSource:** This parameter can be one of the following values:
  - LL_RCC_CLKP_CLKSOURCE_HSI
  - LL_RCC_CLKP_CLKSOURCE_MSI
  - LL_RCC_CLKP_CLKSOURCE_HSE
  - LL_RCC_CLKP_CLKSOURCE_IC19
  - LL_RCC_CLKP_CLKSOURCE_IC5
  - LL_RCC_CLKP_CLKSOURCE_IC10
  - LL_RCC_CLKP_CLKSOURCE_IC15
  - LL_RCC_CLKP_CLKSOURCE_IC20

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR7 PERSEL LL_RCC_SetCLKPClockSource

LL_RCC_SetDCMIPPClockSource

Function name

__STATIC_INLINE void LL_RCC_SetDCMIPPClockSource (uint32_t ClkSource)

Function description

Configure DCMIPP Kernel clock source.

Parameters

- **ClkSource:** This parameter can be one of the following values:
  - LL_RCC_DCMIPP_CLKSOURCE_PCLK5
  - LL_RCC_DCMIPP_CLKSOURCE_CLKP
  - LL_RCC_DCMIPP_CLKSOURCE_IC17
  - LL_RCC_DCMIPP_CLKSOURCE_HSI

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR1 DCMIPPSEL LL_RCC_SetDCMIPPClockSource

LL_RCC_SetETHClockSource

Function name

__STATIC_INLINE void LL_RCC_SetETHClockSource (uint32_t ClkSource)

Function description

Configure ETHx Kernel clock source.

Parameters

- **ClkSource:** This parameter can be one of the following values:
  - LL_RCC_ETH1_CLKSOURCE_HCLK
  - LL_RCC_ETH1_CLKSOURCE_CLKP
  - LL_RCC_ETH1_CLKSOURCE_IC12
  - LL_RCC_ETH1_CLKSOURCE_HSE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR2 ETH1CLKSEL LL_RCC_SetETHClockSource

LL_RCC_SetETHPTPClockSource

Function name

__STATIC_INLINE void LL_RCC_SetETHPTPClockSource (uint32_t ClkSource)

Function description

Configure ETHx PTP Kernel clock source.

Parameters

- **ClkSource:** This parameter can be one of the following values:
  - LL_RCC_ETH1PTP_CLKSOURCE_HCLK
  - LL_RCC_ETH1PTP_CLKSOURCE_CLKP
  - LL_RCC_ETH1PTP_CLKSOURCE_IC13
  - LL_RCC_ETH1PTP_CLKSOURCE_HSE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR2 ETH1PTPSEL LL_RCC_SetETHPTPClockSource

LL_RCC_SetETHPHYInterface

Function name

__STATIC_INLINE void LL_RCC_SetETHPHYInterface (uint32_t Interface)

Function description

Configure ETHx PHY interface.

Parameters

- **Interface:** This parameter can be one of the following values:
  - LL_RCC_ETH1PHY_IF_MII
  - LL_RCC_ETH1PHY_IF_RGMII
  - LL_RCC_ETH1PHY_IF_RMII

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR2 ETH1SEL LL_RCC_SetETHPHYInterface

LL_RCC_SetETH1PTPDivider

Function name

__STATIC_INLINE void LL_RCC_SetETH1PTPDivider (uint32_t Divider)

Function description

Set ETH1 PTP Kernel clock divider.

Parameters

- **Divider:** This parameter can be one of the following values:
  - LL_RCC_ETH1PTP_DIV_1
  - LL_RCC_ETH1PTP_DIV_2
  - LL_RCC_ETH1PTP_DIV_3
  - LL_RCC_ETH1PTP_DIV_4
  - LL_RCC_ETH1PTP_DIV_5
  - LL_RCC_ETH1PTP_DIV_6
  - LL_RCC_ETH1PTP_DIV_7
  - LL_RCC_ETH1PTP_DIV_8
  - LL_RCC_ETH1PTP_DIV_9
  - LL_RCC_ETH1PTP_DIV_10
  - LL_RCC_ETH1PTP_DIV_11
  - LL_RCC_ETH1PTP_DIV_12
  - LL_RCC_ETH1PTP_DIV_13
  - LL_RCC_ETH1PTP_DIV_14
  - LL_RCC_ETH1PTP_DIV_15
  - LL_RCC_ETH1PTP_DIV_16

Return values

- **None.:**

Reference Manual to LL API cross reference:

- CCIPR2 ETH1PTPDIV LL_RCC_SetETH1PTPDivider

LL_RCC_GetETH1PTPDivider

Function name

__STATIC_INLINE uint32_t LL_RCC_GetETH1PTPDivider (void)

Function description

Get ETH1 PTP Kernel clock divider.

Return values

- **can:** be one of the following values:
  - LL_RCC_ETH1PTP_DIV_1
  - LL_RCC_ETH1PTP_DIV_2
  - LL_RCC_ETH1PTP_DIV_3
  - LL_RCC_ETH1PTP_DIV_4
  - LL_RCC_ETH1PTP_DIV_5
  - LL_RCC_ETH1PTP_DIV_6
  - LL_RCC_ETH1PTP_DIV_7
  - LL_RCC_ETH1PTP_DIV_8
  - LL_RCC_ETH1PTP_DIV_9
  - LL_RCC_ETH1PTP_DIV_10
  - LL_RCC_ETH1PTP_DIV_11
  - LL_RCC_ETH1PTP_DIV_12
  - LL_RCC_ETH1PTP_DIV_13
  - LL_RCC_ETH1PTP_DIV_14
  - LL_RCC_ETH1PTP_DIV_15
  - LL_RCC_ETH1PTP_DIV_16

Reference Manual to LL API cross reference:

- CCIPR2 ETH1PTPDIV LL_RCC_GetETH1PTPDivider

LL_RCC_SetETHREFRXClockSource

Function name

__STATIC_INLINE void LL_RCC_SetETHREFRXClockSource (uint32_t ClkSource)

Function description

Configure ETHx Reference RX Kernel clock source.

Parameters

- **ClkSource:** This parameter can be one of the following values:
  - LL_RCC_ETH1REFRX_CLKSOURCE_EXT
  - LL_RCC_ETH1REFRX_CLKSOURCE_INT

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR2 ETH1REFCLKSEL LL_RCC_SetETHREFRXClockSource

LL_RCC_SetETHREFTXClockSource

Function name

__STATIC_INLINE void LL_RCC_SetETHREFTXClockSource (uint32_t ClkSource)

Function description

Configure ETHx Reference TX Kernel clock source.

Parameters

- **ClkSource:** This parameter can be one of the following values:
  - LL_RCC_ETH1REFTX_CLKSOURCE_EXT
  - LL_RCC_ETH1REFTX_CLKSOURCE_INT

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR2 ETH1GTXCLKSEL LL_RCC_SetETHREFTXClockSource

LL_RCC_SetFDCANClockSource

Function name

__STATIC_INLINE void LL_RCC_SetFDCANClockSource (uint32_t ClkSource)

Function description

Configure FDCANx Kernel clock source.

Parameters

- **ClkSource:** This parameter can be one of the following values:
  - LL_RCC_FDCAN_CLKSOURCE_PCLK1
  - LL_RCC_FDCAN_CLKSOURCE_CLKP
  - LL_RCC_FDCAN_CLKSOURCE_IC19
  - LL_RCC_FDCAN_CLKSOURCE_HSE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR3 FDCANSEL LL_RCC_SetFDCANClockSource

LL_RCC_SetFMCClockSource

Function name

__STATIC_INLINE void LL_RCC_SetFMCClockSource (uint32_t ClkSource)

Function description

Configure FMCx Kernel clock source.

Parameters

- **ClkSource:** This parameter can be one of the following values:
  - LL_RCC_FMC_CLKSOURCE_HCLK
  - LL_RCC_FMC_CLKSOURCE_CLKP
  - LL_RCC_FMC_CLKSOURCE_IC3
  - LL_RCC_FMC_CLKSOURCE_IC4

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR3 FMCSEL LL_RCC_SetFMCClockSource

LL_RCC_SetI2CClockSource

Function name

__STATIC_INLINE void LL_RCC_SetI2CClockSource (uint32_t ClkSource)

Function description

Configure I2Cx clock source.

Parameters

- **ClkSource:** This parameter can be one of the following values:
  - LL_RCC_I2C1_CLKSOURCE_PCLK1
  - LL_RCC_I2C1_CLKSOURCE_CLKP
  - LL_RCC_I2C1_CLKSOURCE_IC10
  - LL_RCC_I2C1_CLKSOURCE_IC15
  - LL_RCC_I2C1_CLKSOURCE_MSI
  - LL_RCC_I2C1_CLKSOURCE_HSI
  - LL_RCC_I2C2_CLKSOURCE_PCLK1
  - LL_RCC_I2C2_CLKSOURCE_CLKP
  - LL_RCC_I2C2_CLKSOURCE_IC10
  - LL_RCC_I2C2_CLKSOURCE_IC15
  - LL_RCC_I2C2_CLKSOURCE_MSI
  - LL_RCC_I2C2_CLKSOURCE_HSI
  - LL_RCC_I2C3_CLKSOURCE_PCLK1
  - LL_RCC_I2C3_CLKSOURCE_CLKP
  - LL_RCC_I2C3_CLKSOURCE_IC10
  - LL_RCC_I2C3_CLKSOURCE_IC15
  - LL_RCC_I2C3_CLKSOURCE_MSI
  - LL_RCC_I2C3_CLKSOURCE_HSI
  - LL_RCC_I2C4_CLKSOURCE_PCLK1
  - LL_RCC_I2C4_CLKSOURCE_CLKP
  - LL_RCC_I2C4_CLKSOURCE_IC10
  - LL_RCC_I2C4_CLKSOURCE_IC15
  - LL_RCC_I2C4_CLKSOURCE_MSI
  - LL_RCC_I2C4_CLKSOURCE_HSI

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR4 I2C1SEL LL_RCC_SetI2CClockSource
-
- CCIPR4 I2C2SEL LL_RCC_SetI2CClockSource
-
- CCIPR4 I2C3SEL LL_RCC_SetI2CClockSource
-
- CCIPR4 I2C4SEL LL_RCC_SetI2CClockSource

LL_RCC_SetI3CClockSource

Function name

__STATIC_INLINE void LL_RCC_SetI3CClockSource (uint32_t ClkSource)

Function description

Configure I3Cx clock source.

Parameters

- **ClkSource:** This parameter can be one of the following values:
  - LL_RCC_I3C1_CLKSOURCE_PCLK1
  - LL_RCC_I3C1_CLKSOURCE_CLKP
  - LL_RCC_I3C1_CLKSOURCE_IC10
  - LL_RCC_I3C1_CLKSOURCE_IC15
  - LL_RCC_I3C1_CLKSOURCE_MSI
  - LL_RCC_I3C1_CLKSOURCE_HSI
  - LL_RCC_I3C2_CLKSOURCE_PCLK1
  - LL_RCC_I3C2_CLKSOURCE_CLKP
  - LL_RCC_I3C2_CLKSOURCE_IC10
  - LL_RCC_I3C2_CLKSOURCE_IC15
  - LL_RCC_I3C2_CLKSOURCE_MSI
  - LL_RCC_I3C2_CLKSOURCE_HSI

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR4 I3C1SEL LL_RCC_SetI3CClockSource
-
- CCIPR4 I3C2SEL LL_RCC_SetI3CClockSource

LL_RCC_SetLPTIMClockSource

Function name

__STATIC_INLINE void LL_RCC_SetLPTIMClockSource (uint32_t ClkSource)

Function description

Configure LPTIMx clock source.

Parameters

- **ClkSource:** This parameter can be one of the following values:
  - LL_RCC_LPTIM1_CLKSOURCE_PCLK1
  - LL_RCC_LPTIM1_CLKSOURCE_CLKP
  - LL_RCC_LPTIM1_CLKSOURCE_IC15
  - LL_RCC_LPTIM1_CLKSOURCE_LSE
  - LL_RCC_LPTIM1_CLKSOURCE_LSI
  - LL_RCC_LPTIM1_CLKSOURCE_TIMG
  - LL_RCC_LPTIM2_CLKSOURCE_PCLK4
  - LL_RCC_LPTIM2_CLKSOURCE_CLKP
  - LL_RCC_LPTIM2_CLKSOURCE_IC15
  - LL_RCC_LPTIM2_CLKSOURCE_LSE
  - LL_RCC_LPTIM2_CLKSOURCE_LSI
  - LL_RCC_LPTIM2_CLKSOURCE_TIMG
  - LL_RCC_LPTIM3_CLKSOURCE_PCLK4
  - LL_RCC_LPTIM3_CLKSOURCE_CLKP
  - LL_RCC_LPTIM3_CLKSOURCE_IC15
  - LL_RCC_LPTIM3_CLKSOURCE_LSE
  - LL_RCC_LPTIM3_CLKSOURCE_LSI
  - LL_RCC_LPTIM3_CLKSOURCE_TIMG
  - LL_RCC_LPTIM4_CLKSOURCE_PCLK4
  - LL_RCC_LPTIM4_CLKSOURCE_CLKP
  - LL_RCC_LPTIM4_CLKSOURCE_IC15
  - LL_RCC_LPTIM4_CLKSOURCE_LSE
  - LL_RCC_LPTIM4_CLKSOURCE_LSI
  - LL_RCC_LPTIM4_CLKSOURCE_TIMG
  - LL_RCC_LPTIM5_CLKSOURCE_PCLK4
  - LL_RCC_LPTIM5_CLKSOURCE_CLKP
  - LL_RCC_LPTIM5_CLKSOURCE_IC15
  - LL_RCC_LPTIM5_CLKSOURCE_LSE
  - LL_RCC_LPTIM5_CLKSOURCE_LSI
  - LL_RCC_LPTIM5_CLKSOURCE_TIMG

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR12 LPTIM1SEL LL_RCC_SetLPTIMClockSource
- CCIPR12 LPTIM2SEL LL_RCC_SetLPTIMClockSource
- CCIPR12 LPTIM3SEL LL_RCC_SetLPTIMClockSource
- CCIPR12 LPTIM4SEL LL_RCC_SetLPTIMClockSource
- CCIPR12 LPTIM5SEL LL_RCC_SetLPTIMClockSource

LL_RCC_SetLPUARTClockSource

Function name

__STATIC_INLINE void LL_RCC_SetLPUARTClockSource (uint32_t ClkSource)

Function description

Configure LPUARTx clock source.

Parameters

- **ClkSource:** This parameter can be one of the following values:
  - LL_RCC_LPUART1_CLKSOURCE_PCLK4
  - LL_RCC_LPUART1_CLKSOURCE_CLKP
  - LL_RCC_LPUART1_CLKSOURCE_IC9
  - LL_RCC_LPUART1_CLKSOURCE_IC14
  - LL_RCC_LPUART1_CLKSOURCE_LSE
  - LL_RCC_LPUART1_CLKSOURCE_MSI
  - LL_RCC_LPUART1_CLKSOURCE_HSI

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR14 LPUART1SEL LL_RCC_SetLPUARTClockSource

LL_RCC_SetLTDCClockSource

Function name

__STATIC_INLINE void LL_RCC_SetLTDCClockSource (uint32_t ClkSource)

Function description

Configure LTDC clock source.

Parameters

- **ClkSource:** This parameter can be one of the following values:
  - LL_RCC_LTDC_CLKSOURCE_PCLK5
  - LL_RCC_LTDC_CLKSOURCE_CLKP
  - LL_RCC_LTDC_CLKSOURCE_IC16
  - LL_RCC_LTDC_CLKSOURCE_HSI

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR4 LTDCSEL LL_RCC_SetLTDCClockSource

LL_RCC_SetMDFClockSource

Function name

__STATIC_INLINE void LL_RCC_SetMDFClockSource (uint32_t ClkSource)

Function description

Configure MDFx Kernel clock source.

Parameters

- **ClkSource:** This parameter can be one of the following values:
  - LL_RCC_MDF1_CLKSOURCE_HCLK
  - LL_RCC_MDF1_CLKSOURCE_CLKP
  - LL_RCC_MDF1_CLKSOURCE_IC7
  - LL_RCC_MDF1_CLKSOURCE_IC8
  - LL_RCC_MDF1_CLKSOURCE_MSI
  - LL_RCC_MDF1_CLKSOURCE_HSI
  - LL_RCC_MDF1_CLKSOURCE_I2S_CKIN
  - LL_RCC_MDF1_CLKSOURCE_TIMG

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR5 MDF1SEL LL_RCC_SetMDFClockSource

LL_RCC_SetOTGPHYClockSource

Function name

__STATIC_INLINE void LL_RCC_SetOTGPHYClockSource (uint32_t ClkSource)

Function description

Configure OTGPHY clock source.

Parameters

- **ClkSource:** This parameter can be one of the following values:
  - LL_RCC_OTGPHY1_CLKSOURCE_HSE_DIV_2
  - LL_RCC_OTGPHY1_CLKSOURCE_CLKP
  - LL_RCC_OTGPHY1_CLKSOURCE_IC15
  - LL_RCC_OTGPHY1_CLKSOURCE_HSE_DIV_2_OSC
  - LL_RCC_OTGPHY2_CLKSOURCE_HSE_DIV_2
  - LL_RCC_OTGPHY2_CLKSOURCE_CLKP
  - LL_RCC_OTGPHY2_CLKSOURCE_IC15
  - LL_RCC_OTGPHY2_CLKSOURCE_HSE_DIV_2_OSC

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR6 OTGPHY1SEL LL_RCC_SetOTGPHYClockSource
- CCIPR6 OTGPHY2SEL LL_RCC_SetOTGPHYClockSource

LL_RCC_SetOTGPHYCKREFClockSource

Function name

__STATIC_INLINE void LL_RCC_SetOTGPHYCKREFClockSource (uint32_t ClkSource)

Function description

Configure OTGPHYCKREF clock source.

Parameters

- **ClkSource:** This parameter can be one of the following values:
  - LL_RCC_OTGPHY1CKREF_CLKSOURCE_OTGPHY1
  - LL_RCC_OTGPHY1CKREF_CLKSOURCE_HSE_DIV_2_OSC
  - LL_RCC_OTGPHY2CKREF_CLKSOURCE_OTGPHY2
  - LL_RCC_OTGPHY2CKREF_CLKSOURCE_HSE_DIV_2_OSC

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR6 OTGPHY1CKREFSEL LL_RCC_SetOTGPHYCKREFClockSource
- CCIPR6 OTGPHY2CKREFSEL LL_RCC_SetOTGPHYCKREFClockSource

LL_RCC_SetPSSIClockSource

Function name

__STATIC_INLINE void LL_RCC_SetPSSIClockSource (uint32_t ClkSource)

Function description

Configure PSSI clock source.

Parameters

- **ClkSource:** This parameter can be one of the following values:
  - LL_RCC_PSSI_CLKSOURCE_HCLK
  - LL_RCC_PSSI_CLKSOURCE_CLKP
  - LL_RCC_PSSI_CLKSOURCE_IC20
  - LL_RCC_PSSI_CLKSOURCE_HSI

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR7 PSSISEL LL_RCC_SetPSSIClockSource

LL_RCC_SetSAIClockSource

Function name

__STATIC_INLINE void LL_RCC_SetSAIClockSource (uint32_t ClkSource)

Function description

Configure SAIx clock source.

Parameters

- **ClkSource:** This parameter can be one of the following values:
  - LL_RCC_SAI1_CLKSOURCE_PCLK2
  - LL_RCC_SAI1_CLKSOURCE_CLKP
  - LL_RCC_SAI1_CLKSOURCE_IC7
  - LL_RCC_SAI1_CLKSOURCE_IC8
  - LL_RCC_SAI1_CLKSOURCE_MSI
  - LL_RCC_SAI1_CLKSOURCE_HSI
  - LL_RCC_SAI1_CLKSOURCE_I2S_CKIN
  - LL_RCC_SAI1_CLKSOURCE_SPDIFRX1
  - LL_RCC_SAI2_CLKSOURCE_PCLK2
  - LL_RCC_SAI2_CLKSOURCE_CLKP
  - LL_RCC_SAI2_CLKSOURCE_IC7
  - LL_RCC_SAI2_CLKSOURCE_IC8
  - LL_RCC_SAI2_CLKSOURCE_MSI
  - LL_RCC_SAI2_CLKSOURCE_HSI
  - LL_RCC_SAI2_CLKSOURCE_I2S_CKIN
  - LL_RCC_SAI2_CLKSOURCE_SPDIFRX1

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR7 SAI1SEL LL_RCC_SetSAIClockSource
- CCIPR7 SAI2SEL LL_RCC_SetSAIClockSource

LL_RCC_SetSDMMCClockSource

Function name

__STATIC_INLINE void LL_RCC_SetSDMMCClockSource (uint32_t ClkSource)

Function description

Configure SDMMCx clock source.

Parameters

- **ClkSource:** This parameter can be one of the following values:
  - LL_RCC_SDMMC1_CLKSOURCE_HCLK
  - LL_RCC_SDMMC1_CLKSOURCE_CLKP
  - LL_RCC_SDMMC1_CLKSOURCE_IC4
  - LL_RCC_SDMMC1_CLKSOURCE_IC5
  - LL_RCC_SDMMC2_CLKSOURCE_HCLK
  - LL_RCC_SDMMC2_CLKSOURCE_CLKP
  - LL_RCC_SDMMC2_CLKSOURCE_IC4
  - LL_RCC_SDMMC2_CLKSOURCE_IC5

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR8 SDMMC1SEL LL_RCC_SetSDMMCClockSource
- CCIPR8 SDMMC2SEL LL_RCC_SetSDMMCClockSource

LL_RCC_SetSPDIFRXClockSource

Function name

__STATIC_INLINE void LL_RCC_SetSPDIFRXClockSource (uint32_t ClkSource)

Function description

Configure SPDIFRX1 Kernel clock source.

Parameters

- **ClkSource:** This parameter can be one of the following values:
  - LL_RCC_SPDIFRX1_CLKSOURCE_PCLK1
  - LL_RCC_SPDIFRX1_CLKSOURCE_CLKP
  - LL_RCC_SPDIFRX1_CLKSOURCE_IC7
  - LL_RCC_SPDIFRX1_CLKSOURCE_IC8
  - LL_RCC_SPDIFRX1_CLKSOURCE_MSI
  - LL_RCC_SPDIFRX1_CLKSOURCE_HSI
  - LL_RCC_SPDIFRX1_CLKSOURCE_I2S_CKIN

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR9 SPDIFRX1SEL LL_RCC_SetSPDIFRXClockSource

LL_RCC_SetSPIClockSource

Function name

__STATIC_INLINE void LL_RCC_SetSPIClockSource (uint32_t ClkSource)

Function description

Configure SPIx Kernel clock source.

Parameters

- **ClkSource:** This parameter can be one of the following values:
  - LL_RCC_SPI1_CLKSOURCE_PCLK2
  - LL_RCC_SPI1_CLKSOURCE_CLKP
  - LL_RCC_SPI1_CLKSOURCE_IC8
  - LL_RCC_SPI1_CLKSOURCE_IC9
  - LL_RCC_SPI1_CLKSOURCE_MSI
  - LL_RCC_SPI1_CLKSOURCE_HSI
  - LL_RCC_SPI1_CLKSOURCE_I2S_CKIN
  - LL_RCC_SPI2_CLKSOURCE_PCLK1
  - LL_RCC_SPI2_CLKSOURCE_CLKP
  - LL_RCC_SPI2_CLKSOURCE_IC8
  - LL_RCC_SPI2_CLKSOURCE_IC9
  - LL_RCC_SPI2_CLKSOURCE_MSI
  - LL_RCC_SPI2_CLKSOURCE_HSI
  - LL_RCC_SPI2_CLKSOURCE_I2S_CKIN
  - LL_RCC_SPI3_CLKSOURCE_PCLK1
  - LL_RCC_SPI3_CLKSOURCE_CLKP
  - LL_RCC_SPI3_CLKSOURCE_IC8
  - LL_RCC_SPI3_CLKSOURCE_IC9
  - LL_RCC_SPI3_CLKSOURCE_MSI
  - LL_RCC_SPI3_CLKSOURCE_HSI
  - LL_RCC_SPI3_CLKSOURCE_I2S_CKIN
  - LL_RCC_SPI4_CLKSOURCE_PCLK2
  - LL_RCC_SPI4_CLKSOURCE_CLKP
  - LL_RCC_SPI4_CLKSOURCE_IC9
  - LL_RCC_SPI4_CLKSOURCE_IC14
  - LL_RCC_SPI4_CLKSOURCE_MSI
  - LL_RCC_SPI4_CLKSOURCE_HSI
  - LL_RCC_SPI4_CLKSOURCE_HSE
  - LL_RCC_SPI5_CLKSOURCE_PCLK2
  - LL_RCC_SPI5_CLKSOURCE_CLKP
  - LL_RCC_SPI5_CLKSOURCE_IC9
  - LL_RCC_SPI5_CLKSOURCE_IC14
  - LL_RCC_SPI5_CLKSOURCE_MSI
  - LL_RCC_SPI5_CLKSOURCE_HSI
  - LL_RCC_SPI5_CLKSOURCE_HSE
  - LL_RCC_SPI6_CLKSOURCE_PCLK4
  - LL_RCC_SPI6_CLKSOURCE_CLKP
  - LL_RCC_SPI6_CLKSOURCE_IC8
  - LL_RCC_SPI6_CLKSOURCE_IC9
  - LL_RCC_SPI6_CLKSOURCE_MSI
  - LL_RCC_SPI6_CLKSOURCE_HSI
  - LL_RCC_SPI6_CLKSOURCE_I2S_CKIN

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR9 SPI1SEL LL_RCC_SetSPIClockSource
- CCIPR9 SPI2SEL LL_RCC_SetSPIClockSource
- CCIPR9 SPI3SEL LL_RCC_SetSPIClockSource
- CCIPR9 SPI4SEL LL_RCC_SetSPIClockSource
- CCIPR9 SPI5SEL LL_RCC_SetSPIClockSource
- CCIPR9 SPI6SEL LL_RCC_SetSPIClockSource

LL_RCC_SetUSARTClockSource

Function name

__STATIC_INLINE void LL_RCC_SetUSARTClockSource (uint32_t ClkSource)

Function description

Configure USARTx clock source.

Parameters

- **ClkSource:** This parameter can be one of the following values:
  - LL_RCC_USART1_CLKSOURCE_PCLK2
  - LL_RCC_USART1_CLKSOURCE_CLKP
  - LL_RCC_USART1_CLKSOURCE_IC9
  - LL_RCC_USART1_CLKSOURCE_IC14
  - LL_RCC_USART1_CLKSOURCE_LSE
  - LL_RCC_USART1_CLKSOURCE_MSI
  - LL_RCC_USART1_CLKSOURCE_HSI
  - LL_RCC_USART2_CLKSOURCE_PCLK1
  - LL_RCC_USART2_CLKSOURCE_CLKP
  - LL_RCC_USART2_CLKSOURCE_IC9
  - LL_RCC_USART2_CLKSOURCE_IC14
  - LL_RCC_USART2_CLKSOURCE_LSE
  - LL_RCC_USART2_CLKSOURCE_MSI
  - LL_RCC_USART2_CLKSOURCE_HSI
  - LL_RCC_USART3_CLKSOURCE_PCLK1
  - LL_RCC_USART3_CLKSOURCE_CLKP
  - LL_RCC_USART3_CLKSOURCE_IC9
  - LL_RCC_USART3_CLKSOURCE_IC14
  - LL_RCC_USART3_CLKSOURCE_LSE
  - LL_RCC_USART3_CLKSOURCE_MSI
  - LL_RCC_USART3_CLKSOURCE_HSI
  - LL_RCC_USART6_CLKSOURCE_PCLK2
  - LL_RCC_USART6_CLKSOURCE_CLKP
  - LL_RCC_USART6_CLKSOURCE_IC9
  - LL_RCC_USART6_CLKSOURCE_IC14
  - LL_RCC_USART6_CLKSOURCE_LSE
  - LL_RCC_USART6_CLKSOURCE_MSI
  - LL_RCC_USART6_CLKSOURCE_HSI
  - LL_RCC_USART10_CLKSOURCE_PCLK2
  - LL_RCC_USART10_CLKSOURCE_CLKP
  - LL_RCC_USART10_CLKSOURCE_IC9
  - LL_RCC_USART10_CLKSOURCE_IC14
  - LL_RCC_USART10_CLKSOURCE_LSE
  - LL_RCC_USART10_CLKSOURCE_MSI
  - LL_RCC_USART10_CLKSOURCE_HSI

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR13 USART1SEL LL_RCC_SetUSARTClockSource
- CCIPR13 USART2SEL LL_RCC_SetUSARTClockSource
- CCIPR13 USART3SEL LL_RCC_SetUSARTClockSource
- CCIPR13 USART6SEL LL_RCC_SetUSARTClockSource
- CCIPR14 USART10SEL LL_RCC_SetUSARTClockSource

LL_RCC_SetUARTClockSource

Function name

__STATIC_INLINE void LL_RCC_SetUARTClockSource (uint32_t ClkSource)

Function description

Configure UARTx clock source.

Parameters

- **ClkSource:** This parameter can be one of the following values:
  - LL_RCC_UART4_CLKSOURCE_PCLK1
  - LL_RCC_UART4_CLKSOURCE_CLKP
  - LL_RCC_UART4_CLKSOURCE_IC9
  - LL_RCC_UART4_CLKSOURCE_IC14
  - LL_RCC_UART4_CLKSOURCE_LSE
  - LL_RCC_UART4_CLKSOURCE_MSI
  - LL_RCC_UART4_CLKSOURCE_HSI
  - LL_RCC_UART5_CLKSOURCE_PCLK1
  - LL_RCC_UART5_CLKSOURCE_CLKP
  - LL_RCC_UART5_CLKSOURCE_IC9
  - LL_RCC_UART5_CLKSOURCE_IC14
  - LL_RCC_UART5_CLKSOURCE_LSE
  - LL_RCC_UART5_CLKSOURCE_MSI
  - LL_RCC_UART5_CLKSOURCE_HSI
  - LL_RCC_UART7_CLKSOURCE_PCLK1
  - LL_RCC_UART7_CLKSOURCE_CLKP
  - LL_RCC_UART7_CLKSOURCE_IC9
  - LL_RCC_UART7_CLKSOURCE_IC14
  - LL_RCC_UART7_CLKSOURCE_LSE
  - LL_RCC_UART7_CLKSOURCE_MSI
  - LL_RCC_UART7_CLKSOURCE_HSI
  - LL_RCC_UART8_CLKSOURCE_PCLK1
  - LL_RCC_UART8_CLKSOURCE_CLKP
  - LL_RCC_UART8_CLKSOURCE_IC9
  - LL_RCC_UART8_CLKSOURCE_IC14
  - LL_RCC_UART8_CLKSOURCE_LSE
  - LL_RCC_UART8_CLKSOURCE_MSI
  - LL_RCC_UART8_CLKSOURCE_HSI
  - LL_RCC_UART9_CLKSOURCE_PCLK2
  - LL_RCC_UART9_CLKSOURCE_CLKP
  - LL_RCC_UART9_CLKSOURCE_IC9
  - LL_RCC_UART9_CLKSOURCE_IC14
  - LL_RCC_UART9_CLKSOURCE_LSE
  - LL_RCC_UART9_CLKSOURCE_MSI
  - LL_RCC_UART9_CLKSOURCE_HSI

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR13 UART4SEL LL_RCC_SetUARTClockSource
- CCIPR13 UART5SEL LL_RCC_SetUARTClockSource
- CCIPR13 UART7SEL LL_RCC_SetUARTClockSource
- CCIPR13 UART8SEL LL_RCC_SetUARTClockSource
- CCIPR14 UART9SEL LL_RCC_SetUARTClockSource

LL_RCC_SetUSBClockSource

Function name

__STATIC_INLINE void LL_RCC_SetUSBClockSource (uint32_t ClkSource)

Function description

Configure USBx clock source.

Parameters

- **ClkSource:** This parameter can be one of the following values:
  - LL_RCC_OTGPHY1_CLKSOURCE_HSE_DIV_2
  - LL_RCC_OTGPHY1_CLKSOURCE_CLKP
  - LL_RCC_OTGPHY1_CLKSOURCE_IC15
  - LL_RCC_OTGPHY1_CLKSOURCE_HSE_DIV_2_OSC
  - LL_RCC_OTGPHY1CKREF_CLKSOURCE_OTGPHY1
  - LL_RCC_OTGPHY1CKREF_CLKSOURCE_HSE_DIV_2_OSC
  - LL_RCC_OTGPHY2_CLKSOURCE_HSE_DIV_2
  - LL_RCC_OTGPHY2_CLKSOURCE_CLKP
  - LL_RCC_OTGPHY2_CLKSOURCE_IC15
  - LL_RCC_OTGPHY2_CLKSOURCE_HSE_DIV_2_OSC
  - LL_RCC_OTGPHY2CKREF_CLKSOURCE_OTGPHY2
  - LL_RCC_OTGPHY2CKREF_CLKSOURCE_HSE_DIV_2_OSC

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR6 OTGPHY1SEL LL_RCC_GetUSBClockSource
- CCIPR6 OTGPHY1CKREFSEL LL_RCC_GetUSBClockSource
- CCIPR6 OTGPHY2SEL LL_RCC_GetUSBClockSource
- CCIPR6 OTGPHY2CKREFSEL LL_RCC_GetUSBClockSource
-

LL_RCC_SetXSPIClockSource

Function name

__STATIC_INLINE void LL_RCC_SetXSPIClockSource (uint32_t ClkSource)

Function description

Configure XSPIx Kernel clock source.

Parameters

- **ClkSource:** This parameter can be one of the following values:
  - LL_RCC_XSPI1_CLKSOURCE_HCLK
  - LL_RCC_XSPI1_CLKSOURCE_CLKP
  - LL_RCC_XSPI1_CLKSOURCE_IC3
  - LL_RCC_XSPI1_CLKSOURCE_IC4
  - LL_RCC_XSPI2_CLKSOURCE_HCLK
  - LL_RCC_XSPI2_CLKSOURCE_CLKP
  - LL_RCC_XSPI2_CLKSOURCE_IC3
  - LL_RCC_XSPI2_CLKSOURCE_IC4
  - LL_RCC_XSPI3_CLKSOURCE_HCLK
  - LL_RCC_XSPI3_CLKSOURCE_CLKP
  - LL_RCC_XSPI3_CLKSOURCE_IC3
  - LL_RCC_XSPI3_CLKSOURCE_IC4

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR6 XSPI1SEL LL_RCC_SetXSPIClockSource
- CCIPR6 XSPI2SEL LL_RCC_SetXSPIClockSource
- CCIPR6 XSPI3SEL LL_RCC_SetXSPIClockSource

LL_RCC_GetClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetClockSource (uint32_t Periph)

Function description

Get periph clock source.

Parameters

- **Periph:** This parameter can be one of the following values:
  - LL_RCC_I2C1_CLKSOURCE
  - LL_RCC_I2C2_CLKSOURCE
  - LL_RCC_I2C3_CLKSOURCE
  - LL_RCC_I2C4_CLKSOURCE
  - LL_RCC_I3C1_CLKSOURCE
  - LL_RCC_I3C2_CLKSOURCE
  - LL_RCC_LPTIM1_CLKSOURCE
  - LL_RCC_LPTIM2_CLKSOURCE
  - LL_RCC_LPTIM3_CLKSOURCE
  - LL_RCC_LPTIM4_CLKSOURCE
  - LL_RCC_LPTIM5_CLKSOURCE
  - LL_RCC_OTGPHY1_CLKSOURCE
  - LL_RCC_OTGPHY1CKREF_CLKSOURCE
  - LL_RCC_OTGPHY2_CLKSOURCE
  - LL_RCC_OTGPHY2CKREF_CLKSOURCE
  - LL_RCC_SAI1_CLKSOURCE
  - LL_RCC_SAI2_CLKSOURCE
  - LL_RCC_SDMMC1_CLKSOURCE
  - LL_RCC_SDMMC2_CLKSOURCE
  - LL_RCC_SPI1_CLKSOURCE
  - LL_RCC_SPI2_CLKSOURCE
  - LL_RCC_SPI3_CLKSOURCE
  - LL_RCC_SPI4_CLKSOURCE
  - LL_RCC_SPI5_CLKSOURCE
  - LL_RCC_SPI6_CLKSOURCE
  - LL_RCC_UART4_CLKSOURCE
  - LL_RCC_UART5_CLKSOURCE
  - LL_RCC_UART7_CLKSOURCE
  - LL_RCC_UART8_CLKSOURCE
  - LL_RCC_UART9_CLKSOURCE
  - LL_RCC_USART1_CLKSOURCE
  - LL_RCC_USART2_CLKSOURCE
  - LL_RCC_USART3_CLKSOURCE
  - LL_RCC_USART6_CLKSOURCE
  - LL_RCC_USART10_CLKSOURCE
  - LL_RCC_XSPI1_CLKSOURCE
  - LL_RCC_XSPI2_CLKSOURCE
  - LL_RCC_XSPI3_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_I2C1_CLKSOURCE_PCLK1
  - LL_RCC_I2C1_CLKSOURCE_CLKP
  - LL_RCC_I2C1_CLKSOURCE_IC10
  - LL_RCC_I2C1_CLKSOURCE_IC15
  - LL_RCC_I2C1_CLKSOURCE_MSI
  - LL_RCC_I2C1_CLKSOURCE_HSI
  - LL_RCC_I2C2_CLKSOURCE_PCLK1
  - LL_RCC_I2C2_CLKSOURCE_CLKP
  - LL_RCC_I2C2_CLKSOURCE_IC10
  - LL_RCC_I2C2_CLKSOURCE_IC15
  - LL_RCC_I2C2_CLKSOURCE_MSI
  - LL_RCC_I2C2_CLKSOURCE_HSI
  - LL_RCC_I2C3_CLKSOURCE_PCLK1
  - LL_RCC_I2C3_CLKSOURCE_CLKP
  - LL_RCC_I2C3_CLKSOURCE_IC10
  - LL_RCC_I2C3_CLKSOURCE_IC15
  - LL_RCC_I2C3_CLKSOURCE_MSI
  - LL_RCC_I2C3_CLKSOURCE_HSI
  - LL_RCC_I2C4_CLKSOURCE_PCLK1
  - LL_RCC_I2C4_CLKSOURCE_CLKP
  - LL_RCC_I2C4_CLKSOURCE_IC10
  - LL_RCC_I2C4_CLKSOURCE_IC15
  - LL_RCC_I2C4_CLKSOURCE_MSI
  - LL_RCC_I2C4_CLKSOURCE_HSI
  - LL_RCC_I3C1_CLKSOURCE_PCLK1
  - LL_RCC_I3C1_CLKSOURCE_CLKP
  - LL_RCC_I3C1_CLKSOURCE_IC10
  - LL_RCC_I3C1_CLKSOURCE_IC15
  - LL_RCC_I3C1_CLKSOURCE_MSI
  - LL_RCC_I3C1_CLKSOURCE_HSI
  - LL_RCC_I3C2_CLKSOURCE_PCLK1
  - LL_RCC_I3C2_CLKSOURCE_CLKP
  - LL_RCC_I3C2_CLKSOURCE_IC10
  - LL_RCC_I3C2_CLKSOURCE_IC15
  - LL_RCC_I3C2_CLKSOURCE_MSI
  - LL_RCC_I3C2_CLKSOURCE_HSI
  - LL_RCC_LPTIM1_CLKSOURCE_PCLK1
  - LL_RCC_LPTIM1_CLKSOURCE_CLKP
  - LL_RCC_LPTIM1_CLKSOURCE_IC15
  - LL_RCC_LPTIM1_CLKSOURCE_LSE
  - LL_RCC_LPTIM1_CLKSOURCE_LSI
  - LL_RCC_LPTIM1_CLKSOURCE_TIMG
  - LL_RCC_LPTIM2_CLKSOURCE_PCLK4
  - LL_RCC_LPTIM2_CLKSOURCE_CLKP
  - LL_RCC_LPTIM2_CLKSOURCE_IC15
  - LL_RCC_LPTIM2_CLKSOURCE_LSE
  - LL_RCC_LPTIM2_CLKSOURCE_LSI
  - LL_RCC_LPTIM2_CLKSOURCE_TIMG

- LL_RCC_LPTIM3_CLKSOURCE_PCLK4
- LL_RCC_LPTIM3_CLKSOURCE_CLKP
- LL_RCC_LPTIM3_CLKSOURCE_IC15
- LL_RCC_LPTIM3_CLKSOURCE_LSE
- LL_RCC_LPTIM3_CLKSOURCE_LSI
- LL_RCC_LPTIM3_CLKSOURCE_TIMG
- LL_RCC_LPTIM4_CLKSOURCE_PCLK4
- LL_RCC_LPTIM4_CLKSOURCE_CLKP
- LL_RCC_LPTIM4_CLKSOURCE_IC15
- LL_RCC_LPTIM4_CLKSOURCE_LSE
- LL_RCC_LPTIM4_CLKSOURCE_LSI
- LL_RCC_LPTIM4_CLKSOURCE_TIMG
- LL_RCC_LPTIM5_CLKSOURCE_PCLK4
- LL_RCC_LPTIM5_CLKSOURCE_CLKP
- LL_RCC_LPTIM5_CLKSOURCE_IC15
- LL_RCC_LPTIM5_CLKSOURCE_LSE
- LL_RCC_LPTIM5_CLKSOURCE_LSI
- LL_RCC_LPTIM5_CLKSOURCE_TIMG
- LL_RCC_OTGPHY1_CLKSOURCE_HSE_DIV_2
- LL_RCC_OTGPHY1_CLKSOURCE_CLKP
- LL_RCC_OTGPHY1_CLKSOURCE_IC15
- LL_RCC_OTGPHY1_CLKSOURCE_HSE_DIV_2_OSC
- LL_RCC_OTGPHY1CKREF_CLKSOURCE_OTGPHY1
- LL_RCC_OTGPHY1CKREF_CLKSOURCE_HSE_DIV_2_OSC
- LL_RCC_OTGPHY2_CLKSOURCE_HSE_DIV_2
- LL_RCC_OTGPHY2_CLKSOURCE_CLKP
- LL_RCC_OTGPHY2_CLKSOURCE_IC15
- LL_RCC_OTGPHY2_CLKSOURCE_HSE_DIV_2_OSC
- LL_RCC_OTGPHY2CKREF_CLKSOURCE_OTGPHY2
- LL_RCC_OTGPHY2CKREF_CLKSOURCE_HSE_DIV_2_OSC
- LL_RCC_SAI1_CLKSOURCE_PCLK2
- LL_RCC_SAI1_CLKSOURCE_CLKP
- LL_RCC_SAI1_CLKSOURCE_IC7
- LL_RCC_SAI1_CLKSOURCE_IC8
- LL_RCC_SAI1_CLKSOURCE_MSI
- LL_RCC_SAI1_CLKSOURCE_HSI
- LL_RCC_SAI1_CLKSOURCE_I2S_CKIN
- LL_RCC_SAI1_CLKSOURCE_SPDIFRX1
- LL_RCC_SAI2_CLKSOURCE_PCLK2
- LL_RCC_SAI2_CLKSOURCE_CLKP
- LL_RCC_SAI2_CLKSOURCE_IC7
- LL_RCC_SAI2_CLKSOURCE_IC8
- LL_RCC_SAI2_CLKSOURCE_MSI
- LL_RCC_SAI2_CLKSOURCE_HSI
- LL_RCC_SAI2_CLKSOURCE_I2S_CKIN
- LL_RCC_SAI2_CLKSOURCE_SPDIFRX1
- LL_RCC_SDMMC1_CLKSOURCE_HCLK
- LL_RCC_SDMMC1_CLKSOURCE_CLKP
- LL_RCC_SDMMC1_CLKSOURCE_IC4
- LL_RCC_SDMMC1_CLKSOURCE_IC5

- LL_RCC_SDMMC2_CLKSOURCE_HCLK
- LL_RCC_SDMMC2_CLKSOURCE_CLKP
- LL_RCC_SDMMC2_CLKSOURCE_IC4
- LL_RCC_SDMMC2_CLKSOURCE_IC5
- LL_RCC_SPI1_CLKSOURCE_PCLK2
- LL_RCC_SPI1_CLKSOURCE_CLKP
- LL_RCC_SPI1_CLKSOURCE_IC8
- LL_RCC_SPI1_CLKSOURCE_IC9
- LL_RCC_SPI1_CLKSOURCE_MSI
- LL_RCC_SPI1_CLKSOURCE_HSI
- LL_RCC_SPI1_CLKSOURCE_I2S_CKIN
- LL_RCC_SPI2_CLKSOURCE_PCLK1
- LL_RCC_SPI2_CLKSOURCE_CLKP
- LL_RCC_SPI2_CLKSOURCE_IC8
- LL_RCC_SPI2_CLKSOURCE_IC9
- LL_RCC_SPI2_CLKSOURCE_MSI
- LL_RCC_SPI2_CLKSOURCE_HSI
- LL_RCC_SPI2_CLKSOURCE_I2S_CKIN
- LL_RCC_SPI3_CLKSOURCE_PCLK1
- LL_RCC_SPI3_CLKSOURCE_CLKP
- LL_RCC_SPI3_CLKSOURCE_IC8
- LL_RCC_SPI3_CLKSOURCE_IC9
- LL_RCC_SPI3_CLKSOURCE_MSI
- LL_RCC_SPI3_CLKSOURCE_HSI
- LL_RCC_SPI3_CLKSOURCE_I2S_CKIN
- LL_RCC_SPI4_CLKSOURCE_PCLK2
- LL_RCC_SPI4_CLKSOURCE_CLKP
- LL_RCC_SPI4_CLKSOURCE_IC9
- LL_RCC_SPI4_CLKSOURCE_IC14
- LL_RCC_SPI4_CLKSOURCE_MSI
- LL_RCC_SPI4_CLKSOURCE_HSI
- LL_RCC_SPI4_CLKSOURCE_HSE
- LL_RCC_SPI5_CLKSOURCE_PCLK2
- LL_RCC_SPI5_CLKSOURCE_CLKP
- LL_RCC_SPI5_CLKSOURCE_IC9
- LL_RCC_SPI5_CLKSOURCE_IC14
- LL_RCC_SPI5_CLKSOURCE_MSI
- LL_RCC_SPI5_CLKSOURCE_HSI
- LL_RCC_SPI5_CLKSOURCE_HSE
- LL_RCC_SPI6_CLKSOURCE_PCLK4
- LL_RCC_SPI6_CLKSOURCE_CLKP
- LL_RCC_SPI6_CLKSOURCE_IC8
- LL_RCC_SPI6_CLKSOURCE_IC9
- LL_RCC_SPI6_CLKSOURCE_MSI
- LL_RCC_SPI6_CLKSOURCE_HSI
- LL_RCC_SPI6_CLKSOURCE_I2S_CKIN
- LL_RCC_UART4_CLKSOURCE_PCLK1
- LL_RCC_UART4_CLKSOURCE_CLKP
- LL_RCC_UART4_CLKSOURCE_IC9
- LL_RCC_UART4_CLKSOURCE_IC14

- LL_RCC_UART4_CLKSOURCE_LSE
- LL_RCC_UART4_CLKSOURCE_MSI
- LL_RCC_UART4_CLKSOURCE_HSI
- LL_RCC_UART5_CLKSOURCE_PCLK1
- LL_RCC_UART5_CLKSOURCE_CLKP
- LL_RCC_UART5_CLKSOURCE_IC9
- LL_RCC_UART5_CLKSOURCE_IC14
- LL_RCC_UART5_CLKSOURCE_LSE
- LL_RCC_UART5_CLKSOURCE_MSI
- LL_RCC_UART5_CLKSOURCE_HSI
- LL_RCC_UART7_CLKSOURCE_PCLK1
- LL_RCC_UART7_CLKSOURCE_CLKP
- LL_RCC_UART7_CLKSOURCE_IC9
- LL_RCC_UART7_CLKSOURCE_IC14
- LL_RCC_UART7_CLKSOURCE_LSE
- LL_RCC_UART7_CLKSOURCE_MSI
- LL_RCC_UART7_CLKSOURCE_HSI
- LL_RCC_UART8_CLKSOURCE_PCLK1
- LL_RCC_UART8_CLKSOURCE_CLKP
- LL_RCC_UART8_CLKSOURCE_IC9
- LL_RCC_UART8_CLKSOURCE_IC14
- LL_RCC_UART8_CLKSOURCE_LSE
- LL_RCC_UART8_CLKSOURCE_MSI
- LL_RCC_UART8_CLKSOURCE_HSI
- LL_RCC_UART9_CLKSOURCE_PCLK2
- LL_RCC_UART9_CLKSOURCE_CLKP
- LL_RCC_UART9_CLKSOURCE_IC9
- LL_RCC_UART9_CLKSOURCE_IC14
- LL_RCC_UART9_CLKSOURCE_LSE
- LL_RCC_UART9_CLKSOURCE_MSI
- LL_RCC_UART9_CLKSOURCE_HSI
- LL_RCC_USART1_CLKSOURCE_PCLK2
- LL_RCC_USART1_CLKSOURCE_CLKP
- LL_RCC_USART1_CLKSOURCE_IC9
- LL_RCC_USART1_CLKSOURCE_IC14
- LL_RCC_USART1_CLKSOURCE_LSE
- LL_RCC_USART1_CLKSOURCE_MSI
- LL_RCC_USART1_CLKSOURCE_HSI
- LL_RCC_USART2_CLKSOURCE_PCLK1
- LL_RCC_USART2_CLKSOURCE_CLKP
- LL_RCC_USART2_CLKSOURCE_IC9
- LL_RCC_USART2_CLKSOURCE_IC14
- LL_RCC_USART2_CLKSOURCE_LSE
- LL_RCC_USART2_CLKSOURCE_MSI
- LL_RCC_USART2_CLKSOURCE_HSI
- LL_RCC_USART3_CLKSOURCE_PCLK1
- LL_RCC_USART3_CLKSOURCE_CLKP
- LL_RCC_USART3_CLKSOURCE_IC9
- LL_RCC_USART3_CLKSOURCE_IC14
- LL_RCC_USART3_CLKSOURCE_LSE

- LL_RCC_USART3_CLKSOURCE_MSI
- LL_RCC_USART3_CLKSOURCE_HSI
- LL_RCC_USART6_CLKSOURCE_PCLK2
- LL_RCC_USART6_CLKSOURCE_CLKP
- LL_RCC_USART6_CLKSOURCE_IC9
- LL_RCC_USART6_CLKSOURCE_IC14
- LL_RCC_USART6_CLKSOURCE_LSE
- LL_RCC_USART6_CLKSOURCE_MSI
- LL_RCC_USART6_CLKSOURCE_HSI
- LL_RCC_USART10_CLKSOURCE_PCLK2
- LL_RCC_USART10_CLKSOURCE_CLKP
- LL_RCC_USART10_CLKSOURCE_IC9
- LL_RCC_USART10_CLKSOURCE_IC14
- LL_RCC_USART10_CLKSOURCE_LSE
- LL_RCC_USART10_CLKSOURCE_MSI
- LL_RCC_USART10_CLKSOURCE_HSI
- LL_RCC_XSPI1_CLKSOURCE_HCLK
- LL_RCC_XSPI1_CLKSOURCE_CLKP
- LL_RCC_XSPI1_CLKSOURCE_IC3
- LL_RCC_XSPI1_CLKSOURCE_IC4
- LL_RCC_XSPI2_CLKSOURCE_HCLK
- LL_RCC_XSPI2_CLKSOURCE_CLKP
- LL_RCC_XSPI2_CLKSOURCE_IC3
- LL_RCC_XSPI2_CLKSOURCE_IC4
- LL_RCC_XSPI3_CLKSOURCE_HCLK
- LL_RCC_XSPI3_CLKSOURCE_CLKP
- LL_RCC_XSPI3_CLKSOURCE_IC3
- LL_RCC_XSPI3_CLKSOURCE_IC4

- **None:**

Reference Manual to LL API cross reference:

- CCIPR1 * LL_RCC_GetClockSource
- CCIPR3 * LL_RCC_GetClockSource
- CCIPR4 * LL_RCC_GetClockSource
- CCIPR6 * LL_RCC_GetClockSource
- CCIPR7 * LL_RCC_GetClockSource
- CCIPR8 * LL_RCC_GetClockSource
- CCIPR9 * LL_RCC_GetClockSource
- CCIPR12 * LL_RCC_GetClockSource
- CCIPR13 * LL_RCC_GetClockSource
- CCIPR14 * LL_RCC_GetClockSource

LL_RCC_GetADCClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetADCClockSource (uint32_t Periph)

Function description

Get ADC Kernel clock source.

Parameters

- **Periph:** This parameter can be one of the following values:
  - LL_RCC_ADC_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_ADC_CLKSOURCE_HCLK
  - LL_RCC_ADC_CLKSOURCE_CLKP
  - LL_RCC_ADC_CLKSOURCE_IC7
  - LL_RCC_ADC_CLKSOURCE_IC8
  - LL_RCC_ADC_CLKSOURCE_MSI
  - LL_RCC_ADC_CLKSOURCE_HSI
  - LL_RCC_ADC_CLKSOURCE_I2S_CKIN
  - LL_RCC_ADC_CLKSOURCE_TIMG

Reference Manual to LL API cross reference:

- CCIPR1 ADC12SEL LL_RCC_GetADCClockSource

LL_RCC_GetADFClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetADFClockSource (uint32_t Periph)

Function description

Get ADFx clock source.

Parameters

- **Periph:** This parameter can be one of the following values:
  - LL_RCC_ADF1_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_ADF1_CLKSOURCE_HCLK
  - LL_RCC_ADF1_CLKSOURCE_CLKP
  - LL_RCC_ADF1_CLKSOURCE_IC7
  - LL_RCC_ADF1_CLKSOURCE_IC8
  - LL_RCC_ADF1_CLKSOURCE_MSI
  - LL_RCC_ADF1_CLKSOURCE_HSI
  - LL_RCC_ADF1_CLKSOURCE_I2S_CKIN
  - LL_RCC_ADF1_CLKSOURCE_TIMG

Reference Manual to LL API cross reference:

- CCIPR1 ADF1SEL LL_RCC_GetADFClockSource

LL_RCC_GetCLKPClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetCLKPClockSource (uint32_t Periph)

Function description

Get CLKP Kernel clock source.

Parameters

- **Periph:** This parameter can be one of the following values:
  - LL_RCC_CLKP_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_CLKP_CLKSOURCE_HSI
  - LL_RCC_CLKP_CLKSOURCE_MSI
  - LL_RCC_CLKP_CLKSOURCE_HSE
  - LL_RCC_CLKP_CLKSOURCE_IC19
  - LL_RCC_CLKP_CLKSOURCE_IC5
  - LL_RCC_CLKP_CLKSOURCE_IC10
  - LL_RCC_CLKP_CLKSOURCE_IC15
  - LL_RCC_CLKP_CLKSOURCE_IC20

Reference Manual to LL API cross reference:

- CCIPR7 PERSEL LL_RCC_GetCLKPClockSource

LL_RCC_GetDCMIPPClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetDCMIPPClockSource (uint32_t Periph)

Function description

Get DCMIPP clock source.

Parameters

- **Periph:** This parameter can be one of the following values:
  - LL_RCC_DCMIPP_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_DCMIPP_CLKSOURCE_PCLK5
  - LL_RCC_DCMIPP_CLKSOURCE_CLKP
  - LL_RCC_DCMIPP_CLKSOURCE_IC17
  - LL_RCC_DCMIPP_CLKSOURCE_HSI

Reference Manual to LL API cross reference:

- CCIPR1 DCMIPPSEL LL_RCC_GetDCMIPPClockSource

LL_RCC_GetETHClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetETHClockSource (uint32_t Periph)

Function description

Get ETHx Kernel clock source.

Parameters

- **Periph:** This parameter can be one of the following values:
  - LL_RCC_ETH1_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_ETH1_CLKSOURCE_HCLK
  - LL_RCC_ETH1_CLKSOURCE_CLKP
  - LL_RCC_ETH1_CLKSOURCE_IC12
  - LL_RCC_ETH1_CLKSOURCE_HSE

Reference Manual to LL API cross reference:

- CCIPR2 ETH1CLKSEL LL_RCC_GetETHClockSource

LL_RCC_GetETHPTPClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetETHPTPClockSource (uint32_t Periph)

Function description

Get ETHx PTP Kernel clock source.

Parameters

- **Periph:** This parameter can be one of the following values:
  - LL_RCC_ETH1PTP_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_ETH1PTP_CLKSOURCE_HCLK
  - LL_RCC_ETH1PTP_CLKSOURCE_CLKP
  - LL_RCC_ETH1PTP_CLKSOURCE_IC13
  - LL_RCC_ETH1PTP_CLKSOURCE_HSE

Reference Manual to LL API cross reference:

- CCIPR2 ETH1PTPSEL LL_RCC_GetETHPTPClockSource

LL_RCC_GetETHPHYInterface

Function name

__STATIC_INLINE uint32_t LL_RCC_GetETHPHYInterface (uint32_t Periph)

Function description

Get ETHx PHY interface.

Parameters

- **Periph:** This parameter can be one of the following values:
  - LL_RCC_ETH1PHY_IF

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_ETH1PHY_IF_MII
  - LL_RCC_ETH1PHY_IF_RGMII
  - LL_RCC_ETH1PHY_IF_RMII

Reference Manual to LL API cross reference:

- CCIPR2 ETH1SEL LL_RCC_GetETHPHYInterface

LL_RCC_GetETHREFRXClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetETHREFRXClockSource (uint32_t Periph)

Function description

Get ETHx Reference RX Kernel clock source.

Parameters

- **Periph:** This parameter can be one of the following values:
  - LL_RCC_ETH1REFRX_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_ETH1REFRX_CLKSOURCE_EXT
  - LL_RCC_ETH1REFRX_CLKSOURCE_INT

Reference Manual to LL API cross reference:

- CCIPR2 ETH1REFCLKSEL LL_RCC_GetETHREFRXClockSource

LL_RCC_GetETHREFTXClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetETHREFTXClockSource (uint32_t Periph)

Function description

Get ETHx Reference TX Kernel clock source.

Parameters

- **Periph:** This parameter can be one of the following values:
  - LL_RCC_ETH1REFTX_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_ETH1REFTX_CLKSOURCE_EXT
  - LL_RCC_ETH1REFTX_CLKSOURCE_INT

Reference Manual to LL API cross reference:

- CCIPR2 ETH1GTXCLKSEL LL_RCC_GetETHREFTXClockSource

LL_RCC_GetFDCANClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetFDCANClockSource (uint32_t Periph)

Function description

Get FDCAN Kernel clock source.

Parameters

- **Periph:** This parameter can be one of the following values:
  - LL_RCC_FDCAN_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_FDCAN_CLKSOURCE_PCLK1
  - LL_RCC_FDCAN_CLKSOURCE_CLKP
  - LL_RCC_FDCAN_CLKSOURCE_IC19
  - LL_RCC_FDCAN_CLKSOURCE_HSE

Reference Manual to LL API cross reference:

- CCIPR3 FDCANSEL LL_RCC_GetFDCANClockSource

LL_RCC_GetFMCClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetFMCClockSource (uint32_t Periph)

Function description

Get FMC Kernel clock source.

Parameters

- **Periph:** This parameter can be one of the following values:
  - LL_RCC_FMC_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_FMC_CLKSOURCE_HCLK
  - LL_RCC_FMC_CLKSOURCE_CLKP
  - LL_RCC_FMC_CLKSOURCE_IC3
  - LL_RCC_FMC_CLKSOURCE_IC4

Reference Manual to LL API cross reference:

- CCIPR3 FMCSEL LL_RCC_GetFMCClockSource

LL_RCC_GetI2CClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetI2CClockSource (uint32_t Periph)

Function description

Get I2Cx clock source.

Parameters

- **Periph:** This parameter can be one of the following values:
  - LL_RCC_I2C1_CLKSOURCE
  - LL_RCC_I2C2_CLKSOURCE
  - LL_RCC_I2C3_CLKSOURCE
  - LL_RCC_I2C4_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_I2C1_CLKSOURCE_PCLK1
  - LL_RCC_I2C1_CLKSOURCE_CLKP
  - LL_RCC_I2C1_CLKSOURCE_IC10
  - LL_RCC_I2C1_CLKSOURCE_IC15
  - LL_RCC_I2C1_CLKSOURCE_MSI
  - LL_RCC_I2C1_CLKSOURCE_HSI
  - LL_RCC_I2C2_CLKSOURCE_PCLK1
  - LL_RCC_I2C2_CLKSOURCE_CLKP
  - LL_RCC_I2C2_CLKSOURCE_IC10
  - LL_RCC_I2C2_CLKSOURCE_IC15
  - LL_RCC_I2C2_CLKSOURCE_MSI
  - LL_RCC_I2C2_CLKSOURCE_HSI
  - LL_RCC_I2C3_CLKSOURCE_PCLK1
  - LL_RCC_I2C3_CLKSOURCE_CLKP
  - LL_RCC_I2C3_CLKSOURCE_IC10
  - LL_RCC_I2C3_CLKSOURCE_IC15
  - LL_RCC_I2C3_CLKSOURCE_MSI
  - LL_RCC_I2C3_CLKSOURCE_HSI
  - LL_RCC_I2C4_CLKSOURCE_PCLK1
  - LL_RCC_I2C4_CLKSOURCE_CLKP
  - LL_RCC_I2C4_CLKSOURCE_IC10
  - LL_RCC_I2C4_CLKSOURCE_IC15
  - LL_RCC_I2C4_CLKSOURCE_MSI
  - LL_RCC_I2C4_CLKSOURCE_HSI

Reference Manual to LL API cross reference:

- CCIPR4 I2C1SEL LL_RCC_GetI2CClockSource
-
- CCIPR4 I2C2SEL LL_RCC_GetI2CClockSource
-
- CCIPR4 I2C3SEL LL_RCC_GetI2CClockSource
-
- CCIPR4 I2C4SEL LL_RCC_GetI2CClockSource

LL_RCC_GetI3CClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetI3CClockSource (uint32_t Periph)

Function description

Get I3Cx clock source.

Parameters

- **Periph:** This parameter can be one of the following values:
  - LL_RCC_I3C1_CLKSOURCE
  - LL_RCC_I3C2_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_I3C1_CLKSOURCE_PCLK1
  - LL_RCC_I3C1_CLKSOURCE_CLKP
  - LL_RCC_I3C1_CLKSOURCE_IC10
  - LL_RCC_I3C1_CLKSOURCE_IC15
  - LL_RCC_I3C1_CLKSOURCE_MSI
  - LL_RCC_I3C1_CLKSOURCE_HSI
  - LL_RCC_I3C2_CLKSOURCE_PCLK1
  - LL_RCC_I3C2_CLKSOURCE_CLKP
  - LL_RCC_I3C2_CLKSOURCE_IC10
  - LL_RCC_I3C2_CLKSOURCE_IC15
  - LL_RCC_I3C2_CLKSOURCE_MSI
  - LL_RCC_I3C2_CLKSOURCE_HSI

Reference Manual to LL API cross reference:

- CCIPR4 I3C1SEL LL_RCC_GetI3CClockSource
-
- CCIPR4 I3C2SEL LL_RCC_GetI3CClockSource

LL_RCC_GetLPTIMClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetLPTIMClockSource (uint32_t Periph)

Function description

Get LPTIM clock source.

Parameters

- **Periph:** This parameter can be one of the following values:
  - LL_RCC_LPTIM1_CLKSOURCE
  - LL_RCC_LPTIM2_CLKSOURCE
  - LL_RCC_LPTIM3_CLKSOURCE
  - LL_RCC_LPTIM4_CLKSOURCE
  - LL_RCC_LPTIM5_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_LPTIM1_CLKSOURCE_PCLK1
  - LL_RCC_LPTIM1_CLKSOURCE_CLKP
  - LL_RCC_LPTIM1_CLKSOURCE_IC15
  - LL_RCC_LPTIM1_CLKSOURCE_LSE
  - LL_RCC_LPTIM1_CLKSOURCE_LSI
  - LL_RCC_LPTIM1_CLKSOURCE_TIMG
  - LL_RCC_LPTIM2_CLKSOURCE_PCLK4
  - LL_RCC_LPTIM2_CLKSOURCE_CLKP
  - LL_RCC_LPTIM2_CLKSOURCE_IC15
  - LL_RCC_LPTIM2_CLKSOURCE_LSE
  - LL_RCC_LPTIM2_CLKSOURCE_LSI
  - LL_RCC_LPTIM2_CLKSOURCE_TIMG
  - LL_RCC_LPTIM3_CLKSOURCE_PCLK4
  - LL_RCC_LPTIM3_CLKSOURCE_CLKP
  - LL_RCC_LPTIM3_CLKSOURCE_IC15
  - LL_RCC_LPTIM3_CLKSOURCE_LSE
  - LL_RCC_LPTIM3_CLKSOURCE_LSI
  - LL_RCC_LPTIM3_CLKSOURCE_TIMG
  - LL_RCC_LPTIM4_CLKSOURCE_PCLK4
  - LL_RCC_LPTIM4_CLKSOURCE_CLKP
  - LL_RCC_LPTIM4_CLKSOURCE_IC15
  - LL_RCC_LPTIM4_CLKSOURCE_LSE
  - LL_RCC_LPTIM4_CLKSOURCE_LSI
  - LL_RCC_LPTIM4_CLKSOURCE_TIMG
  - LL_RCC_LPTIM5_CLKSOURCE_PCLK4
  - LL_RCC_LPTIM5_CLKSOURCE_CLKP
  - LL_RCC_LPTIM5_CLKSOURCE_IC15
  - LL_RCC_LPTIM5_CLKSOURCE_LSE
  - LL_RCC_LPTIM5_CLKSOURCE_LSI
  - LL_RCC_LPTIM5_CLKSOURCE_TIMG
- **None:**

Reference Manual to LL API cross reference:

- CCIPR12 LPTIM1SEL LL_RCC_GetLPTIMClockSource
- CCIPR12 LPTIM2SEL LL_RCC_GetLPTIMClockSource
- CCIPR12 LPTIM3SEL LL_RCC_GetLPTIMClockSource
- CCIPR12 LPTIM4SEL LL_RCC_GetLPTIMClockSource
- CCIPR12 LPTIM5SEL LL_RCC_GetLPTIMClockSource

LL_RCC_GetLPUARTClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetLPUARTClockSource (uint32_t Periph)

Function description

Get LPUART clock source.

Parameters

- **Periph:** This parameter can be one of the following values:
  - LL_RCC_LPUART1_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_LPUART1_CLKSOURCE_PCLK4
  - LL_RCC_LPUART1_CLKSOURCE_CLKP
  - LL_RCC_LPUART1_CLKSOURCE_IC9
  - LL_RCC_LPUART1_CLKSOURCE_IC14
  - LL_RCC_LPUART1_CLKSOURCE_LSE
  - LL_RCC_LPUART1_CLKSOURCE_MSI
  - LL_RCC_LPUART1_CLKSOURCE_HSI

Reference Manual to LL API cross reference:

- CCIPR14 LPUART1SEL LL_RCC_GetLPUARTClockSource

LL_RCC_GetLTDCClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetLTDCClockSource (uint32_t Periph)

Function description

Get LTDC clock source.

Parameters

- **Periph:** This parameter can be one of the following values:
  - LL_RCC_LTDC_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_LTDC_CLKSOURCE_PCLK5
  - LL_RCC_LTDC_CLKSOURCE_CLKP
  - LL_RCC_LTDC_CLKSOURCE_IC16
  - LL_RCC_LTDC_CLKSOURCE_HSI

Reference Manual to LL API cross reference:

- CCIPR4 LTDCSEL LL_RCC_GetLTDCClockSource

LL_RCC_GetMDFClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetMDFClockSource (uint32_t Periph)

Function description

Get MDFx clock source.

Parameters

- **Periph:** This parameter can be one of the following values:
  - LL_RCC_MDF1_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_MDF1_CLKSOURCE_HCLK
  - LL_RCC_MDF1_CLKSOURCE_CLKP
  - LL_RCC_MDF1_CLKSOURCE_IC7
  - LL_RCC_MDF1_CLKSOURCE_IC8
  - LL_RCC_MDF1_CLKSOURCE_MSI
  - LL_RCC_MDF1_CLKSOURCE_HSI
  - LL_RCC_MDF1_CLKSOURCE_I2S_CKIN
  - LL_RCC_MDF1_CLKSOURCE_TIMG

Reference Manual to LL API cross reference:

- CCIPR5 MDF1SEL LL_RCC_GetMDFClockSource

LL_RCC_GetOTGPHYClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetOTGPHYClockSource (uint32_t Periph)

Function description

Get OTGPHY clock source.

Parameters

- **Periph:** This parameter can be one of the following values:
  - LL_RCC_OTGPHY1_CLKSOURCE
  - LL_RCC_OTGPHY2_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_OTGPHY1_CLKSOURCE_HSE_DIV_2
  - LL_RCC_OTGPHY1_CLKSOURCE_CLKP
  - LL_RCC_OTGPHY1_CLKSOURCE_IC15
  - LL_RCC_OTGPHY1_CLKSOURCE_HSE_DIV_2_OSC
  - LL_RCC_OTGPHY2_CLKSOURCE_HSE_DIV_2
  - LL_RCC_OTGPHY2_CLKSOURCE_CLKP
  - LL_RCC_OTGPHY2_CLKSOURCE_IC15
  - LL_RCC_OTGPHY2_CLKSOURCE_HSE_DIV_2_OSC

Reference Manual to LL API cross reference:

- CCIPR6 OTGPHY1SEL LL_RCC_GetOTGPHYClockSource
- CCIPR6 OTGPHY2SEL LL_RCC_GetOTGPHYClockSource

LL_RCC_GetOTGPHYCKREFClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetOTGPHYCKREFClockSource (uint32_t Periph)

Function description

Get OTGPHYCKREF clock source.

Parameters

- **Periph:** This parameter can be one of the following values:
  - LL_RCC_OTGPHY1CKREF_CLKSOURCE
  - LL_RCC_OTGPHY2CKREF_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_OTGPHY1CKREF_CLKSOURCE_OTGPHY1
  - LL_RCC_OTGPHY1CKREF_CLKSOURCE_HSE_DIV_2_OSC
  - LL_RCC_OTGPHY2CKREF_CLKSOURCE_OTGPHY2
  - LL_RCC_OTGPHY2CKREF_CLKSOURCE_HSE_DIV_2_OSC

Reference Manual to LL API cross reference:

- CCIPR6 OTGPHY1CKREFSEL LL_RCC_GetOTGPHYCKREFClockSource
- CCIPR6 OTGPHY2CKREFSEL LL_RCC_GetOTGPHYCKREFClockSource

LL_RCC_GetPSSIClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetPSSIClockSource (uint32_t Periph)

Function description

Get PSSI clock source.

Parameters

- **Periph:** This parameter can be one of the following values:
  - LL_RCC_PSSI_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_PSSI_CLKSOURCE_HCLK
  - LL_RCC_PSSI_CLKSOURCE_CLKP
  - LL_RCC_PSSI_CLKSOURCE_IC20
  - LL_RCC_PSSI_CLKSOURCE_HSI

Reference Manual to LL API cross reference:

- CCIPR7 PSSISEL LL_RCC_GetPSSIClockSource

LL_RCC_GetSAIClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetSAIClockSource (uint32_t Periph)

Function description

Get SAIx clock source.

Parameters

- **Periph:** This parameter can be one of the following values:
  - LL_RCC_SAI1_CLKSOURCE
  - LL_RCC_SAI2_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_SAI1_CLKSOURCE_PCLK2
  - LL_RCC_SAI1_CLKSOURCE_CLKP
  - LL_RCC_SAI1_CLKSOURCE_IC7
  - LL_RCC_SAI1_CLKSOURCE_IC8
  - LL_RCC_SAI1_CLKSOURCE_MSI
  - LL_RCC_SAI1_CLKSOURCE_HSI
  - LL_RCC_SAI1_CLKSOURCE_I2S_CKIN
  - LL_RCC_SAI1_CLKSOURCE_SPDIFRX1
  - LL_RCC_SAI2_CLKSOURCE_PCLK2
  - LL_RCC_SAI2_CLKSOURCE_CLKP
  - LL_RCC_SAI2_CLKSOURCE_IC7
  - LL_RCC_SAI2_CLKSOURCE_IC8
  - LL_RCC_SAI2_CLKSOURCE_MSI
  - LL_RCC_SAI2_CLKSOURCE_HSI
  - LL_RCC_SAI2_CLKSOURCE_I2S_CKIN
  - LL_RCC_SAI2_CLKSOURCE_SPDIFRX1

Reference Manual to LL API cross reference:

- CCIPR7 SAI1SEL LL_RCC_GetSAIClockSource
- CCIPR7 SAI2SEL LL_RCC_GetSAIClockSource

LL_RCC_GetSDMMCClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetSDMMCClockSource (uint32_t Periph)

Function description

Get SDMMC clock source.

Parameters

- **Periph:** This parameter can be one of the following values:
  - LL_RCC_SDMMC1_CLKSOURCE
  - LL_RCC_SDMMC2_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_SDMMC1_CLKSOURCE_HCLK
  - LL_RCC_SDMMC1_CLKSOURCE_CLKP
  - LL_RCC_SDMMC1_CLKSOURCE_IC4
  - LL_RCC_SDMMC1_CLKSOURCE_IC5
  - LL_RCC_SDMMC2_CLKSOURCE_HCLK
  - LL_RCC_SDMMC2_CLKSOURCE_CLKP
  - LL_RCC_SDMMC2_CLKSOURCE_IC4
  - LL_RCC_SDMMC2_CLKSOURCE_IC5

Reference Manual to LL API cross reference:

- CCIPR8 SDMMC1SEL LL_RCC_GetSDMMCClockSource
- CCIPR8 SDMMC2SEL LL_RCC_GetSDMMCClockSource

LL_RCC_GetSPDIFRXClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetSPDIFRXClockSource (uint32_t Periph)

Function description

Get SPDIFRX Kernel clock source.

Parameters

- **Periph:** This parameter can be one of the following values:
  - LL_RCC_SPDIFRX1_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_SPDIFRX1_CLKSOURCE_PCLK1
  - LL_RCC_SPDIFRX1_CLKSOURCE_CLKP
  - LL_RCC_SPDIFRX1_CLKSOURCE_IC7
  - LL_RCC_SPDIFRX1_CLKSOURCE_IC8
  - LL_RCC_SPDIFRX1_CLKSOURCE_MSI
  - LL_RCC_SPDIFRX1_CLKSOURCE_HSI
  - LL_RCC_SPDIFRX1_CLKSOURCE_I2S_CKIN

Reference Manual to LL API cross reference:

- CCIPR9 SPDIFRX1SEL LL_RCC_GetSPDIFRXClockSource

LL_RCC_GetSPIClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetSPIClockSource (uint32_t Periph)

Function description

Get SPIx Kernel clock source.

Parameters

- **Periph:** This parameter can be one of the following values:
  - LL_RCC_SPI1_CLKSOURCE
  - LL_RCC_SPI2_CLKSOURCE
  - LL_RCC_SPI3_CLKSOURCE
  - LL_RCC_SPI4_CLKSOURCE
  - LL_RCC_SPI5_CLKSOURCE
  - LL_RCC_SPI6_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_SPI1_CLKSOURCE_PCLK2
  - LL_RCC_SPI1_CLKSOURCE_CLKP
  - LL_RCC_SPI1_CLKSOURCE_IC8
  - LL_RCC_SPI1_CLKSOURCE_IC9
  - LL_RCC_SPI1_CLKSOURCE_MSI
  - LL_RCC_SPI1_CLKSOURCE_HSI
  - LL_RCC_SPI1_CLKSOURCE_I2S_CKIN
  - LL_RCC_SPI2_CLKSOURCE_PCLK1
  - LL_RCC_SPI2_CLKSOURCE_CLKP
  - LL_RCC_SPI2_CLKSOURCE_IC8
  - LL_RCC_SPI2_CLKSOURCE_IC9
  - LL_RCC_SPI2_CLKSOURCE_MSI
  - LL_RCC_SPI2_CLKSOURCE_HSI
  - LL_RCC_SPI2_CLKSOURCE_I2S_CKIN
  - LL_RCC_SPI3_CLKSOURCE_PCLK1
  - LL_RCC_SPI3_CLKSOURCE_CLKP
  - LL_RCC_SPI3_CLKSOURCE_IC8
  - LL_RCC_SPI3_CLKSOURCE_IC9
  - LL_RCC_SPI3_CLKSOURCE_MSI
  - LL_RCC_SPI3_CLKSOURCE_HSI
  - LL_RCC_SPI3_CLKSOURCE_I2S_CKIN
  - LL_RCC_SPI4_CLKSOURCE_PCLK2
  - LL_RCC_SPI4_CLKSOURCE_CLKP
  - LL_RCC_SPI4_CLKSOURCE_IC9
  - LL_RCC_SPI4_CLKSOURCE_IC14
  - LL_RCC_SPI4_CLKSOURCE_MSI
  - LL_RCC_SPI4_CLKSOURCE_HSI
  - LL_RCC_SPI4_CLKSOURCE_HSE
  - LL_RCC_SPI5_CLKSOURCE_PCLK2
  - LL_RCC_SPI5_CLKSOURCE_CLKP
  - LL_RCC_SPI5_CLKSOURCE_IC9
  - LL_RCC_SPI5_CLKSOURCE_IC14
  - LL_RCC_SPI5_CLKSOURCE_MSI
  - LL_RCC_SPI5_CLKSOURCE_HSI
  - LL_RCC_SPI5_CLKSOURCE_HSE
  - LL_RCC_SPI6_CLKSOURCE_PCLK4
  - LL_RCC_SPI6_CLKSOURCE_CLKP
  - LL_RCC_SPI6_CLKSOURCE_IC8
  - LL_RCC_SPI6_CLKSOURCE_IC9
  - LL_RCC_SPI6_CLKSOURCE_MSI
  - LL_RCC_SPI6_CLKSOURCE_HSI
  - LL_RCC_SPI6_CLKSOURCE_I2S_CKIN

Reference Manual to LL API cross reference:

- CCIPR9 SPI1SEL LL_RCC_GetSPIClockSource
- CCIPR9 SPI2SEL LL_RCC_GetSPIClockSource
- CCIPR9 SPI3SEL LL_RCC_GetSPIClockSource
- CCIPR9 SPI4SEL LL_RCC_GetSPIClockSource
- CCIPR9 SPI5SEL LL_RCC_GetSPIClockSource
- CCIPR9 SPI6SEL LL_RCC_GetSPIClockSource

LL_RCC_GetUSARTClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetUSARTClockSource (uint32_t Periph)

Function description

Get USARTx clock source.

Parameters

- **Periph:** This parameter can be one of the following values:
  - LL_RCC_USART1_CLKSOURCE
  - LL_RCC_USART2_CLKSOURCE
  - LL_RCC_USART3_CLKSOURCE
  - LL_RCC_USART6_CLKSOURCE
  - LL_RCC_USART10_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_USART1_CLKSOURCE_PCLK2
  - LL_RCC_USART1_CLKSOURCE_CLKP
  - LL_RCC_USART1_CLKSOURCE_IC9
  - LL_RCC_USART1_CLKSOURCE_IC14
  - LL_RCC_USART1_CLKSOURCE_LSE
  - LL_RCC_USART1_CLKSOURCE_MSI
  - LL_RCC_USART1_CLKSOURCE_HSI
  - LL_RCC_USART2_CLKSOURCE_PCLK1
  - LL_RCC_USART2_CLKSOURCE_CLKP
  - LL_RCC_USART2_CLKSOURCE_IC9
  - LL_RCC_USART2_CLKSOURCE_IC14
  - LL_RCC_USART2_CLKSOURCE_LSE
  - LL_RCC_USART2_CLKSOURCE_MSI
  - LL_RCC_USART2_CLKSOURCE_HSI
  - LL_RCC_USART3_CLKSOURCE_PCLK1
  - LL_RCC_USART3_CLKSOURCE_CLKP
  - LL_RCC_USART3_CLKSOURCE_IC9
  - LL_RCC_USART3_CLKSOURCE_IC14
  - LL_RCC_USART3_CLKSOURCE_LSE
  - LL_RCC_USART3_CLKSOURCE_MSI
  - LL_RCC_USART3_CLKSOURCE_HSI
  - LL_RCC_USART6_CLKSOURCE_PCLK2
  - LL_RCC_USART6_CLKSOURCE_CLKP
  - LL_RCC_USART6_CLKSOURCE_IC9
  - LL_RCC_USART6_CLKSOURCE_IC14
  - LL_RCC_USART6_CLKSOURCE_LSE
  - LL_RCC_USART6_CLKSOURCE_MSI
  - LL_RCC_USART6_CLKSOURCE_HSI
  - LL_RCC_USART10_CLKSOURCE_PCLK2
  - LL_RCC_USART10_CLKSOURCE_CLKP
  - LL_RCC_USART10_CLKSOURCE_IC9
  - LL_RCC_USART10_CLKSOURCE_IC14
  - LL_RCC_USART10_CLKSOURCE_LSE
  - LL_RCC_USART10_CLKSOURCE_MSI
  - LL_RCC_USART10_CLKSOURCE_HSI

Reference Manual to LL API cross reference:

- CCIPR13 USART1SEL LL_RCC_GetUSARTClockSource
- CCIPR13 USART2SEL LL_RCC_GetUSARTClockSource
- CCIPR13 USART3SEL LL_RCC_GetUSARTClockSource
- CCIPR13 USART6SEL LL_RCC_GetUSARTClockSource
- CCIPR14 USART10SEL LL_RCC_GetUSARTClockSource

LL_RCC_GetUARTClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetUARTClockSource (uint32_t Periph)

Function description

Get USARTx clock source.

Parameters

- **Periph:** This parameter can be one of the following values:
  - LL_RCC_UART4_CLKSOURCE
  - LL_RCC_UART5_CLKSOURCE
  - LL_RCC_UART7_CLKSOURCE
  - LL_RCC_UART8_CLKSOURCE
  - LL_RCC_UART9_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_UART4_CLKSOURCE_PCLK1
  - LL_RCC_UART4_CLKSOURCE_CLKP
  - LL_RCC_UART4_CLKSOURCE_IC9
  - LL_RCC_UART4_CLKSOURCE_IC14
  - LL_RCC_UART4_CLKSOURCE_LSE
  - LL_RCC_UART4_CLKSOURCE_MSI
  - LL_RCC_UART4_CLKSOURCE_HSI
  - LL_RCC_UART5_CLKSOURCE_PCLK1
  - LL_RCC_UART5_CLKSOURCE_CLKP
  - LL_RCC_UART5_CLKSOURCE_IC9
  - LL_RCC_UART5_CLKSOURCE_IC14
  - LL_RCC_UART5_CLKSOURCE_LSE
  - LL_RCC_UART5_CLKSOURCE_MSI
  - LL_RCC_UART5_CLKSOURCE_HSI
  - LL_RCC_UART7_CLKSOURCE_PCLK1
  - LL_RCC_UART7_CLKSOURCE_CLKP
  - LL_RCC_UART7_CLKSOURCE_IC9
  - LL_RCC_UART7_CLKSOURCE_IC14
  - LL_RCC_UART7_CLKSOURCE_LSE
  - LL_RCC_UART7_CLKSOURCE_MSI
  - LL_RCC_UART7_CLKSOURCE_HSI
  - LL_RCC_UART8_CLKSOURCE_PCLK1
  - LL_RCC_UART8_CLKSOURCE_CLKP
  - LL_RCC_UART8_CLKSOURCE_IC9
  - LL_RCC_UART8_CLKSOURCE_IC14
  - LL_RCC_UART8_CLKSOURCE_LSE
  - LL_RCC_UART8_CLKSOURCE_MSI
  - LL_RCC_UART8_CLKSOURCE_HSI
  - LL_RCC_UART9_CLKSOURCE_PCLK2
  - LL_RCC_UART9_CLKSOURCE_CLKP
  - LL_RCC_UART9_CLKSOURCE_IC9
  - LL_RCC_UART9_CLKSOURCE_IC14
  - LL_RCC_UART9_CLKSOURCE_LSE
  - LL_RCC_UART9_CLKSOURCE_MSI
  - LL_RCC_UART9_CLKSOURCE_HSI
- **None:**

Reference Manual to LL API cross reference:

- CCIPR13 UART4SEL LL_RCC_GetUARTClockSource
- CCIPR13 UART5SEL LL_RCC_GetUARTClockSource
- CCIPR13 UART7SEL LL_RCC_GetUARTClockSource
- CCIPR13 UART8SEL LL_RCC_GetUARTClockSource
- CCIPR14 UART9SEL LL_RCC_GetUARTClockSource

LL_RCC_GetUSBClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetUSBClockSource (uint32_t Periph)

Function description

Get USB clock source.

Parameters

- **Periph:** This parameter can be one of the following values:
  - LL_RCC_OTGPHY1_CLKSOURCE
  - LL_RCC_OTGPHY1CKREF_CLKSOURCE
  - LL_RCC_OTGPHY2_CLKSOURCE
  - LL_RCC_OTGPHY2CKREF_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_OTGPHY1_CLKSOURCE_HSE_DIV_2
  - LL_RCC_OTGPHY1_CLKSOURCE_CLKP
  - LL_RCC_OTGPHY1_CLKSOURCE_IC15
  - LL_RCC_OTGPHY1_CLKSOURCE_HSE_DIV_2_OSC
  - LL_RCC_OTGPHY1CKREF_CLKSOURCE_OTGPHY1
  - LL_RCC_OTGPHY1CKREF_CLKSOURCE_HSE_DIV_2_OSC
  - LL_RCC_OTGPHY2_CLKSOURCE_HSE_DIV_2
  - LL_RCC_OTGPHY2_CLKSOURCE_CLKP
  - LL_RCC_OTGPHY2_CLKSOURCE_IC15
  - LL_RCC_OTGPHY2_CLKSOURCE_HSE_DIV_2_OSC
  - LL_RCC_OTGPHY2CKREF_CLKSOURCE_OTGPHY2
  - LL_RCC_OTGPHY2CKREF_CLKSOURCE_HSE_DIV_2_OSC

Reference Manual to LL API cross reference:

- CCIPR6 OTGPHY1SEL LL_RCC_GetUSBClockSource
- CCIPR6 OTGPHY1CKREFSEL LL_RCC_GetUSBClockSource
- CCIPR6 OTGPHY2SEL LL_RCC_GetUSBClockSource
- CCIPR6 OTGPHY2CKREFSEL LL_RCC_GetUSBClockSource
-

LL_RCC_GetXSPIClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetXSPIClockSource (uint32_t Periph)

Function description

Get XSPI Kernel clock source.

Parameters

- **Periph:** This parameter can be one of the following values:
  - LL_RCC_XSPI1_CLKSOURCE
  - LL_RCC_XSPI2_CLKSOURCE
  - LL_RCC_XSPI3_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_XSPI1_CLKSOURCE_HCLK
  - LL_RCC_XSPI1_CLKSOURCE_CLKP
  - LL_RCC_XSPI1_CLKSOURCE_IC3
  - LL_RCC_XSPI1_CLKSOURCE_IC4
  - LL_RCC_XSPI2_CLKSOURCE_HCLK
  - LL_RCC_XSPI2_CLKSOURCE_CLKP
  - LL_RCC_XSPI2_CLKSOURCE_IC3
  - LL_RCC_XSPI2_CLKSOURCE_IC4
  - LL_RCC_XSPI3_CLKSOURCE_HCLK
  - LL_RCC_XSPI3_CLKSOURCE_CLKP
  - LL_RCC_XSPI3_CLKSOURCE_IC3
  - LL_RCC_XSPI3_CLKSOURCE_IC4

Reference Manual to LL API cross reference:

- CCIPR6 XSPI1SEL LL_RCC_GetXSPIClockSource
- CCIPR6 XSPI2SEL LL_RCC_GetXSPIClockSource
- CCIPR6 XSPI3SEL LL_RCC_GetXSPIClockSource

LL_RCC_SetRTCClockSource

Function name

__STATIC_INLINE void LL_RCC_SetRTCClockSource (uint32_t Source)

Function description

Set RTC Clock Source.

Parameters

- **Source:** This parameter can be one of the following values:
  - LL_RCC_RTC_CLKSOURCE_NONE
  - LL_RCC_RTC_CLKSOURCE_LSE
  - LL_RCC_RTC_CLKSOURCE_LSI
  - LL_RCC_RTC_CLKSOURCE_HSE

Return values

- **None:**

Notes

- Once the RTC clock source has been selected, it cannot be changed anymore unless the Backup domain is reset, or unless a failure is detected on LSE (LSECSSD is set).

Reference Manual to LL API cross reference:

- CCIPR7 RTCSEL LL_RCC_SetRTCClockSource

LL_RCC_GetRTCClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetRTCClockSource (void)

Function description

Get RTC Clock Source.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_RTC_CLKSOURCE_NONE
  - LL_RCC_RTC_CLKSOURCE_LSE
  - LL_RCC_RTC_CLKSOURCE_LSI
  - LL_RCC_RTC_CLKSOURCE_HSE

Reference Manual to LL API cross reference:

- CCIPR7 RTCSEL LL_RCC_GetRTCClockSource

LL_RCC_EnableRTC

Function name

__STATIC_INLINE void LL_RCC_EnableRTC (void)

Function description

Enable RTC.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB4ENSR1 RTCENS LL_RCC_EnableRTC

LL_RCC_DisableRTC

Function name

__STATIC_INLINE void LL_RCC_DisableRTC (void)

Function description

Disable RTC.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB4ENCR1 RTCENC LL_RCC_DisableRTC

LL_RCC_IsEnabledRTC

Function name

__STATIC_INLINE uint32_t LL_RCC_IsEnabledRTC (void)

Function description

Check if RTC has been enabled or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- APB4ENR1 RTCEN LL_RCC_IsEnabledRTC

LL_RCC_ForceBackupDomainReset

Function name

__STATIC_INLINE void LL_RCC_ForceBackupDomainReset (void)

Function description

Force the Backup domain reset.

Return values

- **None:**

Reference Manual to LL API cross reference:

- BDCR VSWRST LL_RCC_ForceBackupDomainReset

LL_RCC_ReleaseBackupDomainReset

Function name

__STATIC_INLINE void LL_RCC_ReleaseBackupDomainReset (void)

Function description

Release the Backup domain reset.

Return values

- **None:**

Reference Manual to LL API cross reference:

- BDCR VSWRST LL_RCC_ReleaseBackupDomainReset

LL_RCC_SetRTC_HSEPrescaler

Function name

__STATIC_INLINE void LL_RCC_SetRTC_HSEPrescaler (uint32_t Prescaler)

Function description

Set HSE Prescaler for RTC Clock.

Parameters

- **Prescaler:** This parameter can be one of the following values:
  - LL_RCC_RTC_HSE_DIV_1
  - LL_RCC_RTC_HSE_DIV_2
  - LL_RCC_RTC_HSE_DIV_3
  - LL_RCC_RTC_HSE_DIV_4
  - LL_RCC_RTC_HSE_DIV_5
  - LL_RCC_RTC_HSE_DIV_6
  - LL_RCC_RTC_HSE_DIV_7
  - LL_RCC_RTC_HSE_DIV_8
  - LL_RCC_RTC_HSE_DIV_9
  - LL_RCC_RTC_HSE_DIV_10
  - LL_RCC_RTC_HSE_DIV_11
  - LL_RCC_RTC_HSE_DIV_12
  - LL_RCC_RTC_HSE_DIV_13
  - LL_RCC_RTC_HSE_DIV_14
  - LL_RCC_RTC_HSE_DIV_15
  - LL_RCC_RTC_HSE_DIV_16
  - LL_RCC_RTC_HSE_DIV_17
  - LL_RCC_RTC_HSE_DIV_18
  - LL_RCC_RTC_HSE_DIV_19
  - LL_RCC_RTC_HSE_DIV_20
  - LL_RCC_RTC_HSE_DIV_21
  - LL_RCC_RTC_HSE_DIV_22
  - LL_RCC_RTC_HSE_DIV_23
  - LL_RCC_RTC_HSE_DIV_24
  - LL_RCC_RTC_HSE_DIV_25
  - LL_RCC_RTC_HSE_DIV_26
  - LL_RCC_RTC_HSE_DIV_27
  - LL_RCC_RTC_HSE_DIV_28
  - LL_RCC_RTC_HSE_DIV_29
  - LL_RCC_RTC_HSE_DIV_30
  - LL_RCC_RTC_HSE_DIV_31
  - LL_RCC_RTC_HSE_DIV_32
  - LL_RCC_RTC_HSE_DIV_33
  - LL_RCC_RTC_HSE_DIV_34
  - LL_RCC_RTC_HSE_DIV_35
  - LL_RCC_RTC_HSE_DIV_36
  - LL_RCC_RTC_HSE_DIV_37
  - LL_RCC_RTC_HSE_DIV_38
  - LL_RCC_RTC_HSE_DIV_39
  - LL_RCC_RTC_HSE_DIV_40
  - LL_RCC_RTC_HSE_DIV_41
  - LL_RCC_RTC_HSE_DIV_42
  - LL_RCC_RTC_HSE_DIV_43
  - LL_RCC_RTC_HSE_DIV_44
  - LL_RCC_RTC_HSE_DIV_45
  - LL_RCC_RTC_HSE_DIV_46
  - LL_RCC_RTC_HSE_DIV_47
  - LL_RCC_RTC_HSE_DIV_48

- LL_RCC_RTC_HSE_DIV_49
- LL_RCC_RTC_HSE_DIV_50
- LL_RCC_RTC_HSE_DIV_51
- LL_RCC_RTC_HSE_DIV_52
- LL_RCC_RTC_HSE_DIV_53
- LL_RCC_RTC_HSE_DIV_54
- LL_RCC_RTC_HSE_DIV_55
- LL_RCC_RTC_HSE_DIV_56
- LL_RCC_RTC_HSE_DIV_57
- LL_RCC_RTC_HSE_DIV_58
- LL_RCC_RTC_HSE_DIV_59
- LL_RCC_RTC_HSE_DIV_60
- LL_RCC_RTC_HSE_DIV_61
- LL_RCC_RTC_HSE_DIV_62
- LL_RCC_RTC_HSE_DIV_63

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR7 RTCPRE LL_RCC_SetRTC_HSEPrescaler

LL_RCC_GetRTC_HSEPrescaler

Function name

__STATIC_INLINE uint32_t LL_RCC_GetRTC_HSEPrescaler (void)

Function description

Get HSE Prescaler for RTC Clock.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_RTC_HSE_DIV_1
  - LL_RCC_RTC_HSE_DIV_2
  - LL_RCC_RTC_HSE_DIV_3
  - LL_RCC_RTC_HSE_DIV_4
  - LL_RCC_RTC_HSE_DIV_5
  - LL_RCC_RTC_HSE_DIV_6
  - LL_RCC_RTC_HSE_DIV_7
  - LL_RCC_RTC_HSE_DIV_8
  - LL_RCC_RTC_HSE_DIV_9
  - LL_RCC_RTC_HSE_DIV_10
  - LL_RCC_RTC_HSE_DIV_11
  - LL_RCC_RTC_HSE_DIV_12
  - LL_RCC_RTC_HSE_DIV_13
  - LL_RCC_RTC_HSE_DIV_14
  - LL_RCC_RTC_HSE_DIV_15
  - LL_RCC_RTC_HSE_DIV_16
  - LL_RCC_RTC_HSE_DIV_17
  - LL_RCC_RTC_HSE_DIV_18
  - LL_RCC_RTC_HSE_DIV_19
  - LL_RCC_RTC_HSE_DIV_20
  - LL_RCC_RTC_HSE_DIV_21
  - LL_RCC_RTC_HSE_DIV_22
  - LL_RCC_RTC_HSE_DIV_23
  - LL_RCC_RTC_HSE_DIV_24
  - LL_RCC_RTC_HSE_DIV_25
  - LL_RCC_RTC_HSE_DIV_26
  - LL_RCC_RTC_HSE_DIV_27
  - LL_RCC_RTC_HSE_DIV_28
  - LL_RCC_RTC_HSE_DIV_29
  - LL_RCC_RTC_HSE_DIV_30
  - LL_RCC_RTC_HSE_DIV_31
  - LL_RCC_RTC_HSE_DIV_32
  - LL_RCC_RTC_HSE_DIV_33
  - LL_RCC_RTC_HSE_DIV_34
  - LL_RCC_RTC_HSE_DIV_35
  - LL_RCC_RTC_HSE_DIV_36
  - LL_RCC_RTC_HSE_DIV_37
  - LL_RCC_RTC_HSE_DIV_38
  - LL_RCC_RTC_HSE_DIV_39
  - LL_RCC_RTC_HSE_DIV_40
  - LL_RCC_RTC_HSE_DIV_41
  - LL_RCC_RTC_HSE_DIV_42
  - LL_RCC_RTC_HSE_DIV_43
  - LL_RCC_RTC_HSE_DIV_44
  - LL_RCC_RTC_HSE_DIV_45
  - LL_RCC_RTC_HSE_DIV_46
  - LL_RCC_RTC_HSE_DIV_47
  - LL_RCC_RTC_HSE_DIV_48

- LL_RCC_RTC_HSE_DIV_49
- LL_RCC_RTC_HSE_DIV_50
- LL_RCC_RTC_HSE_DIV_51
- LL_RCC_RTC_HSE_DIV_52
- LL_RCC_RTC_HSE_DIV_53
- LL_RCC_RTC_HSE_DIV_54
- LL_RCC_RTC_HSE_DIV_55
- LL_RCC_RTC_HSE_DIV_56
- LL_RCC_RTC_HSE_DIV_57
- LL_RCC_RTC_HSE_DIV_58
- LL_RCC_RTC_HSE_DIV_59
- LL_RCC_RTC_HSE_DIV_60
- LL_RCC_RTC_HSE_DIV_61
- LL_RCC_RTC_HSE_DIV_62
- LL_RCC_RTC_HSE_DIV_63

Reference Manual to LL API cross reference:

- CCIPR7 RTCPRE LL_RCC_GetRTC_HSEPrescaler

LL_RCC_SetTIMPrescaler

Function name

__STATIC_INLINE void LL_RCC_SetTIMPrescaler (uint32_t Prescaler)

Function description

Set Timers Clock Prescalers.

Parameters

- **Prescaler:** This parameter can be one of the following values:
  - LL_RCC_TIM_PRESCALER_1
  - LL_RCC_TIM_PRESCALER_2
  - LL_RCC_TIM_PRESCALER_4
  - LL_RCC_TIM_PRESCALER_8

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR2 TIMPRE LL_RCC_SetTIMPrescaler

LL_RCC_GetTIMPrescaler

Function name

__STATIC_INLINE uint32_t LL_RCC_GetTIMPrescaler (void)

Function description

Get Timers Clock Prescalers.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_TIM_PRESCALER_1
  - LL_RCC_TIM_PRESCALER_2
  - LL_RCC_TIM_PRESCALER_4
  - LL_RCC_TIM_PRESCALER_8

Reference Manual to LL API cross reference:

- CFGR2 TIMPRE LL_RCC_GetTIMPrescaler

LL_RCC_PLL1_SetSource

Function name

__STATIC_INLINE void LL_RCC_PLL1_SetSource (uint32_t PLLSource)

Function description

Set the oscillator used as PLL clock source.

Parameters

- **PLLSource:** parameter can be one of the following values:
  - LL_RCC_PLLSOURCE_HSI
  - LL_RCC_PLLSOURCE_MSI
  - LL_RCC_PLLSOURCE_HSE
  - LL_RCC_PLLSOURCE_I2S_CKIN

Return values

- **None:**

Notes

- PLL1SEL can be written only when All PLLs are disabled.

Reference Manual to LL API cross reference:

- PLL1CFGR1 PLL1SEL LL_RCC_PLL1_SetSource

LL_RCC_PLL1_GetSource

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL1_GetSource (void)

Function description

Get the oscillator used as PLL clock source.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_PLLSOURCE_HSI
  - LL_RCC_PLLSOURCE_MSI
  - LL_RCC_PLLSOURCE_HSE
  - LL_RCC_PLLSOURCE_I2S_CKIN

Reference Manual to LL API cross reference:

- PLL1CFGR1 PLL1SEL LL_RCC_PLL1_GetSource

LL_RCC_PLL1_Enable

Function name

__STATIC_INLINE void LL_RCC_PLL1_Enable (void)

Function description

Enable PLL1.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR PLL1ONS LL_RCC_PLL1_Enable

LL_RCC_PLL1_Disable

Function name

__STATIC_INLINE void LL_RCC_PLL1_Disable (void)

Function description

Disable PLL1.

Return values

- **None:**

Notes

- Cannot be disabled if the PLL1 clock is used as the system clock

Reference Manual to LL API cross reference:

- CCR PLL1ONC LL_RCC_PLL1_Disable

LL_RCC_PLL1_IsReady

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL1_IsReady (void)

Function description

Check if PLL1 Ready.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR PLL1RDY LL_RCC_PLL1_IsReady

LL_RCC_PLL1_EnableBypass

Function name

__STATIC_INLINE void LL_RCC_PLL1_EnableBypass (void)

Function description

Enable PLL1 bypass (PLL1 output is driven by the PLL1 reference clock)

Return values

- **None:**

Notes

- This API shall be called only when PLL1 is disabled.

Reference Manual to LL API cross reference:

- PLL1CFGR1 PLL1BYP LL_RCC_PLL1_EnableBypass

LL_RCC_PLL1_DisableBypass

Function name

__STATIC_INLINE void LL_RCC_PLL1_DisableBypass (void)

Function description

Disable PLL1 bypass (PLL1 output is driven by the VCO)

Return values

- **None:**

Notes

- This API shall be called only when PLL1 is disabled.

Reference Manual to LL API cross reference:

- PLL1CFGR1 PLL1BYP LL_RCC_PLL1_DisableBypass

LL_RCC_PLL1_IsEnabledBypass

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL1_IsEnabledBypass (void)

Function description

Check if PLL1 bypass is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- PLL1CFGR1 PLL1BYP LL_RCC_PLL1_IsEnabledBypass

LL_RCC_PLL1_AssertModulationSpreadSpectrumReset

Function name

__STATIC_INLINE void LL_RCC_PLL1_AssertModulationSpreadSpectrumReset (void)

Function description

Assert PLL1 modulation spread-spectrum reset.

Return values

- **None:**

Notes

- This API shall be called only when PLL1 is disabled.

Reference Manual to LL API cross reference:

- PLL1CFGR3 PLL1MODSSRST LL_RCC_PLL1_AssertModulationSpreadSpectrumReset

LL_RCC_PLL1_ReleaseModulationSpreadSpectrumReset

Function name

__STATIC_INLINE void LL_RCC_PLL1_ReleaseModulationSpreadSpectrumReset (void)

Function description

Release PLL1 modulation spread-spectrum reset.

Return values

- **None:**

Notes

- This API shall be called only when PLL1 is disabled.

Reference Manual to LL API cross reference:

- PLL1CFGR3 PLL1MODSSRST LL_RCC_PLL1_ReleaseModulationSpreadSpectrumReset

LL_RCC_PLL1_EnableDAC

Function name

__STATIC_INLINE void LL_RCC_PLL1_EnableDAC (void)

Function description

Enable PLL1 noise canceling DAC in fractional mode.

Return values

- **None:**

Notes

- This API shall be called only when PLL1 is disabled.

Reference Manual to LL API cross reference:

- PLL1CFGR3 PLL1DACEN LL_RCC_PLL1_EnableDAC

LL_RCC_PLL1_DisableDAC

Function name

__STATIC_INLINE void LL_RCC_PLL1_DisableDAC (void)

Function description

Disable PLL1 noise canceling DAC in fractional mode.

Return values

- **None:**

Notes

- This API shall be called only when PLL1 is disabled.

Reference Manual to LL API cross reference:

- PLL1CFGR3 PLL1DACEN LL_RCC_PLL1_DisableDAC

LL_RCC_PLL1_IsEnabledDAC

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL1_IsEnabledDAC (void)

Function description

Check if PLL1 noise canceling DAC in fractional mode is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- PLL1CFGR3 PLL1DACEN LL_RCC_PLL1_IsEnabledDAC

LL_RCC_PLL1_EnableModulationSpreadSpectrum

Function name

__STATIC_INLINE void LL_RCC_PLL1_EnableModulationSpreadSpectrum (void)

Function description

Enable PLL1 modulation spread-spectrum with fractional divide inactive.

Return values

- **None:**

Notes

- This API shall be called only when PLL1 is disabled.

Reference Manual to LL API cross reference:

- PLL1CFGR3 PLL1MODSSDIS LL_RCC_PLL1_EnableModulationSpreadSpectrum

LL_RCC_PLL1_DisableModulationSpreadSpectrum

Function name

__STATIC_INLINE void LL_RCC_PLL1_DisableModulationSpreadSpectrum (void)

Function description

Disable PLL1 modulation spread-spectrum but active fractional divide.

Return values

- **None:**

Notes

- This API shall be called only when PLL1 is disabled.

Reference Manual to LL API cross reference:

- PLL1CFGR3 PLL1MODSSDIS LL_RCC_PLL1_DisableModulationSpreadSpectrum

LL_RCC_PLL1_IsEnabledModulationSpreadSpectrum

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL1_IsEnabledModulationSpreadSpectrum (void)

Function description

Check if PLL1 modulation spread-spectrum and inactive fractional divide is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- PLL1CFGR3 PLL1MODSSDIS LL_RCC_PLL1_IsEnabledModulationSpreadSpectrum

LL_RCC_PLL1_EnableFractionalModulationSpreadSpectrum

Function name

__STATIC_INLINE void LL_RCC_PLL1_EnableFractionalModulationSpreadSpectrum (void)

Function description

Enable PLL1 fractional divide and modulation spread-spectrum.

Return values

- **None:**

Notes

- This API shall be called only when PLL1 is disabled.

Reference Manual to LL API cross reference:

- PLL1CFGR3 PLL1MODDSEN LL_RCC_PLL1_EnableFractionalModulationSpreadSpectrum

LL_RCC_PLL1_DisableFractionalModulationSpreadSpectrum

Function name

__STATIC_INLINE void LL_RCC_PLL1_DisableFractionalModulationSpreadSpectrum (void)

Function description

Disable PLL1 fractional divide and modulation spread-spectrum.

Return values

- **None:**

Notes

- This API shall be called only when PLL1 is disabled.

Reference Manual to LL API cross reference:

- PLL1CFGR3 PLL1MODDSEN LL_RCC_PLL1_DisableFractionalModulationSpreadSpectrum

LL_RCC_PLL1_IsEnabledFractionalModulationSpreadSpectrum

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL1_IsEnabledFractionalModulationSpreadSpectrum (void)

Function description

Check if PLL1 fractional divide and modulation spread-spectrum is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- PLL1CFGR3 PLL1MODDSEN LL_RCC_PLL1_IsEnabledFractionalModulationSpreadSpectrum

LL_RCC_PLL1_SetN

Function name

__STATIC_INLINE void LL_RCC_PLL1_SetN (uint32_t N)

Function description

Set PLL1 DIVN Coefficient.

Parameters

- **N:** In integer mode, N parameter can be a value between 16 (0x10) and 2500 (0x9C4). In fractional mode, N parameter can be a value between 20 (0x14) and 500 (0x1F4).

Notes

- This API shall be called only when PLL1 is disabled.

Reference Manual to LL API cross reference:

- PLL1CFGR1 PLL1DIVN LL_RCC_PLL1_SetN

LL_RCC_PLL1_GetN

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL1_GetN (void)

Function description

Get PLL1 DIVN Coefficient.

Return values

- **In:** integer mode, a value between 16 (0x10) and 2500 (0x9C4). In fractional mode, a value between 20 (0x14) and 500 (0x1F4).

Reference Manual to LL API cross reference:

- PLL1CFGR1 PLL1DIVN LL_RCC_PLL1_GetN

LL_RCC_PLL1_SetM

Function name

__STATIC_INLINE void LL_RCC_PLL1_SetM (uint32_t M)

Function description

Set PLL1 DIVM Coefficient.

Parameters

- **M:** parameter can be a value between 1 and 63

Notes

- This API shall be called only when PLL1 is disabled.

Reference Manual to LL API cross reference:

- PLL1CFGR1 PLL1DIVM LL_RCC_PLL1_SetM

LL_RCC_PLL1_GetM

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL1_GetM (void)

Function description

Get PLL1 DIVM Coefficient.

Return values

- **A:** value between 1 and 63

Reference Manual to LL API cross reference:

- PLL1CFGR1 PLL1DIVM LL_RCC_PLL1_GetM

LL_RCC_PLL1_SetP1

Function name

__STATIC_INLINE void LL_RCC_PLL1_SetP1 (uint32_t P1)

Function description

Set PLL1 PDIV1 Coefficient.

Parameters

- **P1:** parameter can be a value between 1 and 7 when PLL1 is enabled, 0 when disabled

Notes

- This API shall be called only when PLL1 is disabled.

Reference Manual to LL API cross reference:

- PLL1CFGR3 PLL1PDIV1 LL_RCC_PLL1_SetP1

LL_RCC_PLL1_GetP1

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL1_GetP1 (void)

Function description

Get PLL1 PDIV2 Coefficient.

Return values

- **A:** value between 0 and 7

Reference Manual to LL API cross reference:

- PLL1CFGR3 PLL1PDIV1 LL_RCC_PLL1_GetP1

LL_RCC_PLL1_SetP2

Function name

__STATIC_INLINE void LL_RCC_PLL1_SetP2 (uint32_t P2)

Function description

Set PLL1 PDIV2 Coefficient.

Parameters

- **P2:** parameter can be a value between 1 and 7

Notes

- This API shall be called only when PLL1 is disabled.

Reference Manual to LL API cross reference:

- PLL1CFGR3 PLL1PDIV2 LL_RCC_PLL1_SetP2

LL_RCC_PLL1_GetP2

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL1_GetP2 (void)

Function description

Get PLL1 PDIV2 Coefficient.

Return values

- **A:** value between 1 and 7

Reference Manual to LL API cross reference:

- PLL1CFGR3 PLL1PDIV2 LL_RCC_PLL1_GetP2

LL_RCC_PLL1P_Enable

Function name

__STATIC_INLINE void LL_RCC_PLL1P_Enable (void)

Function description

Enable PLL1P.

Return values

- **None:**

Notes

- This API shall be called only when PLL1 is disabled.

Reference Manual to LL API cross reference:

- PLL1CFGR3 PLL1PDIVEN LL_RCC_PLL1P_Enable

LL_RCC_PLL1P_Disable

Function name

__STATIC_INLINE void LL_RCC_PLL1P_Disable (void)

Function description

Disable PLL1P.

Return values

- **None:**

Notes

- This API shall be called only when PLL1 is disabled.

Reference Manual to LL API cross reference:

- PLL1CFGR3 PLL1PDIVEN LL_RCC_PLL1P_Disable

LL_RCC_PLL1P_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL1P_IsEnabled (void)

Function description

Check if PLL1 P is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- PLL1CFGR3 PLL1PDIVEN LL_RCC_PLL1P_IsEnabled

LL_RCC_PLL1_SetFRACN

Function name

__STATIC_INLINE void LL_RCC_PLL1_SetFRACN (uint32_t FRACN)

Function description

Set PLL1 FRACN Coefficient.

Parameters

- **FRACN:** parameter can be a value between 0 and 2^24 (0xFFFFFF)

Notes

- This API shall be called only when PLL1 is disabled.

Reference Manual to LL API cross reference:

- PLL1CFGR2 PLL1DIVNFRAC LL_RCC_PLL1_SetFRACN

LL_RCC_PLL1_GetFRACN

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL1_GetFRACN (void)

Function description

Get PLL1 FRACN Coefficient.

Return values

- **A:** value between 0 and 2^24 (0xFFFFFF)

Reference Manual to LL API cross reference:

- PLL1CFGR2 PLL1DIVNFRAC LL_RCC_PLL1_GetFRACN

LL_RCC_PLL2_SetSource

Function name

__STATIC_INLINE void LL_RCC_PLL2_SetSource (uint32_t PLLSource)

Function description

Set the oscillator used as PLL clock source.

Parameters

- **PLLSource:** parameter can be one of the following values:
  - LL_RCC_PLLSOURCE_HSI
  - LL_RCC_PLLSOURCE_MSI
  - LL_RCC_PLLSOURCE_HSE
  - LL_RCC_PLLSOURCE_I2S_CKIN

Return values

- **None:**

Notes

- PLL2SEL can be written only when All PLLs are disabled.

Reference Manual to LL API cross reference:

- PLL2CFGR1 PLL2SEL LL_RCC_PLL2_SetSource

LL_RCC_PLL2_GetSource

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL2_GetSource (void)

Function description

Get the oscillator used as PLL clock source.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_PLLSOURCE_HSI
  - LL_RCC_PLLSOURCE_MSI
  - LL_RCC_PLLSOURCE_HSE
  - LL_RCC_PLLSOURCE_I2S_CKIN

Reference Manual to LL API cross reference:

- PLL2CFGR1 PLL2SEL LL_RCC_PLL2_GetSource

LL_RCC_PLL2_Enable

Function name

__STATIC_INLINE void LL_RCC_PLL2_Enable (void)

Function description

Enable PLL2.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR PLL2ONS LL_RCC_PLL2_Enable

LL_RCC_PLL2_Disable

Function name

__STATIC_INLINE void LL_RCC_PLL2_Disable (void)

Function description

Disable PLL2.

Return values

- **None:**

Notes

- Cannot be disabled if the PLL2 clock is used as the system clock

Reference Manual to LL API cross reference:

- CCR PLL2ONC LL_RCC_PLL2_Disable

LL_RCC_PLL2_IsReady

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL2_IsReady (void)

Function description

Check if PLL2 Ready.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR PLL2RDY LL_RCC_PLL2_IsReady

LL_RCC_PLL2_EnableBypass

Function name

__STATIC_INLINE void LL_RCC_PLL2_EnableBypass (void)

Function description

Enable PLL2 bypass (PLL2 output is driven by the PLL2 reference clock)

Return values

- **None:**

Notes

- This API shall be called only when PLL2 is disabled.

Reference Manual to LL API cross reference:

- PLL2CFGR1 PLL2BYP LL_RCC_PLL2_EnableBypass

LL_RCC_PLL2_DisableBypass

Function name

__STATIC_INLINE void LL_RCC_PLL2_DisableBypass (void)

Function description

Disable PLL2 bypass (PLL2 output is driven by the VCO)

Return values

- **None:**

Notes

- This API shall be called only when PLL2 is disabled.

Reference Manual to LL API cross reference:

- PLL2CFGR1 PLL2BYP LL_RCC_PLL2_DisableBypass

LL_RCC_PLL2_IsEnabledBypass

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL2_IsEnabledBypass (void)

Function description

Check if PLL2 bypass is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- PLL2CFGR1 PLL2BYP LL_RCC_PLL2_IsEnabledBypass

LL_RCC_PLL2_AssertModulationSpreadSpectrumReset

Function name

__STATIC_INLINE void LL_RCC_PLL2_AssertModulationSpreadSpectrumReset (void)

Function description

Assert PLL2 modulation spread-spectrum reset.

Return values

- **None:**

Notes

- This API shall be called only when PLL2 is disabled.

Reference Manual to LL API cross reference:

- PLL2CFGR3 PLL2MODSSRST LL_RCC_PLL2_AssertModulationSpreadSpectrumReset

LL_RCC_PLL2_ReleaseModulationSpreadSpectrumReset

Function name

__STATIC_INLINE void LL_RCC_PLL2_ReleaseModulationSpreadSpectrumReset (void)

Function description

Release PLL2 modulation spread-spectrum reset.

Return values

- **None:**

Notes

- This API shall be called only when PLL2 is disabled.

Reference Manual to LL API cross reference:

- PLL2CFGR3 PLL2MODSSRST LL_RCC_PLL2_ReleaseModulationSpreadSpectrumReset

LL_RCC_PLL2_EnableDAC

Function name

__STATIC_INLINE void LL_RCC_PLL2_EnableDAC (void)

Function description

Enable PLL2 noise canceling DAC in fractional mode.

Return values

- **None:**

Notes

- This API shall be called only when PLL2 is disabled.

Reference Manual to LL API cross reference:

- PLL2CFGR3 PLL2DACEN LL_RCC_PLL2_EnableDAC

LL_RCC_PLL2_DisableDAC

Function name

__STATIC_INLINE void LL_RCC_PLL2_DisableDAC (void)

Function description

Disable PLL2 noise canceling DAC in fractional mode.

Return values

- **None:**

Notes

- This API shall be called only when PLL2 is disabled.

Reference Manual to LL API cross reference:

- PLL2CFGR3 PLL2DACEN LL_RCC_PLL2_DisableDAC

LL_RCC_PLL2_IsEnabledDAC

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL2_IsEnabledDAC (void)

Function description

Check if PLL2 noise canceling DAC in fractional mode is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- PLL2CFGR3 PLL2DACEN LL_RCC_PLL2_IsEnabledDAC

LL_RCC_PLL2_EnableModulationSpreadSpectrum

Function name

__STATIC_INLINE void LL_RCC_PLL2_EnableModulationSpreadSpectrum (void)

Function description

Enable PLL2 modulation spread-spectrum with fractional divide inactive.

Return values

- **None:**

Notes

- This API shall be called only when PLL2 is disabled.

Reference Manual to LL API cross reference:

- PLL2CFGR3 PLL2MODSSDIS LL_RCC_PLL2_EnableModulationSpreadSpectrum

LL_RCC_PLL2_DisableModulationSpreadSpectrum

Function name

__STATIC_INLINE void LL_RCC_PLL2_DisableModulationSpreadSpectrum (void)

Function description

Disable PLL2 modulation spread-spectrum but active fractional divide.

Return values

- **None:**

Notes

- This API shall be called only when PLL2 is disabled.

Reference Manual to LL API cross reference:

- PLL2CFGR3 PLL2MODSSDIS LL_RCC_PLL2_DisableModulationSpreadSpectrum

LL_RCC_PLL2_IsEnabledModulationSpreadSpectrum

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL2_IsEnabledModulationSpreadSpectrum (void)

Function description

Check if PLL2 modulation spread-spectrum and inactive fractional divide is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- PLL2CFGR3 PLL2MODSSDIS LL_RCC_PLL2_IsEnabledModulationSpreadSpectrum

LL_RCC_PLL2_EnableFractionalModulationSpreadSpectrum

Function name

__STATIC_INLINE void LL_RCC_PLL2_EnableFractionalModulationSpreadSpectrum (void)

Function description

Enable PLL2 fractional divide and modulation spread-spectrum.

Return values

- **None:**

Notes

- This API shall be called only when PLL2 is disabled.

Reference Manual to LL API cross reference:

- PLL2CFGR3 PLL2MODDSEN LL_RCC_PLL2_EnableFractionalModulationSpreadSpectrum

LL_RCC_PLL2_DisableFractionalModulationSpreadSpectrum

Function name

__STATIC_INLINE void LL_RCC_PLL2_DisableFractionalModulationSpreadSpectrum (void)

Function description

Disable PLL2 fractional divide and modulation spread-spectrum.

Return values

- **None:**

Notes

- This API shall be called only when PLL2 is disabled.

Reference Manual to LL API cross reference:

- PLL2CFGR3 PLL2MODDSEN LL_RCC_PLL2_DisableFractionalModulationSpreadSpectrum

LL_RCC_PLL2_IsEnabledFractionalModulationSpreadSpectrum

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL2_IsEnabledFractionalModulationSpreadSpectrum (void)

Function description

Check if PLL2 fractional divide and modulation spread-spectrum is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- PLL2CFGR3 PLL2MODDSEN LL_RCC_PLL2_IsEnabledFractionalModulationSpreadSpectrum

LL_RCC_PLL2_SetN

Function name

__STATIC_INLINE void LL_RCC_PLL2_SetN (uint32_t N)

Function description

Set PLL2 DIVN Coefficient.

Parameters

- **N:** In integer mode, N parameter can be a value between 16 (0x10) and 2500 (0x9C4). In fractional mode, N parameter can be a value between 20 (0x14) and 500 (0x1F4).

Notes

- This API shall be called only when PLL2 is disabled.

Reference Manual to LL API cross reference:

- PLL2CFGR1 PLL2DIVN LL_RCC_PLL2_SetN

LL_RCC_PLL2_GetN

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL2_GetN (void)

Function description

Get PLL2 DIVN Coefficient.

Return values

- **In:** integer mode, a value between 16 (0x10) and 2500 (0x9C4). In fractional mode, a value between 20 (0x14) and 500 (0x1F4).

Reference Manual to LL API cross reference:

- PLL2CFGR1 PLL2DIVN LL_RCC_PLL2_GetN

LL_RCC_PLL2_SetM

Function name

__STATIC_INLINE void LL_RCC_PLL2_SetM (uint32_t M)

Function description

Set PLL2 DIVM Coefficient.

Parameters

- **M:** parameter can be a value between 1 and 63

Notes

- This API shall be called only when PLL2 is disabled.

Reference Manual to LL API cross reference:

- PLL2CFGR1 PLL2DIVM LL_RCC_PLL2_SetM

LL_RCC_PLL2_GetM

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL2_GetM (void)

Function description

Get PLL2 DIVM Coefficient.

Return values

- **A:** value between 1 and 63

Reference Manual to LL API cross reference:

- PLL2CFGR1 PLL2DIVM LL_RCC_PLL2_GetM

LL_RCC_PLL2_SetP1

Function name

__STATIC_INLINE void LL_RCC_PLL2_SetP1 (uint32_t P1)

Function description

Set PLL2 PDIV1 Coefficient.

Parameters

- **P1:** parameter can be a value between 1 and 7 when PLL2 is enabled, 0 when disabled

Notes

- This API shall be called only when PLL2 is disabled.

Reference Manual to LL API cross reference:

- PLL2CFGR3 PLL2PDIV1 LL_RCC_PLL2_SetP1

LL_RCC_PLL2_GetP1

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL2_GetP1 (void)

Function description

Get PLL2 PDIV2 Coefficient.

Return values

- **A:** value between 0 and 7

Reference Manual to LL API cross reference:

- PLL2CFGR3 PLL2PDIV1 LL_RCC_PLL2_GetP1

LL_RCC_PLL2_SetP2

Function name

__STATIC_INLINE void LL_RCC_PLL2_SetP2 (uint32_t P2)

Function description

Set PLL2 PDIV2 Coefficient.

Parameters

- **P2:** parameter can be a value between 1 and 7

Notes

- This API shall be called only when PLL2 is disabled.

Reference Manual to LL API cross reference:

- PLL2CFGR3 PLL2PDIV2 LL_RCC_PLL2_SetP2

LL_RCC_PLL2_GetP2

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL2_GetP2 (void)

Function description

Get PLL2 PDIV2 Coefficient.

Return values

- **A:** value between 1 and 7

Reference Manual to LL API cross reference:

- PLL2CFGR3 PLL2PDIV2 LL_RCC_PLL2_GetP2

LL_RCC_PLL2P_Enable

Function name

__STATIC_INLINE void LL_RCC_PLL2P_Enable (void)

Function description

Enable PLL2P.

Return values

- **None:**

Notes

- This API shall be called only when PLL2 is disabled.

Reference Manual to LL API cross reference:

- PLL2CFGR3 PLL2PDIVEN LL_RCC_PLL2P_Enable

LL_RCC_PLL2P_Disable

Function name

__STATIC_INLINE void LL_RCC_PLL2P_Disable (void)

Function description

Disable PLL2P.

Return values

- **None:**

Notes

- This API shall be called only when PLL2 is disabled.

Reference Manual to LL API cross reference:

- PLL2CFGR3 PLL2PDIVEN LL_RCC_PLL2P_Disable

LL_RCC_PLL2P_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL2P_IsEnabled (void)

Function description

Check if PLL2 P is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- PLL2CFGR3 PLL2PDIVEN LL_RCC_PLL2P_IsEnabled

LL_RCC_PLL2_SetFRACN

Function name

__STATIC_INLINE void LL_RCC_PLL2_SetFRACN (uint32_t FRACN)

Function description

Set PLL2 FRACN Coefficient.

Parameters

- **FRACN:** parameter can be a value between 0 and 2^24 (0xFFFFFF)

Notes

- This API shall be called only when PLL2 is disabled.

Reference Manual to LL API cross reference:

- PLL2CFGR2 PLL2DIVNFRAC LL_RCC_PLL2_SetFRACN

LL_RCC_PLL2_GetFRACN

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL2_GetFRACN (void)

Function description

Get PLL2 FRACN Coefficient.

Return values

- **A:** value between 0 and 2^24 (0xFFFFFF)

Reference Manual to LL API cross reference:

- PLL2CFGR2 PLL2DIVNFRAC LL_RCC_PLL2_GetFRACN

LL_RCC_PLL3_SetSource

Function name

__STATIC_INLINE void LL_RCC_PLL3_SetSource (uint32_t PLLSource)

Function description

Set the oscillator used as PLL clock source.

Parameters

- **PLLSource:** parameter can be one of the following values:
  - LL_RCC_PLLSOURCE_HSI
  - LL_RCC_PLLSOURCE_MSI
  - LL_RCC_PLLSOURCE_HSE
  - LL_RCC_PLLSOURCE_I2S_CKIN

Return values

- **None:**

Notes

- PLL3SEL can be written only when All PLLs are disabled.

Reference Manual to LL API cross reference:

- PLL3CFGR1 PLL3SEL LL_RCC_PLL3_SetSource

LL_RCC_PLL3_GetSource

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL3_GetSource (void)

Function description

Get the oscillator used as PLL clock source.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_PLLSOURCE_HSI
  - LL_RCC_PLLSOURCE_MSI
  - LL_RCC_PLLSOURCE_HSE
  - LL_RCC_PLLSOURCE_I2S_CKIN

Reference Manual to LL API cross reference:

- PLL3CFGR1 PLL3SEL LL_RCC_PLL3_GetSource

LL_RCC_PLL3_Enable

Function name

__STATIC_INLINE void LL_RCC_PLL3_Enable (void)

Function description

Enable PLL3.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR PLL3ONS LL_RCC_PLL3_Enable

LL_RCC_PLL3_Disable

Function name

__STATIC_INLINE void LL_RCC_PLL3_Disable (void)

Function description

Disable PLL3.

Return values

- **None:**

Notes

- Cannot be disabled if the PLL3 clock is used as the system clock

Reference Manual to LL API cross reference:

- CCR PLL3ONC LL_RCC_PLL3_Disable

LL_RCC_PLL3_IsReady

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL3_IsReady (void)

Function description

Check if PLL3 Ready.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR PLL3RDY LL_RCC_PLL3_IsReady

LL_RCC_PLL3_EnableBypass

Function name

__STATIC_INLINE void LL_RCC_PLL3_EnableBypass (void)

Function description

Enable PLL3 bypass (PLL3 output is driven by the PLL3 reference clock)

Return values

- **None:**

Notes

- This API shall be called only when PLL3 is disabled.

Reference Manual to LL API cross reference:

- PLL3CFGR1 PLL3BYP LL_RCC_PLL3_EnableBypass

LL_RCC_PLL3_DisableBypass

Function name

__STATIC_INLINE void LL_RCC_PLL3_DisableBypass (void)

Function description

Disable PLL3 bypass (PLL3 output is driven by the VCO)

Return values

- **None:**

Notes

- This API shall be called only when PLL3 is disabled.

Reference Manual to LL API cross reference:

- PLL3CFGR1 PLL3BYP LL_RCC_PLL3_DisableBypass

LL_RCC_PLL3_IsEnabledBypass

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL3_IsEnabledBypass (void)

Function description

Check if PLL3 bypass is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- PLL3CFGR1 PLL3BYP LL_RCC_PLL3_IsEnabledBypass

LL_RCC_PLL3_AssertModulationSpreadSpectrumReset

Function name

__STATIC_INLINE void LL_RCC_PLL3_AssertModulationSpreadSpectrumReset (void)

Function description

Assert PLL3 modulation spread-spectrum reset.

Return values

- **None:**

Notes

- This API shall be called only when PLL3 is disabled.

Reference Manual to LL API cross reference:

- PLL3CFGR3 PLL3MODSSRST LL_RCC_PLL3_AssertModulationSpreadSpectrumReset

LL_RCC_PLL3_ReleaseModulationSpreadSpectrumReset

Function name

__STATIC_INLINE void LL_RCC_PLL3_ReleaseModulationSpreadSpectrumReset (void)

Function description

Release PLL3 modulation spread-spectrum reset.

Return values

- **None:**

Notes

- This API shall be called only when PLL3 is disabled.

Reference Manual to LL API cross reference:

- PLL3CFGR3 PLL3MODSSRST LL_RCC_PLL3_ReleaseModulationSpreadSpectrumReset

LL_RCC_PLL3_EnableDAC

Function name

__STATIC_INLINE void LL_RCC_PLL3_EnableDAC (void)

Function description

Enable PLL3 noise canceling DAC in fractional mode.

Return values

- **None:**

Notes

- This API shall be called only when PLL3 is disabled.

Reference Manual to LL API cross reference:

- PLL3CFGR3 PLL3DACEN LL_RCC_PLL3_EnableDAC

LL_RCC_PLL3_DisableDAC

Function name

__STATIC_INLINE void LL_RCC_PLL3_DisableDAC (void)

Function description

Disable PLL3 noise canceling DAC in fractional mode.

Return values

- **None:**

Notes

- This API shall be called only when PLL3 is disabled.

Reference Manual to LL API cross reference:

- PLL3CFGR3 PLL3DACEN LL_RCC_PLL3_DisableDAC

LL_RCC_PLL3_IsEnabledDAC

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL3_IsEnabledDAC (void)

Function description

Check if PLL3 noise canceling DAC in fractional mode is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- PLL3CFGR3 PLL3DACEN LL_RCC_PLL3_IsEnabledDAC

LL_RCC_PLL3_EnableModulationSpreadSpectrum

Function name

__STATIC_INLINE void LL_RCC_PLL3_EnableModulationSpreadSpectrum (void)

Function description

Enable PLL3 modulation spread-spectrum with fractional divide inactive.

Return values

- **None:**

Notes

- This API shall be called only when PLL3 is disabled.

Reference Manual to LL API cross reference:

- PLL3CFGR3 PLL3MODSSDIS LL_RCC_PLL3_EnableModulationSpreadSpectrum

LL_RCC_PLL3_DisableModulationSpreadSpectrum

Function name

__STATIC_INLINE void LL_RCC_PLL3_DisableModulationSpreadSpectrum (void)

Function description

Disable PLL3 modulation spread-spectrum but active fractional divide.

Return values

- **None:**

Notes

- This API shall be called only when PLL3 is disabled.

Reference Manual to LL API cross reference:

- PLL3CFGR3 PLL3MODSSDIS LL_RCC_PLL3_DisableModulationSpreadSpectrum

LL_RCC_PLL3_IsEnabledModulationSpreadSpectrum

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL3_IsEnabledModulationSpreadSpectrum (void)

Function description

Check if PLL3 modulation spread-spectrum and inactive fractional divide is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- PLL3CFGR3 PLL3MODSSDIS LL_RCC_PLL3_IsEnabledModulationSpreadSpectrum

LL_RCC_PLL3_EnableFractionalModulationSpreadSpectrum

Function name

__STATIC_INLINE void LL_RCC_PLL3_EnableFractionalModulationSpreadSpectrum (void)

Function description

Enable PLL3 fractional divide and modulation spread-spectrum.

Return values

- **None:**

Notes

- This API shall be called only when PLL3 is disabled.

Reference Manual to LL API cross reference:

- PLL3CFGR3 PLL3MODDSEN LL_RCC_PLL3_EnableFractionalModulationSpreadSpectrum

LL_RCC_PLL3_DisableFractionalModulationSpreadSpectrum

Function name

__STATIC_INLINE void LL_RCC_PLL3_DisableFractionalModulationSpreadSpectrum (void)

Function description

Disable PLL3 fractional divide and modulation spread-spectrum.

Return values

- **None:**

Notes

- This API shall be called only when PLL3 is disabled.

Reference Manual to LL API cross reference:

- PLL3CFGR3 PLL3MODDSEN LL_RCC_PLL3_DisableFractionalModulationSpreadSpectrum

LL_RCC_PLL3_IsEnabledFractionalModulationSpreadSpectrum

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL3_IsEnabledFractionalModulationSpreadSpectrum (void)

Function description

Check if PLL3 fractional divide and modulation spread-spectrum is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- PLL3CFGR3 PLL3MODDSEN LL_RCC_PLL3_IsEnabledFractionalModulationSpreadSpectrum

LL_RCC_PLL3_SetN

Function name

__STATIC_INLINE void LL_RCC_PLL3_SetN (uint32_t N)

Function description

Set PLL3 DIVN Coefficient.

Parameters

- **N:** In integer mode, N parameter can be a value between 16 (0x10) and 2500 (0x9C4). In fractional mode, N parameter can be a value between 20 (0x14) and 500 (0x1F4).

Notes

- This API shall be called only when PLL3 is disabled.

Reference Manual to LL API cross reference:

- PLL3CFGR1 PLL3DIVN LL_RCC_PLL3_SetN

LL_RCC_PLL3_GetN

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL3_GetN (void)

Function description

Get PLL3 DIVN Coefficient.

Return values

- **In:** integer mode, a value between 16 (0x10) and 2500 (0x9C4). In fractional mode, a value between 20 (0x14) and 500 (0x1F4).

Reference Manual to LL API cross reference:

- PLL3CFGR1 PLL3DIVN LL_RCC_PLL3_GetN

LL_RCC_PLL3_SetM

Function name

__STATIC_INLINE void LL_RCC_PLL3_SetM (uint32_t M)

Function description

Set PLL3 DIVM Coefficient.

Parameters

- **M:** parameter can be a value between 1 and 63

Notes

- This API shall be called only when PLL3 is disabled.

Reference Manual to LL API cross reference:

- PLL3CFGR1 PLL3DIVM LL_RCC_PLL3_SetM

LL_RCC_PLL3_GetM

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL3_GetM (void)

Function description

Get PLL3 DIVM Coefficient.

Return values

- **A:** value between 1 and 63

Reference Manual to LL API cross reference:

- PLL3CFGR1 PLL3DIVM LL_RCC_PLL3_GetM

LL_RCC_PLL3_SetP1

Function name

__STATIC_INLINE void LL_RCC_PLL3_SetP1 (uint32_t P1)

Function description

Set PLL3 PDIV1 Coefficient.

Parameters

- **P1:** parameter can be a value between 1 and 7 when PLL3 is enabled, 0 when disabled

Notes

- This API shall be called only when PLL3 is disabled.

Reference Manual to LL API cross reference:

- PLL3CFGR3 PLL3PDIV1 LL_RCC_PLL3_SetP1

LL_RCC_PLL3_GetP1

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL3_GetP1 (void)

Function description

Get PLL3 PDIV2 Coefficient.

Return values

- **A:** value between 1 and 7

Reference Manual to LL API cross reference:

- PLL3CFGR3 PLL3PDIV1 LL_RCC_PLL3_GetP1

LL_RCC_PLL3_SetP2

Function name

__STATIC_INLINE void LL_RCC_PLL3_SetP2 (uint32_t P2)

Function description

Set PLL3 PDIV2 Coefficient.

Parameters

- **P2:** parameter can be a value between 0 and 7

Notes

- This API shall be called only when PLL3 is disabled.

Reference Manual to LL API cross reference:

- PLL3CFGR3 PLL3PDIV2 LL_RCC_PLL3_SetP2

LL_RCC_PLL3_GetP2

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL3_GetP2 (void)

Function description

Get PLL3 PDIV2 Coefficient.

Return values

- **A:** value between 1 and 7

Reference Manual to LL API cross reference:

- PLL3CFGR3 PLL3PDIV2 LL_RCC_PLL3_GetP2

LL_RCC_PLL3P_Enable

Function name

__STATIC_INLINE void LL_RCC_PLL3P_Enable (void)

Function description

Enable PLL3P.

Return values

- **None:**

Notes

- This API shall be called only when PLL3 is disabled.

Reference Manual to LL API cross reference:

- PLL3CFGR3 PLL3PDIVEN LL_RCC_PLL3P_Enable

LL_RCC_PLL3P_Disable

Function name

__STATIC_INLINE void LL_RCC_PLL3P_Disable (void)

Function description

Disable PLL3P.

Return values

- **None:**

Notes

- This API shall be called only when PLL3 is disabled.

Reference Manual to LL API cross reference:

- PLL3CFGR3 PLL3PDIVEN LL_RCC_PLL3P_Disable

LL_RCC_PLL3P_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL3P_IsEnabled (void)

Function description

Check if PLL3 P is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- PLL3CFGR3 PLL3PDIVEN LL_RCC_PLL3P_IsEnabled

LL_RCC_PLL3_SetFRACN

Function name

__STATIC_INLINE void LL_RCC_PLL3_SetFRACN (uint32_t FRACN)

Function description

Set PLL3 FRACN Coefficient.

Parameters

- **FRACN:** parameter can be a value between 0 and 2^24 (0xFFFFFF)

Notes

- This API shall be called only when PLL3 is disabled.

Reference Manual to LL API cross reference:

- PLL3CFGR2 PLL3DIVNFRAC LL_RCC_PLL3_SetFRACN

LL_RCC_PLL3_GetFRACN

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL3_GetFRACN (void)

Function description

Get PLL3 FRACN Coefficient.

Return values

- **A:** value between 0 and 2^24 (0xFFFFFF)

Reference Manual to LL API cross reference:

- PLL3CFGR2 PLL3DIVNFRAC LL_RCC_PLL3_GetFRACN

LL_RCC_PLL4_SetSource

Function name

__STATIC_INLINE void LL_RCC_PLL4_SetSource (uint32_t PLLSource)

Function description

Set the oscillator used as PLL clock source.

Parameters

- **PLLSource:** parameter can be one of the following values:
  - LL_RCC_PLLSOURCE_HSI
  - LL_RCC_PLLSOURCE_MSI
  - LL_RCC_PLLSOURCE_HSE
  - LL_RCC_PLLSOURCE_I2S_CKIN

Return values

- **None:**

Notes

- PLL4SEL can be written only when All PLLs are disabled.

Reference Manual to LL API cross reference:

- PLL4CFGR1 PLL4SEL LL_RCC_PLL4_SetSource

LL_RCC_PLL4_GetSource

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL4_GetSource (void)

Function description

Get the oscillator used as PLL clock source.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_PLLSOURCE_HSI
  - LL_RCC_PLLSOURCE_MSI
  - LL_RCC_PLLSOURCE_HSE
  - LL_RCC_PLLSOURCE_I2S_CKIN

Reference Manual to LL API cross reference:

- PLL4CFGR1 PLL4SEL LL_RCC_PLL4_GetSource

LL_RCC_PLL4_Enable

Function name

__STATIC_INLINE void LL_RCC_PLL4_Enable (void)

Function description

Enable PLL4.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR PLL4ONS LL_RCC_PLL4_Enable

LL_RCC_PLL4_Disable

Function name

__STATIC_INLINE void LL_RCC_PLL4_Disable (void)

Function description

Disable PLL4.

Return values

- **None:**

Notes

- Cannot be disabled if the PLL4 clock is used as the system clock

Reference Manual to LL API cross reference:

- CCR PLL4ONC LL_RCC_PLL4_Disable

LL_RCC_PLL4_IsReady

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL4_IsReady (void)

Function description

Check if PLL4 Ready.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR PLL4RDY LL_RCC_PLL4_IsReady

LL_RCC_PLL4_EnableBypass

Function name

__STATIC_INLINE void LL_RCC_PLL4_EnableBypass (void)

Function description

Enable PLL4 bypass (PLL4 output is driven by the PLL4 reference clock)

Return values

- **None:**

Notes

- This API shall be called only when PLL4 is disabled.

Reference Manual to LL API cross reference:

- PLL4CFGR1 PLL4BYP LL_RCC_PLL4_EnableBypass

LL_RCC_PLL4_DisableBypass

Function name

__STATIC_INLINE void LL_RCC_PLL4_DisableBypass (void)

Function description

Disable PLL4 bypass (PLL4 output is driven by the VCO)

Return values

- **None:**

Notes

- This API shall be called only when PLL4 is disabled.

Reference Manual to LL API cross reference:

- PLL4CFGR1 PLL4BYP LL_RCC_PLL4_DisableBypass

LL_RCC_PLL4_IsEnabledBypass

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL4_IsEnabledBypass (void)

Function description

Check if PLL4 bypass is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- PLL4CFGR1 PLL4BYP LL_RCC_PLL4_IsEnabledBypass

LL_RCC_PLL4_AssertModulationSpreadSpectrumReset

Function name

__STATIC_INLINE void LL_RCC_PLL4_AssertModulationSpreadSpectrumReset (void)

Function description

Assert PLL4 modulation spread-spectrum reset.

Return values

- **None:**

Notes

- This API shall be called only when PLL4 is disabled.

Reference Manual to LL API cross reference:

- PLL4CFGR3 PLL4MODSSRST LL_RCC_PLL4_AssertModulationSpreadSpectrumReset

LL_RCC_PLL4_ReleaseModulationSpreadSpectrumReset

Function name

__STATIC_INLINE void LL_RCC_PLL4_ReleaseModulationSpreadSpectrumReset (void)

Function description

Release PLL4 modulation spread-spectrum reset.

Return values

- **None:**

Notes

- This API shall be called only when PLL4 is disabled.

Reference Manual to LL API cross reference:

- PLL4CFGR3 PLL4MODSSRST LL_RCC_PLL4_ReleaseModulationSpreadSpectrumReset

LL_RCC_PLL4_EnableDAC

Function name

__STATIC_INLINE void LL_RCC_PLL4_EnableDAC (void)

Function description

Enable PLL4 noise canceling DAC in fractional mode.

Return values

- **None:**

Notes

- This API shall be called only when PLL4 is disabled.

Reference Manual to LL API cross reference:

- PLL4CFGR3 PLL4DACEN LL_RCC_PLL4_EnableDAC

LL_RCC_PLL4_DisableDAC

Function name

__STATIC_INLINE void LL_RCC_PLL4_DisableDAC (void)

Function description

Disable PLL4 noise canceling DAC in fractional mode.

Return values

- **None:**

Notes

- This API shall be called only when PLL4 is disabled.

Reference Manual to LL API cross reference:

- PLL4CFGR3 PLL4DACEN LL_RCC_PLL4_DisableDAC

LL_RCC_PLL4_IsEnabledDAC

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL4_IsEnabledDAC (void)

Function description

Check if PLL4 noise canceling DAC in fractional mode is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- PLL4CFGR3 PLL4DACEN LL_RCC_PLL4_IsEnabledDAC

LL_RCC_PLL4_EnableModulationSpreadSpectrum

Function name

__STATIC_INLINE void LL_RCC_PLL4_EnableModulationSpreadSpectrum (void)

Function description

Enable PLL4 modulation spread-spectrum with fractional divide inactive.

Return values

- **None:**

Notes

- This API shall be called only when PLL4 is disabled.

Reference Manual to LL API cross reference:

- PLL4CFGR3 PLL4MODSSDIS LL_RCC_PLL4_EnableModulationSpreadSpectrum

LL_RCC_PLL4_DisableModulationSpreadSpectrum

Function name

__STATIC_INLINE void LL_RCC_PLL4_DisableModulationSpreadSpectrum (void)

Function description

Disable PLL4 modulation spread-spectrum but active fractional divide.

Return values

- **None:**

Notes

- This API shall be called only when PLL4 is disabled.

Reference Manual to LL API cross reference:

- PLL4CFGR3 PLL4MODSSDIS LL_RCC_PLL4_DisableModulationSpreadSpectrum

LL_RCC_PLL4_IsEnabledModulationSpreadSpectrum

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL4_IsEnabledModulationSpreadSpectrum (void)

Function description

Check if PLL4 modulation spread-spectrum and inactive fractional divide is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- PLL4CFGR3 PLL4MODSSDIS LL_RCC_PLL4_IsEnabledModulationSpreadSpectrum

LL_RCC_PLL4_EnableFractionalModulationSpreadSpectrum

Function name

__STATIC_INLINE void LL_RCC_PLL4_EnableFractionalModulationSpreadSpectrum (void)

Function description

Enable PLL4 fractional divide and modulation spread-spectrum.

Return values

- **None:**

Notes

- This API shall be called only when PLL4 is disabled.

Reference Manual to LL API cross reference:

- PLL4CFGR3 PLL4MODDSEN LL_RCC_PLL4_EnableFractionalModulationSpreadSpectrum

LL_RCC_PLL4_DisableFractionalModulationSpreadSpectrum

Function name

__STATIC_INLINE void LL_RCC_PLL4_DisableFractionalModulationSpreadSpectrum (void)

Function description

Disable PLL4 fractional divide and modulation spread-spectrum.

Return values

- **None:**

Notes

- This API shall be called only when PLL4 is disabled.

Reference Manual to LL API cross reference:

- PLL4CFGR3 PLL4MODDSEN LL_RCC_PLL4_DisableFractionalModulationSpreadSpectrum

LL_RCC_PLL4_IsEnabledFractionalModulationSpreadSpectrum

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL4_IsEnabledFractionalModulationSpreadSpectrum (void)

Function description

Check if PLL4 fractional divide and modulation spread-spectrum is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- PLL4CFGR3 PLL4MODDSEN LL_RCC_PLL4_IsEnabledFractionalModulationSpreadSpectrum

LL_RCC_PLL4_SetN

Function name

__STATIC_INLINE void LL_RCC_PLL4_SetN (uint32_t N)

Function description

Set PLL4 DIVN Coefficient.

Parameters

- **N:** In integer mode, N parameter can be a value between 16 (0x10) and 2500 (0x9C4). In fractional mode, N parameter can be a value between 20 (0x14) and 500 (0x1F4).

Notes

- This API shall be called only when PLL4 is disabled.

Reference Manual to LL API cross reference:

- PLL4CFGR1 PLL4DIVN LL_RCC_PLL4_SetN

LL_RCC_PLL4_GetN

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL4_GetN (void)

Function description

Get PLL4 DIVN Coefficient.

Return values

- **In:** integer mode, a value between 16 (0x10) and 2500 (0x9C4). In fractional mode, a value between 20 (0x14) and 500 (0x1F4).

Reference Manual to LL API cross reference:

- PLL4CFGR1 PLL4DIVN LL_RCC_PLL4_GetN

LL_RCC_PLL4_SetM

Function name

__STATIC_INLINE void LL_RCC_PLL4_SetM (uint32_t M)

Function description

Set PLL4 DIVM Coefficient.

Parameters

- **M:** parameter can be a value between 1 and 63

Notes

- This API shall be called only when PLL4 is disabled.

Reference Manual to LL API cross reference:

- PLL4CFGR1 PLL4DIVM LL_RCC_PLL4_SetM

LL_RCC_PLL4_GetM

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL4_GetM (void)

Function description

Get PLL4 DIVM Coefficient.

Return values

- **A:** value between 1 and 63

Reference Manual to LL API cross reference:

- PLL4CFGR1 PLL4DIVM LL_RCC_PLL4_GetM

LL_RCC_PLL4_SetP1

Function name

__STATIC_INLINE void LL_RCC_PLL4_SetP1 (uint32_t P1)

Function description

Set PLL4 PDIV1 Coefficient.

Parameters

- **P1:** parameter can be a value between 1 and 7 when PLL4 is enabled, 0 when disabled

Notes

- This API shall be called only when PLL4 is disabled.

Reference Manual to LL API cross reference:

- PLL4CFGR3 PLL4PDIV1 LL_RCC_PLL4_SetP1

LL_RCC_PLL4_GetP1

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL4_GetP1 (void)

Function description

Get PLL4 PDIV2 Coefficient.

Return values

- **A:** value between 0 and 7

Reference Manual to LL API cross reference:

- PLL4CFGR3 PLL4PDIV1 LL_RCC_PLL4_GetP1

LL_RCC_PLL4_SetP2

Function name

__STATIC_INLINE void LL_RCC_PLL4_SetP2 (uint32_t P2)

Function description

Set PLL4 PDIV2 Coefficient.

Parameters

- **P2:** parameter can be a value between 1 and 7

Notes

- This API shall be called only when PLL4 is disabled.

Reference Manual to LL API cross reference:

- PLL4CFGR3 PLL4PDIV2 LL_RCC_PLL4_SetP2

LL_RCC_PLL4_GetP2

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL4_GetP2 (void)

Function description

Get PLL4 PDIV2 Coefficient.

Return values

- **A:** value between 1 and 7

Reference Manual to LL API cross reference:

- PLL4CFGR3 PLL4PDIV2 LL_RCC_PLL4_GetP2

LL_RCC_PLL4P_Enable

Function name

__STATIC_INLINE void LL_RCC_PLL4P_Enable (void)

Function description

Enable PLL4P.

Return values

- **None:**

Notes

- This API shall be called only when PLL4 is disabled.

Reference Manual to LL API cross reference:

- PLL4CFGR3 PLL4PDIVEN LL_RCC_PLL4P_Enable

LL_RCC_PLL4P_Disable

Function name

__STATIC_INLINE void LL_RCC_PLL4P_Disable (void)

Function description

Disable PLL4P.

Return values

- **None:**

Notes

- This API shall be called only when PLL4 is disabled.

Reference Manual to LL API cross reference:

- PLL4CFGR3 PLL4PDIVEN LL_RCC_PLL4P_Disable

LL_RCC_PLL4P_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL4P_IsEnabled (void)

Function description

Check if PLL4 P is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- PLL4CFGR3 PLL4PDIVEN LL_RCC_PLL4P_IsEnabled

LL_RCC_PLL4_SetFRACN

Function name

__STATIC_INLINE void LL_RCC_PLL4_SetFRACN (uint32_t FRACN)

Function description

Set PLL4 FRACN Coefficient.

Parameters

- **FRACN:** parameter can be a value between 0 and 2^24 (0xFFFFFF)

Notes

- This API shall be called only when PLL4 is disabled.

Reference Manual to LL API cross reference:

- PLL4CFGR2 PLL4DIVNFRAC LL_RCC_PLL4_SetFRACN

LL_RCC_PLL4_GetFRACN

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL4_GetFRACN (void)

Function description

Get PLL4 FRACN Coefficient.

Return values

- **A:** value between 0 and 2^24 (0xFFFFFF)

Reference Manual to LL API cross reference:

- PLL4CFGR2 PLL4DIVNFRAC LL_RCC_PLL4_GetFRACN

LL_RCC_IC1_Enable

Function name

__STATIC_INLINE void LL_RCC_IC1_Enable (void)

Function description

Enable IC1.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIVENSR IC1ENS LL_RCC_IC1_Enable

LL_RCC_IC1_Disable

Function name

__STATIC_INLINE void LL_RCC_IC1_Disable (void)

Function description

Disable IC1.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIVENCR IC1ENC LL_RCC_IC1_Disable

LL_RCC_IC1_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_RCC_IC1_IsEnabled (void)

Function description

Check if IC1 is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIVENR IC1EN LL_RCC_IC1_IsEnabled

LL_RCC_IC1_SetSource

Function name

__STATIC_INLINE void LL_RCC_IC1_SetSource (uint32_t Source)

Function description

Set the PLL source used as IC1 clock source.

Parameters

- **Source:** parameter can be one of the following values:
  - LL_RCC_ICCLKSOURCE_PLL1
  - LL_RCC_ICCLKSOURCE_PLL2
  - LL_RCC_ICCLKSOURCE_PLL3
  - LL_RCC_ICCLKSOURCE_PLL4

Return values

- **None:**

Reference Manual to LL API cross reference:

- IC1CFGR IC1SEL LL_RCC_IC1_SetSource

LL_RCC_IC1_GetSource

Function name

__STATIC_INLINE uint32_t LL_RCC_IC1_GetSource (void)

Function description

Get the PLL source used as IC1 clock source.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_ICCLKSOURCE_PLL1
  - LL_RCC_ICCLKSOURCE_PLL2
  - LL_RCC_ICCLKSOURCE_PLL3
  - LL_RCC_ICCLKSOURCE_PLL4

Reference Manual to LL API cross reference:

- IC1CFGR IC1SEL LL_RCC_IC1_GetSource

LL_RCC_IC1_SetDivider

Function name

__STATIC_INLINE void LL_RCC_IC1_SetDivider (uint32_t Divider)

Function description

Set divider.

Parameters

- **Divider:** This parameter can be a value between 1 and 256.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IC1CFGR IC1INT LL_RCC_IC1_SetDivider

LL_RCC_IC1_GetDivider

Function name

__STATIC_INLINE uint32_t LL_RCC_IC1_GetDivider (void)

Function description

Get IC1 divider.

Return values

- **can:** be a value between 1 and 256.

Reference Manual to LL API cross reference:

- IC1CFGR IC1INT LL_RCC_IC1_GetDivider

LL_RCC_IC2_Enable

Function name

__STATIC_INLINE void LL_RCC_IC2_Enable (void)

Function description

Enable IC2.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIVENSR IC2ENS LL_RCC_IC2_Enable

LL_RCC_IC2_Disable

Function name

__STATIC_INLINE void LL_RCC_IC2_Disable (void)

Function description

Disable IC2.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIVENCR IC2ENC LL_RCC_IC2_Disable

LL_RCC_IC2_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_RCC_IC2_IsEnabled (void)

Function description

Check if IC2 is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIVENR IC2EN LL_RCC_IC2_IsEnabled

LL_RCC_IC2_SetSource

Function name

__STATIC_INLINE void LL_RCC_IC2_SetSource (uint32_t Source)

Function description

Set the PLL source used as IC2 clock source.

Parameters

- **Source:** parameter can be one of the following values:
  - LL_RCC_ICCLKSOURCE_PLL1
  - LL_RCC_ICCLKSOURCE_PLL2
  - LL_RCC_ICCLKSOURCE_PLL3
  - LL_RCC_ICCLKSOURCE_PLL4

Return values

- **None:**

Reference Manual to LL API cross reference:

- IC2CFGR IC2SEL LL_RCC_IC2_SetSource

LL_RCC_IC2_GetSource

Function name

__STATIC_INLINE uint32_t LL_RCC_IC2_GetSource (void)

Function description

Get the PLL source used as IC2 clock source.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_ICCLKSOURCE_PLL1
  - LL_RCC_ICCLKSOURCE_PLL2
  - LL_RCC_ICCLKSOURCE_PLL3
  - LL_RCC_ICCLKSOURCE_PLL4

Reference Manual to LL API cross reference:

- IC2CFGR IC2SEL LL_RCC_IC2_GetSource

LL_RCC_IC2_SetDivider

Function name

__STATIC_INLINE void LL_RCC_IC2_SetDivider (uint32_t Divider)

Function description

Set divider.

Parameters

- **Divider:** This parameter can be a value between 1 and 256.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IC2CFGR IC2INT LL_RCC_IC2_SetDivider

LL_RCC_IC2_GetDivider

Function name

__STATIC_INLINE uint32_t LL_RCC_IC2_GetDivider (void)

Function description

Get IC2 divider.

Return values

- **can:** be a value between 1 and 256.

Reference Manual to LL API cross reference:

- IC2CFGR IC2INT LL_RCC_IC2_GetDivider

LL_RCC_IC3_Enable

Function name

__STATIC_INLINE void LL_RCC_IC3_Enable (void)

Function description

Enable IC3.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIVENSR IC3ENS LL_RCC_IC3_Enable

LL_RCC_IC3_Disable

Function name

__STATIC_INLINE void LL_RCC_IC3_Disable (void)

Function description

Disable IC3.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIVENCR IC3ENC LL_RCC_IC3_Disable

LL_RCC_IC3_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_RCC_IC3_IsEnabled (void)

Function description

Check if IC3 is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIVENR IC3EN LL_RCC_IC3_IsEnabled

LL_RCC_IC3_SetSource

Function name

__STATIC_INLINE void LL_RCC_IC3_SetSource (uint32_t Source)

Function description

Set the PLL source used as IC3 clock source.

Parameters

- **Source:** parameter can be one of the following values:
  - LL_RCC_ICCLKSOURCE_PLL1
  - LL_RCC_ICCLKSOURCE_PLL2
  - LL_RCC_ICCLKSOURCE_PLL3
  - LL_RCC_ICCLKSOURCE_PLL4

Return values

- **None:**

Reference Manual to LL API cross reference:

- IC3CFGR IC3SEL LL_RCC_IC3_SetSource

LL_RCC_IC3_GetSource

Function name

__STATIC_INLINE uint32_t LL_RCC_IC3_GetSource (void)

Function description

Get the PLL source used as IC3 clock source.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_ICCLKSOURCE_PLL1
  - LL_RCC_ICCLKSOURCE_PLL2
  - LL_RCC_ICCLKSOURCE_PLL3
  - LL_RCC_ICCLKSOURCE_PLL4

Reference Manual to LL API cross reference:

- IC3CFGR IC3SEL LL_RCC_IC3_GetSource

LL_RCC_IC3_SetDivider

Function name

__STATIC_INLINE void LL_RCC_IC3_SetDivider (uint32_t Divider)

Function description

Set divider.

Parameters

- **Divider:** This parameter can be a value between 1 and 256.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IC3CFGR IC3INT LL_RCC_IC3_SetDivider

LL_RCC_IC3_GetDivider

Function name

__STATIC_INLINE uint32_t LL_RCC_IC3_GetDivider (void)

Function description

Get IC3 divider.

Return values

- **can:** be a value between 1 and 256.

Reference Manual to LL API cross reference:

- IC3CFGR IC3INT LL_RCC_IC3_GetDivider

LL_RCC_IC4_Enable

Function name

__STATIC_INLINE void LL_RCC_IC4_Enable (void)

Function description

Enable IC4.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIVENSR IC4ENS LL_RCC_IC4_Enable

LL_RCC_IC4_Disable

Function name

__STATIC_INLINE void LL_RCC_IC4_Disable (void)

Function description

Disable IC4.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIVENCR IC4ENC LL_RCC_IC4_Disable

LL_RCC_IC4_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_RCC_IC4_IsEnabled (void)

Function description

Check if IC4 is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIVENR IC4EN LL_RCC_IC4_IsEnabled

LL_RCC_IC4_SetSource

Function name

__STATIC_INLINE void LL_RCC_IC4_SetSource (uint32_t Source)

Function description

Set the PLL source used as IC4 clock source.

Parameters

- **Source:** parameter can be one of the following values:
  - LL_RCC_ICCLKSOURCE_PLL1
  - LL_RCC_ICCLKSOURCE_PLL2
  - LL_RCC_ICCLKSOURCE_PLL3
  - LL_RCC_ICCLKSOURCE_PLL4

Return values

- **None:**

Reference Manual to LL API cross reference:

- IC4CFGR IC4SEL LL_RCC_IC4_SetSource

LL_RCC_IC4_GetSource

Function name

__STATIC_INLINE uint32_t LL_RCC_IC4_GetSource (void)

Function description

Get the PLL source used as IC4 clock source.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_ICCLKSOURCE_PLL1
  - LL_RCC_ICCLKSOURCE_PLL2
  - LL_RCC_ICCLKSOURCE_PLL3
  - LL_RCC_ICCLKSOURCE_PLL4

Reference Manual to LL API cross reference:

- IC4CFGR IC4SEL LL_RCC_IC4_GetSource

LL_RCC_IC4_SetDivider

Function name

__STATIC_INLINE void LL_RCC_IC4_SetDivider (uint32_t Divider)

Function description

Set divider.

Parameters

- **Divider:** This parameter can be a value between 1 and 256.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IC4CFGR IC4INT LL_RCC_IC4_SetDivider

LL_RCC_IC4_GetDivider

Function name

__STATIC_INLINE uint32_t LL_RCC_IC4_GetDivider (void)

Function description

Get IC4 divider.

Return values

- **can:** be a value between 1 and 256.

Reference Manual to LL API cross reference:

- IC4CFGR IC4INT LL_RCC_IC4_GetDivider

LL_RCC_IC5_Enable

Function name

__STATIC_INLINE void LL_RCC_IC5_Enable (void)

Function description

Enable IC5.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIVENSR IC5ENS LL_RCC_IC5_Enable

LL_RCC_IC5_Disable

Function name

__STATIC_INLINE void LL_RCC_IC5_Disable (void)

Function description

Disable IC5.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIVENCR IC5ENC LL_RCC_IC5_Disable

LL_RCC_IC5_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_RCC_IC5_IsEnabled (void)

Function description

Check if IC5 is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIVENR IC5EN LL_RCC_IC5_IsEnabled

LL_RCC_IC5_SetSource

Function name

__STATIC_INLINE void LL_RCC_IC5_SetSource (uint32_t Source)

Function description

Set the PLL source used as IC5 clock source.

Parameters

- **Source:** parameter can be one of the following values:
  - LL_RCC_ICCLKSOURCE_PLL1
  - LL_RCC_ICCLKSOURCE_PLL2
  - LL_RCC_ICCLKSOURCE_PLL3
  - LL_RCC_ICCLKSOURCE_PLL4

Return values

- **None:**

Reference Manual to LL API cross reference:

- IC5CFGR IC5SEL LL_RCC_IC5_SetSource

LL_RCC_IC5_GetSource

Function name

__STATIC_INLINE uint32_t LL_RCC_IC5_GetSource (void)

Function description

Get the PLL source used as IC5 clock source.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_ICCLKSOURCE_PLL1
  - LL_RCC_ICCLKSOURCE_PLL2
  - LL_RCC_ICCLKSOURCE_PLL3
  - LL_RCC_ICCLKSOURCE_PLL4

Reference Manual to LL API cross reference:

- IC5CFGR IC5SEL LL_RCC_IC5_GetSource

LL_RCC_IC5_SetDivider

Function name

__STATIC_INLINE void LL_RCC_IC5_SetDivider (uint32_t Divider)

Function description

Set divider.

Parameters

- **Divider:** This parameter can be a value between 1 and 256.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IC5CFGR IC5INT LL_RCC_IC5_SetDivider

LL_RCC_IC5_GetDivider

Function name

__STATIC_INLINE uint32_t LL_RCC_IC5_GetDivider (void)

Function description

Get IC5 divider.

Return values

- **can:** be a value between 1 and 256.

Reference Manual to LL API cross reference:

- IC5CFGR IC5INT LL_RCC_IC5_GetDivider

LL_RCC_IC6_Enable

Function name

__STATIC_INLINE void LL_RCC_IC6_Enable (void)

Function description

Enable IC6.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIVENSR IC6ENS LL_RCC_IC6_Enable

LL_RCC_IC6_Disable

Function name

__STATIC_INLINE void LL_RCC_IC6_Disable (void)

Function description

Disable IC6.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIVENCR IC6ENC LL_RCC_IC6_Disable

LL_RCC_IC6_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_RCC_IC6_IsEnabled (void)

Function description

Check if IC6 is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIVENR IC6EN LL_RCC_IC6_IsEnabled

LL_RCC_IC6_SetSource

Function name

__STATIC_INLINE void LL_RCC_IC6_SetSource (uint32_t Source)

Function description

Set the PLL source used as IC6 clock source.

Parameters

- **Source:** parameter can be one of the following values:
  - LL_RCC_ICCLKSOURCE_PLL1
  - LL_RCC_ICCLKSOURCE_PLL2
  - LL_RCC_ICCLKSOURCE_PLL3
  - LL_RCC_ICCLKSOURCE_PLL4

Return values

- **None:**

Reference Manual to LL API cross reference:

- IC6CFGR IC6SEL LL_RCC_IC6_SetSource

LL_RCC_IC6_GetSource

Function name

__STATIC_INLINE uint32_t LL_RCC_IC6_GetSource (void)

Function description

Get the PLL source used as IC6 clock source.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_ICCLKSOURCE_PLL1
  - LL_RCC_ICCLKSOURCE_PLL2
  - LL_RCC_ICCLKSOURCE_PLL3
  - LL_RCC_ICCLKSOURCE_PLL4

Reference Manual to LL API cross reference:

- IC6CFGR IC6SEL LL_RCC_IC6_GetSource

LL_RCC_IC6_SetDivider

Function name

__STATIC_INLINE void LL_RCC_IC6_SetDivider (uint32_t Divider)

Function description

Set divider.

Parameters

- **Divider:** This parameter can be a value between 1 and 256.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IC6CFGR IC6INT LL_RCC_IC6_SetDivider

LL_RCC_IC6_GetDivider

Function name

__STATIC_INLINE uint32_t LL_RCC_IC6_GetDivider (void)

Function description

Get IC6 divider.

Return values

- **can:** be a value between 1 and 256.

Reference Manual to LL API cross reference:

- IC6CFGR IC6INT LL_RCC_IC6_GetDivider

LL_RCC_IC7_Enable

Function name

__STATIC_INLINE void LL_RCC_IC7_Enable (void)

Function description

Enable IC7.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIVENSR IC7ENS LL_RCC_IC7_Enable

LL_RCC_IC7_Disable

Function name

__STATIC_INLINE void LL_RCC_IC7_Disable (void)

Function description

Disable IC7.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIVENCR IC7ENC LL_RCC_IC7_Disable

LL_RCC_IC7_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_RCC_IC7_IsEnabled (void)

Function description

Check if IC7 is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIVENR IC7EN LL_RCC_IC7_IsEnabled

LL_RCC_IC7_SetSource

Function name

__STATIC_INLINE void LL_RCC_IC7_SetSource (uint32_t Source)

Function description

Set the PLL source used as IC7 clock source.

Parameters

- **Source:** parameter can be one of the following values:
  - LL_RCC_ICCLKSOURCE_PLL1
  - LL_RCC_ICCLKSOURCE_PLL2
  - LL_RCC_ICCLKSOURCE_PLL3
  - LL_RCC_ICCLKSOURCE_PLL4

Return values

- **None:**

Reference Manual to LL API cross reference:

- IC7CFGR IC7SEL LL_RCC_IC7_SetSource

LL_RCC_IC7_GetSource

Function name

__STATIC_INLINE uint32_t LL_RCC_IC7_GetSource (void)

Function description

Get the PLL source used as IC7 clock source.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_ICCLKSOURCE_PLL1
  - LL_RCC_ICCLKSOURCE_PLL2
  - LL_RCC_ICCLKSOURCE_PLL3
  - LL_RCC_ICCLKSOURCE_PLL4

Reference Manual to LL API cross reference:

- IC7CFGR IC7SEL LL_RCC_IC7_GetSource

LL_RCC_IC7_SetDivider

Function name

__STATIC_INLINE void LL_RCC_IC7_SetDivider (uint32_t Divider)

Function description

Set divider.

Parameters

- **Divider:** This parameter can be a value between 1 and 256.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IC7CFGR IC7INT LL_RCC_IC7_SetDivider

LL_RCC_IC7_GetDivider

Function name

__STATIC_INLINE uint32_t LL_RCC_IC7_GetDivider (void)

Function description

Get IC7 divider.

Return values

- **can:** be a value between 1 and 256.

Reference Manual to LL API cross reference:

- IC7CFGR IC7INT LL_RCC_IC7_GetDivider

LL_RCC_IC8_Enable

Function name

__STATIC_INLINE void LL_RCC_IC8_Enable (void)

Function description

Enable IC8.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIVENSR IC8ENS LL_RCC_IC8_Enable

LL_RCC_IC8_Disable

Function name

__STATIC_INLINE void LL_RCC_IC8_Disable (void)

Function description

Disable IC8.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIVENCR IC8ENC LL_RCC_IC8_Disable

LL_RCC_IC8_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_RCC_IC8_IsEnabled (void)

Function description

Check if IC8 is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIVENR IC8EN LL_RCC_IC8_IsEnabled

LL_RCC_IC8_SetSource

Function name

__STATIC_INLINE void LL_RCC_IC8_SetSource (uint32_t Source)

Function description

Set the PLL source used as IC8 clock source.

Parameters

- **Source:** parameter can be one of the following values:
  - LL_RCC_ICCLKSOURCE_PLL1
  - LL_RCC_ICCLKSOURCE_PLL2
  - LL_RCC_ICCLKSOURCE_PLL3
  - LL_RCC_ICCLKSOURCE_PLL4

Return values

- **None:**

Reference Manual to LL API cross reference:

- IC8CFGR IC8SEL LL_RCC_IC8_SetSource

LL_RCC_IC8_GetSource

Function name

__STATIC_INLINE uint32_t LL_RCC_IC8_GetSource (void)

Function description

Get the PLL source used as IC8 clock source.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_ICCLKSOURCE_PLL1
  - LL_RCC_ICCLKSOURCE_PLL2
  - LL_RCC_ICCLKSOURCE_PLL3
  - LL_RCC_ICCLKSOURCE_PLL4

Reference Manual to LL API cross reference:

- IC8CFGR IC8SEL LL_RCC_IC8_GetSource

LL_RCC_IC8_SetDivider

Function name

__STATIC_INLINE void LL_RCC_IC8_SetDivider (uint32_t Divider)

Function description

Set divider.

Parameters

- **Divider:** This parameter can be a value between 1 and 256.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IC8CFGR IC8INT LL_RCC_IC8_SetDivider

LL_RCC_IC8_GetDivider

Function name

__STATIC_INLINE uint32_t LL_RCC_IC8_GetDivider (void)

Function description

Get IC8 divider.

Return values

- **can:** be a value between 1 and 256.

Reference Manual to LL API cross reference:

- IC8CFGR IC8INT LL_RCC_IC8_GetDivider

LL_RCC_IC9_Enable

Function name

__STATIC_INLINE void LL_RCC_IC9_Enable (void)

Function description

Enable IC9.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIVENSR IC9ENS LL_RCC_IC9_Enable

LL_RCC_IC9_Disable

Function name

__STATIC_INLINE void LL_RCC_IC9_Disable (void)

Function description

Disable IC9.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIVENCR IC9ENC LL_RCC_IC9_Disable

LL_RCC_IC9_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_RCC_IC9_IsEnabled (void)

Function description

Check if IC9 is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIVENR IC9EN LL_RCC_IC9_IsEnabled

LL_RCC_IC9_SetSource

Function name

__STATIC_INLINE void LL_RCC_IC9_SetSource (uint32_t Source)

Function description

Set the PLL source used as IC9 clock source.

Parameters

- **Source:** parameter can be one of the following values:
  - LL_RCC_ICCLKSOURCE_PLL1
  - LL_RCC_ICCLKSOURCE_PLL2
  - LL_RCC_ICCLKSOURCE_PLL3
  - LL_RCC_ICCLKSOURCE_PLL4

Return values

- **None:**

Reference Manual to LL API cross reference:

- IC9CFGR IC9SEL LL_RCC_IC9_SetSource

LL_RCC_IC9_GetSource

Function name

__STATIC_INLINE uint32_t LL_RCC_IC9_GetSource (void)

Function description

Get the PLL source used as IC9 clock source.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_ICCLKSOURCE_PLL1
  - LL_RCC_ICCLKSOURCE_PLL2
  - LL_RCC_ICCLKSOURCE_PLL3
  - LL_RCC_ICCLKSOURCE_PLL4

Reference Manual to LL API cross reference:

- IC9CFGR IC9SEL LL_RCC_IC9_GetSource

LL_RCC_IC9_SetDivider

Function name

__STATIC_INLINE void LL_RCC_IC9_SetDivider (uint32_t Divider)

Function description

Set divider.

Parameters

- **Divider:** This parameter can be a value between 1 and 256.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IC9CFGR IC9INT LL_RCC_IC9_SetDivider

LL_RCC_IC9_GetDivider

Function name

__STATIC_INLINE uint32_t LL_RCC_IC9_GetDivider (void)

Function description

Get IC9 divider.

Return values

- **can:** be a value between 1 and 256.

Reference Manual to LL API cross reference:

- IC9CFGR IC9INT LL_RCC_IC9_GetDivider

LL_RCC_IC10_Enable

Function name

__STATIC_INLINE void LL_RCC_IC10_Enable (void)

Function description

Enable IC10.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIVENSR IC10ENS LL_RCC_IC10_Enable

LL_RCC_IC10_Disable

Function name

__STATIC_INLINE void LL_RCC_IC10_Disable (void)

Function description

Disable IC10.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIVENCR IC10ENC LL_RCC_IC10_Disable

LL_RCC_IC10_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_RCC_IC10_IsEnabled (void)

Function description

Check if IC10 is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIVENR IC10EN LL_RCC_IC10_IsEnabled

LL_RCC_IC10_SetSource

Function name

__STATIC_INLINE void LL_RCC_IC10_SetSource (uint32_t Source)

Function description

Set the PLL source used as IC10 clock source.

Parameters

- **Source:** parameter can be one of the following values:
  - LL_RCC_ICCLKSOURCE_PLL1
  - LL_RCC_ICCLKSOURCE_PLL2
  - LL_RCC_ICCLKSOURCE_PLL3
  - LL_RCC_ICCLKSOURCE_PLL4

Return values

- **None:**

Reference Manual to LL API cross reference:

- IC10CFGR IC10SEL LL_RCC_IC10_SetSource

LL_RCC_IC10_GetSource

Function name

__STATIC_INLINE uint32_t LL_RCC_IC10_GetSource (void)

Function description

Get the PLL source used as IC10 clock source.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_ICCLKSOURCE_PLL1
  - LL_RCC_ICCLKSOURCE_PLL2
  - LL_RCC_ICCLKSOURCE_PLL3
  - LL_RCC_ICCLKSOURCE_PLL4

Reference Manual to LL API cross reference:

- IC10CFGR IC10SEL LL_RCC_IC10_GetSource

LL_RCC_IC10_SetDivider

Function name

__STATIC_INLINE void LL_RCC_IC10_SetDivider (uint32_t Divider)

Function description

Set divider.

Parameters

- **Divider:** This parameter can be a value between 1 and 256.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IC10CFGR IC10INT LL_RCC_IC10_SetDivider

LL_RCC_IC10_GetDivider

Function name

__STATIC_INLINE uint32_t LL_RCC_IC10_GetDivider (void)

Function description

Get IC10 divider.

Return values

- **can:** be a value between 1 and 256.

Reference Manual to LL API cross reference:

- IC10CFGR IC10INT LL_RCC_IC10_GetDivider

LL_RCC_IC11_Enable

Function name

__STATIC_INLINE void LL_RCC_IC11_Enable (void)

Function description

Enable IC11.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIVENSR IC11ENS LL_RCC_IC11_Enable

LL_RCC_IC11_Disable

Function name

__STATIC_INLINE void LL_RCC_IC11_Disable (void)

Function description

Disable IC11.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIVENCR IC11ENC LL_RCC_IC11_Disable

LL_RCC_IC11_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_RCC_IC11_IsEnabled (void)

Function description

Check if IC11 is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIVENR IC11EN LL_RCC_IC11_IsEnabled

LL_RCC_IC11_SetSource

Function name

__STATIC_INLINE void LL_RCC_IC11_SetSource (uint32_t Source)

Function description

Set the PLL source used as IC11 clock source.

Parameters

- **Source:** parameter can be one of the following values:
  - LL_RCC_ICCLKSOURCE_PLL1
  - LL_RCC_ICCLKSOURCE_PLL2
  - LL_RCC_ICCLKSOURCE_PLL3
  - LL_RCC_ICCLKSOURCE_PLL4

Return values

- **None:**

Reference Manual to LL API cross reference:

- IC11CFGR IC11SEL LL_RCC_IC11_SetSource

LL_RCC_IC11_GetSource

Function name

__STATIC_INLINE uint32_t LL_RCC_IC11_GetSource (void)

Function description

Get the PLL source used as IC11 clock source.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_ICCLKSOURCE_PLL1
  - LL_RCC_ICCLKSOURCE_PLL2
  - LL_RCC_ICCLKSOURCE_PLL3
  - LL_RCC_ICCLKSOURCE_PLL4

Reference Manual to LL API cross reference:

- IC11CFGR IC11SEL LL_RCC_IC11_GetSource

LL_RCC_IC11_SetDivider

Function name

__STATIC_INLINE void LL_RCC_IC11_SetDivider (uint32_t Divider)

Function description

Set divider.

Parameters

- **Divider:** This parameter can be a value between 1 and 256.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IC11CFGR IC11INT LL_RCC_IC11_SetDivider

LL_RCC_IC11_GetDivider

Function name

__STATIC_INLINE uint32_t LL_RCC_IC11_GetDivider (void)

Function description

Get IC11 divider.

Return values

- **can:** be a value between 1 and 256.

Reference Manual to LL API cross reference:

- IC11CFGR IC11INT LL_RCC_IC11_GetDivider

LL_RCC_IC12_Enable

Function name

__STATIC_INLINE void LL_RCC_IC12_Enable (void)

Function description

Enable IC12.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIVENSR IC12ENS LL_RCC_IC12_Enable

LL_RCC_IC12_Disable

Function name

__STATIC_INLINE void LL_RCC_IC12_Disable (void)

Function description

Disable IC12.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIVENCR IC12ENC LL_RCC_IC12_Disable

LL_RCC_IC12_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_RCC_IC12_IsEnabled (void)

Function description

Check if IC12 is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIVENR IC12EN LL_RCC_IC12_IsEnabled

LL_RCC_IC12_SetSource

Function name

__STATIC_INLINE void LL_RCC_IC12_SetSource (uint32_t Source)

Function description

Set the PLL source used as IC12 clock source.

Parameters

- **Source:** parameter can be one of the following values:
  - LL_RCC_ICCLKSOURCE_PLL1
  - LL_RCC_ICCLKSOURCE_PLL2
  - LL_RCC_ICCLKSOURCE_PLL3
  - LL_RCC_ICCLKSOURCE_PLL4

Return values

- **None:**

Reference Manual to LL API cross reference:

- IC12CFGR IC12SEL LL_RCC_IC12_SetSource

LL_RCC_IC12_GetSource

Function name

__STATIC_INLINE uint32_t LL_RCC_IC12_GetSource (void)

Function description

Get the PLL source used as IC12 clock source.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_ICCLKSOURCE_PLL1
  - LL_RCC_ICCLKSOURCE_PLL2
  - LL_RCC_ICCLKSOURCE_PLL3
  - LL_RCC_ICCLKSOURCE_PLL4

Reference Manual to LL API cross reference:

- IC12CFGR IC12SEL LL_RCC_IC12_GetSource

LL_RCC_IC12_SetDivider

Function name

__STATIC_INLINE void LL_RCC_IC12_SetDivider (uint32_t Divider)

Function description

Set divider.

Parameters

- **Divider:** This parameter can be a value between 1 and 256.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IC12CFGR IC12INT LL_RCC_IC12_SetDivider

LL_RCC_IC12_GetDivider

Function name

__STATIC_INLINE uint32_t LL_RCC_IC12_GetDivider (void)

Function description

Get IC12 divider.

Return values

- **can:** be a value between 1 and 256.

Reference Manual to LL API cross reference:

- IC12CFGR IC12INT LL_RCC_IC12_GetDivider

LL_RCC_IC13_Enable

Function name

__STATIC_INLINE void LL_RCC_IC13_Enable (void)

Function description

Enable IC13.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIVENSR IC13ENS LL_RCC_IC13_Enable

LL_RCC_IC13_Disable

Function name

__STATIC_INLINE void LL_RCC_IC13_Disable (void)

Function description

Disable IC13.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIVENCR IC13ENC LL_RCC_IC13_Disable

LL_RCC_IC13_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_RCC_IC13_IsEnabled (void)

Function description

Check if IC13 is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIVENR IC13EN LL_RCC_IC13_IsEnabled

LL_RCC_IC13_SetSource

Function name

__STATIC_INLINE void LL_RCC_IC13_SetSource (uint32_t Source)

Function description

Set the PLL source used as IC13 clock source.

Parameters

- **Source:** parameter can be one of the following values:
  - LL_RCC_ICCLKSOURCE_PLL1
  - LL_RCC_ICCLKSOURCE_PLL2
  - LL_RCC_ICCLKSOURCE_PLL3
  - LL_RCC_ICCLKSOURCE_PLL4

Return values

- **None:**

Reference Manual to LL API cross reference:

- IC13CFGR IC13SEL LL_RCC_IC13_SetSource

LL_RCC_IC13_GetSource

Function name

__STATIC_INLINE uint32_t LL_RCC_IC13_GetSource (void)

Function description

Get the PLL source used as IC13 clock source.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_ICCLKSOURCE_PLL1
  - LL_RCC_ICCLKSOURCE_PLL2
  - LL_RCC_ICCLKSOURCE_PLL3
  - LL_RCC_ICCLKSOURCE_PLL4

Reference Manual to LL API cross reference:

- IC13CFGR IC13SEL LL_RCC_IC13_GetSource

LL_RCC_IC13_SetDivider

Function name

__STATIC_INLINE void LL_RCC_IC13_SetDivider (uint32_t Divider)

Function description

Set divider.

Parameters

- **Divider:** This parameter can be a value between 1 and 256.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IC13CFGR IC13INT LL_RCC_IC13_SetDivider

LL_RCC_IC13_GetDivider

Function name

__STATIC_INLINE uint32_t LL_RCC_IC13_GetDivider (void)

Function description

Get IC13 divider.

Return values

- **can:** be a value between 1 and 256.

Reference Manual to LL API cross reference:

- IC13CFGR IC13INT LL_RCC_IC13_GetDivider

LL_RCC_IC14_Enable

Function name

__STATIC_INLINE void LL_RCC_IC14_Enable (void)

Function description

Enable IC14.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIVENSR IC14ENS LL_RCC_IC14_Enable

LL_RCC_IC14_Disable

Function name

__STATIC_INLINE void LL_RCC_IC14_Disable (void)

Function description

Disable IC14.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIVENCR IC14ENC LL_RCC_IC14_Disable

LL_RCC_IC14_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_RCC_IC14_IsEnabled (void)

Function description

Check if IC14 is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIVENR IC14EN LL_RCC_IC14_IsEnabled

LL_RCC_IC14_SetSource

Function name

__STATIC_INLINE void LL_RCC_IC14_SetSource (uint32_t Source)

Function description

Set the PLL source used as IC14 clock source.

Parameters

- **Source:** parameter can be one of the following values:
  - LL_RCC_ICCLKSOURCE_PLL1
  - LL_RCC_ICCLKSOURCE_PLL2
  - LL_RCC_ICCLKSOURCE_PLL3
  - LL_RCC_ICCLKSOURCE_PLL4

Return values

- **None:**

Reference Manual to LL API cross reference:

- IC14CFGR IC14SEL LL_RCC_IC14_SetSource

LL_RCC_IC14_GetSource

Function name

__STATIC_INLINE uint32_t LL_RCC_IC14_GetSource (void)

Function description

Get the PLL source used as IC14 clock source.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_ICCLKSOURCE_PLL1
  - LL_RCC_ICCLKSOURCE_PLL2
  - LL_RCC_ICCLKSOURCE_PLL3
  - LL_RCC_ICCLKSOURCE_PLL4

Reference Manual to LL API cross reference:

- IC14CFGR IC14SEL LL_RCC_IC14_GetSource

LL_RCC_IC14_SetDivider

Function name

__STATIC_INLINE void LL_RCC_IC14_SetDivider (uint32_t Divider)

Function description

Set divider.

Parameters

- **Divider:** This parameter can be a value between 1 and 256.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IC14CFGR IC14INT LL_RCC_IC14_SetDivider

LL_RCC_IC14_GetDivider

Function name

__STATIC_INLINE uint32_t LL_RCC_IC14_GetDivider (void)

Function description

Get IC14 divider.

Return values

- **can:** be a value between 1 and 256.

Reference Manual to LL API cross reference:

- IC14CFGR IC14INT LL_RCC_IC14_GetDivider

LL_RCC_IC15_Enable

Function name

__STATIC_INLINE void LL_RCC_IC15_Enable (void)

Function description

Enable IC15.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIVENSR IC15ENS LL_RCC_IC15_Enable

LL_RCC_IC15_Disable

Function name

__STATIC_INLINE void LL_RCC_IC15_Disable (void)

Function description

Disable IC15.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIVENCR IC15ENC LL_RCC_IC15_Disable

LL_RCC_IC15_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_RCC_IC15_IsEnabled (void)

Function description

Check if IC15 is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIVENR IC15EN LL_RCC_IC15_IsEnabled

LL_RCC_IC15_SetSource

Function name

__STATIC_INLINE void LL_RCC_IC15_SetSource (uint32_t Source)

Function description

Set the PLL source used as IC15 clock source.

Parameters

- **Source:** parameter can be one of the following values:
  - LL_RCC_ICCLKSOURCE_PLL1
  - LL_RCC_ICCLKSOURCE_PLL2
  - LL_RCC_ICCLKSOURCE_PLL3
  - LL_RCC_ICCLKSOURCE_PLL4

Return values

- **None:**

Reference Manual to LL API cross reference:

- IC15CFGR IC15SEL LL_RCC_IC15_SetSource

LL_RCC_IC15_GetSource

Function name

__STATIC_INLINE uint32_t LL_RCC_IC15_GetSource (void)

Function description

Get the PLL source used as IC15 clock source.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_ICCLKSOURCE_PLL1
  - LL_RCC_ICCLKSOURCE_PLL2
  - LL_RCC_ICCLKSOURCE_PLL3
  - LL_RCC_ICCLKSOURCE_PLL4

Reference Manual to LL API cross reference:

- IC15CFGR IC15SEL LL_RCC_IC15_GetSource

LL_RCC_IC15_SetDivider

Function name

__STATIC_INLINE void LL_RCC_IC15_SetDivider (uint32_t Divider)

Function description

Set divider.

Parameters

- **Divider:** This parameter can be a value between 1 and 256.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IC15CFGR IC15INT LL_RCC_IC15_SetDivider

LL_RCC_IC15_GetDivider

Function name

__STATIC_INLINE uint32_t LL_RCC_IC15_GetDivider (void)

Function description

Get IC15 divider.

Return values

- **can:** be a value between 1 and 256.

Reference Manual to LL API cross reference:

- IC15CFGR IC15INT LL_RCC_IC15_GetDivider

LL_RCC_IC16_Enable

Function name

__STATIC_INLINE void LL_RCC_IC16_Enable (void)

Function description

Enable IC16.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIVENSR IC16ENS LL_RCC_IC16_Enable

LL_RCC_IC16_Disable

Function name

__STATIC_INLINE void LL_RCC_IC16_Disable (void)

Function description

Disable IC16.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIVENCR IC16ENC LL_RCC_IC16_Disable

LL_RCC_IC16_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_RCC_IC16_IsEnabled (void)

Function description

Check if IC16 is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIVENR IC16EN LL_RCC_IC16_IsEnabled

LL_RCC_IC16_SetSource

Function name

__STATIC_INLINE void LL_RCC_IC16_SetSource (uint32_t Source)

Function description

Set the PLL source used as IC16 clock source.

Parameters

- **Source:** parameter can be one of the following values:
  - LL_RCC_ICCLKSOURCE_PLL1
  - LL_RCC_ICCLKSOURCE_PLL2
  - LL_RCC_ICCLKSOURCE_PLL3
  - LL_RCC_ICCLKSOURCE_PLL4

Return values

- **None:**

Reference Manual to LL API cross reference:

- IC16CFGR IC16SEL LL_RCC_IC16_SetSource

LL_RCC_IC16_GetSource

Function name

__STATIC_INLINE uint32_t LL_RCC_IC16_GetSource (void)

Function description

Get the PLL source used as IC16 clock source.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_ICCLKSOURCE_PLL1
  - LL_RCC_ICCLKSOURCE_PLL2
  - LL_RCC_ICCLKSOURCE_PLL3
  - LL_RCC_ICCLKSOURCE_PLL4

Reference Manual to LL API cross reference:

- IC16CFGR IC16SEL LL_RCC_IC16_GetSource

LL_RCC_IC16_SetDivider

Function name

__STATIC_INLINE void LL_RCC_IC16_SetDivider (uint32_t Divider)

Function description

Set divider.

Parameters

- **Divider:** This parameter can be a value between 1 and 256.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IC16CFGR IC16INT LL_RCC_IC16_SetDivider

LL_RCC_IC16_GetDivider

Function name

__STATIC_INLINE uint32_t LL_RCC_IC16_GetDivider (void)

Function description

Get IC16 divider.

Return values

- **can:** be a value between 1 and 256.

Reference Manual to LL API cross reference:

- IC16CFGR IC16INT LL_RCC_IC16_GetDivider

LL_RCC_IC17_Enable

Function name

__STATIC_INLINE void LL_RCC_IC17_Enable (void)

Function description

Enable IC17.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIVENSR IC17ENS LL_RCC_IC17_Enable

LL_RCC_IC17_Disable

Function name

__STATIC_INLINE void LL_RCC_IC17_Disable (void)

Function description

Disable IC17.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIVENCR IC17ENC LL_RCC_IC17_Disable

LL_RCC_IC17_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_RCC_IC17_IsEnabled (void)

Function description

Check if IC17 is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIVENR IC17EN LL_RCC_IC17_IsEnabled

LL_RCC_IC17_SetSource

Function name

__STATIC_INLINE void LL_RCC_IC17_SetSource (uint32_t Source)

Function description

Set the PLL source used as IC17 clock source.

Parameters

- **Source:** parameter can be one of the following values:
  - LL_RCC_ICCLKSOURCE_PLL1
  - LL_RCC_ICCLKSOURCE_PLL2
  - LL_RCC_ICCLKSOURCE_PLL3
  - LL_RCC_ICCLKSOURCE_PLL4

Return values

- **None:**

Reference Manual to LL API cross reference:

- IC17CFGR IC17SEL LL_RCC_IC17_SetSource

LL_RCC_IC17_GetSource

Function name

__STATIC_INLINE uint32_t LL_RCC_IC17_GetSource (void)

Function description

Get the PLL source used as IC17 clock source.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_ICCLKSOURCE_PLL1
  - LL_RCC_ICCLKSOURCE_PLL2
  - LL_RCC_ICCLKSOURCE_PLL3
  - LL_RCC_ICCLKSOURCE_PLL4

Reference Manual to LL API cross reference:

- IC17CFGR IC17SEL LL_RCC_IC17_GetSource

LL_RCC_IC17_SetDivider

Function name

__STATIC_INLINE void LL_RCC_IC17_SetDivider (uint32_t Divider)

Function description

Set divider.

Parameters

- **Divider:** This parameter can be a value between 1 and 256.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IC17CFGR IC17INT LL_RCC_IC17_SetDivider

LL_RCC_IC17_GetDivider

Function name

__STATIC_INLINE uint32_t LL_RCC_IC17_GetDivider (void)

Function description

Get IC17 divider.

Return values

- **can:** be a value between 1 and 256.

Reference Manual to LL API cross reference:

- IC17CFGR IC17INT LL_RCC_IC17_GetDivider

LL_RCC_IC18_Enable

Function name

__STATIC_INLINE void LL_RCC_IC18_Enable (void)

Function description

Enable IC18.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIVENSR IC18ENS LL_RCC_IC18_Enable

LL_RCC_IC18_Disable

Function name

__STATIC_INLINE void LL_RCC_IC18_Disable (void)

Function description

Disable IC18.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIVENCR IC18ENC LL_RCC_IC18_Disable

LL_RCC_IC18_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_RCC_IC18_IsEnabled (void)

Function description

Check if IC18 is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIVENR IC18EN LL_RCC_IC18_IsEnabled

LL_RCC_IC18_SetSource

Function name

__STATIC_INLINE void LL_RCC_IC18_SetSource (uint32_t Source)

Function description

Set the PLL source used as IC18 clock source.

Parameters

- **Source:** parameter can be one of the following values:
  - LL_RCC_ICCLKSOURCE_PLL1
  - LL_RCC_ICCLKSOURCE_PLL2
  - LL_RCC_ICCLKSOURCE_PLL3
  - LL_RCC_ICCLKSOURCE_PLL4

Return values

- **None:**

Reference Manual to LL API cross reference:

- IC18CFGR IC18SEL LL_RCC_IC18_SetSource

LL_RCC_IC18_GetSource

Function name

__STATIC_INLINE uint32_t LL_RCC_IC18_GetSource (void)

Function description

Get the PLL source used as IC18 clock source.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_ICCLKSOURCE_PLL1
  - LL_RCC_ICCLKSOURCE_PLL2
  - LL_RCC_ICCLKSOURCE_PLL3
  - LL_RCC_ICCLKSOURCE_PLL4

Reference Manual to LL API cross reference:

- IC18CFGR IC18SEL LL_RCC_IC18_GetSource

LL_RCC_IC18_SetDivider

Function name

__STATIC_INLINE void LL_RCC_IC18_SetDivider (uint32_t Divider)

Function description

Set divider.

Parameters

- **Divider:** This parameter can be a value between 1 and 256.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IC18CFGR IC18INT LL_RCC_IC18_SetDivider

LL_RCC_IC18_GetDivider

Function name

__STATIC_INLINE uint32_t LL_RCC_IC18_GetDivider (void)

Function description

Get IC18 divider.

Return values

- **can:** be a value between 1 and 256.

Reference Manual to LL API cross reference:

- IC18CFGR IC18INT LL_RCC_IC18_GetDivider

LL_RCC_IC19_Enable

Function name

__STATIC_INLINE void LL_RCC_IC19_Enable (void)

Function description

Enable IC19.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIVENSR IC19ENS LL_RCC_IC19_Enable

LL_RCC_IC19_Disable

Function name

__STATIC_INLINE void LL_RCC_IC19_Disable (void)

Function description

Disable IC19.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIVENCR IC19ENC LL_RCC_IC19_Disable

LL_RCC_IC19_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_RCC_IC19_IsEnabled (void)

Function description

Check if IC19 is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIVENR IC19EN LL_RCC_IC19_IsEnabled

LL_RCC_IC19_SetSource

Function name

__STATIC_INLINE void LL_RCC_IC19_SetSource (uint32_t Source)

Function description

Set the PLL source used as IC19 clock source.

Parameters

- **Source:** parameter can be one of the following values:
  - LL_RCC_ICCLKSOURCE_PLL1
  - LL_RCC_ICCLKSOURCE_PLL2
  - LL_RCC_ICCLKSOURCE_PLL3
  - LL_RCC_ICCLKSOURCE_PLL4

Return values

- **None:**

Reference Manual to LL API cross reference:

- IC19CFGR IC19SEL LL_RCC_IC19_SetSource

LL_RCC_IC19_GetSource

Function name

__STATIC_INLINE uint32_t LL_RCC_IC19_GetSource (void)

Function description

Get the PLL source used as IC19 clock source.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_ICCLKSOURCE_PLL1
  - LL_RCC_ICCLKSOURCE_PLL2
  - LL_RCC_ICCLKSOURCE_PLL3
  - LL_RCC_ICCLKSOURCE_PLL4

Reference Manual to LL API cross reference:

- IC19CFGR IC19SEL LL_RCC_IC19_GetSource

LL_RCC_IC19_SetDivider

Function name

__STATIC_INLINE void LL_RCC_IC19_SetDivider (uint32_t Divider)

Function description

Set divider.

Parameters

- **Divider:** This parameter can be a value between 1 and 256.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IC19CFGR IC19INT LL_RCC_IC19_SetDivider

LL_RCC_IC19_GetDivider

Function name

__STATIC_INLINE uint32_t LL_RCC_IC19_GetDivider (void)

Function description

Get IC19 divider.

Return values

- **can:** be a value between 1 and 256.

Reference Manual to LL API cross reference:

- IC19CFGR IC19INT LL_RCC_IC19_GetDivider

LL_RCC_IC20_Enable

Function name

__STATIC_INLINE void LL_RCC_IC20_Enable (void)

Function description

Enable IC20.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIVENSR IC20ENS LL_RCC_IC20_Enable

LL_RCC_IC20_Disable

Function name

__STATIC_INLINE void LL_RCC_IC20_Disable (void)

Function description

Disable IC20.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIVENCR IC20ENC LL_RCC_IC20_Disable

LL_RCC_IC20_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_RCC_IC20_IsEnabled (void)

Function description

Check if IC20 is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIVENR IC20EN LL_RCC_IC20_IsEnabled

LL_RCC_IC20_SetSource

Function name

__STATIC_INLINE void LL_RCC_IC20_SetSource (uint32_t Source)

Function description

Set the PLL source used as IC20 clock source.

Parameters

- **Source:** parameter can be one of the following values:
  - LL_RCC_ICCLKSOURCE_PLL1
  - LL_RCC_ICCLKSOURCE_PLL2
  - LL_RCC_ICCLKSOURCE_PLL3
  - LL_RCC_ICCLKSOURCE_PLL4

Return values

- **None:**

Reference Manual to LL API cross reference:

- IC20CFGR IC20SEL LL_RCC_IC20_SetSource

LL_RCC_IC20_GetSource

Function name

__STATIC_INLINE uint32_t LL_RCC_IC20_GetSource (void)

Function description

Get the PLL source used as IC20 clock source.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_ICCLKSOURCE_PLL1
  - LL_RCC_ICCLKSOURCE_PLL2
  - LL_RCC_ICCLKSOURCE_PLL3
  - LL_RCC_ICCLKSOURCE_PLL4

Reference Manual to LL API cross reference:

- IC20CFGR IC20SEL LL_RCC_IC20_GetSource

LL_RCC_IC20_SetDivider

Function name

__STATIC_INLINE void LL_RCC_IC20_SetDivider (uint32_t Divider)

Function description

Set divider.

Parameters

- **Divider:** This parameter can be a value between 1 and 256.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IC20CFGR IC20INT LL_RCC_IC20_SetDivider

LL_RCC_IC20_GetDivider

Function name

__STATIC_INLINE uint32_t LL_RCC_IC20_GetDivider (void)

Function description

Get IC20 divider.

Return values

- **can:** be a value between 1 and 256.

Reference Manual to LL API cross reference:

- IC20CFGR IC20INT LL_RCC_IC20_GetDivider

LL_RCC_CLKP_Enable

Function name

__STATIC_INLINE void LL_RCC_CLKP_Enable (void)

Function description

Enable CLKP.

Return values

- **None:**

Reference Manual to LL API cross reference:

- MISCENSR PERENS LL_RCC_CLKP_Enable

LL_RCC_CLKP_Disable

Function name

__STATIC_INLINE void LL_RCC_CLKP_Disable (void)

Function description

Disable CLKP.

Return values

- **None:**

Reference Manual to LL API cross reference:

- MISCENCR PERENC LL_RCC_CLKP_Disable

LL_RCC_CLKP_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_RCC_CLKP_IsEnabled (void)

Function description

Check if CLKP is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- MISCENR PEREN LL_RCC_CLKP_IsEnabled

LL_RCC_ClearFlag_LSIRDY

Function name

__STATIC_INLINE void LL_RCC_ClearFlag_LSIRDY (void)

Function description

Clear LSI ready interrupt flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CICR LSIRDYC LL_RCC_ClearFlag_LSIRDY

LL_RCC_ClearFlag_LSERDY

Function name

__STATIC_INLINE void LL_RCC_ClearFlag_LSERDY (void)

Function description

Clear LSE ready interrupt flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CICR LSERDYC LL_RCC_ClearFlag_LSERDY

LL_RCC_ClearFlag_HSIRDY

Function name

__STATIC_INLINE void LL_RCC_ClearFlag_HSIRDY (void)

Function description

Clear HSI ready interrupt flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CICR HSIRDYC LL_RCC_ClearFlag_HSIRDY

LL_RCC_ClearFlag_HSERDY

Function name

__STATIC_INLINE void LL_RCC_ClearFlag_HSERDY (void)

Function description

Clear HSE ready interrupt flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CICR HSERDYC LL_RCC_ClearFlag_HSERDY

LL_RCC_ClearFlag_MSIRDY

Function name

__STATIC_INLINE void LL_RCC_ClearFlag_MSIRDY (void)

Function description

Clear MSI ready interrupt flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CICR MSIRDYC LL_RCC_ClearFlag_MSIRDY

LL_RCC_ClearFlag_PLL1RDY

Function name

__STATIC_INLINE void LL_RCC_ClearFlag_PLL1RDY (void)

Function description

Clear PLL1 ready interrupt flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CICR PLL1RDYC LL_RCC_ClearFlag_PLL1RDY

LL_RCC_ClearFlag_PLL2RDY

Function name

__STATIC_INLINE void LL_RCC_ClearFlag_PLL2RDY (void)

Function description

Clear PLL2 ready interrupt flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CICR PLL2RDYC LL_RCC_ClearFlag_PLL2RDY

LL_RCC_ClearFlag_PLL3RDY

Function name

__STATIC_INLINE void LL_RCC_ClearFlag_PLL3RDY (void)

Function description

Clear PLL3 ready interrupt flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CICR PLL3RDYC LL_RCC_ClearFlag_PLL3RDY

LL_RCC_ClearFlag_PLL4RDY

Function name

__STATIC_INLINE void LL_RCC_ClearFlag_PLL4RDY (void)

Function description

Clear PLL4 ready interrupt flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CICR PLL4RDYC LL_RCC_ClearFlag_PLL4RDY

LL_RCC_ClearFlag_LSECSS

Function name

__STATIC_INLINE void LL_RCC_ClearFlag_LSECSS (void)

Function description

Clear LSE Clock security system interrupt flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CICR LSECSSC LL_RCC_ClearFlag_LSECSS

LL_RCC_ClearFlag_HSECSS

Function name

__STATIC_INLINE void LL_RCC_ClearFlag_HSECSS (void)

Function description

Clear HSE Clock security system interrupt flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CICR HSECSSC LL_RCC_ClearFlag_HSECSS

LL_RCC_ClearFlag_WKUP

Function name

__STATIC_INLINE void LL_RCC_ClearFlag_WKUP (void)

Function description

Clear WKUP ready interrupt flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CICR WKUPFC LL_RCC_ClearFlag_WKUP

LL_RCC_IsActiveFlag_LSIRDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_LSIRDY (void)

Function description

Check if LSI ready interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIFR LSIRDYF LL_RCC_IsActiveFlag_LSIRDY

LL_RCC_IsActiveFlag_LSERDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_LSERDY (void)

Function description

Check if LSE ready interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIFR LSERDYF LL_RCC_IsActiveFlag_LSERDY

LL_RCC_IsActiveFlag_HSIRDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_HSIRDY (void)

Function description

Check if HSI ready interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIFR HSIRDYF LL_RCC_IsActiveFlag_HSIRDY

LL_RCC_IsActiveFlag_HSERDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_HSERDY (void)

Function description

Check if HSE ready interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIFR HSERDYF LL_RCC_IsActiveFlag_HSERDY

LL_RCC_IsActiveFlag_MSIRDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_MSIRDY (void)

Function description

Check if MSI ready interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIFR MSIRDYF LL_RCC_IsActiveFlag_MSIRDY

LL_RCC_IsActiveFlag_PLL1RDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_PLL1RDY (void)

Function description

Check if PLL1 ready interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIFR PLL1RDYF LL_RCC_IsActiveFlag_PLL1RDY

LL_RCC_IsActiveFlag_PLL2RDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_PLL2RDY (void)

Function description

Check if PLL2 ready interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIFR PLL2RDYF LL_RCC_IsActiveFlag_PLL2RDY

LL_RCC_IsActiveFlag_PLL3RDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_PLL3RDY (void)

Function description

Check if PLL3 ready interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIFR PLL3RDYF LL_RCC_IsActiveFlag_PLL3RDY

LL_RCC_IsActiveFlag_PLL4RDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_PLL4RDY (void)

Function description

Check if PLL4 ready interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIFR PLL4RDYF LL_RCC_IsActiveFlag_PLL4RDY

LL_RCC_IsActiveFlag_LSECSS

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_LSECSS (void)

Function description

Check if LSE Clock security system interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIFR LSECSSF LL_RCC_IsActiveFlag_LSECSS

LL_RCC_IsActiveFlag_HSECSS

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_HSECSS (void)

Function description

Check if HSE Clock security system interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIFR HSECSSF LL_RCC_IsActiveFlag_HSECSS

LL_RCC_IsActiveFlag_WKUP

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_WKUP (void)

Function description

Check if WKUP from STOP interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIFR WKUPF LL_RCC_IsActiveFlag_WKUP

LL_RCC_IsActiveFlag_LPWRRST

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_LPWRRST (void)

Function description

Check if RCC flag Low Power D1 reset is set or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RSR LPWRRSTF LL_RCC_IsActiveFlag_LPWRRST

LL_RCC_IsActiveFlag_WWDGRST

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_WWDGRST (void)

Function description

Check if RCC flag Window Watchdog reset is set or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RSR WWDGRSTF LL_RCC_IsActiveFlag_WWDGRST

LL_RCC_IsActiveFlag_IWDGRST

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_IWDGRST (void)

Function description

Check if RCC flag Independent Watchdog reset is set or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RSR IWDGRSTF LL_RCC_IsActiveFlag_IWDGRST

LL_RCC_IsActiveFlag_SFTRST

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_SFTRST (void)

Function description

Check if RCC flag Software reset is set or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RSR SFTRSTF LL_RCC_IsActiveFlag_SFTRST

LL_RCC_IsActiveFlag_PORRST

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_PORRST (void)

Function description

Check if RCC flag POR/PDR reset is set or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RSR PORRSTF LL_RCC_IsActiveFlag_PORRST

LL_RCC_IsActiveFlag_PINRST

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_PINRST (void)

Function description

Check if RCC flag Pin reset is set or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RSR PINRSTF LL_RCC_IsActiveFlag_PINRST

LL_RCC_IsActiveFlag_BORRST

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_BORRST (void)

Function description

Check if RCC flag BOR reset is set or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RSR BORRSTF LL_RCC_IsActiveFlag_BORRST

LL_RCC_IsActiveFlag_LCKRST

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_LCKRST (void)

Function description

Check if RCC flag CPU Lockup reset is set or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RSR BORRSTF LL_RCC_IsActiveFlag_LCKRST

LL_RCC_IsActiveFlag_ETH1PWRDOWNACK

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_ETH1PWRDOWNACK (void)

Function description

Check if RCC ETH1 power-down start acknowledged is set or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CCIPR2 ETH1PWRDOWNACK LL_RCC_IsActiveFlag_ETH1PWRDOWNACK

LL_RCC_ClearResetFlags

Function name

__STATIC_INLINE void LL_RCC_ClearResetFlags (void)

Function description

Set RMVF bit to clear all reset flags.

Return values

- **None:**

Reference Manual to LL API cross reference:

- RSR RMVF LL_RCC_ClearResetFlags

LL_RCC_EnableIT_LSIRDY

Function name

__STATIC_INLINE void LL_RCC_EnableIT_LSIRDY (void)

Function description

Enable LSI ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER LSIRDYIE LL_RCC_EnableIT_LSIRDY

LL_RCC_EnableIT_LSERDY

Function name

__STATIC_INLINE void LL_RCC_EnableIT_LSERDY (void)

Function description

Enable LSE ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER LSERDYIE LL_RCC_EnableIT_LSERDY

LL_RCC_EnableIT_MSIRDY

Function name

__STATIC_INLINE void LL_RCC_EnableIT_MSIRDY (void)

Function description

Enable MSI ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER MSIRDYIE LL_RCC_EnableIT_MSIRDY

LL_RCC_EnableIT_HSIRDY

Function name

__STATIC_INLINE void LL_RCC_EnableIT_HSIRDY (void)

Function description

Enable HSI ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER HSIRDYIE LL_RCC_EnableIT_HSIRDY

LL_RCC_EnableIT_HSERDY

Function name

__STATIC_INLINE void LL_RCC_EnableIT_HSERDY (void)

Function description

Enable HSE ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER HSERDYIE LL_RCC_EnableIT_HSERDY

LL_RCC_EnableIT_PLL1RDY

Function name

__STATIC_INLINE void LL_RCC_EnableIT_PLL1RDY (void)

Function description

Enable PLL1 ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER PLL1RDYIE LL_RCC_EnableIT_PLL1RDY

LL_RCC_EnableIT_PLL2RDY

Function name

__STATIC_INLINE void LL_RCC_EnableIT_PLL2RDY (void)

Function description

Enable PLL2 ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER PLL2RDYIE LL_RCC_EnableIT_PLL2RDY

LL_RCC_EnableIT_PLL3RDY

Function name

__STATIC_INLINE void LL_RCC_EnableIT_PLL3RDY (void)

Function description

Enable PLL3 ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER PLL3RDYIE LL_RCC_EnableIT_PLL3RDY

LL_RCC_EnableIT_PLL4RDY

Function name

__STATIC_INLINE void LL_RCC_EnableIT_PLL4RDY (void)

Function description

Enable PLL4 ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER PLL4RDYIE LL_RCC_EnableIT_PLL4RDY

LL_RCC_EnableIT_LSECSS

Function name

__STATIC_INLINE void LL_RCC_EnableIT_LSECSS (void)

Function description

Enable LSECSS interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER LSECSSIE LL_RCC_EnableIT_LSECSS

LL_RCC_EnableIT_HSECSS

Function name

__STATIC_INLINE void LL_RCC_EnableIT_HSECSS (void)

Function description

Enable HSECSS interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER HSECSSIE LL_RCC_EnableIT_HSECSS

LL_RCC_EnableIT_WKUP

Function name

__STATIC_INLINE void LL_RCC_EnableIT_WKUP (void)

Function description

Enable WKUP interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER WKUPIE LL_RCC_EnableIT_WKUP

LL_RCC_DisableIT_LSIRDY

Function name

__STATIC_INLINE void LL_RCC_DisableIT_LSIRDY (void)

Function description

Disable LSI ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER LSIRDYIE LL_RCC_DisableIT_LSIRDY

LL_RCC_DisableIT_LSERDY

Function name

__STATIC_INLINE void LL_RCC_DisableIT_LSERDY (void)

Function description

Disable LSE ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER LSERDYIE LL_RCC_DisableIT_LSERDY

LL_RCC_DisableIT_MSIRDY

Function name

__STATIC_INLINE void LL_RCC_DisableIT_MSIRDY (void)

Function description

Disable MSI ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER MSIRDYIE LL_RCC_DisableIT_MSIRDY

LL_RCC_DisableIT_HSIRDY

Function name

__STATIC_INLINE void LL_RCC_DisableIT_HSIRDY (void)

Function description

Disable HSI ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER HSIRDYIE LL_RCC_DisableIT_HSIRDY

LL_RCC_DisableIT_HSERDY

Function name

__STATIC_INLINE void LL_RCC_DisableIT_HSERDY (void)

Function description

Disable HSE ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER HSERDYIE LL_RCC_DisableIT_HSERDY

LL_RCC_DisableIT_PLL1RDY

Function name

__STATIC_INLINE void LL_RCC_DisableIT_PLL1RDY (void)

Function description

Disable PLL1 ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER PLL1RDYIE LL_RCC_DisableIT_PLL1RDY

LL_RCC_DisableIT_PLL2RDY

Function name

__STATIC_INLINE void LL_RCC_DisableIT_PLL2RDY (void)

Function description

Disable PLL2 ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER PLL2RDYIE LL_RCC_DisableIT_PLL2RDY

LL_RCC_DisableIT_PLL3RDY

Function name

__STATIC_INLINE void LL_RCC_DisableIT_PLL3RDY (void)

Function description

Disable PLL3 ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER PLL3RDYIE LL_RCC_DisableIT_PLL3RDY

LL_RCC_DisableIT_PLL4RDY

Function name

__STATIC_INLINE void LL_RCC_DisableIT_PLL4RDY (void)

Function description

Disable PLL4 ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER PLL4RDYIE LL_RCC_DisableIT_PLL4RDY

LL_RCC_DisableIT_LSECSS

Function name

__STATIC_INLINE void LL_RCC_DisableIT_LSECSS (void)

Function description

Disable LSECSS interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER LSECSSIE LL_RCC_DisableIT_LSECSS

LL_RCC_DisableIT_HSECSS

Function name

__STATIC_INLINE void LL_RCC_DisableIT_HSECSS (void)

Function description

Disable HSECSS interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER HSECSSIE LL_RCC_DisableIT_HSECSS

LL_RCC_DisableIT_WKUP

Function name

__STATIC_INLINE void LL_RCC_DisableIT_WKUP (void)

Function description

Disable WKUP interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER WKUPIE LL_RCC_DisableIT_WKUP

LL_RCC_IsEnabledIT_LSIRDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_LSIRDY (void)

Function description

Checks if LSI ready interrupt source is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIER LSIRDYIE LL_RCC_IsEnabledIT_LSIRDY

LL_RCC_IsEnabledIT_LSERDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_LSERDY (void)

Function description

Checks if LSE ready interrupt source is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIER LSERDYIE LL_RCC_IsEnabledIT_LSERDY

LL_RCC_IsEnabledIT_MSIRDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_MSIRDY (void)

Function description

Checks if MSI ready interrupt source is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIER MSIRDYIE LL_RCC_IsEnabledIT_MSIRDY

LL_RCC_IsEnabledIT_HSIRDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_HSIRDY (void)

Function description

Checks if HSI ready interrupt source is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIER HSIRDYIE LL_RCC_IsEnabledIT_HSIRDY

LL_RCC_IsEnabledIT_HSERDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_HSERDY (void)

Function description

Checks if HSE ready interrupt source is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIER HSERDYIE LL_RCC_IsEnabledIT_HSERDY

LL_RCC_IsEnabledIT_PLL1RDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_PLL1RDY (void)

Function description

Checks if PLL1 ready interrupt source is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIER PLL1RDYIE LL_RCC_IsEnabledIT_PLL1RDY

LL_RCC_IsEnabledIT_PLL2RDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_PLL2RDY (void)

Function description

Checks if PLL2 ready interrupt source is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIER PLL2RDYIE LL_RCC_IsEnabledIT_PLL2RDY

LL_RCC_IsEnabledIT_PLL3RDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_PLL3RDY (void)

Function description

Checks if PLL3 ready interrupt source is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIER PLL3RDYIE LL_RCC_IsEnabledIT_PLL3RDY

LL_RCC_IsEnabledIT_PLL4RDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_PLL4RDY (void)

Function description

Checks if PLL4 ready interrupt source is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIER PLL4RDYIE LL_RCC_IsEnabledIT_PLL4RDY

LL_RCC_IsEnabledIT_LSECSS

Function name

__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_LSECSS (void)

Function description

Checks if LSECSS interrupt source is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIER LSECSSIE LL_RCC_IsEnabledIT_LSECSS

LL_RCC_IsEnabledIT_HSECSS

Function name

__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_HSECSS (void)

Function description

Checks if HSECSS interrupt source is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIER HSECSSIE LL_RCC_IsEnabledIT_HSECSS

LL_RCC_IsEnabledIT_WKUP

Function name

__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_WKUP (void)

Function description

Checks if WKUP interrupt source is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIER WKUPIE LL_RCC_IsEnabledIT_WKUP

LL_RCC_DeInit

Function name

void LL_RCC_DeInit (void)

Function description

Resets the RCC clock configuration to the default reset state.

Return values

- **None:**

Notes

- The default reset state of the clock configuration is given below: HSI ON and used as system clock sourceHSE, PLL1, PLL2 and PLL3 OFFAHB, APB Bus pre-scaler set to 1.CSS, MCO1 and MCO2 OFFAll interrupts disabled
- This function doesn't modify the configuration of the Peripheral clocksLSI, LSE and RTC clocks

LL_RCC_GetSystemClocksFreq

Function name

void LL_RCC_GetSystemClocksFreq (LL_RCC_ClocksTypeDef * RCC_Clocks)

Function description

Return the frequencies of different on chip clocks; CPU, System bus, AHB, APB1, APB2, APB4 and APB5 buses clocks.

Parameters

- **RCC_Clocks:** pointer to a LL_RCC_ClocksTypeDef structure which will hold the clocks frequencies

Return values

- **None:**

Notes

- Each time CPUCLK, SYSCLK, HCLK, PCLK1, PCLK2, PCLK4 and/or PCLK5 clock changes, this function must be called to update application structure fields. Otherwise, any configuration based on this function will be incorrect.

LL_RCC_GetCpuClockFreq

Function name

uint32_t LL_RCC_GetCpuClockFreq (void)

Function description

Return CPU clock frequency.

Return values

- **CPU:** clock frequency (in Hz)

LL_RCC_GetSystemClockFreq

Function name

uint32_t LL_RCC_GetSystemClockFreq (void)

Function description

Return SYSTEM bus clock frequency.

Return values

- **SYSTEM:** bus clock frequency (in Hz)

LL_RCC_GetPLL1ClockFreq

Function name

uint32_t LL_RCC_GetPLL1ClockFreq (void)

Function description

Return PLL1 clock frequency.

Return values

- **PLL1:** output clock frequency

Notes

- LL_RCC_PERIPH_FREQUENCY_NO returned for non activated output or oscillator not ready

LL_RCC_GetPLL2ClockFreq

Function name

uint32_t LL_RCC_GetPLL2ClockFreq (void)

Function description

Return PLL2 clock frequency.

Return values

- **PLL2:** output clock frequency

Notes

- LL_RCC_PERIPH_FREQUENCY_NO returned for non activated output or oscillator not ready

LL_RCC_GetPLL3ClockFreq

Function name

uint32_t LL_RCC_GetPLL3ClockFreq (void)

Function description

Return PLL3 clock frequency.

Return values

- **PLL3:** output clock frequency

Notes

- LL_RCC_PERIPH_FREQUENCY_NO returned for non activated output or oscillator not ready

LL_RCC_GetPLL4ClockFreq

Function name

uint32_t LL_RCC_GetPLL4ClockFreq (void)

Function description

Return PLL4 clock frequency.

Return values

- **PLL4:** output clock frequency

Notes

- LL_RCC_PERIPH_FREQUENCY_NO returned for non activated output or oscillator not ready

LL_RCC_CalcPLLClockFreq

Function name

uint32_t LL_RCC_CalcPLLClockFreq (uint32_t PLLInputFreq, uint32_t M, uint32_t N, uint32_t FRACN, uint32_t P1, uint32_t P2)

Function description

Helper function to calculate the PLL frequency output when used in integer or fractional mode.

Parameters

- **PLLInputFreq:** PLL Input frequency (based on HSE/(HSI/HSIDIV)/MSI)
- **M:** Between 1 and 63
- **N:** Between 16 and 640 in integer mode, 20 to 320 in fractional mode
- **FRACN:** 0 in integer mode, between 0 and 0xFFFFFF in fractional mode
- **P1:** VCO output divider P1 between 1 and 7
- **P2:** VCO output divider P2 between 1 and 7

Return values

- **PLL:** clock frequency (in Hz)

Notes

- ex: LL_RCC_CalcPLLClockFreq (HSE_VALUE, LL_RCC_PLL1_GetM (), LL_RCC_PLL1_GetN (), LL_RCC_PLL1_GetFRACN (), LL_RCC_PLL1_GetP1 (), LL_RCC_PLL1_GetP2 ());

LL_RCC_GetADCClockFreq

Function name

uint32_t LL_RCC_GetADCClockFreq (uint32_t ADCxSource)

Function description

Return ADC clock frequency.

Parameters

- **ADCxSource:** This parameter can be one of the following values:
  - LL_RCC_ADC_CLKSOURCE

Return values

- **ADC:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator is not ready

LL_RCC_GetADFClockFreq

Function name

uint32_t LL_RCC_GetADFClockFreq (uint32_t ADFxSource)

Function description

Return ADFx clock frequency.

Parameters

- **ADFxSource:** This parameter can be one of the following values:
  - LL_RCC_ADF1_CLKSOURCE

Return values

- **ADF:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator is not ready

LL_RCC_GetCLKPClockFreq

Function name

uint32_t LL_RCC_GetCLKPClockFreq (uint32_t CLKPxSource)

Function description

Return CLKP clock frequency.

Parameters

- **CLKPxSource:** This parameter can be one of the following values:
  - LL_RCC_CLKP_CLKSOURCE

Return values

- **CLKP:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator is not ready

LL_RCC_GetDCMIPPClockFreq

Function name

uint32_t LL_RCC_GetDCMIPPClockFreq (uint32_t DCMIPPxSource)

Function description

Return DCMIPP clock frequency.

Parameters

- **DCMIPPxSource:** This parameter can be one of the following values:
  - LL_RCC_DCMIPP_CLKSOURCE

Return values

- **DCMIPP:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator is not ready

LL_RCC_GetETHClockFreq

Function name

uint32_t LL_RCC_GetETHClockFreq (uint32_t ETHxSource)

Function description

Return ETH clock frequency.

Parameters

- **ETHxSource:** This parameter can be one of the following values:
  - LL_RCC_ETH1_CLKSOURCE

Return values

- **ETH1:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator is not ready

LL_RCC_GetETHPTPClockFreq

Function name

uint32_t LL_RCC_GetETHPTPClockFreq (uint32_t ETHxPTPSource)

Function description

Return ETH PTP clock frequency.

Parameters

- **ETHxPTPSource:** This parameter can be one of the following values:
  - LL_RCC_ETH1PTP_CLKSOURCE

Return values

- **ETH1:** PTP clock frequency with divider (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator is not ready

LL_RCC_GetFDCANClockFreq

Function name

uint32_t LL_RCC_GetFDCANClockFreq (uint32_t FDCANxSource)

Function description

Return FDCAN clock frequency.

Parameters

- **FDCANxSource:** This parameter can be one of the following values:
  - LL_RCC_FDCAN_CLKSOURCE

Return values

- **FDCAN:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator is not ready

LL_RCC_GetFMCClockFreq

Function name

uint32_t LL_RCC_GetFMCClockFreq (uint32_t FMCxSource)

Function description

Return FMC clock frequency.

Parameters

- **FMCxSource:** This parameter can be one of the following values:
  - LL_RCC_FMC_CLKSOURCE

Return values

- **FMC:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator is not ready

LL_RCC_GetI2CClockFreq

Function name

uint32_t LL_RCC_GetI2CClockFreq (uint32_t I2CxSource)

Function description

Return I2Cx clock frequency.

Parameters

- **I2CxSource:** This parameter can be one of the following values:
  - LL_RCC_I2C1_CLKSOURCE
  - LL_RCC_I2C2_CLKSOURCE
  - LL_RCC_I2C3_CLKSOURCE
  - LL_RCC_I2C4_CLKSOURCE

Return values

- **I2C:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator is not ready

LL_RCC_GetI3CClockFreq

Function name

uint32_t LL_RCC_GetI3CClockFreq (uint32_t I3CxSource)

Function description

Return I3Cx clock frequency.

Parameters

- **I3CxSource:** This parameter can be one of the following values:
  - LL_RCC_I3C1_CLKSOURCE
  - LL_RCC_I3C2_CLKSOURCE

Return values

- **I3C:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator is not ready

LL_RCC_GetLPTIMClockFreq

Function name

uint32_t LL_RCC_GetLPTIMClockFreq (uint32_t LPTIMxSource)

Function description

Return LPTIMx clock frequency.

Parameters

- **LPTIMxSource:** This parameter can be one of the following values:
  - LL_RCC_LPTIM1_CLKSOURCE
  - LL_RCC_LPTIM2_CLKSOURCE
  - LL_RCC_LPTIM3_CLKSOURCE
  - LL_RCC_LPTIM4_CLKSOURCE
  - LL_RCC_LPTIM5_CLKSOURCE

Return values

- **LPTIM:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator is not ready

LL_RCC_GetLPUARTClockFreq

Function name

uint32_t LL_RCC_GetLPUARTClockFreq (uint32_t LPUARTxSource)

Function description

Return LPUART clock frequency.

Parameters

- **LPUARTxSource:** This parameter can be one of the following values:
  - LL_RCC_LPUART1_CLKSOURCE

Return values

- **LPUART:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator is not ready

LL_RCC_GetLTDCClockFreq

Function name

uint32_t LL_RCC_GetLTDCClockFreq (uint32_t LTDCxSource)

Function description

Return LTDC clock frequency.

Parameters

- **LTDCxSource:** This parameter can be one of the following values:
  - LL_RCC_LTDC_CLKSOURCE

Return values

- **LTDC:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator is not ready

LL_RCC_GetMDFClockFreq

Function name

uint32_t LL_RCC_GetMDFClockFreq (uint32_t MDFxSource)

Function description

Return MDFx clock frequency.

Parameters

- **MDFxSource:** This parameter can be one of the following values:
  - LL_RCC_MDF1_CLKSOURCE

Return values

- **MDF:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator is not ready

LL_RCC_GetOTGPHYClockFreq

Function name

uint32_t LL_RCC_GetOTGPHYClockFreq (uint32_t OTGPHYxSource)

Function description

Return OTGPHYx clock frequency.

Parameters

- **OTGPHYxSource:** This parameter can be one of the following values:
  - LL_RCC_OTGPHY1_CLKSOURCE
  - LL_RCC_OTGPHY2_CLKSOURCE

Return values

- **OTGPHY:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator is not ready

LL_RCC_GetOTGPHYCKREFClockFreq

Function name

uint32_t LL_RCC_GetOTGPHYCKREFClockFreq (uint32_t OTGPHYxCKREFSource)

Function description

Return OTGPHYxCKREF clock frequency.

Parameters

- **OTGPHYxCKREFSource:** This parameter can be one of the following values:
  - LL_RCC_OTGPHY1CKREF_CLKSOURCE
  - LL_RCC_OTGPHY2CKREF_CLKSOURCE

Return values

- **OTGPHYCKREF:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator is not ready

LL_RCC_GetPSSIClockFreq

Function name

uint32_t LL_RCC_GetPSSIClockFreq (uint32_t PSSIxSource)

Function description

Return PSSI clock frequency.

Parameters

- **PSSIxSource:** This parameter can be one of the following values:
  - LL_RCC_PSSI_CLKSOURCE

Return values

- **PSSI:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator is not ready

LL_RCC_GetSAIClockFreq

Function name

uint32_t LL_RCC_GetSAIClockFreq (uint32_t SAIxSource)

Function description

Return SAIx clock frequency.

Parameters

- **SAIxSource:** This parameter can be one of the following values:
  - LL_RCC_SAI1_CLKSOURCE
  - LL_RCC_SAI2_CLKSOURCE

Return values

- **SAI:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator is not ready

LL_RCC_GetSDMMCClockFreq

Function name

uint32_t LL_RCC_GetSDMMCClockFreq (uint32_t SDMMCxSource)

Function description

Return SDMMC clock frequency.

Parameters

- **SDMMCxSource:** This parameter can be one of the following values:
  - LL_RCC_SDMMC1_CLKSOURCE
  - LL_RCC_SDMMC2_CLKSOURCE

Return values

- **SDMMC:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator is not ready

LL_RCC_GetSPDIFRXClockFreq

Function name

uint32_t LL_RCC_GetSPDIFRXClockFreq (uint32_t SPDIFRXxSource)

Function description

Return SPDIFRX clock frequency.

Parameters

- **SPDIFRXxSource:** This parameter can be one of the following values:
  - LL_RCC_SPDIFRX1_CLKSOURCE

Return values

- **SPDIF:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator is not ready

LL_RCC_GetSPIClockFreq

Function name

uint32_t LL_RCC_GetSPIClockFreq (uint32_t SPIxSource)

Function description

Return SPIx clock frequency.

Parameters

- **SPIxSource:** This parameter can be one of the following values:
  - LL_RCC_SPI1_CLKSOURCE
  - LL_RCC_SPI2_CLKSOURCE
  - LL_RCC_SPI3_CLKSOURCE
  - LL_RCC_SPI4_CLKSOURCE
  - LL_RCC_SPI5_CLKSOURCE
  - LL_RCC_SPI6_CLKSOURCE

Return values

- **SPI:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator is not ready

LL_RCC_GetUARTClockFreq

Function name

uint32_t LL_RCC_GetUARTClockFreq (uint32_t UARTxSource)

Function description

Return UARTx clock frequency.

Parameters

- **UARTxSource:** This parameter can be one of the following values:
  - LL_RCC_UART4_CLKSOURCE
  - LL_RCC_UART5_CLKSOURCE
  - LL_RCC_UART7_CLKSOURCE
  - LL_RCC_UART8_CLKSOURCE
  - LL_RCC_UART9_CLKSOURCE

Return values

- **USART:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator is not ready

LL_RCC_GetUSARTClockFreq

Function name

uint32_t LL_RCC_GetUSARTClockFreq (uint32_t USARTxSource)

Function description

Return USARTx clock frequency.

Parameters

- **USARTxSource:** This parameter can be one of the following values:
  - LL_RCC_USART1_CLKSOURCE
  - LL_RCC_USART2_CLKSOURCE
  - LL_RCC_USART3_CLKSOURCE
  - LL_RCC_USART6_CLKSOURCE
  - LL_RCC_USART10_CLKSOURCE

Return values

- **USART:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator is not ready

LL_RCC_GetUSBClockFreq

Function name

uint32_t LL_RCC_GetUSBClockFreq (uint32_t USBxSource)

Function description

LL_RCC_GetXSPIClockFreq

Function name

uint32_t LL_RCC_GetXSPIClockFreq (uint32_t XSPIxSource)

Function description

Return XSPI clock frequency.

Parameters

- **XSPIxSource:** This parameter can be one of the following values:
  - LL_RCC_XSPI1_CLKSOURCE
  - LL_RCC_XSPI2_CLKSOURCE
  - LL_RCC_XSPI3_CLKSOURCE

Return values

- **XSPI:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator is not ready

101.3 RCC Firmware driver defines

The following section lists the various define and macros of the module.

101.3.1 RCC

RCC

Peripheral ADC get clock source

LL_RCC_ADC_CLKSOURCE

Peripheral ADC clock source selection

LL_RCC_ADC_CLKSOURCE_HCLK

LL_RCC_ADC_CLKSOURCE_CLKP

LL_RCC_ADC_CLKSOURCE_IC7

LL_RCC_ADC_CLKSOURCE_IC8

LL_RCC_ADC_CLKSOURCE_MSI

LL_RCC_ADC_CLKSOURCE_HSI

LL_RCC_ADC_CLKSOURCE_I2S_CKIN

LL_RCC_ADC_CLKSOURCE_TIMG

Peripheral ADF get clock source

LL_RCC_ADF1_CLKSOURCE

Peripheral ADF clock source selection

LL_RCC_ADF1_CLKSOURCE_HCLK

LL_RCC_ADF1_CLKSOURCE_CLKP

LL_RCC_ADF1_CLKSOURCE_IC7

LL_RCC_ADF1_CLKSOURCE_IC8

LL_RCC_ADF1_CLKSOURCE_MSI

LL_RCC_ADF1_CLKSOURCE_HSI

LL_RCC_ADF1_CLKSOURCE_I2S_CKIN

LL_RCC_ADF1_CLKSOURCE_TIMG

AHB prescaler

LL_RCC_AHB_DIV_1

LL_RCC_AHB_DIV_2

LL_RCC_AHB_DIV_4

LL_RCC_AHB_DIV_8

LL_RCC_AHB_DIV_16

LL_RCC_AHB_DIV_32

LL_RCC_AHB_DIV_64

LL_RCC_AHB_DIV_128

APB1 prescaler

LL_RCC_APB1_DIV_1

LL_RCC_APB1_DIV_2

LL_RCC_APB1_DIV_4

LL_RCC_APB1_DIV_8

LL_RCC_APB1_DIV_16

LL_RCC_APB1_DIV_32

LL_RCC_APB1_DIV_64

LL_RCC_APB1_DIV_128

APB2 prescaler

LL_RCC_APB2_DIV_1

LL_RCC_APB2_DIV_2

LL_RCC_APB2_DIV_4

LL_RCC_APB2_DIV_8

LL_RCC_APB2_DIV_16

LL_RCC_APB2_DIV_32

LL_RCC_APB2_DIV_64

LL_RCC_APB2_DIV_128

APB4 prescaler

LL_RCC_APB4_DIV_1

LL_RCC_APB4_DIV_2

LL_RCC_APB4_DIV_4

LL_RCC_APB4_DIV_8

LL_RCC_APB4_DIV_16

LL_RCC_APB4_DIV_32

LL_RCC_APB4_DIV_64

LL_RCC_APB4_DIV_128

APB5 prescaler

LL_RCC_APB5_DIV_1

LL_RCC_APB5_DIV_2

LL_RCC_APB5_DIV_4

LL_RCC_APB5_DIV_8

LL_RCC_APB5_DIV_16

LL_RCC_APB5_DIV_32

LL_RCC_APB5_DIV_64

LL_RCC_APB5_DIV_128

Peripheral CLKP get clock source

LL_RCC_CLKP_CLKSOURCE

Peripheral CLKP clock source selection

LL_RCC_CLKP_CLKSOURCE_HSI

LL_RCC_CLKP_CLKSOURCE_MSI

LL_RCC_CLKP_CLKSOURCE_HSE

LL_RCC_CLKP_CLKSOURCE_IC19

LL_RCC_CLKP_CLKSOURCE_IC5

LL_RCC_CLKP_CLKSOURCE_IC10

LL_RCC_CLKP_CLKSOURCE_IC15

LL_RCC_CLKP_CLKSOURCE_IC20

CPU clock switch

LL_RCC_CPU_CLKSOURCE_HSI
Select HSI as CPU clock

LL_RCC_CPU_CLKSOURCE_MSI
Select MSI as CPU clock

LL_RCC_CPU_CLKSOURCE_HSE
Select HSE as CPU clock

LL_RCC_CPU_CLKSOURCE_IC1
Select IC1 as CPU clock

CPU clock switch status

LL_RCC_CPU_CLKSOURCE_STATUS_HSI
HSI used as CPU clock

LL_RCC_CPU_CLKSOURCE_STATUS_MSI
MSI used as CPU clock

LL_RCC_CPU_CLKSOURCE_STATUS_HSE
HSE used as CPU clock

LL_RCC_CPU_CLKSOURCE_STATUS_IC1
IC1 used as CPU clock

Peripheral DCMIPP get clock source

LL_RCC_DCMIPP_CLKSOURCE

Peripheral DCMIPP clock source selection

LL_RCC_DCMIPP_CLKSOURCE_PCLK5

LL_RCC_DCMIPP_CLKSOURCE_CLKP

LL_RCC_DCMIPP_CLKSOURCE_IC17

LL_RCC_DCMIPP_CLKSOURCE_HSI

Peripheral ETH get clock source

LL_RCC_ETH1_CLKSOURCE

ETH1 PTP kernel clock divider selection

LL_RCC_ETH1PTP_DIV_1

LL_RCC_ETH1PTP_DIV_2

LL_RCC_ETH1PTP_DIV_3

LL_RCC_ETH1PTP_DIV_4

LL_RCC_ETH1PTP_DIV_5

LL_RCC_ETH1PTP_DIV_6

LL_RCC_ETH1PTP_DIV_7

LL_RCC_ETH1PTP_DIV_8

LL_RCC_ETH1PTP_DIV_9

LL_RCC_ETH1PTP_DIV_10

LL_RCC_ETH1PTP_DIV_11

LL_RCC_ETH1PTP_DIV_12

LL_RCC_ETH1PTP_DIV_13

LL_RCC_ETH1PTP_DIV_14

LL_RCC_ETH1PTP_DIV_15

LL_RCC_ETH1PTP_DIV_16

Peripheral ETH PHY get interface

LL_RCC_ETH1PHY_IF

Peripheral ETH PHY interface selection

LL_RCC_ETH1PHY_IF_MII

LL_RCC_ETH1PHY_IF_RGMII

LL_RCC_ETH1PHY_IF_RMII

Peripheral ETHPTP get clock source

LL_RCC_ETH1PTP_CLKSOURCE

Peripheral ETH PTP kernel clock source selection

LL_RCC_ETH1PTP_CLKSOURCE_HCLK

LL_RCC_ETH1PTP_CLKSOURCE_CLKP

LL_RCC_ETH1PTP_CLKSOURCE_IC13

LL_RCC_ETH1PTP_CLKSOURCE_HSE

Peripheral ETH Reference RX get clock source

LL_RCC_ETH1REFRX_CLKSOURCE

Peripheral ETH Reference RX clock source selection

LL_RCC_ETH1REFRX_CLKSOURCE_EXT

LL_RCC_ETH1REFRX_CLKSOURCE_INT

Peripheral ETH Reference TX get clock source

LL_RCC_ETH1REFTX_CLKSOURCE

Peripheral ETH Reference TX clock source selection

LL_RCC_ETH1REFTX_CLKSOURCE_EXT

LL_RCC_ETH1REFTX_CLKSOURCE_INT

Peripheral ETH kernel clock source selection

LL_RCC_ETH1_CLKSOURCE_HCLK

LL_RCC_ETH1_CLKSOURCE_CLKP

LL_RCC_ETH1_CLKSOURCE_IC12

LL_RCC_ETH1_CLKSOURCE_HSE

Peripheral FDCAN get clock source

LL_RCC_FDCAN_CLKSOURCE

Peripheral FDCAN clock source selection

LL_RCC_FDCAN_CLKSOURCE_PCLK1

LL_RCC_FDCAN_CLKSOURCE_CLKP

LL_RCC_FDCAN_CLKSOURCE_IC19

LL_RCC_FDCAN_CLKSOURCE_HSE

Peripheral FMC get clock source

LL_RCC_FMC_CLKSOURCE

Peripheral FMC clock source selection

LL_RCC_FMC_CLKSOURCE_HCLK

LL_RCC_FMC_CLKSOURCE_CLKP

LL_RCC_FMC_CLKSOURCE_IC3

LL_RCC_FMC_CLKSOURCE_IC4

HSI divider to apply for replacement of HSE on CSS bypass

LL_RCC_HSECSSBYP_DIV_1

LL_RCC_HSECSSBYP_DIV_2

LL_RCC_HSECSSBYP_DIV_3

LL_RCC_HSECSSBYP_DIV_4

LL_RCC_HSECSSBYP_DIV_5

LL_RCC_HSECSSBYP_DIV_6

LL_RCC_HSECSSBYP_DIV_7

LL_RCC_HSECSSBYP_DIV_8

LL_RCC_HSECSSBYP_DIV_9

LL_RCC_HSECSSBYP_DIV_10

LL_RCC_HSECSSBYP_DIV_11

LL_RCC_HSECSSBYP_DIV_12

LL_RCC_HSECSSBYP_DIV_13

LL_RCC_HSECSSBYP_DIV_14

LL_RCC_HSECSSBYP_DIV_15

LL_RCC_HSECSSBYP_DIV_16

HSI oscillator divider

LL_RCC_HSI_DIV_1

LL_RCC_HSI_DIV_2

LL_RCC_HSI_DIV_4

LL_RCC_HSI_DIV_8

Peripheral I2C get clock source

LL_RCC_I2C1_CLKSOURCE

LL_RCC_I2C2_CLKSOURCE

LL_RCC_I2C3_CLKSOURCE

LL_RCC_I2C4_CLKSOURCE

Peripheral I2C clock source selection

LL_RCC_I2C1_CLKSOURCE_PCLK1

LL_RCC_I2C1_CLKSOURCE_CLKP

LL_RCC_I2C1_CLKSOURCE_IC10

LL_RCC_I2C1_CLKSOURCE_IC15

LL_RCC_I2C1_CLKSOURCE_MSI

LL_RCC_I2C1_CLKSOURCE_HSI

LL_RCC_I2C2_CLKSOURCE_PCLK1

LL_RCC_I2C2_CLKSOURCE_CLKP

LL_RCC_I2C2_CLKSOURCE_IC10

LL_RCC_I2C2_CLKSOURCE_IC15

LL_RCC_I2C2_CLKSOURCE_MSI

LL_RCC_I2C2_CLKSOURCE_HSI

LL_RCC_I2C3_CLKSOURCE_PCLK1

LL_RCC_I2C3_CLKSOURCE_CLKP

LL_RCC_I2C3_CLKSOURCE_IC10

LL_RCC_I2C3_CLKSOURCE_IC15

LL_RCC_I2C3_CLKSOURCE_MSI

LL_RCC_I2C3_CLKSOURCE_HSI

LL_RCC_I2C4_CLKSOURCE_PCLK1

LL_RCC_I2C4_CLKSOURCE_CLKP

LL_RCC_I2C4_CLKSOURCE_IC10

LL_RCC_I2C4_CLKSOURCE_IC15

LL_RCC_I2C4_CLKSOURCE_MSI

LL_RCC_I2C4_CLKSOURCE_HSI

Peripheral I3C get clock source

LL_RCC_I3C1_CLKSOURCE

LL_RCC_I3C2_CLKSOURCE

Peripheral I3C clock source selection

LL_RCC_I3C1_CLKSOURCE_PCLK1

LL_RCC_I3C1_CLKSOURCE_CLKP

LL_RCC_I3C1_CLKSOURCE_IC10

LL_RCC_I3C1_CLKSOURCE_IC15

LL_RCC_I3C1_CLKSOURCE_MSI

LL_RCC_I3C1_CLKSOURCE_HSI

LL_RCC_I3C2_CLKSOURCE_PCLK1

LL_RCC_I3C2_CLKSOURCE_CLKP

LL_RCC_I3C2_CLKSOURCE_IC10

LL_RCC_I3C2_CLKSOURCE_IC15

LL_RCC_I3C2_CLKSOURCE_MSI

LL_RCC_I3C2_CLKSOURCE_HSI

All ICs entry clock source

LL_RCC_ICCLKSOURCE_PLL1

LL_RCC_ICCLKSOURCE_PLL2

LL_RCC_ICCLKSOURCE_PLL3

LL_RCC_ICCLKSOURCE_PLL4

Peripheral LPTIM get clock source

LL_RCC_LPTIM1_CLKSOURCE

LL_RCC_LPTIM2_CLKSOURCE

LL_RCC_LPTIM3_CLKSOURCE

LL_RCC_LPTIM4_CLKSOURCE

LL_RCC_LPTIM5_CLKSOURCE

Peripheral LPTIM clock source selection

LL_RCC_LPTIM1_CLKSOURCE_PCLK1

LL_RCC_LPTIM1_CLKSOURCE_CLKP

LL_RCC_LPTIM1_CLKSOURCE_IC15

LL_RCC_LPTIM1_CLKSOURCE_LSE

LL_RCC_LPTIM1_CLKSOURCE_LSI

LL_RCC_LPTIM1_CLKSOURCE_TIMG

LL_RCC_LPTIM2_CLKSOURCE_PCLK4

LL_RCC_LPTIM2_CLKSOURCE_CLKP

LL_RCC_LPTIM2_CLKSOURCE_IC15

LL_RCC_LPTIM2_CLKSOURCE_LSE

LL_RCC_LPTIM2_CLKSOURCE_LSI

LL_RCC_LPTIM2_CLKSOURCE_TIMG

LL_RCC_LPTIM3_CLKSOURCE_PCLK4

LL_RCC_LPTIM3_CLKSOURCE_CLKP

LL_RCC_LPTIM3_CLKSOURCE_IC15

LL_RCC_LPTIM3_CLKSOURCE_LSE

LL_RCC_LPTIM3_CLKSOURCE_LSI

LL_RCC_LPTIM3_CLKSOURCE_TIMG

LL_RCC_LPTIM4_CLKSOURCE_PCLK4

LL_RCC_LPTIM4_CLKSOURCE_CLKP

LL_RCC_LPTIM4_CLKSOURCE_IC15

LL_RCC_LPTIM4_CLKSOURCE_LSE

LL_RCC_LPTIM4_CLKSOURCE_LSI

LL_RCC_LPTIM4_CLKSOURCE_TIMG

LL_RCC_LPTIM5_CLKSOURCE_PCLK4

LL_RCC_LPTIM5_CLKSOURCE_CLKP

LL_RCC_LPTIM5_CLKSOURCE_IC15

LL_RCC_LPTIM5_CLKSOURCE_LSE

LL_RCC_LPTIM5_CLKSOURCE_LSI

LL_RCC_LPTIM5_CLKSOURCE_TIMG

Peripheral LPUART get clock source

LL_RCC_LPUART1_CLKSOURCE

Peripheral LPUART clock source selection

LL_RCC_LPUART1_CLKSOURCE_PCLK4

LL_RCC_LPUART1_CLKSOURCE_CLKP

LL_RCC_LPUART1_CLKSOURCE_IC9

LL_RCC_LPUART1_CLKSOURCE_IC14

LL_RCC_LPUART1_CLKSOURCE_LSE

LL_RCC_LPUART1_CLKSOURCE_MSI

LL_RCC_LPUART1_CLKSOURCE_HSI

LSE oscillator drive capability

LL_RCC_LSEDRIVE_LOW

LL_RCC_LSEDRIVE_MEDIUMLOW

LL_RCC_LSEDRIVE_MEDIUMHIGH

LL_RCC_LSEDRIVE_HIGH

Peripheral LTDC get clock source

LL_RCC_LTDC_CLKSOURCE

Peripheral LTDC clock source selection

LL_RCC_LTDC_CLKSOURCE_PCLK5

LL_RCC_LTDC_CLKSOURCE_CLKP

LL_RCC_LTDC_CLKSOURCE_IC16

LL_RCC_LTDC_CLKSOURCE_HSI

MCO selection

LL_RCC_MCO1

LL_RCC_MCO2

MCO source selection

LL_RCC_MCO1SOURCE_HSI

LL_RCC_MCO1SOURCE_LSE

LL_RCC_MCO1SOURCE_MSI

LL_RCC_MCO1SOURCE_LSI

LL_RCC_MCO1SOURCE_HSE

LL_RCC_MCO1SOURCE_IC5

LL_RCC_MCO1SOURCE_IC10

LL_RCC_MCO1SOURCE_SYSA

LL_RCC_MCO2SOURCE_HSI

LL_RCC_MCO2SOURCE_LSE

LL_RCC_MCO2SOURCE_MSI

LL_RCC_MCO2SOURCE_LSI

LL_RCC_MCO2SOURCE_HSE

LL_RCC_MCO2SOURCE_IC15

LL_RCC_MCO2SOURCE_IC20

LL_RCC_MCO2SOURCE_SYSB

MCO prescaler

LL_RCC_MCO1_DIV_1

LL_RCC_MCO1_DIV_2

LL_RCC_MCO1_DIV_3

LL_RCC_MCO1_DIV_4

LL_RCC_MCO1_DIV_5

LL_RCC_MCO1_DIV_6

LL_RCC_MCO1_DIV_7

LL_RCC_MCO1_DIV_8

LL_RCC_MCO1_DIV_9

LL_RCC_MCO1_DIV_10

LL_RCC_MCO1_DIV_11

LL_RCC_MCO1_DIV_12

LL_RCC_MCO1_DIV_13

LL_RCC_MCO1_DIV_14

LL_RCC_MCO1_DIV_15

LL_RCC_MCO1_DIV_16

LL_RCC_MCO2_DIV_1

LL_RCC_MCO2_DIV_2

LL_RCC_MCO2_DIV_3

LL_RCC_MCO2_DIV_4

LL_RCC_MCO2_DIV_5

LL_RCC_MCO2_DIV_6

LL_RCC_MCO2_DIV_7

LL_RCC_MCO2_DIV_8

LL_RCC_MCO2_DIV_9

LL_RCC_MCO2_DIV_10

LL_RCC_MCO2_DIV_11

LL_RCC_MCO2_DIV_12

LL_RCC_MCO2_DIV_13

LL_RCC_MCO2_DIV_14

LL_RCC_MCO2_DIV_15

LL_RCC_MCO2_DIV_16

Peripheral MDF get clock source

LL_RCC_MDF1_CLKSOURCE

Peripheral MDF clock source selection

LL_RCC_MDF1_CLKSOURCE_HCLK

LL_RCC_MDF1_CLKSOURCE_CLKP

LL_RCC_MDF1_CLKSOURCE_IC7

LL_RCC_MDF1_CLKSOURCE_IC8

LL_RCC_MDF1_CLKSOURCE_MSI

LL_RCC_MDF1_CLKSOURCE_HSI

LL_RCC_MDF1_CLKSOURCE_I2S_CKIN

LL_RCC_MDF1_CLKSOURCE_TIMG

MSI oscillator frequency select

LL_RCC_MSI_FREQ_4MHZ

LL_RCC_MSI_FREQ_16MHZ

Peripheral OTGPHY get clock source

LL_RCC_OTGPHY1_CLKSOURCE

LL_RCC_OTGPHY2_CLKSOURCE

Peripheral OTGPHYCKREF get clock source

LL_RCC_OTGPHY1CKREF_CLKSOURCE

LL_RCC_OTGPHY2CKREF_CLKSOURCE

Peripheral OTGPHYCKREF clock source selection

LL_RCC_OTGPHY1CKREF_CLKSOURCE_OTGPHY1

LL_RCC_OTGPHY1CKREF_CLKSOURCE_HSE_DIV_2_OSC

LL_RCC_OTGPHY2CKREF_CLKSOURCE_OTGPHY2

LL_RCC_OTGPHY2CKREF_CLKSOURCE_HSE_DIV_2_OSC

Peripheral OTGPHY clock source selection

LL_RCC_OTGPHY1_CLKSOURCE_HSE_DIV_2

LL_RCC_OTGPHY1_CLKSOURCE_CLKP

LL_RCC_OTGPHY1_CLKSOURCE_IC15

LL_RCC_OTGPHY1_CLKSOURCE_HSE_DIV_2_OSC

LL_RCC_OTGPHY2_CLKSOURCE_HSE_DIV_2

LL_RCC_OTGPHY2_CLKSOURCE_CLKP

LL_RCC_OTGPHY2_CLKSOURCE_IC15

LL_RCC_OTGPHY2_CLKSOURCE_HSE_DIV_2_OSC

Peripheral clock frequency

LL_RCC_PERIPH_FREQUENCY_NO

No clock enabled for the peripheral

LL_RCC_PERIPH_FREQUENCY_NA

Frequency cannot be provided as external clock

All PLLs entry clock source

LL_RCC_PLLSOURCE_HSI

LL_RCC_PLLSOURCE_MSI

LL_RCC_PLLSOURCE_HSE

LL_RCC_PLLSOURCE_I2S_CKIN

Peripheral PSSI get clock source

LL_RCC_PSSI_CLKSOURCE

Peripheral PSSI clock source selection

LL_RCC_PSSI_CLKSOURCE_HCLK

LL_RCC_PSSI_CLKSOURCE_CLKP

LL_RCC_PSSI_CLKSOURCE_IC20

LL_RCC_PSSI_CLKSOURCE_HSI

RTC clock source selection

LL_RCC_RTC_CLKSOURCE_NONE

LL_RCC_RTC_CLKSOURCE_LSE

LL_RCC_RTC_CLKSOURCE_LSI

LL_RCC_RTC_CLKSOURCE_HSE

Prescaler for RTC clock

LL_RCC_RTC_HSE_DIV_1

LL_RCC_RTC_HSE_DIV_2

LL_RCC_RTC_HSE_DIV_3

LL_RCC_RTC_HSE_DIV_4

LL_RCC_RTC_HSE_DIV_5

LL_RCC_RTC_HSE_DIV_6

LL_RCC_RTC_HSE_DIV_7

LL_RCC_RTC_HSE_DIV_8

LL_RCC_RTC_HSE_DIV_9

LL_RCC_RTC_HSE_DIV_10

LL_RCC_RTC_HSE_DIV_11

LL_RCC_RTC_HSE_DIV_12

LL_RCC_RTC_HSE_DIV_13

LL_RCC_RTC_HSE_DIV_14

LL_RCC_RTC_HSE_DIV_15

LL_RCC_RTC_HSE_DIV_16

LL_RCC_RTC_HSE_DIV_17

LL_RCC_RTC_HSE_DIV_18

LL_RCC_RTC_HSE_DIV_19

LL_RCC_RTC_HSE_DIV_20

LL_RCC_RTC_HSE_DIV_21

LL_RCC_RTC_HSE_DIV_22

LL_RCC_RTC_HSE_DIV_23

LL_RCC_RTC_HSE_DIV_24

LL_RCC_RTC_HSE_DIV_25

LL_RCC_RTC_HSE_DIV_26

LL_RCC_RTC_HSE_DIV_27

LL_RCC_RTC_HSE_DIV_28

LL_RCC_RTC_HSE_DIV_29

LL_RCC_RTC_HSE_DIV_30

LL_RCC_RTC_HSE_DIV_31

LL_RCC_RTC_HSE_DIV_32

LL_RCC_RTC_HSE_DIV_33

LL_RCC_RTC_HSE_DIV_34

LL_RCC_RTC_HSE_DIV_35

LL_RCC_RTC_HSE_DIV_36

LL_RCC_RTC_HSE_DIV_37

LL_RCC_RTC_HSE_DIV_38

LL_RCC_RTC_HSE_DIV_39

LL_RCC_RTC_HSE_DIV_40

LL_RCC_RTC_HSE_DIV_41

LL_RCC_RTC_HSE_DIV_42

LL_RCC_RTC_HSE_DIV_43

LL_RCC_RTC_HSE_DIV_44

LL_RCC_RTC_HSE_DIV_45

LL_RCC_RTC_HSE_DIV_46

LL_RCC_RTC_HSE_DIV_47

LL_RCC_RTC_HSE_DIV_48

LL_RCC_RTC_HSE_DIV_49

LL_RCC_RTC_HSE_DIV_50

LL_RCC_RTC_HSE_DIV_51

LL_RCC_RTC_HSE_DIV_52

LL_RCC_RTC_HSE_DIV_53

LL_RCC_RTC_HSE_DIV_54

LL_RCC_RTC_HSE_DIV_55

LL_RCC_RTC_HSE_DIV_56

LL_RCC_RTC_HSE_DIV_57

LL_RCC_RTC_HSE_DIV_58

LL_RCC_RTC_HSE_DIV_59

LL_RCC_RTC_HSE_DIV_60

LL_RCC_RTC_HSE_DIV_61

LL_RCC_RTC_HSE_DIV_62

LL_RCC_RTC_HSE_DIV_63

LL_RCC_RTC_HSE_DIV_64

Peripheral SAI get clock source

LL_RCC_SAI1_CLKSOURCE

LL_RCC_SAI2_CLKSOURCE

Peripheral SAI clock source selection

LL_RCC_SAI1_CLKSOURCE_PCLK2

LL_RCC_SAI1_CLKSOURCE_CLKP

LL_RCC_SAI1_CLKSOURCE_IC7

LL_RCC_SAI1_CLKSOURCE_IC8

LL_RCC_SAI1_CLKSOURCE_MSI

LL_RCC_SAI1_CLKSOURCE_HSI

LL_RCC_SAI1_CLKSOURCE_I2S_CKIN

LL_RCC_SAI1_CLKSOURCE_SPDIFRX1

LL_RCC_SAI2_CLKSOURCE_PCLK2

LL_RCC_SAI2_CLKSOURCE_CLKP

LL_RCC_SAI2_CLKSOURCE_IC7

LL_RCC_SAI2_CLKSOURCE_IC8

LL_RCC_SAI2_CLKSOURCE_MSI

LL_RCC_SAI2_CLKSOURCE_HSI

LL_RCC_SAI2_CLKSOURCE_I2S_CKIN

LL_RCC_SAI2_CLKSOURCE_SPDIFRX1

Peripheral SDMMC get clock source

LL_RCC_SDMMC1_CLKSOURCE

LL_RCC_SDMMC2_CLKSOURCE

Peripheral SDMMC clock source selection

LL_RCC_SDMMC1_CLKSOURCE_HCLK

LL_RCC_SDMMC1_CLKSOURCE_CLKP

LL_RCC_SDMMC1_CLKSOURCE_IC4

LL_RCC_SDMMC1_CLKSOURCE_IC5

LL_RCC_SDMMC2_CLKSOURCE_HCLK

LL_RCC_SDMMC2_CLKSOURCE_CLKP

LL_RCC_SDMMC2_CLKSOURCE_IC4

LL_RCC_SDMMC2_CLKSOURCE_IC5

Peripheral SPDIFRX get clock source

LL_RCC_SPDIFRX1_CLKSOURCE

Peripheral SPDIFRX clock source selection

LL_RCC_SPDIFRX1_CLKSOURCE_PCLK1

LL_RCC_SPDIFRX1_CLKSOURCE_CLKP

LL_RCC_SPDIFRX1_CLKSOURCE_IC7

LL_RCC_SPDIFRX1_CLKSOURCE_IC8

LL_RCC_SPDIFRX1_CLKSOURCE_MSI

LL_RCC_SPDIFRX1_CLKSOURCE_HSI

LL_RCC_SPDIFRX1_CLKSOURCE_I2S_CKIN

Peripheral SPI get clock source

LL_RCC_SPI1_CLKSOURCE

LL_RCC_SPI2_CLKSOURCE

LL_RCC_SPI3_CLKSOURCE

LL_RCC_SPI4_CLKSOURCE

LL_RCC_SPI5_CLKSOURCE

LL_RCC_SPI6_CLKSOURCE

Peripheral SPI clock source selection

LL_RCC_SPI1_CLKSOURCE_PCLK2

LL_RCC_SPI1_CLKSOURCE_CLKP

LL_RCC_SPI1_CLKSOURCE_IC8

LL_RCC_SPI1_CLKSOURCE_IC9

LL_RCC_SPI1_CLKSOURCE_MSI

LL_RCC_SPI1_CLKSOURCE_HSI

LL_RCC_SPI1_CLKSOURCE_I2S_CKIN

LL_RCC_SPI2_CLKSOURCE_PCLK1

LL_RCC_SPI2_CLKSOURCE_CLKP

LL_RCC_SPI2_CLKSOURCE_IC8

LL_RCC_SPI2_CLKSOURCE_IC9

LL_RCC_SPI2_CLKSOURCE_MSI

LL_RCC_SPI2_CLKSOURCE_HSI

LL_RCC_SPI2_CLKSOURCE_I2S_CKIN

LL_RCC_SPI3_CLKSOURCE_PCLK1

LL_RCC_SPI3_CLKSOURCE_CLKP

LL_RCC_SPI3_CLKSOURCE_IC8

LL_RCC_SPI3_CLKSOURCE_IC9

LL_RCC_SPI3_CLKSOURCE_MSI

LL_RCC_SPI3_CLKSOURCE_HSI

LL_RCC_SPI3_CLKSOURCE_I2S_CKIN

LL_RCC_SPI4_CLKSOURCE_PCLK2

LL_RCC_SPI4_CLKSOURCE_CLKP

LL_RCC_SPI4_CLKSOURCE_IC9

LL_RCC_SPI4_CLKSOURCE_IC14

LL_RCC_SPI4_CLKSOURCE_MSI

LL_RCC_SPI4_CLKSOURCE_HSI

LL_RCC_SPI4_CLKSOURCE_HSE

LL_RCC_SPI5_CLKSOURCE_PCLK2

LL_RCC_SPI5_CLKSOURCE_CLKP

LL_RCC_SPI5_CLKSOURCE_IC9

LL_RCC_SPI5_CLKSOURCE_IC14

LL_RCC_SPI5_CLKSOURCE_MSI

LL_RCC_SPI5_CLKSOURCE_HSI

LL_RCC_SPI5_CLKSOURCE_HSE

LL_RCC_SPI6_CLKSOURCE_PCLK4

LL_RCC_SPI6_CLKSOURCE_CLKP

LL_RCC_SPI6_CLKSOURCE_IC8

LL_RCC_SPI6_CLKSOURCE_IC9

LL_RCC_SPI6_CLKSOURCE_MSI

LL_RCC_SPI6_CLKSOURCE_HSI

LL_RCC_SPI6_CLKSOURCE_I2S_CKIN

System wakeup clock source

LL_RCC_SYSWAKEUP_CLKSOURCE_HSI

LL_RCC_SYSWAKEUP_CLKSOURCE_MSI

System bus clock switch

LL_RCC_SYS_CLKSOURCE_HSI
: Select HSI as system bus clocks

LL_RCC_SYS_CLKSOURCE_MSI
: Select MSI as system bus clocks

LL_RCC_SYS_CLKSOURCE_HSE
: Select HSE as system bus clocks

LL_RCC_SYS_CLKSOURCE_IC2_IC6_IC11
: Select IC2/IC6/IC11 as system bus clocks

System bus clock switch status

LL_RCC_SYS_CLKSOURCE_STATUS_HSI
: HSI used as system bus clocks

LL_RCC_SYS_CLKSOURCE_STATUS_MSI
: MSI used as system bus clocks

LL_RCC_SYS_CLKSOURCE_STATUS_HSE
: HSE used as system bus clocks

LL_RCC_SYS_CLKSOURCE_STATUS_IC2_IC6_IC11
: IC2/IC6/IC11 used as system bus clocks

Timers clocks prescalers selection

LL_RCC_TIM_PRESCALER_1

LL_RCC_TIM_PRESCALER_2

LL_RCC_TIM_PRESCALER_4

LL_RCC_TIM_PRESCALER_8

Peripheral UART get clock source

LL_RCC_UART4_CLKSOURCE

LL_RCC_UART5_CLKSOURCE

LL_RCC_UART7_CLKSOURCE

LL_RCC_UART8_CLKSOURCE

LL_RCC_UART9_CLKSOURCE

Peripheral UART clock source selection

LL_RCC_UART4_CLKSOURCE_PCLK1

LL_RCC_UART4_CLKSOURCE_CLKP

LL_RCC_UART4_CLKSOURCE_IC9

LL_RCC_UART4_CLKSOURCE_IC14

LL_RCC_UART4_CLKSOURCE_LSE

LL_RCC_UART4_CLKSOURCE_MSI

LL_RCC_UART4_CLKSOURCE_HSI

LL_RCC_UART5_CLKSOURCE_PCLK1

LL_RCC_UART5_CLKSOURCE_CLKP

LL_RCC_UART5_CLKSOURCE_IC9

LL_RCC_UART5_CLKSOURCE_IC14

LL_RCC_UART5_CLKSOURCE_LSE

LL_RCC_UART5_CLKSOURCE_MSI

LL_RCC_UART5_CLKSOURCE_HSI

LL_RCC_UART7_CLKSOURCE_PCLK1

LL_RCC_UART7_CLKSOURCE_CLKP

LL_RCC_UART7_CLKSOURCE_IC9

LL_RCC_UART7_CLKSOURCE_IC14

LL_RCC_UART7_CLKSOURCE_LSE

LL_RCC_UART7_CLKSOURCE_MSI

LL_RCC_UART7_CLKSOURCE_HSI

LL_RCC_UART8_CLKSOURCE_PCLK1

LL_RCC_UART8_CLKSOURCE_CLKP

LL_RCC_UART8_CLKSOURCE_IC9

LL_RCC_UART8_CLKSOURCE_IC14

LL_RCC_UART8_CLKSOURCE_LSE

LL_RCC_UART8_CLKSOURCE_MSI

LL_RCC_UART8_CLKSOURCE_HSI

LL_RCC_UART9_CLKSOURCE_PCLK2

LL_RCC_UART9_CLKSOURCE_CLKP

LL_RCC_UART9_CLKSOURCE_IC9

LL_RCC_UART9_CLKSOURCE_IC14

LL_RCC_UART9_CLKSOURCE_LSE

LL_RCC_UART9_CLKSOURCE_MSI

LL_RCC_UART9_CLKSOURCE_HSI

Peripheral USART get clock source

LL_RCC_USART1_CLKSOURCE

LL_RCC_USART2_CLKSOURCE

LL_RCC_USART3_CLKSOURCE

LL_RCC_USART6_CLKSOURCE

LL_RCC_USART10_CLKSOURCE

Peripheral USART clock source selection

LL_RCC_USART1_CLKSOURCE_PCLK2

LL_RCC_USART1_CLKSOURCE_CLKP

LL_RCC_USART1_CLKSOURCE_IC9

LL_RCC_USART1_CLKSOURCE_IC14

LL_RCC_USART1_CLKSOURCE_LSE

LL_RCC_USART1_CLKSOURCE_MSI

LL_RCC_USART1_CLKSOURCE_HSI

LL_RCC_USART2_CLKSOURCE_PCLK1

LL_RCC_USART2_CLKSOURCE_CLKP

LL_RCC_USART2_CLKSOURCE_IC9

LL_RCC_USART2_CLKSOURCE_IC14

LL_RCC_USART2_CLKSOURCE_LSE

LL_RCC_USART2_CLKSOURCE_MSI

LL_RCC_USART2_CLKSOURCE_HSI

LL_RCC_USART3_CLKSOURCE_PCLK1

LL_RCC_USART3_CLKSOURCE_CLKP

LL_RCC_USART3_CLKSOURCE_IC9

LL_RCC_USART3_CLKSOURCE_IC14

LL_RCC_USART3_CLKSOURCE_LSE

LL_RCC_USART3_CLKSOURCE_MSI

LL_RCC_USART3_CLKSOURCE_HSI

LL_RCC_USART6_CLKSOURCE_PCLK2

LL_RCC_USART6_CLKSOURCE_CLKP

LL_RCC_USART6_CLKSOURCE_IC9

LL_RCC_USART6_CLKSOURCE_IC14

LL_RCC_USART6_CLKSOURCE_LSE

LL_RCC_USART6_CLKSOURCE_MSI

LL_RCC_USART6_CLKSOURCE_HSI

LL_RCC_USART10_CLKSOURCE_PCLK2

LL_RCC_USART10_CLKSOURCE_CLKP

LL_RCC_USART10_CLKSOURCE_IC9

LL_RCC_USART10_CLKSOURCE_IC14

LL_RCC_USART10_CLKSOURCE_LSE

LL_RCC_USART10_CLKSOURCE_MSI

LL_RCC_USART10_CLKSOURCE_HSI

Peripheral XSPI get clock source

LL_RCC_XSPI1_CLKSOURCE

LL_RCC_XSPI2_CLKSOURCE

LL_RCC_XSPI3_CLKSOURCE

Peripheral XSPI clock source selection

LL_RCC_XSPI1_CLKSOURCE_HCLK

LL_RCC_XSPI1_CLKSOURCE_CLKP

LL_RCC_XSPI1_CLKSOURCE_IC3

LL_RCC_XSPI1_CLKSOURCE_IC4

LL_RCC_XSPI2_CLKSOURCE_HCLK

LL_RCC_XSPI2_CLKSOURCE_CLKP

LL_RCC_XSPI2_CLKSOURCE_IC3

LL_RCC_XSPI2_CLKSOURCE_IC4

LL_RCC_XSPI3_CLKSOURCE_HCLK

LL_RCC_XSPI3_CLKSOURCE_CLKP

LL_RCC_XSPI3_CLKSOURCE_IC3

LL_RCC_XSPI3_CLKSOURCE_IC4

Calculate frequencies

LL_RCC_CALC_HCLK_FREQ

Description:

- Helper macro to calculate the HCLK frequency.

Parameters:

- __SYSCLKFREQ__: SYSCLK frequency.
- __HPRESCALER__: This parameter can be one of the following values:
  - LL_RCC_AHB_DIV_1
  - LL_RCC_AHB_DIV_2
  - LL_RCC_AHB_DIV_4
  - LL_RCC_AHB_DIV_8
  - LL_RCC_AHB_DIV_16
  - LL_RCC_AHB_DIV_32
  - LL_RCC_AHB_DIV_64
  - LL_RCC_AHB_DIV_128

Return value:

- HCLK: clock frequency (in Hz)

LL_RCC_CALC_PCLK1_FREQ

Description:

- Helper macro to calculate the PCLK1 frequency (ABP1)

Parameters:

- __HCLKFREQ__: HCLK frequency
- __APB1PRESCALER__: This parameter can be one of the following values:
  - LL_RCC_APB1_DIV_1
  - LL_RCC_APB1_DIV_2
  - LL_RCC_APB1_DIV_4
  - LL_RCC_APB1_DIV_8
  - LL_RCC_APB1_DIV_16
  - LL_RCC_APB1_DIV_32
  - LL_RCC_APB1_DIV_64
  - LL_RCC_APB1_DIV_128

Return value:

- PCLK1: clock frequency (in Hz)

LL_RCC_CALC_PCLK2_FREQ

Description:

- Helper macro to calculate the PCLK2 frequency (ABP2)

Parameters:

- __HCLKFREQ__: HCLK frequency
- __APB2PRESCALER__: This parameter can be one of the following values:
  - LL_RCC_APB2_DIV_1
  - LL_RCC_APB2_DIV_2
  - LL_RCC_APB2_DIV_4
  - LL_RCC_APB2_DIV_8
  - LL_RCC_APB2_DIV_16
  - LL_RCC_APB2_DIV_32
  - LL_RCC_APB2_DIV_64
  - LL_RCC_APB2_DIV_128

Return value:

- PCLK2: clock frequency (in Hz)

LL_RCC_CALC_PCLK4_FREQ

Description:

- Helper macro to calculate the PCLK4 frequency (ABP4)

Parameters:

- __HCLKFREQ__: HCLK frequency
- __APB4PRESCALER__: This parameter can be one of the following values:
  - LL_RCC_APB4_DIV_1
  - LL_RCC_APB4_DIV_2
  - LL_RCC_APB4_DIV_4
  - LL_RCC_APB4_DIV_8
  - LL_RCC_APB4_DIV_16
  - LL_RCC_APB4_DIV_32
  - LL_RCC_APB4_DIV_64
  - LL_RCC_APB4_DIV_128

Return value:

- PCLK1: clock frequency (in Hz)

LL_RCC_CALC_PCLK5_FREQ

Description:

- Helper macro to calculate the PCLK5 frequency (APB5)

Parameters:

- __HCLKFREQ__: HCLK frequency
- __APB5PRESCALER__: This parameter can be one of the following values:
  - LL_RCC_APB5_DIV_1
  - LL_RCC_APB5_DIV_2
  - LL_RCC_APB5_DIV_4
  - LL_RCC_APB5_DIV_8
  - LL_RCC_APB5_DIV_16
  - LL_RCC_APB5_DIV_32
  - LL_RCC_APB5_DIV_64
  - LL_RCC_APB5_DIV_128

Return value:

- PCLK1: clock frequency (in Hz)

Common Write and read registers Macros

LL_RCC_WriteReg

Description:

- Write a value in RCC register.

Parameters:

- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_RCC_ReadReg

Description:

- Read a value in RCC register.

Parameters:

- __REG__: Register to be read

Return value:

- Register: value

VENC_REG

LL_VENC_WriteRegister

Description:

- Write a value in VENC register.

Parameters:

- __REG__: Number of the register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_VENC_ReadRegister

Description:

- Read a value in VENC register.

Parameters:

- __REG__: Number of the register to be read

Return value:

- Register: value

102 LL RNG Generic Driver

102.1 RNG Firmware driver registers structures

102.1.1 LL_RNG_InitTypeDef

LL_RNG_InitTypeDef is defined in the stm32n6xx_ll_rng.h

Data Fields

- ***uint32_t ClockErrorDetection***

Field Documentation

- ***uint32_t LL_RNG_InitTypeDef::ClockErrorDetection***
  Clock error detection. This parameter can be one value of ***RNG_LL_CED***. This parameter can be modified using unitary functions **LL_RNG_EnableClkErrorDetect()**.

102.2 RNG Firmware driver API description

The following section lists the various functions of the RNG library.

102.2.1 Detailed description of functions

LL_RNG_Enable

Function name

__STATIC_INLINE void LL_RNG_Enable (RNG_TypeDef * RNGx)

Function description

Enable Random Number Generation.

Parameters

- **RNGx:** RNG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR RNGEN LL_RNG_Enable

LL_RNG_Disable

Function name

__STATIC_INLINE void LL_RNG_Disable (RNG_TypeDef * RNGx)

Function description

Disable Random Number Generation.

Parameters

- **RNGx:** RNG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR RNGEN LL_RNG_Disable

LL_RNG_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_RNG_IsEnabled (const RNG_TypeDef * RNGx)

Function description

Check if Random Number Generator is enabled.

Parameters

- **RNGx:** RNG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR RNGEN LL_RNG_IsEnabled

LL_RNG_EnableClkErrorDetect

Function name

__STATIC_INLINE void LL_RNG_EnableClkErrorDetect (RNG_TypeDef * RNGx)

Function description

Enable Clock Error Detection.

Parameters

- **RNGx:** RNG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CED LL_RNG_EnableClkErrorDetect

LL_RNG_DisableClkErrorDetect

Function name

__STATIC_INLINE void LL_RNG_DisableClkErrorDetect (RNG_TypeDef * RNGx)

Function description

Disable RNG Clock Error Detection.

Parameters

- **RNGx:** RNG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CED LL_RNG_DisableClkErrorDetect

LL_RNG_IsEnabledClkErrorDetect

Function name

__STATIC_INLINE uint32_t LL_RNG_IsEnabledClkErrorDetect (const RNG_TypeDef * RNGx)

Function description

Check if RNG Clock Error Detection is enabled.

Parameters

- **RNGx:** RNG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR CED LL_RNG_IsEnabledClkErrorDetect

LL_RNG_EnableCondReset

Function name

__STATIC_INLINE void LL_RNG_EnableCondReset (RNG_TypeDef * RNGx)

Function description

Set RNG Conditioning Soft Reset bit.

Parameters

- **RNGx:** RNG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CONDRST LL_RNG_EnableCondReset

LL_RNG_DisableCondReset

Function name

__STATIC_INLINE void LL_RNG_DisableCondReset (RNG_TypeDef * RNGx)

Function description

Reset RNG Conditioning Soft Reset bit.

Parameters

- **RNGx:** RNG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CONDRST LL_RNG_DisableCondReset

LL_RNG_IsEnabledCondReset

Function name

__STATIC_INLINE uint32_t LL_RNG_IsEnabledCondReset (const RNG_TypeDef * RNGx)

Function description

Check if RNG Conditioning Soft Reset bit is set.

Parameters

- **RNGx:** RNG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR CONDRST LL_RNG_IsEnabledCondReset

LL_RNG_ConfigLock

Function name

__STATIC_INLINE void LL_RNG_ConfigLock (RNG_TypeDef * RNGx)

Function description

Enable RNG Config Lock.

Parameters

- **RNGx:** RNG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CONFIGLOCK LL_RNG_ConfigLock

LL_RNG_IsConfigLocked

Function name

__STATIC_INLINE uint32_t LL_RNG_IsConfigLocked (const RNG_TypeDef * RNGx)

Function description

Check if RNG Config Lock is enabled.

Parameters

- **RNGx:** RNG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR CONFIGLOCK LL_RNG_IsConfigLocked

LL_RNG_EnableNistCompliance

Function name

__STATIC_INLINE void LL_RNG_EnableNistCompliance (RNG_TypeDef * RNGx)

Function description

Enable NIST Compliance.

Parameters

- **RNGx:** RNG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR NISTC LL_RNG_EnableNistCompliance

LL_RNG_DisableNistCompliance

Function name

__STATIC_INLINE void LL_RNG_DisableNistCompliance (RNG_TypeDef * RNGx)

Function description

Disable NIST Compliance.

Parameters

- **RNGx:** RNG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR NISTC LL_RNG_DisableNistCompliance

LL_RNG_IsEnabledNistCompliance

Function name

__STATIC_INLINE uint32_t LL_RNG_IsEnabledNistCompliance (const RNG_TypeDef * RNGx)

Function description

Check if NIST Compliance is enabled.

Parameters

- **RNGx:** RNG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR NISTC LL_RNG_IsEnabledNistCompliance

LL_RNG_SetConfig1

Function name

__STATIC_INLINE void LL_RNG_SetConfig1 (RNG_TypeDef * RNGx, uint32_t Config1)

Function description

Set RNG Config1 Configuration field value.

Parameters

- **RNGx:** RNG Instance
- **Config1:** Value between 0 and 0x3F

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR RNG_CONFIG1 LL_RNG_SetConfig1

LL_RNG_GetConfig1

Function name

__STATIC_INLINE uint32_t LL_RNG_GetConfig1 (const RNG_TypeDef * RNGx)

Function description

Get RNG Config1 Configuration field value.

Parameters

- **RNGx:** RNG Instance

Return values

- **Returned:** Value expressed on 6 bits : Value between 0 and 0x3F

Reference Manual to LL API cross reference:

- CR RNG_CONFIG1 LL_RNG_GetConfig1

LL_RNG_SetConfig2

Function name

__STATIC_INLINE void LL_RNG_SetConfig2 (RNG_TypeDef * RNGx, uint32_t Config2)

Function description

Set RNG Config2 Configuration field value.

Parameters

- **RNGx:** RNG Instance
- **Config2:** Value between 0 and 0x7

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR RNG_CONFIG2 LL_RNG_SetConfig2

LL_RNG_GetConfig2

Function name

__STATIC_INLINE uint32_t LL_RNG_GetConfig2 (const RNG_TypeDef * RNGx)

Function description

Get RNG Config2 Configuration field value.

Parameters

- **RNGx:** RNG Instance

Return values

- **Returned:** Value expressed on 3 bits : Value between 0 and 0x7

Reference Manual to LL API cross reference:

- CR RNG_CONFIG2 LL_RNG_GetConfig2

LL_RNG_SetConfig3

Function name

__STATIC_INLINE void LL_RNG_SetConfig3 (RNG_TypeDef * RNGx, uint32_t Config3)

Function description

Set RNG Config3 Configuration field value.

Parameters

- **RNGx:** RNG Instance
- **Config3:** Value between 0 and 0xF

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR RNG_CONFIG3 LL_RNG_SetConfig3

LL_RNG_GetConfig3

Function name

__STATIC_INLINE uint32_t LL_RNG_GetConfig3 (const RNG_TypeDef * RNGx)

Function description

Get RNG Config3 Configuration field value.

Parameters

- **RNGx:** RNG Instance

Return values

- **Returned:** Value expressed on 4 bits : Value between 0 and 0xF

Reference Manual to LL API cross reference:

- CR RNG_CONFIG3 LL_RNG_GetConfig3

LL_RNG_SetClockDivider

Function name

__STATIC_INLINE void LL_RNG_SetClockDivider (RNG_TypeDef * RNGx, uint32_t Divider)

Function description

Set RNG Clock divider factor.

Parameters

- **RNGx:** RNG Instance
- **Divider:** can be one of the following values:
  - LL_RNG_CLKDIV_BY_1
  - LL_RNG_CLKDIV_BY_2
  - LL_RNG_CLKDIV_BY_4
  - LL_RNG_CLKDIV_BY_8
  - LL_RNG_CLKDIV_BY_16
  - LL_RNG_CLKDIV_BY_32
  - LL_RNG_CLKDIV_BY_64
  - LL_RNG_CLKDIV_BY_128
  - LL_RNG_CLKDIV_BY_256
  - LL_RNG_CLKDIV_BY_512
  - LL_RNG_CLKDIV_BY_1024
  - LL_RNG_CLKDIV_BY_2048
  - LL_RNG_CLKDIV_BY_4096
  - LL_RNG_CLKDIV_BY_8192
  - LL_RNG_CLKDIV_BY_16384
  - LL_RNG_CLKDIV_BY_32768

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CLKDIV LL_RNG_SetClockDivider

LL_RNG_GetClockDivider

Function name

__STATIC_INLINE uint32_t LL_RNG_GetClockDivider (const RNG_TypeDef * RNGx)

Function description

Get RNG Clock divider factor.

Parameters

- **RNGx:** RNG Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RNG_CLKDIV_BY_1
  - LL_RNG_CLKDIV_BY_2
  - LL_RNG_CLKDIV_BY_4
  - LL_RNG_CLKDIV_BY_8
  - LL_RNG_CLKDIV_BY_16
  - LL_RNG_CLKDIV_BY_32
  - LL_RNG_CLKDIV_BY_64
  - LL_RNG_CLKDIV_BY_128
  - LL_RNG_CLKDIV_BY_256
  - LL_RNG_CLKDIV_BY_512
  - LL_RNG_CLKDIV_BY_1024
  - LL_RNG_CLKDIV_BY_2048
  - LL_RNG_CLKDIV_BY_4096
  - LL_RNG_CLKDIV_BY_8192
  - LL_RNG_CLKDIV_BY_16384
  - LL_RNG_CLKDIV_BY_32768

Reference Manual to LL API cross reference:

- CR CLKDIV LL_RNG_GetClockDivider

LL_RNG_IsActiveFlag_DRDY

Function name

__STATIC_INLINE uint32_t LL_RNG_IsActiveFlag_DRDY (const RNG_TypeDef * RNGx)

Function description

Indicate if the RNG Data ready Flag is set or not.

Parameters

- **RNGx:** RNG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR DRDY LL_RNG_IsActiveFlag_DRDY

LL_RNG_IsActiveFlag_CECS

Function name

__STATIC_INLINE uint32_t LL_RNG_IsActiveFlag_CECS (const RNG_TypeDef * RNGx)

Function description

Indicate if the Clock Error Current Status Flag is set or not.

Parameters

- **RNGx:** RNG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR CECS LL_RNG_IsActiveFlag_CECS

LL_RNG_IsActiveFlag_SECS

Function name

__STATIC_INLINE uint32_t LL_RNG_IsActiveFlag_SECS (const RNG_TypeDef * RNGx)

Function description

Indicate if the Seed Error Current Status Flag is set or not.

Parameters

- **RNGx:** RNG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR SECS LL_RNG_IsActiveFlag_SECS

LL_RNG_IsActiveFlag_CEIS

Function name

__STATIC_INLINE uint32_t LL_RNG_IsActiveFlag_CEIS (const RNG_TypeDef * RNGx)

Function description

Indicate if the Clock Error Interrupt Status Flag is set or not.

Parameters

- **RNGx:** RNG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR CEIS LL_RNG_IsActiveFlag_CEIS

LL_RNG_IsActiveFlag_SEIS

Function name

__STATIC_INLINE uint32_t LL_RNG_IsActiveFlag_SEIS (const RNG_TypeDef * RNGx)

Function description

Indicate if the Seed Error Interrupt Status Flag is set or not.

Parameters

- **RNGx:** RNG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR SEIS LL_RNG_IsActiveFlag_SEIS

LL_RNG_ClearFlag_CEIS

Function name

__STATIC_INLINE void LL_RNG_ClearFlag_CEIS (RNG_TypeDef * RNGx)

Function description

Clear Clock Error interrupt Status (CEIS) Flag.

Parameters

- **RNGx:** RNG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR CEIS LL_RNG_ClearFlag_CEIS

LL_RNG_ClearFlag_SEIS

Function name

__STATIC_INLINE void LL_RNG_ClearFlag_SEIS (RNG_TypeDef * RNGx)

Function description

Clear Seed Error interrupt Status (SEIS) Flag.

Parameters

- **RNGx:** RNG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR SEIS LL_RNG_ClearFlag_SEIS

LL_RNG_EnableIT

Function name

__STATIC_INLINE void LL_RNG_EnableIT (RNG_TypeDef * RNGx)

Function description

Enable Random Number Generator Interrupt (applies for either Seed error, Clock Error or Data ready interrupts)

Parameters

- **RNGx:** RNG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR IE LL_RNG_EnableIT

LL_RNG_DisableIT

Function name

__STATIC_INLINE void LL_RNG_DisableIT (RNG_TypeDef * RNGx)

Function description

Disable Random Number Generator Interrupt (applies for either Seed error, Clock Error or Data ready interrupts)

Parameters

- **RNGx:** RNG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR IE LL_RNG_DisableIT

LL_RNG_IsEnabledIT

Function name

__STATIC_INLINE uint32_t LL_RNG_IsEnabledIT (const RNG_TypeDef * RNGx)

Function description

Check if Random Number Generator Interrupt is enabled (applies for either Seed error, Clock Error or Data ready interrupts)

Parameters

- **RNGx:** RNG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR IE LL_RNG_IsEnabledIT

LL_RNG_ReadRandData32

Function name

__STATIC_INLINE uint32_t LL_RNG_ReadRandData32 (const RNG_TypeDef * RNGx)

Function description

Return32-bit Random Number value.

Parameters

- **RNGx:** RNG Instance

Return values

- **Generated:** 32-bit random value

Reference Manual to LL API cross reference:

- DR RNDATA LL_RNG_ReadRandData32

LL_RNG_EnableArdis

Function name

__STATIC_INLINE void LL_RNG_EnableArdis (RNG_TypeDef * RNGx)

Function description

Enable Auto reset.

Parameters

- **RNGx:** RNG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR ARDIS LL_RNG_EnableArdis

LL_RNG_DisableArdis

Function name

__STATIC_INLINE void LL_RNG_DisableArdis (RNG_TypeDef * RNGx)

Function description

Disable Auto reset.

Parameters

- **RNGx:** RNG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR ARDIS LL_RNG_DisableArdis

LL_RNG_IsEnabledArdis

Function name

__STATIC_INLINE uint32_t LL_RNG_IsEnabledArdis (const RNG_TypeDef * RNGx)

Function description

Check if RNG Auto reset is enabled.

Parameters

- **RNGx:** RNG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR ARDIS LL_RNG_IsEnabledArdis

LL_RNG_SetHealthConfig

Function name

__STATIC_INLINE void LL_RNG_SetHealthConfig (RNG_TypeDef * RNGx, uint32_t HTCFG)

Function description

Set RNG Health Test Control.

Parameters

- **RNGx:** RNG Instance
- **HTCFG:** can be values of 32 bits

Return values

- **None:**

Reference Manual to LL API cross reference:

- HTCR HTCFG LL_RNG_SetHealthConfig

LL_RNG_GetHealthConfig

Function name

__STATIC_INLINE uint32_t LL_RNG_GetHealthConfig (const RNG_TypeDef * RNGx)

Function description

Get RNG Health Test Control.

Parameters

- **RNGx:** RNG Instance

Return values

- **Return:** 32-bit RNG Health Test configuration

Reference Manual to LL API cross reference:

- HTCR HTCFG LL_RNG_GetHealthConfig

LL_RNG_Init

Function name

ErrorStatus LL_RNG_Init (RNG_TypeDef * RNGx, const LL_RNG_InitTypeDef * RNG_InitStruct)

Function description

Initialize RNG registers according to the specified parameters in RNG_InitStruct.

Parameters

- **RNGx:** RNG Instance
- **RNG_InitStruct:** pointer to a LL_RNG_InitTypeDef structure that contains the configuration information for the specified RNG peripheral.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: RNG registers are initialized according to RNG_InitStruct content
  - ERROR: not applicable

LL_RNG_StructInit

Function name

void LL_RNG_StructInit (LL_RNG_InitTypeDef * RNG_InitStruct)

Function description

Set each LL_RNG_InitTypeDef field to default value.

Parameters

- **RNG_InitStruct:** pointer to a LL_RNG_InitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

LL_RNG_DeInit

Function name

ErrorStatus LL_RNG_DeInit (const RNG_TypeDef * RNGx)

Function description

De-initialize RNG registers (Registers restored to their default values).

Parameters

- **RNGx:** RNG Instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: RNG registers are de-initialized
  - ERROR: not applicable

102.3 RNG Firmware driver defines

The following section lists the various define and macros of the module.

102.3.1 RNG

RNG

Auto reset disable

LL_RNG_ARDIS_ENABLE

ARDIS enabled automatic reset to clear SECS bit

LL_RNG_ARDIS_DISABLE

ARDIS disabled no automatic reset to clear SECS bit

Clock Error Detection

LL_RNG_CED_ENABLE

Clock error detection enabled

LL_RNG_CED_DISABLE

Clock error detection disabled

Value used to configure an internal

LL_RNG_CLKDIV_BY_1

No clock division

LL_RNG_CLKDIV_BY_2

2 RNG clock cycles per internal RNG clock

LL_RNG_CLKDIV_BY_4

4 RNG clock cycles per internal RNG clock

LL_RNG_CLKDIV_BY_8

8 RNG clock cycles per internal RNG clock

LL_RNG_CLKDIV_BY_16

16 RNG clock cycles per internal RNG clock

LL_RNG_CLKDIV_BY_32

32 RNG clock cycles per internal RNG clock

LL_RNG_CLKDIV_BY_64

64 RNG clock cycles per internal RNG clock

LL_RNG_CLKDIV_BY_128

128 RNG clock cycles per internal RNG clock

LL_RNG_CLKDIV_BY_256

256 RNG clock cycles per internal RNG clock

LL_RNG_CLKDIV_BY_512

512 RNG clock cycles per internal RNG clock

LL_RNG_CLKDIV_BY_1024

1024 RNG clock cycles per internal RNG clock

LL_RNG_CLKDIV_BY_2048

2048 RNG clock cycles per internal RNG clock

LL_RNG_CLKDIV_BY_4096

4096 RNG clock cycles per internal RNG clock

LL_RNG_CLKDIV_BY_8192

8192 RNG clock cycles per internal RNG clock

LL_RNG_CLKDIV_BY_16384

16384 RNG clock cycles per internal RNG clock

LL_RNG_CLKDIV_BY_32768

32768 RNG clock cycles per internal RNG clock

Get Flags Defines

LL_RNG_SR_DRDY

Register contains valid random data

LL_RNG_SR_CECS

Clock error current status

LL_RNG_SR_SECS

Seed error current status

LL_RNG_SR_CEIS

Clock error interrupt status

LL_RNG_SR_SEIS

Seed error interrupt status

IT Defines

LL_RNG_CR_IE

RNG Interrupt enable

Common Write and read registers Macros

LL_RNG_WriteReg

Description:

- Write a value in RNG register.

Parameters:

- __INSTANCE__: RNG Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_RNG_ReadReg

Description:

- Read a value in RNG register.

Parameters:

- __INSTANCE__: RNG Instance
- __REG__: Register to be read

Return value:

- Register: value

NIST Compliance configuration

LL_RNG_NIST_COMPLIANT

Default NIST compliant configuration

LL_RNG_CUSTOM_NIST

Custom NIST configuration

103 LL RTC Generic Driver

103.1 RTC Firmware driver registers structures

103.1.1 LL_RTC_InitTypeDef

LL_RTC_InitTypeDef is defined in the stm32n6xx_ll_rtc.h

Data Fields

- ***uint32_t HourFormat***
- ***uint32_t AsynchPrescaler***
- ***uint32_t SynchPrescaler***

Field Documentation

- ***uint32_t LL_RTC_InitTypeDef::HourFormat***
  Specifies the RTC Hours Format. This parameter can be a value of ***RTC_LL_EC_HOURFORMAT***This feature can be modified afterwards using unitary function **LL_RTC_SetHourFormat()**.
- ***uint32_t LL_RTC_InitTypeDef::AsynchPrescaler***
  Specifies the RTC Asynchronous Predivider value. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0x7FThis feature can be modified afterwards using unitary function **LL_RTC_SetAsynchPrescaler()**.
- ***uint32_t LL_RTC_InitTypeDef::SynchPrescaler***
  Specifies the RTC Synchronous Predivider value. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0x7FFFThis feature can be modified afterwards using unitary function **LL_RTC_SetSynchPrescaler()**.

103.1.2 LL_RTC_TimeTypeDef

LL_RTC_TimeTypeDef is defined in the stm32n6xx_ll_rtc.h

Data Fields

- ***uint32_t TimeFormat***
- ***uint8_t Hours***
- ***uint8_t Minutes***
- ***uint8_t Seconds***

Field Documentation

- ***uint32_t LL_RTC_TimeTypeDef::TimeFormat***
  Specifies the RTC AM/PM Time. This parameter can be a value of ***RTC_LL_EC_TIME_FORMAT***This feature can be modified afterwards using unitary function **LL_RTC_TIME_SetFormat()**.
- ***uint8_t LL_RTC_TimeTypeDef::Hours***
  Specifies the RTC Time Hours. This parameter must be a number between Min_Data = 0 and Max_Data = 12 if the **LL_RTC_TIME_FORMAT_PM** is selected.This parameter must be a number between Min_Data = 0 and Max_Data = 23 if the **LL_RTC_TIME_FORMAT_AM_OR_24** is selected.This feature can be modified afterwards using unitary function **LL_RTC_TIME_SetHour()**.
- ***uint8_t LL_RTC_TimeTypeDef::Minutes***
  Specifies the RTC Time Minutes. This parameter must be a number between Min_Data = 0 and Max_Data = 59This feature can be modified afterwards using unitary function **LL_RTC_TIME_SetMinute()**.
- ***uint8_t LL_RTC_TimeTypeDef::Seconds***
  Specifies the RTC Time Seconds. This parameter must be a number between Min_Data = 0 and Max_Data = 59This feature can be modified afterwards using unitary function **LL_RTC_TIME_SetSecond()**.

103.1.3 LL_RTC_DateTypeDef

LL_RTC_DateTypeDef is defined in the stm32n6xx_ll_rtc.h

Data Fields

- ***uint8_t WeekDay***
- ***uint8_t Month***
- ***uint8_t Day***

- ***uint8_t Year***

Field Documentation

- ***uint8_t LL_RTC_DateTypeDef::WeekDay***
  Specifies the RTC Date WeekDay. This parameter can be a value of ***RTC_LL_EC_WEEKDAY***This feature can be modified afterwards using unitary function **LL_RTC_DATE_SetWeekDay()**.
- ***uint8_t LL_RTC_DateTypeDef::Month***
  Specifies the RTC Date Month. This parameter can be a value of ***RTC_LL_EC_MONTH***This feature can be modified afterwards using unitary function **LL_RTC_DATE_SetMonth()**.
- ***uint8_t LL_RTC_DateTypeDef::Day***
  Specifies the RTC Date Day. This parameter must be a number between Min_Data = 1 and Max_Data = 31This feature can be modified afterwards using unitary function **LL_RTC_DATE_SetDay()**.
- ***uint8_t LL_RTC_DateTypeDef::Year***
  Specifies the RTC Date Year. This parameter must be a number between Min_Data = 0 and Max_Data = 99This feature can be modified afterwards using unitary function **LL_RTC_DATE_SetYear()**.

103.1.4 LL_RTC_AlarmTypeDef

LL_RTC_AlarmTypeDef is defined in the stm32n6xx_ll_rtc.h

Data Fields

- ***LL_RTC_TimeTypeDef AlarmTime***
- ***uint32_t AlarmMask***
- ***uint32_t AlarmDateWeekDaySel***
- ***uint8_t AlarmDateWeekDay***

Field Documentation

- ***LL_RTC_TimeTypeDef LL_RTC_AlarmTypeDef::AlarmTime***
  Specifies the RTC Alarm Time members.
- ***uint32_t LL_RTC_AlarmTypeDef::AlarmMask***
  Specifies the RTC Alarm Masks. This parameter can be a value of ***RTC_LL_EC_ALMA_MASK*** for ALARM A or ***RTC_LL_EC_ALMB_MASK*** for ALARM B.This feature can be modified afterwards using unitary function **LL_RTC_ALMA_SetMask()** for ALARM A or **LL_RTC_ALMB_SetMask()** for ALARM B.
- ***uint32_t LL_RTC_AlarmTypeDef::AlarmDateWeekDaySel***
  Specifies the RTC Alarm is on day or WeekDay. This parameter can be a value of ***RTC_LL_EC_ALMA_WEEKDAY_SELECTION*** for ALARM A or ***RTC_LL_EC_ALMB_WEEKDAY_SELECTION*** for ALARM B.This feature can be modified afterwards using unitary function **LL_RTC_ALMA_EnableWeekday()** or **LL_RTC_ALMA_DisableWeekday()** for ALARM A or **LL_RTC_ALMB_EnableWeekday()** or **LL_RTC_ALMB_DisableWeekday()** for ALARM B.
- ***uint8_t LL_RTC_AlarmTypeDef::AlarmDateWeekDay***
  Specifies the RTC Alarm Day/WeekDay. If AlarmDateWeekDaySel set to day, this parameter must be a number between Min_Data = 1 and Max_Data = 31.This feature can be modified afterwards using unitary function **LL_RTC_ALMA_SetDay()** for ALARM A or **LL_RTC_ALMB_SetDay()** for ALARM B.If AlarmDateWeekDaySel set to Weekday, this parameter can be a value of ***RTC_LL_EC_WEEKDAY***.This feature can be modified afterwards using unitary function **LL_RTC_ALMA_SetWeekDay()** for ALARM A or **LL_RTC_ALMB_SetWeekDay()** for ALARM B.

103.2 RTC Firmware driver API description

The following section lists the various functions of the RTC library.

103.2.1 Detailed description of functions

LL_RTC_SetHourFormat

Function name

__STATIC_INLINE void LL_RTC_SetHourFormat (RTC_TypeDef * RTCx, uint32_t HourFormat)

Function description

Set Hours format (24 hour/day or AM/PM hour format)

Parameters

- **RTCx:** RTC Instance
- **HourFormat:** This parameter can be one of the following values:
  - LL_RTC_HOURFORMAT_24HOUR
  - LL_RTC_HOURFORMAT_AMPM

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- It can be written in initialization mode only (LL_RTC_EnableInitMode function)

Reference Manual to LL API cross reference:

- RTC_CR FMT LL_RTC_SetHourFormat

LL_RTC_GetHourFormat

Function name

__STATIC_INLINE uint32_t LL_RTC_GetHourFormat (const RTC_TypeDef * RTCx)

Function description

Get Hours format (24 hour/day or AM/PM hour format)

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_HOURFORMAT_24HOUR
  - LL_RTC_HOURFORMAT_AMPM

Reference Manual to LL API cross reference:

- RTC_CR FMT LL_RTC_GetHourFormat

LL_RTC_SetAlarmOutEvent

Function name

__STATIC_INLINE void LL_RTC_SetAlarmOutEvent (RTC_TypeDef * RTCx, uint32_t AlarmOutput)

Function description

Select the flag to be routed to RTC_ALARM output.

Parameters

- **RTCx:** RTC Instance
- **AlarmOutput:** This parameter can be one of the following values:
  - LL_RTC_ALARMOUT_DISABLE
  - LL_RTC_ALARMOUT_ALMA
  - LL_RTC_ALARMOUT_ALMB
  - LL_RTC_ALARMOUT_WAKEUP

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR OSEL LL_RTC_SetAlarmOutEvent

LL_RTC_GetAlarmOutEvent

Function name

__STATIC_INLINE uint32_t LL_RTC_GetAlarmOutEvent (const RTC_TypeDef * RTCx)

Function description

Get the flag to be routed to RTC_ALARM output.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_ALARMOUT_DISABLE
  - LL_RTC_ALARMOUT_ALMA
  - LL_RTC_ALARMOUT_ALMB
  - LL_RTC_ALARMOUT_WAKEUP

Reference Manual to LL API cross reference:

- RTC_CR OSEL LL_RTC_GetAlarmOutEvent

LL_RTC_SetAlarmOutputType

Function name

__STATIC_INLINE void LL_RTC_SetAlarmOutputType (RTC_TypeDef * RTCx, uint32_t Output)

Function description

Set RTC_ALARM output type (ALARM in push-pull or open-drain output)

Parameters

- **RTCx:** RTC Instance
- **Output:** This parameter can be one of the following values:
  - LL_RTC_ALARM_OUTPUTTYPE_OPENDRAIN
  - LL_RTC_ALARM_OUTPUTTYPE_PUSHPULL

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_CR TAMPALRM_TYPE LL_RTC_SetAlarmOutputType

LL_RTC_GetAlarmOutputType

Function name

__STATIC_INLINE uint32_t LL_RTC_GetAlarmOutputType (const RTC_TypeDef * RTCx)

Function description

Get RTC_ALARM output type (ALARM in push-pull or open-drain output)

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_ALARM_OUTPUTTYPE_OPENDRAIN
  - LL_RTC_ALARM_OUTPUTTYPE_PUSHPULL

Reference Manual to LL API cross reference:

- RTC_CR TAMPALRM_TYPE LL_RTC_GetAlarmOutputType

LL_RTC_EnableInitMode

Function name

__STATIC_INLINE void LL_RTC_EnableInitMode (RTC_TypeDef * RTCx)

Function description

Enable initialization mode.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Initialization mode is used to program time and date register (RTC_TR and RTC_DR) and prescaler register (RTC_PRER). Counters are stopped and start counting from the new value when INIT is reset.

Reference Manual to LL API cross reference:

- RTC_ICSR INIT LL_RTC_EnableInitMode

LL_RTC_DisableInitMode

Function name

__STATIC_INLINE void LL_RTC_DisableInitMode (RTC_TypeDef * RTCx)

Function description

Disable initialization mode (Free running mode)

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_ICSR INIT LL_RTC_DisableInitMode

LL_RTC_SetBinaryMode

Function name

__STATIC_INLINE void LL_RTC_SetBinaryMode (RTC_TypeDef * RTCx, uint32_t BinaryMode)

Function description

Set Binary mode (Sub Second Register)

Parameters

- **RTCx:** RTC Instance
- **BinaryMode:** can be one of the following values:
  - LL_RTC_BINARY_NONE
  - LL_RTC_BINARY_ONLY
  - LL_RTC_BINARY_MIX

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- It can be written in initialization mode only (LL_RTC_EnableInitMode function).

Reference Manual to LL API cross reference:

- RTC_ICSR BIN LL_RTC_SetBinaryMode

LL_RTC_GetBinaryMode

Function name

__STATIC_INLINE uint32_t LL_RTC_GetBinaryMode (const RTC_TypeDef * RTCx)

Function description

Get Binary mode (Sub Second Register)

Parameters

- **RTCx:** RTC Instance

Return values

- **This:** parameter can be one of the following values:
  - LL_RTC_BINARY_NONE
  - LL_RTC_BINARY_ONLY
  - LL_RTC_BINARY_MIX
- **None:**

Reference Manual to LL API cross reference:

- RTC_ICSR BIN LL_RTC_GetBinaryMode

LL_RTC_SetBinMixBCDU

Function name

__STATIC_INLINE void LL_RTC_SetBinMixBCDU (RTC_TypeDef * RTCx, uint32_t BinMixBcdU)

Function description

Set Binary Mix mode BCDU.

Parameters

- **RTCx:** RTC Instance
- **BinMixBcdU:** can be one of the following values:
  - LL_RTC_BINARY_MIX_BCDU_0
  - LL_RTC_BINARY_MIX_BCDU_1
  - LL_RTC_BINARY_MIX_BCDU_2
  - LL_RTC_BINARY_MIX_BCDU_3
  - LL_RTC_BINARY_MIX_BCDU_4
  - LL_RTC_BINARY_MIX_BCDU_5
  - LL_RTC_BINARY_MIX_BCDU_6
  - LL_RTC_BINARY_MIX_BCDU_7

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- It can be written in initialization mode only (LL_RTC_EnableInitMode function).

Reference Manual to LL API cross reference:

- RTC_ICSR BCDU LL_RTC_SetBinMixBCDU

LL_RTC_GetBinMixBCDU

Function name

__STATIC_INLINE uint32_t LL_RTC_GetBinMixBCDU (const RTC_TypeDef * RTCx)

Function description

Get Binary Mix mode BCDU.

Parameters

- **RTCx:** RTC Instance

Return values

- **This:** parameter can be one of the following values:
  - LL_RTC_BINARY_MIX_BCDU_0
  - LL_RTC_BINARY_MIX_BCDU_1
  - LL_RTC_BINARY_MIX_BCDU_2
  - LL_RTC_BINARY_MIX_BCDU_3
  - LL_RTC_BINARY_MIX_BCDU_4
  - LL_RTC_BINARY_MIX_BCDU_5
  - LL_RTC_BINARY_MIX_BCDU_6
  - LL_RTC_BINARY_MIX_BCDU_7
- **None:**

Reference Manual to LL API cross reference:

- RTC_ICSR BCDU LL_RTC_GetBinMixBCDU

LL_RTC_SetOutputPolarity

Function name

__STATIC_INLINE void LL_RTC_SetOutputPolarity (RTC_TypeDef * RTCx, uint32_t Polarity)

Function description

Set Output polarity (pin is low when ALRAF/ALRBF/WUTF is asserted)

Parameters

- **RTCx:** RTC Instance
- **Polarity:** This parameter can be one of the following values:
  - LL_RTC_OUTPUTPOLARITY_PIN_HIGH
  - LL_RTC_OUTPUTPOLARITY_PIN_LOW

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR POL LL_RTC_SetOutputPolarity

LL_RTC_GetOutputPolarity

Function name

__STATIC_INLINE uint32_t LL_RTC_GetOutputPolarity (const RTC_TypeDef * RTCx)

Function description

Get Output polarity.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_OUTPUTPOLARITY_PIN_HIGH
  - LL_RTC_OUTPUTPOLARITY_PIN_LOW

Reference Manual to LL API cross reference:

- RTC_CR POL LL_RTC_GetOutputPolarity

LL_RTC_EnableShadowRegBypass

Function name

__STATIC_INLINE void LL_RTC_EnableShadowRegBypass (RTC_TypeDef * RTCx)

Function description

Enable Bypass the shadow registers.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR BYPSHAD LL_RTC_EnableShadowRegBypass

LL_RTC_DisableShadowRegBypass

Function name

__STATIC_INLINE void LL_RTC_DisableShadowRegBypass (RTC_TypeDef * RTCx)

Function description

Disable Bypass the shadow registers.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_CR BYPSHAD LL_RTC_DisableShadowRegBypass

LL_RTC_IsShadowRegBypassEnabled

Function name

__STATIC_INLINE uint32_t LL_RTC_IsShadowRegBypassEnabled (const RTC_TypeDef * RTCx)

Function description

Check if Shadow registers bypass is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_CR BYPSHAD LL_RTC_IsShadowRegBypassEnabled

LL_RTC_EnableRefClock

Function name

__STATIC_INLINE void LL_RTC_EnableRefClock (RTC_TypeDef * RTCx)

Function description

Enable RTC_REFIN reference clock detection (50 or 60 Hz)

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- It can be written in initialization mode only (LL_RTC_EnableInitMode function)

Reference Manual to LL API cross reference:

- RTC_CR REFCKON LL_RTC_EnableRefClock

LL_RTC_DisableRefClock

Function name

__STATIC_INLINE void LL_RTC_DisableRefClock (RTC_TypeDef * RTCx)

Function description

Disable RTC_REFIN reference clock detection (50 or 60 Hz)

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- It can be written in initialization mode only (LL_RTC_EnableInitMode function)

Reference Manual to LL API cross reference:

- RTC_CR REFCKON LL_RTC_DisableRefClock

LL_RTC_SetAsynchPrescaler

Function name

__STATIC_INLINE void LL_RTC_SetAsynchPrescaler (RTC_TypeDef * RTCx, uint32_t AsynchPrescaler)

Function description

Set Asynchronous prescaler factor.

Parameters

- **RTCx:** RTC Instance
- **AsynchPrescaler:** Value between Min_Data = 0 and Max_Data = 0x7F

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_PRER PREDIV_A LL_RTC_SetAsynchPrescaler

LL_RTC_SetSynchPrescaler

Function name

__STATIC_INLINE void LL_RTC_SetSynchPrescaler (RTC_TypeDef * RTCx, uint32_t SynchPrescaler)

Function description

Set Synchronous prescaler factor.

Parameters

- **RTCx:** RTC Instance
- **SynchPrescaler:** Value between Min_Data = 0 and Max_Data = 0x7FFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_PRER PREDIV_S LL_RTC_SetSynchPrescaler

LL_RTC_GetAsynchPrescaler

Function name

__STATIC_INLINE uint32_t LL_RTC_GetAsynchPrescaler (const RTC_TypeDef * RTCx)

Function description

Get Asynchronous prescaler factor.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data = 0 and Max_Data = 0x7F

Reference Manual to LL API cross reference:

- RTC_PRER PREDIV_A LL_RTC_GetAsynchPrescaler

LL_RTC_GetSynchPrescaler

Function name

__STATIC_INLINE uint32_t LL_RTC_GetSynchPrescaler (const RTC_TypeDef * RTCx)

Function description

Get Synchronous prescaler factor.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data = 0 and Max_Data = 0x7FFF

Reference Manual to LL API cross reference:

- RTC_PRER PREDIV_S LL_RTC_GetSynchPrescaler

LL_RTC_EnableWriteProtection

Function name

__STATIC_INLINE void LL_RTC_EnableWriteProtection (RTC_TypeDef * RTCx)

Function description

Enable the write protection for RTC registers.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_WPR KEY LL_RTC_EnableWriteProtection

LL_RTC_DisableWriteProtection

Function name

__STATIC_INLINE void LL_RTC_DisableWriteProtection (RTC_TypeDef * RTCx)

Function description

Disable the write protection for RTC registers.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_WPR KEY LL_RTC_DisableWriteProtection

LL_RTC_EnableTamperOutput

Function name

__STATIC_INLINE void LL_RTC_EnableTamperOutput (RTC_TypeDef * RTCx)

Function description

Enable tamper output.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- When the tamper output is enabled, all external and internal tamper flags are ORed and routed to the TAMPALRM output.

Reference Manual to LL API cross reference:

- RTC_CR TAMPOE LL_RTC_EnableTamperOutput

LL_RTC_DisableTamperOutput

Function name

__STATIC_INLINE void LL_RTC_DisableTamperOutput (RTC_TypeDef * RTCx)

Function description

Disable tamper output.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_CR TAMPOE LL_RTC_DisableTamperOutput

LL_RTC_IsTamperOutputEnabled

Function name

__STATIC_INLINE uint32_t LL_RTC_IsTamperOutputEnabled (const RTC_TypeDef * RTCx)

Function description

Check if tamper output is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_CR TAMPOE LL_RTC_IsTamperOutputEnabled

LL_RTC_EnableAlarmPullUp

Function name

__STATIC_INLINE void LL_RTC_EnableAlarmPullUp (RTC_TypeDef * RTCx)

Function description

Enable internal pull-up in output mode.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_CR TAMPALRM_PU LL_RTC_EnableAlarmPullUp

LL_RTC_DisableAlarmPullUp

Function name

__STATIC_INLINE void LL_RTC_DisableAlarmPullUp (RTC_TypeDef * RTCx)

Function description

Disable internal pull-up in output mode.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_CR TAMPALRM_PU LL_RTC_EnableAlarmPullUp

LL_RTC_IsAlarmPullUpEnabled

Function name

__STATIC_INLINE uint32_t LL_RTC_IsAlarmPullUpEnabled (const RTC_TypeDef * RTCx)

Function description

Check if internal pull-up in output mode is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_CR TAMPALRM_PU LL_RTC_IsAlarmPullUpEnabled

LL_RTC_EnableOutput2

Function name

__STATIC_INLINE void LL_RTC_EnableOutput2 (RTC_TypeDef * RTCx)

Function description

Enable RTC_OUT2 output.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- RTC_OUT2 mapping depends on both OSEL (LL_RTC_SetAlarmOutEvent) and COE (LL_RTC_CAL_SetOutputFreq) settings.
- RTC_OUT2 is not available in VBAT mode.

Reference Manual to LL API cross reference:

- RTC_CR OUT2EN LL_RTC_EnableOutput2

LL_RTC_DisableOutput2

Function name

__STATIC_INLINE void LL_RTC_DisableOutput2 (RTC_TypeDef * RTCx)

Function description

Disable RTC_OUT2 output.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_CR OUT2EN LL_RTC_DisableOutput2

LL_RTC_IsOutput2Enabled

Function name

__STATIC_INLINE uint32_t LL_RTC_IsOutput2Enabled (const RTC_TypeDef * RTCx)

Function description

Check if RTC_OUT2 output is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_CR OUT2EN LL_RTC_IsOutput2Enabled

LL_RTC_TIME_SetFormat

Function name

__STATIC_INLINE void LL_RTC_TIME_SetFormat (RTC_TypeDef * RTCx, uint32_t TimeFormat)

Function description

Set time format (AM/24-hour or PM notation)

Parameters

- **RTCx:** RTC Instance
- **TimeFormat:** This parameter can be one of the following values:
  - LL_RTC_TIME_FORMAT_AM_OR_24
  - LL_RTC_TIME_FORMAT_PM

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- It can be written in initialization mode only (LL_RTC_EnableInitMode function)

Reference Manual to LL API cross reference:

- RTC_TR PM LL_RTC_TIME_SetFormat

LL_RTC_TIME_GetFormat

Function name

__STATIC_INLINE uint32_t LL_RTC_TIME_GetFormat (const RTC_TypeDef * RTCx)

Function description

Get time format (AM or PM notation)

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_TIME_FORMAT_AM_OR_24
  - LL_RTC_TIME_FORMAT_PM

Notes

- if RTC shadow registers are not bypassed (BYPSHAD=0), need to check if RSF flag is set before reading this bit
- Read either RTC_SSR or RTC_TR locks the values in the higher-order calendar shadow registers until RTC_DR is read (LL_RTC_ReadReg(RTC, DR)).

Reference Manual to LL API cross reference:

- RTC_TR PM LL_RTC_TIME_GetFormat

LL_RTC_TIME_SetHour

Function name

__STATIC_INLINE void LL_RTC_TIME_SetHour (RTC_TypeDef * RTCx, uint32_t Hours)

Function description

Set Hours in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Hours:** Value between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- It can be written in initialization mode only (LL_RTC_EnableInitMode function)
- helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert hour from binary to BCD format

Reference Manual to LL API cross reference:

- RTC_TR HT LL_RTC_TIME_SetHour
- RTC_TR HU LL_RTC_TIME_SetHour

LL_RTC_TIME_GetHour

Function name

__STATIC_INLINE uint32_t LL_RTC_TIME_GetHour (const RTC_TypeDef * RTCx)

Function description

Get Hours in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23

Notes

- if RTC shadow registers are not bypassed (BYPSHAD=0), need to check if RSF flag is set before reading this bit
- Read either RTC_SSR or RTC_TR locks the values in the higher-order calendar shadow registers until RTC_DR is read (LL_RTC_ReadReg(RTC, DR)).
- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert hour from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_TR HT LL_RTC_TIME_GetHour
- RTC_TR HU LL_RTC_TIME_GetHour

LL_RTC_TIME_SetMinute

Function name

__STATIC_INLINE void LL_RTC_TIME_SetMinute (RTC_TypeDef * RTCx, uint32_t Minutes)

Function description

Set Minutes in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Minutes:** Value between Min_Data=0x00 and Max_Data=0x59

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- It can be written in initialization mode only (LL_RTC_EnableInitMode function)
- helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Minutes from binary to BCD format

Reference Manual to LL API cross reference:

- RTC_TR MNT LL_RTC_TIME_SetMinute
- RTC_TR MNU LL_RTC_TIME_SetMinute

LL_RTC_TIME_GetMinute

Function name

__STATIC_INLINE uint32_t LL_RTC_TIME_GetMinute (const RTC_TypeDef * RTCx)

Function description

Get Minutes in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x59

Notes

- if RTC shadow registers are not bypassed (BYPSHAD=0), need to check if RSF flag is set before reading this bit
- Read either RTC_SSR or RTC_TR locks the values in the higher-order calendar shadow registers until RTC_DR is read (LL_RTC_ReadReg(RTC, DR)).
- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert minute from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_TR MNT LL_RTC_TIME_GetMinute
- RTC_TR MNU LL_RTC_TIME_GetMinute

LL_RTC_TIME_SetSecond

Function name

__STATIC_INLINE void LL_RTC_TIME_SetSecond (RTC_TypeDef * RTCx, uint32_t Seconds)

Function description

Set Seconds in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Seconds:** Value between Min_Data=0x00 and Max_Data=0x59

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- It can be written in initialization mode only (LL_RTC_EnableInitMode function)
- helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Seconds from binary to BCD format

Reference Manual to LL API cross reference:

- RTC_TR ST LL_RTC_TIME_SetSecond
- RTC_TR SU LL_RTC_TIME_SetSecond

LL_RTC_TIME_GetSecond

Function name

__STATIC_INLINE uint32_t LL_RTC_TIME_GetSecond (const RTC_TypeDef * RTCx)

Function description

Get Seconds in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x59

Notes

- if RTC shadow registers are not bypassed (BYPSHAD=0), need to check if RSF flag is set before reading this bit
- Read either RTC_SSR or RTC_TR locks the values in the higher-order calendar shadow registers until RTC_DR is read (LL_RTC_ReadReg(RTC, DR)).
- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Seconds from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_TR ST LL_RTC_TIME_GetSecond
- RTC_TR SU LL_RTC_TIME_GetSecond

LL_RTC_TIME_Config

Function name

__STATIC_INLINE void LL_RTC_TIME_Config (RTC_TypeDef * RTCx, uint32_t Format12_24, uint32_t Hours, uint32_t Minutes, uint32_t Seconds)

Function description

Set time (hour, minute and second) in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Format12_24:** This parameter can be one of the following values:
  - LL_RTC_TIME_FORMAT_AM_OR_24
  - LL_RTC_TIME_FORMAT_PM
- **Hours:** Value between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23
- **Minutes:** Value between Min_Data=0x00 and Max_Data=0x59
- **Seconds:** Value between Min_Data=0x00 and Max_Data=0x59

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- It can be written in initialization mode only (LL_RTC_EnableInitMode function)
- TimeFormat and Hours should follow the same format

Reference Manual to LL API cross reference:

- RTC_TR PM LL_RTC_TIME_Config
- RTC_TR HT LL_RTC_TIME_Config
- RTC_TR HU LL_RTC_TIME_Config
- RTC_TR MNT LL_RTC_TIME_Config
- RTC_TR MNU LL_RTC_TIME_Config
- RTC_TR ST LL_RTC_TIME_Config
- RTC_TR SU LL_RTC_TIME_Config

LL_RTC_TIME_Get

Function name

__STATIC_INLINE uint32_t LL_RTC_TIME_Get (const RTC_TypeDef * RTCx)

Function description

Get time (hour, minute and second) in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Combination:** of hours, minutes and seconds (Format: 0x00HHMMSS).

Notes

- if RTC shadow registers are not bypassed (BYPSHAD=0), need to check if RSF flag is set before reading this bit
- Read either RTC_SSR or RTC_TR locks the values in the higher-order calendar shadow registers until RTC_DR is read (LL_RTC_ReadReg(RTC, DR)).
- helper macros __LL_RTC_GET_HOUR, __LL_RTC_GET_MINUTE and __LL_RTC_GET_SECOND are available to get independently each parameter.

Reference Manual to LL API cross reference:

- RTC_TR HT LL_RTC_TIME_Get
- RTC_TR HU LL_RTC_TIME_Get
- RTC_TR MNT LL_RTC_TIME_Get
- RTC_TR MNU LL_RTC_TIME_Get
- RTC_TR ST LL_RTC_TIME_Get
- RTC_TR SU LL_RTC_TIME_Get

LL_RTC_TIME_EnableDayLightStore

Function name

__STATIC_INLINE void LL_RTC_TIME_EnableDayLightStore (RTC_TypeDef * RTCx)

Function description

Memorize whether the daylight saving time change has been performed.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR BKP LL_RTC_TIME_EnableDayLightStore

LL_RTC_TIME_DisableDayLightStore

Function name

__STATIC_INLINE void LL_RTC_TIME_DisableDayLightStore (RTC_TypeDef * RTCx)

Function description

Disable memorization whether the daylight saving time change has been performed.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR BKP LL_RTC_TIME_DisableDayLightStore

LL_RTC_TIME_IsDayLightStoreEnabled

Function name

__STATIC_INLINE uint32_t LL_RTC_TIME_IsDayLightStoreEnabled (const RTC_TypeDef * RTCx)

Function description

Check if RTC Day Light Saving stored operation has been enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_CR BKP LL_RTC_TIME_IsDayLightStoreEnabled

LL_RTC_TIME_DecHour

Function name

__STATIC_INLINE void LL_RTC_TIME_DecHour (RTC_TypeDef * RTCx)

Function description

Subtract 1 hour (winter time change)

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR SUB1H LL_RTC_TIME_DecHour

LL_RTC_TIME_IncHour

Function name

__STATIC_INLINE void LL_RTC_TIME_IncHour (RTC_TypeDef * RTCx)

Function description

Add 1 hour (summer time change)

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR ADD1H LL_RTC_TIME_IncHour

LL_RTC_TIME_GetSubSecond

Function name

__STATIC_INLINE uint32_t LL_RTC_TIME_GetSubSecond (const RTC_TypeDef * RTCx)

Function description

Get Sub second value in the synchronous prescaler counter.

Parameters

- **RTCx:** RTC Instance

Return values

- **If:** binary mode is none, Value between Min_Data=0x0 and Max_Data=0x7FFF else Value between Min_Data=0x0 and Max_Data=0xFFFFFFFF

Notes

- You can use both SubSeconds value and SecondFraction (PREDIV_S through LL_RTC_GetSynchPrescaler function) terms returned to convert Calendar SubSeconds value in second fraction ratio with time unit following generic formula: ==> Seconds fraction ratio * time_unit= [(SecondFraction-SubSeconds)/(SecondFraction+1)] * time_unit This conversion can be performed only if no shift operation is pending (ie. SHFP=0) when PREDIV_S >= SS.

Reference Manual to LL API cross reference:

- RTC_SSR SS LL_RTC_TIME_GetSubSecond

LL_RTC_TIME_Synchronize

Function name

__STATIC_INLINE void LL_RTC_TIME_Synchronize (RTC_TypeDef * RTCx, uint32_t ShiftSecond, uint32_t Fraction)

Function description

Synchronize to a remote clock with a high degree of precision.

Parameters

- **RTCx:** RTC Instance
- **ShiftSecond:** This parameter can be one of the following values:
  - LL_RTC_SHIFT_SECOND_DELAY
  - LL_RTC_SHIFT_SECOND_ADVANCE
- **Fraction:** Number of Seconds Fractions (any value from 0 to 0x7FFF)

Return values

- **None:**

Notes

- This operation effectively subtracts from (delays) or advance the clock of a fraction of a second.
- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- When REFCKON is set, firmware must not write to Shift control register.

Reference Manual to LL API cross reference:

- RTC_SHIFTR ADD1S LL_RTC_TIME_Synchronize
- RTC_SHIFTR SUBFS LL_RTC_TIME_Synchronize

LL_RTC_DATE_SetYear

Function name

__STATIC_INLINE void LL_RTC_DATE_SetYear (RTC_TypeDef * RTCx, uint32_t Year)

Function description

Set Year in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Year:** Value between Min_Data=0x00 and Max_Data=0x99

Return values

- **None:**

Notes

- helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Year from binary to BCD format

Reference Manual to LL API cross reference:

- RTC_DR YT LL_RTC_DATE_SetYear
- RTC_DR YU LL_RTC_DATE_SetYear

LL_RTC_DATE_GetYear

Function name

__STATIC_INLINE uint32_t LL_RTC_DATE_GetYear (const RTC_TypeDef * RTCx)

Function description

Get Year in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x99

Notes

- if RTC shadow registers are not bypassed (BYPSHAD=0), need to check if RSF flag is set before reading this bit
- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Year from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_DR YT LL_RTC_DATE_GetYear
- RTC_DR YU LL_RTC_DATE_GetYear

LL_RTC_DATE_SetWeekDay

Function name

__STATIC_INLINE void LL_RTC_DATE_SetWeekDay (RTC_TypeDef * RTCx, uint32_t WeekDay)

Function description

Set Week day.

Parameters

- **RTCx:** RTC Instance
- **WeekDay:** This parameter can be one of the following values:
  - LL_RTC_WEEKDAY_MONDAY
  - LL_RTC_WEEKDAY_TUESDAY
  - LL_RTC_WEEKDAY_WEDNESDAY
  - LL_RTC_WEEKDAY_THURSDAY
  - LL_RTC_WEEKDAY_FRIDAY
  - LL_RTC_WEEKDAY_SATURDAY
  - LL_RTC_WEEKDAY_SUNDAY

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_DR WDU LL_RTC_DATE_SetWeekDay

LL_RTC_DATE_GetWeekDay

Function name

__STATIC_INLINE uint32_t LL_RTC_DATE_GetWeekDay (const RTC_TypeDef * RTCx)

Function description

Get Week day.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_WEEKDAY_MONDAY
  - LL_RTC_WEEKDAY_TUESDAY
  - LL_RTC_WEEKDAY_WEDNESDAY
  - LL_RTC_WEEKDAY_THURSDAY
  - LL_RTC_WEEKDAY_FRIDAY
  - LL_RTC_WEEKDAY_SATURDAY
  - LL_RTC_WEEKDAY_SUNDAY

Notes

- if RTC shadow registers are not bypassed (BYPSHAD=0), need to check if RSF flag is set before reading this bit

Reference Manual to LL API cross reference:

- RTC_DR WDU LL_RTC_DATE_GetWeekDay

LL_RTC_DATE_SetMonth

Function name

__STATIC_INLINE void LL_RTC_DATE_SetMonth (RTC_TypeDef * RTCx, uint32_t Month)

Function description

Set Month in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Month:** This parameter can be one of the following values:
  - LL_RTC_MONTH_JANUARY
  - LL_RTC_MONTH_FEBRUARY
  - LL_RTC_MONTH_MARCH
  - LL_RTC_MONTH_APRIL
  - LL_RTC_MONTH_MAY
  - LL_RTC_MONTH_JUNE
  - LL_RTC_MONTH_JULY
  - LL_RTC_MONTH_AUGUST
  - LL_RTC_MONTH_SEPTEMBER
  - LL_RTC_MONTH_OCTOBER
  - LL_RTC_MONTH_NOVEMBER
  - LL_RTC_MONTH_DECEMBER

Return values

- **None:**

Notes

- helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Month from binary to BCD format

Reference Manual to LL API cross reference:

- RTC_DR MT LL_RTC_DATE_SetMonth
- RTC_DR MU LL_RTC_DATE_SetMonth

LL_RTC_DATE_GetMonth

Function name

__STATIC_INLINE uint32_t LL_RTC_DATE_GetMonth (const RTC_TypeDef * RTCx)

Function description

Get Month in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_MONTH_JANUARY
  - LL_RTC_MONTH_FEBRUARY
  - LL_RTC_MONTH_MARCH
  - LL_RTC_MONTH_APRIL
  - LL_RTC_MONTH_MAY
  - LL_RTC_MONTH_JUNE
  - LL_RTC_MONTH_JULY
  - LL_RTC_MONTH_AUGUST
  - LL_RTC_MONTH_SEPTEMBER
  - LL_RTC_MONTH_OCTOBER
  - LL_RTC_MONTH_NOVEMBER
  - LL_RTC_MONTH_DECEMBER

Notes

- if RTC shadow registers are not bypassed (BYPSHAD=0), need to check if RSF flag is set before reading this bit
- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Month from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_DR MT LL_RTC_DATE_GetMonth
- RTC_DR MU LL_RTC_DATE_GetMonth

LL_RTC_DATE_SetDay

Function name

__STATIC_INLINE void LL_RTC_DATE_SetDay (RTC_TypeDef * RTCx, uint32_t Day)

Function description

Set Day in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Day:** Value between Min_Data=0x01 and Max_Data=0x31

Return values

- **None:**

Notes

- helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Day from binary to BCD format

Reference Manual to LL API cross reference:

- RTC_DR DT LL_RTC_DATE_SetDay
- RTC_DR DU LL_RTC_DATE_SetDay

LL_RTC_DATE_GetDay

Function name

__STATIC_INLINE uint32_t LL_RTC_DATE_GetDay (const RTC_TypeDef * RTCx)

Function description

Get Day in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x01 and Max_Data=0x31

Notes

- if RTC shadow registers are not bypassed (BYPSHAD=0), need to check if RSF flag is set before reading this bit
- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Day from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_DR DT LL_RTC_DATE_GetDay
- RTC_DR DU LL_RTC_DATE_GetDay

LL_RTC_DATE_Config

Function name

__STATIC_INLINE void LL_RTC_DATE_Config (RTC_TypeDef * RTCx, uint32_t WeekDay, uint32_t Day, uint32_t Month, uint32_t Year)

Function description

Set date (WeekDay, Day, Month and Year) in BCD format.

Parameters

- **RTCx:** RTC Instance
- **WeekDay:** This parameter can be one of the following values:
  - LL_RTC_WEEKDAY_MONDAY
  - LL_RTC_WEEKDAY_TUESDAY
  - LL_RTC_WEEKDAY_WEDNESDAY
  - LL_RTC_WEEKDAY_THURSDAY
  - LL_RTC_WEEKDAY_FRIDAY
  - LL_RTC_WEEKDAY_SATURDAY
  - LL_RTC_WEEKDAY_SUNDAY
- **Day:** Value between Min_Data=0x01 and Max_Data=0x31
- **Month:** This parameter can be one of the following values:
  - LL_RTC_MONTH_JANUARY
  - LL_RTC_MONTH_FEBRUARY
  - LL_RTC_MONTH_MARCH
  - LL_RTC_MONTH_APRIL
  - LL_RTC_MONTH_MAY
  - LL_RTC_MONTH_JUNE
  - LL_RTC_MONTH_JULY
  - LL_RTC_MONTH_AUGUST
  - LL_RTC_MONTH_SEPTEMBER
  - LL_RTC_MONTH_OCTOBER
  - LL_RTC_MONTH_NOVEMBER
  - LL_RTC_MONTH_DECEMBER
- **Year:** Value between Min_Data=0x00 and Max_Data=0x99

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_DR WDU LL_RTC_DATE_Config
- RTC_DR MT LL_RTC_DATE_Config
- RTC_DR MU LL_RTC_DATE_Config
- RTC_DR DT LL_RTC_DATE_Config
- RTC_DR DU LL_RTC_DATE_Config
- RTC_DR YT LL_RTC_DATE_Config
- RTC_DR YU LL_RTC_DATE_Config

LL_RTC_DATE_Get

Function name

__STATIC_INLINE uint32_t LL_RTC_DATE_Get (const RTC_TypeDef * RTCx)

Function description

Get date (WeekDay, Day, Month and Year) in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Combination:** of WeekDay, Day, Month and Year (Format: 0xWWDDMMYY).

Notes

- if RTC shadow registers are not bypassed (BYPSHAD=0), need to check if RSF flag is set before reading this bit
- helper macros __LL_RTC_GET_WEEKDAY, __LL_RTC_GET_YEAR, __LL_RTC_GET_MONTH, and __LL_RTC_GET_DAY are available to get independently each parameter.

Reference Manual to LL API cross reference:

- RTC_DR WDU LL_RTC_DATE_Get
- RTC_DR MT LL_RTC_DATE_Get
- RTC_DR MU LL_RTC_DATE_Get
- RTC_DR DT LL_RTC_DATE_Get
- RTC_DR DU LL_RTC_DATE_Get
- RTC_DR YT LL_RTC_DATE_Get
- RTC_DR YU LL_RTC_DATE_Get

LL_RTC_ALMA_Enable

Function name

__STATIC_INLINE void LL_RTC_ALMA_Enable (RTC_TypeDef * RTCx)

Function description

Enable Alarm A.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR ALRAE LL_RTC_ALMA_Enable

LL_RTC_ALMA_Disable

Function name

__STATIC_INLINE void LL_RTC_ALMA_Disable (RTC_TypeDef * RTCx)

Function description

Disable Alarm A.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR ALRAE LL_RTC_ALMA_Disable

LL_RTC_ALMA_SetMask

Function name

__STATIC_INLINE void LL_RTC_ALMA_SetMask (RTC_TypeDef * RTCx, uint32_t Mask)

Function description

Specify the Alarm A masks.

Parameters

- **RTCx:** RTC Instance
- **Mask:** This parameter can be a combination of the following values:
  - LL_RTC_ALMA_MASK_NONE
  - LL_RTC_ALMA_MASK_DATEWEEKDAY
  - LL_RTC_ALMA_MASK_HOURS
  - LL_RTC_ALMA_MASK_MINUTES
  - LL_RTC_ALMA_MASK_SECONDS
  - LL_RTC_ALMA_MASK_ALL

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_ALRMAR MSK4 LL_RTC_ALMA_SetMask
- RTC_ALRMAR MSK3 LL_RTC_ALMA_SetMask
- RTC_ALRMAR MSK2 LL_RTC_ALMA_SetMask
- RTC_ALRMAR MSK1 LL_RTC_ALMA_SetMask

LL_RTC_ALMA_GetMask

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMA_GetMask (const RTC_TypeDef * RTCx)

Function description

Get the Alarm A masks.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be can be a combination of the following values:
  - LL_RTC_ALMA_MASK_NONE
  - LL_RTC_ALMA_MASK_DATEWEEKDAY
  - LL_RTC_ALMA_MASK_HOURS
  - LL_RTC_ALMA_MASK_MINUTES
  - LL_RTC_ALMA_MASK_SECONDS
  - LL_RTC_ALMA_MASK_ALL

Reference Manual to LL API cross reference:

- RTC_ALRMAR MSK4 LL_RTC_ALMA_GetMask
- RTC_ALRMAR MSK3 LL_RTC_ALMA_GetMask
- RTC_ALRMAR MSK2 LL_RTC_ALMA_GetMask
- RTC_ALRMAR MSK1 LL_RTC_ALMA_GetMask

LL_RTC_ALMA_EnableWeekday

Function name

__STATIC_INLINE void LL_RTC_ALMA_EnableWeekday (RTC_TypeDef * RTCx)

Function description

Enable AlarmA Week day selection (DU[3:0] represents the week day.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_ALRMAR WDSEL LL_RTC_ALMA_EnableWeekday

LL_RTC_ALMA_DisableWeekday

Function name

__STATIC_INLINE void LL_RTC_ALMA_DisableWeekday (RTC_TypeDef * RTCx)

Function description

Disable AlarmA Week day selection (DU[3:0] represents the date)

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_ALRMAR WDSEL LL_RTC_ALMA_DisableWeekday

LL_RTC_ALMA_SetDay

Function name

__STATIC_INLINE void LL_RTC_ALMA_SetDay (RTC_TypeDef * RTCx, uint32_t Day)

Function description

Set ALARM A Day in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Day:** Value between Min_Data=0x01 and Max_Data=0x31

Return values

- **None:**

Notes

- helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Day from binary to BCD format

Reference Manual to LL API cross reference:

- RTC_ALRMAR DT LL_RTC_ALMA_SetDay
- RTC_ALRMAR DU LL_RTC_ALMA_SetDay

LL_RTC_ALMA_GetDay

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMA_GetDay (const RTC_TypeDef * RTCx)

Function description

Get ALARM A Day in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x01 and Max_Data=0x31

Notes

- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Day from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_ALRMAR DT LL_RTC_ALMA_GetDay
- RTC_ALRMAR DU LL_RTC_ALMA_GetDay

LL_RTC_ALMA_SetWeekDay

Function name

__STATIC_INLINE void LL_RTC_ALMA_SetWeekDay (RTC_TypeDef * RTCx, uint32_t WeekDay)

Function description

Set ALARM A Weekday.

Parameters

- **RTCx:** RTC Instance
- **WeekDay:** This parameter can be one of the following values:
  - LL_RTC_WEEKDAY_MONDAY
  - LL_RTC_WEEKDAY_TUESDAY
  - LL_RTC_WEEKDAY_WEDNESDAY
  - LL_RTC_WEEKDAY_THURSDAY
  - LL_RTC_WEEKDAY_FRIDAY
  - LL_RTC_WEEKDAY_SATURDAY
  - LL_RTC_WEEKDAY_SUNDAY

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_ALRMAR DU LL_RTC_ALMA_SetWeekDay

LL_RTC_ALMA_GetWeekDay

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMA_GetWeekDay (const RTC_TypeDef * RTCx)

Function description

Get ALARM A Weekday.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_WEEKDAY_MONDAY
  - LL_RTC_WEEKDAY_TUESDAY
  - LL_RTC_WEEKDAY_WEDNESDAY
  - LL_RTC_WEEKDAY_THURSDAY
  - LL_RTC_WEEKDAY_FRIDAY
  - LL_RTC_WEEKDAY_SATURDAY
  - LL_RTC_WEEKDAY_SUNDAY

Reference Manual to LL API cross reference:

- RTC_ALRMAR DU LL_RTC_ALMA_GetWeekDay

LL_RTC_ALMA_SetTimeFormat

Function name

__STATIC_INLINE void LL_RTC_ALMA_SetTimeFormat (RTC_TypeDef * RTCx, uint32_t TimeFormat)

Function description

Set Alarm A time format (AM/24-hour or PM notation)

Parameters

- **RTCx:** RTC Instance
- **TimeFormat:** This parameter can be one of the following values:
  - LL_RTC_ALMA_TIME_FORMAT_AM
  - LL_RTC_ALMA_TIME_FORMAT_PM

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_ALRMAR PM LL_RTC_ALMA_SetTimeFormat

LL_RTC_ALMA_GetTimeFormat

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMA_GetTimeFormat (const RTC_TypeDef * RTCx)

Function description

Get Alarm A time format (AM or PM notation)

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_ALMA_TIME_FORMAT_AM
  - LL_RTC_ALMA_TIME_FORMAT_PM

Reference Manual to LL API cross reference:

- RTC_ALRMAR PM LL_RTC_ALMA_GetTimeFormat

LL_RTC_ALMA_SetHour

Function name

__STATIC_INLINE void LL_RTC_ALMA_SetHour (RTC_TypeDef * RTCx, uint32_t Hours)

Function description

Set ALARM A Hours in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Hours:** Value between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23

Return values

- **None:**

Notes

- helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Hours from binary to BCD format

Reference Manual to LL API cross reference:

- RTC_ALRMAR HT LL_RTC_ALMA_SetHour
- RTC_ALRMAR HU LL_RTC_ALMA_SetHour

LL_RTC_ALMA_GetHour

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMA_GetHour (const RTC_TypeDef * RTCx)

Function description

Get ALARM A Hours in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23

Notes

- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Hours from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_ALRMAR HT LL_RTC_ALMA_GetHour
- RTC_ALRMAR HU LL_RTC_ALMA_GetHour

LL_RTC_ALMA_SetMinute

Function name

__STATIC_INLINE void LL_RTC_ALMA_SetMinute (RTC_TypeDef * RTCx, uint32_t Minutes)

Function description

Set ALARM A Minutes in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Minutes:** Value between Min_Data=0x00 and Max_Data=0x59

Return values

- **None:**

Notes

- helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Minutes from binary to BCD format

Reference Manual to LL API cross reference:

- RTC_ALRMAR MNT LL_RTC_ALMA_SetMinute
- RTC_ALRMAR MNU LL_RTC_ALMA_SetMinute

LL_RTC_ALMA_GetMinute

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMA_GetMinute (const RTC_TypeDef * RTCx)

Function description

Get ALARM A Minutes in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x59

Notes

- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Minutes from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_ALRMAR MNT LL_RTC_ALMA_GetMinute
- RTC_ALRMAR MNU LL_RTC_ALMA_GetMinute

LL_RTC_ALMA_SetSecond

Function name

__STATIC_INLINE void LL_RTC_ALMA_SetSecond (RTC_TypeDef * RTCx, uint32_t Seconds)

Function description

Set ALARM A Seconds in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Seconds:** Value between Min_Data=0x00 and Max_Data=0x59

Return values

- **None:**

Notes

- helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Seconds from binary to BCD format

Reference Manual to LL API cross reference:

- RTC_ALRMAR ST LL_RTC_ALMA_SetSecond
- RTC_ALRMAR SU LL_RTC_ALMA_SetSecond

LL_RTC_ALMA_GetSecond

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMA_GetSecond (const RTC_TypeDef * RTCx)

Function description

Get ALARM A Seconds in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x59

Notes

- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Seconds from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_ALRMAR ST LL_RTC_ALMA_GetSecond
- RTC_ALRMAR SU LL_RTC_ALMA_GetSecond

LL_RTC_ALMA_ConfigTime

Function name

__STATIC_INLINE void LL_RTC_ALMA_ConfigTime (RTC_TypeDef * RTCx, uint32_t Format12_24, uint32_t Hours, uint32_t Minutes, uint32_t Seconds)

Function description

Set Alarm A Time (hour, minute and second) in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Format12_24:** This parameter can be one of the following values:
  - LL_RTC_ALMA_TIME_FORMAT_AM
  - LL_RTC_ALMA_TIME_FORMAT_PM
- **Hours:** Value between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23
- **Minutes:** Value between Min_Data=0x00 and Max_Data=0x59
- **Seconds:** Value between Min_Data=0x00 and Max_Data=0x59

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_ALRMAR PM LL_RTC_ALMA_ConfigTime
- RTC_ALRMAR HT LL_RTC_ALMA_ConfigTime
- RTC_ALRMAR HU LL_RTC_ALMA_ConfigTime
- RTC_ALRMAR MNT LL_RTC_ALMA_ConfigTime
- RTC_ALRMAR MNU LL_RTC_ALMA_ConfigTime
- RTC_ALRMAR ST LL_RTC_ALMA_ConfigTime
- RTC_ALRMAR SU LL_RTC_ALMA_ConfigTime

LL_RTC_ALMA_GetTime

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMA_GetTime (const RTC_TypeDef * RTCx)

Function description

Get Alarm B Time (hour, minute and second) in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Combination:** of hours, minutes and seconds.

Notes

- helper macros __LL_RTC_GET_HOUR, __LL_RTC_GET_MINUTE and __LL_RTC_GET_SECOND are available to get independently each parameter.

Reference Manual to LL API cross reference:

- RTC_ALRMAR HT LL_RTC_ALMA_GetTime
- RTC_ALRMAR HU LL_RTC_ALMA_GetTime
- RTC_ALRMAR MNT LL_RTC_ALMA_GetTime
- RTC_ALRMAR MNU LL_RTC_ALMA_GetTime
- RTC_ALRMAR ST LL_RTC_ALMA_GetTime
- RTC_ALRMAR SU LL_RTC_ALMA_GetTime

LL_RTC_ALMA_SetSubSecondMask

Function name

__STATIC_INLINE void LL_RTC_ALMA_SetSubSecondMask (RTC_TypeDef * RTCx, uint32_t Mask)

Function description

Set Alarm A Mask the most-significant bits starting at this bit.

Parameters

- **RTCx:** RTC Instance
- **Mask:** If binary mode is none, Value between Min_Data=0x0 and Max_Data=0xF else Value between Min_Data=0x0 and Max_Data=0x3F

Return values

- **None:**

Notes

- This register can be written only when ALRAE is reset in RTC_CR register, or in initialization mode.

Reference Manual to LL API cross reference:

- RTC_ALRMASSR MASKSS LL_RTC_ALMA_SetSubSecondMask

LL_RTC_ALMA_GetSubSecondMask

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMA_GetSubSecondMask (const RTC_TypeDef * RTCx)

Function description

Get Alarm A Mask the most-significant bits starting at this bit.

Parameters

- **RTCx:** RTC Instance

Return values

- **If:** binary mode is none, Value between Min_Data=0x0 and Max_Data=0xF else Value between Min_Data=0x0 and Max_Data=0x3F

Reference Manual to LL API cross reference:

- RTC_ALRMASSR MASKSS LL_RTC_ALMA_GetSubSecondMask

LL_RTC_ALMA_SetBinAutoClr

Function name

__STATIC_INLINE void LL_RTC_ALMA_SetBinAutoClr (RTC_TypeDef * RTCx, uint32_t BinaryAutoClr)

Function description

Set Alarm A Binary mode auto clear.

Parameters

- **RTCx:** RTC Instance
- **BinaryAutoClr:** This parameter can be one of the following values:
  - LL_RTC_ALMA_SUBSECONDBIN_AUTOCLR_NO
  - LL_RTC_ALMA_SUBSECONDBIN_AUTOCLR_YES

Return values

- **None:**

Notes

- This register can be written only when ALRAE is reset in RTC_CR register, or in initialization mode.

Reference Manual to LL API cross reference:

- RTC_ALRABINR SSCLR LL_RTC_ALMA_SetBinAutoClr

LL_RTC_ALMA_GetBinAutoClr

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMA_GetBinAutoClr (const RTC_TypeDef * RTCx)

Function description

Get Alarm A Binary mode auto clear.

Parameters

- **RTCx:** RTC Instance

Return values

- **It:** can be one of the following values:
  - LL_RTC_ALMA_SUBSECONDBIN_AUTOCLR_NO
  - LL_RTC_ALMA_SUBSECONDBIN_AUTOCLR_YES

Reference Manual to LL API cross reference:

- RTC_ALRABINR SSCLR LL_RTC_ALMA_GetBinAutoClr

LL_RTC_ALMA_SetSubSecond

Function name

__STATIC_INLINE void LL_RTC_ALMA_SetSubSecond (RTC_TypeDef * RTCx, uint32_t Subsecond)

Function description

Set Alarm A Sub seconds value.

Parameters

- **RTCx:** RTC Instance
- **Subsecond:** If binary mode is none, Value between Min_Data=0x0 and Max_Data=0x7FFF else Value between Min_Data=0x0 and Max_Data=0xFFFFFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- RCT_ALRMASSR SS LL_RTC_ALMA_SetSubSecond

LL_RTC_ALMA_GetSubSecond

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMA_GetSubSecond (const RTC_TypeDef * RTCx)

Function description

Get Alarm A Sub seconds value.

Parameters

- **RTCx:** RTC Instance

Return values

- **If:** binary mode is none, Value between Min_Data=0x0 and Max_Data=0x7FFF else Value between Min_Data=0x0 and Max_Data=0xFFFFFFFF

Reference Manual to LL API cross reference:

- RCT_ALRMASSR SS LL_RTC_ALMA_GetSubSecond

LL_RTC_ALMB_Enable

Function name

__STATIC_INLINE void LL_RTC_ALMB_Enable (RTC_TypeDef * RTCx)

Function description

Enable Alarm B.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR ALRBE LL_RTC_ALMB_Enable

LL_RTC_ALMB_Disable

Function name

__STATIC_INLINE void LL_RTC_ALMB_Disable (RTC_TypeDef * RTCx)

Function description

Disable Alarm B.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR ALRBE LL_RTC_ALMB_Disable

LL_RTC_ALMB_SetMask

Function name

__STATIC_INLINE void LL_RTC_ALMB_SetMask (RTC_TypeDef * RTCx, uint32_t Mask)

Function description

Specify the Alarm B masks.

Parameters

- **RTCx:** RTC Instance
- **Mask:** This parameter can be a combination of the following values:
  - LL_RTC_ALMB_MASK_NONE
  - LL_RTC_ALMB_MASK_DATEWEEKDAY
  - LL_RTC_ALMB_MASK_HOURS
  - LL_RTC_ALMB_MASK_MINUTES
  - LL_RTC_ALMB_MASK_SECONDS
  - LL_RTC_ALMB_MASK_ALL

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_ALRMBR MSK4 LL_RTC_ALMB_SetMask
- RTC_ALRMBR MSK3 LL_RTC_ALMB_SetMask
- RTC_ALRMBR MSK2 LL_RTC_ALMB_SetMask
- RTC_ALRMBR MSK1 LL_RTC_ALMB_SetMask

LL_RTC_ALMB_GetMask

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMB_GetMask (const RTC_TypeDef * RTCx)

Function description

Get the Alarm B masks.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be can be a combination of the following values:
  - LL_RTC_ALMB_MASK_NONE
  - LL_RTC_ALMB_MASK_DATEWEEKDAY
  - LL_RTC_ALMB_MASK_HOURS
  - LL_RTC_ALMB_MASK_MINUTES
  - LL_RTC_ALMB_MASK_SECONDS
  - LL_RTC_ALMB_MASK_ALL

Reference Manual to LL API cross reference:

- RTC_ALRMBR MSK4 LL_RTC_ALMB_GetMask
- RTC_ALRMBR MSK3 LL_RTC_ALMB_GetMask
- RTC_ALRMBR MSK2 LL_RTC_ALMB_GetMask
- RTC_ALRMBR MSK1 LL_RTC_ALMB_GetMask

LL_RTC_ALMB_EnableWeekday

Function name

__STATIC_INLINE void LL_RTC_ALMB_EnableWeekday (RTC_TypeDef * RTCx)

Function description

Enable AlarmB Week day selection (DU[3:0] represents the week day.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_ALRMBR WDSEL LL_RTC_ALMB_EnableWeekday

LL_RTC_ALMB_DisableWeekday

Function name

__STATIC_INLINE void LL_RTC_ALMB_DisableWeekday (RTC_TypeDef * RTCx)

Function description

Disable AlarmB Week day selection (DU[3:0] represents the date)

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_ALRMBR WDSEL LL_RTC_ALMB_DisableWeekday

LL_RTC_ALMB_SetDay

Function name

__STATIC_INLINE void LL_RTC_ALMB_SetDay (RTC_TypeDef * RTCx, uint32_t Day)

Function description

Set ALARM B Day in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Day:** Value between Min_Data=0x01 and Max_Data=0x31

Return values

- **None:**

Notes

- helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Day from binary to BCD format

Reference Manual to LL API cross reference:

- RTC_ALRMBR DT LL_RTC_ALMB_SetDay
- RTC_ALRMBR DU LL_RTC_ALMB_SetDay

LL_RTC_ALMB_GetDay

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMB_GetDay (const RTC_TypeDef * RTCx)

Function description

Get ALARM B Day in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x01 and Max_Data=0x31

Notes

- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Day from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_ALRMBR DT LL_RTC_ALMB_GetDay
- RTC_ALRMBR DU LL_RTC_ALMB_GetDay

LL_RTC_ALMB_SetWeekDay

Function name

__STATIC_INLINE void LL_RTC_ALMB_SetWeekDay (RTC_TypeDef * RTCx, uint32_t WeekDay)

Function description

Set ALARM B Weekday.

Parameters

- **RTCx:** RTC Instance
- **WeekDay:** This parameter can be one of the following values:
  - LL_RTC_WEEKDAY_MONDAY
  - LL_RTC_WEEKDAY_TUESDAY
  - LL_RTC_WEEKDAY_WEDNESDAY
  - LL_RTC_WEEKDAY_THURSDAY
  - LL_RTC_WEEKDAY_FRIDAY
  - LL_RTC_WEEKDAY_SATURDAY
  - LL_RTC_WEEKDAY_SUNDAY

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_ALRMBR DU LL_RTC_ALMB_SetWeekDay

LL_RTC_ALMB_GetWeekDay

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMB_GetWeekDay (const RTC_TypeDef * RTCx)

Function description

Get ALARM B Weekday.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_WEEKDAY_MONDAY
  - LL_RTC_WEEKDAY_TUESDAY
  - LL_RTC_WEEKDAY_WEDNESDAY
  - LL_RTC_WEEKDAY_THURSDAY
  - LL_RTC_WEEKDAY_FRIDAY
  - LL_RTC_WEEKDAY_SATURDAY
  - LL_RTC_WEEKDAY_SUNDAY

Reference Manual to LL API cross reference:

- RTC_ALRMBR DU LL_RTC_ALMB_GetWeekDay

LL_RTC_ALMB_SetTimeFormat

Function name

__STATIC_INLINE void LL_RTC_ALMB_SetTimeFormat (RTC_TypeDef * RTCx, uint32_t TimeFormat)

Function description

Set ALARM B time format (AM/24-hour or PM notation)

Parameters

- **RTCx:** RTC Instance
- **TimeFormat:** This parameter can be one of the following values:
  - LL_RTC_ALMB_TIME_FORMAT_AM
  - LL_RTC_ALMB_TIME_FORMAT_PM

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_ALRMBR PM LL_RTC_ALMB_SetTimeFormat

LL_RTC_ALMB_GetTimeFormat

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMB_GetTimeFormat (const RTC_TypeDef * RTCx)

Function description

Get ALARM B time format (AM or PM notation)

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_ALMB_TIME_FORMAT_AM
  - LL_RTC_ALMB_TIME_FORMAT_PM

Reference Manual to LL API cross reference:

- RTC_ALRMBR PM LL_RTC_ALMB_GetTimeFormat

LL_RTC_ALMB_SetHour

Function name

__STATIC_INLINE void LL_RTC_ALMB_SetHour (RTC_TypeDef * RTCx, uint32_t Hours)

Function description

Set ALARM B Hours in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Hours:** Value between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23

Return values

- **None:**

Notes

- helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Hours from binary to BCD format

Reference Manual to LL API cross reference:

- RTC_ALRMBR HT LL_RTC_ALMB_SetHour
- RTC_ALRMBR HU LL_RTC_ALMB_SetHour

LL_RTC_ALMB_GetHour

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMB_GetHour (const RTC_TypeDef * RTCx)

Function description

Get ALARM B Hours in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23

Notes

- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Hours from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_ALRMBR HT LL_RTC_ALMB_GetHour
- RTC_ALRMBR HU LL_RTC_ALMB_GetHour

LL_RTC_ALMB_SetMinute

Function name

__STATIC_INLINE void LL_RTC_ALMB_SetMinute (RTC_TypeDef * RTCx, uint32_t Minutes)

Function description

Set ALARM B Minutes in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Minutes:** between Min_Data=0x00 and Max_Data=0x59

Return values

- **None:**

Notes

- helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Minutes from binary to BCD format

Reference Manual to LL API cross reference:

- RTC_ALRMBR MNT LL_RTC_ALMB_SetMinute
- RTC_ALRMBR MNU LL_RTC_ALMB_SetMinute

LL_RTC_ALMB_GetMinute

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMB_GetMinute (const RTC_TypeDef * RTCx)

Function description

Get ALARM B Minutes in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x59

Notes

- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Minutes from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_ALRMBR MNT LL_RTC_ALMB_GetMinute
- RTC_ALRMBR MNU LL_RTC_ALMB_GetMinute

LL_RTC_ALMB_SetSecond

Function name

__STATIC_INLINE void LL_RTC_ALMB_SetSecond (RTC_TypeDef * RTCx, uint32_t Seconds)

Function description

Set ALARM B Seconds in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Seconds:** Value between Min_Data=0x00 and Max_Data=0x59

Return values

- **None:**

Notes

- helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Seconds from binary to BCD format

Reference Manual to LL API cross reference:

- RTC_ALRMBR ST LL_RTC_ALMB_SetSecond
- RTC_ALRMBR SU LL_RTC_ALMB_SetSecond

LL_RTC_ALMB_GetSecond

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMB_GetSecond (const RTC_TypeDef * RTCx)

Function description

Get ALARM B Seconds in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x59

Notes

- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Seconds from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_ALRMBR ST LL_RTC_ALMB_GetSecond
- RTC_ALRMBR SU LL_RTC_ALMB_GetSecond

LL_RTC_ALMB_ConfigTime

Function name

__STATIC_INLINE void LL_RTC_ALMB_ConfigTime (RTC_TypeDef * RTCx, uint32_t Format12_24, uint32_t Hours, uint32_t Minutes, uint32_t Seconds)

Function description

Set Alarm B Time (hour, minute and second) in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Format12_24:** This parameter can be one of the following values:
  - LL_RTC_ALMB_TIME_FORMAT_AM
  - LL_RTC_ALMB_TIME_FORMAT_PM
- **Hours:** Value between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23
- **Minutes:** Value between Min_Data=0x00 and Max_Data=0x59
- **Seconds:** Value between Min_Data=0x00 and Max_Data=0x59

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_ALRMBR PM LL_RTC_ALMB_ConfigTime
- RTC_ALRMBR HT LL_RTC_ALMB_ConfigTime
- RTC_ALRMBR HU LL_RTC_ALMB_ConfigTime
- RTC_ALRMBR MNT LL_RTC_ALMB_ConfigTime
- RTC_ALRMBR MNU LL_RTC_ALMB_ConfigTime
- RTC_ALRMBR ST LL_RTC_ALMB_ConfigTime
- RTC_ALRMBR SU LL_RTC_ALMB_ConfigTime

LL_RTC_ALMB_GetTime

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMB_GetTime (const RTC_TypeDef * RTCx)

Function description

Get Alarm B Time (hour, minute and second) in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Combination:** of hours, minutes and seconds.

Notes

- helper macros __LL_RTC_GET_HOUR, __LL_RTC_GET_MINUTE and __LL_RTC_GET_SECOND are available to get independently each parameter.

Reference Manual to LL API cross reference:

- RTC_ALRMBR HT LL_RTC_ALMB_GetTime
- RTC_ALRMBR HU LL_RTC_ALMB_GetTime
- RTC_ALRMBR MNT LL_RTC_ALMB_GetTime
- RTC_ALRMBR MNU LL_RTC_ALMB_GetTime
- RTC_ALRMBR ST LL_RTC_ALMB_GetTime
- RTC_ALRMBR SU LL_RTC_ALMB_GetTime

LL_RTC_ALMB_SetSubSecondMask

Function name

__STATIC_INLINE void LL_RTC_ALMB_SetSubSecondMask (RTC_TypeDef * RTCx, uint32_t Mask)

Function description

Set Alarm B Mask the most-significant bits starting at this bit.

Parameters

- **RTCx:** RTC Instance
- **Mask:** If binary mode is none, Value between Min_Data=0x0 and Max_Data=0xF else Value between Min_Data=0x0 and Max_Data=0x3F

Return values

- **None:**

Notes

- This register can be written only when ALRBE is reset in RTC_CR register, or in initialization mode.

Reference Manual to LL API cross reference:

- RTC_ALRMBSSR MASKSS LL_RTC_ALMB_SetSubSecondMask

LL_RTC_ALMB_GetSubSecondMask

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMB_GetSubSecondMask (const RTC_TypeDef * RTCx)

Function description

Get Alarm B Mask the most-significant bits starting at this bit.

Parameters

- **RTCx:** RTC Instance

Return values

- **If:** binary mode is none, Value between Min_Data=0x0 and Max_Data=0xF else Value between Min_Data=0x0 and Max_Data=0x3F

Reference Manual to LL API cross reference:

- RTC_ALRMBSSR MASKSS LL_RTC_ALMB_GetSubSecondMask

LL_RTC_ALMB_SetBinAutoClr

Function name

__STATIC_INLINE void LL_RTC_ALMB_SetBinAutoClr (RTC_TypeDef * RTCx, uint32_t BinaryAutoClr)

Function description

Set Alarm B Binary mode auto clear.

Parameters

- **RTCx:** RTC Instance
- **BinaryAutoClr:** This parameter can be one of the following values:
  - LL_RTC_ALMB_SUBSECONDBIN_AUTOCLR_NO
  - LL_RTC_ALMB_SUBSECONDBIN_AUTOCLR_YES

Return values

- **None:**

Notes

- This register can be written only when ALRBE is reset in RTC_CR register, or in initialization mode.

Reference Manual to LL API cross reference:

- RTC_ALRBBINR SSCLR LL_RTC_ALMB_SetBinAutoClr

LL_RTC_ALMB_GetBinAutoClr

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMB_GetBinAutoClr (const RTC_TypeDef * RTCx)

Function description

Get Alarm B Binary mode auto clear.

Parameters

- **RTCx:** RTC Instance

Return values

- **It:** can be one of the following values:
  - LL_RTC_ALMB_SUBSECONDBIN_AUTOCLR_NO
  - LL_RTC_ALMB_SUBSECONDBIN_AUTOCLR_YES

Reference Manual to LL API cross reference:

- RTC_ALRBBINR SSCLR LL_RTC_ALMB_GetBinAutoClr

LL_RTC_ALMB_SetSubSecond

Function name

__STATIC_INLINE void LL_RTC_ALMB_SetSubSecond (RTC_TypeDef * RTCx, uint32_t Subsecond)

Function description

Set Alarm B Sub seconds value.

Parameters

- **RTCx:** RTC Instance
- **Subsecond:** If binary mode is none, Value between Min_Data=0x0 and Max_Data=0x7FFF else Value between Min_Data=0x0 and Max_Data=0xFFFFFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_ALRMBSSR SS LL_RTC_ALMB_SetSubSecond

LL_RTC_ALMB_GetSubSecond

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMB_GetSubSecond (const RTC_TypeDef * RTCx)

Function description

Get Alarm B Sub seconds value.

Parameters

- **RTCx:** RTC Instance

Return values

- **If:** binary mode is none, Value between Min_Data=0x0 and Max_Data=0x7FFF else Value between Min_Data=0x0 and Max_Data=0xFFFFFFFF

Reference Manual to LL API cross reference:

- RTC_ALRMBSSR SS LL_RTC_ALMB_GetSubSecond

LL_RTC_TS_EnableInternalEvent

Function name

__STATIC_INLINE void LL_RTC_TS_EnableInternalEvent (RTC_TypeDef * RTCx)

Function description

Enable internal event timestamp.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR ITSE LL_RTC_TS_EnableInternalEvent

LL_RTC_TS_DisableInternalEvent

Function name

__STATIC_INLINE void LL_RTC_TS_DisableInternalEvent (RTC_TypeDef * RTCx)

Function description

Disable internal event timestamp.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR ITSE LL_RTC_TS_DisableInternalEvent

LL_RTC_TS_Enable

Function name

__STATIC_INLINE void LL_RTC_TS_Enable (RTC_TypeDef * RTCx)

Function description

Enable Timestamp.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR TSE LL_RTC_TS_Enable

LL_RTC_TS_Disable

Function name

__STATIC_INLINE void LL_RTC_TS_Disable (RTC_TypeDef * RTCx)

Function description

Disable Timestamp.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR TSE LL_RTC_TS_Disable

LL_RTC_TS_SetActiveEdge

Function name

__STATIC_INLINE void LL_RTC_TS_SetActiveEdge (RTC_TypeDef * RTCx, uint32_t Edge)

Function description

Set Time-stamp event active edge.

Parameters

- **RTCx:** RTC Instance
- **Edge:** This parameter can be one of the following values:
  - LL_RTC_TIMESTAMP_EDGE_RISING
  - LL_RTC_TIMESTAMP_EDGE_FALLING

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- TSE must be reset when TSEDGE is changed to avoid unwanted TSF setting

Reference Manual to LL API cross reference:

- RTC_CR TSEDGE LL_RTC_TS_SetActiveEdge

LL_RTC_TS_GetActiveEdge

Function name

__STATIC_INLINE uint32_t LL_RTC_TS_GetActiveEdge (const RTC_TypeDef * RTCx)

Function description

Get Time-stamp event active edge.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_TIMESTAMP_EDGE_RISING
  - LL_RTC_TIMESTAMP_EDGE_FALLING

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR TSEDGE LL_RTC_TS_GetActiveEdge

LL_RTC_TS_GetTimeFormat

Function name

__STATIC_INLINE uint32_t LL_RTC_TS_GetTimeFormat (const RTC_TypeDef * RTCx)

Function description

Get Timestamp AM/PM notation (AM or 24-hour format)

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_TS_TIME_FORMAT_AM
  - LL_RTC_TS_TIME_FORMAT_PM

Reference Manual to LL API cross reference:

- RTC_TSTR PM LL_RTC_TS_GetTimeFormat

LL_RTC_TS_GetHour

Function name

__STATIC_INLINE uint32_t LL_RTC_TS_GetHour (const RTC_TypeDef * RTCx)

Function description

Get Timestamp Hours in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23

Notes

- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Hours from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_TSTR HT LL_RTC_TS_GetHour
- RTC_TSTR HU LL_RTC_TS_GetHour

LL_RTC_TS_GetMinute

Function name

__STATIC_INLINE uint32_t LL_RTC_TS_GetMinute (const RTC_TypeDef * RTCx)

Function description

Get Timestamp Minutes in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x59

Notes

- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Minutes from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_TSTR MNT LL_RTC_TS_GetMinute
- RTC_TSTR MNU LL_RTC_TS_GetMinute

LL_RTC_TS_GetSecond

Function name

__STATIC_INLINE uint32_t LL_RTC_TS_GetSecond (const RTC_TypeDef * RTCx)

Function description

Get Timestamp Seconds in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x59

Notes

- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Seconds from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_TSTR ST LL_RTC_TS_GetSecond
- RTC_TSTR SU LL_RTC_TS_GetSecond

LL_RTC_TS_GetTime

Function name

__STATIC_INLINE uint32_t LL_RTC_TS_GetTime (const RTC_TypeDef * RTCx)

Function description

Get Timestamp time (hour, minute and second) in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Combination:** of hours, minutes and seconds.

Notes

- helper macros __LL_RTC_GET_HOUR, __LL_RTC_GET_MINUTE and __LL_RTC_GET_SECOND are available to get independently each parameter.

Reference Manual to LL API cross reference:

- RTC_TSTR HT LL_RTC_TS_GetTime
- RTC_TSTR HU LL_RTC_TS_GetTime
- RTC_TSTR MNT LL_RTC_TS_GetTime
- RTC_TSTR MNU LL_RTC_TS_GetTime
- RTC_TSTR ST LL_RTC_TS_GetTime
- RTC_TSTR SU LL_RTC_TS_GetTime

LL_RTC_TS_GetWeekDay

Function name

__STATIC_INLINE uint32_t LL_RTC_TS_GetWeekDay (const RTC_TypeDef * RTCx)

Function description

Get Timestamp Week day.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_WEEKDAY_MONDAY
  - LL_RTC_WEEKDAY_TUESDAY
  - LL_RTC_WEEKDAY_WEDNESDAY
  - LL_RTC_WEEKDAY_THURSDAY
  - LL_RTC_WEEKDAY_FRIDAY
  - LL_RTC_WEEKDAY_SATURDAY
  - LL_RTC_WEEKDAY_SUNDAY

Reference Manual to LL API cross reference:

- RTC_TSDR WDU LL_RTC_TS_GetWeekDay

LL_RTC_TS_GetMonth

Function name

__STATIC_INLINE uint32_t LL_RTC_TS_GetMonth (const RTC_TypeDef * RTCx)

Function description

Get Timestamp Month in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_MONTH_JANUARY
  - LL_RTC_MONTH_FEBRUARY
  - LL_RTC_MONTH_MARCH
  - LL_RTC_MONTH_APRIL
  - LL_RTC_MONTH_MAY
  - LL_RTC_MONTH_JUNE
  - LL_RTC_MONTH_JULY
  - LL_RTC_MONTH_AUGUST
  - LL_RTC_MONTH_SEPTEMBER
  - LL_RTC_MONTH_OCTOBER
  - LL_RTC_MONTH_NOVEMBER
  - LL_RTC_MONTH_DECEMBER

Notes

- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Month from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_TSDR MT LL_RTC_TS_GetMonth
- RTC_TSDR MU LL_RTC_TS_GetMonth

LL_RTC_TS_GetDay

Function name

__STATIC_INLINE uint32_t LL_RTC_TS_GetDay (const RTC_TypeDef * RTCx)

Function description

Get Timestamp Day in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x01 and Max_Data=0x31

Notes

- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Day from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_TSDR DT LL_RTC_TS_GetDay
- RTC_TSDR DU LL_RTC_TS_GetDay

LL_RTC_TS_GetDate

Function name

__STATIC_INLINE uint32_t LL_RTC_TS_GetDate (const RTC_TypeDef * RTCx)

Function description

Get Timestamp date (WeekDay, Day and Month) in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Combination:** of Weekday, Day and Month

Notes

- helper macros __LL_RTC_GET_WEEKDAY, __LL_RTC_GET_MONTH, and __LL_RTC_GET_DAY are available to get independently each parameter.

Reference Manual to LL API cross reference:

- RTC_TSDR WDU LL_RTC_TS_GetDate
- RTC_TSDR MT LL_RTC_TS_GetDate
- RTC_TSDR MU LL_RTC_TS_GetDate
- RTC_TSDR DT LL_RTC_TS_GetDate
- RTC_TSDR DU LL_RTC_TS_GetDate

LL_RTC_TS_GetSubSecond

Function name

__STATIC_INLINE uint32_t LL_RTC_TS_GetSubSecond (const RTC_TypeDef * RTCx)

Function description

Get time-stamp sub second value.

Parameters

- **RTCx:** RTC Instance

Return values

- **If:** binary mode is none, Value between Min_Data=0x0 and Max_Data=0x7FFF else Value between Min_Data=0x0 and Max_Data=0xFFFFFFFF

Reference Manual to LL API cross reference:

- RTC_TSDR SS LL_RTC_TS_GetSubSecond

LL_RTC_TS_EnableOnTamper

Function name

__STATIC_INLINE void LL_RTC_TS_EnableOnTamper (RTC_TypeDef * RTCx)

Function description

Activate timestamp on tamper detection event.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_CR TAMPTS LL_RTC_TS_EnableOnTamper

LL_RTC_TS_DisableOnTamper

Function name

__STATIC_INLINE void LL_RTC_TS_DisableOnTamper (RTC_TypeDef * RTCx)

Function description

Disable timestamp on tamper detection event.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_CR TAMPTS LL_RTC_TS_DisableOnTamper

LL_RTC_TAMPER_Enable

Function name

__STATIC_INLINE void LL_RTC_TAMPER_Enable (const RTC_TypeDef * RTCx, uint32_t Tamper)

Function description

Enable TAMPx input detection.

Parameters

- **RTCx:** RTC Instance
- **Tamper:** This parameter can be a combination of the following values:
  - TAMPER

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_CR1 TAMP1E LL_RTC_TAMPER_Enable
- TAMP_CR1 TAMP2E... LL_RTC_TAMPER_Enable
-

LL_RTC_TAMPER_Disable

Function name

__STATIC_INLINE void LL_RTC_TAMPER_Disable (const RTC_TypeDef * RTCx, uint32_t Tamper)

Function description

Clear TAMPx input detection.

Parameters

- **RTCx:** RTC Instance
- **Tamper:** This parameter can be a combination of the following values:
  - TAMPER

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_CR1 TAMP1E LL_RTC_TAMPER_Disable
- TAMP_CR1 TAMP2E... LL_RTC_TAMPER_Disable

LL_RTC_TAMPER_EnableMask

Function name

__STATIC_INLINE void LL_RTC_TAMPER_EnableMask (const RTC_TypeDef * RTCx, uint32_t Mask)

Function description

Enable Tamper mask flag.

Parameters

- **RTCx:** RTC Instance
- **Mask:** This parameter can be a combination of the following values:
  - TAMPER MASK

Return values

- **None:**

Notes

- Associated Tamper IT must not enabled when tamper mask is set.

Reference Manual to LL API cross reference:

- TAMP_CR2 TAMP1MF LL_RTC_TAMPER_EnableMask
- TAMP_CR2 TAMP2MF... LL_RTC_TAMPER_EnableMask

LL_RTC_TAMPER_DisableMask

Function name

__STATIC_INLINE void LL_RTC_TAMPER_DisableMask (const RTC_TypeDef * RTCx, uint32_t Mask)

Function description

Disable Tamper mask flag.

Parameters

- **RTCx:** RTC Instance
- **Mask:** This parameter can be a combination of the following values:
  - TAMPER MASK

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_CR2 TAMP1MF LL_RTC_TAMPER_DisableMask
- TAMP_CR2 TAMP2MF... LL_RTC_TAMPER_DisableMask

LL_RTC_TAMPER_EnableEraseBKP

Function name

__STATIC_INLINE void LL_RTC_TAMPER_EnableEraseBKP (const RTC_TypeDef * RTCx, uint32_t Tamper)

Function description

Enable backup register erase after Tamper event detection.

Parameters

- **RTCx:** RTC Instance
- **Tamper:** This parameter can be a combination of the following values:
  - TAMPER NO ERASE

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_CR2 TAMP1POM LL_RTC_TAMPER_EnableEraseBKP
- TAMP_CR2 TAMP2POM... LL_RTC_TAMPER_EnableEraseBKP

LL_RTC_TAMPER_DisableEraseBKP

Function name

__STATIC_INLINE void LL_RTC_TAMPER_DisableEraseBKP (const RTC_TypeDef * RTCx, uint32_t Tamper)

Function description

Disable backup register erase after Tamper event detection.

Parameters

- **RTCx:** RTC Instance
- **Tamper:** This parameter can be a combination of the following values:
  - TAMPER NO ERASE

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_CR2 TAMP1POM LL_RTC_TAMPER_DisableEraseBKP
- TAMP_CR2 TAMP2POM... LL_RTC_TAMPER_DisableEraseBKP

LL_RTC_TAMPER_DisablePullUp

Function name

__STATIC_INLINE void LL_RTC_TAMPER_DisablePullUp (const RTC_TypeDef * RTCx)

Function description

Disable RTC_TAMPx pull-up disable (Disable precharge of RTC_TAMPx pins)

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_FLTCR TAMPPUDIS LL_RTC_TAMPER_DisablePullUp

LL_RTC_TAMPER_EnablePullUp

Function name

__STATIC_INLINE void LL_RTC_TAMPER_EnablePullUp (const RTC_TypeDef * RTCx)

Function description

Enable RTC_TAMPx pull-up disable (Precharge RTC_TAMPx pins before sampling)

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_FLTCR TAMPPUDIS LL_RTC_TAMPER_EnablePullUp

LL_RTC_TAMPER_SetPrecharge

Function name

__STATIC_INLINE void LL_RTC_TAMPER_SetPrecharge (const RTC_TypeDef * RTCx, uint32_t Duration)

Function description

Set RTC_TAMPx precharge duration.

Parameters

- **RTCx:** RTC Instance
- **Duration:** This parameter can be one of the following values:
  - LL_RTC_TAMPER_DURATION_1RTCCLK
  - LL_RTC_TAMPER_DURATION_2RTCCLK
  - LL_RTC_TAMPER_DURATION_4RTCCLK
  - LL_RTC_TAMPER_DURATION_8RTCCLK

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_FLTCR TAMPPRCH LL_RTC_TAMPER_SetPrecharge

LL_RTC_TAMPER_GetPrecharge

Function name

__STATIC_INLINE uint32_t LL_RTC_TAMPER_GetPrecharge (const RTC_TypeDef * RTCx)

Function description

Get RTC_TAMPx precharge duration.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_TAMPER_DURATION_1RTCCLK
  - LL_RTC_TAMPER_DURATION_2RTCCLK
  - LL_RTC_TAMPER_DURATION_4RTCCLK
  - LL_RTC_TAMPER_DURATION_8RTCCLK

Reference Manual to LL API cross reference:

- TAMP_FLTCR TAMPPRCH LL_RTC_TAMPER_GetPrecharge

LL_RTC_TAMPER_SetFilterCount

Function name

__STATIC_INLINE void LL_RTC_TAMPER_SetFilterCount (const RTC_TypeDef * RTCx, uint32_t FilterCount)

Function description

Set RTC_TAMPx filter count.

Parameters

- **RTCx:** RTC Instance
- **FilterCount:** This parameter can be one of the following values:
  - LL_RTC_TAMPER_FILTER_DISABLE
  - LL_RTC_TAMPER_FILTER_2SAMPLE
  - LL_RTC_TAMPER_FILTER_4SAMPLE
  - LL_RTC_TAMPER_FILTER_8SAMPLE

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_FLTCR TAMPFLT LL_RTC_TAMPER_SetFilterCount

LL_RTC_TAMPER_GetFilterCount

Function name

__STATIC_INLINE uint32_t LL_RTC_TAMPER_GetFilterCount (const RTC_TypeDef * RTCx)

Function description

Get RTC_TAMPx filter count.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_TAMPER_FILTER_DISABLE
  - LL_RTC_TAMPER_FILTER_2SAMPLE
  - LL_RTC_TAMPER_FILTER_4SAMPLE
  - LL_RTC_TAMPER_FILTER_8SAMPLE

Reference Manual to LL API cross reference:

- TAMP_FLTCR TAMPFLT LL_RTC_TAMPER_GetFilterCount

LL_RTC_TAMPER_SetSamplingFreq

Function name

__STATIC_INLINE void LL_RTC_TAMPER_SetSamplingFreq (const RTC_TypeDef * RTCx, uint32_t SamplingFreq)

Function description

Set Tamper sampling frequency.

Parameters

- **RTCx:** RTC Instance
- **SamplingFreq:** This parameter can be one of the following values:
  - LL_RTC_TAMPER_SAMPLFREQDIV_32768
  - LL_RTC_TAMPER_SAMPLFREQDIV_16384
  - LL_RTC_TAMPER_SAMPLFREQDIV_8192
  - LL_RTC_TAMPER_SAMPLFREQDIV_4096
  - LL_RTC_TAMPER_SAMPLFREQDIV_2048
  - LL_RTC_TAMPER_SAMPLFREQDIV_1024
  - LL_RTC_TAMPER_SAMPLFREQDIV_512
  - LL_RTC_TAMPER_SAMPLFREQDIV_256

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_FLTCR TAMPFREQ LL_RTC_TAMPER_SetSamplingFreq

LL_RTC_TAMPER_GetSamplingFreq

Function name

__STATIC_INLINE uint32_t LL_RTC_TAMPER_GetSamplingFreq (const RTC_TypeDef * RTCx)

Function description

Get Tamper sampling frequency.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_TAMPER_SAMPLFREQDIV_32768
  - LL_RTC_TAMPER_SAMPLFREQDIV_16384
  - LL_RTC_TAMPER_SAMPLFREQDIV_8192
  - LL_RTC_TAMPER_SAMPLFREQDIV_4096
  - LL_RTC_TAMPER_SAMPLFREQDIV_2048
  - LL_RTC_TAMPER_SAMPLFREQDIV_1024
  - LL_RTC_TAMPER_SAMPLFREQDIV_512
  - LL_RTC_TAMPER_SAMPLFREQDIV_256

Reference Manual to LL API cross reference:

- TAMP_FLTCR TAMPFREQ LL_RTC_TAMPER_GetSamplingFreq

LL_RTC_TAMPER_EnableActiveLevel

Function name

__STATIC_INLINE void LL_RTC_TAMPER_EnableActiveLevel (const RTC_TypeDef * RTCx, uint32_t Tamper)

Function description

Enable Active level for Tamper input.

Parameters

- **RTCx:** RTC Instance
- **Tamper:** This parameter can be a combination of the following values:
  - TAMPER ACTIVE LEVEL

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_CR2 TAMP1TRG LL_RTC_TAMPER_EnableActiveLevel
- TAMP_CR2 TAMP2TRG LL_RTC_TAMPER_EnableActiveLevel
- TAMP_CR2 TAMPxTRG LL_RTC_TAMPER_EnableActiveLevel
-

LL_RTC_TAMPER_DisableActiveLevel

Function name

__STATIC_INLINE void LL_RTC_TAMPER_DisableActiveLevel (const RTC_TypeDef * RTCx, uint32_t Tamper)

Function description

Disable Active level for Tamper input.

Parameters

- **RTCx:** RTC Instance
- **Tamper:** This parameter can be a combination of the following values:
  - TAMPER ACTIVE LEVEL

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_CR2 TAMP1TRG LL_RTC_TAMPER_DisableActiveLevel
- TAMP_CR2 TAMP2TRG LL_RTC_TAMPER_DisableActiveLevel
- TAMP_CR2 TAMPxTRG LL_RTC_TAMPER_DisableActiveLevel
-

LL_RTC_TAMPER_ITAMP_Enable

Function name

__STATIC_INLINE void LL_RTC_TAMPER_ITAMP_Enable (const RTC_TypeDef * RTCx, uint32_t InternalTamper)

Function description

Enable internal tamper detection.

Parameters

- **RTCx:** RTC Instance
- **InternalTamper:** This parameter can be a combination of the following values:
  - INTERNAL TAMPER

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_CR1 ITAMP1E LL_RTC_TAMPER_ITAMP_Enable
- TAMP_CR1 ITAMP2E LL_RTC_TAMPER_ITAMP_Enable
- TAMP_CR1 ITAMPxE LL_RTC_TAMPER_ITAMP_Enable

LL_RTC_TAMPER_ITAMP_Disable

Function name

__STATIC_INLINE void LL_RTC_TAMPER_ITAMP_Disable (const RTC_TypeDef * RTCx, uint32_t InternalTamper)

Function description

Disable internal tamper detection.

Parameters

- **RTCx:** RTC Instance
- **InternalTamper:** This parameter can be a combination of the following values:
  - INTERNAL TAMPER

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_CR1 ITAMP1E LL_RTC_TAMPER_ITAMP_Disable
- TAMP_CR1 ITAMP2E LL_RTC_TAMPER_ITAMP_Disable
- TAMP_CR1 ITAMPxE LL_RTC_TAMPER_ITAMP_Disable
-

LL_RTC_TAMPER_ITAMP_EnableEraseBKP

Function name

__STATIC_INLINE void LL_RTC_TAMPER_ITAMP_EnableEraseBKP (const RTC_TypeDef * RTCx, uint32_t InternalTamper)

Function description

Enable backup register erase after internal tamper event detection.

Parameters

- **RTCx:** RTC Instance
- **InternalTamper:** This parameter can be a combination of the following values:
  - INTERNAL TAMPER NO ERASE

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_CR3 ITAMP1POM LL_RTC_TAMPER_ITAMP_EnableEraseBKP TAMP_CR3 ITAMP2POM... LL_RTC_TAMPER_ITAMP_EnableEraseBKP

LL_RTC_TAMPER_ITAMP_DisableEraseBKP

Function name

__STATIC_INLINE void LL_RTC_TAMPER_ITAMP_DisableEraseBKP (const RTC_TypeDef * RTCx, uint32_t InternalTamper)

Function description

Disable backup register erase after internal tamper event detection.

Parameters

- **RTCx:** RTC Instance
- **InternalTamper:** This parameter can be a combination of the following values:
  - INTERNAL TAMPER NO ERASE

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_CR3 ITAMP1POM LL_RTC_TAMPER_ITAMP_DisableEraseBKP TAMP_CR3 ITAMP2POM... LL_RTC_TAMPER_ITAMP_DisableEraseBKP

LL_RTC_TAMPER_ATAMP_EnableActiveMode

Function name

__STATIC_INLINE void LL_RTC_TAMPER_ATAMP_EnableActiveMode (uint32_t Tamper)

Function description

Enable tamper active mode.

Parameters

- **Tamper:** to configure as active. This parameter can be a combination of the following values:
  - ACTIVE TAMPER MODE

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_ATCR1 TAMP1AM LL_RTC_TAMPER_ATAMP_EnableActiveMode
-
- TAMP_ATCR1 TAMP2AM LL_RTC_TAMPER_ATAMP_EnableActiveMode
-
- TAMP_ATCR1 TAMPxAM LL_RTC_TAMPER_ATAMP_EnableActiveMode
-

LL_RTC_TAMPER_ATAMP_DisableActiveMode

Function name

__STATIC_INLINE void LL_RTC_TAMPER_ATAMP_DisableActiveMode (uint32_t Tamper)

Function description

Disable tamper active mode.

Parameters

- **Tamper:** to configure as active. This parameter can be a combination of the following values:
  - ACTIVE TAMPER MODE

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_ATCR1 TAMP1AM LL_RTC_TAMPER_ATAMP_DisableActiveMode
-
- TAMP_ATCR1 TAMP2AM LL_RTC_TAMPER_ATAMP_DisableActiveMode
-
- TAMP_ATCR1 TAMPxAM LL_RTC_TAMPER_ATAMP_DisableActiveMode
-

LL_RTC_TAMPER_ATAMP_EnableFilter

Function name

__STATIC_INLINE void LL_RTC_TAMPER_ATAMP_EnableFilter (void)

Function description

Enable active tamper filter.

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_ATCR1 FLTEN LL_RTC_TAMPER_ATAMP_EnableFilter
-

LL_RTC_TAMPER_ATAMP_DisableFilter

Function name

__STATIC_INLINE void LL_RTC_TAMPER_ATAMP_DisableFilter (void)

Function description

Disable active tamper filter.

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_ATCR1 FLTEN LL_RTC_TAMPER_ATAMP_DisableFilter
-

LL_RTC_TAMPER_ATAMP_SetOutputChangePeriod

Function name

__STATIC_INLINE void LL_RTC_TAMPER_ATAMP_SetOutputChangePeriod (uint32_t ActiveOutputChangePeriod)

Function description

Set Active tamper output change period.

Parameters

- **ActiveOutputChangePeriod:** This parameter can be a value from 0 to 7

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_ATCR1 ATPER LL_RTC_TAMPER_ATAMP_SetOutputChangePeriod
-

LL_RTC_TAMPER_ATAMP_GetOutputChangePeriod

Function name

__STATIC_INLINE uint32_t LL_RTC_TAMPER_ATAMP_GetOutputChangePeriod (void)

Function description

Get Active tamper output change period.

Return values

- **Output:** change period. This parameter can be a value from 0 to 7.

Reference Manual to LL API cross reference:

- TAMP_ATCR1 ATPER LL_RTC_TAMPER_ATAMP_GetOutputChangePeriod
-

LL_RTC_TAMPER_ATAMP_SetAsyncPrescaler

Function name

__STATIC_INLINE void LL_RTC_TAMPER_ATAMP_SetAsyncPrescaler (uint32_t ActiveAsynvPrescaler)

Function description

Set Active tamper asynchronous prescaler clock selection.

Parameters

- **ActiveAsynvPrescaler:** Specifies the Active Tamper asynchronous Prescaler clock. This parameter can be a value of the following values:
  - ACTIVE TAMPER ASYNCHRONOUS PRESCALER CLOCK

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_ATCR1 ATCKSEL LL_RTC_TAMPER_ATAMP_SetAsyncPrescaler
-

LL_RTC_TAMPER_ATAMP_GetAsyncPrescaler

Function name

__STATIC_INLINE uint32_t LL_RTC_TAMPER_ATAMP_GetAsyncPrescaler (void)

Function description

Get Active tamper asynchronous prescaler clock selection.

Return values

- **One:** of
  - ACTIVE TAMPER ASYNCHRONOUS PRESCALER CLOCK

Reference Manual to LL API cross reference:

- TAMP_ATCR1 ATCKSEL LL_RTC_TAMPER_ATAMP_GetAsyncPrescaler
-

LL_RTC_TAMPER_ATAMP_EnableOutputSharing

Function name

__STATIC_INLINE void LL_RTC_TAMPER_ATAMP_EnableOutputSharing (void)

Function description

Enable active tamper output sharing.

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_ATCR1 ATOSHARE LL_RTC_TAMPER_ATAMP_EnableOutputSharing
-

LL_RTC_TAMPER_ATAMP_DisableOutputSharing

Function name

__STATIC_INLINE void LL_RTC_TAMPER_ATAMP_DisableOutputSharing (void)

Function description

Disable active tamper output sharing.

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_ATCR1 ATOSHARE LL_RTC_TAMPER_ATAMP_DisableOutputSharing
-

LL_RTC_TAMPER_ATAMP_SetSharedOuputSelection

Function name

__STATIC_INLINE void LL_RTC_TAMPER_ATAMP_SetSharedOuputSelection (uint32_t OutputSelection)

Function description

Set Active tamper shared output selection.

Parameters

- **OutputSelection:** Specifies all the output selection of the Active Tamper. This parameter is a combinasation of the following values: One of
  - ACTIVE TAMPER OUTPUT SELECTION

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_ATCR2 ATOSELx LL_RTC_TAMPER_ATAMP_SetSharedOuputSelection
-

LL_RTC_TAMPER_ATAMP_GetSharedOuputSelection

Function name

__STATIC_INLINE uint32_t LL_RTC_TAMPER_ATAMP_GetSharedOuputSelection (void)

Function description

Get Active tamper shared output selection.

Return values

- **A:** combination of
  - ACTIVE TAMPER OUTPUT SELECTION

Reference Manual to LL API cross reference:

- TAMP_ATCR2 ATOSELx LL_RTC_TAMPER_ATAMP_GetSharedOuputSelection
-

LL_RTC_TAMPER_ATAMP_WriteSeed

Function name

__STATIC_INLINE void LL_RTC_TAMPER_ATAMP_WriteSeed (uint32_t Seed)

Function description

Write active tamper seed.

Parameters

- **Seed:**

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_ATSEEDR SEED LL_RTC_TAMPER_ATAMP_WriteSeed
-

LL_RTC_IsActiveFlag_ATAMP_INITS

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ATAMP_INITS (void)

Function description

Get active tamper initialization status flag.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_ATOR INITS LL_RTC_IsActiveFlag_ATAMP_INITS

LL_RTC_IsActiveFlag_ATAMP_SEEDF

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ATAMP_SEEDF (void)

Function description

Get active tamper seed running status flag.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_ATOR SEEDF LL_RTC_IsActiveFlag_ATAMP_SEEDF

LL_RTC_WAKEUP_Enable

Function name

__STATIC_INLINE void LL_RTC_WAKEUP_Enable (RTC_TypeDef * RTCx)

Function description

Enable Wakeup timer.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR WUTE LL_RTC_WAKEUP_Enable

LL_RTC_WAKEUP_Disable

Function name

__STATIC_INLINE void LL_RTC_WAKEUP_Disable (RTC_TypeDef * RTCx)

Function description

Disable Wakeup timer.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR WUTE LL_RTC_WAKEUP_Disable

LL_RTC_WAKEUP_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_RTC_WAKEUP_IsEnabled (const RTC_TypeDef * RTCx)

Function description

Check if Wakeup timer is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_CR WUTE LL_RTC_WAKEUP_IsEnabled

LL_RTC_WAKEUP_SetClock

Function name

__STATIC_INLINE void LL_RTC_WAKEUP_SetClock (RTC_TypeDef * RTCx, uint32_t WakeupClock)

Function description

Select Wakeup clock.

Parameters

- **RTCx:** RTC Instance
- **WakeupClock:** This parameter can be one of the following values:
  - LL_RTC_WAKEUPCLOCK_DIV_16
  - LL_RTC_WAKEUPCLOCK_DIV_8
  - LL_RTC_WAKEUPCLOCK_DIV_4
  - LL_RTC_WAKEUPCLOCK_DIV_2
  - LL_RTC_WAKEUPCLOCK_CKSPRE
  - LL_RTC_WAKEUPCLOCK_CKSPRE_WUT

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- Bit can be written only when RTC_CR WUTE bit = 0 and RTC_ICSR WUTWF bit = 1

Reference Manual to LL API cross reference:

- RTC_CR WUCKSEL LL_RTC_WAKEUP_SetClock

LL_RTC_WAKEUP_GetClock

Function name

__STATIC_INLINE uint32_t LL_RTC_WAKEUP_GetClock (const RTC_TypeDef * RTCx)

Function description

Get Wakeup clock.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_WAKEUPCLOCK_DIV_16
  - LL_RTC_WAKEUPCLOCK_DIV_8
  - LL_RTC_WAKEUPCLOCK_DIV_4
  - LL_RTC_WAKEUPCLOCK_DIV_2
  - LL_RTC_WAKEUPCLOCK_CKSPRE
  - LL_RTC_WAKEUPCLOCK_CKSPRE_WUT

Reference Manual to LL API cross reference:

- RTC_CR WUCKSEL LL_RTC_WAKEUP_GetClock

LL_RTC_WAKEUP_SetAutoReload

Function name

__STATIC_INLINE void LL_RTC_WAKEUP_SetAutoReload (RTC_TypeDef * RTCx, uint32_t Value)

Function description

Set Wakeup auto-reload value.

Parameters

- **RTCx:** RTC Instance
- **Value:** Value between Min_Data=0x00 and Max_Data=0xFFFF

Return values

- **None:**

Notes

- Bit can be written only when WUTWF is set to 1 in RTC_ICSR

Reference Manual to LL API cross reference:

- RTC_WUTR WUT LL_RTC_WAKEUP_SetAutoReload

LL_RTC_WAKEUP_GetAutoReload

Function name

__STATIC_INLINE uint32_t LL_RTC_WAKEUP_GetAutoReload (const RTC_TypeDef * RTCx)

Function description

Get Wakeup auto-reload value.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0xFFFF

Reference Manual to LL API cross reference:

- RTC_WUTR WUT LL_RTC_WAKEUP_GetAutoReload

LL_RTC_BKP_SetRegister

Function name

__STATIC_INLINE void LL_RTC_BKP_SetRegister (const RTC_TypeDef * RTCx, uint32_t BackupRegister, uint32_t Data)

Function description

Writes a data in a specified Backup data register.

Parameters

- **RTCx:** RTC Instance
- **BackupRegister:** This parameter can be one of the following values:
  - LL_RTC_BKP_DR0
  - LL_RTC_BKP_DR1
  - LL_RTC_BKP_DR2
  - LL_RTC_BKP_DR3
  - LL_RTC_BKP_DR4
  - LL_RTC_BKP_DRx ...
- **Data:** Value between Min_Data=0x00 and Max_Data=0xFFFFFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_BKPxR BKP LL_RTC_BKP_SetRegister

LL_RTC_BKP_GetRegister

Function name

__STATIC_INLINE uint32_t LL_RTC_BKP_GetRegister (const RTC_TypeDef * RTCx, uint32_t BackupRegister)

Function description

Reads data from the specified RTC Backup data Register.

Parameters

- **RTCx:** RTC Instance
- **BackupRegister:** This parameter can be one of the following values:
  - LL_RTC_BKP_DR0
  - LL_RTC_BKP_DR1
  - LL_RTC_BKP_DR2
  - LL_RTC_BKP_DR3
  - LL_RTC_BKP_DR4
  - LL_RTC_BKP_DRx ...

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0xFFFFFFFF

Reference Manual to LL API cross reference:

- TAMP_BKPxR BKP LL_RTC_BKP_GetRegister

LL_RTC_CAL_SetOutputFreq

Function name

__STATIC_INLINE void LL_RTC_CAL_SetOutputFreq (RTC_TypeDef * RTCx, uint32_t Frequency)

Function description

Set Calibration output frequency (1 Hz or 512 Hz)

Parameters

- **RTCx:** RTC Instance
- **Frequency:** This parameter can be one of the following values:
  - LL_RTC_CALIB_OUTPUT_NONE
  - LL_RTC_CALIB_OUTPUT_1HZ
  - LL_RTC_CALIB_OUTPUT_512HZ

Return values

- **None:**

Notes

- Bits are write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR COE LL_RTC_CAL_SetOutputFreq
- RTC_CR COSEL LL_RTC_CAL_SetOutputFreq

LL_RTC_CAL_GetOutputFreq

Function name

__STATIC_INLINE uint32_t LL_RTC_CAL_GetOutputFreq (const RTC_TypeDef * RTCx)

Function description

Get Calibration output frequency (1 Hz or 512 Hz)

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_CALIB_OUTPUT_NONE
  - LL_RTC_CALIB_OUTPUT_1HZ
  - LL_RTC_CALIB_OUTPUT_512HZ

Reference Manual to LL API cross reference:

- RTC_CR COE LL_RTC_CAL_GetOutputFreq
- RTC_CR COSEL LL_RTC_CAL_GetOutputFreq

LL_RTC_CAL_SetPulse

Function name

__STATIC_INLINE void LL_RTC_CAL_SetPulse (RTC_TypeDef * RTCx, uint32_t Pulse)

Function description

Insert or not One RTCCLK pulse every 2exp11 pulses (frequency increased by 488.5 ppm)

Parameters

- **RTCx:** RTC Instance
- **Pulse:** This parameter can be one of the following values:
  - LL_RTC_CALIB_INSERTPULSE_NONE
  - LL_RTC_CALIB_INSERTPULSE_SET

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- Bit can be written only when RECALPF is set to 0 in RTC_ICSR

Reference Manual to LL API cross reference:

- RTC_CALR CALP LL_RTC_CAL_SetPulse

LL_RTC_CAL_IsPulseInserted

Function name

__STATIC_INLINE uint32_t LL_RTC_CAL_IsPulseInserted (const RTC_TypeDef * RTCx)

Function description

Check if one RTCCLK has been inserted or not every 2exp11 pulses (frequency increased by 488.5 ppm)

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_CALR CALP LL_RTC_CAL_IsPulseInserted

LL_RTC_CAL_SetPeriod

Function name

__STATIC_INLINE void LL_RTC_CAL_SetPeriod (RTC_TypeDef * RTCx, uint32_t Period)

Function description

Set the calibration cycle period.

Parameters

- **RTCx:** RTC Instance
- **Period:** This parameter can be one of the following values:
  - LL_RTC_CALIB_PERIOD_32SEC
  - LL_RTC_CALIB_PERIOD_16SEC
  - LL_RTC_CALIB_PERIOD_8SEC

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- Bit can be written only when RECALPF is set to 0 in RTC_ICSR

Reference Manual to LL API cross reference:

- RTC_CALR CALW8 LL_RTC_CAL_SetPeriod
- RTC_CALR CALW16 LL_RTC_CAL_SetPeriod

LL_RTC_CAL_GetPeriod

Function name

__STATIC_INLINE uint32_t LL_RTC_CAL_GetPeriod (const RTC_TypeDef * RTCx)

Function description

Get the calibration cycle period.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_CALIB_PERIOD_32SEC
  - LL_RTC_CALIB_PERIOD_16SEC
  - LL_RTC_CALIB_PERIOD_8SEC

Reference Manual to LL API cross reference:

- RTC_CALR CALW8 LL_RTC_CAL_GetPeriod
- RTC_CALR CALW16 LL_RTC_CAL_GetPeriod

LL_RTC_CAL_SetMinus

Function name

__STATIC_INLINE void LL_RTC_CAL_SetMinus (RTC_TypeDef * RTCx, uint32_t CalibMinus)

Function description

Set Calibration minus.

Parameters

- **RTCx:** RTC Instance
- **CalibMinus:** Value between Min_Data=0x00 and Max_Data=0x1FF

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- Bit can be written only when RECALPF is set to 0 in RTC_ICSR

Reference Manual to LL API cross reference:

- RTC_CALR CALM LL_RTC_CAL_SetMinus

LL_RTC_CAL_GetMinus

Function name

__STATIC_INLINE uint32_t LL_RTC_CAL_GetMinus (const RTC_TypeDef * RTCx)

Function description

Get Calibration minus.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data= 0x1FF

Reference Manual to LL API cross reference:

- RTC_CALR CALM LL_RTC_CAL_GetMinus

LL_RTC_CAL_LowPower_Enable

Function name

__STATIC_INLINE void LL_RTC_CAL_LowPower_Enable (RTC_TypeDef * RTCx)

Function description

Enable Calibration Low Power.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- Bit can be written only when RECALPF is set to 0

Reference Manual to LL API cross reference:

- RTC_CALR LPCAL LL_RTC_CAL_LowPower_Enable

LL_RTC_CAL_LowPower_Disable

Function name

__STATIC_INLINE void LL_RTC_CAL_LowPower_Disable (RTC_TypeDef * RTCx)

Function description

Disable Calibration Low Power.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- Bit can be written only when RECALPF is set to 0

Reference Manual to LL API cross reference:

- RTC_CALR LPCAL LL_RTC_CAL_LowPower_Disable

LL_RTC_CAL_LowPower_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_RTC_CAL_LowPower_IsEnabled (const RTC_TypeDef * RTCx)

Function description

Check if Calibration Low Power is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_CALR LPCAL LL_RTC_CAL_LowPower_IsEnabled

LL_RTC_IsActiveFlag_ITS

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITS (const RTC_TypeDef * RTCx)

Function description

Get Internal Time-stamp flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_SR ITSF LL_RTC_IsActiveFlag_ITS

LL_RTC_IsActiveFlag_RECALP

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_RECALP (const RTC_TypeDef * RTCx)

Function description

Get Recalibration pending Flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_ICSR RECALPF LL_RTC_IsActiveFlag_RECALP

LL_RTC_IsActiveFlag_TSOV

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TSOV (const RTC_TypeDef * RTCx)

Function description

Get Time-stamp overflow flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_SR TSOVF LL_RTC_IsActiveFlag_TSOV

LL_RTC_IsActiveFlag_TS

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TS (const RTC_TypeDef * RTCx)

Function description

Get Time-stamp flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_SR TSF LL_RTC_IsActiveFlag_TS

LL_RTC_IsActiveFlag_WUT

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_WUT (const RTC_TypeDef * RTCx)

Function description

Get Wakeup timer flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_SR WUTF LL_RTC_IsActiveFlag_WUT

LL_RTC_IsActiveFlag_ALRB

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ALRB (const RTC_TypeDef * RTCx)

Function description

Get Alarm B flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_SR ALRBF LL_RTC_IsActiveFlag_ALRB

LL_RTC_IsActiveFlag_ALRA

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ALRA (const RTC_TypeDef * RTCx)

Function description

Get Alarm A flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_SR ALRAF LL_RTC_IsActiveFlag_ALRA

LL_RTC_IsActiveFlag_SSRU

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_SSRU (const RTC_TypeDef * RTCx)

Function description

Get SSR Underflow flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_SR SSRUF LL_RTC_IsActiveFlag_SSRU

LL_RTC_ClearFlag_ITS

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_ITS (RTC_TypeDef * RTCx)

Function description

Clear Internal Time-stamp flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_SCR CITSF LL_RTC_ClearFlag_ITS

LL_RTC_ClearFlag_TSOV

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_TSOV (RTC_TypeDef * RTCx)

Function description

Clear Time-stamp overflow flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_SCR CTSOVF LL_RTC_ClearFlag_TSOV

LL_RTC_ClearFlag_TS

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_TS (RTC_TypeDef * RTCx)

Function description

Clear Time-stamp flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_SCR CTSF LL_RTC_ClearFlag_TS

LL_RTC_ClearFlag_WUT

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_WUT (RTC_TypeDef * RTCx)

Function description

Clear Wakeup timer flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_SCR CWUTF LL_RTC_ClearFlag_WUT

LL_RTC_ClearFlag_ALRB

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_ALRB (RTC_TypeDef * RTCx)

Function description

Clear Alarm B flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_SCR CALRBF LL_RTC_ClearFlag_ALRB

LL_RTC_ClearFlag_ALRA

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_ALRA (RTC_TypeDef * RTCx)

Function description

Clear Alarm A flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_SCR CALRAF LL_RTC_ClearFlag_ALRA

LL_RTC_ClearFlag_SSRU

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_SSRU (RTC_TypeDef * RTCx)

Function description

Clear SSR Underflow flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_SCR CSSRUF LL_RTC_ClearFlag_SSRU

LL_RTC_IsActiveFlag_INIT

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_INIT (const RTC_TypeDef * RTCx)

Function description

Get Initialization flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_ICSR INITF LL_RTC_IsActiveFlag_INIT

LL_RTC_IsActiveFlag_RS

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_RS (const RTC_TypeDef * RTCx)

Function description

Get Registers synchronization flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_ICSR RSF LL_RTC_IsActiveFlag_RS

LL_RTC_ClearFlag_RS

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_RS (RTC_TypeDef * RTCx)

Function description

Clear Registers synchronization flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_ICSR RSF LL_RTC_ClearFlag_RS

LL_RTC_IsActiveFlag_INITS

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_INITS (const RTC_TypeDef * RTCx)

Function description

Get Initialization status flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_ICSR INITS LL_RTC_IsActiveFlag_INITS

LL_RTC_IsActiveFlag_SHP

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_SHP (const RTC_TypeDef * RTCx)

Function description

Get Shift operation pending flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_ICSR SHPF LL_RTC_IsActiveFlag_SHP

LL_RTC_IsActiveFlag_WUTW

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_WUTW (const RTC_TypeDef * RTCx)

Function description

Get Wakeup timer write flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_ICSR WUTWF LL_RTC_IsActiveFlag_WUTW

LL_RTC_IsActiveFlag_ALRAM

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ALRAM (const RTC_TypeDef * RTCx)

Function description

Get Alarm A masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_MISR ALRAMF LL_RTC_IsActiveFlag_ALRAM

LL_RTC_IsActiveFlag_SSRUM

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_SSRUM (const RTC_TypeDef * RTCx)

Function description

Get SSR Underflow masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_MISR SSRUMF LL_RTC_IsActiveFlag_SSRUM

LL_RTC_IsActiveFlag_ALRBM

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ALRBM (const RTC_TypeDef * RTCx)

Function description

Get Alarm B masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_MISR ALRBMF LL_RTC_IsActiveFlag_ALRBM

LL_RTC_IsActiveFlag_WUTM

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_WUTM (const RTC_TypeDef * RTCx)

Function description

Get Wakeup timer masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_MISR WUTMF LL_RTC_IsActiveFlag_WUTM

LL_RTC_IsActiveFlag_TSM

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TSM (const RTC_TypeDef * RTCx)

Function description

Get Time-stamp masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_MISR TSMF LL_RTC_IsActiveFlag_TSM

LL_RTC_IsActiveFlag_TSOVM

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TSOVM (const RTC_TypeDef * RTCx)

Function description

Get Time-stamp overflow masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_MISR TSOVMF LL_RTC_IsActiveFlag_TSOVM

LL_RTC_IsActiveFlag_ITSM

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITSM (const RTC_TypeDef * RTCx)

Function description

Get Internal Time-stamp masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_MISR ITSMF LL_RTC_IsActiveFlag_ITSM

LL_RTC_IsActiveFlag_TAMP1

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TAMP1 (const RTC_TypeDef * RTCx)

Function description

Get tamper 1 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_SR TAMP1F LL_RTC_IsActiveFlag_TAMP1

LL_RTC_IsActiveFlag_TAMP2

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TAMP2 (const RTC_TypeDef * RTCx)

Function description

Get tamper 2 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_SR TAMP2F LL_RTC_IsActiveFlag_TAMP2

LL_RTC_IsActiveFlag_TAMP3

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TAMP3 (const RTC_TypeDef * RTCx)

Function description

Get tamper 3 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_SR TAMP3F LL_RTC_IsActiveFlag_TAMP3

LL_RTC_IsActiveFlag_TAMP4

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TAMP4 (const RTC_TypeDef * RTCx)

Function description

Get tamper 4 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_SR TAMP4F LL_RTC_IsActiveFlag_TAMP4

LL_RTC_IsActiveFlag_TAMP5

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TAMP5 (const RTC_TypeDef * RTCx)

Function description

Get tamper 5 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_SR TAMP5F LL_RTC_IsActiveFlag_TAMP5

LL_RTC_IsActiveFlag_TAMP6

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TAMP6 (const RTC_TypeDef * RTCx)

Function description

Get tamper 6 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_SR TAMP6F LL_RTC_IsActiveFlag_TAMP6

LL_RTC_IsActiveFlag_TAMP7

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TAMP7 (const RTC_TypeDef * RTCx)

Function description

Get tamper 7 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_SR TAMP7F LL_RTC_IsActiveFlag_TAMP7

LL_RTC_IsActiveFlag_ITAMP1

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITAMP1 (const RTC_TypeDef * RTCx)

Function description

Get internal tamper 1 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_SR ITAMP1F LL_RTC_IsActiveFlag_ITAMP1

LL_RTC_IsActiveFlag_ITAMP2

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITAMP2 (const RTC_TypeDef * RTCx)

Function description

Get internal tamper 2 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_SR ITAMP2F LL_RTC_IsActiveFlag_ITAMP2

LL_RTC_IsActiveFlag_ITAMP3

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITAMP3 (const RTC_TypeDef * RTCx)

Function description

Get internal tamper 3 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_SR ITAMP3F LL_RTC_IsActiveFlag_ITAMP3

LL_RTC_IsActiveFlag_ITAMP4

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITAMP4 (const RTC_TypeDef * RTCx)

Function description

Get internal tamper 4 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_SR ITAMP4F LL_RTC_IsActiveFlag_ITAMP4

LL_RTC_IsActiveFlag_ITAMP5

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITAMP5 (const RTC_TypeDef * RTCx)

Function description

Get internal tamper 5 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_SR ITAMP5F LL_RTC_IsActiveFlag_ITAMP5

LL_RTC_IsActiveFlag_ITAMP6

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITAMP6 (const RTC_TypeDef * RTCx)

Function description

Get internal tamper 6 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_SR ITAMP6F LL_RTC_IsActiveFlag_ITAMP6

LL_RTC_IsActiveFlag_ITAMP7

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITAMP7 (const RTC_TypeDef * RTCx)

Function description

Get internal tamper 7 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_SR ITAMP7F LL_RTC_IsActiveFlag_ITAMP7

LL_RTC_IsActiveFlag_ITAMP8

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITAMP8 (const RTC_TypeDef * RTCx)

Function description

Get internal tamper 8 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_SR ITAMP8F LL_RTC_IsActiveFlag_ITAMP8

LL_RTC_IsActiveFlag_ITAMP9

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITAMP9 (const RTC_TypeDef * RTCx)

Function description

Get internal tamper 9 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_SR ITAMP9F LL_RTC_IsActiveFlag_ITAMP9

LL_RTC_IsActiveFlag_ITAMP11

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITAMP11 (const RTC_TypeDef * RTCx)

Function description

Get internal tamper 11 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_SR ITAMP11F LL_RTC_IsActiveFlag_ITAMP11

LL_RTC_IsActiveFlag_TAMP1M

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TAMP1M (const RTC_TypeDef * RTCx)

Function description

Get tamper 1 interrupt masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_MISR TAMP1MF LL_RTC_IsActiveFlag_TAMP1M

LL_RTC_IsActiveFlag_TAMP2M

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TAMP2M (const RTC_TypeDef * RTCx)

Function description

Get tamper 2 interrupt masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_MISR TAMP2MF LL_RTC_IsActiveFlag_TAMP2M

LL_RTC_IsActiveFlag_TAMP3M

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TAMP3M (const RTC_TypeDef * RTCx)

Function description

Get tamper 3 interrupt masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_MISR TAMP3MF LL_RTC_IsActiveFlag_TAMP3M

LL_RTC_IsActiveFlag_TAMP4M

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TAMP4M (const RTC_TypeDef * RTCx)

Function description

Get tamper 4 interrupt masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_MISR TAMP4MF LL_RTC_IsActiveFlag_TAMP4M

LL_RTC_IsActiveFlag_TAMP5M

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TAMP5M (const RTC_TypeDef * RTCx)

Function description

Get tamper 5 interrupt masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_MISR TAMP5MF LL_RTC_IsActiveFlag_TAMP5M

LL_RTC_IsActiveFlag_TAMP6M

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TAMP6M (const RTC_TypeDef * RTCx)

Function description

Get tamper 6 interrupt masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_MISR TAMP6MF LL_RTC_IsActiveFlag_TAMP6M

LL_RTC_IsActiveFlag_TAMP7M

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TAMP7M (const RTC_TypeDef * RTCx)

Function description

Get tamper 7 interrupt masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_MISR TAMP7MF LL_RTC_IsActiveFlag_TAMP7M

LL_RTC_IsActiveFlag_ITAMP1M

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITAMP1M (const RTC_TypeDef * RTCx)

Function description

Get internal tamper 1 interrupt masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_MISR ITAMP1MF LL_RTC_IsActiveFlag_ITAMP1M

LL_RTC_IsActiveFlag_ITAMP2M

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITAMP2M (const RTC_TypeDef * RTCx)

Function description

Get internal tamper 2 interrupt masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_MISR ITAMP2MF LL_RTC_IsActiveFlag_ITAMP2M

LL_RTC_IsActiveFlag_ITAMP3M

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITAMP3M (const RTC_TypeDef * RTCx)

Function description

Get internal tamper 3 interrupt masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_MISR ITAMP3MF LL_RTC_IsActiveFlag_ITAMP3M

LL_RTC_IsActiveFlag_ITAMP4M

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITAMP4M (const RTC_TypeDef * RTCx)

Function description

Get internal tamper 4 interrupt masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_MISR ITAMP4MF LL_RTC_IsActiveFlag_ITAMP4M

LL_RTC_IsActiveFlag_ITAMP5M

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITAMP5M (const RTC_TypeDef * RTCx)

Function description

Get internal tamper 5 interrupt masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_MISR ITAMP5MF LL_RTC_IsActiveFlag_ITAMP5M

LL_RTC_IsActiveFlag_ITAMP6M

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITAMP6M (const RTC_TypeDef * RTCx)

Function description

Get internal tamper 6 interrupt masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_MISR ITAMP6MF LL_RTC_IsActiveFlag_ITAMP6M

LL_RTC_IsActiveFlag_ITAMP7M

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITAMP7M (const RTC_TypeDef * RTCx)

Function description

Get internal tamper 7 interrupt masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_MISR ITAMP7MF LL_RTC_IsActiveFlag_ITAMP7M

LL_RTC_IsActiveFlag_ITAMP8M

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITAMP8M (const RTC_TypeDef * RTCx)

Function description

Get internal tamper 8 interrupt masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_MISR ITAMP8MF LL_RTC_IsActiveFlag_ITAMP8M

LL_RTC_IsActiveFlag_ITAMP9M

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITAMP9M (const RTC_TypeDef * RTCx)

Function description

Get internal tamper 9 interrupt masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_MISR ITAMP9MF LL_RTC_IsActiveFlag_ITAMP9M

LL_RTC_IsActiveFlag_ITAMP11M

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITAMP11M (const RTC_TypeDef * RTCx)

Function description

Get internal tamper 11 interrupt masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_MISR ITAMP11MF LL_RTC_IsActiveFlag_ITAMP11M

LL_RTC_ClearFlag_TAMP1

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_TAMP1 (const RTC_TypeDef * RTCx)

Function description

Clear tamper 1 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_SCR CTAMP1F LL_RTC_ClearFlag_TAMP1

LL_RTC_ClearFlag_TAMP2

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_TAMP2 (const RTC_TypeDef * RTCx)

Function description

Clear tamper 2 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_SCR CTAMP2F LL_RTC_ClearFlag_TAMP2

LL_RTC_ClearFlag_TAMP3

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_TAMP3 (const RTC_TypeDef * RTCx)

Function description

Clear tamper 3 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_SCR CTAMP3F LL_RTC_ClearFlag_TAMP3

LL_RTC_ClearFlag_TAMP4

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_TAMP4 (const RTC_TypeDef * RTCx)

Function description

Clear tamper 4 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_SCR CTAMP4F LL_RTC_ClearFlag_TAMP4

LL_RTC_ClearFlag_TAMP5

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_TAMP5 (const RTC_TypeDef * RTCx)

Function description

Clear tamper 5 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_SCR CTAMP5F LL_RTC_ClearFlag_TAMP5

LL_RTC_ClearFlag_TAMP6

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_TAMP6 (const RTC_TypeDef * RTCx)

Function description

Clear tamper 6 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_SCR CTAMP6F LL_RTC_ClearFlag_TAMP6

LL_RTC_ClearFlag_TAMP7

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_TAMP7 (const RTC_TypeDef * RTCx)

Function description

Clear tamper 7 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_SCR CTAMP7F LL_RTC_ClearFlag_TAMP7

LL_RTC_ClearFlag_ITAMP1

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_ITAMP1 (const RTC_TypeDef * RTCx)

Function description

Clear internal tamper 1 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_SCR CITAMP1F LL_RTC_ClearFlag_ITAMP1

LL_RTC_ClearFlag_ITAMP2

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_ITAMP2 (const RTC_TypeDef * RTCx)

Function description

Clear internal tamper 2 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_SCR CITAMP2F LL_RTC_ClearFlag_ITAMP2

LL_RTC_ClearFlag_ITAMP3

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_ITAMP3 (const RTC_TypeDef * RTCx)

Function description

Clear internal tamper 3 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_SCR CITAMP3F LL_RTC_ClearFlag_ITAMP3

LL_RTC_ClearFlag_ITAMP4

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_ITAMP4 (const RTC_TypeDef * RTCx)

Function description

Clear internal tamper 4 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_SCR CITAMP4F LL_RTC_ClearFlag_ITAMP4

LL_RTC_ClearFlag_ITAMP5

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_ITAMP5 (const RTC_TypeDef * RTCx)

Function description

Clear internal tamper 5 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_SCR CITAMP5F LL_RTC_ClearFlag_ITAMP5

LL_RTC_ClearFlag_ITAMP6

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_ITAMP6 (const RTC_TypeDef * RTCx)

Function description

Clear internal tamper 6 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_SCR CITAMP6F LL_RTC_ClearFlag_ITAMP6

LL_RTC_ClearFlag_ITAMP7

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_ITAMP7 (const RTC_TypeDef * RTCx)

Function description

Clear internal tamper 7 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_SCR CITAMP7F LL_RTC_ClearFlag_ITAMP7

LL_RTC_ClearFlag_ITAMP8

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_ITAMP8 (const RTC_TypeDef * RTCx)

Function description

Clear internal tamper 8 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_SCR CITAMP8F LL_RTC_ClearFlag_ITAMP8

LL_RTC_ClearFlag_ITAMP9

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_ITAMP9 (const RTC_TypeDef * RTCx)

Function description

Clear internal tamper 9 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_SCR CITAMP9F LL_RTC_ClearFlag_ITAMP9

LL_RTC_ClearFlag_ITAMP11

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_ITAMP11 (const RTC_TypeDef * RTCx)

Function description

Clear internal tamper 11 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_SCR CITAMP11F LL_RTC_ClearFlag_ITAMP11

LL_RTC_SetRtcSecure

Function name

__STATIC_INLINE void LL_RTC_SetRtcSecure (RTC_TypeDef * RTCx, uint32_t rtcSecure)

Function description

Set RTC secure level.

Parameters

- **RTCx:** RTC Instance
- **rtcSecure:** This parameter can be a combination of the following values:
  - LL_RTC_SECURE_FULL_YES
  - LL_RTC_SECURE_FULL_NO
  - LL_RTC_SECURE_FEATURE_INIT
  - LL_RTC_SECURE_FEATURE_CAL
  - LL_RTC_SECURE_FEATURE_TS
  - LL_RTC_SECURE_FEATURE_WUT
  - LL_RTC_SECURE_FEATURE_ALRA
  - LL_RTC_SECURE_FEATURE_ALRB

Return values

- **None:**

Notes

- secure features are relevant if LL_RTC_SECURE_FULL_NO.

Reference Manual to LL API cross reference:

- RTC_SECCFGR SEC LL_RTC_SetRtcSecure
- RTC_SECCFGR INITSEC LL_RTC_SetRtcSecure
- RTC_SECCFGR CALSEC LL_RTC_SetRtcSecure
- RTC_SECCFGR TSSEC LL_RTC_SetRtcSecure
- RTC_SECCFGR WUTSEC LL_RTC_SetRtcSecure
- RTC_SECCFGR ALRASEC LL_RTC_SetRtcSecure
- RTC_SECCFGR ALRBSEC LL_RTC_SetRtcSecure

LL_RTC_GetRtcSecure

Function name

__STATIC_INLINE uint32_t LL_RTC_GetRtcSecure (const RTC_TypeDef * RTCx)

Function description

Get RTC secure level.

Parameters

- **RTCx:** RTC Instance

Return values

- **Combination:** of the following values:
  - LL_RTC_SECURE_FULL_YES
  - LL_RTC_SECURE_FULL_NO
  - LL_RTC_SECURE_FEATURE_INIT
  - LL_RTC_SECURE_FEATURE_CAL
  - LL_RTC_SECURE_FEATURE_TS
  - LL_RTC_SECURE_FEATURE_WUT
  - LL_RTC_SECURE_FEATURE_ALRA
  - LL_RTC_SECURE_FEATURE_ALRB

Notes

- Secure features is relevant if LL_RTC_SECURE_FULL_NO.

Reference Manual to LL API cross reference:

- RTC_SECCFGR SEC LL_RTC_SetRtcSecure
- RTC_SECCFGR INISEC LL_RTC_SetRtcSecure
- RTC_SECCFGR CALSEC LL_RTC_SetRtcSecure
- RTC_SECCFGR TSSEC LL_RTC_SetRtcSecure
- RTC_SECCFGR WUTSEC LL_RTC_SetRtcSecure
- RTC_SECCFGR ALRASEC LL_RTC_SetRtcSecure
- RTC_SECCFGR ALRBSEC LL_RTC_SetRtcSecure

LL_RTC_SetTampSecure

Function name

__STATIC_INLINE void LL_RTC_SetTampSecure (const RTC_TypeDef * RTCx, uint32_t tampSecure)

Function description

Set TAMPER secure level.

Parameters

- **RTCx:** RTC Instance
- **tampSecure:** This parameter can be one of the following values:
  - LL_TAMP_SECURE_FULL_YES
  - LL_TAMP_SECURE_FULL_NO

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_SECCFGR TAMPSEC LL_RTC_SetTampSecure

LL_RTC_GetTampSecure

Function name

__STATIC_INLINE uint32_t LL_RTC_GetTampSecure (const RTC_TypeDef * RTCx)

Function description

Get TAMPER secure level.

Parameters

- **RTCx:** RTC Instance

Return values

- **This:** parameter can be one of the following values:
  - LL_TAMP_SECURE_FULL_YES
  - LL_TAMP_SECURE_FULL_NO

Reference Manual to LL API cross reference:

- TAMP_SECCFGR TAMPSEC LL_RTC_GetTampSecure

LL_RTC_SetRtcPrivilege

Function name

__STATIC_INLINE void LL_RTC_SetRtcPrivilege (RTC_TypeDef * RTCx, uint32_t rtcPrivilege)

Function description

Set RTC privilege level.

Parameters

- **RTCx:** RTC Instance
- **rtcPrivilege:** This parameter can be a combination of the following values:
  - LL_RTC_PRIVILEGE_FULL_YES
  - LL_RTC_PRIVILEGE_FULL_NO
  - LL_RTC_PRIVILEGE_FEATURE_INIT
  - LL_RTC_PRIVILEGE_FEATURE_CAL
  - LL_RTC_PRIVILEGE_FEATURE_TS
  - LL_RTC_PRIVILEGE_FEATURE_WUT
  - LL_RTC_PRIVILEGE_FEATURE_ALRA
  - LL_RTC_PRIVILEGE_FEATURE_ALRB

Return values

- **None:**

Notes

- Privilege features are relevant if LL_RTC_PRIVILEGE_FULL_NO.

Reference Manual to LL API cross reference:

- RTC_PRIVCFGR PRIV LL_RTC_SetRtcPrivilege
- RTC_PRIVCFGR INITPRIV LL_RTC_SetRtcPrivilege
- RTC_PRIVCFGR CALPRIV LL_RTC_SetRtcPrivilege
- RTC_PRIVCFGR TSPRIV LL_RTC_SetRtcPrivilege
- RTC_PRIVCFGR WUTPRIV LL_RTC_SetRtcPrivilege
- RTC_PRIVCFGR ALRAPRIV LL_RTC_SetRtcPrivilege
- RTC_PRIVCFGR ALRBPRIV LL_RTC_SetRtcPrivilege

LL_RTC_GetRtcPrivilege

Function name

__STATIC_INLINE uint32_t LL_RTC_GetRtcPrivilege (const RTC_TypeDef * RTCx)

Function description

Get RTC privilege level.

Parameters

- **RTCx:** RTC Instance

Return values

- **Combination:** of the following values:
  - LL_RTC_PRIVILEGE_FULL_YES
  - LL_RTC_PRIVILEGE_FULL_NO
  - LL_RTC_PRIVILEGE_FEATURE_INIT
  - LL_RTC_PRIVILEGE_FEATURE_CAL
  - LL_RTC_PRIVILEGE_FEATURE_TS
  - LL_RTC_PRIVILEGE_FEATURE_WUT
  - LL_RTC_PRIVILEGE_FEATURE_ALRA
  - LL_RTC_PRIVILEGE_FEATURE_ALRB

Notes

- Privilege features are relevant if LL_RTC_PRIVILEGE_FULL_NO.

Reference Manual to LL API cross reference:

- RTC_PRIVCFGR PRIV LL_RTC_SetRtcPrivilege
- RTC_PRIVCFGR INITPRIV LL_RTC_SetRtcPrivilege
- RTC_PRIVCFGR CALPRIV LL_RTC_SetRtcPrivilege
- RTC_PRIVCFGR TSPRIV LL_RTC_SetRtcPrivilege
- RTC_PRIVCFGR WUTPRIV LL_RTC_SetRtcPrivilege
- RTC_PRIVCFGR ALRAPRIV LL_RTC_SetRtcPrivilege
- RTC_PRIVCFGR ALRBPRIV LL_RTC_SetRtcPrivilege

LL_RTC_SetTampPrivilege

Function name

__STATIC_INLINE void LL_RTC_SetTampPrivilege (const RTC_TypeDef * RTCx, uint32_t tampPrivilege)

Function description

Set TAMPER privilege level.

Parameters

- **RTCx:** RTC Instance
- **tampPrivilege:** This parameter can be one of the following values:
  - LL_TAMP_PRIVILEGE_FULL_YES
  - LL_TAMP_PRIVILEGE_FULL_NO

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_PRIVCFGR TAMPPRIV LL_RTC_SetTampPrivilege

LL_RTC_GetTampPrivilege

Function name

__STATIC_INLINE uint32_t LL_RTC_GetTampPrivilege (const RTC_TypeDef * RTCx)

Function description

Get TAMPER privilege level.

Parameters

- **RTCx:** RTC Instance

Return values

- **This:** parameter can be one of the following values:
  - LL_TAMP_PRIVILEGE_FULL_YES
  - LL_TAMP_PRIVILEGE_FULL_NO

Reference Manual to LL API cross reference:

- TAMP_PRIVCFGR TAMPPRIV LL_RTC_GetTampPrivilege

LL_RTC_SetBackupRegisterPrivilege

Function name

__STATIC_INLINE void LL_RTC_SetBackupRegisterPrivilege (const RTC_TypeDef * RTCx, uint32_t bckupRegisterPrivilege)

Function description

Set Backup Registers privilege level.

Parameters

- **RTCx:** RTC Instance
- **bckupRegisterPrivilege:** This parameter can be one of the following values:
  - LL_RTC_PRIVILEGE_BKUP_ZONE_NONE
  - LL_RTC_PRIVILEGE_BKUP_ZONE_1
  - LL_RTC_PRIVILEGE_BKUP_ZONE_2
  - LL_RTC_PRIVILEGE_BKUP_ZONE_ALL

Return values

- **None:**

Notes

- bckupRegisterPrivilege is only writable in secure mode or if trustzone is disabled

Reference Manual to LL API cross reference:

- TAMP_PRIVCFGR BKPWPRIV LL_RTC_SetBackupRegisterPrivilege
- TAMP_PRIVCFGR BKPRWPRIV LL_RTC_SetBackupRegisterPrivilege

LL_RTC_GetBackupRegisterPrivilege

Function name

__STATIC_INLINE uint32_t LL_RTC_GetBackupRegisterPrivilege (const RTC_TypeDef * RTCx)

Function description

Get Backup Registers privilege level.

Parameters

- **RTCx:** RTC Instance

Return values

- **This:** parameter can be one of the following values:
  - LL_RTC_PRIVILEGE_BKUP_ZONE_NONE
  - LL_RTC_PRIVILEGE_BKUP_ZONE_1
  - LL_RTC_PRIVILEGE_BKUP_ZONE_2
  - LL_RTC_PRIVILEGE_BKUP_ZONE_ALL

Reference Manual to LL API cross reference:

- TAMP_PRIVCFGR BKPWPRIV LL_RTC_GetBackupRegisterPrivilege
- TAMP_PRIVCFGR BKPRWPRIV LL_RTC_GetBackupRegisterPrivilege

LL_RTC_SetBackupRegProtection

Function name

__STATIC_INLINE void LL_RTC_SetBackupRegProtection (const RTC_TypeDef * RTCx, uint32_t startZone2, uint32_t startZone3)

Function description

Set Backup registers protection level.

Parameters

- **RTCx:** RTC Instance
- **startZone2:** This parameter can be one of the following values:
  - LL_RTC_BKP_DR0
  - LL_RTC_BKP_DR1
  - LL_RTC_BKP_DR2
  - LL_RTC_BKP_DR3
  - LL_RTC_BKP_DR4
  - LL_RTC_BKP_DRx ...
- **startZone3:** This parameter can be one of the following values:
  - LL_RTC_BKP_DR0
  - LL_RTC_BKP_DR1
  - LL_RTC_BKP_DR2
  - LL_RTC_BKP_DR3
  - LL_RTC_BKP_DR4
  - LL_RTC_BKP_DRx ...

Return values

- **None:**

Notes

- Zone 1 : read protection write protection
- Zone 2 : read non-protection write protection
- Zone 3 : read non-protection write non-protection
- zone 1 : start from 0 to startZone2 start value
- zone 2 : start from startZone2 start value to startZone3 start value
- zone 3 : start from to startZone3 to the end of BACKUPREG
- Warning : this parameter is only writable in secure mode or if trustzone is disabled

Reference Manual to LL API cross reference:

- TAMP_SECCFGR BKPWSEC LL_RTC_SetBackupRegProtection
- TAMP_SECCFGR BKPRWSEC LL_RTC_SetBackupRegProtection

LL_RTC_GetBackupRegProtectionStartZone2

Function name

__STATIC_INLINE uint32_t LL_RTC_GetBackupRegProtectionStartZone2 (const RTC_TypeDef * RTCx)

Function description

Get Backup registers protection level start zone 2.

Parameters

- **RTCx:** RTC Instance

Return values

- **Start:** zone 2

Notes

- Zone 1 : read protection write protection
- Zone 2 : read non-protection/non-privile write protection
- Zone 3 : read non-protection write non-protection

Reference Manual to LL API cross reference:

- TAMP_SECCFGR BKPRWSEC LL_RTC_GetBackupRegProtectionStartZone2

LL_RTC_GetBackupRegProtectionStartZone3

Function name

__STATIC_INLINE uint32_t LL_RTC_GetBackupRegProtectionStartZone3 (const RTC_TypeDef * RTCx)

Function description

Get Backup registers protection level start zone 3.

Parameters

- **RTCx:** RTC Instance

Return values

- **Start:** zone 2

Notes

- Zone 1 : read protection write protection
- Zone 2 : read non-protection write protection
- Zone 3 : read non-protection write non-protection

Reference Manual to LL API cross reference:

- TAMP_SECCFGR BKPWSEC LL_RTC_GetBackupRegProtectionStartZone3

LL_RTC_EnableIT_TS

Function name

__STATIC_INLINE void LL_RTC_EnableIT_TS (RTC_TypeDef * RTCx)

Function description

Enable Time-stamp interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR TSIE LL_RTC_EnableIT_TS

LL_RTC_DisableIT_TS

Function name

__STATIC_INLINE void LL_RTC_DisableIT_TS (RTC_TypeDef * RTCx)

Function description

Disable Time-stamp interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR TSIE LL_RTC_DisableIT_TS

LL_RTC_EnableIT_WUT

Function name

__STATIC_INLINE void LL_RTC_EnableIT_WUT (RTC_TypeDef * RTCx)

Function description

Enable Wakeup timer interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR WUTIE LL_RTC_EnableIT_WUT

LL_RTC_DisableIT_WUT

Function name

__STATIC_INLINE void LL_RTC_DisableIT_WUT (RTC_TypeDef * RTCx)

Function description

Disable Wakeup timer interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR WUTIE LL_RTC_DisableIT_WUT

LL_RTC_EnableIT_ALRB

Function name

__STATIC_INLINE void LL_RTC_EnableIT_ALRB (RTC_TypeDef * RTCx)

Function description

Enable Alarm B interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR ALRBIE LL_RTC_EnableIT_ALRB

LL_RTC_DisableIT_ALRB

Function name

__STATIC_INLINE void LL_RTC_DisableIT_ALRB (RTC_TypeDef * RTCx)

Function description

Disable Alarm B interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR ALRBIE LL_RTC_DisableIT_ALRB

LL_RTC_EnableIT_ALRA

Function name

__STATIC_INLINE void LL_RTC_EnableIT_ALRA (RTC_TypeDef * RTCx)

Function description

Enable Alarm A interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR ALRAIE LL_RTC_EnableIT_ALRA

LL_RTC_DisableIT_ALRA

Function name

__STATIC_INLINE void LL_RTC_DisableIT_ALRA (RTC_TypeDef * RTCx)

Function description

Disable Alarm A interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR ALRAIE LL_RTC_DisableIT_ALRA

LL_RTC_EnableIT_SSRU

Function name

__STATIC_INLINE void LL_RTC_EnableIT_SSRU (RTC_TypeDef * RTCx)

Function description

Enable SSR Underflow interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR SSRUIE LL_RTC_EnableIT_SSRU

LL_RTC_DisableIT_SSRU

Function name

__STATIC_INLINE void LL_RTC_DisableIT_SSRU (RTC_TypeDef * RTCx)

Function description

Disable SSR Underflow interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR SSRUIE LL_RTC_DisableIT_SSRU

LL_RTC_IsEnabledIT_TS

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_TS (const RTC_TypeDef * RTCx)

Function description

Check if Time-stamp interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_CR TSIE LL_RTC_IsEnabledIT_TS

LL_RTC_IsEnabledIT_WUT

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_WUT (const RTC_TypeDef * RTCx)

Function description

Check if Wakeup timer interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_CR WUTIE LL_RTC_IsEnabledIT_WUT

LL_RTC_IsEnabledIT_ALRB

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_ALRB (const RTC_TypeDef * RTCx)

Function description

Check if Alarm B interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_CR ALRBIE LL_RTC_IsEnabledIT_ALRB

LL_RTC_IsEnabledIT_ALRA

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_ALRA (const RTC_TypeDef * RTCx)

Function description

Check if Alarm A interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_CR ALRAIE LL_RTC_IsEnabledIT_ALRA

LL_RTC_IsEnabledIT_SSRU

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_SSRU (const RTC_TypeDef * RTCx)

Function description

Check if SSR Underflow interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_CR SSRUIE LL_RTC_IsEnabledIT_SSRU

LL_RTC_EnableIT_TAMP1

Function name

__STATIC_INLINE void LL_RTC_EnableIT_TAMP1 (const RTC_TypeDef * RTCx)

Function description

Enable tamper 1 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER TAMP1IE LL_RTC_EnableIT_TAMP1

LL_RTC_DisableIT_TAMP1

Function name

__STATIC_INLINE void LL_RTC_DisableIT_TAMP1 (const RTC_TypeDef * RTCx)

Function description

Disable tamper 1 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER TAMP1IE LL_RTC_DisableIT_TAMP1

LL_RTC_EnableIT_TAMP2

Function name

__STATIC_INLINE void LL_RTC_EnableIT_TAMP2 (const RTC_TypeDef * RTCx)

Function description

Enable tamper 2 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER TAMP2IE LL_RTC_EnableIT_TAMP2

LL_RTC_DisableIT_TAMP2

Function name

__STATIC_INLINE void LL_RTC_DisableIT_TAMP2 (const RTC_TypeDef * RTCx)

Function description

Disable tamper 2 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER TAMP2IE LL_RTC_DisableIT_TAMP2

LL_RTC_EnableIT_TAMP3

Function name

__STATIC_INLINE void LL_RTC_EnableIT_TAMP3 (const RTC_TypeDef * RTCx)

Function description

Enable tamper 3 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER TAMP3IE LL_RTC_EnableIT_TAMP3

LL_RTC_DisableIT_TAMP3

Function name

__STATIC_INLINE void LL_RTC_DisableIT_TAMP3 (const RTC_TypeDef * RTCx)

Function description

Disable tamper 3 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER TAMP3IE LL_RTC_DisableIT_TAMP3

LL_RTC_EnableIT_TAMP4

Function name

__STATIC_INLINE void LL_RTC_EnableIT_TAMP4 (const RTC_TypeDef * RTCx)

Function description

Enable tamper 4 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER TAMP4IE LL_RTC_EnableIT_TAMP4

LL_RTC_DisableIT_TAMP4

Function name

__STATIC_INLINE void LL_RTC_DisableIT_TAMP4 (const RTC_TypeDef * RTCx)

Function description

Disable tamper 4 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER TAMP4IE LL_RTC_DisableIT_TAMP4

LL_RTC_EnableIT_TAMP5

Function name

__STATIC_INLINE void LL_RTC_EnableIT_TAMP5 (const RTC_TypeDef * RTCx)

Function description

Enable tamper 5 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER TAMP5IE LL_RTC_EnableIT_TAMP5

LL_RTC_DisableIT_TAMP5

Function name

__STATIC_INLINE void LL_RTC_DisableIT_TAMP5 (const RTC_TypeDef * RTCx)

Function description

Disable tamper 5 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER TAMP5IE LL_RTC_DisableIT_TAMP5

LL_RTC_EnableIT_TAMP6

Function name

__STATIC_INLINE void LL_RTC_EnableIT_TAMP6 (const RTC_TypeDef * RTCx)

Function description

Enable tamper 6 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER TAMP6IE LL_RTC_EnableIT_TAMP6

LL_RTC_DisableIT_TAMP6

Function name

__STATIC_INLINE void LL_RTC_DisableIT_TAMP6 (const RTC_TypeDef * RTCx)

Function description

Disable tamper 6 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER TAMP6IE LL_RTC_DisableIT_TAMP6

LL_RTC_EnableIT_TAMP7

Function name

__STATIC_INLINE void LL_RTC_EnableIT_TAMP7 (const RTC_TypeDef * RTCx)

Function description

Enable tamper 7 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER TAMP7IE LL_RTC_EnableIT_TAMP7

LL_RTC_DisableIT_TAMP7

Function name

__STATIC_INLINE void LL_RTC_DisableIT_TAMP7 (const RTC_TypeDef * RTCx)

Function description

Disable tamper 7 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER TAMP7IE LL_RTC_DisableIT_TAMP7

LL_RTC_EnableIT_ITAMP1

Function name

__STATIC_INLINE void LL_RTC_EnableIT_ITAMP1 (const RTC_TypeDef * RTCx)

Function description

Enable internal tamper 1 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP1IE LL_RTC_EnableIT_ITAMP1

LL_RTC_DisableIT_ITAMP1

Function name

__STATIC_INLINE void LL_RTC_DisableIT_ITAMP1 (const RTC_TypeDef * RTCx)

Function description

Disable internal tamper 1 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP1IE LL_RTC_DisableIT_ITAMP1

LL_RTC_EnableIT_ITAMP2

Function name

__STATIC_INLINE void LL_RTC_EnableIT_ITAMP2 (const RTC_TypeDef * RTCx)

Function description

Enable internal tamper 2 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP2IE LL_RTC_EnableIT_ITAMP2

LL_RTC_DisableIT_ITAMP2

Function name

__STATIC_INLINE void LL_RTC_DisableIT_ITAMP2 (const RTC_TypeDef * RTCx)

Function description

Disable internal tamper 2 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP2IE LL_RTC_DisableIT_ITAMP2

LL_RTC_EnableIT_ITAMP3

Function name

__STATIC_INLINE void LL_RTC_EnableIT_ITAMP3 (const RTC_TypeDef * RTCx)

Function description

Enable internal tamper 3 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP3IE LL_RTC_EnableIT_ITAMP3

LL_RTC_DisableIT_ITAMP3

Function name

__STATIC_INLINE void LL_RTC_DisableIT_ITAMP3 (const RTC_TypeDef * RTCx)

Function description

Disable internal tamper 3 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP3IE LL_RTC_DisableIT_ITAMP3

LL_RTC_EnableIT_ITAMP4

Function name

__STATIC_INLINE void LL_RTC_EnableIT_ITAMP4 (const RTC_TypeDef * RTCx)

Function description

Enable internal tamper 4 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP4IE LL_RTC_EnableIT_ITAMP4

LL_RTC_DisableIT_ITAMP4

Function name

__STATIC_INLINE void LL_RTC_DisableIT_ITAMP4 (const RTC_TypeDef * RTCx)

Function description

Disable internal tamper 4 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP4IE LL_RTC_DisableIT_ITAMP4

LL_RTC_EnableIT_ITAMP5

Function name

__STATIC_INLINE void LL_RTC_EnableIT_ITAMP5 (const RTC_TypeDef * RTCx)

Function description

Enable internal tamper 5 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP5IE LL_RTC_EnableIT_ITAMP5

LL_RTC_DisableIT_ITAMP5

Function name

__STATIC_INLINE void LL_RTC_DisableIT_ITAMP5 (const RTC_TypeDef * RTCx)

Function description

Disable internal tamper 5 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP5IE LL_RTC_DisableIT_ITAMP5

LL_RTC_EnableIT_ITAMP6

Function name

__STATIC_INLINE void LL_RTC_EnableIT_ITAMP6 (const RTC_TypeDef * RTCx)

Function description

Enable internal tamper 6 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP6IE LL_RTC_EnableIT_ITAMP6

LL_RTC_DisableIT_ITAMP6

Function name

__STATIC_INLINE void LL_RTC_DisableIT_ITAMP6 (const RTC_TypeDef * RTCx)

Function description

Disable internal tamper 6 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP6IE LL_RTC_DisableIT_ITAMP6

LL_RTC_EnableIT_ITAMP7

Function name

__STATIC_INLINE void LL_RTC_EnableIT_ITAMP7 (const RTC_TypeDef * RTCx)

Function description

Enable internal tamper 7 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP7IE LL_RTC_EnableIT_ITAMP7

LL_RTC_DisableIT_ITAMP7

Function name

__STATIC_INLINE void LL_RTC_DisableIT_ITAMP7 (const RTC_TypeDef * RTCx)

Function description

Disable internal tamper 7 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP7IE LL_RTC_DisableIT_ITAMP7

LL_RTC_EnableIT_ITAMP8

Function name

__STATIC_INLINE void LL_RTC_EnableIT_ITAMP8 (const RTC_TypeDef * RTCx)

Function description

Enable internal tamper 8 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP8IE LL_RTC_EnableIT_ITAMP8

LL_RTC_DisableIT_ITAMP8

Function name

__STATIC_INLINE void LL_RTC_DisableIT_ITAMP8 (const RTC_TypeDef * RTCx)

Function description

Disable internal tamper 8 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP8IE LL_RTC_DisableIT_ITAMP8

LL_RTC_EnableIT_ITAMP9

Function name

__STATIC_INLINE void LL_RTC_EnableIT_ITAMP9 (const RTC_TypeDef * RTCx)

Function description

Enable internal tamper 9 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP9IE LL_RTC_EnableIT_ITAMP9

LL_RTC_DisableIT_ITAMP9

Function name

__STATIC_INLINE void LL_RTC_DisableIT_ITAMP9 (const RTC_TypeDef * RTCx)

Function description

Disable internal tamper 9 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP9IE LL_RTC_DisableIT_ITAMP9

LL_RTC_EnableIT_ITAMP11

Function name

__STATIC_INLINE void LL_RTC_EnableIT_ITAMP11 (const RTC_TypeDef * RTCx)

Function description

Enable internal tamper 11 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP11IE LL_RTC_EnableIT_ITAMP11

LL_RTC_DisableIT_ITAMP11

Function name

__STATIC_INLINE void LL_RTC_DisableIT_ITAMP11 (const RTC_TypeDef * RTCx)

Function description

Disable internal tamper 11 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP11IE LL_RTC_DisableIT_ITAMP11

LL_RTC_IsEnabledIT_TAMP1

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_TAMP1 (const RTC_TypeDef * RTCx)

Function description

Check if tamper 1 interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_IER TAMP1IE LL_RTC_IsEnabledIT_TAMP1

LL_RTC_IsEnabledIT_TAMP2

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_TAMP2 (const RTC_TypeDef * RTCx)

Function description

Check if tamper 2 interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_IER TAMP2IE LL_RTC_IsEnabledIT_TAMP2

LL_RTC_IsEnabledIT_TAMP3

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_TAMP3 (const RTC_TypeDef * RTCx)

Function description

Check if tamper 3 interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_IER TAMP3IE LL_RTC_IsEnabledIT_TAMP3

LL_RTC_IsEnabledIT_TAMP4

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_TAMP4 (const RTC_TypeDef * RTCx)

Function description

Check if tamper 4 interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_IER TAMP4IE LL_RTC_IsEnabledIT_TAMP4

LL_RTC_IsEnabledIT_TAMP5

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_TAMP5 (const RTC_TypeDef * RTCx)

Function description

Check if tamper 5 interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_IER TAMP5IE LL_RTC_IsEnabledIT_TAMP5

LL_RTC_IsEnabledIT_TAMP6

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_TAMP6 (const RTC_TypeDef * RTCx)

Function description

Check if tamper 6 interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_IER TAMP6IE LL_RTC_IsEnabledIT_TAMP6

LL_RTC_IsEnabledIT_TAMP7

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_TAMP7 (const RTC_TypeDef * RTCx)

Function description

Check if tamper 7 interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_IER TAMP7IE LL_RTC_IsEnabledIT_TAMP7

LL_RTC_IsEnabledIT_ITAMP1

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_ITAMP1 (const RTC_TypeDef * RTCx)

Function description

Check if internal tamper 1 interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP1IE LL_RTC_IsEnabledIT_ITAMP1

LL_RTC_IsEnabledIT_ITAMP2

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_ITAMP2 (const RTC_TypeDef * RTCx)

Function description

Check if internal tamper 2 interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP2IE LL_RTC_IsEnabledIT_ITAMP2

LL_RTC_IsEnabledIT_ITAMP3

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_ITAMP3 (const RTC_TypeDef * RTCx)

Function description

Check if internal tamper 3 interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP3IE LL_RTC_IsEnabledIT_ITAMP3

LL_RTC_IsEnabledIT_ITAMP4

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_ITAMP4 (const RTC_TypeDef * RTCx)

Function description

Check if internal tamper 4 interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP4IE LL_RTC_IsEnabledIT_ITAMP4

LL_RTC_IsEnabledIT_ITAMP5

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_ITAMP5 (const RTC_TypeDef * RTCx)

Function description

Check if internal tamper 5 interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP5IE LL_RTC_IsEnabledIT_ITAMP5

LL_RTC_IsEnabledIT_ITAMP6

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_ITAMP6 (const RTC_TypeDef * RTCx)

Function description

Check if internal tamper 6 interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP6IE LL_RTC_IsEnabledIT_ITAMP6

LL_RTC_IsEnabledIT_ITAMP7

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_ITAMP7 (const RTC_TypeDef * RTCx)

Function description

Check if internal tamper 7 interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP7IE LL_RTC_IsEnabledIT_ITAMP7

LL_RTC_IsEnabledIT_ITAMP8

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_ITAMP8 (const RTC_TypeDef * RTCx)

Function description

Check if internal tamper 8 interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP8IE LL_RTC_IsEnabledIT_ITAMP8

LL_RTC_IsEnabledIT_ITAMP9

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_ITAMP9 (const RTC_TypeDef * RTCx)

Function description

Check if internal tamper 9 interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP9IE LL_RTC_IsEnabledIT_ITAMP9

LL_RTC_IsEnabledIT_ITAMP11

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_ITAMP11 (const RTC_TypeDef * RTCx)

Function description

Check if internal tamper 11 interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP11IE LL_RTC_IsEnabledIT_ITAMP11

LL_RTC_IncrementMonotonicCounter

Function name

__STATIC_INLINE void LL_RTC_IncrementMonotonicCounter (const RTC_TypeDef * RTCx)

Function description

Increment Monotonic counter.

Parameters

- **RTCx:** RTC Instance

Return values

- **None.:**

Reference Manual to LL API cross reference:

- TAMP_COUNT1R COUNT LL_RTC_IncrementMonotonicCounter

LL_RTC_GetMonotonicCounter

Function name

__STATIC_INLINE uint32_t LL_RTC_GetMonotonicCounter (const RTC_TypeDef * RTCx)

Function description

Increment Monotonic counter.

Parameters

- **RTCx:** RTC Instance

Return values

- **Monotonic:** counter value.

Reference Manual to LL API cross reference:

- TAMP_COUNT1R COUNT LL_RTC_GetMonotonicCounter

LL_RTC_DeInit

Function name

ErrorStatus LL_RTC_DeInit (RTC_TypeDef * RTCx)

Function description

De-Initializes the RTC registers to their default reset values.

Parameters

- **RTCx:** RTC Instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: RTC registers are de-initialized
  - ERROR: RTC registers are not de-initialized

Notes

- This function does not reset the RTC Clock source and RTC Backup Data registers.

LL_RTC_Init

Function name

ErrorStatus LL_RTC_Init (RTC_TypeDef * RTCx, LL_RTC_InitTypeDef * RTC_InitStruct)

Function description

Initializes the RTC registers according to the specified parameters in RTC_InitStruct.

Parameters

- **RTCx:** RTC Instance
- **RTC_InitStruct:** pointer to a LL_RTC_InitTypeDef structure that contains the configuration information for the RTC peripheral.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: RTC registers are initialized
  - ERROR: RTC registers are not initialized

Notes

- The RTC Prescaler register is write protected and can be written in initialization mode only.

LL_RTC_StructInit

Function name

void LL_RTC_StructInit (LL_RTC_InitTypeDef * RTC_InitStruct)

Function description

Set each LL_RTC_InitTypeDef field to default value.

Parameters

- **RTC_InitStruct:** pointer to a LL_RTC_InitTypeDef structure which will be initialized.

Return values

- **None:**

LL_RTC_TIME_Init

Function name

ErrorStatus LL_RTC_TIME_Init (RTC_TypeDef * RTCx, uint32_t RTC_Format, LL_RTC_TimeTypeDef * RTC_TimeStruct)

Function description

Set the RTC current time.

Parameters

- **RTCx:** RTC Instance
- **RTC_Format:** This parameter can be one of the following values:
  - LL_RTC_FORMAT_BIN
  - LL_RTC_FORMAT_BCD
- **RTC_TimeStruct:** pointer to a RTC_TimeTypeDef structure that contains the time configuration information for the RTC.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: RTC Time register is configured
  - ERROR: RTC Time register is not configured

LL_RTC_TIME_StructInit

Function name

void LL_RTC_TIME_StructInit (LL_RTC_TimeTypeDef * RTC_TimeStruct)

Function description

Set each LL_RTC_TimeTypeDef field to default value (Time = 00h:00min:00sec).

Parameters

- **RTC_TimeStruct:** pointer to a LL_RTC_TimeTypeDef structure which will be initialized.

Return values

- **None:**

LL_RTC_DATE_Init

Function name

ErrorStatus LL_RTC_DATE_Init (RTC_TypeDef * RTCx, uint32_t RTC_Format, LL_RTC_DateTypeDef * RTC_DateStruct)

Function description

Set the RTC current date.

Parameters

- **RTCx:** RTC Instance
- **RTC_Format:** This parameter can be one of the following values:
  - LL_RTC_FORMAT_BIN
  - LL_RTC_FORMAT_BCD
- **RTC_DateStruct:** pointer to a RTC_DateTypeDef structure that contains the date configuration information for the RTC.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: RTC Day register is configured
  - ERROR: RTC Day register is not configured

LL_RTC_DATE_StructInit

Function name

void LL_RTC_DATE_StructInit (LL_RTC_DateTypeDef * RTC_DateStruct)

Function description

Set each LL_RTC_DateTypeDef field to default value (date = Monday, January 01 xx00)

Parameters

- **RTC_DateStruct:** pointer to a LL_RTC_DateTypeDef structure which will be initialized.

Return values

- **None:**

LL_RTC_ALMA_Init

Function name

ErrorStatus LL_RTC_ALMA_Init (RTC_TypeDef * RTCx, uint32_t RTC_Format, LL_RTC_AlarmTypeDef * RTC_AlarmStruct)

Function description

Set the RTC Alarm A.

Parameters

- **RTCx:** RTC Instance
- **RTC_Format:** This parameter can be one of the following values:
  - LL_RTC_FORMAT_BIN
  - LL_RTC_FORMAT_BCD
- **RTC_AlarmStruct:** pointer to a LL_RTC_AlarmTypeDef structure that contains the alarm configuration parameters.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: ALARMA registers are configured
  - ERROR: ALARMA registers are not configured

Notes

- The Alarm register can only be written when the corresponding Alarm is disabled (Use LL_RTC_ALMA_Disable function).

LL_RTC_ALMB_Init

Function name

ErrorStatus LL_RTC_ALMB_Init (RTC_TypeDef * RTCx, uint32_t RTC_Format, LL_RTC_AlarmTypeDef * RTC_AlarmStruct)

Function description

Set the RTC Alarm B.

Parameters

- **RTCx:** RTC Instance
- **RTC_Format:** This parameter can be one of the following values:
  - LL_RTC_FORMAT_BIN
  - LL_RTC_FORMAT_BCD
- **RTC_AlarmStruct:** pointer to a LL_RTC_AlarmTypeDef structure that contains the alarm configuration parameters.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: ALARMB registers are configured
  - ERROR: ALARMB registers are not configured

Notes

- The Alarm register can only be written when the corresponding Alarm is disabled (LL_RTC_ALMB_Disable function).

LL_RTC_ALMA_StructInit

Function name

void LL_RTC_ALMA_StructInit (LL_RTC_AlarmTypeDef * RTC_AlarmStruct)

Function description

Set each LL_RTC_AlarmTypeDef of ALARMA field to default value (Time = 00h:00mn:00sec / Day = 1st day of the month/Mask = all fields are masked).

Parameters

- **RTC_AlarmStruct:** pointer to a LL_RTC_AlarmTypeDef structure which will be initialized.

Return values

- **None:**

LL_RTC_ALMB_StructInit

Function name

void LL_RTC_ALMB_StructInit (LL_RTC_AlarmTypeDef * RTC_AlarmStruct)

Function description

Set each LL_RTC_AlarmTypeDef of ALARMA field to default value (Time = 00h:00mn:00sec / Day = 1st day of the month/Mask = all fields are masked).

Parameters

- **RTC_AlarmStruct:** pointer to a LL_RTC_AlarmTypeDef structure which will be initialized.

Return values

- **None:**

LL_RTC_EnterInitMode

Function name

ErrorStatus LL_RTC_EnterInitMode (RTC_TypeDef * RTCx)

Function description

Enters the RTC Initialization mode.

Parameters

- **RTCx:** RTC Instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: RTC is in Init mode
  - ERROR: RTC is not in Init mode

Notes

- The RTC Initialization mode is write protected, use the LL_RTC_DisableWriteProtection before calling this function.

LL_RTC_ExitInitMode

Function name

ErrorStatus LL_RTC_ExitInitMode (RTC_TypeDef * RTCx)

Function description

Exit the RTC Initialization mode.

Parameters

- **RTCx:** RTC Instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: RTC exited from in Init mode
  - ERROR: Not applicable

Notes

- When the initialization sequence is complete, the calendar restarts counting after 4 RTCCLK cycles.
- The RTC Initialization mode is write protected, use the LL_RTC_DisableWriteProtection before calling this function.

LL_RTC_WaitForSynchro

Function name

ErrorStatus LL_RTC_WaitForSynchro (RTC_TypeDef * RTCx)

Function description

Waits until the RTC Time and Day registers (RTC_TR and RTC_DR) are synchronized with RTC APB clock.

Parameters

- **RTCx:** RTC Instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: RTC registers are synchronised
  - ERROR: RTC registers are not synchronised

Notes

- The RTC Resynchronization mode is write protected, use the LL_RTC_DisableWriteProtection before calling this function.
- To read the calendar through the shadow registers after Calendar initialization, calendar update or after wakeup from low power modes the software must first clear the RSF flag. The software must then wait until it is set again before reading the calendar, which means that the calendar registers have been correctly copied into the RTC_TR and RTC_DR shadow registers.

103.3 RTC Firmware driver defines

The following section lists the various define and macros of the module.

103.3.1 RTC

RTC

ACTIVE TAMPER ASYNCHRONOUS PRESCALER CLOCK

LL_RTC_TAMPER_ATAMP_ASYNCPRES_RTCCLK

RTCCLK

LL_RTC_TAMPER_ATAMP_ASYNCPRES_RTCCLK_2

RTCCLK/2

LL_RTC_TAMPER_ATAMP_ASYNCPRES_RTCCLK_4

RTCCLK/4

LL_RTC_TAMPER_ATAMP_ASYNCPRES_RTCCLK_8

RTCCLK/8

LL_RTC_TAMPER_ATAMP_ASYNCPRES_RTCCLK_16

RTCCLK/16

LL_RTC_TAMPER_ATAMP_ASYNCPRES_RTCCLK_32

RTCCLK/32

LL_RTC_TAMPER_ATAMP_ASYNCPRES_RTCCLK_64

RTCCLK/64

LL_RTC_TAMPER_ATAMP_ASYNCPRES_RTCCLK_128

RTCCLK/128

LL_RTC_TAMPER_ATAMP_ASYNCPRES_RTCCLK_256

RTCCLK/256

LL_RTC_TAMPER_ATAMP_ASYNCPRES_RTCCLK_512

RTCCLK/512

LL_RTC_TAMPER_ATAMP_ASYNCPRES_RTCCLK_1024

RTCCLK/1024

LL_RTC_TAMPER_ATAMP_ASYNCPRES_RTCCLK_2048

RTCCLK/2048

LL_RTC_TAMPER_ATAMP_ASYNCPRES_RTCCLK_4096

RTCCLK/4096

LL_RTC_TAMPER_ATAMP_ASYNCPRES_RTCCLK_8192

RTCCLK/8192

LL_RTC_TAMPER_ATAMP_ASYNCPRES_RTCCLK_16384

RTCCLK/16384

LL_RTC_TAMPER_ATAMP_ASYNCPRES_RTCCLK_32768

RTCCLK/32768

ACTIVE TAMPER MODE

LL_RTC_TAMPER_ATAMP_TAMP1AM

Tamper 1 is active

LL_RTC_TAMPER_ATAMP_TAMP2AM

Tamper 2 is active

LL_RTC_TAMPER_ATAMP_TAMP3AM

Tamper 3 is active

LL_RTC_TAMPER_ATAMP_TAMP4AM

Tamper 4 is active

LL_RTC_TAMPER_ATAMP_TAMP5AM

Tamper 5 is active

LL_RTC_TAMPER_ATAMP_TAMP6AM

Tamper 6 is active

LL_RTC_TAMPER_ATAMP_TAMP7AM

Tamper 7 is active

ACTIVE TAMPER OUTPUT SELECTION

LL_RTC_TAMPER_ATAMP1IN_ATAMP1OUT

LL_RTC_TAMPER_ATAMP1IN_ATAMP2OUT

LL_RTC_TAMPER_ATAMP1IN_ATAMP3OUT

LL_RTC_TAMPER_ATAMP1IN_ATAMP4OUT

LL_RTC_TAMPER_ATAMP1IN_ATAMP5OUT

LL_RTC_TAMPER_ATAMP1IN_ATAMP6OUT

LL_RTC_TAMPER_ATAMP1IN_ATAMP7OUT

LL_RTC_TAMPER_ATAMP1IN_ATAMP8OUT

LL_RTC_TAMPER_ATAMP2IN_ATAMP1OUT

LL_RTC_TAMPER_ATAMP2IN_ATAMP2OUT

LL_RTC_TAMPER_ATAMP2IN_ATAMP3OUT

LL_RTC_TAMPER_ATAMP2IN_ATAMP4OUT

LL_RTC_TAMPER_ATAMP2IN_ATAMP5OUT

LL_RTC_TAMPER_ATAMP2IN_ATAMP6OUT

LL_RTC_TAMPER_ATAMP2IN_ATAMP7OUT

LL_RTC_TAMPER_ATAMP2IN_ATAMP8OUT

LL_RTC_TAMPER_ATAMP3IN_ATAMP1OUT

LL_RTC_TAMPER_ATAMP3IN_ATAMP2OUT

LL_RTC_TAMPER_ATAMP3IN_ATAMP3OUT

LL_RTC_TAMPER_ATAMP3IN_ATAMP4OUT

LL_RTC_TAMPER_ATAMP3IN_ATAMP5OUT

LL_RTC_TAMPER_ATAMP3IN_ATAMP6OUT

LL_RTC_TAMPER_ATAMP3IN_ATAMP7OUT

LL_RTC_TAMPER_ATAMP3IN_ATAMP8OUT

LL_RTC_TAMPER_ATAMP4IN_ATAMP1OUT

LL_RTC_TAMPER_ATAMP4IN_ATAMP2OUT

LL_RTC_TAMPER_ATAMP4IN_ATAMP3OUT

LL_RTC_TAMPER_ATAMP4IN_ATAMP4OUT

LL_RTC_TAMPER_ATAMP4IN_ATAMP5OUT

LL_RTC_TAMPER_ATAMP4IN_ATAMP6OUT

LL_RTC_TAMPER_ATAMP4IN_ATAMP7OUT

LL_RTC_TAMPER_ATAMP4IN_ATAMP8OUT

LL_RTC_TAMPER_ATAMP5IN_ATAMP1OUT

LL_RTC_TAMPER_ATAMP5IN_ATAMP2OUT

LL_RTC_TAMPER_ATAMP5IN_ATAMP3OUT

LL_RTC_TAMPER_ATAMP5IN_ATAMP4OUT

LL_RTC_TAMPER_ATAMP5IN_ATAMP5OUT

LL_RTC_TAMPER_ATAMP5IN_ATAMP6OUT

LL_RTC_TAMPER_ATAMP5IN_ATAMP7OUT

LL_RTC_TAMPER_ATAMP5IN_ATAMP8OUT

LL_RTC_TAMPER_ATAMP6IN_ATAMP1OUT

LL_RTC_TAMPER_ATAMP6IN_ATAMP2OUT

LL_RTC_TAMPER_ATAMP6IN_ATAMP3OUT

LL_RTC_TAMPER_ATAMP6IN_ATAMP4OUT

LL_RTC_TAMPER_ATAMP6IN_ATAMP5OUT

LL_RTC_TAMPER_ATAMP6IN_ATAMP6OUT

LL_RTC_TAMPER_ATAMP6IN_ATAMP7OUT

LL_RTC_TAMPER_ATAMP6IN_ATAMP8OUT

LL_RTC_TAMPER_ATAMP7IN_ATAMP1OUT

LL_RTC_TAMPER_ATAMP7IN_ATAMP2OUT

LL_RTC_TAMPER_ATAMP7IN_ATAMP3OUT

LL_RTC_TAMPER_ATAMP7IN_ATAMP4OUT

LL_RTC_TAMPER_ATAMP7IN_ATAMP5OUT

LL_RTC_TAMPER_ATAMP7IN_ATAMP6OUT

LL_RTC_TAMPER_ATAMP7IN_ATAMP7OUT

LL_RTC_TAMPER_ATAMP7IN_ATAMP8OUT

ALARM OUTPUT

LL_RTC_ALARMOUT_DISABLE

Output disabled

LL_RTC_ALARMOUT_ALMA

Alarm A output enabled

LL_RTC_ALARMOUT_ALMB

Alarm B output enabled

LL_RTC_ALARMOUT_WAKEUP

Wakeup output enabled

ALARM OUTPUT TYPE

LL_RTC_ALARM_OUTPUTTYPE_PUSHPULL

RTC_ALARM is push-pull output

LL_RTC_ALARM_OUTPUTTYPE_OPENDRAIN

RTC_ALARM is open-drain output

ALARMA MASK

LL_RTC_ALMA_MASK_NONE

No masks applied on Alarm A

LL_RTC_ALMA_MASK_DATEWEEKDAY

Date/day do not care in Alarm A comparison

LL_RTC_ALMA_MASK_HOURS

Hours do not care in Alarm A comparison

LL_RTC_ALMA_MASK_MINUTES

Minutes do not care in Alarm A comparison

LL_RTC_ALMA_MASK_SECONDS

Seconds do not care in Alarm A comparison

LL_RTC_ALMA_MASK_ALL

Masks all

RTC Alarm Sub Seconds with binary mode auto clear Definitions

LL_RTC_ALMA_SUBSECONDBIN_AUTOCLR_NO

The synchronous binary counter (SS[31:0] in RTC_SSR) is free-running.

LL_RTC_ALMA_SUBSECONDBIN_AUTOCLR_YES

The synchronous binary counter (SS[31:0] in RTC_SSR) is running from 0xFFFF FFFF to RTC_ALRMABINR -> SS[31:0] value and is automatically reloaded with 0xFFFF FFFF when reaching RTC_ALRMABINR -> SS[31:0].

ALARMA TIME FORMAT

LL_RTC_ALMA_TIME_FORMAT_AM

AM or 24-hour format

LL_RTC_ALMA_TIME_FORMAT_PM

PM

RTC Alarm A Date WeekDay

LL_RTC_ALMA_DATEWEEKDAYSEL_DATE

Alarm A Date is selected

LL_RTC_ALMA_DATEWEEKDAYSEL_WEEKDAY

Alarm A WeekDay is selected

ALARMB MASK

LL_RTC_ALMB_MASK_NONE

No masks applied on Alarm B

LL_RTC_ALMB_MASK_DATEWEEKDAY

Date/day do not care in Alarm B comparison

LL_RTC_ALMB_MASK_HOURS

Hours do not care in Alarm B comparison

LL_RTC_ALMB_MASK_MINUTES

Minutes do not care in Alarm B comparison

LL_RTC_ALMB_MASK_SECONDS

Seconds do not care in Alarm B comparison

LL_RTC_ALMB_MASK_ALL

Masks all

Alarm Sub Seconds with binary mode auto clear Definitions

LL_RTC_ALMB_SUBSECONDBIN_AUTOCLR_NO

The synchronous binary counter (SS[31:0] in RTC_SSR) is free-running.

LL_RTC_ALMB_SUBSECONDBIN_AUTOCLR_YES

The synchronous binary counter (SS[31:0] in RTC_SSR) is running from 0xFFFF FFFF to RTC_ALRMBBINR -> SS[31:0] value and is automatically reloaded with 0xFFFF FFFF when reaching RTC_ALRMBBINR -> SS[31:0].

ALARMB TIME FORMAT

LL_RTC_ALMB_TIME_FORMAT_AM

AM or 24-hour format

LL_RTC_ALMB_TIME_FORMAT_PM

PM

RTC Alarm B Date WeekDay

LL_RTC_ALMB_DATEWEEKDAYSEL_DATE

Alarm B Date is selected

LL_RTC_ALMB_DATEWEEKDAYSEL_WEEKDAY

Alarm B WeekDay is selected

Calendar second incrementation in Binary mix mode

LL_RTC_BINARY_MIX_BCDU_0

1s calendar increment is generated each time SS[7:0] = 0

LL_RTC_BINARY_MIX_BCDU_1

1s calendar increment is generated each time SS[8:0] = 0

LL_RTC_BINARY_MIX_BCDU_2

1s calendar increment is generated each time SS[9:0] = 0

LL_RTC_BINARY_MIX_BCDU_3

1s calendar increment is generated each time SS[10:0] = 0

LL_RTC_BINARY_MIX_BCDU_4

1s calendar increment is generated each time SS[11:0] = 0

LL_RTC_BINARY_MIX_BCDU_5

1s calendar increment is generated each time SS[12:0] = 0

LL_RTC_BINARY_MIX_BCDU_6

1s calendar increment is generated each time SS[13:0] = 0

LL_RTC_BINARY_MIX_BCDU_7

1s calendar increment is generated each time SS[14:0] = 0

Binary mode (Sub Second Register)

LL_RTC_BINARY_NONE

Free running BCD calendar mode (Binary mode disabled)

LL_RTC_BINARY_ONLY

Free running Binary mode (BCD mode disabled)

LL_RTC_BINARY_MIX

Free running BCD calendar and Binary mode enable

BACKUP

LL_RTC_BKP_NUMBER

LL_RTC_BKP_DR0

LL_RTC_BKP_DR1

LL_RTC_BKP_DR2

LL_RTC_BKP_DR3

LL_RTC_BKP_DR4

LL_RTC_BKP_DR5

LL_RTC_BKP_DR6

LL_RTC_BKP_DR7

LL_RTC_BKP_DR8

LL_RTC_BKP_DR9

LL_RTC_BKP_DR10

LL_RTC_BKP_DR11

LL_RTC_BKP_DR12

LL_RTC_BKP_DR13

LL_RTC_BKP_DR14

LL_RTC_BKP_DR15

LL_RTC_BKP_DR16

LL_RTC_BKP_DR17

LL_RTC_BKP_DR18

LL_RTC_BKP_DR19

LL_RTC_BKP_DR20

LL_RTC_BKP_DR21

LL_RTC_BKP_DR22

LL_RTC_BKP_DR23

LL_RTC_BKP_DR24

LL_RTC_BKP_DR25

LL_RTC_BKP_DR26

LL_RTC_BKP_DR27

LL_RTC_BKP_DR28

LL_RTC_BKP_DR29

LL_RTC_BKP_DR30

LL_RTC_BKP_DR31

Calibration pulse insertion

LL_RTC_CALIB_INSERTPULSE_NONE

No RTCCLK pulses are added

LL_RTC_CALIB_INSERTPULSE_SET

One RTCCLK pulse is effectively inserted every 2exp11 pulses (frequency increased by 488.5 ppm)

Calibration low power

LL_RTC_CALIB_LOWPOWER_NONE

High conso mode

LL_RTC_CALIB_LOWPOWER_SET

Low power mode

Calibration output

LL_RTC_CALIB_OUTPUT_NONE

Calibration output disabled

LL_RTC_CALIB_OUTPUT_1HZ

Calibration output is 1 Hz

LL_RTC_CALIB_OUTPUT_512HZ

Calibration output is 512 Hz

Calibration period

LL_RTC_CALIB_PERIOD_32SEC

Use a 32-second calibration cycle period

LL_RTC_CALIB_PERIOD_16SEC

Use a 16-second calibration cycle period

LL_RTC_CALIB_PERIOD_8SEC

Use a 8-second calibration cycle period

FORMAT

LL_RTC_FORMAT_BIN

Binary data format

LL_RTC_FORMAT_BCD

BCD data format

Get Flags Defines

LL_RTC_SCR_SSRUF

LL_RTC_SCR_ITSF

LL_RTC_SCR_TSOVF

LL_RTC_SCR_TSF

LL_RTC_SCR_WUTF

LL_RTC_SCR_ALRBF

LL_RTC_SCR_ALRAF

LL_RTC_ICSR_RECALPF

LL_RTC_ICSR_BCDU_2

LL_RTC_ICSR_BCDU_1

LL_RTC_ICSR_BCDU_0

LL_RTC_ICSR_BIN_1

LL_RTC_ICSR_BIN_0

LL_RTC_ICSR_INITF

LL_RTC_ICSR_RSF

LL_RTC_ICSR_INITS

LL_RTC_ICSR_SHPF

LL_RTC_ICSR_WUTWF

HOUR FORMAT

LL_RTC_HOURFORMAT_24HOUR

24 hour/day format

LL_RTC_HOURFORMAT_AMPM

AM/PM hour format

INTERNAL TAMPER

LL_RTC_TAMPER_ITAMP1

Internal tamper 1: RTC supply voltage monitoring

LL_RTC_TAMPER_ITAMP2

Internal tamper 2: Temperature monitoring

LL_RTC_TAMPER_ITAMP3

Internal tamper 3: LSE monitoring

LL_RTC_TAMPER_ITAMP4

Internal tamper 4: HSE monitoring

LL_RTC_TAMPER_ITAMP5

Internal tamper 5: RTC calendar overflow

LL_RTC_TAMPER_ITAMP6

Internal tamper 6: JTAG/SWD access when RDP > 0

LL_RTC_TAMPER_ITAMP7

Internal tamper 7: Voltage monitoring (VCORE, VREF+), through ADC analog watchdog

LL_RTC_TAMPER_ITAMP8

Internal tamper 8: Monotonic counter overflow

LL_RTC_TAMPER_ITAMP9

Internal tamper 9: Cryptographic IPs fault

LL_RTC_TAMPER_ITAMP11

Internal tamper 11: IWDG reset when tamper flag is set

IT Defines

LL_RTC_CR_TSIE

LL_RTC_CR_WUTIE

LL_RTC_CR_ALRBIE

LL_RTC_CR_ALRAIE

INTERNAL TAMPER NO ERASE

LL_RTC_TAMPER_NOERASE_ITAMPER1

Internal tamper 1 event does not erase the backup registers

LL_RTC_TAMPER_NOERASE_ITAMPER2

Internal tamper 2 event does not erase the backup registers

LL_RTC_TAMPER_NOERASE_ITAMPER3

Internal tamper 3 event does not erase the backup registers

LL_RTC_TAMPER_NOERASE_ITAMPER4

Internal tamper 4 event does not erase the backup registers

LL_RTC_TAMPER_NOERASE_ITAMPER5

Internal tamper 5 event does not erase the backup registers

LL_RTC_TAMPER_NOERASE_ITAMPER6

Internal tamper 6 event does not erase the backup registers

LL_RTC_TAMPER_NOERASE_ITAMPER7

Internal tamper 7 event does not erase the backup registers

LL_RTC_TAMPER_NOERASE_ITAMPER8

Internal tamper 8 event does not erase the backup registers

LL_RTC_TAMPER_NOERASE_ITAMPER9

Internal tamper 9 event does not erase the backup registers

LL_RTC_TAMPER_NOERASE_ITAMPER11

Internal tamper 11 event does not erase the backup registers

LL_RTC_TAMPER_NOERASE_ITAMPER15

Internal tamper 15 event does not erase the backup registers

MONTH

LL_RTC_MONTH_JANUARY

January

LL_RTC_MONTH_FEBRUARY

February

LL_RTC_MONTH_MARCH

March

LL_RTC_MONTH_APRIL

April

LL_RTC_MONTH_MAY

May

LL_RTC_MONTH_JUNE

June

LL_RTC_MONTH_JULY

July

LL_RTC_MONTH_AUGUST

August

LL_RTC_MONTH_SEPTEMBER

September

LL_RTC_MONTH_OCTOBER

October

LL_RTC_MONTH_NOVEMBER

November

LL_RTC_MONTH_DECEMBER

December

OUTPUT POLARITY PIN

LL_RTC_OUTPUTPOLARITY_PIN_HIGH

Pin is high when ALRAF/ALRBF/WUTF is asserted (depending on OSEL)

LL_RTC_OUTPUTPOLARITY_PIN_LOW

Pin is low when ALRAF/ALRBF/WUTF is asserted (depending on OSEL)

Privilege Backup register privilege zone

LL_RTC_PRIVILEGE_BKUP_ZONE_NONE

LL_RTC_PRIVILEGE_BKUP_ZONE_1

LL_RTC_PRIVILEGE_BKUP_ZONE_2

LL_RTC_PRIVILEGE_BKUP_ZONE_ALL

Privilege rtc features in case of LL_RTC_PRIVILEGE_FULL_NO.

LL_RTC_PRIVILEGE_FEATURE_INIT

Initialization feature is privilege

LL_RTC_PRIVILEGE_FEATURE_CAL

Calibration feature is privilege

LL_RTC_PRIVILEGE_FEATURE_TS

Time stamp feature is privilege

LL_RTC_PRIVILEGE_FEATURE_WUT

Wake up timer feature is privilege

LL_RTC_PRIVILEGE_FEATURE_ALRA

Alarm A feature is privilege

LL_RTC_PRIVILEGE_FEATURE_ALRB

Alarm B feature is privilege

Privilege full rtc

LL_RTC_PRIVILEGE_FULL_YES

RTC full privilege

LL_RTC_PRIVILEGE_FULL_NO

RTC is not full privilege, features can be unprivilege. See RTC_LL_EC_PRIVILEGE_RTC_FEATURE

Privilege full tamp

LL_TAMP_PRIVILEGE_FULL_YES

TAMP full privilege

LL_TAMP_PRIVILEGE_FULL_NO

TAMP is not privilege

Secure features rtc in case of LL_RTC_SECURE_FULL_NO.

LL_RTC_SECURE_FEATURE_INIT

Initialization feature is secure

LL_RTC_SECURE_FEATURE_CAL

Calibration feature is secure

LL_RTC_SECURE_FEATURE_TS

Time stamp feature is secure

LL_RTC_SECURE_FEATURE_WUT

Wake up timer feature is secure

LL_RTC_SECURE_FEATURE_ALRA

Alarm A feature is secure

LL_RTC_SECURE_FEATURE_ALRB

Alarm B feature is secure

Secure full rtc

LL_RTC_SECURE_FULL_YES

RTC full secure

LL_RTC_SECURE_FULL_NO

RTC is not full secure, features can be secure. See RTC_LL_EC_SECURE_RTC_FEATURE

Secure tamp

LL_TAMP_SECURE_FULL_YES

TAMP full secure

LL_TAMP_SECURE_FULL_NO

TAMP is not secure

SHIFT SECOND

LL_RTC_SHIFT_SECOND_DELAY

Delay (seconds) = SUBFS / (PREDIV_S + 1)

LL_RTC_SHIFT_SECOND_ADVANCE

Advance (seconds) = (1 - (SUBFS / (PREDIV_S + 1)))

TAMPER

LL_RTC_TAMPER_1

Tamper 1 input detection

LL_RTC_TAMPER_2

Tamper 2 input detection

LL_RTC_TAMPER_3

Tamper 3 input detection

LL_RTC_TAMPER_4

Tamper 4 input detection

LL_RTC_TAMPER_5

Tamper 5 input detection

LL_RTC_TAMPER_6

Tamper 6 input detection

LL_RTC_TAMPER_7

Tamper 7 input detection

TAMPER ACTIVE LEVEL

LL_RTC_TAMPER_ACTIVELEVEL_TAMP1

Tamper 1 input falling edge (if TAMPFLT = 00) or staying high (if TAMPFLT != 00) triggers a tamper detection event

LL_RTC_TAMPER_ACTIVELEVEL_TAMP2

Tamper 2 input falling edge (if TAMPFLT = 00) or staying high (if TAMPFLT != 00) triggers a tamper detection event

LL_RTC_TAMPER_ACTIVELEVEL_TAMP3

Tamper 3 input falling edge (if TAMPFLT = 00) or staying high (if TAMPFLT != 00) triggers a tamper detection event

LL_RTC_TAMPER_ACTIVELEVEL_TAMP4

Tamper 4 input falling edge (if TAMPFLT = 00) or staying high (if TAMPFLT != 00) triggers a tamper detection event

LL_RTC_TAMPER_ACTIVELEVEL_TAMP5

Tamper 5 input falling edge (if TAMPFLT = 00) or staying high (if TAMPFLT != 00) triggers a tamper detection event

LL_RTC_TAMPER_ACTIVELEVEL_TAMP6

Tamper 6 input falling edge (if TAMPFLT = 00) or staying high (if TAMPFLT != 00) triggers a tamper detection event

LL_RTC_TAMPER_ACTIVELEVEL_TAMP7

Tamper 7 input falling edge (if TAMPFLT = 00) or staying high (if TAMPFLT != 00) triggers a tamper detection event

TAMPER DURATION

LL_RTC_TAMPER_DURATION_1RTCCLK

Tamper pins are pre-charged before sampling during 1 RTCCLK cycle

LL_RTC_TAMPER_DURATION_2RTCCLK

Tamper pins are pre-charged before sampling during 2 RTCCLK cycles

LL_RTC_TAMPER_DURATION_4RTCCLK

Tamper pins are pre-charged before sampling during 4 RTCCLK cycles

LL_RTC_TAMPER_DURATION_8RTCCLK

Tamper pins are pre-charged before sampling during 8 RTCCLK cycles

TAMPER FILTER

LL_RTC_TAMPER_FILTER_DISABLE

Tamper filter is disabled

LL_RTC_TAMPER_FILTER_2SAMPLE

Tamper is activated after 2 consecutive samples at the active level

LL_RTC_TAMPER_FILTER_4SAMPLE

Tamper is activated after 4 consecutive samples at the active level

LL_RTC_TAMPER_FILTER_8SAMPLE

Tamper is activated after 8 consecutive samples at the active level

TAMPER MASK

LL_RTC_TAMPER_MASK_TAMPER1

Tamper 1 event generates a trigger event. TAMP1F is masked and internally cleared by hardware. The backup registers are not erased

LL_RTC_TAMPER_MASK_TAMPER2

Tamper 2 event generates a trigger event. TAMP2F is masked and internally cleared by hardware. The backup registers are not erased

LL_RTC_TAMPER_MASK_TAMPER3

Tamper 3 event generates a trigger event. TAMP3F is masked and internally cleared by hardware. The backup registers are not erased

TAMPER NO ERASE

LL_RTC_TAMPER_NOERASE_TAMPER1

Tamper 1 event does not erase the backup registers

LL_RTC_TAMPER_NOERASE_TAMPER2

Tamper 2 event does not erase the backup registers

LL_RTC_TAMPER_NOERASE_TAMPER3

Tamper 3 event does not erase the backup registers

LL_RTC_TAMPER_NOERASE_TAMPER4

Tamper 4 event does not erase the backup registers

LL_RTC_TAMPER_NOERASE_TAMPER5

Tamper 5 event does not erase the backup registers

LL_RTC_TAMPER_NOERASE_TAMPER6

Tamper 6 event does not erase the backup registers

LL_RTC_TAMPER_NOERASE_TAMPER7

Tamper 7 event does not erase the backup registers

TAMPER SAMPLING FREQUENCY DIVIDER

LL_RTC_TAMPER_SAMPLFREQDIV_32768

Each of the tamper inputs are sampled with a frequency = RTCCLK / 32768

LL_RTC_TAMPER_SAMPLFREQDIV_16384

Each of the tamper inputs are sampled with a frequency = RTCCLK / 16384

LL_RTC_TAMPER_SAMPLFREQDIV_8192

Each of the tamper inputs are sampled with a frequency = RTCCLK / 8192

LL_RTC_TAMPER_SAMPLFREQDIV_4096

Each of the tamper inputs are sampled with a frequency = RTCCLK / 4096

LL_RTC_TAMPER_SAMPLFREQDIV_2048

Each of the tamper inputs are sampled with a frequency = RTCCLK / 2048

LL_RTC_TAMPER_SAMPLFREQDIV_1024

Each of the tamper inputs are sampled with a frequency = RTCCLK / 1024

LL_RTC_TAMPER_SAMPLFREQDIV_512

Each of the tamper inputs are sampled with a frequency = RTCCLK / 512

LL_RTC_TAMPER_SAMPLFREQDIV_256

Each of the tamper inputs are sampled with a frequency = RTCCLK / 256

TIMESTAMP EDGE

LL_RTC_TIMESTAMP_EDGE_RISING

RTC_TS input rising edge generates a time-stamp event

LL_RTC_TIMESTAMP_EDGE_FALLING

RTC_TS input falling edge generates a time-stamp even

TIME FORMAT

LL_RTC_TIME_FORMAT_AM_OR_24

AM or 24-hour format

LL_RTC_TIME_FORMAT_PM

PM

TIMESTAMP TIME FORMAT

LL_RTC_TS_TIME_FORMAT_AM

AM or 24-hour format

LL_RTC_TS_TIME_FORMAT_PM

PM

WAKEUP CLOCK DIV

LL_RTC_WAKEUPCLOCK_DIV_16

RTC/16 clock is selected

LL_RTC_WAKEUPCLOCK_DIV_8

RTC/8 clock is selected

LL_RTC_WAKEUPCLOCK_DIV_4

RTC/4 clock is selected

LL_RTC_WAKEUPCLOCK_DIV_2

RTC/2 clock is selected

LL_RTC_WAKEUPCLOCK_CKSPRE

ck_spre (usually 1 Hz) clock is selected

LL_RTC_WAKEUPCLOCK_CKSPRE_WUT

ck_spre (usually 1 Hz) clock is selected and 2exp16 is added to the WUT counter value

WEEK DAY

LL_RTC_WEEKDAY_MONDAY

Monday

LL_RTC_WEEKDAY_TUESDAY

Tuesday

LL_RTC_WEEKDAY_WEDNESDAY

Wednesday

LL_RTC_WEEKDAY_THURSDAY

Thrusday

LL_RTC_WEEKDAY_FRIDAY

Friday

LL_RTC_WEEKDAY_SATURDAY

Saturday

LL_RTC_WEEKDAY_SUNDAY

Sunday

Convert helper Macros

__LL_RTC_CONVERT_BIN2BCD

Description:

- Helper macro to convert a value from 2 digit decimal format to BCD format.

Parameters:

- __VALUE__: Byte to be converted

Return value:

- Converted: byte

__LL_RTC_CONVERT_BCD2BIN

Description:

- Helper macro to convert a value from BCD format to 2 digit decimal format.

Parameters:

- __VALUE__: BCD value to be converted

Return value:

- Converted: byte

Date helper Macros

__LL_RTC_GET_WEEKDAY

Description:

- Helper macro to retrieve weekday.

Parameters:

- __RTC_DATE__: Date returned by

Return value:

- Returned: value can be one of the following values:
  - LL_RTC_WEEKDAY_MONDAY
  - LL_RTC_WEEKDAY_TUESDAY
  - LL_RTC_WEEKDAY_WEDNESDAY
  - LL_RTC_WEEKDAY_THURSDAY
  - LL_RTC_WEEKDAY_FRIDAY
  - LL_RTC_WEEKDAY_SATURDAY
  - LL_RTC_WEEKDAY_SUNDAY

__LL_RTC_GET_YEAR

Description:

- Helper macro to retrieve Year in BCD format.

Parameters:

- __RTC_DATE__: Value returned by

Return value:

- Year: in BCD format (0x00 . . . 0x99)

__LL_RTC_GET_MONTH

Description:

- Helper macro to retrieve Month in BCD format.

Parameters:

- __RTC_DATE__: Value returned by

Return value:

- Returned: value can be one of the following values:
  - LL_RTC_MONTH_JANUARY
  - LL_RTC_MONTH_FEBRUARY
  - LL_RTC_MONTH_MARCH
  - LL_RTC_MONTH_APRIL
  - LL_RTC_MONTH_MAY
  - LL_RTC_MONTH_JUNE
  - LL_RTC_MONTH_JULY
  - LL_RTC_MONTH_AUGUST
  - LL_RTC_MONTH_SEPTEMBER
  - LL_RTC_MONTH_OCTOBER
  - LL_RTC_MONTH_NOVEMBER
  - LL_RTC_MONTH_DECEMBER

__LL_RTC_GET_DAY

Description:

- Helper macro to retrieve Day in BCD format.

Parameters:

- __RTC_DATE__: Value returned by

Return value:

- Day: in BCD format (0x01 . . . 0x31)

Time helper Macros

__LL_RTC_GET_HOUR

Description:

- Helper macro to retrieve hour in BCD format.

Parameters:

- __RTC_TIME__: RTC time returned by

Return value:

- Hours: in BCD format (0x01. . .0x12 or between Min_Data=0x00 and Max_Data=0x23)

__LL_RTC_GET_MINUTE

Description:

- Helper macro to retrieve minute in BCD format.

Parameters:

- __RTC_TIME__: RTC time returned by

Return value:

- Minutes: in BCD format (0x00. . .0x59)

__LL_RTC_GET_SECOND

Description:

- Helper macro to retrieve second in BCD format.

Parameters:

- __RTC_TIME__: RTC time returned by

Return value:

- Seconds: in format (0x00. . .0x59)

Common Write and read registers Macros

LL_RTC_WriteReg

Description:

- Write a value in RTC register.

Parameters:

- __INSTANCE__: RTC Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_RTC_ReadReg

Description:

- Read a value in RTC register.

Parameters:

- __INSTANCE__: RTC Instance
- __REG__: Register to be read

Return value:

- Register: value

104 LL SPI Generic Driver

104.1 SPI Firmware driver registers structures

104.1.1 LL_SPI_InitTypeDef

LL_SPI_InitTypeDef is defined in the stm32n6xx_ll_spi.h

Data Fields

- ***uint32_t TransferDirection***
- ***uint32_t Mode***
- ***uint32_t DataWidth***
- ***uint32_t ClockPolarity***
- ***uint32_t ClockPhase***
- ***uint32_t NSS***
- ***uint32_t BaudRate***
- ***uint32_t BitOrder***
- ***uint32_t CRCCalculation***
- ***uint32_t CRCPoly***

Field Documentation

- ***uint32_t LL_SPI_InitTypeDef::TransferDirection***
  Specifies the SPI unidirectional or bidirectional data mode. This parameter can be a value of ***SPI_LL_EC_TRANSFER_MODE***.This feature can be modified afterwards using unitary function **LL_SPI_SetTransferDirection()**.
- ***uint32_t LL_SPI_InitTypeDef::Mode***
  Specifies the SPI mode (Master/Slave). This parameter can be a value of ***SPI_LL_EC_MODE***.This feature can be modified afterwards using unitary function **LL_SPI_SetMode()**.
- ***uint32_t LL_SPI_InitTypeDef::DataWidth***
  Specifies the SPI data width. This parameter can be a value of ***SPI_LL_EC_DATAWIDTH***.This feature can be modified afterwards using unitary function **LL_SPI_SetDataWidth()**.
- ***uint32_t LL_SPI_InitTypeDef::ClockPolarity***
  Specifies the serial clock steady state. This parameter can be a value of ***SPI_LL_EC_POLARITY***.This feature can be modified afterwards using unitary function **LL_SPI_SetClockPolarity()**.
- ***uint32_t LL_SPI_InitTypeDef::ClockPhase***
  Specifies the clock active edge for the bit capture. This parameter can be a value of ***SPI_LL_EC_PHASE***.This feature can be modified afterwards using unitary function **LL_SPI_SetClockPhase()**.
- ***uint32_t LL_SPI_InitTypeDef::NSS***
  Specifies whether the NSS signal is managed by hardware (NSS pin) or by software using the SSI bit.This parameter can be a value of ***SPI_LL_EC_NSS_MODE***.This feature can be modified afterwards using unitary function **LL_SPI_SetNSSMode()**.
- ***uint32_t LL_SPI_InitTypeDef::BaudRate***
  Specifies the BaudRate prescaler value which will be used to configure the transmit and receive SCK clock. This parameter can be a value of ***SPI_LL_EC_BAUDRATEPRESCALER***.
  Note:
  – The communication clock is derived from the master clock. The slave clock does not need to be set.

  This feature can be modified afterwards using unitary function **LL_SPI_SetBaudRatePrescaler()**.
- ***uint32_t LL_SPI_InitTypeDef::BitOrder***
  Specifies whether data transfers start from MSB or LSB bit. This parameter can be a value of ***SPI_LL_EC_BIT_ORDER***.This feature can be modified afterwards using unitary function **LL_SPI_SetTransferBitOrder()**.
- ***uint32_t LL_SPI_InitTypeDef::CRCCalculation***
  Specifies if the CRC calculation is enabled or not. This parameter can be a value of ***SPI_LL_EC_CRC_CALCULATION***.This feature can be modified afterwards using unitary functions **LL_SPI_EnableCRC()** and **LL_SPI_DisableCRC()**.

- ***uint32_t LL_SPI_InitTypeDef::CRCPoly***
  Specifies the polynomial used for the CRC calculation. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFFFFFFFF.This feature can be modified afterwards using unitary function **LL_SPI_SetCRCPolynomial()**.

104.1.2 LL_I2S_InitTypeDef

LL_I2S_InitTypeDef is defined in the stm32n6xx_ll_spi.h

Data Fields

- ***uint32_t Mode***
- ***uint32_t Standard***
- ***uint32_t DataFormat***
- ***uint32_t MCLKOutput***
- ***uint32_t AudioFreq***
- ***uint32_t ClockPolarity***

Field Documentation

- ***uint32_t LL_I2S_InitTypeDef::Mode***
  Specifies the I2S operating mode. This parameter can be a value of ***I2S_LL_EC_MODE***This feature can be modified afterwards using unitary function **LL_I2S_SetTransferMode()**.
- ***uint32_t LL_I2S_InitTypeDef::Standard***
  Specifies the standard used for the I2S communication. This parameter can be a value of ***I2S_LL_EC_STANDARD***This feature can be modified afterwards using unitary function **LL_I2S_SetStandard()**.
- ***uint32_t LL_I2S_InitTypeDef::DataFormat***
  Specifies the data format for the I2S communication. This parameter can be a value of ***I2S_LL_EC_DATA_FORMAT***This feature can be modified afterwards using unitary function **LL_I2S_SetDataFormat()**.
- ***uint32_t LL_I2S_InitTypeDef::MCLKOutput***
  Specifies whether the I2S MCLK output is enabled or not. This parameter can be a value of ***I2S_LL_EC_MCLK_OUTPUT***This feature can be modified afterwards using unitary functions **LL_I2S_EnableMasterClock()** or **LL_I2S_DisableMasterClock**.
- ***uint32_t LL_I2S_InitTypeDef::AudioFreq***
  Specifies the frequency selected for the I2S communication. This parameter can be a value of ***I2S_LL_EC_AUDIO_FREQ***Audio Frequency can be modified afterwards using Reference manual formulas to calculate Prescaler Linear, Parity and unitary functions **LL_I2S_SetPrescalerLinear()** and **LL_I2S_SetPrescalerParity()** to set it.
- ***uint32_t LL_I2S_InitTypeDef::ClockPolarity***
  Specifies the idle state of the I2S clock. This parameter can be a value of ***I2S_LL_EC_POLARITY***This feature can be modified afterwards using unitary function **LL_I2S_SetClockPolarity()**.

104.2 SPI Firmware driver API description

The following section lists the various functions of the SPI library.

104.2.1 Detailed description of functions

LL_SPI_Enable

Function name

__STATIC_INLINE void LL_SPI_Enable (SPI_TypeDef * SPIx)

Function description

Enable SPI peripheral.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 SPE LL_SPI_Enable

LL_SPI_Disable

Function name

__STATIC_INLINE void LL_SPI_Disable (SPI_TypeDef * SPIx)

Function description

Disable SPI peripheral.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Notes

- When disabling the SPI, follow the procedure described in the Reference Manual.

Reference Manual to LL API cross reference:

- CR1 SPE LL_SPI_Disable

LL_SPI_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabled (const SPI_TypeDef * SPIx)

Function description

Check if SPI peripheral is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- CR1 SPE LL_SPI_IsEnabled

LL_SPI_EnableIOSwap

Function name

__STATIC_INLINE void LL_SPI_EnableIOSwap (SPI_TypeDef * SPIx)

Function description

Swap the MOSI and MISO pin.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled.

Reference Manual to LL API cross reference:

- CFG2 IOSWP LL_SPI_EnableIOSwap

LL_SPI_DisableIOSwap

Function name

__STATIC_INLINE void LL_SPI_DisableIOSwap (SPI_TypeDef * SPIx)

Function description

Restore default function for MOSI and MISO pin.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled.

Reference Manual to LL API cross reference:

- CFG2 IOSWP LL_SPI_DisableIOSwap

LL_SPI_IsEnabledIOSwap

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledIOSwap (const SPI_TypeDef * SPIx)

Function description

Check if MOSI and MISO pin are swapped.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- CFG2 IOSWP LL_SPI_IsEnabledIOSwap

LL_SPI_EnableGPIOControl

Function name

__STATIC_INLINE void LL_SPI_EnableGPIOControl (SPI_TypeDef * SPIx)

Function description

Enable GPIO control.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled.

Reference Manual to LL API cross reference:

- CFG2 AFCNTR LL_SPI_EnableGPIOControl

LL_SPI_DisableGPIOControl

Function name

__STATIC_INLINE void LL_SPI_DisableGPIOControl (SPI_TypeDef * SPIx)

Function description

Disable GPIO control.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled.

Reference Manual to LL API cross reference:

- CFG2 AFCNTR LL_SPI_DisableGPIOControl

LL_SPI_IsEnabledGPIOControl

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledGPIOControl (const SPI_TypeDef * SPIx)

Function description

Check if GPIO control is active.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- CFG2 AFCNTR LL_SPI_IsEnabledGPIOControl

LL_SPI_SetMode

Function name

__STATIC_INLINE void LL_SPI_SetMode (SPI_TypeDef * SPIx, uint32_t Mode)

Function description

Set SPI Mode to Master or Slave.

Parameters

- **SPIx:** SPI Instance
- **Mode:** This parameter can be one of the following values:
  - LL_SPI_MODE_MASTER
  - LL_SPI_MODE_SLAVE

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled.

Reference Manual to LL API cross reference:

- CFG2 MASTER LL_SPI_SetMode

LL_SPI_GetMode

Function name

__STATIC_INLINE uint32_t LL_SPI_GetMode (const SPI_TypeDef * SPIx)

Function description

Get SPI Mode (Master or Slave)

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_MODE_MASTER
  - LL_SPI_MODE_SLAVE

Reference Manual to LL API cross reference:

- CFG2 MASTER LL_SPI_GetMode

LL_SPI_SetMasterSSIdleness

Function name

__STATIC_INLINE void LL_SPI_SetMasterSSIdleness (SPI_TypeDef * SPIx, uint32_t MasterSSIdleness)

Function description

Configure the Idleness applied by master between active edge of SS and first send data.

Parameters

- **SPIx:** SPI Instance
- **MasterSSIdleness:** This parameter can be one of the following values:
  - LL_SPI_SS_IDLENESS_00CYCLE
  - LL_SPI_SS_IDLENESS_01CYCLE
  - LL_SPI_SS_IDLENESS_02CYCLE
  - LL_SPI_SS_IDLENESS_03CYCLE
  - LL_SPI_SS_IDLENESS_04CYCLE
  - LL_SPI_SS_IDLENESS_05CYCLE
  - LL_SPI_SS_IDLENESS_06CYCLE
  - LL_SPI_SS_IDLENESS_07CYCLE
  - LL_SPI_SS_IDLENESS_08CYCLE
  - LL_SPI_SS_IDLENESS_09CYCLE
  - LL_SPI_SS_IDLENESS_10CYCLE
  - LL_SPI_SS_IDLENESS_11CYCLE
  - LL_SPI_SS_IDLENESS_12CYCLE
  - LL_SPI_SS_IDLENESS_13CYCLE
  - LL_SPI_SS_IDLENESS_14CYCLE
  - LL_SPI_SS_IDLENESS_15CYCLE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG2 MSSI LL_SPI_SetMasterSSIdleness

LL_SPI_GetMasterSSIdleness

Function name

__STATIC_INLINE uint32_t LL_SPI_GetMasterSSIdleness (const SPI_TypeDef * SPIx)

Function description

Get the configured Idleness applied by master.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_SS_IDLENESS_00CYCLE
  - LL_SPI_SS_IDLENESS_01CYCLE
  - LL_SPI_SS_IDLENESS_02CYCLE
  - LL_SPI_SS_IDLENESS_03CYCLE
  - LL_SPI_SS_IDLENESS_04CYCLE
  - LL_SPI_SS_IDLENESS_05CYCLE
  - LL_SPI_SS_IDLENESS_06CYCLE
  - LL_SPI_SS_IDLENESS_07CYCLE
  - LL_SPI_SS_IDLENESS_08CYCLE
  - LL_SPI_SS_IDLENESS_09CYCLE
  - LL_SPI_SS_IDLENESS_10CYCLE
  - LL_SPI_SS_IDLENESS_11CYCLE
  - LL_SPI_SS_IDLENESS_12CYCLE
  - LL_SPI_SS_IDLENESS_13CYCLE
  - LL_SPI_SS_IDLENESS_14CYCLE
  - LL_SPI_SS_IDLENESS_15CYCLE

Reference Manual to LL API cross reference:

- CFG2 MSSI LL_SPI_GetMasterSSIdleness

LL_SPI_SetInterDataIdleness

Function name

__STATIC_INLINE void LL_SPI_SetInterDataIdleness (SPI_TypeDef * SPIx, uint32_t MasterInterDataIdleness)

Function description

Configure the idleness applied by master between data frame.

Parameters

- **SPIx:** SPI Instance
- **MasterInterDataIdleness:** This parameter can be one of the following values:
  - LL_SPI_ID_IDLENESS_00CYCLE
  - LL_SPI_ID_IDLENESS_01CYCLE
  - LL_SPI_ID_IDLENESS_02CYCLE
  - LL_SPI_ID_IDLENESS_03CYCLE
  - LL_SPI_ID_IDLENESS_04CYCLE
  - LL_SPI_ID_IDLENESS_05CYCLE
  - LL_SPI_ID_IDLENESS_06CYCLE
  - LL_SPI_ID_IDLENESS_07CYCLE
  - LL_SPI_ID_IDLENESS_08CYCLE
  - LL_SPI_ID_IDLENESS_09CYCLE
  - LL_SPI_ID_IDLENESS_10CYCLE
  - LL_SPI_ID_IDLENESS_11CYCLE
  - LL_SPI_ID_IDLENESS_12CYCLE
  - LL_SPI_ID_IDLENESS_13CYCLE
  - LL_SPI_ID_IDLENESS_14CYCLE
  - LL_SPI_ID_IDLENESS_15CYCLE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG2 MIDI LL_SPI_SetInterDataIdleness

LL_SPI_GetInterDataIdleness

Function name

__STATIC_INLINE uint32_t LL_SPI_GetInterDataIdleness (const SPI_TypeDef * SPIx)

Function description

Get the configured inter data idleness.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_ID_IDLENESS_00CYCLE
  - LL_SPI_ID_IDLENESS_01CYCLE
  - LL_SPI_ID_IDLENESS_02CYCLE
  - LL_SPI_ID_IDLENESS_03CYCLE
  - LL_SPI_ID_IDLENESS_04CYCLE
  - LL_SPI_ID_IDLENESS_05CYCLE
  - LL_SPI_ID_IDLENESS_06CYCLE
  - LL_SPI_ID_IDLENESS_07CYCLE
  - LL_SPI_ID_IDLENESS_08CYCLE
  - LL_SPI_ID_IDLENESS_09CYCLE
  - LL_SPI_ID_IDLENESS_10CYCLE
  - LL_SPI_ID_IDLENESS_11CYCLE
  - LL_SPI_ID_IDLENESS_12CYCLE
  - LL_SPI_ID_IDLENESS_13CYCLE
  - LL_SPI_ID_IDLENESS_14CYCLE
  - LL_SPI_ID_IDLENESS_15CYCLE

Reference Manual to LL API cross reference:

- CFG2 MIDI LL_SPI_SetInterDataIdleness

LL_SPI_SetTransferSize

Function name

__STATIC_INLINE void LL_SPI_SetTransferSize (SPI_TypeDef * SPIx, uint32_t Count)

Function description

Set transfer size.

Parameters

- **SPIx:** SPI Instance
- **Count:** 0..0xFFFF

Return values

- **None:**

Notes

- Count is the number of frame to be transferred

Reference Manual to LL API cross reference:

- CR2 TSIZE LL_SPI_SetTransferSize

LL_SPI_GetTransferSize

Function name

__STATIC_INLINE uint32_t LL_SPI_GetTransferSize (const SPI_TypeDef * SPIx)

Function description

Get transfer size.

Parameters

- **SPIx:** SPI Instance

Return values

- **0..0xFFFF:**

Notes

- Count is the number of frame to be transferred

Reference Manual to LL API cross reference:

- CR2 TSIZE LL_SPI_GetTransferSize

LL_SPI_EnableIOLock

Function name

__STATIC_INLINE void LL_SPI_EnableIOLock (SPI_TypeDef * SPIx)

Function description

Lock the AF configuration of associated IOs.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Notes

- Once this bit is set, the AF configuration remains locked until a hardware reset occurs. the reset of the IOLock bit is done by hardware. for that, LL_SPI_DisableIOLock can not exist.

Reference Manual to LL API cross reference:

- CR1 IOLOCK LL_SPI_EnableIOLock

LL_SPI_IsEnabledIOLock

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledIOLock (const SPI_TypeDef * SPIx)

Function description

Check if the AF configuration is locked.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- CR1 IOLOCK LL_SPI_IsEnabledIOLock

LL_SPI_SetTxCRCInitPattern

Function name

__STATIC_INLINE void LL_SPI_SetTxCRCInitPattern (SPI_TypeDef * SPIx, uint32_t TXCRCInitAll)

Function description

Set Tx CRC Initialization Pattern.

Parameters

- **SPIx:** SPI Instance
- **TXCRCInitAll:** This parameter can be one of the following values:
  - LL_SPI_TXCRCINIT_ALL_ZERO_PATTERN
  - LL_SPI_TXCRCINIT_ALL_ONES_PATTERN

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TCRCINI LL_SPI_SetTxCRCInitPattern

LL_SPI_GetTxCRCInitPattern

Function name

__STATIC_INLINE uint32_t LL_SPI_GetTxCRCInitPattern (const SPI_TypeDef * SPIx)

Function description

Get Tx CRC Initialization Pattern.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_TXCRCINIT_ALL_ZERO_PATTERN
  - LL_SPI_TXCRCINIT_ALL_ONES_PATTERN

Reference Manual to LL API cross reference:

- CR1 TCRCINI LL_SPI_GetTxCRCInitPattern

LL_SPI_SetRxCRCInitPattern

Function name

__STATIC_INLINE void LL_SPI_SetRxCRCInitPattern (SPI_TypeDef * SPIx, uint32_t RXCRCInitAll)

Function description

Set Rx CRC Initialization Pattern.

Parameters

- **SPIx:** SPI Instance
- **RXCRCInitAll:** This parameter can be one of the following values:
  - LL_SPI_RXCRCINIT_ALL_ZERO_PATTERN
  - LL_SPI_RXCRCINIT_ALL_ONES_PATTERN

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RCRCINI LL_SPI_SetRxCRCInitPattern

LL_SPI_GetRxCRCInitPattern

Function name

__STATIC_INLINE uint32_t LL_SPI_GetRxCRCInitPattern (const SPI_TypeDef * SPIx)

Function description

Get Rx CRC Initialization Pattern.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_RXCRCINIT_ALL_ZERO_PATTERN
  - LL_SPI_RXCRCINIT_ALL_ONES_PATTERN

Reference Manual to LL API cross reference:

- CR1 RCRCINI LL_SPI_GetRxCRCInitPattern

LL_SPI_SetInternalSSLevel

Function name

__STATIC_INLINE void LL_SPI_SetInternalSSLevel (SPI_TypeDef * SPIx, uint32_t SSLevel)

Function description

Set internal SS input level ignoring what comes from PIN.

Parameters

- **SPIx:** SPI Instance
- **SSLevel:** This parameter can be one of the following values:
  - LL_SPI_SS_LEVEL_HIGH
  - LL_SPI_SS_LEVEL_LOW

Return values

- **None:**

Notes

- This configuration has effect only with config LL_SPI_NSS_SOFT

Reference Manual to LL API cross reference:

- CR1 SSI LL_SPI_SetInternalSSLevel

LL_SPI_GetInternalSSLevel

Function name

__STATIC_INLINE uint32_t LL_SPI_GetInternalSSLevel (const SPI_TypeDef * SPIx)

Function description

Get internal SS input level.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_SS_LEVEL_HIGH
  - LL_SPI_SS_LEVEL_LOW

Reference Manual to LL API cross reference:

- CR1 SSI LL_SPI_GetInternalSSLevel

LL_SPI_EnableFullSizeCRC

Function name

__STATIC_INLINE void LL_SPI_EnableFullSizeCRC (SPI_TypeDef * SPIx)

Function description

Enable CRC computation on 33/17 bits.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 CRC33_17 LL_SPI_EnableFullSizeCRC

LL_SPI_DisableFullSizeCRC

Function name

__STATIC_INLINE void LL_SPI_DisableFullSizeCRC (SPI_TypeDef * SPIx)

Function description

Disable CRC computation on 33/17 bits.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 CRC33_17 LL_SPI_DisableFullSizeCRC

LL_SPI_IsEnabledFullSizeCRC

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledFullSizeCRC (const SPI_TypeDef * SPIx)

Function description

Check if Enable CRC computation on 33/17 bits is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- CR1 CRC33_17 LL_SPI_IsEnabledFullSizeCRC

LL_SPI_SuspendMasterTransfer

Function name

__STATIC_INLINE void LL_SPI_SuspendMasterTransfer (SPI_TypeDef * SPIx)

Function description

Suspend an ongoing transfer for Master configuration.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 CSUSP LL_SPI_SuspendMasterTransfer

LL_SPI_StartMasterTransfer

Function name

__STATIC_INLINE void LL_SPI_StartMasterTransfer (SPI_TypeDef * SPIx)

Function description

Start effective transfer on wire for Master configuration.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 CSTART LL_SPI_StartMasterTransfer

LL_SPI_IsActiveMasterTransfer

Function name

__STATIC_INLINE uint32_t LL_SPI_IsActiveMasterTransfer (const SPI_TypeDef * SPIx)

Function description

Check if there is an unfinished master transfer.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- CR1 CSTART LL_SPI_IsActiveMasterTransfer

LL_SPI_EnableMasterRxAutoSuspend

Function name

__STATIC_INLINE void LL_SPI_EnableMasterRxAutoSuspend (SPI_TypeDef * SPIx)

Function description

Enable Master Rx auto suspend in case of overrun.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 MASRX LL_SPI_EnableMasterRxAutoSuspend

LL_SPI_DisableMasterRxAutoSuspend

Function name

__STATIC_INLINE void LL_SPI_DisableMasterRxAutoSuspend (SPI_TypeDef * SPIx)

Function description

Disable Master Rx auto suspend in case of overrun.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 MASRX LL_SPI_DisableMasterRxAutoSuspend

LL_SPI_IsEnabledMasterRxAutoSuspend

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledMasterRxAutoSuspend (const SPI_TypeDef * SPIx)

Function description

Check if Master Rx auto suspend is activated.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- CR1 MASRX LL_SPI_IsEnabledMasterRxAutoSuspend

LL_SPI_SetUDRConfiguration

Function name

__STATIC_INLINE void LL_SPI_SetUDRConfiguration (SPI_TypeDef * SPIx, uint32_t UDRConfig)

Function description

Set Underrun behavior.

Parameters

- **SPIx:** SPI Instance
- **UDRConfig:** This parameter can be one of the following values:
  - LL_SPI_UDR_CONFIG_REGISTER_PATTERN
  - LL_SPI_UDR_CONFIG_LAST_RECEIVED

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled.

Reference Manual to LL API cross reference:

- CFG1 UDRCFG LL_SPI_SetUDRConfiguration

LL_SPI_GetUDRConfiguration

Function name

__STATIC_INLINE uint32_t LL_SPI_GetUDRConfiguration (const SPI_TypeDef * SPIx)

Function description

Get Underrun behavior.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_UDR_CONFIG_REGISTER_PATTERN
  - LL_SPI_UDR_CONFIG_LAST_RECEIVED

Reference Manual to LL API cross reference:

- CFG1 UDRCFG LL_SPI_GetUDRConfiguration

LL_SPI_SetStandard

Function name

__STATIC_INLINE void LL_SPI_SetStandard (SPI_TypeDef * SPIx, uint32_t Standard)

Function description

Set Serial protocol used.

Parameters

- **SPIx:** SPI Instance
- **Standard:** This parameter can be one of the following values:
  - LL_SPI_PROTOCOL_MOTOROLA
  - LL_SPI_PROTOCOL_TI

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled.

Reference Manual to LL API cross reference:

- CFG2 SP LL_SPI_SetStandard

LL_SPI_GetStandard

Function name

__STATIC_INLINE uint32_t LL_SPI_GetStandard (const SPI_TypeDef * SPIx)

Function description

Get Serial protocol used.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_PROTOCOL_MOTOROLA
  - LL_SPI_PROTOCOL_TI

Reference Manual to LL API cross reference:

- CFG2 SP LL_SPI_GetStandard

LL_SPI_SetClockPhase

Function name

__STATIC_INLINE void LL_SPI_SetClockPhase (SPI_TypeDef * SPIx, uint32_t ClockPhase)

Function description

Set Clock phase.

Parameters

- **SPIx:** SPI Instance
- **ClockPhase:** This parameter can be one of the following values:
  - LL_SPI_PHASE_1EDGE
  - LL_SPI_PHASE_2EDGE

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled. This bit is not used in SPI TI mode.

Reference Manual to LL API cross reference:

- CFG2 CPHA LL_SPI_SetClockPhase

LL_SPI_GetClockPhase

Function name

__STATIC_INLINE uint32_t LL_SPI_GetClockPhase (const SPI_TypeDef * SPIx)

Function description

Get Clock phase.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_PHASE_1EDGE
  - LL_SPI_PHASE_2EDGE

Reference Manual to LL API cross reference:

- CFG2 CPHA LL_SPI_GetClockPhase

LL_SPI_SetClockPolarity

Function name

__STATIC_INLINE void LL_SPI_SetClockPolarity (SPI_TypeDef * SPIx, uint32_t ClockPolarity)

Function description

Set Clock polarity.

Parameters

- **SPIx:** SPI Instance
- **ClockPolarity:** This parameter can be one of the following values:
  - LL_SPI_POLARITY_LOW
  - LL_SPI_POLARITY_HIGH

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled. This bit is not used in SPI TI mode.

Reference Manual to LL API cross reference:

- CFG2 CPOL LL_SPI_SetClockPolarity

LL_SPI_GetClockPolarity

Function name

__STATIC_INLINE uint32_t LL_SPI_GetClockPolarity (const SPI_TypeDef * SPIx)

Function description

Get Clock polarity.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_POLARITY_LOW
  - LL_SPI_POLARITY_HIGH

Reference Manual to LL API cross reference:

- CFG2 CPOL LL_SPI_GetClockPolarity

LL_SPI_SetNSSPolarity

Function name

__STATIC_INLINE void LL_SPI_SetNSSPolarity (SPI_TypeDef * SPIx, uint32_t NSSPolarity)

Function description

Set NSS polarity.

Parameters

- **SPIx:** SPI Instance
- **NSSPolarity:** This parameter can be one of the following values:
  - LL_SPI_NSS_POLARITY_LOW
  - LL_SPI_NSS_POLARITY_HIGH

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled. This bit is not used in SPI TI mode.

Reference Manual to LL API cross reference:

- CFG2 SSIOP LL_SPI_SetNSSPolarity

LL_SPI_GetNSSPolarity

Function name

__STATIC_INLINE uint32_t LL_SPI_GetNSSPolarity (const SPI_TypeDef * SPIx)

Function description

Get NSS polarity.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_NSS_POLARITY_LOW
  - LL_SPI_NSS_POLARITY_HIGH

Reference Manual to LL API cross reference:

- CFG2 SSIOP LL_SPI_GetNSSPolarity

LL_SPI_SetBaudRatePrescaler

Function name

__STATIC_INLINE void LL_SPI_SetBaudRatePrescaler (SPI_TypeDef * SPIx, uint32_t Baudrate)

Function description

Set Baudrate Prescaler.

Parameters

- **SPIx:** SPI Instance
- **Baudrate:** This parameter can be one of the following values:
  - LL_SPI_BAUDRATEPRESCALER_BYPASS
  - LL_SPI_BAUDRATEPRESCALER_DIV2
  - LL_SPI_BAUDRATEPRESCALER_DIV4
  - LL_SPI_BAUDRATEPRESCALER_DIV8
  - LL_SPI_BAUDRATEPRESCALER_DIV16
  - LL_SPI_BAUDRATEPRESCALER_DIV32
  - LL_SPI_BAUDRATEPRESCALER_DIV64
  - LL_SPI_BAUDRATEPRESCALER_DIV128
  - LL_SPI_BAUDRATEPRESCALER_DIV256

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled. SPI BaudRate = fPCLK/Pescaler.

Reference Manual to LL API cross reference:

- CFG1 MBR LL_SPI_SetBaudRatePrescaler
- CFG1 BPASS LL_SPI_SetBaudRatePrescaler

LL_SPI_GetBaudRatePrescaler

Function name

__STATIC_INLINE uint32_t LL_SPI_GetBaudRatePrescaler (const SPI_TypeDef * SPIx)

Function description

Get Baudrate Prescaler.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_BAUDRATEPRESCALER_BYPASS
  - LL_SPI_BAUDRATEPRESCALER_DIV2
  - LL_SPI_BAUDRATEPRESCALER_DIV4
  - LL_SPI_BAUDRATEPRESCALER_DIV8
  - LL_SPI_BAUDRATEPRESCALER_DIV16
  - LL_SPI_BAUDRATEPRESCALER_DIV32
  - LL_SPI_BAUDRATEPRESCALER_DIV64
  - LL_SPI_BAUDRATEPRESCALER_DIV128
  - LL_SPI_BAUDRATEPRESCALER_DIV256

Reference Manual to LL API cross reference:

- CFG1 MBR LL_SPI_GetBaudRatePrescaler
- CFG1 BPASS LL_SPI_GetBaudRatePrescaler

LL_SPI_SetTransferBitOrder

Function name

__STATIC_INLINE void LL_SPI_SetTransferBitOrder (SPI_TypeDef * SPIx, uint32_t BitOrder)

Function description

Set Transfer Bit Order.

Parameters

- **SPIx:** SPI Instance
- **BitOrder:** This parameter can be one of the following values:
  - LL_SPI_LSB_FIRST
  - LL_SPI_MSB_FIRST

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled. This bit is not used in SPI TI mode.

Reference Manual to LL API cross reference:

- CFG2 LSBFRST LL_SPI_SetTransferBitOrder

LL_SPI_GetTransferBitOrder

Function name

__STATIC_INLINE uint32_t LL_SPI_GetTransferBitOrder (const SPI_TypeDef * SPIx)

Function description

Get Transfer Bit Order.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_LSB_FIRST
  - LL_SPI_MSB_FIRST

Reference Manual to LL API cross reference:

- CFG2 LSBFRST LL_SPI_GetTransferBitOrder

LL_SPI_SetTransferDirection

Function name

__STATIC_INLINE void LL_SPI_SetTransferDirection (SPI_TypeDef * SPIx, uint32_t TransferDirection)

Function description

Set Transfer Mode.

Parameters

- **SPIx:** SPI Instance
- **TransferDirection:** This parameter can be one of the following values:
  - LL_SPI_FULL_DUPLEX
  - LL_SPI_SIMPLEX_TX
  - LL_SPI_SIMPLEX_RX
  - LL_SPI_HALF_DUPLEX_RX
  - LL_SPI_HALF_DUPLEX_TX

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled except for half duplex direction using LL_SPI_SetHalfDuplexDirection.

Reference Manual to LL API cross reference:

- CR1 HDDIR LL_SPI_SetTransferDirection
- CFG2 COMM LL_SPI_SetTransferDirection

LL_SPI_GetTransferDirection

Function name

__STATIC_INLINE uint32_t LL_SPI_GetTransferDirection (const SPI_TypeDef * SPIx)

Function description

Get Transfer Mode.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_FULL_DUPLEX
  - LL_SPI_SIMPLEX_TX
  - LL_SPI_SIMPLEX_RX
  - LL_SPI_HALF_DUPLEX_RX
  - LL_SPI_HALF_DUPLEX_TX

Reference Manual to LL API cross reference:

- CR1 HDDIR LL_SPI_GetTransferDirection
- CFG2 COMM LL_SPI_GetTransferDirection

LL_SPI_SetHalfDuplexDirection

Function name

__STATIC_INLINE void LL_SPI_SetHalfDuplexDirection (SPI_TypeDef * SPIx, uint32_t HalfDuplexDirection)

Function description

Set direction for Half-Duplex Mode.

Parameters

- **SPIx:** SPI Instance
- **HalfDuplexDirection:** This parameter can be one of the following values:
  - LL_SPI_HALF_DUPLEX_RX
  - LL_SPI_HALF_DUPLEX_TX

Return values

- **None:**

Notes

- In master mode the MOSI pin is used and in slave mode the MISO pin is used for Half-Duplex.

Reference Manual to LL API cross reference:

- CR1 HDDIR LL_SPI_SetHalfDuplexDirection

LL_SPI_GetHalfDuplexDirection

Function name

__STATIC_INLINE uint32_t LL_SPI_GetHalfDuplexDirection (const SPI_TypeDef * SPIx)

Function description

Get direction for Half-Duplex Mode.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_HALF_DUPLEX_RX
  - LL_SPI_HALF_DUPLEX_TX

Notes

- In master mode the MOSI pin is used and in slave mode the MISO pin is used for Half-Duplex.

Reference Manual to LL API cross reference:

- CR1 HDDIR LL_SPI_GetHalfDuplexDirection

LL_SPI_SetDataWidth

Function name

__STATIC_INLINE void LL_SPI_SetDataWidth (SPI_TypeDef * SPIx, uint32_t DataWidth)

Function description

Set Frame Data Size.

Parameters

- **SPIx:** SPI Instance
- **DataWidth:** This parameter can be one of the following values:
  - LL_SPI_DATAWIDTH_4BIT
  - LL_SPI_DATAWIDTH_5BIT
  - LL_SPI_DATAWIDTH_6BIT
  - LL_SPI_DATAWIDTH_7BIT
  - LL_SPI_DATAWIDTH_8BIT
  - LL_SPI_DATAWIDTH_9BIT
  - LL_SPI_DATAWIDTH_10BIT
  - LL_SPI_DATAWIDTH_11BIT
  - LL_SPI_DATAWIDTH_12BIT
  - LL_SPI_DATAWIDTH_13BIT
  - LL_SPI_DATAWIDTH_14BIT
  - LL_SPI_DATAWIDTH_15BIT
  - LL_SPI_DATAWIDTH_16BIT
  - LL_SPI_DATAWIDTH_17BIT
  - LL_SPI_DATAWIDTH_18BIT
  - LL_SPI_DATAWIDTH_19BIT
  - LL_SPI_DATAWIDTH_20BIT
  - LL_SPI_DATAWIDTH_21BIT
  - LL_SPI_DATAWIDTH_22BIT
  - LL_SPI_DATAWIDTH_23BIT
  - LL_SPI_DATAWIDTH_24BIT
  - LL_SPI_DATAWIDTH_25BIT
  - LL_SPI_DATAWIDTH_26BIT
  - LL_SPI_DATAWIDTH_27BIT
  - LL_SPI_DATAWIDTH_28BIT
  - LL_SPI_DATAWIDTH_29BIT
  - LL_SPI_DATAWIDTH_30BIT
  - LL_SPI_DATAWIDTH_31BIT
  - LL_SPI_DATAWIDTH_32BIT

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled.

Reference Manual to LL API cross reference:

- CFG1 DSIZE LL_SPI_SetDataWidth

LL_SPI_GetDataWidth

Function name

__STATIC_INLINE uint32_t LL_SPI_GetDataWidth (const SPI_TypeDef * SPIx)

Function description

Get Frame Data Size.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_DATAWIDTH_4BIT
  - LL_SPI_DATAWIDTH_5BIT
  - LL_SPI_DATAWIDTH_6BIT
  - LL_SPI_DATAWIDTH_7BIT
  - LL_SPI_DATAWIDTH_8BIT
  - LL_SPI_DATAWIDTH_9BIT
  - LL_SPI_DATAWIDTH_10BIT
  - LL_SPI_DATAWIDTH_11BIT
  - LL_SPI_DATAWIDTH_12BIT
  - LL_SPI_DATAWIDTH_13BIT
  - LL_SPI_DATAWIDTH_14BIT
  - LL_SPI_DATAWIDTH_15BIT
  - LL_SPI_DATAWIDTH_16BIT
  - LL_SPI_DATAWIDTH_17BIT
  - LL_SPI_DATAWIDTH_18BIT
  - LL_SPI_DATAWIDTH_19BIT
  - LL_SPI_DATAWIDTH_20BIT
  - LL_SPI_DATAWIDTH_21BIT
  - LL_SPI_DATAWIDTH_22BIT
  - LL_SPI_DATAWIDTH_23BIT
  - LL_SPI_DATAWIDTH_24BIT
  - LL_SPI_DATAWIDTH_25BIT
  - LL_SPI_DATAWIDTH_26BIT
  - LL_SPI_DATAWIDTH_27BIT
  - LL_SPI_DATAWIDTH_28BIT
  - LL_SPI_DATAWIDTH_29BIT
  - LL_SPI_DATAWIDTH_30BIT
  - LL_SPI_DATAWIDTH_31BIT
  - LL_SPI_DATAWIDTH_32BIT

Reference Manual to LL API cross reference:

- CFG1 DSIZE LL_SPI_GetDataWidth

LL_SPI_SetFIFOThreshold

Function name

__STATIC_INLINE void LL_SPI_SetFIFOThreshold (SPI_TypeDef * SPIx, uint32_t Threshold)

Function description

Set threshold of FIFO that triggers a transfer event.

Parameters

- **SPIx:** SPI Instance
- **Threshold:** This parameter can be one of the following values:
  - LL_SPI_FIFO_TH_01DATA
  - LL_SPI_FIFO_TH_02DATA
  - LL_SPI_FIFO_TH_03DATA
  - LL_SPI_FIFO_TH_04DATA
  - LL_SPI_FIFO_TH_05DATA
  - LL_SPI_FIFO_TH_06DATA
  - LL_SPI_FIFO_TH_07DATA
  - LL_SPI_FIFO_TH_08DATA
  - LL_SPI_FIFO_TH_09DATA
  - LL_SPI_FIFO_TH_10DATA
  - LL_SPI_FIFO_TH_11DATA
  - LL_SPI_FIFO_TH_12DATA
  - LL_SPI_FIFO_TH_13DATA
  - LL_SPI_FIFO_TH_14DATA
  - LL_SPI_FIFO_TH_15DATA
  - LL_SPI_FIFO_TH_16DATA

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled.

Reference Manual to LL API cross reference:

- CFG1 FTHLV LL_SPI_SetFIFOThreshold

LL_SPI_GetFIFOThreshold

Function name

__STATIC_INLINE uint32_t LL_SPI_GetFIFOThreshold (const SPI_TypeDef * SPIx)

Function description

Get threshold of FIFO that triggers a transfer event.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_FIFO_TH_01DATA
  - LL_SPI_FIFO_TH_02DATA
  - LL_SPI_FIFO_TH_03DATA
  - LL_SPI_FIFO_TH_04DATA
  - LL_SPI_FIFO_TH_05DATA
  - LL_SPI_FIFO_TH_06DATA
  - LL_SPI_FIFO_TH_07DATA
  - LL_SPI_FIFO_TH_08DATA
  - LL_SPI_FIFO_TH_09DATA
  - LL_SPI_FIFO_TH_10DATA
  - LL_SPI_FIFO_TH_11DATA
  - LL_SPI_FIFO_TH_12DATA
  - LL_SPI_FIFO_TH_13DATA
  - LL_SPI_FIFO_TH_14DATA
  - LL_SPI_FIFO_TH_15DATA
  - LL_SPI_FIFO_TH_16DATA

Reference Manual to LL API cross reference:

- CFG1 FTHLV LL_SPI_GetFIFOThreshold

LL_SPI_EnableCRC

Function name

__STATIC_INLINE void LL_SPI_EnableCRC (SPI_TypeDef * SPIx)

Function description

Enable CRC.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled.

Reference Manual to LL API cross reference:

- CFG1 CRCEN LL_SPI_EnableCRC

LL_SPI_DisableCRC

Function name

__STATIC_INLINE void LL_SPI_DisableCRC (SPI_TypeDef * SPIx)

Function description

Disable CRC.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG1 CRCEN LL_SPI_DisableCRC

LL_SPI_IsEnabledCRC

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledCRC (const SPI_TypeDef * SPIx)

Function description

Check if CRC is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CFG1 CRCEN LL_SPI_IsEnabledCRC

LL_SPI_SetCRCWidth

Function name

__STATIC_INLINE void LL_SPI_SetCRCWidth (SPI_TypeDef * SPIx, uint32_t CRCLength)

Function description

Set CRC Length.

Parameters

- **SPIx:** SPI Instance
- **CRCLength:** This parameter can be one of the following values:
  - LL_SPI_CRC_4BIT
  - LL_SPI_CRC_5BIT
  - LL_SPI_CRC_6BIT
  - LL_SPI_CRC_7BIT
  - LL_SPI_CRC_8BIT
  - LL_SPI_CRC_9BIT
  - LL_SPI_CRC_10BIT
  - LL_SPI_CRC_11BIT
  - LL_SPI_CRC_12BIT
  - LL_SPI_CRC_13BIT
  - LL_SPI_CRC_14BIT
  - LL_SPI_CRC_15BIT
  - LL_SPI_CRC_16BIT
  - LL_SPI_CRC_17BIT
  - LL_SPI_CRC_18BIT
  - LL_SPI_CRC_19BIT
  - LL_SPI_CRC_20BIT
  - LL_SPI_CRC_21BIT
  - LL_SPI_CRC_22BIT
  - LL_SPI_CRC_23BIT
  - LL_SPI_CRC_24BIT
  - LL_SPI_CRC_25BIT
  - LL_SPI_CRC_26BIT
  - LL_SPI_CRC_27BIT
  - LL_SPI_CRC_28BIT
  - LL_SPI_CRC_29BIT
  - LL_SPI_CRC_30BIT
  - LL_SPI_CRC_31BIT
  - LL_SPI_CRC_32BIT

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled.

Reference Manual to LL API cross reference:

- CFG1 CRCSIZE LL_SPI_SetCRCWidth

LL_SPI_GetCRCWidth

Function name

__STATIC_INLINE uint32_t LL_SPI_GetCRCWidth (const SPI_TypeDef * SPIx)

Function description

Get CRC Length.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_CRC_4BIT
  - LL_SPI_CRC_5BIT
  - LL_SPI_CRC_6BIT
  - LL_SPI_CRC_7BIT
  - LL_SPI_CRC_8BIT
  - LL_SPI_CRC_9BIT
  - LL_SPI_CRC_10BIT
  - LL_SPI_CRC_11BIT
  - LL_SPI_CRC_12BIT
  - LL_SPI_CRC_13BIT
  - LL_SPI_CRC_14BIT
  - LL_SPI_CRC_15BIT
  - LL_SPI_CRC_16BIT
  - LL_SPI_CRC_17BIT
  - LL_SPI_CRC_18BIT
  - LL_SPI_CRC_19BIT
  - LL_SPI_CRC_20BIT
  - LL_SPI_CRC_21BIT
  - LL_SPI_CRC_22BIT
  - LL_SPI_CRC_23BIT
  - LL_SPI_CRC_24BIT
  - LL_SPI_CRC_25BIT
  - LL_SPI_CRC_26BIT
  - LL_SPI_CRC_27BIT
  - LL_SPI_CRC_28BIT
  - LL_SPI_CRC_29BIT
  - LL_SPI_CRC_30BIT
  - LL_SPI_CRC_31BIT
  - LL_SPI_CRC_32BIT

Reference Manual to LL API cross reference:

- CFG1 CRCSIZE LL_SPI_GetCRCWidth

LL_SPI_SetNSSMode

Function name

__STATIC_INLINE void LL_SPI_SetNSSMode (SPI_TypeDef * SPIx, uint32_t NSS)

Function description

Set NSS Mode.

Parameters

- **SPIx:** SPI Instance
- **NSS:** This parameter can be one of the following values:
  - LL_SPI_NSS_SOFT
  - LL_SPI_NSS_HARD_INPUT
  - LL_SPI_NSS_HARD_OUTPUT

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled. This bit is not used in SPI TI mode.

Reference Manual to LL API cross reference:

- CFG2 SSM LL_SPI_SetNSSMode
- CFG2 SSOE LL_SPI_SetNSSMode

LL_SPI_GetNSSMode

Function name

__STATIC_INLINE uint32_t LL_SPI_GetNSSMode (const SPI_TypeDef * SPIx)

Function description

Set NSS Mode.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_NSS_SOFT
  - LL_SPI_NSS_HARD_INPUT
  - LL_SPI_NSS_HARD_OUTPUT

Reference Manual to LL API cross reference:

- CFG2 SSM LL_SPI_GetNSSMode
- CFG2 SSOE LL_SPI_GetNSSMode

LL_SPI_EnableNSSPulseMgt

Function name

__STATIC_INLINE void LL_SPI_EnableNSSPulseMgt (SPI_TypeDef * SPIx)

Function description

Enable NSS pulse mgt.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled. This bit is not used in SPI TI mode.

Reference Manual to LL API cross reference:

- CFG2 SSOM LL_SPI_EnableNSSPulseMgt

LL_SPI_DisableNSSPulseMgt

Function name

__STATIC_INLINE void LL_SPI_DisableNSSPulseMgt (SPI_TypeDef * SPIx)

Function description

Disable NSS pulse mgt.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled. This bit is not used in SPI TI mode.

Reference Manual to LL API cross reference:

- CFG2 SSOM LL_SPI_DisableNSSPulseMgt

LL_SPI_IsEnabledNSSPulse

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledNSSPulse (const SPI_TypeDef * SPIx)

Function description

Check if NSS pulse is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- CFG2 SSOM LL_SPI_IsEnabledNSSPulse

LL_SPI_IsActiveFlag_RXP

Function name

__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_RXP (const SPI_TypeDef * SPIx)

Function description

Check if there is enough data in FIFO to read a full packet.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- SR RXP LL_SPI_IsActiveFlag_RXP

LL_SPI_IsActiveFlag_TXP

Function name

__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_TXP (const SPI_TypeDef * SPIx)

Function description

Check if there is enough space in FIFO to hold a full packet.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- SR TXP LL_SPI_IsActiveFlag_TXP

LL_SPI_IsActiveFlag_DXP

Function name

__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_DXP (const SPI_TypeDef * SPIx)

Function description

Check if there enough space in FIFO to hold a full packet, AND enough data to read a full packet.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- SR DXP LL_SPI_IsActiveFlag_DXP

LL_SPI_IsActiveFlag_EOT

Function name

__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_EOT (const SPI_TypeDef * SPIx)

Function description

Check that end of transfer event occurred.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR EOT LL_SPI_IsActiveFlag_EOT

LL_SPI_IsActiveFlag_TXTF

Function name

__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_TXTF (const SPI_TypeDef * SPIx)

Function description

Check that all required data has been filled in the fifo according to transfer size.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR TXTF LL_SPI_IsActiveFlag_TXTF

LL_SPI_IsActiveFlag_UDR

Function name

__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_UDR (const SPI_TypeDef * SPIx)

Function description

Get Underrun error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR UDR LL_SPI_IsActiveFlag_UDR

LL_SPI_IsActiveFlag_CRCERR

Function name

__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_CRCERR (const SPI_TypeDef * SPIx)

Function description

Get CRC error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR CRCE LL_SPI_IsActiveFlag_CRCERR

LL_SPI_IsActiveFlag_MODF

Function name

__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_MODF (const SPI_TypeDef * SPIx)

Function description

Get Mode fault error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR MODF LL_SPI_IsActiveFlag_MODF

LL_SPI_IsActiveFlag_OVR

Function name

__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_OVR (const SPI_TypeDef * SPIx)

Function description

Get Overrun error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR OVR LL_SPI_IsActiveFlag_OVR

LL_SPI_IsActiveFlag_FRE

Function name

__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_FRE (const SPI_TypeDef * SPIx)

Function description

Get TI Frame format error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR TIFRE LL_SPI_IsActiveFlag_FRE

LL_SPI_IsActiveFlag_SUSP

Function name

__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_SUSP (const SPI_TypeDef * SPIx)

Function description

Check if a suspend operation is done.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- SR SUSP LL_SPI_IsActiveFlag_SUSP

LL_SPI_IsActiveFlag_TXC

Function name

__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_TXC (const SPI_TypeDef * SPIx)

Function description

Check if last TxFIFO or CRC frame transmission is completed.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR TXC LL_SPI_IsActiveFlag_TXC

LL_SPI_IsActiveFlag_RXWNE

Function name

__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_RXWNE (const SPI_TypeDef * SPIx)

Function description

Check if at least one 32-bit data is available in RxFIFO.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- SR RXWNE LL_SPI_IsActiveFlag_RXWNE

LL_SPI_GetRemainingDataFrames

Function name

__STATIC_INLINE uint32_t LL_SPI_GetRemainingDataFrames (const SPI_TypeDef * SPIx)

Function description

Get number of data framed remaining in current TSIZE.

Parameters

- **SPIx:** SPI Instance

Return values

- **0..0xFFFF:**

Reference Manual to LL API cross reference:

- SR CTSIZE LL_SPI_GetRemainingDataFrames

LL_SPI_GetRxFIFOPackingLevel

Function name

__STATIC_INLINE uint32_t LL_SPI_GetRxFIFOPackingLevel (const SPI_TypeDef * SPIx)

Function description

Get RxFIFO packing Level.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_RX_FIFO_0PACKET
  - LL_SPI_RX_FIFO_1PACKET
  - LL_SPI_RX_FIFO_2PACKET
  - LL_SPI_RX_FIFO_3PACKET

Reference Manual to LL API cross reference:

- SR RXPLVL LL_SPI_GetRxFIFOPackingLevel

LL_SPI_ClearFlag_EOT

Function name

__STATIC_INLINE void LL_SPI_ClearFlag_EOT (SPI_TypeDef * SPIx)

Function description

Clear End Of Transfer flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR EOTC LL_SPI_ClearFlag_EOT

LL_SPI_ClearFlag_TXTF

Function name

__STATIC_INLINE void LL_SPI_ClearFlag_TXTF (SPI_TypeDef * SPIx)

Function description

Clear TXTF flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR TXTFC LL_SPI_ClearFlag_TXTF

LL_SPI_ClearFlag_UDR

Function name

__STATIC_INLINE void LL_SPI_ClearFlag_UDR (SPI_TypeDef * SPIx)

Function description

Clear Underrun error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR UDRC LL_SPI_ClearFlag_UDR

LL_SPI_ClearFlag_OVR

Function name

__STATIC_INLINE void LL_SPI_ClearFlag_OVR (SPI_TypeDef * SPIx)

Function description

Clear Overrun error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR OVRC LL_SPI_ClearFlag_OVR

LL_SPI_ClearFlag_CRCERR

Function name

__STATIC_INLINE void LL_SPI_ClearFlag_CRCERR (SPI_TypeDef * SPIx)

Function description

Clear CRC error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CRCEC LL_SPI_ClearFlag_CRCERR

LL_SPI_ClearFlag_MODF

Function name

__STATIC_INLINE void LL_SPI_ClearFlag_MODF (SPI_TypeDef * SPIx)

Function description

Clear Mode fault error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR MODFC LL_SPI_ClearFlag_MODF

LL_SPI_ClearFlag_FRE

Function name

__STATIC_INLINE void LL_SPI_ClearFlag_FRE (SPI_TypeDef * SPIx)

Function description

Clear Frame format error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR TIFREC LL_SPI_ClearFlag_FRE

LL_SPI_ClearFlag_SUSP

Function name

__STATIC_INLINE void LL_SPI_ClearFlag_SUSP (SPI_TypeDef * SPIx)

Function description

Clear SUSP flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR SUSPC LL_SPI_ClearFlag_SUSP

LL_SPI_EnableIT_RXP

Function name

__STATIC_INLINE void LL_SPI_EnableIT_RXP (SPI_TypeDef * SPIx)

Function description

Enable Rx Packet available IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER RXPIE LL_SPI_EnableIT_RXP

LL_SPI_EnableIT_TXP

Function name

__STATIC_INLINE void LL_SPI_EnableIT_TXP (SPI_TypeDef * SPIx)

Function description

Enable Tx Packet space available IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER TXPIE LL_SPI_EnableIT_TXP

LL_SPI_EnableIT_DXP

Function name

__STATIC_INLINE void LL_SPI_EnableIT_DXP (SPI_TypeDef * SPIx)

Function description

Enable Duplex Packet available IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER DXPIE LL_SPI_EnableIT_DXP

LL_SPI_EnableIT_EOT

Function name

__STATIC_INLINE void LL_SPI_EnableIT_EOT (SPI_TypeDef * SPIx)

Function description

Enable End Of Transfer IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER EOTIE LL_SPI_EnableIT_EOT

LL_SPI_EnableIT_TXTF

Function name

__STATIC_INLINE void LL_SPI_EnableIT_TXTF (SPI_TypeDef * SPIx)

Function description

Enable TXTF IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER TXTFIE LL_SPI_EnableIT_TXTF

LL_SPI_EnableIT_UDR

Function name

__STATIC_INLINE void LL_SPI_EnableIT_UDR (SPI_TypeDef * SPIx)

Function description

Enable Underrun IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER UDRIE LL_SPI_EnableIT_UDR

LL_SPI_EnableIT_OVR

Function name

__STATIC_INLINE void LL_SPI_EnableIT_OVR (SPI_TypeDef * SPIx)

Function description

Enable Overrun IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER OVRIE LL_SPI_EnableIT_OVR

LL_SPI_EnableIT_CRCERR

Function name

__STATIC_INLINE void LL_SPI_EnableIT_CRCERR (SPI_TypeDef * SPIx)

Function description

Enable CRC Error IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER CRCEIE LL_SPI_EnableIT_CRCERR

LL_SPI_EnableIT_FRE

Function name

__STATIC_INLINE void LL_SPI_EnableIT_FRE (SPI_TypeDef * SPIx)

Function description

Enable TI Frame Format Error IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER TIFREIE LL_SPI_EnableIT_FRE

LL_SPI_EnableIT_MODF

Function name

__STATIC_INLINE void LL_SPI_EnableIT_MODF (SPI_TypeDef * SPIx)

Function description

Enable MODF IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER MODFIE LL_SPI_EnableIT_MODF

LL_SPI_DisableIT_RXP

Function name

__STATIC_INLINE void LL_SPI_DisableIT_RXP (SPI_TypeDef * SPIx)

Function description

Disable Rx Packet available IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER RXPIE LL_SPI_DisableIT_RXP

LL_SPI_DisableIT_TXP

Function name

__STATIC_INLINE void LL_SPI_DisableIT_TXP (SPI_TypeDef * SPIx)

Function description

Disable Tx Packet space available IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER TXPIE LL_SPI_DisableIT_TXP

LL_SPI_DisableIT_DXP

Function name

__STATIC_INLINE void LL_SPI_DisableIT_DXP (SPI_TypeDef * SPIx)

Function description

Disable Duplex Packet available IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER DXPIE LL_SPI_DisableIT_DXP

LL_SPI_DisableIT_EOT

Function name

__STATIC_INLINE void LL_SPI_DisableIT_EOT (SPI_TypeDef * SPIx)

Function description

Disable End Of Transfer IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER EOTIE LL_SPI_DisableIT_EOT

LL_SPI_DisableIT_TXTF

Function name

__STATIC_INLINE void LL_SPI_DisableIT_TXTF (SPI_TypeDef * SPIx)

Function description

Disable TXTF IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER TXTFIE LL_SPI_DisableIT_TXTF

LL_SPI_DisableIT_UDR

Function name

__STATIC_INLINE void LL_SPI_DisableIT_UDR (SPI_TypeDef * SPIx)

Function description

Disable Underrun IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER UDRIE LL_SPI_DisableIT_UDR

LL_SPI_DisableIT_OVR

Function name

__STATIC_INLINE void LL_SPI_DisableIT_OVR (SPI_TypeDef * SPIx)

Function description

Disable Overrun IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER OVRIE LL_SPI_DisableIT_OVR

LL_SPI_DisableIT_CRCERR

Function name

__STATIC_INLINE void LL_SPI_DisableIT_CRCERR (SPI_TypeDef * SPIx)

Function description

Disable CRC Error IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER CRCEIE LL_SPI_DisableIT_CRCERR

LL_SPI_DisableIT_FRE

Function name

__STATIC_INLINE void LL_SPI_DisableIT_FRE (SPI_TypeDef * SPIx)

Function description

Disable TI Frame Format Error IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER TIFREIE LL_SPI_DisableIT_FRE

LL_SPI_DisableIT_MODF

Function name

__STATIC_INLINE void LL_SPI_DisableIT_MODF (SPI_TypeDef * SPIx)

Function description

Disable MODF IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER MODFIE LL_SPI_DisableIT_MODF

LL_SPI_IsEnabledIT_RXP

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledIT_RXP (const SPI_TypeDef * SPIx)

Function description

Check if Rx Packet available IT is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- IER RXPIE LL_SPI_IsEnabledIT_RXP

LL_SPI_IsEnabledIT_TXP

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledIT_TXP (const SPI_TypeDef * SPIx)

Function description

Check if Tx Packet space available IT is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- IER TXPIE LL_SPI_IsEnabledIT_TXP

LL_SPI_IsEnabledIT_DXP

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledIT_DXP (const SPI_TypeDef * SPIx)

Function description

Check if Duplex Packet available IT is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- IER DXPIE LL_SPI_IsEnabledIT_DXP

LL_SPI_IsEnabledIT_EOT

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledIT_EOT (const SPI_TypeDef * SPIx)

Function description

Check if End Of Transfer IT is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- IER EOTIE LL_SPI_IsEnabledIT_EOT

LL_SPI_IsEnabledIT_TXTF

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledIT_TXTF (const SPI_TypeDef * SPIx)

Function description

Check if TXTF IT is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- IER TXTFIE LL_SPI_IsEnabledIT_TXTF

LL_SPI_IsEnabledIT_UDR

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledIT_UDR (const SPI_TypeDef * SPIx)

Function description

Check if Underrun IT is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- IER UDRIE LL_SPI_IsEnabledIT_UDR

LL_SPI_IsEnabledIT_OVR

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledIT_OVR (const SPI_TypeDef * SPIx)

Function description

Check if Overrun IT is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- IER OVRIE LL_SPI_IsEnabledIT_OVR

LL_SPI_IsEnabledIT_CRCERR

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledIT_CRCERR (const SPI_TypeDef * SPIx)

Function description

Check if CRC Error IT is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- IER CRCEIE LL_SPI_IsEnabledIT_CRCERR

LL_SPI_IsEnabledIT_FRE

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledIT_FRE (const SPI_TypeDef * SPIx)

Function description

Check if TI Frame Format Error IT is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- IER TIFREIE LL_SPI_IsEnabledIT_FRE

LL_SPI_IsEnabledIT_MODF

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledIT_MODF (const SPI_TypeDef * SPIx)

Function description

Check if MODF IT is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- IER MODFIE LL_SPI_IsEnabledIT_MODF

LL_SPI_EnableDMAReq_RX

Function name

__STATIC_INLINE void LL_SPI_EnableDMAReq_RX (SPI_TypeDef * SPIx)

Function description

Enable DMA Rx.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG1 RXDMAEN LL_SPI_EnableDMAReq_RX

LL_SPI_DisableDMAReq_RX

Function name

__STATIC_INLINE void LL_SPI_DisableDMAReq_RX (SPI_TypeDef * SPIx)

Function description

Disable DMA Rx.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG1 RXDMAEN LL_SPI_DisableDMAReq_RX

LL_SPI_IsEnabledDMAReq_RX

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledDMAReq_RX (const SPI_TypeDef * SPIx)

Function description

Check if DMA Rx is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- CFG1 RXDMAEN LL_SPI_IsEnabledDMAReq_RX

LL_SPI_EnableDMAReq_TX

Function name

__STATIC_INLINE void LL_SPI_EnableDMAReq_TX (SPI_TypeDef * SPIx)

Function description

Enable DMA Tx.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG1 TXDMAEN LL_SPI_EnableDMAReq_TX

LL_SPI_DisableDMAReq_TX

Function name

__STATIC_INLINE void LL_SPI_DisableDMAReq_TX (SPI_TypeDef * SPIx)

Function description

Disable DMA Tx.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG1 TXDMAEN LL_SPI_DisableDMAReq_TX

LL_SPI_IsEnabledDMAReq_TX

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledDMAReq_TX (const SPI_TypeDef * SPIx)

Function description

Check if DMA Tx is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- CFG1 TXDMAEN LL_SPI_IsEnabledDMAReq_TX

LL_SPI_DMA_GetTxRegAddr

Function name

__STATIC_INLINE uint32_t LL_SPI_DMA_GetTxRegAddr (const SPI_TypeDef * SPIx)

Function description

Get the data register address used for DMA transfer.

Parameters

- **SPIx:** SPI Instance

Return values

- **Address:** of data register

Reference Manual to LL API cross reference:

- TXDR TXDR LL_SPI_DMA_GetTxRegAddr

LL_SPI_DMA_GetRxRegAddr

Function name

__STATIC_INLINE uint32_t LL_SPI_DMA_GetRxRegAddr (const SPI_TypeDef * SPIx)

Function description

Get the data register address used for DMA transfer.

Parameters

- **SPIx:** SPI Instance

Return values

- **Address:** of data register

Reference Manual to LL API cross reference:

- RXDR RXDR LL_SPI_DMA_GetRxRegAddr

LL_SPI_ReceiveData8

Function name

__STATIC_INLINE uint8_t LL_SPI_ReceiveData8 (SPI_TypeDef * SPIx)

Function description

Read Data Register.

Parameters

- **SPIx:** SPI Instance

Return values

- **0..0xFF:**

Reference Manual to LL API cross reference:

- RXDR . LL_SPI_ReceiveData8

LL_SPI_ReceiveData16

Function name

__STATIC_INLINE uint16_t LL_SPI_ReceiveData16 (SPI_TypeDef * SPIx)

Function description

Read Data Register.

Parameters

- **SPIx:** SPI Instance

Return values

- **0..0xFFFF:**

Reference Manual to LL API cross reference:

- RXDR . LL_SPI_ReceiveData16

LL_SPI_ReceiveData32

Function name

__STATIC_INLINE uint32_t LL_SPI_ReceiveData32 (SPI_TypeDef * SPIx)

Function description

Read Data Register.

Parameters

- **SPIx:** SPI Instance

Return values

- **0..0xFFFFFFFF:**

Reference Manual to LL API cross reference:

- RXDR . LL_SPI_ReceiveData32

LL_SPI_TransmitData8

Function name

__STATIC_INLINE void LL_SPI_TransmitData8 (SPI_TypeDef * SPIx, uint8_t TxData)

Function description

Write Data Register.

Parameters

- **SPIx:** SPI Instance
- **TxData:** 0..0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- TXDR . LL_SPI_TransmitData8

LL_SPI_TransmitData16

Function name

__STATIC_INLINE void LL_SPI_TransmitData16 (SPI_TypeDef * SPIx, uint16_t TxData)

Function description

Write Data Register.

Parameters

- **SPIx:** SPI Instance
- **TxData:** 0..0xFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- TXDR . LL_SPI_TransmitData16

LL_SPI_TransmitData32

Function name

__STATIC_INLINE void LL_SPI_TransmitData32 (SPI_TypeDef * SPIx, uint32_t TxData)

Function description

Write Data Register.

Parameters

- **SPIx:** SPI Instance
- **TxData:** 0..0xFFFFFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- TXDR . LL_SPI_TransmitData32

LL_SPI_SetCRCPolynomial

Function name

__STATIC_INLINE void LL_SPI_SetCRCPolynomial (SPI_TypeDef * SPIx, uint32_t CRCPoly)

Function description

Set polynomial for CRC calcul.

Parameters

- **SPIx:** SPI Instance
- **CRCPoly:** 0..0xFFFFFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- CRCPOLY CRCPOLY LL_SPI_SetCRCPolynomial

LL_SPI_GetCRCPolynomial

Function name

__STATIC_INLINE uint32_t LL_SPI_GetCRCPolynomial (const SPI_TypeDef * SPIx)

Function description

Get polynomial for CRC calcul.

Parameters

- **SPIx:** SPI Instance

Return values

- **0..0xFFFFFFFF:**

Reference Manual to LL API cross reference:

- CRCPOLY CRCPOLY LL_SPI_GetCRCPolynomial

LL_SPI_SetUDRPattern

Function name

__STATIC_INLINE void LL_SPI_SetUDRPattern (SPI_TypeDef * SPIx, uint32_t Pattern)

Function description

Set the underrun pattern.

Parameters

- **SPIx:** SPI Instance
- **Pattern:** 0..0xFFFFFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- UDRDR UDRDR LL_SPI_SetUDRPattern

LL_SPI_GetUDRPattern

Function name

__STATIC_INLINE uint32_t LL_SPI_GetUDRPattern (const SPI_TypeDef * SPIx)

Function description

Get the underrun pattern.

Parameters

- **SPIx:** SPI Instance

Return values

- **0..0xFFFFFFFF:**

Reference Manual to LL API cross reference:

- UDRDR UDRDR LL_SPI_GetUDRPattern

LL_SPI_GetRxCRC

Function name

__STATIC_INLINE uint32_t LL_SPI_GetRxCRC (const SPI_TypeDef * SPIx)

Function description

Get Rx CRC.

Parameters

- **SPIx:** SPI Instance

Return values

- **0..0xFFFFFFFF:**

Reference Manual to LL API cross reference:

- RXCRCR RXCRC LL_SPI_GetRxCRC

LL_SPI_GetTxCRC

Function name

__STATIC_INLINE uint32_t LL_SPI_GetTxCRC (const SPI_TypeDef * SPIx)

Function description

Get Tx CRC.

Parameters

- **SPIx:** SPI Instance

Return values

- **0..0xFFFFFFFF:**

Reference Manual to LL API cross reference:

- TXCRCR TXCRC LL_SPI_GetTxCRC

LL_SPI_DeInit

Function name

ErrorStatus LL_SPI_DeInit (const SPI_TypeDef * SPIx)

Function description

De-initialize the SPI registers to their default reset values.

Parameters

- **SPIx:** SPI Instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: SPI registers are de-initialized
  - ERROR: SPI registers are not de-initialized

LL_SPI_Init

Function name

ErrorStatus LL_SPI_Init (SPI_TypeDef * SPIx, LL_SPI_InitTypeDef * SPI_InitStruct)

Function description

Initialize the SPI registers according to the specified parameters in SPI_InitStruct.

Parameters

- **SPIx:** SPI Instance
- **SPI_InitStruct:** pointer to a LL_SPI_InitTypeDef structure

Return values

- **An:** ErrorStatus enumeration value. (Return always SUCCESS)

Notes

- As some bits in SPI configuration registers can only be written when the SPI is disabled (SPI_CR1_SPE bit =0), SPI IP should be in disabled state prior calling this function. Otherwise, ERROR result will be returned.

LL_SPI_StructInit

Function name

void LL_SPI_StructInit (LL_SPI_InitTypeDef * SPI_InitStruct)

Function description

Set each LL_SPI_InitTypeDef field to default value.

Parameters

- **SPI_InitStruct:** pointer to a LL_SPI_InitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

LL_I2S_SetDataFormat

Function name

__STATIC_INLINE void LL_I2S_SetDataFormat (SPI_TypeDef * SPIx, uint32_t DataLength)

Function description

Set I2S Data frame format.

Parameters

- **SPIx:** SPI Handle
- **DataLength:** This parameter can be one of the following values:
  - LL_I2S_DATAFORMAT_16B
  - LL_I2S_DATAFORMAT_16B_EXTENDED
  - LL_I2S_DATAFORMAT_24B
  - LL_I2S_DATAFORMAT_24B_LEFT_ALIGNED
  - LL_I2S_DATAFORMAT_32B

Return values

- **None:**

Reference Manual to LL API cross reference:

- I2SCFGR DATLEN LL_I2S_SetDataFormat
- I2SCFGR CHLEN LL_I2S_SetDataFormat
- I2SCFGR DATFMT LL_I2S_SetDataFormat

LL_I2S_GetDataFormat

Function name

__STATIC_INLINE uint32_t LL_I2S_GetDataFormat (const SPI_TypeDef * SPIx)

Function description

Get I2S Data frame format.

Parameters

- **SPIx:** SPI Handle

Return values

- **Return:** value can be one of the following values:
  - LL_I2S_DATAFORMAT_16B
  - LL_I2S_DATAFORMAT_16B_EXTENDED
  - LL_I2S_DATAFORMAT_24B
  - LL_I2S_DATAFORMAT_24B_LEFT_ALIGNED
  - LL_I2S_DATAFORMAT_32B

Reference Manual to LL API cross reference:

- I2SCFGR DATLEN LL_I2S_GetDataFormat
- I2SCFGR CHLEN LL_I2S_GetDataFormat
- I2SCFGR DATFMT LL_I2S_GetDataFormat

LL_I2S_SetChannelLengthType

Function name

__STATIC_INLINE void LL_I2S_SetChannelLengthType (SPI_TypeDef * SPIx, uint32_t ChannelLengthType)

Function description

Set I2S Channel Length Type.

Parameters

- **SPIx:** SPI Handle
- **ChannelLengthType:** This parameter can be one of the following values:
  - LL_I2S_SLAVE_VARIABLE_CH_LENGTH
  - LL_I2S_SLAVE_FIXED_CH_LENGTH

Return values

- **None:**

Notes

- This feature is useful with SLAVE only

Reference Manual to LL API cross reference:

- I2SCFGR FIXCH LL_I2S_SetChannelLengthType

LL_I2S_GetChannelLengthType

Function name

__STATIC_INLINE uint32_t LL_I2S_GetChannelLengthType (const SPI_TypeDef * SPIx)

Function description

Get I2S Channel Length Type.

Parameters

- **SPIx:** SPI Handle

Return values

- **Return:** value can be one of the following values:
  - LL_I2S_SLAVE_VARIABLE_CH_LENGTH
  - LL_I2S_SLAVE_FIXED_CH_LENGTH

Notes

- This feature is useful with SLAVE only

Reference Manual to LL API cross reference:

- I2SCFGR FIXCH LL_I2S_GetChannelLengthType

LL_I2S_EnableWordSelectInversion

Function name

__STATIC_INLINE void LL_I2S_EnableWordSelectInversion (SPI_TypeDef * SPIx)

Function description

Invert the default polarity of WS signal.

Parameters

- **SPIx:** SPI Handle

Return values

- **None:**

Reference Manual to LL API cross reference:

- I2SCFGR WSINV LL_I2S_EnableWordSelectInversion

LL_I2S_DisableWordSelectInversion

Function name

__STATIC_INLINE void LL_I2S_DisableWordSelectInversion (SPI_TypeDef * SPIx)

Function description

Use the default polarity of WS signal.

Parameters

- **SPIx:** SPI Handle

Return values

- **None:**

Reference Manual to LL API cross reference:

- I2SCFGR WSINV LL_I2S_DisableWordSelectInversion

LL_I2S_IsEnabledWordSelectInversion

Function name

__STATIC_INLINE uint32_t LL_I2S_IsEnabledWordSelectInversion (const SPI_TypeDef * SPIx)

Function description

Check if polarity of WS signal is inverted.

Parameters

- **SPIx:** SPI Handle

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- I2SCFGR WSINV LL_I2S_IsEnabledWordSelectInversion

LL_I2S_SetClockPolarity

Function name

__STATIC_INLINE void LL_I2S_SetClockPolarity (SPI_TypeDef * SPIx, uint32_t ClockPolarity)

Function description

Set 2S Clock Polarity.

Parameters

- **SPIx:** SPI Handle
- **ClockPolarity:** This parameter can be one of the following values:
  - LL_I2S_POLARITY_LOW
  - LL_I2S_POLARITY_HIGH

Return values

- **None:**

Reference Manual to LL API cross reference:

- I2SCFGR CKPOL LL_I2S_SetClockPolarity

LL_I2S_GetClockPolarity

Function name

__STATIC_INLINE uint32_t LL_I2S_GetClockPolarity (const SPI_TypeDef * SPIx)

Function description

Get 2S Clock Polarity.

Parameters

- **SPIx:** SPI Handle

Return values

- **Return:** value can be one of the following values:
  - LL_I2S_POLARITY_LOW
  - LL_I2S_POLARITY_HIGH

Reference Manual to LL API cross reference:

- I2SCFGR CKPOL LL_I2S_GetClockPolarity

LL_I2S_SetStandard

Function name

__STATIC_INLINE void LL_I2S_SetStandard (SPI_TypeDef * SPIx, uint32_t Standard)

Function description

Set I2S standard.

Parameters

- **SPIx:** SPI Handle
- **Standard:** This parameter can be one of the following values:
  - LL_I2S_STANDARD_PHILIPS
  - LL_I2S_STANDARD_MSB
  - LL_I2S_STANDARD_LSB
  - LL_I2S_STANDARD_PCM_SHORT
  - LL_I2S_STANDARD_PCM_LONG

Return values

- **None:**

Reference Manual to LL API cross reference:

- I2SCFGR I2SSTD LL_I2S_SetStandard
- I2SCFGR PCMSYNC LL_I2S_SetStandard

LL_I2S_GetStandard

Function name

__STATIC_INLINE uint32_t LL_I2S_GetStandard (const SPI_TypeDef * SPIx)

Function description

Get I2S standard.

Parameters

- **SPIx:** SPI Handle

Return values

- **Return:** value can be one of the following values:
  - LL_I2S_STANDARD_PHILIPS
  - LL_I2S_STANDARD_MSB
  - LL_I2S_STANDARD_LSB
  - LL_I2S_STANDARD_PCM_SHORT
  - LL_I2S_STANDARD_PCM_LONG

Reference Manual to LL API cross reference:

- I2SCFGR I2SSTD LL_I2S_GetStandard
- I2SCFGR PCMSYNC LL_I2S_GetStandard

LL_I2S_SetTransferMode

Function name

__STATIC_INLINE void LL_I2S_SetTransferMode (SPI_TypeDef * SPIx, uint32_t Standard)

Function description

Set I2S config.

Parameters

- **SPIx:** SPI Handle
- **Standard:** This parameter can be one of the following values:
  - LL_I2S_MODE_SLAVE_TX
  - LL_I2S_MODE_SLAVE_RX
  - LL_I2S_MODE_SLAVE_FULL_DUPLEX
  - LL_I2S_MODE_MASTER_TX
  - LL_I2S_MODE_MASTER_RX
  - LL_I2S_MODE_MASTER_FULL_DUPLEX

Return values

- **None:**

Reference Manual to LL API cross reference:

- I2SCFGR I2SCFG LL_I2S_SetTransferMode

LL_I2S_GetTransferMode

Function name

__STATIC_INLINE uint32_t LL_I2S_GetTransferMode (const SPI_TypeDef * SPIx)

Function description

Get I2S config.

Parameters

- **SPIx:** SPI Handle

Return values

- **Return:** value can be one of the following values:
  - LL_I2S_MODE_SLAVE_TX
  - LL_I2S_MODE_SLAVE_RX
  - LL_I2S_MODE_SLAVE_FULL_DUPLEX
  - LL_I2S_MODE_MASTER_TX
  - LL_I2S_MODE_MASTER_RX
  - LL_I2S_MODE_MASTER_FULL_DUPLEX

Reference Manual to LL API cross reference:

- I2SCFGR I2SCFG LL_I2S_GetTransferMode

LL_I2S_Enable

Function name

__STATIC_INLINE void LL_I2S_Enable (SPI_TypeDef * SPIx)

Function description

Select I2S mode and Enable I2S peripheral.

Parameters

- **SPIx:** SPI Handle

Return values

- **None:**

Reference Manual to LL API cross reference:

- I2SCFGR I2SMOD LL_I2S_Enable
- CR1 SPE LL_I2S_Enable

LL_I2S_Disable

Function name

__STATIC_INLINE void LL_I2S_Disable (SPI_TypeDef * SPIx)

Function description

Disable I2S peripheral and disable I2S mode.

Parameters

- **SPIx:** SPI Handle

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 SPE LL_I2S_Disable
- I2SCFGR I2SMOD LL_I2S_Disable

LL_I2S_EnableIOSwap

Function name

__STATIC_INLINE void LL_I2S_EnableIOSwap (SPI_TypeDef * SPIx)

Function description

Swap the SDO and SDI pin.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Notes

- This configuration can not be changed when I2S is enabled.

Reference Manual to LL API cross reference:

- CFG2 IOSWP LL_I2S_EnableIOSwap

LL_I2S_DisableIOSwap

Function name

__STATIC_INLINE void LL_I2S_DisableIOSwap (SPI_TypeDef * SPIx)

Function description

Restore default function for SDO and SDI pin.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Notes

- This configuration can not be changed when I2S is enabled.

Reference Manual to LL API cross reference:

- CFG2 IOSWP LL_I2S_DisableIOSwap

LL_I2S_IsEnabledIOSwap

Function name

__STATIC_INLINE uint32_t LL_I2S_IsEnabledIOSwap (const SPI_TypeDef * SPIx)

Function description

Check if SDO and SDI pin are swapped.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- CFG2 IOSWP LL_I2S_IsEnabledIOSwap

LL_I2S_EnableGPIOControl

Function name

__STATIC_INLINE void LL_I2S_EnableGPIOControl (SPI_TypeDef * SPIx)

Function description

Enable GPIO control.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Notes

- This configuration can not be changed when I2S is enabled.

Reference Manual to LL API cross reference:

- CFG2 AFCNTR LL_I2S_EnableGPIOControl

LL_I2S_DisableGPIOControl

Function name

__STATIC_INLINE void LL_I2S_DisableGPIOControl (SPI_TypeDef * SPIx)

Function description

Disable GPIO control.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Notes

- This configuration can not be changed when I2S is enabled.

Reference Manual to LL API cross reference:

- CFG2 AFCNTR LL_I2S_DisableGPIOControl

LL_I2S_IsEnabledGPIOControl

Function name

__STATIC_INLINE uint32_t LL_I2S_IsEnabledGPIOControl (const SPI_TypeDef * SPIx)

Function description

Check if GPIO control is active.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- CFG2 AFCNTR LL_I2S_IsEnabledGPIOControl

LL_I2S_EnableIOLock

Function name

__STATIC_INLINE void LL_I2S_EnableIOLock (SPI_TypeDef * SPIx)

Function description

Lock the AF configuration of associated IOs.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Notes

- Once this bit is set, the SPI_CFG2 register content can not be modified until a hardware reset occurs. The reset of the IOLock bit is done by hardware. for that, LL_SPI_DisableIOLock can not exist.

Reference Manual to LL API cross reference:

- CR1 IOLOCK LL_SPI_EnableIOLock

LL_I2S_IsEnabledIOLock

Function name

__STATIC_INLINE uint32_t LL_I2S_IsEnabledIOLock (const SPI_TypeDef * SPIx)

Function description

Check if the the SPI_CFG2 register is locked.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- CR1 IOLOCK LL_I2S_IsEnabledIOLock

LL_I2S_SetTransferBitOrder

Function name

__STATIC_INLINE void LL_I2S_SetTransferBitOrder (SPI_TypeDef * SPIx, uint32_t BitOrder)

Function description

Set Transfer Bit Order.

Parameters

- **SPIx:** SPI Instance
- **BitOrder:** This parameter can be one of the following values:
  - LL_I2S_LSB_FIRST
  - LL_I2S_MSB_FIRST

Return values

- **None:**

Notes

- This configuration can not be changed when I2S is enabled.

Reference Manual to LL API cross reference:

- CFG2 LSBFRST LL_I2S_SetTransferBitOrder

LL_I2S_GetTransferBitOrder

Function name

__STATIC_INLINE uint32_t LL_I2S_GetTransferBitOrder (const SPI_TypeDef * SPIx)

Function description

Get Transfer Bit Order.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_I2S_LSB_FIRST
  - LL_I2S_MSB_FIRST

Reference Manual to LL API cross reference:

- CFG2 LSBFRST LL_I2S_GetTransferBitOrder

LL_I2S_StartTransfer

Function name

__STATIC_INLINE void LL_I2S_StartTransfer (SPI_TypeDef * SPIx)

Function description

Start effective transfer on wire.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 CSTART LL_I2S_StartTransfer

LL_I2S_IsActiveTransfer

Function name

__STATIC_INLINE uint32_t LL_I2S_IsActiveTransfer (const SPI_TypeDef * SPIx)

Function description

Check if there is an unfinished transfer.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- CR1 CSTART LL_I2S_IsActiveTransfer

LL_I2S_SetFIFOThreshold

Function name

__STATIC_INLINE void LL_I2S_SetFIFOThreshold (SPI_TypeDef * SPIx, uint32_t Threshold)

Function description

Set threshold of FIFO that triggers a transfer event.

Parameters

- **SPIx:** SPI Instance
- **Threshold:** This parameter can be one of the following values:
  - LL_I2S_FIFO_TH_01DATA
  - LL_I2S_FIFO_TH_02DATA
  - LL_I2S_FIFO_TH_03DATA
  - LL_I2S_FIFO_TH_04DATA
  - LL_I2S_FIFO_TH_05DATA
  - LL_I2S_FIFO_TH_06DATA
  - LL_I2S_FIFO_TH_07DATA
  - LL_I2S_FIFO_TH_08DATA

Return values

- **None:**

Notes

- This configuration can not be changed when I2S is enabled.

Reference Manual to LL API cross reference:

- CFG1 FTHLV LL_I2S_SetFIFOThreshold

LL_I2S_GetFIFOThreshold

Function name

__STATIC_INLINE uint32_t LL_I2S_GetFIFOThreshold (const SPI_TypeDef * SPIx)

Function description

Get threshold of FIFO that triggers a transfer event.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_I2S_FIFO_TH_01DATA
  - LL_I2S_FIFO_TH_02DATA
  - LL_I2S_FIFO_TH_03DATA
  - LL_I2S_FIFO_TH_04DATA
  - LL_I2S_FIFO_TH_05DATA
  - LL_I2S_FIFO_TH_06DATA
  - LL_I2S_FIFO_TH_07DATA
  - LL_I2S_FIFO_TH_08DATA

Reference Manual to LL API cross reference:

- CFG1 FTHLV LL_I2S_GetFIFOThreshold

LL_I2S_SetPrescalerLinear

Function name

__STATIC_INLINE void LL_I2S_SetPrescalerLinear (SPI_TypeDef * SPIx, uint32_t PrescalerLinear)

Function description

Set I2S linear prescaler.

Parameters

- **SPIx:** SPI Instance
- **PrescalerLinear:** Value between Min_Data=0x00 and Max_Data=0xFF

Return values

- **None:**

Notes

- PrescalerLinear '1' is not authorized with parity LL_I2S_PRESCALER_PARITY_ODD

Reference Manual to LL API cross reference:

- I2SCFGR I2SDIV LL_I2S_SetPrescalerLinear

LL_I2S_GetPrescalerLinear

Function name

__STATIC_INLINE uint32_t LL_I2S_GetPrescalerLinear (const SPI_TypeDef * SPIx)

Function description

Get I2S linear prescaler.

Parameters

- **SPIx:** SPI Instance

Return values

- **PrescalerLinear:** Value between Min_Data=0x00 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- I2SCFGR I2SDIV LL_I2S_GetPrescalerLinear

LL_I2S_SetPrescalerParity

Function name

__STATIC_INLINE void LL_I2S_SetPrescalerParity (SPI_TypeDef * SPIx, uint32_t PrescalerParity)

Function description

Set I2S parity prescaler.

Parameters

- **SPIx:** SPI Instance
- **PrescalerParity:** This parameter can be one of the following values:
  - LL_I2S_PRESCALER_PARITY_EVEN
  - LL_I2S_PRESCALER_PARITY_ODD

Return values

- **None:**

Reference Manual to LL API cross reference:

- I2SCFGR ODD LL_I2S_SetPrescalerParity

LL_I2S_GetPrescalerParity

Function name

__STATIC_INLINE uint32_t LL_I2S_GetPrescalerParity (const SPI_TypeDef * SPIx)

Function description

Get I2S parity prescaler.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_I2S_PRESCALER_PARITY_EVEN
  - LL_I2S_PRESCALER_PARITY_ODD

Reference Manual to LL API cross reference:

- I2SCFGR ODD LL_I2S_GetPrescalerParity

LL_I2S_EnableMasterClock

Function name

__STATIC_INLINE void LL_I2S_EnableMasterClock (SPI_TypeDef * SPIx)

Function description

Enable the Master Clock Output (Pin MCK)

Parameters

- **SPIx:** SPI Handle

Return values

- **None:**

Reference Manual to LL API cross reference:

- I2SCFGR MCKOE LL_I2S_EnableMasterClock

LL_I2S_DisableMasterClock

Function name

__STATIC_INLINE void LL_I2S_DisableMasterClock (SPI_TypeDef * SPIx)

Function description

Disable the Master Clock Output (Pin MCK)

Parameters

- **SPIx:** SPI Handle

Return values

- **None:**

Reference Manual to LL API cross reference:

- I2SCFGR MCKOE LL_I2S_DisableMasterClock

LL_I2S_IsEnabledMasterClock

Function name

__STATIC_INLINE uint32_t LL_I2S_IsEnabledMasterClock (const SPI_TypeDef * SPIx)

Function description

Check if the master clock output (Pin MCK) is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- I2SCFGR MCKOE LL_I2S_IsEnabledMasterClock

LL_I2S_IsActiveFlag_RXP

Function name

__STATIC_INLINE uint32_t LL_I2S_IsActiveFlag_RXP (const SPI_TypeDef * SPIx)

Function description

Check if there enough data in FIFO to read a full packet.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- SR RXP LL_I2S_IsActiveFlag_RXP

LL_I2S_IsActiveFlag_TXP

Function name

__STATIC_INLINE uint32_t LL_I2S_IsActiveFlag_TXP (const SPI_TypeDef * SPIx)

Function description

Check if there enough space in FIFO to hold a full packet.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- SR TXP LL_I2S_IsActiveFlag_TXP

LL_I2S_IsActiveFlag_UDR

Function name

__STATIC_INLINE uint32_t LL_I2S_IsActiveFlag_UDR (const SPI_TypeDef * SPIx)

Function description

Get Underrun error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- SR UDR LL_I2S_IsActiveFlag_UDR

LL_I2S_IsActiveFlag_OVR

Function name

__STATIC_INLINE uint32_t LL_I2S_IsActiveFlag_OVR (const SPI_TypeDef * SPIx)

Function description

Get Overrun error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR OVR LL_I2S_IsActiveFlag_OVR

LL_I2S_IsActiveFlag_FRE

Function name

__STATIC_INLINE uint32_t LL_I2S_IsActiveFlag_FRE (const SPI_TypeDef * SPIx)

Function description

Get TI Frame format error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR TIFRE LL_I2S_IsActiveFlag_FRE

LL_I2S_ClearFlag_UDR

Function name

__STATIC_INLINE void LL_I2S_ClearFlag_UDR (SPI_TypeDef * SPIx)

Function description

Clear Underrun error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR UDRC LL_I2S_ClearFlag_UDR

LL_I2S_ClearFlag_OVR

Function name

__STATIC_INLINE void LL_I2S_ClearFlag_OVR (SPI_TypeDef * SPIx)

Function description

Clear Overrun error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR OVRC LL_I2S_ClearFlag_OVR

LL_I2S_ClearFlag_FRE

Function name

__STATIC_INLINE void LL_I2S_ClearFlag_FRE (SPI_TypeDef * SPIx)

Function description

Clear Frame format error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR TIFREC LL_I2S_ClearFlag_FRE

LL_I2S_EnableIT_RXP

Function name

__STATIC_INLINE void LL_I2S_EnableIT_RXP (SPI_TypeDef * SPIx)

Function description

Enable Rx Packet available IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER RXPIE LL_I2S_EnableIT_RXP

LL_I2S_EnableIT_TXP

Function name

__STATIC_INLINE void LL_I2S_EnableIT_TXP (SPI_TypeDef * SPIx)

Function description

Enable Tx Packet space available IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER TXPIE LL_I2S_EnableIT_TXP

LL_I2S_EnableIT_UDR

Function name

__STATIC_INLINE void LL_I2S_EnableIT_UDR (SPI_TypeDef * SPIx)

Function description

Enable Underrun IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER UDRIE LL_I2S_EnableIT_UDR

LL_I2S_EnableIT_OVR

Function name

__STATIC_INLINE void LL_I2S_EnableIT_OVR (SPI_TypeDef * SPIx)

Function description

Enable Overrun IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER OVRIE LL_I2S_EnableIT_OVR

LL_I2S_EnableIT_FRE

Function name

__STATIC_INLINE void LL_I2S_EnableIT_FRE (SPI_TypeDef * SPIx)

Function description

Enable TI Frame Format Error IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER TIFREIE LL_I2S_EnableIT_FRE

LL_I2S_DisableIT_RXP

Function name

__STATIC_INLINE void LL_I2S_DisableIT_RXP (SPI_TypeDef * SPIx)

Function description

Disable Rx Packet available IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER RXPIE LL_I2S_DisableIT_RXP

LL_I2S_DisableIT_TXP

Function name

__STATIC_INLINE void LL_I2S_DisableIT_TXP (SPI_TypeDef * SPIx)

Function description

Disable Tx Packet space available IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER TXPIE LL_I2S_DisableIT_TXP

LL_I2S_DisableIT_UDR

Function name

__STATIC_INLINE void LL_I2S_DisableIT_UDR (SPI_TypeDef * SPIx)

Function description

Disable Underrun IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER UDRIE LL_I2S_DisableIT_UDR

LL_I2S_DisableIT_OVR

Function name

__STATIC_INLINE void LL_I2S_DisableIT_OVR (SPI_TypeDef * SPIx)

Function description

Disable Overrun IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER OVRIE LL_I2S_DisableIT_OVR

LL_I2S_DisableIT_FRE

Function name

__STATIC_INLINE void LL_I2S_DisableIT_FRE (SPI_TypeDef * SPIx)

Function description

Disable TI Frame Format Error IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER TIFREIE LL_I2S_DisableIT_FRE

LL_I2S_IsEnabledIT_RXP

Function name

__STATIC_INLINE uint32_t LL_I2S_IsEnabledIT_RXP (const SPI_TypeDef * SPIx)

Function description

Check if Rx Packet available IT is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- IER RXPIE LL_I2S_IsEnabledIT_RXP

LL_I2S_IsEnabledIT_TXP

Function name

__STATIC_INLINE uint32_t LL_I2S_IsEnabledIT_TXP (const SPI_TypeDef * SPIx)

Function description

Check if Tx Packet space available IT is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- IER TXPIE LL_I2S_IsEnabledIT_TXP

LL_I2S_IsEnabledIT_UDR

Function name

__STATIC_INLINE uint32_t LL_I2S_IsEnabledIT_UDR (const SPI_TypeDef * SPIx)

Function description

Check if Underrun IT is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- IER UDRIE LL_I2S_IsEnabledIT_UDR

LL_I2S_IsEnabledIT_OVR

Function name

__STATIC_INLINE uint32_t LL_I2S_IsEnabledIT_OVR (const SPI_TypeDef * SPIx)

Function description

Check if Overrun IT is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- IER OVRIE LL_I2S_IsEnabledIT_OVR

LL_I2S_IsEnabledIT_FRE

Function name

__STATIC_INLINE uint32_t LL_I2S_IsEnabledIT_FRE (const SPI_TypeDef * SPIx)

Function description

Check if TI Frame Format Error IT is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- IER TIFREIE LL_I2S_IsEnabledIT_FRE

LL_I2S_EnableDMAReq_RX

Function name

__STATIC_INLINE void LL_I2S_EnableDMAReq_RX (SPI_TypeDef * SPIx)

Function description

Enable DMA Rx.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG1 RXDMAEN LL_I2S_EnableDMAReq_RX

LL_I2S_DisableDMAReq_RX

Function name

__STATIC_INLINE void LL_I2S_DisableDMAReq_RX (SPI_TypeDef * SPIx)

Function description

Disable DMA Rx.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG1 RXDMAEN LL_I2S_DisableDMAReq_RX

LL_I2S_IsEnabledDMAReq_RX

Function name

__STATIC_INLINE uint32_t LL_I2S_IsEnabledDMAReq_RX (const SPI_TypeDef * SPIx)

Function description

Check if DMA Rx is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- CFG1 RXDMAEN LL_I2S_IsEnabledDMAReq_RX

LL_I2S_EnableDMAReq_TX

Function name

__STATIC_INLINE void LL_I2S_EnableDMAReq_TX (SPI_TypeDef * SPIx)

Function description

Enable DMA Tx.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG1 TXDMAEN LL_I2S_EnableDMAReq_TX

LL_I2S_DisableDMAReq_TX

Function name

__STATIC_INLINE void LL_I2S_DisableDMAReq_TX (SPI_TypeDef * SPIx)

Function description

Disable DMA Tx.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG1 TXDMAEN LL_I2S_DisableDMAReq_TX

LL_I2S_IsEnabledDMAReq_TX

Function name

__STATIC_INLINE uint32_t LL_I2S_IsEnabledDMAReq_TX (const SPI_TypeDef * SPIx)

Function description

Check if DMA Tx is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- CFG1 TXDMAEN LL_I2S_IsEnabledDMAReq_TX

LL_I2S_ReceiveData16

Function name

__STATIC_INLINE uint16_t LL_I2S_ReceiveData16 (SPI_TypeDef * SPIx)

Function description

Read Data Register.

Parameters

- **SPIx:** SPI Instance

Return values

- **0..0xFFFF:**

Reference Manual to LL API cross reference:

- RXDR . LL_I2S_ReceiveData16

LL_I2S_ReceiveData32

Function name

__STATIC_INLINE uint32_t LL_I2S_ReceiveData32 (SPI_TypeDef * SPIx)

Function description

Read Data Register.

Parameters

- **SPIx:** SPI Instance

Return values

- **0..0xFFFFFFFF:**

Reference Manual to LL API cross reference:

- RXDR . LL_I2S_ReceiveData32

LL_I2S_TransmitData16

Function name

__STATIC_INLINE void LL_I2S_TransmitData16 (SPI_TypeDef * SPIx, uint16_t TxData)

Function description

Write Data Register.

Parameters

- **SPIx:** SPI Instance
- **TxData:** 0..0xFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- TXDR . LL_I2S_TransmitData16

LL_I2S_TransmitData32

Function name

__STATIC_INLINE void LL_I2S_TransmitData32 (SPI_TypeDef * SPIx, uint32_t TxData)

Function description

Write Data Register.

Parameters

- **SPIx:** SPI Instance
- **TxData:** 0..0xFFFFFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- TXDR . LL_I2S_TransmitData32

LL_I2S_DeInit

Function name

ErrorStatus LL_I2S_DeInit (const SPI_TypeDef * SPIx)

Function description

De-initialize the SPI/I2S registers to their default reset values.

Parameters

- **SPIx:** SPI Instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: SPI registers are de-initialized
  - ERROR: SPI registers are not de-initialized

LL_I2S_Init

Function name

ErrorStatus LL_I2S_Init (SPI_TypeDef * SPIx, const LL_I2S_InitTypeDef * I2S_InitStruct)

Function description

Initializes the SPI/I2S registers according to the specified parameters in I2S_InitStruct.

Parameters

- **SPIx:** SPI Instance
- **I2S_InitStruct:** pointer to a LL_I2S_InitTypeDef structure

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: SPI registers are Initialized
  - ERROR: SPI registers are not Initialized

Notes

- As some bits in I2S configuration registers can only be written when the SPI is disabled (SPI_CR1_SPE bit =0), SPI IP should be in disabled state prior calling this function. Otherwise, ERROR result will be returned.
- I2S (SPI) source clock must be ready before calling this function. Otherwise will results in wrong programming.

LL_I2S_StructInit

Function name

void LL_I2S_StructInit (LL_I2S_InitTypeDef * I2S_InitStruct)

Function description

Set each LL_I2S_InitTypeDef field to default value.

Parameters

- **I2S_InitStruct:** pointer to a LL_I2S_InitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

LL_I2S_ConfigPrescaler

Function name

void LL_I2S_ConfigPrescaler (SPI_TypeDef * SPIx, uint32_t PrescalerLinear, uint32_t PrescalerParity)

Function description

Set linear and parity prescaler.

Parameters

- **SPIx:** SPI Instance
- **PrescalerLinear:** Value between Min_Data=0x00 and Max_Data=0xFF
- **PrescalerParity:** This parameter can be one of the following values:
  - LL_I2S_PRESCALER_PARITY_EVEN
  - LL_I2S_PRESCALER_PARITY_ODD

Return values

- **None:**

Notes

- To calculate value of PrescalerLinear(I2SDIV[7:0] bits) and PrescalerParity(ODD bit) Check Audio frequency table and formulas inside Reference Manual (SPI/I2S).
- PrescalerLinear '1' is not authorized with parity LL_I2S_PRESCALER_PARITY_ODD

104.3 SPI Firmware driver defines

The following section lists the various define and macros of the module.

104.3.1 SPI

SPI

Baud Rate Prescaler

LL_SPI_BAUDRATEPRESCALER_BYPASS

LL_SPI_BAUDRATEPRESCALER_DIV2

LL_SPI_BAUDRATEPRESCALER_DIV4

LL_SPI_BAUDRATEPRESCALER_DIV8

LL_SPI_BAUDRATEPRESCALER_DIV16

LL_SPI_BAUDRATEPRESCALER_DIV32

LL_SPI_BAUDRATEPRESCALER_DIV64

LL_SPI_BAUDRATEPRESCALER_DIV128

LL_SPI_BAUDRATEPRESCALER_DIV256

Bit Order

LL_SPI_LSB_FIRST

LL_SPI_MSB_FIRST

CRC

LL_SPI_CRC_4BIT

LL_SPI_CRC_5BIT

LL_SPI_CRC_6BIT

LL_SPI_CRC_7BIT

LL_SPI_CRC_8BIT

LL_SPI_CRC_9BIT

LL_SPI_CRC_10BIT

LL_SPI_CRC_11BIT

LL_SPI_CRC_12BIT

LL_SPI_CRC_13BIT

LL_SPI_CRC_14BIT

LL_SPI_CRC_15BIT

LL_SPI_CRC_16BIT

LL_SPI_CRC_17BIT

LL_SPI_CRC_18BIT

LL_SPI_CRC_19BIT

LL_SPI_CRC_20BIT

LL_SPI_CRC_21BIT

LL_SPI_CRC_22BIT

LL_SPI_CRC_23BIT

LL_SPI_CRC_24BIT

LL_SPI_CRC_25BIT

LL_SPI_CRC_26BIT

LL_SPI_CRC_27BIT

LL_SPI_CRC_28BIT

LL_SPI_CRC_29BIT

LL_SPI_CRC_30BIT

LL_SPI_CRC_31BIT

LL_SPI_CRC_32BIT

CRC Calculation

LL_SPI_CRCCALCULATION_DISABLE

CRC calculation disabled

LL_SPI_CRCCALCULATION_ENABLE

CRC calculation enabled

Data Width

LL_SPI_DATAWIDTH_4BIT

LL_SPI_DATAWIDTH_5BIT

LL_SPI_DATAWIDTH_6BIT

LL_SPI_DATAWIDTH_7BIT

LL_SPI_DATAWIDTH_8BIT

LL_SPI_DATAWIDTH_9BIT

LL_SPI_DATAWIDTH_10BIT

LL_SPI_DATAWIDTH_11BIT

LL_SPI_DATAWIDTH_12BIT

LL_SPI_DATAWIDTH_13BIT

LL_SPI_DATAWIDTH_14BIT

LL_SPI_DATAWIDTH_15BIT

LL_SPI_DATAWIDTH_16BIT

LL_SPI_DATAWIDTH_17BIT

LL_SPI_DATAWIDTH_18BIT

LL_SPI_DATAWIDTH_19BIT

LL_SPI_DATAWIDTH_20BIT

LL_SPI_DATAWIDTH_21BIT

LL_SPI_DATAWIDTH_22BIT

LL_SPI_DATAWIDTH_23BIT

LL_SPI_DATAWIDTH_24BIT

LL_SPI_DATAWIDTH_25BIT

LL_SPI_DATAWIDTH_26BIT

LL_SPI_DATAWIDTH_27BIT

LL_SPI_DATAWIDTH_28BIT

LL_SPI_DATAWIDTH_29BIT

LL_SPI_DATAWIDTH_30BIT

LL_SPI_DATAWIDTH_31BIT

LL_SPI_DATAWIDTH_32BIT

FIFO Threshold

LL_SPI_FIFO_TH_01DATA

LL_SPI_FIFO_TH_02DATA

LL_SPI_FIFO_TH_03DATA

LL_SPI_FIFO_TH_04DATA

LL_SPI_FIFO_TH_05DATA

LL_SPI_FIFO_TH_06DATA

LL_SPI_FIFO_TH_07DATA

LL_SPI_FIFO_TH_08DATA

LL_SPI_FIFO_TH_09DATA

LL_SPI_FIFO_TH_10DATA

LL_SPI_FIFO_TH_11DATA

LL_SPI_FIFO_TH_12DATA

LL_SPI_FIFO_TH_13DATA

LL_SPI_FIFO_TH_14DATA

LL_SPI_FIFO_TH_15DATA

LL_SPI_FIFO_TH_16DATA

Get Flags Defines

LL_SPI_SR_RXP

LL_SPI_SR_TXP

LL_SPI_SR_DXP

LL_SPI_SR_EOT

LL_SPI_SR_TXTF

LL_SPI_SR_UDR

LL_SPI_SR_CRCE

LL_SPI_SR_MODF

LL_SPI_SR_OVR

LL_SPI_SR_TIFRE

LL_SPI_SR_SUSP

LL_SPI_SR_TXC

LL_SPI_SR_RXWNE

Master Inter-Data Idleness

LL_SPI_ID_IDLENESS_00CYCLE

LL_SPI_ID_IDLENESS_01CYCLE

LL_SPI_ID_IDLENESS_02CYCLE

LL_SPI_ID_IDLENESS_03CYCLE

LL_SPI_ID_IDLENESS_04CYCLE

LL_SPI_ID_IDLENESS_05CYCLE

LL_SPI_ID_IDLENESS_06CYCLE

LL_SPI_ID_IDLENESS_07CYCLE

LL_SPI_ID_IDLENESS_08CYCLE

LL_SPI_ID_IDLENESS_09CYCLE

LL_SPI_ID_IDLENESS_10CYCLE

LL_SPI_ID_IDLENESS_11CYCLE

LL_SPI_ID_IDLENESS_12CYCLE

LL_SPI_ID_IDLENESS_13CYCLE

LL_SPI_ID_IDLENESS_14CYCLE

LL_SPI_ID_IDLENESS_15CYCLE

IT Defines

LL_SPI_IER_RXPIE

LL_SPI_IER_TXPIE

LL_SPI_IER_DXPIE

LL_SPI_IER_EOTIE

LL_SPI_IER_TXTFIE

LL_SPI_IER_UDRIE

LL_SPI_IER_OVRIE

LL_SPI_IER_CRCEIE

LL_SPI_IER_TIFREIE

LL_SPI_IER_MODFIE

Mode

LL_SPI_MODE_MASTER

LL_SPI_MODE_SLAVE

NSS Mode

LL_SPI_NSS_SOFT

LL_SPI_NSS_HARD_INPUT

LL_SPI_NSS_HARD_OUTPUT

NSS Polarity

LL_SPI_NSS_POLARITY_LOW

LL_SPI_NSS_POLARITY_HIGH

Phase

LL_SPI_PHASE_1EDGE

LL_SPI_PHASE_2EDGE

Polarity

LL_SPI_POLARITY_LOW

LL_SPI_POLARITY_HIGH

Protocol

LL_SPI_PROTOCOL_MOTOROLA

LL_SPI_PROTOCOL_TI

RXCRC Init All

LL_SPI_RXCRCINIT_ALL_ZERO_PATTERN

LL_SPI_RXCRCINIT_ALL_ONES_PATTERN

RxFIFO Packing LeVel

LL_SPI_RX_FIFO_0PACKET

LL_SPI_RX_FIFO_1PACKET

LL_SPI_RX_FIFO_2PACKET

LL_SPI_RX_FIFO_3PACKET

SS Idleness

LL_SPI_SS_IDLENESS_00CYCLE

LL_SPI_SS_IDLENESS_01CYCLE

LL_SPI_SS_IDLENESS_02CYCLE

LL_SPI_SS_IDLENESS_03CYCLE

LL_SPI_SS_IDLENESS_04CYCLE

LL_SPI_SS_IDLENESS_05CYCLE

LL_SPI_SS_IDLENESS_06CYCLE

LL_SPI_SS_IDLENESS_07CYCLE

LL_SPI_SS_IDLENESS_08CYCLE

LL_SPI_SS_IDLENESS_09CYCLE

LL_SPI_SS_IDLENESS_10CYCLE

LL_SPI_SS_IDLENESS_11CYCLE

LL_SPI_SS_IDLENESS_12CYCLE

LL_SPI_SS_IDLENESS_13CYCLE

LL_SPI_SS_IDLENESS_14CYCLE

LL_SPI_SS_IDLENESS_15CYCLE

SS Level

LL_SPI_SS_LEVEL_HIGH

LL_SPI_SS_LEVEL_LOW

Transfer Mode

LL_SPI_FULL_DUPLEX

LL_SPI_SIMPLEX_TX

LL_SPI_SIMPLEX_RX

LL_SPI_HALF_DUPLEX_RX

LL_SPI_HALF_DUPLEX_TX

TXCRC Init All

LL_SPI_TXCRCINIT_ALL_ZERO_PATTERN

LL_SPI_TXCRCINIT_ALL_ONES_PATTERN

UDR Config Register

LL_SPI_UDR_CONFIG_REGISTER_PATTERN

LL_SPI_UDR_CONFIG_LAST_RECEIVED

Common Write and read registers Macros

LL_SPI_WriteReg

Description:

- Write a value in SPI register.

Parameters:

- __INSTANCE__: SPI Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_SPI_ReadReg

Description:

- Read a value in SPI register.

Parameters:

- __INSTANCE__: SPI Instance
- __REG__: Register to be read

Return value:

- Register: value

105 LL SYSTEM Generic Driver

105.1 SYSTEM Firmware driver defines

The following section lists the various define and macros of the module.

105.1.1 SYSTEM

SYSTEM

DBGMCU AHB1 GRP1 STOP IP

LL_DBGMCU_AHB1_GRP1_GPDMA1_CH0_STOP

GPDMA1_CH0 suspend in debug

LL_DBGMCU_AHB1_GRP1_GPDMA1_CH1_STOP

GPDMA1_CH1 suspend in debug

LL_DBGMCU_AHB1_GRP1_GPDMA1_CH2_STOP

GPDMA1_CH2 suspend in debug

LL_DBGMCU_AHB1_GRP1_GPDMA1_CH3_STOP

GPDMA1_CH3 suspend in debug

LL_DBGMCU_AHB1_GRP1_GPDMA1_CH4_STOP

GPDMA1_CH4 suspend in debug

LL_DBGMCU_AHB1_GRP1_GPDMA1_CH5_STOP

GPDMA1_CH5 suspend in debug

LL_DBGMCU_AHB1_GRP1_GPDMA1_CH6_STOP

GPDMA1_CH6 suspend in debug

LL_DBGMCU_AHB1_GRP1_GPDMA1_CH7_STOP

GPDMA1_CH7 suspend in debug

LL_DBGMCU_AHB1_GRP1_GPDMA1_CH8_STOP

GPDMA1_CH8 suspend in debug

LL_DBGMCU_AHB1_GRP1_GPDMA1_CH9_STOP

GPDMA1_CH9 suspend in debug

LL_DBGMCU_AHB1_GRP1_GPDMA1_CH10_STOP

GPDMA1_CH10 suspend in debug

LL_DBGMCU_AHB1_GRP1_GPDMA1_CH11_STOP

GPDMA1_CH11 suspend in debug

LL_DBGMCU_AHB1_GRP1_GPDMA1_CH12_STOP

GPDMA1_CH12 suspend in debug

LL_DBGMCU_AHB1_GRP1_GPDMA1_CH13_STOP

GPDMA1_CH13 suspend in debug

LL_DBGMCU_AHB1_GRP1_GPDMA1_CH14_STOP

GPDMA1_CH14 suspend in debug

LL_DBGMCU_AHB1_GRP1_GPDMA1_CH15_STOP

GPDMA1_CH15 suspend in debug

DBGMCU AHB5 GRP1 STOP IP

LL_DBGMCU_AHB5_GRP1_HPDMA1_CH0_STOP

HPDMA1_CH0 suspend in debug

LL_DBGMCU_AHB5_GRP1_HPDMA1_CH1_STOP

HPDMA1_CH1 suspend in debug

LL_DBGMCU_AHB5_GRP1_HPDMA1_CH2_STOP

HPDMA1_CH2 suspend in debug

LL_DBGMCU_AHB5_GRP1_HPDMA1_CH3_STOP

HPDMA1_CH3 suspend in debug

LL_DBGMCU_AHB5_GRP1_HPDMA1_CH4_STOP

HPDMA1_CH4 suspend in debug

LL_DBGMCU_AHB5_GRP1_HPDMA1_CH5_STOP

HPDMA1_CH5 suspend in debug

LL_DBGMCU_AHB5_GRP1_HPDMA1_CH6_STOP

HPDMA1_CH6 suspend in debug

LL_DBGMCU_AHB5_GRP1_HPDMA1_CH7_STOP

HPDMA1_CH7 suspend in debug

LL_DBGMCU_AHB5_GRP1_HPDMA1_CH8_STOP

HPDMA1_CH8 suspend in debug

LL_DBGMCU_AHB5_GRP1_HPDMA1_CH9_STOP

HPDMA1_CH9 suspend in debug

LL_DBGMCU_AHB5_GRP1_HPDMA1_CH10_STOP

HPDMA1_CH10 suspend in debug

LL_DBGMCU_AHB5_GRP1_HPDMA1_CH11_STOP

HPDMA1_CH11 suspend in debug

LL_DBGMCU_AHB5_GRP1_HPDMA1_CH12_STOP

HPDMA1_CH12 suspend in debug

LL_DBGMCU_AHB5_GRP1_HPDMA1_CH13_STOP

HPDMA1_CH13 suspend in debug

LL_DBGMCU_AHB5_GRP1_HPDMA1_CH14_STOP

HPDMA1_CH14 suspend in debug

LL_DBGMCU_AHB5_GRP1_HPDMA1_CH15_STOP

HPDMA1_CH15 suspend in debug

LL_DBGMCU_AHB5_GRP1_NPU_STOP

NPU stop in debug mode

DBGMCU APB1 GRP1 STOP IP

LL_DBGMCU_APB1_GRP1_TIM2_STOP

LL_DBGMCU_APB1_GRP1_TIM3_STOP

TIM3 stop in debug

LL_DBGMCU_APB1_GRP1_TIM4_STOP

TIM4 stop in debug

LL_DBGMCU_APB1_GRP1_TIM5_STOP

TIM5 stop in debug

LL_DBGMCU_APB1_GRP1_TIM6_STOP

TIM6 stop in debug

LL_DBGMCU_APB1_GRP1_TIM7_STOP

TIM7 stop in debug

LL_DBGMCU_APB1_GRP1_TIM12_STOP

TIM12 stop in debug

LL_DBGMCU_APB1_GRP1_TIM13_STOP

TIM13 stop in debug

LL_DBGMCU_APB1_GRP1_TIM14_STOP

TIM14 stop in debug

LL_DBGMCU_APB1_GRP1_LPTIM1_STOP

LPTIM1 stop in debug

LL_DBGMCU_APB1_GRP1_WWDG_STOP

WWDG1 stop in debug

LL_DBGMCU_APB1_GRP1_TIM10_STOP

TIM10 stop in debug

LL_DBGMCU_APB1_GRP1_TIM11_STOP

TIM11 stop in debug

LL_DBGMCU_APB1_GRP1_I2C1_STOP

I2C1 SMBUS timeout stop in debug

LL_DBGMCU_APB1_GRP1_I2C2_STOP

I2C2 SMBUS timeout stop in debug

LL_DBGMCU_APB1_GRP1_I2C3_STOP

I2C3 SMBUS timeout stop in debug

LL_DBGMCU_APB1_GRP1_I3C1_STOP

I3C1 SMBUS timeout stop in debug

LL_DBGMCU_APB1_GRP1_I3C2_STOP

I3C2 SMBUS timeout stop in debug

DBGMCU APB1 GRP2 STOP IP

LL_DBGMCU_APB1_GRP2_FDCAN_STOP

FDCAN stop in debug

DBGMCU APB2 GRP1 STOP IP

LL_DBGMCU_APB2_GRP1_TIM1_STOP

TIM1 stop in debug

LL_DBGMCU_APB2_GRP1_TIM8_STOP

TIM8 stop in debug

LL_DBGMCU_APB2_GRP1_TIM18_STOP

TIM18 stop in debug

LL_DBGMCU_APB2_GRP1_TIM15_STOP

TIM15 stop in debug

LL_DBGMCU_APB2_GRP1_TIM16_STOP

TIM16 stop in debug

LL_DBGMCU_APB2_GRP1_TIM17_STOP

TIM17 stop in debug

LL_DBGMCU_APB2_GRP1_TIM9_STOP

TIM9 stop in debug

DBGMCU APB4 GRP1 STOP IP

LL_DBGMCU_APB4_GRP1_I2C4_STOP

I2C4 stop in debug

LL_DBGMCU_APB4_GRP1_LPTIM2_STOP

LPTIM2 stop in debug

LL_DBGMCU_APB4_GRP1_LPTIM3_STOP

LPTIM3 stop in debug

LL_DBGMCU_APB4_GRP1_LPTIM4_STOP

LPTIM4 stop in debug

LL_DBGMCU_APB4_GRP1_LPTIM5_STOP

LPTIM5 stop in debug

LL_DBGMCU_APB4_GRP1_RTC_STOP

RTC stop in debug

LL_DBGMCU_APB4_GRP1_IWDG_STOP

IWDG stop in debug

DBGMCU APB5 GRP1 STOP IP

LL_DBGMCU_APB5_GRP1_GFXTIM_STOP

GFXTIM stop in debug

SYSCFG Vdd compensation cell Code selection

LL_SYSCFG_VDD_CELL_CODE

LL_SYSCFG_VDD_REGISTER_CODE

EXTI EXTI LINE

LL_EXTI_EXTI_LINE0

EXTI_POSITION_0 | EXTICR[0]

LL_EXTI_EXTI_LINE1

EXTI_POSITION_8 | EXTICR[0]

LL_EXTI_EXTI_LINE2

EXTI_POSITION_16 | EXTICR[0]

LL_EXTI_EXTI_LINE3

EXTI_POSITION_24 | EXTICR[0]

LL_EXTI_EXTI_LINE4

EXTI_POSITION_0 | EXTICR[1]

LL_EXTI_EXTI_LINE5

EXTI_POSITION_8 | EXTICR[1]

LL_EXTI_EXTI_LINE6

EXTI_POSITION_16 | EXTICR[1]

LL_EXTI_EXTI_LINE7

EXTI_POSITION_24 | EXTICR[1]

LL_EXTI_EXTI_LINE8

EXTI_POSITION_0 | EXTICR[2]

LL_EXTI_EXTI_LINE9

EXTI_POSITION_8 | EXTICR[2]

LL_EXTI_EXTI_LINE10

EXTI_POSITION_16 | EXTICR[2]

LL_EXTI_EXTI_LINE11

EXTI_POSITION_24 | EXTICR[2]

LL_EXTI_EXTI_LINE12

EXTI_POSITION_0 | EXTICR[3]

LL_EXTI_EXTI_LINE13

EXTI_POSITION_8 | EXTICR[3]

LL_EXTI_EXTI_LINE14

EXTI_POSITION_16 | EXTICR[3]

LL_EXTI_EXTI_LINE15

EXTI_POSITION_24 | EXTICR[3]

EXTI EXTI PORT

LL_EXTI_EXTI_PORTA

EXTI PORT A

LL_EXTI_EXTI_PORTB

EXTI PORT B

LL_EXTI_EXTI_PORTC

EXTI PORT C

LL_EXTI_EXTI_PORTD

EXTI PORT D

LL_EXTI_EXTI_PORTE

EXTI PORT E

LL_EXTI_EXTI_PORTF

EXTI PORT F

LL_EXTI_EXTI_PORTG

EXTI PORT G

LL_EXTI_EXTI_PORTH

EXTI PORT H

LL_EXTI_EXTI_PORTN

EXTI PORT N

LL_EXTI_EXTI_PORTO

EXTI PORT O

LL_EXTI_EXTI_PORTP

EXTI PORT P

LL_EXTI_EXTI_PORTQ

EXTI PORT Q

SYSCFG I2C FASTMODEPLUS

LL_SYSCFG_I2C_FASTMODEPLUS_PA6

Enable Fast Mode Plus on PA6

LL_SYSCFG_I2C_FASTMODEPLUS_PA7

Enable Fast Mode Plus on PA7

LL_SYSCFG_I2C_FASTMODEPLUS_PA15

Enable Fast Mode Plus on PA15

LL_SYSCFG_I2C_FASTMODEPLUS_PB3

Enable Fast Mode Plus on PB3

Secure attributes

LL_SYSCFG_CLOCK_SEC

SYSCFG clock configuration secure-only access

LL_SYSCFG_CLOCK_NSEC

SYSCFG clock configuration secure/non-secure access

LL_SYSCFG_CLASSB_SEC

Class B configuration secure-only access

LL_SYSCFG_CLASSB_NSEC

Class B configuration secure/non-secure access

LL_SYSCFG_FPU_SEC

FPU configuration secure-only access

LL_SYSCFG_FPU_NSEC

FPU configuration secure/non-secure access

SYSCFG TIMER BREAK

LL_SYSCFG_TIMBREAK_ECC

Enables and locks the ECC error signal with Break Input of TIM1/16/17

LL_SYSCFG_TIMBREAK_PVD

Enables and locks the PVD connection with TIM1/16/17 Break Input and also the PVDE and PLS bits of the Power Control Interface

LL_SYSCFG_TIMBREAK_SRAM_ECC_LOCK

Enables and locks the SRAM ECC double error signal with Break Input of TIM1/16/17

LL_SYSCFG_TIMBREAK_LOCKUP

Enables and locks the LOCKUP output of CortexM33 with Break Input of TIM1/16/17

LL_DBGMCU_DBTRGIN

DBTRGIO connected to DBTRGIN

LL_DBGMCU_DBTRGOUT

DBTRGIO connected to DBTRGOUT

106 LL TIM Generic Driver

106.1 TIM Firmware driver registers structures

106.1.1 LL_TIM_InitTypeDef

LL_TIM_InitTypeDef is defined in the stm32n6xx_ll_tim.h

Data Fields

- ***uint16_t Prescaler***
- ***uint32_t CounterMode***
- ***uint32_t Autoreload***
- ***uint32_t ClockDivision***
- ***uint32_t RepetitionCounter***

Field Documentation

- ***uint16_t LL_TIM_InitTypeDef::Prescaler***
  Specifies the prescaler value used to divide the TIM clock. This parameter can be a number between Min_Data=0x0000 and Max_Data=0xFFFF.This feature can be modified afterwards using unitary function **LL_TIM_SetPrescaler()**.
- ***uint32_t LL_TIM_InitTypeDef::CounterMode***
  Specifies the counter mode. This parameter can be a value of ***TIM_LL_EC_COUNTERMODE***.This feature can be modified afterwards using unitary function **LL_TIM_SetCounterMode()**.
- ***uint32_t LL_TIM_InitTypeDef::Autoreload***
  Specifies the auto reload value to be loaded into the active Auto-Reload Register at the next update event. This parameter must be a number between Min_Data=0x0000 and Max_Data=0xFFFF. Some timer instances may support 32 bits counters. In that case this parameter must be a number between 0x0000 and 0xFFFFFFFF.This feature can be modified afterwards using unitary function **LL_TIM_SetAutoReload()**.
- ***uint32_t LL_TIM_InitTypeDef::ClockDivision***
  Specifies the clock division. This parameter can be a value of ***TIM_LL_EC_CLOCKDIVISION***.This feature can be modified afterwards using unitary function **LL_TIM_SetClockDivision()**.
- ***uint32_t LL_TIM_InitTypeDef::RepetitionCounter***
  Specifies the repetition counter value. Each time the RCR downcounter reaches zero, an update event is generated and counting restarts from the RCR value (N). This means in PWM mode that (N+1) corresponds to:
  - the number of PWM periods in edge-aligned mode
  - the number of half PWM period in center-aligned mode GP timers: this parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF. Advanced timers: this parameter must be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF.

  This feature can be modified afterwards using unitary function **LL_TIM_SetRepetitionCounter()**.

106.1.2 LL_TIM_OC_InitTypeDef

LL_TIM_OC_InitTypeDef is defined in the stm32n6xx_ll_tim.h

Data Fields

- ***uint32_t OCMode***
- ***uint32_t OCState***
- ***uint32_t OCNState***
- ***uint32_t CompareValue***
- ***uint32_t OCPolarity***
- ***uint32_t OCNPolarity***
- ***uint32_t OCIdleState***
- ***uint32_t OCNIdleState***

Field Documentation

- ***uint32_t LL_TIM_OC_InitTypeDef::OCMode***
  Specifies the output mode. This parameter can be a value of ***TIM_LL_EC_OCMODE***.This feature can be modified afterwards using unitary function **LL_TIM_OC_SetMode()**.
- ***uint32_t LL_TIM_OC_InitTypeDef::OCState***
  Specifies the TIM Output Compare state. This parameter can be a value of ***TIM_LL_EC_OCSTATE***.This feature can be modified afterwards using unitary functions **LL_TIM_CC_EnableChannel()** or **LL_TIM_CC_DisableChannel()**.
- ***uint32_t LL_TIM_OC_InitTypeDef::OCNState***
  Specifies the TIM complementary Output Compare state. This parameter can be a value of ***TIM_LL_EC_OCSTATE***.This feature can be modified afterwards using unitary functions **LL_TIM_CC_EnableChannel()** or **LL_TIM_CC_DisableChannel()**.
- ***uint32_t LL_TIM_OC_InitTypeDef::CompareValue***
  Specifies the Compare value to be loaded into the Capture Compare Register. This parameter can be a number between Min_Data=0x0000 and Max_Data=0xFFFF.This feature can be modified afterwards using unitary function LL_TIM_OC_SetCompareCHx (x=1..6).
- ***uint32_t LL_TIM_OC_InitTypeDef::OCPolarity***
  Specifies the output polarity. This parameter can be a value of ***TIM_LL_EC_OCPOLARITY***.This feature can be modified afterwards using unitary function **LL_TIM_OC_SetPolarity()**.
- ***uint32_t LL_TIM_OC_InitTypeDef::OCNPolarity***
  Specifies the complementary output polarity. This parameter can be a value of ***TIM_LL_EC_OCPOLARITY***.This feature can be modified afterwards using unitary function **LL_TIM_OC_SetPolarity()**.
- ***uint32_t LL_TIM_OC_InitTypeDef::OCIdleState***
  Specifies the TIM Output Compare pin state during Idle state. This parameter can be a value of ***TIM_LL_EC_OCIDLESTATE***.This feature can be modified afterwards using unitary function **LL_TIM_OC_SetIdleState()**.
- ***uint32_t LL_TIM_OC_InitTypeDef::OCNIdleState***
  Specifies the TIM Output Compare pin state during Idle state. This parameter can be a value of ***TIM_LL_EC_OCIDLESTATE***.This feature can be modified afterwards using unitary function **LL_TIM_OC_SetIdleState()**.

106.1.3 LL_TIM_IC_InitTypeDef

LL_TIM_IC_InitTypeDef is defined in the stm32n6xx_ll_tim.h

Data Fields

- ***uint32_t ICPolarity***
- ***uint32_t ICActiveInput***
- ***uint32_t ICPrescaler***
- ***uint32_t ICFilter***

Field Documentation

- ***uint32_t LL_TIM_IC_InitTypeDef::ICPolarity***
  Specifies the active edge of the input signal. This parameter can be a value of ***TIM_LL_EC_IC_POLARITY***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetPolarity()**.
- ***uint32_t LL_TIM_IC_InitTypeDef::ICActiveInput***
  Specifies the input. This parameter can be a value of ***TIM_LL_EC_ACTIVEINPUT***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetActiveInput()**.
- ***uint32_t LL_TIM_IC_InitTypeDef::ICPrescaler***
  Specifies the Input Capture Prescaler. This parameter can be a value of ***TIM_LL_EC_ICPSC***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetPrescaler()**.
- ***uint32_t LL_TIM_IC_InitTypeDef::ICFilter***
  Specifies the input capture filter. This parameter can be a value of ***TIM_LL_EC_IC_FILTER***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetFilter()**.

106.1.4 LL_TIM_ENCODER_InitTypeDef

LL_TIM_ENCODER_InitTypeDef is defined in the stm32n6xx_ll_tim.h

Data Fields

- ***uint32_t EncoderMode***
- ***uint32_t IC1Polarity***
- ***uint32_t IC1ActiveInput***
- ***uint32_t IC1Prescaler***
- ***uint32_t IC1Filter***
- ***uint32_t IC2Polarity***
- ***uint32_t IC2ActiveInput***
- ***uint32_t IC2Prescaler***
- ***uint32_t IC2Filter***

Field Documentation

- ***uint32_t LL_TIM_ENCODER_InitTypeDef::EncoderMode***
  Specifies the encoder resolution (x2 or x4). This parameter can be a value of ***TIM_LL_EC_ENCODERMODE***.This feature can be modified afterwards using unitary function **LL_TIM_SetEncoderMode()**.
- ***uint32_t LL_TIM_ENCODER_InitTypeDef::IC1Polarity***
  Specifies the active edge of TI1 input. This parameter can be a value of ***TIM_LL_EC_IC_POLARITY***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetPolarity()**.
- ***uint32_t LL_TIM_ENCODER_InitTypeDef::IC1ActiveInput***
  Specifies the TI1 input source This parameter can be a value of ***TIM_LL_EC_ACTIVEINPUT***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetActiveInput()**.
- ***uint32_t LL_TIM_ENCODER_InitTypeDef::IC1Prescaler***
  Specifies the TI1 input prescaler value. This parameter can be a value of ***TIM_LL_EC_ICPSC***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetPrescaler()**.
- ***uint32_t LL_TIM_ENCODER_InitTypeDef::IC1Filter***
  Specifies the TI1 input filter. This parameter can be a value of ***TIM_LL_EC_IC_FILTER***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetFilter()**.
- ***uint32_t LL_TIM_ENCODER_InitTypeDef::IC2Polarity***
  Specifies the active edge of TI2 input. This parameter can be a value of ***TIM_LL_EC_IC_POLARITY***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetPolarity()**.
- ***uint32_t LL_TIM_ENCODER_InitTypeDef::IC2ActiveInput***
  Specifies the TI2 input source This parameter can be a value of ***TIM_LL_EC_ACTIVEINPUT***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetActiveInput()**.
- ***uint32_t LL_TIM_ENCODER_InitTypeDef::IC2Prescaler***
  Specifies the TI2 input prescaler value. This parameter can be a value of ***TIM_LL_EC_ICPSC***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetPrescaler()**.
- ***uint32_t LL_TIM_ENCODER_InitTypeDef::IC2Filter***
  Specifies the TI2 input filter. This parameter can be a value of ***TIM_LL_EC_IC_FILTER***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetFilter()**.

106.1.5 LL_TIM_HALLSENSOR_InitTypeDef

LL_TIM_HALLSENSOR_InitTypeDef is defined in the stm32n6xx_ll_tim.h

Data Fields

- ***uint32_t IC1Polarity***
- ***uint32_t IC1Prescaler***
- ***uint32_t IC1Filter***
- ***uint32_t CommutationDelay***

Field Documentation

- ***uint32_t LL_TIM_HALLSENSOR_InitTypeDef::IC1Polarity***
  Specifies the active edge of TI1 input. This parameter can be a value of ***TIM_LL_EC_IC_POLARITY***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetPolarity()**.
- ***uint32_t LL_TIM_HALLSENSOR_InitTypeDef::IC1Prescaler***
  Specifies the TI1 input prescaler value. Prescaler must be set to get a maximum counter period longer than the time interval between 2 consecutive changes on the Hall inputs. This parameter can be a value of ***TIM_LL_EC_ICPSC***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetPrescaler()**.

- ***uint32_t LL_TIM_HALLSENSOR_InitTypeDef::IC1Filter***
  Specifies the TI1 input filter. This parameter can be a value of ***TIM_LL_EC_IC_FILTER***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetFilter()**.
- ***uint32_t LL_TIM_HALLSENSOR_InitTypeDef::CommutationDelay***
  Specifies the compare value to be loaded into the Capture Compare Register. A positive pulse (TRGO event) is generated with a programmable delay every time a change occurs on the Hall inputs. This parameter can be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF.This feature can be modified afterwards using unitary function **LL_TIM_OC_SetCompareCH2()**.

106.1.6 LL_TIM_BDTR_InitTypeDef

LL_TIM_BDTR_InitTypeDef is defined in the stm32n6xx_ll_tim.h

Data Fields

- ***uint32_t OSSRState***
- ***uint32_t OSSIState***
- ***uint32_t LockLevel***
- ***uint8_t DeadTime***
- ***uint16_t BreakState***
- ***uint32_t BreakPolarity***
- ***uint32_t BreakFilter***
- ***uint32_t BreakAFMode***
- ***uint32_t Break2State***
- ***uint32_t Break2Polarity***
- ***uint32_t Break2Filter***
- ***uint32_t Break2AFMode***
- ***uint32_t AutomaticOutput***

Field Documentation

- ***uint32_t LL_TIM_BDTR_InitTypeDef::OSSRState***
  Specifies the Off-State selection used in Run mode. This parameter can be a value of ***TIM_LL_EC_OSSR***This feature can be modified afterwards using unitary function **LL_TIM_SetOffStates()**
  Note:
  – This bit-field cannot be modified as long as LOCK level 2 has been programmed.
- ***uint32_t LL_TIM_BDTR_InitTypeDef::OSSIState***
  Specifies the Off-State used in Idle state. This parameter can be a value of ***TIM_LL_EC_OSSI***This feature can be modified afterwards using unitary function **LL_TIM_SetOffStates()**
  Note:
  – This bit-field cannot be modified as long as LOCK level 2 has been programmed.
- ***uint32_t LL_TIM_BDTR_InitTypeDef::LockLevel***
  Specifies the LOCK level parameters. This parameter can be a value of ***TIM_LL_EC_LOCKLEVEL***
  Note:
  – The LOCK bits can be written only once after the reset. Once the TIMx_BDTR register has been written, their content is frozen until the next reset.
- ***uint8_t LL_TIM_BDTR_InitTypeDef::DeadTime***
  Specifies the delay time between the switching-off and the switching-on of the outputs. This parameter can be a number between Min_Data = 0x00 and Max_Data = 0xFF.This feature can be modified afterwards using unitary function **LL_TIM_OC_SetDeadTime()**
  Note:
  – This bit-field can not be modified as long as LOCK level 1, 2 or 3 has been programmed.
- ***uint16_t LL_TIM_BDTR_InitTypeDef::BreakState***
  Specifies whether the TIM Break input is enabled or not. This parameter can be a value of ***TIM_LL_EC_BREAK_ENABLE***This feature can be modified afterwards using unitary functions **LL_TIM_EnableBRK()** or **LL_TIM_DisableBRK()**
  Note:
  – This bit-field can not be modified as long as LOCK level 1 has been programmed.

- ***uint32_t LL_TIM_BDTR_InitTypeDef::BreakPolarity***
  Specifies the TIM Break Input pin polarity. This parameter can be a value of ***TIM_LL_EC_BREAK_POLARITY***This feature can be modified afterwards using unitary function **LL_TIM_ConfigBRK()**
  Note:
  – This bit-field can not be modified as long as LOCK level 1 has been programmed.
- ***uint32_t LL_TIM_BDTR_InitTypeDef::BreakFilter***
  Specifies the TIM Break Filter. This parameter can be a value of ***TIM_LL_EC_BREAK_FILTER***This feature can be modified afterwards using unitary function **LL_TIM_ConfigBRK()**
  Note:
  – This bit-field can not be modified as long as LOCK level 1 has been programmed.
- ***uint32_t LL_TIM_BDTR_InitTypeDef::BreakAFMode***
  Specifies the alternate function mode of the break input. This parameter can be a value of ***TIM_LL_EC_BREAK_AFMODE***This feature can be modified afterwards using unitary functions **LL_TIM_ConfigBRK()**
  Note:
  – Bidirectional break input is only supported by advanced timers instances.
  – This bit-field can not be modified as long as LOCK level 1 has been programmed.
- ***uint32_t LL_TIM_BDTR_InitTypeDef::Break2State***
  Specifies whether the TIM Break2 input is enabled or not. This parameter can be a value of ***TIM_LL_EC_BREAK2_ENABLE***This feature can be modified afterwards using unitary functions **LL_TIM_EnableBRK2()** or **LL_TIM_DisableBRK2()**
  Note:
  – This bit-field can not be modified as long as LOCK level 1 has been programmed.
- ***uint32_t LL_TIM_BDTR_InitTypeDef::Break2Polarity***
  Specifies the TIM Break2 Input pin polarity. This parameter can be a value of ***TIM_LL_EC_BREAK2_POLARITY***This feature can be modified afterwards using unitary function **LL_TIM_ConfigBRK2()**
  Note:
  – This bit-field can not be modified as long as LOCK level 1 has been programmed.
- ***uint32_t LL_TIM_BDTR_InitTypeDef::Break2Filter***
  Specifies the TIM Break2 Filter. This parameter can be a value of ***TIM_LL_EC_BREAK2_FILTER***This feature can be modified afterwards using unitary function **LL_TIM_ConfigBRK2()**
  Note:
  – This bit-field can not be modified as long as LOCK level 1 has been programmed.
- ***uint32_t LL_TIM_BDTR_InitTypeDef::Break2AFMode***
  Specifies the alternate function mode of the break2 input. This parameter can be a value of ***TIM_LL_EC_BREAK2_AFMODE***This feature can be modified afterwards using unitary functions **LL_TIM_ConfigBRK2()**
  Note:
  – Bidirectional break input is only supported by advanced timers instances.
  – This bit-field can not be modified as long as LOCK level 1 has been programmed.
- ***uint32_t LL_TIM_BDTR_InitTypeDef::AutomaticOutput***
  Specifies whether the TIM Automatic Output feature is enabled or not. This parameter can be a value of ***TIM_LL_EC_AUTOMATICOUTPUT_ENABLE***This feature can be modified afterwards using unitary functions **LL_TIM_EnableAutomaticOutput()** or **LL_TIM_DisableAutomaticOutput()**
  Note:
  – This bit-field can not be modified as long as LOCK level 1 has been programmed.

106.2 TIM Firmware driver API description

The following section lists the various functions of the TIM library.

106.2.1 Detailed description of functions

LL_TIM_EnableCounter

Function name

__STATIC_INLINE void LL_TIM_EnableCounter (TIM_TypeDef * TIMx)

Function description

Enable timer counter.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 CEN LL_TIM_EnableCounter

LL_TIM_DisableCounter

Function name

__STATIC_INLINE void LL_TIM_DisableCounter (TIM_TypeDef * TIMx)

Function description

Disable timer counter.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 CEN LL_TIM_DisableCounter

LL_TIM_IsEnabledCounter

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledCounter (const TIM_TypeDef * TIMx)

Function description

Indicates whether the timer counter is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 CEN LL_TIM_IsEnabledCounter

LL_TIM_EnableUpdateEvent

Function name

__STATIC_INLINE void LL_TIM_EnableUpdateEvent (TIM_TypeDef * TIMx)

Function description

Enable update event generation.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 UDIS LL_TIM_EnableUpdateEvent

LL_TIM_DisableUpdateEvent

Function name

__STATIC_INLINE void LL_TIM_DisableUpdateEvent (TIM_TypeDef * TIMx)

Function description

Disable update event generation.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 UDIS LL_TIM_DisableUpdateEvent

LL_TIM_IsEnabledUpdateEvent

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledUpdateEvent (const TIM_TypeDef * TIMx)

Function description

Indicates whether update event generation is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **Inverted:** state of bit (0 or 1).

Reference Manual to LL API cross reference:

- CR1 UDIS LL_TIM_IsEnabledUpdateEvent

LL_TIM_SetUpdateSource

Function name

__STATIC_INLINE void LL_TIM_SetUpdateSource (TIM_TypeDef * TIMx, uint32_t UpdateSource)

Function description

Set update event source.

Parameters

- **TIMx:** Timer instance
- **UpdateSource:** This parameter can be one of the following values:
  - LL_TIM_UPDATESOURCE_REGULAR
  - LL_TIM_UPDATESOURCE_COUNTER

Return values

- **None:**

Notes

- Update event source set to LL_TIM_UPDATESOURCE_REGULAR: any of the following events generate an update interrupt or DMA request if enabled: Counter overflow/underflowSetting the UG bitUpdate generation through the slave mode controller
- Update event source set to LL_TIM_UPDATESOURCE_COUNTER: only counter overflow/underflow generates an update interrupt or DMA request if enabled.

Reference Manual to LL API cross reference:

- CR1 URS LL_TIM_SetUpdateSource

LL_TIM_GetUpdateSource

Function name

__STATIC_INLINE uint32_t LL_TIM_GetUpdateSource (const TIM_TypeDef * TIMx)

Function description

Get actual event update source.

Parameters

- **TIMx:** Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_TIM_UPDATESOURCE_REGULAR
  - LL_TIM_UPDATESOURCE_COUNTER

Reference Manual to LL API cross reference:

- CR1 URS LL_TIM_GetUpdateSource

LL_TIM_SetOnePulseMode

Function name

__STATIC_INLINE void LL_TIM_SetOnePulseMode (TIM_TypeDef * TIMx, uint32_t OnePulseMode)

Function description

Set one pulse mode (one shot v.s.

Parameters

- **TIMx:** Timer instance
- **OnePulseMode:** This parameter can be one of the following values:
  - LL_TIM_ONEPULSEMODE_SINGLE
  - LL_TIM_ONEPULSEMODE_REPETITIVE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 OPM LL_TIM_SetOnePulseMode

LL_TIM_GetOnePulseMode

Function name

__STATIC_INLINE uint32_t LL_TIM_GetOnePulseMode (const TIM_TypeDef * TIMx)

Function description

Get actual one pulse mode.

Parameters

- **TIMx:** Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_TIM_ONEPULSEMODE_SINGLE
  - LL_TIM_ONEPULSEMODE_REPETITIVE

Reference Manual to LL API cross reference:

- CR1 OPM LL_TIM_GetOnePulseMode

LL_TIM_SetCounterMode

Function name

__STATIC_INLINE void LL_TIM_SetCounterMode (TIM_TypeDef * TIMx, uint32_t CounterMode)

Function description

Set the timer counter counting mode.

Parameters

- **TIMx:** Timer instance
- **CounterMode:** This parameter can be one of the following values:
  - LL_TIM_COUNTERMODE_UP
  - LL_TIM_COUNTERMODE_DOWN
  - LL_TIM_COUNTERMODE_CENTER_UP
  - LL_TIM_COUNTERMODE_CENTER_DOWN
  - LL_TIM_COUNTERMODE_CENTER_UP_DOWN

Return values

- **None:**

Notes

- Macro IS_TIM_COUNTER_MODE_SELECT_INSTANCE(TIMx) can be used to check whether or not the counter mode selection feature is supported by a timer instance.
- Switching from Center Aligned counter mode to Edge counter mode (or reverse) requires a timer reset to avoid unexpected direction due to DIR bit readonly in center aligned mode.

Reference Manual to LL API cross reference:

- CR1 DIR LL_TIM_SetCounterMode
- CR1 CMS LL_TIM_SetCounterMode

LL_TIM_GetCounterMode

Function name

__STATIC_INLINE uint32_t LL_TIM_GetCounterMode (const TIM_TypeDef * TIMx)

Function description

Get actual counter mode.

Parameters

- **TIMx:** Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_TIM_COUNTERMODE_UP
  - LL_TIM_COUNTERMODE_DOWN
  - LL_TIM_COUNTERMODE_CENTER_UP
  - LL_TIM_COUNTERMODE_CENTER_DOWN
  - LL_TIM_COUNTERMODE_CENTER_UP_DOWN

Notes

- Macro IS_TIM_COUNTER_MODE_SELECT_INSTANCE(TIMx) can be used to check whether or not the counter mode selection feature is supported by a timer instance.

Reference Manual to LL API cross reference:

- CR1 DIR LL_TIM_GetCounterMode
- CR1 CMS LL_TIM_GetCounterMode

LL_TIM_EnableARRPreload

Function name

__STATIC_INLINE void LL_TIM_EnableARRPreload (TIM_TypeDef * TIMx)

Function description

Enable auto-reload (ARR) preload.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 ARPE LL_TIM_EnableARRPreload

LL_TIM_DisableARRPreload

Function name

__STATIC_INLINE void LL_TIM_DisableARRPreload (TIM_TypeDef * TIMx)

Function description

Disable auto-reload (ARR) preload.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 ARPE LL_TIM_DisableARRPreload

LL_TIM_IsEnabledARRPreload

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledARRPreload (const TIM_TypeDef * TIMx)

Function description

Indicates whether auto-reload (ARR) preload is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 ARPE LL_TIM_IsEnabledARRPreload

LL_TIM_SetClockDivision

Function name

__STATIC_INLINE void LL_TIM_SetClockDivision (TIM_TypeDef * TIMx, uint32_t ClockDivision)

Function description

Set the division ratio between the timer clock and the sampling clock used by the dead-time generators (when supported) and the digital filters.

Parameters

- **TIMx:** Timer instance
- **ClockDivision:** This parameter can be one of the following values:
  - LL_TIM_CLOCKDIVISION_DIV1
  - LL_TIM_CLOCKDIVISION_DIV2
  - LL_TIM_CLOCKDIVISION_DIV4

Return values

- **None:**

Notes

- Macro IS_TIM_CLOCK_DIVISION_INSTANCE(TIMx) can be used to check whether or not the clock division feature is supported by the timer instance.

Reference Manual to LL API cross reference:

- CR1 CKD LL_TIM_SetClockDivision

LL_TIM_GetClockDivision

Function name

__STATIC_INLINE uint32_t LL_TIM_GetClockDivision (const TIM_TypeDef * TIMx)

Function description

Get the actual division ratio between the timer clock and the sampling clock used by the dead-time generators (when supported) and the digital filters.

Parameters

- **TIMx:** Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_TIM_CLOCKDIVISION_DIV1
  - LL_TIM_CLOCKDIVISION_DIV2
  - LL_TIM_CLOCKDIVISION_DIV4

Notes

- Macro IS_TIM_CLOCK_DIVISION_INSTANCE(TIMx) can be used to check whether or not the clock division feature is supported by the timer instance.

Reference Manual to LL API cross reference:

- CR1 CKD LL_TIM_GetClockDivision

LL_TIM_SetCounter

Function name

__STATIC_INLINE void LL_TIM_SetCounter (TIM_TypeDef * TIMx, uint32_t Counter)

Function description

Set the counter value.

Parameters

- **TIMx:** Timer instance
- **Counter:** Counter value (between Min_Data=0 and Max_Data=0xFFFF or 0xFFFFFFFF)

Return values

- **None:**

Notes

- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- If dithering is activated, pay attention to the Counter value interpretation

Reference Manual to LL API cross reference:

- CNT CNT LL_TIM_SetCounter

LL_TIM_GetCounter

Function name

__STATIC_INLINE uint32_t LL_TIM_GetCounter (const TIM_TypeDef * TIMx)

Function description

Get the counter value.

Parameters

- **TIMx:** Timer instance

Return values

- **Counter:** value (between Min_Data=0 and Max_Data=0xFFFF or 0xFFFFFFFF)

Notes

- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- If dithering is activated, pay attention to the Counter value interpretation

Reference Manual to LL API cross reference:

- CNT CNT LL_TIM_GetCounter

LL_TIM_GetDirection

Function name

__STATIC_INLINE uint32_t LL_TIM_GetDirection (const TIM_TypeDef * TIMx)

Function description

Get the current direction of the counter.

Parameters

- **TIMx:** Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_TIM_COUNTERDIRECTION_UP
  - LL_TIM_COUNTERDIRECTION_DOWN

Reference Manual to LL API cross reference:

- CR1 DIR LL_TIM_GetDirection

LL_TIM_SetPrescaler

Function name

__STATIC_INLINE void LL_TIM_SetPrescaler (TIM_TypeDef * TIMx, uint32_t Prescaler)

Function description

Set the prescaler value.

Parameters

- **TIMx:** Timer instance
- **Prescaler:** between Min_Data=0 and Max_Data=65535

Return values

- **None:**

Notes

- The counter clock frequency CK_CNT is equal to fCK_PSC / (PSC[15:0] + 1).
- The prescaler can be changed on the fly as this control register is buffered. The new prescaler ratio is taken into account at the next update event.
- Helper macro __LL_TIM_CALC_PSC can be used to calculate the Prescaler parameter

Reference Manual to LL API cross reference:

- PSC PSC LL_TIM_SetPrescaler

LL_TIM_GetPrescaler

Function name

__STATIC_INLINE uint32_t LL_TIM_GetPrescaler (const TIM_TypeDef * TIMx)

Function description

Get the prescaler value.

Parameters

- **TIMx:** Timer instance

Return values

- **Prescaler:** value between Min_Data=0 and Max_Data=65535

Reference Manual to LL API cross reference:

- PSC PSC LL_TIM_GetPrescaler

LL_TIM_SetAutoReload

Function name

__STATIC_INLINE void LL_TIM_SetAutoReload (TIM_TypeDef * TIMx, uint32_t AutoReload)

Function description

Set the auto-reload value.

Parameters

- **TIMx:** Timer instance
- **AutoReload:** between Min_Data=0 and Max_Data=65535

Return values

- **None:**

Notes

- The counter is blocked while the auto-reload value is null.
- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- Helper macro __LL_TIM_CALC_ARR can be used to calculate the AutoReload parameter In case dithering is activated,macro __LL_TIM_CALC_ARR_DITHER can be used instead, to calculate the AutoReload parameter.

Reference Manual to LL API cross reference:

- ARR ARR LL_TIM_SetAutoReload

LL_TIM_GetAutoReload

Function name

__STATIC_INLINE uint32_t LL_TIM_GetAutoReload (const TIM_TypeDef * TIMx)

Function description

Get the auto-reload value.

Parameters

- **TIMx:** Timer instance

Return values

- **Auto-reload:** value

Notes

- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- If dithering is activated, pay attention to the returned value interpretation

Reference Manual to LL API cross reference:

- ARR ARR LL_TIM_GetAutoReload

LL_TIM_SetRepetitionCounter

Function name

__STATIC_INLINE void LL_TIM_SetRepetitionCounter (TIM_TypeDef * TIMx, uint32_t RepetitionCounter)

Function description

Set the repetition counter value.

Parameters

- **TIMx:** Timer instance
- **RepetitionCounter:** between Min_Data=0 and Max_Data=255 or 65535 for advanced timer.

Return values

- **None:**

Notes

- For advanced timer instances RepetitionCounter can be up to 65535.
- Macro IS_TIM_REPETITION_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a repetition counter.

Reference Manual to LL API cross reference:

- RCR REP LL_TIM_SetRepetitionCounter

LL_TIM_GetRepetitionCounter

Function name

__STATIC_INLINE uint32_t LL_TIM_GetRepetitionCounter (const TIM_TypeDef * TIMx)

Function description

Get the repetition counter value.

Parameters

- **TIMx:** Timer instance

Return values

- **Repetition:** counter value

Notes

- Macro IS_TIM_REPETITION_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a repetition counter.

Reference Manual to LL API cross reference:

- RCR REP LL_TIM_GetRepetitionCounter

LL_TIM_EnableUIFRemap

Function name

__STATIC_INLINE void LL_TIM_EnableUIFRemap (TIM_TypeDef * TIMx)

Function description

Force a continuous copy of the update interrupt flag (UIF) into the timer counter register (bit 31).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- This allows both the counter value and a potential roll-over condition signalled by the UIFCPY flag to be read in an atomic way.

Reference Manual to LL API cross reference:

- CR1 UIFREMAP LL_TIM_EnableUIFRemap

LL_TIM_DisableUIFRemap

Function name

__STATIC_INLINE void LL_TIM_DisableUIFRemap (TIM_TypeDef * TIMx)

Function description

Disable update interrupt flag (UIF) remapping.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 UIFREMAP LL_TIM_DisableUIFRemap

LL_TIM_IsActiveUIFCPY

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveUIFCPY (const uint32_t Counter)

Function description

Indicate whether update interrupt flag (UIF) copy is set.

Parameters

- **Counter:** Counter value

Return values

- **State:** of bit (1 or 0).

LL_TIM_EnableDithering

Function name

__STATIC_INLINE void LL_TIM_EnableDithering (TIM_TypeDef * TIMx)

Function description

Enable dithering.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_DITHERING_INSTANCE(TIMx) can be used to check whether or not a timer instance provides dithering.

Reference Manual to LL API cross reference:

- CR1 DITHEN LL_TIM_EnableDithering

LL_TIM_DisableDithering

Function name

__STATIC_INLINE void LL_TIM_DisableDithering (TIM_TypeDef * TIMx)

Function description

Disable dithering.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_DITHERING_INSTANCE(TIMx) can be used to check whether or not a timer instance provides dithering.

Reference Manual to LL API cross reference:

- CR1 DITHEN LL_TIM_DisableDithering

LL_TIM_IsEnabledDithering

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledDithering (const TIM_TypeDef * TIMx)

Function description

Indicates whether dithering is activated.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_DITHERING_INSTANCE(TIMx) can be used to check whether or not a timer instance provides dithering.

Reference Manual to LL API cross reference:

- CR1 DITHEN LL_TIM_IsEnabledDithering

LL_TIM_CC_EnablePreload

Function name

__STATIC_INLINE void LL_TIM_CC_EnablePreload (TIM_TypeDef * TIMx)

Function description

Enable the capture/compare control bits (CCxE, CCxNE and OCxM) preload.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- CCxE, CCxNE and OCxM bits are preloaded, after having been written, they are updated only when a commutation event (COM) occurs.
- Only on channels that have a complementary output.
- Macro IS_TIM_COMMUTATION_EVENT_INSTANCE(TIMx) can be used to check whether or not a timer instance is able to generate a commutation event.

Reference Manual to LL API cross reference:

- CR2 CCPC LL_TIM_CC_EnablePreload

LL_TIM_CC_DisablePreload

Function name

__STATIC_INLINE void LL_TIM_CC_DisablePreload (TIM_TypeDef * TIMx)

Function description

Disable the capture/compare control bits (CCxE, CCxNE and OCxM) preload.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_COMMUTATION_EVENT_INSTANCE(TIMx) can be used to check whether or not a timer instance is able to generate a commutation event.

Reference Manual to LL API cross reference:

- CR2 CCPC LL_TIM_CC_DisablePreload

LL_TIM_CC_IsEnabledPreload

Function name

__STATIC_INLINE uint32_t LL_TIM_CC_IsEnabledPreload (const TIM_TypeDef * TIMx)

Function description

Indicates whether the capture/compare control bits (CCxE, CCxNE and OCxM) preload is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR2 CCPC LL_TIM_CC_IsEnabledPreload

LL_TIM_CC_SetUpdate

Function name

__STATIC_INLINE void LL_TIM_CC_SetUpdate (TIM_TypeDef * TIMx, uint32_t CCUpdateSource)

Function description

Set the updated source of the capture/compare control bits (CCxE, CCxNE and OCxM).

Parameters

- **TIMx:** Timer instance
- **CCUpdateSource:** This parameter can be one of the following values:
  - LL_TIM_CCUPDATESOURCE_COMG_ONLY
  - LL_TIM_CCUPDATESOURCE_COMG_AND_TRGI

Return values

- **None:**

Notes

- Macro IS_TIM_COMMUTATION_EVENT_INSTANCE(TIMx) can be used to check whether or not a timer instance is able to generate a commutation event.

Reference Manual to LL API cross reference:

- CR2 CCUS LL_TIM_CC_SetUpdate

LL_TIM_CC_SetDMAReqTrigger

Function name

__STATIC_INLINE void LL_TIM_CC_SetDMAReqTrigger (TIM_TypeDef * TIMx, uint32_t DMAReqTrigger)

Function description

Set the trigger of the capture/compare DMA request.

Parameters

- **TIMx:** Timer instance
- **DMAReqTrigger:** This parameter can be one of the following values:
  - LL_TIM_CCDMAREQUEST_CC
  - LL_TIM_CCDMAREQUEST_UPDATE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 CCDS LL_TIM_CC_SetDMAReqTrigger

LL_TIM_CC_GetDMAReqTrigger

Function name

__STATIC_INLINE uint32_t LL_TIM_CC_GetDMAReqTrigger (const TIM_TypeDef * TIMx)

Function description

Get actual trigger of the capture/compare DMA request.

Parameters

- **TIMx:** Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_TIM_CCDMAREQUEST_CC
  - LL_TIM_CCDMAREQUEST_UPDATE

Reference Manual to LL API cross reference:

- CR2 CCDS LL_TIM_CC_GetDMAReqTrigger

LL_TIM_CC_SetLockLevel

Function name

__STATIC_INLINE void LL_TIM_CC_SetLockLevel (TIM_TypeDef * TIMx, uint32_t LockLevel)

Function description

Set the lock level to freeze the configuration of several capture/compare parameters.

Parameters

- **TIMx:** Timer instance
- **LockLevel:** This parameter can be one of the following values:
  - LL_TIM_LOCKLEVEL_OFF
  - LL_TIM_LOCKLEVEL_1
  - LL_TIM_LOCKLEVEL_2
  - LL_TIM_LOCKLEVEL_3

Return values

- **None:**

Notes

- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not the lock mechanism is supported by a timer instance.

Reference Manual to LL API cross reference:

- BDTR LOCK LL_TIM_CC_SetLockLevel

LL_TIM_CC_EnableChannel

Function name

__STATIC_INLINE void LL_TIM_CC_EnableChannel (TIM_TypeDef * TIMx, uint32_t Channels)

Function description

Enable capture/compare channels.

Parameters

- **TIMx:** Timer instance
- **Channels:** This parameter can be a combination of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH1N
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH2N
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH3N
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH4N
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCER CC1E LL_TIM_CC_EnableChannel
- CCER CC1NE LL_TIM_CC_EnableChannel
- CCER CC2E LL_TIM_CC_EnableChannel
- CCER CC2NE LL_TIM_CC_EnableChannel
- CCER CC3E LL_TIM_CC_EnableChannel
- CCER CC3NE LL_TIM_CC_EnableChannel
- CCER CC4E LL_TIM_CC_EnableChannel
- CCER CC4NE LL_TIM_CC_EnableChannel
- CCER CC5E LL_TIM_CC_EnableChannel
- CCER CC6E LL_TIM_CC_EnableChannel

LL_TIM_CC_DisableChannel

Function name

__STATIC_INLINE void LL_TIM_CC_DisableChannel (TIM_TypeDef * TIMx, uint32_t Channels)

Function description

Disable capture/compare channels.

Parameters

- **TIMx:** Timer instance
- **Channels:** This parameter can be a combination of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH1N
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH2N
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH3N
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH4N
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCER CC1E LL_TIM_CC_DisableChannel
- CCER CC1NE LL_TIM_CC_DisableChannel
- CCER CC2E LL_TIM_CC_DisableChannel
- CCER CC2NE LL_TIM_CC_DisableChannel
- CCER CC3E LL_TIM_CC_DisableChannel
- CCER CC3NE LL_TIM_CC_DisableChannel
- CCER CC4E LL_TIM_CC_DisableChannel
- CCER CC4NE LL_TIM_CC_DisableChannel
- CCER CC5E LL_TIM_CC_DisableChannel
- CCER CC6E LL_TIM_CC_DisableChannel

LL_TIM_CC_IsEnabledChannel

Function name

__STATIC_INLINE uint32_t LL_TIM_CC_IsEnabledChannel (const TIM_TypeDef * TIMx, uint32_t Channels)

Function description

Indicate whether channel(s) is(are) enabled.

Parameters

- **TIMx:** Timer instance
- **Channels:** This parameter can be a combination of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH1N
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH2N
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH3N
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH4N
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CCER CC1E LL_TIM_CC_IsEnabledChannel
- CCER CC1NE LL_TIM_CC_IsEnabledChannel
- CCER CC2E LL_TIM_CC_IsEnabledChannel
- CCER CC2NE LL_TIM_CC_IsEnabledChannel
- CCER CC3E LL_TIM_CC_IsEnabledChannel
- CCER CC3NE LL_TIM_CC_IsEnabledChannel
- CCER CC4E LL_TIM_CC_IsEnabledChannel
- CCER CC4NE LL_TIM_CC_IsEnabledChannel
- CCER CC5E LL_TIM_CC_IsEnabledChannel
- CCER CC6E LL_TIM_CC_IsEnabledChannel

LL_TIM_OC_ConfigOutput

Function name

__STATIC_INLINE void LL_TIM_OC_ConfigOutput (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t Configuration)

Function description

Configure an output channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6
- **Configuration:** This parameter must be a combination of all the following values:
  - LL_TIM_OCPOLARITY_HIGH or LL_TIM_OCPOLARITY_LOW
  - LL_TIM_OCIDLESTATE_LOW or LL_TIM_OCIDLESTATE_HIGH

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCMR1 CC1S LL_TIM_OC_ConfigOutput
- CCMR1 CC2S LL_TIM_OC_ConfigOutput
- CCMR2 CC3S LL_TIM_OC_ConfigOutput
- CCMR2 CC4S LL_TIM_OC_ConfigOutput
- CCMR3 CC5S LL_TIM_OC_ConfigOutput
- CCMR3 CC6S LL_TIM_OC_ConfigOutput
- CCER CC1P LL_TIM_OC_ConfigOutput
- CCER CC2P LL_TIM_OC_ConfigOutput
- CCER CC3P LL_TIM_OC_ConfigOutput
- CCER CC4P LL_TIM_OC_ConfigOutput
- CCER CC5P LL_TIM_OC_ConfigOutput
- CCER CC6P LL_TIM_OC_ConfigOutput
- CR2 OIS1 LL_TIM_OC_ConfigOutput
- CR2 OIS2 LL_TIM_OC_ConfigOutput
- CR2 OIS3 LL_TIM_OC_ConfigOutput
- CR2 OIS4 LL_TIM_OC_ConfigOutput
- CR2 OIS5 LL_TIM_OC_ConfigOutput
- CR2 OIS6 LL_TIM_OC_ConfigOutput

LL_TIM_OC_SetMode

Function name

__STATIC_INLINE void LL_TIM_OC_SetMode (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t Mode)

Function description

Define the behavior of the output reference signal OCxREF from which OCx and OCxN (when relevant) are derived.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6
- **Mode:** This parameter can be one of the following values:
  - LL_TIM_OCMODE_FROZEN
  - LL_TIM_OCMODE_ACTIVE
  - LL_TIM_OCMODE_INACTIVE
  - LL_TIM_OCMODE_TOGGLE
  - LL_TIM_OCMODE_FORCED_INACTIVE
  - LL_TIM_OCMODE_FORCED_ACTIVE
  - LL_TIM_OCMODE_PWM1
  - LL_TIM_OCMODE_PWM2
  - LL_TIM_OCMODE_RETRIG_OPM1
  - LL_TIM_OCMODE_RETRIG_OPM2
  - LL_TIM_OCMODE_COMBINED_PWM1
  - LL_TIM_OCMODE_COMBINED_PWM2
  - LL_TIM_OCMODE_ASYMMETRIC_PWM1
  - LL_TIM_OCMODE_ASYMMETRIC_PWM2
  - LL_TIM_OCMODE_PULSE_ON_COMPARE (for channel 3 or channel 4 only)
  - LL_TIM_OCMODE_DIRECTION_OUTPUT (for channel 3 or channel 4 only)

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCMR1 OC1M LL_TIM_OC_SetMode
- CCMR1 OC2M LL_TIM_OC_SetMode
- CCMR2 OC3M LL_TIM_OC_SetMode
- CCMR2 OC4M LL_TIM_OC_SetMode
- CCMR3 OC5M LL_TIM_OC_SetMode
- CCMR3 OC6M LL_TIM_OC_SetMode

LL_TIM_OC_GetMode

Function name

__STATIC_INLINE uint32_t LL_TIM_OC_GetMode (const TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Get the output compare mode of an output channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6

Return values

- **Returned:** value can be one of the following values:
  - LL_TIM_OCMODE_FROZEN
  - LL_TIM_OCMODE_ACTIVE
  - LL_TIM_OCMODE_INACTIVE
  - LL_TIM_OCMODE_TOGGLE
  - LL_TIM_OCMODE_FORCED_INACTIVE
  - LL_TIM_OCMODE_FORCED_ACTIVE
  - LL_TIM_OCMODE_PWM1
  - LL_TIM_OCMODE_PWM2
  - LL_TIM_OCMODE_RETRIG_OPM1
  - LL_TIM_OCMODE_RETRIG_OPM2
  - LL_TIM_OCMODE_COMBINED_PWM1
  - LL_TIM_OCMODE_COMBINED_PWM2
  - LL_TIM_OCMODE_ASYMMETRIC_PWM1
  - LL_TIM_OCMODE_ASYMMETRIC_PWM2
  - LL_TIM_OCMODE_PULSE_ON_COMPARE (for channel 3 or channel 4 only)
  - LL_TIM_OCMODE_DIRECTION_OUTPUT (for channel 3 or channel 4 only)

Reference Manual to LL API cross reference:

- CCMR1 OC1M LL_TIM_OC_GetMode
- CCMR1 OC2M LL_TIM_OC_GetMode
- CCMR2 OC3M LL_TIM_OC_GetMode
- CCMR2 OC4M LL_TIM_OC_GetMode
- CCMR3 OC5M LL_TIM_OC_GetMode
- CCMR3 OC6M LL_TIM_OC_GetMode

LL_TIM_OC_SetPolarity

Function name

__STATIC_INLINE void LL_TIM_OC_SetPolarity (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t Polarity)

Function description

Set the polarity of an output channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH1N
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH2N
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH3N
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH4N
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6
- **Polarity:** This parameter can be one of the following values:
  - LL_TIM_OCPOLARITY_HIGH
  - LL_TIM_OCPOLARITY_LOW

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCER CC1P LL_TIM_OC_SetPolarity
- CCER CC1NP LL_TIM_OC_SetPolarity
- CCER CC2P LL_TIM_OC_SetPolarity
- CCER CC2NP LL_TIM_OC_SetPolarity
- CCER CC3P LL_TIM_OC_SetPolarity
- CCER CC3NP LL_TIM_OC_SetPolarity
- CCER CC4P LL_TIM_OC_SetPolarity
- CCER CC4NP LL_TIM_OC_SetPolarity
- CCER CC5P LL_TIM_OC_SetPolarity
- CCER CC6P LL_TIM_OC_SetPolarity

LL_TIM_OC_GetPolarity

Function name

__STATIC_INLINE uint32_t LL_TIM_OC_GetPolarity (const TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Get the polarity of an output channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH1N
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH2N
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH3N
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH4N
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6

Return values

- **Returned:** value can be one of the following values:
  - LL_TIM_OCPOLARITY_HIGH
  - LL_TIM_OCPOLARITY_LOW

Reference Manual to LL API cross reference:

- CCER CC1P LL_TIM_OC_GetPolarity
- CCER CC1NP LL_TIM_OC_GetPolarity
- CCER CC2P LL_TIM_OC_GetPolarity
- CCER CC2NP LL_TIM_OC_GetPolarity
- CCER CC3P LL_TIM_OC_GetPolarity
- CCER CC3NP LL_TIM_OC_GetPolarity
- CCER CC4P LL_TIM_OC_GetPolarity
- CCER CC4NP LL_TIM_OC_GetPolarity
- CCER CC5P LL_TIM_OC_GetPolarity
- CCER CC6P LL_TIM_OC_GetPolarity

LL_TIM_OC_SetIdleState

Function name

__STATIC_INLINE void LL_TIM_OC_SetIdleState (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t IdleState)

Function description

Set the IDLE state of an output channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH1N
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH2N
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH3N
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH4N
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6
- **IdleState:** This parameter can be one of the following values:
  - LL_TIM_OCIDLESTATE_LOW
  - LL_TIM_OCIDLESTATE_HIGH

Return values

- **None:**

Notes

- This function is significant only for the timer instances supporting the break feature. Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.

Reference Manual to LL API cross reference:

- CR2 OIS1 LL_TIM_OC_SetIdleState
- CR2 OIS2N LL_TIM_OC_SetIdleState
- CR2 OIS2 LL_TIM_OC_SetIdleState
- CR2 OIS2N LL_TIM_OC_SetIdleState
- CR2 OIS3 LL_TIM_OC_SetIdleState
- CR2 OIS3N LL_TIM_OC_SetIdleState
- CR2 OIS4 LL_TIM_OC_SetIdleState
- CR2 OIS4N LL_TIM_OC_SetIdleState
- CR2 OIS5 LL_TIM_OC_SetIdleState
- CR2 OIS6 LL_TIM_OC_SetIdleState

LL_TIM_OC_GetIdleState

Function name

__STATIC_INLINE uint32_t LL_TIM_OC_GetIdleState (const TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Get the IDLE state of an output channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH1N
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH2N
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH3N
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH4N
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6

Return values

- **Returned:** value can be one of the following values:
  - LL_TIM_OCIDLESTATE_LOW
  - LL_TIM_OCIDLESTATE_HIGH

Reference Manual to LL API cross reference:

- CR2 OIS1 LL_TIM_OC_GetIdleState
- CR2 OIS2N LL_TIM_OC_GetIdleState
- CR2 OIS2 LL_TIM_OC_GetIdleState
- CR2 OIS2N LL_TIM_OC_GetIdleState
- CR2 OIS3 LL_TIM_OC_GetIdleState
- CR2 OIS3N LL_TIM_OC_GetIdleState
- CR2 OIS4 LL_TIM_OC_GetIdleState
- CR2 OIS4N LL_TIM_OC_GetIdleState
- CR2 OIS5 LL_TIM_OC_GetIdleState
- CR2 OIS6 LL_TIM_OC_GetIdleState

LL_TIM_OC_EnableFast

Function name

__STATIC_INLINE void LL_TIM_OC_EnableFast (TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Enable fast mode for the output channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6

Return values

- **None:**

Notes

- Acts only if the channel is configured in PWM1 or PWM2 mode.

Reference Manual to LL API cross reference:

- CCMR1 OC1FE LL_TIM_OC_EnableFast
- CCMR1 OC2FE LL_TIM_OC_EnableFast
- CCMR2 OC3FE LL_TIM_OC_EnableFast
- CCMR2 OC4FE LL_TIM_OC_EnableFast
- CCMR3 OC5FE LL_TIM_OC_EnableFast
- CCMR3 OC6FE LL_TIM_OC_EnableFast

LL_TIM_OC_DisableFast

Function name

__STATIC_INLINE void LL_TIM_OC_DisableFast (TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Disable fast mode for the output channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCMR1 OC1FE LL_TIM_OC_DisableFast
- CCMR1 OC2FE LL_TIM_OC_DisableFast
- CCMR2 OC3FE LL_TIM_OC_DisableFast
- CCMR2 OC4FE LL_TIM_OC_DisableFast
- CCMR3 OC5FE LL_TIM_OC_DisableFast
- CCMR3 OC6FE LL_TIM_OC_DisableFast

LL_TIM_OC_IsEnabledFast

Function name

__STATIC_INLINE uint32_t LL_TIM_OC_IsEnabledFast (const TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Indicates whether fast mode is enabled for the output channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CCMR1 OC1FE LL_TIM_OC_IsEnabledFast
- CCMR1 OC2FE LL_TIM_OC_IsEnabledFast
- CCMR2 OC3FE LL_TIM_OC_IsEnabledFast
- CCMR2 OC4FE LL_TIM_OC_IsEnabledFast
- CCMR3 OC5FE LL_TIM_OC_IsEnabledFast
- CCMR3 OC6FE LL_TIM_OC_IsEnabledFast

LL_TIM_OC_EnablePreload

Function name

__STATIC_INLINE void LL_TIM_OC_EnablePreload (TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Enable compare register (TIMx_CCRx) preload for the output channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCMR1 OC1PE LL_TIM_OC_EnablePreload
- CCMR1 OC2PE LL_TIM_OC_EnablePreload
- CCMR2 OC3PE LL_TIM_OC_EnablePreload
- CCMR2 OC4PE LL_TIM_OC_EnablePreload
- CCMR3 OC5PE LL_TIM_OC_EnablePreload
- CCMR3 OC6PE LL_TIM_OC_EnablePreload

LL_TIM_OC_DisablePreload

Function name

__STATIC_INLINE void LL_TIM_OC_DisablePreload (TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Disable compare register (TIMx_CCRx) preload for the output channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCMR1 OC1PE LL_TIM_OC_DisablePreload
- CCMR1 OC2PE LL_TIM_OC_DisablePreload
- CCMR2 OC3PE LL_TIM_OC_DisablePreload
- CCMR2 OC4PE LL_TIM_OC_DisablePreload
- CCMR3 OC5PE LL_TIM_OC_DisablePreload
- CCMR3 OC6PE LL_TIM_OC_DisablePreload

LL_TIM_OC_IsEnabledPreload

Function name

__STATIC_INLINE uint32_t LL_TIM_OC_IsEnabledPreload (const TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Indicates whether compare register (TIMx_CCRx) preload is enabled for the output channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CCMR1 OC1PE LL_TIM_OC_IsEnabledPreload
- CCMR1 OC2PE LL_TIM_OC_IsEnabledPreload
- CCMR2 OC3PE LL_TIM_OC_IsEnabledPreload
- CCMR2 OC4PE LL_TIM_OC_IsEnabledPreload
- CCMR3 OC5PE LL_TIM_OC_IsEnabledPreload
- CCMR3 OC6PE LL_TIM_OC_IsEnabledPreload

LL_TIM_OC_EnableClear

Function name

__STATIC_INLINE void LL_TIM_OC_EnableClear (TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Enable clearing the output channel on an external event.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6

Return values

- **None:**

Notes

- This function can only be used in Output compare and PWM modes. It does not work in Forced mode.
- Macro IS_TIM_OCXREF_CLEAR_INSTANCE(TIMx) can be used to check whether or not a timer instance can clear the OCxREF signal on an external event.

Reference Manual to LL API cross reference:

- CCMR1 OC1CE LL_TIM_OC_EnableClear
- CCMR1 OC2CE LL_TIM_OC_EnableClear
- CCMR2 OC3CE LL_TIM_OC_EnableClear
- CCMR2 OC4CE LL_TIM_OC_EnableClear
- CCMR3 OC5CE LL_TIM_OC_EnableClear
- CCMR3 OC6CE LL_TIM_OC_EnableClear

LL_TIM_OC_DisableClear

Function name

__STATIC_INLINE void LL_TIM_OC_DisableClear (TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Disable clearing the output channel on an external event.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6

Return values

- **None:**

Notes

- Macro IS_TIM_OCXREF_CLEAR_INSTANCE(TIMx) can be used to check whether or not a timer instance can clear the OCxREF signal on an external event.

Reference Manual to LL API cross reference:

- CCMR1 OC1CE LL_TIM_OC_DisableClear
- CCMR1 OC2CE LL_TIM_OC_DisableClear
- CCMR2 OC3CE LL_TIM_OC_DisableClear
- CCMR2 OC4CE LL_TIM_OC_DisableClear
- CCMR3 OC5CE LL_TIM_OC_DisableClear
- CCMR3 OC6CE LL_TIM_OC_DisableClear

LL_TIM_OC_IsEnabledClear

Function name

__STATIC_INLINE uint32_t LL_TIM_OC_IsEnabledClear (const TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Indicates clearing the output channel on an external event is enabled for the output channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6

Return values

- **State:** of bit (1 or 0).

Notes

- This function enables clearing the output channel on an external event.
- This function can only be used in Output compare and PWM modes. It does not work in Forced mode.
- Macro IS_TIM_OCXREF_CLEAR_INSTANCE(TIMx) can be used to check whether or not a timer instance can clear the OCxREF signal on an external event.

Reference Manual to LL API cross reference:

- CCMR1 OC1CE LL_TIM_OC_IsEnabledClear
- CCMR1 OC2CE LL_TIM_OC_IsEnabledClear
- CCMR2 OC3CE LL_TIM_OC_IsEnabledClear
- CCMR2 OC4CE LL_TIM_OC_IsEnabledClear
- CCMR3 OC5CE LL_TIM_OC_IsEnabledClear
- CCMR3 OC6CE LL_TIM_OC_IsEnabledClear

LL_TIM_OC_SetDeadTime

Function name

__STATIC_INLINE void LL_TIM_OC_SetDeadTime (TIM_TypeDef * TIMx, uint32_t DeadTime)

Function description

Set the dead-time delay (delay inserted between the rising edge of the OCxREF signal and the rising edge of the Ocx and OCxN signals).

Parameters

- **TIMx:** Timer instance
- **DeadTime:** between Min_Data=0 and Max_Data=255

Return values

- **None:**

Notes

- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not dead-time insertion feature is supported by a timer instance.
- Helper macro __LL_TIM_CALC_DEADTIME can be used to calculate the DeadTime parameter

Reference Manual to LL API cross reference:

- BDTR DTG LL_TIM_OC_SetDeadTime

LL_TIM_OC_SetCompareCH1

Function name

__STATIC_INLINE void LL_TIM_OC_SetCompareCH1 (TIM_TypeDef * TIMx, uint32_t CompareValue)

Function description

Set compare value for output channel 1 (TIMx_CCR1).

Parameters

- **TIMx:** Timer instance
- **CompareValue:** between Min_Data=0 and Max_Data=65535

Return values

- **None:**

Notes

- In 32-bit timer implementations compare value can be between 0x00000000 and 0xFFFFFFFF.
- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- Macro IS_TIM_CC1_INSTANCE(TIMx) can be used to check whether or not output channel 1 is supported by a timer instance.
- If dithering is activated, CompareValue can be calculated with macro __LL_TIM_CALC_DELAY_DITHER .

Reference Manual to LL API cross reference:

- CCR1 CCR1 LL_TIM_OC_SetCompareCH1

LL_TIM_OC_SetCompareCH2

Function name

__STATIC_INLINE void LL_TIM_OC_SetCompareCH2 (TIM_TypeDef * TIMx, uint32_t CompareValue)

Function description

Set compare value for output channel 2 (TIMx_CCR2).

Parameters

- **TIMx:** Timer instance
- **CompareValue:** between Min_Data=0 and Max_Data=65535

Return values

- **None:**

Notes

- In 32-bit timer implementations compare value can be between 0x00000000 and 0xFFFFFFFF.
- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- Macro IS_TIM_CC2_INSTANCE(TIMx) can be used to check whether or not output channel 2 is supported by a timer instance.
- If dithering is activated, CompareValue can be calculated with macro __LL_TIM_CALC_DELAY_DITHER .

Reference Manual to LL API cross reference:

- CCR2 CCR2 LL_TIM_OC_SetCompareCH2

LL_TIM_OC_SetCompareCH3

Function name

__STATIC_INLINE void LL_TIM_OC_SetCompareCH3 (TIM_TypeDef * TIMx, uint32_t CompareValue)

Function description

Set compare value for output channel 3 (TIMx_CCR3).

Parameters

- **TIMx:** Timer instance
- **CompareValue:** between Min_Data=0 and Max_Data=65535

Return values

- **None:**

Notes

- In 32-bit timer implementations compare value can be between 0x00000000 and 0xFFFFFFFF.
- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- Macro IS_TIM_CC3_INSTANCE(TIMx) can be used to check whether or not output channel is supported by a timer instance.
- If dithering is activated, CompareValue can be calculated with macro __LL_TIM_CALC_DELAY_DITHER .

Reference Manual to LL API cross reference:

- CCR3 CCR3 LL_TIM_OC_SetCompareCH3

LL_TIM_OC_SetCompareCH4

Function name

__STATIC_INLINE void LL_TIM_OC_SetCompareCH4 (TIM_TypeDef * TIMx, uint32_t CompareValue)

Function description

Set compare value for output channel 4 (TIMx_CCR4).

Parameters

- **TIMx:** Timer instance
- **CompareValue:** between Min_Data=0 and Max_Data=65535

Return values

- **None:**

Notes

- In 32-bit timer implementations compare value can be between 0x00000000 and 0xFFFFFFFF.
- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- Macro IS_TIM_CC4_INSTANCE(TIMx) can be used to check whether or not output channel 4 is supported by a timer instance.
- If dithering is activated, CompareValue can be calculated with macro __LL_TIM_CALC_DELAY_DITHER .

Reference Manual to LL API cross reference:

- CCR4 CCR4 LL_TIM_OC_SetCompareCH4

LL_TIM_OC_SetCompareCH5

Function name

__STATIC_INLINE void LL_TIM_OC_SetCompareCH5 (TIM_TypeDef * TIMx, uint32_t CompareValue)

Function description

Set compare value for output channel 5 (TIMx_CCR5).

Parameters

- **TIMx:** Timer instance
- **CompareValue:** between Min_Data=0 and Max_Data=65535

Return values

- **None:**

Notes

- Macro IS_TIM_CC5_INSTANCE(TIMx) can be used to check whether or not output channel 5 is supported by a timer instance.
- If dithering is activated, CompareValue can be calculated with macro __LL_TIM_CALC_DELAY_DITHER .

Reference Manual to LL API cross reference:

- CCR5 CCR5 LL_TIM_OC_SetCompareCH5

LL_TIM_OC_SetCompareCH6

Function name

__STATIC_INLINE void LL_TIM_OC_SetCompareCH6 (TIM_TypeDef * TIMx, uint32_t CompareValue)

Function description

Set compare value for output channel 6 (TIMx_CCR6).

Parameters

- **TIMx:** Timer instance
- **CompareValue:** between Min_Data=0 and Max_Data=65535

Return values

- **None:**

Notes

- Macro IS_TIM_CC6_INSTANCE(TIMx) can be used to check whether or not output channel 6 is supported by a timer instance.
- If dithering is activated, CompareValue can be calculated with macro __LL_TIM_CALC_DELAY_DITHER .

Reference Manual to LL API cross reference:

- CCR6 CCR6 LL_TIM_OC_SetCompareCH6

LL_TIM_OC_GetCompareCH1

Function name

__STATIC_INLINE uint32_t LL_TIM_OC_GetCompareCH1 (const TIM_TypeDef * TIMx)

Function description

Get compare value (TIMx_CCR1) set for output channel 1.

Parameters

- **TIMx:** Timer instance

Return values

- **CompareValue:** (between Min_Data=0 and Max_Data=65535)

Notes

- In 32-bit timer implementations returned compare value can be between 0x00000000 and 0xFFFFFFFF.
- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- Macro IS_TIM_CC1_INSTANCE(TIMx) can be used to check whether or not output channel 1 is supported by a timer instance.
- If dithering is activated, pay attention to the returned value interpretation.

Reference Manual to LL API cross reference:

- CCR1 CCR1 LL_TIM_OC_GetCompareCH1

LL_TIM_OC_GetCompareCH2

Function name

__STATIC_INLINE uint32_t LL_TIM_OC_GetCompareCH2 (const TIM_TypeDef * TIMx)

Function description

Get compare value (TIMx_CCR2) set for output channel 2.

Parameters

- **TIMx:** Timer instance

Return values

- **CompareValue:** (between Min_Data=0 and Max_Data=65535)

Notes

- In 32-bit timer implementations returned compare value can be between 0x00000000 and 0xFFFFFFFF.
- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- Macro IS_TIM_CC2_INSTANCE(TIMx) can be used to check whether or not output channel 2 is supported by a timer instance.
- If dithering is activated, pay attention to the returned value interpretation.

Reference Manual to LL API cross reference:

- CCR2 CCR2 LL_TIM_OC_GetCompareCH2

LL_TIM_OC_GetCompareCH3

Function name

__STATIC_INLINE uint32_t LL_TIM_OC_GetCompareCH3 (const TIM_TypeDef * TIMx)

Function description

Get compare value (TIMx_CCR3) set for output channel 3.

Parameters

- **TIMx:** Timer instance

Return values

- **CompareValue:** (between Min_Data=0 and Max_Data=65535)

Notes

- In 32-bit timer implementations returned compare value can be between 0x00000000 and 0xFFFFFFFF.
- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- Macro IS_TIM_CC3_INSTANCE(TIMx) can be used to check whether or not output channel 3 is supported by a timer instance.
- If dithering is activated, pay attention to the returned value interpretation.

Reference Manual to LL API cross reference:

- CCR3 CCR3 LL_TIM_OC_GetCompareCH3

LL_TIM_OC_GetCompareCH4

Function name

__STATIC_INLINE uint32_t LL_TIM_OC_GetCompareCH4 (const TIM_TypeDef * TIMx)

Function description

Get compare value (TIMx_CCR4) set for output channel 4.

Parameters

- **TIMx:** Timer instance

Return values

- **CompareValue:** (between Min_Data=0 and Max_Data=65535)

Notes

- In 32-bit timer implementations returned compare value can be between 0x00000000 and 0xFFFFFFFF.
- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- Macro IS_TIM_CC4_INSTANCE(TIMx) can be used to check whether or not output channel 4 is supported by a timer instance.
- If dithering is activated, pay attention to the returned value interpretation.

Reference Manual to LL API cross reference:

- CCR4 CCR4 LL_TIM_OC_GetCompareCH4

LL_TIM_OC_GetCompareCH5

Function name

__STATIC_INLINE uint32_t LL_TIM_OC_GetCompareCH5 (const TIM_TypeDef * TIMx)

Function description

Get compare value (TIMx_CCR5) set for output channel 5.

Parameters

- **TIMx:** Timer instance

Return values

- **CompareValue:** (between Min_Data=0 and Max_Data=65535)

Notes

- Macro IS_TIM_CC5_INSTANCE(TIMx) can be used to check whether or not output channel 5 is supported by a timer instance.
- If dithering is activated, pay attention to the returned value interpretation.

Reference Manual to LL API cross reference:

- CCR5 CCR5 LL_TIM_OC_GetCompareCH5

LL_TIM_OC_GetCompareCH6

Function name

__STATIC_INLINE uint32_t LL_TIM_OC_GetCompareCH6 (const TIM_TypeDef * TIMx)

Function description

Get compare value (TIMx_CCR6) set for output channel 6.

Parameters

- **TIMx:** Timer instance

Return values

- **CompareValue:** (between Min_Data=0 and Max_Data=65535)

Notes

- Macro IS_TIM_CC6_INSTANCE(TIMx) can be used to check whether or not output channel 6 is supported by a timer instance.
- If dithering is activated, pay attention to the returned value interpretation.

Reference Manual to LL API cross reference:

- CCR6 CCR6 LL_TIM_OC_GetCompareCH6

LL_TIM_SetCH5CombinedChannels

Function name

__STATIC_INLINE void LL_TIM_SetCH5CombinedChannels (TIM_TypeDef * TIMx, uint32_t GroupCH5)

Function description

Select on which reference signal the OC5REF is combined to.

Parameters

- **TIMx:** Timer instance
- **GroupCH5:** This parameter can be a combination of the following values:
  - LL_TIM_GROUPCH5_NONE
  - LL_TIM_GROUPCH5_OC1REFC
  - LL_TIM_GROUPCH5_OC2REFC
  - LL_TIM_GROUPCH5_OC3REFC

Return values

- **None:**

Notes

- Macro IS_TIM_COMBINED3PHASEPWM_INSTANCE(TIMx) can be used to check whether or not a timer instance supports the combined 3-phase PWM mode.

Reference Manual to LL API cross reference:

- CCR5 GC5C3 LL_TIM_SetCH5CombinedChannels
- CCR5 GC5C2 LL_TIM_SetCH5CombinedChannels
- CCR5 GC5C1 LL_TIM_SetCH5CombinedChannels

LL_TIM_OC_SetPulseWidthPrescaler

Function name

__STATIC_INLINE void LL_TIM_OC_SetPulseWidthPrescaler (TIM_TypeDef * TIMx, uint32_t PulseWidthPrescaler)

Function description

Set the pulse on compare pulse width prescaler.

Parameters

- **TIMx:** Timer instance
- **PulseWidthPrescaler:** This parameter can be one of the following values:
  - LL_TIM_PWPRSC_X1
  - LL_TIM_PWPRSC_X2
  - LL_TIM_PWPRSC_X4
  - LL_TIM_PWPRSC_X8
  - LL_TIM_PWPRSC_X16
  - LL_TIM_PWPRSC_X32
  - LL_TIM_PWPRSC_X64
  - LL_TIM_PWPRSC_X128

Return values

- **None:**

Notes

- Macro IS_TIM_PULSEONCOMPARE_INSTANCE(TIMx) can be used to check whether or not the pulse on compare feature is supported by the timer instance.

Reference Manual to LL API cross reference:

- ECR PWPRSC LL_TIM_OC_SetPulseWidthPrescaler

LL_TIM_OC_GetPulseWidthPrescaler

Function name

__STATIC_INLINE uint32_t LL_TIM_OC_GetPulseWidthPrescaler (const TIM_TypeDef * TIMx)

Function description

Get the pulse on compare pulse width prescaler.

Parameters

- **TIMx:** Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_TIM_PWPRSC_X1
  - LL_TIM_PWPRSC_X2
  - LL_TIM_PWPRSC_X4
  - LL_TIM_PWPRSC_X8
  - LL_TIM_PWPRSC_X16
  - LL_TIM_PWPRSC_X32
  - LL_TIM_PWPRSC_X64
  - LL_TIM_PWPRSC_X128

Notes

- Macro IS_TIM_PULSEONCOMPARE_INSTANCE(TIMx) can be used to check whether or not the pulse on compare feature is supported by the timer instance.

Reference Manual to LL API cross reference:

- ECR PWPRSC LL_TIM_OC_GetPulseWidthPrescaler

LL_TIM_OC_SetPulseWidth

Function name

__STATIC_INLINE void LL_TIM_OC_SetPulseWidth (TIM_TypeDef * TIMx, uint32_t PulseWidth)

Function description

Set the pulse on compare pulse width duration.

Parameters

- **TIMx:** Timer instance
- **PulseWidth:** This parameter can be between Min_Data=0 and Max_Data=255

Return values

- **None:**

Notes

- Macro IS_TIM_PULSEONCOMPARE_INSTANCE(TIMx) can be used to check whether or not the pulse on compare feature is supported by the timer instance.

Reference Manual to LL API cross reference:

- ECR PW LL_TIM_OC_SetPulseWidth

LL_TIM_OC_GetPulseWidth

Function name

__STATIC_INLINE uint32_t LL_TIM_OC_GetPulseWidth (const TIM_TypeDef * TIMx)

Function description

Get the pulse on compare pulse width duration.

Parameters

- **TIMx:** Timer instance

Return values

- **Returned:** value can be between Min_Data=0 and Max_Data=255:

Notes

- Macro IS_TIM_PULSEONCOMPARE_INSTANCE(TIMx) can be used to check whether or not the pulse on compare feature is supported by the timer instance.

Reference Manual to LL API cross reference:

- ECR PW LL_TIM_OC_GetPulseWidth

LL_TIM_IC_Config

Function name

__STATIC_INLINE void LL_TIM_IC_Config (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t Configuration)

Function description

Configure input channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
- **Configuration:** This parameter must be a combination of all the following values:
  - LL_TIM_ACTIVEINPUT_DIRECTTI or LL_TIM_ACTIVEINPUT_INDIRECTTI or LL_TIM_ACTIVEINPUT_TRC
  - LL_TIM_ICPSC_DIV1 or ... or LL_TIM_ICPSC_DIV8
  - LL_TIM_IC_FILTER_FDIV1 or ... or LL_TIM_IC_FILTER_FDIV32_N8
  - LL_TIM_IC_POLARITY_RISING or LL_TIM_IC_POLARITY_FALLING or LL_TIM_IC_POLARITY_BOTHEDGE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCMR1 CC1S LL_TIM_IC_Config
- CCMR1 IC1PSC LL_TIM_IC_Config
- CCMR1 IC1F LL_TIM_IC_Config
- CCMR1 CC2S LL_TIM_IC_Config
- CCMR1 IC2PSC LL_TIM_IC_Config
- CCMR1 IC2F LL_TIM_IC_Config
- CCMR2 CC3S LL_TIM_IC_Config
- CCMR2 IC3PSC LL_TIM_IC_Config
- CCMR2 IC3F LL_TIM_IC_Config
- CCMR2 CC4S LL_TIM_IC_Config
- CCMR2 IC4PSC LL_TIM_IC_Config
- CCMR2 IC4F LL_TIM_IC_Config
- CCER CC1P LL_TIM_IC_Config
- CCER CC1NP LL_TIM_IC_Config
- CCER CC2P LL_TIM_IC_Config
- CCER CC2NP LL_TIM_IC_Config
- CCER CC3P LL_TIM_IC_Config
- CCER CC3NP LL_TIM_IC_Config
- CCER CC4P LL_TIM_IC_Config
- CCER CC4NP LL_TIM_IC_Config

LL_TIM_IC_SetActiveInput

Function name

__STATIC_INLINE void LL_TIM_IC_SetActiveInput (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t ICActiveInput)

Function description

Set the active input.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
- **ICActiveInput:** This parameter can be one of the following values:
  - LL_TIM_ACTIVEINPUT_DIRECTTI
  - LL_TIM_ACTIVEINPUT_INDIRECTTI
  - LL_TIM_ACTIVEINPUT_TRC

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCMR1 CC1S LL_TIM_IC_SetActiveInput
- CCMR1 CC2S LL_TIM_IC_SetActiveInput
- CCMR2 CC3S LL_TIM_IC_SetActiveInput
- CCMR2 CC4S LL_TIM_IC_SetActiveInput

LL_TIM_IC_GetActiveInput

Function name

__STATIC_INLINE uint32_t LL_TIM_IC_GetActiveInput (const TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Get the current active input.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4

Return values

- **Returned:** value can be one of the following values:
  - LL_TIM_ACTIVEINPUT_DIRECTTI
  - LL_TIM_ACTIVEINPUT_INDIRECTTI
  - LL_TIM_ACTIVEINPUT_TRC

Reference Manual to LL API cross reference:

- CCMR1 CC1S LL_TIM_IC_GetActiveInput
- CCMR1 CC2S LL_TIM_IC_GetActiveInput
- CCMR2 CC3S LL_TIM_IC_GetActiveInput
- CCMR2 CC4S LL_TIM_IC_GetActiveInput

LL_TIM_IC_SetPrescaler

Function name

__STATIC_INLINE void LL_TIM_IC_SetPrescaler (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t ICPrescaler)

Function description

Set the prescaler of input channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
- **ICPrescaler:** This parameter can be one of the following values:
  - LL_TIM_ICPSC_DIV1
  - LL_TIM_ICPSC_DIV2
  - LL_TIM_ICPSC_DIV4
  - LL_TIM_ICPSC_DIV8

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCMR1 IC1PSC LL_TIM_IC_SetPrescaler
- CCMR1 IC2PSC LL_TIM_IC_SetPrescaler
- CCMR2 IC3PSC LL_TIM_IC_SetPrescaler
- CCMR2 IC4PSC LL_TIM_IC_SetPrescaler

LL_TIM_IC_GetPrescaler

Function name

__STATIC_INLINE uint32_t LL_TIM_IC_GetPrescaler (const TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Get the current prescaler value acting on an input channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4

Return values

- **Returned:** value can be one of the following values:
  - LL_TIM_ICPSC_DIV1
  - LL_TIM_ICPSC_DIV2
  - LL_TIM_ICPSC_DIV4
  - LL_TIM_ICPSC_DIV8

Reference Manual to LL API cross reference:

- CCMR1 IC1PSC LL_TIM_IC_GetPrescaler
- CCMR1 IC2PSC LL_TIM_IC_GetPrescaler
- CCMR2 IC3PSC LL_TIM_IC_GetPrescaler
- CCMR2 IC4PSC LL_TIM_IC_GetPrescaler

LL_TIM_IC_SetFilter

Function name

__STATIC_INLINE void LL_TIM_IC_SetFilter (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t ICFilter)

Function description

Set the input filter duration.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
- **ICFilter:** This parameter can be one of the following values:
  - LL_TIM_IC_FILTER_FDIV1
  - LL_TIM_IC_FILTER_FDIV1_N2
  - LL_TIM_IC_FILTER_FDIV1_N4
  - LL_TIM_IC_FILTER_FDIV1_N8
  - LL_TIM_IC_FILTER_FDIV2_N6
  - LL_TIM_IC_FILTER_FDIV2_N8
  - LL_TIM_IC_FILTER_FDIV4_N6
  - LL_TIM_IC_FILTER_FDIV4_N8
  - LL_TIM_IC_FILTER_FDIV8_N6
  - LL_TIM_IC_FILTER_FDIV8_N8
  - LL_TIM_IC_FILTER_FDIV16_N5
  - LL_TIM_IC_FILTER_FDIV16_N6
  - LL_TIM_IC_FILTER_FDIV16_N8
  - LL_TIM_IC_FILTER_FDIV32_N5
  - LL_TIM_IC_FILTER_FDIV32_N6
  - LL_TIM_IC_FILTER_FDIV32_N8

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCMR1 IC1F LL_TIM_IC_SetFilter
- CCMR1 IC2F LL_TIM_IC_SetFilter
- CCMR2 IC3F LL_TIM_IC_SetFilter
- CCMR2 IC4F LL_TIM_IC_SetFilter

LL_TIM_IC_GetFilter

Function name

__STATIC_INLINE uint32_t LL_TIM_IC_GetFilter (const TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Get the input filter duration.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4

Return values

- **Returned:** value can be one of the following values:
  - LL_TIM_IC_FILTER_FDIV1
  - LL_TIM_IC_FILTER_FDIV1_N2
  - LL_TIM_IC_FILTER_FDIV1_N4
  - LL_TIM_IC_FILTER_FDIV1_N8
  - LL_TIM_IC_FILTER_FDIV2_N6
  - LL_TIM_IC_FILTER_FDIV2_N8
  - LL_TIM_IC_FILTER_FDIV4_N6
  - LL_TIM_IC_FILTER_FDIV4_N8
  - LL_TIM_IC_FILTER_FDIV8_N6
  - LL_TIM_IC_FILTER_FDIV8_N8
  - LL_TIM_IC_FILTER_FDIV16_N5
  - LL_TIM_IC_FILTER_FDIV16_N6
  - LL_TIM_IC_FILTER_FDIV16_N8
  - LL_TIM_IC_FILTER_FDIV32_N5
  - LL_TIM_IC_FILTER_FDIV32_N6
  - LL_TIM_IC_FILTER_FDIV32_N8

Reference Manual to LL API cross reference:

- CCMR1 IC1F LL_TIM_IC_GetFilter
- CCMR1 IC2F LL_TIM_IC_GetFilter
- CCMR2 IC3F LL_TIM_IC_GetFilter
- CCMR2 IC4F LL_TIM_IC_GetFilter

LL_TIM_IC_SetPolarity

Function name

__STATIC_INLINE void LL_TIM_IC_SetPolarity (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t ICPolarity)

Function description

Set the input channel polarity.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
- **ICPolarity:** This parameter can be one of the following values:
  - LL_TIM_IC_POLARITY_RISING
  - LL_TIM_IC_POLARITY_FALLING
  - LL_TIM_IC_POLARITY_BOTHEDGE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCER CC1P LL_TIM_IC_SetPolarity
- CCER CC1NP LL_TIM_IC_SetPolarity
- CCER CC2P LL_TIM_IC_SetPolarity
- CCER CC2NP LL_TIM_IC_SetPolarity
- CCER CC3P LL_TIM_IC_SetPolarity
- CCER CC3NP LL_TIM_IC_SetPolarity
- CCER CC4P LL_TIM_IC_SetPolarity
- CCER CC4NP LL_TIM_IC_SetPolarity

LL_TIM_IC_GetPolarity

Function name

__STATIC_INLINE uint32_t LL_TIM_IC_GetPolarity (const TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Get the current input channel polarity.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4

Return values

- **Returned:** value can be one of the following values:
  - LL_TIM_IC_POLARITY_RISING
  - LL_TIM_IC_POLARITY_FALLING
  - LL_TIM_IC_POLARITY_BOTHEDGE

Reference Manual to LL API cross reference:

- CCER CC1P LL_TIM_IC_GetPolarity
- CCER CC1NP LL_TIM_IC_GetPolarity
- CCER CC2P LL_TIM_IC_GetPolarity
- CCER CC2NP LL_TIM_IC_GetPolarity
- CCER CC3P LL_TIM_IC_GetPolarity
- CCER CC3NP LL_TIM_IC_GetPolarity
- CCER CC4P LL_TIM_IC_GetPolarity
- CCER CC4NP LL_TIM_IC_GetPolarity

LL_TIM_IC_EnableXORCombination

Function name

__STATIC_INLINE void LL_TIM_IC_EnableXORCombination (TIM_TypeDef * TIMx)

Function description

Connect the TIMx_CH1, CH2 and CH3 pins to the TI1 input (XOR combination).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_XOR_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an XOR input.

Reference Manual to LL API cross reference:

- CR2 TI1S LL_TIM_IC_EnableXORCombination

LL_TIM_IC_DisableXORCombination

Function name

__STATIC_INLINE void LL_TIM_IC_DisableXORCombination (TIM_TypeDef * TIMx)

Function description

Disconnect the TIMx_CH1, CH2 and CH3 pins from the TI1 input.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_XOR_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an XOR input.

Reference Manual to LL API cross reference:

- CR2 TI1S LL_TIM_IC_DisableXORCombination

LL_TIM_IC_IsEnabledXORCombination

Function name

__STATIC_INLINE uint32_t LL_TIM_IC_IsEnabledXORCombination (const TIM_TypeDef * TIMx)

Function description

Indicates whether the TIMx_CH1, CH2 and CH3 pins are connectected to the TI1 input.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_XOR_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an XOR input.

Reference Manual to LL API cross reference:

- CR2 TI1S LL_TIM_IC_IsEnabledXORCombination

LL_TIM_IC_GetCaptureCH1

Function name

__STATIC_INLINE uint32_t LL_TIM_IC_GetCaptureCH1 (const TIM_TypeDef * TIMx)

Function description

Get captured value for input channel 1.

Parameters

- **TIMx:** Timer instance

Return values

- **CapturedValue:** (between Min_Data=0 and Max_Data=65535)

Notes

- In 32-bit timer implementations returned captured value can be between 0x00000000 and 0xFFFFFFFF.
- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- Macro IS_TIM_CC1_INSTANCE(TIMx) can be used to check whether or not input channel 1 is supported by a timer instance.
- If dithering is activated, pay attention to the returned value interpretation.

Reference Manual to LL API cross reference:

- CCR1 CCR1 LL_TIM_IC_GetCaptureCH1

LL_TIM_IC_GetCaptureCH2

Function name

__STATIC_INLINE uint32_t LL_TIM_IC_GetCaptureCH2 (const TIM_TypeDef * TIMx)

Function description

Get captured value for input channel 2.

Parameters

- **TIMx:** Timer instance

Return values

- **CapturedValue:** (between Min_Data=0 and Max_Data=65535)

Notes

- In 32-bit timer implementations returned captured value can be between 0x00000000 and 0xFFFFFFFF.
- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- Macro IS_TIM_CC2_INSTANCE(TIMx) can be used to check whether or not input channel 2 is supported by a timer instance.
- If dithering is activated, pay attention to the returned value interpretation.

Reference Manual to LL API cross reference:

- CCR2 CCR2 LL_TIM_IC_GetCaptureCH2

LL_TIM_IC_GetCaptureCH3

Function name

__STATIC_INLINE uint32_t LL_TIM_IC_GetCaptureCH3 (const TIM_TypeDef * TIMx)

Function description

Get captured value for input channel 3.

Parameters

- **TIMx:** Timer instance

Return values

- **CapturedValue:** (between Min_Data=0 and Max_Data=65535)

Notes

- In 32-bit timer implementations returned captured value can be between 0x00000000 and 0xFFFFFFFF.
- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- Macro IS_TIM_CC3_INSTANCE(TIMx) can be used to check whether or not input channel 3 is supported by a timer instance.
- If dithering is activated, pay attention to the returned value interpretation.

Reference Manual to LL API cross reference:

- CCR3 CCR3 LL_TIM_IC_GetCaptureCH3

LL_TIM_IC_GetCaptureCH4

Function name

__STATIC_INLINE uint32_t LL_TIM_IC_GetCaptureCH4 (const TIM_TypeDef * TIMx)

Function description

Get captured value for input channel 4.

Parameters

- **TIMx:** Timer instance

Return values

- **CapturedValue:** (between Min_Data=0 and Max_Data=65535)

Notes

- In 32-bit timer implementations returned captured value can be between 0x00000000 and 0xFFFFFFFF.
- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- Macro IS_TIM_CC4_INSTANCE(TIMx) can be used to check whether or not input channel 4 is supported by a timer instance.
- If dithering is activated, pay attention to the returned value interpretation.

Reference Manual to LL API cross reference:

- CCR4 CCR4 LL_TIM_IC_GetCaptureCH4

LL_TIM_EnableExternalClock

Function name

__STATIC_INLINE void LL_TIM_EnableExternalClock (TIM_TypeDef * TIMx)

Function description

Enable external clock mode 2.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- When external clock mode 2 is enabled the counter is clocked by any active edge on the ETRF signal.
- Macro IS_TIM_CLOCKSOURCE_ETRMODE2_INSTANCE(TIMx) can be used to check whether or not a timer instance supports external clock mode2.

Reference Manual to LL API cross reference:

- SMCR ECE LL_TIM_EnableExternalClock

LL_TIM_DisableExternalClock

Function name

__STATIC_INLINE void LL_TIM_DisableExternalClock (TIM_TypeDef * TIMx)

Function description

Disable external clock mode 2.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_CLOCKSOURCE_ETRMODE2_INSTANCE(TIMx) can be used to check whether or not a timer instance supports external clock mode2.

Reference Manual to LL API cross reference:

- SMCR ECE LL_TIM_DisableExternalClock

LL_TIM_IsEnabledExternalClock

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledExternalClock (const TIM_TypeDef * TIMx)

Function description

Indicate whether external clock mode 2 is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_CLOCKSOURCE_ETRMODE2_INSTANCE(TIMx) can be used to check whether or not a timer instance supports external clock mode2.

Reference Manual to LL API cross reference:

- SMCR ECE LL_TIM_IsEnabledExternalClock

LL_TIM_SetClockSource

Function name

__STATIC_INLINE void LL_TIM_SetClockSource (TIM_TypeDef * TIMx, uint32_t ClockSource)

Function description

Set the clock source of the counter clock.

Parameters

- **TIMx:** Timer instance
- **ClockSource:** This parameter can be one of the following values:
  - LL_TIM_CLOCKSOURCE_INTERNAL
  - LL_TIM_CLOCKSOURCE_EXT_MODE1
  - LL_TIM_CLOCKSOURCE_EXT_MODE2

Return values

- **None:**

Notes

- when selected clock source is external clock mode 1, the timer input the external clock is applied is selected by calling the LL_TIM_SetTriggerInput() function. This timer input must be configured by calling the LL_TIM_IC_Config() function.
- Macro IS_TIM_CLOCKSOURCE_ETRMODE1_INSTANCE(TIMx) can be used to check whether or not a timer instance supports external clock mode1.
- Macro IS_TIM_CLOCKSOURCE_ETRMODE2_INSTANCE(TIMx) can be used to check whether or not a timer instance supports external clock mode2.

Reference Manual to LL API cross reference:

- SMCR SMS LL_TIM_SetClockSource
- SMCR ECE LL_TIM_SetClockSource

LL_TIM_SetEncoderMode

Function name

__STATIC_INLINE void LL_TIM_SetEncoderMode (TIM_TypeDef * TIMx, uint32_t EncoderMode)

Function description

Set the encoder interface mode.

Parameters

- **TIMx:** Timer instance
- **EncoderMode:** This parameter can be one of the following values:
  - LL_TIM_ENCODERMODE_X2_TI1
  - LL_TIM_ENCODERMODE_X2_TI2
  - LL_TIM_ENCODERMODE_X4_TI12
  - LL_TIM_ENCODERMODE_CLOCKPLUSDIRECTION_X2
  - LL_TIM_ENCODERMODE_CLOCKPLUSDIRECTION_X1
  - LL_TIM_ENCODERMODE_DIRECTIONALCLOCK_X2
  - LL_TIM_ENCODERMODE_DIRECTIONALCLOCK_X1_TI12
  - LL_TIM_ENCODERMODE_X1_TI1
  - LL_TIM_ENCODERMODE_X1_TI2

Return values

- **None:**

Notes

- Macro IS_TIM_ENCODER_INTERFACE_INSTANCE(TIMx) can be used to check whether or not a timer instance supports the encoder mode.

Reference Manual to LL API cross reference:

- SMCR SMS LL_TIM_SetEncoderMode

LL_TIM_SetTriggerOutput

Function name

__STATIC_INLINE void LL_TIM_SetTriggerOutput (TIM_TypeDef * TIMx, uint32_t TimerSynchronization)

Function description

Set the trigger output (TRGO) used for timer synchronization .

Parameters

- **TIMx:** Timer instance
- **TimerSynchronization:** This parameter can be one of the following values:
  - LL_TIM_TRGO_RESET
  - LL_TIM_TRGO_ENABLE
  - LL_TIM_TRGO_UPDATE
  - LL_TIM_TRGO_CC1IF
  - LL_TIM_TRGO_OC1REF
  - LL_TIM_TRGO_OC2REF
  - LL_TIM_TRGO_OC3REF
  - LL_TIM_TRGO_OC4REF
  - LL_TIM_TRGO_ENCODERCLK

Return values

- **None:**

Notes

- Macro IS_TIM_MASTER_INSTANCE(TIMx) can be used to check whether or not a timer instance can operate as a master timer.

Reference Manual to LL API cross reference:

- CR2 MMS LL_TIM_SetTriggerOutput

LL_TIM_SetTriggerOutput2

Function name

__STATIC_INLINE void LL_TIM_SetTriggerOutput2 (TIM_TypeDef * TIMx, uint32_t ADCSynchronization)

Function description

Set the trigger output 2 (TRGO2) used for ADC synchronization .

Parameters

- **TIMx:** Timer Instance
- **ADCSynchronization:** This parameter can be one of the following values:
  - LL_TIM_TRGO2_RESET
  - LL_TIM_TRGO2_ENABLE
  - LL_TIM_TRGO2_UPDATE
  - LL_TIM_TRGO2_CC1F
  - LL_TIM_TRGO2_OC1
  - LL_TIM_TRGO2_OC2
  - LL_TIM_TRGO2_OC3
  - LL_TIM_TRGO2_OC4
  - LL_TIM_TRGO2_OC5
  - LL_TIM_TRGO2_OC6
  - LL_TIM_TRGO2_OC4_RISINGFALLING
  - LL_TIM_TRGO2_OC6_RISINGFALLING
  - LL_TIM_TRGO2_OC4_RISING_OC6_RISING
  - LL_TIM_TRGO2_OC4_RISING_OC6_FALLING
  - LL_TIM_TRGO2_OC5_RISING_OC6_RISING
  - LL_TIM_TRGO2_OC5_RISING_OC6_FALLING

Return values

- **None:**

Notes

- Macro IS_TIM_TRGO2_INSTANCE(TIMx) can be used to check whether or not a timer instance can be used for ADC synchronization.

Reference Manual to LL API cross reference:

- CR2 MMS2 LL_TIM_SetTriggerOutput2

LL_TIM_SetSlaveMode

Function name

__STATIC_INLINE void LL_TIM_SetSlaveMode (TIM_TypeDef * TIMx, uint32_t SlaveMode)

Function description

Set the synchronization mode of a slave timer.

Parameters

- **TIMx:** Timer instance
- **SlaveMode:** This parameter can be one of the following values:
  - LL_TIM_SLAVEMODE_DISABLED
  - LL_TIM_SLAVEMODE_RESET
  - LL_TIM_SLAVEMODE_GATED
  - LL_TIM_SLAVEMODE_TRIGGER
  - LL_TIM_SLAVEMODE_COMBINED_RESETTRIGGER
  - LL_TIM_SLAVEMODE_COMBINED_GATEDRESET

Return values

- **None:**

Notes

- Macro IS_TIM_SLAVE_INSTANCE(TIMx) can be used to check whether or not a timer instance can operate as a slave timer.

Reference Manual to LL API cross reference:

- SMCR SMS LL_TIM_SetSlaveMode

LL_TIM_SetTriggerInput

Function name

__STATIC_INLINE void LL_TIM_SetTriggerInput (TIM_TypeDef * TIMx, uint32_t TriggerInput)

Function description

Set the selects the trigger input to be used to synchronize the counter.

Parameters

- **TIMx:** Timer instance
- **TriggerInput:** This parameter can be one of the following values:
  - LL_TIM_TS_ITR0
  - LL_TIM_TS_ITR1
  - LL_TIM_TS_ITR2
  - LL_TIM_TS_ITR3
  - LL_TIM_TS_ITR4
  - LL_TIM_TS_ITR5
  - LL_TIM_TS_ITR6
  - LL_TIM_TS_ITR7
  - LL_TIM_TS_ITR8
  - LL_TIM_TS_ITR9
  - LL_TIM_TS_ITR10
  - LL_TIM_TS_ITR11
  - LL_TIM_TS_ITR12
  - LL_TIM_TS_ITR13
  - LL_TIM_TS_TI1F_ED
  - LL_TIM_TS_TI1FP1
  - LL_TIM_TS_TI2FP2
  - LL_TIM_TS_ETRF

Return values

- **None:**

Notes

- Macro IS_TIM_SLAVE_INSTANCE(TIMx) can be used to check whether or not a timer instance can operate as a slave timer.

Reference Manual to LL API cross reference:

- SMCR TS LL_TIM_SetTriggerInput

LL_TIM_EnableMasterSlaveMode

Function name

__STATIC_INLINE void LL_TIM_EnableMasterSlaveMode (TIM_TypeDef * TIMx)

Function description

Enable the Master/Slave mode.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_SLAVE_INSTANCE(TIMx) can be used to check whether or not a timer instance can operate as a slave timer.

Reference Manual to LL API cross reference:

- SMCR MSM LL_TIM_EnableMasterSlaveMode

LL_TIM_DisableMasterSlaveMode

Function name

__STATIC_INLINE void LL_TIM_DisableMasterSlaveMode (TIM_TypeDef * TIMx)

Function description

Disable the Master/Slave mode.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_SLAVE_INSTANCE(TIMx) can be used to check whether or not a timer instance can operate as a slave timer.

Reference Manual to LL API cross reference:

- SMCR MSM LL_TIM_DisableMasterSlaveMode

LL_TIM_IsEnabledMasterSlaveMode

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledMasterSlaveMode (const TIM_TypeDef * TIMx)

Function description

Indicates whether the Master/Slave mode is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_SLAVE_INSTANCE(TIMx) can be used to check whether or not a timer instance can operate as a slave timer.

Reference Manual to LL API cross reference:

- SMCR MSM LL_TIM_IsEnabledMasterSlaveMode

LL_TIM_ConfigETR

Function name

__STATIC_INLINE void LL_TIM_ConfigETR (TIM_TypeDef * TIMx, uint32_t ETRPolarity, uint32_t ETRPrescaler, uint32_t ETRFilter)

Function description

Configure the external trigger (ETR) input.

Parameters

- **TIMx:** Timer instance
- **ETRPolarity:** This parameter can be one of the following values:
  - LL_TIM_ETR_POLARITY_NONINVERTED
  - LL_TIM_ETR_POLARITY_INVERTED
- **ETRPrescaler:** This parameter can be one of the following values:
  - LL_TIM_ETR_PRESCALER_DIV1
  - LL_TIM_ETR_PRESCALER_DIV2
  - LL_TIM_ETR_PRESCALER_DIV4
  - LL_TIM_ETR_PRESCALER_DIV8
- **ETRFilter:** This parameter can be one of the following values:
  - LL_TIM_ETR_FILTER_FDIV1
  - LL_TIM_ETR_FILTER_FDIV1_N2
  - LL_TIM_ETR_FILTER_FDIV1_N4
  - LL_TIM_ETR_FILTER_FDIV1_N8
  - LL_TIM_ETR_FILTER_FDIV2_N6
  - LL_TIM_ETR_FILTER_FDIV2_N8
  - LL_TIM_ETR_FILTER_FDIV4_N6
  - LL_TIM_ETR_FILTER_FDIV4_N8
  - LL_TIM_ETR_FILTER_FDIV8_N6
  - LL_TIM_ETR_FILTER_FDIV8_N8
  - LL_TIM_ETR_FILTER_FDIV16_N5
  - LL_TIM_ETR_FILTER_FDIV16_N6
  - LL_TIM_ETR_FILTER_FDIV16_N8
  - LL_TIM_ETR_FILTER_FDIV32_N5
  - LL_TIM_ETR_FILTER_FDIV32_N6
  - LL_TIM_ETR_FILTER_FDIV32_N8

Return values

- **None:**

Notes

- Macro IS_TIM_ETR_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an external trigger input.

Reference Manual to LL API cross reference:

- SMCR ETP LL_TIM_ConfigETR
- SMCR ETPS LL_TIM_ConfigETR
- SMCR ETF LL_TIM_ConfigETR

LL_TIM_SetETRSource

Function name

__STATIC_INLINE void LL_TIM_SetETRSource (TIM_TypeDef * TIMx, uint32_t ETRSource)

Function description

Select the external trigger (ETR) input source.

Parameters

- **TIMx:** Timer instance
- **ETRSource:** This parameter can be one of the following values:

Notes

- Macro IS_TIM_ETRSEL_INSTANCE(TIMx) can be used to check whether or not a timer instance supports ETR source selection.

Reference Manual to LL API cross reference:

- AF1 ETRSEL LL_TIM_SetETRSource

LL_TIM_EnableSMSPreload

Function name

__STATIC_INLINE void LL_TIM_EnableSMSPreload (TIM_TypeDef * TIMx)

Function description

Enable SMS preload.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_SMS_PRELOAD_INSTANCE(TIMx) can be used to check whether or not a timer instance supports the preload of SMS field in SMCR register.

Reference Manual to LL API cross reference:

- SMCR SMSPE LL_TIM_EnableSMSPreload

LL_TIM_DisableSMSPreload

Function name

__STATIC_INLINE void LL_TIM_DisableSMSPreload (TIM_TypeDef * TIMx)

Function description

Disable SMS preload.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_SMS_PRELOAD_INSTANCE(TIMx) can be used to check whether or not a timer instance supports the preload of SMS field in SMCR register.

Reference Manual to LL API cross reference:

- SMCR SMSPE LL_TIM_DisableSMSPreload

LL_TIM_IsEnabledSMSPreload

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledSMSPreload (const TIM_TypeDef * TIMx)

Function description

Indicate whether SMS preload is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_SMS_PRELOAD_INSTANCE(TIMx) can be used to check whether or not a timer instance supports the preload of SMS field in SMCR register.

Reference Manual to LL API cross reference:

- SMCR SMSPE LL_TIM_IsEnabledSMSPreload

LL_TIM_SetSMSPreloadSource

Function name

__STATIC_INLINE void LL_TIM_SetSMSPreloadSource (TIM_TypeDef * TIMx, uint32_t PreloadSource)

Function description

Set the preload source of SMS.

Parameters

- **TIMx:** Timer instance
- **PreloadSource:** This parameter can be one of the following values:
  - LL_TIM_SMSPS_TIMUPDATE
  - LL_TIM_SMSPS_INDEX

Return values

- **None:**

Notes

- Macro IS_TIM_SMS_PRELOAD_INSTANCE(TIMx) can be used to check whether or not a timer instance supports the preload of SMS field in SMCR register.

Reference Manual to LL API cross reference:

- SMCR SMSPS LL_TIM_SetSMSPreloadSource
-

LL_TIM_GetSMSPreloadSource

Function name

__STATIC_INLINE uint32_t LL_TIM_GetSMSPreloadSource (const TIM_TypeDef * TIMx)

Function description

Get the preload source of SMS.

Parameters

- **TIMx:** Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_TIM_SMSPS_TIMUPDATE
  - LL_TIM_SMSPS_INDEX

Notes

- Macro IS_TIM_SMS_PRELOAD_INSTANCE(TIMx) can be used to check whether or not a timer instance supports the preload of SMS field in SMCR register.

Reference Manual to LL API cross reference:

- SMCR SMSPS LL_TIM_GetSMSPreloadSource
-

LL_TIM_EnableADCSynchronization

Function name

__STATIC_INLINE void LL_TIM_EnableADCSynchronization (TIM_TypeDef * TIMx)

Function description

Enable ADC synchronization.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_MASTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports ADC synchronization.

Reference Manual to LL API cross reference:

- CR2 ADSYNC LL_TIM_EnableADCSynchronization

LL_TIM_DisableADCSynchronization

Function name

__STATIC_INLINE void LL_TIM_DisableADCSynchronization (TIM_TypeDef * TIMx)

Function description

Enable ADC synchronization.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_MASTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports ADC synchronization.

Reference Manual to LL API cross reference:

- CR2 ADSYNC LL_TIM_DisableADCSynchronization

LL_TIM_IsEnabledADCSynchronization

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledADCSynchronization (const TIM_TypeDef * TIMx)

Function description

Indicate whether ADC sycnhronization is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_MASTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports ADC synchronization.

Reference Manual to LL API cross reference:

- CR2 ADSYNC LL_TIM_IsEnabledADCSynchronization

LL_TIM_EnableBRK

Function name

__STATIC_INLINE void LL_TIM_EnableBRK (TIM_TypeDef * TIMx)

Function description

Enable the break function.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.

Reference Manual to LL API cross reference:

- BDTR BKE LL_TIM_EnableBRK

LL_TIM_DisableBRK

Function name

__STATIC_INLINE void LL_TIM_DisableBRK (TIM_TypeDef * TIMx)

Function description

Disable the break function.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.

Reference Manual to LL API cross reference:

- BDTR BKE LL_TIM_DisableBRK

LL_TIM_ConfigBRK

Function name

__STATIC_INLINE void LL_TIM_ConfigBRK (TIM_TypeDef * TIMx, uint32_t BreakPolarity, uint32_t BreakFilter, uint32_t BreakAFMode)

Function description

Configure the break input.

Parameters

- **TIMx:** Timer instance
- **BreakPolarity:** This parameter can be one of the following values:
  - LL_TIM_BREAK_POLARITY_LOW
  - LL_TIM_BREAK_POLARITY_HIGH
- **BreakFilter:** This parameter can be one of the following values:
  - LL_TIM_BREAK_FILTER_FDIV1
  - LL_TIM_BREAK_FILTER_FDIV1_N2
  - LL_TIM_BREAK_FILTER_FDIV1_N4
  - LL_TIM_BREAK_FILTER_FDIV1_N8
  - LL_TIM_BREAK_FILTER_FDIV2_N6
  - LL_TIM_BREAK_FILTER_FDIV2_N8
  - LL_TIM_BREAK_FILTER_FDIV4_N6
  - LL_TIM_BREAK_FILTER_FDIV4_N8
  - LL_TIM_BREAK_FILTER_FDIV8_N6
  - LL_TIM_BREAK_FILTER_FDIV8_N8
  - LL_TIM_BREAK_FILTER_FDIV16_N5
  - LL_TIM_BREAK_FILTER_FDIV16_N6
  - LL_TIM_BREAK_FILTER_FDIV16_N8
  - LL_TIM_BREAK_FILTER_FDIV32_N5
  - LL_TIM_BREAK_FILTER_FDIV32_N6
  - LL_TIM_BREAK_FILTER_FDIV32_N8
- **BreakAFMode:** This parameter can be one of the following values:
  - LL_TIM_BREAK_AFMODE_INPUT
  - LL_TIM_BREAK_AFMODE_BIDIRECTIONAL

Return values

- **None:**

Notes

- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.
- Bidirectional mode is only supported by advanced timer instances. Macro IS_TIM_ADVANCED_INSTANCE(TIMx) can be used to check whether or not a timer instance is an advanced-control timer.
- In bidirectional mode (BKBID bit set), the Break input is configured both in input mode and in open drain output mode. Any active Break event will assert a low logic level on the Break input to indicate an internal break event to external devices.
- When bidirectional mode isn't supported, BreakAFMode must be set to LL_TIM_BREAK_AFMODE_INPUT.

Reference Manual to LL API cross reference:

- BDTR BKP LL_TIM_ConfigBRK
- BDTR BKF LL_TIM_ConfigBRK
- BDTR BKBID LL_TIM_ConfigBRK

LL_TIM_DisarmBRK

Function name

__STATIC_INLINE void LL_TIM_DisarmBRK (TIM_TypeDef * TIMx)

Function description

Disarm the break input (when it operates in bidirectional mode).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- The break input can be disarmed only when it is configured in bidirectional mode and when when MOE is reset.
- Purpose is to be able to have the input voltage back to high-state, whatever the time constant on the output .

Reference Manual to LL API cross reference:

- BDTR BKDSRM LL_TIM_DisarmBRK

LL_TIM_EnableBRK2

Function name

__STATIC_INLINE void LL_TIM_EnableBRK2 (TIM_TypeDef * TIMx)

Function description

Enable the break 2 function.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_BKIN2_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a second break input.

Reference Manual to LL API cross reference:

- BDTR BK2E LL_TIM_EnableBRK2

LL_TIM_DisableBRK2

Function name

__STATIC_INLINE void LL_TIM_DisableBRK2 (TIM_TypeDef * TIMx)

Function description

Disable the break 2 function.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_BKIN2_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a second break input.

Reference Manual to LL API cross reference:

- BDTR BK2E LL_TIM_DisableBRK2

LL_TIM_ConfigBRK2

Function name

__STATIC_INLINE void LL_TIM_ConfigBRK2 (TIM_TypeDef * TIMx, uint32_t Break2Polarity, uint32_t Break2Filter, uint32_t Break2AFMode)

Function description

Configure the break 2 input.

Parameters

- **TIMx:** Timer instance
- **Break2Polarity:** This parameter can be one of the following values:
  - LL_TIM_BREAK2_POLARITY_LOW
  - LL_TIM_BREAK2_POLARITY_HIGH
- **Break2Filter:** This parameter can be one of the following values:
  - LL_TIM_BREAK2_FILTER_FDIV1
  - LL_TIM_BREAK2_FILTER_FDIV1_N2
  - LL_TIM_BREAK2_FILTER_FDIV1_N4
  - LL_TIM_BREAK2_FILTER_FDIV1_N8
  - LL_TIM_BREAK2_FILTER_FDIV2_N6
  - LL_TIM_BREAK2_FILTER_FDIV2_N8
  - LL_TIM_BREAK2_FILTER_FDIV4_N6
  - LL_TIM_BREAK2_FILTER_FDIV4_N8
  - LL_TIM_BREAK2_FILTER_FDIV8_N6
  - LL_TIM_BREAK2_FILTER_FDIV8_N8
  - LL_TIM_BREAK2_FILTER_FDIV16_N5
  - LL_TIM_BREAK2_FILTER_FDIV16_N6
  - LL_TIM_BREAK2_FILTER_FDIV16_N8
  - LL_TIM_BREAK2_FILTER_FDIV32_N5
  - LL_TIM_BREAK2_FILTER_FDIV32_N6
  - LL_TIM_BREAK2_FILTER_FDIV32_N8
- **Break2AFMode:** This parameter can be one of the following values:
  - LL_TIM_BREAK2_AFMODE_INPUT
  - LL_TIM_BREAK2_AFMODE_BIDIRECTIONAL

Return values

- **None:**

Notes

- Macro IS_TIM_BKIN2_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a second break input.
- Bidirectional mode is only supported by advanced timer instances. Macro IS_TIM_ADVANCED_INSTANCE(TIMx) can be used to check whether or not a timer instance is an advanced-control timer.
- In bidirectional mode (BK2BID bit set), the Break 2 input is configured both in input mode and in open drain output mode. Any active Break event will assert a low logic level on the Break 2 input to indicate an internal break event to external devices.
- When bidirectional mode isn't supported, Break2AFMode must be set to LL_TIM_BREAK2_AFMODE_INPUT.

Reference Manual to LL API cross reference:

- BDTR BK2P LL_TIM_ConfigBRK2
- BDTR BK2F LL_TIM_ConfigBRK2
- BDTR BK2BID LL_TIM_ConfigBRK2

LL_TIM_DisarmBRK2

Function name

__STATIC_INLINE void LL_TIM_DisarmBRK2 (TIM_TypeDef * TIMx)

Function description

Disarm the break 2 input (when it operates in bidirectional mode).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- The break 2 input can be disarmed only when it is configured in bidirectional mode and when when MOE is reset.
- Purpose is to be able to have the input voltage back to high-state, whatever the time constant on the output.

Reference Manual to LL API cross reference:

- BDTR BK2DSRM LL_TIM_DisarmBRK2

LL_TIM_SetOffStates

Function name

__STATIC_INLINE void LL_TIM_SetOffStates (TIM_TypeDef * TIMx, uint32_t OffStateIdle, uint32_t OffStateRun)

Function description

Select the outputs off state (enabled v.s.

Parameters

- **TIMx:** Timer instance
- **OffStateIdle:** This parameter can be one of the following values:
  - LL_TIM_OSSI_DISABLE
  - LL_TIM_OSSI_ENABLE
- **OffStateRun:** This parameter can be one of the following values:
  - LL_TIM_OSSR_DISABLE
  - LL_TIM_OSSR_ENABLE

Return values

- **None:**

Notes

- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.

Reference Manual to LL API cross reference:

- BDTR OSSI LL_TIM_SetOffStates
- BDTR OSSR LL_TIM_SetOffStates

LL_TIM_EnableAutomaticOutput

Function name

__STATIC_INLINE void LL_TIM_EnableAutomaticOutput (TIM_TypeDef * TIMx)

Function description

Enable automatic output (MOE can be set by software or automatically when a break input is active).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.

Reference Manual to LL API cross reference:

- BDTR AOE LL_TIM_EnableAutomaticOutput

LL_TIM_DisableAutomaticOutput

Function name

__STATIC_INLINE void LL_TIM_DisableAutomaticOutput (TIM_TypeDef * TIMx)

Function description

Disable automatic output (MOE can be set only by software).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.

Reference Manual to LL API cross reference:

- BDTR AOE LL_TIM_DisableAutomaticOutput

LL_TIM_IsEnabledAutomaticOutput

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledAutomaticOutput (const TIM_TypeDef * TIMx)

Function description

Indicate whether automatic output is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.

Reference Manual to LL API cross reference:

- BDTR AOE LL_TIM_IsEnabledAutomaticOutput

LL_TIM_EnableAllOutputs

Function name

__STATIC_INLINE void LL_TIM_EnableAllOutputs (TIM_TypeDef * TIMx)

Function description

Enable the outputs (set the MOE bit in TIMx_BDTR register).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- The MOE bit in TIMx_BDTR register allows to enable /disable the outputs by software and is reset in case of break or break2 event
- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.

Reference Manual to LL API cross reference:

- BDTR MOE LL_TIM_EnableAllOutputs

LL_TIM_DisableAllOutputs

Function name

__STATIC_INLINE void LL_TIM_DisableAllOutputs (TIM_TypeDef * TIMx)

Function description

Disable the outputs (reset the MOE bit in TIMx_BDTR register).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- The MOE bit in TIMx_BDTR register allows to enable /disable the outputs by software and is reset in case of break or break2 event.
- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.

Reference Manual to LL API cross reference:

- BDTR MOE LL_TIM_DisableAllOutputs

LL_TIM_IsEnabledAllOutputs

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledAllOutputs (const TIM_TypeDef * TIMx)

Function description

Indicates whether outputs are enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.

Reference Manual to LL API cross reference:

- BDTR MOE LL_TIM_IsEnabledAllOutputs

LL_TIM_EnableBreakInputSource

Function name

__STATIC_INLINE void LL_TIM_EnableBreakInputSource (TIM_TypeDef * TIMx, uint32_t BreakInput, uint32_t Source)

Function description

Enable the signals connected to the designated timer break input.

Parameters

- **TIMx:** Timer instance
- **BreakInput:** This parameter can be one of the following values:
  - LL_TIM_BREAK_INPUT_BKIN
  - LL_TIM_BREAK_INPUT_BKIN2
- **Source:** This parameter can be one of the following values:
  - LL_TIM_BKIN_SOURCE_BKIN
  - LL_TIM_BKIN_SOURCE_MDF1

Return values

- **None:**

Notes

- Macro IS_TIM_BREAKSOURCE_INSTANCE(TIMx) can be used to check whether or not a timer instance allows for break input selection.

Reference Manual to LL API cross reference:

- AF1 BKINE LL_TIM_EnableBreakInputSource
- AF1 BKCMP8E LL_TIM_EnableBreakInputSource
- AF2 BK2INE LL_TIM_EnableBreakInputSource
- AF2 BK2CMP8E LL_TIM_EnableBreakInputSource
-

LL_TIM_DisableBreakInputSource

Function name

__STATIC_INLINE void LL_TIM_DisableBreakInputSource (TIM_TypeDef * TIMx, uint32_t BreakInput, uint32_t Source)

Function description

Disable the signals connected to the designated timer break input.

Parameters

- **TIMx:** Timer instance
- **BreakInput:** This parameter can be one of the following values:
  - LL_TIM_BREAK_INPUT_BKIN
  - LL_TIM_BREAK_INPUT_BKIN2
- **Source:** This parameter can be one of the following values:
  - LL_TIM_BKIN_SOURCE_BKIN
  - LL_TIM_BKIN_SOURCE_MDF1

Return values

- **None:**

Notes

- Macro IS_TIM_BREAKSOURCE_INSTANCE(TIMx) can be used to check whether or not a timer instance allows for break input selection.

Reference Manual to LL API cross reference:

- AF1 BKINE LL_TIM_DisableBreakInputSource
- AF1 BKCMP8E LL_TIM_DisableBreakInputSource
- AF2 BK2INE LL_TIM_DisableBreakInputSource
- AF2 BK2CMP8E LL_TIM_DisableBreakInputSource
-

LL_TIM_SetBreakInputSourcePolarity

Function name

__STATIC_INLINE void LL_TIM_SetBreakInputSourcePolarity (TIM_TypeDef * TIMx, uint32_t BreakInput, uint32_t Source, uint32_t Polarity)

Function description

Set the polarity of the break signal for the timer break input.

Parameters

- **TIMx:** Timer instance
- **BreakInput:** This parameter can be one of the following values:
  - LL_TIM_BREAK_INPUT_BKIN
  - LL_TIM_BREAK_INPUT_BKIN2
- **Source:** This parameter can be one of the following values:
  - LL_TIM_BKIN_SOURCE_BKIN
- **Polarity:** This parameter can be one of the following values:
  - LL_TIM_BKIN_POLARITY_LOW
  - LL_TIM_BKIN_POLARITY_HIGH

Return values

- **None:**

Notes

- Macro IS_TIM_BREAKSOURCE_INSTANCE(TIMx) can be used to check whether or not a timer instance allows for break input selection.

Reference Manual to LL API cross reference:

- AF1 BKINP LL_TIM_SetBreakInputSourcePolarity
- AF2 BK2INP LL_TIM_SetBreakInputSourcePolarity

LL_TIM_EnableAsymmetricalDeadTime

Function name

__STATIC_INLINE void LL_TIM_EnableAsymmetricalDeadTime (TIM_TypeDef * TIMx)

Function description

Enable asymmetrical deadtime.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_DEADTIME_ASYMMETRICAL_INSTANCE(TIMx) can be used to check whether or not a timer instance provides asymmetrical deadtime.

Reference Manual to LL API cross reference:

- DTR2 DTAE LL_TIM_EnableAsymmetricalDeadTime

LL_TIM_DisableAsymmetricalDeadTime

Function name

__STATIC_INLINE void LL_TIM_DisableAsymmetricalDeadTime (TIM_TypeDef * TIMx)

Function description

Disable asymmetrical dead-time.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_DEADTIME_ASYMMETRICAL_INSTANCE(TIMx) can be used to check whether or not a timer instance provides asymmetrical deadtime.

Reference Manual to LL API cross reference:

- DTR2 DTAE LL_TIM_DisableAsymmetricalDeadTime

LL_TIM_IsEnabledAsymmetricalDeadTime

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledAsymmetricalDeadTime (const TIM_TypeDef * TIMx)

Function description

Indicates whether asymmetrical deadtime is activated.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_DEADTIME_ASYMMETRICAL_INSTANCE(TIMx) can be used to check whether or not a timer instance provides asymmetrical deadtime.

Reference Manual to LL API cross reference:

- DTR2 DTAE LL_TIM_IsEnabledAsymmetricalDeadTime

LL_TIM_SetFallingDeadTime

Function name

__STATIC_INLINE void LL_TIM_SetFallingDeadTime (TIM_TypeDef * TIMx, uint32_t DeadTime)

Function description

Set the falling edge dead-time delay (delay inserted between the falling edge of the OCxREF signal and the rising edge of OCxN signals).

Parameters

- **TIMx:** Timer instance
- **DeadTime:** between Min_Data=0 and Max_Data=255

Return values

- **None:**

Notes

- Macro IS_TIM_DEADTIME_ASYMMETRICAL_INSTANCE(TIMx) can be used to check whether or not asymmetrical dead-time insertion feature is supported by a timer instance.
- Helper macro __LL_TIM_CALC_DEADTIME can be used to calculate the DeadTime parameter
- This bit-field can not be modified as long as LOCK level 1, 2 or 3 has been programmed (LOCK bits in TIMx_BDTR register).

Reference Manual to LL API cross reference:

- DTR2 DTGF LL_TIM_SetFallingDeadTime

LL_TIM_GetFallingDeadTime

Function name

__STATIC_INLINE uint32_t LL_TIM_GetFallingDeadTime (const TIM_TypeDef * TIMx)

Function description

Get the falling edge dead-time delay (delay inserted between the falling edge of the OCxREF signal and the rising edge of OCxN signals).

Parameters

- **TIMx:** Timer instance

Return values

- **Returned:** value can be between Min_Data=0 and Max_Data=255:

Notes

- Macro IS_TIM_DEADTIME_ASYMMETRICAL_INSTANCE(TIMx) can be used to check whether or not asymmetrical dead-time insertion feature is supported by a timer instance.
- This bit-field can not be modified as long as LOCK level 1, 2 or 3 has been programmed (LOCK bits in TIMx_BDTR register).

Reference Manual to LL API cross reference:

- DTR2 DTGF LL_TIM_GetFallingDeadTime

LL_TIM_EnableDeadTimePreload

Function name

__STATIC_INLINE void LL_TIM_EnableDeadTimePreload (TIM_TypeDef * TIMx)

Function description

Enable deadtime preload.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides deadtime preload.

Reference Manual to LL API cross reference:

- DTR2 DTPE LL_TIM_EnableDeadTimePreload

LL_TIM_DisableDeadTimePreload

Function name

__STATIC_INLINE void LL_TIM_DisableDeadTimePreload (TIM_TypeDef * TIMx)

Function description

Disable dead-time preload.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides deadtime preload.

Reference Manual to LL API cross reference:

- DTR2 DTPE LL_TIM_DisableDeadTimePreload

LL_TIM_IsEnabledDeadTimePreload

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledDeadTimePreload (const TIM_TypeDef * TIMx)

Function description

Indicates whether deadtime preload is activated.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides deadtime preload.

Reference Manual to LL API cross reference:

- DTR2 DTPE LL_TIM_IsEnabledDeadTimePreload

LL_TIM_ConfigDMABurst

Function name

__STATIC_INLINE void LL_TIM_ConfigDMABurst (TIM_TypeDef * TIMx, uint32_t DMABurstBaseAddress, uint32_t DMABurstLength, uint32_t DMABurstSource)

Function description

Configures the timer DMA burst feature.

Parameters

- **TIMx:** Timer instance
- **DMABurstBaseAddress:** This parameter can be one of the following values:
  - LL_TIM_DMABURST_BASEADDR_CR1
  - LL_TIM_DMABURST_BASEADDR_CR2
  - LL_TIM_DMABURST_BASEADDR_SMCR
  - LL_TIM_DMABURST_BASEADDR_DIER
  - LL_TIM_DMABURST_BASEADDR_SR
  - LL_TIM_DMABURST_BASEADDR_EGR
  - LL_TIM_DMABURST_BASEADDR_CCMR1
  - LL_TIM_DMABURST_BASEADDR_CCMR2
  - LL_TIM_DMABURST_BASEADDR_CCER
  - LL_TIM_DMABURST_BASEADDR_CNT
  - LL_TIM_DMABURST_BASEADDR_PSC
  - LL_TIM_DMABURST_BASEADDR_ARR
  - LL_TIM_DMABURST_BASEADDR_RCR
  - LL_TIM_DMABURST_BASEADDR_CCR1
  - LL_TIM_DMABURST_BASEADDR_CCR2
  - LL_TIM_DMABURST_BASEADDR_CCR3
  - LL_TIM_DMABURST_BASEADDR_CCR4
  - LL_TIM_DMABURST_BASEADDR_BDTR
  - LL_TIM_DMABURST_BASEADDR_CCR5
  - LL_TIM_DMABURST_BASEADDR_CCR6
  - LL_TIM_DMABURST_BASEADDR_CCMR3
  - LL_TIM_DMABURST_BASEADDR_DTR2
  - LL_TIM_DMABURST_BASEADDR_ECR
  - LL_TIM_DMABURST_BASEADDR_TISEL
  - LL_TIM_DMABURST_BASEADDR_AF1
  - LL_TIM_DMABURST_BASEADDR_AF2

- **DMABurstLength:** This parameter can be one of the following values:
  - LL_TIM_DMABURST_LENGTH_1TRANSFER
  - LL_TIM_DMABURST_LENGTH_2TRANSFERS
  - LL_TIM_DMABURST_LENGTH_3TRANSFERS
  - LL_TIM_DMABURST_LENGTH_4TRANSFERS
  - LL_TIM_DMABURST_LENGTH_5TRANSFERS
  - LL_TIM_DMABURST_LENGTH_6TRANSFERS
  - LL_TIM_DMABURST_LENGTH_7TRANSFERS
  - LL_TIM_DMABURST_LENGTH_8TRANSFERS
  - LL_TIM_DMABURST_LENGTH_9TRANSFERS
  - LL_TIM_DMABURST_LENGTH_10TRANSFERS
  - LL_TIM_DMABURST_LENGTH_11TRANSFERS
  - LL_TIM_DMABURST_LENGTH_12TRANSFERS
  - LL_TIM_DMABURST_LENGTH_13TRANSFERS
  - LL_TIM_DMABURST_LENGTH_14TRANSFERS
  - LL_TIM_DMABURST_LENGTH_15TRANSFERS
  - LL_TIM_DMABURST_LENGTH_16TRANSFERS
  - LL_TIM_DMABURST_LENGTH_17TRANSFERS
  - LL_TIM_DMABURST_LENGTH_18TRANSFERS
  - LL_TIM_DMABURST_LENGTH_19TRANSFERS
  - LL_TIM_DMABURST_LENGTH_20TRANSFERS
  - LL_TIM_DMABURST_LENGTH_21TRANSFERS
  - LL_TIM_DMABURST_LENGTH_22TRANSFERS
  - LL_TIM_DMABURST_LENGTH_23TRANSFERS
  - LL_TIM_DMABURST_LENGTH_24TRANSFERS
  - LL_TIM_DMABURST_LENGTH_25TRANSFERS
  - LL_TIM_DMABURST_LENGTH_26TRANSFERS
- **DMABurstSource:** This parameter can be one of the following values:
  - LL_TIM_DMA_UPDATE
  - LL_TIM_DMA_CC1
  - LL_TIM_DMA_CC2
  - LL_TIM_DMA_CC3
  - LL_TIM_DMA_CC4
  - LL_TIM_DMA_COM
  - LL_TIM_DMA_TRIGGER

Return values

- **None:**

Notes

- Macro IS_TIM_DMABURST_INSTANCE(TIMx) can be used to check whether or not a timer instance supports the DMA burst mode.

Reference Manual to LL API cross reference:

- DCR DBL LL_TIM_ConfigDMABurst
- DCR DBA LL_TIM_ConfigDMABurst

LL_TIM_EnableEncoderIndex

Function name

__STATIC_INLINE void LL_TIM_EnableEncoderIndex (TIM_TypeDef * TIMx)

Function description

Enable encoder index.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_INDEX_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an index input.

Reference Manual to LL API cross reference:

- ECR IE LL_TIM_EnableEncoderIndex

LL_TIM_DisableEncoderIndex

Function name

__STATIC_INLINE void LL_TIM_DisableEncoderIndex (TIM_TypeDef * TIMx)

Function description

Disable encoder index.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_INDEX_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an index input.

Reference Manual to LL API cross reference:

- ECR IE LL_TIM_DisableEncoderIndex

LL_TIM_IsEnabledEncoderIndex

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledEncoderIndex (const TIM_TypeDef * TIMx)

Function description

Indicate whether encoder index is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_INDEX_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an index input.

Reference Manual to LL API cross reference:

- ECR IE LL_TIM_IsEnabledEncoderIndex

LL_TIM_SetIndexDirection

Function name

__STATIC_INLINE void LL_TIM_SetIndexDirection (TIM_TypeDef * TIMx, uint32_t IndexDirection)

Function description

Set index direction.

Parameters

- **TIMx:** Timer instance
- **IndexDirection:** This parameter can be one of the following values:
  - LL_TIM_INDEX_UP_DOWN
  - LL_TIM_INDEX_UP
  - LL_TIM_INDEX_DOWN

Return values

- **None:**

Notes

- Macro IS_TIM_INDEX_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an index input.

Reference Manual to LL API cross reference:

- ECR IDIR LL_TIM_SetIndexDirection

LL_TIM_GetIndexDirection

Function name

__STATIC_INLINE uint32_t LL_TIM_GetIndexDirection (const TIM_TypeDef * TIMx)

Function description

Get actual index direction.

Parameters

- **TIMx:** Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_TIM_INDEX_UP_DOWN
  - LL_TIM_INDEX_UP
  - LL_TIM_INDEX_DOWN

Notes

- Macro IS_TIM_INDEX_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an index input.

Reference Manual to LL API cross reference:

- ECR IDIR LL_TIM_GetIndexDirection

LL_TIM_SetIndexblanking

Function name

__STATIC_INLINE void LL_TIM_SetIndexblanking (TIM_TypeDef * TIMx, uint32_t Indexblanking)

Function description

Set index blanking.

Parameters

- **TIMx:** Timer instance
- **Indexblanking:** This parameter can be one of the following values:
  - LL_TIM_INDEX_BLANK_ALWAYS
  - LL_TIM_INDEX_BLANK_TI3
  - LL_TIM_INDEX_BLANK_TI4

Return values

- **None:**

Notes

- Macro IS_TIM_INDEX_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an index input.

Reference Manual to LL API cross reference:

- ECR IBLK LL_TIM_SetIndexblanking

LL_TIM_GetIndexblanking

Function name

__STATIC_INLINE uint32_t LL_TIM_GetIndexblanking (const TIM_TypeDef * TIMx)

Function description

Get actual index blanking.

Parameters

- **TIMx:** Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_TIM_INDEX_BLANK_ALWAYS
  - LL_TIM_INDEX_BLANK_TI3
  - LL_TIM_INDEX_BLANK_TI4

Notes

- Macro IS_TIM_INDEX_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an index input.

Reference Manual to LL API cross reference:

- ECR IBLK LL_TIM_GetIndexblanking

LL_TIM_EnableFirstIndex

Function name

__STATIC_INLINE void LL_TIM_EnableFirstIndex (TIM_TypeDef * TIMx)

Function description

Enable first index.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_INDEX_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an index input.

Reference Manual to LL API cross reference:

- ECR FIDX LL_TIM_EnableFirstIndex

LL_TIM_DisableFirstIndex

Function name

__STATIC_INLINE void LL_TIM_DisableFirstIndex (TIM_TypeDef * TIMx)

Function description

Disable first index.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_INDEX_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an index input.

Reference Manual to LL API cross reference:

- ECR FIDX LL_TIM_DisableFirstIndex

LL_TIM_IsEnabledFirstIndex

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledFirstIndex (const TIM_TypeDef * TIMx)

Function description

Indicates whether first index is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_INDEX_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an index input.

Reference Manual to LL API cross reference:

- ECR FIDX LL_TIM_IsEnabledFirstIndex

LL_TIM_SetIndexPositionning

Function name

__STATIC_INLINE void LL_TIM_SetIndexPositionning (TIM_TypeDef * TIMx, uint32_t IndexPositionning)

Function description

Set index positioning.

Parameters

- **TIMx:** Timer instance
- **IndexPositionning:** This parameter can be one of the following values:
  - LL_TIM_INDEX_POSITION_DOWN_DOWN
  - LL_TIM_INDEX_POSITION_DOWN_UP
  - LL_TIM_INDEX_POSITION_UP_DOWN
  - LL_TIM_INDEX_POSITION_UP_UP
  - LL_TIM_INDEX_POSITION_DOWN
  - LL_TIM_INDEX_POSITION_UP

Return values

- **None:**

Notes

- Macro IS_TIM_INDEX_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an index input.

Reference Manual to LL API cross reference:

- ECR IPOS LL_TIM_SetIndexPositionning

LL_TIM_GetIndexPositionning

Function name

__STATIC_INLINE uint32_t LL_TIM_GetIndexPositionning (const TIM_TypeDef * TIMx)

Function description

Get actual index positioning.

Parameters

- **TIMx:** Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_TIM_INDEX_POSITION_DOWN_DOWN
  - LL_TIM_INDEX_POSITION_DOWN_UP
  - LL_TIM_INDEX_POSITION_UP_DOWN
  - LL_TIM_INDEX_POSITION_UP_UP
  - LL_TIM_INDEX_POSITION_DOWN
  - LL_TIM_INDEX_POSITION_UP

Notes

- Macro IS_TIM_INDEX_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an index input.

Reference Manual to LL API cross reference:

- ECR IPOS LL_TIM_GetIndexPositionning

LL_TIM_ConfigIDX

Function name

__STATIC_INLINE void LL_TIM_ConfigIDX (TIM_TypeDef * TIMx, uint32_t Configuration)

Function description

Configure encoder index.

Parameters

- **TIMx:** Timer instance
- **Configuration:** This parameter must be a combination of all the following values:
  - LL_TIM_INDEX_UP or LL_TIM_INDEX_DOWN or LL_TIM_INDEX_UP_DOWN
  - LL_TIM_INDEX_BLANK_ALWAYS or LL_TIM_INDEX_BLANK_TI3 or LL_TIM_INDEX_BLANK_TI4
  - LL_TIM_INDEX_ALL or LL_TIM_INDEX_FIRST_ONLY
  - LL_TIM_INDEX_POSITION_DOWN_DOWN or ... or LL_TIM_INDEX_POSITION_UP

Return values

- **None:**

Notes

- Macro IS_TIM_INDEX_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an index input.

Reference Manual to LL API cross reference:

- ECR IDIR LL_TIM_ConfigIDX
- ECR IBLK LL_TIM_ConfigIDX
- ECR FIDX LL_TIM_ConfigIDX
- ECR IPOS LL_TIM_ConfigIDX

LL_TIM_SetRemap

Function name

__STATIC_INLINE void LL_TIM_SetRemap (TIM_TypeDef * TIMx, uint32_t Remap)

Function description

Remap TIM inputs (input channel, internal/external triggers).

Parameters

- **TIMx:** Timer instance
- **Remap:** Remap param depends on the TIMx. Description available only in CHM version of the User Manual (not in .pdf). Otherwise see Reference Manual description of TISEL registers.

Return values

- **None:**

Notes

- Macro IS_TIM_REMAP_INSTANCE(TIMx) can be used to check whether or not a some timer inputs can be remapped.

Reference Manual to LL API cross reference:

- TIM3_TISEL TI1SEL LL_TIM_SetRemap
- TIM5_TISEL TI1SEL LL_TIM_SetRemap
- TIM9_TISEL TI1SEL LL_TIM_SetRemap
- TIM10_TISEL TI1SEL LL_TIM_SetRemap
- TIM11_TISEL TI1SEL LL_TIM_SetRemap
- TIM12_TISEL TI1SEL LL_TIM_SetRemap
- TIM13_TISEL TI1SEL LL_TIM_SetRemap
- TIM14_TISEL TI1SEL LL_TIM_SetRemap
- TIM15_TISEL TI1SEL LL_TIM_SetRemap
- TIM16_TISEL TI1SEL LL_TIM_SetRemap
- TIM17_TISEL TI1SEL LL_TIM_SetRemap
-

LL_TIM_ClearFlag_UPDATE

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_UPDATE (TIM_TypeDef * TIMx)

Function description

Clear the update interrupt flag (UIF).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR UIF LL_TIM_ClearFlag_UPDATE

LL_TIM_IsActiveFlag_UPDATE

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_UPDATE (const TIM_TypeDef * TIMx)

Function description

Indicate whether update interrupt flag (UIF) is set (update interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR UIF LL_TIM_IsActiveFlag_UPDATE

LL_TIM_ClearFlag_CC1

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_CC1 (TIM_TypeDef * TIMx)

Function description

Clear the Capture/Compare 1 interrupt flag (CC1F).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR CC1IF LL_TIM_ClearFlag_CC1

LL_TIM_IsActiveFlag_CC1

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC1 (const TIM_TypeDef * TIMx)

Function description

Indicate whether Capture/Compare 1 interrupt flag (CC1F) is set (Capture/Compare 1 interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR CC1IF LL_TIM_IsActiveFlag_CC1

LL_TIM_ClearFlag_CC2

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_CC2 (TIM_TypeDef * TIMx)

Function description

Clear the Capture/Compare 2 interrupt flag (CC2F).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR CC2IF LL_TIM_ClearFlag_CC2

LL_TIM_IsActiveFlag_CC2

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC2 (const TIM_TypeDef * TIMx)

Function description

Indicate whether Capture/Compare 2 interrupt flag (CC2F) is set (Capture/Compare 2 interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR CC2IF LL_TIM_IsActiveFlag_CC2

LL_TIM_ClearFlag_CC3

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_CC3 (TIM_TypeDef * TIMx)

Function description

Clear the Capture/Compare 3 interrupt flag (CC3F).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR CC3IF LL_TIM_ClearFlag_CC3

LL_TIM_IsActiveFlag_CC3

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC3 (const TIM_TypeDef * TIMx)

Function description

Indicate whether Capture/Compare 3 interrupt flag (CC3F) is set (Capture/Compare 3 interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR CC3IF LL_TIM_IsActiveFlag_CC3

LL_TIM_ClearFlag_CC4

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_CC4 (TIM_TypeDef * TIMx)

Function description

Clear the Capture/Compare 4 interrupt flag (CC4F).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR CC4IF LL_TIM_ClearFlag_CC4

LL_TIM_IsActiveFlag_CC4

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC4 (const TIM_TypeDef * TIMx)

Function description

Indicate whether Capture/Compare 4 interrupt flag (CC4F) is set (Capture/Compare 4 interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR CC4IF LL_TIM_IsActiveFlag_CC4

LL_TIM_ClearFlag_CC5

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_CC5 (TIM_TypeDef * TIMx)

Function description

Clear the Capture/Compare 5 interrupt flag (CC5F).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR CC5IF LL_TIM_ClearFlag_CC5

LL_TIM_IsActiveFlag_CC5

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC5 (const TIM_TypeDef * TIMx)

Function description

Indicate whether Capture/Compare 5 interrupt flag (CC5F) is set (Capture/Compare 5 interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR CC5IF LL_TIM_IsActiveFlag_CC5

LL_TIM_ClearFlag_CC6

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_CC6 (TIM_TypeDef * TIMx)

Function description

Clear the Capture/Compare 6 interrupt flag (CC6F).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR CC6IF LL_TIM_ClearFlag_CC6

LL_TIM_IsActiveFlag_CC6

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC6 (const TIM_TypeDef * TIMx)

Function description

Indicate whether Capture/Compare 6 interrupt flag (CC6F) is set (Capture/Compare 6 interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR CC6IF LL_TIM_IsActiveFlag_CC6

LL_TIM_ClearFlag_COM

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_COM (TIM_TypeDef * TIMx)

Function description

Clear the commutation interrupt flag (COMIF).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR COMIF LL_TIM_ClearFlag_COM

LL_TIM_IsActiveFlag_COM

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_COM (const TIM_TypeDef * TIMx)

Function description

Indicate whether commutation interrupt flag (COMIF) is set (commutation interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR COMIF LL_TIM_IsActiveFlag_COM

LL_TIM_ClearFlag_TRIG

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_TRIG (TIM_TypeDef * TIMx)

Function description

Clear the trigger interrupt flag (TIF).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR TIF LL_TIM_ClearFlag_TRIG

LL_TIM_IsActiveFlag_TRIG

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_TRIG (const TIM_TypeDef * TIMx)

Function description

Indicate whether trigger interrupt flag (TIF) is set (trigger interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR TIF LL_TIM_IsActiveFlag_TRIG

LL_TIM_ClearFlag_BRK

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_BRK (TIM_TypeDef * TIMx)

Function description

Clear the break interrupt flag (BIF).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR BIF LL_TIM_ClearFlag_BRK

LL_TIM_IsActiveFlag_BRK

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_BRK (const TIM_TypeDef * TIMx)

Function description

Indicate whether break interrupt flag (BIF) is set (break interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR BIF LL_TIM_IsActiveFlag_BRK

LL_TIM_ClearFlag_BRK2

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_BRK2 (TIM_TypeDef * TIMx)

Function description

Clear the break 2 interrupt flag (B2IF).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR B2IF LL_TIM_ClearFlag_BRK2

LL_TIM_IsActiveFlag_BRK2

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_BRK2 (const TIM_TypeDef * TIMx)

Function description

Indicate whether break 2 interrupt flag (B2IF) is set (break 2 interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR B2IF LL_TIM_IsActiveFlag_BRK2

LL_TIM_ClearFlag_CC1OVR

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_CC1OVR (TIM_TypeDef * TIMx)

Function description

Clear the Capture/Compare 1 over-capture interrupt flag (CC1OF).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR CC1OF LL_TIM_ClearFlag_CC1OVR

LL_TIM_IsActiveFlag_CC1OVR

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC1OVR (const TIM_TypeDef * TIMx)

Function description

Indicate whether Capture/Compare 1 over-capture interrupt flag (CC1OF) is set (Capture/Compare 1 interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR CC1OF LL_TIM_IsActiveFlag_CC1OVR

LL_TIM_ClearFlag_CC2OVR

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_CC2OVR (TIM_TypeDef * TIMx)

Function description

Clear the Capture/Compare 2 over-capture interrupt flag (CC2OF).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR CC2OF LL_TIM_ClearFlag_CC2OVR

LL_TIM_IsActiveFlag_CC2OVR

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC2OVR (const TIM_TypeDef * TIMx)

Function description

Indicate whether Capture/Compare 2 over-capture interrupt flag (CC2OF) is set (Capture/Compare 2 over-capture interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR CC2OF LL_TIM_IsActiveFlag_CC2OVR

LL_TIM_ClearFlag_CC3OVR

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_CC3OVR (TIM_TypeDef * TIMx)

Function description

Clear the Capture/Compare 3 over-capture interrupt flag (CC3OF).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR CC3OF LL_TIM_ClearFlag_CC3OVR

LL_TIM_IsActiveFlag_CC3OVR

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC3OVR (const TIM_TypeDef * TIMx)

Function description

Indicate whether Capture/Compare 3 over-capture interrupt flag (CC3OF) is set (Capture/Compare 3 over-capture interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR CC3OF LL_TIM_IsActiveFlag_CC3OVR

LL_TIM_ClearFlag_CC4OVR

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_CC4OVR (TIM_TypeDef * TIMx)

Function description

Clear the Capture/Compare 4 over-capture interrupt flag (CC4OF).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR CC4OF LL_TIM_ClearFlag_CC4OVR

LL_TIM_IsActiveFlag_CC4OVR

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC4OVR (const TIM_TypeDef * TIMx)

Function description

Indicate whether Capture/Compare 4 over-capture interrupt flag (CC4OF) is set (Capture/Compare 4 over-capture interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR CC4OF LL_TIM_IsActiveFlag_CC4OVR

LL_TIM_ClearFlag_SYSBRK

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_SYSBRK (TIM_TypeDef * TIMx)

Function description

Clear the system break interrupt flag (SBIF).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR SBIF LL_TIM_ClearFlag_SYSBRK

LL_TIM_IsActiveFlag_SYSBRK

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_SYSBRK (const TIM_TypeDef * TIMx)

Function description

Indicate whether system break interrupt flag (SBIF) is set (system break interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR SBIF LL_TIM_IsActiveFlag_SYSBRK

LL_TIM_ClearFlag_TERR

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_TERR (TIM_TypeDef * TIMx)

Function description

Clear the transition error interrupt flag (TERRF).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_ENCODER_ERROR_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder error management.

Reference Manual to LL API cross reference:

- SR TERRF LL_TIM_ClearFlag_TERR

LL_TIM_IsActiveFlag_TERR

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_TERR (const TIM_TypeDef * TIMx)

Function description

Indicate whether transition error interrupt flag (TERRF) is set (transition error interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_ENCODER_ERROR_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder error management.

Reference Manual to LL API cross reference:

- SR TERRF LL_TIM_IsActiveFlag_TERR

LL_TIM_ClearFlag_IERR

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_IERR (TIM_TypeDef * TIMx)

Function description

Clear the index error interrupt flag (IERRF).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_ENCODER_ERROR_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder error management.

Reference Manual to LL API cross reference:

- SR IERRF LL_TIM_ClearFlag_IERR

LL_TIM_IsActiveFlag_IERR

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_IERR (const TIM_TypeDef * TIMx)

Function description

Indicate whether index error interrupt flag (IERRF) is set (index error interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_ENCODER_ERROR_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder error management.

Reference Manual to LL API cross reference:

- SR IERRF LL_TIM_IsActiveFlag_IERR

LL_TIM_ClearFlag_DIR

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_DIR (TIM_TypeDef * TIMx)

Function description

Clear the direction change interrupt flag (DIRF).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_FUNCTINONAL_ENCODER_INTERRUPT_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder interrupt management.

Reference Manual to LL API cross reference:

- SR DIRF LL_TIM_ClearFlag_DIR

LL_TIM_IsActiveFlag_DIR

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_DIR (const TIM_TypeDef * TIMx)

Function description

Indicate whether direction change interrupt flag (DIRF) is set (direction change interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_FUNCTINONAL_ENCODER_INTERRUPT_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder interrupt management.

Reference Manual to LL API cross reference:

- SR DIRF LL_TIM_IsActiveFlag_DIR

LL_TIM_ClearFlag_IDX

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_IDX (TIM_TypeDef * TIMx)

Function description

Clear the index interrupt flag (IDXF).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_FUNCTINONAL_ENCODER_INTERRUPT_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder interrupt management.

Reference Manual to LL API cross reference:

- SR IDXF LL_TIM_ClearFlag_IDX

LL_TIM_IsActiveFlag_IDX

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_IDX (const TIM_TypeDef * TIMx)

Function description

Indicate whether index interrupt flag (IDXF) is set (index interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_FUNCTINONAL_ENCODER_INTERRUPT_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder interrupt management.

Reference Manual to LL API cross reference:

- SR IDXF LL_TIM_IsActiveFlag_IDX

LL_TIM_EnableIT_UPDATE

Function name

__STATIC_INLINE void LL_TIM_EnableIT_UPDATE (TIM_TypeDef * TIMx)

Function description

Enable update interrupt (UIE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER UIE LL_TIM_EnableIT_UPDATE

LL_TIM_DisableIT_UPDATE

Function name

__STATIC_INLINE void LL_TIM_DisableIT_UPDATE (TIM_TypeDef * TIMx)

Function description

Disable update interrupt (UIE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER UIE LL_TIM_DisableIT_UPDATE

LL_TIM_IsEnabledIT_UPDATE

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_UPDATE (const TIM_TypeDef * TIMx)

Function description

Indicates whether the update interrupt (UIE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER UIE LL_TIM_IsEnabledIT_UPDATE

LL_TIM_EnableIT_CC1

Function name

__STATIC_INLINE void LL_TIM_EnableIT_CC1 (TIM_TypeDef * TIMx)

Function description

Enable capture/compare 1 interrupt (CC1IE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC1IE LL_TIM_EnableIT_CC1

LL_TIM_DisableIT_CC1

Function name

__STATIC_INLINE void LL_TIM_DisableIT_CC1 (TIM_TypeDef * TIMx)

Function description

Disable capture/compare 1 interrupt (CC1IE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC1IE LL_TIM_DisableIT_CC1

LL_TIM_IsEnabledIT_CC1

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_CC1 (const TIM_TypeDef * TIMx)

Function description

Indicates whether the capture/compare 1 interrupt (CC1IE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER CC1IE LL_TIM_IsEnabledIT_CC1

LL_TIM_EnableIT_CC2

Function name

__STATIC_INLINE void LL_TIM_EnableIT_CC2 (TIM_TypeDef * TIMx)

Function description

Enable capture/compare 2 interrupt (CC2IE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC2IE LL_TIM_EnableIT_CC2

LL_TIM_DisableIT_CC2

Function name

__STATIC_INLINE void LL_TIM_DisableIT_CC2 (TIM_TypeDef * TIMx)

Function description

Disable capture/compare 2 interrupt (CC2IE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC2IE LL_TIM_DisableIT_CC2

LL_TIM_IsEnabledIT_CC2

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_CC2 (const TIM_TypeDef * TIMx)

Function description

Indicates whether the capture/compare 2 interrupt (CC2IE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER CC2IE LL_TIM_IsEnabledIT_CC2

LL_TIM_EnableIT_CC3

Function name

__STATIC_INLINE void LL_TIM_EnableIT_CC3 (TIM_TypeDef * TIMx)

Function description

Enable capture/compare 3 interrupt (CC3IE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC3IE LL_TIM_EnableIT_CC3

LL_TIM_DisableIT_CC3

Function name

__STATIC_INLINE void LL_TIM_DisableIT_CC3 (TIM_TypeDef * TIMx)

Function description

Disable capture/compare 3 interrupt (CC3IE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC3IE LL_TIM_DisableIT_CC3

LL_TIM_IsEnabledIT_CC3

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_CC3 (const TIM_TypeDef * TIMx)

Function description

Indicates whether the capture/compare 3 interrupt (CC3IE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER CC3IE LL_TIM_IsEnabledIT_CC3

LL_TIM_EnableIT_CC4

Function name

__STATIC_INLINE void LL_TIM_EnableIT_CC4 (TIM_TypeDef * TIMx)

Function description

Enable capture/compare 4 interrupt (CC4IE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC4IE LL_TIM_EnableIT_CC4

LL_TIM_DisableIT_CC4

Function name

__STATIC_INLINE void LL_TIM_DisableIT_CC4 (TIM_TypeDef * TIMx)

Function description

Disable capture/compare 4 interrupt (CC4IE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC4IE LL_TIM_DisableIT_CC4

LL_TIM_IsEnabledIT_CC4

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_CC4 (const TIM_TypeDef * TIMx)

Function description

Indicates whether the capture/compare 4 interrupt (CC4IE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER CC4IE LL_TIM_IsEnabledIT_CC4

LL_TIM_EnableIT_COM

Function name

__STATIC_INLINE void LL_TIM_EnableIT_COM (TIM_TypeDef * TIMx)

Function description

Enable commutation interrupt (COMIE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER COMIE LL_TIM_EnableIT_COM

LL_TIM_DisableIT_COM

Function name

__STATIC_INLINE void LL_TIM_DisableIT_COM (TIM_TypeDef * TIMx)

Function description

Disable commutation interrupt (COMIE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER COMIE LL_TIM_DisableIT_COM

LL_TIM_IsEnabledIT_COM

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_COM (const TIM_TypeDef * TIMx)

Function description

Indicates whether the commutation interrupt (COMIE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER COMIE LL_TIM_IsEnabledIT_COM

LL_TIM_EnableIT_TRIG

Function name

__STATIC_INLINE void LL_TIM_EnableIT_TRIG (TIM_TypeDef * TIMx)

Function description

Enable trigger interrupt (TIE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER TIE LL_TIM_EnableIT_TRIG

LL_TIM_DisableIT_TRIG

Function name

__STATIC_INLINE void LL_TIM_DisableIT_TRIG (TIM_TypeDef * TIMx)

Function description

Disable trigger interrupt (TIE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER TIE LL_TIM_DisableIT_TRIG

LL_TIM_IsEnabledIT_TRIG

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_TRIG (const TIM_TypeDef * TIMx)

Function description

Indicates whether the trigger interrupt (TIE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER TIE LL_TIM_IsEnabledIT_TRIG

LL_TIM_EnableIT_BRK

Function name

__STATIC_INLINE void LL_TIM_EnableIT_BRK (TIM_TypeDef * TIMx)

Function description

Enable break interrupt (BIE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER BIE LL_TIM_EnableIT_BRK

LL_TIM_DisableIT_BRK

Function name

__STATIC_INLINE void LL_TIM_DisableIT_BRK (TIM_TypeDef * TIMx)

Function description

Disable break interrupt (BIE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER BIE LL_TIM_DisableIT_BRK

LL_TIM_IsEnabledIT_BRK

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_BRK (const TIM_TypeDef * TIMx)

Function description

Indicates whether the break interrupt (BIE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER BIE LL_TIM_IsEnabledIT_BRK

LL_TIM_EnableIT_TERR

Function name

__STATIC_INLINE void LL_TIM_EnableIT_TERR (TIM_TypeDef * TIMx)

Function description

Enable transition error interrupt (TERRIE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_ENCODER_ERROR_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder error management.

Reference Manual to LL API cross reference:

- DIER TERRIE LL_TIM_EnableIT_TERR

LL_TIM_DisableIT_TERR

Function name

__STATIC_INLINE void LL_TIM_DisableIT_TERR (TIM_TypeDef * TIMx)

Function description

Disable transition error interrupt (TERRIE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_ENCODER_ERROR_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder error management.

Reference Manual to LL API cross reference:

- DIER TERRIE LL_TIM_DisableIT_TERR

LL_TIM_IsEnabledIT_TERR

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_TERR (const TIM_TypeDef * TIMx)

Function description

Indicates whether the transition error interrupt (TERRIE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_ENCODER_ERROR_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder error management.

Reference Manual to LL API cross reference:

- DIER TERRIE LL_TIM_IsEnabledIT_TERR

LL_TIM_EnableIT_IERR

Function name

__STATIC_INLINE void LL_TIM_EnableIT_IERR (TIM_TypeDef * TIMx)

Function description

Enable index error interrupt (IERRIE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_ENCODER_ERROR_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder error management.

Reference Manual to LL API cross reference:

- DIER IERRIE LL_TIM_EnableIT_IERR

LL_TIM_DisableIT_IERR

Function name

__STATIC_INLINE void LL_TIM_DisableIT_IERR (TIM_TypeDef * TIMx)

Function description

Disable index error interrupt (IERRIE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_ENCODER_ERROR_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder error management.

Reference Manual to LL API cross reference:

- DIER IERRIE LL_TIM_DisableIT_IERR

LL_TIM_IsEnabledIT_IERR

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_IERR (const TIM_TypeDef * TIMx)

Function description

Indicates whether the index error interrupt (IERRIE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_ENCODER_ERROR_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder error management.

Reference Manual to LL API cross reference:

- DIER IERRIE LL_TIM_IsEnabledIT_IERR

LL_TIM_EnableIT_DIR

Function name

__STATIC_INLINE void LL_TIM_EnableIT_DIR (TIM_TypeDef * TIMx)

Function description

Enable direction change interrupt (DIRIE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_FUNCTINONAL_ENCODER_INTERRUPT_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder interrupt management.

Reference Manual to LL API cross reference:

- DIER DIRIE LL_TIM_EnableIT_DIR

LL_TIM_DisableIT_DIR

Function name

__STATIC_INLINE void LL_TIM_DisableIT_DIR (TIM_TypeDef * TIMx)

Function description

Disable direction change interrupt (DIRIE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_FUNCTINONAL_ENCODER_INTERRUPT_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder interrupt management.

Reference Manual to LL API cross reference:

- DIER DIRIE LL_TIM_DisableIT_DIR

LL_TIM_IsEnabledIT_DIR

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_DIR (const TIM_TypeDef * TIMx)

Function description

Indicates whether the direction change interrupt (DIRIE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_FUNCTINONAL_ENCODER_INTERRUPT_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder interrupt management.

Reference Manual to LL API cross reference:

- DIER DIRIE LL_TIM_IsEnabledIT_DIR

LL_TIM_EnableIT_IDX

Function name

__STATIC_INLINE void LL_TIM_EnableIT_IDX (TIM_TypeDef * TIMx)

Function description

Enable index interrupt (IDXIE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_FUNCTINONAL_ENCODER_INTERRUPT_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder interrupt management.

Reference Manual to LL API cross reference:

- DIER IDXIE LL_TIM_EnableIT_IDX

LL_TIM_DisableIT_IDX

Function name

__STATIC_INLINE void LL_TIM_DisableIT_IDX (TIM_TypeDef * TIMx)

Function description

Disable index interrupt (IDXIE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_FUNCTINONAL_ENCODER_INTERRUPT_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder interrupt management.

Reference Manual to LL API cross reference:

- DIER IDXIE LL_TIM_DisableIT_IDX

LL_TIM_IsEnabledIT_IDX

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_IDX (const TIM_TypeDef * TIMx)

Function description

Indicates whether the index interrupt (IDXIE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_FUNCTINONAL_ENCODER_INTERRUPT_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder interrupt management.

Reference Manual to LL API cross reference:

- DIER IDXIE LL_TIM_IsEnabledIT_IDX

LL_TIM_EnableDMAReq_UPDATE

Function name

__STATIC_INLINE void LL_TIM_EnableDMAReq_UPDATE (TIM_TypeDef * TIMx)

Function description

Enable update DMA request (UDE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER UDE LL_TIM_EnableDMAReq_UPDATE

LL_TIM_DisableDMAReq_UPDATE

Function name

__STATIC_INLINE void LL_TIM_DisableDMAReq_UPDATE (TIM_TypeDef * TIMx)

Function description

Disable update DMA request (UDE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER UDE LL_TIM_DisableDMAReq_UPDATE

LL_TIM_IsEnabledDMAReq_UPDATE

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledDMAReq_UPDATE (const TIM_TypeDef * TIMx)

Function description

Indicates whether the update DMA request (UDE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER UDE LL_TIM_IsEnabledDMAReq_UPDATE

LL_TIM_EnableDMAReq_CC1

Function name

__STATIC_INLINE void LL_TIM_EnableDMAReq_CC1 (TIM_TypeDef * TIMx)

Function description

Enable capture/compare 1 DMA request (CC1DE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC1DE LL_TIM_EnableDMAReq_CC1

LL_TIM_DisableDMAReq_CC1

Function name

__STATIC_INLINE void LL_TIM_DisableDMAReq_CC1 (TIM_TypeDef * TIMx)

Function description

Disable capture/compare 1 DMA request (CC1DE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC1DE LL_TIM_DisableDMAReq_CC1

LL_TIM_IsEnabledDMAReq_CC1

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledDMAReq_CC1 (const TIM_TypeDef * TIMx)

Function description

Indicates whether the capture/compare 1 DMA request (CC1DE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER CC1DE LL_TIM_IsEnabledDMAReq_CC1

LL_TIM_EnableDMAReq_CC2

Function name

__STATIC_INLINE void LL_TIM_EnableDMAReq_CC2 (TIM_TypeDef * TIMx)

Function description

Enable capture/compare 2 DMA request (CC2DE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC2DE LL_TIM_EnableDMAReq_CC2

LL_TIM_DisableDMAReq_CC2

Function name

__STATIC_INLINE void LL_TIM_DisableDMAReq_CC2 (TIM_TypeDef * TIMx)

Function description

Disable capture/compare 2 DMA request (CC2DE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC2DE LL_TIM_DisableDMAReq_CC2

LL_TIM_IsEnabledDMAReq_CC2

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledDMAReq_CC2 (const TIM_TypeDef * TIMx)

Function description

Indicates whether the capture/compare 2 DMA request (CC2DE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER CC2DE LL_TIM_IsEnabledDMAReq_CC2

LL_TIM_EnableDMAReq_CC3

Function name

__STATIC_INLINE void LL_TIM_EnableDMAReq_CC3 (TIM_TypeDef * TIMx)

Function description

Enable capture/compare 3 DMA request (CC3DE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC3DE LL_TIM_EnableDMAReq_CC3

LL_TIM_DisableDMAReq_CC3

Function name

__STATIC_INLINE void LL_TIM_DisableDMAReq_CC3 (TIM_TypeDef * TIMx)

Function description

Disable capture/compare 3 DMA request (CC3DE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC3DE LL_TIM_DisableDMAReq_CC3

LL_TIM_IsEnabledDMAReq_CC3

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledDMAReq_CC3 (const TIM_TypeDef * TIMx)

Function description

Indicates whether the capture/compare 3 DMA request (CC3DE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER CC3DE LL_TIM_IsEnabledDMAReq_CC3

LL_TIM_EnableDMAReq_CC4

Function name

__STATIC_INLINE void LL_TIM_EnableDMAReq_CC4 (TIM_TypeDef * TIMx)

Function description

Enable capture/compare 4 DMA request (CC4DE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC4DE LL_TIM_EnableDMAReq_CC4

LL_TIM_DisableDMAReq_CC4

Function name

__STATIC_INLINE void LL_TIM_DisableDMAReq_CC4 (TIM_TypeDef * TIMx)

Function description

Disable capture/compare 4 DMA request (CC4DE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC4DE LL_TIM_DisableDMAReq_CC4

LL_TIM_IsEnabledDMAReq_CC4

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledDMAReq_CC4 (const TIM_TypeDef * TIMx)

Function description

Indicates whether the capture/compare 4 DMA request (CC4DE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER CC4DE LL_TIM_IsEnabledDMAReq_CC4

LL_TIM_EnableDMAReq_COM

Function name

__STATIC_INLINE void LL_TIM_EnableDMAReq_COM (TIM_TypeDef * TIMx)

Function description

Enable commutation DMA request (COMDE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER COMDE LL_TIM_EnableDMAReq_COM

LL_TIM_DisableDMAReq_COM

Function name

__STATIC_INLINE void LL_TIM_DisableDMAReq_COM (TIM_TypeDef * TIMx)

Function description

Disable commutation DMA request (COMDE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER COMDE LL_TIM_DisableDMAReq_COM

LL_TIM_IsEnabledDMAReq_COM

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledDMAReq_COM (const TIM_TypeDef * TIMx)

Function description

Indicates whether the commutation DMA request (COMDE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER COMDE LL_TIM_IsEnabledDMAReq_COM

LL_TIM_EnableDMAReq_TRIG

Function name

__STATIC_INLINE void LL_TIM_EnableDMAReq_TRIG (TIM_TypeDef * TIMx)

Function description

Enable trigger interrupt (TDE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER TDE LL_TIM_EnableDMAReq_TRIG

LL_TIM_DisableDMAReq_TRIG

Function name

__STATIC_INLINE void LL_TIM_DisableDMAReq_TRIG (TIM_TypeDef * TIMx)

Function description

Disable trigger interrupt (TDE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER TDE LL_TIM_DisableDMAReq_TRIG

LL_TIM_IsEnabledDMAReq_TRIG

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledDMAReq_TRIG (const TIM_TypeDef * TIMx)

Function description

Indicates whether the trigger interrupt (TDE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER TDE LL_TIM_IsEnabledDMAReq_TRIG

LL_TIM_GenerateEvent_UPDATE

Function name

__STATIC_INLINE void LL_TIM_GenerateEvent_UPDATE (TIM_TypeDef * TIMx)

Function description

Generate an update event.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- EGR UG LL_TIM_GenerateEvent_UPDATE

LL_TIM_GenerateEvent_CC1

Function name

__STATIC_INLINE void LL_TIM_GenerateEvent_CC1 (TIM_TypeDef * TIMx)

Function description

Generate Capture/Compare 1 event.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- EGR CC1G LL_TIM_GenerateEvent_CC1

LL_TIM_GenerateEvent_CC2

Function name

__STATIC_INLINE void LL_TIM_GenerateEvent_CC2 (TIM_TypeDef * TIMx)

Function description

Generate Capture/Compare 2 event.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- EGR CC2G LL_TIM_GenerateEvent_CC2

LL_TIM_GenerateEvent_CC3

Function name

__STATIC_INLINE void LL_TIM_GenerateEvent_CC3 (TIM_TypeDef * TIMx)

Function description

Generate Capture/Compare 3 event.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- EGR CC3G LL_TIM_GenerateEvent_CC3

LL_TIM_GenerateEvent_CC4

Function name

__STATIC_INLINE void LL_TIM_GenerateEvent_CC4 (TIM_TypeDef * TIMx)

Function description

Generate Capture/Compare 4 event.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- EGR CC4G LL_TIM_GenerateEvent_CC4

LL_TIM_GenerateEvent_COM

Function name

__STATIC_INLINE void LL_TIM_GenerateEvent_COM (TIM_TypeDef * TIMx)

Function description

Generate commutation event.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- EGR COMG LL_TIM_GenerateEvent_COM

LL_TIM_GenerateEvent_TRIG

Function name

__STATIC_INLINE void LL_TIM_GenerateEvent_TRIG (TIM_TypeDef * TIMx)

Function description

Generate trigger event.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- EGR TG LL_TIM_GenerateEvent_TRIG

LL_TIM_GenerateEvent_BRK

Function name

__STATIC_INLINE void LL_TIM_GenerateEvent_BRK (TIM_TypeDef * TIMx)

Function description

Generate break event.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- EGR BG LL_TIM_GenerateEvent_BRK

LL_TIM_GenerateEvent_BRK2

Function name

__STATIC_INLINE void LL_TIM_GenerateEvent_BRK2 (TIM_TypeDef * TIMx)

Function description

Generate break 2 event.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- EGR B2G LL_TIM_GenerateEvent_BRK2

LL_TIM_DeInit

Function name

ErrorStatus LL_TIM_DeInit (const TIM_TypeDef * TIMx)

Function description

Set TIMx registers to their reset values.

Parameters

- **TIMx:** Timer instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: TIMx registers are de-initialized
  - ERROR: invalid TIMx instance

LL_TIM_StructInit

Function name

void LL_TIM_StructInit (LL_TIM_InitTypeDef * TIM_InitStruct)

Function description

Set the fields of the time base unit configuration data structure to their default values.

Parameters

- **TIM_InitStruct:** pointer to a LL_TIM_InitTypeDef structure (time base unit configuration data structure)

Return values

- **None:**

LL_TIM_Init

Function name

ErrorStatus LL_TIM_Init (TIM_TypeDef * TIMx, const LL_TIM_InitTypeDef * TIM_InitStruct)

Function description

Configure the TIMx time base unit.

Parameters

- **TIMx:** Timer Instance
- **TIM_InitStruct:** pointer to a LL_TIM_InitTypeDef structure (TIMx time base unit configuration data structure)

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: TIMx registers are de-initialized
  - ERROR: not applicable

LL_TIM_OC_StructInit

Function name

void LL_TIM_OC_StructInit (LL_TIM_OC_InitTypeDef * TIM_OC_InitStruct)

Function description

Set the fields of the TIMx output channel configuration data structure to their default values.

Parameters

- **TIM_OC_InitStruct:** pointer to a LL_TIM_OC_InitTypeDef structure (the output channel configuration data structure)

Return values

- **None:**

LL_TIM_OC_Init

Function name

ErrorStatus LL_TIM_OC_Init (TIM_TypeDef * TIMx, uint32_t Channel, const LL_TIM_OC_InitTypeDef * TIM_OC_InitStruct)

Function description

Configure the TIMx output channel.

Parameters

- **TIMx:** Timer Instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6
- **TIM_OC_InitStruct:** pointer to a LL_TIM_OC_InitTypeDef structure (TIMx output channel configuration data structure)

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: TIMx output channel is initialized
  - ERROR: TIMx output channel is not initialized

LL_TIM_IC_StructInit

Function name

void LL_TIM_IC_StructInit (LL_TIM_IC_InitTypeDef * TIM_ICInitStruct)

Function description

Set the fields of the TIMx input channel configuration data structure to their default values.

Parameters

- **TIM_ICInitStruct:** pointer to a LL_TIM_IC_InitTypeDef structure (the input channel configuration data structure)

Return values

- **None:**

LL_TIM_IC_Init

Function name

ErrorStatus LL_TIM_IC_Init (TIM_TypeDef * TIMx, uint32_t Channel, const LL_TIM_IC_InitTypeDef * TIM_IC_InitStruct)

Function description

Configure the TIMx input channel.

Parameters

- **TIMx:** Timer Instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
- **TIM_IC_InitStruct:** pointer to a LL_TIM_IC_InitTypeDef structure (TIMx input channel configuration data structure)

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: TIMx output channel is initialized
  - ERROR: TIMx output channel is not initialized

LL_TIM_ENCODER_StructInit

Function name

void LL_TIM_ENCODER_StructInit (LL_TIM_ENCODER_InitTypeDef * TIM_EncoderInitStruct)

Function description

Fills each TIM_EncoderInitStruct field with its default value.

Parameters

- **TIM_EncoderInitStruct:** pointer to a LL_TIM_ENCODER_InitTypeDef structure (encoder interface configuration data structure)

Return values

- **None:**

LL_TIM_ENCODER_Init

Function name

ErrorStatus LL_TIM_ENCODER_Init (TIM_TypeDef * TIMx, const LL_TIM_ENCODER_InitTypeDef * TIM_EncoderInitStruct)

Function description

Configure the encoder interface of the timer instance.

Parameters

- **TIMx:** Timer Instance
- **TIM_EncoderInitStruct:** pointer to a LL_TIM_ENCODER_InitTypeDef structure (TIMx encoder interface configuration data structure)

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: TIMx registers are de-initialized
  - ERROR: not applicable

LL_TIM_HALLSENSOR_StructInit

Function name

void LL_TIM_HALLSENSOR_StructInit (LL_TIM_HALLSENSOR_InitTypeDef * TIM_HallSensorInitStruct)

Function description

Set the fields of the TIMx Hall sensor interface configuration data structure to their default values.

Parameters

- **TIM_HallSensorInitStruct:** pointer to a LL_TIM_HALLSENSOR_InitTypeDef structure (HALL sensor interface configuration data structure)

Return values

- **None:**

LL_TIM_HALLSENSOR_Init

Function name

ErrorStatus LL_TIM_HALLSENSOR_Init (TIM_TypeDef * TIMx, const LL_TIM_HALLSENSOR_InitTypeDef * TIM_HallSensorInitStruct)

Function description

Configure the Hall sensor interface of the timer instance.

Parameters

- **TIMx:** Timer Instance
- **TIM_HallSensorInitStruct:** pointer to a LL_TIM_HALLSENSOR_InitTypeDef structure (TIMx HALL sensor interface configuration data structure)

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: TIMx registers are de-initialized
  - ERROR: not applicable

Notes

- TIMx CH1, CH2 and CH3 inputs connected through a XOR to the TI1 input channel
- TIMx slave mode controller is configured in reset mode. Selected internal trigger is TI1F_ED.
- Channel 1 is configured as input, IC1 is mapped on TRC.
- Captured value stored in TIMx_CCR1 correspond to the time elapsed between 2 changes on the inputs. It gives information about motor speed.
- Channel 2 is configured in output PWM 2 mode.
- Compare value stored in TIMx_CCR2 corresponds to the commutation delay.
- OC2REF is selected as trigger output on TRGO.
- LL_TIM_IC_POLARITY_BOTHEDGE must not be used for TI1 when it is used when TIMx operates in Hall sensor interface mode.

LL_TIM_BDTR_StructInit

Function name

void LL_TIM_BDTR_StructInit (LL_TIM_BDTR_InitTypeDef * TIM_BDTRInitStruct)

Function description

Set the fields of the Break and Dead Time configuration data structure to their default values.

Parameters

- **TIM_BDTRInitStruct:** pointer to a LL_TIM_BDTR_InitTypeDef structure (Break and Dead Time configuration data structure)

Return values

- **None:**

LL_TIM_BDTR_Init

Function name

ErrorStatus LL_TIM_BDTR_Init (TIM_TypeDef * TIMx, const LL_TIM_BDTR_InitTypeDef * TIM_BDTRInitStruct)

Function description

Configure the Break and Dead Time feature of the timer instance.

Parameters

- **TIMx:** Timer Instance
- **TIM_BDTRInitStruct:** pointer to a LL_TIM_BDTR_InitTypeDef structure (Break and Dead Time configuration data structure)

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: Break and Dead Time is initialized
  - ERROR: not applicable

Notes

- As the bits BK2P, BK2E, BK2F[3:0], BKF[3:0], AOE, BKP, BKE, OSSI, OSSR and DTG[7:0] can be write-locked depending on the LOCK configuration, it can be necessary to configure all of them during the first write access to the TIMx_BDTR register.
- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.
- Macro IS_TIM_BKIN2_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a second break input.

106.3 TIM Firmware driver defines

The following section lists the various define and macros of the module.

106.3.1 TIM

TIM

Active Input Selection

LL_TIM_ACTIVEINPUT_DIRECTTI

ICx is mapped on TIx

LL_TIM_ACTIVEINPUT_INDIRECTTI

ICx is mapped on TIy

LL_TIM_ACTIVEINPUT_TRC

ICx is mapped on TRC

Automatic output enable

LL_TIM_AUTOMATICOUTPUT_DISABLE

MOE can be set only by software

LL_TIM_AUTOMATICOUTPUT_ENABLE

MOE can be set by software or automatically at the next update event

BKIN POLARITY

LL_TIM_BKIN_POLARITY_LOW

BRK BKIN input is active low

LL_TIM_BKIN_POLARITY_HIGH

BRK BKIN input is active high

BKIN SOURCE

LL_TIM_BKIN_SOURCE_BKIN

BKIN input from AF controller

LL_TIM_BKIN_SOURCE_MDF1

internal signal: MDF1 output

BREAK2 AF MODE

LL_TIM_BREAK2_AFMODE_INPUT

Break2 input BRK2 in input mode

LL_TIM_BREAK2_AFMODE_BIDIRECTIONAL

Break2 input BRK2 in bidirectional mode

Break2 Enable

LL_TIM_BREAK2_DISABLE

Break2 function disabled

LL_TIM_BREAK2_ENABLE

Break2 function enabled

BREAK2 FILTER

LL_TIM_BREAK2_FILTER_FDIV1

No filter, BRK acts asynchronously

LL_TIM_BREAK2_FILTER_FDIV1_N2

fSAMPLING=fCK_INT, N=2

LL_TIM_BREAK2_FILTER_FDIV1_N4

fSAMPLING=fCK_INT, N=4

LL_TIM_BREAK2_FILTER_FDIV1_N8

fSAMPLING=fCK_INT, N=8

LL_TIM_BREAK2_FILTER_FDIV2_N6

fSAMPLING=fDTS/2, N=6

LL_TIM_BREAK2_FILTER_FDIV2_N8

fSAMPLING=fDTS/2, N=8

LL_TIM_BREAK2_FILTER_FDIV4_N6

fSAMPLING=fDTS/4, N=6

LL_TIM_BREAK2_FILTER_FDIV4_N8

fSAMPLING=fDTS/4, N=8

LL_TIM_BREAK2_FILTER_FDIV8_N6

fSAMPLING=fDTS/8, N=6

LL_TIM_BREAK2_FILTER_FDIV8_N8

fSAMPLING=fDTS/8, N=8

LL_TIM_BREAK2_FILTER_FDIV16_N5

fSAMPLING=fDTS/16, N=5

LL_TIM_BREAK2_FILTER_FDIV16_N6

fSAMPLING=fDTS/16, N=6

LL_TIM_BREAK2_FILTER_FDIV16_N8

fSAMPLING=fDTS/16, N=8

LL_TIM_BREAK2_FILTER_FDIV32_N5

fSAMPLING=fDTS/32, N=5

LL_TIM_BREAK2_FILTER_FDIV32_N6

fSAMPLING=fDTS/32, N=6

LL_TIM_BREAK2_FILTER_FDIV32_N8

fSAMPLING=fDTS/32, N=8

BREAK2 POLARITY

LL_TIM_BREAK2_POLARITY_LOW

Break input BRK2 is active low

LL_TIM_BREAK2_POLARITY_HIGH

Break input BRK2 is active high

BREAK AF MODE

LL_TIM_BREAK_AFMODE_INPUT

Break input BRK in input mode

LL_TIM_BREAK_AFMODE_BIDIRECTIONAL

Break input BRK in bidirectional mode

Break Enable

LL_TIM_BREAK_DISABLE

Break function disabled

LL_TIM_BREAK_ENABLE

Break function enabled

break filter

LL_TIM_BREAK_FILTER_FDIV1

No filter, BRK acts asynchronously

LL_TIM_BREAK_FILTER_FDIV1_N2

fSAMPLING=fCK_INT, N=2

LL_TIM_BREAK_FILTER_FDIV1_N4

fSAMPLING=fCK_INT, N=4

LL_TIM_BREAK_FILTER_FDIV1_N8

fSAMPLING=fCK_INT, N=8

LL_TIM_BREAK_FILTER_FDIV2_N6

fSAMPLING=fDTS/2, N=6

LL_TIM_BREAK_FILTER_FDIV2_N8

fSAMPLING=fDTS/2, N=8

LL_TIM_BREAK_FILTER_FDIV4_N6

fSAMPLING=fDTS/4, N=6

LL_TIM_BREAK_FILTER_FDIV4_N8

fSAMPLING=fDTS/4, N=8

LL_TIM_BREAK_FILTER_FDIV8_N6

fSAMPLING=fDTS/8, N=6

LL_TIM_BREAK_FILTER_FDIV8_N8

fSAMPLING=fDTS/8, N=8

LL_TIM_BREAK_FILTER_FDIV16_N5

fSAMPLING=fDTS/16, N=5

LL_TIM_BREAK_FILTER_FDIV16_N6

fSAMPLING=fDTS/16, N=6

LL_TIM_BREAK_FILTER_FDIV16_N8

fSAMPLING=fDTS/16, N=8

LL_TIM_BREAK_FILTER_FDIV32_N5

fSAMPLING=fDTS/32, N=5

LL_TIM_BREAK_FILTER_FDIV32_N6

fSAMPLING=fDTS/32, N=6

LL_TIM_BREAK_FILTER_FDIV32_N8

fSAMPLING=fDTS/32, N=8

BREAK INPUT

LL_TIM_BREAK_INPUT_BKIN

TIMx_BKIN input

LL_TIM_BREAK_INPUT_BKIN2

TIMx_BKIN2 input

break polarity

LL_TIM_BREAK_POLARITY_LOW

Break input BRK is active low

LL_TIM_BREAK_POLARITY_HIGH

Break input BRK is active high

Capture Compare DMA Request

LL_TIM_CCDMAREQUEST_CC

CCx DMA request sent when CCx event occurs

LL_TIM_CCDMAREQUEST_UPDATE

CCx DMA requests sent when update event occurs

Capture Compare Update Source

LL_TIM_CCUPDATESOURCE_COMG_ONLY

Capture/compare control bits are updated by setting the COMG bit only

LL_TIM_CCUPDATESOURCE_COMG_AND_TRGI

Capture/compare control bits are updated by setting the COMG bit or when a rising edge occurs on trigger input (TRGI)

Channel

LL_LPTIM_CHANNEL_CH1

LPTIM input/output channel 1

LL_LPTIM_CHANNEL_CH2

LPTIM input/output channel 2

LL_TIM_CHANNEL_CH1

Timer input/output channel 1

LL_TIM_CHANNEL_CH1N

Timer complementary output channel 1

LL_TIM_CHANNEL_CH2

Timer input/output channel 2

LL_TIM_CHANNEL_CH2N

Timer complementary output channel 2

LL_TIM_CHANNEL_CH3

Timer input/output channel 3

LL_TIM_CHANNEL_CH3N

Timer complementary output channel 3

LL_TIM_CHANNEL_CH4

Timer input/output channel 4

LL_TIM_CHANNEL_CH4N

Timer complementary output channel 4

LL_TIM_CHANNEL_CH5

Timer output channel 5

LL_TIM_CHANNEL_CH6

Timer output channel 6

Clock Division

LL_TIM_CLOCKDIVISION_DIV1

tDTS=tCK_INT

LL_TIM_CLOCKDIVISION_DIV2

tDTS=2*tCK_INT

LL_TIM_CLOCKDIVISION_DIV4

tDTS=4*tCK_INT

Clock Source

LL_TIM_CLOCKSOURCE_INTERNAL

The timer is clocked by the internal clock provided from the RCC

LL_TIM_CLOCKSOURCE_EXT_MODE1

Counter counts at each rising or falling edge on a selected input

LL_TIM_CLOCKSOURCE_EXT_MODE2

Counter counts at each rising or falling edge on the external trigger input ETR

Counter Direction

LL_TIM_COUNTERDIRECTION_UP

Timer counter counts up

LL_TIM_COUNTERDIRECTION_DOWN

Timer counter counts down

Counter Mode

LL_TIM_COUNTERMODE_UP

Counter used as upcounter

LL_TIM_COUNTERMODE_DOWN

Counter used as downcounter

LL_TIM_COUNTERMODE_CENTER_DOWN

The counter counts up and down alternatively. Output compare interrupt flags of output channels are set only when the counter is counting down.

LL_TIM_COUNTERMODE_CENTER_UP

The counter counts up and down alternatively. Output compare interrupt flags of output channels are set only when the counter is counting up

LL_TIM_COUNTERMODE_CENTER_UP_DOWN

The counter counts up and down alternatively. Output compare interrupt flags of output channels are set only when the counter is counting up or down.

DMA Burst Base Address

LL_TIM_DMABURST_BASEADDR_CR1

TIMx_CR1 register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_CR2

TIMx_CR2 register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_SMCR

TIMx_SMCR register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_DIER

TIMx_DIER register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_SR

TIMx_SR register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_EGR

TIMx_EGR register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_CCMR1

TIMx_CCMR1 register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_CCMR2

TIMx_CCMR2 register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_CCER

TIMx_CCER register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_CNT

TIMx_CNT register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_PSC

TIMx_PSC register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_ARR

TIMx_ARR register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_RCR

TIMx_RCR register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_CCR1

TIMx_CCR1 register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_CCR2

TIMx_CCR2 register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_CCR3

TIMx_CCR3 register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_CCR4

TIMx_CCR4 register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_BDTR

TIMx_BDTR register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_CCR5

TIMx_CCR5 register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_CCR6

TIMx_CCR6 register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_CCMR3

TIMx_CCMR3 register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_DTR2

TIMx_DTR2 register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_ECR

TIMx_ECR register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_TISEL

TIMx_TISEL register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_AF1

TIMx_AF1 register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_AF2

TIMx_AF2 register is the DMA base address for DMA burst

DMA Burst Length

LL_TIM_DMABURST_LENGTH_1TRANSFER

Transfer is done to 1 register starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_2TRANSFERS

Transfer is done to 2 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_3TRANSFERS

Transfer is done to 3 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_4TRANSFERS

Transfer is done to 4 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_5TRANSFERS

Transfer is done to 5 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_6TRANSFERS

Transfer is done to 6 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_7TRANSFERS

Transfer is done to 7 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_8TRANSFERS

Transfer is done to 1 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_9TRANSFERS

Transfer is done to 9 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_10TRANSFERS

Transfer is done to 10 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_11TRANSFERS

Transfer is done to 11 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_12TRANSFERS

Transfer is done to 12 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_13TRANSFERS

Transfer is done to 13 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_14TRANSFERS

Transfer is done to 14 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_15TRANSFERS

Transfer is done to 15 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_16TRANSFERS

Transfer is done to 16 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_17TRANSFERS

Transfer is done to 17 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_18TRANSFERS

Transfer is done to 18 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_19TRANSFERS

Transfer is done to 19 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_20TRANSFERS

Transfer is done to 20 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_21TRANSFERS

Transfer is done to 21 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_22TRANSFERS

Transfer is done to 22 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_23TRANSFERS

Transfer is done to 23 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_24TRANSFERS

Transfer is done to 24 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_25TRANSFERS

Transfer is done to 25 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_26TRANSFERS

Transfer is done to 26 registers starting from the DMA burst base address

DMA Burst Source

LL_TIM_DMA_UPDATE

Transfer source is update event

LL_TIM_DMA_CC1

Transfer source is CC1 event

LL_TIM_DMA_CC2

Transfer source is CC2 event

LL_TIM_DMA_CC3

Transfer source is CC3 event

LL_TIM_DMA_CC4

Transfer source is CC4 event

LL_TIM_DMA_COM

Transfer source is COM event

LL_TIM_DMA_TRIGGER

Transfer source is trigger event

Encoder Mode

LL_TIM_ENCODERMODE_X2_TI1

Quadrature encoder mode 1, x2 mode - Counter counts up/down on TI1FP1 edge depending on TI2FP2 level

LL_TIM_ENCODERMODE_X2_TI2

Quadrature encoder mode 2, x2 mode - Counter counts up/down on TI2FP2 edge depending on TI1FP1 level

LL_TIM_ENCODERMODE_X4_TI12

Quadrature encoder mode 3, x4 mode - Counter counts up/down on both TI1FP1 and TI2FP2 edges depending on the level of the other input

LL_TIM_ENCODERMODE_CLOCKPLUSDIRECTION_X2

Encoder mode: Clock plus direction - x2 mode

LL_TIM_ENCODERMODE_CLOCKPLUSDIRECTION_X1

Encoder mode: Clock plus direction, x1 mode, TI2FP2 edge sensitivity is set by CC2P

LL_TIM_ENCODERMODE_DIRECTIONALCLOCK_X2

Encoder mode: Directional Clock, x2 mode

LL_TIM_ENCODERMODE_DIRECTIONALCLOCK_X1_TI12

Encoder mode: Directional Clock, x1 mode, TI1FP1 and TI2FP2 edge sensitivity is set by CC1P and CC2P

LL_TIM_ENCODERMODE_X1_TI1

Quadrature encoder mode: x1 mode, counting on TI1FP1 edges only, edge sensitivity is set by CC1P

LL_TIM_ENCODERMODE_X1_TI2

Quadrature encoder mode: x1 mode, counting on TI2FP2 edges only, edge sensitivity is set by CC1P

External Trigger Filter

LL_TIM_ETR_FILTER_FDIV1

No filter, sampling is done at fDTS

LL_TIM_ETR_FILTER_FDIV1_N2

fSAMPLING=fCK_INT, N=2

LL_TIM_ETR_FILTER_FDIV1_N4

fSAMPLING=fCK_INT, N=4

LL_TIM_ETR_FILTER_FDIV1_N8

fSAMPLING=fCK_INT, N=8

LL_TIM_ETR_FILTER_FDIV2_N6

fSAMPLING=fDTS/2, N=6

LL_TIM_ETR_FILTER_FDIV2_N8

fSAMPLING=fDTS/2, N=8

LL_TIM_ETR_FILTER_FDIV4_N6

fSAMPLING=fDTS/4, N=6

LL_TIM_ETR_FILTER_FDIV4_N8

fSAMPLING=fDTS/4, N=8

LL_TIM_ETR_FILTER_FDIV8_N6

fSAMPLING=fDTS/8, N=6

LL_TIM_ETR_FILTER_FDIV8_N8

fSAMPLING=fDTS/16, N=8

LL_TIM_ETR_FILTER_FDIV16_N5

fSAMPLING=fDTS/16, N=5

LL_TIM_ETR_FILTER_FDIV16_N6

fSAMPLING=fDTS/16, N=6

LL_TIM_ETR_FILTER_FDIV16_N8

fSAMPLING=fDTS/16, N=8

LL_TIM_ETR_FILTER_FDIV32_N5

fSAMPLING=fDTS/32, N=5

LL_TIM_ETR_FILTER_FDIV32_N6

fSAMPLING=fDTS/32, N=6

LL_TIM_ETR_FILTER_FDIV32_N8

fSAMPLING=fDTS/32, N=8

External Trigger Polarity

LL_TIM_ETR_POLARITY_NONINVERTED

ETR is non-inverted, active at high level or rising edge

LL_TIM_ETR_POLARITY_INVERTED

ETR is inverted, active at low level or falling edge

External Trigger Prescaler

LL_TIM_ETR_PRESCALER_DIV1

ETR prescaler OFF

LL_TIM_ETR_PRESCALER_DIV2

ETR frequency is divided by 2

LL_TIM_ETR_PRESCALER_DIV4

ETR frequency is divided by 4

LL_TIM_ETR_PRESCALER_DIV8

ETR frequency is divided by 8

first index selection

LL_TIM_INDEX_ALL

Index is always active

LL_TIM_INDEX_FIRST_ONLY

The first Index only resets the counter

Get Flags Defines

LL_TIM_SR_UIF

Update interrupt flag

LL_TIM_SR_CC1IF

Capture/compare 1 interrupt flag

LL_TIM_SR_CC2IF

Capture/compare 2 interrupt flag

LL_TIM_SR_CC3IF

Capture/compare 3 interrupt flag

LL_TIM_SR_CC4IF

Capture/compare 4 interrupt flag

LL_TIM_SR_CC5IF

Capture/compare 5 interrupt flag

LL_TIM_SR_CC6IF

Capture/compare 6 interrupt flag

LL_TIM_SR_COMIF

COM interrupt flag

LL_TIM_SR_TIF

Trigger interrupt flag

LL_TIM_SR_BIF

Break interrupt flag

LL_TIM_SR_B2IF

Second break interrupt flag

LL_TIM_SR_CC1OF

Capture/Compare 1 overcapture flag

LL_TIM_SR_CC2OF

Capture/Compare 2 overcapture flag

LL_TIM_SR_CC3OF

Capture/Compare 3 overcapture flag

LL_TIM_SR_CC4OF

Capture/Compare 4 overcapture flag

LL_TIM_SR_SBIF

System Break interrupt flag

LL_TIM_SR_IDXF

Index interrupt flag

LL_TIM_SR_DIRF

Direction Change interrupt flag

LL_TIM_SR_IERRF

Index Error flag

LL_TIM_SR_TERRF

Transition Error flag

GROUPCH5

LL_TIM_GROUPCH5_NONE

No effect of OC5REF on OC1REFC, OC2REFC and OC3REFC

LL_TIM_GROUPCH5_OC1REFC

OC1REFC is the logical AND of OC1REFC and OC5REF

LL_TIM_GROUPCH5_OC2REFC

OC2REFC is the logical AND of OC2REFC and OC5REF

LL_TIM_GROUPCH5_OC3REFC

OC3REFC is the logical AND of OC3REFC and OC5REF

Input Configuration Prescaler

LL_TIM_ICPSC_DIV1

No prescaler, capture is done each time an edge is detected on the capture input

LL_TIM_ICPSC_DIV2

Capture is done once every 2 events

LL_TIM_ICPSC_DIV4

Capture is done once every 4 events

LL_TIM_ICPSC_DIV8

Capture is done once every 8 events

Input Configuration Filter

LL_TIM_IC_FILTER_FDIV1

No filter, sampling is done at fDTS

LL_TIM_IC_FILTER_FDIV1_N2

fSAMPLING=fCK_INT, N=2

LL_TIM_IC_FILTER_FDIV1_N4

fSAMPLING=fCK_INT, N=4

LL_TIM_IC_FILTER_FDIV1_N8

fSAMPLING=fCK_INT, N=8

LL_TIM_IC_FILTER_FDIV2_N6

fSAMPLING=fDTS/2, N=6

LL_TIM_IC_FILTER_FDIV2_N8

fSAMPLING=fDTS/2, N=8

LL_TIM_IC_FILTER_FDIV4_N6

fSAMPLING=fDTS/4, N=6

LL_TIM_IC_FILTER_FDIV4_N8

fSAMPLING=fDTS/4, N=8

LL_TIM_IC_FILTER_FDIV8_N6

fSAMPLING=fDTS/8, N=6

LL_TIM_IC_FILTER_FDIV8_N8

fSAMPLING=fDTS/8, N=8

LL_TIM_IC_FILTER_FDIV16_N5

fSAMPLING=fDTS/16, N=5

LL_TIM_IC_FILTER_FDIV16_N6

fSAMPLING=fDTS/16, N=6

LL_TIM_IC_FILTER_FDIV16_N8

fSAMPLING=fDTS/16, N=8

LL_TIM_IC_FILTER_FDIV32_N5

fSAMPLING=fDTS/32, N=5

LL_TIM_IC_FILTER_FDIV32_N6

fSAMPLING=fDTS/32, N=6

LL_TIM_IC_FILTER_FDIV32_N8

fSAMPLING=fDTS/32, N=8

Input Configuration Polarity

LL_TIM_IC_POLARITY_RISING

The circuit is sensitive to TIxFP1 rising edge, TIxFP1 is not inverted

LL_TIM_IC_POLARITY_FALLING

The circuit is sensitive to TIxFP1 falling edge, TIxFP1 is inverted

LL_TIM_IC_POLARITY_BOTHEDGE

The circuit is sensitive to both TIxFP1 rising and falling edges, TIxFP1 is not inverted

index blanking selection

LL_TIM_INDEX_BLANK_ALWAYS

Index always active

LL_TIM_INDEX_BLANK_TI3

Index disabled when TI3 input is active, as per CC3P bitfield

LL_TIM_INDEX_BLANK_TI4

Index disabled when TI4 input is active, as per CC4P bitfield

index direction selection

LL_TIM_INDEX_UP_DOWN

Index resets the counter whatever the direction

LL_TIM_INDEX_UP

Index resets the counter when up-counting only

LL_TIM_INDEX_DOWN

Index resets the counter when down-counting only

index positioning selection

LL_TIM_INDEX_POSITION_DOWN_DOWN

Index resets the counter when AB = 00

LL_TIM_INDEX_POSITION_DOWN_UP

Index resets the counter when AB = 01

LL_TIM_INDEX_POSITION_UP_DOWN

Index resets the counter when AB = 10

LL_TIM_INDEX_POSITION_UP_UP

Index resets the counter when AB = 11

LL_TIM_INDEX_POSITION_DOWN

Index resets the counter when clock is 0

LL_TIM_INDEX_POSITION_UP

Index resets the counter when clock is 1

IT Defines

LL_TIM_DIER_UIE

Update interrupt enable

LL_TIM_DIER_CC1IE

Capture/compare 1 interrupt enable

LL_TIM_DIER_CC2IE

Capture/compare 2 interrupt enable

LL_TIM_DIER_CC3IE

Capture/compare 3 interrupt enable

LL_TIM_DIER_CC4IE

Capture/compare 4 interrupt enable

LL_TIM_DIER_COMIE

COM interrupt enable

LL_TIM_DIER_TIE

Trigger interrupt enable

LL_TIM_DIER_BIE

Break interrupt enable

LL_TIM_DIER_IDXIE

Index interrupt enable

LL_TIM_DIER_DIRIE

Direction Change interrupt enable

LL_TIM_DIER_IERRIE

Index Error interrupt enable

LL_TIM_DIER_TERRIE

Transition Error interrupt enable

Lock Level

LL_TIM_LOCKLEVEL_OFF

LOCK OFF - No bit is write protected

LL_TIM_LOCKLEVEL_1

LOCK Level 1

LL_TIM_LOCKLEVEL_2

LOCK Level 2

LL_TIM_LOCKLEVEL_3

LOCK Level 3

Output Configuration Idle State

LL_TIM_OCIDLESTATE_LOW

OCx=0 (after a dead-time if OC is implemented) when MOE=0

LL_TIM_OCIDLESTATE_HIGH

OCx=1 (after a dead-time if OC is implemented) when MOE=0

Output Configuration Mode

LL_TIM_OCMODE_FROZEN

The comparison between the output compare register TIMx_CCRy and the counter TIMx_CNT has no effect on the output channel level

LL_TIM_OCMODE_ACTIVE

OCyREF is forced high on compare match

LL_TIM_OCMODE_INACTIVE

OCyREF is forced low on compare match

LL_TIM_OCMODE_TOGGLE

OCyREF toggles on compare match

LL_TIM_OCMODE_FORCED_INACTIVE

OCyREF is forced low

LL_TIM_OCMODE_FORCED_ACTIVE

OCyREF is forced high

LL_TIM_OCMODE_PWM1

In upcounting, channel y is active as long as TIMx_CNT<TIMx_CCRy else inactive. In downcounting, channel y is inactive as long as TIMx_CNT>TIMx_CCRy else active.

LL_TIM_OCMODE_PWM2

In upcounting, channel y is inactive as long as TIMx_CNT<TIMx_CCRy else active. In downcounting, channel y is active as long as TIMx_CNT>TIMx_CCRy else inactive

LL_TIM_OCMODE_RETRIG_OPM1

Retrigerrable OPM mode 1

LL_TIM_OCMODE_RETRIG_OPM2

Retrigerrable OPM mode 2

LL_TIM_OCMODE_COMBINED_PWM1

Combined PWM mode 1

LL_TIM_OCMODE_COMBINED_PWM2

Combined PWM mode 2

LL_TIM_OCMODE_ASYMMETRIC_PWM1

Asymmetric PWM mode 1

LL_TIM_OCMODE_ASYMMETRIC_PWM2

Asymmetric PWM mode 2

LL_TIM_OCMODE_PULSE_ON_COMPARE

Pulse on Compare mode

LL_TIM_OCMODE_DIRECTION_OUTPUT

Direction output mode

Output Configuration Polarity

LL_TIM_OCPOLARITY_HIGH

OCxactive high

LL_TIM_OCPOLARITY_LOW

OCxactive low

Output Configuration State

LL_TIM_OCSTATE_DISABLE

OCx is not active

LL_TIM_OCSTATE_ENABLE

OCx signal is output on the corresponding output pin

One Pulse Mode

LL_TIM_ONEPULSEMODE_SINGLE

Counter stops counting at the next update event

LL_TIM_ONEPULSEMODE_REPETITIVE

Counter is not stopped at update event

OSSI

LL_TIM_OSSI_DISABLE

When inactive, OCx/OCxN outputs are disabled

LL_TIM_OSSI_ENABLE

When inactive, OxC/OCxN outputs are first forced with their inactive level then forced to their idle level after the deadtime

OSSR

LL_TIM_OSSR_DISABLE

When inactive, OCx/OCxN outputs are disabled

LL_TIM_OSSR_ENABLE

When inactive, OC/OCN outputs are enabled with their inactive level as soon as CCxE=1 or CCxNE=1

Pulse on compare pulse width prescaler

LL_TIM_PWPRSC_X1

Pulse on compare pulse width prescaler 1

LL_TIM_PWPRSC_X2

Pulse on compare pulse width prescaler 2

LL_TIM_PWPRSC_X4

Pulse on compare pulse width prescaler 4

LL_TIM_PWPRSC_X8

Pulse on compare pulse width prescaler 8

LL_TIM_PWPRSC_X16

Pulse on compare pulse width prescaler 16

LL_TIM_PWPRSC_X32

Pulse on compare pulse width prescaler 32

LL_TIM_PWPRSC_X64

Pulse on compare pulse width prescaler 64

LL_TIM_PWPRSC_X128

Pulse on compare pulse width prescaler 128

Slave Mode

LL_TIM_SLAVEMODE_DISABLED

Slave mode disabled

LL_TIM_SLAVEMODE_RESET

Reset Mode - Rising edge of the selected trigger input (TRGI) reinitializes the counter

LL_TIM_SLAVEMODE_GATED

Gated Mode - The counter clock is enabled when the trigger input (TRGI) is high

LL_TIM_SLAVEMODE_TRIGGER

Trigger Mode - The counter starts at a rising edge of the trigger TRGI

LL_TIM_SLAVEMODE_COMBINED_RESETTRIGGER

Combined reset + trigger mode - Rising edge of the selected trigger input (TRGI) reinitializes the counter, generates an update of the registers and starts the counter

LL_TIM_SLAVEMODE_COMBINED_GATEDRESET

Combined gated + reset mode - The counter clock is enabled when the trigger input (TRGI) is high. The counter stops and is reset) as soon as the trigger becomes low.Both startand stop of the counter are controlled.

SMS Preload Source

LL_TIM_SMSPS_TIMUPDATE

The SMS preload transfer is triggered by the Timer's Update event

LL_TIM_SMSPS_INDEX

The SMS preload transfer is triggered by the Index event

TIM10 External Input Ch1 Remap

LL_TIM_TIM10_TI1_RMP_GPIO

TIM10_TI1 is connected to GPIO

LL_TIM_TIM10_TI1_RMP_I3C1_IBIACK

TIM10_TI1 is connected to I3C1 IBI ACK

TIM11 External Input Ch1 Remap

LL_TIM_TIM11_TI1_RMP_GPIO

TIM11_TI1 is connected to GPIO

LL_TIM_TIM11_TI1_RMP_I3C2_IBIACK

TIM11_TI1 is connected to I3C2 IBI ACK

TIM12 External Input Ch1 Remap

LL_TIM_TIM12_TI1_RMP_GPIO

TIM12_TI1 is connected to GPIO

LL_TIM_TIM12_TI1_RMP_SPDIF_FS

TIM12 TI1 is connected to SPDIF FS

LL_TIM_TIM12_TI1_RMP_HSI_1024

TIM12_TI1 is connected to HSI/1024

LL_TIM_TIM12_TI1_RMP_MSI_128

TIM12_TI1 is connected to MSI/128

LL_TIM_TIM12_TI1_RMP_MCO1

TIM12_TI1 is connected to MCO1

LL_TIM_TIM12_TI1_RMP_MCO2

TIM12_TI1 is connected to MCO2

TIM13 External Input Ch1 Remap

LL_TIM_TIM13_TI1_RMP_GPIO

TIM13_TI1 is connected to GPIO

LL_TIM_TIM13_TI1_RMP_I3C1_IBIACK

TIM13_TI1 is connected to I3C1 IBI ACK

TIM14 External Input Ch1 Remap

LL_TIM_TIM14_TI1_RMP_GPIO

TIM14_TI1 is connected to GPIO

LL_TIM_TIM14_TI1_RMP_I3C2_IBIACK

TIM14_TI1 is connected to I3C2 IBI ACK

TIM15 External Input Ch1 Remap

LL_TIM_TIM15_TI1_RMP_GPIO

TIM15_TI1 is connected to GPIO

LL_TIM_TIM15_TI1_RMP_TIM2_CH1

TIM15_TI1 is connected to TIM2 CH1 GPIO

LL_TIM_TIM15_TI1_RMP_TIM3_CH1

TIM15_TI1 is connected to TIM3 CH1 GPIO

LL_TIM_TIM15_TI1_RMP_TIM4_CH1

TIM15_TI1 is connected to TIM4 CH1 GPIO

LL_TIM_TIM15_TI1_RMP_MCO1

TIM15_TI1 is connected to MCO1

LL_TIM_TIM15_TI1_RMP_MCO2

TIM15_TI1 is connected to MCO2

TIM15 External Input Ch2 Remap

LL_TIM_TIM15_TI2_RMP_GPIO

TIM15_TI2 is connected to GPIO

LL_TIM_TIM15_TI2_RMP_TIM2_CH2

TIM15_TI2 is connected to TIM2 CH2 GPIO

LL_TIM_TIM15_TI2_RMP_TIM3_CH2

TIM15_TI2 is connected to TIM3 CH2 GPIO

LL_TIM_TIM15_TI2_RMP_TIM4_CH2

TIM15_TI2 is connected to TIM4 CH2 GPIO

TIM16 External Input Ch1 Remap

LL_TIM_TIM16_TI1_RMP_GPIO

TIM16_TI1 is connected to GPIO

LL_TIM_TIM16_TI1_RMP_LSI

TIM16 TI1 is connected to LSI

LL_TIM_TIM16_TI1_RMP_LSE

TIM16 TI1 is connected to LSE

LL_TIM_TIM16_TI1_RMP_RTC_WKUP

TIM16_TI1 is connected to RTC Wakeup

TIM17 External Input Ch1 Remap

LL_TIM_TIM17_TI1_RMP_GPIO

TIM17_TI1 is connected to GPIO

LL_TIM_TIM17_TI1_RMP_SPDIF_FS

TIM17_TI1 is connected to SPDIFRX FS

LL_TIM_TIM17_TI1_RMP_HSE_1024

TIM17 TI1 is connected to HSE/1024

External Trigger Source TIM1

LL_TIM_TIM1_ETRSOURCE_GPIO

TIM1_ETR is connected to I/O

LL_TIM_TIM1_ETRSOURCE_ADC1_AWD1

TIM1_ETR is connected to ADC1 AWD1

LL_TIM_TIM1_ETRSOURCE_ADC1_AWD2

TIM1_ETR is connected to ADC1 AWD2

LL_TIM_TIM1_ETRSOURCE_ADC1_AWD3

TIM1_ETR is connected to ADC1 AWD3

LL_TIM_TIM1_ETRSOURCE_ADC2_AWD1

TIM1_ETR is connected to ADC2 AWD1

LL_TIM_TIM1_ETRSOURCE_ADC2_AWD2

TIM1_ETR is connected to ADC2 AWD2

LL_TIM_TIM1_ETRSOURCE_ADC2_AWD3

TIM1_ETR is connected to ADC2 AWD3

External Trigger Source TIM2

LL_TIM_TIM2_ETRSOURCE_GPIO

TIM2_ETR is connected to I/O

LL_TIM_TIM2_ETRSOURCE_DCMIPP_HSYNC

TIM2_ETR is connected to DCMIPP HSYNC

LL_TIM_TIM2_ETRSOURCE_LCD_HSYNC

TIM2_ETR is connected to LCD HSYNC

LL_TIM_TIM2_ETRSOURCE_SAI1_FSA

TIM2_ETR is connected to SAI1 FS_A

LL_TIM_TIM2_ETRSOURCE_SAI1_FSB

TIM2_ETR is connected to SAI1 FS_B

LL_TIM_TIM2_ETRSOURCE_GFXTIM_TE

TIM2_ETR is connected to GFXTIM TE

LL_TIM_TIM2_ETRSOURCE_DCMIPP_VSYNC

TIM2_ETR is connected to DCMIPP VSYNC

LL_TIM_TIM2_ETRSOURCE_LCD_VSYNC

TIM2_ETR is connected to LCD VSYNC

LL_TIM_TIM2_ETRSOURCE_TIM3_ETR

TIM2_ETR is connected to TIM3 ETR

LL_TIM_TIM2_ETRSOURCE_TIM4_ETR

TIM2_ETR is connected to TIM4 ETR

LL_TIM_TIM2_ETRSOURCE_TIM5_ETR

TIM2_ETR is connected to TIM5 ETR

LL_TIM_TIM2_ETRSOURCE_ETH1_PPS

TIM2_ETR is connected to ETH1 PPS

LL_TIM_TIM2_ETRSOURCE_USB1_SOF

TIM2_ETR is connected to USB1 OTG SOF

LL_TIM_TIM2_ETRSOURCE_USB2_SOF

TIM2_ETR is connected to USB2 OTG SOF

TIM2 External Input Ch1 Remap

LL_TIM_TIM2_TI1_RMP_GPIO

TIM2_TI1 is connected to GPIO

LL_TIM_TIM2_TI1_RMP_ETH1_PPS

TIM2_TI1 is connected to ETH1 PPS

External Trigger Source TIM3

LL_TIM_TIM3_ETRSOURCE_GPIO

TIM3_ETR is connected to I/O

LL_TIM_TIM3_ETRSOURCE_DCMIPP_HSYNC

TIM3_ETR is connected to DCMIPP HSYNC

LL_TIM_TIM3_ETRSOURCE_LCD_HSYNC

TIM3_ETR is connected to LCD HSYNC

LL_TIM_TIM3_ETRSOURCE_GFXTIM_TE

TIM3_ETR is connected to GFXTIM TE

LL_TIM_TIM3_ETRSOURCE_DCMIPP_VSYNC

TIM3_ETR is connected to DCMIPP VSYNC

LL_TIM_TIM3_ETRSOURCE_LCD_VSYNC

TIM3_ETR is connected to LCD VSYNC

LL_TIM_TIM3_ETRSOURCE_TIM2_ETR

TIM3_ETR is connected to TIM2 ETR

LL_TIM_TIM3_ETRSOURCE_TIM4_ETR

TIM3_ETR is connected to TIM4 ETR

LL_TIM_TIM3_ETRSOURCE_TIM5_ETR

TIM3_ETR is connected to TIM5 ETR

LL_TIM_TIM3_ETRSOURCE_ETH1_PPS

TIM3_ETR is connected to ETH1 PPS

TIM3 External Input Ch1 Remap

LL_TIM_TIM3_TI1_RMP_GPIO

TIM3_TI1 is connected to GPIO

LL_TIM_TIM3_TI1_RMP_ETH1_PPS

TIM3_TI1 is connected to ETH1 PPS

LL_TIM_TIM3_TI1_RMP_FDCAN_RTP

TIM3_TI1 is connected to FDCAN RTP

LL_TIM_TIM3_TI1_RMP_FDCAN_TMP

TIM3_TI1 is connected to FDCAN TMP

LL_TIM_TIM3_TI1_RMP_FDCAN_SOC

TIM3_TI1 is connected to FDCAN SOC

External Trigger Source TIM4

LL_TIM_TIM4_ETRSOURCE_GPIO

TIM4_ETR is connected to I/O

LL_TIM_TIM4_ETRSOURCE_DCMIPP_HSYNC

TIM4_ETR is connected to DCMIPP HSYNC

LL_TIM_TIM4_ETRSOURCE_LCD_HSYNC

TIM4_ETR is connected to LCD HSYNC

LL_TIM_TIM4_ETRSOURCE_GFXTIM_TE

TIM4_ETR is connected to GFXTIM TE

LL_TIM_TIM4_ETRSOURCE_DCMIPP_VSYNC

TIM4_ETR is connected to DCMIPP VSYNC

LL_TIM_TIM4_ETRSOURCE_LCD_VSYNC

TIM4_ETR is connected to LCD VSYNC

LL_TIM_TIM4_ETRSOURCE_TIM2_ETR

TIM4_ETR is connected to TIM2 ETR

LL_TIM_TIM4_ETRSOURCE_TIM3_ETR

TIM4_ETR is connected to TIM3 ETR

LL_TIM_TIM4_ETRSOURCE_TIM5_ETR

TIM4_ETR is connected to TIM5 ETR

External Trigger Source TIM5

LL_TIM_TIM5_ETRSOURCE_GPIO

TIM5_ETR is connected to I/O

LL_TIM_TIM5_ETRSOURCE_SAI2_FSA

TIM5_ETR is connected to SAI2 FS_A

LL_TIM_TIM5_ETRSOURCE_SAI2_FSB

TIM5_ETR is connected to SAI2 FS_B

LL_TIM_TIM5_ETRSOURCE_DCMIPP_HSYNC

TIM5_ETR is connected to DCMIPP HSYNC

LL_TIM_TIM5_ETRSOURCE_LCD_HSYNC

TIM5_ETR is connected to LCD HSYNC

LL_TIM_TIM5_ETRSOURCE_GFXTIM_TE

TIM5_ETR is connected to GFXTIM TE

LL_TIM_TIM5_ETRSOURCE_DCMIPP_VSYNC

TIM5_ETR is connected to DCMIPP VSYNC

LL_TIM_TIM5_ETRSOURCE_LCD_VSYNC

TIM5_ETR is connected to LCD VSYNC

LL_TIM_TIM5_ETRSOURCE_TIM2_ETR

TIM5_ETR is connected to TIM2 ETR

LL_TIM_TIM5_ETRSOURCE_TIM3_ETR

TIM5_ETR is connected to TIM3 ETR

LL_TIM_TIM5_ETRSOURCE_TIM4_ETR

TIM5_ETR is connected to TIM5 ETR

LL_TIM_TIM5_ETRSOURCE_USB1_SOF

TIM5_ETR is connected to USB1 OTG SOF

LL_TIM_TIM5_ETRSOURCE_USB2_SOF

TIM5_ETR is connected to USB2 OTG SOF

TIM5 External Input Ch1 Remap

LL_TIM_TIM5_TI1_RMP_GPIO

TIM5_TI1 is connected to GPIO

LL_TIM_TIM5_TI1_RMP_FDCAN_RTP

TIM5_TI1 is connected to FDCAN RTP

LL_TIM_TIM5_TI1_RMP_FDCAN_TMP

TIM5_TI1 is connected to FDCAN TMP

External Trigger Source TIM8

LL_TIM_TIM8_ETRSOURCE_GPIO

TIM8_ETR is connected to I/O

LL_TIM_TIM8_ETRSOURCE_ADC2_AWD1

TIM8_ETR is connected to ADC2 AWD1

LL_TIM_TIM8_ETRSOURCE_ADC2_AWD2

TIM8_ETR is connected to ADC2 AWD2

LL_TIM_TIM8_ETRSOURCE_ADC2_AWD3

TIM8_ETR is connected to ADC2 AWD3

TIM9 External Input Ch1 Remap

LL_TIM_TIM9_TI1_RMP_GPIO

TIM9_TI1 is connected to GPIO

LL_TIM_TIM9_TI1_RMP_MCO1

TIM9_TI1 is connected to MCO1

LL_TIM_TIM9_TI1_RMP_MCO2

TIM9_TI1 is connected to MCO2

Trigger Output

LL_TIM_TRGO_RESET

UG bit from the TIMx_EGR register is used as trigger output

LL_TIM_TRGO_ENABLE

Counter Enable signal (CNT_EN) is used as trigger output

LL_TIM_TRGO_UPDATE

Update event is used as trigger output

LL_TIM_TRGO_CC1IF

CC1 capture or a compare match is used as trigger output

LL_TIM_TRGO_OC1REF

OC1REF signal is used as trigger output

LL_TIM_TRGO_OC2REF

OC2REF signal is used as trigger output

LL_TIM_TRGO_OC3REF

OC3REF signal is used as trigger output

LL_TIM_TRGO_OC4REF

OC4REF signal is used as trigger output

LL_TIM_TRGO_ENCODERCLK

Encoder clock signal is used as trigger output

Trigger Output 2

LL_TIM_TRGO2_RESET

UG bit from the TIMx_EGR register is used as trigger output 2

LL_TIM_TRGO2_ENABLE

Counter Enable signal (CNT_EN) is used as trigger output 2

LL_TIM_TRGO2_UPDATE

Update event is used as trigger output 2

LL_TIM_TRGO2_CC1F

CC1 capture or a compare match is used as trigger output 2

LL_TIM_TRGO2_OC1

OC1REF signal is used as trigger output 2

LL_TIM_TRGO2_OC2

OC2REF signal is used as trigger output 2

LL_TIM_TRGO2_OC3

OC3REF signal is used as trigger output 2

LL_TIM_TRGO2_OC4

OC4REF signal is used as trigger output 2

LL_TIM_TRGO2_OC5

OC5REF signal is used as trigger output 2

LL_TIM_TRGO2_OC6

OC6REF signal is used as trigger output 2

LL_TIM_TRGO2_OC4_RISINGFALLING

OC4REF rising or falling edges are used as trigger output 2

LL_TIM_TRGO2_OC6_RISINGFALLING

OC6REF rising or falling edges are used as trigger output 2

LL_TIM_TRGO2_OC4_RISING_OC6_RISING

OC4REF or OC6REF rising edges are used as trigger output 2

LL_TIM_TRGO2_OC4_RISING_OC6_FALLING

OC4REF rising or OC6REF falling edges are used as trigger output 2

LL_TIM_TRGO2_OC5_RISING_OC6_RISING

OC5REF or OC6REF rising edges are used as trigger output 2

LL_TIM_TRGO2_OC5_RISING_OC6_FALLING

OC5REF rising or OC6REF falling edges are used as trigger output 2

Trigger Selection

LL_TIM_TS_ITR0

Internal Trigger 0 (ITR0) is used as trigger input

LL_TIM_TS_ITR1

Internal Trigger 1 (ITR1) is used as trigger input

LL_TIM_TS_ITR2

Internal Trigger 2 (ITR2) is used as trigger input

LL_TIM_TS_ITR3

Internal Trigger 3 (ITR3) is used as trigger input

LL_TIM_TS_ITR4

Internal Trigger 4 (ITR4) is used as trigger input

LL_TIM_TS_ITR5

Internal Trigger 5 (ITR5) is used as trigger input

LL_TIM_TS_ITR6

Internal Trigger 6 (ITR6) is used as trigger input

LL_TIM_TS_ITR7

Internal Trigger 7 (ITR7) is used as trigger input

LL_TIM_TS_ITR8

Internal Trigger 8 (ITR8) is used as trigger input

LL_TIM_TS_ITR9

Internal Trigger 9 (ITR9) is used as trigger input

LL_TIM_TS_ITR10

Internal Trigger 10 (ITR10) is used as trigger input

LL_TIM_TS_ITR11

Internal Trigger 11 (ITR11) is used as trigger input

LL_TIM_TS_ITR12

Internal Trigger 12 (ITR12) is used as trigger input

LL_TIM_TS_ITR13

Internal Trigger 13 (ITR13) is used as trigger input

LL_TIM_TS_TI1F_ED

TI1 Edge Detector (TI1F_ED) is used as trigger input

LL_TIM_TS_TI1FP1

Filtered Timer Input 1 (TI1FP1) is used as trigger input

LL_TIM_TS_TI2FP2

Filtered Timer Input 2 (TI12P2) is used as trigger input

LL_TIM_TS_ETRF

Filtered external Trigger (ETRF) is used as trigger input

Update Source

LL_TIM_UPDATESOURCE_REGULAR

Counter overflow/underflow, Setting the UG bit or Update generation through the slave mode controller generates an update request

LL_TIM_UPDATESOURCE_COUNTER

Only counter overflow/underflow generates an update request

Common Write and read registers Macros

LL_TIM_WriteReg

Description:

- Write a value in TIM register.

Parameters:

- __INSTANCE__: TIM Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_TIM_ReadReg

Description:

- Read a value in TIM register.

Parameters:

- __INSTANCE__: TIM Instance
- __REG__: Register to be read

Return value:

- Register: value

TIM Exported Macros

__LL_TIM_GETFLAG_UIFCPY

Description:

- HELPER macro retrieving the UIFCPY flag from the counter value.

Parameters:

- __CNT__: Counter value

Return value:

- UIF: status bit

Notes:

- ex: __LL_TIM_GETFLAG_UIFCPY (LL_TIM_GetCounter ()); Relevant only if UIF flag remapping has been enabled (UIF status bit is copied to TIMx_CNT register bit 31)

__LL_TIM_CALC_DEADTIME

Description:

- HELPER macro calculating DTG[0:7] in the TIMx_BDTR register to achieve the requested dead time duration.

Parameters:

- __TIMCLK__: timer input clock frequency (in Hz)
- __CKD__: This parameter can be one of the following values:
  - LL_TIM_CLOCKDIVISION_DIV1
  - LL_TIM_CLOCKDIVISION_DIV2
  - LL_TIM_CLOCKDIVISION_DIV4
- __DT__: deadtime duration (in ns)

Return value:

- DTG[0:7]

Notes:

- ex: __LL_TIM_CALC_DEADTIME (80000000, LL_TIM_GetClockDivision (), 120);

__LL_TIM_CALC_PSC

Description:

- HELPER macro calculating the prescaler value to achieve the required counter clock frequency.

Parameters:

- __TIMCLK__: timer input clock frequency (in Hz)
- __CNTCLK__: counter clock frequency (in Hz)

Return value:

- Prescaler: value (between Min_Data=0 and Max_Data=65535)

Notes:

- ex: __LL_TIM_CALC_PSC (80000000, 1000000);

__LL_TIM_CALC_ARR

Description:

- HELPER macro calculating the auto-reload value to achieve the required output signal frequency.

Parameters:

- __TIMCLK__: timer input clock frequency (in Hz)
- __PSC__: prescaler
- __FREQ__: output signal frequency (in Hz)

Return value:

- Auto-reload: value (between Min_Data=0 and Max_Data=65535)

Notes:

- ex: __LL_TIM_CALC_ARR (1000000, LL_TIM_GetPrescaler (), 10000);

__LL_TIM_CALC_ARR_DITHER

Description:

- HELPER macro calculating the auto-reload value, with dithering feature enabled, to achieve the required output signal frequency.

Parameters:

- __TIMCLK__: timer input clock frequency (in Hz)
- __PSC__: prescaler
- __FREQ__: output signal frequency (in Hz)

Return value:

- Auto-reload: value (between Min_Data=0 and Max_Data=65535)

Notes:

- ex: __LL_TIM_CALC_ARR_DITHER (1000000, LL_TIM_GetPrescaler (), 10000);

__LL_TIM_CALC_DELAY

Description:

- HELPER macro calculating the compare value required to achieve the required timer output compare active/inactive delay.

Parameters:

- __TIMCLK__: timer input clock frequency (in Hz)
- __PSC__: prescaler
- __DELAY__: timer output compare active/inactive delay (in us)

Return value:

- Compare: value (between Min_Data=0 and Max_Data=65535)

Notes:

- ex: __LL_TIM_CALC_DELAY (1000000, LL_TIM_GetPrescaler (), 10);

__LL_TIM_CALC_DELAY_DITHER

Description:

- HELPER macro calculating the compare value, with dithering feature enabled, to achieve the required timer output compare active/inactive delay.

Parameters:

- __TIMCLK__: timer input clock frequency (in Hz)
- __PSC__: prescaler
- __DELAY__: timer output compare active/inactive delay (in us)

Return value:

- Compare: value (between Min_Data=0 and Max_Data=65535)

Notes:

- ex: __LL_TIM_CALC_DELAY_DITHER (1000000, LL_TIM_GetPrescaler (), 10);

__LL_TIM_CALC_PULSE

Description:

- HELPER macro calculating the auto-reload value to achieve the required pulse duration (when the timer operates in one pulse mode).

Parameters:

- __TIMCLK__: timer input clock frequency (in Hz)
- __PSC__: prescaler
- __DELAY__: timer output compare active/inactive delay (in us)
- __PULSE__: pulse duration (in us)

Return value:

- Auto-reload: value (between Min_Data=0 and Max_Data=65535)

Notes:

- ex: __LL_TIM_CALC_PULSE (1000000, LL_TIM_GetPrescaler (), 10, 20);

__LL_TIM_CALC_PULSE_DITHER

Description:

- HELPER macro calculating the auto-reload value, with dithering feature enabled, to achieve the required pulse duration (when the timer operates in one pulse mode).

Parameters:

- __TIMCLK__: timer input clock frequency (in Hz)
- __PSC__: prescaler
- __DELAY__: timer output compare active/inactive delay (in us)
- __PULSE__: pulse duration (in us)

Return value:

- Auto-reload: value (between Min_Data=0 and Max_Data=65535)

Notes:

- ex: __LL_TIM_CALC_PULSE_DITHER (1000000, LL_TIM_GetPrescaler (), 10, 20);

__LL_TIM_GET_ICPSC_RATIO

Description:

- HELPER macro retrieving the ratio of the input capture prescaler.

Parameters:

- __ICPSC__: This parameter can be one of the following values:
  - LL_TIM_ICPSC_DIV1
  - LL_TIM_ICPSC_DIV2
  - LL_TIM_ICPSC_DIV4
  - LL_TIM_ICPSC_DIV8

Return value:

- Input: capture prescaler ratio (1, 2, 4 or 8)

Notes:

- ex: __LL_TIM_GET_ICPSC_RATIO (LL_TIM_IC_GetPrescaler ());

107 LL UCPD Generic Driver

107.1 UCPD Firmware driver registers structures

107.1.1 LL_UCPD_InitTypeDef

LL_UCPD_InitTypeDef is defined in the stm32n6xx_ll_ucpd.h

Data Fields

- ***uint32_t psc_ucpdclk***
- ***uint32_t transwin***
- ***uint32_t IfrGap***
- ***uint32_t HbitClockDiv***

Field Documentation

- ***uint32_t LL_UCPD_InitTypeDef::psc_ucpdclk***
  Specify the prescaler for the UCPD clock. This parameter can be a value of ***UCPD_LL_EC_PSC***. This feature can be modified afterwards using function **LL_UCPD_SetPSCClk()**.
- ***uint32_t LL_UCPD_InitTypeDef::transwin***
  Specify the number of cycles (minus 1) of the half bit clock (see HBITCLKDIV) to achieve a legal tTransitionWindow (set according to peripheral clock to define an interval of between 12 and 20 us). This parameter can be a value between Min_Data=0x1 and Max_Data=0x1F This value can be modified afterwards using function **LL_UCPD_SetTransWin()**.
- ***uint32_t LL_UCPD_InitTypeDef::IfrGap***
  Specify the definition of the clock divider (minus 1) in order to generate tInterframeGap from the peripheral clock. This parameter can be a value between Min_Data=0x1 and Max_Data=0x1F This feature can be modified afterwards using function **LL_UCPD_SetIfrGap()**.
- ***uint32_t LL_UCPD_InitTypeDef::HbitClockDiv***
  Specify the number of cycles (minus one) at UCPD peripheral for a half bit clock e.g. program 3 for a bit clock that takes 8 cycles of the peripheral clock : "UCPD1_CLK". This parameter can be a value between Min_Data=0x0 and Max_Data=0x3F. This feature can be modified using function **LL_UCPD_SetHbitClockDiv()**.

107.2 UCPD Firmware driver API description

The following section lists the various functions of the UCPD library.

107.2.1 Detailed description of functions

LL_UCPD_Enable

Function name

__STATIC_INLINE void LL_UCPD_Enable (UCPD_TypeDef * UCPDx)

Function description

Enable UCPD peripheral.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG1 UCPDEN LL_UCPD_Enable

LL_UCPD_Disable

Function name

__STATIC_INLINE void LL_UCPD_Disable (UCPD_TypeDef * UCPDx)

Function description

Disable UCPD peripheral.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Notes

- When disabling the UCPD, follow the procedure described in the Reference Manual.

Reference Manual to LL API cross reference:

- CFG1 UCPDEN LL_UCPD_Disable

LL_UCPD_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsEnabled (UCPD_TypeDef const *const UCPDx)

Function description

Check if UCPD peripheral is enabled.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CFG1 UCPDEN LL_UCPD_IsEnabled

LL_UCPD_SetRxOrderSet

Function name

__STATIC_INLINE void LL_UCPD_SetRxOrderSet (UCPD_TypeDef * UCPDx, uint32_t OrderSet)

Function description

Set the receiver ordered set detection enable.

Parameters

- **UCPDx:** UCPD Instance
- **OrderSet:** This parameter can be combination of the following values:
  - LL_UCPD_ORDERSET_SOP
  - LL_UCPD_ORDERSET_SOP1
  - LL_UCPD_ORDERSET_SOP2
  - LL_UCPD_ORDERSET_HARDRST
  - LL_UCPD_ORDERSET_CABLERST
  - LL_UCPD_ORDERSET_SOP1_DEBUG
  - LL_UCPD_ORDERSET_SOP2_DEBUG
  - LL_UCPD_ORDERSET_SOP_EXT1
  - LL_UCPD_ORDERSET_SOP_EXT2

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG1 RXORDSETEN LL_UCPD_SetRxOrderSet

LL_UCPD_SetPSCClk

Function name

__STATIC_INLINE void LL_UCPD_SetPSCClk (UCPD_TypeDef * UCPDx, uint32_t Psc)

Function description

Set the prescaler for ucpd clock.

Parameters

- **UCPDx:** UCPD Instance
- **Psc:** This parameter can be one of the following values:
  - LL_UCPD_PSC_DIV1
  - LL_UCPD_PSC_DIV2
  - LL_UCPD_PSC_DIV4
  - LL_UCPD_PSC_DIV8
  - LL_UCPD_PSC_DIV16

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG1 UCPDCLK LL_UCPD_SetPSCClk

LL_UCPD_SetTransWin

Function name

__STATIC_INLINE void LL_UCPD_SetTransWin (UCPD_TypeDef * UCPDx, uint32_t TransWin)

Function description

Set the number of cycles (minus 1) of the half bit clock.

Parameters

- **UCPDx:** UCPD Instance
- **TransWin:** a value between Min_Data=0x1 and Max_Data=0x1F

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG1 TRANSWIN LL_UCPD_SetTransWin

LL_UCPD_SetIfrGap

Function name

__STATIC_INLINE void LL_UCPD_SetIfrGap (UCPD_TypeDef * UCPDx, uint32_t IfrGap)

Function description

Set the clock divider value to generate an interframe gap.

Parameters

- **UCPDx:** UCPD Instance
- **IfrGap:** a value between Min_Data=0x1 and Max_Data=0x1F

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG1 IFRGAP LL_UCPD_SetIfrGap

LL_UCPD_SetHbitClockDiv

Function name

__STATIC_INLINE void LL_UCPD_SetHbitClockDiv (UCPD_TypeDef * UCPDx, uint32_t HbitClock)

Function description

Set the clock divider value to generate an interframe gap.

Parameters

- **UCPDx:** UCPD Instance
- **HbitClock:** a value between Min_Data=0x0 and Max_Data=0x3F

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG1 HBITCLKDIV LL_UCPD_SetHbitClockDiv

LL_UCPD_RxAnalogFilterEnable

Function name

__STATIC_INLINE void LL_UCPD_RxAnalogFilterEnable (UCPD_TypeDef * UCPDx)

Function description

Enable Rx Analog Filter.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG2 RXAFILTEN LL_UCPD_RxAnalogFilterEnable

LL_UCPD_RxAnalogFilterDisable

Function name

__STATIC_INLINE void LL_UCPD_RxAnalogFilterDisable (UCPD_TypeDef * UCPDx)

Function description

Disable Rx Analog Filter.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG2 RXAFILTEN LL_UCPD_RxAnalogFilterDisable

LL_UCPD_WakeUpEnable

Function name

__STATIC_INLINE void LL_UCPD_WakeUpEnable (UCPD_TypeDef * UCPDx)

Function description

Enable the wakeup mode.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG2 WUPEN LL_UCPD_WakeUpEnable

LL_UCPD_WakeUpDisable

Function name

__STATIC_INLINE void LL_UCPD_WakeUpDisable (UCPD_TypeDef * UCPDx)

Function description

Disable the wakeup mode.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG2 WUPEN LL_UCPD_WakeUpDisable

LL_UCPD_ForceClockEnable

Function name

__STATIC_INLINE void LL_UCPD_ForceClockEnable (UCPD_TypeDef * UCPDx)

Function description

Force clock enable.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG2 FORCECLK LL_UCPD_ForceClockEnable

LL_UCPD_ForceClockDisable

Function name

__STATIC_INLINE void LL_UCPD_ForceClockDisable (UCPD_TypeDef * UCPDx)

Function description

Force clock disable.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG2 FORCECLK LL_UCPD_ForceClockDisable

LL_UCPD_RxFilterEnable

Function name

__STATIC_INLINE void LL_UCPD_RxFilterEnable (UCPD_TypeDef * UCPDx)

Function description

RxFilter enable.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG2 RXFILTDIS LL_UCPD_RxFilterEnable

LL_UCPD_RxFilterDisable

Function name

__STATIC_INLINE void LL_UCPD_RxFilterDisable (UCPD_TypeDef * UCPDx)

Function description

RxFilter disable.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG2 RXFILTDIS LL_UCPD_RxFilterDisable

LL_UCPD_TypeCDetectionCC2Enable

Function name

__STATIC_INLINE void LL_UCPD_TypeCDetectionCC2Enable (UCPD_TypeDef * UCPDx)

Function description

Type C detector for CC2 enable.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CC2TCDIS LL_UCPD_TypeCDetectionCC2Enable

LL_UCPD_TypeCDetectionCC2Disable

Function name

__STATIC_INLINE void LL_UCPD_TypeCDetectionCC2Disable (UCPD_TypeDef * UCPDx)

Function description

Type C detector for CC2 disable.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CC2TCDIS LL_UCPD_TypeCDetectionCC2Disable

LL_UCPD_TypeCDetectionCC1Enable

Function name

__STATIC_INLINE void LL_UCPD_TypeCDetectionCC1Enable (UCPD_TypeDef * UCPDx)

Function description

Type C detector for CC1 enable.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CC1TCDIS LL_UCPD_TypeCDetectionCC1Enable

LL_UCPD_TypeCDetectionCC1Disable

Function name

__STATIC_INLINE void LL_UCPD_TypeCDetectionCC1Disable (UCPD_TypeDef * UCPDx)

Function description

Type C detector for CC1 disable.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CC1TCDIS LL_UCPD_TypeCDetectionCC1Disable

LL_UCPD_VconnDischargeEnable

Function name

__STATIC_INLINE void LL_UCPD_VconnDischargeEnable (UCPD_TypeDef * UCPDx)

Function description

Source Vconn discharge enable.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR RDCH LL_UCPD_VconnDischargeEnable

LL_UCPD_VconnDischargeDisable

Function name

__STATIC_INLINE void LL_UCPD_VconnDischargeDisable (UCPD_TypeDef * UCPDx)

Function description

Source Vconn discharge disable.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR RDCH LL_UCPD_VconnDischargeDisable

LL_UCPD_SignalFRSTX

Function name

__STATIC_INLINE void LL_UCPD_SignalFRSTX (UCPD_TypeDef * UCPDx)

Function description

Signal Fast Role Swap request.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR FRSTX LL_UCPD_VconnDischargeDisable

LL_UCPD_SetccEnable

Function name

__STATIC_INLINE void LL_UCPD_SetccEnable (UCPD_TypeDef * UCPDx, uint32_t CCEnable)

Function description

Set cc enable.

Parameters

- **UCPDx:** UCPD Instance
- **CCEnable:** This parameter can be one of the following values:
  - LL_UCPD_CCENABLE_NONE
  - LL_UCPD_CCENABLE_CC1
  - LL_UCPD_CCENABLE_CC2
  - LL_UCPD_CCENABLE_CC1CC2

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CC1VCONNEN LL_UCPD_SetccEnable

LL_UCPD_SetSNKRole

Function name

__STATIC_INLINE void LL_UCPD_SetSNKRole (UCPD_TypeDef * UCPDx)

Function description

Set UCPD SNK role.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR ANAMODE LL_UCPD_SetSNKRole

LL_UCPD_SetSRCRole

Function name

__STATIC_INLINE void LL_UCPD_SetSRCRole (UCPD_TypeDef * UCPDx)

Function description

Set UCPD SRC role.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR ANAMODE LL_UCPD_SetSRCRole

LL_UCPD_GetRole

Function name

__STATIC_INLINE uint32_t LL_UCPD_GetRole (UCPD_TypeDef const *const UCPDx)

Function description

Get UCPD Role.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_UCPD_ROLE_SNK
  - LL_UCPD_ROLE_SRC

Reference Manual to LL API cross reference:

- CR ANAMODE LL_UCPD_GetRole

LL_UCPD_SetRpResistor

Function name

__STATIC_INLINE void LL_UCPD_SetRpResistor (UCPD_TypeDef * UCPDx, uint32_t Resistor)

Function description

Set Rp resistor.

Parameters

- **UCPDx:** UCPD Instance
- **Resistor:** This parameter can be one of the following values:
  - LL_UCPD_RESISTOR_DEFAULT
  - LL_UCPD_RESISTOR_1_5A
  - LL_UCPD_RESISTOR_3_0A
  - LL_UCPD_RESISTOR_NONE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR ANASUBMODE LL_UCPD_SetRpResistor

LL_UCPD_SetCCPin

Function name

__STATIC_INLINE void LL_UCPD_SetCCPin (UCPD_TypeDef * UCPDx, uint32_t CCPin)

Function description

Set CC pin.

Parameters

- **UCPDx:** UCPD Instance
- **CCPin:** This parameter can be one of the following values:
  - LL_UCPD_CCPIN_CC1
  - LL_UCPD_CCPIN_CC2

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR PHYCCSEL LL_UCPD_SetCCPin

LL_UCPD_RxEnable

Function name

__STATIC_INLINE void LL_UCPD_RxEnable (UCPD_TypeDef * UCPDx)

Function description

Rx enable.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR PHYRXEN LL_UCPD_RxEnable

LL_UCPD_RxDisable

Function name

__STATIC_INLINE void LL_UCPD_RxDisable (UCPD_TypeDef * UCPDx)

Function description

Rx disable.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR PHYRXEN LL_UCPD_RxDisable

LL_UCPD_SetRxMode

Function name

__STATIC_INLINE void LL_UCPD_SetRxMode (UCPD_TypeDef * UCPDx, uint32_t RxMode)

Function description

Set Rx mode.

Parameters

- **UCPDx:** UCPD Instance
- **RxMode:** This parameter can be one of the following values:
  - LL_UCPD_RXMODE_NORMAL
  - LL_UCPD_RXMODE_BIST_TEST_DATA

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR RXMODE LL_UCPD_SetRxMode

LL_UCPD_SendHardReset

Function name

__STATIC_INLINE void LL_UCPD_SendHardReset (UCPD_TypeDef * UCPDx)

Function description

Send Hard Reset.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR TXHRST LL_UCPD_SendHardReset

LL_UCPD_SendMessage

Function name

__STATIC_INLINE void LL_UCPD_SendMessage (UCPD_TypeDef * UCPDx)

Function description

Send message.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR TXSEND LL_UCPD_SendMessage

LL_UCPD_SetTxMode

Function name

__STATIC_INLINE void LL_UCPD_SetTxMode (UCPD_TypeDef * UCPDx, uint32_t TxMode)

Function description

Set Tx mode.

Parameters

- **UCPDx:** UCPD Instance
- **TxMode:** This parameter can be one of the following values:
  - LL_UCPD_TXMODE_NORMAL
  - LL_UCPD_TXMODE_CABLE_RESET
  - LL_UCPD_TXMODE_BIST_CARRIER2

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR TXMODE LL_UCPD_SetTxMode

LL_UCPD_EnableIT_TypeCEventCC2

Function name

__STATIC_INLINE void LL_UCPD_EnableIT_TypeCEventCC2 (UCPD_TypeDef * UCPDx)

Function description

Enable type c event on CC2.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR TYPECEVT2IE LL_UCPD_EnableIT_TypeCEventCC2

LL_UCPD_EnableIT_TypeCEventCC1

Function name

__STATIC_INLINE void LL_UCPD_EnableIT_TypeCEventCC1 (UCPD_TypeDef * UCPDx)

Function description

Enable type c event on CC1.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR TYPECEVT1IE LL_UCPD_EnableIT_TypeCEventCC1

LL_UCPD_EnableIT_RxMsgEnd

Function name

__STATIC_INLINE void LL_UCPD_EnableIT_RxMsgEnd (UCPD_TypeDef * UCPDx)

Function description

Enable Rx message end interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR RXMSGENDIE LL_UCPD_EnableIT_RxMsgEnd

LL_UCPD_EnableIT_RxOvr

Function name

__STATIC_INLINE void LL_UCPD_EnableIT_RxOvr (UCPD_TypeDef * UCPDx)

Function description

Enable Rx overrun interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR RXOVRIE LL_UCPD_EnableIT_RxOvr

LL_UCPD_EnableIT_RxHRST

Function name

__STATIC_INLINE void LL_UCPD_EnableIT_RxHRST (UCPD_TypeDef * UCPDx)

Function description

Enable Rx hard reset interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR RXHRSTDETIE LL_UCPD_EnableIT_RxHRST

LL_UCPD_EnableIT_RxOrderSet

Function name

__STATIC_INLINE void LL_UCPD_EnableIT_RxOrderSet (UCPD_TypeDef * UCPDx)

Function description

Enable Rx orderset interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR RXORDDETIE LL_UCPD_EnableIT_RxOrderSet

LL_UCPD_EnableIT_RxNE

Function name

__STATIC_INLINE void LL_UCPD_EnableIT_RxNE (UCPD_TypeDef * UCPDx)

Function description

Enable Rx non empty interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR RXNEIE LL_UCPD_EnableIT_RxNE

LL_UCPD_EnableIT_TxUND

Function name

__STATIC_INLINE void LL_UCPD_EnableIT_TxUND (UCPD_TypeDef * UCPDx)

Function description

Enable TX underrun interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR TXUNDIE LL_UCPD_EnableIT_TxUND

LL_UCPD_EnableIT_TxHRSTSENT

Function name

__STATIC_INLINE void LL_UCPD_EnableIT_TxHRSTSENT (UCPD_TypeDef * UCPDx)

Function description

Enable hard reset sent interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR HRSTSENTIE LL_UCPD_EnableIT_TxHRSTSENT

LL_UCPD_EnableIT_TxHRSTDISC

Function name

__STATIC_INLINE void LL_UCPD_EnableIT_TxHRSTDISC (UCPD_TypeDef * UCPDx)

Function description

Enable hard reset discard interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR HRSTDISCIE LL_UCPD_EnableIT_TxHRSTDISC

LL_UCPD_EnableIT_TxMSGABT

Function name

__STATIC_INLINE void LL_UCPD_EnableIT_TxMSGABT (UCPD_TypeDef * UCPDx)

Function description

Enable Tx message abort interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR TXMSGABTIE LL_UCPD_EnableIT_TxMSGABT

LL_UCPD_EnableIT_TxMSGSENT

Function name

__STATIC_INLINE void LL_UCPD_EnableIT_TxMSGSENT (UCPD_TypeDef * UCPDx)

Function description

Enable Tx message sent interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR TXMSGSENTIE LL_UCPD_EnableIT_TxMSGSENT

LL_UCPD_EnableIT_TxMSGDISC

Function name

__STATIC_INLINE void LL_UCPD_EnableIT_TxMSGDISC (UCPD_TypeDef * UCPDx)

Function description

Enable Tx message discarded interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR TXMSGDISCIE LL_UCPD_EnableIT_TxMSGDISC

LL_UCPD_EnableIT_TxIS

Function name

__STATIC_INLINE void LL_UCPD_EnableIT_TxIS (UCPD_TypeDef * UCPDx)

Function description

Enable Tx data receive interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR TXISIE LL_UCPD_EnableIT_TxIS

LL_UCPD_DisableIT_TypeCEventCC2

Function name

__STATIC_INLINE void LL_UCPD_DisableIT_TypeCEventCC2 (UCPD_TypeDef * UCPDx)

Function description

Disable type c event on CC2.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR TYPECEVT2IE LL_UCPD_DisableIT_TypeCEventCC2

LL_UCPD_DisableIT_TypeCEventCC1

Function name

__STATIC_INLINE void LL_UCPD_DisableIT_TypeCEventCC1 (UCPD_TypeDef * UCPDx)

Function description

Disable type c event on CC1.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR TYPECEVT1IE LL_UCPD_DisableIT_TypeCEventCC1

LL_UCPD_DisableIT_RxMsgEnd

Function name

__STATIC_INLINE void LL_UCPD_DisableIT_RxMsgEnd (UCPD_TypeDef * UCPDx)

Function description

Disable Rx message end interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR RXMSGENDIE LL_UCPD_DisableIT_RxMsgEnd

LL_UCPD_DisableIT_RxOvr

Function name

__STATIC_INLINE void LL_UCPD_DisableIT_RxOvr (UCPD_TypeDef * UCPDx)

Function description

Disable Rx overrun interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR RXOVRIE LL_UCPD_DisableIT_RxOvr

LL_UCPD_DisableIT_RxHRST

Function name

__STATIC_INLINE void LL_UCPD_DisableIT_RxHRST (UCPD_TypeDef * UCPDx)

Function description

Disable Rx hard reset interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR RXHRSTDETIE LL_UCPD_DisableIT_RxHRST

LL_UCPD_DisableIT_RxOrderSet

Function name

__STATIC_INLINE void LL_UCPD_DisableIT_RxOrderSet (UCPD_TypeDef * UCPDx)

Function description

Disable Rx orderset interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR RXORDDETIE LL_UCPD_DisableIT_RxOrderSet

LL_UCPD_DisableIT_RxNE

Function name

__STATIC_INLINE void LL_UCPD_DisableIT_RxNE (UCPD_TypeDef * UCPDx)

Function description

Disable Rx non empty interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR RXNEIE LL_UCPD_DisableIT_RxNE

LL_UCPD_DisableIT_TxUND

Function name

__STATIC_INLINE void LL_UCPD_DisableIT_TxUND (UCPD_TypeDef * UCPDx)

Function description

Disable TX underrun interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR TXUNDIE LL_UCPD_DisableIT_TxUND

LL_UCPD_DisableIT_TxHRSTSENT

Function name

__STATIC_INLINE void LL_UCPD_DisableIT_TxHRSTSENT (UCPD_TypeDef * UCPDx)

Function description

Disable hard reset sent interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR HRSTSENTIE LL_UCPD_DisableIT_TxHRSTSENT

LL_UCPD_DisableIT_TxHRSTDISC

Function name

__STATIC_INLINE void LL_UCPD_DisableIT_TxHRSTDISC (UCPD_TypeDef * UCPDx)

Function description

Disable hard reset discard interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR HRSTDISCIE LL_UCPD_DisableIT_TxHRSTDISC

LL_UCPD_DisableIT_TxMSGABT

Function name

__STATIC_INLINE void LL_UCPD_DisableIT_TxMSGABT (UCPD_TypeDef * UCPDx)

Function description

Disable Tx message abort interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR TXMSGABTIE LL_UCPD_DisableIT_TxMSGABT

LL_UCPD_DisableIT_TxMSGSENT

Function name

__STATIC_INLINE void LL_UCPD_DisableIT_TxMSGSENT (UCPD_TypeDef * UCPDx)

Function description

Disable Tx message sent interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR TXMSGSENTIE LL_UCPD_DisableIT_TxMSGSENT

LL_UCPD_DisableIT_TxMSGDISC

Function name

__STATIC_INLINE void LL_UCPD_DisableIT_TxMSGDISC (UCPD_TypeDef * UCPDx)

Function description

Disable Tx message discarded interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR TXMSGDISCIE LL_UCPD_DisableIT_TxMSGDISC

LL_UCPD_DisableIT_TxIS

Function name

__STATIC_INLINE void LL_UCPD_DisableIT_TxIS (UCPD_TypeDef * UCPDx)

Function description

Disable Tx data receive interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR TXISIE LL_UCPD_DisableIT_TxIS

LL_UCPD_IsEnableIT_TypeCEventCC2

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsEnableIT_TypeCEventCC2 (UCPD_TypeDef const *const UCPDx)

Function description

Check if type c event on CC2 enabled.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IMR TYPECEVT2IE LL_UCPD_DisableIT_TypeCEventCC2

LL_UCPD_IsEnableIT_TypeCEventCC1

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsEnableIT_TypeCEventCC1 (UCPD_TypeDef const *const UCPDx)

Function description

Check if type c event on CC1 enabled.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IMR2 TYPECEVT1IE LL_UCPD_IsEnableIT_TypeCEventCC1

LL_UCPD_IsEnableIT_RxMsgEnd

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsEnableIT_RxMsgEnd (UCPD_TypeDef const *const UCPDx)

Function description

Check if Rx message end interrupt enabled.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IMR RXMSGENDIE LL_UCPD_IsEnableIT_RxMsgEnd

LL_UCPD_IsEnableIT_RxOvr

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsEnableIT_RxOvr (UCPD_TypeDef const *const UCPDx)

Function description

Check if Rx overrun interrupt enabled.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IMR RXOVRIE LL_UCPD_IsEnableIT_RxOvr

LL_UCPD_IsEnableIT_RxHRST

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsEnableIT_RxHRST (UCPD_TypeDef const *const UCPDx)

Function description

Check if Rx hard reset interrupt enabled.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IMR RXHRSTDETIE LL_UCPD_IsEnableIT_RxHRST

LL_UCPD_IsEnableIT_RxOrderSet

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsEnableIT_RxOrderSet (UCPD_TypeDef const *const UCPDx)

Function description

Check if Rx orderset interrupt enabled.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IMR RXORDDETIE LL_UCPD_IsEnableIT_RxOrderSet

LL_UCPD_IsEnableIT_RxNE

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsEnableIT_RxNE (UCPD_TypeDef const *const UCPDx)

Function description

Check if Rx non empty interrupt enabled.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IMR RXNEIE LL_UCPD_IsEnableIT_RxNE

LL_UCPD_IsEnableIT_TxUND

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsEnableIT_TxUND (UCPD_TypeDef const *const UCPDx)

Function description

Check if TX underrun interrupt enabled.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IMR TXUNDIE LL_UCPD_IsEnableIT_TxUND

LL_UCPD_IsEnableIT_TxHRSTSENT

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsEnableIT_TxHRSTSENT (UCPD_TypeDef const *const UCPDx)

Function description

Check if hard reset sent interrupt enabled.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IMR HRSTSENTIE LL_UCPD_IsEnableIT_TxHRSTSENT

LL_UCPD_IsEnableIT_TxHRSTDISC

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsEnableIT_TxHRSTDISC (UCPD_TypeDef const *const UCPDx)

Function description

Check if hard reset discard interrupt enabled.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IMR HRSTDISCIE LL_UCPD_IsEnableIT_TxHRSTDISC

LL_UCPD_IsEnableIT_TxMSGABT

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsEnableIT_TxMSGABT (UCPD_TypeDef const *const UCPDx)

Function description

Check if Tx message abort interrupt enabled.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IMR TXMSGABTIE LL_UCPD_IsEnableIT_TxMSGABT

LL_UCPD_IsEnableIT_TxMSGSENT

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsEnableIT_TxMSGSENT (UCPD_TypeDef const *const UCPDx)

Function description

Check if Tx message sent interrupt enabled.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IMR TXMSGSENTIE LL_UCPD_IsEnableIT_TxMSGSENT

LL_UCPD_IsEnableIT_TxMSGDISC

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsEnableIT_TxMSGDISC (UCPD_TypeDef const *const UCPDx)

Function description

Check if Tx message discarded interrupt enabled.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IMR TXMSGDISCIE LL_UCPD_IsEnableIT_TxMSGDISC

LL_UCPD_IsEnableIT_TxIS

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsEnableIT_TxIS (UCPD_TypeDef const *const UCPDx)

Function description

Check if Tx data receive interrupt enabled.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IMR TXISIE LL_UCPD_IsEnableIT_TxIS

LL_UCPD_ClearFlag_TypeCEventCC2

Function name

__STATIC_INLINE void LL_UCPD_ClearFlag_TypeCEventCC2 (UCPD_TypeDef * UCPDx)

Function description

Clear type c event on CC2.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IIMR TYPECEVT2IE LL_UCPD_ClearFlag_TypeCEventCC2

LL_UCPD_ClearFlag_TypeCEventCC1

Function name

__STATIC_INLINE void LL_UCPD_ClearFlag_TypeCEventCC1 (UCPD_TypeDef * UCPDx)

Function description

Clear type c event on CC1.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IIMR TYPECEVT1IE LL_UCPD_ClearFlag_TypeCEventCC1

LL_UCPD_ClearFlag_RxMsgEnd

Function name

__STATIC_INLINE void LL_UCPD_ClearFlag_RxMsgEnd (UCPD_TypeDef * UCPDx)

Function description

Clear Rx message end interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR RXMSGENDIE LL_UCPD_ClearFlag_RxMsgEnd

LL_UCPD_ClearFlag_RxOvr

Function name

__STATIC_INLINE void LL_UCPD_ClearFlag_RxOvr (UCPD_TypeDef * UCPDx)

Function description

Clear Rx overrun interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR RXOVRIE LL_UCPD_ClearFlag_RxOvr

LL_UCPD_ClearFlag_RxHRST

Function name

__STATIC_INLINE void LL_UCPD_ClearFlag_RxHRST (UCPD_TypeDef * UCPDx)

Function description

Clear Rx hard reset interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR RXHRSTDETIE LL_UCPD_ClearFlag_RxHRST

LL_UCPD_ClearFlag_RxOrderSet

Function name

__STATIC_INLINE void LL_UCPD_ClearFlag_RxOrderSet (UCPD_TypeDef * UCPDx)

Function description

Clear Rx orderset interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR RXORDDETIE LL_UCPD_ClearFlag_RxOrderSet

LL_UCPD_ClearFlag_TxUND

Function name

__STATIC_INLINE void LL_UCPD_ClearFlag_TxUND (UCPD_TypeDef * UCPDx)

Function description

Clear TX underrun interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR TXUNDIE LL_UCPD_ClearFlag_TxUND

LL_UCPD_ClearFlag_TxHRSTSENT

Function name

__STATIC_INLINE void LL_UCPD_ClearFlag_TxHRSTSENT (UCPD_TypeDef * UCPDx)

Function description

Clear hard reset sent interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR HRSTSENTIE LL_UCPD_ClearFlag_TxHRSTSENT

LL_UCPD_ClearFlag_TxHRSTDISC

Function name

__STATIC_INLINE void LL_UCPD_ClearFlag_TxHRSTDISC (UCPD_TypeDef * UCPDx)

Function description

Clear hard reset discard interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR HRSTDISCIE LL_UCPD_ClearFlag_TxHRSTDISC

LL_UCPD_ClearFlag_TxMSGABT

Function name

__STATIC_INLINE void LL_UCPD_ClearFlag_TxMSGABT (UCPD_TypeDef * UCPDx)

Function description

Clear Tx message abort interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR TXMSGABTIE LL_UCPD_ClearFlag_TxMSGABT

LL_UCPD_ClearFlag_TxMSGSENT

Function name

__STATIC_INLINE void LL_UCPD_ClearFlag_TxMSGSENT (UCPD_TypeDef * UCPDx)

Function description

Clear Tx message sent interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR TXMSGSENTIE LL_UCPD_ClearFlag_TxMSGSENT

LL_UCPD_ClearFlag_TxMSGDISC

Function name

__STATIC_INLINE void LL_UCPD_ClearFlag_TxMSGDISC (UCPD_TypeDef * UCPDx)

Function description

Clear Tx message discarded interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR TXMSGDISCIE LL_UCPD_ClearFlag_TxMSGDISC

LL_UCPD_IsActiveFlag_TypeCEventCC2

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsActiveFlag_TypeCEventCC2 (UCPD_TypeDef const *const UCPDx)

Function description

Check if type c event on CC2.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR TYPECEVT2 LL_UCPD_IsActiveFlag_TypeCEventCC2

LL_UCPD_IsActiveFlag_TypeCEventCC1

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsActiveFlag_TypeCEventCC1 (UCPD_TypeDef const *const UCPDx)

Function description

Check if type c event on CC1.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR TYPECEVT1 LL_UCPD_IsActiveFlag_TypeCEventCC1

LL_UCPD_IsActiveFlag_RxErr

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsActiveFlag_RxErr (UCPD_TypeDef const *const UCPDx)

Function description

Check if Rx error flag is active.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR RXERR LL_UCPD_IsActiveFlag_RxErr

LL_UCPD_IsActiveFlag_RxMsgEnd

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsActiveFlag_RxMsgEnd (UCPD_TypeDef const *const UCPDx)

Function description

Check if Rx message end flag is active.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR RXMSGEND LL_UCPD_IsActiveFlag_RxMsgEnd

LL_UCPD_IsActiveFlag_RxOvr

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsActiveFlag_RxOvr (UCPD_TypeDef const *const UCPDx)

Function description

Check if Rx overrun flag is active.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR RXOVR LL_UCPD_IsActiveFlag_RxOvr

LL_UCPD_IsActiveFlag_RxHRST

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsActiveFlag_RxHRST (UCPD_TypeDef const *const UCPDx)

Function description

Check if Rx hard reset flag is active.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR RXHRSTDET LL_UCPD_IsActiveFlag_RxHRST

LL_UCPD_IsActiveFlag_RxOrderSet

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsActiveFlag_RxOrderSet (UCPD_TypeDef const *const UCPDx)

Function description

Check if Rx orderset flag is active.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR RXORDDET LL_UCPD_IsActiveFlag_RxOrderSet

LL_UCPD_IsActiveFlag_RxNE

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsActiveFlag_RxNE (UCPD_TypeDef const *const UCPDx)

Function description

Check if Rx non empty flag is active.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR RXNE LL_UCPD_IsActiveFlag_RxNE

LL_UCPD_IsActiveFlag_TxUND

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsActiveFlag_TxUND (UCPD_TypeDef const *const UCPDx)

Function description

Check if TX underrun flag is active.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR TXUND LL_UCPD_IsActiveFlag_TxUND

LL_UCPD_IsActiveFlag_TxHRSTSENT

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsActiveFlag_TxHRSTSENT (UCPD_TypeDef const *const UCPDx)

Function description

Check if hard reset sent flag is active.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR HRSTSENT LL_UCPD_IsActiveFlag_TxHRSTSENT

LL_UCPD_IsActiveFlag_TxHRSTDISC

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsActiveFlag_TxHRSTDISC (UCPD_TypeDef const *const UCPDx)

Function description

Check if hard reset discard flag is active.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR HRSTDISC LL_UCPD_IsActiveFlag_TxHRSTDISC

LL_UCPD_IsActiveFlag_TxMSGABT

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsActiveFlag_TxMSGABT (UCPD_TypeDef const *const UCPDx)

Function description

Check if Tx message abort flag is active.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR TXMSGABT LL_UCPD_IsActiveFlag_TxMSGABT

LL_UCPD_IsActiveFlag_TxMSGSENT

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsActiveFlag_TxMSGSENT (UCPD_TypeDef const *const UCPDx)

Function description

Check if Tx message sent flag is active.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR TXMSGSENT LL_UCPD_IsActiveFlag_TxMSGSENT

LL_UCPD_IsActiveFlag_TxMSGDISC

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsActiveFlag_TxMSGDISC (UCPD_TypeDef const *const UCPDx)

Function description

Check if Tx message discarded flag is active.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR TXMSGDISC LL_UCPD_IsActiveFlag_TxMSGDISC

LL_UCPD_IsActiveFlag_TxIS

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsActiveFlag_TxIS (UCPD_TypeDef const *const UCPDx)

Function description

Check if Tx data interrupt flag is active.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR TXIS LL_UCPD_IsActiveFlag_TxIS

LL_UCPD_GetTypeCVstateCC2

Function name

__STATIC_INLINE uint32_t LL_UCPD_GetTypeCVstateCC2 (UCPD_TypeDef const *const UCPDx)

Function description

return the vstate value for CC2

Parameters

- **UCPDx:** UCPD Instance

Return values

- **val:**

Reference Manual to LL API cross reference:

- SR TXIS LL_UCPD_GetTypeCVstateCC2

LL_UCPD_GetTypeCVstateCC1

Function name

__STATIC_INLINE uint32_t LL_UCPD_GetTypeCVstateCC1 (UCPD_TypeDef const *const UCPDx)

Function description

return the vstate value for CC1

Parameters

- **UCPDx:** UCPD Instance

Return values

- **val:**

Reference Manual to LL API cross reference:

- SR TXIS LL_UCPD_GetTypeCVstateCC1

LL_UCPD_RxDMAEnable

Function name

__STATIC_INLINE void LL_UCPD_RxDMAEnable (UCPD_TypeDef * UCPDx)

Function description

Rx DMA Enable.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG1 RXDMAEN LL_UCPD_RxDMAEnable

LL_UCPD_RxDMADisable

Function name

__STATIC_INLINE void LL_UCPD_RxDMADisable (UCPD_TypeDef * UCPDx)

Function description

Rx DMA Disable.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG1 RXDMAEN LL_UCPD_RxDMADisable

LL_UCPD_TxDMAEnable

Function name

__STATIC_INLINE void LL_UCPD_TxDMAEnable (UCPD_TypeDef * UCPDx)

Function description

Tx DMA Enable.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG1 TXDMAEN LL_UCPD_TxDMAEnable

LL_UCPD_TxDMADisable

Function name

__STATIC_INLINE void LL_UCPD_TxDMADisable (UCPD_TypeDef * UCPDx)

Function description

Tx DMA Disable.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG1 TXDMAEN LL_UCPD_TxDMADisable

LL_UCPD_IsEnabledTxDMA

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsEnabledTxDMA (UCPD_TypeDef const *const UCPDx)

Function description

Check if DMA Tx is enabled.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR2 TXDMAEN LL_UCPD_IsEnabledTxDMA

LL_UCPD_IsEnabledRxDMA

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsEnabledRxDMA (UCPD_TypeDef const *const UCPDx)

Function description

Check if DMA Rx is enabled.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR2 RXDMAEN LL_UCPD_IsEnabledRxDMA

LL_UCPD_WriteTxOrderSet

Function name

__STATIC_INLINE void LL_UCPD_WriteTxOrderSet (UCPD_TypeDef * UCPDx, uint32_t TxOrderSet)

Function description

write the orderset for Tx message

Parameters

- **UCPDx:** UCPD Instance
- **TxOrderSet:** one of the following value
  - LL_UCPD_ORDERED_SET_SOP
  - LL_UCPD_ORDERED_SET_SOP1
  - LL_UCPD_ORDERED_SET_SOP2
  - LL_UCPD_ORDERED_SET_HARD_RESET
  - LL_UCPD_ORDERED_SET_CABLE_RESET
  - LL_UCPD_ORDERED_SET_SOP1_DEBUG
  - LL_UCPD_ORDERED_SET_SOP2_DEBUG

Return values

- **None:**

Reference Manual to LL API cross reference:

- TX_ORDSET TXORDSET LL_UCPD_WriteTxOrderSet

LL_UCPD_WriteTxPaySize

Function name

__STATIC_INLINE void LL_UCPD_WriteTxPaySize (UCPD_TypeDef * UCPDx, uint32_t TxPaySize)

Function description

write the Tx paysize

Parameters

- **UCPDx:** UCPD Instance
- **TxPaySize:**

Return values

- **None.:**

Reference Manual to LL API cross reference:

- TX_PAYSZ TXPAYSZ LL_UCPD_WriteTxPaySize

LL_UCPD_WriteData

Function name

__STATIC_INLINE void LL_UCPD_WriteData (UCPD_TypeDef * UCPDx, uint8_t Data)

Function description

Write data.

Parameters

- **UCPDx:** UCPD Instance
- **Data:** Value between Min_Data=0x00 and Max_Data=0xFF

Return values

- **None.:**

Reference Manual to LL API cross reference:

- TXDR DR LL_UCPD_WriteData

LL_UCPD_ReadRxOrderSet

Function name

__STATIC_INLINE uint32_t LL_UCPD_ReadRxOrderSet (UCPD_TypeDef const *const UCPDx)

Function description

read RX the orderset

Parameters

- **UCPDx:** UCPD Instance

Return values

- **RxOrderSet:** one of the following value
  - LL_UCPD_RXORDSET_SOP
  - LL_UCPD_RXORDSET_SOP1
  - LL_UCPD_RXORDSET_SOP2
  - LL_UCPD_RXORDSET_SOP1_DEBUG
  - LL_UCPD_RXORDSET_SOP2_DEBUG
  - LL_UCPD_RXORDSET_CABLE_RESET
  - LL_UCPD_RXORDSET_SOPEXT1
  - LL_UCPD_RXORDSET_SOPEXT2

Reference Manual to LL API cross reference:

- RX_ORDSET RXORDSET LL_UCPD_ReadRxOrderSet

LL_UCPD_ReadRxPaySize

Function name

__STATIC_INLINE uint32_t LL_UCPD_ReadRxPaySize (UCPD_TypeDef const *const UCPDx)

Function description

Read the Rx paysize.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **RXPaysize.:**

Reference Manual to LL API cross reference:

- RX_PAYSZ RXPAYSZ LL_UCPD_ReadRxPaySize

LL_UCPD_ReadData

Function name

__STATIC_INLINE uint32_t LL_UCPD_ReadData (UCPD_TypeDef const *const UCPDx)

Function description

Read data.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **RxData:** Value between Min_Data=0x00 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- RXDR RXDATA LL_UCPD_ReadData

LL_UCPD_SetRxOrdExt1

Function name

__STATIC_INLINE void LL_UCPD_SetRxOrdExt1 (UCPD_TypeDef * UCPDx, uint32_t SOPExt)

Function description

Set Rx OrderSet Ext1.

Parameters

- **UCPDx:** UCPD Instance
- **SOPExt:** Value between Min_Data=0x00000 and Max_Data=0xFFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- RX_ORDEXT1 RXSOPX1 LL_UCPD_SetRxOrdExt1

LL_UCPD_SetRxOrdExt2

Function name

__STATIC_INLINE void LL_UCPD_SetRxOrdExt2 (UCPD_TypeDef * UCPDx, uint32_t SOPExt)

Function description

Set Rx OrderSet Ext2.

Parameters

- **UCPDx:** UCPD Instance
- **SOPExt:** Value between Min_Data=0x00000 and Max_Data=0xFFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- RX_ORDEXT2 RXSOPX2 LL_UCPD_SetRxOrdExt2

LL_UCPD_DeInit

Function name

ErrorStatus LL_UCPD_DeInit (UCPD_TypeDef * UCPDx)

Function description

De-initialize the UCPD registers to their default reset values.

Parameters

- **UCPDx:** ucpd Instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: ucpd registers are de-initialized
  - ERROR: ucpd registers are not de-initialized

LL_UCPD_Init

Function name

ErrorStatus LL_UCPD_Init (UCPD_TypeDef * UCPDx, const LL_UCPD_InitTypeDef * UCPD_InitStruct)

Function description

Initialize the ucpd registers according to the specified parameters in UCPD_InitStruct.

Parameters

- **UCPDx:** UCPD Instance
- **UCPD_InitStruct:** pointer to a LL_UCPD_InitTypeDef structure that contains the configuration information for the UCPD peripheral.

Return values

- **An:** ErrorStatus enumeration value. (Return always SUCCESS)

Notes

- As some bits in ucpd configuration registers can only be written when the ucpd is disabled (ucpd_CR1_SPE bit =0), UCPD peripheral should be in disabled state prior calling this function. Otherwise, ERROR result will be returned.

LL_UCPD_StructInit

Function name

void LL_UCPD_StructInit (LL_UCPD_InitTypeDef * UCPD_InitStruct)

Function description

Set each LL_UCPD_InitTypeDef field to default value.

Parameters

- **UCPD_InitStruct:** pointer to a LL_UCPD_InitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

107.3 UCPD Firmware driver defines

The following section lists the various define and macros of the module.

107.3.1 UCPD

UCPD

CC pin enable

LL_UCPD_CCENABLE_NONE

Neither PHY is activated (e.g. disabled state of source)

LL_UCPD_CCENABLE_CC1

Controls apply to only CC1

LL_UCPD_CCENABLE_CC2

Controls apply to only CC1

LL_UCPD_CCENABLE_CC1CC2

Controls apply to both CC1 and CC2 (normal usage for sink/source)

CC pin selection

LL_UCPD_CCPIN_CC1

Use CC1 IO for power delivery communication

LL_UCPD_CCPIN_CC2

Use CC2 IO for power delivery communication

CCx event

LL_UCPD_SNK_CC1_VOPEN

CC1 Sink Open state

LL_UCPD_SNK_CC1_VRP

CC1 Sink vRP default state

LL_UCPD_SNK_CC1_VRP15A

CC1 Sink vRP 1.5A state

LL_UCPD_SNK_CC1_VRP30A

CC1 Sink vRP 3.0A state

LL_UCPD_SNK_CC2_VOPEN

CC2 Sink Open state

LL_UCPD_SNK_CC2_VRP

CC2 Sink vRP default state

LL_UCPD_SNK_CC2_VRP15A

CC2 Sink vRP 1.5A state

LL_UCPD_SNK_CC2_VRP30A

CC2 Sink vRP 3.0A state

LL_UCPD_SRC_CC1_VRA

CC1 Source vRA state

LL_UCPD_SRC_CC1_VRD

CC1 Source vRD state

LL_UCPD_SRC_CC1_OPEN

CC1 Source Open state

LL_UCPD_SRC_CC2_VRA

CC2 Source vRA state

LL_UCPD_SRC_CC2_VRD

CC2 Source vRD state

LL_UCPD_SRC_CC2_OPEN

CC2 Source Open state

ordered set configuration

LL_UCPD_ORDERSET_SOP

SOP Ordered set detection enabled

LL_UCPD_ORDERSET_SOP1

SOP' Ordered set detection enabled

LL_UCPD_ORDERSET_SOP2

SOP'' Ordered set detection enabled

LL_UCPD_ORDERSET_HARDRST

Hard Reset Ordered set detection enabled

LL_UCPD_ORDERSET_CABLERST

Cable Reset Ordered set detection enabled

LL_UCPD_ORDERSET_SOP1_DEBUG

SOP' Debug Ordered set detection enabled

LL_UCPD_ORDERSET_SOP2_DEBUG

SOP'' Debug Ordered set detection enabled

LL_UCPD_ORDERSET_SOP_EXT1

SOP extension#1 Ordered set detection enabled

LL_UCPD_ORDERSET_SOP_EXT2

SOP extension#2 Ordered set detection enabled

Get Flags Defines

LL_UCPD_SR_TXIS

Transmit interrupt status

LL_UCPD_SR_TXMSGDISC

Transmit message discarded interrupt

LL_UCPD_SR_TXMSGSENT

Transmit message sent interrupt

LL_UCPD_SR_TXMSGABT

Transmit message abort interrupt

LL_UCPD_SR_HRSTDISC

HRST discarded interrupt

LL_UCPD_SR_HRSTSENT

HRST sent interrupt

LL_UCPD_SR_TXUND

Tx data underrun condition interrupt

LL_UCPD_SR_RXNE

Receive data register not empty interrupt

LL_UCPD_SR_RXORDDET

Rx ordered set (4 K-codes) detected interrupt

LL_UCPD_SR_RXHRSTDET

Rx Hard Reset detect interrupt

LL_UCPD_SR_RXOVR

Rx data overflow interrupt

LL_UCPD_SR_RXMSGEND

Rx message received

LL_UCPD_SR_RXERR

Rx error

LL_UCPD_SR_TYPECEVT1

Type C voltage level event on CC1

LL_UCPD_SR_TYPECEVT2

Type C voltage level event on CC2

LL_UCPD_SR_TYPEC_VSTATE_CC1

Status of DC level on CC1 pin

LL_UCPD_SR_TYPEC_VSTATE_CC2

Status of DC level on CC2 pin

IT Defines

LL_UCPD_IMR_TXIS

Enable transmit interrupt status

LL_UCPD_IMR_TXMSGDISC

Enable transmit message discarded interrupt

LL_UCPD_IMR_TXMSGSENT

Enable transmit message sent interrupt

LL_UCPD_IMR_TXMSGABT

Enable transmit message abort interrupt

LL_UCPD_IMR_HRSTDISC

Enable HRST discarded interrupt

LL_UCPD_IMR_HRSTSENT

Enable HRST sent interrupt

LL_UCPD_IMR_TXUND

Enable tx data underrun condition interrupt

LL_UCPD_IMR_RXNE

Enable Receive data register not empty interrupt

LL_UCPD_IMR_RXORDDET

Enable Rx ordered set (4 K-codes) detected interrupt

LL_UCPD_IMR_RXHRSTDET

Enable Rx Hard Reset detect interrupt

LL_UCPD_IMR_RXOVR

Enable Rx data overflow interrupt

LL_UCPD_IMR_RXMSGEND

Enable Rx message received

LL_UCPD_IMR_TYPECEVT1

Enable Type C voltage level event on CC1

LL_UCPD_IMR_TYPECEVT2

Enable Type C voltage level event on CC2

Role Mode

LL_UCPD_ROLE_SNK

Mode SNK Rd

LL_UCPD_ROLE_SRC

Mode SRC Rp

Ordered sets value

LL_UCPD_SYNC1

K-code for Startsynch #1

LL_UCPD_SYNC2

K-code for Startsynch #2

LL_UCPD_SYNC3

K-code for Startsynch #3

LL_UCPD_RST1

K-code for Hard Reset #1

LL_UCPD_RST2

K-code for Hard Reset #2

LL_UCPD_EOP

K-code for EOP End of Packet

LL_UCPD_ORDERED_SET_SOP

SOP Ordered set coding

LL_UCPD_ORDERED_SET_SOP1

SOP' Ordered set coding

LL_UCPD_ORDERED_SET_SOP2

SOP'' Ordered set coding

LL_UCPD_ORDERED_SET_HARD_RESET

Hard Reset Ordered set coding

LL_UCPD_ORDERED_SET_CABLE_RESET

Cable Reset Ordered set coding

LL_UCPD_ORDERED_SET_SOP1_DEBUG

SOP' Debug Ordered set coding

LL_UCPD_ORDERED_SET_SOP2_DEBUG

SOP'' Debug Ordered set coding

prescaler for UCPDCLK

LL_UCPD_PSC_DIV1

Bypass pre-scaling / divide by 1

LL_UCPD_PSC_DIV2

Pre-scale clock by dividing by 2

LL_UCPD_PSC_DIV4

Pre-scale clock by dividing by 4

LL_UCPD_PSC_DIV8

Pre-scale clock by dividing by 8

LL_UCPD_PSC_DIV16

Pre-scale clock by dividing by 16

Resistor value

LL_UCPD_RESISTOR_DEFAULT

Rp default

LL_UCPD_RESISTOR_1_5A

Rp 1.5 A

LL_UCPD_RESISTOR_3_0A

Rp 3.0 A

LL_UCPD_RESISTOR_NONE

No resistor

Receiver mode

LL_UCPD_RXMODE_NORMAL

Normal receive mode

LL_UCPD_RXMODE_BIST_TEST_DATA

BIST receive mode (BIST Test Data Mode)

Rx ordered set code detected

LL_UCPD_RXORDSET_SOP

SOP code detected in receiver

LL_UCPD_RXORDSET_SOP1

SOP' code detected in receiver

LL_UCPD_RXORDSET_SOP2

SOP'' code detected in receiver

LL_UCPD_RXORDSET_SOP1_DEBUG

SOP' Debug code detected in receiver

LL_UCPD_RXORDSET_SOP2_DEBUG

SOP'' Debug code detected in receiver

LL_UCPD_RXORDSET_CABLE_RESET

Cable Reset code detected in receiver

LL_UCPD_RXORDSET_SOPEXT1

SOP extension#1 code detected in receiver

LL_UCPD_RXORDSET_SOPEXT2

SOP extension#2 code detected in receiver

Type of Tx packet

LL_UCPD_TXMODE_NORMAL

Initiate the transfer of a Tx message

LL_UCPD_TXMODE_CABLE_RESET

Trigger a the transfer of a Cable Reset sequence

LL_UCPD_TXMODE_BIST_CARRIER2

Trigger a BIST test sequence send (BIST Carrier Mode 2)

Common Write and read registers Macros

LL_UCPD_WriteReg

Description:

- Write a value in UCPD register.

Parameters:

- __INSTANCE__: UCPD Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_UCPD_ReadReg

Description:

- Read a value in UCPD register.

Parameters:

- __INSTANCE__: UCPD Instance
- __REG__: Register to be read

Return value:

- Register: value

108 LL USART Generic Driver

108.1 USART Firmware driver registers structures

108.1.1 LL_USART_InitTypeDef

LL_USART_InitTypeDef is defined in the stm32n6xx_ll_usart.h

Data Fields

- ***uint32_t PrescalerValue***
- ***uint32_t BaudRate***
- ***uint32_t DataWidth***
- ***uint32_t StopBits***
- ***uint32_t Parity***
- ***uint32_t TransferDirection***
- ***uint32_t HardwareFlowControl***
- ***uint32_t OverSampling***

Field Documentation

- ***uint32_t LL_USART_InitTypeDef::PrescalerValue***
  Specifies the Prescaler to compute the communication baud rate. This parameter can be a value of ***USART_LL_EC_PRESCALER***.This feature can be modified afterwards using unitary function **LL_USART_SetPrescaler()**.
- ***uint32_t LL_USART_InitTypeDef::BaudRate***
  This field defines expected Usart communication baud rate.This feature can be modified afterwards using unitary function **LL_USART_SetBaudRate()**.
- ***uint32_t LL_USART_InitTypeDef::DataWidth***
  Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of ***USART_LL_EC_DATAWIDTH***.This feature can be modified afterwards using unitary function **LL_USART_SetDataWidth()**.
- ***uint32_t LL_USART_InitTypeDef::StopBits***
  Specifies the number of stop bits transmitted. This parameter can be a value of ***USART_LL_EC_STOPBITS***.This feature can be modified afterwards using unitary function **LL_USART_SetStopBitsLength()**.
- ***uint32_t LL_USART_InitTypeDef::Parity***
  Specifies the parity mode. This parameter can be a value of ***USART_LL_EC_PARITY***.This feature can be modified afterwards using unitary function **LL_USART_SetParity()**.
- ***uint32_t LL_USART_InitTypeDef::TransferDirection***
  Specifies whether the Receive and/or Transmit mode is enabled or disabled. This parameter can be a value of ***USART_LL_EC_DIRECTION***.This feature can be modified afterwards using unitary function **LL_USART_SetTransferDirection()**.
- ***uint32_t LL_USART_InitTypeDef::HardwareFlowControl***
  Specifies whether the hardware flow control mode is enabled or disabled. This parameter can be a value of ***USART_LL_EC_HWCONTROL***.This feature can be modified afterwards using unitary function **LL_USART_SetHWFlowCtrl()**.
- ***uint32_t LL_USART_InitTypeDef::OverSampling***
  Specifies whether USART oversampling mode is 16 or 8. This parameter can be a value of ***USART_LL_EC_OVERSAMPLING***.This feature can be modified afterwards using unitary function **LL_USART_SetOverSampling()**.

108.1.2 LL_USART_ClockInitTypeDef

LL_USART_ClockInitTypeDef is defined in the stm32n6xx_ll_usart.h

Data Fields

- ***uint32_t ClockOutput***
- ***uint32_t ClockPolarity***
- ***uint32_t ClockPhase***

- ***uint32_t LastBitClockPulse***

Field Documentation

- ***uint32_t LL_USART_ClockInitTypeDef::ClockOutput***
  Specifies whether the USART clock is enabled or disabled. This parameter can be a value of ***USART_LL_EC_CLOCK***.USART HW configuration can be modified afterwards using unitary functions **LL_USART_EnableSCLKOutput()** or **LL_USART_DisableSCLKOutput()**. For more details, refer to description of this function.
- ***uint32_t LL_USART_ClockInitTypeDef::ClockPolarity***
  Specifies the steady state of the serial clock. This parameter can be a value of ***USART_LL_EC_POLARITY***.USART HW configuration can be modified afterwards using unitary functions **LL_USART_SetClockPolarity()**. For more details, refer to description of this function.
- ***uint32_t LL_USART_ClockInitTypeDef::ClockPhase***
  Specifies the clock transition on which the bit capture is made. This parameter can be a value of ***USART_LL_EC_PHASE***.USART HW configuration can be modified afterwards using unitary functions **LL_USART_SetClockPhase()**. For more details, refer to description of this function.
- ***uint32_t LL_USART_ClockInitTypeDef::LastBitClockPulse***
  Specifies whether the clock pulse corresponding to the last transmitted data bit (MSB) has to be output on the SCLK pin in synchronous mode. This parameter can be a value of ***USART_LL_EC_LASTCLKPULSE***.USART HW configuration can be modified afterwards using unitary functions **LL_USART_SetLastClkPulseOutput()**. For more details, refer to description of this function.

108.2 USART Firmware driver API description

The following section lists the various functions of the USART library.

108.2.1 Detailed description of functions

LL_USART_Enable

Function name

__STATIC_INLINE void LL_USART_Enable (USART_TypeDef * USARTx)

Function description

USART Enable.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 UE LL_USART_Enable

LL_USART_Disable

Function name

__STATIC_INLINE void LL_USART_Disable (USART_TypeDef * USARTx)

Function description

USART Disable (all USART prescalers and outputs are disabled)

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- When USART is disabled, USART prescalers and outputs are stopped immediately, and current operations are discarded. The configuration of the USART is kept, but all the status flags, in the USARTx_ISR are set to their default values.

Reference Manual to LL API cross reference:

- CR1 UE LL_USART_Disable

LL_USART_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabled (const USART_TypeDef * USARTx)

Function description

Indicate if USART is enabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 UE LL_USART_IsEnabled

LL_USART_EnableFIFO

Function name

__STATIC_INLINE void LL_USART_EnableFIFO (USART_TypeDef * USARTx)

Function description

FIFO Mode Enable.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 FIFOEN LL_USART_EnableFIFO

LL_USART_DisableFIFO

Function name

__STATIC_INLINE void LL_USART_DisableFIFO (USART_TypeDef * USARTx)

Function description

FIFO Mode Disable.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 FIFOEN LL_USART_DisableFIFO

LL_USART_IsEnabledFIFO

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledFIFO (const USART_TypeDef * USARTx)

Function description

Indicate if FIFO Mode is enabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 FIFOEN LL_USART_IsEnabledFIFO

LL_USART_SetTXFIFOThreshold

Function name

__STATIC_INLINE void LL_USART_SetTXFIFOThreshold (USART_TypeDef * USARTx, uint32_t Threshold)

Function description

Configure TX FIFO Threshold.

Parameters

- **USARTx:** USART Instance
- **Threshold:** This parameter can be one of the following values:
  - LL_USART_FIFOTHRESHOLD_1_8
  - LL_USART_FIFOTHRESHOLD_1_4
  - LL_USART_FIFOTHRESHOLD_1_2
  - LL_USART_FIFOTHRESHOLD_3_4
  - LL_USART_FIFOTHRESHOLD_7_8
  - LL_USART_FIFOTHRESHOLD_8_8

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 TXFTCFG LL_USART_SetTXFIFOThreshold

LL_USART_GetTXFIFOThreshold

Function name

__STATIC_INLINE uint32_t LL_USART_GetTXFIFOThreshold (const USART_TypeDef * USARTx)

Function description

Return TX FIFO Threshold Configuration.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_FIFOTHRESHOLD_1_8
  - LL_USART_FIFOTHRESHOLD_1_4
  - LL_USART_FIFOTHRESHOLD_1_2
  - LL_USART_FIFOTHRESHOLD_3_4
  - LL_USART_FIFOTHRESHOLD_7_8
  - LL_USART_FIFOTHRESHOLD_8_8

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 TXFTCFG LL_USART_GetTXFIFOThreshold

LL_USART_SetRXFIFOThreshold

Function name

__STATIC_INLINE void LL_USART_SetRXFIFOThreshold (USART_TypeDef * USARTx, uint32_t Threshold)

Function description

Configure RX FIFO Threshold.

Parameters

- **USARTx:** USART Instance
- **Threshold:** This parameter can be one of the following values:
  - LL_USART_FIFOTHRESHOLD_1_8
  - LL_USART_FIFOTHRESHOLD_1_4
  - LL_USART_FIFOTHRESHOLD_1_2
  - LL_USART_FIFOTHRESHOLD_3_4
  - LL_USART_FIFOTHRESHOLD_7_8
  - LL_USART_FIFOTHRESHOLD_8_8

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 RXFTCFG LL_USART_SetRXFIFOThreshold

LL_USART_GetRXFIFOThreshold

Function name

__STATIC_INLINE uint32_t LL_USART_GetRXFIFOThreshold (const USART_TypeDef * USARTx)

Function description

Return RX FIFO Threshold Configuration.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_FIFOTHRESHOLD_1_8
  - LL_USART_FIFOTHRESHOLD_1_4
  - LL_USART_FIFOTHRESHOLD_1_2
  - LL_USART_FIFOTHRESHOLD_3_4
  - LL_USART_FIFOTHRESHOLD_7_8
  - LL_USART_FIFOTHRESHOLD_8_8

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 RXFTCFG LL_USART_GetRXFIFOThreshold

LL_USART_ConfigFIFOsThreshold

Function name

__STATIC_INLINE void LL_USART_ConfigFIFOsThreshold (USART_TypeDef * USARTx, uint32_t TXThreshold, uint32_t RXThreshold)

Function description

Configure TX and RX FIFOs Threshold.

Parameters

- **USARTx:** USART Instance
- **TXThreshold:** This parameter can be one of the following values:
  - LL_USART_FIFOTHRESHOLD_1_8
  - LL_USART_FIFOTHRESHOLD_1_4
  - LL_USART_FIFOTHRESHOLD_1_2
  - LL_USART_FIFOTHRESHOLD_3_4
  - LL_USART_FIFOTHRESHOLD_7_8
  - LL_USART_FIFOTHRESHOLD_8_8
- **RXThreshold:** This parameter can be one of the following values:
  - LL_USART_FIFOTHRESHOLD_1_8
  - LL_USART_FIFOTHRESHOLD_1_4
  - LL_USART_FIFOTHRESHOLD_1_2
  - LL_USART_FIFOTHRESHOLD_3_4
  - LL_USART_FIFOTHRESHOLD_7_8
  - LL_USART_FIFOTHRESHOLD_8_8

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 TXFTCFG LL_USART_ConfigFIFOsThreshold
- CR3 RXFTCFG LL_USART_ConfigFIFOsThreshold

LL_USART_EnableInStopMode

Function name

__STATIC_INLINE void LL_USART_EnableInStopMode (USART_TypeDef * USARTx)

Function description

USART enabled in STOP Mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- When this function is enabled, USART is able to wake up the MCU from Stop mode, provided that USART clock selection is HSI or LSE in RCC.
- Macro IS_UART_WAKEUP_FROMSTOP_INSTANCE(USARTx) can be used to check whether or not Wake-up from Stop mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 UESM LL_USART_EnableInStopMode

LL_USART_DisableInStopMode

Function name

__STATIC_INLINE void LL_USART_DisableInStopMode (USART_TypeDef * USARTx)

Function description

USART disabled in STOP Mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- When this function is disabled, USART is not able to wake up the MCU from Stop mode
- Macro IS_UART_WAKEUP_FROMSTOP_INSTANCE(USARTx) can be used to check whether or not Wake-up from Stop mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 UESM LL_USART_DisableInStopMode

LL_USART_IsEnabledInStopMode

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledInStopMode (const USART_TypeDef * USARTx)

Function description

Indicate if USART is enabled in STOP Mode (able to wake up MCU from Stop mode or not)

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_WAKEUP_FROMSTOP_INSTANCE(USARTx) can be used to check whether or not Wake-up from Stop mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 UESM LL_USART_IsEnabledInStopMode

LL_USART_EnableDirectionRx

Function name

__STATIC_INLINE void LL_USART_EnableDirectionRx (USART_TypeDef * USARTx)

Function description

Receiver Enable (Receiver is enabled and begins searching for a start bit)

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RE LL_USART_EnableDirectionRx

LL_USART_DisableDirectionRx

Function name

__STATIC_INLINE void LL_USART_DisableDirectionRx (USART_TypeDef * USARTx)

Function description

Receiver Disable.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RE LL_USART_DisableDirectionRx

LL_USART_EnableDirectionTx

Function name

__STATIC_INLINE void LL_USART_EnableDirectionTx (USART_TypeDef * USARTx)

Function description

Transmitter Enable.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TE LL_USART_EnableDirectionTx

LL_USART_DisableDirectionTx

Function name

__STATIC_INLINE void LL_USART_DisableDirectionTx (USART_TypeDef * USARTx)

Function description

Transmitter Disable.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TE LL_USART_DisableDirectionTx

LL_USART_SetTransferDirection

Function name

__STATIC_INLINE void LL_USART_SetTransferDirection (USART_TypeDef * USARTx, uint32_t TransferDirection)

Function description

Configure simultaneously enabled/disabled states of Transmitter and Receiver.

Parameters

- **USARTx:** USART Instance
- **TransferDirection:** This parameter can be one of the following values:
  - LL_USART_DIRECTION_NONE
  - LL_USART_DIRECTION_RX
  - LL_USART_DIRECTION_TX
  - LL_USART_DIRECTION_TX_RX

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RE LL_USART_SetTransferDirection
- CR1 TE LL_USART_SetTransferDirection

LL_USART_GetTransferDirection

Function name

__STATIC_INLINE uint32_t LL_USART_GetTransferDirection (const USART_TypeDef * USARTx)

Function description

Return enabled/disabled states of Transmitter and Receiver.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_DIRECTION_NONE
  - LL_USART_DIRECTION_RX
  - LL_USART_DIRECTION_TX
  - LL_USART_DIRECTION_TX_RX

Reference Manual to LL API cross reference:

- CR1 RE LL_USART_GetTransferDirection
- CR1 TE LL_USART_GetTransferDirection

LL_USART_SetParity

Function name

__STATIC_INLINE void LL_USART_SetParity (USART_TypeDef * USARTx, uint32_t Parity)

Function description

Configure Parity (enabled/disabled and parity mode if enabled).

Parameters

- **USARTx:** USART Instance
- **Parity:** This parameter can be one of the following values:
  - LL_USART_PARITY_NONE
  - LL_USART_PARITY_EVEN
  - LL_USART_PARITY_ODD

Return values

- **None:**

Notes

- This function selects if hardware parity control (generation and detection) is enabled or disabled. When the parity control is enabled (Odd or Even), computed parity bit is inserted at the MSB position (9th or 8th bit depending on data width) and parity is checked on the received data.

Reference Manual to LL API cross reference:

- CR1 PS LL_USART_SetParity
- CR1 PCE LL_USART_SetParity

LL_USART_GetParity

Function name

__STATIC_INLINE uint32_t LL_USART_GetParity (const USART_TypeDef * USARTx)

Function description

Return Parity configuration (enabled/disabled and parity mode if enabled)

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_PARITY_NONE
  - LL_USART_PARITY_EVEN
  - LL_USART_PARITY_ODD

Reference Manual to LL API cross reference:

- CR1 PS LL_USART_GetParity
- CR1 PCE LL_USART_GetParity

LL_USART_SetWakeUpMethod

Function name

__STATIC_INLINE void LL_USART_SetWakeUpMethod (USART_TypeDef * USARTx, uint32_t Method)

Function description

Set Receiver Wake Up method from Mute mode.

Parameters

- **USARTx:** USART Instance
- **Method:** This parameter can be one of the following values:
  - LL_USART_WAKEUP_IDLELINE
  - LL_USART_WAKEUP_ADDRESSMARK

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 WAKE LL_USART_SetWakeUpMethod

LL_USART_GetWakeUpMethod

Function name

__STATIC_INLINE uint32_t LL_USART_GetWakeUpMethod (const USART_TypeDef * USARTx)

Function description

Return Receiver Wake Up method from Mute mode.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_WAKEUP_IDLELINE
  - LL_USART_WAKEUP_ADDRESSMARK

Reference Manual to LL API cross reference:

- CR1 WAKE LL_USART_GetWakeUpMethod

LL_USART_SetDataWidth

Function name

__STATIC_INLINE void LL_USART_SetDataWidth (USART_TypeDef * USARTx, uint32_t DataWidth)

Function description

Set Word length (i.e.

Parameters

- **USARTx:** USART Instance
- **DataWidth:** This parameter can be one of the following values:
  - LL_USART_DATAWIDTH_7B
  - LL_USART_DATAWIDTH_8B
  - LL_USART_DATAWIDTH_9B

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 M0 LL_USART_SetDataWidth
- CR1 M1 LL_USART_SetDataWidth

LL_USART_GetDataWidth

Function name

__STATIC_INLINE uint32_t LL_USART_GetDataWidth (const USART_TypeDef * USARTx)

Function description

Return Word length (i.e.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_DATAWIDTH_7B
  - LL_USART_DATAWIDTH_8B
  - LL_USART_DATAWIDTH_9B

Reference Manual to LL API cross reference:

- CR1 M0 LL_USART_GetDataWidth
- CR1 M1 LL_USART_GetDataWidth

LL_USART_EnableMuteMode

Function name

__STATIC_INLINE void LL_USART_EnableMuteMode (USART_TypeDef * USARTx)

Function description

Allow switch between Mute Mode and Active mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 MME LL_USART_EnableMuteMode

LL_USART_DisableMuteMode

Function name

__STATIC_INLINE void LL_USART_DisableMuteMode (USART_TypeDef * USARTx)

Function description

Prevent Mute Mode use.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 MME LL_USART_DisableMuteMode

LL_USART_IsEnabledMuteMode

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledMuteMode (const USART_TypeDef * USARTx)

Function description

Indicate if switch between Mute Mode and Active mode is allowed.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 MME LL_USART_IsEnabledMuteMode

LL_USART_SetOverSampling

Function name

__STATIC_INLINE void LL_USART_SetOverSampling (USART_TypeDef * USARTx, uint32_t OverSampling)

Function description

Set Oversampling to 8-bit or 16-bit mode.

Parameters

- **USARTx:** USART Instance
- **OverSampling:** This parameter can be one of the following values:
  - LL_USART_OVERSAMPLING_16
  - LL_USART_OVERSAMPLING_8

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 OVER8 LL_USART_SetOverSampling

LL_USART_GetOverSampling

Function name

__STATIC_INLINE uint32_t LL_USART_GetOverSampling (const USART_TypeDef * USARTx)

Function description

Return Oversampling mode.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_OVERSAMPLING_16
  - LL_USART_OVERSAMPLING_8

Reference Manual to LL API cross reference:

- CR1 OVER8 LL_USART_GetOverSampling

LL_USART_SetLastClkPulseOutput

Function name

__STATIC_INLINE void LL_USART_SetLastClkPulseOutput (USART_TypeDef * USARTx, uint32_t LastBitClockPulse)

Function description

Configure if Clock pulse of the last data bit is output to the SCLK pin or not.

Parameters

- **USARTx:** USART Instance
- **LastBitClockPulse:** This parameter can be one of the following values:
  - LL_USART_LASTCLKPULSE_NO_OUTPUT
  - LL_USART_LASTCLKPULSE_OUTPUT

Return values

- **None:**

Notes

- Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 LBCL LL_USART_SetLastClkPulseOutput

LL_USART_GetLastClkPulseOutput

Function name

__STATIC_INLINE uint32_t LL_USART_GetLastClkPulseOutput (const USART_TypeDef * USARTx)

Function description

Retrieve Clock pulse of the last data bit output configuration (Last bit Clock pulse output to the SCLK pin or not)

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_LASTCLKPULSE_NO_OUTPUT
  - LL_USART_LASTCLKPULSE_OUTPUT

Notes

- Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 LBCL LL_USART_GetLastClkPulseOutput

LL_USART_SetClockPhase

Function name

__STATIC_INLINE void LL_USART_SetClockPhase (USART_TypeDef * USARTx, uint32_t ClockPhase)

Function description

Select the phase of the clock output on the SCLK pin in synchronous mode.

Parameters

- **USARTx:** USART Instance
- **ClockPhase:** This parameter can be one of the following values:
  - LL_USART_PHASE_1EDGE
  - LL_USART_PHASE_2EDGE

Return values

- **None:**

Notes

- Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 CPHA LL_USART_SetClockPhase

LL_USART_GetClockPhase

Function name

__STATIC_INLINE uint32_t LL_USART_GetClockPhase (const USART_TypeDef * USARTx)

Function description

Return phase of the clock output on the SCLK pin in synchronous mode.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_PHASE_1EDGE
  - LL_USART_PHASE_2EDGE

Notes

- Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 CPHA LL_USART_GetClockPhase

LL_USART_SetClockPolarity

Function name

__STATIC_INLINE void LL_USART_SetClockPolarity (USART_TypeDef * USARTx, uint32_t ClockPolarity)

Function description

Select the polarity of the clock output on the SCLK pin in synchronous mode.

Parameters

- **USARTx:** USART Instance
- **ClockPolarity:** This parameter can be one of the following values:
  - LL_USART_POLARITY_LOW
  - LL_USART_POLARITY_HIGH

Return values

- **None:**

Notes

- Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 CPOL LL_USART_SetClockPolarity

LL_USART_GetClockPolarity

Function name

__STATIC_INLINE uint32_t LL_USART_GetClockPolarity (const USART_TypeDef * USARTx)

Function description

Return polarity of the clock output on the SCLK pin in synchronous mode.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_POLARITY_LOW
  - LL_USART_POLARITY_HIGH

Notes

- Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 CPOL LL_USART_GetClockPolarity

LL_USART_ConfigClock

Function name

__STATIC_INLINE void LL_USART_ConfigClock (USART_TypeDef * USARTx, uint32_t Phase, uint32_t Polarity, uint32_t LBCPOutput)

Function description

Configure Clock signal format (Phase Polarity and choice about output of last bit clock pulse)

Parameters

- **USARTx:** USART Instance
- **Phase:** This parameter can be one of the following values:
  - LL_USART_PHASE_1EDGE
  - LL_USART_PHASE_2EDGE
- **Polarity:** This parameter can be one of the following values:
  - LL_USART_POLARITY_LOW
  - LL_USART_POLARITY_HIGH
- **LBCPOutput:** This parameter can be one of the following values:
  - LL_USART_LASTCLKPULSE_NO_OUTPUT
  - LL_USART_LASTCLKPULSE_OUTPUT

Return values

- **None:**

Notes

- Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.
- Call of this function is equivalent to following function call sequence : Clock Phase configuration using LL_USART_SetClockPhase() functionClock Polarity configuration using LL_USART_SetClockPolarity() functionOutput of Last bit Clock pulse configuration using LL_USART_SetLastClkPulseOutput() function

Reference Manual to LL API cross reference:

- CR2 CPHA LL_USART_ConfigClock
- CR2 CPOL LL_USART_ConfigClock
- CR2 LBCL LL_USART_ConfigClock

LL_USART_SetPrescaler

Function name

__STATIC_INLINE void LL_USART_SetPrescaler (USART_TypeDef * USARTx, uint32_t PrescalerValue)

Function description

Configure Clock source prescaler for baudrate generator and oversampling.

Parameters

- **USARTx:** USART Instance
- **PrescalerValue:** This parameter can be one of the following values:
  - LL_USART_PRESCALER_DIV1
  - LL_USART_PRESCALER_DIV2
  - LL_USART_PRESCALER_DIV4
  - LL_USART_PRESCALER_DIV6
  - LL_USART_PRESCALER_DIV8
  - LL_USART_PRESCALER_DIV10
  - LL_USART_PRESCALER_DIV12
  - LL_USART_PRESCALER_DIV16
  - LL_USART_PRESCALER_DIV32
  - LL_USART_PRESCALER_DIV64
  - LL_USART_PRESCALER_DIV128
  - LL_USART_PRESCALER_DIV256

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- PRESC PRESCALER LL_USART_SetPrescaler

LL_USART_GetPrescaler

Function name

__STATIC_INLINE uint32_t LL_USART_GetPrescaler (const USART_TypeDef * USARTx)

Function description

Retrieve the Clock source prescaler for baudrate generator and oversampling.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_PRESCALER_DIV1
  - LL_USART_PRESCALER_DIV2
  - LL_USART_PRESCALER_DIV4
  - LL_USART_PRESCALER_DIV6
  - LL_USART_PRESCALER_DIV8
  - LL_USART_PRESCALER_DIV10
  - LL_USART_PRESCALER_DIV12
  - LL_USART_PRESCALER_DIV16
  - LL_USART_PRESCALER_DIV32
  - LL_USART_PRESCALER_DIV64
  - LL_USART_PRESCALER_DIV128
  - LL_USART_PRESCALER_DIV256

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- PRESC PRESCALER LL_USART_GetPrescaler

LL_USART_EnableSCLKOutput

Function name

__STATIC_INLINE void LL_USART_EnableSCLKOutput (USART_TypeDef * USARTx)

Function description

Enable Clock output on SCLK pin.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 CLKEN LL_USART_EnableSCLKOutput

LL_USART_DisableSCLKOutput

Function name

__STATIC_INLINE void LL_USART_DisableSCLKOutput (USART_TypeDef * USARTx)

Function description

Disable Clock output on SCLK pin.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 CLKEN LL_USART_DisableSCLKOutput

LL_USART_IsEnabledSCLKOutput

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledSCLKOutput (const USART_TypeDef * USARTx)

Function description

Indicate if Clock output on SCLK pin is enabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 CLKEN LL_USART_IsEnabledSCLKOutput

LL_USART_SetStopBitsLength

Function name

__STATIC_INLINE void LL_USART_SetStopBitsLength (USART_TypeDef * USARTx, uint32_t StopBits)

Function description

Set the length of the stop bits.

Parameters

- **USARTx:** USART Instance
- **StopBits:** This parameter can be one of the following values:
  - LL_USART_STOPBITS_0_5
  - LL_USART_STOPBITS_1
  - LL_USART_STOPBITS_1_5
  - LL_USART_STOPBITS_2

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 STOP LL_USART_SetStopBitsLength

LL_USART_GetStopBitsLength

Function name

__STATIC_INLINE uint32_t LL_USART_GetStopBitsLength (const USART_TypeDef * USARTx)

Function description

Retrieve the length of the stop bits.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_STOPBITS_0_5
  - LL_USART_STOPBITS_1
  - LL_USART_STOPBITS_1_5
  - LL_USART_STOPBITS_2

Reference Manual to LL API cross reference:

- CR2 STOP LL_USART_GetStopBitsLength

LL_USART_ConfigCharacter

Function name

__STATIC_INLINE void LL_USART_ConfigCharacter (USART_TypeDef * USARTx, uint32_t DataWidth, uint32_t Parity, uint32_t StopBits)

Function description

Configure Character frame format (Datawidth, Parity control, Stop Bits)

Parameters

- **USARTx:** USART Instance
- **DataWidth:** This parameter can be one of the following values:
  - LL_USART_DATAWIDTH_7B
  - LL_USART_DATAWIDTH_8B
  - LL_USART_DATAWIDTH_9B
- **Parity:** This parameter can be one of the following values:
  - LL_USART_PARITY_NONE
  - LL_USART_PARITY_EVEN
  - LL_USART_PARITY_ODD
- **StopBits:** This parameter can be one of the following values:
  - LL_USART_STOPBITS_0_5
  - LL_USART_STOPBITS_1
  - LL_USART_STOPBITS_1_5
  - LL_USART_STOPBITS_2

Return values

- **None:**

Notes

- Call of this function is equivalent to following function call sequence : Data Width configuration using LL_USART_SetDataWidth() functionParity Control and mode configuration using LL_USART_SetParity() functionStop bits configuration using LL_USART_SetStopBitsLength() function

Reference Manual to LL API cross reference:

- CR1 PS LL_USART_ConfigCharacter
- CR1 PCE LL_USART_ConfigCharacter
- CR1 M0 LL_USART_ConfigCharacter
- CR1 M1 LL_USART_ConfigCharacter
- CR2 STOP LL_USART_ConfigCharacter

LL_USART_SetTXRXSwap

Function name

__STATIC_INLINE void LL_USART_SetTXRXSwap (USART_TypeDef * USARTx, uint32_t SwapConfig)

Function description

Configure TX/RX pins swapping setting.

Parameters

- **USARTx:** USART Instance
- **SwapConfig:** This parameter can be one of the following values:
  - LL_USART_TXRX_STANDARD
  - LL_USART_TXRX_SWAPPED

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 SWAP LL_USART_SetTXRXSwap

LL_USART_GetTXRXSwap

Function name

__STATIC_INLINE uint32_t LL_USART_GetTXRXSwap (const USART_TypeDef * USARTx)

Function description

Retrieve TX/RX pins swapping configuration.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_TXRX_STANDARD
  - LL_USART_TXRX_SWAPPED

Reference Manual to LL API cross reference:

- CR2 SWAP LL_USART_GetTXRXSwap

LL_USART_SetRXPinLevel

Function name

__STATIC_INLINE void LL_USART_SetRXPinLevel (USART_TypeDef * USARTx, uint32_t PinInvMethod)

Function description

Configure RX pin active level logic.

Parameters

- **USARTx:** USART Instance
- **PinInvMethod:** This parameter can be one of the following values:
  - LL_USART_RXPIN_LEVEL_STANDARD
  - LL_USART_RXPIN_LEVEL_INVERTED

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 RXINV LL_USART_SetRXPinLevel

LL_USART_GetRXPinLevel

Function name

__STATIC_INLINE uint32_t LL_USART_GetRXPinLevel (const USART_TypeDef * USARTx)

Function description

Retrieve RX pin active level logic configuration.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_RXPIN_LEVEL_STANDARD
  - LL_USART_RXPIN_LEVEL_INVERTED

Reference Manual to LL API cross reference:

- CR2 RXINV LL_USART_GetRXPinLevel

LL_USART_SetTXPinLevel

Function name

__STATIC_INLINE void LL_USART_SetTXPinLevel (USART_TypeDef * USARTx, uint32_t PinInvMethod)

Function description

Configure TX pin active level logic.

Parameters

- **USARTx:** USART Instance
- **PinInvMethod:** This parameter can be one of the following values:
  - LL_USART_TXPIN_LEVEL_STANDARD
  - LL_USART_TXPIN_LEVEL_INVERTED

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 TXINV LL_USART_SetTXPinLevel

LL_USART_GetTXPinLevel

Function name

__STATIC_INLINE uint32_t LL_USART_GetTXPinLevel (const USART_TypeDef * USARTx)

Function description

Retrieve TX pin active level logic configuration.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_TXPIN_LEVEL_STANDARD
  - LL_USART_TXPIN_LEVEL_INVERTED

Reference Manual to LL API cross reference:

- CR2 TXINV LL_USART_GetTXPinLevel

LL_USART_SetBinaryDataLogic

Function name

__STATIC_INLINE void LL_USART_SetBinaryDataLogic (USART_TypeDef * USARTx, uint32_t DataLogic)

Function description

Configure Binary data logic.

Parameters

- **USARTx:** USART Instance
- **DataLogic:** This parameter can be one of the following values:
  - LL_USART_BINARY_LOGIC_POSITIVE
  - LL_USART_BINARY_LOGIC_NEGATIVE

Return values

- **None:**

Notes

- Allow to define how Logical data from the data register are send/received : either in positive/direct logic (1=H, 0=L) or in negative/inverse logic (1=L, 0=H)

Reference Manual to LL API cross reference:

- CR2 DATAINV LL_USART_SetBinaryDataLogic

LL_USART_GetBinaryDataLogic

Function name

__STATIC_INLINE uint32_t LL_USART_GetBinaryDataLogic (const USART_TypeDef * USARTx)

Function description

Retrieve Binary data configuration.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_BINARY_LOGIC_POSITIVE
  - LL_USART_BINARY_LOGIC_NEGATIVE

Reference Manual to LL API cross reference:

- CR2 DATAINV LL_USART_GetBinaryDataLogic

LL_USART_SetTransferBitOrder

Function name

__STATIC_INLINE void LL_USART_SetTransferBitOrder (USART_TypeDef * USARTx, uint32_t BitOrder)

Function description

Configure transfer bit order (either Less or Most Significant Bit First)

Parameters

- **USARTx:** USART Instance
- **BitOrder:** This parameter can be one of the following values:
  - LL_USART_BITORDER_LSBFIRST
  - LL_USART_BITORDER_MSBFIRST

Return values

- **None:**

Notes

- MSB First means data is transmitted/received with the MSB first, following the start bit. LSB First means data is transmitted/received with data bit 0 first, following the start bit.

Reference Manual to LL API cross reference:

- CR2 MSBFIRST LL_USART_SetTransferBitOrder

LL_USART_GetTransferBitOrder

Function name

__STATIC_INLINE uint32_t LL_USART_GetTransferBitOrder (const USART_TypeDef * USARTx)

Function description

Return transfer bit order (either Less or Most Significant Bit First)

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_BITORDER_LSBFIRST
  - LL_USART_BITORDER_MSBFIRST

Notes

- MSB First means data is transmitted/received with the MSB first, following the start bit. LSB First means data is transmitted/received with data bit 0 first, following the start bit.

Reference Manual to LL API cross reference:

- CR2 MSBFIRST LL_USART_GetTransferBitOrder

LL_USART_EnableAutoBaudRate

Function name

__STATIC_INLINE void LL_USART_EnableAutoBaudRate (USART_TypeDef * USARTx)

Function description

Enable Auto Baud-Rate Detection.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_USART_AUTOBAUDRATE_DETECTION_INSTANCE(USARTx) can be used to check whether or not Auto Baud Rate detection feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 ABREN LL_USART_EnableAutoBaudRate

LL_USART_DisableAutoBaudRate

Function name

__STATIC_INLINE void LL_USART_DisableAutoBaudRate (USART_TypeDef * USARTx)

Function description

Disable Auto Baud-Rate Detection.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_USART_AUTOBAUDRATE_DETECTION_INSTANCE(USARTx) can be used to check whether or not Auto Baud Rate detection feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 ABREN LL_USART_DisableAutoBaudRate

LL_USART_IsEnabledAutoBaud

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledAutoBaud (const USART_TypeDef * USARTx)

Function description

Indicate if Auto Baud-Rate Detection mechanism is enabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_USART_AUTOBAUDRATE_DETECTION_INSTANCE(USARTx) can be used to check whether or not Auto Baud Rate detection feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 ABREN LL_USART_IsEnabledAutoBaud

LL_USART_SetAutoBaudRateMode

Function name

__STATIC_INLINE void LL_USART_SetAutoBaudRateMode (USART_TypeDef * USARTx, uint32_t AutoBaudRateMode)

Function description

Set Auto Baud-Rate mode bits.

Parameters

- **USARTx:** USART Instance
- **AutoBaudRateMode:** This parameter can be one of the following values:
  - LL_USART_AUTOBAUD_DETECT_ON_STARTBIT
  - LL_USART_AUTOBAUD_DETECT_ON_FALLINGEDGE
  - LL_USART_AUTOBAUD_DETECT_ON_7F_FRAME
  - LL_USART_AUTOBAUD_DETECT_ON_55_FRAME

Return values

- **None:**

Notes

- Macro IS_USART_AUTOBAUDRATE_DETECTION_INSTANCE(USARTx) can be used to check whether or not Auto Baud Rate detection feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 ABRMODE LL_USART_SetAutoBaudRateMode

LL_USART_GetAutoBaudRateMode

Function name

__STATIC_INLINE uint32_t LL_USART_GetAutoBaudRateMode (const USART_TypeDef * USARTx)

Function description

Return Auto Baud-Rate mode.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_AUTOBAUD_DETECT_ON_STARTBIT
  - LL_USART_AUTOBAUD_DETECT_ON_FALLINGEDGE
  - LL_USART_AUTOBAUD_DETECT_ON_7F_FRAME
  - LL_USART_AUTOBAUD_DETECT_ON_55_FRAME

Notes

- Macro IS_USART_AUTOBAUDRATE_DETECTION_INSTANCE(USARTx) can be used to check whether or not Auto Baud Rate detection feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 ABRMODE LL_USART_GetAutoBaudRateMode

LL_USART_EnableRxTimeout

Function name

__STATIC_INLINE void LL_USART_EnableRxTimeout (USART_TypeDef * USARTx)

Function description

Enable Receiver Timeout.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 RTOEN LL_USART_EnableRxTimeout

LL_USART_DisableRxTimeout

Function name

__STATIC_INLINE void LL_USART_DisableRxTimeout (USART_TypeDef * USARTx)

Function description

Disable Receiver Timeout.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 RTOEN LL_USART_DisableRxTimeout

LL_USART_IsEnabledRxTimeout

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledRxTimeout (const USART_TypeDef * USARTx)

Function description

Indicate if Receiver Timeout feature is enabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR2 RTOEN LL_USART_IsEnabledRxTimeout

LL_USART_ConfigNodeAddress

Function name

__STATIC_INLINE void LL_USART_ConfigNodeAddress (USART_TypeDef * USARTx, uint32_t AddressLen, uint32_t NodeAddress)

Function description

Set Address of the USART node.

Parameters

- **USARTx:** USART Instance
- **AddressLen:** This parameter can be one of the following values:
  - LL_USART_ADDRESS_DETECT_4B
  - LL_USART_ADDRESS_DETECT_7B
- **NodeAddress:** 4 or 7 bit Address of the USART node.

Return values

- **None:**

Notes

- This is used in multiprocessor communication during Mute mode or Stop mode, for wake up with address mark detection.
- 4bits address node is used when 4-bit Address Detection is selected in ADDM7. (b7-b4 should be set to 0) 8bits address node is used when 7-bit Address Detection is selected in ADDM7. (This is used in multiprocessor communication during Mute mode or Stop mode, for wake up with 7-bit address mark detection. The MSB of the character sent by the transmitter should be equal to 1. It may also be used for character detection during normal reception, Mute mode inactive (for example, end of block detection in ModBus protocol). In this case, the whole received character (8-bit) is compared to the ADD[7:0] value and CMF flag is set on match)

Reference Manual to LL API cross reference:

- CR2 ADD LL_USART_ConfigNodeAddress
- CR2 ADDM7 LL_USART_ConfigNodeAddress

LL_USART_GetNodeAddress

Function name

__STATIC_INLINE uint32_t LL_USART_GetNodeAddress (const USART_TypeDef * USARTx)

Function description

Return 8 bit Address of the USART node as set in ADD field of CR2.

Parameters

- **USARTx:** USART Instance

Return values

- **Address:** of the USART node (Value between Min_Data=0 and Max_Data=255)

Notes

- If 4-bit Address Detection is selected in ADDM7, only 4bits (b3-b0) of returned value are relevant (b31-b4 are not relevant) If 7-bit Address Detection is selected in ADDM7, only 8bits (b7-b0) of returned value are relevant (b31-b8 are not relevant)

Reference Manual to LL API cross reference:

- CR2 ADD LL_USART_GetNodeAddress

LL_USART_GetNodeAddressLen

Function name

__STATIC_INLINE uint32_t LL_USART_GetNodeAddressLen (const USART_TypeDef * USARTx)

Function description

Return Length of Node Address used in Address Detection mode (7-bit or 4-bit)

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_ADDRESS_DETECT_4B
  - LL_USART_ADDRESS_DETECT_7B

Reference Manual to LL API cross reference:

- CR2 ADDM7 LL_USART_GetNodeAddressLen

LL_USART_EnableRTSHWFlowCtrl

Function name

__STATIC_INLINE void LL_USART_EnableRTSHWFlowCtrl (USART_TypeDef * USARTx)

Function description

Enable RTS HW Flow Control.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 RTSE LL_USART_EnableRTSHWFlowCtrl

LL_USART_DisableRTSHWFlowCtrl

Function name

__STATIC_INLINE void LL_USART_DisableRTSHWFlowCtrl (USART_TypeDef * USARTx)

Function description

Disable RTS HW Flow Control.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 RTSE LL_USART_DisableRTSHWFlowCtrl

LL_USART_EnableCTSHWFlowCtrl

Function name

__STATIC_INLINE void LL_USART_EnableCTSHWFlowCtrl (USART_TypeDef * USARTx)

Function description

Enable CTS HW Flow Control.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 CTSE LL_USART_EnableCTSHWFlowCtrl

LL_USART_DisableCTSHWFlowCtrl

Function name

__STATIC_INLINE void LL_USART_DisableCTSHWFlowCtrl (USART_TypeDef * USARTx)

Function description

Disable CTS HW Flow Control.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 CTSE LL_USART_DisableCTSHWFlowCtrl

LL_USART_SetHWFlowCtrl

Function name

__STATIC_INLINE void LL_USART_SetHWFlowCtrl (USART_TypeDef * USARTx, uint32_t HardwareFlowControl)

Function description

Configure HW Flow Control mode (both CTS and RTS)

Parameters

- **USARTx:** USART Instance
- **HardwareFlowControl:** This parameter can be one of the following values:
  - LL_USART_HWCONTROL_NONE
  - LL_USART_HWCONTROL_RTS
  - LL_USART_HWCONTROL_CTS
  - LL_USART_HWCONTROL_RTS_CTS

Return values

- **None:**

Notes

- Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 RTSE LL_USART_SetHWFlowCtrl
- CR3 CTSE LL_USART_SetHWFlowCtrl

LL_USART_GetHWFlowCtrl

Function name

__STATIC_INLINE uint32_t LL_USART_GetHWFlowCtrl (const USART_TypeDef * USARTx)

Function description

Return HW Flow Control configuration (both CTS and RTS)

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_HWCONTROL_NONE
  - LL_USART_HWCONTROL_RTS
  - LL_USART_HWCONTROL_CTS
  - LL_USART_HWCONTROL_RTS_CTS

Notes

- Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 RTSE LL_USART_GetHWFlowCtrl
- CR3 CTSE LL_USART_GetHWFlowCtrl

LL_USART_EnableOneBitSamp

Function name

__STATIC_INLINE void LL_USART_EnableOneBitSamp (USART_TypeDef * USARTx)

Function description

Enable One bit sampling method.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 ONEBIT LL_USART_EnableOneBitSamp

LL_USART_DisableOneBitSamp

Function name

__STATIC_INLINE void LL_USART_DisableOneBitSamp (USART_TypeDef * USARTx)

Function description

Disable One bit sampling method.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 ONEBIT LL_USART_DisableOneBitSamp

LL_USART_IsEnabledOneBitSamp

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledOneBitSamp (const USART_TypeDef * USARTx)

Function description

Indicate if One bit sampling method is enabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 ONEBIT LL_USART_IsEnabledOneBitSamp

LL_USART_EnableOverrunDetect

Function name

__STATIC_INLINE void LL_USART_EnableOverrunDetect (USART_TypeDef * USARTx)

Function description

Enable Overrun detection.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 OVRDIS LL_USART_EnableOverrunDetect

LL_USART_DisableOverrunDetect

Function name

__STATIC_INLINE void LL_USART_DisableOverrunDetect (USART_TypeDef * USARTx)

Function description

Disable Overrun detection.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 OVRDIS LL_USART_DisableOverrunDetect

LL_USART_IsEnabledOverrunDetect

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledOverrunDetect (const USART_TypeDef * USARTx)

Function description

Indicate if Overrun detection is enabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 OVRDIS LL_USART_IsEnabledOverrunDetect

LL_USART_SetWKUPType

Function name

__STATIC_INLINE void LL_USART_SetWKUPType (USART_TypeDef * USARTx, uint32_t Type)

Function description

Select event type for Wake UP Interrupt Flag (WUS[1:0] bits)

Parameters

- **USARTx:** USART Instance
- **Type:** This parameter can be one of the following values:
  - LL_USART_WAKEUP_ON_ADDRESS
  - LL_USART_WAKEUP_ON_STARTBIT
  - LL_USART_WAKEUP_ON_RXNE

Return values

- **None:**

Notes

- Macro IS_UART_WAKEUP_FROMSTOP_INSTANCE(USARTx) can be used to check whether or not Wake-up from Stop mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 WUS LL_USART_SetWKUPType

LL_USART_GetWKUPType

Function name

__STATIC_INLINE uint32_t LL_USART_GetWKUPType (const USART_TypeDef * USARTx)

Function description

Return event type for Wake UP Interrupt Flag (WUS[1:0] bits)

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_WAKEUP_ON_ADDRESS
  - LL_USART_WAKEUP_ON_STARTBIT
  - LL_USART_WAKEUP_ON_RXNE

Notes

- Macro IS_UART_WAKEUP_FROMSTOP_INSTANCE(USARTx) can be used to check whether or not Wake-up from Stop mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 WUS LL_USART_GetWKUPType

LL_USART_SetBaudRate

Function name

__STATIC_INLINE void LL_USART_SetBaudRate (USART_TypeDef * USARTx, uint32_t PeriphClk, uint32_t PrescalerValue, uint32_t OverSampling, uint32_t BaudRate)

Function description

Configure USART BRR register for achieving expected Baud Rate value.

Parameters

- **USARTx:** USART Instance
- **PeriphClk:** Peripheral Clock
- **PrescalerValue:** This parameter can be one of the following values:
  - LL_USART_PRESCALER_DIV1
  - LL_USART_PRESCALER_DIV2
  - LL_USART_PRESCALER_DIV4
  - LL_USART_PRESCALER_DIV6
  - LL_USART_PRESCALER_DIV8
  - LL_USART_PRESCALER_DIV10
  - LL_USART_PRESCALER_DIV12
  - LL_USART_PRESCALER_DIV16
  - LL_USART_PRESCALER_DIV32
  - LL_USART_PRESCALER_DIV64
  - LL_USART_PRESCALER_DIV128
  - LL_USART_PRESCALER_DIV256
- **OverSampling:** This parameter can be one of the following values:
  - LL_USART_OVERSAMPLING_16
  - LL_USART_OVERSAMPLING_8
- **BaudRate:** Baud Rate

Return values

- **None:**

Notes

- Compute and set USARTDIV value in BRR Register (full BRR content) according to used Peripheral Clock, Oversampling mode, and expected Baud Rate values
- Peripheral clock and Baud rate values provided as function parameters should be valid (Baud rate value != 0)
- In case of oversampling by 16 and 8, BRR content must be greater than or equal to 16d.

Reference Manual to LL API cross reference:

- BRR BRR LL_USART_SetBaudRate

LL_USART_GetBaudRate

Function name

__STATIC_INLINE uint32_t LL_USART_GetBaudRate (const USART_TypeDef * USARTx, uint32_t PeriphClk, uint32_t PrescalerValue, uint32_t OverSampling)

Function description

Return current Baud Rate value, according to USARTDIV present in BRR register (full BRR content), and to used Peripheral Clock and Oversampling mode values.

Parameters

- **USARTx:** USART Instance
- **PeriphClk:** Peripheral Clock
- **PrescalerValue:** This parameter can be one of the following values:
  - LL_USART_PRESCALER_DIV1
  - LL_USART_PRESCALER_DIV2
  - LL_USART_PRESCALER_DIV4
  - LL_USART_PRESCALER_DIV6
  - LL_USART_PRESCALER_DIV8
  - LL_USART_PRESCALER_DIV10
  - LL_USART_PRESCALER_DIV12
  - LL_USART_PRESCALER_DIV16
  - LL_USART_PRESCALER_DIV32
  - LL_USART_PRESCALER_DIV64
  - LL_USART_PRESCALER_DIV128
  - LL_USART_PRESCALER_DIV256
- **OverSampling:** This parameter can be one of the following values:
  - LL_USART_OVERSAMPLING_16
  - LL_USART_OVERSAMPLING_8

Return values

- **Baud:** Rate

Notes

- In case of non-initialized or invalid value stored in BRR register, value 0 will be returned.
- In case of oversampling by 16 and 8, BRR content must be greater than or equal to 16d.

Reference Manual to LL API cross reference:

- BRR BRR LL_USART_GetBaudRate

LL_USART_SetRxTimeout

Function name

__STATIC_INLINE void LL_USART_SetRxTimeout (USART_TypeDef * USARTx, uint32_t Timeout)

Function description

Set Receiver Time Out Value (expressed in nb of bits duration)

Parameters

- **USARTx:** USART Instance
- **Timeout:** Value between Min_Data=0x00 and Max_Data=0x00FFFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTOR RTO LL_USART_SetRxTimeout

LL_USART_GetRxTimeout

Function name

__STATIC_INLINE uint32_t LL_USART_GetRxTimeout (const USART_TypeDef * USARTx)

Function description

Get Receiver Time Out Value (expressed in nb of bits duration)

Parameters

- **USARTx:** USART Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x00FFFFFF

Reference Manual to LL API cross reference:

- RTOR RTO LL_USART_GetRxTimeout

LL_USART_SetBlockLength

Function name

__STATIC_INLINE void LL_USART_SetBlockLength (USART_TypeDef * USARTx, uint32_t BlockLength)

Function description

Set Block Length value in reception.

Parameters

- **USARTx:** USART Instance
- **BlockLength:** Value between Min_Data=0x00 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTOR BLEN LL_USART_SetBlockLength

LL_USART_GetBlockLength

Function name

__STATIC_INLINE uint32_t LL_USART_GetBlockLength (const USART_TypeDef * USARTx)

Function description

Get Block Length value in reception.

Parameters

- **USARTx:** USART Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- RTOR BLEN LL_USART_GetBlockLength

LL_USART_EnableIrda

Function name

__STATIC_INLINE void LL_USART_EnableIrda (USART_TypeDef * USARTx)

Function description

Enable IrDA mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_IRDA_INSTANCE(USARTx) can be used to check whether or not IrDA feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 IREN LL_USART_EnableIrda

LL_USART_DisableIrda

Function name

__STATIC_INLINE void LL_USART_DisableIrda (USART_TypeDef * USARTx)

Function description

Disable IrDA mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_IRDA_INSTANCE(USARTx) can be used to check whether or not IrDA feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 IREN LL_USART_DisableIrda

LL_USART_IsEnabledIrda

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIrda (const USART_TypeDef * USARTx)

Function description

Indicate if IrDA mode is enabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_IRDA_INSTANCE(USARTx) can be used to check whether or not IrDA feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 IREN LL_USART_IsEnabledIrda

LL_USART_SetIrdaPowerMode

Function name

__STATIC_INLINE void LL_USART_SetIrdaPowerMode (USART_TypeDef * USARTx, uint32_t PowerMode)

Function description

Configure IrDA Power Mode (Normal or Low Power)

Parameters

- **USARTx:** USART Instance
- **PowerMode:** This parameter can be one of the following values:
  - LL_USART_IRDA_POWER_NORMAL
  - LL_USART_IRDA_POWER_LOW

Return values

- **None:**

Notes

- Macro IS_IRDA_INSTANCE(USARTx) can be used to check whether or not IrDA feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 IRLP LL_USART_SetIrdaPowerMode

LL_USART_GetIrdaPowerMode

Function name

__STATIC_INLINE uint32_t LL_USART_GetIrdaPowerMode (const USART_TypeDef * USARTx)

Function description

Retrieve IrDA Power Mode configuration (Normal or Low Power)

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_IRDA_POWER_NORMAL
  - LL_USART_PHASE_2EDGE

Notes

- Macro IS_IRDA_INSTANCE(USARTx) can be used to check whether or not IrDA feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 IRLP LL_USART_GetIrdaPowerMode

LL_USART_SetIrdaPrescaler

Function name

__STATIC_INLINE void LL_USART_SetIrdaPrescaler (USART_TypeDef * USARTx, uint32_t PrescalerValue)

Function description

Set Irda prescaler value, used for dividing the USART clock source to achieve the Irda Low Power frequency (8 bits value)

Parameters

- **USARTx:** USART Instance
- **PrescalerValue:** Value between Min_Data=0x00 and Max_Data=0xFF

Return values

- **None:**

Notes

- Macro IS_IRDA_INSTANCE(USARTx) can be used to check whether or not IrDA feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- GTPR PSC LL_USART_SetIrdaPrescaler

LL_USART_GetIrdaPrescaler

Function name

__STATIC_INLINE uint32_t LL_USART_GetIrdaPrescaler (const USART_TypeDef * USARTx)

Function description

Return Irda prescaler value, used for dividing the USART clock source to achieve the Irda Low Power frequency (8 bits value)

Parameters

- **USARTx:** USART Instance

Return values

- **Irda:** prescaler value (Value between Min_Data=0x00 and Max_Data=0xFF)

Notes

- Macro IS_IRDA_INSTANCE(USARTx) can be used to check whether or not IrDA feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- GTPR PSC LL_USART_GetIrdaPrescaler

LL_USART_EnableSmartcardNACK

Function name

__STATIC_INLINE void LL_USART_EnableSmartcardNACK (USART_TypeDef * USARTx)

Function description

Enable Smartcard NACK transmission.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 NACK LL_USART_EnableSmartcardNACK

LL_USART_DisableSmartcardNACK

Function name

__STATIC_INLINE void LL_USART_DisableSmartcardNACK (USART_TypeDef * USARTx)

Function description

Disable Smartcard NACK transmission.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 NACK LL_USART_DisableSmartcardNACK

LL_USART_IsEnabledSmartcardNACK

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledSmartcardNACK (const USART_TypeDef * USARTx)

Function description

Indicate if Smartcard NACK transmission is enabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 NACK LL_USART_IsEnabledSmartcardNACK

LL_USART_EnableSmartcard

Function name

__STATIC_INLINE void LL_USART_EnableSmartcard (USART_TypeDef * USARTx)

Function description

Enable Smartcard mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 SCEN LL_USART_EnableSmartcard

LL_USART_DisableSmartcard

Function name

__STATIC_INLINE void LL_USART_DisableSmartcard (USART_TypeDef * USARTx)

Function description

Disable Smartcard mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 SCEN LL_USART_DisableSmartcard

LL_USART_IsEnabledSmartcard

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledSmartcard (const USART_TypeDef * USARTx)

Function description

Indicate if Smartcard mode is enabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 SCEN LL_USART_IsEnabledSmartcard

LL_USART_SetSmartcardAutoRetryCount

Function name

__STATIC_INLINE void LL_USART_SetSmartcardAutoRetryCount (USART_TypeDef * USARTx, uint32_t AutoRetryCount)

Function description

Set Smartcard Auto-Retry Count value (SCARCNT[2:0] bits)

Parameters

- **USARTx:** USART Instance
- **AutoRetryCount:** Value between Min_Data=0 and Max_Data=7

Return values

- **None:**

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.
- This bit-field specifies the number of retries in transmit and receive, in Smartcard mode. In transmission mode, it specifies the number of automatic retransmission retries, before generating a transmission error (FE bit set). In reception mode, it specifies the number or erroneous reception trials, before generating a reception error (RXNE and PE bits set)

Reference Manual to LL API cross reference:

- CR3 SCARCNT LL_USART_SetSmartcardAutoRetryCount

LL_USART_GetSmartcardAutoRetryCount

Function name

__STATIC_INLINE uint32_t LL_USART_GetSmartcardAutoRetryCount (const USART_TypeDef * USARTx)

Function description

Return Smartcard Auto-Retry Count value (SCARCNT[2:0] bits)

Parameters

- **USARTx:** USART Instance

Return values

- **Smartcard:** Auto-Retry Count value (Value between Min_Data=0 and Max_Data=7)

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 SCARCNT LL_USART_GetSmartcardAutoRetryCount

LL_USART_SetSmartcardPrescaler

Function name

__STATIC_INLINE void LL_USART_SetSmartcardPrescaler (USART_TypeDef * USARTx, uint32_t PrescalerValue)

Function description

Set Smartcard prescaler value, used for dividing the USART clock source to provide the SMARTCARD Clock (5 bits value)

Parameters

- **USARTx:** USART Instance
- **PrescalerValue:** Value between Min_Data=0 and Max_Data=31

Return values

- **None:**

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- GTPR PSC LL_USART_SetSmartcardPrescaler

LL_USART_GetSmartcardPrescaler

Function name

__STATIC_INLINE uint32_t LL_USART_GetSmartcardPrescaler (const USART_TypeDef * USARTx)

Function description

Return Smartcard prescaler value, used for dividing the USART clock source to provide the SMARTCARD Clock (5 bits value)

Parameters

- **USARTx:** USART Instance

Return values

- **Smartcard:** prescaler value (Value between Min_Data=0 and Max_Data=31)

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- GTPR PSC LL_USART_GetSmartcardPrescaler

LL_USART_SetSmartcardGuardTime

Function name

__STATIC_INLINE void LL_USART_SetSmartcardGuardTime (USART_TypeDef * USARTx, uint32_t GuardTime)

Function description

Set Smartcard Guard time value, expressed in nb of baud clocks periods (GT[7:0] bits : Guard time value)

Parameters

- **USARTx:** USART Instance
- **GuardTime:** Value between Min_Data=0x00 and Max_Data=0xFF

Return values

- **None:**

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- GTPR GT LL_USART_SetSmartcardGuardTime

LL_USART_GetSmartcardGuardTime

Function name

__STATIC_INLINE uint32_t LL_USART_GetSmartcardGuardTime (const USART_TypeDef * USARTx)

Function description

Return Smartcard Guard time value, expressed in nb of baud clocks periods (GT[7:0] bits : Guard time value)

Parameters

- **USARTx:** USART Instance

Return values

- **Smartcard:** Guard time value (Value between Min_Data=0x00 and Max_Data=0xFF)

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- GTPR GT LL_USART_GetSmartcardGuardTime

LL_USART_EnableHalfDuplex

Function name

__STATIC_INLINE void LL_USART_EnableHalfDuplex (USART_TypeDef * USARTx)

Function description

Enable Single Wire Half-Duplex mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_HALFDUPLEX_INSTANCE(USARTx) can be used to check whether or not Half-Duplex mode is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 HDSEL LL_USART_EnableHalfDuplex

LL_USART_DisableHalfDuplex

Function name

__STATIC_INLINE void LL_USART_DisableHalfDuplex (USART_TypeDef * USARTx)

Function description

Disable Single Wire Half-Duplex mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_HALFDUPLEX_INSTANCE(USARTx) can be used to check whether or not Half-Duplex mode is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 HDSEL LL_USART_DisableHalfDuplex

LL_USART_IsEnabledHalfDuplex

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledHalfDuplex (const USART_TypeDef * USARTx)

Function description

Indicate if Single Wire Half-Duplex mode is enabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_HALFDUPLEX_INSTANCE(USARTx) can be used to check whether or not Half-Duplex mode is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 HDSEL LL_USART_IsEnabledHalfDuplex

LL_USART_EnableSPISlave

Function name

__STATIC_INLINE void LL_USART_EnableSPISlave (USART_TypeDef * USARTx)

Function description

Enable SPI Synchronous Slave mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_SPI_SLAVE_INSTANCE(USARTx) can be used to check whether or not SPI Slave mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 SLVEN LL_USART_EnableSPISlave

LL_USART_DisableSPISlave

Function name

__STATIC_INLINE void LL_USART_DisableSPISlave (USART_TypeDef * USARTx)

Function description

Disable SPI Synchronous Slave mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_SPI_SLAVE_INSTANCE(USARTx) can be used to check whether or not SPI Slave mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 SLVEN LL_USART_DisableSPISlave

LL_USART_IsEnabledSPISlave

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledSPISlave (const USART_TypeDef * USARTx)

Function description

Indicate if SPI Synchronous Slave mode is enabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_SPI_SLAVE_INSTANCE(USARTx) can be used to check whether or not SPI Slave mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 SLVEN LL_USART_IsEnabledSPISlave

LL_USART_EnableSPISlaveSelect

Function name

__STATIC_INLINE void LL_USART_EnableSPISlaveSelect (USART_TypeDef * USARTx)

Function description

Enable SPI Slave Selection using NSS input pin.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_SPI_SLAVE_INSTANCE(USARTx) can be used to check whether or not SPI Slave mode feature is supported by the USARTx instance.
- SPI Slave Selection depends on NSS input pin (The slave is selected when NSS is low and deselected when NSS is high).

Reference Manual to LL API cross reference:

- CR2 DIS_NSS LL_USART_EnableSPISlaveSelect

LL_USART_DisableSPISlaveSelect

Function name

__STATIC_INLINE void LL_USART_DisableSPISlaveSelect (USART_TypeDef * USARTx)

Function description

Disable SPI Slave Selection using NSS input pin.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_SPI_SLAVE_INSTANCE(USARTx) can be used to check whether or not SPI Slave mode feature is supported by the USARTx instance.
- SPI Slave will be always selected and NSS input pin will be ignored.

Reference Manual to LL API cross reference:

- CR2 DIS_NSS LL_USART_DisableSPISlaveSelect

LL_USART_IsEnabledSPISlaveSelect

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledSPISlaveSelect (const USART_TypeDef * USARTx)

Function description

Indicate if SPI Slave Selection depends on NSS input pin.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_SPI_SLAVE_INSTANCE(USARTx) can be used to check whether or not SPI Slave mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 DIS_NSS LL_USART_IsEnabledSPISlaveSelect

LL_USART_SetLINBrkDetectionLen

Function name

__STATIC_INLINE void LL_USART_SetLINBrkDetectionLen (USART_TypeDef * USARTx, uint32_t LINBDLength)

Function description

Set LIN Break Detection Length.

Parameters

- **USARTx:** USART Instance
- **LINBDLength:** This parameter can be one of the following values:
  - LL_USART_LINBREAK_DETECT_10B
  - LL_USART_LINBREAK_DETECT_11B

Return values

- **None:**

Notes

- Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 LBDL LL_USART_SetLINBrkDetectionLen

LL_USART_GetLINBrkDetectionLen

Function name

__STATIC_INLINE uint32_t LL_USART_GetLINBrkDetectionLen (const USART_TypeDef * USARTx)

Function description

Return LIN Break Detection Length.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_LINBREAK_DETECT_10B
  - LL_USART_LINBREAK_DETECT_11B

Notes

- Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 LBDL LL_USART_GetLINBrkDetectionLen

LL_USART_EnableLIN

Function name

__STATIC_INLINE void LL_USART_EnableLIN (USART_TypeDef * USARTx)

Function description

Enable LIN mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 LINEN LL_USART_EnableLIN

LL_USART_DisableLIN

Function name

__STATIC_INLINE void LL_USART_DisableLIN (USART_TypeDef * USARTx)

Function description

Disable LIN mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 LINEN LL_USART_DisableLIN

LL_USART_IsEnabledLIN

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledLIN (const USART_TypeDef * USARTx)

Function description

Indicate if LIN mode is enabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 LINEN LL_USART_IsEnabledLIN

LL_USART_SetDEDeassertionTime

Function name

__STATIC_INLINE void LL_USART_SetDEDeassertionTime (USART_TypeDef * USARTx, uint32_t Time)

Function description

Set DEDT (Driver Enable De-Assertion Time), Time value expressed on 5 bits ([4:0] bits).

Parameters

- **USARTx:** USART Instance
- **Time:** Value between Min_Data=0 and Max_Data=31

Return values

- **None:**

Notes

- Macro IS_UART_DRIVER_ENABLE_INSTANCE(USARTx) can be used to check whether or not Driver Enable feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 DEDT LL_USART_SetDEDeassertionTime

LL_USART_GetDEDeassertionTime

Function name

__STATIC_INLINE uint32_t LL_USART_GetDEDeassertionTime (const USART_TypeDef * USARTx)

Function description

Return DEDT (Driver Enable De-Assertion Time)

Parameters

- **USARTx:** USART Instance

Return values

- **Time:** value expressed on 5 bits ([4:0] bits) : Value between Min_Data=0 and Max_Data=31

Notes

- Macro IS_UART_DRIVER_ENABLE_INSTANCE(USARTx) can be used to check whether or not Driver Enable feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 DEDT LL_USART_GetDEDeassertionTime

LL_USART_SetDEAssertionTime

Function name

__STATIC_INLINE void LL_USART_SetDEAssertionTime (USART_TypeDef * USARTx, uint32_t Time)

Function description

Set DEAT (Driver Enable Assertion Time), Time value expressed on 5 bits ([4:0] bits).

Parameters

- **USARTx:** USART Instance
- **Time:** Value between Min_Data=0 and Max_Data=31

Return values

- **None:**

Notes

- Macro IS_UART_DRIVER_ENABLE_INSTANCE(USARTx) can be used to check whether or not Driver Enable feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 DEAT LL_USART_SetDEAssertionTime

LL_USART_GetDEAssertionTime

Function name

__STATIC_INLINE uint32_t LL_USART_GetDEAssertionTime (const USART_TypeDef * USARTx)

Function description

Return DEAT (Driver Enable Assertion Time)

Parameters

- **USARTx:** USART Instance

Return values

- **Time:** value expressed on 5 bits ([4:0] bits) : Value between Min_Data=0 and Max_Data=31

Notes

- Macro IS_UART_DRIVER_ENABLE_INSTANCE(USARTx) can be used to check whether or not Driver Enable feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 DEAT LL_USART_GetDEAssertionTime

LL_USART_EnableDEMode

Function name

__STATIC_INLINE void LL_USART_EnableDEMode (USART_TypeDef * USARTx)

Function description

Enable Driver Enable (DE) Mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_DRIVER_ENABLE_INSTANCE(USARTx) can be used to check whether or not Driver Enable feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 DEM LL_USART_EnableDEMode

LL_USART_DisableDEMode

Function name

__STATIC_INLINE void LL_USART_DisableDEMode (USART_TypeDef * USARTx)

Function description

Disable Driver Enable (DE) Mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_DRIVER_ENABLE_INSTANCE(USARTx) can be used to check whether or not Driver Enable feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 DEM LL_USART_DisableDEMode

LL_USART_IsEnabledDEMode

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledDEMode (const USART_TypeDef * USARTx)

Function description

Indicate if Driver Enable (DE) Mode is enabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_DRIVER_ENABLE_INSTANCE(USARTx) can be used to check whether or not Driver Enable feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 DEM LL_USART_IsEnabledDEMode

LL_USART_SetDESignalPolarity

Function name

__STATIC_INLINE void LL_USART_SetDESignalPolarity (USART_TypeDef * USARTx, uint32_t Polarity)

Function description

Select Driver Enable Polarity.

Parameters

- **USARTx:** USART Instance
- **Polarity:** This parameter can be one of the following values:
  - LL_USART_DE_POLARITY_HIGH
  - LL_USART_DE_POLARITY_LOW

Return values

- **None:**

Notes

- Macro IS_UART_DRIVER_ENABLE_INSTANCE(USARTx) can be used to check whether or not Driver Enable feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 DEP LL_USART_SetDESignalPolarity

LL_USART_GetDESignalPolarity

Function name

__STATIC_INLINE uint32_t LL_USART_GetDESignalPolarity (const USART_TypeDef * USARTx)

Function description

Return Driver Enable Polarity.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_DE_POLARITY_HIGH
  - LL_USART_DE_POLARITY_LOW

Notes

- Macro IS_UART_DRIVER_ENABLE_INSTANCE(USARTx) can be used to check whether or not Driver Enable feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 DEP LL_USART_GetDESignalPolarity

LL_USART_ConfigAsyncMode

Function name

__STATIC_INLINE void LL_USART_ConfigAsyncMode (USART_TypeDef * USARTx)

Function description

Perform basic configuration of USART for enabling use in Asynchronous Mode (UART)

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- In UART mode, the following bits must be kept cleared: LINEN bit in the USART_CR2 register,CLKEN bit in the USART_CR2 register,SCEN bit in the USART_CR3 register,IREN bit in the USART_CR3 register,HDSEL bit in the USART_CR3 register.
- Call of this function is equivalent to following function call sequence : Clear LINEN in CR2 using LL_USART_DisableLIN() functionClear CLKEN in CR2 using LL_USART_DisableSCLKOutput() functionClear SCEN in CR3 using LL_USART_DisableSmartcard() functionClear IREN in CR3 using LL_USART_DisableIrda() functionClear HDSEL in CR3 using LL_USART_DisableHalfDuplex() function
- Other remaining configurations items related to Asynchronous Mode (as Baud Rate, Word length, Parity, ...) should be set using dedicated functions

Reference Manual to LL API cross reference:

- CR2 LINEN LL_USART_ConfigAsyncMode
- CR2 CLKEN LL_USART_ConfigAsyncMode
- CR3 SCEN LL_USART_ConfigAsyncMode
- CR3 IREN LL_USART_ConfigAsyncMode
- CR3 HDSEL LL_USART_ConfigAsyncMode

LL_USART_ConfigSyncMode

Function name

__STATIC_INLINE void LL_USART_ConfigSyncMode (USART_TypeDef * USARTx)

Function description

Perform basic configuration of USART for enabling use in Synchronous Mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- In Synchronous mode, the following bits must be kept cleared: LINEN bit in the USART_CR2 register,SCEN bit in the USART_CR3 register,IREN bit in the USART_CR3 register,HDSEL bit in the USART_CR3 register. This function also sets the USART in Synchronous mode.
- Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.
- Call of this function is equivalent to following function call sequence : Clear LINEN in CR2 using LL_USART_DisableLIN() functionClear IREN in CR3 using LL_USART_DisableIrda() functionClear SCEN in CR3 using LL_USART_DisableSmartcard() functionClear HDSEL in CR3 using LL_USART_DisableHalfDuplex() functionSet CLKEN in CR2 using LL_USART_EnableSCLKOutput() function
- Other remaining configurations items related to Synchronous Mode (as Baud Rate, Word length, Parity, Clock Polarity, ...) should be set using dedicated functions

Reference Manual to LL API cross reference:

- CR2 LINEN LL_USART_ConfigSyncMode
- CR2 CLKEN LL_USART_ConfigSyncMode
- CR3 SCEN LL_USART_ConfigSyncMode
- CR3 IREN LL_USART_ConfigSyncMode
- CR3 HDSEL LL_USART_ConfigSyncMode

LL_USART_ConfigLINMode

Function name

__STATIC_INLINE void LL_USART_ConfigLINMode (USART_TypeDef * USARTx)

Function description

Perform basic configuration of USART for enabling use in LIN Mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- In LIN mode, the following bits must be kept cleared: STOP and CLKEN bits in the USART_CR2 register,SCEN bit in the USART_CR3 register,IREN bit in the USART_CR3 register,HDSEL bit in the USART_CR3 register. This function also set the UART/USART in LIN mode.
- Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.
- Call of this function is equivalent to following function call sequence : Clear CLKEN in CR2 using LL_USART_DisableSCLKOutput() functionClear STOP in CR2 using LL_USART_SetStopBitsLength() functionClear SCEN in CR3 using LL_USART_DisableSmartcard() functionClear IREN in CR3 using LL_USART_DisableIrda() functionClear HDSEL in CR3 using LL_USART_DisableHalfDuplex() functionSet LINEN in CR2 using LL_USART_EnableLIN() function
- Other remaining configurations items related to LIN Mode (as Baud Rate, Word length, LIN Break Detection Length, ...) should be set using dedicated functions

Reference Manual to LL API cross reference:

- CR2 CLKEN LL_USART_ConfigLINMode
- CR2 STOP LL_USART_ConfigLINMode
- CR2 LINEN LL_USART_ConfigLINMode
- CR3 IREN LL_USART_ConfigLINMode
- CR3 SCEN LL_USART_ConfigLINMode
- CR3 HDSEL LL_USART_ConfigLINMode

LL_USART_ConfigHalfDuplexMode

Function name

__STATIC_INLINE void LL_USART_ConfigHalfDuplexMode (USART_TypeDef * USARTx)

Function description

Perform basic configuration of USART for enabling use in Half Duplex Mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- In Half Duplex mode, the following bits must be kept cleared: LINEN bit in the USART_CR2 register,CLKEN bit in the USART_CR2 register,SCEN bit in the USART_CR3 register,IREN bit in the USART_CR3 register, This function also sets the UART/USART in Half Duplex mode.
- Macro IS_UART_HALFDUPLEX_INSTANCE(USARTx) can be used to check whether or not Half-Duplex mode is supported by the USARTx instance.
- Call of this function is equivalent to following function call sequence : Clear LINEN in CR2 using LL_USART_DisableLIN() functionClear CLKEN in CR2 using LL_USART_DisableSCLKOutput() functionClear SCEN in CR3 using LL_USART_DisableSmartcard() functionClear IREN in CR3 using LL_USART_DisableIrda() functionSet HDSEL in CR3 using LL_USART_EnableHalfDuplex() function
- Other remaining configurations items related to Half Duplex Mode (as Baud Rate, Word length, Parity, ...) should be set using dedicated functions

Reference Manual to LL API cross reference:

- CR2 LINEN LL_USART_ConfigHalfDuplexMode
- CR2 CLKEN LL_USART_ConfigHalfDuplexMode
- CR3 HDSEL LL_USART_ConfigHalfDuplexMode
- CR3 SCEN LL_USART_ConfigHalfDuplexMode
- CR3 IREN LL_USART_ConfigHalfDuplexMode

LL_USART_ConfigSmartcardMode

Function name

__STATIC_INLINE void LL_USART_ConfigSmartcardMode (USART_TypeDef * USARTx)

Function description

Perform basic configuration of USART for enabling use in Smartcard Mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- In Smartcard mode, the following bits must be kept cleared: LINEN bit in the USART_CR2 register,IREN bit in the USART_CR3 register,HDSEL bit in the USART_CR3 register. This function also configures Stop bits to 1.5 bits and sets the USART in Smartcard mode (SCEN bit). Clock Output is also enabled (CLKEN).
- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.
- Call of this function is equivalent to following function call sequence : Clear LINEN in CR2 using LL_USART_DisableLIN() functionClear IREN in CR3 using LL_USART_DisableIrda() functionClear HDSEL in CR3 using LL_USART_DisableHalfDuplex() functionConfigure STOP in CR2 using LL_USART_SetStopBitsLength() functionSet CLKEN in CR2 using LL_USART_EnableSCLKOutput() functionSet SCEN in CR3 using LL_USART_EnableSmartcard() function
- Other remaining configurations items related to Smartcard Mode (as Baud Rate, Word length, Parity, ...) should be set using dedicated functions

Reference Manual to LL API cross reference:

- CR2 LINEN LL_USART_ConfigSmartcardMode
- CR2 STOP LL_USART_ConfigSmartcardMode
- CR2 CLKEN LL_USART_ConfigSmartcardMode
- CR3 HDSEL LL_USART_ConfigSmartcardMode
- CR3 SCEN LL_USART_ConfigSmartcardMode

LL_USART_ConfigIrdaMode

Function name

__STATIC_INLINE void LL_USART_ConfigIrdaMode (USART_TypeDef * USARTx)

Function description

Perform basic configuration of USART for enabling use in Irda Mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- In IRDA mode, the following bits must be kept cleared: LINEN bit in the USART_CR2 register,STOP and CLKEN bits in the USART_CR2 register,SCEN bit in the USART_CR3 register,HDSEL bit in the USART_CR3 register. This function also sets the UART/USART in IRDA mode (IREN bit).
- Macro IS_IRDA_INSTANCE(USARTx) can be used to check whether or not IrDA feature is supported by the USARTx instance.
- Call of this function is equivalent to following function call sequence : Clear LINEN in CR2 using LL_USART_DisableLIN() functionClear CLKEN in CR2 using LL_USART_DisableSCLKOutput() functionClear SCEN in CR3 using LL_USART_DisableSmartcard() functionClear HDSEL in CR3 using LL_USART_DisableHalfDuplex() functionConfigure STOP in CR2 using LL_USART_SetStopBitsLength() functionSet IREN in CR3 using LL_USART_EnableIrda() function
- Other remaining configurations items related to Irda Mode (as Baud Rate, Word length, Power mode, ...) should be set using dedicated functions

Reference Manual to LL API cross reference:

- CR2 LINEN LL_USART_ConfigIrdaMode
- CR2 CLKEN LL_USART_ConfigIrdaMode
- CR2 STOP LL_USART_ConfigIrdaMode
- CR3 SCEN LL_USART_ConfigIrdaMode
- CR3 HDSEL LL_USART_ConfigIrdaMode
- CR3 IREN LL_USART_ConfigIrdaMode

LL_USART_ConfigMultiProcessMode

Function name

__STATIC_INLINE void LL_USART_ConfigMultiProcessMode (USART_TypeDef * USARTx)

Function description

Perform basic configuration of USART for enabling use in Multi processor Mode (several USARTs connected in a network, one of the USARTs can be the master, its TX output connected to the RX inputs of the other slaves USARTs).

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- In MultiProcessor mode, the following bits must be kept cleared: LINEN bit in the USART_CR2 register,CLKEN bit in the USART_CR2 register,SCEN bit in the USART_CR3 register,IREN bit in the USART_CR3 register,HDSEL bit in the USART_CR3 register.
- Call of this function is equivalent to following function call sequence : Clear LINEN in CR2 using LL_USART_DisableLIN() functionClear CLKEN in CR2 using LL_USART_DisableSCLKOutput() functionClear SCEN in CR3 using LL_USART_DisableSmartcard() functionClear IREN in CR3 using LL_USART_DisableIrda() functionClear HDSEL in CR3 using LL_USART_DisableHalfDuplex() function
- Other remaining configurations items related to Multi processor Mode (as Baud Rate, Wake Up Method, Node address, ...) should be set using dedicated functions

Reference Manual to LL API cross reference:

- CR2 LINEN LL_USART_ConfigMultiProcessMode
- CR2 CLKEN LL_USART_ConfigMultiProcessMode
- CR3 SCEN LL_USART_ConfigMultiProcessMode
- CR3 HDSEL LL_USART_ConfigMultiProcessMode
- CR3 IREN LL_USART_ConfigMultiProcessMode

LL_USART_IsActiveFlag_PE

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_PE (const USART_TypeDef * USARTx)

Function description

Check if the USART Parity Error Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR PE LL_USART_IsActiveFlag_PE

LL_USART_IsActiveFlag_FE

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_FE (const USART_TypeDef * USARTx)

Function description

Check if the USART Framing Error Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR FE LL_USART_IsActiveFlag_FE

LL_USART_IsActiveFlag_NE

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_NE (const USART_TypeDef * USARTx)

Function description

Check if the USART Noise error detected Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR NE LL_USART_IsActiveFlag_NE

LL_USART_IsActiveFlag_ORE

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_ORE (const USART_TypeDef * USARTx)

Function description

Check if the USART OverRun Error Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR ORE LL_USART_IsActiveFlag_ORE

LL_USART_IsActiveFlag_IDLE

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_IDLE (const USART_TypeDef * USARTx)

Function description

Check if the USART IDLE line detected Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR IDLE LL_USART_IsActiveFlag_IDLE

LL_USART_IsActiveFlag_RXNE_RXFNE

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_RXNE_RXFNE (const USART_TypeDef * USARTx)

Function description

Check if the USART Read Data Register or USART RX FIFO Not Empty Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ISR RXNE_RXFNE LL_USART_IsActiveFlag_RXNE_RXFNE

LL_USART_IsActiveFlag_TC

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_TC (const USART_TypeDef * USARTx)

Function description

Check if the USART Transmission Complete Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TC LL_USART_IsActiveFlag_TC

LL_USART_IsActiveFlag_TXE_TXFNF

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_TXE_TXFNF (const USART_TypeDef * USARTx)

Function description

Check if the USART Transmit Data Register Empty or USART TX FIFO Not Full Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ISR TXE_TXFNF LL_USART_IsActiveFlag_TXE_TXFNF

LL_USART_IsActiveFlag_LBD

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_LBD (const USART_TypeDef * USARTx)

Function description

Check if the USART LIN Break Detection Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ISR LBDF LL_USART_IsActiveFlag_LBD

LL_USART_IsActiveFlag_nCTS

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_nCTS (const USART_TypeDef * USARTx)

Function description

Check if the USART CTS interrupt Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ISR CTSIF LL_USART_IsActiveFlag_nCTS

LL_USART_IsActiveFlag_CTS

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_CTS (const USART_TypeDef * USARTx)

Function description

Check if the USART CTS Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ISR CTS LL_USART_IsActiveFlag_CTS

LL_USART_IsActiveFlag_RTO

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_RTO (const USART_TypeDef * USARTx)

Function description

Check if the USART Receiver Time Out Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR RTOF LL_USART_IsActiveFlag_RTO

LL_USART_IsActiveFlag_EOB

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_EOB (const USART_TypeDef * USARTx)

Function description

Check if the USART End Of Block Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ISR EOBF LL_USART_IsActiveFlag_EOB

LL_USART_IsActiveFlag_UDR

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_UDR (const USART_TypeDef * USARTx)

Function description

Check if the SPI Slave Underrun error flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_SPI_SLAVE_INSTANCE(USARTx) can be used to check whether or not SPI Slave mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ISR UDR LL_USART_IsActiveFlag_UDR

LL_USART_IsActiveFlag_ABRE

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_ABRE (const USART_TypeDef * USARTx)

Function description

Check if the USART Auto-Baud Rate Error Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_USART_AUTOBAUDRATE_DETECTION_INSTANCE(USARTx) can be used to check whether or not Auto Baud Rate detection feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ISR ABRE LL_USART_IsActiveFlag_ABRE

LL_USART_IsActiveFlag_ABR

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_ABR (const USART_TypeDef * USARTx)

Function description

Check if the USART Auto-Baud Rate Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_USART_AUTOBAUDRATE_DETECTION_INSTANCE(USARTx) can be used to check whether or not Auto Baud Rate detection feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ISR ABRF LL_USART_IsActiveFlag_ABR

LL_USART_IsActiveFlag_BUSY

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_BUSY (const USART_TypeDef * USARTx)

Function description

Check if the USART Busy Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR BUSY LL_USART_IsActiveFlag_BUSY

LL_USART_IsActiveFlag_CM

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_CM (const USART_TypeDef * USARTx)

Function description

Check if the USART Character Match Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR CMF LL_USART_IsActiveFlag_CM

LL_USART_IsActiveFlag_SBK

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_SBK (const USART_TypeDef * USARTx)

Function description

Check if the USART Send Break Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR SBKF LL_USART_IsActiveFlag_SBK

LL_USART_IsActiveFlag_RWU

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_RWU (const USART_TypeDef * USARTx)

Function description

Check if the USART Receive Wake Up from mute mode Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR RWU LL_USART_IsActiveFlag_RWU

LL_USART_IsActiveFlag_WKUP

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_WKUP (const USART_TypeDef * USARTx)

Function description

Check if the USART Wake Up from stop mode Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_WAKEUP_FROMSTOP_INSTANCE(USARTx) can be used to check whether or not Wake-up from Stop mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ISR WUF LL_USART_IsActiveFlag_WKUP

LL_USART_IsActiveFlag_TEACK

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_TEACK (const USART_TypeDef * USARTx)

Function description

Check if the USART Transmit Enable Acknowledge Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TEACK LL_USART_IsActiveFlag_TEACK

LL_USART_IsActiveFlag_REACK

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_REACK (const USART_TypeDef * USARTx)

Function description

Check if the USART Receive Enable Acknowledge Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR REACK LL_USART_IsActiveFlag_REACK

LL_USART_IsActiveFlag_TXFE

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_TXFE (const USART_TypeDef * USARTx)

Function description

Check if the USART TX FIFO Empty Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ISR TXFE LL_USART_IsActiveFlag_TXFE

LL_USART_IsActiveFlag_RXFF

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_RXFF (const USART_TypeDef * USARTx)

Function description

Check if the USART RX FIFO Full Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ISR RXFF LL_USART_IsActiveFlag_RXFF

LL_USART_IsActiveFlag_TCBGT

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_TCBGT (const USART_TypeDef * USARTx)

Function description

Check if the Smartcard Transmission Complete Before Guard Time Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TCBGT LL_USART_IsActiveFlag_TCBGT

LL_USART_IsActiveFlag_TXFT

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_TXFT (const USART_TypeDef * USARTx)

Function description

Check if the USART TX FIFO Threshold Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ISR TXFT LL_USART_IsActiveFlag_TXFT

LL_USART_IsActiveFlag_RXFT

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_RXFT (const USART_TypeDef * USARTx)

Function description

Check if the USART RX FIFO Threshold Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ISR RXFT LL_USART_IsActiveFlag_RXFT

LL_USART_ClearFlag_PE

Function name

__STATIC_INLINE void LL_USART_ClearFlag_PE (USART_TypeDef * USARTx)

Function description

Clear Parity Error Flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR PECF LL_USART_ClearFlag_PE

LL_USART_ClearFlag_FE

Function name

__STATIC_INLINE void LL_USART_ClearFlag_FE (USART_TypeDef * USARTx)

Function description

Clear Framing Error Flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR FECF LL_USART_ClearFlag_FE

LL_USART_ClearFlag_NE

Function name

__STATIC_INLINE void LL_USART_ClearFlag_NE (USART_TypeDef * USARTx)

Function description

Clear Noise Error detected Flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR NECF LL_USART_ClearFlag_NE

LL_USART_ClearFlag_ORE

Function name

__STATIC_INLINE void LL_USART_ClearFlag_ORE (USART_TypeDef * USARTx)

Function description

Clear OverRun Error Flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR ORECF LL_USART_ClearFlag_ORE

LL_USART_ClearFlag_IDLE

Function name

__STATIC_INLINE void LL_USART_ClearFlag_IDLE (USART_TypeDef * USARTx)

Function description

Clear IDLE line detected Flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR IDLECF LL_USART_ClearFlag_IDLE

LL_USART_ClearFlag_TXFE

Function name

__STATIC_INLINE void LL_USART_ClearFlag_TXFE (USART_TypeDef * USARTx)

Function description

Clear TX FIFO Empty Flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ICR TXFECF LL_USART_ClearFlag_TXFE

LL_USART_ClearFlag_TC

Function name

__STATIC_INLINE void LL_USART_ClearFlag_TC (USART_TypeDef * USARTx)

Function description

Clear Transmission Complete Flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR TCCF LL_USART_ClearFlag_TC

LL_USART_ClearFlag_TCBGT

Function name

__STATIC_INLINE void LL_USART_ClearFlag_TCBGT (USART_TypeDef * USARTx)

Function description

Clear Smartcard Transmission Complete Before Guard Time Flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR TCBGTCF LL_USART_ClearFlag_TCBGT

LL_USART_ClearFlag_LBD

Function name

__STATIC_INLINE void LL_USART_ClearFlag_LBD (USART_TypeDef * USARTx)

Function description

Clear LIN Break Detection Flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ICR LBDCF LL_USART_ClearFlag_LBD

LL_USART_ClearFlag_nCTS

Function name

__STATIC_INLINE void LL_USART_ClearFlag_nCTS (USART_TypeDef * USARTx)

Function description

Clear CTS Interrupt Flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ICR CTSCF LL_USART_ClearFlag_nCTS

LL_USART_ClearFlag_RTO

Function name

__STATIC_INLINE void LL_USART_ClearFlag_RTO (USART_TypeDef * USARTx)

Function description

Clear Receiver Time Out Flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR RTOCF LL_USART_ClearFlag_RTO

LL_USART_ClearFlag_EOB

Function name

__STATIC_INLINE void LL_USART_ClearFlag_EOB (USART_TypeDef * USARTx)

Function description

Clear End Of Block Flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ICR EOBCF LL_USART_ClearFlag_EOB

LL_USART_ClearFlag_UDR

Function name

__STATIC_INLINE void LL_USART_ClearFlag_UDR (USART_TypeDef * USARTx)

Function description

Clear SPI Slave Underrun Flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_SPI_SLAVE_INSTANCE(USARTx) can be used to check whether or not SPI Slave mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ICR UDRCF LL_USART_ClearFlag_UDR

LL_USART_ClearFlag_CM

Function name

__STATIC_INLINE void LL_USART_ClearFlag_CM (USART_TypeDef * USARTx)

Function description

Clear Character Match Flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR CMCF LL_USART_ClearFlag_CM

LL_USART_ClearFlag_WKUP

Function name

__STATIC_INLINE void LL_USART_ClearFlag_WKUP (USART_TypeDef * USARTx)

Function description

Clear Wake Up from stop mode Flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_WAKEUP_FROMSTOP_INSTANCE(USARTx) can be used to check whether or not Wake-up from Stop mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ICR WUCF LL_USART_ClearFlag_WKUP

LL_USART_EnableIT_IDLE

Function name

__STATIC_INLINE void LL_USART_EnableIT_IDLE (USART_TypeDef * USARTx)

Function description

Enable IDLE Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 IDLEIE LL_USART_EnableIT_IDLE

LL_USART_EnableIT_RXNE_RXFNE

Function name

__STATIC_INLINE void LL_USART_EnableIT_RXNE_RXFNE (USART_TypeDef * USARTx)

Function description

Enable RX Not Empty and RX FIFO Not Empty Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 RXNEIE_RXFNEIE LL_USART_EnableIT_RXNE_RXFNE

LL_USART_EnableIT_TC

Function name

__STATIC_INLINE void LL_USART_EnableIT_TC (USART_TypeDef * USARTx)

Function description

Enable Transmission Complete Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TCIE LL_USART_EnableIT_TC

LL_USART_EnableIT_TXE_TXFNF

Function name

__STATIC_INLINE void LL_USART_EnableIT_TXE_TXFNF (USART_TypeDef * USARTx)

Function description

Enable TX Empty and TX FIFO Not Full Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 TXEIE_TXFNFIE LL_USART_EnableIT_TXE_TXFNF

LL_USART_EnableIT_PE

Function name

__STATIC_INLINE void LL_USART_EnableIT_PE (USART_TypeDef * USARTx)

Function description

Enable Parity Error Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 PEIE LL_USART_EnableIT_PE

LL_USART_EnableIT_CM

Function name

__STATIC_INLINE void LL_USART_EnableIT_CM (USART_TypeDef * USARTx)

Function description

Enable Character Match Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 CMIE LL_USART_EnableIT_CM

LL_USART_EnableIT_RTO

Function name

__STATIC_INLINE void LL_USART_EnableIT_RTO (USART_TypeDef * USARTx)

Function description

Enable Receiver Timeout Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RTOIE LL_USART_EnableIT_RTO

LL_USART_EnableIT_EOB

Function name

__STATIC_INLINE void LL_USART_EnableIT_EOB (USART_TypeDef * USARTx)

Function description

Enable End Of Block Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 EOBIE LL_USART_EnableIT_EOB

LL_USART_EnableIT_TXFE

Function name

__STATIC_INLINE void LL_USART_EnableIT_TXFE (USART_TypeDef * USARTx)

Function description

Enable TX FIFO Empty Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 TXFEIE LL_USART_EnableIT_TXFE

LL_USART_EnableIT_RXFF

Function name

__STATIC_INLINE void LL_USART_EnableIT_RXFF (USART_TypeDef * USARTx)

Function description

Enable RX FIFO Full Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RXFFIE LL_USART_EnableIT_RXFF

LL_USART_EnableIT_LBD

Function name

__STATIC_INLINE void LL_USART_EnableIT_LBD (USART_TypeDef * USARTx)

Function description

Enable LIN Break Detection Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 LBDIE LL_USART_EnableIT_LBD

LL_USART_EnableIT_ERROR

Function name

__STATIC_INLINE void LL_USART_EnableIT_ERROR (USART_TypeDef * USARTx)

Function description

Enable Error Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- When set, Error Interrupt Enable Bit is enabling interrupt generation in case of a framing error, overrun error or noise flag (FE=1 or ORE=1 or NF=1 in the USARTx_ISR register). 0: Interrupt is inhibited 1: An interrupt is generated when FE=1 or ORE=1 or NF=1 in the USARTx_ISR register.

Reference Manual to LL API cross reference:

- CR3 EIE LL_USART_EnableIT_ERROR

LL_USART_EnableIT_CTS

Function name

__STATIC_INLINE void LL_USART_EnableIT_CTS (USART_TypeDef * USARTx)

Function description

Enable CTS Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 CTSIE LL_USART_EnableIT_CTS

LL_USART_EnableIT_WKUP

Function name

__STATIC_INLINE void LL_USART_EnableIT_WKUP (USART_TypeDef * USARTx)

Function description

Enable Wake Up from Stop Mode Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_WAKEUP_FROMSTOP_INSTANCE(USARTx) can be used to check whether or not Wake-up from Stop mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 WUFIE LL_USART_EnableIT_WKUP

LL_USART_EnableIT_TXFT

Function name

__STATIC_INLINE void LL_USART_EnableIT_TXFT (USART_TypeDef * USARTx)

Function description

Enable TX FIFO Threshold Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 TXFTIE LL_USART_EnableIT_TXFT

LL_USART_EnableIT_TCBGT

Function name

__STATIC_INLINE void LL_USART_EnableIT_TCBGT (USART_TypeDef * USARTx)

Function description

Enable Smartcard Transmission Complete Before Guard Time Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 TCBGTIE LL_USART_EnableIT_TCBGT

LL_USART_EnableIT_RXFT

Function name

__STATIC_INLINE void LL_USART_EnableIT_RXFT (USART_TypeDef * USARTx)

Function description

Enable RX FIFO Threshold Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 RXFTIE LL_USART_EnableIT_RXFT

LL_USART_DisableIT_IDLE

Function name

__STATIC_INLINE void LL_USART_DisableIT_IDLE (USART_TypeDef * USARTx)

Function description

Disable IDLE Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 IDLEIE LL_USART_DisableIT_IDLE

LL_USART_DisableIT_RXNE_RXFNE

Function name

__STATIC_INLINE void LL_USART_DisableIT_RXNE_RXFNE (USART_TypeDef * USARTx)

Function description

Disable RX Not Empty and RX FIFO Not Empty Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 RXNEIE_RXFNEIE LL_USART_DisableIT_RXNE_RXFNE

LL_USART_DisableIT_TC

Function name

__STATIC_INLINE void LL_USART_DisableIT_TC (USART_TypeDef * USARTx)

Function description

Disable Transmission Complete Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TCIE LL_USART_DisableIT_TC

LL_USART_DisableIT_TXE_TXFNF

Function name

__STATIC_INLINE void LL_USART_DisableIT_TXE_TXFNF (USART_TypeDef * USARTx)

Function description

Disable TX Empty and TX FIFO Not Full Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 TXEIE_TXFNFIE LL_USART_DisableIT_TXE_TXFNF

LL_USART_DisableIT_PE

Function name

__STATIC_INLINE void LL_USART_DisableIT_PE (USART_TypeDef * USARTx)

Function description

Disable Parity Error Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 PEIE LL_USART_DisableIT_PE

LL_USART_DisableIT_CM

Function name

__STATIC_INLINE void LL_USART_DisableIT_CM (USART_TypeDef * USARTx)

Function description

Disable Character Match Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 CMIE LL_USART_DisableIT_CM

LL_USART_DisableIT_RTO

Function name

__STATIC_INLINE void LL_USART_DisableIT_RTO (USART_TypeDef * USARTx)

Function description

Disable Receiver Timeout Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RTOIE LL_USART_DisableIT_RTO

LL_USART_DisableIT_EOB

Function name

__STATIC_INLINE void LL_USART_DisableIT_EOB (USART_TypeDef * USARTx)

Function description

Disable End Of Block Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 EOBIE LL_USART_DisableIT_EOB

LL_USART_DisableIT_TXFE

Function name

__STATIC_INLINE void LL_USART_DisableIT_TXFE (USART_TypeDef * USARTx)

Function description

Disable TX FIFO Empty Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 TXFEIE LL_USART_DisableIT_TXFE

LL_USART_DisableIT_RXFF

Function name

__STATIC_INLINE void LL_USART_DisableIT_RXFF (USART_TypeDef * USARTx)

Function description

Disable RX FIFO Full Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 RXFFIE LL_USART_DisableIT_RXFF

LL_USART_DisableIT_LBD

Function name

__STATIC_INLINE void LL_USART_DisableIT_LBD (USART_TypeDef * USARTx)

Function description

Disable LIN Break Detection Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 LBDIE LL_USART_DisableIT_LBD

LL_USART_DisableIT_ERROR

Function name

__STATIC_INLINE void LL_USART_DisableIT_ERROR (USART_TypeDef * USARTx)

Function description

Disable Error Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- When set, Error Interrupt Enable Bit is enabling interrupt generation in case of a framing error, overrun error or noise flag (FE=1 or ORE=1 or NF=1 in the USARTx_ISR register). 0: Interrupt is inhibited 1: An interrupt is generated when FE=1 or ORE=1 or NF=1 in the USARTx_ISR register.

Reference Manual to LL API cross reference:

- CR3 EIE LL_USART_DisableIT_ERROR

LL_USART_DisableIT_CTS

Function name

__STATIC_INLINE void LL_USART_DisableIT_CTS (USART_TypeDef * USARTx)

Function description

Disable CTS Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 CTSIE LL_USART_DisableIT_CTS

LL_USART_DisableIT_WKUP

Function name

__STATIC_INLINE void LL_USART_DisableIT_WKUP (USART_TypeDef * USARTx)

Function description

Disable Wake Up from Stop Mode Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_WAKEUP_FROMSTOP_INSTANCE(USARTx) can be used to check whether or not Wake-up from Stop mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 WUFIE LL_USART_DisableIT_WKUP

LL_USART_DisableIT_TXFT

Function name

__STATIC_INLINE void LL_USART_DisableIT_TXFT (USART_TypeDef * USARTx)

Function description

Disable TX FIFO Threshold Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 TXFTIE LL_USART_DisableIT_TXFT

LL_USART_DisableIT_TCBGT

Function name

__STATIC_INLINE void LL_USART_DisableIT_TCBGT (USART_TypeDef * USARTx)

Function description

Disable Smartcard Transmission Complete Before Guard Time Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 TCBGTIE LL_USART_DisableIT_TCBGT

LL_USART_DisableIT_RXFT

Function name

__STATIC_INLINE void LL_USART_DisableIT_RXFT (USART_TypeDef * USARTx)

Function description

Disable RX FIFO Threshold Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 RXFTIE LL_USART_DisableIT_RXFT

LL_USART_IsEnabledIT_IDLE

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_IDLE (const USART_TypeDef * USARTx)

Function description

Check if the USART IDLE Interrupt source is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 IDLEIE LL_USART_IsEnabledIT_IDLE

LL_USART_IsEnabledIT_RXNE_RXFNE

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_RXNE_RXFNE (const USART_TypeDef * USARTx)

Function description

Check if the USART RX Not Empty and USART RX FIFO Not Empty Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 RXNEIE_RXFNEIE LL_USART_IsEnabledIT_RXNE_RXFNE

LL_USART_IsEnabledIT_TC

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_TC (const USART_TypeDef * USARTx)

Function description

Check if the USART Transmission Complete Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 TCIE LL_USART_IsEnabledIT_TC

LL_USART_IsEnabledIT_TXE_TXFNF

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_TXE_TXFNF (const USART_TypeDef * USARTx)

Function description

Check if the USART TX Empty and USART TX FIFO Not Full Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 TXEIE_TXFNFIE LL_USART_IsEnabledIT_TXE_TXFNF

LL_USART_IsEnabledIT_PE

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_PE (const USART_TypeDef * USARTx)

Function description

Check if the USART Parity Error Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 PEIE LL_USART_IsEnabledIT_PE

LL_USART_IsEnabledIT_CM

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_CM (const USART_TypeDef * USARTx)

Function description

Check if the USART Character Match Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 CMIE LL_USART_IsEnabledIT_CM

LL_USART_IsEnabledIT_RTO

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_RTO (const USART_TypeDef * USARTx)

Function description

Check if the USART Receiver Timeout Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 RTOIE LL_USART_IsEnabledIT_RTO

LL_USART_IsEnabledIT_EOB

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_EOB (const USART_TypeDef * USARTx)

Function description

Check if the USART End Of Block Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 EOBIE LL_USART_IsEnabledIT_EOB

LL_USART_IsEnabledIT_TXFE

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_TXFE (const USART_TypeDef * USARTx)

Function description

Check if the USART TX FIFO Empty Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 TXFEIE LL_USART_IsEnabledIT_TXFE

LL_USART_IsEnabledIT_RXFF

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_RXFF (const USART_TypeDef * USARTx)

Function description

Check if the USART RX FIFO Full Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 RXFFIE LL_USART_IsEnabledIT_RXFF

LL_USART_IsEnabledIT_LBD

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_LBD (const USART_TypeDef * USARTx)

Function description

Check if the USART LIN Break Detection Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 LBDIE LL_USART_IsEnabledIT_LBD

LL_USART_IsEnabledIT_ERROR

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_ERROR (const USART_TypeDef * USARTx)

Function description

Check if the USART Error Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 EIE LL_USART_IsEnabledIT_ERROR

LL_USART_IsEnabledIT_CTS

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_CTS (const USART_TypeDef * USARTx)

Function description

Check if the USART CTS Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 CTSIE LL_USART_IsEnabledIT_CTS

LL_USART_IsEnabledIT_WKUP

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_WKUP (const USART_TypeDef * USARTx)

Function description

Check if the USART Wake Up from Stop Mode Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_WAKEUP_FROMSTOP_INSTANCE(USARTx) can be used to check whether or not Wake-up from Stop mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 WUFIE LL_USART_IsEnabledIT_WKUP

LL_USART_IsEnabledIT_TXFT

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_TXFT (const USART_TypeDef * USARTx)

Function description

Check if USART TX FIFO Threshold Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 TXFTIE LL_USART_IsEnabledIT_TXFT

LL_USART_IsEnabledIT_TCBGT

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_TCBGT (const USART_TypeDef * USARTx)

Function description

Check if the Smartcard Transmission Complete Before Guard Time Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 TCBGTIE LL_USART_IsEnabledIT_TCBGT

LL_USART_IsEnabledIT_RXFT

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_RXFT (const USART_TypeDef * USARTx)

Function description

Check if USART RX FIFO Threshold Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 RXFTIE LL_USART_IsEnabledIT_RXFT

LL_USART_EnableDMAReq_RX

Function name

__STATIC_INLINE void LL_USART_EnableDMAReq_RX (USART_TypeDef * USARTx)

Function description

Enable DMA Mode for reception.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DMAR LL_USART_EnableDMAReq_RX

LL_USART_DisableDMAReq_RX

Function name

__STATIC_INLINE void LL_USART_DisableDMAReq_RX (USART_TypeDef * USARTx)

Function description

Disable DMA Mode for reception.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DMAR LL_USART_DisableDMAReq_RX

LL_USART_IsEnabledDMAReq_RX

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledDMAReq_RX (const USART_TypeDef * USARTx)

Function description

Check if DMA Mode is enabled for reception.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 DMAR LL_USART_IsEnabledDMAReq_RX

LL_USART_EnableDMAReq_TX

Function name

__STATIC_INLINE void LL_USART_EnableDMAReq_TX (USART_TypeDef * USARTx)

Function description

Enable DMA Mode for transmission.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DMAT LL_USART_EnableDMAReq_TX

LL_USART_DisableDMAReq_TX

Function name

__STATIC_INLINE void LL_USART_DisableDMAReq_TX (USART_TypeDef * USARTx)

Function description

Disable DMA Mode for transmission.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DMAT LL_USART_DisableDMAReq_TX

LL_USART_IsEnabledDMAReq_TX

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledDMAReq_TX (const USART_TypeDef * USARTx)

Function description

Check if DMA Mode is enabled for transmission.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 DMAT LL_USART_IsEnabledDMAReq_TX

LL_USART_EnableDMADeactOnRxErr

Function name

__STATIC_INLINE void LL_USART_EnableDMADeactOnRxErr (USART_TypeDef * USARTx)

Function description

Enable DMA Disabling on Reception Error.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DDRE LL_USART_EnableDMADeactOnRxErr

LL_USART_DisableDMADeactOnRxErr

Function name

__STATIC_INLINE void LL_USART_DisableDMADeactOnRxErr (USART_TypeDef * USARTx)

Function description

Disable DMA Disabling on Reception Error.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DDRE LL_USART_DisableDMADeactOnRxErr

LL_USART_IsEnabledDMADeactOnRxErr

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledDMADeactOnRxErr (const USART_TypeDef * USARTx)

Function description

Indicate if DMA Disabling on Reception Error is disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 DDRE LL_USART_IsEnabledDMADeactOnRxErr

LL_USART_DMA_GetRegAddr

Function name

__STATIC_INLINE uint32_t LL_USART_DMA_GetRegAddr (const USART_TypeDef * USARTx, uint32_t Direction)

Function description

Get the data register address used for DMA transfer.

Parameters

- **USARTx:** USART Instance
- **Direction:** This parameter can be one of the following values:
  - LL_USART_DMA_REG_DATA_TRANSMIT
  - LL_USART_DMA_REG_DATA_RECEIVE

Return values

- **Address:** of data register

Reference Manual to LL API cross reference:

- RDR RDR LL_USART_DMA_GetRegAddr
-
- TDR TDR LL_USART_DMA_GetRegAddr

LL_USART_ReceiveData8

Function name

__STATIC_INLINE uint8_t LL_USART_ReceiveData8 (const USART_TypeDef * USARTx)

Function description

Read Receiver Data register (Receive Data value, 8 bits)

Parameters

- **USARTx:** USART Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- RDR RDR LL_USART_ReceiveData8

LL_USART_ReceiveData9

Function name

__STATIC_INLINE uint16_t LL_USART_ReceiveData9 (const USART_TypeDef * USARTx)

Function description

Read Receiver Data register (Receive Data value, 9 bits)

Parameters

- **USARTx:** USART Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x1FF

Reference Manual to LL API cross reference:

- RDR RDR LL_USART_ReceiveData9

LL_USART_TransmitData8

Function name

__STATIC_INLINE void LL_USART_TransmitData8 (USART_TypeDef * USARTx, uint8_t Value)

Function description

Write in Transmitter Data Register (Transmit Data value, 8 bits)

Parameters

- **USARTx:** USART Instance
- **Value:** between Min_Data=0x00 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- TDR TDR LL_USART_TransmitData8

LL_USART_TransmitData9

Function name

__STATIC_INLINE void LL_USART_TransmitData9 (USART_TypeDef * USARTx, uint16_t Value)

Function description

Write in Transmitter Data Register (Transmit Data value, 9 bits)

Parameters

- **USARTx:** USART Instance
- **Value:** between Min_Data=0x00 and Max_Data=0x1FF

Return values

- **None:**

Reference Manual to LL API cross reference:

- TDR TDR LL_USART_TransmitData9

LL_USART_RequestAutoBaudRate

Function name

__STATIC_INLINE void LL_USART_RequestAutoBaudRate (USART_TypeDef * USARTx)

Function description

Request an Automatic Baud Rate measurement on next received data frame.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_USART_AUTOBAUDRATE_DETECTION_INSTANCE(USARTx) can be used to check whether or not Auto Baud Rate detection feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- RQR ABRRQ LL_USART_RequestAutoBaudRate

LL_USART_RequestBreakSending

Function name

__STATIC_INLINE void LL_USART_RequestBreakSending (USART_TypeDef * USARTx)

Function description

Request Break sending.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RQR SBKRQ LL_USART_RequestBreakSending

LL_USART_RequestEnterMuteMode

Function name

__STATIC_INLINE void LL_USART_RequestEnterMuteMode (USART_TypeDef * USARTx)

Function description

Put USART in mute mode and set the RWU flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RQR MMRQ LL_USART_RequestEnterMuteMode

LL_USART_RequestRxDataFlush

Function name

__STATIC_INLINE void LL_USART_RequestRxDataFlush (USART_TypeDef * USARTx)

Function description

Request a Receive Data and FIFO flush.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.
- Allows to discard the received data without reading them, and avoid an overrun condition.

Reference Manual to LL API cross reference:

- RQR RXFRQ LL_USART_RequestRxDataFlush

LL_USART_RequestTxDataFlush

Function name

__STATIC_INLINE void LL_USART_RequestTxDataFlush (USART_TypeDef * USARTx)

Function description

Request a Transmit data and FIFO flush.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- RQR TXFRQ LL_USART_RequestTxDataFlush

LL_USART_DeInit

Function name

ErrorStatus LL_USART_DeInit (const USART_TypeDef * USARTx)

Function description

De-initialize USART registers (Registers restored to their default values).

Parameters

- **USARTx:** USART Instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: USART registers are de-initialized
  - ERROR: USART registers are not de-initialized

LL_USART_Init

Function name

ErrorStatus LL_USART_Init (USART_TypeDef * USARTx, const LL_USART_InitTypeDef * USART_InitStruct)

Function description

Initialize USART registers according to the specified parameters in USART_InitStruct.

Parameters

- **USARTx:** USART Instance
- **USART_InitStruct:** pointer to a LL_USART_InitTypeDef structure that contains the configuration information for the specified USART peripheral.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: USART registers are initialized according to USART_InitStruct content
  - ERROR: Problem occurred during USART Registers initialization

Notes

- As some bits in USART configuration registers can only be written when the USART is disabled (USART_CR1_UE bit =0), USART Peripheral should be in disabled state prior calling this function. Otherwise, ERROR result will be returned.
- Baud rate value stored in USART_InitStruct BaudRate field, should be valid (different from 0).

LL_USART_StructInit

Function name

void LL_USART_StructInit (LL_USART_InitTypeDef * USART_InitStruct)

Function description

Set each LL_USART_InitTypeDef field to default value.

Parameters

- **USART_InitStruct:** pointer to a LL_USART_InitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

LL_USART_ClockInit

Function name

ErrorStatus LL_USART_ClockInit (USART_TypeDef * USARTx, const LL_USART_ClockInitTypeDef * USART_ClockInitStruct)

Function description

Initialize USART Clock related settings according to the specified parameters in the USART_ClockInitStruct.

Parameters

- **USARTx:** USART Instance
- **USART_ClockInitStruct:** pointer to a LL_USART_ClockInitTypeDef structure that contains the Clock configuration information for the specified USART peripheral.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: USART registers related to Clock settings are initialized according to USART_ClockInitStruct content
  - ERROR: Problem occurred during USART Registers initialization

Notes

- As some bits in USART configuration registers can only be written when the USART is disabled (USART_CR1_UE bit =0), USART Peripheral should be in disabled state prior calling this function. Otherwise, ERROR result will be returned.

LL_USART_ClockStructInit

Function name

void LL_USART_ClockStructInit (LL_USART_ClockInitTypeDef * USART_ClockInitStruct)

Function description

Set each field of a LL_USART_ClockInitTypeDef type structure to default value.

Parameters

- **USART_ClockInitStruct:** pointer to a LL_USART_ClockInitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

108.3 USART Firmware driver defines

The following section lists the various define and macros of the module.

108.3.1 USART

USART

Address Length Detection

LL_USART_ADDRESS_DETECT_4B

4-bit address detection method selected

LL_USART_ADDRESS_DETECT_7B

7-bit address detection (in 8-bit data mode) method selected

Autobaud Detection

LL_USART_AUTOBAUD_DETECT_ON_STARTBIT

Measurement of the start bit is used to detect the baud rate

LL_USART_AUTOBAUD_DETECT_ON_FALLINGEDGE

Falling edge to falling edge measurement. Received frame must start with a single bit = 1 -> Frame = Start10xxxxxx

LL_USART_AUTOBAUD_DETECT_ON_7F_FRAME

0x7F frame detection

LL_USART_AUTOBAUD_DETECT_ON_55_FRAME

0x55 frame detection

Binary Data Inversion

LL_USART_BINARY_LOGIC_POSITIVE

Logical data from the data register are send/received in positive/direct logic. (1=H, 0=L)

LL_USART_BINARY_LOGIC_NEGATIVE

Logical data from the data register are send/received in negative/inverse logic. (1=L, 0=H). The parity bit is also inverted.

Bit Order

LL_USART_BITORDER_LSBFIRST

data is transmitted/received with data bit 0 first, following the start bit

LL_USART_BITORDER_MSBFIRST

data is transmitted/received with the MSB first, following the start bit

Clear Flags Defines

LL_USART_ICR_PECF

Parity error clear flag

LL_USART_ICR_FECF

Framing error clear flag

LL_USART_ICR_NECF

Noise error detected clear flag

LL_USART_ICR_ORECF

Overrun error clear flag

LL_USART_ICR_IDLECF

Idle line detected clear flag

LL_USART_ICR_TXFECF

TX FIFO Empty clear flag

LL_USART_ICR_TCCF

Transmission complete clear flag

LL_USART_ICR_TCBGTCF

Transmission completed before guard time clear flag

LL_USART_ICR_LBDCF

LIN break detection clear flag

LL_USART_ICR_CTSCF

CTS clear flag

LL_USART_ICR_RTOCF

Receiver timeout clear flag

LL_USART_ICR_EOBCF

End of block clear flag

LL_USART_ICR_UDRCF

SPI Slave Underrun clear flag

LL_USART_ICR_CMCF

Character match clear flag

LL_USART_ICR_WUCF

Wakeup from Stop mode clear flag

Clock Signal

LL_USART_CLOCK_DISABLE

Clock signal not provided

LL_USART_CLOCK_ENABLE

Clock signal provided

Datawidth

LL_USART_DATAWIDTH_7B

7 bits word length : Start bit, 7 data bits, n stop bits

LL_USART_DATAWIDTH_8B

8 bits word length : Start bit, 8 data bits, n stop bits

LL_USART_DATAWIDTH_9B

9 bits word length : Start bit, 9 data bits, n stop bits

Driver Enable Polarity

LL_USART_DE_POLARITY_HIGH

DE signal is active high

LL_USART_DE_POLARITY_LOW

DE signal is active low

Communication Direction

LL_USART_DIRECTION_NONE

Transmitter and Receiver are disabled

LL_USART_DIRECTION_RX

Transmitter is disabled and Receiver is enabled

LL_USART_DIRECTION_TX

Transmitter is enabled and Receiver is disabled

LL_USART_DIRECTION_TX_RX

Transmitter and Receiver are enabled

DMA Register Data

LL_USART_DMA_REG_DATA_TRANSMIT

Get address of data register used for transmission

LL_USART_DMA_REG_DATA_RECEIVE

Get address of data register used for reception

FIFO Threshold

LL_USART_FIFOTHRESHOLD_1_8

FIFO reaches 1/8 of its depth

LL_USART_FIFOTHRESHOLD_1_4

FIFO reaches 1/4 of its depth

LL_USART_FIFOTHRESHOLD_1_2

FIFO reaches 1/2 of its depth

LL_USART_FIFOTHRESHOLD_3_4

FIFO reaches 3/4 of its depth

LL_USART_FIFOTHRESHOLD_7_8

FIFO reaches 7/8 of its depth

LL_USART_FIFOTHRESHOLD_8_8

FIFO becomes empty for TX and full for RX

Get Flags Defines

LL_USART_ISR_PE

Parity error flag

LL_USART_ISR_FE

Framing error flag

LL_USART_ISR_NE

Noise detected flag

LL_USART_ISR_ORE

Overrun error flag

LL_USART_ISR_IDLE

Idle line detected flag

LL_USART_ISR_RXNE_RXFNE

Read data register or RX FIFO not empty flag

LL_USART_ISR_TC

Transmission complete flag

LL_USART_ISR_TXE_TXFNF

Transmit data register empty or TX FIFO Not Full flag

LL_USART_ISR_LBDF

LIN break detection flag

LL_USART_ISR_CTSIF

CTS interrupt flag

LL_USART_ISR_CTS

CTS flag

LL_USART_ISR_RTOF

Receiver timeout flag

LL_USART_ISR_EOBF

End of block flag

LL_USART_ISR_UDR

SPI Slave underrun error flag

LL_USART_ISR_ABRE

Auto baud rate error flag

LL_USART_ISR_ABRF

Auto baud rate flag

LL_USART_ISR_BUSY

Busy flag

LL_USART_ISR_CMF

Character match flag

LL_USART_ISR_SBKF

Send break flag

LL_USART_ISR_RWU

Receiver wakeup from Mute mode flag

LL_USART_ISR_WUF

Wakeup from Stop mode flag

LL_USART_ISR_TEACK

Transmit enable acknowledge flag

LL_USART_ISR_REACK

Receive enable acknowledge flag

LL_USART_ISR_TXFE

TX FIFO empty flag

LL_USART_ISR_RXFF

RX FIFO full flag

LL_USART_ISR_TCBGT

Transmission complete before guard time completion flag

LL_USART_ISR_RXFT

RX FIFO threshold flag

LL_USART_ISR_TXFT

TX FIFO threshold flag

Hardware Control

LL_USART_HWCONTROL_NONE

CTS and RTS hardware flow control disabled

LL_USART_HWCONTROL_RTS

RTS output enabled, data is only requested when there is space in the receive buffer

LL_USART_HWCONTROL_CTS

CTS mode enabled, data is only transmitted when the nCTS input is asserted (tied to 0)

LL_USART_HWCONTROL_RTS_CTS

CTS and RTS hardware flow control enabled

IrDA Power

LL_USART_IRDA_POWER_NORMAL

IrDA normal power mode

LL_USART_IRDA_POWER_LOW

IrDA low power mode

IT Defines

LL_USART_CR1_IDLEIE

IDLE interrupt enable

LL_USART_CR1_RXNEIE_RXFNEIE

Read data register and RXFIFO not empty interrupt enable

LL_USART_CR1_TCIE

Transmission complete interrupt enable

LL_USART_CR1_TXEIE_TXFNFIE

Transmit data register empty and TX FIFO not full interrupt enable

LL_USART_CR1_PEIE

Parity error

LL_USART_CR1_CMIE

Character match interrupt enable

LL_USART_CR1_RTOIE

Receiver timeout interrupt enable

LL_USART_CR1_EOBIE

End of Block interrupt enable

LL_USART_CR1_TXFEIE

TX FIFO empty interrupt enable

LL_USART_CR1_RXFFIE

RX FIFO full interrupt enable

LL_USART_CR2_LBDIE

LIN break detection interrupt enable

LL_USART_CR3_EIE

Error interrupt enable

LL_USART_CR3_CTSIE

CTS interrupt enable

LL_USART_CR3_WUFIE

Wakeup from Stop mode interrupt enable

LL_USART_CR3_TXFTIE

TX FIFO threshold interrupt enable

LL_USART_CR3_TCBGTIE

Transmission complete before guard time interrupt enable

LL_USART_CR3_RXFTIE

RX FIFO threshold interrupt enable

Last Clock Pulse

LL_USART_LASTCLKPULSE_NO_OUTPUT

The clock pulse of the last data bit is not output to the SCLK pin

LL_USART_LASTCLKPULSE_OUTPUT

The clock pulse of the last data bit is output to the SCLK pin

LIN Break Detection Length

LL_USART_LINBREAK_DETECT_10B

10-bit break detection method selected

LL_USART_LINBREAK_DETECT_11B

11-bit break detection method selected

Oversampling

LL_USART_OVERSAMPLING_16

Oversampling by 16

LL_USART_OVERSAMPLING_8

Oversampling by 8

Parity Control

LL_USART_PARITY_NONE

Parity control disabled

LL_USART_PARITY_EVEN

Parity control enabled and Even Parity is selected

LL_USART_PARITY_ODD

Parity control enabled and Odd Parity is selected

Clock Phase

LL_USART_PHASE_1EDGE

The first clock transition is the first data capture edge

LL_USART_PHASE_2EDGE

The second clock transition is the first data capture edge

Clock Polarity

LL_USART_POLARITY_LOW

Steady low value on SCLK pin outside transmission window

LL_USART_POLARITY_HIGH

Steady high value on SCLK pin outside transmission window

Clock Source Prescaler

LL_USART_PRESCALER_DIV1

Input clock not divided

LL_USART_PRESCALER_DIV2

Input clock divided by 2

LL_USART_PRESCALER_DIV4

Input clock divided by 4

LL_USART_PRESCALER_DIV6

Input clock divided by 6

LL_USART_PRESCALER_DIV8

Input clock divided by 8

LL_USART_PRESCALER_DIV10

Input clock divided by 10

LL_USART_PRESCALER_DIV12

Input clock divided by 12

LL_USART_PRESCALER_DIV16

Input clock divided by 16

LL_USART_PRESCALER_DIV32

Input clock divided by 32

LL_USART_PRESCALER_DIV64

Input clock divided by 64

LL_USART_PRESCALER_DIV128

Input clock divided by 128

LL_USART_PRESCALER_DIV256

Input clock divided by 256

RX Pin Active Level Inversion

LL_USART_RXPIN_LEVEL_STANDARD

RX pin signal works using the standard logic levels

LL_USART_RXPIN_LEVEL_INVERTED

RX pin signal values are inverted.

Stop Bits

LL_USART_STOPBITS_0_5

0.5 stop bit

LL_USART_STOPBITS_1

1 stop bit

LL_USART_STOPBITS_1_5

1.5 stop bits

LL_USART_STOPBITS_2

2 stop bits

TX Pin Active Level Inversion

LL_USART_TXPIN_LEVEL_STANDARD

TX pin signal works using the standard logic levels

LL_USART_TXPIN_LEVEL_INVERTED

TX pin signal values are inverted.

TX RX Pins Swap

LL_USART_TXRX_STANDARD

TX/RX pins are used as defined in standard pinout

LL_USART_TXRX_SWAPPED

TX and RX pins functions are swapped.

Wakeup

LL_USART_WAKEUP_IDLELINE

USART wake up from Mute mode on Idle Line

LL_USART_WAKEUP_ADDRESSMARK

USART wake up from Mute mode on Address Mark

Wakeup Activation

LL_USART_WAKEUP_ON_ADDRESS

Wake up active on address match

LL_USART_WAKEUP_ON_STARTBIT

Wake up active on Start bit detection

LL_USART_WAKEUP_ON_RXNE

Wake up active on RXNE

FLAG_Management

LL_USART_IsActiveFlag_RXNE

LL_USART_IsActiveFlag_TXE

IT_Management

LL_USART_EnableIT_RXNE

LL_USART_EnableIT_TXE

LL_USART_DisableIT_RXNE

LL_USART_DisableIT_TXE

LL_USART_IsEnabledIT_RXNE

LL_USART_IsEnabledIT_TXE

Exported_Macros_Helper

__LL_USART_DIV_SAMPLING8

Description:

- Compute USARTDIV value according to Peripheral Clock and expected Baud Rate in 8 bits sampling mode (32 bits value of USARTDIV is returned)

Parameters:

- __PERIPHCLK__: Peripheral Clock frequency used for USART instance
- __PRESCALER__: This parameter can be one of the following values:
  - LL_USART_PRESCALER_DIV1
  - LL_USART_PRESCALER_DIV2
  - LL_USART_PRESCALER_DIV4
  - LL_USART_PRESCALER_DIV6
  - LL_USART_PRESCALER_DIV8
  - LL_USART_PRESCALER_DIV10
  - LL_USART_PRESCALER_DIV12
  - LL_USART_PRESCALER_DIV16
  - LL_USART_PRESCALER_DIV32
  - LL_USART_PRESCALER_DIV64
  - LL_USART_PRESCALER_DIV128
  - LL_USART_PRESCALER_DIV256
- __BAUDRATE__: Baud rate value to achieve

Return value:

- USARTDIV: value to be used for BRR register filling in OverSampling_8 case

__LL_USART_DIV_SAMPLING16

Description:

- Compute USARTDIV value according to Peripheral Clock and expected Baud Rate in 16 bits sampling mode (32 bits value of USARTDIV is returned)

Parameters:

- __PERIPHCLK__: Peripheral Clock frequency used for USART instance
- __PRESCALER__: This parameter can be one of the following values:
  - LL_USART_PRESCALER_DIV1
  - LL_USART_PRESCALER_DIV2
  - LL_USART_PRESCALER_DIV4
  - LL_USART_PRESCALER_DIV6
  - LL_USART_PRESCALER_DIV8
  - LL_USART_PRESCALER_DIV10
  - LL_USART_PRESCALER_DIV12
  - LL_USART_PRESCALER_DIV16
  - LL_USART_PRESCALER_DIV32
  - LL_USART_PRESCALER_DIV64
  - LL_USART_PRESCALER_DIV128
  - LL_USART_PRESCALER_DIV256
- __BAUDRATE__: Baud rate value to achieve

Return value:

- USARTDIV: value to be used for BRR register filling in OverSampling_16 case

Common Write and read registers Macros

LL_USART_WriteReg

Description:

- Write a value in USART register.

Parameters:

- __INSTANCE__: USART Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_USART_ReadReg

Description:

- Read a value in USART register.

Parameters:

- __INSTANCE__: USART Instance
- __REG__: Register to be read

Return value:

- Register: value

109 LL UTILS Generic Driver

109.1 UTILS Firmware driver registers structures

109.1.1 LL_UTILS_PLLInitTypeDef

LL_UTILS_PLLInitTypeDef is defined in the stm32n6xx_ll_utils.h

Data Fields

- ***uint32_t PLLM***
- ***uint32_t PLLN***
- ***uint32_t PLLP1***
- ***uint32_t PLLP2***
- ***uint32_t FRACN***

Field Documentation

- ***uint32_t LL_UTILS_PLLInitTypeDef::PLLM***
  Division factor M for PLL VCO input clock. This parameter must be a number between Min_Data = 1 and Max_Data = 63 and shall ensure that VCO input is below 50 MHz.This feature can be modified afterwards using unitary function **LL_RCC_PLL1_SetM()**.
- ***uint32_t LL_UTILS_PLLInitTypeDef::PLLN***
  Multiplication factor N for PLL VCO output clock. This parameter must be a number between Min_Data = 16 and Max_Data = 640 in integer mode and between Min_Data = 20 and Max_Data = 320 in fractional modeThis feature can be modified afterwards using unitary function **LL_RCC_PLL1_SetN()**.
- ***uint32_t LL_UTILS_PLLInitTypeDef::PLLP1***
  Division factor level 1 for PLL VCO output clock. This parameter must be a number between Min_Data = 1 and Max_Data = 7This feature can be modified afterwards using unitary function **LL_RCC_PLL1_SetP1()**.
- ***uint32_t LL_UTILS_PLLInitTypeDef::PLLP2***
  Division factor level 2 for PLL VCO output clock. This parameter must be a number between Min_Data = 1 and Max_Data = 7This feature can be modified afterwards using unitary function **LL_RCC_PLL1_SetP2()**.
- ***uint32_t LL_UTILS_PLLInitTypeDef::FRACN***
  Fractional part of the multiplication factor for PLL VCO. This parameter can be a value between 0 and (2^24)-1This feature can be modified afterwards using unitary function **LL_RCC_PLL1_SetFRACN()**.

109.1.2 LL_UTILS_ICInitTypeDef

LL_UTILS_ICInitTypeDef is defined in the stm32n6xx_ll_utils.h

Data Fields

- ***uint32_t IC1Source***
- ***uint32_t IC1Divider***
- ***uint32_t IC2Source***
- ***uint32_t IC2Divider***
- ***uint32_t IC6Source***
- ***uint32_t IC6Divider***
- ***uint32_t IC11Source***
- ***uint32_t IC11Divider***

Field Documentation

- ***uint32_t LL_UTILS_ICInitTypeDef::IC1Source***
  The IC1 clock source for sysa_ck(cpu_ck)
- ***uint32_t LL_UTILS_ICInitTypeDef::IC1Divider***
  The IC1 clock divider for sysa_ck(cpu_ck)
- ***uint32_t LL_UTILS_ICInitTypeDef::IC2Source***
  The IC2 clock source for sysb_ck
- ***uint32_t LL_UTILS_ICInitTypeDef::IC2Divider***
  The IC2 clock divider for sysb_ck

- ***uint32_t LL_UTILS_ICInitTypeDef::IC6Source***
  The IC6 clock source for sysc_ck
- ***uint32_t LL_UTILS_ICInitTypeDef::IC6Divider***
  The IC6 clock divider for sysc_ck
- ***uint32_t LL_UTILS_ICInitTypeDef::IC11Source***
  The IC11 clock source for sysd_ck
- ***uint32_t LL_UTILS_ICInitTypeDef::IC11Divider***
  The IC11 clock divider for sysd_ck

109.1.3 LL_UTILS_ClkInitTypeDef

LL_UTILS_ClkInitTypeDef is defined in the stm32n6xx_ll_utils.h

Data Fields

- ***uint32_t AHBCLKDivider***
- ***uint32_t APB1CLKDivider***
- ***uint32_t APB2CLKDivider***
- ***uint32_t APB4CLKDivider***
- ***uint32_t APB5CLKDivider***

Field Documentation

- ***uint32_t LL_UTILS_ClkInitTypeDef::AHBCLKDivider***
  The AHB clock (HCLK) divider. This clock is derived from the system clock (SYSCLK). This parameter can be a value of ***RCC_LL_EC_AHB_DIV***This feature can be modified afterwards using unitary function **LL_RCC_SetAHBPrescaler()**.
- ***uint32_t LL_UTILS_ClkInitTypeDef::APB1CLKDivider***
  The APB1 clock (PCLK1) divider. This clock is derived from the AHB clock (HCLK). This parameter can be a value of ***RCC_LL_EC_APB1_DIV***This feature can be modified afterwards using unitary function **LL_RCC_SetAPB1Prescaler()**.
- ***uint32_t LL_UTILS_ClkInitTypeDef::APB2CLKDivider***
  The APB2 clock (PCLK2) divider. This clock is derived from the AHB clock (HCLK). This parameter can be a value of ***RCC_LL_EC_APB2_DIV***This feature can be modified afterwards using unitary function **LL_RCC_SetAPB2Prescaler()**.
- ***uint32_t LL_UTILS_ClkInitTypeDef::APB4CLKDivider***
  The APB4 clock (PCLK4) divider. This clock is derived from the AHB clock (HCLK). This parameter can be a value of ***RCC_LL_EC_APB4_DIV***This feature can be modified afterwards using unitary function **LL_RCC_SetAPB4Prescaler()**.
- ***uint32_t LL_UTILS_ClkInitTypeDef::APB5CLKDivider***
  The APB5 clock (PCLK5) divider. This clock is derived from the AHB clock (HCLK). This parameter can be a value of ***RCC_LL_EC_APB5_DIV***This feature can be modified afterwards using unitary function **LL_RCC_SetAPB5Prescaler()**.

109.2 UTILS Firmware driver API description

The following section lists the various functions of the UTILS library.

109.2.1 System Configuration functions

System, AXI, AHB and APB buses clocks configuration

- The maximum frequency of the CPU is 800 MHz and AXI is 400 MHz.
- The maximum frequency of the HCLK, PCLK1, PCLK2, PCLK4 and PCLK5 is 200 MHz.

This section contains the following APIs:

- ***LL_SetSystemCoreClock()***
- ***LL_PLL_ConfigSystemClock_HSI()***
- ***LL_PLL_ConfigSystemClock_MSI()***
- ***LL_PLL_ConfigSystemClock_HSE()***

109.2.2 Detailed description of functions

LL_GetUID_Word0

Function name

__STATIC_INLINE uint32_t LL_GetUID_Word0 (void)

Function description

Get Word0 of the unique device identifier (UID based on 96 bits)

Return values

- **UID[31:0]:**

Notes

- The application must ensures that SYSCFG clock is enabled.

LL_GetUID_Word1

Function name

__STATIC_INLINE uint32_t LL_GetUID_Word1 (void)

Function description

Get Word1 of the unique device identifier (UID based on 96 bits)

Return values

- **UID[63:32]:**

Notes

- The application must ensures that SYSCFG clock is enabled.

LL_GetUID_Word2

Function name

__STATIC_INLINE uint32_t LL_GetUID_Word2 (void)

Function description

Get Word2 of the unique device identifier (UID based on 96 bits)

Return values

- **UID[95:64]:**

Notes

- The application must ensures that SYSCFG clock is enabled.

LL_GetPackageType

Function name

__STATIC_INLINE uint32_t LL_GetPackageType (void)

Function description

Get Package type.

Return values

- **Returned:** value can be one of the following values:
  - LL_UTILS_PACKAGETYPE_BGA142
  - LL_UTILS_PACKAGETYPE_BGA169
  - LL_UTILS_PACKAGETYPE_BGA178
  - LL_UTILS_PACKAGETYPE_BGA198
  - LL_UTILS_PACKAGETYPE_BGA223
  - LL_UTILS_PACKAGETYPE_BGA264

Notes

- The application must ensures that SYSCFG clock is enabled.

LL_GetDevicePartNumber

Function name

__STATIC_INLINE uint32_t LL_GetDevicePartNumber (void)

Function description

Get Device Part Number (RPN)

Return values

- **Returned:** value can be one of the following values:
  - LL_UTILS_DEVICE_STM32N657
  - LL_UTILS_DEVICE_STM32N647
  - LL_UTILS_DEVICE_STM32N645

Notes

- The application must ensures that SYSCFG clock is enabled.

LL_InitTick

Function name

__STATIC_INLINE void LL_InitTick (uint32_t HCLKFrequency, uint32_t Ticks)

Function description

This function configures the Cortex-M SysTick source of the time base.

Parameters

- **HCLKFrequency:** HCLK frequency in Hz (can be calculated thanks to RCC helper macro)
- **Ticks:** Number of ticks

Return values

- **None:**

Notes

- When a RTOS is used, it is recommended to avoid changing the SysTick configuration by calling this function, for a delay use rather osDelay RTOS service.

LL_Init1msTick

Function name

void LL_Init1msTick (uint32_t CPU_Frequency)

Function description

This function configures the Cortex-M SysTick source to have 1ms time base.

Parameters

- **CPU_Frequency:** Core frequency in Hz

Return values

- **None:**

Notes

- When a RTOS is used, it is recommended to avoid changing the Systick configuration by calling this function, for a delay use rather osDelay RTOS service.
- CPU_Frequency can be calculated thanks to RCC helper macro or function LL_RCC_GetSystemClocksFreq

LL_mDelay

Function name

void LL_mDelay (uint32_t Delay)

Function description

This function provides accurate delay (in milliseconds) based on SysTick counter flag.

Parameters

- **Delay:** specifies the delay time length, in milliseconds.

Return values

- **None:**

Notes

- When a RTOS is used, it is recommended to avoid using blocking delay and use rather osDelay service.
- To respect 1ms timebase, user should call LL_Init1msTick function which will configure Systick to 1ms

LL_SetSystemCoreClock

Function name

void LL_SetSystemCoreClock (uint32_t CPU_Frequency)

Function description

This function sets directly SystemCoreClock CMSIS variable.

Parameters

- **CPU_Frequency:** Core frequency in Hz

Return values

- **None:**

Notes

- Variable can be calculated also through SystemCoreClockUpdate function.
- CPU_Frequency can be calculated thanks to RCC helper macro or function LL_RCC_GetSystemClocksFreq

LL_PLL_ConfigSystemClock_HSI

Function name

ErrorStatus LL_PLL_ConfigSystemClock_HSI (const LL_UTILS_PLLInitTypeDef * pUTILS_PLLInitStruct, const LL_UTILS_ICInitTypeDef * pUTILS_ICInitStruct, const LL_UTILS_ClkInitTypeDef * pUTILS_ClkInitStruct)

Function description

This function configures the CPU system clock with HSI as clock source of the PLL1 used in integer mode.

Parameters

- **pUTILS_PLLInitStruct:** pointer to a LL_UTILS_PLLInitTypeDef structure that contains the configuration information for the PLL.
- **pUTILS_ICInitStruct:** pointer to a LL_UTILS_ICInitTypeDef structure that contains the configuration information for the IC.
- **pUTILS_ClkInitStruct:** pointer to a LL_UTILS_ClkInitTypeDef structure that contains the configuration information for the BUS prescalers.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: Max frequency configuration done
  - ERROR: Max frequency configuration not done

Notes

- The application needs to ensure that PLL1 is disabled.
- Function is based on the following formula: PLL output frequency = (((HSI frequency / PLLM) * PLLN) / PLLP1 / PLLP2)PLLM: ensure that the VCO input frequency ranges from 1 to 50 MHz (PLLVCO_input = HSI frequency / PLLM)PLLN: ensure that the VCO output frequency is maximum 3200 MHz (PLLVCO_output = PLLVCO_input * PLLN)PLLP1, PLLP2: ensure that max frequency at 800 MHz is reached (PLLVCO_output / PLLP1 / PLLP2)

LL_PLL_ConfigSystemClock_MSI

Function name

ErrorStatus LL_PLL_ConfigSystemClock_MSI (const LL_UTILS_PLLInitTypeDef * pUTILS_PLLInitStruct, const LL_UTILS_ICInitTypeDef * pUTILS_ICInitStruct, const LL_UTILS_ClkInitTypeDef * pUTILS_ClkInitStruct)

Function description

This function configures the CPU system clock with MSI as clock source of the PLL1 used in integer mode.

Parameters

- **pUTILS_PLLInitStruct:** pointer to a LL_UTILS_PLLInitTypeDef structure that contains the configuration information for the PLL.
- **pUTILS_ICInitStruct:** pointer to a LL_UTILS_ICInitTypeDef structure that contains the configuration information for the IC.
- **pUTILS_ClkInitStruct:** pointer to a LL_UTILS_ClkInitTypeDef structure that contains the configuration information for the BUS prescalers.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: Max frequency configuration done
  - ERROR: Max frequency configuration not done

Notes

- The application needs to ensure that PLL1 is disabled.
- Function is based on the following formula: PLL output frequency = (((MSI frequency / PLLM) * PLLN) / PLLP1 / PLLP2)PLLM: ensure that the VCO input frequency ranges from 1 to 50 MHz (PLLVCO_input = MSI frequency / PLLM)PLLN: ensure that the VCO output frequency is maximum 3200 MHz (PLLVCO_output = PLLVCO_input * PLLN)PLLP1, PLLP2: ensure that max frequency at 800 MHz is reached (PLLVCO_output / PLLP1 / PLLP2)

LL_PLL_ConfigSystemClock_HSE

Function name

ErrorStatus LL_PLL_ConfigSystemClock_HSE (uint32_t HSEFrequency, uint32_t HSEBypass, const LL_UTILS_PLLInitTypeDef * pUTILS_PLLInitStruct, const LL_UTILS_ICInitTypeDef * pUTILS_ICInitStruct, const LL_UTILS_ClkInitTypeDef * pUTILS_ClkInitStruct)

Function description

This function configures the CPU system clock with HSE as clock source of the PLL1 used in integer mode.

Parameters

- **HSEFrequency:** Value between Min_Data = 4000000 and Max_Data = 50000000
- **HSEBypass:** This parameter can be one of the following values:
  - LL_UTILS_HSEBYPASS_ON
  - LL_UTILS_HSEBYPASS_OFF
- **pUTILS_PLLInitStruct:** pointer to a LL_UTILS_PLLInitTypeDef structure that contains the configuration information for the PLL.
- **pUTILS_ICInitStruct:** pointer to a LL_UTILS_ICInitTypeDef structure that contains the configuration information for the IC.
- **pUTILS_ClkInitStruct:** pointer to a LL_UTILS_ClkInitTypeDef structure that contains the configuration information for the BUS prescalers.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: Max frequency configuration done
  - ERROR: Max frequency configuration not done

Notes

- The application needs to ensure that PLL1 is disabled.
- Function is based on the following formula: PLL output frequency = (((HSE frequency / PLLM) * PLLN) / PLLP1 / PLLP2)PLLM: ensure that the VCO input frequency ranges from 1 to 50 MHz (PLLVCO_input = HSE frequency / PLLM)PLLN: ensure that the VCO output frequency is maximum 3200 MHz (PLLVCO_output = PLLVCO_input * PLLN)PLLP1, PLLP2: ensure that max frequency at 800 MHz is reached (PLLVCO_output / PLLP1 / PLLP2)

109.3 UTILS Firmware driver defines

The following section lists the various define and macros of the module.

109.3.1 UTILS

UTILS

DEVICE RPN

LL_UTILS_DEVICE_STM32N657

STM32N657

LL_UTILS_DEVICE_STM32N647

STM32N647

LL_UTILS_DEVICE_STM32N655

STM32N655

LL_UTILS_DEVICE_STM32N645

STM32N645

HSE Bypass activation

LL_UTILS_HSEBYPASS_OFF

HSE Bypass is not enabled

LL_UTILS_HSEBYPASS_ON

HSE Bypass is enabled

PACKAGE TYPE

LL_UTILS_PACKAGETYPE_BGA142

BGA142 package type

LL_UTILS_PACKAGETYPE_BGA169

BGA169 package type

LL_UTILS_PACKAGETYPE_BGA178

BGA178 package type

LL_UTILS_PACKAGETYPE_BGA198

BGA198 package type

LL_UTILS_PACKAGETYPE_BGA223

BGA223 package type

LL_UTILS_PACKAGETYPE_BGA264

BGA264 package type

110 LL VENC Generic Driver

110.1 VENC Firmware driver API description

The following section lists the various functions of the VENC library.

110.1.1 Detailed description of functions

LL_VENC_Init

Function name

void LL_VENC_Init (void)

Function description

Initialize the VENC hardware.

Return values

- **None:**

Notes

- to check that the initialization is correct, ASIC ID should be checked

LL_VENC_DeInit

Function name

void LL_VENC_DeInit (void)

Function description

De-Initialize the VENC hardware.

Return values

- **None:**

110.2 VENC Firmware driver defines

The following section lists the various define and macros of the module.

110.2.1 VENC

VENC

111 LL WWDG Generic Driver

111.1 WWDG Firmware driver API description

The following section lists the various functions of the WWDG library.

111.1.1 Detailed description of functions

LL_WWDG_Enable

Function name

__STATIC_INLINE void LL_WWDG_Enable (WWDG_TypeDef * WWDGx)

Function description

Enable Window Watchdog.

Parameters

- **WWDGx:** WWDG Instance

Return values

- **None:**

Notes

- It is enabled by setting the WDGA bit in the WWDG_CR register, then it cannot be disabled again except by a reset. This bit is set by software and only cleared by hardware after a reset. When WDGA = 1, the watchdog can generate a reset.

Reference Manual to LL API cross reference:

- CR WDGA LL_WWDG_Enable

LL_WWDG_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_WWDG_IsEnabled (const WWDG_TypeDef * WWDGx)

Function description

Checks if Window Watchdog is enabled.

Parameters

- **WWDGx:** WWDG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR WDGA LL_WWDG_IsEnabled

LL_WWDG_SetCounter

Function name

__STATIC_INLINE void LL_WWDG_SetCounter (WWDG_TypeDef * WWDGx, uint32_t Counter)

Function description

Set the Watchdog counter value to provided value (7-bits T[6:0])

Parameters

- **WWDGx:** WWDG Instance
- **Counter:** 0..0x7F (7 bit counter value)

Return values

- **None:**

Notes

- When writing to the WWDG_CR register, always write 1 in the MSB b6 to avoid generating an immediate reset This counter is decremented every (4096 x 2expWDGTB) PCLK cycles A reset is produced when it rolls over from 0x40 to 0x3F (bit T6 becomes cleared) Setting the counter lower then 0x40 causes an immediate reset (if WWDG enabled)

Reference Manual to LL API cross reference:

- CR T LL_WWDG_SetCounter

LL_WWDG_GetCounter

Function name

__STATIC_INLINE uint32_t LL_WWDG_GetCounter (const WWDG_TypeDef * WWDGx)

Function description

Return current Watchdog Counter Value (7 bits counter value)

Parameters

- **WWDGx:** WWDG Instance

Return values

- **7:** bit Watchdog Counter value

Reference Manual to LL API cross reference:

- CR T LL_WWDG_GetCounter

LL_WWDG_SetPrescaler

Function name

__STATIC_INLINE void LL_WWDG_SetPrescaler (WWDG_TypeDef * WWDGx, uint32_t Prescaler)

Function description

Set the time base of the prescaler (WDGTB).

Parameters

- **WWDGx:** WWDG Instance
- **Prescaler:** This parameter can be one of the following values:
  - LL_WWDG_PRESCALER_1
  - LL_WWDG_PRESCALER_2
  - LL_WWDG_PRESCALER_4
  - LL_WWDG_PRESCALER_8
  - LL_WWDG_PRESCALER_16
  - LL_WWDG_PRESCALER_32
  - LL_WWDG_PRESCALER_64
  - LL_WWDG_PRESCALER_128

Return values

- **None:**

Notes

- Prescaler is used to apply ratio on PCLK clock, so that Watchdog counter is decremented every (4096 x 2expWDGTB) PCLK cycles

Reference Manual to LL API cross reference:

- CFR WDGTB LL_WWDG_SetPrescaler

LL_WWDG_GetPrescaler

Function name

__STATIC_INLINE uint32_t LL_WWDG_GetPrescaler (const WWDG_TypeDef * WWDGx)

Function description

Return current Watchdog Prescaler Value.

Parameters

- **WWDGx:** WWDG Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_WWDG_PRESCALER_1
  - LL_WWDG_PRESCALER_2
  - LL_WWDG_PRESCALER_4
  - LL_WWDG_PRESCALER_8
  - LL_WWDG_PRESCALER_16
  - LL_WWDG_PRESCALER_32
  - LL_WWDG_PRESCALER_64
  - LL_WWDG_PRESCALER_128

Reference Manual to LL API cross reference:

- CFR WDGTB LL_WWDG_GetPrescaler

LL_WWDG_SetWindow

Function name

__STATIC_INLINE void LL_WWDG_SetWindow (WWDG_TypeDef * WWDGx, uint32_t Window)

Function description

Set the Watchdog Window value to be compared to the downcounter (7-bits W[6:0]).

Parameters

- **WWDGx:** WWDG Instance
- **Window:** 0x00..0x7F (7 bit Window value)

Return values

- **None:**

Notes

- This window value defines when write in the WWDG_CR register to program Watchdog counter is allowed. Watchdog counter value update must occur only when the counter value is lower than the Watchdog window register value. Otherwise, a MCU reset is generated if the 7-bit Watchdog counter value (in the control register) is refreshed before the downcounter has reached the watchdog window register value. Physically is possible to set the Window lower then 0x40 but it is not recommended. To generate an immediate reset, it is possible to set the Counter lower than 0x40.

Reference Manual to LL API cross reference:

- CFR W LL_WWDG_SetWindow

LL_WWDG_GetWindow

Function name

__STATIC_INLINE uint32_t LL_WWDG_GetWindow (const WWDG_TypeDef * WWDGx)

Function description

Return current Watchdog Window Value (7 bits value)

Parameters

- **WWDGx:** WWDG Instance

Return values

- **7:** bit Watchdog Window value

Reference Manual to LL API cross reference:

- CFR W LL_WWDG_GetWindow

LL_WWDG_IsActiveFlag_EWKUP

Function name

__STATIC_INLINE uint32_t LL_WWDG_IsActiveFlag_EWKUP (const WWDG_TypeDef * WWDGx)

Function description

Indicates if the WWDG Early Wakeup Interrupt Flag is set or not.

Parameters

- **WWDGx:** WWDG Instance

Return values

- **State:** of bit (1 or 0).

Notes

- This bit is set by hardware when the counter has reached the value 0x40. It must be cleared by software by writing 0. A write of 1 has no effect. This bit is also set if the interrupt is not enabled.

Reference Manual to LL API cross reference:

- SR EWIF LL_WWDG_IsActiveFlag_EWKUP

LL_WWDG_ClearFlag_EWKUP

Function name

__STATIC_INLINE void LL_WWDG_ClearFlag_EWKUP (WWDG_TypeDef * WWDGx)

Function description

Clear WWDG Early Wakeup Interrupt Flag (EWIF)

Parameters

- **WWDGx:** WWDG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR EWIF LL_WWDG_ClearFlag_EWKUP

LL_WWDG_EnableIT_EWKUP

Function name

__STATIC_INLINE void LL_WWDG_EnableIT_EWKUP (WWDG_TypeDef * WWDGx)

Function description

Enable the Early Wakeup Interrupt.

Parameters

- **WWDGx:** WWDG Instance

Return values

- **None:**

Notes

- When set, an interrupt occurs whenever the counter reaches value 0x40. This interrupt is only cleared by hardware after a reset

Reference Manual to LL API cross reference:

- CFR EWI LL_WWDG_EnableIT_EWKUP

LL_WWDG_IsEnabledIT_EWKUP

Function name

__STATIC_INLINE uint32_t LL_WWDG_IsEnabledIT_EWKUP (const WWDG_TypeDef * WWDGx)

Function description

Check if Early Wakeup Interrupt is enabled.

Parameters

- **WWDGx:** WWDG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CFR EWI LL_WWDG_IsEnabledIT_EWKUP

111.2 WWDG Firmware driver defines

The following section lists the various define and macros of the module.

111.2.1 WWDG

WWDG

IT Defines

LL_WWDG_CFR_EWI

PRESCALER

LL_WWDG_PRESCALER_1

WWDG counter clock = (PCLK1/4096)/1

LL_WWDG_PRESCALER_2

WWDG counter clock = (PCLK1/4096)/2

LL_WWDG_PRESCALER_4

WWDG counter clock = (PCLK1/4096)/4

LL_WWDG_PRESCALER_8

WWDG counter clock = (PCLK1/4096)/8

LL_WWDG_PRESCALER_16

WWDG counter clock = (PCLK1/4096)/16

LL_WWDG_PRESCALER_32

WWDG counter clock = (PCLK1/4096)/32

LL_WWDG_PRESCALER_64

WWDG counter clock = (PCLK1/4096)/64

LL_WWDG_PRESCALER_128

WWDG counter clock = (PCLK1/4096)/128

Common Write and read registers macros

LL_WWDG_WriteReg

Description:

- Write a value in WWDG register.

Parameters:

- __INSTANCE__: WWDG Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_WWDG_ReadReg

Description:

- Read a value in WWDG register.

Parameters:

- __INSTANCE__: WWDG Instance
- __REG__: Register to be read

Return value:

- Register: value

112 FAQs

General subjects

Why should I use the HAL drivers?

There are many advantages in using the HAL drivers:

- Ease of use: you can use the HAL drivers to configure and control any peripheral embedded within your STM32 MCU without prior in-depth knowledge of the product.
- HAL drivers provide intuitive and ready-to-use APIs to configure the peripherals and support polling, interrupt and DMA programming model to accommodate all application requirements, thus allowing the end-user to build a complete application by calling a few APIs.
- Higher level of abstraction than a standard peripheral library allowing to transparently manage:
  - Data transfers and processing using blocking mode (polling) or non-blocking mode (interrupt or DMA)
  - Error management through peripheral error detection and timeout mechanism.
- Generic architecture speeding up initialization and porting, thus allowing customers to focus on innovation.
- Generic set of APIs with full compatibility across the STM32 Series/lines, to ease the porting task between STM32 MCUs.
- The APIs provided within the HAL drivers are feature-oriented and do not require in-depth knowledge of peripheral operation.
- The APIs provided are modular. They include initialization, IO operation and control functions. The end-user has to call init function, then start the process by calling one IO operation functions (write, read, transmit, receive, …). Most of the peripherals have the same architecture.
- The number of functions required to build a complete and useful application is very reduced. As an example, to build a UART communication process, the user only has to call HAL_UART_Init() then HAL_UART_Transmit() or HAL_UART_Receive().

Which devices are supported by the HAL drivers?

The HAL drivers are developed to support all STM32N6 devices. To ensure compatibility between all devices and portability with others Series and lines, the API is split into the generic and the extension APIs . For more details, please refer to *section* ***Devices supported by the HAL drivers***.

What is the cost of using HAL drivers in term of code size and performance?

Like generic architecture drivers, the HAL drivers may induce firmware overhead.

This is due to the high abstraction level and ready-to-use APIs which allow data transfers, errors management and offloads the user application from implementation details.

Architecture

How many files should I modify to configure the HAL drivers?

Only one file needs to be modified: stm32n6xxxx_hal_conf.h. You can modify this file by disabling unused modules, or adjusting some parameters (i.e. HSE value, System configuration…)

A template is provided in the HAL drivers folders (stm32n6xxxx_hal_conf_template.c).

Which header files should I include in my application to use the HAL drivers?

Only stm32n6xxxx_hal.h file has to be included.

What is the difference between xx_hal_ppp.c/.h and xx_hal_ppp_ *ex* .c/.h?

The HAL driver architecture supports common features across STM32 Series/lines. To support specific features, the drivers are split into two groups.

- The generic APIs (stm32n6xxxx_hal_ppp.c): It includes the common set of APIs across all the STM32 product lines
- The extension APIs (stm32n6xxxx_hal_ppp_ex.c): It includes the specific APIs for specific device part number or family.

Initialization and I/O operation functions

How do I configure the system clock?

Unlike the standard library, the system clock configuration is not performed in CMSIS drivers file (system_stm32n6xxxx.c) but in the main user application by calling the two main functions, HAL_RCC_OscConfig() and HAL_RCC_ClockConfig(). It can be modified in any user application section.

What is the purpose of the *PPP_HandleTypeDef *pHandle* structure located in each driver in addition to the Initialization structure

PPP_HandleTypeDef *pHandle is the main structure implemented in the HAL drivers. It handles the peripheral configuration and registers, and embeds all the structures and variables required to follow the peripheral device flow (pointer to buffer, Error code, State,...)

However, this structure is not required to service peripherals such as GPIO, SYSTICK, PWR, and RCC.

What is the purpose of HAL_PPP_MspInit() and HAL_PPP_MspDeInit() functions?

These function are called within HAL_PPP_Init() and HAL_PPP_DeInit(), respectively. They are used to perform the low level Initialization/de-initialization related to the additional hardware resources (RCC, GPIO, NVIC and DMA).

These functions are declared in stm32n6xxxx_hal_msp.c. A template is provided in the HAL driver folders (stm32n6xxxx_hal_msp_template.c).

When and how should I use callbacks functions (functions declared with the attribute *__weak*)?

Use callback functions for the I/O operations used in DMA or interrupt mode. The PPP process complete callbacks are called to inform the user about process completion in real-time event mode (interrupts).

The Errors callbacks are called when a processing error occurs in DMA or interrupt mode. These callbacks are customized by the user to add user proprietary code. They can be declared in the application. Note that the same process completion callbacks are used for DMA and interrupt mode.

Is it mandatory to use HAL_Init() function at the beginning of the user application?

It is mandatory to use HAL_Init() function to enable the system configuration (Prefetch, Data instruction cache,…), configure the systTick and the NVIC priority grouping and the hardware low level initialization.

The SysTick configuration shall be adjusted by calling ***HAL_RCC_ClockConfig()*** function, to obtain 1 ms whatever the system clock.

Why do I need to configure the SysTick timer to use the HAL drivers?

The SysTick timer is configured to be used to generate variable increments by calling *HAL_IncTick()* function in SysTick ISR and retrieve the value of this variable by calling *HAL_GetTick()* function.

The call HAL_GetTick() function is mandatory when using HAL drivers with Polling Process or when using HAL_Delay().

Why is the SysTick timer configured to have 1 ms?

This is mandatory to ensure correct IO operation in particular for polling mode operation where the 1 ms is required as timebase.

Could HAL_Delay() function block my application under certain conditions?

Care must be taken when using HAL_Delay() since this function provides accurate delay based on a variable incremented in SysTick ISR. This implies that if HAL_Delay() is called from a peripheral ISR process, then the SysTick interrupt must have higher priority (numerically lower) than the peripheral interrupt, otherwise the caller ISR process will be blocked. Use HAL_NVIC_SetPriority() function to change the SysTick interrupt priority.

What programming model sequence should I follow to use HAL drivers ?

Follow the sequence below to use the APIs provided in the HAL drivers:

1. Call HAL_Init() function to initialize the system (data cache, NVIC priority,…).

2. Initialize the system clock by calling HAL_RCC_OscConfig() followed by HAL_RCC_ClockConfig().
3. Add HAL_IncTick() function under SysTick_Handler() ISR function to enable polling process when using HAL_Delay() function
4. Start initializing your peripheral by calling HAL_PPP_Init().
5. Implement the hardware low level initialization (Peripheral clock, GPIO, DMA,..) by calling HAL_PPP_MspInit() in stm32n6xxxx_hal_msp.c
6. Start your process operation by calling IO operation functions.

What is the purpose of HAL_PPP_IRQHandler() function and when should I use it?

HAL_PPP_IRQHandler() is used to handle interrupt process. It is called under PPP_IRQHandler() function in stm32n6xxxx_it.c. In this case, the end-user has to implement only the callbacks functions (prefixed by __weak) to perform the appropriate action when an interrupt is detected. Advanced users can implement their own code in PPP_IRQHandler() without calling HAL_PPP_IRQHandler().

Can I use directly the macros defined in xx_hal_ppp.h ?

Yes, you can: a set of macros is provided with the APIs. They allow accessing directly some specific features using peripheral flags.

Where must PPP_HandleTypedef structure peripheral handler be declared?

PPP_HandleTypedef structure peripheral handler must be declared as a global variable, so that all the structure fields are set to 0 by default. In this way, the peripheral handler default state are set to HAL_PPP_STATE_RESET, which is the default state for each peripheral after a system reset.

When should I use HAL versus LL drivers?

HAL drivers offer high-level and function-oriented APIs, with a high level of portability. Product/IPs complexity is hidden for end users. LL drivers offer low-level APIs at registers level, with a better optimization but less portability. They require a deep knowledge of product/IPs specifications.

How can I include LL drivers in my environment? Is there any LL configuration file as for HAL?

There is no configuration file. Source code shall directly include the necessary stm32n6xxxx_ll_ppp.h file(s).

Can I use HAL and LL drivers together? If yes, what are the constraints?

It is possible to use both HAL and LL drivers. One can handle the IP initialization phase with HAL and then manage the I/O operations with LL drivers. The major difference between HAL and LL is that HAL drivers require to create and use handles for operation management while LL drivers operates directly on peripheral registers. Mixing HAL and LL is illustrated in Examples_MIX example.

Is there any LL APIs which are not available with HAL?

Yes, there are. A few Cortex® APIs have been added in stm32n6xxxx_ll_cortex.h e.g. for accessing SCB or SysTick registers.

Why are SysTick interrupts not enabled on LL drivers?

When using LL drivers in standalone mode, you do not need to enable SysTick interrupts because they are not used in LL APIs, while HAL functions requires SysTick interrupts to manage timeouts.

Revision history

Table 25. **Document revision history**

Date	Revision	Changes
25-Nov-2024	1	Initial release.
18-Mar-2025	2	Added Section 65: HAL SDIO Generic Driver.

Contents

1 General information . . . 3
2 Acronyms and definitions . . . 4
3 Overview of HAL drivers . . . 7
3.1 HAL and user-application files . . . 8
3.1.1 HAL driver files . . . 8
3.1.2 User-application files . . . 8
3.2 HAL data structures . . . 10
3.2.1 Peripheral handle structures . . . 10
3.2.2 Initialization and configuration structure . . . 12
3.2.3 Specific process structures . . . 12
3.3 API classification . . . 13
3.4 Devices supported by HAL drivers . . . 14
3.5 HAL driver rules . . . 17
3.5.1 HAL API naming rules . . . 17
3.5.2 HAL general naming rules . . . 18
3.5.3 HAL interrupt handler and callback functions . . . 19
3.6 HAL generic APIs . . . 19
3.7 HAL extension APIs . . . 20
3.7.1 HAL extension model overview . . . 20
3.7.2 HAL extension model cases . . . 21
3.8 File inclusion model . . . 22
3.9 HAL common resources . . . 23
3.10 HAL configuration . . . 23
3.11 HAL system peripheral handling . . . 24
3.11.1 Clocks . . . 24
3.11.2 GPIOs . . . 24
3.11.3 Cortex® NVIC and SysTick timer . . . 25
3.11.4 PWR . . . 26
3.11.5 EXTI . . . 26
3.11.6 DMA . . . 27
3.12 How to use HAL drivers . . . 30
3.12.1 HAL usage models . . . 30
3.12.2 HAL initialization . . . 31
3.12.3 HAL I/O operation process . . . 34
3.12.4 Timeout and error management . . . 37

4 Overview of low-layer drivers 40
4.1 Low-layer files 40
4.2 Overview of low-layer APIs and naming rules 42
4.2.1 Peripheral initialization functions 42
4.2.2 Peripheral register-level configuration functions 43
5 Cohabiting of HAL and LL 45
5.1 Low-layer driver used in standalone mode 45
5.2 Mixed use of low-layer APIs and HAL drivers 45
6 HAL System Driver 46
6.1 HAL Firmware driver API description 46
6.1.1 How to use this driver 46
6.1.2 Initialization and de-initialization functions 46
6.1.3 HAL Control functions 46
6.1.4 HAL Debug functions 47
6.1.5 HAL SYSCFG configuration functions 47
6.1.6 Detailed description of functions 50
6.2 HAL Firmware driver defines 79
6.2.1 HAL 79
7 HAL ADC Generic Driver 91
7.1 ADC Firmware driver registers structures 91
7.1.1 ADC_OversamplingTypeDef 91
7.1.2 ADC_InitTypeDef 91
7.1.3 ADC_ChannelConfTypeDef 93
7.1.4 ADC_AnalogWDGConfTypeDef 94
7.1.5 __ADC_HandleTypeDef 95
7.2 ADC Firmware driver API description 96
7.2.1 ADC peripheral features 96
7.2.2 How to use this driver 97
7.2.3 Detailed description of functions 99
7.3 ADC Firmware driver defines 104
7.3.1 ADC 104
8 HAL ADC Extension Driver 130
8.1 ADCEx Firmware driver registers structures 130
8.1.1 ADC_InjOversamplingTypeDef 130
8.1.2 ADC_InjectionConfTypeDef 130
8.1.3 ADC_MultiModeTypeDef 132
8.2 ADCEx Firmware driver API description 132

8.2.1 Detailed description of functions . . . 132
8.3 ADCEx Firmware driver defines . . . 136
8.3.1 ADCEx . . . 136
9 HAL BSEC Generic Driver . . . 140
9.1 BSEC Firmware driver registers structures . . . 140
9.1.1 BSEC_HandleTypeDef . . . 140
9.1.2 BSEC_ScratchRegTypeDef . . . 140
9.1.3 BSEC_DebugCfgTypeDef . . . 140
9.2 BSEC Firmware driver API description . . . 140
9.2.1 How to use this driver . . . 140
9.2.2 Detailed description of functions . . . 142
9.3 BSEC Firmware driver defines . . . 146
9.3.1 BSEC . . . 146
10 HAL CACHEAXI Generic Driver . . . 151
10.1 CACHEAXI Firmware driver registers structures . . . 151
10.1.1 __CACHEAXI_HandleTypeDef . . . 151
10.2 CACHEAXI Firmware driver API description . . . 151
10.2.1 How to use this driver . . . 151
10.2.2 Detailed description of functions . . . 152
10.3 CACHEAXI Firmware driver defines . . . 156
10.3.1 CACHEAXI . . . 157
11 HAL CORTEX Generic Driver . . . 161
11.1 CORTEX Firmware driver registers structures . . . 161
11.1.1 MPU_Region_InitTypeDef . . . 161
11.1.2 MPU_Attributes_InitTypeDef . . . 161
11.2 CORTEX Firmware driver API description . . . 162
11.2.1 How to use this driver . . . 162
11.2.2 Initialization and Configuration functions . . . 163
11.2.3 Peripheral Control functions . . . 163
11.2.4 Detailed description of functions . . . 164
11.3 CORTEX Firmware driver defines . . . 170
11.3.1 CORTEX . . . 170
12 HAL CRC Generic Driver . . . 174
12.1 CRC Firmware driver registers structures . . . 174
12.1.1 CRC_InitTypeDef . . . 174
12.1.2 CRC_HandleTypeDef . . . 175
12.2 CRC Firmware driver API description . . . 175

12.2.1 How to use this driver . . . 175
12.2.2 Detailed description of functions . . . 175
12.3 CRC Firmware driver defines . . . 176
12.3.1 CRC . . . 176
13 HAL CRC Extension Driver . . . 180
13.1 CRCEx Firmware driver API description . . . 180
13.1.1 How to use this driver . . . 180
13.1.2 Detailed description of functions . . . 180
13.2 CRCEx Firmware driver defines . . . 180
13.2.1 CRCEx . . . 180
14 HAL CRYP Generic Driver . . . 183
14.1 CRYP Firmware driver registers structures . . . 183
14.1.1 CRYP_ConfigTypeDef . . . 183
14.1.2 __CRYP_HandleTypeDef . . . 184
14.2 CRYP Firmware driver defines . . . 185
14.2.1 CRYP . . . 185
15 HAL CRYP Extension Driver . . . 193
15.1 CRYPEx Firmware driver API description . . . 193
15.1.1 How to use this driver . . . 193
15.1.2 Detailed description of functions . . . 193
16 HAL DCMIPP Generic Driver . . . 195
16.1 DCMIPP Firmware driver registers structures . . . 195
16.1.1 DCMIPP_EmbeddedSyncUnmaskTypeDef . . . 195
16.1.2 DCMIPP_EmbeddedSyncCodesTypeDef . . . 195
16.1.3 DCMIPP_CSI_PIPE_ConfTypeDef . . . 195
16.1.4 DCMIPP_CSI_ConfTypeDef . . . 196
16.1.5 DCMIPP_CSI_VCFilteringConfTypeDef . . . 196
16.1.6 DCMIPP_CSI_TimerConfTypeDef . . . 196
16.1.7 DCMIPP_CSI_LineByteCounterConfTypeDef . . . 196
16.1.8 DCMIPP_ParallelConfTypeDef . . . 197
16.1.9 DCMIPP_PipeConfTypeDef . . . 197
16.1.10 DCMIPP_IPPlugConfTypeDef . . . 198
16.1.11 DCMIPP_CropConfTypeDef . . . 198
16.1.12 DCMIPP_StatisticExtractionAreaConfTypeDef . . . 199
16.1.13 DCMIPP_StatisticExtractionConfTypeDef . . . 199
16.1.14 DCMIPP_ExposureConfTypeDef . . . 199
16.1.15 DCMIPP_ContrastConfTypeDef . . . 200

16.1.16 DCMIPP_RawBayer2RGBConfTypeDef . . . 200
16.1.17 DCMIPP_ColorConversionConfTypeDef . . . 201
16.1.18 DCMIPP_RegionOfInterestConfTypeDef . . . 202
16.1.19 DCMIPP_BlackLevelConfTypeDef . . . 202
16.1.20 DCMIPP_DownsizeTypeDef . . . 203
16.1.21 DCMIPP_DecimationConfTypeDef . . . 203
16.1.22 DCMIPP_SemiPlanarDstAddressTypeDef . . . 203
16.1.23 DCMIPP_FullPlanarDstAddressTypeDef . . . 204
16.1.24 __DCMIPP_HandleTypeDef . . . 204
16.2 DCMIPP Firmware driver API description . . . 205
16.2.1 How to use this driver . . . 205
16.2.2 Detailed description of functions . . . 207
16.3 DCMIPP Firmware driver defines . . . 229
16.3.1 DCMIPP . . . 230
17 HAL DCMI Generic Driver . . . 265
17.1 DCMI Firmware driver registers structures . . . 265
17.1.1 DCMI_CodesInitTypeDef . . . 265
17.1.2 DCMI_SyncUnmaskTypeDef . . . 265
17.1.3 DCMI_InitTypeDef . . . 265
17.1.4 __DCMI_HandleTypeDef . . . 266
17.2 DCMI Firmware driver API description . . . 267
17.2.1 How to use this driver . . . 267
17.2.2 Detailed description of functions . . . 268
17.3 DCMI Firmware driver defines . . . 271
17.3.1 DCMI . . . 271
18 HAL DMA2D Generic Driver . . . 278
18.1 DMA2D Firmware driver registers structures . . . 278
18.1.1 DMA2D_CLUTCfgTypeDef . . . 278
18.1.2 DMA2D_InitTypeDef . . . 278
18.1.3 DMA2D_LayerCfgTypeDef . . . 279
18.1.4 __DMA2D_HandleTypeDef . . . 279
18.2 DMA2D Firmware driver API description . . . 280
18.2.1 How to use this driver . . . 280
18.2.2 Detailed description of functions . . . 282
18.3 DMA2D Firmware driver defines . . . 286
18.3.1 DMA2D . . . 286
19 HAL DMA Generic Driver . . . 294

19.1 DMA Firmware driver registers structures . . . 294
19.1.1 DMA_InitTypeDef . . . 294
19.1.2 DMA_InitLinkedListTypeDef . . . 295
19.1.3 DMA_IsolationConfigTypeDef . . . 295
19.1.4 __DMA_HandleTypeDef . . . 295
19.2 DMA Firmware driver API description . . . 296
19.2.1 How to use this driver . . . 296
19.2.2 Initialization and de-initialization functions . . . 299
19.2.3 IO operation functions . . . 299
19.2.4 State and Errors functions . . . 299
19.2.5 DMA Attributes functions . . . 300
19.2.6 Detailed description of functions . . . 300
19.3 DMA Firmware driver defines . . . 305
19.3.1 DMA . . . 305
20 HAL DMA Extension Driver . . . 329
20.1 DMAEx Firmware driver registers structures . . . 329
20.1.1 DMA_DataHandlingConfTypeDef . . . 329
20.1.2 DMA_TriggerConfTypeDef . . . 329
20.1.3 DMA_RepeatBlockConfTypeDef . . . 329
20.1.4 DMA_NodeConfTypeDef . . . 330
20.1.5 DMA_NodeTypeDef . . . 331
20.1.6 __DMA_QListTypeDef . . . 331
20.1.7 DMA_NodeInQInfoTypeDef . . . 331
20.2 DMAEx Firmware driver API description . . . 331
20.2.1 How to use this driver . . . 332
20.2.2 Linked-List Initialization and De-Initialization Functions . . . 339
20.2.3 Linked-List IO Operation Functions . . . 339
20.2.4 Linked-List Management Functions . . . 339
20.2.5 Data handling, repeated block and trigger configuration functions . . . 341
20.2.6 Suspend and resume operation functions . . . 341
20.2.7 Fifo status function . . . 341
20.2.8 Detailed description of functions . . . 342
20.3 DMAEx Firmware driver defines . . . 351
20.3.1 DMAEx . . . 351
21 HAL DTS Generic Driver . . . 364
21.1 DTS Firmware driver registers structures . . . 364
21.1.1 __DTS_HandleTypeDef . . . 364

21.1.2 DTS_SensorConfigTypeDef . . . 364
21.1.3 DTS_AlarmConfigTypeDef . . . 365
21.2 DTS Firmware driver API description . . . 365
21.2.1 How to use this driver . . . 365
21.2.2 Detailed description of functions . . . 367
21.3 DTS Firmware driver defines . . . 370
21.3.1 DTS . . . 370
22 HAL ETH Generic Driver . . . 373
22.1 ETH Firmware driver registers structures . . . 373
22.1.1 ETH_DMADescTypeDef . . . 373
22.1.2 __ETH_BufferTypeDef . . . 373
22.1.3 ETH_TxDescListTypeDef . . . 373
22.1.4 ETH_TxPacketConfigTypeDef . . . 374
22.1.5 ETH_TimeStampTypeDef . . . 375
22.1.6 ETH_RxDescListTypeDef . . . 375
22.1.7 ETH_MACConfigTypeDef . . . 375
22.1.8 DMAChannelInstanceTypeDef . . . 378
22.1.9 ETH_DMAConfigTypeDef . . . 378
22.1.10 ETH_InitTypeDef . . . 379
22.1.11 __ETH_HandleTypeDef . . . 379
22.1.12 ETH_MACFilterConfigTypeDef . . . 381
22.1.13 ETH_PowerDownConfigTypeDef . . . 382
22.2 ETH Firmware driver API description . . . 382
22.2.1 How to use this driver . . . 382
22.2.2 Detailed description of functions . . . 383
22.3 ETH Firmware driver defines . . . 390
22.3.1 ETH . . . 390
23 HAL ETH Extension Driver . . . 415
23.1 ETHEx Firmware driver registers structures . . . 415
23.1.1 ETH_RxVLANConfigTypeDef . . . 415
23.1.2 ETH_MACMTLMappingTypeDef . . . 416
23.1.3 MTL_TxQueueInstance_t . . . 417
23.1.4 MTL_RxQueueInstance_t . . . 417
23.1.5 ETH_MTLConfigTypeDef . . . 418
23.1.6 ETH_PacketTypeQueueConfigTypeDef . . . 418
23.1.7 ETH_TxVLANConfigTypeDef . . . 418
23.1.8 ETH_L3FilterConfigTypeDef . . . 419

23.1.9 ETH_L4FilterConfigTypeDef . . . 419
23.2 ETHEx Firmware driver API description . . . 420
23.2.1 Detailed description of functions . . . 420
23.3 ETHEx Firmware driver defines . . . 424
23.3.1 ETHEx . . . 424
24 HAL EXTI Generic Driver . . . 428
24.1 EXTI Firmware driver registers structures . . . 428
24.1.1 EXTI_HandleTypeDef . . . 428
24.1.2 EXTI_ConfigTypeDef . . . 428
24.2 EXTI Firmware driver API description . . . 428
24.2.1 EXTI Peripheral features . . . 428
24.2.2 How to use this driver . . . 429
24.2.3 EXTI attributes functions . . . 429
24.2.4 Configuration functions . . . 429
24.2.5 Detailed description of functions . . . 429
24.3 EXTI Firmware driver defines . . . 433
24.3.1 EXTI . . . 433
25 HAL FDCAN Generic Driver . . . 438
25.1 FDCAN Firmware driver registers structures . . . 438
25.1.1 FDCAN_InitTypeDef . . . 438
25.1.2 FDCAN_ClkCalUnitTypeDef . . . 439
25.1.3 FDCAN_FilterTypeDef . . . 440
25.1.4 FDCAN_TxHeaderTypeDef . . . 441
25.1.5 FDCAN_RxHeaderTypeDef . . . 442
25.1.6 FDCAN_TxEventFifoTypeDef . . . 442
25.1.7 FDCAN_HpMsgStatusTypeDef . . . 443
25.1.8 FDCAN_ProtocolStatusTypeDef . . . 444
25.1.9 FDCAN_ErrorCountersTypeDef . . . 445
25.1.10 FDCAN_TT_ConfigTypeDef . . . 445
25.1.11 FDCAN_TriggerTypeDef . . . 447
25.1.12 FDCAN_TTOperationStatusTypeDef . . . 447
25.1.13 FDCAN_MsgRamAddressTypeDef . . . 449
25.1.14 __FDCAN_HandleTypeDef . . . 449
25.2 FDCAN Firmware driver API description . . . 451
25.2.1 How to use this driver . . . 451
25.2.2 Detailed description of functions . . . 453
25.3 FDCAN Firmware driver defines . . . 469

25.3.1 FDCAN . . . 469
26 HAL GFXMMU Generic Driver . . . 497
26.1 GFXMMU Firmware driver registers structures . . . 497
26.1.1 GFXMMU_BuffersTypeDef . . . 497
26.1.2 GFXMMU_InterruptsTypeDef . . . 497
26.1.3 GFXMMU_InitTypeDef . . . 497
26.1.4 __GFXMMU_HandleTypeDef . . . 498
26.1.5 GFXMMU_LutLineTypeDef . . . 498
26.1.6 GFXMMU_PackingTypeDef . . . 499
26.2 GFXMMU Firmware driver API description . . . 499
26.2.1 How to use this driver . . . 499
26.2.2 Detailed description of functions . . . 501
26.3 GFXMMU Firmware driver defines . . . 503
26.3.1 GFXMMU . . . 503
27 HAL GFXTIM Generic Driver . . . 505
27.1 GFXTIM Firmware driver registers structures . . . 505
27.1.1 GFXTIM_InitTypeDef . . . 505
27.1.2 __GFXTIM_HandleTypeDef . . . 505
27.1.3 GFXTIM_ClockGeneratorConfigTypeDef . . . 506
27.1.4 GFXTIM_AbsoluteTimerConfigTypeDef . . . 507
27.1.5 GFXTIM_RelativeTimerConfigTypeDef . . . 508
27.1.6 GFXTIM_EventGeneratorConfigTypeDef . . . 508
27.1.7 GFXTIM_WatchdogConfigTypeDef . . . 508
27.2 GFXTIM Firmware driver API description . . . 509
27.2.1 How to use this driver . . . 509
27.2.2 Detailed description of functions . . . 509
27.3 GFXTIM Firmware driver defines . . . 516
27.3.1 GFXTIM . . . 516
28 HAL GPIO Generic Driver . . . 524
28.1 GPIO Firmware driver registers structures . . . 524
28.1.1 GPIO_InitTypeDef . . . 524
28.1.2 GPIO_DelayTypeDef . . . 524
28.1.3 GPIO_RetimeTypeDef . . . 524
28.2 GPIO Firmware driver API description . . . 524
28.2.1 GPIO Peripheral features . . . 525
28.2.2 How to use this driver . . . 525
28.2.3 IO attributes functions . . . 525

28.2.4 Initialization and de-initialization functions . . . 526
28.2.5 Detailed description of functions . . . 526
28.3 GPIO Firmware driver defines . . . 532
28.3.1 GPIO . . . 532
29 HAL GPIO Extension Driver . . . 537
29.1 GPIOEx Firmware driver defines . . . 537
29.1.1 GPIOEx . . . 537
30 HAL GPU2D Generic Driver . . . 543
30.1 GPU2D Firmware driver registers structures . . . 543
30.1.1 GPU2D_HandleTypeDef . . . 543
30.2 GPU2D Firmware driver API description . . . 543
30.2.1 How to use this driver . . . 543
30.2.2 Detailed description of functions . . . 544
30.3 GPU2D Firmware driver defines . . . 546
30.3.1 GPU2D . . . 546
31 HAL HASH Generic Driver . . . 548
31.1 HASH Firmware driver registers structures . . . 548
31.1.1 HASH_ConfigTypeDef . . . 548
31.1.2 __HASH_HandleTypeDef . . . 548
31.2 HASH Firmware driver API description . . . 549
31.2.1 How to use this driver . . . 549
31.2.2 Detailed description of functions . . . 550
31.3 HASH Firmware driver defines . . . 554
31.3.1 HASH . . . 554
32 HAL HCD Generic Driver . . . 559
32.1 HCD Firmware driver registers structures . . . 559
32.1.1 __HCD_HandleTypeDef . . . 559
32.2 HCD Firmware driver API description . . . 560
32.2.1 How to use this driver . . . 560
32.2.2 Detailed description of functions . . . 560
32.3 HCD Firmware driver defines . . . 564
32.3.1 HCD . . . 564
33 HAL I2C Generic Driver . . . 566
33.1 I2C Firmware driver registers structures . . . 566
33.1.1 I2C_InitTypeDef . . . 566
33.1.2 __I2C_HandleTypeDef . . . 566
33.2 I2C Firmware driver API description . . . 568

33.2.1 How to use this driver . . . 568
33.2.2 Detailed description of functions . . . 573
33.3 I2C Firmware driver defines . . . 581
33.3.1 I2C . . . 581
34 HAL I2C Extension Driver . . . 588
34.1 I2CEx Firmware driver API description . . . 588
34.1.1 I2C peripheral Extended features . . . 588
34.1.2 How to use this driver . . . 588
34.1.3 Detailed description of functions . . . 588
34.2 I2CEx Firmware driver defines . . . 589
34.2.1 I2CEx . . . 589
35 HAL I2S Generic Driver . . . 590
35.1 I2S Firmware driver registers structures . . . 590
35.1.1 I2S_InitTypeDef . . . 590
35.1.2 __I2S_HandleTypeDef . . . 590
35.2 I2S Firmware driver API description . . . 592
35.2.1 How to use this driver . . . 592
35.2.2 Detailed description of functions . . . 594
35.3 I2S Firmware driver defines . . . 598
35.3.1 I2S . . . 598
36 HAL I3C Generic Driver . . . 604
36.1 I3C Firmware driver registers structures . . . 604
36.1.1 I3C_InitTypeDef . . . 604
36.1.2 I3C_FifoConfTypeDef . . . 604
36.1.3 I3C_CtrlConfTypeDef . . . 604
36.1.4 I3C_TgtConfTypeDef . . . 605
36.1.5 I3C_DeviceConfTypeDef . . . 606
36.1.6 I3C_CCCInfoTypeDef . . . 607
36.1.7 I3C_ControlTypeDef . . . 608
36.1.8 I3C_DataTypeDef . . . 608
36.1.9 I3C_CCCTypeDef . . . 608
36.1.10 I3C_BCRTypeDef . . . 608
36.1.11 I3C_PIDTypeDef . . . 609
36.1.12 I3C_ENTDAAPayloadTypeDef . . . 609
36.1.13 I3C_PrivateTypeDef . . . 610
36.1.14 I3C_XferTypeDef . . . 610
36.1.15 __I3C_HandleTypeDef . . . 610

36.2 I3C Firmware driver API description . . . 612
36.2.1 How to use this driver . . . 612
36.2.2 Detailed description of functions . . . 617
36.3 I3C Firmware driver defines . . . 627
36.3.1 I3C . . . 627
37 HAL ICACHE Generic Driver . . . 638
37.1 ICACHE Firmware driver API description . . . 638
37.1.1 How to use this driver . . . 638
37.1.2 Detailed description of functions . . . 638
37.2 ICACHE Firmware driver defines . . . 640
37.2.1 ICACHE . . . 640
38 HAL IRDA Generic Driver . . . 643
38.1 IRDA Firmware driver registers structures . . . 643
38.1.1 IRDA_InitTypeDef . . . 643
38.1.2 __IRDA_HandleTypeDef . . . 643
38.2 IRDA Firmware driver API description . . . 645
38.2.1 How to use this driver . . . 645
38.2.2 Callback registration . . . 646
38.2.3 Detailed description of functions . . . 647
38.3 IRDA Firmware driver defines . . . 651
38.3.1 IRDA . . . 651
39 HAL IRDA Extension Driver . . . 662
39.1 IRDAEx Firmware driver defines . . . 662
39.1.1 IRDAEx . . . 662
40 HAL IWDG Generic Driver . . . 663
40.1 IWDG Firmware driver registers structures . . . 663
40.1.1 IWDG_InitTypeDef . . . 663
40.1.2 __IWDG_HandleTypeDef . . . 663
40.2 IWDG Firmware driver API description . . . 663
40.2.1 IWDG Generic features . . . 663
40.2.2 How to use this driver . . . 664
40.2.3 Detailed description of functions . . . 665
40.3 IWDG Firmware driver defines . . . 666
40.3.1 IWDG . . . 666
41 HAL JPEG Generic Driver . . . 668
41.1 JPEG Firmware driver registers structures . . . 668
41.1.1 JPEG_ConfTypeDef . . . 668

41.1.2 JPEG_HandleTypeDef . . . 668
41.2 JPEG Firmware driver API description . . . 669
41.2.1 How to use this driver . . . 669
41.2.2 Detailed description of functions . . . 671
41.3 JPEG Firmware driver defines . . . 675
41.3.1 JPEG . . . 675
42 HAL LPTIM Generic Driver . . . 680
42.1 LPTIM Firmware driver registers structures . . . 680
42.1.1 LPTIM_ClockConfigTypeDef . . . 680
42.1.2 LPTIM_ULPClockConfigTypeDef . . . 680
42.1.3 LPTIM_TriggerConfigTypeDef . . . 680
42.1.4 LPTIM_InitTypeDef . . . 680
42.1.5 LPTIM_OC_ConfigTypeDef . . . 681
42.1.6 LPTIM_IC_ConfigTypeDef . . . 681
42.1.7 __LPTIM_HandleTypeDef . . . 682
42.2 LPTIM Firmware driver API description . . . 683
42.2.1 How to use this driver . . . 683
42.2.2 Detailed description of functions . . . 685
42.3 LPTIM Firmware driver defines . . . 693
42.3.1 LPTIM . . . 693
43 HAL LTDC Generic Driver . . . 706
43.1 LTDC Firmware driver registers structures . . . 706
43.1.1 LTDC_ColorTypeDef . . . 706
43.1.2 LTDC_InitTypeDef . . . 706
43.1.3 LTDC_LayerCfgTypeDef . . . 707
43.1.4 LTDC_LayerTypeDef . . . 708
43.1.5 LTDC_FlexARGBTypeDef . . . 709
43.1.6 LTDC_LayerFlexARGBTypeDef . . . 710
43.1.7 LTDC_FlexYUVTypeDef . . . 710
43.1.8 LTDC_LayerFlexYUVCoPlanarTypeDef . . . 710
43.1.9 LTDC_YUVSemiPlanarAddressTypeDef . . . 711
43.1.10 LTDC_YUVFullPlanarAddressTypeDef . . . 711
43.1.11 LTDC_LayerFlexYUVSemiPlanarTypeDef . . . 711
43.1.12 LTDC_LayerFlexYUVFullPlanarTypeDef . . . 711
43.1.13 __LTDC_HandleTypeDef . . . 712
43.2 LTDC Firmware driver API description . . . 713
43.2.1 How to use this driver . . . 713

43.2.2 Detailed description of functions . . . 714
43.3 LTDC Firmware driver defines . . . 723
43.3.1 LTDC . . . 723
44 HAL MCE Generic Driver . . . 732
44.1 MCE Firmware driver registers structures . . . 732
44.1.1 MCE_RegionConfigTypeDef . . . 732
44.1.2 MCE_AESConfigTypeDef . . . 732
44.1.3 MCE_NoekeonConfigTypeDef . . . 732
44.1.4 __MCE_HandleTypeDef . . . 733
44.2 MCE Firmware driver defines . . . 733
44.2.1 MCE . . . 733
45 HAL MDF Generic Driver . . . 738
45.1 MDF Firmware driver registers structures . . . 738
45.1.1 MDF_ClockTriggerTypeDef . . . 738
45.1.2 MDF_OutputClockTypeDef . . . 738
45.1.3 MDF_CommonParamTypeDef . . . 738
45.1.4 MDF_SerialInterfaceTypeDef . . . 739
45.1.5 MDF_InitTypeDef . . . 739
45.1.6 __MDF_HandleTypeDef . . . 739
45.1.7 MDF_ReshapeFilterTypeDef . . . 740
45.1.8 MDF_HighPassFilterTypeDef . . . 740
45.1.9 MDF_IntegratorTypeDef . . . 741
45.1.10 MDF_SoundActivityTypeDef . . . 741
45.1.11 MDF_FilterTriggerTypeDef . . . 742
45.1.12 MDF_FilterConfigTypeDef . . . 742
45.1.13 MDF_SnapshotParamTypeDef . . . 743
45.1.14 MDF_DmaConfigTypeDef . . . 744
45.1.15 MDF_ScdConfigTypeDef . . . 744
45.1.16 MDF_OldConfigTypeDef . . . 744
45.2 MDF Firmware driver API description . . . 745
45.2.1 How to use this driver . . . 745
45.2.2 Detailed description of functions . . . 747
45.3 MDF Firmware driver defines . . . 754
45.3.1 MDF . . . 754
46 HAL MDIOS Generic Driver . . . 765
46.1 MDIOS Firmware driver registers structures . . . 765
46.1.1 MDIOS_InitTypeDef . . . 765

46.1.2 MDIOS_HandleTypeDef . . . 765
46.2 MDIOS Firmware driver API description . . . 765
46.2.1 How to use this driver . . . 765
46.2.2 Detailed description of functions . . . 766
46.3 MDIOS Firmware driver defines . . . 768
46.3.1 MDIOS . . . 768
47 HAL MMC Generic Driver . . . 776
47.1 MMC Firmware driver registers structures . . . 776
47.1.1 HAL_MMC_CardInfoTypeDef . . . 776
47.1.2 __MMC_HandleTypeDef . . . 776
47.1.3 HAL_MMC_CardCSDTypeDef . . . 777
47.1.4 HAL_MMC_CardCIDTypeDef . . . 780
47.2 MMC Firmware driver API description . . . 780
47.2.1 How to use this driver . . . 780
47.2.2 Detailed description of functions . . . 784
47.3 MMC Firmware driver defines . . . 790
47.3.1 MMC . . . 790
48 HAL MMC Extension Driver . . . 802
48.1 MMCEx Firmware driver API description . . . 802
48.1.1 How to use this driver . . . 802
48.1.2 Detailed description of functions . . . 802
48.2 MMCEx Firmware driver defines . . . 803
48.2.1 MMCEx . . . 803
49 HAL NAND Generic Driver . . . 805
49.1 NAND Firmware driver registers structures . . . 805
49.1.1 NAND_IDTypeDef . . . 805
49.1.2 NAND_AddressTypeDef . . . 805
49.1.3 NAND_DeviceConfigTypeDef . . . 805
49.1.4 __NAND_HandleTypeDef . . . 806
49.2 NAND Firmware driver API description . . . 806
49.2.1 How to use this driver . . . 806
49.2.2 Detailed description of functions . . . 807
49.3 NAND Firmware driver defines . . . 811
49.3.1 NAND . . . 811
50 HAL NOR Generic Driver . . . 812
50.1 NOR Firmware driver registers structures . . . 812
50.1.1 NOR_IDTypeDef . . . 812

50.1.2 NOR_CFITypeDef . . . 812
50.1.3 __NOR_HandleTypeDef . . . 812
50.2 NOR Firmware driver API description . . . 813
50.2.1 How to use this driver . . . 813
50.2.2 Detailed description of functions . . . 814
50.3 NOR Firmware driver defines . . . 816
50.3.1 NOR . . . 816
51 HAL PCD Generic Driver . . . 818
51.1 PCD Firmware driver registers structures . . . 818
51.1.1 __PCD_HandleTypeDef . . . 818
51.2 PCD Firmware driver API description . . . 820
51.2.1 How to use this driver . . . 820
51.2.2 Detailed description of functions . . . 820
51.3 PCD Firmware driver defines . . . 826
51.3.1 PCD . . . 826
52 HAL PCD Extension Driver . . . 828
52.1 PCDEx Firmware driver API description . . . 828
52.1.1 Detailed description of functions . . . 828
53 HAL PSSI Generic Driver . . . 830
53.1 PSSI Firmware driver registers structures . . . 830
53.1.1 PSSI_InitTypeDef . . . 830
53.1.2 __PSSI_HandleTypeDef . . . 830
53.2 PSSI Firmware driver API description . . . 831
53.2.1 How to use this driver . . . 831
53.2.2 Detailed description of functions . . . 832
53.3 PSSI Firmware driver defines . . . 835
53.3.1 PSSI . . . 835
54 HAL PWR Generic Driver . . . 840
54.1 PWR Firmware driver registers structures . . . 840
54.1.1 PWR_PVDTypeDef . . . 840
54.2 PWR Firmware driver API description . . . 840
54.2.1 PWR peripheral overview . . . 840
54.2.2 How to use this driver . . . 840
54.2.3 Initialization and De-Initialization Functions . . . 841
54.2.4 Peripheral Control Functions . . . 841
54.2.5 PWR Attributes Functions . . . 843
54.2.6 Detailed description of functions . . . 843

54.3 PWR Firmware driver defines . . . 851
54.3.1 PWR . . . 851
55 HAL RAMCFG Generic Driver . . . 861
55.1 RAMCFG Firmware driver registers structures . . . 861
55.1.1 __RAMCFG_HandleTypeDef . . . 861
55.2 RAMCFG Firmware driver API description . . . 861
55.2.1 How to use this driver . . . 861
55.2.2 Detailed description of functions . . . 862
55.3 RAMCFG Firmware driver defines . . . 865
55.3.1 RAMCFG . . . 865
56 HAL RCC Generic Driver . . . 869
56.1 RCC Firmware driver registers structures . . . 869
56.1.1 RCC_PLLInitTypeDef . . . 869
56.1.2 RCC_OscInitTypeDef . . . 869
56.1.3 RCC_ICInitTypeDef . . . 870
56.1.4 RCC_ClkInitTypeDef . . . 870
56.2 RCC Firmware driver API description . . . 871
56.2.1 RCC specific features . . . 871
56.2.2 RCC Limitations . . . 872
56.2.3 Initialization and de-initialization functions . . . 872
56.2.4 Peripheral Control functions . . . 873
56.2.5 RCC attributes functions . . . 873
56.2.6 Detailed description of functions . . . 873
56.3 RCC Firmware driver defines . . . 880
56.3.1 RCC . . . 880
57 HAL RCC Extension Driver . . . 957
57.1 RCCEx Firmware driver registers structures . . . 957
57.1.1 RCC_PeriphCLKInitTypeDef . . . 957
57.1.2 RCC_PLLSSCGInitTypeDef . . . 960
57.2 RCCEx Firmware driver API description . . . 961
57.2.1 Extended Peripheral Control functions . . . 961
57.2.2 Detailed description of functions . . . 961
57.3 RCCEx Firmware driver defines . . . 967
57.3.1 RCCEx . . . 967
58 HAL RIF Generic Driver . . . 1021
58.1 RIF Firmware driver registers structures . . . 1021
58.1.1 RIMC_MasterConfig_t . . . 1021

58.1.2 RISAF_BaseRegionConfig_t . . . 1021
58.1.3 RISAF_SubRegionConfig_t . . . 1021
58.1.4 RISAF_DelegationConfig_t . . . 1022
58.1.5 RISAF_IllegalAccessData_t . . . 1022
58.1.6 RISAF_IllegalAccess_t . . . 1022
58.2 RIF Firmware driver API description . . . 1023
58.2.1 How to use this driver . . . 1023
58.2.2 Detailed description of functions . . . 1024
58.3 RIF Firmware driver defines . . . 1027
58.3.1 RIF . . . 1027
59 HAL RNG Generic Driver . . . 1038
59.1 RNG Firmware driver registers structures . . . 1038
59.1.1 RNG_InitTypeDef . . . 1038
59.1.2 __RNG_HandleTypeDef . . . 1038
59.2 RNG Firmware driver API description . . . 1038
59.2.1 How to use this driver . . . 1039
59.2.2 Callback registration . . . 1039
59.2.3 Detailed description of functions . . . 1039
59.3 RNG Firmware driver defines . . . 1041
59.3.1 RNG . . . 1041
60 HAL RNG Extension Driver . . . 1045
60.1 RNGEx Firmware driver registers structures . . . 1045
60.1.1 RNG_ConfigTypeDef . . . 1045
60.2 RNGEx Firmware driver API description . . . 1045
60.2.1 Detailed description of functions . . . 1045
60.3 RNGEx Firmware driver defines . . . 1046
60.3.1 RNGEx . . . 1046
61 HAL RTC Generic Driver . . . 1048
61.1 RTC Firmware driver registers structures . . . 1048
61.1.1 RTC_InitTypeDef . . . 1048
61.1.2 RTC_TimeTypeDef . . . 1048
61.1.3 RTC_DateTypeDef . . . 1049
61.1.4 RTC_AlarmTypeDef . . . 1049
61.1.5 __RTC_HandleTypeDef . . . 1050
61.2 RTC Firmware driver API description . . . 1052
61.2.1 RTC Operating Condition . . . 1052
61.2.2 Backup Domain Reset . . . 1052

61.2.3 Backup Domain Access . . . 1052
61.2.4 How to use RTC Driver . . . 1052
61.2.5 RTC and low power modes . . . 1053
61.2.6 Detailed description of functions . . . 1053
61.3 RTC Firmware driver defines . . . 1057
61.3.1 RTC . . . 1057
62 HAL RTC Extension Driver . . . 1067
62.1 RTCEx Firmware driver registers structures . . . 1067
62.1.1 RTC_TamperTypeDef . . . 1067
62.1.2 RTC_ATampInputTypeDef . . . 1067
62.1.3 RTC_ActiveTampersTypeDef . . . 1068
62.1.4 RTC_InternalTamperTypeDef . . . 1068
62.1.5 RTC_SecureStateTypeDef . . . 1068
62.1.6 RTC_PrivilegeStateTypeDef . . . 1069
62.2 RTCEx Firmware driver API description . . . 1070
62.2.1 How to use this driver . . . 1070
62.2.2 Detailed description of functions . . . 1072
62.3 RTCEx Firmware driver defines . . . 1081
62.3.1 RTCEx . . . 1081
63 HAL SAI Generic Driver . . . 1110
63.1 SAI Firmware driver registers structures . . . 1110
63.1.1 SAI_PdmInitTypeDef . . . 1110
63.1.2 SAI_InitTypeDef . . . 1110
63.1.3 SAI_FrameInitTypeDef . . . 1111
63.1.4 SAI_SlotInitTypeDef . . . 1112
63.1.5 __SAI_HandleTypeDef . . . 1112
63.2 SAI Firmware driver API description . . . 1114
63.2.1 How to use this driver . . . 1114
63.2.2 Detailed description of functions . . . 1116
63.3 SAI Firmware driver defines . . . 1120
63.3.1 SAI . . . 1120
64 HAL SAI Extension Driver . . . 1129
64.1 SAIEx Firmware driver registers structures . . . 1129
64.1.1 SAIEx_PdmMicDelayParamTypeDef . . . 1129
64.2 SAIEx Firmware driver API description . . . 1129
64.2.1 Detailed description of functions . . . 1129
65 HAL SDIO Generic Driver . . . 1130

65.1 SDIO Firmware driver registers structures . . . 1130
65.1.1 HAL_SDIO_CCCR_TypeDef . . . 1130
65.1.2 HAL_SDIO_FBR_t . . . 1130
65.1.3 HAL_SDIO_DirectCmd_TypeDef . . . 1130
65.1.4 HAL_SDIO_ExtendedCmd_TypeDef . . . 1131
65.1.5 __SDIO_HandleTypeDef . . . 1131
65.2 SDIO Firmware driver API description . . . 1132
65.2.1 How to use this driver . . . 1132
65.2.2 Detailed description of functions . . . 1135
65.3 SDIO Firmware driver defines . . . 1140
65.3.1 SDIO . . . 1140
66 HAL SDRAM Generic Driver . . . 1145
66.1 SDRAM Firmware driver registers structures . . . 1145
66.1.1 __SDRAM_HandleTypeDef . . . 1145
66.2 SDRAM Firmware driver API description . . . 1145
66.2.1 How to use this driver . . . 1145
66.2.2 Detailed description of functions . . . 1146
66.3 SDRAM Firmware driver defines . . . 1150
66.3.1 SDRAM . . . 1150
67 HAL SD Generic Driver . . . 1151
67.1 SD Firmware driver registers structures . . . 1151
67.1.1 HAL_SD_CardInfoTypeDef . . . 1151
67.1.2 __SD_HandleTypeDef . . . 1151
67.1.3 HAL_SD_CardCSDTypeDef . . . 1152
67.1.4 HAL_SD_CardCIDTypeDef . . . 1155
67.1.5 HAL_SD_CardStatusTypeDef . . . 1155
67.2 SD Firmware driver API description . . . 1156
67.2.1 How to use this driver . . . 1156
67.2.2 Detailed description of functions . . . 1159
67.3 SD Firmware driver defines . . . 1163
67.3.1 SD . . . 1163
68 HAL SD Extension Driver . . . 1173
68.1 SDEx Firmware driver API description . . . 1173
68.1.1 How to use this driver . . . 1173
68.1.2 Detailed description of functions . . . 1173
68.2 SDEx Firmware driver defines . . . 1174
68.2.1 SDEx . . . 1174

69 HAL SMARTCARD Generic Driver . . . 1176
69.1 SMARTCARD Firmware driver registers structures . . . 1176
69.1.1 SMARTCARD_InitTypeDef . . . 1176
69.1.2 SMARTCARD_AdvFeatureInitTypeDef . . . 1177
69.1.3 __SMARTCARD_HandleTypeDef . . . 1178
69.2 SMARTCARD Firmware driver API description . . . 1180
69.2.1 How to use this driver . . . 1180
69.2.2 Callback registration . . . 1181
69.2.3 Detailed description of functions . . . 1182
69.3 SMARTCARD Firmware driver defines . . . 1186
69.3.1 SMARTCARD . . . 1186
70 HAL SMARTCARD Extension Driver . . . 1199
70.1 SMARTCARDEx Firmware driver API description . . . 1199
70.1.1 SMARTCARD peripheral extended features . . . 1199
70.1.2 Detailed description of functions . . . 1199
70.2 SMARTCARDEx Firmware driver defines . . . 1200
70.2.1 SMARTCARDEx . . . 1200
71 HAL SMBUS Generic Driver . . . 1205
71.1 SMBUS Firmware driver registers structures . . . 1205
71.1.1 SMBUS_InitTypeDef . . . 1205
71.1.2 __SMBUS_HandleTypeDef . . . 1206
71.2 SMBUS Firmware driver API description . . . 1207
71.2.1 How to use this driver . . . 1207
71.2.2 Detailed description of functions . . . 1209
71.3 SMBUS Firmware driver defines . . . 1213
71.3.1 SMBUS . . . 1213
72 HAL SMBUS Extension Driver . . . 1221
72.1 SMBUSEx Firmware driver API description . . . 1221
72.1.1 SMBUS peripheral Extended features . . . 1221
72.1.2 How to use this driver . . . 1221
72.1.3 Detailed description of functions . . . 1221
72.2 SMBUSEx Firmware driver defines . . . 1221
72.2.1 SMBUSEx . . . 1221
73 HAL SPDIFRX Generic Driver . . . 1222
73.1 SPDIFRX Firmware driver registers structures . . . 1222
73.1.1 SPDIFRX_InitTypeDef . . . 1222
73.1.2 SPDIFRX_SetDataFormatTypeDef . . . 1223

73.1.3 __SPDIFRX_HandleTypeDef . . . 1223
73.2 SPDIFRX Firmware driver API description . . . 1224
73.2.1 How to use this driver . . . 1224
73.2.2 Detailed description of functions . . . 1226
73.3 SPDIFRX Firmware driver defines . . . 1229
73.3.1 SPDIFRX . . . 1229
74 HAL SPI Generic Driver . . . 1235
74.1 SPI Firmware driver registers structures . . . 1235
74.1.1 SPI_InitTypeDef . . . 1235
74.1.2 __SPI_HandleTypeDef . . . 1236
74.2 SPI Firmware driver API description . . . 1238
74.2.1 How to use this driver . . . 1238
74.2.2 Detailed description of functions . . . 1240
74.3 SPI Firmware driver defines . . . 1244
74.3.1 SPI . . . 1244
75 HAL SPI Extension Driver . . . 1257
75.1 SPIEx Firmware driver API description . . . 1257
75.1.1 Detailed description of functions . . . 1257
76 HAL SRAM Generic Driver . . . 1258
76.1 SRAM Firmware driver registers structures . . . 1258
76.1.1 __SRAM_HandleTypeDef . . . 1258
76.2 SRAM Firmware driver API description . . . 1258
76.2.1 How to use this driver . . . 1258
76.2.2 Detailed description of functions . . . 1259
76.3 SRAM Firmware driver defines . . . 1262
76.3.1 SRAM . . . 1262
77 HAL TIM Generic Driver . . . 1263
77.1 TIM Firmware driver registers structures . . . 1263
77.1.1 TIM_Base_InitTypeDef . . . 1263
77.1.2 TIM_OC_InitTypeDef . . . 1263
77.1.3 TIM_OnePulse_InitTypeDef . . . 1264
77.1.4 TIM_IC_InitTypeDef . . . 1265
77.1.5 TIM_Encoder_InitTypeDef . . . 1265
77.1.6 TIM_ClockConfigTypeDef . . . 1266
77.1.7 TIM_ClearInputConfigTypeDef . . . 1266
77.1.8 TIM_MasterConfigTypeDef . . . 1267
77.1.9 TIM_SlaveConfigTypeDef . . . 1267

77.1.10 TIM_BreakDeadTimeConfigTypeDef . . . 1267
77.1.11 __TIM_HandleTypeDef . . . 1268
77.2 TIM Firmware driver API description . . . 1271
77.2.1 TIMER Generic features . . . 1271
77.2.2 How to use this driver . . . 1271
77.2.3 Detailed description of functions . . . 1273
77.3 TIM Firmware driver defines . . . 1287
77.3.1 TIM . . . 1287
78 HAL TIM Extension Driver . . . 1317
78.1 TIMEx Firmware driver registers structures . . . 1317
78.1.1 TIM_HallSensor_InitTypeDef . . . 1317
78.1.2 TIMEx_BreakInputConfigTypeDef . . . 1317
78.1.3 TIMEx_EncoderIndexConfigTypeDef . . . 1317
78.2 TIMEx Firmware driver API description . . . 1318
78.2.1 TIMER Extended features . . . 1318
78.2.2 How to use this driver . . . 1318
78.2.3 Detailed description of functions . . . 1319
78.3 TIMEx Firmware driver defines . . . 1328
78.3.1 TIMEx . . . 1328
79 HAL UART Generic Driver . . . 1338
79.1 UART Firmware driver registers structures . . . 1338
79.1.1 UART_InitTypeDef . . . 1338
79.1.2 UART_AdvFeatureInitTypeDef . . . 1339
79.1.3 __UART_HandleTypeDef . . . 1339
79.2 UART Firmware driver API description . . . 1342
79.2.1 How to use this driver . . . 1342
79.2.2 Callback registration . . . 1342
79.2.3 Detailed description of functions . . . 1343
79.3 UART Firmware driver defines . . . 1350
79.3.1 UART . . . 1350
80 HAL UART Extension Driver . . . 1371
80.1 UARTEx Firmware driver registers structures . . . 1371
80.1.1 UART_WakeUpTypeDef . . . 1371
80.2 UARTEx Firmware driver API description . . . 1371
80.2.1 UART peripheral extended features . . . 1371
80.2.2 Detailed description of functions . . . 1371
80.3 UARTEx Firmware driver defines . . . 1373

80.3.1 UARTEx . . . 1373
81 HAL USART Generic Driver . . . 1376
81.1 USART Firmware driver registers structures . . . 1376
81.1.1 USART_InitTypeDef . . . 1376
81.1.2 __USART_HandleTypeDef . . . 1376
81.2 USART Firmware driver API description . . . 1378
81.2.1 How to use this driver . . . 1378
81.2.2 Callback registration . . . 1379
81.2.3 Detailed description of functions . . . 1380
81.3 USART Firmware driver defines . . . 1384
81.3.1 USART . . . 1384
82 HAL USART Extension Driver . . . 1396
82.1 USARTEx Firmware driver API description . . . 1396
82.1.1 USART peripheral extended features . . . 1396
82.1.2 Detailed description of functions . . . 1396
82.2 USARTEx Firmware driver defines . . . 1397
82.2.1 USARTEx . . . 1397
83 HAL WWDG Generic Driver . . . 1399
83.1 WWDG Firmware driver registers structures . . . 1399
83.1.1 WWDG_InitTypeDef . . . 1399
83.1.2 __WWDG_HandleTypeDef . . . 1399
83.2 WWDG Firmware driver API description . . . 1399
83.2.1 WWDG Specific features . . . 1399
83.2.2 How to use this driver . . . 1400
83.2.3 Detailed description of functions . . . 1401
83.3 WWDG Firmware driver defines . . . 1402
83.3.1 WWDG . . . 1402
84 HAL XSPI Generic Driver . . . 1405
84.1 XSPI Firmware driver registers structures . . . 1405
84.1.1 XSPI_InitTypeDef . . . 1405
84.1.2 __XSPI_HandleTypeDef . . . 1406
84.1.3 XSPI_RegularCmdTypeDef . . . 1407
84.1.4 XSPI_HyperbusCfgTypeDef . . . 1408
84.1.5 XSPI_HyperbusCmdTypeDef . . . 1409
84.1.6 XSPI_AutoPollingTypeDef . . . 1409
84.1.7 XSPI_MemoryMappedTypeDef . . . 1410
84.1.8 XSPIM_CfgTypeDef . . . 1410

84.1.9 XSPI_HSCalTypeDef . . . 1411
84.2 XSPI Firmware driver API description . . . 1411
84.2.1 How to use this driver . . . 1411
84.2.2 Detailed description of functions . . . 1414
84.3 XSPI Firmware driver defines . . . 1420
84.3.1 XSPI . . . 1420
85 LL ADC Generic Driver . . . 1435
85.1 ADC Firmware driver registers structures . . . 1435
85.1.1 LL_ADC_CommonInitTypeDef . . . 1435
85.1.2 LL_ADC_InitTypeDef . . . 1435
85.1.3 LL_ADC_REG_InitTypeDef . . . 1435
85.1.4 LL_ADC_INJ_InitTypeDef . . . 1436
85.2 ADC Firmware driver API description . . . 1437
85.2.1 Detailed description of functions . . . 1437
85.3 ADC Firmware driver defines . . . 1541
85.3.1 ADC . . . 1541
86 LL BUS Generic Driver . . . 1580
86.1 BUS Firmware driver API description . . . 1580
86.1.1 Detailed description of functions . . . 1580
86.2 BUS Firmware driver defines . . . 1669
86.2.1 BUS . . . 1669
87 LL CACHEAXI Generic Driver . . . 1676
87.1 CACHEAXI Firmware driver API description . . . 1676
87.1.1 Detailed description of functions . . . 1676
87.2 CACHEAXI Firmware driver defines . . . 1689
87.2.1 CACHEAXI . . . 1689
88 LL CORTEX Generic Driver . . . 1691
88.1 CORTEX Firmware driver API description . . . 1691
88.1.1 Detailed description of functions . . . 1691
88.2 CORTEX Firmware driver defines . . . 1707
88.2.1 CORTEX . . . 1707
89 LL CRC Generic Driver . . . 1711
89.1 CRC Firmware driver API description . . . 1711
89.1.1 Detailed description of functions . . . 1711
89.2 CRC Firmware driver defines . . . 1720
89.2.1 CRC . . . 1720

90 LL DMA2D Generic Driver . . . 1723
90.1 DMA2D Firmware driver registers structures . . . 1723
90.1.1 LL_DMA2D_InitTypeDef . . . 1723
90.1.2 LL_DMA2D_LayerCfgTypeDef . . . 1725
90.1.3 LL_DMA2D_ColorTypeDef . . . 1727
90.2 DMA2D Firmware driver API description . . . 1729
90.2.1 Detailed description of functions . . . 1729
90.3 DMA2D Firmware driver defines . . . 1775
90.3.1 DMA2D . . . 1775
91 LL DMA Generic Driver . . . 1779
91.1 DMA Firmware driver registers structures . . . 1779
91.1.1 LL_DMA_InitTypeDef . . . 1779
91.1.2 LL_DMA_InitLinkedListTypeDef . . . 1782
91.1.3 LL_DMA_InitNodeTypeDef . . . 1783
91.1.4 LL_DMA_LinkNodeTypeDef . . . 1785
91.2 DMA Firmware driver API description . . . 1786
91.2.1 LL DMA driver acronyms . . . 1786
91.2.2 Detailed description of functions . . . 1786
91.3 DMA Firmware driver defines . . . 1972
91.3.1 DMA . . . 1972
92 LL EXTI Generic Driver . . . 2007
92.1 EXTI Firmware driver registers structures . . . 2007
92.1.1 LL_EXTI_InitTypeDef . . . 2007
92.2 EXTI Firmware driver API description . . . 2007
92.2.1 Detailed description of functions . . . 2007
92.3 EXTI Firmware driver defines . . . 2072
92.3.1 EXTI . . . 2072
93 LL GPIO Generic Driver . . . 2078
93.1 GPIO Firmware driver registers structures . . . 2078
93.1.1 LL_GPIO_InitTypeDef . . . 2078
93.2 GPIO Firmware driver API description . . . 2078
93.2.1 Detailed description of functions . . . 2078
93.3 GPIO Firmware driver defines . . . 2111
93.3.1 GPIO . . . 2111
94 LL I2C Generic Driver . . . 2116
94.1 I2C Firmware driver registers structures . . . 2116
94.1.1 LL_I2C_InitTypeDef . . . 2116

94.2 I2C Firmware driver API description . . . 2116
94.2.1 Detailed description of functions . . . 2116
94.3 I2C Firmware driver defines . . . 2168
94.3.1 I2C . . . 2168
95 LL I3C Generic Driver . . . 2174
95.1 I3C Firmware driver registers structures . . . 2174
95.1.1 LL_I3C_CtrlBusConfTypeDef . . . 2174
95.1.2 LL_I3C_TgtBusConfTypeDef . . . 2174
95.1.3 LL_I3C_InitTypeDef . . . 2175
95.2 I3C Firmware driver API description . . . 2175
95.2.1 Detailed description of functions . . . 2175
95.3 I3C Firmware driver defines . . . 2272
95.3.1 I3C . . . 2272
96 LL ICACHE Generic Driver . . . 2282
96.1 ICACHE Firmware driver API description . . . 2282
96.1.1 Detailed description of functions . . . 2282
96.2 ICACHE Firmware driver defines . . . 2288
96.2.1 ICACHE . . . 2288
97 LL IWDG Generic Driver . . . 2290
97.1 IWDG Firmware driver API description . . . 2290
97.1.1 Detailed description of functions . . . 2290
97.2 IWDG Firmware driver defines . . . 2297
97.2.1 IWDG . . . 2297
98 LL LPTIM Generic Driver . . . 2299
98.1 LPTIM Firmware driver registers structures . . . 2299
98.1.1 LL_LPTIM_InitTypeDef . . . 2299
98.2 LPTIM Firmware driver API description . . . 2299
98.2.1 Detailed description of functions . . . 2299
98.3 LPTIM Firmware driver defines . . . 2348
98.3.1 LPTIM . . . 2348
99 LL LPUART Generic Driver . . . 2357
99.1 LPUART Firmware driver registers structures . . . 2357
99.1.1 LL_LPUART_InitTypeDef . . . 2357
99.2 LPUART Firmware driver API description . . . 2357
99.2.1 Detailed description of functions . . . 2357
99.3 LPUART Firmware driver defines . . . 2412

99.3.1 LPUART 2412
100 LL PKA Generic Driver 2421
100.1 PKA Firmware driver registers structures 2421
100.1.1 LL_PKA_InitTypeDef 2421
100.2 PKA Firmware driver API description 2421
100.2.1 Detailed description of functions 2421
100.3 PKA Firmware driver defines 2431
100.3.1 PKA 2432
101 LL RCC Generic Driver 2435
101.1 RCC Firmware driver registers structures 2435
101.1.1 LL_RCC_ClocksTypeDef 2435
101.2 RCC Firmware driver API description 2435
101.2.1 Detailed description of functions 2435
101.3 RCC Firmware driver defines 2620
101.3.1 RCC 2620
102 LL RNG Generic Driver 2648
102.1 RNG Firmware driver registers structures 2648
102.1.1 LL_RNG_InitTypeDef 2648
102.2 RNG Firmware driver API description 2648
102.2.1 Detailed description of functions 2648
102.3 RNG Firmware driver defines 2661
102.3.1 RNG 2661
103 LL RTC Generic Driver 2664
103.1 RTC Firmware driver registers structures 2664
103.1.1 LL_RTC_InitTypeDef 2664
103.1.2 LL_RTC_TimeTypeDef 2664
103.1.3 LL_RTC_DateTypeDef 2664
103.1.4 LL_RTC_AlarmTypeDef 2665
103.2 RTC Firmware driver API description 2665
103.2.1 Detailed description of functions 2665
103.3 RTC Firmware driver defines 2794
103.3.1 RTC 2794
104 LL SPI Generic Driver 2813
104.1 SPI Firmware driver registers structures 2813
104.1.1 LL_SPI_InitTypeDef 2813
104.1.2 LL_I2S_InitTypeDef 2814
104.2 SPI Firmware driver API description 2814

104.2.1 Detailed description of functions . . . 2814
104.3 SPI Firmware driver defines . . . 2890
104.3.1 SPI . . . 2890
105 LL SYSTEM Generic Driver . . . 2898
105.1 SYSTEM Firmware driver defines . . . 2898
105.1.1 SYSTEM . . . 2898
106 LL TIM Generic Driver . . . 2905
106.1 TIM Firmware driver registers structures . . . 2905
106.1.1 LL_TIM_InitTypeDef . . . 2905
106.1.2 LL_TIM_OC_InitTypeDef . . . 2905
106.1.3 LL_TIM_IC_InitTypeDef . . . 2906
106.1.4 LL_TIM_ENCODER_InitTypeDef . . . 2906
106.1.5 LL_TIM_HALLSENSOR_InitTypeDef . . . 2907
106.1.6 LL_TIM_BDTR_InitTypeDef . . . 2908
106.2 TIM Firmware driver API description . . . 2909
106.2.1 Detailed description of functions . . . 2910
106.3 TIM Firmware driver defines . . . 3025
106.3.1 TIM . . . 3025
107 LL UCPD Generic Driver . . . 3055
107.1 UCPD Firmware driver registers structures . . . 3055
107.1.1 LL_UCPD_InitTypeDef . . . 3055
107.2 UCPD Firmware driver API description . . . 3055
107.2.1 Detailed description of functions . . . 3055
107.3 UCPD Firmware driver defines . . . 3093
107.3.1 UCPD . . . 3094
108 LL USART Generic Driver . . . 3100
108.1 USART Firmware driver registers structures . . . 3100
108.1.1 LL_USART_InitTypeDef . . . 3100
108.1.2 LL_USART_ClockInitTypeDef . . . 3100
108.2 USART Firmware driver API description . . . 3101
108.2.1 Detailed description of functions . . . 3101
108.3 USART Firmware driver defines . . . 3197
108.3.1 USART . . . 3197
109 LL UTILS Generic Driver . . . 3207
109.1 UTILS Firmware driver registers structures . . . 3207
109.1.1 LL_UTILS_PLLInitTypeDef . . . 3207
109.1.2 LL_UTILS_ICInitTypeDef . . . 3207

109.1.3 LL_UTILS_ClkInitTypeDef 3208
109.2 UTILS Firmware driver API description 3208
109.2.1 System Configuration functions 3208
109.2.2 Detailed description of functions 3209
109.3 UTILS Firmware driver defines 3213
109.3.1 UTILS 3213
110 LL VENC Generic Driver 3215
110.1 VENC Firmware driver API description 3215
110.1.1 Detailed description of functions 3215
110.2 VENC Firmware driver defines 3215
110.2.1 VENC 3215
111 LL WWDG Generic Driver 3216
111.1 WWDG Firmware driver API description 3216
111.1.1 Detailed description of functions 3216
111.2 WWDG Firmware driver defines 3220
111.2.1 WWDG 3220
112 FAQs 3222
Revision history 3225

List of tables

Table 1. Acronyms and definitions . . . 4
Table 2. HAL driver files . . . 8
Table 3. User-application files . . . 8
Table 4. API classification . . . 13
Table 5. List of devices supported by HAL/LL drivers . . . 14
Table 6. HAL API naming rules . . . 17
Table 7. Macros handling interrupts and specific clock configurations . . . 18
Table 8. Callback functions . . . 19
Table 9. HAL generic APIs . . . 20
Table 10. HAL extension APIs . . . 20
Table 11. Define statements used for HAL configuration . . . 23
Table 12. Description of GPIO_InitTypeDef structure . . . 25
Table 13. Description of EXTI configuration macros . . . 26
Table 14. MSP functions . . . 34
Table 15. Timeout values . . . 37
Table 16. LL driver files . . . 40
Table 17. Common peripheral initialization functions . . . 42
Table 18. Optional peripheral initialization functions . . . 42
Table 19. Specific interrupt, DMA request and status flags management . . . 43
Table 20. Available function formats . . . 43
Table 21. Peripheral clock activation/deactivation management . . . 44
Table 22. Peripheral activation/deactivation management . . . 44
Table 23. Peripheral configuration management . . . 44
Table 24. Peripheral register management . . . 44
Table 25. Document revision history . . . 3225

List of figures

Figure 1. Example of project template . . . 10
Figure 2. Adding device-specific functions . . . 21
Figure 3. Adding family-specific functions . . . 21
Figure 4. Adding new peripherals . . . 22
Figure 5. File inclusion model . . . 22
Figure 6. HAL driver model . . . 31
Figure 7. Low-layer driver folders . . . 41
Figure 8. Low-layer driver CMSIS files . . . 41

IMPORTANT NOTICE – READ CAREFULLY

STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST’s terms and conditions of sale in place at the time of order acknowledgment.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of purchasers’ products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. For additional information about ST trademarks, refer to www.st.com/trademarks. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2025 STMicroelectronics – All rights reserved