

< STM32 MCUs

**B. Montanari**

ST Technical Moderator · 1 year ago



STM32N6 FSBL explained

0 replies

Summary

The first stage bootloader (FSBL) is a key component in the boot process of STM32N6 microcontrollers. It is responsible for initializing the system, configuring the hardware, and loading the application code from external memory into the internal or external memories for execution. This article provides a quick tutorial on how to use the FSBL in the simplified mode.

Introduction

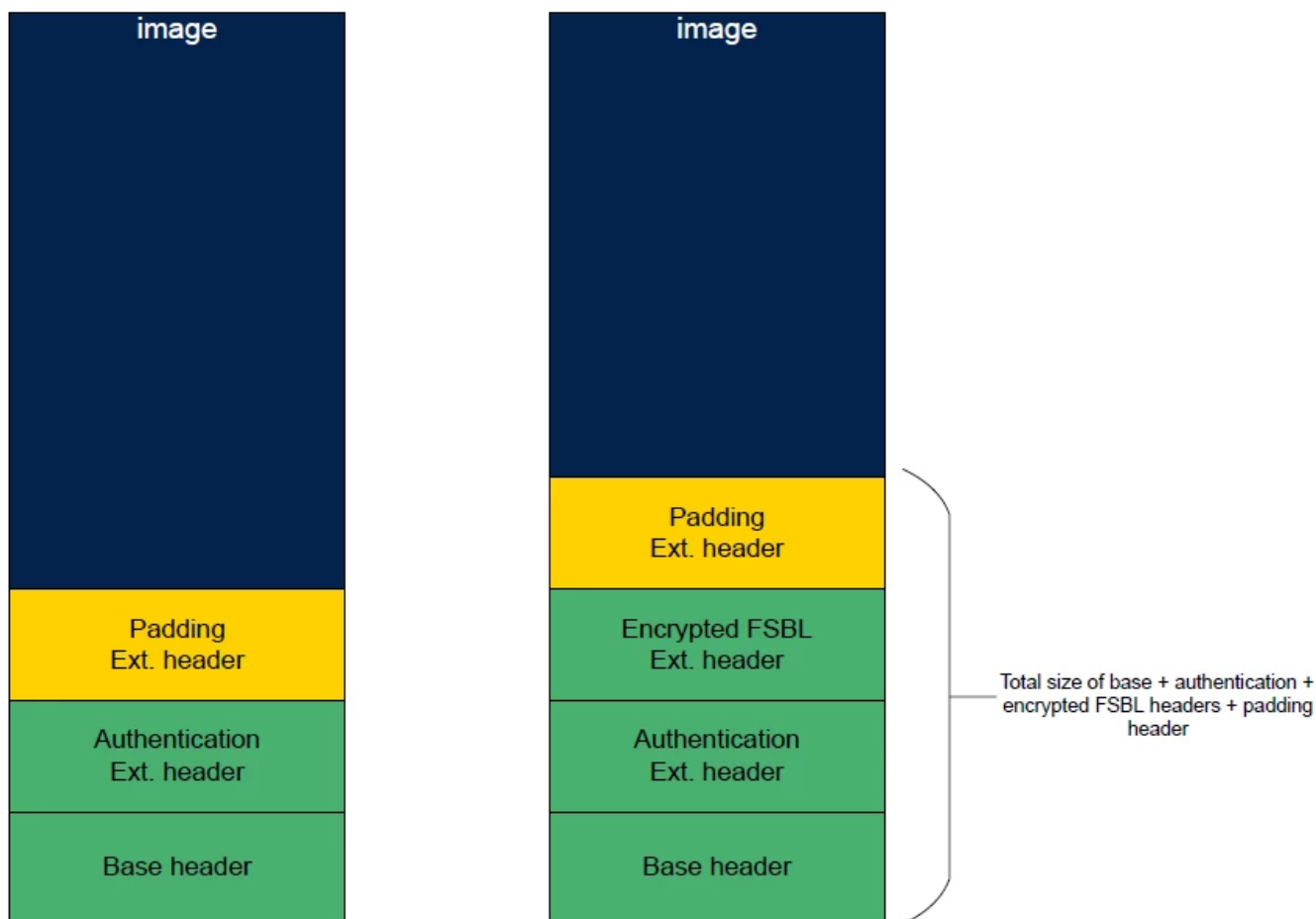
At power-on, the boot ROM copies the FSBL binary from the external memory into the internal SRAM. Once the boot ROM task is completed, it jumps to the FSBL project. It is usually responsible for executing the clock and system settings and configuring the external memories. Finally, it copies the application binary in internal SRAM or sets the external memory in memory-mapped mode. When done, the application itself starts up and runs. If you want to know more about the boot ROM, refer to this [article](#).

On STM32N6 MCUs, the first-stage bootloader (FSBL) must be signed, so the boot ROM can execute it in a secured-locked state. To ensure that the FSBL signature is done correctly, the user can request the boot ROM traces, which are stored in AXISRAM2.

The FSBL layout includes several key components: the image header, base header, authentication process, and padding. These elements work together to ensure the integrity and authenticity of the boot process. The outlook is described by this image:

Feedback

 Ask AI



1.1 Image header

The image header contains metadata about the FSBL, such as its size, version, and entry point. This information is used by the boot ROM to load and execute the FSBL correctly.

1.2 Base header

The base header provides additional configuration details, including memory addresses and security settings. It ensures that the FSBL is loaded into the correct memory location and that any necessary security measures are applied.

1.3 Authentication process

The authentication process verifies the integrity and authenticity of the FSBL. This is typically done using cryptographic techniques, such as digital signatures, to ensure that the FSBL has not been tampered with.

1.4 Padding

Padding is used to align the FSBL image to specific memory boundaries. This ensures that the image is loaded correctly and can be executed without any issues.

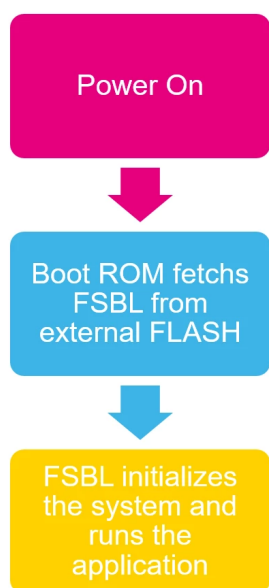
2. FSBL main features

The FSBL can be used in several different ways, each suited to specific application requirements. The following sections describe the various modes supported by the FSBL.

2.1 FSBL with basic load and run mode

In the basic load and run mode, the boot ROM fetches the FSBL from external flash memory. The FSBL in this case is not only responsible for initializing the system, but also contains the application code, which has its execution from the same SRAM address. This mode is used in approximately 90% of the examples available in the STM32Cube_FW_N6. The main limitation is that the entire code cannot surpass 511 KB. This size limitation happens because the boot ROM copies at most 512 KB. The first 1 KB is reserved for the headers, authentication padding.

This is the simplified diagram on the usual steps:



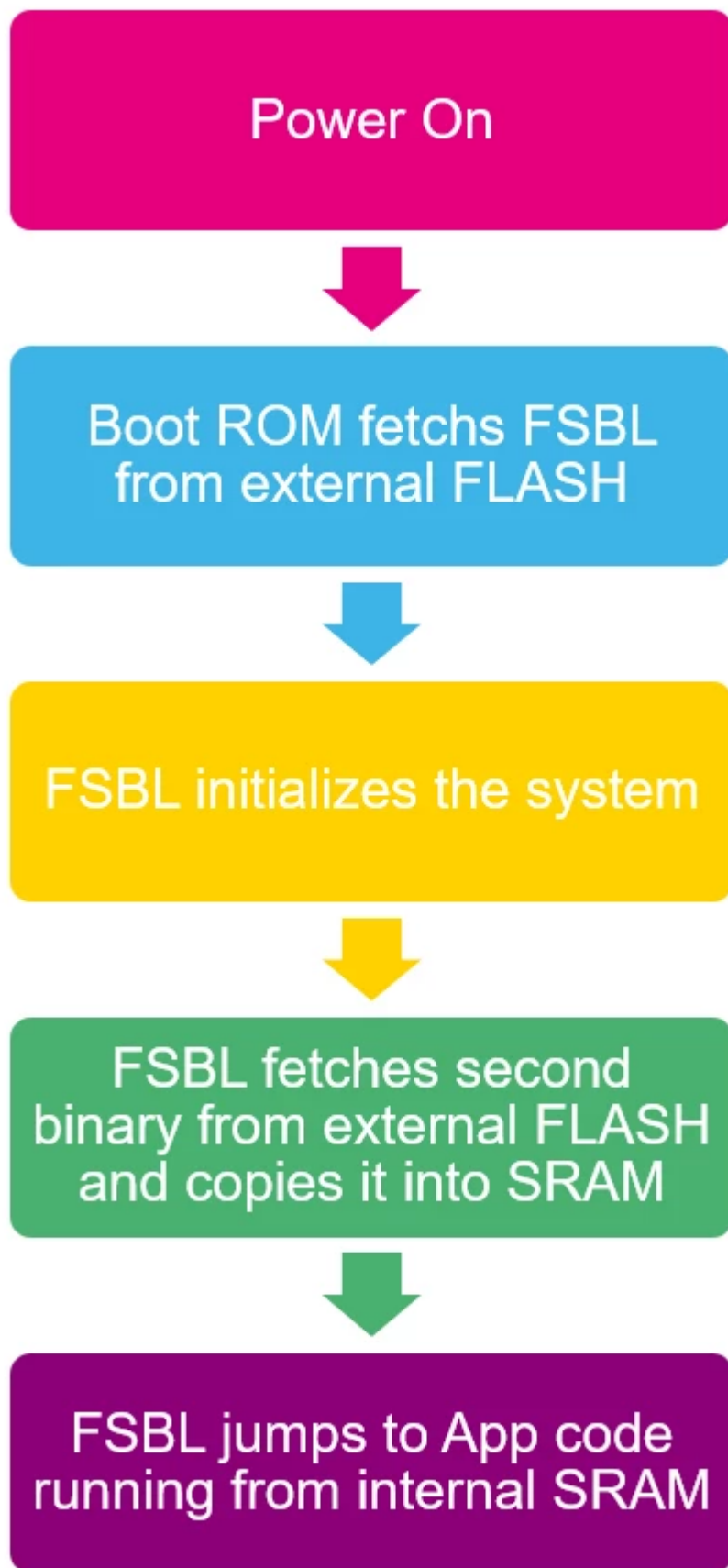
When in DEV mode and with some linker script adjustments, the programmer can load the FSBL code directly into the SRAM memory. It facilitates the debug process, as it skips the need to program the external memory.

2.2 FSBL with load and run application

In this mode, the boot ROM fetches the FSBL from external flash memory. The FSBL then fetches a second binary stored in an external flash and copies it into an internal SRAM. Once the binary is loaded, the FSBL jumps to the application code for execution. This mode is applicable to a few examples available in the STM32Cube_FW_N6. The interesting aspect is that the 511 KB size limitation is no longer applicable, as the user code can be placed in the remaining area of the internal RAM.

With some adjustments on the application side, it is even possible to have the copied FSBL at boot ROM time to be erase-overwritten to allow a contiguous firmware loading in SRAM. With such a config, the boot firmware load in SRAM is possible from flexRAM area up to SRAM7.

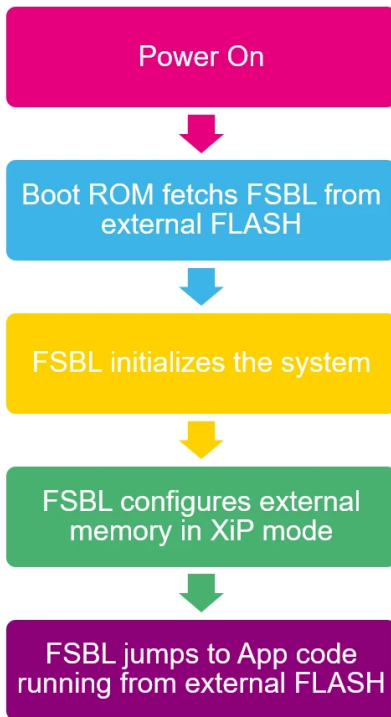




2.3 FSBL with execution in place (XiP)

In the XiP mode, the boot ROM fetches the FSBL from external flash memory. The FSBL configures the external memory in XiP mode and then jumps to the application code stored in external memory for execution. This mode is also applicable to a few examples.

 Ask AI



2.4 FSBL with secure and nonsecure applications

In this mode, same as all others, the boot ROM fetches the FSBL from external flash memory and copies the code into internal SRAM. The FSBL then fetches two binaries stored in external flash: one for the secure application and one for the nonsecure application. The FSBL copies both binaries into internal SRAM, each one in different address regions, and then jumps to the secure application for execution. This mode is very similar to the regular load and run.

2.5 FSBL with PSRAM and NOR

This mode is not a code example available as part of the regular offering in the driver. However, the same approach can be done to have the application code executing from PSRAM instead of the NOR. Once again, the boot ROM fetches the FSBL from external FLASH and copies its content to the internal RAM. From this point, the FSBL can map the two external memories, NOR, and PSRAM and have both configured in memory-mapped mode. This allows the FSBL to copy the application code from NOR to PSRAM.

Once the process is completed, the FSBL jumps to PSRAM, from where the application code will be executed. The main attention point needed in this case is to properly configure the MPU to ensure a proper copy.

3. FSBL with simplified load and run usage: UART demo

This article assumes you have installed STM32CubeMX, the latest version of the STM32N6 HAL driver and STM32CubeIDE. The hardware used to showcase is the STM32N6570-DK and make sure you have it in DEV boot mode:



Board Dev Boot configuration

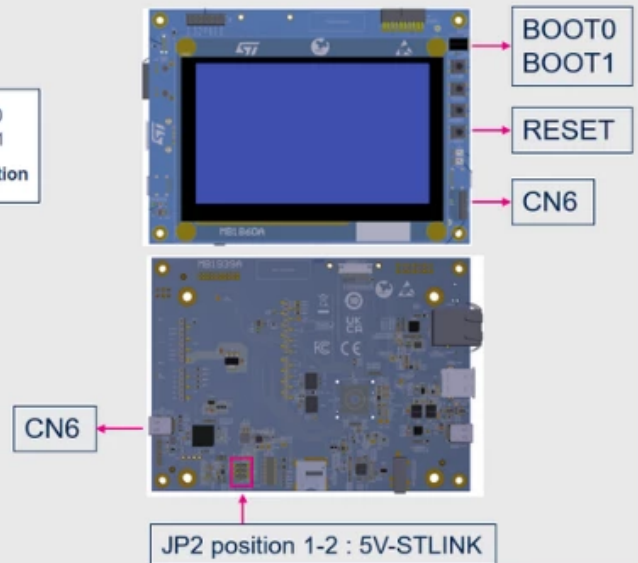
1. Disconnect board to User computer (CN6)

2. Set DEV BOOT configuration



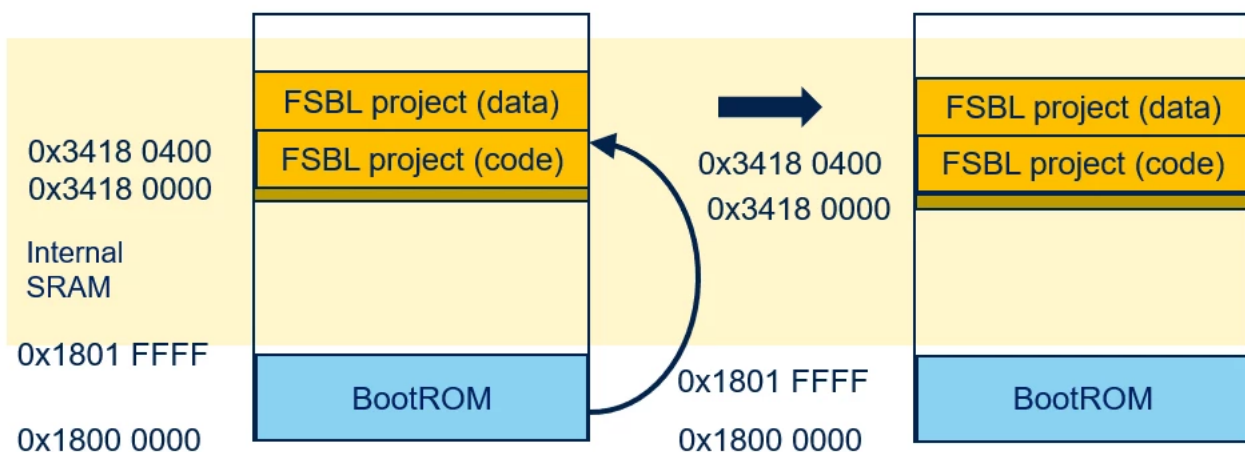
3. Set JP2 to position 1-2

4. Connect board to User computer (CN6)



Refer to the readme.html of the project, which will give you all the information. To summarize, the USART1 is configured in asynchronous mode with GPIO's PE5 (USART1_TX) and PE6 (USART1_RX) configured. PE5 and PE6 are connected to ST-LINK that acts as virtual COM port (VCP). The application counts the number of bytes that are received. Once it has received 10 bytes, it sets the LED1 ON and outputs a message **[Example Finished]**.

The overview is quite simple. Once the program loads the code, the boot ROM jumps to the internal RAM address. From there, the FSBL code acts as shown below:



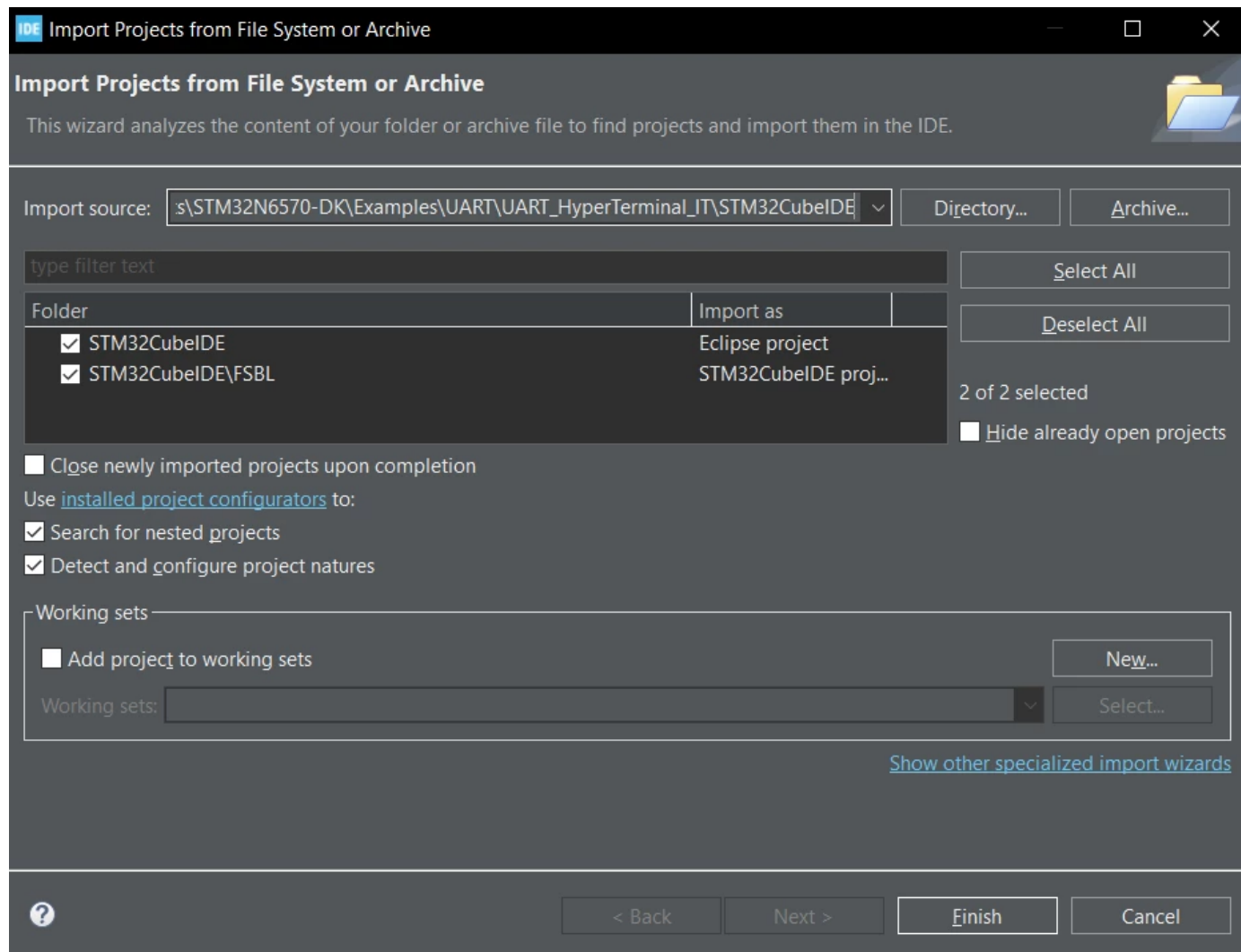
- 1) BootROM execution with DEV mode selected
- 2) FSBL project is programmed into the internal SRAM2 via SWD and Program Counter (PC) jumps to FSBL project first instruction location.

3) Application project execution

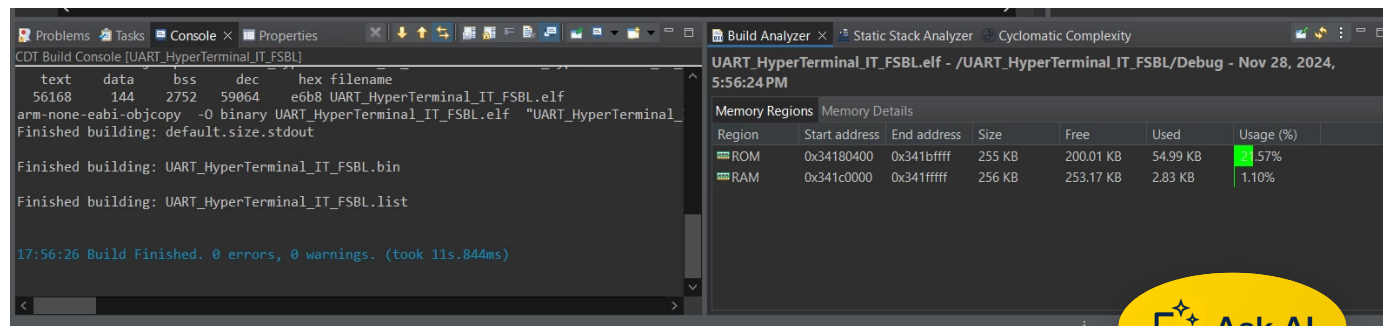


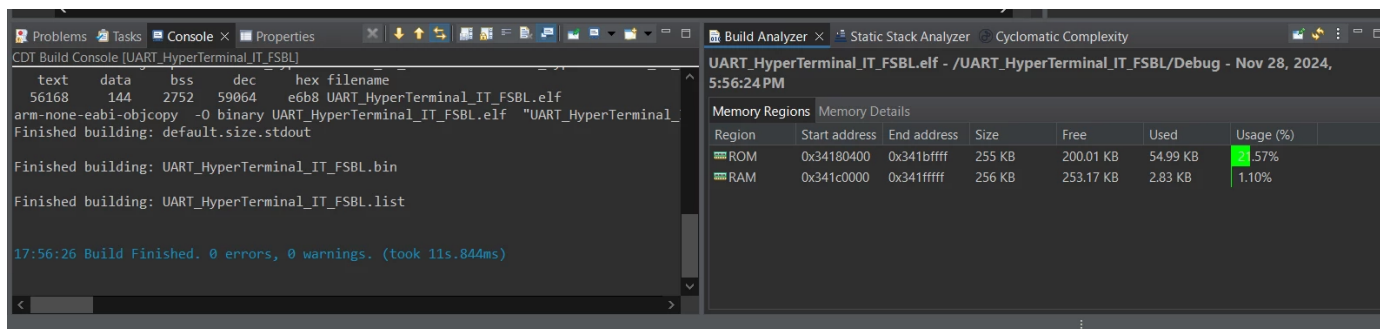
To get started, open STM32CubeIDE and import the example available in this folder

`C:\Users\%username%\STM32Cube\Repository\STM32Cube_FW_N6_V1.0.0\Projects\STM32N6570-DK\Examples\UART\UART_HyperTerminal_IT\STM32CubeIDE`

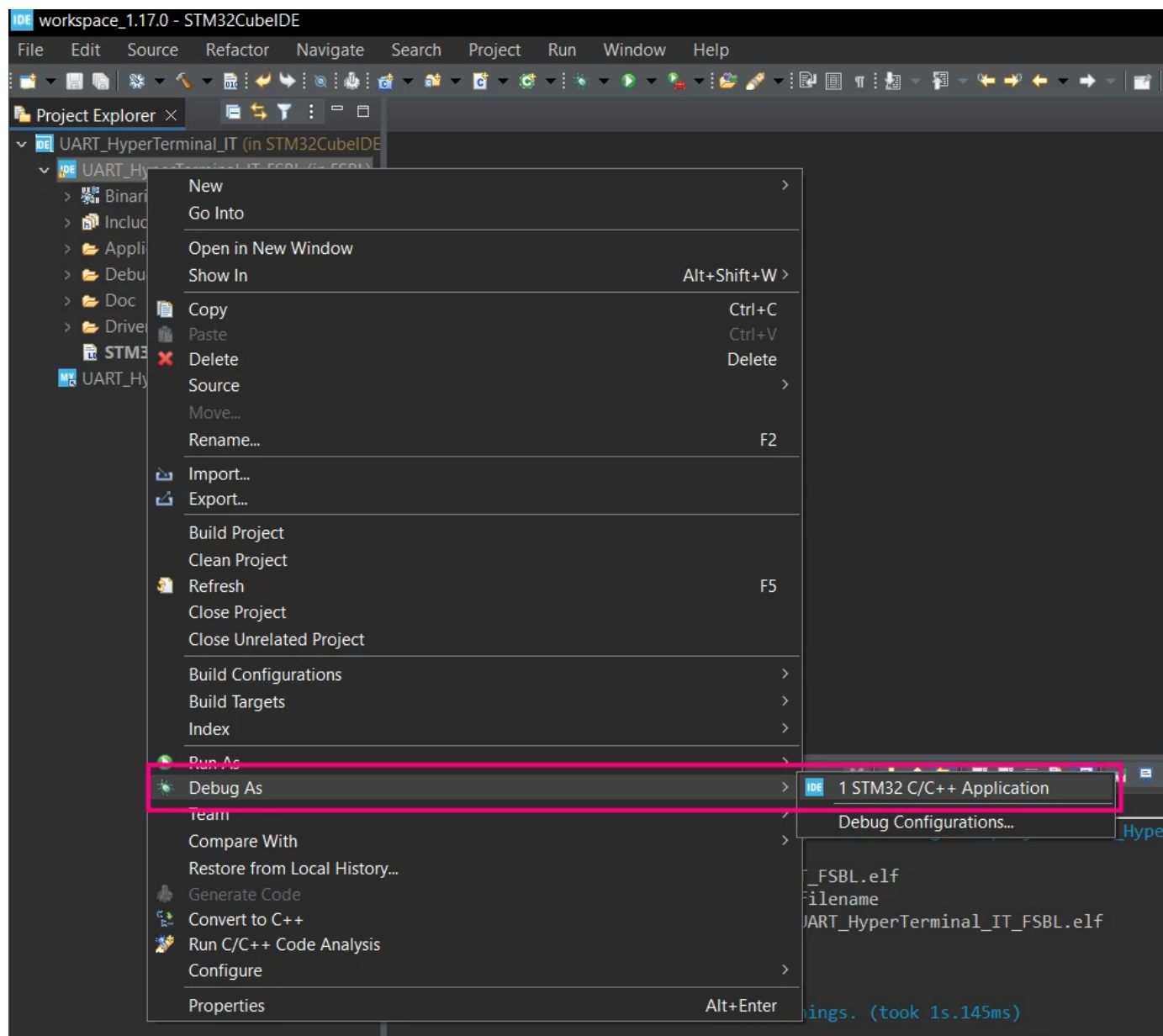


Once imported, it is worth noting that the linker script of the demo already is prepared to have the FSBL positioned in the AXSRAM2 region. It divides the 511 KB region into ROM (255 KB) and RAM (256 KB). From this point, just build the project and you are ready to debug and see it execute.

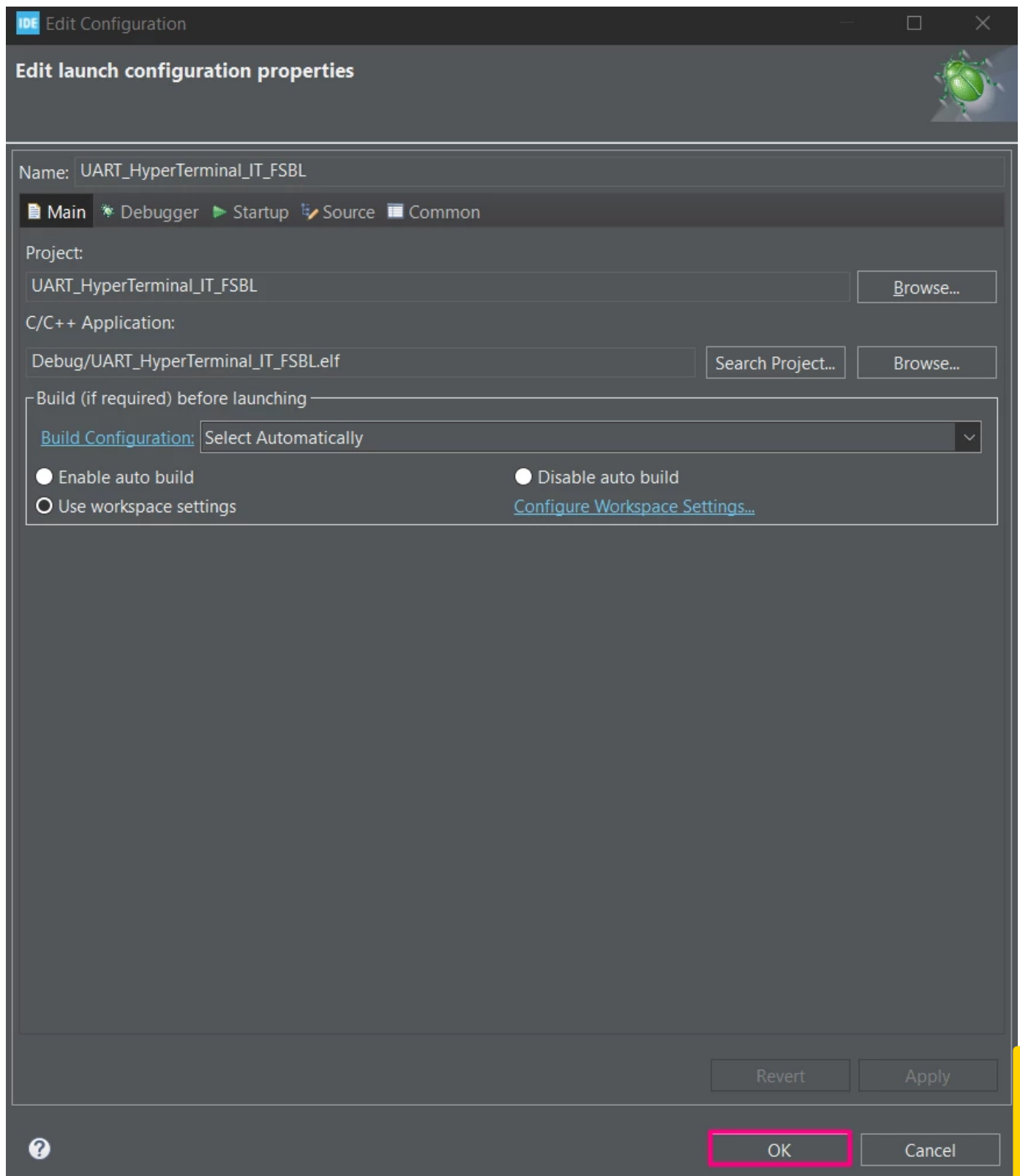




Press the NRST button to reset the device and ensure it is in DEV boot mode. Then select **[Debug As] [STM32 C/C++ Application]**. Keep the debugger with the default settings, and click **[OK]** in the launch configuration to load the firmware

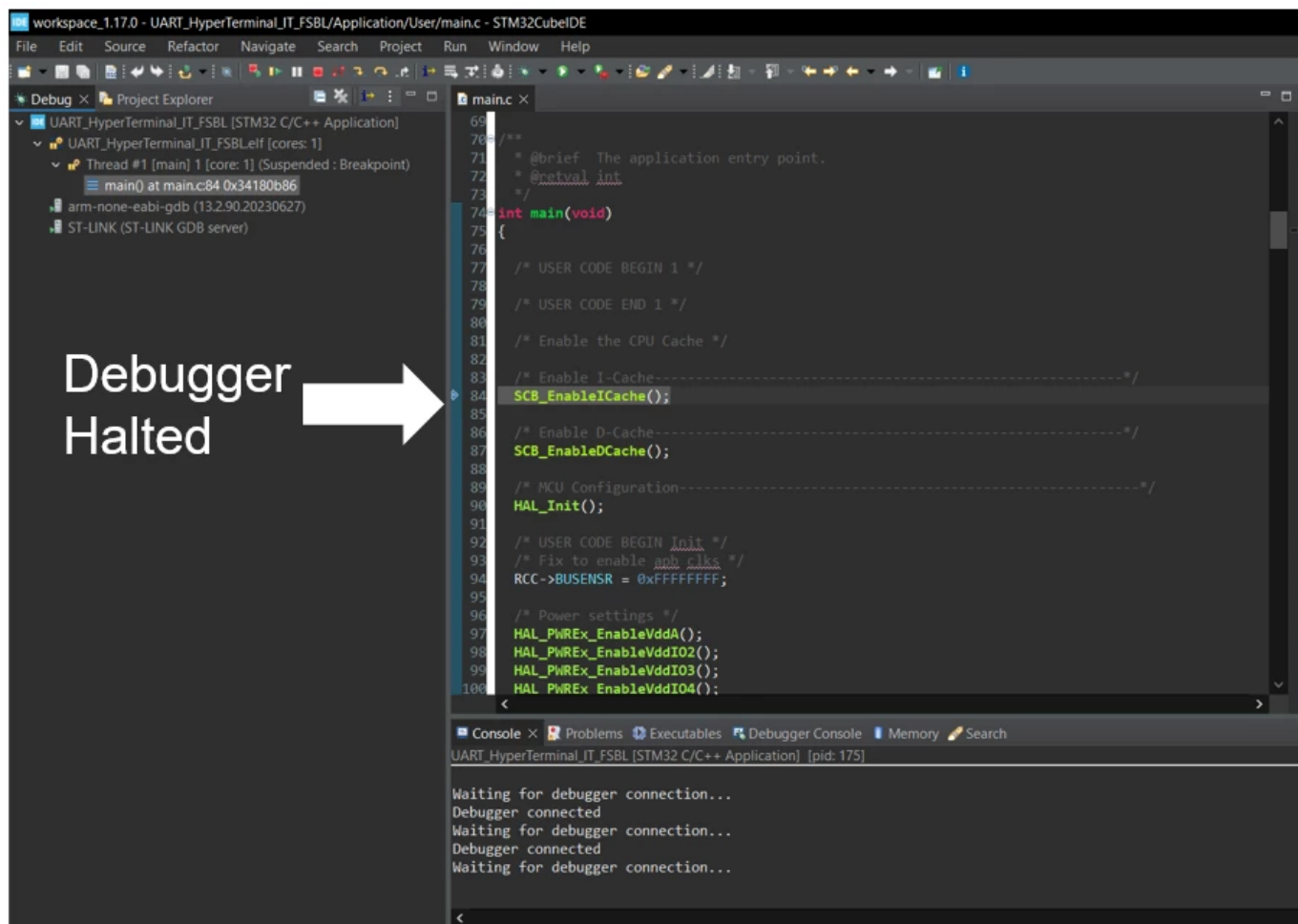


Ask AI

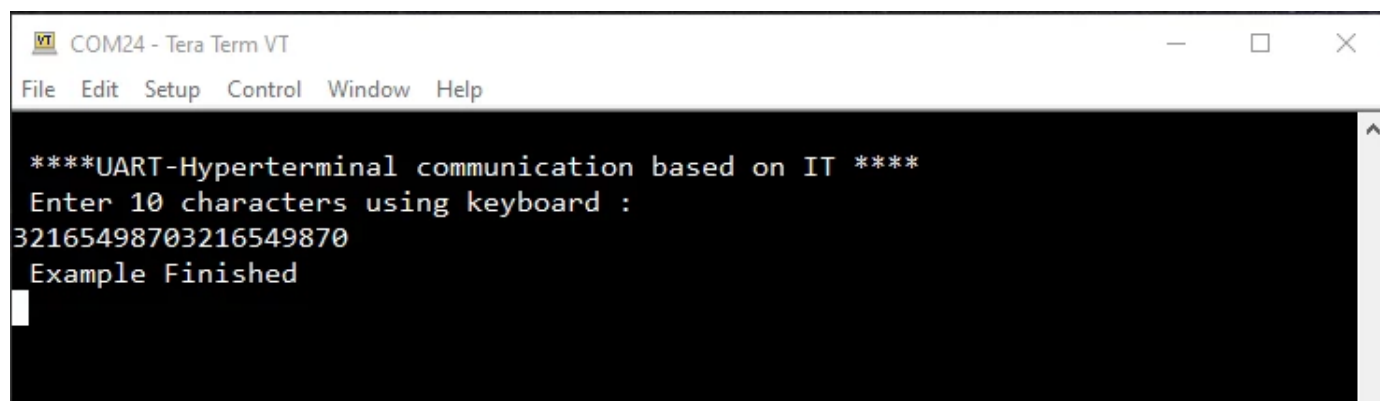


After establishing ST-LINK GDB server connection and programming, the internal SRAM2 STM32CubeIDE switches to the debugger perspective and halts the debugger at the beginning of the main() function.

 Ask AI



Before executing the code, make sure you have your preferred terminal these settings: 115200/7/Odd/1Stop/No Flow control. From this point, resume the firmware execution and type your 10 characters. LD1 stops blinking and is turns on statically once completed.



Conclusion

The FSBL is a versatile and essential component of the STM32N6 boot process. It supports various boot modes and configurations, ensuring a secure and reliable startup for different application requirements. By understanding the FSBL layout and its main features, developers can effectively utilize the FSBL to meet their specific needs.



Related links

- [User manual 3249: Getting started with STM32CubeN6 for STM32N6 series](#)
- [User manual 3234 How to proceed with boot ROM on STM32N6 MCUs](#)
- [User manual 3300: Discovery kit with STM32N657X0 MCU](#)

Bootloader

STM32N6 series

FSBL



5



Leave a reply...

ST Community highlights – April to June 2026



Related topics

How to add the STM32N6's header signature as post build

STM32 MCUs

STM32N657L0 NOR flash XIP/LRUN problem ✓

STM32 MCUs Products

First project on STM32N6750-DK ✓

STM32 MCUs Boards and hardware tools

STM32N657x0 Secure Region And Application Only Without FSBL in Keil. ✓

STM32 MCUs Security

How to understand the STM32N6 using STM32CubeMX ✓

STM32CubeMX (MCUs)

Popular tags

STM32H7 series

STM32F4 series

STM32CubeMX

STM32CubeIDE

TouchGFX

STM32L4 series

STM32F7 series

USB

UART-USART

STM32Cube MCUs package

Ask AI





About STMicroelectronics

Who we are

Investor relations

Sustainability

Innovation & technology

Careers

Suppliers to ST

Blog

General terms and conditions

Browse

Shortcuts

Sitemap

Connect with us

Contact ST offices

Find sales offices & distributors

Community

Newsroom

Events & trainings

AI at ST

Compliance, Ethics, Security & Privacy

Ethics and compliance

ST Ethics Hotline

ST's Code of Conduct

Security & privacy portal

Follow us



 Ask AI

All rights reserved © 2026 STMicroelectronics | [Terms of use](#) | [Sales terms & conditions](#) | [Trademarks](#) | [Privacy portal](#) | [Manage cookies](#)

