How to proceed with boot ROM on STM32N6 MCUs

Introduction

This document describes the boot ROM code for the STMicroelectronics Arm® Cortex®-M55 core-based STM32N6 microcontroller (MCU) cluster, mainly from a black-box perspective, and serves as a guideline for users of boot ROM code functionality.

It provides a description of how boot ROM code interacts with other components in the system, covering both hardware and software interfaces. It also describes the application boot image formats supported by the boot ROM code.

This document applies to the STM32N6 MCU devices. It focuses on the startup firmware located in the boot ROM of the STM32N6 MCU, which executes on the Arm® Cortex®-M55.

This specification follows a commonly used syntax to represent requirements: the key words "MUST," "MUST NOT," "REQUIRED," "SHOULD," "SHOULD NOT," "RECOMMENDED," "MAY," and "OPTIONAL" are interpreted as described in RFC 2119 [13].

- **MUST**: This word, or the terms "REQUIRED" or "SHALL," means that the definition is an absolute requirement of the specification.
- **MUST NOT**: This phrase, or the phrase "SHALL NOT," means that the definition is an absolute prohibition of the specification.
- **SHOULD**: This word, or the adjective "RECOMMENDED," means that there may exist valid reasons in particular circumstances to ignore a particular item. However, the full implications must be understood and carefully weighed before choosing a different course.
- **SHOULD NOT**: This phrase, or the phrase "NOT RECOMMENDED," means that there may exist valid reasons in particular circumstances when the particular behavior is acceptable or even useful. However, the full implications must be understood and the case carefully weighed before implementing any behavior described with this label.
- **MAY**: This word, or the adjective "OPTIONAL," means that an item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because the vendor feels that it enhances the product, while another vendor may omit the same item. An implementation that does not include a particular option must be prepared to interoperate with another implementation that does include the option, though perhaps with reduced functionality. Similarly, an implementation that does include a particular option must be prepared to interoperate with another implementation that does not include the option, except, of course, for the feature that the option provides.

1 General information

This document applies to STM32N6 Arm®-based MCUs.

STM32N6 MCUs are microcontrollers that are based on the Arm® Cortex®-M55.

Note: Arm and Cortex are registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

arm

Table 1. Referenced documents

N°	Description
[1]	Reference manual STM32N647/657xx Arm®-based 32-bit MCUs (RM0486)
[2]	SD Specifications Part 1 Physical Layer Simplified Specification version 6.00

Table 2. Glossary

Acronyms	Meanings
Bootloader	Software that loads and runs some other software, typically an operating system.
Boot ROM	Boot firmware stored in Cortex®-M55 ROM, also named ROM code.
BSEC	Boot and security block: it is used to manipulate OTP fuse words.
CRYP	Cryptography
CM55	Cortex®-M55
CPvK	CUK private key
CUK	Chip unique key
E1CPvK	TK1 encrypted chip private key
ECC	Error correcting code
ECDSA	Elliptic curve digital signature algorithm
FSBL	First stage bootloader: name of the binary loaded by the boot ROM code.
HDPL	Hardware protection level
MBR	Master boot record
MCU	Microcontroller unit
OTP	One-time programmable fuse bits
RHUK	The root hardware unique key: nonvolatile hardware secret unique per device
UART	Universal asynchronous receiver transmitter
USB HS	USB high-speed profile at 480 Mbps

2 Description

2.1 Boot ROM features and functions

The boot ROM code is the initial code executed on Arm® Cortex®-M55 at power-on or reset of the STM32N6 MCU or Arm® Cortex®-M55. This boot ROM code resides in the STM32N6 on-chip boot ROM IP and typically implements the first stage of a multistage boot sequence.

The main boot ROM code features and functions are:

- Basic system initialization
- Detection of reset source, specific reset condition, and chip mode
- Bootstrapping from an attached boot memory device supporting various types of memory devices
- Downloading code over serial boot interfaces and jumping to the downloaded code
- Implementation of USB 2.0 device according to USB 2.0 HS supporting DFU 1.1
- Handling life cycle
- Validation of signed images using hardware accelerators for cryptographic functions
- Support of configuration options (customization), mainly via fuses
- Support of ST key provisioning
- Support of SSP, OEM key provisioning
- Support of blocking failure processing

See also the following subchapters for further details.

2.1.1 Applicable reset types

The boot ROM code placed in the STM32N6 MCU boot ROM function is generally executed whenever the Arm® Cortex®-M55 in the STM32N6 MCU is released from reset. Depending on the detected type of reset, different branches within the boot ROM code implementation are executed.

The following logical reset types are applicable and distinguished by the boot ROM code:

- SYSTEM: Logical system reset
- ST_KEY_PROVISIONING: Logical reset of the ST key provisioning stage condition with SFT and PIN resets

2.1.2 Supported boot memory devices

The boot memory device is the external flash memory device attached to the STM32N6 MCU on which the first stage bootloader (FSBL) is located. The boot ROM code first loads the FSBL into the internal RAM of the STM32N6 MCU and analyze it from there.

The boot ROM code supports the following types of boot memory devices:

- sNOR x4 and x8 flash devices
- HyperFlash™ Flash device
- SD memory card device (SD specifications V6.0)
- eMMC type of embedded memory card devices (eMMC specifications V5.1)

2.1.3 Supported serial boot interfaces

The boot ROM code provides functionality for downloading code over a serial boot interface into the internal RAM of the STM32N6 MCU. Typically, downloading code over a serial boot interface is used to update the FSBL stored on a flash-type attached boot memory device.

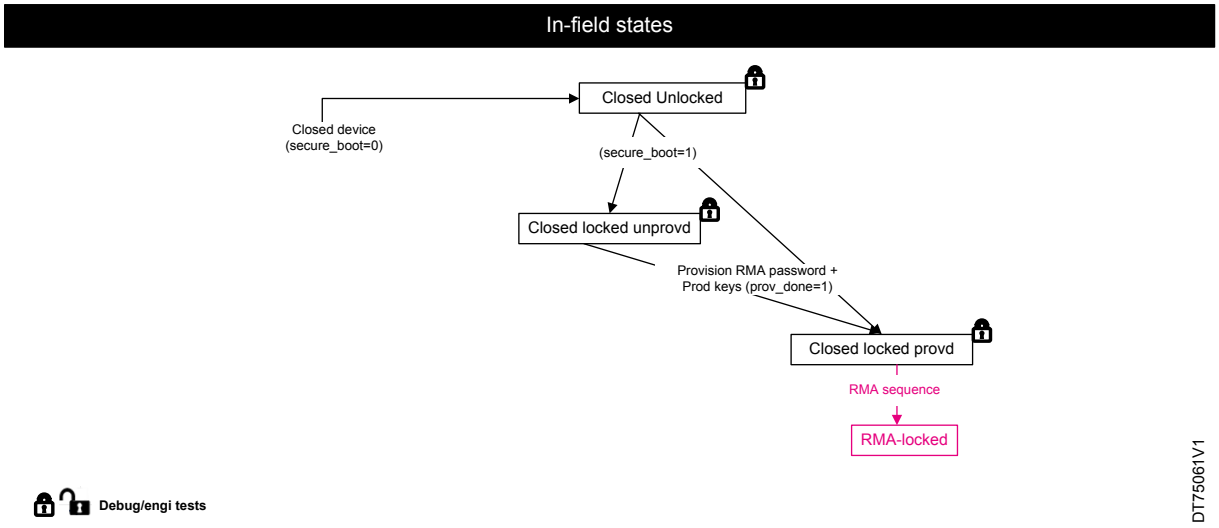
The boot ROM code supports the following types of serial boot interfaces:

- USART type of serial interface using three different USART hardware function instances
- USB interface (USB 2.0 HS)

2.1.4 Supported life cycle

The life cycle defines the states that determine the behavior and available features of the STM32N6 MCU device during the different steps of its life. The value of fuses determines the life cycle state. The BSEC function controls these fuses. Figure 1 shows the STM32N6 MCU life cycle flow.

Figure 1. STM32N6 life cycle



The STM32N6 chip life cycles are described in the STM32N6 security architecture document. The STM32N6 chip life cycles are set through fuses. The detailed fuse configuration to select the STM32N6 life cycle is described in Section 3.2.6 and the fuses handled by the boot ROM are listed in Section 3.11.1.

The boot ROM code follows this life cycle to provide the expected features according to the life cycle state, as described in Table 3.

Table 3. Boot ROM code scenario following the STM32N6 life cycle.

STM32N6 life cycle state	ST-FSBL secure boot	OEM-FSBL secure boot	Development boot	ST key provisioning
CLOSED/UNLOCKED	o	x (1)	x	o
CLOSED/LOCKED/ UNPROVD	x (2)	o	o	o
CLOSED/LOCKED/ PROVD	o	x (2)	o	o

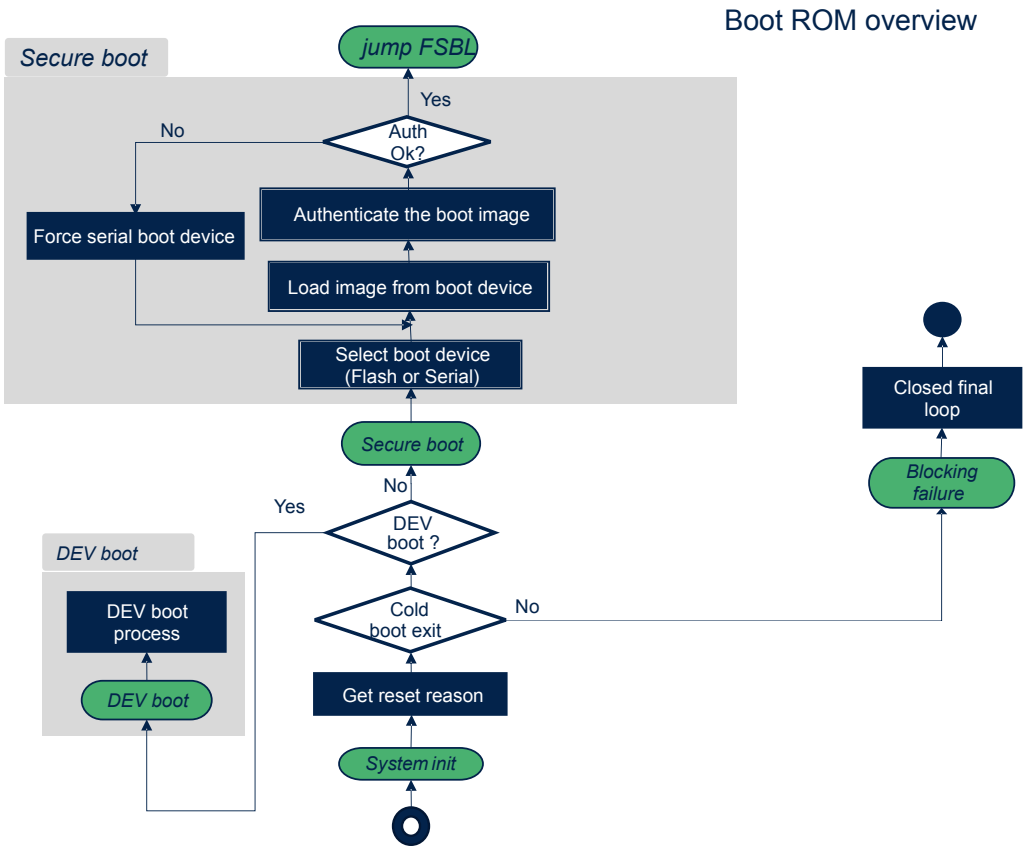
1. *Authentication is not mandatory.*
2. *Secure boot enforced: authentication is mandatory.*

The ST-RSSE-FW (formerly named ST-FSBL) and OEM-FSBL secure boot use cases are also detailed in the following sections.

2.2 Boot ROM flow diagram

Figure 2 depicts the overall boot ROM code flow diagram, showing the main activities and branches implemented in the STM32N6 boot ROM code. More detailed descriptions are provided in the subsequent chapters of this document.

Figure 2. Boot ROM code flow diagram

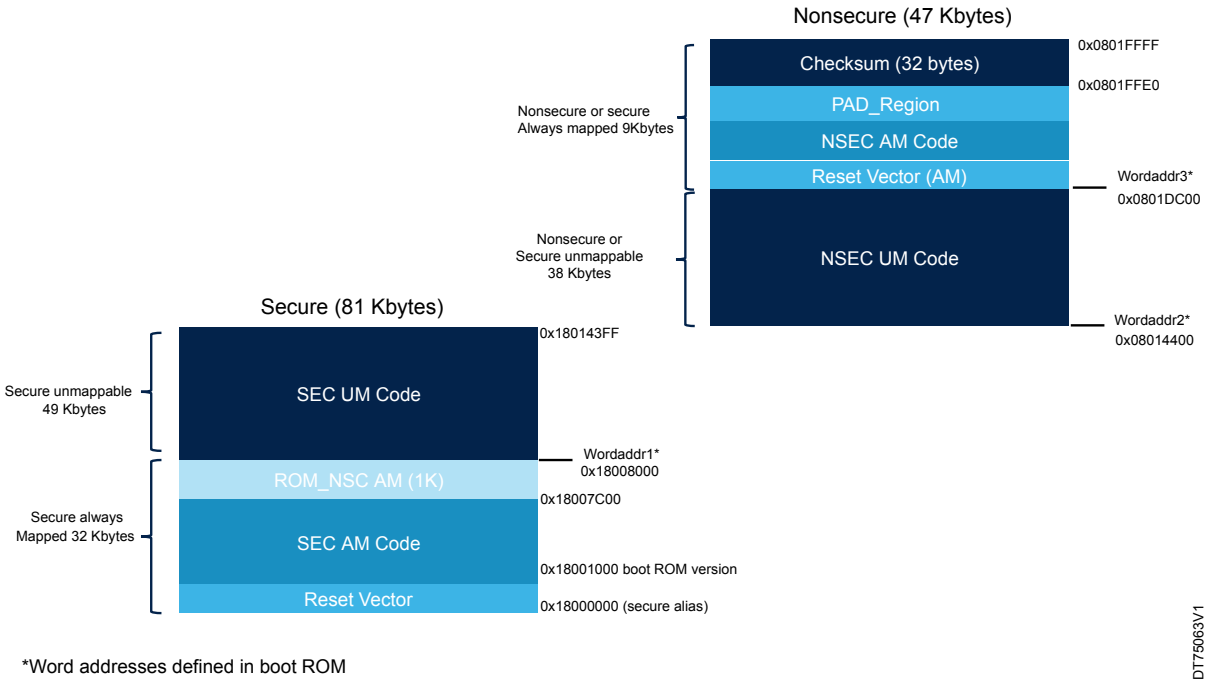


3 Interfaces

3.1 Boot ROM hardware function

Figure 3 shows the layout of the boot ROM function of the STM32N6 device. In the boot ROM hardware function, there are three defined word address boundaries that allow the definition of SEC and NSEC areas, as well as always mapped (AM) and unmappable areas (UM).

Figure 3. Boot ROM function layout



*Word addresses defined in boot ROM

3.2 Initialization steps and evaluation of reset condition and boot scenario

3.2.1 Exception vector setting

In the Arm® Cortex®-M55 core, a vector table allows handling exception handlers, with one defined for each secure and nonsecure area. An undefined exception handler results in a blocking failure scenario, which is detailed in a specific chapter of this document. Index 0 of the vector table holds the address of the Main Stack Pointer (MSP). The next index is the ResetHandler vector. The start of the vector table is executed when the reset of the Arm® Cortex®-M55 core is released. It means that it sets the MSP and then executes the ResetHandler exception.

3.2.2 TrustZone® protection, RISAF, and cache handling

The boot ROM code implements TrustZone® protection, RISAF, and a cache handling mechanism. Table 4 shows the state of the security mechanisms depending on the boot ROM scenario executed.

Table 4. TrustZone® support setting following boot ROM scenario

STM32N6 life cycle state	Boot ROM scenario	SAU/MPU	RISAF	Cache
CLOSED-UNLOCKED	DEV Boot	Disabled SAU and reset MPU settings	Clear RISAF3 settings	Disable cache
CLOSED-UNLOCKED	OEM FSBL	Disabled SAU and reset MPU settings	Clear RISAF3 settings	Keep ICACHE enabled

STM32N6 life cycle state	Boot ROM scenario	SAU/MPU	RISAF	Cache
CLOSED-LOCKED-PROVD	OEM FSBL	Disable SAU and MPU setting update(1)	Clear RISAF3 settings	Keep ICACHE enabled
CLOSED-LOCKED-UNPROVD	ST boot extension	Disabled SAU and MPU setting update(1)	Clear RISAF3 settings	Keep ICACHE enabled
All life cycles	Blocking failure	SAU and MPU settings kept	RISAF3 settings to allow only SEC accesses	Disable ICACHE, disable DCACHE if secure boot

1. *If required, setunexecutable na payload area.*

3.2.2.1 *IDAU / SAU / MPU*

An external implementation defined attribution unit (IDAU) controlled by hardware determines the security state of a memory region. In combination with the IDAU, the boot ROM code defines eight regions using the internal secure attribution unit (SAU). The SAU is programmable in the secure state. Table 5 shows the boot ROM code configuration settings of the SAU.

Note: When a region is not defined, it is marked as secure by default. When the SAU and IDAU provide different settings for the security attribute, the most conservative one is selected. Additionally, the SAU region granularity is 32 bytes.

Table 5. SAU Region Address Mapping and Security Attributes

SAU region	Address range	Security attribute	Mapping
0	0x08010000 0x0801FFFF	Nonsecure	NSEC ROM
1	0x18007C00 0x18007FFF	Secure, nonsecure-callable	NSC ROM
2	0x20000000 0x24101FFF	Nonsecure	DTCM, SRAM1, SRAM2 until secure boot ROM data
3	0x24104000 0x2FFFFFFF	Nonsecure	SRAM2 nonsecure boot ROM data, reserved area, download buffer, SRAM3, and other SRAMs
4	0x30000000 0x340FFFFF	Secure, nonsecure-callable	DTCM, SRAM1
5	0x34106000 0x3FFFFFFF	Secure, nonsecure-callable	SRAM2 reserved area, download buffer, and other SRAMS
6	0x40000000 0x4FFFFFFF	Nonsecure	Nonsecure aliasing peripherals
7	0x60000000 0xFFFFFFFF	Nonsecure	Nonsecure external devices and core peripherals

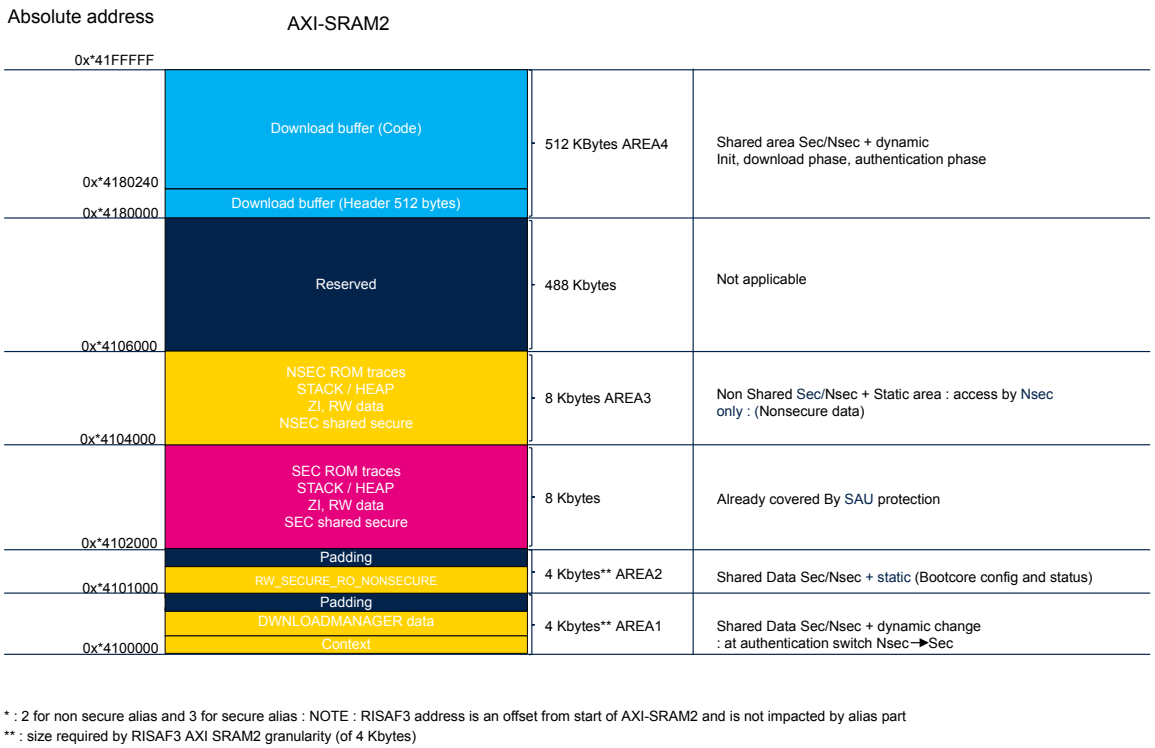
The boot ROM code also sets the MPU_S (from secure) to segment memory regions in AXI SRAM2, primarily to restrict execution rights in selected regions. Through MPU settings, execution is allowed only in the region containing the authenticated binary, thereby excluding the nonauthenticated payload. See details on the nonauthenticated payload in Section 4.2.

This implies that the FSBL code must be aligned on 32-byte addresses (start and end addresses).

3.2.2.2 *RISAF handling*

The boot ROM code handles all its data in AXI SRAM2 internal RAM. The AXI SRAM2 layout is detailed in Section 3.9.2. To protect and isolate its data, the boot ROM code configures the RISAF3 hardware functional block, dedicated to RISAF for AXI SRAM2 internal memory. The boot ROM code configures seven regions inside RISAF3, as described in Figure 4, and dynamically sets each region depending on the boot ROM code phase being executed.

Figure 4. RISAF3 region configuration



* : 2 for non secure alias and 3 for secure alias : NOTE : RISAF3 address is an offset from start of AXI-SRAM2 and is not impacted by alias part
** : size required by RISAF3 AXI SRAM2 granularity (of 4 Kbytes)

3.2.2.3 *Cache handling*

The boot ROM code enables the instruction cache in a secure state in the early stage of the boot flow. In addition, the boot ROM code enables data cache during secure boot execution to speed up data access during authentication and decryption processing. At the end of secure boot execution, the boot ROM code cleans and invalidates the data cache.

Before jumping to the payload code, the boot ROM code configures the caches as described in Table 4.

3.2.3 Reset condition determination

The boot ROM code evaluates which reset source has caused the system to restart. The source of a reset is determined by reading the RCC_HWRSR register. Then, it requests the clearing of this register to avoid accumulating reset reasons. Next, it evaluates the setting of relevant bits.

The following reset sources are applicable for and distinguished by the boot ROM code:

- **PIN**: Pin reset of STM32N6xx MCU triggered by NRST pin (external reset)
- **POR**: Power-on reset (cold) of STM32N6xx MCU triggered by power-on/off reset block
- **BOR**: Brownout reset of STM32N6xx triggered by brownout reset block
- **SFT**: Software reset of STM32N6xx triggered by the Arm® Cortex®-M55
- **WWDG**: Windows watchdog reset of STM32N6xx triggered by internal window watchdog timer
- **IWDG**: Independent watchdog reset of STM32N6xx triggered by internal independent watchdog timer
- **LPWRILL**: Low-power mode security reset of STM32N6xx triggered by internal power signal

According to the bit settings, reset types are defined as follows:

Table 6. Reset source analysis and reset type selection

PIN RST	POR RST	BOR RST	SFT RST	WWDG RST	IWDG RST	LPWRILL RST	Reset type
1	0	0	0	0	0	0	System reset
0	1	0	0	0	0	0	System reset
0	0	1	0	0	0	0	System reset
0	0	0	1	0	0	0	System reset
0	0	0	0	1	0	0	System reset
0	0	0	0	0	1	0	System reset
0	0	0	0	0	0	1	System reset

3.2.4 Watchdog timer handling

The boot ROM code reloads the independent watchdog (IWDG) if a fuse enables the IWDG. Blowing a fuse (OTP_WORD124.bit0) starts the IWDG on reset.

3.2.5 Tamper determination

The reference manual [1] details the tampered function and types of tampers. An application can define two types of tampers:

- **Potential tamper**: This tamper blocks some strategic resources (crypto blocks and some memories) but does not erase them immediately.
- **Confirmed tamper**: This tamper automatically erases secrets and keeps crypto blocks in reset.

In the case of a potential tamper, the boot ROM code activates protection against the tamper effect, that is, unblocking crypto blocks at the beginning of its execution. This allows the loading and execution of the FSBL. Before jumping to the FSBL or in a blocking failure scenario, the boot ROM code deactivates the protection. The application decides to filter the potential tamper and eventually confirm it. In the case of a confirmed tamper, the boot ROM code ends in an infinite loop.

Additionally, the boot ROM code handles an added tamper mechanism called boot tamper. In this case, the tampers are configured through fuses and are applicable during the boot process. The following OTPs support this feature:

- **OTP_WORD29.1**: Enables the configuration of tampers in boot ROM.
- **OTP_WORD56.[0..7]**: Enables external tamper from 1 to 8 (TAMP1 to TAMP8).
- **OTP_WORD56.[8..18]**: Enables internal tamper from 1 to 11 (ITAMP1 to ITAMP11, excluding ITAMP10).
- **OTP_WORD57.[0..7]**: Configures TAMP1 to TAMP8 as either confirmed or potential tamper.
- **OTP_WORD57.[8..18]**: Configures ITAMP1 to ITAMP11 (excluding ITAMP10) as either confirmed or potential tamper.
- **OTP_WORD58.[0..7]**: Configures the mode level (0: low, 1: high) of each external tamper from TAMP1 to TAMP8.

For more information, see Section 3.11.1 that details the OTP list used by the boot ROM code.

3.2.6 Life cycle level determination

The boot ROM code determines the applied life cycle level at an early stage of the boot flow. It is based on fuses and is detailed in Table 7.

Table 7. Life cycle fuse configuration

STM32N6 life cycle State	Fuse configuration
CLOSED_LOCKED_UNPROVD	CLOSED_UNLOCKED fuse configuration + OTP_WORD124.20 = 1 (DFT_disable bit) + OTP_WORD18 = 0xF (secure_boot)
CLOSED_LOCKED_PROVD	CLOSED_UNLOCKED fuse configuration + OTP_WORD124.20 = 1 (DFT_disable bit) + OTP_WORD18 = 0x1EF (secure_boot and prov_done)

3.2.7 Boot ROM configuration determination

The boot ROM code evaluates the selected boot configuration. The boot configuration defines which type of external boot memory device or serial boot is selected, or if dev boot is applicable. The boot configuration is either specified using external pins or by fuses.

The following categories of external boot memory devices are supported:

- sNOR x4 and x8 flash devices
- HyperFlash™ flash device
- SD memory card device (SD specifications V6.0)
- eMMC type of embedded memory card devices (eMMC specifications V5.1)

Table 8. Supported boot configurations

Boot Config.	Attached boot	Applied boot memory setup	Description
0	Dev boot	-	Dev boot mode selected in CLOSED-UNLOCKED life cycle
1	Serial boot	-	Serial boot monitors in parallel USB and USART links
2	SD device	SD1	SD device, connected to SDMMC1
3	SD device	SD2	SD device, connected to SDMMC2
4	eMMC device	eMMC1	eMMC device, connected to SDMMC1
5	eMMC device	eMMC2	eMMC device, connected to SDMMC2
6	sNOR device	XSPI NOR	sNOR device, connected to XSPIM_P2
7	HyperFlash™	XSPI HYPER	HyperFlash™ device, connected to XSPIM_P2

The boot ROM code evaluates the register bits BOOTSR[0:1] to determine the applied boot configuration. The register bits BOOTSR[0:1] reflect the level of the external boot pins as latched at reset.

Allocated pins for BOOTSR[0:1]:

- **BOOTSR[0]**: Boot0 pin (dedicated pin)
- **BOOTSR[1]**: Boot1 pin (nondedicated pin, PA6)

Note: The selected Boot1 pin, PA6, can be overwritten by fuses, BOOTROM_CONFIG_10[24:21] (port ID) and BOOTROM_CONFIG_10[28:25] (pin ID). The Boot1 pin check has priority over the Boot0 pin check. If the Boot1 pin is not set, the Boot0 pin is checked. If the Boot1 pin is selected but not allowed in the current life cycle, the Boot0 pin is then checked.

The flash boot configuration is checked using fuses in BOOTROM_CONFIG_2[8:5], OTP_WORD11. The boot configuration coding rules are defined in Table 9.

Table 9. Boot configuration coding

BOOTSR[1:0]	BOOTROM_CONFIG_2[8:5]	Applied boot configuration
0	1	Boot config. 2
0	2	Boot config. 4
0	3	Boot config. 6
0	5	Boot config. 7
0	7	Boot config. 3
0	8	Boot config. 5
0	0	Boot config. 6 (default configuration)
1	X	Boot config. 1
2	X	Boot config. 0

BOOTSR[1:0]	BOOTROM_CONFIG_2[8:5]	Applied boot configuration
3	X	Boot config. 0

The boot ROM code must configure STM32N6 on-chip hardware blocks to enable access to external boot memory or serial links. Table 10 specifies the required on-chip hardware blocks for the supported external boot memory or serial link categories.

Table 10. Hardware blocks required per boot category

Boot category	Required hardware blocks
Serial boot	USB OTG1, USART1, USART2, UART4
SD1	SDMMC1
eMMC1	SDMMC1
SD2	SDMMC2
eMMC2	SDMMC2
XSPI NOR	XSPIM_P2, XSPI1
XSPI HyperFlash™	XSPIM_P2, XSPI1

3.2.8 Usage of hardware timers

The boot ROM code supports timeout and timestamp functions. To enable these features, the boot ROM code configures the TIMER2 hardware block function. The timer module base is set to 1 MHz using the hardware timer prescaler.

3.3 Execution of special boot branches

3.3.1 Dev boot execution

The dev boot mode is executed only in the CLOSED_UNLOCKED life cycle.

In this scenario, the boot ROM code protects its assets, reopens the debug secure and nonsecure, and then goes into an endless loop. The dev boot mode is selected using the Boot1 pin.

3.3.2 Blocking failure execution

The blocking failure scenario occurs during boot ROM code execution if an error happens (for example, exception, invalid scenario, invalid parameters). In this scenario, the boot ROM code:

- Clears and locks sensitive data
- Clears the download area
- Locks the debug
- Switches all GPIO to secure
- Sends UART status traces using the PG10 BootFailed pin (status values are described in Section 3.11.2)
- Sets the PG10 LED on

As described below, the PG10 (AF11) is used as the BootFailed pin. This pin is multiplexed to UART5_TX to send UART status traces.

3.4 Boot ROM memory device setup

3.4.1 SDMMC hardware block configuration for SD device

To access an SD card device as a boot memory device attached to the STM32N6 MCU, the boot ROM code configures the SDMMC1 or SDMMC2 hardware blocks. Both SDMMC controller instances have the same configuration settings.

The selection of the SD card device as a boot memory device is done by the applied boot configuration. The SDMMC host controller hardware block is configured for data transfer from the SD card device according to Table 11 before starting the actual boot operation. The SD boot operation implemented in the boot ROM code is described in the following sections. Registers and register fields not explicitly mentioned are not relevant for SD boot operation and can retain their default settings.

Table 11. SDMMC settings for SD card identification mode

Register	Register field	Applied configuration setting
SDMMC_CLKCR	NEGEDGE	0x0: Rising edge, SDMMC_CK dephasing for data and command
	WIDBUS	Default 1-bit wide bus mode: SDMMC_D0 used (does not support DDR)
	PWRSAV	0x0: SDMMC_CK clock is always enabled
	HWFC_EN	0x0: Hardware flow control is disabled
	BUSSPEED	0x0: DS, HS speed mode selected
	CLKDIV	0x84 for SD card identification mode 0x2 for SD card transfer mode: Clock divide factor = sdmmc_ker_ck / (2*SDMMC_CK) with • sdmmc_ker_ck = 64 MHz • sdmmc_ck – = 177 kHz for SD card identification or – = 16 MHz for SD card transfer mode
SDMMC_POWER	DIRPOL	0x1: Voltage transceiver I/Os driven as output when the direction signal is high
	PWRCTRL	0x3: Power on, the card is clocked

In addition to configuring the actual SDMMC hardware block, the boot ROM code also carries out the I/O settings, the power domain settings, and the clock settings dedicated to the SDMMC hardware clock.

3.4.2 Configuration settings of SD device

The SD card is addressed by sector. The boot ROM code is forcing the sector size to 512 bytes.

3.4.3 SDMMC hardware block configuration for eMMC device

To access an eMMC device as a boot memory device attached to the STM32N6 MCU, the boot ROM code configures the SDMMC1 or SDMMC2 hardware blocks. Both SDMMC controller instances have the same configuration settings.

The selection of the eMMC device as a boot memory device is done by the applied boot configuration described in Section 3.2.7: Boot ROM configuration determination. The SDMMC host controller hardware block is configured for data transfer from the eMMC device according to Table 12 before starting the actual boot operation. The SD boot operation implemented in the boot ROM code is described in Section 3.5.1: Access image on SD card device. Registers and register fields not explicitly mentioned are not relevant for SD boot operation and can retain their default settings.

Table 12. SDMMC settings for eMMC boot mode

Register	Register field	Applied configuration setting
SDMMC_CLKCR	NEGEDGE	0x0: Rising edge, SDMMC_CK dephasing for data and command
	WIDBUS	Default 1-bit wide bus mode: SDMMC_D0 used (does not support DDR)
	PWRSAV	0x0: SDMMC_CK clock is always enabled
	HWFC_EN	0x0: Hardware flow control is disabled
	BUSSPEED	0x0: DS, HS speed mode selected
	CLKDIV	0x2 Clock divide factor = sdmmc_ker_ck / (2*SDMMC_CK) with • sdmmc_ker_ck = 64 MHz • sdmmc_ck = 16 MHz for eMMC transfer mode
SDMMC_POWER	DIRPOL	0x0: Voltage transceiver I/Os driven as output when the direction signal is low
	PWRCTRL	0x3: Power on, the eMMC is clocked

In addition to configuring the actual SDMMC hardware block, the boot ROM code also carries out the I/O settings (see Section 3.12.3), the power domain settings (see Section 3.12.2), and the clock settings dedicated to the SDMMC hardware clock (see Section 3.12.1).

3.4.4 Configuration settings of eMMC device

For the implemented eMMC boot mode, the boot-related configuration settings on an eMMC device, as shown in Table 13, are expected to be in place.

Table 13. Required boot settings on eMMC device

eMMC register	Register field	Applied configuration setting
EXT_CSD[179]	BOOT_PARTITION_ENABLE	Boot partition 1 enabled for boot
EXT_CSD	BOOT_BUS_WIDTH	EMMC1: x1 bus width in boot operation mode (default setting)

3.4.5 XSPI / XSPIM configuration for serial NOR device

To access a serial NOR device as a boot memory device attached to the STM32N6 MCU, the boot ROM code configures the XSPI1 controller and the XSPIM controller to use the XSPIM_P2 dedicated port for the Flash device.

The selection of the serial NOR device as a boot memory device is done by the applied boot configuration. The XSPI1 controller hardware block is configured for data transfer from the NOR device according to Table 14 before starting the actual boot operation. The serial NOR boot operation implemented in the boot ROM code is described in the Section 3.4.6. Registers and register fields not explicitly mentioned are not relevant for serial NOR boot operation and can retain their default settings.

Table 14. XSPI1/XSPIM settings for sNOR device

Register	Register field	Applied configuration setting
XSPI1_DCR2	Prescaler	0x1: fclock = fkernel_clock / 2
XSPI1_DCR1	CSHT	0x0: NCS stays high for at least 1 cycle between external device commands
	DEVSIZE	0x1F: set to maximum size of external device 4 GB
XSPI1_CCR	IMODE	0x1: instruction on a single line
	ADMODE	0x1: address on a single line
	ADSIZE	0x2: 24-bit address
	DMODE	0x1: data on a single line

Register	Register field	Applied configuration setting
XSPI1_TCR	DCYC	0x8: 8 CLK cycles of dummy phase
XSPI1_IR	INSTRUCTION	0xB: NOR instruction. Instruction to be sent to the external SPI device
XSPI1_DLR	DL	Number of data to be retrieved in indirect mode
XSPI1_CR	FMODE	0x1: Indirect-read mode
XSPI1_AR	ADDRESS	Address to be sent to the external device
XSPI1_DR	DATA	32-bit word: Data to be received from the external SPI device
XSPI1_FCR	CTCF	Clear transfer complete (TCF) flag in XSPI1_SR register
XSPIM_CR	MUXEN	0x0: No multiplexing
	MODE	0x1: if MUXEN = 0, swapped mode => XSPI1 connected to XSPIM_P2

In addition to configuring the actual XSPI1 / XSPIM hardware blocks, the boot ROM code also carries out the I/O settings, the power domain settings, and the clock settings dedicated to the XSPI1 / XSPIM hardware clocks.

3.4.6 Configuration settings of serial NOR device

There are no specific boot ROM settings to apply to the serial NOR device.

3.4.7 XSPI / XSPIM configuration for HyperFlash™ device

To access a HyperFlash™ device as a boot memory device attached to the STM32N6 MCU, the boot ROM code configures the XSPI1 controller and the XSPIM controller to use the XSPIM_P2 dedicated port for the flash device.

The selection of the HyperFlash™ device as a boot memory device is done by the applied boot configuration. The XSPI1 controller hardware block is configured for data transfer from the HyperFlash™ device according to Table 13 before starting the actual boot operation. The HyperFlash™ boot operation implemented in the boot ROM code is described the following sections. Registers and register fields not explicitly mentioned are not relevant for HyperFlash™ boot operation and can retain their default settings.

Table 15. XSPI1/XSPIM settings for HyperFlash™ device

Register	Register field	Applied configuration setting
XSPI1_DCR2	Prescaler	0x1: fclock = fkernel_clock / 2
XSPI1_DCR1	CSHT	0x7: NCS stays high for at least 8 cycles between external device commands
	DEVSIZE	0x1F: set to maximum size of external device 4 GB
	MTYP	0x4: Hyper-Bus memory mode
XSPI1_HLCR	TACC	0x10: Access time, 16 clock cycles
	TRWR	0x10: Read write recovery time, 16 clock cycles
XSPI1_CCR	DQSE	0x1: DQS enabled
	ADMODE	0x4: Address on eight lines
	ADSIZE	0x3: 32-bit address
	DMODE	0x4: Data on eight lines
	DDTR	0x1: Data double transfer rate (DTR) enabled for data phase
	ADDTR	0x1: Data double transfer rate (DTR) enabled for address phase
XSPI1_TCR	DHQC	0x1: ¼ cycle hold
XSPI1_IR	INSTRUCTION	0x0: Hyper-bus instruction. Instruction to be sent to the external SPI device
XSPI1_DLR	DL	Number of data to be retrieved in indirect mode
XSPI1_CR	FMODE	0x1: Indirect-read mode

Register	Register field	Applied configuration setting
XSPI1_AR	ADDRESS	Address to be sent to the external device
XSPI1_DR	DATA	32-bit word: data to be received from the external SPI device
XSPI1_FCR	CTCF	Clear transfer complete (TCF) flag in the XSPI1_SR register
XSPIM_CR	MUXEN	0x0: No multiplexing
	MODE	0x1: If MUXEN = 0, swapped mode => XSPI1 connected to XSPIM_P2

In addition to configuring the actual XSPI1 / XSPIM hardware blocks, the boot ROM code also carries out the I/O settings, the power domain settings, and the clock settings dedicated to the XSPI1 / XSPIM hardware clocks.

3.4.8 Configuration settings of HyperFlash™ device

There are no specific boot ROM settings to apply to the HyperFlash™ device.

3.5 Access image on boot memory device

3.5.1 Access image on SD card device

The boot ROM code accesses a next-stage boot image, FSBL, on an SD card device. Before accessing the SD card, the SDMMC hardware block must be configured.

The SD card does not support the boot operation mode as defined for eMMC devices. Therefore, it is always accessed through the card identification mode and then the data transfer mode based on the protocol defined in the standard specification (see reference SD_SPEC [2]).

Card identification mode

During card identification mode, several commands are sent to follow the standard specification (CMD0, CMD8, ACMD41, CMD55, CMD2, CMD3). The ACMD41 command identifies cards that do not match the power supply range supported by the SDMMC hardware block. The boot ROM code repeatedly issues ACMD41 until a response is provided, allowing the SD card to switch to the Ready state. The boot ROM code then completes the card identification mode by obtaining the new relative card address, switching the SDMMC hardware block to data transfer mode, and placing the SD card device in Standby state.

Data transfer mode

In data transfer mode, the boot ROM code searches for FSBLs. It searches for two FSBL images. First, it looks for the GPT magic number in the header. The GPT header is stored in LBA1 (logical block address #1 = 512 bytes). The boot ROM code then checks the GPT table to find FSBL offset addresses in LBA block offset. If GPT is not found, the boot ROM code uses default offsets set to 128@LBA and 640@LBA for FSBL1 and FSBL2, respectively.

The boot ROM code first downloads FSBL1 into the download buffer located in internal RAM, starting at the address DOWNLOAD_BUFFER_BASE_ADDR defined in Table 18, by reading multiple data blocks of 512-byte size using the SDMMC internal DMA. The image is then accessed from the download buffer for further processing by the boot ROM code. Once the complete SD boot image has been loaded into the download buffer, the boot ROM code executes it from there, after performing applicable image analysis and validation tasks.

If there is an issue during FSBL1 processing, the boot ROM code attempts to load FSBL2 into the download buffer and follows the same flow.

3.5.2 Access image on eMMC device boot partition

The boot ROM code accesses a next-stage boot image, FSBL, on an eMMC device. Before accessing the eMMC device, the SDMMC hardware block must be configured, and the eMMC card device must be programmed.

The boot operation, as defined for eMMC version 4.51 (see reference JEDEC-EMMC), is applied by the boot ROM code for reading an image from an attached eMMC device. For this boot operation, no commands are sent to the eMMC device. The transfer of boot partition data is initiated by holding the CMD line low for a specified minimum number of clock cycles after power-up or reset of the eMMC device and before any command is sent to the eMMC device.

The boot partition is a multiple of 128 Kbytes and is defined in the EXT_CSD register on the eMMC device. Once the FSBL is copied from the eMMC card to the internal RAM download buffer using internal DMA, the boot ROM code ends the boot mode and puts the eMMC device in a state to receive new commands.

3.5.3 Access image on serial NOR device

The boot ROM code accesses a next-stage boot image, FSBL, on a serial NOR device. Before accessing the serial NOR device, the XSPI/XSPIM hardware blocks must be configured.

The boot ROM code searches for FSBL1 and FSBL2 in the serial NOR device at offsets 0x0 and 0x40000, respectively. If the boot ROM code detects FSBL1 first, it loads it into the download buffer area at DOWNLOAD_BUFFER_BASE_ADDR, as defined in Table 18, and then executes the secure boot processing. If the secure boot processing fails with FSBL1, the boot ROM code clears the download buffer, and then downloads the FSBL2 image into the download buffer area. The boot ROM code updates the context structure accordingly for the fields: *bootPartitionUsedToBoot*, *bootInterfaceInstance*, and *bootInterfaceSelected*.

The boot ROM code context structure is described in the following sections. If the secure boot processing fails with FSBL2, the download buffer is cleared, and a serial boot is executed.

3.5.4 Access image on HyperFlash™ device

The boot ROM code is accessing a next-stage boot image, FSBL, on a HyperFlash™ device. Before accessing the HyperFlash™ device, it is required that the XSPI / XSPIM hardware blocks are configured.

The same boot ROM code processing than serial NOR device is applied to the HyperFlash™device as described in the following sections.

3.6 Peripheral boot interfaces

3.6.1 Peripheral boot interfaces activation

The boot ROM code activates and configures the interfaces used for DFU boot (download via external link) based on the current boot scenario. This activation of peripheral boot interfaces occurs at an early stage of the normal boot flow. The actual usage of these interfaces - starting with the serial peripheral boot link establishment procedure - happens in two possible stages: either selected by boot pins or after local boot (flash boot) fails, as described in Figure 2.

The following serial interfaces are supported as serial peripheral boot interfaces:

- Three USART interfaces: USART1, USART2, and UART4
- USB 2.0 OTG_HS 1 instance

Dedicated fuses can disable the usage of each individual serial interface for serial peripheral boot. OTP BOOTROM_CONFIG_11[16:9] disables a serial boot source (USART or USB). OTP BOOTROM_CONFIG_11[22:20] disables each USART instance.

Note: If OTP disables both USART and USB boot sources, the system forces UARTs to enable. Similarly, if OTP disables all USART instances, the system forces all USART instances to enable.

3.6.2 Peripheral boot link establishment

The boot ROM code can establish a serial peripheral boot channel on one of the serial boots interfaces activated for the current boot event. Therefore, the boot ROM code polls all activated serial boot interfaces for a certain period. A serial boot channel is established when the link establishment succeeds on a serial boot interface. For USB-based interfaces, the USB enumeration must be completed before entering the serial link establishment phase.

3.6.3 USART configuration

USART1, USART2, and UART4 share the same configuration:

- **GPIO mode**: Push-pull high speed with pull-up. Both Rx pins are configured as alternate functions.
- **Baud rate**: Configured by the host. If CubeProgrammer is used as the host, the baud rate is set to 115 200.
- **Data width**: Nine bits
- **Stop bits**: One stop bit
- **Parity bit**: Even
- **Hardware flow control**: None
- **Transfer direction**: TX_RX with 16-oversampling configured

Details on kernel and bus clocks are provided in the following sections.

Note: If USB is connected, USART is not available during this boot. To use USART boot, disconnect USB and apply a reset.

3.6.4 USB DFU device configuration

The USB DFU device configuration is split into multiple parts:

- **USBD_DFU_SRAM_InitVars**: The boot ROM code initializes packet descriptors, the SRAM write status, and file operations-related functions.
- **USBD_Desc_InitVars**: The boot ROM code initializes USB descriptors such as device, LangID, and serial descriptors.
- **USBD_DFU_InitVars**: The boot ROM code initializes USB DFU variables.
- **DFU_Desc_InitVars**: The boot ROM code initializes DFU descriptors such as device, LangID, manufacturer, product, configurations, serial, and interface descriptors.
- **USBD_Init**: The boot ROM code initializes the device stack, assigns USBD descriptors, and loads the class driver. Specifically, in the low-level portion of the device driver, the boot ROM code sets the LL driver parameters as follows:
  - **Instance**: USB1_OTG_HS
  - **Speed**: USB_OTG_SPEED_HIGH
  - **Number of endpoints**: 8
  - **PHY interface**: USB_OTG_HS_EMBEDDED_PHY
  - **Low power**: Disabled
  - **Link power management**: Disabled
  - **Vbus sensing**: Disabled
  - **Battery charging**: Disabled
  - **SOF**: Disabled
  - **RxFifo and TxFifo sizes**: 0x80

3.6.4.1 HSE bypassing

HSE bypass is explained in the RCC section of the reference manual [1].

When the boot ROM code finishes retrieving the type of HSE clock and finds it different from the HSE oscillator mode, it enters the HSE programming sequence. The boot ROM code executes these steps in a specific order:

1. Disable HSE
2. Get HSERDY flag
3. Set the HSE bypass
4. Select the external clock type
5. Start HSE

3.6.4.2 HSE autodetection

HSE crystal autodetection is explained in the RCC section of the reference manual (RM0486).

The boot ROM code can detect the mode in which the HSE should be used: either analog/digital bypass or oscillator mode. By reading two signal levels (GPIO PH1 inputs), the boot ROM code can detect four scenarios. If the HSE bypass detection disable bit is 0, then the detection bypass is enabled, and three scenarios are possible:

- **HSE digital external clock mode**: If level1 is low and level2 is low, HSE digital bypass is detected.
- **HSE analog external clock mode**: If level1 is high and level2 is high, HSE analog bypass is detected.
- **HSE oscillator mode**: If neither of the previous two modes is detected, no HSE bypass is detected.

If the HSE bypass detection is disabled, the oscillator mode is used.

The boot ROM code sets the USB clock tree after enabling HSE bypass.

3.7 Boot protocol on peripheral interface

3.7.1 Boot protocol overview

The USART and USB serial peripherals protocols are based on the STM32CubeProgrammer tool protocol described in the AN5275 application note.

Note that the above application note is specific to STM32MP1. The STM32N6 specifics and differences are described in the subsequent subchapters starting from Section 3.7.2.

Note: For boot ROM flow, only the protocol details for phase ID: 0x01 (FSBL image) are relevant.

3.7.2 USART FSBL download sequence

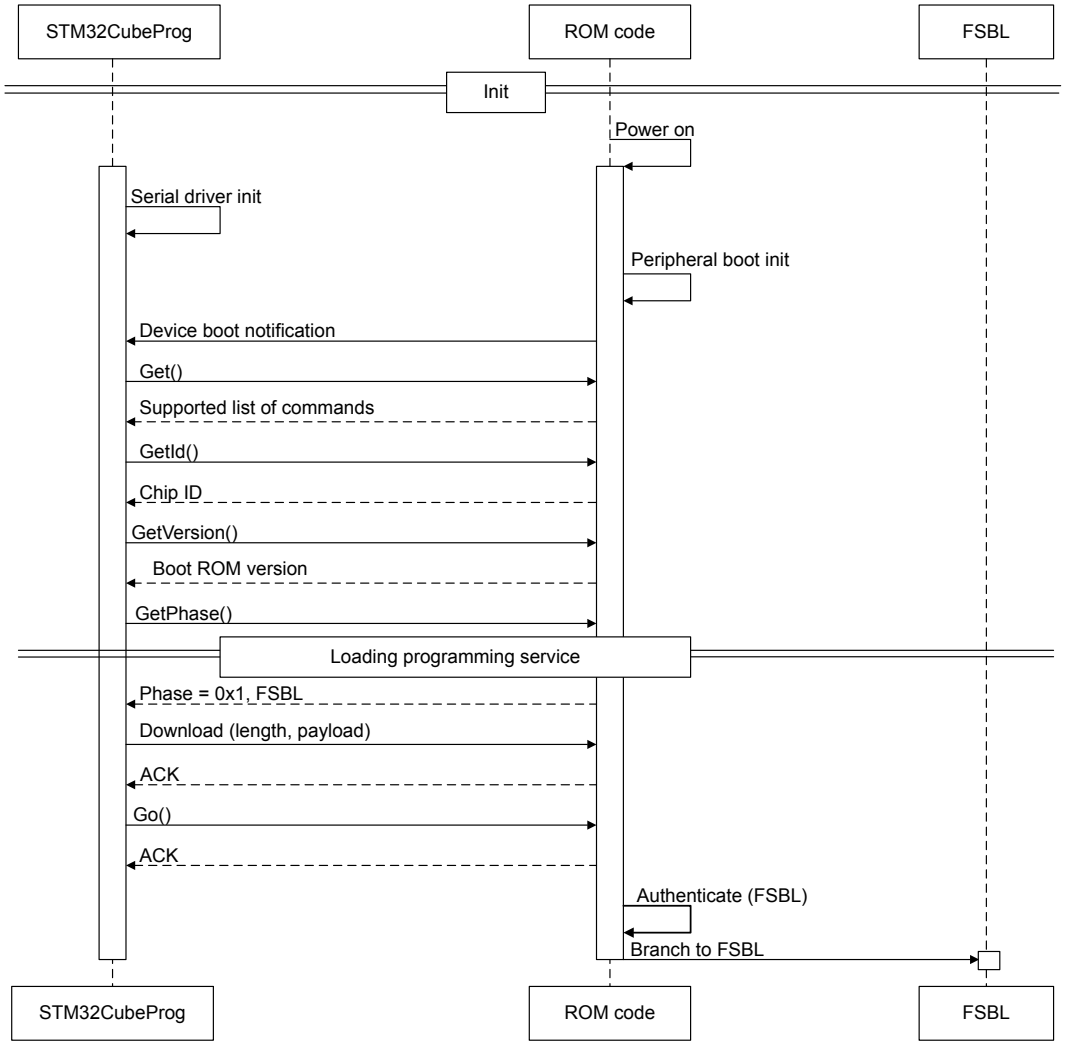
The USART boot sequence involves the following steps:

1. **Initialization (Init)**: The process begins with the initialization of the programming sequence.
2. **Loading programming service**: The programming service is loaded.

The sequence diagram illustrates the interaction between the following components:

- **STM32CubeProg**: The programming tool used for the USART boot sequence.
- **ROM code**: The read-only memory code that interacts with the programming tool.
- **FSBL**: The first stage bootloader, which is involved in the later stages of the process.

Figure 5. USART boot sequence



3.7.3 USART protocol

The supported commands for the STM32N6 USART protocol are listed in the table below:

Table 16. **USART commands**

Command	Code	Description
Get	0x00	Get a list of available USART commands.
Get version	0x01	Get the version (0x31).
Get ID	0x02	Get the device ID (0x0486).
Get phase	0x03	Get phase ID (0x01 load FSBL).
Read partition	0x12	Read data from partition (0xF3): Certificate
Start (Go)	0x21	Follow the boot ROM flow execution on the downloaded image.
Download (Write memory)	0x31	Download the image to the download buffer

3.7.4 USB serial protocol

The Boot ROM USB protocol is based on the DFU 1.1 protocol. The key difference is:

- **DFU_DETACH** is acceptable in the `dfuIDLE` state (before the execution of FSBL in internal RAM). When this occurs, the USB connection is disconnected, and a new USB enumeration is performed using the FSBL USB stack.

3.7.5 USB DFU STM32N6 enumeration

The USB parameters for the DFU STM32N6 enumeration are listed in the table below:

Table 17. **USB DFU STM32N6 enumeration parameters**

Parameter	Values
idVendor	0x0483
idProduct	0xDF11
iSerial	String with unique device ID
iProduct (HS)	"DFU in HS mode @Device ID /0x..., @Revision ID /0x...."
iProduct (FS)	"DFU in FS mode @Device ID /0x..., @Revision ID /0x...."

3.7.6 USB programming sequence

The USB programming sequence involves the following steps:

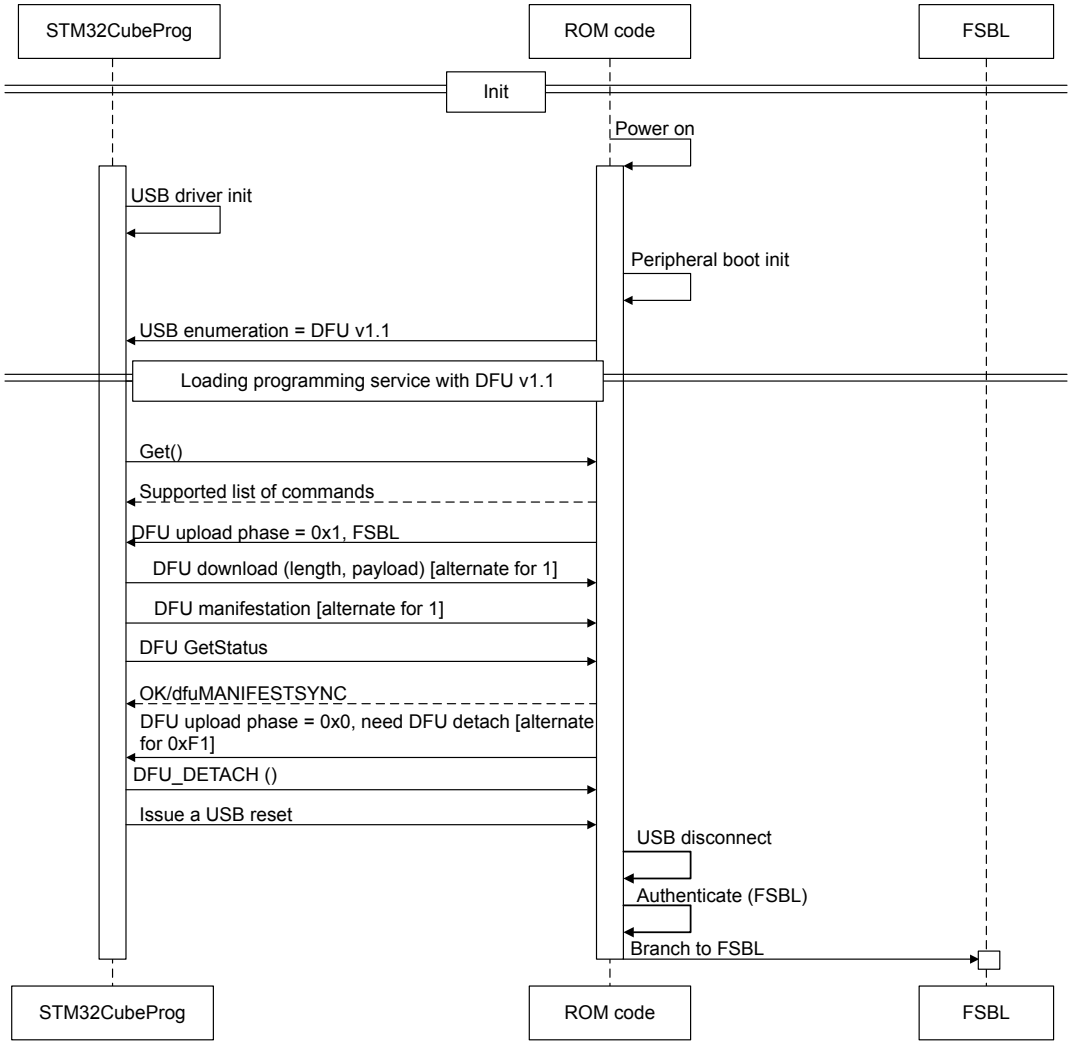
1. **Initialization (Init)**: The process begins with the initialization of the programming sequence.
2. **Loading programming Service with DFU v1.1**: The programming service is loaded using the DFU v1.1 protocol.

The sequence diagram illustrates the interaction between the following components:

- **STM32CubeProg**: The programming tool used for the USB programming sequence.
- **ROM code**: The read-only memory code that interacts with the programming tool.
- **FSBL**: The first stage bootloader, which is involved in the later stages of the process.

The arrows in the diagram indicate the flow of commands and responses between these components during the programming sequence.

Figure 6. **USB programming sequence**



3.8 Boot ROM version definition

The boot ROM code version structure information is contained within the boot ROM code function itself as a structure located at the starting address `BOOTROMCODE_VERS_ADDR`, as defined in Table 18. This structure is split into subelements according to the layout specified in Figure 7, with the contents defined in Table 17.

Table 18. **Defined boot ROM code version structure elements**

Element	Value	Description
ChipVersion	0x00008604	Chip version: 0x486 for STM32N6
CutVersion	0x00000200	Cut version: Cut 2.0
RommaskVersion	0x00000001	ROM mask version: 1
Bootrom_version	0x00000501	Boot ROM delivery release: DV5.1
forChipDesignRTL	0x001F0202	Chip design RTL version: ASSY 31.2.2
Platform_version	-	Platform version: not relevant

Figure 7. Boot ROM code version structure layout

ChipVersion

	MSB			LSB
Bytes	3	2	1	0
	0	0	ChipV_byte1	ChipV_byte0

CutVersion

	MSB			LSB
Bytes	3	2	1	0
	0	0	Major version	Minor version

RommaskVersion

	MSB			LSB
Bytes	3	2	1	0
	0	0	0	version

Bootrom_version

	MSB			LSB
Bytes	3	2	1	0
	0	0	Major version	Minor version

forChipDesignRTL

	MSB			LSB
Bytes	3	2	1	0
		Major version	Medium version	Minor version

3.9 Address definitions and memory layout of internal RAM

3.9.1 Definition of fixed memory addresses and base addresses

The fixed memory addresses used in the boot ROM code are listed in Table 19. The term "base address" is typically used for a fixed memory address when it relates to the start address of a memory area.

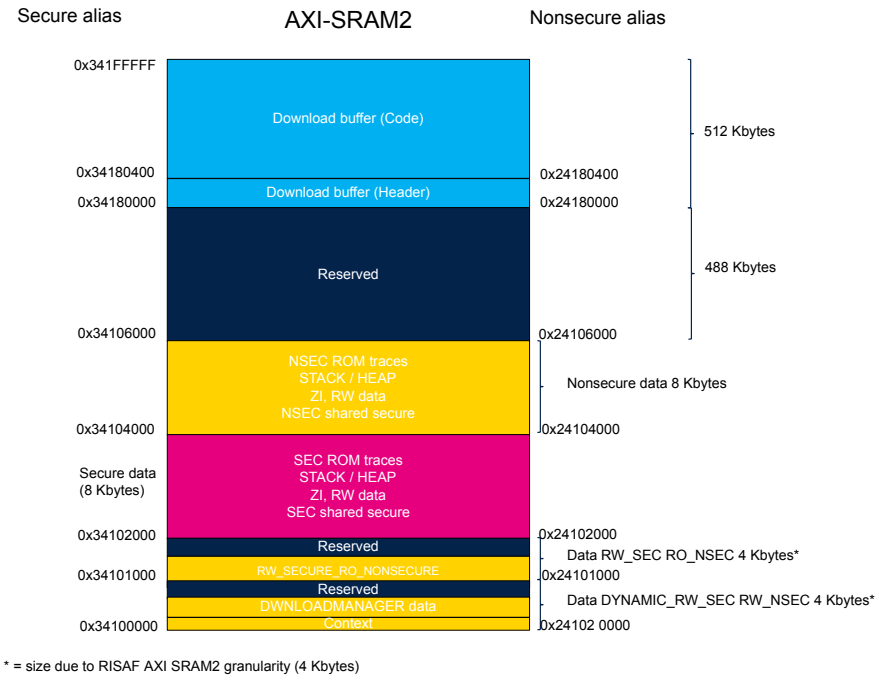
Table 19. Absolute memory addresses

Defined item	Physical address	Description
DOWNLOAD_BUFFER_BASE_ADDR	0xX4108000	Start address of the download buffer in internal AXI SRAM2 memory, used for copying the FSBL image from an external boot interface (Flash boot or serial boot). X = 2 for nonsecure access, X = 3 for secure access.
CONTEXT_BASE_ADDR	0xX4100000	Start address of the context information structure. X = 2 for nonsecure access, X = 3 for secure access.
BOOTROMCODE_VERS_ADDR	0x18001000	Start address of the boot ROM code version information structure.
TRACE_BUFFER_SEC_BASE_ADDR	0x341037F0	Start address of the SEC trace buffer during the current boot ROM execution.
TRACE_BUFFER_NSEC_BASE_ADDR	0x241077F0	Start address of the NSEC trace buffer during the current boot ROM execution.
E1CPVK_COPY_BASE_ADDR	0x38004000	Start address of the E1CPvK copy in the AHB SRAM2 internal RAM area, supported for the CLOSED_LOCKED_UNPROVD life cycle only.

3.9.2 Memory layout of internal RAM

The main internal RAM used by the boot ROM code to handle its own data is located in the internal AXI SRAM2 memory. The following figure shows the layout of this internal RAM for the STM32N6 project.

Figure 8. Boot ROM internal RAM memory layout



3.10 Error and trace logging

Errors and traces are managed by the boot ROM code. To indicate errors, the boot ROM code uses a dedicated GPIO to communicate statuses. A LED can be connected to the BOOTFAILN pin. In case of a blocking failure, this LED is switched on and set to low open drain.

During its execution, the boot ROM code writes binary traces to its memory. These traces are stored in the internal AXI-SRAM2 memory at the addresses `TRACE_BUFFER_SEC_BASE_ADDR` and `TRACE_BUFFER_NSEC_BASE_ADDR` (as defined in Section 3.9) for secure and nonsecure regions, respectively. There is a fuse bitfield to disable traces, `OTP_WORD16.bit0`, detailed in Section 3.11.1.

In case of a blocking failure, the boot ROM code writes a UART log error at 9600 baud to the debug GPIO pin PG10 through UART5_TX. The following table contains the boot ROM code statuses with their respective values. A specific status sets the corresponding bit to 1 on the `uint64_t` value.

Table 20. Boot ROM code status secure

Boot ROM code status	Bit number
BOOTCORE_STATUS_SEC_BOOT_CONFIG_ANALYZED	11
BOOTCORE_STATUS_SEC_ARM_EXCEPTION	12
BOOTCORE_STATUS_SEC_CHIPMODE_CLOSED_UNLOCKED	20
BOOTCORE_STATUS_SEC_CHIPMODE_CLOSED_LOCKED_UNPROVD	21
BOOTCORE_STATUS_SEC_CHIPMODE_CLOSED_LOCKED_PROVD	22
BOOTCORE_STATUS_SEC_CHIPMODE_INVALID	23
BOOTCORE_STATUS_SEC_NO_BOOT	24
BOOTCORE_STATUS_SEC_NO_BOOT_LOOP	25
BOOTCORE_STATUS_SEC_BLOCKING_FAILURE	26
BOOTCORE_STATUS_SEC_SECURE_BOOT	32

Boot ROM code status	Bit number
BOOTCORE_STATUS_SEC_DEV_BOOT	33
BOOTCORE_STATUS_SEC_PLL1_LOCKED	39
BOOTCORE_STATUS_SEC_SIGNATURE_OK	43
BOOTCORE_STATUS_SEC_SIGNATURE_FAIL	44
BOOTCORE_STATUS_SEC_WRONG_IMAGE_VERSION	45
BOOTCORE_STATUS_SEC_CRC_KO	53
BOOTCORE_STATUS_SEC_DECRYPT_OK	56
BOOTCORE_STATUS_SEC_DECRYPT_KO	57
BOOTCORE_STATUS_SEC_IMGVERSION_PRG	61
BOOTCORE_STATUS_SEC_IMGVERSION_PRGERR	62
BOOTCORE_STATUS_SEC_EXIT_FSBL_DONE	63

Table 21. Boot ROM code status nonsecure

Boot ROM code status	Bit number
BOOTCORE_STATUS_NSEC_PLLUSB_LOCKED	1
BOOTCORE_STATUS_NSEC_TRACE_IS_INITIALIZED	2

3.11 Supported configurations

3.11.1 Boot ROM related fuse settings

OTP words and bits description

The following table provides detailed descriptions of the OTP words and bits used in the boot ROM code.

Table 22. OTP words and bits

OTP word	OTP bits	Name / description	Detailed description
OTP0	-	OTP_HW_WORD0	Virgin check
OTP1	-	OTP_HW_WORD1	OTP security word to close security state
OTP2	-	OTP_HW_WORD2	OTP word for reopening (CLOSE->OPEN) via RMA password: RMA bits
OTP3	-	OTP_HW_WORD3	OTP Word for reopening (CLOSE->OPEN) via RMA password: RMA tries bits
OTP4	-	OTP_HW_WORD4	OTP word for TK retries (ECIES), used during ST key provisioning process
OTP5	-	ID0	-
OTP6	-	ID1	-
OTP7	-	ID2	-
OTP8	-	RPN_CODING	-
OTP9	-	FEATURE_DISABLING	Feature disabling
OTP10	-	BOOTROM_CONFIG_1	-
	[26:19]	rssefw_active_signing_key	Monotonic key index of active signing key used for RSSe_FW authentication
OTP11	-	BOOTROM_CONFIG_2	-
	[0]	no_data_cache	0 (enabled): Data cache is used by boot ROM.

OTP word	OTP bits	Name / description	Detailed description
OTP11			1 (disabled): Data cache is not used by boot ROM.
	[1]	no_cpu_pll	0 (enabled): PLLs for CPU/AXI are enabled for cold boot. 1 (disabled): PLLs for CPU/AXI are not enabled for cold boot.
	[2]	sdmmc1_not_default_af	0 (no): SDMMC1 uses a default hard-coded AFmux. 1 (yes): SDMMC1 uses AFmux defined in OTP.
	[3]	sdmmc2_not_default_af	0 (no): SDMMC2 uses the default hard-coded AFmux. 1 (yes): SDMMC2 uses AFmux defined in OTP.
	[8:5]	flash_boot_source	1 (SD card): SD card SDMMC1. 2 (emmc): eMMC SDMMC1. 3 (snor): XSPI NOR. 4 reserved. 5 (hflash): XSPI HyperFlash™. 6 reserved. 7 (SD card): SD card SDMMC2. 8 (emmc): eMMC SDMMC2. Other: invalid.
	[16:9]	boot_source_disable	0x01 (usb): disable USB boot source. 0x02 (uart): disable UART boot source. 0x04 reserved. 0x08 reserved. 0x10 reserved.
	[19:17]	reserved	-
	[22:20]	uart_instance_disable	0b001: disable USART1 instance. 0b010: disable USART2 instance. 0b100: disable USART3 instance.
	[28:26]	reserved	-
	[29]	tamp_boot_cfg_glob_enable	Enable the configuration of tampers in the boot ROM before the boot process. 0: configuration of tampers is disabled. 1: configuration of tampers is enabled.
	[30]	xspi_3v3	HyperFlash™ is a 3.3V device. 0: not a 3.3V XSPI. 1: It is a 3.3V XSPI.
OTP12	[31:0]	BOOTROM_CONFIG_3	rssefw_version_monotonic_counter
OTP13	-	BOOTROM_CONFIG_4	-
	[3:0]	mode0	0 (af_nopull_ls): AF; no pull; low speed. 1 (af_nopull_ms): AF; no pull; medium speed. 2 (af_nopull_hs): AF; no pull; high speed. 3 (af_pullup_ls): AF; pull up; low speed. 4 (af_pullup_ms): AF; pull up; medium speed. 5 (af_pullup_hs): AF; pull up; high speed. 6 (af_pulldown_ls): AF; pull down; low speed. 7 (af_pulldown_ms): AF; pull down; medium speed.

OTP word	OTP bits	Name / description	Detailed description
OTP13			8 (af_pulldown_hs): AF; pull down; high speed. 9 (gpio_out_high): GPIO output high. 10 (gpio_out_low): GPIO output low. 11 (gpio_in): GPIO **input**. 12 (gpio_open_nopull): GPIO **open** drain; no pull. 13 (gpio_open_pullup): GPIO **open** drain; pull up. 14 (gpio_open_pulldown): GPIO **open** drain; pull down. 15 (gpio_analog): GPIO analog mode.
	[7:4]	afmux0	Value between 0 and 15.
	[11:8]	pin0	[0-15]: pin id between 0 and 15 for GPIOA to GPIOG and GPIOP. [0-12]: pin id between 0 and 12 for GPION. [0-8]: pin id between 0 and 8 for GPIOH and GPIOQ. [0-5]: pin id between 0 and 5 for GPIOO.
	[15:12]	port0	0: unused. 1 (PA): Bank A. 2 (PB): Bank B. 3 (PC): Bank C. 4 (PD): Bank D. 5 (PE): Bank E. 6 (PF): Bank F. 7 (PG): Bank G. 8 (PH): Bank H. 9 (PN): Bank N. 10 (PO): Bank O. 11 (PP): Bank P. 12 (PQ): Bank Q. 0b1111 (invalid): Invalid configuration.
	[19:16]	mode1	Same as BOOTROM_CONFIG_4.mode0
	[23:20]	afmux1	Same as BOOTROM_CONFIG_4.afmux0
	[27:24]	pin1	Same as BOOTROM_CONFIG_4.pin0
OTP14	-	BOOTROM_CONFIG_5	Same as BOOTROM_CONFIG_4
OTP15	-	BOOTROM_CONFIG_6	Same as BOOTROM_CONFIG_4
OTP16	-	BOOTROM_CONFIG_7	-
	[0]	disable_traces	0 (no): Boot ROM traces are enabled. 1 (yes): Boot ROM traces are disabled.
	[1]	disable_hse_freq_detect	0 (no): HSE frequency autodetection is enabled. 1 (yes): HSE frequency autodetection is disabled.
	[2]	disable_hse_bypass_detect	0 (no): HSE bypass detection is enabled. 1 (yes): HSE bypass detection is disabled.
	[6]	emergency_debug_req	0 (no): emergency debug is not requested. 1 (yes): emergency debug is requested.
	[7]	emmc_128k_boot_partition	0 (no): Boot ROM does not support eMMC with 128 Kbytes boot partition.

OTP word	OTP bits	Name / description	Detailed description
OTP16			1 (yes): Boot ROM supports eMMC with 128 Kbytes boot partition.
	[9]	iomgr_port	reserved
	[10]	iomgr_muxen	reserved
	[13:11]	HSE_value	0b000 (auto): HSE value is autodetected at 19.2, 20, 24, 38.4, 40, 48 MHz. 0b001 (19.2 MHz): HSE = 19.2 MHz. 0b010 (20 MHz): HSE = 20 MHz. 0b011 (24 MHz): HSE = 24 MHz. 0b100 (38.4 MHz): HSE = 38.4 MHz. 0b101 (40 MHz): HSE = 40 MHz. 0b110 (48 MHz): HSE = 48 MHz. 0b111 (unused): Reserved.
OTP17	-	BOOTROM_CONFIG_8	-
	[7:0]	oem_active_signing_key	[1-256] -> [1-8]: The value of the monotonic counter is X where X is the position of the most significant bit at 1. 8 possible OEM public keys (OEM key revocation feature for OEM-FSBL authentication).
OTP18	-	BOOTROM_CONFIG_9	-
	[3:0]	secure_boot	0 (unlocked): The chip is in a CLOSED_UNLOCKED state. Secure boot is not enforced (FSBL authentication is not mandatory). [1-64] (locked): The chip is in a CLOSED_LOCKED state. Secure boot is enforced (FSBL authentication is mandatory).
	[4]	fsbl_decrypt_prio	0 (speed): Priority speed: Boot ROM uses CRYP to decrypt FSBL. 1 (security): Priority security (DPA protection): Boot ROM uses SAES (with integrated hardware DPA protection) to decrypt FSBL.
	[8:5]	prov_done	Used only when the chip is CLOSED_LOCKED. Determines if the chip is in CLOSED_LOCKED_UNPROVD or CLOSED_LOCKED_PROVD. 0 (no): The provisioning was not done or did not finish successfully. The chip is CLOSED_LOCKED_UNPROVD and only accepts ST-RSSE-FW. [1-64] (yes): The provisioning was done successfully and the chip is CLOSED_LOCKED_PROVD and only accepts OEM FSBL.
	[12:9]	enable_fingerprint	0 (yes): The fingerprint feature is disabled. 1 (no): Fingerprint feature is enabled.
	[21:16]	nb_added_stsecrets	Number of OTP words located in the upper area [360-nb_added_stsecrets..359] that were provisioned (in encrypted mode) with ST secrets. These are decoded and used by RSSE firmware. Coding up to 64 ST secrets to provision in EWS (with DEV_BOOT).
	[25:22]	debug_lock	0: Do not lock debug enabling. [1-64]: Lock debug enabling.
	[26]	ns_epoch_enable	0: The boot ROM only sets bsec_epoch0.

OTP word	OTP bits	Name / description	Detailed description
OTP18			1: The boot ROM sets both bsec_epoch0 and bsec_epoch1.
OTP19	-	BOOTROM_CONFIG_10	-
	[20:18]	rng_htcr_value	0: Default value, RNG HTCR not modified. 1: 0xA2B3. 2: 0xAA74. 3: 0xA6BA. 4: 0x9AAE. 5: 0x72AC. 6: 0xAAC7. Other: Default value, RNG HTCR not modified.
	[24:21]	dev_boot_port	0: Unused. 1 (PA): Bank A. 2 (PB): Bank B. 3 (PC): Bank C. 4 (PD): Bank D. 5 (PE): Bank E. 6 (PF): Bank F. 7 (PG): Bank G. 8 (PH): Bank H. 9 (PN): Bank N. 10 (PO): Bank O. 11 (PP): Bank P. 12 (PQ): Bank Q.
	[28:25]	dev_boot_pin	[0-15]: Pin ID between 0 and 15 for GPIOA to GPIOG and GPIOP. [0-12]: Pin ID between 0 and 12 for GPION. [0-8]: Pin ID between 0 and 8 for GPIOH and GPIOQ. [0-5]: Pin ID between 0 and 5 for GPIOO.
OTP20	-	BOOTROM_CONFIG_11	-
	[31:0]	oem_fsbl_monotonic_counter	[1-0xFFFF] -> [1-32]: The value of monotonic counter is X where X is the position of the most significant bit at 1.
OTP21	-	BOOTROM_CONFIG_12	-
	[31:0]	oem_fsbl_monotonic_counter	[1-0xFFFF] -> [33-64]: The value of the monotonic counter is 32+X where X is the position of the most significant bit at 1.
OTP22	-	BOOTROM_CONFIG_13	-
OTP23	-	BOOTROM_CONFIG_14	-
	[31:0]	h32e1cpvk	Fused by boot ROM after checking it from ST Key provisioning. Used by the final test program.
OTP24	-	BOOTROM_TZ_EPOCH0	TZ epoch counter. If the highest blown bit is the nth bit of these 256 bits, the boot ROM sets BSEC3_EPOCH_TZ = n.
OTP25	-	BOOTROM_TZ_EPOCH1	-
OTP26	-	BOOTROM_TZ_EPOCH2	-
OTP27	-	BOOTROM_TZ_EPOCH3	-
OTP28	-	BOOTROM_TZ_EPOCH4	-
OTP29	-	BOOTROM_TZ_EPOCH5	-
OTP30	-	BOOTROM_TZ_EPOCH6	-
OTP31	-	BOOTROM_TZ_EPOCH7	-
OTP32	-	BOOTROM_NS_EPOCH0	NS epoch counter. If the highest blown bit is the nth bit of these 256 bits, the boot ROM sets BSEC3_EPOCH_NS = n.
OTP33	-	BOOTROM_NS_EPOCH1	-
OTP34	-	BOOTROM_NS_EPOCH2	-
OTP35	-	BOOTROM_NS_EPOCH3	-

OTP word	OTP bits	Name / description	Detailed description
OTP36	-	BOOTROM_NS_EPOCH4	-
OTP37	-	BOOTROM_NS_EPOCH5	-
OTP38	-	BOOTROM_NS_EPOCH6	-
OTP39	-	BOOTROM_NS_EPOCH7	-
OTP40	-	BOOTROM_TZ_COUNT0	TZ provisioning NV counter (blown to regress to RoT-READY state).
OTP41	-	BOOTROM_TZ_COUNT1	-
OTP42	-	BOOTROM_TZ_COUNT2	-
OTP43	-	BOOTROM_TZ_COUNT3	-
OTP44	-	BOOTROM_TZ_COUNT4	-
OTP45	-	BOOTROM_TZ_COUNT5	-
OTP46	-	BOOTROM_TZ_COUNT6	-
OTP47	-	BOOTROM_TZ_COUNT7	-
OTP48	-	BOOTROM_NS_COUNT0	NS provisioning NV counter (blown to regress to TZ-OEM_CLOSED).
OTP49	-	BOOTROM_NS_COUNT1	-
OTP50	-	BOOTROM_NS_COUNT2	-
OTP51	-	BOOTROM_NS_COUNT3	-
OTP52	-	BOOTROM_NS_COUNT4	-
OTP53	-	BOOTROM_NS_COUNT5	-
OTP54	-	BOOTROM_NS_COUNT6	-
OTP55	-	BOOTROM_NS_COUNT7	-
OTP56	-	OTP_TAMP_EN	Boot ROM tampers enabling
OTP57	-	OTP_TAMP_CFM	Boot ROM tampers confirmed / potential configuration
OTP58	-	OTP_TAMP_LVL	Boot ROM tampers external (TAMPINx) individual configuration
OTP59-123	-	ST RESERVED	-
OTP124	-	HW_CONF1	-
	[0]	IWDG1_HW	-
	[1]	IWDG1_FZ_STOP	-
	[2]	IWDG1_FZ_STANDBY	-
	[10]	RST_STOP	-
	[11]	RST_STDBY	-
OTP125	-	MEM_REPAIR	-
OTP127	-	MEM_REPAIR	-
OTP128	-	STM32CERTIF0	STM32 device certificate [511:480]
OTP129	-	STM32CERTIF1	-
OTP130	-	STM32CERTIF2	-
OTP131	-	STM32CERTIF3	-
OTP132	-	STM32CERTIF4	-
OTP133	-	STM32CERTIF5	-
OTP134	-	STM32CERTIF6	-

OTP word	OTP bits	Name / description	Detailed description
OTP135	-	STM32CERTIF7	-
OTP136	-	STM32CERTIF8	-
OTP137	-	STM32CERTIF9	-
OTP138	-	STM32CERTIF10	-
OTP139	-	STM32CERTIF11	-
OTP140	-	STM32CERTIF12	-
OTP141	-	STM32CERTIF13	-
OTP142	-	STM32CERTIF14	-
OTP143	-	STM32CERTIF15	[31:0]
OTP144	-	STM32PUBKEY0	STM32 PublicKey [511:480]
OTP145	-	STM32PUBKEY1	-
OTP146	-	STM32PUBKEY2	-
OTP147	-	STM32PUBKEY3	-
OTP148	-	STM32PUBKEY4	-
OTP149	-	STM32PUBKEY5	-
OTP150	-	STM32PUBKEY6	-
OTP151	-	STM32PUBKEY7	-
OTP152	-	STM32PUBKEY8	-
OTP153	-	STM32PUBKEY9	-
OTP154	-	STM32PUBKEY10	-
OTP155	-	STM32PUBKEY11	-
OTP156	-	STM32PUBKEY12	-
OTP157	-	STM32PUBKEY13	-
OTP158	-	STM32PUBKEY14	-
OTP159	-	STM32PUBKEY15	[31:0]
OTP160	-	OEM_ROT0	OEM rot implementation example: if hash = 01 02 03 04 05 06 07 08… then OEM_ROT0 = 0x01020304, OEM_ROT1 = 0x05060708, etc…
OTP161	-	OEM_ROT1	
OTP162	-	OEM_ROT2	
OTP163	-	OEM_ROT3	
OTP164	-	OEM_ROT4	
OTP165	-	OEM_ROT5	
OTP166	-	OEM_ROT6	
OTP167	-	OEM_ROT7	
OTP168	-	ST_RSSE_EDMK_DERIV_CSTE	The derivation constant used to generate ST decryption key from ST encryption/decryption controller Key
OTP169	-	MAC_ADDR0_LOW	-
OTP170	-	MAC_ADDR0_HIGH	-
OTP171	-	MAC_ADDR1_LOW	-
OTP172	-	MAC_ADDR1_HIGH	-
OTP173	-	-	NOT USED
OTP239	-	-	NOT USED

OTP word	OTP bits	Name / description	Detailed description
OTP244	-	-	OEM debug public key and device mask
OTP255	-	-	-
OTP256	-	RMA_PASWD0	-
OTP257	-	RMA_PASWD1	-
OTP258	-	RMA_PASWD2	-
OTP259	-	RMA_PASWD3	-
OTP260	-	-	NOT USED
OTP363	-	-	NOT USED
OTP364	-	OEM_EDMK0	OEM MasterKey [127:96]
OTP365	-	OEM_EDMK1	-
OTP366	-	OEM_EDMK2	-
OTP367	-	OEM_EDMK3	[31:0]
OTP368	-	STM32PRVKEY0	STM32 ECC CHIP PRIV KEY [255:244]
OTP369	-	STM32PRVKEY1	-
OTP370	-	STM32PRVKEY2	-
OTP371	-	STM32PRVKEY3	-
OTP372	-	STM32PRVKEY4	-
OTP373	-	STM32PRVKEY5	-
OTP374	-	STM32PRVKEY6	-
OTP375	-	STM32PRVKEY7	[31:0]
OTP376	-	HWKEY0	HWKEY [31:0]
OTP377	-	HWKEY1	-
OTP378	-	HWKEY2	-
OTP379	-	HWKEY3	-
OTP380	-	HWKEY4	-
OTP381	-	HWKEY5	-
OTP382	-	HWKEY6	-
OTP383	-	HWKEY7	[255:224]

3.11.2 Boot ROM related status information

The status for boot ROM code execution is available in boot ROM traces/status word, described in the Section 3.10.

Additionally, the boot ROM code provides a context structure to the next image software. This context structure is stored at the starting address `CONTEXT_BASE_ADDR`, defined in Section 3.9. Its layout is detailed in the following table:

Table 23. Context structure elements

Element	Length (in bytes)	Description
bootPartitionUsedToBoot	4	Boot partition selected.
		0: No partition
		1: FSBL1 in flash boot
		2: FSBL2 in flash boot

Element	Length (in bytes)	Description
SdErrInternalTimeoutCnt	4	SD Overall internal timeout waiting on flags error count
SdErrDcrcFailCnt	4	SD Overall DCRCFAIL error count
SdErrDtimeoutCnt	4	SD Overall DTIMEOUT error count
SdErrCtimeoutCnt	4	SD Overall CTIMEOUT error count
SdErrCcrcFailCnt	4	SD Overall CCRCFAIL error count
SdOverallRetryCnt	4	SD Overall retry command/data sent/receive count
EmmcXferStatus	4	eMMC transfer status
EmmcErrorStatus	4	eMMC error status
EmmcNbBytesRxCopiedToSysramDownloadArea	4	eMMC number of bytes received from card and copied to download buffer
bootInterfaceSelected	2	Boot link interface. 0: No interface 1: SD 2: eMMC 4: Snor XSPI 5: Serial UART 6: Serial USB 8: HyperFlash™ XSPI
bootInterfaceInstance	2	Boot interface instance. For USART link: 1: USART1 2: USART2 3: USART4 For SD, eMMC: 1: SDMMC1 2: SDMMC2 Otherwise, set to 1 for all other links
HseClockValueInHz	4	Value used for HSE clock in Hz using USB serial link. Possible values: 0, 19200000, 20000000, 24000000, 38400000, 40000000, 48000000
Reserved	4	Reserved
authStatus	4	Authentication status. 0: No authentication is done 1: Authentication failed 2: Authentication success
RomVersioninfo	24	Boot ROM version info structure is detailed in Section 3.8

3.12 Applied configuration of hardware resources

3.12.1 Configuration of clock resources

The Boot ROM code sets system-level clock resources following two predefined scenarios: the nominal clock scenario and the default clock scenario used in engineering modes.

The scenario is determined by the `no_pll_clock` bit set in the `BOOTROM_CONFIG_2` fuse (see Boot ROM Related Fuse Settings).

Clock Scenarios

Clock scenario	no_pll_bit	CPU	AHB/APB	AXI/Timer
nominal	0	400 MHz	150 MHz	300 MHz
default	1	64 MHz	32 MHz	64 MHz

Nominal bus interface clocks for supported peripherals (UART/OCTOSPI/USB) are driven by the nominal AHB/APB clock.

Each peripheral requires one or several bus interface clocks named `rcc_perx_bus_ck` (for peripheral 'x'). These clocks can be APB, AHB, or AXI clocks, depending on which bus or buses the peripheral is connected to.

Some peripherals also require dedicated clocks for their communication interface. These clocks are generally asynchronous with respect to the bus interface clock and are named kernel clocks (`perx_ker_ck`). Both bus interface and kernel clocks are generated by the RCC and can be gated according to several conditions detailed hereafter.

Enabling the kernel and bus interface clocks of each peripheral depends on several input signals described in RCC section of the reference manual (RM0486).

Summary of different bus interface clocks

- **Pclk1**: Used as the bus interface clock for UART4 and UART5.
- **Pclk2**: Used as the bus interface clock for USART1.
- **Hclk1**: Used as the bus interface clock for OTG-HS.
- **Hclk2**: Used as the bus interface clock for SDMMC2.
- **Hclk5**: Used as the bus interface clock for OCTOSPI1/2 and SDMMC1.

Default clock scenario

- `ck_cpu = sysa_ck = sys_cpu_ck = hsi_ck = 64 MHz`
- `aclkx = sysb_ck = hsi_ck = 64 MHz`
- `hclk1,2,3,4,5 = sysb_ck / 2 = 64 / 2 = 32 MHz`
- `pclk1,2,3,4,5 = hclk1..5 / 1 = 32 / 1 = 32 MHz`
- `per_ck = his_ck = 64 MHz`

Nominal clock scenario

During secure boot, the boot ROM sets up PLL1 to switch to CM55, APB, AHB, and AXI clocking at the nominal boot frequency. The clock values are:

- `ck_cpu = ic1_ck = 400 MHz`
- `aclkx = sysb_ck = ic2_ck = 300 MHz`
- `hclk1,2,3,4,5 = 150 MHz`
- `pclk1,2,3,4,5 = 150 MHz`
- `per_ck = his_ck = 400 MHz`

TIM2 initialization

- Uses the HSI clock in the default scenario.
- Uses the SYSB clock in the nominal scenario.

Kernel clocks

The kernel clocks are set according to the needs of some peripherals:

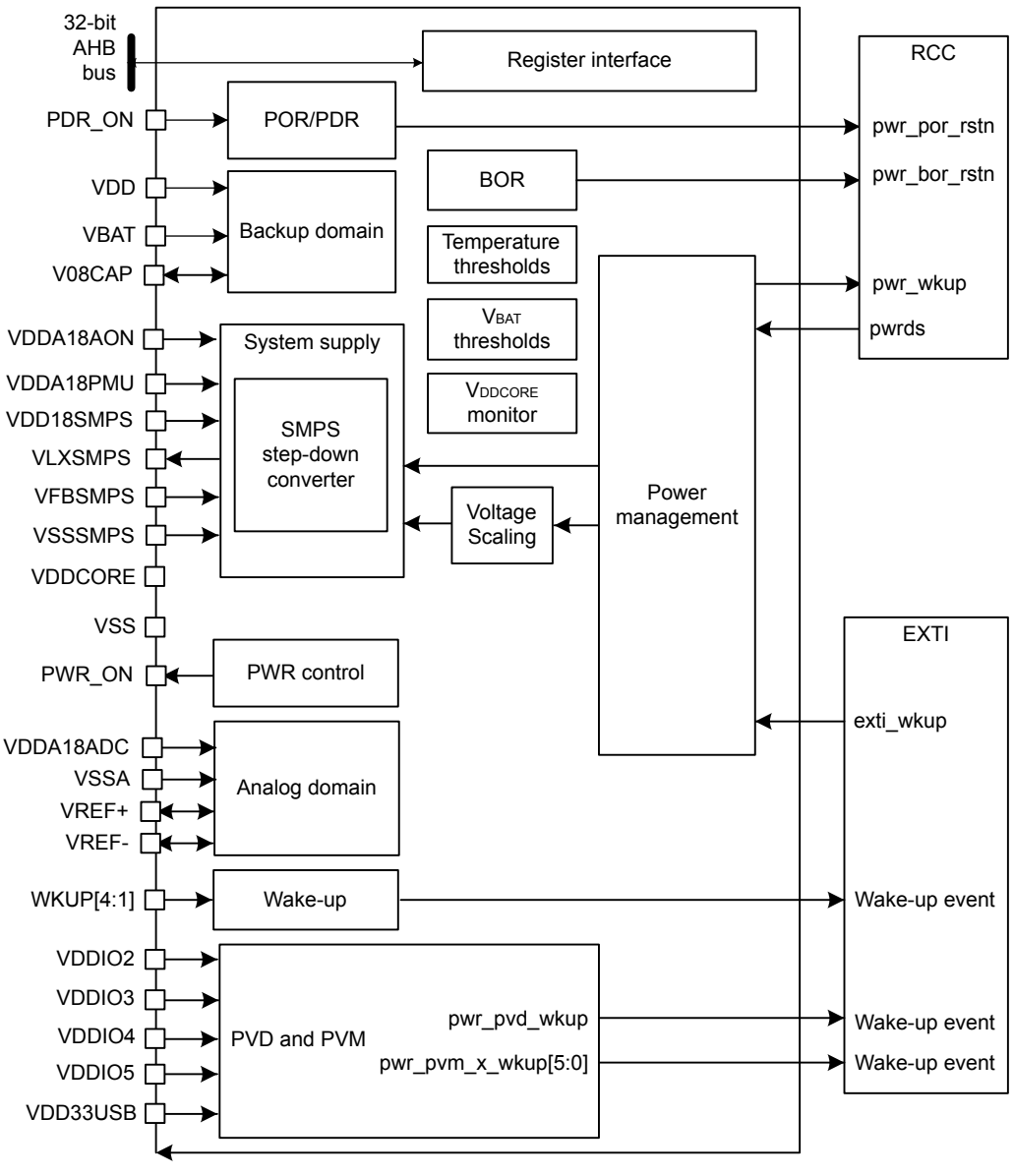
- `per_ck` is selected for UART4, USART1, USART2, XSPI1, SDMMC1, and SDMMC2.

- `hsi_div_ck` is selected for UART5. `hsi_div_ck` is set to `hsi_ck` with a prescaler programmed to 8 MHz to obtain a baud rate of 9600.
- `hse_ker_ck / 2` is selected for OTG1.

3.12.2 Configuration of power domains

The following figure describes the power control block diagram.

Figure 9. Power control block diagram



The boot ROM code uses multiple supplies for I/Os:

- **V_{DDIO3}**: Independent I/O supply 3 (PN[12:0]), configured for XSPI1M_P2 (XSPI).
- **V_{DDIO4}**: Independent I/O supply 4 (PC[1], PC[12:6], and PH[2,9]), configured for eMMC.
- **V_{DDIO5}**: Independent I/O supply 5 (PC[0], PC[5:2], and PE[4]), configured for SD card.
- **$V_{DD33USB}$**: Supply input for USB HS PHYs and USB Type-C® PHY 3V3.

These supplies can be independent of VDD and can be monitored with peripheral voltage monitoring. The voltage range configuration is described in the PWR section of the reference manual (RM0486).

3.12.3 Configuration of ports and pads

The boot ROM code configures and uses multiple ports and pads to configure XSPI1, USART1, USART2, UART4, UART5, SDMMC1, and SDMMC2. Configurations are fixed and labeled by AFx, which are AFmux settings:

- 0 (af_nopull_ls): AF; no pull; low speed
- 1 (af_nopull_ms): AF; no pull; medium speed
- 2 (af_nopull_hs): AF; no pull; high speed
- 3 (af_pullup_ls): AF; pull up; low speed
- 4 (af_pullup_ms): AF; pull up; medium speed
- 5 (af_pullup_hs): AF; pull up; high speed
- 6 (af_pulldown_ls): AF; pull down; low speed
- 7 (af_pulldown_ms): AF; pull down; medium speed
- 8 (af_pulldown_hs): AF; pull down; high speed
- 9 (gpio_out_high): GPIO output high
- 10 (gpio_out_low): GPIO output low
- 11 (gpio_in): GPIO input
- 12 (gpio_open_nopull): GPIO open drain; no pull
- 13 (gpio_open_pullup): GPIO open drain; pull up
- 14 (gpio_open_pulldown): GPIO open drain; pull down
- 15 (gpio_analog): GPIO analog mode

XSPI1 configuration

XSPI1 is configured differently depending on the boot sources.

Table 24. Pin configuration XSPI1 for sNOR

XSPI1_NOR	XSPIM_P2
XSPI1_CLK	PN6 (AF9)
XSPI1_NCS1	PN1 (AF9)
XSPI1_IO0	PN2 (AF9)
XSPI1_IO1	PN3 (AF9)

Only two pins are used for communication (SPI legacy mode MOSI/MISO uses two pins, one for sending and one for receiving). The two pins are configured in indirect mode without the use of DMA. XSPIM_P2 (port N) is selected for flash boot as it is supported in all packages. IOM is configured to map XSPI1 to XSPIM_P2 (MUXEN = 0, MODE = 1).

For the boot category HyperFlash™, the following pins are used:

Table 25. Pin configuration XSPI1 for HyperFlash™

XSPI1_HYPERFLASH	XSPIM_P2
XSPI1_CLK	PN6 (AF9)
XSPI1_NCLK	PN7 (AF9)
XSPI1_NCS	PN1 (AF9)
XSPI1_DQS0	PN0 (AF9)
XSPI1_IO0	PN2 (AF9)
XSPI1_IO1	PN3 (AF9)
XSPI1_IO2	PN4 (AF9)
XSPI1_IO3	PN5 (AF9)
XSPI1_IO4	PN8 (AF9)

XSPI1_HYPERFLASH	XSPIM_P2
XSPI1_IO5	PN9 (AF9)
XSPI1_IO6	PN10 (AF9)
XSPI1_IO7	PN11 (AF9)

XSPIM_P2 (port N) is selected for flash boot as it is supported in all packages. IOM is configured to map XSPI1 to XSPIM_P2 (MUXEN = 0, MODE = 1). The boot ROM code supports 3V3 HyperFlash™ devices by selecting the fuse OTP_WORD, which selects different I/O configurations accordingly.

USART1 configuration

Table 26. Pin configuration for USART1

USART1	
USART1_RX	PE6 (AF7)
USART1_TX	PE5 (AF7)

USART2 configuration

Table 27. Pin configuration for USART2

USART2	
USART2_RX	PF6 (AF7)
USART2_TX	PA2 (AF7)

UART4 configuration

Table 28. Pin configuration for UART4

UART4	
UART4_RX	PA1 (AF8)
UART4_TX	PA0 (AF8)

All Rx instances are scanned in parallel. Tx is only selected until activity is detected on Rx. These AFmux configurations cannot be overwritten.

UART5 configuration

Table 29. Pin configuration for UART5

UART5	
UART5_TX	PG10 (AF11)

It is a specific UART instance to retrieve data in case of blocking failure.

SDMMC1 configuration

Table 30. Pin configuration for SDMMC1

SDMMC1	
SDMMC1_CK	PC12 (AF10)
SDMMC1_CMD	PH2 (AF10)

SDMMC1	
SDMMC1_D0	PC8 (AF10)

SDMMC2 configuration

Table 31. Pin configuration for SDMMC2

SDMMC2	
SDMMC2_CK	PC2 (AF11)
SDMMC2_CMD	PC3 (AF11)
SDMMC2_D0	PC4 (AF11)

The same pins are used whether the default configuration is SD card or eMMC as the boot flash source. Both instances are one data bit width.

3.12.4 Configuration of hardware timers

The boot ROM code supports timeout and timestamp functions. To enable these features, the boot ROM code configures the TIMER2 hardware block. The timer operates in edge-aligned mode and upcounting mode. The timer module base is set at 1 MHz using the hardware timer prescaler.

4 Image layout

4.1 Image header layout

The image layout includes the base header and extension headers. These extension headers are the authentication extension header, the FSBL decryption extension header, and the padding extension header.

4.1.1 Base header

The base header is described as follows:

Table 32. Base header

Name	Length	Byte offset (Dec, hex)	Description	Part of signature
Magic number	32 bits	0, 0x0	‘S’; ‘T’; ‘M’; 0x32	n
Image signature	768 bits	4, 0x4	ECDSA signature: calculated on header + image	n
Image checksum	32 bits	100, 0x64	Checksum of the secured payload	n
Header version	32 bits	104, 0x68	Header version v2.3 = 0x00MMmm00MM: major version = 0x02mm: minor version = 0x03	y
Image length	32 bits	108, 0x6C	Length of FSBL image in bytes	y
Image entry point	32 bits	112, 0x70	Entry point of image	y
Reserved1	32 bits	116, 0x74	Reserved (64-bits entry point)	y
Load address	32 bits	120, 0x78	Load address of image	y
Reserved2	32 bits	124, 0x7C	Reserved (64-bits load address)	y
Version number	32 bits	128, 0x80	Image version (monotonic number)	y
Extension flags	32 bits	132, 0x84	b0=1: Authentication extension headerb1=1: FSBL encryption extension headerb31=1: Padding extension header	y
Post header length	32 bits	136, 0x88	Length in bytes of all extension headers	y
Binary type	32 bits	140, 0x90	Used to check the binary type	y
PAD	64 bits	144, 0x94	Reserved padding bytes. Must all be set to 0	y
Nonsecure payload length	32 bits	152, 0x98	Length in bytes of optional nonsecured payload	n
Nonsecure payload hash	32 msb bits	156, 0x9C	32 msb bits of SHA256 of nonauthenticated payload	n

Each extension has a type, a length, and N parameters. The byte offset is relative to the extension header base.

Table 33. Extension headers

Name	Length	Byte offset	Description	Part of signature
Extension1 header type	32 bits	0	Enumerate to determine the type of extension headerEx: 1: extension X, 2: extension Y, …	y
Extension1 header length	32 bits	4	Number of bytes of extension header	y
Extension1 param1	varies	varies	Parameters of extension1	y
…	varies	varies	-	-
Extension1 paramN	varies	varies	-	-

4.1.2 Authentication extension header

The authentication extension header is mandatory in the CLOSED_LOCKED_xxx life cycle but optional in the CLOSED_UNLOCKED life cycle. The authentication extension header is enabled when bit 0 of the “Extension flags” in the base image header is set.

The ECDSA public key field is 768 bit length, but for ECDSA256, the public key is 512 bits length. In this case, zero bytes (0x00) need to be added to extend the key from 512 bits to 768 bits. This means that if ECDSA256 is selected, the user only needs to consider the first 512-bits of this ECDSA public key field.

Table 34. Authentication extension header

Name	Length	Byte offset (Dec, hex)	Description
Extension header type	32 bits	0, 0x0	Enumerate to determine the type of extension header‘S’; ‘T’; 0x00; 0x02
Extension header length	32 bits	4, 0x4	Number of bytes of extension header116 + N * 32
Public key idx	32 bits	8, 0x8	The index of ECDSA public key is used in the table (monotonic number)
Number of public keys in table	32 bits	12, 0xC	Number of ECDSA public keys in the table (= N)
ECDSA algorithm	32 bits	16, 0x10	1: P-256 NIST; 2: brain pool 256; 3: P-384 NIST; 4: brain pool 384
ECDSA public key	768 bits	20, 0x14	ECDSA public key to be used to check the signature
Algo + ECDSA public key1 hash	256 bits	116, 0x74	Hash of algo + ECDSA public key1
…	…	…	Hashes of algo + ECDSA public keys
Algo + ECDSA public keyN hash	256 bits	116 + (N - 1) * 32	Hash of algo + ECDSA public keyN

4.1.3 Decryption extension header

The encrypted FSBL extension header is enabled when bit 1 of the “Extension flags” in the base image header is set.

Table 35. Encrypted FSBL extension header

Name	Length	Byte offset	Description
Extension header type	32 bits	0	‘S’; ‘T’; 0x00; 0x01 (FSBL encryption extension)
Extension header length	32 bits	4	Number of bytes of extension header
Key size	32 bits	8	Size of encryption key (128 or 256)
Derivation Constant	32 bits	12	Constant used to derive key from secret stored in OTP
Plain hash	128 bits	16	128 msb bits of plain payload SHA256

4.1.4 Padding extension header

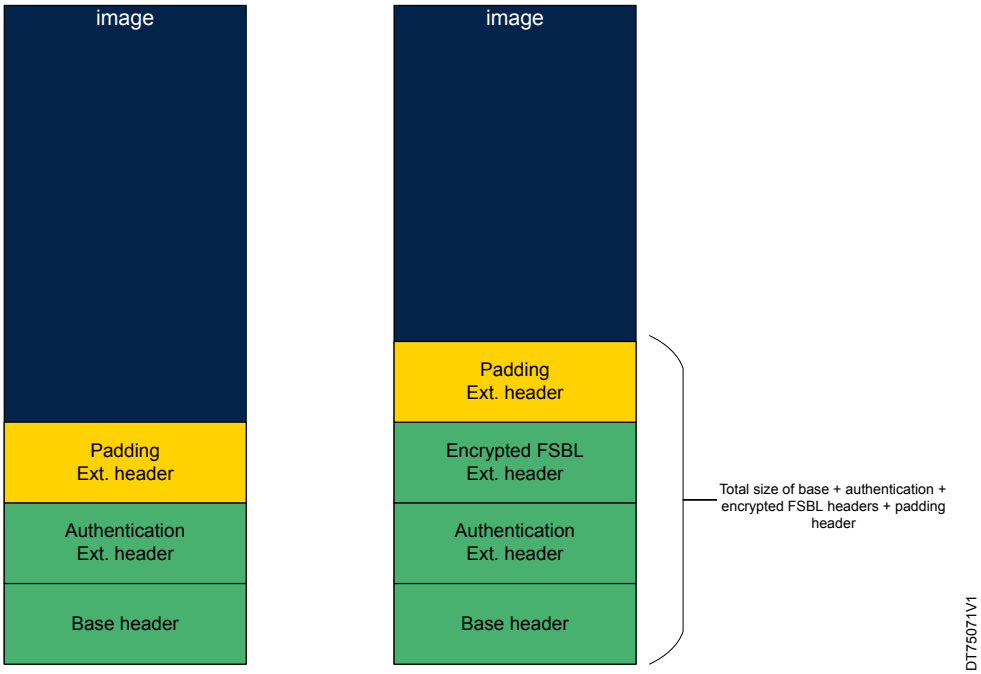
The padding extension header is enabled when bit 31 of the “Extension flags” in the base image header is set.

Table 36. Padding extension header

Name	Length	Byte offset	Description
Extension header type	32 bits	0	‘S’; ‘T’; 0xFF; 0xFF (padding extension)
Extension header length	32 bits	4	Number of bytes of extension header = N + 8
Padding bytes	N	8	Padding bytes

The padding extension header ensures that the header size is fixed.

Figure 10. Headers examples



4.2 Image nonauthenticated payload

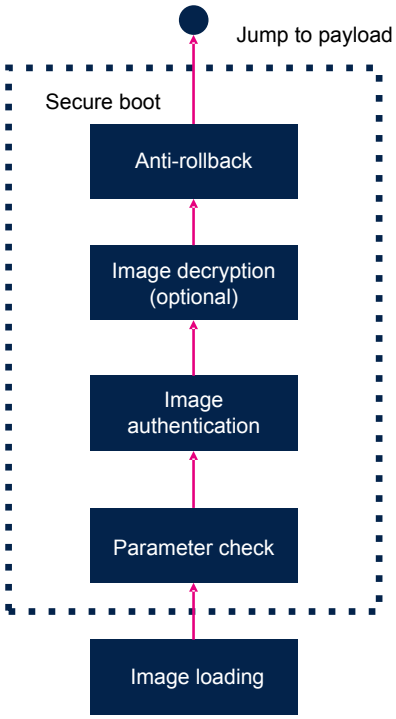
The nonauthenticated payload is neither signed, nor encrypted, and does not contain executable code. The nonauthenticated payload is following the signed image.

5 Secure boot

5.1 Secure boot overview

The boot ROM code oversees the first stage in the trust chain. To ensure this trust chain, the boot ROM code implements a secure boot that follows this sequence:

Figure 11. Boot ROM code secure boot process overview



The secure boot process is available from the CLOSED_UNLOCKED life cycle.

- If the secure boot succeeds, the boot ROM code stores the address of the boot context in the `r0` register and jumps to the FSBL entry point defined in the image header (see Section 4.1).
- If the secure boot fails, there are two possibilities depending on the life cycle:
  - **CLOSED_UNLOCKED**: The secure boot continues the process.
  - **CLOSED_LOCKED**: The secure boot stops the authentication process, cleans the download buffer, and returns to download mode to try another firmware.

5.2 Parameter checks

First, the boot ROM code checks all parameters needed for authentication:

- Check the presence of headers extension flag (the authentication extension header is mandatory in CLOSED_LOCKED life cycles).
- Verify headers parameters and geometry.
- Ensure that the image version in the base header is equal to or greater than the OTP (20-21) monotonic counter (Antiroll back version handling).

5.3 Image authentication

After checking each parameter, the image authentication process can start if there is an authentication extension or decryption header.

5.3.1 Signature verification

The boot ROM code implements two algorithm sequences:

- ECDSA 256 (NIST and Brain pool)
- ECDSA 384 (NIST and Brain pool)

The boot ROM code selects the algorithm based on the algorithm number inserted in the authentication header.

The public key used for steps 3 and 6 must match the size of the selected algorithm: 512 bits for ECDSA256 and 768 bits for ECDSA384 (see Section 4.1.2: Authentication extension header).

Figure 12. ECDSA256 signature verification

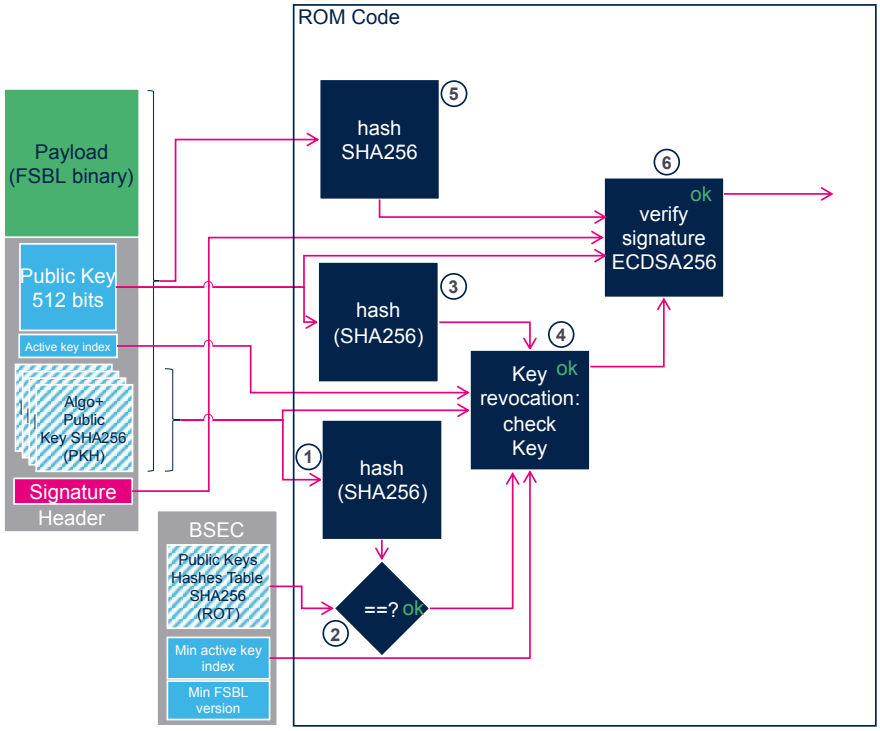
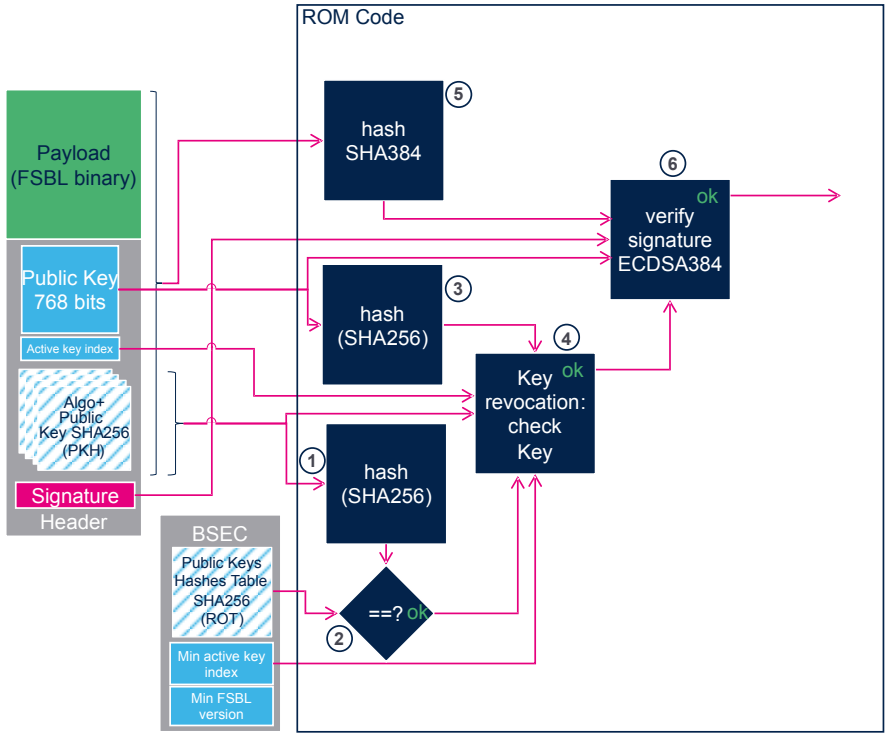


Figure 13. **ECDSA384 signature verification**



A signature verification failure in CLOSED_UNLOCKED does not stop the secure boot.

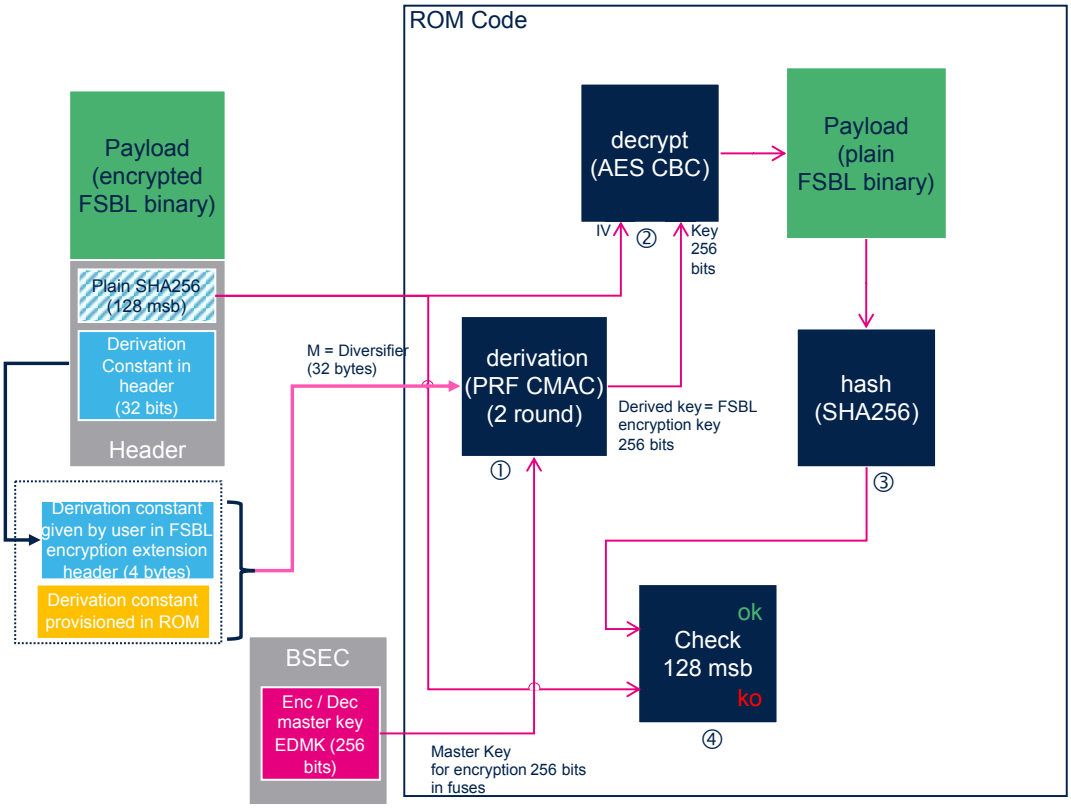
5.3.2 ECDSA key revocation

If the signature verification is successful, even in the CLOSED_UNLOCKED life cycle, the boot ROM code applies key revocation as required. This revocation is necessary if the key index used to verify the signature is higher than the key index in OTP_WORD17. In this case, the boot ROM code disables all keys from index 0 up to the new index.

5.3.3 Image decryption

This part of the authentication process is optional. The boot ROM code follows this sequence:

Figure 14. **FSBL decryption sequence**



1. *Derive the decryption key from the controller key EDMK and constant.*
2. *Decrypt the FSBL with the key and IV using SAES or CRYP (from OTP18 bit 4).*
3. *Calculate the plain FSBL SHA256.*
4. *Compare the 128 MSB bits (the last 16 bytes of SHA256).*

A failure in this part, even in CLOSED_UNLOCKED, stops the secure boot.

5.3.4 FSBL version update

If the FSBL version is higher than the OTP monotonic counter (OTP_WORD20-OTP_WORD21), the boot ROM code updates the OTP to match the FSBL version. If the version number in the header is greater than 63, the monotonic counter is updated only to its maximum value (63). An FSBL version number greater than 63 does not prevent the start of the FSBL, but the monotonic counter is no longer updated.

Note: This operation is performed only in CLOSED_LOCKED life cycles. In the CLOSED_UNLOCKED life cycle, the FSBL version update process is not performed.

Revision history

Table 37. **Revision history**

Date	Revision	Changes
19-Nov-2024	1	Initial release
12-May-2025	2	Updated: • Section 3.2.2: TrustZone® protection, RISAF, and cache handling • Section 3.2.2.2: RISAF handling • Section 3.11.1: Boot ROM related fuse settings • Section 4.1.2: Authentication extension header • Section 5.3.1: Signature verification Changed RISAF2 into RISAF3 across the document.
25-Sep-2025	3	Updated: • Table 16. USART commands • Figure 8. Boot ROM internal RAM memory layout • Table 23. Context structure elements

Contents

List of tables

List of figures

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.

In the event of any conflict between the provisions of this document and the provisions of any contractual arrangement in force between the purchasers and ST, the provisions of such contractual arrangement shall prevail.

The 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.

The 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 the 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.

If the purchasers identify an ST product that meets their functional and performance requirements but that is not designated for the purchasers' market segment, the purchasers shall contact ST for more information.

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.