
ST firmware upgrade services for STM32WB Series

Introduction

This document describes the firmware upgrade services (FUS) available on STM32WB Series microcontrollers. These services are provided by ST code located in a secure portion of the embedded Flash memory, and is used by any code running on Cortex[®]-M4 with an user Flash memory or through embedded bootloader commands (also running on Cortex[®]-M4).

1 General information

This document applies to STM32WB Series Arm®-based devices.

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



1.1 Firmware upgrade services definition

FUS (firmware upgrade services) is a firmware running on STM32WB Cortex®-M0+ and offering following features:

1. Install, upgrade or delete STM32WB Cortex®-M0+ wireless stack:
 - Only encrypted and signed by ST
 - Optionally, additionally double signed by customer if needed
2. FUS self-upgrade:
 - Only encrypted and signed by ST
 - Optionally, additionally double signed by customer if needed
3. Customer authentication key management:
 - Used for images double signature
 - Install, update and lock the customer authentication key
4. User key management:
 - Store customer keys
 - Master key
 - Simple clear key
 - Encrypted key (by master key)
 - In secure area accessible only by Cortex®-M0+ code.
 - Write stored key (simple or encrypted) into AES1 (advanced encryption standard) in secure mode (key not accessible by Cortex®-M4)
 - Lock a stored key to prevent its usage till next system reset
 - Unload a previously loaded key from AES to prevent its usage by other applications
 - Key width: 128 or 256 bits
 - Up to 100 user keys (encrypted by master key or clear) and 1 user master key
5. Communication with Cortex®-M4 (user code or bootloader):
 - Through IPCC commands and response model (same as wireless stack model)
 - Commands already supported by STM32WB bootloader (in ROM)

1.2 FUS versioning and identification

The user needs to read the shared table memory in SRAM2a to identify FUS version, as explained in [Section 1.6 Shared tables memory usage](#) and in [Section 6.1 Shared tables usage](#).

The first word in SRAM2a pointed by IPCCDBA option byte is the "Device info table" address. This table (described in [Table 5. Device information table](#)) contains the FUS version at offset 0xC which is encoded on four bytes. Typically, if IPCCDBA=0x0000 and @0x20030000 contains 0x20030024, then FUS version is @0x20030030.

Installation of a FUS image respects the conditions stated in the image binary release notes.

Table 1. FUS versions

FUS version	Description
V0.5.3	<p>Default version programmed in production for all STM32WB5xx devices.</p> <p>Must be upgraded to V1.0.1 on STM32WB5xG devices or to V1.0.2 on STM32WB5xE/5xC devices.</p> <p>This version is not available for download on www.st.com and cannot be installed by users.</p>
V1.0.1	<p>First official release available on st.com and dedicated to STM32WB5xG devices only (1-MBytes Flash memory size)</p> <p>This version must not be installed on STM32WB5xE/5xC devices, otherwise device enters a locked state and no further updates are possible.</p>
V1.0.2	<p>First official release available on www.st.com and dedicated to STM32WB5xE/5xC devices (512-KBytes and 256-KBytes Flash memory size)</p> <p>V1.0.2 can be installed on STM32WB5xG devices if the devices present FUS V0.5.3.</p> <p>If an STM32WB5xG device has FUS V1.0.1, then there is no need to upgrade to V1.0.2, since it does not bring any new feature/change vs. V1.0.1.</p> <p>In case FUS V1.0.2 installation is started by user on an STM32WB5xG device with FUS V1.0.1, FUS returns FUS_STATE_IMG_NOT_AUTHENTIC error and discard the upgrade.</p>
V1.1.0	<p>FUS update to support following features:</p> <ul style="list-style-type: none"> • Add FUS_ACTIVATE_ANTIROLLBACK command that allows activating anti-rollback on wireless stack by user. User can activate this feature in order to prevent any installation of older wireless stack. • Replace Safeboot by V1.1.0 version (replace full chip lock by factory reset) • Add factory reset in case of Flash ECC, Flash corruption or option bytes corruption error. <p>Factory reset means erase of wireless stack if present and reboot on FUS and full erase of other user sectors.</p> <p>FUS V1.1.0 can be installed only on devices containing V1.0.1 or V1.0.2 FUS.</p> <p>In case of device with V0.5.3 installed, user shall first install V1.0.2 then install V1.1.0.</p> <p>In case of device with FUS V0.5.3, installing FUS V1.1.0 will result in FUS_STATE_IMAGE_NOT_AUTHENTIC error and discarding the upgrade.</p>
V1.1.1	<p>FUS update to support STM32WB5xx 640KB sales-types.</p> <p>This version is not available on www.st.com and cannot be used for upgrade.</p> <p>This version is fully compatible with V1.1.0 and does not present any difference except management of new 640 KB salestype.</p>

1.3 How to activate FUS

The FUS runs on Cortex[®]-M0+ and on the protected Flash memory zone dedicated for FUS and wireless stack. There are two possible situations:

Table 2. FUS activation cases

Situation	How to activate FUS
No wireless stack is running (ie. first time STM32WB is running or wireless stack has been removed)	<p>Ensure Cortex[®]-M0+ is activated by setting C2BOOT bit in PWR_CR4 register</p> <p>Ensure IPCCDBA (option bytes) points to a valid shared table information structure in SRAM2a (fill correct pointers to device information table and system table)</p> <p>Note that both of these conditions are performed automatically by system bootloader. So if device boot is configured on system memory, the FUS must be activated with no need for further user actions.</p> <p>Otherwise, these actions must be performed by user code running on Cortex[®]-M4 CPU.</p>
Wireless stack is installed and running	<p>Perform same steps as above</p> <p>Request wireless stack to launch FUS by sending two consecutive FUS_GET_STATE commands. The first one must return FUS_STATE_NOT_RUNNING state and the second causes FUS to start.</p>

In order to check if FUS is running or not, the following options are available:

- Send a single FUS_GET_STATE command and check the return status. If it is FUS_STATE_NOT_RUNNING then FUS is not running.
- Check the SBRV option byte value:
 - if it is 0x3D800 (for FUS V0.5.3) or 0x3D000 (for FUS V1.x.z) then FUS must be running
 - If it is different from 0x3D800 (for FUS V0.5.3) and from 0x3D000 (for FUS V1.x.z) then FUS is not running
- Send a wireless stack command:
 - If it is acknowledged, then FUS is not running
 - If it is not acknowledged, then FUS is running
- Read the shared table information:
 - Read IPCCDBA (in option bytes) to get the shared tables start address in SRAM2a
 - Get the device information table address
 - Read the field “Last FUS active state”
 - 0x04 means that stack must be running
 - Other values mean that FUS must be running

1.4 Memory mapping

The FUS has a dedicated space in Flash memory that depends on the FUS size. It also uses a dedicated space in SRAM2a and SRAM2b and a shared space in SRAM2a (shared tables). The size of dedicated space in Flash memory, SRAM2a and SRAM2b is defined by option bytes. For more information, refer to the product reference manual.

The dedicated Flash memory and SRAM areas are shared with the wireless stack if it is installed. But at a given time, either FUS or wireless stack is running on Cortex®-M0+.

Figure 1. Flash memory mapping

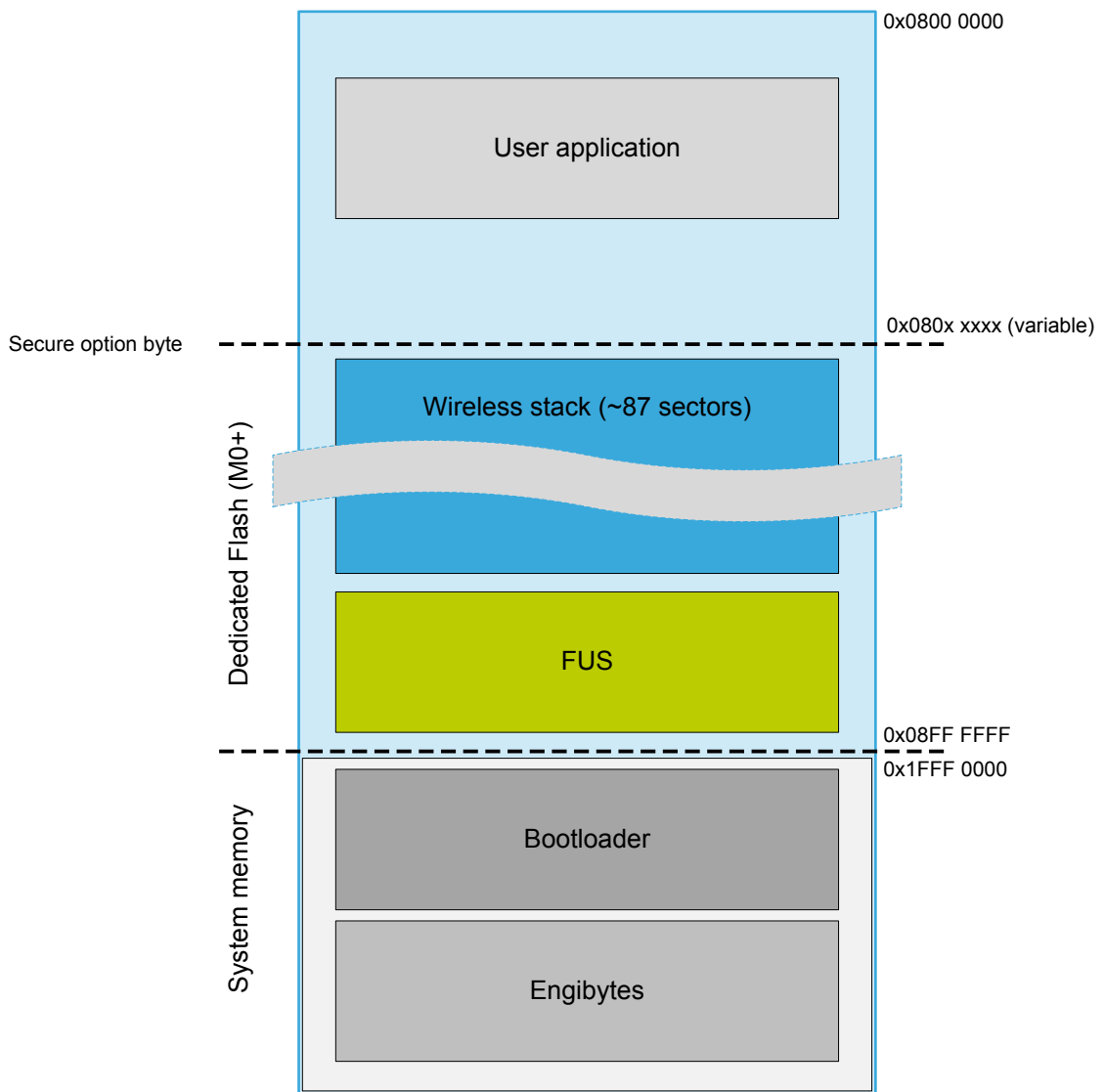
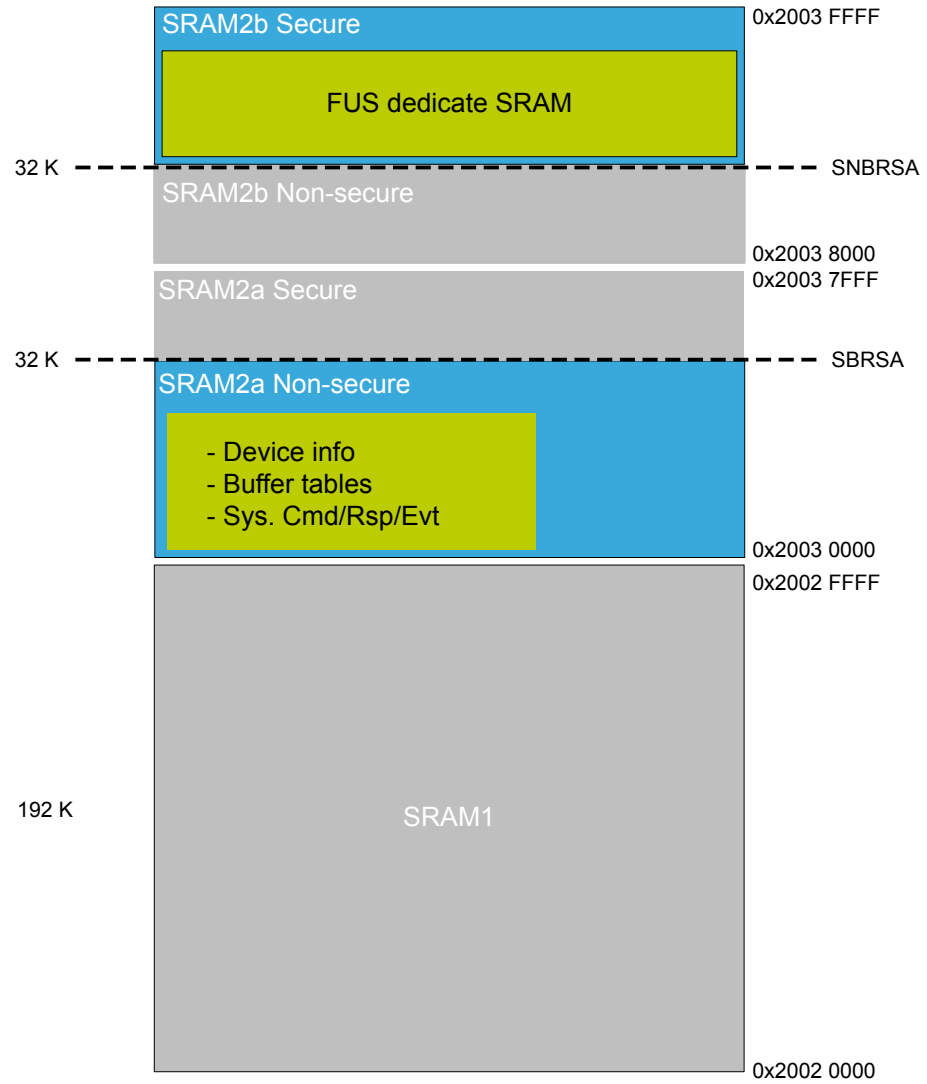


Figure 2. SRAM memory mapping



1.5 FUS resources usage

The FUS only configures / uses the resources listed in [Table 3](#).

The RCC (reset and clock control), Flash memory, PWR (power control) and all necessary components for the STM32 normal operation must be configured by Cortex[®]-M4 application prior to enabling Cortex[®]-M0+ (it is automatically done by system bootloader when started).

Table 3. FUS resources usage

Resource	Case	Configuration
Flash	Always	Dedicated Flash is used by FUS depending on its size and on the size of the current wireless stack and the image requested for install. Parts of dedicated Flash memory may be written and/or erased during FUS operations. Take care of operations performing write/erase cycles on the Flash memory while FUS is running.
SRAM2b	Always	SRAM2b secure area is used by FUS depending on its version.
SRAM2a	Always	SRAM2a secure area is used by FUS depending on its version. SRAM2a public area is used by FUS to write into shared tables for information table and commands table.
IPCC	Always	IPCC is used by FUS for mail boxing between Cortex [®] -M0+ and Cortex [®] -M4 user application or bootloader or JTAG. Two channels are used: P1CH2 (command/response channel) and P1CH4 (trace channel).
PKA	When install is required	PKA is enabled, configured and used for signature verification.
AES	When key service is required	AES1 is configured in secure mode (key register is accessible only by Cortex [®] -M0+) AES1 key register is written by FUS with the key requested by user.

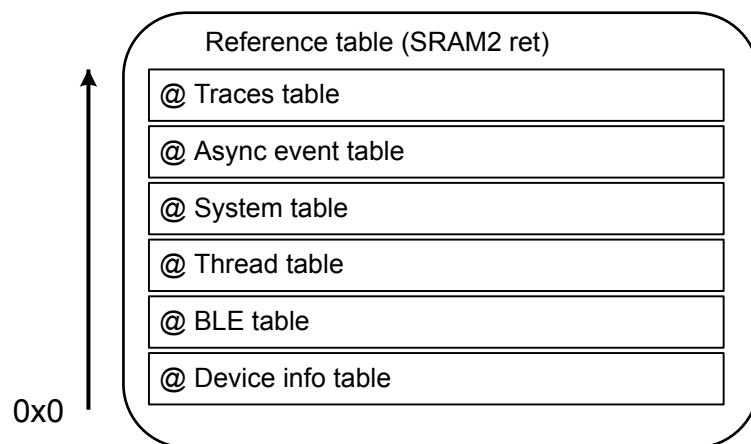
1.6 Shared tables memory usage

Communication data buffers are pointed by lookup table which address is determined by an option byte: IPCCDBA (IPCC mailbox data buffer base address). This address provides the base address of the buffer tables pointers as detailed in *Building a wireless application* (AN5289).

If IPCCDBA points to an address that does not fit all table pointers (ie. $(\text{SRAM2a_END_ADDRESS} - \text{SharedTable_BaseAddress}) < \text{SizeOf}(\text{SharedTable})$) then the FUS must discard usage of shared table completely and thus no communication or commands are possible with FUS.

User application has to setup correctly the shared table base address, otherwise it must stop FUS services initialization.

Figure 3. Shared table architecture



FUS uses only two tables:

- **Device information table:** this table allows to provide useful information from FUS to Cortex[®]-M4 user application (or JTAG) at startup (content written by FUS at startup).
- **System table:** this table allows to exchange commands and responses between FUS and Cortex[®]-M4 user application.

2 Wireless stack image operations

The FUS allows user to install, upgrade and delete the wireless stack.

The wireless stack has to be provided by ST (encrypted and signed) in order to be installed by FUS. User may add its own signature to the wireless stack image binary using the ST tools and as detailed in user authentication section (if the user authentication key has already been loaded by FUS).

Wireless stack install, upgrade and delete operations is performed through bootloader, JTAG, or user application. STM32CubeProgrammer provide the tools to perform this operation through bootloader interfaces USART and USB-DFU and also directly through SWD interface.

2.1 Wireless stack install and upgrade

Wireless stack install means first installation on a chip where there is no wireless stack already installed.

Wireless stack upgrade means installation on a chip where there is already a wireless stack installed (may be running or not).

Operation instructions

In order to perform a wireless stack install or upgrade, following steps have to be followed:

1. Download the wireless stack image from st.com or from the STM32CubeMx repository.
2. Write the wireless stack image in the user Flash memory at the address equal to:
 - If new install (no wireless stack currently installed): $0x08000000 + (\text{SFSA} \times 4\text{KB}) - \text{ImageSize}$
 - If a wireless stack is already installed:
 - If new image size \leq current image size: $\text{WirelessStackAddress} - \text{ImageSize}$
 - If new image size $>$ current image size: $\text{WirelessStackAddress} - (2 \times \text{ImageSize} - \text{WirelessStackSize})$

It is advised to perform a FUS_FW_DELETE operation before starting a wireless stack installation. (This is performed by CubeProgrammer when option -firstinstall=0 is selected)
3. Ensure FUS is running (follow steps in [Section 1.3 How to activate FUS](#))
4. Send FUS_FW_UPGRADE command through IPCC mechanism (explained in sections below)
5. Send FUS_GET_STATE till getting state equal to FUS_STATE_NOT_RUNNING (this means that the wireless stack has been installed and is now running).

During the installation process, expect multiple system resets to occur. These system resets are performed by FUS and are necessary for the modification of dedicated memory parameters and to make Cortex[®]-M0+ run the installed wireless stack. The number of system resets depends on the configuration and the location of new and old images.

The following table explains possible errors when install/upgrade operation is requested and their respective results.

Table 4. FUS upgrade returned errors

Error	Reason	Result
No enough space	Space between current installed wireless stack and the address of loaded image is too small.	Installation request is rejected. FUS return error state and go back to Idle state.
Image signature not found	Incorrect or corrupted signature header or body.	FUS returns authentication error then go back to idle state. Image is not installed and no changes on Flash memory/SRAM.
Image customer signature not found	Incorrect or corrupted signature header or body.	FUS returns authentication error then go back to idle state. Image is not installed and no changes on Flash memory/SRAM.
Image corrupted	Incorrect image header or corrupted image.	FUS returns image corruption error then go back to idle state. Image is not installed and it is erased by FUS.

Error	Reason	Result
No state is returned by FUS	A reset performed by FUS has occurred before reception of the command response.	Command resending must result in getting the response from FUS.
Other failures	External power interruption or external reset during FUS operation	FUS must be able to recover and delete the corrupted image and go back to default state. It may perform several system resets in order to complete the recovery operation.

Memory considerations

At first install or when no wireless stack is installed, the FUS does not make any optimization on the address where the wireless stack is installed. The wireless stack image must be installed at the same address where it has been loaded by user.

At wireless stack upgrade (a wireless stack is already installed), the FUS may move the upgraded stack after upgrade and before running it.

The remaining space in this case is left free for Cortex[®]-M4 user application usage.

After install/upgrade operation is done successfully, the SRAM2a, SRAM2b, Flash memory secure boundaries and SBRV value are changed according to requirements of the installed wireless stack.

2.2 Wireless stack delete

Wireless stack delete means removing the wireless stack that is already installed on a chip (may be running or not).

Operation instructions

In order to perform a wireless stack delete following steps have to be followed:

1. Ensure FUS is running (follow steps in [Section 1.3 How to activate FUS](#))
2. Send FUS_FW_DELETE command through IPCC mechanism (explained in sections below)
3. Send FUS_GET_STATE till getting state equal to FUS_STATE_NOT_RUNNING (this means that the wireless stack has been installed and is now running).

During the delete process, expect multiple system resets to occur. These system resets are performed by FUS and are necessary for the modification of dedicated memory parameters. The number of system resets depends on the configuration and the location of the wireless stack.

If no wireless stack is installed and a delete request is sent, then the FUS returns error state informing that no wireless stack was found (FUS_STATE_IMG_NOT_FOUND).

Memory considerations

After the delete operation is done successfully, all the space used by wireless stack becomes free for usage by Cortex[®]-M4 user application or for further wireless stack install operations.

Image start address must be aligned to sector start (ie. multiple of 4-KBytes) and the image size must be multiple of 4 bytes, otherwise, FUS rejects the installation procedure.

2.3 Wireless stack start

It is possible that a wireless stack is installed but not currently running (ie. installation done, wireless stack is running and then user application send two consecutive FUS_GET_STATE which leads to FUS to be started again).

In that case, it is possible to launch the wireless stack execution by sending FUS_START_WS command. This command allow switching Cortex[®]-M0+ execution to wireless stack and results in at least one system reset.

The command is completed when FUS_GET_STATE returns FUS_STATE_NOT_RUNNING value. On reception of this value, no other FUS_GET_STATE must be issued, otherwise the FUS is executed again.

2.4 Anti-rollback activation

When FUS support anti-rollback, it is possible to activate this feature by sending a command to the FUS.

When this command is executed by FUS, it is no more possible to deactivate it by any means.

This feature is executed through FUS_ACTIVATE_ANTIROLLBACK command.

After sending this command it is possible to check its status by sending FUS_GET_STATE command.

The FUS shall then return the state FUS_STATE_IDLE.

This command is not reversible.

This command do not apply to FUS since no rollback is possible on FUS anyway.

3 FUS upgrade

The FUS is capable of self-upgrade in the same way as wireless stack upgrade. Deleting FUS is not possible.

3.1 Operation instructions

In order to perform a FUS upgrade, following steps have to be followed:

- Download the FUS image from st.com or from the STM32CubeMx repository.
- Write the FUS image in the user Flash memory at the address indicated in the FUS image directory Release_Notes.html file.
- Ensure FUS is running (follow steps in [Section 1.3 How to activate FUS](#)).
- Send FUS_FW_UPGRADE command through IPCC mechanism (explained in sections below).
- Send FUS_GET_STATE till getting state equal to FUS_STATE_NOT_RUNNING (this means that the wireless stack has been installed and is now running).

During the installation process, expect multiple system resets to occur. These system resets are performed by FUS and are necessary for the modification of dedicated memory parameters and to make Cortex®-M0+ run the installed wireless tack. The number of system resets depends on the configuration and the location of new and old images.

FUS identifies the image as FUS upgrade image and launches the FUS upgrade accordingly. This operation might result in a relocation of the firmware stack if it is already installed and if the size of the new FUS is larger than the size of the current FUS. This information and any relative constraints are detailed in the FUS image release note.

3.2 Memory considerations

The FUS upgrade requires no specific memory conditions. But if the new FUS image size is larger than existing FUS size, the upgrade may result in moving the wireless stack lower in Flash memory in order to grant sufficient space for FUS upgrade.

This means that:

- Less Flash memory is available for Cortex®-M4 user application.
- The wireless stack is moved from its current address to another address defined by FUS.
- If a user code is written in the sectors neighboring wireless stack start sector, they risk to be erased during this operation.

The size of the FUS and results of its upgrade are detailed in its Release_Notes.html file.

Image start address must be aligned to sector start (ie. multiple of 4 KBytes) and the image size must be multiple of 4 bytes, otherwise, FUS rejects the installation procedure.

4 User authentication

The FUS services allow user to add his own signature to any image (wireless stack or FUS image) provided by ST (encrypted and signed by ST).

The instruction to sign a binary with user authentication key are provided in STM32CubeProgrammer user manual.

FUS checks on the user signature only if a user authentication key has already been installed.

The signature is a 64 bytes data buffer based on RSA ECC Prime256v1 (NIST P-256) and HASH-256. It is generated by STM32CubeProgrammer tool.

4.1 Install user authentication key

FUS allows storing a user authentication key through following steps:

1. Ensure FUS is running (follow steps in [Section 1.3 How to activate FUS](#)).
2. Send FUS_UPDATE_AUTH_KEY command through IPCC mechanism (explained in sections below)
3. Send FUS_GET_STATE till getting state equal to FUS_STATE_IDLE.

This operation does not generate any system resets.

Once the user authentication key is installed, it is changed (unless lock user authentication key operation is done) using same flow above. But it cannot be removed.

Once it is installed, FUS must systematically check on the binary user signature before performing the installation or upgrade. If the signature is not present or if it is not authentic, the install or upgrade is rejected with error equal to FUS_STATE_IMG_NOT_AUTHENTIC.

4.2 Lock user authentication key

FUS allows locking the user authentication key. It means that this key can no more be changed during the entire product life cycle. There is no way to undo this operation once performed.

To lock user authentication key:

1. Ensure FUS is running (follow steps in [Section 1.3 How to activate FUS](#)).
2. Send FUS_LOCK_AUTH_KEY command through IPCC mechanism (explained in sections below).
3. Send FUS_GET_STATE till getting state equal to FUS_STATE_IDLE.

This operation does not generate any system resets.

Once this operation is done, the user authentication key is locked.

5 Customer key storage

The FUS allows customer keys to be stored in the dedicated FUS Flash memory area and then to load the stored key to the AES1 in secure mode (AES1 key register accessed only by Cortex[®]-M0+ and data registers accessible by Cortex[®]-M4 user application).

5.1 Key types and structure

FUS supports storing 101 keys (1 master key and 100 clear/encrypted keys)

Key size can be 128 or 256 bits. The key size and structure is the same for all type of keys. Any stored key cannot be changed or removed.

FUS supports three key types:

- **Clear key:** a key sent to FUS in clear (not encrypted).
- **Master key:** a key sent to FUS in clear and used to decrypt other keys to be sent to FUS later. The storage of this key must be done in a trusted environment (where the key cannot be extracted on the communication path). It allows user to share encrypted keys in untrusted environments without exposing their content. Master key cannot be written in AES1 key register. It is exclusively used for decryption and cannot be changed or removed. Master key can be written only once and can never be updated afterwards. Once the master key is written, any request to write master key again is rejected with error message.
- **Encrypted key:** a key that is sent to FUS in encrypted format. It is then decrypted by FUS using the master key before using it. This key must be accompanied by an IV (initialization vector) allowing its decryption by FUS. 16-bit IV is sent in the same command packet as the key itself.

The user key encryption must be based on AES-128 GCM mode. The FUS decrypt the key without using the hardware AES.

The key type must be communicated to FUS in the command packet where the key is sent (more details in commands description).

Keys are managed through their index:

When a key is sent to FUS, FUS acknowledge reception and responds with the key allocated index. This index is assigned by FUS and cannot be changed by user application.

To store a key, the user application must send FUS_STORE_USR_KEY to FUS (with key type and relative IV if any) and then receive key index.

To use the stored key, the user application must:

- Configure AES1 initialization registers and IV register.
- Send FUS_LOAD_USR_KEY to FUS and wait for response to be received which means the key has been written in AES1 key register.
- Write in AES1 data register to decrypt/encrypt data using the stored key. (the key register remains protected and cannot be accessed by Cortex[®]-M4 user application). If more than 100 keys is written, results that command is rejected.

There are two additional services provided by FUS for user keys management. These two services are intended for use by the Cortex-M4 user application in the context of a secure application and they are not exposed by bootloader or ST tools (ie. STM32CubeProgrammer).

- **User key lock:**
 This service ensures a key cannot be used anymore by any application (cannot be loaded into AES) till next device reset. It is possible to use this service by sending FUS_LOCK_USR_KEY command containing the index of the key to be locked (Master key index is always 0 and it cannot be locked neither loaded).
 When FUS_LOCK_USR_KEY command is sent, FUS stores the state of the requested key as locked and issuing any FUS_LOAD_USR_KEY for that key index results in operation fail (0x01 returned by the command response).

- **User key unload:**
This service is used to unload the currently key loaded in AES (if FUS_LOAD_USR_KEY has been used) and prevent any further operation using the loaded key by user application.
It is possible to use this service by sending FUS_UNLOAD_USR_KEY command containing the index of the key to be unloaded (Master key index is always 0 and it cannot be loaded neither unloaded).
When FUS_UNLOAD_USR_KEY is sent, FUS writes zeros into the key registers of the AES and thus the loaded key cannot be used anymore.

6 Communication with FUS

Communication with FUS is performed through IPCC channels and by Cortex[®]-M4 user application or by bootloader or by JTAG. In all cases, the principles of communication are exactly the same.

Using STM32 system bootloader to communicate with FUS provides abstraction of all the low layer by directly using bootloader interfaces (USART or USB-DFU).

To communicate with the FUS, there are two elements to be used:

- Shared tables: used to store information of FUS and to get the command/response packets.
- IPCC: used to exchange messages notifications (messages content is located in the shared tables).

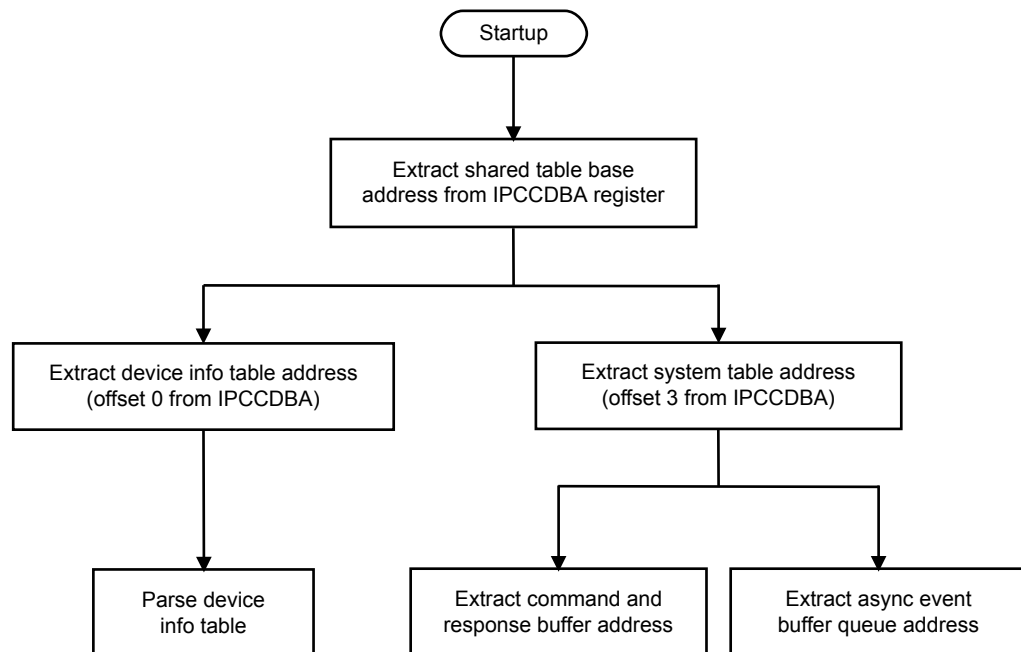
6.1 Shared tables usage

Shared tables are an information structure located in SRAM2a public area and which structure is explained in . FUS uses two shared tables:

- Device information table
- System table

Both of them must be parsed by the Cortex[®]-M4 user application (or JTAG application) in order to correctly communicate with FUS.

Figure 4. Shared table usage process



6.1.1 Device information table

Device information table is 42-byte buffer used to update current status of the device.

This table may be updated either by FUS code or wireless stack code at startup or before a programmed system reset.

Table 5. Device information table

Field	Size (bytes)	Values
Device info table state	4	0xA94656B9: Device info table valid Any other value: Device info table not valid
Reserved	1	Reserved
Last FUS active state	1	<ul style="list-style-type: none"> • 0x00: FUS idle • 0x01: Wireless stack firmware upgrade • 0x02: FUS firmware upgrade • 0x03: FUS service • 0x04: Wireless stack running • 0x05-0xFE: Not used • 0xFF: Error
Last wireless stack state	1	0x00: Not Started 0x01: Running 0x08-0xFE: Not used 0xFF: Error
Current wireless stack type	1	0x00 : None 0x01 : BLE 0x02 : Thread type1 0x03 : Thread type2 More details available in wireless stack documentation.
Safe boot version	4	Firmware version: [24:31]: Major (updated when backward compatibility is broke) [16:23]: Minor (updated when a major feature is added) [8:15]: Sub-version (updated for minor changes) [4:7]: Branch (specific build) [0:3]: Build (build version)
FUS version	4	Firmware version : [24:31]: Major (updated when backward compatibility is broke) [16:23]: Minor (updated when a major feature is added) [8:15]: Sub-version (updated for minor changes) [4:7]: Branch (specific build) [0:3]: Build (build version)
FUS memory size	4	Current FUS stack memory usage: [24:31]: SRAM2b number of 1 K sectors used [16:23]: SRAM2a number of 1 K sectors used [8:15]: Reserved [0:14]: Flash memory number of 4 K sectors used
Wireless stack version	4	Firmware version: [24:31]: Major (updated when backward compatibility is broke) [16:23]: Minor (updated when a major feature is added) [8:15]: Sub-version (updated for minor changes) [4:7]: Branch (specific build) [0:3]: Build (build version) When no stack present, all data is 0xFFFF FFFF

Field	Size (bytes)	Values
Wireless stack memory size	4	Current wireless stack memory usage: [24:31]: SRAM2b number of 1 K sectors used [16:23]: SRAM2a number of 1 K sectors used [8:15]: Reserved [0:14]: Flash memory number of 4 K sectors used When no stack present, all data is 0xFFFF FFFF
Wireless FW-BLE info	4	[0:31]: Reserved for wireless stack usage When no stack present, all data is 0xFFFF FFFF
Wireless FW-thread info	4	[0:31]: Reserved for wireless stack usage When no stack present, all data is 0xFFFF FFFF
Reserved	4	0x00000000
UID64	8	STM32 device unique ID 64-bit
Device ID	2	STM32 generic device ID

6.1.2 System table

System table is an 8-byte table containing two buffer pointers, described in table below.

Table 6. System table content

Address	Size (bytes)	Content	Description
0x00	4	Address of system command/response buffer	A single buffer is used as at a given time, only a command or its response must be written. Response overwrites the command. The new command overwrites any previous command's response.
0x04	4	Address of system events queue buffer (address of first event)	FUS code has to parse and fill the queue when necessary. Events messages are managed as chained list and are freed once Cortex [®] -M4 has read them (notification through IPCC). Parsing of the event is done through their size only. (not chained list structure),

In order to get useful information to communicate with FUS, the Cortex[®]-M4 code (application or bootloader) perform parsing as described in [Figure 4](#).

6.2 IPCC usage

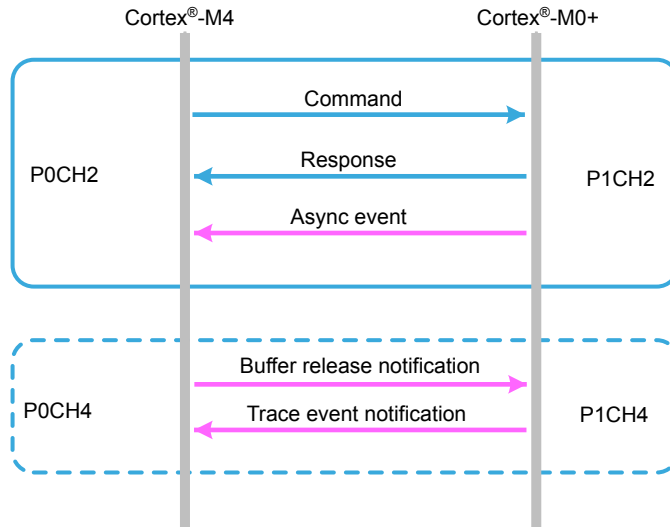
FUS uses system IPCC allocated channels: P0CH2 (on Cortex[®]-M4 side) and P1CH2 (on Cortex[®]-M0+ side). These channels offer three communication ways:

- Cmd: Command request from Cortex[®]-M4 to Cortex[®]-M0+. This way is used to send a command to Cortex[®]-M0+.
- Rsp: Response to command from Cortex[®]-M0+ to Cortex[®]-M4. This way is used only to answer a command requested by Cortex[®]-M4.
- Asynch Evt: Asynchronous event from Cortex[®]-M0+ to Cortex[®]-M4. This way is used to inform Cortex[®]-M4 about an asynchronous event, without answer required from Cortex[®]-M4 on this event.

There are optional channels that may be used by FUS:

- P1CH4 may be used by FUS (Cortex[®]-M0+) to output traces events
- P0CH4 may be used by Cortex[®]-M4 in order to notify Cortex[®]-M0+ about buffer release events.

Figure 5. IPCC channels used by FUS

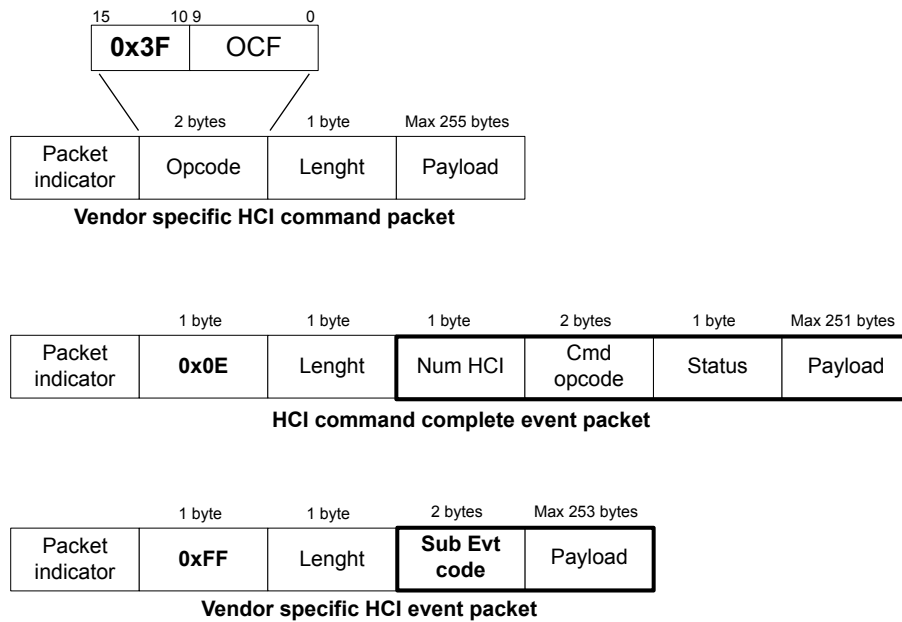


6.3 FUS commands

FUS uses same commands/response structure as wireless stacks and based on HCI model. FUS uses a subset of the HCI commands, namely:

- Vendor specific HCI command packet: used to send command from Cortex[®]-M4 to Cortex[®]-M0+.
- HCI command complete event packet: used to send response from Cortex[®]-M0+ to Cortex[®]-M4
- Vendor specific HCI event packet: used to send asynchronous events from Cortex[®]-M0+ to Cortex[®]-M4.

Figure 6. FUS HCI subset



6.3.1 Packet indicators

Packet indicator is one byte and its value depends on the packet type.

Table 7. Packet indicator values

Packet type	Packet indicator value
Command packet	0x10
Response packet	0x11
Event packet	0x12

6.3.2 Event packet

Only one asynchronous event is sent by FUS. It is sent only at startup of the FUS. The length field represent the length of SubEvtCode+Payload.

Table 8. FUS asynch event (vendor specific HCI event)

Length	SubEvtCode	Payload	Meaning
3	0x9200	Error code: 0x00: Wireless stack running 0x01: FUS running 0x02: SW Error 0x03..0xFF: Not used	FUS initialization phase done and the error code presented in payload byte.

6.3.3 Command packet

Table below details all commands supported by FUS and their HCI format values.

Table 9. FUS commands (vendor specific HCI command packet)

Command	Opcode	Length (bytes)	Payload
Reserved	0xFC00	N/A	N/A
FUS_GET_STATE	0xFC52	0	None
Reserved	0xFC53	N/A	N/A
FUS_FW_UPGRADE	0xFC54	0 / 4 / 8 ⁽¹⁾ ₍₂₎	None (Optional 4 bytes) address of the firmware image location (Optional 8 bytes) address of the firmware destination
FUS_FW_DELETE	0xFC55	0	None
FUS_UPDATE_AUTH_KEY	0xFC56	Up to 65	Byte0: Authentication key size N in bytes Byte1-ByteN-1: Authentication key data
FUS_LOCK_AUTH_KEY	0xFC57	0	None
FUS_STORE_USR_KEY	0xFC58	N+2	Byte0: Key type: • 0x00:None • 0x01:Simple key • 0x02: Master key • 0x03: Encrypted key Byte1: Key size N in bytes Byte2-ByteN-1: Key data (key value + IV if any)

Command	Opcode	Length (bytes)	Payload
FUS_LOAD_USR_KEY	0xFC59	1	Byte0: Key index (from 0 to 124)
FUS_START_WS	0xFC5A	0	None
FUS_LOCK_USR_KEY	0xFC5D	1	1 byte, index of the key to be locked
FUS_UNLOAD_USR_KEY	0xFC5E	1	1 byte, index of the key to be unloaded
FUS_ACTIVATE_ANTIROLLBACK	0xFC5F	0	None
Reserved	0xFC60-0xFCFF	N/A	N/A

1. 4 bytes, not used in current version.

2. 8 bytes, not used in current version.

6.3.4 Response packet

For each command packet, a response packet is sent by FUS containing information detailed in table below. The NumHCI field value is always set to 0xFF.

The length field indicates the length of NumHCI+CmdOpcode+Status+Payload. So if there is no payload, the length value is four.

Table 10. FUS responses (HCI command complete packet)

Status	Length	Cmd Opcode value	Status value	Payload
FUS_STATE	5	0xFC52	Values in table FUS state values	Values in table FUS state error values.
FW_UPGRADE_STATE	4	0xFC54	0x00: Operation started	None
FW_DELETE_STATE	4	0xFC55	0x01: Fail 0x02-0xFF : Not used	None
UPDATE_AUTH_KEY_STATE	4	0xFC56		None
LOCK_AUTH_KEY_STATE	4	0xFC57	0x00: Operation done	None
STORE_USR_KEY_STATE	5	0xFC58	0x01: Fail 0x02-0xFF: Not used	One byte: Stored key index (from 0 to 100)
LOAD_USR_KEY_STATE	4	0xFC59		None
FUS_START_WS_START	4	0xFC5A	0x00 : Operation started 0x01 : Fail 0x02-0xFF : Not Used	None
FUS_LOCK_USR_KEY	4	0xFC5D	0x00: Operation done, 0x01: Fail, 0x02-0xFF: Not used	None
FUS_UNLOAD_USR_KEY	4	0xFC5E	0x00: Operation done, 0x01: Fail, 0x02-0xFF: Not used	None
FUS_ACTIVATE_ANTIROLLBACK	4	0xFC5F	0x00: Operation done, 0x01: Fail, 0x02-0xFF: Not used	None

FUS response state values are detailed in table below. Some values are represented as a range (ie. 0x10..0x1F), which means all values from that range provide same state meaning. (ie. 0x12 or 0x1E both mean FUS_STATE_FW_UPGRD_ONGOING). This range of values is reserved for future extensions of the protocol.

Table 11. FUS state values

Value	Name	Meaning
0x00	FUS_STATE_IDLE	FUS is in idle state. Last operation done successfully and returned its state. No operation is ongoing.
0x01..0x0F	Not used	These values are reserved for future use.
0x10..0x1F	FUS_STATE_FW_UPGRD_ONGOING	The firmware upgrade operation is ongoing.
0x20..0x2F	FUS_STATE_FUS_UPGRD_ONGOING	The FUS upgrade operation is ongoing.
0x30..0x3F	FUS_STATE_SERVICE_ONGOING	A service is ongoing: Authentication key service (update/lock) or user key service (store/load).
0x40..0xFE	Not Used	These values are reserved for future use.
0xFF	FUS_STATE_ERROR	An error occurred. For more details about the error origin, refer to the response payload.

Table 12. FUS state error values

Value	Name	Meaning
0x00	FUS_STATE_NO_ERROR	No error occurred.
0x01	FUS_STATE_IMG_NOT_FOUND	Firmware/FUS upgrade requested but no image found. (ie. image header corrupted or Flash memory corrupted)
0x02	FUS_SATE_IMC_CORRUPT	Firmware/FUS upgrade requested, image found, authentic but not integer (corruption on the data)
0x03	FUS_STATE_IMG_NOT_AUTHENTIC	Firmware/FUS upgrade requested, image found, but its signature is not valid (wrong signature, wrong signature header)
0x04	FUS_SATE_NO_ENOUGH_SPACE	Firmware/FUS upgrade requested, image found and authentic, but there is no enough space to install it due to already installed image. Install the stack in a lower location then try again.
0x05	FUS_IMAGE_USRABORT	Operation aborted by user or power off occurred
0x06	FUS_IMAGE_ERSEERROR	Flash Erase Error
0x07	FUS_IMAGE_WRTERROR	Flash Write Error
0x08	FUS_AUTH_TAG_ST_NOTFOUND	ST Authentication tag not found error in the image
0x09	FUS_AUTH_TAG_CUST_NOTFOUND	Customer Authentication tag not found in the image
0x0A	FUS_AUTH_KEY_LOCKED	The key that user tries to load is currently locked
0x11	FUS_FW_ROLLBACK_ERROR	Rollback to older version of FW detected and not allowed
0x12..0xFD	N/A	Reserved for future use.
0xFE	FUS_STATE_NOT_RUNNING	FUS is not currently running. wireless stack is running and returned this state.
0xFF	FUS_STATE_ERR_UNKOWN	Unknown error

6.4 Image footers

Each element of the image upgrade has its own footer:

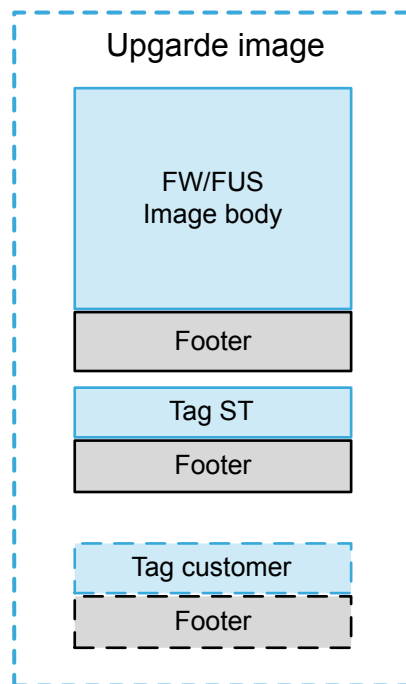
- The image body
- The ST signature (mandatory element)
- The customer signature (optional element)

The footer must directly follow the end of their relative element as a footer (ie. the image body header address must be exactly at the end of the Image body)

The authentication tags do not have to be collated to the image. They are located anywhere in the user Flash memory. FUS looks for them independently of the image location.

All images and footers addresses and sizes must be four bytes multiple and four bytes aligned, otherwise, they cannot be recognized by FUS.

Figure 7. Image footers placement



Each footer contain an identification value allowing FUS to recognize it.

Figure 8. FW/FUS upgrade image footer structure

Info1 (i.e. BLE)	Data																															
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Info2 (i.e. thresd)	Data																															
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Memory size	SRAM2b (nb of 1 K sectors)								SRAM2a (nb of 1 K sectors)								Reserved								Flash (nb of 4 K sectors)							
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Version	Version major								Version major								Subversion								Branch				Build			
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Magic number	Magic number																															
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Table 13. Parsing of image footer structure

Field	Meaning
Info1	Specific to wireless stack / FUS image
Info2	Specific to wireless stack / FUS image
Flash memory	Image total size expressed as multiple of 4 KBytes
SRAM2a	Image total required space in SRAM2a secure area
SRAM2b	Image total required spec in SRAM2b secure area
Build	Version build number
Branch	Version branch number
SubVersion	Version subversion number
VersionMinor	Version minor number
VersionMajor	Version major number
Magic Number	Specific value allowing to identify the nature of the image.

Figure 9. Signature (tag) footer structure

Reserved	Reserved																															
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved	Reserved																															
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Memory size	Reserved								Reserved								Source (ST/Cust.)								Size in bytes							
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Version	Version major								Version major								Subversion								Branch				Build			
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Magic number	Magic number																															
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Table 14. Parsing of signature footer

Field	Meaning
Reserved	Not used in this version
Size	Signature total size in bytes (without footer)
Source	Signature nature: 0x00: ST signature 0x01: Customer signature
Build	Version build number
Branch	Version branch number
SubVersion	Version subversion number
VersionMinor	Version minor number
VersionMajor	Version major number
Magic Number	Specific value allowing to identify the nature of the image.

The magic number values allowing to identify the image nature are detailed in table below:

Table 15. Magic number values

Value	Nature
0x23372991	Wireless stack image
0x32279221	FUS Image
0xD3A12C5E	ST signature
0xE2B51D4A	Customer signature
0x42769811	Other firmware image

7 STM32 system bootloader extension for FUS

A command set extension has been added to STM32WB system bootloader in order to support FUS operation. These commands are implemented on USART and USB-DFU interfaces and follow the same rules as existing standard bootloader commands.

In order to help to understand this section, a prior reading of *STM32 microcontroller system memory boot mode* (AN2606) and *USART protocol used in the STM32 bootloader* (AN3155) and *USB DFU protocol used in the STM32 bootloader* (AN3156) documentation is required.

7.1 USART extension

Two commands have been added to bootloader USART standard protocol in order to support the FUS extension. All FUS commands are passed through these two special commands: one for writing (used for all FUS commands from host to FUS) and one for reading (used for all FUS commands from FUS to host).

Table 16. Bootloader USART commands extension

Command	Opcode	Usage
Special read command	0x50 (complement 0xAF)	Get data from FUS
Special write command	0x51 (complement 0xAE)	Send data to FUS

Note: For bootloader, the following commands added in FUS are not supported (neither on UART nor USB DFU)"

- `FUS_LOCK_USR_KEY`
- `FUS_UNLOAD_USR_KEY`
- `FUS_ACTIVATE_ANTIROLLBACK`

Lock and Unload user key are two commands that are meant for use by Cortex-M4 user application only. Activate anti-rollback can be used either by implementing it in Cortex-M4 user application code, or by using STM32CubeProgrammer features or by using STM32 open bootloader example code.

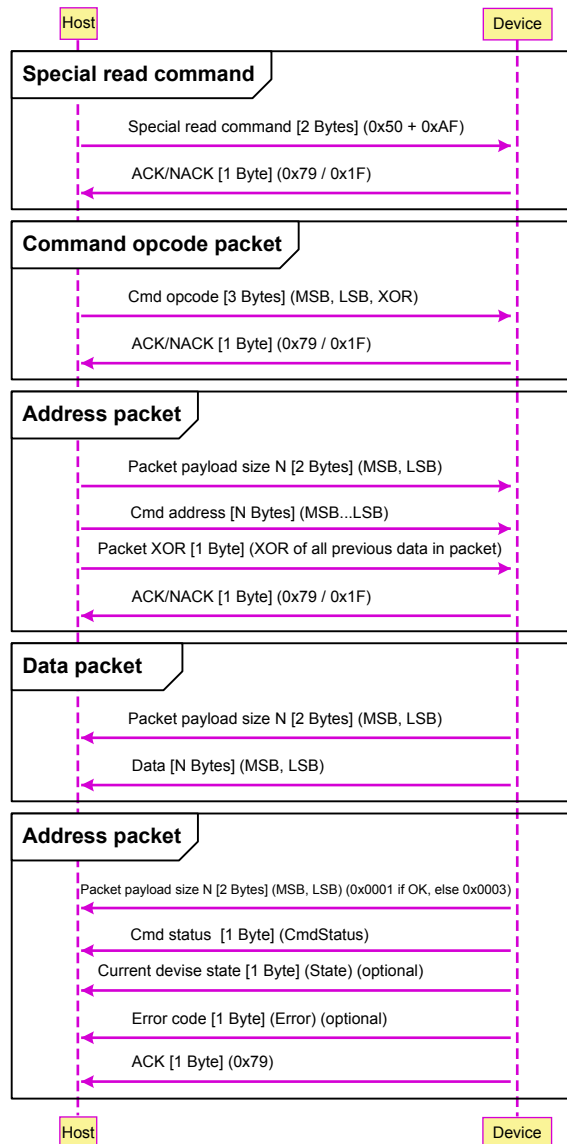
7.1.1 USART special read

Special read command is used to perform FUS command sending requesting data from device. It is divided into five separate packets:

- Special read command packet:
 - Host sends the special read command code and complement (0x50, 0xAF) and waits for ACK/NACK byte. In case of NACK, it means the command is not supported.
- Command opcode packet
 - Host sends the command packet containing:
 - FUS command opcode (2 bytes)
 - XOR of the FUS command opcode (2 bytes)
 - Device sends ACK if opcode is supported. NACK otherwise.
- Address packet:
 - Host sends address packet payload size on two bytes (MSB first).
 - Host sends address payload bytes (MSB first).
 - Host sends packet XOR value (checksum of all previous bytes in current packet, 1 byte).
 - Device sends ACK if data are correct and supported. NACK otherwise.
- Response data packet: (optional)
 - Device sends packet data payload size in bytes on 2 bytes (MSB first).
 - Device sends data payload bytes (MSB first). Some commands require not data payload.

- Response status packet:
 - Device sends packet payload size in bytes on 2 bytes (MSB first).
 - Device sends command status on one byte (status of current command requested by host).
 - Device sends current device state on 1 byte (optional, if payload size > 3).
 - Device sends current command error code (or key index) on 1 byte.
 - Device sends ACK to signal end of response packet.

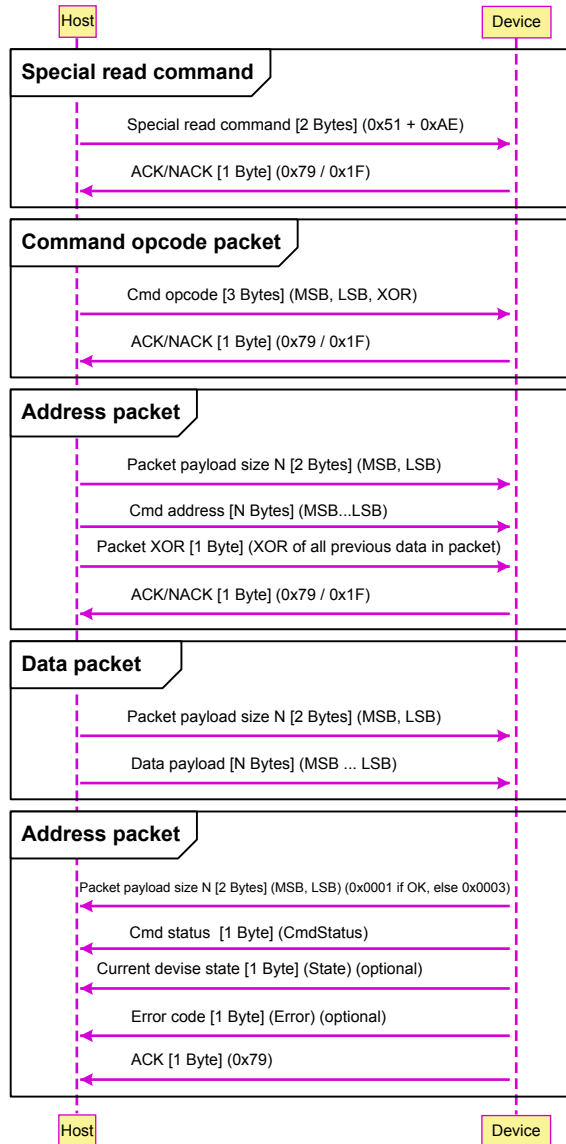
Figure 10. USART special read command



7.1.2 USART special write

Special write command is used to perform FUS command sending requesting data from device. It is divided into four separate packets:

- Special write command packet:
 - Host sends the special write command code and complement (0x51, 0xAE) and waits for ACK/NACK byte. In case of NACK, it means the command is not supported.
- Command opcode packet:
 - Host sends the command packet containing:
 - FUS Command Opcode (2 bytes)
 - XOR of the FUS command opcode (2 bytes)
 - Device sends ACK if opcode is supported. NACK otherwise.
- Address packet:
 - Host sends address packet payload size on two bytes (MSB first).
 - Host sends address payload bytes (MSB first).
 - Host sends packet XOR value (checksum of all previous bytes in current packet, 1 byte).
 - Device sends ACK if data are correct and supported. NACK otherwise.
- Data packet:
 - Host sends packet data payload size in bytes on 2 bytes (MSB first). This number may be zero when no data is needed for the command.
 - Host sends data payload bytes (MSB first). No data are sent if payload size is zero.
 - Host sends packet XOR value (checksum of all previous bytes in current packet, 1 byte).
 - Device sends ACK if data are correct and supported. NACK otherwise.
- Response packet:
 - Device sends packet payload size in bytes on 2 bytes (MSB first).
 - Device sends command status on one byte (status of current command requested by host).
 - Device may send current device state on 1 byte (optional, if payload size > 1).
 - Device sends current command error code on 1 byte (optional, if payload size > 1).
 - Device sends ACK to signal end of response packet.

Figure 11. USART special write command


7.1.3 USART FUS commands mapping

There is only one FUS command mapped on special read command.

Table 17. USART FUS command mapping on read command

Command	Opcode	Address packet	Data packet	Cmd status packet
FUS_GET_STATE	0x54	Size = 0x0000 Data = None	Size = 0x0003 Data = [0x00, FUS_STATE, ErrorCode]	Size = 0x0001 or 0x0003 Data = [0x00] if OK or [0x01, state, error] if KO

There are seven FUS commands mapped on special write command.

Table 18. USART FUS command mapping on write command

Command	Opcode	Address packet	Data packet	Cmd status packet
FUS_FW_DELETE	0x52	Size = 0x0000 Data = None	Size = 0x0000 Data = None	Size = 0x0001 or 0x0003 Data = [0x00] if OK or [0x01, state, error] if KO
FUS_FW_UPGRADE	0x53	Size = 0x0000 Data = None	Size = 0x0000 Data = None	
FUS_UPDATE_AUTH_KEY	0x56	Size = 0x0000 Data = None	Size = up to 65 Data = Key (1 byte key size + 64 bytes key data)	
FUS_LOCK_AUTH_KEY	0x57	Size = 0x0000 Data = None	Size = 0x0000 Data = None	
FUS_STORE_USR_KEY	0x58	Size = 0x0000 Data = None	Size = up to 34 Data = [KeyType (1byte), KeySize(1byte), KeyData (16/32bytes)]	Size = 0x0003 Data = [0x00, state, KeyIndex]
FUS_LOAD_USR_KEY	0x59	Size = 0x0000 Data = None	Size = 0x0001 Data = [KeyIndex]	Size = 0x0001 or 0x0003
FUS_START_WS	0x5A	Size = 0x0000 Data = None	Size = 0x0000 Data = None	Data = [0x00] if OK or [0x01, state, error] if KO

7.2 USB-DFU extension

FUS commands are processed over bootloader USB-DFU standard download and upload commands.

7.2.1 USB-DFU download FUS extension

Bootloader USB-DFU download FUS extension is managed in same way as SET_ADDRESS_POINTER and ERASE standard commands: Value = 0 and following bytes are command data MSB first.

Exception is made for FUS_STORE_USR_KEY which is split over two steps:

- Step1: Download command, only allows to send the key data (up to 34 bytes)
- Step2: Upload command, must be done after download step and allows to get the key index (1 byte)

Table 19. USB-DFU download extension

Command	Opcode	Data
FUS_FW_DELETE	0x52	None
FUS_FW_UPGRADE	0x53	None
FUS_UPDATE_AUTH_KEY	0x56	Key Buffer = [KeySize (1byte), KeyData (64bytes MSB first)]
FUS_LOCK_AUTH_KEY	0x57	None
FUS_STORE_USR_KEY	0x58	Key Buffer = [KeyType (1byte), KeySize(1byte), KeyData (16/32bytes)]
FUS_LOAD_USR_KEY	0x59	Key Index (1byte)
FUS_START_WS	0x5A	None

7.2.2 USB-DFU upload FUS extension

Bootloader USB-DFU upload FUS extension is managed in same ways as regular upload command for reading physical address (wBlockNum > 1). But in this case, a virtual memory address mask is used: 0xFFFF0000. So the FUS read command is managed through a read to virtual address 0xFFFF00YY where YY is the FUS command opcode.

Upload command allows to perform the second step of FUS_STORE_USR_KEY which is getting the key index.

Table 20. USB-DFU upload extension

Command	Address	Returned data
FUS_GET_STATE	0xFFFF0054	State buffer = [FUS state (1byte), FUS error code (1byte)]
FUS_STORE_USR_KEY	0xFFFF0058	Key index (1byte)

8 FAQ and troubleshooting

Table 21. Frequently ask and answer

Question/troubleshooting	Answer
When I receive a virgin STM32WB device from ST, what does it contain exactly?	All STM32WB devices delivered by ST contain by default the FUS and the bootloader. They don't contain pre-installed wireless stack.
I cannot read the FUS version	<p>Accessing device information table is possible when following conditions are met :</p> <ol style="list-style-type: none"> 1. Device info table address is written in location pointed by IPCCDBA option byte. 2. CPU2 (CM0+) is enabled 3. FUS is running on CPU2 (and not wireless stack) (If wireless stack is running, it is possible to force FUS to run by sending 2 FUS_GET_STATE commands). So when accessing device via SWD, it is normal to not find device info table valid because it has not yet been written or CPU2 has not been enabled yet. That's why it is more convenient to read device info table when bootloader is running because it performs the actions (1) and (2) above. <p><i>Note: It is possible to connect through SWD and disable the hardware reset option (hot plug) and keep boot on bootloader which allows user to read the device info table.</i></p>
I want to upgrade FUS image and I already have a wireless stack installed. Do I need to delete the wireless stack prior to upgrading FUS?	<p>It is advised to delete the wireless stack before performing the FUS upgrade in general and especially when upgrading from FUS V0.5.3.</p> <p>If the existing FUS version is higher than V0.5.3, then, it is not mandatory to perform the wireless stack delete.</p>
How do I know quickly if my device is running FUS or wireless stack?	<p>There are multiple ways to check it:</p> <ul style="list-style-type: none"> • Read option bytes and check the value of SBRV. If FUS is running it shall be 0x3D000 (or 0x3D800 if FUS V0.5.3 is running) • Read the device information table @0x20030030, if it is different for the FUS version, then wireless stack is running or Cortex-CM0+ is not enabled. • Send FUS_GET_STATE command, if FUS_STATE_NOT_RUNNING is received, then the wireless stack is running or Cortex-CM0+ is not enabled.
What is IPCCDBA option byte used for?	IPCCDBA is used to change the offset where to read/write the device information table
After an upgrade operation, I cannot access flash memory anymore and can't communicate with FUS.	<p>First check if SFSA=0x00. If it is the case, then it means safeboot has been triggered. Safeboot is triggered when option bytes corruption occurs.</p> <p>This may occur during FUS upgrade operation or during any user application operation dealing with option bytes.</p> <p>When safeboot is triggered it locks the device by setting SFSA=0x00 (all FLASH memory secure) and so no user application/debugger can access the user FLASH memory anymore.</p> <p>This operation is not reversible.</p> <p>Starting from FUS V1.1.0 the safeboot is modified to perform a factory reset instead of locking the device.</p>
Is it possible to downgrade FUS version (ie. when current FUS running version is V1.0.2, is it possible to install FUS V1.0.1?)	<p>FUS downgrade is not possible in any combination. It can be installed only forward.</p> <p>In case of downgrade tentative, FUS will simply reject the upgrade and return error message.</p>

Question/troubleshooting	Answer
<p>What is a typical STM32CubeProgrammer command to perform an upgrade using FUS?</p>	<ol style="list-style-type: none"> 1. First check that FUS is running by sending FUS_GET_STATE commands till receiving FUS_STATE_IDLE state response: <ul style="list-style-type: none"> - STM32_Programmer_CLI.exe -c port=usb1 -fusgetstate - STM32_Programmer_CLI.exe -c port=usb1 -fusgetstate - STM32_Programmer_CLI.exe -c port=usb1 -fusgetstate <p>Sending 3 times FUS_GET_STATE command will ensure that FUS is running and idle in most cases.</p> 2. delete the existing wireless stack and install the new one (case of wireless stack upgrade): <ul style="list-style-type: none"> - STM32_Programmer_CLI.exe -c port=usb1 -fwupgrade stm32wb5x_BLE_Stack_fw.bin 0x080CB000 firstinstall=0 - STM32_Programmer_CLI.exe -c port=usb1 -fusgetstate - STM32_Programmer_CLI.exe -c port=usb1 -fusgetstate - ... (keep sending -fusgetstate till received state is FUS_STATE_NOT_RUNNING) <p>Setting "firstinstall=0" will ensure that previous existing stack will be deleted before new one is installed.</p> <p>Even if there is no previously installed stack, setting "firstinstall=0" would not cause any problem.</p> <p>Alternately proceed to FUS image installation (case of FUS upgrade):</p> <ul style="list-style-type: none"> • STM32_Programmer_CLI.exe -c port=usb1 -fwupgrade stm32wb5x_FUS_fw.bin 0x080EC000 firstinstall=0 • STM32_Programmer_CLI.exe -c port=usb1 -fusgetstate • STM32_Programmer_CLI.exe -c port=usb1 -fusgetstate • ... (keep sending -fusgetstate till received state is FUS_STATE_IDLE) <p>"firstinstall=0" means existing wireless stack will be deleted prior to upgrading FUS.</p> <p>It is possible to use "firstinstall=1" if upgrading from FUS version different from FUS V0.5.3.</p>
<p>What is safeboot and how can it be used?</p>	<p>Safeboot is an independent part of the FUS that manages specifically one case: option bytes corruption.</p> <p>When option bytes are corrupted, the STM32WB hardware forces the boot to safeboot whatever the running firmware.</p> <p>The safeboot then either:</p> <ul style="list-style-type: none"> • Locks the device in full secure mode (on FUS versions lower than V1.1.0) which means all the device FLASH memory can't be accessed and this operation is not reversible (there is no mean to cancel it and the device cannot be used anymore). • or performs a factory reset (on FUS versions V1.1.0 and higher) which means the wireless stack is removed if any and the Cortex-M4 code is erased and boot is reset to FUS (virgin part state). This operation is also not reversible.
<p>Does FUS erase the shadow of encrypted firmware after installation?</p>	<p>Yes, FUS does erase the shadow remaining sectors of the encrypted firmware after it has been installed and moved to upper address.</p>

Revision history

Table 22. Document revision history

Date	Version	Changes
21-Mar-2019	1	Initial release.
17-Jun-2019	2	Added Section 1.2 FUS versioning and identification Updated: <ul style="list-style-type: none"> Section 2 Wireless stack image operations, Section 5.1 Key types and structure Table 18. USART FUS command mapping on write command, Table 20. USB-DFU upload extension
10-Jul-2019	3	Updated Table 5. Device information table
31-Mar-2020	4	Updated: <ul style="list-style-type: none"> Section 1.1 Firmware upgrade services definition, Section 2 Wireless stack image operations, Section 2.1 Wireless stack install and upgrade, Section 2.2 Wireless stack delete, Section 5.1 Key types and structure, Section 7.1 USART extension Table 1. FUS versions, Table 9. FUS commands (vendor specific HCI command packet), Table 10. FUS responses (HCI command complete packet), Table 12. FUS state error values Added: Section 2.4 Anti-rollback activation and Section 8 FAQ and troubleshooting

Contents

1	General information	2
1.1	Firmware upgrade services definition	2
1.2	FUS versioning and identification	3
1.3	How to activate FUS	4
1.4	Memory mapping	5
1.5	FUS resources usage	7
1.6	Shared tables memory usage	8
2	Wireless stack image operations	9
2.1	Wireless stack install and upgrade	9
2.2	Wireless stack delete	10
2.3	Wireless stack start	10
2.4	Anti-rollback activation	11
3	FUS upgrade	12
3.1	Operation instructions	12
3.2	Memory considerations	12
4	User authentication	13
4.1	Install user authentication key	13
4.2	Lock user authentication key	13
5	Customer key storage	14
5.1	Key types and structure	14
6	Communication with FUS	16
6.1	Shared tables usage	16
6.1.1	Device information table	16
6.1.2	System table	18
6.2	IPCC usage	18
6.3	FUS commands	19
6.3.1	Packet indicators	20
6.3.2	Event packet	20
6.3.3	Command packet	20

6.3.4	Response packet	21
6.4	Image footers	23
7	STM32 system bootloader extension for FUS	26
7.1	USART extension	26
7.1.1	USART special read	26
7.1.2	USART special write	28
7.1.3	USART FUS commands mapping	29
7.2	USB-DFU extension	30
7.2.1	USB-DFU download FUS extension	30
7.2.2	USB-DFU upload FUS extension	31
8	FAQ and troubleshooting	32
	Revision history	34

List of tables

Table 1.	FUS versions	3
Table 2.	FUS activation cases	4
Table 3.	FUS resources usage	7
Table 4.	FUS upgrade returned errors	9
Table 5.	Device information table	17
Table 6.	System table content	18
Table 7.	Packet indicator values	20
Table 8.	FUS asynch event (vendor specific HCI event)	20
Table 9.	FUS commands (vendor specific HCI command packet)	20
Table 10.	FUS responses (HCI command complete packet)	21
Table 11.	FUS state values	22
Table 12.	FUS state error values	22
Table 13.	Parsing of image footer structure	24
Table 14.	Parsing of signature footer	25
Table 15.	Magic number values	25
Table 16.	Bootloader USART commands extension	26
Table 17.	USART FUS command mapping on read command	29
Table 18.	USART FUS command mapping on write command	30
Table 19.	USB-DFU download extension	30
Table 20.	USB-DFU upload extension	31
Table 21.	Frequently ask and answer	32
Table 22.	Document revision history	34

List of figures

Figure 1.	Flash memory mapping	5
Figure 2.	SRAM memory mapping	6
Figure 3.	Shared table architecture	8
Figure 4.	Shared table usage process	16
Figure 5.	IPCC channels used by FUS	19
Figure 6.	FUS HCI subset	19
Figure 7.	Image footers placement	23
Figure 8.	FW/FUS upgrade image footer structure	24
Figure 9.	Signature (tag) footer structure	25
Figure 10.	USART special read command	27
Figure 11.	USART special write command	29

IMPORTANT NOTICE – PLEASE READ CAREFULLY

STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST’s terms and conditions of sale in place at the time of order acknowledgement.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of Purchasers’ products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. For additional information about ST trademarks, please 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.

© 2020 STMicroelectronics – All rights reserved