life.augmented

AN5275

Application note

USB DFU/USART protocols used in STM32MP1 Series bootloaders

Introduction

This application note describes the protocols used by the bootloader programming tools for the STM32MP1 Series of microprocessors. It details each USB DFU or USART command supported by the embedded software and the sequences expected by the STM32CubeProgrammer tool.

1 Embedded programming service

1.1 Introduction

This document applies to the STM32MP1 Series Arm®-based microprocessors.

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

It defines the protocols used by the STM32MP1 Series bootloaders, including ROM code, to provide the programming service that is, the embedded part needed by the STM32CubeProgrammer.

This service provides a way to program the non-volatile memories (NVM): external Flash memory device or on chip non-volatile memory (OTP).

This document describes the USB or USART protocols used by the STM32MP1 embedded software and by the STM32CubeProgrammer as well as the expected sequences.

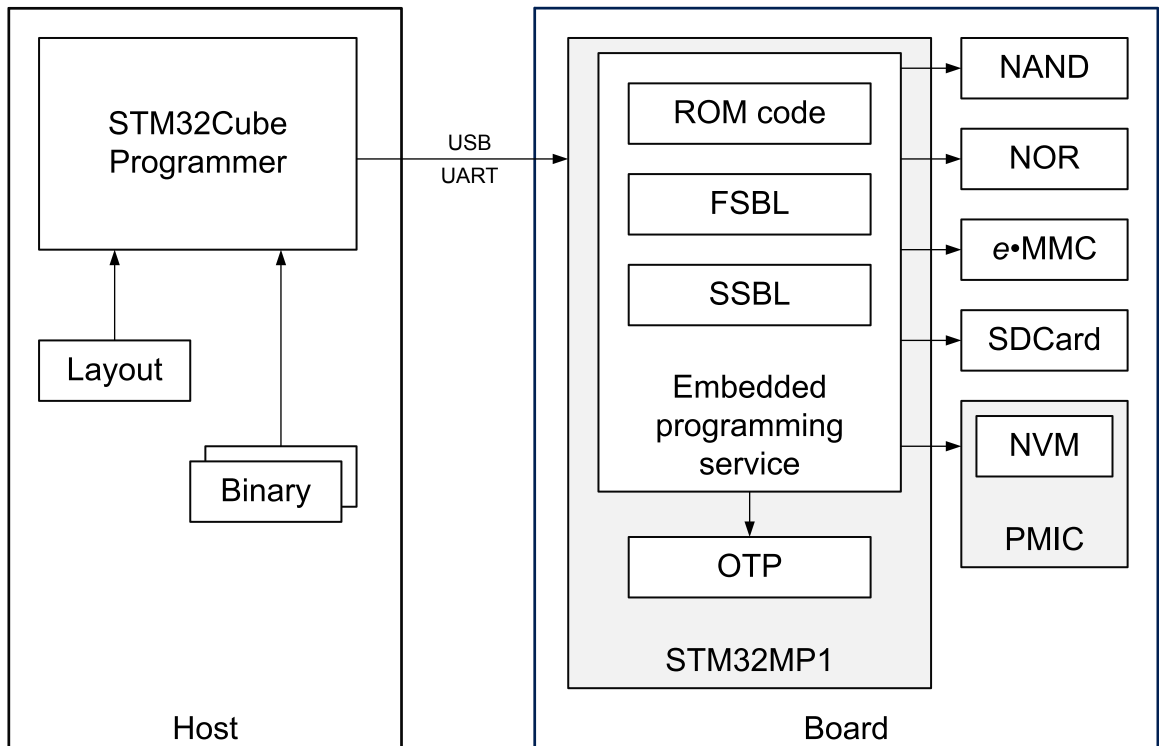
1.2 Reference

[1]	AN3155, Application note, USART protocol used in the STM32 bootloader
[2]	AN3156, Application note, USB DFU protocol used in the STM32 bootloader
[3]	UM0412, User manual, Getting started with DfuSe USB device firmware upgrade STMicroelectronics extension
[4]	UM0424, User manual, STM32 USB-FS-Device development kit
[5]	Universal Serial Bus, Device Class Specification for Device Firmware Upgrade, version 1.1 from https://usb.org/
[6]	The GUID Partition Table (GPT) is a standard for the layout of partition tables, part of the Unified Extensible Firmware Interface (UEFI) standard (https://uefi.org/)
[7]	Signing tool in wiki https://wiki.st.com/stm32mpu/wiki/Signing_tool
[8]	ROM code overview in wiki https://wiki.st.com/stm32mpu/wiki/Category:ROM_code
[9]	FlashLayout in wiki https://wiki.st.com/stm32mpu/wiki/STM32CubeProgrammer_flashlayout
[10]	OTP in wiki https://wiki.st.com/stm32mpu/wiki/STM32CubeProgrammer_OTP_management
[11]	STPMIC1 NVM in wiki https://wiki.st.com/stm32mpu/wiki/STM32CubeProgrammer_STPMIC1_NVM_management
[12]	STM32 header for binary files https://wiki.st.com/stm32mpu/wiki/STM32_header_for_binary_files
[13]	Chapter “Serial boot” in “Rom code overview” found in https://wiki.st.com/stm32mpu/wiki/Category:ROM_code

1.3 Overview

The programming operation consists in writing binaries initially stored on a host computer, via an interface, to any non-volatile memory (NVM) on the platform. This process involves the STM32CubeProgrammer tool that communicates with an embedded programming service consisting of the ROM code, the first stage bootloader (FSBL) and the second stage bootloader (SSBL).

Figure 1. **Programming operation**



A communication protocol is defined for each serial interface (USART, USB) involving a set of commands and some sequences that are as much compatible as possible with existing STM32 MCU devices (see [1] and [2]).

The possible NVMs are:

- An external Flash memory device:
  - NAND Flash
  - *e*•MMC
  - SD card
  - NOR Flash
- An on-chip non-volatile memory:
  - STM32MP1 OTP
  - NVM of a PMIC (STPMIC1 for example)

A Layout file gives the list of binaries to program, their type (binary, file system...), the targeted NVM and the position in the NVM.

The eventual signing steps of the binary files with STM32 header are done previously with the SigningTool (see [7]).

The embedded programming service is based on the ROM code, the FSBL = Arm Trusted Firmware (TF-A) and SSBL = U-Boot. The same protocol is used for both downloading the FSBL and SSBL in RAM and for loading the partition to be programmed in the device NVM.

The ROM code is embedded in the STM32MP device. Its main task is to load, verify and execute the first stage bootloader (FSBL) in internal RAM through one of the available serial peripherals. In turn, after some initializations (clock and DDR), the FSBL loads the second loader (SSBL) in DDR, verifies the signature and executes it.

1.4 Layout file format

The Flash memory Layout file is a tab-separated-value (tsv) text file with one line per partition or binary to be sent to the device. The complete description can be found in ST wiki [9].

1.5 Phase ID

It is a unique identifier present in the Layout file for each download phase request made to the STM32CubeProgrammer either by the ROM code/FSBL (load in RAM) or by the SSBL = U-Boot .

It is used by the embedded programming service to identify the next partition (Layout, TF-A, U-Boot, etc.) to be downloaded in the Get phase command answer.

Table 1. Phase ID

Code	Partition
0x00	Layout file
0x01 to 0x0F	Boot partitions with STM32 header [12] SSBL, FSBL, other (TEE, M4 firmware)
0x01 (reserved)	FSBL (first copy) : used by ROM code
0x03(reserved)	SSBL : used by FSBL=TF-A
0x10 to 0xF0	User partitions programmed without STM32 header (uimage, dtb, rootfs, user…)
0xF1 to 0xFD	Reserved for internal purpose
0xF1	Command GetPhase
0xF2	OTP
0xF3	Reserved
0xF4	PMIC NVM (optional)
0xFE	End of operation
0xFF	Reset

The IDs 0x01 and 0x03 are reserved for FSBL and SSBL. The FSBL is loaded in RAM by the ROM code, the SSBL is loaded in RAM by the FSBL.

The IDs 0xF1 to 0xFD are reserved for internal purpose.

A complete description for the content of the NVM partitions can be found in ST wikis [10] and [11].

1.6 STM32 image header

The details of the STM32 image header can be found in STM32 MPU wiki [12].

This header (size 0x100 bytes, signed or not) is used for the boot partitions loaded by ROM code and by the FSBL TF-A. The header is allowed and used only for the boot partitions (ID < 0x10).

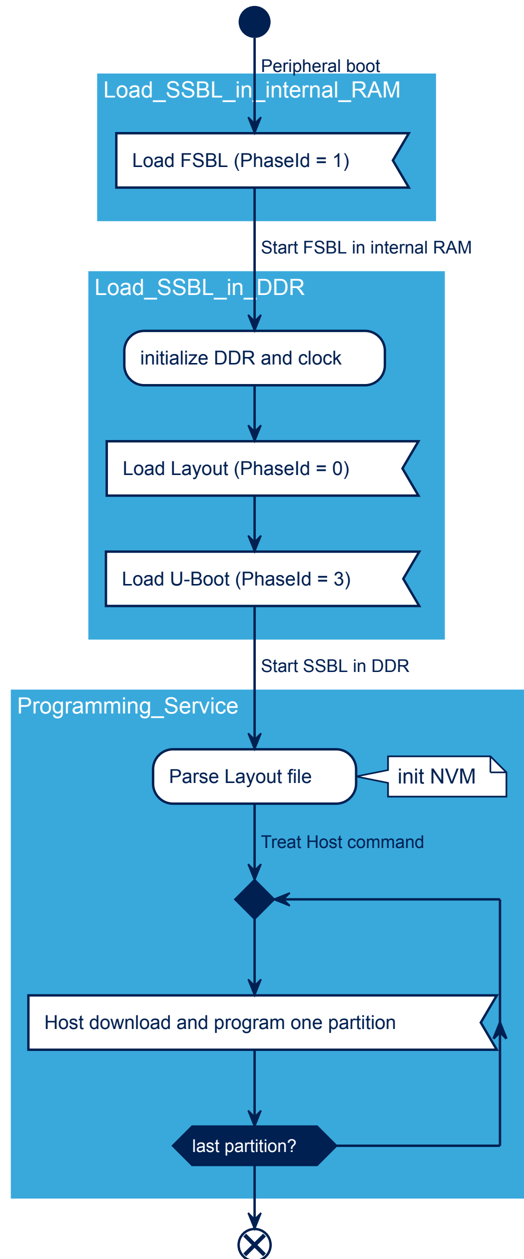
The signature is mandatory only on "closed" devices (see signing tools [7] for details on the signature process, see [8] for definition of "closed" devices.).

1.7 Programming sequence

1.7.1 Case 1 – programming from reset

In this normal use case of the STM32CubeProgrammer, the voltage level on the boot pins is used to determine the peripheral used for boot (USB or UART see [13]). The bootloaders are loaded by the STM32CubeProgrammer in RAM and the SSBL U-Boot provides the programming service running in DDR.

Figure 2. **Programing chart**



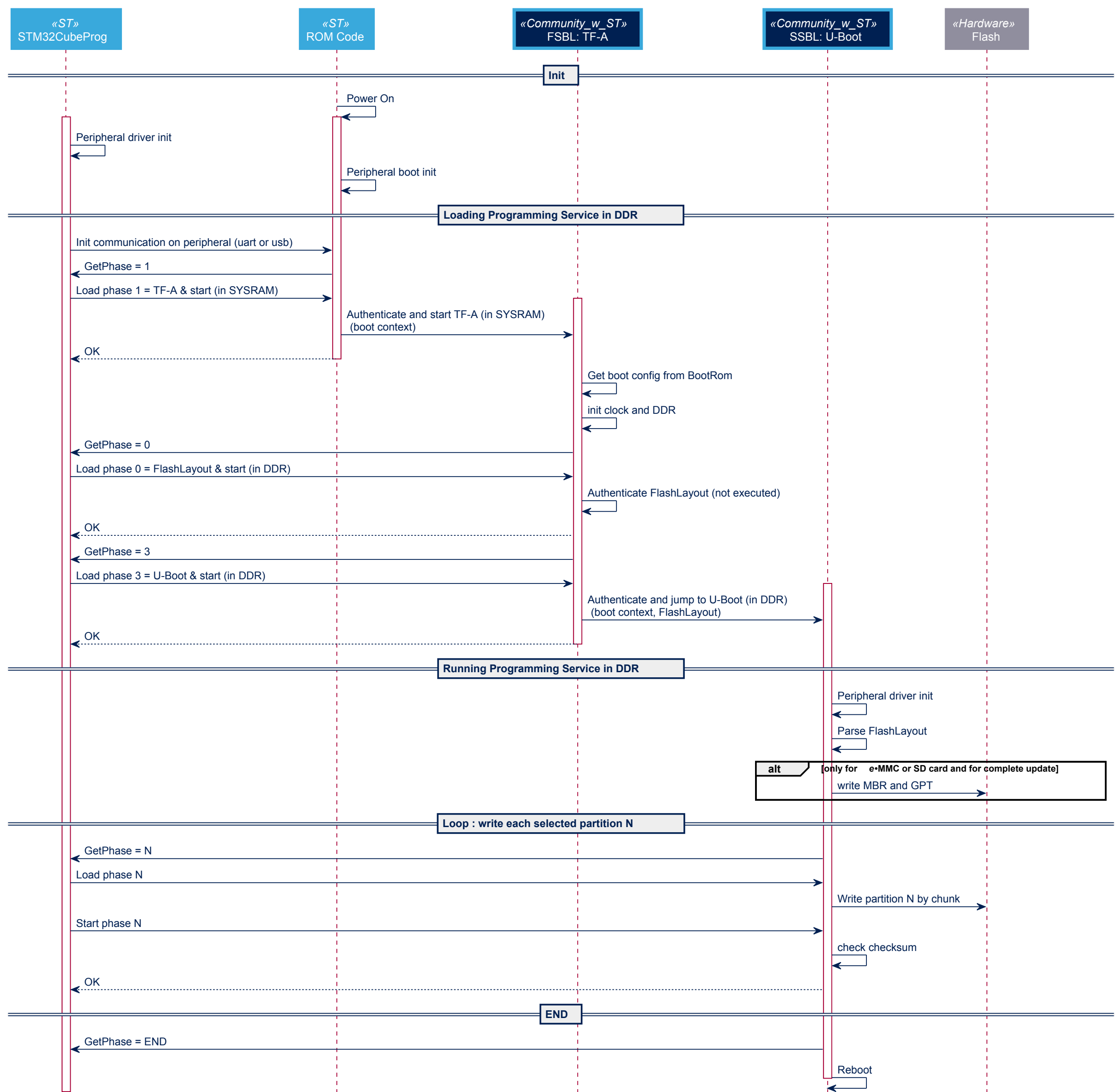
For a full update, the same boot stage partitions (TF-A and U-Boot) is requested twice, once for loading and execution in RAM and once for the NVM update, however the provided binary is not necessary the same.

In U-Boot, the data chunks received on UART and on USB are buffered in DDR, the size of this buffer is device dependent.

The buffer is flushed to the NVM when it is full, thus the ACK for each transferred chunk is delayed to when this NVM update is performed.

The diagram below gives an overview of a typical programming sequence.

Figure 3. Programming Sequence



1.7.2 Case 2 – programming from U-Boot for a device already programmed

In this use case (code update), the NVM is already programmed with a valid boot chain and selected by the boot pins (see [13]).

The expected sequence is:

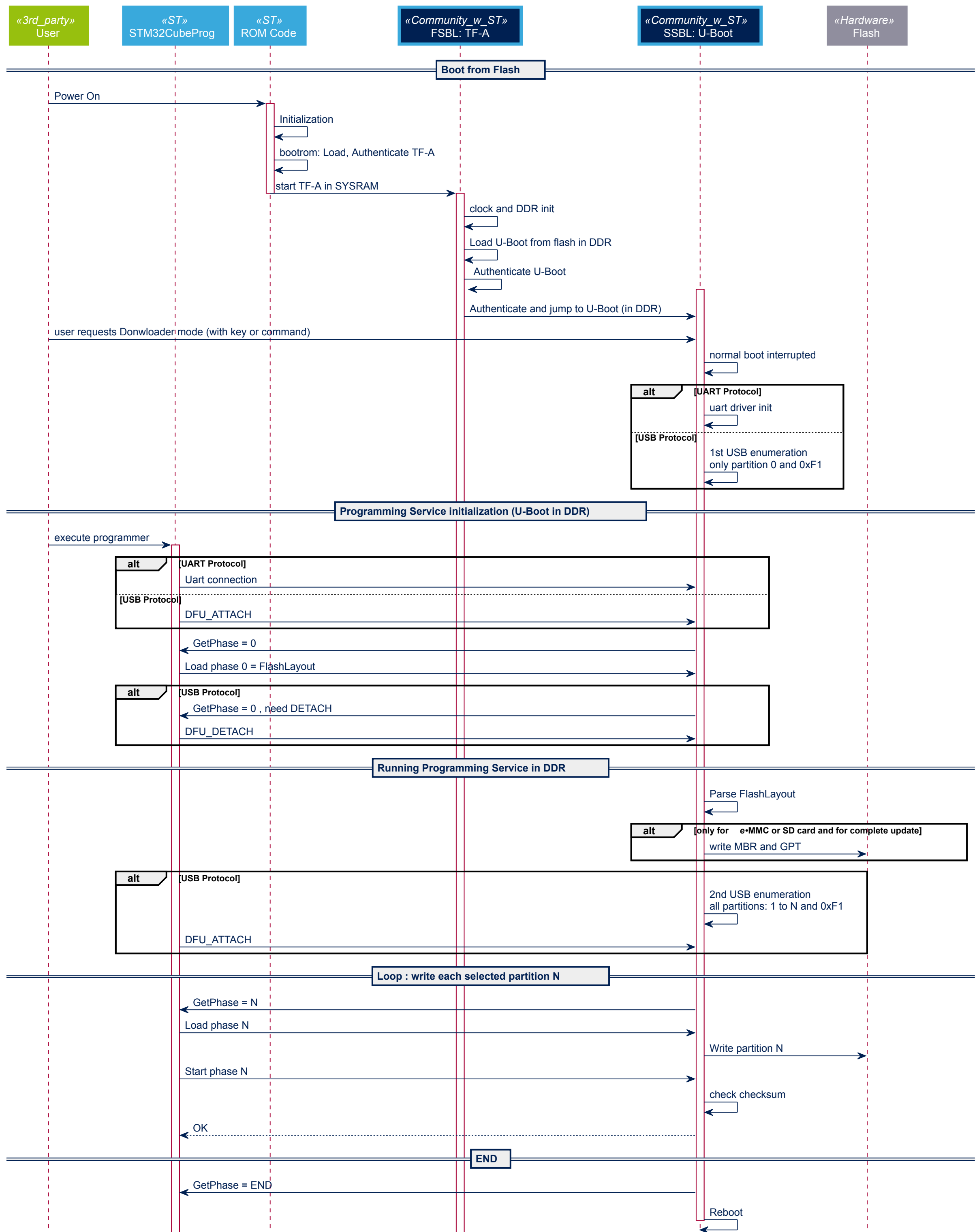
1. **User** switches on the board
2. The bootloaders are loaded from NVM and executed in RAM
3. **User** interrupts the platform boot and starts **U-Boot** in programmer mode (UART or USB). Several solution are possible:
   – Key press detection to enter in programming mode (PA14)
   – **User** launches the download command in U-Boot console (command stm32prog)
   – Programmer mode requested by Linux via reboot mode
4. **User** starts STM32CubeProgrammer on PC
   a. Connection to U-Boot
   b. Requested Layout file for phase 0x0

5. **U-Boot:** loads and treats the Layout file
   - **USB => 2nd enumeration with all partitions**
   - **UART => continues with the first requested phase**

Then the **programming service** is running in the DDR exactly as in the previous case.

The diagram below gives an overview of this typical programming sequence.

Figure 4. Programming sequence for boot from NVM



2 UART/USART

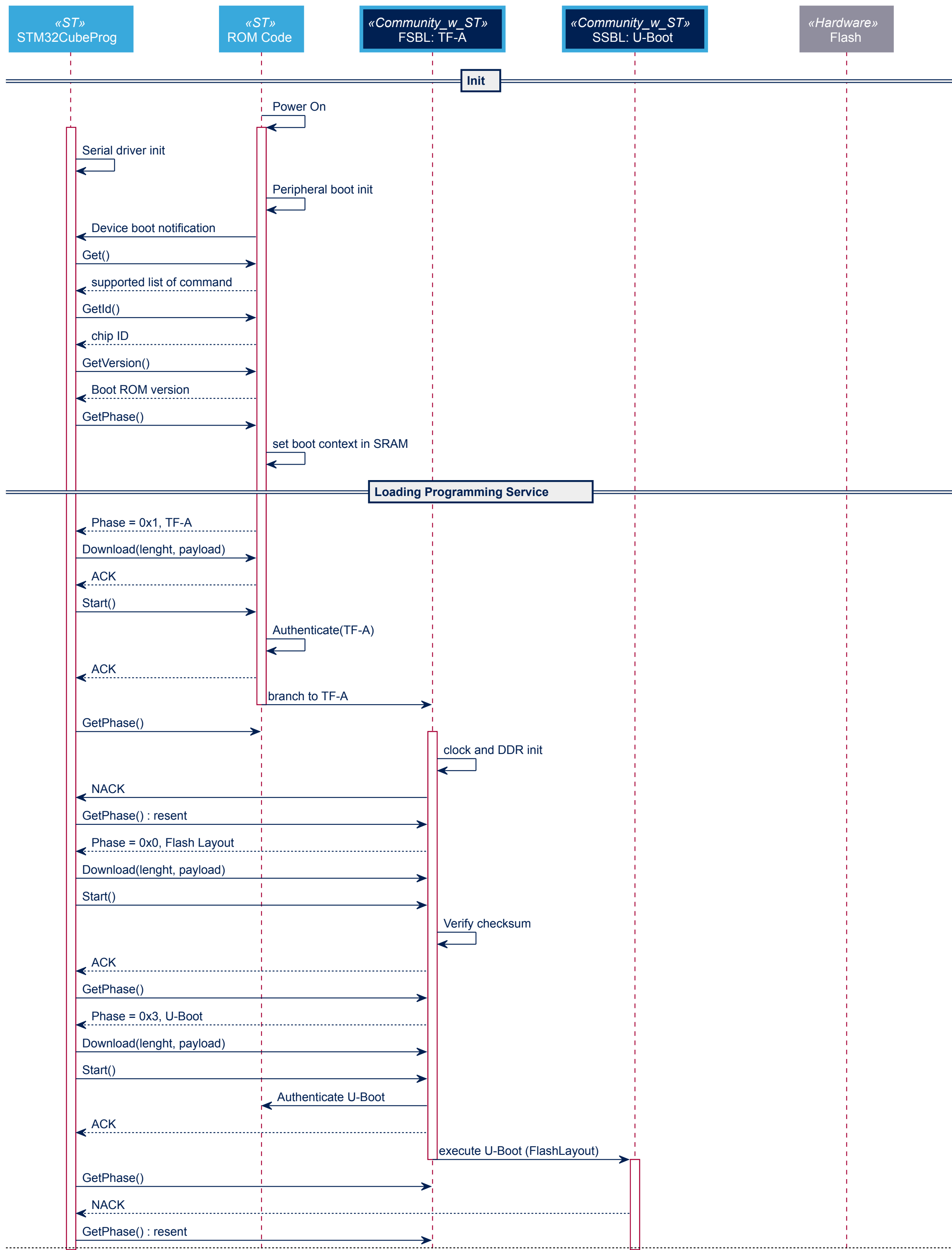
2.1 UART/USART protocol

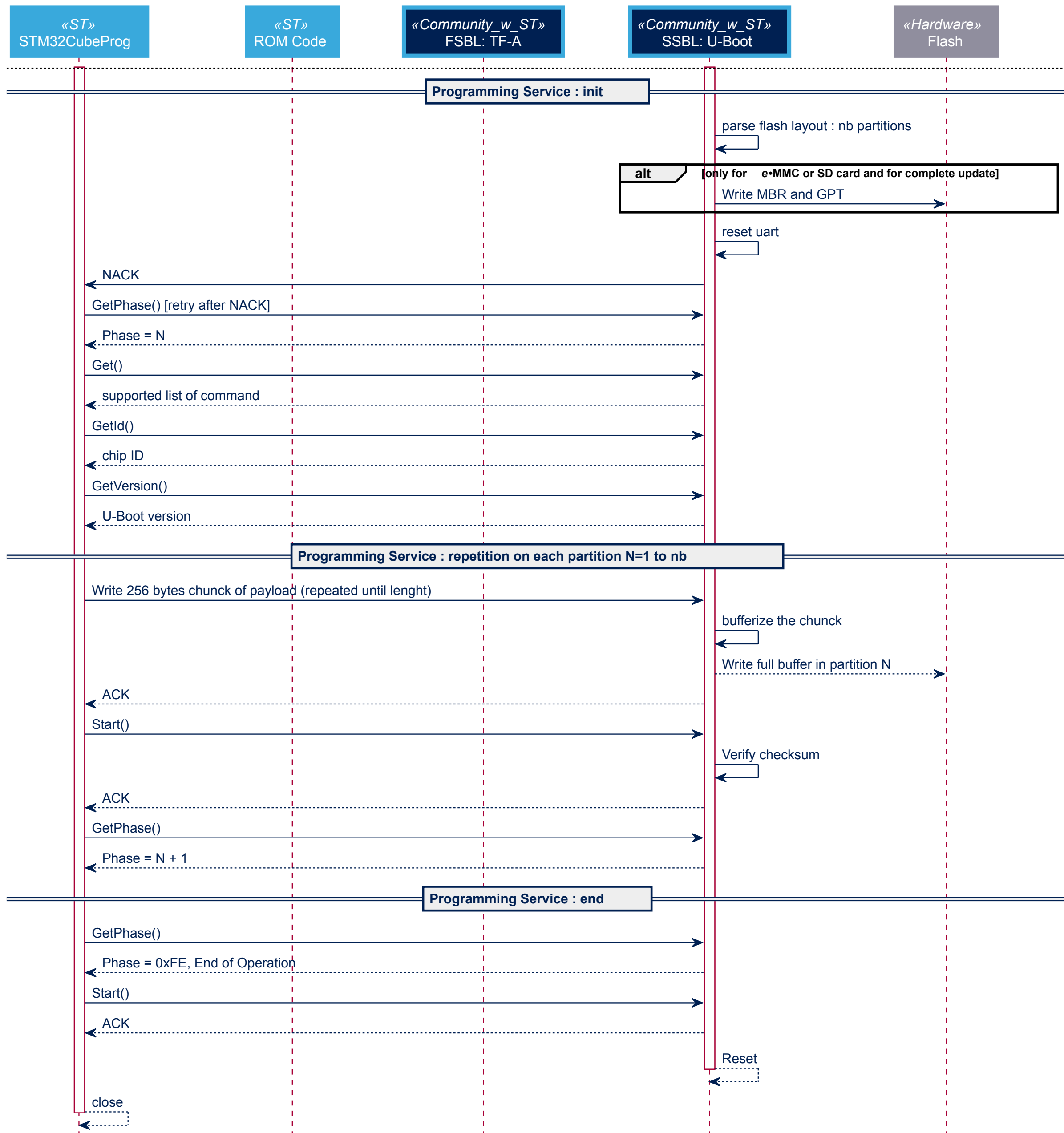
The UART/USART protocol follows as much as possible the STM32 MCU behavior described in [1].

The differences are highlighted here below.

The expected sequence is detailed in the figure below.

Figure 5. Usart programming sequence





2.2 UART/USART configuration

The configuration is set by the ROM code ([13]) with the following settings:

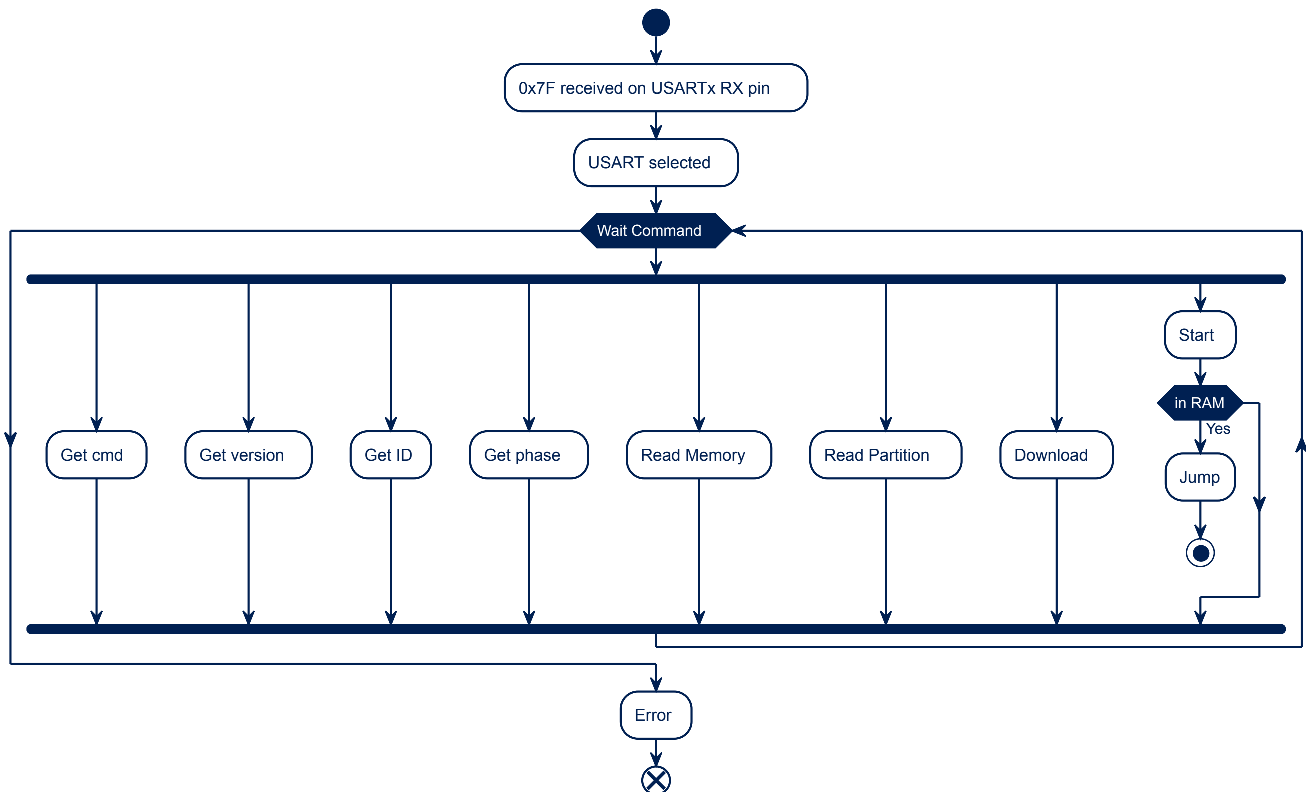
- baudrate = 115200 baud
- 8-bit data
- EVEN parity
- 1 start bit
- 1 stop bit

2.3 UART/USART connection

Once the serial boot mode is entered (see boot pin in [13]), all the UART/USART instances are scanned by the ROM code, monitoring for each instance the USARTx_RX line pin, waiting to receive the 0x7F data frame (one start bit, 0x7F data bits, none parity bit and one stop bit).

2.4 UART/USART main loop

Figure 6. **Main UART/USART sequence**



The device supports the following commands set:

- **Get cmd** (to get the bootloader version and the allowed commands supported by the current version of the bootloader)
- **Get version**(to get the software version)
- **Get ID**(to get the chip ID)
- **Get phase**(*) (to get the phase ID: the partition that is going to be downloaded)
- **Download** (to download an image into the device)
- **Read Memory**(to read SRAM memory)
- **Read Partition**(*)
- **Start** (to go to user code)

(*) STM32MP1 specific commands.

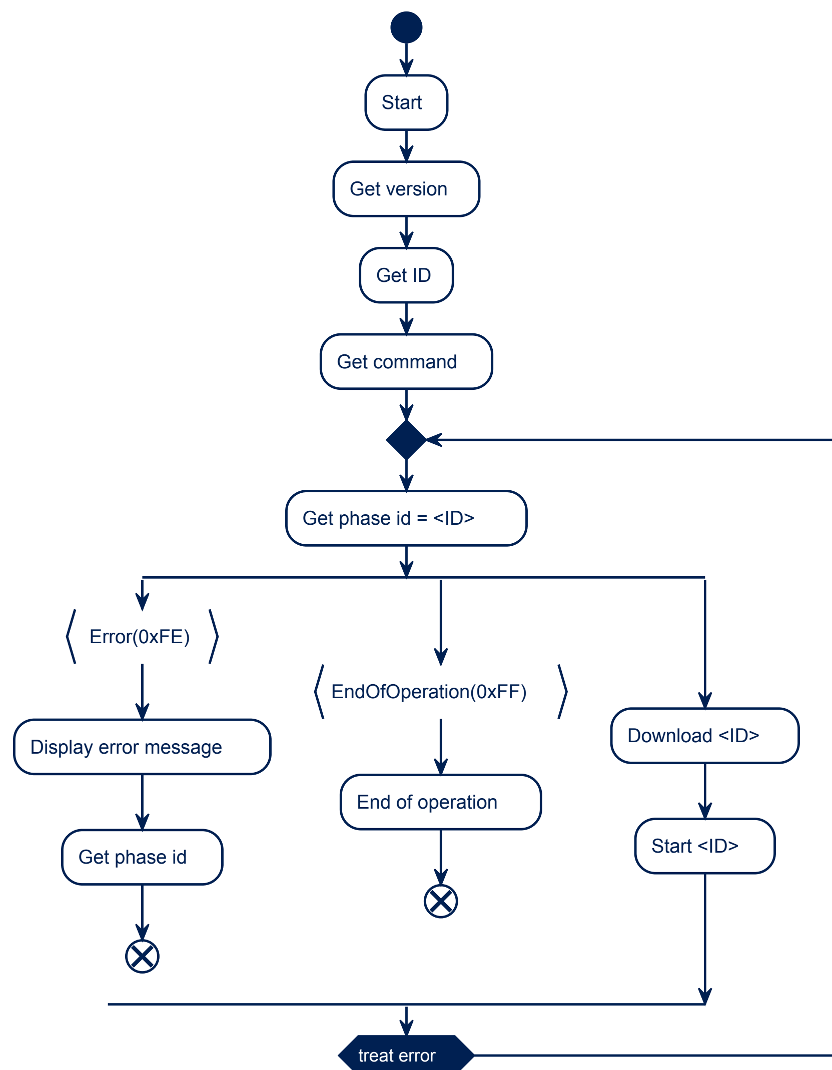
Figure 8 illustrates the device code execution during the different download phases. The host starts by the get cmd command then the get phase command and depend the Phase ID the host sends the adequate data.

- *0 : The host sends the data to download the Layout file.*
- *1 : The host sends the data to download the FSBL1 (=TF-A 1).*
- *2 : The host sends the data to download the FSBL2 (=TF-A 2).*
- *3 : The host sends the data to download the SSBL (=U-Boot).*

- *[4, F] : Other boot partition.*
- *[10, F0] : User partition .*
- *F1 : Reserved for UART/USART.*
- *[F2, FD] : Virtual Partition.*
- *FE : End of operation.*
- *FF : Reset .*

The device returns an error when there is a corruption of data.
In case of NACK error the system re-starts the operation.
In case of ABORT error the system reboots after the next GetPhaseId with answered value RESET (0xFF).

Figure 7. STM32CubeProgrammer download sequence



Abort response: The abort response is used to inform the host in case of a major issue (error during the signature check or error during writing). The only way to recover is to perform a system reset and to restart communication. The downloaded image is discarded immediately after Abort response and the device forces a system reset after the next GetPhaseId with answered value RESET (0xFF).

2.5 UART/USART command set

The supported commands are listed in the table below. Each command is further described in the following sections.

Table 2. **USART commands**

Command	Command code	Command description
Get	0x00	Gets the version of the running element and the allowed commands supported.
Get Version	0x01	Gets the version
Get ID	0x02	Gets the device ID
Get phase	0x03	Gets the phase ID: the identifier of the partition in the Layout file that is going to be downloaded
Read Memory	0x11	Reads up to 256 bytes of memory starting from an address specified by the application
Read Partition	0x12	Reads up to 256 bytes of partition at offset specified by the application (new in UART/USART bootloader protocol v4.0)
Start (Go)	0x21	Jumps to the user application located in the internal SRAM
Download (Write Memory)	0x31	Download the image
Erase	0x43	Existing in USART protocol v3, not used in STM32MP1
Extended Erase	0x44	Existing in USART protocol v3, not used in STM32MP1
Write Unprotect	0x73	Existing in USART protocol v3, not used in STM32MP1
Readout Protect	0x82	Existing in USART protocol v3, not used in STM32MP1
Readout Unprotect	0x92	Existing in USART protocol v3, not used in STM32MP1

Communication safety

All communications from STM32CubeProgrammer (PC) to the device are verified as follows:

- The UART/USART even parity is checked
- For each command the host sends a byte and its complement (XOR = 0x00)
- The device performs a checksum on the sent/received datablocks. A byte containing the computed XOR of all previous bytes is appended at the end of each communication (checksum byte). By XORing all received bytes, data + checksum, the result at the end of the packet must be 0x00.

Each command packet is either accepted (ACK answer), discarded (NACK answer) or aborted (unrecoverable error):

- ACK = 0x79
- NACK = 0x1F
- ABORT = 0x5F

2.5.1 Get command (0x00)

The Get command returns the bootloader version and the supported commands. When the device receives the Get command, it transmits the version and the supported command codes to the host, as described in Figure 7.

The device sends the bytes as described in the table below.

Table 3. **Get command response**

Byte	Description
1	ACK
2	N = 8 = the number of following bytes – 1 (except current and ACKs)
3	UART/USART bootloader version (0 < version < 255)

Byte	Description
	example: 0x10 = version 1.0 On STM32MP1 the USART protocol version is V4.0, so the value is 0x40
4	0x00: Get command
5	0x01: Get version
6	0x02: Get ID
7	0x03: Get phase ID
8	0x31: Download command
9	0x11: Read command
10	0x12: Read partition command
11	0x21: Start command
Last	ACK

2.5.2 Get ID command (0x02)

The Get ID command is used to get the version of the device ID (identification). When the device receives the command, it transmits the device ID to the host.

The device sends the bytes as follows:

Table 4. Get ID command response

Byte	Description
1	ACK
2	N = 1 = the number of following bytes – 1 (except current and ACKs)
3-4	Device ID = 0x0500 for STM32MP15x
Last	ACK

2.5.3 Get version command (0x01)

The Get version command is used to get the version of the running component. When the device receives the command, it transmits the version to the host.

The device sends the bytes as follows:

Table 5. Get version command response

Byte	Description
1	ACK
2	Bootloader version => software version (ROM code/TF-A/U-Boot) (0 < version ≤ 255), example: 0x10 = version 1.0
3	Option byte 1: 0x00[1]
4	Option byte 2: 0x00[1]
Last	ACK

1. *Option byte keeps the compatibility with generic bootloader protocol see [Ref1].*

2.5.4 Get phase command (0x03)

Note: This command is STM32MP1 specific.

The Get phase command enables the host to get the phase ID, in order to identify the next partition that is going to be downloaded.

When the device receives the Get phase command, it transmits the partition ID to the host as follows:

Table 6. **Get phase command response**

Byte	Description
1	ACK
2	N = the number of following bytes -1 (except current and ACKs, 0 ≤ N ≤..255)
3	Phase ID
4-7	Download address
8	X = the number of bytes in additional information (X = N-5)
X	X bytes of additional information
Last	ACK

The download address, when present, provides the destination address in memory. A value of 0xFFFFFFFF means than the partition is going to be written in NVM.

Phase ID = 0xFF corresponds to an answered value Reset, in this case the information bytes provide the cause of the error in a string just before executing the reset.

- **ROM code**

  The ROM code sends Phase = TF-A

  Byte 1: ACK
  Byte 2 N = 6
  Byte 3: Phase ID (TF-A =1)
  Byte 4-7: 0x2FFF0000
  Byte 8: X = 1
  Byte 9: 0: Reserved
  Byte 10: ACK

- **TF-A**

  The TF-A sends

  Byte 1: ACK
  Byte 2: N = 5
  Byte 3: Phase ID (SSBL: U-Boot = 3, Layout = 0)
  Byte 3-7: 0x2FFF0000
  Byte 8: X = 0
  Byte 9: ACK

- **U-Boot**

  The U-Boot sends the bytes as follows:

  Byte 1: ACK
  Byte 2: N = 5
  Byte 3: Phase ID (next partition to program)
  Byte 4-7: 0xFFFFFFFF
  Byte 8: X = 0
  Byte 9: ACK

  For TF-A case (phase = 1), no additional information is provided by U-Boot (N=5, X=0).

  Byte 1: ACK
  Byte 2: N = 5
  Byte 2: Phase ID (FSBL1: TF-A =1)
  Byte 3-7: 0xFFFFFFFF
  Byte 7: X =0
  Byte 9: ACK

  For the error case, U-Boot sends the bytes as follows:

  Byte 1: ACK
  Byte 2: N = X-5
  Byte 3: Phase ID = Reset (0xFF)
  Byte 3- 7: 0xFFFFFFFF
  Byte 8 X= String size
  X bytes: string in ascii (Max 250 bytes): the cause of error
  Last Byte: ACK

2.5.5 Download command (0x31)

The download command is used to download a binary code (image) into the SRAM memory or to write a partition in NVM.

Two types of operations are available:

- Normal operation: download current partition binary to device
  - For a signed binary, the signature verification is be done before execution else only the checksum is verified.
  - For initialization phase the partitions are loaded in SRAM and executed, otherwise for writing phase the partition are written in NVM
  - A Start command is necessary to finalize the operation.
- Special operation: download non-signed data to non-executable memory space.

Packet number: the packet number is used to specify the type of operation and the number of the current packet. The description of the packet number is provided below:

Table 7. **Packet number**

Byte	Value	Description
3	0x00	Normal operation
	0xF2	Special operation : OTP write
	0xF3	Special operation : Reserved
	0xF4	Special operation : PMIC NVM write
	Others	Reserved
0-2		Packet relative position or physical address in case of special operation

Examples:

1. Packet number = 0x00603102
   Operation = normal operation, Packet number = 0x603102
2. Packet number = 0xF200000N -> send Nth OTP part
   Operation = OTP write, OTP part number

The host sends the bytes to the device as follows:

Table 8. **Download command**

Byte	Description
1	0x31 = download (write memory)
2	0xCE = XOR of byte 1
-	Wait for ACK or NACK
3-6	Packet number
7	Checksum byte: XOR (byte 3 to byte 6)
-	Wait for ACK or NACK
8	Packet size (0 < N < 255)
9-(Last-1)	N+1 data bytes (max 256 bytes)
Last	Checksum byte: XOR (byte 8 to Last-1)
-	Wait for ACK or NACK

2.5.6 Read memory command (0x11)

The Read memory command is used to read data from any valid memory address. When the device receives the Read memory command, it transmits the ACK byte to the application. After the transmission of the ACK byte, the device waits for an address (4 bytes) and a checksum byte, then it checks the received address. If the address is valid and the checksum is correct, the ROM code transmits an ACK byte, otherwise it transmits a NACK byte and aborts the command.

When the address is valid and the checksum is correct, the device waits for N (N = number of bytes to be received -1) and for its complemented byte (checksum). If the checksum is correct the device transmits the needed data (N+1 bytes) to the application, starting from the received address. If the checksum is not correct, it sends a NACK before aborting the command.

To read memory mapped content, the Host sends bytes to the device as follows:

Table 9. **Read memory command**

Byte	Description
1	0x11 = read memory
2	0xEE = XOR of byte 1
-	Wait for ACK or NACK
3-6	Start address
7	Checksum byte: XOR (byte 3 to byte 6)
-	Wait for ACK or NACK
8	Number of bytes to be received – 1 (N = [0, 255])
9	Checksum byte: XOR (byte 8)
-	Wait for ACK or NACK

2.5.7 Read partition command (0x12)

Note: This command is STM32MP1 specific.

The Read command is used to read data from any valid memory-mapped address. When the device receives the Read memory command, it transmits the ACK byte to the application. After the transmission of the ACK byte, the device waits for a partition ID, an offset (4 bytes) and a checksum byte, then it checks the received address. If the address is valid and the checksum is correct, the ROM code transmits an ACK byte, otherwise it transmits a NACK byte and aborts the command.

When the address is valid and the checksum is correct, the device waits for the number of bytes to be transmitted – 1 (N bytes) and for its complemented byte (checksum). If the checksum is correct it then transmits the needed data (N bytes) to the application, starting from the received address. If the checksum is not correct, it sends a NACK before aborting the command.

To read partition in NVM, the Host sends bytes to the device as follows:

Table 10. Read Partition command

Byte	Description
1	0x12 = Read Partition
2	0xED = XOR of byte 1
-	Wait for ACK or NACK
3	partition Id = value
4-7	offset address
8	Checksum byte: XOR (byte 3 to byte 7)
-	Wait for ACK or NACK
9	Number of bytes to be received – 1 (N = [0,255])
10	Checksum byte: XOR (byte 9)
-	Wait for ACK or NACK

2.5.8 Start command (0x21)

The Start command is used:

- To execute the code just downloaded in memory or any other code by branching to an address specified by the application. When the device receives the Start command, it transmits the ACK byte to the application. If the address is valid the device transmits an ACK byte and jumps to this address, otherwise it transmits a NACK byte and aborts the command.
- The finalize the last Download command, when the host indicate the address = 0xFFFFFFFF, then the device first check the integrity of the downloaded image (with signature or checksum), and use the “Starting Address” provided in the partition header.

  If address is invalid or if checksum check failed, NACK byte is transmited.

  If authentication failed, no byte is transmitted (to avoid security attack).

The Host sends bytes to the device as follows:

Table 11. Start command

Byte	Description
1	0x21 = start
2	0xDE = XOR of byte 1
-	Wait for ACK or NACK
3-6	Start address or 0xFFFFFFFF
7	Checksum byte: XOR (byte 3 to byte 6)
-	Wait for ACK or NACK

3 USB

3.1 DFU protocol

The embedded programming service uses the DFU protocols v1.1 (see [5] for details).

The only difference with DFU V1.1 protocol is:

DFU_DETACH is acceptable in dfuIDLE state and get back to runtime mode appIDLE.

Note: This behavior is already supported in U-Boot DFU stack and in dfu-utils (with -e Option).

In dfuIDLE state, the device expects a USB reset to continue the execution.

In the following sections only the STM32MP1 specificities are presented.

Caution: the DFUSE used in STM32 MCU is not supported (the non supported case is DFUSE = DFU with ST extension and DFU version equal to v1.1a, see [2] for details).

The DFU states are described in the interface state transition diagram (see figure below, source: chapter A.1 of [5]):

Figure 8. Interface state transition diagram

3.2 USB sequence

Since the USB context is provided to TF-A by the ROM code, a new enumeration is NOT needed after TF-A manifestation.

As a consequence, the ROM code needs to present the alternate settings used by TF-A, even if they are not supported in the ROM code and the device needs to indicate bitManifestationTolerant=1.

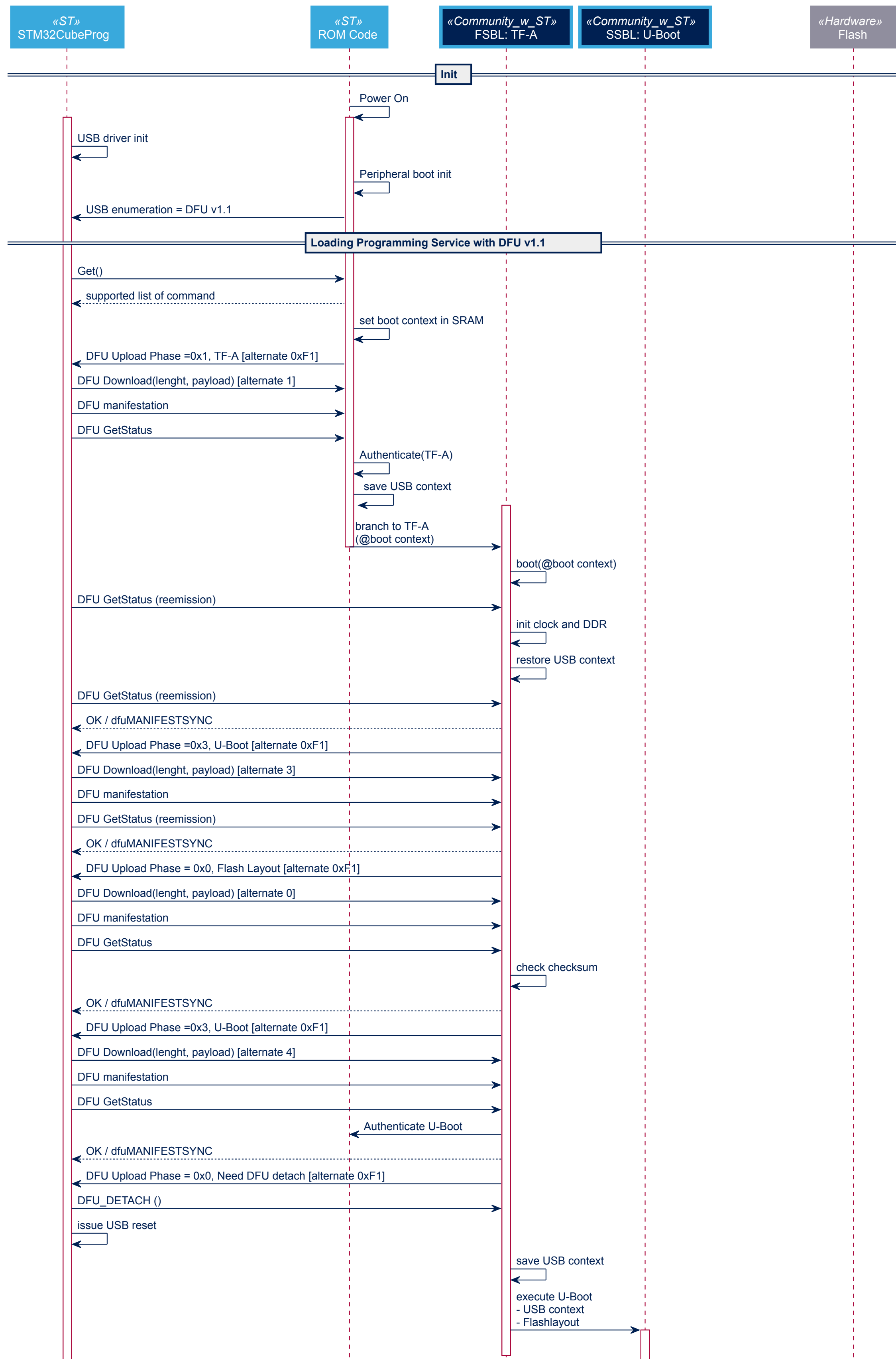
Since the Layout file is needed for USB enumeration in U-Boot, the phase ID = 0 is loaded by TF-A and provided to U-Boot.

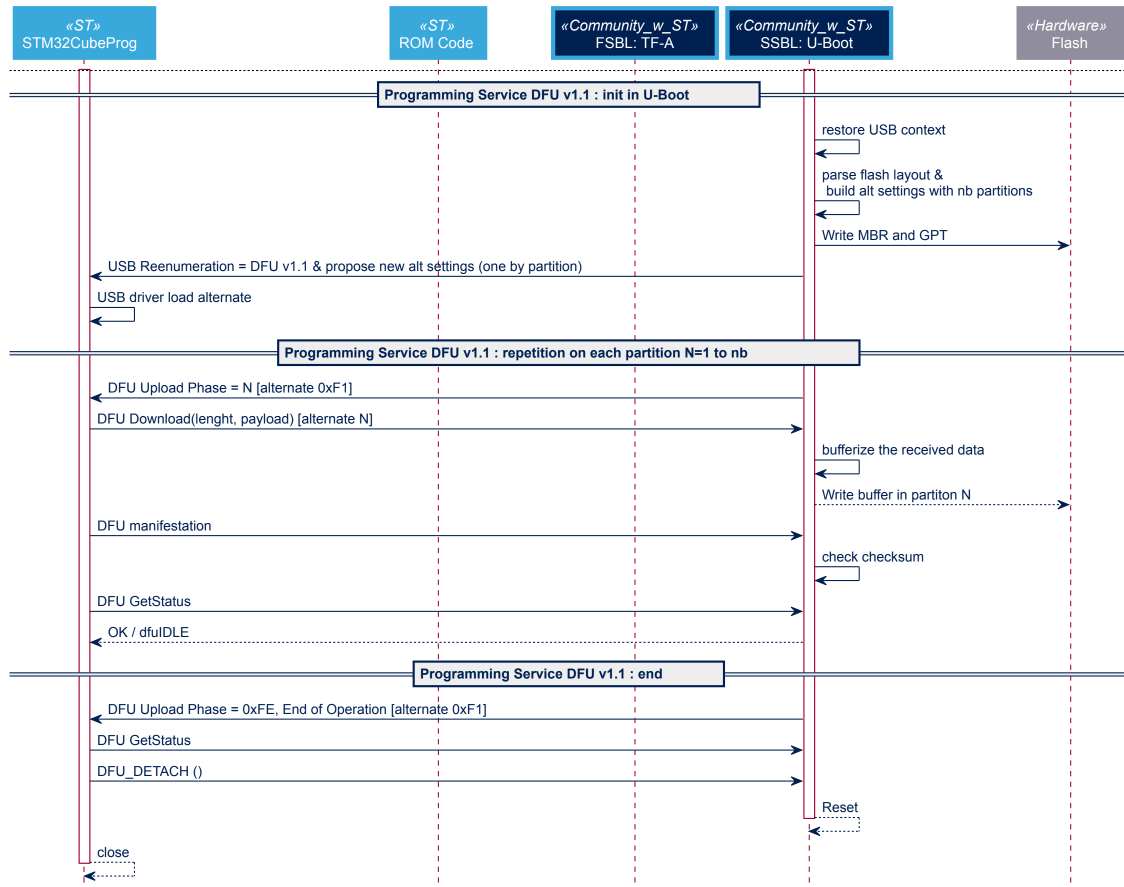
Between TF-A and U-Boot when the embedded programming service is loaded in RAM and started, the USB re-enumeration is requested according the following sequence:

1. Host request state, device answer is dfuIdle (state 2)
2. Host request DFU_UPLOAD (GetPhase)
3. Device indicate the same Phase 0x0 but with the “need detach” flag in additional information bytes.
4. The host sends command DFU_DETACH:
   the U-Boot code is started by TF-A,
   USB stack is initialized : DFU in dfuIdle mode
5. the Host reset the USB

Figure 10 presents the sequence in more details.

Figure 9. USB sequence





3.3 DFU enumeration and alternate settings

Each partition to be loaded in RAM for ROM code and TF-A or to be programmed in NVM for U-Boot is available with alternate settings of the DFU profile.

The Phase ID is aligned with the alternate setting identifier.

The device support GetPhase with UPLOAD on a specific alternate setting (the last one).

The name of the alternate setting string descriptor respect the description of [3], chapter 10:

```
@Target Memory Name/Start Address/Sector(1)_Count*Sector(1)_Size
Sector(1)_Type,Sector(2)_Count*Sector(2)_SizeSector(2)_Type,...
...,Sector(n)_Count*Sector(n)_SizeSector(n)_Type
```

Since the partitions for STM32MP1 are not memory mapped, the “Start Address” is replaced by partition IDs (0x + 2 digits for phase ID).

The partitions have only one sector and the size is computed from the Layout file.

Only 2 type of partitions are supported for STM32MP1:

- a (0x41): Readable for partition NOT selected in Layout file.
- e (0x45): Readable and writeable for partition expected to be updated.

Some virtual partitions (not directly linked to memory on NVM) are added and use reserved phase ID:

- 0xF1 = partition ID reserved for command GetPhase.
- 0xF2 = partition ID reserved for OTP
- 0xF3 = partition ID reserved for PMIC NVM

3.3.1 The virtual command partitions 0xF1 (GetPhase/SetOffset/Start):

The virtual partition 0xF1 is used for the following commands:
- DFU_UPLOAD : the GetPhase command.
- DFU_DOWNLOAD : the SetOffset and Start command.

3.3.1.1 DFU download

Only supported in U-Boot and in provisioning TF-A
- U-Boot.

The format of SetOffset command is:

Table 12. DFU SetOffset command

Byte	Description
1	Phase ID : normal partition between 0x00 to 0xF0
2-5	Offset in the partition

The current Phase ID is updated (for GetPhase) and the offset is going to be used for next upload/download on the selected partitions.
The format of Start command, to execute code loaded in memory, is:

Table 13. DFU Start command

Byte	Description
1	0xFF
2-5	Address

3.3.1.2 DFU upload

The content of the upload on this alternate setting is GetPhase command result:

Table 14. DFU GetPhase command response

Byte	Description
1	Phase ID
2-5	Download address
6-9	Offset
10-Last	Additional information optional, size: 0 up to 250 bytes

The download address, when present, provides the load address in memory. A value of 0xFFFFFFFF means than the partition identified by phase ID is going to be written in NVM.
The offset field provides the current offset used in partition operation, it is 0 by default.
The content of the additional information is determined by the phase ID value:
- Phase ID = 0xFF, the cause of the error which causes the reset request in a string
- Phase ID = 0x0, for Layout file the information byte is:
  - Byte 1 Need Reset Indication:

    = 1 if DFU_DETACH is requested
    = 0 or absent if DFU_DETACH is not requested

Example for TF-A execution:

1. Load of phase 0, Layout file in RAM
   Byte 1 0x00
   Byte 2-5 0xFFFFFFFF
2. Request DFU_DETACH after manifestation of phase 0:
   Byte 1 0x00
   Byte 2-5 0xFFFFFFFF
   Byte 6 0x01

Example for ROM code execution:

1. Load of phase 1, FSBL1: TF-A in internal RAM
   Byte 1 0x01
   Byte 2-5 0x2FFFFFF
   Byte 6 0x01 or 0x00 SSP state

3.3.2 ROM code – 1st enumeration

Since the STMP32MP DFU application supports the download of several partitions, the ROM code needs to indicate it is manifestation-tolerant, (bitMainfestationTolerant = 1 in DFU attributes).

At the first enumeration, the alternate settings present the partitions that are used by the ROM code and by FSBL.

Table 15. ROM code USB enumeration

Alternate setting	PhaseID	String descriptor	ROM code support	TF-A support
0	0	@Partition0 /0x00/1*256Ke	No	Layout
1	1	@FSBL /0x01/1*1Me	Yes	No
2	2	@Partition2 /0x02/1*1Me	No	No
3	3	@Partition3 /0x03/1*16Me	No	SSBL
4	0xF1	@virtual /0xF1/1*512Ba	GetPhase	GetPhase

The sizes are chosen to be flexible even if they are fixed in ROM code:

- Partititon0 = Layout file expected less than 256KB
- FSBL = TF-A size is limited by internal RAM (set max to 1MB)
- Partition2 = Reserved for future use (1MB).
- Partition3 = SSBL : 16MB used to load kernel directly
- Partition4 = Reserved for future use (16MB).
- 0xF1 = GetPhase : read-only, limited to 256 bytes normally

3.3.3 U-Boot – 2nd enumeration

U-Boot presents all the needed alternate settings:

- Each partition in NVM or each device, defined in the Layout file (phase, size, type is writable only if selected), with Address = Phase ID
- Memory mapping region, when it is added in U-Boot
- Special region (address = reserved, phase ID > 0xF0)

It depends on U-Boot settings and Layout file content; here is an example:

Table 16. U-Boot USB enumeration

Alternate setting	PhaseID	String descriptor
0	0	**@Flashlayout** /0x00/1*4Ke
1	1	**@fsbl1 /0x01/**1*256Ke

Alternate setting	PhaseID	String descriptor
2	2	**@fsbl2 /0x02/**1*256Ke
3	3	**@ssbl /0x03/**1*512Ke
4	0x10	@/bootfs /0x10/1*64Me
5	0x11	@/rootfs /0x10/1*512Me
N	M	Last user partion in Layout (N<0xF0)
N+1	-	@sysram /0x2FFF000/1*256Ke
N+2	-	@mcuram /0x30000000/1*284Ke
N+3	-	@ddr /0xC0000000/1*1Ge
Last-1	0xF1	@virtual /0xF1/1*512Ba
Last	0xF2	@OTP /0xF2/1*512Be

3.4 DFU stack in ROM code and TF-A

TF-A and ROM code share the same DFU v1.1 stack.

3.4.1 ROM code

The boot ROM initializes the USB and the DFU, and allows the loading and the execution of TF-A in internal RAM .

Before execution of TF-A, the USB stack needs to be interrupted, the USB context saved and provided in boot parameter to TF-A

3.4.2 TF-A

In TF-A, the USB context is restored and the DFU stack is resumed (no USB reset, no enumeration). At the end of the ROM code, the manifestation is requested, the state is dfuMANIFESTSYNC (state 6) and the first request received by TF-A should return dfuIDLE.

3.5 DFU stack in U-Boot

The existing DFU stack of U-Boot is used.

The USB stack, the controller and the USB PHY should be initialized in DFU mode when the USB download mode is indicated by ROM code or TF-A.

The partitions list is built from the Layout file provided by FSBL (TF-A).

The GetPhase command is available by download command on partition 0xF1.

STM32CubeProgrammer should loop until phase is 0xFE or 0xFF:

- Get current Phase: DFU_UPLOAD (GetPhase = 0xF1)
- Found associated file in field “Binary” of Layout file
- Download file in associated alternate settings
- Manifestation and pool manifestation end

For NVM partition, the upload / download operation accesses the NVM and the manifestation flushes the last operation in the device.

For “memory” partition, the manifestation only flushes the current operation and the cache. The start in memory operation is only done with DFU_DOWNLOAD on virtual partition (see 3.3.1).

Revision history

Table 17. Document revision history

Date	Version	Changes
11-Oct-2019	1	Initial release.
17-Mar-2022	2	Updated Section 1.1 Introduction
23-Mar-2022	3	Updated Figure 9. Interface state transition diagram

Contents

1 **Embedded programming service** 2
1.1 Introduction 2
1.2 Reference 2
1.3 Overview 3
1.4 Layout file format 4
1.5 Phase ID 4
1.6 STM32 image header 4
1.7 Programming sequence 4
1.7.1 Case 1 – programming from reset 4
1.7.2 Case 2 – programming from U-Boot for a device already programmed 6
2 **UART/USART** 9
2.1 UART/USART protocol 9
2.2 UART/USART configuration 11
2.3 UART/USART connection 12
2.4 UART/USART main loop 12
2.5 UART/USART command set 14
2.5.1 Get command (0x00) 14
2.5.2 Get ID command (0x02) 15
2.5.3 Get version command (0x01) 15
2.5.4 Get phase command (0x03) 15
2.5.5 Download command (0x31) 17
2.5.6 Read memory command (0x11) 18
2.5.7 Read partition command (0x12) 18
2.5.8 Start command (0x21) 19
3 **USB** 20
3.1 DFU protocol 20
3.2 USB sequence 22
3.3 DFU enumeration and alternate settings 24
3.3.1 The virtual command partitions 0xF1 (GetPhase/SetOffset/Start): 25
3.3.2 ROM code – 1st enumeration 26

3.3.3 U-Boot – 2nd enumeration 26
3.4 DFU stack in ROM code and TF-A 27
3.4.1 ROM code 27
3.4.2 TF-A 27
3.5 DFU stack in U-Boot 27
Revision history 28

List of tables

Table 1. Phase ID . . . 4
Table 2. USART commands . . . 14
Table 3. Get command response . . . 14
Table 4. Get ID command response . . . 15
Table 5. Get version command response . . . 15
Table 6. Get phase command response . . . 16
Table 7. Packet number . . . 17
Table 8. Download command. . . . 18
Table 9. Read memory command. . . . 18
Table 10. Read Partition command. . . . 19
Table 11. Start command . . . 19
Table 12. DFU SetOffset command . . . 25
Table 13. DFU Start command. . . . 25
Table 14. DFU GetPhase command response . . . 25
Table 15. ROM code USB enumeration . . . 26
Table 16. U-Boot USB enumeration . . . 26
Table 17. Document revision history . . . 28

List of figures

Figure 1. Programming operation . . . 3
Figure 2. Programing chart . . . 5
Figure 3. Programming Sequence . . . 6
Figure 4. Programming sequence for boot from NVM . . . 8
Figure 5. Usart programming sequence . . . 10
Figure 6. Main UART/USART sequence . . . 12
Figure 7. STM32CubeProgrammer download sequence . . . 13
Figure 8. Interface state transition diagram . . . 21
Figure 9. USB sequence . . . 23

IMPORTANT NOTICE – READ CAREFULLY

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

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

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

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

ST and the ST logo are trademarks of ST. For additional information about ST trademarks, refer to www.st.com/trademarks. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2022 STMicroelectronics – All rights reserved