# AN2662
# Application note

## STM32F105xx and STM32F107xx system memory boot mode

### Introduction

This application note describes the bootloader stored in the system memory of the STM32F105xx and STM32F107xx Connectivity Line Microcontrollers. All STM32F105xx and STM32F107xx in production (rev. Z) include the bootloader detailed in this application note.

The bootloader is used to program the application into the internal Flash memory. For more information about the Flash module organization, refer to the STM32F10xxx Flash programming manual (PM0042).

The specifications cover the architectural model of the bootloader for the STM32F105xx and STM32F107xx family, but the low-level communication software supports all the microcontroller families that implement the bootloader.

# Contents

# List of tables

# List of figures

# 1 Bootloader description

## 1.1 Bootloader introduction

The bootloader is stored in the internal boot ROM memory (system memory), and its main task is to download the application program to the internal Flash memory through one of the following communication interfaces: USART1, USART2 (remapped), CAN2 (remapped) or USB OTG FS in Device mode (DFU: device firmware upgrade).

The main features of the bootloader are the following:

- It uses an embedded communication peripheral to download the code
- It transfers and updates the Flash memory code, the data, and the vector table sections

*Note: The protocol used for STM32F105xx and STM32F107xx's USART1/2 bootloader is fully compatible with the protocol used for the USART1 bootloader in STM32 Low-, Medium- and High-density devices (as described in AN2606.)*

## 1.2 Bootloader activation

The bootloader is automatically activated by configuring the BOOT0 and BOOT1 pins in the specific “System memory” configuration (see *Table 1*) and then by applying a reset.

Depending on the used pin configuration, the Flash memory, system memory or SRAM is selected as the boot space, as shown in *Table 1* below.

**Table 1. Boot pin configuration**

| Boot mode selection pins | | Boot mode | Aliasing |
|---|---|---|---|
| BOOT1 | BOOT0 | | |
| X | 0 | User Flash memory | User Flash memory is selected as the boot space |
| 0 | 1 | System memory | System memory is selected as the boot space |
| 1 | 1 | Embedded SRAM | Embedded SRAM is selected as the boot space |

*Table 1* shows that the STM32F105xx and STM32F107xx enters the System memory boot mode if the BOOT pins are configured as follows:

- BOOT0 = 1
- BOOT1 = 0

The values on the BOOT pins are latched on the fourth rising edge of SYSCLK after a reset.

**Table 2. STM32F105xx and STM32F107xx configuration in System memory boot mode**

| Bootloader | Feature/Peripheral | State | Comment |
|---|---|---|---|
| Common to all bootloaders | RCC | HSI enabled | The system clock frequency is 24 MHz using the PLL. This is used only for USART1 and USART2 bootloaders and during CAN2, USB detection for CAN and DFU bootloaders (Once CAN or DFU bootloader is selected, the clock source will be derived from external crystal). |
| | | HSE enabled | The external clock is mandatory only for DFU and CAN bootloaders and it must provide one of the following frequencies: 8 MHz, 14.7456 MHz or 25 MHz.<br>For CAN Bootloader, the PLL is used only to generate 48 MHz when 14.7456 MHz is used as HSE.<br>For DFU Bootloader, the PLL is used to generate a 48 MHz system clock from all supported external clock frequencies. |
| | | - | The clock security system (CSS) interrupt is enabled for the CAN and DFU bootloaders. Any failure (or removal) of the external clock will generate system reset. |
| | IWDG | - | The independent watchdog (IWDG) prescaler is configured to its maximum value and is periodically refreshed to prevent watchdog reset (in case the hardware IWDG option was previously enabled by the user). |
| | System memory | - | 18 Kbytes starting from address 0x1FFF B000, contain the bootloader firmware |
| | RAM | - | 4 Kbytes starting from address 0x2000 0000 are used by the bootloader firmware. |
| USART1 bootloader | USART1 | Enabled | Once initialized the USART1 configuration is: 8-bits, even parity and 1 Stop bit |
| | USART1_RX pin | Input | PA10 pin: USART1 receive |
| | USART1_TX pin | Output | PA9 pin: USART1 transmit |
| USART1 and USART2 bootloaders | SysTick timer | Enabled | Used to automatically detect the serial baud rate from the host for USARTx bootloader. |
| USART2 bootloader | USART2 | Enabled | Once initialized the USART2 configuration is: 8-bits, even parity and 1 Stop bit. The USART2 uses its remapped pins. |
| | USART2_RX pin | Input | PD6 pin: USART2 receive (remapped pin) |
| | USART2_TX pin | Output | PD5 pin: USART2 transmit (remapped pin) |

**Table 2. STM32F105xx and STM32F107xx configuration in System memory boot mode**

| Bootloader | Feature/Peripheral | State | Comment |
|---|---|---|---|
| CAN2 bootloader | CAN2 | Enabled | Once initialized the CAN2 configuration is: Baudrate 125 kbps,<br>11-bit identifier.<br>**Note:** CAN2 uses remapped pins. |
| | CAN2_RX pin | Input | PB5 pin: CAN2 receive (remapped pin) |
| | CAN2_TX pin | Output | PB6 pin: CAN2 transmit (remapped pin) |
| DFU bootloader | USB OTG FS | Enabled | USB OTG FS configured in Forced device mode |
| | OTG_FS_VBUS pin | Input or alternate function, automatically controlled by the USB OTG FS controller | PA9: Power supply voltage line |
| | OTG_FS_DM pin | | PA11: USB Send-Receive data line |
| | OTG_FS_DP pin | | PA12: USB Send-Receive data line |
| | Interrupts | Enabled | USB_OTG_FS interrupt vector is enabled and used for USB DFU communication. |

The system clock is derived from the embedded internal high-speed RC for USARTx bootloader. This internal clock is used also for DFU and CAN bootloaders but only for the selection phase. An external clock (8 MHz, 14.7456 MHz or 25 MHz.) is required for DFU and CAN bootloader execution after the selection phase.

After downloading the application binary, if you choose to execute the Go command, all peripheral registers used by the bootloader (shown in the above table) will be initialized to their default reset values before jumping to the user application.
If the user application uses the IWDG, the IWDG prescaler value has to be adapted to meet the requirements of the application (since the prescaler was set to its maximum value by the bootloader).

## 1.3 Hardware requirements

The hardware required to put the STM32F105xx and STM32F107xx into System memory boot mode consists of any circuitry, switch or jumper, capable of holding the BOOT0 pin high and the BOOT1 pin low during reset.

To connect to the STM32F105xx and STM32F107xx during System memory boot mode, the following conditions have to be verified:

- The RX pins of the unused peripherals in this bootloader have to be kept at a known (low or high) level, and should not be left floating during the detection phase as described below:
  - If USART1 is used to connect to the bootloader: the USART2_RX (PD6) and CAN2_RX (PB5) pins have to be kept at a high or low level and must not be left floating during the detection phase.
  - If USART2 is used to connect to the bootloader: the USART1_RX (PA10), CAN2_RX (PB5), OTG_FS_DM (PA11) and OTG_FS_DP (PA12) pins have to be kept at a high or low level and must not be left floating during the detection phase.
  - If CAN2 is used to connect to the bootloader: the USART1_RX (PA10), USART2_RX (PD6), OTG_FS_DM (PA11) and OTG_FS_DP (PA12) pins have to be kept at a high or low level and must not be left floating during the detection phase.

  - If DFU is used to connect to the bootloader: the USART1_RX (PA10), USART2_RX (PD6) and CAN2_RX (PB5) pins have to be kept at a high or low level and must not be left floating during the detection phase.

- Connect to the peripheral to be used through:
  - an RS232 serial interface (example, ST3232 RS232 transceiver) has to be directly connected to the USART1_RX (PA10) and USART1_TX (PA9) pins when USART1 is used, or to the USART2_RX (PD6) and USART2_TX (PD5) pins when USART2 is used
  - a CAN interface (CAN transceiver) has to be directly connected to the CAN2_RX (PB5) and CAN2_TX (PB6) pins
  - a certified USB cable has to be connected to the microcontroller (optionally an ESD protection circuitry can be used)

The USART1_CK, USART1_CTS and USART1_RTS pins are not used, therefore the application can use these pins for other peripherals or GPIOs. The same note is applicable for USART2.

Once the USB Device is enabled, all its related pins are dedicated to USB communication only, and cannot be used for other application purposes.

The user can control the BOOT0 and Reset pins from a PC serial applet using the RS232 serial interface which controls BOOT0 through the CTS line and Reset through the DCD line. The user must use a full null modem cable. The necessary hardware to implement for this control exists in the STM3210C-EVAL board. For more details about this, refer to document: “STM3210C-EVAL board user manual”, available from the STMicroelectronics website: www.st.com.

## 1.4 Bootloader selection

The STM32F105xx and STM32F107xx  embedded bootloader supports four peripherals interfaces: USART1, USART2, CAN2 and DFU (USB). Any one of these peripheral interfaces can be used to communicate with the bootloader and download the application code to the internal Flash.

The embedded bootloader firmware is able to auto-detect the peripheral interface to be used. In an infinite loop, it detects any communication on the supported bootloader interfaces.

*Note: The RX pins of the unused peripherals in this bootloader have to be maintained at a known (low or high) level and should not be left floating during the detection phase as described below:*

- *If USART1 is used to connect to the bootloader: the USART2_RX (PD6) and CAN2_RX (PB5) pins have to be kept at a high or low level and must not be left floating during the detection phase.*
- *If USART2 is used to connect to the bootloader: the USART1_RX (PA10), CAN2_RX (PB5), OTG_FS_DM (PA11) and OTG_FS_DP (PA12) pins have to be kept at a high or low level and must not be left floating during the detection phase.*
- *If CAN2 is used to connect to the bootloader: USART1_RX (PA10), USART2_RX (PD6), OTG_FS_DM (PA11) and OTG_FS_DP (PA12) pins have to be kept at a high or low level and must not be left floating during the detection phase.*
- *If DFU is used to connect to the bootloader: the USART1_RX (PA10), USART2_RX (PD6) and CAN2_RX (PB5) pins have to be kept at a high or low level and must not be left floating during the detection phase.*

To use the USART bootloader on USART1 or USART2, connect the serial cable to the desired interface. Once the bootloader detects the data byte 0x7F on this interface, the bootloader firmware executes the autobaudrate sequence and then enters a loop, waiting for any USART bootloader command.

To use the CAN2 interface, connect the CAN cable to CAN2. Once the bootloader detects a frame on the CAN2_RX pin (PB5), the bootloader firmware enters a CAN loop and starts to check the external clock frequency value, if the HSE is 8 MHz, 14.7456 MHz or 25 MHz CAN bootloader firmware enters an infinite loop and waits until it receives a message, otherwise a system reset is generated.

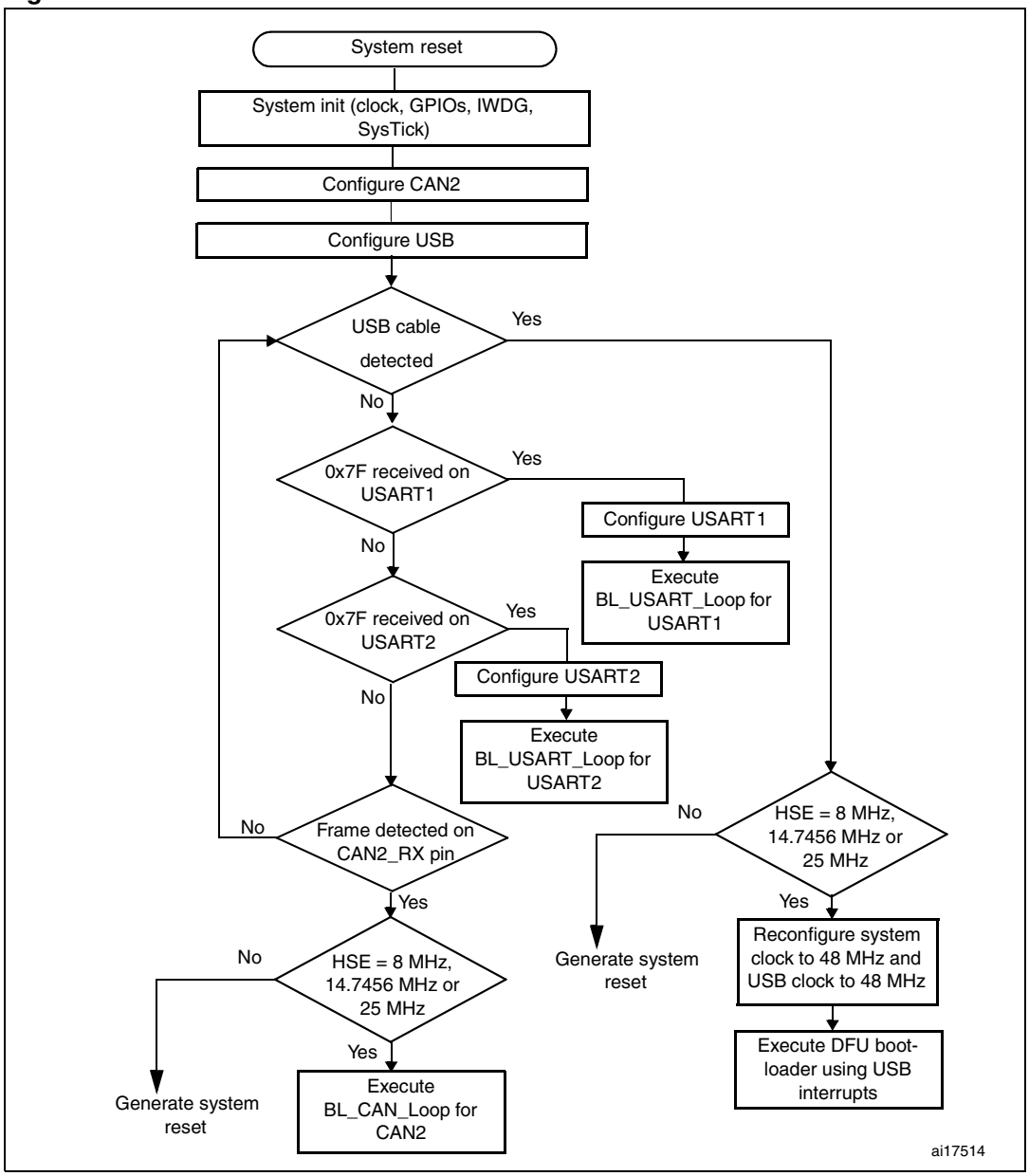
If a USB cable is plugged into the microcontroller's USB interface at any time during the bootloader firmware selection sequence, the bootloader then enters the DFU bootloader loop waiting for any DFU bootloader command.

To use the USART or the CAN bootloader, it is mandatory that no USB cable is connected to the USB peripheral during the selection phase. Once the USART or CAN bootloader is selected, the user can plug a USB cable without impacting the selected bootloader execution except commands which generate a system reset.

Once one interface is selected for the bootloader, all other interfaces are disabled.

The figure below shows the bootloader detection mechanism. More details are provided in the sections corresponding to each peripheral bootloader.

**Figure 1. Bootloader selection**



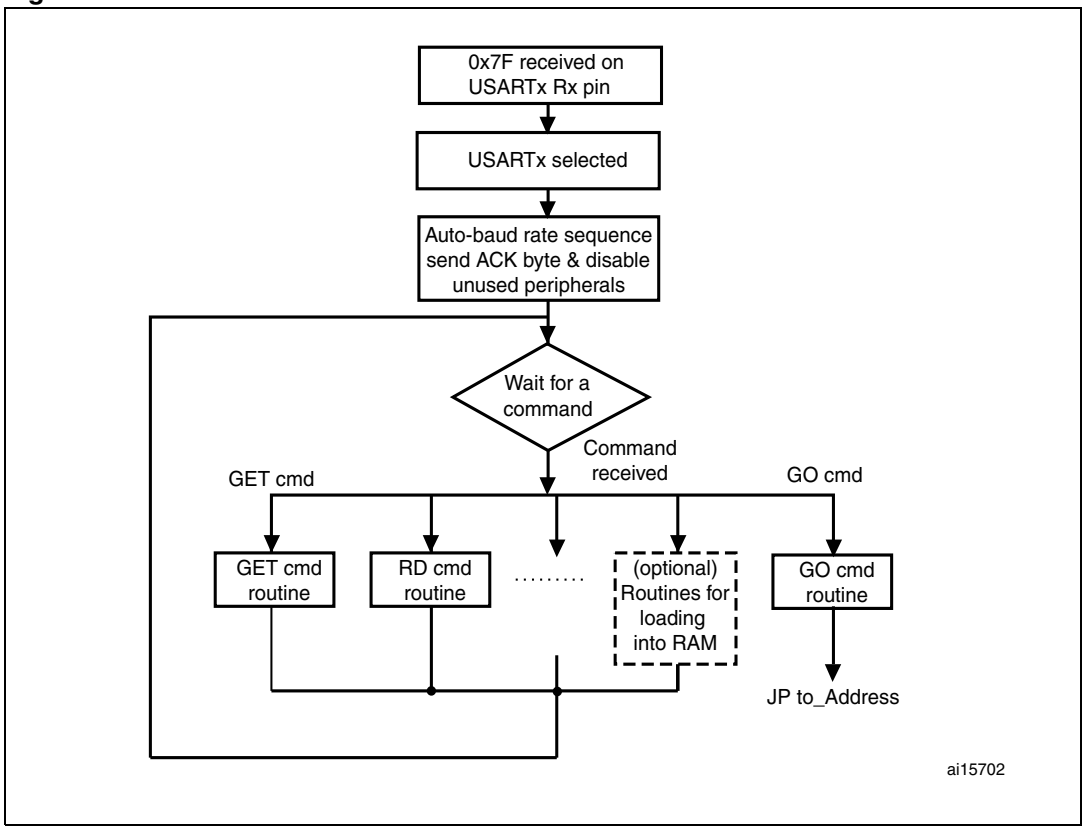

## 1.5 Exiting system memory boot mode

System memory boot mode must be exited in order to start execution of the application program. This can be done by applying a hardware reset. During reset, the BOOT pins (BOOT0 and BOOT1) must be set at the proper levels to select the desired boot mode (see *Table 1*). Following the reset, the CPU starts code execution from the boot memory located at the bottom of the memory address space starting from 0x0000 0000.

# 2 USART bootloader

## 2.1 Bootloader code sequence

**Figure 2. Bootloader for STM32F105xx and STM32F107xx with USART1/USART2**




Once System memory boot mode is entered and the microcontroller has been configured as described above, the bootloader code begins to scan the USARTx_RX line pin, waiting to receive the 0x7F data frame: one start bit, 0x7F data bits, even parity bit and one stop bit.

The duration of this data frame is measured using the Systick timer. The count value of the timer is then used to calculate the corresponding baud rate factor with respect to the current system clock.

Next, the code initializes the serial interface accordingly. Using this calculated baud rate, an acknowledge byte *(0x79)* is returned to the host, which signals that the STM32F105xx and STM32F107xx is ready to receive user commands.

## 2.2 Choosing the USARTx baud rate

The calculation of the serial baud rate for USARTx, from the length of the first byte that is received, is used to operate the bootloader within a wide range of baud rates. However, the upper and lower limits have to be kept, in order to ensure proper data transfer.

For a correct data transfer from the host to the microcontroller, the maximum deviation between the internal initialized baud rate for USARTx and the real baud rate of the host should be below 2.5%. The deviation ($f_B$, in percent) between the host baud rate and the microcontroller baud rate can be calculated using the formula below:

$$f_B = \left| \frac{\text{STM32Fxxx baud rate} - \text{Host baud rate}}{\text{STM32Fxxx baud rate}} \right| \times 100\% \text{ , where } f_B \leq 2.5\%.$$

This baud rate deviation is a nonlinear function depending on the CPU clock and the baud rate of the host. The maximum of the function ($f_B$) increases with the host baud rate. This is due to the smaller baud rate prescale factors, and the implied higher quantization error.

## 2.3 Minimum baud rate

The lowest tested baud rate ($B_{Low}$) is 1200. Baud rates below $B_{Low}$ would cause the SysTick timer to overflow. In this event, USARTx would not be correctly initialized.

## 2.4 Maximum baud rate

$B_{High}$ is the highest baud rate for which the deviation still does not exceed the limit. All baud rates between $B_{Low}$ and $B_{High}$ are below the deviation limit.
The highest tested baud rate ($B_{High}$) is 115 200.

## 2.5 Bootloader command set

The supported commands are listed in *Table 3* below. Each command is further described in this section.

**Table 3. USART bootloader commands**

| Command(1) | Command code | Command description |
|---|---|---|
| Get(2) | 0x00 | Gets the version and the allowed commands supported by the current version of the bootloader |
| Get Version & Read Protection Status(2) | 0x01 | Gets the bootloader version and the Read Protection status of the Flash memory |
| Get ID(2) | 0x02 | Gets the chip ID |
| Read Memory | 0x11 | Reads up to 256 bytes of memory starting from an address specified by the user |
| Go | 0x21 | Jumps to user application code located in the internal Flash memory or in SRAM |
| Write Memory | 0x31 | Writes up to 256 bytes to the RAM or Flash memory starting from an address specified by the user |

**Table 3. USART bootloader commands (continued)**

| Command[1] | Command code | Command description |
|---|---|---|
| Erase | 0x43 | Erases from one to all the Flash memory pages |
| Write Protect[3] | 0x63 | Enables the write protection for some sectors |
| Write Unprotect[3] | 0x73 | Disables the write protection for all Flash memory sectors |
| Readout Protect[2] | 0x82 | Enables the read protection |
| Readout Unprotect[2] | 0x92 | Disables the read protection |

1. If a denied command is received or an error occurs during the command execution, the bootloader sends NACK byte and goes back to command checking.
2. Read protection – When the RDP (read protection) option is active, only this limited subset of commands is available. All other commands are NACKed and have no effect on the device. Once the RDP has been removed, the other commands become active.
3. On the STM32F105xx and STM32F107xx , the sector size is 4 Kbytes (2 pages) for the Write Protect and Write Unprotect commands.

## Communication safety

All communications from the programming tool (PC) to the device are verified by:

1. checksum: all received bytes are XORed. A byte containing the computed XOR of all previous bytes is added to 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.
2. for each command the host sends a byte and its complement (XOR = 0x00)
3. UART: parity check active (even parity)

Each packet is either accepted (ACK answer) or discarded (NACK answer):

- ACK = 0x79
- NACK = 0x1F

## 2.6 Get command

The Get command allows the user to get the version of the bootloader and the supported commands. When the bootloader receives the Get command, it transmits the bootloader version and the supported command codes to the host, as described in *Figure 3*.

**Figure 3. Get command: host side**

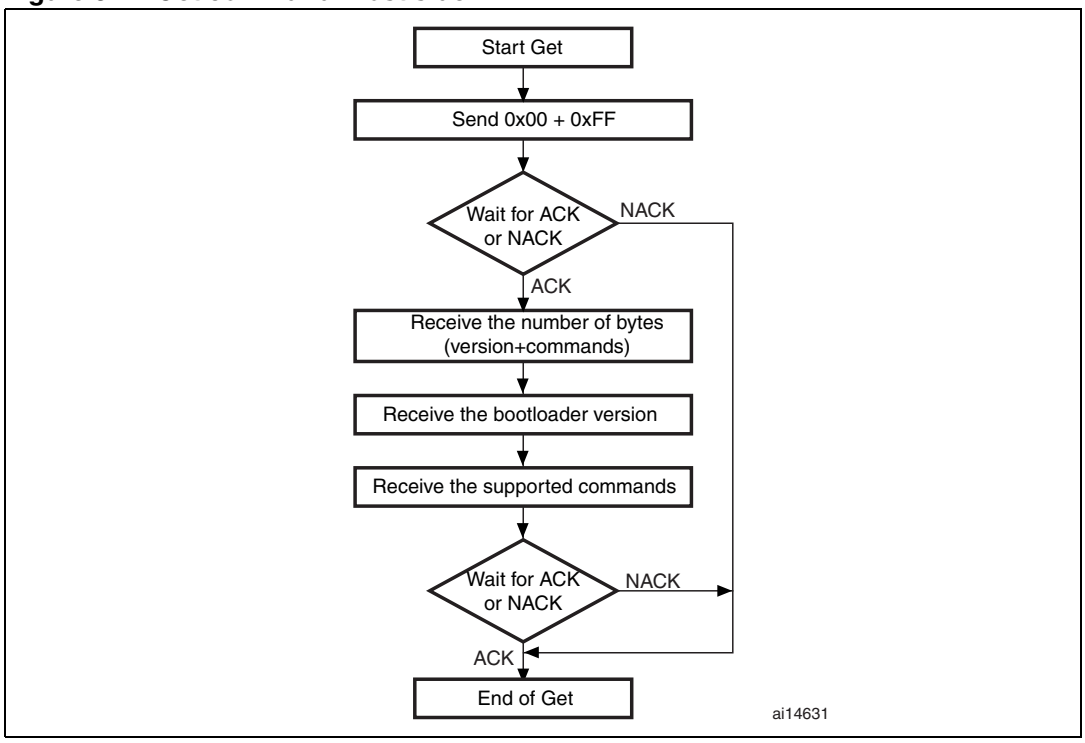
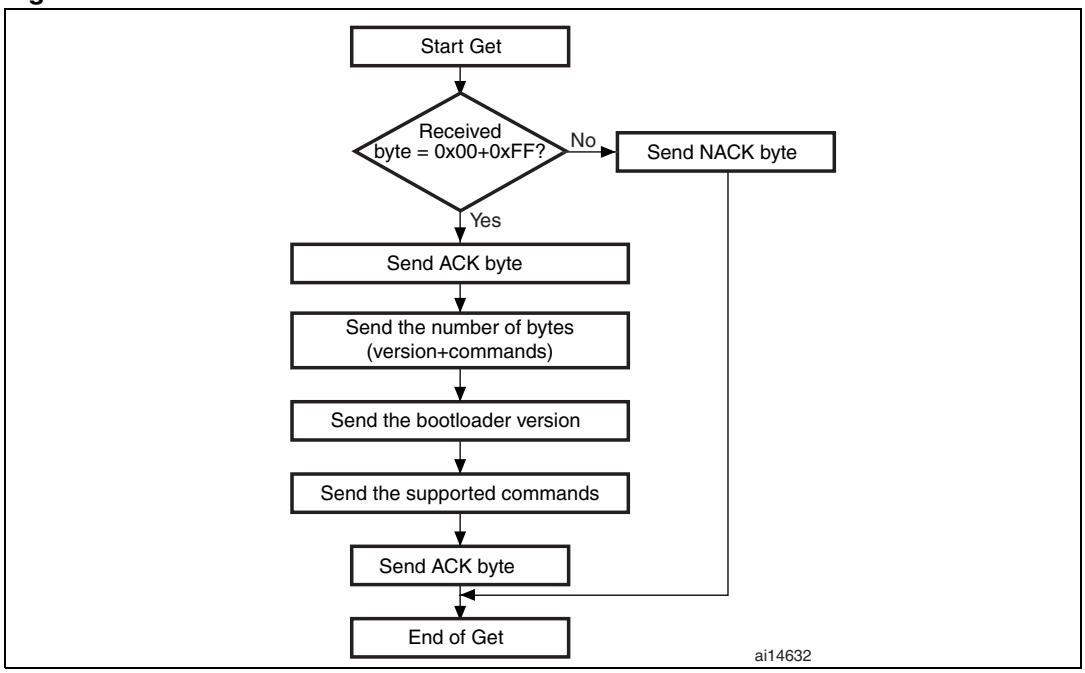



**Figure 4. Get command: device side**



The STM32F105xx and STM32F107xx sends the bytes as follows:

Byte 1: ACK

Byte 2: N = 11 = the number of bytes to follow – 1 except current and ACKs.

Byte 3: Bootloader version (0 < Version < 255), example: 0x10 = Version 1.0

Byte 4: 0x00 – Get command

Byte 5: 0x01 – Get Version and Read Protection Status

Byte 6: 0x02 – Get ID

Byte 7: 0x11 – Read Memory command

Byte 8: 0x21 – Go command

Byte 9: 0x31 – Write Memory command

Byte 10: 0x43 – Erase command

Byte 11: 0x63 – Write Protect command

Byte 12: 0x73 – Write Unprotect command

Byte 13: 0x82 – Readout Protect command

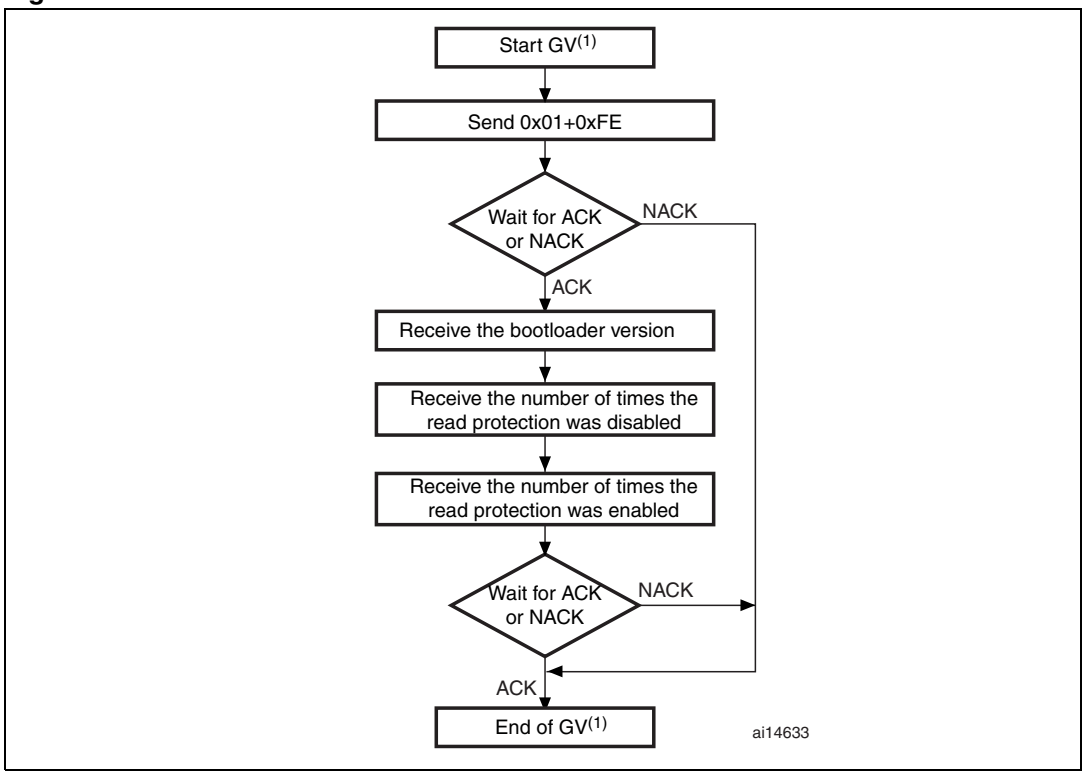
Byte 14: 0x92 – Readout Unprotect command

Last byte (15): ACK

## 2.7 Get Version & Read Protection Status command

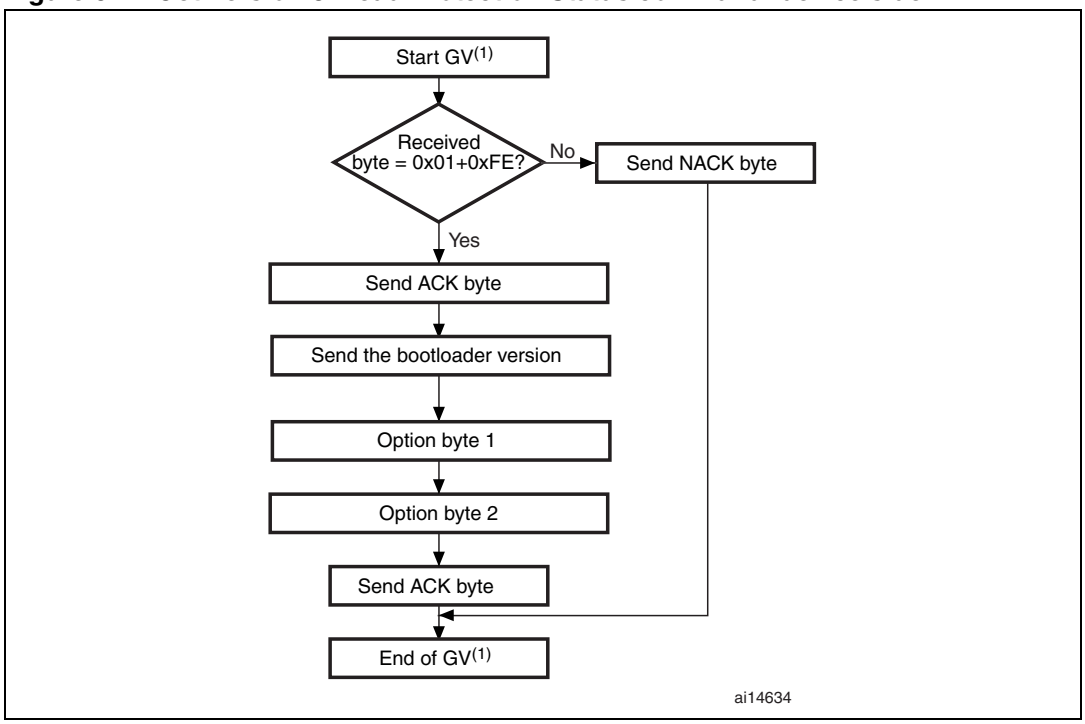
The Get Version & Read Protection Status command is used to get the bootloader version and the read protection status. When the bootloader receives the command, it transmits the information described below (version, read protection: number of times it was enabled and disabled) to the host.

**Figure 5. Get Version & Read Protection Status command: host side**




1. GV = Get Version & Read Protection Status.

**Figure 6. Get Version & Read Protection Status command: device side**




1. GV = Get Version & Read Protection Status.

The STM32F105xx and STM32F107xx sends the bytes as follows:

Byte 1: ACK

Byte 2: Bootloader version (0 < Version ≤ 255), example: 0x10 = Version 1.0

Byte 3: Option byte 1: 0x00 to keep the compatibility with generic bootloader protocol

Byte 4: Option byte 2: 0x00 to keep the compatibility with generic bootloader protocol

Byte 5: ACK

## 2.8 Get ID command

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

The STM32F105xx and STM32F107xx sends the bytes as follows:

Byte 1: ACK

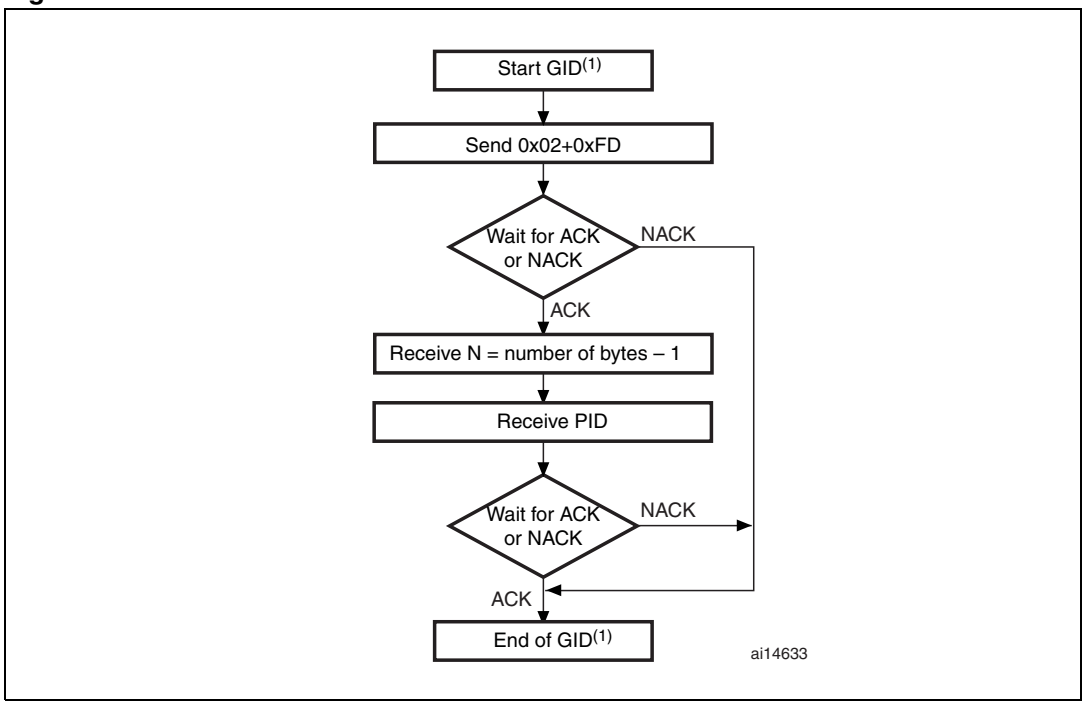
Byte 2: N = the number of bytes – 1 (N = 1 for STM32F105xx and STM32F107xx ), except for current byte and ACKs.

Bytes 3-4: PID(1) byte 3 = 0x04, byte 4 = 0x1X

Byte 5: ACK

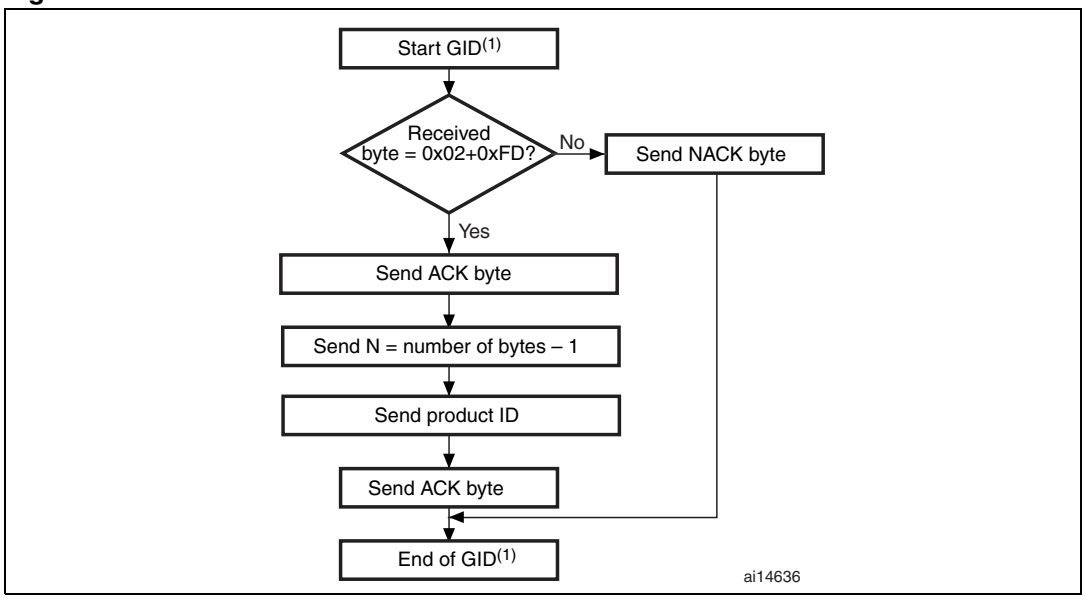
1. PID stands for product ID. It is 0x0418 for STM32F105xx and STM32F107xx products.

**Figure 7. Get ID command: host side**




1. GID = Get ID.

**Figure 8. Get ID command: device side**




1. GID = Get ID.

## 2.9 Read Memory command

The Read Memory command is used to read data from any memory address in RAM (starting from address 0x20001000), Flash memory and information block (System memory or option byte areas).
When the bootloader receives the Read Memory command, it transmits the ACK byte to the user. After the transmission of the ACK byte, the bootloader waits for an address (4 bytes, byte 1 is the address MSB and byte 4 is the LSB) and a checksum byte, then it checks the received address. If the address is valid and the checksum is correct, the bootloader 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 bootloader 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 + 1) bytes) to the user, starting from the received address. If the checksum is not correct, it sends a NACK before aborting the command.

The host sends the bytes to the STM32F105xx and STM32F107xx as follows:

Bytes 1-2: 0x11+0xEE

Wait for ACK

Bytes 3 to 6: start address

- byte 3: MSB
- byte 6: LSB

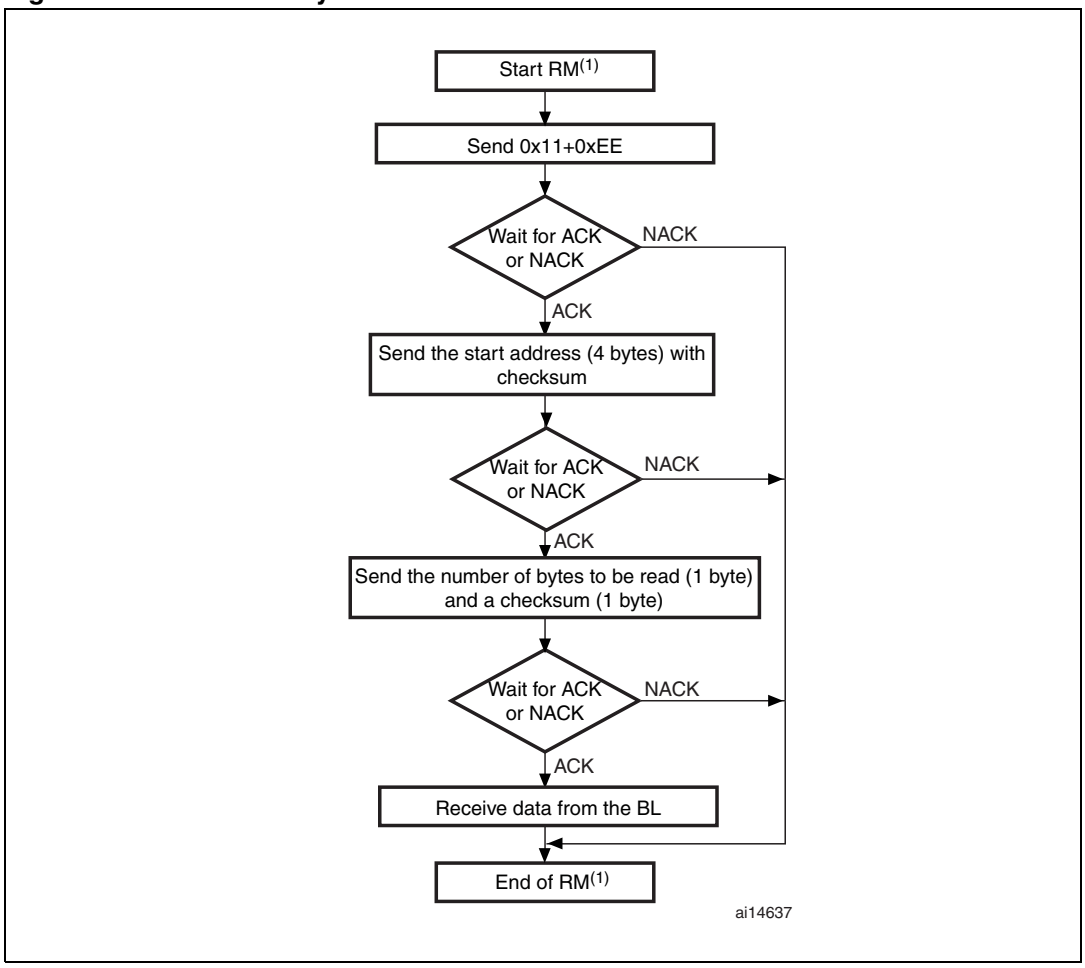
Byte 7: Checksum: XOR (byte 3, byte 4, byte 5, byte 6)

Wait for ACK

Byte 8: The number of bytes to be read – 1 ($0 < N \leq 255$);

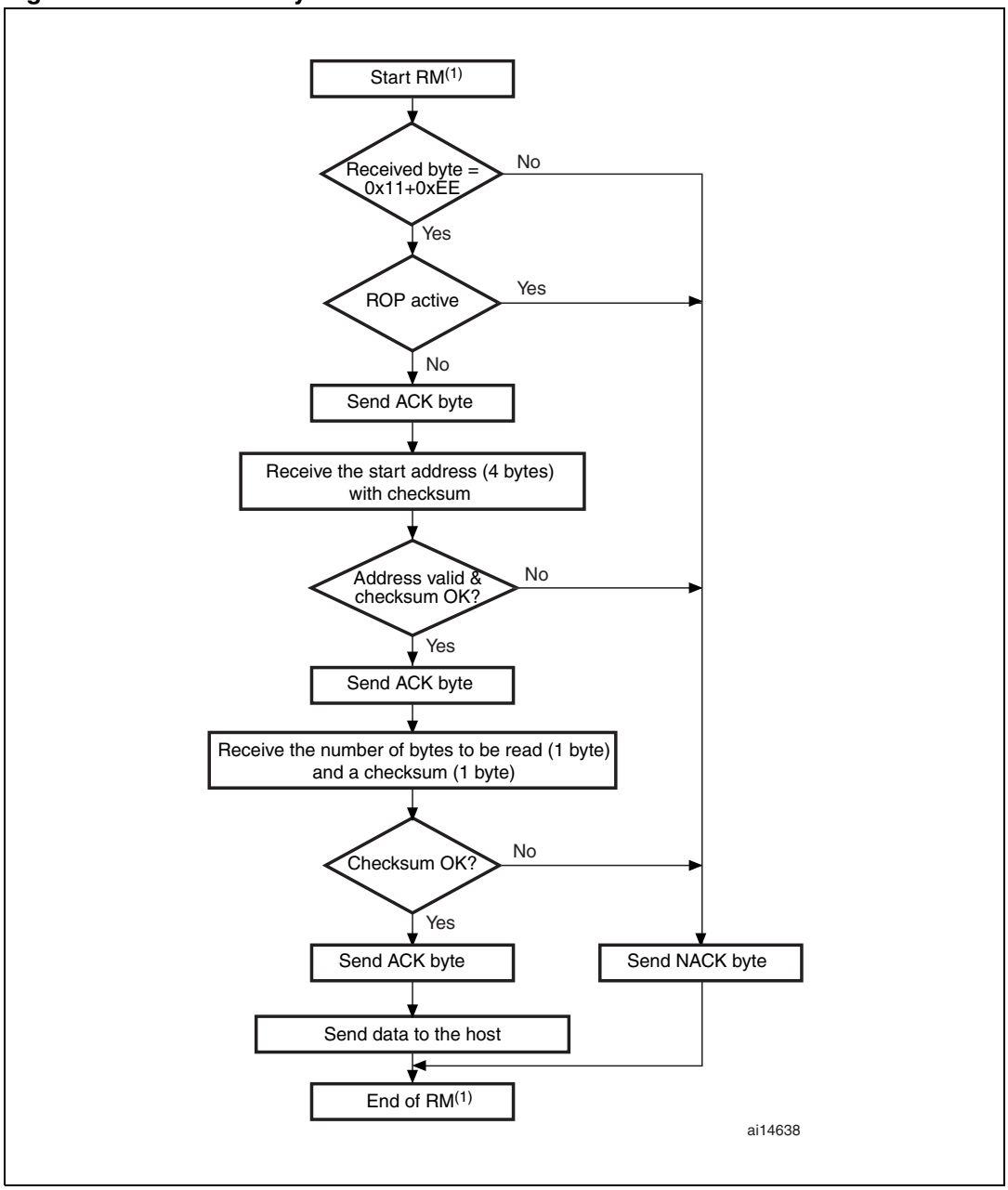
Byte 9: Checksum: XOR byte 8 (complement of byte 8)

**Figure 9. Read Memory command: host side**




1. RM = Read Memory.

**Figure 10. Read Memory command: device side**




1. RM = Read Memory.

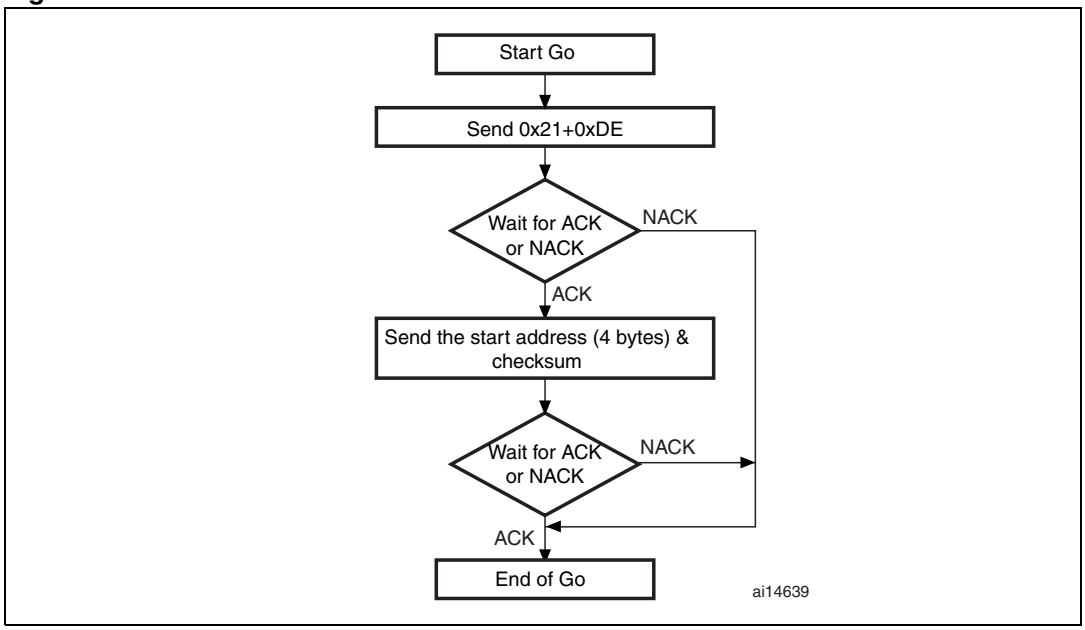
## 2.10 Go command

The Go command is used to execute the downloaded code or any other code by branching to an address specified by the user. When the bootloader receives the Go command, it transmits the ACK byte to the user. After the transmission of the ACK byte, the bootloader waits for an address (4 bytes, byte 1 is the address MSB and byte 4 is LSB) and a checksum byte, then it checks the received address. If the address is valid and the checksum is correct, the bootloader 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 bootloader firmware performs the following:

- it initializes the registers of the peripherals used by the bootloader (see *Table 2*) to their default reset values
- it initializes the user application's main stack pointer
- it jumps to the memory location programmed in the received 'address + 4' (which corresponds to the address of the application's reset handler).
  For example if the received address is 0x0800 0000, the bootloader will jump to the memory location programmed at address 0x0800 0004.
  In general, the host should send the base address where the application to jump to is programmed

**Figure 11. Go command: host side**



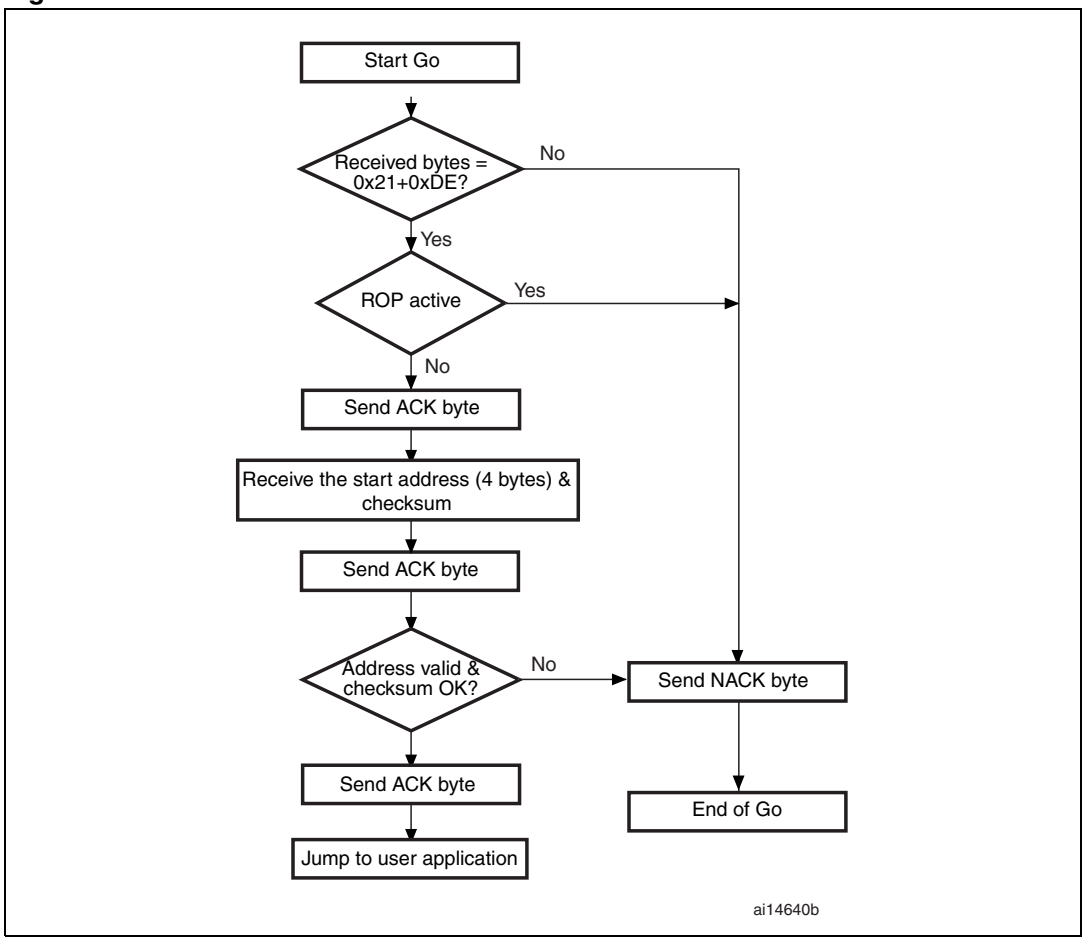

*Note: 1 Valid addresses are RAM (starting from 0x2000 1000 to the end of the RAM) or Flash memory (starting from 0x800 0000 to the end of the Flash memory) addresses. All other addresses are considered not valid and will be NACKed by the device.*

*2 When an application is loaded into RAM and then a jump is made to it, the program must be configured to run with an offset of at least 0x1000 to avoid overlapping with the first 0x1000 RAM memory used by the bootloader firmware.*

*3 The Jump to the application works only if the user application sets the vector table correctly to point to the application address.*

*4 When performing a jump from the Bootloader to a loaded application code which uses the USB IP, the user application has to disable all pending USB interrupts and reset the core before enabling interrupts. Otherwise, a pending interrupt (issued from the bootloader code) may interfere with the user code and cause a functional failure. This procedure is not needed after exiting system memory boot mode.*

**Figure 12. Go command: device side**




The host sends the bytes as follow to the STM32F105xx and STM32F107xx :

Byte 1: 0x21

Byte 2: 0xDE

Wait for ACK

Byte 3 to byte 6: start address

byte 3: MSB

byte 6: LSB

Byte 7: checksum: XOR (byte 3, byte 4, byte 5, byte 6)

## 2.11 Write Memory command

The Write Memory command is used to write data to any memory address of RAM starting from 0x2000 1000, Flash memory, or Option byte area. Refer to the STM32F10xxx Flash programming manual (PM0042). When the bootloader receives the Write Memory command, it transmits the ACK byte to the user. After the transmission of the ACK byte, the bootloader waits for an address (4 bytes, byte 1 is the address MSB and byte 4 is the LSB) and a checksum byte, it then checks the received address. For the Option byte area, the

start address must be 0x1FFFF800 to avoid writing inopportunely in this area.
If the received address is valid and the checksum is correct, the bootloader 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 bootloader:

- gets a byte, N, which contains the number of data bytes to be received
- receives the user data ((N + 1) bytes) and the checksum (XOR of N and of all data bytes)
- programs the user data to memory starting from the received address
- at the end of the command, if the write operation was successful, the bootloader transmits the ACK byte; otherwise it transmits a NACK byte to the user and aborts the command

The maximum length of the block to be written for the STM32F105xx and STM32F107xx is 256 bytes.

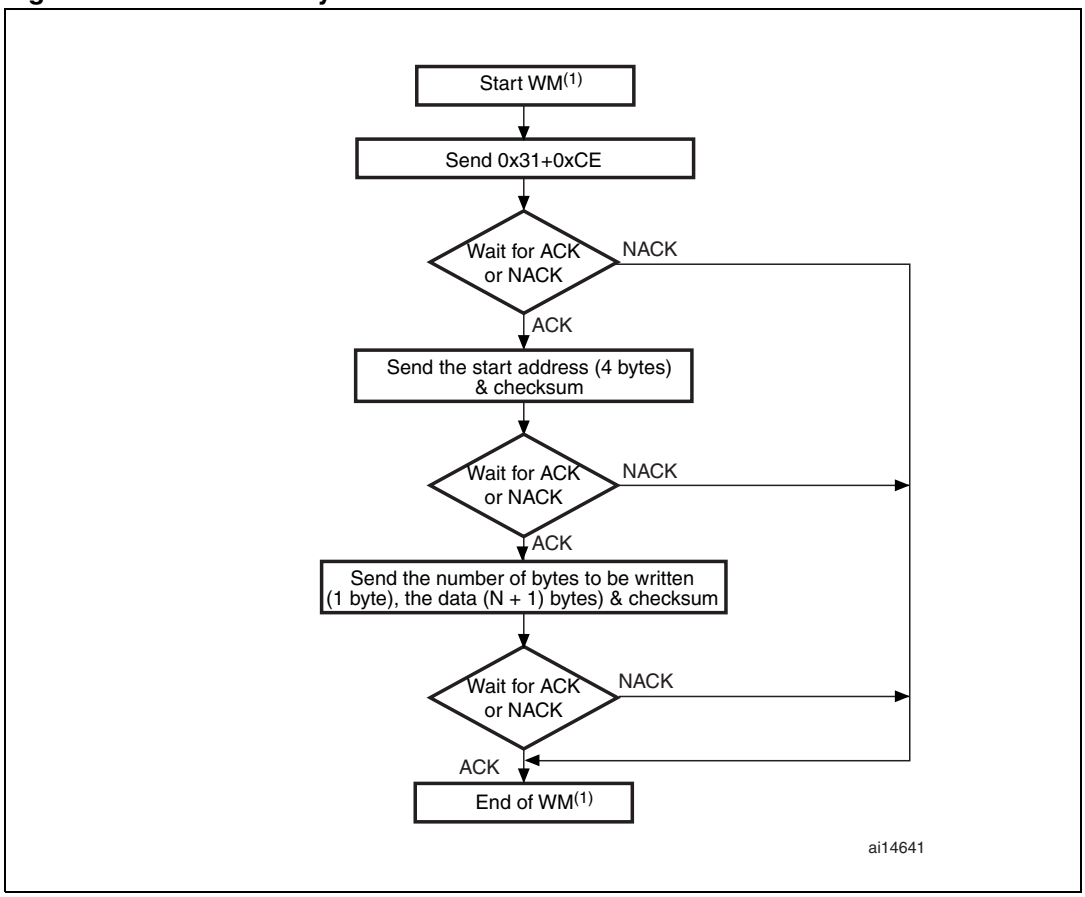
If the Write Memory command is issued to the Option byte area, all options are erased before writing the new values, and at the end of the command the bootloader generates a system Reset to take into account the new configuration of the option byte.

*Note: When writing to the RAM, care must be taken to avoid overlapping the first 4 Kbytes (0x1000) in RAM because they are used by the bootloader firmware.*

*Note: No error is returned when performing write operations on write protected sectors.*

*Write operations to FLASH/SRAM must be word aligned, if less data are written the remaining bytes should be filled by 0xFF.*

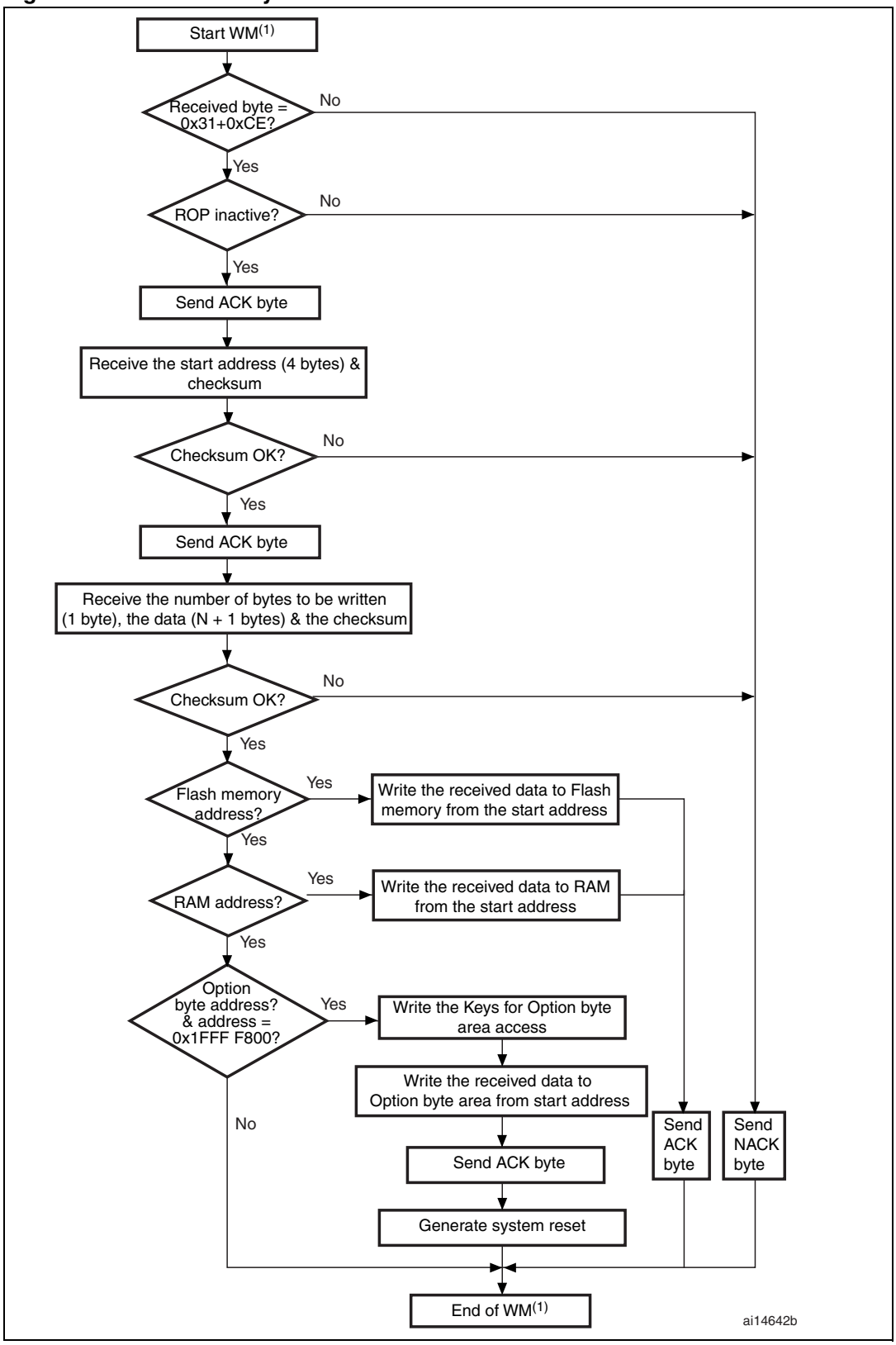
**Figure 13. Write Memory command: host side**




1. WM = Write Memory.

*Note: If the start address is invalid, the command is NACKed by the device.*

**Figure 14. Write Memory command: device side**




1. WM = Write Memory.

The host sends the bytes to the STM32F105xx and STM32F107xx as follows:

Byte 1: 0x31

Byte 2: 0xCE

Wait for ACK

Byte 3 to byte 6: start address

byte 3: MSB

byte 6: LSB

Byte 7: Checksum: XOR (Byte3, Byte4, Byte5, Byte6)

Wait for ACK

Byte 8: Number of bytes to be received ($0 < N \leq 255$)

N +1 data bytes: (Max 256 bytes)

Checksum byte: XOR (N, N+1 data bytes)

## 2.12 Erase Memory command

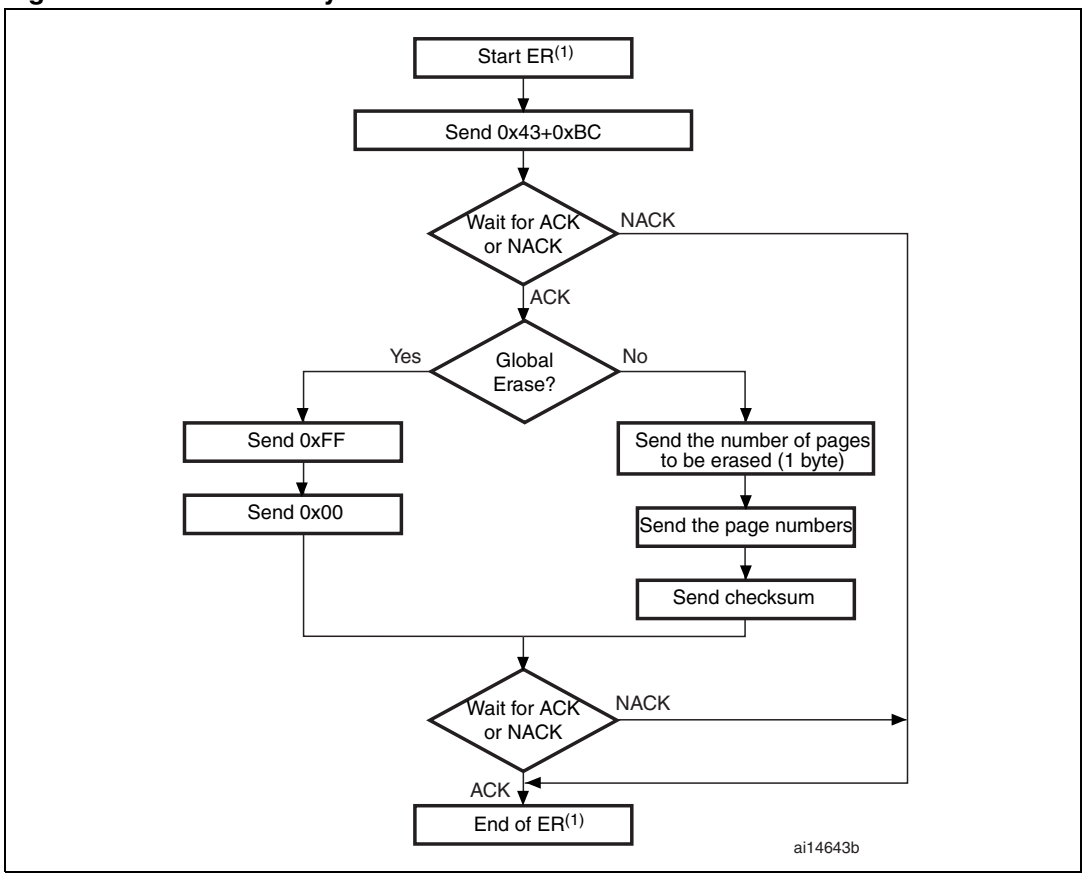
The Erase Memory command allows the host to erase Flash memory pages. When the bootloader receives the Erase Memory command, it transmits the ACK byte to the host. After the transmission of the ACK byte, the bootloader receives one byte (number of pages to be erased), the Flash memory page codes and a checksum byte; if the checksum is correct then bootloader erases the memory and sends an ACK byte to the host, otherwise it sends a NACK byte to the host and the command is aborted.

Erase Memory command specifications:

1. the bootloader receives one byte that contains N, the number of pages to be erased – 1.
   N = 255 is reserved for global erase requests. For $0 \leq N \leq 254$, N + 1 pages are erased.
2. the bootloader receives (N + 1) bytes, each byte containing a page number

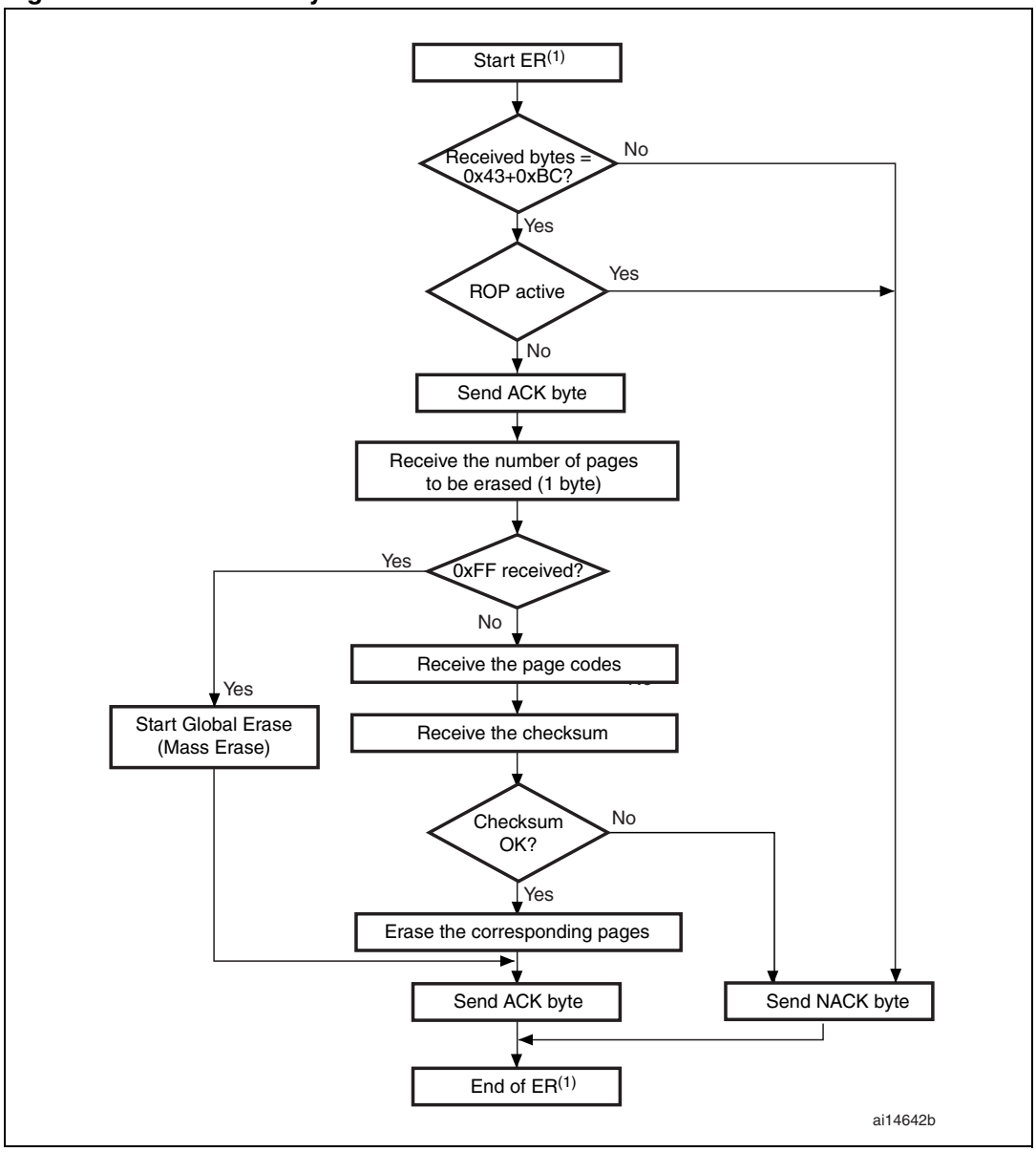
*Note: No error is returned when performing erase operations on write protected sectors.*

**Figure 15. Erase Memory command: host side**




1. ER = Erase Memory.

**Figure 16. Erase Memory command: device side**




1. ER = Erase Memory.

The host sends the bytes to the STM32F105xx and STM32F107xx as follows:

Byte 1: 0x43

Byte 2: 0xBC

Wait for ACK

Byte 3: 0xFF or number of pages to be erased – 1 ($0 \leq N \leq$ maximum number of pages)

Byte 4: 0x00 (in case of global erase) or ((N + 1 bytes (page numbers) and then checksum XOR (N, N+1 bytes))

## 2.13 Write Protect command

The Write Protect command is used to enable the write protection for some or all Flash memory sectors. When the bootloader receives the Write Protect command, it transmits the ACK byte to the host. After the transmission of the ACK byte, the bootloader waits for the number of bytes to be received (sectors to be protected) and then receives the Flash memory sector codes from the user.

At the end of the Write Protect command, the bootloader transmits the ACK byte and generates a system Reset to take into account the new configuration of the option byte.

*Note: On the STM32F105xx and STM32F107xx , the sector size is 4 Kbytes (2 pages) for the Write Protect command.*

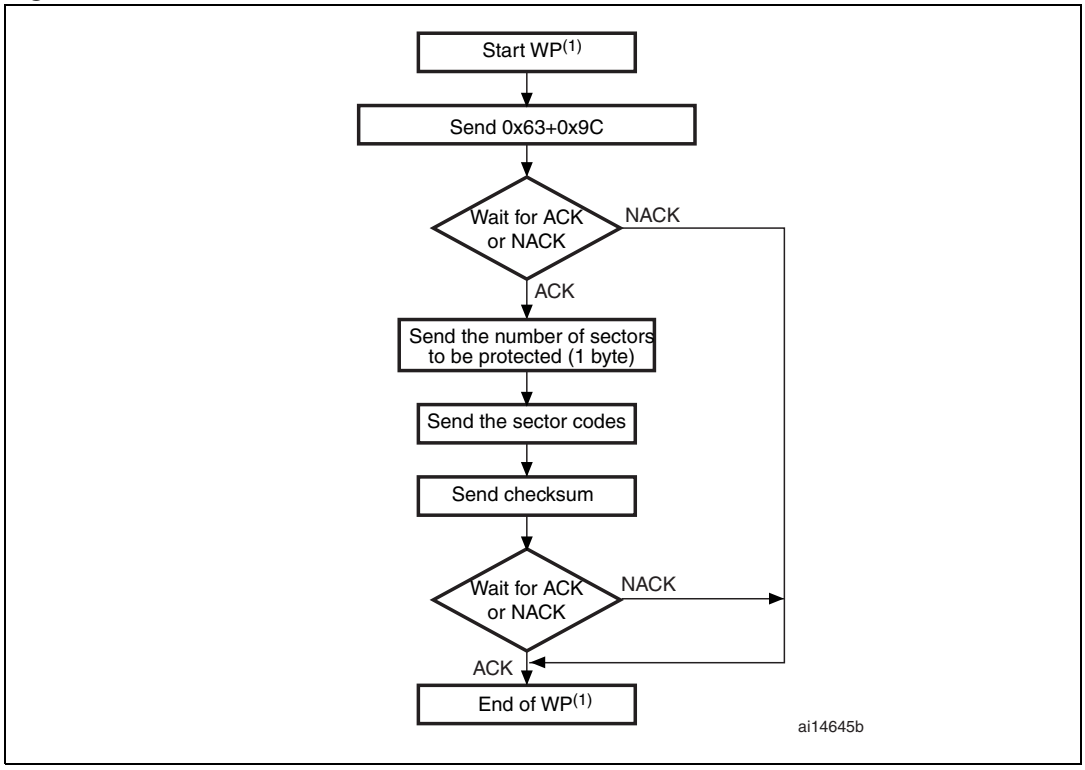
The Write Protect command sequence is as follows:

- the bootloader receives one byte that contains N, the number of sectors to be write-protected – 1 ($0 \leq N \leq 255$)
- the bootloader receives (N + 1) bytes, each byte contains a sector code

*Note: The total number of sectors and the sector number to be protected are not checked, this means that no error is returned when a command is passed with a wrong number of sectors to be protected or a wrong sector number.*

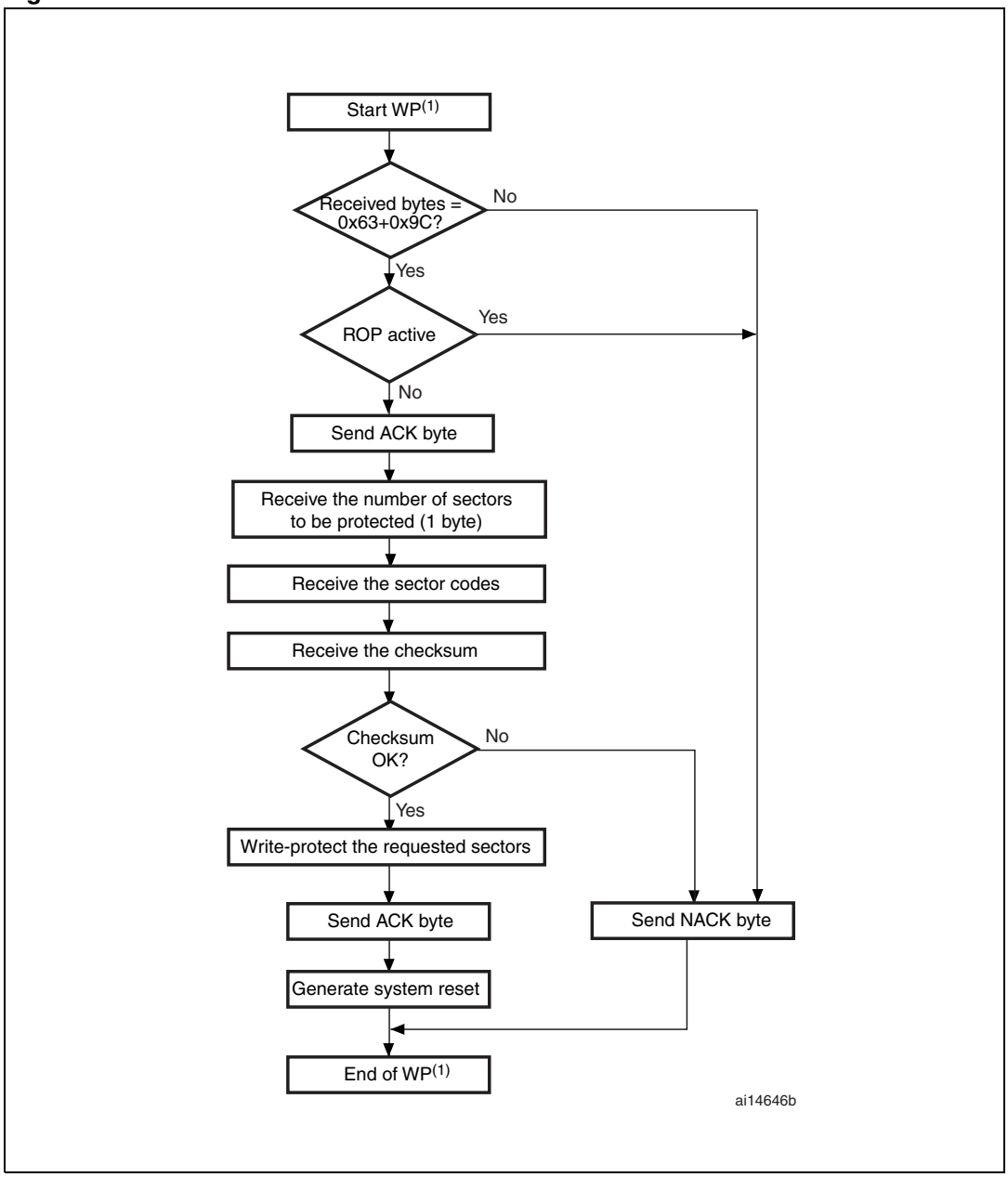
*Note: If a second Write Protect command is executed, the Flash memory sectors that had been protected by the first command become unprotected and only the sectors passed within the second Write Protect command become protected.*

**Figure 17. Write Protect command: host side**




1. WP = Write Protect.

**Figure 18. Write Protect command: device side**



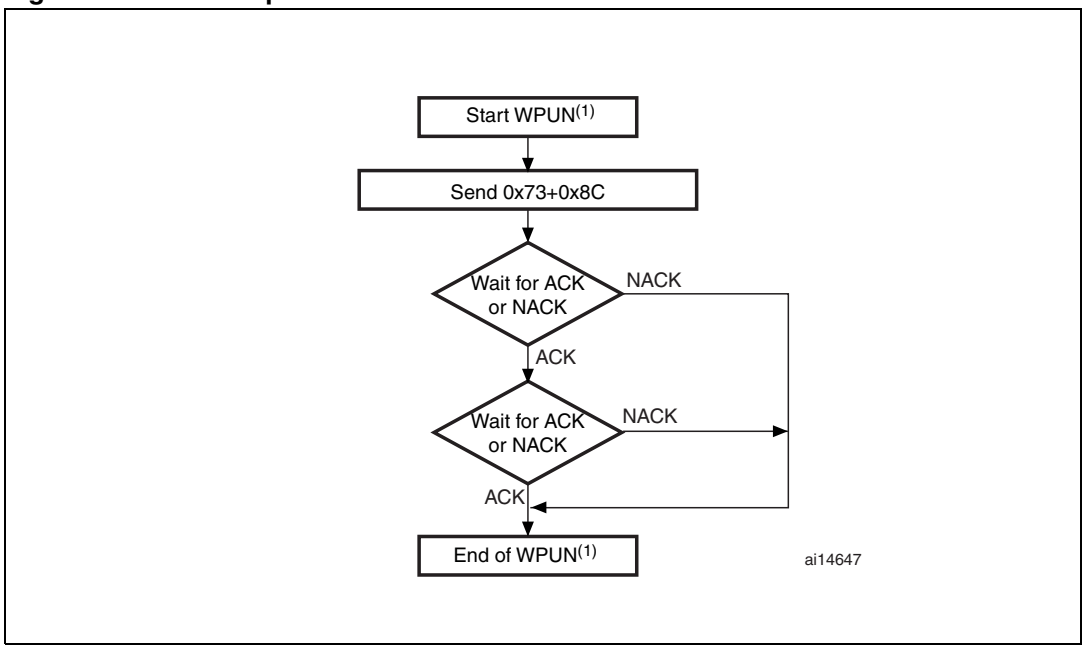

1. WP = Write Protect.

## 2.14 Write Unprotect command

The Write Unprotect command is used to disable the write protection of all the Flash memory sectors. When the bootloader receives the Write Unprotect command, it transmits the ACK byte to the host. After the transmission of the ACK byte, the bootloader disables the write protection of all the Flash memory sectors. After the unprotection operation the bootloader transmits the ACK byte.

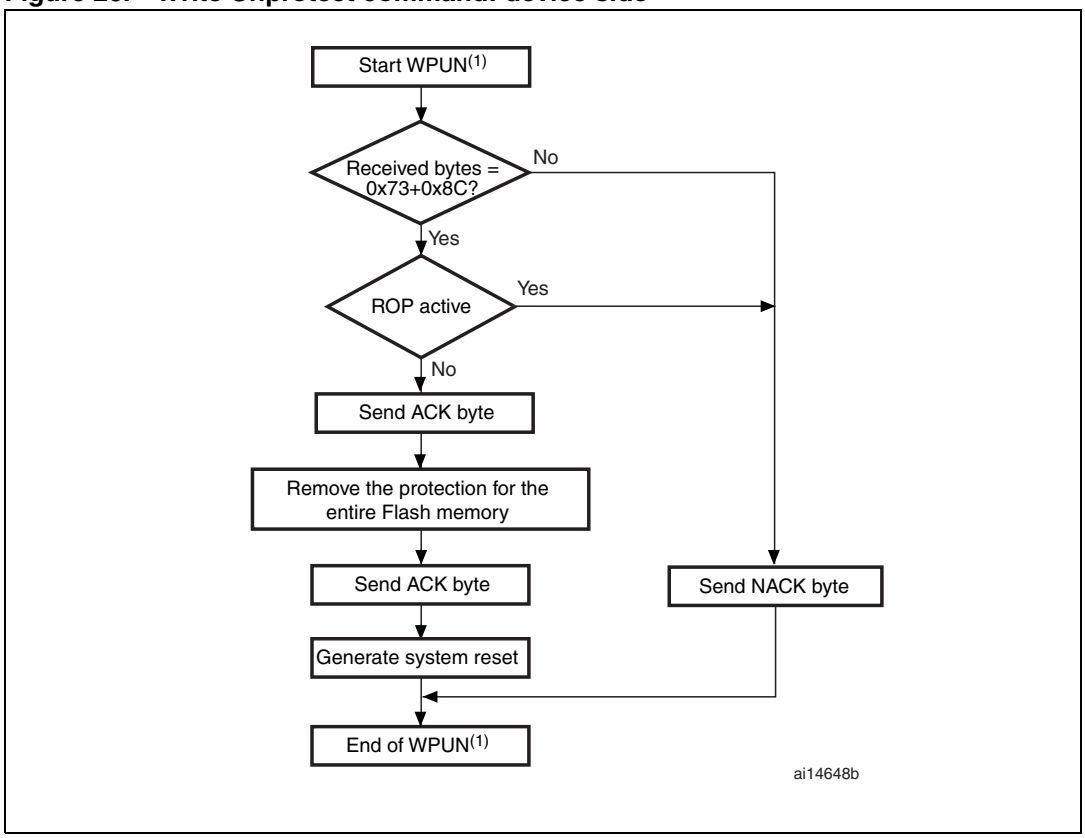
At the end of the Write Unprotect command, the bootloader transmits the ACK byte and generates a system Reset to take into account the new configuration of the option byte.

**Figure 19. Write Unprotect command: host side**




1. WPUN = Write Unprotect.

**Figure 20. Write Unprotect command: device side**



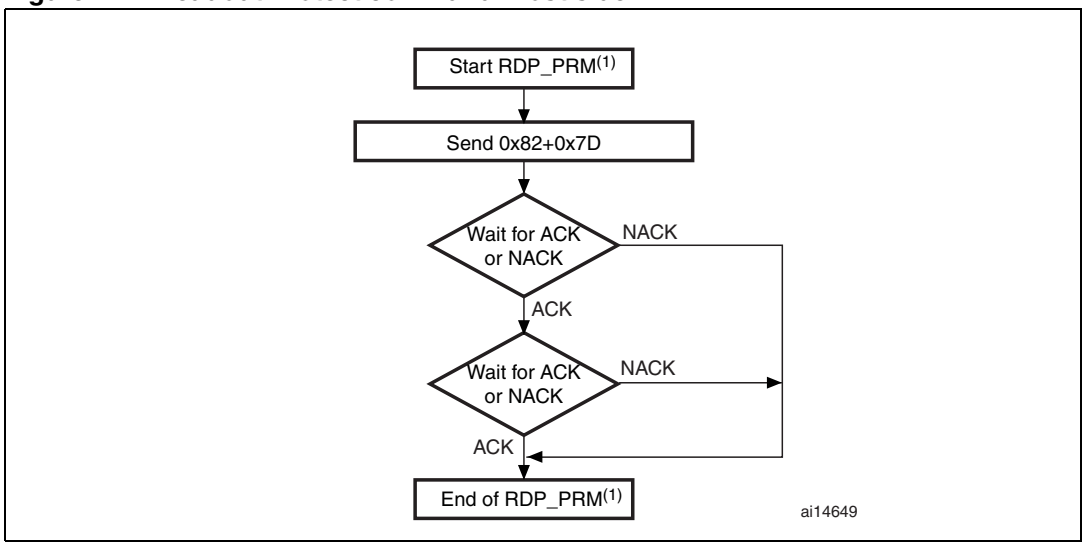

1. WPUN = Write Unprotect.

## 2.15 Readout Protect command

The Readout Protect command is used to enable the Flash memory read protection. When the bootloader receives the Readout Protect command, it transmits the ACK byte to the host. After the transmission of the ACK byte, the bootloader enables the read protection for the Flash memory.

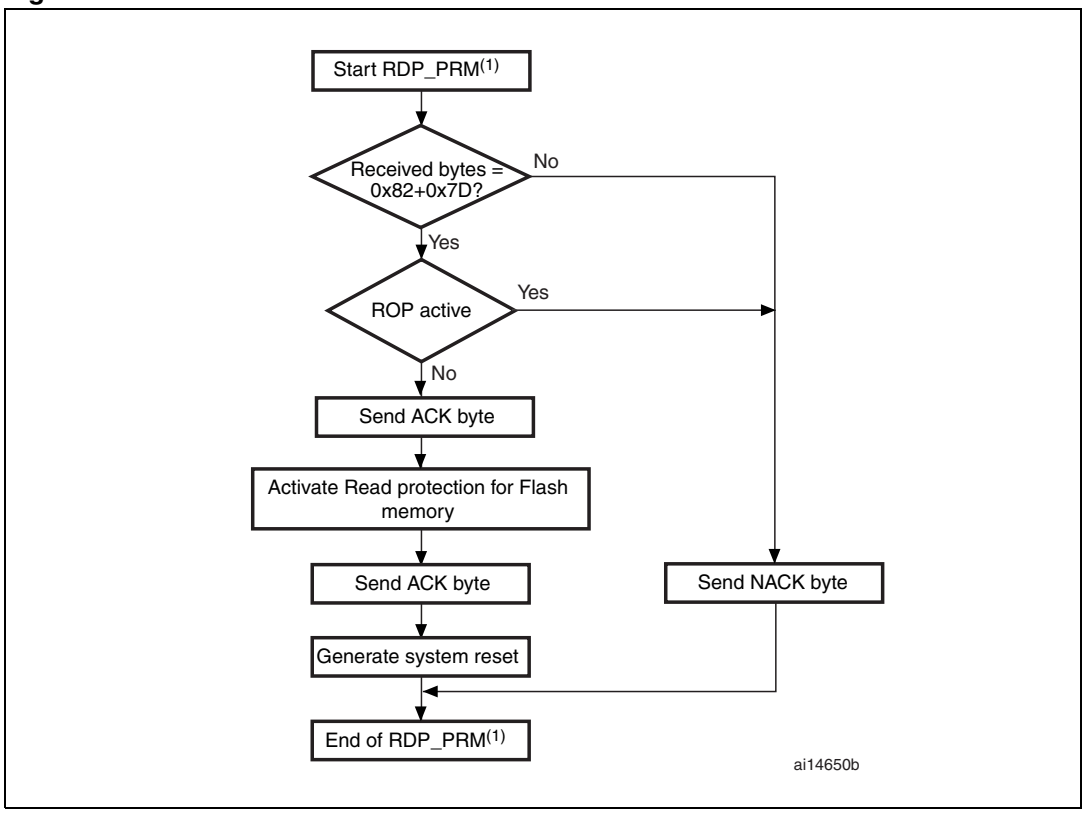
At the end of the Readout Protect command, the bootloader transmits the ACK byte and generates a system Reset to take into account the new configuration of the option byte.

**Figure 21. Readout Protect command: host side**




1. RDP_PRM = Readout Protect.

**Figure 22. Readout Protect command: device side**




1. RDP_PRM = Readout Protect.

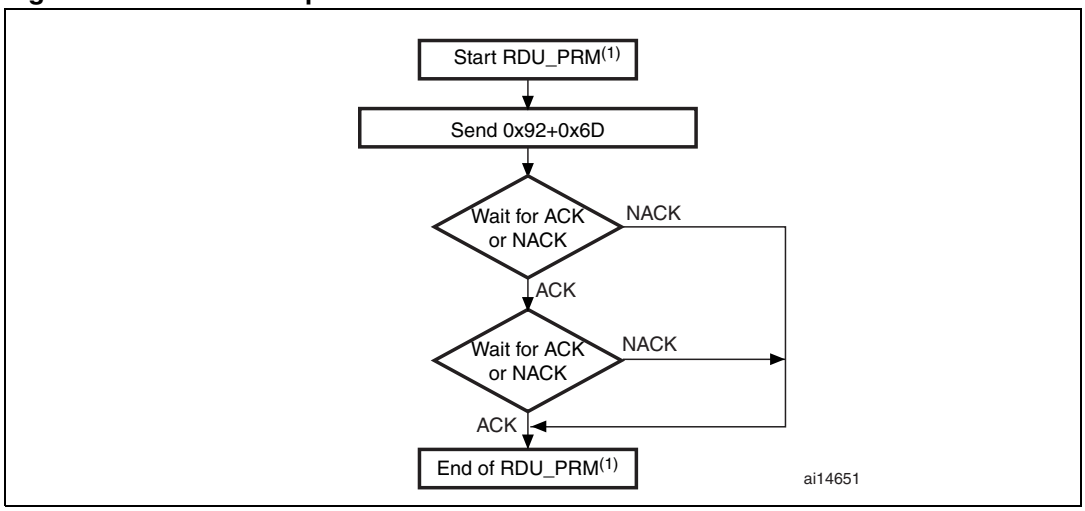
## 2.16 Readout Unprotect command

The Readout Unprotect command is used to disable the Flash memory read protection. When the bootloader receives the Readout Unprotect command, it transmits the ACK byte to the host. After the transmission of the ACK byte, the bootloader erases all the Flash memory sectors and it disables the read protection for the entire Flash memory. If the erase operation is successful, the bootloader deactivates the RDP.

If the erase operation is unsuccessful, the bootloader transmits a NACK and the read protection remains active.

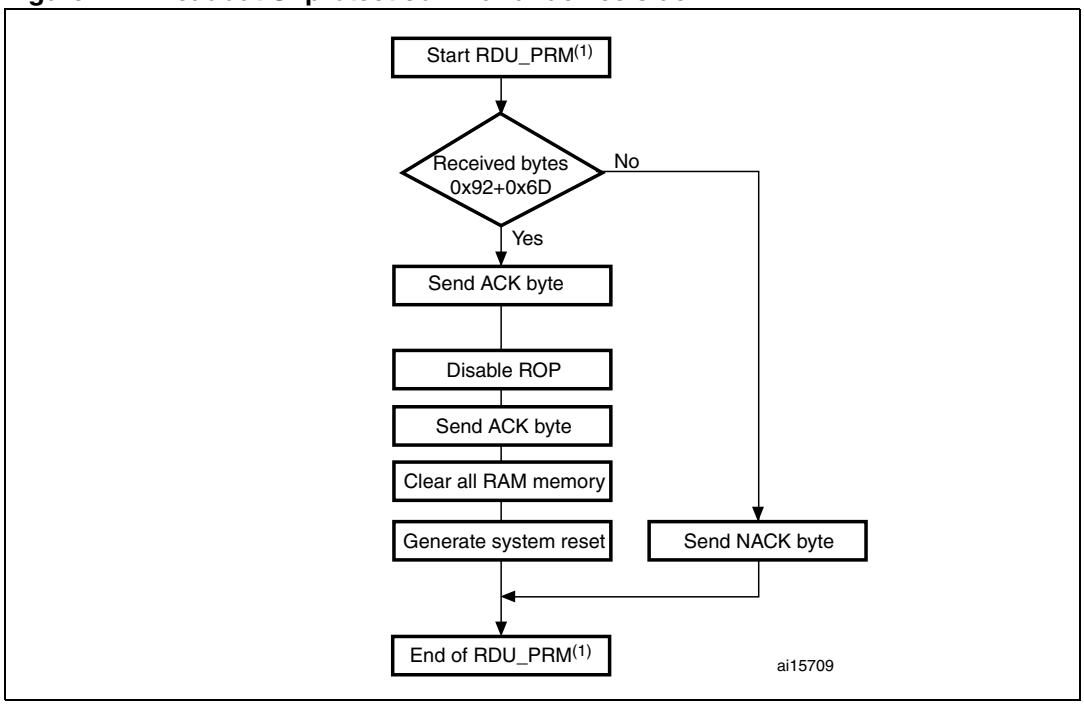
At the end of the Readout Unprotect command, the bootloader transmits an ACK and generates a system Reset to take into account the new configuration of the option byte.

**Figure 23. Readout Unprotect command: host side**




1. RDU_PRM = Readout Unprotect.

**Figure 24. Readout Unprotect command: device side**

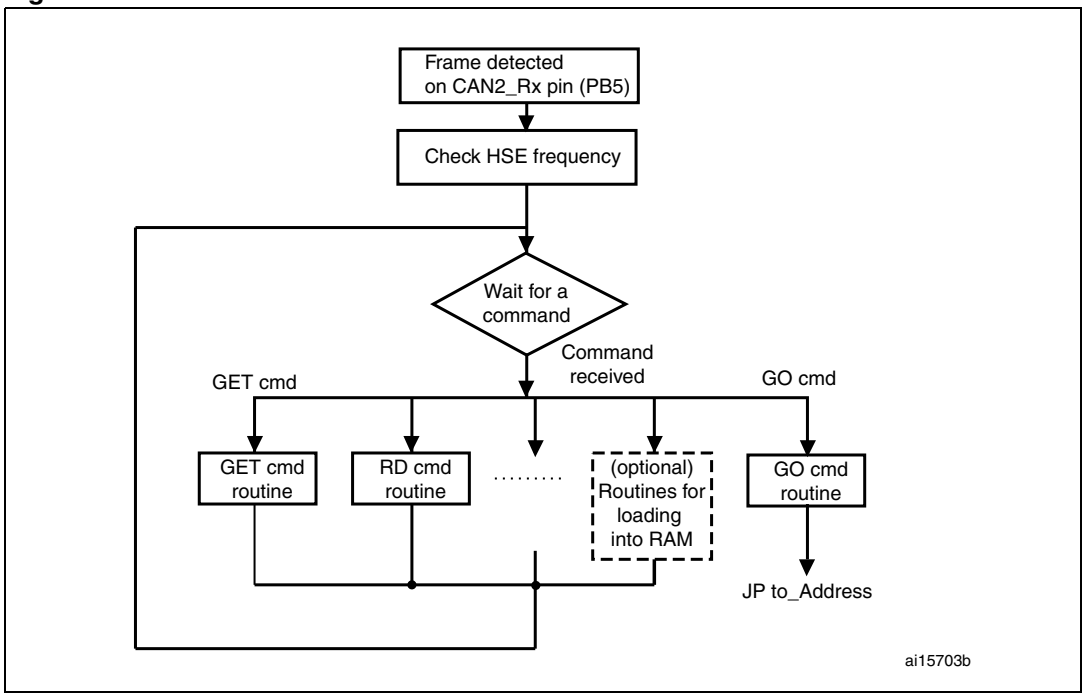



1. RDU_PRM = Readout Unprotect.

# 3 CAN bootloader

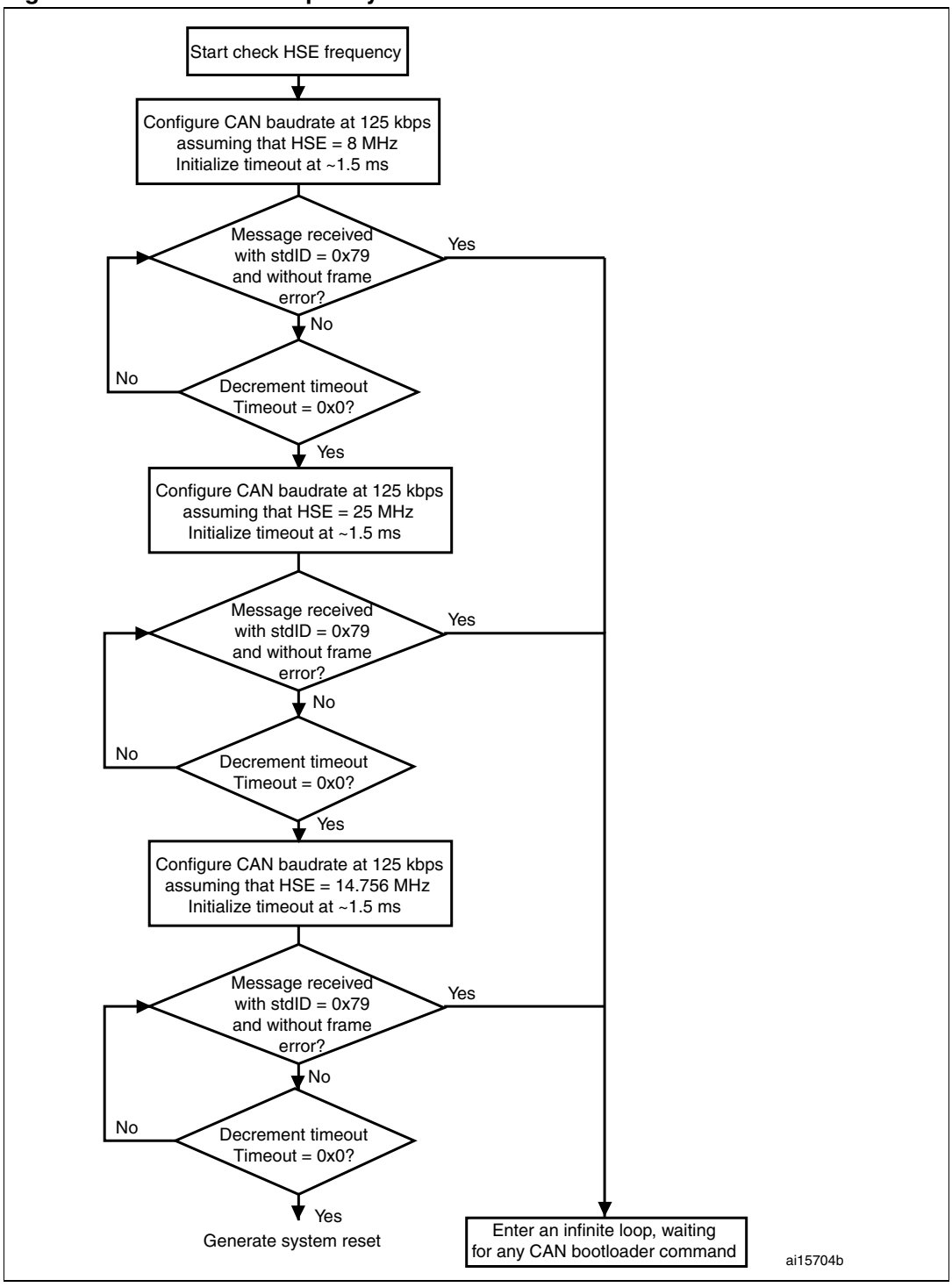
## 3.1 Bootloader code sequence

**Figure 25. Bootloader for STM32F105xx and STM32F107xx withCAN2**




Once System memory boot mode is entered and the microcontroller has been configured as described above, the bootloader code waits for a frame on the CAN2_Rx pin. When a detection occurs the CAN Bootloader firmware starts to check the external clock frequency, *Figure 26* shows the flowchart of the frequency check.

**Figure 26. Check HSE frequency**



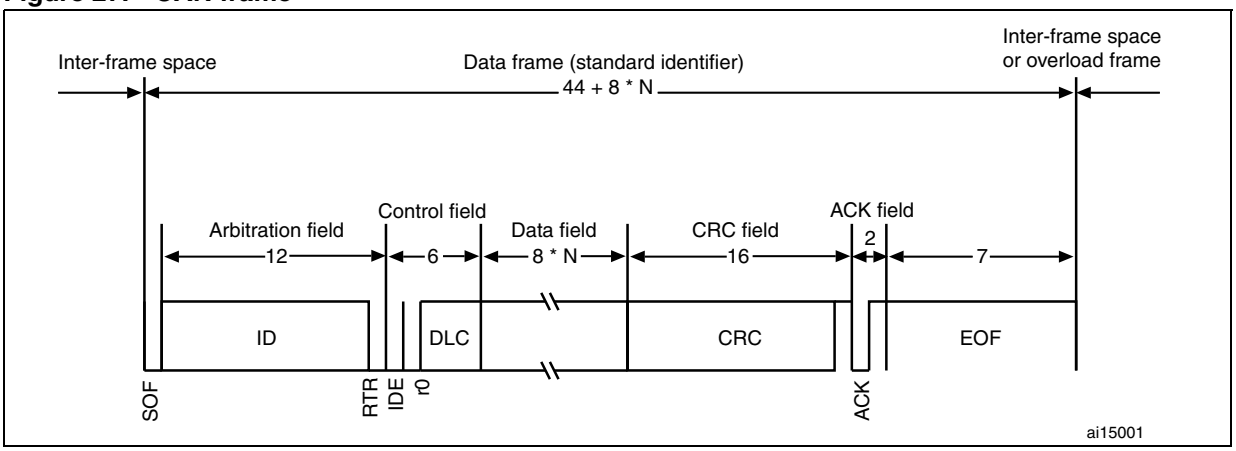

Next, the code initializes the serial interface accordingly. Using this calculated baud rate, an acknowledge byte *(0x79)* is returned to the host, which signals that the STM32F105xx and STM32F107xx is ready to receive user commands.

## 3.2 CAN settings

The STM32F105xx and STM32F107xx CAN2 is compliant with the 2.0A and B (active) specifications with a bitrate up to 1Mbit/s. It can receive and transmit standard frames with 11-bit identifiers as well as extended frames with 29-bit identifiers.

*Figure 27* shows a CAN frame that uses the standard identifier only.

**Figure 27. CAN frame**




In this application, the CAN 2 settings are

- Standard identifier (not extended)
- Bit rate: At the beginning it is 125 kbps; during run time it can be changed via the speed command to achieve a maximum bit rate of 1 Mbps.

The transmit settings (from STM32F105xx and STM32F107xx to the host) are:

- Tx mailbox0: On
- Tx mailbox1 and Tx mailbox2: Off
- Tx identifier: (0x00, 0x01, 0x02, v03, 0x11, 0x21, 0x31, 0x43, 0x63, 0x73, 0x82, 0x92).

The receive settings (from the host to STM32F105xx and STM32F107xx ) are:

- Synchronization byte, 0x79, is in the RX identifier and not in the data field.
- RX identifier depends on the command (0x00, 0x01, 0x02, 0x03, 0x11, 0x21, 0x31, 0x43, 0x63, 0x73, 0x82, 0x92).
- Error checking: If the error field (bit [6:4] in the CAN_ESR register) is different from 000b, the message is discarded and a NACK is sent to the host.
- In FIFO overrun condition, the message is discarded and a NACK is sent to the host.
- Incoming messages can contain from 1 to 8 data bytes.

*The CAN2 peripheral is accessible via pins PB6 (TX) and PB5 (RX)*

Note: CAN1 is clocked during CAN bootloader execution because in STM32F105xx and STM32F107xx CAN1 manages the communication between CAN2 and SRAM.

Note: The CAN Bootloader firmware supports only one node at the same time. This means that CAN Network Management is not supported by the firmware.

## 3.3 Bootloader command set

The supported commands are listed in *Table 3* below. Each command is further described in this section.

**Table 4. CAN bootloader commands**

| Command | Command code | Command description |
|---|---|---|
| Get(1) | 0x00 | Gets the version and the allowed commands supported by the current version of the bootloader |
| Get Version & Read Protection Status(1) | 0x01 | Gets the bootloader version and the Read Protection status of the Flash memory |
| Get ID(1) | 0x02 | Gets the chip ID |
| Speed | 0x03 | The speed command allows the baud rate for CAN run-time to be changed. |
| Read Memory | 0x11 | Reads up to 256 bytes of memory starting from an address specified by the user |
| Go | 0x21 | Jumps to user application code located in the internal Flash memory or in SRAM |
| Write Memory | 0x31 | Writes up to 256 bytes to the RAM or Flash memory starting from an address specified by the user |
| Erase | 0x43 | Erases from one to all the Flash memory sectors |
| Write Protect(2) | 0x63 | Enables the write protection for some sectors |
| Write Unprotect(2) | 0x73 | Disables the write protection for all Flash memory sectors |
| Readout Protect(1) | 0x82 | Enables the read protection |
| Readout Unprotect(1) | 0x92 | Disables the read protection |

1. Read protection – When the RDP (read protection) option is active, only this limited subset of commands is available. All other commands are NACKed and have no effect on the device. Once the RDP has been removed, the other commands become active.
2. On the STM32F105xx and STM32F107xx , the sector size is 4 Kbytes (2 pages) for the Write Protect, Write Unprotect and Erase commands.

### Communication safety

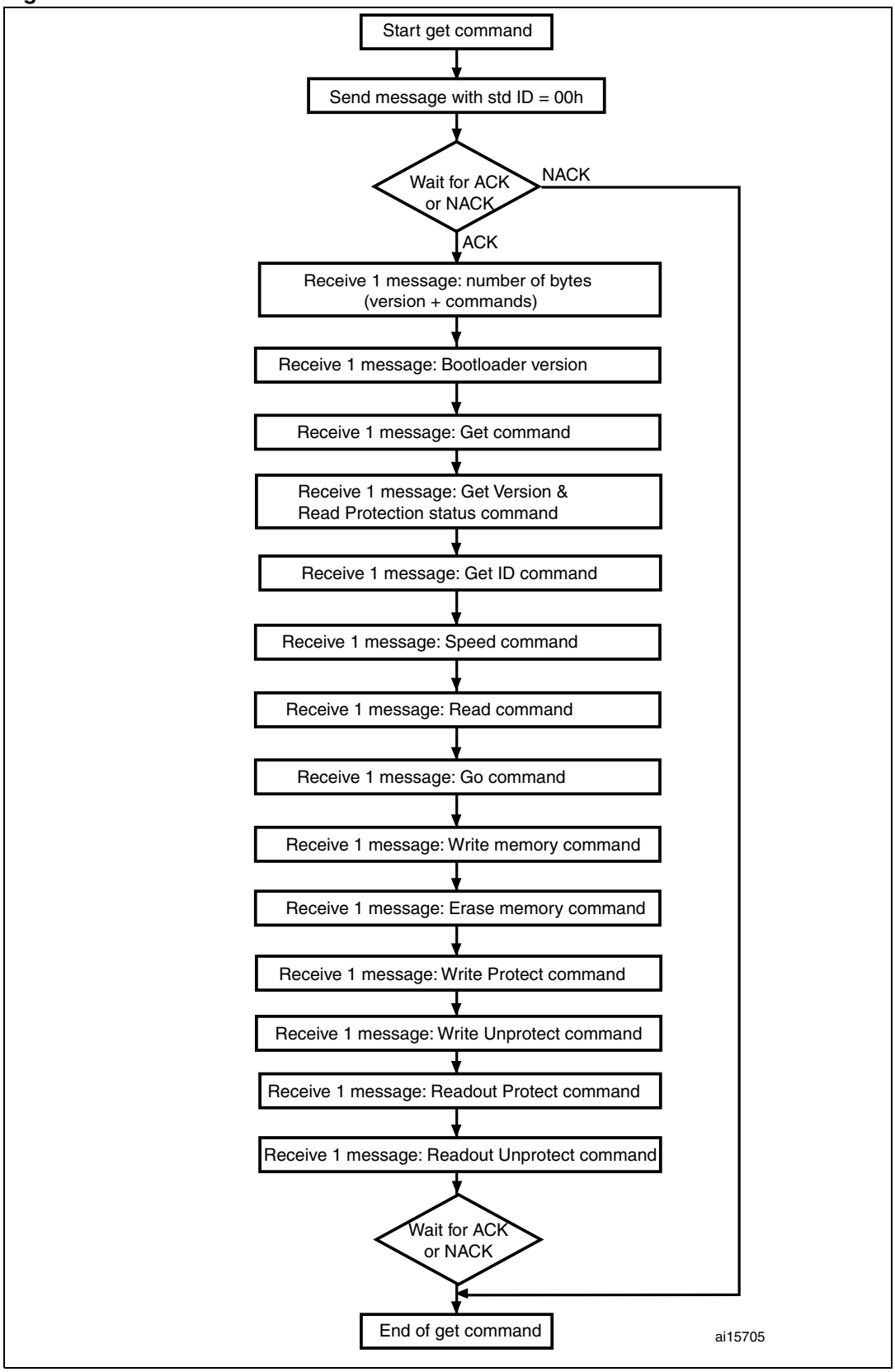
Each packet is either accepted (ACK answer) or discarded (NACK answer):

- ACK message = 0x79
- NACK message = 0x1F

## 3.4 Get command

The get command allows the host to get the version of the bootloader and the supported commands. When the bootloader receives the get command, it transmits the bootloader version and the supported command codes to the host.

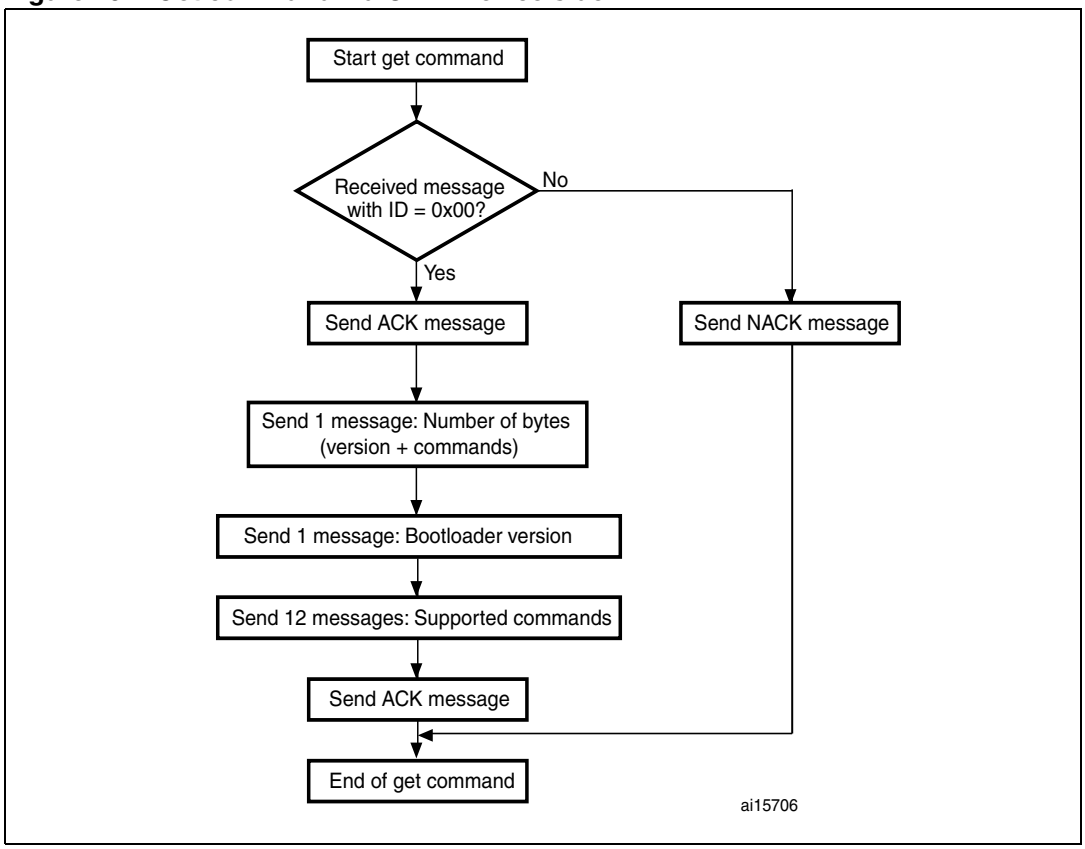
**Figure 28. Get command via CAN: Host side**



### The host sends the messages as follows:

Command message: Std ID = 0x00, data length code (DLC) = ‘not important’.

**Figure 29. Get command via CAN: Device side**




### The STM32F105xx and STM32F107xx send the messages as follows:

Message 1: Std ID = 0x00, DLC = 1, data = 0x79 - ACK

Message 2: Std ID = 0x00, DLC = 1 data = N = 12 = the number of bytes to be sent -1 (1 <= N +1 <= 256)

Message 3: Std ID = 0x00, DLC = 1, data = bootloader version (0 < version <= 255)

Message 4: Std ID = 0x00, DLC = 1, data = 0x00 - Get command

Message 5: Std ID = 0x00, DLC = 1, data = 0x01 - Get Version & Read Protection Status command

Message 6: Std ID = 0x00, DLC = 1, data = 0x02 - Get ID command

Message 7: Std ID = 0x00, DLC = 1, data = 0x03 - Speed command

Message 8: Std ID = 0x00, DLC = 1, data = 0x11 - Read memory command

Message 9: Std ID = 0x00, DLC = 1, data = 0x21 - Go command

Message 10: Std ID = 0x00, DLC = 1, data = 0x31 - Write memory command

Message 11: Std ID = 0x00, DLC = 1, data = 0x43 - Erase memory command

Message 12: Std ID = 0x00, DLC = 1, data = 0x63 - Write Protect command

Message 13: Std ID = 0x00, DLC = 1, data = 0x73 - Write Unprotect command

Message 14: Std ID = 0x00, DLC = 1, data = 82h - Readout Protect command

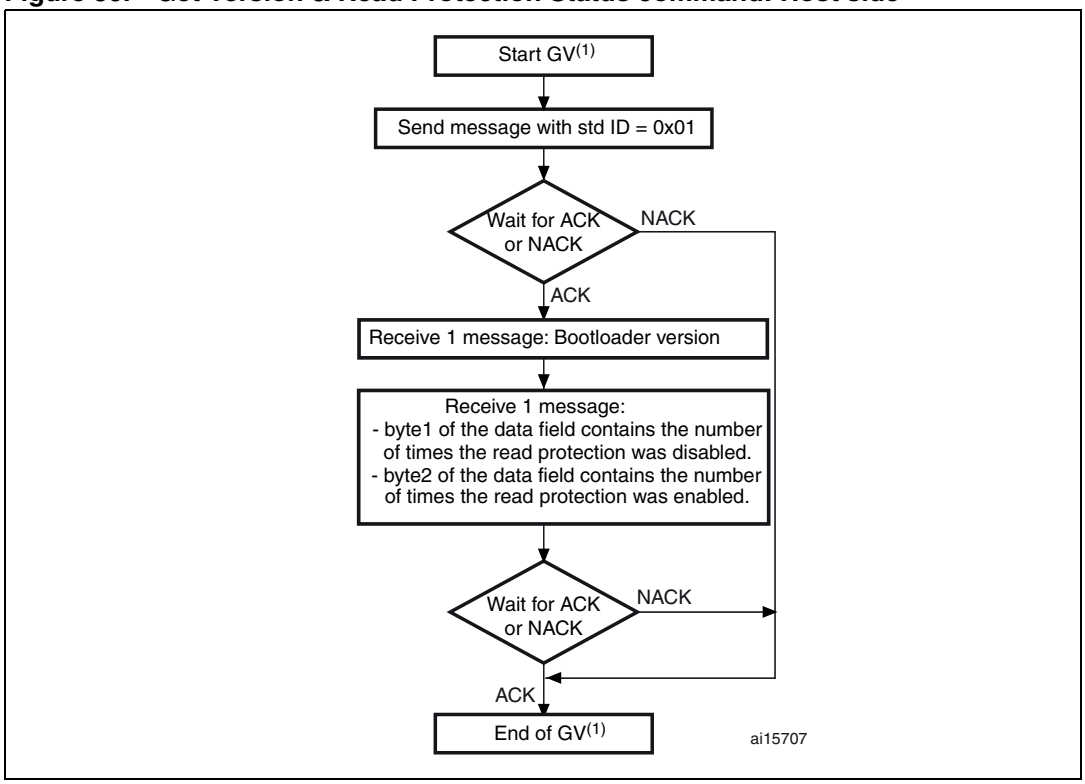
Message 15: Std ID = 0x00, DLC = 1, data = 92h - Readout Unprotect command

Message 1: Std ID = 0x00, DLC = 1, data = 0x79 - ACK

## 3.5 Get Version & Read Protection Status command

The Get Version & Read Protection Status command is used to get the bootloader version and the read protection status. When the bootloader receives the command, it transmits the information described below (version, read protection: number of times it was enabled and disabled) to the host.

**Figure 30. Get Version & Read Protection Status command: Host side**



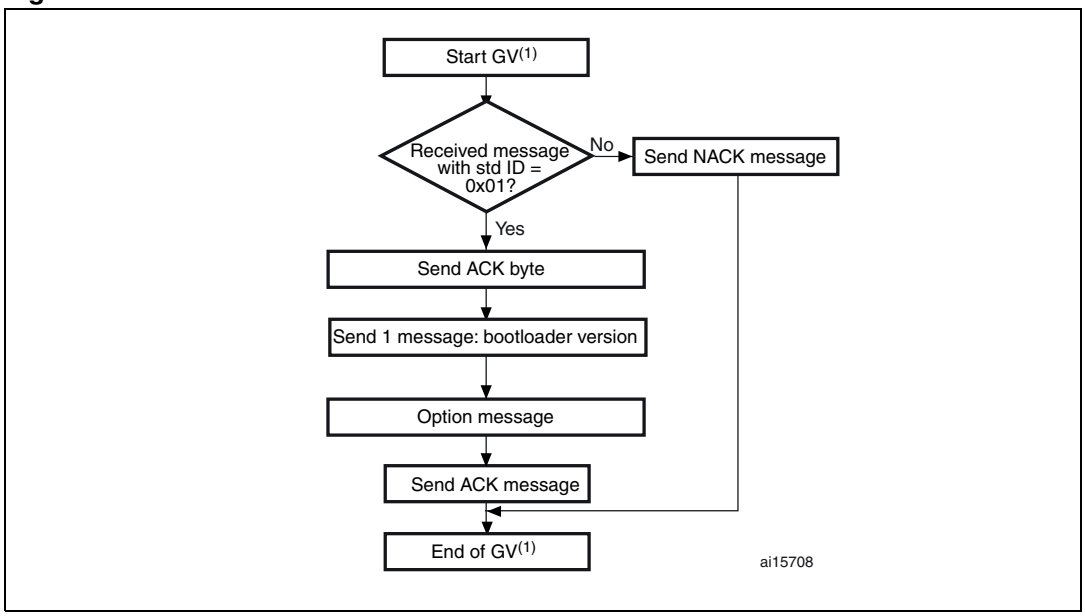

1. GV = Get Version & Read Protection Status.

### The host sends the messages as follows:

Command message: Std ID = 0x01, data length code (DLC) = ‘not important’.

ACK Message contain: Std ID = 0x01, DLC = 1, data = 0x79 - ACK

**Figure 31. Get Version & Read Protection Status command: device side**




1. GV = Get Version & Read Protection Status.

### The STM32F105xx and STM32F107xx send the messages as follows:

Message 1: Std ID = 0x01, DLC = 1, data = ACK

Message 2: Std ID = 0x01, DLC = 1, data[0] = bootloader version (0 < version <= 255), example: 0x10 = Version 1.0

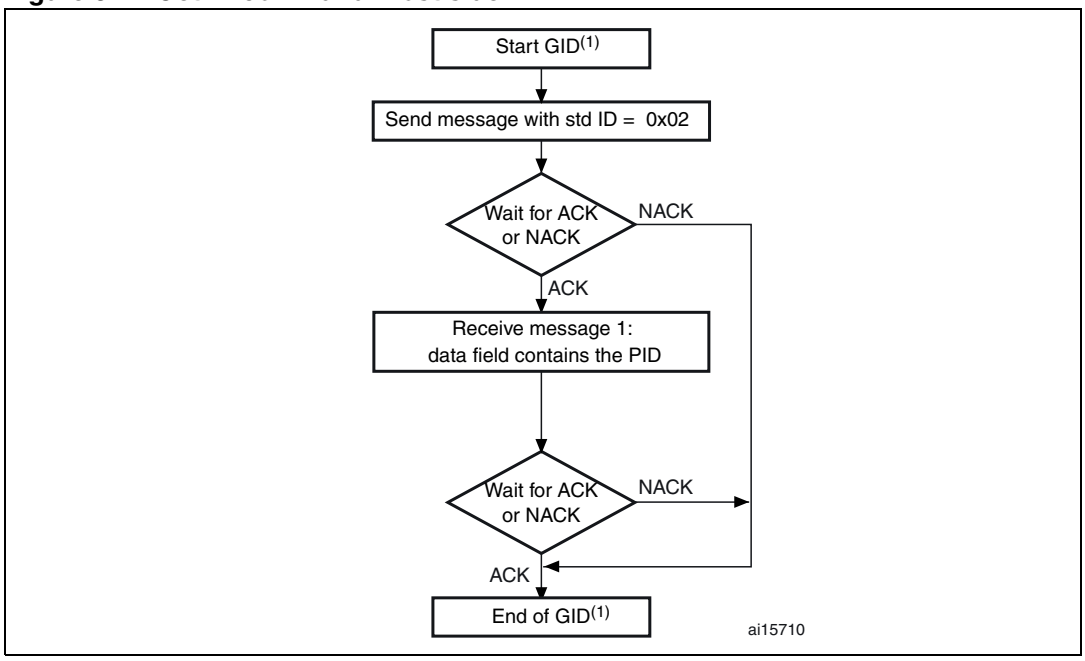
Message 3: Option message 1: Std ID = 0x01, DLC = 2, data = 0x00(byte1 and byte2)

Message 4: Std ID = 0x01, DLC = 1, data = ACK

## 3.6 Get ID command

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

**Figure 32. Get ID command: host side**



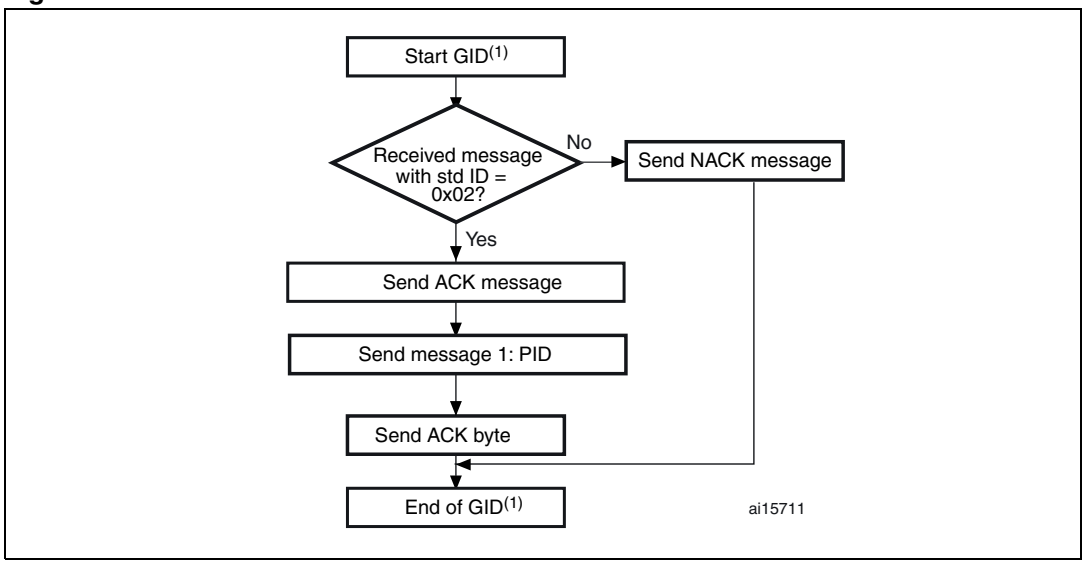

1. GID = Get ID.
2. PID stands for product ID. It is 0x0418 for STM32F105xx and STM32F107xx products. Byte 1 is the MSB and byte 2 is LSB of the address.

### The host sends the messages as follows:

Command message: Std ID = 0x02, data length code (DLC) = ‘not important’.

ACK Message contain: Std ID = 0x02, DLC = 1, data = 0x79 - ACK

**Figure 33. Get ID command: device side**




1. GID = Get ID.
2. PID stands for product ID. It is 0x0418 for STM32F105xx and STM32F107xx products. Byte 1 is the MSB and byte 2 is LSB of the address.

The STM32F105xx and STM32F107xx sends the bytes as follows:

Message 1: Std ID = 0x02, DLC = 1, data = ACK with DLC except for current message and ACKs.

Message 2: Std ID = 0x02, DLC = N (the number of bytes – 1. For STM32F105xx and STM32F107xx , N = 1), data = PID with byte 0 is MSB and byte N is the LSB of the product ID

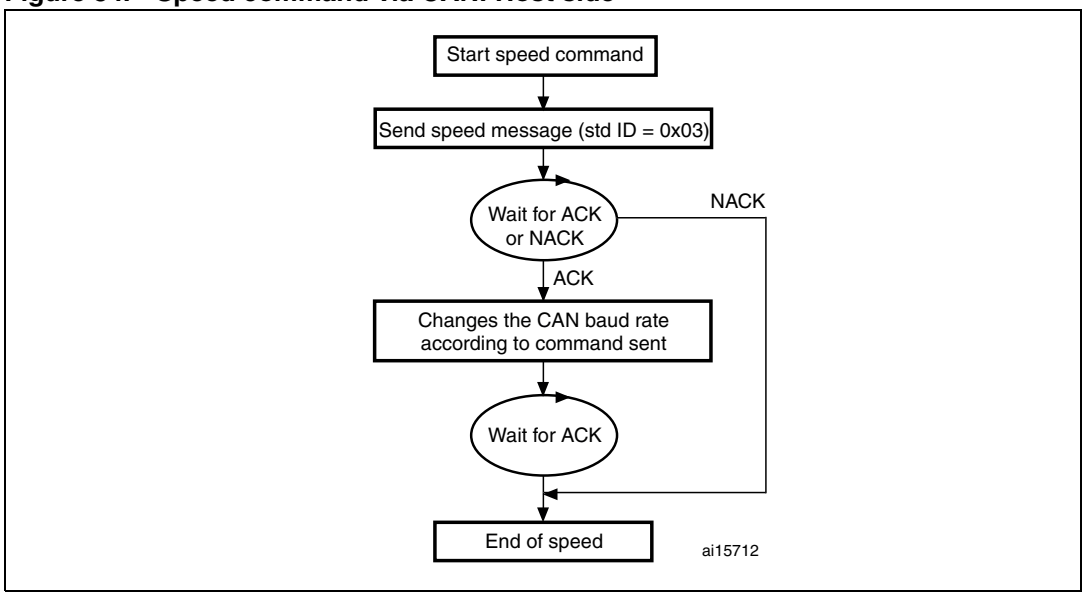
Message 3: Std ID = 0x02, DLC = 1, data = ACK = 0x79

## 3.7 Speed command

The speed command allows the baud rate for CAN run-time to be changed. It can be used only if CAN is the peripheral being used.

A system reset is generated if CAN2 receives the correct message but the operation to set the new baudrate fails, which prevents it from entering or leaving initialization mode.

**Figure 34. Speed command via CAN: Host side**



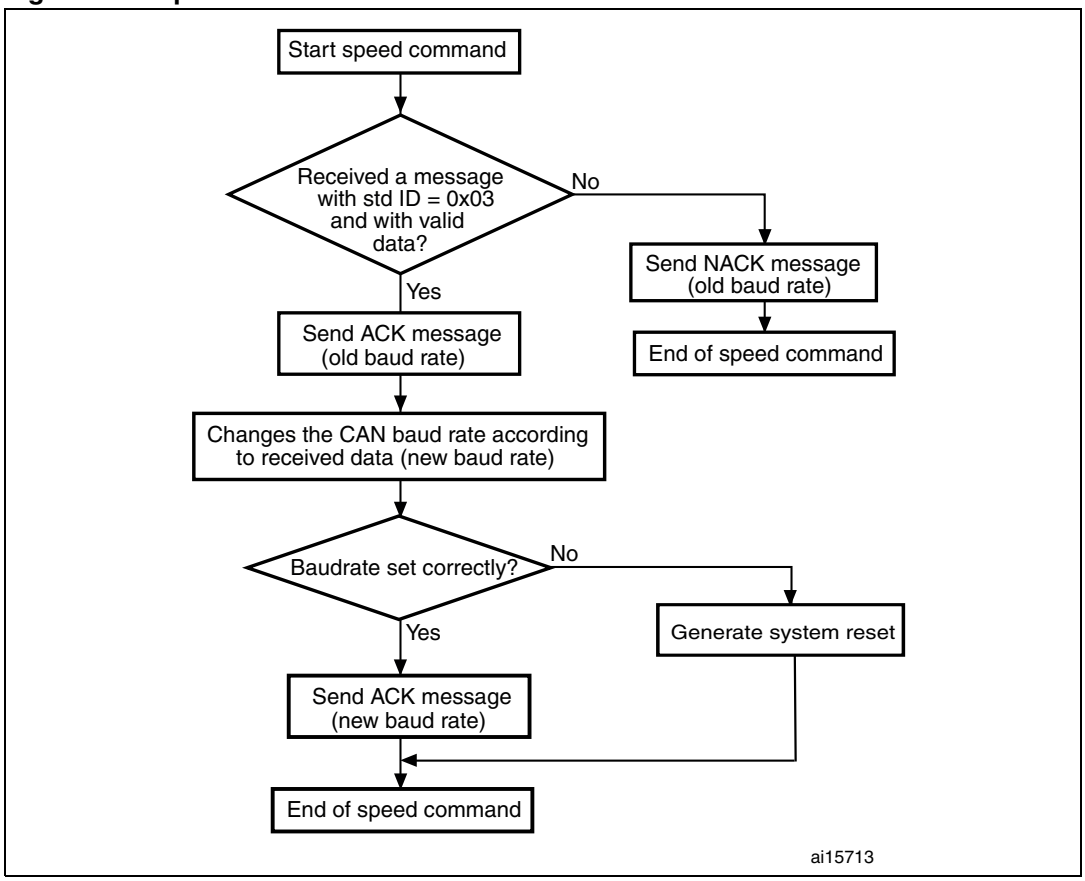

1. After setting the new baud rate, the bootloader sends the ACK message. Therefore, the host sets its baud rate while waiting for the ACK.

### The host sends the message as follows:

Command message: Std ID = 0x03, DLC = 0x01, data[0] = XXh where XXh takes the following values depending on the baud rate to be set:

- 0x01 -> baud rate = 125 kbps
- 0x02 -> baud rate = 250 kbps
- 0x03 -> baud rate = 500 kbps
- 0x04 -> baud rate = 1 Mbps

**Figure 35. Speed command via CAN: Device side**




The STM32F105xx and STM32F107xx sends the bytes as follows:

Message 1: Std ID = 0x03, DLC = 1, data[0] = ACK= 0x79: with old baudrate if the receive message is correct else data[0] = NACK= 0x1F

Message 2: Std ID = 0x02, DLC = 1, data[0] = ACK = 0x79 with new baudrate

## 3.8 Read Memory command

The Read Memory command is used to read data from any memory address in RAM (starting from address 0x20001000), Flash memory and information block (System memory or option byte areas)

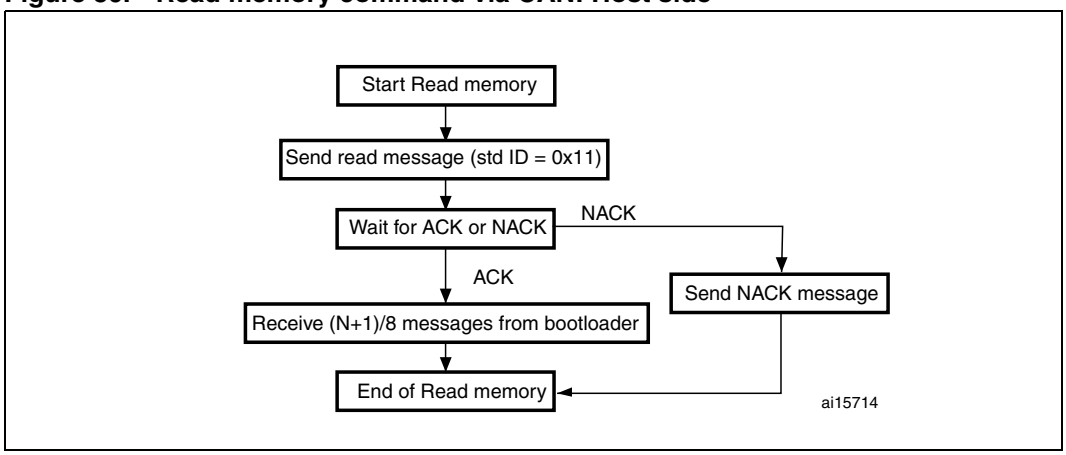
When the bootloader receives the Read Memory command, it starts to verify the contents of the message:

- ID of the command is correct or not
- ReadOutProtection is disabled or enabled
- Address to be read is valid or not

If the message content is correct it transmits an ACK message otherwise it transmits a NACK message.

After sending an ACK message, then it transmits the required data to the user ((N + 1) bytes) via (N+1) messages /8 (since each message contains 8 bytes), starting from the received address.

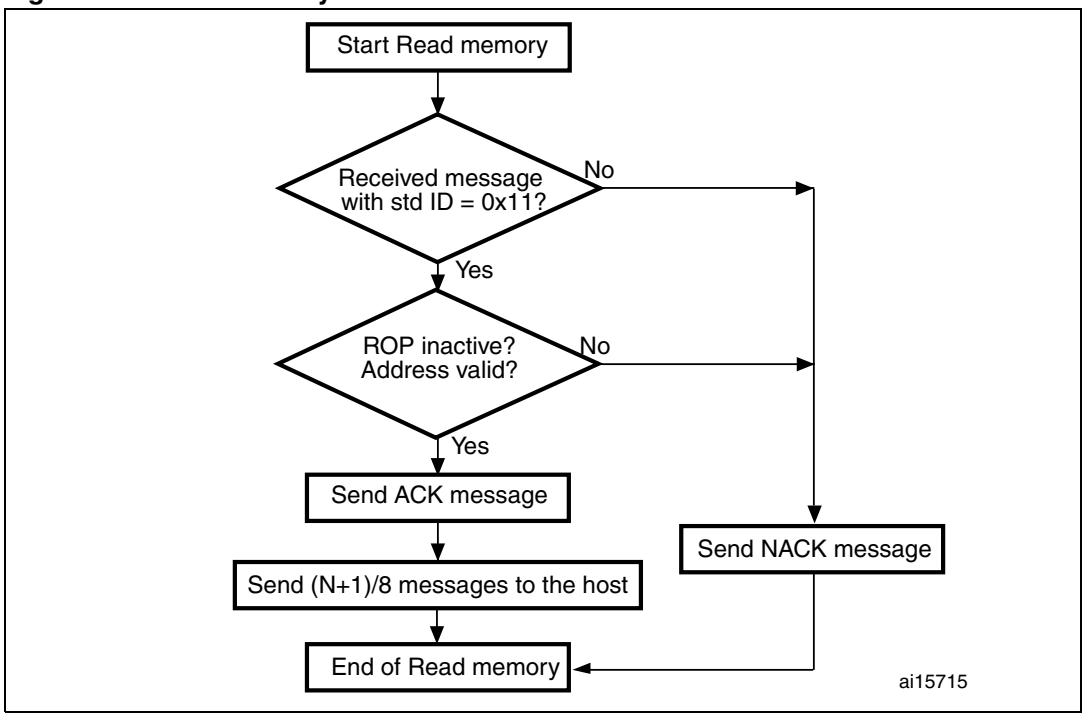
**Figure 36. Read memory command via CAN: Host side**




### The host sends the messages as follows:

Command message: Std ID = 0x11, DLC = 0x05, data[0] = 0xXX: MSB of the address... data[3] = 0xYY: LSB of the address, data[4] = N: number of bytes to be read (where 0 < N <= 255).

**Figure 37. Read memory command via CAN: Device side**




**The STM32F105xx and STM32F107xx send the messages as follows:**

ACK message: Std ID = 0x11, DLC = 1, data[0] = ACK if content of the command is correct else data[0] = NACK

Data message (N+1) / 8: Std ID = 0x11, DLC = Number of Byte, data[0] = 0xXX... data[Number of Byte - 1] = 0xYY

ACK message: Std ID = 0x11, DLC = 1, data[0] = ACK

## 3.9 Go command via CAN

The Go command is used to execute the downloaded code or any other code by branching to an address specified by the user. When the bootloader receives the Go command, it starts if the message contains the following valid information:

- ID of the command is correct or not
- ReadOutProtection is disabled or enabled
- Branch destination address is valid or not(data[0] is the address MSB and data[3] 4 is LSB

If the message content is correct it transmits an ACK message otherwise it transmits a NACK message.

After sending an ACK message to the user, the bootloader firmware performs the following:

- it initializes the registers of the peripherals used by the bootloader (see *Table 2*) to their default reset values
- it initializes the user application's main stack pointer
- it jumps to the memory location programmed in the received 'address + 4' (which corresponds to the address of the application's reset handler).
  For example if the received address is 0x0800 0000, the bootloader will jump to the memory location programmed at address 0x0800 0004.
  In general, the host should send the base address where the application to jump to is programmed

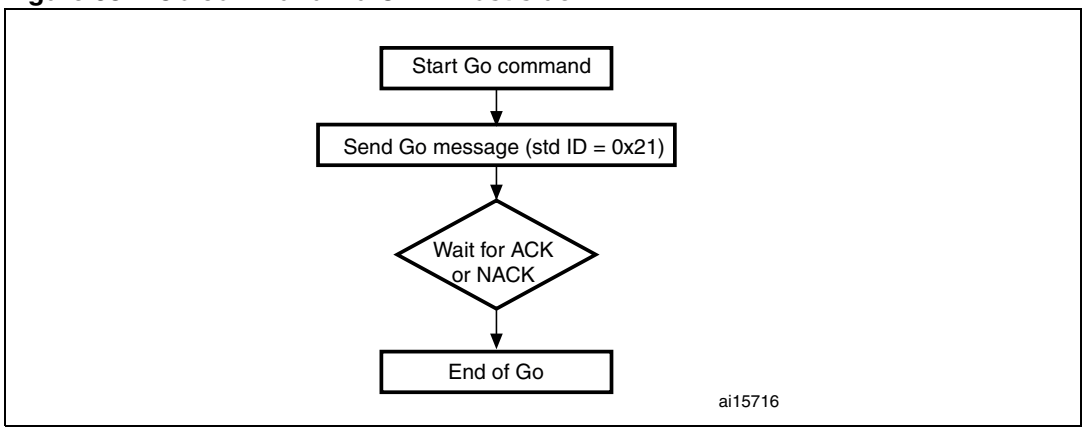
*Note: 1 The Jump to the application works only if the user application sets the vector table correctly to point to the application address.*

*2 When performing a jump from the Bootloader to a loaded application code which uses the USB IP, the user application has to disable all pending USB interrupts and reset the core before enabling interrupts. Otherwise, a pending interrupt (issued from the bootloader code) may interfere with the user code and cause a functional failure. This procedure is not needed after exiting system memory boot mode.*

*3 Valid addresses are RAM (starting from 0x2000 1000 to the end of the RAM) or Flash memory (starting from 0x0800 0000 to the end of the Flash memory) addresses. All other addresses are considered not valid and will be NACKed by the device.*

*4 When an application is loaded into RAM and then a jump is made to it, the program must be configured to run with an offset of at least 0x1000 to avoid overlapping with the first 0x1000 RAM memory used by the bootloader firmware.*

**Figure 38. Go command via CAN: Host side**

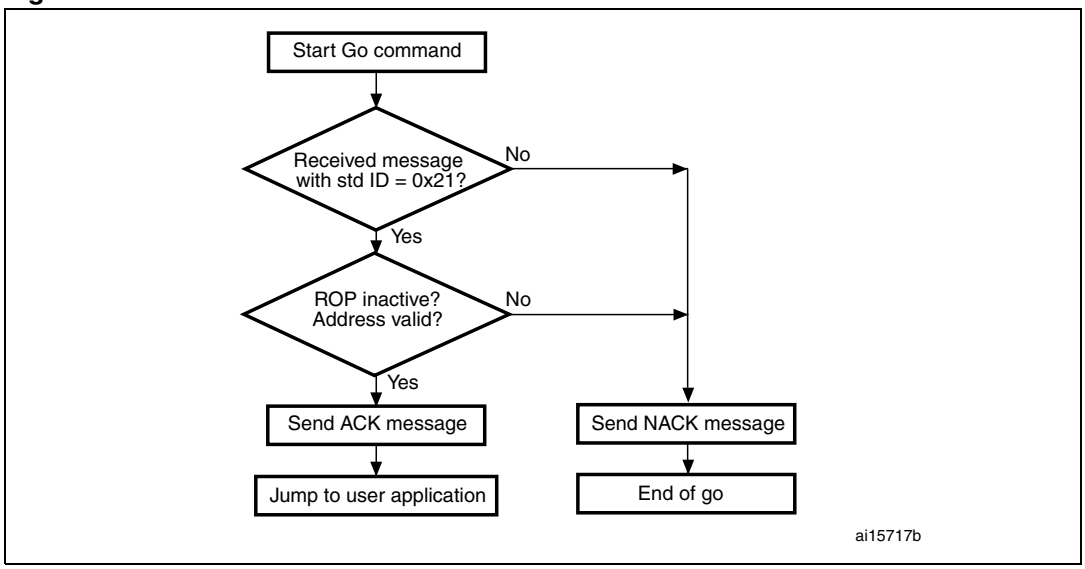



1. See product datasheet for valid addresses.

### The host sends the bytes as follows

Go command message: Std ID = 0x21, DLC = 0x04, data[0] = 0xXX: MSB address,...data[3] = 0xYY LSB address.

**Figure 39. Go command via CAN: Device side**




**The STM32F105xx and STM32F107xx send the messages as follows:**

ACK message: Std ID = 0x21, DLC = 1, data[0] = ACK if content of the command is correct else data[0] = NACK

## 3.10 Write Memory command via CAN

The Write Memory command is used to write data to any memory address of RAM starting from 0x2000 1000, Flash memory, or Option byte area. Refer to the STM32F10xxx Flash programming manual (PM0042). When the bootloader receives the Write Memory command, (message with 5 bytes data length, data[0] is the address MSB, data[3] is the LSB and data[4] is the number of data bytes to be received), it then checks the received address. For the Option byte area, the start address must be 0x1FFFF800 to avoid writing unintentionally in this area.
If the received address is valid, the bootloader transmits an ACK message, otherwise it transmits a NACK message and aborts the command. When the address is valid, the bootloader:

- Receives the user data ((N + 1) bytes) so the device receive (N + 1)/8 messages (each message contains 8 data bytes)
- Programs the user data into memory starting from the received address
- At the end of the command, if the write operation was successful, the bootloader transmits the ACK message; otherwise it transmits a NACK message to the user and aborts the command

The maximum length of the block to be written for the STM32F105xx and STM32F107xx is 256 bytes.

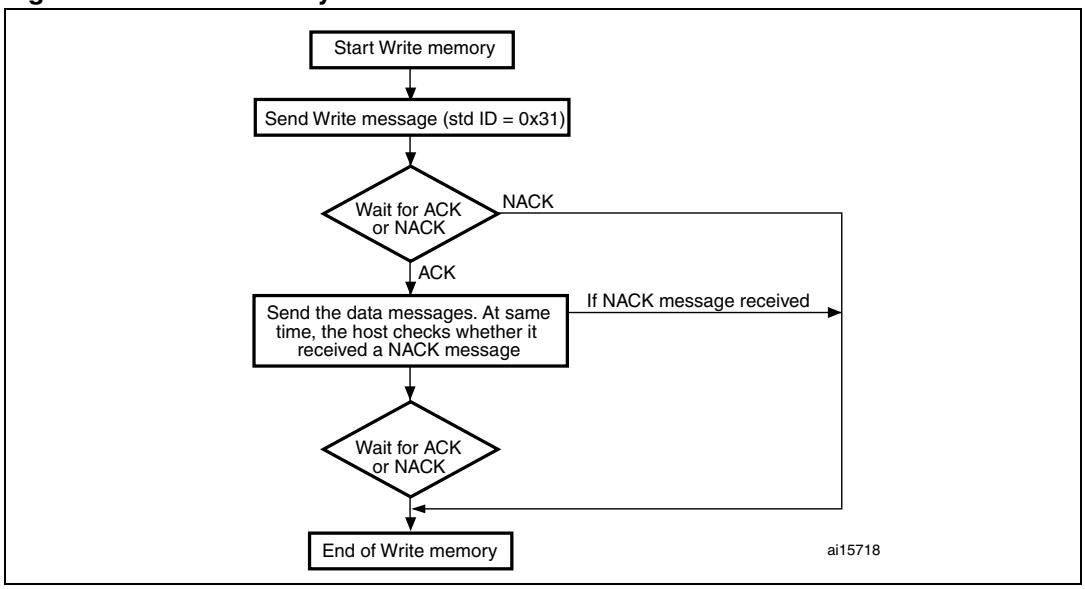
If the Write Memory command is issued to the Option byte area, all options are erased before writing the new values, and at the end of the command the bootloader generates a system Reset to take into account the new configuration of the option byte.

*Note: When writing to the RAM, care must be taken to avoid overlapping the first 4Kbytes (0x1000) in RAM because they are used by the bootloader firmware.*

*Note: No error is returned when performing write operations on write protected sectors.*

*Write operations to FLASH/SRAM must be word aligned, if less data are written the remaining bytes must be filled with 0xFF.*

**Figure 40. Write Memory command via CAN: Host side**




*Note: If the start address is invalid, the command is NACKed by the device.*

### The host sends the messages as follows:

Command message: Std ID = 0x31, DLC = 0x05, data[0] = 0xXX: MSB address,..., data[3] = 0xYY: LSB address, data[4] = N(number of bytes to be written), 0 < N <= 255).

then the host send N/8 message

Data message: Std ID = 0x31, DLC_1 = to 8, data = byte_11, … byte_18…

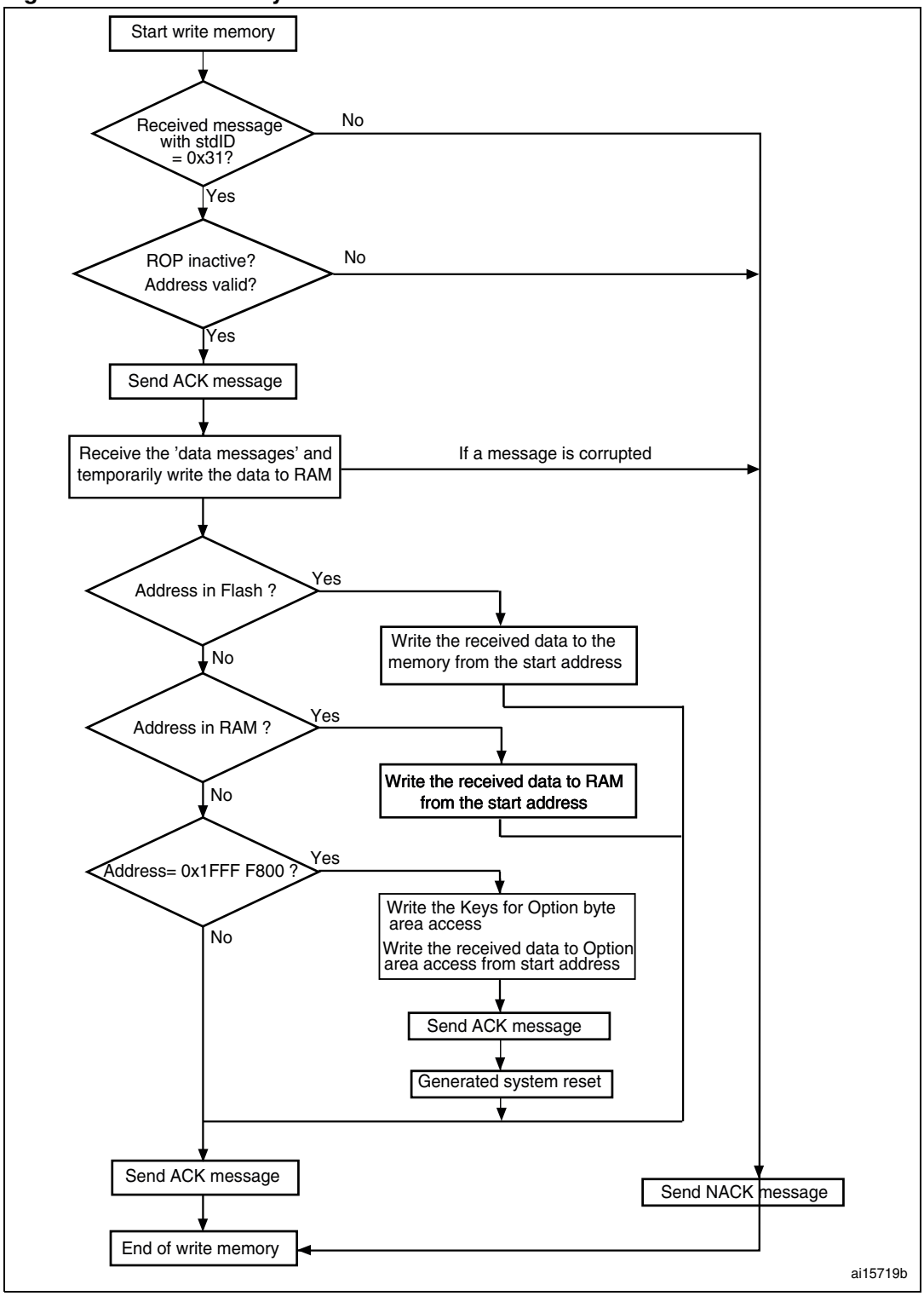
Data message_M: Std ID = 0x04, DLC_M = 1 to 8, data = byte_m1, …, byte_M8

*Note: 1 DLC_1 + DLC_2 + ... DLC_M = 256 maximum*

*2 After each message the host receives the ACK or NACK message from the device*

*3 The bootloader does not check the standard ID of the data, so any ID from 0h to 0xFF can be used. It is recommended to use 0x04.*

**Figure 41. Write memory command via CAN: Device side**




### The STM32F105xx and STM32F107xx sends the messages as follows:

ACK message: Std ID = 0x31, DLC = 1, data[0] = ACK if the content of the command is correct else data[0] = NACK

## 3.11 Erase Memory command via CAN

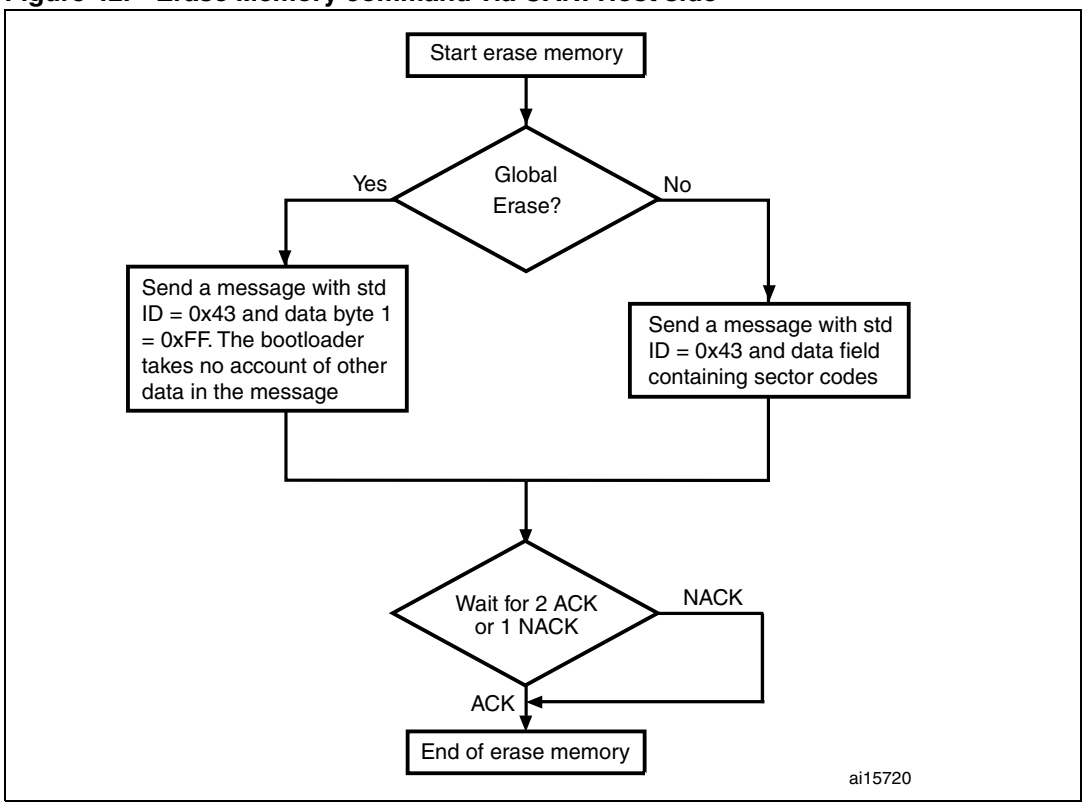
The Erase Memory command allows the host to erase Flash memory pages. When the bootloader receives the Erase Memory command and ROP is disabled, it transmits the ACK message to the host. After the transmission of the ACK message, the bootloader checks if the message that contain data[0].

Erase Memory command specifications:

1. The bootloader receives one message that contains N, the number of pages to be erased – 1.
   N = 255 is reserved for global erase requests. For $0 \leq N \leq 254$, N + 1 pages are erased.
2. The bootloader receives (N + 1) bytes, each byte containing a page number

*Note: No error is returned when performing erase operations on write protected sectors.*

**Figure 42. Erase Memory command via CAN: Host side**

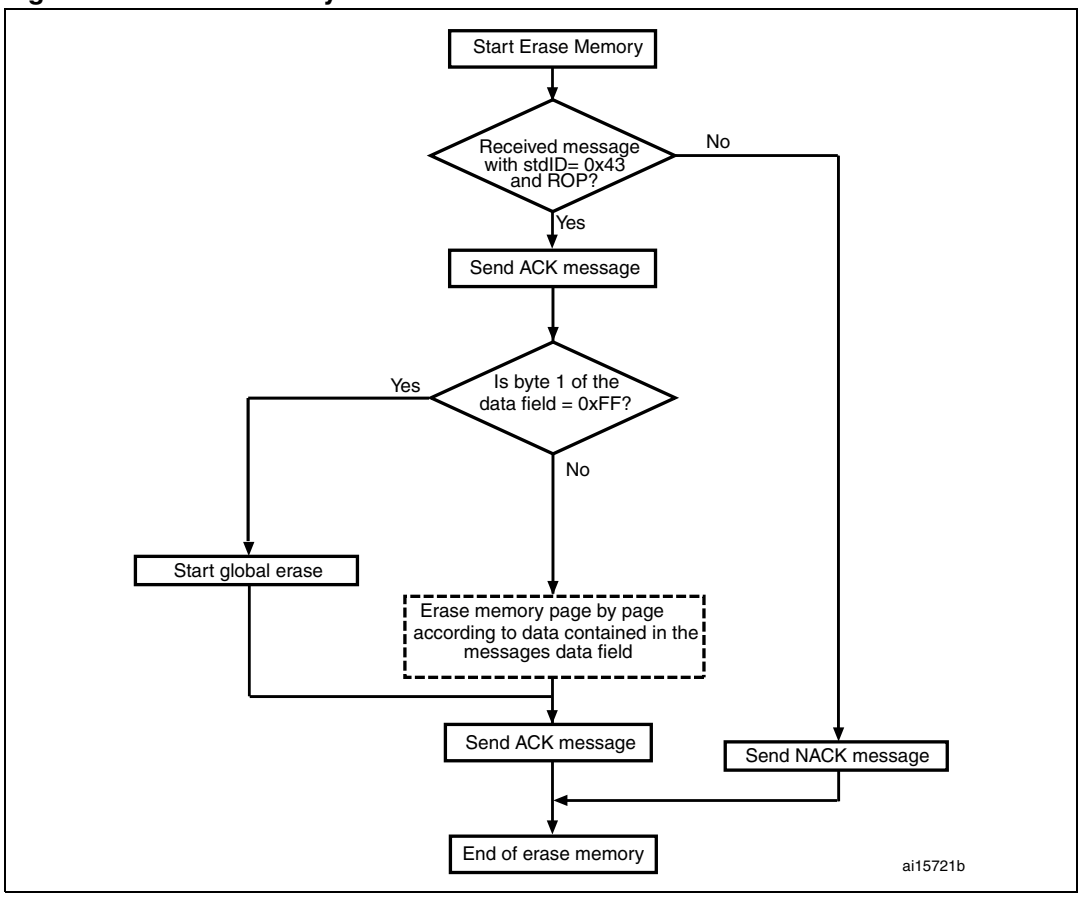



### The host sends the message as follows:

The ID contains the command type (0x43):

- Total erase message: Std ID = 0x43, DLC = 0x01, data = 0xFF.
- Erase sector by sector message: Std ID = 0x43, DLC = 0x01 to 0x08, data = see product datasheet.

**Figure 43. Erase Memory command via CAN: Device side**




### The STM32F105xx and STM32F107xx send the messages as follows:

ACK message: Std ID = 0x43, DLC = 1, data[0] = ACK if content of the command is correct and ROP is not active else data[0] = NACK

## 3.12 Write Protect command

The Write Protect command is used to enable the write protection for some or all Flash memory sectors. When the bootloader receives the Write Protect command, it transmits the ACK message to the host if ROP is disabled else it transmits NACK.

After the transmission of the ACK byte, the bootloader waits to receive the Flash memory sector codes from the user.

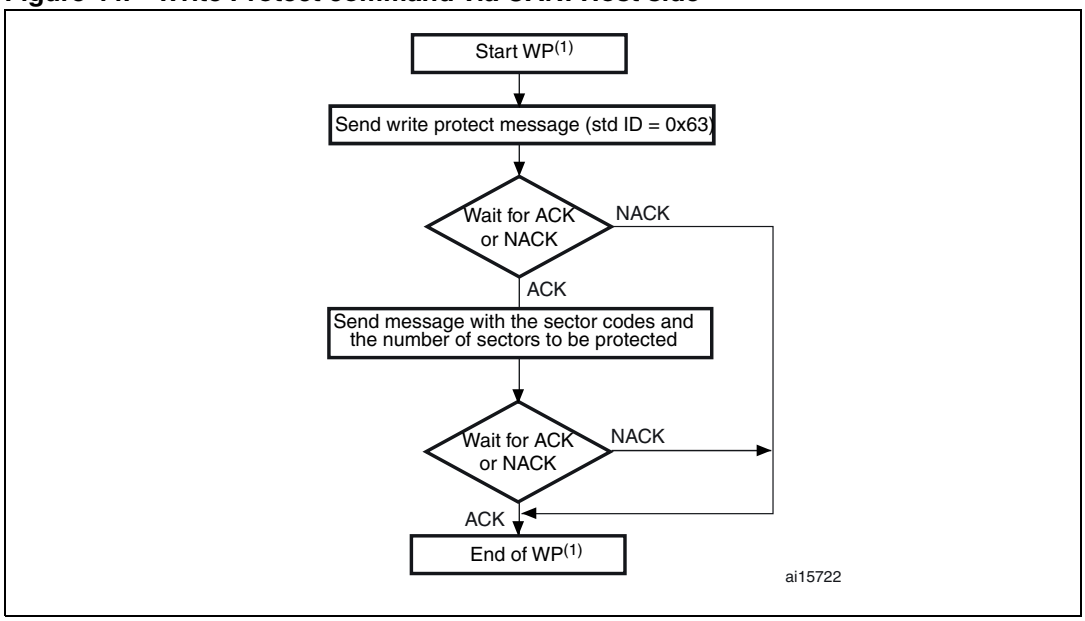
At the end of the Write Protect command, the bootloader transmits the ACK message and generates a system Reset to take into account the new configuration of the option byte.

*Note: On the STM32F105xx and STM32F107xx , the sector size is 4 Kbytes (2 pages) for the Write Protect command.*

*Note: The total number of sectors and the sector number to be protected are not checked, this means that no error is returned when a command is passed with a wrong number of sectors to be protected or a wrong sector number.*

*If a second Write Protect command is executed, the Flash memory sectors that were protected by the first command become unprotected and only the sectors passed within the second Write Protect command become protected.*

**Figure 44. Write Protect command via CAN: Host side**



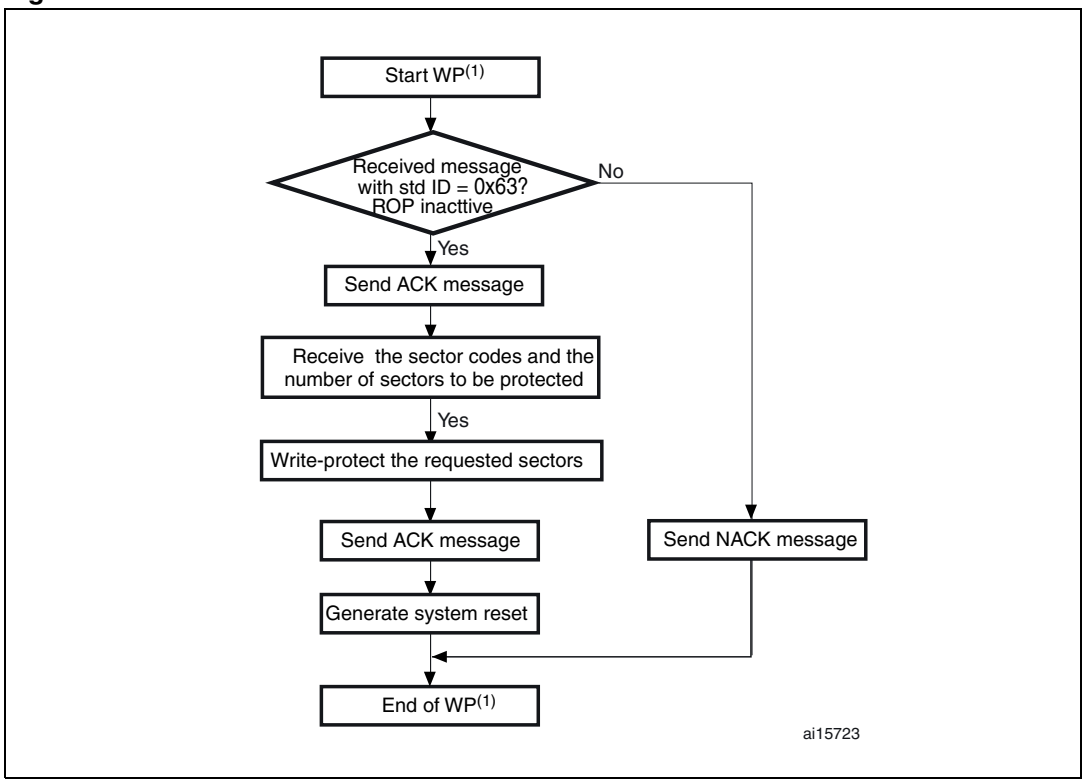

1. WP = Write Protect.

### The host sends the messages as follows:

Command message: Std ID = 0x63, DLC = 0x01, data[0] = N (where 0 < N <= 255).

Command message: Std ID = 0x63, DLC = 0x01..08, data[0] = N (where 0 < N <= 255).

**Figure 45. Write Protect command via CAN: device side**




1. WP = Write Protect

### The STM32F105xx and STM32F107xx send the messages as follows:

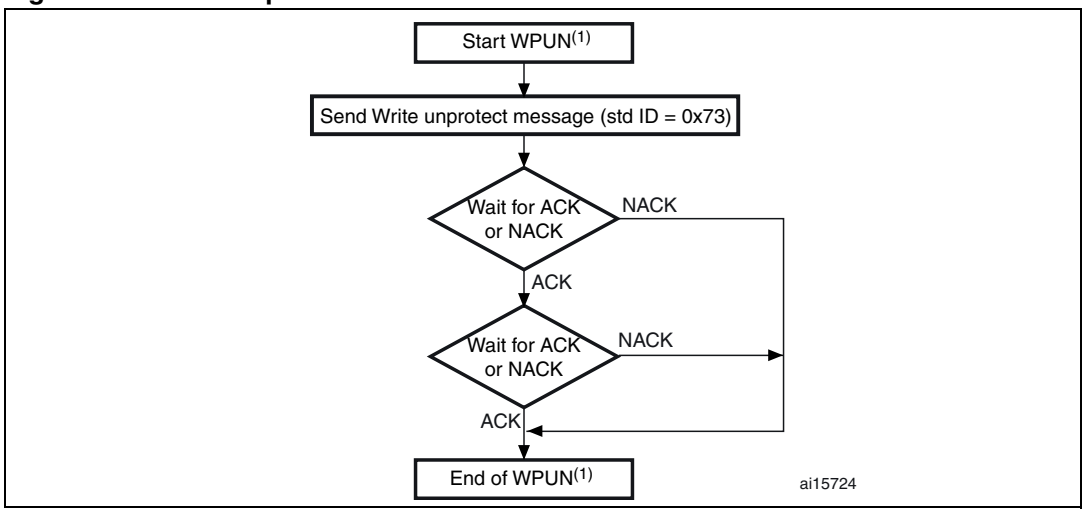
ACK message: Std ID = 0x43, DLC = 1, data[0] = ACK if the content of the command is correct and ROP is not active else data[0] = NACK.

## 3.13 Write Unprotect command

The Write Unprotect command is used to disable the write protection of all the Flash memory sectors. When the bootloader receives the Write Unprotect command, it transmits the ACK message to the host if ROP is disabled else it transmits NACK. After the transmission of the ACK message, the bootloader disables the write protection of all the Flash memory sectors.

At the end of the Write Unprotect command, the bootloader transmits the ACK message and generates a system Reset to take into account the new configuration of the option byte.

**Figure 46. Write Unprotect command: Host side**

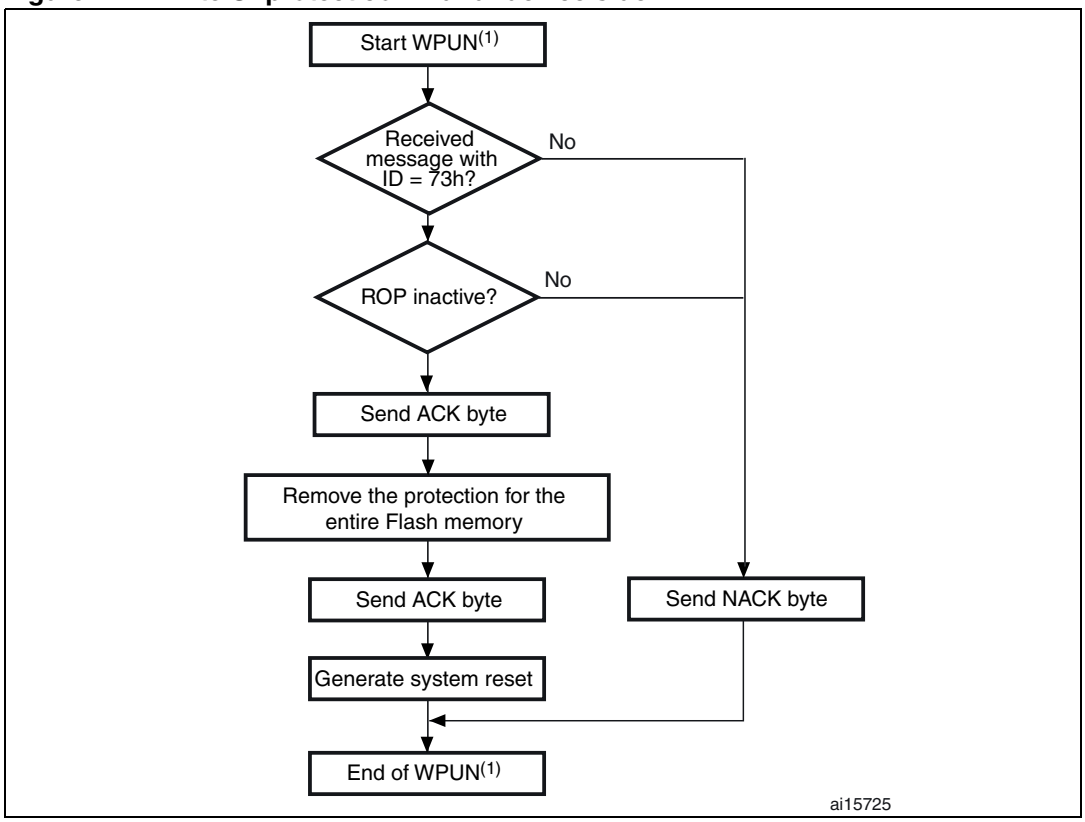



1. WPUN = Write Unprotect.

### The host sends the messages as follows:

Command message: Std ID = 0x73, DLC = 0x01, data = 00.

**Figure 47. Write Unprotect command: device side**




1. WPUN = Write Unprotect.

### The STM32F105xx and STM32F107xx send the messages as follows:

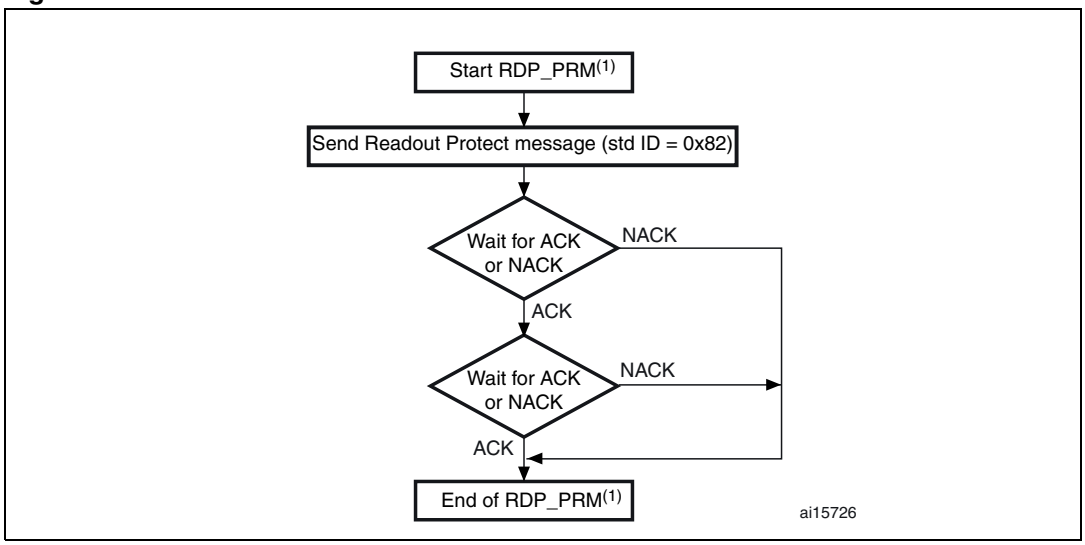
ACK message: Std ID = 0x73, DLC = 1, data[0] = ACK if the content of the command is correct and ROP is not active else data[0] = NACK.

## 3.14 Readout Protect command

The Readout Protect command is used to enable the Flash memory read protection. When the bootloader receives the Readout Protect command, it transmits the ACK message to the host if ROP is disabled else it transmits NACK. After the transmission of the ACK message, the bootloader enables the read protection for the Flash memory.

At the end of the Readout Protect command, the bootloader transmits the ACK message and generates a system Reset to take into account the new configuration of the option byte.

**Figure 48. Readout Protect command via CAN: Host side**

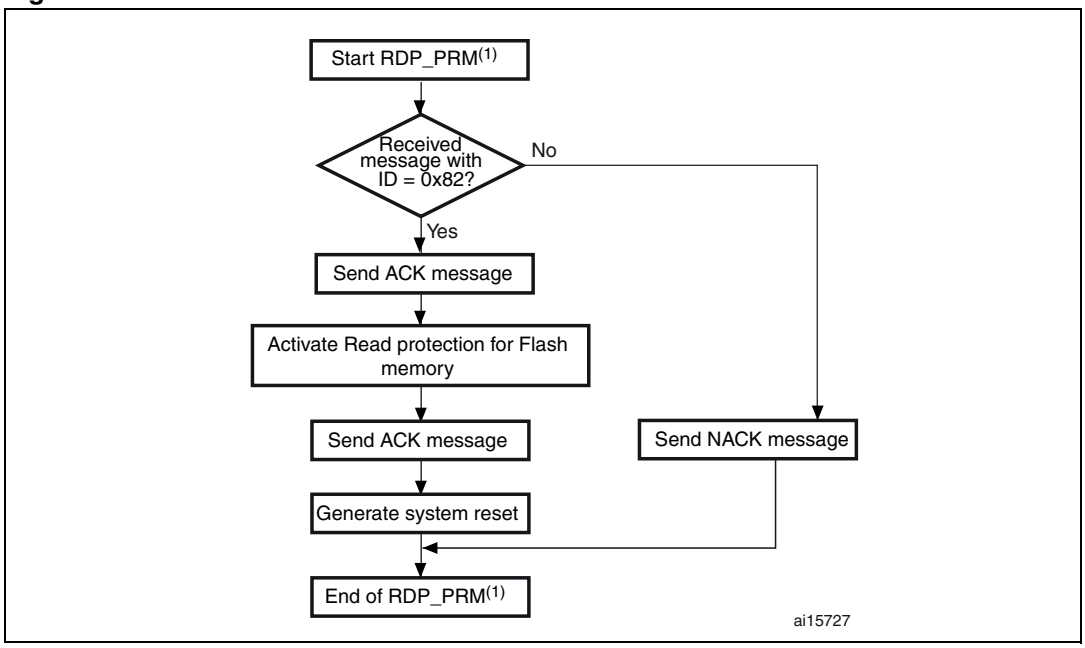



1. RDP_PRM = Readout Protect.

### The host sends the messages as follows

Command message: Std ID = 0x82, DLC = 0x01, data[0] = 00.

**Figure 49. Readout Protect command via CAN: device side**




1. RDP_PRM = Readout Protect

### The STM32F105xx and STM32F107xx send the messages as follows:

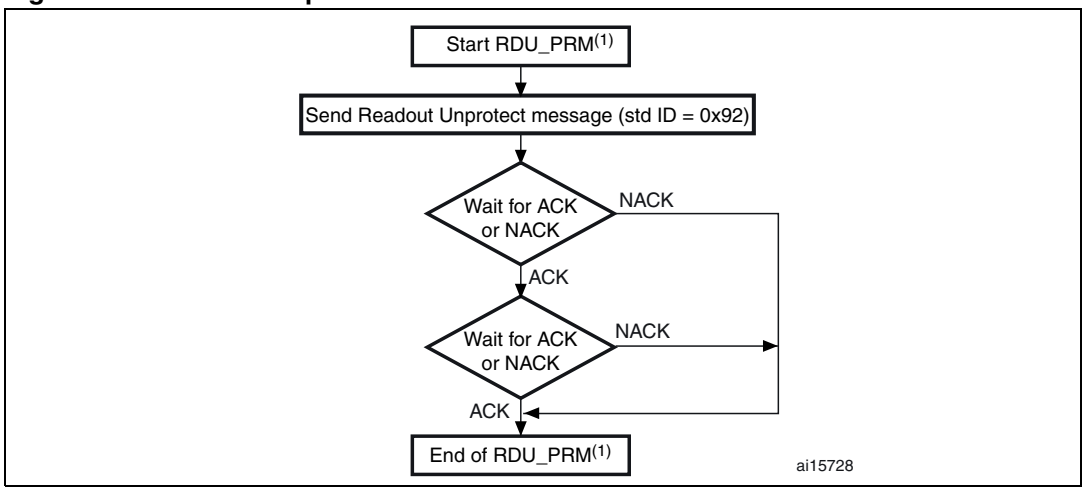
ACK message: Std ID = 0x82, DLC = 1, data[0] = ACK if the content of the command is correct and ROP is not active else data[0] = NACK.

## 3.15 Readout Unprotect command

The Readout Unprotect command is used to disable the Flash memory read protection. When the bootloader receives the Readout Unprotect command, it transmits the ACK message to the host. After the transmission of the ACK message, the bootloader erases all the Flash memory sectors and it disables the read protection for the entire Flash memory. If the erase operation is successful, the bootloader deactivates the RDP.

At the end of the Readout Unprotect command, the bootloader transmits an ACK message and generates a system Reset to take into account the new configuration of the option bytes.

**Figure 50. Readout Unprotect command via CAN: Host side**

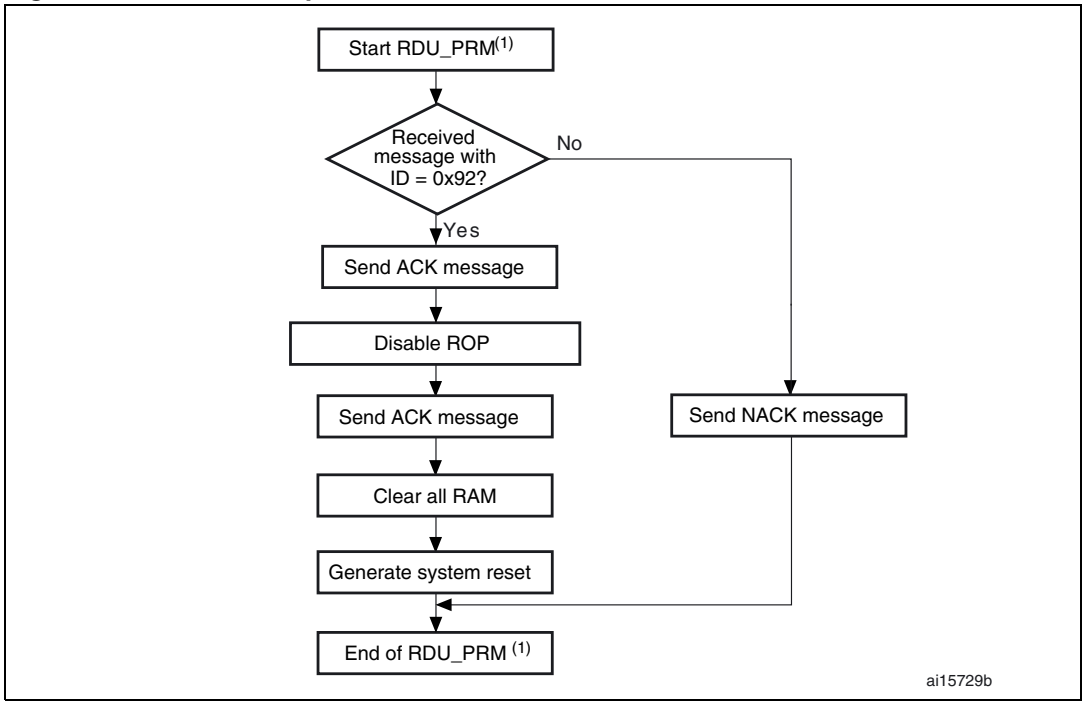



1. RDU_PRM = Readout Unprotect.

### The host sends the messages as follows

Command message: Std ID = 0x92, DLC = 0x01, data = 00.

**Figure 51. Readout Unprotect command via CAN: device side**




1. RDU_PRM = Readout Unprotect.

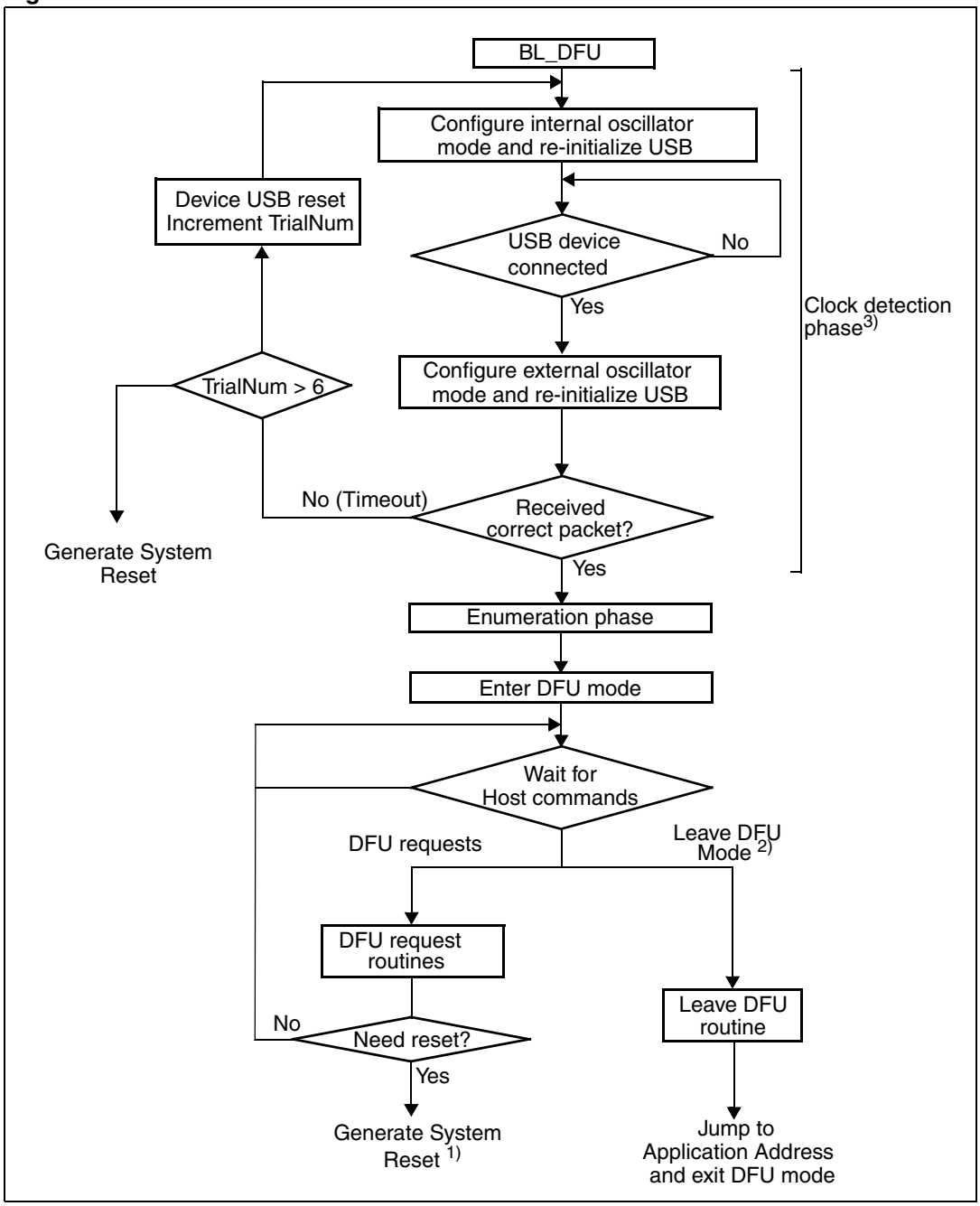
### The STM32F105xx and STM32F107xx send the messages as follows:

ACK message: Std ID = 0x92, DLC = 1, data[0] = ACK if the content of the command is correct and ROP is not active else data[0] = NACK.

# 4 DFU bootloader

## 4.1 Bootloader code sequence

**Figure 52. Bootloader for STM32F105xx and STM32F107xx with USB DFU**




1. After system reset, the device may return to the BL_DFU loop, enter the USART or CAN bootloader loops or execute code from Flash/RAM depending on the connection states and the boot pins status.
2. Leave DFU is achieved by a 0 Data Download request followed by GetStatus request and Device Reset.
3. After six trials (the three clock configurations are tested twice), a System Reset is generated.

Once system memory boot mode is entered and the **STM32F105xx and STM32F107xx** has been configured as described above, the bootloader code configures the USB and its interrupts and waits for the “enumeration done” interrupt.

Once this interrupt is detected (A Host is present, has connected the device and enumerated it), the system is configured in External Oscillator mode and the USB device is re-initialized.

The device first tries the 25 MHz configuration, then, if it fails, the 14.7456 MHz configuration, then, if it fails, the 8 MHz configuration. If it fails, this operation is repeated with a large timeout value (the three configurations are re-tested). If the second trial also fails, a system reset is then generated.

The USB enumeration is performed as soon as the USB cable is plugged (or immediately if the cable is already plugged). If you do not want the **STM32F105xx and STM32F107xx** to enter the USB DFU bootloader application, the USB cable has to be unplugged before reset.

The bootloader version is returned in the device descriptor in the MSB of the bcdDevice field (example: 0x1000 = Version 1.0).

## 4.2 USB DFU bootloader requests

USB DFU Bootloader supports DFU protocol and requests compliant with the “Universal Serial Bus Device Upgrade Specification for Device Firmware Upgrade” Version 1.1, Aug 5,2004. For more details concerning these requests refer to this specification document.

*Table 5* and *Table 6* enumerate the DFU Class-Specific requests and their parameters.

**Table 5. DFU class requests**

| Request | Request code | Request description |
|---|---|---|
| DFU_DETACH | 0x00 | Requests the device to leave DFU mode and enter the application. |
| DFU_DNLOAD | 0x01 | Requests data transfer from Host to the device in order to load them into device internal Flash. Includes also erase commands. |
| DFU_UPLOAD | 0x02 | Requests data transfer from device to Host in order to load content of device internal Flash into a Host file. |
| DFU_GETSTATUS | 0x03 | Requests device to send status report to the Host (including status resulting from the last request execution and the state the device will enter immediately after this request). |
| DFU_CLRSTATUS | 0x04 | Requests device to clear error status and move to next step. |
| DFU_GETSTATE | 0x05 | Requests the device to send only the state it will enter immediately after this request. |
| DFU_ABORT | 0x06 | Requests device to exit the current state/operation and enter idle state immediately. |

*Note: The Detach request is not meaningful in the case of the bootloader. The bootloader is started by a system reset depending on the boot mode configuration settings, which means that no other application is running at this time.*

**Table 6. Summary of DFU class-specific requests**

| bmRequest | bRequest | wValue | wIndex | wLength | Data |
|---|---|---|---|---|---|
| 00100001b | DFU_DETACH | wTimeout | Interface | Zero | None |
| 00100001b | DFU_DNLOAD | wBlockNum | Interface | Length | Firmware |
| 10100001b | DFU_UPLOAD | Zero | Interface | Length | Firmware |
| 00100001b | DFU_GETSTATUS | Zero | Interface | 6 | Status |
| 00100001b | DFU_CLRSTATUS | Zero | Interface | Zero | None |
| 00100001b | DFU_GETSTATE | Zero | Interface | 1 | State |
| 00100001b | DFU_ABORT | Zero | Interface | Zero | None |

### Communication safety

The communication between Host and Device is secured by the embedded USB protection mechanisms (CRC checking, Acknowledgements...). No further protection is performed for transferred data or for bootloader specific commands/data.

## 4.3 DFU bootloader commands

The DFU_DNLOAD and DFU_UPLOAD requests are mainly used to perform simple Write Memory and Read Memory operations. They are also used to initiate the integrated bootloader commands (write, read unprotect, erase, set address...). The DFU_GETSTATUS command then triggers the command execution.

The selection of a command through the DFU download request is done through the **wValue** parameter in the USB request structure. If **wValue** = 0 then the data sent by the Host after the request is a bootloader command code. The first byte is the command code and the other bytes (if any) are the data related to this command.

The selection of a command through the DFU upload request is done through the **wValue** parameter in the USB request structure. If **wValue** = 0 then Get Command is selected and performed.

**Table 7. DFU bootloader commands**

| DFU request | Bootloader command | Write protection disabled<br>Read protection disabled | Write protection enabled<br>Read protection disabled | Read protection enabled |
|---|---|---|---|---|
| DFU_UPLOAD | **Read Memory** | Allowed | Allowed | Not allowed |
| | **Get Commands** | Allowed | Allowed | Allowed |
| DFU_DNLOAD | **Write Memory** | Allowed | Allowed(1) | Not allowed |
| | **Erase** | Allowed | Allowed(1) | Not allowed |
| | **Read Unprotect** | NA(2) | NA(2) | Allowed(3) |
| | **Set Address Pointer** | Allowed | Allowed | Allowed |
| | **Leave DFU mode** | Allowed | Allowed | Allowed |

1. This operation is allowed but not effective: the bootloader does not return an error but the operation is not executed since the sectors are write-protected. This applies only to the Flash memory. It does not apply to the RAM memory or the option byte area.
2. This operation is allowed but has no meaning since the memory is not protected.
3. In this case, both the Flash memory (from 0x0800 0000) and the RAM are erased. The option byte area is reset to default values.

If you perform a Read Unprotect operation while the memory is not protected, the entire RAM memory is cleared by the bootloader firmware and the Flash memory is not erased (since it was not previously read-protected).

There are no commands for the Write Protect, Write Unprotect and Read Protect operations. These operations should be performed through the Write Memory and Read Memory commands used for the option byte area.

## 4.4 DFU_UPLOAD request commands

The upload request allows different commands to be performed. The command selection is done through the value of parameter **wValue** in the USB request structure. The operations described in *Section 4.4.1* to *Section 4.5.5* are supported.

### 4.4.1 Read memory

The Read operation is selected when **wValue** > 1.

The host requests the device to send a specified number of data bytes (**wLength**) from Internal Flash, Embedded RAM (starting from 0x2000 1000 address), system memory or from option bytes. The allowed number of bytes to be read depends on the memory target:

- For internal Flash, embedded RAM and system memory: read size can be from 2 to 2048 bytes.
- For Option Bytes: read size should be 16 bytes.

The address, from which the Host requests to read data, is computed using the value of wBlockNumber (**wValue**) and the address pointer according to the following formula:

Address = ((wBlockNum - 2) * wTransferSize) + Address_Pointer.

Where wTransferSize is the length of the requested data buffer.

The address pointer should have been previously specified through a Set Address Pointer command (using a DFU_DNLOAD request). Otherwise (if no address is previously specified) the device assumes that it will be the internal Flash start address (0x08000000).

If the Flash Read Protection is enabled, the Read operation is not performed and the device status returned is (Status = dfuERROR, State = errVENDOR) whatever the target (internal Flash, embedded RAM, system memory or option bytes).

### 4.4.2 Get commands

This command is selected when **wValue** = 0.

The Host requests to read the commands supported by the bootloader. After receiving this command, the device returns N bytes representing the command codes.

The STM32F105xx and STM32F107xx sends the bytes as follows (N = 4):

Byte 1: 0x00 - Get command

Byte 2: 0x21 - Set Address Pointer

Byte 3: 0x41 - Erase

Byte 4: 0x92 - Read Unprotect

The processing of DFU_UPLOAD command is shown in *Figure 53* and *Figure 54*.

**Figure 53. DFU_UPLOAD request: Device side**

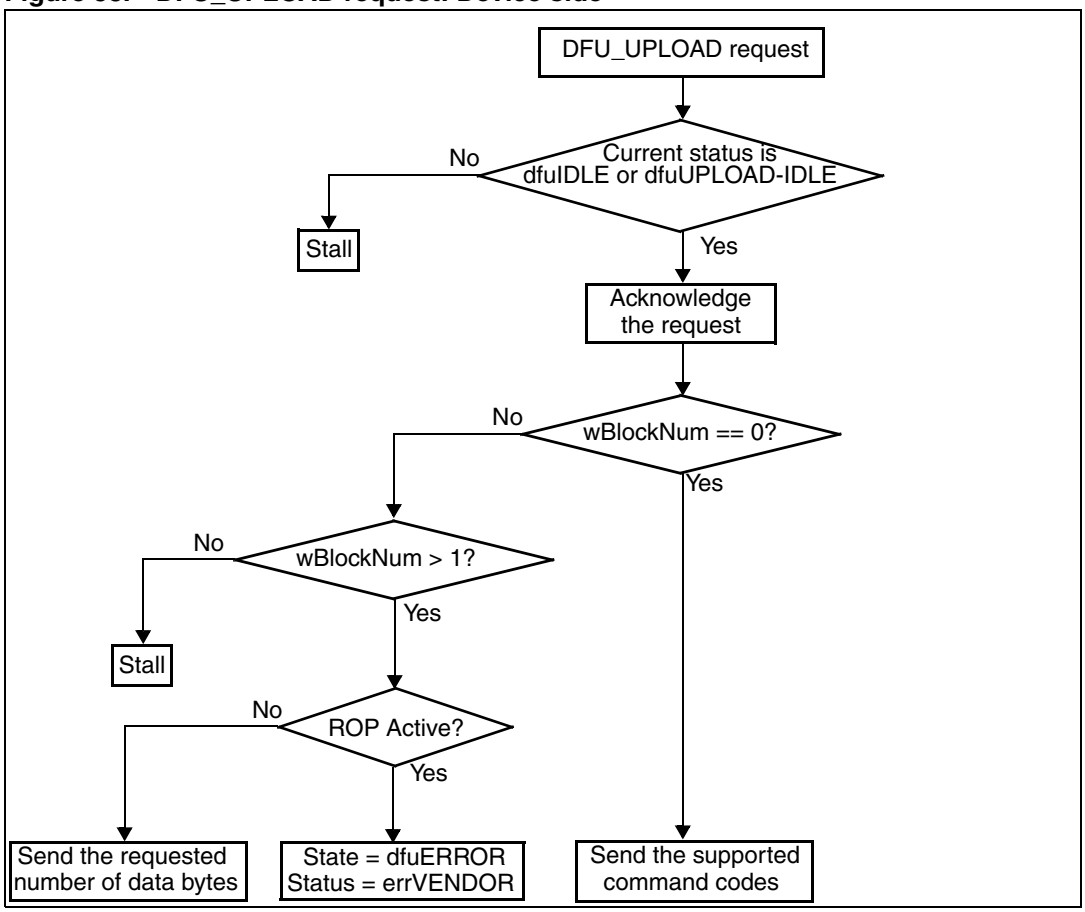
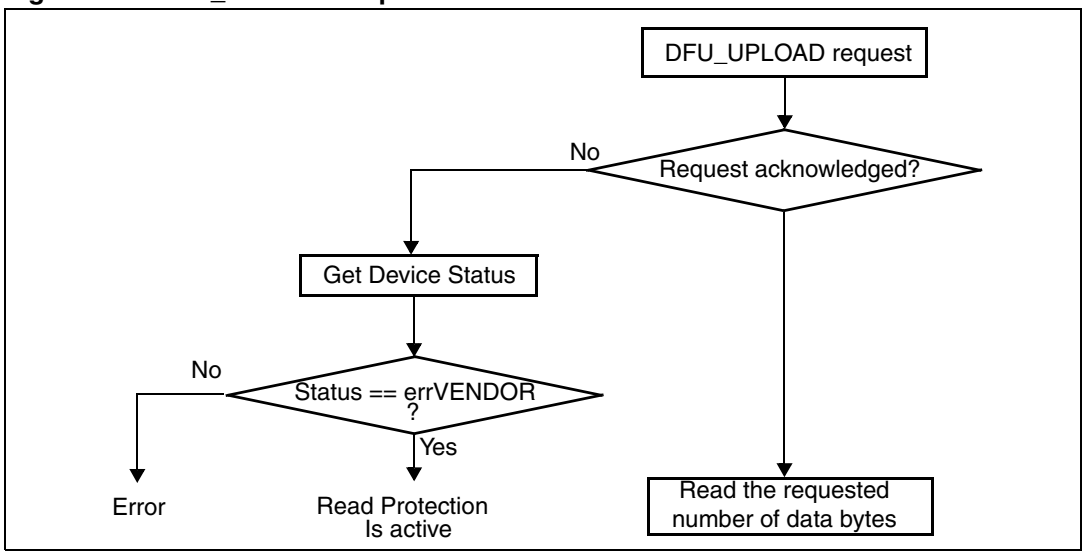


**Figure 54. DFU_UPLOAD request: Host side**



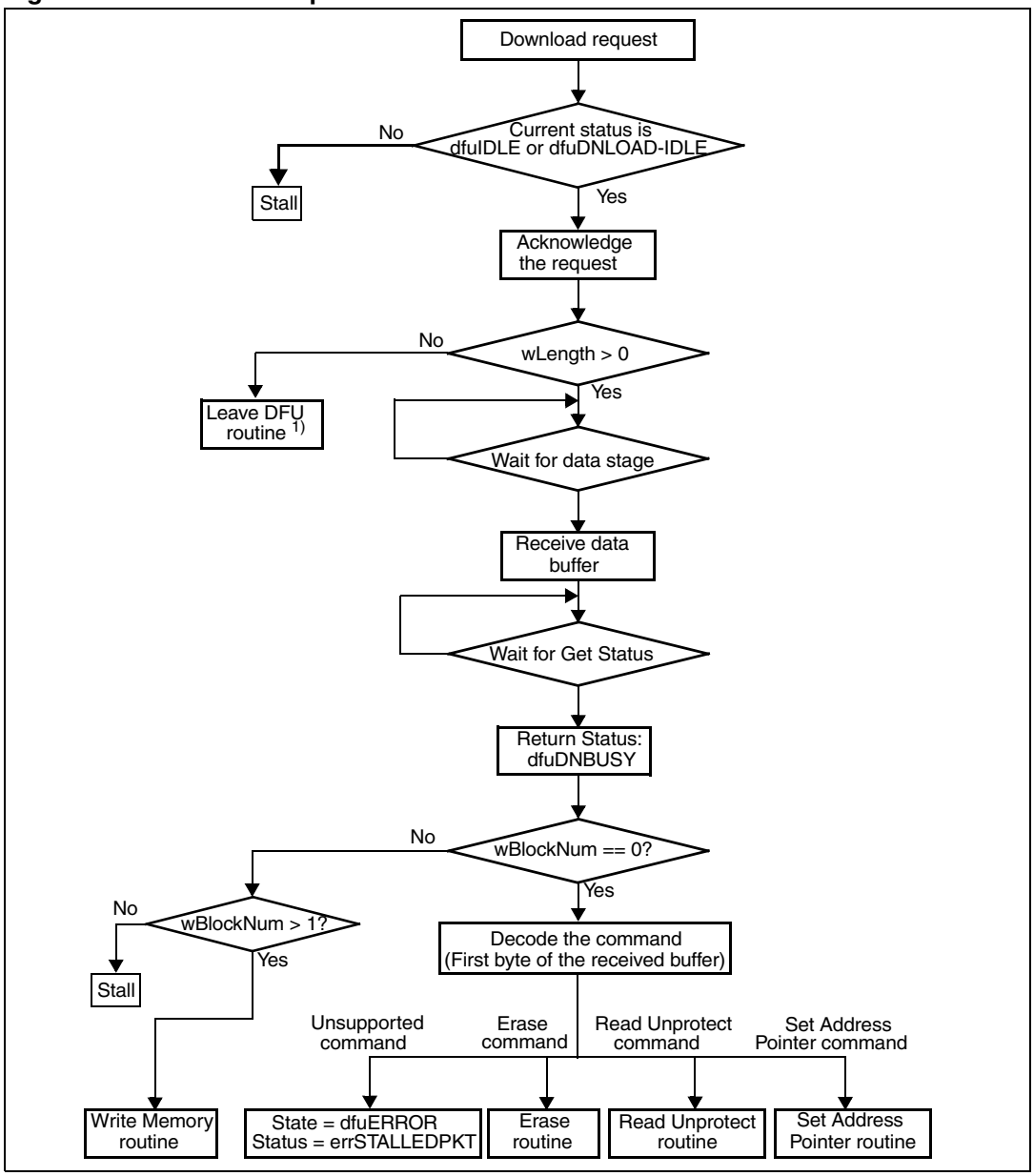

*Note: Before issuing an Upload request, the host has to check that the device is in a correct state (dfuIDLE or dfuUPLOAD-IDLE state) and that there is no error reported in the status. If the Device is not in the required state/status, the Host has to clear any error (DFU_CLRSTATUS request) and get the new status till the Device returns to dfuIDLE state.*

## 4.5 DFU_DNLOAD request commands

The download request allows to perform different commands. The command selection is done through the value of parameter **wValue** in the USB request structure. The following operations are supported:

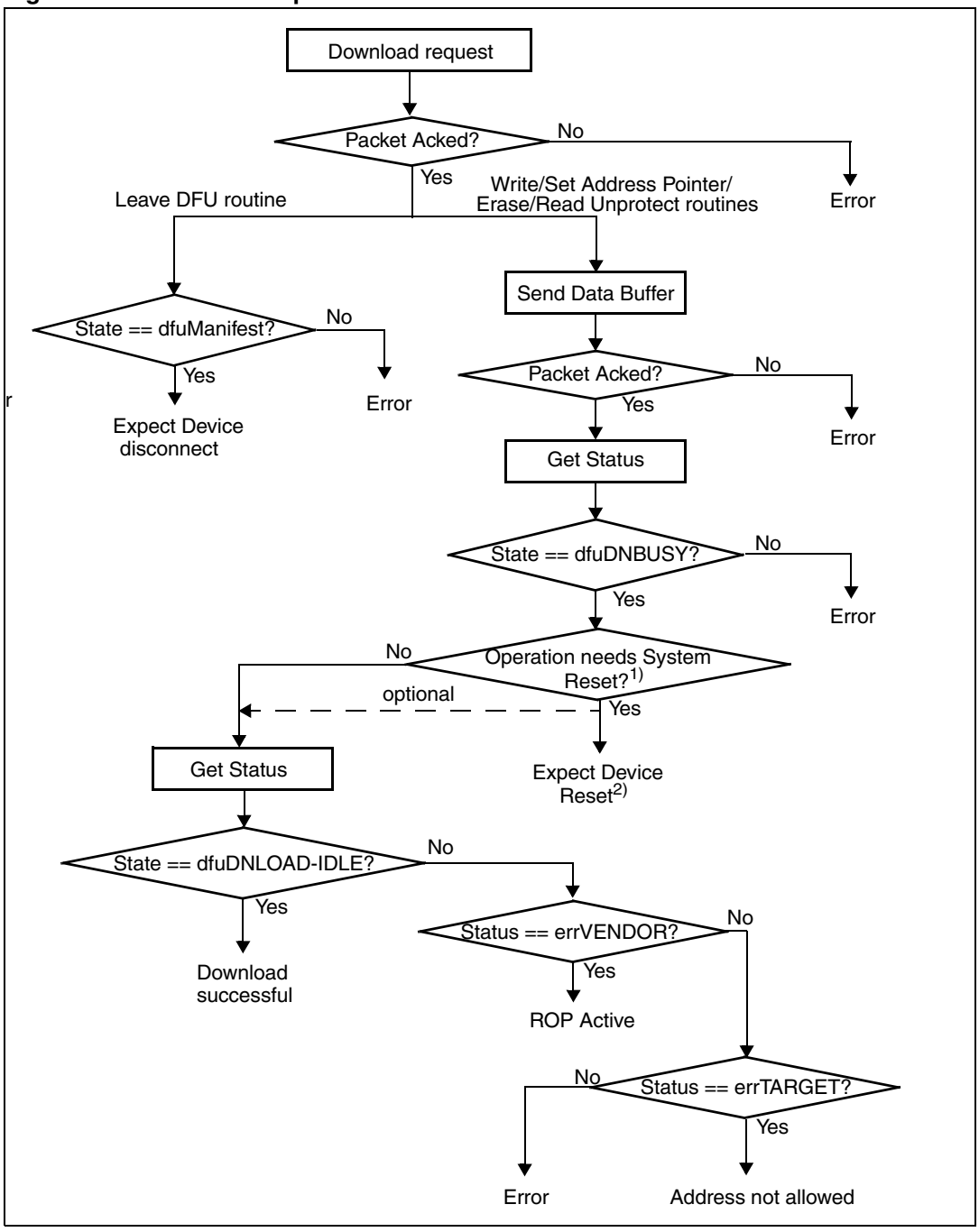
- Write Memory (**wValue** > 1)
- Set Address Pointer (**wValue** = 0 and first byte = 0x21)
- Erase (**wValue** = 0 and First Byte = 0x41)
- Read Unprotect (**wValue** = 0 and first byte = 0x92)
- Leave DFU (leave DFU mode and Jump to application)

**Figure 55. Download request: Device side**




1. This routine can be used to reset the device or to jump to the application.

**Figure 56. Download request: Host side**




1. Operations needing System Reset are: Read Unprotect command and Write operations to the Option Bytes.
2. After returning dfuDNBUSY state, the Device executes the requested operation and performs a System Reset. The Host may simply wait for next enumeration or perform Get status again but the device won't be able to respond, unless it fails to execute the requested operation.

*Note: Before issuing a Download request, the host has to check that the device is in a correct state: dfuIDLE or dfuDNLOD-IDLE, and that there is no error reported in the status. If the Device is not in the required state/status, the Host has to clear any error (DFU_CLRSTATUS request) and get again the status till the Device returns to dfuIDLE state.*

### 4.5.1 Write memory

The Write Memory operation is selected when **wValue** > 1.

The host requests the device to receive a specified number of data bytes (**wLength**) to load them into internal Flash, embedded RAM (starting from 0x2000 1000) or into Option Bytes. The allowed number of bytes to be written depends on the memory target:

- For internal Flash and embedded RAM: write size can be from 2 to 2048 Bytes.
- For Option Bytes: write size should be 16 Bytes.

*Note: A different write size is possible for the Option Bytes but it is recommended to write the entire block (16 bytes) at one time in order to insure data integrity. When the target is the Option Byte area, the Address pointer must always be the start address of the Option Bytes, otherwise, the request is not performed.*

The Write operation is effectively executed only when a DFU_GETSTATUS request is issued by the Host. If the status returned by the device is other than dfuDNBUSY, then an error has occurred.

A second DFU_GETSTATUS request is needed to check if the command has been correctly executed, except when the destination is the Option Bytes area (in this case the device will immediately reset after write operation completion). If the received address is wrong or unsupported, the device status will then be (Status = dfuERROR, State = errTARGET).

The address, to which the Host requests to write data, is computed using the value of wBlockNumber (**wValue**) and the address pointer according to the same formula as for an upload request:

Address = ((wBlockNum - 2) * wTransferSize) + Addres_Pointer.

Where wTransferSize is the length of the data buffer sent by the host and wBlockNumber is the value of **wValue** parameter.

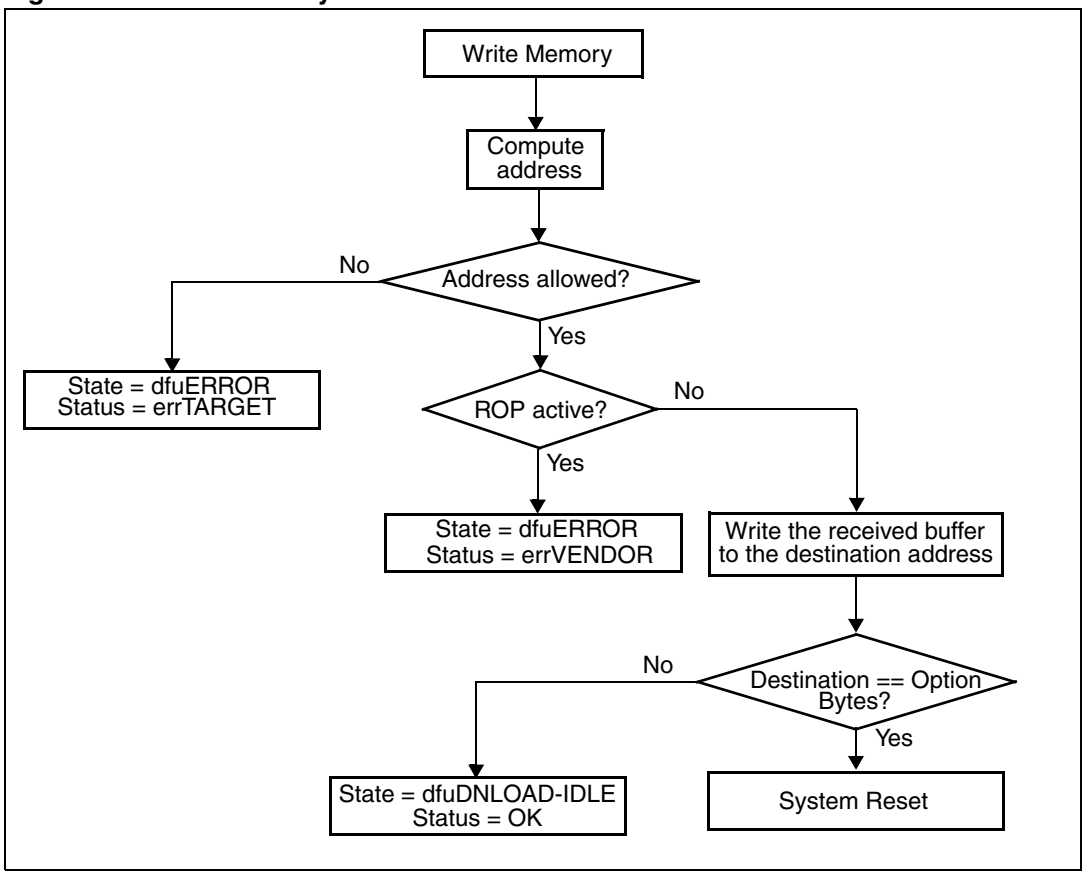
If the Flash Read Protection is enabled, the Write operation is not performed and the device status returned is (Status = dfuERROR, State = errVENDOR) whatever the target (internal Flash, embedded RAM or Option Bytes).

If the Write Memory command is issued to the Option byte area, all options are erased before writing the new values, and at the end of the command the bootloader generates a system Reset to take into account the new configuration of the option byte

*Note: When writing to the RAM, care must be taken to avoid overlapping the first 4 Kbytes (0x1000) in RAM because they are used by the bootloader firmware.*

*Note: No error is returned when performing write operations on write protected sectors.*

**Figure 57. Write Memory: Device side**




### 4.5.2 Set Address Pointer command

The Set Address Pointer command is selected when wValue = 0 and the first byte of the buffer sent by the Host is 0x21. The Buffer length should be 5 (the four remaining bytes are the address bytes, LSB first (32-bit address format)).

The Host sends a DFU_DNLOAD request with the parameters above to set the address pointer value used for computing the start address for Read and Write operations.

The STM32F105xx and STM32F107xx receives the bytes as follows:

Byte 1: 0x21 - Set Address Pointer command

Byte 2: A[7:0] - LSB of the address pointer

Byte 3: A[15:8] - Second byte of the address pointer

Byte 4: A[22:16] - Third byte of the address pointer

Byte 4: A[31:23] - MSB of the address pointer

After sending Set Address Pointer command, the host has to send DFU_GETSTATUS request.

The Set AddressPointer command is effectively executed only when a DFU_GETSTATUS request is issued by the Host. If the status returned by the device is other than dfuDNBUSY, then an error has occurred.

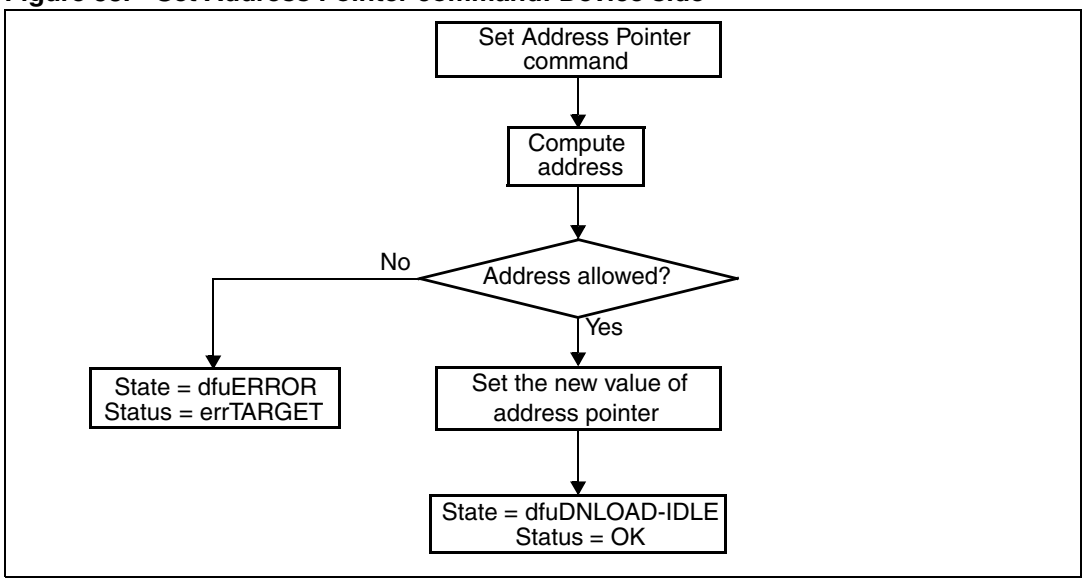
A second DFU_GETSTATUS request is needed to check if the command has been correctly executed. If the received address is wrong or unsupported, the device status will then be (Status = dfuERROR, State = errTARGET).

The allowed locations for address pointer values are:

- Internal Flash and embedded RAM (starting from 0x2000 1000) addresses.
- System memory addresses
- Option byte addresses

*Note: The Set Address Pointer command is allowed and executed when the Flash Read Protection is enabled or disabled.*

**Figure 58. Set Address Pointer command: Device side**




### 4.5.3 Erase command

The Erase command is selected when **wValue** = 0 and the first byte of the buffer sent by the Host is 0x41. The Buffer length may be 5 bytes (the four remaining bytes are the address bytes, LSB first) for the page erase operation or only 1 byte (only the command byte) for the Mass erase operation.

The Host sends a DFU_DNLOAD request with the above parameters to erase one page of the internal Flash memory or to perform a mass erase of this Flash.

The STM32F105xx and STM32F107xx receives the bytes as follows (page erase):

Byte 1: 0x41 - Erase command

Byte 2: A[7:0] - LSB of the page address

Byte 3: A[15:8] - Second byte of the page address

Byte 4: A[22:16] - Third byte of the page address

Byte 4: A[31:23] - MSB of the page address

Or, if a 1-byte command is received:

The STM32F105xx and STM32F107xx receives the bytes as follows (Mass Erase):

Byte 1: 0x41 - Erase command

After sending an Erase command, the host has to send a DFU_GETSTATUS request.

The Erase command is effectively executed only when a DFU_GETSTATUS request is issued by the Host. If the status returned by the device is other than dfuDNBUSY, then an error has occurred.

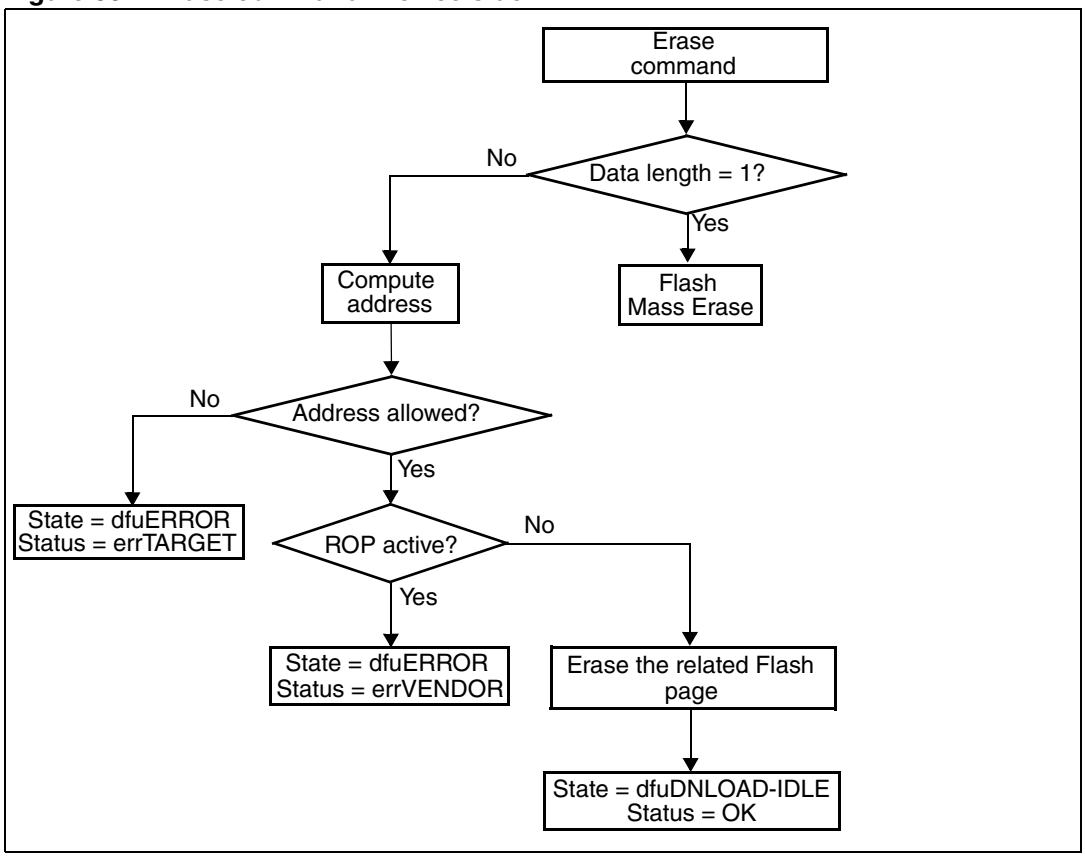
A second DFU_GETSTATUS request is needed to check if the command has been correctly executed. If the received page address is wrong or unsupported, the device status will then be (Status = dfuERROR, State = errTARGET). If the Flash Read Protection is active, then the device returns the status (Status = dfuERROR, State = errVENDOR) and the erase operation is ignored by the device.

The allowed values for Erase page address are:

- Internal Flash memory addresses.

*Note: No error is returned when performing erase operations on write protected sectors.*

**Figure 59. Erase command: Device side**



### 4.5.4 Read Unprotect command

The Read Unprotect command is selected when **wValue** = 0 and the first byte of the buffer sent by the Host is 0x92. The Buffer length should be only 1 byte (only the command byte).

The Host sends a DFU_DNLOAD request with the above parameters to remove the read protection of the internal Flash memory.

The STM32F105xx and STM32F107xx receives the byte as follows:

Byte 1: 0x92 - Read Unprotect command

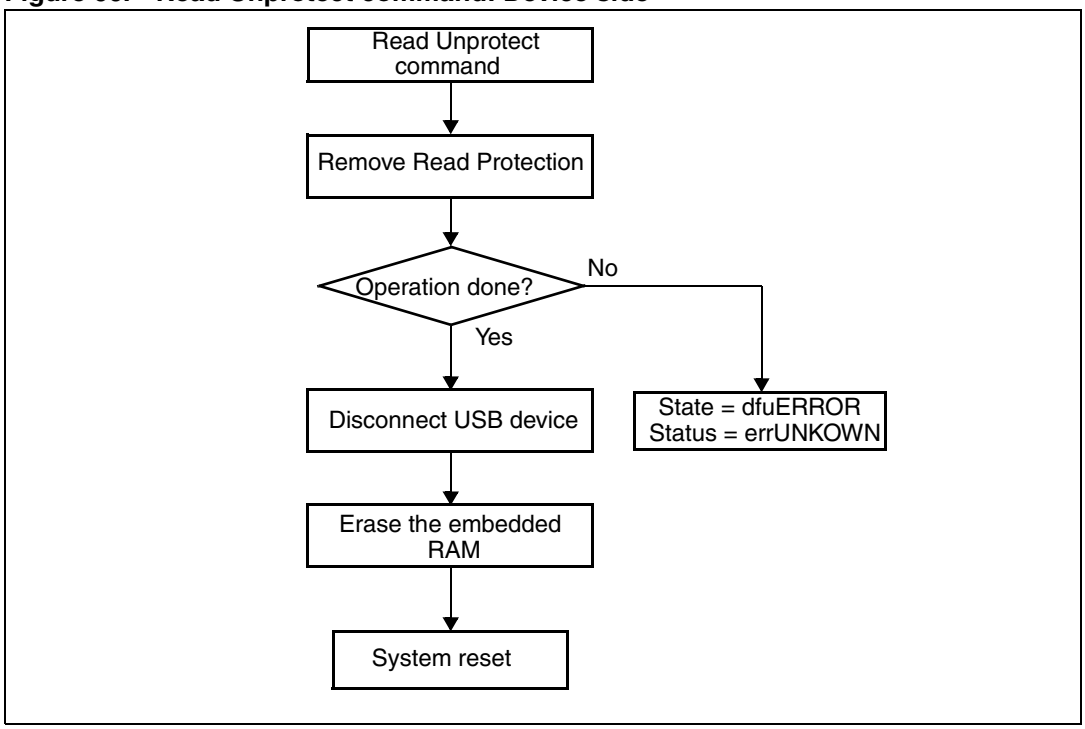
After sending a Read Unprotect command, the Host has to send a DFU_GETSTATUS request.

The Read Unprotect command is effectively executed only when a DFU_GETSTATUS request is issued by the Host. If the status returned by the device is other than dfuDNBUSY, then an error has occurred. After this operation, the device removes the Read Protection and, consequently, both Internal Flash and Embedded RAM are fully erased.

Hence, just after executing this command, the device disconnects itself and executes a System Reset. In this case, the device is not able to respond to a second Get Status request. And the Host may wait till the Device is enumerated again.

A second DFU_GETSTATUS request may also be issued (if the device is still connected) to check if the command has been correctly executed. If the device fails to execute the command it will return an error status (depending on the error type).

**Figure 60. Read Unprotect command: Device side**



### 4.5.5 Leave DFU mode

It is possible to exit DFU mode (and bootloader) and jump to a loaded application (in the internal Flash or in the embedded RAM) using the DFU download request.

The Host sends a DFU_DNLOAD request with 0 data length (no data stage after the request) in order to inform the device that it will have to exit DFU mode. The device acknowledges this request if the current state is dfuDNLOAD-IDLE or dfuIDLE.

The DFU Leave operation is effectively executed only when a DFU_GETSTATUS request is issued by the Host. If the status returned by the device is other than dfuMANIFEST, then an error has occurred. After this operation, the device performs the following:

- it disconnects itself
- it initializes the registers of the peripherals used by the bootloader (see *Table 2*) to their default reset values
- it initializes the user application's main stack pointer
- it jumps to the memory location programmed in the received 'address pointer + 4', which corresponds to the address of the application's reset handler
  For example if the received address is 0x0800 0000, the bootloader will jump to the memory location programmed at address 0x0800 0004.
  In general, the host should send the base address where the application to jump to is programmed.

The address pointer has to be set (using Set Address Pointer command) before launching the Leave DFU routine, otherwise, the bootloader will jump to the default address (internal Flash memory start address: 0x08000000).

The address pointer can also be set through the last Write Memory operation: if a download operation is performed, the address pointer used for this download will be stored and used later for the jump.

*Note: If the address pointer points to an address that doesn't contain executable code, then the device will be reset and, depending on the state of the boot pins, may re-enter bootloader mode.*

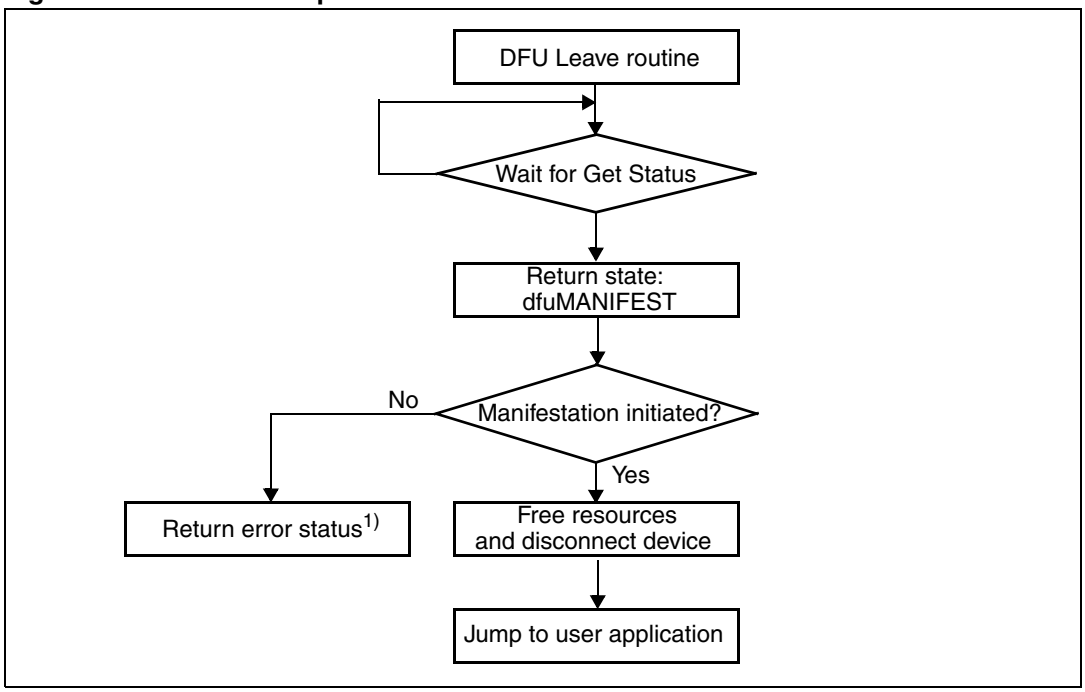
Since the Bootloader DFU application is not manifestation-tolerant, the device will not be able to respond to Host requests after a manifestation phase is completed.

A second DFU_GETSTATUS request may also be issued (if the device is still connected) to check if the command has been correctly executed. If the device fails to execute the command it will return an error status (depending on the error type).

*Note: 1 The Jump to application works only if the user application sets the vector table correctly to point to the application address*

*2 When performing a jump from the Bootloader to a loaded application code which uses the USB IP, the user application has to disable all pending USB interrupts and reset the core before enabling interrupts. Otherwise, a pending interrupt (issued from the bootloader code) may interfere with the user code and cause a functional failure. This procedure is not needed after exiting system memory boot mode.*

**Figure 61. Leave DFU operation: Device side**




1. This status depends on the error origin and the current status.

# 5 Description of the bootloader versions

The table below lists the bootloader versions and the changes between versions V1.0 and V2.0.

**Table 8. Bootloader versions**

| Bootloader version number | Description |
|---|---|
| V1.0 | Initial bootloader version. |
| V2.0 | – Bootloader detection mechanism updated to fix the issue when GPIOs of unused peripherals in this bootloader are connected to low level or left floating during the detection phase. For more details please refer to limitation 2.12 “Boot loader unavailability on STM32F105xx and STM32F107xx devices with a date code below 937” as described in Revision 2 of the “STM32F105xx and STM32F107xx revision Z” errata sheet available from www.st.com<br>– Vector table set to 0x1FFF B000 instead of 0x0000 0000<br>– Go command updated (for all bootloaders): USART1, USART2, CAN2, GPIOA, GPIOB, GPIOD and SysTick peripheral registers are set to their default reset values<br>– DFU bootloader: USB pending interrupt cleared before executing the Leave DFU command<br>– DFU subprotocol version changed from V1.0 to V1.2<br>– Bootloader version updated to V2.0 |

# 6 Revision history

**Table 9. Document revision history**

| Date | Revision | Changes |
|---|---|---|
| 08-Jul-2009 | 1 | Initial release. |
| 26-Oct-2009 | 2 | Text clarified in *Section 1.2: Bootloader activation* and *Table 2: STM32F105xx and STM32F107xx configuration in System memory boot mode* modified:<br>– “clock source” removed, replaced by “RCC” and “IWDG” in Common to all bootloaders row<br>– System memory start address corrected<br>– DFU bootloader rows modified<br>*Section 1.3: Hardware requirements*: bootloader connection conditions added.<br>*Section 1.4: Bootloader selection*: note added and *Figure 1: Bootloader selection* modified.<br>Go command description modified in *Table 3: USART bootloader commands* and *Table 4: CAN bootloader commands*.<br>*Figure 51: Readout Unprotect command via CAN: device side* modified.<br>*Section 2.10: Go command* specified and *Figure 12: Go command: device side* modified.<br>Byte 3 and byte 4 descriptions modified in *Section 2.12: Erase Memory command*.<br>PID specified throughout the document.<br>*Figure 26: Check HSE frequency* modified. *Section 3.9: Go command via CAN*: description clarified, notes added and *Figure 39: Go command via CAN: Device side* modified.<br>*Figure 43: Erase Memory command via CAN: Device side* modified.<br>*Section 4.1: Bootloader code sequence* modified.<br>*Table 7: DFU bootloader commands* added.<br>*Section 4.5.5: Leave DFU mode* specified and *Figure 61: Leave DFU operation: Device side* modified.<br>*Section 5: Description of the bootloader versions* added.<br>Small text changes. |