AN4656
Application note

Bootloading procedure for STLUX™ and STNRG™ digital controllers

Max Cortiana, Ambrogio D'Adda

Introduction

This specification contains the description of how to load a code on STLUX and where not differently specified also on STNRG devices through the bootloader embedded in the system memory of the device (the ROM memory). Through this firmware, the device memory can be erased and programmed using a standard communication interface.This code allows the memories, including the program, code and RAM, to be written into the device through the standard serial interface UART.

Following a step-by-step guided procedure, it is possible through the standard "Flash loader demonstrator" application released by STMicroelectronics to download an application code into the device and it is also possible to specifically configure STLUX features.

For further information on the STLUX family features, pinout, electrical characteristics, mechanical data and ordering information, please refer to the specific STLUX device datasheet.

Reference document

For hardware information on the STLUX controller and product specific SMED configuration, please refer to the STLUX and STNRG product datasheets.

Contents

List of tables

List of figures

1 Acronyms

A list of acronyms used in this document:

Table 1. List of acronyms

Acronym	Description
BL	Bootloader- used to load the user program without the emulator
DAC	Digital-to-analog converter
DALI	Digital addressable lighting interface
ECC	Error Correction Code
FSM	Finite state machine
FW	Firmware loaded and running on the CPU
GPIO	General purpose input/output
HSE	High-speed external crystal - ceramic resonator
HSI	High-speed internal RC oscillator
I2C	Inter-integrated circuit interface
IAP	In-application programming
ICP	In-circuit programming
ITC	Interrupt controller
IWDG	Independent watchdog
LSI	Low-speed Internal RC oscillator
MCU	Microprocessor central unit
MSC	Miscellaneous
PM	Power management
RFU	Reserved for future use
ROP	Read-out protection
RST	Reset control unit
RTC	Real-time clock
SMED	State machine event driven
STMR	System timer
SW	Software, is the firmware loaded and running on the CPU (synonymous of FW)
SWIM	Single-wire interface module
UART	Universal asynchronous receiver/transmitter
WWDG	Window watchdog

2 Description

The bootloading can be operated either with a user developed downloader application or with the “Flash loader demonstrator” (loader) available from STMicroelectronics.

Before trying to load your code through the UART interface using the bootloader, a series of checks should be performed (checks are not needed if you use a brand new STLUX device).

The bootloader is able to accept connections on different configuration of the UART device. By default the devices are delivered from the factory configured to enable the bootloader on all the available channels.This means that during the initial synchronization phase the bootloader performs polling for the synchronization character on each channel for 1/Nth of the available time where *N* is the number of enabled channels. In this condition it is possible that when the external loader sends the synchronization character to the physically connected channel, the bootloader is checking one of the other channels. As a consequence the bootloader does not detect the synchronization character and then the external loader signals a missing connection.

3 Bootloader protocol interface

3.1 Peripherals settings

This section describes the hardware setting of the communication peripherals:

The UART setting:

- Data frame: 1 start bit, 8 data bit, even parity bit, 1 stop bit
- Automatic speed detection - min.: 4800 bps - max.: 460.8 kbps

Mandatory: in order to perform the automatic speed detection the RX lines shall be stable in the application board. To speed-up the automatic speed detection connect the not used RX UART line to GND or pull-down with the 10 KΩ resistor.

Note: All communication is verified by:
Checksum: all received bytes are XORed. A byte containing the computed XOR of all previous bytes is added as the last byte of the data field (checksum byte). By XORing all the received bytes, data + checksum, the result, at the end of the packet, must be 0x00;

According to the protocol established between the Host and BL (see *Section 3.2* to *Section 3.2.6*) there will be a byte accepted - ACK answer, or discarded - NACK answer.

ACK = 0x79

NACK = 0x1F

SYNCHR = 0x7F

3.2 Commands description

The supported commands are listed in *Table 2*:

Table 2. Allowed commands

Command	Command code	Command description
GET	0x00	Gets the version and the allowed commands supported by the current version of the BL.
Read Memory	0x11	Reads maximum 256 bytes of memory starting from an address specified by the Host.
GO	0x21	Jumps to an address specified by the Host to execute a loaded code.
Write Memory	0x31	Writes maximum 128 bytes to the RAM or the EEPROM starting from an address specified by the Host.
Erase Memory	0x43	Erases from one to all the EEPROM sectors.

3.2.1 Synchronization

Figure 1. Synchronization: Host side

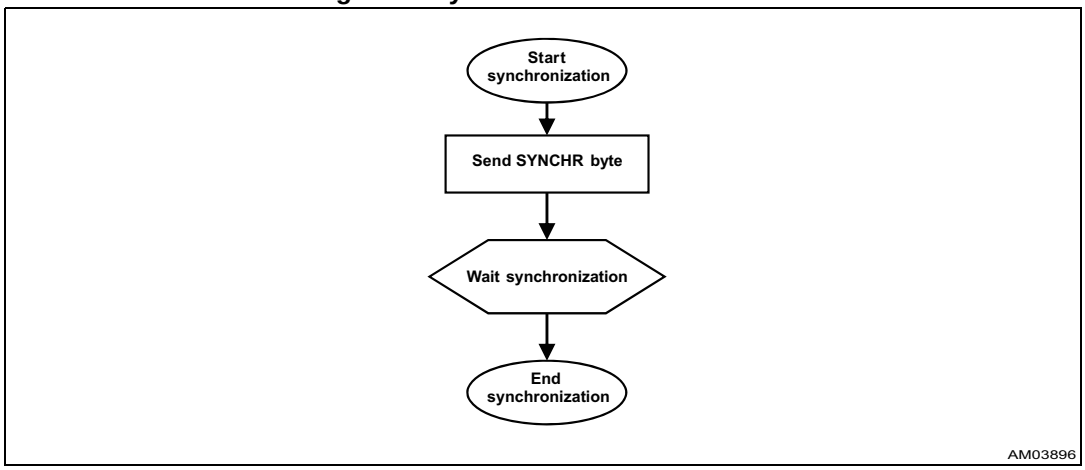
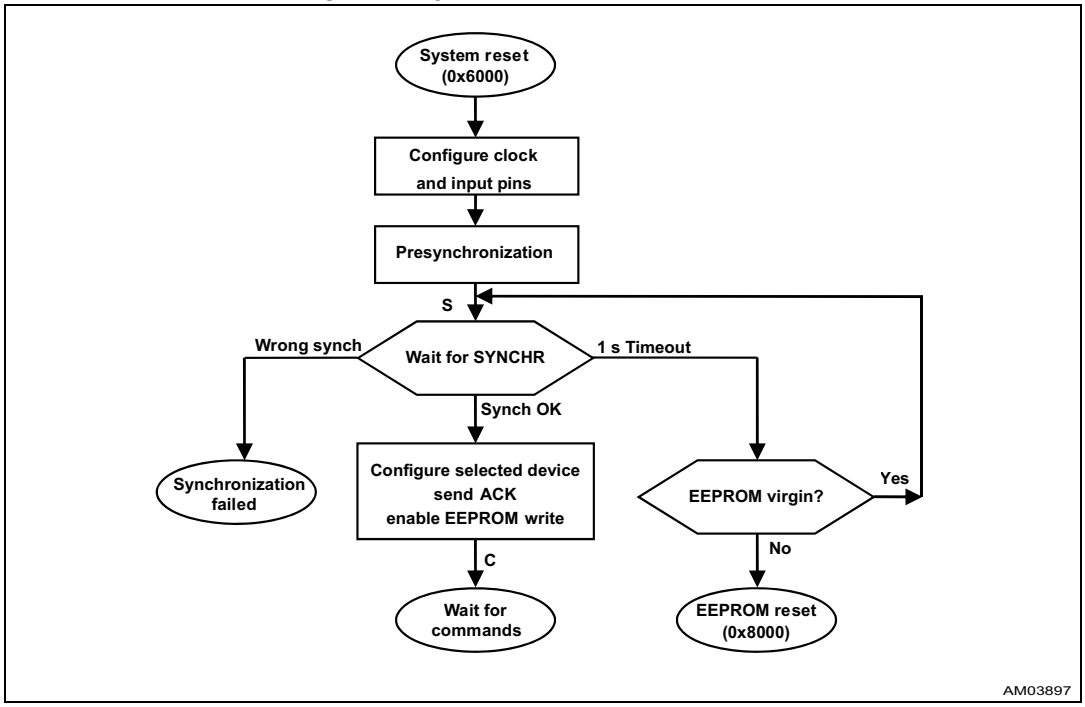


Figure 2. Synchronization: STLUX side



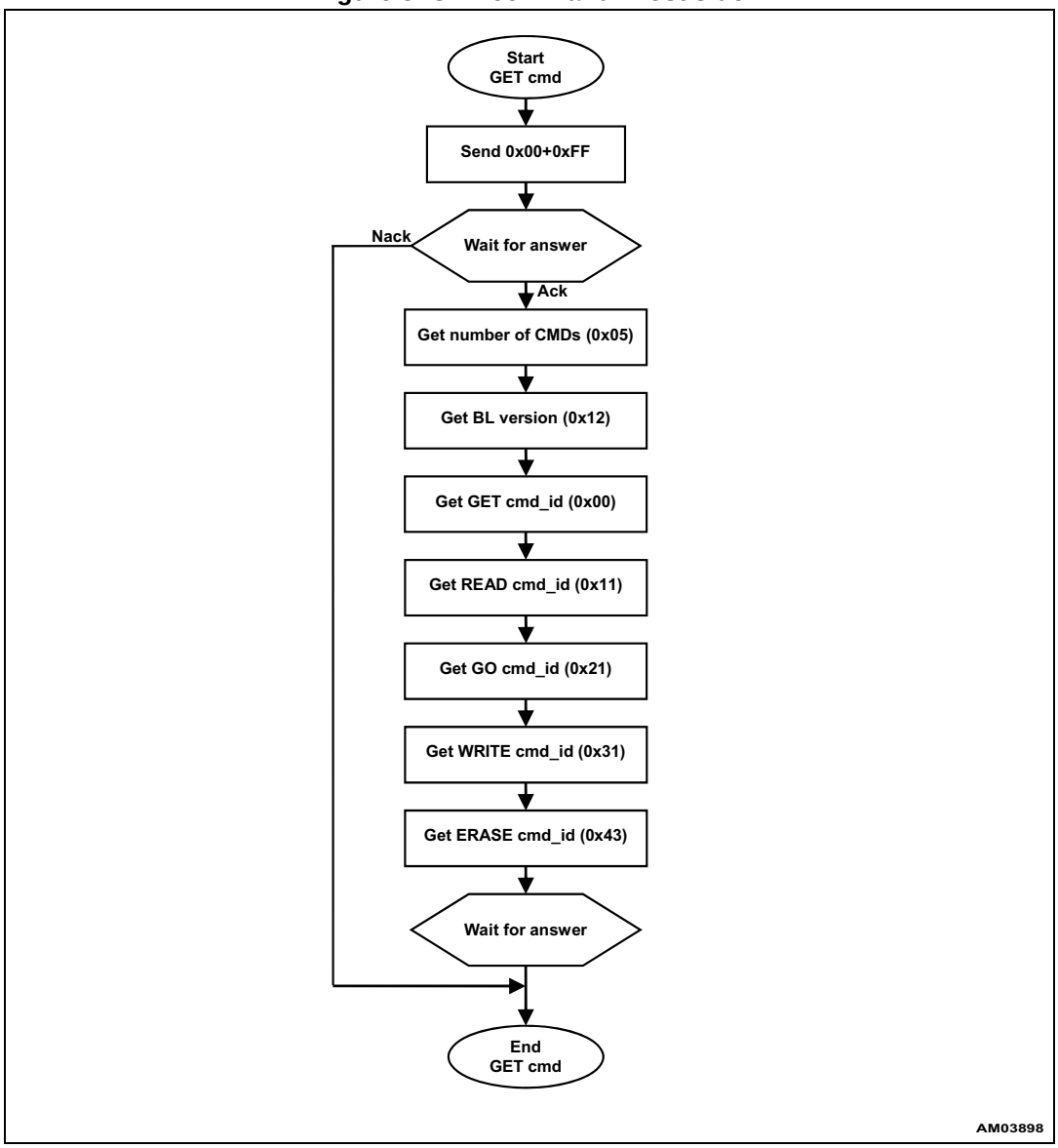
The Host sends the SYNCHR byte as follows:

UART
SYNCHR byte: 0x7F

3.2.2 GET command

By this command the Host gets the version of the BL and the supported commands.

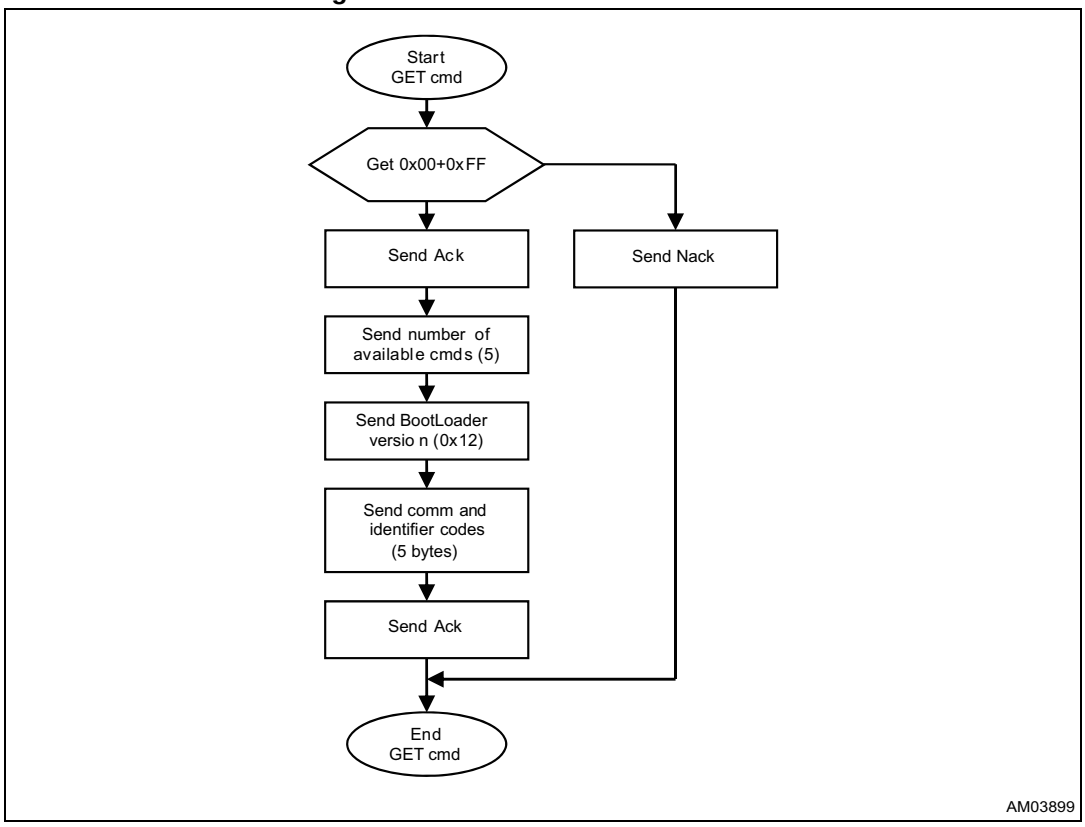
Figure 3. GET command: Host side



The Host sends the bytes as follows:

UART
Byte 1: 0x00 - command ID **Byte 2:** 0xFF - complement

Figure 4. GET command: STLUX side



The STLUX sends the bytes as follows:

Byte 1: ACK

Byte 2: N = 5 = the number of bytes to be sent -1 (1 ≤ N +1 ≤ 256)

Byte 3: version of the BL (0 < Version ≤ 255)

Byte 4: 0x00 - GET command

Byte 5: 0x11 - Read Memory command

Byte 6: 0x21 - GO command

Byte 7: 0x31 - Write Memory command

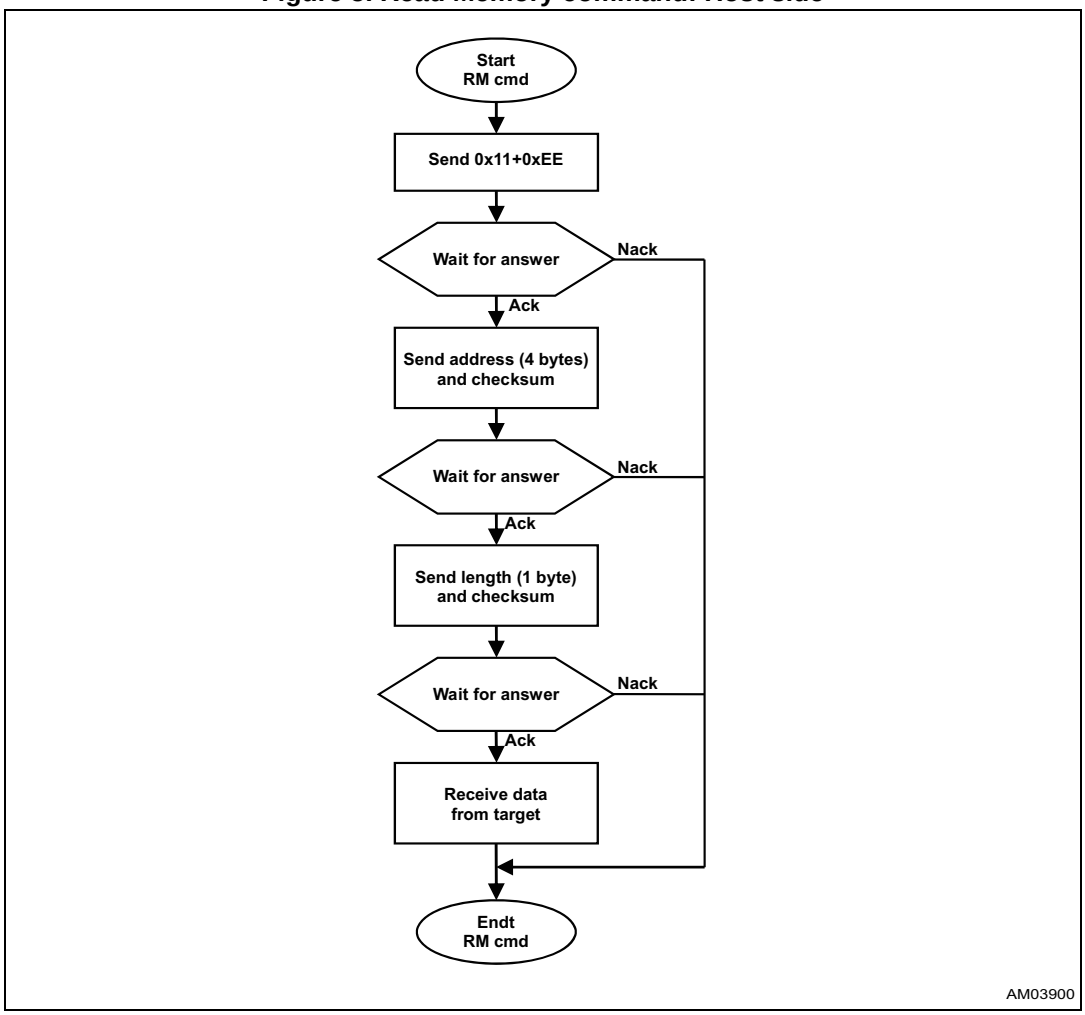
Byte 8: 0x43 - Erase Memory command

Byte 9: ACK

3.2.3 Read Memory command

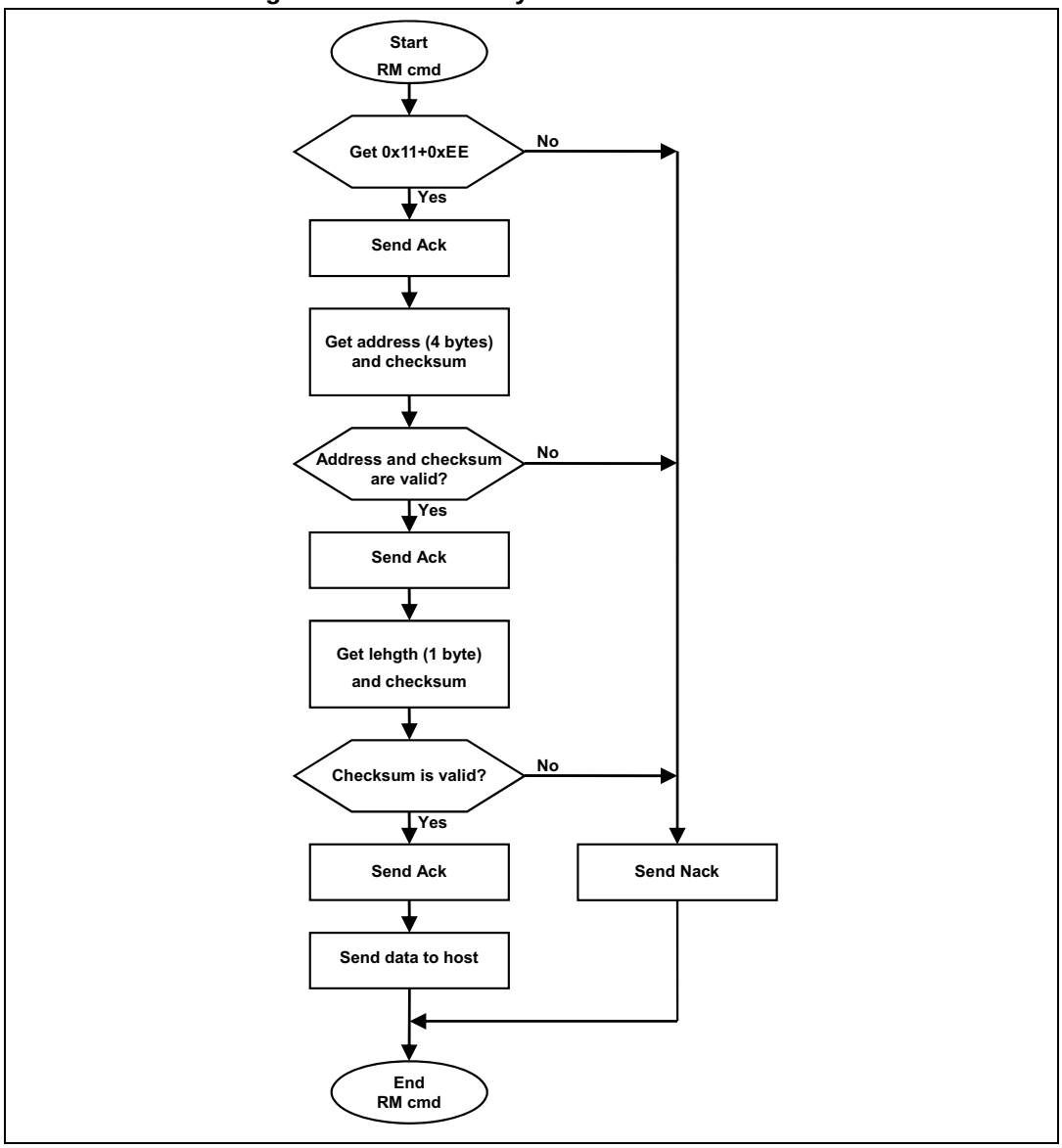
This command allows reading from the memory (RAM, EEPROM or registers). When the BL receives a Read Memory command it answers by an ACK byte and then waits for an address (4 bytes; the 1st received is the MSB one) and a checksum byte. The BL checks this address: if it is a valid address and the checksum is OK then the BL sends an ACK byte otherwise sends a NACK byte and ends the command. When the address is valid and the checksum is correct the BL waits for the number of bytes to be transmitted minus 1 (N) and for its complemented byte (checksum of N); if the checksum is correct then the BL sends to the Host the requested data [(N+1) bytes] starting from the received address, otherwise sends an NACK before ending the command.

Figure 5. Read Memory command: Host side



Note: Table 5 on page 24 shows the valid addresses. If the BL receives a not valid address an ADD_ERROR occurs (see Table 4 on page 24).

Figure 6. Read Memory command: STLUX side



The Host sends the bytes as follows:

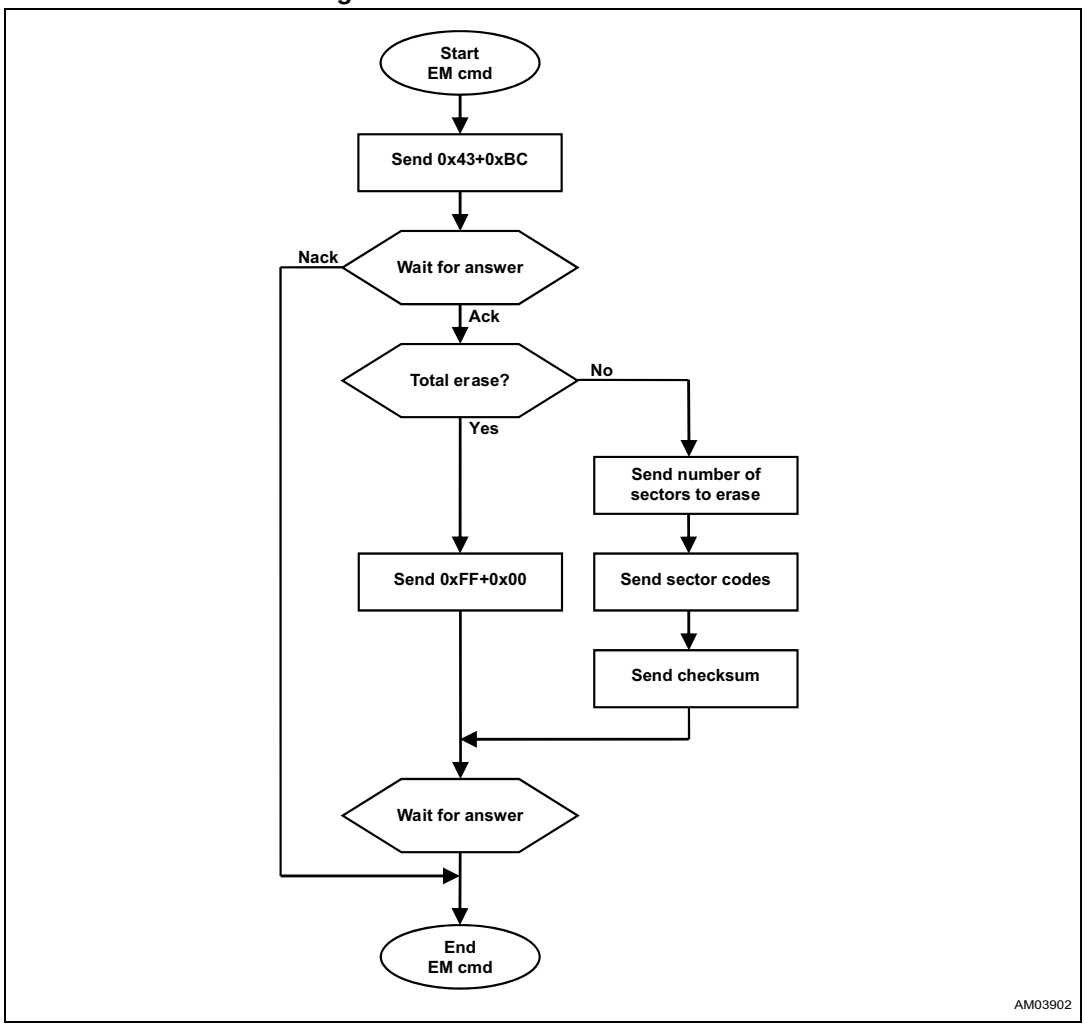
UART
Byte 1: 0x11 - command ID
Byte 2: 0xEE - complement
Byte3 to Byte 6: the start address
Byte 3: MSB
Byte 6: LSB
Byte 7: checksum: XOR (Byte 3, Byte 4, Byte 5, Byte 6)
Byte 8: the number of bytes to be read - 1 (0 < N ≤ 255)
Byte 9: checksum: NOT Byte 8

3.2.4 Erase Memory command

The Erase command allows erasing the EEPROM memory sector by sector. When the BL receives an Erase command, it answers by an ACK byte and then waits for the number of sectors to erase (one byte), the sector codes and a checksum byte; if the checksum is correct then the BL erases the memory and sends an ACK byte to the Host, otherwise sends a NACK byte to the Host and ends the command. Some details:

1. *N* is the number of sectors to erase: 0 ≤ N ≤ 32.
2. The TBL receives N+1 bytes (see *Table 3*).

Figure 7. Erase command: Host side

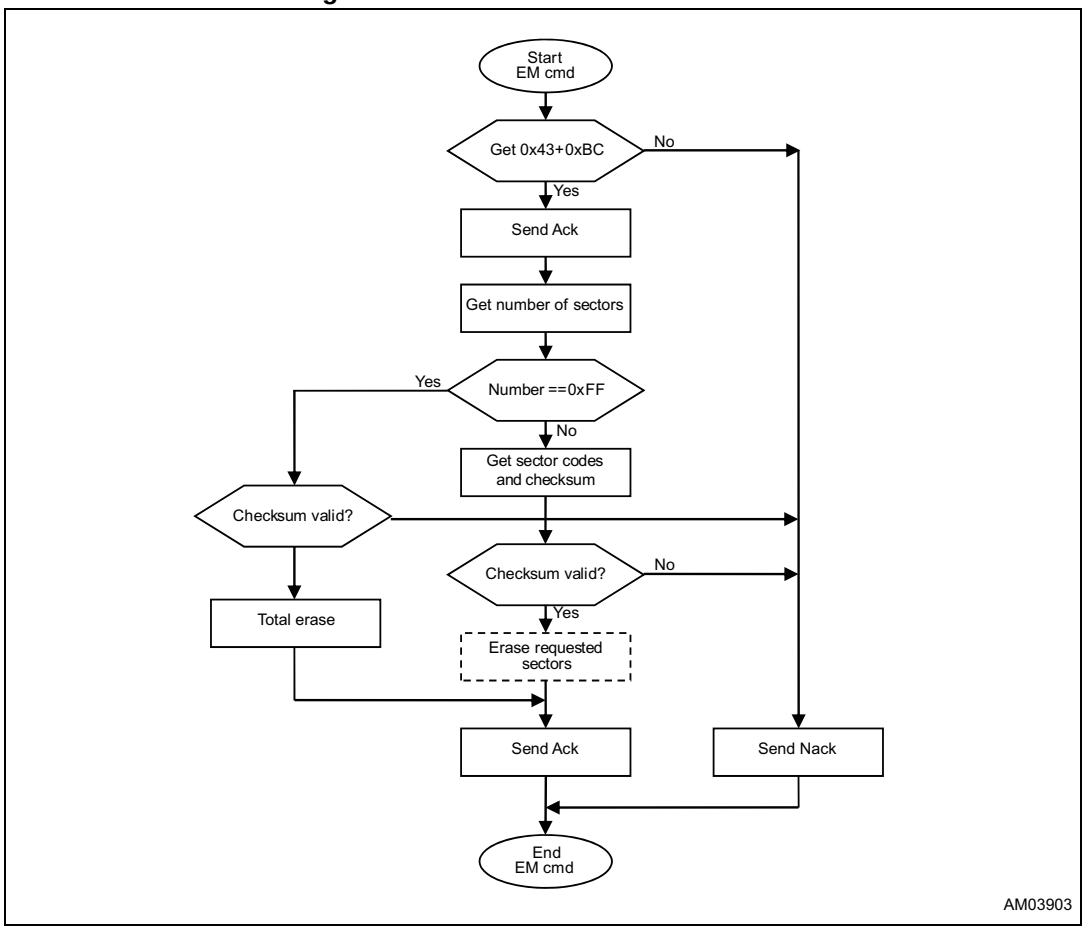


Note: 1. The "Total Erase" erases PROGRAM EEPROM and DATA EEPROM (33 KB). The BL erases the memory sector by sector and not by a "Global Erase operation" because it doesn't work in user mode.

2. A sector is 1 kbyte; therefore the granularity with the Erase command is 8 blocks.

Warning: **If the Host sends a sector code not allowed (see *Table 3*) the command fails, therefore also the correct sector code will be ignored.**

Figure 8. Erase command: STLUX side



The Host sends the bytes as follows:

UART
Byte 1: 0x43 - command ID
Byte 2: 0xBC - complement
Byte 3: 0xFF or number of sectors to erase (0 ≤ N ≤ 32); if N > 32 a CMD_ERROR occurs.
Byte 4 or N+1 Bytes: 0x00 or [N+1 bytes and then checksum: XOR (N, N+1 bytes)]

Table 3. Sector codes

Sector code	EEPROM Addr<15:0>
0x00	8000h → 83FFh
0x01	8400h → 87FFh
0x02	8800h → 8BFFh
0x03	8C00h → 8FFFh
0x04	9000h → 93FFh
0x05	9400h → 97FFh
0x06	9800h → 9BFFh
0x07	9C00h → 9FFFh
0x08	A000h → A3FFh
0x09	A400h → A7FFh
0x0A	A800h → ABFFh
0x0B	AC00h → AFFFh
0x0C	B000h → B3FFh
0x0D	B400h → B7FFh
0x0E	B800h → BBFFh
0x0F	BC00h → BFFFh
0x10	C000h → C3FFh
0x11	C400h → C7FFh
0x12	C800h → CBFFh
0x13	CC00h → CFFFh
0x14	D000h → D3FFh
0x15	D400h → D7FFh
0x16	D800h → DBFFh
0x17	DC00h → DFFFh
0x18	E000h → E3FFh
0x19	E400h → E7FFh
0x1A	E800h → EBFFh
0x1B	EC00h → EFFFh
0x1C	F000h → F3FFh
0x1D	F400h → F7FFh
0x1E	F800h → FBFFh
0x1F	FC00h → FFFFh
0x20	4000h → 43FFh

3.2.5 Write Memory command

This command allows writing the memory (RAM, EEPROM or registers). When the BL receives a Write Memory command, it sends an ACK to the Host and then waits for an address (4 bytes; the 1st received is the MSB one) and a checksum byte. The BL checks this address: if it is a valid address and the checksum is OK, the BL sends an ACK byte otherwise it sends a NACK byte and ends the command. If the address is valid and the checksum is OK, the BL will receive the following next bytes in sequence:

- *N* (one byte), which contains the number of data bytes to be received minus 1
- *N+1* data bytes and the checksum (XOR of “N” and N+1 data bytes).

The incoming data are always written in the RAM before being loaded in the final location.

At this point, the BL:

- Checks whether the Host wants to write in the RAM or in EEPROM
- Programs the Host data to the memory starting from the received address
- Reads the programmed data and calculates the checksum in order to check if the programming operation was successful.

Finally, at the end of the command, the BL sends the ACK byte if the write operation is completed successfully otherwise sends a NACK byte and ends the command.

Even if the Host receives an NACK, and IF THIS ERROR IS NOT DUE TO N > 127, the memory is being programmed with “something”, BUT in any case it is necessary to reprogramm it because there has been an error during the transmission or programming. If the error is due to N > 127 the memory is not programmed.

The Host can send a Write command with at most 128 data bytes (N = 127). In order to write the data in the EEPROM memory locations, the BL can performs two different write operations:

- **WordWrite:** writes a word in the EEPROM. It is used when the bytes number (N+1) sent from the Host is less than 128, in this case the BL will perform this operation N+1 times.

Note: Even if the BL writes a byte the hardware write operation executed is a WordWrite.

- **BlockWrite:** writes a block in the EEPROM. It is used when the bytes number (N+1) sent from the Host is 128 AND the destination address is an integer module of 128, in other words, in order to use this operation the block sent from the Host shall be aligned with a memory block.

Figure 9. Write Memory command: Host side

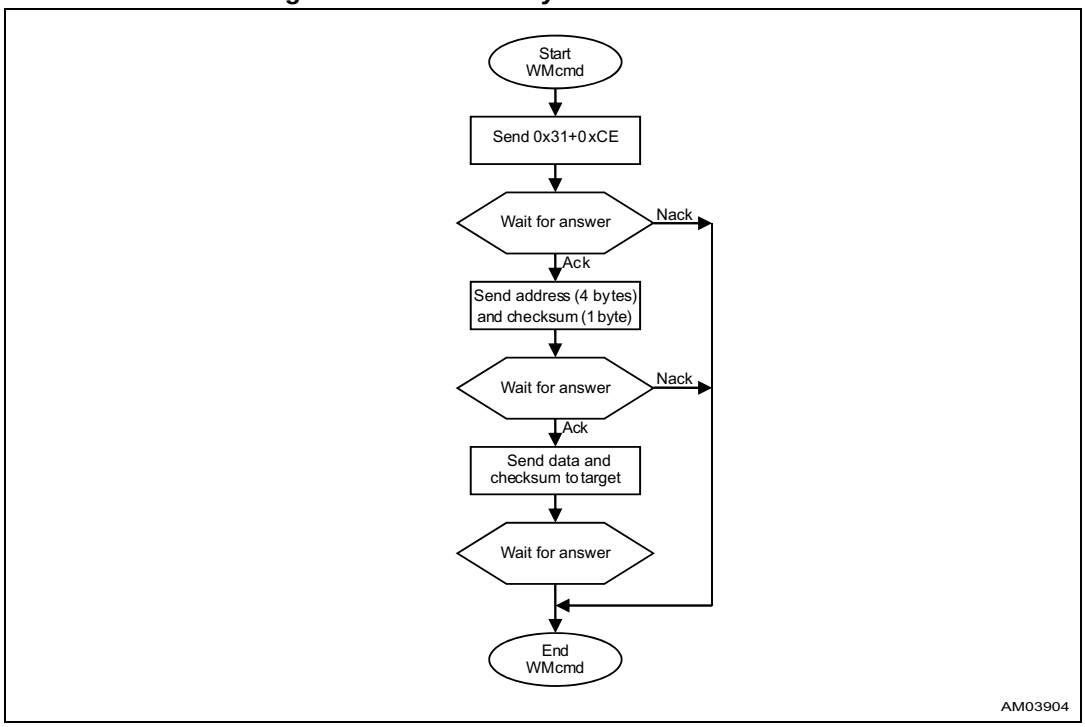
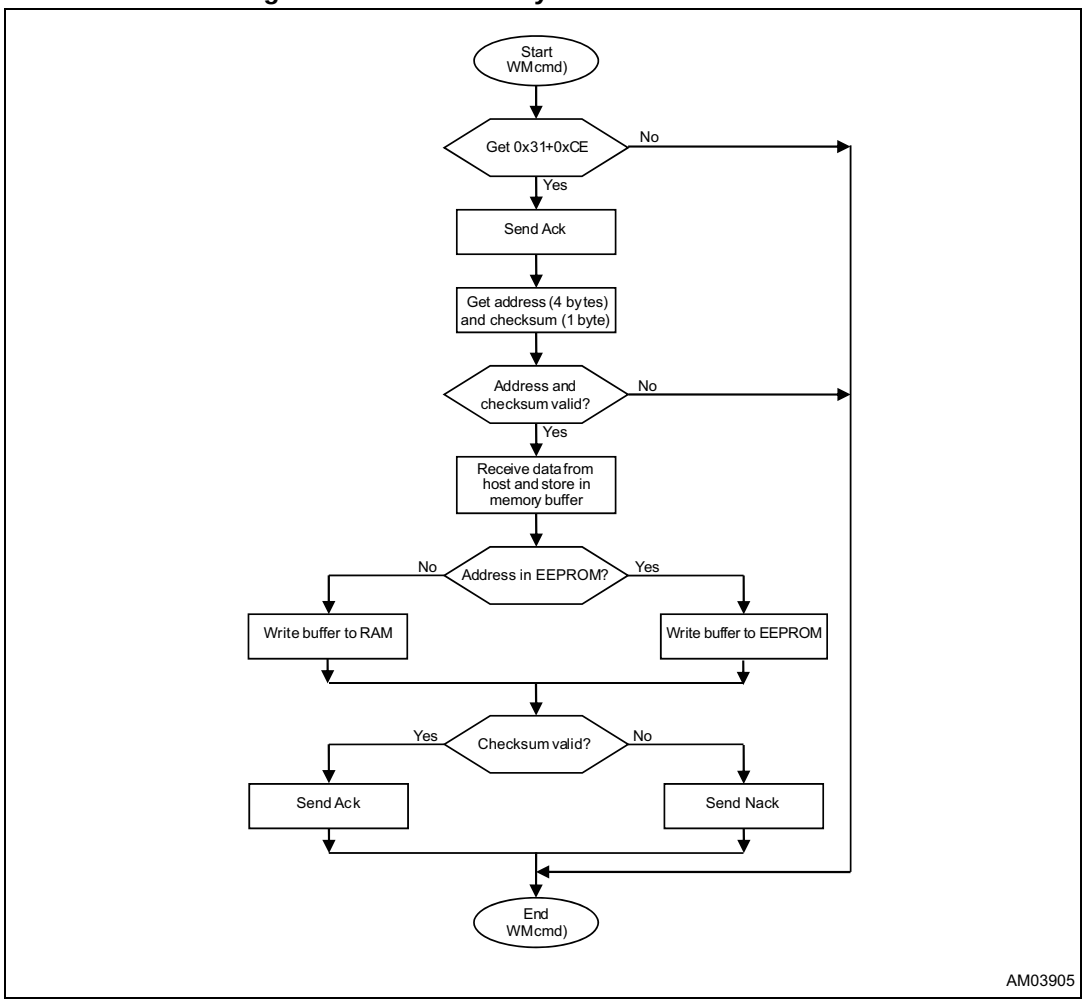


Figure 10. Write Memory command: STLUX side



The Host sends the bytes as follows:

UART
Byte 1: 0x31 - command ID
Byte 2: 0xCE - complement
Byte 3 to Byte 6: the start address
Byte 4: MSB
Byte 7: LSB
Byte 7: checksum: XOR (Byte 3, Byte 4, Byte 5, Byte 6)
Byte 8: Bytes number to receive (0 ≤ N ≤ 127); if N > 127 a CMD_ERROR occurs in the BL.
N+1 data bytes: (max. 128 bytes)
Checksum byte: XOR (N, N+1 data bytes)

Note: 1. Table 5 on page 24 shows the valid addresses. If the BL receives a not valid address an ADD_ERROR occurs (see Table 4 on page 24).

2. In order to download an application into the EEPROM the Host shall send more consecutive "Write commands". The only way to accomplish fast writing is to send as much as possible "Write commands" aligned to the EEPROM data block.

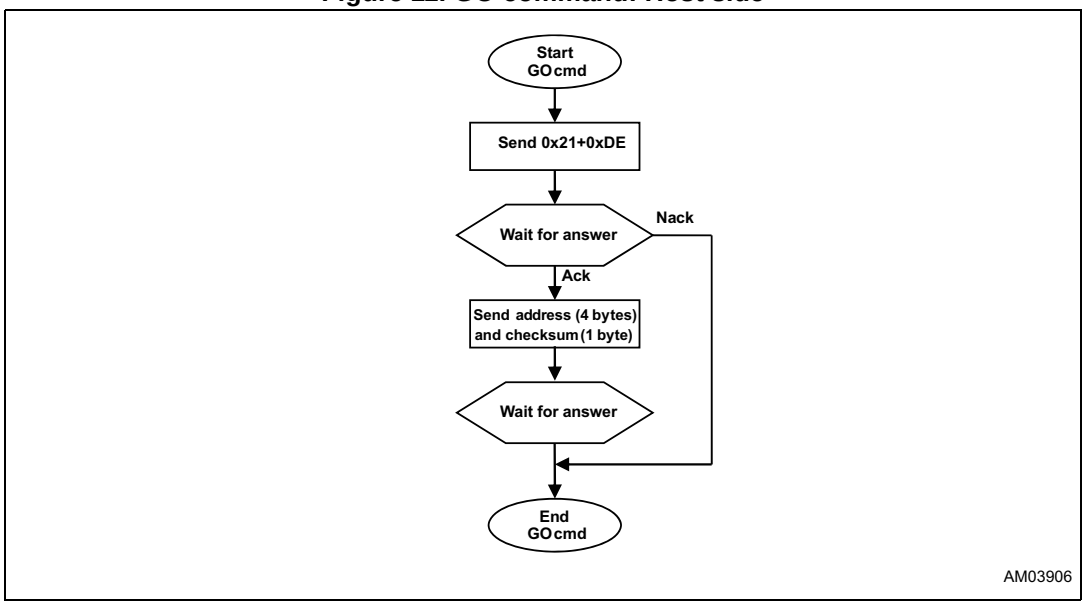
As a reference value, to program the full 32 kByte EEPROM memory with the BL connected at 115200 bps transferring data blocks of 128 bytes aligned to memory sectors, the following timing should apply:

- 128 data bytes + 10 overhead protocol bytes = 138 bytes corresponding to 1518 bits
- 32 kbyte / 128 bytes = 256 data packets
- 1518 x 256 / 115200 = 3.373 s minimum transfer time
- 256 x 3.5 ms = 0.896 s programming time (with typical block programming time)
- 3.373 + 0.896 = 4.269 s total memory programming time (typical value).

3.2.6 GO command

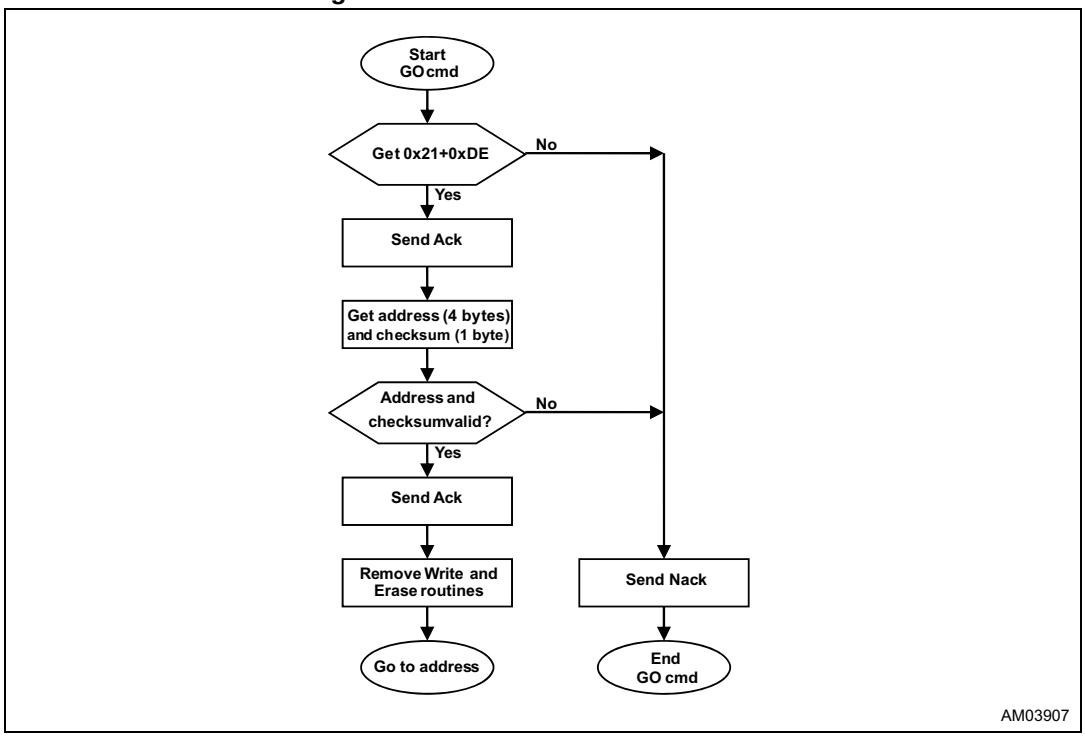
This command allows running the application downloaded in the EEPROM or any other code by making a branch to an address specified by the Host. When the BL receives a GO command, it answers by an ACK byte and then waits for an address (4 bytes; the 1st byte received is the MSB one) and a checksum byte; if it is a valid address and the checksum is OK, the BL sends an ACK byte otherwise it sends a NACK byte and ends the command. When the address is valid and the checksum is OK, the BL removes the Erase and Write routines from the RAM and then the program counter of the CPU jumps automatically to this address.

Figure 11. GO command: Host side



Note: Table 5 on page 24 shows the valid addresses. If the BL receives a not valid address an ADD_ERROR occurs (see Table 4 on page 24).

Figure 12. GO command: STLUX side



The Host sends the bytes as follows:

UART
Byte 1: 0x21 - command ID
Byte 2: 0xDE - complement
Byte 3 to Byte 6: the start address
Byte 3: MSB
Byte 6: LSB
Byte 7: checksum: XOR (Byte 3, Byte 4, Byte 5, Byte 6)

4 Memory model

The STLUX microcontroller has:

- 2-kByte RAM split to:
  - Short addressing zero page, 256 Bytes
  - 16-bit addressing, 1.25 kbytes
  - Stack, 512 Bytes
  - (16-bit addressing, 2048 Bytes)
- 1-kByte data EEPROM
- 2-kByte boot ROM
- 32-kByte EEPROM split to:
  - 32x4 Bytes interrupt vector
  - All remaining Host programming area

For more information see the STLUX family memory map in the specific device datasheet.

The STNRG microcontroller has:

- 6-kByte RAM split to:
  - Short addressing zero page 256 Bytes
  - 16-bit addressing, 1.25 kBytes
  - Stack, 512 Bytes
  - (16-bit addressing, 6144 Bytes)
- 1-kByte data EEPROM
- 2-kByte boot ROM
- 32-kByte EEPROM split to:
  - 32x4 bytes interrupt vector
  - All remaining Host programming area

For more information see the STNRG family memory map in the specific device datasheet.

5 Software model

The boot code can download up to 128 byte at a time. The first 130 bytes of the RAM (from 0x00) will be used to store the data coming from the serial interface, thus allowing the boot using the stack. Moreover, other 26 bytes are used by the BL as temporary variables.

The RAM memory contains the Erase routine starting from 0x00A0 and the Write routine starting from 0x0150; total memory space allocated for both routines is 304 bytes.

The usage of the stack is limited to less than 16 bytes for internal function calls with the maximum nesting of three levels.

The Boot code does not use in any case interrupt functions, and all the internal devices are handled in the polling mode.

Resuming, the RAM memory allocated by the BL is from 0x0000 to 0x01CF, plus 16 bytes allocated before the default address of the stack pointer (0x07FF).

Note: 1. The peripheral (UART) used during the boot phase remains in power on when the user leaves the boot to execute an application.

2. The ROM part not used from the bootloader (“empty”) shall be filled by 0x71 in order to avoid that the BL falls in an infinite loop without any reset if it jumps into these “empty” locations.

6 Timing

This section reports some information about the timing of the bootloader. In order to use correctly the bootloader is necessary to respect some temporal intervals as described in following paragraphs.

After the hardware reset the bootloader goes in an initialization phase before going into the synchronization phase. Therefore the user shall wait a time of at least 10 ms before sending a synchronization message.

If the EEPROM memory has been at least once already programmed and the user wants to reprogram it (see *Table 6 on page 27*) then he shall send the synchronization message within 1 second from the hardware reset.

After a GO command the bootloader removes the Erase and Write routines from the RAM memory before sending to the Host address. The time necessary to remove these routines is about 150 µs.

7 Error management

Error management

Table 4. Errors

ERROR	Description	Actions
CMD_ERROR	If a denied command is received. If an parity error occurs during its transmission. If an error occurs during its execution.	Sends NACK and goes back to command checking.
ADD_ERROR	If a received command contains a denied destination address (see *Table 5*).	Sends NACK and goes back to command checking.

Table 5. Valid addresses

Device	Memory	Address <15:0>
32k	RAM	0000h → 07FFh (0FFFh or 017FFh)
	DATA EEPROM	4000h → 43FFh
	OPTION BYTES	4800h → 487Fh
	PERIPHERAL REGISTERs	5000h → 57FFh
	PROGRAM EEPROM	8000h → FFFFh

Note: Table 5 depends on the specific device of the STLUX family. Please refer to the specific device datasheet.

8 Erase/Write EEPROM routines in RAM

There are some routines or a part of them that shall be downloaded into the RAM. They are:

- Erase routine
- Write EEPROM routine

The Erase routine shall be loaded into the RAM starting from 0xA0 whereas the Write EEPROM routine is starting from 0x150. They are contiguous to the RAM.

The user can download them by Write commands in the RAM.

The routines are contained in an *.s19 file (E_W_ROUTINEs_STLux_ver_1.2.s19).

9 How to bootload your code to a STLUX device

9.1 Introduction

As previously said, the bootloader is stored into the internal 2 Kbytes boot ROM memory and its main task is to download the application program into the internal program memory through the UART peripheral. Data are provided by any device (Host) that can send information through the serial interface. To avoid system locks due to application execution errors (e.g.: the application jumps erroneously into the BL code), all the unused ROM memory is padded with the hexadecimal value 0x71 that corresponds to an illegal opcode.

STLUX devices have a single UART peripheral which configuration may be switched to different IO lines depending on the application requirements. The bootloader (BL) may be configured either to use a specific UART configuration or to autodetect it.

The bootloading procedure is enabled by setting the option bytes OPTBL / nOPTBL corresponding to the address 0x487E / 0x487F as specified in *Table 6*.

9.2 Bootloading in AutoDetect mode

The default configuration for the BL is the AutoDetect mode. In this mode, after the reset the BL performs a polling in order to detect which channel is connected to a boot source via the UART.

The AutoDetect mode automatically polls all the available UART channels on STLUX or STNRG family devices. The polling procedure uses 4 mS to check one RX line, so when checking one line, the others are not considered. For this reason, when all serial lines are checked (as the default), the first synchronization is problematic.

Note: To speed-up the automatic speed detection, connect to GND or pull-down the not used RX UART line with the 10 KΩ resistor. This disables the bootloading check on this RX UART line.

9.3 Bootloading with ST Flash loader demonstrator

Since the Flash loader demonstrator doesn't handle this AutoDetect mode, the bootloader should be configured to check only the desired channel.

9.3.1 Configuring the desired UART channel

Setting the option bytes

To properly configure the UART boot source, the MSC_OPT0 and nMSC_OPT0 option bytes must be modified so to indicate the proper UART source to be scanned during the bootloading procedure. For the option bytes configuration, please refer to STLUX product datasheets.

Writing the option bytes can be performed via the SWIM building a simple Raisonance project with the declaration.

- At 0x4815 CONST const. u16 MSC_OPT0 = 0x11EE;
  this indicates the UART channel is OP0(0, 1) and the bootloader must check for this channel only as a boot source.

The same thing can be performed via the SWIM with the IAR toolset - new release which allows directly to handle the option bytes content.

9.3.2 Checking the memory content

In order to enable the bootloader, the Read-out protection for the EEPROM must be disabled. So the address 0x4800h must be set to its default value 0x00h (brand new devices are configured like that). If so, the loader can check the EEPROM content of the 0x8000h address as specified below and starts checking for boot sources according to the UART channel configuration.

Table 6. Initial checking

Checks	EEPROM location 0x8000	EE check Opt_byte 0x487E	EE check Opt_byteN 0x487F	Actual EEPROM status → action
1st	Not 0x82 and not 0xAC	Don't care	Don't care	EE virgin → jump to BL
2nd	0x82 or 0xAC	0x55	0xAA	EE programmed booting allowed → jump to BL
3rd	0x82 or 0xAC	Not 0x55	Not 0xAA	EE programmed booting not allowed → jump to EEPROM reset

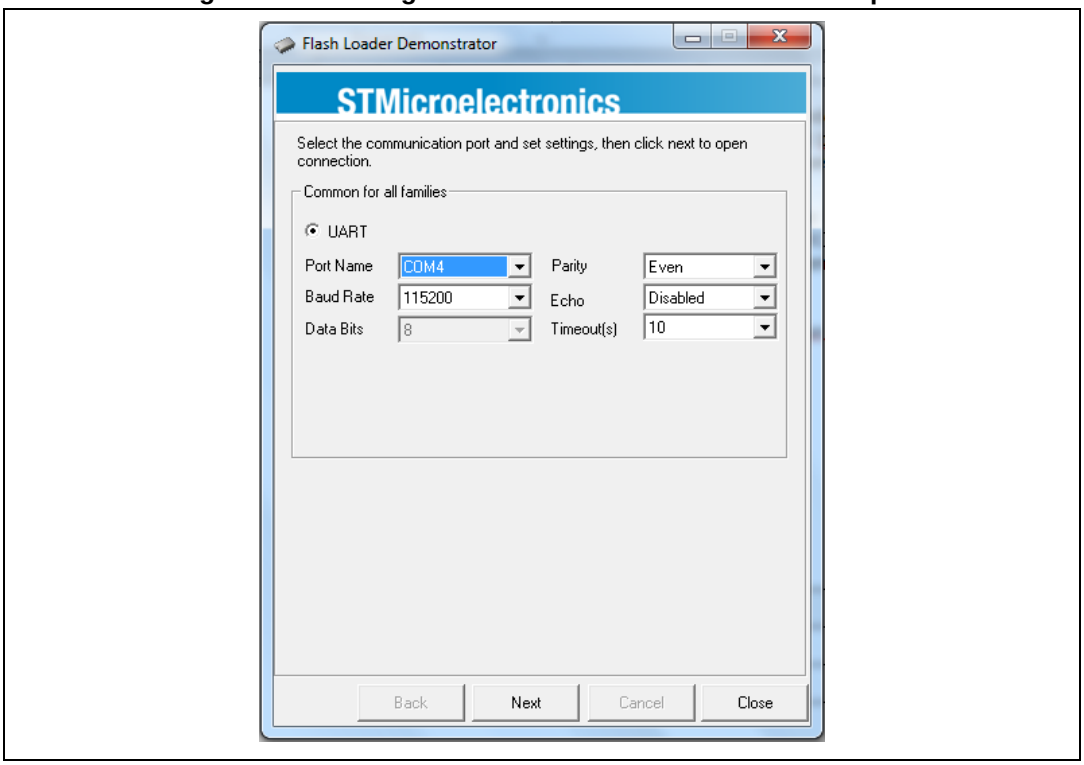
If the EEPROM is virgin, then the bootloader waits for an indefinite time for a connection on the set UART channel. If the EEPROM is programmed and the booling is allowed, the bootloader waits for one second checking for a connection on the set UART channel, then jumps to the code stored in the EEPROM.

So basically using a brand new STLUX device with the MSC_OPT0 option byte properly configured to a single UART channel and a virgin EEPROM allows to easily download your code through the UART connection with the Flash loader demonstrator.

9.3.3 Running the Flash loader demonstrator

Now you are ready to connect your PC with your STLUX device and run the Flash loader demonstrator. The program Shell will appear.

Figure 13. Running the Flash loader demonstrator - step 1



Select the right COM port for the connection and push the NEXT button. Then a new menu will appear.

Figure 14. Running the Flash loader demonstrator - step 2

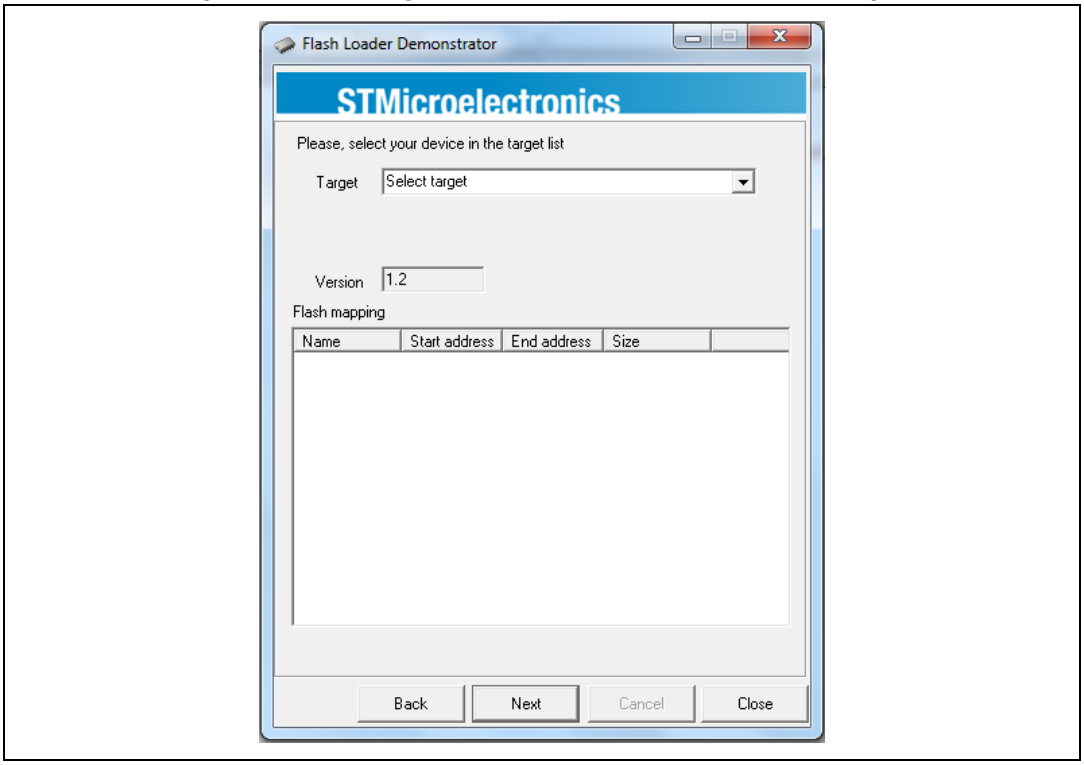
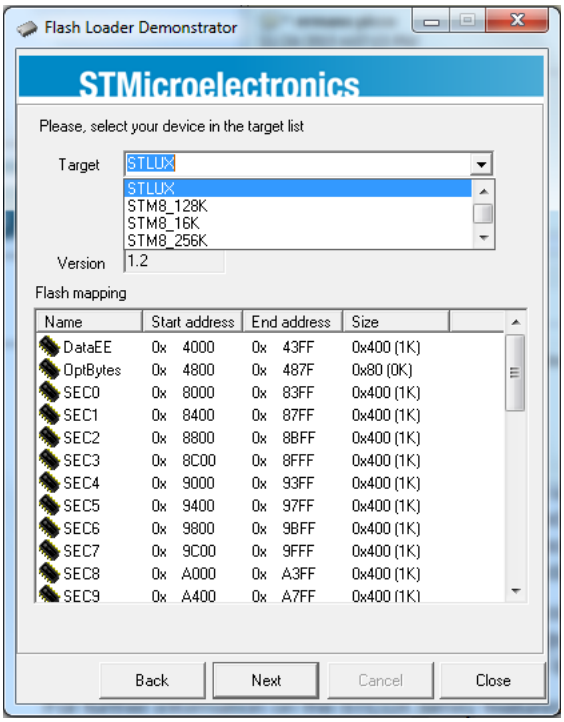
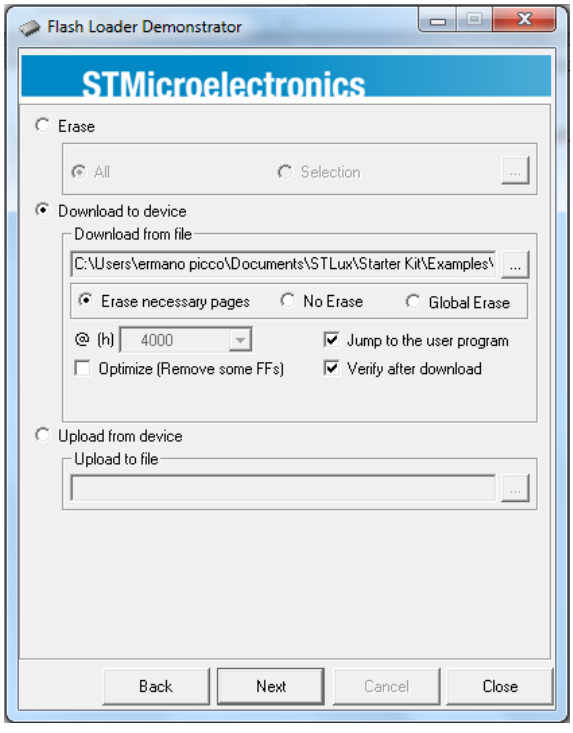


Figure 15. Running the Flash loader demonstrator - step 3



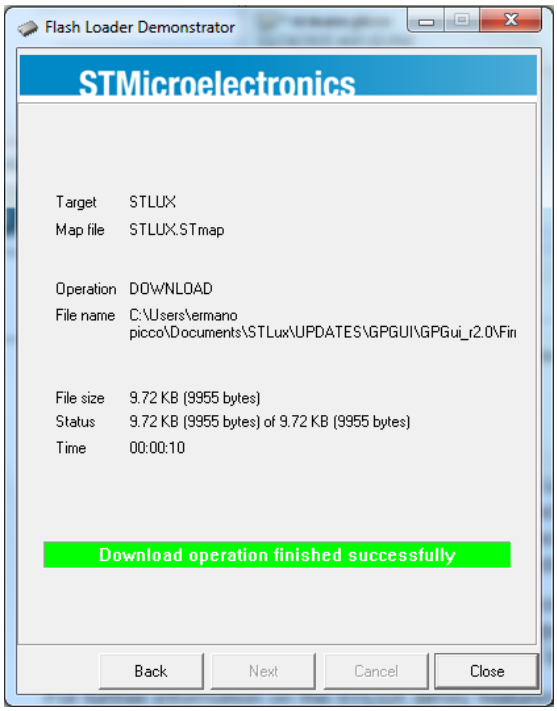
Select as a target device for the connection the STLUX or STNRG, then press NEXT to enter the next step.

Figure 16. Running the Flash loader demonstrator - step 4



Now to download your code, you need to choose “Download to device” and to give the complete path and name of the file you want to download to the STLUX. The acceptable file formats are *.bin and *.hex. When you set the proper file indications, press NEXT to start downloading the code. You can follow the downloading process reading the bar in the next shell. When the download will be complete, the Flash loader demonstrator will highlight it.

Figure 17. Running the Flash loader demonstrator - step 5



10 Revision history

Table 7. Document revision history

Date	Revision	Changes
15-Jun-2015	1	Initial release.
04-Dec-2015	2	Updated the main title, added STNRG device to the main title, *Section : Introduction on page 1* and *Section 9.3.3: Running the Flash loader demonstrator on page 27*. Updated *Section 4: Memory model on page 21* (added STNRG description). Updated *Figure 15: Running the Flash loader demonstrator - step 3 on page 29* and *Figure 17: Running the Flash loader demonstrator - step 5 on page 30* (replaced by new figures). Minor modifications throughout document.
09-Jan-2018	3	Updated *Section 3.1: Peripherals settings on page 7* and *Section 9.2: Bootloading in AutoDetect mode on page 26* (added text).