



1 Introduction

This user manual contains the bootloader specifications for STM8S devices which contain a bootloader embedded in the system memory of the device (the ROM memory). Through this firmware, the device memory can be erased and programmed using one of the standard communication interfaces present on the particular device. For each device, please refer to the corresponding datasheets to know if the bootloader is present and which peripherals are supported.

The document describes the features and operation of the STM8S integrated bootloader program. This code allows memories, including Flash program, data EEPROM, and RAM, to be written into the device using the standard serial interfaces UART1, UART2, UART3, CAN, and SPI (only available on STM8S105xx devices).

The bootloader code is the same for all STM8S versions, including STM8S20xxx and STM8S105xx devices. However, even though a peripheral may be present in a product, the product may not support it (for example, the SPI is not supported in STM8S20xxx devices). In addition, different cuts support different peripherals: The bootloader code can be accessed via the UART1, UART3 and CAN peripherals in STM8S20xxx devices and via UART2 and SPI in STM8S105xx devices.

For further information on the STM8S family features, pinout, electrical characteristics, mechanical data and ordering information, please refer to the STM8S datasheets.

Contents

- 1 Introduction 1**
- 2 Bootloader introduction 6**
 - 2.1 Bootloader activation 6
- 3 Peripheral settings 10**
 - 3.1 UART1 settings 10
 - 3.2 UART2 and UART3 settings 10
 - 3.3 SPI settings 11
 - 3.4 CAN settings 11
- 4 Bootloader command set 13**
 - 4.1 Get command 14
 - 4.1.1 Get command via UART1/ UART2/UART3 14
 - 4.1.2 Get command via SPI 16
 - 4.1.3 Get command via CAN 18
 - 4.2 Read memory command 20
 - 4.2.1 Read memory command via UART1/UART2/UART3 20
 - 4.2.2 Read memory command via SPI 22
 - 4.2.3 Read memory command via CAN 25
 - 4.3 Erase memory command 26
 - 4.3.1 Erase memory command via UART1/UART2/UART3 27
 - 4.3.2 Erase memory command via SPI 30
 - 4.3.3 Erase memory command via CAN 32
 - 4.4 Write memory command 34
 - 4.4.1 Write memory command via UART1/UART2/UART3 34
 - 4.4.2 Write memory command via SPI 37
 - 4.4.3 Write memory command via CAN 40
 - 4.5 Speed command 43
 - 4.5.1 Speed command via CAN 43
 - 4.6 Go command 45
 - 4.6.1 Go command via UART1/UART2/UART3 45
 - 4.6.2 Go command via SPI 47

4.6.3	Go command via CAN	49
4.7	Sector codes	50
5	Software model	54
6	Error management	55
7	Programming time	56
8	Revision history	57

List of tables

Table 1.	Initial checking	9
Table 2.	Serial interfaces associated with STM8S devices	10
Table 3.	Bootloader commands	13
Table 4.	Examples of delay	38
Table 5.	STM8S sector codes	50
Table 6.	Error table	55
Table 7.	UART1/UART2/UART3 programming times.	56
Table 8.	SPI programming time	56
Table 9.	CAN programming time	56
Table 10.	Document revision history	57

List of figures

Figure 1.	Bootloader activation flow chart	7
Figure 2.	CAN frame	11
Figure 3.	Get command via UART1/UART2/UART3 Host side	14
Figure 4.	Get command via UART1/UART2/UART3: Device side	15
Figure 5.	Get command via SPI: Host side	16
Figure 6.	Get command via SPI: Device side	17
Figure 7.	Get command via CAN: Host side	18
Figure 8.	Get command via CAN: Device side	19
Figure 9.	Read memory command via UART1/UART2/UART3: Host side	20
Figure 10.	Read memory command via UART1/UART2/UART3: Device side	21
Figure 11.	Read memory command via SPI: Host side	22
Figure 12.	Read memory command via SPI: Device side	24
Figure 13.	Read memory command via CAN: Host side	25
Figure 14.	Read memory command via CAN: Device side	25
Figure 15.	Erase memory command via UART1/UART2/UART3: Host side	27
Figure 16.	Erase memory command via UART1/UART2/UART3: Device side	29
Figure 17.	Erase memory command via SPI: Host side	30
Figure 18.	Erase memory command via SPI: Device side	31
Figure 19.	Erase memory command via CAN: Host side	32
Figure 20.	Erase memory command via CAN: Device side	33
Figure 21.	Write memory command via UART1/UART2/UART3: Host side	34
Figure 22.	Write memory command via UART1/UART2/UART3: Device side	36
Figure 23.	Write memory command via SPI: Host side	37
Figure 24.	Write memory command via SPI: Device side	39
Figure 25.	Write memory command via CAN: Host side	40
Figure 26.	Write memory command via CAN: Device side	42
Figure 27.	Speed command via CAN: Host side	43
Figure 28.	Speed command via CAN: Device side	44
Figure 29.	Go command via UART1/UART2/UART3: Host side	45
Figure 30.	Go command via UART1/UART2/UART3: Device side	46
Figure 31.	Go command via SPI: Host side	47
Figure 32.	Go command via SPI: Device side	48
Figure 33.	Go command via CAN: Host side	49
Figure 34.	Go command via CAN: Device side	49

2 Bootloader introduction

The main task of the bootloader is to download the application program into the internal memories through the UART1, UART2, UART3, SPI or CAN peripherals without using the SWIM protocol and dedicated hardware. Data are provided by any device (host) capable of sending information through one of the above serial interfaces.

The bootloader permits downloading of application software into the device memories, including the program memory, using standard serial interfaces (UART1, UART2, UART3, SPI and CAN) without dedicated hardware. It is a complementary solution to programming via the SWIM debugging interface.

The bootloader code is stored in the internal boot ROM memory. After a reset, the bootloader code checks whether the program memory is virgin or whether a specific option byte is set allowing code modifications.

If these conditions are not fulfilled, the bootloader resumes and the user application is started.

In case of a successful check the bootloader is executed.

When the bootloader procedure starts, the main tasks are:

- Polling all supported serial interfaces to check which peripheral is used.
- Programming code, data, option bytes and/or vector tables at the address(es) received from the host.

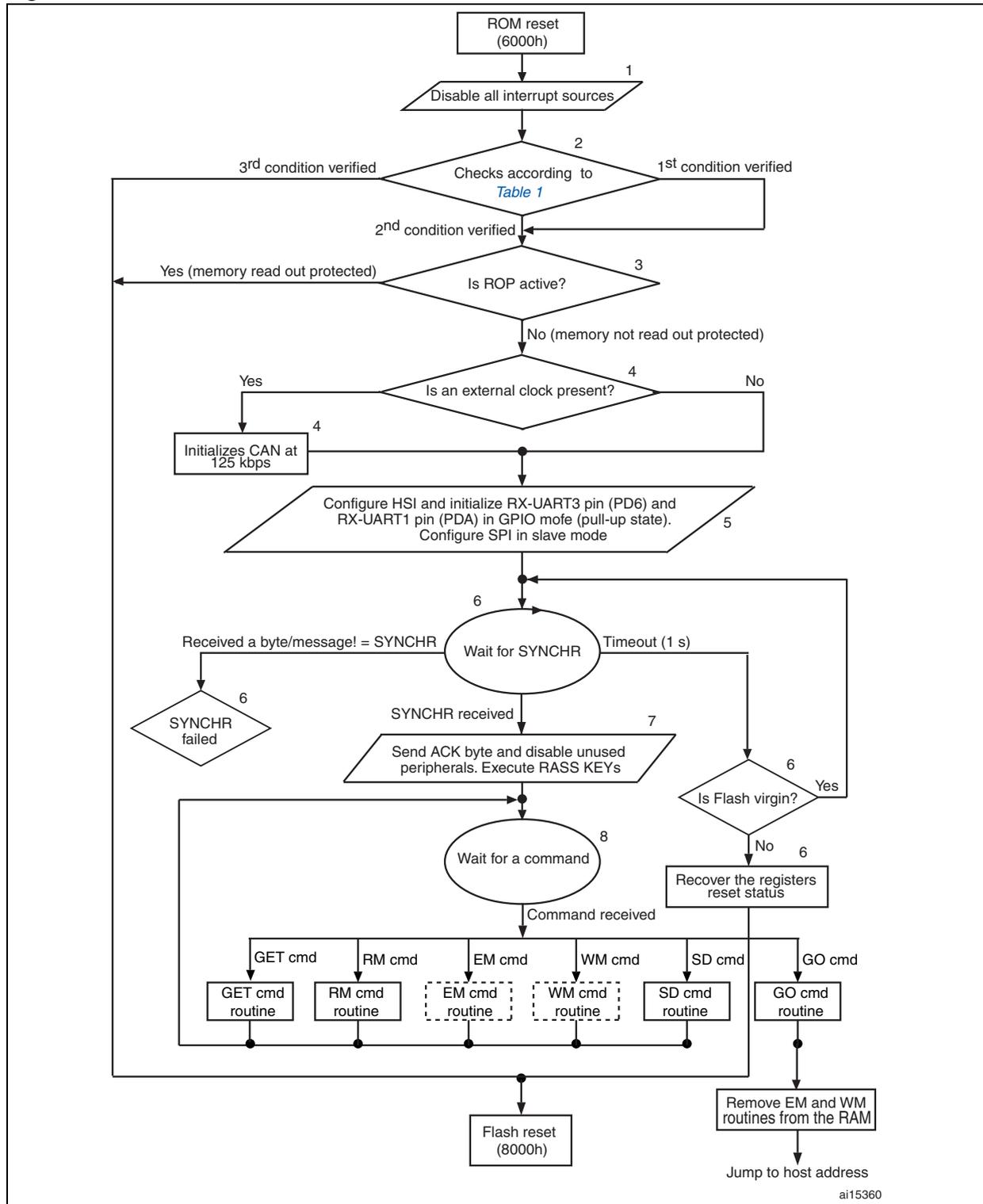
2.1 Bootloader activation

The STM8S reset vector is located at the beginning of the boot ROM (6000h), while the other vectors are in the Flash program memory starting at address 8004h.

The device executes the boot ROM, jumps inside the boot ROM area and after checking certain address locations (see [Table 1: Initial checking on page 9](#)), it starts to execute the user code via the reset vector (8000h) in the Flash program memory.

The bootloader activation flowchart is described in [Figure 1 on page 7](#).

Figure 1. Bootloader activation flow chart



1. See [Flow chart description on page 8](#) for explanation of points 1 to 8.
2. Dotted routines are loaded in RAM by the host. They are removed by the go command before jumping to the Flash program memory to execute an application.

Flow chart description

1. Disable all interrupt sources.
2. The host can reprogram the Flash program memory and the bootloader option byte values, as shown in [Table 1](#) according to the content of the first Flash program memory location (8000h).
3. When read out protection (ROP) is equal to AAh (ROP active), the Flash program memory is read out protected. In this case, the bootloader stops and the user application starts. If ROP is not equal to AAh, the bootloader continues to be executed.
4. The CAN peripheral can only be used if an external clock (8 MHz, 16 MHz, or 24 MHz) is present. It is initialized at 125 kbps. The UART1, UART2, UART3 and SPI peripherals do not require an external clock.
5. Set the high speed internal RC oscillator (HSI) to 16 MHz and initialize the UART1, UART2 and UART3 receiver pins in input pull-up mode in the GPIO registers. Initialize the SPI in slave mode.
6. Interface polling (point S): The bootloader polls the peripherals waiting for a synchronization byte/message (SYNCHR) within a timeout of 1 s. If a timeout occurs, either the Flash program memory is virgin in which case it waits for a synchronization byte/message in an infinite loop, or the Flash program memory is not virgin and the bootloader restores the registers' reset status before going to the Flash program memory reset vector at 8000h.

Note: When synchronization fails and the bootloader receives a byte/message different to 'SYNCHR', two different situations can be distinguished according to the peripheral:
With UART1, UART2 or UART3, a device reset or power-down is necessary before synchronization can be tried again.

With CAN or SPI, the user can continue to poll the interfaces until a synchronization or a timeout occurs.

7. If the synchronization message is received by the UART1, UART2 or UART3, the bootloader detects the baud rate and initializes the UART1, UART2 or UART3 respectively and goes to step [8](#) below. If the synchronization message is received by the CAN or SPI, the bootloader goes directly to step [8](#) below.

Note: Once one of the available interfaces receives the synchronization message, all others are disabled.

8. Waiting for commands (point C): Commands are checked in an infinite loop and executed. To exit from the bootloader, the host has to send a 'go' command. When this is done, the bootloader removes the EM and WM routines from the RAM memory before jumping to the address selected by the host.

Table 1. Initial checking⁽¹⁾

Checks	Program memory location 8000h	Bootloader check opt_byte 487Eh	Bootloader check opt_byteN 487Fh	Actual Flash program memory status -> Flash action
1 st	XXh! = (82h or ACh)	XXh	XXh	Flash program memory virgin -> jump to bootloader
2 nd	XXh = (82h or ACh)	55h	AAh	Flash program memory already written -> jump to bootloader
3 rd	XXh = (82h or ACh)	XXh! = 55h	XXh! = AAh	Flash program memory already written -> jump to Flash program memory reset

1. After interface initialization, a write protection test is performed to avoid non-authorized reading of the Flash program memory/data EEPROM,

3 Peripheral settings

This section describes the hardware settings of the STM8S communication peripherals:

- UART1
- UART2
- UART3
- SPI
- CAN

Note: During bootloading only one peripheral is enabled; all others are disabled.

Table 2. Serial interfaces associated with STM8S devices

Device	Serial interface
STM8S20xxx	UART1, UART3, CAN
STM8S105xx	UART2, SPI

3.1 UART1 settings

All UART peripherals support asynchronous serial communication.

UART1 settings are:

- Data frame: 1 start bit, 8 data bit, 1 parity bit even, 1 stop bit
- Baud rate: The baud rate is autodetected by the bootloader. When the user sends the synchronization byte, 7Fh, the bootloader automatically detects the baud rate and sets the UART1 to the same baud rate. Maximum baud rate = 1 Mbps; minimum baud rate = 4800 bps.

Mandatory: To perform the automatic speed detection, the RX line (PA4) has to be stable in the application board.

Note: The UART1 peripheral is accessible via pins PA4 (RX) and PA5 (TX).

3.2 UART2 and UART3 settings

All UART peripherals support asynchronous serial communication.

UART2 and UART3 settings are:

- Data frame: 1 start bit, 8 data bit, no parity bit, 1 stop bit
- Baud rate: The baud rate is autodetected by the bootloader. When the user sends the byte 7Fh, the bootloader automatically detects the baud rate and sets the UART2 or UART3 to the same baud rate. Maximum baud rate = 550 kbps; minimum baud rate = 4800 bps.

Mandatory: To perform the automatic speed detection, the RX line (PD6 for UART3) has to be stable in the application board.

Note: UART2 and UART3 peripherals are accessible via pins PD5 (TX) and PD6 (RX).

3.3 SPI settings

The SPI settings are:

- 8 data bit, LSB first
- Bit rate: Set by the host which acts as a master
- Peripheral set in slave mode with NSS not used.
- Data polarity: CPOL = CPOH = 0

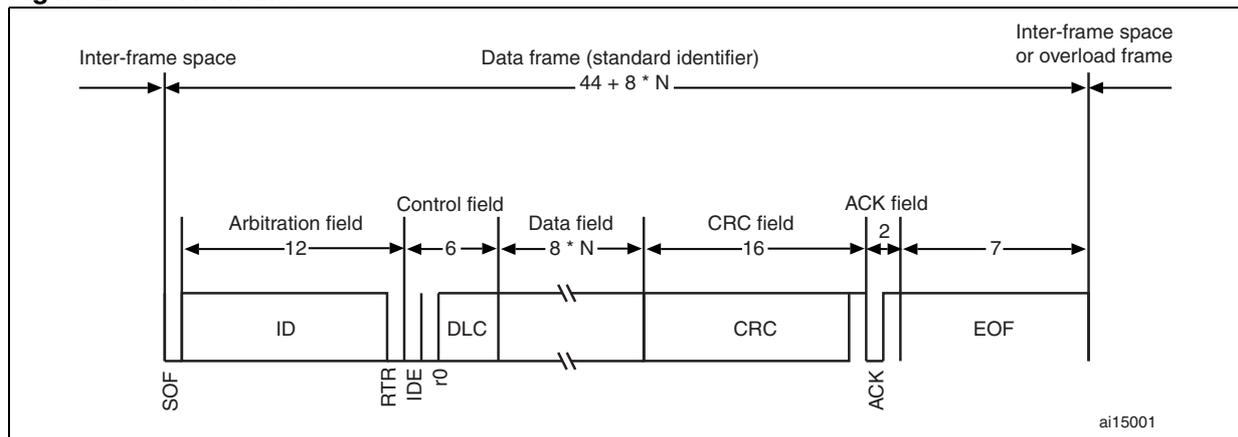
- Note:*
- 1 Before sending a 'token' byte, the host has to wait for a delay of a specified period of time. If this period is not quantified, it is equal to 6 μ s.
 - 2 The SPI peripheral is accessible via pins PC5 (SCK), PC6 (MOSI), and PD7 (MISO).

3.4 CAN settings

To address additional devices on the same bus, the CAN protocol provides a standard identifier field (11-bit) and an optional extended identifier field (18-bit) in the frame.

Figure 2 shows the CAN frame that uses the standard identifier only.

Figure 2. CAN frame



The CAN settings are:

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

The transmit settings (from STM8S to the host) are:

- Tx mailbox0: On
- Tx mailbox1 and Tx mailbox2: Off
- Tx identifier: 02h
- Outgoing messages contain 1 data byte.

The receive settings (from the host to STM8S) are:

- Synchronization byte, 7Fh, is in the RX identifier and not in the data field.
- RX identifier depends on the command (00h, 03h, 11h, 21h, 31h, 43h).
- Error checking: If the error field (bit [6:4] in the CESR 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.

Note: The CAN peripheral is accessible via pins PG0 (TX) and PG1 (RX).

4 Bootloader command set

The supported commands are listed in [Table 3](#) below.

Table 3. Bootloader commands

Command	Command code	Command description
Get	00h	Gets the version and the allowed commands supported by the current version of the bootloader
Read memory	11h	Reads up to 256 bytes of memory starting from an address specified by the host
Erase memory	43h	Erases from one to all of the Flash program memory/data EEPROM memory sectors
Write memory	31h	Writes up to 128 bytes to the RAM or the Flash program memory/data EEPROM memory starting from an address specified by the host
Speed	03h	Allows the baud rate for CAN run-time to be changed
Go	21h	Jumps to an address specified by the host to execute a loaded code

When the bootloader receives a command via the UART1, UART2, UART3, CAN or SPI peripherals, the general protocol is as follows: The bootloader sends an ACK byte to the host and waits for an address and for a checksum byte, both of which are checked when received. 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 (N bytes) and for its complemented byte (checksum). If the checksum is correct, it then carries out the command, starting from the received address. If the checksum is incorrect, it sends a NACK byte before aborting the command.

The bootloader protocol via UART1, UART2, UART3 and SPI are identical on the device side, but, differ regarding the host. A token byte is needed when sending each byte to the host via SPI (see [Figure 5](#), [Figure 11](#), [Figure 17](#), [Figure 23](#), and [Figure 31](#)). The bootloader protocol via CAN differs from all other peripherals.

The following sections are organised as follows:

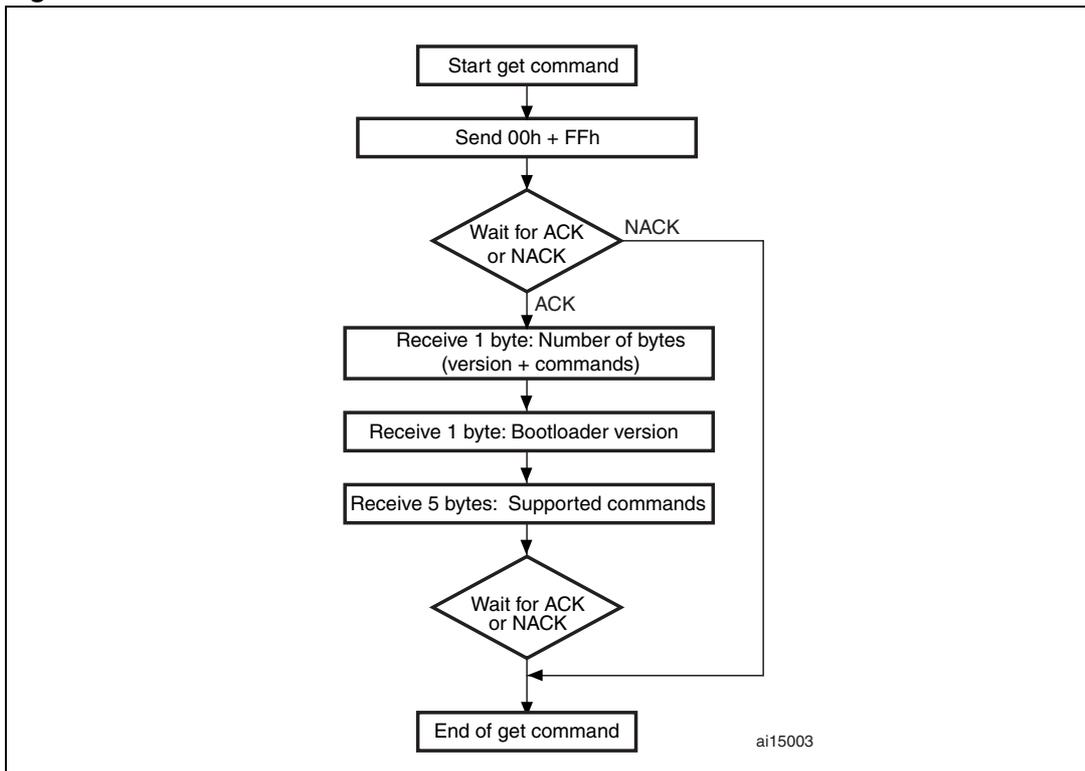
- Commands via UART1/ UART2/ UART3
- Commands via SPI
- Commands via CAN

4.1 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.

4.1.1 Get command via UART1/ UART2/UART3

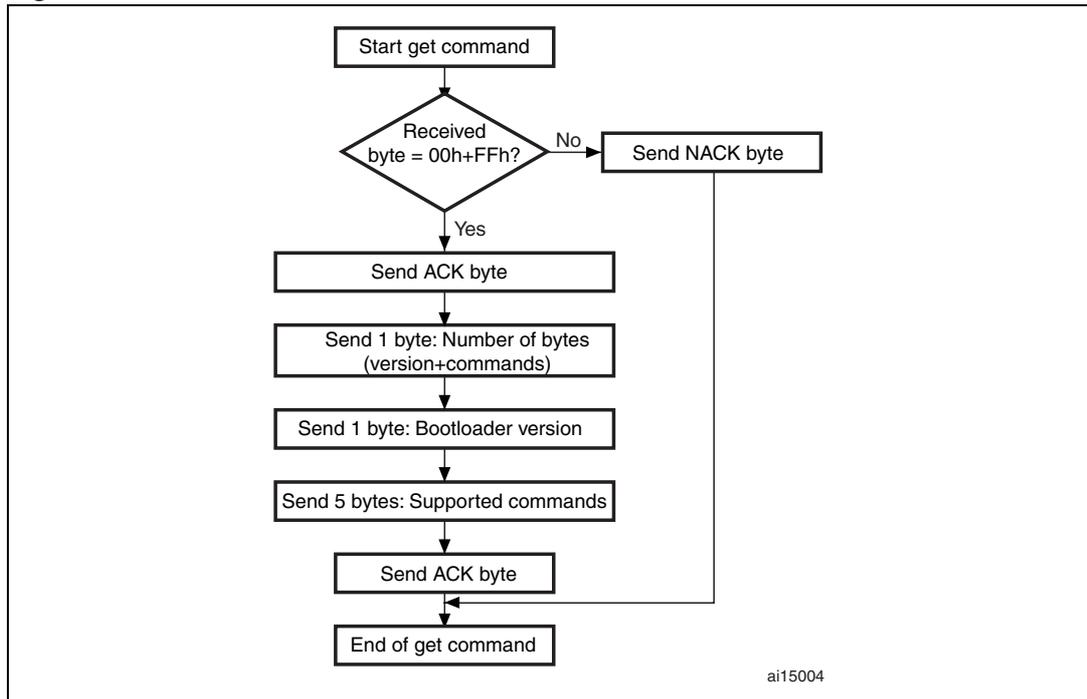
Figure 3. Get command via UART1/UART2/UART3 Host side



The host sends the bytes as follows

- Byte 1: 00h - Command ID
- Byte 2: FFh - Complement

Figure 4. Get command via UART1/UART2/UART3: Device side

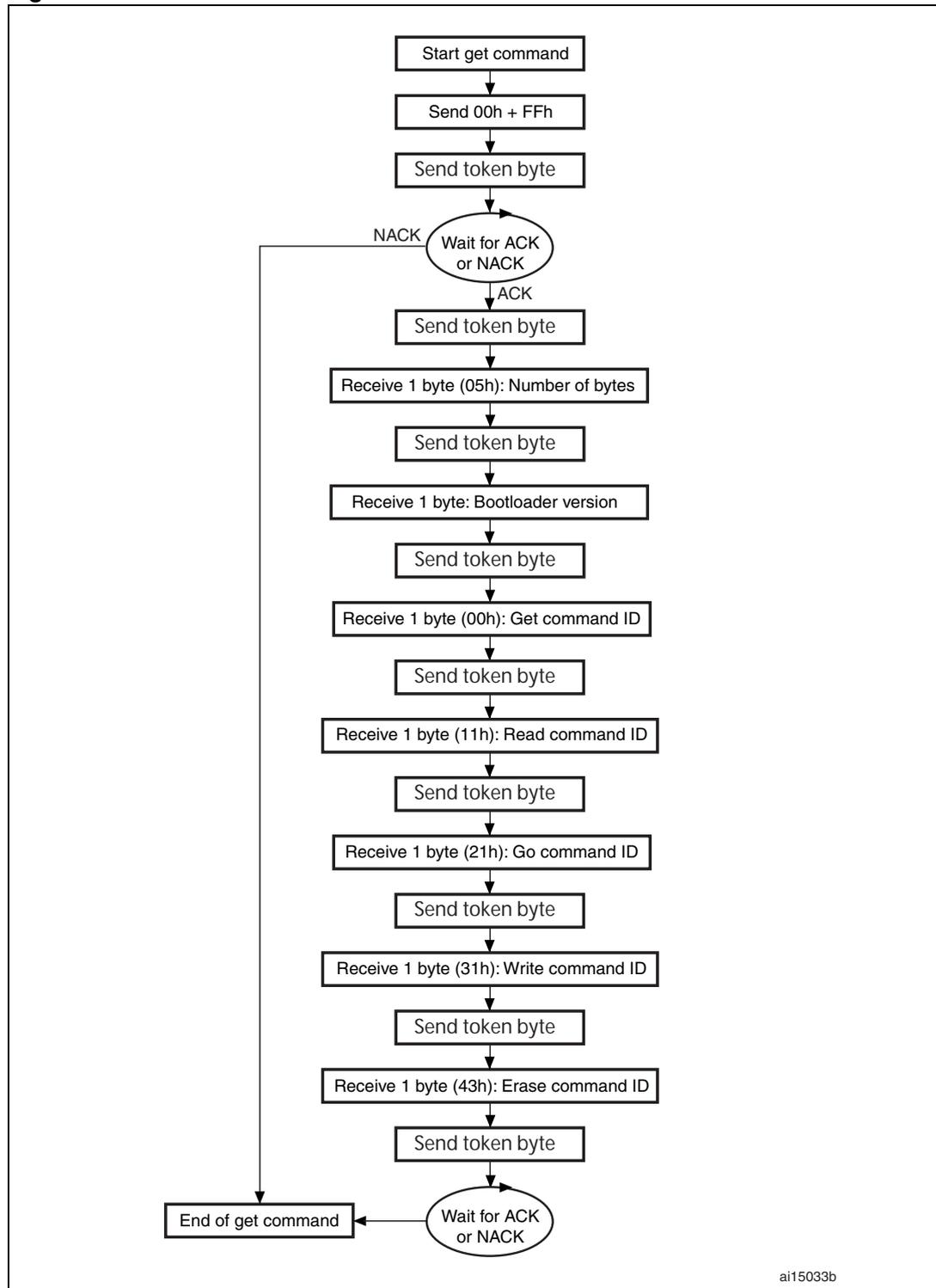


The STM8S sends the bytes as follows

- Byte 1: ACK (after the host has sent the command)
- Byte 2: $N = 5 =$ the number of bytes to be sent -1 ($1 \leq N + 1 \leq 256$)
- Byte 3: Bootloader version ($0 < \text{version} \leq 255$)
- Byte 4: 00h - Get command
- Byte 5: 11h - Read memory command
- Byte 6: 21h - Go command
- Byte 7: 31h - Write memory command
- Byte 8: 43h - Erase memory command
- Byte 9: ACK

4.1.2 Get command via SPI

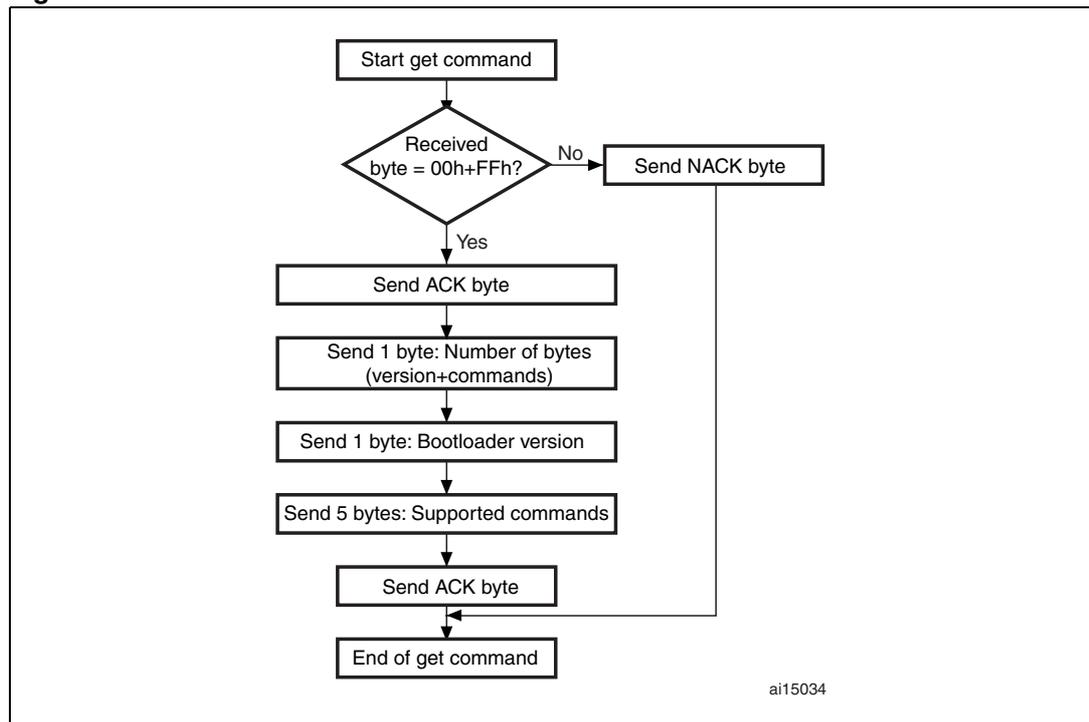
Figure 5. Get command via SPI: Host side



The host sends the bytes as follows

- Byte 1: 00h - Command ID
- Byte 2: FFh - Complement
- Byte 3 (token): XYh; host waits for ACK or NACK
- Byte 4 (token): XYh; host waits for 05h
- ...
- Byte 4 (token): XYh; host waits for ACK or NACK.

Figure 6. Get command via SPI: Device side

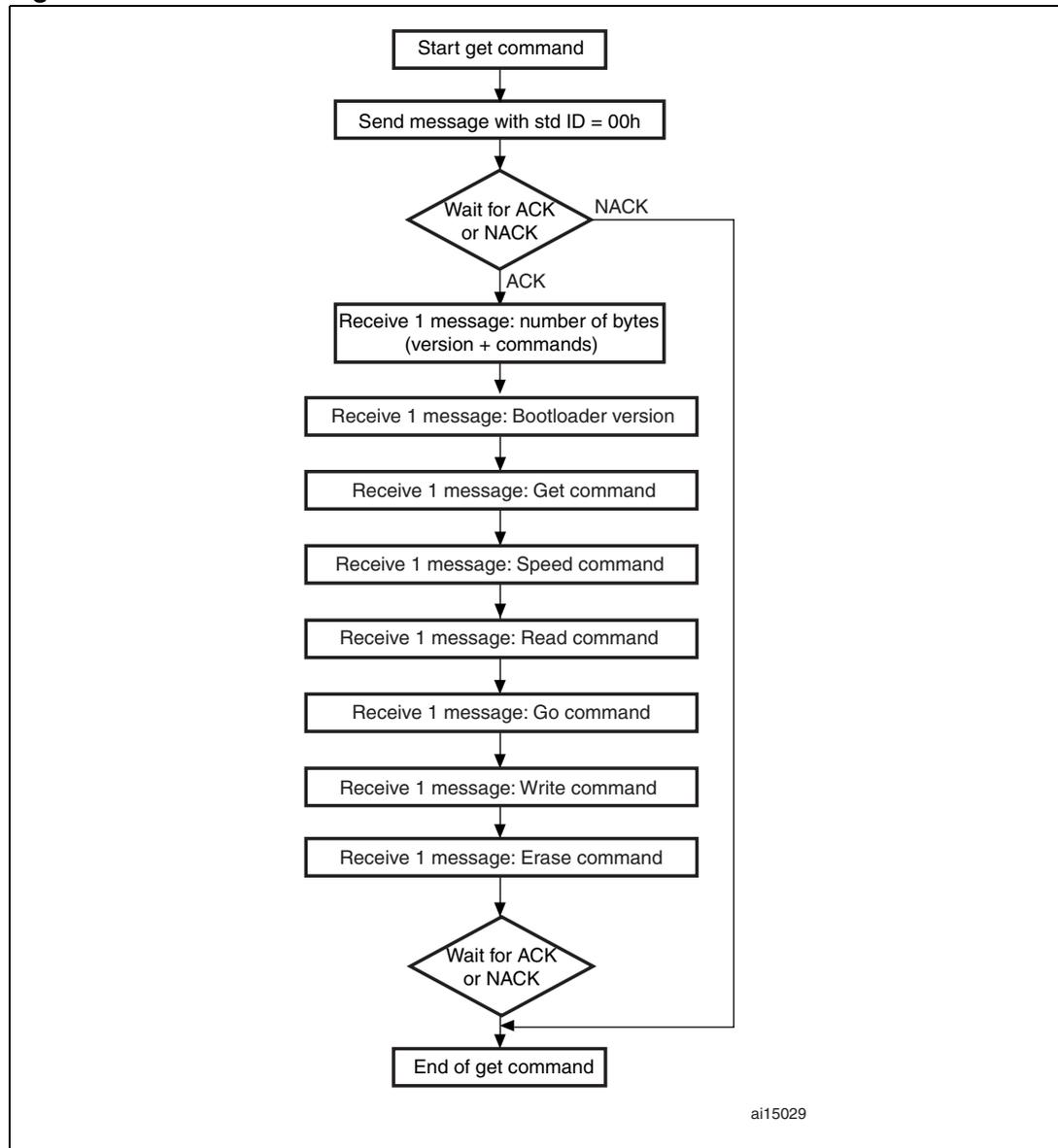


The STM8S sends the bytes as follows

- Byte 1: ACK
- Byte 2: N = 5 = the number of bytes to be sent -1 (1 <= N +1 <= 256)
- Byte 3: Bootloader version (0 < version <= 255)
- Byte 4: 00h - Get command
- Byte 5: 11h - Read memory command
- Byte 6: 21h - Go command
- Byte 7: 31h - Write memory command
- Byte 8: 43h - Erase memory command
- Byte 9: ACK

4.1.3 Get command via CAN

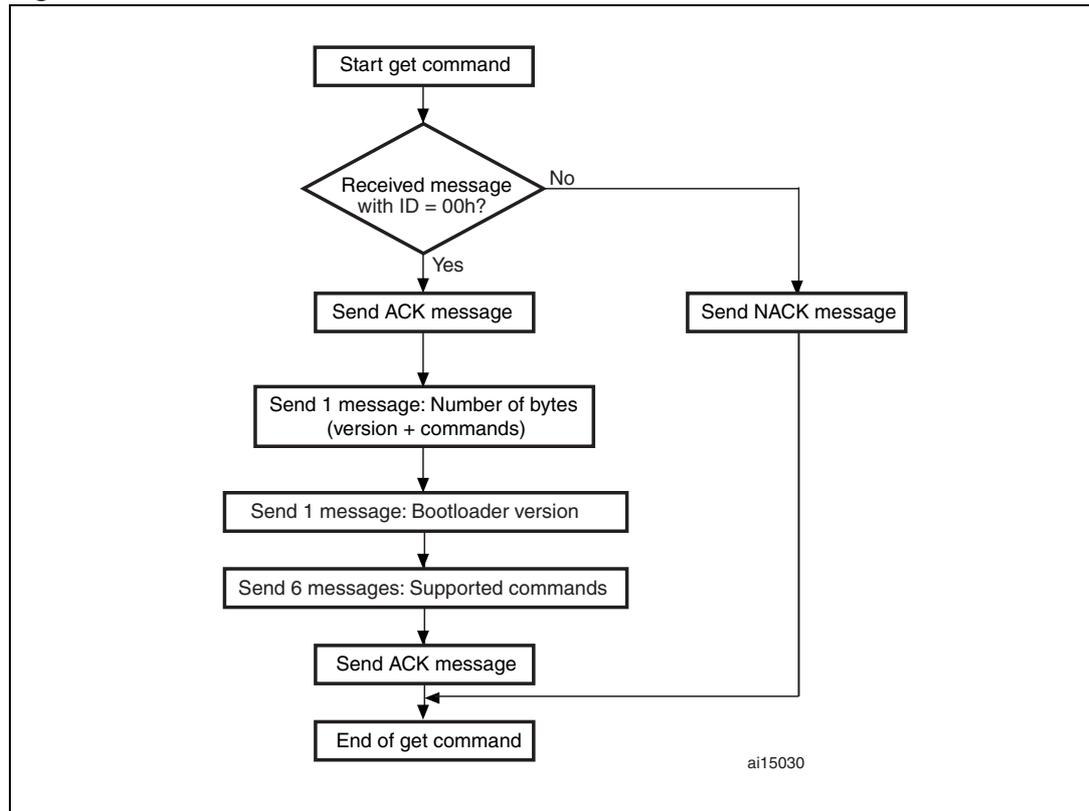
Figure 7. Get command via CAN: Host side



The host sends the messages as follows

Command message: Std ID = 00h, data length code (DLC) = 'not important'.

Figure 8. Get command via CAN: Device side



The STM8S sends the messages as follows

Message 1: Std ID = 02h, DLC = 1, data = ACK

Message 2: Std ID = 02h, DLC = 1 data = N = 6 = the number of bytes to be sent -1
(1 <= N +1 <= 256)

Message 3: Std ID = 02h, DLC = 1, data = bootloader version (0 < version <= 255)

Message 4: Std ID = 02h, DLC = 1, data = 00h - Get command

Message 5: Std ID = 02h, DLC = 1, data = 03h - Speed command

Message 6: Std ID = 02h, DLC = 1, data = 11h - Read memory command

Message 7: Std ID= 02h, DLC = 1, data = 21h - Go command

Message 8: Std ID = 02h, DLC = 1, data = 31h - Write memory command

Message 9: Std ID= 02h, DLC = 1, data = 43h - Erase memory command

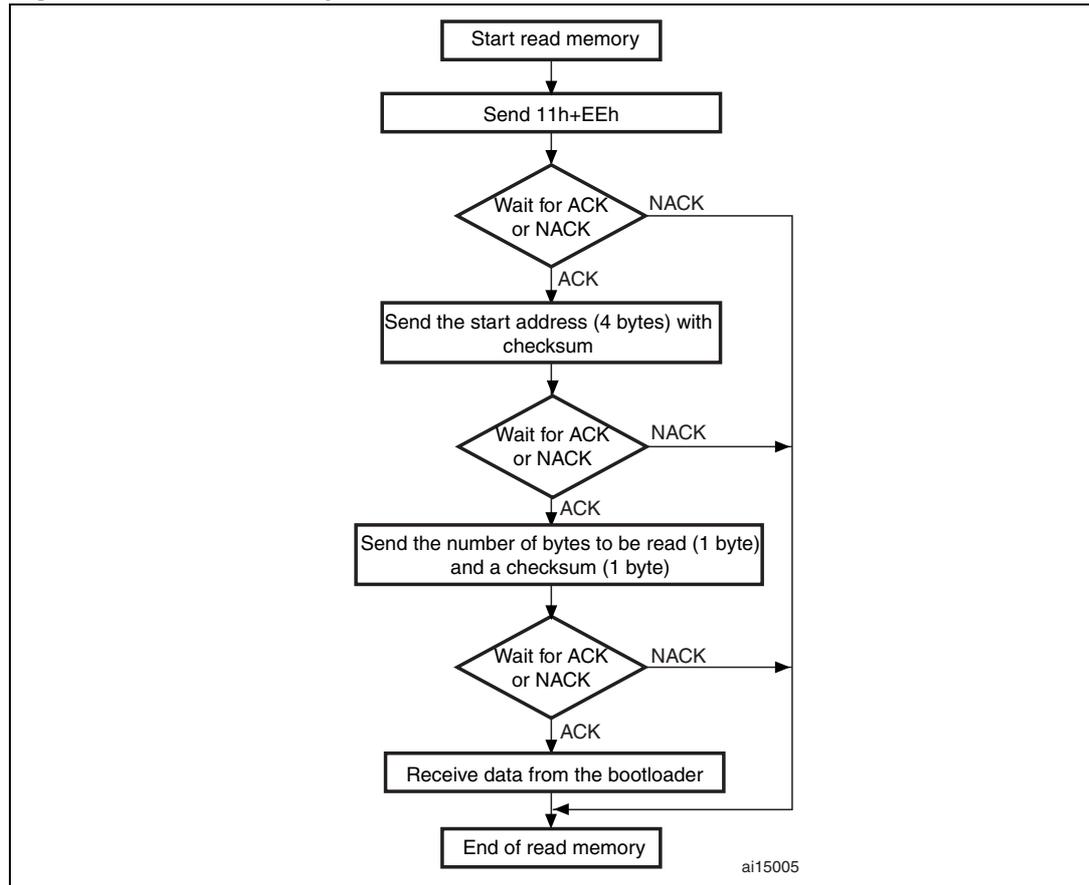
Message 10: Std ID = 02h, DLC = 1, data = ACK

4.2 Read memory command

The read memory command is used to read the memory (RAM, Flash program memory/data EEPROM or registers). When the bootloader receives the read memory command, it transmits the needed data ((N + 1) bytes) to the host, starting from the received address.

4.2.1 Read memory command via UART1/UART2/UART3

Figure 9. Read memory command via UART1/UART2/UART3: Host side

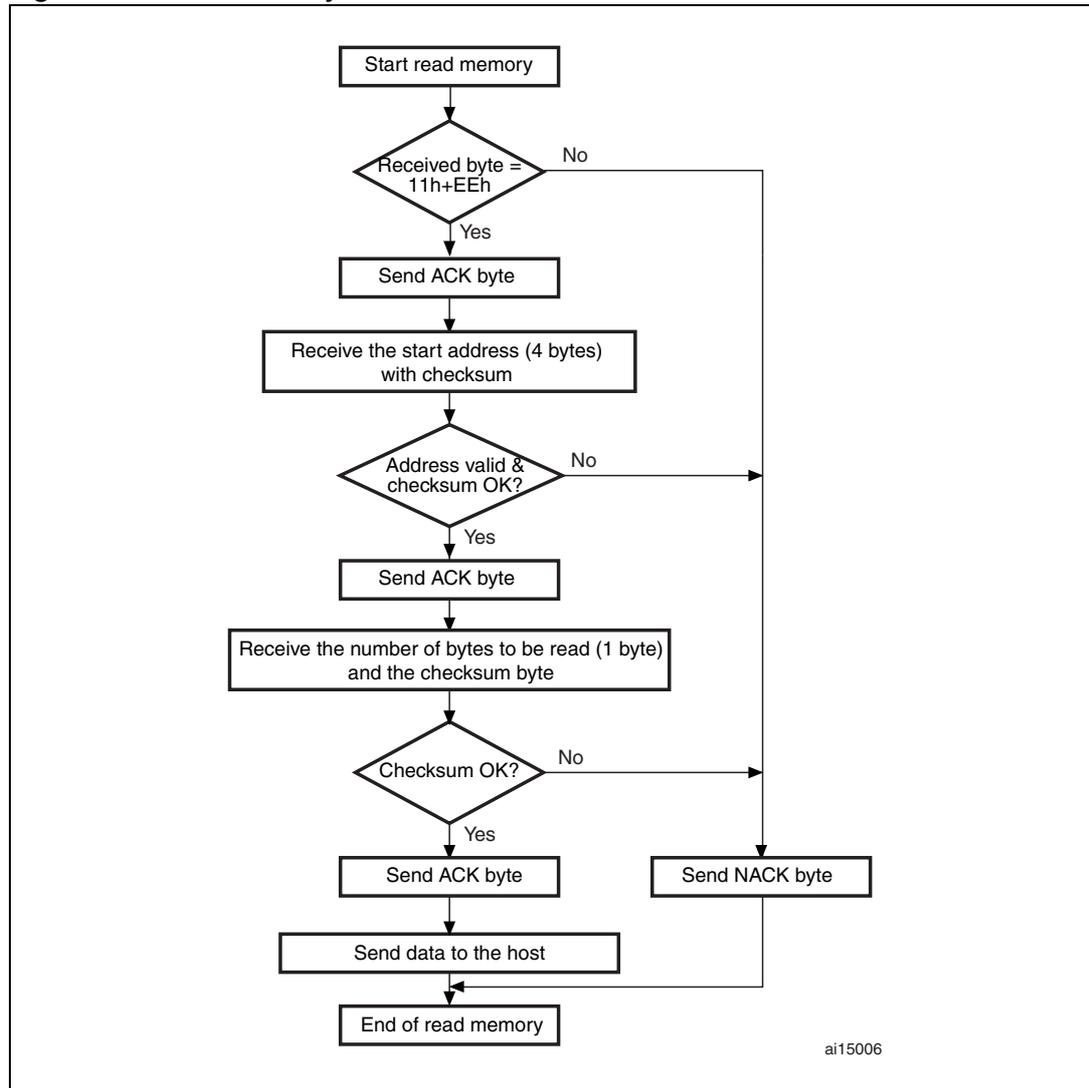


1. The valid addresses are RAM, Flash program memory/data EEPROM and register addresses (see product datasheet). If the bootloader receives an invalid address, an 'add error' occurs (see [Table 6: Error table on page 55](#)).

The host sends the bytes to the STM8S as follows

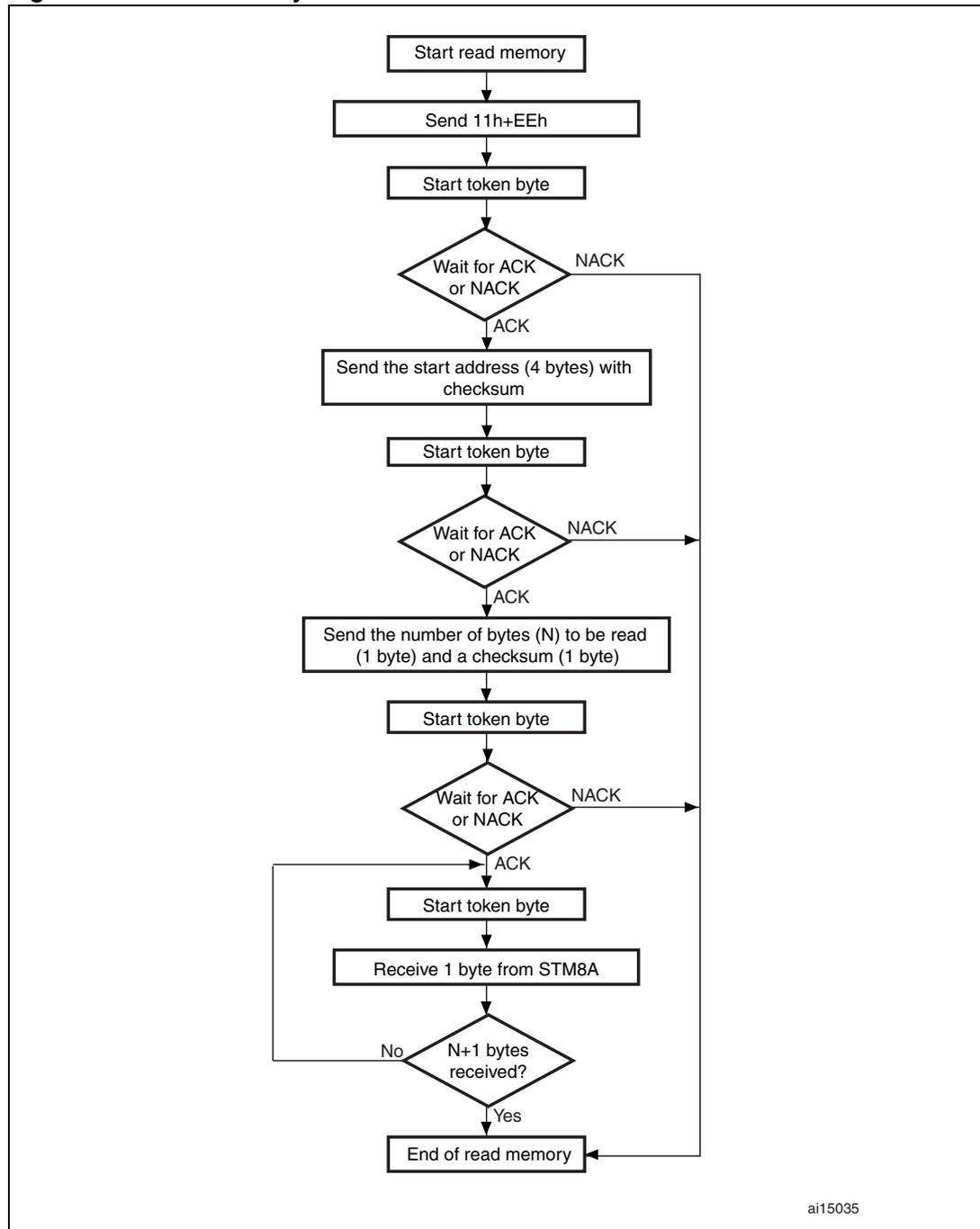
- Bytes 1-2: 11h+EEh
- Bytes 3-6: The start address
 - Byte 3 = MSB
 - Byte 6 = LSB
- Byte 7: Checksum = XOR (byte 3, byte 4, byte 5, byte 6)
- Byte 8: The number of bytes to be read ($0 < N \leq 255$)
- Byte 9: Checksum \neq byte 8.

Figure 10. Read memory command via UART1/UART2/UART3: Device side



4.2.2 Read memory command via SPI

Figure 11. Read memory command via SPI: Host side

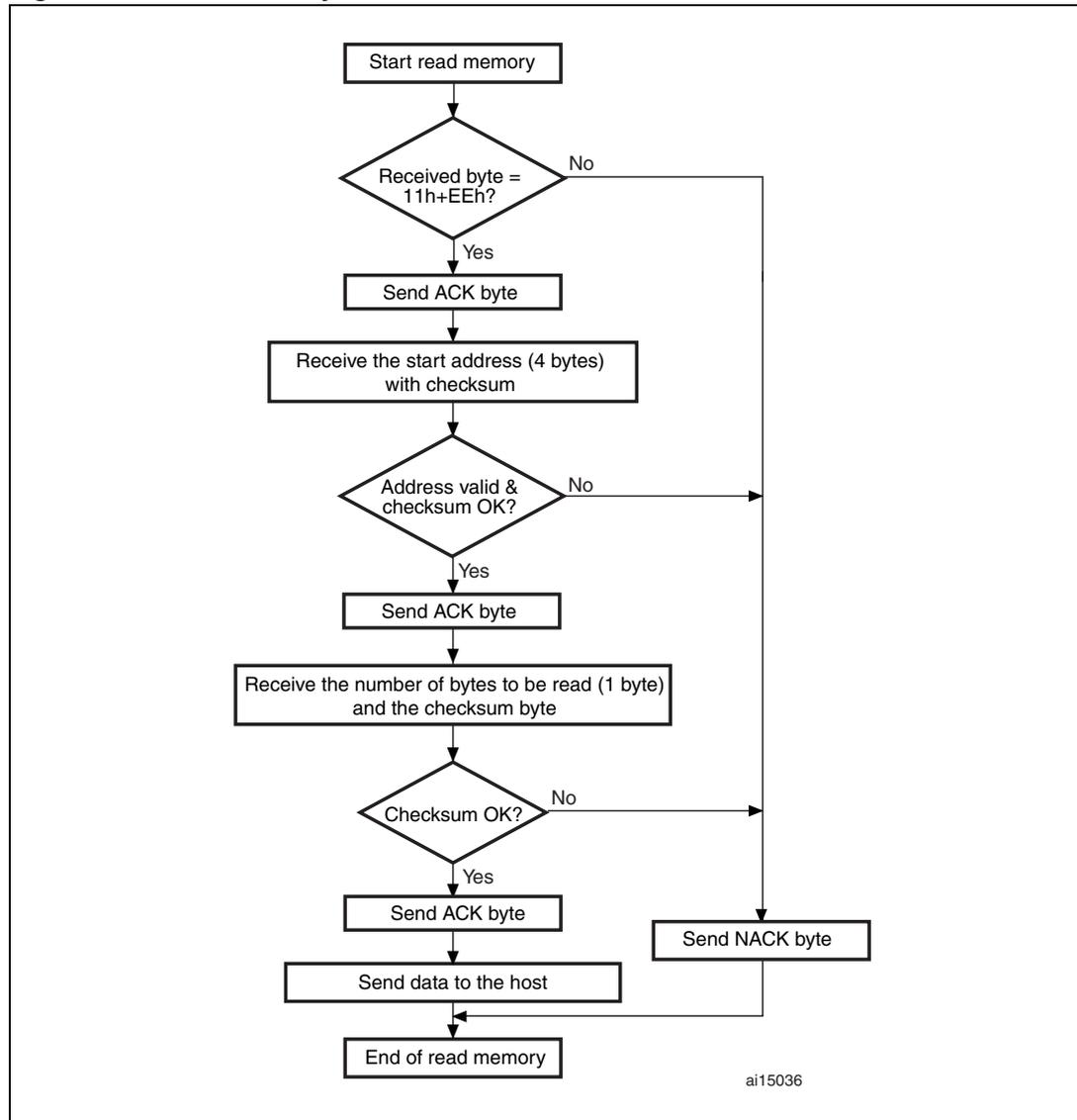


1. The valid addresses are RAM, Flash program memory/data EEPROM and register addresses (see product datasheet). If the bootloader receives an invalid address, an 'add error' occurs (see [Table 6: Error table on page 55](#)).

The host sends the bytes to the STM8S as follows

Byte 1:	11h	- Command ID
Byte 2:	EEh	- Complement
Byte 3 (token):	XYh;	host waits for ACK or NACK
Bytes 4 to 7:	The start address	
	Byte 4 = MSB	
	Byte 7 = LSB	
Byte 8:	Checksum = XOR (byte 4, byte 5, byte 6, byte 7)	
Byte 9 (token):	XYh;	host waits for ACK or NACK
Byte 10:	The number of bytes to be read ($0 < N \leq 255$)	
Byte 11:	Checksum = Complement of byte 10	
Byte 12 (token):	XYh;	host waits for the 1 st data byte
Byte 12+N (token):	XYh;	host waits for the N+1 th data byte

Figure 12. Read memory command via SPI: Device side

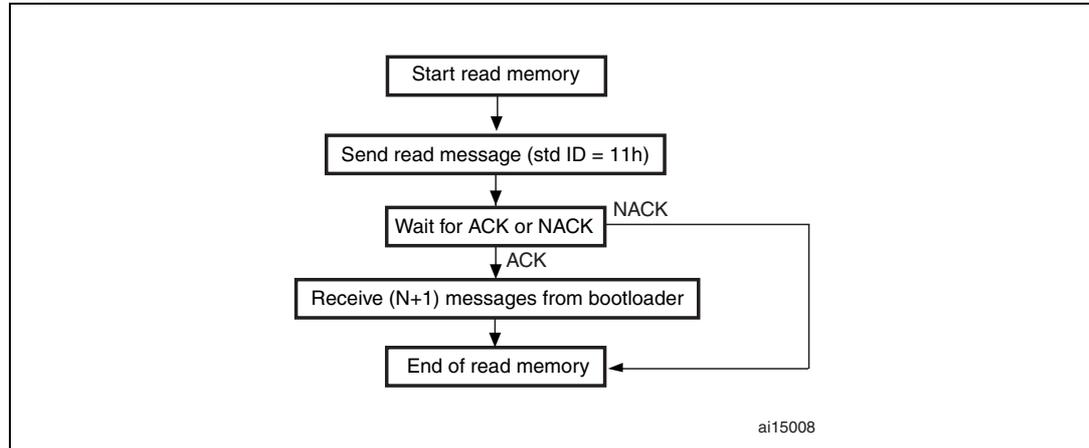


4.2.3 Read memory command via CAN

The CAN message sent by the host is as follows:

- The ID contains the command type (11h)
- The data field contains a destination address (4 bytes, byte 1 is the MSB and byte 4 is LSB of the address) and the 'number of bytes' (N) to be read.

Figure 13. Read memory command via CAN: Host side

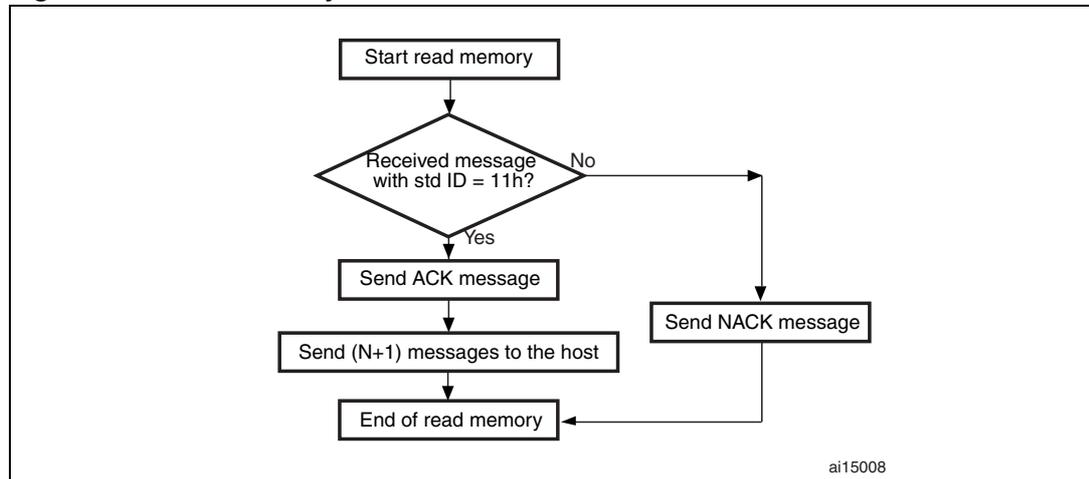


1. The valid addresses are RAM, Flash program memory/data EEPROM and register addresses (see product datasheet). If the bootloader receives an invalid address, an 'add error' occurs (see [Table 6: Error table on page 55](#)).

The host sends the messages as follows

Command message: Std ID = 11h, DLC = 05h, data = MSB, XXh, YYh, LSB, N (where 0 < N <= 255).

Figure 14. Read memory command via CAN: Device side



The STM8S sends the messages as follows

ACK message: Std ID = 02h, DLC = 1, data = ACK

Data message 1: Std ID = 02h, DLC = 1, data = XXh

Data message 2: Std ID = 02h, DLC = 1, data = XXh

...

Data message (N+1): Std ID = 02h, DLC = 1, data = XXh

Note: The bootloader sends as many data messages as bytes which can be read.

4.3 Erase memory command

The erase memory command allows the host to erase sectors of the Flash program memory/data EEPROM memory.

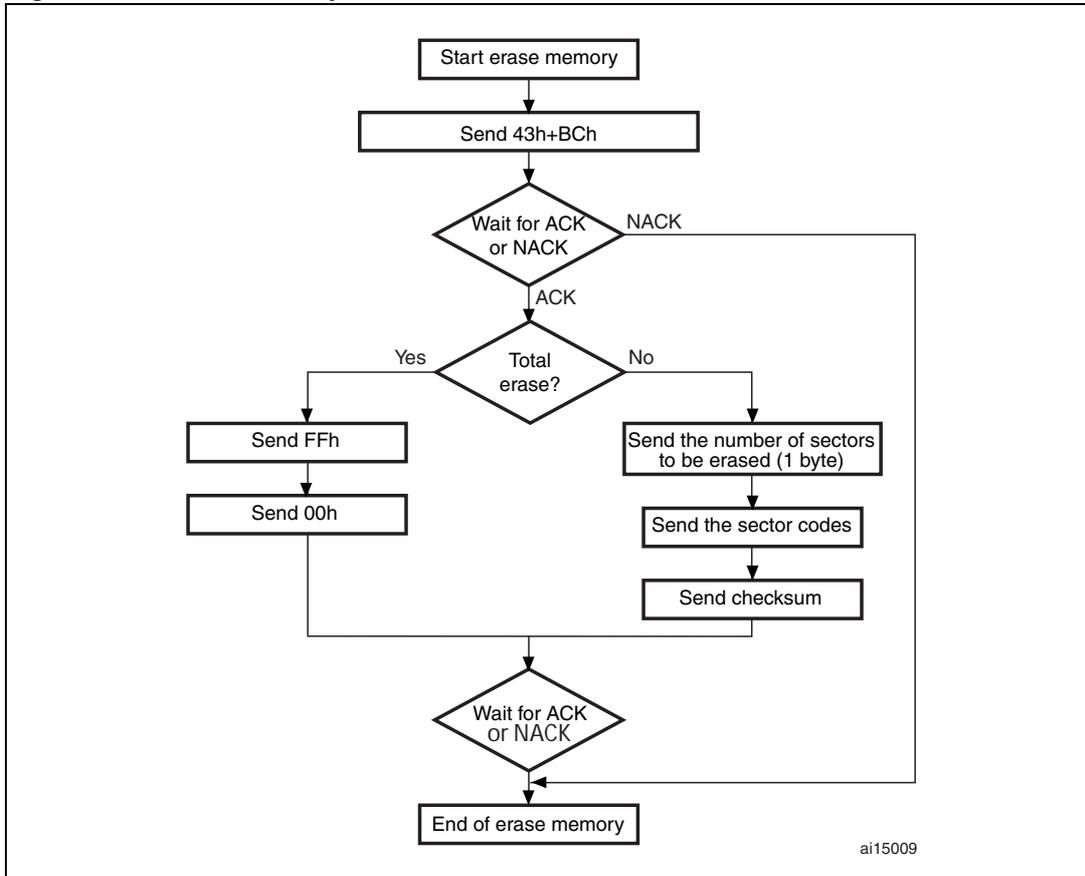
The bootloader receives the erase command message, when the ID contains the command type (43h) and the data field contains the sectors to be erased (see [Table 5: STM8S sector codes on page 50](#)). A sector is 1 Kbyte, therefore, the granularity with the erase command is 8 blocks. If the host wants to erase one byte, the write command (write 00h) can be used.

Erase memory command specifications:

1. The bootloader receives one byte which contains the number (N) of sectors to be erased. N is device dependent.
2. The bootloader receives (N + 1) bytes, where each byte contains a sector code (see).

4.3.1 Erase memory command via UART1/UART2/UART3

Figure 15. Erase memory command via UART1/UART2/UART3: Host side



1. Memory erased =130 Kbytes for devices with 128 Kbytes of Flash program memory and 2 Kbytes of Data EEPROM and 33 Kbytes for devices with 32 Kbytes of Flash program memory and 1 Kbyte of Data EEPROM.
2. A sector is 8 blocks i.e. 1 Kbyte for any device. Therefore, the granularity of the erase command is 1 Kbyte.

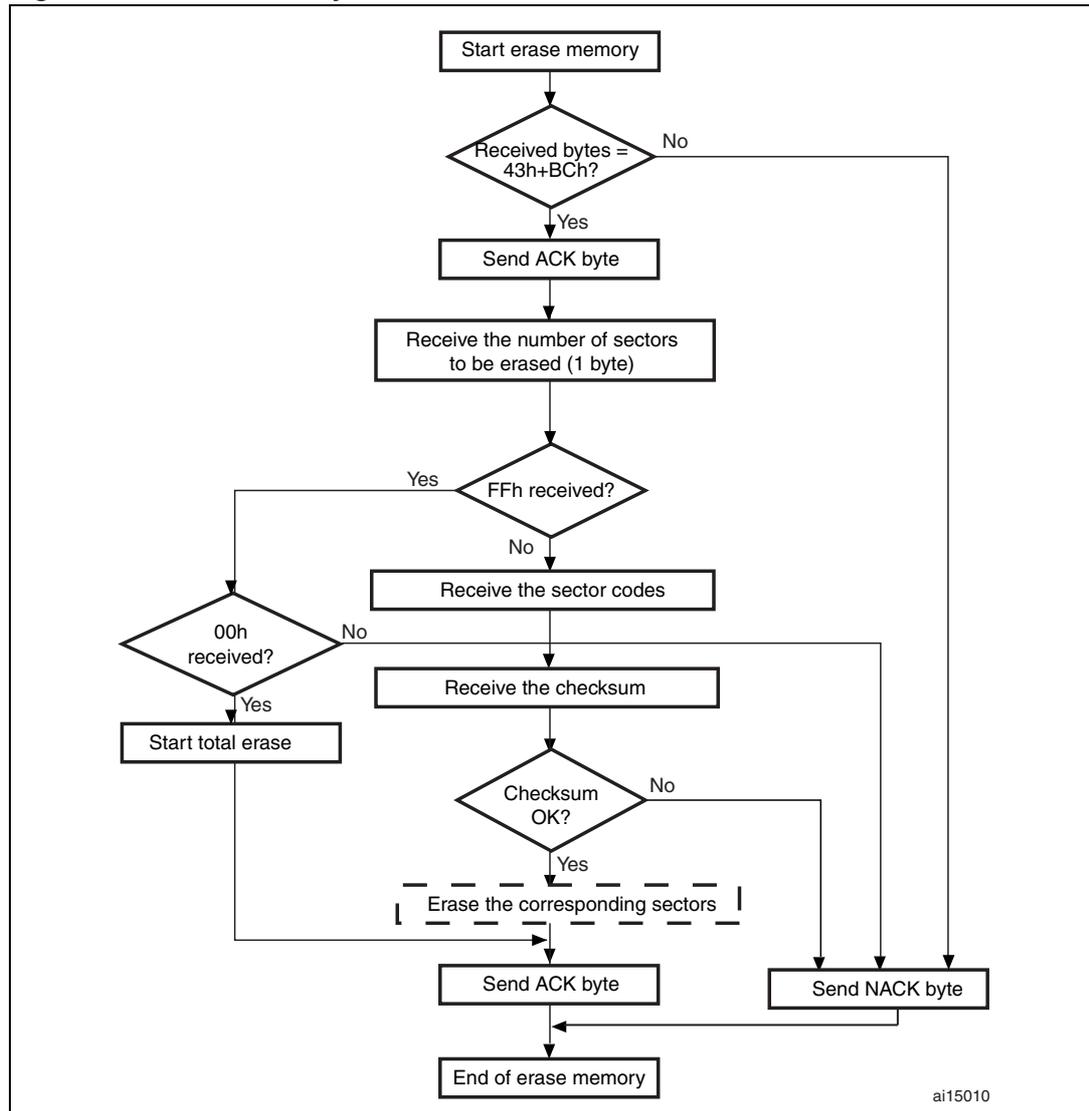
Warning: If the host sends an erase command that includes some correct sector code and one or more forbidden sector codes, the command fails.

The host sends the bytes as follows

Byte 1: 43h - Command ID
Byte 2: BCh - Complement
Byte 3: FFh or number of sectors to be erased ($0 \leq N \leq M$);
if $N > M$, a cmd_error occurs in the bootloader, after which the
bootloader receives $N + 1$ data bytes and the checksum (i.e. the host
completes the command).

Note: N is product dependent.
 $M = (\text{size of the Flash program memory in Kbyte}) + (\text{size of Data EEPROM in Kbyte}) - 1$.
For instance :
For STM8S20xxB, $M = 129$ as Flash program memory is 128 Kbytes and Data EEPROM
memory is 2 Kbytes ($128 + 2 - 1$).
For STM8S105x6, $M = 32$ as Flash program memory is 32Kbytes and Data EEPROM is
1Kbyte ($32 + 1 - 1$).
Byte 4 or $N+1$ bytes: 00h or ($N+1$ bytes and then checksum: $XOR(N, N+1 \text{ bytes})$).

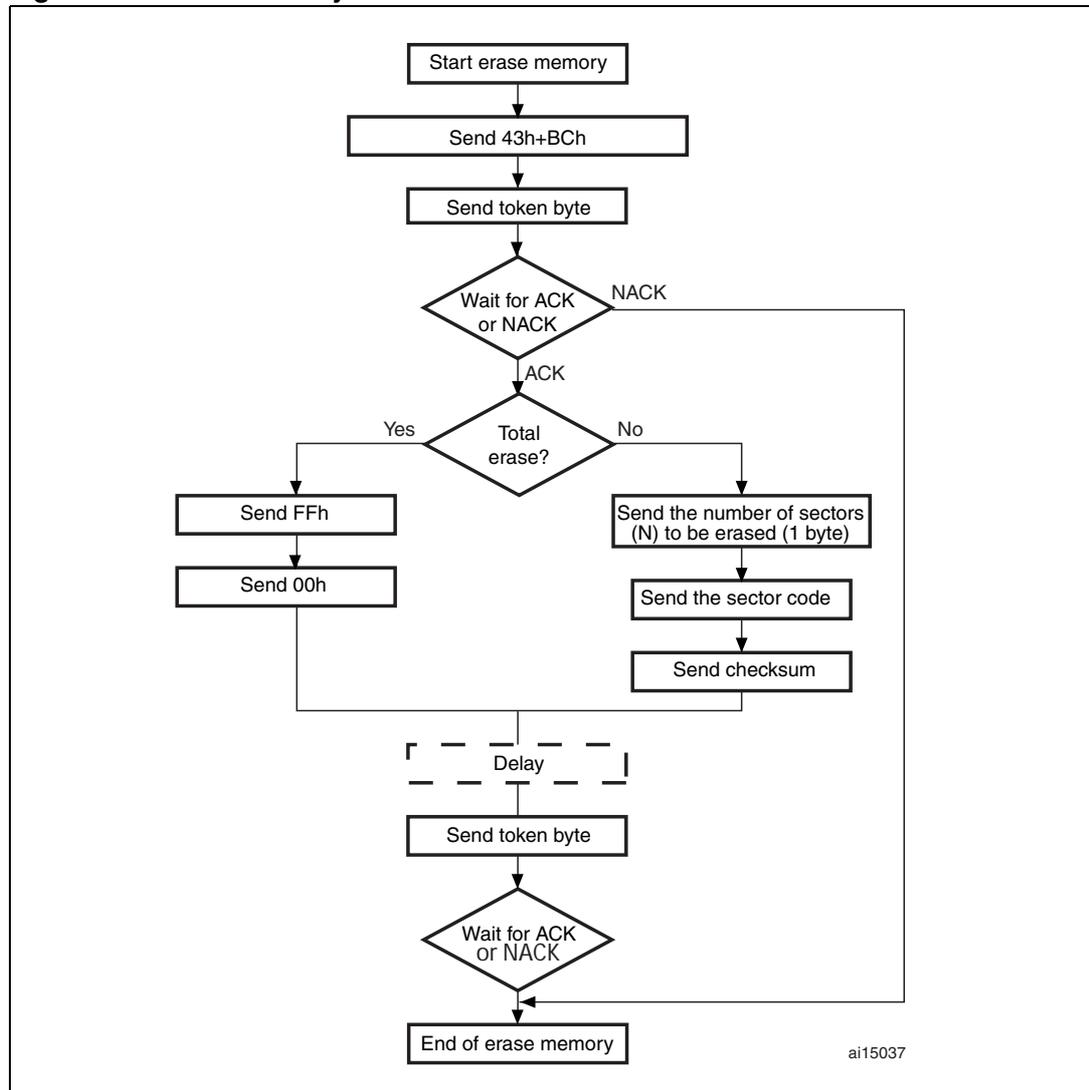
Figure 16. Erase memory command via UART1/UART2/UART3: Device side



1. For safety reasons no erase routine is placed into ROM. The 'erase the corresponding sectors' routine (see [Figure 16](#)) is performed in RAM. The user, therefore has to download the erase routine in RAM before sending an erase command.

4.3.2 Erase memory command via SPI

Figure 17. Erase memory command via SPI: Host side

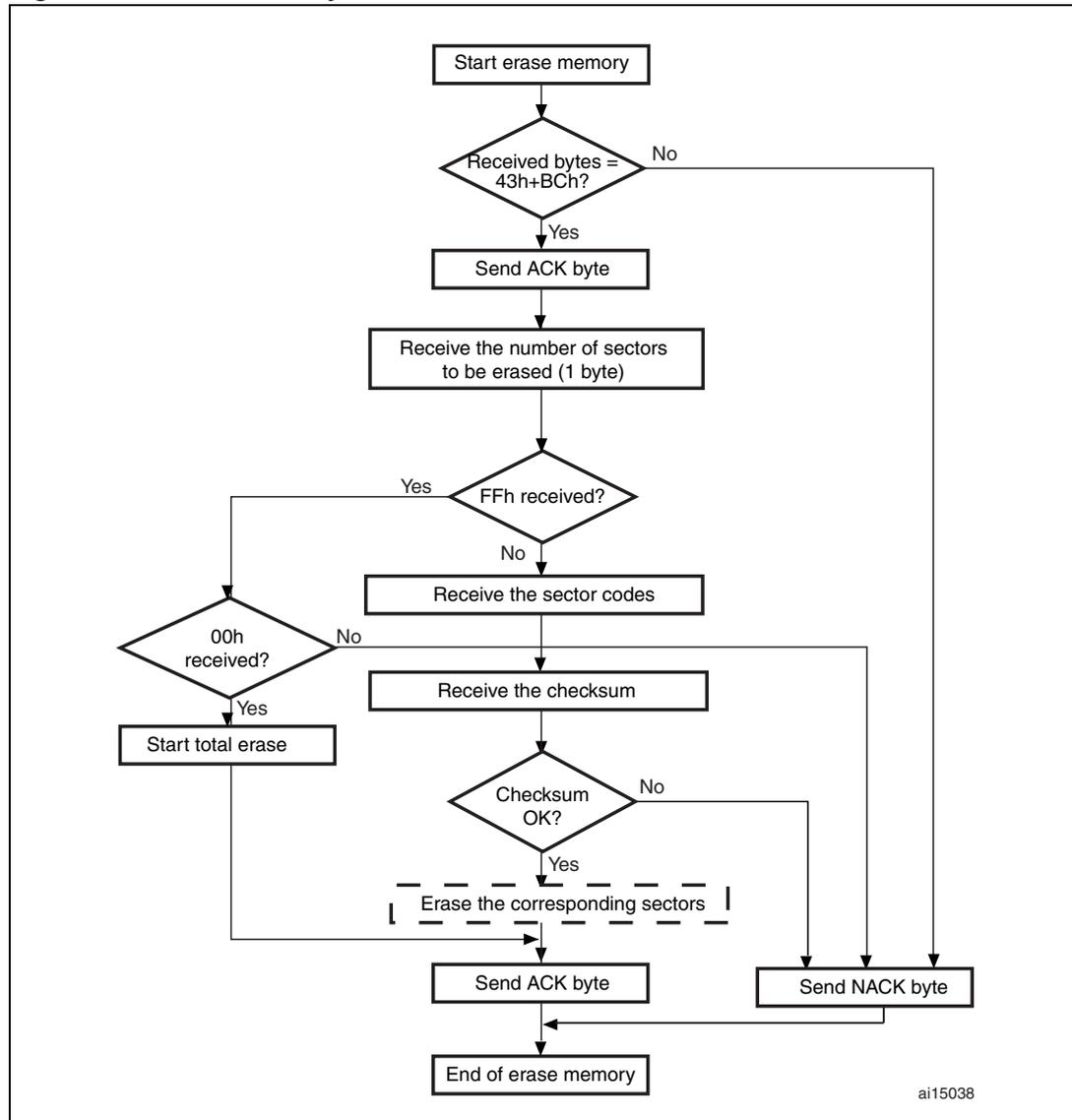


1. When using the erase command via SPI, it is necessary to wait for a brief time interval (see 'delay' in Figure 17 above) before sending the last token byte. This delay interval depends on the SPI baud rate and on the number of sectors to be erased. Delay = 13 * (N + 1)[ms], where 0 ≤ N ≤ 32. N = 32 in the case of total erase.

The host sends the bytes as follows

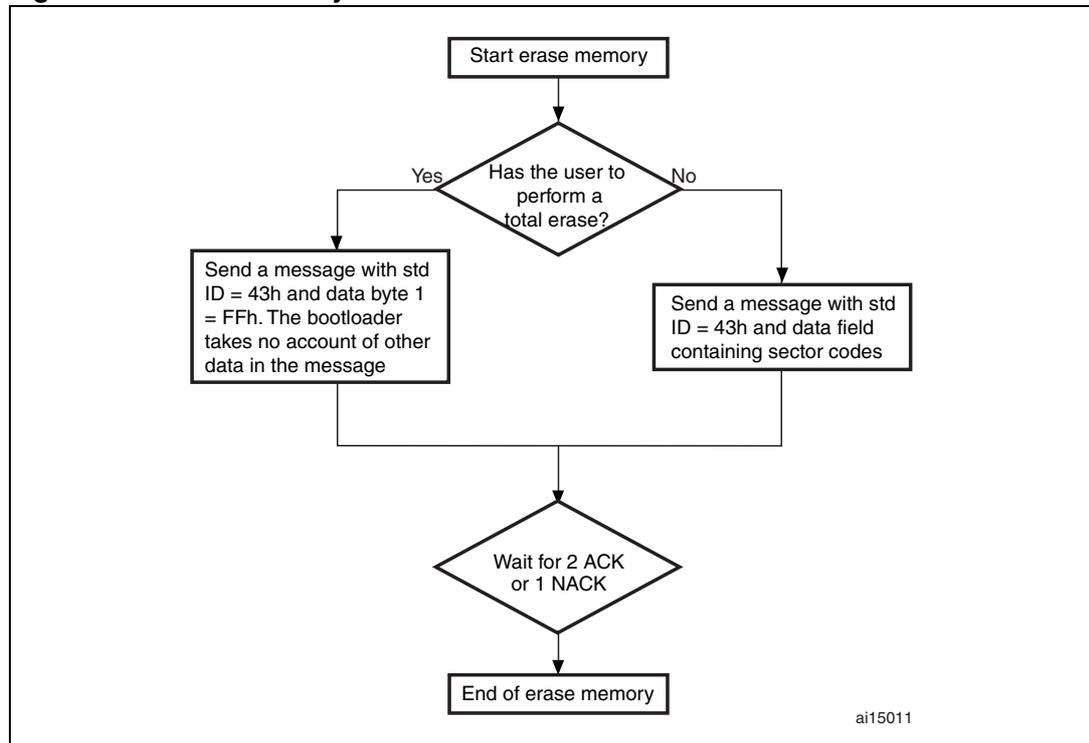
- Byte 1: 43h - Command ID
- Byte 2: BC h - Complement
- Byte 3 (token): XYh; host waits for ACK or NACK
- Byte 4: FFh or number of sectors to be erased (0 ≤ N ≤ 32); If N > 32 a 'cmd_error' occurs.
- Byte 5 or N+1 bytes: 00h or (N+1 bytes and then checksum: XOR (N, N+1 bytes))
- Last byte (token): XYh; host waits for ACK or NACK.

Figure 18. Erase memory command via SPI: Device side



4.3.3 Erase memory command via CAN

Figure 19. Erase memory command via CAN: Host side



1. Memory erased =130 Kbytes for devices with 128 Kbytes of Flash program memory and 2 Kbytes of Data EEPROM and 33 Kbytes for devices with 32 Kbytes of Flash program memory and 1 Kbyte of Data EEPROM.
2. A sector is 8 blocks i.e. 1 Kbyte for any devices. Therefore, the granularity of the erase command is 1 Kbyte.

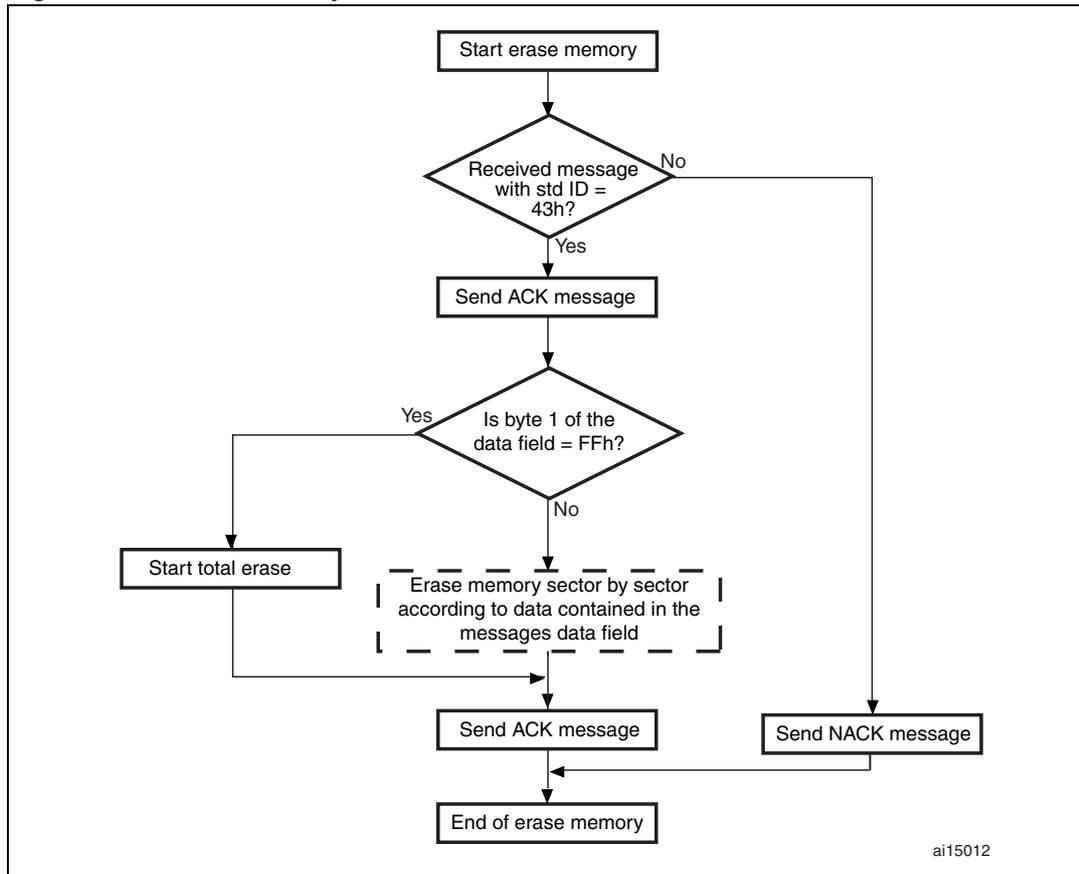
Warning: If the host sends an erase command that includes some correct sector code and one or more forbidden sector codes, the command fails.

The host sends the message as follows

Total erase message: Std ID = 43h, DLC = 01h, data = FFh.

Erase sector by sector message: Std ID = 43h, DLC = 01h to 08h, data = see product datasheet.

Figure 20. Erase memory command via CAN: Device side



ai15012

1. For safety reasons no erase routine is placed into ROM. The 'erase memory sector by sector according to data contained in the messages data field' (see [Figure 20](#)) is performed in RAM. The user, therefore has to download the erase routine in RAM before sending an erase command.

4.4 Write memory command

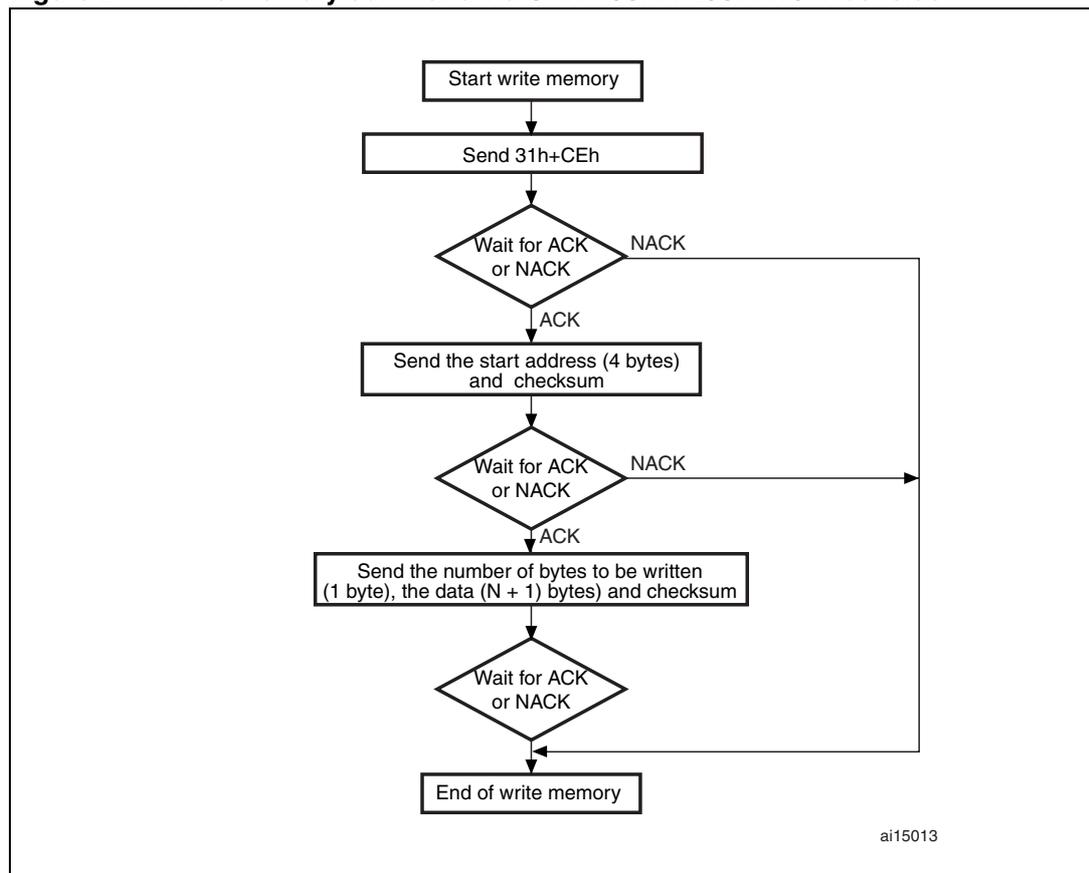
The write memory command allows the host to write data to any memory address (RAM, Flash program memory/data EEPROM or registers) starting from the received address. Incoming data are always written in RAM before being loaded in the memory locations decided by the host. The bootloader then checks whether the host wants to write in RAM or in the Flash program memory/data EEPROM.

The maximum length of the block to be written for the STM8S is 128 data bytes. To write the data in the Flash program memory/data EEPROM memory locations, the bootloader performs two different write operations:

1. **WordWrite/FastWordWrite:** Writes a byte in the Flash program memory/data EEPROM. It is used when the number of bytes (N) sent from the host is less than 128. In this case the bootloader performs the operation N times.
2. **BlockWrite:** Writes a block in the Flash program memory/data EEPROM. It is used when the number of bytes (N) sent from the host is 128 and the destination address is an integer module of 128. In other words, to use this operation, the block sent from the host has to be aligned with a memory block.

4.4.1 Write memory command via UART1/UART2/UART3

Figure 21. Write memory command via UART1/UART2/UART3: Host side

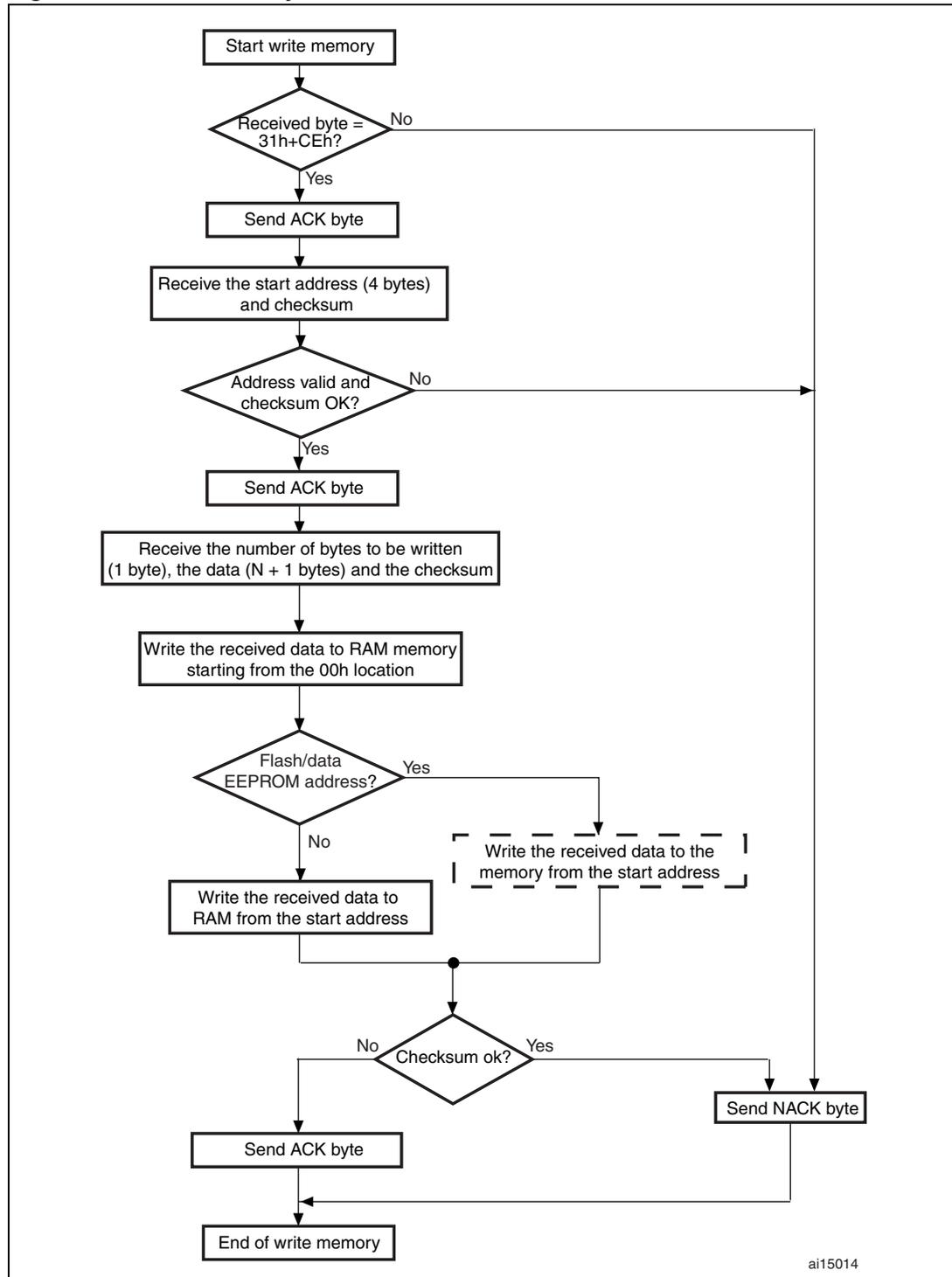


1. See product datasheet for valid addresses. If the start address is invalid, an add_error occurs (see [Table 6: Error table on page 55](#)).

The host sends the bytes as follows

Byte 1:	31h	- Command ID
Byte 2:	CEh	- Complement
Bytes 3-6:	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 received ($0 < N \leq 127$); if $N > 127$ a <code>cmd_error</code> occurs in the bootloader.	
	N+1 data bytes: Max 128 bytes	
Checksum byte:	XOR (N, N+1 databytes).	

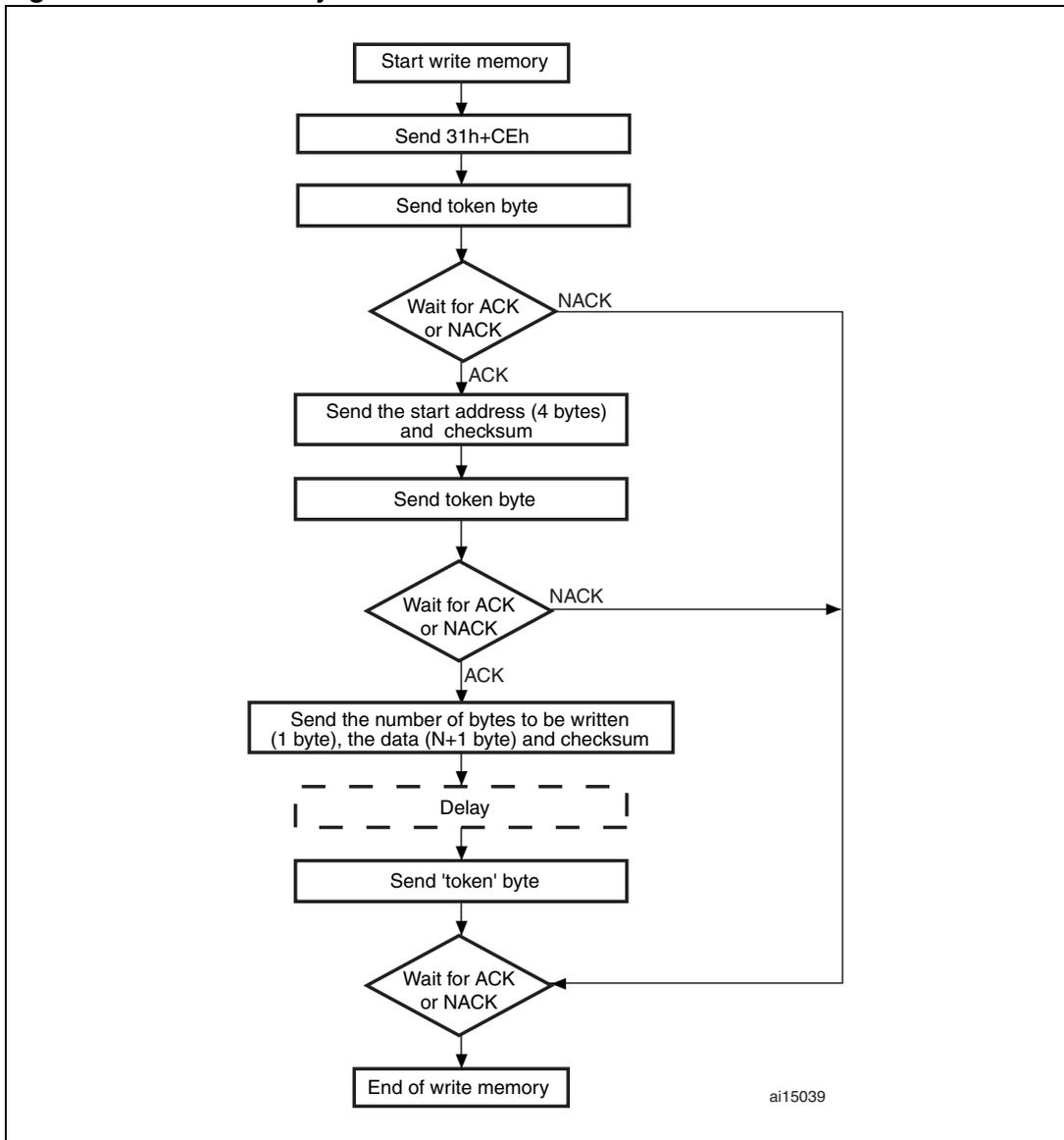
Figure 22. Write memory command via UART1/UART2/UART3: Device side



1. For safety reasons no write routine is placed into ROM. The 'write the received data to RAM from the start address' (see [Figure 22](#)) is performed in RAM. The user, therefore has to download the write routine in RAM before sending a write command.

4.4.2 Write memory command via SPI

Figure 23. Write memory command via SPI: Host side



The 'delay' is calculated as follows:

$$\text{Delay} = 8.45 * (N + 1) \text{ [ms]}$$

Where $0 \leq N \leq 127$.

Table 4. Examples of delay

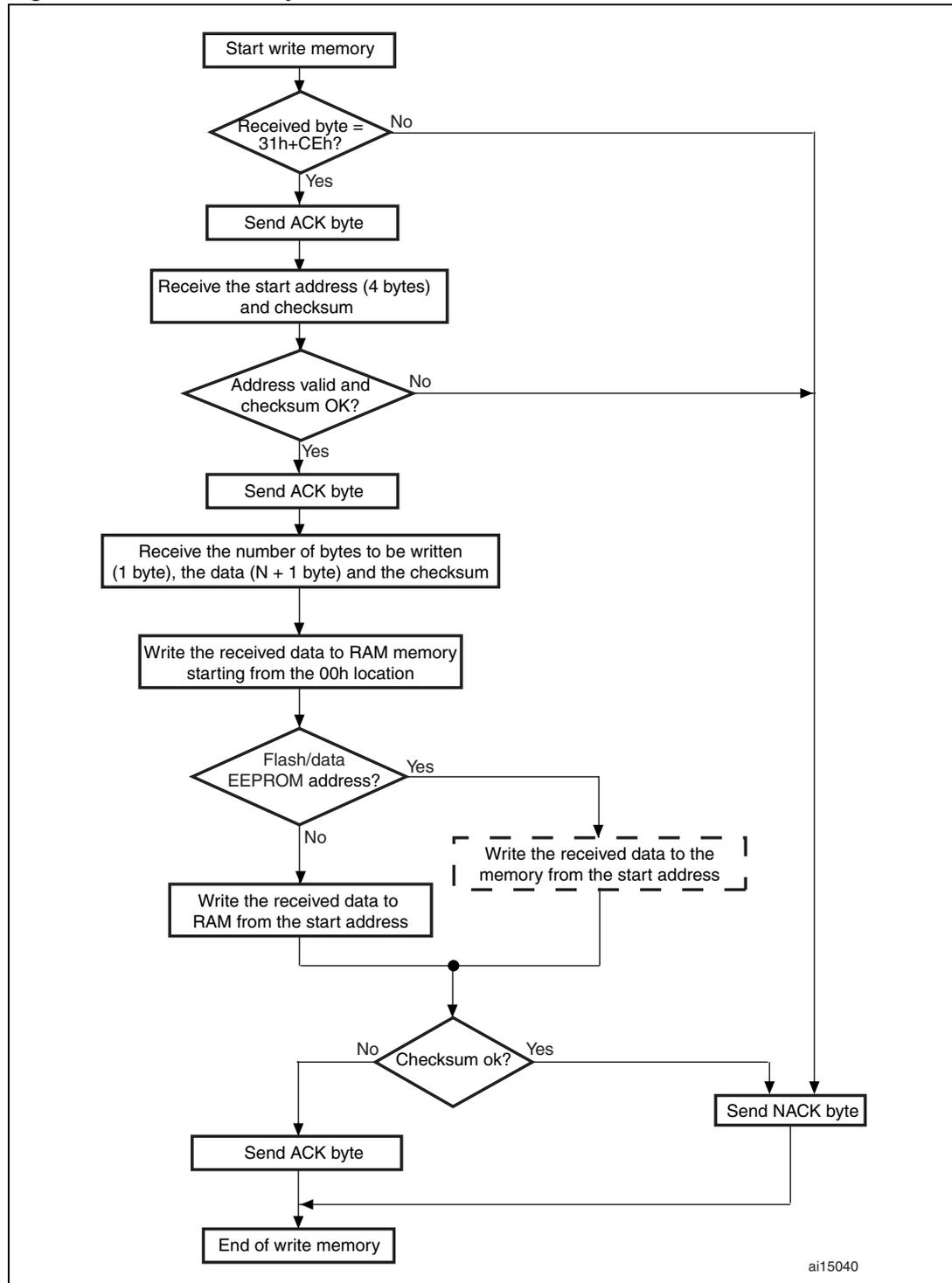
Write command	Delay (ms)
128 bytes aligned with a memory block	8.45
128 bytes not aligned with a memory block	1082

The host sends the bytes as follows

- Byte 1: 31h - Command ID
- Byte 2: CEh - Complement
- Byte 3 (token): XY; host waits for ACK or NACK
- Bytes 4 to 7: The start address
 - Byte 4 = MSB
 - Byte 7 = LSB
- Byte 8: Checksum = XOR (byte 3, byte 4, byte 5, byte 6)
- Byte 9 (token): XY; host waits for ACK or NACK
- Byte 10: The number of bytes to be received ($0 < N \leq 127$);
If $N > 127$ a cmd_error occurs in the bootloader.
N+1 data bytes: Max 128 bytes
- Checksum byte: XOR (N, N+1 databytes).

before sending the token byte, the host must wait for the bootloader to finish writing all data into the memory.
- Last byte (token): XYh; host waits for ACK or NACK.

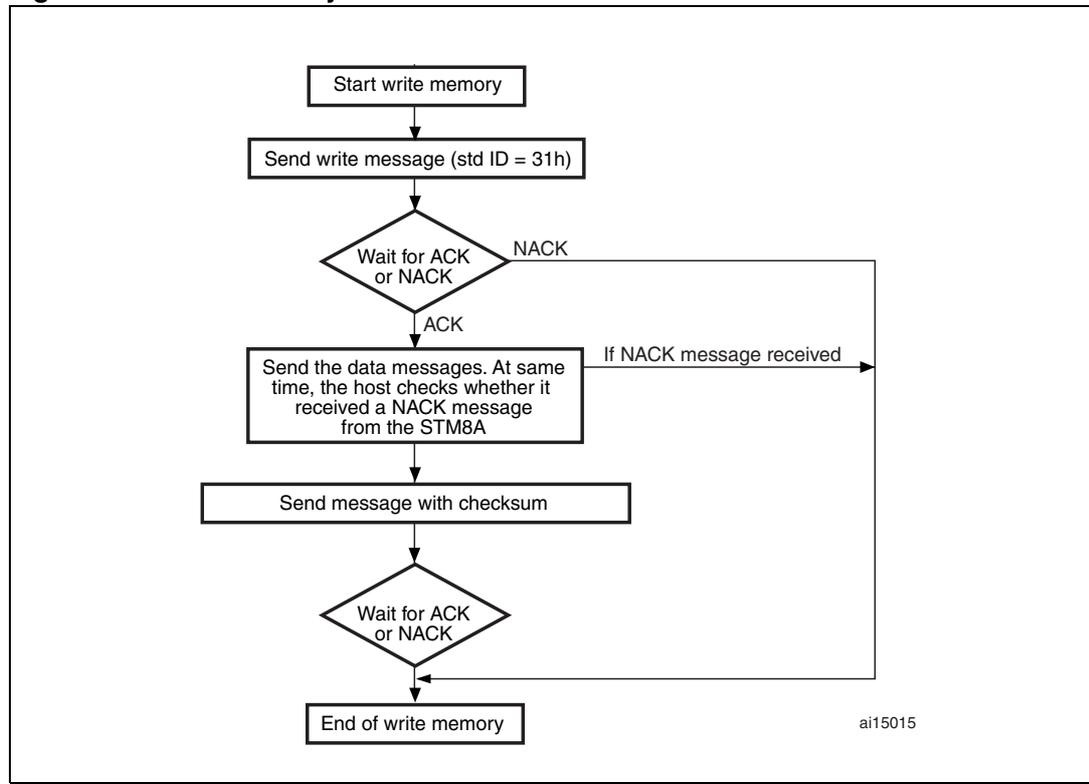
Figure 24. Write memory command via SPI: Device side



1. For safety reasons no write routine is placed into ROM. The 'write the received data to RAM from the start address' (see Figure 22) is performed in RAM. The user, therefore has to download the write routine in RAM before sending a write command.

4.4.3 Write memory command via CAN

Figure 25. Write memory command via CAN: Host side



1. See product datasheet for valid addresses. If the bootloader receives an invalid address, an add_error occurs (see [Table 6: Error table on page 55](#)).

The host sends the messages as follows

Command message: Std ID = 31h, DLC = 05h, data = MSB, XXh, YYh, LSB, N, $0 < N \leq 127$). If $N > 127$ a cmd_error occurs in the bootloader.

Data message_1: Std ID = 04h, DLC_1 = 1 to 8, data = byte_11, ... byte_18

Data message_2: Std ID = 04h, DLC_2 = 1 to 8, data = byte_21, ... byte_28

Data message_3: Std ID = 04h, DLC_3 = 1 to 8, data = byte_31, ... byte_38

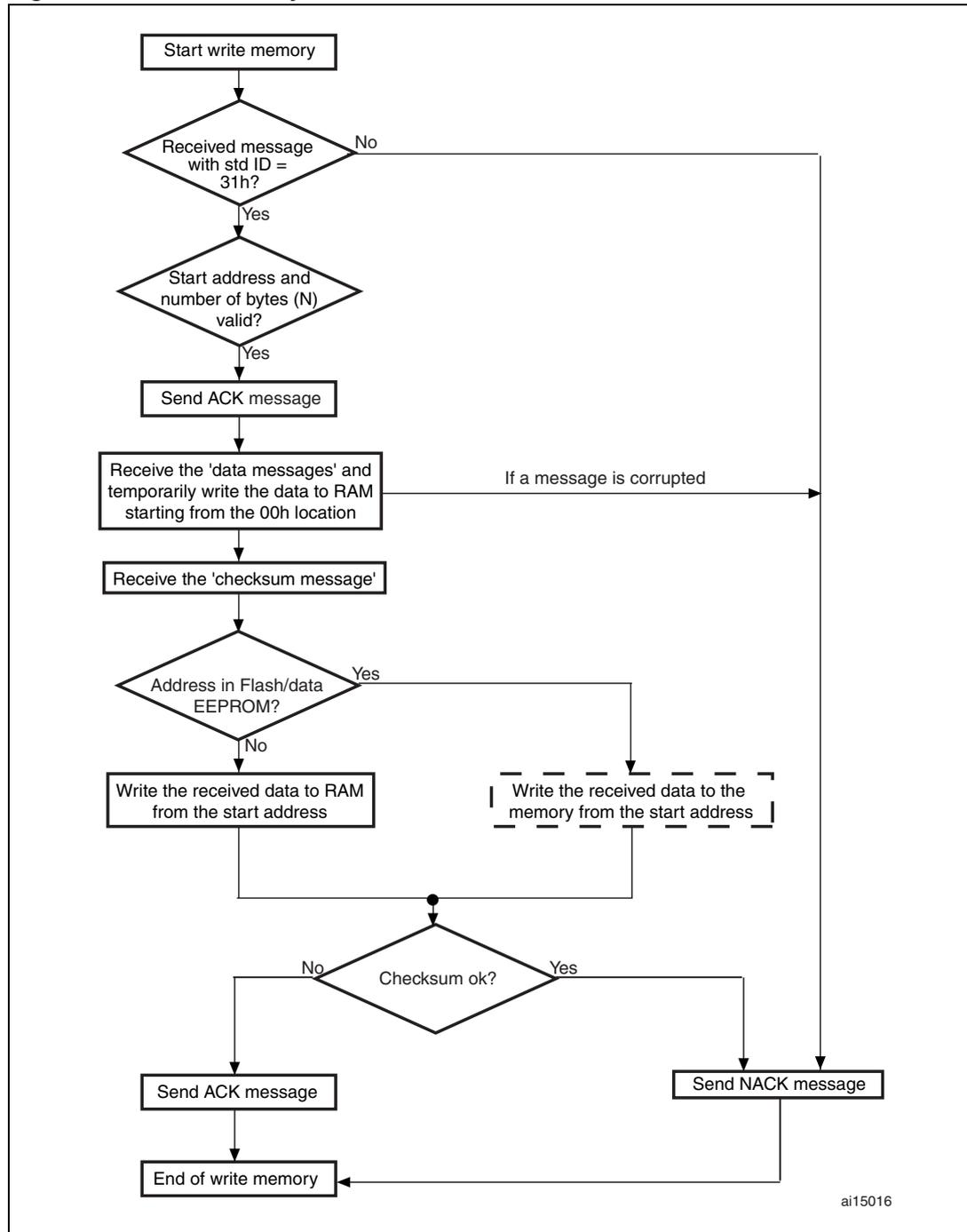
...

Data message_M: Std ID = 04h, DLC_M = 1 to 8, data = byte_m1, ..., byte_M8

Checksum message: Std ID = 04h, DLC = 1, data = XOR (N, N+1 data bytes)

- Note:*
- 1 *DLC_1 + DLC_2 + ... DLC_M = 128 maximum*
 - 2 *The bootloader does not check the standard ID of the data and checksum messages. Therefore, an ID from 0h to FFh can be used. It is recommended to use 04h.*

Figure 26. Write memory command via CAN: Device side



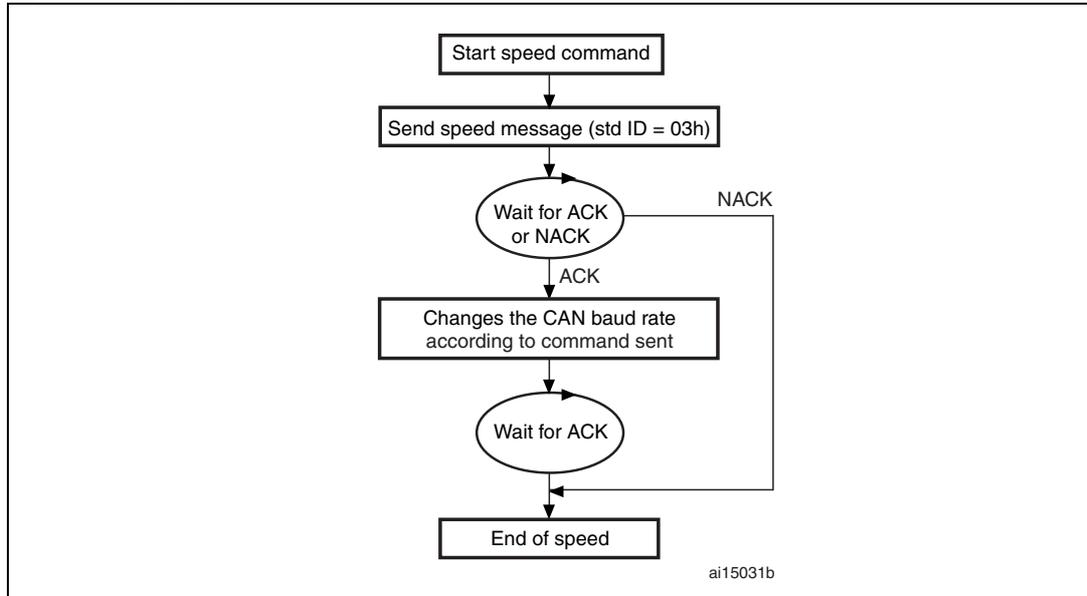
1. For safety reasons no write routine is placed into ROM. The 'write the received data to the Flash program memory/data EEPROM from the start address' (see [Figure 26](#)) is performed in RAM. The user, therefore has to download the write routine in RAM before sending a write command.
2. See product datasheet for valid addresses. If the bootloader receives an invalid address, an add_error occurs (see [Table 6: Error table on page 55](#)).

4.5 Speed command

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

4.5.1 Speed command via CAN

Figure 27. Speed command via CAN: Host side



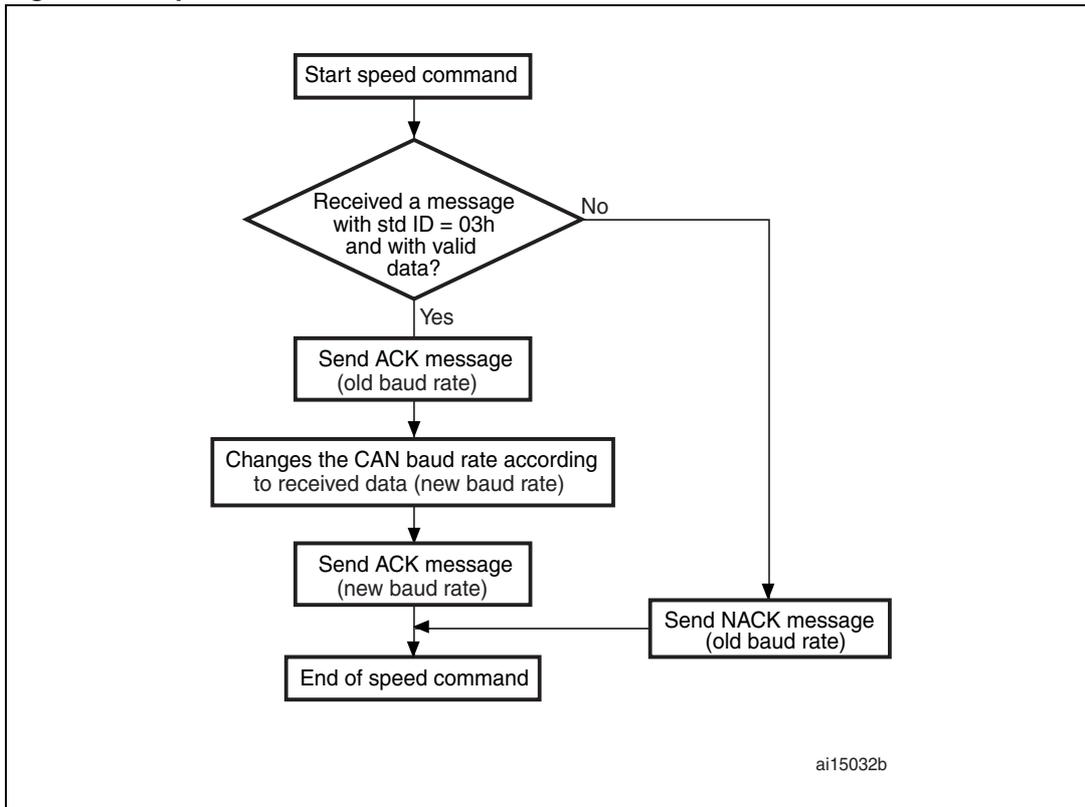
1. After setting the new baud rate, the bootloader sends the ACK message. Therefore, the host sets its baud rate while waiting for the ACK.

The host sends the message as follows

Command message: Std ID = 03h, DLC = 01h, data = XXh where XXh assumes the following values depending on the baud rate to be set:

- 01h -> baud rate = 125 kbps
- 02h -> baud rate = 250 kbps
- 03h -> baud rate = 500 kbps
- 04h -> baud rate = 1 Mkbps

Figure 28. Speed command via CAN: Device side

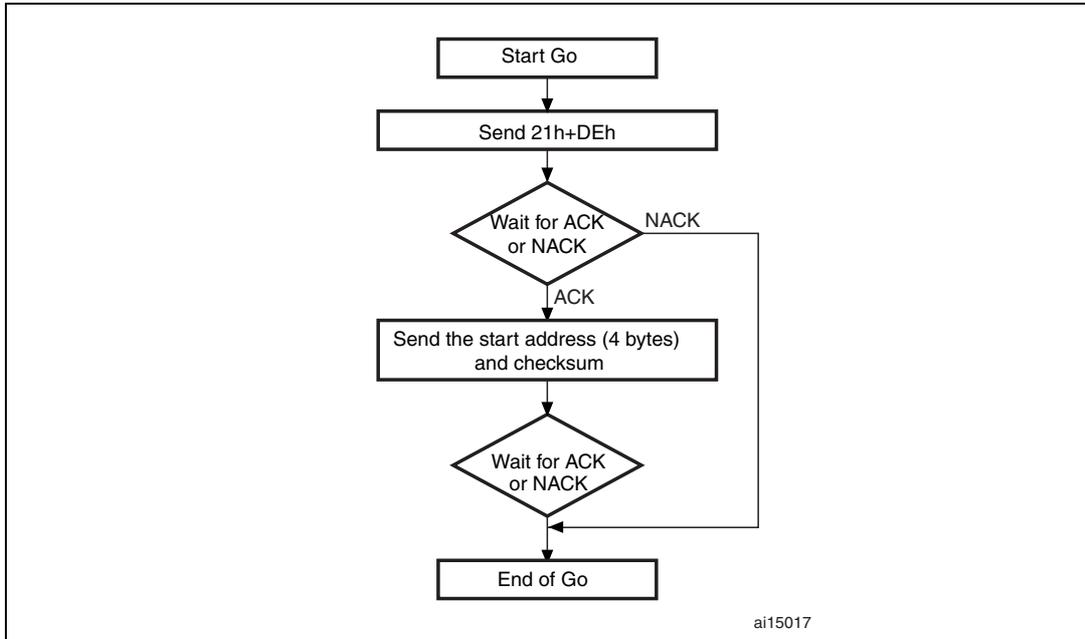


4.6 Go command

The go command is used to execute the downloaded code or any other code by branching to an address specified by the host.

4.6.1 Go command via UART1/UART2/UART3

Figure 29. Go command via UART1/UART2/UART3: Host side



1. The valid addresses are RAM, Flash program memory/data EEPROM and register addresses (see product datasheet). If the bootloader receives an invalid address, an 'add error' occurs (see [Table 6: Error table on page 55](#)).

The host sends the bytes as follows

Byte 1: 21h - Command ID

Byte 2: DEh - Complement

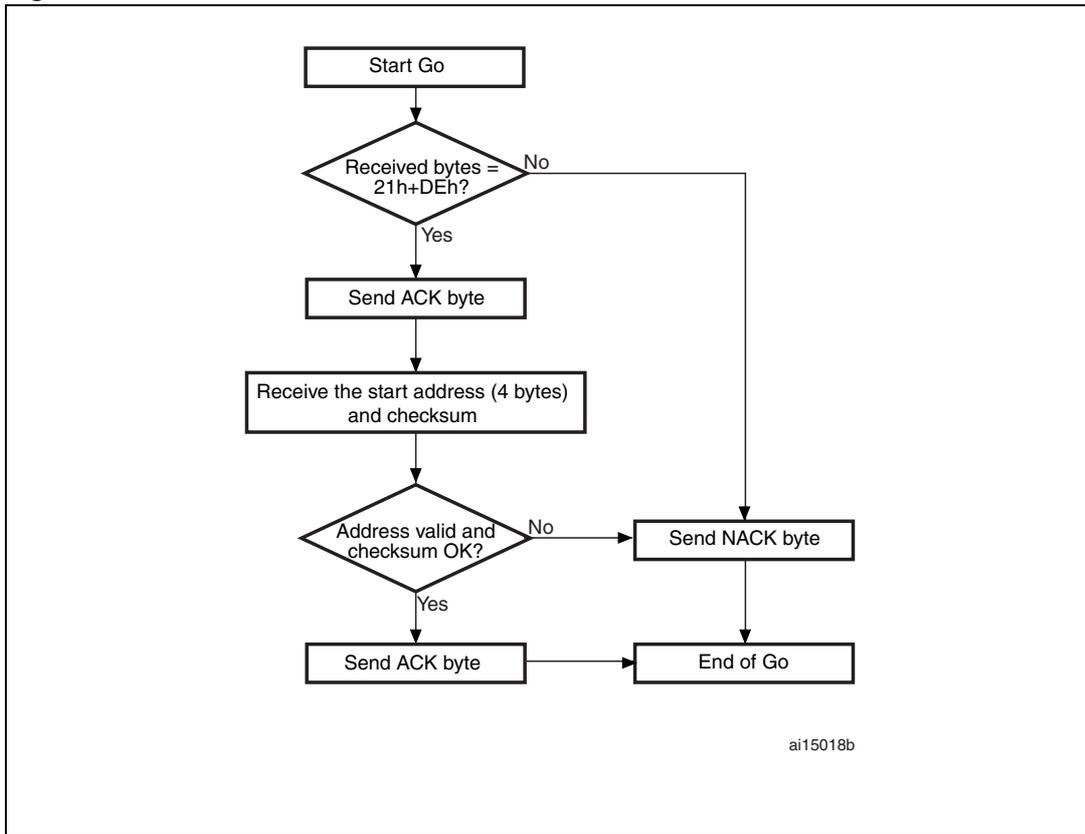
Bytes 3-6: The start address

Byte 3 = MSB

Byte 6 = LSB

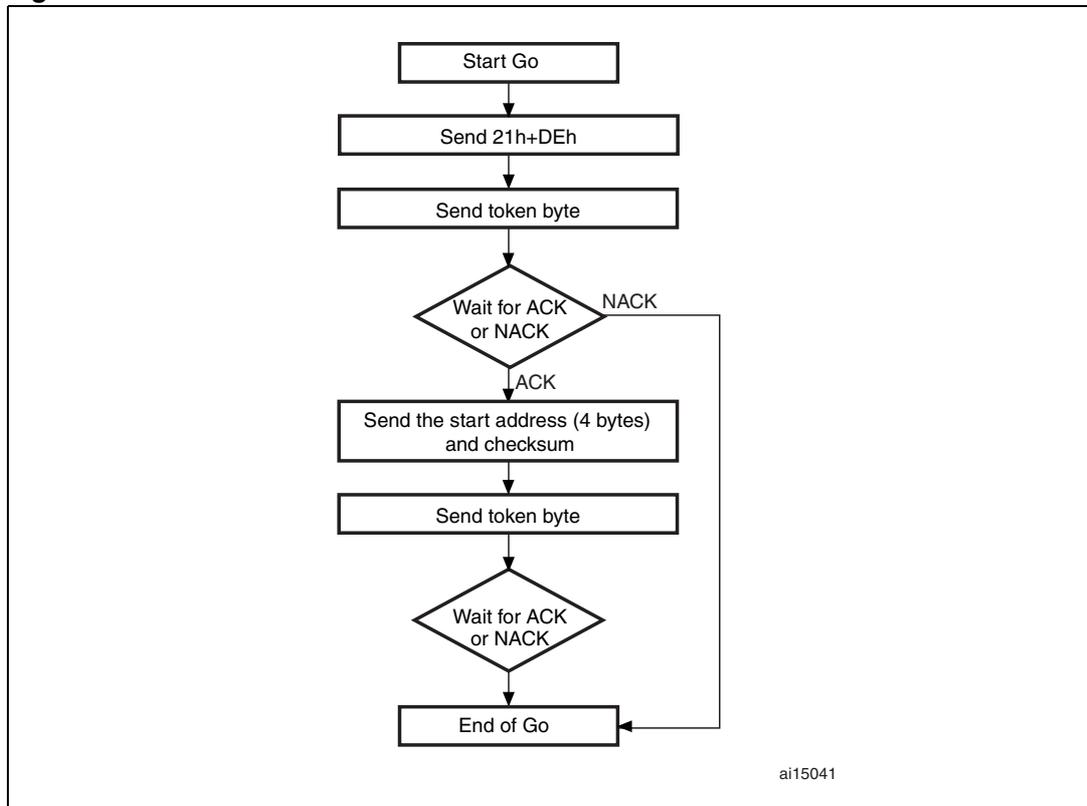
Byte 7: Checksum = XOR (byte 3, byte 4, byte 5, byte 6).

Figure 30. Go command via UART1/UART2/UART3: Device side



4.6.2 Go command via SPI

Figure 31. Go command via SPI: Host side

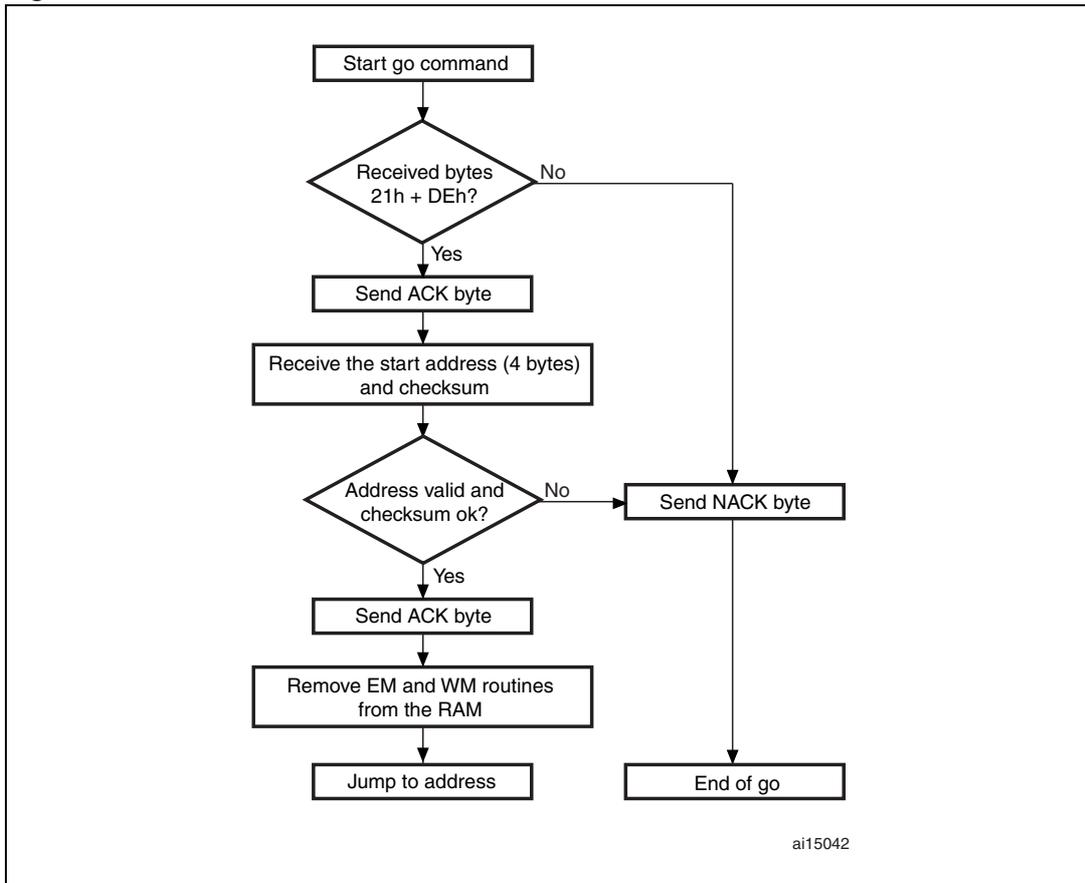


1. The valid addresses are RAM, Flash program memory/data EEPROM and register addresses (see product datasheet). If the bootloader receives an invalid address, an 'add error' occurs (see [Table 6: Error table on page 55](#)).

The host sends the bytes as follows

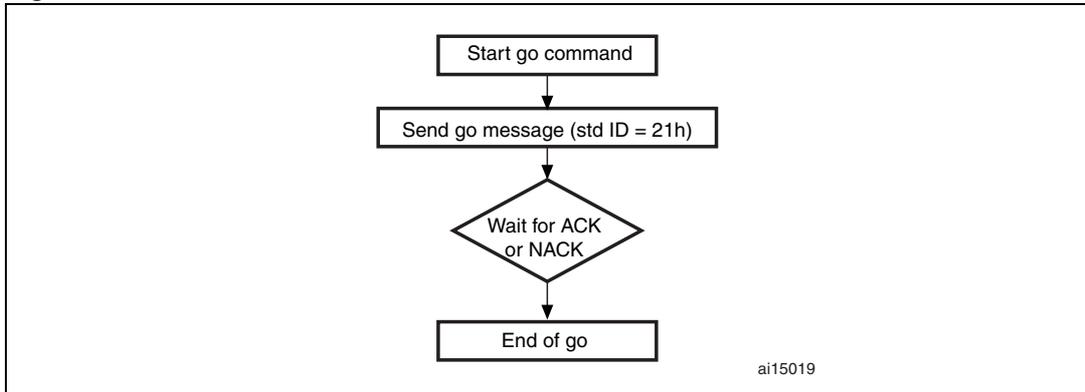
- Byte 1: 21h - Command ID
- Byte 2: DEh - Complement
- Byte 3 (token): XYh; host waits for ACK or NACK
- Bytes 4 to 7: The start address
- Byte 4 = MSB
- Byte 7 = LSB
- Byte 8: Checksum = XOR (byte 3, byte 4, byte 5, byte 6)
- Byte 9 (token): XYh; host waits for ACK or NACK.

Figure 32. Go command via SPI: Device side



4.6.3 Go command via CAN

Figure 33. Go command via CAN: Host side

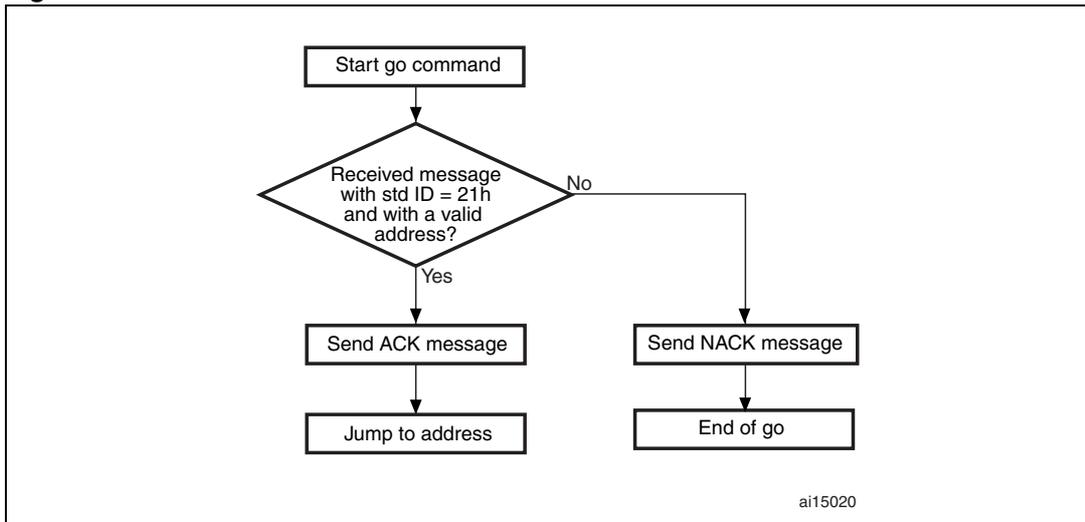


1. See product datasheet for valid addresses.

The host sends the bytes as follows

Go command message: Std ID = 21h, DLC = 04h, data = MSB, XXh, YYh, LSB

Figure 34. Go command via CAN: Device side



4.7 Sector codes

Table 5. STM8S sector codes

Sector code	Flash program memory/ data EEPROM (STM8S20xxx devices)	Flash program memory/ data EEPROM (STM8S105xx devices)
00h	8000h -> 83FFh	8000h -> 83FFh
01h	8400h -> 87FFh	8400h -> 87FFh
02h	8800h -> 8BFFh	8800h -> 8BFFh
03h	8C00h -> 8FFFh	8C00h -> 8FFFh
04h	9000h -> 93FFh	9000h -> 93FFh
05h	9400h -> 97FFh	9400h -> 97FFh
06h	9800h -> 9BFFh	9800h -> 9BFFh
07h	9C00h -> 9FFFh	9C00h -> 9FFFh
08h	A000h -> A3FFh	A000h -> A3FFh
09h	A400h -> A7FFh	A400h -> A7FFh
0Ah	A800h -> ABFFh	A800h -> ABFFh
0Bh	AC00h -> AFFFh	AC00h -> AFFFh
0Ch	B000h -> B3FFh	B000h -> B3FFh
0Dh	B400h -> B7FFh	B400h -> B7FFh
0Eh	B800h -> BBFFh	B800h -> BBFFh
0Fh	BC00h -> BFFFh	BC00h -> BFFFh
10h	C000h -> C3FFh	C000h -> C3FFh
11h	C400h -> C7FFh	C400h -> C7FFh
12h	C800h -> CBFFh	C800h -> CBFFh
13h	CC00h -> CFFFh	CC00h -> CFFFh
14h	D000h -> D3FFh	D000h -> D3FFh
15h	D400h -> D7FFh	D400h -> D7FFh
16h	D800h -> DBFFh	D800h -> DBFFh
17h	DC00h -> DFFFh	DC00h -> DFFFh
18h	E000h -> E3FFh	E000h -> E3FFh
19h	E400h -> E7FFh	E400h -> E7FFh
1Ah	E800h -> EBFFh	E800h -> EBFFh
1Bh	EC00h -> EFFFh	EC00h -> EFFFh
1Ch	F000h -> F3FFh	F000h -> F3FFh
1Dh	F400h -> F7FFh	F400h -> F7FFh
1Eh	F800h -> FBFFh	F800h -> FBFFh

Table 5. STM8S sector codes (continued)

Sector code	Flash program memory/ data EEPROM (STM8S20xxx devices)	Flash program memory/ data EEPROM (STM8S105xx devices)
1Fh	FC00h -> FFFFh	FC00h -> FFFFh
20h	10000h -> 103FFh	4000h -> 43FFh
21h	10400h -> 107FFh	
22h	10800h -> 10BFFh	
23h	10C00h -> 10FFFh	
24h	11000h -> 113FFh	
25h	11400h -> 117FFh	
26h	11800h -> 11BFFh	
27h	11C00h -> 11FFFh	
28h	12000h -> 123FFh	
29h	12400h -> 127FFh	
2Ah	12800h -> 12BFFh	
2Bh	12C00h -> 12FFFh	
2Ch	13000h -> 133FFh	
2Dh	13400h -> 137FFh	
2Eh	13800h -> 13BFFh	
2Fh	13C00h -> 13FFFh	
30h	14000h -> 143FFh	
31h	14400h -> 147FFh	
32h	14800h -> 14BFFh	
33h	14C00h -> 14FFFh	
34h	15000h -> 153FFh	
35h	15400h -> 157FFh	
36h	15800h -> 15BFFh	
37h	15C00h -> 15FFFh	
38h	16000h -> 163FFh	
39h	16400h -> 167FFh	
3Ah	16800h -> 16BFFh	
3Bh	16C00h -> 16FFFh	
3Ch	17000h -> 173FFh	
3Dh	17400h -> 177FFh	
3Eh	17800h -> 17BFFh	
3Fh	17C00h -> 17FFFh	

Table 5. STM8S sector codes (continued)

Sector code	Flash program memory/ data EEPROM (STM8S20xxx devices)	Flash program memory/ data EEPROM (STM8S105xx devices)
40h	18000h -> 183FFh	
41h	18400h -> 187FFh	
42h	18800h -> 18BFFh	
43h	18C00h -> 18FFFh	
44h	19000h -> 193FFh	
45h	19400h -> 197FFh	
46h	19800h -> 19BFFh	
47h	19C00h -> 19FFFh	
48h	1A000h -> 1A3FFh	
49h	1A400h -> 1A7FFh	
4Ah	1A800h -> 1ABFFh	
4Bh	1AC00h -> 1AFFh	
4Ch	1B000h -> 1B3FFh	
4Dh	1B400h -> 1B7FFh	
4Eh	1B800h -> 1BBFFh	
4Fh	1BC00h -> 1BFFFh	
50h	1C000h -> 1C3FFh	
51h	1C400h -> 1C7FFh	
52h	1C800h -> 1CBFFh	
53h	1CC00h -> 1CFFFh	
54h	1D000h -> 1D3FFh	
55h	1D400h -> 1D7FFh	
56h	1D800h -> 1DBFFh	
57h	1DC00h -> 1DFFFh	
58h	1E000h -> 1E3FFh	
59h	1E400h -> 1E7FFh	
5Ah	1E800h -> 1EBFFh	
5Bh	1EC00h -> 1EFFFh	
5Ch	1F000h -> 1F3FFh	
5Dh	1F400h -> 1F7FFh	
5Eh	1F800h -> 1FBFFh	
5Fh	1FC00h -> 1FFFFh	
60h	20000h -> 203FFh	

Table 5. STM8S sector codes (continued)

Sector code	Flash program memory/ data EEPROM (STM8S20xxx devices)	Flash program memory/ data EEPROM (STM8S105xx devices)
61h	20400h -> 207FFh	
62h	20800h -> 20BFFh	
63h	20C00h -> 20FFFh	
64h	21000h -> 213FFh	
65h	21400h -> 217FFh	
66h	21800h -> 21BFFh	
67h	21C00h -> 21FFFh	
68h	22000h -> 223FFh	
69h	22400h -> 227FFh	
6Ah	22800h -> 22BFFh	
6Bh	22C00h -> 22FFFh	
6Ch	22000h -> 223FFh	
6Dh	22400h -> 227FFh	
6Eh	23800h -> 23BFFh	
6Fh	23C00h -> 23FFFh	
70h	24000h -> 243FFh	
71h	24400h -> 247FFh	
72h	24800h -> 24BFFh	
73h	24C00h -> 24FFFh	
74h	25000h -> 253FFh	
75h	25400h -> 257FFh	
76h	25800h -> 25BFFh	
77h	25C00h -> 25FFFh	
78h	26000h -> 263FFh	
79h	26400h -> 267FFh	
7Ah	26800h -> 26BFFh	
7Bh	26C00h -> 26FFFh	
7Ch	27000h -> 273FFh	
7Dh	27400h -> 277FFh	
7Eh	27800h -> 27BFFh	
7Fh	27C00h -> 27FFFh	
80h	4000h -> 43FFh	
81h	4400h -> 47FFh	

5 Software model

The boot code has been designed with the same logical protocol for exchanging command frames between host and any STM8S device.

The boot code can download up to 128 bytes at a time. The first 130 bytes of RAM (from 00h) store data coming from the serial interface, thus allowing the boot to use the stack. The 25 bytes following the first 130 bytes are used for the bootloader's variables.

Note: *Unused (empty) bootloader ROM is filled with an opcode that is not allowed (71h). If, for any reason (i.e. noise), the core starts to execute in the 71h area, an illegal opcode is returned and consequently a reset. This prevents the bootloader falling into an infinite loop with no reset in the event that it jumps in the 'empty' locations. Thus normal execution can resume.*

Caution: To erase or program the Flash program memory, the respective routines must be downloaded in RAM starting at 00A0h. This is imperative because the routines are consecutive with no empty memory locations between them.

The routines are included in the following files:

- STM8S20xxx devices: E_W_ROUTINES_128K.s19
- STM8S105xx devices: E_W_ROUTINES_32K.s19

Note: *To execute any of the commands (get, read, erase, write, speed, and go), the bootloader uses part of the RAM memory. Therefore, it is forbidden to run write commands (with the exception of write commands that are useful for downloading routines to erase and program the Flash program memory) with destination addresses in the following RAM memory locations:*

- STM8S20xxx devices: 0h to 01FFh
- STM8S105xx devices: 0h to 01CFh

6 Error management

[Table 6](#) describes the error type and the bootloader behavior.

Table 6. Error table

Error	Description	Bootloader actions
cmd_error	If a denied command is received If a parity error occurs during command transmission If an error occurs during the command execution See Table 3: Bootloader commands on page 13	Sends NACK byte and goes back to command checking
add_error	If a received command contains a denied destination address (for information on valid address ranges, refer to the product datasheet)	Sends NACK byte and goes back to command checking

7 Programming time

The programming time depends on the baud rate of the peripheral.

- UART1/UART2/UART3: 128 kbps, 256 kbps, and 500 kbps
- SPI: 125 kbps, 250 kbps, 500 kbps, and 1 Mbps
- CAN: 125 kbps, 250 kbps, 500 kbps, and 1 Mbps

- Note:*
- 1 The programming time of the SPI is measured based on 32 Kbytes instead of 48 Kbytes because the boot code via the SPI is implemented on the STM8S 32 Kbyte device only.
 - 2 After a go command, the bootloader removes the erase and write routines from the RAM memory before jumping to the host address. It takes about 150 μ s to remove these routines.

Table 7, Table 8, and Table 9 show the programming times for the UART1/UART2/UART3, SPI, and CAN respectively.

Table 7. UART1/UART2/UART3 programming times

Time to load Kbytes/block/bytes in the Flash program memory	Baud rate (bps)		
	128000	256000	500000
48 Kbytes	7.73 s	5.34 s	4.08 s
1 block	20.13 ms	13.53 ms	10.65 ms
1 byte	7.52 ms	6.93 ms	6.65 ms

Table 8. SPI programming time

Time to load Kbytes/block/bytes in the Flash program memory	Baud rate			
	125 kbps	250 kbps	500 kbps	1 Mbps
32 Kbytes	4.46 s	3.34 s	2.81 s	2.55 s
1 block	17.44 ms	13.04 ms	11.01 ms	9.95 ms
1 byte	8.52 ms	8.12 ms	7.93 ms	7.84 ms

Table 9. CAN programming time

Time to load Kbytes/block/bytes in the Flash program memory	Baud rate			
	125 kbps	250 kbps	500 kbps	1 Mbps
48 Kbytes	9.50 s	6.40 s	4.85 s	4.07 s
1 block	24.73 ms	16.66 ms	12.63 ms	10.60 ms
1 byte	9.23 ms	8.53 ms	7.91 ms	7.68 ms

8 Revision history

Table 10. Document revision history

Date	Revision	Changes
15-Dec-2008	1	Initial release

Please Read Carefully:

Information in this document is provided solely in connection with ST products. STMicroelectronics NV and its subsidiaries ("ST") reserve the right to make changes, corrections, modifications or improvements, to this document, and the products and services described herein at any time, without notice.

All ST products are sold pursuant to ST's terms and conditions of sale.

Purchasers are solely responsible for the choice, selection and use of the ST products and services described herein, and ST assumes no liability whatsoever relating to the choice, selection or use of the ST products and services described herein.

No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted under this document. If any part of this document refers to any third party products or services it shall not be deemed a license grant by ST for the use of such third party products or services, or any intellectual property contained therein or considered as a warranty covering the use in any manner whatsoever of such third party products or services or any intellectual property contained therein.

UNLESS OTHERWISE SET FORTH IN ST'S TERMS AND CONDITIONS OF SALE ST DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY WITH RESPECT TO THE USE AND/OR SALE OF ST PRODUCTS INCLUDING WITHOUT LIMITATION IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE (AND THEIR EQUIVALENTS UNDER THE LAWS OF ANY JURISDICTION), OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

UNLESS EXPRESSLY APPROVED IN WRITING BY AN AUTHORIZED ST REPRESENTATIVE, ST PRODUCTS ARE NOT RECOMMENDED, AUTHORIZED OR WARRANTED FOR USE IN MILITARY, AIR CRAFT, SPACE, LIFE SAVING, OR LIFE SUSTAINING APPLICATIONS, NOR IN PRODUCTS OR SYSTEMS WHERE FAILURE OR MALFUNCTION MAY RESULT IN PERSONAL INJURY, DEATH, OR SEVERE PROPERTY OR ENVIRONMENTAL DAMAGE. ST PRODUCTS WHICH ARE NOT SPECIFIED AS "AUTOMOTIVE GRADE" MAY ONLY BE USED IN AUTOMOTIVE APPLICATIONS AT USER'S OWN RISK.

Resale of ST products with provisions different from the statements and/or technical features set forth in this document shall immediately void any warranty granted by ST for the ST product or service described herein and shall not create or extend in any manner whatsoever, any liability of ST.

ST and the ST logo are trademarks or registered trademarks of ST in various countries.

Information in this document supersedes and replaces all information previously supplied.

The ST logo is a registered trademark of STMicroelectronics. All other names are the property of their respective owners.

© 2008 STMicroelectronics - All rights reserved

STMicroelectronics group of companies

Australia - Belgium - Brazil - Canada - China - Czech Republic - Finland - France - Germany - Hong Kong - India - Israel - Italy - Japan - Malaysia - Malta - Morocco - Singapore - Spain - Sweden - Switzerland - United Kingdom - United States of America

www.st.com