Application note

Guidelines for designing a UART bootloader protocol for STM32WL3 MCUs

Introduction

STM32WL3 is a very-low-power sub-1GHz system-on-chip radio, based on the 32-bit Arm® Cortex®-M0+ architecture core. This application note contains the specifications of the STM32WL3 *UART* bootloader.

Table 1. **Applicability**

Reference	Products
STM32WL3	STM32WL3x line

1 General information

This document applies to the STM32WL3 single-core Arm®-based microprocessors.

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

2 UART bootloader configuration

To communicate with the STM32WL3 bootloader, the host *UART* must be configured as follows:

- *UART* data 8-bit
- No parity
- Stop bit 1
- No flow control
- Baud rate range [500 – 460800]

The bootloader is configured to use the following *UART* pins:

- For QFN48 package:
  – *UART Rx* = PA15
  – *UART Tx* = PA1
- For QFN32 package:
  – *UART Rx* = PB14
  – *UART Tx* = PA1

Note: Bootloader is hardware activated by pulling PA10 high during hardware reset (otherwise, the application residing in flash memory is launched)

3 UART bootloader activation

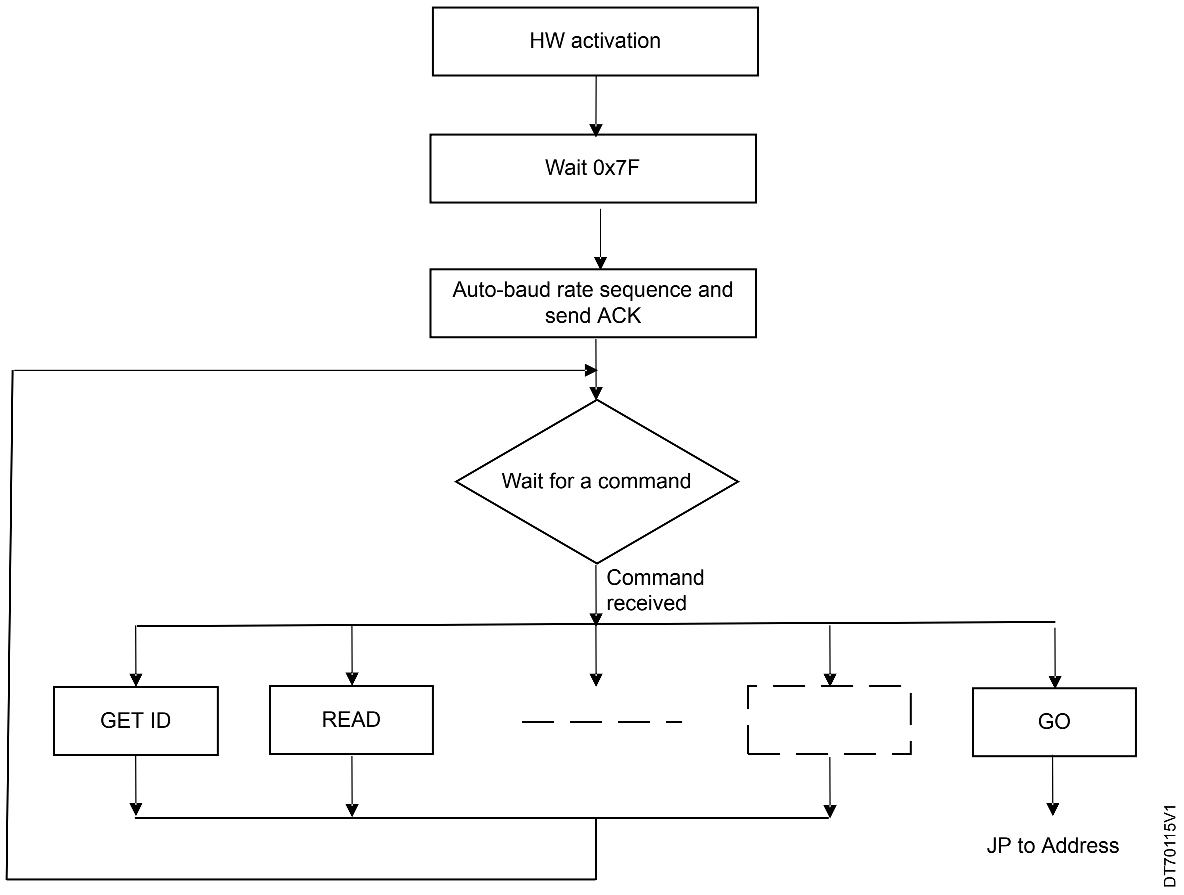
The STM32WL3 bootloader is activated by pulling PA10 high on device hardware reset. Once the bootloader is activated, the firmware initializes the STM32WL3 serial interface and starts a procedure to auto-detect the host *UART* baud rate and begins scanning the USART RX line pin, waiting for the 0x7F data frame from the host. This frame is composed of :

- One start bit
- 0x7F data bits
- No parity bit
- One stop bit.

Once the baud rate is calculated, an acknowledge byte (0x79) is returned to the host, which signals that the STM32WL3 is ready to receive commands.

The *UART* bootloader process is illustrated in the figure below.

Figure 1. UART bootloader for STM32WL3 devices



4 UART bootloader commands

The table below lists the supported commands, fully detailed in the following subsections.

Table 2. **STM32WL3 UART bootloader commands**

Command	Command code	Command description
Get list command	0x00	Gets the version and the commands supported by the current version of the bootloader
Get version command	0x01	Gets the bootloader version
Get ID command	0x02	Gets the chip *ID*
Read memory command	0x11(1)	Reads up to 256 bytes of memory starting from an address specified by the application
Go command	0x21(1)	Jumps to the user application code located in the internal flash memory or in RAM
Write memory command	0x31(1)	Writes up to 256 bytes to *RAM* or flash memory starting from an address specified by the application
Erase memory command	0x43	Erases part or all the flash memory pages
Readout protect command	0x82	Enables the read protection
Readout unprotect command	0x92	Disables the read protection
OTP write command	0xA2	Write a 32-bit word in the OTP sector

1. *This command is disabled when readout protection is enabled.*

All communication from the host to the STM32WL3 device is safe because it is verified by the following processes:

- *checksum*: A byte containing the computed *XOR* of all previous bytes is added to the end of each communication called a checksum byte. By XORing all received bytes, data plus checksum, the result at the end of the packet must be 0x00
- For each bootloader command, the host sends a byte and its complement
- Each packet is either accepted (*ACK* answer) or discarded (*NACK* answer):
  - *ACK* = 0x79
  - *NACK* = 0x1F

4.1 Get list command

The *Get list command* command returns the version of the bootloader and the supported commands.

When the STM32WL3 bootloader receives the *Get list command*, it transmits the bootloader version and the supported command codes to the host. Figure 2 illustrates the host side process and Figure 3 the device side process.

Figure 2. **Get list command: host side**

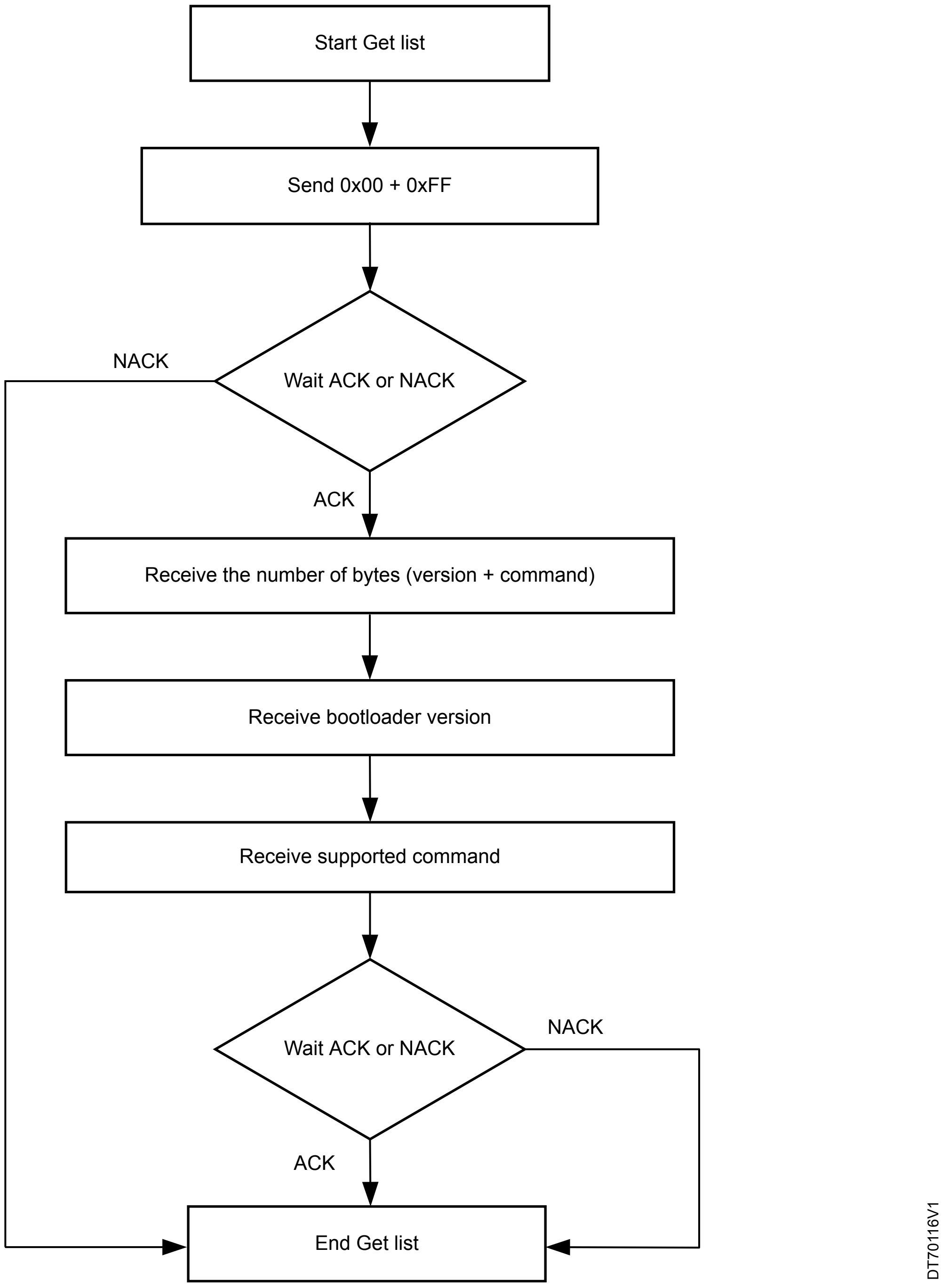
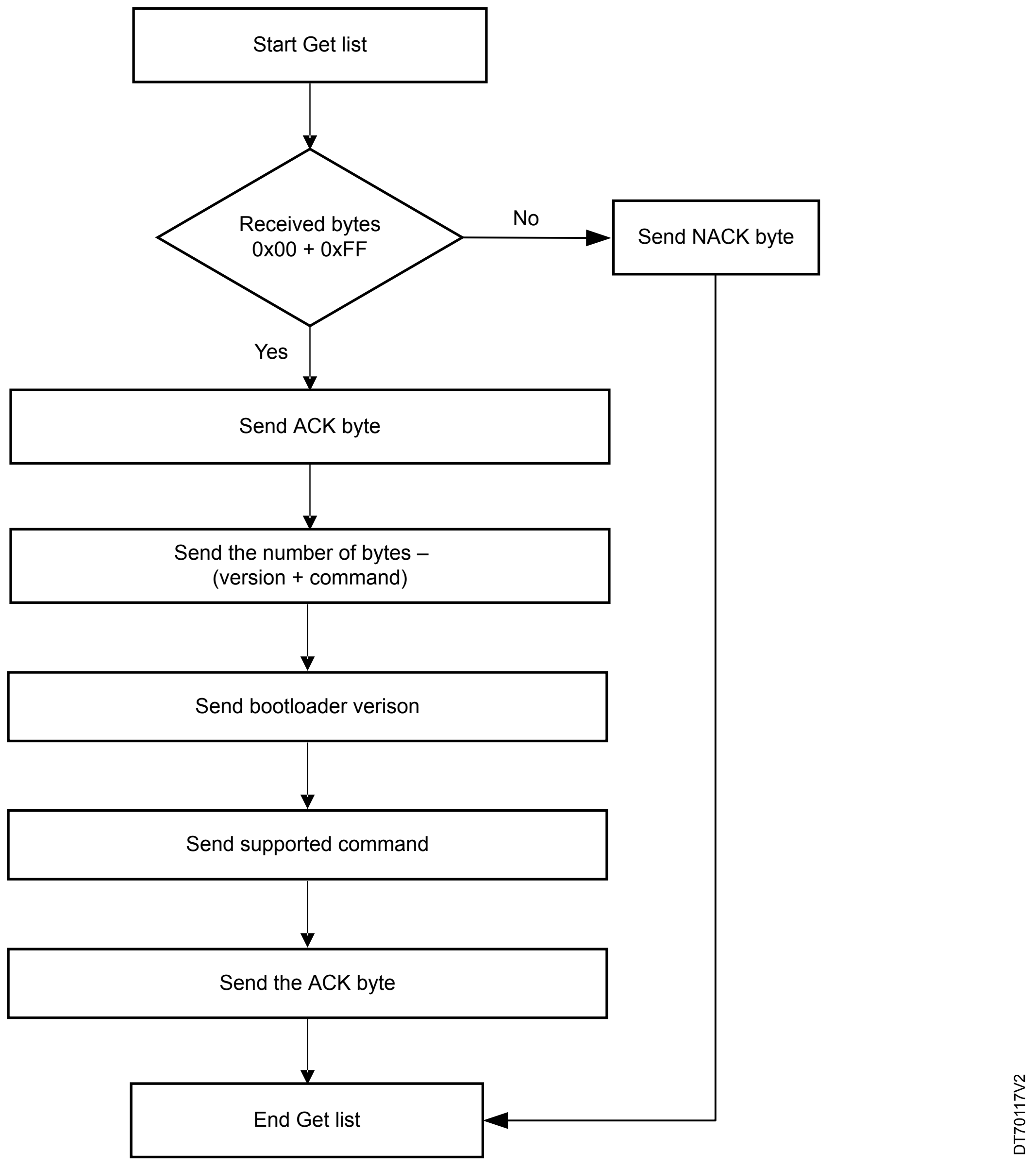

Start Get list
Send 0x00 + 0xFF
NACK
Wait ACK or NACK
ACK
Receive the number of bytes (version + command)
Receive bootloader version
Receive supported command
Wait ACK or NACK
NACK
ACK
End Get list
DT70116V1

Figure 3. Get list command: device side



The STM32WL3 sends the bytes as described in the following table.

Table 3. **Get list command description**

Byte number	Value	Description
Byte 1	0x79	*ACK*
	0x1F	*NACK* (the command is abandoned)
Byte 2	N = 9	This defines the number of bytes to follow byte 1 except current and *ACK*s
Byte 3	0 < Version <= 255	Bootloader version For example: 0x01 = version 1.0
Byte 4	0x00	Get list command
Byte 5	0x01	Get version command
Byte 6	0x02	Get ID command
Byte 7	0x11	Read memory command
Byte 8	0x21	Go command
Byte 9	0x31	Write memory command
Byte 10	0x43	Erase memory command
Byte 11	0x82	Readout protect command
Byte 12	0x92	Readout unprotect command
Byte 13(1)	0x79	*ACK*
	0x1F	*NACK* (the command is abandoned)

1. *This is the last byte*

4.2 Get version command

The *Get version command* is used to get the bootloader version. When the bootloader receives the command, it transmits the information shown the figures below to the host.

Figure 4. **Get version command: host side**

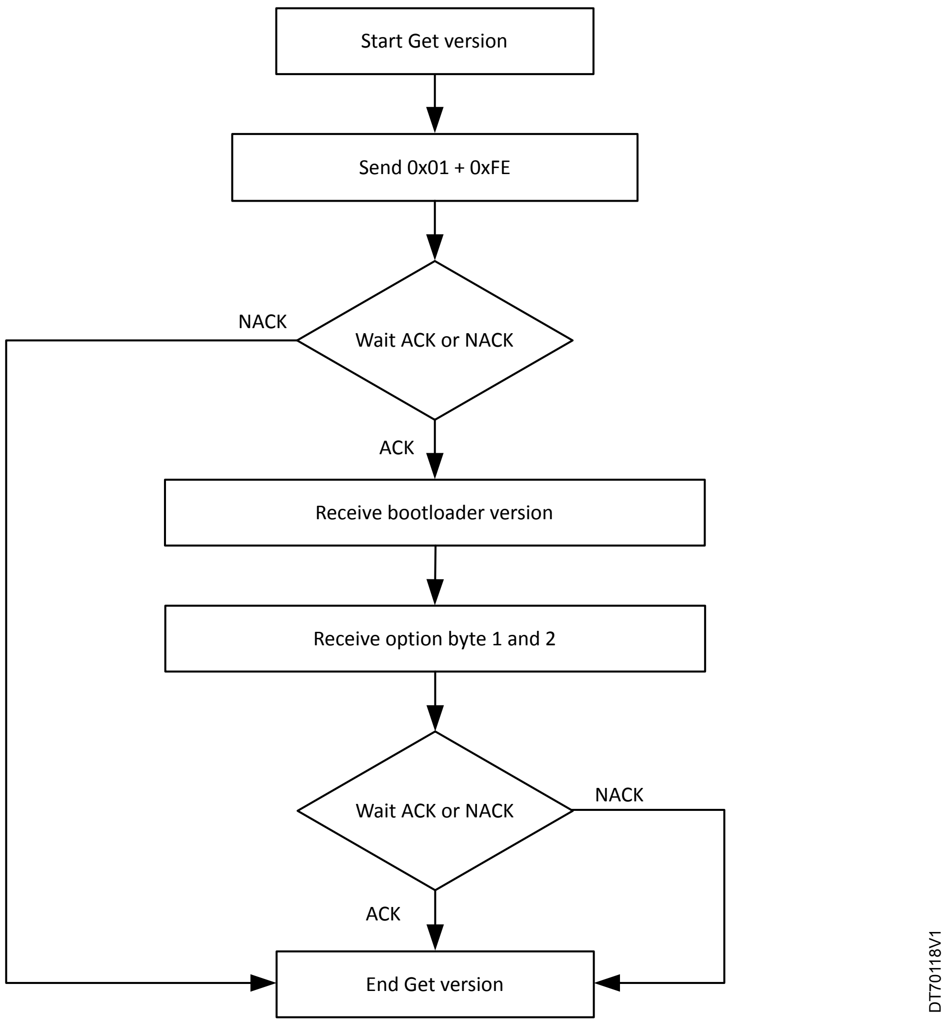
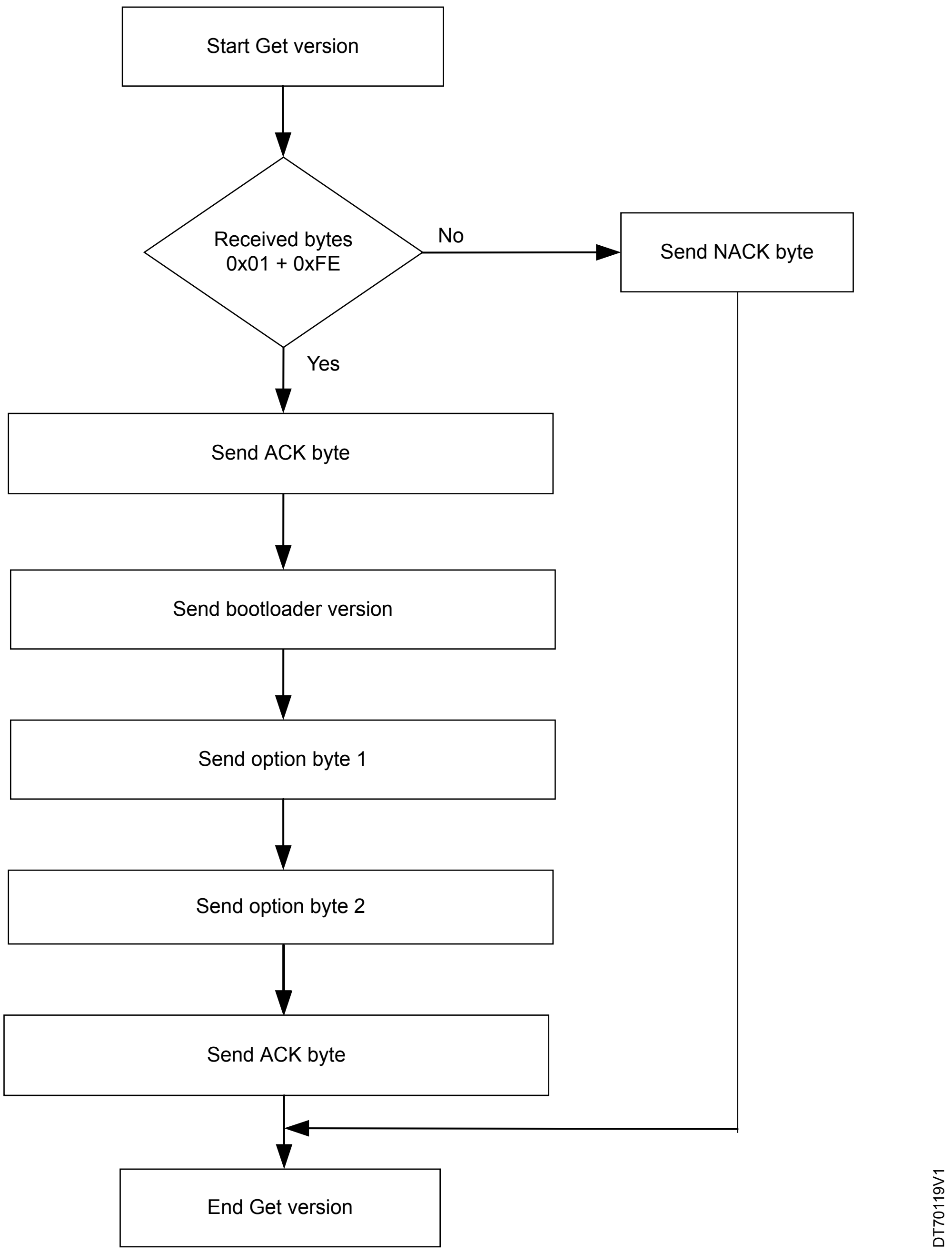


Figure 5. **Get version command: device side**



The STM32WL3 device sends the bytes as described in the following table.

Table 4. **Get version command description**

Byte number	Value	Description
Byte 1	0x79	*ACK*
	0x1F	*NACK* (the command is abandoned)
Byte 2	0 < Version <= 255	Bootloader version For example: 0x01 = version 1.0
Byte 3	0x00	Option byte 1
Byte 4	0x00	Option byte 2
Byte 5	0x79	*ACK*
	0x1F	*NACK* (the command is abandoned)

4.3 Get ID command

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

Figure 6. **Get ID command: host side**

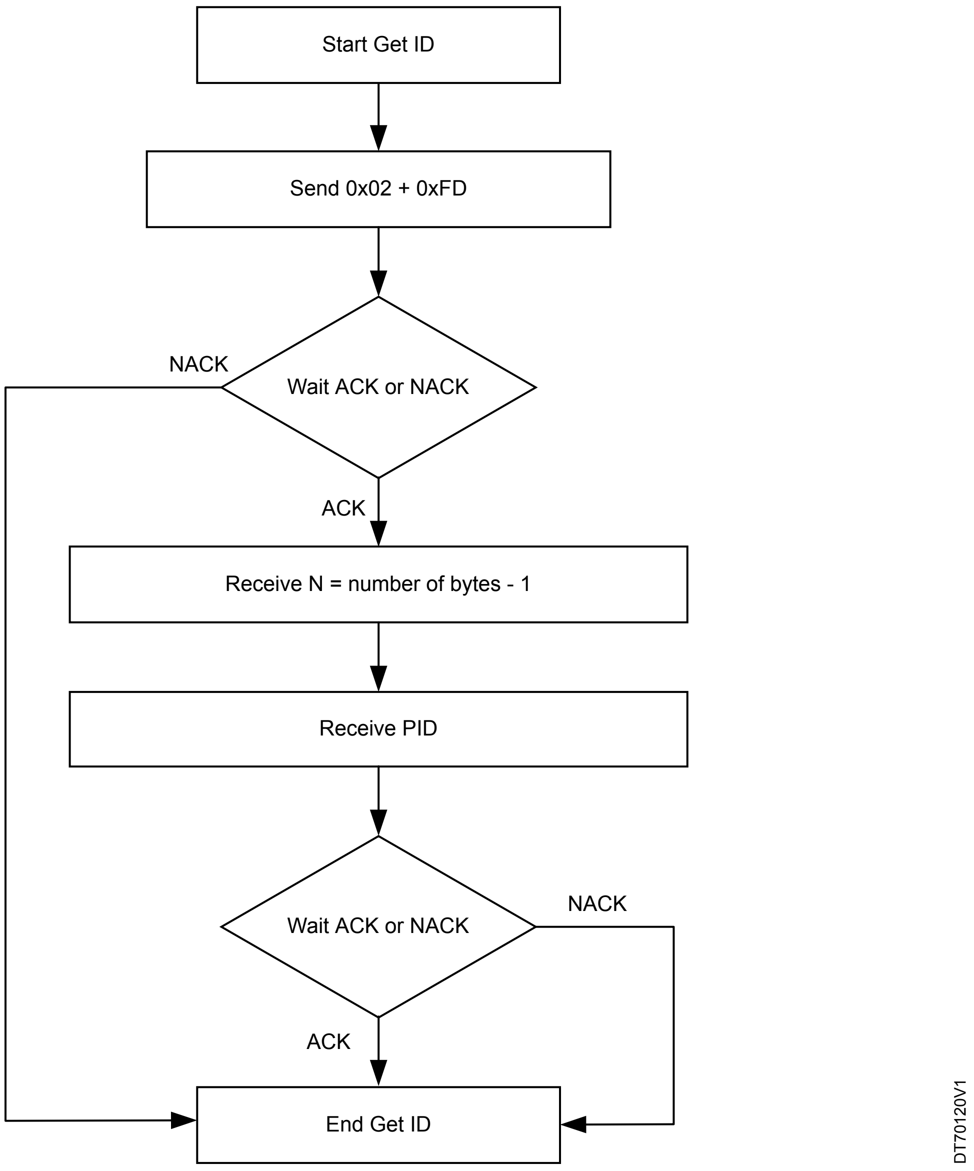
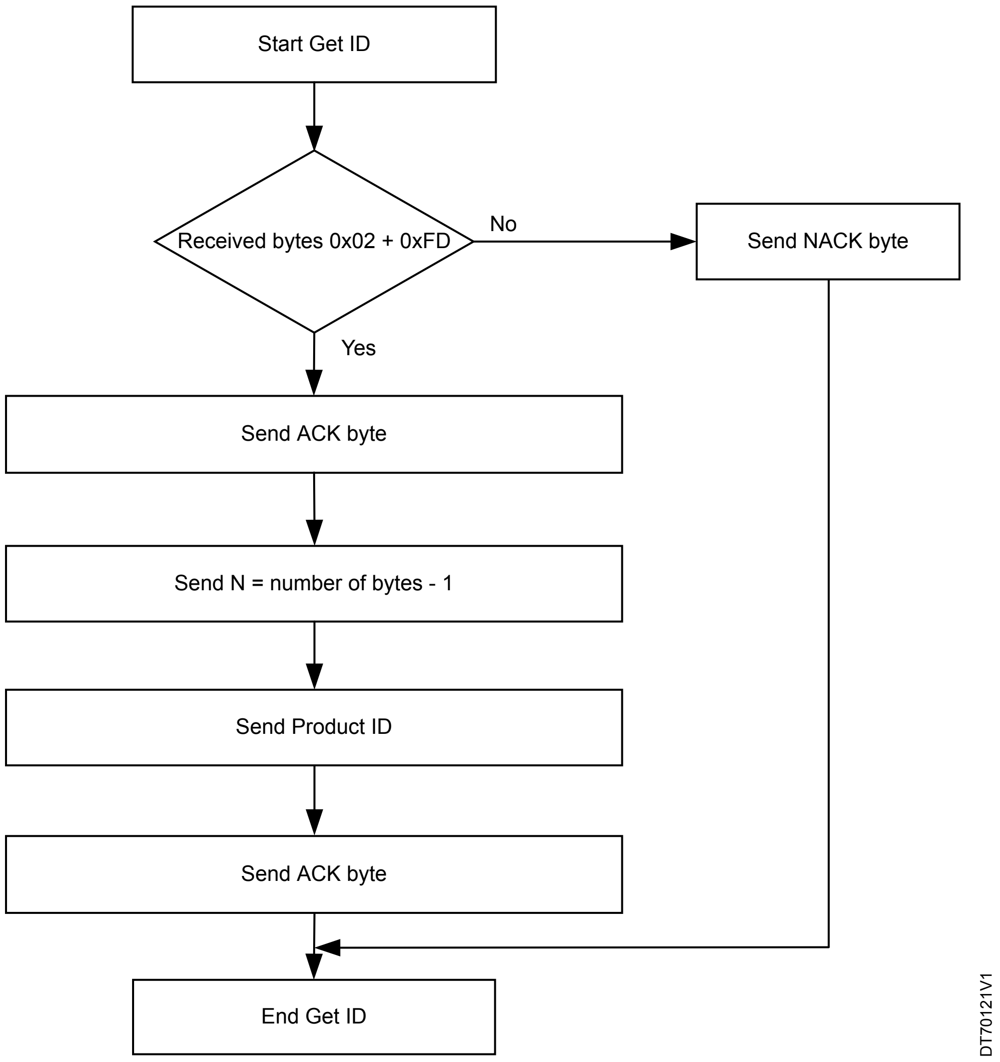


Figure 7. **Get ID command: device side**



The STM32WL3 device sends the bytes as described in the following table.

Table 5. **Get ID command description**

Byte number	Value	Description
Byte 1	0x79	*ACK*
	0x1F	*NACK* (the command is abandoned)
Byte 2	0x02	The number of bytes -1, except for current byte and *ACK*s
Byte 3	0x00	Device metal fix
Byte 4	0x02	Device mask set
Byte 5	0x5F	STM32WL3 with 256 Kbytes of FLASH
Byte 6	0x79	*ACK*
	0x1F	*NACK* (the command is abandoned)

Note: *The Get ID command format returns three bytes:*

- *BYTE1: Metal fix version (for example cut 1.0 is 0)*
- *BYTE2: Mask set version (for example cut 1.0 is 1)*
- *BYTE3: Product specification. This byte is split into two nibbles: 0xHL:*
  - *H is the product ID where 5 means STM32WL3*
  - *L is the flash size code where F means 256 kB for STM32WL3.*

4.4 Read memory command

The *Read memory command* is used to read data from any valid memory address in *RAM* or flash memory. When the bootloader receives the read memory command, it transmits the *ACK* byte to the application. After which, the bootloader waits for:

- An address composed of four bytes (where byte 1 is the address *MSB* and byte 4 is the *LSB*)
- A *checksum* byte.

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 collects the number of waits for bytes to be transmitted less one byte which is expressed as N minus 1 and for its complementary byte (*checksum*).

If the *checksum* is correct, it transmits the needed data, which corresponds to N plus 1 bytes, to the application, starting from the received address.

If the *checksum* is not correct, it sends a *NACK* before aborting the command.

If the readout protection is active, a *NACK* byte is sent to the host when the Read memory command is received.

The process is illustrated in the figures below.

Figure 8. Read memory command: host side

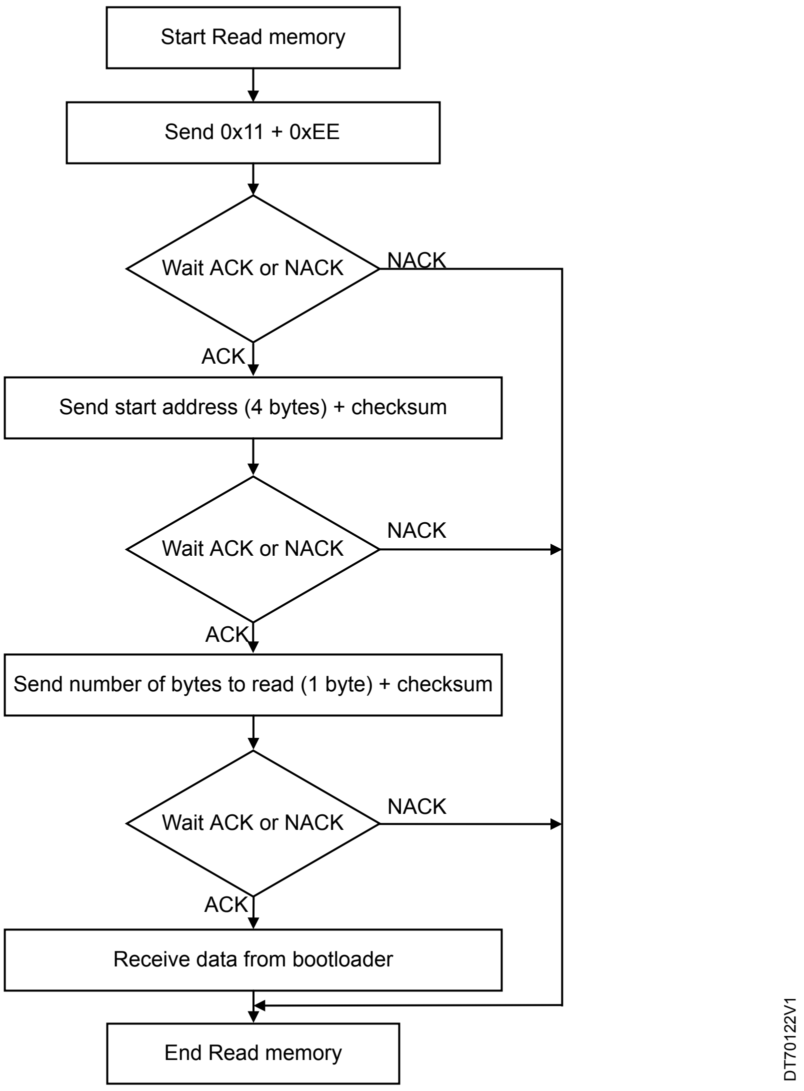
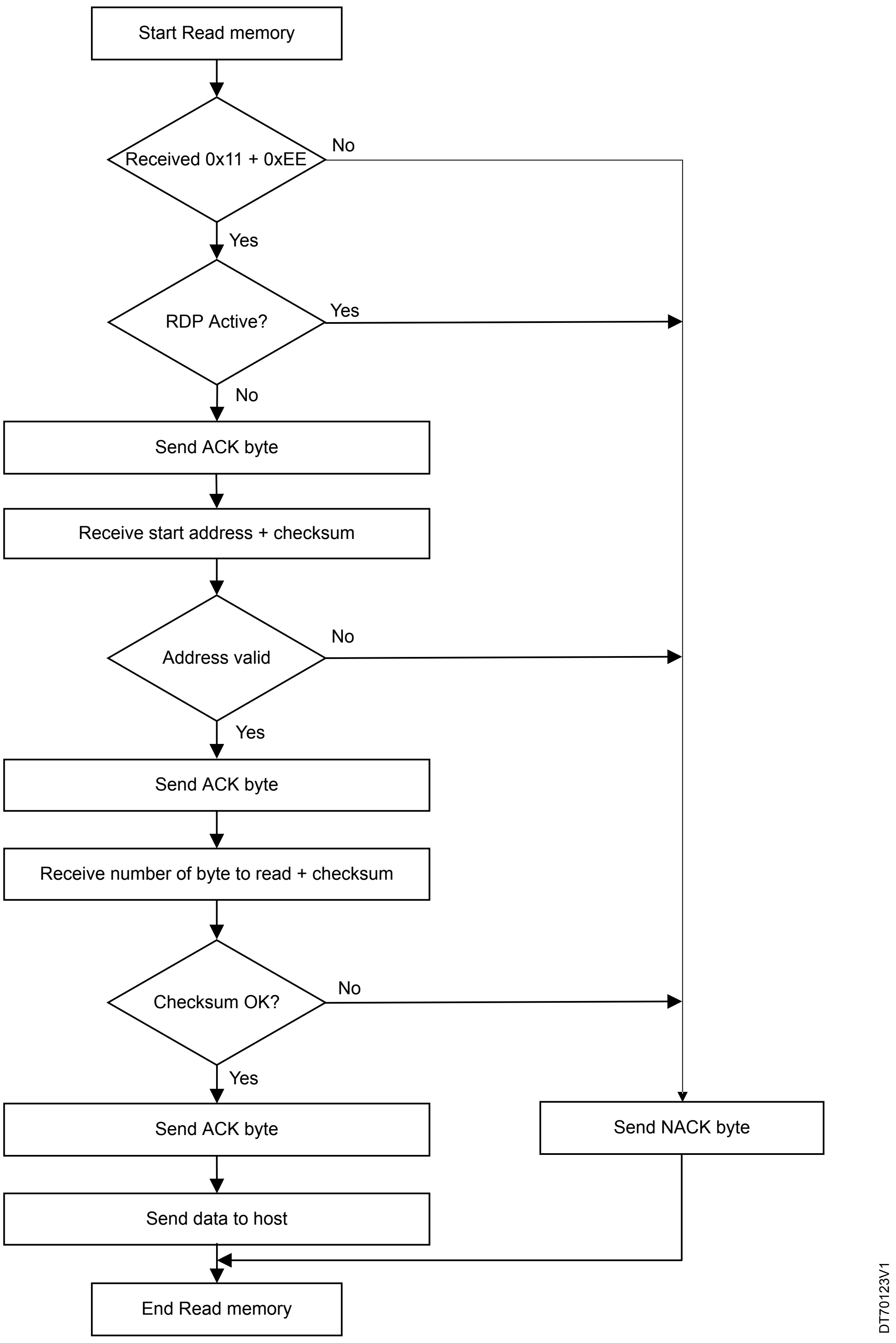


Figure 9. Read memory command: device side



The host sends bytes to the STM32WL3 as described in the following table.

Table 6. **Read memory command description**

Byte number	Value	Description
Byte 1	0x11	-
Byte 2	0xEE	Wait for *ACK*
Byte 3 - 6	-	Start address: • Byte 3: *MSB* • Byte 6: *LSB*
	-	
Byte 7	-	Checksum: *XOR* of address bytes Wait for *ACK*
Byte 8	1	Number of bytes to read (0 < N <= 255)
Byte 9	-	Checksum: *XOR* byte 8 (complement of byte 8)

4.5 Go command

The *Go command* is used to execute the downloaded code or any other code by jumping to an address specified by the application. When the bootloader receives the *Go command*, it transmits the *ACK* byte to the application. After which, the bootloader waits for an address composed of 4 bytes where byte 1 is the *MSB* address and byte 4 is *LSB*, and the last byte being the *checksum* byte.

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 actions:

- Initializes the registers of the peripherals used by the bootloader to their default reset values
- Initializes the user application main stack pointer
- Jumps to the memory location programmed in the received ‘address plus 4’. This corresponds to the address of the application reset handler. For example, if the received address is 0x10040000, the bootloader jumps to the memory location programmed at address 0x10040004.

The process is illustrated in the figures below.

Figure 10. Go command: host side

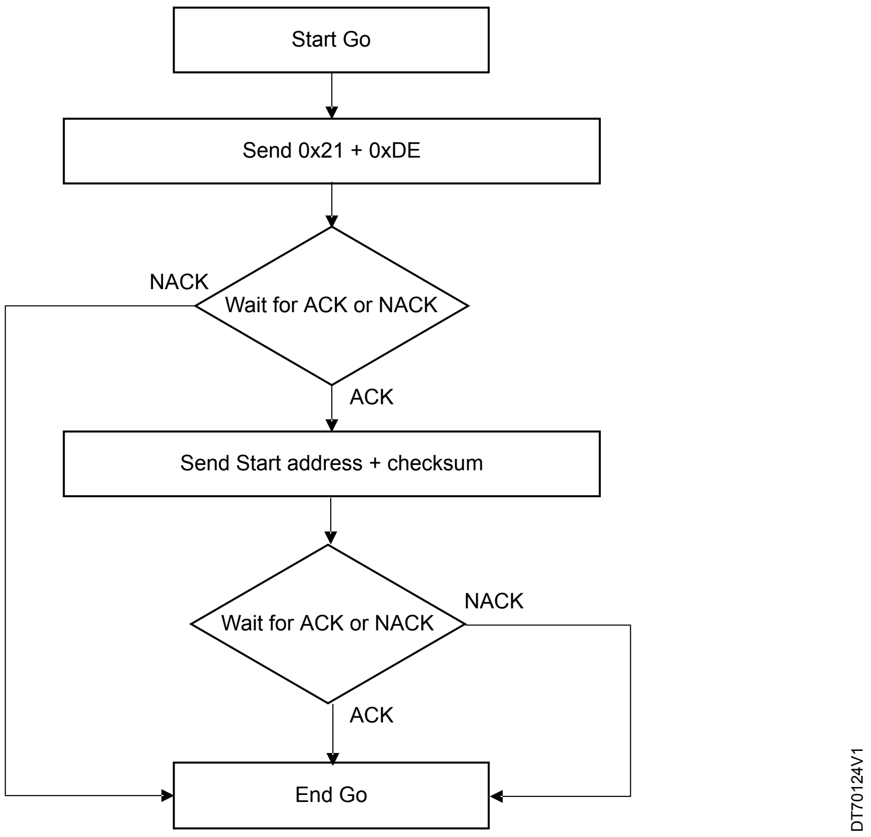
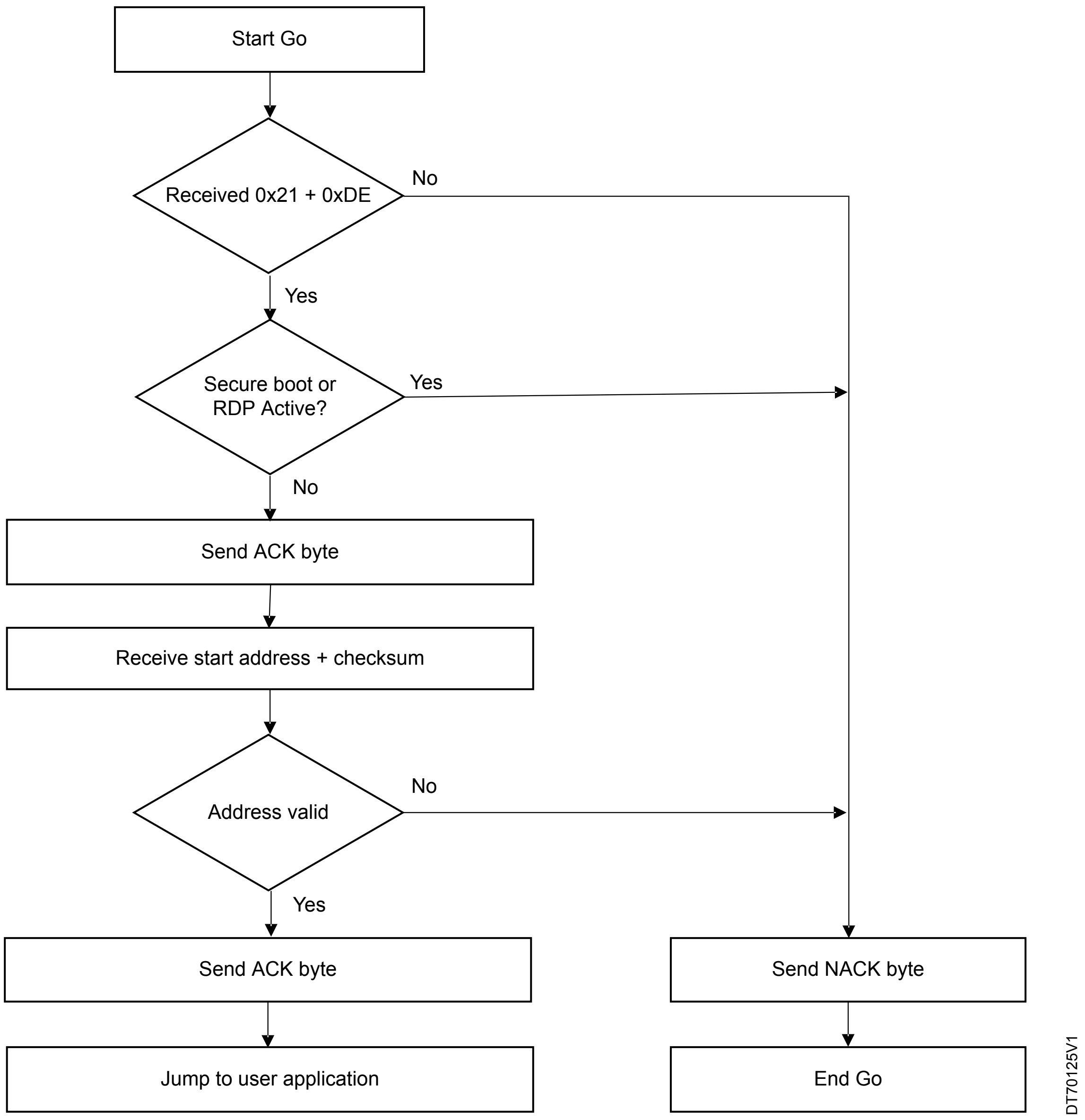


Figure 11. **Go command: device side**



The host sends bytes to the STM32WL3 as described in the following table.

Table 7. **Go command description**

Byte number	Value	Description
Byte 1	0x21	-
Byte 2	0xDE	Wait for *ACK*
Byte 3 – 6	0xXX	Start application address • Byte 3: *MSB* • Byte 6: *LSB*
Byte 7	0xXX	Checksum: *XOR* of address bytes
Wait for *ACK*		

4.6 Write memory command

The *Write memory command* is used to write data to any valid memory address in *RAM* or flash memory.

When the bootloader receives the *Write memory command*, it transmits the *ACK* byte to the application. After which, the bootloader waits for an address composed of four bytes, where byte 1 is the address *MSB* and byte 4 is the *LSB*, and a *checksum* byte.

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 performs these actions:

- Gets a byte, N, which contains the number of data bytes to be received
- Receives the user data which is N plus 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 is successful, the bootloader transmits the *ACK* byte; otherwise, it transmits a *NACK* byte to the application and aborts the command.

The maximum length of the block to be written for the STM32WL3 is 256 bytes.

If the readout protection is active, a *NACK* byte is sent to the host when the Write memory command is received.

The process is illustrated in the figures below.

Figure 12. **Write memory command: host side**

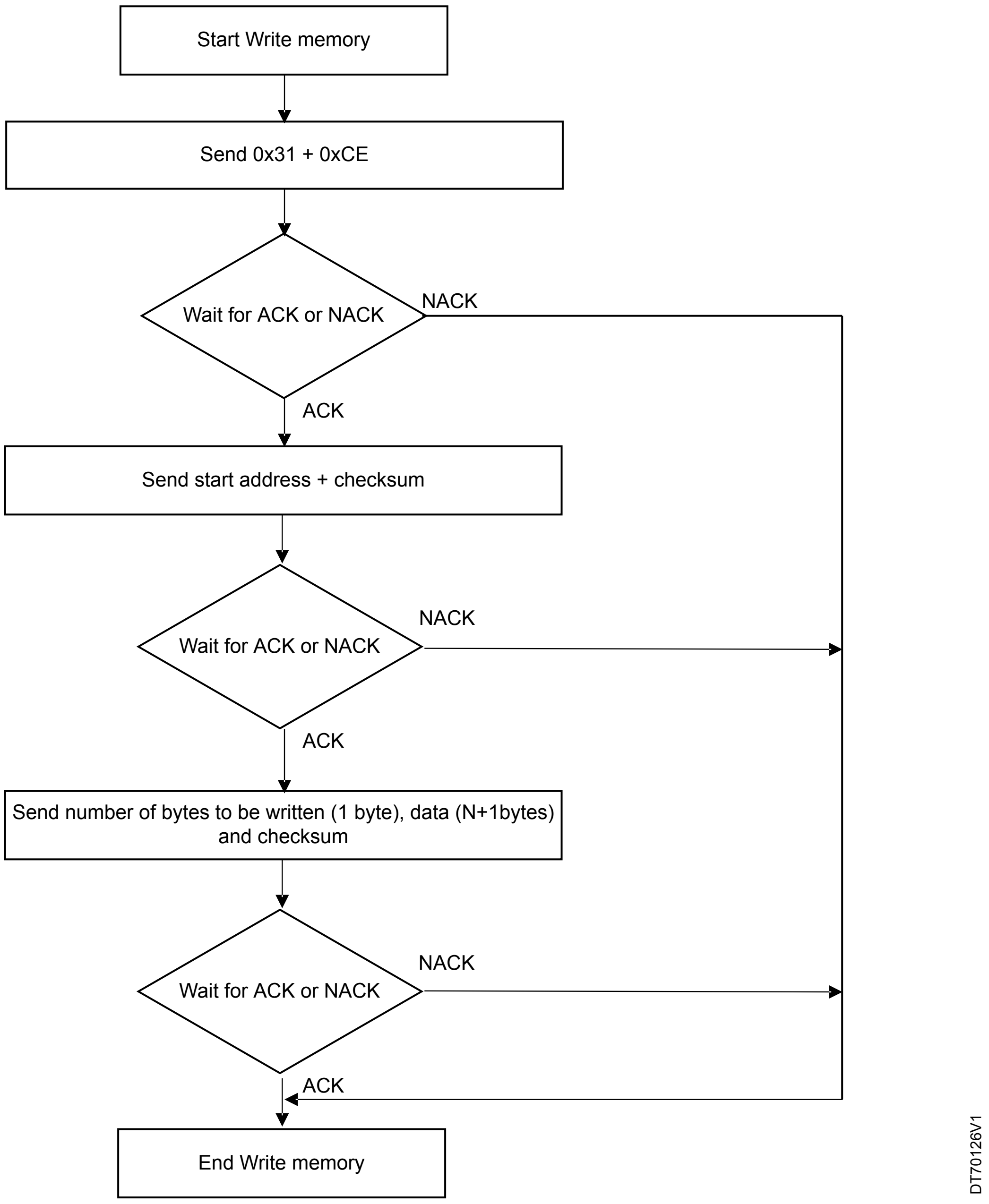
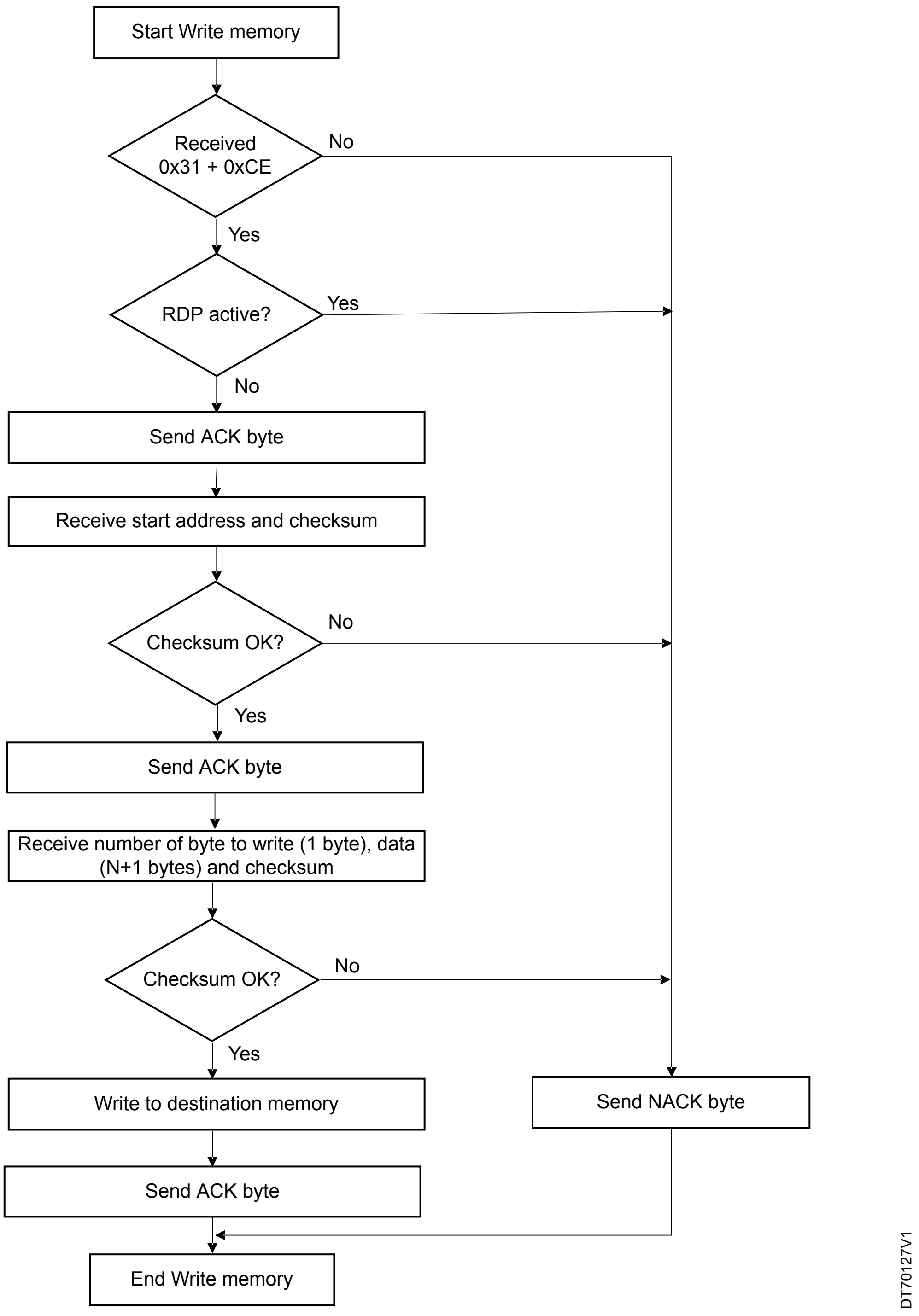


Figure 13. **Write memory command: device side**



The host sends the bytes to the STM32WL3 as described in the following table.

Table 8. **Write memory command description**

Byte number	Value	Description
Byte 1	0x31	-
Byte 2	0xCE	Wait for *ACK*
Byte 3 - byte 6	0xXX	Start address: • Byte 3: *MSB* • Byte 6: *LSB*
Byte 7	0xXX	Checksum: *XOR* (byte 3, byte 4, byte 5, byte 6) Wait for *ACK*
Byte 8	0xXX	Number of bytes to be received (0< N <= 255) N + 1 data bytes (Max. 256 bytes)

Note: : Checksum byte: XOR (N, N+1 data bytes) Wait for ACK

4.7 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 which, the bootloader receives the following instruction:

- One byte corresponding to the number of pages to be erased
- The flash memory page codes of the pages to be deleted. The amount of information transmitted depends on the number of pages to be deleted. This ranges from 0 to 127.
- A *checksum* byte.

If the *checksum* is correct, the bootloader then 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.

The *Erase memory command* specifications are:

1. the bootloader receives one byte that contains N, the number of pages to be erased minus 1. N= 255 is reserved for mass erase request. For 0 < N <= 127, N plus 1 pages are erased.
2. the bootloader receives N plus 1 bytes, each byte containing a page number

The process is illustrated in the figures below.

Figure 14. **Erase memory command: host side**

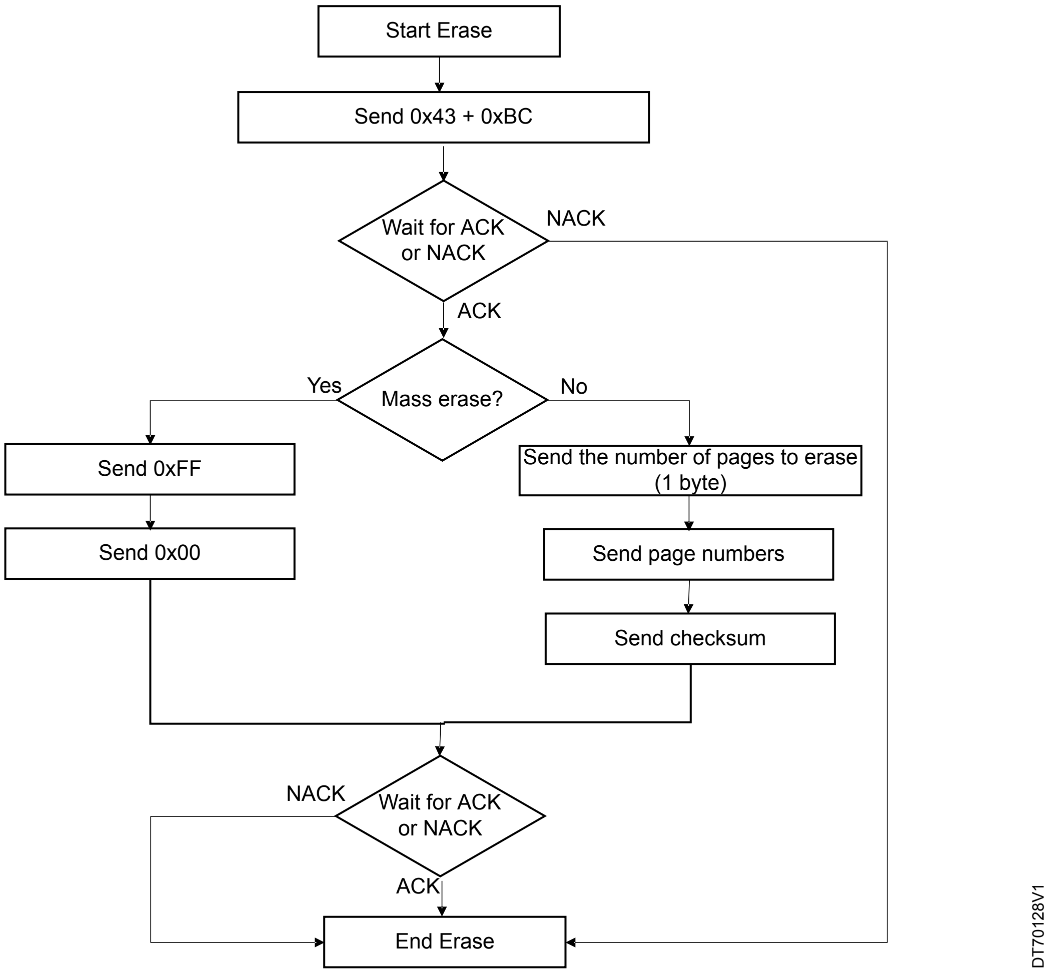
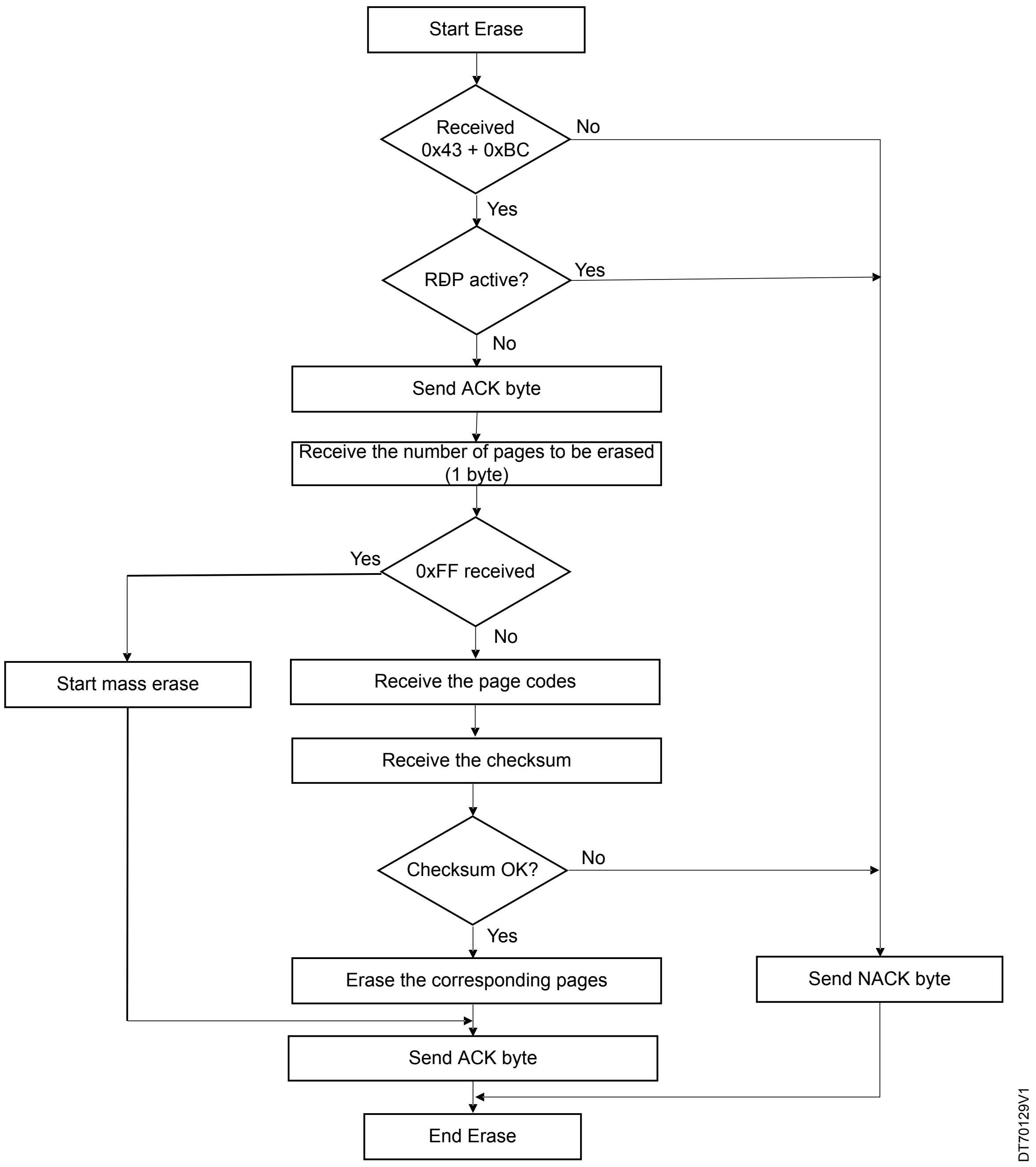


Figure 15. **Erase memory command: device side**



The host sends bytes to the STM32WL3 as described in the following table.

Table 9. **Erase memory command description**

Byte number	Value	Description
Byte 1	0x43	-
Byte 2	0xBC	Wait for *ACK*
Byte 3	0xFF or number of pages to be erased-1	-
Byte 4	0xXX	0x00 (in case of mass erase) or ((N+1 bytes (page numbers) and then checksum *XOR* (N, N+1 bytes)) Wait for *ACK*

4.8 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 which, the bootloader enables the read protection for the flash memory.

At the end of the *Readout protect command*, the bootloader transmits the *ACK* byte to signal the end of the command.

The process is illustrated in the figures below.

Figure 16. **Readout protect command: host side**

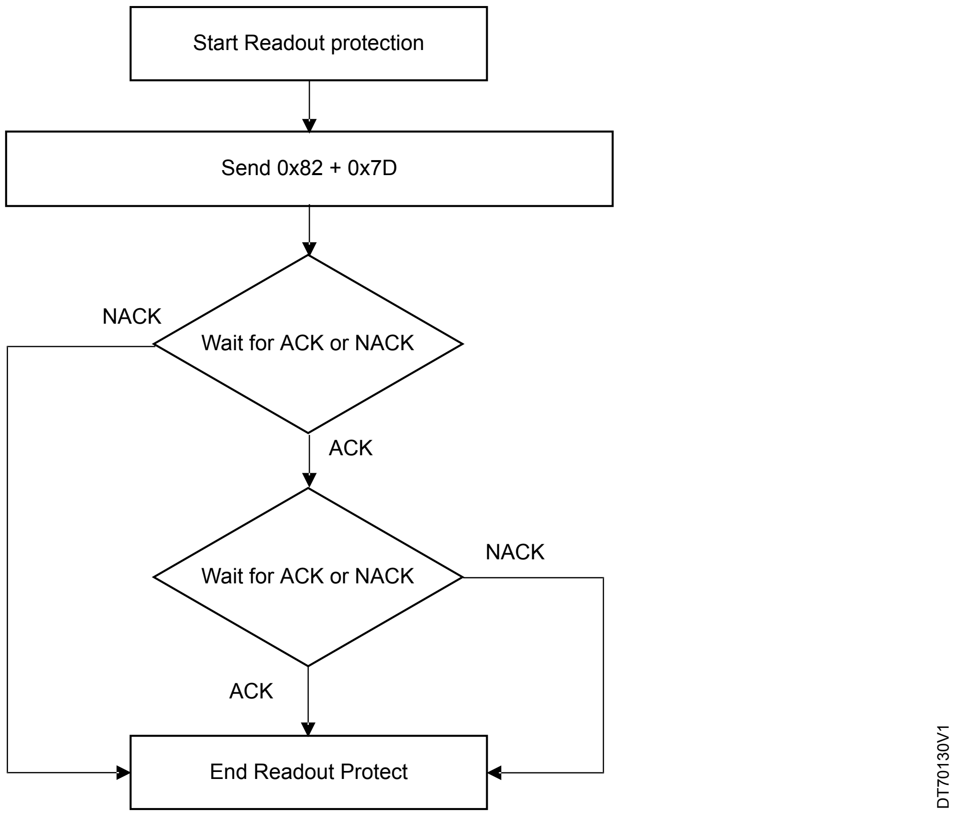
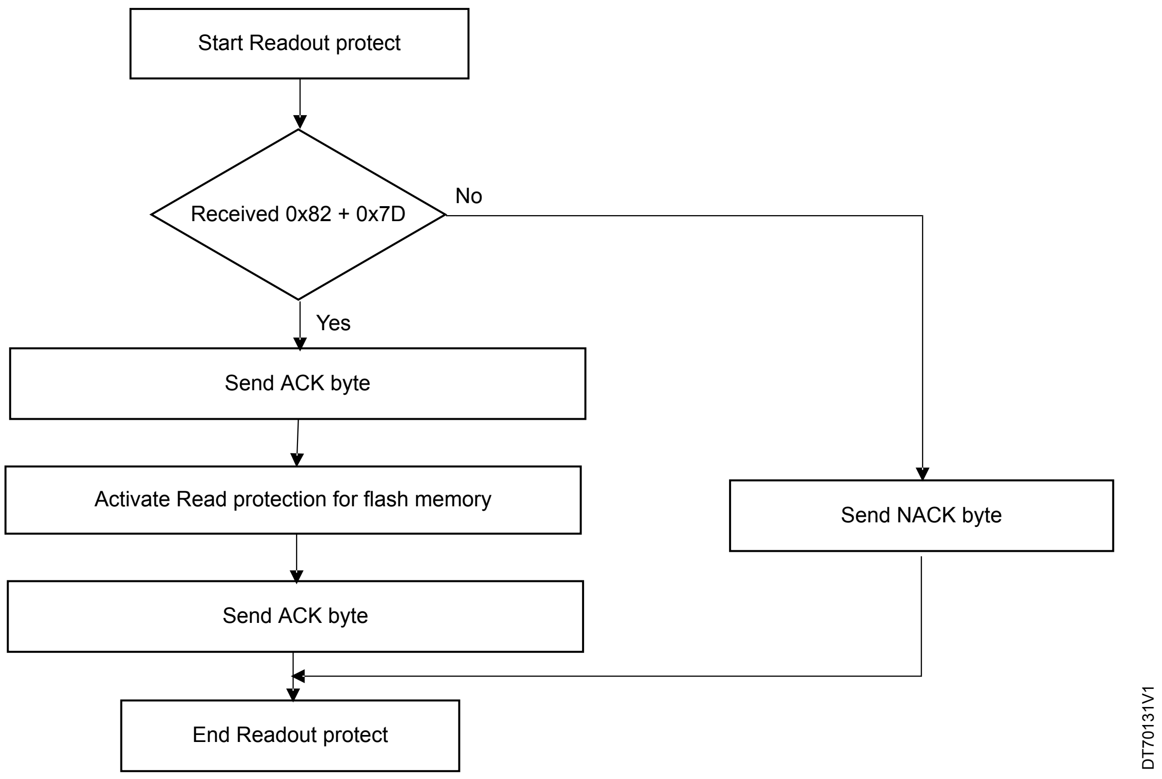


Figure 17. **Readout protect command: device side**



4.9 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 which, 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 readout protection. 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 generates a system reset. If other bootloader commands need to be executed, the bootloader activation needs to be re-executed, a hardware reset is needed and PA10 must be pulled high, as described in the *UART* bootloader activation.

The process is illustrated in the figures below.

Figure 18. **Readout unprotect command: host side**

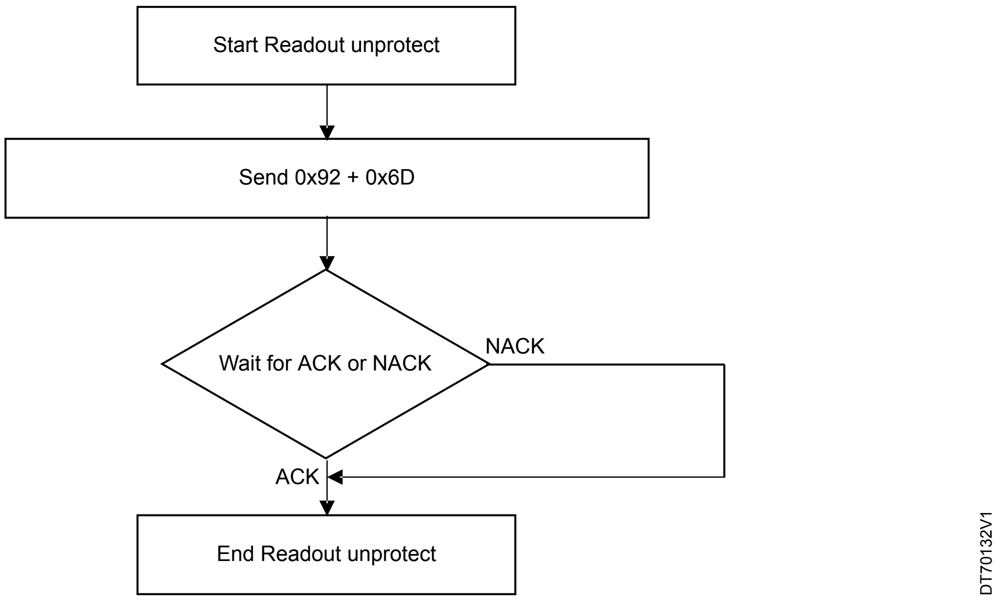
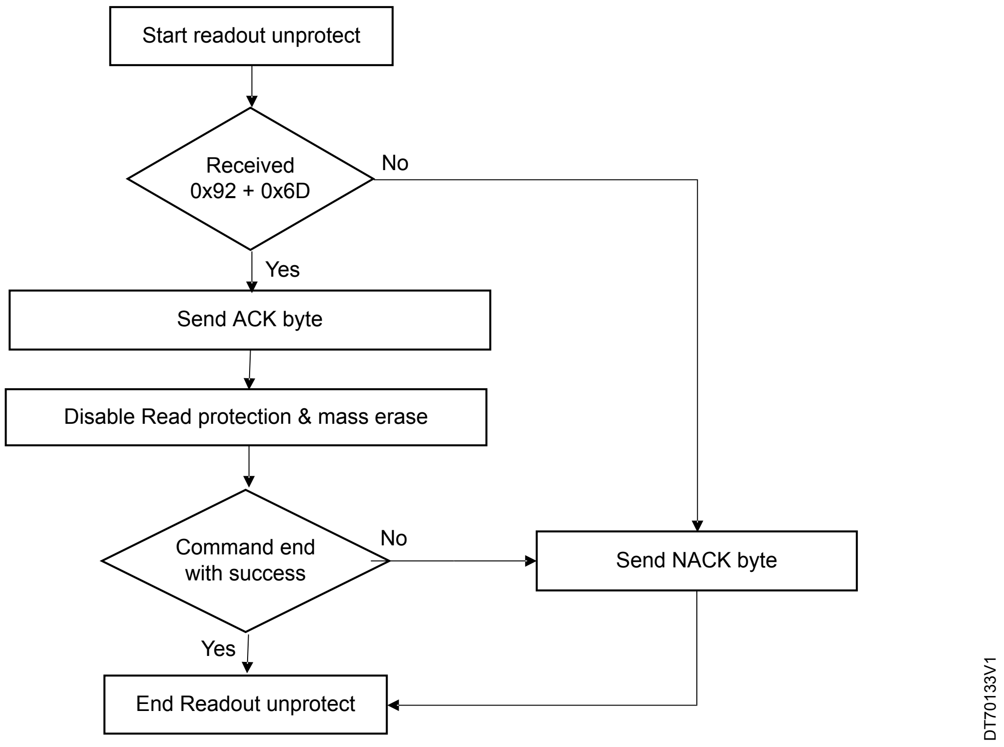


Figure 19. **Readout unprotect command: device side**



4.10 OTP write command

The OTP write command is used to write 32-bit word data to any valid memory address in OTP memory.

When the bootloader receives the OTP write command, it transmits the ACK byte to the application. After which, the bootloader waits for an address (4 bytes, byte 1 is the address MSB and byte 4 is the LSB) and a checksum byte.

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 performs these actions:

- Receives the user data (4 bytes) and the checksum (XOR of all data bytes)
- Programs the user data to memory in the received address

At the end of the command, if the OTP write operation is successful, the bootloader transmits the ACK byte. Otherwise, it transmits a NACK byte to the application and aborts the command.

Figure 20. OTP write command: host side

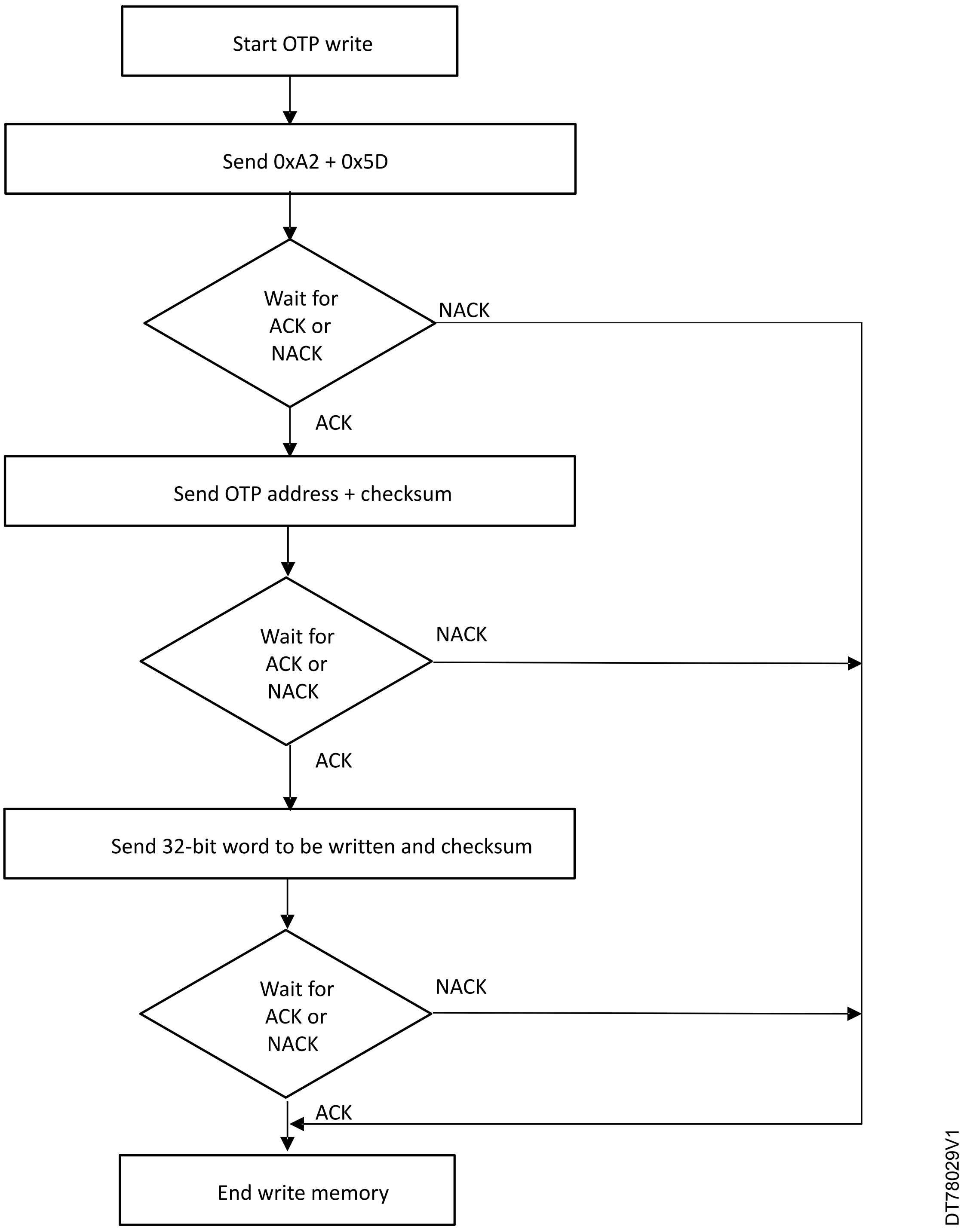
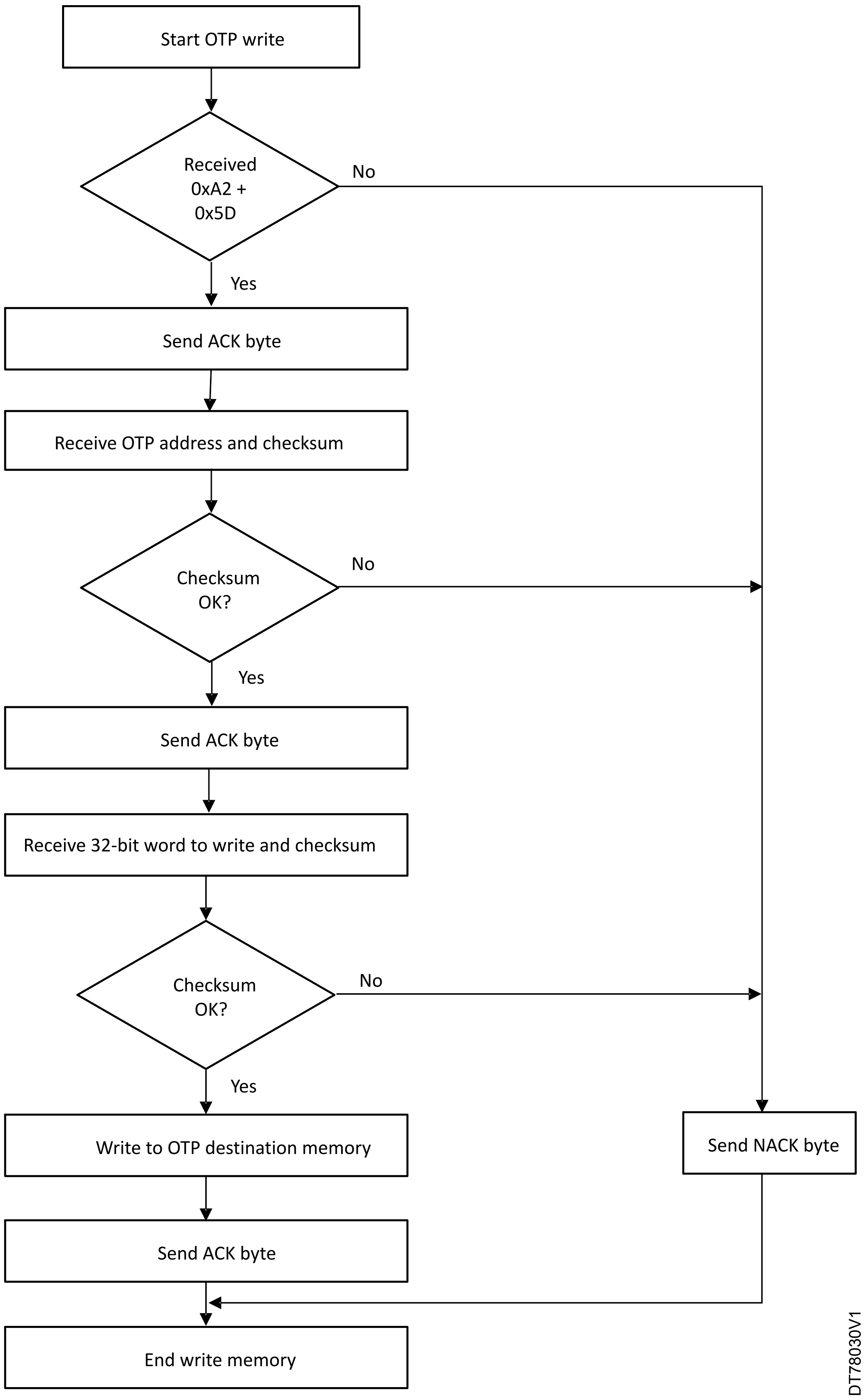


Figure 21. OTP write command: device side



The host sends the bytes to the device in the following sequence:

1. Byte 1: `0xA2`
2. Byte 2: `0x5D`
   – Wait for ACK.
3. Byte 3 to byte 6: OTP address
   – Byte 3: MSB
   – Byte 6: LSB
4. Byte 7: Checksum calculated as `XOR (byte 3, byte 4, byte 5, byte 6)`
   – Wait for ACK.
5. Byte 8 to byte 11: Data word to write
6. Checksum byte: Calculated as `XOR` of the data word bytes
   – Wait for ACK.

5 Secure bootloader

The secure bootloader feature is part of the *UART* bootloader preprogramed on the STM32WL3 at manufacturing time: it allows a user application to be authenticated in order to determine whether it is an authorized firmware or not. Only authorized, trusted applications are allowed to run.

The STM32WL3 secure bootloader uses asymmetric cryptographic algorithms with key pair (public, private). RSA-2048 is used with 256 bytes public key size.

This authentication method allows an application firmware to be authenticated by generating and appending a digital signature to it.

The secure bootloader feature is activated by writing a specific value in the *OTP* area.

5.1 How it works

The following steps describe the process to be followed to correctly authenticate an application fimware at user production time:

The user generates the key pair (public and private) and programs its device using the signed firmware:

- The application firmware is signed by the owner using the private key, and the generated sign is appended to the application firmware (digital sign size is 2048 bits).
- The STM32WL3 *OTP* section is used to store the generated public key.

The secure bootloader uses the following components to verify and authenticate the application image:

- The public key stored in the device *OTP*.
- The application firmware image in the flash memory (with information related to the size)
- The digital signature appended at the end of the application firmware image
- The *OTP* dedicated location for enabling secure bootloader.

Only the application firmware image signed with the correct private key can be executed and the private key is never shared or stored inside the STM32WL3.

The key point of this method is that the application firmware signing is done using the private key (not stored on the device) and the authentication and verification is done with associated public key. So, if the application firmware has not been signed with the correct private key (owned by user), the STM32WL3 is not able to execute it.

5.1.1 How to store the public key in the OTP memory

The *OTP* memory is used to store the public key and other basic information needed to activate the secure bootloader feature.

To enable the secure bootloader feature, the *OTP* memory must contain the following information at specified addresses:

Table 10. **OTP memory location addresses**

Address	Description	Comments
0x1000 1800	Activation word	The activation word is: 0xEC1C C10B. This word only activates the secure boot feature in the bootloader code.
0x1000 1804	The start flash location of the signed application firmware	This is the interrupt vector table start address of the signed application firmware which is 0x1004 0000(1).
0x1000 1808	R2 public key	R2 public key with length of 256 bytes.
0x1000 1908	Modulus public key	Modulus public key with length of 256 bytes.
0x1000 1A08	Exponent public key	Exponents public key with length of 4 bytes.

1. *To ensure correct secure verification, the application must start from the flash memory bottom at address `0x1004 0000`. The same address must be stored at the OTP address `0x1000 1804`. This ensures that only genuine signed firmware can execute from the flash memory.*

The total size of the information to store inside the *OTP* area is 524 bytes.

When all the data is stored inside the *OTP* area, it is mandatory to enable the *OTP* lock memory mechanism by writing a word different from 0xFFFF FFFF at *OTP* address 0x1000 1BFC.

5.1.2 How to sign an application firmware

This section provides a basic example of the following steps:

1. Generate public/private keys
2. Generate a digital sign from an input file
3. Append the digital sign to the input file
4. Store the public key in the STM32WL3 OTP
5. Enable the secure bootloader on STM32WL3 device

To enable the secure bootloader feature, the following secure bootloader utilities are available within the STM32Cube Programmer software package:

Table 11. Secure bootloader description

Bootloader utility	Description
`STM32TrustedPackageCreator_CLI.exe -keygen`	This utility generates the public and private keys for the RSA-2048 algorithm, and requires the following parameters: • -k <Destination folder for private key generated> <Destination folder for public key generated> • -f <Destination folder to save the `public_key.c` and `public_key.py` files> The option "k" creates folders where the public and private keys are saved. The option "f" creates a folder that contains the public key saved in two formats. The first format supports C projects, and the second format supports Python scripts. Utility example: `STM32TrustedPackageCreator_CLI.exe -keygen -k <Output_File_priv_path> <Output_File_pub_path> -f <public_key.c path> <public_key.py path>`
`STM32TrustedPackageCreator_CLI.exe -signbin`	This utility takes an unsigned raw binary file in input, and executes the following operations in this order: • Addition of the firmware size inside the binary file at offset 0x18. • Generation of the signature for the application firmware. To create the signature, the utility uses the private key already generated by the STM32TrustedPackageCreator_CLI.exe -keygen utility. • Addition of the signature at the end of the binary file. A new signed file is created. This utility requires the following parameters: • -k <Folder with private key used to create the signature. This is the same folder created with the "k" option of the STM32TrustedPackageCreator_CLI.exe -keygen utility)> • -f <Input raw binary image file not signed> • -o <Output raw binary image file signed> The file generated with the "o" option is the signed application that can be stored in the main flash memory. Utility example: `STM32TrustedPackageCreator_CLI.exe -signbin -k <Input_File_priv_path> -f <binary_to_sign_path> -o <output_signed_bin_path>`

Bootloader utility	Description
`STM32_programmer_CLI.exe -storekeyotp`	This utility requires the device to be in bootloader mode. It requires the following parameters: • -c port=<PortName> • <key_path>: folder with the `public_key.py` file to store in OTP • <start_address> : start firmware address, which is the interrupt vector table start address • <lock>: used to lock the OTP Utility examples: • Store OTP key with lock: `STM32_programmer_CLI.exe -c port=COMx -storekeyotp "c:\key\key.pem" 0x10040000` • Store OTP key without lock: `STM32_programmer_CLI.exe -c port=COMx -storekeyotp "c:\key\key.pem" 0x10040000`

Revision history

Table 12. **Document revision history**

Date	Version	Changes
26-Mar-2024	1	Initial public release
29-Oct-2024	2	Updated: • Section 5.1: How it works • Table 10. OTP memory location addresses • Section 5.1.2: How to sign an application firmware
24-Jul-2025	3	Updated Section 4: UART bootloader commands. Added Section 4.10: OTP write command.

Glossary

ACK transaction acknowledged

checksum A byte containing the computed XOR of all previous bytes is added to the end of each communication called a checksum byte. By XORing all received bytes, data plus checksum, the result at the end of the packet must be 0x00.

ID identification

LSB least significant bit

MSB most signficant bit

NACK transaction not acknowledged

OTP one time programming

RAM random access memory

UART universal asynchronous receiver transmitter

UART Rx universal asynchronous receiver transmitte read

UART Tx universal asynchronous receiver transmitte transmit

XOR exclusive OR function

Contents

List of figures

List of tables