XMC7000 bootloader

Based on controller area network

About this document

Scope and purpose

This application note describes a controller area network (CAN)-based bootloader for XMC7000. This application note also explains how to communicate with a CAN-based bootloader.

Intended audience

Anyone using XMC7000 bootloader.

Associated part family

XMC7000 family XMC7100/XMC7200 series of **XMC™ industrial microcontrollers**.

Table of contents

XMC7000 bootloader

1 Introduction to bootloading

Bootloaders are commonly present in an MCU system design and make it possible for a product's firmware to be updated in the field. At the factory, the firmware is initially programmed into a product typically through the MCU's Joint Test Action Group (JTAG) interface or the Arm® serial wire debug (SWD) interface. However, these interfaces are usually not accessible in the field.

Bootloading is a process that allows you to upgrade your system firmware over an automotive standard communication interface such as CAN. A bootloader communicates with a host to get new application code or data and writes it into the device's flash memory.

In this application note, you will learn how to communicate with a CAN-based bootloader.

This application note assumes that you are familiar with bootloader concepts and the CAN protocol. For more details on the CAN Component, see the "Flash Boot" and "CAN FD Controller" chapters of the **Architecture Technical Reference Manual**.

1.1 Terms and definitions

Figure 1 illustrates the main elements in a bootloader system. It shows that the product's embedded firmware must be able to use the communication port for two different purposes: normal operation and updating the flash. The portion of the embedded firmware that knows how to update the flash is called the "bootloader." The other terms in **Figure 1** are defined in the following paragraphs.

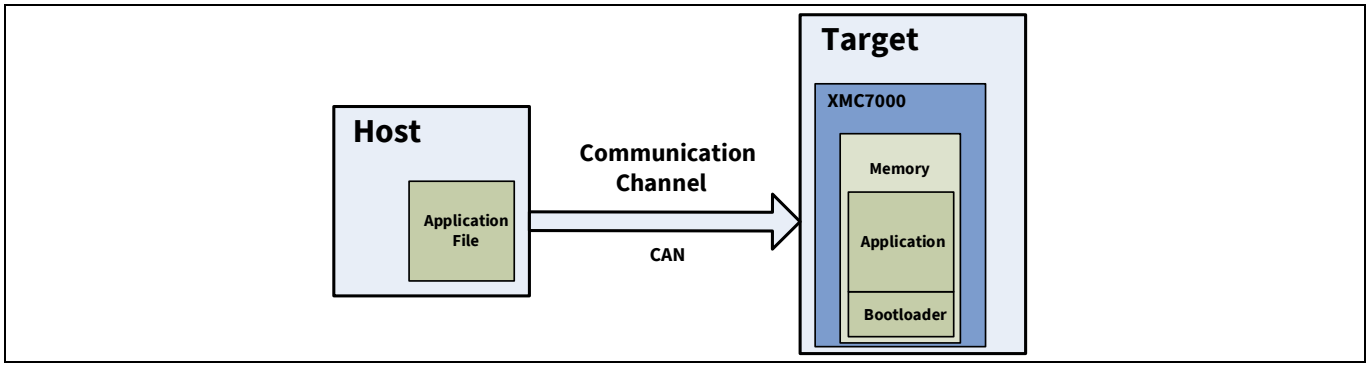

Figure 1 Bootloading system diagram

The system that provides the data to update the flash is called the "host", and the system being updated is called the "target". The host can be an external PC (PC host) or another MCU.

The act of transferring data from the host to the target is called "bootloading", or a "bootload" operation, or a "bootload" for short. The firmware that is placed in the memory is called the "application" or the "bootloadable".

1.2 Using a bootloader

A bootloader communication port is typically shared between the bootloader and the actual application. The first step in using a bootloader is to manipulate the target, so that the bootloader and not the application is executing.

Once the bootloader is running, the host can send a `Enter Bootloader` command over the communication channel. If the bootloader sends an `OK` response, bootloading can begin.

1.3 Bootloader function flow

During bootloading, the host reads the file for the new application, parses it with the commands downloaded to the RAM, and sends those commands to the bootloader. After the entire file is sent, the bootloader can pass control to the new application.

An internal bootloader typically executes in flash boot after the device resets. The bootloader can then perform the following actions:

- Check the new application's validity before transferring control to that application
- Manage the timing to start host communication
- Perform the bootloading operation
- Pass control to the new application

Figure 2 shows the bootloading sequence.

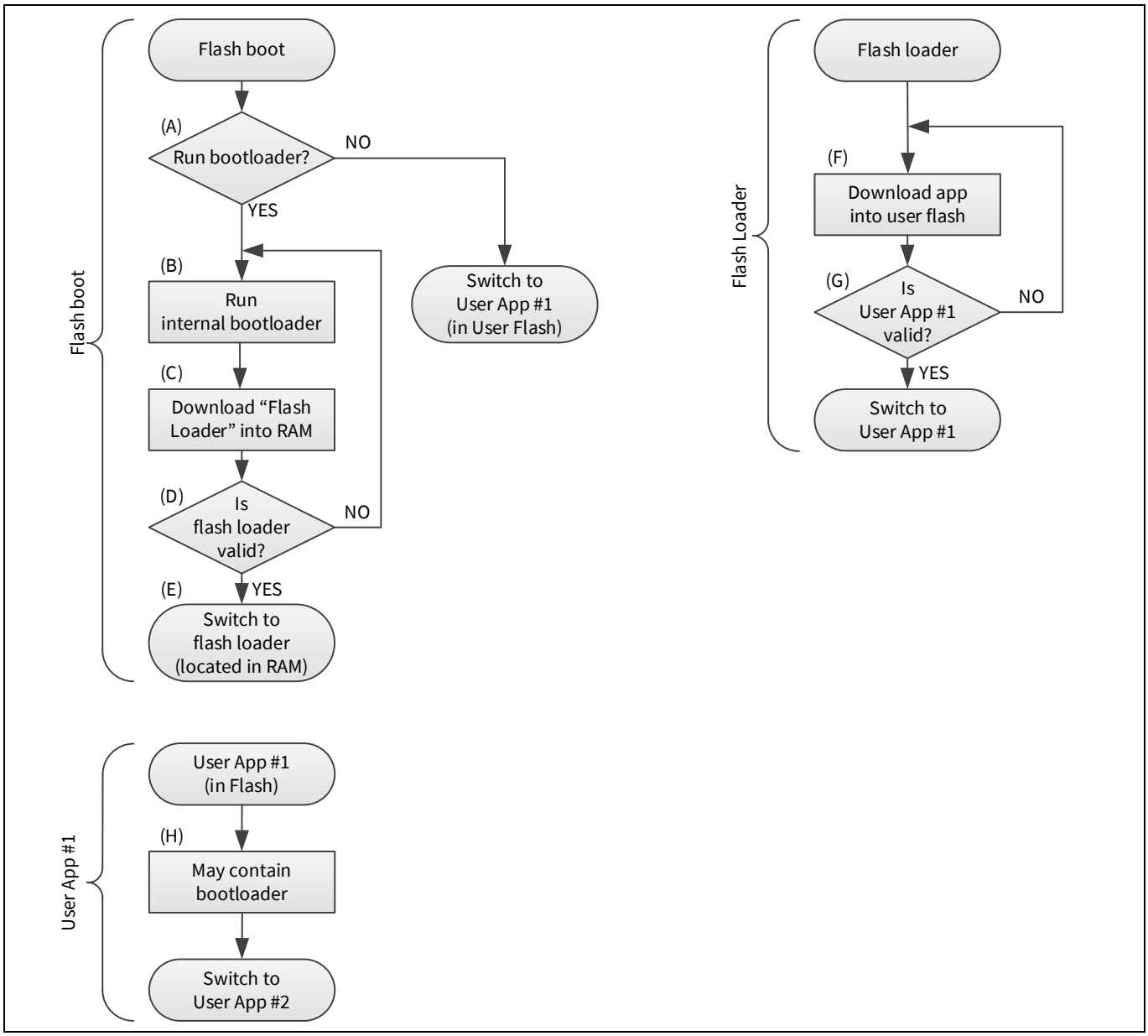

Figure 2 Bootloading sequence

The bootloading sequence is as follows:

(A) The flash boot checks whether the internal bootloader (part of the flash boot) should be run.

(B) The internal bootloader is a part of the flash boot firmware that has a goal to download the flash loader into the RAM (C) and launch it (E).

(D) The flash loader requires neither a secure signature nor an encryption. However, the checksum (CRC-32C) must be placed in the last 4 bytes of the flash loader if the host uses the **Verify Application** command.

(F) The flash loader downloads a user application through CAN communication and stores it into the code flash or work flash.

(G) The flash loader verifies the user application for integrity. If the user application signature verification fails, the flash loader tries to restart bootloading and receive a new image.

(H) The user application may or may not contain a bootloader. It is dependent on the user.

Note that only the flash boot part of the bootloading sequence (A) to (E) is developed as the flash boot firmware; the remaining sequence is developed by the user.

1.4 Device interface configuration

The bootloader enables the end-of-line programming using only CAN when the following conditions are met:

- Two words at the start of the flash must be '0xFFFFFFFF'.
- TOC2 is valid and internal bootloader is enabled (default) by TOC2_FLAGS.FB_BOOTLOADER_CTL bits, or TOC2 is empty
- Protection mode is not SECURE and not SECURE_DEAD.
- No debugger connection happened during the one-second wait window.

First, the bootloader prepares the channel configuration for CAN and waits for the preconfigured time for the frame from the host. In case of a timeout, there is a period of idle time that follows. If no frame from the host is received, this procedure is repeated for 300 seconds, which is the overall bootloading time as shown in **Figure 3**.

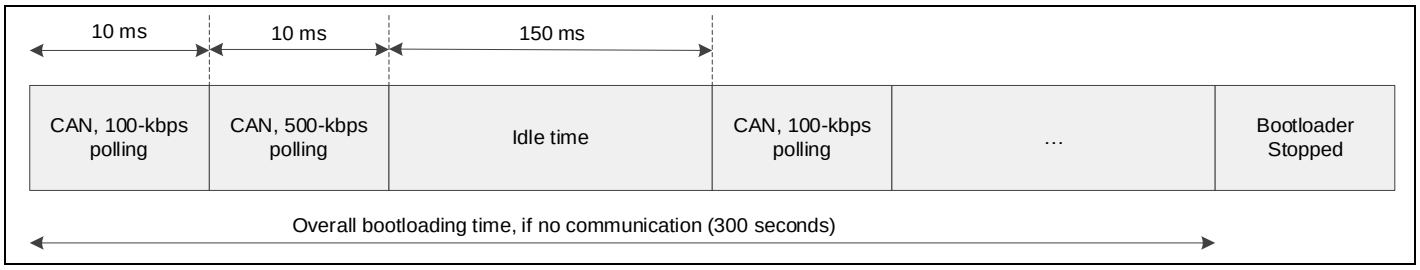

Figure 3 Bootloader polling sequence

If an **Enter Bootloader** command is received on the CAN communication interface, the polling stops and the bootloader starts using the interface. If the bootloading succeeds, the bootloader launches the updated application in the RAM. This application is named a "flash loader".

Figure 4 shows a default startup timing on a new device without a firmware in the flash. Note that once the firmware is written to the flash, the internal bootloader is no longer launched. See the device-specific datasheet for the defined time.

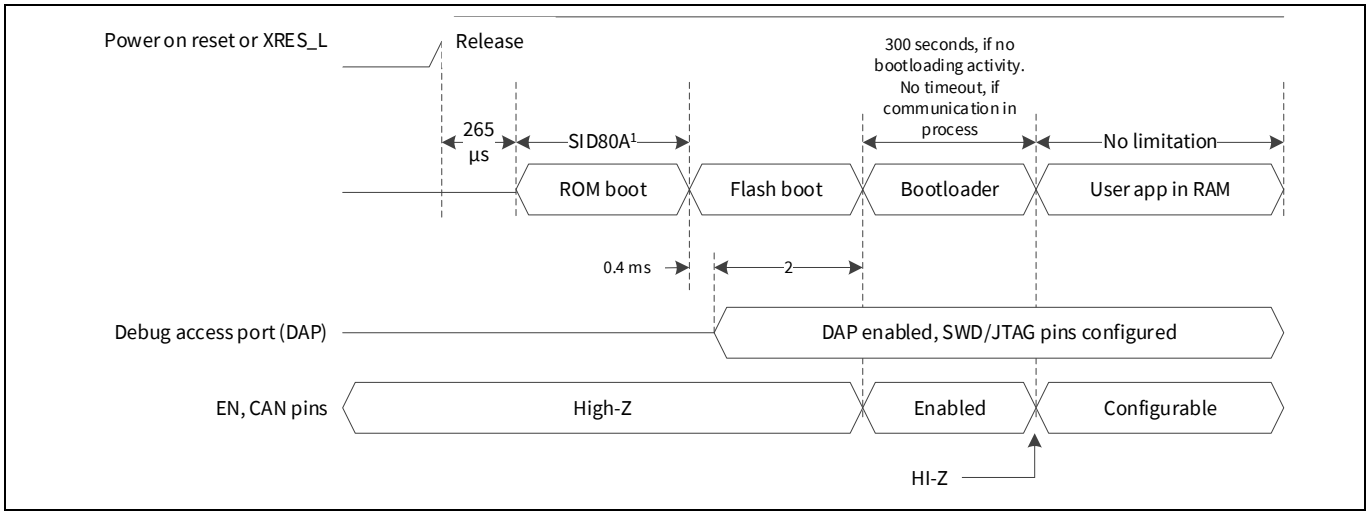

Figure 4 Startup timing

1.4.1 CAN configuration

Table 1 shows the CAN configuration. **Figure 5** shows the CAN interface configuration. The flash boot sets two EN pins and a TX pin as strong drive mode on entering the bootloader. The RX pin is in the High-Z mode. Before or after the bootloader, these pins are configured in the High-Z mode. The EN pins can be used to enable the CAN transceiver. If you keep the CAN transceiver always enabled, you do not need to use the EN pins.

Table 1 CAN configuration

Parameter	Configuration
	XMC7100, XMC7200
CAN instance	CAN0_1
TX pin	P0.2/CAN0_1_TX
RX pin	P0.3/CAN0_1_RX
EN (HIGH) pin	P2.1 (optional)
EN (LOW) pin	P23.3 (optional)
CAN mode	CAN classic mode (CAN FD mode is not in use)
Baud rate	100 kbps or 500 kbps
RX message ID	0x1A1
TX message ID	0x1B1
Phase segment 1	39 tq (time quantum)
Phase segment 2	10 tq (time quantum)
SJW (Resynchronization jump width)	10 tq (time quantum)
Sampling point	80 %

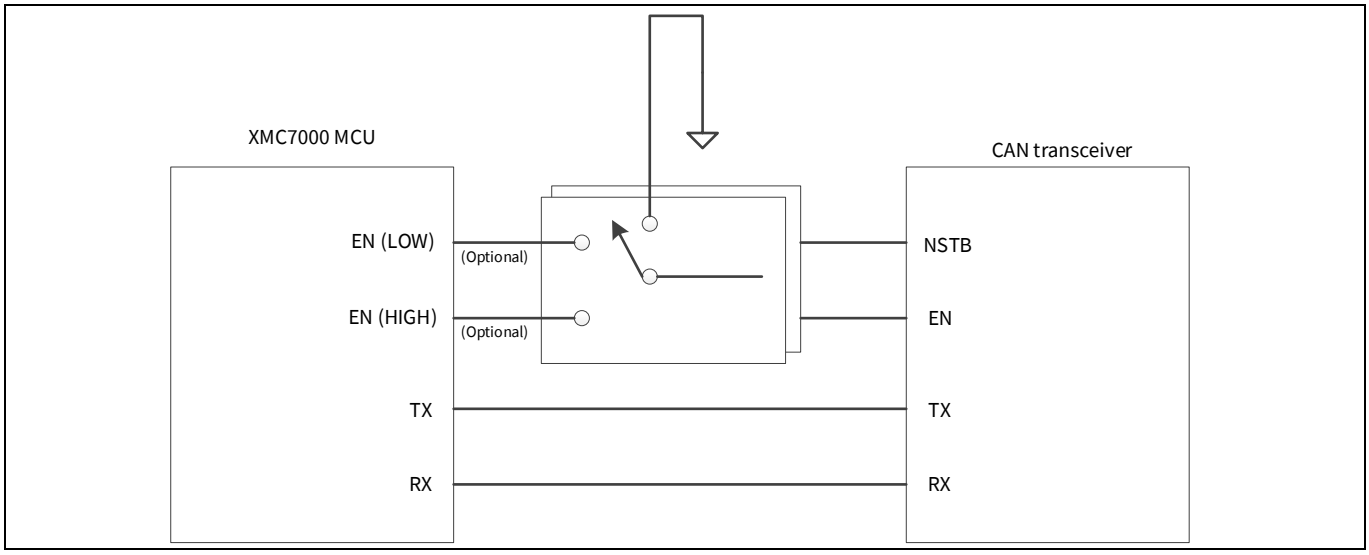

Figure 5 CAN interface configuration

1.5 Communication flow

Figure 6 shows the example of a communication flow between the host and bootloader. **Figure 6** gives the order in which commands are issued to the target and responses are received. See **Command/response packet structure** and **Commands** for a complete list of bootloader commands, their codes, and their expected responses.

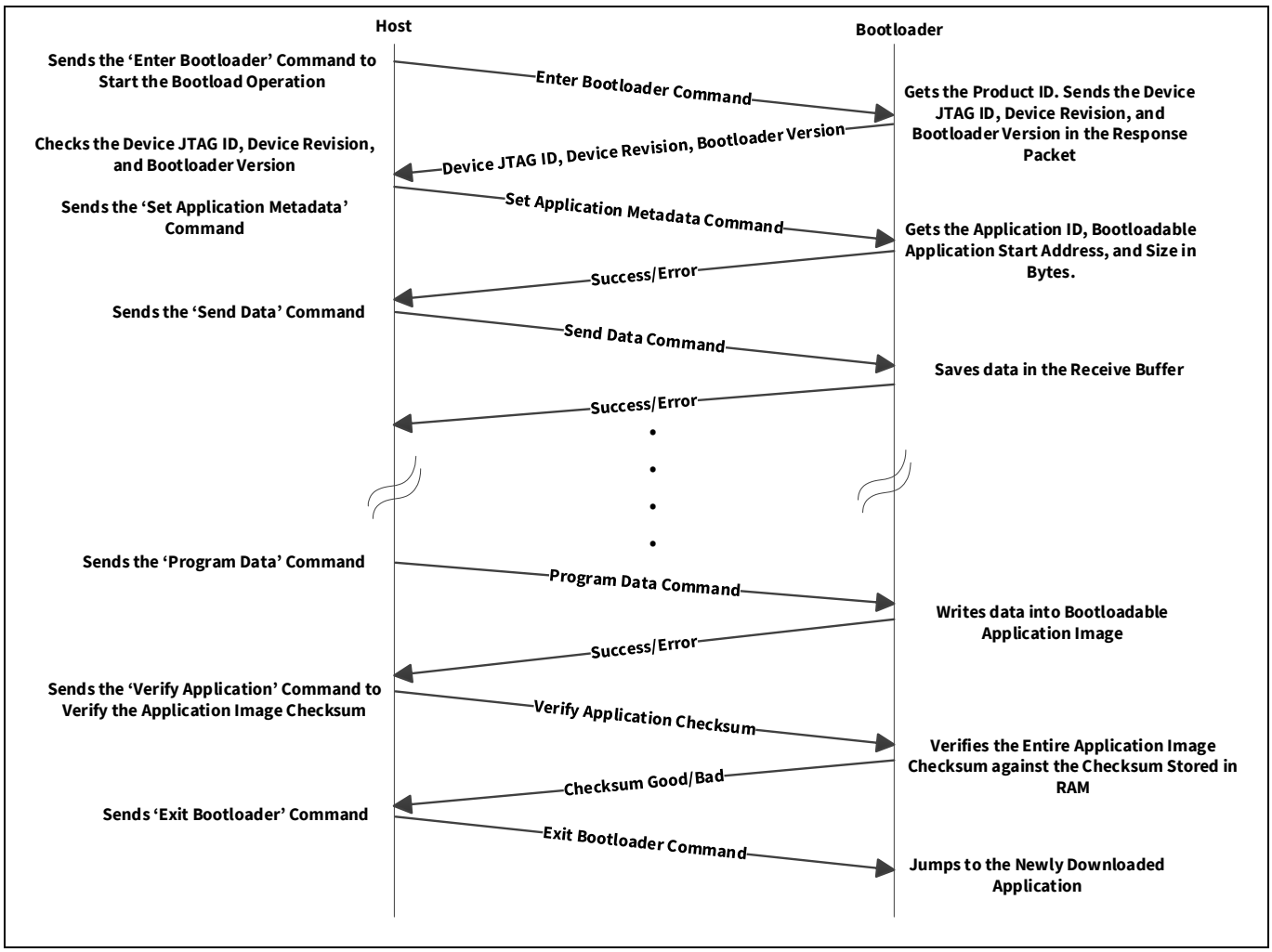

Figure 6 Communication flow

1.6 Command/response packet structure

The commands and responses are in the form of a byte stream, packetized in a manner that ensures the integrity of the data being transmitted. Each packet includes checksum bytes. The checksum is a basic summation (2's complement).

When sending multi-byte data such as data length and checksum, the least significant byte is sent first. The bootloader packet length is limited to four CAN messages, each with 8 bytes of data. Each CAN message can contain up to 8 bytes of user data, which hold bootloader command data. The message length needs to be adapted to the actual packet size.

Figure 7 shows the structure of the communication packets sent from the host to the bootloader.

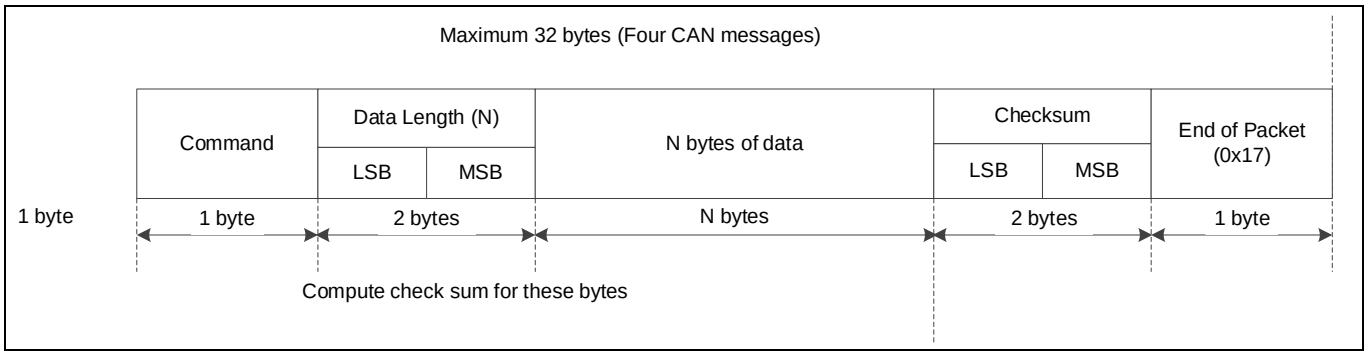

Figure 7 Command packet structure

Figure 8 shows the structure of the response packets sent from the bootloader to the host.

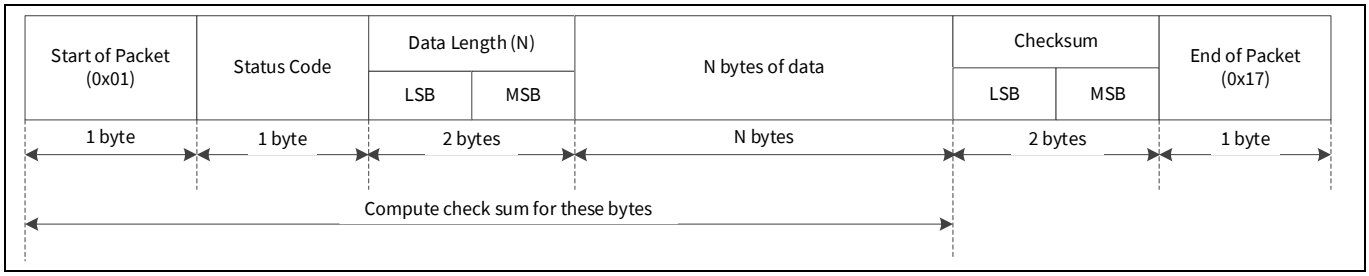

Figure 8 Response packet structure

The bootloader responds to each command from the host with a response packet. The format of the response packet is similar to the command packet except that there is a status code instead of the command code.

1.7 Commands

Table 2 shows a list of commands supported by the bootloader. All commands except 'Exit Bootloader' are ignored until the 'Enter Bootloader' command is received.

Table 2 Commands list

Commands		
Enter/Exit	**Bootload operation**	**Miscellaneous**
Enter Bootloader	Send Data	Verify Application
Sync Bootloader	Send Data Without Response	Set Application Metadata
Exit Bootloader	Program Data	

There is no specific requirement for command execution time.

Table 3 shows a list of status codes supported by the bootloader.

Table 3 Status codes list

Status code	Value	Description
CY_BOOTLOAD_SUCCESS	0x00	Bootload operation successful
CY_BOOTLOAD_ERROR_VERIFY	0x02	Error verifying the application image
CY_BOOTLOAD_ERROR_LENGTH	0x03	Unexpected or incorrect data length
CY_BOOTLOAD_ERROR_DATA	0x04	Data in bootloader command packet is incorrect
CY_BOOTLOAD_ERROR_CMD	0x05	Command byte is not recognized
CY_BOOTLOAD_ERROR_CHECKSUM	0x08	Bootloader packet has incorrect checksum
CY_BOOTLOAD_ERROR_ROW	0x0A	Incorrect address to bootload an application
CY_BOOTLOAD_ERROR_ROW_ACCESS	0x0B	Address cannot be accessed due to MPU or SWPU protection
CY_BOOTLOAD_UNKNOWN	0x0F	Any other error condition

1.7.1 Enter bootloader

This command begins a bootloading operation. All other commands except 'Exit Bootloader' are ignored until this command is received. This command responds with the device information and bootloader version.

Input

- Command byte: 0x38
- Data bytes:
  - 4 bytes: Product ID. Must be 0x01020304.

Output

- Status codes:
  - Success
  - Error command
  - Error data used for product ID mismatch
  - Error length
  - Error checksum
- Data bytes:
  - 4 bytes: Device JTAG ID
  - 1 byte: Device revision
  - 3 bytes: Bootloader version

1.7.2 Sync bootloader

This command resets the bootloader communication to the initial state, making it ready to accept a new command. Any data that was buffered is discarded. This command is needed only if the bootloader and the host get out of sync with each other.

Input

- Command byte: 0x35
- Data bytes: N/A

Output

- N/A – This command is not acknowledged

1.7.3 Exit bootloader

This command stops listening for other bootloader commands and jumps to the newly downloaded application (flash loader).

Input

- Command byte: 0x3B
- Data bytes: N/A

Output

- N/A – This command is not acknowledged

1.7.4 Send data

This command transfers a block of data to the bootloader. This data is buffered in anticipation of a **Program Data** command. The bootloader buffer size for the data received by **Send Data** and **Program Data** commands is 256 bytes. If the data is not programmed using **Program Data** and the data is still sent, the buffer will overflow and the `CY_BOOTLOAD_ERROR_LENGTH` error will be sent in the response packet. If a sequence of multiple send data commands is sent, the data is appended to the previous block.

This command is used to break up large data transfers into smaller pieces to prevent channel starvation in some communication protocols. If the host uses the **Verify Application** command, the checksum (CRC-32C) for the entire application must be placed in the last four bytes of the application image.

Input

- Command byte: 0x37
- Data bytes:
  - n bytes: Data to write

Output

- Status codes:
  - Success
  - Error command
  - Error data
  - Error length
  - Error checksum
- Data bytes: N/A

1.7.5 Send data without response

This command is the same as the **Send Data** command, except that no response is generated by the bootloader. This reduces bootloading time for some applications.

Input

- Command byte: 0x47
- Data bytes:
  - n bytes: Data to write

Output

- N/A

1.7.6 Program data

This command writes data into the bootloadable application image, and might follow a series of **Send Data** or **Send Data Without Response** commands.

Input

- Command byte: 0x49
- Data bytes:
  - 4 bytes: Address. Must be aligned to 256 bytes and within a valid RAM memory length – [RAM_START + 3 KB, RAM_END – 6 KB].
  - 4 bytes: CRC-32C of the entire n bytes of the data in the buffer, which has been previously transferred using the **Send Data** command.
  - n bytes: An arbitrary value.

Output

- Status codes:
  - Success
  - Error command
  - Error data
  - Error length
  - Error checksum
  - Error row
  - Error row access
- Data bytes: N/A

1.7.7 Verify application

This command reports whether the checksum (CRC-32C) for the entire application image (flash loader) in the RAM is valid. The host can decide to use the **Verify Application** command or to skip it. The checksum (CRC-32C) for the entire application must be placed in the last four bytes of the application image.

Input

- Command byte: 0x31
- Data bytes:
  - 1 byte: Application ID of the application to be verified. Must be the same value as in the **Set Application Metadata** command.

Output

- Status codes:
  - Success
  - Error command
  - Error data
  - Error length
  - Error checksum
  - Error row access
- Data bytes:
  - 1 byte: 0x01 indicates that application is valid. 0x00 indicates that application is invalid.

1.7.8 Set application metadata

This command is used to set a given application's metadata. This command must be the second bootloader command that the host delivers to the MCU, the first one being **Enter Bootloader**.

Input

- Command byte: 0x4C
- Data bytes:
  - 1 byte: Application ID

Table 4 shows the values of application ID.

Table 4 Application ID

Application ID value	Description
0	For CAN

  - 4 bytes: Bootloadable application start address. Must be aligned to 256 bytes and within a valid RAM memory length – [RAM_START + 3 KB, RAM_END – 6 KB].
  - 4 bytes: Bootloadable application size in bytes. Must be a value for which the bootloadable application image fits into a RAM address range [RAM_START + 3 KB, RAM_END – 6 KB].

Output

- Status codes:
  - Success
  - Error command
  - Error length
  - Error data
  - Error checksum
  - Error row access
- Data bytes: N/A

1.8 Application format

Figure 9 shows an example of an application format. If the host uses the **Verify Application** command, the checksum (CRC-32C) for the entire application must be placed in the last four bytes of the application image.

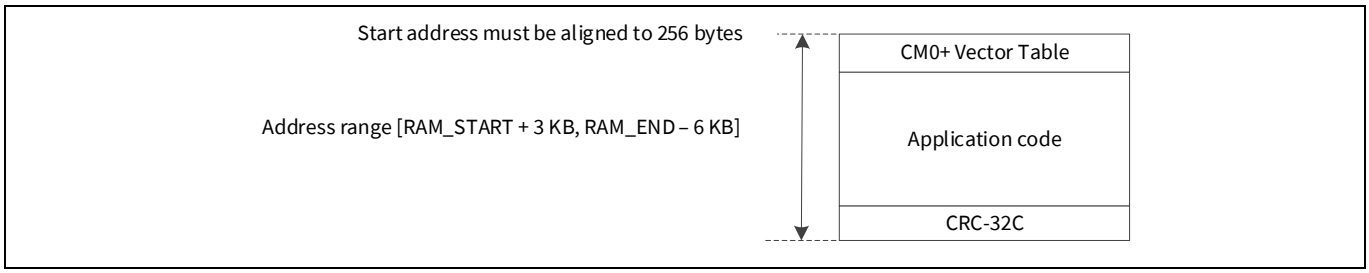

Figure 9 Example for application format

1.9 Example command/response data

Table 5 shows the example data for each command/response. If a sequence of multiple Send Data commands is sent, the data is appended to the previous block. This command is used to break up large data transfers into smaller pieces to prevent channel starvation in some communication protocols.

Table 5 Example command/response data

Command/ response	Start of packet	Command/ status code	Data length	N bytes of data	Checksum	End of packet
Enter Bootloader	0x01	0x38	0x04, 0x00	0x04, 0x03, 0x02, 0x01	0xB9, 0xFF	0x17
Response	0x01	0x00	0x08, 0x00	0x00, 0x00, 0x00, 0x00, 0x00, 0x14, 0x02, 0x01	0xE0, 0xFF	0x17
Set Application Metadata	0x01	0x4C	0x09, 0x00	0x00, 0x00, 0x40, 0x00, 0x08, 0xFC, 0x7F, 0x00, 0x00	0xE7, 0xFD	0x17
Response	0x01	0x00	0x00, 0x00	–	0xFF, 0xFF	0x17
Send Data	0x01	0x37	0x19, 0x00	0x00, 0xE0, 0x00, 0x08, 0xF1, 0x49, 0x00, 0x08, 0x7F, 0x49, 0x00, 0x08, 0xF9, 0x4A, 0x00, 0x08, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00	0x6A, 0xFB	0x17
Response	0x01	0x00	0x00, 0x00	–	0xFF, 0xFF	0x17
Program Data	0x01	0x49	0xE0, 0x00	0x00, 0x40, 0x00, 0x08, 0x91, 0xE6, 0x0D, 0xD8, 0xFF, 0xFF, 0xFF 0xFF, 0xFF, 0xFF	0x0A, 0xF7	0x17
Response	0x01	0x00	0x00, 0x00	-	0xFF, 0xFF	0x17
Verify Application	0x01	0x31	0x01, 0x00	0x00	0xCD, 0xFF	0x17
Response	0x01	0x00	0x01, 0x00	0x01	0xFD, 0xFF	0x17
Exit Bootloader	0x01	0x3B	0x00, 0x00	-	0xC4, 0xFF	0x17

2 Glossary

Table 6 Glossary

Terms	Description
CAN FD	Controller area network with flexible data rate
CRC	Cyclic redundancy check
DAP	Debug access port
JTAG	Joint Test Action Group
MPU	Memory protection unit
SJW	Resynchronization jump width
SWD	Single wire debug
TOC2	Table of contents 2
tq	Time quantum

3 Related documents

Refer to the following XMC7000 family series datasheets and technical reference manuals for more information. Contact **Technical Support** to obtain these documents.

- Device datasheet
  - Datasheet 32-Bit Arm® Cortex®-M7 Microcontroller XMC7100 Family (Doc No. 002-33896)
  - Datasheet 32-Bit Arm® Cortex®-M7 Microcontroller XMC7200 Family (Doc No. 002-33522)
- Technical Reference Manual (TRM)
  - XMC7000 Architecture Technical Reference Manual (Doc No. 002-33816)
  - XMC7100 Register Technical Reference Manual (Doc No. 002-33817)
  - XMC7200 Register Technical Reference Manual (Doc No. 002-33812)

Revision history

Document version	Date of release	Description of changes
**	2022-05-06	New application note.

Trademarks
All referenced product or service names and trademarks are the property of their respective owners.

Edition 2022-05-06
Published by
Infineon Technologies AG
81726 Munich, Germany

Do you have a question about this document?
Go to www.cypress.com/support

Document reference
002-35060 Rev. **

IMPORTANT NOTICE
The information contained in this application note is given as a hint for the implementation of the product only and shall in no event be regarded as a description or warranty of a certain functionality, condition or quality of the product. Before implementation of the product, the recipient of this application note must verify any function and other technical information given herein in the real application. Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind (including without limitation warranties of non-infringement of intellectual property rights of any third party) with respect to any and all information given in this application note.

The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer's technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

For further information on the product, technology, delivery terms and conditions and prices please contact your nearest Infineon Technologies office (**www.infineon.com**).

WARNINGS
Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies' products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.