Traveo II Bootloader

Author: Kenichi Sunada

Associated Part Family: Traveo™ II Family

Related Documents: For a complete list, see Related Documents

This application note describes a CAN/LIN-based bootloader for Traveo II Family. This application note also explains how to communicate with a CAN/LIN-based bootloader.

Contents

1 Introduction

Bootloaders are commonly present in an MCU system design. A bootloader makes 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) 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 or LIN. A bootloader communicates with a host to get new application code or data and writes it into the device's flash memory.

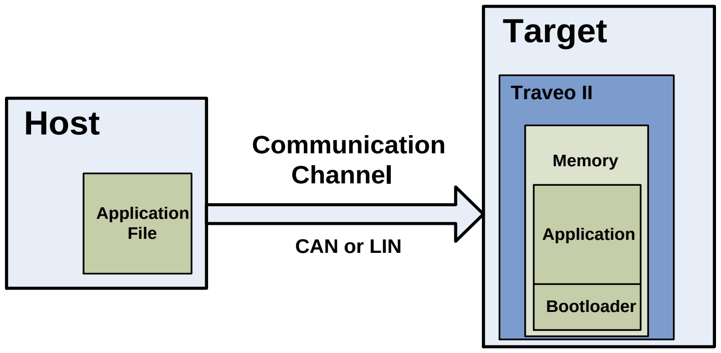
In this application note, you will learn how to communicate with a CAN/LIN-based bootloader.

This application note assumes that you are familiar with bootloader concepts, CAN and LIN protocol. For more details on CAN and LIN Components, see the “Flash Boot”, “CAN FD Controller”, and “Local Interconnect Network” 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 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

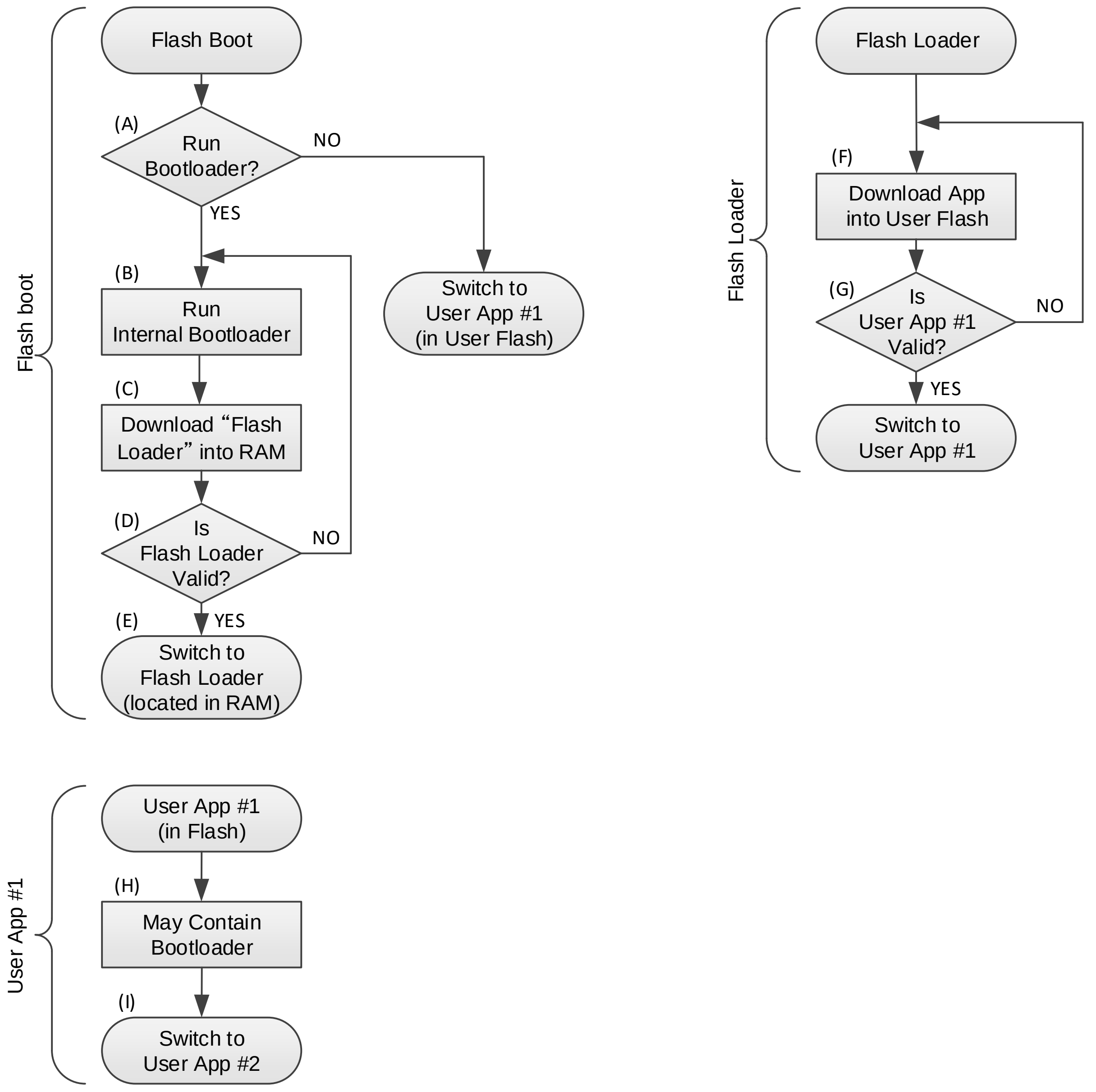
During bootloading, the host reads the file for the new application, parses it with the commands downloaded to RAM, and sends those commands to the bootloader. After the entire file is sent, the bootloader can pass control to the new application.

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



(A) The flash boot checks if 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 RAM (C) and launch it (E).

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

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

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

(H) The user application may or may not contain a bootloader. It is up to 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.

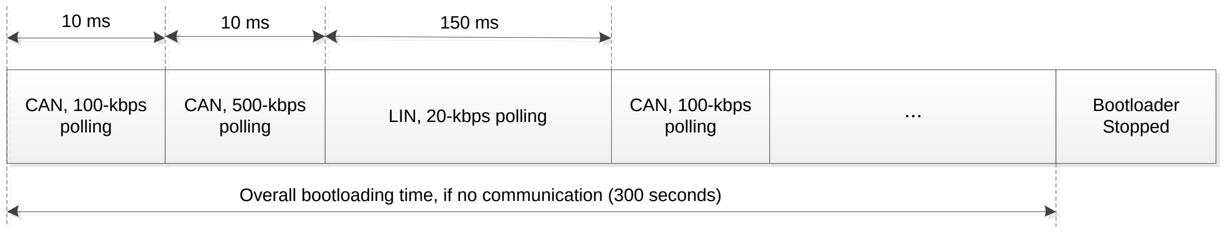
1.4 Device Interface Configurations

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

- Two words at the start of the flash must be both equal to ‘0’ or ‘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. If there is a timeout, the channel is reconfigured for LIN and it again waits for the frame. 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.

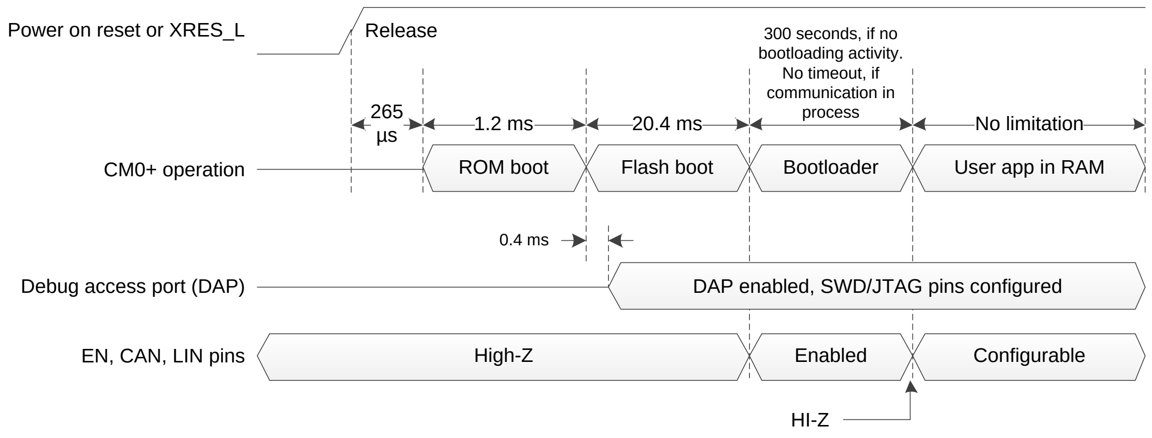
Figure 3. Bootloader Polling Sequence



If an Enter Bootloader command is received on either of the communication interface, the polling stops and the bootloader starts using this interface only. If the bootloading succeeds, the bootloader launches the updated application in 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 flash, the internal bootloader is no longer launched.

Figure 4. Startup Timing



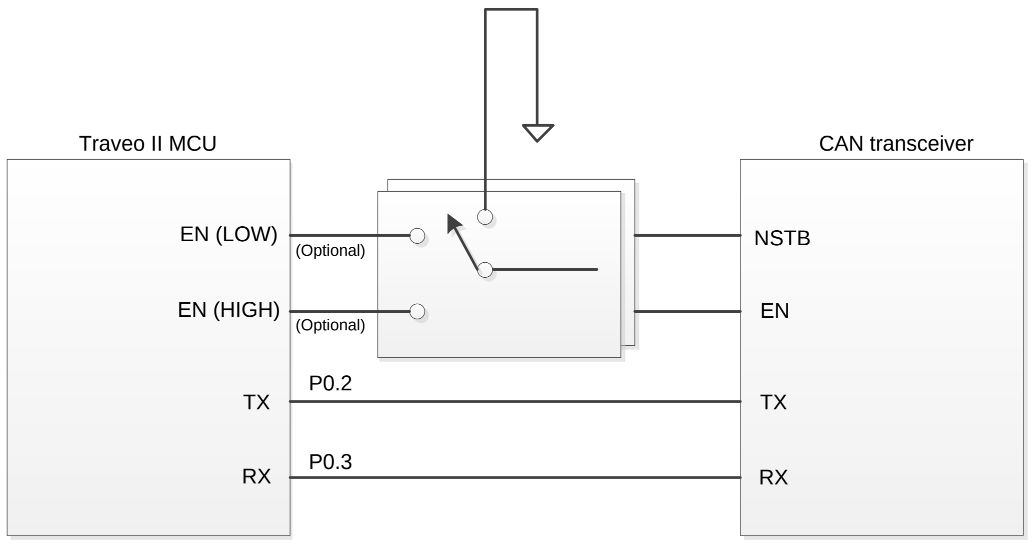
1.4.1 CAN Configuration

Table 1 shows the CAN configuration. Figure 5 shows the CAN interface configuration. The flash boot sets two EN pins as strong drive outputs on entering the bootloader. Before or after the bootloader, EN pins are configured as the high impedance inputs. 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	
	CYT2B, CYT4B	CYT4D
CAN instance	CAN0_1	CAN1_0
TX pin	P0.2/CAN0_1_TX	P2.3/CAN1_0_TX
RX pin	P0.3/CAN0_1_RX	P2.4/CAN1_0_RX
EN (HIGH) pin	P2.1 (optional)	P0.2 (optional)
EN (LOW) pin	P23.3 (optional)	P0.5 (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)	5 tq (time quantum)	
Sampling point	80 %	

Figure 5. CAN Interface Configuration



1.4.2 LIN Configuration

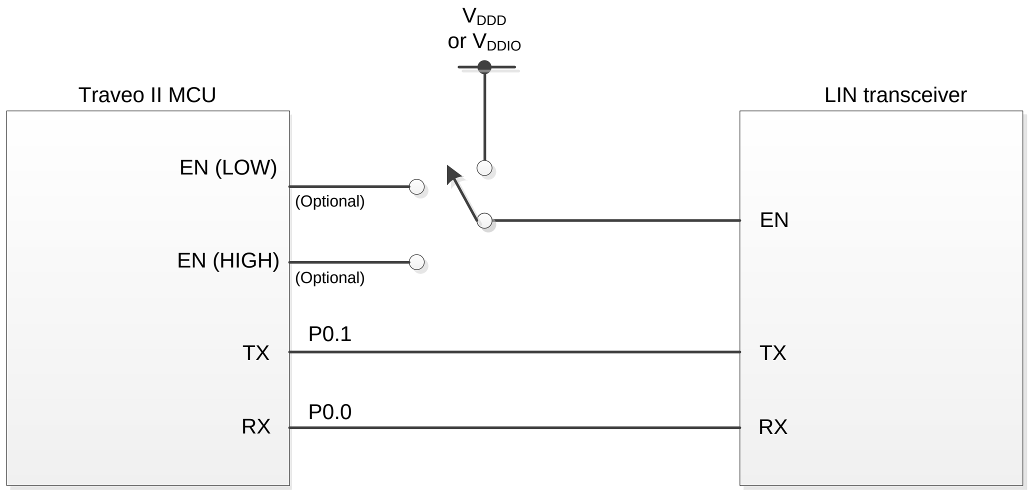
Table 2 shows the LIN configuration. Figure 6 shows the LIN interface configuration. Note that not all LIN transceivers support 115.2 kbps (Fast mode).

Table 2. LIN Configuration

Parameter	Configuration	
	CYT2B, CYT4B	CYT4D
LIN instance	LIN1	
TX pin	P0.1/LIN1_TX	
RX pin	P0.0/LIN1_RX	
EN (HIGH) pin	P2.1 (optional)	P0.2 (optional)
EN (LOW) pin	P23.3 (optional)	P0.5 (optional)

Parameter	Configuration	
	CYT2B, CYT4B	CYT4D
LIN mode	Slave	
Baud rate	20 kbps or 115.2 kbps (as an option for fast flash programming)	
Break field length	11 bits	
Break delimiter length	1 bit	
Stop bit	1 bit	
PID (RX)	45	
PID (TX)	46	
Checksum type	Classic	

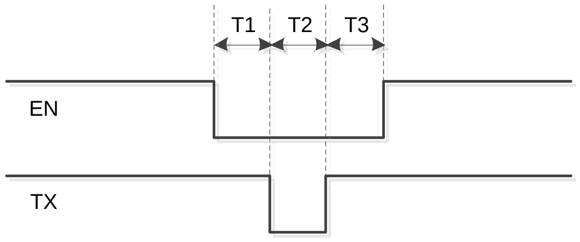
Figure 6. LIN Interface Configuration



1.4.3 LIN Configuration for 115.2 kbps

Some LIN transceivers require the special signals on TX and EN pins, as shown in Figure 7, to enter a Fast mode which supports 115.2 kbps. Some LIN transceivers support 115.2 kbps without the special signals.

Figure 7. Signals Timing for Fast Mode



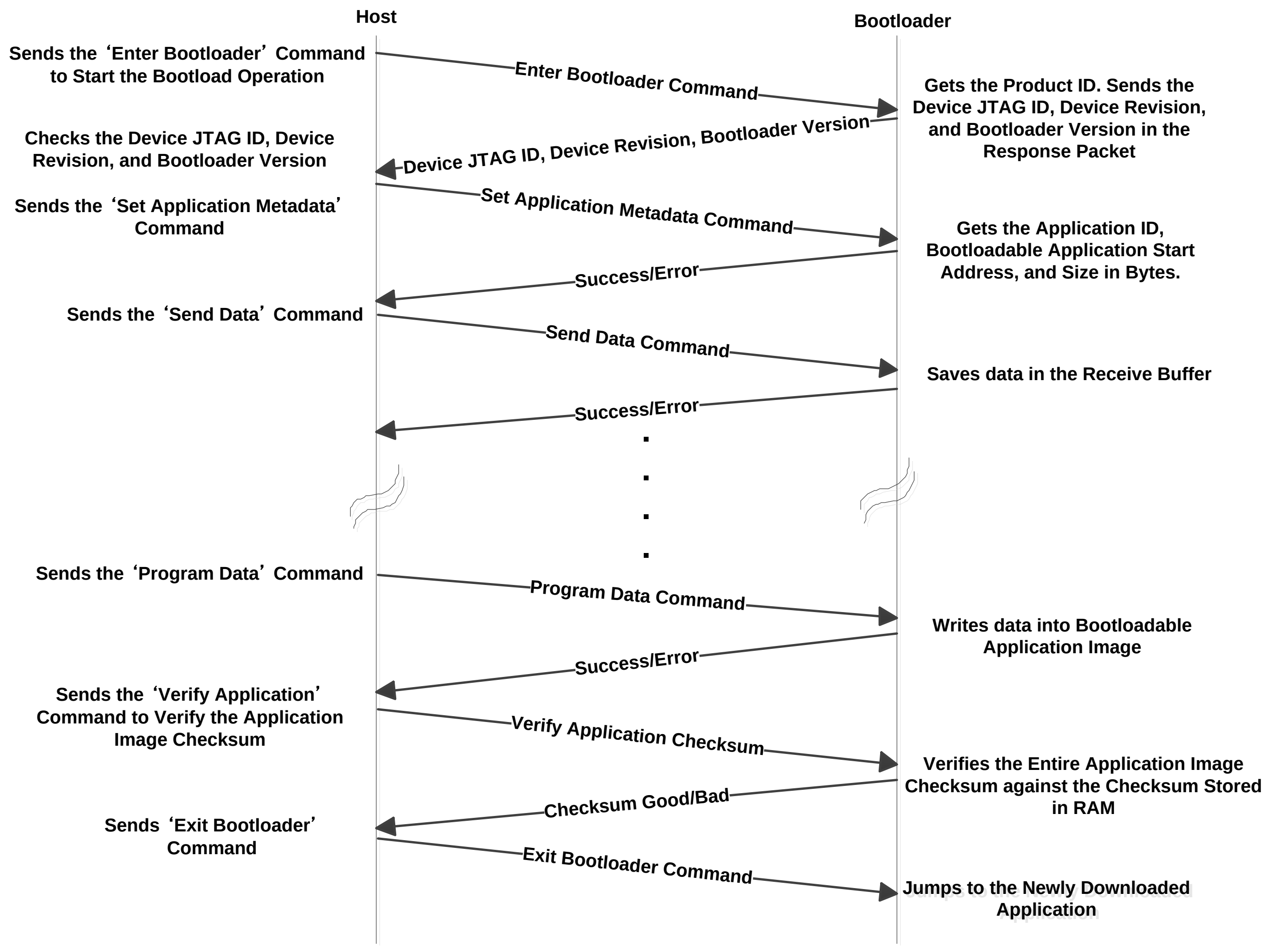
T1 = T2 = T3 = 12 μs

First, the bootloader waits for an Enter Bootloader command on LIN at 20 kbps. When the Enter Bootloader command is received, the bootloader expects the next command to be Set Application Metadata. If Set Application Metadata has Application ID = ‘0’, the bootloader continues at 20 kbps. If Set Application Metadata has Application ID = ‘1’, the bootloader switches to 115.2 kbps using the special signals. If Set Application Metadata has Application ID = ‘2’, the bootloader switches to 115.2 kbps without the special signals.

1.5 Communication Flow

Figure 8 shows the example of a communication flow between the host and bootloader. Figure 8 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 8. Communication Flow


Host
Bootloader
Sends the 'Enter Bootloader' Command to Start the Bootload Operation
Enter Bootloader Command
Gets the Product ID. Sends the Device JTAG ID, Device Revision, and Bootloader Version in the Response Packet
Device JTAG ID, Device Revision, Bootloader Version
Checks the Device JTAG ID, Device Revision, and Bootloader Version
Sends the 'Set Application Metadata' Command
Set Application Metadata Command
Gets the Application ID, Bootloadable Application Start Address, and Size in Bytes.
Success/Error
Sends the 'Send Data' Command
Send Data Command
Saves data in the Receive Buffer
Success/Error
Sends the 'Program Data' Command
Program Data Command
Writes data into Bootloadable Application Image
Success/Error
Sends the 'Verify Application' Command to Verify the Application Image Checksum
Verify Application Checksum
Verifies the Entire Application Image Checksum against the Checksum Stored in RAM
Checksum Good/Bad
Sends 'Exit Bootloader' Command
Exit Bootloader Command
Jumps to the Newly Downloaded Application

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 multibyte data such as Data Length and Checksum, the least significant byte is sent first. Bootloader packet length is limited to four CAN or LIN messages, each with 8 bytes of data. Each CAN or LIN 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 9 shows the structure of the communication packets sent from the host to the bootloader.

Figure 9. Command Packet Structure

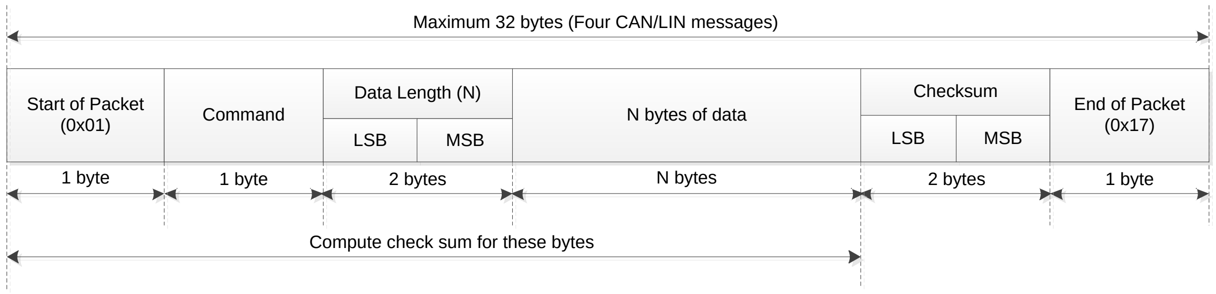
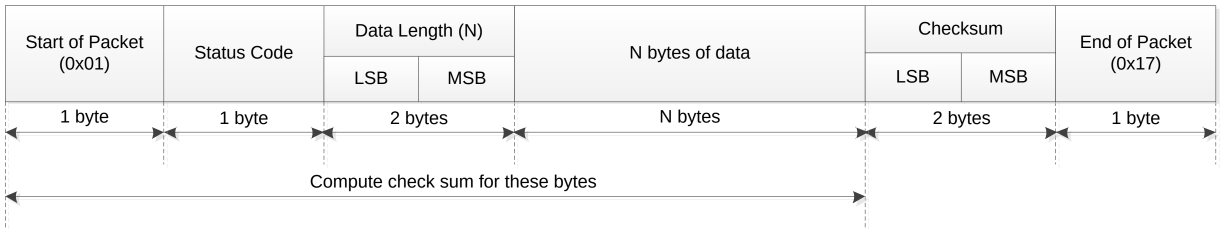


Figure 10 shows the structure of the response packets sent from the bootloader to the host.

Figure 10. 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 will be a status code instead of the command code.

1.7 Commands

Table 3 shows a list of commands supported by the bootloader. All commands except Exit Bootloader are ignored until the Enter Bootloader command is received.

Table 3. 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 4 shows a list of status codes supported by the bootloader.

Table 4. Status Codes List

Status Code	Value	Description
CY_BOOTLOAD_SUCCESS	0x00	Successful status
CY_BOOTLOAD_ERROR_VERIFY	0x02	Error verifying application image
CY_BOOTLOAD_ERROR_LENGTH	0x03	Unexpected or wrong data length
CY_BOOTLOAD_ERROR_DATA	0x04	Data in bootloader command packet is wrong
CY_BOOTLOAD_ERROR_CMD	0x05	Command byte is not recognized
CY_BOOTLOAD_ERROR_CHECKSUM	0x08	Bootloader packet has wrong checksum
CY_BOOTLOAD_ERROR_ROW	0x0A	Wrong 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 device information and the 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 command is 256 bytes of data. If the data is not programmed using Program Data and the data is still sent, the buffer will overflow and CY_BOOTLOAD_ERROR_LENGTH error will be send 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 needs to be placed in the last 4 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 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 + 512, RAM_END -4096].
    - 4 bytes: CRC-32C of the entire n bytes of the data in the buffer which has been previous 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 RAM is valid. The host can decide to use Verify Application command or to skip it. The checksum (CRC-32C) for the entire application needs to be placed in the last 4 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 which the host delivers to the MCU; the first one being Enter Bootloader.

- Input
  - Command Byte: 0x4C
  - Data Bytes:
    - 1 byte: Application ID

      Table 5 shows the values of application ID.

      Table 5. Application ID

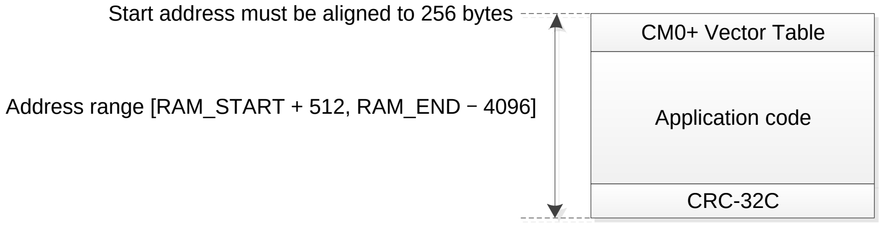
Application ID Value	Description
0	For either LIN at 20 kbps or CAN
1	For LIN at 115.2 kbps with a Fast mode. See LIN Configuration for 115.2 kbps.
2	For LIN at 115.2 kbps without a Fast mode.

    -
    - 4 bytes: Bootloadable application start address. Must be aligned to 256 bytes and within a valid RAM memory length – [RAM_START + 512, RAM_END – 4096].
    - 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 + 512, RAM_END -4096].
- Output
  - Status Codes:
    - Success
    - Error Command
    - Error Length
    - Error Data
    - Error Checksum
    - Error Row Access
  - Data Bytes: N/A

1.8 Application Format

Figure 11 shows an example of an application format. If the host uses the Verify Application command, the checksum (CRC-32C) for the entire application needs to be placed in the last 4 bytes of the application image.

Figure 11. Example for Application Format


Start address must be aligned to 256 bytes
Address range [RAM_START + 512, RAM_END − 4096]
CM0+ Vector Table
Application code
CRC-32C

1.9 Example Command/Response Data

Table 6 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 6. 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, 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

Terms	Description
CAN FD	Controller Area Network with Flexible Data rate
CRC	Cyclic Redundancy Check
DAP	Debug Access Port
JTAG	Joint Test Action Group
LIN	Local Interconnect Network
MPU	Memory Protection Unit
SJW	Resynchronization Jump Width
SWD	Single Wire Debug
TOC2	Table of Contents 2
tq	Time Quantum

3 Related Documents

The following are the Traveo II family series datasheets and technical reference manuals. Contact Technical Support to obtain these documents.

- Device datasheet
  - CYT2B7 Datasheet 32-Bit Arm® Cortex®-M4F Microcontroller Traveo™ II Family
  - CYT2B9 Datasheet 32-Bit Arm® Cortex®-M4F Microcontroller Traveo™ II Family
  - CYT4BF Datasheet 32-Bit Arm® Cortex®-M7 Microcontroller Traveo™ II Family
  - CYT4DN Datasheet 32-Bit Arm® Cortex®-M7 Microcontroller Traveo™ II Family
- CYT2B Series
  - Traveo™ II Automotive Body Controller Entry Family Architecture Technical Reference Manual (TRM)
  - Traveo™ II Automotive Body Controller Entry Registers Technical Reference Manual (TRM) for CYT2B7
  - Traveo™ II Automotive Body Controller Entry Registers Technical Reference Manual (TRM) for CYT2B9
- CYT4B Series
  - Traveo™ II Automotive Body Controller High Family Architecture Technical Reference Manual (TRM)
  - Traveo™ II Automotive Body Controller High Registers Technical Reference Manual (TRM)
- CYT4D Series
  - Traveo™ II Automotive Cluster 2D Family Architecture Technical Reference Manual (TRM)
  - Traveo™ II Automotive Cluster 2D Registers Technical Reference Manual (TRM)

Document History

Document Title: AN227076 - Traveo II Bootloader

Document Number: 002-27076

Revision	ECN	Submission Date	Description of Change
**	6648564	08/23/2019	New application note.

Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

Products

Arm® Cortex® Microcontrollers	cypress.com/arm
Automotive	cypress.com/automotive
Clocks & Buffers	cypress.com/clocks
Interface	cypress.com/interface
Internet of Things	cypress.com/iot
Memory	cypress.com/memory
Microcontrollers	cypress.com/mcu
PSoC	cypress.com/psoc
Power Management ICs	cypress.com/pmic
Touch Sensing	cypress.com/touch
USB Controllers	cypress.com/usb
Wireless Connectivity	cypress.com/wireless

PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6 MCU

Cypress Developer Community

Community | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

All other trademarks or registered trademarks referenced herein are the property of their respective owners.

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709