

MAX78000 Bootloader

User Guide

UG6471; Rev 1; 10/21

Abstract

The MAX78000 bootloader user guide provides flow charts; timing diagrams; GPIOs/pin usage; I2C, SPI, and UART interface protocols and an annotated trace between the host microcontroller; MAX78000 bootloader protocol definitions; and the MAX78000 for in-application programming (IAP).

Table of Contents

[Overview 6](#_Toc86363614)

[Detailed Description 7](#_Toc86363615)

[MAX78000 Bootloader Memory Map 9](#_Toc86363616)

[Bootloader Pin Definitions 10](#_Toc86363617)

[Activating the Bootloader 10](#_Toc86363618)

[Entering Bootloader Mode from Application Mode 10](#_Toc86363619)

[Host Serial Command Using Power-On or Hard Reset 10](#_Toc86363620)

[Without Using the RSTN Pin or GPIO Pins 10](#_Toc86363621)

[Using the Enter Bootloader GPIO Pin and the RSTN Pin 11](#_Toc86363622)

[Entering Application Mode from Bootloader Mode 12](#_Toc86363623)

[A Valid Application Is Programmed 12](#_Toc86363624)

[Using the EBL GPIO Pin and the RSTN Pin 12](#_Toc86363625)

[Configuring the Bootloader 13](#_Toc86363626)

[Bootloader Configuration Parameters 13](#_Toc86363627)

[EBL Pin Check (1 bit) 13](#_Toc86363628)

[EBL Pin Assignment (4 bits) 13](#_Toc86363629)

[EBL GPIO Pin Polarity (1 bit) 14](#_Toc86363630)

[Valid Mark Check (1 bit) 14](#_Toc86363631)

[I2C Interface Selection (1 bit) 14](#_Toc86363632)

[CRC Check (1 bit) 14](#_Toc86363633)

[SWD Lock (1 bit) 14](#_Toc86363634)

[Timeout Mode (2 bits) 15](#_Toc86363635)

[Timeout Window (4 bits) 16](#_Toc86363636)

[I2C Slave Address Selection (7 bits) 16](#_Toc86363637)

[Bootloader Configuration Error Detection 17](#_Toc86363638)

[Bootloader Interfaces 18](#_Toc86363639)

[I2C Interface 18](#_Toc86363640)

[I2C Bit Transfer Process 18](#_Toc86363641)

[I2C Write 19](#_Toc86363642)

[I2C Read 20](#_Toc86363643)

[Communicating with the Bootloader 21](#_Toc86363644)

[MAX78000 Bootloader Message Protocol Definitions 21](#_Toc86363645)

[MAX78000 In-Application Programming, Annotated Trace 29](#_Toc86363646)

[Appendix A: Maxim Special Bootloader (.msbl) File Format 33](#_Toc86363647)

[Appendix C: How to Program the Application by Using a Binary File 35](#_Toc86363648)

[Revision History 39](#_Toc86363649)

List of Figures

[Figure 1. MAX78000 bootloader top-level flow chart. 7](#_Toc86363650)

[Figure 2. MAX78000 bootloader application loader flow chart. 8](#_Toc86363651)

[Figure 3. Entering bootloader mode through the EBL GPIO and RSTN pins. 11](#_Toc86363652)

[Figure 4. Entering application mode through the EBL GPIO and RSTN pins. 12](#_Toc86363653)

[Figure 5. Bootloader CRC checking. 17](#_Toc86363654)

[Figure 6. I2C write/read data transfer from host microcontroller. 18](#_Toc86363655)

[Figure 7. MAX78000 Bootloader Communication Protocol. 21](file:///C:\Users\reza.ahmadi\Documents\MAX78000_SecureBoot_cloned\secure-loader\MAX78000_MSBL\MAX78000_Bootloader_UG.docx#_Toc86363656)

[Figure 8. Hex header data in the .msbl. 33](#_Toc86363657)

List of Tables

[Table 1. Bootloader Memory Map 9](#_Toc86363658)

[Table 2. Bootloader Data Section Description 9](#_Toc86363659)

[Table 3. GPIO and RSTN Pin Descriptions 10](#_Toc86363660)

[Table 4. Read Status Byte Values 19](#_Toc86363661)

[Table 5. MAX78000 Bootloader Message Protocol Definitions 21](#_Toc86363662)

[Table 6. Application Programming Example by Using the .msbl File 29](#_Toc86363663)

[Table 7. Application Programming Example by Using Partial Pages 31](#_Toc86363664)

[Table 8. Example .msbl File Format 34](#_Toc86363665)

[Table 9. Binary File Application Programming Example 35](#_Toc86363666)

[Table 10. Binary File Partial Application Programming Example 37](#_Toc86363667)

Overview

The MAX78000 bootloader is embedded firmware that gives the the ability to update application code provided by a host microcontroller. The bootloader can be accessed through the I2C interface. This interface provides the data channel and the control channel for communicating between the host microcontroller and the . The bootloader application load mode is enabled and disabled by either a serial command or hardware connectivity. The serial command is interpreted by the user application, which configures the device to enter bootloader mode. When using the hardware connectivity option, a single GPIO pin and the RSTN pin on the can be configured to allow the to enter bootloader mode.

Detailed Description

**Figure 1** and **Figure 2** show the program flow for the bootloader.



Figure 1. bootloader top-level flow chart.

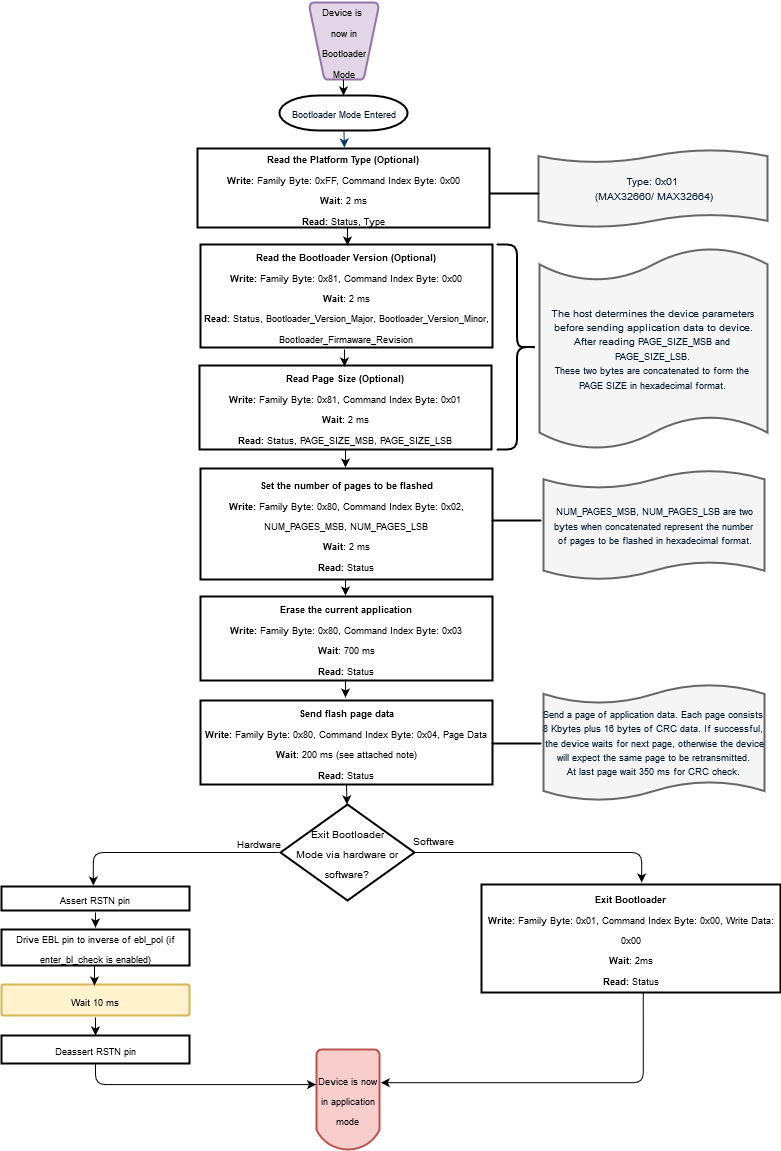


Figure 2. bootloader application loader flow chart.

MAX78000 Bootloader Memory Map

The MAX78000 bootloader memory map can be seen in Table 1.

Table 1. Bootloader Memory Map

|  |  |
| --- | --- |
| ADDRESS | DESCRIPTION |
| 0x10000000 - 0x10007FFF | Bootloader |
| 0x10008000 - 0x1007DFBF | Main Application |
| 0x1007DFC0 - 0x1007DFFF | Bootloader Data |
| 0x1007E000 - 0x1007FFFF | Rom Bootloader Page |

**Table 2** lists the descriptions of the bootloader data sections. The application start address is 0x4000 and the maximum size of an application that can be programmed is 245696 bytes.

Table 2. Bootloader Data Section Description

|  |  |  |
| --- | --- | --- |
| ADDRESS | DESCRIPTION | LENGTH (bytes) |
| 0x1007DFC0 | Application CRC | 4 |
| 0x1007DFC4 | Application length | 4 |
| 0x1007DFC8 | Valid mark | 4 |
| 0x1007DFCC | Boot mode | 4 |
| 0x1007DFD0 | Bootloader configuration | 8 |
| 0x1007DFD8 | Bootloader configuration CRC | 4 |
| 0x1007DFDC | RFU | 36 |

Bootloader Pin Definitions

**Table 3** lists the descriptions for the GPIO and RSTN pins of the MAX78000 bootloader.

Table 3. GPIO and RSTN Pin Descriptions

|  |  |  |
| --- | --- | --- |
| MAX78000 | DESCRIPTION | DIRECTION FROM MAX78000 SIDE |
| Pin RSTN | Reset\_N | Input |
| GPIO P0.16 | I2C1\_SCL | Input |
| GPIO P0.17 | I2C1\_SDA | Input/Output |

Activating the Bootloader

Entering Bootloader Mode from Application Mode

This section defines several methods for entering bootloader mode from application mode.

Host Serial Command Using Power-On or Hard Reset

The MAX78000 can enter bootloader mode by performing the following steps:

1. Power cycle the MAX78000 or perform a hard reset with the RSTN pin.
2. The host microcontroller sends the command 0x01, 0x00, 0x08 over the selected interface to the MAX78000 within 20ms of the reset operation. This is a signal to the cold boot process to enter bootloader mode.

Without Using the RSTN Pin or GPIO Pins

“Boot\_mode” is a 4-byte flag located at 0x3FFCC. Change the “boot\_mode” flag in the flash memory to 0xAAAAAAAA for staying in the bootloader even when there is a valid application in the memory. The number of write cycles to flash the memory is limited to 10,000 cycles. Consequently, this method should be not be used frequently. In addition, the bootloader firmware can become inoperable if power is lost during this operation or if the code is not implemented correctly.

The example code to implement this method is in the “main.c” file in the folder “example\Enter\_Bootloader.” If this method is used, the application code needs to implement code like the provided example.

Using the Enter Bootloader GPIO Pin and the RSTN Pin

Another method for entering bootloader mode is to use the enter bootloader (EBL) GPIO pin and the RSTN pin. The EBL pin is disabled in the bootloader by default and can be enabled by command. The MAX78000 enters bootloader mode based on the sequencing of the RSTN pin and the EBL pin.

The sequence to enter bootloader mode using the EBL GPIO pin and the RSTN pin is as follows:

1. Set the RSTN pin low for 10ms.
2. During that time, set the EBL GPIO pin to low. This polarity is configurable and active-low for bootloader mode by default.
3. After 10ms, set the RSTN pin high.
4. After an additional 20ms, the MAX78000 is in bootloader mode.

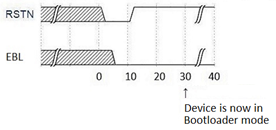


Figure 3. Entering bootloader mode through the EBL GPIO and RSTN pins.

Entering Application Mode from Bootloader Mode

This section discusses various methods of entering application mode from bootloader mode.

A Valid Application Is Programmed

If a valid application is programmed into the MAX78000 using in-application programming (IAP), the bootloader automatically runs the application code (assuming that the EBL GPIO pin is disabled) after reset.

Using the EBL GPIO Pin and the RSTN Pin

The MAX78000 enters application mode based on the sequencing of the EBL GPIO pin and the RSTN pin if there is a programmed valid application. The EBL GPIO pin is disabled in the bootloader by default and can be enabled by the serial commands.

The sequence to enter application mode using the EBL GPIO pin and the RSTN pin is as follows:

1. Set the RSTN pin low for 10ms.
2. During that time, set the EBL GPIO pin to high. This polarity is configurable and active-low for bootloader mode by default.
3. After 10ms, set the RSTN pin high.
4. After an additional 20ms, the MAX78000 is in application mode.

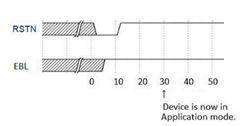


Figure 4. Entering application mode through the EBL GPIO and RSTN pins.

Configuring the Bootloader

Bootloader Configuration Parameters

Bootloader configuration parameters are used to enable and disable some functions of the bootloader. These parameters are located at the memory address 0x3FFD0. The bootloader configuration can be changed by the serial commands. Definitions and default values for the bit fields are provided as follows:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| BYTE # | BIT 7 | BIT 6 | BIT 5 | BIT 4 | BIT 3 | BIT 2 | BIT 1 | BIT 0 |
| BYTE 0 | RFU | RFU | EBL GPIO pin polarity | EBL pin assignment | | | | EBL pin check |
| BYTE 1 | RFU | RFU | RFU | RFU | RFU | RFU | I2C interface selection | RFU |
| BYTE 2 | RFU | RFU | Timeout mode | | Timeout window | | | |
| BYTE 3 | RFU | RFU | RFU | RFU | RFU | SWD lock | Valid mark check | CRC check |
| BYTE 4 | RFU | I2C slave address selection | | | | | | |
| BYTE 5 | RFU | RFU | RFU | RFU | RFU | RFU | RFU | RFU |
| BYTE 6 | RFU | RFU | RFU | RFU | RFU | RFU | RFU | RFU |
| BYTE 7 | RFU | RFU | RFU | RFU | RFU | RFU | RFU | RFU |

EBL Pin Check (1 bit)

According to the enter\_bl\_check bit, the bootloader checks the status of the EBL GPIO pin at startup. If the EBL pin is left floating after the EBL pin check is enabled, this can lead to unexpected behavior such as getting stuck in bootloader mode even if there is a valid application.

|  |  |
| --- | --- |
| BIT VALUE | OPERATION |
| 0 | Do not check EBL pin (Default) |
| 1 | Check EBL pin |

EBL Pin Assignment (4 bits)

The ebl\_pin bits are used to choose the EBL GPIO pin. The selected pin is checked at bootloader startup to make a decision for staying in the bootloader or jumping to the application if the EBL GPIO pin check is enabled by the enter\_bl\_check bit.

| BIT VALUE | OPERATION |
| --- | --- |
| 0b0000 | P0.0 |
| 0b0001 | P0.1 |
| 0b0010 | P0.2 |
| 0b0011 | P0.3 |
| 0b0100 | P0.4 |
| 0b0101 | P0.5 (Default) |
| 0b0110 | P0.6 |
| 0b0111 | P0.7 |
| 0b1000 | P0.8 |
| 0b1001 | P0.9 |
| 0b1010 | P0.10 |
| 0b1011 | P0.11 |
| 0b1100 | P0.12 |
| 0b1101 | P0.13 |
| 0b1110 | — |
| 0b1111 | — |

EBL GPIO Pin Polarity (1 bit)

The EBL GPIO pin is used to keep the device at bootloader mode after reset if enter\_bl\_check is enabled. The ebl\_pol bit defines whether the polarity EBL GPIO pin enters bootloader mode.

|  |  |
| --- | --- |
| BIT VALUE | OPERATION |
| 0 | Active-low signal puts the device in bootloader mode (Default) |
| 1 | Active-high signal puts the device in bootloader mode |

Valid Mark Check (1 bit)

The valid mark is a 32-bit value (0x4D41524B 'MARK') written by the bootloader to 0x3FFC8 after application loading. It is checked at every startup before jumping to the application and is fully controlled by the bootloader when the application is loaded by the bootloader. If this check is disabled, the bootloader does not check the valid mark, so the application can be downloaded by using SWD without using the bootloader.

|  |  |
| --- | --- |
| BIT VALUE | OPERATION |
| 0 | Do not check application valid mark |
| 1 | Check application valid mark (Default) |

I2C Interface Selection (1 bit)

The I2C interface selection is done according to the i2c\_enable bit.

|  |  |
| --- | --- |
| BIT VALUE | OPERATION |
| 0 | Interface is not used |
| 1 | Interface is enabled (Default) |

CRC Check (1 bit)

The CRC polynomial 0x04C11DB7 (CRC32) of the application is always checked after application programming. According to the crc\_check bit, the bootloader decides to check CRC32 of the application before jumping to the application at every startup.

|  |  |
| --- | --- |
| BIT VALUE | OPERATION |
| 0 | Do not check application CRC (Default)\* |
| 1 | Check application CRC |

SWD Lock (1 bit)

The MAX78000 bootloader has a debugger lock capability that can be enabled and disabled up to four time. The bootloader allows debuggers to run on MAX78000 to use debugging capabilities on default configurations. After SWD lock is enabled, debuggers cannot enter debugging mode on MAX78000 anymore. When SWD state is changed from locked to unlocked loaded keys and application is erased,

|  |  |
| --- | --- |
| BIT VALUE | OPERATION |
| 0 | SWD is not locked (Default) |
| 1 | SWD is locked |

Timeout Mode (2 bits)

The exit\_bl\_mode bits define timeout mode for the bootloader.

|  |  |
| --- | --- |
| BIT VALUE | OPERATION |
| 0b00 | Jump after 20ms |
| 0b01 | Wait for programmable delay (ebl\_timeout) (Default) |
| 0b10 | Remain in bootloader mode until exit command is received |
| 0b11 | — |

Timeout Window (4 bits)

The timeout window is the time to wait for a serial command from a host to stay in bootloader mode before jumping to a valid application. The wait time is calculated according to the following formula:

tWAIT= 20ms + 2EBL\_TIMEOUTms

|  |  |
| --- | --- |
| BIT VALUE | tWAIT (ms) |
| 0b0000 | 21 |
| 0b0001 | 22 |
| 0b0010 | 24 |
| 0b0011 | 28 |
| 0b0100 | 36 |
| 0b0101 | 52 |
| 0b0110 | 84 |
| 0b0111 | 148 |
| 0b1000 | 276 |
| 0b1001 | 532 |
| 0b1010 | 1044 |
| 0b1011 | 2068 |
| 0b1100 | 4116 |
| 0b1101 | 8212 |
| 0b1110 | 16404 |
| 0b1111 | 32788 |

As an example, if ebl\_timeout is selected as 0b0000, tWAIT is calculated as follows:

tWAIT= 20ms + 20ms = 21ms

I2C Slave Address Selection (7 bits)

The i2c\_addr bits define the selected I2C slave address. Default I2C slave address is 0x55 (which is equal to 0xAA as 8 bit address including RW bit). Note that the address representation is in 7 bits. Valid slave addresses are between 0x08 and 0x77. Both 0x08 and 0x77 are also valid addresses. It is not possible to configure the address to an invalid value from host but it is possible for configuration changes from the application software itself. In case of invalid configurations, bootloader ignores the value and uses its default slave address 0x55. To restore the functionality, a correct value has to be rewritten to I2C Slave Address Selection bits.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Slave Address | | | | | | |
| A6 | A5 | A4 | A3 | A2 | A1 | A0 |
| MSB |  | | | | | LSB |

Bootloader Configuration Error Detection

There are not multiple copies of the bootloader configuration on the flash. Due to possible problems while modifying the flash (e.g., loss of power), the bootloader configuration has a 4-byte CRC value at the end of the bootloader configuration bytes. The polynomial 0x104C11DB7 is used with a bit-reverse algorithm to generate the 32-bit CRC configuration value. The bootloader automatically calculates and updates the correct flash address when the configuration is updated in bootloader mode. However, if the configuration is modified from an application, the CRC value also needs to be updated. **Figure 5** shows the error checking mechanism.



Figure 5. Bootloader CRC checking.

Bootloader Interfaces

I2C Interface

The I2C bus expects SCL and SDA to be open-drain signals and that the SDA and SCL pad circuits are automatically configured as open-drain outputs for the MAX78000 bootloader. The I2C interface supports transfer rates up to 400kbit/s (fast mode). The I2C slave address is 0xAA at default and can be configured by using the i2c\_addr bits at bootloader configuration.

I2C Bit Transfer Process

The SDA and SCL signals are open-drain circuits. Each has an external pullup resistor that ensures each circuit is high when idle. The I2C specification states that during data transfer, the SDA line can change state only when the SCL is low, and when the SCL is high, the SDA is stable and able to be read. Typical I2C write/read transactions are shown in **Figure 6**.



Figure 6. I2C write/read data transfer from host microcontroller.

The read status byte indicates the success or failure of the write transaction. The read status byte must be accessed after each write transaction to the device to ensure that the write transaction process is understood and any errors in the device command handling can be corrected. The read status byte value is summarized in **Table 4**.

Table 4. Read Status Byte Values

|  |  |
| --- | --- |
| READ STATUS BYTE VALUE | DESCRIPTION |
| 0xAA | SUCCESS. The write transaction was successful. |
| 0xAB | PARTIAL\_ACK. Partial page data received, there is still more page data. |
| 0x01 | ERR\_UNAVAIL\_CMD. Illegal Family Byte and/or Command Byte was used. |
| 0x02 | ERR\_UNAVAIL\_FUNC. This function is not implemented. |
| 0x03 | ERR\_DATA\_FORMAT. Incorrect number of bytes sent for the requested Family Byte. |
| 0x04 | ERR\_INPUT\_VALUE. Illegal configuration value was attempted to be set. |
| 0x80 | ERR\_BTLDR\_GENERAL. General error while receiving/flashing a page during the bootloader sequence. |
| 0x81 | ERR\_BTLDR\_CHECKSUM. Checksum error while decrypting/checking page data. |
| 0x82 | ERR\_BTLDR\_AUTH. Authorization error. |
| 0x83 | ERR\_BTLDR\_INVALID\_APP. Application not valid. |
| 0x84 | ERR\_BTLDR\_APP\_NOT\_ERASED. Application was not erased before trying to flash a new one. |
| 0x85 | BL\_ERR\_BTLDR\_DECRYPTION. Error during decryption. Check keys are correctly loaded. |
| 0x86 | BL\_ERR\_KEY\_EXIST: Key is already loaded, cannot load new key |
| 0x87 | BL\_ERR\_NO\_KEY\_MEM: No key slot available |
| 0xFE | ERR\_TRY\_AGAIN. Device is busy. Try again. |
| 0xFF | ERR\_UNKNOWN. Unknown error. |

I2C Write

The process for an I2C write data transfer is as follows:

1. The bus master indicates a data transfer to the device with a START condition.
2. The master transmits 1 byte with the 7-bit slave address and a single write bit set to zero. The 8 bits transferred as a slave address for the MAX78000 are 0xAA for a write transaction.
3. During the next SCL clock that follows the write bit, the master releases SDA. During this clock period, the device responds with an ACK by pulling SDA low.
4. The master senses the ACK condition and begins to transfer the Family Byte. The master drives data on the SDA circuit for each of the 8 bits of the Family Byte and then floats SDA during the ninth bit to allow the device to reply with the ACK indication.
5. The master senses the ACK condition and begins to transfer the Command Index Byte. The master drives data on the SDA circuit for each of the 8 bits of the Command Index Byte and then floats SDA during the ninth bit to allow the device to reply with the ACK indication.
6. The master senses the ACK condition and begins to transfer the Write Data Byte 0. The master drives data on the SDA circuit for each of the 8 bits of the Write Data Byte 0 and then floats SDA during the ninth bit to allow the device to reply with the ACK indication.
7. The master senses the ACK condition and can begin to transfer another Write Data Byte if required. The master drives data on the SDA circuit for each of the 8 bits of the Write Data Byte and then floats SDA during the ninth bit to allow the device to reply with the ACK indication. If another Write Data Byte is not required, the master indicates the transfer is complete by generating a STOP condition. A STOP condition is generated when the master pulls SDA from a low to high while SCL is high.
8. The master waits for a period of CMD\_DELAY (60µs) for the device to have the data ready.
9. The master indicates a data transfer to the slave with a START condition.
10. The master transmits 1 byte with the 7-bit slave address and a single write bit set to one. This is an indication from the master to read the device from the previously written location defined by the Family Byte and the Command Index. The master then floats SDA and allows the device to drive SDA to send the Status Byte. The Status Byte reveals the success of the previous write sequence. After the Status Byte is read, the master drives SDA low to signal the end of data to the device.
11. The master indicates the transfer is complete by generating a STOP condition.
12. After the completion of the write data transfer, the Status Byte must be analyzed to determine if the write sequence was successful and the device has received the intended command.

I2C Read

The process for an I2C read data transfer is as follows:

1. The bus master indicates a data transfer to the device with a START condition.
2. The master transmits 1 byte with the 7-bit slave address and a single write bit set to zero. The 8 bits transferred as a slave address for the are 0xAA for a write transaction. This write transaction precedes the actual read transaction to indicate to the device which section is to be read.
3. During the next SCL clock that follows the write bit, the master releases SDA. During this clock period, the device responds with an ACK by pulling SDA low.
4. The master senses the ACK condition and begins to transfer the Family Byte. The master drives data on the SDA circuit for each of the 8 bits of the Family Byte and then floats SDA during the ninth bit to allow the device to reply with the ACK indication.
5. The master senses the ACK condition and begins to transfer the Command Index Byte. The master drives data on the SDA circuit for each of the 8 bits of the Command Index Byte and then floats SDA during the ninth bit to allow the device to reply with the ACK indication.
6. The master senses the ACK condition and begins to transfer the Write Data Byte if necessary for the read instruction. The master drives data on the SDA circuit for each of the 8 bits of the Write Data Byte and then floats SDA during the ninth bit to allow the device to reply with the ACK indication.
7. The master indicates the transfer is complete by generating a STOP condition.
8. The master waits for a period of CMD\_DELAY (60µs) for the device to have its data ready.
9. The master indicates a data transfer to the slave with a START condition.
10. The master transmits 1 byte with the 7-bit slave address and a single write bit set to 1. This is an indication from the master to read the device from the previously written location defined by the Family Byte and the Command Index. The master then floats SDA and allows the device to drive SDA to send the Status Byte. The Status Byte reveals the success of the previous write sequence. After the Status Byte is read, the master drives SDA low to acknowledge the byte.
11. The master floats SDA and allows the device to drive SDA to send Read Data Byte 0. After Read Data Byte 0 is read, the master drives SDA low to acknowledge the byte.
12. The master floats SDA and allows the device to drive SDA to send the Read Data Byte N. After Read Data Byte N is read, the master drives SDA low to acknowledge the byte. This process continues until the device has provided all the data that the master expects based upon the Family Byte and Command Index Byte definition.
13. The master indicates the transfer is complete by generating a STOP condition.

Communicating with the Bootloader

MAX78000 Bootloader Message Protocol Definitions

There are two protocolsfor command and response for MAX78000 Bootloader. Host command contains Family byte, index byte and optional additional bytes. The response protocol has status information and optional additional bytes, these are shown in **Figure 7.** MAX78000 Bootloader Communication Protocol.

Figure 7. MAX78000 Bootloader Communication Protocol.

**Table 4.** Read Status Byte Values lists the available status response bytes and **Table 5** lists the MAX78000 bootloader message protocol definitions. Note that status bytes are not included in the table for each response, firstly status byte then additional response bytes are sent by MAX78000 Bootloader. DEFAULT\_DELAY for MAX78000 Bootloader responses are 60µs.

Table 5. MAX78000 Bootloader Message Protocol Definitions

| HOST COMMAND | | | | | MAX78000 BOOTLOADER | Min Delay Required |
| --- | --- | --- | --- | --- | --- | --- |
| FAMILY NAME | DESCRIPTION | FAMILY BYTE | INDEX BYTE | WRITE BYTES | ADDITIONAL RESPONSE BYTES | Time |
| Device Mode | Select the device operating mode. | 0x01 | 0x00 | **0x00**: Exit bootloader mode.  **0x02**: Reset. (The application must implement this.)  **0x08:** Enter bootloader mode. (The application must implement this. See section Host Serial Command using Power-On or Hard Reset.) | — | DEFAULT\_DELAY |
| Device Mode | Read the device operating mode. | 0x02 | 0x00 | (The application must implement this.) | **0x00**: Application operating mode.  **0x08**: Bootloader operating mode. | DEFAULT\_DELAY |
| Bootloader Flash | Set the initialization vector bytes.  This is not required for a non-secure bootloader. | 0x80 | 0x00 | Use the 11 bytes 0x28 to 0x32 from the .msbl file. | — | DEFAULT\_DELAY |
| Bootloader Flash | Set the authentication bytes.  This is not required for a non-secure bootloader. | 0x80 | 0x01 | Use the 16 bytes 0x34 to 0x43 from the .msbl file. | — | DEFAULT\_DELAY |
| Bootloader Flash | Set the number of pages. | 0x80 | 0x02 | **0x00:** Number of pages specified by byte 0x44 from the .msbl file. (Total of 2 bytes) | — | DEFAULT\_DELAY |
| Bootloader Flash | Erase the application flash memory. | 0x80 | 0x03 | — | — | 700ms |
| Bootloader Flash | Send the page values. | 0x80 | 0x04 | The first page is specified by byte 0x4C from the .msbl file. The total bytes for each message protocol is the page size + 16 bytes (consisting of the page CRC32 and 12 dummy bytes). | — | If waiting for final page and crc check enabled 550ms else 200ms |
| Bootloader Flash | Erase Page Memory  (Valid for bootloaders after version 3.4.3) | 0x80 | 0x05 | **0x00:** Number of page to be erased. (Total of 2 bytes) | — | 40ms |
| Bootloader Flash | Partial Page Load Length | 0x80 | 0x06 | 2 bytes MSB first (Between 1 and 8208) | — | DEFAULT\_DELAY |
| Bootloader Flash | Key Load | 0x80 | 0x07 | 1 byte Valid AES Key Length, 32 byte Key Data padded with 0x00 if necessary, 1 byte Valid AAD Length, 32 bytes AAD Data padded with 0x00 if necessary | — | 200ms |
| Bootloader Information | Get the bootloader version. | 0x81 | 0x00 | — | Major version byte, Minor version byte, Revision byte | DEFAULT\_DELAY |
| Bootloader Information | Get the page size in bytes. | 0x81 | 0x01 | — | Upper byte of page size, Lower byte of page size | DEFAULT\_DELAY |
| Bootloader Information | Get unique serial number (USN). | 0x81 | 0x02 | — | 24-byte USN | DEFAULT\_DELAY |
| Bootloader Configuration | Save bootloader configurations.  Write this command after changes are made to any of the Bootloader Configuration settings. The bootloader should be restarted for the new configuration to be active. | 0x82 | 0x00 | — | — | 1000ms if SWD is unlocked from locked state, otherwise 300 ms |
| Bootloader Configuration | Select enter\_bl\_check.  Configure the device to check the state of the EBL GPIO pin to decide whether to enter bootloader mode. | 0x82 | 0x01 | **0x00, 0x00:** The device does not check the state of the EBL GPIO pin. (Default)  **0x00, 0x01:** The device checks the state of the EBL GPIO pin before entering bootloader mode. | — | DEFAULT\_DELAY |
| Bootloader Configuration | Select the EBL GPIO pin (ebl\_pin).  Select which pin to use as the EBL GPIO pin. This command is only used if the Bootloader Configuration enter bootloader check is set to 1 (0x82 0x01 0x00 0x01). | 0x82 | 0x01 | **0x01,0x00–0x0B:** Acceptable pin range | — | DEFAULT\_DELAY |
| Bootloader Configuration | Select the active state for the EBL GPIO pin (ebl\_pol). This command is only used if the Bootloader Configuration enter bootloader check is set to 1 (0x82 0x01 0x00 0x01). | 0x82 | 0x01 | **0x02, 0x00:** Active-low. The device enters bootloader mode if the EBL GPIO pin is held low during power-on or during a RSTN device pin cycle. (Default)  **0x02, 0x01:** Active-high. The device enters bootloader mode if the EBL GPIO pin is held high during power-on or during a RSTN device pin cycle. | — | DEFAULT\_DELAY |
| Bootloader Configuration | Enable or Disable Valid Mark Check (valid\_mark\_check) | 0x82 | 0x01 | **0x03, 0x00:** Disable Valid Mark Check.  **0x03, 0x01:** Enable Valid Mark Check (Default). | — | DEFAULT\_DELAY |
| Bootloader Configuration | Enable or Disable I2C Interface (i2c\_enable) | 0x82 | 0x01 | **0x05, 0x00:** Disable I2C interface.  **0x05, 0x01:** Enable I2C interface (Default). | — | DEFAULT\_DELAY |
| Bootloader Configuration | I2C Slave Address (i2c\_addr) | 0x82 | 0x01 | **0x07, 0x08–0x77:** Acceptable range I2C Slave Address. | — | DEFAULT\_DELAY |
| Bootloader Configuration | Enable or Disable CRC Check before jumping application (crc\_check) | 0x82 | 0x01 | **0x08, 0x00:** Disable CRC Check (Default).  **0x08, 0x01:** Enable CRC Check. (Only Default for v3.4.3) | — | DEFAULT\_DELAY |
| Bootloader Configuration | Enable SWD Lock (swd\_lock) | 0x82 | 0x01 | **0x09, 0x00:** SWD is not locked (Default, cmd has no effect).  **0x09, 0x01:** Enable SWD lock (irreversible). | — | DEFAULT\_DELAY |
| Bootloader Configuration | Exit bootloader mode (exit\_bl\_mode).  Determine how the bootloader enters application mode. Note that: if crc check is enabled, it may take up to 350 ms to check app crc. Until the crc check is completed, this command will return invalid app response. | 0x82 | 0x02 | **0x00, 0x00:** Enter application mode if an application is present and valid. If EBL GPIO pin was used to enter bootloader mode, the jump does not occur until the EBL GPIO pin is in a non-active state (Default).  **0x00, 0x01:** Wait for a programmable delay. If no commands are received and a valid application is present, enter application mode.  **0x00, 0x02:** Stay in bootloader mode. | — | DEFAULT\_DELAY |
| Bootloader Configuration | Configure timeout exit (ebl\_timeout).  Set the length of the additional programmable timeout to use when the Bootloader Configuration exit bootloader mode is set to 1 (AA 82 02 01).  The system requires a 20ms non-programmable delay to switch to application mode. | 0x82 | 0x02 | **0x01, 0x00–0x0F:** Timeout  Note: Timeout is cancelled if any commands are received during this period. | — | DEFAULT\_DELAY |
| Bootloader Configuration | Read bootloader check configuration (enter\_bl\_check).  Read the device configuration to check the state of the EBL GPIO pin to decide whether to enter bootloader mode. | 0x83 | 0x01 | 0x00 | **0x00:** The device does not check the state of the EBL GPIO pin.  **0x01:** The device checks the state of the EBL GPIO pin before entering bootloader mode. | DEFAULT\_DELAY |
| Bootloader Configuration | Read the EBL GPIO pin (ebl\_pin).  Read which pin is used as the EBL GPIO pin. This command is only used if the Bootloader Configuration enter bootloader check is set to 1 (AA 82 01 00 01). | 0x83 | 0x01 | 0x01 | **0x00–0x09:** Expected range for the 16-bump WLP package.  **0x00–0x0B:** Expected range for the 20-pin TQFN-EP and the 24-pin TQFN-EP | DEFAULT\_DELAY |
| Bootloader Configuration | Read the active state for the EBL GPIO pin (ebl\_pol). | 0x83 | 0x01 | 0x02 | **0x00:** Active-low. The device enters bootloader mode if the EBL GPIO pin is held low during power-on or during a RSTN device pin cycle.  **0x01:** Active-high. The device enters bootloader mode if the EBL GPIO pin is held high during power-on or during a RSTN device pin cycle. | DEFAULT\_DELAY |
| Bootloader Configuration | Read the Valid Mark Check (valid\_mark\_check) | 0x83 | 0x01 | 0x03 | **0x00:** Valid Mark Check is disabled.  **0x01:** Valid Mark Check is enabled. | DEFAULT\_DELAY |
| Bootloader Configuration | Read the I2C interface enable status (i2c\_enable). | 0x83 | 0x01 | 0x05 | **0x00:** I2C interface is disabled.  **0x01:** I2C interface is enabled. | DEFAULT\_DELAY |
| Bootloader Configuration | Read I2C Slave Address (i2c\_addr). | 0x83 | 0x01 | 0x07 | **0x08-0x77:** 7-bit I2C Address | DEFAULT\_DELAY |
| Bootloader Configuration | Read CRC Check status (crc\_check). | 0x83 | 0x01 | 0x08 | **0x00:** CRC Check is disabled.  **0x01:** CRC Check is enabled. | DEFAULT\_DELAY |
| Bootloader Configuration | Read exit bootloader mode configuration.  Read how the bootloader enters application mode. | 0x83 | 0x02 | 0x00 | **0x00:** If an application is present and valid, enter application mode. If the EBL GPIO pin was used to enter bootloader mode, the jump does not occur until the EBL GPIO pin is in a non-active state. (Default)  **0x01:** Wait for a programmable delay. If no commands are received and a valid application is present, enter application mode.  **0x02:** Stay in bootloader mode. | DEFAULT\_DELAY |
| Bootloader Configuration | Read exit timeout configuration (ebl\_timeout).  Read the timeout to use when the Bootloader Configuration exit bootloader mode is set to 1 (AA 82 02 01).  Timeout is cancelled if any commands are received during this period. | 0x83 | 0x02 | 0x01 | **0x00–0x0F:** Timeout | DEFAULT\_DELAY |
| Bootloader Configuration | Read bootloader configuration parameters. | 0x83 | 0xFF | 0x00 | **Byte7, Byte6, Byte 5, Byte4, Byte3, Byte2, Byte1, Byte0:** | DEFAULT\_DELAY |
| **Byte0.bit0:** enter\_bl\_check  **Byte0.bit1-bit4:** ebl\_pin  **Byte0.bit5:** ebl\_polarity  **Byte0.bit6-bit7:** RFU |
| **Byte1.bit0:** uart\_enable  **Byte1.bit1:** i2c\_enable  **Byte1.bit2:** spi\_enable  **Byte1.bit3-bit7:** RFU |
| **Byte2.bit0-bit3:** ebl\_timeout  **Byte2.bit4-bit5:** exit\_bl\_mode  **Byte2.bit6-bit7:** RFU |
| **Byte3.bit0:** crc\_check  **Byte3.bit1:** valid\_mark\_check  **Byte3.bit2-bit7:** RFU |
| **Byte4.bit0-bit6:** i2c\_addr  **Byte4.bit7:** RFU |
| **Byte5.bit0-bit7:** RFU |
| **Byte6.bit0-bit7:** RFU |
| **Byte7.bit0-bit7:** RFU |
| Identity | Read the MCU type. | 0xFF | 0x00 | — | **0x00:** MAX32625  **0x01:** MAX32660/MAX32664  **0x05:** MAX78000 | DEFAULT\_DELAY |

MAX78000 In-Application Programming, Annotated Trace

The MAX78000 bootloader firmware supports IAP.

This section shows the necessary commands to flash the application to MAX78000. Each   
8192-byte page data is appended with 4-byte CRC32 of the page and 12 bytes of 0x00, therefore payload of the bootloader flash page message is 8208 bytes for each page. The number of pages can be found by computing:



Necessary commands to flash an application image of 25922 (0x6542) bytes are shown in the following example, where the number of pages is calculated as:



**Table 6** shows how to download the application by using the .msbl file. See Appendix A for more details about the .msbl file.

Table 6. Application Programming Example by Using the .msbl File

| HOST COMMAND | COMMAND DESCRIPTION | MAX78000 BOOTLOADER RESPONSE | RESPONSE DESCRIPTION |
| --- | --- | --- | --- |
| 0x01 0x00 0x08\* | Set mode to 0x08 for bootloader mode.  Note that this is one of the alternative methods for entering bootloader mode. If this command is used, the EBL pin is not necessary See the Entering Bootloader Mode from the Application Mode section for alternative ways to enter bootloader mode. | 0xAA | No error. |
| 0x02 0x00\* | Read mode. | 0xAA 0x08 | No error. Mode is bootloader. |
| 0xFF 0x00+ | Get ID and MCU type. | 0xAA 0x05 | No error. MCU is MAX78000. |
| 0x81 0x00 | Read bootloader firmware version. | 0xAA 0xXX 0xXX 0xXX | No error. Version is XX.XX.XX |
| 0x81 0x01 | Read bootloader page size. | 0xAA 0x20 0x00 | No error. Page size is 8192. |
| 0x80 0x02 0x00 0x05\* | Bootloader flash. Set the number of pages to 5 based on byte 0x44 from the application .msbl file, which is created from the user application .bin file. | 0xAA | No error. |
|  | | |
| 0x80 0x03\* | Bootloader flash. Erase application. | 0xAA | No error. |
| 0x80 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 | Bootloader flash. Set Initialization Vector based on the bytes 0x28 to 0x32 from the application .msbl file, which is created from the user application .bin file. | 0xAA | No error. |
|  | | |
| 0x80 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00  0x00 0x00 0x00 0x00 0x00 0x00 | Bootloader flash. Set Authentication based on the bytes 0x34 to 0x43 from the application .msbl file, which is created from the user application .bin file. | 0xAA | No error. |
|  | | |
| 0x80 0x04 0x00 0x80 0x01 ... 0x00 0x00 0x00\* | Bootloader flash. Send first page bytes 0x4C to 0x205B from the .msbl file. | 0xAA | No error. |
| … | | |
| 0x80 0x04 0x01 0x21 0x00 ... 0x00 0x00 0x00\* | Bootloader flash. Send second page bytes 0x205C to 0x406B from the .msbl file. | 0xAA | No error. |
| 0x80 0x04 0x02 0x02 0xC1 ... 0x00 0x00 0x00\* | Bootloader flash. Send third page bytes 0x406C to 0607B from the .msbl file. | 0xAA | No error. |
| 0x80 0x04 0xEO 0x6C 0x1C ... 0x00 0x00 0x00\* | Bootloader flash. Send fourth page bytes 0x607C to 0x808B from the .msbl file. | 0xAA | No error. |
| 0x80 0x04 0xFF 0xC3 0x0D ... 0x00 0x00 0x00\* | Bootloader flash. Send fifth page bytes 0x808C to 0XA09B from the .msbl file. | 0xAA | No error. |
| 0x01 0x00 0x00\* | Exit bootloader mode and jump to application.  Note: Sending the bootloader command is not mandatory. The microcontroller can be reset and the EBL pin can be set to reverse the polarity to jump on the application. | 0xAA | No error. |

*\*Mandatory*

*+Recommended*

**Table 7** shows how to download the application by using the .msbl file with the partial page download feature. By using this method, each flash page payload (8208-byte) data can be sent as different length packet sizes varying between 1 byte and 8208 bytes. In this example, the partial page data load size is selected as 4000.

Table 7. Application Programming Example by Using Partial Pages

| HOST COMMAND | COMMAND DESCRIPTION | MAX78000 BOOTLOADER RESPONSE | RESPONSE  DESCRIPTION |
| --- | --- | --- | --- |
| 0x01 0x00 0x08\* | Set mode to 0x08 for bootloader mode.  Note that this is one of the alternative methods for entering bootloader mode. If this command is used, the EBL pin is not necessary. See the Entering Bootloader Mode from the Application Mode section for alternative ways to enter bootloader mode. | 0xAA | No error. |
| 0x02 0x00\* | Read mode. | 0xAA 0x08 | No error. Mode is bootloader. |
| 0xFF 0x00+ | Get ID and MCU type. | 0xAA 0x05 | No error. MCU is MAX78000. |
| 0x81 0x00 | Read bootloader firmware version. | 0xAA 0xXX 0xXX 0xXX | No error. Version is XX.XX.XX. |
| 0x81 0x01 | Read bootloader page size. | 0xAA 0x20 0x00 | No error. Page size is 8192. |
| 0x80 0x02 0x00 0x05\* | Bootloader flash. Set the number of pages to 5 based on byte 0x44 from the application .msbl file, which is created from the user application .bin file. | 0xAA | No error. |
|  | | |
| 0x80 0x06 0x0F 0xA0\* | Set partial page load size as 4000 (0x0FA0) | 0xAA | No error. |
| 0x80 0x03\* | Bootloader flash. Erase application. | 0xAA | No error. |
| 0x80 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 | Bootloader flash. Set Initialization Vector based on the bytes 0x28 to 0x32 from the application .msbl file, which is created from the user application .bin file. | 0xAA | No error. |
|  | | |
| 0x80 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00  0x00 0x00 0x00 0x00 0x00 0x00 | Bootloader flash. Set Authentication based on the bytes 0x34 to 0x43 from the application .msbl file, which is created from the user application .bin file. | 0xAA | No error. |
|  | | |
| 0x80 0x04 0x00 0x80 0x01 ... 0x00 0x00 0x00\* | Bootloader flash. Send the first 4000 bytes of first page 0x4C to 0xFEB from the .msbl file. | 0xAB (Wait for remaining page data) | No error. |
| 0x80 0x04 0x12 0x34 0x56 ... 0x00 0x00 0x00\* | Bootloader flash. Send the second 4000 bytes of the first page 0xFEC to 0x1F8B from the .msbl file. | 0xAB (Wait for remaining page data) | No error. |
| 0x80 0x04 0x98 0x67 0x00 ... 0x00 0x00 0x00\* | Bootloader flash. Send the last 208 bytes of the first page 0x1F8C to 0x205B from the .msbl file. | 0xAA | No error. |
| 0x80 0x04 0x01 0x21 0x00 ... 0x00 0x00 0x00\* | Bootloader flash. Send the first 4000 bytes of second page 0x205C to 0x2FFB from the .msbl file. | 0xAB (Wait for remaining page data) | No error. |
| 0x80 0x04 0x12 0x34 0x56 ... 0x00 0x00 0x00\* | Bootloader flash. Send the second 4000 bytes of the second page 0x2FFC to 0x3F9B from the .msbl file. | 00xAB (Wait for remaining page data) | No error. |
| 0x80 0x04 0x98 0x67 0x00 ... 0x00 0x00 0x00\* | Bootloader flash. Send the last 208 bytes of the second page 0x3F9C to 0x406B from the .msbl file. | 0xAA | No error. |
| 0x80 0x04 0x02 0x02 0xC1 ... 0x00 0x00 0x00\* | Bootloader flash. Send the first 4000 bytes of third page 0x406C to 0x500B from the .msbl file. | 0xAB (Wait for remaining page data) | No error. |
| 0x80 0x04 0x12 0x34 0x56 ... 0x00 0x00 0x00\* | Bootloader flash. Send the second 4000 bytes of the third page 0x500C to 0x5FAB from the .msbl file. | 0xAB (Wait for remaining page data) | No error. |
| 0x80 0x04 0x90 0x77 0x00 ... 0x00 0x00 0x00\* | Bootloader flash. Send the last 208 bytes of the third page 0x5FAC to 0x607B from the .msbl file. | 0xAA | No error. |
| 0x80 0x04 0xEO 0x6C 0x1C ... 0x00 0x00 0x00\* | Bootloader flash. Send the first 4000 bytes of fourth page 0x607C to | 0xAB (Wait for remaining page data) | No error. |
| 0x80 0x04 0x00 0x00 0x00 ... 0x00 0x00 0x00\* | Bootloader flash. Send the second 4000 bytes of the fourth page 0x701C to 0x7FBB from the .msbl file. | 0xAB (Wait for remaining page data) | No error. |
| 0x80 0x04 0x00 0x00 0x00 ... 0x00 0x00 0x00\* | Bootloader flash. Send the last 208 bytes of the fourth page 0x7FBC to 0x808B from the .msbl file. | 0xAA | No error. |
| 0x80 0x04 0xFF 0xC3 0x0D ... 0x00 0x00 0x00\* | Bootloader flash. Send the first 4000 bytes of the last page 0x808C to | 0xAB (Wait for remaining page data) | No error. |
| 0x80 0x04 0x00 0x00 0x00 ... 0x00 0x00 0x00\* | Bootloader flash. Send the second 4000 bytes of the last page 0x902C to 0x9FCB from the .msbl file. | 0xAB (Wait for remaining page data) | No error. |
| 0x80 0x04 0x00 0x00 0x00 ... 0x00 0x00 0x00\* | Bootloader flash. Send the last 208 bytes of the last page 0x9FCC to 0XA09B from the .msbl file. | 0xAA | No error. |
| 0x01 0x00 0x00\* | Exit bootloader mode and jump to application.  Note: Sending the bootloader command is not mandatory. The microcontroller can be reset and the EBL pin can be set to reverse the polarity to jump on the application. | 0xAA | No error. |

*\*Mandatory*

*+Recommended*

Appendix A: Maxim Special Bootloader (.msbl) File Format

The .msbl file is a special binary file format that is generated from the application update .bin file by using the MAX78000 bootloader. The .msbl file has the following sections:

* Header

The header consists of the following:

* + 4-byte magic value (.msbl)
  + 4-byte RFU
  + 16-byte target type (e.g., MAX78000)
  + 16-byte Encryption Algorithm (e.g., AES-192)
  + 11-byte Initialization Vector
  + 1-byte RFU
  + 16-byte Authentication Data
  + 2-byte number of pages (LSB first) (e.g., 0x05 0x00 means there are five pages)
  + Number of pages =  (for sending application information)
  + 2-byte page size (LSB first) (e.g., 0x00 0x20 means the page size is 8192)
  + 1-byte CRC byte size (0x04 means 4 bytes and denotes CRC32)
  + 3-byte RFU

**Figure 8** shows an example of the format of the raw hex header data in the .msbl file.

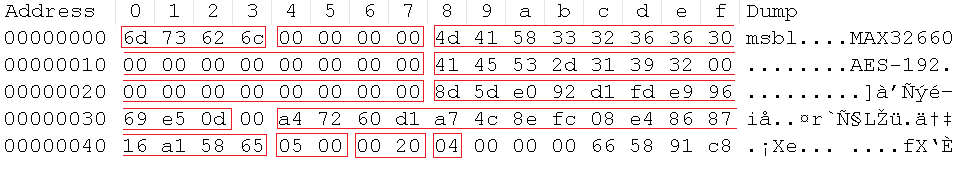


Figure 8. Hex header data in the .msbl.

* (Number of pages – 1) × Page Data:
  + Page Data 1: First 8192-byte data from the .bin file + 4-byte CRC32 of the first   
    8192-byte data + 12-byte dummy data (0x00)
  + Page Data 2: Second 8192-byte data from the .bin file + 4-byte CRC32 of the first 8192-byte data + 12-byte dummy data (0x00)
  + Page Data 3: Third 8192-byte data from the .bin file + 4-byte CRC32 of the first   
    8192-byte data + 12-byte dummy data (0x00)

…

* + Page Data (Number of pages – 1): (Number of pages – 1)th 8192-byte data from the .bin file + 4-byte CRC32 of the first 8192-byte data + 12-byte dummy data (0x00)
* Application information: 4-byte CRC32 of the application + 4-byte length of the application + 8184-byte dummy data (0x00) + 4-byte CRC32 + 12-byte dummy data (0x00)
* 4-byte CRC32 of the .msbl file which is the CRC32 value of the total .msbl file

**Table 8** shows the .msbl file format for an application with a size of 17384 bytes.

Table 8. Example .msbl File Format

|  |  |  |  |
| --- | --- | --- | --- |
| ADDRESS | LENGTH (bytes) | NAME | DESCRIPTION |
| 0x0000 | 4 | Magic (.msbl) | A marker that indicates the beginning of the .msbl file |
| 0x0004 | 4 | RFU | Reserved for future use (Fill 0x00) |
| 0x0008 | 16 | Target Type | Target microcontroller. For example, MAX78000 with zeros appended. |
| 0x0018 | 16 | Encryption Algorithm | Defines used Encryption Algorithm (e.g. AES-192) |
| 0x0028 | 11 | Initialization Vector | Initialization Vector used for encryption algorithm |
| 0x0033 | 1 | RFU | Reserved for future use (Fill 0x00) |
| 0x0034 | 16 | Authentication Data | Authentication Data of the all image |
| 0x0044 | 2 | Number of pages | Number of pages (0x04 for this application) |
| 0x0046 | 2 | Page size | Number of bytes per page. Always 0x2000 (8192 as a decimal). |
| 0x0048 | 1 | CRC byte size | 0x04 bytes denoting CRC32 |
| 0x0049 | 3 | RFU | Reserved for future use (Fill 0x00) |
| 0x004C | 8192 | First 8192 bytes of the .bin file | The first page of application data |
| 0x204C | 4 | CRC32 of the first page | Calculated CRC32 value for the first page of application data |
| 0x2050 | 12 | RFU | Reserved for future use (Fill 0x00) |
| 0x205C | 8192 | Second 8192 bytes of the .bin file | Second page of application data |
| 0x405C | 4 | CRC32 of the second page | Calculated CRC32 value for the second page of application data appended with 0x00 |
| 0x4060 | 12 | RFU | Reserved for future use (Fill 0x00) |
| 0x406C | 8192 | Last 1000 bytes of the .bin file appended with 7192 bytes of 0x00 | The last page of application data |
| 0x606C | 4 | CRC32 of the last page | Calculated CRC32 value for the last page of application data |
| 0x6070 | 12 | RFU | Reserved for future use (Fill 0x00) |
| 0x607C | 4 | CRC32 of complete .bin file | CRC32 of application |
| 0x6080 | 4 | Length of .bin file | Length of .bin file (0xE8, 0x43, 0x00, 0x00) (17384 as decimal) |
| 0x6084 | 8184 | RFU | Reserved for future use (Fill 0x00) |
| 0x807C | 4 | CRC32 of application data | Calculated CRC32 value of 8192 bytes starting from 0x607C |
| 0x8080 | 12 | RFU | Reserved for future use (Fill 0x00) |
| 0x808C | 4 | CRC32 of .msbl file | CRC32 of all data up to this point in the .msbl file |

*MinGW is a registered trademark of Software in the Public Interest*, *Inc.*

Appendix C: How to Program the Application by Using a Binary File

The MAX78000 bootloader firmware supports IAP.

This appendix shows the necessary commands to flash the application to the MAX78000 by using the application binary image (.bin file). Each 8192-byte page data is appended with the 4-byte CRC32 of the page and 12 bytes of 0x00. Therefore, the payload of the bootloader flash page message is 8208 bytes for each page. The number of pages is calculated using the following equation:



This example provides the necessary commands to flash an application image of 25922 (0x6542) bytes and the number of pages is calculated as:



**Table 9** shows how to download an application by using the application binary image.

Table 9. Binary File Application Programming Example

| HOST COMMAND | COMMAND DESCRIPTION | MAX78000 BOOTLOADER RESPONSE | RESPONSE  DESCRIPTION |
| --- | --- | --- | --- |
| 0x01 0x00 0x08\* | Set mode to 0x08 for bootloader mode.  Note that this is one of the alternative methods for entering bootloader mode. If this command is used, the EBL pin is not necessary. See the Entering Bootloader Mode from the Application Mode section for alternative ways to enter bootloader mode. | 0xAA | No error. |
| 0x02 0x00\* | Read mode. | 0xAA 0x08 | No error. Mode is bootloader. |
| 0xFF 0x00+ | Get ID and MCU type. | 0xAA 0x05 | No error. MCU is MAX78000. |
| 0x81 0x00 | Read bootloader firmware version. | 0xAA 0xXX 0xXX 0xXX | No error. Version is XX.XX.XX. |
| 0x81 0x01 | Read bootloader page size. | 0xAA 0x20 0x00 | No error. Page size is 8192. |
| 0x80 0x02 0x00 0x05\* | Bootloader flash. Set the number of pages to 5 based on the whole number of complete and partial pages in the user application .bin file. | 0xAA | No error. |
| 0x80 0x03\* | Bootloader flash. Erase application. | 0xAA | No error. |
| 0x80 0x04 0x00 0x80 0x01 ... 0x00 0x00 0x00\* | Bootloader flash. Send the first 8K page from the user binary (0x0000 through 0x1FFF), followed by the computed CRC32 and then the 12 bytes of zero. See the *crc32.c* file in the example for code to compute this value. | 0xAA | No error. |
| 0x80 0x04 0x01 0x21 0x00 ... 0x00 0x00 0x00\* | Bootloader flash. Send the second 8K page from the user binary (0x2000 through 0x3FFF), followed by the computed CRC32, then the 12 bytes of zero. | 0xAA | No error. |
| 0x80 0x04 0x02 0x02 0xC1 ... 0x00 0x00 0x00\* | Bootloader flash. Send the third 8K page from the user binary (0x4000 through 0x5FFF), followed by the computed CRC32, and then the 12 bytes of zero. | 0xAA | No error. |
| 0x80 0x04 0xFF 0xC3 0x0D ... 0x00 0x00 0x00\* | Bootloader flash. Send the last 1346 bytes of user binary (0x6000 through 0x6541) appended with 6846 bytes of 0x00, followed by the computed CRC32 of the sent 8192 bytes, and then the 12 bytes of zero. | 0xAA | No error. |
| 0x80 0x04 0xE3 0x2D 0x1F ... 0x00 0x00 0x00\* | Bootloader flash. Send 4-byte CRC32 of bin image (LSB first), appended with 4-byte length of bin image (LSB first 0x42, 0x65, 0x00, 0x00), appended with 8184 bytes of 0x00, followed by the computed CRC32 of sent 8192 bytes, and the 12 bytes of zero. | 0xAA | No error. |
| 0x01 0x00 0x00\* | Exit bootloader mode and jump to application.  Note: Sending the bootloader command is not mandatory. The microcontroller can be reset and the EBL pin can be set to reverse the polarity to jump on the application. | 0xAA | No error. |

*\*Mandatory*

*+Recommended*

Some host devices might not support sending the flash page payload as a single chunk. **Table 10** shows how to download the application by using the partial page download feature. Each flash page payload (8208-byte) data can be sent by different length packet sizes varying between   
1 byte and 8208 bytes. In this example, the partial page data load size is selected as 4000.

Table 10. Binary File Partial Application Programming Example

| HOST COMMAND | COMMAND DESCRIPTION | MAX78000 BOOTLOADER RESPONSE | RESPONSE  DESCRIPTION |
| --- | --- | --- | --- |
| 0x01 0x00 0x08\* | Set mode to 0x08 for bootloader mode.  Note that this is one of the alternative methods for entering bootloader mode. If this command is used, the EBL pin is not necessary See the Entering Bootloader Mode from the Application Mode section for alternative ways to enter bootloader mode. | 0xAA | No error. |
| 0x02 0x00\* | Read mode. | 0xAA 0x08 | No error. Mode is bootloader. |
| 0xFF 0x00+ | Get ID and MCU type. | 0xAA 0x05 | No error. MCU is MAX78000. |
| 0x81 0x00+ | Read bootloader firmware version. | 0xAA 0xXX 0xXX 0xXX | No error. Version is XX.XX.XX. |
| 0x81 0x01 | Read bootloader page size. | 0xAA 0x20 0x00 | No error. Page size is 8192. |
| 0x80 0x02 0x00 0x05\* | Bootloader flash. Set the number of pages to 5 based on the whole number of complete and partial pages in the user application .bin file. | 0xAA | No error. |
| 0x80 0x06 0x0F 0xA0\* | Set partial page load size as 4000 (0x0FA0). | 0xAA | No error. |
| 0x80 0x03\* | Bootloader flash. Erase application. | 0xAA | No error. |
| 0x80 0x04 0x00 0x80 0x01 ... 0x00 0x00 0x00\* | Bootloader flash. Send the first 4000 bytes of the first page from the user binary (0x0000 through 0x0F9F). | 0xAB (Wait for remaining page data) | No error. |
| 0x80 0x04 0x12 0x34 0x56 ... 0x00 0x00 0x00\* | Bootloader flash. Send the second 4000 bytes of the first page from the user binary (0x0FA0 through 0x1F3F). | 0xAB (Wait for remaining page data) | No error. |
| 0x80 0x04 0x98 0x67 0x00 ... 0x00 0x00 0x00\* | Bootloader flash. Send the last 192 bytes of the first page from the user binary (0x1F40 through 0x1FFF), followed by the computed CRC32 of the first page (8192 bytes), and the 12 bytes of zero. | 0xAA | No error. |
| 0x80 0x04 0x01 0x21 0x00 ... 0x00 0x00 0x00\* | Bootloader flash. Send the first 4000 bytes of the second page from the user binary (0x2000 through 0x2F9F). | 0xAB (Wait for remaining page data) | No error. |
| 0x80 0x04 0x12 0x34 0x56 ... 0x00 0x00 0x00\* | Bootloader flash. Send the second 4000 bytes of the second page from the user binary (0x2FA0 through 0x3F3F). | 0xAB (Wait for remaining page data) | No error. |
| 0x80 0x04 0x98 0x67 0x00 ... 0x00 0x00 0x00\* | Bootloader flash. Send the last 192 bytes of the second page from the user binary (0x3F40 through 0x3FFF), followed by the computed CRC32 of the second page (8192 bytes), and the 12 bytes of zero. | 0xAA | No error. |
| 0x80 0x04 0x02 0x02 0xC1 ... 0x00 0x00 0x00\* | Bootloader flash. Send the first 4000 bytes of the third page from the user binary (0x4000 through 0x4F9F). | 0xAB (Wait for remaining page data) | No error. |
| 0x80 0x04 0x12 0x34 0x56 ... 0x00 0x00 0x00\* | Bootloader flash. Send the second 4000 bytes of the third page from the user binary (0x4FA0 through 0x5F3F). | 0xAB (Wait for remaining page data) | No error. |
| 0x80 0x04 0x90 0x77 0x00 ... 0x00 0x00 0x00\* | Bootloader flash. Send the last 192 bytes of the third page from the user binary (0x5F40 through 0x5FFF), followed by the computed CRC32 of the third page (8192 bytes), and the 12 bytes of zero. | 0xAA | No error. |
| 0x80 0x04 0xFF 0xC3 0x0D ... 0x00 0x00 0x00\* | Bootloader flash. Send the last 1346 bytes of user binary (0x6000 through 0x6541) appended with 2654 bytes of 0x00. | 0xAB (Wait for remaining page data) | No error. |
| 0x80 0x04 0x00 0x00 0x00 ... 0x00 0x00 0x00\* | Bootloader flash. Send 4000 bytes of 0x00. | 0xAB (Wait for remaining page data) | No error. |
| 0x80 0x04 0x00 0x00 0x00 ... 0x00 0x00 0x00\* | Bootloader flash. Send 192 bytes of 0x00, followed by the computed CRC32 of the last 1346 bytes of user binary and appended 6846 of 0x00 (total of 8192 bytes), followed by 12 bytes of zero. | 0xAA | No error. |
| 0x80 0x04 0xE3 0x2D 0x1F ... 0x00 0x00 0x00\* | Bootloader flash. Send the 4-byte CRC32 of the .bin image (LSB first), appended with the 4-byte length of the .bin image (LSB first, 0x42, 0x65, 0x00, 0x00), appended with 3992 bytes of 0x00. | 0xAB (Wait for remaining page data) | No error. |
| 0x80 0x04 0x00 0x00 0x00 ... 0x00 0x00 0x00\* | Bootloader flash. Send 4000 bytes of 0x00. | 0xAB (Wait for remaining page data) | No error. |
| 0x80 0x04 0x00 0x00 0x00 ... 0x00 0x00 0x00\* | Bootloader flash. Send 192 bytes of 0x00, followed by the computed CRC32 of the last sent 8192 bytes (4-byte CRC32 of the .bin image, appended with the 4-byte length of the .bin image and 8184 bytes of 0x00) followed by 12 bytes of zero. | 0xAA | No error. |
| 0x01 0x00 0x00\* | Exit bootloader mode and jump to application.  Note: Sending the bootloader command is not mandatory. The microcontroller can be reset and the EBL pin can be set to reverse the polarity to jump on the application. | 0xAA | No error. |

*\*Mandatory*

*+Recommended*

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| REVISION NUMBER | REVISION DATE | DESCRIPTION | PAGES CHANGED |
| 0 | 03/21 | Initial release | — |
| 1 | 10/28 | Improve for readability | — |

©2020 by Maxim Integrated Products, Inc. All rights reserved. Information in this publication concerning the devices, applications, or technology described is intended to suggest possible uses and may be superseded. MAXIM INTEGRATED PRODUCTS, INC. DOES NOT ASSUME LIABILITY FOR OR PROVIDE A REPRESENTATION OF ACCURACY OF THE INFORMATION, DEVICES, OR TECHNOLOGY DESCRIBED IN THIS DOCUMENT. MAXIM ALSO DOES NOT ASSUME LIABILITY FOR INTELLECTUAL PROPERTY INFRINGEMENT RELATED IN ANY MANNER TO USE OF INFORMATION, DEVICES, OR TECHNOLOGY DESCRIBED HEREIN OR OTHERWISE. The information contained within this document has been verified according to the general principles of electrical and mechanical engineering or registered trademarks of Maxim Integrated Products, Inc. All other product or service names are the property of their respective owners.