MAX32660 Bootloader User Guide

UG6471; Rev 0; 10/18

Abstract

The MAX32660 bootloader user guide provides flow charts, timing diagrams, GPIOs/pin usage, I2C interface protocol, and an annotated I2C trace between the host microcontroller and the MAX32660 for in-application programming (IAP). Typical application uses the MAX32660 bootloader as a low-power microcontroller in a sensor hub configuration to process and collect data while the SoC is in sleep mode.

Table of Contents

List of Figures

List of Tables

Overview

The MAX32660 bootloader is an embedded firmware that provides the MAX32660 with the ability to have its application code updated by a host microcontroller. The bootloader is accessed through the I2C interface. The I2C interface provides the data channel and the control channel for communicating between the host microcontroller and the MAX32660. The bootloader is enabled and disabled by either a serial command or hardware connectivity. The serial command is interpreted by the application, which configures the device to enter the bootloader mode. When using the hardware connectivity option, a single GPIO pin and the RSTN pin on the MAX32660 can be configured to allow the MAX32660 to enter the bootloader mode.

Detailed Description

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

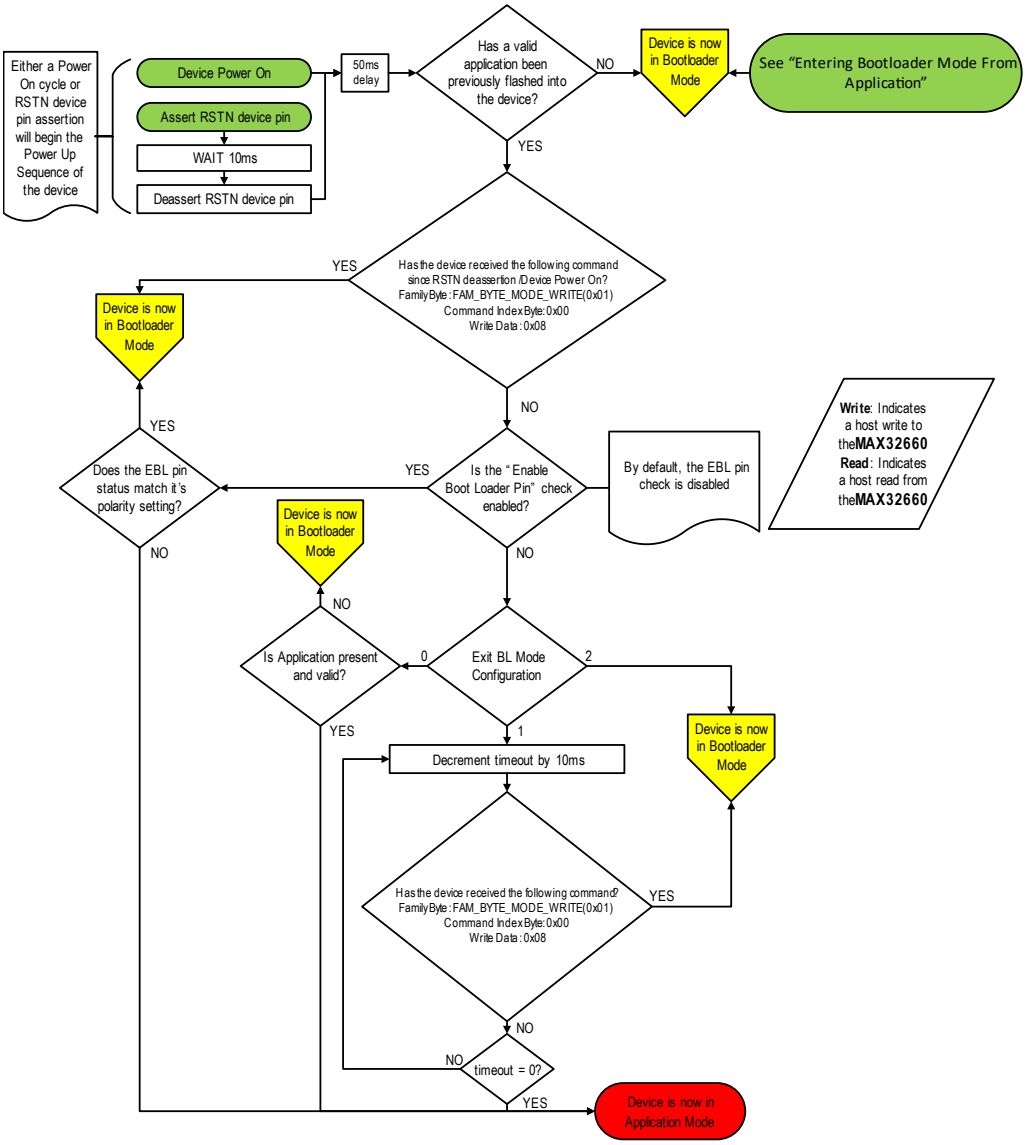


Figure 1. MAX32660 bootloader top level flow chart.

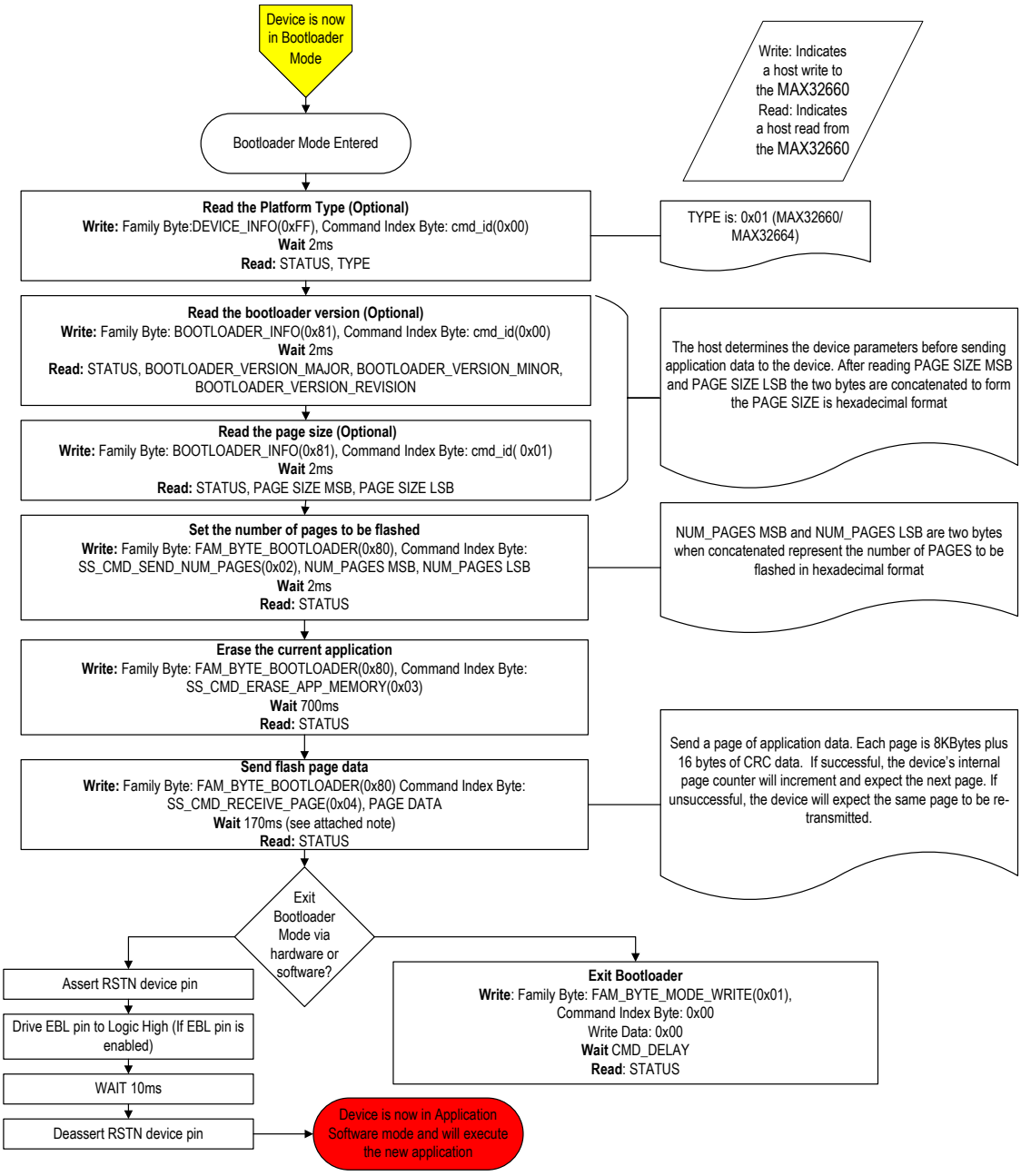


Figure 2. MAX32660 bootloader application loader flow chart.

Default I2C Slave Address

The default 8-bit I2C slave address for the bootloader is 0xAA.

GPIO Pins and RSTN Pin

Table 1 lists the descriptions for the GPIO and RSTN pins.

Table 1. GPIO and RSTN Pin Descriptions

MAX32660	DESCRIPTION	DIRECTION FROM MAX32660 SIDE
Pin RSTN	Reset_N	Input
GPIO P0.2	I2C0_SCL	Input
GPIO P0.3	I2C0_SDA	Input/Output

SWD Pins

Soft lock and disable the SWD pins (i.e., software debug pins, programming pins) by defining the SWD_LOCK definition in the configuration file of the source code. Soft-locking prevents the bootloader from accidental flashing through the SWD pins.

Table 2. SWD Pin Descriptions

MAX32660	DESCRIPTION	DIRECTION FROM MAX32660 SIDE
GPIO P0.0	SWDIO	Input/Output
GPIO P0.1	SDWDCLK	Input

Entering Bootloader Mode from Application Mode

This section discusses several methods for entering the bootloader mode from the application mode.

Host Serial Command using Power-On or Hard Reset

The MAX32660 can enter bootloader mode by performing the following steps:

1. Power cycle the MAX32660 or perform a hard reset with the RSTN pin.
2. The host microcontroller sends the command 0x01, 0x00, 0x08 to the MAX32660 to enter bootloader mode.

Without Using the RSTN Pin or GPIO Pins

Command the MAX32660 to enter the bootloader mode by using host serial commands. Change the "boot_mode" flag in the flash 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 the 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 either enabled through the build configuration file, "config\EvKit_V1," or through the I2C message protocol command. The MAX32660 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 50ms, the MAX32660 is in bootloader mode.

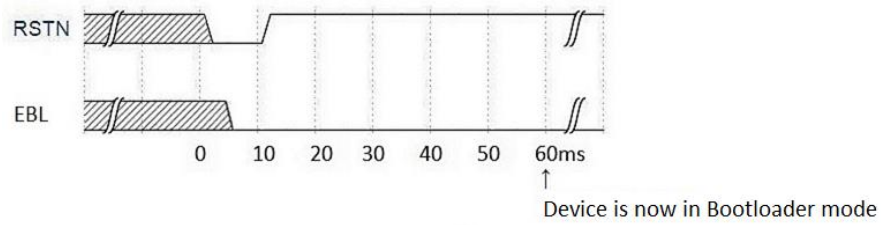


Figure 3. Entering bootloader mode through the EBL GPIO and RSTN pins.

Entering Application Mode from Bootloader Mode

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

A Valid Application is Programmed

If the EBL GPIO pin is disabled and a valid application is programmed into the MAX32660 using In-application Programming (IAP), the firmware automatically runs the application code.

Using the EBL GPIO Pin and the RSTN Pin

The MAX32660 enters application mode based on the sequencing of the EBL GPIO pin and the RSTN pin. The EBL GPIO pin is disabled in the bootloader by default and can be enabled either through the build configuration file ("config\EvKit_V1") or 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 50ms, the MAX32660 is in application mode.

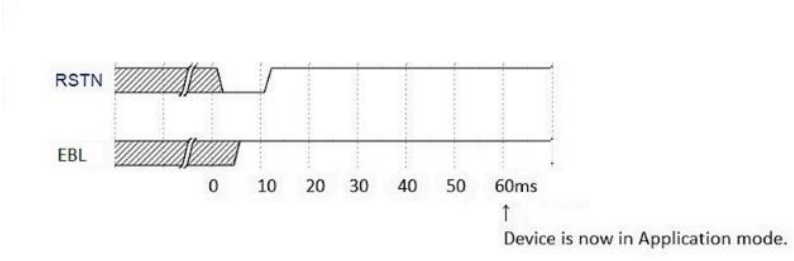


Figure 4. Entering application mode through the EBL GPIO pin and RSTN pin.

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 the SDA is stable and able to be read when the SCL is high. Typical I2C write/read transactions are shown in Figure 5.

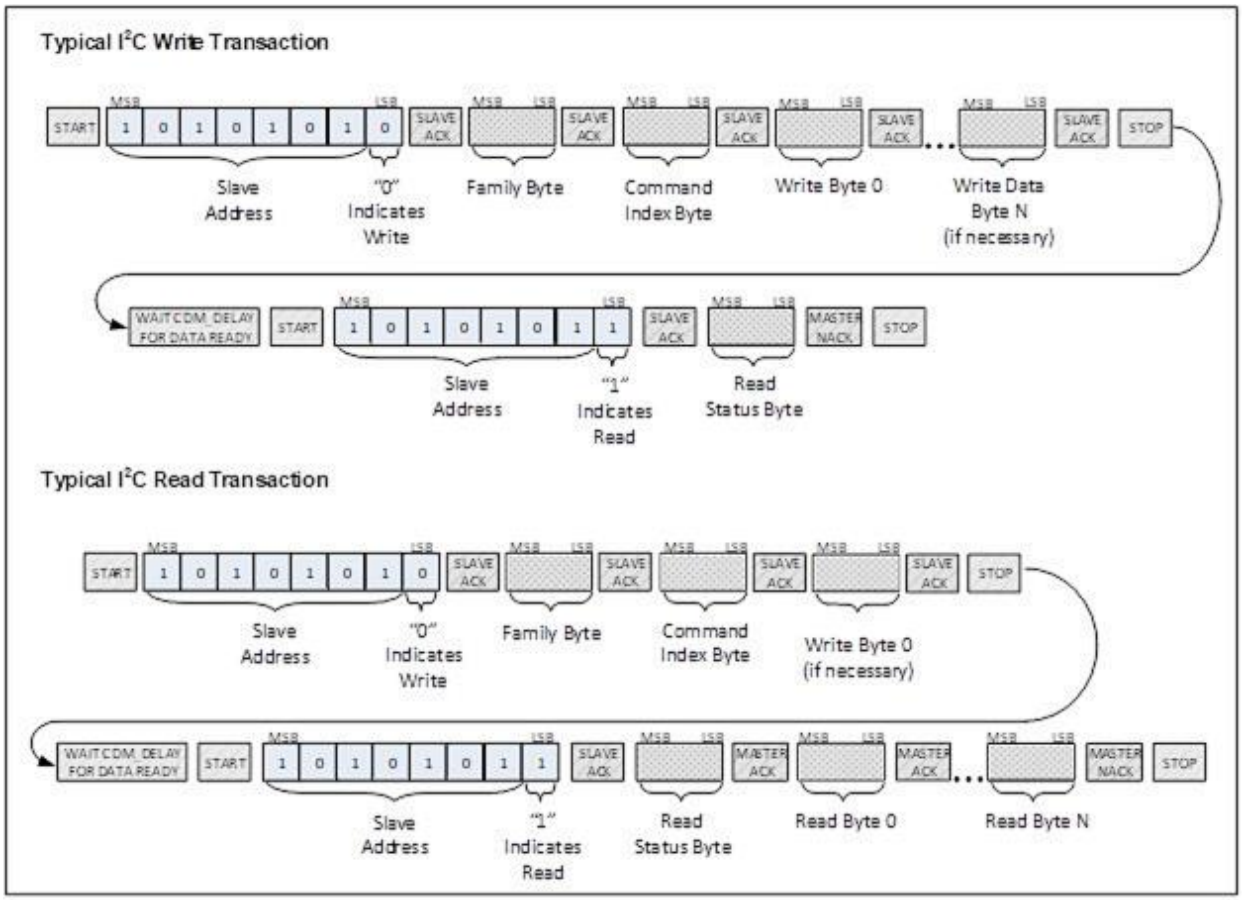


Figure 5. I2C write/read data transfer from host microcontroller.

The read status byte is an indicator of the success or failure of the write transaction. The read status byte must be accessed after each write transaction to the device. This ensures 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 3.

Table 3. Read Status Byte Values

READ STATUS BYTE VALUE	DESCRIPTION
0x00	SUCCESS. The write transaction was successful.
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.
0x05	ERR_TRY_AGAIN. Device is busy. Try again.
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.
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 one byte with the 7-bit slave address and a single write bit set to zero. The eight bits transferred as a slave address for the MAX32660 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 eight 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 eight 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 eight 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 eight 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 its data ready.
9. The master indicates a data transfer to the slave with a START condition.
10. The master transmits one byte with the 7-bit slave address and a single write bit set to one. This is an indication from the master its intent 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 command intended.

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 one byte with the 7-bit slave address and a single write bit set to zero. The eight bits transferred as a slave address for the MAX32660 are 0xAA for a write transaction. This write transaction precedes the actual read transaction to indicate to the device what 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 eight 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 eight 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 eight 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 (60µsec) 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 one byte with the 7-bit slave address and a single write bit set to one. This is an indication from the master its intent 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.

SCL and SDA Bus Drivers

The I²C bus expects SCL and SDA to be open-drain signals and the SDA and SCL pad circuits are automatically configured as open-drain outputs for the MAX32660 bootloader.

MAX32660 Bootloader I2C Message Protocol Definitions

Table 4 lists the MAX32660 bootloader I2C message protocol definitions.

Table 4. MAX32660 Bootloader I2C Message Protocol Definitions

HOST COMMAND					MAX32660 BOOTLOADER
FAMILY NAME	DESCRIPTION	FAMILY BYTE	INDEX BYTE	WRITE BYTES	RESPONSE BYTES
Device Mode	Select the device operating mode. The application must implement this.	0x01	0x00	**0x00**: Exit bootloader mode. **0x02**: Reset. **0x08**: Enter bootloader mode.	-
Device Mode	Read the device operating mode.	0x02	0x00	-	**0x00**: Application operating mode. **0x08**: Bootloader operating mode.
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.	-
Bootloader Flash	Set the authentication bytes. This is not required for a non-secure bootloader.	0x80	0x01	Use bytes the 16 bytes 0x34 to 0x43 from the .msbl file.	-
Bootloader Flash	Set the number of pages.	0x80	0x02	0x00, Number of pages specified by byte 0x44 from the .msbl file.	
Bootloader Flash	Erase the application flash memory.	0x80	0x03	-	-
Bootloader Flash	Send the page values. Each page includes 16 bytes of CRC.	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.	-
Bootloader Information	Get bootloader version.	0x81	0x00	-	Major version byte, Minor version byte, Revision byte
Bootloader Information	Get the page size in bytes.	0x81	0x01	-	Upper byte of page size, Lower byte of page size
Bootloader Configuration	Save bootloader configurations. Write this command after changes are made to any of the Bootloader Configuration settings.	0x82	0x00	-	-

HOST COMMAND					MAX32660 BOOTLOADER
FAMILY NAME	DESCRIPTION	FAMILY BYTE	INDEX BYTE	WRITE BYTES	RESPONSE BYTES
Bootloader Configuration	Select bootloader check. Configure the device to check the state of the EBL GPIO pin to decide whether to enter bootloader mode.	0x82	0x01	**0x00:** The device does not check the state of the EBL GPIO pin. (Default) **0x01:** The device checks the state of the EBL GPIO pin before entering bootloader mode.	-
Bootloader Configuration	Read bootloader check configuration. 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.
Bootloader Configuration	Select the EBL GPIO pin. Select which pin to use as the enter bootloader (EBL) GPIO pin. This command is only used if the Bootloader Configuration enter bootloader check is set to 1 (0xAA 0x82 0x01 0x00 0x01).	0x82	0x01	**0x00–0x09:** Acceptable range for the 16-bump WLP package. **0x00–0x0B:** Acceptable range for the 20-pin TQFN-EP and the 24-pin TQFN-EP.	-
Bootloader Configuration	Read the EBL GPIO pin. Read which pin is used as the enter bootloader (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

HOST COMMAND					MAX32660 BOOTLOADER
FAMILY NAME	DESCRIPTION	FAMILY BYTE	INDEX BYTE	WRITE BYTES	RESPONSE BYTES
Bootloader Configuration	Select the active state for the EBL GPIO pin.	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.	-
Bootloader Configuration	Read the active state for the EBL GPIO pin.	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. (Default) **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.
Bootloader Configuration	Exit bootloader mode. Determine how the bootloader enters application mode.	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.	-
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.

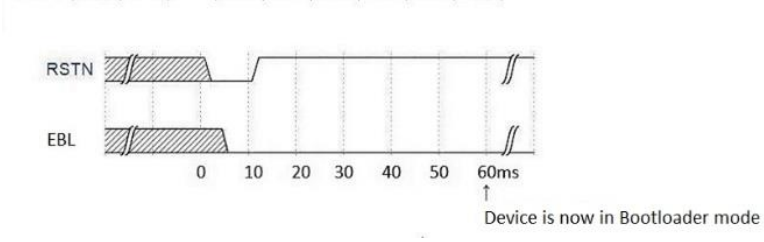
HOST COMMAND					MAX32660 BOOTLOADER
FAMILY NAME	DESCRIPTION	FAMILY BYTE	INDEX BYTE	WRITE BYTES	RESPONSE BYTES
Bootloader Configuration	Configure timeout exit. Set the length of the additional programmable timeout to use when Bootloader Configuration exit bootloader mode is set to 1 (AA 82 02 01). The system requires a 50ms non-programmable delay to switch to application mode.	0x82	0x02	**0x01, 0x00–0xFF:** Timeout is in increments of 10ms. Default timeout is 0ms at 0x00. Note: Timeout is cancelled if any commands are received during this period.	-
Bootloader Configuration	Read exit timeout configuration. Read the timeout to use when 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	**0x00–0xFF:** Timeout is in increments of 10ms. Default timeout is 0ms at 0x00.
Identity	Read the MCU type.	0xFF	0x00	-	**0x00:** MAX32625 **0x01:** MAX32660/MAX32664

MAX32660 In-Application Programming, Annotated Trace

The MAX32660 bootloader firmware supports In-Application Programming (IAP). The application must be converted to the .msbl format. See Appendix A for the procedure to create the .msbl file from the application binary.

Table 5 shows an example of the necessary I²C commands to flash the application to the MAX32660 using the .msbl file. The MAX32630FTHR acts as the host microcontroller. The MAX32660 uses the 8-bit slave address of 0xAA. Each 8192-byte page sent includes 16 CRC bytes for that page with 8208 bytes per page sent in the payload of the message. The number of pages is located at 0x44 in the .msbl file.

Table 5. I²C Command Example

HOST COMMAND	COMMAND DESCRIPTION	MAX32660 BOOTLOADER RESPONSE	RESPONSE DESCRIPTION
Sequence the MAX32660 to enter bootloader mode.  *Figure 6. Entering bootloader mode.*			
0xAA 0x01 0x00 0x08*	Set mode to 0x08 for bootloader mode.	0xAB 0x00	No error.
0xAA 0x02 0x00	Read mode.	0xAB 0x00 0x08	No error. Mode is bootloader.
0xAA 0xFF 0x00+	Get ID and MCU type.	0xAB 0x00 0x01	No error. MCU is MAX32660/MAX32664.
0xAA 0x81 0x00	Read bootloader firmware version.	0xAB 0x00 0x03 0x01 0x07	No error. Version is 3.1.7.
0xAA 0x81 0x01	Read bootloader page size.	0xAB 0x00 0x20 0x00	No error. Page size is 8192.
0xAA 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.	0xAB 0x00	No error.
00000044 00 00 00 00 **05** 00 00 20 04 00 00 00 00 80 01 20 *Figure 7. Page number byte 0x44 from the .msbl file.*			
0xAA 0x80 0x03*	Bootloader flash. Erase application.	0xAB 0x00	No error.
0xAA 0x80 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	Bootloader flash. Set the initialization vector bytes 0x28 to 0x32 from the .msbl file.	0xAB 0x00	No error.
00000028 00 00 00 00 00 00 00 00 **00 00 00 00 00 00 00 00** **00 00 00** 00 00 00 00 00 00 00 00 00 00 00 00 00 *Figure 8. Initialization vector bytes 0x28 to 0x32 from the .msbl file.*			

HOST COMMAND	COMMAND DESCRIPTION	MAX32660 BOOTLOADER RESPONSE	RESPONSE DESCRIPTION
0xAA 0x80 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	Bootloader flash. Set the authentication bytes 0x34 to 0x43 from the .msbl file.	0xAB 0x00	No error.

```
00000034  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00000040  00 00 00 00 05 00 00 20  04 00 00 00 00 80 01 20
```

Figure 9. Authentication bytes 0x34 to 0x43 from the .msbl file.

0xAA 0x80 0x04 0x00 0x80 0x01 … 0x00 0x00 0x00*	Bootloader flash. Send page bytes 0x4C to 0x205B from the .msbl file.	0xAB 0x00	No error.

```
0000004c  00 00 00 00 05 00 00 20  04 00 00 00 00 80 01 20
00000050  d1 23 00 00 41 22 00 00  0f 24 00 00 11 24 00 00
…
00002040  08 bf 0f 32 03 f1 ff 33  06 bf d2 b2 45 9d 1e df
0000205b  00 00 00 00 00 00 00 00  00 00 00 00 01 21 00 21
```

Figure 10. Send page bytes 0x4C to 0x205B from the .msbl file.

0xAA 0x80 0x04 0x01 0x21 0x00 … 0x00 0x00 0x00*	Bootloader flash. Send page bytes 0x205C to 0x406B from the .msbl file.	0xAB 0x00	No error.
0xAA 0x80 0x04 0x02 0x02 0xC1 … 0x00 0x00 0x00*	Bootloader flash. Send page bytes 0x406C to 0607B from the .msbl file.	0xAB 0x00	No error.
0xAA 0x80 0x04 0xEO 0x6C 0x1C … 0x00 0x00 0x00*	Bootloader flash. Send page bytes 0x607C to 0x808B from the .msbl file.	0xAB 0x00	No error.
0xAA 0x80 0x04 0xFF 0xC3 0x0D … 0x00 0x00 0x00*	Bootloader flash. Send page bytes 0x808C to 0XA09B from the .msbl file.	0xAB 0x00	No error.

Sequence MAX32660 to enter application mode.

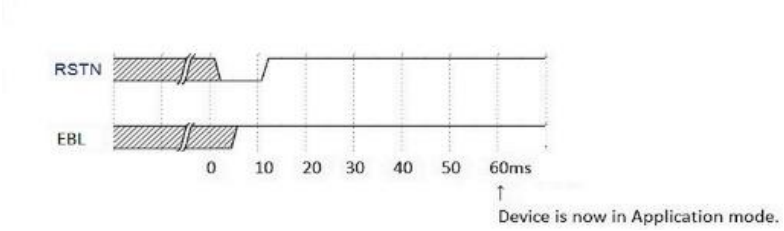


Figure 11. Entering application mode.

**Mandatory*

+Recommended

Example Python Code and Sample Host Code

Sample Python code to perform In-Application Programming using the MAX32630FTHR as the host can be found in the folder "max32660_bootloader_src\scripts\py."

Appendix

Download the MAX32660 bootloader software, sample host code, and sample .msbl code from the MAX32660 Evaluation Kit Design Resources tab.

Revision History

REVISION NUMBER	REVISION DATE	DESCRIPTION	PAGES CHANGED
0	10/18	Initial release	—