



Edvin Mellberg

ABSTRACT

This application report provides a brief overview of the serial bootloader that resides in ROM on the CC2538, CC13xx, and CC26xx device families. This document shows how the bootloader protocol can be used to perform basic operations like erasing and programming the flash of the devices. The device bootloaders support universal asynchronous receiver-transmitter (UART) and serial peripheral interface (SPI) as the protocol transportation layer.

This application report covers UART and is intended to be used with the associated example file, which can be downloaded from the following URL: <http://www.ti.com/lit/zip/swra466>. The example is created in Microsoft® Visual Studio® Professional 2015 and utilizes a library called Serial Bootloader Library to demonstrate an implementation of the serial bootloader protocol on Microsoft® Windows®.

Table of Contents

1 Introduction	3
2 ROM Bootloader	3
2.1 Configuring the Bootloader.....	5
2.2 Communication Protocol.....	9
2.3 Interface Configuration.....	10
3 Serial Bootloader Library (SBL)	12
3.1 SBL Return Values.....	12
3.2 SBL API.....	13
4 Example Project	14
4.1 Hardware Setup.....	16
4.2 Software Setup.....	18
4.3 Program Flow.....	18
5 References	23
6 Revision History	23

List of Figures

Figure 2-1. Simplified Flowchart for Entering Bootloader (CC13xx, CC26xx).....	4
Figure 2-2. Sequence Chart for Send and Receive Protocol.....	9
Figure 2-3. Sequence Chart for Connection Initialization.....	11
Figure 3-1. Sequence Chart for Ping Function Call.....	12
Figure 4-1. Successful Execution of SblAppEx for the CC1310 device.....	14
Figure 4-2. Execution of SblAppEx for the CC1310 Device Using Argument Parsing.....	15
Figure 4-3. PC to UART Connection.....	16
Figure 4-4. EM TX and RX Pins on XDS100v3 Emulator Bypass Header.....	17
Figure 4-5. Sequence Chart for initCommunication Function With Uninitialized Bootloader.....	19
Figure 4-6. Sequence Chart for Flash Sector Erase.....	19
Figure 4-7. Sequence Chart for Flash Write.....	21
Figure 4-8. Sequence Chart for CRC32 Command.....	22
Figure 4-9. Sequence Chart for SBL Function Reset.....	22

List of Tables

Table 2-1. Address of 8-Bit Bootloader Configuration Field (CC2538 variants).....	5
Table 2-2. CC2538 Bootloader Backdoor Encoding.....	5
Table 2-3. CC13x0, CC26x0 CCFG:BL_CONFIG Encoding.....	6

Table 2-4. CC13x1x3 and CC26x1x3 CCFG:BL_CONFIG Encoding.....	6
Table 2-5. CC13x2, CC26x2 CCFG: BL_CONFIG Encoding.....	7
Table 2-6. CC13x4, CC26x4 CCFG: BL_CONFIG Encoding.....	8
Table 2-7. ROM Bootloader Packet Format.....	9
Table 2-8. Packet Format Field Description.....	9
Table 2-9. Acknowledge or Not-Acknowledge Response.....	10
Table 2-10. Serial Interface Configuration: Evaluation Module Kits.....	10
Table 2-11. Serial Interface Configuration for CC13x2 and CC26x2 LaunchPad™ Development Kits.....	10
Table 2-12. Serial Interface Configuration for CC13x1x3, CC26x1x3, CC13x4, and CC26x4 LaunchPad™ Development Kits.....	10
Table 2-13. Possible Status Return Values From Bootloader.....	11
Table 3-1. SBL Function Return Values.....	12
Table 3-2. SBL Functions.....	13
Table 4-1. Application Example Bootloader Backdoor Enable I/O Pin: Evaluation Module Kits.....	17
Table 4-2. Application Example Bootloader Backdoor Enable I/O Pin: LaunchPad™.....	17
Table 4-3. Configuration: deviceType	18

Trademarks

Code Composer Studio™ and LaunchPad™ are trademarks of Texas Instruments.

Microsoft® and Visual Studio® are registered trademarks of Microsoft Corporation.

Windows® is a registered trademark of Microsoft Corporation in the United States and/or other countries, or both.

All trademarks are the property of their respective owners.

1 Introduction

The main purpose of the CC2538, CC13xx, and CC26xx ROM bootloader is to support functionality for programming a flash image into the device flash over either SPI or UART.

In this document, CC13xx and CC26xx refer to all devices in the device families CC13x0, CC26x0, CC13x1x3, CC26x1x3, CC13x2, CC26x2, CC13x4, and CC26x4. The CC13x2x7, CC26x2x7, CC2652RB, CC13x2PSIP, and CC26x2PSIP devices are included when referring to the CC13x2 and CC26x2 device family, unless the devices are explicitly stated separately.

The scope of this document is to show how to use the bootloader to perform basic operations such as erasing and programming flash. This document uses UART as the bootloader transportation layer.

2 ROM Bootloader

The built-in bootloader on the CC2538, CC13xx, and CC26xx devices starts running after a power-on reset if there is no valid application image in flash, determined by an *image valid* field in the customer configuration area (CCA) or , customer configuration (CCFG). For more information about the *image valid* field in CCA or CCFG, see the following documentation for each device family:

- [CC2538 ROM User's Guide](#)
- [CC13x0, CC26x0 SimpleLink™ Wireless MCU Technical Reference Guide](#)
- [CC13x2, CC26x2 SimpleLink™ Wireless MCU Technical Reference Manual](#)
- [CC13x1x3, CC26x1x3 SimpleLink™ Wireless MCU Technical Reference Manual](#)
- [CC13x2x7, CC26x2x7 SimpleLink™ Wireless MCU Technical Reference Manual](#)
- [CC13x4x10, CC26x4x10 SimpleLink™ Wireless MCU Technical Reference Manual](#)

Alternatively, the bootloader starts if the so-called bootloader backdoor is enabled and the associated pin that opens the backdoor is set to the correct logic level. If the bootloader is activated, the bootloader is ready for communicating with an external host 10ms after power-on-reset. Since the CC2538, CC13xx, and CC26xx ROM bootloader supports commands that can read the flash, it is also possible to disable the bootloader entirely for security reasons. The bootloader and backdoor functionality is configured in the CCA, CCFG.

Figure 2-1 shows as simplified flow chart for the CC13xx and CC26xx boot code. The flow is similar for CC2538 devices.

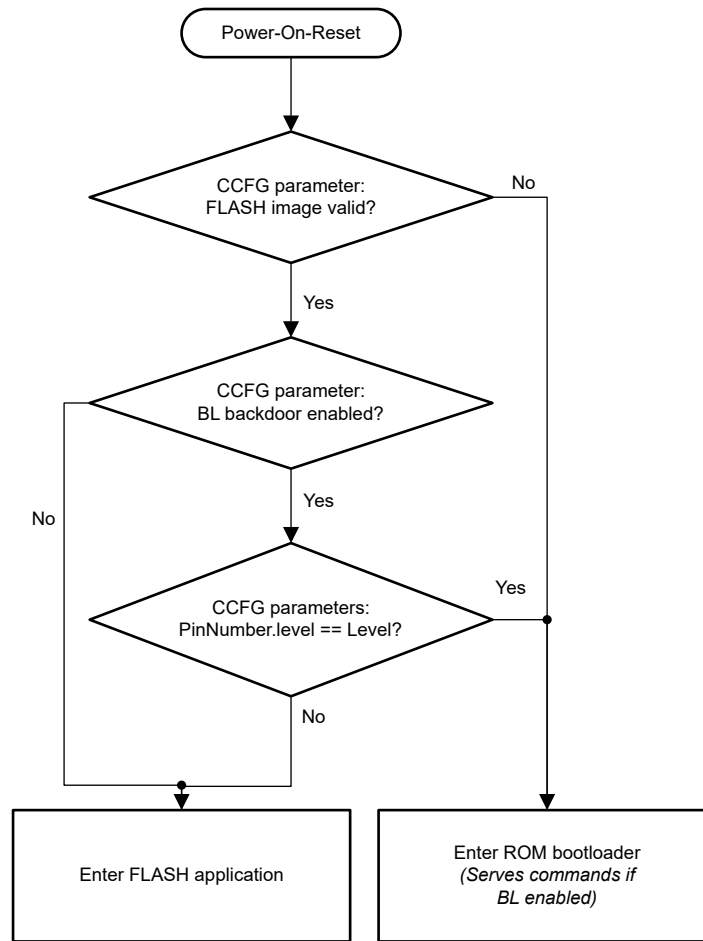


Figure 2-1. Simplified Flowchart for Entering Bootloader (CC13xx, CC26xx)

2.1 Configuring the Bootloader

2.1.1 CC2538

The customer configuration area for CC2538 is called CCA and is placed in the uppermost flash sector, so the absolute address of the CCA depends on the device flash size. An 8-bit field in the CCA configures the bootloader backdoor functionality (byte offset 0x7D7). [Table 2-1](#) lists the absolute address of this byte for different CC2538 variants.

Table 2-1. Address of 8-Bit Bootloader Configuration Field (CC2538 variants)

CC2538 Variant	Bootloader Configuration Address
Cx2538xF53 (512KB flash)	0x0027.FFD7
Cx2538xF23 (256KB flash)	0x0023.FFD7
Cx2538xF11 (128KB flash)	0x0021.FFD7

The structure of the bootloader configuration byte is shown in [Table 2-2](#). The pins that can open the bootloader backdoor are PA0 to PA7. Select which pin to use by writing a value from 0 to 7 in the three least significant bits of the backdoor configuration byte.

Table 2-2. CC2538 Bootloader Backdoor Encoding

Bit	Field	Value	Description	Default Value
7-5	Reserved	0	Reserved. Should be all ones.	111b
4	Enabled	0	Enable and disable backdoor function	1
		0	Backdoor and bootloader disable	
		1	Backdoor and bootloader enable	
3	Level		Sets active level for selected pin on pad A	1
		0	Active low	
		1	Active high	
2-0	Pin number		The number (0 - 7) of the pin on pad A that is used when backdoor is enable.	111b (7)

2.1.2 CC13x0, CC26x0

The customer configuration area for CC13x0 and CC26x0 is called CCFG and is located in the uppermost flash sector, so the absolute address of the CCFG depends on the device flash size. The bootloader configuration absolute address for the 128KB flash variant is 0x0001.FFD8, the 64KB flash variant is 0x0000.FFD8 and the 32KB flash variant is 0x0000.7FD8. The CC13x0, CC26x0 CCFG is also memory mapped with read access to address 0x5000.3000 for all flash variants. A 32-bit field in the CCFG configures the bootloader and backdoor functionality (byte offset 0xFD8).

The structure of the bootloader configuration field is shown in [Table 2-3](#). The configuration structure is little endian, meaning that the least significant byte is at the lowest address. Select which pin to use by writing the DIO number to the second byte of the configuration structure.

Table 2-3. CC13x0, CC26x0 CCFG:BL_CONFIG Encoding

Bit	Field	Value	Description	Byte Offset	Default Value
31:24	BOOTLOADER_ENABLE	0xC5 Any other value	Enable and disable bootloader Bootloader enabled Bootloader disabled	0xFDB	0xC5
23:17	RESERVED	0		0xFDA	0b111 1111
16	BL_LEVEL	0 1	Sets the active level of the selected pin. Active low Active high	0xFDA	1
15:8	BL_PIN_NUMBER		The number of the I/O pin that is level checked if the bootloader backdoor is enabled.	0xFD9	0xFF
7:0	BL_ENABLE	0xC5 Any other value	Enables and disables the bootloader backdoor. Bootloader enabled Bootloader disabled	0xFD8	0xFF

2.1.3 CC13x1x3, CC26x1x3

The customer configuration area for CC13x1x3 and CC26x1x3 is called CCFG and is located in the uppermost flash sector. The CC13x1x3 and CC26x1x3 devices have one flash size, 352KB, and the bootloader configuration absolute address is 0x0005.7FD8. The CC13x1x3 and CC26x1x3 CCFG is also memory mapped with read access to address 0x5000.3000. A 32-bit field in the CCFG configures the bootloader and backdoor functionality (byte offset 0x1FD8). The CCFG can be modified through the *Device Configuration* view in SysConfig when using Code Composer Studio™.

[Table 2-4](#) shows the structure of the bootloader configuration field. The configuration structure is little endian, meaning that the least significant byte is at the lowest address.

Table 2-4. CC13x1x3 and CC26x1x3 CCFG:BL_CONFIG Encoding

Bit	Field	Value	Description	Byte Offset	Default Value
31:24	BOOTLOADER_ENABLE	0xC5 Any other value	Enable and disable bootloader Bootloader enabled Bootloader disabled	0x1FDB	0xC5
23:17	RESERVED	0		0x1FDA	0b111 1111
16	BL_LEVEL	0 1	Sets the active level of the selected pin. Active low Active high	0x1FDA	1
15:8	BL_PIN_NUMBER ⁽¹⁾		The number of the I/O pin that is level checked if the bootloader backdoor is enabled.	0x1FD9	0xFF
7:0	BL_ENABLE	0xC5 Any other value	Enables and disables the bootloader backdoor. Bootloader enabled Bootloader disabled	0x1FD8	0xFF

(1) For the CC13x1x3 or CC26x1x3 device family, the number of DIOs available for the BL_PIN_NUMBER is limited. Refer to the Errata (advisory IOC_01) of the specific device for more information. This does not apply to the CC1312PSIP, CC2652xSIP, or CC2652RB devices.

2.1.4 CC13x2, CC26x2

The customer configuration area for CC13x2, CC26x2 is called CCFG and is located in the uppermost flash sector. The CC13x2, CC26x2 devices have two different flash sizes: all CC13x2, CC26x2 devices have a flash size of 352KB, except for the CC13x2x7, CC26x2x7 devices which have a flash size of 704KB. The bootloader configuration absolute address for CC13x2, CC26x2 is 0x0005.7FD8, while for CC13x2x7, CC26x2x7 the absolute address is 0x000A.FFD8. The CCFG is also memory mapped with read access to address 0x5000.3000 for both flash sizes. A 32-bit field in the CCFG configures the bootloader and backdoor functionality (byte offset 0x1FD8). The CCFG can be modified through the Device Configuration view in SysConfig when using Code Composer Studio.

Table 2-5 shows the structure of the bootloader configuration field. The configuration structure is little endian, meaning that the least significant byte is at the lowest address.

Table 2-5. CC13x2, CC26x2 CCFG: BL_CONFIG Encoding

Bit	Field	Value	Description	Byte Offset	Default Value
31:24	BOOTLOADER_ENABLE	0xC5 Any other value	Enable and disable bootloader Bootloader enabled Bootloader disabled	0x1FDB	0xC5
23:17	RESERVED	0		0x1FDA	0b111 1111
16	BL_LEVEL	0 1	Sets the active level of the selected pin Active low Active high	0x1FDA	1
15:8	BL_PIN_NUMBER ⁽¹⁾		The number of the I/O pin that is level checked if the bootloader backdoor is enabled.	0x1FD9	0xFF
7:0	BL_ENABLE	0xC5 Any other value	Enables and disables the bootloader backdoor. Bootloader enabled Bootloader disabled	0x1FD8	0xFF

(1) For the CC13x2, CC26x2 device family, the number of DIOs available for the BL_PIN_NUMBER is limited. Refer to the device-specific Errata (advisory IOC_01) for more information. This does not apply to the CC1312PSIP, CC2652xSIP, or CC2652RB devices.

2.1.5 CC13x4, CC26x4

The customer configuration area for CC13x4 and CC26x4 is called CCFG and is located outside of the main flash. The CC13x4 and CC26x4 devices all have a flash size of 1MB. The CC13x4 and CC26x4 devices CCFG are memory mapped with read access to address 0x5000.0000. A 32-bit field in the CCFG configures the bootloader and backdoor functionality (byte offset 0x0028). The CCFG can be modified through the *Device Configuration* view in SysConfig when using Code Composer Studio.

Table 2-6 shows the structure of the bootloader configuration field. The configuration structure is little endian, meaning that the least significant byte is at the lowest address.

Table 2-6. CC13x4, CC26x4 CCFG: BL_CONFIG Encoding

Bit	Field	Value	Description	Byte Offset	Default Value
31:24	BOOTLOADER_ENABLE	0xC5 Any other value	Enable and disable bootloader Bootloader enabled Bootloader disabled	0x002B	0xC5
23:17	RESERVED	0		0x002A	0b111 1111
16	BL_LEVEL	0 1	Sets the active level of the selected pin Active low Active high	0x002A	1
15:8	BL_PIN_NUMBER ⁽¹⁾		The number of the I/O pin that is level checked if the bootloader backdoor is enabled.	0x0029	0xFF
7:0	BL_ENABLE	0xC5 Any other value	Enables and disables the bootloader backdoor Bootloader enabled Bootloader disabled	0x0028	0xFF

- (1) For the CC13x2, CC26x2 device family, the number of DIOs available for the BL_PIN_NUMBER is limited. Refer to your specific device's Errata (advisory IOC_01) for more information. This does not apply to the CC1312PSIP, CC2652xSIP, CC2652RB devices.

2.2 Communication Protocol

The CC2538, CC13xx, and CC26xx bootloader uses the same format for receiving and sending packets. The actual signaling on SPI and UART transportation layers is different, but the packet format remains the same. [Table 2-7](#) shows the packet format and each field is described in [Table 2-8](#).

Table 2-7. ROM Bootloader Packet Format

Size (1 Byte)	Checksum (1 Byte)	Data byte 1	...	Data byte N
---------------	-------------------	-------------	-----	-------------

Table 2-8. Packet Format Field Description

Packet Field	Size (bytes)	Description
Size	1	The number of bytes in the packet, including the size byte.
Checksum	1	The checksum of the data. The checksum algorithm is the sum of the data bytes truncated to 8 bit. $Checksum = (\sum data) \bmod 256$
Data	1–253	The actual data bytes. The first data byte is typically the command byte of the bootloader.

Packet send and packet receive must adhere to the simple protocol shown in [Figure 2-2](#). Both the host device and the CC2538, CC13xx, or CC26xx bootloader can act as sender and receiver. The host device becomes the receiver when the device waits for a data response from the bootloader.

For more details about the communication protocol, see references [1](#) through [6](#) in the [References](#) section (also listed in [Section 2](#)).

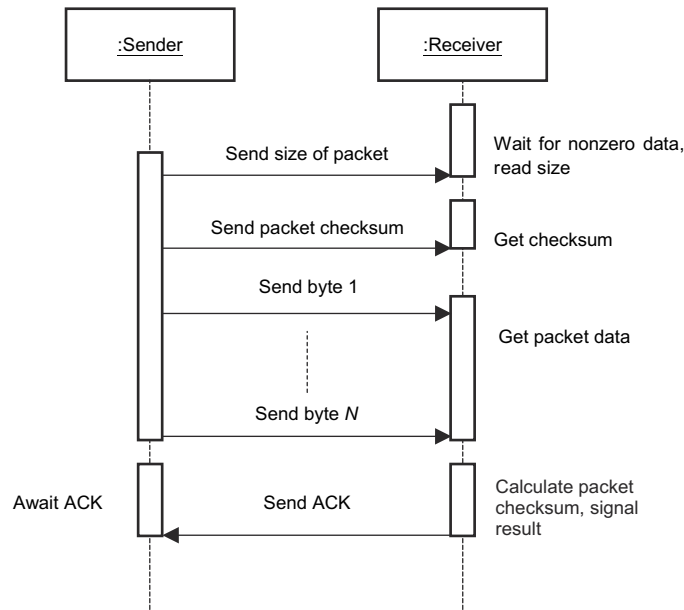


Figure 2-2. Sequence Chart for Send and Receive Protocol

2.2.1 ACK or NACK

The receiver responds with an acknowledgment (ACK) or not-acknowledged (NACK) to indicate whether the command was received properly or not. [Table 2-9](#) shows the ACK and NACK signature.

Table 2-9. Acknowledge or Not-Acknowledge Response

Protocol Byte	Value
ACK	0xCC
NACK	0x33

2.3 Interface Configuration

2.3.1 Hardware Pins

[Table 2-10](#), [Table 2-11](#), and [Table 2-12](#) show the hardware pins used by the ROM bootloader to communicate over UART and SPI. See also the bootloader transport layer section of the specific device family documentation ([1] through [6] in the [References](#) section).

Table 2-10. Serial Interface Configuration: Evaluation Module Kits

Signal	CC2538	CC13x0, CC26x0			EM Pin
		QFN48, 7x7	QFN32, 5x5	QFN32, 4x4	
UART_RX	PA0	DIO2	DIO1	DIO1	1.07
UART_TX	PA1	DIO3	DIO0	DIO2	1.09
SPI_CLK	PA2	DIO10	DIO10	DIO8	1.16
SPI_CSn	PA3	DIO11	DIO9	DIO7	1.14
SPI_MOSI	PA4	DIO9	DIO11	DIO9	1.18
SPI_MISO	PA5	DIO8	DIO12	DIO0	1.20

Table 2-11. Serial Interface Configuration for CC13x2 and CC26x2 LaunchPad™ Development Kits

Signal	CC2640R2 ⁽¹⁾	CC26x2R	CC1312R	CC1352x	LaunchPad™ Pin
UART_RX	DIO2	DIO2	DIO2	DIO12	3 ⁽²⁾
UART_TX	DIO3	DIO3	DIO3	DIO13	4 ⁽²⁾
SPI_CLK	DIO10	DIO10	DIO10	DIO10	7
SPI_CSn	DIO11	DIO11	DIO11	DIO11	18
SPI_MOSI	DIO9	DIO9	DIO9	DIO9	15
SPI_MISO	DIO8	DIO8	DIO8	DIO8	14

(1) The pinout is only valid for the QFN48, 7x7 package

(2) Reverse the order for the CC2640R2 LaunchPad™

Table 2-12. Serial Interface Configuration for CC13x1x3, CC26x1x3, CC13x4, and CC26x4 LaunchPad™ Development Kits

Signal	CC26x4x10	CC1314R10, CC13x1R3, CC26x1R3	CC1354x10, CC13x1P3, CC26x1P3	LaunchPad™ Pin
UART_RX	DIO12	DIO2	DIO12	3
UART_TX	DIO13	DIO3	DIO13	4
SPI_CLK	DIO10	DIO10	DIO10	7
SPI_CSn	DIO11	DIO11	DIO11	18
SPI_MOSI	DIO9	DIO9	DIO9	15
SPI_MISO	DIO8	DIO8	DIO8	14

The bootloader selects the first interface accessed by the external device. The inactive interface (UART or SPI) is disabled. To switch to the other interface, the device must be reset using, for example, pin reset.

2.3.2 UART Configuration

The UART data format is fixed at eight data bits, no parity, and one stop-bit. The UART bootloader utilizes auto detection of the baud rate (see [Section 2.3.3](#)); therefore, any baud rate below the maximum can be used.

- Maximum UART baud rate for CC2538: 460800 baud ¹
- Maximum UART baud rate for CC13xx and CC26xx (except for CC13x4 and CC26x4): 1.6M baud
- Maximum UART baud rate for CC13x4 and CC26x4: 1.2M baud

2.3.3 Establishing Communication

[Figure 2-3](#) shows the bare minimum needed to establish communication with the bootloader over UART, which includes sending two bytes with the value 0x55 to let the device detect the baud rate, followed by reading the device response, expecting an ACK if the auto baud rate routine was successful. If the device does not respond to the auto baud bytes, it is possible the device is not in bootloader mode, or the baud rate is not supported.

After a connection is made, any command can be sent to the bootloader. For more details about the bootloader commands, see the respective device family references [1] through [6] in the [References](#) section.

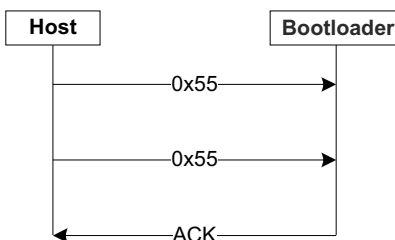


Figure 2-3. Sequence Chart for Connection Initialization

2.3.4 Status Command

Use the `CMD_GET_STATUS` command to check the status of the bootloader. Check the status after erasing or writing the flash memory to be sure that the erase and write were successful before proceeding. [Table 2-13](#) shows the possible status codes for the `CMD_GET_STATUS` command.

Table 2-13. Possible Status Return Values From Bootloader

Status Definition	Value	Description
<code>COMMAND_RET_SUCCESS</code>	0x40	Status for successful command
<code>COMMAND_RET_UNKNOWN_CMD</code>	0x41	Status for unknown command
<code>COMMAND_RET_INVALID_CMD</code>	0x42	Status for invalid command (incorrect packet size)
<code>COMMAND_RET_INVALID_ADR</code>	0x43	Status for invalid input address
<code>COMMAND_RET_FLASH_FAIL</code>	0x44	Status for failed attempt to program or erase the flash

¹ This data rate number can be doubled if an external 32MHz crystal oscillator is in use and selected using the `COMMAND_SET_XOSC` bootloader command. UART communication must be re-established after calling this command (see [Section 2.3.3](#)).

3 Serial Bootloader Library (SBL)

The SBL is a PC library for Microsoft Windows that implements a host API for communicating with the CC2538, CC13xx, and CC26xx serial bootloaders. The SBL library project is created in Visual Studio C++ Professional 2015. The serial bootloader library uses the Windows API to communicate with the serial COM port and is therefore *not* cross-platform compatible.

All functions in SBL are synchronous; meaning that the function does not return until ACK or NACK are received or an error occurs. [Figure 3-1](#) demonstrates a sequence chart of the SBL *ping()* function.

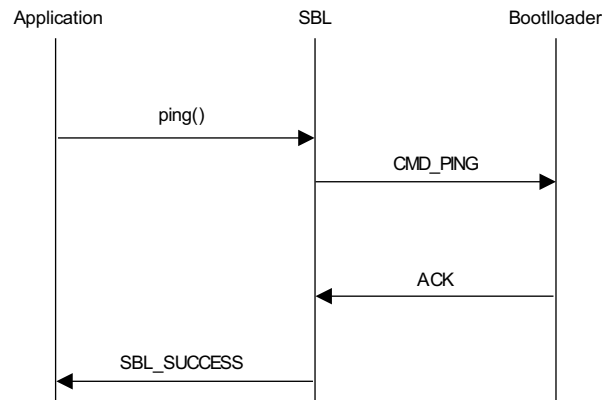


Figure 3-1. Sequence Chart for Ping Function Call

All bootloader commands can be accessed through functions within SBL; which allows easy execution of operations like erasing and writing to the flash memory directly through SBL.

For a more detailed description of the ROM bootloader and how to use all the serial commands, see the device-specific ROM user's guide [1] through [6] in the [References](#) section.

3.1 SBL Return Values

Each SBL function confirms whether the desired operation was successful or not by interpreting the bootloader response. [Table 3-1](#) shows a list of the possible return values from SBL functions and possible causes for them.

Table 3-1. SBL Function Return Values

Constant Name	Value	Cause
SBL_SUCCESS	0	Command successfully executed by bootloader
SBL_ERROR	1	Error during execution of command
SBL_ARGUMENT_ERROR	2	SBL function arguments invalid
SBL_TIMEOUT_ERROR	3	Bootloader response not received
SBL_PORT_ERROR	4	Failed to send data to or receive data from bootloader
SBL_ENUM_ERROR	5	Failed to enumerate COM devices
SBL_UNSUPPORTED_FUNCTION	6	Function is not supported for the chosen hardware

3.2 SBL API

Table 3-2 shows an overview of the SBL API. API functions that directly map to a bootloader command are marked with an X. There are a few differences in the bootloader commands between some of the device families, which also means there are differences in the API functions between some device families. If a function is not supported, this is explained in the *Description* field of Table 3-2. If called, an SBL function that is not supported for the chosen hardware returns the constant `SBL_UNSUPPORTED_FUNCTION` without doing anything.

Table 3-2. SBL Functions

SBL FUNCTION NAME	BOOTLOADER CMD		DESCRIPTION
	CC2538	CC13xx, CC26xx	
Create	N/A	N/A	Static function for creating an SBL device object.
calculateCrc32	X	X	Calculate CRC32 over the specified range.
connect	—	—	Initialize connection with ROM bootloader
cmdDownloadCrc	—	X	Prepares flash programming with the specified CRC32 value. Not implemented in the CC2538, CC13x0, or CC26x0 bootloader.
enumerate	N/A	N/A	Static function for enumerating COM ports on PC.
eraseFlashBank	—	X	Erases the entire flash. Not supported by CC2538, but the same functionality can be achieved by using <code>eraseFlashRange</code> .
eraseFlashRange	X	X	Erase the sectors in the specified range. Uses <code>CMD_SECTOR_ERASE</code> .
ping	X	X	Sends ping command.
readDeviceId	—	—	Uses <code>CMD_MEMORY_READ</code> to read device ID.
readFlashSize	—	—	Uses <code>CMD_MEMORY_READ</code> to read flash size.
readMemory32	X	X	Reads 32-bit word from device memory.
readMemory8	—	—	Uses <code>CMD_MEMORY_READ</code> to read 8-bit from device memory.
readRamSize	—	—	Uses <code>CMD_MEMORY_READ</code> to read RAM size.
readStatus	X	X	Reads bootloader status.
reset	X	X	Resets device using <code>CMD_RESET</code> .
run	X	—	Runs the device CPU from the specified address. Not supported by CC13xx or CC26xx.
setCCFG	—	X	Set CC13xx and CC26xx CCFG. Not supported by CC2538.
setXosc	X	—	Switch to external oscillator. Not supported by CC13xx or CC26xx.
writeFlashRange	X	X	Writes FLASH using <code>CMD_DOWNLOAD</code> and <code>CMD_DATA_SEND</code> .
writeMemory32	X	X	Writes 32-bit word to device memory using <code>CMD_MEMORY_WRITE</code> .
writeMemory8	—	—	Implements 8-bit write to device memory using <code>CMD_MEMORY_READ</code> and <code>CMD_MEMORY_WRITE</code> .

4 Example Project

The example application for SBL is created for Visual Studio C++ Professional 2015 and is tested using the hardware included in the CC2538, CC13xx, and CC26xx development kits. development kits; note that this example application only applies to certain devices. The example application can be downloaded from the following URL: <http://www.ti.com/lit/zip/swra466>

Sb1AppEx is a test application that performs the following actions using the CC2538, CC13xx, or CC26xx ROM bootloader:

- Erase flash
- Program flash
- Verify flash content
- Reset device

Figure 4-1 illustrates a successful execution of the test application.

```

+-----+
| Serial Bootloader Library Example Application
+-----+

+-----+
| COM ports:
+-----+
| Description
| XDS110 Class Application/User UART (COM24)
| XDS110 Class Auxiliary Data Port (COM25)
| Intel(R) Active Management Technology - SOL (COM3)
+-----+

Select COM port: 24
+-----+
| Supported devices:
+-----+
| IDx | Device
+-----+
| 0 | CC2538
| 1 | CC13x0/CC26x0
| 2 | CC2640R2
| 3 | CC13x2/CC26x2
| 4 | CC13x2PSIP/CC26x2PSIP
| 5 | CC13x2x7/CC26x2x7
| 6 | CC2652RB
| 7 | CC13x1x3/CC26x1x3
| 8 | CC13x4/CC26x4
+-----+

Select target device: 1
+-----+
| Select Binary File:
+-----+
| IDx | Binary File
| 0 | Blinky Example
| 1 | Use Custom Binary (enter binary file name in next step)
+-----+

Select Binary Option: 0

Connecting (COM24 @ 230400 baud) ...
100% (626.00ms)
Erasing flash ...
100% (1910.00ms)
Writing flash ...
100% (36526.00ms)
Calculating CRC on device ...
100% (94.00ms)
Comparing CRC ...
OK
Resetting device ...
OK

```

Figure 4-1. Successful Execution of Sb1AppEx for the CC1310 device

Figure 4-2 shows that the application can also be executed in a single command using argument parsing.

```

PS Z:\serial-bootloader-library\bin> .\sblAppEx.exe -help

usage: sblAppEx.exe [-help] COM_PORT DEVICE_ID BINARY [BINARY_CCFG]

    "Flashes a custom binary file using the ROM Bootloader."

arguments:
  -help, help      prints this user help guide and exits
  COM_PORT         (int) the COM port which the device is connected to, ex: '12' for the COM12 port
  DEVICE_ID        (int [0-7]) the IDx of the connected device, ex: '2' for the CC2640R2 device
                  (see list of supported devices under 'extra info' below)
  BINARY           (string) the name of the binary file to be loaded, ex: 'blinky_example.bin'
  BINARY_CCFG      (string) the name of the CCFG binary file to be loaded, ex: 'blinky_example_onlyccfg.bin'
                  (Only required for CC13x4/CC26x4 devices, where the CCFG is located outside of the main flash)

extra info:
  The following devices are currently connected to the host:

      | XDS110 Class Application/User UART (COM24)
      | XDS110 Class Auxiliary Data Port (COM25)
      | Intel(R) Active Management Technology - SOL (COM3)

  The following devices are supported:

      | IDx | Device
      |----|-----
      | 0  | CC2538
      | 1  | CC13x0/CC26x0
      | 2  | CC2640R2
      | 3  | CC13x2/CC26x2
      | 4  | CC13x2PSIP/CC26x2PSIP
      | 5  | CC13x2x7/CC26x2x7
      | 6  | CC2652RB
      | 7  | CC13x1x3/CC26x1x3
      | 8  | CC13x4/CC26x4

PS Z:\serial-bootloader-library\bin> .\sblAppEx.exe 24 1 .\blinky_backdoor_select_btn26x0.bin

Connecting (COM24 @ 230400 baud) ...
100% (627.00ms)
Erasing flash ...
100% (1923.00ms)
Writing flash ...
100% (36529.00ms)
Calculating CRC on device ...
100% (78.00ms)
Comparing CRC ...
OK
Resetting device ...
OK
  
```

Figure 4-2. Execution of SblAppEx for the CC1310 Device Using Argument Parsing

4.1 Hardware Setup

The SBL communicates with the ROM bootloader over a serial COM port on the PC. If a built-in COM port is not available, a USB-to-serial interface can act as a virtual COM port.

Figure 4-3 demonstrates two different ways to connect the PC to the device. One is using a level shifter to convert the UART signal from RS232 to TTL signals. The other option is to use a USB-to-UART bridge similar to the one integrated into the XDS110 that is used for the CC13xx or CC26xx LaunchPad Development Kits.

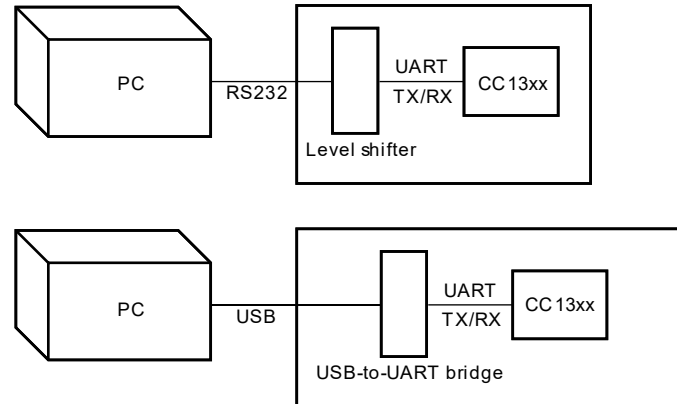


Figure 4-3. PC to UART Connection

4.1.1 LaunchPad™ Development Kit Virtual COM Port

The LaunchPad Development Kits for CC13xx and CC26xx devices come with built-in support for virtual COM port, which is enabled by default. The virtual COM port is achieved through the XDS110 Class Application - User UART. The XDS110 is either integrated directly on the LaunchPad or can otherwise be added separately using the LP-XDS110 development kit. See the specific LaunchPad documentation for details.




4.1.2 SmartRF06EB Virtual COM Port

SmartRF06EB [7] comes with a built-in support for virtual COM port that can be used together with a CC2538EM [8] or a CC2650EM [9].

To enable the virtual COM port on SmartRF06EB, a jumper must be mounted on the *Enable UART over XDS100v3* header and mount all the jumpers on the *XDS100V3 BYPASS* header.

4.1.2.1 External Serial Interface

If a SmartRF06EB is being used to bypass the XDS100v3 Emulator to use an external serial interface, connect the external serial interface to the EM RX and EM TX pins on the *XDS100v3 BYPASS* header as shown in Figure 4-4.

-  Jumper
-  EM TX
-  EM RX

XDS100v3 BYPASS

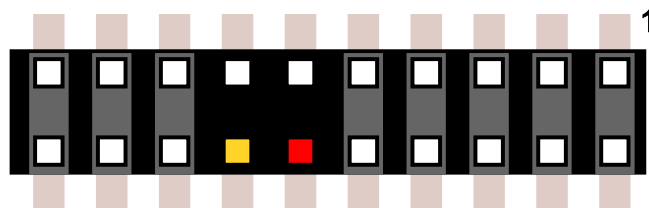


Figure 4-4. EM TX and RX Pins on XDS100v3 Emulator Bypass Header

4.1.3 Bootloader Backdoor

The `Sb1AppEx` example is written for CC2538 and CC2650 (7×7) Evaluation Modules (EVMs) and CC13xx and CC26xx LaunchPad Development Kits. The application example image, called `blinky_backdoor_select_btn26x*.bin`, programmed onto the device triggers the SmartRF06EB or the LaunchPad to blink the LEDs. The example image enables the bootloader backdoor so that the bootloader can be triggered using an I/O pin.

Table 4-1 and Table 4-2 show the I/O pin used by the application image for opening the bootloader backdoor. For the SmartRF06EB, this I/O pin is connected to the SELECT button. To enter the bootloader backdoor, hold down the SELECT button (corresponds to logic '0') while pressing the EM reset button on the SmartRF06EB. For the LaunchPad, the bootloader backdoor enable pin is connected to the BTN-1 button. The BTN-1 must be pressed when the LaunchPad reset button is pressed to enter the bootloader backdoor.

Table 4-1. Application Example Bootloader Backdoor Enable I/O Pin: Evaluation Module Kits

CC2538	CC2650			EM Pin
	QFN48 (7×7)	QFN32 (5×5)	QFN32 (4×4)	
PA3	DIO11	DIO9	DIO7	1.14

Table 4-2. Application Example Bootloader Backdoor Enable I/O Pin: LaunchPad™

CC13x0, CC26x0	CC13x1x3, CC26x1x3	CC13x2, CC26x2	CC13x2x7, CC26x2x7	CC2652RB	CC13x4, CC26x4
DIO13	DIO15	DIO15	DIO13	DIO13	DIO15

Note

The application also allows for programming a custom flash image besides the example image. When programming a custom flash image, the bootloader backdoor enable I/O pin assignments, as shown in Table 4-1 and Table 4-2, no longer applies. The designer must make sure that the bootloader is enabled in the CCA, CCFG of the flash image to be programmed; otherwise, the ROM bootloader is disabled and programming over UART and SPI is no longer possible.

4.2 Software Setup

The SblAppEx example has two configuration options: device type and baud rate.

4.2.1 Device Type

The device type is configured using the `deviceType` variable found in `sblAppEx.cpp`. The variable controls which bootloader commands the SBL can use, and which firmware image the SblAppEx programs onto the device. The `deviceType` variable is a binary-coded decimal (BCD) of the device name. Table 4-3 lists the supported device types and corresponding `deviceType` value.

Table 4-3. Configuration: deviceType

Device	DeviceType Value
CC2538	0x2538
CC13x0, CC26x0	0x2650
CC2640R2	0x2640
CC13x2, CC26x2; CC13x2x7, CC26x2x7; CC13x2PSIP, CC26x2PSIP; CC2652RB	0x2652
CC13x1x3, CC26x1x3	0x2651
CC13x4, CC26x4	0x2674

4.2.2 Baud Rate

The baud rate is configured by using the `baudRate` variable found in `sblAppEx.cpp`. The supported UART baud rates for CC2538, CC13xx, and CC26xx devices are documented in Section 2.3.2. The default baud rate is supported by all devices.

4.3 Program Flow

This section covers the SBL function calls discussed in the SblAppEx example project, which can be downloaded from: <http://www.ti.com/lit/zip/swra466>, and the underlying bootloader commands used.

4.3.1 Enumerate COM Ports

The enumerate function in SBL uses the Windows API to list the available COM ports. The first argument is a pointer to a `ComPortElement` array. The second argument specifies the maximum number of COM ports to enumerate. If the COM port to use is known, this function call can be skipped.

4.3.2 Create Device

The SBL must be told which device the SBL is working with. The Create function supports a single argument, the supported input values are given in Table 4-3. The Create function returns an instance of the `SblDevice` class that supports the specified hardware.

4.3.3 Connect

The connect function takes two parameters: the COM port number (see Section 4.3.1) and the baud rate (see Section 2.3.2).

The CC2538 ROM bootloader supports switching from the internal oscillator of the device to an external oscillator (if available). Switching to an external oscillator increases the maximum baud rate supported by the CC2538 ROM bootloader. If an external oscillator is to be used, a third argument (Boolean TRUE) can be passed to the connect function, this third parameter is optional and FALSE by default.

To check whether the connection already is initialized, the `initCommunication` function in the SBL sends a dummy command and waits for the bootloader to respond with an ACK. If no connection already exists, the `initCommunication` function sends the auto baud rate routine (described in Section 2.3.2), expecting an ACK from the ROM bootloader. Figure 4-5 shows an example of this sequence.

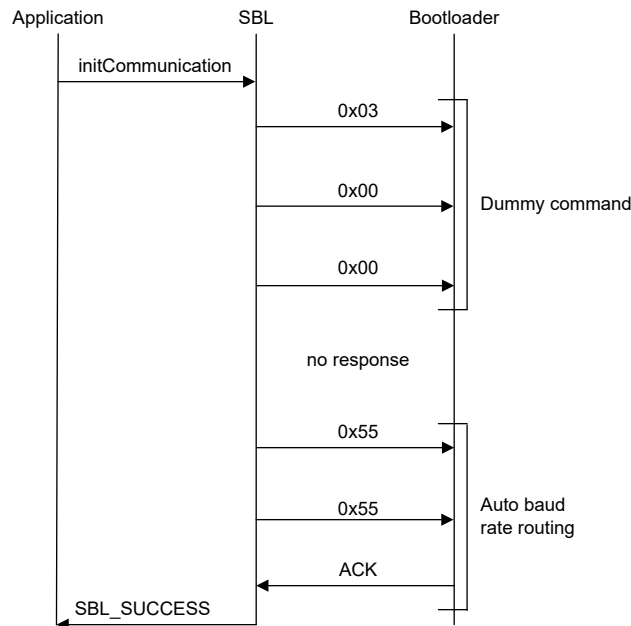


Figure 4-5. Sequence Chart for `initCommunication` Function With Uninitialized Bootloader

When the connection is established, the connect function retrieves the device ID by using the serial bootloader command `CMD_GET_CHIP_ID` and FLASH size and RAM size by using the command `CMD_MEMORY_READ` to read from a location storing these values.

4.3.4 Erase Flash Range

The `eraseFlashRange` function uses the bootloader command `CMD_ERASE` for CC2538 and `CMD_SECTOR_ERASE` for CC13xx and CC26xx.

The CC13xx or CC26xx `CMD_SECTOR_ERASE` takes an address parameter and erases the flash sector in which the address is located.

The CC2538 `CMD_ERASE` command requires a second argument for the erase size. The CC2538 bootloader erases the flash sectors (2KB) that are covered by the range [address, address + size].

After each bootloader erase command, `eraseFlashRange` checks the bootloader status using the `CMD_GET_STATUS` command.

Figure 4-6 shows the sequence chart for a flash erase using the serial bootloader protocol. The last four bytes in the command (`data size`) is specific for CC2538. For CC13xx or CC26xx, the `CMD_SECTOR_ERASE` command (and consequent `CMD_GET_STATUS`) must be repeated for each flash sector to erase.

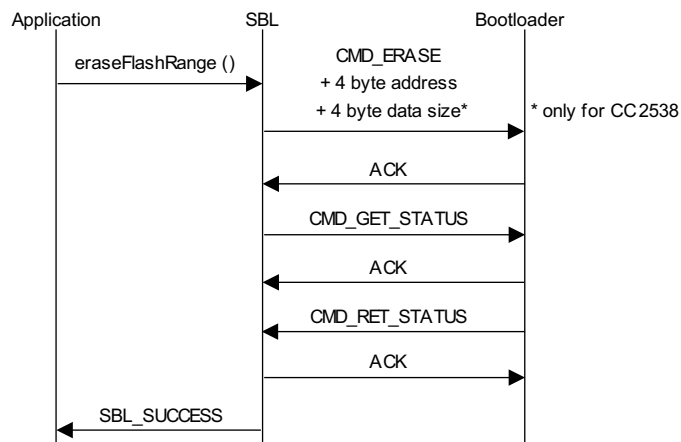


Figure 4-6. Sequence Chart for Flash Sector Erase

If the whole Flash memory is to be erased on CC13xx or CC26xx, use the `CMD_BANK_ERASE` command. This erases the whole Flash memory in one operation, which is faster than deleting sectors individually.

4.3.5 Write Flash Range

To write data to the flash memory, use the SBL function `writeFlashRange`. `writeFlashRange` sends the `CMD_DOWNLOAD` command to the bootloader together with the start address and the download size in bytes. The bootloader is now prepared to receive the specified amount of data and write the data to flash, starting at the specified address.

To transfer the data, use the `CMD_SEND_DATA` command. A maximum of 252 bytes of data can be transferred per the `CMD_SEND_DATA` command. If the data to be downloaded is larger than 252 bytes, the `CMD_SEND_DATA` command must be repeated. The SBL `writeFlashRange` function handles splitting data transfer into multiple `CMD_SEND_DATA` commands.

The status of the bootloader is read after both the `CMD_DOWNLOAD` command and after each `CMD_SEND_DATA` command by using the `CMD_GET_STATUS` command. This is to make sure that the start address and firmware size are valid, and that the data was successfully programmed into the flash. If the status indicates an error, the internal address pointer of the bootloader is not incremented, allowing the data to be re-transferred.

For the CC13x4 or CC26x4 device family, the CCFG is located outside of the main flash. This means that two iterations of the SBL `writeFlashRange` function are needed; one for the application binary and one for the CCFG binary.

Figure 4-7 demonstrates the flash write sequence using the SBL function `writeFlashRange`.

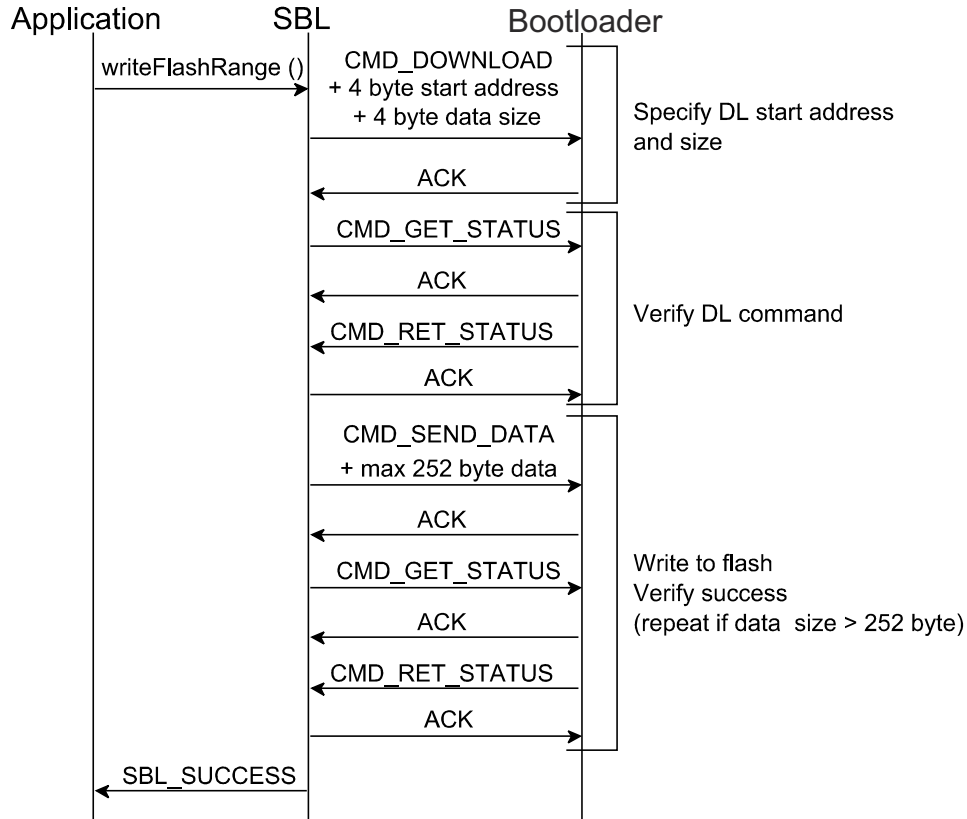


Figure 4-7. Sequence Chart for Flash Write

4.3.6 Calculate CRC32

To verify that the firmware was successfully programmed into the Flash memory, use the SBL function `calculateCrc32` to get a CRC32 checksum of a specified part of the Flash memory from the bootloader. The `calculateCrc32` function uses the command `CMD_CRC32` together with a start address and the number of bytes to include in the CRC32 checksum.

For CC13xx and CC26xx, the bootloader also expects a read repeat count. Setting this to `0x00000000` makes sure that the data locations are only read once.

The CC2538, CC13xx, and CC26xx bootloaders use the CRC-32-IEEE 802.3 with the following polynomial to calculate CRC checksum.

$$\text{CRC32}_{\text{poly}} = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$$

An example of how to calculate the checksum using the CRC32_{poly} is implemented in the SBL example project. Figure 4-8 shows the sequence chart for the calculateCrc32 function.

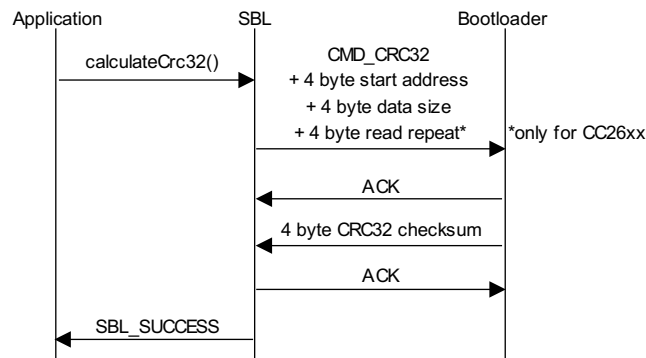


Figure 4-8. Sequence Chart for CRC32 Command

4.3.7 Reset

To run the firmware after it is written and verified, the device needs to be reset. This reset is done using the SBL reset function. The SBL reset function sends the CMD_RESET command to the bootloader to invoke a system reset. The connection between the host and the device breaks after the CMD_RESET command is sent and an ACK is received from the bootloader. Figure 4-9 shows the sequence chart for the reset function.

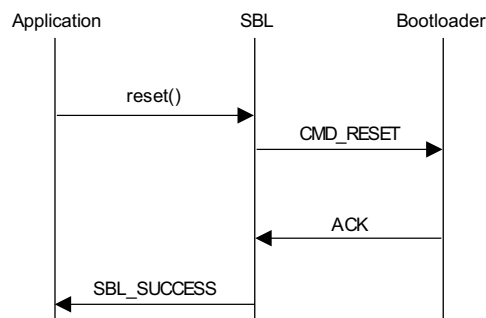


Figure 4-9. Sequence Chart for SBL Function Reset

5 References

1. Texas Instruments, [CC2538 ROM User's Guide](#)
2. Texas Instruments, [CC13x0, CC26x0 SimpleLink™ Wireless MCU Technical Reference Guide](#)
3. Texas Instruments, [CC13x2, CC26x2 SimpleLink™ Wireless MCU Technical Reference Manual](#)
4. Texas Instruments, [CC13x1x3, CC26x1x3 SimpleLink™ Wireless MCU Technical Reference Manual](#)
5. Texas Instruments, [CC13x2x7, CC26x2x7 SimpleLink™ Wireless MCU Technical Reference Manual](#)
6. Texas Instruments, [CC13x4x10, CC26x4x10 SimpleLink™ Wireless MCU Technical Reference Manual](#)
7. Texas Instruments, [SmartRF06 Evaluation Board \(EVM\) User's Guide](#)
8. Texas Instruments, [CC2538DK Product Page](#)
9. Texas Instruments, [CC2650DK Product Page](#)

6 Revision History

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

Changes from Revision D (August 2021) to Revision E (July 2024) Page

- Added the new CC13x1x3, CC26x1x3; CC13x2x7, CC26x2x7; CC2652RB, CC13x2PSIP, CC26x2PSIP, and CC13x4, CC26x4 devices throughout the document..... 1
- The SBL and accompanying example software are updated to support the new devices..... 1

Changes from Revision C (March 2020) to Revision D (August 2021) Page

- Updated the numbering format for tables, figures and cross-references throughout the document..... 3

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES "AS IS" AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI's products are provided subject to [TI's Terms of Sale](#) or other applicable terms available either on [ti.com](https://www.ti.com) or provided in conjunction with such TI products. TI's provision of these resources does not expand or otherwise alter TI's applicable warranties or warranty disclaimers for TI products.

TI objects to and rejects any additional or different terms you may have proposed.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265
Copyright © 2024, Texas Instruments Incorporated