Application Report
SWRA466B–February 2015–Revised June 2019

CC2538/CC26x0/CC26x2 Serial Bootloader Interface

Elin Wollert

ABSTRACT

This application report provides a brief overview on the serial bootloader that resides in ROM on the CC2538, CC13x0/CC26x0, and CC13x2/CC26x2 devices. 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 associated example file, which can be downloaded from the following URL: http://www.ti.com/lit/zip/swra466. The example is created in Visual Studio® Professional 2015 and utilizes a library called Serial Bootloader Library to demonstrate an implementation of the serial bootloader protocol on Windows®.

Contents

List of Figures

List of Tables

Trademarks

1 Introduction

The main purpose of the CC2538, CC13x0/CC26x0 and CC13x2/CC26x2 ROM bootloader is to support functionality for programming a flash image into the device flash over either SPI or UART. This document will hereafter refer to the CC13x0/CC26x2 device family, including CC2640R2, as CC26x0 and CC13x2/CC26x2 device family as CC26x2 for shortness. When used, CC26xx refers to both CC26x0 and CC26x2 device families.

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

2 ROM Bootloader

The built-in bootloader on the CC2538, CC26x0, and CC26x2 devices start 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/CCFG). For more information about the "image valid" field in CCA/CCFG, see the *CC2538 ROM user's guide* [1], the *CC13x0, CC26x0 SimpleLink™ wireless MCU technical reference guide* [2], and the *CC13x2, CC26x2 SimpleLink™ wireless MCU technical reference manual* [3].

Alternatively, the bootloader start 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, it is ready for communicating with an external host 10 ms after power-on-reset.

Since the CC2538, CC26x0, and CC26x2 ROM bootloaders support 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 1 shows as simplified flow chart for the CC26x0 and CC26x2 boot code. The flow is similar for CC2538 devices.

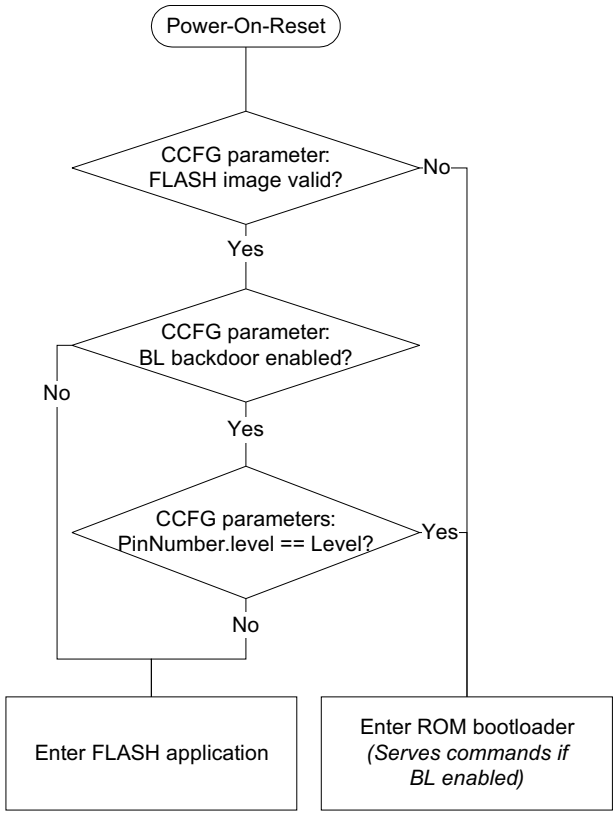

Figure 1. Simplified Flowchart for Entering Bootloader (CC26x0 and CC26x2)

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 1 lists the absolute address of this byte for different CC2538 variants.

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

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

The structure of the bootloader configuration byte is shown in Table 2. The pins that can open the bootloader backdoor are PA0 - 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. CC2538 Bootloader Backdoor Encoding

Bit	Field	Value	Description	Default Value
7-5	Reserved	0	Reserved. Should be all ones.	111b
4	Enabled		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 CC26x0

The customer configuration area for 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 128 KB flash variant is 0x0001.FFD8, the 64 KB flash variant is 0x0000.FFD8 and the 32 KB flash variant is 0x0000.7FD8. The 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 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 3. CC26x0 CCFG:BL_CONFIG Encoding

Bit	Field	Value	Description	Byte Offset	Default Value
31:24	BOOTLOADER_ENABLE		Enable and disable bootloader	0xFDB	0xC5
		0xC5	Bootloader enabled		
		Any other value	Bootloader disabled		
23:17	RESERVED	0		0xFDA	0b111 1111
16	BL_LEVEL		Sets the active level of the selected pin.	0xFDA	1
		0	Active low		
		1	Active high		
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		Enables and disables the bootloader backdoor.	0xFD8	0xFF
		0xC5	Bootloader enabled		
		Any other value	Bootloader disabled		

2.2 CC26x2

The customer configuration area for CC26x2 is called CCFG and is located in the uppermost flash sector. The CC26x2 devices have one flash size, 352 KB, and the bootloader configuration absolute address is 0x0005.7FD8. The CC26x2 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 structure of the bootloader configuration field is shown in Table 4. 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 4. CC26x2 CCFG:BL_CONFIG Encoding

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

2.3 Communication Protocol

The CC2538, CC26x0, and CC26x2 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. The packet format is shown in Table 5 and each field is described in Table 6.

Table 5. ROM Bootloader Packet Format

Size (1 Byte)	Checksum (1 Byte)	Data byte 1	…	Data byte N

Table 6. 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 = (∑data) mod 256*
Data	0-253	The actual data bytes. The first data byte is typically the bootloader's command byte.

Packet send and packet receive must adhere to the simple protocol shown in Figure 2. Both the host device and the CC2538/CC26x0/CC26x2 bootloader can act as sender and receiver. The host device becomes the receiver when it waits for a data response from the bootloader.

For more details about the communication protocol, see [1], [2], and [3].

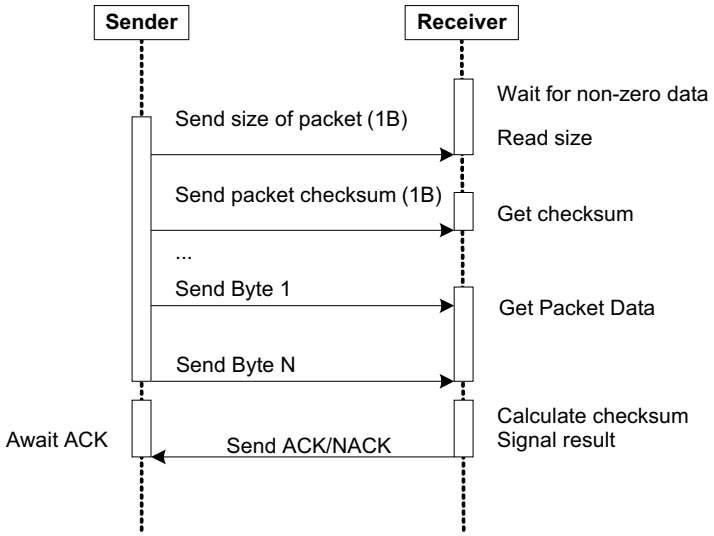

Figure 2. Sequence Chart for Send and Receive Protocol

2.3.1 ACK/NACK

The receiver should respond with an acknowledgment (ACK) or not-acknowledged (NACK) to indicate whether the command was received properly or not. The ACK and NACK signature is shown in Table 7.

Table 7. Acknowledge/Not-Acknowledge Response

Protocol Byte	Value
ACK	0xCC
NACK	0x33

2.4 Interface Configuration

2.4.1 Hardware Pins

The hardware pins used by the ROM bootloader to communicate over UART and SPI are shown in Table 8 and Table 9.

Table 8. Serial Interface Configuration: Evaluation Module Kits

Signal	CC2538	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 9. Serial Interface Configuration: LaunchPad

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 order for the CC2640R2 LaunchPad.

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

2.4.2 UART Configuration

The UART data format is fixed at 8 data bits, no parity, and one stop-bit. The UART bootloader utilizes auto detection of the baud rate (see Section 2.4.3); therefore, any baud rate below the maximum can be used.

- Maximum UART baud rate for CC2538: 460800 baud (1)
- Maximum UART baud rate for CC26x0 and CC26x2: 1.5 M baud

2.4.3 Establishing Communication

The bare minimum needed to establish communication with the bootloader over UART is shown in Figure 3, 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 may not be in bootloader mode, or the baud rate is not supported.

After a connection has been made, any command can be sent to the bootloader. The complete list of bootloader commands can be found in [1], [2], and [3] for CC2538, CC26x0, and CC26x2 respectively.

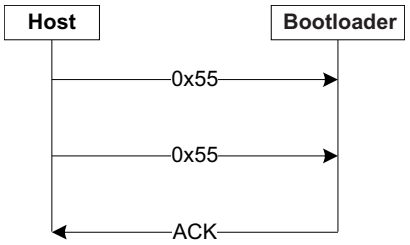

Figure 3. Sequence Chart for Connection Initialization

(1) This data rate number can be doubled if an external 32 MHz 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 2.4.3).

2.4.4 Status Command

To check the status of the bootloader, the CMD_GET_STATUS command can be used; this should be used after erasing or writing the flash memory to be sure that the erase and write were successful before proceeding. The possible status codes for the CMD_GET_STATUS command are shown in Table 10.

Table 10. Possible Status Return Values From Bootloader

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

3 Serial Bootloader Library (SBL)

The SBL is a PC library for Microsoft Windows that implements a host API for communicating with the CC2538, CC26x0, and CC26x2 serial bootloaders. The SBL library project is created in Visual Studio C++ Professional 2015. The serial bootloader library uses Windows API to communicate with the serial COM port and therefore is not cross-platform compatible.

All functions in SBL are synchronous; meaning that the function will not return until ACK or NACK have been received or an error has occurred. Figure 4 demonstrates a sequence chart of the SBL *ping()* function.

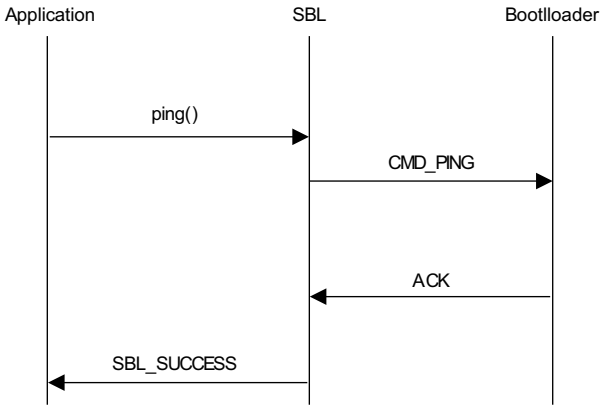

Figure 4. Sequence Chart for ping Function Call

All bootloader commands can be accessed through functions within SBL; which makes it easy to execute 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], [2], and [3].

3.1 SBL Return Values

Each SBL function will return whether the desired operation was successful or not by interpreting the bootloader response. A list of the possible return values from SBL functions and possible causes for them are presented in Table 11.

Table 11. 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 within a given number of tries.
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

An overview of the SBL API is shown in Table 12. API functions that directly map to a bootloader command are marked with an X.

Table 12. SBL Functions

SBL Function Name	Bootloader CMD			Description
	CC2538	CC26x0	CC26x2	
Create	NA	NA	NA	Static function for creating a SBL device object.
calculateCrc32	X	X	X	Calculate CRC32 over the specified range.
connect				Initialize connection with ROM bootloader
enumerate	NA	NA	NA	Static function for enumerating COM ports on PC.
eraseFlashBank		X	X	Erases the entire flash. Not supported by CC2538.
eraseFlashRange	X	X	X	Erase the sectors in the specified range. Uses CMD_SECTOR_ERASE.
ping	X	X	X	Sends ping command.
readDeviceId				Uses CMD_MEMORY_READ to read device ID.
readFlashSize				Uses CMD_MEMORY_READ to read flash size.
readMemory32	X	X	X	Reads 32 bit word from device memory device memory.
readMemory8				Uses CMD_MEMORY_READ to read 8 bit from device memory.
readRamSize				Uses CMD_MEMORY_READ to read RAM size.
readStatus	X	X	X	Reads bootloader status.
reset	X	X	X	Resets device using CMD_RESET.
run	X			Runs the device CPU from the specified address. Not supported by CC26xx.
setCCFG		X	X	Set CC26xx CCFG. Not supported by CC2538.
setXosc	X			Switch to external oscillator. Not supported by CC26xx.
writeFlashRange	X	X	X	Writes FLASH using CMD_DOWNLOAD and CMD_DATA_SEND.
writeMemory32	X	X	X	Writes 32-bit word to device memory using CMD_MEMORY_WRITE.
writeMemory8				Implements 8-bit write to device memory using CMD_MEMORY_READ and CMD_MEMORY_WRITE.

3.2.1 Device-Specific Functions

There are a few commands in the ROM bootloader that differ between CC2538, CC26x0, and CC26x2, this means that there are also a few differences in SBL functions for these devices; these differences are presented in Table 13.

An SBL function that is not supported for the chosen hardware returns the constant SBL_UNSUPPORTED_FUNCTION without doing anything.

Table 13. Device-Specific SBL Functions

SBL Function Name	CC2538	CC26x0	CC26x2	Description
cmdDownloadCrc	Not supported	Not supported	Supported (CMD_DOWNLOAD_CRC)	Not implemented in the CC2538 and CC26x0 bootloader.
eraseFlashBank()	Not supported	Supported (CMD_BANK_ERASE)	Supported (CMD_BANK_ERASE)	This erases all the unprotected flash sectors for CC26x0 and CC26x2; this can be achieved for CC2538 by using eraseFlashRange for the whole Flash memory size.
setCCFG	Not supported	Supported (CMD_SET_CCFG)	Supported (CMD_SET_CCFG)	Not implemented in the CC2538 bootloader.
setXosc	Supported (CMD_SET_XOSC)	Not supported	Not supported	Not implemented in the CC26x0 and CC26x2 bootloader.
run	Supported (CMD_RUN)	Not supported	Not supported	Not implemented in the CC26x0 and CC26x2 bootloader.

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, CC2650, and CC2652R development kits.

SblAppEx is a test application that performs the following actions using the CC2538, CC26x0, or CC26x2 ROM bootloader:

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

A successful execution of the test application should look similar to Figure 5.

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

+---------------------------------------------------------------
| COM ports:
+---------------------------------------------------------------
|Idx    | Description
| 0     | XDS100v3 Class USB Serial Port (COM4)
+---------------------------------------------------------------

Select COM port index: 0
+---------------------------------------------------------------
| Supported devices:
+---------------------------------------------------------------
|Idx    | Device
|  0    | CC2538
|  1    | CC13x0/CC26x0
|  2    | CC2640R2
|  3    | CC13x2/CC26x2
+---------------------------------------------------------------
Select target device: 0
Enable device CC2538 XOSC? (Y/N): y

Connecting (COM4 @ 230400 baud) ...
Trying to set device XOSC.
Device XOSC activated.
100% (902.00ms)
Erasing flash ...
100% (5528.00ms)
Writing flash ...
100% (66620.00ms)
Calculating CRC on device ...
100% (301.00ms)
Comparing CRC ...
OK
Resetting device ...
OK
```

Figure 5. Successful Execution of the CC2538 Example Application

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 6 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 and the other is using a USB-to-UART bridge similar to what is used on the SmartRF06EB [4].

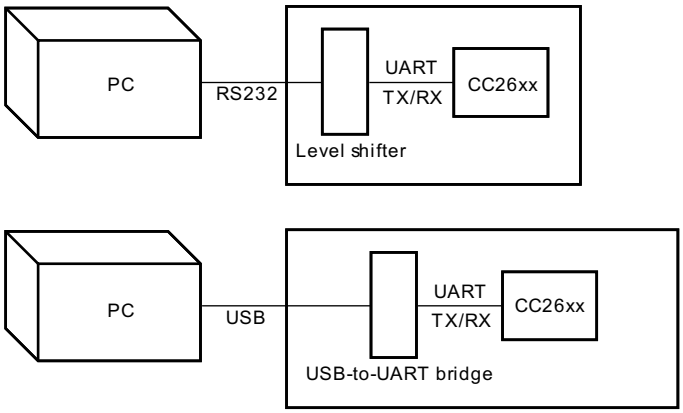

Figure 6. PC to UART Connection

4.1.1 SmartRF06EB Virtual COM Port

SmartRF06EB [4] comes with a built-in support for virtual COM port that can be used together with a CC2538EM [5] or a CC2650EM [6].

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

4.1.1.1 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 7.

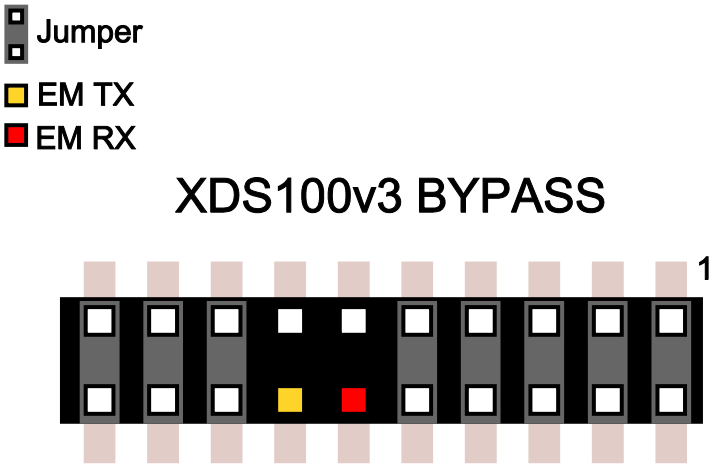

Figure 7. EM TX and RX Pins on XDS100v3 Emulator Bypass Header

4.1.2 LaunchPad Virtual COM Port

The LaunchPads for CC2640R2 [7], CC1312R [8], CC1352R/P [9]/[10], CC2642R [11], and CC2652R [11] comes with built-in support for virtual COM port which is enabled by default. The LaunchPad comes equipped with an XDS110 Class Auxiliary Data Port and XDS110 Class Application/User UART. In this application, the XDS110 Class Application/User UART should be used.

4.1.3 Bootloader Backdoor

The sblAppEx example is written for CC2538 and CC2650 (7x7) Evaluation Modules (EMs) and CC26x2 and CC2640R2 LaunchPads. The application image programmed onto the device triggers the SmartRF06EB or the LaunchPad to blink the LEDs. The firmware image enables the bootloader backdoor, so that the bootloader can be triggered using an IO pin.

The I/O pin used by the application image for opening the bootloader backdoor is shown in Table 14 and Table 15. This I/O pin is connected to the SmartRF06EB SELECT button. To enter the bootloader backdoor, hold down the SELECT button (corresponds to logic '0') while you press the EM reset button on the SmartRF06EB. For the LaunchPad, the bootloader backdoor enable pin must be grounded when the LaunchPad reset button is pressed to enter the bootloader backdoor.

Table 14. Application Example IO Configuration: Evaluation Module Kits

Signal	CC2538	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
Bootloader backdoor enable	PA3	DIO11	DIO9	DIO7	1.14

Table 15. Application Example IO Configuration: LaunchPad

Signal	CC2640R2 (1)	CC26x2R	CC1312R	CC1352x	LaunchPad Pin
UART_RX	DIO2	DIO2	DIO2	DIO12	3 (2)
UART_TX	DIO3	DIO3	DIO3	DIO13	4 (2)
Bootloader backdoor enable	DIO11	DIO11	DIO11	DIO11	18

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

(2) Reverse order for the CC2640R2 LaunchPad.

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. It controls which bootloader commands the SBL is allowed to use, and which firmware image the SblAppEx programs onto the device. The deviceType variable is binary-coded decimal (BCD) of the device name. Table 16 lists the supported device types and corresponding deviceType value.

Table 16. Configuration: deviceType

Device	deviceType Value
CC2538	0x2538
CC13x0/CC26x0	0x2650
CC2640R2	0x2640
CC13x2/CC26x2	0x2652

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, CC26x0, and CC26x2 are covered in Section 2.4.2. 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 it's working with. The Create function supports a single argument, the supported input values are given in Table 16. 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 2.4.2).

The CC2538 ROM bootloader supports switching from the device's internal oscillator 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 has been initialized, the SBL's initCommunication function 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 2.4.2), expecting an ACK from the ROM bootloader. An example of this sequence is shown in Figure 8.

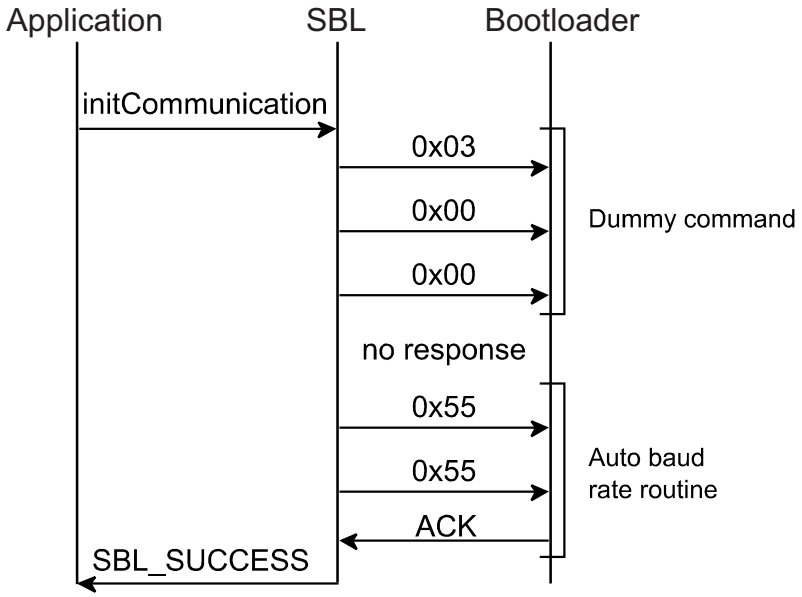

Figure 8. Sequence Chart for initCommunication Function With Uninitialized Bootloader

When the connection has been 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 CC26x0 and CC26x2.

The CC26x0 and CC26x2 CMD_SECTOR_ERASE takes an address parameter and erases the flash sector (4 KB for CC26x0 devices and 8 KB for CC26x2 devices) in which the address is located.

The CC2538 CMD_ERASE command requires a second argument for specifying the erase size. The CC2538 bootloader erases the flash sectors (2 KB) 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 9 shows the sequence chart for a flash erase using the serial bootloader protocol. The last four bytes in the command (datasize) is specific for CC2538. For CC26x0 and CC26x2, the CMD_SECTOR_ERASE command (and consequent CMD_GET_STATUS) must be repeated for each flash sector to erase.

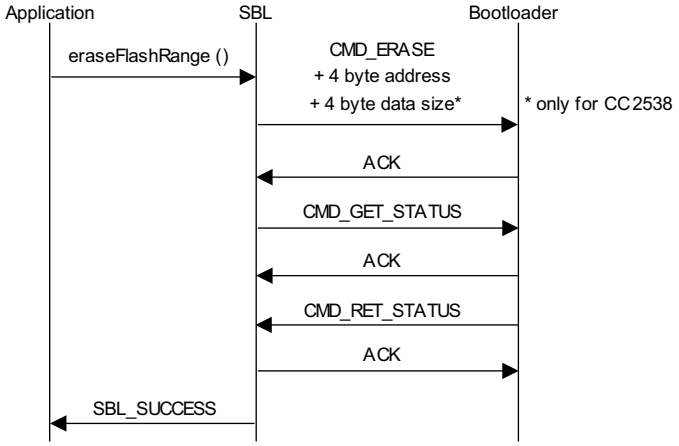

Figure 9. Sequence Chart for Flash Sector Erase

If the whole Flash memory is to be erased on CC26x0 and CC26x2, the CMD_BANK_ERASE command should be used. 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, the SBL function writeFlashRange can be used; this function 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 it to flash, starting at the specified address.

To transfer the data, the CMD_SEND_DATA command is used. A maximum of 252 bytes of data can be transferred per 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 ensure 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 bootloader's internal address pointer is not incremented, allowing the data to be re-transferred.

Figure 10 demonstrates the flash write sequence using the SBL function writeFlashRange.

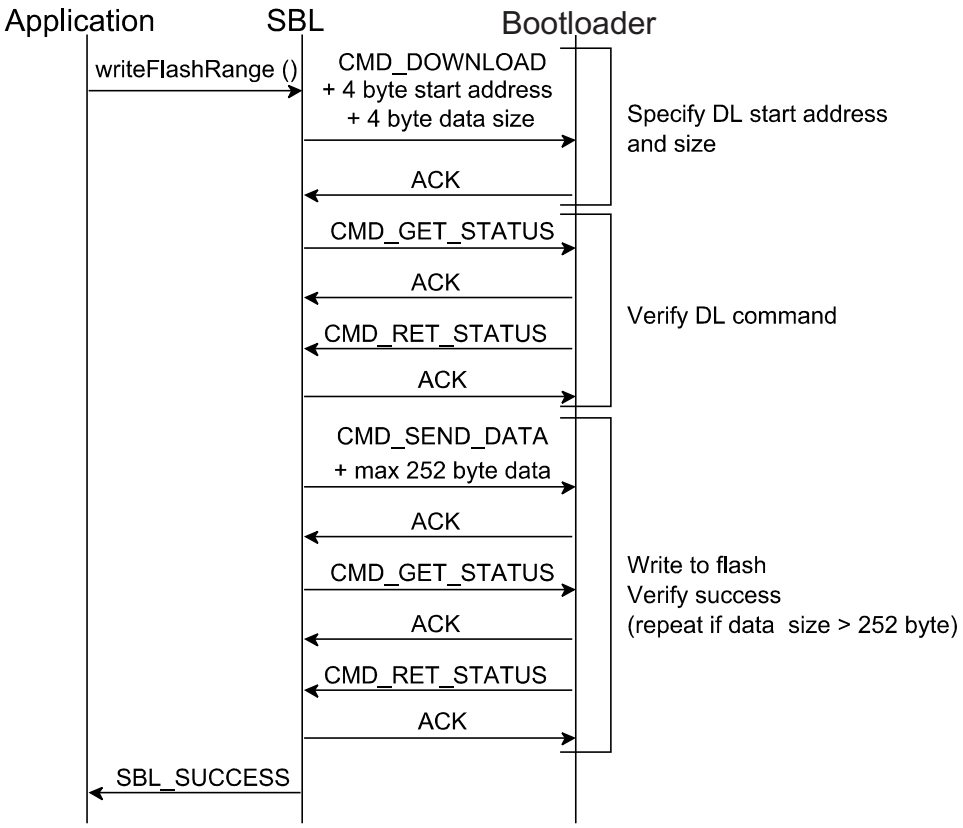

Figure 10. Sequence Chart for Flash Write

4.3.6 Calculate CRC32

To verify that the firmware was successfully programmed into the Flash memory, the SBL function calculateCrc32 can be used 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 CC26x0 and CC26x2, the bootloader also expects a read repeat count. Setting this to 0x00000000 ensures that the data locations are only read once.

The CC2538, CC26x0, and CC26x2 bootloaders uses the CRC-32-IEEE 802.3 with the following polynomial to calculate CRC checksum.

$CRC32_{poly} = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^{8} + x^{7} + x^{5} + x^{4} + x^{2} + x + 1$

An example of how to calculate the checksum using the $CRC32_{poly}$ is implemented in the SBL example project. The sequence chart for the calculateCrc32 function is shown in Figure 11.

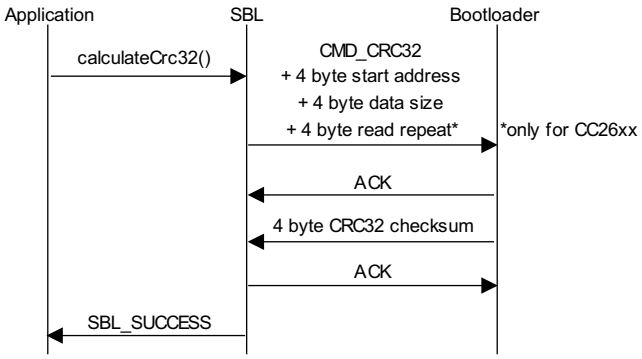

Figure 11. Sequence Chart for CRC32 Command

4.3.7 Reset

To run the firmware after it has been written and verified, the SBL function reset has to be used. The reset function sends the CMD_RESET command to the bootloader to invoke a reset. The connection between the host and the device will break after the CMD_RESET command has been sent and an ACK has been received from the bootloader. The sequence chart for the reset function can be observed in Figure 12.

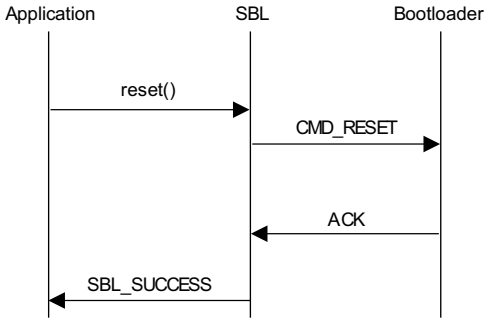

Figure 12. Sequence Chart for SBL Function Reset

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: *SmartRF06 evaluation board (EVM) user's guide*
5. CC2538DK: http://www.ti.com/tool/CC2538DK
6. CC2650DK: http://www.ti.com/tool/CC2650DK
7. CC2640R2 LaunchPad: http://www.ti.com/tool/LAUNCHXL-CC2640R2
8. CC1312R LaunchPad: http://www.ti.com/tool/LAUNCHXL-CC1312R1
9. CC1352R LaunchPad: http://www.ti.com/tool/LAUNCHXL-CC1352R1
10. CC1352P LaunchPad: http://www.ti.com/tool/LAUNCHXL-CC1352P
11. CC26x2R LaunchPad: http://www.ti.com/tool/LAUNCHXL-CC26X2R1

Revision History

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

Changes from A Revision (May 2015) to B Revision	Page
• Added CC13x2/CC26x2 throughout the document.	1
• Changed "page" to "sector" throughout the document.	4
• Update was made in Section 4.	12
• Added new Section 4.1.1.1.	13
• Added section about the Virtual COM Port.	14

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), 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, 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 (www.ti.com/legal/termsofsale.html) or other applicable terms available either on 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.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265