User's Guide
SWPA231–January 2019

CC313x and CC323x SimpleLink™ Wi-Fi® Embedded Programming

The CC313x and CC323x devices are part of the SimpleLink™ microcontroller (MCU) platform, which consists of Wi-Fi®, *Bluetooth*® low energy, Sub-1 GHz, and host MCUs, which all share a common, easy-to- use development environment with a single core software development kit (SDK) and rich tool set. A one-time integration of the SimpleLink™ platform lets you add any combination of devices of the portfolio into your design, allowing 100 percent code reuse when your design requirements change. For more information, visit www.ti.com/SimpleLink.

Contents

List of Figures

List of Tables

Trademarks

1 Introduction

The SimpleLink™ CC313x and CC323x devices are Wi-Fi® and networking devices that provide a comprehensive networking solution for low-cost and low-power microcontrollers (MCU) using a thin driver and simple API set.

Each product with an embedded CC313x or CC323x device onboard must also have a serial flash device connected. The serial flash must be formatted, and at a minimum programmed with the Service Pack, which contains necessary software updates and additional features. For the CC323x, a binary image running on the internal MCU processor must also be programmed.

There are several options for serial flash programming, as follows:

- UniFlash – a PC-based utility offering image creation and programming. Content is programmed using the UART.
- Industrial flash programmer – flashes a complete image prepared with UniFlash directly to the serial flash. Can be applied when no SimpleLink device is attached to the serial flash. Content is programmed using the serial flash SPI lines.
- Over-the-air programming – the serial flash must be formatted in advance. Content is delivered through a network connection.

This application note describes in details additional options that leverage all the features UniFlash has to offer, but without the necessary connected PC. This option is referred to as *Embedded Programming*. To achieve embedded programming, bootloader protocol implemented over UART is described in detail.

The following sections describe the setup, bootloader protocol, and procedure of the embedded programming feature.

2 Embedded Programming Schemes

Several schemes can leverage full image programming over the UART, as follows.

- Embedded programming in production line – there are setups on the production line that do not include the PC. Instead, programmable devices such as the MCU, DSP, or FPGA are used.
- Main external processor (other than the CC323x MCU) – in many cases, CC313x and CC323x devices are just another component in the device enabling network communication. Such devices have main processors that usually control and schedule everything in the system. These cores must have the ability to upgrade and program CC313x or CC323x peripherals.

3 Setup

The UART interface must be connected between the CC313x or CC323x device and the main processor. Only two pins are required, UART TX and UART RX. Flow control is not required.

The common configuration that applies to all chipsets follows:

- Baud rate of 921600 bps
- 8 bits
- No parity
- 1 stop bit

In addition, the nHib/nReset pin is required and it must be have the ability to be temporarily pulled to GND during a reset to make the device go into bootloader mode.

For the CC323x device, an additional SOP1 pin must be pulled up during the device reset to make the device go into bootloader mode.

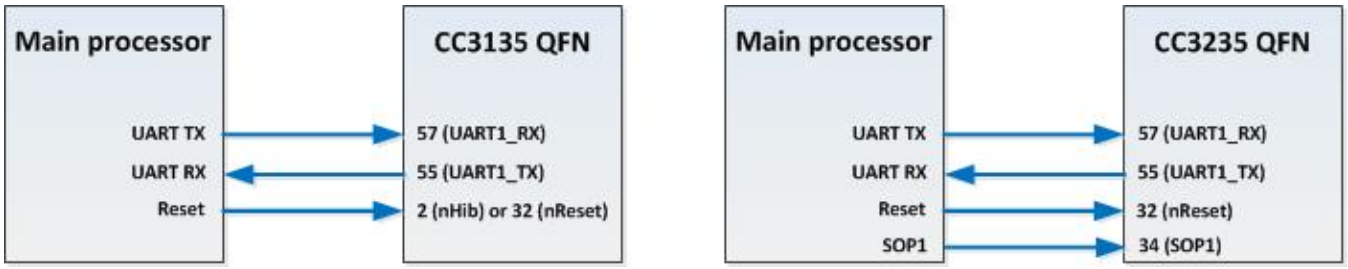

Figure 1. CC313x and CC323x QFN Programming Setup

4 Bootloader Protocol

4.1 Overview

The CC313x and CC323x bootloader protocol is a simple command-response protocol. The protocol is based on the protocol used by the Stellaris® LM Flash Programmer. The commands are serially executed and there are no unsolicited events during the bootloader phase.

4.2 General Message Format

Table 1 provides the general message format.

Table 1. General Message Format

Length (Big Endian)	Checksum	Opcode	Data [optional]
2 bytes	1 byte	1 byte	n bytes

The length includes all fields except the checksum (including the Length field itself), so in general:

Length = Length(Length) + Length(Opcode) + Length(Data) = 3 + Length(Data) (1)

Checksum is a simple hexadecimal addition of the Opcode and Data fields. Checksum is then clipped to occupy the least significant byte only.

Table 2 lists all the commands and responses of the bootloader protocol.

Table 2. Bootloader Commands and Responses

Item	Command or Response	Link
Get Storage List	Command	Section 4.3.1
Get Version Information	Command	Section 4.3.2
Switch UART to APPS MCU	Command	Section 4.3.3
FS Programming	Command	Section 4.3.4
Ack	Response	Section 4.4.1
Nack	Response	Section 4.4.2
Storage List	Response	Section 4.4.3
Version Info	Response	Section 4.4.4

4.3 Commands

4.3.1 Get Storage List

Table 3 lists the Get Storage List command.

Table 3. Get Storage List

	Description
Brief	This command is used to fetch the list of existing storages.
Opcode	0x27
Direction	Host to target
Response	Ack + Storage List response
Format	[USHORT] Length (exclude checksum) [BYTE] Checksum (exclude length) [BYTE] Opcode
Comments	—

4.3.2 Get Version Info

Table 4 lists the Get Version Info command.

Table 4. Get Version Info

	Description
Brief	This command is used to fetch version information of all cores and chip types.
Opcode	0x2F
Direction	Host to target
Response	Ack + Version info response
Format	[USHORT] Length (exclude checksum) [BYTE] Checksum (exclude length) [BYTE] Opcode
Comments	—

4.3.3 Switch UART to APPS MCU

Table 5 lists the Switch UART to APPS MCU command.

Table 5. Switch UART to APPS MCU

	Description
Brief	This command instructs UART lines to get MUX from the CC323x MCU to the CC313x network processor.
Opcode	0x33
Direction	Host to target
Response	Ack
Format	[USHORT] Length (exclude checksum) [BYTE] Checksum (exclude length) [BYTE] Opcode [UINT32] Delay [in ticks]
Comments	Applies only to CC323x. One second is required, hence, the value 26666667 must be used.

4.3.4 FS Programming

Table 6 lists the FS Programming command.

Table 6. FS Programming

	Description
Brief	This command is used for programming a single chunk of a complete image.
Opcode	0x34
Direction	Host to target
Response	Ack + 4 bytes status code
Format	[USHORT] Length (exclude checksum) [BYTE] Checksum (exclude length) [BYTE] Opcode [UINT16] key size [UINT16] chunk size [UINT32] flags [key buffer in bytes] key buffer [chunk buffer in bytes] chunk buffer
Comments	The status code indicates the accumulated number of bytes received. The status for the last chunk must be 0 to indicate successful programming, otherwise, a negative status is returned. Chunks must be serially activated as the NWP assumes orderliness. Chunk size must be the minimum between 4096 and the residue left from the complete image. Flags are for future use and must be 0. If an image is encrypted, a 16-byte key is required.

4.4 Responses

4.4.1 Ack

Table 7 lists the Ack response.

Table 7. Ack

	Description
Brief	This message is sent to acknowledge a packet.
Opcode	0xCC
Direction	Target to host
Response	None
Format	[BYTE] 0 (zero) [BYTE] Opcode
Comments	—

4.4.2 Nack

Table 8 lists the Nack response.

Table 8. Nack

	Description
Brief	This message is sent to indicate an acknowledge error.
Opcode	0x33
Direction	Target to host
Response	None
Format	[BYTE] 0 (zero) [BYTE] Opcode
Comments	—

4.4.3 Storage List

Table 9 lists the Storage List response.

Table 9. Storage List

	Description
Brief	This message is sent as a response to the Get Storage List command.
Opcode	N/A
Direction	Target to host
Response	None
Format	[BYTE] Storage list bitmap: • FLASH_DEV_BIT (0x02) • SFLASH_DEV_BIT (0x04) • SRAM_DEV_BIT (0x80)
Comments	—

4.4.4 Version Info

Table 10 lists the Version Information response.

Table 10. Version Info

	Description
Brief	This message is sent as a response to the Get Version Info command.
Opcode	N/A
Direction	Target to host
Response	Ack
Format	[USHORT] Length (exclude checksum) [BYTE] Checksum (exclude length) [4 BYTES] Bootloader version (x.x.x.x) [4 BYTES] NWP Version [4 BYTES] MAC Version [4 BYTES] PHY Version [4 BYTES] Chip type [4 BYTES] Reserved [4 BYTES] Reserved
Comments	—

5 Embedded Programming Procedure

5.1 Overview

The following sections describe the bootloader commands and responses, as well as the ordered steps during image programming.

Because content and timing are important during the procedure, all steps were captured for reference. Saleae is the logic used. This utility is free for use and can be downloaded from https://www.saleae.com/downloads.

5.2 High-Level Flow Diagram

Figure 2 shows the programming flow in high level. For a low-level description, see the following subsections.

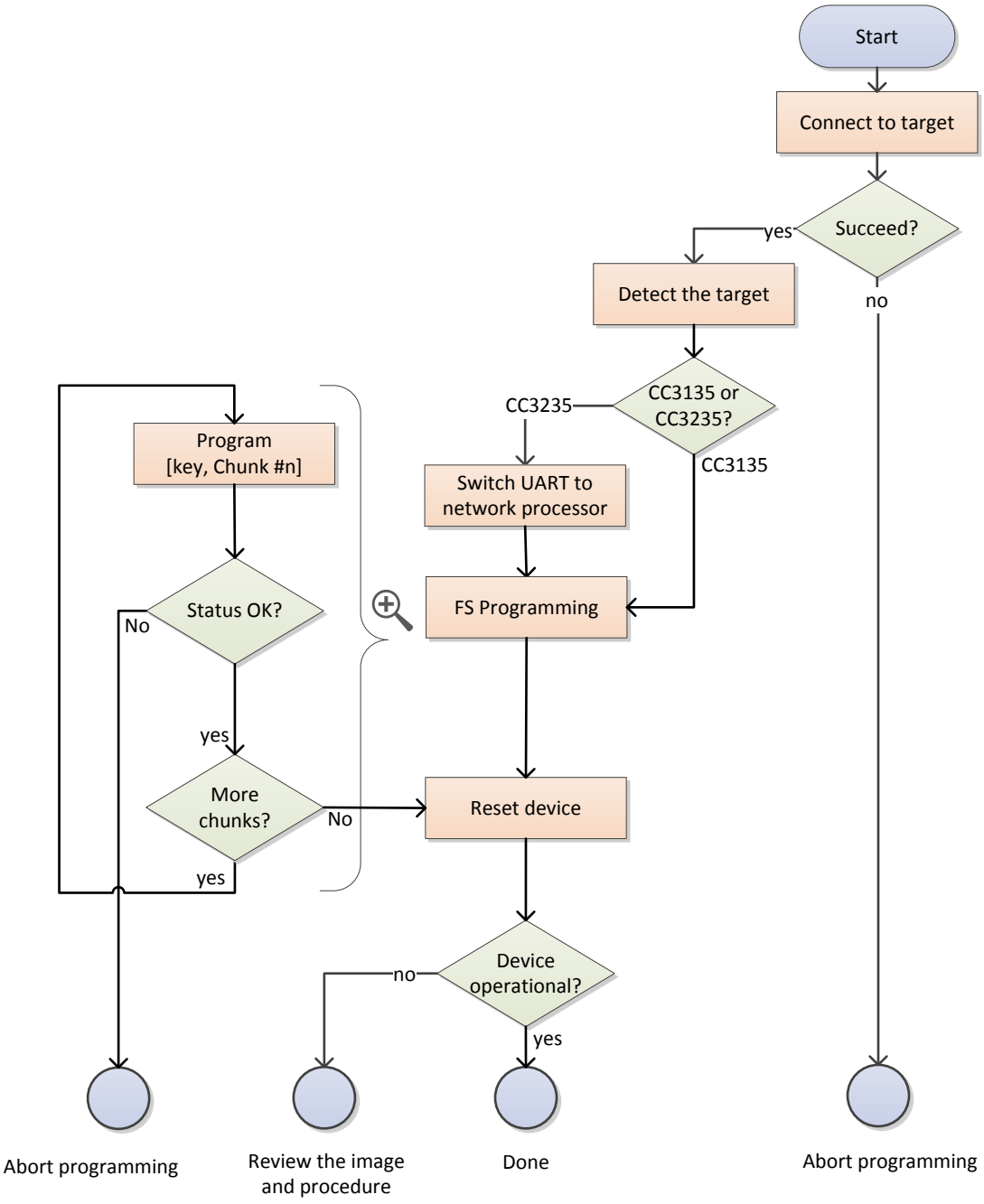

Figure 2. High-Level Programming Flow

5.3 Image Programming in Detail

5.3.1 Step 1: Target Connection

Before commands can be sent to the device, it must go into bootloader mode. The procedure for connecting to the target in bootloader phase follows:

1. Flush the UART RX line (CC313x or CC323x UART TX line).
2. Send a break signal (sending continuous spacing values, no start or stop bits) on the CC313x or CC323x UART RX line. The CC313x or CC323x device must sense this break signal during power up.
3. Power on the device (or reset it if it is already up and running).
4. The CC313x or CC323x device sends an acknowledgment indication. The acknowledgment indication is described in Ack.
5. On receiving the acknowledgment indication from the CC313x or CC323x device, the main processor stops sending the break signal and flushes the UART lines.

NOTE: At this point, the CC313x or CC323x device is ready to receive any command. The main processor has 5 seconds to send any command. Failing to do so before the time-out expires results in the CC313x or CC323x device initializing normally (aborting bootloader mode).

6. The main processor sends the Get Storage List command.
7. The CC313x or CC323x device responds with an Ack followed by a 1-byte storage list bitmap.

Figure 3 shows the Get Storage List command.

Figure 3. Get Storage List

5.3.2 Step 2: Target Detection

It is essential to determine whether the connected device is a CC313x or CC323x. It is important because additional steps are required for the CC323x. The provided information indicates whether the device is a CC323x (nonsecured), CC323xS, or CC323xSF (flash device).

The procedure for detecting the target follows:

1. The main processor sends the Get Version Info command.
2. The CC313x or CC323x device responds with an Ack, followed by the Version Info response. The first byte of the Chip type field must be tested.

```
If (chip type & 0x10)        //the device is CC3235 flavor//
If (chip type == 0x10)       //the device is CC3235 (nonsecured)//
If (chip type == 0x18)       //the device is CC3235S (secured ROM)//
If (chip type == 0x19)       //the device is CC3235SF (secured flash)//
else                         //the device is CC3235 flavor//
```

3. The main processor responds with an Ack.

Figure 4 shows the Get Version Info command and the Version Info response.

Figure 4. Get Version Info

5.3.3 Step 3: MUX UART to the Network Processor

If the device is a CC323x, it is required to MUX the UART lines from the internal application processor (Arm® Cortex®-M4) to the network processor.

The procedure for switching UART lines in CC323x case follows:

1. The main processor sends the Switch UART to APPS MCU command. The user must provide the delay until the network processor is ready. One second is sufficient and should be used. During this time, the UART lines are switched, and the network processor is rebooted into bootloader mode. The reset is internal so the user does not need to externally apply it.
2. The CC323x device responds with an Ack. This is the command response.
3. The main processor should send a break signal (sending continuous Spacing values [no Start or Stop bits]) on the CC323x UART RX line. The network processor must sense this break signal during power up.
4. Because there are cases in which the break signal is missed or the Ack packet is being missed, TI recommends sending the break signal up to four times. Follow the following pseudocode for details.
5. The network processor responds with an Ack. This response is an indication that the network processor sensed the break signal and entered bootloader mode.
6. The main processor deasserts the break signal.

```
For i=1..4
set BREAK sleep 100mSec read ACK
unset BREAK
If successful break
Else continue
```

Figure 5 shows the Switch UART to APPS MCU command and the Ack command response.

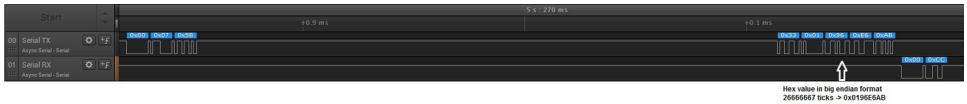

Figure 5. Switch UART Lines to APPS MCU Command

Figure 6 shows the BREAK assertion and deassertion followed by an Ack.

Figure 6. Switch UART to APPS MCU Procedure

5.3.4 Step 4: FS Programming

The next step is the actual programming of the image to the SFLASH. Programming is applied in chunks of 4096 bytes. The image can be either unencrypted or encrypted with a 16-byte symmetric key.

- If the image is unencrypted, the key size is irrelevant and uses a length of 0.
- If the image is encrypted, the key size must be 16 bytes. The key buffer precedes the data chunk.

In both cases, flags are for future use and must be 0. The procedure for programming the SFLASH follows:

1. The main processor sends the FS Programming command in chunks of 4096 bytes. The user must provide the following elements in order:
    1. Key size (in bytes for an encrypted image) must be 16, otherwise 0.
    2. Chunk size (in bytes) must be 4096, except for the last chunk, which may be smaller according to the residue from the total size.
    3. Flags must be 0.
    4. Key buffer
    5. Data buffer
2. The CC313x or CC323x device responds with an Ack followed by a 4-byte response indicating the accumulated number of bytes received. The status for the last chunk must be 0 to indicate successful programming, otherwise, a negative status is returned.
3. Steps 1 and 2 repeat until the entire image is programmed.
4. The image gets extracted and the file system is created.

Figure 7 shows the programming procedure in a zoomed-out pane. The last status code is 0 to indicate a successful programming and is delayed in time to reflect the period of image extraction.

Figure 7. FS Programming (Zoomed Out)

Figure 8 shows the programming procedure in a zoomed-in pane for an unencrypted image.

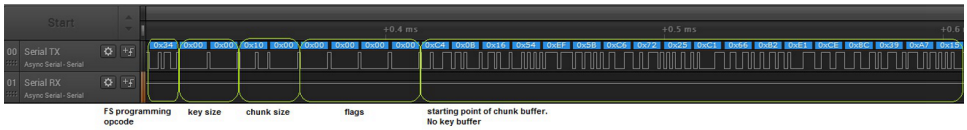

Figure 8. FS Programming of Unencrypted Image (Zoom In)

Figure 9 shows the programming procedure in a zoomed-in pane for an encrypted image.

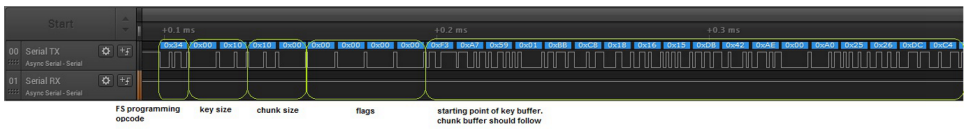

Figure 9. FS Programming of Encrypted Image (Zoomed In)

5.3.5 Step 5: Device Reset

The final step is the device reset.

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