*User's Guide*

# *CC3100 and CC3200 SimpleLink™ Wi-Fi® Embedded Programming*

## Table of Contents



## List of Figures



## List of Tables



## Trademarks

SimpleLink™ is a trademark of Texas Instruments.
Wi-Fi® is a registered trademark of Wi-Fi Alliance.

## 1 Introduction

The SimpleLink™ CC3100 and CC3200 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 CC3100 or CC3200 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 CC3200, 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 CC3200 MCU) – in many cases, CC3100 and CC3200 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 CC3100 or CC3200 peripherals.

## 3 Setup

The UART interface must be connected between the CC3100 or CC3200 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 CC3200 device, an additional SOP1 pin must be pulled up during the device reset to make the device go into bootloader mode.


**Figure 3-1. CC3200 and CC3200 QFN Programming Setup**

# 4 Bootloader Protocol

## 4.1 Overview

The CC3100 and CC3200 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 4-1 provides the general message format.

**Table 4-1. 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:

$$\text{Length} = \text{Length(Length)} + \text{Length(Opcode)} + \text{Length(Data)} = 3 + \text{Length(Data)} \quad (1)$$

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

Table 4-2 lists all the commands and responses of the bootloader protocol.

**Table 4-2. Bootloader Commands and Responses**

| Item | Command or Response | Link |
|---|---|---|
| Get Storage List | Command | Section 4.3.2 |
| Raw Storage Write | Command | Section 4.3.3 |
| Get Version Information | Command | Section 4.3.4 |
| Raw Storage Erase | Command | Section 4.3.5 |
| Switch UART to APPS MCU | Command | Section 4.3.8 |
| Ack | Response | Section 4.4.1 |
| Version Info | Response | Section 4.4.6 |

## 4.3 Commands

### 4.3.1 Get Status

Table 4-3 lists the Get Status command.

**Table 4-3. Get Status**

| | Description |
|---|---|
| Brief | This command returns the status of the last command executed.<br>Typically, this command must be invoked after every command sent to ensure that the previous command was successful or, if unsuccessful, to properly respond upon failure.<br>The bootloader responds by sending a 1-byte packet containing the current status code. |
| Opcode | 0x23 |
| Direction | Host to target |
| Response | Ack + Last Status response |
| Format | [USHORT] Length (exclude checksum)<br>[BYTE] Checksum (exclude length)<br>[BYTE] Opcode |
| Comments | Host responds back with Ack (Ack + Last Status response). |

### 4.3.2 Get Storage List

Table 4-4 lists the Get Storage List command.

**Table 4-4. 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)<br>[BYTE] Checksum (exclude length)<br>[BYTE] Opcode |
| Comments | — |

### 4.3.3 Raw Storage Write

Table 4-5 lists the Raw Storage Write command.

**Table 4-5. Raw Storage Write**

| | Description |
|---|---|
| Brief | This command triggers sending a chunk of raw storage data specified by StorageID, starting from a position specified by Offset and with a size specified by Length. |
| Opcode | 0x2D |
| Direction | Host to target |
| Response | Ack |
| Format | [USHORT] Length (exclude checksum)<br>[BYTE] Checksum (exclude length)<br>[BYTE] Opcode<br>[UINT32] StorageID<br>[UINT32] Offset (starting offset relative to the start of storage, in bytes)<br>[UINT32] Length (amount of bytes to write)<br>[BYTE Stream] Data |
| Comments | The chunk is 4096 bytes and the Length must be smaller than (chunk_size-16).<br>The SRAM storage ID is 0x0.<br>The serial flash storage ID is 0x2. |

### 4.3.4 Get Version Info

Table 4-6 lists the Get Version Info command.

**Table 4-6. 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)<br>[BYTE] Checksum (exclude length)<br>[BYTE] Opcode |
| Comments | — |

### 4.3.5 Raw Storage Erase

Table 4-7 lists the Raw Storage Erase command.

**Table 4-7. Raw Storage Erase**

| | Description |
|---|---|
| Brief | This command erases the specified blocks (making them writable) from storage specified by StorageID. NumOfBlocks specifies the amount of blocks to erase and Offset specifies the offset of the first blocks from the start of the device. |
| Opcode | 0x30 |
| Direction | Host to target |
| Response | Ack |
| Format | [USHORT] Length (exclude checksum)<br>[BYTE] Checksum (exclude length)<br>[BYTE] Opcode<br>[UINT32] StorageID<br>[UINT32] Offset (in blocks, relative to start of device)<br>[UINT32] NumOfBlocks (number of blocks to erase) |
| Comments | The SRAM storage ID is 0x0.<br>The serial flash storage ID is 0x2. |

### 4.3.6 Get Storage Info

Table 4-8 lists the Get Storage Info command.

**Table 4-8. Get Storage Info**

| | Description |
|---|---|
| Brief | This command is used to fetch storage information of a requested storage ID. |
| Opcode | 0x31 |
| Direction | Host to target |
| Response | Ack + Storage Info response |
| Format | [USHORT] Length (exclude checksum)<br>[BYTE] Checksum (exclude length)<br>[BYTE] Opcode<br>[UINT32] StorageID |
| Comments | The SRAM storage ID is 0x0.<br>The serial flash storage ID is 0x2.<br>Host responds back with Ack on the (Ack + Storage Info) response. |

### 4.3.7 Execute From RAM

Table 4-9 lists the Execute from RAM command.

**Table 4-9. Execute from RAM**

| | Description |
|---|---|
| Brief | This command is used to execute a preloaded program in the SRAM. |
| Opcode | 0x32 |
| Direction | Host to target |
| Response | Ack followed by another Ack |
| Format | [USHORT] Length (exclude checksum)<br>[BYTE] Checksum (exclude length)<br>[BYTE] Opcode |
| Comments | First Ack indicates a complete command and the second Ack indicates that initialization has completed. |

### 4.3.8 Switch UART to APPS MCU

Table 4-10 lists the Switch UART to APPS MCU command.

**Table 4-10. Switch UART to APPS MCU**

| | Description |
|---|---|
| Brief | This command instructs UART lines to get MUX from the CC3200 MCU to the CC3100 network processor. |
| Opcode | 0x33 |
| Direction | Host to target |
| Response | Ack |
| Format | [USHORT] Length (exclude checksum)<br>[BYTE] Checksum (exclude length)<br>[BYTE] Opcode<br>[UINT32] Delay [in ticks] |
| Comments | Applies only to CC3200. One second is required, hence, the value 26666667 must be used. |

### 4.3.9 FS Programming

Table 4-11 lists the FS Programming command.

**Table 4-11. 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)<br>[BYTE] Checksum (exclude length)<br>[BYTE] Opcode<br>[UINT16] key size<br>[UINT16] chunk size<br>[UINT32] flags<br>[key buffer in bytes] key buffer<br>[chunk buffer in bytes] chunk buffer |
| Comments | The status code indicates the accumulated number of bytes received.<br>The status for the last chunk must be 0 to indicate successful programming, otherwise, a negative status is returned.<br>Chunks must be serially activated as the NWP assumes orderliness.<br>Chunk size must be the minimum between 4096 and the residue left from the complete image.<br>Flags are for future use and must be 0.<br>If an image is encrypted, a 16-byte key is required. |

## 4.4 Responses

### 4.4.1 Ack

Table 4-12 lists the Ack response.

**Table 4-12. Ack**

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

### 4.4.2 Nack

Table 4-13 lists the Nack response.

**Table 4-13. Nack**

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

### 4.4.3 Last Status

Table 4-14 lists the Last Status response.

**Table 4-14. Last Status**

| | Description |
|---|---|
| Brief | This message is sent to indicate last operation status. |
| Opcode | N/A |
| Direction | Target to host |
| Response | Ack from host |
| Format | [4 BYTES] Status code |
| Comments | The data includes 2 bytes for length, 1 byte of checksum, and 1 byte for status. |

### 4.4.4 Storage List

Table 4-15 lists the Storage List response.

**Table 4-15. 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:<br>• FLASH_DEV_BIT (0x02)<br>• SFLASH_DEV_BIT (0x04)<br>• SRAM_DEV_BIT (0x80) |
| Comments | — |

### 4.4.5 Storage Info

Table 4-16 lists the Storage Information response.

**Table 4-16. Storage Info**

| | Description |
|---|---|
| Brief | This message is sent as a response to the Get Storage Info command. |
| Opcode | N/A |
| Direction | Target to host |
| Response | Ack from Host |
| Format | [USHORT] Length (exclude checksum)<br>[BYTE] Checksum (exclude length)<br>[2 BYTES] block size<br>[2 BYTES] number of blocks<br>[4 BYTES] reserved |
| Comments | — |

### 4.4.6 Version Info

Table 4-17 lists the Version Information response.

**Table 4-17. 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)<br>[BYTE] Checksum (exclude length)<br>[4 BYTES] Bootloader version (x.x.x.x)<br>[4 BYTES] NWP Version<br>[4 BYTES] MAC Version<br>[4 BYTES] PHY Version<br>[4 BYTES] Chip type<br>[4 BYTES] Reserved<br>[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.

The image must first be created using UniFlash. Detailed instructions on how to create an image are listed in the *UniFlash CC3120 and CC3220 SimpleLink™ Wi-Fi® and IoC Solution ImageCreator and Programming Tool User's Guide* under *Image Creation and Programming Using UniFlash*.

## 5.2 High-Level Flow Diagram

Figure 5-1 shows the programming flow in high level. For a low-level description, see the following subsections.

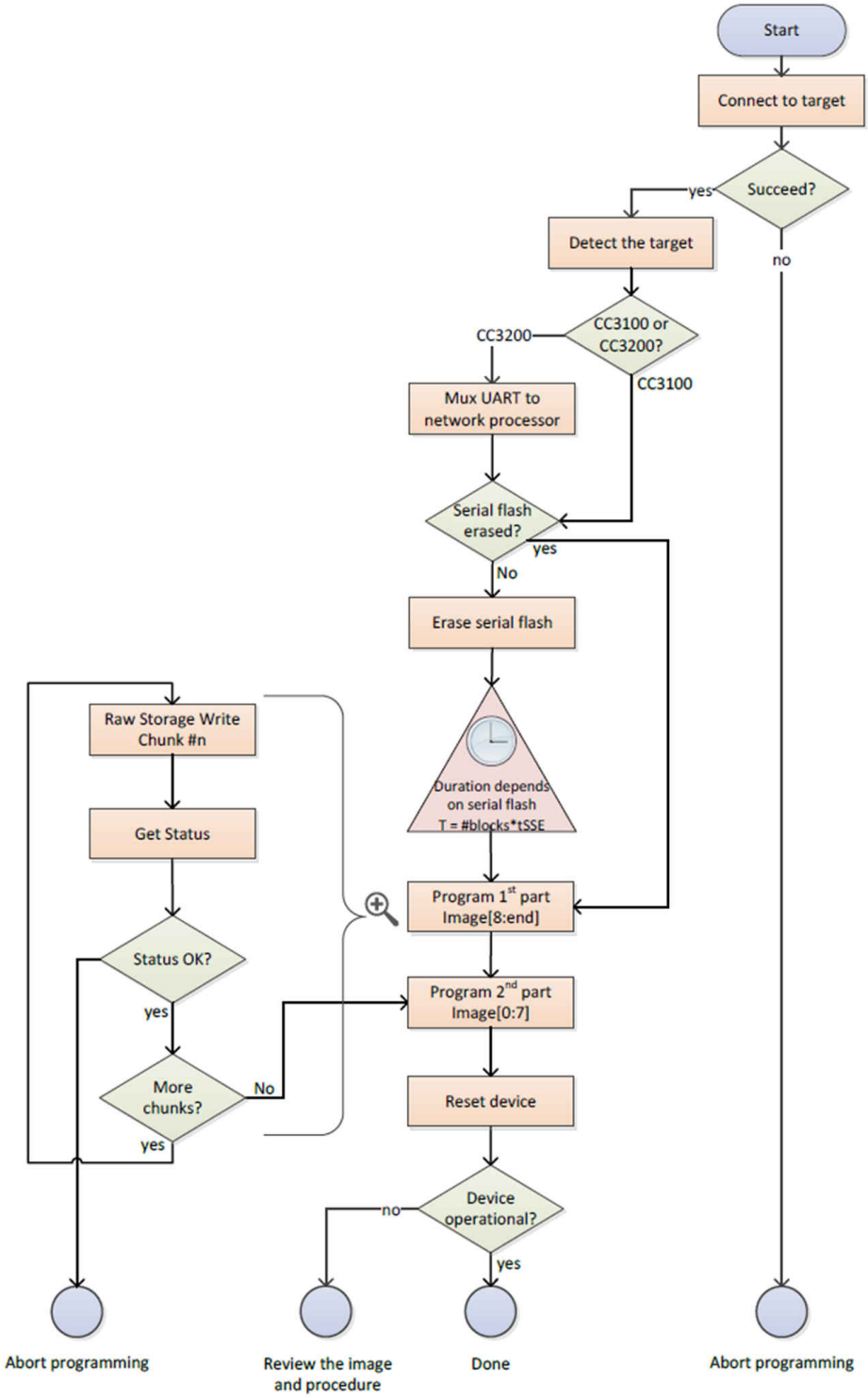


**Figure 5-1. High-Level Programming Flow**

## 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 (CC3100 or CC3200 UART TX line).
2. Send a break signal (sending continuous spacing values, no start or stop bits) on the CC3100 or CC3200 UART RX line. The CC3100 or CC3200 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 CC3100 or CC3200 device sends an acknowledgment indication. The acknowledgment indication is described in Ack.
5. On receiving the acknowledgment indication from the CC3100 or CC3200 device, the main processor stops sending the break signal and flushes the UART lines.

**Note**

At this point, the CC3100 or CC3200 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 CC3100 or CC3200 device initializing normally (aborting bootloader mode).

6. The main processor sends the Get Storage List command.
7. The CC3100 or CC3200 device responds with an Ack followed by a 1-byte storage list bitmap.

Figure 5-2 shows the Get Storage List command.

**Figure 5-2. Get Storage List**

### 5.3.2 Step 2: Target Detection

It is essential to determine whether the connected device is a CC3100 or CC3200. It is important because additional steps are required for the CC3200. The provided information indicates whether the device is a CC3200 (nonsecured), CC3200S, or CC3200SF (flash device).

The procedure for detecting the target follows:

1. The main processor sends the Get Version Info command.
2. The CC3100 or CC3200 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 CC3200 flavor
else //the device is CC3100 flavor
```

3. The main processor responds with an Ack.

Figure 5-3 shows the Get Version Info command and the Version Info response.

**Figure 5-3. Get Version Info**

### 5.3.3 Step 3: MUX UART to the Network Processor

If the device is a CC3200, 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 CC3200 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 CC3200 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 CC3200 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 5-4 shows the Switch UART to APPS MCU command and the Ack command response.

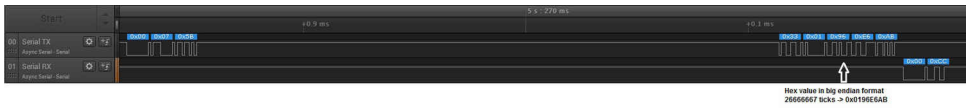


**Figure 5-4. Switch UART Lines to APPS MCU Command**

Figure 5-5 shows the BREAK assertion and deassertion followed by an Ack.


**Figure 5-5. Switch UART to APPS MCU Procedure**

### 5.3.4 Step 4: Get SRAM Storage Info

To introduce some fixes to the ROM bootloader, it is necessary to download patches to the SRAM. The procedure for getting information of SRAM storage follows:

1. The main processor sends the Get Storage Info command. The user must provide the storage ID for the SRAM.
2. The CC3100 or CC3200 device responds with an Ack followed by the Storage Info response. The response includes the block size and number of blocks for the SRAM.
3. The main processor responds with an Ack.

Figure 5-6 shows the Get Storage Info command followed by an Ack and Storage Info Response.


**Figure 5-6. Get Storage Info**

### 5.3.5 Step 9: Raw Storage Erase – SFLASH

Before downloading the patches to the SFLASH, it is essential to erase the relevant memory location in the SFLASH. The procedure for erasing the SFLASH follows:

1. The main processor sends the Raw Storage Erase command. The user must provide the storage ID for the SFLASH, offset in blocks, and number of blocks to erase. In this case, erase two blocks starting from offset 33.
2. The CC3100 or CC3200 device responds with an Ack.
3. The main processor responds with an Ack.
4. The main processor sends the Get Status command.
5. The CC3100 or CC3200 device responds with an Ack followed by the Last Status response. Only the fourth byte should be inspected; 0x40 means success whereas 0x4A indicates an error.
6. The main processor sends an Ack response.

Figure 5-7 shows the Raw Storage Erase command.


**Figure 5-7. Raw Storage Erase to SFLASH**

### 5.3.6 Step 10: Raw Storage Write – SFLASH

Programming the SFLASH is done in chunks. The chunk size is 4096 bytes (4KB).

It is important to note that the image must be programmed in two phases. First, the image is programmed from offset 8 to the end. Then, program the first 8 bytes. This is mandatory since the first 8 bytes is a header that indicates to the bootloader that the image is valid. If the programming procedure is aborted for any reason such as an unexpected power failure the presence of a valid header will make the device unresponsive, as it will try to extract the corrupt image. As such, the 8 byte header must be programmed last.

1. The main processor sends the Raw Storage Write command in 4KB chunks. The first chunk starts at offset 8.
2. The CC3100 or CC3200 device responds with an Ack.
3. The main processor sends the Get Status command.
4. The CC3100 or CC3200 device responds with an Ack followed a 4-byte response. Only the 4th byte should be inspected; 0x40 means success whereas 0x4A indicates an error.
5. The main processor sends an Ack response.
6. Repeat Steps 1 to 5 as needed until the data is completely written.
7. The first 8 bytes of data are written at offset 0 using steps 1-5 to complete the programming of the entire contents of the image.

Figure 5-8 shows the SFLASH programming procedure in a zoomed-out pane. As observed, the procedure is divided into three separate programming instances.

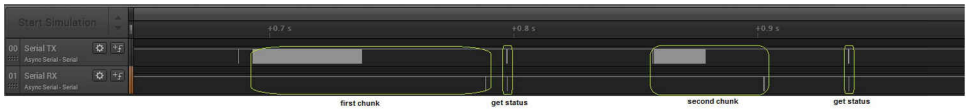


**Figure 5-8. Raw Storage Write to SFLASH (Zoomed Out)**

### 5.3.7 Step 11: 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:
   a. Key size (in bytes for an encrypted image) must be 16, otherwise 0.
   b. Chunk size (in bytes) must be 4096, except for the last chunk, which may be smaller according to the residue from the total size.
   c. Flags must be 0.
   d. Key buffer
   e. Data buffer
2. The CC3100 or CC3200 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 5-9 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.

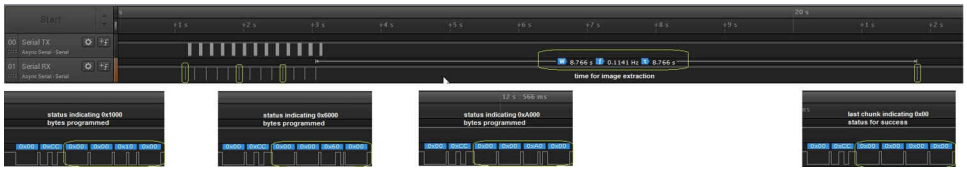


**Figure 5-9. FS Programming (Zoomed Out)**

Figure 5-10 shows the programming procedure in a zoomed-in pane for an unencrypted image.


**Figure 5-10. FS Programming of Unencrypted Image (Zoom In)**

Figure 5-11 shows the programming procedure in a zoomed-in pane for an encrypted image.

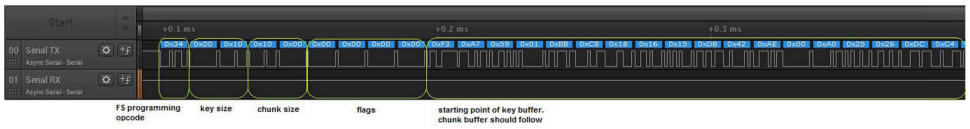


**Figure 5-11. FS Programming of Encrypted Image (Zoomed In)**

### 5.3.8 Step 12: Device Reset

The final step is the device reset.

# 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 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