



Skyler Baumer

**ABSTRACT**

The goal of this application note is to detail an example project that provides a flash-based implementation to demonstrate a Firmware-Over-The-Air (FOTA) upgrade over UART on F29H85x. The implementation enables FOTA upgrades for both CPU1 and CPU3 while application interrupts are still being serviced. Once the firmware upgrade is complete and a device reset has been issued, a bank swap is triggered and the firmware received over UART executes in the newly activated flash address space.

**Table of Contents**

<b>1 Introduction</b>	<b>3</b>
1.1 Hardware Security Module	4
1.2 Flash Programming Fundamentals	5
1.3 High-Level Flow	5
1.4 Flow Chart	6
<b>2 Flash-Based UART SBL with FOTA</b>	<b>8</b>
2.1 Implementation	8
2.2 Triggering a Bank Swap	8
<b>3 FOTA_Example_Application</b>	<b>11</b>
3.1 led_blinky_cpu1.c	11
3.2 Combining the Flash-Based SBL with the FOTA_Example_Application	15
3.3 Adding a CPU3 Application	17
<b>4 Host Application: UART Flash Programmer</b>	<b>18</b>
4.1 Overview	18
<b>5 Example Usage</b>	<b>20</b>
5.1 Loading the SBL onto the Device	20
5.2 Example UART Loading Process	21
<b>6 FAQ</b>	<b>25</b>
6.1 General	25
6.2 Application Load	25
<b>7 Summary</b>	<b>26</b>
<b>8 References</b>	<b>26</b>

**List of Figures**

Figure 1-1. A/B Swapping on F29H85x	3
Figure 1-2. Flow Chart	7
Figure 2-1. BANKMGMT Region	9
Figure 2-2. BANKMAP Before Swapping	9
Figure 2-3. Inactive BANKMGMT Updated for Swap	10
Figure 2-4. Active BANKMGMT Region	10
Figure 2-5. BANKMAP Swap	10
Figure 3-1. CPU Timer ISR Configuration in SysConfig	11
Figure 3-2. CPU Timer ISR	12
Figure 3-3. UART Configuration in SysConfig	12
Figure 3-4. UART RX ISR	13
Figure 3-5. Main Loop	13
Figure 3-6. Addresses to Branch to in SBL to Perform FOTA Upgrade	14
Figure 3-7. Linker CMD Memory Sections and Output Section Creation	14
Figure 3-8. Output Section Creation in SBL Project	15

Figure 3-9. Creating CPU1 FOTA Output Section.....	15
Figure 3-10. Linker CMD File in SBL.....	15
Figure 3-11. FOTA_Example_Application Post-Build Step.....	16
Figure 3-12. SBL Post-Build Step.....	16
Figure 3-13. CPU1 Code Required to Take CPU3 Out of Reset.....	17
Figure 5-1. FOTA Enable Checkbox in Flash Plugin.....	20
Figure 5-2. SSU_GEN_REG.BANKMODE Showing Bank Mode 0.....	21
Figure 5-3. Host Output After Sending Kernel.....	22
Figure 5-4. Kernel Running in RAM.....	22
Figure 5-5. Host Output of DFU CPU1 Command.....	23
Figure 5-6. Device Successfully Boots to Flash.....	23
Figure 5-7. Host Programmer Output of DFU CPU1 Command.....	24
Figure 5-8. FLC1.B2 and FLC1.B3 After Firmware Programmed to Inactive Flash Region.....	24

## List of Tables

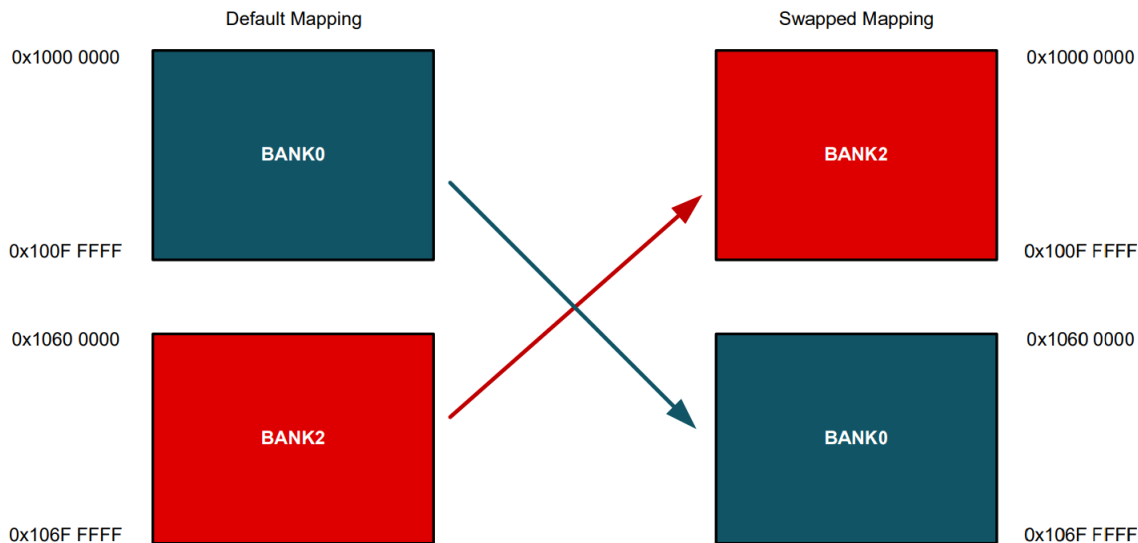
Table 1-1. Device Security State.....	4
Table 2-1. Flash BANKMGMT Region Address Mapping (BANKMODE = 1).....	9
Table 4-1. Commands.....	18

## Trademarks

Code Composer Studio™ is a trademark of Texas Instruments.  
All trademarks are the property of their respective owners.

## 1 Introduction

On the F29H85x, FOTA is supported by using an A/B swapping mechanism. An A/B swap is performed by changing the physical flash banks mapped to the active and inactive address space in flash. The active address space is the flash memory that is executed during runtime while the inactive address space is reserved for programming incoming firmware that can be activated after the A/B swap. [Figure 1-1](#) shows an example of how flash banks are mapped to different address spaces during an A/B swap.



**Figure 1-1. A/B Swapping on F29H85x**

The flash-based UART secondary boot-loader with FOTA example was created to demonstrate this capability of the F29H85x. Hereafter, the example is referred to as SBL.

A key consideration when implementing FOTA on F29H85x is that only bank mode 1 and bank mode 3 support the A/B swapping mechanism. Bank mode 1 supports A/B swapping for only CPU1 and bank mode 3 supports A/B swapping for both CPU1 and CPU3. For more details about bank modes and the associated memory maps, refer to the [F29H85x Technical Reference Manual \(TRM\)](#).

The SBL supports the following features when the device is in bank mode 1:

1. Non-secure CPU1 FOTA upgrades
2. Secure CPU1 FOTA upgrades
  - a. *Device must be in HS-SE state*
3. Secure HSM FOTA upgrades
  - a. *Device must be in HS-SE state*

When the device is bank mode 3, the SBL supports the following features:

1. Non-secure CPU1 FOTA upgrades
2. Non-secure CPU3 FOTA upgrades
3. Secure CPU1 FOTA upgrades
  - a. *Device must be in HS-SE state*
4. Secure CPU3 FOTA upgrades
  - a. *Device must be in HS-SE state*
5. Secure HSM FOTA upgrades
  - a. *Device must be in HS-SE state*

To convert an HS-FS device to the HS-SE state, refer to the [Serial Flash Programming of F29H85x application note](#). This project includes 4 different build configurations to be selected based on the desired features.

BANKMODE_1	Non-Secure CPU1 FOTA Upgrades
BANKMODE_1_CP <i>Device must be in HS-SE state</i>	Secure CPU1 and HSM FOTA Upgrades
BANKMODE_3	Non-Secure CPU1 and CPU3 FOTA Upgrades
BANKMODE_3_CP <i>Device must be in HS-SE state</i>	Secure CPU1, CPU3, and HSM FOTA Upgrades

### CAUTION

**This document assumes no SSU configuration.**

### CAUTION

To successfully build and use this example, [F29H85x SDK 1.01.00.00](#) (or newer) must be installed. For secure firmware upgrades, [F29H85x TIFS SDK 1.01.00](#) (or newer) must be installed as well. The project can be found at the following location in the F29H85x SDK: f29h85x-sdk\_xx\_xx\_xx\_xx\examples\driverlib\single\_core\flash\flash\_based\_UART\_SBL\_with\_FOTA.

Additional documentation on secure firmware upgrades can be found in the TIFS SDK (tifs\_f29h85x\_xx\_xx\_xx\_xx/docs/api\_guide\_f29h85x/html/docs\_src/secure\_firmware\_update/secure\_firmware\_update.html)

## 1.1 Hardware Security Module

A key difference between C28-based devices and the F29H85x is the integration of the Hardware Security Module (HSM). The HSM is a subsystem that provides security and cryptographic functions. The C29 CPUs interface with the HSM to perform cryptographic operations required for code authentication, secure boot, secure firmware upgrades, and encrypted run-time communications.

During the flash boot sequence, the HSM is responsible for authentication of image present in flash. For the authentication to succeed, the programmed image must include an X.509 certificate. The post-build steps of the project are responsible for generating a valid X.509 certificate. Please refer to [this section](#) for details on the post-build steps.

The HSM introduces the concept of different device *security states*. The device states are High Security - Field Securable (HS-FS), High Security - Key Provisioned (HS-KP), and High Security - Security Enabled (HS-SE). By default, the F29H85x device ships with HS-FS. [Table 1-1](#) describes the differences between these three states.

**Table 1-1. Device Security State**

	HS-FS	HS-KP	HS-SE
C29 boot image (flash kernel)	Secure boot not enforced	Secure boot enforced with customer keys programmed by keywriter	Secure boot enforced with customer keys programmed by keywriter
HSM boot image	Secure boot enforced (with default TI-provided key)	Secure boot enforced with customer keys programmed by keywriter	Secure boot enforced with customer keys programmed by keywriter
C29 JTAG	Open by default	Open by default	Closed by default
SoC firewalls	Open by default	Disabled for HSM and enabled for C29	Disabled for HSM and enabled fro C29
C29 CPU access to C29 flash banks	Enabled	Disabled	Enabled

The FOTA upgrade process differs between HS-FS and HS-SE devices. For details regarding the C29 CPU1/ CPU3 FOTA upgrades on an HS-FS device, refer to [the High-Level Flow section](#).

For details regarding C29 CPU1/3 or HSM FOTA upgrades on an HS-SE device, refer to the Secure Firmware Upgrade section of the TIFS SDK (tifs\_f29h85x\_xx\_xx\_xx/docs/api\_guide\_f29h85x/html/docs\_src/secure\_firmware\_update/secure\_firmware\_update.html)

## 1.2 Flash Programming Fundamentals

Before programming a device, understanding how the non-volatile memory of C2000 devices works is necessary. Flash is a non-volatile memory that allows users to easily erase and re-program. Erase operations set all the bits in a sector to '1' while programming operations selectively set bits to '0'.

Flash operations on all C2000 devices are performed using the CPU. Algorithms are loaded into RAM and executed by the CPU to perform any flash operation. For example, erasing or programming the flash of a C2000 device with Code Composer Studio™ entails loading flash algorithms into RAM and letting the processor execute them. There are no special JTAG commands that are used. All flash operations are performed using the flash application programming interface (API). Because all flash operations are done using the CPU, there are many possibilities for device programming.

## 1.3 High-Level Flow

On an HS-FS device, the flash-based UART SBL with FOTA project follows this flow:

1. Boot to flash (standalone or emulation boot).
2. Initialize device (clocks, GPIOs, peripherals, and so forth).
3. Compare the current bank mode to the bank mode in which the most recent successful firmware upgrade occurred.
  - a. If this is the first time the device has booted and the SBL or application was loaded through CCS or JTAG, then the SBL bypasses this check.
  - b. This is possible to boot existing firmware after a bank mode change, but, this example requires a firmware upgrade every time the bank mode changes.
4. If the current bank mode is the same as the previous bank mode, then start the timeout period to wait for a firmware upgrade command (5 seconds). If the bank mode has changed, then the device waits indefinitely for a firmware upgrade and does not attempt to boot to existing firmware.
5. If a firmware upgrade command is received, then CPU1 receives the firmware and programs the upgrade over UART for the requested CPU (CPU1 or CPU3).
6. If no firmware upgrade command is received before the timeout, then branch to the application entry point and begin executing.
7. Once the application is executing, interrupts are configured to make sure that a firmware upgrade command can be received and processed over UART.
8. If the firmware upgrade command is received, then the application branches to the SBL code and performs the requested firmware upgrade. Any application ISRs in the existing application is serviced throughout the duration of the firmware upgrade.
9. Once the firmware has been successfully programmed to the inactive flash region, the SBL programs the inactive BANKMGMT region such that a swap is triggered on a device reset.
  - a. For more details on bank swaps, refer to [Section 2.2](#).
10. If successful, then CPU1 programs the current bank mode to the first byte of the data flash. This indicates the bank mode of the most recent successfully firmware upgrade next time the device boots.
11. After the firmware upgrade is completed, the SBL returns control to the application if this exists.
12. If successful, then a bank swap is triggered and the new firmware executes after a device reset.

On an HS-SE device, the flash-based UART SBL with FOTA project follows this flow:

1. Boot to flash (standalone or emulation boot)
2. Initialize device (clocks, GPIOs, peripherals, and so forth.)

3. Compare the current bank mode to the bank mode in which the most recent successful firmware upgrade occurred.
  - a. If this is the first time the device has booted and the SBL or application was loaded through CCS or JTAG, then the SBL bypasses this check
  - b. This is possible to boot existing firmware after a bank mode change, but for the sake of simplicity this example requires a firmware upgrade every time the bank mode changes.
4. If the current bank mode is the same as the previous bank mode, then start the timeout period to wait for a firmware upgrade command. If the bank mode has changed, then the device waits indefinitely for a firmware upgrade and does not attempt to boot to existing firmware.
5. If a firmware upgrade command is received, then CPU1 receives the firmware and sends to the HSM in chunks. The HSM authenticates and programs each chunk to the inactive flash.
6. Once all chunks have been programmed by the HSM, CPU1 requests the HSM to perform an integrity check on the firmware and programs the certificate to flash.
7. If no firmware upgrade command is received before the timeout, then branch to the application entry point and begin executing.
8. Once the application is executing, interrupts are configured to make sure that a firmware upgrade command can be received and processed over UART.
9. If firmware upgrade command is received, then the application branches to the SBL code and performs the requested firmware upgrade. Any application ISRs in the existing application is serviced throughout the duration of the firmware upgrade.
10. Once the firmware has been successfully programmed to the inactive flash region, the SBL programs the inactive BANKMGMT region such that a swap is triggered on a device reset.
11. If successful, then CPU1 programs the current bank mode to the first byte of the data flash. This indicates the bank mode of the most recent successfully firmware upgrade next time the device boots.
12. After the firmware upgrade is completed, the new firmware executes after a device reset.

---

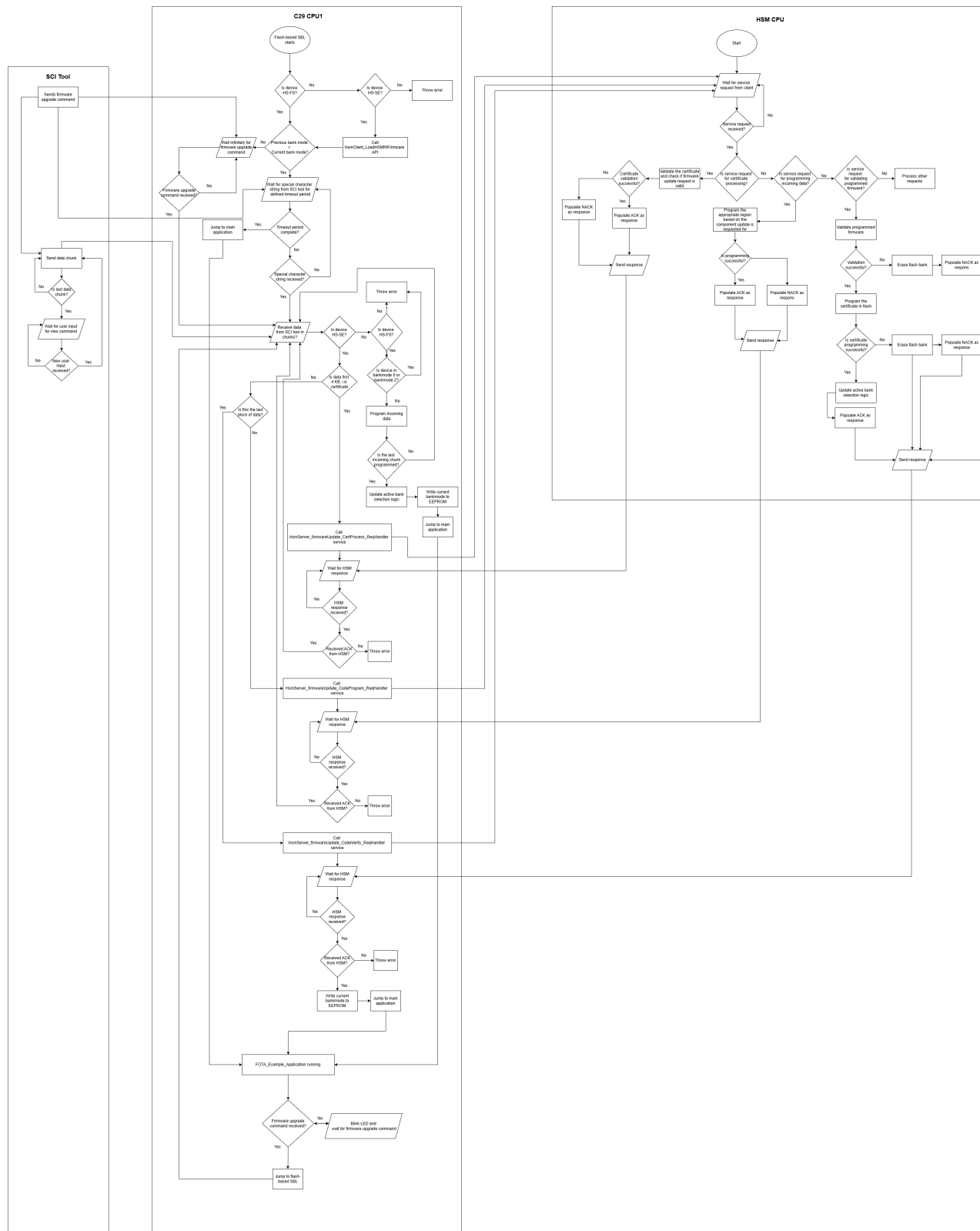
#### Note

The general process of a FOTA upgrade is similar between HS-FS and HS-SE devices, but the key difference is the integration of the HSM. The HSM is responsible for authenticating existing and incoming flash image, programming the authenticated incoming image, and performing an integrity check of the incoming image. On an HS-FS device, these steps are not required. CPU1 is responsible for receiving the incoming image and programming to flash without any authentication or integrity checks.

---

## 1.4 Flow Chart

A detailed flow chart of both the HS-FS and HS-SE firmware upgrade flows can be found in [Figure 1-2](#).



### Figure 1-2. Flow Chart

## 2 Flash-Based UART SBL with FOTA

To begin developing with the flash-based SBL project, begin by importing into CCS (*F29h85x\_sdk\_1\_01\_00\_00\examples\driverlib\single\_core\flash\flash\_based\_UART\_SBL\_with\_FOTA*) Once imported, there are two projects are present: *flash\_based\_UART\_SBL\_with\_FOTA* and *FOTA\_Example\_Application*. First, this document details the flash-based UART SBL.

### 2.1 Implementation

As stated above, the primary purpose of the flash-based UART SBL with FOTA project is to provide an example implementation for flash-based firmware upgrades.

The implementation has a lot in common with the *ex3\_uart\_flash\_kernel* project (*F29h85x\_sdk\_xx\_xx\_xx\_xx\examples\driverlib\single\_core\flash\uart\_flash\_kernel*), so TI recommends to review the [Serial Flash Programming of F29H85x™ application note](#) describing the *ex3\_uart\_flash\_kernel* project. The application note provides more context on how the flash-based SBL can be programmed to both an HS-FS and HS-SE device over UART.

A key difference between the UART flash kernel implementation and the flash-based SBL is that the SBL does not need to be loaded in by bootROM after every reset. Once the project is loaded to CPU1 flash (either by CCS or the UART flash kernel), the flash boot mode can be used to enter the SBL. The SBL occupies the first 83KB of CPU1 flash memory beginning at 0x10001000. The sample application provided in the *FOTA\_Example\_Application* project is stored at 0x10020000 and is separate from the SBL. The location of the sample application can be changed as needed, but make sure to update both the linker file of the SBL and the example application. For more details about this, refer to [Section 3.2](#).

Another key difference in the flash-based SBL project is that the bank swapping mechanism is used to swap between firmware images. Thus, the F29H85x must be in bank mode 1 or bank mode 3 (0 and 2 do not support bank swapping). If only CPU1 is running, then use bank mode 1. If both CPU1 and CPU3 are executing an application, then use bank mode 3. Refer to the TRM and data sheet for more details about how flash memory is distributed based on bank mode. To program bank mode 1 or 3 on the F29H85x, use the flash plugin in CCS or the flash API function (*Fapi\_issueProgBankMode()*). To do so in CCS, refer to [Section 5.1.1](#). After programming the bank mode, this takes effect after a device reset.

Once the flash-based SBL has been programmed and is executing in CPU1 flash, this is ready to receive commands from the Host Application. More details about the host application can be found in [Section 4](#). A demonstration of using the project can be found in [Section 5](#).

### 2.2 Triggering a Bank Swap

In this section, the details of how a bank swap for CPU1 or CPU3 is triggered in bank mode 1 or bank mode 3 are described. First, the basics of the Bank Management region of flash are important to understand. The *Flash BANKMGMT Region Address Map* section of the [F29H85x and F29P58x Real-Time Microcontrollers data sheet](#) shows the memory map of the bank management region in all four bank modes. This document focuses on the *Flash BANKMGMT Region Address Mapping (BANKMODE = 1)* table, which is applicable for bank mode 1.

As shown below, the BANKMGMT region in bank mode 1 is all mapped to CPU1. There are four regions listed in the table and these are split evenly between the active (FRI1) and inactive (FRI3) region of CPU1 flash.

**However, the BANKMGMT regions in FRI1.RP1 and FRI3.RP1 are unused. These regions are reserved for determining the SWAP value for CPU3. Since CPU3 does not have any flash allocated in Mode 1, these regions are unused.**

**Table 2-1. Flash BANKMGMT Region Address Mapping (BANKMODE = 1)**

FRI	READ PORT	SIZE	START ADDRESS	END ADDRESS	FLASH BANKS (SWAP = 0)	FLASH BANKS (SWAP = 1)
FRI-1 (CPU1 program)	RP0	4KB	0x10D80000	0x10D80FFF	FLC1.B0/B1	FLC1.B2/B3
	RP1	4KB	0x10D84000	0x10D84FFF	FLC2.B0/B1	FLC2.B2/B3
	RP2	4KB	0x10D88000	0x10D88FFF	N/A	N/A
	RP3	4KB	0x10D8C000	0x10D8CFFF	N/A	N/A
FRI-2 (CPU3 program)	RP0	4KB	0x10D90000	0x10D90FFF	N/A	N/A
	RP1	4KB	0x10D94000	0x10D94FFF	N/A	N/A
FRI-3 (Update region)	RP0	4KB	0x10D98000	0x10D98FFF	FLC1.B2/B3	FLC1.B0/B1
	RP1	4KB	0x10D9C000	0x10D9CFFF	FLC2.B2/B3	FLC2.B0/B1

The BANKMGMT region of each bank pair contains three fields that are used to manage bank status and firmware updates. The *BANKMGMT Sectors* section of the [F29H85x Technical Reference Manual \(TRM\)](#) describes these fields in detail.

1. **BANK\_STATUS[63:0]** (Offset 0x00): This field specifies whether the bank contents are currently valid or invalid. A valid bank pair has a status value of 0x55555555\_55555555. If both banks are inactive, the device is unable to execute Flash code and can fail to boot. If both banks have valid active statuses, then the current active bank is determined by BANK\_UPDATE\_CTR.
2. **BANK\_UPDATE\_CTR[63:0]** (Offset 0x08): This is a 64-bit *decrementing* counter value that is used to determine the most recent firmware revision. The counter has a value of 0xFFFFFFFF\_FFFFFFFF when shipped from the TI factory. When both bank pairs have valid status values, the BANK\_UPDATE\_CTR for each bank pair is compared against the other to determine the most recent firmware revision. The bank pair with the lower counter value is determined to be the most recent revision.
3. **BANKMODE[63:0]** (Offset 0x10): This field only exists in FLC1 BANKMGMT regions (for CPU1). The BANKMODE field configures the device bankmode and is loaded into the SSU\_GEN\_REGS.BANKMODE register at device boot. After BANKMODE is configured, the boot ROM then reads the value of each bank pair's BANK\_UPDATE\_CTR to determine whether CPUxSWAP is enabled.

Figure 2-1 is an example BANKMGMT region of a device in bank mode 1.

0x10D80000	55	55	55	55	55	55	55	55	55	D6	FF	FF	FF	FF	FF	FF	FF	06	00	00	00	00	00	00	00
0x10D98000	55	55	55	55	55	55	55	55	55	D7	FF	FF	FF	FF	FF	FF	FF	0C	00	00	00	00	00	00	00

**Figure 2-1. BANKMGMT Region**

The 0x10D80000 region (FRI1 RP0) has the lowest BANK\_UPDATE\_CTR. Thus, this is selected as the active BANKMGMT region for CPU1 which puts the device in bank mode 1 (0x06). The inactive region (FRI3 RP0) has the BANKMODE field set to mode 3 (0x0C), but BANK\_UPDATE\_CTR is higher than FRI1 RP0 so this is not used.

Another register that can be used to track the active or inactive bank status is the SSU\_GEN\_REG.BANKMAP registers. Refer to the data-sheet or TRM for exact definitions. Notice that the CPU1SWAP field is set to 0x36. This is one of the two possible values and indicates which physical flash banks are active.

▼ BANKMAP	0x0000C936
CPU3SWAP	0xC9
CPU1SWAP	0x36

**Figure 2-2. BANKMAP Before Swapping**

To trigger a swap, update the inactive BANKMGMT region (FRI3) to decrement the BANK\_UPDATE\_COUNTER such that this is lower than the value found in the active region (FRI1 RP0). As mentioned above, this can be done with CCS by the flash plugin or can be done manually using the flash API. In the flash-based SBL project, the ProgramInactiveBankManagement() function is used to do this in ex4\_uart\_boot\_cpu1.c. At a high-level, here is the implementation of the function:

1. First, the function determines the appropriate bank management address to program (CPU1 or CPU3)
2. Erase the inactive bank management sector
3. Program the inactive BANKMODE field with the desired bank mode
4. Program the inactive BANK\_STATUS and BANK\_UPDATE\_CTR fields
  - a. The BANK\_UPDATE\_CTR field is programmed with a value that is one less than the BANK\_UPDATE\_CTR field in the active BANKMGMT region

#### CAUTION

**TI recommends to program the BANKMODE field before programming the BANK\_STATUS BANK\_UPDATE\_CTR fields just in case of an interruption (power loss, and so forth.) during this process. If the device loses power after BANK\_STATUS and BANK\_UPDATE\_CTR fields are programmed without a valid BANKMODE field, then the device does not boot properly.**

After calling this function, the inactive BANKMGMT region is updated.

```
0x10D98000 55 55 55 55 55 55 55 55 D5 FF FF FF FF FF FF FF 06 00 00 00 00 00 00 00
```

**Figure 2-3. Inactive BANKMGMT Updated for Swap**

The BANK\_UPDATE\_COUNTER is now lower than the counter in the active region and the BANKMODE field is now mode 1 (0x6). After triggering a device reset, these fields are in the active BANKMGMT address space (FRI1 RP0).

```
0x10D80000 55 55 55 55 55 55 55 55 D5 FF FF FF FF FF FF FF 06 00 00 00 00 00 00 00
```

**Figure 2-4. Active BANKMGMT Region**

Additionally, the BANKMAP.CPU1SWAP register also indicates that a swap has occurred.

▼ BANKMAP	0x0000C9C9
CPU3SWAP	0xC9
CPU1SWAP	0xC9

**Figure 2-5. BANKMAP Swap**

The CPU1SWAP value is now 0xC9, as opposed to 0x36. This indicates that the swap has successfully occurred and the physical flash banks associated with the CPU1 flash active address space have swapped.

## 3 FOTA\_Example\_Application

The FOTA\_Example\_Application project is a modified LED blinky example that is able to leverage the flash-based SBL project. The main capability that allows this application to be compatible with the flash-based SBL is the ability to branch to receive commands from the host programmer and branch to the appropriate region of the SBL. In the following section, this document describes how that is achieved.

### 3.1 led\_blinky\_cpu1.c

This source file within the FOTA\_Example\_Application project contains the logic required to blink an LED, receive UART commands, and branch back to the flash-based SBL project depending on which UART command is received.

The program begins by initializing the device (clocks, peripheral clocks, interrupts), initializing the Sysconfig settings (UART config, interrupt config) and then enabling interrupts. Once initialization is complete, the device sits in a while loop that waits for a recognized command to be received by the UART module. There is also a CPUTIMER interrupt configured that blinks an LED every second. This acts as the application ISR for the purposes of demonstrating FOTA. Both of these ISRs are configured in the led\_blinky\_cpu1.syscfg file.

Figure 3-1 shows the configuration of the CPUTIMER interrupt. This is configured with a priority level of 250.

**CPUTIMER (1 of 3 Added)** [ADD] [REMOVE ALL]

myCPUTIMER1

Name: myCPUTIMER1

**Peripheral Configuration**

- CPUTIMER Instance: CPUTIMER1
- Timer Input Frequency (MHz): 200
- Emulation Mode: Timer Stop after the next dec...
- Timer Prescaler: 0
- Timer Period (cycles): 200000000
- Timer Period (s): 1
- Timer Frequency (Hz): 1
- Start Timer: ☒

**Interrupt Configuration**

- Enable Interrupt: ☒
- Register Interrupt in PIPE: ☒

**Timer Interrupt**

- Name: myCPUTIMER1\_INT
- Interrupt Handler: INT\_myCPUTIMER1\_ISR
- Enable Interrupt in PIPE: ☒
- Interrupt Priority: 250
- Interrupt Context ID: Context-ID 0

**Figure 3-1. CPU Timer ISR Configuration in SysConfig**

The associated ISR is shown in Figure 3-2.

```
//
// ISR to blink LED
//
void INT_myCPUTIMER1_ISR(void)
{
    ledBit = ledBit ^ 1;
    GPIO_writePin(CPU1_LED, ledBit);
}
```

**Figure 3-2. CPU Timer ISR**

As shown in [Figure 3-3](#), the UART module is configured with a baud rate of 115200 and triggers an ISR if the RX FIFO receives 8 bytes.

Baud Rate Select	115200
Actual Baud Rate Value	115207
Baud Rate Error Percent	0.006400409626217475
<b>Word Settings</b>	
Word Length	8 bit data
Stop Bits	1 stop bit
Parity Enable	<input type="checkbox"/>
Enable FIFO	<input checked="" type="checkbox"/>
Enable Loopback Mode	<input type="checkbox"/>
Use Case	ALL
<b>Interrupt Configuration</b>	
Enable Interrupts	<input checked="" type="checkbox"/>
Register Interrupt Handler	<input checked="" type="checkbox"/>
Enabled Interrupts	Receive Interrupt Mask
Transmit FIFO Interrupt Level	Transmit interrupt at 1/2 Full
Receive FIFO Interrupt Level	Receive interrupt at 1/2 Full
<b>UART Interrupt</b>	
Name	myUART0_UART_INT
Interrupt Handler	INT_myUART0_ISR
Enable Interrupt in PIPE	<input checked="" type="checkbox"/>
Interrupt Priority	244
Interrupt Context ID	Context-ID 0

**Figure 3-3. UART Configuration in SysConfig**

The source code of the UART RX ISR is shown in [Figure 3-4](#).

```
//
// This ISR will catch cmd packets from the host
//
void INT_myUART0_ISR(void)
{
    //
    // Get the interrupt status.
    //
    uint32_t ui32Status;
    ui32Status = UART_getInterruptStatus(myUART0_BASE, UART_RAW_INT);

    //
    // Clear the asserted interrupts.
    //
    UART_clearInterruptStatus(myUART0_BASE, ui32Status);
    command = uartGetPacket(&length, data);
    // Clear UART global interrupt flag
    UART_clearGlobalInterruptFlag(myUART0_BASE);
}
```

**Figure 3-4. UART RX ISR**

This assigns the command packet variable by reading the UART data and then clear the interrupt flag. Once the command packet has been parsed and assigned to the command variable, the application resumes the while loop in the main function.

```
//
// DFU_CPU1
//
if(command == DFU_CPU1)
{
    //
    // Jump to CPU 1 FOTA function
    //
    [[clang::noinline]] CPU_jumpToAddr(CPU1_FOTA_ENTRY_POINT);

    command = 0;
}
else if(command == DFU_CPU3)
{
    //
    // Jump to CPU 3 FOTA function
    //
    [[clang::noinline]] CPU_jumpToAddr(CPU3_FOTA_ENTRY_POINT);

    command = 0;
}
```

**Figure 3-5. Main Loop**

Within this main function, the command value is constantly compared to the predefined command packet values in f29h85x\_kernel\_commands\_cpu1.h. These values are common between the flash-based SBL, UART host programmer, and the FOTA\_Example\_Application. Once a valid command is received, the example project jumps to the appropriate entry point. The macros used in the CPU\_jumpToAddr() function can be found at the top of led\_blinky\_cpu1.c.

```

59  //
60  #define CPU1_FOTA_ENTRY_POINT 0x1001F000
61  #define CPU3_FOTA_ENTRY_POINT 0x1001F200
62  #define CPU1_SECURE_FOTA_ENTRY_POINT 0x1001F400
63  #define CPU3_SECURE_FOTA_ENTRY_POINT 0x1001F600
64  #define HSM_SECURE_FOTA_ENTRY_POINT 0x1001F800
65  #define SECCFG_PROGRAM_ENTRY_POINT 0x1001FA00
66

```

**Figure 3-6. Addresses to Branch to in SBL to Perform FOTA Upgrade**

These correspond directly to memory sections created in the flash-based SBL linker cmd files and output sections created in the ex4\_uart\_sbl.c file.

```

10
11  MEMORY
12  {
13      SRAM_LDAX : o=0x200E0000, l=0x1C000
14      SRAM_LDA0 : o=0x200FC000, l=0x04000
15      SRAM_LPAx : o=0x20100000, l=0x10000
16      SRAM_CPAx : o=0x20110000, l=0x10000
17      SRAM_CDAx : o=0x20120000, l=0x30000
18
19      CERT      : o=0x10000000, l=0x001000
20      CPU1_FLASH_RP0 : o=0x10001000, l=0x01E000
21      CPU1_FOTA   : o=0x1001F000, l=0x200
22      CPU3_FOTA   : o=0x1001F200, l=0x200
23      CPU1_SECURE_FOTA : o=0x1001F400, l=0x200
24      CPU3_SECURE_FOTA : o=0x1001F600, l=0x200
25      HSM_SECURE_FOTA : o=0x1001F800, l=0x200
26      SECCFG_PROGRAM : o=0x1001FA00, l=0x200
27      CPU1_APP     : o=0x10020000, l=0x00F000
28      CPU3_FLASH_RP0 : o=0x10400000, l=0x100000
29      CPU1_UPDATE_FLASH : o=0x10600000, l=0x200000
30
31      MAILBOX_R5F           : o=0x302C0800, l=0x000007FF
32      MAILBOX_HSM           : o=0x302C1000, l=0x000007FF
33  }
34

```

```

354  //
355  // Function that firmware will branch to when DFU CPU1 command received from firmware
356  //
357  __attribute__((__used__))
358  __attribute__((section("CPU1_FOTA_ENTRY")))
359  void cpu1FOTA(void)
360  {
361

```

**Figure 3-7. Linker CMD Memory Sections and Output Section Creation**

**WARNING**

All of these addresses need to be updated if the entry points are changed.

Once the FOTA\_Example\_Application has received a valid command from the host and has branched back to the flash-based SBL project, the application ISRs continue to be serviced. The LED continues blinking for the entirety of the firmware update process. Once the flash-based SBL has completed the firmware upgrade, the device branches back to the FOTA\_Example\_Application and waits for another command. After a device reset, the banks swap and the newly programmed firmware are in the active address space.

### 3.2 Combining the Flash-Based SBL with the FOTA\_Example\_Application

The flash-based SBL and FOTA\_Example\_Application can be combined to reduce the steps required to program these two projects into flash. This section details the source code and post build steps required to accomplish this.

First, the flash-based SBL project creates an output section that contains the FOTA\_Example\_Application data. In ex4\_uart\_sbl.c, the following code is used to create this empty section.

```
77
78  __attribute__((retain, section("firmware"))) const uint8_t firmwaredata[27000] = {0U};
79
```

Figure 3-8. Output Section Creation in SBL Project

The size of this output section can be modified based on the size of the application.

In the flash-based SBL linker file, several modifications to the available memory sections. The CPU1\_FOTA, CPU3\_FOTA, and so forth, memory sections correspond to the output sections created in the ex4\_uart\_sbl.c file. For example, the source code in Figure 3-9 is creating an output section for the CPU1 FOTA function.

```
//
// Function that firmware will branch to when DFU CPU1 command received from firmware
//
__attribute__((__used__))
__attribute__((section("CPU1_FOTA_ENTRY")))
void cpu1FOTA(void)
```

Figure 3-9. Creating CPU1 FOTA Output Section

The CPU1\_FOTA memory section is being populated with the CPU1\_FOTA\_ENTRY output section that was created in Figure 3-10.

```
18
19  CERT      : o=0x10000000, l=0x001000
20  CPU1_FLASH_RP0 : o=0x10001000, l=0x01E000
21  CPU1_FOTA   : o=0x1001F000, l=0x200
22  CPU3_FOTA   : o=0x1001F200, l=0x200
23  CPU1_SECURE_FOTA : o=0x1001F400, l=0x200
24  CPU3_SECURE_FOTA : o=0x1001F600, l=0x200
25  HSM_SECURE_FOTA : o=0x1001F800, l=0x200
26  SECCFG_PROGRAM : o=0x1001FA00, l=0x200
27  CPU1_APP     : o=0x10020000, l=0x00F000
28  CPU3_FLASH_RP0 : o=0x10400000, l=0x100000
29  CPU1_UPDATE_FLASH : o=0x10600000, l=0x200000
30
```

```

CPU1_FOTA_ENTRY : {} > CPU1_FOTA, palign(8)
CPU3_FOTA_ENTRY : {} > CPU3_FOTA, palign(8)
CPU1_SECURE_FOTA_ENTRY : {} > CPU1_SECURE_FOTA, palign(8)
CPU3_SECURE_FOTA_ENTRY : {} > CPU3_SECURE_FOTA, palign(8)
HSM_SECURE_FOTA_ENTRY : {} > HSM_SECURE_FOTA, palign(8)
SECCFG_PROGRAM_ENTRY : {} > SECCFG_PROGRAM, palign(8)

firmware      : {} > CPU1_APP,    palign(8)

```

**Figure 3-10. Linker CMD File in SBL**

Additionally, the CPU1\_FLASH\_RP0 section has been modified such that this does not overlap with any of the other sections. The CPU1\_APP section is used to store the firmware section created in the [flash-based SBL source code](#).

To populate the firmware section with the FOTA\_Example\_Application, refer to the post-build steps of both projects (right click the project -> Properties -> Build -> Steps). The post-build steps of the FOTA\_Example\_Application uses the C29 OBJCOPY tool to convert the compiled .out file into a .bin file.

#### Post-build steps

Enter optional description...

```

${CG_TOOL_OBJCOPY} -O binary ${ProjName}.out ../../firmware.bin

```

**Figure 3-11. FOTA\_Example\_Application Post-Build Step**

Once that is complete, the flash-based UART SBL project can be built. The post-build steps in the flash-based UART SBL project are as shown:

#### Post-build steps

Enter optional description...

```

${CG_TOOL_OBJCOPY} --update-section firmware=../../firmware.bin ${ProjName}.out ${ProjName}_firmware.out
${CG_TOOL_OBJCOPY} --remove-section=cert -O binary ${ProjName}_firmware.out ${ProjName}.bin
${PYTHON} ${COM_TI_MCU_SDK_INSTALL_DIR}/tools/boot/signing/mcu_rom_image_gen.py --image-bin ${ProjName}.bin --core C29 --swrv 1
${CG_TOOL_OBJCOPY} --update-section cert=C29-cert-pad.bin ${ProjName}_firmware.out ${ProjName}_cert.out
$(DELETE) ${ProjName}.out C29-cert-pad.bin
$(RENAME) ${ProjName}_cert.out ${ProjName}.out
${CG_TOOL_OBJCOPY} -O binary ${ProjName}.out ${ProjName}.bin

```

**Figure 3-12. SBL Post-Build Step**

1. The firmware.bin file created by the FOTA\_Example\_Application is inserted into the firmware output section mentioned earlier. A new flash\_based\_uart\_sbl\_with\_fota\_firmware.out file is created
2. The cert output section is removed from the .out file and a new flash\_based\_uart\_sbl\_with\_fota.bin file is created
3. This bin file is passed to the mcu\_rom\_image\_gen.py script to create a valid X.509 certificate for the flash\_based\_uart\_sbl\_with\_fota.bin file
4. The cert output section is updated with this X.509 certificate (C29-cert-pad.bin) and a new .out file is created (flash\_based\_uart\_sbl\_with\_fota\_cert.out)
5. The flash\_based\_uart\_sbl\_with\_fota\_cert.out file is renamed as flash\_based\_uart\_sbl\_with\_fota.out
6. flash\_based\_uart\_sbl\_with\_fota.bin is created by using the C29 OBJCOPY tool

Now, flash\_based\_uart\_sbl\_with\_fota.bin and flash\_based\_uart\_sbl\_with\_fota.out contain both a valid X.509 certificate and the data from FOTA\_Example\_Application. The .out file can be loaded to the device via CCS and the .bin file can be loaded to the device via the UART flash kernel.

### 3.3 Adding a CPU3 Application

Although this example project does not provide a CPU3 application, there are only a few minor modifications needed to add CPU3 compatibility. The flash-based SBL supports CPU3 firmware upgrades, but the CPU1 application is responsible for programming the firmware and taking CPU3 out of reset. To demonstrate this, the FOTA\_Example\_Application can be modified to take CPU3 out of reset. The steps required for CPU1 to take CPU3 out of reset are as follows:

1. Configure the address and link for CPU3 to boot to
2. Configure the NMI address for CPU3
3. Bring CPU3 out of reset

Figure 3-13 shows the source code that can be added to have CPU1 take CPU3 out of reset. This occurs after device initialization but before interrupt initialization. Refer to any of the multi-core example in the F29H85x SDK for more details.

```

100 //
101 // Defines the address and link to which CPU3 Boots
102 //
103 SSU_configBootAddress(SSU_CPU3, (uint32_t)CPU3_RESET_VECTOR, SSU_LINK2);
104 SSU_configNmiAddress(SSU_CPU3, CPU3_NMI_VECTOR, SSU_LINK2);
105
106 //
107 // Bring CPU3 out of reset. Wait for CPU3 to go out of reset.
108 //
109 SSU_controlCPUReset(SSU_CPU3, SSU_CORE_RESET_DEACTIVE);
110 while(SysCtl_isCPU3Reset() == 0x1U);
111

```

**Figure 3-13. CPU1 Code Required to Take CPU3 Out of Reset**

---

***Additionally, the BANKMODE\_3 build configuration must be used for CPU3 upgrades.***

---

## 4 Host Application: UART Flash Programmer

### 4.1 Overview

The user interacts with the flash-based SBL project in an almost identical manner as the UART flash kernel. The host application used to send commands and firmware images is common between the two projects, and the host has been updated to leverage several new features available in the flash-based SBL. The UART host flash programmer can be found at the following location: f29h85x-sdk\_x\_xx\_xx\_xx > tools> flash\_programmers > uart\_flash\_programmer.

#### Note

**The key difference between the host application usage for the UART kernel and the flash-based SBL is that there is no need to load a kernel .bin file to the device by bootROM. Thus, use the uart\_flash\_programmer\_appln.exe with the flash-based SBL project. This executable has all the same functionality as uart\_flash\_programmer.exe, but does not begin with sending a kernel .bin file over UART.**

Table 4-1 lists the commands accepted by the flash-based SBL project:

**Table 4-1. Commands**

Command	Description
DFU CPU1 (Device Firmware Upgrade CPU1)	<ol style="list-style-type: none"> <li>1. Receive the command packet</li> <li>2. Erase the inactive region of flash allocated to CPU1</li> <li>3. Receive the flash-based application over UART</li> <li>4. Program and verify the application to the inactive region of CPU1 flash</li> <li>5. Program the inactive bank management region of CPU1 such that a bank swap is triggered after reset</li> <li>6. Program the bank mode to the first byte of data flash</li> <li>7. Send message back to host for final status</li> </ol>
DFU CPU3 (Device Firmware Upgrade CPU3) <i>Device must be in Bank Mode 3</i>	<ol style="list-style-type: none"> <li>1. Receive the command packet</li> <li>2. Erase the inactive region of flash allocated to CPU3</li> <li>3. Receive the flash-based application over UART</li> <li>4. Program and verify the application to the inactive region of CPU3 flash</li> <li>5. Program the inactive bank management region of CPU3 such that a bank swap is triggered after reset</li> <li>6. Program the bank mode to the first byte of data flash</li> <li>7. Send message back to host for final status</li> </ol>
HSM_CP_FLASH_IMAGE <i>Device must be in the HS-SE State</i>	<ol style="list-style-type: none"> <li>1. Receive the command packet</li> <li>2. Pass X.509 certificate to the HSM to authenticate incoming firmware</li> <li>3. Receive chunks of firmware over UART</li> <li>4. Pass chunks to HSM for programming <ol style="list-style-type: none"> <li>a. HSM programs firmware to HSM inactive flash region</li> </ol> </li> <li>5. Once all chunks are programmed, request the HSM to verify the programmed firmware</li> <li>6. Program the bank mode to the first byte of data flash</li> <li>7. Send message back to host for final status</li> </ol>

**Table 4-1. Commands (continued)**

Command	Description
CPU1_CP_FLASH_IMAGE <i>Device must be in the HS-SE State</i>	<ol style="list-style-type: none"> <li>1. Receive the command packet</li> <li>2. Receive chunks of firmware over UART</li> <li>3. Pass X.509 certificate to the HSM to authenticate incoming firmware</li> <li>4. Pass chunks to HSM for programming <ol style="list-style-type: none"> <li>a. HSM programs firmware to CPU1 inactive flash region</li> </ol> </li> <li>5. Once all chunks are programmed, request the HSM to verify the programmed firmware</li> <li>6. Program the inactive bank management region of CPU1 such that a bank swap is triggered after reset</li> <li>7. Program the bank mode to the first byte of data flash</li> <li>8. Send message back to host for final status</li> </ol>
CPU3_CP_FLASH_IMAGE <i>Device must be in Bank Mode 3</i> <i>Device must be in the HS-SE State</i>	<ol style="list-style-type: none"> <li>1. Receive the command packet</li> <li>2. Receive chunks of firmware over UART</li> <li>3. Pass X.509 certificate to the HSM to authenticate incoming firmware</li> <li>4. Pass chunks to HSM for programming <ol style="list-style-type: none"> <li>a. HSM programs firmware to CPU3 inactive flash region</li> </ol> </li> <li>5. Once all chunks are programmed, request the HSM to verify the programmed firmware</li> <li>6. Program the inactive bank management region of CPU3 such that a bank swap is triggered after reset</li> <li>7. Program the bank mode to the first byte of data flash</li> <li>8. Send message back to host for final status</li> </ol>
SEC_CFG_IMAGE <i>Device must be in the HS-SE State</i>	<ol style="list-style-type: none"> <li>1. Receive the command packet</li> <li>2. Receive SECCFG image over UART and place in shared RAM</li> <li>3. Pass X.509 certificate to the HSM to authenticate incoming firmware</li> <li>4. Once the firmware is authenticated, request the HSM to program the SECCFG image</li> <li>5. Send command packet</li> </ol>
SYNC_STATUS	<ol style="list-style-type: none"> <li>1. Receive the command packet</li> <li>2. Send data packet indicating that flash-based SBL is live</li> <li>3. Send command packet</li> </ol>

Similar to what is described in the [Serial Flash Programming of F29H85x™ application note](#), all firmware passed to the host is in the .bin file format with a valid X.509 certificate in the first 0x1000 bytes. Refer to the *Combined Image with X.509 Certificate* section of the application note for details on generating a valid .bin file and X.509 certificate.

For more details on how to use the `uart_flash_programmer` tool, refer to `f29h85x-sdk_1_01_00_00\docs\html\UART_FLASH_PROGRAMMER_PAGE.html`

Here is an example of how the `uart_flash_programmer_appIn.exe` is used with the flash-based SBL:

```
uart_flash_programmer_appIn.exe -d f29h85x -k flash_kernel.bin -a1 cpu1_application.bin -a3
cpu3_application.bin -p COM34
```

## 5 Example Usage

This section discusses how to use the flash-based SBL to perform a FOTA upgrade on an HS-FS device. For details on using the project with an HS-SE device, please refer to the Secure Firmware Update documentation available in the TIFS SDK (tifs\_f29h85x\_xx\_xx\_xx\_xx/docs/api\_guide\_f29h85x/html/docs\_src/secure\_firmware\_update/secure\_firmware\_update.html).

### 5.1 Loading the SBL onto the Device

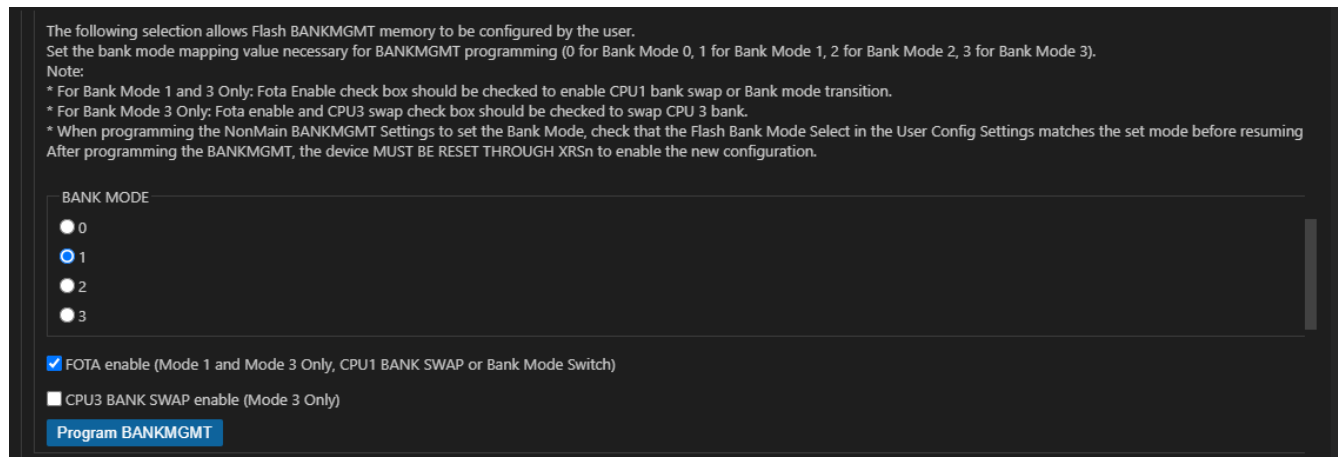
This section of the document details how the flash-based SBL (and example application) can be loaded onto the device during development or over UART.

#### 5.1.1 Loading by CCS (JTAG)

Loading the project to CCS is exactly the same as loading any other .out file by the flash plugin tool.

1. Launch the F29H850TU9.ccxml file as *Project-less Debug*
  - a. The F29H850TU9.ccxml file can be found in the TargetConfig folder of the project
2. Connect to CPU1
3. Verify that the device is in either bank mode 1 or bank mode 3
  - a. SSU\_GEN\_REGS.BANKMODE is 0x6 (mode 1) or 0xC (mode 3)
  - b. Right-click CPU1 -> Properties
    - i. Category drop-down menu -> Flash Settings
    - ii. Make sure bank mode 1 or 3 is selected
    - iii. Only select Program BANKMGMT if the device is not in one of these two modes already. If not, then program the BANKMGMT and issue a device reset
4. Select Run -> Load -> Load Program -> flash\_based\_uart\_sbl\_with\_fota.out

*If users want to program the flash-based SBL to the inactive address space, then make sure the FOTA enable checkbox is checked in the flash plugin before loading.*



**Figure 5-1. FOTA Enable Checkbox in Flash Plugin**

Once these steps are completed, flash-based SBL is loaded to flash and is ready to execute.

#### 5.1.2 Loading via UART Boot and the UART Flash Kernel

In an environment where CCS is not used to load the flash of the F29H85x, the UART flash kernel can be used to program the flash-based SBL. Refer to the [Serial Flash Programming of F29H85x™ application note](#) for details about the UART flash kernel. At a high-level, the kernel acts as a secondary bootloader that enables flash programming or firmware updates over UART.

An important detail is that the UART flash kernel only supports programming CPU1 firmware in bank mode 0 (for now, this is updated in the future). Thus, this cannot be used to place the flash-based SBL code and application in both the active and inactive regions of flash for bank mode 1 or bank mode 3. In a scenario where the flash-based SBL needs to reside in both the active and inactive regions, the UART kernel can be used to program the active region and the flash-based SBL can be used to program the inactive region. The flow required to have flash-based SBL and an application in CPU1 active or inactive flash in bank mode 1 is as follows:

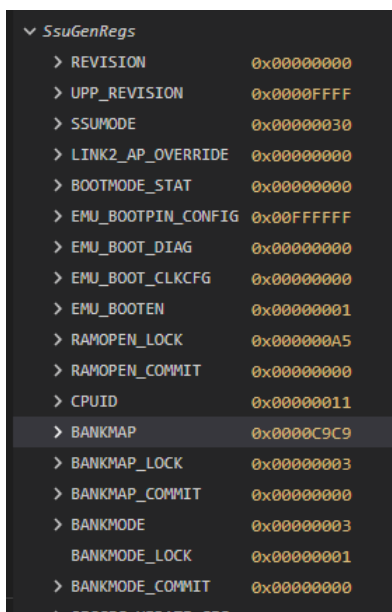
1. Put the device in UART boot mode and make sure device is in bank mode 0
2. Trigger a device reset
3. Send the UART flash kernel to F29H85x using the `uart_host_programmer.exe`
4. Once completed, send the `flash_based_uart_sbl_with_fota.bin` file to the device
  - a. Flash-based SBL is programmed in CPU1 flash
5. Put the device in bank mode 1 and configure flash boot mode
6. Trigger a device reset
  - a. Device boots to flash-based SBL
7. Send the `flash_based_uart_sbl_with_fota.bin` file to the device using `uart_host_programmer.exe`
  - a. `flash_based_uart_sbl_with_fota.bin` is programmed to the inactive region of CPU1 flash

### Note

`uart_host_programmer_appln.exe` is used for step 7 if the COM port needs to be reinitialized after the device reset.

## 5.2 Example UART Loading Process

- Verify the device is in bank mode 0



SSuGenRegs	
> REVISION	0x00000000
> UPP_REVISION	0x0000FFFF
> SSUMODE	0x00000030
> LINK2_AP_OVERRIDE	0x00000000
> BOOTMODE_STAT	0x00000000
> EMU_BOOTPIN_CONFIG	0x00FFFFFF
> EMU_BOOT_DIAG	0x00000000
> EMU_BOOT_CLKCFG	0x00000000
> EMU_BOOTEN	0x00000001
> RAMOPEN_LOCK	0x000000A5
> RAMOPEN_COMMIT	0x00000000
> CPUID	0x00000011
> BANKMAP	0x0000C9C9
> BANKMAP_LOCK	0x00000003
> BANKMAP_COMMIT	0x00000000
> BANKMODE	0x00000003
> BANKMODE_LOCK	0x00000001
> BANKMODE_COMMIT	0x00000000
> SEC_CFG_UPDATE_CFG	0x00000000

**Figure 5-2. SSU\_GEN\_REG.BANKMODE Showing Bank Mode 0**

- Put the device in UART Boot Mode (GPIO 72 -> 0 and GPIO 84 -> 1)
- Trigger a device reset
- Send the kernel with `uart_flash_programmer.exe`
  1. `.\uart_flash_programmer.exe -d f29h85x -p COM89 -k .\ex3_uart_flash_kernel.cert.bin -a1 .\flash_based_uart_sbl_with_fota.bin`
- The kernel is running at this point

```

Recvd 512 bytes, port elapsed for 47 ms, 0 left to validate, 6114 bytes remain
Wrote 512 bytes, port elapsed for 1 ms, 512 left to validate, 5602 bytes remain
Recvd 512 bytes, port elapsed for 45 ms, 0 left to validate, 5602 bytes remain
Wrote 512 bytes, port elapsed for 1 ms, 512 left to validate, 5090 bytes remain
Recvd 512 bytes, port elapsed for 45 ms, 0 left to validate, 5090 bytes remain
Wrote 512 bytes, port elapsed for 1 ms, 512 left to validate, 4578 bytes remain
Recvd 512 bytes, port elapsed for 45 ms, 0 left to validate, 4578 bytes remain
Wrote 512 bytes, port elapsed for 1 ms, 512 left to validate, 4066 bytes remain
Recvd 512 bytes, port elapsed for 47 ms, 0 left to validate, 4066 bytes remain
Wrote 512 bytes, port elapsed for 1 ms, 512 left to validate, 3554 bytes remain
Recvd 512 bytes, port elapsed for 46 ms, 0 left to validate, 3554 bytes remain
Wrote 512 bytes, port elapsed for 1 ms, 512 left to validate, 3042 bytes remain
Recvd 512 bytes, port elapsed for 45 ms, 0 left to validate, 3042 bytes remain
Wrote 512 bytes, port elapsed for 1 ms, 512 left to validate, 2530 bytes remain
Recvd 512 bytes, port elapsed for 46 ms, 0 left to validate, 2530 bytes remain
Wrote 512 bytes, port elapsed for 1 ms, 512 left to validate, 2018 bytes remain
Recvd 512 bytes, port elapsed for 47 ms, 0 left to validate, 2018 bytes remain
Wrote 512 bytes, port elapsed for 1 ms, 512 left to validate, 1506 bytes remain
Recvd 512 bytes, port elapsed for 45 ms, 0 left to validate, 1506 bytes remain
Wrote 512 bytes, port elapsed for 1 ms, 512 left to validate, 994 bytes remain
Recvd 512 bytes, port elapsed for 45 ms, 0 left to validate, 994 bytes remain
Wrote 512 bytes, port elapsed for 1 ms, 512 left to validate, 482 bytes remain
Recvd 512 bytes, port elapsed for 47 ms, 0 left to validate, 482 bytes remain
Wrote 482 bytes, port elapsed for 1 ms, 482 left to validate, 0 bytes remain
Recvd 482 bytes, port elapsed for 44 ms, 0 left to validate, 0 bytes remain

```

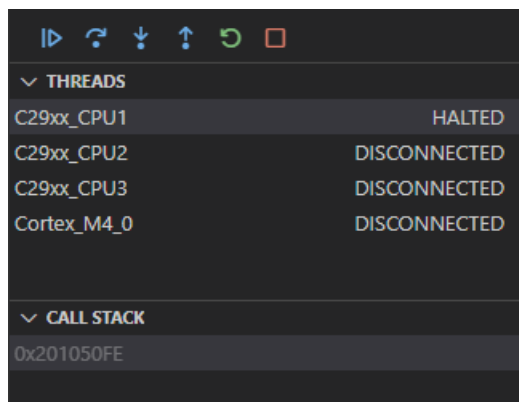
Data bandwidth is 89507.829 bits per second (bps)  
Kernel successfully sent!

What operation do you want to perform?

- 1-DFU CPU1 (HS-FS only)
- 2-DFU CPU3 (HS-FS only)
- 3-Load HSMRt Image (Prerequisite for KP & CP)
- 4-Load HSM Keys (HS-KP Key Provision)
- 5-Program Sec Cfg (HS-SE Code Provision)
- 6-Load HSM Image (HS-SE Code Provision)
- 7-Load C29 CPU1 Image (HS-SE Code Provision)
- 8-Load C29 CPU3 Image (HS-SE Code Provision)
- 9-Run CPU1
- 10-Run CPU3
- 11-Sync (Verify kernel or Boot Manager is running)
- 12-Reset CPU1
- 0-Done

**Figure 5-3. Host Output After Sending Kernel**

- Can verify that the kernel is running by connecting to CPU1 and observing that the PC is in LPAX RAM



**Figure 5-4. Kernel Running in RAM**

- Resume kernel execution
- Send DFU CPU1 command to download the flash-based SBL

```
Wrote 48 bytes, port elapsed for 0 ms, 512 left to validate, 1696 bytes remain
Recv 464 bytes, port elapsed for 40 ms, 48 left to validate, 1744 bytes remain
Wrote 464 bytes, port elapsed for 1 ms, 512 left to validate, 1232 bytes remain
Recv 48 bytes, port elapsed for 4 ms, 464 left to validate, 1696 bytes remain
Wrote 48 bytes, port elapsed for 0 ms, 512 left to validate, 1184 bytes remain
Recv 464 bytes, port elapsed for 41 ms, 48 left to validate, 1232 bytes remain
Wrote 464 bytes, port elapsed for 1 ms, 512 left to validate, 720 bytes remain
Recv 48 bytes, port elapsed for 4 ms, 464 left to validate, 1184 bytes remain
Wrote 48 bytes, port elapsed for 0 ms, 512 left to validate, 672 bytes remain
Recv 464 bytes, port elapsed for 40 ms, 48 left to validate, 720 bytes remain
Wrote 464 bytes, port elapsed for 1 ms, 512 left to validate, 208 bytes remain
Recv 48 bytes, port elapsed for 4 ms, 464 left to validate, 672 bytes remain
Wrote 48 bytes, port elapsed for 1 ms, 512 left to validate, 160 bytes remain
Recv 464 bytes, port elapsed for 41 ms, 48 left to validate, 208 bytes remain
Wrote 160 bytes, port elapsed for 1 ms, 208 left to validate, 0 bytes remain
Recv 48 bytes, port elapsed for 4 ms, 160 left to validate, 160 bytes remain
Recv 160 bytes, port elapsed for 14 ms, 0 left to validate, 0 bytes remain

Data bandwidth is 91065.712 bits per second (bps)
Application successfully sent!

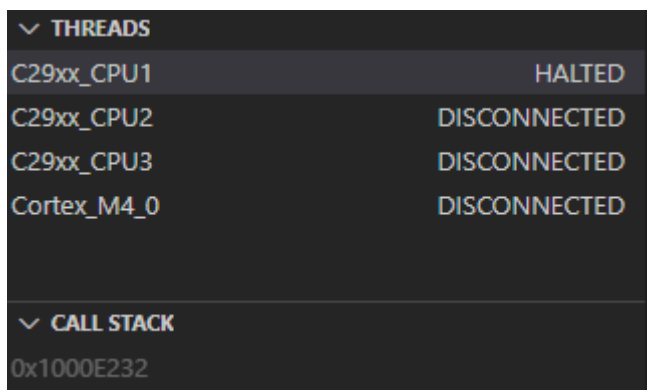
Command packet SUCCESS

Waiting for logging messages...
Command packet SUCCESS
Received message:
CPU1 image has been written to flash, please proceed to Run CPU1 or Reset CPU1(in flash boot mode) to start the new firm-
ware.

What operation do you want to perform?
1-DFU CPU1 (HS-FS only)
2-DFU CPU3 (HS-FS only)
3-Load HSMRt Image (Prerequisite for KP & CP)
4-Load HSM Keys (HS-KP Key Provision)
5-Program Sec Cfg (HS-SE Code Provision)
6-Load HSM Image (HS-SE Code Provision)
7-Load C29 CPU1 Image (HS-SE Code Provision)
8-Load C29 CPU3 Image (HS-SE Code Provision)
9-Run CPU1
10-Run CPU3
11-Sync (Verify kernel or Boot Manager is running)
12-Reset CPU1
0-Done
```

**Figure 5-5. Host Output of DFU CPU1 Command**

- Connect to CPU1 and program bank mode 1
- Configure flash boot mode (GPIO 72 -> 1 GPIO 84 -> 1)
- Trigger a device reset
- Can verify that flash-based SBL is now running by connecting to CPU1 and seeing that the PC is running in flash



**Figure 5-6. Device Successfully Boots to Flash**

- Program flash-based SBL to the inactive region of flash by sending the DFU CPU1 command

```

Recv 512 bytes, port elapsed for 46 ms, 0 left to validate, 3864 bytes remain
Wrote 512 bytes, port elapsed for 1 ms, 512 left to validate, 3352 bytes remain
Recv 512 bytes, port elapsed for 47 ms, 0 left to validate, 3352 bytes remain
Wrote 512 bytes, port elapsed for 2 ms, 512 left to validate, 2840 bytes remain
Recv 512 bytes, port elapsed for 46 ms, 0 left to validate, 2840 bytes remain
Wrote 512 bytes, port elapsed for 1 ms, 512 left to validate, 2328 bytes remain
Recv 512 bytes, port elapsed for 46 ms, 0 left to validate, 2328 bytes remain
Wrote 512 bytes, port elapsed for 1 ms, 512 left to validate, 1816 bytes remain
Recv 512 bytes, port elapsed for 45 ms, 0 left to validate, 1816 bytes remain
Wrote 512 bytes, port elapsed for 1 ms, 512 left to validate, 1304 bytes remain
Recv 512 bytes, port elapsed for 45 ms, 0 left to validate, 1304 bytes remain
Wrote 512 bytes, port elapsed for 1 ms, 512 left to validate, 792 bytes remain
Recv 512 bytes, port elapsed for 46 ms, 0 left to validate, 792 bytes remain
Wrote 512 bytes, port elapsed for 1 ms, 512 left to validate, 280 bytes remain
Recv 512 bytes, port elapsed for 46 ms, 0 left to validate, 280 bytes remain
Wrote 280 bytes, port elapsed for 1 ms, 280 left to validate, 0 bytes remain
Recv 280 bytes, port elapsed for 27 ms, 0 left to validate, 0 bytes remain

Data bandwidth is 89589.076 bits per second (bps)
Application successfully sent!

Command packet SUCCESS

Waiting for logging messages...
Command packet SUCCESS
Received message:
CPU1 image has been written to flash, please proceed to Run CPU1 or Reset CPU1(in flash boot mode) to start the new firm
ware.

What operation do you want to perform?
1-DFU CPU1 (HS-FS only)
2-DFU CPU3 (HS-FS only)
3-Load HSMRT Image (Prerequisite for KP & CP)
4-Load HSM Keys (HS-KP Key Provision)
5-Program Sec Cfg (HS-SE Code Provision)
6-Load HSM Image (HS-SE Code Provision)
7-Load C29 CPU1 Image (HS-SE Code Provision)
8-Load C29 CPU3 Image (HS-SE Code Provision)
9-Run CPU1
10-Run CPU3
11-Sync (Verify kernel or Boot Manager is running)
12-Reset CPU1
0-Done

```

**Figure 5-7. Host Programmer Output of DFU CPU1 Command**

- Can verify that flash-based SBL is programmed to the inactive flash region with the memory browser

Location	0x10601000	Format	8-Bit Hex - TI Style	⌕	⌕
0x10601000	00 90 30 0C 7F 11 52 0B 7F 11 52 0B C0 10 68 00 00 00 C0 1D 52 8C 20 30 AC 0F 34 9A 00 00 00 00 44 0A 70 00 08 30				
0x10601026	62 37 04 00 C3 3B 03 00 B8 3C 43 08 43 24 00 00 DD 33 25 00 DD 33 04 00 F0 34 08 01 32 3C 03 00 42 24 09 00 F1 34				
0x1060104C	D1 02 42 24 06 00 F1 34 D5 01 43 24 03 00 F1 34 E2 03 D9 33 9C 18 C2 20 00 00 E9 34 0D 00 D7 33 A2 18 00 32 C4 18				
0x10601072	47 0A 01 00 20 10 8A 32 C7 00 E9 34 CF 03 D9 33 95 18 C6 22 83 00 E9 34 0D 00 D7 33 A2 1C 00 32 E4 1C 48 0A 01 00				
0x10601098	80 10 8A 32 E8 00 E9 34 BE 03 C6 22 87 00 E9 34 0D 00 D7 33 A2 18 00 32 C4 18 47 0A 01 00 E4 10 8A 32 C7 00 E9 34				
0x106010BE	AF 03 46 0A 00 00 EC 10 8A 32 86 00 E9 34 0D 00 D7 33 A2 18 00 32 C4 18 47 0A 01 00 F0 10 8A 32 C7 00 E9 34 9D 03				
0x106010E4	D9 33 96 18 C6 22 43 00 E9 34 0D 00 D7 33 A2 18 00 32 C4 18 47 0A 01 00 C4 10 8A 32 C7 00 E9 34 8C 03 46 0A 00 00				
0x1060110A	F8 10 8A 32 86 00 E9 34 0D 00 D7 33 A2 18 00 32 C4 18 47 0A 01 80 F8 10 8A 32 C7 00 E9 34 7A 03 46 0A 00 00 D8 10				

**Figure 5-8. FLC1.B2 and FLC1.B3 After Firmware Programmed to Inactive Flash Region**

At this point, the flash-based SBL and FOTA\_Example\_Application have been programmed to both the active and inactive regions of CPU1 flash memory.

## 6 FAQ

Answers to some common questions and issues encountered by users when utilizing the flash-based UART SBL are listed below.

### 6.1 General

**Question:** Can users program the data flash while performing a FOTA upgrade?

**Answer:** The data flash (FLC1.B4) can be programmed while executing program code in FLC1.B0 and FLC1.B1 or FLC1.B2 and FLC1.B3. However, do not attempt to simultaneously program FLC1.B4 while another bank pair within FLC1 is being programmed. For example, if the device is performing a FOTA upgrade in FLC1.B2 and FLC1.B3 and executing program code in FLC1.B0 and FLC1.B1, then programming of FLC1.B2 and FLC1.B3 must not be interrupted with the intent of programming the data flash (FLC1.B4). If users are programming the data flash in an ISR during this FOTA scenario, then the ISR verifies that there is not an ongoing flash operation (program or erase). If so, then the ISR must wait until the previous command has completed before attempting another operation. The Fapi\_checkFsmForReady() function of the F29H85x Flash API can be used for this. There is a caveat such that after programming to the data flash in the ISR, these must set the protection masks back to 0x0 (or whatever is being used to program the main array [FLC1.B2 and FLC1.B3]). This is because after a successful program or erase operation, the protection masks are set back to 0xFFFF\_FFFF (enabling write and erase protection for all sectors). If the interrupt occurs after the main array protection mask configuration, but before the program command is issued, then this is a flash state machine error unless reconfigured.

**Question:** Is a firmware upgrade necessary every time the bank mode is changed?

**Answer:** Users are free to switch the device bank mode at any point (before or after a firmware upgrade), but users have to account for any changes in the flash memory map. For example, when changing from bank mode 1 to bank mode 3, CPU1 no longer has 2MB of flash allocated to the active region. Instead, CPU1 only has 1MB available for the active region. Thus, any active firmware that is in flash no longer allocated to CPU1 cannot be executed after the bank mode switch and a firmware upgrade needs to be performed.

**Question:** If users cannot find the flash-based UART SBL with FOTA project, then?

**Answer:**

Device	Build Configurations	Location
F29H85x	BANKMODE_1 BANKMODE_3 BANKMODE_1_CP BANKMODE_3_CP	mcu_sdk_f29h85x\examples\driverlib\single_core\flash\flash_based_UART_SBL_with_FOTA

### 6.2 Application Load

**Issue:** Users cannot successfully load the application to flash.

**Answer:**

- Make all sections in the linker cmd file that are allocated to flash are aligned to 512-bit boundaries. This can be done by adding `palign(32)` to the appropriate sections as shown below.

```
.text : {} > FLASH_RP0, palign(32)
```

- Make sure there is an appropriate X.509 certificate in the first 0x1000 bytes of the binary file.

**Issue:** There is an error in the host programmer when performing a CPU3 FOTA Upgrade.

**Answer:**

- Make sure the device is in bank mode 3 and build the flash-based SBL with the BANKMODE\_3 build configuration selected.

**Issue:** There is an error in the host programmer when performing a secure firmware upgrade.

**Answer:**

- Make sure the device is in the HS-SE state and build the flash-based SBL with the BANKMODE\_1\_CP or BANKMODE\_3\_CP build configuration.

**Issue:** The UART communication is failing.

**Answer:**

- Make sure to use the COM port connected to the proper UART transceiver and double-check the UART GPIO configuration and setup

## 7 Summary

As applications grow in complexity, the need to fix bugs, add features, and otherwise modify embedded firmware is increasingly critical in end applications. Enabling functionality like this can be easily and inexpensively accomplished through the use of boot-loaders.

This application note aims to solve this problem by the introduction of a secondary boot-loader (SBL). This document discusses the specifics of the SBL and the host application tool found in the F29H85x MCU SDK.

## 8 References

1. Texas Instruments: [F29H85x and F29P58x Real-Time Microcontrollers](#), technical reference manual
2. Texas Instruments, [F29H85x and F29P58x Real-Time Microcontrollers](#), data sheet

## 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 © 2025, Texas Instruments Incorporated