Application Note

Live Firmware Update With Device Reset on C2000™ MCUs

Sira Rao, Baskaran Chidambaram, Alex Wasinger, and Matt Kukucka

ABSTRACT

This document presents details on Live Firmware Update (LFU) with Device Reset on devices with two Flash banks, detailing the challenges involved and suggestions on how to address them. For simplicity of illustration, an LED-based example is used (included as part of C2000ware).

Table of Contents

List of Figures

Trademarks

C2000™ Code Composer Studio™ are trademarks of Texas Instruments.
All trademarks are the property of their respective owners.

1 Introduction

In applications like server power supply, metering, and so forth. the system is desired to be run continuously to reduce downtime. But typically during firmware upgrades due to bug fixes, new features, and/or performance improvements, the system is removed from service causing downtime for associated entities as well. This can be handled with redundant modules but with increase in total system cost. An alternate approach, called Live Firmware Update (LFU), allows firmware to be updated while the system is still operating. Switching to new firmware can be done either with or without resetting the device, with the latter being more complex.

2 Resources Required for LFU

LFU is feasible when the device has enough resources of various kinds – CPU bandwidth, Memory, and Peripheral availability:

- CPU Bandwidth – The new firmware has to be transferred using a communication peripheral and written to flash memory while the application is still operating. This means CPU need to have enough available bandwidth to support LFU.
- Memory – The non-volatile memory that is used here is Flash memory. Flash read and write operations cannot be simultaneously performed on the same Flash bank. However read and write operations can be simultaneously performed on different Flash banks. Hence, the ideal scenario is for the device to contain dual Flash banks. In devices with single Flash banks, LFU is particularly challenging, but still feasible provided:
  - The complete application code (or a portion of it that controls the output(s) the user is interested in) and Flash APIs need to run from RAM memory while the new firmware is updated to Flash. This means there should be enough RAM memory that can be utilized.
  - Some devices support Flash APIs in ROM; in those devices the application code can run from RAM and Flash APIs can run from ROM memory, thus reducing the RAM requirement.
- Peripheral Availability – A spare communication peripheral using the new image can be transferred from the host to the device.

LFU is easier to implement in devices with multiple Flash banks. In this document and the reference example, it is assumed that the device has 2 Flash banks. The device considered is TMS320F28004x, which has dual flash banks. Their address space is 64K x 16 each, with addresses ranging from 0x80000-0x8FFFF, and 0x90000-0x9FFFF.

3 Memory Layout

The Flash memory has been partitioned as shown in Figure 3-1.

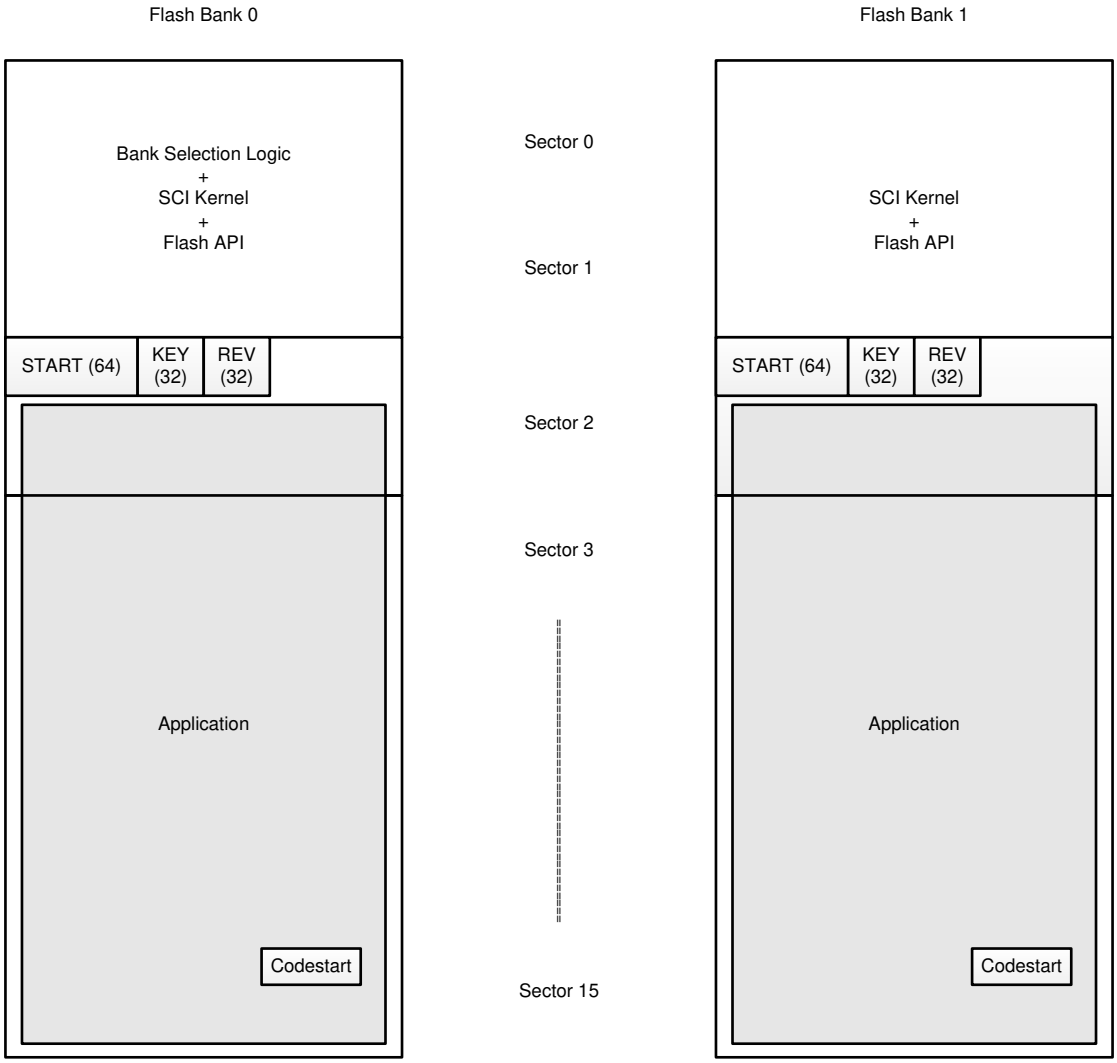

Figure 3-1. Flash Memory Contents for Bank 0 and Bank 1

Assuming each Flash bank has 16 sectors, two sectors have been allocated to Static code (code that will not change between applications). This is described in Section 4.

A few locations in sector 2 have been reserved to store the below entries:

- START – when this 64-bit field is set in a specific Flash bank (to 0x5A5A5A5A5A5A5A5A) by the Serial Communications Interface (SCI) Flash Kernel, it indicates that the corresponding Flash bank has been erased (Application sectors) and programming/verification is about to begin. In this example, the START field is located at addresses 0x82000 in BANK0 and 0x92000 in BANK1.

- REV – this represents a 32-bit Firmware revision number that is set by the SCI Flash Kernel which is used by the Bank Selection Logic to determine which the latest image is between Flash bank 0 and 1. REV starts at 0xFFFFFFFF and is decremented on each subsequent Flash programming cycle. Thus, the Bank with the lower revision number is considered the latest one. Firmware revision field is handled by the flash kernel for simplicity. In practice, the last downloaded image may not be the latest firmware version and the application image would contain the firmware version, but that assumption is made with this model where the kernel updates the REV field. In this example, the REV field is located at address 0x82006 in BANK0 and 0x92006 in BANK1. As an example, if the REV in BANK0 is FFFF FFFA, and the REV in BANK1 is FFFF FFF9, BANK1 will be considered the latest firmware and will execute.
- KEY– The image in a bank is considered as valid if this location contains a valid KEY (0x5B5B5B5B). This Key is written to by the SCI Flash Kernel, and is read and tested by the Bank Selection Logic. In this example, the KEY field is located at addresses 0x82004 in BANK0 and 0x92004 in BANK1.

The rest of sector 2 and sectors 3-15 can be used to store the application image. This allows the static code in sectors 0 and 1 to be programmed once, and remain unchanged during LFU.

4 Static Code in LFU

Static code that is used to support LFU consists of:

- Bank Selection Logic – when there are two (or more) Flash banks containing application firmware, logic that determines which Flash bank to boot is necessary. A common implementation of this logic centers on a firmware revision number. As described, the lower revision number is the latest image in this Example. Bank selection logic is placed at the default flash boot address (0x80000 in F28004x), so that once Boot ROM code completes execution, execution will transfer to bank selection logic. Bank selection logic is only included in Bank0, not in Bank1.
- Flash Kernel – it is the job of the Flash kernel to receive the Firmware image from the host using a peripheral, and call the Flash programming APIs to write it to flash memory. In this document, the SCI flash Kernel is used since the SCI peripheral is used to transfer the new firmware image. The detailed step by step flow is documented in the file header of flashapi_ex2_ldfu.c.
  - In a blank device (where application firmware is not present in either flash banks) the bank selection logic will identify there is no valid KEY in either Flash bank, and will wait for an LFU command over SCI. This will use the Flash Kernel to update the firmware on to bank1 as the kernel code is first programmed to bank0, and therefore executes from bank0.
  - If one or both banks contains a valid application, bank selection logic will transfer control to the code entry point (codestart) of the corresponding bank. In this example, the codestart address is 0x8EFF0 for Bank0 and 0x9EFF0 for Bank1.
  - During LFU, the application will make a call to the Flash Kernel to receive and update the firmware.
- Flash API – Flash APIs provide interfaces to erase and program Flash memory. These APIs need to run from the bank which is NOT being updated.

The static code is configured as a single example - flashapi_ex2_sci_kernel project (included in C2000Ware at <C2000Ware>\driverlib\f28004x\examples\flash). This example supports multiple build configurations, of which those relevant to LFU are listed below:

- BANK0_LDFU - Links the bank selection logic and flash kernel to Bank 0 (addresses 0x80000 - 0x81FFF). Uses Flash API symbols in flash.
- BANK0_LDFU_ROM - Links the bank selection logic and flash kernel to Bank 0 (addresses 0x80000 - 0x81FFF). Uses Flash API symbols in ROM; Rev A of F28004x cannot be used with this build configuration, since it does not support Flash APIs in ROM.
- BANK1_LDFU - Links the flash kernel to Bank 1 (0x90000 - 0x91FFF). Uses Flash API symbols in flash.
- BANK1_LDFU_ROM - Links the flash kernel to Bank 1 (0x90000 - 0x91FFF). Uses Flash API symbols in ROM; Rev A of F28004x cannot be used with this build configuration, since it does not support Flash APIs in ROM.

For more details, see the flashapi_ex2_sciKernel.c in the flashapi_ex2_sci_kernel project (included in C2000Ware at <C2000Ware>\driverlib\f28004x\examples\flash).

5 LED Example Application and LFU Flow

This example (flashapi_ex3_live_firmware_update project) is designed to demonstrate LFU with the LED blinking periodically. This is achieved using the Live Device Firmware Update (Live DFU or LDFU) command, which is part of SCI kernel. This is to be used with the Serial Flash Programmer (PC tool).

In this example, an SCI auto baud lock is performed and the byte used for auto baud lock is echoed back. Two interrupts are initialized and enabled: SCI Rx FIFO interrupt and CPU Timer 0 interrupt. The CPU Timer 0 interrupt occurs every 1 second; the interrupt service routine (ISR) for CPU Timer 0 toggles an LED based on the build configuration that is running.

- LED1 is toggled with the BANK0_FLASH build configuration. “BANK0” is a pre-defined symbol in this build configuration, and when this symbol is defined, the project sets up GPIOs associated with LED1.
- LED2 is toggled with the BANK1_FLASH build configuration. “BANK1” is a pre-defined symbol in this build configuration, and when this symbol is defined, the project sets up GPIOs associated with LED2.

The application images generated by building the above build configurations are the ones that will be used to illustrate LFU in this document. Note that other than the changes described above between the two build configurations, there are no other differences between the two application images. Hence, this is a relatively simple example for LFU illustration.

The SCI Rx FIFO interrupt is set for a FIFO interrupt level of 10 bytes. The number of bytes in a packet from the Serial Flash Programmer (when using the LDFU command) is 10. When a command is sent to the device from the Serial Flash Programmer, the SCI Rx FIFO ISR receives a command from the 10 byte packet in the FIFO. If the command matches the Live Device Firmware Update (Live DFU) command, then the code branches to the Live DFU function (liveDFU()) located inside of the SCI Flash Kernel (flashapi_ex2_ldfu.c) for the corresponding bank. So if the application on Bank0 is executing, control will pass to liveDFU() on Bank0, located at 0x81000. If the application on Bank1 is executing, control will pass to liveDFU() on Bank1, located at 0x91000. Within this function, execution passes to the ldfuLoad() function in order to erase the appropriate bank, load a hex formatted program (in the appropriate SCI boot format) into flash, and verify the program. Then the watchdog is configured for a reset. At the end, the watchdog is enabled in order for a reset to occur. When the device resets, it boots and loads the new Firmware.

Figure 5-1 depicts the flow of the code at a high level after the code enters main() of the application. For more details, see the flashapi_ex3_live_firmware_update.c located in the flashapi_live_firmware_update project (included in C2000Ware at <C2000Ware>\driverlib\f28004x\examples\flash).

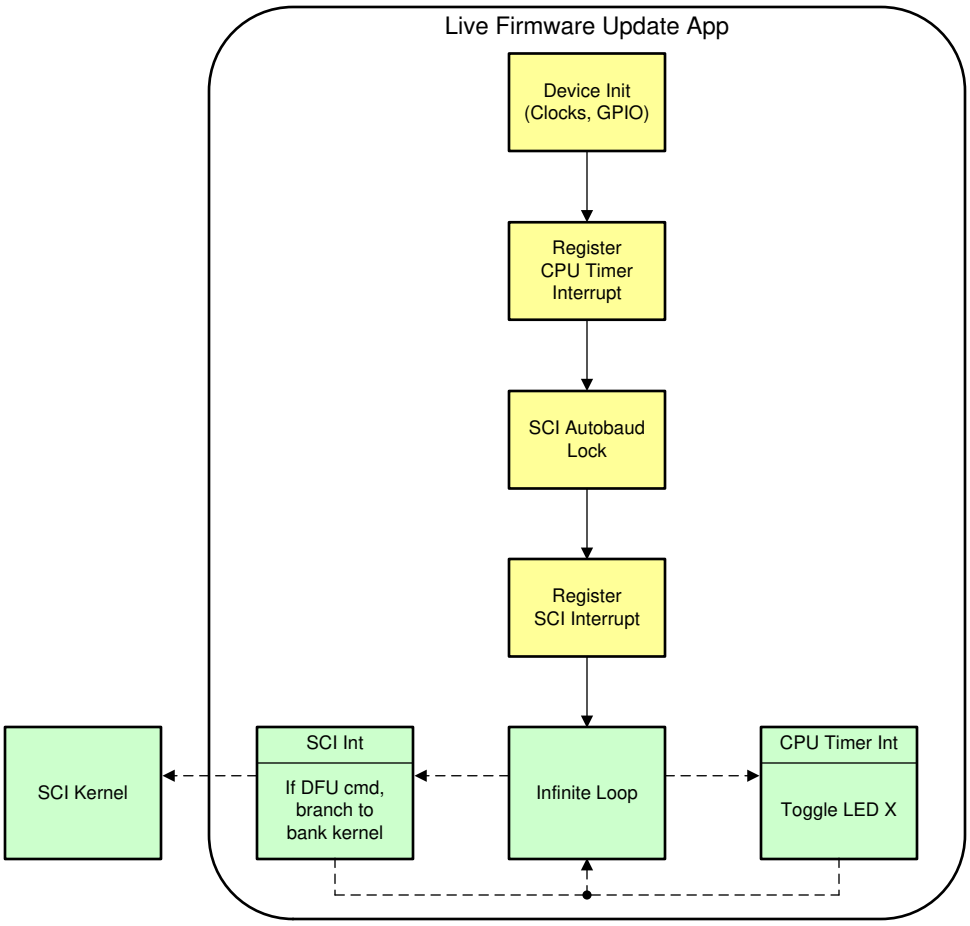

Figure 5-1. Code Flow After Entering main() of Application

6 Running the LED Example

6.1 Serial Flash Programmer Update

The serial flash programmer (serial_flash_programmer.exe) supplied with C2000ware takes both the kernel and application image as parameters. Typically, the kernel is transferred first over to the SCI bootloader and executed from RAM or Flash on the device. The kernel program then takes the application image over SCI (from serial programmer running on PC) and programs the application image in flash memory.

In the case of LFU, the static content including flash kernel is first programmed to flash sectors 0 and 1 of Flash banks 0 and 1. This is described in Section 6.2. After this, the serial flash programmer needs to be modified to transfer only the application image. This can be done by commenting the line “#define kernel” in serial_flash_programmer.cpp. The serial flash programmer can be regenerated by compiling the project in Visual C (called serial_flash_programmer_appln.exe). The pre-built executable is placed at <C2000Ware>\utilities\flash_programmers\serial_flash_programmer\). Thus, the user needs to take no action here.

6.2 Programming Static Code – Loading via Code Composer Studio™ (CCS)

The hardware used in the illustrated steps is an F28004x ControlCARD on a ControlCARD docking station Rev4.1. If a JTAG connection is available, then CCS can be used to load Flash banks 0 and 1 with static code. Note:

Note

Make sure you have the settings in CCS (or your target configuration file – by right clicking and selecting properties) as shown in Figure 6-1 before loading the images. Build the flashapi_ex2_sci_kernel project in both BANK0_LDFU and BANK1_LDFU configurations, and load each to target. The CCS flash plugin will load the contents on to flash.

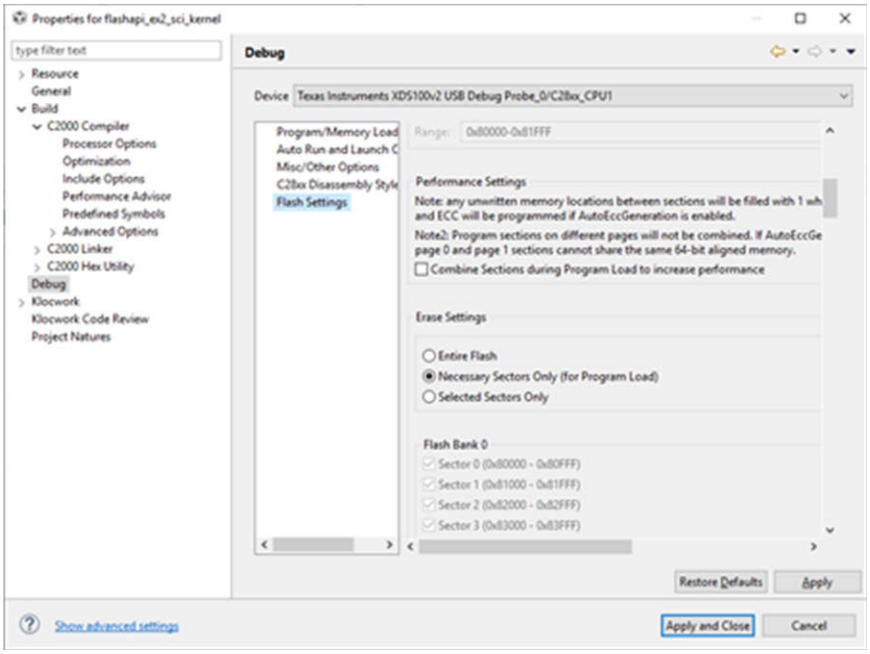

Figure 6-1. Flash Settings to Only Erase Necessary Sectors

1. Load BANK0 static image (BANK0_LDFU of flashapi_ex2_sci_kernel project).

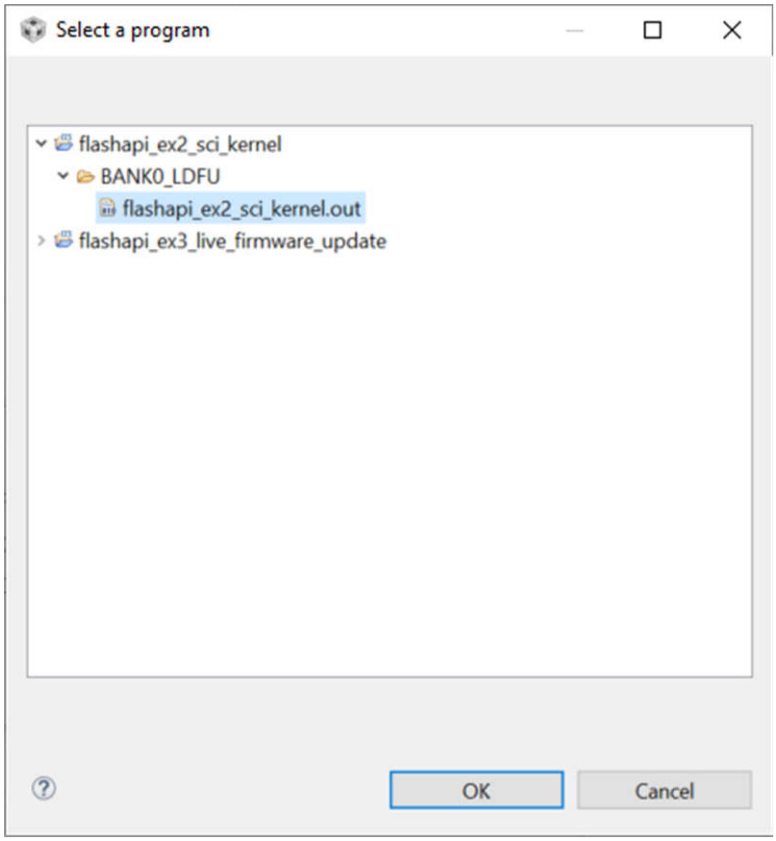

Figure 6-2. Selecting Kernel to Load to Flash Bank 0

After the static contents are loaded on BANK0, the first thing that happens is the execution of the bank selection logic that will determine that application firmware is not programmed in either bank.

Control will pass to the flash kernel, which will be ready to program the application in BANK1. The CCS view will look Figure 6-3 (the program will not stop at main(), but will be running, awaiting a SCI command):

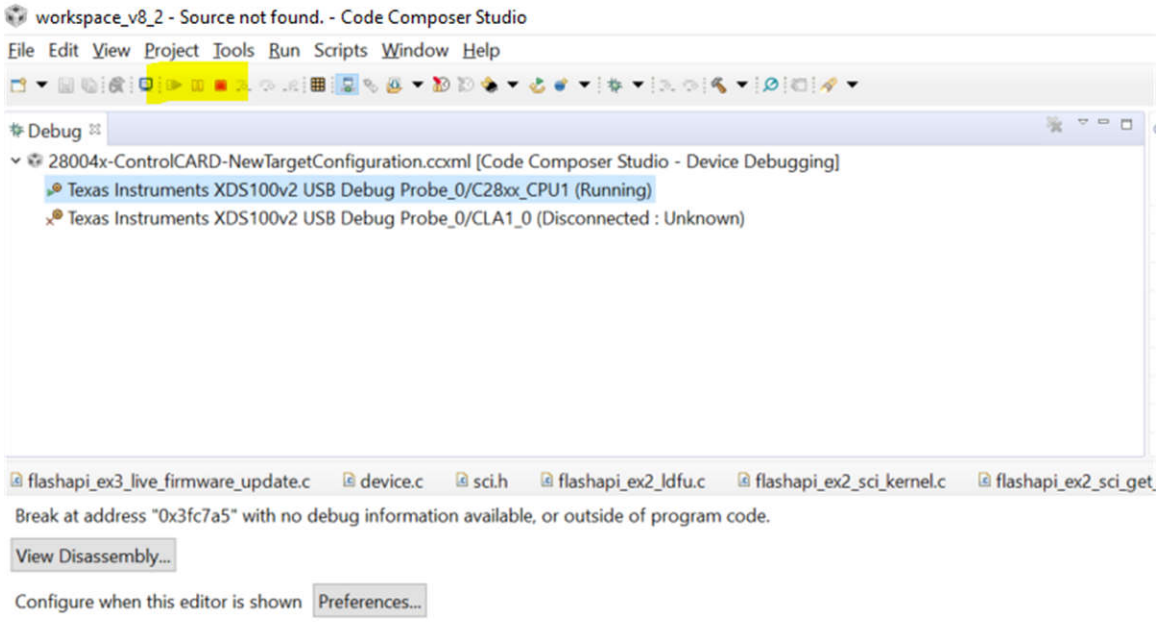

Figure 6-3. CCS Window view After Programming Bank 0 Flash Kernel

2. Verify the Kernel content of BANK0 by opening the Memory Browser window in CCS, and entering address 0x80000.

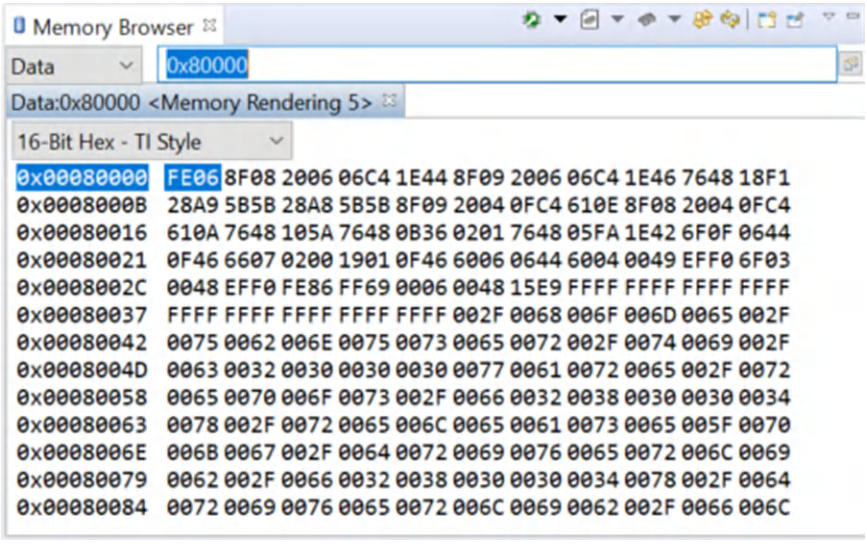

Figure 6-4. CCS Memory Browser View to Verify Successful Kernel Programming of Bank 0

3. Now switch to the Windows command prompt, and execute the command below: serial_flash_programmer_appln.exe -d f28004x -k f28004x_fw_upgrade_example\flashapi_ex2_sci_kernel.txt -a flashapi_ex3_live_firmware_updateBANK1FLASH.txt -b 9600 -p COMx where x = COM port corresponding to the JTAG connection between the PC and the target board. The COM port number can be found by looking up Ports in Device Manager. This will be populated by virtue of the fact that there is a USB cable connected between the target and the computer, which provides both JTAG and SCI functionality. The example uses a baud rate of 9600. flashapi_ex3_live_firmware_updateBANK1FLASH.txt is generated by building the CCS project flashapi_ex3_live_firmware_update in build configuration BANK1_FLASH.

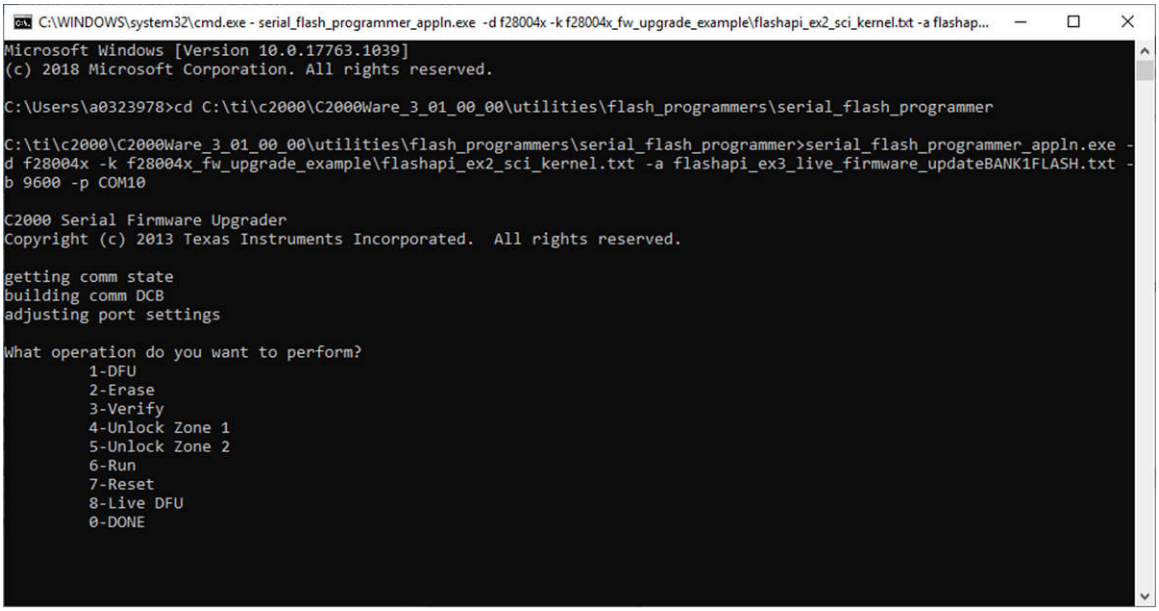

Figure 6-5. LFU Serial Command Invoked From Windows Command Prompt

4. Once LDFU (8) command is selected, the kernel will receive and program the application in BANK1. The application size is about 36KB. Download time will be about 30 seconds.

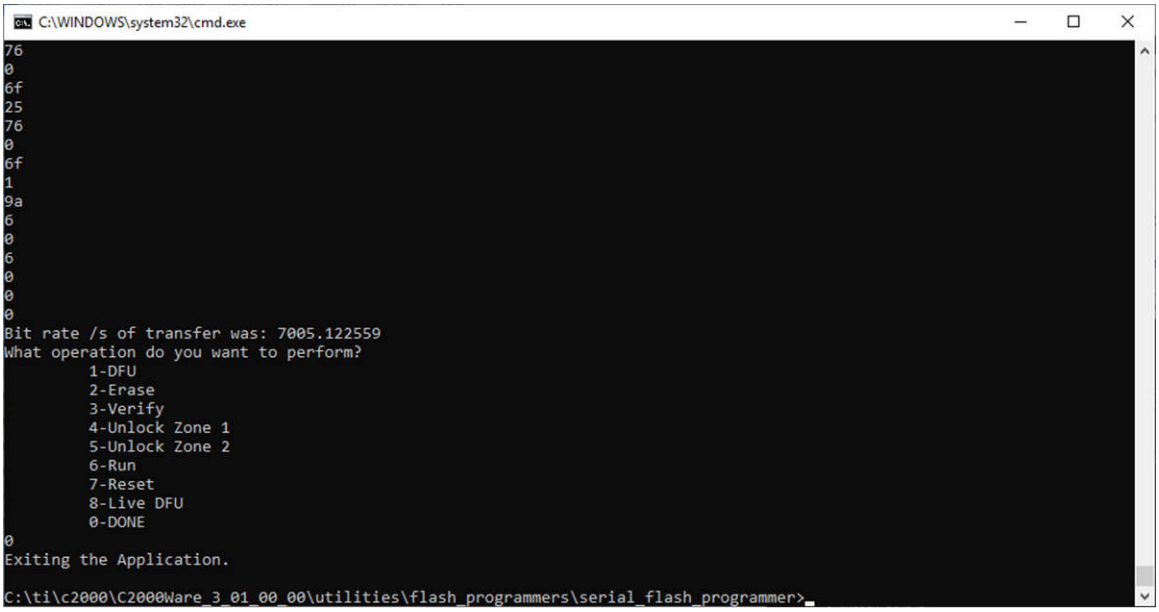

Figure 6-6. Successful Completion of LFU Command to Program Flash Bank 1

5. After transfer is complete, enter 0 to indicate end of command operations. Verify the Application content of BANK1 by opening the Memory Browser window in CCS, and entering address 0x92000.

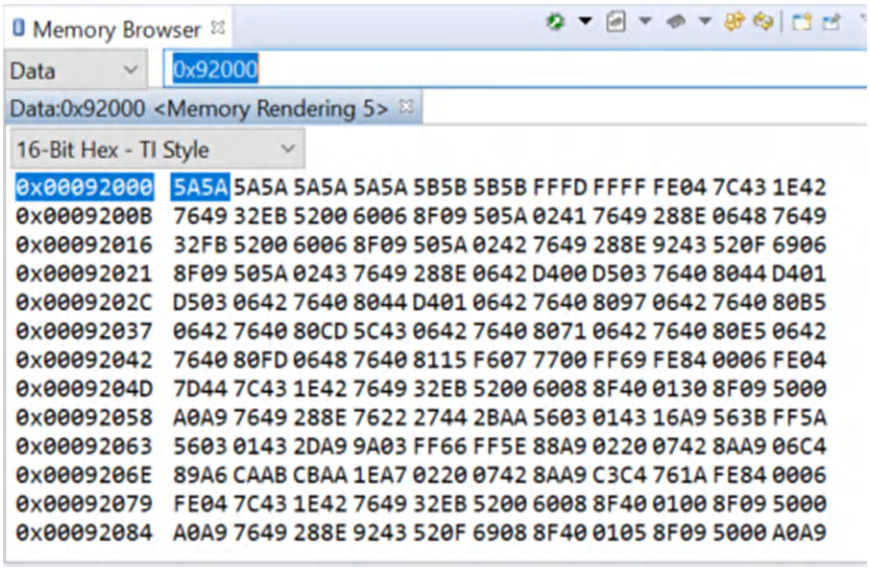

Figure 6-7. CCS Memory Browser View to Verify Successful Programming of Application on Bank 1

The kernel will also update the KEY and revision number in BANK1 sector 2. Now the static image, in programmed BANK0 and application image, is programmed in BANK1.

6. At this point, the user can reset the board to see LED2 start blinking.
7. Next load BANK1 static image (BANK1_LDFU of flashapi_ex2_sci_kernel project).

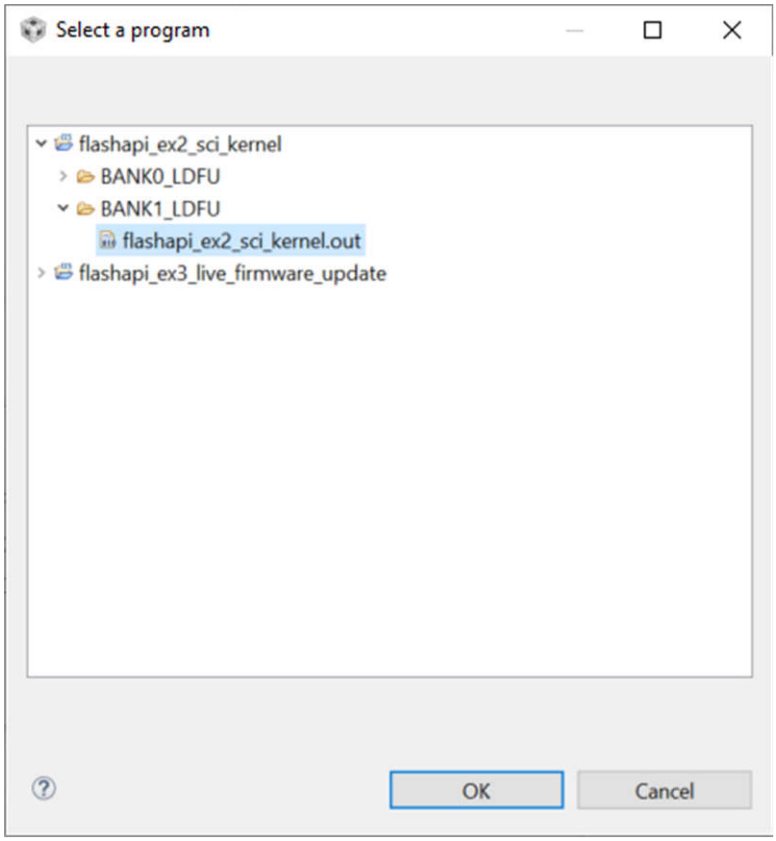

Figure 6-8. Selecting Kernel to Load to Flash Bank 1

8. Verify the Kernel content of BANK1 by opening the Memory Browser window in CCS, and entering address 0x90000.

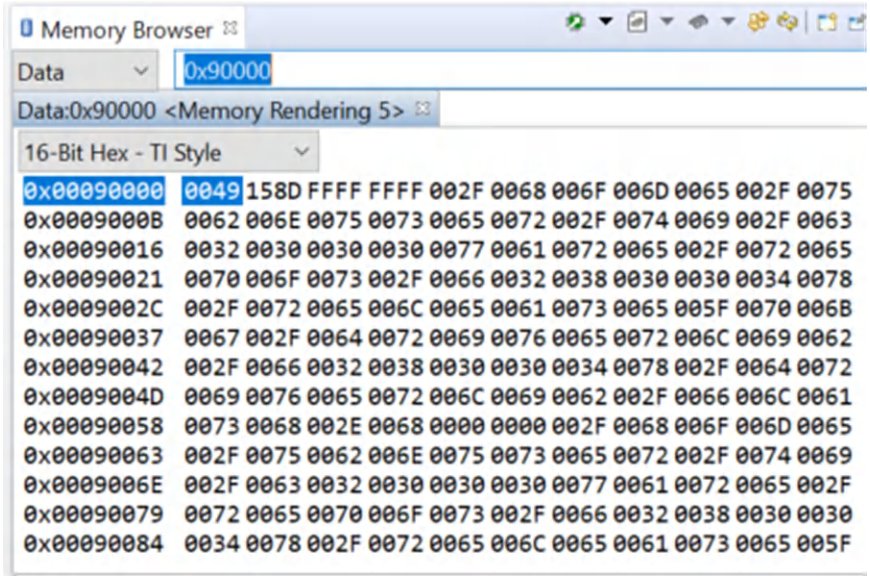

Figure 6-9. CCS Memory Browser View to Verify Successful Kernel Programming of Bank 1

After the static contents are loaded on to BANK1, execution stops at main(). This occurs for the kernel on Bank1 because bank selection logic resides only on Bank0, not on Bank1. Thus, for Bank1, execution flow follows the conventional flow of codestart leading up to main().

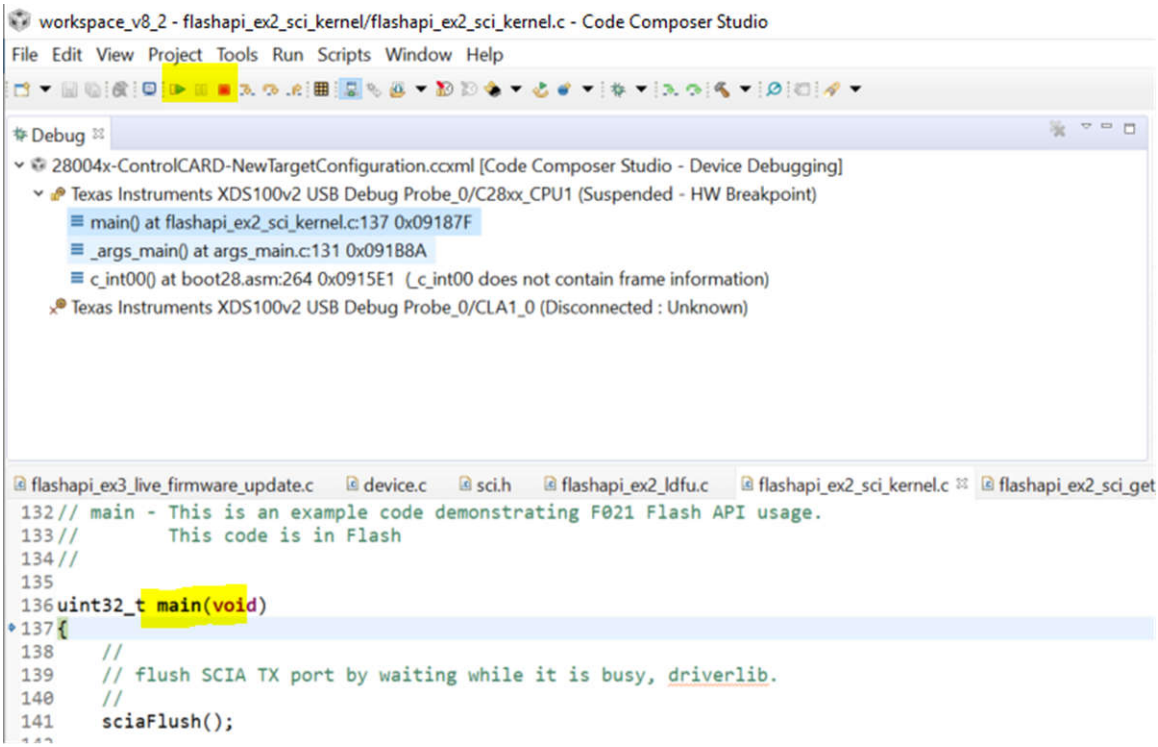

Figure 6-10. CCS Window View After Programming Bank 1 Flash Kernel

9. Press Run in CCS, the bank selection logic will run and execute application from BANK1. Then, execute the below command from command line in PC.

   serial_flash_programmer_appln.exe -d f28004x
   -k f28004x_fw_upgrade_example\flashapi_ex2_sci_kernel.txt -a
   flashapi_ex3_live_firmware_updateBANK0FLASH.txt -b 9600 -p COMx where x = COM
   port corresponding to the JTAG connection between the PC and the target board.
   flashapi_ex3_live_firmware_updateBANK0FLASH.txt is generated by building the CCS project
   flashapi_ex3_live_firmware_update in build configuration BANK0_FLASH.

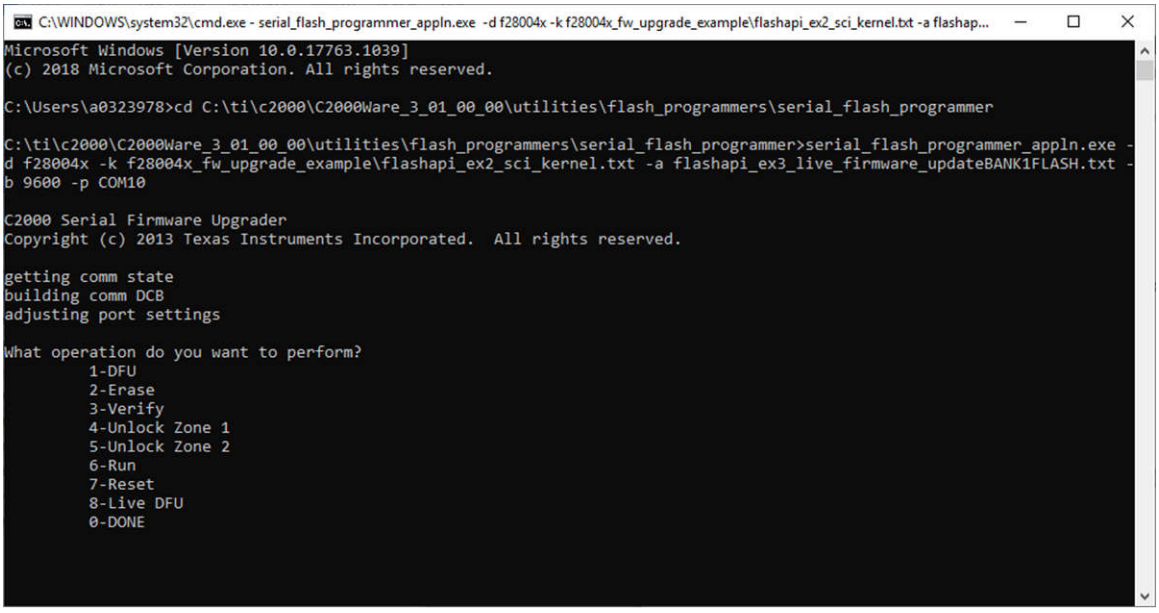

Figure 6-11. LFU Serial Command Invoked From Windows Command Prompt

10. Control will pass to flash kernel from the application and, once LDFU (8) command is selected, the kernel will receive and program the application in BANK0.
11. After transfer is complete, enter 0 to indicate end of command operations.

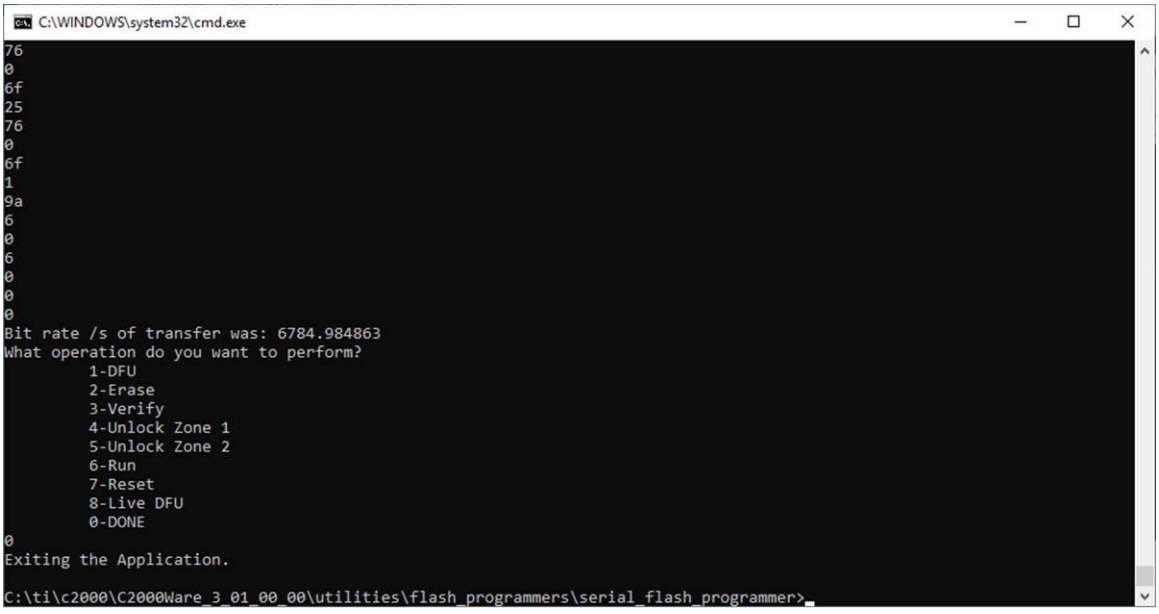

Figure 6-12. Successful Completion of LFU Command to Program Flash Bank 0

The kernel will also update the KEY and revision number in BANK0 sector 2. Now the static image in programmed BANK1 and application image is programmed in BANK0.

12. At this point, as before, the user can reset the board, to see LED1 start blinking.

6.3 Live Firmware Update of Application

After programming the static contents, disconnect the debugger and set the boot mode switches to flash boot mode. When the device boots up, it will jump to Flash. The default flash entry point is 0x80000, which is where the static code (Bank selection logic + SCI Flash Kernel) has been programmed in Bank0. The bank selection logic will execute and determine that valid images exist in both Banks 0 and 1 (based on KEY and revision number). Bank0 will be selected to run because, in Section 6.1, it was programmed later so it will be deemed the newest version based on the REV field. So the application firmware in Bank0 will be executed.

The application blinks LED1 at every 1 second. At the same time the application also monitors the serial port to check if it is getting any image for live firmware update.

To perform live firmware updates, build the updated application for BANK1 configuration and execute the command shown in Figure 6-13 from the host PC in the command line. Enter '8' for LDFU when the menu is listed.
serial_flash_programmer_appln.exe -d f28004x -k f28004x_fw_upgrade_example\flashapi_ex2_sci_kernel.txt -a flashapi_ex3_live_firmware_updateBANK1FLASH.txt -b 9600 -p COMx.

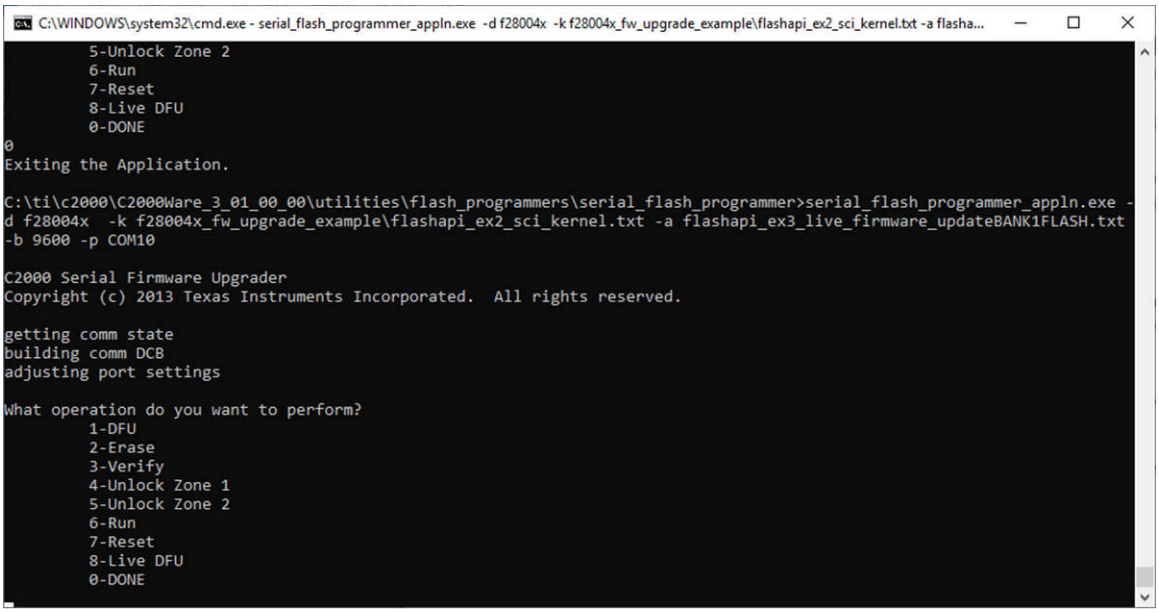

Figure 6-13. LFU Serial Command Invoked From Windows Command Prompt

If it receives an image, control will jump to the flash kernel in Bank0 and update the firmware image on Bank1. After transfer is complete, enter 0 to indicate end of command operations. When the new image is being downloaded to Bank1, the application continues to run on Bank0 (LED1 continues to blink at the usual 1s rate). This is because LFU processing occurs in a background loop, not the SCI Receive interrupt. This allows other interrupts, such as the CPU Timer interrupt that toggles the LEDs, to be serviced.

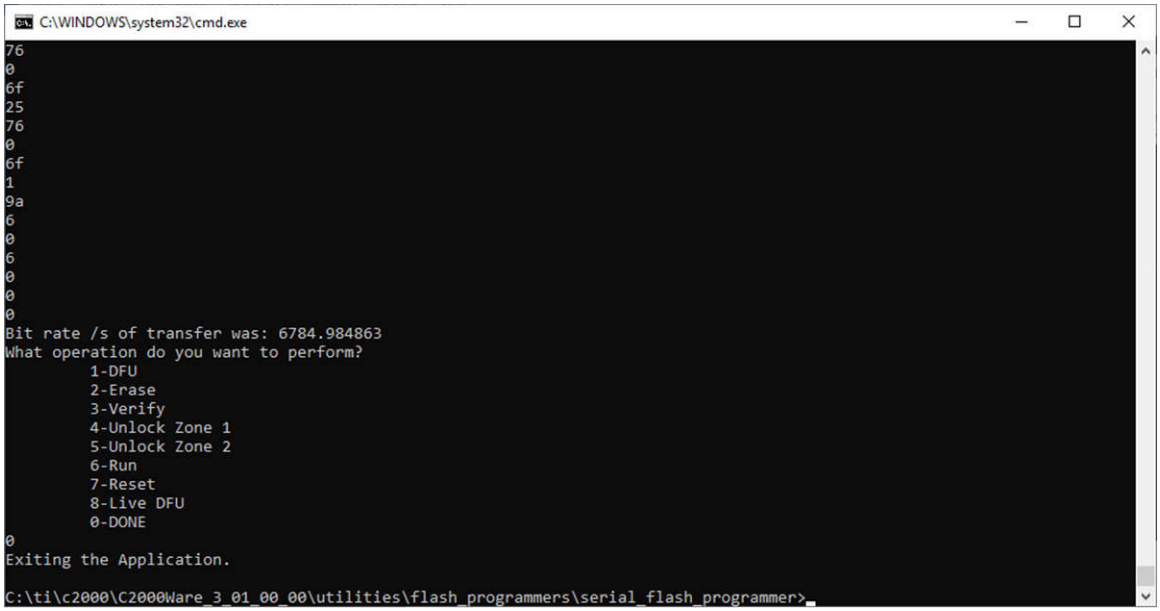

Figure 6-14. Successful completion of LFU Command to Program Flash Bank

After the Watchdog timer resets, the device resets, and control is passed to the newest application image in Bank1, and LED2 will start blinking.

Note

The manual device reset is no longer required.

This process can be repeated to update the image in alternate banks. Figure 6-15 illustrates the flow described above.

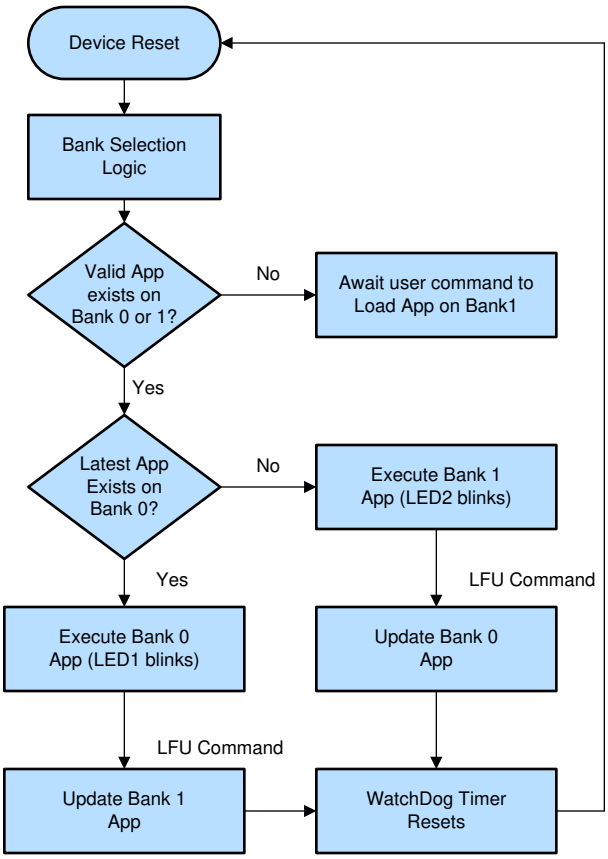

Figure 6-15. LFU Code Flow Diagram

6.4 Limitations and Troubleshooting

- One point for the user to note is that since the Bank Selection Logic resides in Bank0, if Flash corruption occurs when the Application is being updated on Bank0, it is possible that the Static contents of Bank0, although located at different sectors, end up corrupted as well. This would include the Bank selection logic + SCI Flash Kernel + Flash APIs (if running from Flash). The user would then have to repeat the steps involved in programming static code to get the system operational again.
- While programming the Flash kernel in Section 6.2, the erase settings for Flash should be set to “Necessary Sectors only”, otherwise, while Programming Kernel on BANK1, the Application on BANK1 will be erased.

7 Extended Implementations

7.1 Live Firmware Update with Reset on F28P65x MCUs

Extending upon the LFU principles discussed earlier, a dual-core LFU with reset example for F28P65x devices is available for reference in C2000Ware v26 or later:

C2000Ware_XX_XX_XX_XX\driverlib\f28p65x\examples\c28x_dual\flash_kernel\can_flash_lfu_sbl_multi_f28p65x

The main difference in the LFU architecture is that the F28P65x implementation offloads LFU operations to CPU2. This means that CPU1's only task is to execute the main application while CPU2 performs live firmware updates *for both cores* over MCAN with an external host. Please note that CPU2 is not running any other applications besides LFU in this example. Figure 7-1 illustrates the general LFU flow for the F28P65x implementation.

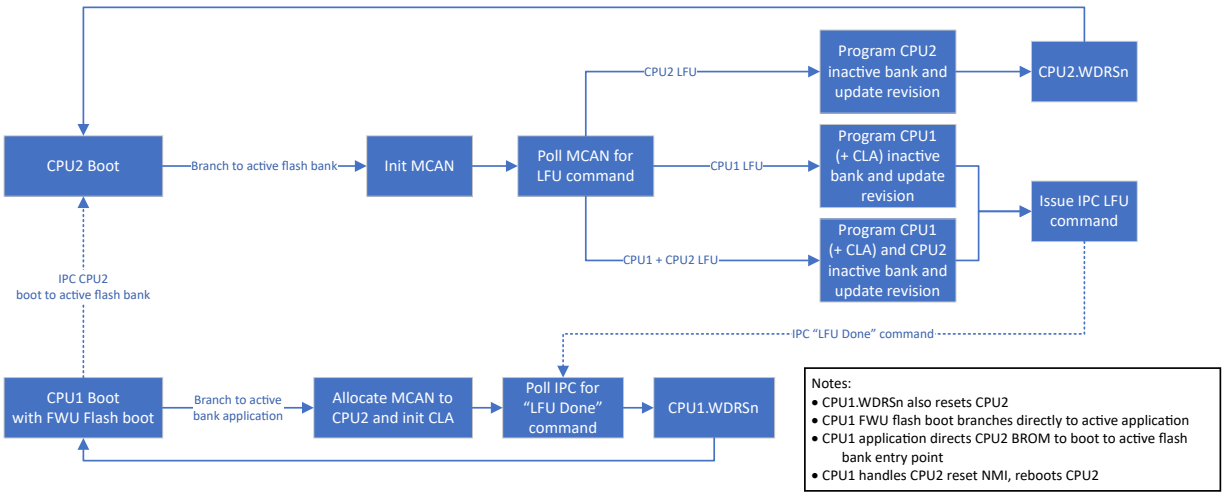

Figure 7-1. F28P65x LFU Flow

7.1.1 F28P65x LFU Hardware Requirements

The hardware components needed to run the LFU example is a F28P65x device connected to a CAN transceiver and a PEAK PCAN-USB Pro FD Analyzer. For controlCards, a custom-designed CAN transceiver board needs to be used, as well as the HSEC-180-pin ControlCard Docking Station. The custom-designed transceiver board is connected to the ControlCard using four connections: GND, 3V3, MCANTX and MCANRX. The LaunchPad™ devices contain an on-board CAN transceiver. The PEAK PCAN-USB Pro FD Analyzer is connected to the LaunchPad through the ground, CAN-Lo and CAN-Hi connections. The onboard CAN Routing switch needs to be set low for the transceiver to communicate using the GPIOs. After downloading the preliminary application images for CPU1 and CPU2 (see Section 7.1.5.1), it is important to setup the device correctly to be able to communicate with the host PC running the host programmer. Connect the MCAN TX and RX pins using a transceiver to the CAN port on the host side.

7.1.2 Flash Organization

In this example, CPU1 (+ CLA) and CPU2 are each allocated two flash banks for use:

- CPU1 + CLA (Application): Flash Banks 2 and 3
- CPU2 (LFU): Flash Banks 0 and 1

There is an "Active" and "Inactive" flash bank per core, which specify the latest firmware to execute on reset. When an LFU occurs, CPU2 programs the inactive flash bank for CPU1 and/or CPU2, updating the flash bank's

metadata to indicate the latest firmware version for FWU boot mode (provided by the Boot ROM) to decode on reset (see Section 7.1.3 for more information).

CPU1 uses the FWU boot mode to branch to the active flash bank. Then, the CPU1 application allocates the MCAN peripheral, CPU1 inactive flash, and CPU2 active/inactive flash banks to CPU2 before bringing CPU2 out of reset, booting CPU2 with FWU boot mode. After these steps are completed, CPU2 can perform an LFU as requested by the host over MCAN.

Note

CPU2 is also allocated CPU1's inactive flash bank to allow CPU2 to perform LFU for CPU1.

The LFU architecture is shown in Figure 7-2.

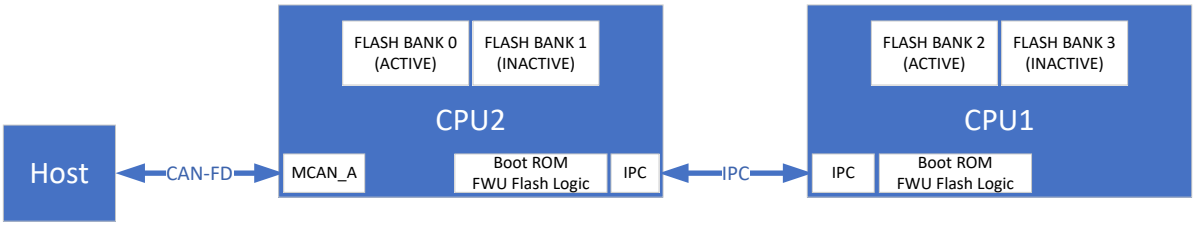

Figure 7-2. F28P65x LFU Architecture

7.1.3 FWU Boot Mode

CPU1 and CPU2 are both booted with the Firmware Update (FWU) boot mode. This requires that a firmware validity key (0x5A5A5A5A) and a 32-bit version number be placed at offsets 0xA and 0xC, respectively, from the application's entry point. The flash entry points for CPU1 and CPU2 will be organized as detailed in Table 7-1.

FWU boot supports additional boot options with distinct entry points. This example leverages the various FWU boot options to ensure that CPU1 and CPU2 application versions can be distinguished by the Boot ROM. Thus, CPU1 won't mistakenly branch to a CPU2 flash bank and CPU2 won't branch to a CPU1 flash bank.

For more details on FWU boot mode, please refer to the F28P65x Technical Reference Manual's *Firmware Update (FWU) Flash Boot* section.

Table 7-1. FWU Application Image Format

Image Address Offset	Content
0x0	Application entry point (32-bit)
0xA	Key (32-bit), Valid Key = 0x5A5A5A5A
0xC	Firmware version number (32-bit)

7.1.4 LED Example Application

This example (can_flash_lfu_sbl_multi_f28p65x) is designed to demonstrate LFU for CPU1 (+ CLA) and CPU2 with the LED blinking periodically. This is achieved using the LFU over MCAN Flash Programmer host utility (described in Section 7.1.5.3).

In this example, CPU2 executes the LFU routine, which includes MCAN_A communication, flash operations, and IPC command issuance. After initializing MCAN_A (with extended ID filters), CPU2 polls and processes commands received over MCAN.

Meanwhile, CPU1 (+ CLA) executes the main application. CPU1 initializes and enables three interrupts: IPC interrupt, CPU2 Watchdog Reset NMI interrupt, and CPU Timer 0 interrupt.

- The IPC ISR responds to IPC commands sent by CPU2; this includes CPU1 reset requests, CPU1 inactive flash bank number requests, and CPU1 active version requests.
- The NMI ISR handles CPU2.NMIWDRSn reset events and directs CPU to boot in FWU mode (see Section 7.1.3 for more details).
- The CPU Timer 0 interrupt occurs at a set period (either ½ or 1 second depending on the firmware version); the interrupt for CPU Timer 0 triggers the CLA to toggle LED2 at a frequency dependent on the active firmware version.

CPU1 also handles resource allocation between the cores:

- Flash Banks 0 + 1 and MCAN_A are allocated to CPU2
- Flash Banks 2 + 3 are allocated to CPU1

Lastly, CPU1 boots CPU2 in FWU mode. CPU1 starts CPU Timer 0 for the CLA to blink LED2 while CPU1 also blinks LED1 in the main background loop (at the same frequency as the CLA).

There are two firmware versions for CPU1 and CPU2 in this example: "A" and "B".

In firmware version A, the LEDs are toggled every second by CPU1 and the CLA. The build configurations "FLASH_A" and "FLASH_A_LAUNCHXL" are associated with version A. “FW_A” is a pre-defined symbol in these build configurations. When this symbol is defined, the project allocates the application to Flash Bank 0 or 2 (for CPU2 or CPU1 respectively) and sets up the LED toggle period accordingly. If the F28P65x LaunchPad is used, the *LaunchXL* build configurations configure GPIOs associated with LEDs for the F28P65x LaunchPad, if used.

In firmware version B, the LEDs are toggled every second by CPU1 and the CLA. The build configurations "FLASH_B" and "FLASH_B_LAUNCHXL" are associated with version B. “FW_B” is a pre-defined symbol in these build configurations. When this symbol is defined, the project allocates the application to Flash Bank 1 or 3 (for CPU2 or CPU1 respectively) and sets up the LED toggle period accordingly. If the F28P65x LaunchPad is used, the *LaunchXL* build configurations configure GPIOs associated with LEDs for the LaunchPad, if used.

The application images generated by building the above build configurations are the ones that are used to illustrate LFU in this document. Note that there are no other differences between the two application versions other than the changes described above. Hence, this is a relatively simple example for LFU illustration.

7.1.4.1 LFU Command Processing

When a new command is sent to the device from the host utility, CPU2 receives and parses the command to determine the LFU mode. The LFU commands available are the following:

CPU2 LFU Command

If the command matches the CPU2 LFU (COMMAND_LFU_CPU2) command, then the code branches to the CPU2 LFU function (LFUCPU2Flow() in sbl_command_flow.c) in the active CPU2 flash bank. Within this function, execution passes to the downloadBlock() function after erasing the inactive CPU2 flash bank. The firmware (in the appropriate hex format) is programmed and verified in the inactive flash bank. Then, CPU2 configures a Watchdog reset for CPU2. After the CPU2 resets, CPU1 handles the CPU2.NMIWDRSn NMI and boots CPU2 in FWU mode to execute the active firmware.

CPU1 LFU Command

If the command matches the CPU1 LFU (COMMAND_LFU_CPU1) command, then the code branches to the CPU1 LFU function (LFUCPU1Flow() in sbl_command_flow.c) in the active CPU2 flash bank. Within this function, execution passes to the downloadBlock() function after erasing the inactive CPU1 flash bank. The firmware (in the appropriate hex format) is programmed and verified in the inactive CPU1 flash bank. Then, CPU2 issues an IPC command to configure a CPU1 Watchdog reset. Note that CPU2 also resets with a CPU1 Watchdog reset. Thus, after CPU1 boots in FWU mode to the active CPU1 flash bank, the CPU1 application must boot CPU2 in FWU mode to execute the active firmware.

CPU1 + CPU2 LFU Command

If the command matches the CPU1 + CPU2 LFU (COMMAND_LFU_CPU1_CPU2) command, then the code branches to the CPU1 + CPU2 LFU function (LFUCPU1CPU2Flow() in sbl_command_flow.c) in the active CPU2 flash bank. Within this function, execution passes to the downloadBlock() function (called twice to program CPU1 and CPU2 flash) after erasing the inactive CPU1 and CPU2 flash banks. The firmware (in the appropriate hex format) is programmed and verified in the inactive CPU1 and CPU2 flash banks. Then, CPU2 issues an IPC command to configure a CPU1 Watchdog reset. Note that CPU2 also resets with a CPU1 Watchdog reset. Thus, after CPU1 boots in FWU mode to the active CPU1 flash bank, the CPU1 application must boot CPU2 in FWU mode to execute the active firmware.

7.1.5 Running the Example

7.1.5.1 Loading the Example

Both the host and client expect to read/receive images in the TI ASCII hex format, which can be generated by adding the following post-build step to your project:

```
"${CG_TOOL_HEX}" "${BuildArtifactFileName}" -boot -sci8 -a -o "${BuildArtifactFileBaseName}.txt"
```

To perform the flash A/B swap, two linker file versions are employed to toggle between using CCS build configurations, each placing the application in a different flash bank. For this example, the following scheme is used:

Table 7-2. Flash Bank Organization

Application	Flash Bank	Entry Point	FWU Boot Option
CPU1 FW Version A	2	0x000C 0000	0x0B
CPU1 FW Version B	3	0x000E 0000	0x0B
CPU2 FW Version A	0	0x0009 0000	0x4B
CPU2 FW Version B	1	0x000B 0000	0x4B

Note

CPU1 and CPU2 must use different FWU boot options to avoid conflicts between booting the CPU1 and CPU2 firmware after a device reset.

After generating the application images, the initial versions need to be loaded onto the device through the following procedure:

1. Configure CPU1 for FWU boot mode 0 (BOOTDEF = 0x0B). Users can do this in emulation boot by configuring the emulation boot registers at addresses 0xD00-0xD01 and 0xD04 in the memory browser.
   a. Users can also program the boot registers in the DCSM OTP to configure device for standalone FWU boot. Please refer to *Getting Started with C2000 Bootloading on C2000 Microcontrollers* for more details.

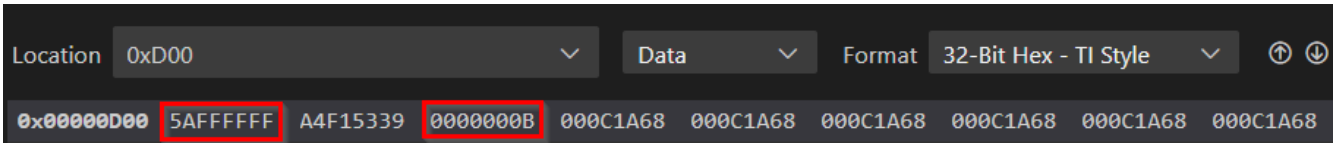

Figure 7-3. Emulating FWU Boot using CCS Memory Browser

2. Configure CCS Flash Settings for CPU1 to load the image by allocating the flash banks to the correct CPU. In this case, CPU2 is allocated Banks 0 and 1.

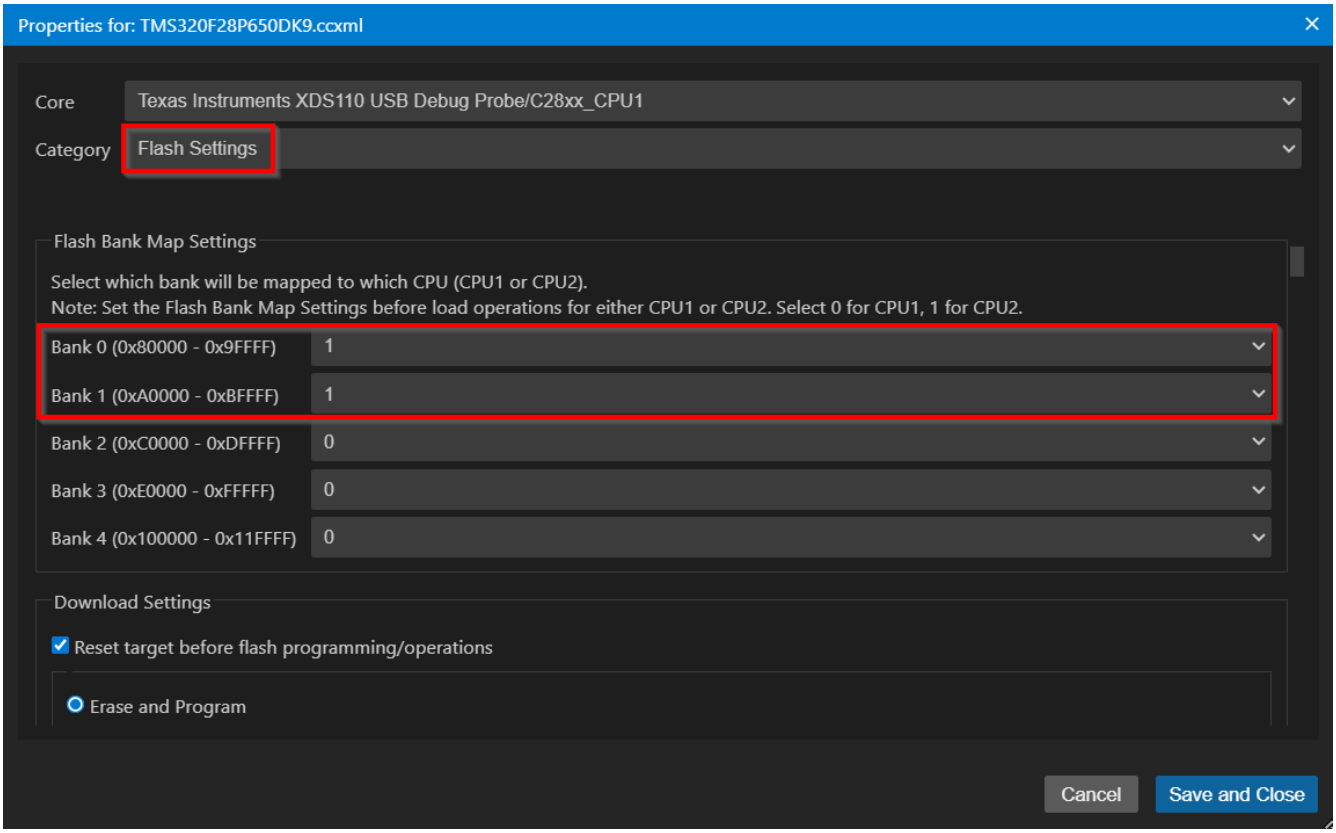

Figure 7-4. Flash Bank Mapping in CCS CPU1 Flash Settings

3. Configure the CCS Flash Settings for CPU1 to not erase CPU2's Flash Banks by using the "Selected Banks Only" setting.

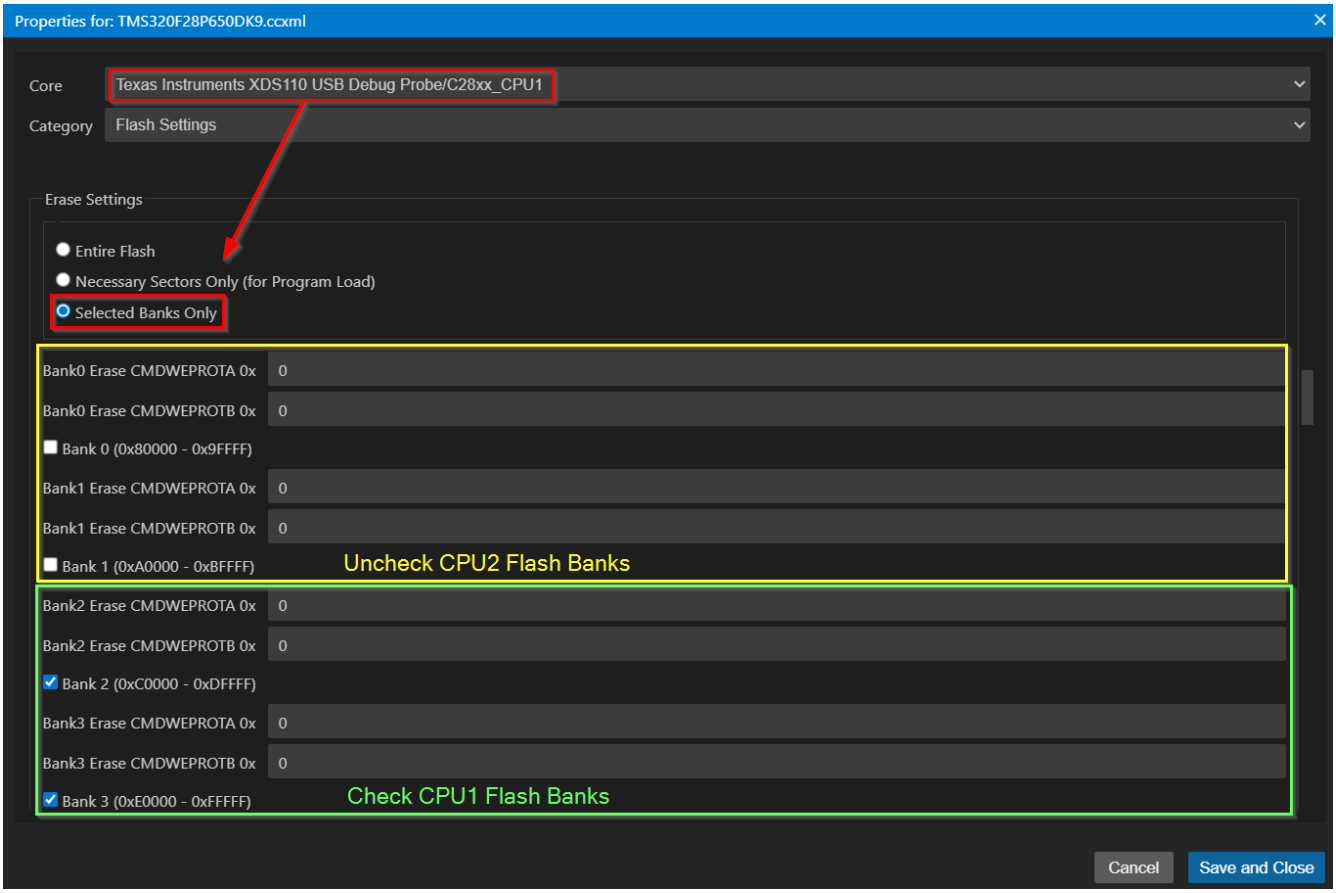

Figure 7-5. CPU1 Flash Bank Erase Settings

4. Load the CPU1 application image.
5. Run CPU1.
   a. CPU1 allocates GS4RAM and Flash Banks 0 + 1 to CPU2. Therefore, CPU2 variables can be properly initialized on CPU2 application load.
6. Configure the CCS Flash Settings for CPU2 to not erase CPU1's Flash Banks.

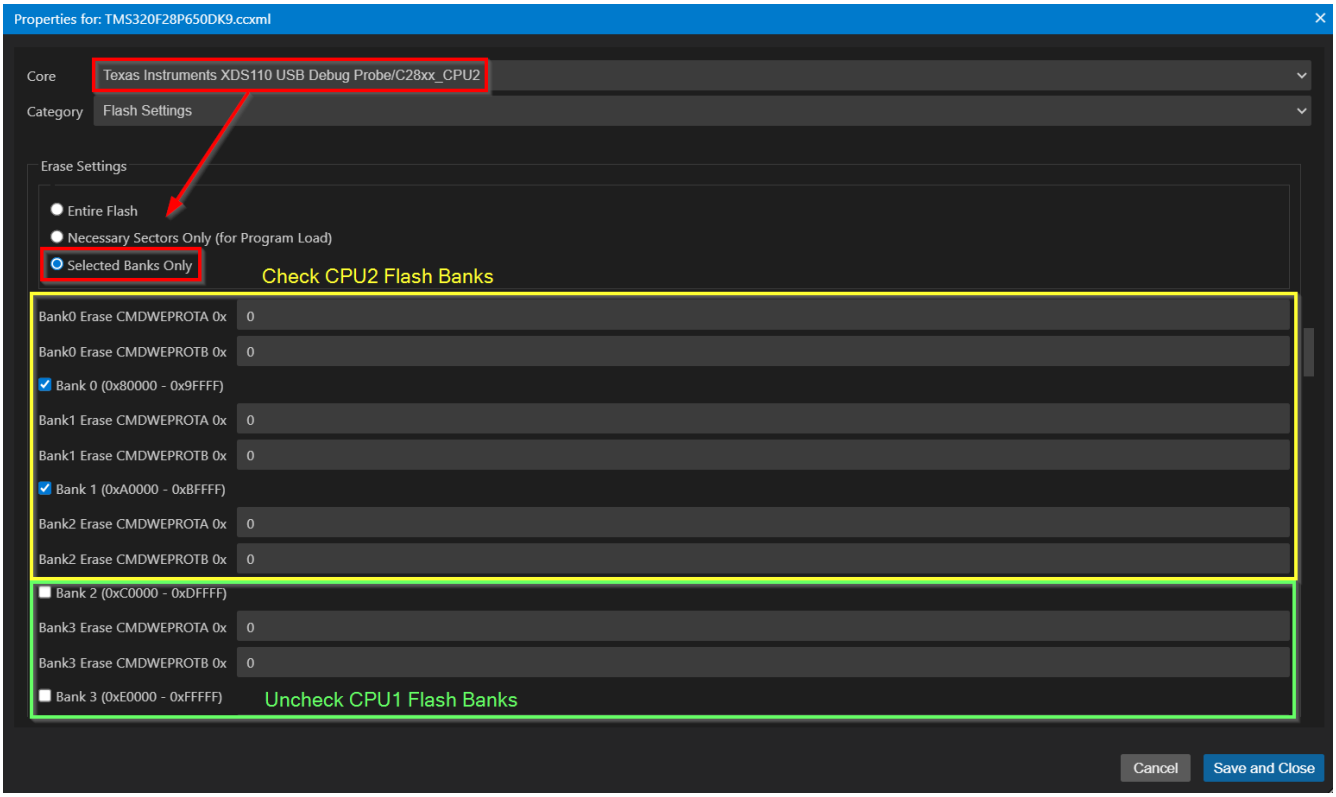

Figure 7-6. CPU2 Flash Bank Erase Settings

7. Load the CPU2 application image.
8. Run CPU2.

7.1.5.2 Combing CPU1 and CPU2 Firmware Images

The F28P65x LFU implementation supports sending the CPU1 and CPU2 firmware together in a combined image. To create this image, simply concatenate the CPU2 firmware image to the CPU1 firmware image from the TI ASCII hex file output.

For example, see the following snippet from a combined image file:

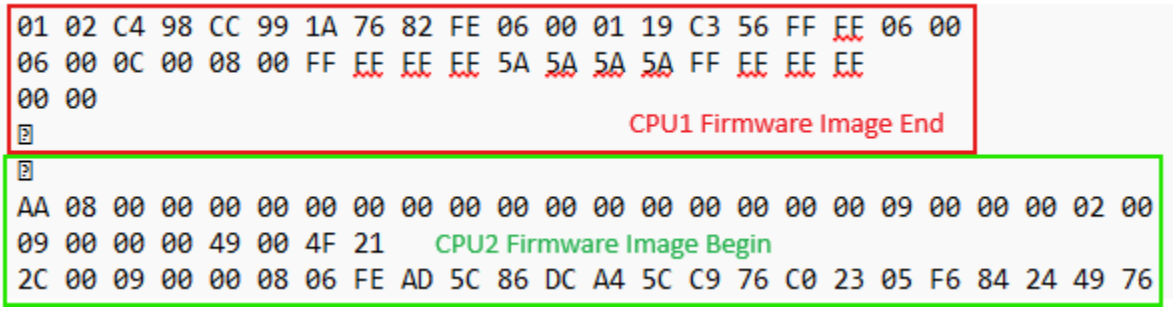

Figure 7-7. Combing CPU1 and CPU2 Firmware Images

7.1.5.3 LFU over MCAN Host Programmer

The MCAN LFU host programmer (can_lfu_flash_programmer.exe) provided in C2000Ware takes the CPU1 firmware, CPU2 firmware, or a combined CPU1 + CPU2 firmware image as parameters (please see Section 7.1.5.3.2 for more details). The host programmer communicates with CPU2 over MCAN with the PEAK PCAN-USB Pro FD Analyzer to program the inactive flash banks of CPU1 and/or CPU2. The flash programmer project is built and run on Visual Studio 2019. The host programmer uses the PCAN-Basic API from PEAK. The PCAN-Basic API can be used to send and receive CAN-FD frames on the CAN analyzer.

The LFU host programmer and source code can be found in C2000Ware v26 or later:

- *C2000Ware_XX_XX_XX_XX\utilities\flash_programmers\can_lfu_flash_programmer*

The clock to the MCAN module is switched to the external clock source by CPU2. The external clock is 25MHz in the LaunchPad and the ControlCard. CPU2 also configures the nominal bit rate to be 1Mbps, and the data bit rate to be 2Mbps. The host programmer configures the PEAK CAN analyzer to have the same clock, nominal and data bit rate values.

7.1.5.3.1 Compiling the Host Programmer

The host programmer (can_lfu_flash_programmer.exe) can be compiled/recompiled using CMake in the command line or using Visual Studio.

Command Line

To do so from the command line:

1. Download CMake.
2. Open a terminal and change to the directory containing the CMakeLists.txt file.
3. Run "cmake -S . -B build" to generate the build configuration files.
4. Run "cmake --build build" to compile the executible.

Visual Studio

To do so from Visual Studio:

1. Navigate to File > Open > CMake
2. Navigate to and open the CMakeLists.txt file. Visual Studio will then automatically configure the project for you.
3. Build the executable from the Visual Studio *Build* menu.

7.1.5.3.2 Using the Host Programmer

To perform an LFU, the host programmer must be used to send the application image over MCAN to the MCU, which then programs the received data into the inactive flash bank.

To use the host programmer, compile and run the generated can_lfu_host_programmer.exe. The following command line arguments are supported:

Table 7-3. LFU Host Programmer Arguments

Argument	Description	Requirement
-d	The target device, currently the only valid option is "f28p65x"	Required
-a1	The CPU1 application image to be programmed to the CPU1 inactive flash bank	Required for CPU1-only LFU (optional otherwise)
-a2	The CPU2 application image to be programmed to the CPU2 inactive flash bank	Required for CPU2-only LFU (optional otherwise)
-a3	The combined CPU1 and CPU2 image to be programmed to the CPU1 and CPU2 inactive flash banks	Required for CPU1 + CPU2 Multi-LFU (optional otherwise)

After starting the host programmer, the user is presented with several command options (described further in detail in Section 7.1.4.1):

- **CPU1 LFU**: Perform a CPU1 LFU by downloading the application image provided by the -a1 command line argument, then reset CPU1 to activate the new firmware.
  - Note: Provided CPU1 reset is also a system reset, CPU2 also resets.
- **CPU2 LFU**: Perform a CPU2 LFU by downloading the application image provided by the -a2 command line argument, then reset CPU2 to activate the new firmware.
  - Note: A CPU2 reset issues an NMI to CPU1. CPU1 must handle the NMI and bring CPU2 out of reset to activate the new CPU2 firmware, otherwise the entire system resets.

- **CPU1+2 Multi-LFU**: Perform a CPU1 + CPU LFU by downloading the combined application image provided by the -a3 command line argument, then reset the system to activate the new firmware.
- **Done**: Exit the host programmer application.

An example usage is shown in Figure 7-8:

```
PS C:\ti\c2000\C2000Ware_26_00_00_00\utilities\flash_programmers\can_lfu_flash_programmer> .\can_lfu_flash_programmer.exe -d f28p65x -a3 .\cpu1_cpu
2_fw_b.txt
F28x CAN LFU Firmware Programmer
Version: 1.0.0; Packet protocol: 1.0.0
Copyright (c) 2024-2025 Texas Instruments Incorporated.

Initializing CAN USB Channel1 with nominal Bit rate set to 1 MBit/s and data bit rate set to 2 MBit/s


Please select from the available operations below:
Programming operations:
  1-CPU1 LFU
  2-CPU2 LFU
  3-CPU1 + CPU2 Multi-LFU
Utility operations:
  0-Done
3
Performing CPU1 + CPU2 Multi-LFU...
Sending block, start address = 0xe0000, size = 0x2...
Sending block, start address = 0xe22e8, size = 0x1c...
Sending block, start address = 0xe1dd8, size = 0x41e...
Sending block, start address = 0xe21f8, size = 0xeb...
Sending block, start address = 0xe1c28, size = 0x1aa...
Sending block, start address = 0xe0008, size = 0x6...
Sending block, start address = 0xe0010, size = 0x1c12...
Sending block, start address = 0xb0000, size = 0x2...
Sending block, start address = 0xb0800, size = 0x29...
Sending block, start address = 0xa24b8, size = 0x3e...
Sending block, start address = 0xa06a0, size = 0x1ec...
Sending block, start address = 0xa0000, size = 0x699...
Sending block, start address = 0xb0008, size = 0x6...
Sending block, start address = 0xa0890, size = 0x1c21...

Please select from the available operations below:
Programming operations:
  1-CPU1 LFU
  2-CPU2 LFU
  3-CPU1 + CPU2 Multi-LFU
Utility operations:
  0-Done
0
Exiting the application.
PS C:\ti\c2000\C2000Ware_26_00_00_00\utilities\flash_programmers\can_lfu_flash_programmer>
```

Figure 7-8. F28P65x LFU Example Usage

7.1.6 Restrictions

The following restrictions must be obeyed for this LFU scheme. Please note the below are already implemented for the example provided in C2000Ware.

1. If the user needs to program the DCSM as a part of your application image, Flash Bank 0 must be a part of CPU2's active/inactive bank scheme and not CPU1's. This maintains that CPU2 always has ownership of, and thus programming permissions for, Flash Bank 0 and Flash User OTP Bank 0.
2. The CPU2 application loader's codestart must use a different FWU boot option than CPU1 to avoid conflicts between version numbers when booting the device. For example, if CPU1 is using FWU boot option 0, then CPU2 must use one of FWU alternate boot options 1-3.
3. All flash memory sections must be 128-bit aligned using ALIGN(8) in the linker command file since CPU2 issues 128-bit programming commands.

8 Revision History

Changes from Revision A (August 2021) to Revision B (March 2026)	Page
• Updated the numbering format for tables, figures and cross-references throughout the document	3
• Added *Extended Implementations* section with details on the F28P65x dual-core MCAN LFU implementation	20

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, 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 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, TI’s General Quality Guidelines, 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. Unless TI explicitly designates a product as custom or customer-specified, TI products are standard, catalog, general purpose devices.

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

Last updated 10/2025