*Application Note*

# *Live Firmware Update via Flash Bank Swap*

*Ryan Kim* *Korea Sales – Major Account*

**ABSTRACT**

This application note describes the live firmware update algorithm using bank swap and a CSC (Customer Secure Code), and explains how to implement it on MSPM0 devices that support dual-bank memory. In this example, the LP-MSPM0G3519 EVM is used, and the new firmware image is transmitted via UART communication.

## Table of Contents

## Trademarks

All trademarks are the property of their respective owners.

## 1 Introduction

After a device is released to the field, developers may identify unexpected software bugs or wish to add new functionalities. In such cases, firmware updates are required even while the MCU remains operational. However, traditional update methods using a bootloader or SWD (Serial Wire Debug) are not suitable for this scenario, as they typically require the MCU to stop running.

To overcome this limitation, a Live Firmware Update method can be applied, allowing firmware to be updated in real time without halting system operation. This approach leverages the dual-bank flash memory architecture, which enables one bank to execute the existing firmware while the other bank is programmed with the new image. After the update and verification processes are completed, a bank swap is performed to activate the new firmware.

As of November 2025, there are 17 MCU products that support dual-bank memory and can be utilized for live firmware updates via bank swap. In this implementation, the new firmware is downloaded into Bank 1 while the existing application continues to run in Bank 0. A CSC (Customer Secure Code) is used to verify the firmware version and determine whether a bank swap should be triggered. The new firmware image is transmitted from a PC to the MCU via UART communication, using a defined data frame structure that includes a CRC32 (JAMCRC) checksum to ensure reliable and error-free data transmission.

To perform the live firmware update efficiently, several software components and tools are provided:

- Provided by TI
  - CSC (Customer Secure Code) Image: Version checking, and bank swap operations.
  - Bank 0 / Bank 1 Application Images: Contain the active and updated firmware, respectively. Receive new firmware through UART. Verify firmware through CRC32 (JAMCRC)
  - Data Frame Generator (uart_frame_gui.exe): A PC-side utility that converts a raw .bin file into a framed data stream with CRC32 (JAMCRC) validation.
- User-provided
  - Tera Term (https://teratermproject.github.io/index-en.html): Used to transmit the framed firmware image from PC to the MCU (via XDS-110 Debugger)
  - LP-MSPM0G3519 (https://www.ti.com/tool/LP-MSPM0G3519): This evaluation board has both XDS-110 debugger and MSPM0G3519SPZR. XDS-110 also provide USB to UART bridge that will be used to send new FW from PC.

Please download CSC, Application images, Data frame generator and Slides from the link: https://e2e.ti.com/cfs-file/__key/communityserver-discussions-components-files/908/Live-Firmware-Update-via-Bank-Swap_5F00_Shared-files_5F00_260116.zip

For more information about CSC(Customer Secure Code) and multi bank feature in MSPM0, please refer to 'Flash Multi Bank Feature in MSPM0 Family' Application note.

By combining dual-bank architecture with these supporting tools, developers can safely and efficiently update firmware in deployed systems without interrupting device operation, achieving a reliable in-field firmware update solution.

# 2 Detailed Description

## 2.1 Overview

### *2.1.1 Live Firmware Update Flow*

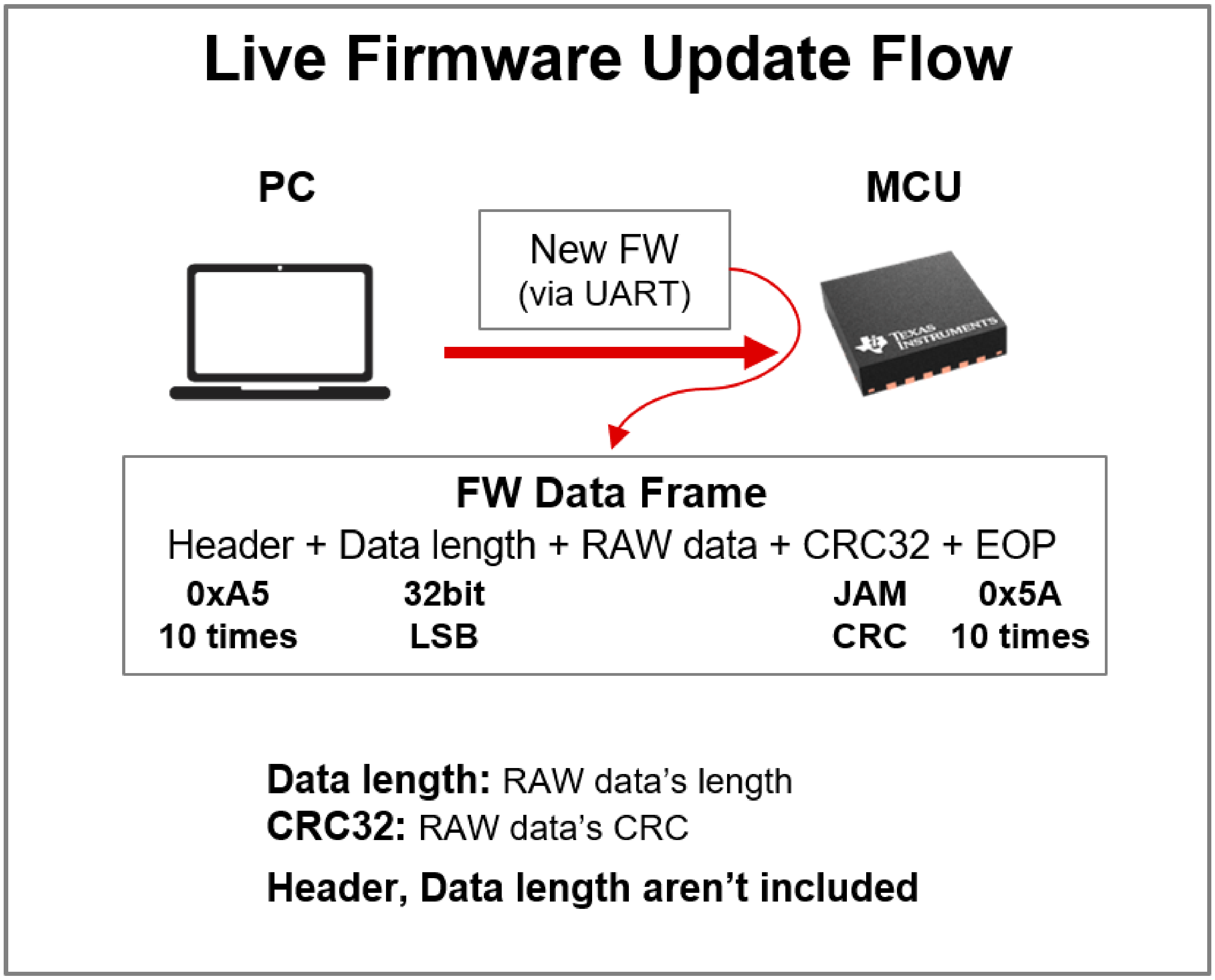

Live Firmware Update Flow
PC
MCU
New FW
(via UART)
FW Data Frame
Header + Data length + RAW data + CRC32 + EOP
0xA5
10 times
32bit
LSB
JAM
CRC
0x5A
10 times
Data length: RAW data's length
CRC32: RAW data's CRC
Header, Data length aren't included

**Figure 2-1. Live Firmware Update Flow**

The PC sends a firmware data frame to the MCU via UART. The firmware data frame consists of a Header (0xA5 × 10), Data Length (32-bit, LSB first), Raw Data, CRC32 (JAMCRC), and an End of Packet (EOP, 0x5A × 10). The Data Length field represents the size of the Raw Data, and the CRC32 value is calculated over the Raw Data.

### *2.1.2 Memory Organization*

**Table 2-1. Memory Organization**

| MEMORY REGION | SUBREGION | Address (ex. MSPM0G3519) | Remark |
|---|---|---|---|
| Bank 0<br>(Run) | CSC | 0x0000.0000 ~ 0x0000.1999 | Same as Bank 1 CSC, decide bank swap |
| | App | 0x0000.2000 ~ 0x0003.FFFF | Running App |

**Table 2-1. Memory Organization (continued)**

| MEMORY REGION | SUBREGION | Address (ex. MSPM0G3519) | Remark |
|---|---|---|---|
| Bank 1 (Inactive) | CSC | 0x0004.0000 ~ 0x0004.1999 | Same as Bank 0 CSC, decide bank swap |
| | App | 0x0004.2000 ~ 0x0007.FFFF | Old FW or new FW will be downloaded here |

The memory region is divided into Bank 0 and Bank 1, and each bank is further split into a CSC (Customer Secure Code) section and an application (App) section.

The CSC reads the firmware version stored at the beginning of each bank (Bank 0: 0x0000.2000, Bank 1: 0x0004.2000, both in uint32_t format), determines which bank contains the latest firmware, and decides whether a bank swap is required.

For example, if Bank 0 already contains the latest firmware, the CSC does not trigger a bank swap. However, if Bank 1 contains a newer firmware version, the CSC initiates a bank swap.

After completing its checks, the CSC transfers execution to the application in Bank 0.

## 2.2 Block Diagram

The following figure shows the system block diagram for the live firmware update system.

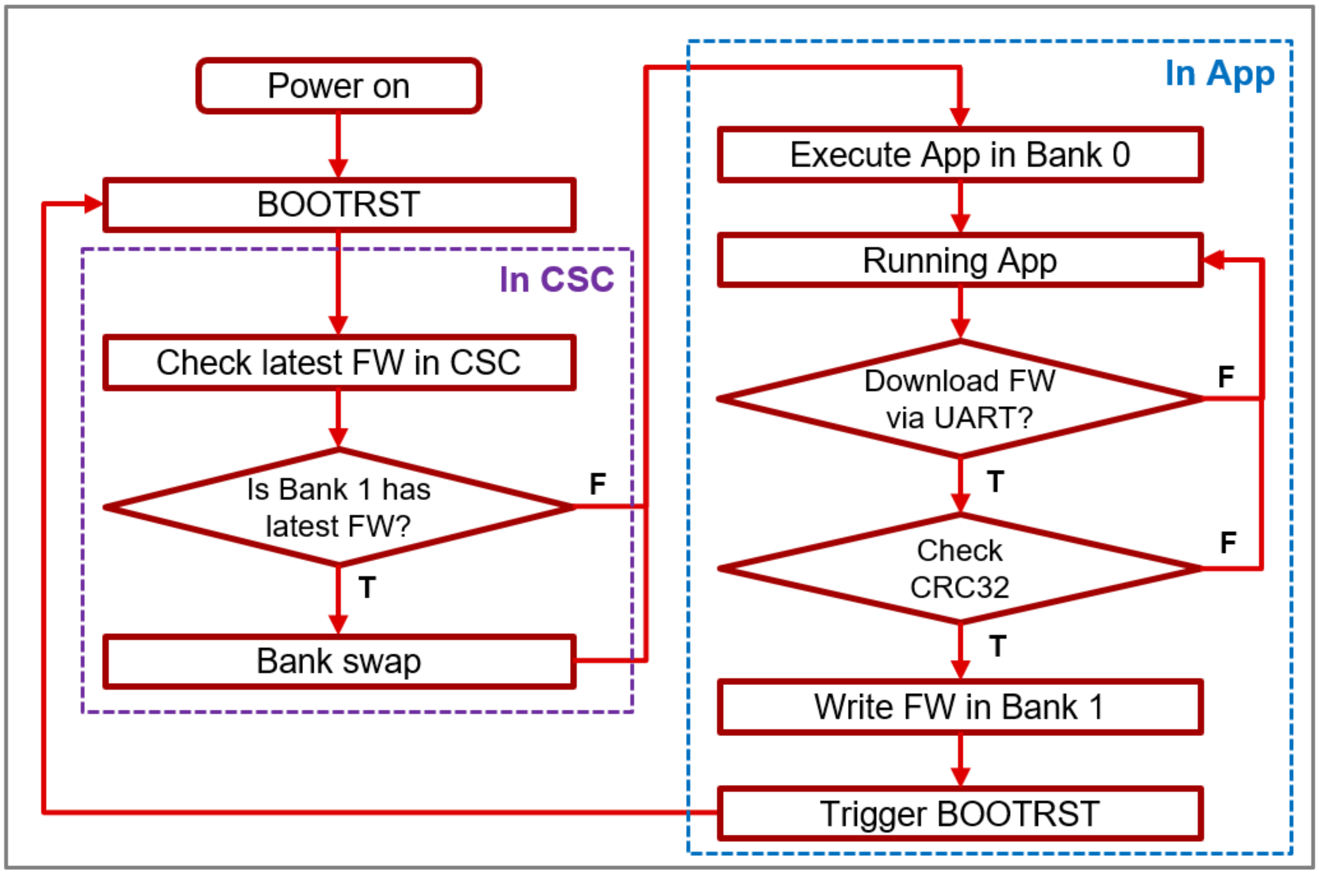


**Figure 2-2. Live Firmware Update Block Diagram**

The system consists of two execution domains: the CSC and the Application.

The CSC (Customer Secure Code) is responsible for verifying the firmware version and determining whether a bank swap should be performed.

The Application handles downloading the new firmware via UART, verifying data integrity using CRC32, and executing the user-defined functions included in the application image.

## 2.3 Code

### *2.3.1 CSC (Customer Secure Code, Bankswap_CSC_G3519_v2)*

#### 2.3.1.1 CSC - Main Function (Bankswap_CSC_G3519_v2.c)

```
#define BANK0_APP_START 0x00002000
#define BANK1_APP_START 0x00042000

uint32_t gVer_info_LB0, gVer_info_LB1;
bool gBankswap_conduct;

int main(void)
{
    SYSCFG_DL_init();

    /* Get version information */
    gVer_info_LB0 = *((uint32_t*)BANK0_APP_START);
    gVer_info_LB1 = *((uint32_t*)BANK1_APP_START);

    /* Decide bank swap */
    if (gVer_info_LB0 < gVer_info_LB1) {
        uint32_t* ptr_version_info = (uint32_t*)BANK1_APP_START + 1; // 0x00042004
        uint32_t* ptr_SP = (uint32_t*)BANK1_APP_START + 64; // 0x00042256

        gBankswap_conduct = true;

        /* Check App data format */
        for(int i=0; i<63; i++) {
            if(ptr_version_info[i] != 0xFFFFFFFF) {
                gBankswap_conduct = false;
            }
        }

        /* Check stack pointer, it depends on device */
        /* MSPM0G3519 RAM 64kB - 0x20210000 */
        if(*ptr_SP != 0x20210000)  {
            gBankswap_conduct = false;
        }
    }

    if (DL_SYSCTL_isINITDONEIssued())
    {
        start_app((uint32_t *)VECTOR_ADDRESS_BANK0);
    }
    else  //Init and SWAP
    {
        if (gBankswap_conduct){
            DL_SYSCTL_executeFromUpperFlashBank(); // set flash bank swap bit
            delay_cycles(160);
            DL_SYSCTL_issueINITDONE(); // Issue INITDOEN to trigger System Reset -> swap to bank1
        }else{
            DL_SYSCTL_executeFromLowerFlashBank(); // still execute program from bank0
            delay_cycles(160);
            DL_SYSCTL_issueINITDONE(); // Issue INITDOEN to trigger System Reset -> jump to bank0
app program
        }
    }
}
```

The CSC (Customer Secure Code) checks the firmware versions in Bank 0 and Bank 1 to determine whether a bank swap is required. Version information is stored at the beginning of each Bank's Application region. Specifically, the Bank 0 Application version is located at address 0x00002000, while the Bank 1 Application version is located at address 0x00042000.

After checking the version information, if the firmware in Bank 1 is the latest version, the CSC verifies the firmware header before deciding to perform a bank swap. If the firmware has an incorrect header, or if Bank 0 already contains the latest version, the bank swap is not executed.

#### 2.3.1.2 CSC - Linker File (Bootloader.cmd)

```
--define=_BOOT_SIZE_=(8*1024)

MEMORY
{
    FLASH_BOOT      (RX)  : origin = 0x00000000,  length = _BOOT_SIZE_
    FLASH_APP       (RX)  : origin = _BOOT_SIZE_, length = (0x00040000 - _BOOT_SIZE_)
}

SECTIONS
{
    .intvecs      : > 0x00000000
    .text         : palign(8) {} > FLASH_BOOT
    .const        : palign(8) {} > FLASH_BOOT
    .cinit        : palign(8) {} > FLASH_BOOT
    .pinit        : palign(8) {} > FLASH_BOOT
    .rodata       : palign(8) {} > FLASH_BOOT
    .ARM.exidx    : palign(8) {} > FLASH_BOOT
    .init_array   : palign(8) {} > FLASH_BOOT
    .binit        : palign(8) {} > FLASH_BOOT
}
```

The CSC boundary is set to 8,192 bytes (0x0000 ~ 0x2000). Users must align the boundary to 0x400 due to the Arm Cortex-M0+ VTOR (Vector Table Offset Register) alignment requirement of 0x100 and the flash erase sector size of 1 kB (0x400).

### *2.3.2 App (Bankswap_G3519_gpio_output_toggle_v2_SW_Version55_CRC32)*

#### 2.3.2.1 App - Main Function (Bankswap_G3519_gpio_output_toggle_v2_SW_Version55_CRC32.c)

```
#define APP_VERSION 55 // 0 ~ 4294967295

/* Version information */
__attribute__((section(".version_info"), retain))
const uint32_t gVersionInfo[64] = {APP_VERSION, [1 ... 63] = 0xFFFFFFFF};

while (1) {
    /* Wait until new FW is downloaded */
    while (!gFrameReady) {
        __WFI();
    }
    gFrameReady = false;

    func_CRC32_check();

    if (gCRCChecksumMatch) {
        gCRCChecksumMatch = false;
        func_reset_sectors();
        func_program_to_bank1();
        DL_SYSCTL_resetDevice(DL_SYSCTL_RESET_BOOT);
    }
}
```

After downloading the new firmware, the 'gFrameReady' flag is set, triggering the following processes in sequence: CRC check (func_CRC32_check), flash sector reset (func_reset_sectors), and writing to flash (func_program_to_bank1). Upon completion, a BOOTRST is triggered, and the CSC performs the bank swap between Bank 0 and Bank 1.

### 2.3.2.2 App – UART ISR (Bankswap_G3519_gpio_output_toggle_v2_SW_Version55_CRC32.c)

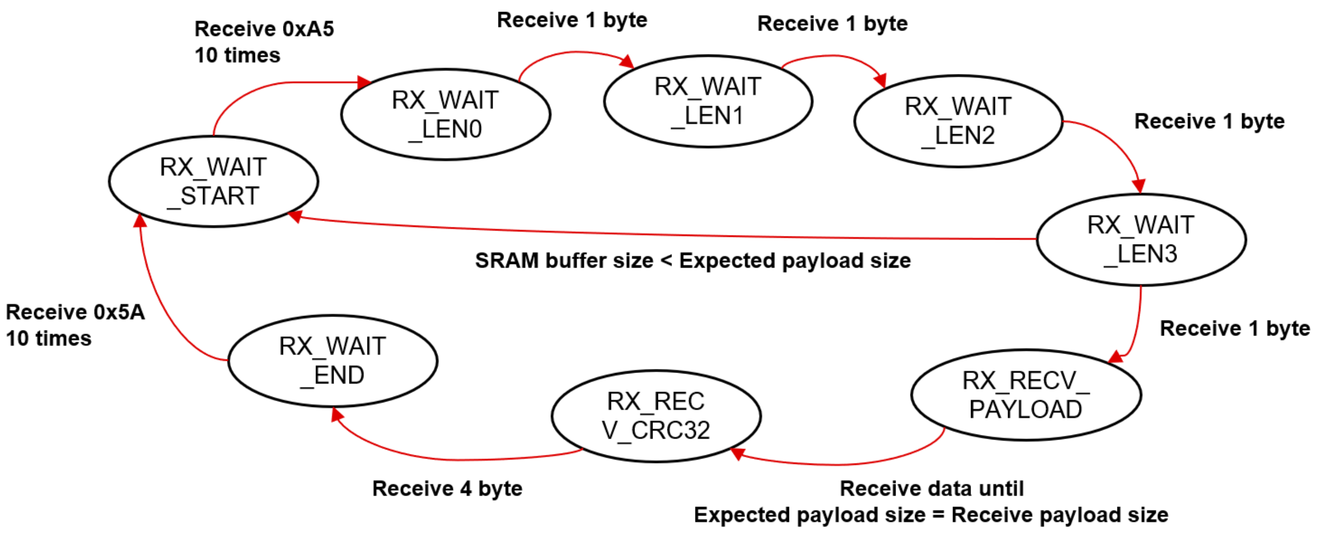


**Figure 2-3. App – UART ISR State Machine**

The UART ISR (Interrupt Service Routine) is designed based on a state machine architecture. The process is divided into five states: receiving the header (RX_WAIT_START), expected data length (RX_WAIT_LEN0, 1, 2, 3), payload (RX_RECV_PAYLOAD), CRC32 (RX_ RECV_CRC32), and EOP (End of Packet, RX_ WAIT_END).

The implementation also validates the expected payload size to prevent buffer overflow.

### 2.3.2.3 App - Linker File (device_linker.cmd)

```
--define=_BOOT_SIZE_=(8*1024)
--define=_VERSION_SIZE_=(256)
--define=_TOTAL_SIZE_=(8*1024+256)

MEMORY
{
    BOOT            (RX)   : origin = 0x00000000,  length = _BOOT_SIZE_
    FLASH_VERSION   (RWX)  : origin = _BOOT_SIZE_,  length = _VERSION_SIZE_
    FLASH           (RX)   : origin = _BOOT_SIZE_ + _VERSION_SIZE_, length = 0x00040000 -
_BOOT_SIZE_ - _VERSION_SIZE_
    SRAM_BANK0      (RWX)  : origin = 0x20200000,  length = 0x00010000
    SRAM_BANK1      (RWX)  : origin = 0x20210000,  length = 0x00010000
    BCR_CONFIG      (R)    : origin = 0x41C00000,  length = 0x000000FF
    BSL_CONFIG      (R)    : origin = 0x41C00100,  length = 0x00000080
    DATA            (R)    : origin = 0x41D00000,  length = 0x00004000
}
SECTIONS
{
    .version_info : palign(8) {} > FLASH_VERSION
    .intvecs:   > _TOTAL_SIZE_
    .text    : palign(8) {} > FLASH
    .const   : palign(8) {} > FLASH
    .cinit   : palign(8) {} > FLASH
    .pinit   : palign(8) {} > FLASH
    .rodata  : palign(8) {} > FLASH
    .ARM.exidx     : palign(8) {} > FLASH
    .init_array    : palign(8) {} > FLASH
    .binit         : palign(8) {} > FLASH
}
```

The Application region is located immediately after the CSC, starting at address 0x0000.2000.

At the beginning of the Application region, version information (.version_info) is stored in uint32_t format, which requires only 4 bytes. However, due to the Arm Cortex-M0+ VTOR (Vector Table Offset Register) alignment requirement of 0x100 (256 bytes), the version information section is allocated 0x100 (256 bytes).

Flash memory capacity varies depending on the MCU model. Therefore, users must configure the appropriate memory boundaries for their specific device. This example is based on the MSPM0G3519, which has 256 kB of flash memory.

## 2.4 Implementation

### 2.4.1 Implementation Overview

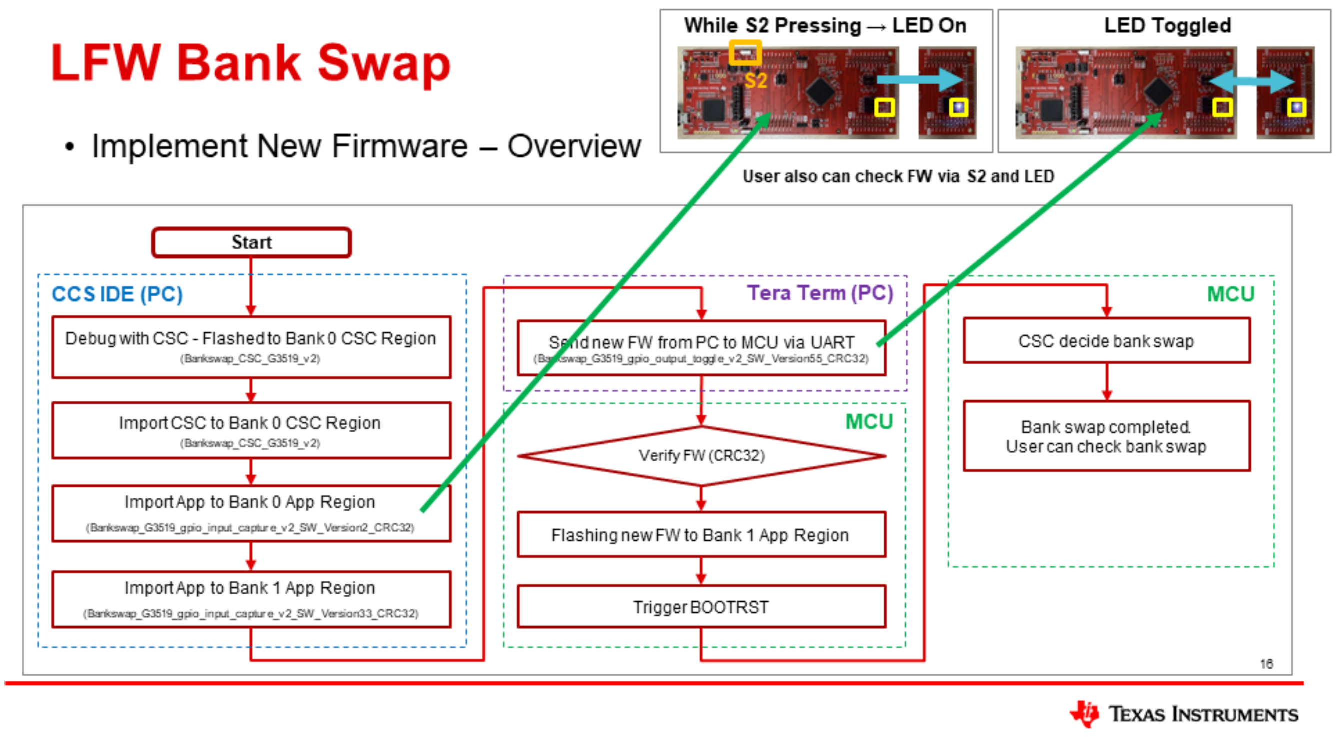

LFW Bank Swap
• Implement New Firmware – Overview
While S2 Pressing → LED On
S2
LED Toggled
User also can check FW via S2 and LED
Start
CCS IDE (PC)
Debug with CSC - Flashed to Bank 0 CSC Region
(Bankswap_CSC_G3519_v2)
Import CSC to Bank 0 CSC Region
(Bankswap_CSC_G3519_v2)
Import App to Bank 0 App Region
(Bankswap_G3519_gpio_input_capture_v2_SW_Version2_CRC32)
Import App to Bank 1 App Region
(Bankswap_G3519_gpio_input_capture_v2_SW_Version33_CRC32)
Tera Term (PC)
Send new FW from PC to MCU via UART
(Bankswap_G3519_gpio_output_toggle_v2_SW_Version55_CRC32)
MCU
Verify FW (CRC32)
Flashing new FW to Bank 1 App Region
Trigger BOOTRST
MCU
CSC decide bank swap
Bank swap completed.
User can check bank swap
16
Texas Instruments

**Figure 2-4. Implementation Overview**

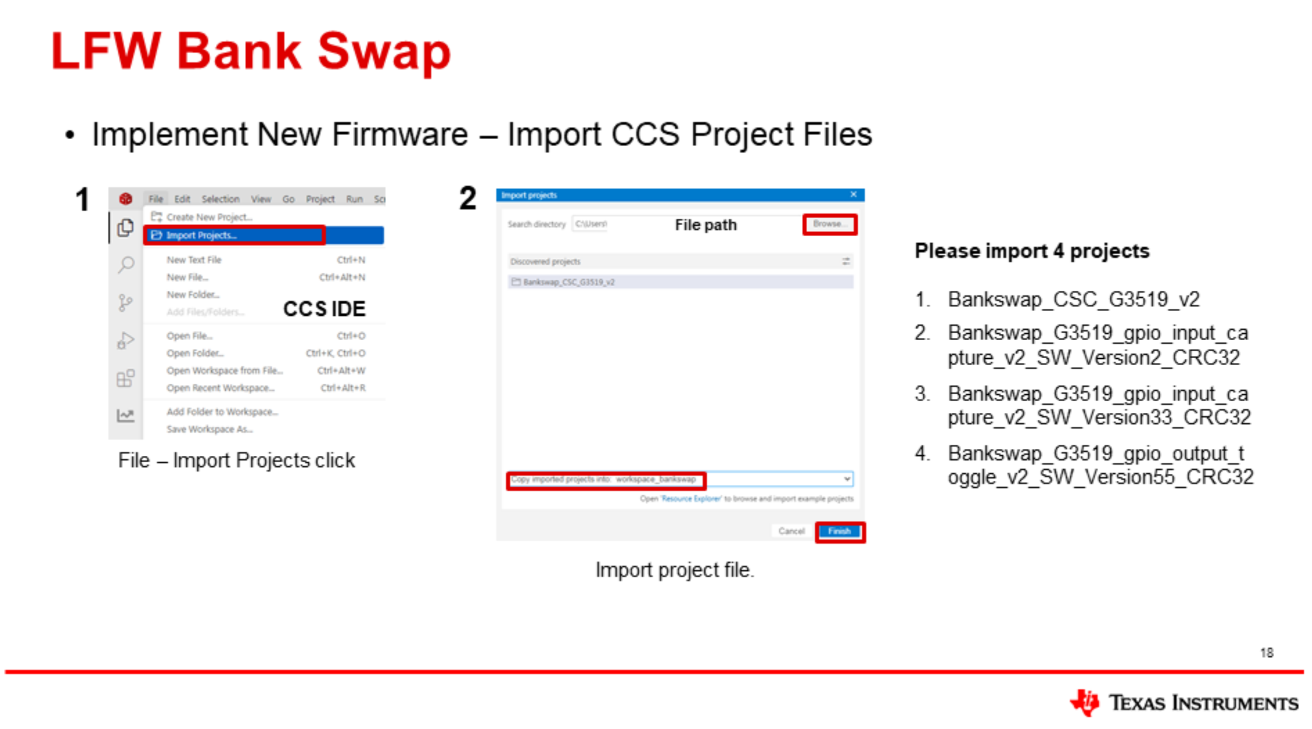
Firmware update completion can be verified in CCS IDE, but it can also be checked using the switch and LED indicators.

Before sending the new firmware, the application 'Bankswap_G3519_gpio_input_capture_v2_SW_Version2_CRC32 (without pressing NRST)' or 'Bankswap_G3519_gpio_input_capture_v2_SW_Version33_CRC32 (with NRST pressed)' is running in the Bank 0 application region. During this time, the LED turns on when the user presses S2.

After sending the new firmware, the application Bankswap_G3519_gpio_output_toggle_v2_SW_Version55_CRC32 runs in the Bank 0 application region, and the LED starts blinking, indicating that the firmware update has been successfully applied.

### 2.4.2 Implementation Process

### 2.4.2.1 Import CCS Project Files (TI CCS IDE)


LFW Bank Swap

- Implement New Firmware – Import CCS Project Files

1 File – Import Projects click

2 Import project file.

**Please import 4 projects**

1. Bankswap_CSC_G3519_v2
2. Bankswap_G3519_gpio_input_capture_v2_SW_Version2_CRC32
3. Bankswap_G3519_gpio_input_capture_v2_SW_Version33_CRC32
4. Bankswap_G3519_gpio_output_toggle_v2_SW_Version55_CRC32

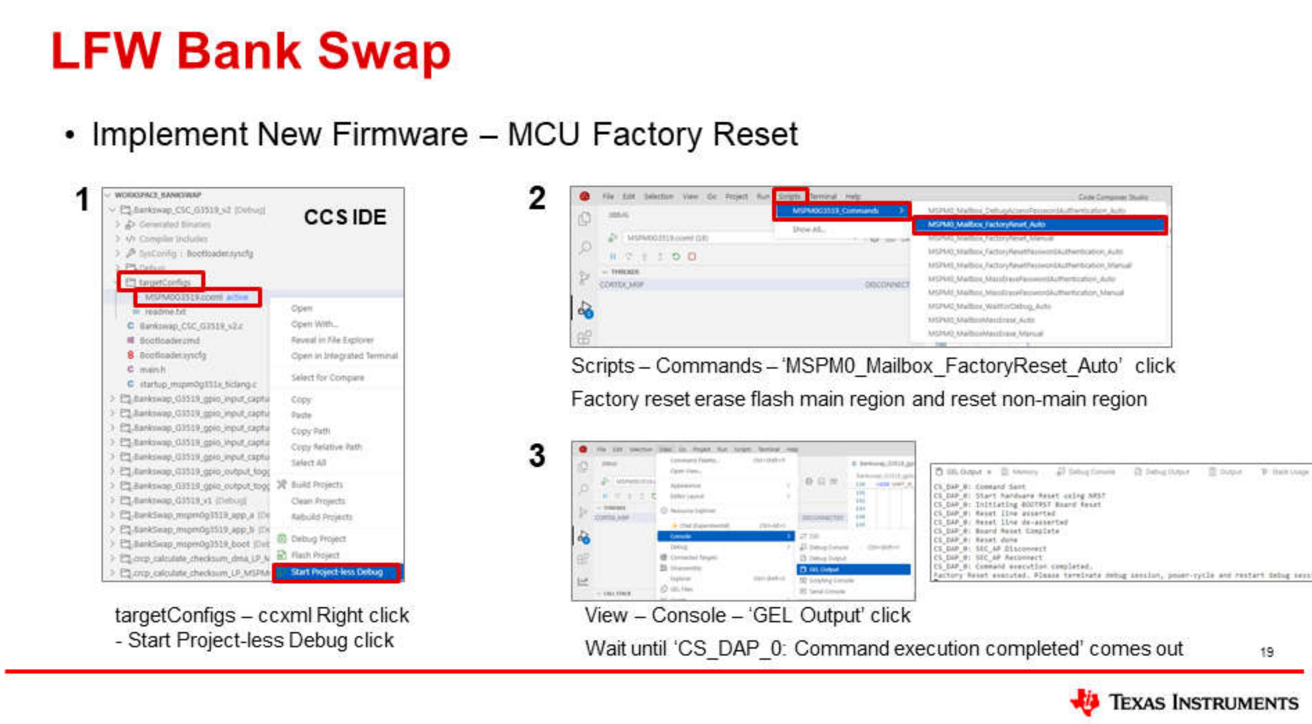
18

Figure 2-5. Import CCS Project Files

Import the following 4 projects:

- Bankswap_CSC_G3519_v2
- Bankswap_G3519_gpio_input_capture_v2_SW_Version2_CRC32
- Bankswap_G3519_gpio_input_capture_v2_SW_Version33_CRC32
- Bankswap_G3519_gpio_output_toggle_v2_SW_Version55_CRC32

#### 2.4.2.2 Conduct MCU Factory Reset (TI CCS IDE)



LFW Bank Swap

- Implement New Firmware – MCU Factory Reset

1 CCS IDE

targetConfigs – ccxml Right click
- Start Project-less Debug click

2

Scripts – Commands – 'MSPM0_Mailbox_FactoryReset_Auto' click

Factory reset erase flash main region and reset non-main region

3

```
CS_DAP_0: Command Sent
CS_DAP_0: Start hardware Reset using NRST
CS_DAP_0: Initiating BOOTRST Board Reset
CS_DAP_0: Reset line asserted
CS_DAP_0: Reset line de-asserted
CS_DAP_0: Board Reset Complete
CS_DAP_0: Reset done
CS_DAP_0: SEC_AP Disconnect
CS_DAP_0: SEC_AP Reconnect
CS_DAP_0: Command execution completed.
Factory Reset executed. Please terminate debug session, power-cycle and restart debug session.
```

View – Console – 'GEL Output' click

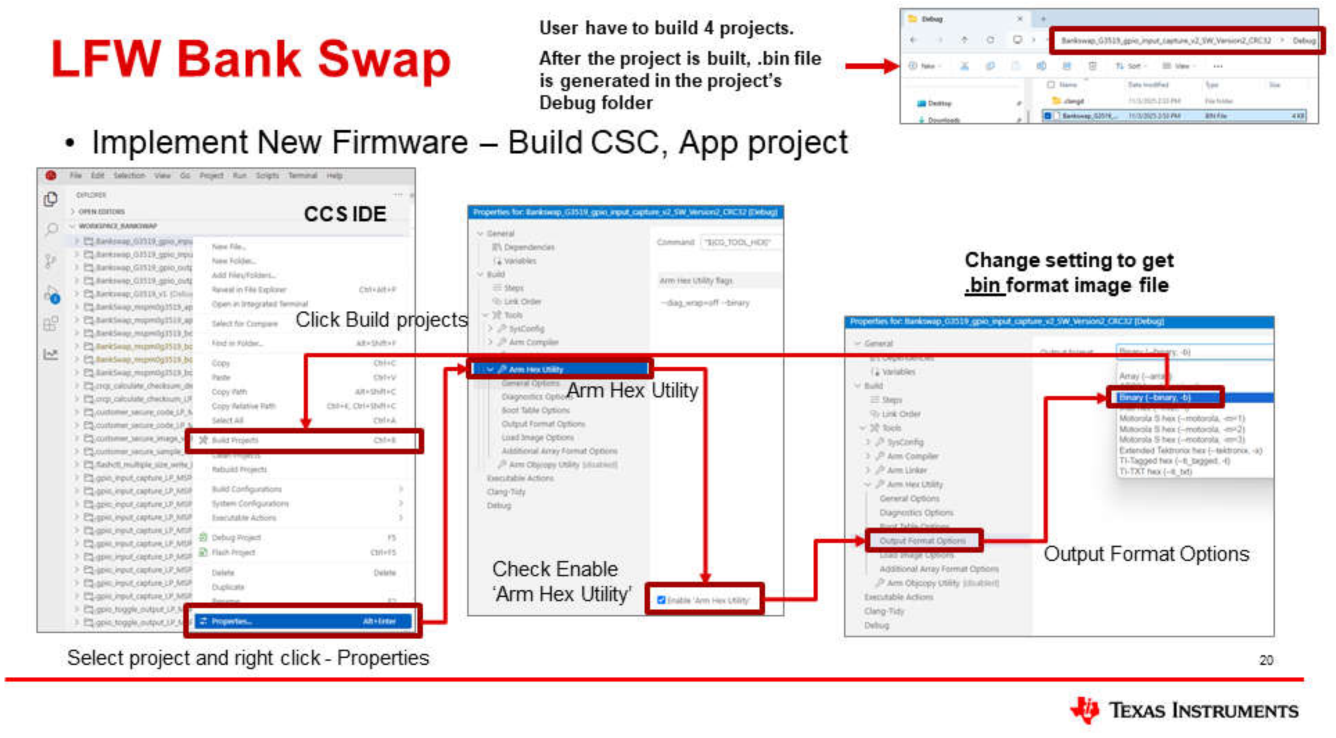
Wait until 'CS_DAP_0: Command execution completed' comes out

19

Figure 2-6. Conduct MCU Factory Reset

To avoid potential conflicts, TI strongly recommends performing a factory reset. The CCS IDE provides a built-in factory reset function, and users can select any one of the four 'MSPM0G3519.ccxml' files to execute the reset.

#### 2.4.2.3 Build CSC, App in CCS (TI CCS IDE)



LFW Bank Swap

User have to build 4 projects.

After the project is built, .bin file is generated in the project's Debug folder

- Implement New Firmware – Build CSC, App project

CCS IDE

Click Build projects

Arm Hex Utility

Check Enable 'Arm Hex Utility'

Change setting to get .bin format image file

Output Format Options

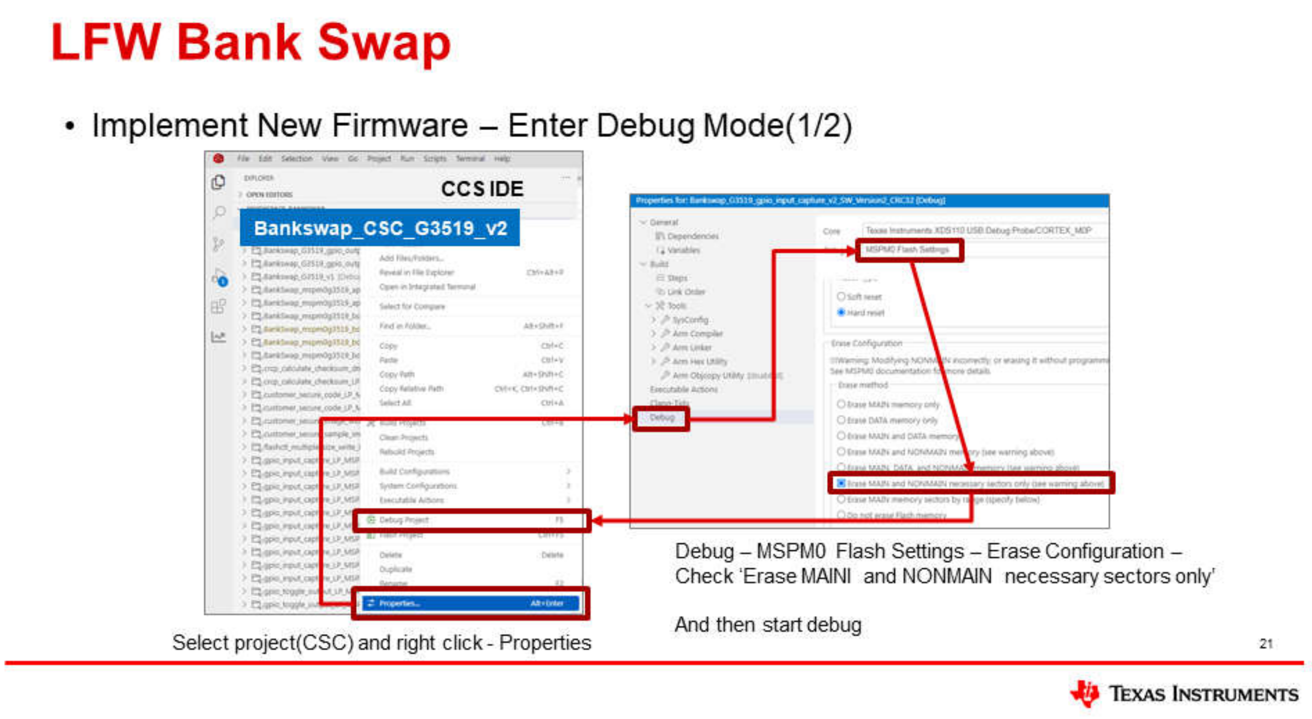
Select project and right click - Properties

20

Figure 2-7. Build CSC, App in CCS

Before building the project, enable the Arm Hex Utility option. After enabling this feature, proceed to build all four projects.

#### 2.4.2.4 Start Debug and Download Image into MCU in CCS (TI CCS IDE)



LFW Bank Swap

- Implement New Firmware – Enter Debug Mode(1/2)

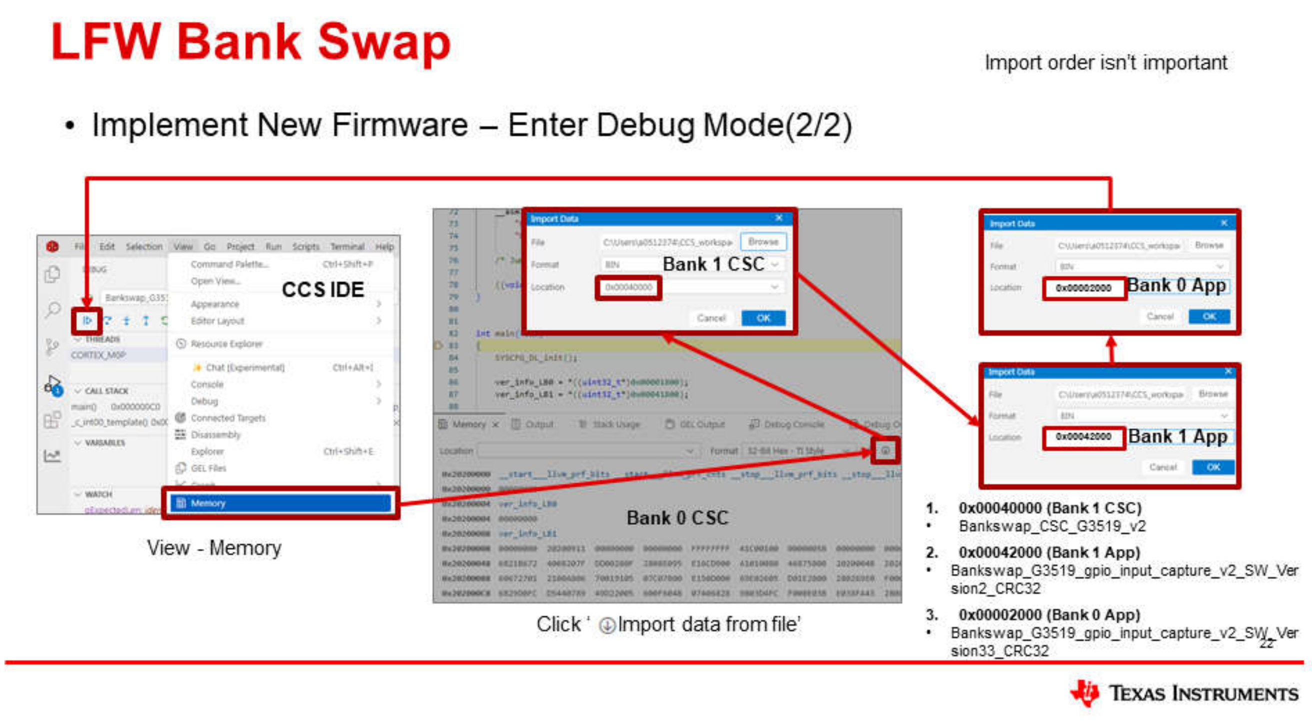
Debug – MSPM0 Flash Settings – Erase Configuration – Check 'Erase MAINI and NONMAIN necessary sectors only'

And then start debug

Select project(CSC) and right click - Properties

21

Figure 2-8. Start Debug and Download Image into MCU in CCS 1



LFW Bank Swap

Import order isn't important

- Implement New Firmware – Enter Debug Mode(2/2)

View - Memory

Click ' Import data from file'

1. 0x00040000 (Bank 1 CSC)
   - Bankswap_CSC_G3519_v2
2. 0x00042000 (Bank 1 App)
   - Bankswap_G3519_gpio_input_capture_v2_SW_Version2_CRC32
3. 0x00002000 (Bank 0 App)
   - Bankswap_G3519_gpio_input_capture_v2_SW_Version33_CRC32

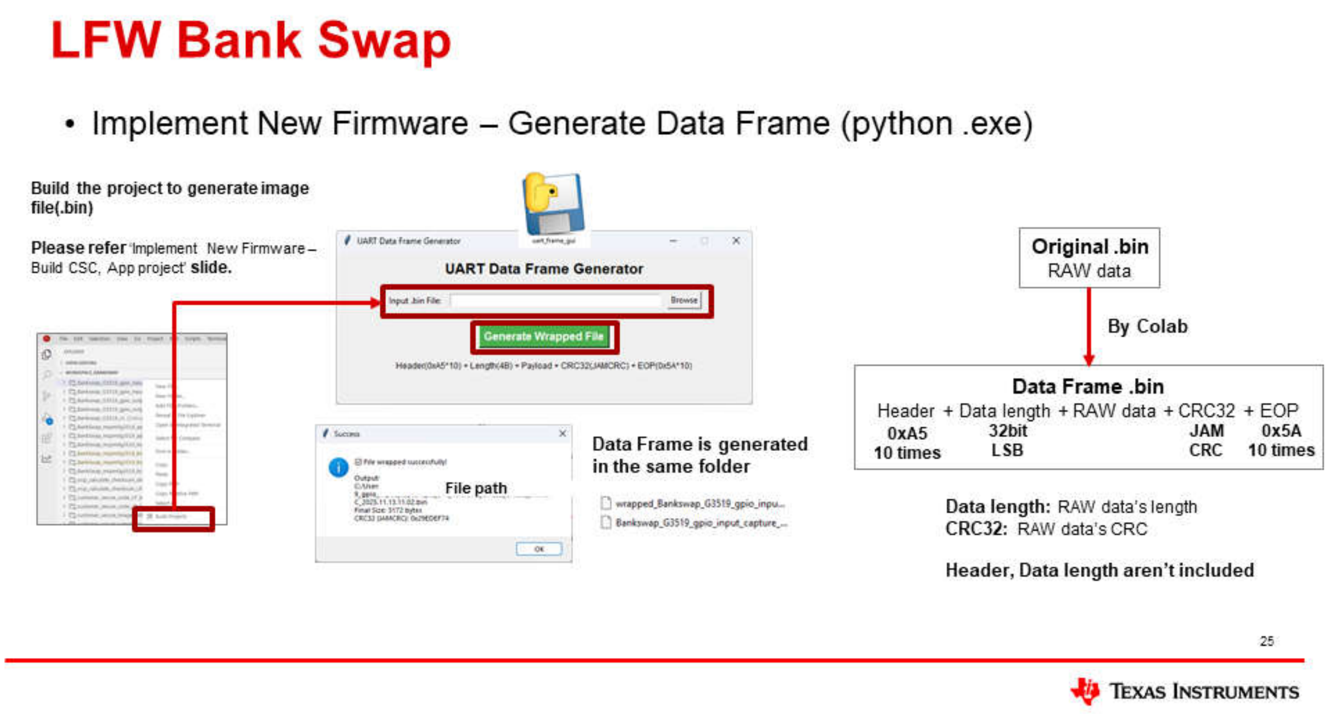
22

Figure 2-9. Start Debug and Download Image into MCU in CCS 2

Before starting to debug the CSC project (Bankswap_CSC_G3519_v2), users must adjust the flashing settings. Ensure that “Erase MAIN and NONMAIN necessary sectors only” is selected.

After entering debug mode, the CSC project is programmed into Bank 0. Users then need to program the CSC into Bank 1 and the Application into both Bank 0 and Bank 1.

#### 2.4.2.5 Generate Data Frame to Send (uart_frame_gui.exe)



## LFW Bank Swap

- Implement New Firmware – Generate Data Frame (python .exe)

Build the project to generate image file(.bin)

Please refer 'Implement New Firmware – Build CSC, App project' slide.

UART Data Frame Generator

Input .bin File: Browse

Generate Wrapped File

Header(0xA5*10) + Length(4B) + Payload + CRC32(JAMCRC) + EOP(0x5A*10)

Success

☑ File wrapped successfully!

Output:
C:\User
_gpio_
C_2025.11.13.11.02.bin
Final Size: 3172 bytes
CRC32 (JAMCRC): 0x29EDEF74

File path

OK

Data Frame is generated in the same folder

wrapped_Bankswap_G3519_gpio_inpu...
Bankswap_G3519_gpio_input_capture_...

Original .bin
RAW data

By Colab

Data Frame .bin

| Header | + Data length | + RAW data | + CRC32 | + EOP |
|---|---|---|---|---|
| 0xA5 10 times | 32bit LSB | | JAM CRC | 0x5A 10 times |

**Data length:** RAW data's length
**CRC32:** RAW data's CRC

**Header, Data length aren't included**

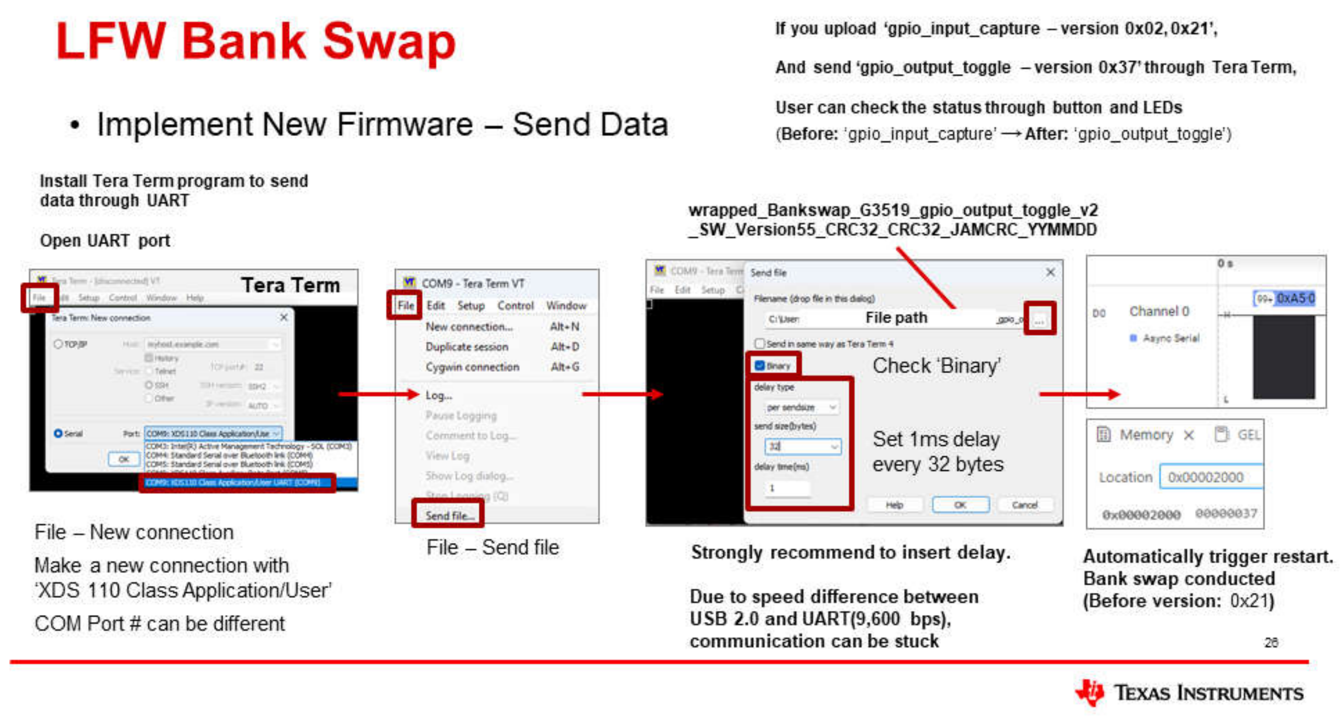
25

Figure 2-10. Generate Data Frame to Send

To improve efficiency and stability, a data-frame structure is used. The firmware data frame consists of a Header (0xA5 × 10), Data Length (32-bit, LSB first), Raw Data, CRC32 (JAMCRC), and an End of Packet (EOP, 0x5A × 10). The Data Length field represents the size of the Raw Data, and the CRC32 value is calculated over the Raw Data.

TI provides an executable tool that simplifies the generation of firmware data frames. By uploading a raw binary (.bin) file, the program automatically produces the corresponding firmware data frame.

### 2.4.2.6 Send New FW via UART in PC (Tera Term)


LFW Bank Swap

- Implement New Firmware – Send Data

If you upload 'gpio_input_capture – version 0x02, 0x21',
And send 'gpio_output_toggle – version 0x37' through Tera Term,
User can check the status through button and LEDs
(Before: 'gpio_input_capture' → After: 'gpio_output_toggle')

Install Tera Term program to send data through UART

Open UART port

wrapped_Bankswap_G3519_gpio_output_toggle_v2_SW_Version55_CRC32_CRC32_JAMCRC_YYMMDD

File – New connection
Make a new connection with 'XDS 110 Class Application/User'
COM Port # can be different

File – Send file

Check 'Binary'
Set 1ms delay every 32 bytes

Strongly recommend to insert delay.
Due to speed difference between USB 2.0 and UART(9,600 bps), communication can be stuck

Automatically trigger restart.
Bank swap conducted
(Before version: 0x21)

**Figure 2-11. Send New FW via UART in PC**

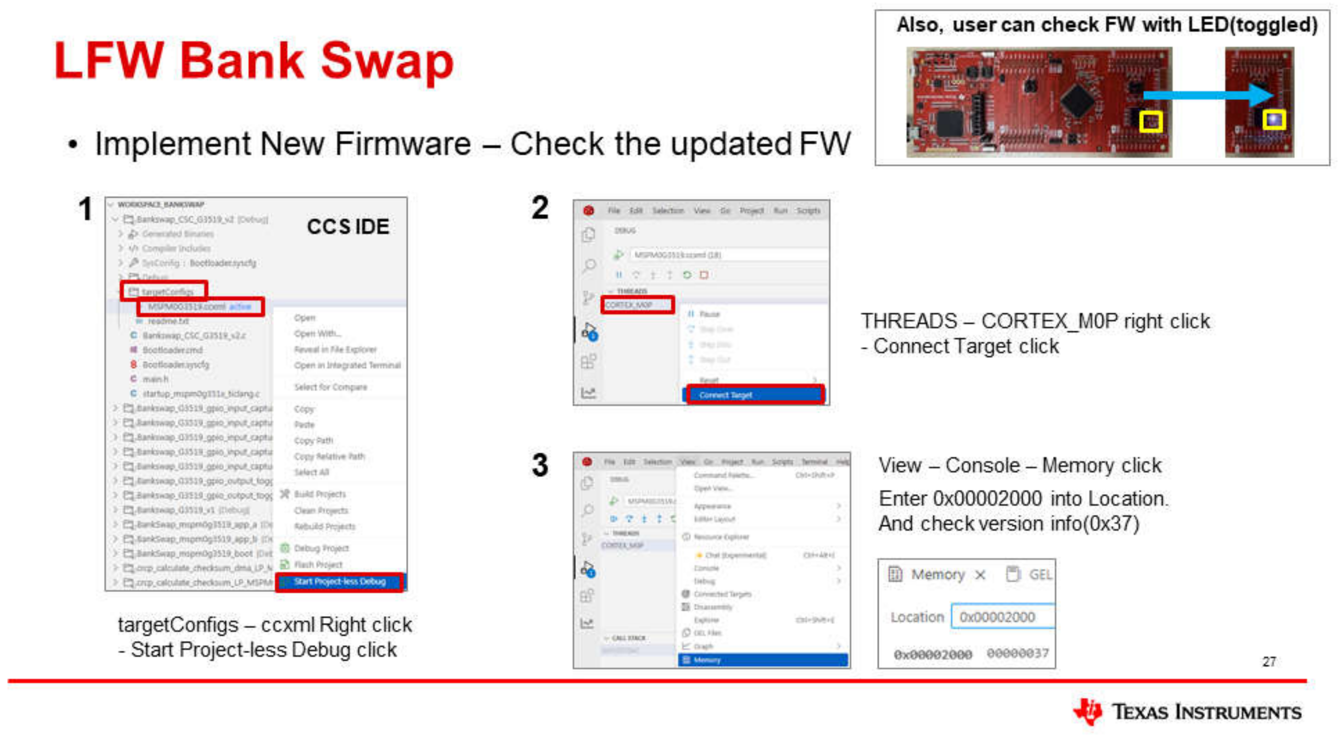
Connect the XDS-110, which functions as a USB-to-UART bridge. Using Tera Term, the firmware data frame (.bin) is transferred to the MCU.

Due to the speed difference between USB 2.0 and UART, communication may occasionally stall. To address this, TI recommends inserting delays. In this implementation, a 1 ms delay is added every 32 bytes. This issue is related to the XDS-110 and does not affect the MCU.

After the transfer is complete, the Application verifies the firmware using CRC32. If the firmware is valid, it is programmed into Bank 1. Upon completing the flashing process, the Application triggers a BOOTRST, which transfers execution to the CSC. The CSC then checks which bank contains the latest firmware. Firmware version information is stored at the beginning of each application image (Bank 0: 0x0000.2000, Bank 1: 0x0004.2000).

If the newly flashed firmware is the latest version, the CSC initiates a bank swap, exchanging Bank 0 and Bank 1. Instructions for verifying the updated firmware are provided in the next section.

### 2.4.2.7 Check the Updated FW (TI CCS IDE)



LFW Bank Swap

- Implement New Firmware – Check the updated FW

Also, user can check FW with LED(toggled)

1 targetConfigs – ccxml Right click
- Start Project-less Debug click

2 THREADS – CORTEX_M0P right click
- Connect Target click

3 View – Console – Memory click
Enter 0x00002000 into Location.
And check version info(0x37)

**Figure 2-12. Check the Updated FW**

Users can verify whether the firmware has been swapped using CCS IDE. Open the debug view and check the Application firmware version in Bank 0 at address 0x0000.2000. The firmware version is stored in uint32_t format, aligned to LSB.

## 3 Summary

This document provides basic theory of bank swap and CSC. And guide how to implement live firmware update with example codes and program(.exe).

With this example, user can update firmware without debugging tool (SWD) while running application program.

This is the basic example of live firmware update via bank swap and user can add functions like security, stability based on this code.

## 4 References

- Flash Multi Bank Feature in MSPM0 Family - https://www.ti.com/lit/an/spradn2/spradn2.pdf?ts=1763307924246&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252Fko-kr%252FMSPM0G3519
- MSPM0 G-Series 80-MHz Microcontrollers Technical Reference Manual (Rev. C) - https://www.ti.com/lit/ug/slau846c/slau846c.pdf?ts=1763536485745&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252Fko-kr%252FMSPM0G3519
- Arm M0+ Documentation 'Vector Table Offset Register' - https://developer.arm.com/documentation/dui0662/b/Cortex-M0--Peripherals/System-Control-Block/Vector-Table-Offset-Register