Application Note

通过闪存组交换进行实时固件更新

Ryan Kim *Korea Sales – Major Account*

摘要

本应用手册介绍了使用组交换和 CSC（客户安全代码）的实时固件更新算法并说明了如何在支持双组存储器的 MSPM0 器件上实现该算法。本例中使用 LP-MSPM0G3519 EVM，并且新的固件映像通过 UART 通信传输。

内容

1 简介.....2
2 详细说明.....3
2.1 概述.....3
2.2 方框图.....4
2.3 代码.....5
2.4 实现.....8
3 总结.....16
4 参考资料.....17

商标

1 简介

在器件部署到现场后，开发人员可能会发现意外软件漏洞，或需要添加新功能。在这种情况下，即使 MCU 仍在运行，也需要进行固件更新。但是，使用引导加载程序或 SWD（串行线调试）的传统更新方法并不适用于该场景，这类方法通常需要中止 MCU 的运行。

为解决这一限制，可采用在线固件更新方法，在不中断系统运行的前提下实现固件的实时更新。此方法利用了双组闪存存储器架构，使一个组能够执行现有固件，而另一个组则用于编程新映像。更新和验证过程完成后，会执行组交换以激活新固件。

截至 2025 年 11 月，已有 17 款 MCU 产品支持双组存储器，可通过存储组切换实现在线固件更新。在该实现方案中，新固件会下载到闪存组 1 中，而现有应用程序继续在闪存组 0 中运行。CSC（客户安全代码）用于验证固件版本并确定是否应触发存储组交换。新的固件映像通过 UART 通信从电脑传输到 MCU，该过程采用定义的数据帧结构，其中包含 CRC32 (JAMCRC) 校验和，可确保可靠且无错误的数据传输。

为了高效地执行实时固件更新，本方案提供了一系列软件组件与工具：

- 由 TI 提供
  - CSC（客户安全代码）映像：版本检查和存储组交换操作。
  - 组 0/组 1 应用程序映像：分别包含活动固件和更新后的固件。通过 UART 接收新固件。通过 CRC32 (JAMCRC) 验证固件
  - 数据帧发生器 (uart_frame_gui.exe)：电脑端实用程序，用于将原始.bin 文件转换为具有 CRC32 (JAMCRC) 验证的帧数据流。
- 用户提供
  - Tera Term (https://teratermproject.github.io/index-en.html)：用于将帧固件映像从电脑传输到 MCU（通过 XDS-110 调试器）
  - LP-MSPM0G3519 (https://www.ti.com/tool/LP-MSPM0G3519)：该评估板同时集成了 XDS-110 调试器与 MSPM0G3519SPZR。XDS-110 还提供 USB 转 UART 桥接功能，用于从电脑发送新固件。

请通过以下链接下载 CSC、应用程序映像、数据帧生成器和幻灯片：https://e2e.ti.com/cfs-file/__key/communityserver-discussions-components-files/908/Live-Firmware-Update-via-Bank-Swap_5F00_Shared-files_5F00_260116.zip

有关 MSPM0 中的 CSC（客户安全代码）和多组功能的更多信息，请参阅“MSPM0 系列中的闪存多组功能”应用手册。

通过将双组架构与这些支持工具相结合，开发人员可在不中断器件运行的前提下，在已部署的系统中安全、高效地完成固件更新，从而实现可靠的现场固件更新解决方案。

2 详细说明

2.1 概述

2.1.1 实时固件更新流程

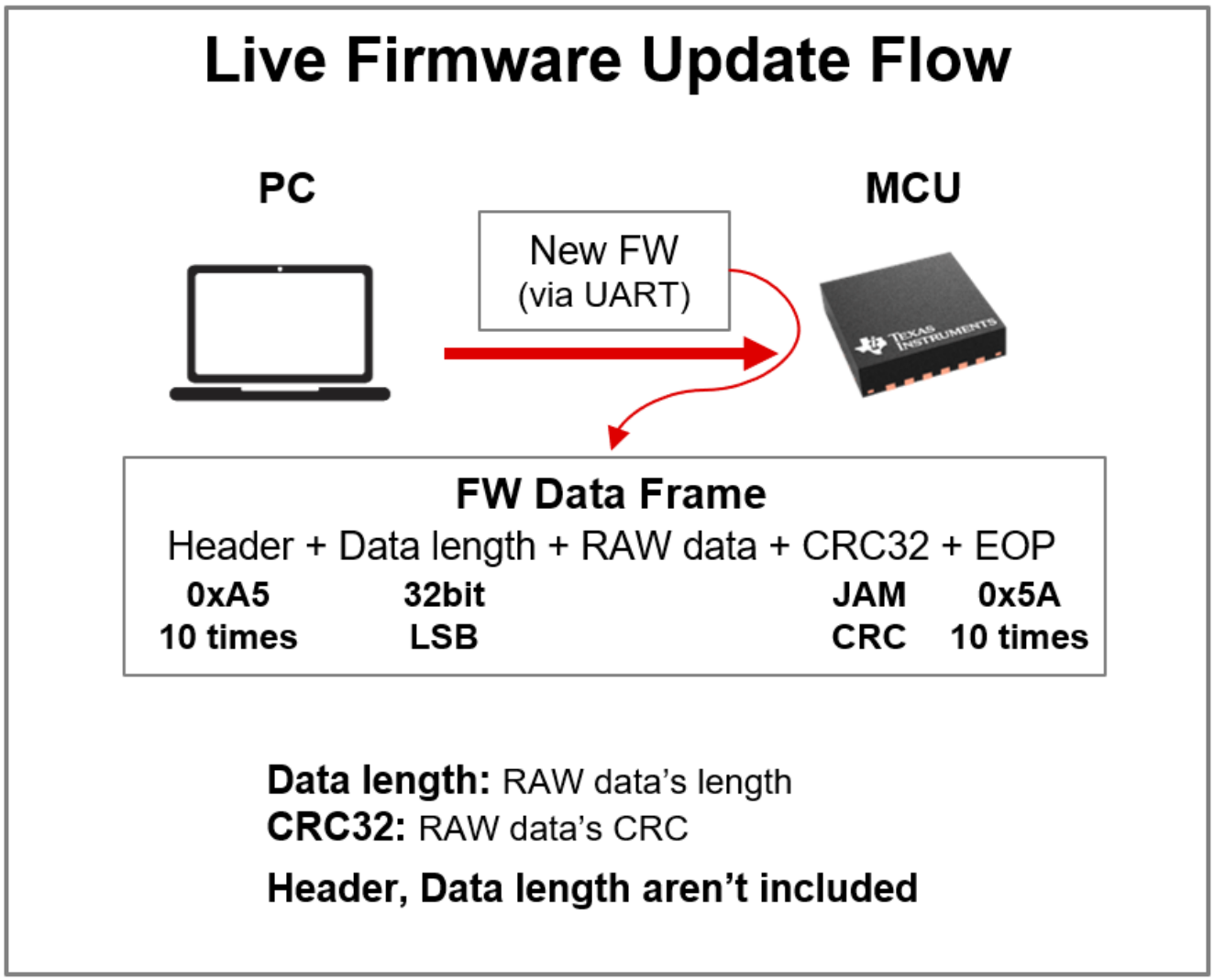

图 2-1. 实时固件更新流程

电脑通过 UART 向 MCU 发送固件数据帧。固件数据帧包含帧头 (0xA5 × 10)、数据长度（32 位，LSB 优先）、原始数据、CRC32 (JAMCRC) 和数据包帧尾（EOP，0x5A× 10）。数据长度字段表示原始数据的大小，并且 CRC32 值基于原始数据计算得出。

2.1.2 内存组织

表 2-1. 内存组织

存储器区域	子区域	地址（例如 MSPM0G3519）	注释
组 0 （运行）	CSC	0x0000.0000 ~ 0x0000.1999	与组 1 CSC 相同，决定组交换
	应用程序	0x0000.2000 ~ 0x0003.FFFF	运行应用程序
组 1 （未激活）	CSC	0x0004.0000 ~ 0x0004.1999	与组 0 CSC 相同，决定组交换
	应用程序	0x0004.2000 ~ 0x0007.FFFF	将在此处下载旧固件或新固件

存储器区域分为组 0 和组 1，每个组进一步拆分为 CSC（客户安全代码）部分和应用程序（应用）部分。

CSC 读取存储在每个组起始位置的固件版本（组 0：0x0000.2000，组 1：0x0004.2000，两者均为 uint32_t 格式），以确定哪个组包含最新固件并决定是否需要进行组交换。

例如，如果组 0 已经包含最新固件，则 CSC 不会触发组交换。但是，如果组 1 包含更新的固件版本，则 CSC 会启动组交换。

完成检查后，CSC 会将执行权移交给组 0 中的应用程序。

2.2 方框图

下图为实时固件更新系统的系统方框图。

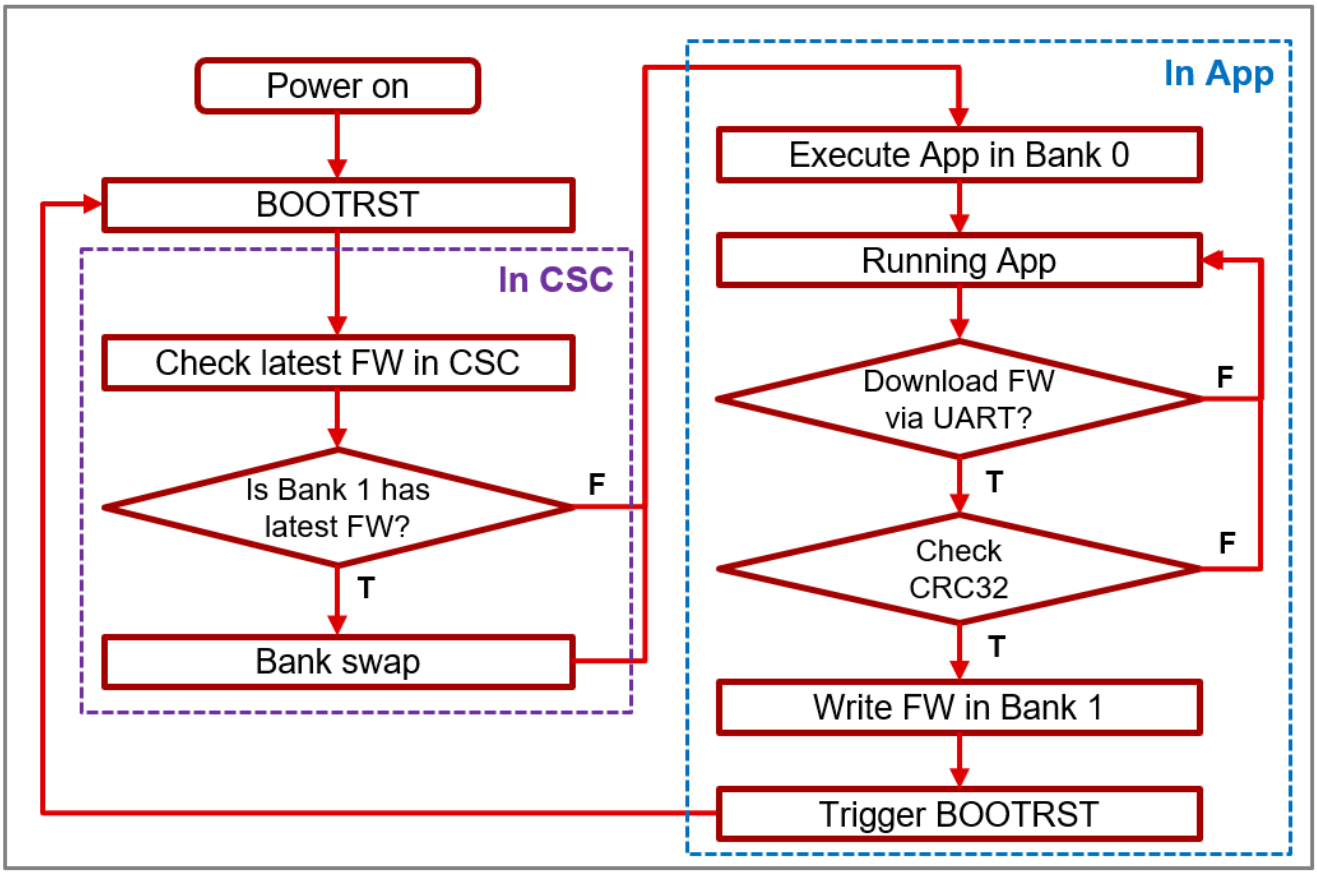

图 2-2. 实时固件更新方框图

该系统由两个执行域组成：CSC 和应用程序。

CSC（客户安全代码）负责验证固件版本并确定是否应执行组交换。

应用程序负责通过 UART 下载新固件、利用 CRC32 校验数据完整性，并执行应用程序映像中包含的用户定义功能。

2.3 代码

2.3.1 CSC（客户安全代码，Bankswap_CSC_G3519_v2）

2.3.1.1 CSC — 主函数 (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
        }
    }
}
```

客户安全代码 (CSC) 会检查组 0 和组 1 中的固件版本，以确定是否需要组交换。版本信息存储在每个闪存组的应用程序区域的起始位置。具体而言，组 0 应用程序版本位于地址 0x00002000，而组 1 应用程序版本位于地址 0x00042000。

检查版本信息后，如果闪存组 1 中的固件是最新版本，CSC 会先验证固件标头，再决定是否执行组交换。如果固件的标头不正确，或者组 0 已经包含最新版本，则不会执行组交换。

2.3.1.2 CSC — 链接器文件 (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
}
```

CSC 边界设置为 8,192 字节 (0x0000 ~ 0x2000)。由于 Arm Cortex-M0+ VTOR (矢量表偏移寄存器) 对齐要求为 0x100，并且闪存擦除扇区大小为 1kB (0x400)，因此用户必须将边界按 0x400 对齐。

2.3.2 应用 (Bankswap_G3519_gpio_output_toggle_v2_SW_Version55_CRC32)

2.3.2.1 应用程序 — 主函数 (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);
    }
}
```

下载新固件后，会设置“gFrameReady”标志，并按顺序触发以下流程：CRC 校验 (func_CRC32_check)、闪存扇区复位 (func_reset_sectors) 和写入闪存 (func_program_to_bank1)。完成后，将触发 BOOTRST，由 CSC 执行组 0 和组 1 之间的组交换。

2.3.2.2 应用程序 – UART ISR (Bankswap_G3519_gpio_output_toggle_v2_SW_Version55_CRC32.c)

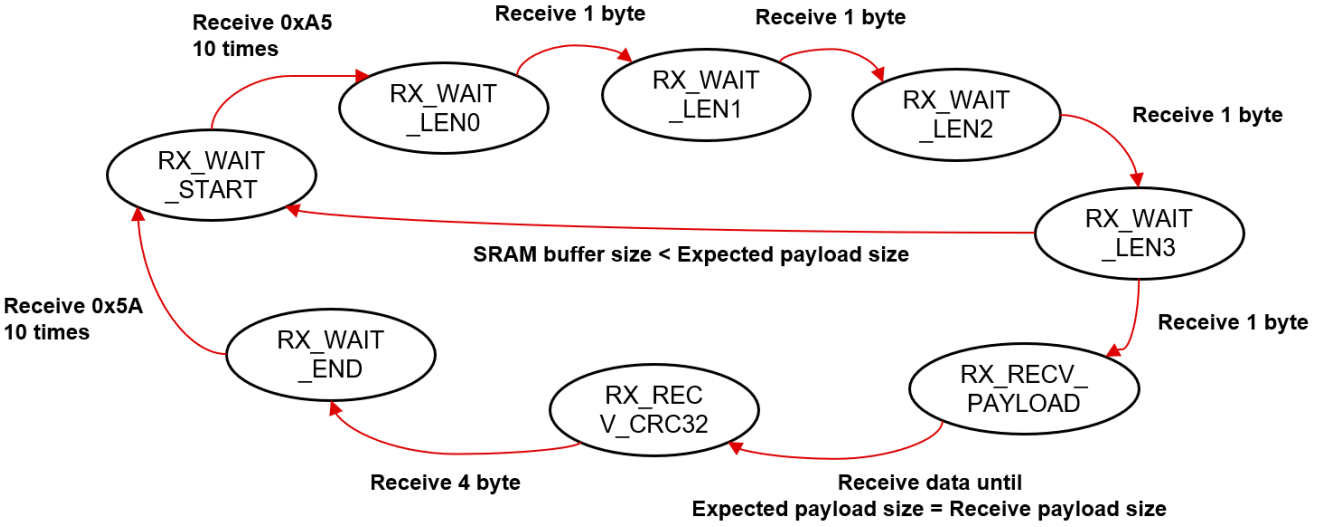

图 2-3. 应用程序 – UART ISR 状态机

UART ISR（中断服务例程）基于状态机架构设计。该过程分为五种状态：接收帧头 (RX_WAIT_START)、预期数据长度 (RX_WAIT_LEN0, 1, 2, 3)、有效负载 (RX_RECV_PAYLOAD)、CRC32 (RX_ RECV_CRC32) 和 EOP（数据包帧尾，RX_ WAIT_END）。

该实现还会验证预期有效载荷大小，防止缓冲区溢出。

2.3.2.3 应用程序 — 链接器文件 (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
}
```

应用程序区域紧跟在 CSC 之后，从 0x0000.2000 地址开始。

在应用程序区域的起始位置，版本信息 (.version_info) 以 uint32_t 格式存储，只需 4 个字节。但是，由于 Arm Cortex-M0+ VTOR（矢量表偏移寄存器）对齐要求为 0x100（256 字节），因此版本信息段被分配 0x100（256 字节）。

闪存存储器容量因 MCU 型号而异。因此，用户必须为其具体器件配置适当的存储器边界。本示例基于具有 256kB 闪存的 MSPM0G3519。

2.4 实现

2.4.1 实现概述

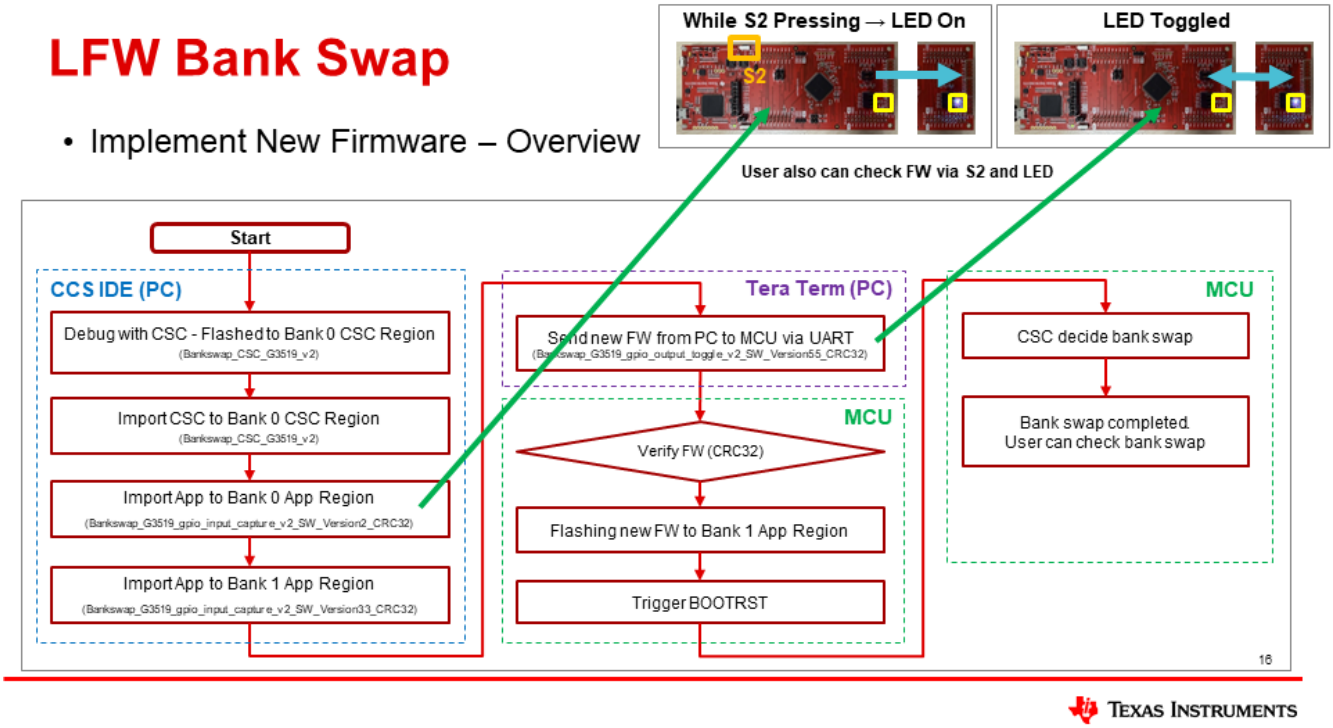

图 2-4. 实现概述

固件更新完成情况可通过 CCS IDE 进行验证，同时也可通过开关与 LED 指示灯进行检查。

在发送新固件前，组 0 应用程序区域中正在运行应用程序“Bankswap_G3519_gpio_input_capture_v2_SW_Version2_CRC32 (without pressing NRST)”或“Bankswap_G3519_gpio_input_capture_v2_SW_Version33_CRC32 (with NRST pressed)”。在此期间，当用户按下 S2 时，LED 亮起。

在发送新固件后，组 0 应用程序区域中运行应用程序 Bankswap_G3519_gpio_output_toggle_v2_SW_Version55_CRC32，此时 LED 灯开始闪烁，表明固件更新已成功应用。

2.4.2 实现流程

2.4.2.1 导入 CCS 项目文件 (TI CCS IDE)

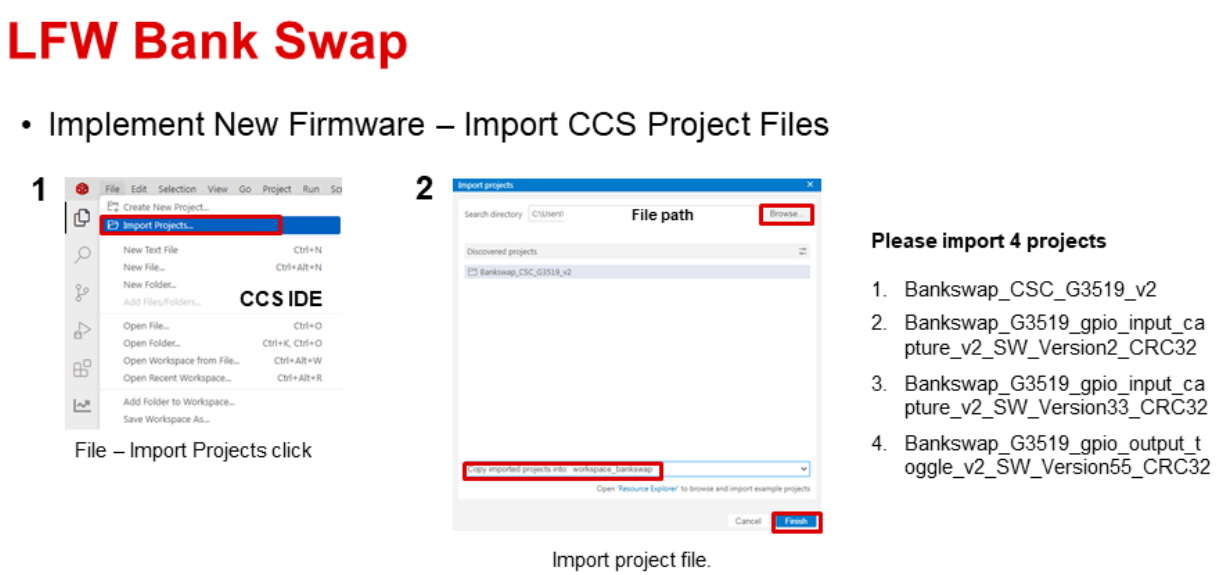

图 2-5. 导入 CCS 项目文件

导入以下 4 个项目：

- 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 执行 MCU 恢复出厂设置 (TI CCS IDE)

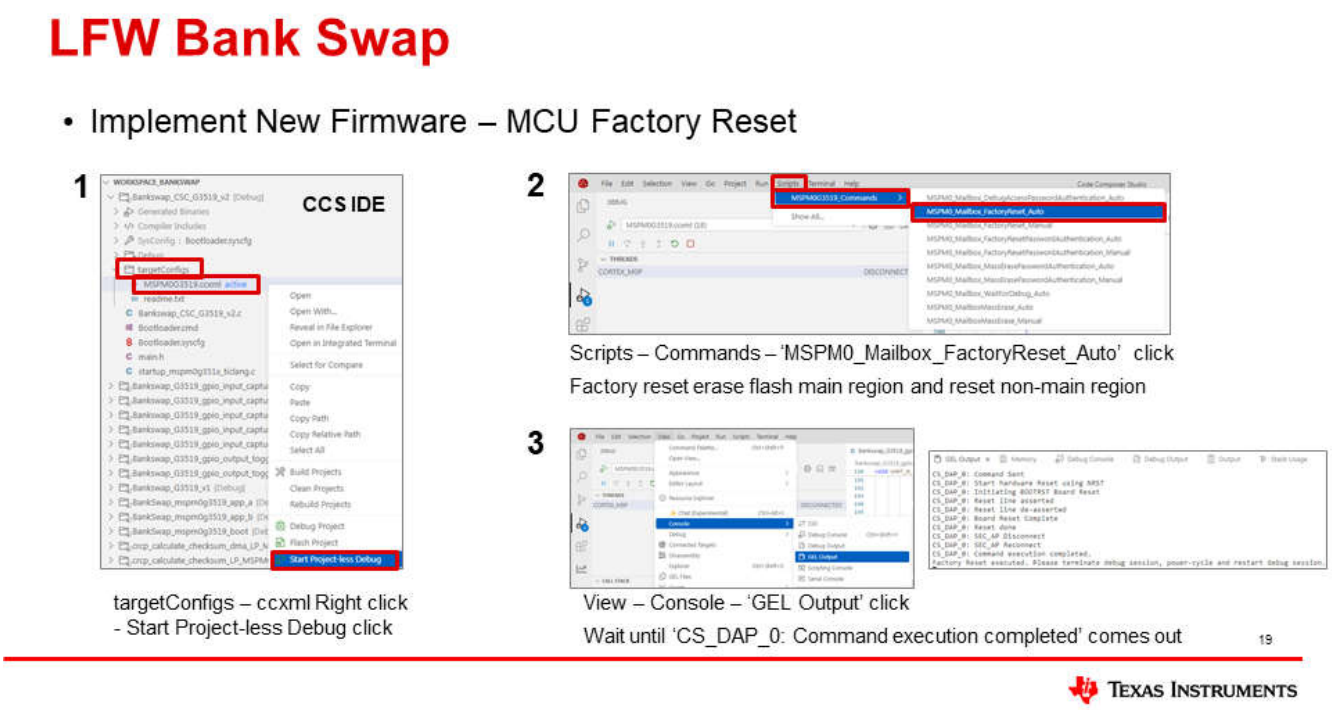

图 2-6. 执行 MCU 恢复出厂设置

为避免潜在冲突，TI 强烈建议恢复出厂设置。CCS IDE 提供内置的恢复出厂设置功能，用户可以从四个“MSPM0G3519.ccxml”文件中任选其一执行恢复出厂设置。

2.4.2.3 在 CCS 中构建 CSC、应用程序 (TI CCS IDE)

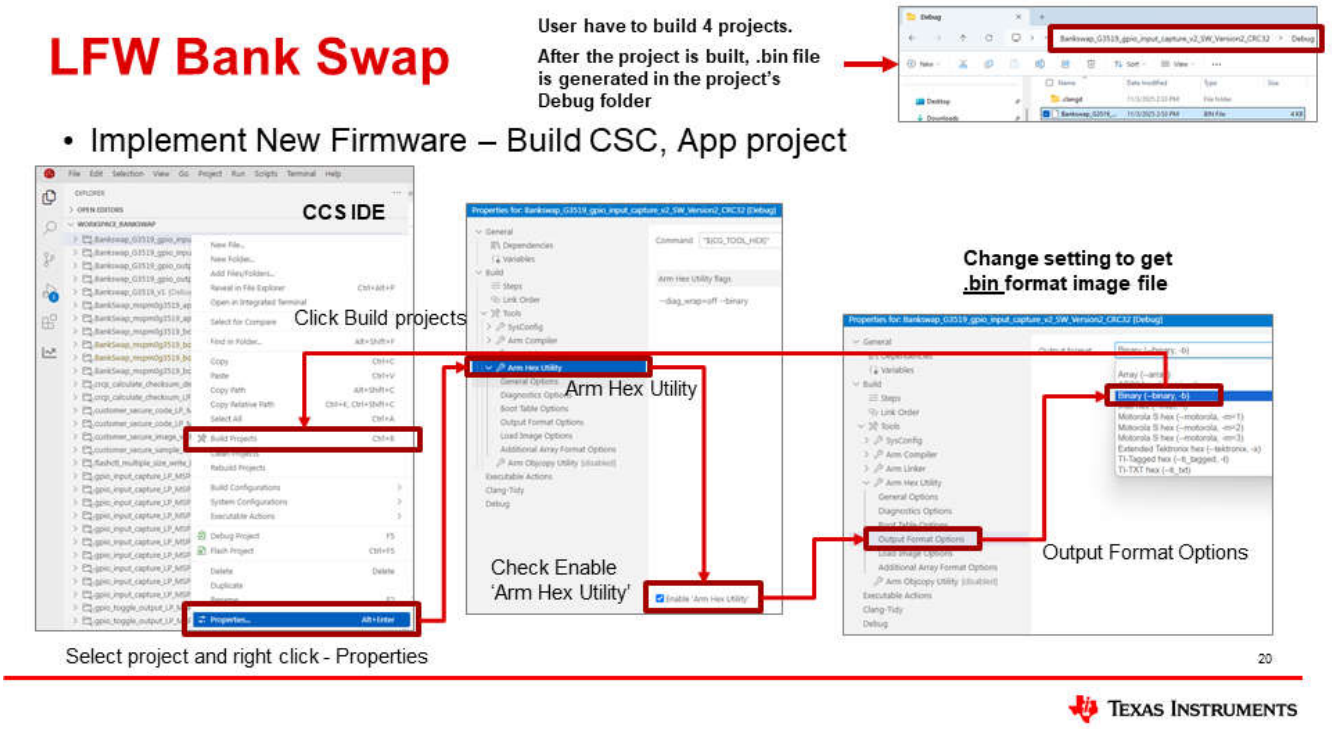

图 2-7. 在 CCS 中构建 CSC、应用程序

在构建项目之前，请启用 Arm Hex Utility 选项。启用此功能后，即可构建所有四个项目。

2.4.2.4 在 CCS 中启动调试并将映像下载到 MCU 中 (TI CCS IDE)

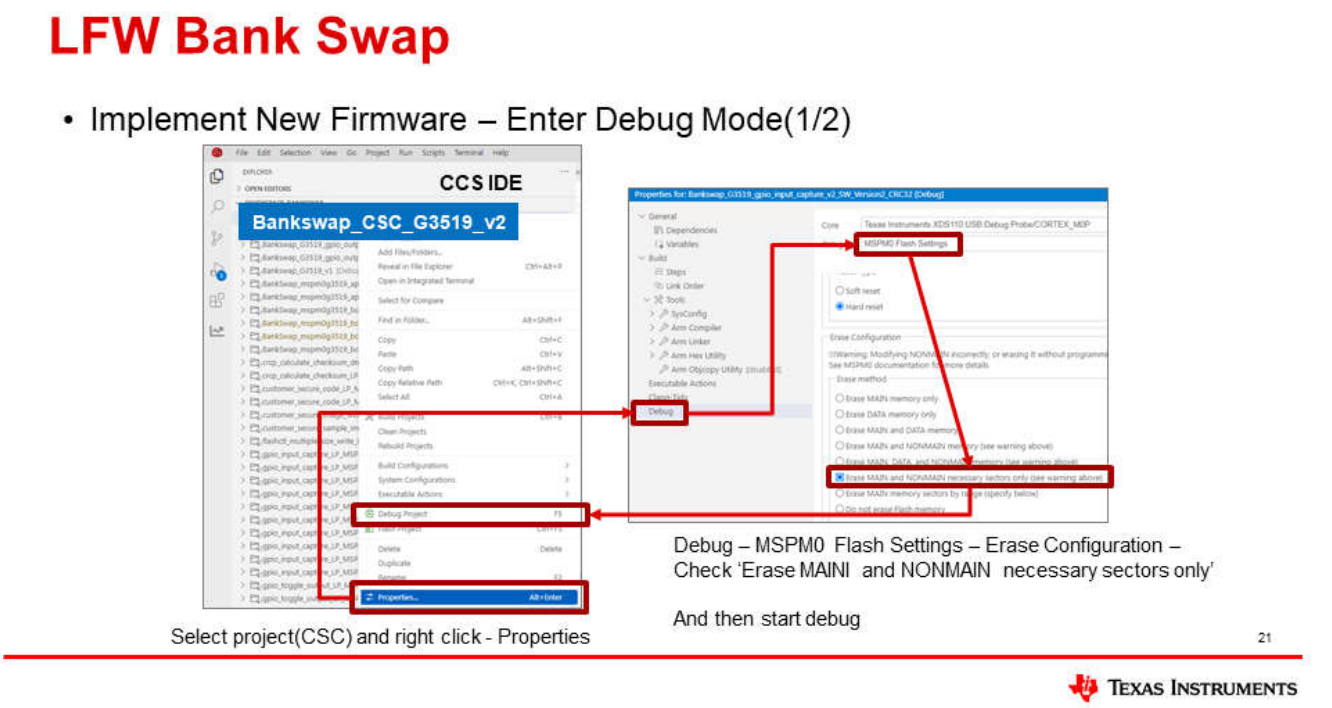

图 2-8. 在 CCS 1 中启动调试并将映像下载到 MCU 中

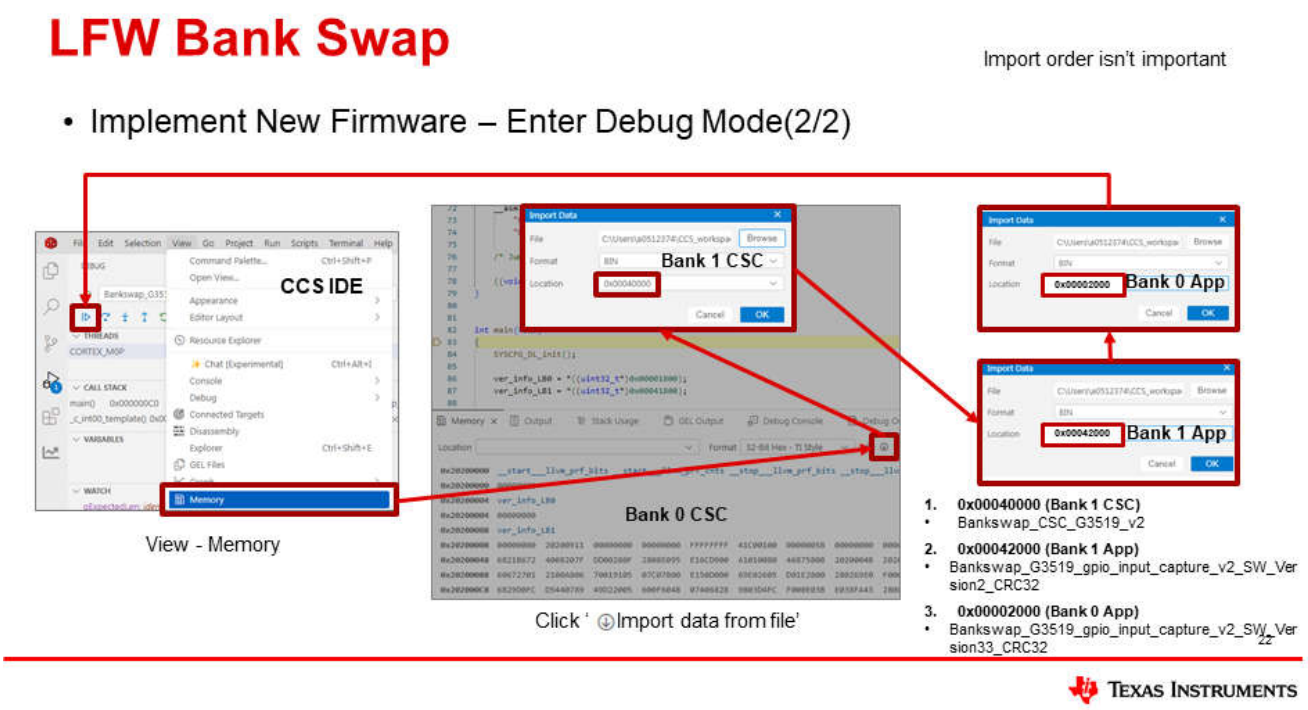

图 2-9. 在 CCS 2 中启动调试并将映像下载到 MCU 中

在开始调试 CSC 项目 (Bankswap_CSC_G3519_v2) 之前，用户必须调整刷写设置。确保已选中“Erase MAIN and NONMAIN necessary sectors only”。

进入调试模式后，CSC 项目被编程到组 0 中。然后，用户需要将 CSC 编程到组 1 中，将应用程序编程到组 0 和组 1 中。

2.4.2.5 生成要发送的数据帧 (uart_frame_gui.exe)

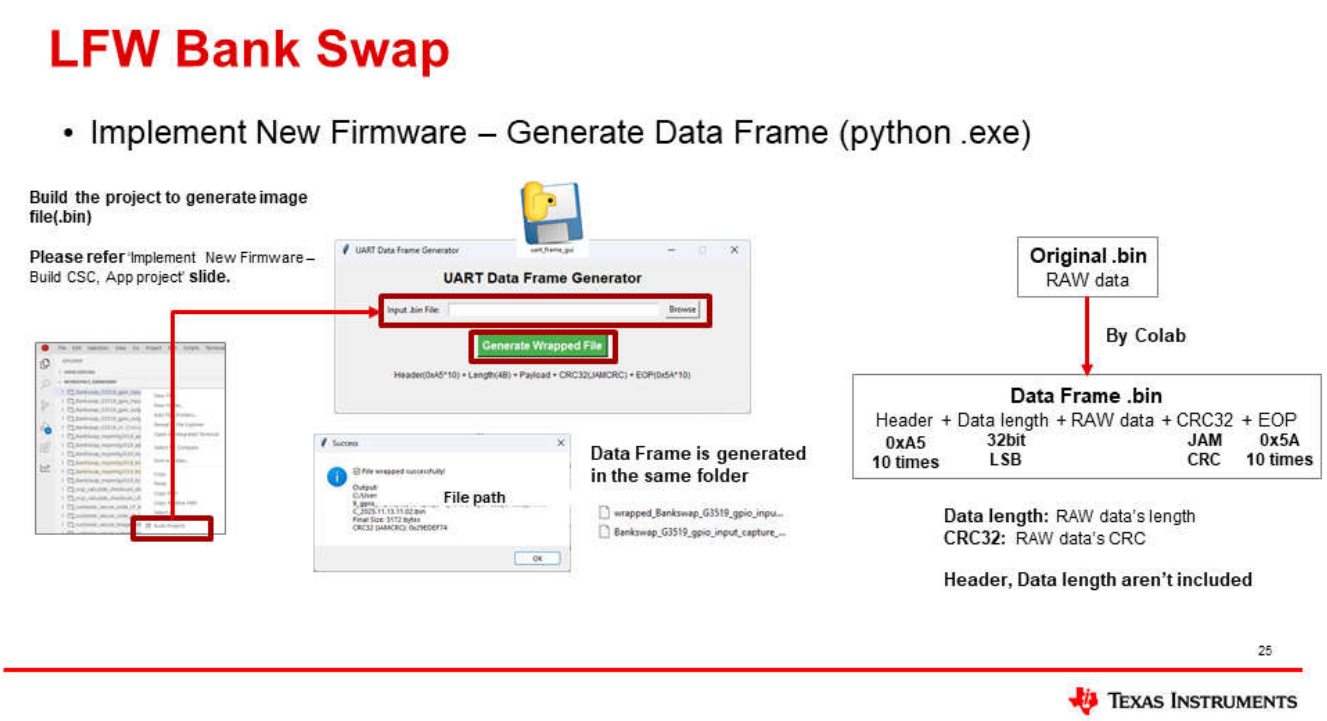

图 2-10. 生成要发送的数据帧

为了提高效率和稳定性，使用了数据帧结构。固件数据帧包含帧头 (0xA5 × 10)、数据长度（32 位，LSB 优先）、原始数据、CRC32 (JAMCRC) 和数据包帧尾（EOP，0x5A× 10）。数据长度字段表示原始数据的大小，并且 CRC32 值基于原始数据计算得出。

TI 提供了一个可执行工具，可简化固件数据帧的生成。通过上传原始二进制 (.bin) 文件，该程序即可自动生成相应的固件数据帧。

2.4.2.6 在电脑上通过 UART 发送新固件 (Tera Term)

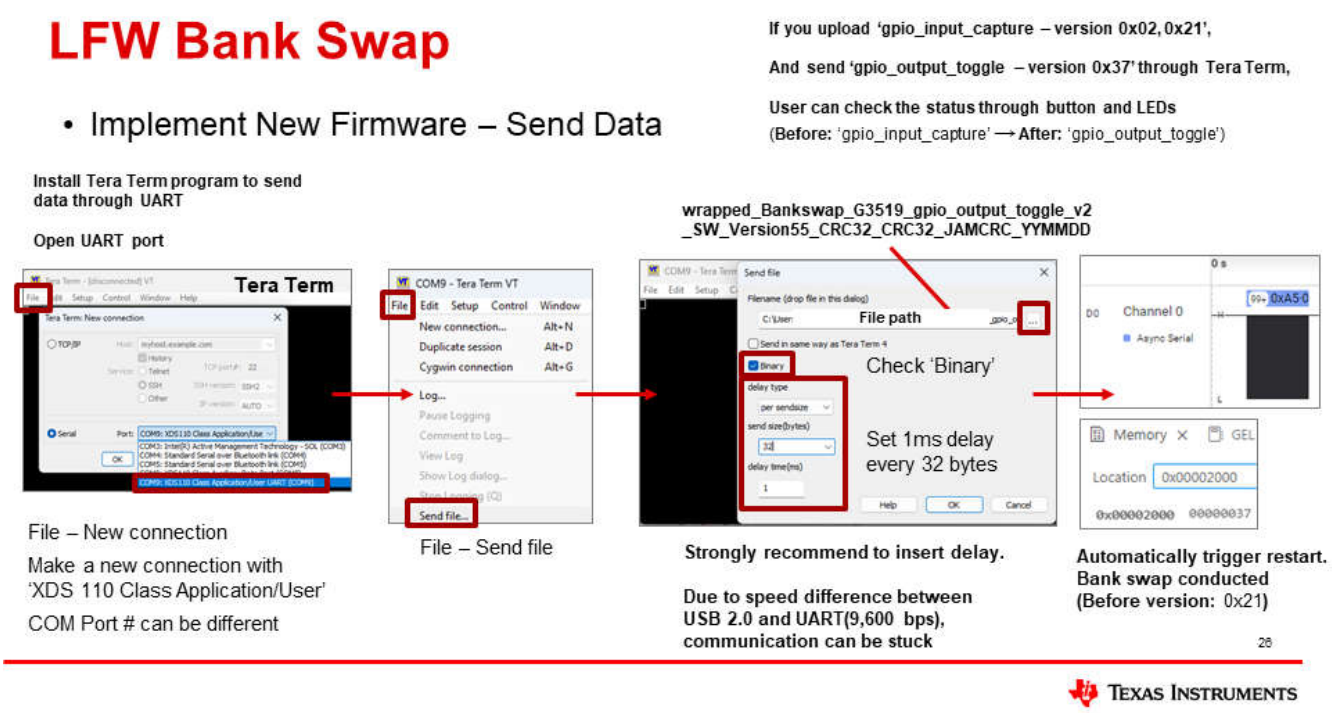

图 2-11. 在电脑上通过 UART 发送新固件

连接 XDS-110，其承担 USB 转 UART 桥接器的功能。使用 Tera Term 将固件数据帧 (.bin) 传输到 MCU。

由于 USB 2.0 和 UART 之间的速度差异，通信过程可能出现偶发停滞。为了解决该问题，TI 建议插入延迟。在该实现中，每 32 个字节增加 1ms 延迟。此问题与 XDS-110 有关，不会对 MCU 造成影响。

传输完成后，应用程序会使用 CRC32 验证固件。如果固件有效，则会将其编程到闪存组 1 中。完成刷写过程后，应用程序会触发 BOOTRST，从而将执行权移交至 CSC。然后，CSC 会检查哪个组包含最新固件。固件版本信息存储在每个应用程序映像的起始位置（组 0：0x0000.2000，组 1：0x0004.2000）。

如果新刷写的固件是最新版本，CSC 会启动组交换，交换组 0 和组 1。已更新固件的验证方法将在下一节中说明。

2.4.2.7 检查更新的固件 (TI CCS IDE)

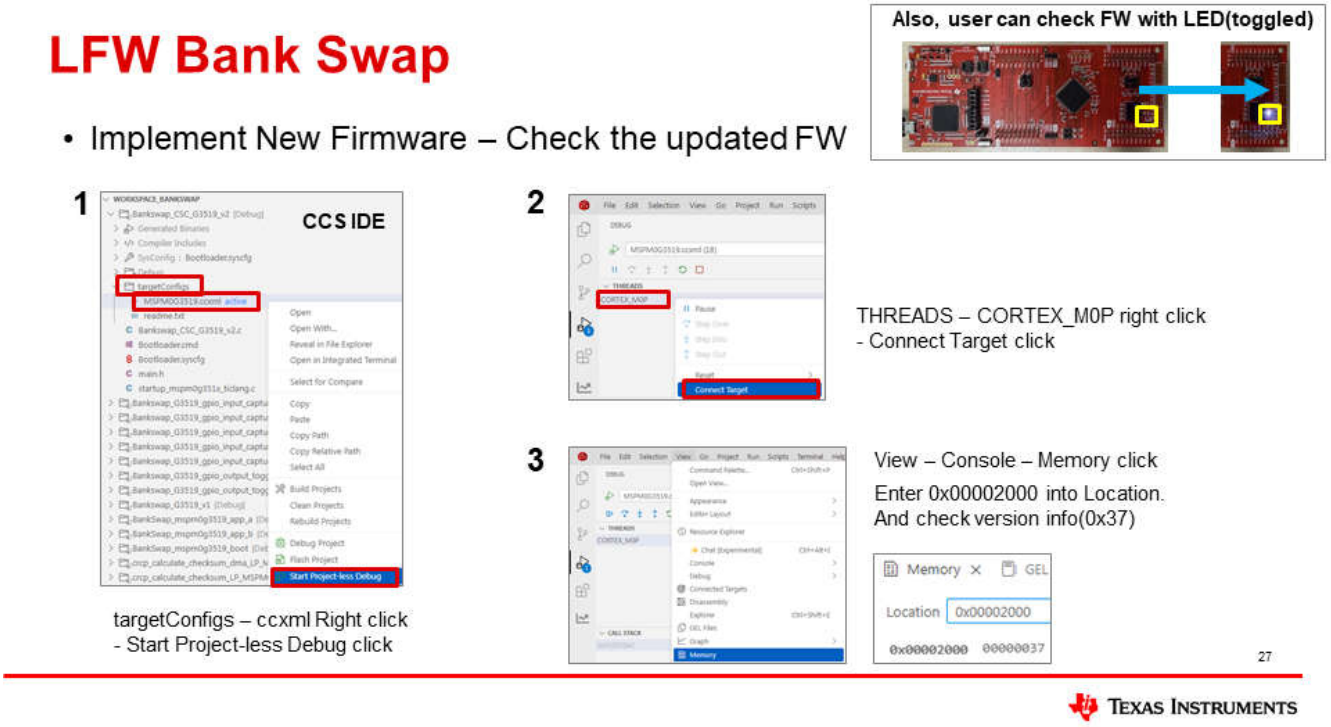

图 2-12. 检查更新的固件

用户可以使用 CCS IDE 验证是否交换了固件。打开调试视图，检查地址 0x0000.2000 处组 0 中的应用程序固件版本。固件版本以 uint32_t 格式存储，并按 LSB 对齐。

3 总结

本文档阐述了组交换和 CSC 的基本原理。以及指导如何通过示例代码和可执行程序 (.exe) 实现实时固件更新。

通过本示例，用户可以在应用程序运行期间进行固件更新，无需使用调试工具 (SWD)。

这是通过组交换进行实时固件更新的基本示例，用户可在此代码基础上，添加安全、稳定性等功能。

4 参考资料

- MSPM0 系列中的闪存多组功能 — 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 系列 80MHz 微控制器技术参考手册（修订版 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+ 文档“矢量表偏移寄存器”— https://developer.arm.com/documentation/dui0662/b/Cortex-M0--Peripherals/System-Control-Block/Vector-Table-Offset-Register

重要通知和免责声明

TI“按原样”提供技术和可靠性数据（包括数据表）、设计资源（包括参考设计）、应用或其他设计建议、网络工具、安全信息和其他资源，不保证没有瑕疵且不做出任何明示或暗示的担保，包括但不限于对适销性、与某特定用途的适用性或不侵犯任何第三方知识产权的暗示担保。

这些资源可供使用 TI 产品进行设计的熟练开发人员使用。您将自行承担以下全部责任：(1) 针对您的应用选择合适的 TI 产品，(2) 设计、验证并测试您的应用，(3) 确保您的应用满足相应标准以及任何其他安全、安保法规或其他要求。

这些资源如有变更，恕不另行通知。TI 授权您仅可将这些资源用于研发本资源所述的 TI 产品的相关应用。严禁以其他方式对这些资源进行复制或展示。您无权使用任何其他 TI 知识产权或任何第三方知识产权。对于因您对这些资源的使用而对 TI 及其代表造成的任何索赔、损害、成本、损失和债务，您将全额赔偿，TI 对此概不负责。

TI 提供的产品受 TI 销售条款)、TI 通用质量指南 或 ti.com 上其他适用条款或 TI 产品随附的其他适用条款的约束。TI 提供这些资源并不会扩展或以其他方式更改 TI 针对 TI 产品发布的适用的担保或担保免责声明。 除非德州仪器 (TI) 明确将某产品指定为定制产品或客户特定产品，否则其产品均为按确定价格收入目录的标准通用器件。

TI 反对并拒绝您可能提出的任何其他或不同的条款。

最后更新日期：2025 年 10 月