# Dual Bank Firmware Upgrade Mechanism

Application Note for 32-bit NuMicro® Family

## Document Information

| | |
|---|---|
| **Abstract** | Introduce how to update firmware reliably with a dual bank Flash architecture and a rollback mechanism supported for user to roll back to the previous firmware if a new firmware works abnormally. |
| **Apply to** | NuMicro® M261 Series. |

## Table of Contents

# 1 Overview

Although products are well tested before being delivered to customers, sometimes, product developers still need to upgrade the product firmware to fix vulnerabilities or improve the compatibility. To minimize the maintenance cost and maximize user convenience, upgrading firmware on user side is necessary. However, upgrading firmware on user side may encounter many unexpected conditions, such as, power lose, incorrect firmware, older version, etc. That will cause product failure and increase more maintenance efforts when firmware update failed. Here, a reliable firmware upgrade mechanism based on the NuMicro® M261 series microcontroller (MCU) is provided to avoid such firmware upgrade failures.

The FMC (Flash Memory Controller) in the M261 series provides APROM Flash memory for users to develop application. APROM is 512Kbytes while its Flash address ranges from 0x0 to 0x7FFFF. The M261 FMC also provides dual bank architecture where APROM is separated into Bank0 and Bank1. Bank0's Flash address range is from 0x0 to 0x3FFFF while Bank1's Flash address range is from 0x40000 to 0x7FFFF. In such dual bank architecture, a firmware upgrade mechanism is provided which uses APROM Bank0 as firmware executing region and APROM Bank1 as storage region for newer version of firmware, as shown in Figure 1-1.

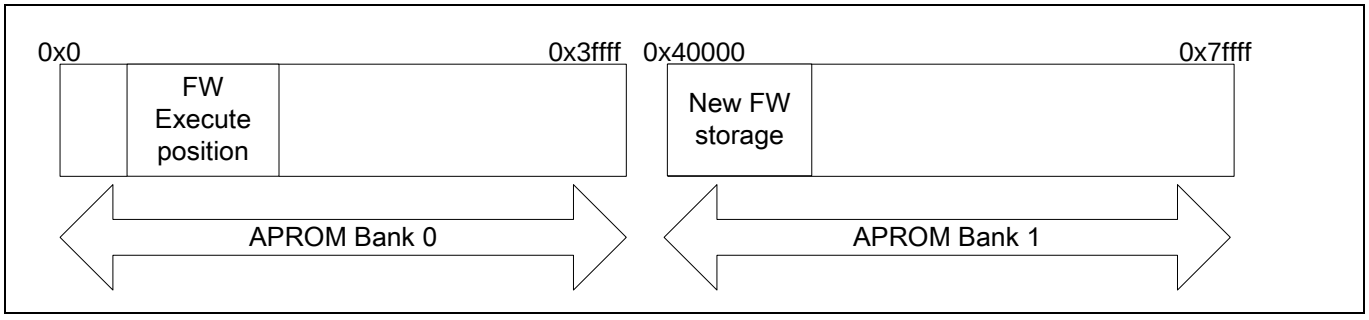


Figure 1-1 APROM Bank0 and Bank1 Usage

The firmware in Bank0 can erase Bank1 and program new firmware to Bank1 without stopping instruction fetching form Bank0. This is so called read-while-write (RWW), which provides the ability for firmware to keep software services while firmware updating. The firmware upgrade mechanism here also provides a rollback process to roll back to the original firmware if a new firmware woks abnormally. Such mechanism ensures the firmware to be safe to update in any circumstance.

# 2 Memory Map for Firmware Upgrade

In the dual bank firmware upgrade mechanism, several regions are arranged in APROM, as shown in Figure 2-1.

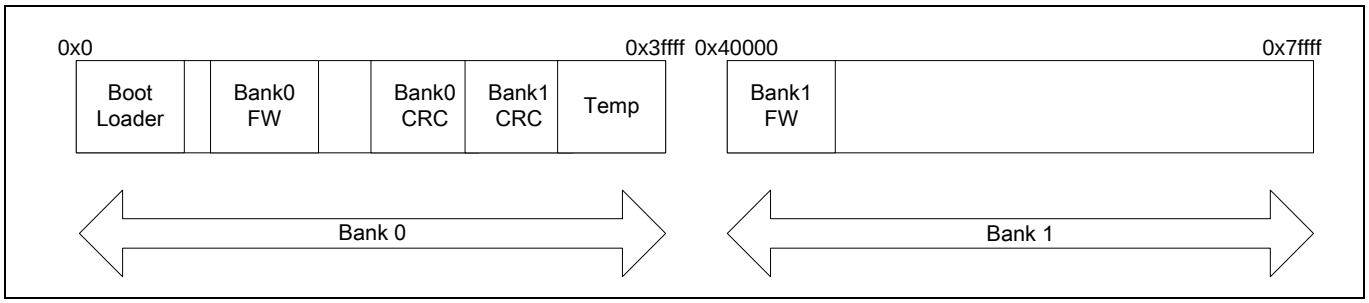


Figure 2-1 Memory Map for Firmware Upgrade

Each region is described as follows:

- Boot loader:
  The boot loader is used to detect new firmware, verify firmware, update firmware and load firmware to be executed. A reliable firmware upgrade mechanism is implemented in boot loader.
- Bank0 firmware region:
  This region is used to store the active firmware that boot loader always loads the firmware in Bank0 to execute. Any firmware required to be executed must be copied to Bank0 first, and then it will be loaded and executed by boot loader.
- Bank0/Bank1 CRC (checksum) region:
  To verify the firmware integrity, both the firmware in Bank0 and Bank1 has their own CRC check region. Each CRC region is 2 Kbytes.
- Temp region:
  To support rollback function, the firmware upgrade is implemented by swapping the old firmware with new firmware. This ensures the old firmware still existed and could be rolled back after updating. By the way, it is also necessary to prevent data lost during swapping. Therefore, a temp region is necessary to back up the swapping data to avoid data lost when the swap process is incomplete due to any circumstance.
- Bank1 firmware region:
  The Bank1 firmware region is used to store the new firmware before it can be activated. In general, when application detected remote new firmware and wants to use it to replace local firmware. The application must download the new firmware to Bank1 region in local first. Then the boot loader will find it and start firmware upgrade procedure at booting time.

# 3 Dual Bank Firmware Upgrade Process

## 3.1 Firmware Upgrade Flow

Figure 3-1 shows the firmware upgrade flow.

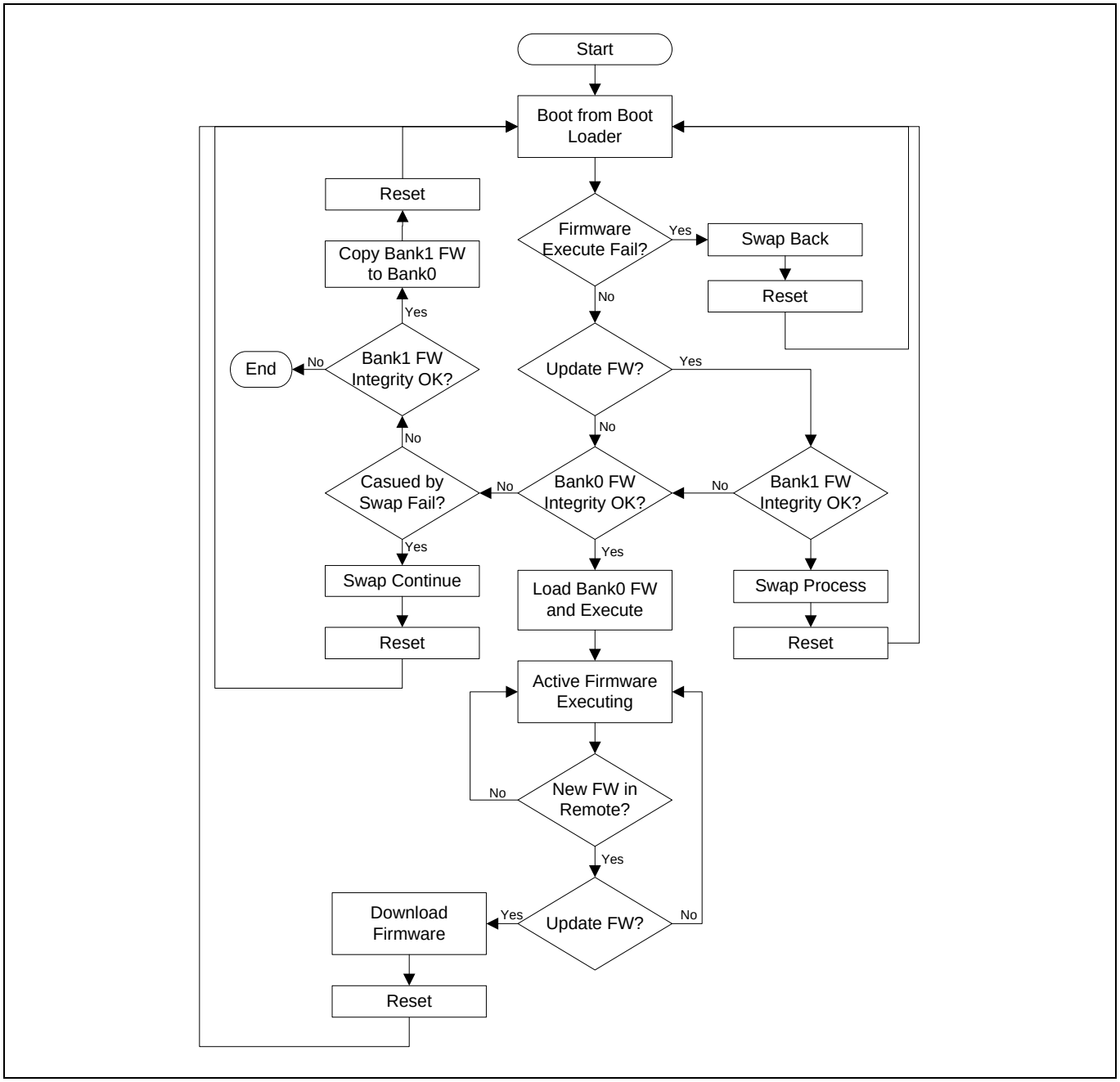


Figure 3-1 Firmware Upgrade Flow

In general case, the system should boot from boot loader. The boot loader will check integrate of Bank0 firmware and load it. Therefore, the Bank0 firmware is called as active firmware. If there is a new firmware to update, the active firmware should start the download process to download the new firmware. For example, if the new firmware is stored in a SD card, the download process will start to read new firmware data from external SD card and program the data to Bank1 region. Figure 3-2 shows the firmware download process.

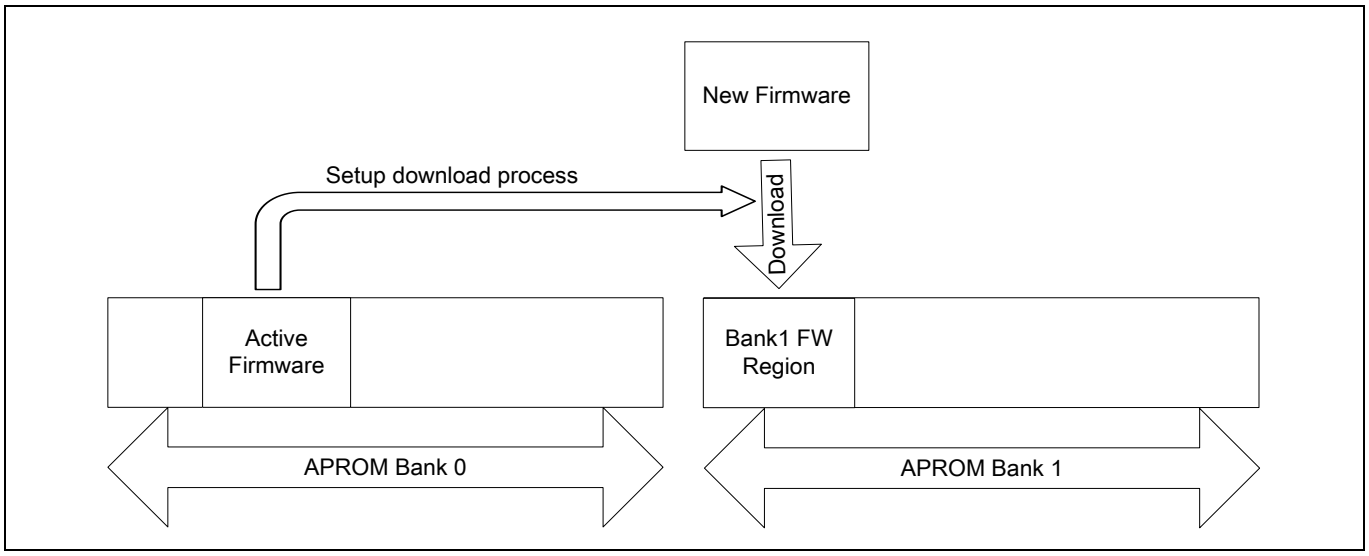


Figure 3-2 Firmware Download Process

After the firmware download process is completed, the new firmware should be located at Bank1 firmware region as shown in Figure 3-3.

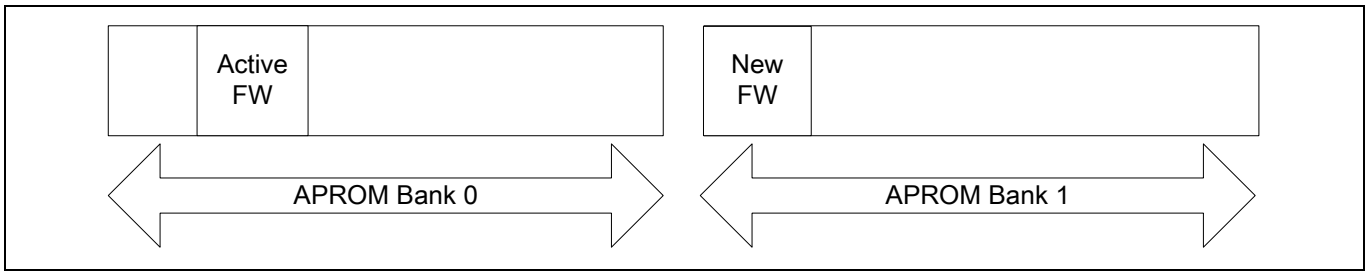


Figure 3-3 Download Process Completed

User needs to reset system to boot loader. Then the boot loader will start the swap process. In the swap process, the new firmware will be swapped from Bank1 to Bank0 page by page while the original firmware will be swapped to Bank1 for backup, as shown in Figure 3-4.

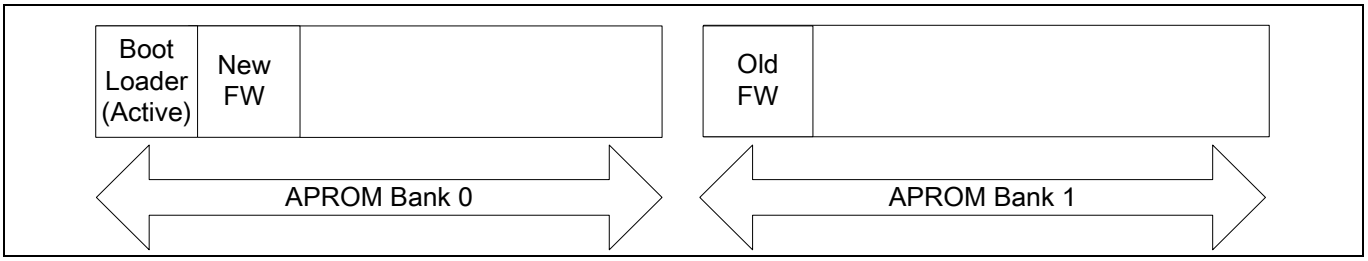


Figure 3-4 Swap Process Completed

Now, the new firmware is located at Bank0 firmware region as active firmware. Because the old firmware is still backed up at Bank1 region, it is possible to start the swap process again to swap the original firmware back to Bank0 when there are problems with new firmware.

A swap continue process is provided to prevent the swap process from being incomplete due to unexpected conditions such as power off, randomly reset, and system crash. If the swap process is incomplete, Bank0 firmware and Bank1 firmware are both corrupted. The swap continue process will analyze the corrupted condition to find the corrupted page and continue to complete the swap process.

## 3.2 Verify Firmware

If there is no firmware in the Bank1 region, the value of Bank1 CRC region should be all 0xFFFFFFFF. After active firmware downloads a remote new firmware to the Bank1 region, the relative CRC value and version number will be written into the Bank1 CRC region.

The format of CRC region is show in Figure 3-5.

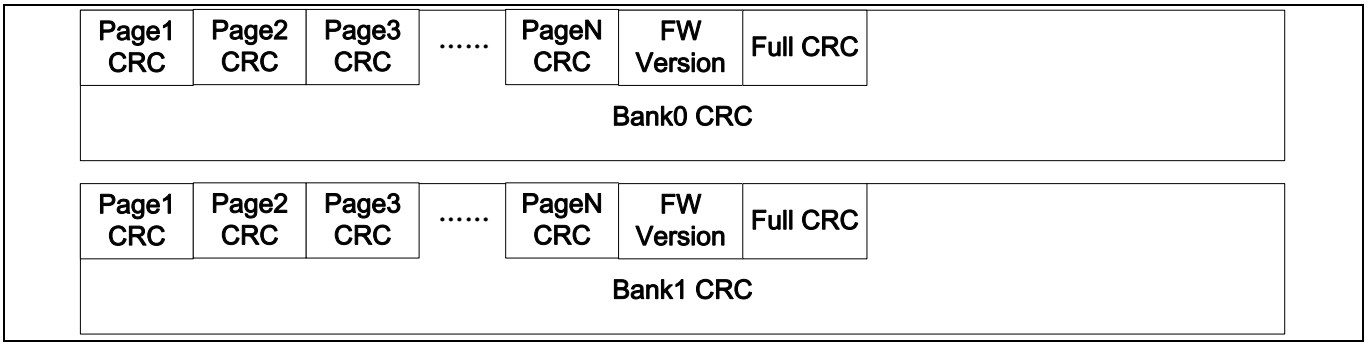


Figure 3-5 Format of CRC Region

The firmware CRC is calculated every 2 Kbytes (one page size in M261), and the CRC value of first 2 Kbytes is written to Page1 CRC. The CRC value of the second 2 Kbytes is written to Page2 CRC and so on. Each CRC value is 32 bits wide. The Page CRC values are used for the swap continue process to avoid firmware upgrade procedure fail.

The firmware version is stored following by the Page CRC values. It could be used to determine whether the firmware in Bank1 region is newer than the firmware in Bank0 region. Finally, the whole firmware CRC are calculated and written to the following firmware version. It could be used to check firmware integrity quickly without comparing all page CRC values.

## 3.3 Execute Active Firmware

If there is a firmware in the Bank0 firmware region and integrity check with Bank0 CRC region is correct, this firmware will be executed and called active firmware. The active firmware execution process is shown in Figure 3-6.

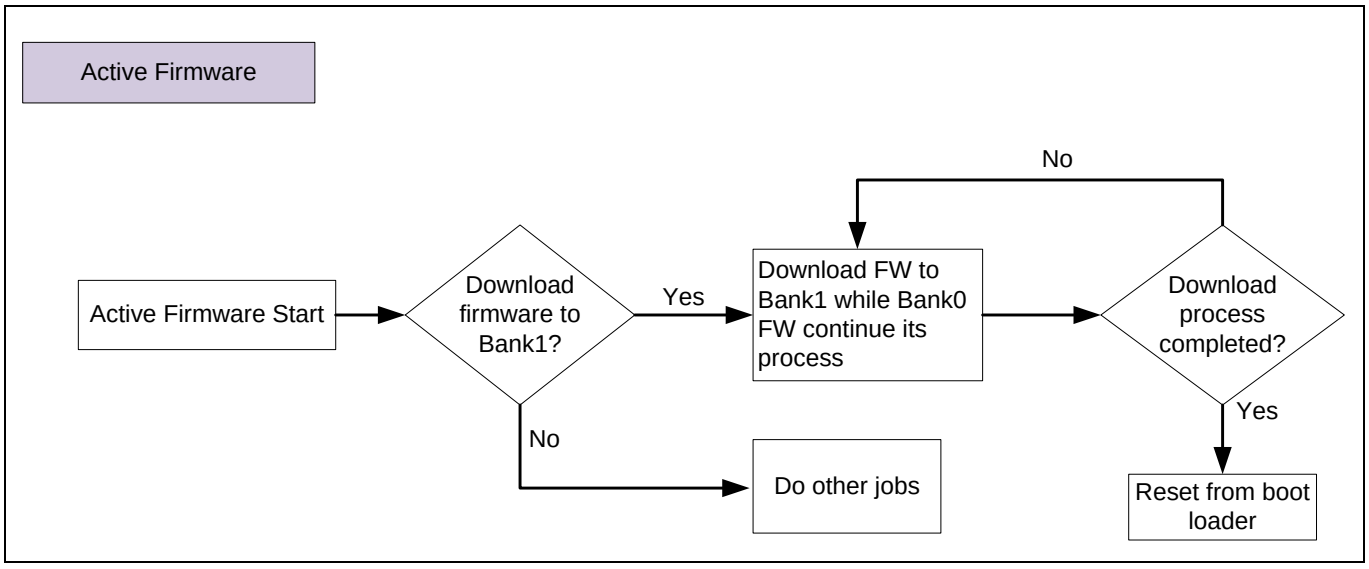


Figure 3-6 Execute Active Firmware Process

The active firmware process is as follows:

1. Active firmware starts.
2. If any firmware exists in PC, SD card, or other storage, user can decide to download the firmware to Bank1 firmware region or not.
3. Execute Bank0 firmware if user decides not to download firmware to Bank1. Otherwise, the download firmware process will start.
4. Download firmware to Bank1 firmware region while active firmware executing in Bank0. Because the M261 series supports Read-While-Write (RWW), the instruction fetching will not be suspended while programming Bank1 firmware.
5. After Bank1 firmware download, user can decide to execute new firmware by swap process to swap the new firmware to Bank0 firmware buffer to execute.

## 3.4 Swap Firmware

To make sure that the system always has workable firmware in MCU, the firmware upgrade is

performed by the swap process. By swapping, the firmware could be swapped to a new firmware or an old firmware.

### 3.4.1 Swap Process

When the boot loader starts, it will check if it is necessary to update firmware. If yes, the boot loader will check Bank1 firmware integrity and compare the Bank1 firmware version with Bank0 firmware. If Bank1 firmware has a newer version (Bank1 firmware version number is higher than Bank0 firmware version number), the boot loader will start a swap process. The swap process is performed page by page with a temp page. Figure 3-7 shows the page swap procedure of swap process.

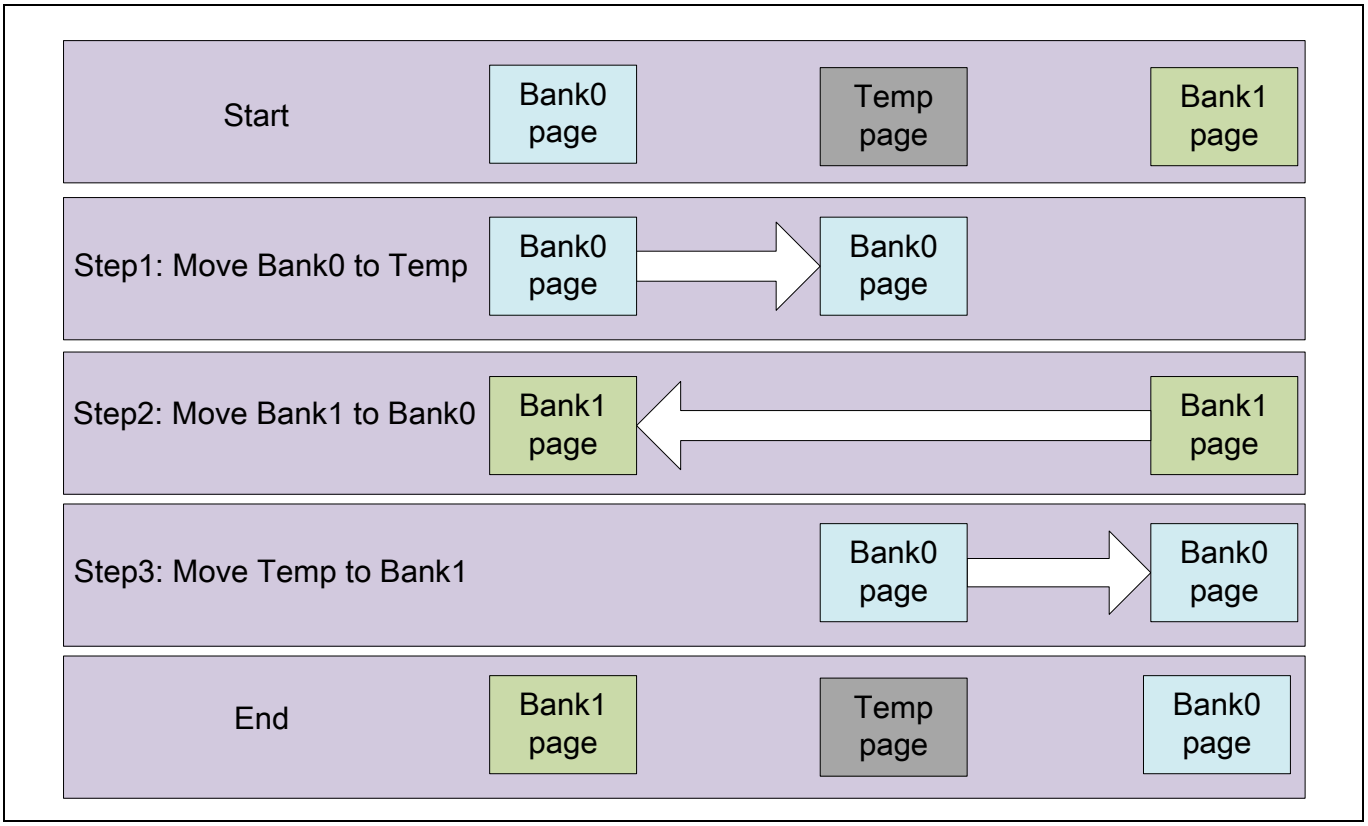


Figure 3-7 Page Swap Procedure

The page swap procedure is as follows:

1. Move Bank0 page to temp page.
2. Move Bank1 page to Bank0 page.
3. Move temp page to Bank1 page.

The whole swap process is shown in Figure 3-8.

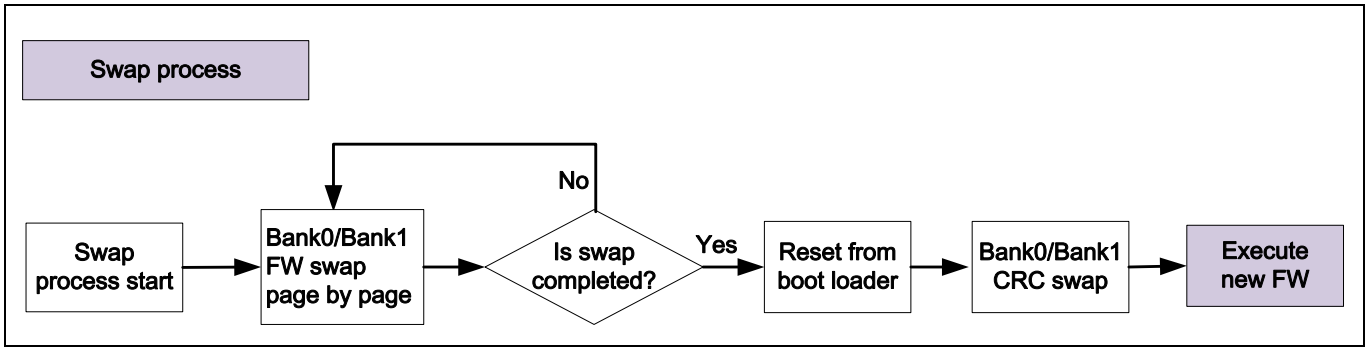


Figure 3-8 Swap Process

The swap process is as follows:

1. Boot loader starts the swap process.
2. Swap the firmware of Bank0 and Bank1 page by page. In this step, the temp region is used to complete the process. Such temp region also ensures that each correct firmware page could be kept even if the swap process is incomplete due to unknown events.
3. After the swap process is completed, the system is reset to restart boot loader.
4. Swap Bank0 and Bank1 CRC values to be synchronous with Bank0/Bank1 firmware.
5. Execute the new firmware in Bank0.

### 3.4.2 Swap Continue Process

Under abnormal conditions, the swap process cannot be completed due to unexpectedly power off, reset, or system crash. When system restarts from boot loader, the boot loader will find the firmware in Bank0 and Bank1 are both corrupted. The corrupted firmware condition shows the incorrect full firmware CRC and partial of page CRC values are correct while the others are incorrect. In this case, the boot loader will start the swap continue process. The swap continue process is shown in Figure 3-9.

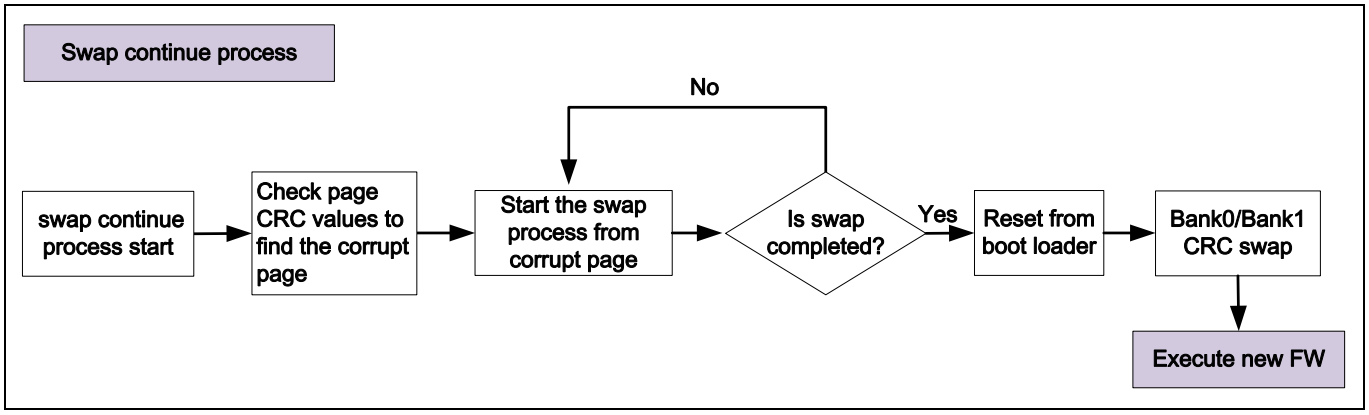


Figure 3-9 Swap Continue Process

The swap continue process is as follows:

1. Boot loader starts the swap continue process.
2. Get Bank0 and Bank1 CRC values to check which page is corrupted in the last incomplete swap process.
3. Start the swap process from corrupted page.
4. After the swap process is completed, reset system to start from boot loader.
5. Swap Bank0 and Bank1 CRC values to synchronous with Bank0/Bank1 firmware.
6. Execute new firmware in Bank0.

### 3.4.3 Swap CRC Region

Once the swap process or swap continue process is completed, the system will be reset to boot loader. The boot loader will find that the Bank0 CRC value is the same as Bank1 firmware and Bank1 CRC value is the same as Bank0 firmware. This is because Bank0/1 firmware is swapped but CRC regions are not swapped yet. Therefore, the boot loader needs to swap the Bank0/1 CRC region to keep the CRC and firmware synchronized.

### 3.4.4 Swap Back Process

To make sure the reliability, the boot loader starts the swap back process if the new firmware execution failed due to self-test fail or watchdog reset fail. The swap back process is shown in Figure 3-10.

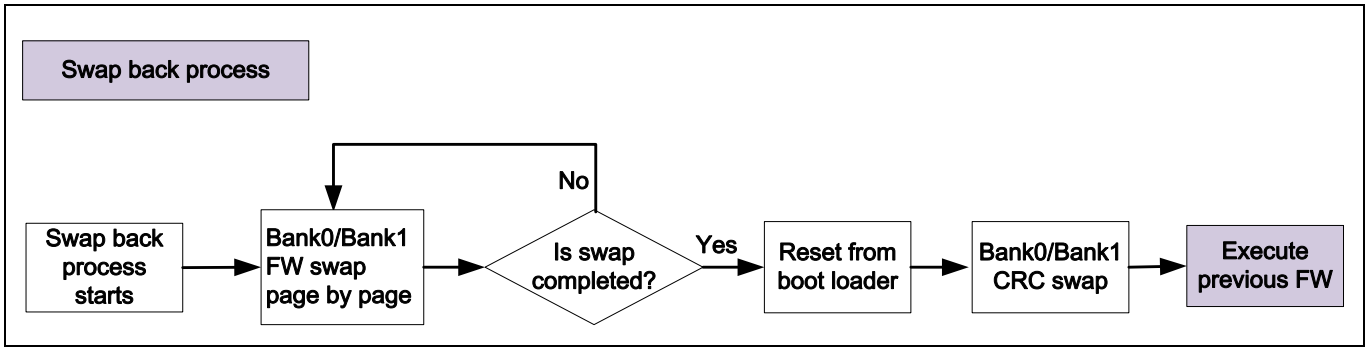


Figure 3-10 Swap Back Process

The swap back process is as follows:

1. Boot loader starts the swap back process.
2. Swap the firmware of Bank0 and Bank1 page by page.
3. Reset system form boot loader.
4. Swap Bank0 and Bank1 CRC values to synchronous with Bank0/Bank1 firmware.
5. Execute previous firmware in Bank0.

## 3.5 Firmware Execution Failure Detection

After the swap process is done, the new firmware is in Bank0 firmware region while the original firmware is in Bank1 firmware region. Then user can reset the system to execute the new firmware in Bank0 firmware region. To prevent the new firmware execution from being failed, user can set the watchdog (WDT) timer with a time-out value and enable WDT reset function. If the new firmware execution failed, the swap back process will start. If new firmware crashed, the system will be reset by watchdog. The boot loader can detect watchdog reset and set the swap back process to roll back to the original firmware. In addition to the watchdog timer, user can use a self-test mechanism to detect whether the new firmware works well or not. The detection result could be a failure flag for boot loader. Figure 3-11 shows fail recovery flow.

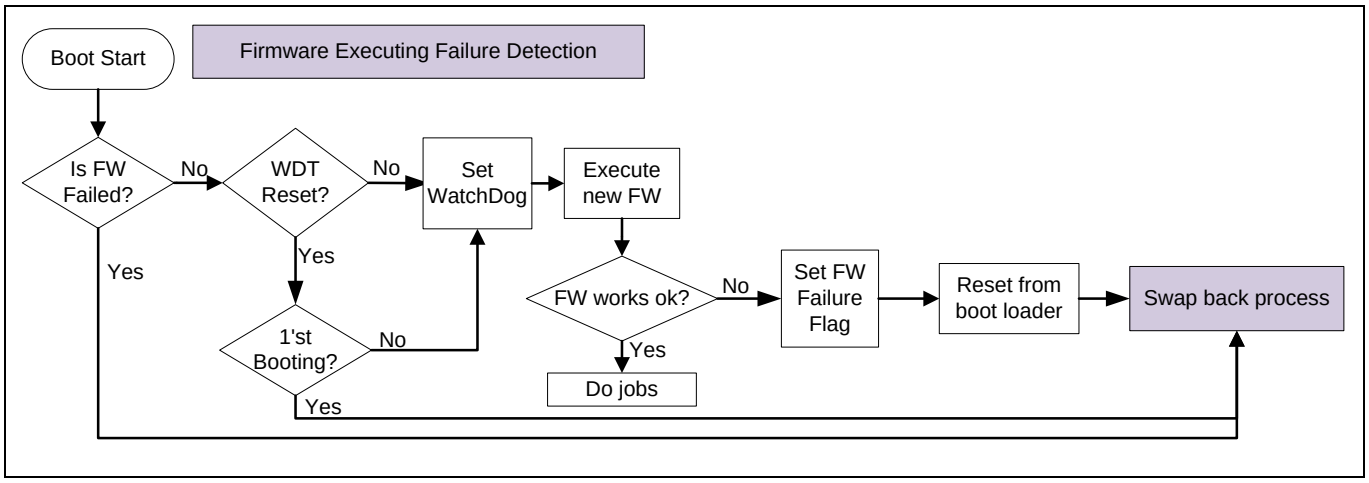


Figure 3-11 Firmware Executing Failure Detection

# 4 Sample Code

User can get sample code of dual bank firmware upgrade mechanism form the M261 BSP folder. The path is:

bsp\SampleCode\StdDriver\FMC_DualBankFwUpdate

FMC_DualBankFwUpdate folder includes 3 sub-folders. The function description of these sub-folders is as follows:

- Common:
  - Definition of the address of each region in APROM。
  - Definition of CRC function.
- FMC_DualBankBoot:
  - System start.
  - Control the executing flow by checking the status of firmware in Bank0/Bank1.
  - Execute the swap process, swap continue process, and swap back process.
  - Watch dog reset control.
- FMC_DualBankFW
  - Execute firmware in Bank0 firmware region.
  - Download new firmware to Bank1 firmware region.

Before system starts, the system environment should be set up first.

1. The boot loader, Bank0/Bank1 firmware region, Bank0/Bank1 CRC region, and temp region should be erased first, as shown in Figure 4-1.

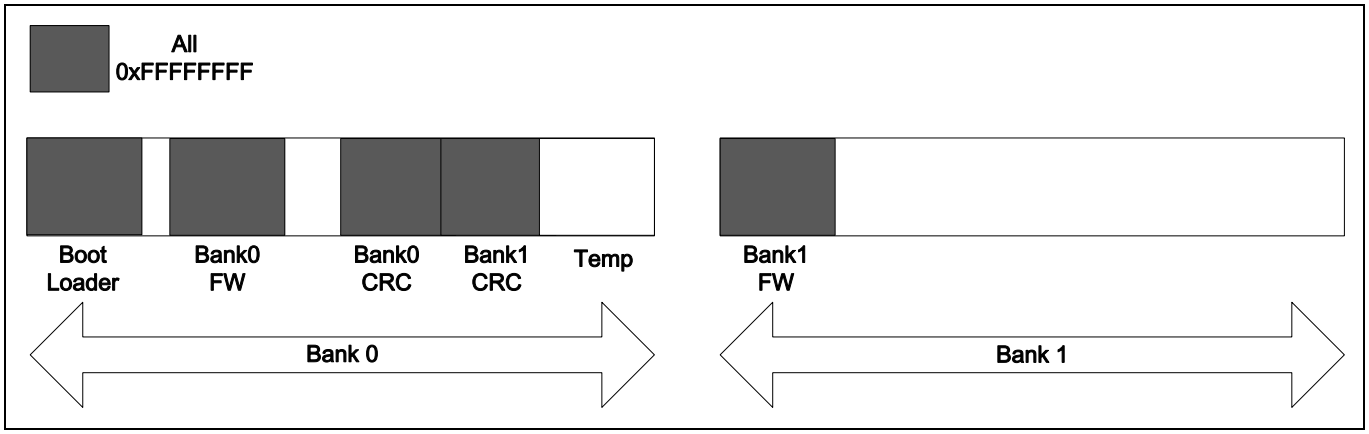


Figure 4-1 Initial Memory Status

If there is no firmware in Bank0/1 firmware region, the value of CRC region should be all 0xFFFFFFFF.

2. Load Boot Loader program into APROM address 0x0 and load active firmware into Bank0 firmware region, as shown in Figure 4-2.

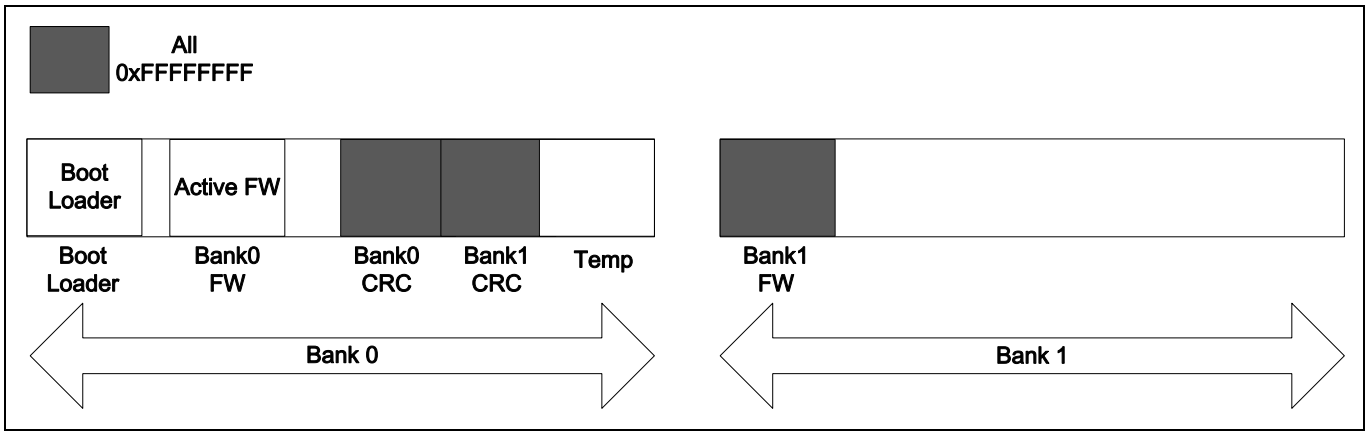


Figure 4-2 Load Boot Loader and Active Firmware

3. Execute active firmware in Bank0 firmware region and calculate the CRC of each Bank0 firmware page and CRC of full Bank0 firmware.
4. Write CRC value and version number of active firmware into Bank0 CRC region, as shown in Figure 4-3.

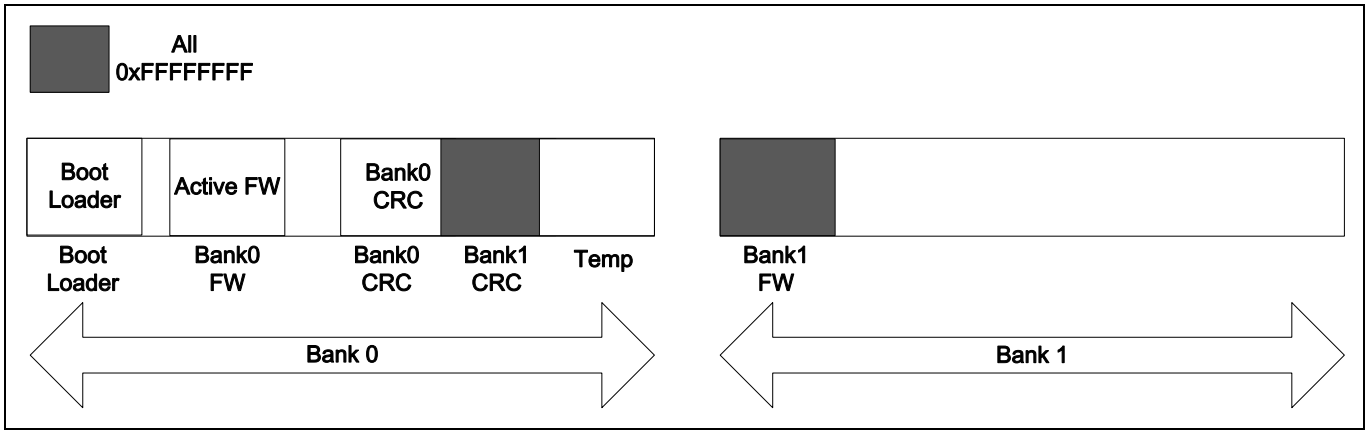


Figure 4-3 Write Bank0 Firmware CRC value

5. If there is any other version of firmware in PC, SD card, or other storage, the CRC of each firmware page and full CRC of the firmware should also be calculated after the firmware being downloaded to Bank1 firmware region.
6. Writes CRC value and version number of the firmware into Bank1 CRC region, as shown in Figure 4-4.

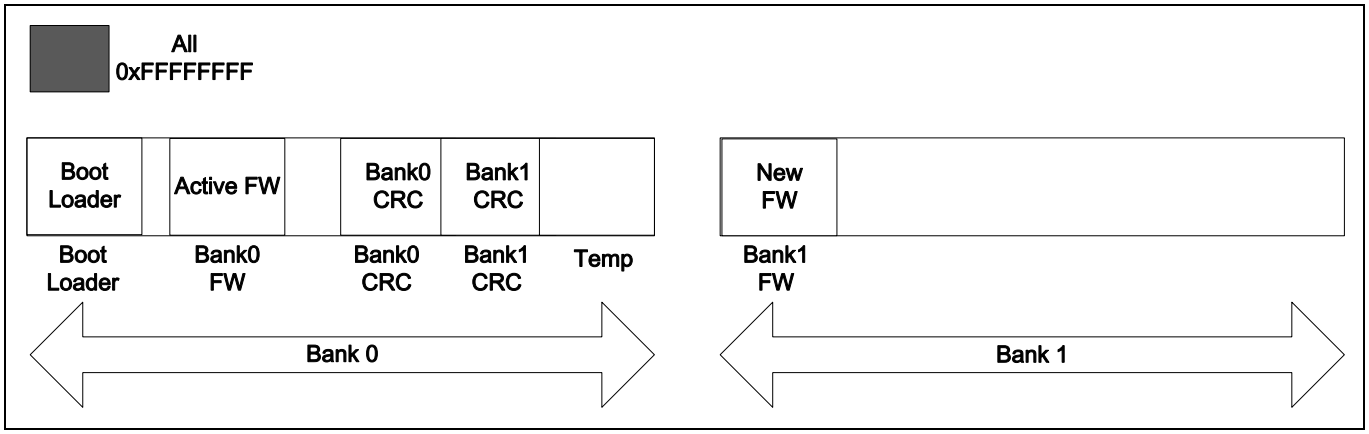


Figure 4-4 Write Bank1 CRC value

Definition for each region in APROM: (in common)

```
#define BOOT_BASE 0x0

#define TMP_PAGE_SIZE  0x800
/* CRC of each PAGE for bank0 */
#define BANK0_PAGE_CRC_BASE  0x3E800
/* CRC of each PAGE for bank1 */
#define BANK1_PAGE_CRC_BASE  0x3F000

/* Bank0 firmware related definition */
#define BANK0_FW_BASE  0x10000
#define BANK0_FW_SIZE  TMP_PAGE_SIZE*8
#define BANK0_FW_VER_BASE  (BANK0_PAGE_CRC_BASE + (BANK0_FW_SIZE/TMP_PAGE_SIZE)*4)
#define BANK0_FW_CRC_BASE  (BANK0_PAGE_CRC_BASE + (BANK0_FW_SIZE/TMP_PAGE_SIZE+1)*4)

/* Bank1 firmware related definition */
#define BANK1_FW_BASE  0x40000
#define BANK1_FW_SIZE  TMP_PAGE_SIZE*8
#define BANK1_FW_VER_BASE  (BANK1_PAGE_CRC_BASE + (BANK1_FW_SIZE/TMP_PAGE_SIZE)*4)
#define BANK1_FW_CRC_BASE  (BANK1_PAGE_CRC_BASE + (BANK1_FW_SIZE/TMP_PAGE_SIZE+1)*4)

/* For FW Swap tmp buffer, size is 1 page (2 Kbytes) */
#define TMP_PAGE_BASE  0x3F800
```

CRC calculating function: (in common)

```
 /**
  * @brief      Checksum calculation function
```

```
  * @param[in]  start       check sum calculation start address
  * @param[in]  len         check sum calculation block size
  * @retval     sum         check sum value
  */
uint32_t  func_crc32(uint32_t start, uint32_t len)
{
   uint32_t  idx, data32 = 0UL;
   int      i;

   /* WDTAT_RVS, CHECKSUM_RVS, CHECKSUM_COM */
   for(idx = 0; idx < len; idx += 4)
   {
      data32 += *(uint32_t *)(start + idx);
   }
            data32 = 0xFFFFFFFF - data32 + 1UL;

   return data32;
}
```

Reset and start from Bank0 firmware buffer: (in FMC_DualBankBoot)

```
/* Set remmaping address */
FMC->ISPADDR = BANK0_FW_BASE;
/* Set VECMAP */
FMC->ISPCMD = 0x2E;
FMC->ISPTRG = 1;
while(FMC->ISPTRG);
/* CPU Reset */
SYS->IPRST0 |= SYS_IPRST0_CPURST_Msk;
```

Execute the current firmware in Bank0 and writes CRC, version information into Bank0 CRC buffer: (in FMC_DualBankFW)

```
/* Erase Bank0 CRC Buffer */
FMC_Erase(BANK0_PAGE_CRC_BASE);
/* Calculate Bank0 CRC */
sum = func_crc32(BANK0_FW_BASE, BANK0_FW_SIZE);

/* Write Bank0 CRC values*/
if(FMC_Read(BANK0_FW_CRC_BASE)==0xFFFFFFFF)
      FMC_Write(BANK0_FW_CRC_BASE, sum);

/* Write Bank0 version value*/
```

```
uf(FMC_Read(BANK0_FW_VER_BASE)==0xFFFFFFFF)
        FMC_Write(BANK0_FW_VER_BASE, version number); /* version number set by user*/

/* Write CRC for each Page */
for(i = 0; i < BANK0_FW_SIZE/TMP_PAGE_SIZE; i++)
{
        sum = func_crc32(BANK0_FW_BASE + i*TMP_PAGE_SIZE, TMP_PAGE_SIZE );
        if(FMC_Read(BANK0_PAGE_CRC_BASE + i*4) == 0xFFFFFFFF)
                FMC_Write(BANK0_PAGE_CRC_BASE + i*4, sum);
}
```

System initial settings：(in FMC_DualBankBoot)

```
void SYS_Init(void)
{
  /* Enable HIRC clock */
  CLK_EnableXtalRC(CLK_PWRCTL_HIRCEN_Msk);

  /* Wait for HIRC clock ready */
  CLK_WaitClockReady(CLK_STATUS_HIRCSTB_Msk);

  /* Select HCLK clock source as HIRC and HCLK source divider as 1 */
  CLK_SetHCLK(CLK_CLKSEL0_HCLKSEL_HIRC, CLK_CLKDIV0_HCLK(1));

  /* Enable HXT clock */
  CLK_EnableXtalRC(CLK_PWRCTL_HXTEN_Msk);

  /* Wait for HXT clock ready */
  CLK_WaitClockReady(CLK_STATUS_HXTSTB_Msk);

  /* Enable PLL */
  CLK->PLLCTL = CLK_PLLCTL_128MHz_HIRC;

  /* Waiting for PLL stable */
  CLK_WaitClockReady(CLK_STATUS_PLLSTB_Msk);

  /* Select HCLK clock source as PLL and HCLK source divider as 2 */
  CLK_SetHCLK(CLK_CLKSEL0_HCLKSEL_PLL, CLK_CLKDIV0_HCLK(2));

  /* Enable UART module clock */
  CLK_EnableModuleClock(UART0_MODULE);
```

```
    /* Select UART module clock source as HXT and UART module clock divider as 1 */
    CLK_SetModuleClock(UART0_MODULE, CLK_CLKSEL1_UART0SEL_HXT, CLK_CLKDIV0_UART0(1));

    /* Enable WDT module clock */
    CLK_EnableModuleClock(WDT_MODULE);
    CLK_SetModuleClock(WDT_MODULE, CLK_CLKSEL1_WDTSEL_LIRC, 0);

    /* Set multi-function pins for UART0 RXD and TXD */
    SYS->GPB_MFPH = (SYS->GPB_MFPH & (~(UART0_RXD_PB12_Msk | UART0_TXD_PB13_Msk))) |
UART0_RXD_PB12 | UART0_TXD_PB13;
}
```

Get Bank0/Bank1 CRC, firmware version and calculate Bank0/Bank1 CRC value. System will compare the obtained values and the calculated values to check integrity and correctness of firmware in Bank0 and Bank1. (in FMC_DualBankBoot)

```
/* Get Bank0 firmware version, crc values*/
ver0 = FMC_Read(BANK0_FW_VER_BASE);
crc0 = FMC_Read(BANK0_FW_CRC_BASE);

/* Get Bank1 firmware version, crc values*/
ver1 = FMC_Read(BANK1_FW_VER_BASE);
crc1 = FMC_Read(BANK1_FW_CRC_BASE);

/* Calculate Bank0 CRC */
crcGet0 = func_crc32(BANK0_FW_BASE, BANK0_FW_SIZE - 8);
/* Calculate Bank1 CRC */
crcGet1 = func_crc32(BANK1_FW_BASE, BANK1_FW_SIZE - 8);

/* Get Bank0 CRC for each page */
for(i = 0; i < BANK0_FW_SIZE/TMP_PAGE_SIZE; i++)
{
      bank0PageSumInFlash[i] = FMC_Read(BANK0_PAGE_CRC_BASE + i*4);
}
/* Get Bank1 CRC for each page */
for(i = 0; i < BANK1_FW_SIZE/TMP_PAGE_SIZE; i++)
{
      bank1PageSumInFlash[i] = FMC_Read(BANK1_PAGE_CRC_BASE + i*4);
}
```

Execute the current firmware in Bank0 and download new firmware to Bank1.

Global variable definition(in FMC_DualBankFW)

```
/* Dual bank background program state */
enum
{
   DB_STATE_START,              /* Start background dual bank program       */
   DB_STATE_ERASE,              /* Executing ISP page erase                 */
   DB_STATE_PROGRAM,            /* Executing ISP write                      */
   DB_STATE_DONE,               /* All queued ISP operations finished. Idle */
   DB_STATE_FAIL                /* ISP command failed or verify error       */
};

static volatile int  db_state = DB_STATE_DONE;  /* dual bank background program state */
static volatile uint32_t  db_length;      /* dual bank program remaining length */
static volatile uint32_t  db_addr;       /* dual bank program current flash address  */
```

Download new firmware flow in main function: (in FMC_DualBankFW).

```
while(1)
{
      db_state = DB_STATE_DONE;     /* dual bank program state idle */
      db_addr = BANK1_FW_BASE;     /* Dual bank background program address  */
      db_length = BANK1_FW_SIZE;    /* Dual bank background length  */

      db_state = DB_STATE_START;    /* Start background dual bank program */

     /*User can do jobs here*/
      for(loop = 0; loop < CRC32_LOOP_CNT; loop++)
      {
           /* Calculate 64KB CRC32 value, just to consume CPU time */
           func_crc32(0x0, 0x10000);
      }
      while(db_state != DB_STATE_DONE) ;
      sum = func_crc32(BANK1_FW_BASE, BANK1_FW_SIZE);
}
```

Program the new firmware word by word into Bank1 firmware buffer in interrupt function: (in FMC_DualBankFW).

```
void SysTick_Handler(void)
{
  /* Background program is in idle state */
   if(db_state == DB_STATE_DONE)
  {
     return;
  }
```

```
 /* Background program done? */
if(db_length == 0)
{
  /* enter idle state */
   db_state = DB_STATE_DONE;
   return;
}
/* ISP is busy, postpone to next called */
if(FMC->MPSTS & FMC_MPSTS_MPBUSY_Msk)
   return;

/* Dual-bank background program */
switch(db_state)
{
   case DB_STATE_START:
      if(db_addr & ~FMC_PAGE_ADDR_MASK)
      {
         printf("Warning - dual bank start address is not page aligned!\n");
         db_state = DB_STATE_FAIL;
         break;
      }
      if(db_length & ~FMC_PAGE_ADDR_MASK)
      {
         printf("Warning - dual bank length is not page aligned!\n");
         db_state = DB_STATE_FAIL;
         break;
      }
      /* Next state is to erase flash */
      db_state = DB_STATE_ERASE
      break;

   case DB_STATE_ERASE:
      FMC->ISPCMD = FMC_ISPCMD_PAGE_ERASE; /* ISP page erase command            */
      FMC->ISPADDR = db_addr;              /* page address                      */
      FMC->ISPTRG = FMC_ISPTRG_ISPGO_Msk;  /* trigger ISP page erase and no wait */
       printf("Erase [0x%8x].\n", db_addr);
       db_addr += TMP_PAGE_SIZE;

       if(db_addr >= (BANK1_FW_BASE + BANK1_FW_SIZE) )
       {
            printf("\nErase Done db_addr[0x%8x]\n", db_addr);
```

```
                db_addr = BANK1_FW_BASE;
                db_state = DB_STATE_PROGRAM; /* Next state is to program flash   */
            }
            break;

        case DB_STATE_PROGRAM:
            FMC->ISPCMD = FMC_ISPCMD_PROGRAM;   /* ISP word program command   */
            FMC->ISPADDR = db_addr;             /* word program address       */
            FMC->ISPDAT = input source;         /* 32-bits data to be programmed, user decide
the downloading source*/
            /* trigger ISP program and no wait  */
            FMC->ISPTRG = FMC_ISPTRG_ISPGO_Msk;
            /* advance to next word */
            db_addr += 4;
            db_length -= 4;

            if(db_addr % 0x800 == 0)
                printf("[0x%8x] page programing done\n", db_addr);

            if(db_length == 0)
                printf("\nAll programing done!!!!!db_addr[0x%8x]\n", db_addr);

            break;

        default:
            printf("Unknown db_state state!\n");
            while(1);
    }
}
```

After getting Bank0/Bank1 firmware version number and CRC information, check the CRC value and compare firmware version to determine whether to start the swap process. (in FMC_DualBankBoot)

```
/* Bank0 CRC and Bank1 CRC are correct */
if( (crcGet0 == crc0) && (crcGet1 == crc1) )
{
    if(ver0 < ver1)
    {
        printf("Bank1 FW is new, do bank0/bank1 swap \n");
        Bank_Swap();
        printf("Swap done!!! \n");
        /* Set remmaping address */
```

```
            FMC->ISPADDR = BOOT_BASE;
            /* Set VECMAP */
            FMC->ISPCMD = 0x2E;
            FMC->ISPTRG = 1;
            while(FMC->ISPTRG);
            /* CPU Reset to Boot Loader to update CRC*/
            SYS->IPRST0 |= SYS_IPRST0_CPURST_Msk;
        }
        else
        {
            printf(" Execute Bank0 FW\n");
            /* Set remmaping address */
            FMC->ISPADDR = BANK0_FW_BASE;
            /* Set VECMAP */
            FMC->ISPCMD = 0x2E;
            FMC->ISPTRG = 1;
            while(FMC->ISPTRG);
            /* CPU Reset */
            SYS->IPRST0 |= SYS_IPRST0_CPURST_Msk;
        }
}
```

Swap Function: (in FMC_DualBankBoot)

```
static void Bank_Swap(void)
{
    uint32_t     i, j;

    FMC_Erase(TMP_PAGE_BASE);
    for(i = 0; i < BANK0_FW_SIZE; i+= TMP_PAGE_SIZE)
    {
        Dump_Swap_Info(i);
      for(j = 0; j < TMP_PAGE_SIZE; j +=4)
      {
        /*Bank0 --> tmp*/
        FMC_Write(TMP_PAGE_BASE + j, FMC_Read(BANK0_FW_BASE + i + j));
      }
      Dump_Swap_Info(i);
      FMC_Erase(BANK0_FW_BASE + i);
      Dump_Swap_Info(i);

      for(j = 0; j < TMP_PAGE_SIZE; j +=4)
```

```
        {
            /* Bank1 --> bank0 */
            FMC_Write(BANK0_FW_BASE + i + j, FMC_Read(BANK1_FW_BASE + i + j));                    }
        Dump_Swap_Info(i);
        FMC_Erase(BANK1_FW_BASE + i);
        Dump_Swap_Info(i);

        for(j = 0; j < TMP_PAGE_SIZE; j +=4)
        {
            /* Tmp --> bank1 */
            FMC_Write(BANK1_FW_BASE + i + j, FMC_Read(TMP_PAGE_BASE + j));
        }
        Dump_Swap_Info(i);
        FMC_Erase(TMP_PAGE_BASE);
        Dump_Swap_Info(i);
    }
}
```

Swap Continued Function: (in FMC_DualBankBoot)

```
static void BankSwapContinue(void)
{
    uint32_t    u32i, u32j, u32StartPage = 0;
    uint32_t    u32TmpSum = 0;

    /* Calculate current bank0, bank1 CRC */
    for(u32i = 0; u32i < BANK0_FW_SIZE / TMP_PAGE_SIZE; u32i++)
    {
        g_au32Bank0PageSum[u32i] = func_crc32(BANK0_FW_BASE + u32i * TMP_PAGE_SIZE,
TMP_PAGE_SIZE);
        g_au32Bank1PageSum[u32i] = func_crc32(BANK1_FW_BASE + u32i * TMP_PAGE_SIZE,
TMP_PAGE_SIZE);
    }

    /* calculate tmp CRC */
    u32TmpSum = func_crc32(TMP_PAGE_BASE, TMP_PAGE_SIZE);

    for(u32i = 0; u32i < BANK0_FW_SIZE / TMP_PAGE_SIZE; u32i++)
    {

        if((g_au32Bank0PageSum[u32i] == g_au32Bank0PageSumInFlash[u32i])
                && (g_au32Bank1PageSum[u32i] == g_au32Bank1PageSumInFlash[u32i]))
        {
```

```
        /* Re-start from step 1 */
        u32StartPage = u32i;
        break;
    }
    else if((u32TmpSum == g_au32Bank0PageSumInFlash[u32i])
        && (g_au32Bank1PageSum[u32i] == g_au32Bank1PageSumInFlash[u32i])
        && (g_au32Bank0PageSum[u32i] != g_au32Bank0PageSumInFlash[u32i]))
    {
        /* Re-start from step 2 */
        FMC_Erase(BANK0_FW_BASE + u32i * TMP_PAGE_SIZE);
        DumpSwapInfo(u32i * TMP_PAGE_SIZE);

        /* Swap step2: move from Bank1 to Bank0 */
        for(u32j = 0; u32j < TMP_PAGE_SIZE; u32j += 4)
        {
            FMC_Write(BANK0_FW_BASE + u32i * TMP_PAGE_SIZE + u32j, FMC_Read(BANK1_FW_BASE +
u32i * TMP_PAGE_SIZE + u32j));
        }
        DumpSwapInfo(u32i * TMP_PAGE_SIZE);
        FMC_Erase(BANK1_FW_BASE + u32i * TMP_PAGE_SIZE);
        DumpSwapInfo(u32i * TMP_PAGE_SIZE);

        for(u32j = 0; u32j < TMP_PAGE_SIZE; u32j += 4)
        {
            FMC_Write(BANK1_FW_BASE + u32i * TMP_PAGE_SIZE + u32j, FMC_Read(TMP_PAGE_BASE +
u32j));
        }
        DumpSwapInfo(u32i * TMP_PAGE_SIZE);

        u32StartPage = u32i + 1;
        break;
    }
    else if((u32TmpSum == g_au32Bank0PageSumInFlash[u32i])
        && (g_au32Bank0PageSum[u32i] == g_au32Bank1PageSumInFlash[u32i])
        && (g_au32Bank1PageSum[u32i] != g_au32Bank1PageSumInFlash[u32i]))
    {
        /* Re-start from step 3 */
        FMC_Erase(BANK1_FW_BASE + u32i * TMP_PAGE_SIZE);
        DumpSwapInfo(u32i * TMP_PAGE_SIZE);

        /* Swap step3: move from tmp to Bank1 */
```

```
            for(u32j = 0; u32j < TMP_PAGE_SIZE; u32j += 4)
            {
                FMC_Write(BANK1_FW_BASE + u32i * TMP_PAGE_SIZE + u32j, FMC_Read(TMP_PAGE_BASE + u32j));
            }
            DumpSwapInfo(u32i * TMP_PAGE_SIZE);

            u32StartPage = u32i + 1;
            break;
        }
        else if((g_au32Bank0PageSum[u32i] == g_au32Bank1PageSumInFlash[u32i])
                && (g_au32Bank1PageSum[u32i] == g_au32Bank0PageSumInFlash[u32i]))
        {
            printf("Page[%d] has been completed!!!\n", u32i);
            u32StartPage = u32i + 1;
        }
        else
        {
            printf("Other condition!!!  page [%d]\n", u32i);
        }
    }
    /* Start the remained swap process from the start page */
    if(u32StartPage < BANK0_FW_SIZE / TMP_PAGE_SIZE)
    {
        FMC_Erase(TMP_PAGE_BASE);
        for(u32i = u32StartPage * TMP_PAGE_SIZE; u32i < BANK0_FW_SIZE; u32i += TMP_PAGE_SIZE)
        {

            DumpSwapInfo(u32i);
            /* Bank0 → temp */
            for(u32j = 0; u32j < TMP_PAGE_SIZE; u32j += 4)
            {
                FMC_Write(TMP_PAGE_BASE + u32j, FMC_Read(BANK0_FW_BASE + u32i + u32j));
            }
            DumpSwapInfo(u32i);
            FMC_Erase(BANK0_FW_BASE + u32i);
            DumpSwapInfo(u32i);

            /* Bank1 → bank0 */
            for(u32j = 0; u32j < TMP_PAGE_SIZE; u32j += 4)
            {
```

```
            FMC_Write(BANK0_FW_BASE + u32i + u32j, FMC_Read(BANK1_FW_BASE + u32i + u32j));
        }
        DumpSwapInfo(u32i);
        FMC_Erase(BANK1_FW_BASE + u32i);
        DumpSwapInfo(u32i);

        /* temp → bank1 */
        for(u32j = 0; u32j < TMP_PAGE_SIZE; u32j += 4)
        {
            FMC_Write(BANK1_FW_BASE + u32i + u32j, FMC_Read(TMP_PAGE_BASE + u32j));
        }
        DumpSwapInfo(u32i);
        FMC_Erase(TMP_PAGE_BASE);
        DumpSwapInfo(u32i);

    }
  }
}
```

After the swap process is done, system restarts from boot loader. When system starts, it updates CRC information for Bank0/Bank1CRC buffer and restarts from Bank0 firmware buffer to execute the new firmware. (in FMC_DualBankBoot)

```
FMC_Erase(TMP_PAGE_BASE);
/*Bank0 CRC → temp*/
for (i = 0; i <(BANK0_FW_SIZE/TMP_PAGE_SIZE+2); i++){
      FMC_Write(TMP_PAGE_BASE + i*4, FMC_Read(BANK0_PAGE_CRC_BASE + i*4));
}
FMC_Erase(BANK0_PAGE_CRC_BASE);
/* Bank1 CRC → Bank0 CRC*/
for (i = 0; i <(BANK0_FW_SIZE/TMP_PAGE_SIZE+2); i++){
      FMC_Write(BANK0_PAGE_CRC_BASE + i*4,   FMC_Read(BANK1_PAGE_CRC_BASE + i*4));
}
FMC_Erase(BANK1_PAGE_CRC_BASE);
/* Temp → Bank1 CRC*/
for (i = 0; i <(BANK0_FW_SIZE/TMP_PAGE_SIZE+2); i++){
      FMC_Write(BANK1_PAGE_CRC_BASE + i*4,   FMC_Read(TMP_PAGE_BASE + i*4));
}
FMC_Erase(TMP_PAGE_BASE);

FMC->ISPADDR = BANK0_FW_BASE;
/* Set VECMAP */
```

```
FMC->ISPCMD = 0x2E;
FMC->ISPTRG = 1;
while(FMC->ISPTRG);
/* CPU Reset */
SYS->IPRST0 |= SYS_IPRST0_CPURST_Msk;
```

When system hangs up in the new firmware process, it will be reset by watchdog and reset to start from boot loader and swap back to the previous firmware to execute. (in FMC_DualBankBoot)

```
if(WDT_GET_RESET_FLAG() == 1)
{
    WDT_CLEAR_RESET_FLAG();
    /* System reset by WDT time-out event */

    /* swap page CRC data */
    FMC_Erase(TMP_PAGE_BASE);
    for (i = 0; i <(BANK0_FW_SIZE/TMP_PAGE_SIZE+2); i++){
        FMC_Write(TMP_PAGE_BASE + i*4, FMC_Read(BANK0_PAGE_CRC_BASE + i*4));
    }
    FMC_Erase(BANK0_PAGE_CRC_BASE);
    for (i = 0; i <(BANK0_FW_SIZE/TMP_PAGE_SIZE+2); i++){
        FMC_Write(BANK0_PAGE_CRC_BASE + i*4,   FMC_Read(BANK1_PAGE_CRC_BASE + i*4));
    }
    FMC_Erase(BANK1_PAGE_CRC_BASE);
    for (i = 0; i <(BANK0_FW_SIZE/TMP_PAGE_SIZE+2); i++){
        FMC_Write(BANK1_PAGE_CRC_BASE + i*4,   FMC_Read(TMP_PAGE_BASE + i*4));
    }
    FMC_Erase(TMP_PAGE_BASE);
    Bank_Swap();

    /* Set remmaping address */
    FMC->ISPADDR = BANK0_FW_BASE;
    /* Set VECMAP */
    FMC->ISPCMD = 0x2E;
    FMC->ISPTRG = 1;
    while(FMC->ISPTRG);
    /* CPU Reset */
    SYS->IPRST0 |= SYS_IPRST0_CPURST_Msk;
}
```

# 5 Conclusion

There are so many reasons to update firmware after products have been delivered, including fixing bugs, adding new functions or increasing security. The products may be delivered anywhere with so many circumstances. Therefore, a reliable method to update the firmware is very important to avoid products return to factory maintenance.

The dual bank firmware upgrade mechanism can avoid product failure caused by firmware upgrade. The Swap continue process can avoid failure caused by unexpected conditions, such as power off, random reset, and system crash. The Swap back process is used to prevent new firmware execution fail. Furthermore, the hardware dual bank architecture improves the performance of firmware download by Flash Read-While-Write (RWW) capability.

## Revision History

| Date | Revision | Description |
|---|---|---|
| 2019.04.12 | 1.00 | 1. Initially issued. |

## Important Notice

**Nuvoton Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, "Insecure Usage".**

**Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, the control or operation of dynamic, brake or safety systems designed for vehicular use, traffic signal instruments, all types of safety devices, and other applications intended to support or sustain life.**

**All Insecure Usage shall be made at customer's risk, and in the event that third parties lay claims to Nuvoton as a result of customer's Insecure Usage, customer shall indemnify the damages and liabilities thus incurred by Nuvoton.**