Using OTA for Firmware Upgrade

Application Note for 32-bit NuMicro® Family

Document Information

Abstract	Introduce the use of OTA for firmware upgrade by the TrustZone® architecture and dual bank Flash to improve system security and performance.
Apply to	NuMicro® M2351 Series

For additional information or questions, please contact: Nuvoton Technology Corporation.

www.nuvoton.com

Table of Contents

1 Overview

OTA (Over-the-Air Technology) is a technique for deploying new firmware to a terminal device and transmitting firmware information by wireless technology. The wireless technology adopted in this document is the Bluetooth 2.0 + EDR SPP profile. Bluetooth 2.0 + EDR is a wireless technology standard for data exchange over short distances, with a wireless operating band of 2.4 to 2.485 GHz for industrial, scientific and medical frequency bands. The SPP is an abbreviation for Bluetooth Serial Port Profile, which simulates a serial cable to replace an existing RS-232 and defines how to set up a virtual serial port and connect two Bluetooth devices. In the wireless sensor network and internet of things (IoT), by hundreds or thousands of nodes in the network, the OTA can achieve system firmware upgrade through the Bluetooth SPP profile. The Arm® TrustZone® technology in the NuMicro® M2351 series microcontroller (MCU) provides a trusted security execution environment to protect the software confidential data including security keys and other security applications.

2 OTA Application Architecture

The Bluetooth 2.0+EDR wireless technology standard can be used for two mobile devices to exchange data over short distances. The OTA technology enables a device with Bluetooth peripheral, according to product requirements, to upgrade the terminal device system firmware of Bluetooth slave side, correct the terminal equipment system problems, or expand its service functions. The use of Arm® TrustZone® technology can strengthen the OTA upgrade system firmware security.

2.1 Firmware Upgrade Architecture

For the Bluetooth 2.0+EDR wireless technology standard, in a piconet network topology, only one is the Bluetooth master, while the other is the Bluetooth slave. The Bluetooth master is served as the OTA server, and the Bluetooth slave is served as the OTA client. The implementation of an OTA upgrade architecture is that the new version of the terminal equipment system firmware is deployed to the OTA client device within the OTA server's wireless communication range through the OTA server. Thus, the system architecture configuration may be different depending on the system application and OTA role.

The main function of OTA server is to transfer the new version of the system firmware and application firmware, through the wireless protocol channel to the OTA client side. Thus, the implementation of OTA server is not limited to the use of the M2351 series, but needs to follow the same OTA protocol. The OTA application architecture example in Figure 2-1 uses the OTA protocol defined by users on a mobile device to transfer the new version of the firmware information through the Bluetooth SPP profile.

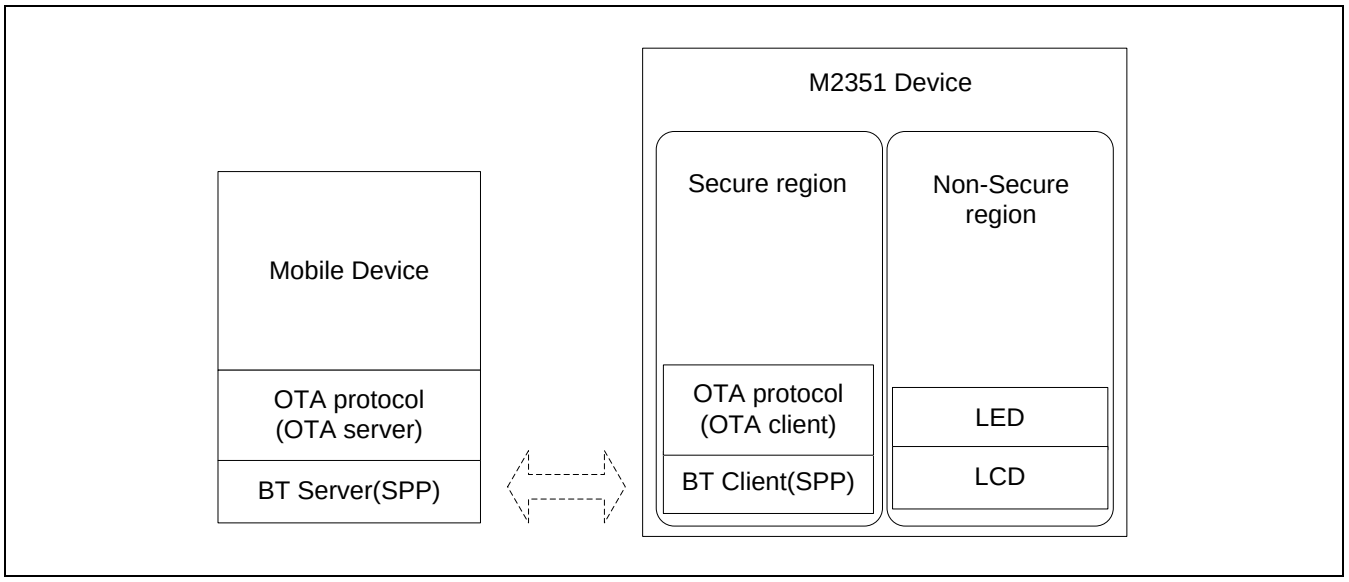

Figure 2-1 OTA Application Architecture

The dual bank Flash diagram is shown in Figure 2-2. When using the M2351 series to implement the OTA client-side system architecture, in response to its upgrade system firmware requirements, Flash memory is divided into system firmware and application firmware. The bank0 of the dual bank Flash memory is configured as a Firmware Upgrade Buffer and the bank1 is configured as an Active Firmware Block. Firmware The Upgrade Buffer is used to write the new version of the system and application firmware; and the Active Firmware Block is used to place the old system and application firmware (that is, the currently operating firmware). To protect the confidentiality of the system firmware, the old system firmware and Firmware Upgrade Buffer are placed in the secure region, and a boot loader is placed in the secure region to determine whether the system is switched to the new version after booting. The current version of the application firmware is placed in the non-secure region. SRAM and the peripherals are also divided into secure region and non-secure region for the system firmware and application firmware respectively according to the application and needs.

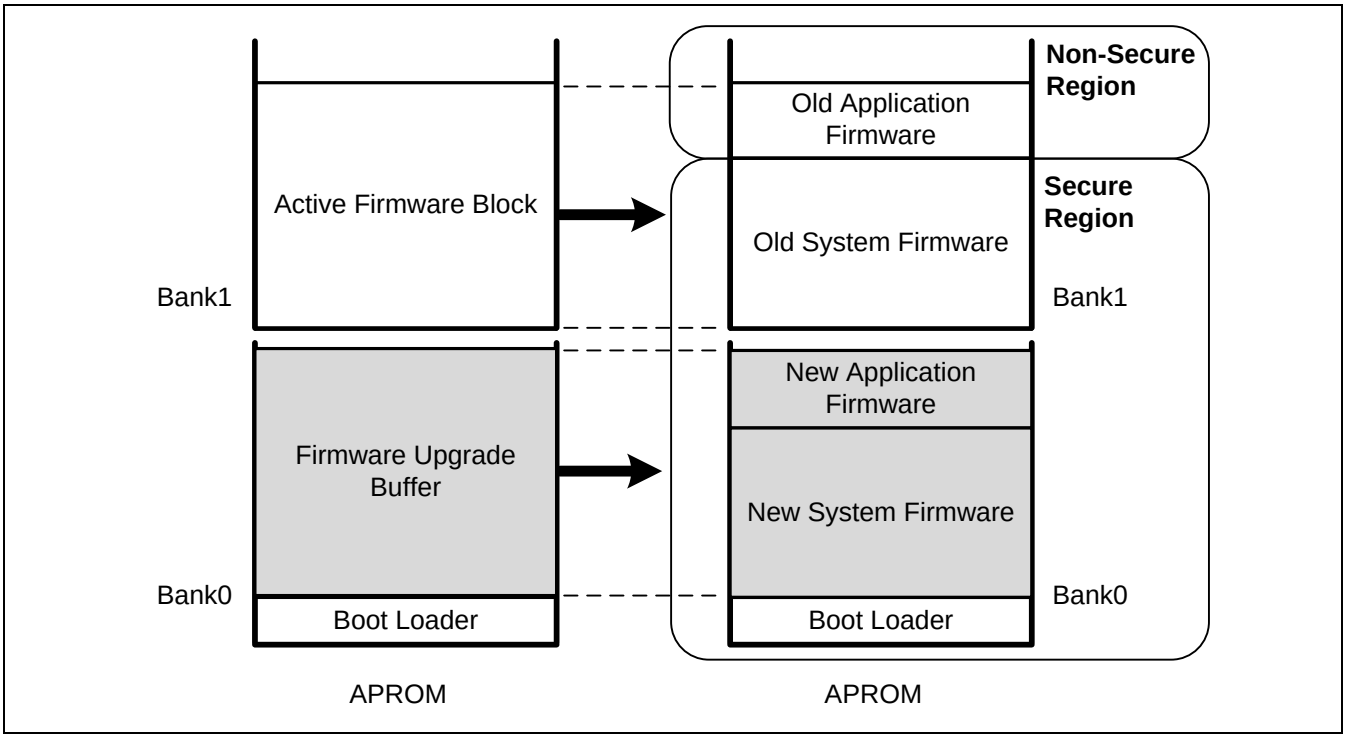

Figure 2-2 OTA Client Dual Bank Flash Diagram

2.2 System and Application Firmware Upgrade Process

The following sections will describe the use of OTA technology to upgrade dual bank Flash, firmware version control, and OTA protocol communication.

2.2.1 Update Process in Dual Bank Flash

In the M2351 secure and non-secure coexistence structure, all the Flash can be divided into two regions, secure and non-secure region, only by setting non-secure boundary address. The non-secure regions cannot be placed between a plurality of secure regions.

To ensure that each time the process of OTA client system or application firmware upgrade can effectively use the dual bank Flash memory architecture advantage, the operating system firmware in the secure region and the application firmware in the non-secure region must be programmed in the Flash bank 1. Also, the new system and application firmware downloaded through the OTA protocol are programmed in the secure region of Flash bank 0.

When the new version of firmware is completely written to Flash bank 0, the system needs to re-boot, so that the boot loader will complete the final step for firmware update. First, confirm that there is a new version of the firmware, and then check the firmware's CRC checksum is correct. If the firmware's CRC checksum is not correct, do not process the firmware update; if it is correct, read the new version of the firmware from Flash bank 0 and write to Flash bank 1 to completely replace the old version of the original firmware.

Figure 2-3 is the firmware upgrade process state in OTA client-side Flash. This example shows the update of both system and application firmware. The OTA client also supports only upgrading one of the system or application firmware in the OTA architecture.

The update process is described below:

1. When the OTA client has received a new version of the firmware through the OTA server, only the Flash bank 1 maintains the currently operating system and application firmware. After firmware update flow is finished, the Flash bank 0 will also keep a copy of the same version of the system and application firmware can be used as a backup firmware.
2. The OTA client receives the new version of the system firmware from the OTA server side, and writes to the Flash bank 0.
3. The OTA client receives the new version of the application firmware from the OTA server side, and writes to the Flash bank 0.
4. The OTA client checks if the new version of the system and application firmware information written to the Flash bank 0 is correct, then resets the system. Then, the boot loader confirms once again the new version of the system firmware information is correct, and the new version of the system firmware will be written into the Flash bank 1.
5. The boot loader confirms again the new version of the application firmware information is correct, and the new version of the application firmware will be written into Flash bank 1. Then the boot loader activates the system firmware in Flash bank 1.

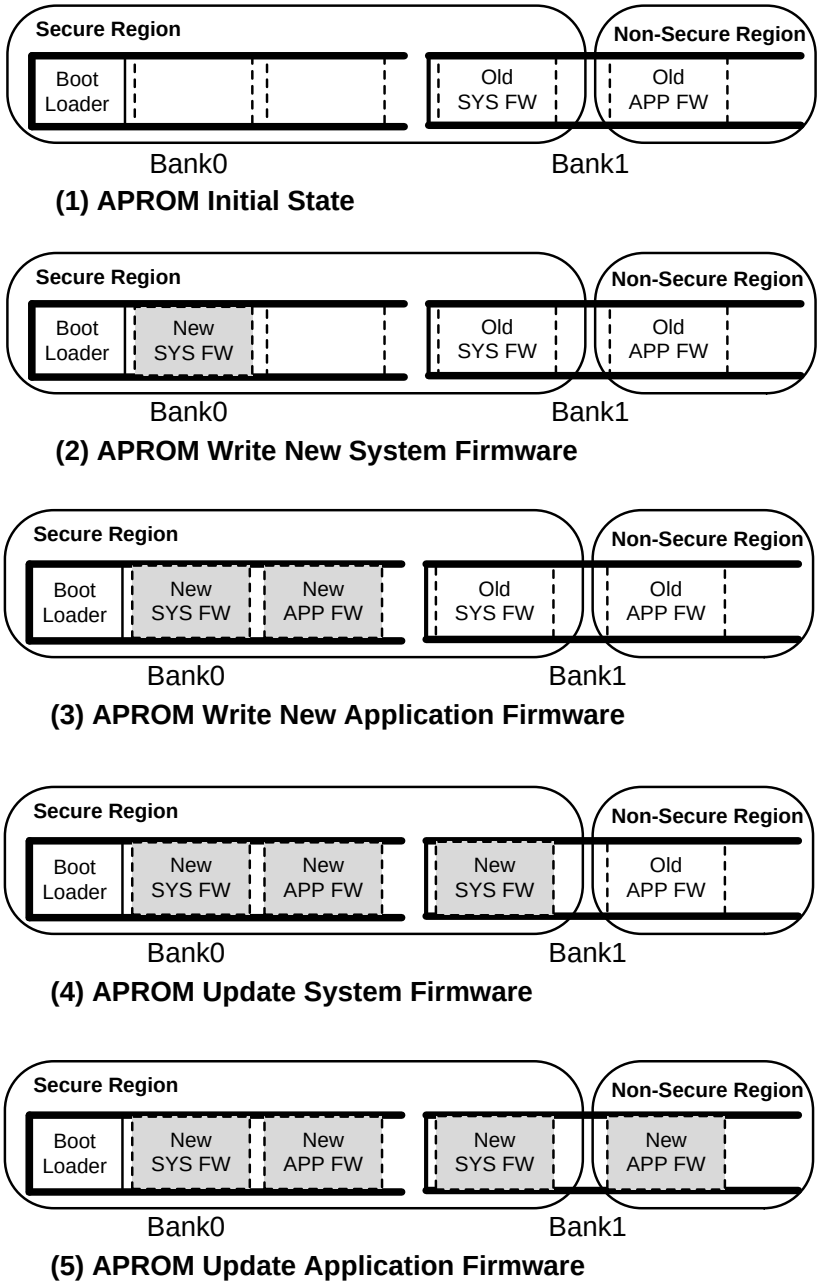

Figure 2-3 Firmware Update State in Dual Bank Flash

6. After the new version of the system or application firmware is released in the OTA server side, the OTA client side will repeat Step 2 ~ 5 for firmware upgrade.

2.2.2 Version Control of System and Application Firmware

The firmware version number of system and application will be included in the bin file in the fixed address, by reading the version number address in each bin file, that is, the version of the bin file. The version number uses 32 bits of data, as shown in Table 2-1, bit 31 ~ 24 for major version number, bit 23 to 16 for minor version number, bit 15 to 8 for the bug fix release, and bit 7 ~ 0 for build number. The greater value of the higher bit indicates a newer version. For example, in the case of (1) 1.0.0.10 and (2) 2.0.0.1, the version number of (2) is newer than (1) because the major version number of (2) is greater than the major version number of (1), even if the build number of (2) is less than (1).

Bit 31 ~ 24	**Bit 23 ~ 16**	**Bit 15 ~ 8**	**Bit 7 ~ 0**
Major version	Minor version	Bug fix release	Build number

Table 2-1 Firmware Version Number

2.2.3 Communication by OTA Protocol

Figure 2-4 illustrates the handshake flow when performing OTA firmware upgrade. The requirements of the OTA application definition firmware upgrade are issued by the OTA server side, which facilitates the control of the OTA client system and the application firmware version.

The whole process is as follows:

1. The OTA server side issues firmware update request (CMD_FWUPDATE_REQ), and sends the verification pattern and the new system or application firmware version number. The purpose of the pattern validation is to allow the OTA client to verify that the new firmware to be deployed by the OTA server is matched.
2. After the OTA client receives the firmware update request, verify if the pattern is correct, and then reply to firmware update confirm (CMD_FWUPDATE_CFM) containing the result to inform the OTA server.
3. When the OTA client side has verified the pattern is correct, it will decide to update only the system firmware, only update the application firmware, update system and application firmware both, or both not update based on OTA server sent to the system and application firmware version number. Then the OTA client sends the result by firmware update type selection request (CMD_UPTYPSEL_REQ) to the OTA server side.
4. After the OTA server receives the OTA client's information of firmware update type selection, the OTA server replies to the firmware update type selection confirm (CMD_UPTYPSEL_CFM) and the OTA client gets the size of the system or application firmware, such that the OTA client can know the reception progress of firmware upgrade.

5. Next, the OTA server side starts to send the firmware data. The whole data of the system firmware will be divided into multiple frames and sent to the OTA client by data of system firmware update indication (CMD_UPSYSDAT_IND) with the system firmware information, and wait for response by OTA client reply data of system firmware update status indication (CMD_UPSYSDATSTS_IND). If the result is correct, the next frame data will be transferred. If the OTA client needs to update the application firmware, the OTA server will send data of application firmware update indication (CMD_UPAPPDAT_IND), and then continue to send the next frame if it receives the correct result from the data of firmware update status indication (CMD_UPAPPDATSTS_IND) from the OTA client.

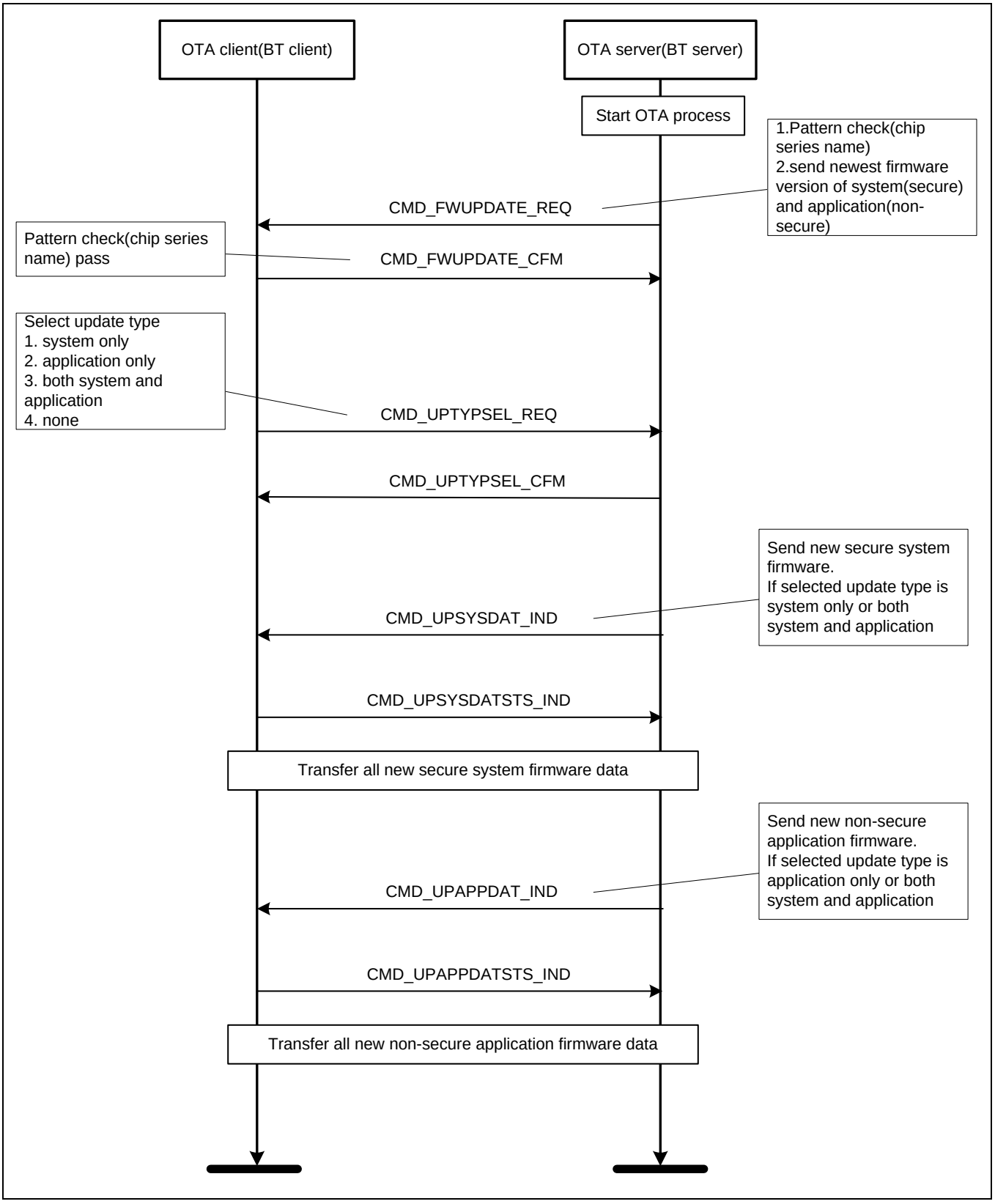

Figure 2-4 OTA Protocol Communication Flow

2.3 Architectural Advantages

The OTA system upgrade service with the above system upgrade architecture provides the following advantages and features - ensure that the deployment of the system source code is secured, the system source code is intact, the new system can be stable and operating normally, and system upgrades can be performed efficiently in the background.

2.3.1 Confidentiality for Update Firmware

Two firmware regions are reserved in the secure region, one is running and the other is for system upgrade to ensure that the system source code is not stolen by applications in the non-secure region of the OTA client device. By placing the OTA protocol in the security region of the TrustZone®, the information used by the OTA protocol and the authentication data for the OTA handshake can be protected from being tampered with.

On the Bluetooth network transmission, the Bluetooth 2.0 + EDR provides Security Mode 1 ~ 3 for connection and data transmission security, and provide Encryption Mode1 ~ 3 for the transmission of information confidentiality. it is recommended that both are set to Mode 3 to enhance the security of network packet transmission.

For users who have higher confidentiality needs for firmware data of the system and application, the new firmware can be encrypted by OTA server and the OTA client needs to decrypt the encrypted firmware to get the raw firmware data.

2.3.2 Integrity for Update Firmware

The new system and application firmware data deployed by the OTA server shall each contain a CRC checksum of the complete system and application firmware. When the OTA client receives the system and application firmware to be upgraded, it needs to recalculate the CRC checksum of new system and application firmware data for comparison, so the OTA client can confirm that the system firmware to be upgraded has been intact write into Flash to ensure that the switch to the new system can work properly.

2.3.3 Stability for System Operation

The OTA client implements a boot loader to select the appropriate system firmware that is not damaged to ensure that the system of OTA client side can work properly. Because the Flash has two system firmware and two application firmware at the same time, and their version number is usually the same. Then the system boot, the boot loader will first check the currently enabled system and application firmware CRC checksum. If the value is correct, the boot loader will read the version number of backup system and application firmware. If the version number is newer than the original enabled firmware, the boot loader will check if the CRC checksum is correct. If the CRC checksum is correct, the boot loader will erase the old version of the system or application firmware, and write the latest system application firmware. Next, the new version of the system or application firmware will start operation. If the new version of the firmware CRC checksum is checked as failed, the boot loader

determines to enable the original system or application firmware, and overwrites the damaged firmware with original system or application firmware. This implementation ensures that the firmware upgrade process can be continuous in the next system boot if power is lost when the system or application is updating.

If the OTA client device takes into account the worst case for the Flash when the two system firmware are damaged, the boot loader can put a copy of the initial system and application firmware version in Flash, which is never overwritten by performing firmware update.

2.3.4 Efficiency for Firmware Upgrade

To take the advantage of dual bank Flash memory hardware architecture, by writing the current operation of the system firmware and the new system firmware to different Flash banks, the M2351 can still perform the current tasks while updating firmware at the same time. This method does not affect the performance of current system, and can process firmware upgrade in the background efficiently.

3 Sample Code

This sample implements an OTA client that uses the M2351 NuTiny board with an HC-05 Bluetooth module, the SPP profile over the Bluetooth 2.0 + EDR protocol, and an OTA server for OTA to upgrade the OTA client-side system and application firmware. The visual difference before and after the firmware update is a flashing frequency of a LED on the M2351 NuTiny board. While the system and application firmware will be upgraded from version 1.0.0.1 to 1.0.0.2.

3.1 Programming Architecture

The firmware programming architecture of the OTA client is shown in Figure 3-1. To add an OTA porting layer under the OTA protocol layer, this porting layer is an interface adaptor for MCU peripheral resources, such as FMC, secure UART, and secure TIMER. The OTA porting layer facilitates the subsequent use of other MCU or replaces Bluetooth wireless module with other wireless module.

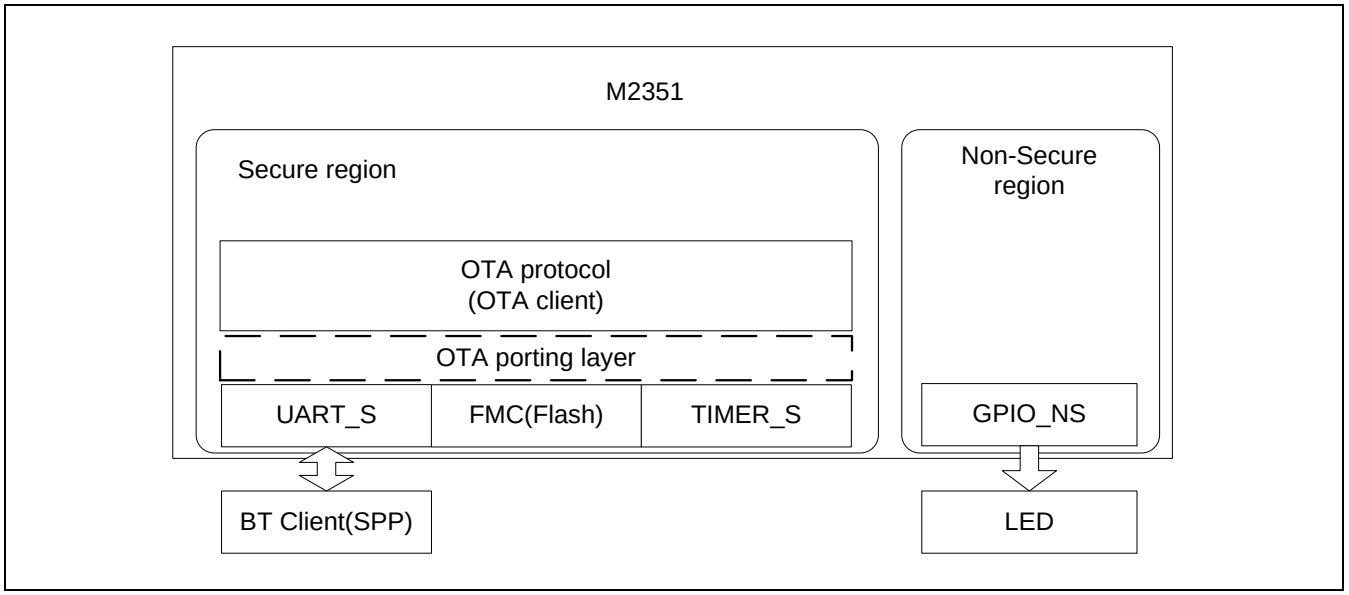

Figure 3-1 OTA Client Firmware Architecture

The currently defined OTA porting layer interface is as follows:

```
/* send OTA command to another device */
void OTA_API_SendFrame(uint8_t* pu8Buff, uint32_t u32Len);

/* configure next receiving buffer length */
void OTA_API_RecvFrame(uint32_t u32Len);

/* the callback interface for a command has received */
```

```
int8_t OTA_API_RecvCallBack(uint8_t* pu8Buff, uint32_t u32Len, uint32_t u32StartIdx,
uint32_t u32ValidLen);
/* init function for any hardware requirements(optional) */
void OTA_API_Init(void);

/* give user to define their own firmware version number definition */
uint8_t OTA_API_GetFWUpdateTypeSel(uint32_t u32NewSysFwVer, uint32_t u32NewAppFwVer);

/* firmware CRC checksum calculation function */
uint32_t OTA_API_CalCrcChkSum32(uint32_t u32addr, uint32_t u32count);

/* get each flash page size for different chip */
uint32_t OTA_API_GetFlashPageSize(void);

/* erase flash */
uint8_t OTA_API_EraseFlash(uint32_t u32FlashAddr);

/* write flash */
uint8_t OTA_API_WriteFlash(uint32_t u32FlashAddr, uint32_t u32Data);

/* get new system firmware CRC checksum */
uint32_t OTA_API_GetNewSysFwChkSum(void);

/* get new application firmware CRC checksum */
uint32_t OTA_API_GetNewAppFwChkSum(void);

/* inform firmware upgrade operation has finish */
void OTA_API_NewFwReady(void);

/* init a timer for receiving data timeout, timeout period is 1s */
void OTA_API_ProgressTimerInit(void);

/* start timer counting*/
void OTA_API_ProgressTimerStart(void);

/* stop timer counting*/
void OTA_API_ProgressTimerStop(void);
```

3.2 OTA Client Sample Code

The following lists definitions of the firmware upgrade macro.

```
#define SYS_FW_BASE          (0x40000UL) /* base address of current system firmware */
#define SYS_FW_BLOCK_SIZE    (0x8000UL) /* block size of current system firmware */
#define SYS_FW_CHECKSUM_BASE (0x47FFCUL) /* CRC checksum address of current system
firmware */


#define APP_FW_BASE          (0x60000UL) /* base address of current application firmware*/
#define APP_FW_BLOCK_SIZE    (0x4000UL) /* block size of current application firmware */
#define APP_FW_CHECKSUM_BASE (0x00063FFCUL) /* CRC checksum address of current application
firmware */


#define SYS_FW_VERSION_BASE (0x00047FF8) /* current system firmware version address */
#define APP_FW_VERSION_BASE (0x00063FF8) /* current application firmware version address*/


/*****************************************************************************/
/****** System Firmware Upgrade Definitions                             ******/
/*****************************************************************************/
#define   SYS_NEW_FW_BASE           (0x00006000UL)        /* 0x0 ~ 0x6000 was reserved for
boot loader(default one flash page size is 2Kbytes) */
#define   SYS_NEW_FW_BLOCK_SIZE     (0x8000UL)           /* include system firmware, non-
secure callable block, CRC checksum and firmware version */
#define   SYS_NEW_FW_VERSION_BASE   (SYS_NEW_FW_BASE + SYS_NEW_FW_BLOCK_SIZE - 0x8UL)   /*
firmware version of new system firmware location, size is one word */
#define   SYS_NEW_FW_CHECKSUM_BASE  (SYS_NEW_FW_VERSION_BASE + 0x4UL)                   /*
CRC checksum of firmware location, size is one word */


/*****************************************************************************/
/****** Application Firmware Upgrade Definitions                        ******/
/*****************************************************************************/
#define   APP_NEW_FW_BASE           (SYS_NEW_FW_BASE + SYS_NEW_FW_BLOCK_SIZE)
#define   APP_NEW_FW_BLOCK_SIZE     (0x4000UL)           /*include application
firmware ,CRC checksum and firmware version(default one flash page size is 2Kbytes) */
#define   APP_NEW_FW_VERSION_BASE   (APP_NEW_FW_BASE + APP_NEW_FW_BLOCK_SIZE - 0x8UL)   /*
firmware version of new application firmware location, size is one word */
#define   APP_NEW_FW_CHECKSUM_BASE  (APP_NEW_FW_VERSION_BASE + 0x4UL)     /* CRC checksum
of firmware location, size is one word */
```

3.2.1 Secure Region Sample Code of OTA Client

The following is the sample code for the main function of the secure region in the OTA client side.

```
/* ------------- */
/* Main function */
/* ------------- */
int main(void)
{
    uint32_t u32MSP_s, u32PSP_s;
    uint32_t u32MSP_ns, u32PSP_ns;
    volatile uint32_t NonSecure_ResetHandler;
    NonSecure_funcptr fp;

    SYS_UnlockReg();
    SYS_Init();
    UART2_Init();

    /* Set relative IO/Peripheral of LCD to non-secure */
    SCU_SET_IONSSET(SCU_IONSSET_PA_Msk);
    SCU_SET_IONSSET(SCU_IONSSET_PC_Msk);

    /* For debug message of non-secure code */
    SCU_SET_PNSSET(UART5_Attr);
    UART5_NS_Init();

    SysTick_Config(SystemCoreClock/10);

    printf("\n");
    printf("+-----------------------------------------+\n");
    printf("|         OTA Client(BT Client)           |\n");
    printf("+-----------------------------------------+\n");

    /* init OTA function */
    OTA_Init();

    Nonsecure_Init();
    while(SYS->PDID)__WFI();

    return 0;
}
```

```
void SYS_Init(void)
{
    int32_t i;
    /*---------------------------------------------------------------------------------------------------------*/
    /* Init System Clock                                                                                       */
    /*---------------------------------------------------------------------------------------------------------*/

    /* Enable PLL */
    CLK->PLLCTL = CLK_PLLCTL_96MHz_HIRC;
    /* Waiting for PLL stable */
    while((CLK->STATUS & CLK_STATUS_PLLSTB_Msk) == 0);

    /* Set HCLK divider to 2 */
    CLK->CLKDIV0 = (CLK->CLKDIV0 & (~CLK_CLKDIV0_HCLKDIV_Msk)) | 1;

    /* Switch HCLK clock source to PLL */
    CLK->CLKSEL0 = (CLK->CLKSEL0 & (~CLK_CLKSEL0_HCLKSEL_Msk)) | CLK_CLKSEL0_HCLKSEL_PLL;

    /* Enable IP clock */
    CLK->APBCLK0 |= CLK_APBCLK0_UART5CKEN_Msk | CLK_APBCLK0_UART2CKEN_Msk;

    /* Select IP clock source */
    CLK->CLKSEL1 = CLK_CLKSEL1_UART0SEL_HIRC;
    CLK->CLKSEL3 = CLK_CLKSEL3_UART2SEL_HIRC | CLK_CLKSEL3_UART5SEL_HIRC;

    /* Update System Core Clock */
    /* User can use SystemCoreClockUpdate() to calculate PllClock, SystemCoreClock and
CycylesPerUs automatically. */
    PllClock        = 96000000;             // PLL
    SystemCoreClock = 96000000 / 2;         // HCLK
    CyclesPerUs     = 48000000 / 1000000;   // For SYS_SysTickDelay()

    /*---------------------------------------------------------------------------------------------------------*/
    /* Init I/O Multi-function                                                                                 */
    /*---------------------------------------------------------------------------------------------------------*/

    /* Init UART2 for Bluetooth */
    SYS->GPB_MFPL |= SYS_GPB_MFPL_PB5MFP_UART2_RXD;
    SYS->GPA_MFPH |= SYS_GPA_MFPH_PA13MFP_UART2_TXD;
```

```
    /* Init UART5 for debug message of non-secure code */
    SYS->GPG_MFPH = SYS_GPG_MFPH_PG9MFP_UART5_RXD | SYS_GPG_MFPH_PG10MFP_UART5_TXD;
}

void UART5_NS_Init(void)
{
    /*---------------------------------------------------------------------------------------*/
    /* Init UART                                                                             */
    /*---------------------------------------------------------------------------------------*/

    /* Configure UART0 and set UART0 Baudrate */
    UART5_NS->BAUD = UART_BAUD_MODE2 | UART_BAUD_MODE2_DIVIDER(__HIRC, 115200);
    UART5_NS->LINE = UART_WORD_LEN_8 | UART_PARITY_NONE | UART_STOP_BIT_1;
}



void UART2_Init(void)
{
    /*---------------------------------------------------------------------------------------*/
    /* Init UART                                                                             */
    /*---------------------------------------------------------------------------------------*/

    /* Configure UART2 and set UART0 Baudrate */
    UART2->BAUD = UART_BAUD_MODE2 | UART_BAUD_MODE2_DIVIDER(__HIRC, 9600);
    UART2->LINE = UART_WORD_LEN_8 | UART_PARITY_NONE | UART_STOP_BIT_1;

    /* Enable UART2 Interrupt */
    UART_ENABLE_INT(UART2, (UART_INTEN_RDAIEN_Msk));
    NVIC_EnableIRQ(UART2_IRQn);

}

/*---------------------------------------------------------------------------
  SysTick IRQ Handler
 *---------------------------------------------------------------------------*/
volatile uint32_t g_u32Ticks = 0;
volatile uint32_t g_u32Ticks_NS = 0;
volatile uint32_t g_u32TimeTicks = 0;
volatile int32_t g_u32Violation = 0;
void SysTick_Handler(void)
{
```

```
    uint32_t lr;
    static uint32_t ticks = 0;

    __ASM volatile ("mov %0, lr\n" : "=r" (lr));

    if(lr & (1 << 6))
    {
        g_u32Ticks++;
    }
    else
    {
        g_u32Ticks_NS++;
    }

    ticks++;

    if(ticks >= 100)
    {
        printf("\n[%05d] NS CPU Usage %3d%%\n",g_u32TimeTicks, g_u32Ticks_NS);
        ticks = 0;
        g_u32Ticks = 0;
        g_u32Ticks_NS = 0;
        g_u32TimeTicks++; // second count

        if(g_u32Violation)
        {
            g_u32Violation--;
            if(!g_u32Violation)
            {
                IO_LED1 = !IO_LED1_ACTIVE;
                IO_LED2 = 0;
            }
        }
    }
}

*-------------------------------------------------------------------------
  Secure functions exported to NonSecure application
  Must place in Non-secure Callable
 *-------------------------------------------------------------------------*/
__attribute__((cmse_nonsecure_entry))
```

```
uint32_t OTA_API_GetSysFwVer()
{
    uint32_t u32SysFwVer;

    SYS_UnlockReg();
    FMC_Open();

    u32SysFwVer = FMC_Read(SYS_FW_VERSION_BASE);

    FMC_Close();
    SYS_LockReg();

    return u32SysFwVer;
}

__attribute__((cmse_nonsecure_entry))
uint32_t OTA_API_GetAppFwVer()
{
    uint32_t u32AppFwVer;

    SYS_UnlockReg();
    FMC_Open();

    u32AppFwVer = FMC_Read(APP_FW_VERSION_BASE);

    FMC_Close();
    SYS_LockReg();

    return u32AppFwVer;
}
```

3.2.2 Non-secure Region Sample Code of OTA Client

The following is the sample code for the main function of the non-secure region of the OTA client.

```
/*----------------------------------------------------------------------------
  NonSecure Callable Functions from Secure Region
 *----------------------------------------------------------------------------*/
extern int32_t Secure_LED_On_callback(void *callback);
extern int32_t Secure_LED_Off_callback(void *callback);
extern int32_t Secure_LED_On(uint32_t num);
extern int32_t Secure_LED_Off(uint32_t num);
extern uint32_t GetSystemCoreClock(void);
extern int32_t SecureFunction(void);

extern uint32_t OTA_API_GetSysFwVer(void);
extern uint32_t OTA_API_GetAppFwVer(void);

/*----------------------------------------------------------------------------
  NonSecure functions used for callbacks
 *----------------------------------------------------------------------------*/
int32_t NonSecure_LED_On(uint32_t num);
int32_t NonSecure_LED_On(uint32_t num)
{
    DEBUG_MSG("Nonsecure LED On call by secure callback\n");
    PA0_NS = 0;
    return 0;
}

int32_t NonSecure_LED_Off(uint32_t num);
int32_t NonSecure_LED_Off(uint32_t num)
{
    DEBUG_MSG("Nonsecure LED Off call by secure callback\n");
    PA0_NS = 1;
    return 0;
}

void LED_On(uint32_t us)
{
    DEBUG_MSG("NS LED On call by NS\n");
    PC14_NS = 0;
}
```

```
void LED_Off(uint32_t us)
{
    DEBUG_MSG("NS LED OFF call by NS\n");
    PC14_NS = 1;
}


/*---------------------------------------------------------------------------
  SysTick IRQ Handler
 *---------------------------------------------------------------------------*/
void SysTick_Handler(void);
void SysTick_Handler(void)
{
    static uint32_t ticks;

    switch(ticks++)
    {
        case   0:
            LED_On(7u);
            break;
        case 20:
            Secure_LED_On(6u);
            break;
        case 40:
            LED_Off(7u);
            break;
        case 60:
            Secure_LED_Off(6u);
            break;
        case 80:
            ticks = 0;
            break;

        default:
            if(ticks > 80)
            {
                ticks = 0;
            }
    }
}
```

```
/*---------------------------------------------------------------------------
  Main function
 *---------------------------------------------------------------------------*/
int main(void)
{
    volatile uint32_t x;
    uint32_t i;
    uint32_t u32SysFwVer;
    uint32_t u32AppFwVer;

    printf("\n");
    printf("+-----------------------------------------+\n");
    printf("|    Nonsecure code is running ...        |\n");
    printf("+-----------------------------------------+\n");

    /* exercise some core register from Non Secure Mode */
    x = __get_MSP();
    x = __get_PSP();

    /* register NonSecure callbacks in Secure application */
    Secure_LED_On_callback(&NonSecure_LED_On);
    Secure_LED_Off_callback(&NonSecure_LED_Off);
    u32SysFwVer = OTA_API_GetSysFwVer();
    u32AppFwVer = OTA_API_GetAppFwVer();
    printf("+-----------------------------------------+\n");
    printf("System Firmware Version: %02d.%02d.%02d.%02d\n", \
           u32SysFwVer>>24, (int32_t)(u32SysFwVer&BYTE2_Msk)>>16,
(int32_t)(u32SysFwVer&BYTE1_Msk)>>8, (int32_t)(u32SysFwVer&BYTE0_Msk));
    printf("Application Firmware Version: %02d.%02d.%02d.%02d\n", \
           u32AppFwVer>>24, (int32_t)(u32AppFwVer&BYTE2_Msk)>>16,
(int32_t)(u32AppFwVer&BYTE1_Msk)>>8, (int32_t)(u32AppFwVer&BYTE0_Msk));
    printf("+-----------------------------------------+\n");

    /* Call secure API to get system core clock */
    SystemCoreClock = GetSystemCoreClock();
    printf("System core clock = %d\n", SystemCoreClock);
    SysTick_Config(SystemCoreClock / 100); /* Generate interrupt each 1 ms */

    /* Non-secure GPIO init for non-secure LED */
    PA_NS->MODE |= (GPIO_MODE_OUTPUT << 0 * 2);
```

```
    PC_NS->MODE |= (GPIO_MODE_OUTPUT << 14 * 2);

    /* Waiting for secure/non-secure SysTick interrupt */
    while(1);
}
```

4 Conclusion

The TrustZone® hardware security architecture of the M2351 series, with its dual bank Flash hardware architecture, implements the technology of the OTA firmware update. This can be easily implemented with the M2351 series to protect the security and confidentiality between firmware upgrade and the firmware update processing in the background without affecting the original system and application firmware operating efficiency.

Revision History

Date	Revision	Description
2018.08.31	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.