How to Use OTA for Firmware Upgrade

Application Note for 32-bit NuMicro® Family

Document Information

Abstract	This document describes a method of OTA for firmware upgrade that implemented by the dual bank Flash to improve system performance.
Apply to	NuMicro® M261 Series

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

www.nuvoton.com

Table of Contents

1 Introduction

Over-the-Air Technology, the abbreviation is OTA, which is a technique for deploying new firmware to a terminal device and transmitting firmware information by wireless technology. The wireless technology adopted in this application note 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 2.4GHz ISM wireless band, belonging to the agreement between low energy and flow consumption without permission to use the band. 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, by hundreds or thousands of nodes in the network, OTA can through the Bluetooth SPP profile to achieve the purpose of system firmware upgrade.

2 Application Overview

Bluetooth 2.0+EDR wireless technology standard, which can be used to allow two mobile devices to exchange data between short distances. The OTA technology can make a device that was equipped with Bluetooth peripheral, according to product requirements to upgrade the terminal device system firmware of Bluetooth slave side, to correct the terminal equipment system problems, or expand its service functions.

2.1 Firmware Upgrade Architecture

In the Bluetooth 2.0+EDR wireless technology standard, in a piconet network topology, only one as the Bluetooth master, while the other is the Bluetooth slave. Bluetooth master is the OTA server, and the Bluetooth slave is the OTA client. The implementation of an OTA upgrade architecture, that is, through the OTA server will be the new version of the terminal equipment system firmware deployed to the OTA client device which within the OTA server's wireless communication range. So planning the system architecture can be different from the system application and OTA role.

The main function of OTA server is to transfer the new version of the firmware, through the wireless protocol channel to the OTA client side, so the implementation of OTA server is not limited to the use of this series of chips, just follow the same OTA agreement can be. The OTA application architecture example in Figure 2-1 is to use the OTA protocol defined by ourselves on a mobile device and transfer the new version of the firmware information through the Bluetooth SPP profile.

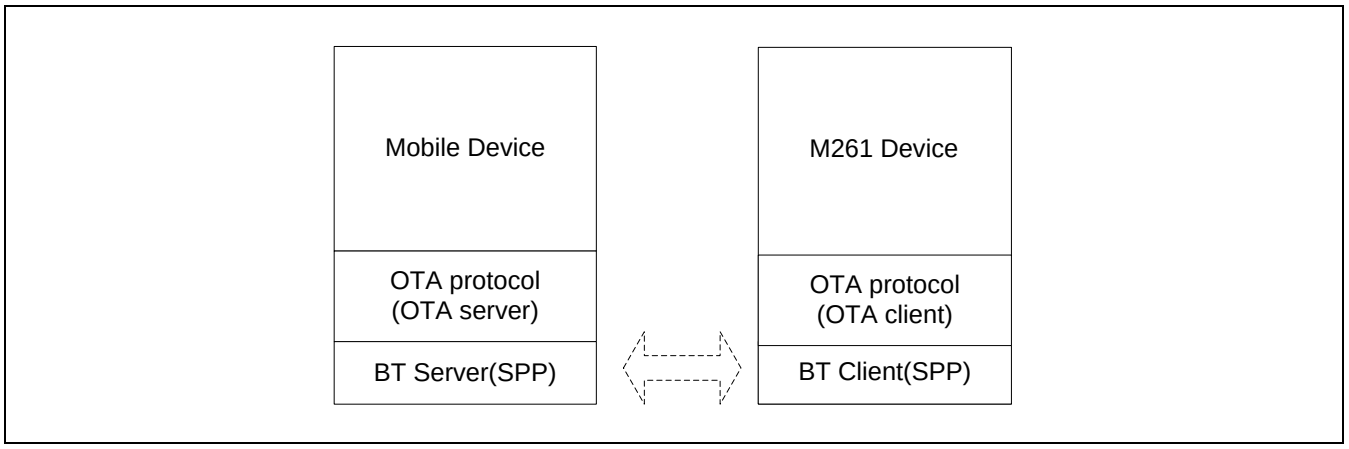

Figure 2-1 OTA Application Architecture

The use of this series of chips to implement the OTA client-side system architecture, in response to its upgrade firmware requirements the bank0 of the dual bank Flash memory is planned as a Firmware Upgrade Buffer and the bank1 is planned as an Active Firmware Block. Firmware Upgrade Buffer is used to write the new version of the firmware; and the Active Firmware Block is to place the old firmware, is also the currently operating firmware. A boot loader in the bank0 of Flash, used to determine the system after power up to switch to

the new version system operation; while the current version of the operating firmware is divided into bank1 of Flash.

The dual bank Flash scheme is as shown in Figure 2-2.

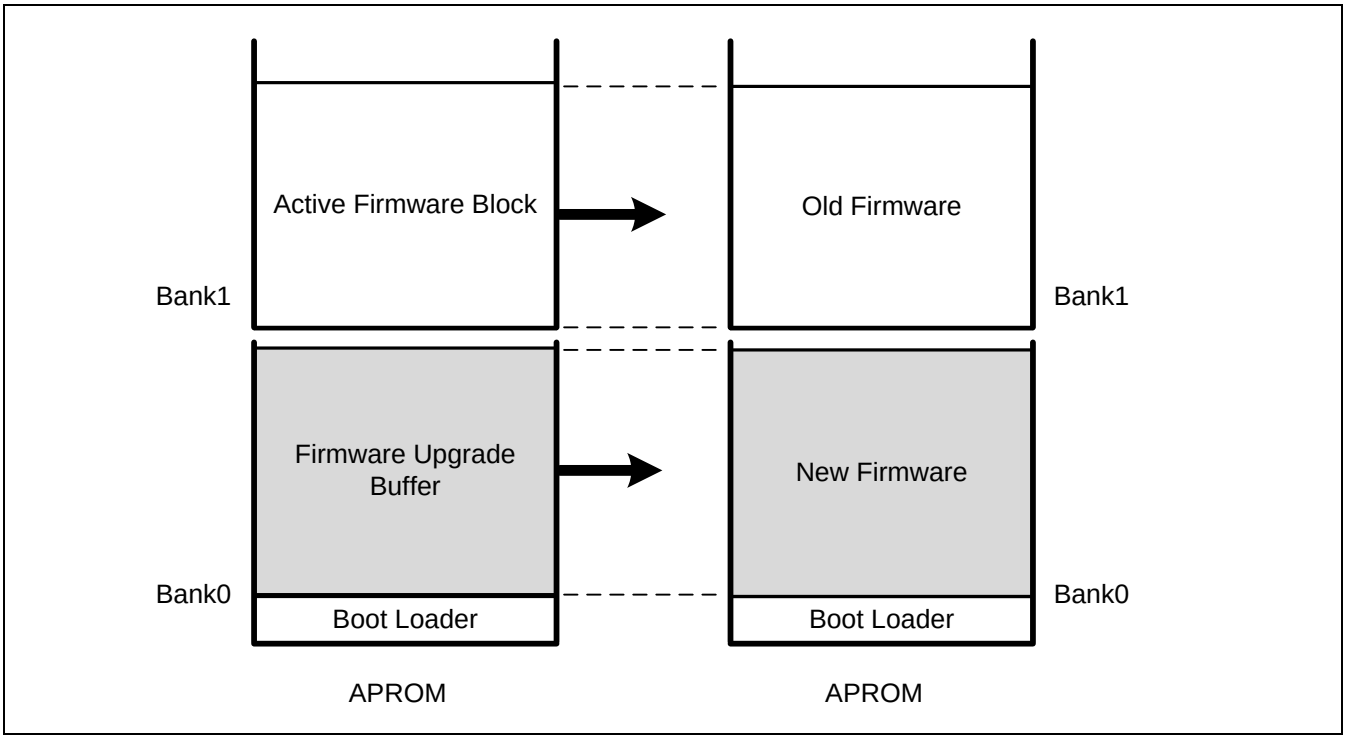

Figure 2-2 OTA Client Dual Bank Flash Diagram

2.2 System and Application Firmware Upgrade Process

The following three sections will depict 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 order to ensure that each time the process of OTA client firmware upgrade, can effectively use the dual bank Flash memory architecture advantage. The currently running firmware must to be stored in the Flash bank 1, and through the OTA protocol download the new version of the firmware are stored in the 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 to 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 it is wrong then do not process the firmware update; if the firmware's CRC checksum 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.

The update process is described below:

(1) When the OTA client has not yet 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 finished, the Flash bank 0 will also keep a copy of the same version of the firmware, can be used as a backup firmware.

(2) The OTA client received the new version of the firmware from the OTA server side, and write to the Flash bank 0.

(3) The current system of OTA client will to determine the new version of the firmware information that wrote to the Flash bank 0 is correct, then reset the system. Then, boot loader confirmed once again the new version of the firmware information is correct, the new version of the firmware will be written into the Flash bank 1. Then boot loader active the system firmware in Flash bank 1 to operation.

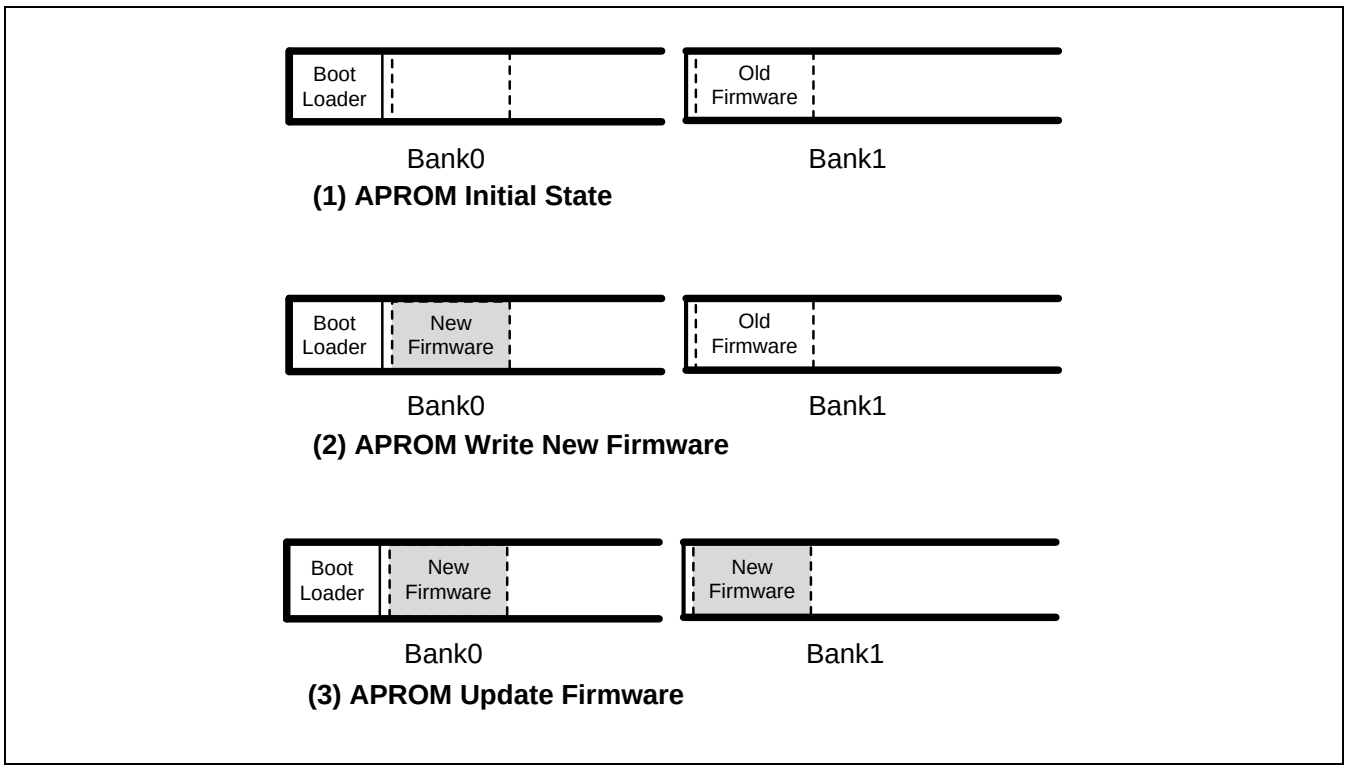

Figure 2-3 Firmware Update State in Dual Bank Flash

After the new version of the firmware were released in the OTA server side, the OTA client side will repeat (2) ~ (3) step for firmware upgrade.

2.2.2 Version Control of Firmware

The firmware version number will be included in the bin file in the fixed address, so by reading the value of version number address in each bin file, that is, that bin file's version. And the version number is the use of 32 bits of data, as shown in Table 2-1, bit 31 ~ 24 for the major version number, bit 23 to 16 for minor version number, bit 15 to 8 for the bug fix release, and bit 7 ~ 0 Build number. The greater value of the higher bit indicates a newer version. For example as (1) 1.0.0.10 and (2) 2.0.0.1, then the version number of (2) is new, because of the major version number of (2) is greater than (1)'s , 2 is greater than 1, even if the build number of (2) is less than (1), 1 is less than 10.

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 firmware version.

The whole steps are as follows:

(1) OTA server side issued firmware update request (CMD_FWUPDATE_REQ), send out the verification pattern and the new firmware version number. The purpose of the validation pattern 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 received the firmware update request, verify that 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 to verify pattern passed, will be based on OTA server sent to the firmware version number, to decide to update firmware or do not update. And then send this decision by firmware update type selection request (CMD_UPTYPSEL_REQ) to the OTA server side.

(4) After receiving the OTA client, the OTA client will reply to the firmware update type selection confirm (CMD_UPTYPSEL_CFM) and give the size of the firmware so that the OTA client can know the reception progress of firmware upgrade.

(5) Next, OTA server side will start to send the firmware data. Whole data of the firmware will be divided into multiple frame and send to OTA client by data of firmware update indication (CMD_UPSYSDAT_IND) with the firmware information, and wait for response by OTA client reply data of firmware update status indication (CMD_UPSYSDATSTS_IND). If the result is correct, the next frame data will be transferred.

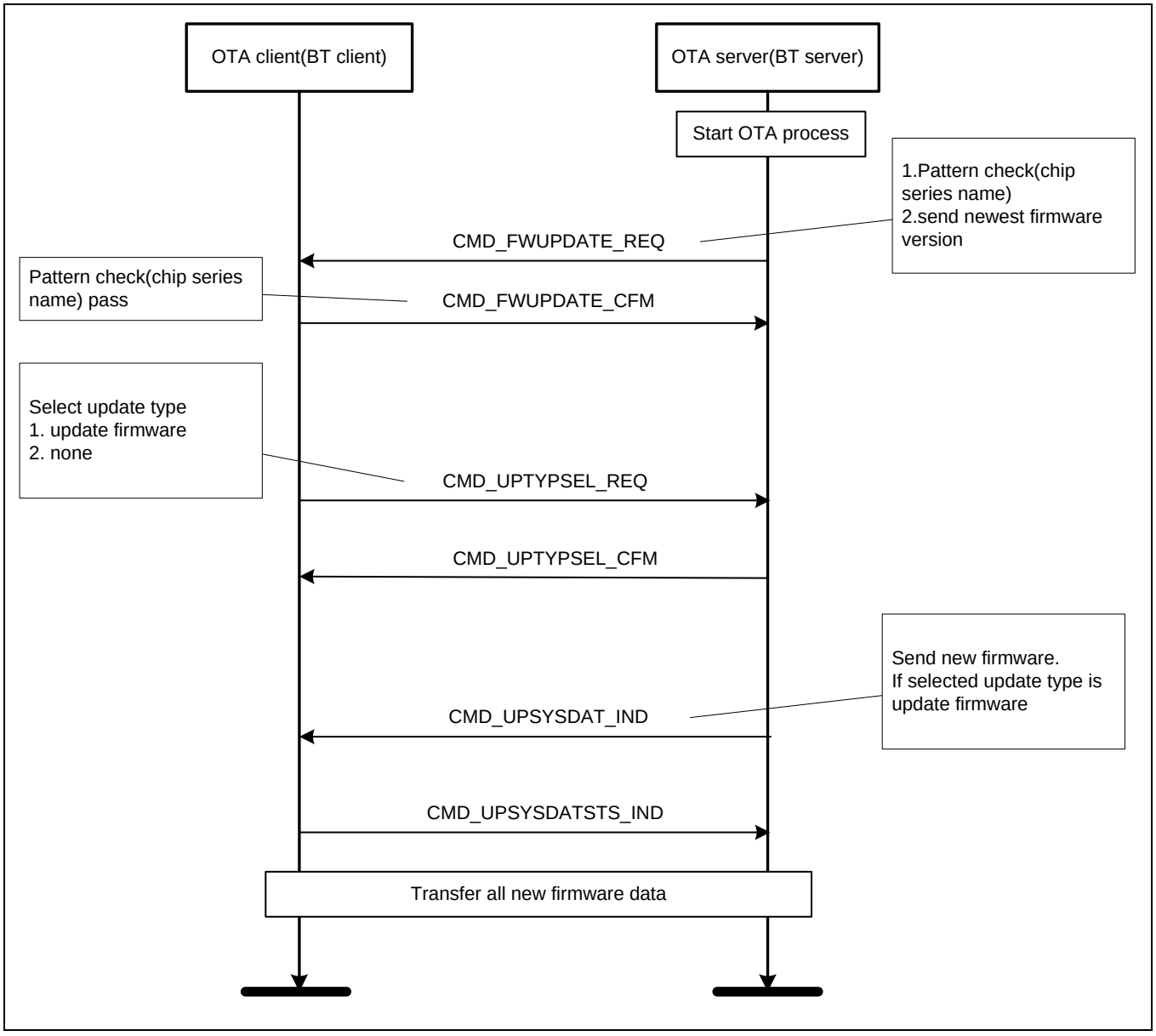

Figure 2-4 OTA Protocol Communication Flow

2.3 Architectural Advantages

The OTA system upgrade service with the above system upgrade architecture provides the following advantages and features. To ensure that the system source code intact, the new system can be stable and normal operation, as well as perform system upgrades efficiently in the background.

2.3.1 Integrity for Update Firmware

OTA protocol can be used to communication the update selection of OTA client side, update the firmware or do not update. The new firmware data deployed by the OTA server shall each contain a CRC checksum of the complete firmware. When the OTA client receives the firmware to be upgraded, it needs to recalculate the CRC checksum of new firmware data to do comparison, so OTA client can confirm that the firmware to be upgraded has been intact write into Flash, in order to ensure that the switch to the new system can work properly.

2.3.2 Stability for System Operation

In the OTA client to implement a boot loader, is to select the appropriate and no damage to the firmware, to ensure that the system of OTA client side can work properly. Because the Flash will exist two firmware, and these usually are have the same version number. Then the system boot, the first will check the currently enabled firmware CRC checksum. If the value is correct, boot loader will read the backup of firmware version number. If the version number is newer than the original enabled firmware, then check its CRC checksum is correct. If checked by CRC checksum, then erase the old version of the firmware, and write the latest firmware. Next, the new version of the firmware will start operation. If the new version of the firmware CRC checksum failed to check, then choose to enable the original firmware version, and erase the firmware version of the new firmware; write the currently enabled firmware, used as a spare firmware. This implementation ensures that the firmware upgrade process can be continuous in the next system boot if power is lost when the system is updating.

If OTA client device take into account the worst case that both the two firmware in the Flash are damaged. It can place a copy of the initial firmware version in Flash, and never be overwrote by perform firmware update.

2.3.3 Efficiency for Firmware Upgrade

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

3 Sample Description

This sample implements an OTA client that uses an NuTiny board with an HC-05 Bluetooth module, the SPP profile over the Bluetooth 2.0 + EDR protocol, and an OTA server for OTA upgrades 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 NuTiny board. While the 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, UART, and TIMER. This OTA porting layer facilitates the subsequent use of other MCU or replace Bluetooth wireless module with other wireless module.

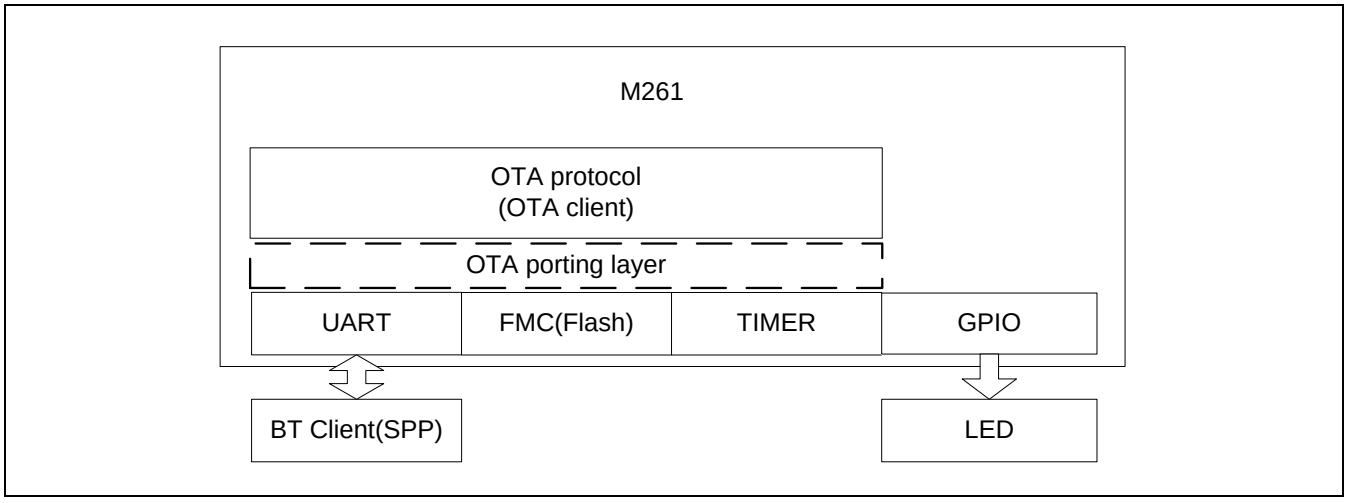

Figure 3-1 OTA Client Firmware Architecture

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 u32NewFwVer);

/* 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);

/* 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 Sample Code of OTA Client

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 SYS_FW_VERSION_BASE (0x00047FF8) /* current 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, 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 */
```

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

```
#define IO_LED          PC14     // LED on NuTiny

void SYS_Init(void);
void UART5_Init(void);
void UART2_Init(void);

uint32_t GetSystemCoreClock(void)
{
    return SystemCoreClock;
}

void LED_On(uint32_t us)
{
    DEBUG_MSG("LED On\n");
    IO_LED = 0;
}

void LED_Off(uint32_t us)
{
```

```
    DEBUG_MSG("LED OFF\n");
    IO_LED = 1;
}

/*---------------------------------------------------------------------------
  SysTick IRQ Handler
 *---------------------------------------------------------------------------*/
void SysTick_Handler(void)
{
    static uint32_t ticks = 0;

    switch(ticks++)
    {
        case   0:
            LED_On(7u);
            break;
        case 40:
            LED_Off(7u);
            break;
        case 80:
            ticks = 0;
            break;

        default:
            if(ticks > 80)
            {
                ticks = 0;
            }
    }
}
/* ------------- */
/* Main function */
/* ------------- */
int main(void)
{
    uint32_t u32SysFwVer;

    SYS_UnlockReg();
    SYS_Init();

    /* For BT control */
```

```
    UART2_Init();

    /* For debug message */
    UART5_Init();

    /* Configure PC.14 as output mode for IO_LED */
    PC->MODE |= (GPIO_MODE_OUTPUT << 14 * 2);

    /* Configre timuout time for system tick */
    SysTick_Config(SystemCoreClock / 100);

    printf("\n");
    printf("+-----------------------------------------+\n");
    printf("|         OTA Client(BT Client)           |\n");
    printf("+-----------------------------------------+\n");
    u32SysFwVer = OTA_API_GetSysFwVer();
    printf("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("System core clock = %d\n", SystemCoreClock);

    /* OTA initialization */
    OTA_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. */
    //SystemCoreClockUpdate();
    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 */
    SYS->GPG_MFPH = SYS_GPG_MFPH_PG9MFP_UART5_RXD | SYS_GPG_MFPH_PG10MFP_UART5_TXD;
}

void UART5_Init(void)
{
    /*---------------------------------------------------------------------------------------*/
    /* Init UART                                                                             */
    /*---------------------------------------------------------------------------------------*/
    /* Configure UART5 and set UART5 Baudrate */
    UART5->BAUD = UART_BAUD_MODE2 | UART_BAUD_MODE2_DIVIDER(__HIRC, 115200);
```

```
    UART5->LINE = UART_WORD_LEN_8 | UART_PARITY_NONE | UART_STOP_BIT_1;
}

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

    /* Configure UART2 and set UART2 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);

}
```

4 Conclusion

The M261 series chip, with its dual bank Flash hardware architecture, implements the technology of the OTA firmware update. The OTA firmware update can be processed in the background without affecting the original firmware operating efficiency.

Revision History

Date	Revision	Description
2019.04.08	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.