Using M480 IAP to Upgrade APROM and SPI Flash

Example Code Introduction for 32-bit NuMicro® Family

Information

Application	This sample code uses IAP bootloader program to upgrade APROM and SPI Flash program.
BSP Version	M480 Series BSP CMSIS V3.03.000
Hardware	NuMaker-M487KM v1.0

The information described in this document is the exclusive intellectual property of Nuvoton Technology Corporation and shall not be reproduced without permission from Nuvoton.

Nuvoton is providing this document only for reference purposes of NuMicro microcontroller and microprocessor based system design. Nuvoton assumes no responsibility for errors or omissions.

All data and specifications are subject to change without notice.

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

www.nuvoton.com

1 Overview

IAP (In-Application Programming) bootloader allows the M480 microcontrollers to update the program in other Flash address block. When using IAP, the M480 will point to the address of the bootloader program by vector table, and the program in other Flash address is able to be updated. When the upgrade is completed, the vector table will be redirected to the new program address to execute it.

There are two key points with regard to IAP:

1. For CPU, all space in APROM and LDROM are visible, and user's code can be put anywhere in APROM or LDROM.
2. User's application program (APP) can be located at any address in APROM/LDROM, instead of the lowest address 0x00.

As shown in Figure 1-1, CPU will run the code in APP.

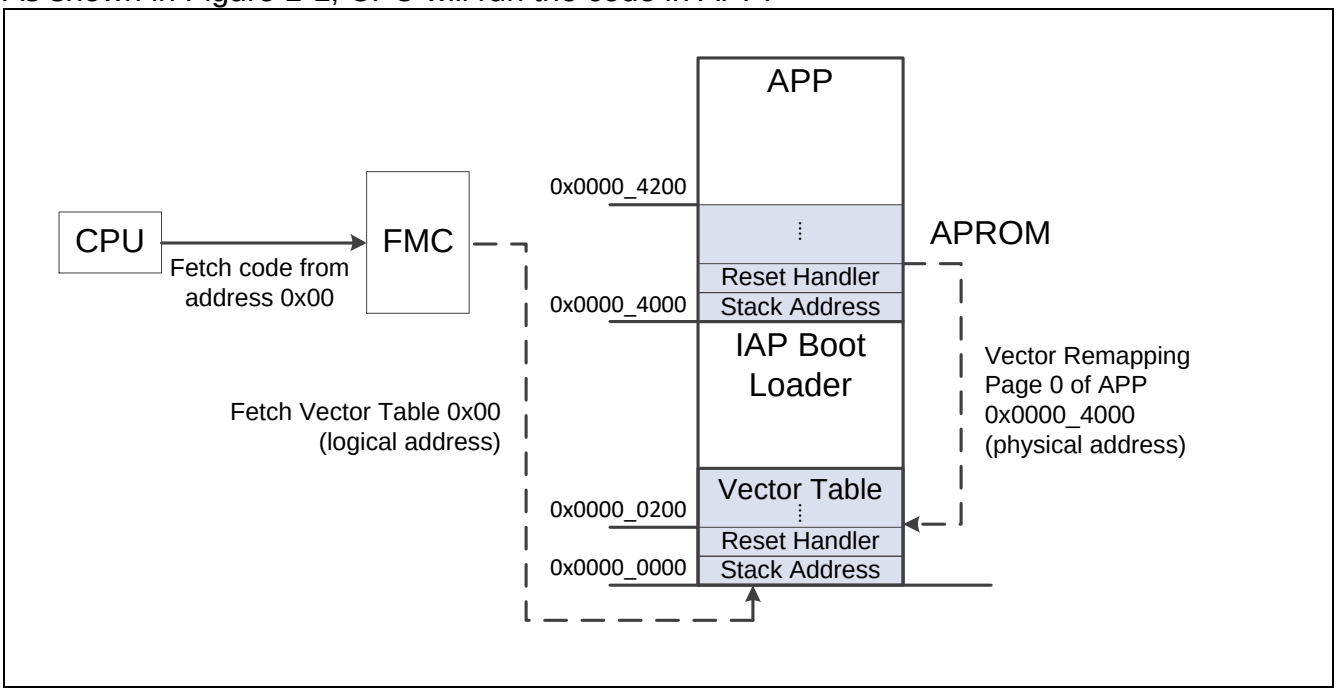

Figure 1-1 Booting from APP in APROM with IAP

After the APP in APROM is updated, the IAP allows APP to be called each other with system reset or without any reset. Some APPs do not need to be switched with system reset. At this time, as long as the stack pointer R13 is modified and the vector table is remapped, the APP will be switched and executed.

1.1 Demo Result

The demo result is shown below.

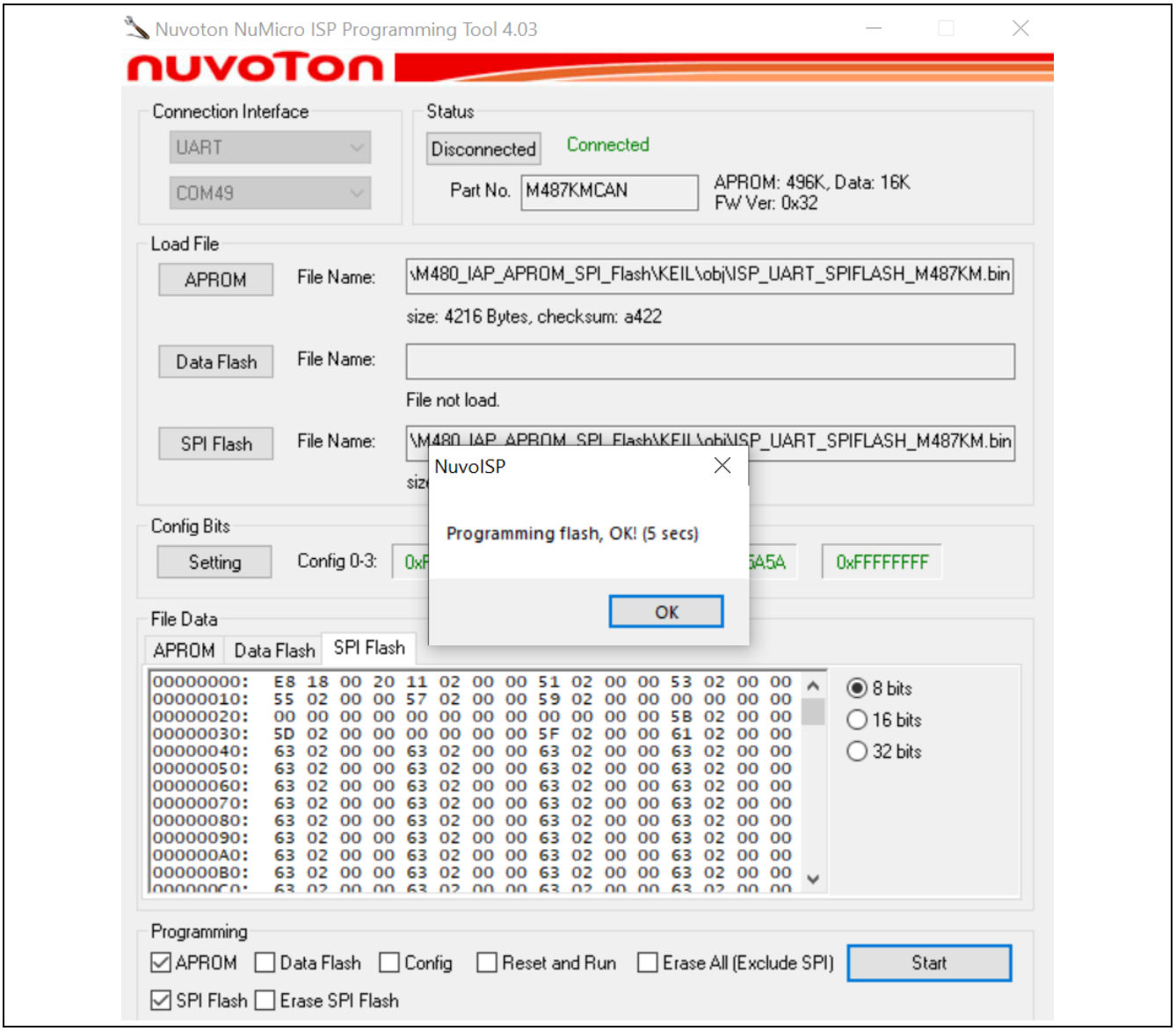

Figure 1-2 Demo Result

The IAP bootloader program is connected with ISP tool to update the APP in APROM by UART port.

The download path of ISP tool:

https://www.nuvoton.com/resource-download.jsp?tp_GUID=SW1720201209134839

2 Code Description

The IAP bootloader main function sets system clock and UART function, and will use UART to communicate with ISP tool for upgrade. If user does not need to update APP, the M480 will call func() function to switch program in FMC_AP_BASE address.

```
int main(void)
{
    FUNC_PTR    *func;                  /* function pointer */

    /* Unlock protected registers */
    SYS_UnlockReg();
    /* Init System, peripheral clock and multi-function I/O */
    SYS_Init();
    SPIM_Init();
    /* Init UART to 115200-8n1 */
    UART_Init();
    CLK->AHBCLK |= CLK_AHBCLK_ISPCKEN_Msk;
    FMC->ISPCTL |= FMC_ISPCTL_ISPEN_Msk;
    FMC->ISPCTL |= FMC_ISPCTL_APUEN_Msk;
    g_apromSize = GetApromSize();
    GetDataFlashInfo(&g_dataFlashAddr, &g_dataFlashSize);
    SysTick->LOAD = 300000 * CyclesPerUs;
    SysTick->VAL   = (0x00);
    SysTick->CTRL = SysTick->CTRL | SysTick_CTRL_CLKSOURCE_Msk |
SysTick_CTRL_ENABLE_Msk;//using cpu clock

    while(1) {
        if((bufhead >= 4) || (bUartDataReady == TRUE)) {
            uint32_t lcmd;
            lcmd = inpw(uart_rcvbuf);

            if(lcmd == CMD_CONNECT) {
                goto _ISP;
            } else {
                bUartDataReady = FALSE;
                bufhead = 0;
            }
        }

        //timeout, then goto APROM
        if(SysTick->CTRL & SysTick_CTRL_COUNTFLAG_Msk) {
            goto _APROM;
        }
    }

_ISP:

    while(1) {
        if(bUartDataReady == TRUE) {
            bUartDataReady = FALSE;
            ParseCmd(uart_rcvbuf, 64);
            PutString();
        }
        if(RUN_AP == 1) {
            goto _APROM;
        }
    }

_APROM:

    while(!(UART0->FIFOSTS & UART_FIFOSTS_TXEMPTY_Msk));
    /*  NOTE! Before change VECMAP, user MUST disable all interrupts. */
```

```
    /* Vector remap APROM page 0 to address 0. */
    FMC_SetVectorPageAddr(FMC_AP_BASE);
    SYS_LockReg();

    /* The reset handler address of an executable image is located at offset 0x4.
  Thus, this sample get reset handler address of APROM code from FMC_APROM_BASE + 0x4. */
    func = (FUNC_PTR *)*(uint32_t *)(FMC_AP_BASE + 4);

    /* The stack base address of an executable image is located at offset 0x0.
       Thus, this sample get stack base address of APROM code from FMC_APROM_BASE + 0x0.*/
    __set_SP(*(uint32_t *)FMC_AP_BASE);

    /* Branch to the LDROM code's reset handler in way of function call.  */
    func();
    while(1);
}
```

This function parses the command from ISP tool to complete upgrade flow.

```
int ParseCmd(unsigned char *buffer, uint8_t len)
{
    static uint32_t StartAddress, StartAddress_bak, TotalLen, TotalLen_bak, LastDataLen,
g_packno = 1;
    uint8_t *response;
    uint16_t lcksum;
    uint32_t lcmd, srclen, regcnf0, security;
    unsigned char *pSrc;
    static uint32_t gcmd;
    response = response_buff;
    pSrc = buffer;
    srclen = len;
    lcmd = inpw(pSrc);
    outpw(response + 4, 0);
    pSrc += 8;
    srclen -= 8;
    ReadData(Config0, Config0 + 16, (uint32_t *)(response + 8)); //read config
    regcnf0 = *(uint32_t *)(response + 8);
    security = regcnf0 & 0x2;

    if(lcmd == CMD_SYNC_PACKNO) {
        g_packno = inpw(pSrc);
    }

    if((lcmd) && (lcmd != CMD_RESEND_PACKET)) {
        gcmd = lcmd;
    }

    if(lcmd == CMD_GET_FWVER) {
        response[8] = FW_VERSION;//version 2.3
    } else if(lcmd == CMD_GET_DEVICEID) {
        outpw(response + 8, SYS->PDID);
        goto out;
    } else if(lcmd == CMD_RUN_APROM || lcmd == CMD_RUN_LDROM || lcmd == CMD_RESET) {
        RUN_AP=1;
    } else if(lcmd == CMD_CONNECT) {
        outpw(response + 8, 0x001540EF);
        g_packno = 1;
        goto out;
    } else if((lcmd == CMD_UPDATE_APROM) || (lcmd == CMD_ERASE_ALL)) {
        EraseAP(FMC_AP_BASE, (g_apromSize < g_dataFlashAddr) ? (g_apromSize -
FMC_AP_BASE) : (g_dataFlashAddr - FMC_AP_BASE)); // erase APROM, g_dataFlashAddr,
g_apromSize

        if(lcmd == CMD_ERASE_ALL) {  //erase data flash
```

```
            EraseAP(g_dataFlashAddr, g_dataFlashSize);
            *(uint32_t *)(response + 8) = regcnf0 | 0x02;
            UpdateConfig((uint32_t *)(response + 8), NULL);
        }

        bUpdateApromCmd = TRUE;
    } else if(lcmd == CMD_GET_FLASHMODE) {
        //return 1: APROM, 2: LDROM
        outpw(response + 8, (FMC->ISPCTL & 0x2) ? 2 : 1);
    }

    if((lcmd == CMD_UPDATE_APROM) || (lcmd == CMD_UPDATE_DATAFLASH)) {
        if(lcmd == CMD_UPDATE_DATAFLASH) {
            StartAddress = g_dataFlashAddr;

            if(g_dataFlashSize) {
                EraseAP(g_dataFlashAddr, g_dataFlashSize);
            } else {
                goto out;
            }
        } else {
            StartAddress = FMC_AP_BASE;
        }

        TotalLen = inpw(pSrc + 4);
        pSrc += 8;
        srclen -= 8;
        StartAddress_bak = StartAddress;
        TotalLen_bak = TotalLen;
    } else if(lcmd == CMD_UPDATE_CONFIG) {
        if((security == 0) && (!bUpdateApromCmd)) {  //security lock
            goto out;
        }

        UpdateConfig((uint32_t *)(pSrc), (uint32_t *)(response + 8));
        GetDataFlashInfo(&g_dataFlashAddr, &g_dataFlashSize);
        goto out;
    } else if(lcmd == CMD_RESEND_PACKET) {  //for APROM & Data flash only
        uint32_t PageAddress;
        StartAddress -= LastDataLen;
        TotalLen += LastDataLen;
        PageAddress = StartAddress & (0x100000 - FMC_FLASH_PAGE_SIZE);

        if(PageAddress >= Config0) {
            goto out;
        }

        ReadData(PageAddress, StartAddress, (uint32_t *)aprom_buf);
        FMC_Erase_User(PageAddress);
        WriteData(PageAddress, StartAddress, (uint32_t *)aprom_buf);

        if((StartAddress % FMC_FLASH_PAGE_SIZE) >= (FMC_FLASH_PAGE_SIZE - LastDataLen)) {
            FMC_Erase_User(PageAddress + FMC_FLASH_PAGE_SIZE);
        }

        goto out;
    } else if(lcmd == CMD_ERASE_SPIFLASH) {
        uint32_t u32Addr = inpw(pSrc);
        SPIM_EraseBlock(u32Addr, 0, OPCODE_BE_64K, 1, 1);
    } else if(lcmd == CMD_UPDATE_SPIFLASH) {
        uint32_t u32Addr, u32NTx;
        u32Addr = inpw(pSrc);
        u32NTx = inpw(pSrc + 4);
        pSrc += 8;
        SPIM_DMA_Write(u32Addr, 0, u32NTx, pSrc, CMD_NORMAL_PAGE_PROGRAM);
    }
```

```
    if((gcmd == CMD_UPDATE_APROM) || (gcmd == CMD_UPDATE_DATAFLASH)) {
        if(TotalLen < srclen) {
            srclen = TotalLen;//prevent last package from over writing
        }

        TotalLen -= srclen;
        WriteData(StartAddress, StartAddress + srclen, (uint32_t *)pSrc);
        memset(pSrc, 0, srclen);
        ReadData(StartAddress, StartAddress + srclen, (uint32_t *)pSrc);
        StartAddress += srclen;
        LastDataLen =  srclen;

        if(TotalLen == 0) {
            lcksum = CalCheckSum(StartAddress_bak, TotalLen_bak);
            outps(response + 8, lcksum);
        }
    }

out:
    lcksum = Checksum(buffer, len);
    outps(response, lcksum);
    ++g_packno;
    outpw(response + 4, g_packno);
    g_packno++;
    return 0;
}
```

This part of ParseCmd() function is for SPI Flash upgrade.

```
    } else if(lcmd == CMD_ERASE_SPIFLASH) {
        uint32_t u32Addr = inpw(pSrc);
        SPIM_EraseBlock(u32Addr, 0, OPCODE_BE_64K, 1, 1);
    } else if(lcmd == CMD_UPDATE_SPIFLASH) {
        uint32_t u32Addr, u32NTx;
        u32Addr = inpw(pSrc);
        u32NTx = inpw(pSrc + 4);
        pSrc += 8;
        SPIM_DMA_Write(u32Addr, 0, u32NTx, pSrc, CMD_NORMAL_PAGE_PROGRAM);
    }
```

This part of ParseCmd() function is for APROM and Data Flash upgrade.

```
    if((gcmd == CMD_UPDATE_APROM) || (gcmd == CMD_UPDATE_DATAFLASH)) {
        if(TotalLen < srclen) {
            srclen = TotalLen;//prevent last package from over writing
        }

        TotalLen -= srclen;
        WriteData(StartAddress, StartAddress + srclen, (uint32_t *)pSrc);
        memset(pSrc, 0, srclen);
        ReadData(StartAddress, StartAddress + srclen, (uint32_t *)pSrc);
        StartAddress += srclen;
        LastDataLen =  srclen;

        if(TotalLen == 0) {
            lcksum = CalCheckSum(StartAddress_bak, TotalLen_bak);
            outps(response + 8, lcksum);
        }
    }
```

FMC_AP_BASE is the APP start address. If the APP has other address, user could revise it to fit the APP.

```
#define Config0         FMC_CONFIG_BASE
#define Config1         (FMC_CONFIG_BASE+4)
#define FMC_AP_BASE     0x4000     //APP start address
#define ISPGO           0x01
```

3 Software and Hardware Requirements

3.1 Software Requirements

- BSP version
  - M480 Series BSP CMSIS V3.03.000
- IDE version
  - Keil uVersion 5.26
- ISP version
  - NuMicro ISP Programming Tool V4.03

3.2 Hardware Requirements

- Circuit components
  - NuMaker-M487KM v1.0

- Pin Connect

Use printf function to output message and the UART0 TX RX pin are PB.13 and PB.12 respectively.

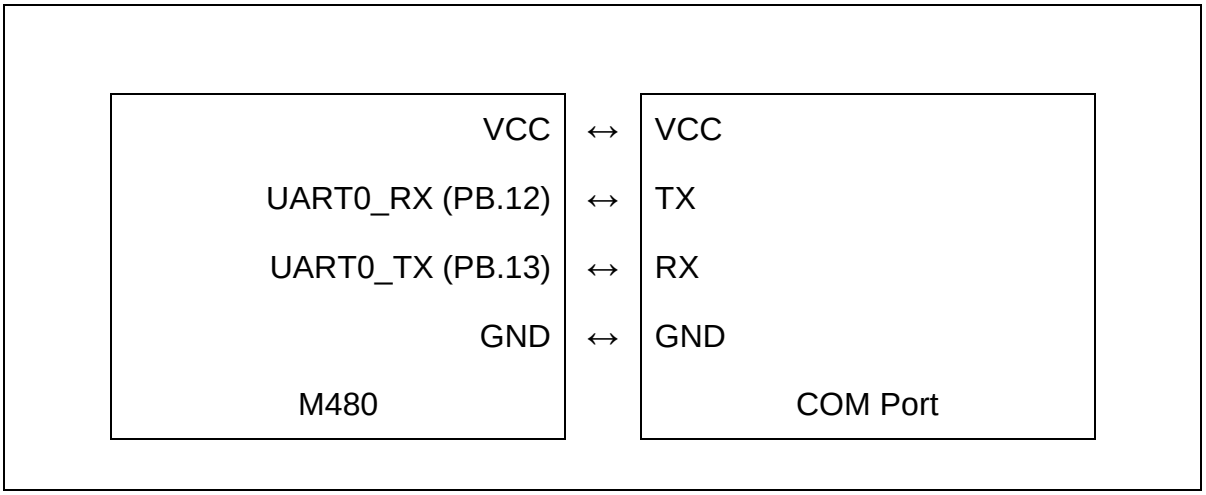

Figure 3-1 Pin Connect

4 Directory Information

The directory structure is shown below.

EC_M480_IAP_APROM_SPI_Flash_V1.00	
Library	Sample code header and source files
CMSIS	Cortex® Microcontroller Software Interface Standard (CMSIS) by Arm® Corp.
Device	CMSIS compliant device header file
StdDriver	All peripheral driver header and source files
SampleCode	
ExampleCode	Source file of example code

Figure 4-1 Directory Structure

5 Example Code Execution

1. Browse the sample code folder as described in the Directory Information section and double-click *M480_IAP_APROM_SPI_Flash.uvproj*.
2. Enter Keil compile mode.
   - Build
   - Download
   - Start/Stop debug session
3. Enter debug mode.
   - Run

6 Revision History

Date	Revision	Description
2021.08.20	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.

Please note that all data and specifications are subject to change without notice.
All the trademarks of products and companies mentioned in this datasheet belong to their respective owners.