Updating APROM and SPI Flash via M032 ISP

Example Code Introduction for 32-bit NuMicro® Family

Document Information

Application	This example code uses ISP LDROM program to update APROM and data in SPI Flash.
BSP Version	M031_Series_BSP_CMSIS_V3.07.000
Hardware	NuMaker-M032KI 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

APROM usually stores user code, while LDROM generally stores the Bootloader for In-System-Programming (ISP). This LDROM example code receives user code via UART interface and then updates the user code into APROM through software register control to achieve the In-System Programming (ISP) function. It also supports updating data to external SPI Flash through the instruction set of the SPI Flash.

1.1 Principle

The ISP Receive/Send data packet is 64 bytes no matter ACK packet is received or command packet is sent. The data packet is sent and received via the UART interface.

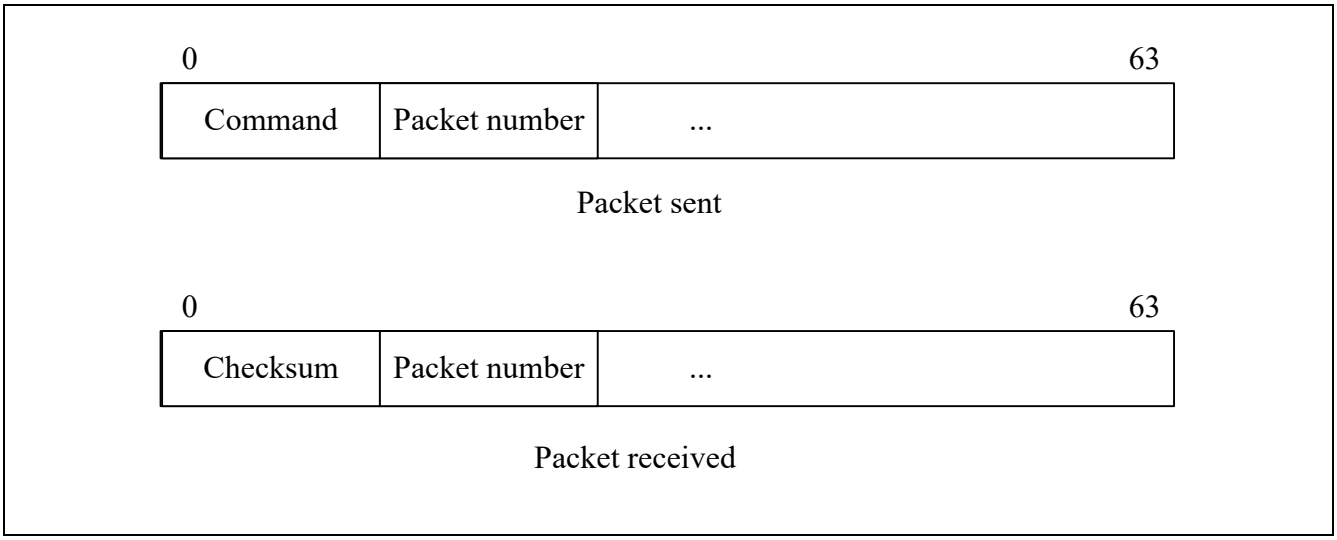

Figure 1-1 ISP Packet Sent and Received

1.1.1 ISP Command Set

The ISP supported commands are listed below.

Command	Command Code	ACK	Command Description
CMD_ERASE_SPI_FLASH	0x000000D0	Yes	Start erasing 64 KB SPI Flash from the offset address (default is 0) until all blocks are erased.
CMD_UPDATE_SPI_FLASH	0x000000D1	Yes	Program SPI Flash from the address specified by user.
CMD_GET_SPI_JEDEC	0x000000D2	Yes	Get JEDEC ID of SPI Flash.

Table 1-1 ISP Command Set

CMD_ERASE_SPI_FLASH Command Format

Host Send (unit by byte):

4	4	4	4	n
CMD	Packet Number	Start Address	Data	Data……

Device ACK:

4	4	
Checksum	Packet Number	Total Checksum

This command is used to instruct ISP to erase SPI Flash. The user can specify start address to be erased. When ISP receives the command packet, it will send the block erase instruction of SPI Flash to erase 64 KB SPI Flash. After the erase operation is completed, it calculates the checksum of the whole packet received, then transmits ACK packet to the host.

After ACK packet is received, the host checks whether the checksum is correct, if checksum is correct, it indicates that the data transfer is successful. The host will continue sending the erase command to ISP until the entire SPI Flash is erased.

CMD_UPDATE_SPI_FLASH Command Format

Host Send:

4	4	4	4	n
CMD	Packet Number	Start Address	Length	Data……

Device ACK:

4	4	
Checksum	Packet Number	Total Checksum

This command is used to instruct ISP to program SPI Flash. The user can specify the start address to be programmed. If the program data is too big to be transferred in a packet, ISP Programming Tool will split the data into several packets by multiple transfers.

ISP reads the program data from the received packet, and programs them into the specified address of SPI Flash. After program operation is completed, it calculates the checksum of the whole packet received, then transmits ACK packet to the host.

After the ACK packet is received, the host will check whether the checksum is correct, if checksum is correct, it indicates that the data transfer is successful.

CMD_GET_SPI_JEDEC Command Format

Host Send:

4	4
CMD	Packet Number

Device ACK:

4	4	4
Checksum	Packet Number	JEDEC ID

This command is used to instruct ISP to read JEDEC ID information of SPI Flash and transmit them to the host.

1.1.2 ISP Updating SPI Flash Flow

The procedure for updating SPI Flash is as follows:

1. Use a Keil project to open the **Chip Settings** dialog and set the **Chip Booting Selection** option to **LDROM** and click **OK** as shown in Figure 1-2.

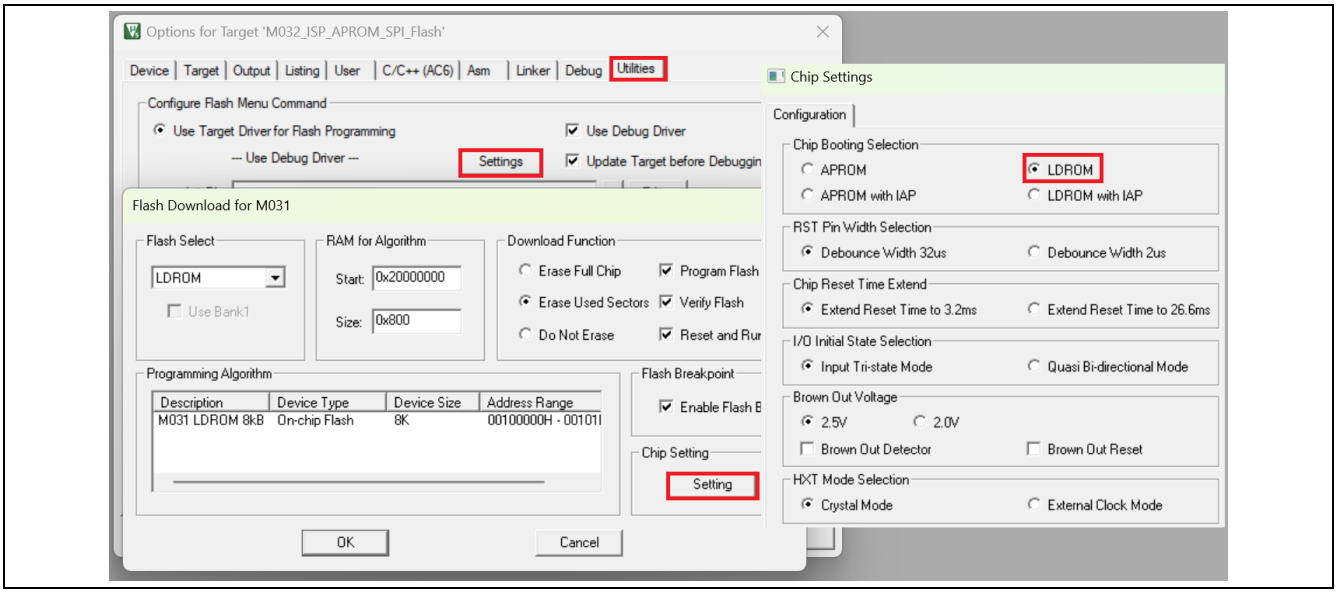

Figure 1-2 Boot from LDROM Setting

2. Build and download code to LDROM of the target chip.
3. Add the information of SPI Flash to the *SPIFLASH ID.ini* file as shown in Figure 1-3. For example, **EF4016** is the JEDEC ID of Winbond W25Q32JV SPI Flash, and **4M** indicates that its storage capacity is 4 MB. This step allows the ISP Programming Tool to display the correct capacity of the SPI Flash when connected to the target chip.

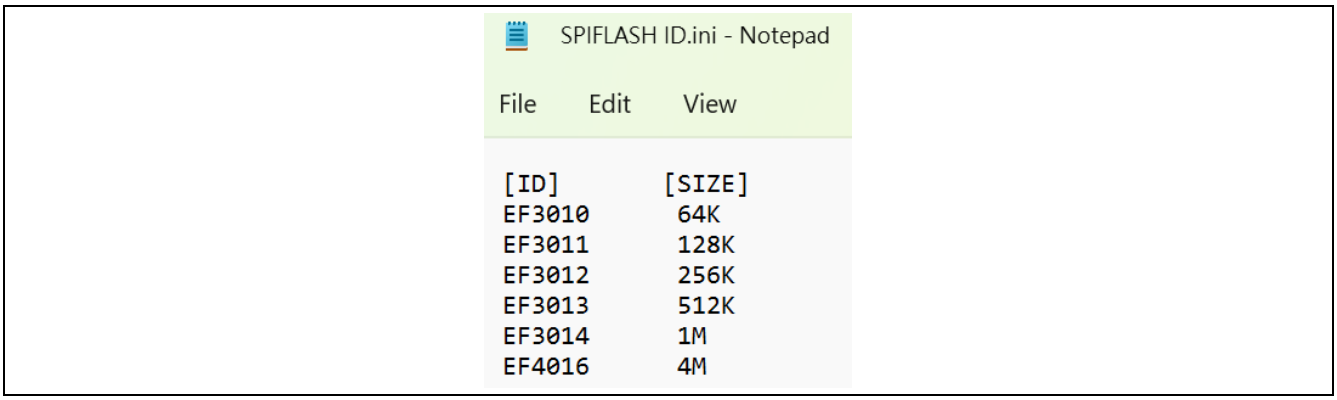

Figure 1-3 Information File of SPI Flash

4. Open the ISP Programming Tool and select UART interface to connect.

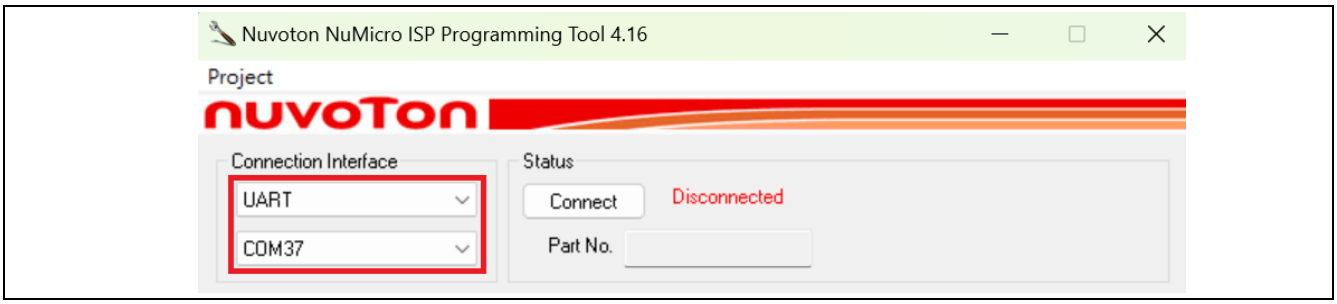

Figure 1-4 Setting Connection Interface of ISP Tool

5. Click the **Project** button and select the **Open SPI Flash** option, as shown in Figure 1-5.

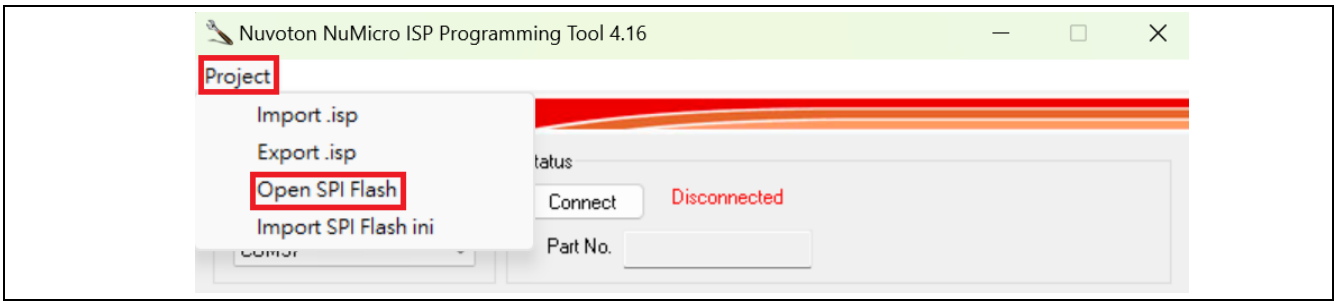

Figure 1-5 Open SPI Flash Option

6. Click the **Project** button to import SPI Flash ini file, as shown in Figure 1-6.

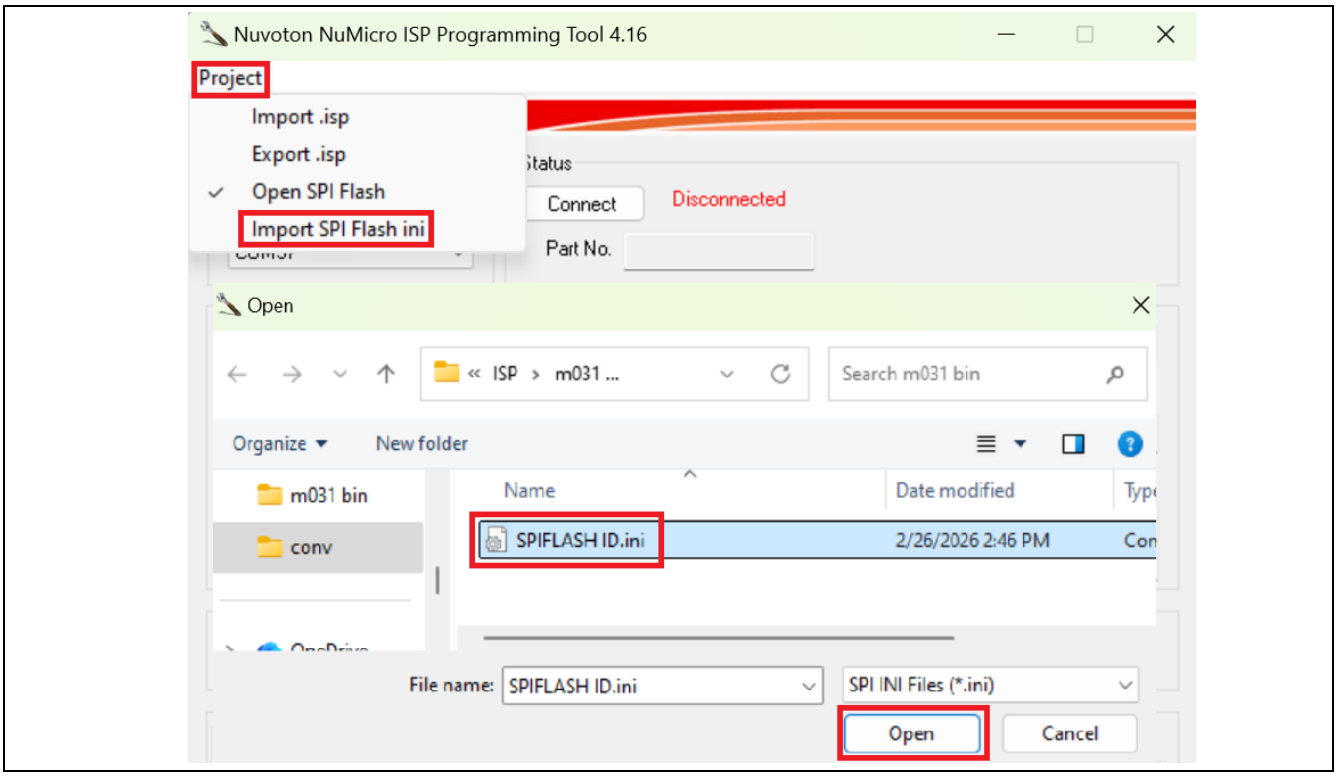

Figure 1-6 Import SPI Flash Information File

7. Click the **Connect** button and reset the target chip to run ISP code. The ISP Programming Tool will connect to the target chip.

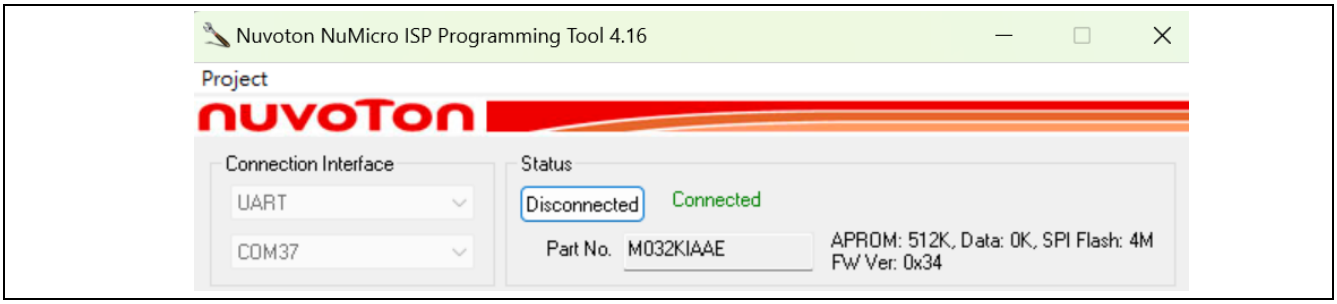

Figure 1-7 Connect to Target Chip with UART Interface

8. Load the programming file, select the programming option **SPI Flash** and click the **Start** button. The ISP Programming Tool starts to program the file to the base address specified by user, as shown in Figure 1-8. Since the ISP Programming Tool does not issue block erase command before programming data, if the user is unsure whether the address of SPI Flash has been programmed, please select the **Erase SPI Flash** option to erase the entire SPI Flash before programming to ensure that data can be written to SPI Flash correctly. Please also note that the **Base Address** option has a 2 KB alignment limitation.

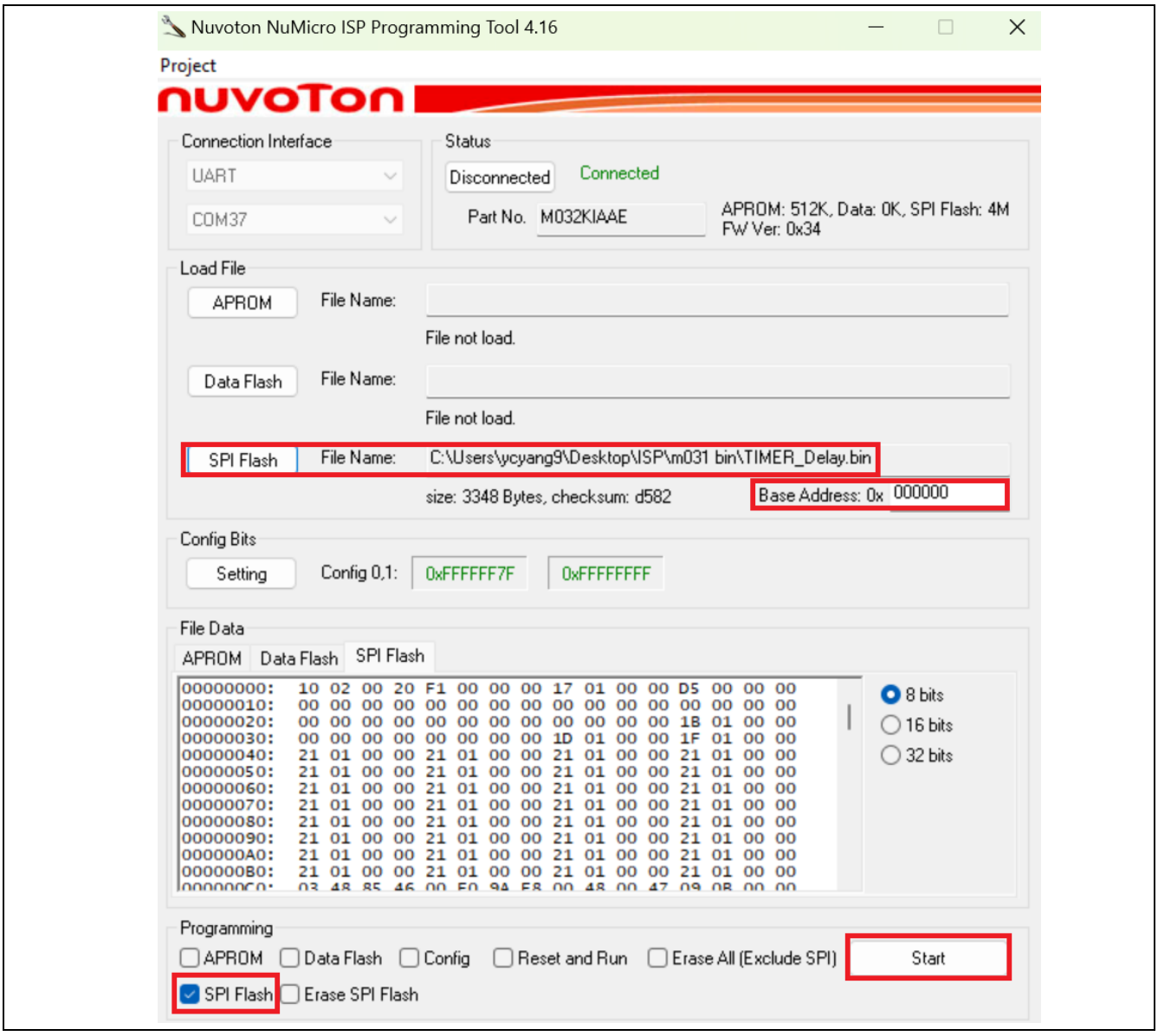

Figure 1-8 Program Data to External SPI Flash with UART Interface

1.2 Execution Result

The demo result is shown in Figure 1-9.

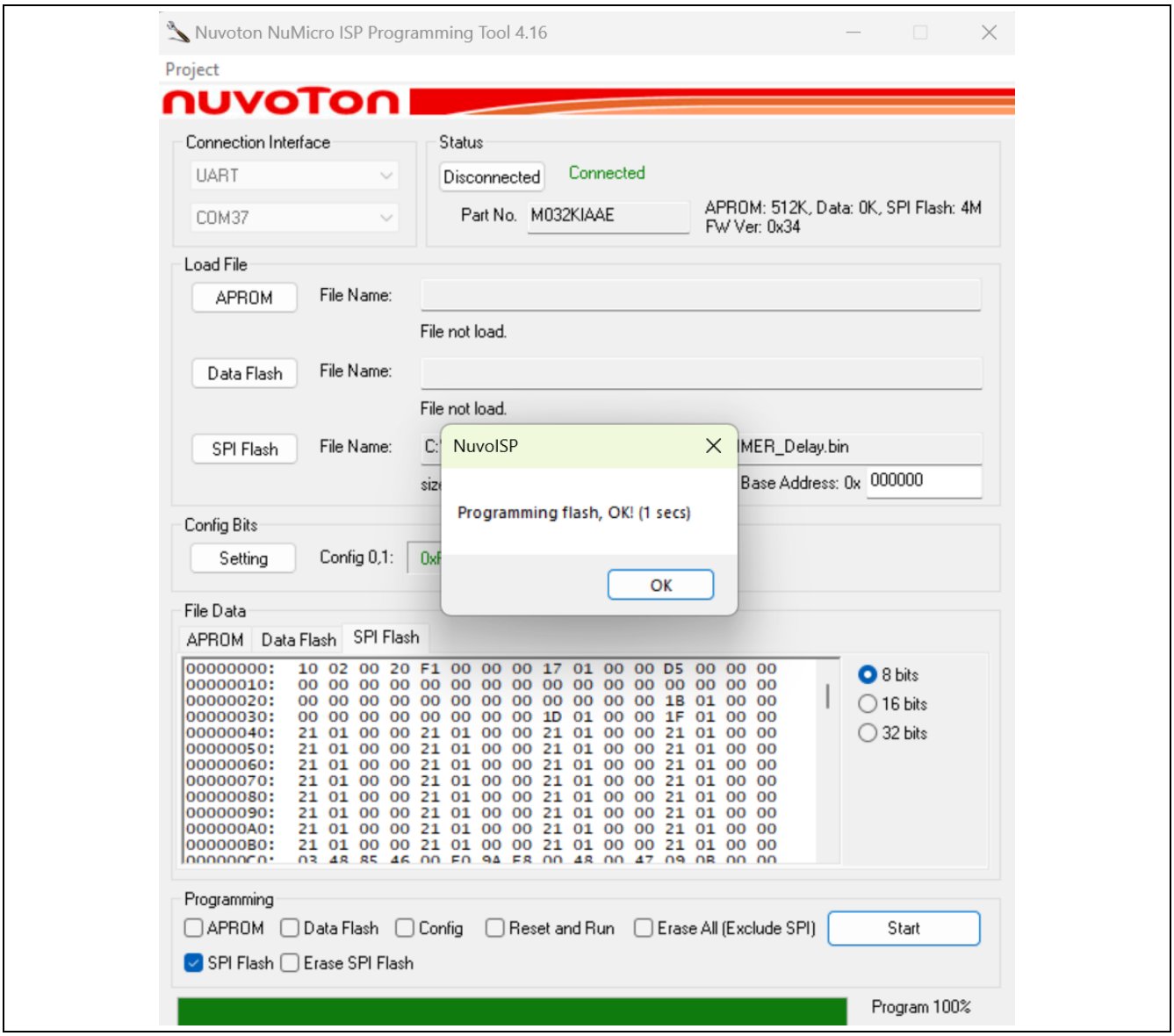

Figure 1-9 Programming Results

2. Code Description

The ISP Bootloader's main function sets the system clock and initializes the UART and SPI function. It uses UART to communicate with the ISP Programming Tool for updates. If the user does not need to update APROM or SPI Flash, the code will configure the chip to boot from APROM and execute system reset.

```
int32_t main(void)
{
    /* Init System, peripheral clock and multi-function I/O */
    SYS_Init();

    /* Init UART to 115200-8n1 */
    UART_Init();

    /* Configure SPI_FLASH_PORT as a master, MSB first, 8-bit transaction, SPI Mode-0
    timing, clock is 2MHz */
    SPI_Open(SPI_FLASH_PORT, SPI_MASTER, SPI_MODE_0, 8, 2000000);

    /* Disable auto SS function, control SS signal manually. */
    SPI_DisableAutoSS(SPI_FLASH_PORT);

    /* Checking if flash page size matches with target chip's */
    if ((GET_CHIP_SERIES_NUM == CHIP_SERIES_NUM_I) || (GET_CHIP_SERIES_NUM ==
    CHIP_SERIES_NUM_G))
    {
        if (FMC_FLASH_PAGE_SIZE != 2048)
        {
            /* FMC_FLASH_PAGE_SIZE is different from target device */
            /* Please enable the compiler option PAGE_SIZE_2048 in fmc.h */
            while(SYS->PDID);
        }
    }
    else
    {
        if (FMC_FLASH_PAGE_SIZE != 512)
        {
            /* FMC_FLASH_PAGE_SIZE is different from target device */
            /* Please disable the compiler option PAGE_SIZE_2048 in fmc.h */
            while(SYS->PDID);
        }
    }

    CLK->AHBCLK |= CLK_AHBCLK_ISPCKEN_Msk;

    /* Unlock protected registers */
    SYS_UnlockReg();

    /* Enable FMC ISP function. Before using FMC function, it should unlock system
    register first. */
    FMC->ISPCTL |= (FMC_ISPCTL_ISPEN_Msk | 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;

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

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

        if (SysTick->CTRL & SysTick_CTRL_COUNTFLAG_Msk)
        {
            SYS->RSTSTS = (SYS_RSTSTS_PORF_Msk | SYS_RSTSTS_PINRF_Msk);
            FMC->ISPCTL &= ~(FMC_ISPCTL_ISPEN_Msk | FMC_ISPCTL_BS_Msk);
            SCB->AIRCR = (V6M_AIRCR_VECTKEY_DATA | V6M_AIRCR_SYSRESETREQ);

            /* Trap the CPU */
            while (1);
        }
    }

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

This function parses the command from the ISP Programming Tool to complete the update flow.

```
int ParseCmd(unsigned char *buffer, uint8_t len)
{
    static uint32_t StartAddress, TotalLen, LastDataLen;
    uint32_t lcmd, srclen, i, regcnf0, security;
    unsigned char *pSrc;
    static uint32_t gcmd;
    uint32_t u32Addr, u32NTx, u32Id;
```

```
    g_pu8Response = response_buff;
    pSrc = buffer;
    srclen = len;
    lcmd = inpw(pSrc);
    outpw(g_pu8Response + 4, 0);
    pSrc += 8;
    srclen -= 8;
    ReadData(Config0, Config0 + 16, (uint32_t *)(g_pu8Response + 8)); /* Read config */
    regcnf0 = *(uint32_t *)(g_pu8Response + 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)
    {
        g_pu8Response[8] = FW_VERSION;
    }
    else if (lcmd == CMD_GET_DEVICEID)
    {
        outpw(g_pu8Response + 8, SYS->PDID);

        ExitParseCmd(buffer, len);
        return (0);
    }
    else if (lcmd == CMD_RUN_APROM || lcmd == CMD_RUN_LDROM || lcmd == CMD_RESET)
    {
        SYS->RSTSTS = 3; /* Clear bit */

        /* Set BS */
        if (lcmd == CMD_RUN_APROM)
        {
            i = (FMC->ISPCTL & 0xFFFFFFFC);
        }
        else if (lcmd == CMD_RUN_LDROM)
        {
            i = (FMC->ISPCTL & 0xFFFFFFFC);
            i |= 0x00000002;
        }
        else
        {
            i = (FMC->ISPCTL & 0xFFFFFFFE);/* ISP disable */
        }

        FMC->ISPCTL = i;
        SCB->AIRCR = (V6M_AIRCR_VECTKEY_DATA | V6M_AIRCR_SYSRESETREQ);

        /* Trap the CPU */
        while (1);
```

```
    }
    else if (lcmd == CMD_CONNECT)
    {
        g_packno = 1;

        ExitParseCmd(buffer, len);
        return (0);
    }
    else if ((lcmd == CMD_UPDATE_APROM) || (lcmd == CMD_ERASE_ALL))
    {
        EraseAP(FMC_APROM_BASE, (g_apromSize < g_dataFlashAddr) ? g_apromSize :
        g_dataFlashAddr); /* Erase APROM */

        if (lcmd == CMD_ERASE_ALL)
        {
            EraseAP(g_dataFlashAddr, g_dataFlashSize);
            *(uint32_t *)(g_pu8Response + 8) = regcnf0 | 0x02;
            UpdateConfig((uint32_t *)(g_pu8Response + 8), NULL);
        }

        bUpdateApromCmd = TRUE;
    }

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

            if (g_dataFlashSize)   /*g_dataFlashAddr*/
            {
                EraseAP(g_dataFlashAddr, g_dataFlashSize);
            }
            else
            {
                ExitParseCmd(buffer, len);
                return (0);
            }
        }
        else
        {
            StartAddress = 0;
        }

        TotalLen = inpw(pSrc + 4);
        pSrc += 8;
        srclen -= 8;
    }
    else if (lcmd == CMD_UPDATE_CONFIG)
    {
        if ((security == 0) && (!bUpdateApromCmd))   /* Security lock */
        {
            ExitParseCmd(buffer, len);
            return (0);
        }
```

```
        UpdateConfig((uint32_t *)(pSrc), (uint32_t *)(g_pu8Response + 8));
        GetDataFlashInfo(&g_dataFlashAddr, &g_dataFlashSize);

        ExitParseCmd(buffer, len);
        return (0);
    }
    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)
        {
            ExitParseCmd(buffer, len);
            return (0);
        }

        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);
        }

        ExitParseCmd(buffer, len);
        return (0);
    }
    else if (lcmd == CMD_GET_JEDECID)
    {
        u32Id = SpiFlash_ReadJEDECID();
        outpw(g_pu8Response + 8, u32Id);

        ExitParseCmd(buffer, len);
        return (0);
    }
    else if (lcmd == CMD_ERASE_SPIFLASH)
    {
        u32Addr = inpw(pSrc);
        SpiFlash_EraseBlock(u32Addr);
    }
    else if (lcmd == CMD_UPDATE_SPIFLASH)
    {
        u32Addr = inpw(pSrc);
        u32NTx = inpw(pSrc + 4);
        pSrc += 8;
        SpiFlash_NormalPageProgram(u32Addr, u32NTx, pSrc);
        memset(pSrc, 0, u32NTx);
        SpiFlash_NormalRead(u32Addr, u32NTx, pSrc);
    }

    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;
    }

    ExitParseCmd(buffer, len);
    return (0);
}
```

This part of ParseCmd() function is used for SPI Flash update and returns the JEDEC ID to the ISP Programming Tool.

```
    else if (lcmd == CMD_GET_JEDECID)
    {
        u32Id = SpiFlash_ReadJEDECID();
        outpw(g_pu8Response + 8, u32Id);

        ExitParseCmd(buffer, len);
        return (0);
    }
    else if (lcmd == CMD_ERASE_SPIFLASH)
    {
        u32Addr = inpw(pSrc);
        SpiFlash_EraseBlock(u32Addr);
    }
    else if (lcmd == CMD_UPDATE_SPIFLASH)
    {
        u32Addr = inpw(pSrc);
        u32NTx = inpw(pSrc + 4);
        pSrc += 8;
        SpiFlash_NormalPageProgram(u32Addr, u32NTx, pSrc);
        memset(pSrc, 0, u32NTx);
        SpiFlash_NormalRead(u32Addr, u32NTx, pSrc);
    }
```

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

```
    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;
    }
```

3. Software and Hardware Requirements

3.1 Software Requirements

- BSP version
  - M031_Series_BSP_CMSIS_V3.07.000
- IDE version
  - Keil uVision 5.38

3.2 Hardware Requirements

- Circuit components
  - NuMaker-M032KI V1.0
  - Level1-Training
- Pin Connect
  - The USB interface of the Nu-Link2-Me is connected to the USB port of a PC to show the execution results of this example code.

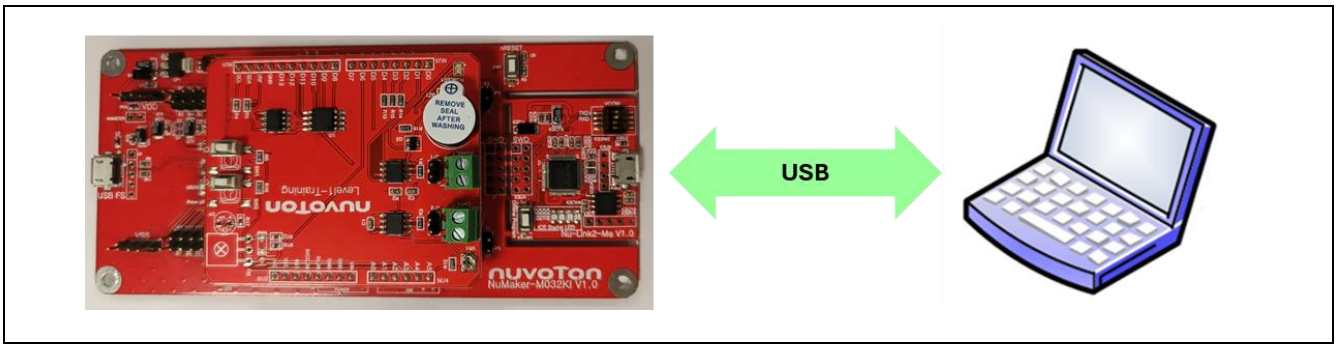

Figure 3-1 Pin Connect

4. Directory Information

The directory structure is shown below.

EC_M032_ISP_Update_APROM_and_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	
Project	Source file of example code
WindowsTool	ISP Programming Tool

Figure 4-1 Directory Structure

5. Example Code Execution

5.1 Keil MDK

1. Navigate to the **Keil** project folder in Windows Explorer and open the *M032_ISP_Update_APROM_and_SPI_Flash.uvprojx* project file.
2. Build
3. Download
4. Enter debug mode
   - Run

5.2 VS Code

1. Select **File** in VS Code IDE
   - Click **Open Folder**
   - Select the **VSCode** folder under the project directory
2. Select **CMSIS**
   - Click **Build solution** to compile the project
   - Click **Load & Run application** to program the target chip
   - Click **Load & Debug application** to enter debug mode

6. Revision History

Date	Revision	Description
2026.06.23	1.00	Initial version.

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.