USB to SPI Bridge

Example Code Introduction for 32-bit NuMicro® Family

Document Information

Application	This example code uses USB Device HID class to transfer data to SPI Flash and simulate an USB to SPI bridge.
BSP Version	M480_Series_BSP_CMSIS_V3.05.001
Hardware	NuMaker-PFM-M487 V3.0

1. Overview

This example is mainly to simulate the bridge from USB to SPI. Connect the PC to execute the application program through the USB, and use the application program to transmit data to the SPI device. This example code uses the USB HID transfer and SPI Flash sample in the BSP to realize the simulation. The PC-side application program can read and write SPI Flash through USB.

This example provides a simple "*HIDTransferTest.exe*" to test the USB to SPI bridge. The PC will issue erase/ read/ program commands to the M480 through the USB, and then the SPI Flash will perform the corresponding erase/ read/ program actions. The schematic diagram is shown in Figure 1-1.

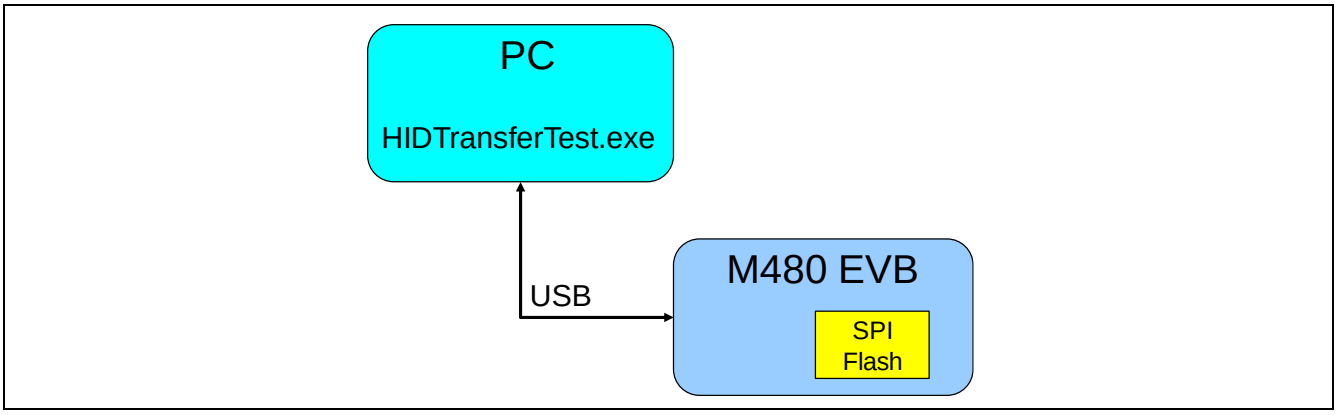

Figure 1-1 Relationship of PC and M480 Evaluation Board (EVB)

1.1 Demo Result

The sector size of this SPI Flash is 4096 bytes. The page size of this SPI Flash is 2048 bytes. The *HIDTransferTest.exe* will erase two sectors, read back erased data and check whether the data is 0xFF or not; program four pages, read back programmed data and check whether the data is correct or not; erase one sector, and read back four-page data and check whether the data is correct or not. The test result is as follows.

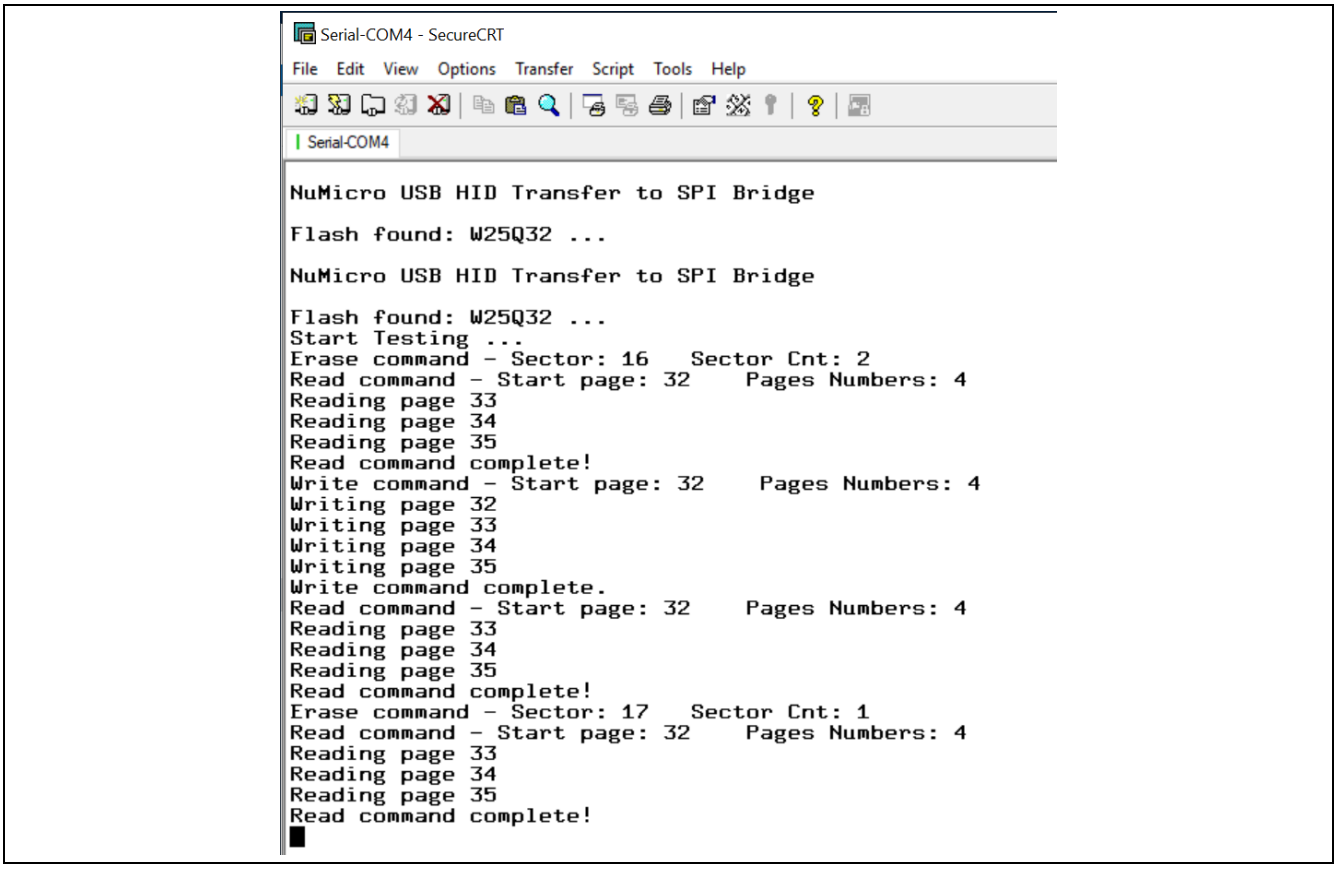

Figure 1-2 Test Results

2. Code Description

SPI relative functions, which include Read, Program, Erase, Read Status, and Wait ready functions. The code is located in *main.c*.

```
uint8_t SpiFlash_ReadStatusReg(void)
{
    // /CS: active
    QSPI_SET_SS_LOW(QSPI_FLASH_PORT);

    // send Command: 0x05, Read status register
    QSPI_WRITE_TX(QSPI_FLASH_PORT, 0x05);

    // read status
    QSPI_WRITE_TX(QSPI_FLASH_PORT, 0x00);

    // wait tx finish
    while(QSPI_IS_BUSY(QSPI_FLASH_PORT));

    // /CS: de-active
    QSPI_SET_SS_HIGH(QSPI_FLASH_PORT);

    // skip first rx data
    QSPI_READ_RX(QSPI_FLASH_PORT);

    return (QSPI_READ_RX(QSPI_FLASH_PORT) & 0xff);
}

void SpiFlash_WaitReady(void)
{
    uint8_t volatile ReturnValue;

    do
    {
        ReturnValue = SpiFlash_ReadStatusReg();
        ReturnValue = ReturnValue & 1;
    }
    while(ReturnValue!=0);   // check the BUSY bit
}

void SpiFlash_NormalRead(uint32_t StartAddress, uint8_t *u8DataBuffer)
{
    uint32_t i;

    // /CS: active
    QSPI_SET_SS_LOW(QSPI_FLASH_PORT);

    // send Command: 0x03, Read data
    QSPI_WRITE_TX(QSPI_FLASH_PORT, 0x03);

    // send 24-bit start address
    QSPI_WRITE_TX(QSPI_FLASH_PORT, (StartAddress>>16) & 0xFF);
    QSPI_WRITE_TX(QSPI_FLASH_PORT, (StartAddress>>8)  & 0xFF);
    QSPI_WRITE_TX(QSPI_FLASH_PORT, StartAddress       & 0xFF);
```

```
    while(QSPI_IS_BUSY(QSPI_FLASH_PORT));
    // clear RX buffer
    QSPI_ClearRxFIFO(QSPI_FLASH_PORT);

    // read data
    for(i=0; i<256; i++)
    {
        QSPI_WRITE_TX(QSPI_FLASH_PORT, 0x00);
        while(QSPI_IS_BUSY(QSPI_FLASH_PORT));
        u8DataBuffer[i] = QSPI_READ_RX(QSPI_FLASH_PORT);
    }

    // wait tx finish
    while(QSPI_IS_BUSY(QSPI_FLASH_PORT));

    // /CS: de-active
    QSPI_SET_SS_HIGH(QSPI_FLASH_PORT);
}

void SpiFlash_NormalPageProgram(uint32_t StartAddress, uint8_t *u8DataBuffer)
{
    uint32_t i = 0;

    // /CS: active
    QSPI_SET_SS_LOW(QSPI_FLASH_PORT);

    // send Command: 0x06, Write enable
    QSPI_WRITE_TX(QSPI_FLASH_PORT, 0x06);

    // wait tx finish
    while(QSPI_IS_BUSY(QSPI_FLASH_PORT));

    // /CS: de-active
    QSPI_SET_SS_HIGH(QSPI_FLASH_PORT);


    // /CS: active
    QSPI_SET_SS_LOW(QSPI_FLASH_PORT);

    // send Command: 0x02, Page program
    QSPI_WRITE_TX(QSPI_FLASH_PORT, 0x02);

    // send 24-bit start address
    QSPI_WRITE_TX(QSPI_FLASH_PORT, (StartAddress>>16) & 0xFF);
    QSPI_WRITE_TX(QSPI_FLASH_PORT, (StartAddress>>8)  & 0xFF);
    QSPI_WRITE_TX(QSPI_FLASH_PORT, StartAddress       & 0xFF);

    // write data
    while(1)
    {
        if(!QSPI_GET_TX_FIFO_FULL_FLAG(QSPI_FLASH_PORT))
        {
            QSPI_WRITE_TX(QSPI_FLASH_PORT, u8DataBuffer[i++]);
            if(i > 255) break;
        }
```

```
    }

    // wait tx finish
    while(QSPI_IS_BUSY(QSPI_FLASH_PORT));

    // /CS: de-active
    QSPI_SET_SS_HIGH(QSPI_FLASH_PORT);

    QSPI_ClearRxFIFO(QSPI_FLASH_PORT);
}

/* one sector is 4KB */
void SpiFlash_SecotrErase(uint32_t StartSector, uint32_t EraseCount)
{
    uint32_t volatile i, offset;

    for (i=0; i<EraseCount; i++)
    {
        offset = (StartSector + i) * 0x1000;
        // /CS: active
        QSPI_SET_SS_LOW(QSPI_FLASH_PORT);

        // send Command: 0x06, Write enable
        QSPI_WRITE_TX(QSPI_FLASH_PORT, 0x06);

        // wait tx finish
        while(QSPI_IS_BUSY(QSPI_FLASH_PORT));

        // /CS: de-active
        QSPI_SET_SS_HIGH(QSPI_FLASH_PORT);

        // /CS: active
        QSPI_SET_SS_LOW(QSPI_FLASH_PORT);

        // send Command: 0x20, sector erase
        QSPI_WRITE_TX(QSPI_FLASH_PORT, 0x20);

        // send 24-bit start address
        QSPI_WRITE_TX(QSPI_FLASH_PORT, (offset>>16) & 0xFF);
        QSPI_WRITE_TX(QSPI_FLASH_PORT, (offset>>8)  & 0xFF);
        QSPI_WRITE_TX(QSPI_FLASH_PORT, offset       & 0xFF);

        // wait tx finish
        while(QSPI_IS_BUSY(QSPI_FLASH_PORT));

        // /CS: de-active
        QSPI_SET_SS_HIGH(QSPI_FLASH_PORT);

        QSPI_ClearRxFIFO(QSPI_FLASH_PORT);
    }
}
```

After initializing the system, configure the QSPI first, and then set up the USB device. The code is located in *main.c*.

```
int32_t main(void)
{
    uint32_t u32TrimInit;
    uint16_t u16ID;

    /* Unlock protected registers */
    SYS_UnlockReg();

    SYS_Init();

    /* Configure UART0 and set UART0 Baudrate */
    UART_Open(UART0, 115200);

    /* Configure QSPI_FLASH_PORT as a master, MSB first, 8-bit transaction, QSPI Mode-0
timing, clock is 2MHz */
    QSPI_Open(QSPI_FLASH_PORT, QSPI_MASTER, QSPI_MODE_0, 8, 2000000);

    /* Enable the automatic hardware slave select function. Select the SS pin and
configure as low-active. */
    QSPI_EnableAutoSS(QSPI_FLASH_PORT, QSPI_SS, QSPI_SS_ACTIVE_LOW);

    printf("\nNuMicro USB HID Transfer to SPI Bridge\n\n");

    /* Read SPI Flash ID : W25Q32(0xEF15), W25Q16(0xEF14) */
    if((u16ID = SpiFlash_ReadMidDid()) != 0xEF15)
    {
        printf("Wrong ID, 0x%x\n", u16ID);
        while(1);
    }
    else
        printf("Flash found: W25Q32 ...\n");

    USBD_Open(&gsInfo, HID_ClassRequest, NULL);

    printf("Start Testing ...\n");
    /* Endpoint configuration */
    HID_Init();
    USBD_Start();

    ……

    NVIC_EnableIRQ(USBD_IRQn);

    while(1)
    {
    ……
    }
}
```

Define the command and structure for USB HID Transfer. The code is located in *hid_to_spi_bridge.c*.

```
/* HID Transfer Commands */
#define HID_CMD_NONE     0x00
#define HID_CMD_ERASE    0x71
#define HID_CMD_READ     0xD2
#define HID_CMD_WRITE    0xC3
#define HID_CMD_TEST     0xB4

#define PAGE_SIZE        2048
#define TEST_PAGES       4
#define SECTOR_SIZE      4096

typedef struct __attribute__((__packed__))
{
    uint8_t u8Cmd;
    uint8_t u8Size;
    uint32_t u32Arg1;
    uint32_t u32Arg2;
    uint32_t u32Signature;
    uint32_t u32Checksum;
}
CMD_T;
```

Define the corresponding functions of USB HID Transfer. The code is located in *hid_to_spi_bridge.c*.

```
int32_t HID_CmdEraseSectors(CMD_T *pCmd)
{
    uint32_t u32StartSector;
    uint32_t u32Sectors;

    u32StartSector = pCmd->u32Arg1;
    u32Sectors = pCmd->u32Arg2;

    printf("Erase command - Sector: %d   Sector Cnt: %d\n", u32StartSector, u32Sectors);

    /* To erase the sector of storage */
    SpiFlash_SecotrErase(u32StartSector, u32Sectors);
    SpiFlash_WaitReady();

    /* To note the command has been done */
    pCmd->u8Cmd = HID_CMD_NONE;

    return 0;
}

int32_t HID_CmdReadPages(CMD_T *pCmd)
{
    uint32_t u32StartPage;
    uint32_t u32Pages;
    uint32_t volatile i, offset;

    u32StartPage = pCmd->u32Arg1;
    u32Pages     = pCmd->u32Arg2;
```

```
    printf("Read command - Start page: %d    Pages Numbers: %d\n", u32StartPage,
u32Pages);

    if(u32Pages)
    {
        /* Update data to page buffer to upload */
        offset = u32StartPage * PAGE_SIZE;
        for (i=0; i<PAGE_SIZE/256; i++)
            SpiFlash_NormalRead(offset+i*256, g_u8PageBuff+i*256);
        g_u32BytesInPageBuf = PAGE_SIZE;

        /* The signature word is used as page counter */
        pCmd->u32Signature = 1;

        /* Trigger HID IN */
        USBD_MemCopy((uint8_t *)(USBD_BUF_BASE + USBD_GET_EP_BUF_ADDR(EP2)), (void
*)g_u8PageBuff, EP2_MAX_PKT_SIZE);
        USBD_SET_PAYLOAD_LEN(EP2, EP2_MAX_PKT_SIZE);
        g_u32BytesInPageBuf -= EP2_MAX_PKT_SIZE;
    }

    return 0;
}

int32_t HID_CmdWritePages(CMD_T *pCmd)
{
    uint32_t u32StartPage;
    uint32_t u32Pages;

    u32StartPage = pCmd->u32Arg1;
    u32Pages     = pCmd->u32Arg2;

    printf("Write command - Start page: %d    Pages Numbers: %d\n", u32StartPage,
u32Pages);
    g_u32BytesInPageBuf = 0;

    /* The signature is used to page counter */
    pCmd->u32Signature = 0;

    return 0;
}
```

3. Software and Hardware Requirements

3.1 Software Requirements

- BSP version
  - M480_Series_BSP_CMSIS_V3.05.001
- IDE version
  - Keil uVersion 5.28

3.2 Hardware Requirements

- Circuit components
  - NuMaker-PFM-M487 V3.0
- Pin Connect
  - Connect the UART0 TX (PB.13) pin to the PC UART RX to show the execution results of this example code.
  - Connect the USB to the PC.

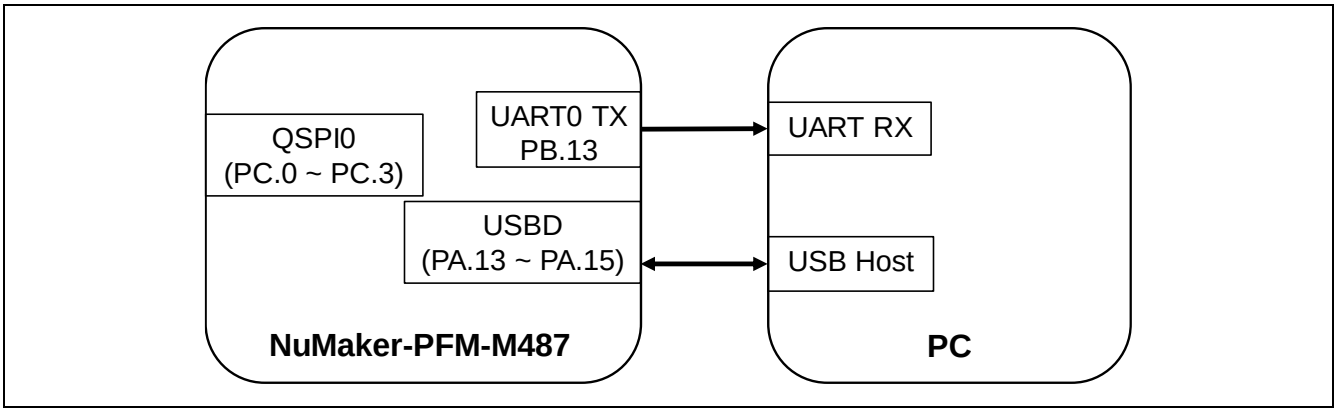

Figure 3-1 Pin Connect

4. Directory Information

The directory structure is shown below.

EC_M480_USBD_HID_to_SPI_Bridge_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 *USBD_HID_to_SPI_bridge.uvproj*.
2. Enter Keil compile mode.
   - Build
   - Download
   - Start/Stop debug session
3. Enter debug mode.
   - Run

6. Revision History

Date	Revision	Description
2023.08.16	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.