USB 转 SPI 桥接器

NuMicro® 32 位系列微控制器范例代码介绍

文件信息

应用简述	本范例代码使用 USB HID 将数据传输到 SPI Flash。模拟 USB 转 SPI 桥接器。
BSP 版本	M480_Series_BSP_CMSIS_V3.05.001
开发平台	NuMaker-PFM-M487 V3.0

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. 概述

本范例主要是仿真 USB 到 SPI 的桥接。 通过 USB 连接 PC 执行应用程序，并使用应用程序向 SPI 设备传输数据。 本范例使用 BSP 中的 USB HID Transfer 和 SPI Flash 范例来实现此仿真。 PC 端应用程序可以通过 USB 读写 SPI Flash。

本范例提供了一个简单的应用程序 HIDTransferTest.exe 来测试 USB 转 SPI 桥接器。 PC 通过 USB 向 M480 发出擦除/ 读取/ 写入命令，然后 SPI Flash 执行相应的擦除/ 读取/ 写入动作。 示意图如图 1-1 所示。

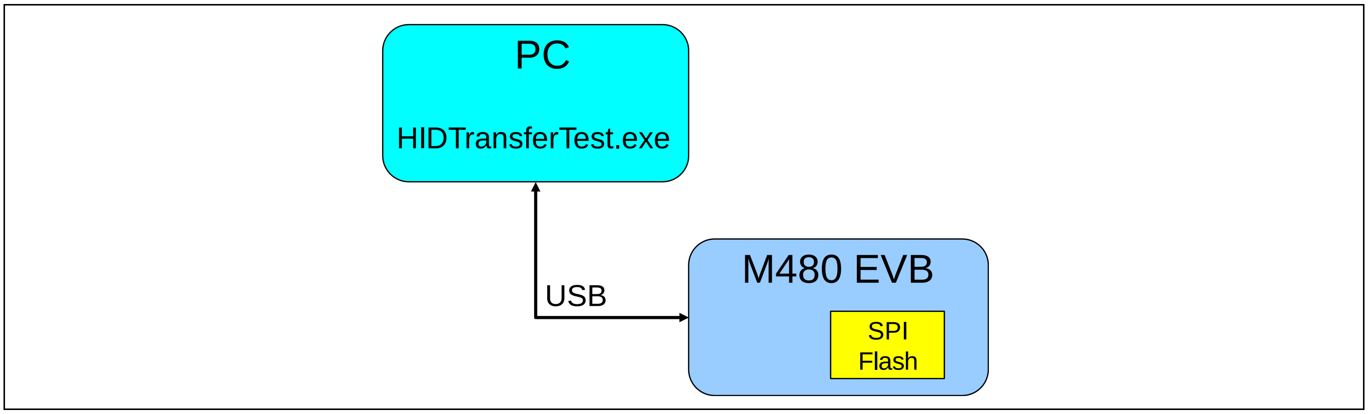
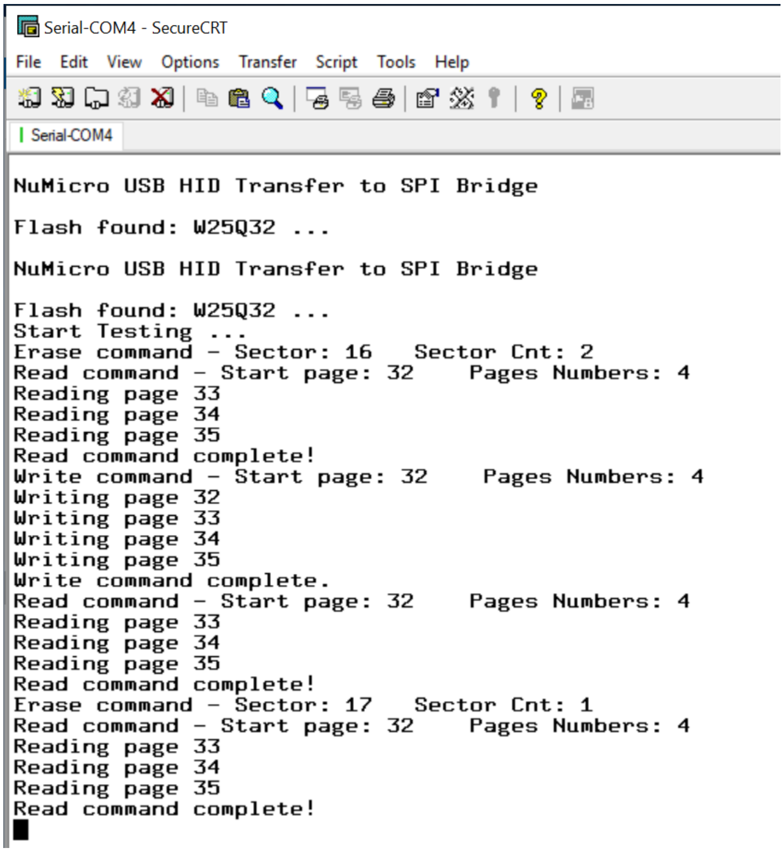


图 1-1 PC 与 M480 EVB 的关系

1.1 执行结果

这个 SPI Flash 的扇区大小为 4096 字节，而且页的大小为 2048 字节。 HIDTransferTest.exe 会擦除两个扇区，读回擦除的数据并检查是否为 0xFF；写入四页，读回写入数据并检查是否正确；擦除一个扇区，读回四页数据并检查是否正确。测试结果如下。



```
NuMicro USB HID Transfer to SPI Bridge

Flash found: W25Q32 ...

NuMicro USB HID Transfer to SPI Bridge

Flash found: W25Q32 ...
Start Testing ...
Erase command - Sector: 16   Sector Cnt: 2
Read command - Start page: 32    Pages Numbers: 4
Reading page 33
Reading page 34
Reading page 35
Read command complete!
Write command - Start page: 32    Pages Numbers: 4
Writing page 32
Writing page 33
Writing page 34
Writing page 35
Write command complete.
Read command - Start page: 32    Pages Numbers: 4
Reading page 33
Reading page 34
Reading page 35
Read command complete!
Erase command - Sector: 17   Sector Cnt: 1
Read command - Start page: 32    Pages Numbers: 4
Reading page 33
Reading page 34
Reading page 35
Read command complete!
█
```

图 1-2 执行结果

2. 代码介绍

SPI 相关函数。它包括读取、写入、擦除、读取状态和等待就绪功能。代码位于 *main.c* 中。

```
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);
    }
}
```

初始化系统后，首先配置 QSPI，然后设置 USB 设备。 代码位于 *main.c* 中。

```
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)
    {
    ………
    }
}
```

定义 USB HID 传输的命令和结构。代码位于 *hid_to_spi_bridge.c* 中。

```
/* 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;
```

定义 USB HID Transfer 对应的函数。代码位于 *hid_to_spi_bridge.c* 中。

```
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. 软件与硬件需求

3.1 软件需求

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

3.2 硬件需求

- 电路组件
  - NuMaker-PFM-M487 V3.0
- 线路示意图
  - 将 UART0 TX (PB.13) pin 脚连接到 PC UART RX，以显示范例代码的执行结果。
  - 将 USB 连接到 PC。

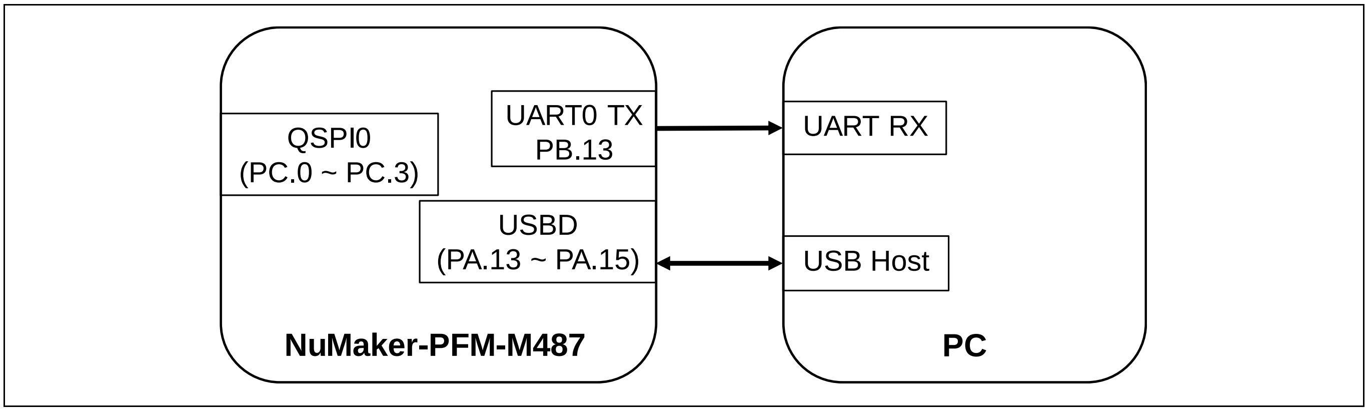


图 3-1 线路示意图

4. 目录信息

🗁 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

图 4-1 目录信息

5. 范例程序执行

1. 根据目录信息章节进入 ExampleCode 路径中的 KEIL 文件夹，双击 *USBD_HID_to_SPI_bridge.uvproj*。
2. 进入编译模式界面
   - 编译
   - 下载代码至内存
   - 进入 / 离开仿真模式
3. 进入仿真模式界面
   - 执行代码

6. 修订纪录

Date	Revision	Description
2023.08.16	1.00	初始发布。

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.