5 SPI FLASH 读写实验

5.1. 背景介绍

本章我们将使用 Cortex-M3 自带的 SPI 来实现对外部 Flash（W25Q64）的读写，并将结果通过串口助手显示出来。

5.1.1. SPI 简介

SPI 是串行外围设备接口（Serial Peripheral Interface）的缩写。串行外围设备接口是一种高速全双工同步的通信总线，在芯片的管脚上只占用四根线，节约了芯片的管脚，同时为 PCB 的布局上节省空间，提供方便，正是出于这种简单易用的特性，越来越多的芯片集成了这种通信协议，比如存储器、温度传感器、压力传感器、模拟转换器、实时时钟、任何支持串行模式的 SD 卡。

SPI 通讯包括两条数据线（进、出）、一条同步时钟线和一条控制线。如下所示：

1. Master Out Slave In (mosi)—主设备输入数据到从设备的数据线。
2. Master In Slave Out (miso)—从设备输出数据到主设备的数据线。
3. Serial Clock (sclk)—主设备驱动从设备的同步时钟。
1. Slave Select (ss_n)—主设备驱动，用于选择从设备，置低时有效。

5.1.2. SPI 寄存器映射

下表 5- 1 显示了 SPI 寄存器的映射。需要注意的是，寄存器地址=基地址+基地址偏移，SPI0 的基地址为 0x4000_B000，SPI1 的基地址为 0x4000_C000。表中的内容只是简单的介绍了一下寄存器，更详细的内容请查看官方的文档。

表 5- 1 SPI 寄存器映射表

Name	Base offset	Type	Width	Reset value	Description
SSPCR0	0x00	RW	16	0x0000	[3:0]: Data Size Select [5:4]: Frame format [6]: SSPCLKOUT polarity, applicable to Motorola SPI frame format only [7]: SSPCLKOUT phase, applicable to Motorola SPI frame format only [15:8]: Serial clock rate. The value SCR is used to generate the transmit and receive bit rate of the PrimeCell SSP.
SSPCR1	0x04	RW	4	0x0	[0]: Loop back mode [1]: Synchronous serial port enable [2]: Master or slave mode select

					[3]: Slave-mode output disable [5:4]: data width select [6]: 0: Time-shareing receive and transmit; 1: simultaneous transmission and reception [15:7] Reserved, read unpredictable, should be written as0
SSPDR	0x08	RW	16	0x----	Transmit/Receive FIFO: Read: Receive FIFO. Write: Transmit FIFO.
SSPSR	0x0C	RO	5	0x03	[0]: Transmit FIFO empty, RO [1]: Transmit FIFO not full, RO [2]: Receive FIFO not empty, RO [3]: Receive FIFO full, RO [4]: PrimeCell SSP busy flag, RO [5]: SSP TX busy flag [15:6]: Reserved, read unpredictable, should be written as 0.
SSPCPSR	0x10	RW	8	0x00	[7:0]: Clock prescale divisor. [15:8]: Reserved, read unpredictable, should be written as 0.
SSPIMSC	0x14	RW	4	0x0	[0]: Receive overrun interrupt mask [1]: Receive timeout interrupt mask [2]: Receive FIFO interrupt mask [3]: Transmit FIFO interrupt mask [4]:0:Receive FIFO Amost Full condition interrupt is masked； 1: Receive FIFO Amost Full condition interrupt is not masked [15:5]: Reserved, read as zero, do not modify.
SSPRIS	0x18	RO	4	0x8	[0]:Gives the raw interrupt state, prior to masking, of the SSPRORINTR interrupt [1]:Gives the raw interrupt state, prior to masking, of the SSPRTINTR interrupt [2]:Gives the raw interrupt state, prior to masking, of the SSPRXINTR interrupt [3]:Gives the raw interrupt state, prior to masking, of the SSPTXINTR interrupt [4]:Gives the raw interrupt state, prior to masking, of the SSPRAFTINTR interrupt [15:5]: Reserved, read as zero, do not modify
SSPMIS	0x1C	RO	4	0x0	[0]:Gives the receive over run masked interrupt status, after masking, of the SSPRORINTR interrupt [1]:Gives the receive timeout masked interrupt state, after masking, of the SSPRTINTR interrupt [2]:Gives the receive FIFO masked interrupt state, after masking, of the SSPRXINTR interrupt [3]:Gives the transmit FIFO masked interrupt state, after masking, of the SSPTXINTR

					interrupt [4]:Gives the receive FIFO Amost Full Timeout masked interruptstatus, after masking, of the SSPRAFTINTR interrupt [15:5]: Reserved, read as zero, do not modify
SSPICR	0x20	WO	4	0x0	[0]: Clears the SSPRORINTR interrupt [1]: Clears the SSPRTINTR interrupt [15:2]: Reserved, read as zero, do not modify
SSPDMACR	0x24	RW	2	0x0	[0]: Receive DMA Enable. If this bit is set to 1, DMA for the receive FIFO is enabled [1]: Transmit DMA Enable. If this bit is set to 1, DMA for the transmit FIFO is enabled [15:2]: Reserved, read as zero, do not modify
SSP_RX_CNT_MSB	0x2C	RW	16	0x0000	MSB of the number of frame
SSP_RX_CNT_LSB	0x30	RW	16	0x0000	LSB of the number of frame
SSPITOP_EXD	0x90	RW	8	0x00	[2:0]: TXD[3:1] [5:3]: OE[3:1]
PID0	0xFE0	RO	8	0x22	Peripheral ID Register 4: value = 8'h22
PID1	0xFE4	RO	8	0x10	Peripheral ID Register 5: value = 8'h10
PID2	0xFE8	RO	8	0x24	Peripheral ID Register 6: value = 8'h24
PID3	0xFEC	RO	8	0x00	Peripheral ID Register 7: value = 8'h00
CID0	0xFF0	RO	8	0x0D	Component ID Register 0: value = 8'hd
CID1	0xFF4	RO	8	0xF0	Component ID Register 1: value = 8'hf0
CID2	0xFF8	RO	8	0x05	Component ID Register 2: value = 8'h5
CID3	0xFFC	RO	8	0xB1	Component ID Register 3: value = 8'hb1

5.1.3. W25Q64 简介

W25Q64 是华邦公司推出的大容量 SPI FLASH 产品，其容量为 64Mb。W25Q64 将 8M 字节的容量分为 128 块，每块大小为 64K 字节，每块又分为 16 个扇区，每个扇区 4K 个字节。W25Q64 的最小擦除单位为 1 个扇区，也就是每次必须擦除 4K 个字节。W25Q64 的擦写周期多达 10W 次，可将数据保存 20 年之久，支持 2.7~3.6V 的电压，标准的 SPI 通信，双输出/四输出的 SPI，最大 SPI 时钟可达 80Mhz。

5.1.3.1. W25Q64 的工作方式

W25Q64 支持 SPI 数据传输时序模式 0（CPOL=0、CPHA=0）和模式 3（CPOL=1、CPHA=1），模式 0 和模式 3 的主要区别就在于当 SPI 主机硬件处于空闲状态时，SCLK 的电平状态是高电平或者是低电平。对于模式 0 来说，SCLK 处于低电平；对于模式 3 来说，SCLK 处于高电平。不过，在这两种模式下，芯片都是在 SCLK 的上升沿采集输入数据，下降沿输出数据。W25Q64 传输的数据长度的大小为 8 位，先发高位，再发低位。

5.1.3.2. W25Q64 控制和状态寄存器

通过对“读状态寄存器”指令读出状态数据可以知道芯片存储器阵列是否可

以写入或者是否处于写保护状态。通过“写状态寄存器”指令可以配置芯片的写保护特征。对于状态寄存器的每一位的介绍如下图表 5- 2 所示。

表 5- 2 W25Q64 状态寄存器介绍表

标志位	标志位名称	描述
S0	BUSY	总线忙标志位，只读。当 W25Q64 执行“页编程”、“扇区擦除”、“块区擦除”、“芯片擦除”以及“写状态寄存器”指令时，该位被置 1。此时，除了“读状态寄存器”指令外的所有操作指令将会被芯片忽略。当芯片执行完这些指令之后，硬件将会自动将该位清零，表示此时可以对芯片进行其它操作。
S1	WEL	写保护位，只读。在执行完“写使能”指令之后，该位将会被硬件自动置 1。当芯片掉电后，执行“写禁止”、“页编程”、“扇区擦除”、“块区擦除”以及“芯片擦除”指令都会进入“写保护状态”。
S2	BP0	块区保护位，可读可写。这 3 位默认为 0，即块区处于未保护的状态，可以利用“写状态”指令对这几个位置 1 来达到块区保护的目的。注意：当状态寄存器的 SRP 位或读写保护管脚（WP）为低电平时，这 3 位不可被更改。
S3	BP1	
S4	BP2	
S5	TB	底部和顶部块保护位，可读可写。默认值为 0，可以利用“写状态寄存器”指令对这个位进行置 1 或者清零。当 TB=0 的时候，表示保护位从顶部开始；当 TB=1 时，表示保护位从底部开始。注意：当状态寄存器的 SRP 位或读写保护管脚（WP）为低电平时，这 3 位不可被更改。
S6	SEC	扇区/块保护，可读可写。默认值为 0，可以利用“写状态寄存器”指令对这个位进行置 1 或清零。当 SEC=0 的时候，表示每次保护的区域大小为 4K；当 SEC=1 时，表示每次保护的区域大小为 8K。
S7	SRP	状态寄存器的保护位，可读可写。默认值为 0，可以利用“写状态寄存器”指令对这个位进行置 1 或清零。这两位和读写保护管脚（WP）决定了状态寄存器写保护的方式。状态寄存器写保护的方式有：软件保护、硬件保护、电源锁定或一次性可编程（OTP）保护。
S8	SRL	
S9	QE	快速 SPI 通讯使能，可读可写。默认值为 0，可以利用“写状态寄存器”指令对这个位进行置 1 或清零。当 QE=0 时，W25Q64 设置为标准速度模式或快速模式，保持管脚（HOLE）和读写保护管脚（WP）被启用；当 QE=1 时，W25Q64 设置为高速模式，保存管脚（HOLE）和读写保护管脚（WP）被设置为 IO2 和 IO3 可用。
S10	(R)	未定义
S11	LB1	安全寄存器锁定位，可读可写。默认值为 0，表示安全寄存器此时处于解锁状态，可以利用“写状态寄存器”指令对这个位置 1，一旦被置为 1 之后，对应的 256 字节的安全寄存器将永久变为只读。
S12	LB2	
S13	LB3	
S14	CMP	补充保护位，可读可写。与 SEC、TB、BP2 和 BP0 位结合使用，为阵列保护提供了更大的灵活性，一旦被设置为 1，先前由 SEC、TB、BP2 和 BP0 设置的阵列保护状态将被反转。
S15	SUS	擦除/程序暂停状态位，只读。执行“擦除/编程挂起”指令之后，通过“擦除/程序恢复”指令使对应的功能继续执行。
S16	(R)	未定义
S17	(R)	未定义
S18	WPS	写保护选择位，可读可写。当 WPS=0 时，设备将使用 CMP、SEC、TB、BP[2:0]这几位的组合来保护内存阵列的特定区域；WPS=1 时，设备将利用单个块锁来保护任何单个扇区或块。当设备通电或复位后，所有单个块锁定位将会被置 1。

S19	(R)	未定义
S20	(R)	未定义
S21	DRV0	输出驱动强度位，可读可写。用于确定读取操作的输出驱动器强度：00：100%；10：75%；01：50%；11：25%。
S22	DRV1	
S23	(R)	未定义

5.1.3.3. W25Q64 常用操作指令

W25Q64 的常用操作指令如下表 5- 3 所示。

表 5- 3 W25Q64 常用操作指令表

指令名称	字节 1(CODE)	字节 2	字节 3	字节 4	字节 5	字节 6
写使能	06h					
写禁止	04h					
读状态寄存器 1	05h	S7~S0				
读状态寄存器 2	35h	S15~S8				
读数据	03h	A23~A16	A15~A8	A7~A0	D7~D0	直至读完所有
写状态寄存器	01h	S7~S0	S15~S8			
页编程	02h	A23~A16	A15~A8	A7~A0	D7~D0	直至读完所有
块擦除（64K）	D8h	A23~A16	A15~A8	A7~A0		
半块擦除（32K）	52h	A23~A16	A15~A8	A7~A0		
扇区擦除	20h	A23~A16	A15~A8	A7~A0		
芯片擦除	C7/60h					
芯片掉电	B9h					
释放掉电/器件 ID	ABh	伪字节	伪字节	伪字节	ID7~ID0	
制造/器件 ID	90h	伪字节	伪字节	00h	MF7~MF0	ID7~ID0
JEDEC ID	9Fh	MF7~MF0	ID15~ID8	ID7~ID0		

5.2. 实验介绍

本次实验将使用 CM3 自带的 SPI 和 IO 模拟 SPI 这两种方式来实现对外部 Flash（W25Q64）的读写，并将结果通过串口助手显示出来。

5.3. 建立 HQFPGA 工程

将已有的 HQ 工程复制到存放本次实验代码的文件夹下，并且对其进行修改，操作方式参考实验二中 2.3 节的内容。

5.4. 物理管脚约束

本次实验使用 SPI0，在引脚约束文件“ac208.upc”中对其进行引脚分配，分配完成之后保存文件，如下图 5. 1 所示。

```
86  #SPI0_D1/SPI0_MISO
87  #phycst.pin.set {GPIO[10]} F11
88
89  #----SPI0 接 SPI FLASH-----
90  #SPI0_SCLK
91  phycst.pin.set {GPIO[7]} A1
92  #SPI0_CS
93  phycst.pin.set {GPIO[8]} A8
94  #SPI0_D0/SPI0_MOSI
95  phycst.pin.set {GPIO[9]} G2
96  #SPI0_D1/SPI0_MISO
97  phycst.pin.set {GPIO[10]} G1
```

图 5. 1 分配 SPI0 给 SPI Flash

5.5. 编译设计

点击“全部运行”按钮，对设计进行全编译并生成 bin 文件。操作步骤如下图 5. 2 所示。

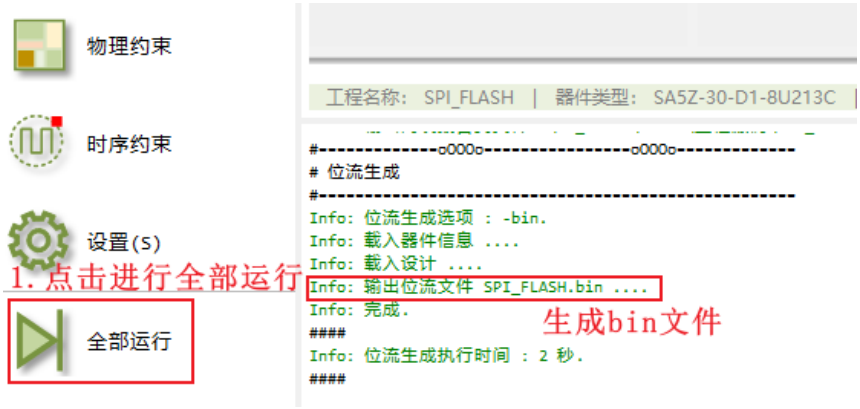

图 5. 2 编译运行整个文件

5.6. 建立 MDK 工程

将已有的 MDK 工程复制至本次工程的文件下，并对其进行修改，操作方式参考实验二中 2.8 节的内容。

5.7. 软件设计

本次实验需要新加的库文件一共有三个：第一个是官方提供的 SPI 库文件“CM3DS_spi”；第二个是我们提供的 IO 模拟 SPI 的库文件“io_spi”；第三个是操作 SPI Flash 的库文件“spi_flash”。添加方式参考 3.6.1 节中的内容。下面我们将介绍一下一些“spi_flash”中一些常用的函数。

5.7.1. SPI_FLASH 库

5.7.1.1. SPI_FLASH_Init

初始化 SPI_FLASH。函数分为两种方式进行初始化：

1. IO 口模拟 SPI 的初始化，主要就是将对应的 IO 口设置为输入或者输出，并且关闭其复用和中断功能。

2. 配置 CM3 自带的 SPI：首先是设置对应的 IO 口（位于 SSP_PinRemap 这个函数中），主要就是启用 SPI0 的 CLK、MOSI、MISO 这三个 IO 口的复用功能，对于 CS 这个 IO 口，我们关闭其复用和中断的功能，通过手动的方式去控制这个信号（经过测试发现，如果需要连续向 SPI FLASH 写入多个数据就必须通过手动控制 CS 信号）；然后就是配置 SPI 的相关的参数，对于相关参数的说明如下表 5- 4 所示。

表 5- 4 SPI 的相关参数配置表

参数名称	参数说明
CLK	设置 SPI 的分频系数。向分频寄存器 SSPCPSR 中写入对应的分频值
Data_Size	SPI 数据传输的位数选择。向 SSPCR0 寄存器的[3:0]写入对应的值
Mode	SPI 工作模式的选择，一共有四种模式可供选择，SPI FLASH 使用模式 0 和模式 3，在这里将其设置为模式 0（SSP_SPH0SPO0）。向 SSPCR0 寄存器的第 6 位和第 7 位写入对应的值：模式 0：00；模式 1：10；模式 2：01；模式 3：11。
LBM	回环模式或者正常串口传输方式的选择。SSPCR1 的第[0]位控制这两种传输方式的选择：0：启用正常的串行端口操作；1：回环模式，将串口移位寄存器的输出连接到输入形成回环。
Width	数据位宽的选择。SSPCR1 寄存器的[5:4]位控制数据位宽：00：Standard/single；01：Dual；10：Quad
TXRXSIMULT	传输方式的选择，分时（SSP_TXRXSIMULT_Time_Sharing）或者同时（SSP_TXRXSIMULT_simultaneous）。如果是回环模式的时候就选择同时，其它都选择分时。

上表中对于寄存器的操作都是在 SSP_Init 这个函数中进行实现的，在使用的时候我们只需要配置相关的参数，然后启用该函数就可以了，该函数的参数总过有两个：CM3DS_MPS2_SSPx：选择哪一个 SPI（SPI0: CM3DS_MPS2_SSP0; SPI1: CM3DS_MPS2_SSP1）；SSP_InitStruct：参数结构体选择，选择 SSP_Initstructure。SPI FLASH 初始化函数的代码如下所示：

```
void SPI_FLASH_Init()
{
    #if SPI_FLASH_HAL
    SSP_InitTypeDef  SSP_Initstructure;
    SSP_PinRemap();
    SSP_Initstructure.CLK = SSP_CLK_4Prescale;//SPI 的工作频率
    SSP_Initstructure.Data_Size  = SSP_Data_Size_8bit;//数据位数
    SSP_Initstructure.Mode = SSP_SPH0SPO0; //工作模式选择：模式 0：CPOL=0 CPOHA=0
    SSP_Initstructure.LBM  = SSP_LBM_Normal; //SSP_LBM_Normal; SSP_LBM_LoopBack
    SSP_Initstructure.Width = SSP_Width_Standard;   //数据位宽
    //传输方式（分时或同时）：同时收发只适用于回环模式；分时收发适用于单独的传输或接收
    SSP_Initstructure.TXRXSIMULT = SSP_TXRXSIMULT_Time_Sharing;
```

```
    SSP_Init(CM3DS_MPS2_SSP0,&SSP_Initstructure);
    #endif
    #if SPI_FLASH_IO
    SPI_GPIO_Init();//SPI IO 初始化
    #endif
}
```

在代码中包含两种 SPI 的实现方式，可以在“spi_flash.c”中进行选择，如下所示：

```
#define SPI_FLASH_HAL 1 //选择 CM3 自带的 SPI
#define SPI_FLASH_IO 0 //选择 IO 模拟
```

在需要使用到 SPI FLASH 的时候，都需要先进行初始化，函数的使用方式如下所示：

```
SPI_FLASH_Init();//SPI_FLASH 初始化
```

5.7.1.2. SPI_FLASH_SendByte

SPI FLASH 发送一个字节的数据，分为两种方式：

1. IO 模拟：SPI_Write_Byte，SPI 发送一个字节的数据，输入的参数就是待发送的数据，在第一跳变沿（CLK 信号从低电平变成高电平）开始发送数据，函数的实现方式如下所示：

```
void SPI_Write_Byte(uint8_t dt)
{
    uint8_t i;
    for(i = 0;i < 8;i++){
        SPI_GPIO_WriteBit(SPI_SCLK,0);
        if((dt << i) & 0x80)
            SPI_GPIO_WriteBit(SPI_MOSI,1);
        else
            SPI_GPIO_WriteBit(SPI_MOSI,0);
        SPI_GPIO_WriteBit(SPI_SCLK,1);
    }
    SPI_GPIO_WriteBit(SPI_SCLK,0);
}
```

2. CM3 自带的 SPI：SSP_SendData，函数的主要参数就是 CM3DS_MPS2_SSPx：选择哪一个 SPI，SPI0（CM3DS_MPS2_SSP0），SPI1（CM3DS_MPS2_SSP1）；Data：需要写入的数据。通过 SPI 实现数据的传送就是向 DR 寄存器中写入需要发送的数据，该函数的实现方式如下：

```
void SSP_SendData(CM3DS_MPS2_SSP_TypeDef* CM3DS_MPS2_SSPx, uint16_t Data)
{
    assert_param(IS_SSP_ALL_PERIPH(CM3DS_MPS2_SSPx));
```

```
    CM3DS_MPS2_SSPx-> DR = Data;
}
```

SPI_FLASH_SendByte 函数中包含两种方式的选择，代码如下所示：

```
void SPI_FLASH_SendByte(uint8_t byte)
{
    #if SPI_FLASH_IO
    SPI_Write_Byte(byte);
    #endif

    #if SPI_FLASH_HAL
    SSP_SendData(CM3DS_MPS2_SSP0, byte);
    #endif
}
```

函数的使用方式如下，使用的时候只需要填入你想要传输的数据。

```
SPI_FLASH_SendByte(W25X_ReadData); //W25X_ReadData:需要发送的数据
```

5.7.1.3. SPI_FLASH_ReadByte

读取 SPI FLASH 内部的一个字节数据，返回读取到的数据。一共有两种实现方式：

1. IO 模拟的方式：SPI FLASH 支持 SPI 的模式 0 及模式 3，这里采用模式 0：当 SPI 为空闲状态的时候，时钟信号 SCLK 的电平为低电平，SPI 在 SCLK 第一个跳变沿开始采样数据，函数的实现代码如下所示：

```
unsigned char SPI_Read_Byte(void)
{
    unsigned char i, rByte = 0;
    //SPI_SCLK=0;
    SPI_GPIO_WriteBit(SPI_SCLK,0);
    for(i = 0;i < 8;i++){
        //SPI_SCLK=1;
        SPI_GPIO_WriteBit(SPI_SCLK,1);
        rByte <<= 1;
        //rByte|=SPI_MISO;
        rByte |= SPI_GPIO_RdData(SPI_MISO);
        //SPI_SCLK=0;
        SPI_GPIO_WriteBit(SPI_SCLK,0);
    }
        return rByte;
}
```

2. CM3 自带的 SPI：首先需要查看 SPI 的 SSPSR 寄存器的第[2]位是否为 1，如果为 1 的话表示接收 FIFO 非空，此时有数据可以进行传输，这一步是在 SSP_Receie

veData 这个函数中实现的。然后读取 DR 数据寄存器中的数据，返回读取到的数据。

函数代码如下所示：

```
uint8_t SPI_FLASH_ReadByte(void) //SPI 读一个字节
{
    uint8_t rxdata;
    #if SPI_FLASH_IO
    rxdata = SPI_Read_Byte();
    #endif

    #if SPI_FLASH_HAL
    SSP_ReceiveReady(CM3DS_MPS2_SSP0);//是否可以开始传输数据
    rxdata = SSP_ReceiveData(CM3DS_MPS2_SSP0);
    #endif
    return rxdata;
}
```

函数的使用方式如下：

```
uint8_t rxdata; //读取到的数据、
rxdata = SPI_FLASH_ReadByte();
```

5.7.1.4. SPI_FLASH_ReadDeviceID

读取 SPI FLASH 的设备 ID，其实现步骤如下所示：

1. 拉低片选，选中器件：输出低电平给 CS 信号对应的 IO 口。

2. 如果使用的 CM3 自带的 SPI，需要确定 SPI 接收的字节数（IO 模拟跳过此步骤）：通过 SSP_ReceiveDataNum 函数实现，主要是对 SSP_RX_CNT_MSB 寄存器和 SSP_RX_CNT_LSB 寄存器进行操作，通过向寄存器中写入对应的值，来确定需要接收的字节数。举个例子，需要接收一个字节的数据，实现方式如下所示：

```
SSP_ReceiveDataNum(CM3DS_MPS2_SSP0, 1);
```

3. 通过 SPI_FLASH_SendByte 函数发送读设备 ID 命令，查看表 4-3 中的内容得知，查看设备 ID 的命令可以是 ABh 或者 90h，这里选择 ABh，然后连续发送三个字节的伪字节（Dummy_Byte）。

4. 使能 SPI，确定 SPI 传输结束（IO 模拟跳过此步骤）：在使用 CM3 自带的 SPI 的时候，将需要传输的数据写入 DR 之后，还需要使能 SPI 才能将数据发送出去，使能 SPI 就是通过 SSP_Cmd 函数将 SSPCR1 寄存器的第[1]位置 1。使能 SPI 之后还需要通过 SSP_SendFinish 函数读取 SSPSR 寄存器的第[5]位的值，一直等到该位为 0 则跳出该函数，表示此时 SPI 发送处于空闲状态，也就是说数据传输完成，才能进行接下来的操作。

5. 通过 SPI_FLASH_ReadByte 函数读取得到的设备 ID 号。

6. SPI 不使能（IO 模拟跳过此步骤），当 SPI 数据传输完成之后，需要通过 SSP_Cmd 函数失能 SPI。

7. 拉高片选，传输结束，返回读取到的设备 ID，对于 W25Q64 来说，读取到的设备 ID 号应该为 0x16。

上述步骤中详细说明了如何使用 CM3 自带的 SPI 去传输和发送数据，后续在需要使用 SPI 进行通信的时候，可以参考以上步骤。

函数的代码实现如下所示：

```
uint32_t SPI_FLASH_ReadDeviceID(void)
{
    uint32_t Temp = 0;
    /* 拉低片选选中器件 */
    SPI_FLASH_CS_LOW();
    //SPI 接收多少字节的数据
    #if SPI_FLASH_HAL
    SSP_ReceiveDataNum(CM3DS_MPS2_SSP0, 1);
    #endif
    /* 发送 "RDID "命令 */
    SPI_FLASH_SendByte(W25X_DeviceID);
    SPI_FLASH_SendByte(Dummy_Byte);
    SPI_FLASH_SendByte(Dummy_Byte);
    SPI_FLASH_SendByte(Dummy_Byte);
    #if SPI_FLASH_HAL
    SSP_Cmd(CM3DS_MPS2_SSP0,ENABLE);//SPI 使能
    SSP_SendFinish(CM3DS_MPS2_SSP0);//传输结束
    #endif
    Temp = SPI_FLASH_ReadByte();//SPI 读取
    #if SPI_FLASH_HAL
    SSP_Cmd(CM3DS_MPS2_SSP0,DISABLE);//SPI 不使能
    #endif
    /* 传输结束，拉高片选 */
    SPI_FLASH_CS_HIGH();
    return Temp;
}
```

函数的使用方式如下：

```
uint8_t DeviceID = 0;
DeviceID = SPI_FLASH_ReadDeviceID();
```

5.7.1.5. SPI_FLASH_WriteEnable

SPI FLASH 写使能，在需要向 SPI FLASH 写入数据之前，都需要执行该操作，才能写入，写使能命令为 0x06。对于 SPI 的操作可以参考 5.7.1.4 一节中内

容。函数的实现代码如下所示：

```
void SPI_FLASH_WriteEnable(void)
{
    /* 拉低片选选中器件 */
    SPI_FLASH_CS_LOW();
    /* 发送 "Write Enable"命令 */
    SPI_FLASH_SendByte(W25X_WriteEnable);
    #if SPI_FLASH_HAL
    SSP_Cmd(CM3DS_MPS2_SSP0,ENABLE);//SPI 使能
    SSP_SendFinish(CM3DS_MPS2_SSP0);//传输结束
    SSP_Cmd(CM3DS_MPS2_SSP0,DISABLE);//SPI 不使能
    #endif
     /* 传输结束，拉高片选 */
     SPI_FLASH_CS_HIGH();
}
```

函数的使用方式如下所示：

```
SPI_FLASH_WriteEnable();
```

5.7.1.6. SPI_FLASH_WaitForWriteEnd

轮询 FLASH 状态寄存器中的 BUSY 标志位的状态，一直等到该位为 0，表示写结束或者擦除结束，可以进行接下来的操作了。

函数内容：拉低片选，选中器件，发送读状态寄存器指令（0x05），返回读取到的数据，通过 while 循环等待 BUSY 位为 0，最后拉高片选，传输结束。函数的实现代码如下所示：

```
void SPI_FLASH_WaitForWriteEnd(void)
{
    uint8_t FLASH_Status = 0;
    /* 拉低片选选中器件 */
    SPI_FLASH_CS_LOW();
    #if SPI_FLASH_IO
    /* 发送 "Read Status Register" 命令 */
    SPI_FLASH_SendByte(W25X_ReadStatusReg);
    #endif
    /* Loop as long as the memory is busy with a write cycle */
    do
    {
        #if SPI_FLASH_HAL
        SSP_ReceiveDataNum(CM3DS_MPS2_SSP0, 1);
        #endif
        /* 发送 "Read Status Register" 命令 */
        SPI_FLASH_SendByte(W25X_ReadStatusReg);
```

```
        #if SPI_FLASH_HAL
        SSP_Cmd(CM3DS_MPS2_SSP0,ENABLE);//SPI 使能
        SSP_SendFinish(CM3DS_MPS2_SSP0);//传输结束
        #endif
        FLASH_Status = SPI_FLASH_ReadByte();//SPI 读取
        #if SPI_FLASH_HAL
        SSP_Cmd(CM3DS_MPS2_SSP0,DISABLE);//SPI 不使能
        #endif
    }
    while ((FLASH_Status & WIP_Flag) == SET); /* Write in progress */
    /* 传输结束，拉高片选 */
    SPI_FLASH_CS_HIGH();
}
```

函数的使用方式参考如下所示：

```
SPI_FLASH_WaitForWriteEnd();
```

5.7.1.7. SPI_FLASH_PageWrite

SPI FLASH 页写入，函数的参数说明如下表 5- 5 所示：

表 5- 5 SPI_FLASH_PageWrite 函数参数说明表

参数名称	参数意义
pBuffer	指向包含要写入缓存的数据的缓冲区的指针
WriteAddr	需要写入的数据的 FLASH 内部地址
NumByteToWrite	写入缓存的字节数

函数内容：写使能，拉低片选，发送页写指令（0x02），发送 24 位的地址，错误处理，一页最多写 256 个字节的数据，向缓冲区写入数据，注意在使用 CM3 自带的 SPI 的时候在发送的时候要实时检测接收 FIFO 是否已经满了，如果满了，那么就要使能 SPI，等待传输结束，然后再重新向接收 FIFO 中写入数据进行传输。最后拉高片选，等待写结束。

函数代码如下所示：

```
void SPI_FLASH_PageWrite(uint8_t* pBuffer, uint32_t WriteAddr,
uint16_t NumByteToWrite)
{
    unsigned short i;
    /* 发送允许写入指令 */
    SPI_FLASH_WriteEnable();
    /* 拉低片选选中器件 */
    SPI_FLASH_CS_LOW();
    /* 发送”写入内存”指令 */
    SPI_FLASH_SendByte(W25X_PageProgram);
    /* 发送 24 位的地址 */
```

```
    SPI_FLASH_SendByte((WriteAddr & 0xFF0000) >> 16);
    SPI_FLASH_SendByte((WriteAddr & 0xFF00) >> 8);
    SPI_FLASH_SendByte(WriteAddr & 0xFF);
    //错误处理,一页最多写 256 个字节的数据
    if(NumByteToWrite > SPI_FLASH_PerWritePageSize)
    {
        NumByteToWrite = SPI_FLASH_PerWritePageSize;
    }
    /* 向缓存区写入数据 */
    for (i = 0; i < NumByteToWrite; i++)
    {
        /* 发送当前字节 */
        SPI_FLASH_SendByte(*pBuffer++);
        //判断 spi 接收 fifo 是否满了
        #if SPI_FLASH_HAL
        if(!(CM3DS_MPS2_SSP0->SR & (SSP_SR_TNF_Msk)))
        {
            SSP_Cmd(CM3DS_MPS2_SSP0,ENABLE);//SPI 使能
            SSP_SendFinish(CM3DS_MPS2_SSP0);//传输结束
            SSP_Cmd(CM3DS_MPS2_SSP0,DISABLE);
        }
    #endif
    }
    #if SPI_FLASH_HAL
    SSP_Cmd(CM3DS_MPS2_SSP0,ENABLE);//SPI 使能
    SSP_SendFinish(CM3DS_MPS2_SSP0);//传输结束
    SSP_Cmd(CM3DS_MPS2_SSP0,DISABLE);
    #endif
    /* 传输结束，拉高片选 */
    SPI_FLASH_CS_HIGH();
    /* 等待写结束 */
    SPI_FLASH_WaitForWriteEnd();
}
```

函数的使用参考方式如下：

```
SPI_FLASH_PageWrite(C_Data, add, len);
```

5.7.1.8. SPI_FLASH_BufferWrite

将数据块写入缓存，函数的参数说明如下表 5- 6 所示。

表 5- 6 SPI_FLASH_BufferWrite 函数参数说明表

变量名称	变量意义
pBuffer	指向包含要写入数据的缓冲区的指针
WriteAddr	需要写入数据的 FLASH 的内部地址

NumByteToWrite	写入缓存的字节数

函数内容：取模，检查有几页，对齐到页地址差多少个数据，计算不满一页的字节数。当 addr=0 的时候，刚好按页对齐，依次通过 SPI_FLASH_PageWrite 函数连续页写入。当地址不对齐的情况下，当小于一页的时候，先把当前页写完，如果当前页写不完，另起一页进行写入。详细的函数代码请自行查看“spi_flash.c”中的内容。

函数的使用参考方式如下：

```
SPI_FLASH_BufferWrite(Tx_Buffer, FLASH_WriteAddress, BufferSize);
```

5.7.1.9. SPI_FLASH_SectorErase

扇区擦除函数，一次擦除 4k 的空间。函数的输入参数只有一个：SectorAddr，需要擦除的扇区的起始地址。

函数内容：写使能，拉低片选，发送扇区擦除指令（0x20），发送 24 位的地址字节，最后拉高片选，传输结束，等待擦除完成。

函数的实现代码如下所示：

```
void SPI_FLASH_SectorErase(uint32_t SectorAddr)
{
    /* 发送允许写入指令 */
    SPI_FLASH_WriteEnable();
    SPI_FLASH_WaitForWriteEnd();
    /* 拉低片选选中器件 */
    SPI_FLASH_CS_LOW();
    /* 发送扇区清除指令 */
    SPI_FLASH_SendByte(W25X_SectorErase);
    /* 发送 24 位的地址字节 */
    SPI_FLASH_SendByte((SectorAddr & 0xFF0000) >> 16);
    SPI_FLASH_SendByte((SectorAddr & 0xFF00) >> 8);
    SPI_FLASH_SendByte(SectorAddr & 0xFF);
    #if SPI_FLASH_HAL
    SSP_Cmd(CM3DS_MPS2_SSP0,ENABLE);//SPI 使能
    SSP_SendFinish(CM3DS_MPS2_SSP0);//传输结束
    SSP_Cmd(CM3DS_MPS2_SSP0,DISABLE);
    #endif
    /* 传输结束，拉高片选 */
    SPI_FLASH_CS_HIGH();
    /* 等待擦除完成 */
    SPI_FLASH_WaitForWriteEnd();
}
```

函数的使用方式如下所示：

```
SPI_FLASH_SectorErase(0x000000);
```

5.7.1.10. SPI_FLASH_BufferRead

从缓存中读取数据，函数的参数说明如下表 5- 7 所示。

表 5- 7 SPI_FLASH_BufferRead 函数参数说明表

参数名称	参数意义
pBuffer	指向读取的数据的缓冲区的指针。
ReadAddr	缓存区的内部地址
NumByteToRead	从缓存区读取的数据长度

函数内容：拉低片选，选中器件，发送“从内存读取”的指令，写 24 位的地址，再通过 SPI_FLASH_ReadByte 函数循环读取数据，最后拉高片选，传输完成。函数的代码实现如下：

```
void SPI_FLASH_BufferRead(uint8_t* pBuffer, uint32_t ReadAddr,
uint16_t NumByteToRead)
{
    /* 拉低片选选中器件 */
    SPI_FLASH_CS_LOW();
    #if SPI_FLASH_HAL
    //需要读取的数据的长度
    SSP_ReceiveDataNum(CM3DS_MPS2_SSP0, NumByteToRead);
    #endif
    /* 发送”从内存读取”指令 */
    SPI_FLASH_SendByte(W25X_ReadData);
    /* 写 24 位的地址 */
    SPI_FLASH_SendByte((ReadAddr & 0xFF0000) >> 16);
    SPI_FLASH_SendByte((ReadAddr& 0xFF00) >> 8);
    SPI_FLASH_SendByte(ReadAddr & 0xFF);
    #if SPI_FLASH_HAL
    SSP_Cmd(CM3DS_MPS2_SSP0,ENABLE);//SPI 使能
    SSP_SendFinish(CM3DS_MPS2_SSP0);//发送结束
    #endif
    while (NumByteToRead--)
    {
        *pBuffer = SPI_FLASH_ReadByte();//SPI 读取
        /* 指向将要保存所读字节的下一个位置 */
        pBuffer++;
    }

    #if SPI_FLASH_HAL
    SSP_Cmd(CM3DS_MPS2_SSP0,DISABLE);//传输不使能
    #endif
    /* 传输结束，拉高片选 */
    SPI_FLASH_CS_HIGH();
```

```
}
```

函数的使用方式参考如下：

```
SPI_FLASH_BufferRead(Rx_Buffer, FLASH_ReadAddress, BufferSize);
```

5.7.2. 添加用户程序

本次实验的功能：对 SPI FLASH 进行读写操作，在这里通过按键控制 SPI FLASH 的读写：当按键 S0 被按下的时候，擦除 FLASH 的缓存区，向擦除的区域写入数据；当按键 S1 被按下的时候，将刚刚写入的数据读取出来。

函数代码编写：定义两个缓存区：Tx_Buffer：发送缓存区，存放我们需要发送的数据，本次实验，向发送缓存区写入“Flash Test OK!!!”；Rx_Buffer：接收缓存区，存放读取到的数据。设置需要读取和写入的地址为 FLASH_WriteAddress（0x00000）。在 main 函数中首先依次对 SPI FLASH、串口和按键进行初始化，然后读取 FLASH 的 ID，当 ID 读取成功之后，就可以通过按键控制 FLASH 的读写。当按下按键 S0 的时候，擦除缓存区，将发送缓存区的数据写入到 FLASH 当中，然后将写入的数据通过串口打印出来；当按下按键 S1 的时候，将刚刚写入的数据读取出来，同时也将读取到的数据也通过串口打印出来，最后我们可以查看写入的数据和读取到的数据是否相同，来判断读写 FLASH 是否成功。main 函数中的代码如下所示：

```
int main(void)
{
    uint8_t DeviceID = 0;
    SPI_FLASH_Init();//SPI_FLASH 初始化
    Uart_Init(CM3DS_MPS2_UART0,9600);//初始化串口
    KEY_INT_Init(KEY0,PORT0_KEY0_IRQn);//启用按键 KEY0 的中断
    KEY_INT_Init(KEY1,PORT0_KEY1_IRQn);//启用按键 KEY1 的中断
    while(1){
        /* 得到 SPI Flash Device ID */
        DeviceID = SPI_FLASH_ReadDeviceID();
        /* 检查 SPI Flash ID */
        if (DeviceID == sFLASH_ID)
        {
            if(key0_status == 1) { //按下按键 0
                /* 擦除 SPI FLASH 缓存区来写*/
                SPI_FLASH_SectorErase(FLASH_SectorToErase);
                /* 将发送缓冲区的数据写到 flash 中 */
                SPI_FLASH_BufferWrite(Tx_Buffer, FLASH_WriteAddress, BufferSize);
                printf("\r\n 写入的数据为：%s \r\n", Tx_Buffer);
                key0_status = 0;
            }
```

```
            if(key1_status == 1){ //按下按键 1
                /* 将刚刚写入的数据读出来放到接收缓冲区中 */
                SPI_FLASH_BufferRead(Rx_Buffer, FLASH_ReadAddress, BufferSize);
                printf("\r\n 读出的数据为：%s \r\n", Rx_Buffer);
                key1_status = 0;
            }
        }
        else{
            printf("\r\n 获取不到 W25Q64 ID! \r\n ");
        }
    }
}
```

5.8. 板级验证

5.8.1. 实验所需硬件

(1) AC208 开发板
(2) FPGA 下载器：XIST USB Cable
(3) CM3 仿真器：DAP Link
(4) 电源线一根

5.8.2. 硬件连接

硬件连接参考实验一。

5.8.3. 下载文件至目标板

下载文件的方式参考实验一。

5.8.4. 功能演示

将串口调试助手打开，波特率设置为 9600。
按下按键 S0，串口显示写入的数据，如下图 5. 3 所示。

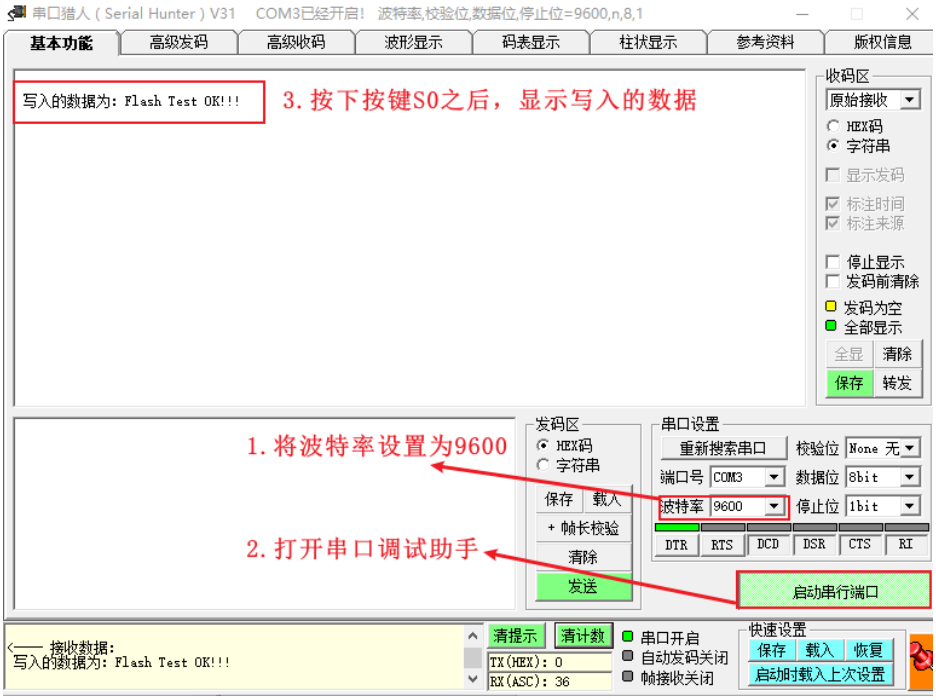

图 5.3 按键 S0 按下之后串口显示的结果

按下按键 S1 之后，通过串口助手显示刚刚写入 Flash 的数据，如下图 5.4 所示。

图 5.4 按下按键 S1 之后串口显示的结果

通过上图打印出来的数据可以看出，写入和读出的数据是一致的，说明对于 Flash 的读写成功了。

5.9. 思考与总结

本次实验完成了对 Flash 读写的操作，在实验中需要注意以下几点：

1. 在使用 CM3 自带的 SPI 连续进行写操作的时候，发送 FIFO 中最多能存储 8 个字节的数据。当通过 SPI_FLASH_SendByte 函数连续发送 8 个字节的数据之后，SSPSR 寄存器的 TNF 位（第[1]位）将会被置 0，也就是说明发送 FIFO 已满，如果此时不使能 SPI，将发送 FIFO 中的数据发送出去，继续向 FIFO 中写入数据的话，数据将会丢失。本次实验中采取的操作是：在发送数据的时候不断读取 SSPSR 寄存器的 TNF 位的状态，当检测到发送 FIFO 满了，就使能 SPI，将发送 FIFO 数据发送出去，然后再通过 SPI_FLASH_SendByte 函数向发送 FIFO 中写入数据。

2. 向 Flash 中写入数据之前，都必须先进行擦除，W25Q64 每次至少擦除一个扇区，也就是每次必须擦除 4K 个字节。擦除的时候首先指定需要擦除的扇区地址，地址可以是指定扇区内的任意一个地址（都是擦除的指定的扇区），建议选择指定扇区的起始地址。

3. 本次实验指定的擦除地址为芯片的起始地址 0x000000，擦除之后，Flash 存放的本次实验的代码将会被清除，当开发板重新上电之后，需要重新下载本次实验代码，才能进行实验。