应用笔记

Application Note

文档编号：**AN1103**

APM32F103 软件模拟 SPI 读写外部 Flash

版本：**V 1.0**

1 引言

本应用笔记提供如何在 APM32F103 系列上通过软件模拟 SPI 读写外部 Flash。以规避在开发过程中串行通信串口有限的问题。同时，也可以降低开发板的成本和 PCB 复杂度。本文内容基于 APM32F103 系列进行开发。

目录

2 APM32F103 SPI 简介

SPI，全称是 Serial Peripheral Interface，即串行外设接口。SPI 接口通常运用于 EEPROM，Flash，OLED 等设备之间。APM32F103 SPI 接口可以配置为支持 SPI 协议和 I2S 音频协议，默认工作为 SPI 模式。串行外设接口（SPI）提供了基于 SPI 协议的数据收发功能，允许芯片与外部设备以半双工、全双工、同步和串行方式通信，可以工作于主机或从机模式。

2.1 SPI 接口介绍

SPI 接口通常由 MOSI（Master Output Slave Input）、MISO（Master Input Slave Output）、SCLK（Serial Clock）、CS（Chip Select）四条线组成，具体情况如下图 1：

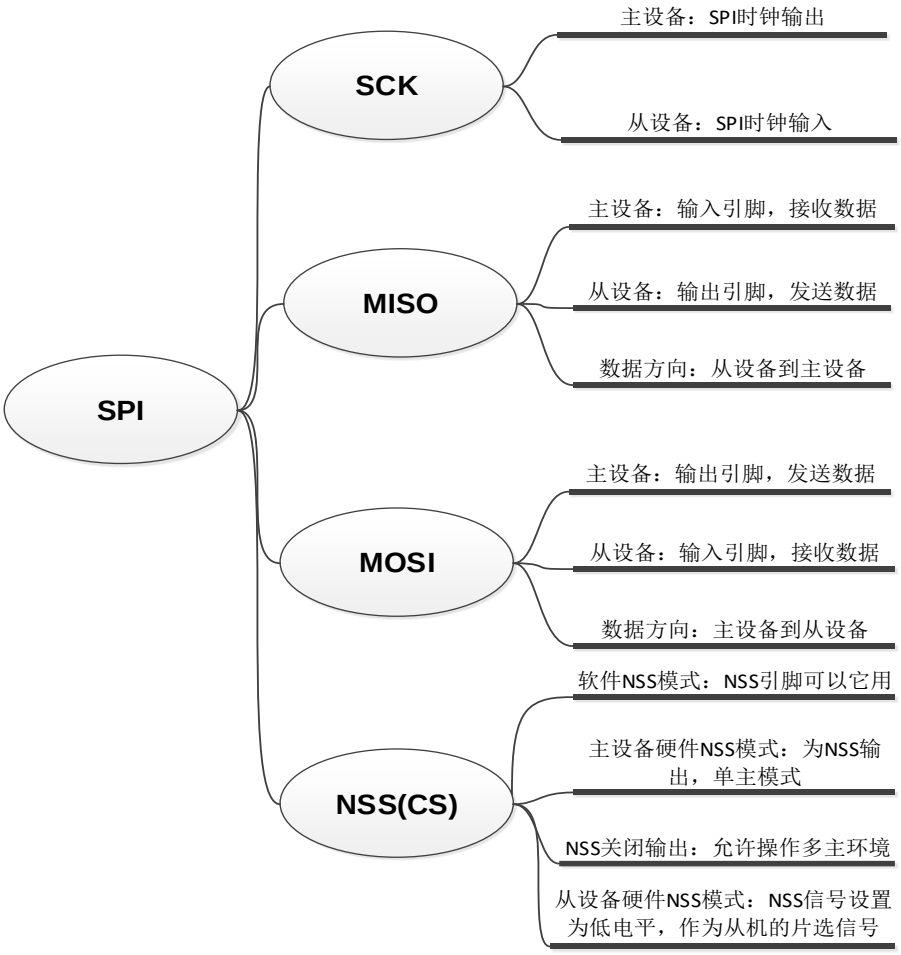

图 1 SPI 四线介绍图

2.2 SPI 传输模式

SPI 有四种传输模式,其中主要区别在于 SPI_CTRL1 寄存器的 CPOL（时钟极性）和 CPHA（时钟相位）。CPOL 是指 SPI 处于空闲状态时，SCK 信号线的电平信号。CPHA 是指数据的采样时刻。根据二者的不同状态，可以将 SPI 分成四种模式，如下图 2：

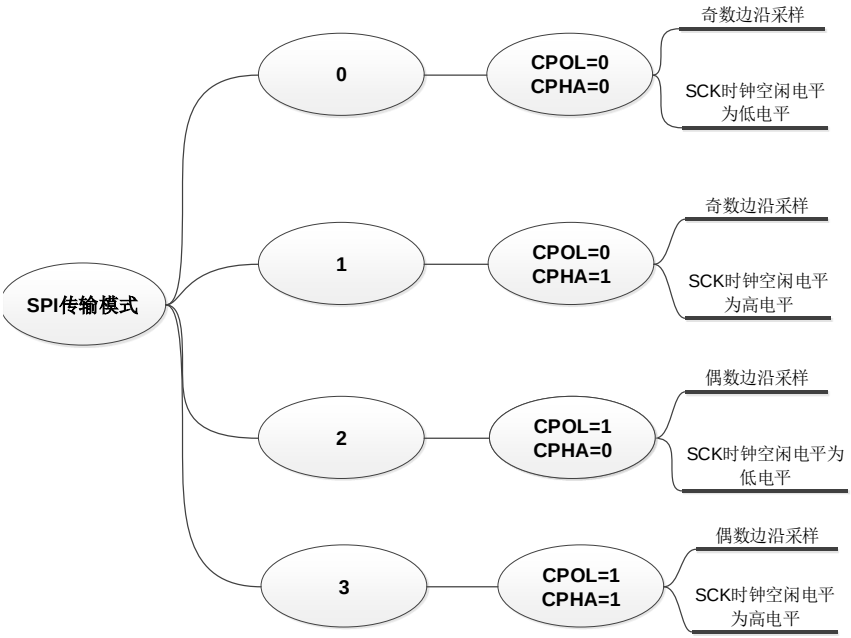

图 2 SPI 传输模式

根据上面的图可知,当 CPHA=0 的时候,即表示会在第一个时钟边沿开始对数据进行采样。其中，当 CPOL=0 时，即表示 SCK 时钟空闲电平为低电平，由低电平变为高电平时，会在上升沿进行采样。反之，当 CPOL=1 时，即 SCK 时钟空闲时为高电平，由高电平变为低电平，会在其下降沿进行采样。如下图 3：

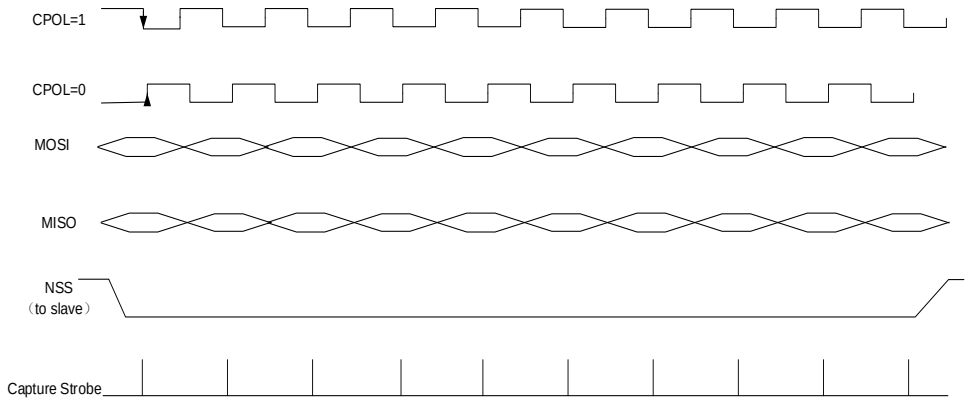

图 3 SPI 传输时序图

当 CPHA=1 的时候，即表示会在第二个时钟边沿开始对数据进行采样。其中，当 CPOL=0 时，即表示 SCK 时钟空闲电平为低电平，由低电平变为高电平在变为低电平时，会在上升沿进行采样。反之，当 CPOL=1 时，即 SCK 时钟空闲时为高电平，由高电平变为低电平在变为高电平，会在下降沿进行采样。如下图 4：

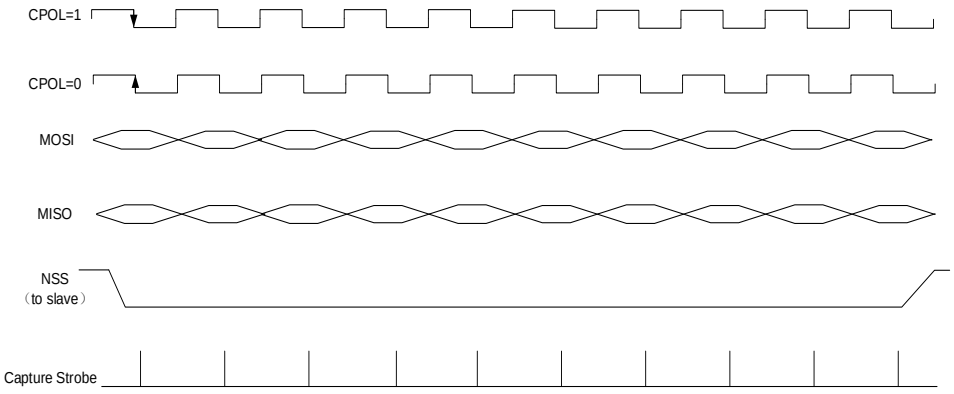

图 4 SPI 传输时序图

2.3 数据交换

在 SCK 时钟周期下，MOSI 和 MISO 同时处于工作状态。如下图 5：

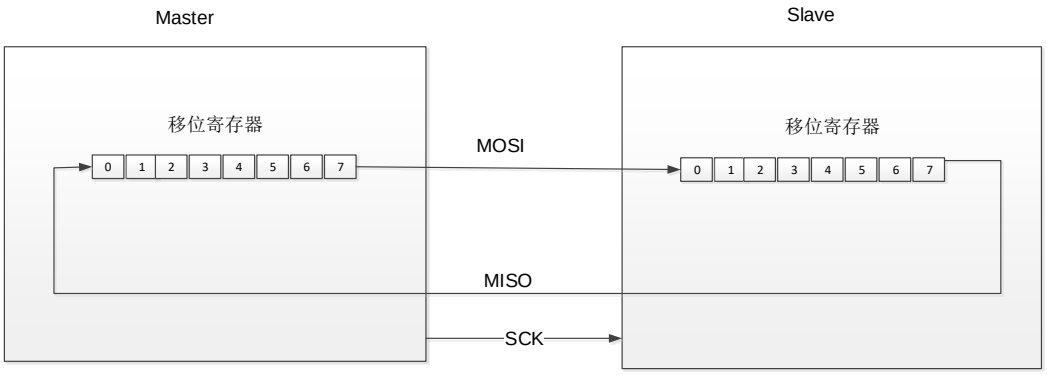

图 5 数据交换图

如上，SPI 通信中，主机和从机都有属于自己的移位寄存器，其用于数据的传输。在传输过程中，MOSI 的作用是将主机移位寄存器的数据写入从机的移位寄存器，同时，MISO 的作用是将从机的移位寄存器数据传给主机的移位寄存器，实现数据的交换。

3 SPI 实现方式

3.1 硬件 SPI

APM32F103 系列开发板本身自带硬件电路设计而成的硬件 SPI。在通信过程中，其依赖于内置的 SPI 外设，由硬件直接管理数据的传输过程，无需额外编程来控制 SPI 通信过程中的各个步骤。

3.1.1 硬件 SPI 使用的优缺点

优点：

（1） 可以提供更高的通信速度和一致性

（2） 可以实现更高的通信模式。如 DMA 传输、中断处理和主机模式。

（3） 代码复杂度低，易于操作。

缺点：

（1） 硬件 SPI 的配置和功能是固定的，不易修改。

（2） 硬件 SPI 依赖于特定的硬件控制器，不同平台开发存在兼容性问题。

3.2 GPIO 模拟 SPI

在开发过程中，为了规避因硬件 SPI 不足而导致开发受阻的情况，可以通过 GPIO 口和软件控制来实现模拟 SPI 的通信过程。

3.2.1 GPIO 模拟 SPI 使用的优缺点

优点：

（1） GPIO 引脚配置灵活，可以任意选用开发板中 GPIO 引脚来进行模拟操作。

（2） 成本低，软件模拟 SPI 不需要额外的硬件设计，只需要有 GPIO 引脚即可。

缺点：

（1） 通信速度低，受困于 mcu 的处理能力。

（2） 代码复杂度高，需要更多的资源。

4 Flash 介绍

Flash，全称是 Flash Memory Controller，即闪存存储器控制器。FLASH 通常用于存储程序代码、数据等信息。Flash 包括内部 Flash 和外部 Flash。在本文中采用到的 W25Q64 模块，便是一个 SPI 接口的外部 Flash 模块。在运行过程中，Flash 存放的代码不会进行修改，且 Flash 按照扇区来进行操作。

Flash 在写入之前数据前，通常需要进行擦除操作。因为 Flash 要写入数据时，其无法直接在已经被编程的扇区上进行修改，而是需要将整个扇区擦除成初始状态，然后在进行写入操作。

4.1 W25Q64 介绍

W25Q64 是一款基于外设接口（SPI）的闪存芯片，它具有 8Mbyte 的存储空间，是一款大容量的 SPI Flash 产品。W25Q64 将 8M 的字节容量分为 128 个块，每个块大小为 64K，每个块又分为 16 个扇区，每个扇区有 4K 大小。W25Q64 的最小擦除单位是扇区，这需要给其开辟一个至少 4K 的缓存区。

4.1.1 W25Q64 常见命令指令

通过 SPI 向 W25Q64 发送指令，即可操作 W25Q64。本文中操作所用到的一些指令表，如下表 1 所示：

Data Input Output	Byte1	Byte2	Byte3	Byte4	Byte5	Byte6	Byte7	Byte8
Number of Clock	8	8	8	8	8	8	8	8
Write Enable	06h	—						
Write Disable	04h	—						
Volatile SR Write Enable	50h	—						
Read Data	03h	A23-A16	A15-A8	A7-A0	（D7-D0）	—		
Page Program	02h	A23-A16	A15-A8	A7-A0	（D7-D0）	—		
Sector Erase（4KB）	20h	A23-A16	A15-A8	A7-A0	—			
Block Erase（32KB）	52h	A23-A16	A15-A8	A7-A0	—			
Block Erase（64KB）	D8h	A23-A16	A15-A8	A7-A0	—			

表格 1 W25Q64 常见操作命令

其中，在页编程操作里。首先要拉低 CS 片选，随后的 8 个时钟周期里，指令 0x02 由 DI 发出，之后，24 个时钟周期里，发送 24Bit 的地址。在最后的 256*8 个时钟周期里，发送 256Byte 数据。其时序图如下图 6 所示：

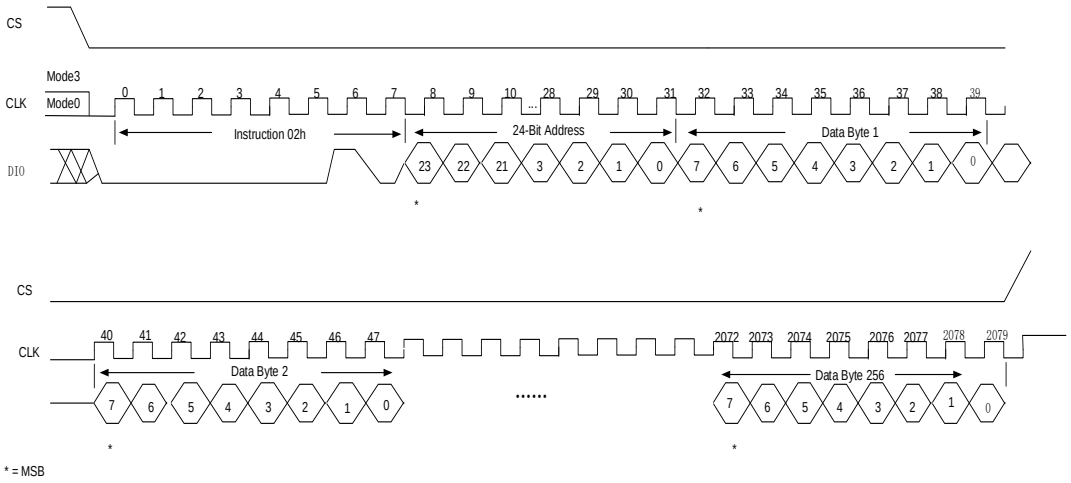

图 6 W25Q64 命令操作图

4.1.2 W25Q64 使用的优缺点

优点：

（1） 存储空间大

（2） 支持高速的串行读写操作，具有较快的数据传输速率

（3） 功耗低

（4） 采用 SPI 接口进行通信，容易实现连接和控制。

缺点：

（1） 价格高

（2） 有限的使用寿命

（3） 擦除操作慢

5 APM32F103 GPIO 模拟 SPI 读写外部 Flash 例程

5.1 硬件设计

本次实验所需要的硬件设计：

（1） 外部 Flash 模块

（2） USB 转 TTL 线

外部 Flash 模块使用的是 W25Q64 型号，W25Q64 模块和 APM32F103 开发板的硬件接线图如下图 7 所示：

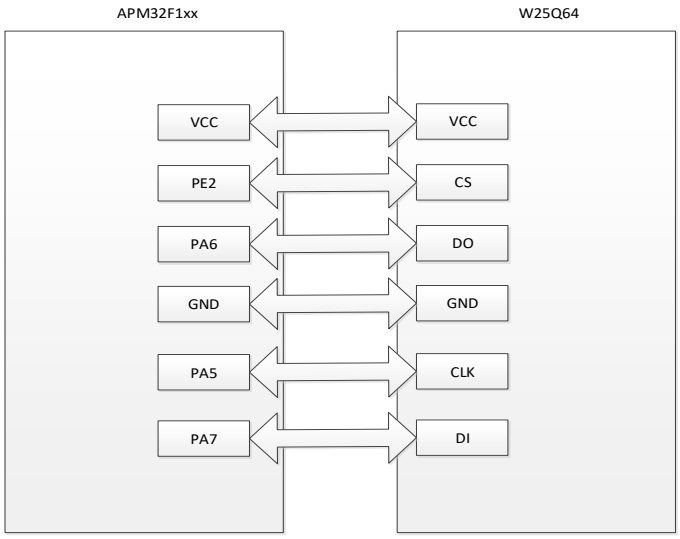

图 7 硬件接线图

USB 转 TTL 线所使用到的 GPIO 管脚为 PA9，PA10，利用 USB 转 TTL 线进行数据的接收和发送。其原理图如下图 8 所示：

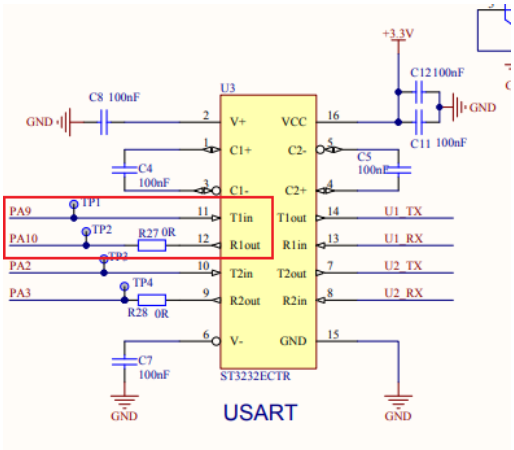

图 8 USART 接线原理图

5.2 软件设计

通过 APM32F103 实现 GPIO 模拟 SPI 读写外部 Flash，需要根据模块的型号配置好外部 Flash 的 ID 信息，并且完成 GPIO 的管脚配置设计以及 SPI 通信时序，完成之后即可进行读写操作，并对读写操作完成结果输出。代码实现流程图如下图 9 所示：

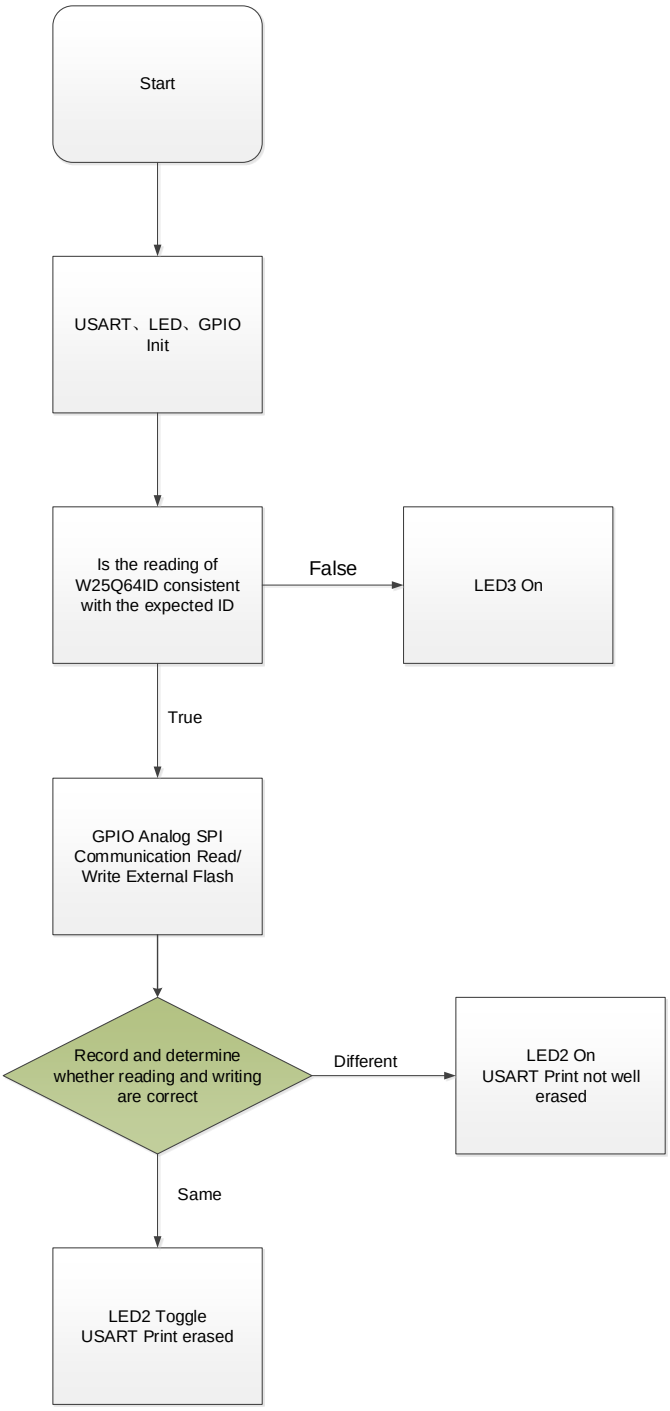

图 9 主程序流程图

5.2.1 GPIO 引脚配置

对模拟 SPI 所使用到的 GPIO 管脚，将 SCK、CS、MOSI 管脚配置为推挽输出模式，MISO 配置为下拉输入模式。参考代码如下：

```
void SPI_Init（void）
    {
        GPIO_Config_T   GPIO_ConfigStruct;

        RCM_EnableAPB2PeriphClock（RCM_APB2_PERIPH_GPIOA）;
        RCM_EnableAPB2PeriphClock（RCM_APB2_PERIPH_GPIOE）;

        GPIO_ConfigStruct.pin           = sFLASH_CS_PIN ;
        GPIO_ConfigStruct.mode          = GPIO_MODE_OUT_PP;
        GPIO_ConfigStruct.speed         = GPIO_SPEED_2MHz;
        GPIO_Config（GPIOE, &GPIO_ConfigStruct）; // SCK CS MOSI Output

        GPIO_ConfigStruct.pin           = sFLASH_SPI_SCK_PIN | sFLASH_SPI_MOSI_PIN ;
        GPIO_ConfigStruct.mode          = GPIO_MODE_OUT_PP;
        GPIO_ConfigStruct.speed         = GPIO_SPEED_2MHz;
        GPIO_Config（GPIOA, &GPIO_ConfigStruct）; // SCK CS MOSI Output

        GPIO_ConfigStruct.pin           = sFLASH_SPI_MISO_PIN;
        GPIO_ConfigStruct.mode          = GPIO_MODE_IN_FLOATING;
        GPIO_ConfigStruct.speed         = GPIO_SPEED_2MHz;
        GPIO_Config（GPIOA, &GPIO_ConfigStruct）; // MISO Input
        sFLASH_CS_HIGH;         // CS Init High
        sFLASH_SCK_LOW;         // SCK Init Low
    }
```

5.2.2 模拟 SPI 时序读写操作

如上图 5 数据交换图可知，SPI 的传输可以看作一个虚拟的环形拓扑结构，表示输入和输出是在同一时间下进行。当主机发送时钟信号，并拉低片选信号后选择从机进行发送。根据图 2 SPI 传输模式和图 3 SPI 传输时序图处介绍，此处选择 SPI 模式 0，时钟空闲状态为低电平，数据在上升沿处采集，在下降沿处移出。参考代码如下：

```
uint8_t sFLASH_TxRxOneByte（uint8_t byte）
    {
        uint8_t temp = 0;
        uint8_t rx_data = 0xFF;
        for（uint8_t j=0;j<8;j++）
        {
            temp =  （（byte&0x80）==0x80）? 1:0;
            byte = byte<<1;
            rx_data = rx_data<<1;

            sFLASH_SCK_LOW;
            if（temp == 1）
            {
                sFLASH_MOSI_HIGH;
            }
            else
            {
                sFLASH_MOSI_LOW;
            }
            Delay（10）;
            sFLASH_SCK_HIGH;
            Delay（10）;
            if（sFLASH_MISO_READ == 1）
            {
                rx_data = rx_data + 1;
            }
        }
        sFLASH_SCK_LOW;
        return rx_data;
    }
```

根据代码得知：

（1） 在每次读和写一个 Byte 时，都会循环 8 次，每次接收和发送一个 Bit。在循环中，每次将 Byte 的最高位存放到 temp 变量中；

（2） 将 Byte 的最高位存放好后，便会将 Byte 的次高位变为最高位，这里采取的操作便是将 Byte 左移一位。

（3） 将 rx_data 右移一位，挪动最低位，8 次循环后，rx_data 将高位在前。

（4） 此时，将时钟的空闲电平设置为低电平，所采用的操作便是拉低时钟线。

（5） 同时，在根据 Byte 当前最高位所存放在 temp 变量中的值，设置 MOSI 的相关引脚电平。

（6） 之后，将时钟的空闲电平设置为高电平，所采用的操作便是拉高时钟线。然后，从设备开始读取 MOSI 传递的数据，并将读取到的数据写入到 MISO 上。

（7） rx_data 变量中的最低位用来存储 MISO 上读取到的数据。

（8） SPI 读写结束后，将时钟的空闲电平设置为低电平，拉低时钟线，此时，SPI 进入空闲状态。

5.2.3 W25Q64 设备信息读取操作

W25Q64 主要有三个 ID：制造商编号 ID（Manufacturer）、设备 ID（Device ID）、唯一表示 ID（Unique ID）。其中，制造商编号 ID 是由 Winbond 生产的，它的值 MF[7:0]为 0xEF。设备 ID 包含两部分，ID[15:0]是芯片类型，为 0x4017。ID[7:0]是存储容量。最后，唯一标识 ID 表示芯片的唯一性，常用于加密。其 W25Q64 ID 内容介绍如下表 2 所示，操作开发板读取 W25Q64 ID 如下表 3 所示：

Type And Instruct	Address Content and Operation Instruction	
MANUFACTURER ID	（MF7-MF0）	-
Winbond Serial Flash	EFh	
Device ID	（ID7-ID0）	（ID15-ID0）
Instruction	ABh,90h,92h,94h	9Fh
W25Q64JV-IQ/JQ	16h	4017h

表格 2 W25Q64 ID 内容介绍

Read Type	Instruct	Address Content				
Release Power-down/ID	ABh	Dummy	Dummy	Dummy	（ID7-ID0）	-
Manufacturer/Device ID	90h	Dummy	Dummy	00h	（MF7-MF0）	（ID7-ID0）
JEDEC ID	9Fh	（MF7-MF0）	（ID15-ID8）	（ID7-ID0）	-	-
Read Unique ID	4Bh	Dummy	Dummy	Dummy	Dummy	（UID63-0）

表格 3 W25Q64 读取 ID 操作指令

在代码实现过程中，使用 0xAB 指令获取设备 ID[7:0]，使用 0x9F 指令获取 JEDEC ID（MF[7:0] + ID[15:0]）。在实现过程中，均要先把/CS 引脚拉低，然后分别把”ABh”、”9Fh”通过 DI 发送到芯片。在发送完”ABh”后，会经过一段 tRES1 时间间隔，芯片便恢复正常工作。在这当中，编程、擦除和写状态寄存器指令执行周期内，执行该指令无效。在发送完”9Fh”后，3 个 ID 信息会分别从 DO 管脚在 SCK 的下降沿发送出去。参考代码如下：

```
uint32_t sFLASH_ReadDeviceID（void）
{
    uint32_t Temp[4];

    sFLASH_CS_LOW;
    sFLASH_TxRxOneByte（0xAB）;
    Temp[0] = sFLASH_TxRxOneByte（sFLASH_DUMMY_BYTE）;
    Temp[1] = sFLASH_TxRxOneByte（sFLASH_DUMMY_BYTE）;
    Temp[2] = sFLASH_TxRxOneByte（sFLASH_DUMMY_BYTE）;
    Temp[3] = sFLASH_TxRxOneByte（sFLASH_DUMMY_BYTE）;
    sFLASH_CS_HIGH;

    return Temp[3];
}
```

```
uint32_t sFLASH_ReadID（void）
{
    uint32_t Temp = 0, Temp0 = 0, Temp1 = 0, Temp2 = 0;

    sFLASH_CS_LOW;
    sFLASH_TxRxOneByte（0x9F）;
    Temp0 = sFLASH_TxRxOneByte（sFLASH_DUMMY_BYTE）;
    Temp1 = sFLASH_TxRxOneByte（sFLASH_DUMMY_BYTE）;
    Temp2 = sFLASH_TxRxOneByte（sFLASH_DUMMY_BYTE）;
    sFLASH_CS_HIGH;

    Temp = （Temp0 << 16） | （Temp1 << 8） | Temp2;
    return Temp;
}
```

5.2.4 扇区擦除操作

根据表格 1 W25Q64 常见操作命令。可知，发送 0xD8 指令，在发送 24 位的扇区地址，对整个块里面的扇区内容进行全擦除，在擦除完之后，所有字节均变为”FFh”，参考代码如下：

```
void sFLASH_SectorErase（uint32_t SectorAddr）
{
    sFLASH_WriteEnable（）;

    sFLASH_CS_LOW;
    sFLASH_TxRxOneByte（0xD8）;
    sFLASH_TxRxOneByte（（SectorAddr & 0xFF0000） >> 16）;
    sFLASH_TxRxOneByte（（SectorAddr & 0xFF00） >> 8）;
    sFLASH_TxRxOneByte（SectorAddr & 0xFF）;
    sFLASH_CS_HIGH;

    /* Wait FLASH completed */
    sFLASH_WaitForWriteEnd（）;
}
```

5.2.5 页写入数据操作

W25Q64 只能按页写数据，每页数据为 256 字节，因此一次最多写 256 字节。根据表格 1 W25Q64 常见操作命令。可知，进行写入数据时，需要先发送 0x02 指令，在发送 24 位的扇区地址，即可对该扇区进行写入操作。参考代码如下：

```
void sFLASH_PageProgram（uint8_t* pBuffer, uint32_t WriteAddr, uint16_t NumByteToWrite）
    {
        sFLASH_WriteEnable（）;

        sFLASH_CS_LOW;

        sFLASH_TxRxOneByte（sFLASH_CMD_WRITE）;
        sFLASH_TxRxOneByte（（WriteAddr & 0xFF0000） >> 16）;
        sFLASH_TxRxOneByte（（WriteAddr & 0xFF00） >> 8）;
        sFLASH_TxRxOneByte（WriteAddr & 0xFF）;
        while （NumByteToWrite--）
        {
            sFLASH_TxRxOneByte（*pBuffer）;
            pBuffer++;
        }
        sFLASH_CS_HIGH;
        sFLASH_WaitForWriteEnd（）;
    }
```

5.2.6 读取数据操作

根据表格 1 W25Q64 常见操作命令。可知，进行读取数据时，需要先发送 0x03 指令，在发送 24 位的扇区地址，即可对该扇区进行读取操作。参考代码如下：

```
void sFLASH_ReadData（uint8_t* pBuffer, uint32_t ReadAddr, uint16_t NumByteToRead）
{
    sFLASH_CS_LOW;

    sFLASH_TxRxOneByte（sFLASH_CMD_READ）;
    sFLASH_TxRxOneByte（（ReadAddr & 0xFF0000） >> 16）;
    sFLASH_TxRxOneByte（（ReadAddr& 0xFF00） >> 8）;
    sFLASH_TxRxOneByte（ReadAddr & 0xFF）;
    while （NumByteToRead--）
    {
        *pBuffer = sFLASH_TxRxOneByte（sFLASH_DUMMY_BYTE）;
        pBuffer++;
    }
    sFLASH_CS_HIGH;
}
```

5.2.7 等待 Flash 写完成操作

W25Q64 有三个状态寄存器，bit[0]代表的是擦除/写入状态标志位。根据表格 1 W25Q64 常见操作命令。可知，当写入或者读取操作结束时，需要对状态寄存器进行判断，发送”05h”数据获取状态寄存器的数据，当状态寄存器 bit[0]的值为 1 时，还在进行擦除或者写入操作，当该位为 0 时，表示这时没有任何操作在进行，可以对 W25Q64 进行擦除或者写入操作。参考代码如下：

```
void sFLASH_WaitForWriteEnd（void）
    {
        uint8_t flashstatus = 0;

        sFLASH_CS_LOW;
        sFLASH_TxRxOneByte（0x05）;
        do
        {
            flashstatus = sFLASH_TxRxOneByte（sFLASH_DUMMY_BYTE）;
        }
        while （（flashstatus & sFLASH_WIP_FLAG） == SET）;
        sFLASH_CS_HIGH;
    }
```

5.2.8 主函数逻辑操作

本例程功能介绍：下载程序后，对灯、串口、GPIO 模拟 SPI 的管脚进行初始化和使能操作。之后，对 FlashID 和 DeviceID 进行读取操作，读取结束之后会根据判断条件进入不同的状态。如果 FlashID 读取的数值跟定义好的 ID 一致，将会进行 Flash 的读写操作。如果不一致，LED3 将会常亮。在 Flash 的读写操作中，进入写操作之前，先进行一次擦除操作，之后会分别进行写和读数据。操作结束之后，将读取和写入的数据进行对比并放到一个变量 1 中。之后，进行一次擦除操作，并在进行一次读取操作，与 OXFF 进行一次对比，并将对比放在变量 2 中，用于判断是否擦除成功。最后，将变量所得进行一一对比，并且串口和灯会分别做出不同的输出。参考代码如下：

```
int main（void）
{
    USART_Config_T usartConfig;

    APM_MINI_LEDInit（LED2）;
    APM_MINI_LEDInit（LED3）;
    APM_MINI_LEDOff（LED2）;
    APM_MINI_LEDOff（LED3）;

    usartConfig.baudRate = 115200;
    usartConfig.mode = USART_MODE_TX_RX;
    usartConfig.parity = USART_PARITY_NONE;
    usartConfig.stopBits = USART_STOP_BIT_1;
    usartConfig.wordLength = USART_WORD_LEN_8B;
    usartConfig.hardwareFlow = USART_HARDWARE_FLOW_NONE;
    APM_MINI_COMInit（COM1,&usartConfig）;

    //Init SPI GPIO
    SPI_Init（）;
    // Read SPI Device ID
    DeviceID = sFLASH_ReadDeviceID（）;
    /* Read SPI Flash ID */
    FlashID = sFLASH_ReadID（）;

    printf（"Device ID: 0x%x\r\n",DeviceID）;
    printf（"Flash ID: 0x%x\r\n",FlashID）;

    //whether is flash id
    if （FlashID == sFLASH_ID）
    {
        APM_MINI_LEDOn（LED2）;
        //erase exist
        sFLASH_SectorErase（FLASH_SectorToErase）;
```

```
        //write data
        sFLASH_PageProgram（Tx_Buffer, FLASH_WriteAddress, BufferSize）;
        //read data
        sFLASH_ReadData（Rx_Buffer, FLASH_ReadAddress, BufferSize）;
        //compare write and read
        TransferStatus1 = Buffercmp（Tx_Buffer, Rx_Buffer, BufferSize）;
        //erase operate
        sFLASH_SectorErase（FLASH_SectorToErase）;
        //read again for check whether erase success
        sFLASH_ReadData（Rx_Buffer, FLASH_ReadAddress, BufferSize）;
        //compare
        for （Index = 0; Index < BufferSize; Index++）
        {
            if （Rx_Buffer[Index] != 0xFF）
            {
                TransferStatus2 = FAILED;
            }
        }
    }
    //if not flash id ,led3 on
    else
    {
        APM_MINI_LEDOn（LED3）;
    }
    //if erase success
    if（TransferStatus2 == PASSED）
    {
        printf（"the specified sector part is erased!!!\r\n"）;
    }//not erase success
    else{
        printf（"the specified sector part is not well erased!!!\r\n"）;
    }
    while（1）
    {
        //compare for write and read whether same
        if（TransferStatus1 == PASSED）
        {
            APM_MINI_LEDToggle（LED2）;
        }
        Delay（0xfffff）;
    }
}
```

6 版本历史

表格 4 文件版本历史

日期	版本	变更历史
2023.08.29	1.0	新建

声明

本手册由珠海极海半导体有限公司（以下简称“极海”）制订并发布，所列内容均受商标、著作权、软件著作权相关法律法规保护，极海保留随时更正、修改本手册的权利。使用极海产品前请仔细阅读本手册，一旦使用产品则表明您（以下称“用户”）已知悉并接受本手册的所有内容。用户必须按照相关法律法规和本手册的要求使用极海产品。

1、权利所有

本手册仅应当被用于与极海所提供的对应型号的芯片产品、软件产品搭配使用，未经极海许可，任何单位或个人均不得以任何理由或方式对本手册的全部或部分内容进行复制、抄录、修改、编辑或传播。

本手册中所列带有“®”或“TM”的“极海”或“Geehy”字样或图形均为极海的商标，其他在极海产品上显示的产品或服务名称均为其各自所有者的财产。

2、无知识产权许可

极海拥有本手册所涉及的全部权利、所有权及知识产权。

极海不应因销售、分发极海产品及本手册而被视为将任何知识产权的许可或权利明示或默示地授予用户。

如果本手册中涉及任何第三方的产品、服务或知识产权，不应被视为极海授权用户使用前述第三方产品、服务或知识产权，除非在极海销售订单或销售合同中另有约定。

3、版本更新

用户在下单购买极海产品时可获取相应产品的最新版的手册。

如果本手册中所述的内容与极海产品不一致的，应以极海销售订单或销售合同中的约定为准。

4、信息可靠性

本手册相关数据经极海实验室或合作的第三方测试机构批量测试获得，但本手册相关数据难免会出现校正笔误或因测试环境差异所导致的误差，因此用户应当理解，极海对本手册中可能出现的该等错误无需承担任何责任。本手册相关数据仅用于指导用户作为性能参数参照，不构成极海对任何产品性能方面的保证。

用户应根据自身需求选择合适的极海产品，并对极海产品的应用适用性进行有效验证和测试，以确认极海产品满足用户自身的需求、相应标准、安全或其它可靠性要求；若因用户未充分对极海产品进行有效验证和测试而致使用户损失的，极海不承担任何责任。

5、合规要求

用户在使用本手册及所搭配的极海产品时，应遵守当地所适用的所有法律法规。用户应了解产品可能受到产品供应商、极海、极海经销商及用户所在地等各国有关出口、再出口或其它法律的限制，用户（代表其本身、子公司及关联企业）应同意并保证遵守所有关于取得极海产品及／或技术与直接产品的出口和再出口适用法律与法规。

6、免责声明

本手册由极海“按原样”（as is）提供，在适用法律所允许的范围内，极海不提供任何形式的明示或暗示担保，包括但不限于对产品适销性和特定用途适用性的担保。

对于用户后续在针对极海产品进行设计、使用的过程中所引起的任何纠纷，极海概不承担责任。

7、责任限制

在任何情况下，除非适用法律要求或书面同意，否则极海和/或以“按原样”形式提供本手册的任何第三方均不承担损害赔偿责任，包括任何一般、特殊因使用或无法使用本手册相关信息而产生的直接、间接或附带损害（包括但不限于数据丢失或数据不准确，或用户或第三方遭受的损失）。

8、适用范围

本手册的信息用以取代本手册所有早期版本所提供的信息。