不讲理论的STM32教程

进阶部分

第十章：SPI通信

SPI的介绍：

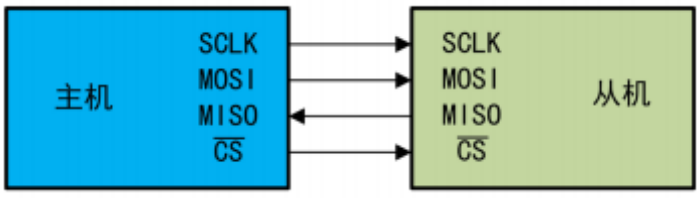
上一章节我们学习了IIC通信，本节课我们继续来学一下SPI通信

SPI简介：

SPI（Serial Peripheral Interface，串行外设接口）是由摩托罗拉（Motorola）在1980前后提出的一种全双工同步串行通信接口，它用于MCU与各种外围设备以串行方式进行通信以交换信息，通信速度最高可达25MHz以上。

SPI接口主要应用在EEPROM、FLASH、实时时钟、网络控制器、OLED显示驱动器、AD转换器，数字信号处理器、数字信号解码器等设备之间。

SPI通常由四条线组成：



一条主设备输出与从设备输入（Master Output Slave Input，MOSI）

一条主设备输入与从设备输出（Master Input Slave Output，MISO）

一条时钟信号（Serial Clock，SCLK）

一条从设备使能选择（Chip Select，CS）

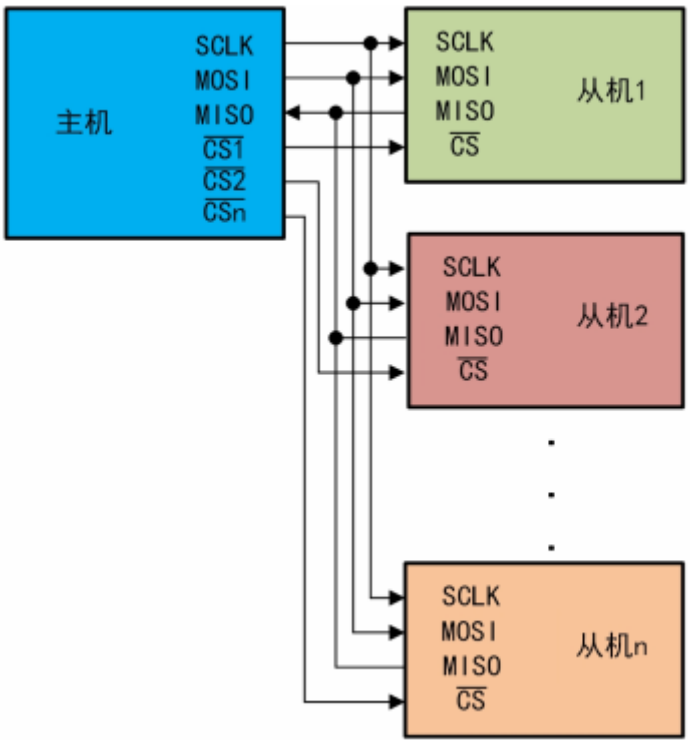
与I²C类似，协议都比较简单，也可以使用GPIO模拟SPI时序。

下面做了一个SPI和I²C对比表格

功能说明	**SPI总线**	**I2C总线**
通信方式	同步 串行 全双工	同步 串行 半双工
通信速度	一般50MHz以下	100KHz、400KHz、3.4MHz
从设备选择	引脚片选	设备地址片选
总线接口	MOSI、MISO、SCK、CS	SDA、SCL

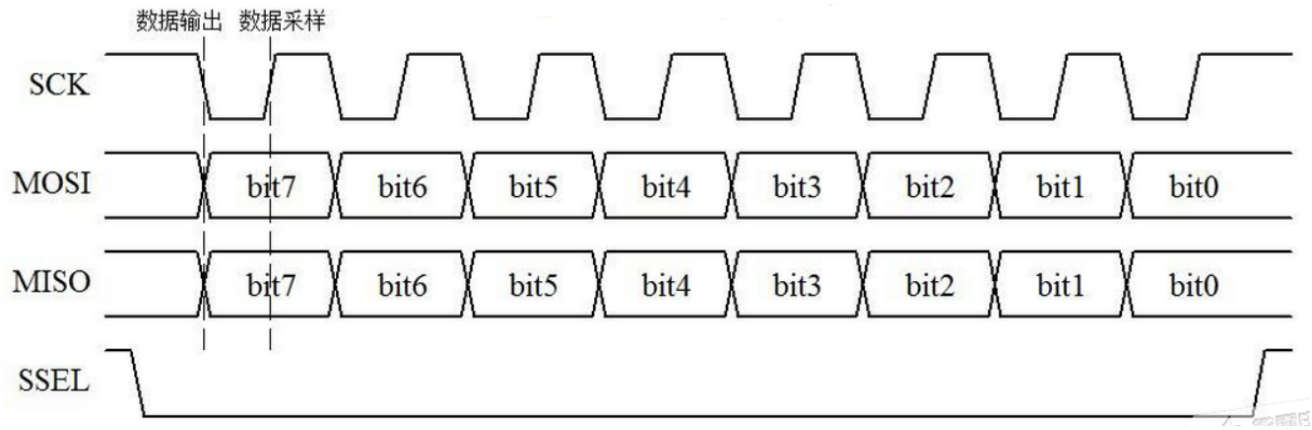
SPI可以同时发出和接收数据，因此SPI的理论传输速度比I²C更快。SPI通过片选引脚选择从机，一个片选一个从机，因此在多从机结构中，需要占用较多引脚，而I²C通过设备地址选择从机，只要设备地址不冲突，始终只需要两个引脚

这里有一个多从机的示意图



可以看出来，我们每一个从机，都需要一个独立的CS（设备使能选择，Chip Select），当对应的CS为低电平时候，就是给这一个从机发送信息，当我们的CS线为高电平时代表我们的数据发送结束

通信流程：



这里流程示意图，我们现在逐一分析每一个阶段

- 起始信号：NSS（CS）信号线由高变低，是SPI通讯的起始信号。NSS（CS）是每个从机各自独占的信号线，当从机检在自己的NSS（CS）线检测到起始信号后，就知道自己被主机选中了，开始准备与主机通讯
- 结束信号：NSS信号由低变高，是SPI通讯的停止信号，表示本次通讯结束，从机的选中状态被取消

采样阶段：

我们的SPI通信相比IIC更加高级，有四种不同的采样模式，需要我们自己去选择是由两个参数“时钟极性（CPOL）”及“相位（CPHA）”来决定，都有0和1两张状态，组合起来使用就是4种模式

根据CPOL和CPHA的不同，SPI有四种工作模式

时钟极性(CPOL)定义了时钟空闲状态电平：

CPOL=0时钟空闲时为低电平

CPOL=1时钟空闲时为高电平

时钟相位(CPHA)定义数据的采集时间：

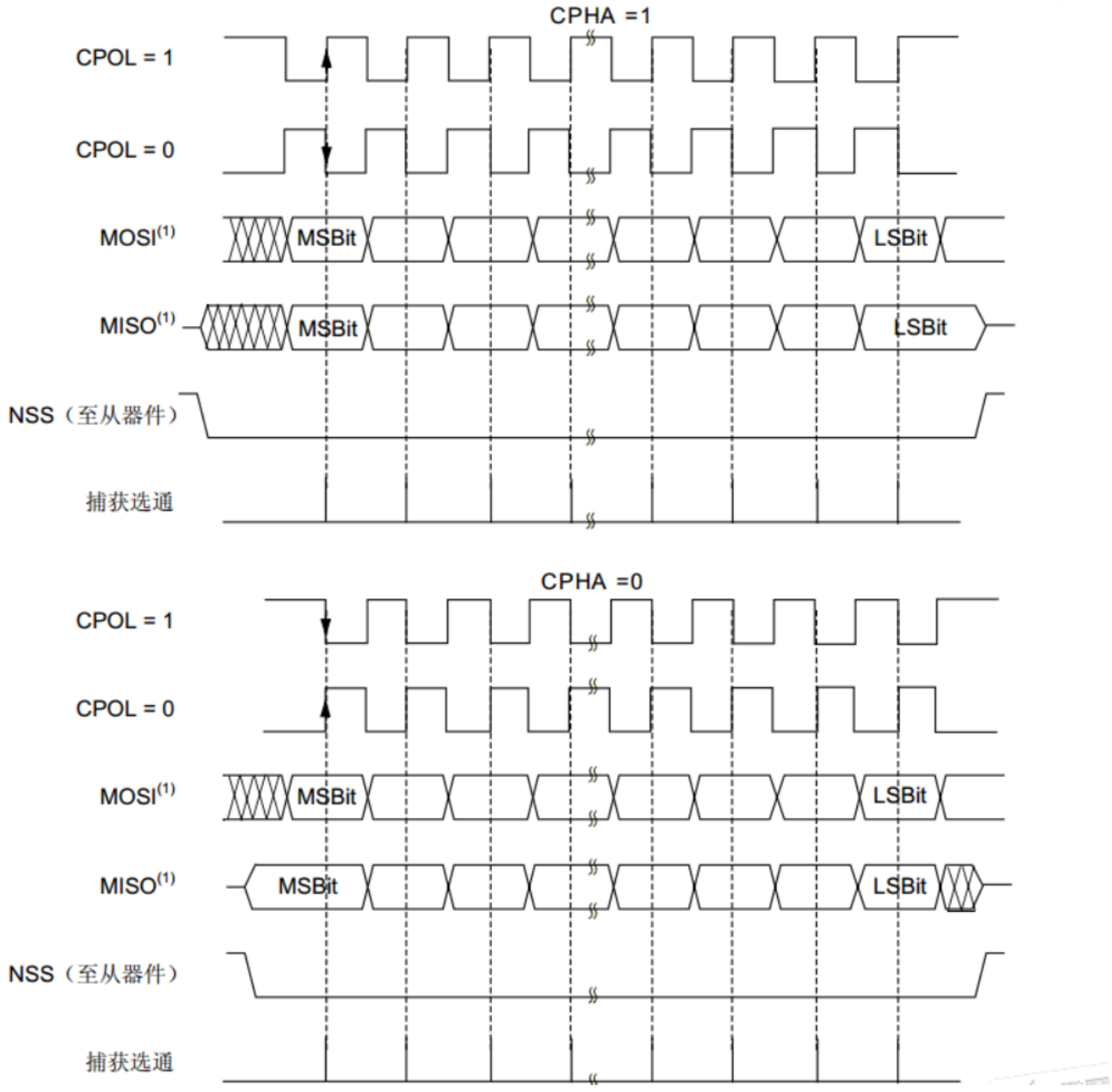
CPHA=0:在时钟的奇数跳变沿（上升沿或下降沿）进行数据采样

CPHA=1:在时钟的偶数跳变沿（上升沿或下降沿）进行数据采样

用表格整理一下就是：

SPI模式	CPOL	CPHA	空闲SCK时钟	采样时刻
0	0	0	低电平	奇数边沿
1	0	1	低电平	偶数边沿
2	1	0	高电平	奇数边沿
3	1	1	高电平	偶数边沿

还是可能有点抽象，这里阿熊找了一张示意图给大家



相信这样一看，就非常清楚了

虽然我们有四种模式，但是大体就是两种效果，上升沿采样，或者下降沿采样，要注意的是我们的主机和从机的SPI的模式必须要有才可以正常通信哦！

数据读取应该就不用过多介绍，就是在我们选好的上升沿或者下降沿进行数据的读取

另外我们的SPI除了CPOL和CPHA外还有一些参数配置，这里就简单的给大家提一下，具体的解释会在后面的项目实战给大家讲解

我们前面介绍了SPI的4中采样模式，其实SPI自己也是有着很多种模式的，这里简单的给大家列举一下：

Full-Duplex Master全双工主机

Full-Duplex Slave全双工从机

Half-Duplex Master半双工主机

Half-Duplex Slave半双工从机

Receive Only Master只接收型主机

Receive Only Slave只接收型从机

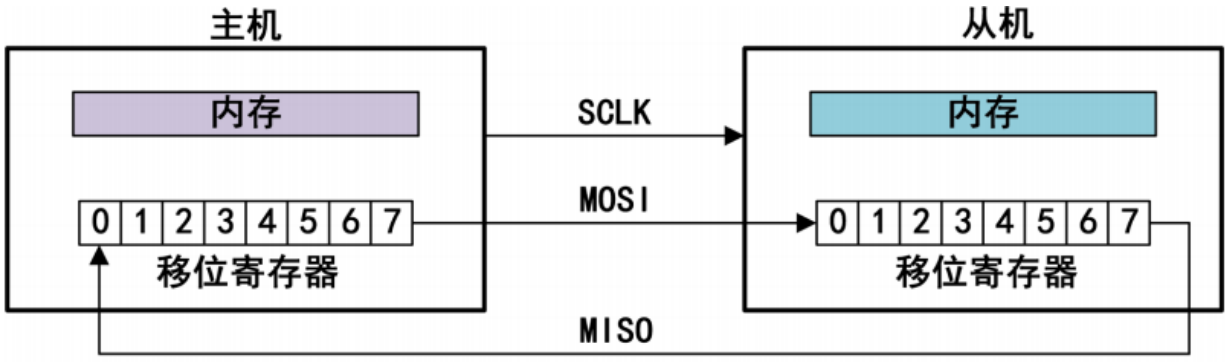
Transmit Only Master只传输型主机

其实字面意思上就很清楚了，就是全双工，半双工，单项传输三种模式下的主机从机，一般最常用的就是第一种

然后就是数据传输的大小，可以选择8位或者16位，这个的选择要看具体的需求，一般都是默认选择8位

除了数据大小的选择还有数据传输顺序，是高位传输还是低位传输，这个应该也不难理解，就是我们读取或者输出数据时，我们时送高位开始输出还是低位开始输出，可以供我们选择，一般情况都是从高位开始传输

由于我们的SPI发送和接收是同时进行的，这里就需要让大家知道我们的SPI的数据交换是怎么进行的



这里有一个简单的示意图，主机和从机都有一个移位寄存器，主机移位寄存器数据经过MOSI将数据写入从机的移位寄存器，此时从机移位寄存器的数据也通过MISO传给了主机，实现了两个移位寄存器的数据交换。无论主机还是从机，发送和接收都是同时进行的，如同一个“环”。

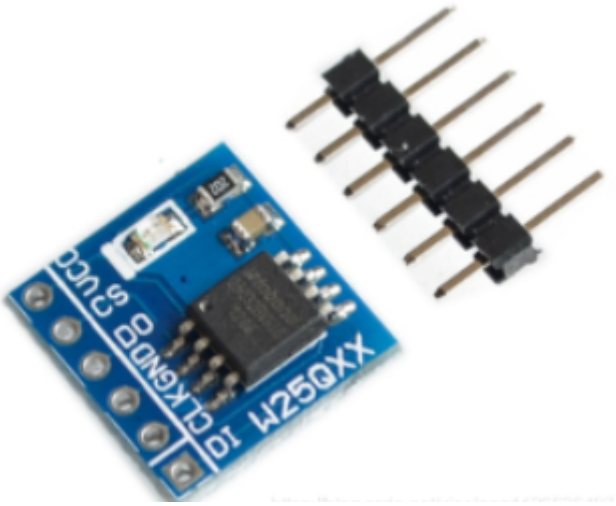
如果主机只对从机进行写操作，主机只需忽略接收的从机数据即可。如果主机要读取从机数据，需要主机发送一个空数据来引发从机发送数据

SPI的使用：

SPI最常用的个就是我们的FLASH存储芯片，本次SPI的使用就是以我们的W25Q64存储芯片为例，教大家SPI最基本的使用

这里先介绍一下我们的W25Q64：

W25Q64简介：



W25Q64是为系统提供一个最小空间、最少引脚，最低功耗的串行Flash存储器，25Q系列比普通的串行Flash存储器更灵活，性能更优越。

W25Q64支持双倍/四倍的SPI，可以储存包括声音、文本、图片和其他数据；芯片支持的工作电压 2.7V 到 3.6V，正常工作时电流小于5mA，掉电时低于1uA，所有芯片提供标准的封装。

W25Q64的内存空间结构：一页256字节，4K(4096 字节)为一个扇区，16个扇区为1块，容量为8M字节，共有128个块，2048 个扇区。

W25Q64每页大小由256字节组成，每页的256字节用一次页编程指令即可完成。

擦除指令分别支持：16页（1个扇区）、128页、256页、全片擦除。

W25Q64支持标准串行外围接口（SPI），和高速的双倍/四倍输出，双倍/四倍用的引脚：串行时钟、片选端、串行数据 I/O0(DI)、I/O1(DO)、I/O2(WP)和 I/O3(HOLD)。SPI 最高支持 80MHz，当用快读双倍/四倍指令时，相当于双倍输出时最高速率160MHz，四倍输出时最高速率320MHz。这个传输速率比得上8位和16位的并行Flash存储器。

- 其同系列产品：

-W25Q16:16M 位/2M 字节

-W25Q32:32M 位/4M 字节

-W25Q64:64M 位/8M 字节

-W25Q128:128M 位/16M 字节

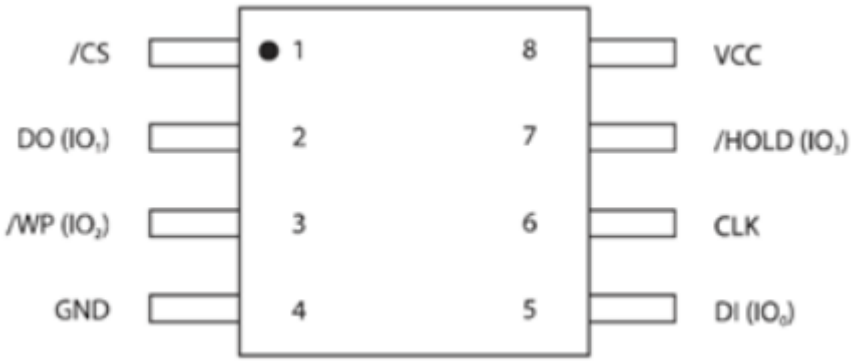
- 灵活的4KB扇区结构
  -统一的扇区擦除（4K 字节）
  -块擦除（32K 和 64K 字节）
  -一次编程 256 字节
  -至少 100,000 写/擦除周期
  -数据保存 20 年

- 标准、双倍和四倍SPI
  -标准 SPI:CLK、CS、DI、DO、WP、HOLD
  -双倍 SPI:CLK、CS、IO0、IO1、WP、HOLD

-四倍 SPI:CLK、CS、IO0、IO1、IO2、IO3

- 高级的安全特点
  -软件和硬件写保护
  -选择扇区和块保护
  -一次性编程保护(1)
  -每个设备具有唯一的64位ID(1)
- 高性能串行Flash存储器
  -比普通串行Flash性能高6倍
  -80MHz时钟频率
  -双倍SPI相当于160MHz
  -四倍SPI相当于320MHz
  -40MB/S连续传输数据
  -30MB/S随机存取（每32字节）
  -比得上16位并行存储器
- 低功耗、宽温度范围
  -单电源 2.7V-3.6V
  -工作电流 4mA，掉电<1μA（典型值）
  -40℃～+85℃工作

引脚介绍:



引脚编号	引脚名称	**I/O**	功能
1	**/CS**	I	片选端输入
2	**DO(IO1)**	I/O	数据输出（数据输入输出 **1**）
3	**/WP(IO2)**	I/O	写保护输入（数据输入输出 **2**）
4	**GND**	\	地
5	**DI(IO0)**	I/O	数据输入（数据输入输出 **0**）

引脚编号	引脚名称	I/O	功能
6	**CLK**	I	串行时钟输入
7	**/HOLD(IO3)**	I/O	保持端输入（数据输入输出 3）
8	**VCC**	\	电源

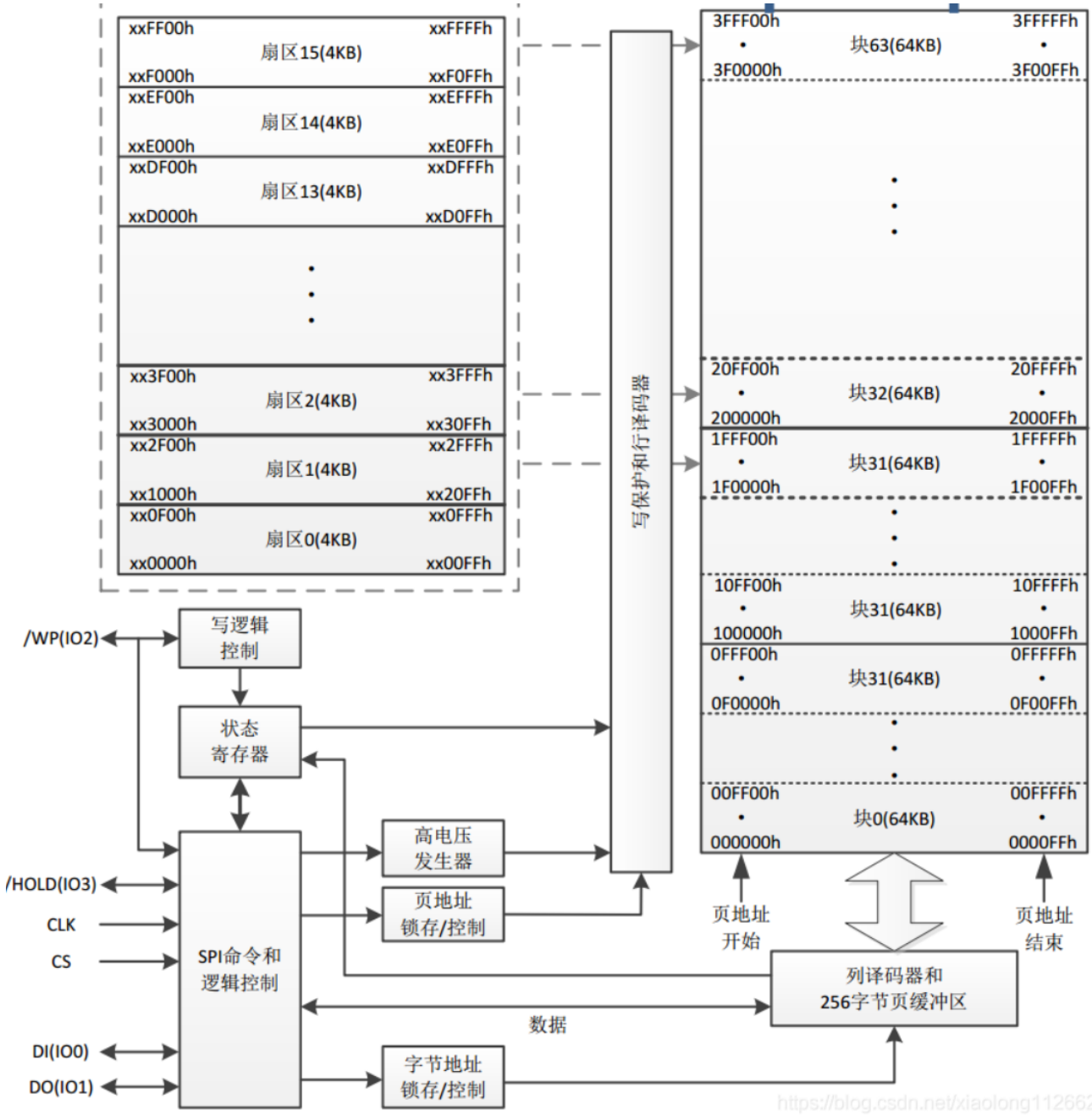
- 写保护（/WP）

写保护引脚（/WP）用来保护状态寄存器。和状态寄存器的块保护位（SEC、TB、BP2、BP1 和 BP0）和状态寄存器保护位（SRP）对存储器进行一部分或者全部的硬件保护。/WP 引脚低电平有效。当状态寄存器 2 的 QE 位被置位了，/WP 引脚（硬件写保护）的功能不可用

- 保持端（/HOLD）

当/HOLD 引脚是有效时，允许芯片暂停工作。在/CS 为低电平时，当/HOLD 变为低电平，DO 引脚将变为高阻态，在 DI 和 CLK 引脚上的信号将无效。当/HOLD 变为高电平，芯片恢复工作。/HOLD 功能用在当有多个设备共享同一 SPI 总线时。/HOLD 引脚低电平有效。当状态寄存器 2 的 QE 位被置位了，/ HOLD 引脚的功能不可用

原理介绍：



由于主要是讲SPI通信，如果讲我们的芯片原理恐怕要将几个小时，毕竟芯片手册是全英文，并且长达一百页，这里只为大家进行基础功能的演示，用SPI发送命令获取芯片ID

思路分析：

通过查阅手册

8.2.25 Read Manufacturer / Device ID (90h)

The Read Manufacturer/Device ID instruction is an alternative to the Release from Power-down / Device ID instruction that provides both the JEDEC assigned manufacturer ID and the specific device ID.

The Read Manufacturer/Device ID instruction is very similar to the Release from Power-down / Device ID instruction. The instruction is initiated by driving the /CS pin low and shifting the instruction code "90h" followed by a 24-bit address (A23-A0) of 000000h. After which, the Manufacturer ID for Winbond (EFh) and the Device ID are shifted out on the falling edge of CLK with most significant bit (MSB) first as shown in Figure 39. The Device ID values for the W25Q128FV are listed in Manufacturer and Device Identification table. The instruction is completed by driving /CS high.

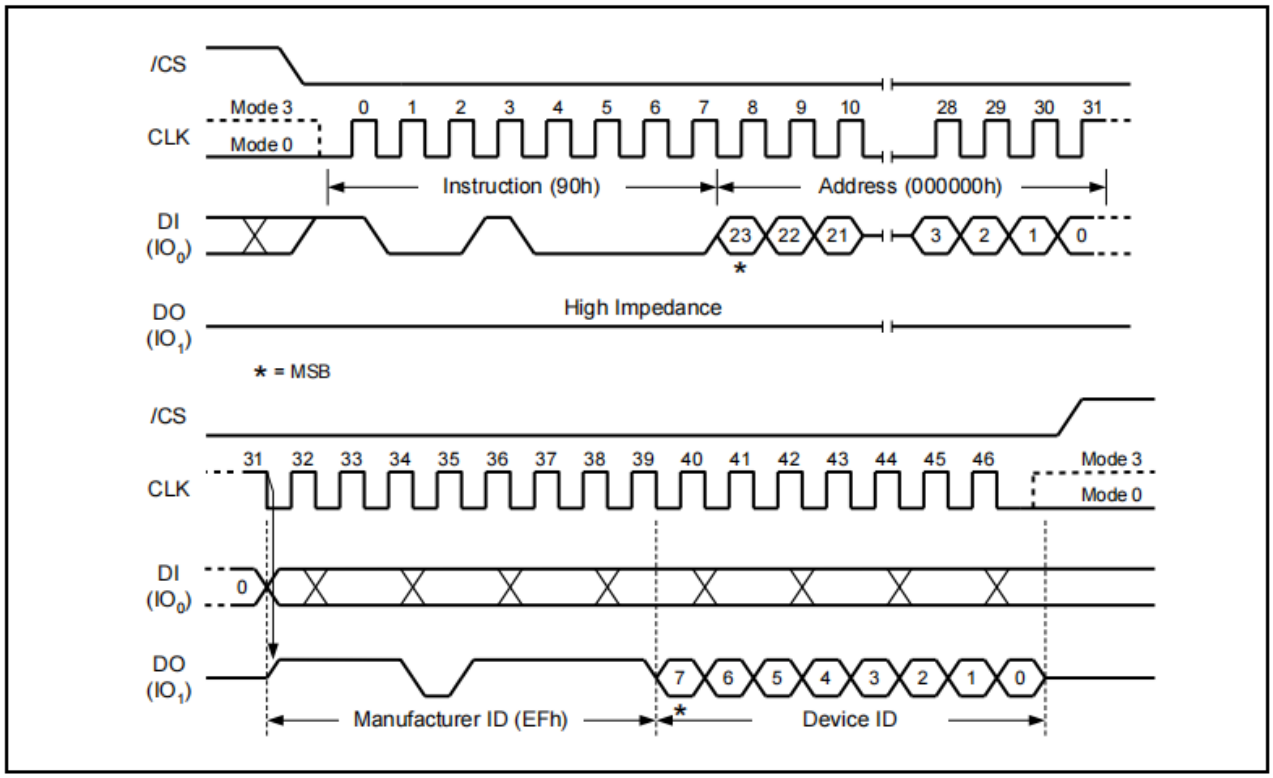


Figure 39. Read Manufacturer / Device ID Instruction (SPI Mode)

可以知道我们的90H就是获取ID的指令，并且给了非常详细的时序图，可以看出，首先是我们的/CS拉低代表通信开始，接着是我们的DI引脚发送数据发送我们的指令90H，接着要发送24位的地址00000H，拆分开就是3个00H，然后我们的DO就会回应我们的ID首先回复的是芯片厂家代号EFH，然后就是芯片ID

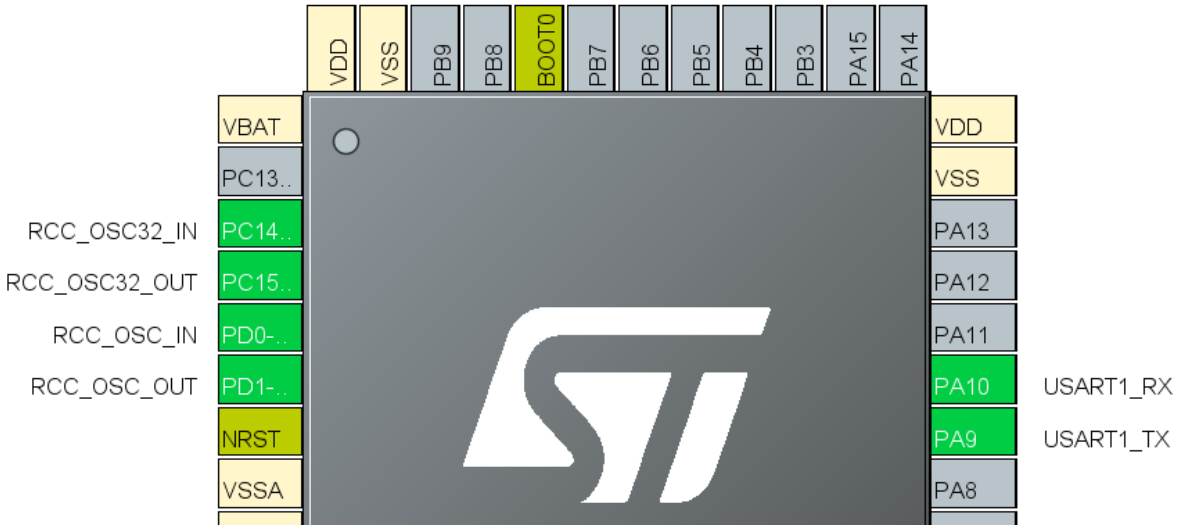
ID对应我们的的存储容量，像我们的W25Q64 ID就是16，如果是W23Q128 ID就是17，其他的小伙伴自己查手册哦!

最后就是/CS拉高，结束通信

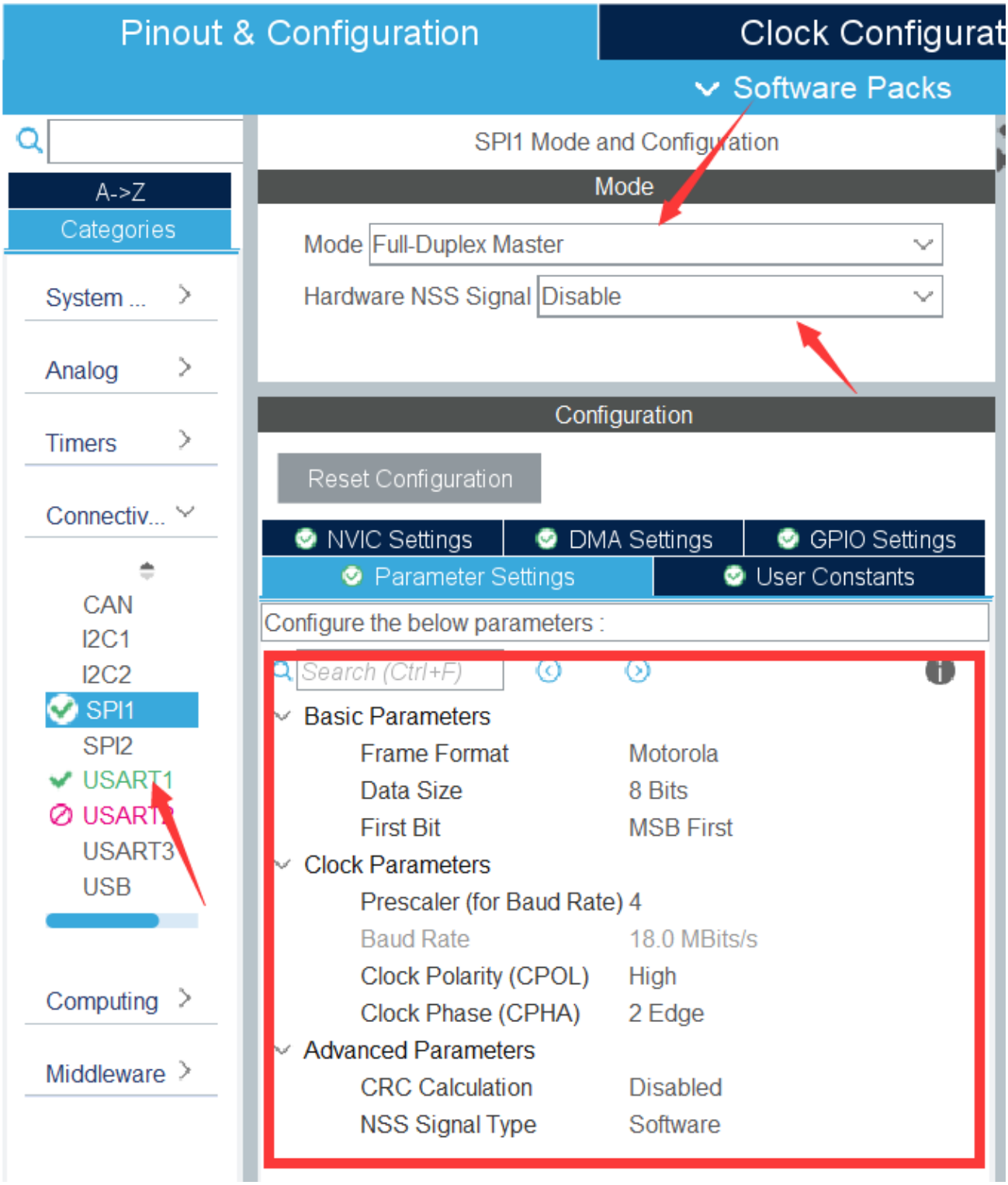
理论讲完了，接下来是代码书写

项目初始化：

由于项目需要使用串口将读取到的数据进行输出，我们先将我们的的时钟，频率，串口进行对应的设置

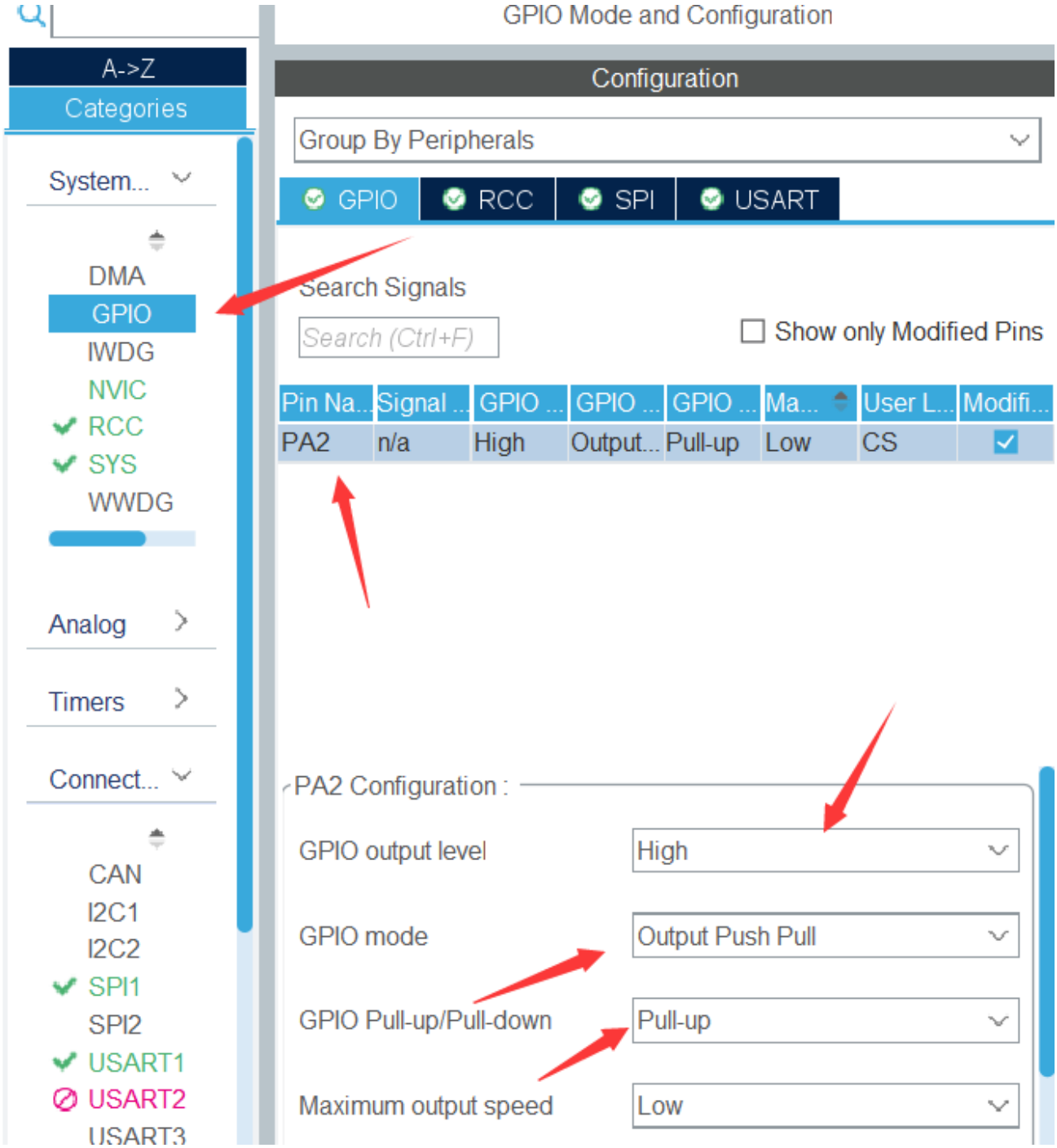


接着按照我们下图的操作打开我的SPI功能



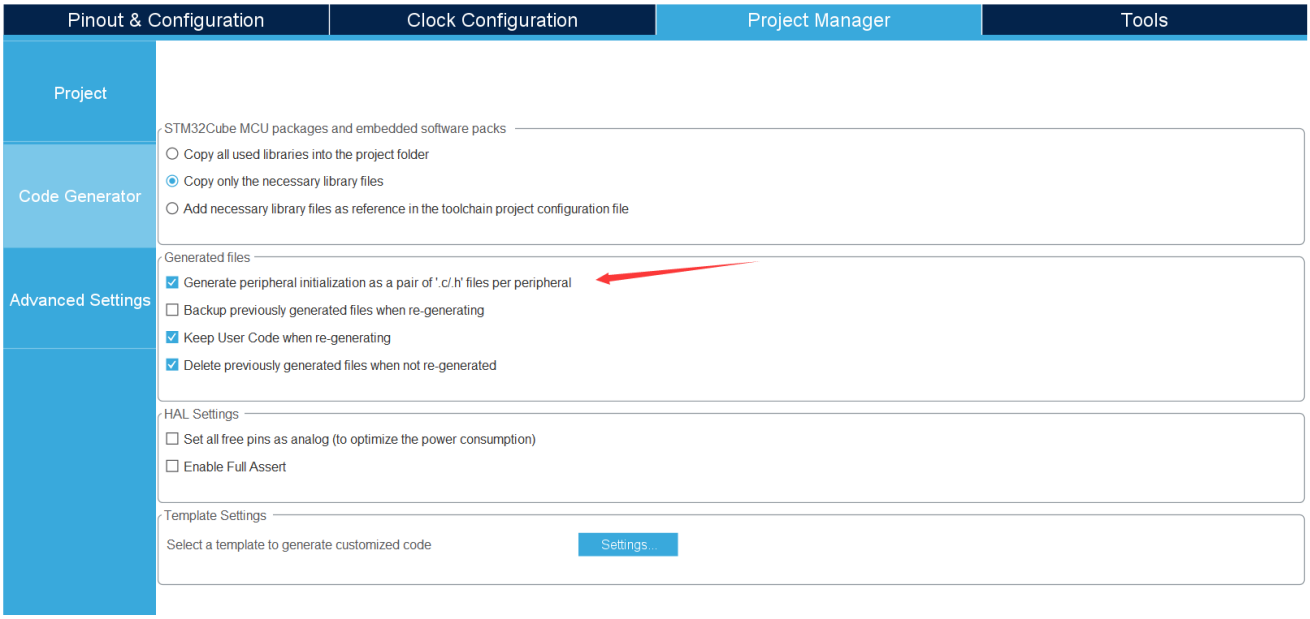
可以看到SPI需要配置的参数较多，这里对较为重要的进行讲解，也就是我们的前面理论部分涉及到的数据大小的选择，我们一般选择较为常用的8位模式，然后就是数据传输顺序的选择，这里的MSB代表高位输出，LSB代表地位输出，我们选择较为常用的MSB模式，接着就是我们的速率进行分频，我们的STM32F103C8T6最高只支持18MBits/s的速率，所以我们这里最少要选择4分频，就可以到达18MBits/s

然后我们的的CS线需要我们的自己设置，因为方便我们控制，正常项目开发中，一般都有多个SPI设备，所以系统自带的CS是不够用的，这里阿熊选择了PA2进行初始化



这里要注意的是我们要将其设置为默认高电平，电平上拉，因为根据SPI通信协议的特性，我们的CS如果为低电平，通信就开始了

接着点击我们的项目管理，勾选上生成独立C/H的文件



最后保存，等待初始化代码生成

代码书写：

由于代码较多，这里就抽比较重要的部分进行讲解，先讲解原理，后续会将完整驱动附在章节末尾

首先是最基础的数据读写，由于我们的SPI数据的读和写是同时进行的所以我们的HAL库中就有现成的函数可以实现，但是为了方便使用我们这里对其进行了一些封装

```
//SPI1总线读写一个字节
//参数是写入的字节，返回值是读出的字节
uint8_t SPI1_ReadWriteByte(uint8_t TxData)
{
    uint8_t Rxdata;//定义一个变量Rxdata
    HAL_SPI_TransmitReceive(&hspi1,&TxData,&Rxdata,1,1000);//调用固件库函数收发数据
    return Rxdata;//返回收到的数据
}
```

封装以后我们将需要发送的数据传入参数即可，将读出的数据进行函数返回，就完成了我们的SPI数据的读写

接着是封装了一个控制CS的函数

```
void W25QXX_CS(uint8_t a)//软件控制函数（0为低电平，其他值为高电平）
{
    if(a==0)HAL_GPIO_WritePin(CS_GPIO_Port,CS_Pin, GPIO_PIN_RESET);
    else  HAL_GPIO_WritePin(CS_GPIO_Port,CS_Pin, GPIO_PIN_SET);
}
```

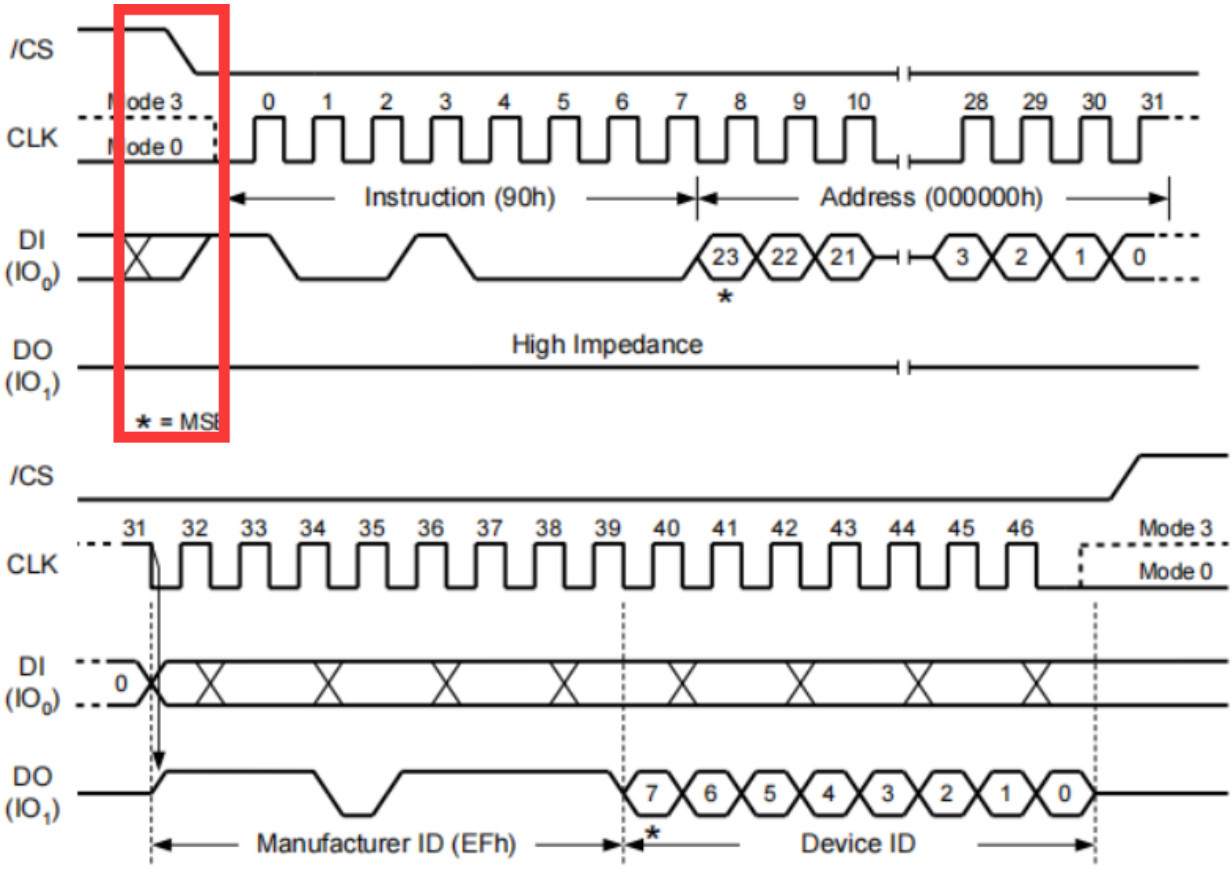
这样我们就可以使用W25QXX_CS();函数去直接控制我们的CS的电平高低，从而开启或者结束我们的通信

然后就是我们项目的主角，我们自己封装的芯片读取函数

```c
uint16_t W25QXX_ReadID(void)
{
    uint16_t Temp = 0;
    W25QXX_CS(0);//0片选开启，1片选关闭
    SPI1_ReadWriteByte(0x90);//发送读取ID命令
    SPI1_ReadWriteByte(0x00);
    SPI1_ReadWriteByte(0x00);
    SPI1_ReadWriteByte(0x00);
    Temp|=SPI1_ReadWriteByte(0xFF)<<8;
    Temp|=SPI1_ReadWriteByte(0xFF);
    W25QXX_CS(1);//0片选开启，1片选关闭
    return Temp;
}
```

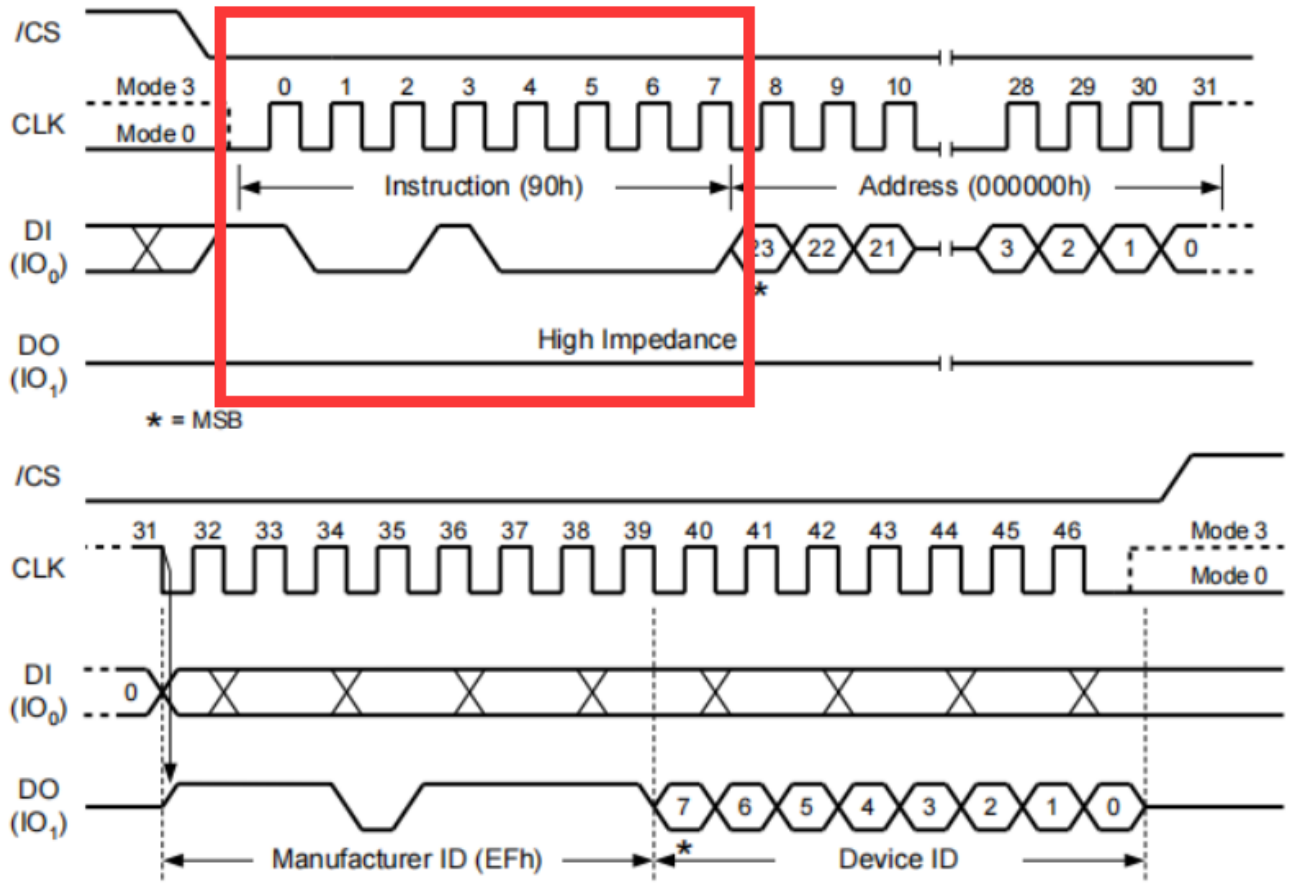
可以看到函数代码非常的少，然后这里我们可以对照前面讲过的时序图图进行一一对应

1.开启SPI通信，CS拉低



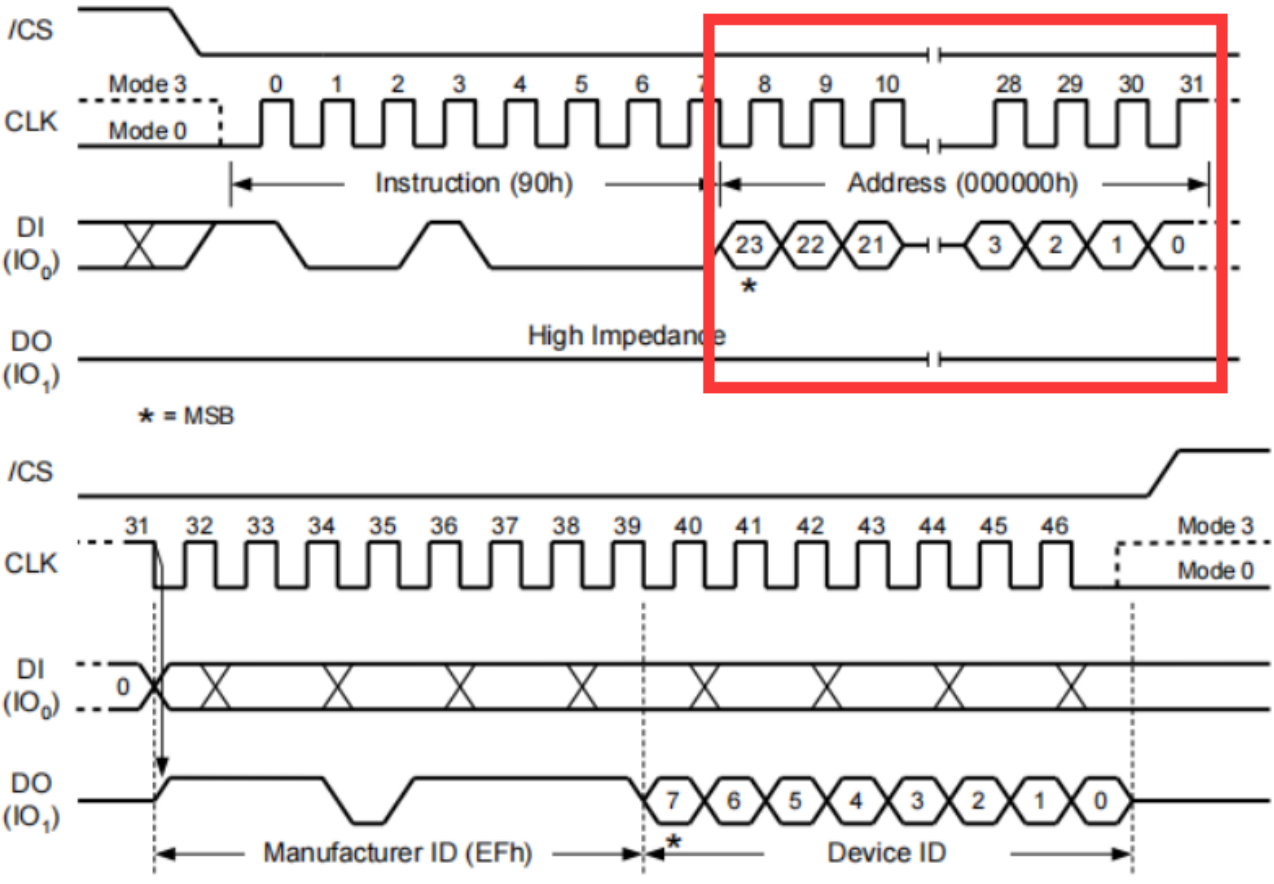
对应的就是我们函数的第二句W25QXX_CS(0);//0片选开启，1片选关闭

2.发送指令-90H



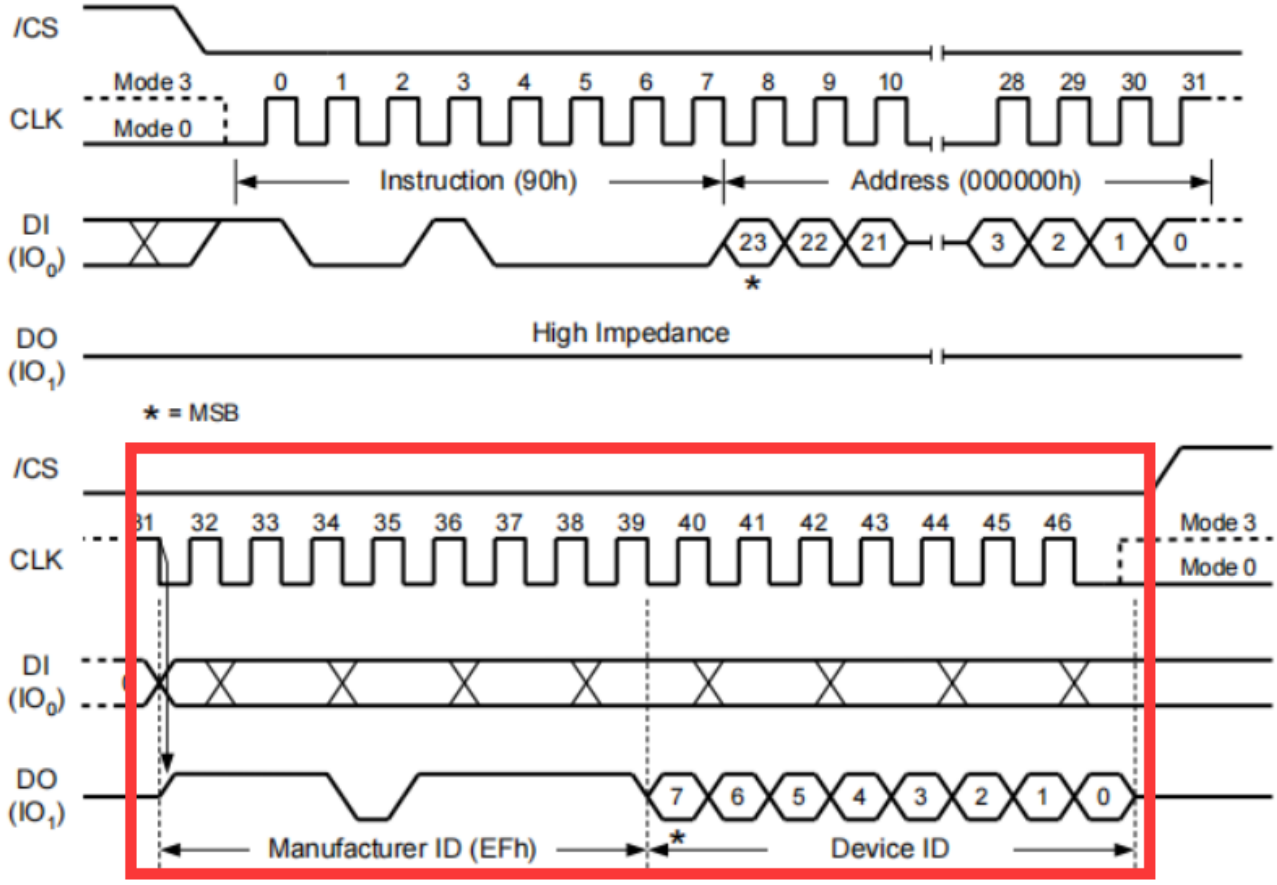
这里就对应我们代码的第三行SPI1_ReadWriteByte(0x90);//发送读取ID命令

3.发送地址-000000H



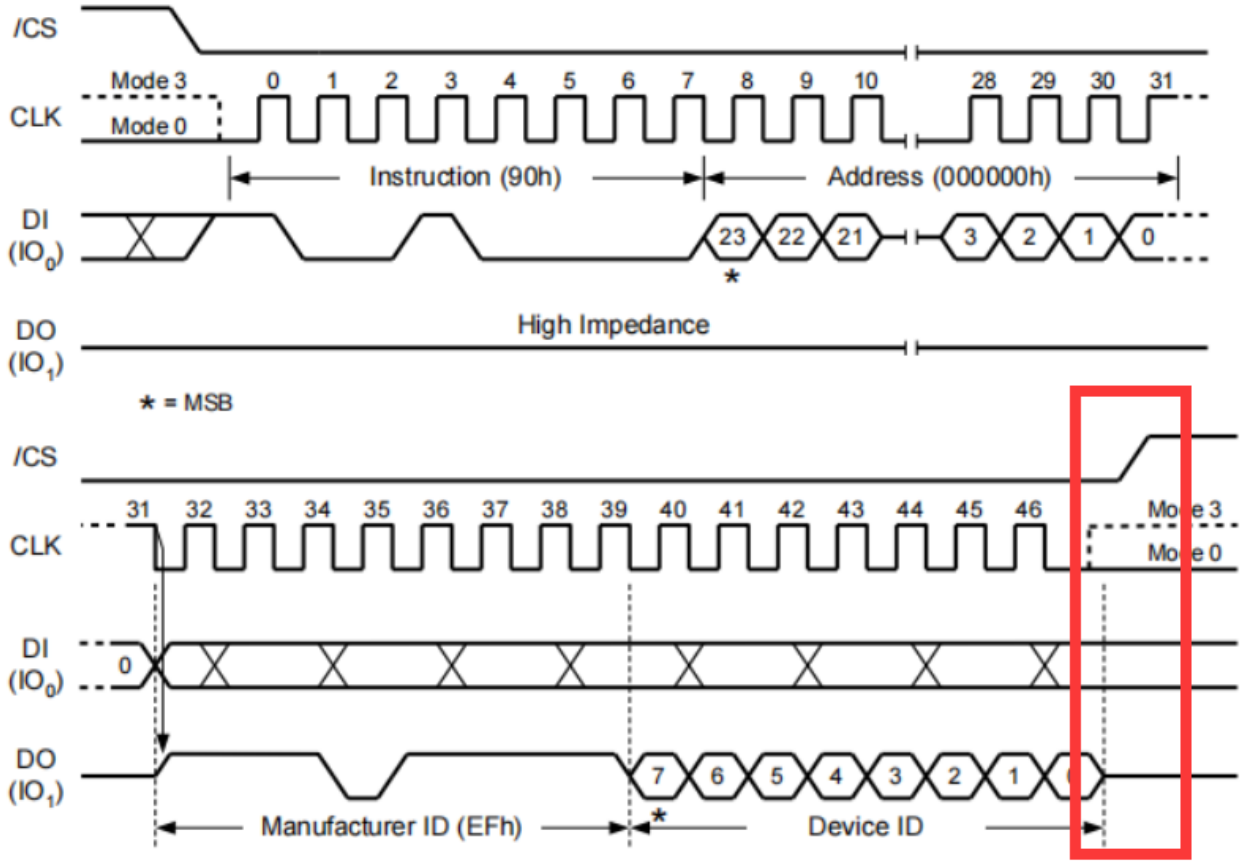
对应代码就是第4、5、6句，由于我们每次只能发送一个字节，也就是8位，我们将其拆分位三组00H，进行发送

4.接收从机应答信号



命令发出后，从机会做出应答，会返回我们两个字节，由于发送和接收是同时进行的，所以我们想要接收数据就需要发送两组数据，将其存在我们的零时变量，最后进行返回，对因函数中的1、7、8、10行

5.结束通信



最后使用我的W25QXX_CS(1);将CS线点位拉高，这样就结束了通信，对应代码中的第9行

好了，代码分析完了，我们只需要在主函数中调用，使用串口将其输出即可

```
/* Includes ------------------------------------------------------------------
*/
#include "main.h"
#include "spi.h"
#include "usart.h"
#include "gpio.h"

/* Private includes ----------------------------------------------------------
*/
/* USER CODE BEGIN Includes */
#include "w25qxx.h"
#include "retarget.h"


int main(void)
{
  /* Reset of all peripherals, Initializes the Flash interface and the
Systick. */
  HAL_Init();


  /* Configure the system clock */
  SystemClock_Config();

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_SPI1_Init();
```

```
  MX_USART1_UART_Init();
  /* USER CODE BEGIN 2 */
  RetargetInit(&huart1);//将printf()函数映射到UART1串口上
  W25QXX_Init();//W25QXX初始化
  uint16_t FLASH_BUF;//W25Q64芯片数据缓存数组
  printf("W25Q64测试程序：\n\r");//显示程序说明文字
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
      FLASH_BUF = W25QXX_ReadID();
      printf("芯片ID：%X \n\r",FLASH_BUF);//显示芯片ID
      HAL_Delay(1000);
  }
  /* USER CODE END 3 */
}
```

好了我们的代码就是这样了，接着我们来看我的实验结果

结果和我们预想的一样

成功输出了我们的EF16，对应的就是我们的芯片厂家的序号，以及我们芯片容量的序号

好了章节的内容到这里就要结束了，最后附上W25QXX的通用驱动：

w25qxx.h

```
#ifndef W25Q128_W25QXX_H_
#define W25Q128_W25QXX_H_

#include "stm32f1xx_hal.h" //HAL库文件声明

//25系列FLASH芯片厂商与容量代号（厂商代号EF）
#define W25Q80    0XEF13
#define W25Q16    0XEF14
#define W25Q32    0XEF15
#define W25Q64    0XEF16
#define W25Q128   0XEF17
#define W25Q256 0XEF18
#define EX_FLASH_ADD 0x000000 //W25Q64的地址是24位宽
extern uint16_t W25QXX_TYPE;//定义W25QXX芯片型号
extern SPI_HandleTypeDef hspi1;
////////////////////////////////////////////////////////////////////////////////
////
//指令表
#define W25X_WriteEnable            0x06
#define W25X_WriteDisable           0x04
#define W25X_ReadStatusReg1      0x05
#define W25X_ReadStatusReg2      0x35
#define W25X_ReadStatusReg3      0x15
#define W25X_WriteStatusReg1        0x01
#define W25X_WriteStatusReg2        0x31
#define W25X_WriteStatusReg3     0x11
#define W25X_ReadData             0x03
#define W25X_FastReadData         0x0B
#define W25X_FastReadDual         0x3B
#define W25X_PageProgram          0x02
#define W25X_BlockErase             0xD8
#define W25X_SectorErase          0x20
#define W25X_ChipErase            0xC7
#define W25X_PowerDown            0xB9
#define W25X_ReleasePowerDown    0xAB
#define W25X_DeviceID             0xAB
#define W25X_ManufactDeviceID    0x90
#define W25X_JedecDeviceID          0x9F
#define W25X_Enable4ByteAddr        0xB7
#define W25X_Exit4ByteAddr        0xE9
uint8_t SPI1_ReadWriteByte(uint8_t  TxData);//SPI1总线底层读写
void W25QXX_CS(uint8_t a);//W25QXX片选引脚控制
uint8_t W25QXX_Init(void);//初始化W25QXX函数
uint16_t  W25QXX_ReadID(void);//读取FLASH ID
uint8_t W25QXX_ReadSR(uint8_t regno);//读取状态寄存器
void W25QXX_4ByteAddr_Enable(void);//使能4字节地址模式
void W25QXX_Write_SR(uint8_t regno,uint8_t  sr);//写状态寄存器
void W25QXX_Write_Enable(void);//写使能
void W25QXX_Write_Disable(void);//写保护
```

```
void W25QXX_Write_NoCheck(uint8_t*  pBuffer,uint32_t WriteAddr,uint16_t
 NumByteToWrite);//无检验写SPI FLASH
void W25QXX_Read(uint8_t* pBuffer,uint32_t  ReadAddr,uint16_t
NumByteToRead);//读取flash
void W25QXX_Write(uint8_t* pBuffer,uint32_t  WriteAddr,uint16_t
NumByteToWrite);//写入flash
void W25QXX_Erase_Chip(void);//整片擦除
void W25QXX_Erase_Sector(uint32_t  Dst_Addr);//扇区擦除
void W25QXX_Wait_Busy(void);//等待空闲

#endif /* W25Q128_W25QXX_H_ */
```

w25qxx.C

```
#include "w25qxx.h"
#include "main.h"
uint16_t W25QXX_TYPE=W25Q64;

//SPI1总线读写一个字节
//参数是写入的字节，返回值是读出的字节
uint8_t SPI1_ReadWriteByte(uint8_t TxData)
{
    uint8_t Rxdata;//定义一个变量Rxdata
    HAL_SPI_TransmitReceive(&hspi1,&TxData,&Rxdata,1,1000);//调用固件库函数收发
数据
    return Rxdata;//返回收到的数据
}
void W25QXX_CS(uint8_t a)//软件控制函数（0为低电平，其他值为高电平）
{
    if(a==0)HAL_GPIO_WritePin(CS_GPIO_Port,CS_Pin, GPIO_PIN_RESET);
    else  HAL_GPIO_WritePin(CS_GPIO_Port,CS_Pin, GPIO_PIN_SET);
}
//初始化SPI FLASH的IO口
uint8_t W25QXX_Init(void)
{
    uint8_t temp;//定义一个变量temp
    W25QXX_CS(1);//0片选开启，1片选关闭
    W25QXX_TYPE = W25QXX_ReadID();//读取FLASH  ID.
    if(W25QXX_TYPE == W25Q256)//SPI FLASH为W25Q256时才用设置为4字节地址模式
    {
       temp = W25QXX_ReadSR(3);//读取状态寄存器3，判断地址模式
       if((temp&0x01)==0)//如果不是4字节地址模式,则进入4字节地址模式
       {
           W25QXX_CS(0);//0片选开启，1片选关闭
           SPI1_ReadWriteByte(W25X_Enable4ByteAddr);//发送进入4字节地址模式指令
           W25QXX_CS(1);//0片选开启，1片选关闭
       }
    }
    if(W25QXX_TYPE==W25Q256||W25QXX_TYPE==W25Q128||W25QXX_TYPE==W25Q64
    ||W25QXX_TYPE==W25Q32||W25QXX_TYPE==W25Q16||W25QXX_TYPE==W25Q80)
    return 0; else return 1;//如果读出ID是现有型号列表中的一个，则识别芯片成功！
}
//读取W25QXX的状态寄存器，W25QXX一共有3个状态寄存器
```

```
//状态寄存器1:
//BIT7  6   5   4   3   2   1   0
//SPR   RV  TB BP2 BP1 BP0 WEL BUSY
//SPR:默认0,状态寄存器保护位,配合WP使用
//TB,BP2,BP1,BP0:FLASH区域写保护设置
//WEL:写使能锁定
//BUSY:忙标记位(1,忙;0,空闲)
//默认:0x00
//状态寄存器2:
//BIT7  6   5   4   3   2   1   0
//SUS   CMP LB3 LB2 LB1 (R) QE  SRP1
//状态寄存器3:
//BIT7      6    5    4   3   2   1   0
//HOLD/RST  DRV1 DRV0 (R) (R) WPS (R) (R)
//regno:状态寄存器号，范:1~3
//返回值:状态寄存器值
uint8_t W25QXX_ReadSR(uint8_t regno)
{
    uint8_t byte=0,command=0;
    switch(regno)
    {
        case 1:
            command=W25X_ReadStatusReg1;//读状态寄存器1指令
            break;
        case 2:
            command=W25X_ReadStatusReg2;//读状态寄存器2指令
            break;
        case 3:
            command=W25X_ReadStatusReg3;//读状态寄存器3指令
            break;
        default:
            command=W25X_ReadStatusReg1;//读状态寄存器1指令
            break;
    }
    W25QXX_CS(0);//0片选开启，1片选关闭
    SPI1_ReadWriteByte(command);//发送读取状态寄存器命令
    byte=SPI1_ReadWriteByte(0Xff);//读取一个字节
    W25QXX_CS(1);//0片选开启，1片选关闭
    return byte;//返回变量byte
}
//写W25QXX状态寄存器
void W25QXX_Write_SR(uint8_t regno,uint8_t  sr)
{
    uint8_t command=0;
    switch(regno)
    {
        case 1:
            command=W25X_WriteStatusReg1;//写状态寄存器1指令
            break;
        case 2:
            command=W25X_WriteStatusReg2;//写状态寄存器2指令
            break;
        case 3:
            command=W25X_WriteStatusReg3;//写状态寄存器3指令
            break;
```

```
        default:
            command=W25X_WriteStatusReg1;
            break;
    }
    W25QXX_CS(0);//0片选开启，1片选关闭
    SPI1_ReadWriteByte(command);//发送写取状态寄存器命令
    SPI1_ReadWriteByte(sr);//写入一个字节
    W25QXX_CS(1);//0片选开启，1片选关闭
}
//W25QXX写使能
//将WEL置位
void W25QXX_Write_Enable(void)
{
    W25QXX_CS(0);//0片选开启，1片选关闭
    SPI1_ReadWriteByte(W25X_WriteEnable);//发送写使能
    W25QXX_CS(1);//0片选开启，1片选关闭
}
//W25QXX写禁止
//将WEL清零
void W25QXX_Write_Disable(void)
{
    W25QXX_CS(0);//0片选开启，1片选关闭
    SPI1_ReadWriteByte(W25X_WriteDisable);//发送写禁止指令
    W25QXX_CS(1);//0片选开启，1片选关闭
}
//读取芯片ID
//高8位是厂商代号（本程序不判断厂商代号）
//低8位是容量大小
//0XEF13型号为W25Q80
//0XEF14型号为W25Q16
//0XEF15型号为W25Q32
//0XEF16型号为W25Q64
//0XEF17型号为W25Q128
//0XEF18型号为W25Q256
uint16_t W25QXX_ReadID(void)
{
    uint16_t Temp = 0;
    W25QXX_CS(0);//0片选开启，1片选关闭
    SPI1_ReadWriteByte(0x90);//发送读取ID命令
    SPI1_ReadWriteByte(0x00);
    SPI1_ReadWriteByte(0x00);
    SPI1_ReadWriteByte(0x00);
    Temp|=SPI1_ReadWriteByte(0xFF)<<8;
    Temp|=SPI1_ReadWriteByte(0xFF);
    W25QXX_CS(1);//0片选开启，1片选关闭
    return Temp;
}
//读取SPI FLASH
//在指定地址开始读取指定长度的数据
//pBuffer:数据存储区
//ReadAddr:开始读取的地址(24bit)
//NumByteToRead:要读取的字节数(最大65535)
void W25QXX_Read(uint8_t* pBuffer,uint32_t  ReadAddr,uint16_t NumByteToRead)
{
    uint16_t i;
```

```
    W25QXX_CS(0);//0片选开启，1片选关闭
    SPI1_ReadWriteByte(W25X_ReadData);//发送读取命令
    if(W25QXX_TYPE==W25Q256)//如果是W25Q256的话地址为4字节的，要发送最高8位
    {
        SPI1_ReadWriteByte((uint8_t)((ReadAddr)>>24));
    }
    SPI1_ReadWriteByte((uint8_t)((ReadAddr)>>16));//发送24bit地址
    SPI1_ReadWriteByte((uint8_t)((ReadAddr)>>8));
    SPI1_ReadWriteByte((uint8_t)ReadAddr);
    for(i=0;i<NumByteToRead;i++)
    {
        pBuffer[i]=SPI1_ReadWriteByte(0XFF);//循环读数
    }
    W25QXX_CS(1);//0片选开启，1片选关闭
}
//SPI在一页(0~65535)内写入少于256个字节的数据
//在指定地址开始写入最大256字节的数据
//pBuffer:数据存储区
//WriteAddr:开始写入的地址(24bit)
//NumByteToWrite:要写入的字节数(最大256),该数不应该超过该页的剩余字节数!!!
void W25QXX_Write_Page(uint8_t*  pBuffer,uint32_t WriteAddr,uint16_t
 NumByteToWrite)
{
    uint16_t i;
    W25QXX_Write_Enable();//SET WEL
    W25QXX_CS(0);//0片选开启，1片选关闭
    SPI1_ReadWriteByte(W25X_PageProgram);//发送写页命令
    if(W25QXX_TYPE==W25Q256)//如果是W25Q256的话地址为4字节的，要发送最高8位
    {
        SPI1_ReadWriteByte((uint8_t)((WriteAddr)>>24));
    }
     SPI1_ReadWriteByte((uint8_t)((WriteAddr)>>16));//发送24bit地址
     SPI1_ReadWriteByte((uint8_t)((WriteAddr)>>8));
    SPI1_ReadWriteByte((uint8_t)WriteAddr);
     for(i=0;i<NumByteToWrite;i++)SPI1_ReadWriteByte(pBuffer[i]);//循环写数
    W25QXX_CS(1);//0片选开启，1片选关闭
    W25QXX_Wait_Busy();//等待写入结束
}
//无检验写SPI FLASH
//必须确保所写的地址范围内的数据全部为0XFF,否则在非0XFF处写入的数据将失败!
//具有自动换页功能
//在指定地址开始写入指定长度的数据,但是要确保地址不越界!
//pBuffer:数据存储区
//WriteAddr:开始写入的地址(24bit)
//NumByteToWrite:要写入的字节数(最大65535)
//CHECK OK
void W25QXX_Write_NoCheck(uint8_t*  pBuffer,uint32_t WriteAddr,uint16_t
 NumByteToWrite)
{
    uint16_t pageremain;
    pageremain=256-WriteAddr%256; //单页剩余的字节数
    if(NumByteToWrite<=pageremain)pageremain=NumByteToWrite;//不大于256个字节
    while(1)
    {
       W25QXX_Write_Page(pBuffer,WriteAddr,pageremain);
```

```
        if(NumByteToWrite==pageremain)break;//写入结束了
         else //NumByteToWrite>pageremain
        {
            pBuffer+=pageremain;
            WriteAddr+=pageremain;
            NumByteToWrite-=pageremain;             //减去已经写入了的字节数
            if(NumByteToWrite>256)pageremain=256; //一次可以写入256个字节
            else pageremain=NumByteToWrite;      //不够256个字节了
        }
    };
}
//写SPI FLASH
//在指定地址开始写入指定长度的数据
//该函数带擦除操作!
//pBuffer:数据存储区
//WriteAddr:开始写入的地址(24bit)
//NumByteToWrite:要写入的字节数(最大65535)
uint8_t W25QXX_BUFFER[4096];
void W25QXX_Write(uint8_t* pBuffer,uint32_t  WriteAddr,uint16_t
NumByteToWrite)
{
    uint32_t secpos;
    uint16_t secoff;
    uint16_t secremain;
    uint16_t i;
    uint8_t* W25QXX_BUF;
    W25QXX_BUF=W25QXX_BUFFER;
    secpos=WriteAddr/4096;//扇区地址
    secoff=WriteAddr%4096;//在扇区内的偏移
    secremain=4096-secoff;//扇区剩余空间大小
    //printf("ad:%X,nb:%X\r\n",WriteAddr,NumByteToWrite);//测试用
    if(NumByteToWrite<=secremain)secremain=NumByteToWrite;//不大于4096个字节
    while(1)
    {
        W25QXX_Read(W25QXX_BUF,secpos*4096,4096);//读出整个扇区的内容
        for(i=0;i<secremain;i++)//校验数据
        {
            if(W25QXX_BUF[secoff+i]!=0XFF)break;//需要擦除
        }
        if(i<secremain)//需要擦除
        {
            W25QXX_Erase_Sector(secpos);//擦除这个扇区
            for(i=0;i<secremain;i++)//复制
            {
                W25QXX_BUF[i+secoff]=pBuffer[i];
            }
            W25QXX_Write_NoCheck(W25QXX_BUF,secpos*4096,4096);//写入整个扇区
        }else  W25QXX_Write_NoCheck(pBuffer,WriteAddr,secremain);//写已经擦除了
的,直接写入扇区剩余区间.
        if(NumByteToWrite==secremain)break;//写入结束了
        else//写入未结束
        {
            secpos++;//扇区地址增1
            secoff=0;//偏移位置为0
            pBuffer+=secremain;  //指针偏移
```

```
            WriteAddr+=secremain;//写地址偏移
            NumByteToWrite-=secremain;//字节数递减
            if(NumByteToWrite>4096)secremain=4096;//下一个扇区还是写不完
            else  secremain=NumByteToWrite;//下一个扇区可以写完了
        }
    };
}
//擦除整个芯片
//等待时间超长...
void W25QXX_Erase_Chip(void)
{
    W25QXX_Write_Enable();//SET WEL
    W25QXX_Wait_Busy();//等待忙状态
    W25QXX_CS(0);//0片选开启，1片选关闭
    SPI1_ReadWriteByte(W25X_ChipErase);//发送片擦除命令
    W25QXX_CS(1);//0片选开启，1片选关闭
    W25QXX_Wait_Busy();//等待芯片擦除结束
}
//擦除一个扇区
//Dst_Addr:扇区地址 根据实际容量设置
//擦除一个扇区的最少时间:150ms
void W25QXX_Erase_Sector(uint32_t Dst_Addr)
{
    Dst_Addr*=4096;
    W25QXX_Write_Enable();//SET WEL
    W25QXX_Wait_Busy();
    W25QXX_CS(0);//0片选开启，1片选关闭
    SPI1_ReadWriteByte(W25X_SectorErase);//发送扇区擦除指令
    if(W25QXX_TYPE==W25Q256)//如果是W25Q256的话地址为4字节的，要发送最高8位
    {
        SPI1_ReadWriteByte((uint8_t)((Dst_Addr)>>24));
    }
    SPI1_ReadWriteByte((uint8_t)((Dst_Addr)>>16));//发送24bit地址
    SPI1_ReadWriteByte((uint8_t)((Dst_Addr)>>8));
    SPI1_ReadWriteByte((uint8_t)Dst_Addr);
    W25QXX_CS(1);//0片选开启，1片选关闭
    W25QXX_Wait_Busy();//等待擦除完成
}
//等待空闲
void W25QXX_Wait_Busy(void)
{
    while((W25QXX_ReadSR(1)&0x01)==0x01);//等待BUSY位清空
}
```