7. STM32 SPI通信

北京科技大学
计算机与通信工程学院

目录

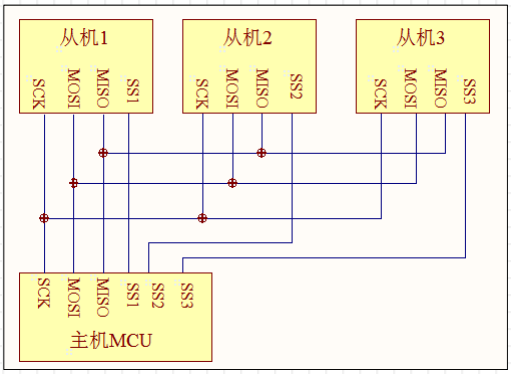
CONTENTS.

STM32-7

SPI协议简介

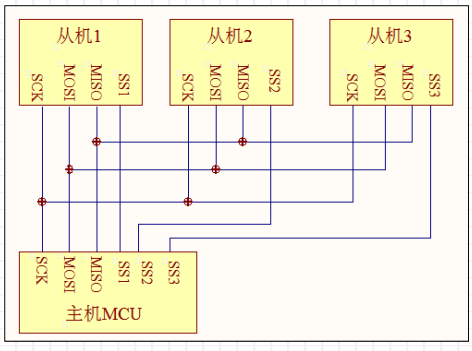
SPI协议是由摩托罗拉公司提出的通讯协议(Serial Peripheral Interface)，即串行外围设备接口，是一种高速全双工的通信总线。它被广泛地使用在ADC、LCD等设备与MCU间，要求通讯速率较高的场合。

SPI物理层的特点



SPI协议简介

SPI物理层的特点

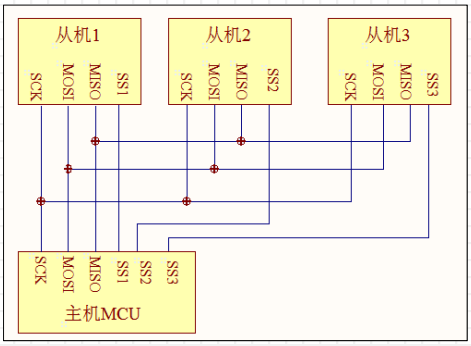


- SS:从设备选择信号线，常称为片选信号线，也称为NSS、CS。

每个从设备都有独立的这一条SS信号线，本信号线独占主机的一个引脚，即有多少个从设备，就有多少条片选信号线。I2C协议中通过设备地址来寻址、选中总线上的某个设备并与其进行通讯；而SPI协议中没有设备地址，它使用SS信号线来寻址，当主机要选择从设备时，把该从设备的SS信号线设置为低电平，该从设备即被选中，即片选有效，接着主机开始与被选中的从设备进行SPI通讯。所以SPI通讯以SS线置低电平为开始信号，以SS线被拉高作为结束信号。

SPI协议简介

SPI物理层的特点

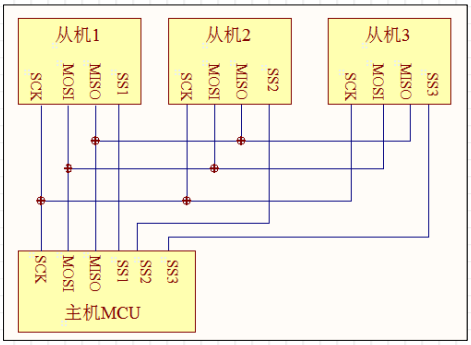


- SCK (Serial Clock)：时钟信号线，用于通讯数据同步。

它由通讯主机产生，决定了通讯的速率，不同的设备支持的最高时钟频率不一样，如STM32的SPI时钟频率最大为$f_{pclk}/2$，两个设备之间通讯时，通讯速率受限于低速设备。

SPI协议简介

SPI物理层的特点



- MOSI (Master Output， Slave Input)：主设备输出/从设备输入引脚。

主机的数据从这条信号线输出，从机由这条信号线读入主机发送的数据，即这条线上数据的方向为主机到从机。

- MISO(Master Input,，Slave Output)：主设备输入/从设备输出引脚。

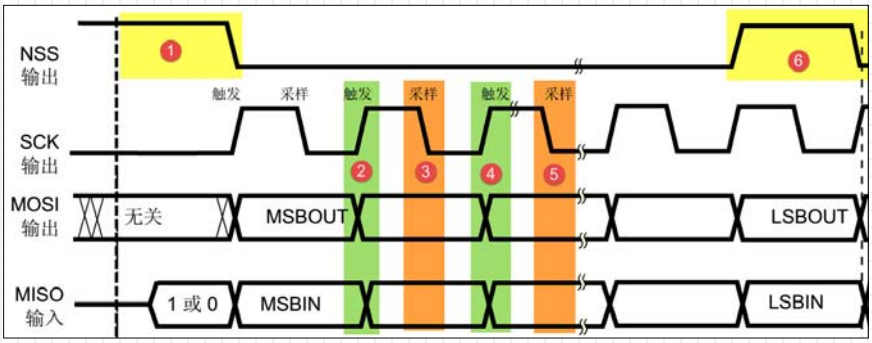
主机从这条信号线读入数据，从机的数据由这条信号线输出到主机，即在这条线上数据的方向为从机到主机。

SPI协议简介

SPI的协议层

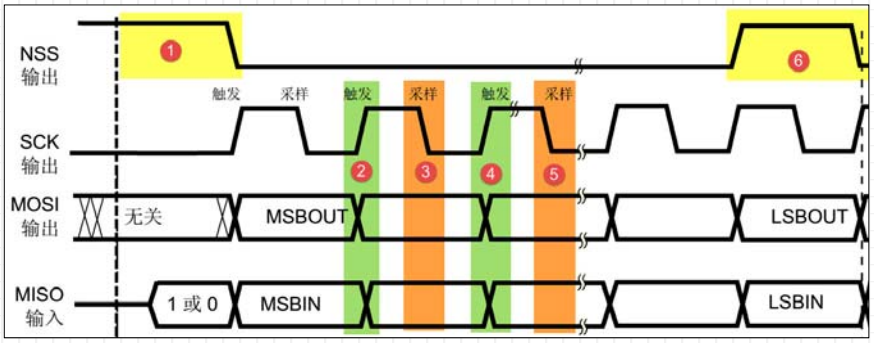
SPI协议定义了通讯的起始和停止信号、数据有效性、时钟同步等环节。

1.SPI基本通讯过程



SPI协议简介

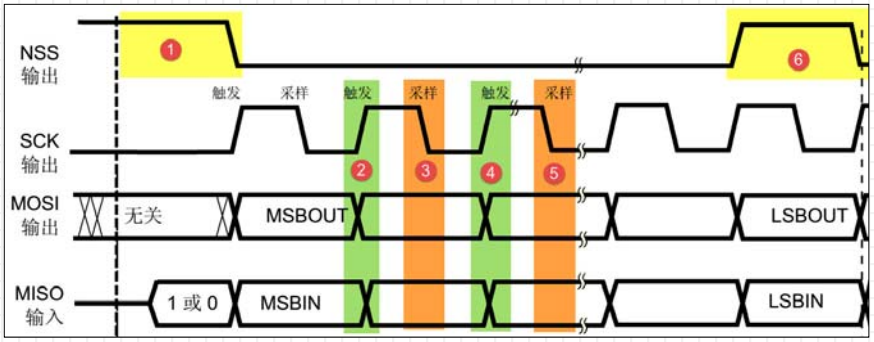
2. 通讯的起始和停止信号



- 标号①处，NSS信号线由高变低，是SPI通讯的起始信号。NSS是每个从机各自独占的信号线，当从机检在自己的NSS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。
- 在图中的标号⑥处，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。

SPI协议简介

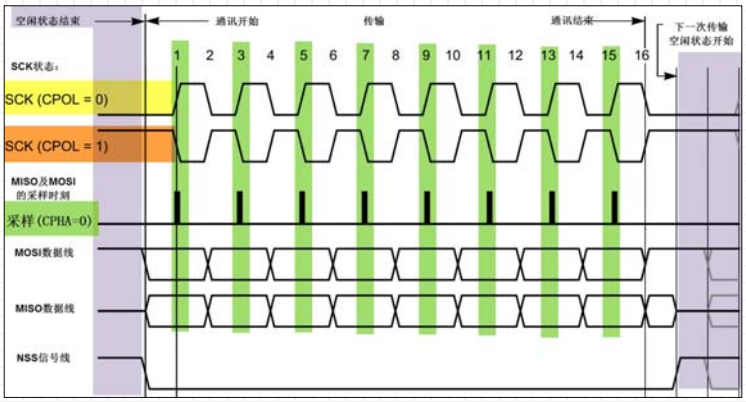
3.数据有效性



- SPI使用MOSI及MISO信号线来传输数据，使用SCK信号线进行数据同步。MOSI及MISO数据线在SCK的每个时钟周期传输一位数据，且数据输入输出是同时进行的。

SPI协议简介

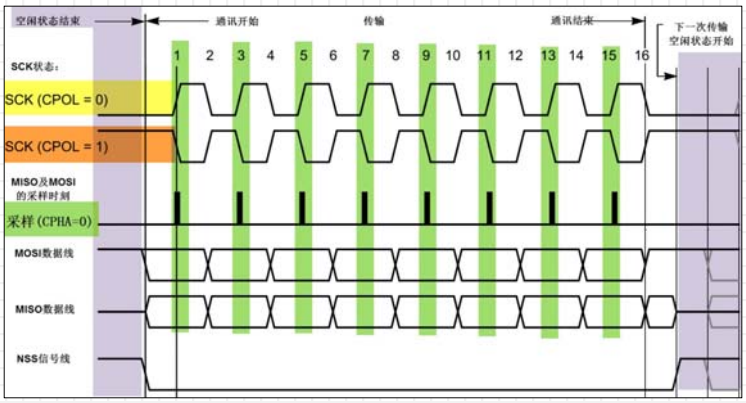
4.CPOL/CPHA及通讯模式



- 时钟极性CPOL是指SPI通讯设备处于空闲状态时，SCK信号线的电平信号(即SPI通讯开始前、 NSS线为高电平时SCK的状态)。CPOL=0时， SCK在空闲状态时为低电平，CPOL=1时，则相反。
- 时钟相位CPHA是指数据的采样的时刻，当CPHA=0时，MOSI或MISO数据线上的信号将会在SCK时钟线的“奇数边沿”被采样。当CPHA=1时，数据线在SCK的“偶数边沿”采样。

SPI协议简介

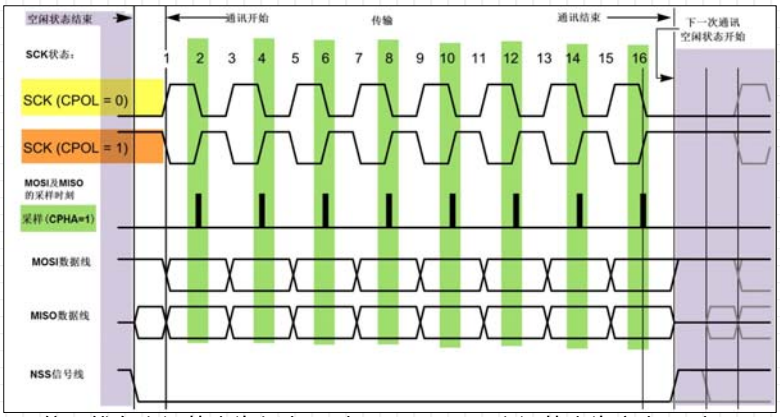
4.CPOL/CPHA及通讯模式



- SCK信号线在空闲状态为低电平时，CPOL=0；空闲状态为高电平时，CPOL=1。
- CPHA=0，MOSI和MISO数据线的有效信号在SCK的奇数边沿保持不变，数据信号将在SCK奇数边沿时被采样，在非采样时刻，MOSI和MISO的有效信号才发生切换。

SPI协议简介

4.CPOL/CPHA及通讯模式



- SCK信号线在空闲状态为低电平时，CPOL=0；空闲状态为高电平时，CPOL=1。
- CPHA=1，MOSI和MISO数据线的有效信号在SCK的偶数边沿保持不变，数据信号将在SCK偶数边沿时被采样，在非采样时刻，MOSI和MISO的有效信号才发生切换。

SPI协议简介

4.CPOL/CPHA及通讯模式

由CPOL及CPHA的不同状态，SPI分成了四种模式，主机与从机需要工作在相同的模式下才可以正常通讯，实际中采用较多的是“模式0”与“模式3”。

SPI模式	CPOL	CPHA	空闲时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

目录

CONTENTS.

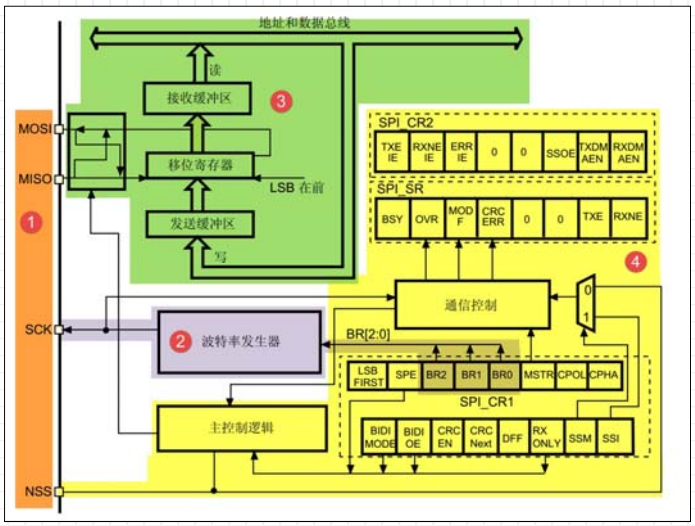
STM32-7

14

STM32的SPI外设简介

STM32的SPI外设可用作通讯的主机及从机，支持最高的SCK时钟频率为f_{pclk}/2 (STM32F10x型号的芯片默认f_{pclk1}为36MHz，f_{pclk2}为72MHz)，完全支持SPI协议的4种模式，数据帧长度可设置为8位或16位，可设置数据MSB先行或LSB先行。它还支持双线全双工(前面小节说明的都是这种模式)、双线单向以及单线模式。

STM32的SPI架构剖析



- 通讯引脚
- 时钟控制逻辑
- 数据控制逻辑
- 整体控制逻辑

1.通讯引脚

STM32芯片有多个SPI外设，它们的SPI通讯信号引出到不同的GPIO引脚上，使用时必须配置到这些指定的引脚，以《STM32F10x规格书》为准。

引脚	SPI编号		
	SPI1	SPI2	SPI3
NSS	PA4	PB12	PA15下载口的TDI
CLK	PA5	PB13	PB3下载口的TDO
MISO	PA6	PB14	PB4下载口的NTRST
MOSI	PA7	PB15	PB5

其中SPI1是APB2上的设备，最高通信速率达36Mbtis/s，SPI2、SPI3是APB1上的设备，最高通信速率为18Mbits/s。除了通讯速率，在其它功能上没有差异。

2.时钟控制逻辑

SCK线的时钟信号，由波特率发生器根据“控制寄存器CR1”中的BR[0:2]位控制，该位是对f_{pclk}时钟的分频因子，对f_{pclk}的分频结果就是SCK引脚的输出时钟频率

BR[0:2]	分频结果(SCK频率)		BR[0:2]	分频结果(SCK频率)
000	$f_{pclk}/2$		100	$f_{pclk}/32$
001	$f_{pclk}/4$		101	$f_{pclk}/64$
010	$f_{pclk}/8$		110	$f_{pclk}/128$
011	$f_{pclk}/16$		111	$f_{pclk}/256$

其中的f_{pclk}频率是指SPI所在的APB总线频率，APB1为f_{pclk1}，APB2为f_{pckl2}。

3.数据控制逻辑

SPI的MOSI及MISO都连接到数据移位寄存器上，数据移位寄存器的数据来源来源于接收缓冲区及发送缓冲区。

- 通过写SPI的“数据寄存器DR”把数据填充到发送缓冲区中。
- 通过读“数据寄存器DR”，可以获取接收缓冲区中的内容。

- 其中数据帧长度可以通过“控制寄存器CR1”的“DFF位”配置成8位及16位模式；配置“LSBFIRST位”可选择MSB先行还是LSB先行。

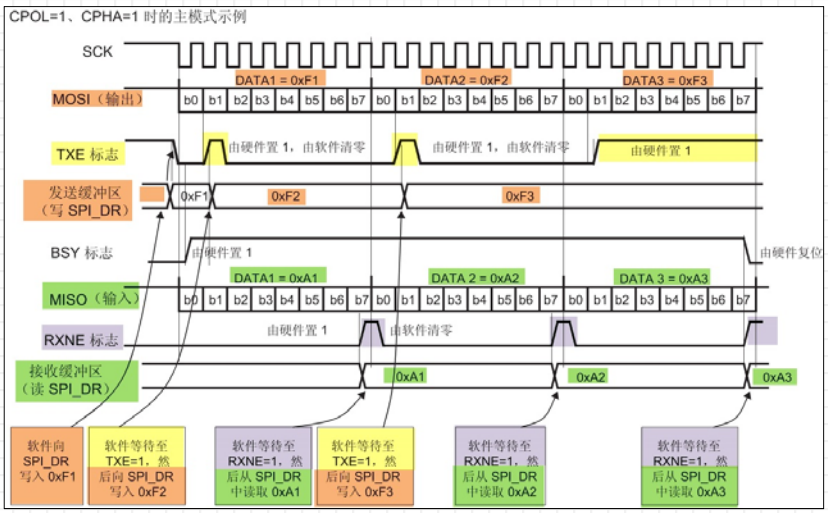
4.整体控制逻辑

- 整体控制逻辑负责协调整个SPI外设，控制逻辑的工作模式根据“控制寄存器(CR1/CR2)”的参数而改变，基本的控制参数包括前面提到的SPI模式、波特率、LSB先行、主从模式、单双向模式等等。

- 在外设工作时，控制逻辑会根据外设的工作状态修改“状态寄存器(SR)”，只要读取状态寄存器相关的寄存器位，就可以了解SPI的工作状态了。除此之外，控制逻辑还根据要求，负责控制产生SPI中断信号、DMA请求及控制NSS信号线。

- 实际应用中，一般不使用STM32 SPI外设的标准NSS信号线，而是更简单地使用普通的GPIO，软件控制它的电平输出，从而产生通讯起始和停止信号。

通讯过程



通讯过程

- 控制NSS信号线，产生起始信号(图中没有画出)；
- 把要发送的数据写入到“数据寄存器DR”中，该数据会被存储到发送缓冲区；
- 通讯开始，SCK时钟开始运行。MOSI把发送缓冲区中的数据一位一位地传输出去；MISO则把数据一位一位地存储进接收缓冲区中；
- 当发送完一帧数据的时候，“状态寄存器SR”中的“TXE标志位”会被置1，表示传输完一帧，发送缓冲区已空；类似地，当接收完一帧数据的时候，“RXNE标志位”会被置1，表示传输完一帧，接收缓冲区非空；
- 等待到“TXE标志位”为1时，若还要继续发送数据，则再次往“数据寄存器DR”写入数据即可；等待到“RXNE标志位”为1时，通过读取“数据寄存器DR”可以获取接收缓冲区中的内容。

假如使能了TXE或RXNE中断，TXE或RXNE置1时会产生SPI中断信号，进入同一个中断服务函数，到SPI中断服务程序后，可通过检查寄存器位来了解是哪一个事件，再分别进行处理。也可以使用DMA方式来收发“数据寄存器DR”中的数据。

目录

CONTENTS.

STM32-7

SPI初始化结构体

跟其它外设一样，STM32标准库提供了SPI初始化结构体及初始化函数来配置SPI外设。初始化结构体及函数定义在库文件“stm32f10x_spi.h”及“stm32f10x_spi.c”中，编程时我们可以结合这两个文件内的注释使用或参考库帮助文档。

```
typedef struct
{
    uint16_t SPI_Direction;          /*设置 SPI 的单双向模式 */
    uint16_t SPI_Mode;               /*设置 SPI 的主/从机端模式 */
    uint16_t SPI_DataSize;           /*设置 SPI 的数据帧长度，可选 8/16 位 */
    uint16_t SPI_CPOL;               /*设置时钟极性 CPOL，可选高/低电平*/
    uint16_t SPI_CPHA;               /*设置时钟相位，可选奇/偶数边沿采样 */
    uint16_t SPI_NSS;                /*设置 NSS 引脚由 SPI 硬件控制还是软件控制*/
    uint16_t SPI_BaudRatePrescaler;  /*设置时钟分频因子，fpclk/分频数=fSCK */
    uint16_t SPI_FirstBit;           /*设置 MSB/LSB 先行 */
    uint16_t SPI_CRCPolynomial;      /*设置 CRC 校验的表达式 */
} SPI_InitTypeDef;
```

SPI初始化结构体

```
typedef struct
{
    uint16_t SPI_Direction;          /*设置 SPI 的单双向模式 */
    uint16_t SPI_Mode;              /*设置 SPI 的主/从机端模式 */
    uint16_t SPI_DataSize;           /*设置 SPI 的数据帧长度，可选 8/16 位 */
    uint16_t SPI_CPOL;              /*设置时钟极性 CPOL，可选高/低电平*/
    uint16_t SPI_CPHA;              /*设置时钟相位，可选奇/偶数边沿采样 */
    uint16_t SPI_NSS;              /*设置 NSS 引脚由 SPI 硬件控制还是软件控制*/
    uint16_t SPI_BaudRatePrescaler;  /*设置时钟分频因子，fpclk/分频数=fSCK */
    uint16_t SPI_FirstBit;           /*设置 MSB/LSB 先行 */
    uint16_t SPI_CRCPolynomial;      /*设置 CRC 校验的表达式 */
} SPI_InitTypeDef;
```

- SPI_Direction

本成员设置SPI的通讯方向，可设置为双线全双工(SPI_Direction_2Lines_FullDuplex)，双线只接收(SPI_Direction_2Lines_RxOnly)，单线只接收(SPI_Direction_1Line_Rx)、单线只发送模式(SPI_Direction_1Line_Tx)。

SPI初始化结构体

```
typedef struct
{
    uint16_t SPI_Direction;          /*设置 SPI 的单双向模式 */
    uint16_t SPI_Mode;              /*设置 SPI 的主/从机端模式 */
    uint16_t SPI_DataSize;           /*设置 SPI 的数据帧长度，可选 8/16 位 */
    uint16_t SPI_CPOL;              /*设置时钟极性 CPOL，可选高/低电平*/
    uint16_t SPI_CPHA;              /*设置时钟相位，可选奇/偶数边沿采样 */
    uint16_t SPI_NSS;              /*设置 NSS 引脚由 SPI 硬件控制还是软件控制*/
    uint16_t SPI_BaudRatePrescaler;  /*设置时钟分频因子，fpclk/分频数=fSCK */
    uint16_t SPI_FirstBit;           /*设置 MSB/LSB 先行 */
    uint16_t SPI_CRCPolynomial;      /*设置 CRC 校验的表达式 */
} SPI_InitTypeDef;
```

- SPI_Mode

本成员设置SPI工作在主机模式(SPI_Mode_Master)或从机模式(SPI_Mode_Slave)，这两个模式的最大区别为SPI的SCK信号线的时序，SCK的时序是由通讯中的主机产生的。若被配置为从机模式，STM32的SPI外设将接受外来的SCK信号。

SPI初始化结构体

```
typedef struct
{
    uint16_t SPI_Direction;          /*设置 SPI 的单双向模式 */
    uint16_t SPI_Mode;               /*设置 SPI 的主/从机端模式 */
    uint16_t SPI_DataSize;           /*设置 SPI 的数据帧长度，可选 8/16 位 */
    uint16_t SPI_CPOL;               /*设置时钟极性 CPOL，可选高/低电平*/
    uint16_t SPI_CPHA;               /*设置时钟相位，可选奇/偶数边沿采样 */
    uint16_t SPI_NSS;                /*设置 NSS 引脚由 SPI 硬件控制还是软件控制*/
    uint16_t SPI_BaudRatePrescaler;  /*设置时钟分频因子，fpclk/分频数=fSCK */
    uint16_t SPI_FirstBit;           /*设置 MSB/LSB 先行 */
    uint16_t SPI_CRCPolynomial;      /*设置 CRC 校验的表达式 */
} SPI_InitTypeDef;
```

- SPI_DataSize

本成员可以选择SPI通讯的数据帧大小是为8位(SPI_DataSize_8b)还是16位(SPI_DataSize_16b)。

SPI初始化结构体

```
typedef struct
{
    uint16_t SPI_Direction;          /*设置 SPI 的单双向模式 */
    uint16_t SPI_Mode;               /*设置 SPI 的主/从机端模式 */
    uint16_t SPI_DataSize;           /*设置 SPI 的数据帧长度，可选 8/16 位 */
    uint16_t SPI_CPOL;               /*设置时钟极性 CPOL，可选高/低电平*/
    uint16_t SPI_CPHA;               /*设置时钟相位，可选奇/偶数边沿采样 */
    uint16_t SPI_NSS;                /*设置 NSS 引脚由 SPI 硬件控制还是软件控制*/
    uint16_t SPI_BaudRatePrescaler;  /*设置时钟分频因子，fpclk/分频数=fSCK */
    uint16_t SPI_FirstBit;           /*设置 MSB/LSB 先行 */
    uint16_t SPI_CRCPolynomial;      /*设置 CRC 校验的表达式 */
} SPI_InitTypeDef;
```

- SPI_CPOL和SPI_CPHA

这两个成员配置SPI的时钟极性CPOL和时钟相位CPHA，这两个配置影响到SPI的通讯模式，

时钟极性CPOL成员，可设置为高电平(SPI_CPOL_High)或低电平(SPI_CPOL_Low)。

时钟相位CPHA 则可以设置为SPI_CPHA_1Edge(在SCK的奇数边沿采集数据) 或SPI_CPHA_2Edge (在SCK的偶数边沿采集数据) 。

SPI初始化结构体

```
typedef struct
{
    uint16_t SPI_Direction;          /*设置 SPI 的单双向模式 */
    uint16_t SPI_Mode;              /*设置 SPI 的主/从机端模式 */
    uint16_t SPI_DataSize;           /*设置 SPI 的数据帧长度，可选 8/16 位 */
    uint16_t SPI_CPOL;              /*设置时钟极性 CPOL，可选高/低电平*/
    uint16_t SPI_CPHA;              /*设置时钟相位，可选奇/偶数边沿采样 */
    uint16_t SPI_NSS;              /*设置 NSS 引脚由 SPI 硬件控制还是软件控制*/
    uint16_t SPI_BaudRatePrescaler;  /*设置时钟分频因子，fpclk/分频数=fSCK */
    uint16_t SPI_FirstBit;           /*设置 MSB/LSB 先行 */
    uint16_t SPI_CRCPolynomial;      /*设置 CRC 校验的表达式 */
} SPI_InitTypeDef;
```

- SPI_NSS

本成员配置NSS引脚的使用模式，可以选择为硬件模式(SPI_NSS_Hard)与软件模式(SPI_NSS_Soft)，在硬件模式中的SPI片选信号由SPI硬件自动产生，而软件模式则需要亲自把相应的GPIO端口拉高或置低产生非片选和片选信号。

实际中软件模式应用比较多。

SPI初始化结构体

```
typedef struct
{
    uint16_t SPI_Direction;          /*设置 SPI 的单双向模式 */
    uint16_t SPI_Mode;              /*设置 SPI 的主/从机端模式 */
    uint16_t SPI_DataSize;           /*设置 SPI 的数据帧长度，可选 8/16 位 */
    uint16_t SPI_CPOL;              /*设置时钟极性 CPOL，可选高/低电平*/
    uint16_t SPI_CPHA;              /*设置时钟相位，可选奇/偶数边沿采样 */
    uint16_t SPI_NSS;              /*设置 NSS 引脚由 SPI 硬件控制还是软件控制*/
    uint16_t SPI_BaudRatePrescaler;  /*设置时钟分频因子，fpclk/分频数=fSCK */
    uint16_t SPI_FirstBit;           /*设置 MSB/LSB 先行 */
    uint16_t SPI_CRCPolynomial;      /*设置 CRC 校验的表达式 */
} SPI_InitTypeDef;
```

- SPI_BaudRatePrescaler

本成员设置波特率分频因子，分频后的时钟即为SPI的SCK信号线的时钟频率。这个成员参数可设置为fpclk的2、4、6、8、16、32、64、128、256分频。

SPI初始化结构体

```
typedef struct
{
    uint16_t SPI_Direction;          /*设置 SPI 的单双向模式 */
    uint16_t SPI_Mode;              /*设置 SPI 的主/从机端模式 */
    uint16_t SPI_DataSize;           /*设置 SPI 的数据帧长度，可选 8/16 位 */
    uint16_t SPI_CPOL;              /*设置时钟极性 CPOL，可选高/低电平*/
    uint16_t SPI_CPHA;              /*设置时钟相位，可选奇/偶数边沿采样 */
    uint16_t SPI_NSS;              /*设置 NSS 引脚由 SPI 硬件控制还是软件控制*/
    uint16_t SPI_BaudRatePrescaler;  /*设置时钟分频因子，fpclk/分频数=fSCK */
    uint16_t SPI_FirstBit;           /*设置 MSB/LSB 先行 */
    uint16_t SPI_CRCPolynomial;      /*设置 CRC 校验的表达式 */
} SPI_InitTypeDef;
```

- SPI_FirstBit

所有串行的通讯协议都会有MSB先行(高位数据在前)还是LSB先行(低位数据在前)的问题，而STM32的SPI模块可以通过这个结构体成员，对该特性编程控制。

SPI初始化结构体

```
typedef struct
{
    uint16_t SPI_Direction;          /*设置 SPI 的单双向模式 */
    uint16_t SPI_Mode;              /*设置 SPI 的主/从机端模式 */
    uint16_t SPI_DataSize;           /*设置 SPI 的数据帧长度，可选 8/16 位 */
    uint16_t SPI_CPOL;              /*设置时钟极性 CPOL，可选高/低电平*/
    uint16_t SPI_CPHA;              /*设置时钟相位，可选奇/偶数边沿采样 */
    uint16_t SPI_NSS;              /*设置 NSS 引脚由 SPI 硬件控制还是软件控制*/
    uint16_t SPI_BaudRatePrescaler;  /*设置时钟分频因子，fpclk/分频数=fSCK */
    uint16_t SPI_FirstBit;           /*设置 MSB/LSB 先行 */
    uint16_t SPI_CRCPolynomial;      /*设置 CRC 校验的表达式 */
} SPI_InitTypeDef;
```

- SPI_CRCPolynomial

这是SPI的CRC校验中的多项式，若我们使用CRC校验时，就使用这个成员的参数(多项式)，来计算CRC的值。

配置完这些结构体成员后，要调用SPI_Init函数把这些参数写入到寄存器中，实现SPI的初始化，然后调用SPI_Cmd来使能SPI外设。

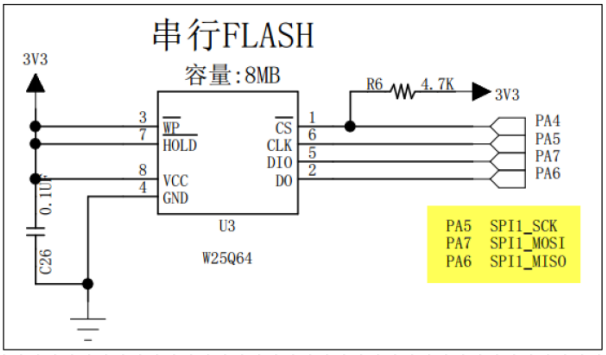
目录

CONTENTS.

STM32-7

33

STM32与SPI 串行 FLASH的 硬件连接



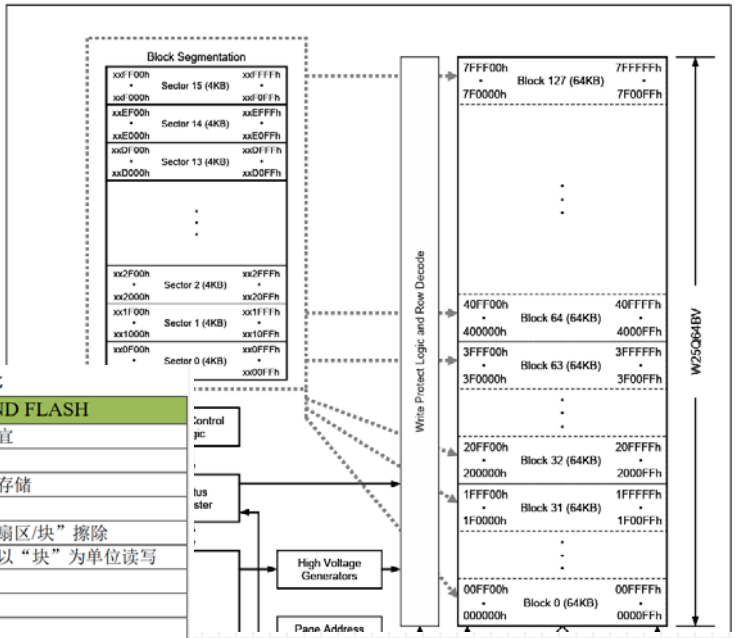
- FLASH 芯片(型号：W25Q64)是一种使用SPI 通讯协议的NOR FLASH 存储器 （注意：外部FLASH, 不是STM32的内部FLASH)
- CS/CLK/DIO/DO 引 脚 分 别 连 接 到 了 STM32 对 应 的 SPI 引 脚NSS/SCK/MOSI/MISO 上
- STM32 的 NSS 引脚虽然是其片上 SPI 外设的硬件引脚，但实际编程时通常把它当成一个普通的 GPIO，使用软件的方式控制 NSS 信号

FLASH介绍

FLASH 也是可重复擦写的储器，容量比EEPROM大得多，擦除操作一般是以“扇区/块”为单位。

根据存储单元电路的不同，FLASH存储器又分为NOR FLASH 和NAND FLASH。

9. BLOCK DIAGRAM



NOR FLASH 与 NAND FLASH 特性对比

特性	NOR FLASH	NAND FLASH
同容量存储器成本	较贵	较便宜
集成度	较低	较高
介质类型	随机存储	连续存储
地址线和数据线	独立分开	共用
擦除单元	以“扇区/块”擦除	以“扇区/块”擦除
读写单元	可以基于字节读写	必须以“块”为单位读写
读取速度	较高	较低
写入速度	较低	较高
坏块	较少	较多
是否支持 XIP	支持	不支持

35

示例程序 编程要点

本示例实现：使用 SPI 通讯的串行 FLASH 存储芯片的读写

(1) 初始化通讯使用的目标引脚及端口时钟；
(2) 使能 SPI 外设的时钟；
(3) 配置 SPI 外设的模式、地址、速率等参数并使能 SPI 外设；
(4) 编写基本SPI 按字节收发的函数；
(5) 编写对FLASH 擦除及读写操作的的函数；
(6) 编写测试程序，对读写数据进行校验。

36

SPI FLASH测试例程顶层代码

main.c代码

```
#include "stm32f10x.h"
#include "./led/bsp_led.h"
#include "./usart/bsp_usart.h"
#include "./flash/bsp_spi_flash.h"
#include <string.h>

uint8_t readBuff[4096];
uint8_t writeBuff[4096];

int main(void){

        uint32_t id;
        uint16_t i;

 LED_GPIO_Config();
 LED_BLUE;

 /* 串口初始化 */
 USART_Config();
  printf("\r\n 这是一个SPI-FLASH读写测
试例程 \r\n");

 SPI_FLASH_Init();

 id = SPI_Read_ID();
  printf("\r\n id =0x%x \r\n",id);

 SPI_Erase_Sector(0);

  for(i=0;i<25;i++)        {
       writeBuff[i]=i+25;
  }

 SPI_Write_Data(0,writeBuff,25);

 SPI_Read_Data(0,readBuff,4096);
  for(i=0;i<4096;i++) {
         printf("0x%x ",readBuff[i]);
         if(i%10==0) printf("\r\n");
       }

 while (1)  {      }
}
```


（1）初始化通讯使用的目标引脚及端口时钟

bsp_spi_flash.h 定义SPI引脚

```
#include "stm32f10x.h"

#define         FLASH_SPIx                      SPI1
#define         FLASH_SPI_APBxClock_FUN             RCC_APB2PeriphClockCmd
#define         FLASH_SPI_CLK                   RCC_APB2Periph_SPI1
#define         FLASH_SPI_GPIO_APBxClock_FUN        RCC_APB2PeriphClockCmd

#define         FLASH_SPI_SCK_PORT              GPIOA
#define         FLASH_SPI_SCK_PIN               GPIO_Pin_5

#define         FLASH_SPI_MOSI_PORT             GPIOA
#define         FLASH_SPI_MOSI_PIN              GPIO_Pin_7

#define         FLASH_SPI_MISO_PORT             GPIOA
#define         FLASH_SPI_MISO_PIN              GPIO_Pin_6

#define         FLASH_SPI_GPIO_CLK              RCC_APB2Periph_GPIOA
#define         FLASH_SPI_CS_PORT               GPIOA
#define         FLASH_SPI_CS_PIN                GPIO_Pin_4


//CS引脚配置
#define FLASH_SPI_CS_HIGH                   GPIO_SetBits(FLASH_SPI_CS_PORT,FLASH_SPI_CS_PIN);
#define FLASH_SPI_CS_LOW                    GPIO_ResetBits(FLASH_SPI_CS_PORT,FLASH_SPI_CS_PIN);
```

（1）初始化通讯使用的目标引脚及端口时钟 bsp_spi_flash.h 定义命令字节、声明SPI操作函数

```
/*等待超时时间*/
#define SPIT_FLAG_TIMEOUT         ((uint32_t)0x1000)
#define SPIT_LONG_TIMEOUT         ((uint32_t)(10 *
SPIT_FLAG_TIMEOUT))

#define DUMMY      0x00
#define READ_JEDEC_ID     0x9f
#define ERASE_SECTOR      0x20
#define READ_STATUS       0x05
#define READ_DATA         0x03
#define WRITE_ENABLE      0x06
#define WRITE_DATA        0x02

void SPI_FLASH_Init(void);
uint32_t SPI_Read_ID(void);
void SPI_Erase_Sector(uint32_t addr);
void SPI_Read_Data(uint32_t addr,uint8_t *readBuff,uint32_t
numByteToRead);
void SPI_Write_Data(uint32_t addr,uint8_t *writeBuff,uint32_t
numByteToWrite);

void SPI_WaitForWriteEnd(void);
```

```
/*FLASH 常用命令*/
#define W25X_WriteEnable              0x06
#define W25X_WriteDisable             0x04
#define W25X_ReadStatusReg            0x05
#define W25X_WriteStatusReg           0x01
#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
/*其它*/
#define  sFLASH_ID                    0XEF4017
#define Dummy_Byte                    0xFF
```

39

（1）初始化通讯使用的目标引脚及端口时钟

bsp_spi_flash.c代码

```
static void SPI_GPIO_Config(void)  {
  GPIO_InitTypeDef  GPIO_InitStructure;

/* 使能 SPI 引脚相关的时钟 */
  FLASH_SPI_GPIO_APBxClock_FUN ( FLASH_SPI_GPIO_CLK, ENABLE );

/* 配置 SPI 的 SCK 引脚*/
  GPIO_InitStructure.GPIO_Pin = FLASH_SPI_SCK_PIN;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
  GPIO_Init(FLASH_SPI_SCK_PORT, &GPIO_InitStructure);

/* 配置 SPI 的 MOSI 引脚*/
  GPIO_InitStructure.GPIO_Pin = FLASH_SPI_MOSI_PIN;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
  GPIO_Init(FLASH_SPI_MOSI_PORT, &GPIO_InitStructure);

/* 配置 SPI 的 MISO引脚*/
  GPIO_InitStructure.GPIO_Pin = FLASH_SPI_MISO_PIN;
  //GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  //GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
  GPIO_Init(FLASH_SPI_MISO_PORT, &GPIO_InitStructure);
  //初始化CS引脚，使用软件控制，所以直接设置成推挽输出
  GPIO_InitStructure.GPIO_Pin = FLASH_SPI_CS_PIN;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
  GPIO_Init(FLASH_SPI_CS_PORT, &GPIO_InitStructure);

/* 停止信号 FLASH: CS 引脚高电平*/
  FLASH_SPI_CS_HIGH;
}
```

（2）使能 SPI 外设的时钟；

（3）配置 SPI 外设的模式、地址、速率等参数并使能 SPI 外设；

bsp_spi_flash.c代码

```
static void SPI_Mode_Config(void)  {

  SPI_InitTypeDef  SPI_InitStructure;
  /* 使能 SPI 外设时钟 */
  FLASH_SPI_APBxClock_FUN ( FLASH_SPI_CLK, ENABLE );
  /* 配置 SPI波特率 */
  SPI_InitStructure.SPI_BaudRatePrescaler = SPI_BaudRatePrescaler_2 ;
  //SPI 使用模式3
  SPI_InitStructure.SPI_CPHA = SPI_CPHA_2Edge ;
  SPI_InitStructure.SPI_CPOL = SPI_CPOL_High ;
  SPI_InitStructure.SPI_CRCPolynomial = 0;//不使用CRC功能，数值随便写
  SPI_InitStructure.SPI_DataSize = SPI_DataSize_8b;
  SPI_InitStructure.SPI_Direction = SPI_Direction_2Lines_FullDuplex ;//双线全双工
  SPI_InitStructure.SPI_FirstBit = SPI_FirstBit_MSB  ;
  SPI_InitStructure.SPI_Mode = SPI_Mode_Master  ;
  SPI_InitStructure.SPI_NSS = SPI_NSS_Soft  ;
  //写入配置到寄存器
  SPI_Init(FLASH_SPIx,&SPI_InitStructure);
 //使能SPI
  SPI_Cmd(FLASH_SPIx,ENABLE);
}
```

根据FLASH芯片的说明，它支持SPI模式0(CPOL=0，CPHA=0)及模式3(CPOL=1，CPHA=1)，支持双线全双工，使用MSB先行模式，支持最高通讯时钟为104MHz，数据帧长度为8位。

```
void SPI_FLASH_Init(void){

    SPI_GPIO_Config();
    SPI_Mode_Config();
}
```

41

（4）编写基本SPI按字节收发的函数；

bsp_spi_flash.c代码

```
static __IO uint32_t  SPITimeout = SPIT_LONG_TIMEOUT;
static uint32_t SPI_TIMEOUT_UserCallback(uint8_t errorCode);

//发送并接收一个字节
uint8_t SPI_FLASH_Send_Byte(uint8_t data) {
        SPITimeout = SPIT_FLAG_TIMEOUT;
        //通过检测 TXE 标志，获取发送缓冲区的状态，若发送缓冲区为空，则表示可能存在的上一个数据已经发送完毕；
        while(SPI_I2S_GetFlagStatus(FLASH_SPIx,SPI_I2S_FLAG_TXE) == RESET)    {
                if((SPITimeout--) == 0) return SPI_TIMEOUT_UserCallback(0);
        }

        //程序执行到此处，TX缓冲区已空
        SPI_I2S_SendData (FLASH_SPIx,data);

        SPITimeout = SPIT_FLAG_TIMEOUT;
        //检查并等待至RX缓冲区为非空
        while(SPI_I2S_GetFlagStatus(FLASH_SPIx,SPI_I2S_FLAG_RXNE) == RESET)
        {
                if((SPITimeout--) == 0) return SPI_TIMEOUT_UserCallback(0);
        }

        //程序执行到此处，说明数据发送完毕，并接收到一字字节
        return SPI_I2S_ReceiveData(FLASH_SPIx);
}
/*使用 SPI 读取一个字节的数据*/
uint8_t SPI_FLASH_Read_Byte(void){
        return SPI_FLASH_Send_Byte(DUMMY);
}
```

```
static  uint32_t
SPI_TIMEOUT_UserCallback(uint8_t
errorCode)
{
  /* Block communication and all processes
*/
  FLASH_ERROR("SPI 等待超时!errorCode
= %d",errorCode);

  return 0;
}
```

42

(5) 编写对FLASH 擦除及读写操作的的函数；

根据“JEDEC”指令的时序，把读取 FLASH ID 的过程编写成一个函数

```c
/**
  * @brief  读取 FLASH ID
  * @param  无
  * @retval FLASH ID
  */
u32 SPI_FLASH_ReadID(void)
{
    u32 Temp = 0, Temp0 = 0, Temp1 = 0, Temp2 = 0;

    /* 开始通讯：CS 低电平 */
    SPI_FLASH_CS_LOW();

    /* 发送 JEDEC 指令，读取 ID */
    SPI_FLASH_SendByte(W25X_JedecDeviceID);

    /* 读取一个字节数据 */
    Temp0 = SPI_FLASH_SendByte(Dummy_Byte);

    /* 读取一个字节数据 */
    Temp1 = SPI_FLASH_SendByte(Dummy_Byte);

    /* 读取一个字节数据 */
    Temp2 = SPI_FLASH_SendByte(Dummy_Byte);

    /* 停止通讯：CS 高电平 */
    SPI_FLASH_CS_HIGH();

    /*把数据组合起来，作为函数的返回值*/
    Temp = (Temp0 << 16) | (Temp1 << 8) | Temp2;

    return Temp;
}
```

```c
#define W25X_JedecDeviceID    0x9F
```

11.2.2 Instruction Set Table 1 (1)

INSTRUCTION NAME	BYTE 1 (CODE)	BYTE 2	BYTE 3	BYTE 4	BYTE 5	BYTE 6
Write Enable	06h					
Read Status Register-1	05h	(S7–S0) (2)				
Page Program	02h	A23–A16	A15–A8	A7–A0	(D7–D0)	
Sector Erase (4KB)	20h	A23–A16	A15–A8	A7–A0		
JEDEC ID	9Fh	(MF7-MF0) Manufacturer	(ID15-ID8) Memory Type	(ID7-ID0) Capacity		

11.2.3 Instruction Set Table 2 (Read Instructions)

INSTRUCTION NAME	BYTE 1 (CODE)	BYTE 2	BYTE 3	BYTE 4	BYTE 5	BYTE 6
Read Data	03h	A23-A16	A15-A8	A7-A0	(D7-D0)	

(5) 编写对FLASH 擦除及读写操作的的函数；

```c
#define W25X_WriteEnable 0x06
```

FLASH 写使能以及读取当前状态

```c
/**
  * @brief  向 FLASH 发送 写使能 命令
  * @param  none
  * @retval none
  */
void SPI_FLASH_WriteEnable(void)
{
    /* 通讯开始：CS 低 */
    SPI_FLASH_CS_LOW();

    /* 发送写使能命令*/
    SPI_FLASH_SendByte(W25X_WriteEnable);

    /*通讯结束：CS 高 */
    SPI_FLASH_CS_HIGH();
}
```

与 EEPROM 一样，由于 FLASH 芯片向内部存储矩阵写入数据需要消耗一定的时间， 并不是在总线通讯结束的一瞬间完成的，所以在写操作后需要确认FLASH 芯片“空闲”时才能进行再次写入。

```c
#define W25X_ReadStatusReg  0x05
```

以下函数通过读状态寄存器等待 FLASH 芯片空闲

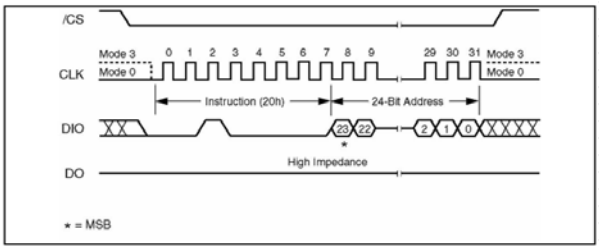
```c
/* WIP(busy)标志，FLASH 内部正在写入 */
#define WIP_Flag                  0x01

/**
  * @brief  等待 WIP(BUSY)标志被置 0，即等待到 FLASH 内部数据写入完毕
  * @param  none
  * @retval none
  */
void SPI_FLASH_WaitForWriteEnd(void)
{
    u8 FLASH_Status = 0;

    /* 选择 FLASH: CS 低 */
    SPI_FLASH_CS_LOW();

    /* 发送 读状态寄存器 命令 */
    SPI_FLASH_SendByte(W25X_ReadStatusReg);

    /* 若 FLASH 忙碌，则等待 */
    do
    {
        /* 读取 FLASH 芯片的状态寄存器 */
        FLASH_Status = SPI_FLASH_SendByte(Dummy_Byte);
    }
    while ((FLASH_Status & WIP_Flag) == SET);  /* 正在写入标志 */

    /* 停止信号  FLASH: CS 高 */
    SPI_FLASH_CS_HIGH();
}
```

（5）编写对FLASH 擦除及读写操作的的函数；

```
#define W25X_SectorErase  0x20
```

FLASH 扇区擦除
由于FLASH 存储器的特性决定了它只能把原来为“1”的数据位改写成“0”，而原来为“0”的数据位不能直接改写为“1”。所以这里涉及到数据“擦除”的概念，在写入前，必须要对目标存储矩阵进行擦除操作，把矩阵中的数据位擦除为“1”

对存储矩阵擦除的基本操作单位都是多个字节进行，如本FLASH 芯片 支持“扇区擦除”(4KB)“块擦除”(64KB)以及“整片擦除”



扇区擦除时序

```
/**
 * @brief  擦除 FLASH 扇区
 * @param  SectorAddr: 要擦除的扇区地址
 * @retval 无
 */
void SPI_FLASH_SectorErase(u32 SectorAddr)
{
    /* 发送 FLASH 写使能命令 */
    SPI_FLASH_WriteEnable();
    SPI_FLASH_WaitForWriteEnd();
    /* 擦除扇区 */
    /* 选择 FLASH: CS 低电平 */
    SPI_FLASH_CS_LOW();
    /* 发送扇区擦除指令*/
    SPI_FLASH_SendByte(W25X_SectorErase);
    /*发送擦除扇区地址的高位*/
    SPI_FLASH_SendByte((SectorAddr & 0xFF0000) >> 16);
    /* 发送擦除扇区地址的中位 */
    SPI_FLASH_SendByte((SectorAddr & 0xFF00) >> 8);
    /* 发送擦除扇区地址的低位 */
    SPI_FLASH_SendByte(SectorAddr & 0xFF);
    /* 停止信号 FLASH: CS 高电平 */
    SPI_FLASH_CS_HIGH();
    /* 等待擦除完毕*/
    SPI_FLASH_WaitForWriteEnd();
}
```


（5）编写对FLASH 擦除及读写操作的的函数；

```
//向FLASH写入内容          #define WRITE_DATA        0x02
//读取FLASH的内容
void SPI_Write_Data(uint32_t addr,uint8_t *writeBuff,uint32_t numByteToWrite)
{
        SPI_FLASH_WriteEnable();
        //片选使能
        FLASH_SPI_CS_LOW;
        SPI_FLASH_Send_Byte(WRITE_DATA);
        SPI_FLASH_Send_Byte((addr>>16)&0xff);
        SPI_FLASH_Send_Byte((addr>>8)&0xff);
        SPI_FLASH_Send_Byte(addr&0xff);

        while(numByteToWrite--)
        {
                SPI_FLASH_Send_Byte(*writeBuff);
                writeBuff++;
        }

        FLASH_SPI_CS_HIGH;
        SPI_WaitForWriteEnd();
}
```

(5) 编写对FLASH 擦除及读写操作的的函数;

```
#define READ_DATA 0x03

//读取FLASH的内容
void SPI_Read_Data(uint32_t addr,uint8_t *readBuff,uint32_t numByteToRead)
{
        //片选使能
        FLASH_SPI_CS_LOW;

        SPI_FLASH_Send_Byte(READ_DATA);

        SPI_FLASH_Send_Byte((addr>>16)&0xff);

        SPI_FLASH_Send_Byte((addr>>8)&0xff);

        SPI_FLASH_Send_Byte(addr&0xff);

        while(numByteToRead--)
        {
                *readBuff = SPI_FLASH_Send_Byte(DUMMY);
                readBuff++;
        }

        FLASH_SPI_CS_HIGH;

}
```


运行测试结果

```
printf("\r\n 这是一个SPI-FLASH读写测试例程 \r\n");

SPI_FLASH_Init();
id = SPI_Read_ID();   printf("\r\n id =0x%x \r\n",id);
SPI_Erase_Sector(0);

for(i=0;i<25;i++)   writeBuff[i]=i+25;
SPI_Write_Data(0,writeBuff,25);

SPI_Read_Data(0,readBuff,4096);
for(i=0;i<4096;i++) {
      printf("0x%x ",readBuff[i]);
      if(i%10==0) printf("\r\n");
}
```

第0扇区的4096字节都读出来了

代码缺陷：不能跨页 ； W25Q64一页256B