第15章 SPI控制器

15.1 SPI协议简介
15.2 SPI控制器
15.3 SPI应用步骤及常用库函数
15.4 应用实例

电子与通信工程系

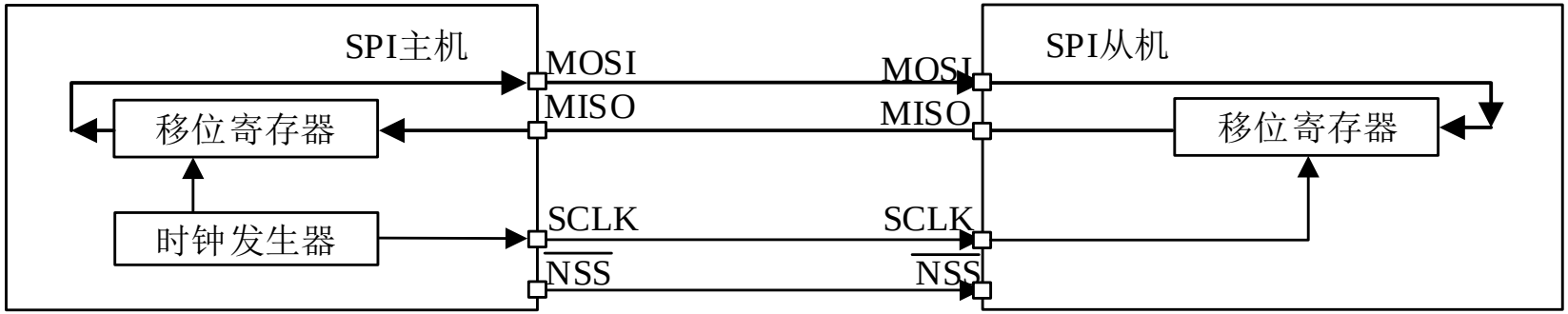
华东理工大学信息学院

15.1 SPI协议简介

15.1 SPI协议简介

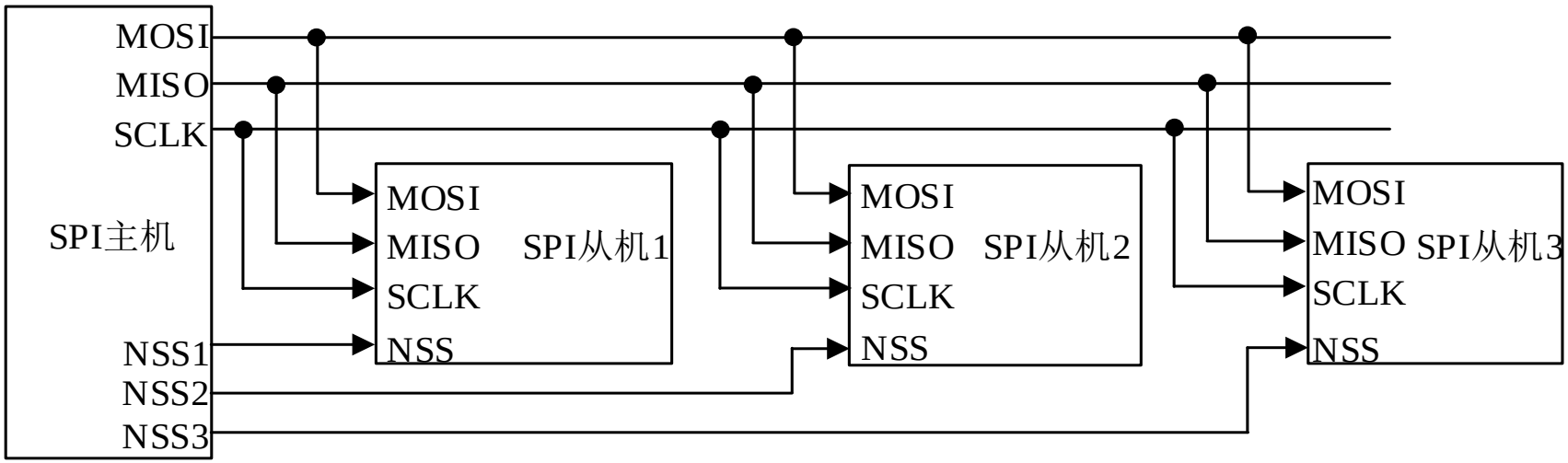
1、SPI是Motorola首先提出的全双工四线同步串行外围接口，采用主从模式（Master-Slave）架构。支持单主多从模式应用，时钟由Master控制，在时钟移位脉冲下，数据按位传输，高位在前，低位在后（MSB first）。

2、4线SPI器件有四个信号：时钟(SPI CLK, SCLK)、主机输出从机输入(MOSI)、主机输入从机输出(MISO)、片选(CS/NSS)。



15.1 SPI协议简介

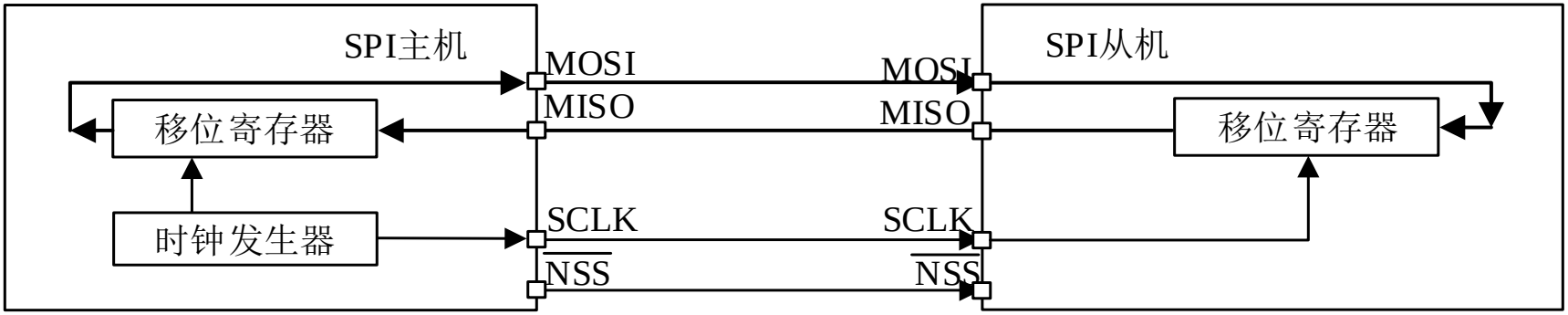
3、产生时钟信号的器件称为主机。主机和从机之间传输的数据与主机产生的时钟同步。



15.1 SPI协议简介

15.1.1 SPI数据传输

1、在MOSI、MISO和SPI主从机内部的数据寄存器构成一个数据串行传输的环路，在时钟SCLK的控制下实现数据的环形传输。



要开始SPI通信，主机必须发送时钟信号，并通过使能NSS信号选择从机。

15.1 SPI协议简介

15.1.2 时钟极性和时钟相位

1、CPOL极性(Clock Polarity ,CPOL)

SPI的CPOL，表示当总线空闲时，SCLK的极性，其电平的值是低电平0还是高电平1：

- **CPOL=0，时钟空闲时候的电平是低电平，所以当SCLK有效的时候，就是高电平；**
- **CPOL=1，时钟空闲时候的电平是高电平，所以当SCLK有效的时候，就是低电平。**

2、CPHA相位(Clock Phase ,CPHA)

CPHA相位对应着数据采样是在第一个边沿（空闲态转电平切换到相反电平）还是第二个边沿，0对应着第一个边沿，1对应着第二个边沿。

15.1 SPI协议简介

15.1.3 四种SPI模式

主机必须根据从机的要求选择时钟极性和时钟相位。

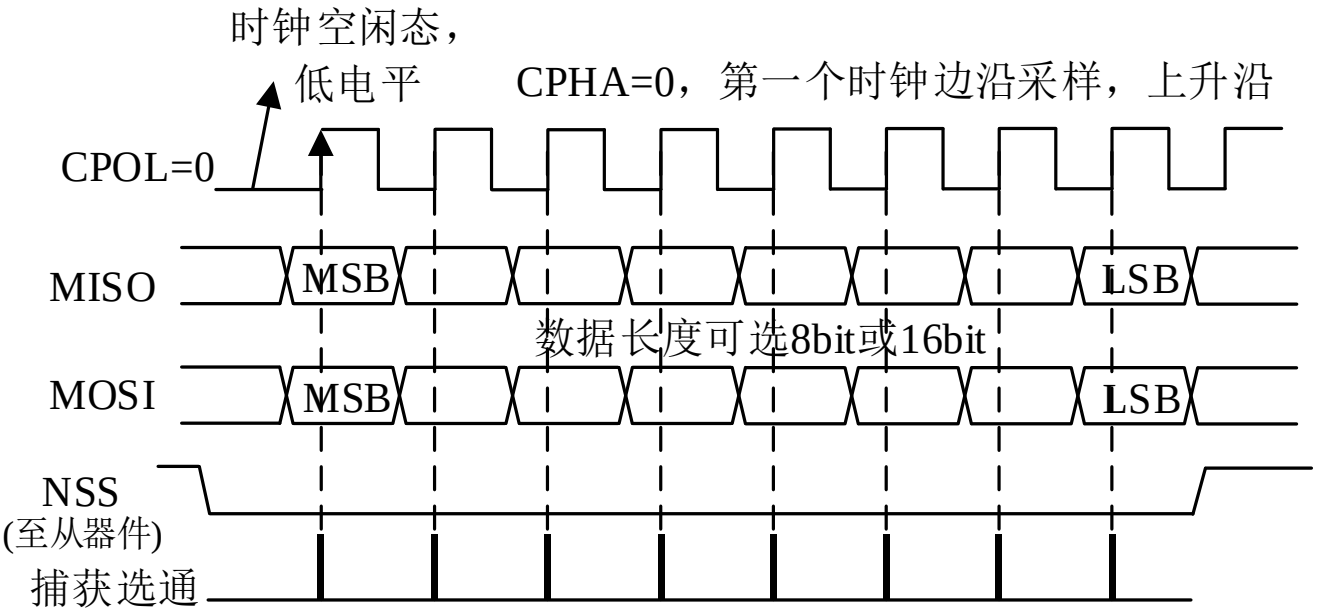
根据CPOL和CPHA位的选择，有四种SPI模式可用。

SPI 模式	CPOL	CPHA	空闲时时钟极性	采样/移位数据的时钟相位
0	0	0	低电平	第一个边沿，数据在上升沿采样，在下降沿移出
1	0	1	低电平	第二个边沿，数据在下降沿采样，在上升沿移出
2	1	0	高电平	第一个边沿，数据在下降沿采样，在上升沿移出
3	1	1	高电平	第二个边沿，数据在上升沿采样，在下降沿移出

15.1 SPI协议简介

15.1.3 四种SPI模式

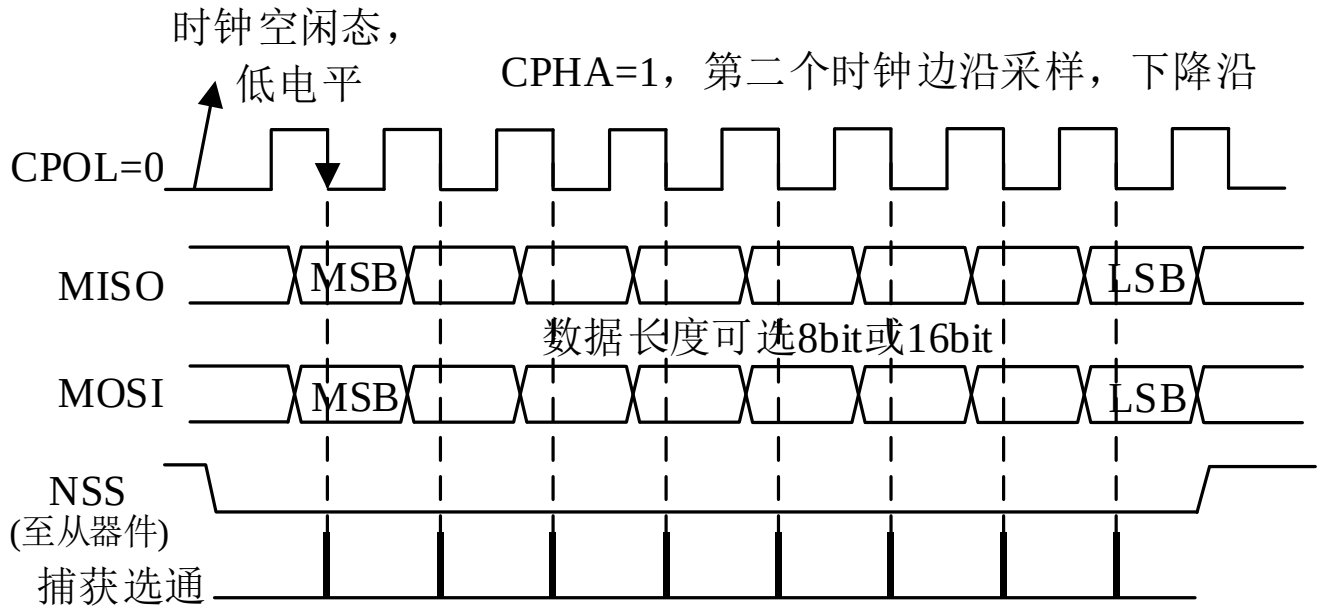
1、模式0：在此模式下，时钟极性CPOL=0，表示时钟信号的空闲状态为低电平。时钟相位CPHA=0，数据在第一个边沿（上升沿）采样，并且数据在时钟信号接下来的下降沿移出。



15.1 SPI协议简介

15.1.3 四种SPI模式

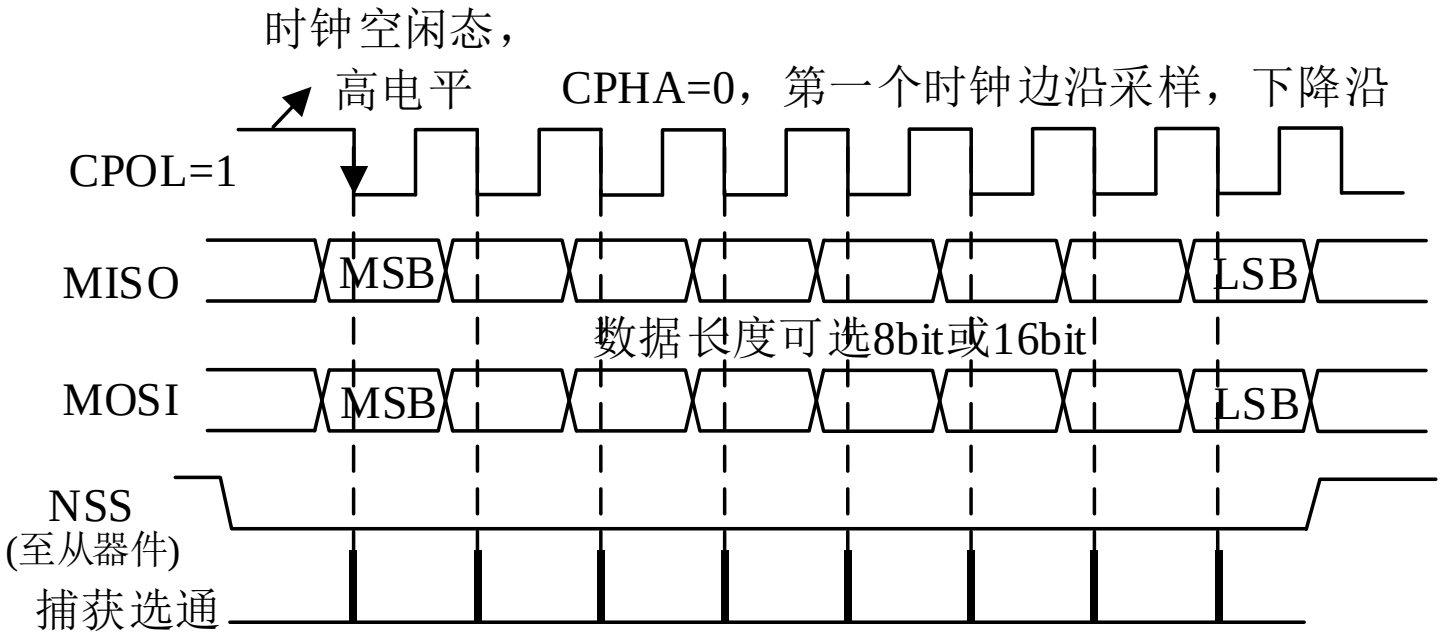
2、模式1：在此模式下，时钟极性CPOL=0，表示时钟信号的空闲状态为低电平。时钟相位CPHA=1，数据在第二个边沿（下降沿）采样，并且数据在时钟信号接下来的上升沿移出。



15.1 SPI协议简介

15.1.3 四种SPI模式

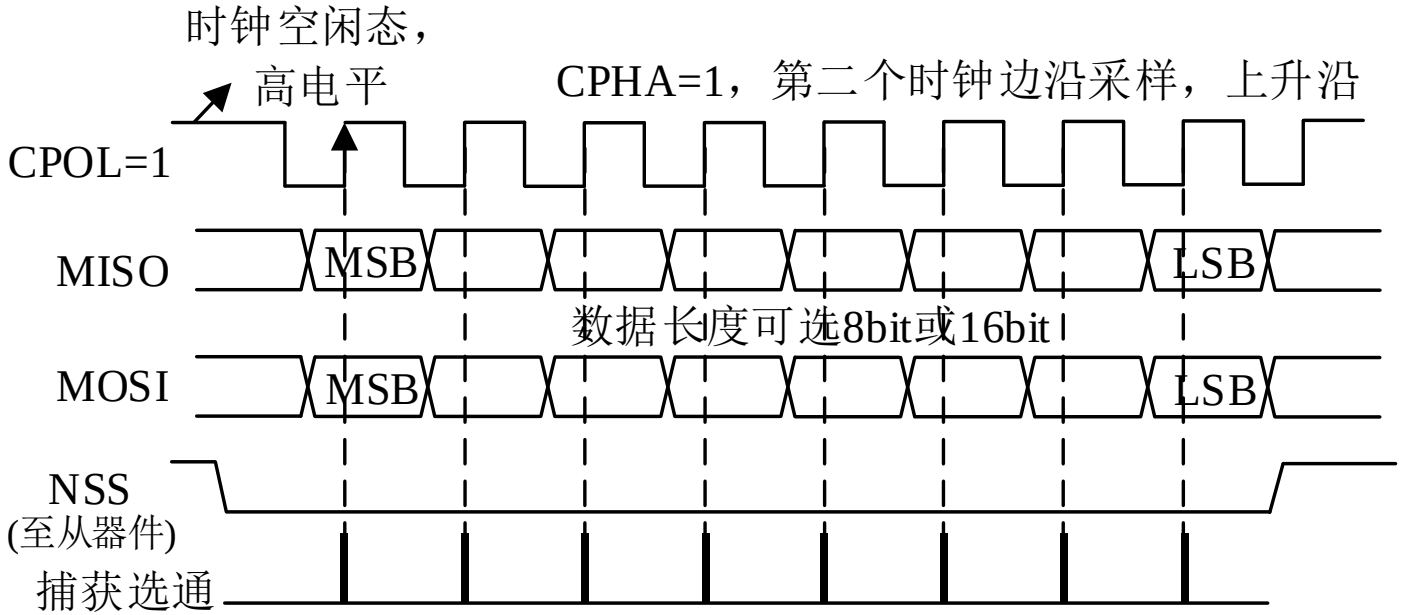
3、模式2：在此模式下，时钟极性CPOL=1，表示时钟信号的空闲状态为高电平。时钟相位CPHA=0，数据在第一个边沿（下降沿）采样，并且数据在时钟信号接下来的上升沿移出。



15.1 SPI协议简介

15.1.3 四种SPI模式

4、模式3：在此模式下，时钟极性CPOL=1，表示时钟信号的空闲状态为高电平。时钟相位CPHA=1，数据在第二个边沿（上升沿）采样，并且数据在时钟信号接下来的下降沿移出。



15.2 SPI控制器

15.2 SPI控制器

15.2.1 概述

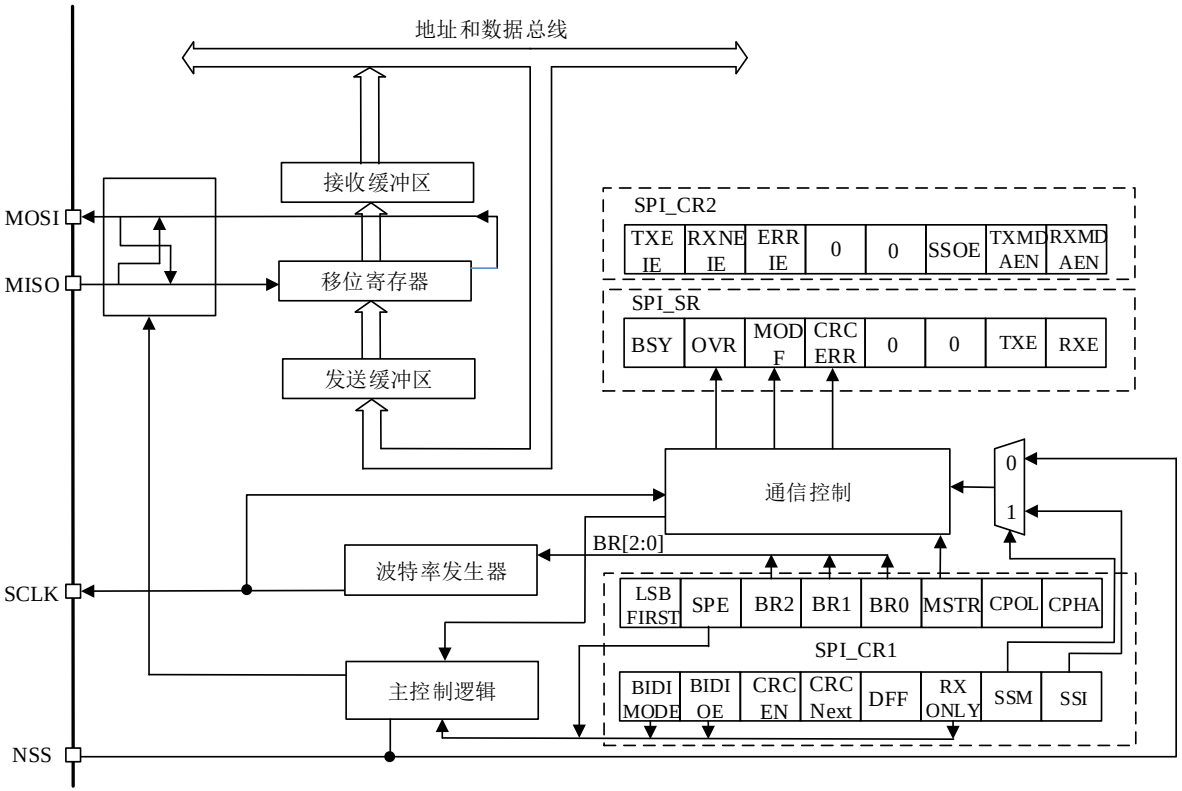
STM32F429内部有6个SPI控制器，可与外部器件进行半双工/全双工的同步串行通信。

SPI控制器主要有以下特性：

- 全双工同步传输；
- 8位或 16 位传输帧格式选择；
- 支持最高的SCK时钟频率为 $f_{pclk}/2$
- 主模式或从模式操作、多主模式功能；
- 可编程的时钟极性和相位；
- 可编程的数据顺序，先移位 MSB 或 LSB；
- 可触发中断的专用发送和接收标志；
- 具有 DMA 功能。

15.2 SPI控制器

15.2.2 结构



15.2 SPI控制器

15.2.2 结构

STM32 芯片有多个 SPI 外设，它们的 SPI 通讯引脚（MOSI、MISO、SCLK 及 NSS）通过 GPIO 引脚复用映射实现。

	SPI1	SPI2	SPI3	SPI4	SPI5	SPI6
总线	APB2	APB1	APB1	APB2	APB2	APB2
MOSI	PA7/PB5	PB15/PC3/PI3	PB5/PC12/PD6	PE6/PE14	PF9/PF11	PG14
MISO	PA6/PB4	PB14/PC2/PI2	PB4/PC11	PE5/PE13	PF8/PH7	PG12
SCLK	PA5/PB3	PB10/PB13/PD3	PB3/PC10	PE2/PE12	PF7/PH6	PG13
NSS	PA4/PA15	PB9/PB12/PI0	PA4/PA15	PE4/PE11	PF6/PH5	PG8

6个SPI控制器中SPI1、SPI4、SPI5、SPI6 挂载在 APB2总线上，最高通信速率达45Mbtis/s，SPI2、SPI3挂载在 APB1 总线上，最高通信速率为 22.5Mbits/s。

15.2 SPI控制器

15.2.3 SPI主机配置

在此配置中，MOSI 引脚为数据输出，MISO 引脚为数据输入。

1、发送数据：在发送缓冲区中写入字节时，SPI控制器开始发送数据。当移位寄存器中的数据都串行输出之后，数据从发送缓冲区传输到移位寄存器，并将TXE 标志置 1，并且在 SPI_CR2寄存器中的 TXEIE 位置 1 时将生成中断。仅当 TXE 标志为 1 时，才可以对发送缓冲区执行写操作。

2、接收数据：对于接收器，在数据传输完成时：移位寄存器中的数据将传输到接收缓冲区，并且 RXNE 标志置 1。如果 SPI_CR2 寄存器中的 RXNEIE 位置 1，则生成中断。在出现最后一个采样时钟边沿时，RXNE 位置 1，移位寄存器中接收的数据字节被拷贝到接收缓冲区中。通过读取 SPI_DR 寄存器获取接收到的数据，并将 RXNE 位清零。

15.2 SPI控制器

15.2.4 SPI从机配置

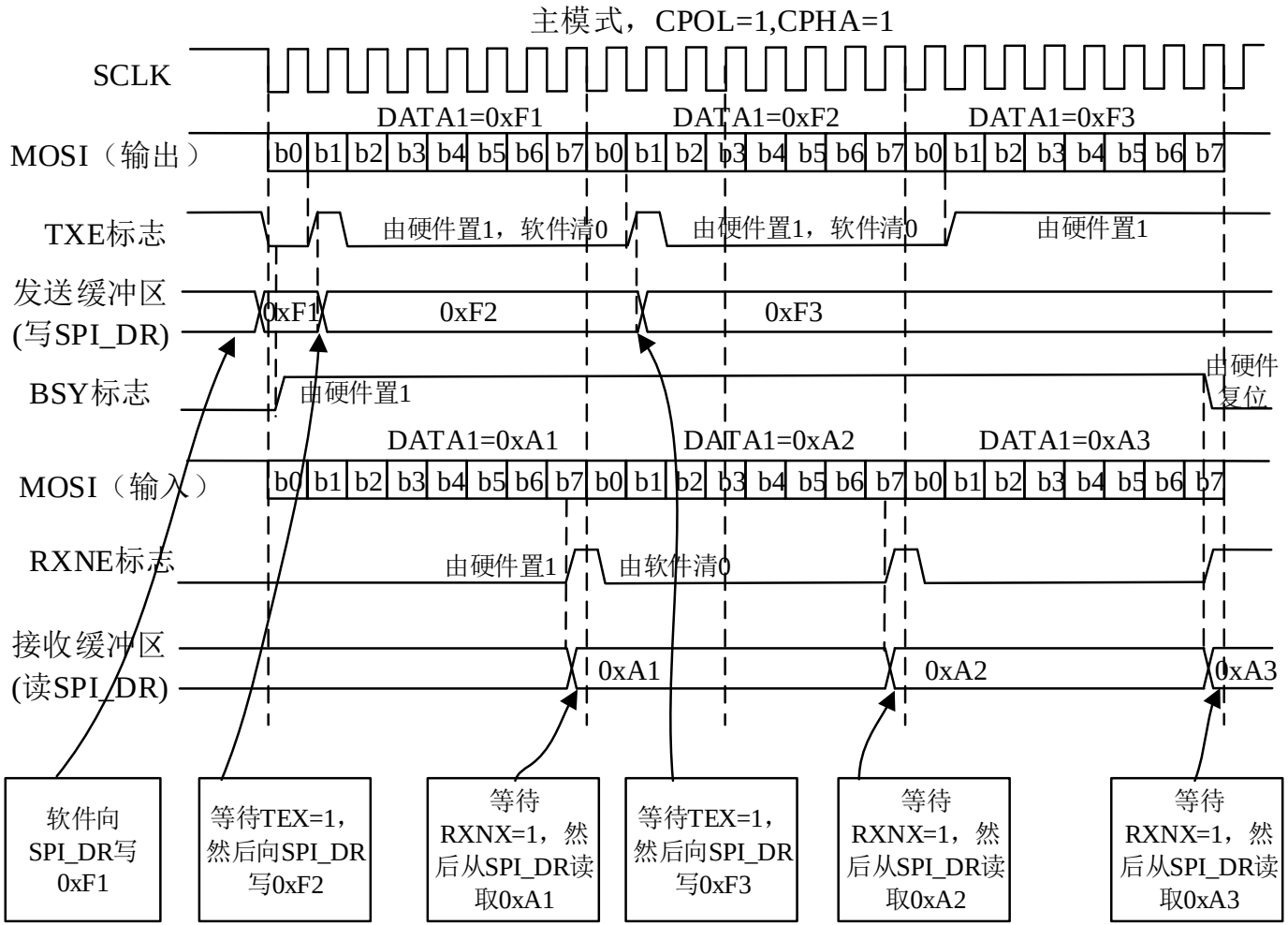
在此配置中，从 SCK 引脚上接收主器件的串行时钟。

1、发送数据：数据字节在写周期内被并行加载到发送缓冲区中，当从器件收到时钟信号和数据的高有效位时，开始发送数据。SPI_SR 寄存器中的 TXE 标志在数据从发送缓冲区传输到移位寄存器时置 1，并且在SPI_CR2 寄存器中的 TXEIE 位置 1 时将生成中断。

2、接收数据：对于接收器，在数据传输完成时：移位寄存器中的数据将传输到接收缓冲区，并且 RXNE 标志（SPI_SR 寄存器）置 1。如果 SPI_CR2 寄存器中的 RXNEIE 位置 1，则生成中断。通过读取SPI_DR寄存器获取接收到的数据，并将 RXNE 位清零。

15.2 SPI控制器

15.2.5 主模式的全双工收发过程



15.2 SPI控制器

15.2.6 SPI状态标志

软件可通过三种状态标志监视 SPI 总线的状态。

1、发送缓冲区为空 (TXE)：此标志置1时，表示发送缓冲区为空，可以将待发送的下一个数据加载到缓冲区中。对SPI_DR 寄存器执行写操作时，将清零 TXE 标志。

2、接收缓冲区非空 (RXNE)：此标志置 1 时，表示接收缓冲区中存在有效的已接收数据。读取 SPI_DR 时，将清零该标志。

3、BUSY：BSY 标志由硬件置 1 和清零（对此标志执行写操作没有任何作用）。BSY 标志用于指示SPI 通信的状态。BSY 置 1 时，表示 SPI 正忙于通信。在主模式下的双向通信接收模式（MSTR=1 且 BDM=1且 BDOE=0）有一个例外情况，BSY 标志在接收过程中保持低电平。

15.2 SPI控制器

15.2.7 SPI中断

中断事件	事件标志	使能控制位
发送缓冲区为空	TXE	TXEIE
接收缓冲区非空	RXNE	RXNEIE
主模式故障	MODF	ERRIE
溢出错误	OVR	
CRC 错误	CRCERR	
TI 帧格式错误	FRE	ERRIE

15.3 SPI应用步骤及常用库函数

15.3 SPI应用步骤及常用库函数

15.3.1 SPI典型应用步骤

以SPI5工作在全双工主模式为例，使用PF6、PF7、PF8、PF9分别作为作为SPI1的NSS、SCLK、MISO和MOSI的复用引脚。

1、开启SPI5控制器时钟和通信线复用引脚端口GPIOF的时钟。

使能SPI5时钟：

```
RCC_APB2PeriphClockCmd (RCC_APB2Periph_SPI5, ENABLE);
```

使能GPIOA时钟：

```
RCC_AHB1PeriphClockCmd (RCC_AHB1Periph_GPIOA, ENABLE);
```

15.3 SPI应用步骤及常用库函数

15.3.1 SPI典型应用步骤

2、初始化引脚

复用PF6~PF9到SPI5：

```
GPIO_PinAFConfig(GPIOF,GPIO_PinSource7,GPIO_AF_SPI5);
GPIO_PinAFConfig(GPIOF,GPIO_PinSource8,GPIO_AF_SPI5);
GPIO_PinAFConfig(GPIOF,GPIO_PinSource9,GPIO_AF_SPI5);
```

将引脚设置为复用模式，并初始化：

```
GPIO_Init(GPIOF, &GPIO_InitStructure);
```

3、初始化SPI5控制器工作模式。

```
SPI_Init(SPI5, &SPI_InitStructure);
```

4、使能SPI5控制器

```
SPI_Cmd(SPI5, ENABLE);
```

5、中断使能

如果需要使用中断，需要配置NVIC和使能相应的SPI5中断事件。

15.3 SPI应用步骤及常用库函数

15.3.2 常用库函数

头文件：**stm32f4xx_spi.h**
源文件：**stm32f4xx_spi.c**

1、SPI初始化函数

void　void SPI_Init(SPI_TypeDef*　SPIx, SPI_InitTypeDef*　SPI_InitStruct);

1）、参数1：SPI_TypeDef*　SPIx，SPI应用对象，是一个结构体指针

```
#define SPI1    ((SPI_TypeDef *) SPI1_BASE)
```

2）、参数2：SPI_InitTypeDef*　SPI_InitStruct，是SPI应用对象初始化结构体指针

```
typedef struct
{
  uint16_t  SPI_Direction;   //定义SPI单向还是双向传输
  uint16_t  SPI_Mode;        //定义SPI的主从模式
  uint16_t  SPI_DataSize;      //定义SPI传输数据宽度
  uint16_t  SPI_CPOL;        // SPI的CPOL极性，定义当SPI空闲时，SCLK的极性
  uint16_t  SPI_CPHA;        // SPI的CPHA 相位
  uint16_t  SPI_NSS;        //NSS引脚管理方式
  uint16_t  SPI_BaudRatePrescaler;    //时钟频率
  uint16_t  SPI_FirstBit;         //最先输出的数据位
  uint16_t  SPI_CRCPolynomial;  //CRC多项式
}SPI_InitTypeDef;
```

15.3 SPI应用步骤及常用库函数

15.3.2 常用库函数

头文件：**stm32f4xx_spi.h**
源文件：**stm32f4xx_spi.c**

```
SPI_InitTypeDef  SPI_InitStructure;


/* FLASH_SPI 模式配置 */
// FLASH芯片 支持SPI模式0及模式3，据此设置CPOL CPHA
SPI_InitStructure.SPI_Direction = SPI_Direction_2Lines_FullDuplex;
SPI_InitStructure.SPI_Mode = SPI_Mode_Master;//主机模式
SPI_InitStructure.SPI_DataSize = SPI_DataSize_8b;//数据位=8bit
SPI_InitStructure.SPI_CPOL = SPI_CPOL_High;//SCK 引脚在 空闲状态处于高电平
SPI_InitStructure.SPI_CPHA = SPI_CPHA_2Edge;//SCK第二个边沿采样数据线
SPI_InitStructure.SPI_NSS = SPI_NSS_Soft;//内部 从器件管理
SPI_InitStructure.SPI_BaudRatePrescaler = SPI_BaudRatePrescaler_2;//波特
率预分频器
SPI_InitStructure.SPI_FirstBit = SPI_FirstBit_MSB;//先发送数据高位
SPI_InitStructure.SPI_CRCPolynomial = 7;//
SPI_Init(SPI5, &SPI_InitStructure);
```

15.3 SPI应用步骤及常用库函数

15.3.2 常用库函数

头文件：**stm32f4xx_spi.h**
源文件：**stm32f4xx_spi.c**

2、SPI使能函数

```
void  SPI_Cmd(SPI_TypeDef*  SPIx, FunctionalState  NewState);
```

3、SPI发送数据函数

```
void  SPI_I2S_SendData(SPI_TypeDef*  SPIx, uint16_t  Data);
```

4、SPI接收数据函数

```
uint16_t  SPI_I2S_ReceiveData(SPI_TypeDef*  SPIx);
```

5、SPI检测状态标志函数

```
FlagStatus  SPI_I2S_GetFlagStatus(SPI_TypeDef*  SPIx, uint16_t
SPI_I2S_FLAG);
```