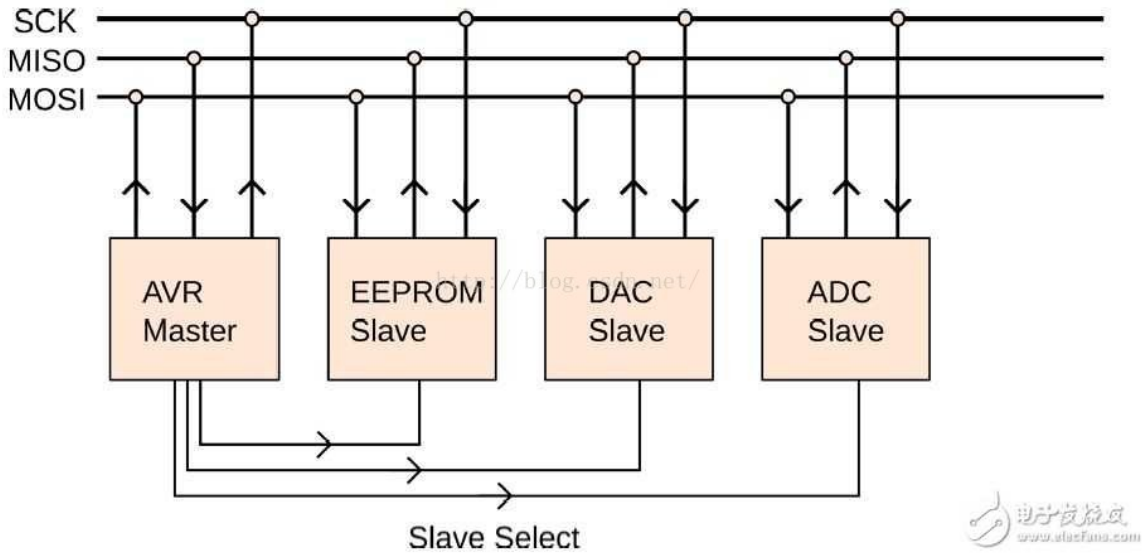
SPI

参考资料

```
https://blog.csdn.net/qq_42282258/article/details/81436882
https://blog.csdn.net/bleauchat/article/details/84821031
```

百度百科：SPI是串行外设接口（Serial Peripheral Interface）的缩写，是一种高速的，全双工，同步的[通信总线，并且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AT91RM9200。

硬件连接



SCK：时钟信号，由主设备产生

MISO (Master Input Slave Output)：主机输入从机输出数据线

MOSI (Master Output Slave Input)：主机输出从机输入数据线

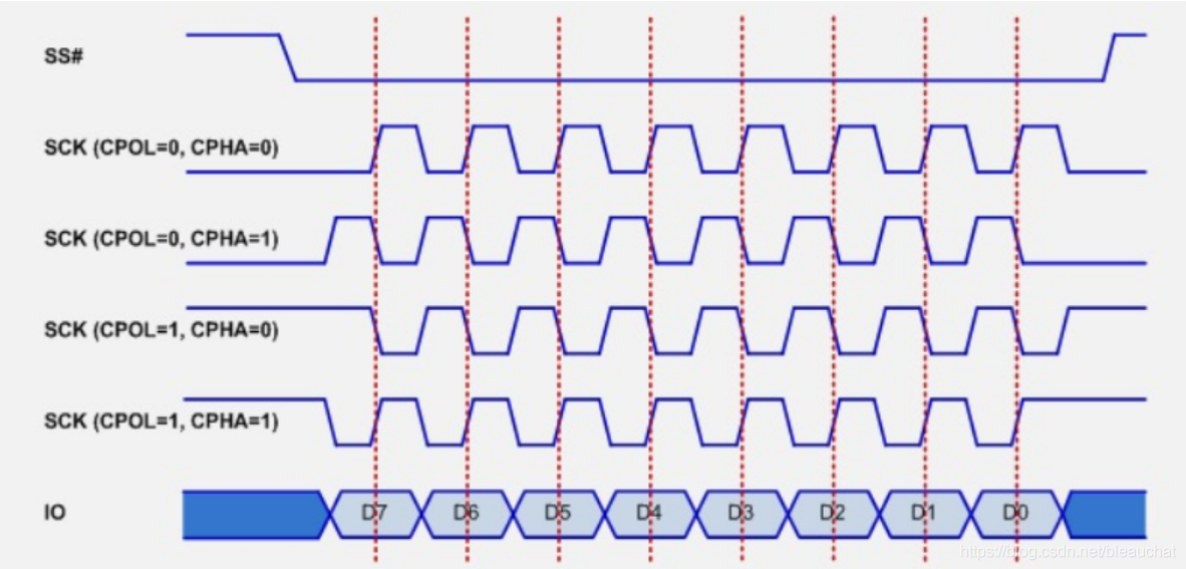
CS(Chip Select)：对应图中的Slave Select 从设备使能信号，由主设备控制

特点

1. 采用**主-从模式(**Master-Slave) 的控制方式
2. 采用**同步方式**(Synchronous)传输数据

传输模式

SPI总线传输一共有4种模式，这4种模式分别由**时钟极性**(CPOL，Clock Polarity)和**时钟相位**(CPHA，Clock Phase)来定义，其中CPOL参数规定了SCK时钟信号空闲状态的电平，CPHA规定了数据是在SCK时钟的第一沿被采样还是第二沿被采样。这四种模式的时序图如下图



模式0：CPOL= 0，CPHA=0。SCK串行时钟线空闲是为低电平，数据在SCK时钟的上升沿被采样，数据在SCK时钟的下降沿切换

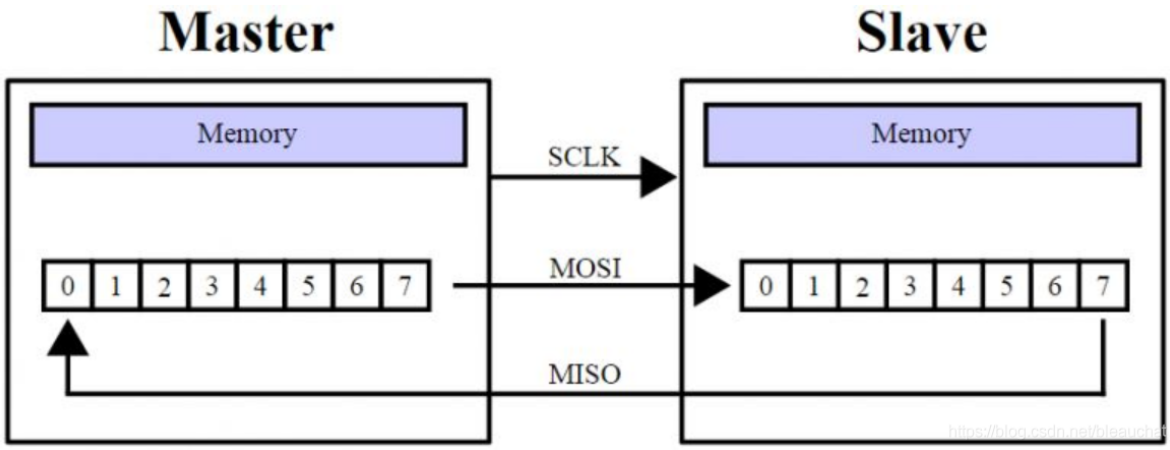
模式1：CPOL= 0，CPHA=1。SCK串行时钟线空闲是为低电平，数据在SCK时钟的下降沿被采样，数据在SCK时钟的上升沿切换

模式2：CPOL= 1，CPHA=0。SCK串行时钟线空闲是为高电平，数据在SCK时钟的下降沿被采样，数据在SCK时钟的上升沿切换

模式3：CPOL= 1，CPHA=1。SCK串行时钟线空闲是为高电平，数据在SCK时钟的上升沿被采样，数据在SCK时钟的下降沿切换

数据交换

SPI 设备间的数据传输之所以又被称为数据交换, 是因为 SPI 协议规定一个 SPI 设备不能在数据通信过程中仅仅只充当一个 "发送者(Transmitter)" 或者 "接收者(Receiver)". **在每个 Clock 周期内, SPI 设备都会发送并接收一个 bit 大小的数据, 相当于该设备有一个 bit 大小的数据被交换了.**

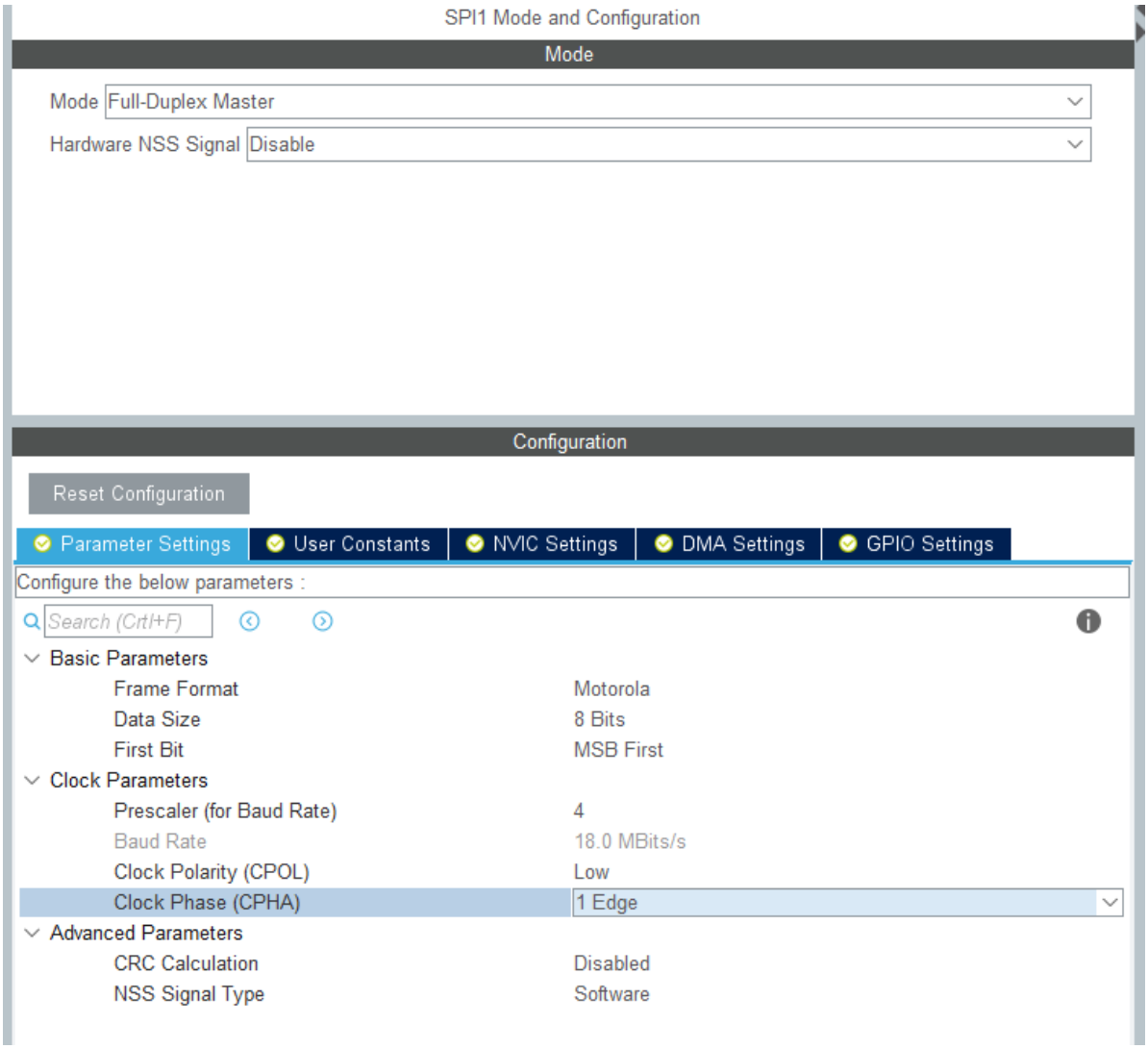


一个 Slave 设备要想能够接收到 Master 发过来的控制信号, 必须在此之前能够被 Master 设备进行访问(Access). 所以, Master 设备必须首先通过 SS/CS pin 对 Slave 设备进行片选, 把想要访问的 Slave 设备选上.

在数据传输的过程中，每次接收到的数据必须在下一次数据传输之前被采样. 如果之前接收到的数据没有被读取, 那么这些已经接收完成的数据将有可能会被丢弃，导致 SPI 物理模块最终失效. 因此, 在程序中一般都会在 SPI 传输完数据后, 去读取 SPI 设备里的数据, 即使这些数据(Dummy Data)在我们的程序里是无用的.

Cube配置

通常的，一个SPI通信的元件/设备可能除了SPI四根线以外还有其他控制线，我们更倾向于把CS和这些其他控制线都提供GPIO_Output模式控制，这样会比较方便



SCK时钟信号由波特率发生器产生和其速率和配置的波特率有关

CPHA中 1Edge和2Edge 就分别对应前文中讲的 CPHA=0和1

HAL库函数

```
HAL_StatusTypeDef HAL_SPI_Transmit(SPI_HandleTypeDef *hspi, uint8_t *pData,
uint16_t Size, uint32_t Timeout);
HAL_StatusTypeDef HAL_SPI_Receive(SPI_HandleTypeDef *hspi, uint8_t *pData,
uint16_t Size, uint32_t Timeout);
HAL_StatusTypeDef HAL_SPI_TransmitReceive(SPI_HandleTypeDef *hspi, uint8_t
*pTxData, uint8_t *pRxData, uint16_t Size,uint32_t Timeout);
HAL_StatusTypeDef HAL_SPI_Transmit_IT(SPI_HandleTypeDef *hspi, uint8_t *pData,
uint16_t Size);
HAL_StatusTypeDef HAL_SPI_Receive_IT(SPI_HandleTypeDef *hspi, uint8_t *pData,
uint16_t Size);
HAL_StatusTypeDef HAL_SPI_TransmitReceive_IT(SPI_HandleTypeDef *hspi, uint8_t
*pTxData, uint8_t *pRxData,
                                             uint16_t Size);
HAL_StatusTypeDef HAL_SPI_Transmit_DMA(SPI_HandleTypeDef *hspi, uint8_t *pData,
uint16_t Size);
HAL_StatusTypeDef HAL_SPI_Receive_DMA(SPI_HandleTypeDef *hspi, uint8_t *pData,
uint16_t Size);
```

```
HAL_StatusTypeDef HAL_SPI_TransmitReceive_DMA(SPI_HandleTypeDef *hspi, uint8_t
*pTxData, uint8_t *pRxData,
                                              uint16_t Size);
HAL_StatusTypeDef HAL_SPI_DMAPause(SPI_HandleTypeDef *hspi);
HAL_StatusTypeDef HAL_SPI_DMAResume(SPI_HandleTypeDef *hspi);
HAL_StatusTypeDef HAL_SPI_DMAStop(SPI_HandleTypeDef *hspi);
/* Transfer Abort functions */
HAL_StatusTypeDef HAL_SPI_Abort(SPI_HandleTypeDef *hspi);
HAL_StatusTypeDef HAL_SPI_Abort_IT(SPI_HandleTypeDef *hspi);

void HAL_SPI_IRQHandler(SPI_HandleTypeDef *hspi);
void HAL_SPI_TxCpltCallback(SPI_HandleTypeDef *hspi);
void HAL_SPI_RxCpltCallback(SPI_HandleTypeDef *hspi);
void HAL_SPI_TxRxCpltCallback(SPI_HandleTypeDef *hspi);
void HAL_SPI_TxHalfCpltCallback(SPI_HandleTypeDef *hspi);
void HAL_SPI_RxHalfCpltCallback(SPI_HandleTypeDef *hspi);
void HAL_SPI_TxRxHalfCpltCallback(SPI_HandleTypeDef *hspi);
void HAL_SPI_ErrorCallback(SPI_HandleTypeDef *hspi);
void HAL_SPI_AbortCpltCallback(SPI_HandleTypeDef *hspi);
```

针对 `HAL_SPI_TransmitReceive` 这一函数 st论坛上专门有一个帖子说明其工作机制

https://community.st.com/s/question/0D50X0000AAGTbJSQX/what-is-halspitransmitreceive-purpose-and-how-it-works

Homework

- 看完 `HAL_SPI_TransmitReceive` 的感受 如何去使用和定义size大小