**三大通讯-------SPI通讯**

参考《野火》第25章 SPI—读写串行FLASH

1. 内存梳理

IIC读写EEPROM,SPI读写FLASH(FLASH是ROM的一种)。ROM的特点是体积小，掉电数据不丢失，速度慢。RAM体积大，掉电丢失，速度快。

**EEPROM(24C02) ：256byte**

**FLASH(W25Q128) ：16Mbyte**

FLASH内存大，价格低廉，比EEPROM使用更广

1. **基本概念**
2. 并不是所有FLASH都用SPI通讯，有的也用FSMC并行通信。FLASH有很多种接口,SPI是一种通讯协议。
3. STM32有硬件SPI,没有硬件IIC那么水，可以直接用。
4. **SPI概念**

SPI (Serial Peripheral Interface)，即串行外围设备接口，是一种**高速全双(同时收发)**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。

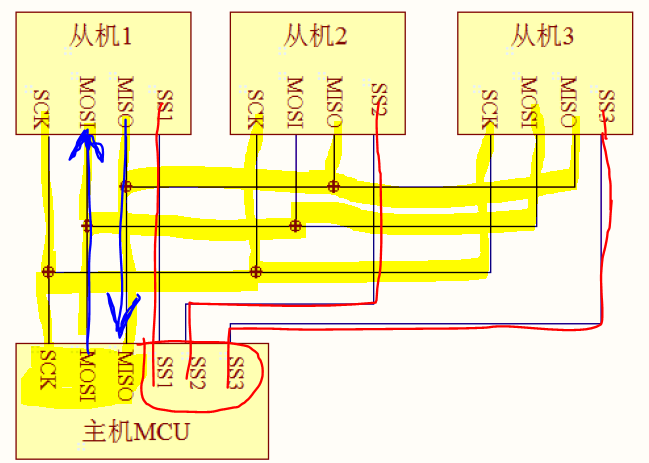
IIC由于是半双工，不能同时收发。所以一些FPGA等速率要求高的就用SPI通信。

IIC速度：10-400Khz

SPI速度：最高达45Mhz(32最高32M)

4.

1. **通讯协议**
2. **物理连线如图：**



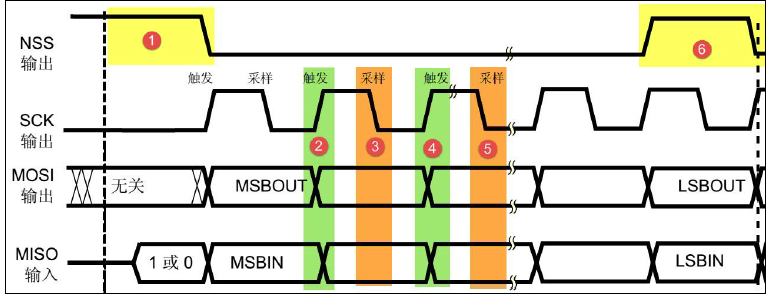
1. **红色部分SSx**：就是片选信号线，也称为NSS,CS。随着外接芯片越多，SSx越多。IIC是通过发送地址+应答来控制向哪个芯片发送数据，而SPI是直接让MCU对SSx使能来控制选择向哪个芯片发送数据。当SSx=0时表明芯片被选中开始传输，**SSx=1表示结束**信号。再通过**黄色部分传输数据**
2. **黄色部分：SCK,MOSI,MISO**三根线用来传输数据。所有从芯片都公共接上这三根线。
3. **SCK**:类似于IIC的SCL,时钟同步线，用于标记有效信号无效信号等。SCK最大=fpclkx/2。(fpclkx 即 APBx的时钟，32有多个SPI,不同SPI挂载在不同时钟上)。
4. **MOSI(Master Output, Slave Input):**信号输出(从MCU角度说)
5. **MISO(Master Input, Slave Output):**信号输入(从MCU角度说)

SCK一个时钟，能发送2bit数据(收发各1bit) (IIC只能1位)

1. **通讯时序**

**以MCU的视角来传输**

1. **数据输出**



* 1. NSS:1-🡪0 片选中从设备，代表通讯开始。0-🡪1停止
  2. SCK:时钟信号，在时钟的上升沿触发，下降沿采样。**触发代表了数据的变化**，采样则是在数据稳定处。
  3. MOSI/MISO可同时进行。
  4. 数据的传输时间=SCK一个周期。触发和采样都是在数据前半个周期完成的。所以SCK最大=fpclkx/2(总之SCK周期<数据传输周期)
  5. MSB**高位字节先行**(配置要注意查询从设备的MSB/LSB)

1. **时钟极性(CPOL/CPHA模式)**

上面讲述的图 25-2 中的时序只是SPI 中的其中一种通讯模式，SPI 一共有**四种通讯模式**，它们的主要区别是**总线空闲时SCK 的时钟状态**以及**数据采样时刻**。为方便说明，在此引入**“时钟极性CPOL(Clock Polarity)”和“时钟相位CPHA(Clock Phase)”**的概念.

这两个哥们可以合力配置SPI的通讯样式

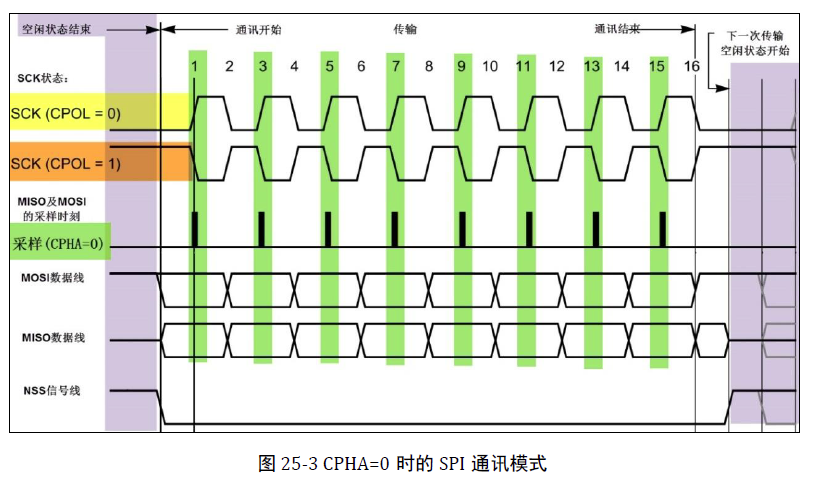
“时钟极性CPOL”：SPI开始前NSS=1时SCK电平 2种

“时钟相位CPHA”：在采样的时刻SCK时钟线的奇数/偶数边沿采样

因此一共4种

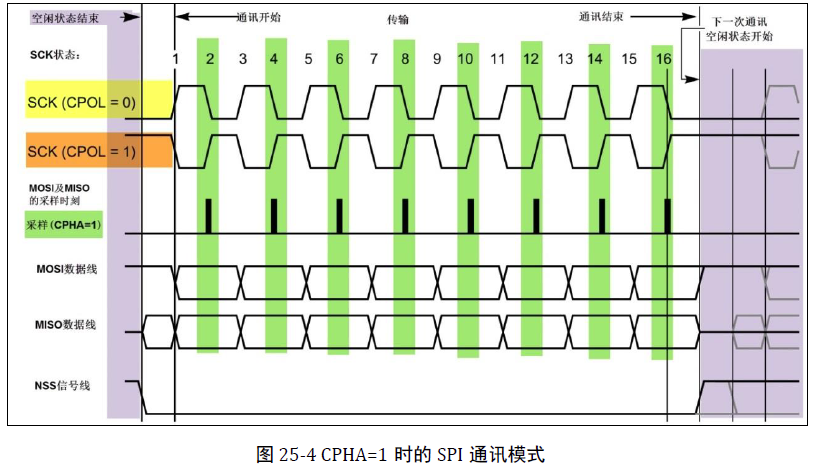
* 1. CPHA=0 **奇数边沿**采样下 开始前SCK的两种电平(0/1)

思考：为什么用奇数/偶数边沿，为啥不用上升/下降沿描述？



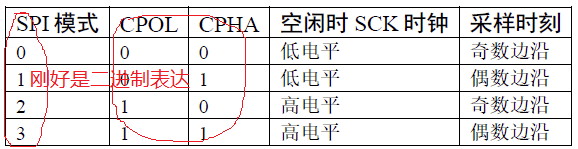
* 1. CPHA=1 **偶数边沿采样**下 开始前SCK的两种电平(0/1)

采样为下降沿则触发肯定为上升沿(触发暂不考虑)



* 1. **SPI四种模式**

CPOL:2 CPAJ:2 2\*2=4种



思考：为什么用奇数/偶数边沿，为啥不用上升/下降沿描述？

从SPI的通讯时序图可知：

CPHA=0时有两种情况，SCK=0时是上降沿采样，SCK=1时是下降沿采样。但他们都是偶数的跳变时采样数据。

CPHA=1时有两种情况，SCK=0时是下降沿采样，SCK=1时是上降沿采样。但他们都是奇数的跳变时采样数据。

所以**只能按奇数/偶数采样来说**

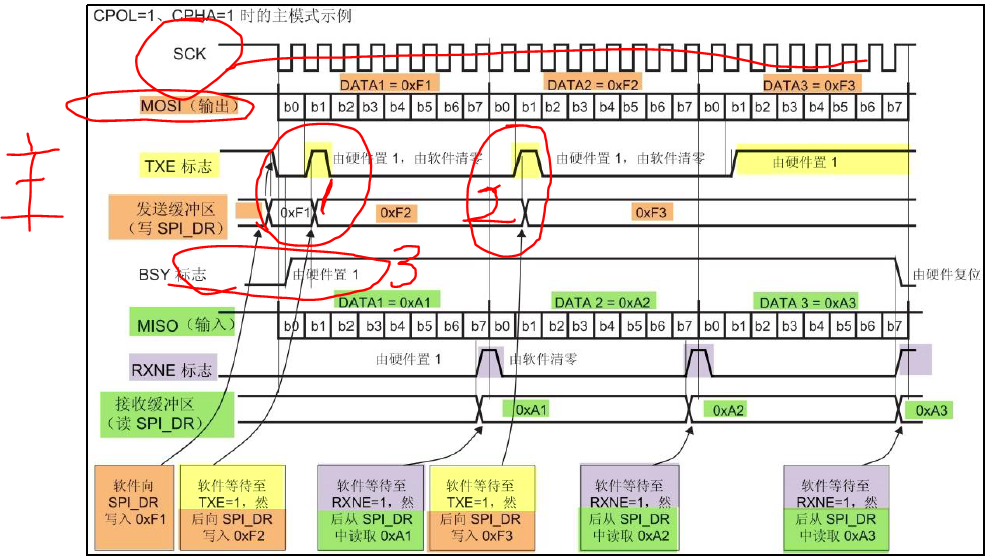
**主设备(MCU)和从设备的SPI模式要配置的一致**

1. **SPI通讯过程**

STM32 使用SPI 外设通讯时，在通讯的不同阶段它会对“状态寄存器SR”的不同数据位写入参数，我们通过读取这些寄存器标志来了解通讯状态。

1. **MUC主模式全双工的发送过程。**

此模式下发送接收同时进行，不管发送/接收有没有数据都同时进行



TXE：发送缓冲为空 (Transmit buffer empty) 0：发送缓冲非空；

1：发送缓冲为空。

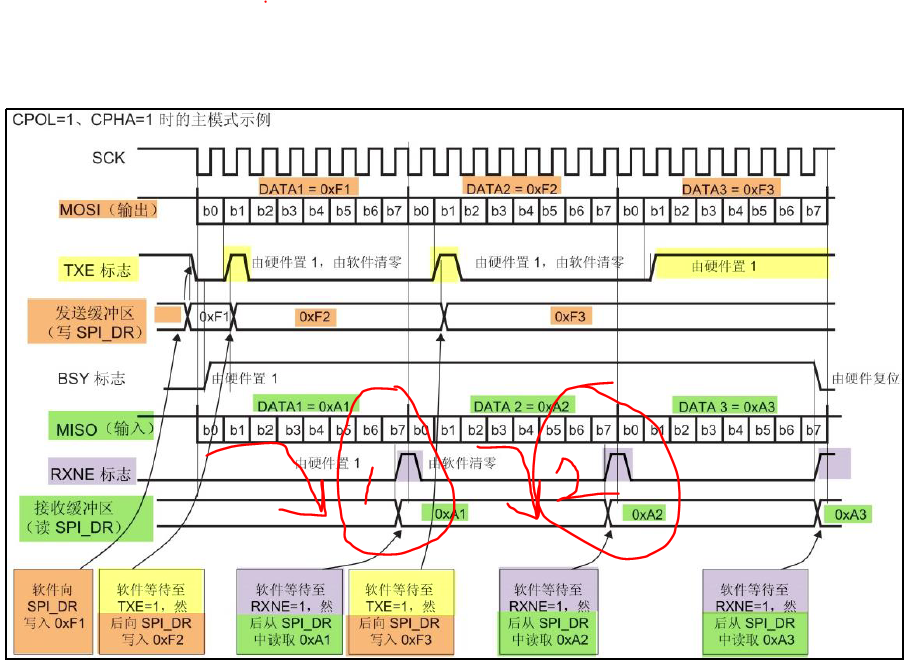
* 1. 当TXE=0.数据0xF1先不被写入SPI\_DR，当TXE有上升沿=1(黄色部分)，将0xF1写入SPI\_DR。此时标志位TXE=0，不能有新的数据写入SPI\_DR。写入SPI\_DR的数据，会被发送的移位寄存器，然后根据SCK的时钟逐位从MOSI发送出去。
  2. 当0xF1逐位发送完，SPI\_DR为空TXE=1，下一个数据0xF2再写入，再重复以上过程
  3. 此时开始通讯，NCSS拉低(图上没有)，BUSY位置1表示忙碌。方式其他从设备打扰

1. **MUC主模式全双工的接收过程。**

此模式下发送接收同时进行，不管发送/接收有没有数据都同时进行

注意一个方向问题，接收是先MISO再SPI\_DR

发送时限SPI\_DR再MOSI



RXNE：接收缓冲非空 (Receive buffer not empty)

0：接收缓冲为空；

1：接收缓冲非空。

* 1. 在MISO未接收到数据时RXNE=0。MISO接收到数据 0xA1，通过移位寄存器发送到SPI\_DR里。此时RXNE=1，判断此标志位后，就从SPI\_DR里读取数据。
  2. 读取后RXNE=0，当0xA2移位发送到SPI\_DR里。此时再次RXNE=1，我们再从SPI\_DR里开始读取数据。

1. **注意**
   1. MOSI/MISO先传输b0还是b8(LSB还是MSB)要根据从设备要求进行设置
   2. 此模式下发送接收同时进行，不管发送/接收有没有数据都同时进行。因此在**读取数据时，我们必须要同时向发送缓存区SPI\_DR写入数据，这样SCK才会工作**，SCK工作从设备才会传输数据，才能读取从设备数据。
   3. 注意一个**方向问题**：接收是先MISO再SPI\_DR

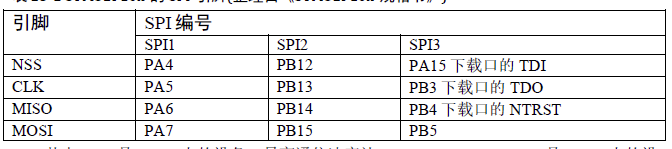
发送时先SPI\_DR再MOSI

* 1. TXE RXNE ERR这些位的变化还能**产生中断**，从而对数据进行处理

1. **STM32的SPI的硬件逻辑**

**参考《官方手册》23章**

1. **通讯引脚**



1. **SPI 1/2/3**

SPI 1高速 72/2M (这是因为sck寄存器最低就是2分频)

SPI 2/3低速 36/2M

1. **NSS**

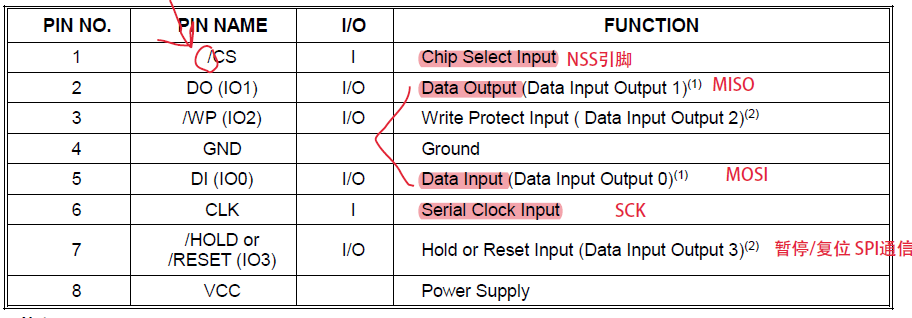
NSS不用寄存器控制，**直接使用一个GPIO模拟控制(**类似IIC模拟)，开始时拉低，结束拉高即可。其他的引脚SCK,MOSI,MISO用硬件逻辑来控制。

“本人板子NSS等各个接口是都按SPI2口,没按上述方法。但实际要用GPIO控制NSS(官方推荐)”

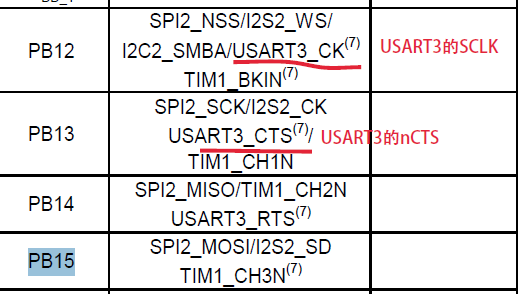
1. **CR1 CR2 SR DR寄存器的使用**

**看手册**

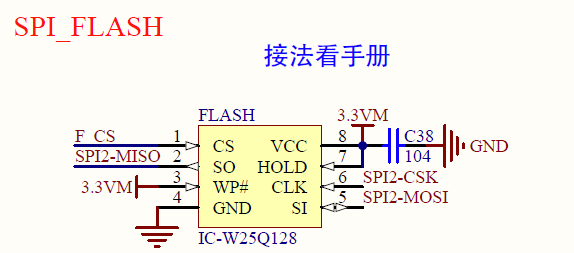
1. **Flash接线图**
   1. FLASH引脚



* 1. SPI2引脚



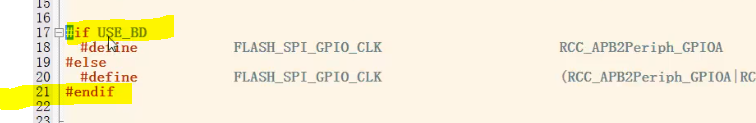
* 1. FLASH接线图

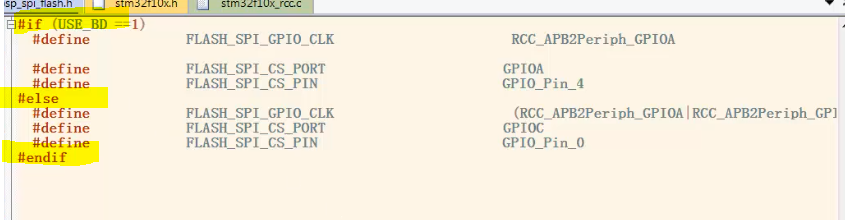


* 1. **强调两点**
     1. FLASH的NSS虽然连接的SPI2\_NSS,但并不使用硬件SPI，而是**做软件引脚设置使用**。把SPI2\_NSS这个GPIO\_PB12当作IO口来使用。这样硬件/软件随意切换。当然NSS也可以随便选一个GPIO
     2. 注意FLASH的**输入接MCU的输出**

FLASH的**输出接MCU的输入**

1. **编程(重点写一个兼容性比较好的 兼容性！！！！！！)**
   * 1. 野火的兼容性是真的牛逼，要学习野火写程序的方法，方便自己以后改程序
   1. **定义一个开关 再宏定义**

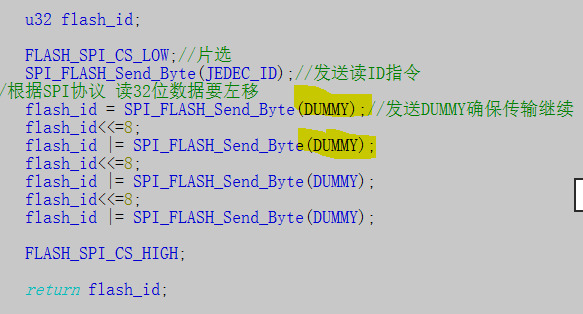




* + 1. 在程序有雏形，**能测试的时候立马测试下，看是否能跑通**。不然后面写的越写越找不出错误，越写越错！！！！！

每到一阶段，每写一个函数，能测试就测试下！！！！

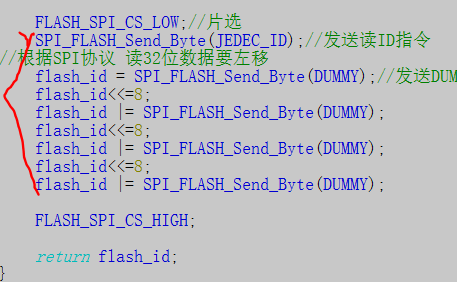
* + 1. 关于DUMMY字节



* + 1. SPI设置的是 SPI\_DataSize\_8b 一次传8位.而根据W25Q128手册传回来的字节有长有短,要做相应处理。读ID要传回3byte的数据，所以首先要定义一个>3byte的数据存放传回来的值

② 因为一次通讯只传8bit，所以要通讯3次才能把数据传完。但想穿必须要先发，这样才能打开SLK时钟。因此**定义了一个DUMMY,来打开通讯的同时又不影响上次命令**

1. 时序问题



* 1. SPI\_FLASH\_Send\_Byte(JEDEC\_ID) 此时发送指令，同时传回的数据为空
  2. flash\_id = SPI\_FLASH\_Send\_Byte(DUMMY); 此时上次发送指令生效，发送DUMMY提高SLK,让其继续SPI通讯，然后同时**传回第一个字节**
  3. flash\_id |= SPI\_FLASH\_Send\_Byte(DUMMY); 发送DUMMY提高SLK,让其继续SPI通讯，然后同时**传回第二个字节**

**结论：**先发送指令，从第**二次通讯**开始一直发送DUMMY,**才开始接收传回的数据**

1. **debug**

5.1 #define FLASH\_ERROR(fmt,arg...) printf("<<-FLASH-ERROR->> "fmt"\n",##arg)

调用：

FLASH\_ERROR("SPI 等待超时!errorCode = %d",68);

结果为：

<<-FLASH-ERROR->> SPI 等待超时!errorCode = 68

**解释：**

printf("<<-FLASH-ERROR->> "fmt"\n",##arg)

* + - 1. 表示打印<<-FLASH-ERROR->>
      2. "fmt"表示调用FLASH\_ERROR时**""里**的东西("fmt"不固定就是一串英文字母而已)
      3. ##arg语法表示可以在<<-FLASH-ERROR->>后续东西(arg不固定，重要的是##)
      4. printf("<<-FLASH-ERROR->> "fmt"\n",##arg)其实是

printf("<<-FLASH-ERROR->> **"fmt"** \n ",##arg)

5.2 此外还有一些ANSI C标准中有几个标准预定义宏（也是常用的）：

\_\_LINE\_\_：在源代码中插入当前源代码行号；

\_\_FILE\_\_：在源文件中插入当前源文件名；

\_\_DATE\_\_：在源文件中插入当前的编译日期

\_\_TIME\_\_：在源文件中插入当前编译时间；

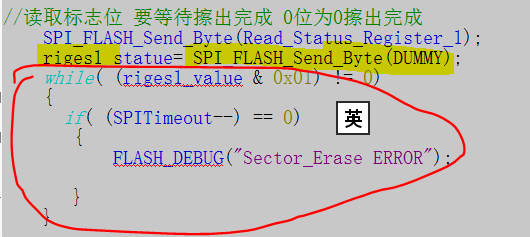
\_\_STDC\_\_：当要求程序严格遵循ANSI C标准时该标识被赋值为1；

\_\_cplusplus：当编写C++程序时该标识符被定义。

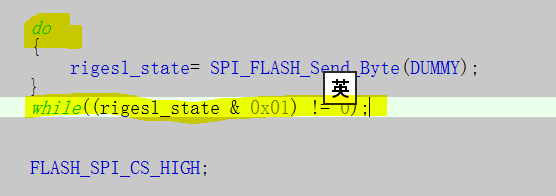
举例：

#define EEPROM\_DEBUG(message,agr...) printf("\r\nDebug: file:[%s] line:[%d] Message:"message" \r\n",\_\_FILE\_\_, \_\_LINE\_\_, ##agr)

1. **为啥我的等待擦出的函数不行**

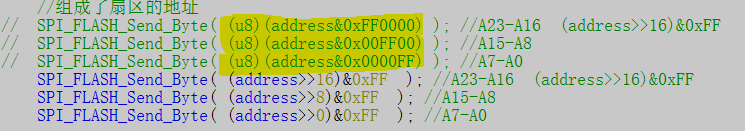


这牵扯到通讯的准则了，要读取到擦出结束标志位才能执行下一步。但是此函数其实只进行了一次读，所以就算擦出结束了。但你只读了一次，所以也读不到。应该是用while()循环等待读的结果，**读不到就一直读**。应该如下图：



do{}while()语句，读不到结果就一直读

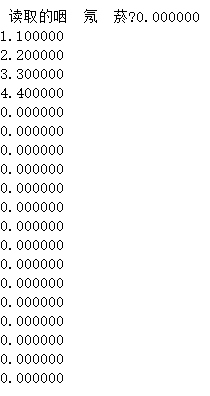
1. **截取函数位**



* 1. 假设address=0x12FF0675.：0x12FF0675&0xFF0000=0x120000
  2. 再强制转换成u8： (u8) 0x120000=0x00，显然是错误的，我们需要的是0x12.
  3. 所以应该是 0x120000>>16=0x12 .**所以应该是**(address&0xFF0000)>>16

**或**(address&>>16) 0xFF

1. **为什么小数读出来后几位全为0**



问题出在这里，Write\_More\_256Byte\_YeHuo();一次写入8位，但你定义的小数float是4byte=32bit。所以写的位数错的，只能写20个8bit，即只能写5个小树。所以后面的自然出不来。

建议黄色部分改成sizeof(xxx) 或 数组数\*4

