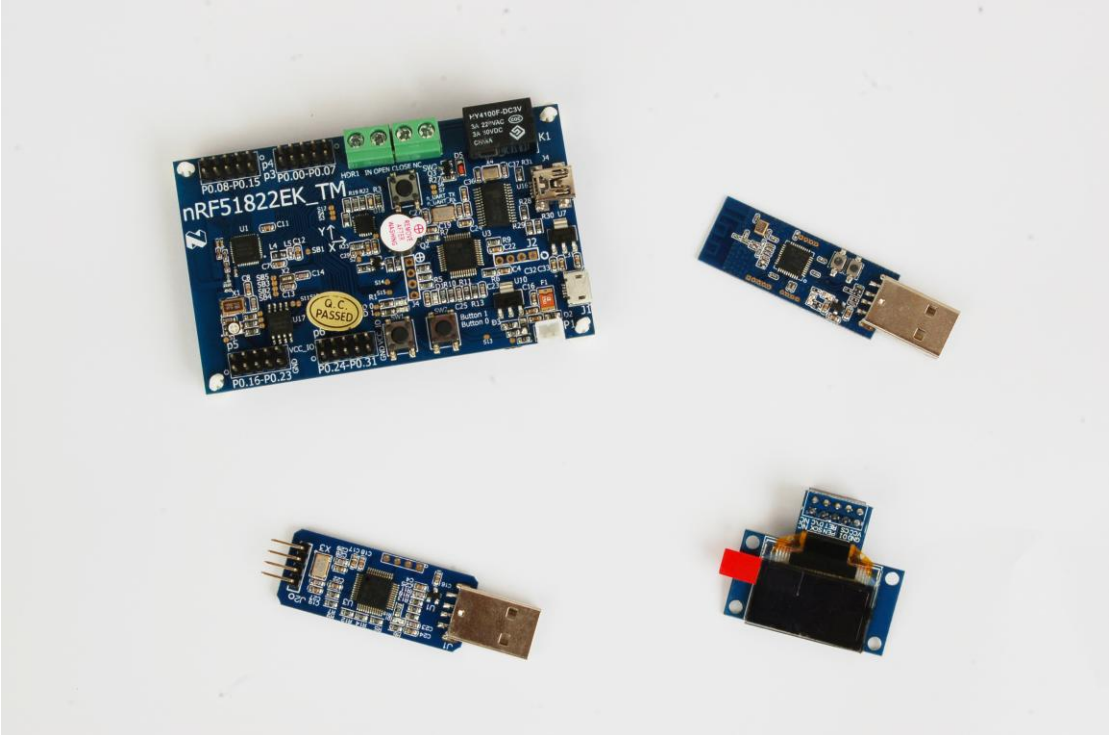
讯联电子nRF51822蓝牙4.0开发实战

SPI(主)

V:1.0

作者：不离不弃　qq 574912883

申明：本教程版权归讯联电子所有。本教程仅供内部客户交流之用。如需引用，请注明出处。由于工程师水平有限，文档难免有所疏漏和错误，由此造成的损失，讯联电子不承担任何责任。



关于 SPI 总线的介绍这里就不细说了，网上有很多介绍 SPI 总线时序的。

SPI 总线的本质就是一个环形总线结构，在时钟驱动下两个双向移位寄存器进行数据交换。
所以 SPI 总线的特色就是：传输一字节数据的同时也会接收到一字节数据。支持 SPI 操作的芯片通常都会有一个 CS 引脚作为片选信号，所以总线上可以挂多个支持 SPI 操作的芯片，每次想对哪个操作就直接使能那个芯片就可以了。

对比于 IIC 总线，IIC 总是在一个时钟周期内传输一个 bit， IIC 总线总是在每个时钟周期的高电平部分采样数据。
而 SPI 的特点是在时钟周期的跳变沿时采样数据或传输数据。一个时钟周期有两个跳变沿，一个上升沿一个下降沿。于是 SPI 总线的传输的特性就是在一个时钟周期的上升沿发送自己的一个 bit 数据，同时在该时钟周期的下降沿接收对端设备发送过来的一 bit 数据。或者是上升沿接收数据，下降沿发送数据。这都是可以设置的。

所以使用 SPI 总线时，当你要传输一个数据时总是同时会收到对端发来的一个数据，但是你没想要这个数据啊，很简单直接忽略就行了。
同理当你想读取对端的一个数据时，也随便发一个数据过去就行了。

51822 发送数据是通过将数据写入 TXD 寄存器，因为每发送一个数据的同时会收到一个数据，该数据会放在 RXD 寄存器中以供读取使用。每当发送和接收到一个字节后，就会产生 READY 事件，以方便告知我们传输完成。所以 SPI 基本的一个字节的传输和接收操作通过如下函数实现

```
//SPI 传输的基本函数，该函数传输一个字节给对方同时取得对方发送过来的字
//节
uint8_t spi_transfer(uint8_t data){
    uint8_t ret;
    NRF_SPI0->TXD = data;
    while ( NRF_SPI0->EVENTS_READY == 0 ); //等待传输结束
    NRF_SPI0->EVENTS_READY = 0;
    ret = NRF_SPI0->RXD;
    return ret;
}
```

下面介绍对开发板上的 flash GD25Q128B 进行数据的写入和读取操作。
上面提到过，SPI 总线可以设置在一个 上升沿接收数据，下降沿发送数据，或者相反。 那么我们就需要看一下这个板子上的这个 flash 芯片对这个有没有要求，如果没要求我们就可以随便设置。查看 GD25Q128B 手册知道该芯片要求每个字节的最高 bit 先传输，同时要求在时钟的上升沿会获取总线上的数据。这里是针对这个 flash 芯片来说的，那么对于 51822，就需要在上升沿把数据发送出去。

GigaDevice Dual and Quad Serial Flash GD25Q128B

7. COMMANDS DESCRIPTION

All commands, addresses and data are shifted in and out of the device, beginning with the most significant bit on the first rising edge of SCLK after CS# is driven low. Then, the one-byte command code must be shifted in to the device, most significant bit first on SI, each bit being latched on the rising edges of SCLK.

See Table2, every command sequence starts with a one-byte command code. Depending on the command, this might be followed by address bytes, or by data bytes, or by both or none. CS# must be driven high after the last bit of the command sequence has been shifted in. For the command of Read, Fast Read, Read Status Register or Release from Deep Power-Down, and Read Device ID, the shifted-in command sequence is followed by a data-out sequence. CS# can be driven high after any bit of the data-out sequence is being shifted out.

For the command of Page Program, Sector Erase, Block Erase, Chip Erase, Write Status Register, Write Enable, Write Disable or Deep Power-Down command, CS# must be driven high exactly at a byte boundary, otherwise the command is rejected, and is not executed. That is CS# must driven high when the number of clock pulses after CS# being driven low is an exact multiple of eight. For Page Program, if at any time the input byte is not a full byte, nothing will happen and WEL will not be reset.

Table2. Commands

Command Name	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	n-Bytes
Write Enable	06H						

Table 234: CONFIG

Bit number				31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4
Id				
Reset				0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Id	RW	Field	Value Id	Value	Description
A	RW	ORDER			Bit order
			MsbFirst	0	Most significant bit shifted out first
			LsbFirst	1	Least significant bit shifted out first
B	RW	CPHA			Serial clock (SCK) phase
			Leading	0	Sample on leading edge of clock, shift serial data on trailing edge
			Trailing	1	Sample on trailing edge of clock, shift serial data on leading edge
C	RW	CPOL			Serial clock (SCK) polarity
			ActiveHigh	0	Active high
			ActiveLow	1	Active low

查看 51822 手册的 SPI 部分，找到 CONFIG 寄存器，根据上图的要求，则需要设置第一 bit 位 0， 关于时钟极性(总线空闲时是保持低电平还是高电平)flash 手册中没要求所以可以随便设置，所以我们只要满足 flash 芯片可以在时钟周期的上升沿能收到数据就行了。也就是 51822 在上升沿将数据发送出去。

所以 如果设置 CPOL 为 0，即空闲时保持高电平，那么当开始工作时，每个时钟

的第一个沿就是下降沿，第二个沿是上升沿。根据下图说明这时候我们就要设置 CPHA 为 0，即让 51822 在时钟周期第二个沿 将数据发送出去。
同理，如果 CPOL 位 1，那么开始工作时第一个沿就是上升沿，第二个沿就是下降沿，于是就要设置 51822 在第一个沿将数据发送出去也就是设置 CPHA 为 1.

//flash 芯片要求在 MSB 先传输，并且在第一个上升沿采样。所以设置下。
//CPOL 和 CPHA 都设置成 1 也是可以的,这里都设置成 0 了。

```
NRF_SPI0->CONFIG = (0<<0)    //MSB first
                 | (0<< 1)
                 | (0 << 2)
```

基本传输要求满足了后，现在需要操作 flash 的读写，那么就需要看 flash 手册。了解关于读写命令的要求和命令码。查看手册说明，我们需要用到的命令有如下几个读/写命令。Flash 操作都是以页为单位写(该芯片一页 256 字节)，并且写之前需要先擦除，所以还需要用到 sector erase(以 4K 为单位擦除)命令来擦数，block erase 擦除的单位更大,这里用 sector erase 就行了。
读操作直接就可以完成,但是写操作发送后 flash 内被还需要时间去执行,以将数据写入 flash 中。所以我们还需要获取 flash 的执行状态，即当前是否在忙。所以还需要用到 读状态命令
另外，GD25Q128B 这个 flash 芯片每次发送 页写或者 擦除命令之前需要先发送 Write Enable 命令。

Command Name	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	n-Bytes
Write Enable	06H						
Write Disable	04H						
Read Status Register	05H	(S7-S0)					(continuous)
Read Status Register-1	35H	(S15-S8)					(continuous)
Write Status Register	01H	(S7-S0)	(S15-S8)				
Read Data	03H	A23-A16	A15-A8	A7-A0	(D7-D0)	(Next byte)	(continuous)
Fast Read	0BH	A23-A16	A15-A8	A7-A0	dummy	(D7-D0)	(continuous)
Dual Output Fast Read	3BH	A23-A16	A15-A8	A7-A0	dummy	(D7-D0)(1)	(continuous)
Dual I/O Fast Read	BBH	A23-A8(2)	A7-A0 M7-M0(2)	(D7-D0)(1)			(continuous)
Quad Output Fast Read	6BH	A23-A16	A15-A8	A7-A0	dummy	(D7-D0)(3)	(continuous)
Quad I/O Fast Read	EBH	A23-A0 M7-M0(4)	dummy(5)	(D7-D0)(3)			(continuous)
Quad I/O Word Fast Read(7)	E7H	A23-A0 M7-M0(4)	dummy(6)	(D7-D0)(3)			(continuous)
Continuous Read Reset	FFH						
Page Program	02 H	A23-A16	A15-A8	A7-A0	(D7-D0)	Next byte	
Quad Page Program	32H	A23-A16	A15-A8	A7-A0	(D7-D0)(3)		
Sector Erase	20H	A23-A16	A15-A8	A7-A0			
Block Erase(32K)	52H	A23-A16	A15-A8	A7-A0			
Block Erase(64K)	D8H	A23-A16	A15-A8	A7-A0			
Chip Erase	C7/60 H						
Program/Erase Suspend	75H						
Program/Erase Resume	7AH						

下面针对几个用到的命令看下 flash 的手册说明

先看下读状态寄存器 命令，这里我们只需要读取状态寄存器的前 7 位就行了。然后判断 第 0 位的 WIP 是否为 1，为 1 则表示 flash 芯片正在忙(正在写数据到 flash 中或者正在擦除)，否则为不忙，则可以执行 写或擦除命令。

6. STATUS REGISTER

S15	S14	S13	S12	S11	S10	S9	S8
SUS	CMP	Reserved	Reserved	Reserved	LB	QE	SRP1

S7	S6	S5	S4	S3	S2	S1	S0
SRP0	BP4	BP3	BP2	BP1	BP0	WEL	WIP

The status and control bits of the Status Register are as follows:

WIP bit

The Write In Progress (WIP) bit indicates whether the memory is busy in program/erase/write status register progress. When WIP bit sets to 1, means the device is busy in program/erase/write status register progress, when WIP bit sets 0, means the device is not in program/erase/write status register progress.

WEL bit

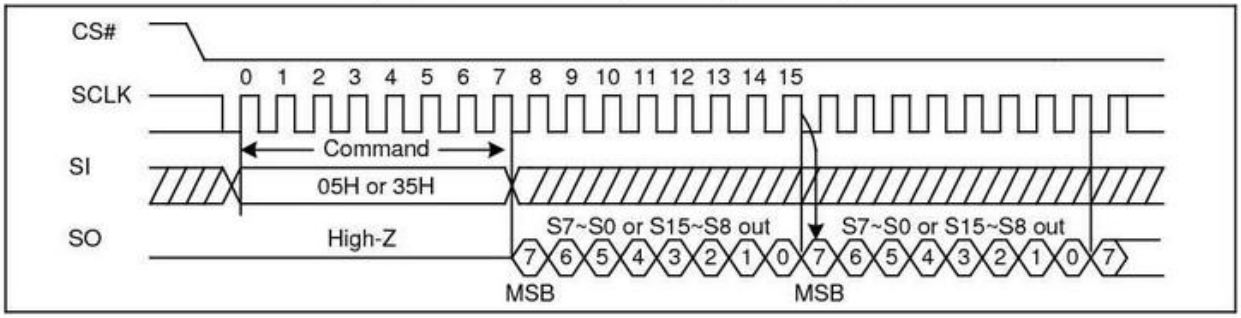
The Write Enable Latch (WEL) bit indicates the status of the internal Write Enable Latch. When set to 1 the internal Write Enable Latch is set, when set to 0 the internal Write Enable Latch is reset and no Write Status Register, Program or Erase command is accepted.

所以读状态寄存器命令我们用 0x05 就行了，因为我们只需要看第 0 位指示的 flash 芯片是否在忙，所以我们只要用 0x05 命令读取 s0-s7 位就行了。

7.3. Read Status Register (RDSR) (05H or 35H)

The Read Status Register (RDSR) command is for reading the Status Register. The Status Register may be read at any time, even while a Program, Erase or Write Status Register cycle is in progress. When one of these cycles is in progress, it is recommended to check the Write In Progress (WIP) bit before sending a new command to the device. It is also possible to read the Status Register continuously. For command code "05H", the SO will output Status Register bits S7~S0. The command code "35H", the SO will output Status Register bits S15~S8.

Figure 4. Read Status Register Sequence Diagram



通过读取状态寄存器，然后判断第 0 位，我们就可以知道设备是否在忙。

因为 基本的 SPI 传输和接收一个字节的函数我们已经实现了。所以等待设备不忙函数实现如下：

```
uint8_t flash_status_read(void){
```

```
    uint8_t ret;

    enable_cs_pin();
    spi_transfer(READ_STATUS);//发命令
    ret = spi_transfer(0xff);//读取状态寄存器的 s0-s7， 写数据随便填的。
    disable_cs_pin();

    return ret;
}

#define WAIT_COUNT   (200000UL)

uint8_t wait_flash_ready(void){
    uint8_t ret;
    uint32_t i ;
    for(i= 0; i<WAIT_COUNT; i++ ){
        ret = flash_status_read();
        if( (ret&0x01) == 0) break;
    }
    if ( i == WAIT_COUNT ) return FAIL;
    return SUCCESS;
}
```

再看下 write enable 命令，由下图可知该命令只需要 使能片选信号 cs（拉低）后，发送一个 0x06 命令码就可以了。之后才可以执行 页写和擦除命令
有了前面的基本 spi 的 写入和读取 一个字节的基本函数后，我们就可以实现如下的 write enable 函数

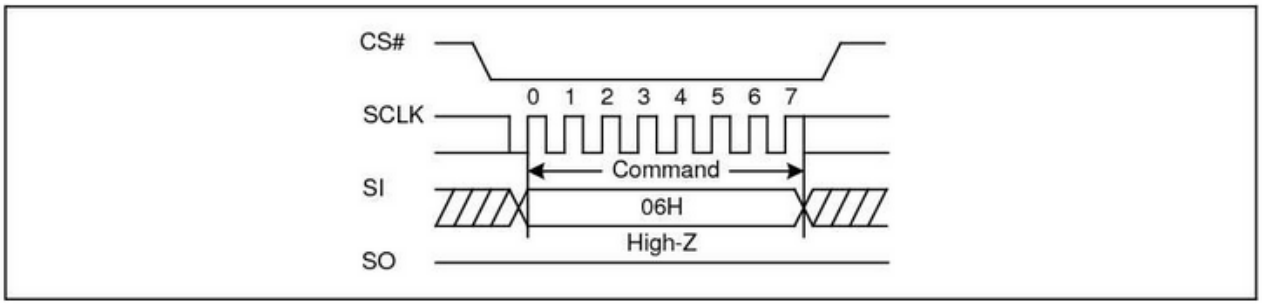
```
uint8_t flash_write_enable(void){
    //需要等待设备不忙
    if(wait_flash_ready() != SUCCESS ) return FAIL;

    enable_cs_pin(); //命令发送前需要先 使能片选信号
    spi_transfer(WRITE_ENABLE_CMD);
```

```
    disable_cs_pin();
    // 等待设备不忙，即该命令再 flash 芯片内部执行好了，然后才能去执行
    // 写/擦除命令
    if(wait_flash_ready() != SUCCESS ) return FAIL;

    return SUCCESS;
}
```

7.1. Write Enable (WREN) (06H)

The Write Enable (WREN) command is for setting the Write Enable Latch (WEL) bit. The Write Enable Latch (WEL) bit must be set prior to every Page Program (PP), Sector Erase (SE), Block Erase (BE), Chip Erase (CE) and Write Status Register (WRSR) command. The Write Enable (WREN) command sequence: CS# goes low → sending the Write Enable command → CS# goes high.

Figure 2. Write Enable Sequence Diagram



再看下读命令：
由下图可知 通过写命令码 0x03 + 3 字节的地址，然后就可以获取指定地址的数据了。获得得的数据量由自己定义。该 flash 芯片每次发送一字节数据后，内部会对读取地址自动加一，所以我们只要一直随便发一个数据给该 flash 芯片，flash 芯片就会一直回复数据，并自动对读取地址加 1.
同样由基本 传输/接收函数实现 读取函数如下：

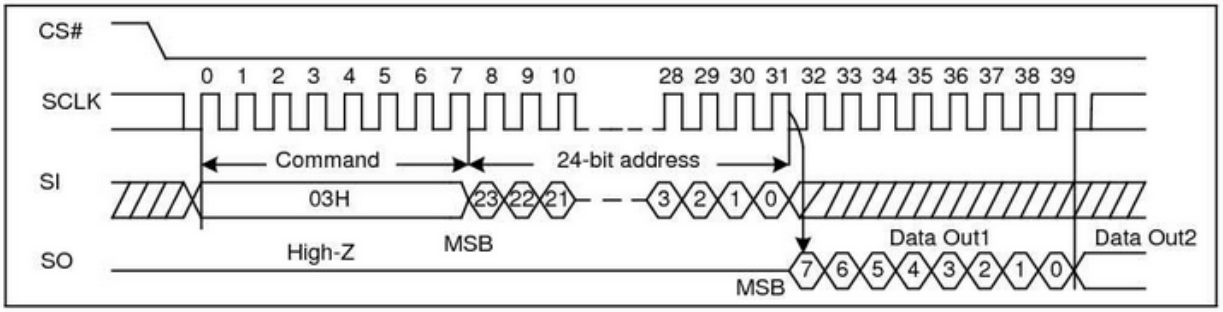
```
uint8_t flash_read(uint32_t addr, uint8_t *buff, uint32_t len){
    uint8_t ret;
    uint8_t i;

    if( NULL == buff ) return FAIL;

    wait_flash_ready();  //等待设备当前是不忙的
    enable_cs_pin();
    spi_transfer(READ_CMD);     //先传命令
    spi_transfer( (addr>>16)&0xff );//依次传输 3 字节地址，高字节先发送
    spi_transfer( (addr>>8)&0xff );
```

```
    spi_transfer( addr&0xff);
    //开始获取读到的数据
    for( i = 0; i < len; i++){
        ret = spi_transfer(0xff);    //这里主要是读数据，所以随便传输一
                                     //个数据给对方
        buff[i] = ret;
    }
    disable_cs_pin();

    return SUCCESS;
}
```

7.5. Read Data Bytes (READ) (03H)

The Read Data Bytes (READ) command is followed by a 3-byte address (A23-A0), each bit being latched-in during the rising edge of SCLK. Then the memory content, at that address, is shifted out on SO, each bit being shifted out, at a Max frequency f_R, during the falling edge of SCLK. The first byte addressed can be at any location. The address is automatically incremented to the next higher address after each byte of data is shifted out. The whole memory can, therefore, be read with a single Read Data Bytes (READ) command. Any Read Data Bytes (READ) command, while an Erase, Program or Write cycle is in progress, is rejected without having any effects on the cycle that is in progress.

Figure 6. Read Data Bytes Sequence Diagram



页写命令：该命令执行前需要先执行了 write enable 命令。
由下面可知 通过写 命令码 0x02+3 字节地址+要写的数据，数据量应该小于等于 256 字节。不然只有最后的 256 字节才会被写入。
函数实现如下：

```
//板子上的 flash 芯片一页是 256 字节，如果写超过 256 字节，只有最后的 256
字节能被写到 flash 中
uint8_t flash_page_write(uint32_t addr, uint8_t *data, uint32_t len){
    uint8_t ret;
    uint8_t i;
    ret = flash_write_enable();     //先执行 write enable
    if(ret!= SUCCESS) return ret;
```

```
    enable_cs_pin();
    spi_transfer(PAGE_PROGRAM);     //先传命令
    spi_transfer( (addr>>16)&0xff );   //依次传输 3 字节地址，高字节先
发送
    spi_transfer( (addr>>8)&0xff );
    spi_transfer( addr&0xff);
    //开始传数据给 flsh 芯片
    for( i = 0; i < len; i++){
        spi_transfer(data[i]);  //这里主要是写数据，所以忽略返回值。
    }
    disable_cs_pin();

    return SUCCESS;

}
```

7.12. Page Program (PP) (02H)

The Page Program (PP) command is for programming the memory. A Write Enable (WREN) command must previously have been executed to set the Write Enable Latch (WEL) bit before sending the Page Program command.

The Page Program (PP) command is entered by driving CS# Low, followed by the command code, three address bytes and at least one data byte on SI. If the 8 least significant address bits (A7-A0) are not all zero, all transmitted data that goes beyond the end of the current page are programmed from the start address of the same page (from the address whose 8 least significant bits (A7-A0) are all zero). CS# must be driven low for the entire duration of the sequence. The Page Program command sequence: CS# goes low → sending Page Program command → 3-byte address on SI → at least 1 byte data on SI → CS# goes high. The command sequence is shown in Figure16. If more than 256 bytes are sent to the device, previously latched data are discarded and the last 256 data bytes are guaranteed to be programmed correctly within the same page. If less than 256 data bytes are sent to device, they are correctly programmed at the requested addresses without having any effects on the other bytes of the same page. CS# must be driven high after the eighth bit of the last data byte has been latched in; otherwise the Page Program (PP) command is not executed.

As soon as CS# is driven high, the self-timed Page Program cycle (whose duration is t_{PP}) is initiated. While the Page Program cycle is in progress, the Status Register may be read to check the value of the Write In Progress (WIP) bit. The Write In Progress (WIP) bit is 1 during the self-timed Page Program cycle, and is 0 when it is completed. At some unspecified time before the cycle is completed, the Write Enable Latch (WEL) bit is reset.

A Page Program (PP) command applied to a page which is protected by the Block Protect (BP4, BP3, BP2, BP1, BP0) is not executed.

最后看一下 擦除命令：该命令执行前需要先执行 write enable 命令。

由下图可知通过写命令码 0x20+3 字节地址就可以了。
所以实现 擦除函数如下:

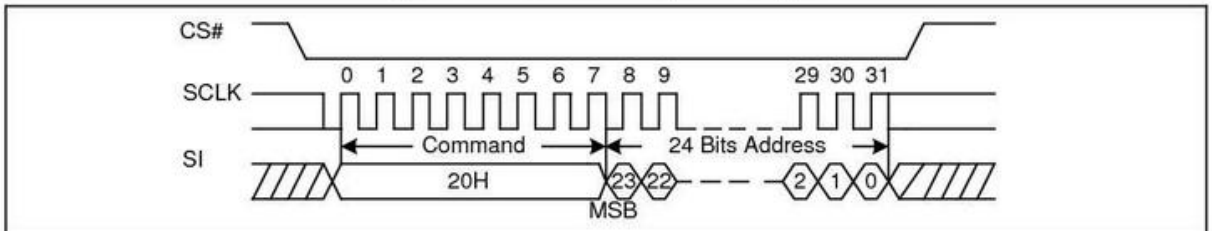
```
uint8_t flash_sector_erase(uint32_t addr){
    uint8_t ret;
    ret = flash_write_enable();     //先写 write enable 命令
    if ( ret != SUCCESS ) return ret;

    enable_cs_pin();
    spi_transfer(SECTOR_ERASE);     //先传命令
    spi_transfer( (addr>>16)&0xff );//依次传输 3 字节地址，高字节先发送
    spi_transfer( (addr>>8)&0xff );
    spi_transfer( addr&0xff);
    disable_cs_pin();

    return SUCCESS;

}
```

7.14. Sector Erase (SE) (20H)

The Sector Erase (SE) command is erased the all data of the chosen sector. A Write Enable (WREN) command must previously have been executed to set the Write Enable Latch (WEL) bit. The Sector Erase (SE) command is entered by driving CS# low, followed by the command code, and 3-address byte on SI. Any address inside the sector is a valid address for the Sector Erase (SE) command. CS# must be driven low for the entire duration of the sequence.

The Sector Erase command sequence: CS# goes low → sending Sector Erase command → 3-byte address on SI → CS# goes high. The command sequence is shown in Figure18. CS# must be driven high after the eighth bit of the last address byte has been latched in; otherwise the Sector Erase (SE) command is not executed. As soon as CS# is driven high, the self-timed Sector Erase cycle (whose duration is t_{SE}) is initiated. While the Sector Erase cycle is in progress, the Status Register may be read to check the value of the Write In Progress (WIP) bit. The Write In Progress (WIP) bit is 1 during the self-timed Sector Erase cycle, and is 0 when it is completed. At some unspecified time before the cycle is completed, the Write Enable Latch (WEL) bit is reset. A Sector Erase (SE) command applied to a sector which is protected by the Block Protect (BP4, BP3, BP2, BP1, BP0) bit (see Table1.0&1.1) is not executed.

Figure 18. Sector Erase Sequence Diagram



各个功能函数都有了就可以操作 flash 了。

下面将各个功能函数和 测试的 main 函数都贴出来。
为了可以通过打印来验证 flash 操作的正确性，可以直接用 sdk 下面的 uart 的 ble demo。虽然该 demo 是跑 ble 的。但是我们直接将 main 函数换掉就行了。用该例子的原因就是为了方便初学者可以使用 Printf 函数来打印信息验证 flash 操作的正确性。

直接打开 sdk 中的 ble_app_uart 例子，然后将下列代码复制到 main 函数的上面，然后将 Main 函数替换成 最下面提供的那个 main 函数就行了。

```
#define PIN_CS              28
#define PIN_MISO        29
#define PIN_MOSI        24
#define PIN_CLK             25


#define WRITE_ENABLE_CMD 0x06
#define READ_CMD        0x03
#define PAGE_PROGRAM    0x02
#define SECTOR_ERASE    0x20
#define READ_STATUS         0x05 //我们读状态寄存器只是为了判断 flash 当前是不是正在忙，所以只需判断 s0 位，所以用 0x05 命令读出 s0-s7 就可以了

#define SUCCESS             0x01
#define FAIL            0x00
void init_spi_master(void){

    //按手册要求设置
    nrf_gpio_cfg_input(PIN_MISO, NRF_GPIO_PIN_NOPULL);
    nrf_gpio_cfg_output(PIN_MOSI);
    nrf_gpio_cfg_output(PIN_CLK);
    nrf_gpio_cfg_output(PIN_CS);
    nrf_gpio_pin_set(PIN_CS);


    //flash 芯片要求在 MSB 先传输，并且在第一个上升沿采样。所以设置下。CPOL 和 CPHA 都设置成 1 也是可以的,这里都设置成 0 了。
    NRF_SPI0->CONFIG = (0<<0)   //MSB first
                     | (0<< 1)
                     | (0 << 2);
    NRF_SPI0->FREQUENCY = 0x10000000;
    NRF_SPI0->PSELSCK = PIN_CLK;
```

```
    NRF_SPI0->PSELMOSI = PIN_MOSI;
    NRF_SPI0->PSELMISO = PIN_MISO;

    NRF_SPI0->ENABLE = 1;

}

//SPI 传输的基本函数，该函数传输一个字节给对方同时取得对方发送过来的字节
uint8_t spi_transfer(uint8_t data){
    uint8_t ret;
    NRF_SPI0->TXD = data;
    while ( NRF_SPI0->EVENTS_READY == 0 );//等待传输结束
    NRF_SPI0->EVENTS_READY = 0;

    ret = NRF_SPI0->RXD;

    return ret;
}
void enable_cs_pin(void){
    nrf_gpio_pin_clear(PIN_CS);
}

void disable_cs_pin(void){
    nrf_gpio_pin_set(PIN_CS);
}

uint8_t flash_status_read(void){
    uint8_t ret;

    enable_cs_pin();
    spi_transfer(READ_STATUS);//发命令
    ret = spi_transfer(0xff);//读取状态寄存器的 s0-s7， 写数据随便填的。
    disable_cs_pin();

    return ret;
}

#define WAIT_COUNT    (200000UL)

uint8_t wait_flash_ready(void){
    uint8_t ret;
    uint32_t i ;
```

```
    for(i= 0; i<WAIT_COUNT; i++ ){
        ret = flash_status_read();
        if( (ret&0x01) == 0) break;
    }
    if ( i == WAIT_COUNT ) return FAIL;

    return SUCCESS;

}

uint8_t flash_write_enable(void){

    if(wait_flash_ready() != SUCCESS ) return FAIL;

    enable_cs_pin();
    spi_transfer(WRITE_ENABLE_CMD);
    disable_cs_pin();
    if(wait_flash_ready() != SUCCESS ) return FAIL;

    return SUCCESS;
}

uint8_t flash_read(uint32_t addr, uint8_t *buff, uint32_t len){
    uint8_t ret;
    uint8_t i;

    if( NULL == buff ) return FAIL;

    wait_flash_ready();
    enable_cs_pin();
    spi_transfer(READ_CMD);      //先传命令
    spi_transfer( (addr>>16)&0xff );   //依次传输 3 字节地址，高字节先
发送
    spi_transfer( (addr>>8)&0xff );
    spi_transfer( addr&0xff);
    //开始获取读到的数据
    for( i = 0; i < len; i++){
        ret = spi_transfer(0xff);   //这里主要是读数据，所以随便传输一
个数据给对方
        buff[i] = ret;
    }
    disable_cs_pin();
```

```
    return SUCCESS;
}

//板子上的 flash 芯片一页是 256 字节，如果写超过 256 字节，只有最后的 256 字节能被写到 flash 中
uint8_t flash_page_write(uint32_t addr, uint8_t *data, uint32_t len){
    uint8_t ret;
    uint8_t i;
    ret = flash_write_enable();
    if(ret!= SUCCESS) return ret;

    enable_cs_pin();
    spi_transfer(PAGE_PROGRAM);     //先传命令
    spi_transfer( (addr>>16)&0xff );   //依次传输 3 字节地址，高字节先发送
    spi_transfer( (addr>>8)&0xff );
    spi_transfer( addr&0xff);
    //开始传数据给 flsh 芯片
    for( i = 0; i < len; i++){
        spi_transfer(data[i]);   //这里主要是写数据，所以忽略返回值。
    }
    disable_cs_pin();

    return SUCCESS;

}
uint8_t flash_sector_erase(uint32_t addr){
    uint8_t ret;
    ret = flash_write_enable();
    if ( ret != SUCCESS ) return ret;

    enable_cs_pin();
    spi_transfer(SECTOR_ERASE);     //先传命令
    spi_transfer( (addr>>16)&0xff );   //依次传输 3 字节地址，高字节先发送
    spi_transfer( (addr>>8)&0xff );
    spi_transfer( addr&0xff);
    disable_cs_pin();

    return SUCCESS;

}
```

```
/**@brief Application main function.
 */
uint8_t *data;
int main(void)
{

    uint32_t err_code;
    bool erase_bonds;
    uint8_t  start_string[] = START_STRING;
   uint8_t buff[200];
   uint8_t ret;

    uart_init();
   printf("start\r\n");

   init_spi_master();
   //先读出 200 字节
   ret = flash_read(0, buff, 200);
   if(ret!=SUCCESS)printf("read error\r\n");
   printf("原始数据\r\n");
   for(uint8_t i = 0; i < 200; i++){
       printf("%d ",buff[i]);
   }
   printf("\r\n");

   //擦除第 0 个 sector,并再读取下，
   ret = flash_sector_erase(0x0);
   if ( ret!=SUCCESS ) printf("erase error\r\n");
   ret = flash_read(0, buff, 200);
   printf("擦除后数据\r\n");
   if(ret!=SUCCESS)printf("read error\r\n");
   for(uint8_t i = 0; i < 200; i++){
       printf("%d ",buff[i]);
   }
   printf("\r\n");

   for(uint8_t i = 0; i < 200; i++){
       buff[i] = i;
   }
   //现在已经擦除过了，写数据进去
```

```
    ret = flash_page_write(0, buff, 200);
    if ( ret!=SUCCESS ) printf("write error\r\n");

    //再读数据看对不对
    ret = flash_read(0, buff, 200);
    printf("写后数据：\r\n");
    if(ret!=SUCCESS)printf("read error\r\n");
    for(uint8_t i = 0; i < 200; i++){
        printf("%d ",buff[i]);
    }
    printf("\r\n");

    while(1);

}
```