STM32Cube高效开发教程（基础篇）

第16章 SPI接口通信

王维波
中国石油大学（华东）控制科学与工程学院

STM32Cube高效开发教程（基础篇）

作者：王维波，鄢志丹，王钊

人民邮电出版社

2021年9月出版

如果有读者需要本书课件的PPT版本用于备课，可以给作者发邮件免费获取，并可加入专门的教学和技术交流QQ群

邮箱：wangwb@upc. edu. cn

16.1 SPI接口和通讯协议

16.2 SPI的HAL驱动程序

16.3 Flash存储芯片W25Q128

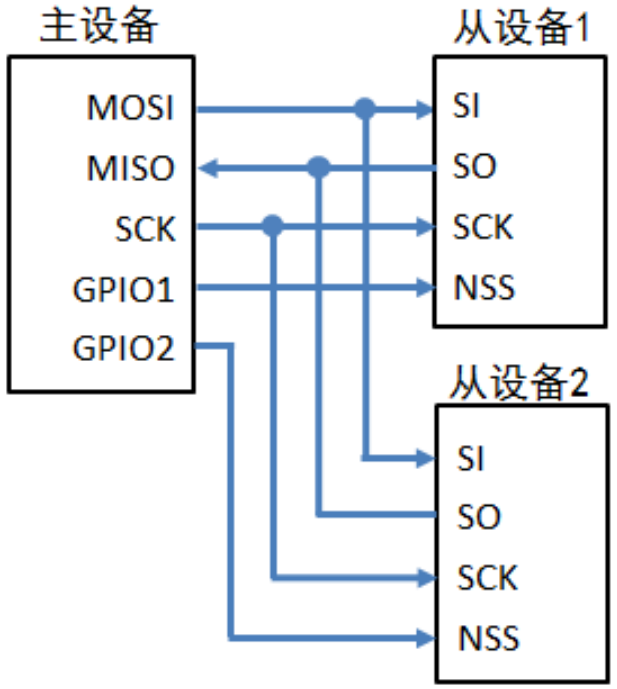
16.4 示例1：轮询方式读写W25Q128

16.1.1 SPI硬件接口

SPI是串行外设接口（Serial Peripheral Interface）

SPI接口的设备分为主设备（Master）和从设备（Slave），一个主设备可以连接一个或多个从设备

- MOSI（Mater Output Slave Input）
- MISO（Mater Input Slave Output）
- SCK，时钟信号
- NSS，片选信号

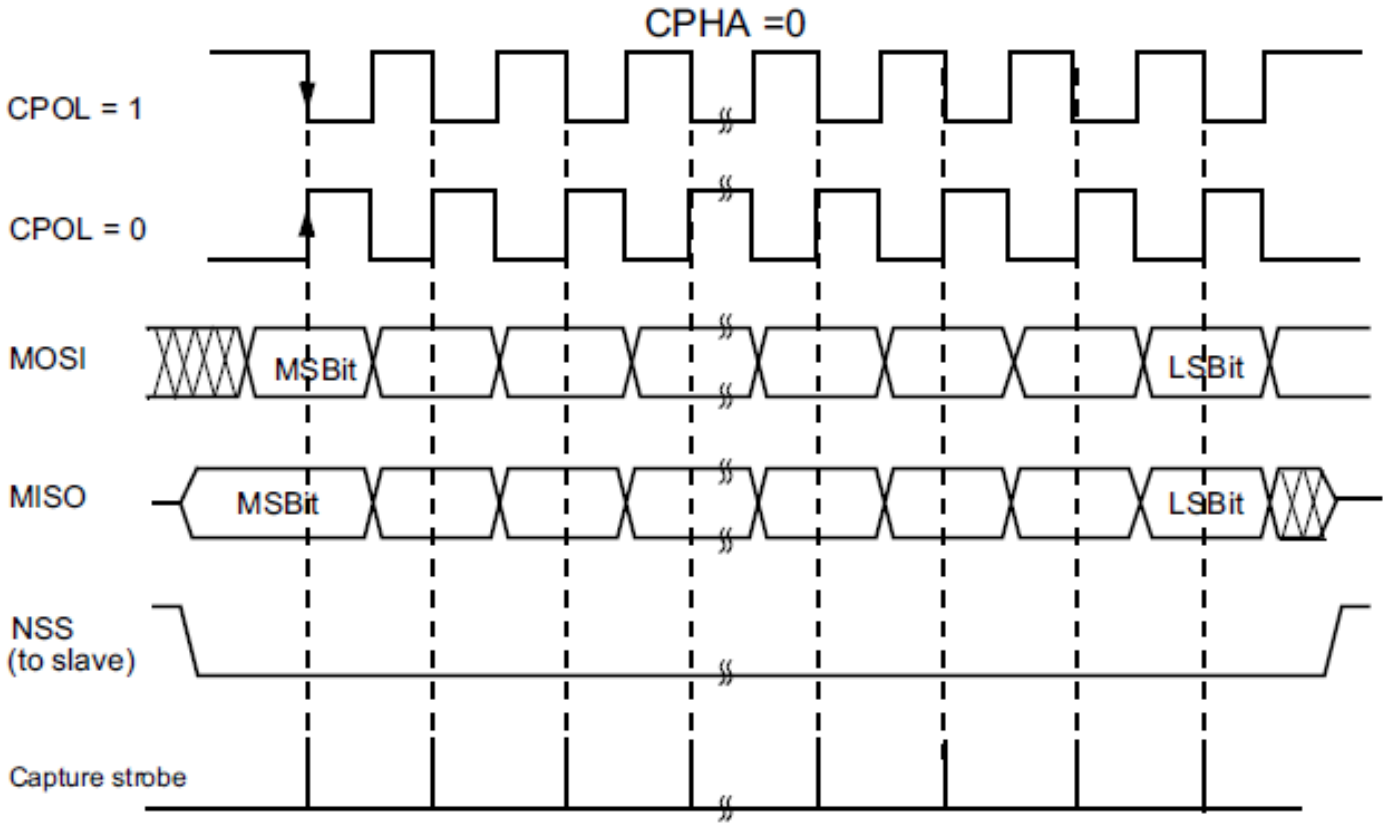


16.1.2 SPI传输协议

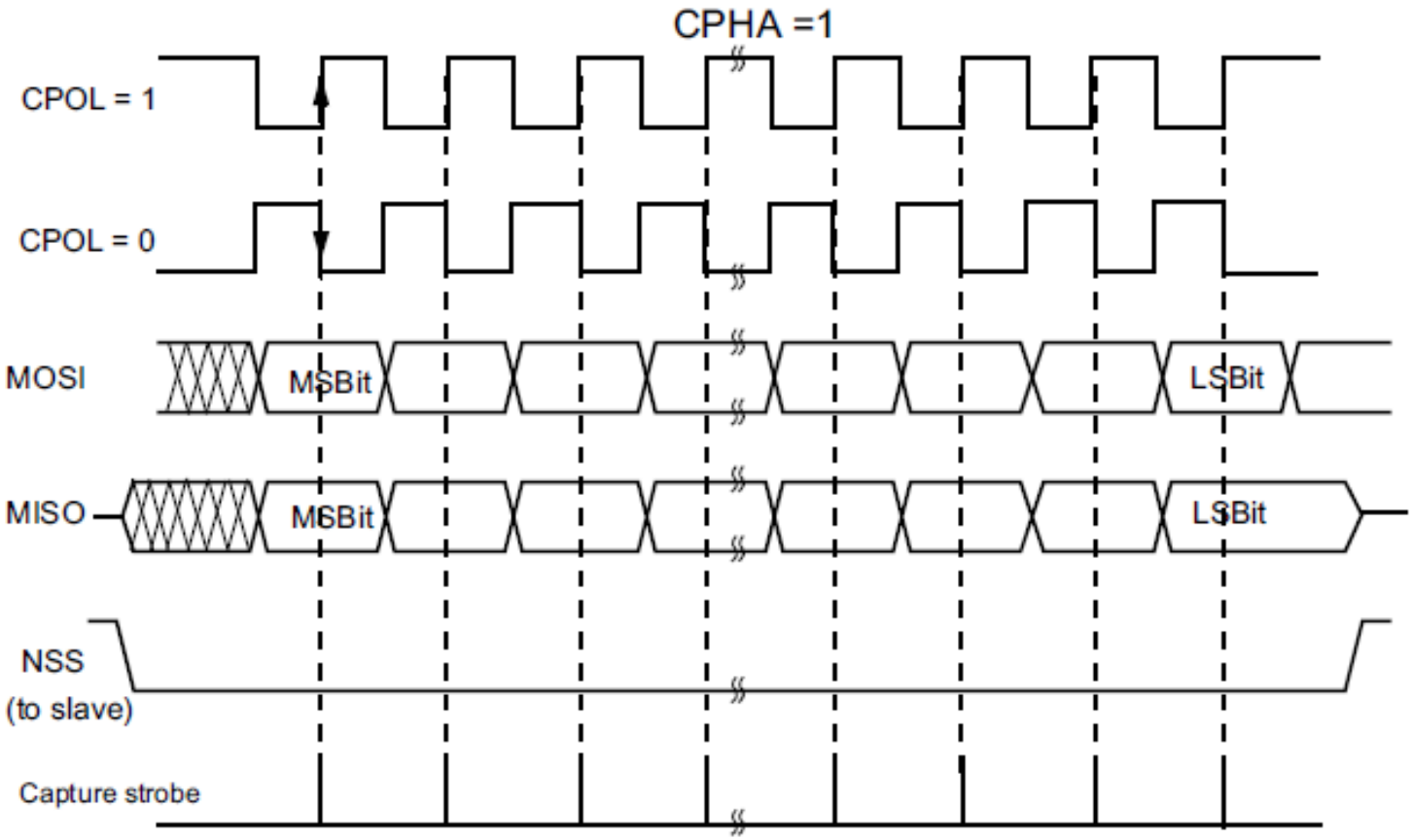
SPI通讯有4种时序模式，由SPI控制寄存器SPI_CR1中的CPOL和CPHA位控制。

- CPOL（Clock Polority）时钟极性，控制SCK引脚在空闲状态时的电平。如果CPOL=0，则空闲时SCK为低电平；若CPOL=1，则空闲时SCK为高电平。
- CPHA（Clock Phase）时钟相位，若CPHA=0，则在SCK的第1个边沿对数据采样；如果CPHA=1，则在SCK的第2个边沿对数据采样。

CPHA=0表示在SCK的第1个边沿读取数据，即图中虚线表示的时刻。读取数据的时刻发生在SCK的下跳沿（CPOL=1）时刻或上跳沿（CPOL=0）时刻。MISO、MOSI上的数据变化在读取数据的SCK前一个跳变沿时刻发生变化。



CPHA=1表示在SCK的第2个边沿读取数据，即图中的虚线表示的时刻。读取数据的时刻发生在SCK上跳沿（CPOL=1）时刻或下跳沿（CPOL=0）时刻。MISO、MOSI上的数据在读取数据的SCK前一个跳变沿时刻发生变化。



SPI的4种时序模式

SPI时序模式	CPOL 时钟极性	CPHA 时钟相位	空闲时 SCK电平	采样时刻
模式0	0	0	低电平	第1跳变沿
模式1	0	1	低电平	第2跳变沿
模式2	1	0	高电平	第1跳变沿
模式3	1	1	高电平	第2跳变沿

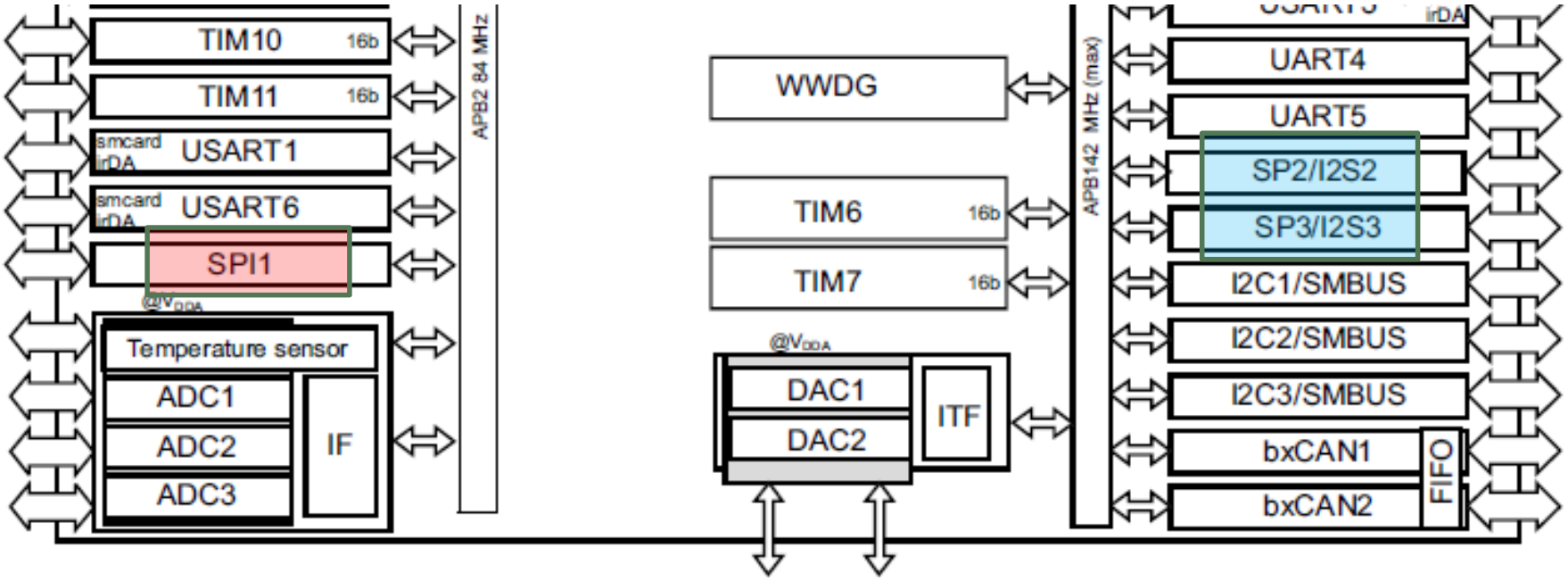
SPI主机和从机必须使用相同的SPI时序。

16.1.3 STM32F407的SPI接口

STM32F407芯片上有3个硬件SPI接口，可作为主机或从机

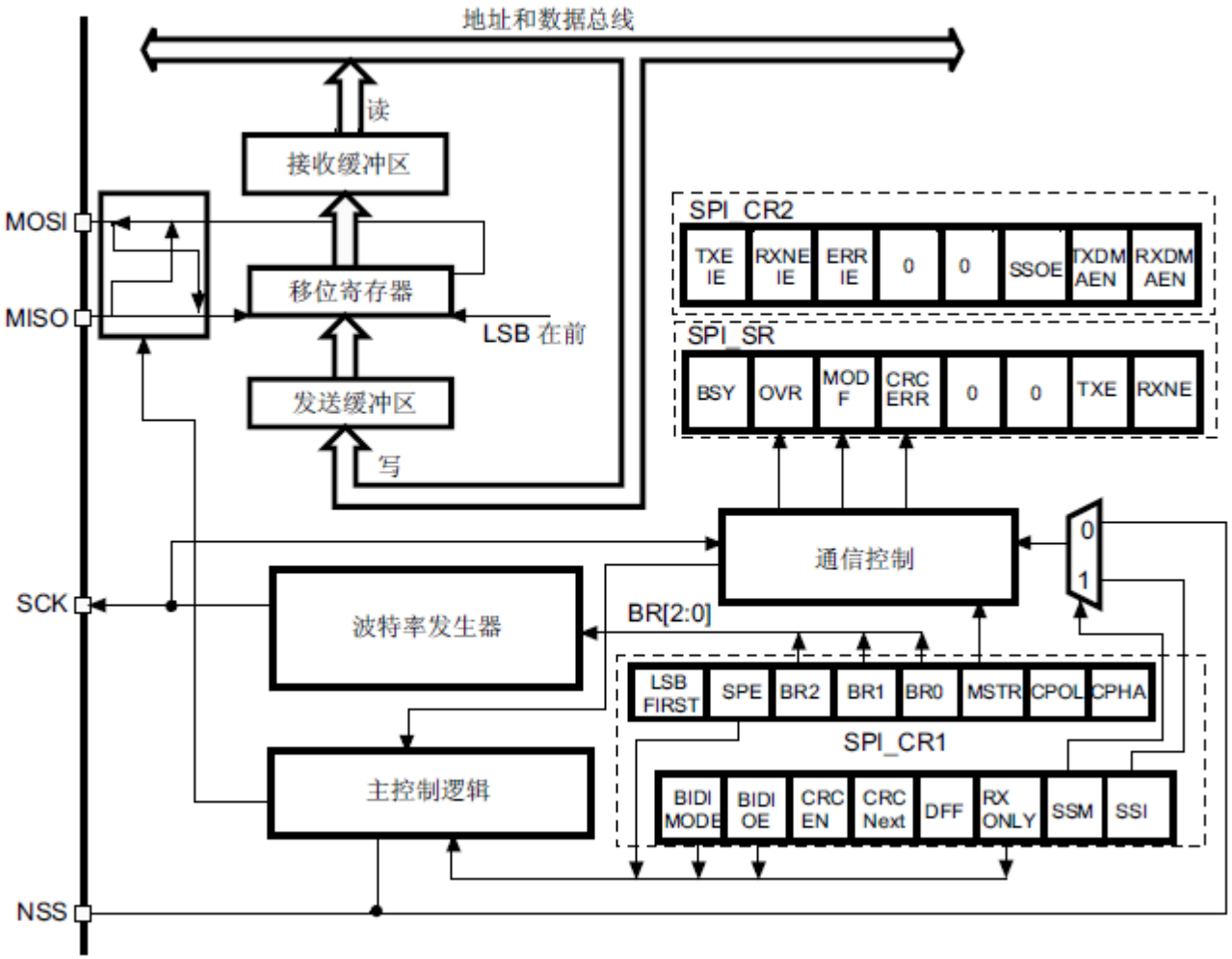
SPI1在APB2总线上，最高波特率42 Mbps

SP2、SPI3在APB1总线上，最高波特率21 Mbps



硬件SPI接口实现了SPI通信时序的控制，有相应的寄存器用于数据发送和接收。

也可以用GPIO口实现软件模拟SPI接口，也就是模拟SPI基本时序实现数据传输



16.1 SPI接口和通讯协议

16.2 SPI的HAL驱动程序

16.3 Flash存储芯片W25Q128

16.4 示例1：轮询方式读写W25Q128

16.2.1 SPI寄存器操作宏函数

宏函数	功能描述
__HAL_SPI_DISABLE(__HANDLE__)	禁用某个SPI接口
__HAL_SPI_ENABLE(__HANDLE__)	启用某个SPI接口
__HAL_SPI_DISABLE_IT(__HANDLE__, __INTERRUPT__)	禁止SPI的某个中断事件源
__HAL_SPI_ENABLE_IT(__HANDLE__, __INTERRUPT__)	开启SPI的某个中断事件源
__HAL_SPI_GET_IT_SOURCE(__HANDLE__, __INTERRUPT__)	检查SPI的某个中断事件源是否使能
__HAL_SPI_GET_FLAG(__HANDLE__, __FLAG__)	获取某个中断事件的挂起标志，检查中断事件是否发生
__HAL_SPI_CLEAR_CRCERRFLAG(__HANDLE__)	清除SPI的CRC校验错误中断挂起标志
__HAL_SPI_CLEAR_FREFLAG(__HANDLE__)	清除SPI的TI帧格式错误中断挂起标志
__HAL_SPI_CLEAR_MODFFLAG(__HANDLE__)	清除SPI的主模式故障中断挂起标志
__HAL_SPI_CLEAR_OVRFLAG(__HANDLE__)	清除SPI的溢出错误中断挂起标志

表16-2 SPI的中断事件和宏定义

中断事件	SPI状态寄存器（SPI_SR）中的中断挂起标志位	表示事件中断挂起标志位的宏	SPI控制寄存器2（SPI_CR2）中的中断事件使能位	表示中断事件使能位的宏（用于表示中断事件类型）
发送缓存区为空	TXE	SPI_FLAG_RXNE	TXEIE	SPI_IT_TXE
接收缓存区非空	RXNE	SPI_FLAG_TXE	EXNEIE	SPI_IT_RXNE
主模式故障	MODF	SPI_FLAG_MODF	ERRIE	SPI_IT_ERR
溢出错误	OVR	SPI_FLAG_OVR		
CRC校验错误	CRCERR	SPI_FLAG_CRCERR		
TI帧格式错误	FRE	SPI_FLAG_FRE		

一个SPI有6个中断事件，但只有3个中断使能位，其中一个错误事件中断使能位ERRIE控制了4种错误中断事件的使能。

16.2.2 SPI初始化和阻塞式数据传输

函数名	功能描述
HAL_SPI_Init()	SPI初始化，配置SPI接口参数
HAL_SPI_MspInit()	SPI的MSP初始化弱函数，重新实现时一般用于SPI接口引脚GPIO初始化和中断设置
HAL_SPI_GetState()	返回SPI接口当前状态，返回值是枚举类型HAL_SPI_StateTypeDef
HAL_SPI_GetError()	返回SPI接口最后的错误码，错误码有一组宏定义
HAL_SPI_Transmit()	阻塞式发送一个缓存区的数据
HAL_SPI_Receive()	阻塞式接收指定长度的数据到缓存区
HAL_SPI_TransmitReceive()	阻塞式同时发送和接收一定长度的数据

16.2.3 中断方式数据传输

函数名	函数功能	中断事件类型	对应的回调函数
HAL_SPI_Transmit_IT()	中断方式发送一个缓存区的数据	SPI_IT_TXE	HAL_SPI_TxCpltCallback()
HAL_SPI_Receive_IT()	中断方式接收指定长度的数据到缓存区	SPI_IT_RXNE	HAL_SPI_RxCpltCallback()
HAL_SPI_TransmitReceive_IT()	中断方式发送和接收一定长度的数据	SPI_IT_TXE和SPI_IT_RXNE	HAL_SPI_TxRxCpltCallback()
前3个中断方式传输函数	前3个中断模式传输函数都可能产生SPI_IT_ERR中断事件	SPI_IT_ERR	HAL_SPI_ErrorCallback()
HAL_SPI_IRQHandler()	SPI中断ISR函数里调用的通用处理函数	---	---
HAL_SPI_Abort()	取消非阻塞式数据传输，本函数以阻塞模式运行	---	---
HAL_SPI_Abort_IT()	取消非阻塞式数据传输，本函数以中断模式运行	---	HAL_SPI_AbortCpltCallback()

16.2.4 DMA方式数据传输

SPI的发送和接收有各自的DMA请求，能以DMA方式进行数据发送和接收。

DMA方式功能函数	函数功能	DMA流中断事件	对应的回调函数
HAL_SPI_Transmit_DMA()	DMA方式发送数据	DMA传输完成	HAL_SPI_TxCpltCallback()
		DMA传输半完成	HAL_SPI_TxHalfCpltCallback()
HAL_SPI_Receive_DMA()	DMA方式接收数据	DMA传输完成	HAL_SPI_RxCpltCallback()
		DMA传输半完成	HAL_SPI_RxHalfCpltCallback()
HAL_SPI_TransmitReceive_DMA()	DMA方式发送/接收数据	DMA传输完成	HAL_SPI_TxRxCpltCallback()
		DMA传输半完成	HAL_SPI_TxRxHalfCpltCallback()
前3个DMA方式传输函数	前3个DMA方式传输函数都可能产生DMA传输错误中断事件	DMA传输错误	HAL_SPI_ErrorCallback()

16.1 SPI接口和通讯协议

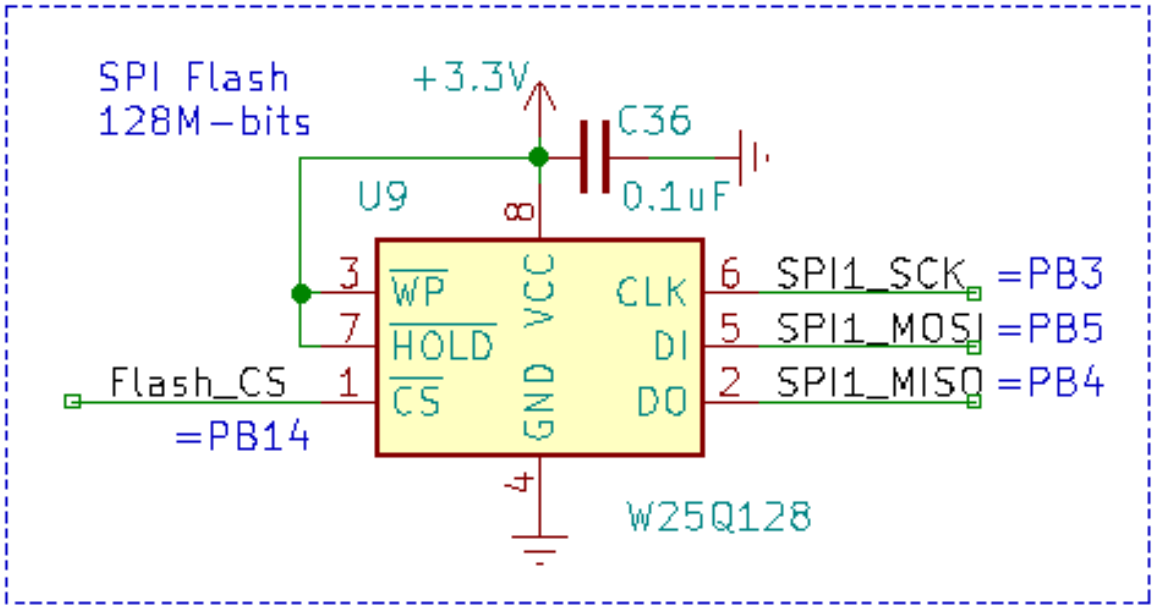
16.2 SPI的HAL驱动程序

16.3 Flash存储芯片W25Q128

16.4 示例1：轮询方式读写W25Q128

16.3.1 硬件接口和连接

W25Q128是一个SPI接口的Flash存储芯片，容量128M位，也就是16M字节。与MCU的SPI1接口连接。



W25Q128支持SPI模式0和模式3。在MCU与W25Q128通讯时，设置使用SPI模式3，即设置CPOL=1，CPHA=1。

16.3.2 存储空间划分

- W25Q128总容量是16M字节，使用24位地址线，地址范围是0x000000至0xFFFFFF
- 16M字节分为256个块（Block），每个块64K字节，16位偏移地址，块内偏移地址范围0x0000--0xFFFF
- 每个块又分为16个扇区（Sector），共4096个扇区，每个扇区4K字节，12位偏移地址，扇区内偏移地址范围0x000--0xFFF
- 每个扇区又分为16个页（Page），共65536个页，每个页256字节，8位偏移地址，页内偏移地址范围0x00--0xFF

16.3.3 数据读写的原则

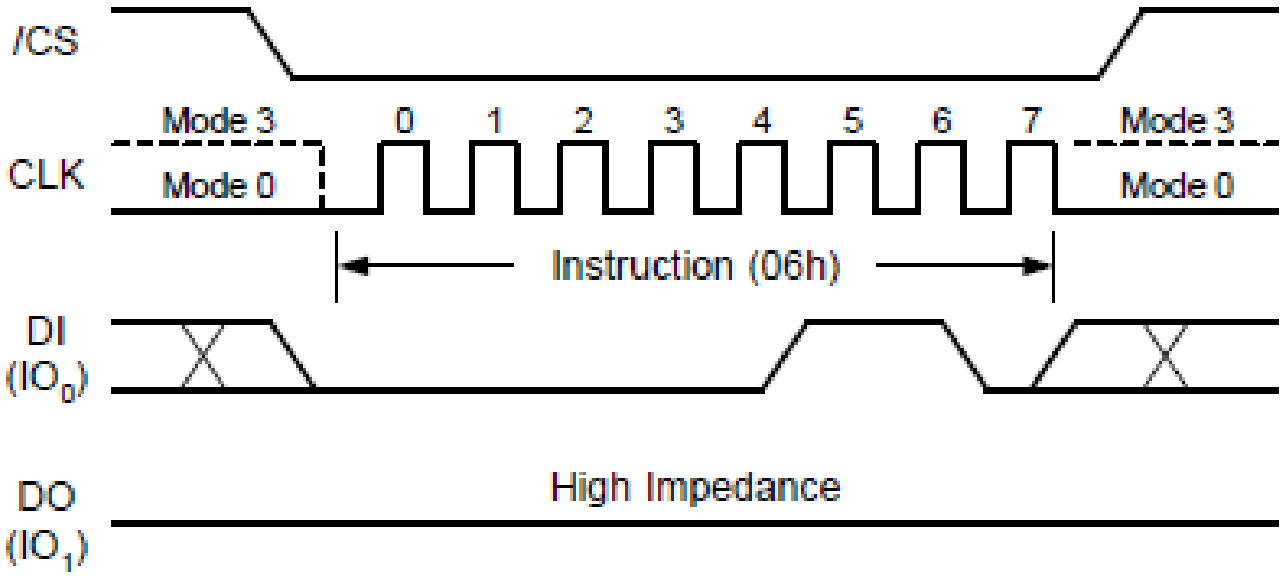
- 可以从任意地址开始读取任意长度的字节数据
- 可以从任何地址开始写数据，但是一次SPI通讯写入的数据范围不能超过一个页的边界。所以，如果从页的起始地址开始写数据，一次最多可写入一个页的数据，即256字节。如果一次写入的数据超过页的边界，会再从页的起始位置开始写
- 向存储区域写入数据时，存储区域必须是被擦除过的，也就是存储内容是0xFF，否则写入数据操作无效。可以对整个器件、某个块、某个扇区进行擦除操作，但是不能对单个页进行擦除

16.3.4 操作指令

W25Q128的操作指令由1个或多个字节组成，指令的第1个字节是指令码，后面跟随指令的参数，或返回的数据。表中用括号表示的部分表示返回的数据，A23-A0是24位的全局地址，dummy表示必须发送的无效字节数据，一般发送0x00

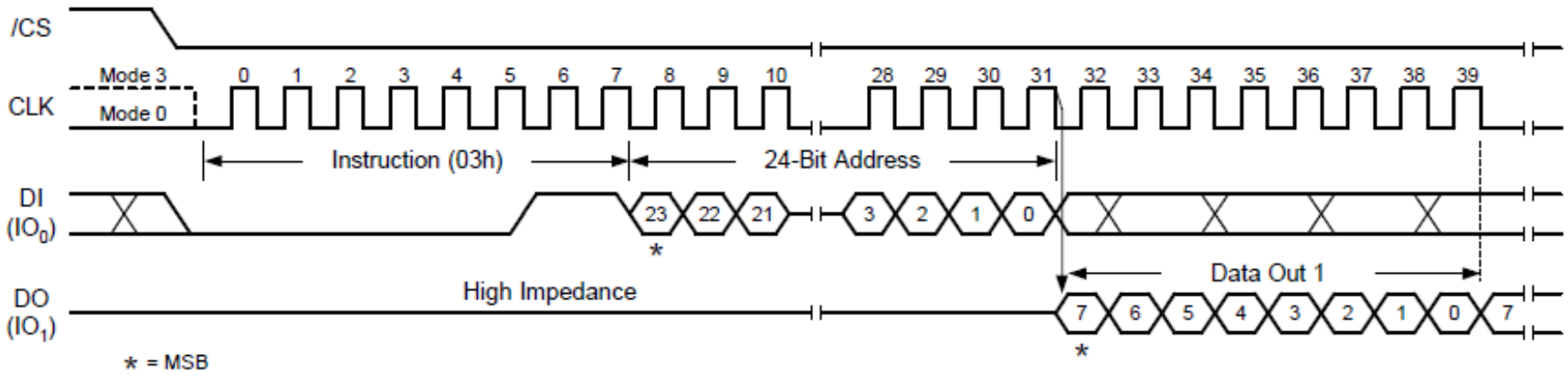
指令名称	BYTE 1指令码	BYTE 2	BYTE 3	BYTE 4	BYTE 5	BYTE 6
写使能	0x06					
读状态寄存器1	0x05	(S7-S0)				
读状态寄存器2	0x35	(S15-S8)				
读厂家和设备ID	0x90	dummy	dummy	0x00	(MF7-MF0)	(ID7-ID0)
读64位序列号	0x4B	dummy	dummy	dummy	dummy	(ID63-ID0)
器件擦除	0xC7/0x60					
块擦除（64K）	0xD8	A23-A16	A15-A8	A7-A0		
扇区擦除（4KB）	0x20	A23-A16	A15-A8	A7-A0		
写数据（页编程）	0x02	A23-A16	A15-A8	A7-A0	D7-D0	
读数据	0x03	A23-A16	A15-A8	A7-A0	(D7-D0)	
快速读数据	0x0B	A23-A16	A15-A8	A7-A0	dummy	(D7-D0)

1. “写使能”指令（指令码0x06）



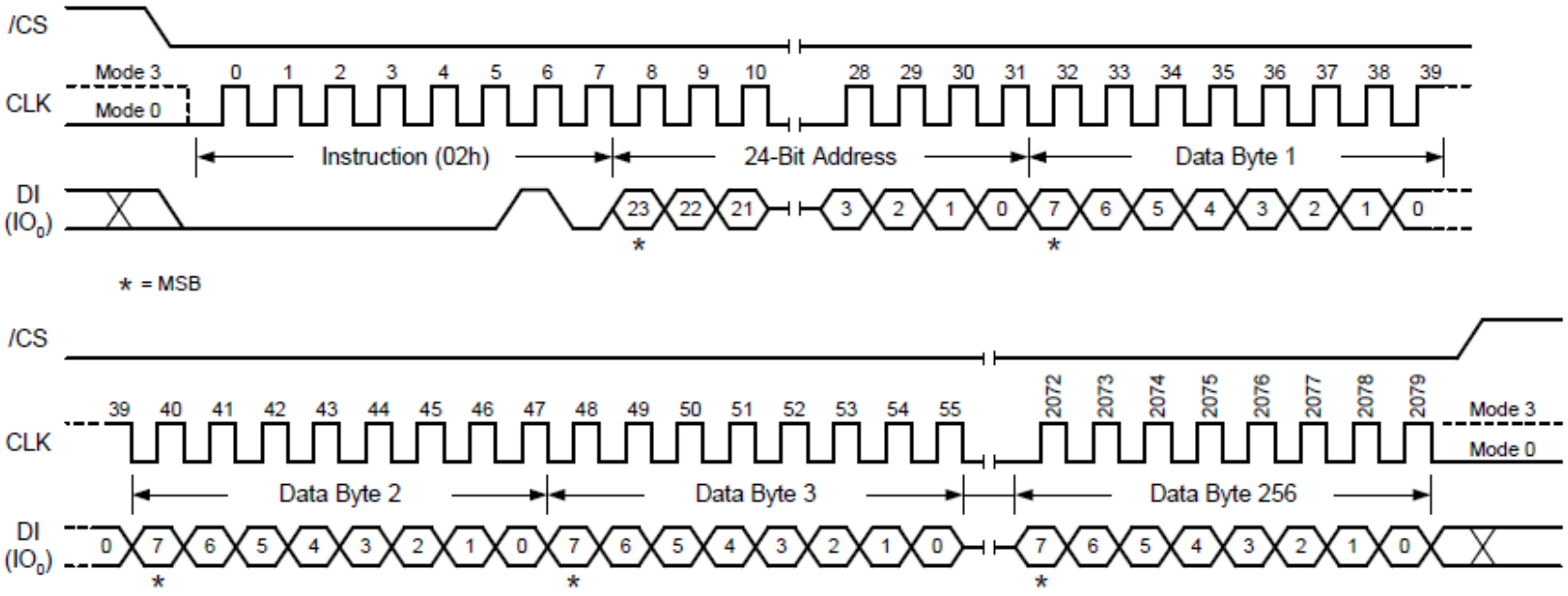
“写使能”指令（指令码0x06）只有一个指令码，其传输过程如图所示。一个指令总是从片选信号CS由高到低的跳变开始，片选信号CS由低到高的跳变结束。

2. “读数据”指令（指令码0x03）



“读数据”指令（指令码0x03）用于从某个地址开始读取一定个数的字节数据，其时序如图所示。地址A23-A0是24位全局地址，分解为3个字节，在发送指令码0x03后，再发送3字节的地址数据。然后MCU开始从DO线上读取数据，一次读取1个字节，可以连续读取，W25Q128会自动返回下一地址的数据。

3. “写数据”指令（指令码0x02）



“写数据”指令（指令码0x02）就是数据手册上的“页编程”指令，用于向任意地址开始写入一定长度的数据。“页编程”指令的时序如图所示，图中是向一个页一次写入256字节数据

16.1 SPI接口和通讯协议

16.2 SPI的HAL驱动程序

16.3 Flash存储芯片W25Q128

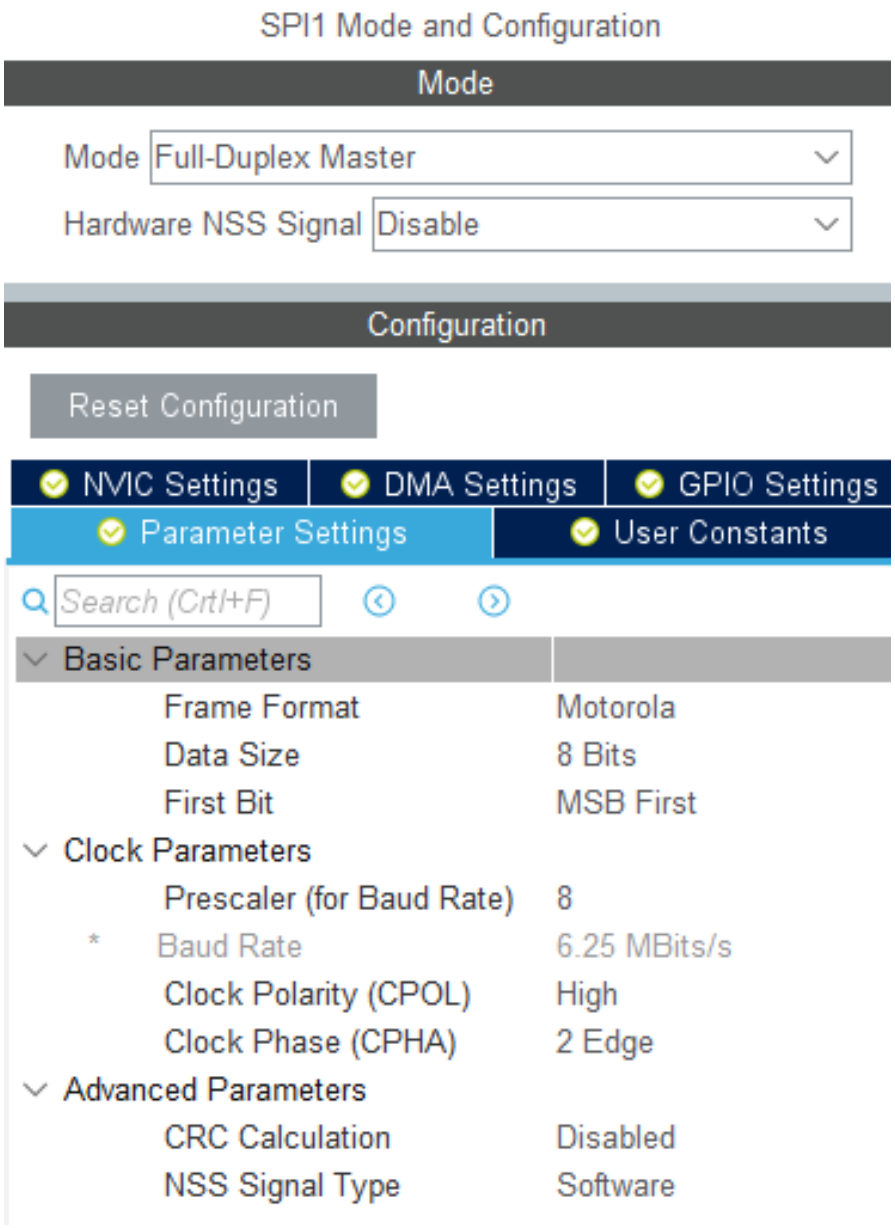
16.4 示例1：轮询方式读写W25Q128

16.4.1 示例功能与CubeMX项目设置

- 使用SPI1接口读写Flash存储器W25Q128。
- 使用阻塞式SPI传输函数编写W25Q128常用功能的驱动程序。
- 通过模拟菜单测试擦除整个芯片、擦除块、写入数据和读出数据的操作。

SPI1的设置

- 设置Prescaler后自动计算波特率，6.25 Mbps比较稳定
- MSB先行
- SPI模式3（CPOL=1, CPHA=1）



16.4.2 初始程序

1. 主程序

```
int main(void)
{
	HAL_Init();
	SystemClock_Config();
	/* Initialize all configured peripherals */
	MX_GPIO_Init();                    //对4个按键引脚和PB14的GPIO初始化
	MX_FSMC_Init();
	MX_SPI1_Init();                    //SPI1接口初始化
	/* Infinite loop */
	while (1)
	{
	}
}
```

2. SPI1初始化

SPI1的初始化函数MX_SPI1_Init()

代码较长，见源代码

```
/* 文件： spi.c  ------------------------------------------------------------*/
#include "spi.h"
SPI_HandleTypeDef hspi1;            //表示SPI1的外设对象变量

/*  SPI1 初始化函数  */
void MX_SPI1_Init(void)
{
    hspi1.Instance = SPI1;       //SPI1的寄存器基址
    hspi1.Init.Mode = SPI_MODE_MASTER;       //主机模式
    hspi1.Init.Direction = SPI_DIRECTION_2LINES;   //2线制，全双工
    hspi1.Init.DataSize = SPI_DATASIZE_8BIT;         //8位数据
    hspi1.Init.CLKPolarity = SPI_POLARITY_HIGH;    //CPOL=1
    hspi1.Init.CLKPhase = SPI_PHASE_2EDGE;              //CPHA=1
    hspi1.Init.NSS = SPI_NSS_SOFT;                //软件产生NSS
    hspi1.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_8;    //预分频系数
    hspi1.Init.FirstBit = SPI_FIRSTBIT_MSB;//MSB先行
    hspi1.Init.TIMode = SPI_TIMODE_DISABLE; //帧格式，Motorola
    hspi1.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;     //禁用CRC
    hspi1.Init.CRCPolynomial = 10;        //CRC多项式
    if (HAL_SPI_Init(&hspi1) != HAL_OK)
        Error_Handler();
}
```

16.4.3 编写W25Q128的驱动程序

1. W25Q128驱动程序头文件
2. SPI 基本发送和接收函数
3. W25Q128基本操作指令
4. 计算地址的辅助功能函数
5. 器件、块、扇区擦除函数
6. 存储区读写函数

1. W25Q128驱动程序头文件

文件w25flash.h，查看源代码

2. SPI 基本发送和接收函数

SPI阻塞式数据传输函数的封装

```
#include “w25flash.h"
#define MAX_TIMEOUT   200   //SPI轮询操作时的最大等待时间,单位ms

//SPI接口发送一个字节, byteData是需要发送的数据
HAL_StatusTypeDef  SPI_TransmitOneByte(uint8_t  byteData)
{
      return HAL_SPI_Transmit(&SPI_HANDLE, &byteData, 1, MAX_TIMEOUT);
}


//SPI接口接收一个字节，  返回接收的一个字节数据
uint8_t  SPI_ReceiveOneByte()
{
      uint8_t  byteData=0;
      HAL_SPI_Receive(&SPI_HANDLE, &byteData, 1, MAX_TIMEOUT);
      return  byteData;
}
```

3. W25Q128基本操作指令

W25Q128基本操作函数的封装

```
//=========2. W25Qxx 基本控制指令==========
uint16_t  Flash_ReadID(void); // Command=0x90, Manufacturer/Device ID
uint64_t  Flash_ReadSerialNum(uint32_t* High32,  uint32_t* Low32);
//Command=0x4B, Read Unique ID, 64-bit

HAL_StatusTypeDef Flash_Write_Enable(void);
//Command=0x06: Write Enable,    使WEL=1

HAL_StatusTypeDef Flash_Write_Disable(void);
//Command=0x04, Write Disable,    使WEL=0

uint8_t Flash_ReadSR1(void);
//Command=0x05:  Read Status Register-1,返回寄存器SR1的值

uint8_t Flash_ReadSR2(void);
//Command=0x35:  Read Status Register-2,返回寄存器SR2的值
```

4. 计算地址的辅助功能函数

通过块、扇区、页编号计算24位的绝对地址，或将24位地址分解为3个字节。

```
//根据Block  绝对编号获取地址,共256个Block
uint32_t  Flash_Addr_byBlock(uint8_t BlockNo);
//根据Sector 绝对编号获取地址,共4096个Sector
uint32_t  Flash_Addr_bySector(uint16_t  SectorNo);
//根据Page  绝对编号获取地址，共65536个Page
uint32_t  Flash_Addr_byPage(uint16_t  PageNo);

//根据Block编号，和内部Sector编号计算地址，一个Block有16个Sector,
uint32_t  Flash_Addr_byBlockSector(uint8_t BlockNo, uint8_t SubSectorNo);
//根据Block编号，内部Sector编号，内部Page编号计算地址
uint32_t  Flash_Addr_byBlockSectorPage(uint8_t BlockNo, uint8_t
SubSectorNo, uint8_t  SubPageNo);
//将24位地址分解为3个字节
Void  Flash_SpliteAddr(uint32_t globalAddr, uint8_t* addrHigh, uint8_t*
addrMid,uint8_t* addrLow);
```

5. 器件、块、扇区擦除函数

写Flash之前，存储区域必须是被擦除过的

```
//Command=0xC7: Chip Erase, 擦除整个器件,大约25秒
void Flash_EraseChip(void);

//Command=0xD8: Block Erase(64KB) 擦除整个Block, globalAddr是全局地址，耗时大约150ms
void Flash_EraseBlock64K(uint32_t globalAddr);

//Command=0x20: Sector Erase(4KB) 扇区擦除, globalAddr是扇区的全局地址，耗时大约30ms
void Flash_EraseSector(uint32_t globalAddr);
```

6. 存储区读写函数

将数据读写封装为函数

```
//Command=0x03，读取一个字节，任意全局地址
uint8_t Flash_ReadOneByte(uint32_t globalAddr);

//Command=0x03，连续读取多个字节，任意全局地址
void Flash_ReadBytes(uint32_t globalAddr, uint8_t* pBuffer，uint16_t byteCount);

//Command=0x0B，高速连续读取多个字节，任意全局地址， 速度大约是常规读取的2倍
void Flash_FastReadBytes(uint32_t globalAddr, uint8_t* pBuffer，uint16_t byteCount);

//Command=0x02: Page program 对一个Page写入数据（最多256字节）, globalAddr是初始位置的全局地址，耗时大约3ms
void Flash_WriteInPage(uint32_t globalAddr, uint8_t* pBuffer, uint16_t byteCount);
```

16.4.4 W25Q128功能测试

修改main()函数代码，在LCD上显示模拟菜单，分别进行响应。

[1]KeyUp = Erase Chip

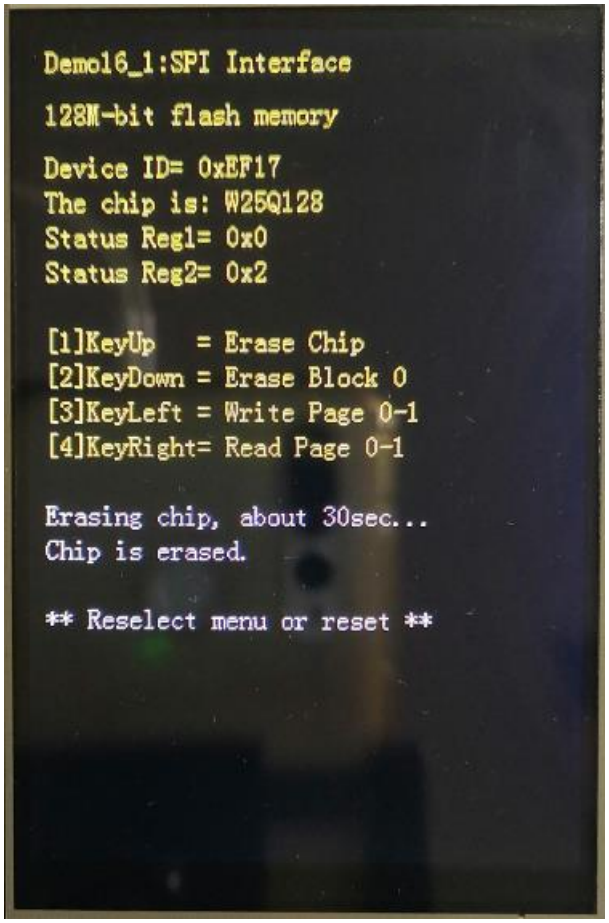
[2]KeyDown = Erase Block 0

[3]KeyLeft = Write Page 0-1

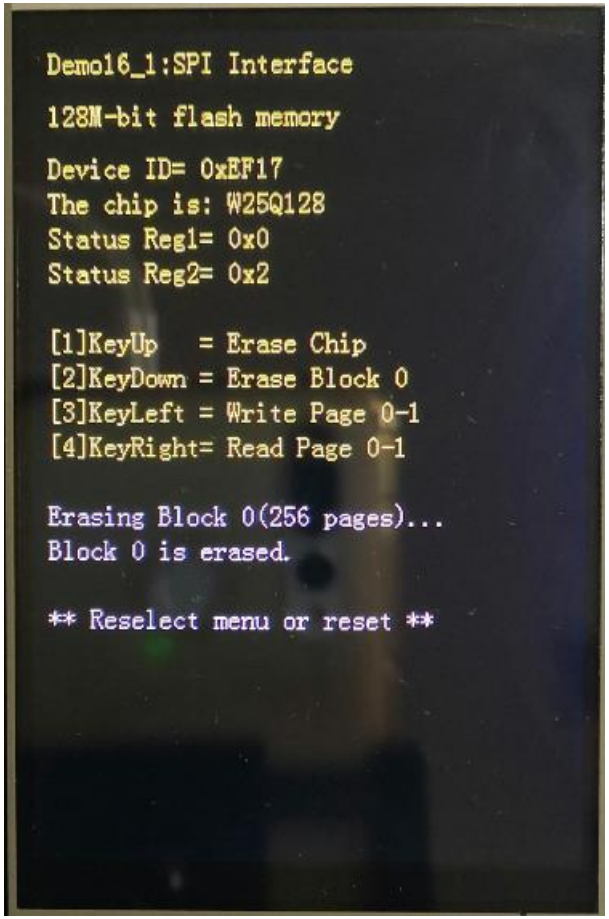
[4]KeyRight = Read Page 0-1

程序较长，看源程序和教材讲解

运行测试

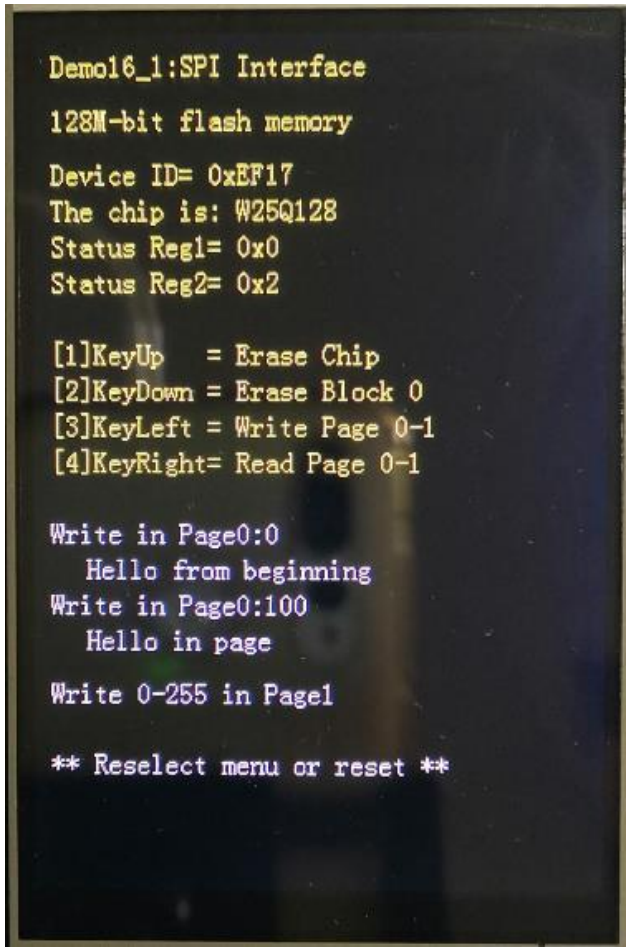


按KeyUp键擦除整个器件

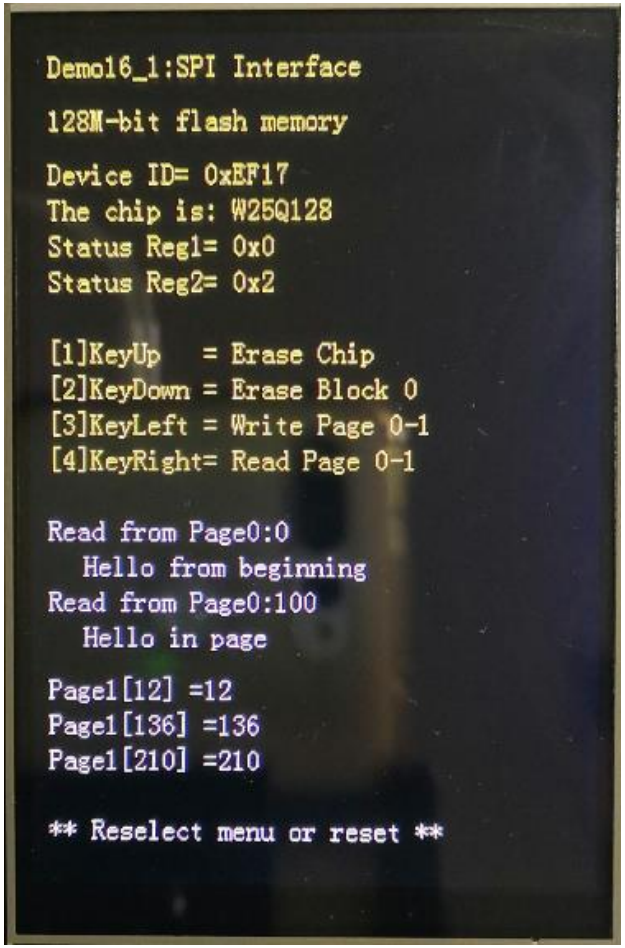


按KeyDown键擦除Block 0

运行测试



按KeyLeft键，写入Page0和Page1



按KeyRight键，读取数据

练习任务

1. 根据讲解，自己做本章示例程序

2. 使用SPI的DMA方式，实现本章的示例。只有数据读写采用DMA方式，擦除等操作仍然使用阻塞式。