第 2-13 讲：SPI 总线的应用

1. 学习目的

1. 了解 SPI 总线的结构、特点以及 4 种通信模式。
2. 掌握通过 SPI 读、写和擦除 SPI Flash W25Q128 的方法以及代码编写。
3. 掌握通过 SPI 读、写铁电存储器 FM25CL64B 的方法以及代码编写。

2. SPI 总线原理

SPI 是串行外设接口(Serial Peripheral Interface)的缩写，是一种高速、全双工、同步的通信总线。SPI 是 Motorola 公司推出的一种同步串行接口技术，SPI 由一个主设备和一个或多个从设备组成，在一次数据传输过程中，接口上只能有一个主机和一个从机能够通信。

SPI 总线的优点是操作简单、数据传输速率较高、全双工，缺点是只支持单个主机、没有指定的流控制，没有应答机制确认是否接收到数据。

2.1. 接口信号定义

SPI 总线接口包括以下四种信号：

1) MOSI（Master Output，Slave Input）：主器件数据输出，从器件数据输入。
2) MISO（Master Input， Slave Output）：主器件数据输入，从器件数据输出。
3) SCK（Serial Clock）：有时也称为 SCLK，时钟信号，由主器件产生。
4) CS（Chip select）：有时也称为 SS，从器件使能信号，由主器件控制，实际使用时，经常用 GPIO 来代替。

SPI 总线支持连接多个从机，如下图所示，SPI 主机通过连接到从机的片选信号使能/禁止从机，并且同时只能使能一个从机，因此总线里面有多少个从机，就需要多少个片选信号。当 SPI 主机需要和总线中某个从机进行通信时，主机会拉低对应的 CS 信号使能该从机，之后发起通信，通信完成后，拉高 CS 信号，解除总线的占用。

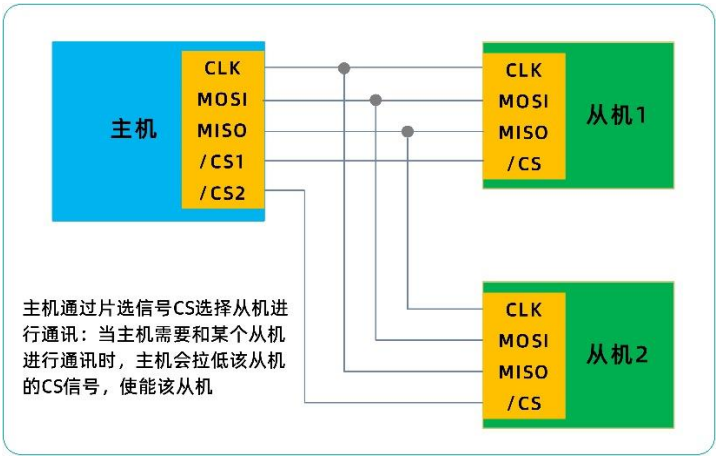


图 1：SPI 总线结构

对于 SPI 总线，我们还需要能深刻理解下面几个知识点。

1. 硬件片选和软件片选的区别

所谓硬件片选指的是 SPI 本身具有片选信号，当我们通过 SPI 发送数据时，SPI 外设自动拉低 CS 信号使能从机，发送完成后自动拉高 CS 信号释放从机，这个过程是不需要软件操作的。而软件片选则是需要使用 GPIO 作为片选信号，SPI 在发送数据之前，需要先通过软件设置作为片选信号的 GPIO 输出低电平，发送完成之后再设置该 GPIO 输出高电平。

2. SPI 总线是回环结构

SPI 是一个环形总线结构，如下图所示，主设备和从设备构成一个环形。在时钟 SCK 的作用下，主设备发送一个位到从设备，因为是环形结构，所以从设备必定会同时传送一个位到主设备。同样，主设备向从设备发送一个字节，从设备也必定会同时传送一个字节到主设备。理解了环形结构，就很容易理解下面几点：

- SPI 是全双工，同步的通信总线。
- 主设备向从设备发送数据时，无论我们需不需要从设备返回数据，从设备都会返回数据。
- 主设备从从设备读取一个字节数据时，为什么需要写一个字节数据到从设备：因为是环形结构，不写数据过去，对方的数据就不会被移位过来。

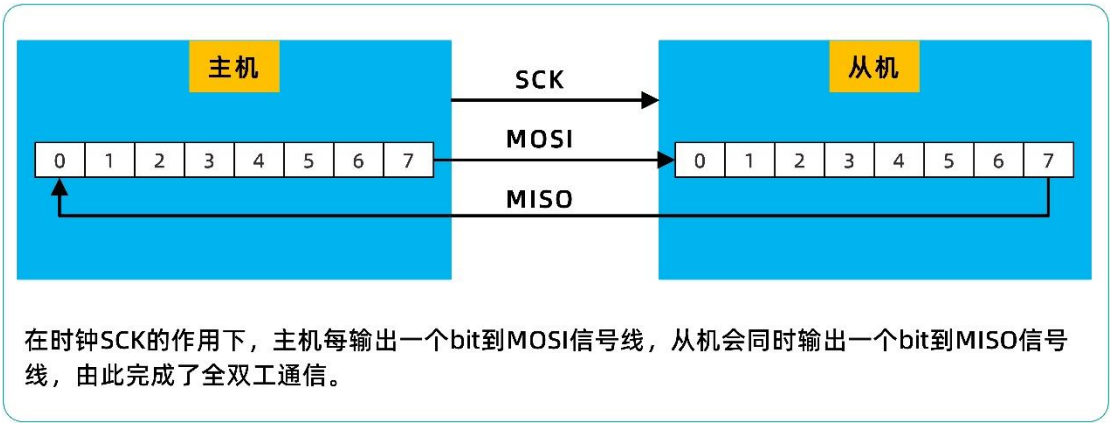


图 2：SPI 数据传输示意图

3. SPI 主机和从机之间连接时信号不需要交叉

SPI 主机和从机连接时，MOSI 和 MISO 信号是不需要交叉连接的，因为 MOSI 本身就表示了主机输出、从机输入，MISO 表示主机输入、从机输出，因此不能交叉连接。

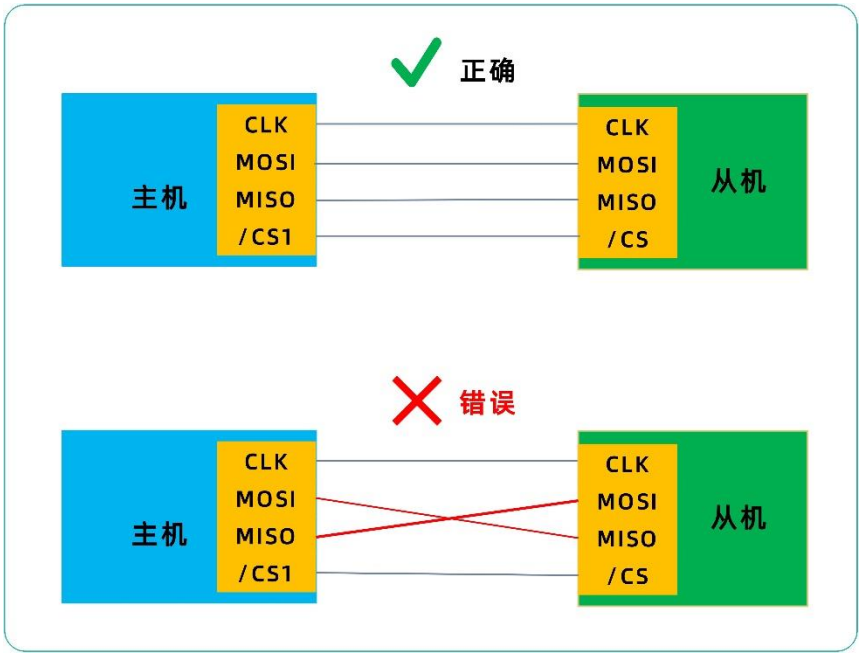


图 3：MOSI 和 MISO 不能交叉连接

2.2. SPI 的 4 种通信模式

SPI 总线共有 4 种通信模式：模式 0~模式 3，这 4 种通信模式是由时钟相位和时钟极性确定的。

1. 时钟相位 CPOL（Clock polarity）：SPI 总线空闲时，时钟信号 SCLK 的电平称为时钟极性，有以下两种模式：
   - CPOL=0：SPI 总线空闲时，时钟信号为低电平。
   - CPOL=1：SPI 总线空闲时，时钟信号为高电平。
2. 时钟极性 CPHA（Clock phase）：SPI 在时钟信号 SCLK 第几个边沿开始采样数据，有以下两种模式：
   - CPHA=0：在第 1 个时钟边沿进行数据采样。
   - CPHA=1：在第 2 个时钟边沿进行数据采样。

时钟极性 CPOL 时钟相位 CPHA 各有 2 种模式，他们两两组合就形成了 SPI 的 4 种通信模式，如下表所示。

表 1：SPI 总线的 4 种工作模式

模式	描述
模式 0	CPOL=0，CPHA=0
模式 1	CPOL=0，CPHA=1
模式 2	CPOL=1，CPHA=0
模式 3	CPOL=1，CPHA=1

SPI 的 4 种模式中，最常用的是模式 0 和模式 3。正是由于 SPI 有 4 种通信模式，因此当我们使用 SPI 总线时，需要去查询 SPI 总线中主机设备（如 STC8A8K64D4）和从机设备

（如 SPI Flash）的数据手册，确定他们支持什么模式，从而选择适合的通信模式。

SPI 的 4 种模式的时序图如下。

1) 时钟相位 CPHA=0 时的时序

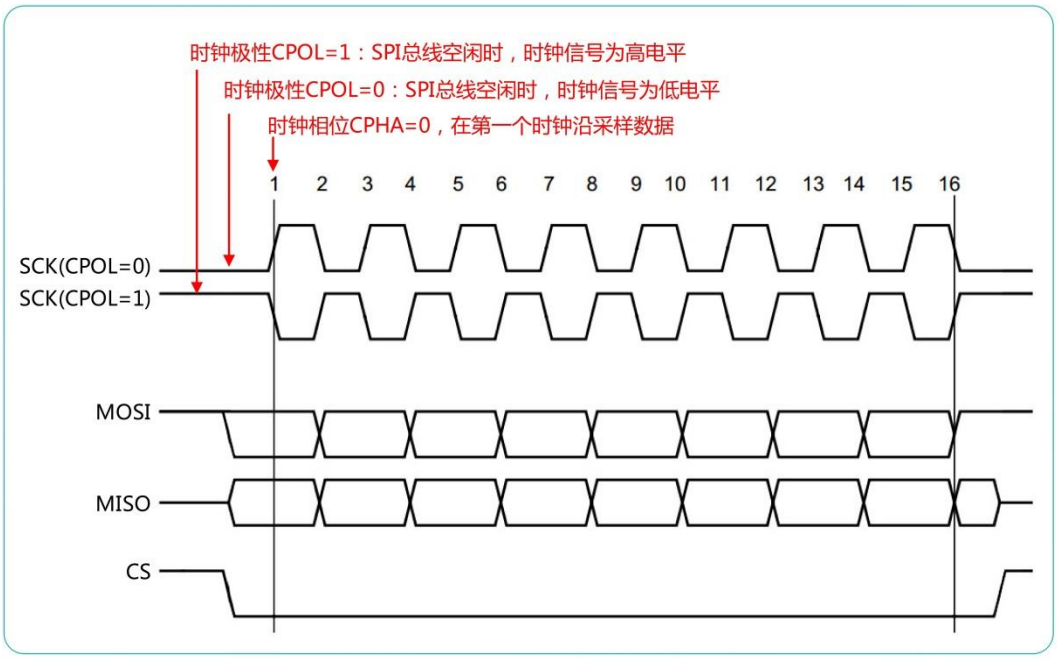


图 4：CPHA=0 时 SPI 时序

2) 时钟相位 CPHA=1 时的时序

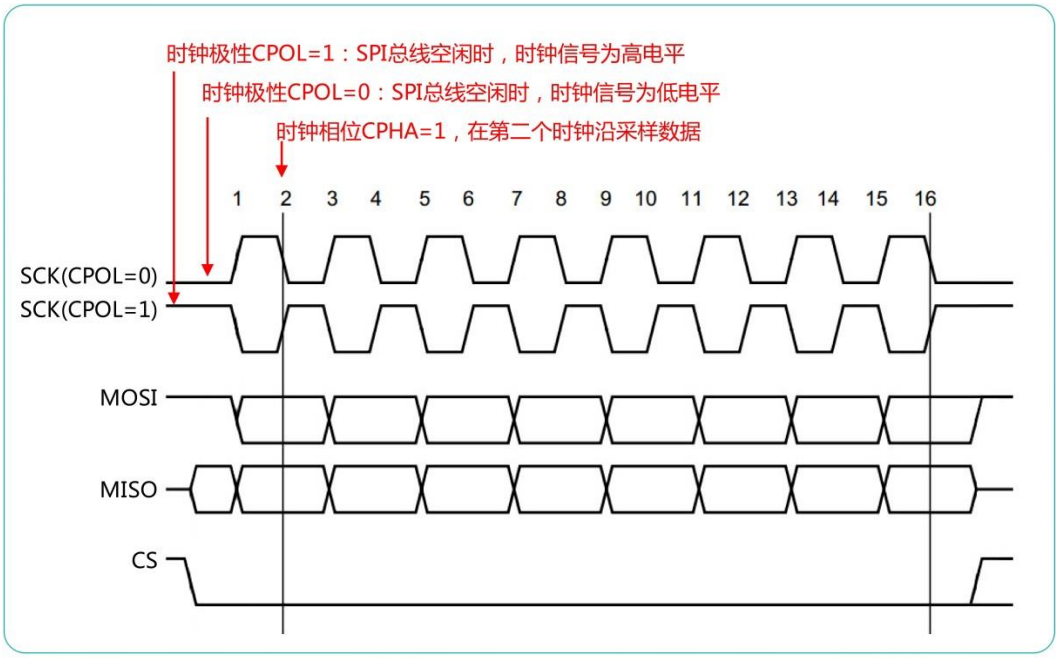


图 5：CPHA=0 时 SPI 时序

3. STC8A8K64D4 的 SPI 应用步骤

STC8A8K64D4 单片机片内集成了一个高速串行通信接口(SPI)，SPI 是一种全双工的高速同步通信总线。STC8A8K64D4 单片机的 SPI 支持主机和从机模式，通过配置寄存器，

可以让 SPI 工作于主机模式或从机模式。

3.1. SPI 引脚配置

SPI 有多组引脚与之对应（具体几组还取决于芯片封装引脚数），同一时刻，只能通过相关寄存器配置其中的一组使用， STC8A8K64D4 单片机 SPI 的引脚分配如下表。

表 2：STC8A8K64D4 单片机 SPI 引脚分配

SPI 信号	信号编号	对应的 IO
SPI 时钟	SCLK	P1.5
	SCLK_2	P2.5
	SCLK_3	P7.7
	SCLK_4	P3.2
SPI 主出从入(MOSI)	MOSI	P1.3
	MOSI_2	P2.3
	MOSI_3	P7.5
	MOSI_4	P3.4
SPI 主入从出(MISO)	MISO	P1.4
	MISO_2	P2.4
	MISO_3	P7.6
	MISO_4	P3.3
SPI 片选	SS	P1.2
	SS_2	P2.2
	SS_3	P7.4
	SS_4	P3.5

SPI 是通过“外设端口切换控制寄存器 1（P_SW1）”中的 SPI_S[1:0]配置引脚的，如下图所示。

外设端口切换控制寄存器 1（P_SW1）：

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P_SW1	A2H	S1_S[1:0]		-	-	SPI_S[1:0]		0	-

P_SW1 寄存器中的 SPI_S[1:0]为 SPI 功能脚选择位，如下表所示。

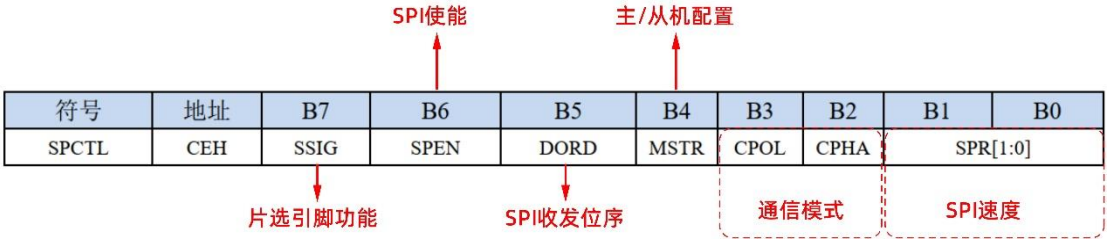
表 3：SPI 功能脚选择位

SPI_S[1:0]	SS	MOSI	MISO	SCLK
00	P1.2	P1.3	P1.4	P1.5
01	P2.2	P2.3	P2.4	P2.5
10	P7.4	P7.5	P7.6	P7.7
11	P3.5	P3.4	P3.3	P3.2

3.2. 配置 SPI 工作参数

SPI 工作参数通过“SPI 控制寄存器（SPCTL）”配置，配置项包括：SS 引脚功能、SPI 收发位序、主/从机、工作模式、速度以及 SPI 使能。

SPI 控制寄存器（SPCTL）：



- SSIG：SS（片选）引脚功能控制位
  - 0：SS 引脚确定器件是主机还是从机。
  - 1：忽略 SS 引脚功能，使用 MSTR 确定器件是主机还是从机。

  通常，我们会将“SS（片选）引脚功能控制位”设置为 1，即通过配置“MSTR”位来设置 SPI 工作于主机或是从机。
- SPEN：SPI 使能控制位
  - 0：关闭 SPI 功能。
  - 1：使能 SPI 功能。
- DORD：SPI 数据位发送/接收的顺序
  - 0：先发送/接收数据的高位（MSB）。
  - 1：先发送/接收数据的低位（LSB）。
- MSTR：器件主/从模式选择位

  设置主机模式：
  - 若 SSIG＝ 0，则 SS 管脚必须为高电平且设置 MSTR 为 1。
  - 若 SSIG＝ 1，则只需要设置 MSTR 为 1（忽略 SS 管脚的电平）。

  设置从机模式：
  - 若 SSIG＝ 0，则 SS 管脚必须为低电平（与 MSTR 位无关）。
  - 若 SSIG＝ 1，则只需要设置 MSTR 为 0（忽略 SS 管脚的电平）。
- CPOL：SPI 时钟极性控制
  - 0：SCLK 空闲时为低电平，SCLK 的前时钟沿为上升沿，后时钟沿为下降沿。
  - 1：SCLK 空闲时为高电平，SCLK 的前时钟沿为下降沿，后时钟沿为上升沿。
- CPHA： SPI 时钟相位控制
  - 0：数据 SS 管脚为低电平驱动第一位数据并在 SCLK 的后时钟沿改变数据，前时钟沿采样数据（必须 SSIG＝ 0）。
  - 1：数据在 SCLK 的前时钟沿驱动，后时钟沿采样。

  由“CPOL”和“CPHA”可见，STC8A8K64D4 的 SPI 支持 SPI 的 4 种通信模式。
- SPR[1:0]：SPI 时钟频率选择

SPR[1:0]设置的是 SPI 的 SCLK 频率，他决定了 SPI 的传输速率，STC8A8K64D4 的 SPI 是快速 SPI，可配置的时钟频率如下表所示。

表 4：SPI 时钟频率

SPR[1:0]的值	SCLK 频率
00	SYSclk/4
01	SYSclk/8
10	SYSclk/16
11	SYSclk/2

3.3. 数据收发

SPI 的数据传输只能由主机启动，因为只有主机能够产生时钟信号。主机对 SPI 数据寄存器 SPDAT 的写操作将启动 SPI 时钟发生器和数据的传输。在数据写入 SPDAT 之后的半个到一个 SPI 位时间后，数据将出现在 MOSI 脚。写入主机 SPDAT 寄存器的数据从 MOSI 脚移出发送到从机的 MOSI 脚，同时从机 SPDAT 寄存器的数据从 MISO 脚移出发送到主机的 MISO 脚。

传输完一个字节后，SPI 时钟发生器停止，传输完成标志（ SPIF）置位，如果 SPI 中断使能则会产生一个 SPI 中断。主机和从机 CPU 的两个移位寄存器可以看作是一个 16 位循环移位寄存器。当数据从主机移位传送到从机的同时，数据也以相反的方向移入。这意味着在一个移位周期中，主机和从机的数据相互交换。

4. 硬件设计

4.1. 外部存储器模块接口电路

为了方便扩展应用，开发板上设计了一个 6 芯的外扩存储器接口，其电路如下图所示。该接口可以接入艾克姆科技的 W25Q128 存储器模块、FM25CL64B 铁电存储器模块和 TF 卡模块。

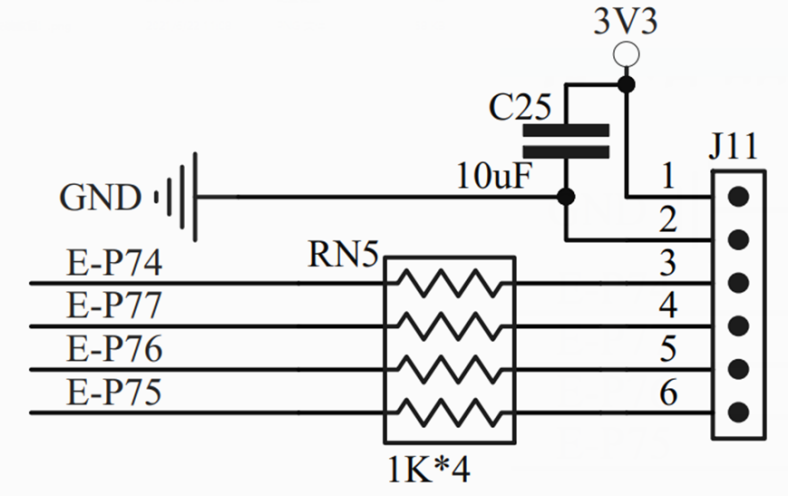


图 6：外部存储器接口

4.2. W25Q128（Flash）存储器模块

W25Q128（Flash）存储器模块是一款 3.3V 单电源供电、存储空间为 128Mbit（16M 字节）的串行 Flash 存储器模块，使用的存储器芯片型号为 W25Q128。

W25Q128 是华邦公司（Winbond）推出的串行 NOR Flash 系列存储器中的一员，该系列还有 W25Q80/16/32/64 等，W25Q128 名称的意义如下：

- W：华邦公司（Winbond）。
- 25Q：SpiFlash 串行 NOR Flash，具有 4KB 扇区，双/四路 I / O。
- 128F：128M-bit。
- V：工作电压范围为（2.7~3.6）V。

W25Q128JV 的容量为 128Mb 共 16MB（注意大写的 B 表示字节，小写的 b 表示位）。W25Q128 将 16M 的容量分为 256 个块（Block），每个块大小为 64KB，每个块又分为 16 个扇区（Sector），每个扇区 4K（4096）字节，每个扇区包含 8 个页（Page），每个页 256 个字节，即 W25Q128 由 65536 个可编程页面构成。

W25Q128 只能按页面编程，也就是每个写操作只能写一个页面，因此，一次最多只能写入 256 个字节数据（从一个页面的起始到结束）。如果将数据写入多个页面，就需要执行多次写操作。

W25Q128 的擦除和编程不一样，擦除时可以按扇区擦除、块擦除和全片擦除。擦除操作最小擦除单位为一个扇区而不是页面，也就是每次至少擦除一个扇区（4K 字节）。

W25Q128 存储器模块的接口为间距 2.54mm 的 6PIN 排针，可以直接安装到开发板的外部存储器接口 J11。模块上设计有电源指示灯，用于指示模块是否正常供电。

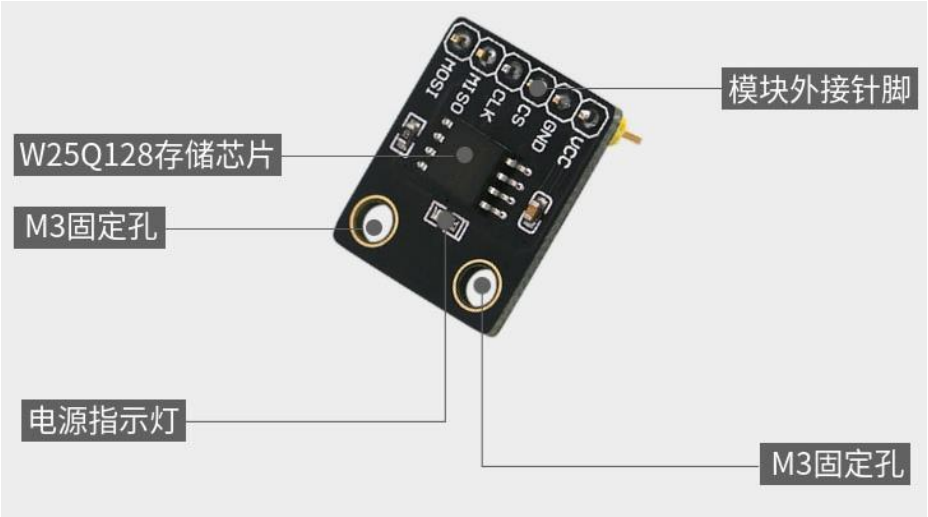


图 7：W25Q128（Flash）存储器模块

W25Q128 存储器模块的参数如下图所示。

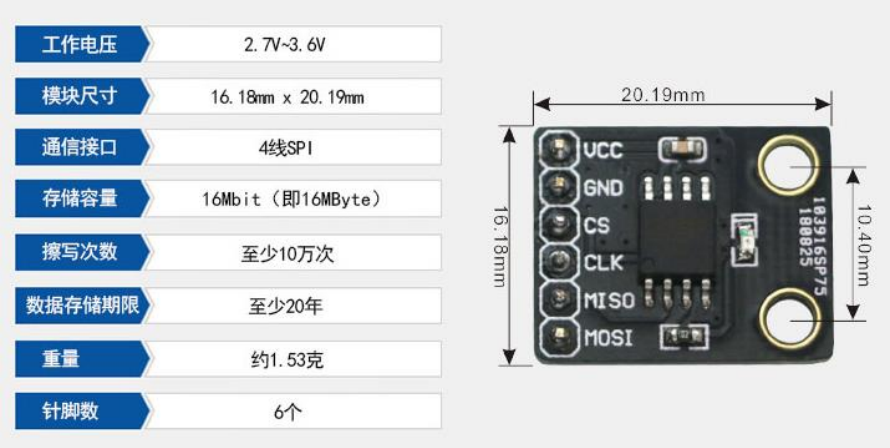


图 8：W25Q128（Flash）存储器模块参数

W25Q128 存储器模块引脚定义如下表所示。

表 5：模块引脚定义

引脚序号	引脚名称	描述
1	VCC	电源正。
2	GND	电源地。
3	CS	SPI 片选。
4	CLK	SPI 时钟。
5	MISO	SPI 主入从出。
6	MOSI	SPI 主出从入。

✧ 注：为了方便读者理解单片机访问 W25Q128 存储器模块的编程，W25Q128 的 SPI 通信时序以及读、写、擦除操作将在软件设计部分讲解。

4.3. FM25CL64B（FRAM）模块

1. FRAM 铁电存储器介绍

FRAM（全称是 Ferroelectric Random Access Memory）铁电存储器，该存储器能兼容 RAM 的一切功能，并且和 ROM 技术一样，是一种非易失性的存储器。可以说铁电存储器在这两类存储类型间搭起了一座跨越沟壑的桥梁：一种非易失性的 RAM。

铁电存储器的工作原理是：当在铁电晶体材料上加入电场，晶体中的中心原子会沿着电场方向运动，达到稳定状态。晶体中的每个自由浮动的中心原子只有 2 个稳定状态，一个记为逻辑中的 0，另一个记为 1。中心原子能在常温、没有电场的情况下，停留在此状态达 100 年以上。铁电存储器不需要定时刷新，能在断电情况下保存数据。由于整个物理过程中没有任何原子碰撞，铁电存储器有高速读写、超低功耗和无限次写入等优点。

说到 FRAM 铁电存储器，就必须要介绍下美国 Ramtron 公司，该公司成立于 1984 年，是一家研究和开发铁电技术用于半导体存储器的公司。2012 年 9 月，Ramtron 公司被美国著名半导体公司赛普拉斯（Cypress）并购。2020 年 4 月，infineon 英飞凌完成了总价值 90

亿欧元（合人民币 693 亿元）对 Cypress 赛普拉斯半导体公司的收购。

Ramtron 公司的 FRAM 主要包括两大类：串行 FRAM 和并行 FRAM。其中串行 FRAM 又分 I2C 总线方式的 FM24xx 系列和 SPI 总线方式的 FM25xx 系列。艾克姆科技 FRAM 选择的存储器芯片是 FM25CL64B 芯片（SPI 总线方式）。

✧ 注：Ramtron 公司的商业 FRAM 产品全部由美国和日本的战略代工厂所制造。

FRAM 的主要特点如下：

- 非易失性：写入的数据掉电不会丢失。
- 高读写耐久性：FRAM 保证最多 10 万亿次写入，远远超过 EEPROM 的写入次数。
- 写入速度快：EEPROM 和 Flash 写入数据之前都需要进行耗时的擦除操作，而 FRAM 不需要擦除，可以直接覆盖写入，这会节省大量的时间。另外，FRAM 自身完成写操作的时间极短，程序中执行写操作时，甚至无需“判忙”。
- 低功耗：一方面，FRAM 自身功耗低，另一方面，FRAM 写入速度快，相对于 EEPROM，写入同样数量的数据所需的时间更少，消耗的电流也会更少。

看到这里，读者可能会有疑惑，既然 FRAM 在性能上碾压 EEPROM 和 Flash，为什么 EEPROM 和 Flash 仍在大量应用，而没有全部被 FRAM 取代？这是因为 FRAM 的制造成本高，价格比 EEPROM、FLASH 更加昂贵，产品设计时，不仅仅需要考虑性能，也需要考虑成本。另外 FRAM 的储存空间更小，如 FM25CL64B 的存储空间为 8K 字节，而 W25Q128 Flash 存储器的存储空间为 16M 字节，因此，应用中需要存储较大数据时，通常会选择 Flash 存储器。

2. FM25CL64B 铁电存储器模块

艾克姆科技的铁电存储器模块使用的 FRAM 芯片是 FM25CL64B，FM25CL64B 是非易失性的，并且像 RAM 一样执行读写，其特性如下。

- 工作电压范围：（2.7~3.65）V。
- 待机电流典型值为 3uA。
- 1MHz 时的工作电流为 200 μA。
- 100 万亿（10^{14}）次读/写。
- 38 年的数据保存时间。
- NoDelay™写操作。
- 频率高达 20 MHz。
- 支持 SPI 通信模式 0 和 3。

和 W25Q128 存储器模块一样，FRAM 模块的接口也是间距 2.54mm 的 6PIN 排针，同

样可以直接安装到开发板的外部存储器接口 J11 上。模块上同样设计有电源指示灯，用于指示模块是否正常供电。

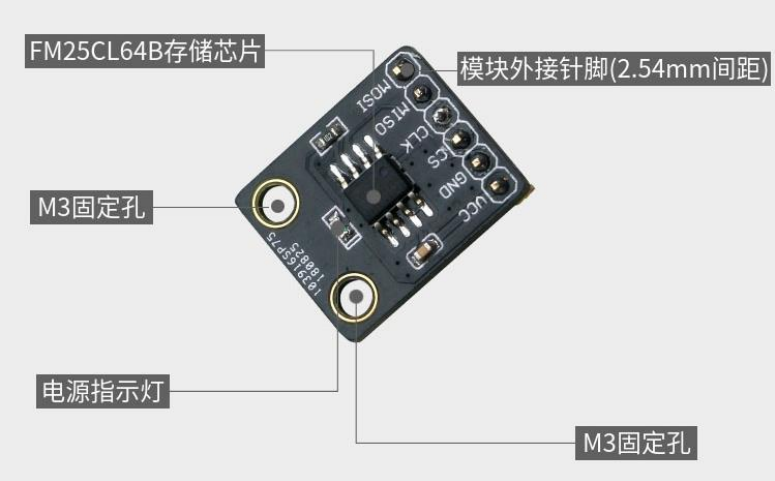


图 9：FM25CL64B 铁电存储器模块

FM25CL64B 存储器模块的参数如下图所示。

图 10：FM25CL64B 铁电存储器模块参数

FM25CL64B 铁电存储器模块引脚定义如下表所示。

表 6：模块引脚定义

引脚序号	引脚名称	描述
1	VCC	电源正。
2	GND	电源地。
3	CS	SPI 片选。
4	CLK	SPI 时钟。
5	MISO	SPI 主入从出。
6	MOSI	SPI 主出从入。

✧ 注：为了方便读者理解单片机访问 FM25CL64B 铁电存储器模块的编程，FM25CL64B 的 SPI 通信时序以及读、写操作将在软件设计部分讲解。

5. 软件设计

5.1. 硬件 SPI 读写 W25Q128 存储器实验

✧ 注：本节的实验是在“实验 2-6-1：串口 1 数据收发实验”的基础上修改，本节对应的实验源码是：“实验 2-13-1：硬件 SPI 读写 W25Q128 存储器”。

5.1.1. 实验内容

将 STC8A8K64D4 单片机的 SPI 配置为主机，通过 SPI 总线访问 W25Q128 存储器，完成以下操作。

- 读取 W25Q128 芯片 ID：通过读芯片 ID 可以判断芯片类型以及判断 W25Q128 是否正确接入。
- 扇区擦除：擦除一个指定的扇区。
- 全片擦除：擦除整个芯片。
- 页编程：向指定页面连续写入不超过页面地址范围的数据。
- 批量编程：向指定地址连续写入指定长度数据的功能，该功能实现了跨页写入。
- 批量读：从指定地址连续读取指定长度数据，并将读取的数据通过串口输出。

✧ 注：本节的实验需要使用艾克姆科技的 W25Q128 存储器模块。

5.1.2. 代码编写

1. 新建一个名称为“w25q128.c”的文件及其头文件“w25q128.h”并保存到工程的“Source”文件夹，并将“w25q128.c”加入到 Keil 工程中的“SOURCE”组。
2. 引用头文件

因为在“main.c”文件中使用了“w25q128.c”文件中的函数，所以需要引用下面的头文件“w25q128.h”。

代码清单：引用头文件

```
1.  //引用头文件
2.  #include    "w25q128.h"
```

3. 初始化 SPI

SPI 初始化包含引脚配置、SPI 工作模式配置、传输速率配置、SPI 通信模式配置、收发数据的位序以及中断配置。

1) SPI 引脚配置

本例中，SPI 连接 W25Q128FV 所用的引脚如下表所示。

表 7：SPI 连接 W25Q128FV 引脚分配

名称	引脚	说明
SS	P7.4	SPI 片选信号，连接到 W25Q128 模块的 CS 引脚。
MOSI	P7.5	SPI 时钟信号，连接到 W25Q128 模块的 MOSI 引脚。
MISO	P7.6	SPI 主入从出，连接到 W25Q128 模块的 MISO 引脚。

SCLK	P7.7	SPI 主出从入，连接到 W25Q128 模块的 CLK 引脚。

2) SPI 模式：主机。
3) SPI 传输速率：6Mbps。
4) SPI 通信模式：模式 0。
5) SPI 收发数据的位序：高位在前（MSB），这是因为 W25Q128 存储器要求访问他的 SPI 主机发送数据时遵循 MSB。
6) 中断：本例中没有使用中断，采用的是查询方式收发数据。

对应的初始化代码清单如下。

代码清单：SPI 初始化

```
/*********************************************************************************
 * 描  述 : SPI 初始化
 * 参  数 : 无
 * 返回值 : 无
 *********************************************************************************/
void spi_init(void)
{
    //设置 P7.4~P7.7 为准双向口，其中 P7.5 P7.6 P7.7 这 3 个引脚将作为 SPI 的 MOSI MISO SCLK 信号，P7.4 将
    //作为 SPI 的片选信号
    P7M1 &= 0x0F;  P7M0 &= 0x0F;
    SPI_CS_HIGH; //拉高 SPI 片选引脚，不选择从机（此时，连接的从机从 SPI 总线上断开）
    //设置 SPI_S[1:0]的值为[1 0]，使用 P7.5 P7.6 P7.7 作为 SPI 的 MOSI MISO SCLK 信号
    P_SW1 &= ~0x0C;
    P_SW1 |= 0x08;
    /*-------------------------------------------------------------------
    SSIG  SPEN  DORD  MSTR  CPOL  CPHA  SPR[1,0]
    1      1     0     1     0     0     00
    SSIG=1：忽略 SS 引脚功能，使用 MSTR 确定器件是主机还是从机
    SPEN=1：使能 SPI
    DORD=0：发送/接收的顺序为高位在前
    MSTR=1：SPI 为主机
    CPOL=0，CPHA=0：SPI 工作于模式 0
    SPR[1,0]=00：SPI 时钟频率选择：SYSclk/4
    -------------------------------------------------------------------*/
    SPCTL = 0xD0;
    //清零 SPI 中断标志位和写冲突标志位
    SPSTAT = SPIF | WCOL;
}
```

4. SPI 数据传输

SPI 是按照字节来逐个发和收数据的，当然，一次 SPI 操作可以发送多个字节数据。下面的代码是我们封装的 SPI 传输函数，用于发/收一个字节数据。注意，这个函数是给其他

SPI 操作函数调用的，因为函数中不能操作片选信号。

代码清单：SP 数据收发函数

```
/*************************************************************************************
 * 描  述 : SPI 发送一个字节数据，并返回一个字节数据
 * 参  数 : dat[in]：待写入的数据
 * 返回值 : SPI 的 MISO 返回的数据
 *************************************************************************************/
static u8 Spi_WriteOneByte(u8 dat)
{
    SPDAT = dat;                //触发 SPI 发送数据
    while (!(SPSTAT & SPIF));   //等待发送完成
    SPSTAT = SPIF | WCOL;       //清除 SPI 状态位
    return SPDAT;               //返回 SPI 数据
}
```

5. 读取 W25Q128FV 芯片 ID

W25Q128 芯片的 ID 固定为 0xEF17，我们可以通过读取 ID 判断 W25Q128 是否在线，读取 ID 的时序如下图所示。读取 ID 执行的操作如下。

- 拉低片选信号 CS，使能 W25Q128。
- 发送扇区擦除命令 0x90。
- 发送 24 位地址（读取 ID 时固定为 0x000000），高地址在前。
- 发送 2 个字节数据（0xFF）读取 ID，读取的 ID 是接收数组的最后 2 个字节。
- 拉高片选信号 CS，释放 W25Q128。

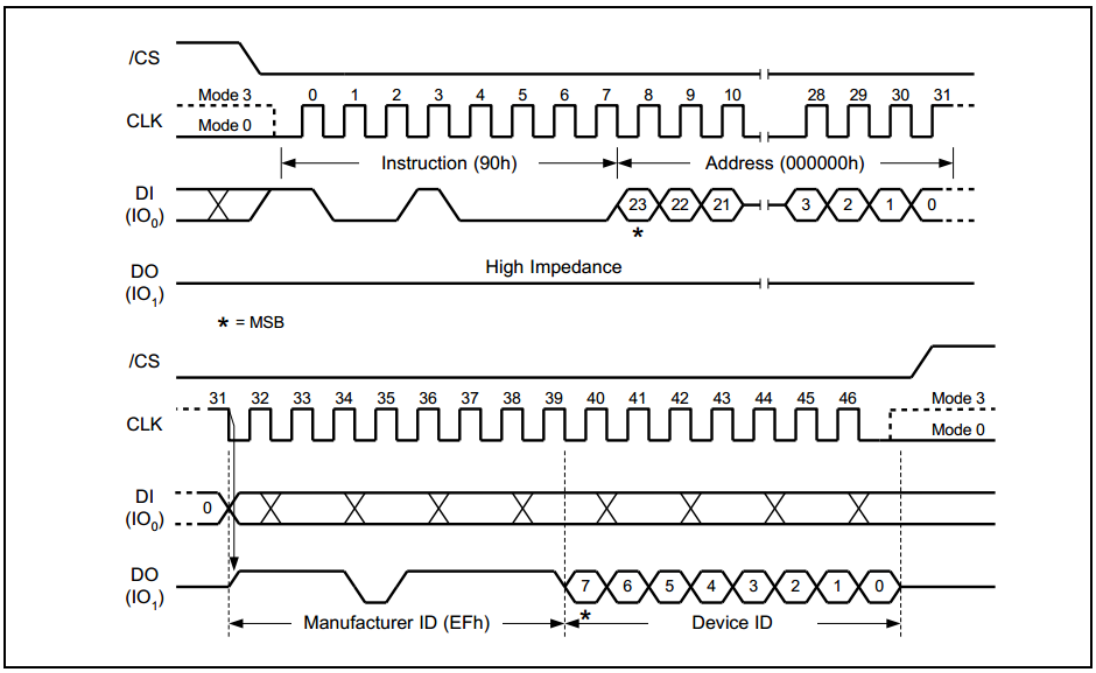


图 11：读取 ID 时序

读取 ID 的函数代码清单如下：

代码清单：读取 W25Q128 的 ID

```
/*******************************************************************************
 * 描  述 : 读取 W25Q128 芯片的 ID，W25Q128 的 ID:0xEF17，另：W25Q16 的 ID:0xEF14  W25Q32 的
 *             ID:0xEF15  W25Q64 的 ID:0xEF16
 * 参  数 : 无
 * 返回值 : id：芯片的 ID
*******************************************************************************/
u16 W25Q_Spi_ReadID(void)
{
    u16 id = 0;
    u8   mf_id,dev_id;

    SPI_CS_LOW;                          //片选拉低，使能从机
    (void)Spi_WriteOneByte(W25_ReadID); //发送读取 ID 指令
    (void)Spi_WriteOneByte(0x00);        //发送 24 位地址，读 ID 时，24 位地址为 0x000000
    (void)Spi_WriteOneByte(0x00);
    (void)Spi_WriteOneByte(0x00);
    mf_id =Spi_WriteOneByte(0xFF);       //读取 MANUFACTURER ID(MF7-MF0)，值应为：0xEF
    dev_id =Spi_WriteOneByte(0xFF);      //读取 Device ID(ID7-ID0)，值应为：0x17
    id = mf_id*256 + dev_id;             //将读取的 MANUFACTURER ID 和 Device ID 合并为 16 位
    SPI_CS_HIGH;                         //片选拉高，断开从机
    return id;                           //返回读取的 ID
}
```

6. 擦除扇区：擦除指定的扇区

■ **Flash 为什么要擦除？**

因为 Flash 的编程原理都是只能将各个 bit 由 1 写为 0，而不能将 0 写为 1，因此在 Flash 编程之前，为了保证写入的正确性，必须将对应的扇区擦除，擦除操作会将该扇区的内容全部恢复为 0xFF，这样执行写入操作就可以正确执行了。

W25Q128 支持扇区擦除、块擦除和全片擦除，W25Q128 的最小擦除单位为一个扇区，也就是每次至少擦除 4K 字节。我们在操作 Flash 的时候，要特别注意 Flash 编程时间和擦除时间，尤其是擦除时间，因为这些操作通常用时较长，程序中如果处理不好的话，可能会导致程序运行堵塞。W25Q128 编程时间和擦除时间如下表所示。

表 8：W25Q128 编程和擦除时间

描述	符号	规格			单位
		最小值	典型值	最大值	
字节编程时间 (第一个字节) （注 1）	t_{BP1}	—	30	50	微秒
另外的字节编程时间 (第一个字节后) （注 1）	t_{BP2}	—	2.5	12	微秒
页编程时间	t_{PP}	—	0.7	3	毫秒
块擦除时间(4KB)	t_{SE}	—	100	400	毫秒

			45		
块擦除时间(32KB)	t_{BE1}	—	120	1600	毫秒
块擦除时间(64KB)	t_{BE2}	—	150	2000	毫秒
全片擦除时间	t_{CE}	—	40	200	秒

注 1：同一个页面内多个字节编程的时间为：tBPN = t_{BP1}+ t_{BP2} * N (max)，N=编程的字节数。

W25Q128FV 扇区擦除时序如下图所示，扇区擦除执行的操作如下，注意擦除扇区之前必须先发送“写使能”命令开启 W25Q128FV 的写使能。

- 拉低片选信号 CS，使能 W25Q128FV。
- 发送扇区擦除命令 0x20。
- 发送扇区 24 位地址，高地址在前。
- 拉高片选信号 CS，释放 W25Q128FV。

命令发送完成并不表示 W25Q128FV 已经执行完成扇区擦除操作，因此命令发送完后还需要通过查询状态寄存器的 BUSY 位来判断擦除操作是否完成，当 BUSY 位的值为 0 时表示操作完成，W25Q128FV 就绪（扇区擦除完成）。

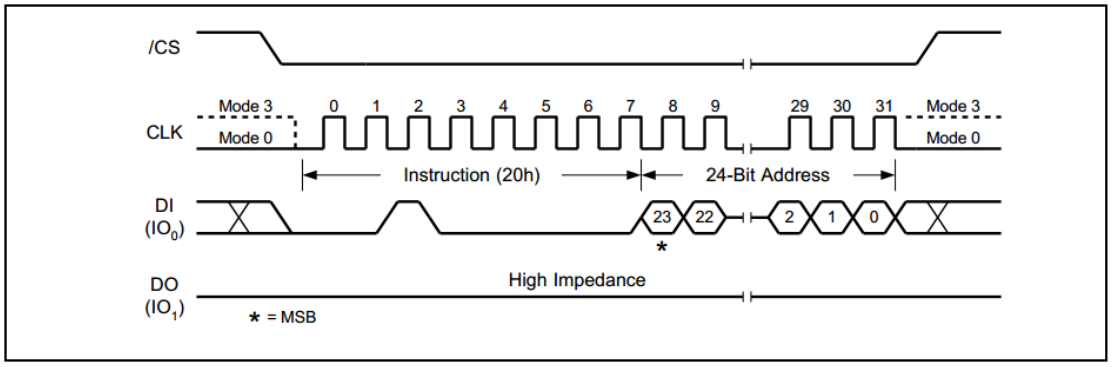


图 12：擦除扇区时序

根据 W25Q128 扇区擦除时序，编写代码如下：

代码清单：扇区擦除

```
1.  /*****************************************************************************************
2.   * 描   述 : 擦除一个扇区，W25Q128 最小的擦除单位是扇区.擦除一个扇区所需时间典型值为 45ms，最大值为
3.               400ms
4.   * 参   数 : SecAddr[in]：扇区地址
5.   * 返回值 : 无
6.  *****************************************************************************************/
7.  void W25Q_Erase_Sector(u32 SecAddr)
8.  {
9.     while(W25Q_Spi_ReadStatus()&0x01);      // 判断是否忙
10.    W25Q_WriteEnable();                     // 写允许
11.    SPI_CS_LOW;                             //片选拉低，使能从机
12.    Spi_WriteOneByte(W25_SectorErase);      // 写之前先擦除
13.    Spi_WriteOneByte((u8)(SecAddr>>16));    //先发送 24 位地址的高 8 位
14.    Spi_WriteOneByte((u8)(SecAddr>>8));     //再发送 24 位地址的中间 8 位
```

```
    Spi_WriteOneByte((u8)SecAddr);          //最后发送 24 位地址的低 8 位
    SPI_CS_HIGH;                            //片选拉高，断开从机
    while(W25Q_Spi_ReadStatus()&0x01);      // 等待擦除操作完成
}
```

7. 块擦除：擦除指定的块

块擦除指令将指定块（64K 字节）内的所有内存设置为 0xFF 的擦除状态。在 W25Q128 接受块擦除指令之前，必须先发送“写使能”命令开启 W25Q128 的写使能。W25Q128 块擦除时序如下图所示。

- 拉低片选信号 CS，使能 W25Q128。
- 发送块擦除命令 0xD8。
- 发送扇区 24 位地址，高地址在前。
- 拉高片选信号 CS，释放 W25Q128。

命令发送完成并不表示 W25Q128 已经执行完成扇区擦除操作，因此命令发送完后还需要通过查询状态寄存器的 BUSY 位来判断擦除操作是否完成，当 BUSY 位的值为 0 时表示操作完成，W25Q128 就绪（块擦除完成）。

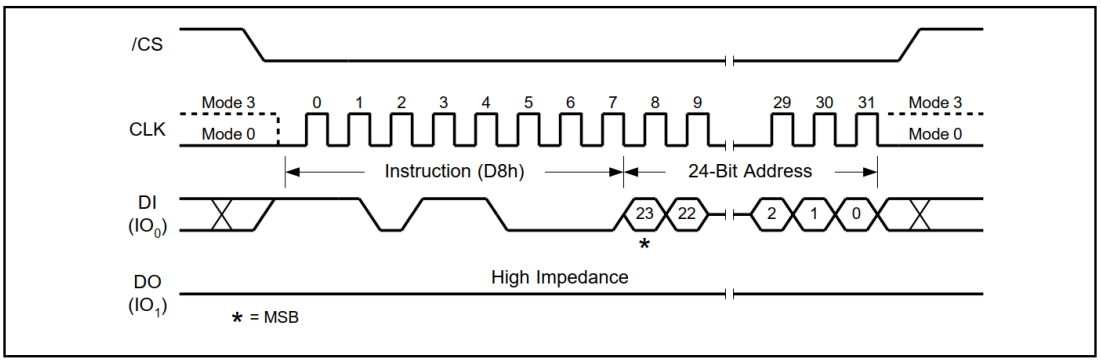


图 13：擦除块时序

根据 W25Q128 块擦除时序，编写的代码清单如下：

代码清单：块擦除

```
/*****************************************************************************************
 * 描  述 : 擦除一个块(64K)
 * 参  数 : BlockAddr[in]：块地址
 * 返回值 : 无
*****************************************************************************************/
void W25Q_Erase_Block(u32 BlockAddr)
{
    while(W25Q_Spi_ReadStatus()&0x01);      //判断是否忙
    W25Q_WriteEnable();                     //写允许
    SPI_CS_LOW;                             //片选拉低，使能从机
    Spi_WriteOneByte(W25_BlockkErase);      //写入扇区擦除命令
    Spi_WriteOneByte((u8)(BlockAddr>>16));  //先发送 24 位地址的高 8 位
    Spi_WriteOneByte((u8)(BlockAddr>>8));   //再发送 24 位地址的中间 8 位
    Spi_WriteOneByte((u8)BlockAddr);        //最后发送 24 位地址的低 8 位
```

```
    SPI_CS_HIGH;                        //片选拉高，断开从机
    while(W25Q_Spi_ReadStatus()&0x01);  //等待擦除完成
}
```

8. 全片擦除：擦除整个芯片

W25Q128 全片擦除时序如下图所示，全片擦除执行的操作如下。注意全片擦除之前必须先发送“写使能”命令开启 W25Q128 的写使能。

- 拉低片选信号 CS，使能 W25Q128。
- 发送扇区擦除命令 0xC7。
- 拉高片选信号 CS，释放 W25Q128。

全片擦除花费时间较长，典型时间是 40 秒，命令发送完后需要查询状态寄存器的 BUSY 位，直到 BUSY 位的值为 0（全片擦除完成，W25Q128 就绪）才可以执行其他操作。

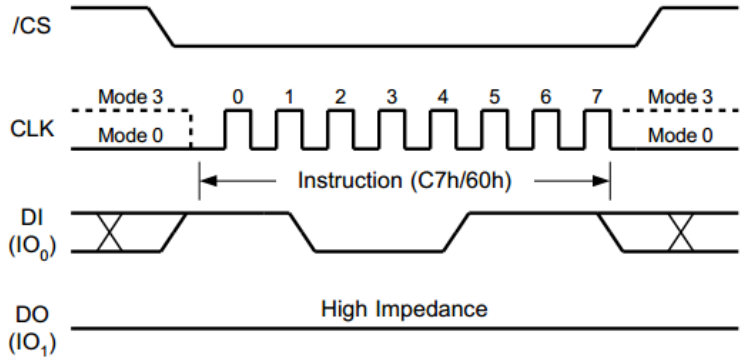


图 14：全片擦除时序

根据 W25Q128 全片擦除时序，编写的代码清单如下：

代码清单：全片擦除

```
/*****************************************************************************
 * 描   述 : 全片擦除 W25Q128，全片擦除所需的时间典型值为：40 秒
 * 参   数 : 无
 * 返回值 : 无
*****************************************************************************/
void W25Q_Erase_Chip(void)
{
    while(W25Q_Spi_ReadStatus()&0x01);      //判断是否忙
    W25Q_WriteEnable();                     //写允许
    SPI_CS_LOW;                               //片选拉低，使能从机
    Spi_WriteOneByte(W25_ChipErase);        //写入全片擦除命令
    SPI_CS_HIGH;                            //从 CS=1 时开始执行擦除
    while(W25Q_Spi_ReadStatus()&0x01);      //等待擦除完成
}
```

9. 按页写：向指定页面连续写入不超过页面地址范围的数据

W25Q128FV 扇区擦除时序如下图所示，扇区擦除执行的操作如下，注意擦除扇区之前必须先发送“写使能”命令开启 W25Q128FV 的写使能。

- 拉低片选信号 CS，使能 W25Q128FV。
- 发送页编程命令 0x02。
- 发送 24 位地址，高地址在前。
- 发送写入到 Flash 的数据，注意最大写入的长度不能超过该地址所处页面的剩余空间。
- 拉高片选信号 CS，释放 W25Q128FV。

页编程时，数据传输完成并不表示 W25Q128FV 已经将接收的数据写入到自身的 Flash 内，因此数据传输完后还需要通过查询状态寄存器的 BUSY 位来判断编程是否完成，当 BUSY 位的值为 0 时表示编程完成，W25Q128FV 就绪。

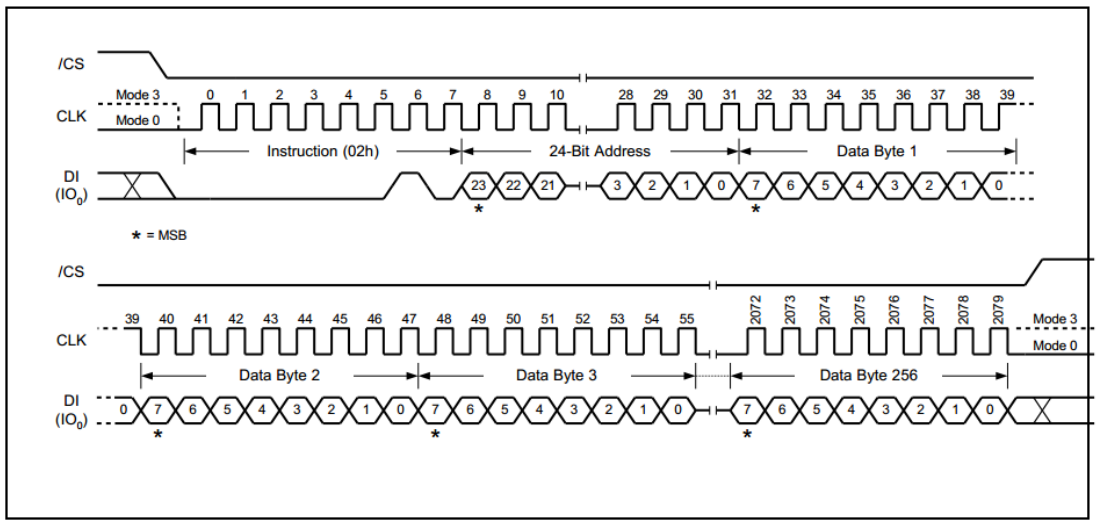


图 15：页编程时序

根据 W25Q128 页编程时序，代码清单如下：

代码清单：页编程

```
/*****************************************************************************************
 * 描   述：  向指定的地址写入数据,最大写入的长度不能超过该地址所处页面的剩余空间
 * 参   数：  *pBuffer[in]:指向待写入的数据
 *            WriteAddr[in]:写入的起始地址
 *            w_size[in]:写入的字节数
 * 返回值：  无
*****************************************************************************************/
void W25Q_Write_Page(u8 *pBuffer, u32 WriteAddr, u32 w_size)
{
    u16 i;
    while(W25Q_Spi_ReadStatus()&0x01);          //判断是否忙
    W25Q_WriteEnable();                         //写使能
    SPI_CS_LOW;                                 //使能器件
    Spi_WriteOneByte(W25_PageProgram);          //发送写页命令
    Spi_WriteOneByte((u8)((WriteAddr)>>16));  //发送 24bit 地址
    Spi_WriteOneByte((u8)((WriteAddr)>>8));
    Spi_WriteOneByte((u8)WriteAddr);
```

```
    for(i=0;i<w_size;i++)                    //循环写入数据
    {
        Spi_WriteOneByte(*pBuffer++);
    }
    SPI_CS_HIGH;                             //取消片选
    while(W25Q_Spi_ReadStatus()&0x01);       //等待 W25Q128 自身编程完成
}
```

10. 批量编程：向指定地址连续写入指定长度数据的功能，该功能实现了跨页写入。

批量编程是在页编程的基础上，将编程的数据拆分进行多次页编程，从而实现跨页编程，也就是实现从任意地址开始写入任意长度的数据，当然，地址范围和编程的数据长度不能超过 W25Q128 的自身的限制。

代码清单：批量编程

```
/*****************************************************************************
 * 描   述：向指定的地址写入指定大小的数据，支持跨页写入数。注：写入数据的地址所在的页面已经执行过擦除操作
 * 参   数：*pBuffer[in]:指向待写入的数据
 *          WriteAddr[in]:写入的起始地址
 *          w_size[in]:写入的字节数
 * 返回值：无
*****************************************************************************/
void W25Q_Write_Bytes(u8 * pBuffer,u32 WriteAddr,u32 w_size)
{
    u32 PageByteRemain = 0;
    //计算起始地址所处页面的剩余空间
    PageByteRemain = W25Q128_PAGE_SIZE - WriteAddr%W25Q128_PAGE_SIZE;
    //如果编程的数据长度不大于页面的剩余空间，编程数据长度等于 w_size
    if(w_size <= PageByteRemain)
    {
        PageByteRemain = w_size;
    }
    //开始编程，直到所有的数据编程完成
    while(1)
    {
        //编程 PageByteRemain 个字节
        W25Q_Write_Page(pBuffer,WriteAddr,PageByteRemain);
        //如果起始地址所处页面的剩余空间足够存放写入的数据，执行写入后，写入结束，退出循环
        if(PageByteRemain == w_size)break;
        else
        {
            //取数据的地址增加 PageByteRemain
            pBuffer += PageByteRemain;
            //计算编程写入的地址
```

```
            WriteAddr += PageByteRemain;
            //剩余待编程的数据大小
            w_size -= PageByteRemain;
            //计算下次编程的数据长度
            if(w_size > W25Q128_PAGE_SIZE)
            {
               PageByteRemain = W25Q128_PAGE_SIZE;
            }
            else
            {
               PageByteRemain = w_size;
            }
         }
      }
}
```

11. 批量读：从指定地址连续读取指定长度数据，并将读取的数据通过串口输出

W25Q128 支持连续读任意长度数据，甚至可以通过读命令一次将整个 Flash 内容读取出来，但是通常单片机内存无法存储这么大的数据，因此批量读取的时候往往也是分次读取的，读取的代码清单如下。

代码清单：批量读出数据

```
/*****************************************************************************************
 * 描   述：  从指定的起始地址连续读取指定长度的数据
 * 参   数：  *pBuffer[in]:指向保存读出数据的缓存的起始地址
 *           ReadAddr[in]:读取数据的起始地址
 *           r_size[in]:读取的字节数
 * 返回值 : 无
*****************************************************************************************/
void W25Q_Read_Bytes(u8 * pBuffer,u32 ReadAddr,u32 r_size)
{
   u32 i;
   while(W25Q_Spi_ReadStatus()&0x01);        //判忙
   SPI_CS_LOW;                               //使能器件
   Spi_WriteOneByte(W25X_ReadDATA8);         //发送读取命令
   Spi_WriteOneByte((u8)((ReadAddr)>>16));   //发送 24bit 地址
   Spi_WriteOneByte((u8)((ReadAddr)>>8));
   Spi_WriteOneByte((u8)ReadAddr);
   for(i=0;i<r_size;i++)                     //从地址地地址开始逐字节读出指定大小的数据
   {
      *pBuffer++=Spi_WriteOneByte(0xFF);
   }
   SPI_CS_HIGH;                              //取消片选
}
```

12. 主函数

主函数中初始化指示灯、按键和 SPI 等外设，接着读取 W25Q128 的 ID，以此判断 W25Q128 模块是否正确安装到开发板。如果读取的 ID 数值为 0xEF17，说明 W25Q128 模块安装正确，程序继续向下运行，否则，闪烁指示灯 D1 指示未检测到 W25Q128。

程序进入主循环后，扫描按键状态，并根据按键状态执行以下动作。

- KEY1 按下：按页写入，扇区 0 的第一页写入 256 个字节，之后读出数据并通过串口输出。
- KEY2 按下：向起始地址 100 连续写入 500 个字节数据，之后读出数据并通过串口输出。这里演示了跨页的批量写，写入的数据占用了 2 个页面。
- KEY3 按下：全片擦除 W25Q128。

代码清单如下：

代码清单：主函数

```
/*************************************************************************
功能描述：主函数
入口参数：无
返 回 值：int 类型
*************************************************************************/
int main(void)
{
    u8 btn_val;
    u16 i;
    u8 j = 0;
    u16 chip_id,test_len;

    P2M1 &= 0x3F;   P2M0 &= 0x3F;     //设置 P2.6~P2.7 为准双向口（指示灯 D1 和 D2）
    P7M1 &= 0xF9;   P7M0 &= 0xF9;     //设置 P7.1~P7.2 为准双向口（指示灯 D4 和 D3）
    P3M1 &= 0x3F;   P3M0 &= 0x3F;     //设置 P3.6~P3.7 为准双向口（按键 KEY2 和 KEY1）
    P0M1 &= 0x5F;   P0M0 &= 0x5F;     //设置 P0.5，P0.7 为准双向口（按键 KEY4 和 KEY3）
    P3M1 &= 0xFE;   P3M0 &= 0xFE;     //设置 P3.0 为准双向口（串口 1 的 RxD）
    P3M1 &= 0xFD;   P3M0 |= 0x02;     //设置 P3.1 为推挽输出（串口 1 的 TxD）

    uart1_init();                   //串口 1 初始化
    spi_init();                     //初始化 SPI
    delay_ms(500);                  //初始化后适当延时
    EA = 1;                         //使能总中断

    //通过读取 w25q128 芯片的 ID 判断 w25q128 模块是否正确安装到开发板,若读取 ID 不成功，指示灯 D1 持续闪烁
    //提示读取 ID 失败
    chip_id = W25Q_Spi_ReadID();

```

```
29.     while(chip_id != W25Q128_ID)
30.     {
31.         led_toggle(LED_1);
32.         delay_ms(1500);
33.     }
34.     printf("CHIP ID: %04hX\r\n",chip_id);    //串口打印出读取的 ID
35.
36.     while(1)
37.     {
38.         btn_val=buttons_scan(0);              //获取开发板用户按键检测值，不支持连按
39.         //按下 KEY1：测试按页写入。扇区 0 的第一页写入 256 个字节，之后读出数据并通过串口输出
40.         if(btn_val == BUTTON1_PRESSED)
41.         {
42.             test_len = 256;
43.             //写之前需要先执行擦除操作
44.             W25Q_Erase_Sector(0);
45.             for(i=0;i<test_len;i++)my_tx_buf[i] = j++;
46.             //写入 256 个字节数据，使用按页写入时，一次写入的数据长度不超过一个页面的剩余空间
47.             W25Q_Write_Page(my_tx_buf,0,test_len);
48.             //读取写入的数据
49.             W25Q_Read_Bytes(my_rx_buf,0,test_len);
50.             //串口打印读取的数据
51.             for(i=0;i<test_len;i++)printf("%02bX ",my_rx_buf[i]);
52.             //翻转指示灯 D1 状态，指示操作完成
53.             led_toggle(LED_1);
54.         }
55.         //按下 KEY2：批量写入和读出,向起始地址 100 连续写入 500 个字节数据，之后读出数据并通过串口输出
56.         else if(btn_val == BUTTON2_PRESSED)
57.         {
58.             test_len = 512;
59.             for(i=0;i<test_len;i++)my_tx_buf[i] = j++;
60.             //写入 256 个字节数据，使用按页写入时，一次写入的数据长度不超过一个页面的剩余空间
61.             W25Q_Write_Bytes(my_tx_buf,100,test_len);
62.             //读取写入的数据
63.             W25Q_Read_Bytes(my_rx_buf,100,test_len);
64.             //串口打印读取的数据
65.             for(i=0;i<test_len;i++)printf("%02bX ",my_rx_buf[i]);
66.             led_toggle(LED_2);                //控制用户指示灯 D2 翻转
67.         }
68.         //按下 KEY3：全片擦除 W25Q128，全片擦除所需的时间典型值为：40 秒
69.         else if(btn_val == BUTTON3_PRESSED)
70.         {
71.             led_on(LED_3);                //点亮指示灯 D3
```

```
72.             W25Q_Erase_Chip();          //全片擦除
73.             led_off(LED_3);             //熄灭指示灯 D3
74.         }
75.     }
76. }
```

5.1.3. 硬件连接

本实验需要使用 LED 指示灯、按键和艾克姆科技的 W25Q128 存储器模块，按照下图所示安装 W25Q128 存储器模块和短接跳线帽。

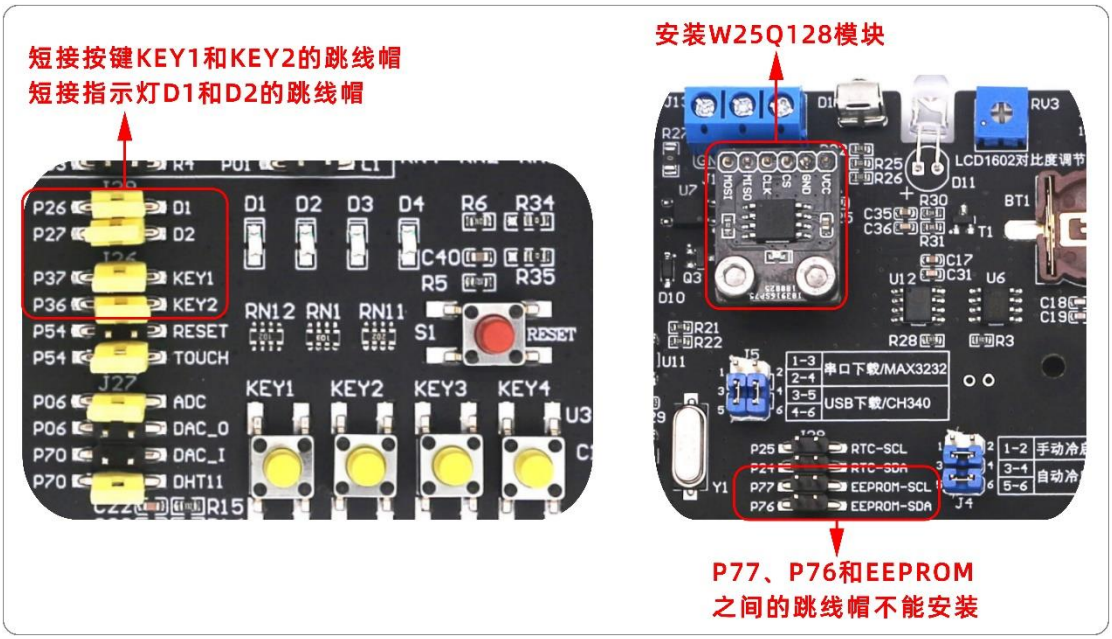


图 16：跳线帽短接

5.1.4. 实验步骤

1) 解压“…\第 3 部分：配套例程源码”目录下的压缩文件“实验 2-13-1：硬件 SPI 读写 W25Q128 存储器”，将解压后得到的文件夹拷贝到合适的目录，如“D\STC8”（这样做的目的是为了防止中文路径或者工程存放的路径过深导致打开工程出现问题）。
2) 双击“…\spi_w25q128\project”目录下的工程文件“spi_w25q128.uvproj”。
3) 点击编译按钮编译工程，编译成功后生成的 HEX 文件“spi_w25q128.hex”位于工程的“…\spi_w25q128\Project\Object”目录下。
4) 打开 STC-ISP 软件下载程序，下载使用内部 IRC 时钟，IRC 频率选择：24MHz。
5) 电脑上打开串口调试助手，选择开发板对应的串口号，将波特率设置为 9600bps。程序运行后，分别按下 KEY1、KEY2 和 KEY3 按键执行读写和擦除 W25Q128。

- 按下 KEY1 按键：按页编程，先擦除扇区 0，接着向扇区 0 的第一页写入 256 个字节，之后读出数据并通过串口输出。

图 17：串口接收的数据

- 按下 KEY2 按键：批量写入和读出，向起始地址 100 连续写入 500 个字节数据，之后读出数据并通过串口输出。这里演示了跨页的批量写，写入的数据占用了 2 个页面。需要注意的是，批量写函数没有对所要写入的扇区进行擦除，读者在调用该函数前，要确定所写的内容占用的扇区是擦除过的。

图 18：串口接收的数据

- 按下 KEY3 按键：全片擦除 W25Q128，按下按键后指示灯 D3 点亮，指示灯开始全片擦除 W25Q128 存储器，擦除完成后指示灯 D3 熄灭，注意，全片擦除所需的时间较长，典型值为：40 秒。

5.2. 模拟 SPI 读写 W25Q128 存储器实验

✧ 注：本节的实验是在“实验 2-13-1：硬件 SPI 读写 W25Q128 存储器”的基础上修改，本节对应的实验源码是：“实验 2-13-2：模拟 SPI 读写 W25Q128 存储器”。

5.2.1. 实验内容

本实验实现的功能和“实验 2-13-1：硬件 SPI 读写 W25Q128 存储器”完全一样，区别

是本实验中不使用 STC8A8K64D4 单片机集成的硬件 SPI 外设，而是使用 GPIO 模拟 SPI 的协议时序完成 SPI 通信。

- ✧ 注 1：相对于硬件 SPI，模拟 SPI 通信时，CPU 需要频繁地操作 IO，会占用更多的 CPU 资源，因此建议开发程序时优先考虑使用硬件 SPI，当硬件 SPI 不够用时，再考虑使用模拟 SPI。
- ✧ 注 2：本节的实验需要使用艾克姆科技的 W25Q128 存储器模块。

5.2.2. 代码编写

模拟 SPI 的程序相对于硬件 SPI 的程序来说，主要修改点包括以下几个方面：

1) 定义模拟 SPI 的信号引脚和相关的操作宏。
2) SPI 初始化函数：初始化函数无需再初始化 SPI 的寄存器，而是配置 SPI 所用的 GPIO。
3) SPI 收发函数：模拟 SPI 时，我们习惯将发和收分开来操作，函数中通过操作 GPIO 来模拟 SPI 的时序实现 SPI 的发和收。

接下来，我们看一下具体的代码实现。

1. 定义模拟 SPI 的信号引脚和相关的操作宏

本例中，我们同样使用 P7.4、P7.5、P7.6 和 P7.7 分别做为 SPI 的 SS、MOSI、MISO 和 SCLK 信号引脚，如下表所示。模拟 SPI 配置引脚时需要注意引脚的方向，STC8A8K64D4 单片机的 GPIO 支持配置为“准双向口”，所以我们在后面的 SPI 初始化函数中我们统一将 4 个引脚全部配置为准双向即可。

表 9：模拟 SPI 引脚

名称	引脚	方向
SS	P7.4	输出。
MOSI	P7.5	输出。
MISO	P7.6	输入。
SCLK	P7.7	输出。

SPI 信号引脚定义的代码清单如下，另外，为了程序操作方便，我们还定义了片选、时钟信号和 MOSI 引脚输出高电平和低电平的宏。

代码清单：定义 SPI 信号所用的引脚

```
1.  sbit    SPI_CS_PIN      = P7.4;         //定义 CS
2.  sbit    SPI_MOSI_PIN    = P7^5;         //定义 MOSI
3.  sbit    SPI_MISO_PIN    = P7^6;         //定义 MISO
4.  sbit    SPI_SCK_PIN     = P7^7;         //定义 SCK
5.
6.  #define  SPI_CS_LOW      SPI_CS_PIN      = 0  //SPI 片选引脚输出低电平
7.  #define  SPI_CS_HIGH     SPI_CS_PIN      = 1  //SPI 片选引脚输出高电平
8.  #define  SPI_SCK_LOW     SPI_SCK_PIN     = 0  //SPI 时钟信号引脚输出低电平
9.  #define  SPI_SCK_HIGH    SPI_SCK_PIN     = 1  //SPI 时钟信号引脚输出高电平
```

```
#define  SPI_MOSI_LOW    SPI_MOSI_PIN    = 0  //SPI 主出从入引脚输出低电平
#define  SPI_MOSI_HIGH   SPI_MOSI_PIN    = 1  //SPI 主出从入引脚输出高电平
```

2. SPI 初始化函数

SPI 初始化函数中将 SPI 的 4 个信号所用的 GPIO 全部配置为准双向口，接着拉高片选引脚，不使能 SPI 从机（W25Q128 模块）。

对于时钟信号，要根据使用的 SPI 通信模式设置其状态，本例中使用的是通信模式 0，时钟信号在空闲状态下为低电平，因此，需要将时钟信号对应的 GPIO 拉低。

代码清单：SPI 初始化

```
/*******************************************************************************
 * 描  述 : SPI 初始化
 * 参  数 : 无
 * 返回值 : 无
 *******************************************************************************/
void spi_init(void)
{
    //设置 P7.4~P7.7 为准双向口，其中 P7.5 P7.6 P7.7 这 3 个引脚将作为 SPI 的 MOSI MISO SCLK 信号，P7.4 将
    //作为 SPI 的片选信号
    P4M1 &= 0xF0;  P4M0 &= 0xF0;
    SPI_CS_HIGH; //拉高 SPI 片选引脚，不选择从机（此时，连接的从机从 SPI 总线上断开）
    SPI_SCK_LOW; //拉低时钟线
}
```

3. SPI 收发函数

SPI 收发函数中，每次发送和接收都是一个字节，收发过程中，通过操作 SPI 信号对应的 GPIO 模拟出 SPI 的时序，代码清单如下。

代码清单：SPI 发送

```
/*******************************************************************************
 * 描  述 : 软件模拟 SPI 时序发送一个字节数据，并返回一个字节数据
 * 参  数 : dat[in]：待写入的数据
 * 返回值 : 无
 *******************************************************************************/
static void Spi_WriteOneByte(u8 dat)
{
    u8 i, temp;

    temp = dat;
    for(i=0; i<8; i++) //发送一个字节数据,循环 8 次
    {
        SPI_SCK_LOW;    //拉低时钟线
        spi_delay();    //延时
        if((temp&0x80) == 0x80)//最高为"1"，则 MOSI 引脚输出高电平，否则输出低电平
```

```
16.       {
17.          SPI_MOSI_HIGH;
18.       }
19.       else
20.       {
21.          SPI_MOSI_LOW;
22.       }
23.       SPI_SCK_HIGH; //拉高时钟线
24.       spi_delay();  //延时
25.       temp <<= 1;   //第一位待发的位移到最高位
26.    }
27.    SPI_MOSI_LOW;    //拉低 MOSI
28. }
```

代码清单：SPI 接收

```
1.  /**************************************************************************************
2.   * 描  述 : 软件模拟 SPI 时序读一个字节数据
3.   * 参  数 : 无
4.   * 返回值 : SPI 读取一个字节数据
5.   **************************************************************************************/
6.  static u8 Spi_ReadOneByte(void)
7.  {
8.     u8 i, temp=0;
9.
10.    for(i=0; i<8; i++) //读取一个字节数据,循环 8 次
11.    {
12.       temp <<= 1;     //每个循环，左移一次，保存读取的位
13.       SPI_SCK_LOW;    //拉低时钟线
14.       spi_delay();      //延时
15.       if (SPI_MISO_PIN == 1)//若 MISO 为高电平，temp 最低位置 1
16.       {
17.          temp++;
18.       }
19.       SPI_SCK_HIGH;   //拉高时钟线
20.       spi_delay();      //延时
21.    }
22.    return temp;       //返回读取的数据
23. }
```

5.2.3. 硬件连接

同“实验 2-13-1：硬件 SPI 读写 W25Q128 存储器”。

5.2.4. 实验步骤

同“实验 2-13-1：硬件 SPI 读写 W25Q128 存储器”。

5.3. SPI 读写铁电存储器（FRAM）实验

✧ 注：本节的实验是在“实验 2-13-1：硬件 SPI 读写 W25Q128 存储器”的基础上修改，本节对应的实验源码是：“实验 2-13-3：硬件 SPI 读写铁电存储器（FM25CL64B）”。

5.3.1. 实验内容

将 STC8A8K64D4 单片机的 SPI 配置为主机，通过 SPI 总线访问 FM25CL64B 铁电存储器模块，完成以下操作。

- 数据写入：向指定地址连续写入指定长度数据。
- 数据读取：从指定地址连续读取指定长度数据，并将读取的数据通过串口输出。

✧ 注：本节的实验需要使用艾克姆科技的 FM25CL64B 铁电存储器模块。

5.3.2. 代码编写

1. 新建一个名称为“fm25cl64b.c”的文件及其头文件“fm25cl64b.h”并保存到工程的“Source”文件夹，并将“fm25cl64b.c”加入到 Keil 工程中的“SOURCE”组。
2. 引用头文件

因为在“main.c”文件中使用了“fm25cl64b.c”文件中的函数，所以需要引用下面的头文件“fm25cl64b.h”。

代码清单：引用头文件

```
1.  //引用头文件
2.  #include    " fm25cl64b.h"
```

3. 初始化 SPI

SPI 初始化配置中的引脚配置、SPI 工作模式配置、传输速率配置、收发数据的位序以及中断配置和访问 W25Q128 的例子配置的完全一样，读者参阅“实验 2-13-1：硬件 SPI 读写 W25Q128 存储器”中的 SPI 初始化代码即可。需要注意的是：FM25CL64B 只支持 SPI 通信模式 0 和 3，因此，SPI 初始化代码中使用相对常用的 SPI 通信模式 0。

4. 写使能

FM25CL64B 上电后是禁止写入的，在执行写入操作包括写入状态寄存器（WRSR）和写入存储器（WRITE）之前需要发送写使能（WREN）命令使能写入。写使能的操作步骤和时序如下。

- 拉低片选信号 CS，使能 FM25CL64B。
- 发送写使能命令 0x06（WREN）。
- 拉高片选信号 CS，释放 FM25CL64B。

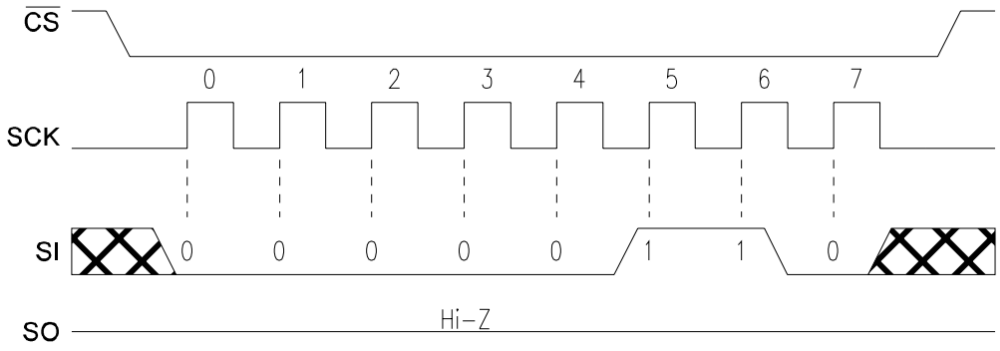


图 19：写使能时序

根据写使能时序，编写的写使能函数代码清单如下。

代码清单：写使能

```
/*********************************************************************************************
 * 描   述 : 写使能
 * 参   数 : 无
 * 返回值 : 无
 *********************************************************************************************/
static void Fram_WriteEnable (void)
{
    SPI_CS_LOW;                     //使能器件
    Spi_WriteOneByte(FRAM_WREN);    //发送写使能指令
    SPI_CS_HIGH;                    //取消片选
}
```

5. 写内存数据

FM25CL64B 写内存数据操作的步骤和时序如下，写操作之前必须先发送“写使能”命令开启 FM25CL64B 的写使能。FM25CL64B 写之前是无需擦除的，数据可以直接覆盖写入。

- 拉低片选信号 CS，使能 FM25CL64B。
- 发送写内存数据命令 0x02（WRITE）。
- 发送 13 位地址，高地址在前。注意发送的地址是两个字节（16 位），其中的高 3 位被忽略。
- 发送写入到 FM25CL64B 的数据，注意最大写入的数据长度不能超过该起始地址到内存结束地址能容纳的字节数。
- 拉高片选信号 CS，释放 FM25CL64B。

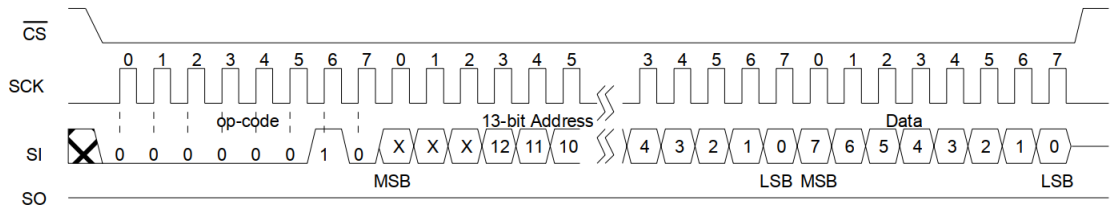


图 20：写数据操作时序

根据写内存数据时序，编写的写内存数据的函数代码清单如下。

代码清单：写使能

```
/*******************************************************************************
 * 描   述：以给定的地址为起始地址连续写入给定大小的数据
 * 参   数：*pBuffer[in]:指向待写入的数据
 *          WriteAddr[in]:写入的起始地址
 *          w_size[in]:写入的字节数
 * 返回值：无
*******************************************************************************/
void Fram_Write_Bytes(u8 * pBuffer,u16 WriteAddr,u16 w_size)
{
   u16 i;
   Fram_WriteEnable();                                  //写使能
   SPI_CS_LOW;
   (void)Spi_WriteOneByte(FRAM_WRITEE);                 //发送写数据指令
   (void)Spi_WriteOneByte((u8)((WriteAddr&0xE0)>>8)); //发送 16bit 地址
   (void)Spi_WriteOneByte((u8)WriteAddr);
   for(i=0;i<w_size;i++)                                //连续写入数据
   {
      Spi_WriteOneByte(pBuffer[i]);
   }
   SPI_CS_HIGH;
}
```

6. 写存储器数据

FM25CL64B 读存储器数据操作的步骤和时序如下。

- 拉低片选信号 CS，使能 FM25CL64B。
- 发送读存储器数据命令 0x03（READ）。
- 发送 13 位地址，高地址在前。注意发送的地址是两个字节（16 位），其中的高 3 位被忽略。
- SPI 通过“虚写”读出数据，每读出一个字节数据，只要主机继续发出时钟和片选为低，地址就在 FM25CL64B 内部自动递增，若地址达到最后一个地址 0x01FFF，则自动返回 0x 0000。
- 拉高片选信号 CS，释放 FM25CL64B。

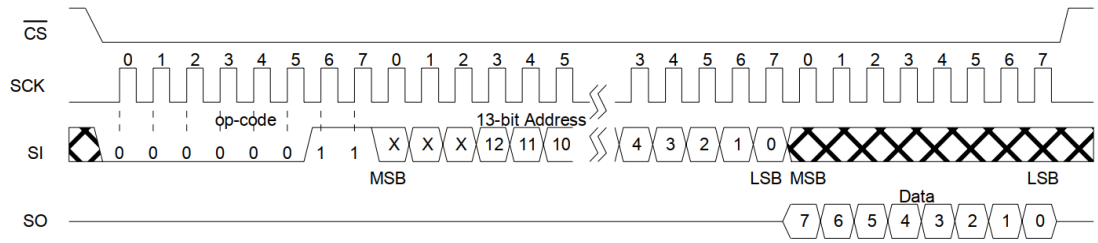


图 21：读存储器时序

根据读存储器数据时序，编写的读存储器数据的函数代码清单如下。

代码清单：读取数据

```
/*****************************************************************************************
 * 描   述：  从指定的起始地址开始连续读取指定长度的数据
 * 参   数：  *pBuffer[in]:指向保存读出数据的缓存的起始地址
 *           ReadAddr[in]:读取数据的起始地址
 *           r_size[in]:读取的字节数
 * 返回值 : 无
*****************************************************************************************/
void Fram_Read_Bytes(u8 * pBuffer,u16 ReadAddr,u16 r_size)
{
   u32 i;

   SPI_CS_LOW;                                  //使能器件
   Spi_WriteOneByte(FRAM_READD);                //发送读取命令
   Spi_WriteOneByte((u8)((ReadAddr&0xE0)>>8)); //发送 16bit 地址
   Spi_WriteOneByte((u8)ReadAddr);
   for(i=0;i<r_size;i++)                         //从地址地地址开始逐字节读出指定大小的数据
   {
      *pBuffer++=Spi_WriteOneByte(0xFF);
   }
    SPI_CS_HIGH;                                 //取消片选
}
```

7. 主函数

主函数中初始化指示灯、按键和 SPI 等外设，之后在主循环中扫描按键状态，并根据按键状态执行以下动作。

- KEY1 按下：从地址 0x0000 开始连续写入 256 个字节数据。
- KEY2 按下：从地址 0x0000 开始连续读出 256 个字节数据。
- KEY3 按下：全片擦除 FM25CL64B（FRAM 是没有擦除操作的，这里所谓的擦除即向 FRAM 全片写入 0x00）。

代码清单如下：

代码清单：主函数

```
/*************************************************************************
功能描述：主函数
入口参数：无
返 回 值：int 类型
*************************************************************************/
int main(void)
{
   u8 btn_val;
   u16 i;
```

```
10.     u8 j = 0;
11.
12.     P2M1 &= 0x3F;   P2M0 &= 0x3F;     //设置 P2.6~P2.7 为准双向口（指示灯 D1 和 D2）
13.     P7M1 &= 0xF9;   P7M0 &= 0xF9;     //设置 P7.1~P7.2 为准双向口（指示灯 D4 和 D3）
14.     P3M1 &= 0x3F;   P3M0 &= 0x3F;     //设置 P3.6~P3.7 为准双向口（按键 KEY2 和 KEY1）
15.     P0M1 &= 0x5F;   P0M0 &= 0x5F;     //设置 P0.5，P0.7 为准双向口（按键 KEY4 和 KEY3）
16.     P3M1 &= 0xFE;   P3M0 &= 0xFE;     //设置 P3.0 为准双向口（串口 1 的 RxD）
17.     P3M1 &= 0xFD;   P3M0 |= 0x02;     //设置 P3.1 为推挽输出（串口 1 的 TxD）
18.
19.     uart1_init();                     //串口 1 初始化
20.     spi_init();                       //初始化 SPI
21.     delay_ms(10);                     //初始化后适当延时
22.     EA = 1;                           //使能总中断
23.
24.     while(1)
25.     {
26.         btn_val=buttons_scan(0);      //获取开发板用户按键检测值，不支持连按
27.         //按下 KEY1：从地址 0x0000 开始连续写入 256 个字节数据
28.         if(btn_val == BUTTON1_PRESSED)
29.         {
30.             j = 0;
31.             for(i=0;i<256;i++)my_tx_buf[i] = j++;
32.             //写入 256 个字节数据
33.             Fram_Write_Bytes(my_tx_buf,0,256);
34.             printf("Write data to fram!\r\n");
35.             //指示灯 D1 状态翻转，指示操作完成
36.             led_toggle(LED_1);
37.         }
38.         //按下 KEY2：从地址 0x0000 开始连续读出 256 个字节数据
39.         else if(btn_val == BUTTON2_PRESSED)
40.         {
41.             printf("Read data from fram!\r\n");
42.             //读取写入的数据
43.             Fram_Read_Bytes(my_rx_buf,0,256);
44.             //串口打印读取的数据
45.             for(i=0;i<256;i++)printf("%02bX ",my_rx_buf[i]);
46.             printf("\r\n");
47.             led_toggle(LED_2);            //指示灯 D2 状态翻转，指示操作完成
48.         }
49.         //按下 KEY3：全片擦除 FRAM，即向 FRAM 全片写入 0x00
50.         else if(btn_val == BUTTON3_PRESSED)
51.         {
52.             led_on(LED_3);            //点亮指示灯 D3
```

```
53.          printf("Clear fram!\r\n");
54.          Fram_Clear_Bytes(0,256);
55.          led_off(LED_3);              //熄灭指示灯 D3
56.      }
57.  }
58. }
```

5.3.3. 硬件连接

本实验需要使用 LED 指示灯、按键和艾克姆科技的铁电存储器（FM25CL64B）模块，按照下图所示安装铁电存储器模块和短接跳线帽。

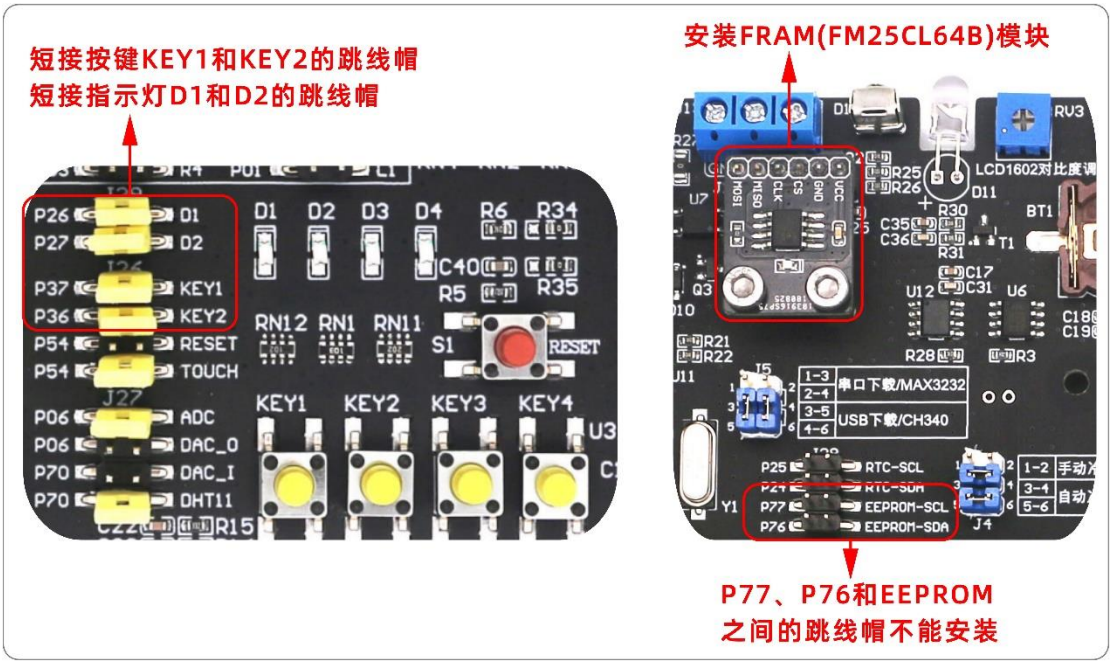


图 22：硬件连接

5.3.4. 实验步骤

1) 解压“…\第 3 部分：配套例程源码”目录下的压缩文件“实验 2-13-3：硬件 SPI 读写铁电存储器（FM25CL64B）”，将解压后得到的文件夹拷贝到合适的目录，如“D\STC8”（这样做的目的是为了防止中文路径或者工程存放的路径过深导致打开工程出现问题）。
2) 双击“…\spi_fm25cl64b\project”目录下的工程文件“spi_fm25cl64b.uvproj”。
3) 点击编译按钮编译工程，编译成功后生成的 HEX 文件“spi_w25q128.hex”位于工程的“…\spi_fm25cl64b\Project\Object”目录下。
4) 打开 STC-ISP 软件下载程序，下载使用内部 IRC 时钟，IRC 频率选择：24MHz。
5) 电脑上打开串口调试助手，选择开发板对应的串口号，将波特率设置为 9600bps。

- 程序运行后，先按下 KEY1 按键，向 FRAM 写入数据：从地址 0x0000 开始连续写入 256 个字节数据。再按下 KEY2 按键，读出刚写入的数据并通过串口输出，如下图所示。

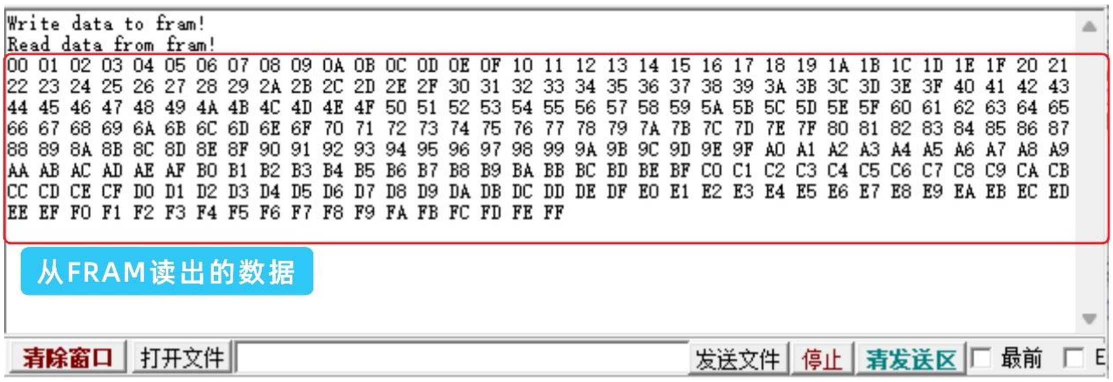


图 23：串口接收的数据

- 按下 KEY3 按键，擦除 FRAM（向 FRAM 全片写入 0x00），之后再按下 KEY2 按键读取数据，可以看到读出的数据均为 0x00。

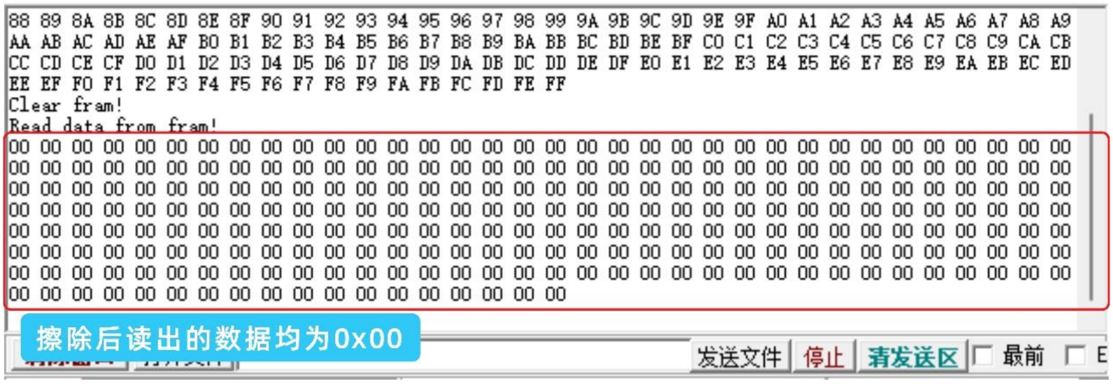


图 24：串口接收的数据

✧ **说明**：模拟 SPI 读写 FRAM 存储器和模拟 SPI 读写 W25Q128 存储器的修改方法一样，这里不再赘述。我们也编写好了 SPI 读写 FRAM 存储器的例子，该例子在资料的“…\第 3 部分：配套例程源码”目录下，名称为“实验 2-13-4：模拟 SPI 读写铁电存储器（FM25CL64B）”，读者在编写的过程中可以参考一下。