青风手把手教你学 stm32f030 系列教程

---------- 库函数操作版本

出品论坛： www.qfv8.com 青风电子社区

作者： 青风
出品论坛: www.qfv8.com
淘宝店：http://qfv5.taobao.com
QQ 技术群：241364123
硬件平台：QF-STM32F030 开发板

2.12 I2C 读写 EEPROM

2.12.1 原理分析

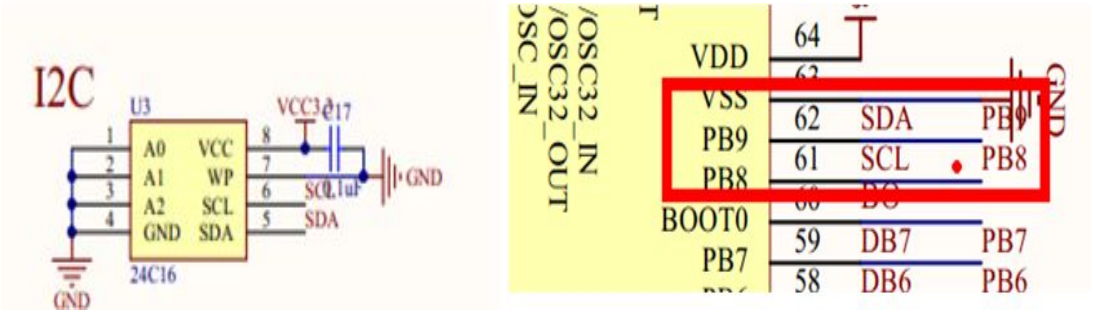
I2C（Inter－Integrated Circuit）总线是由 PHILIPS 公司开发的两线式串行总线，用于连接微控制器及其外围设备。是微电子通信控制领域广泛采用的一种总线标准。它是同步通信的一种特殊形式，具有接口线少，控制方式简单，器件封装形式小，通信速率较高等优点。

只要求两条总线线路：一条串行数据线 SDA，一条串行时钟线 SCL；串行的 8 位双向数据传输位速率在标准模式下可达 100kbit/s，快速模式下可达 400kbit/s，高速模式下可达 3.4Mbit/s；每个连接到总线的器件都可以通过唯一的地址和一直存在的简单的主机/从机关系软件设定地址，主机可以作为主机发送器或主机接收器； 它是一个真正的多主机总线，如果两个或更多主机同时初始化，数据传输可以通过冲突检测和仲裁防止数据被破坏；

I2C 的使用实际上是严格遵循 I2C 总线协议进行的，下面我们结合 STM32F051 一一进行分析：

首先看 I2C 总线硬件：

2.12.2 硬件准备：



青风 STM32F051 开发板采用从机挂 24c02，24c02 是比较常见的 eeprom,具有标志的 i2c 总线，通过 SCL 和 SDA 和 stm32f051 上的 PB8 和 PB9 相连。

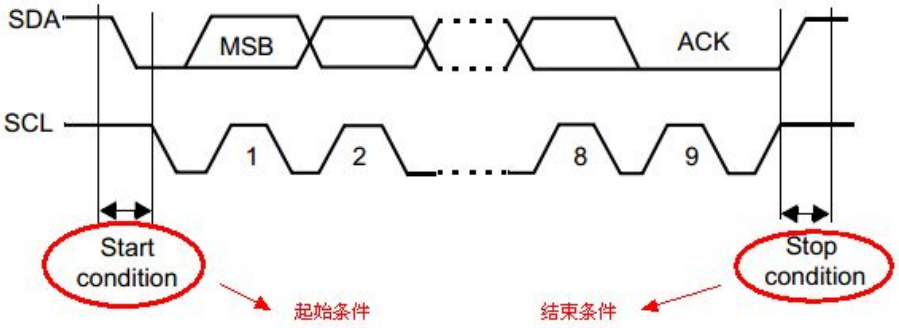
SCL（serial clock）:称为串行时钟线

SDA（serial data ）称为串行数据线

那么如下图所示,i2c 总线上在传输数据的时候具有四种类型信号：分别为开始信号，停止信号，重新开始信号和应答信号。

开始信号 Start:此时 SCL 处于高电平，SDA 由高电平跳转到低电平，产生开始信号。

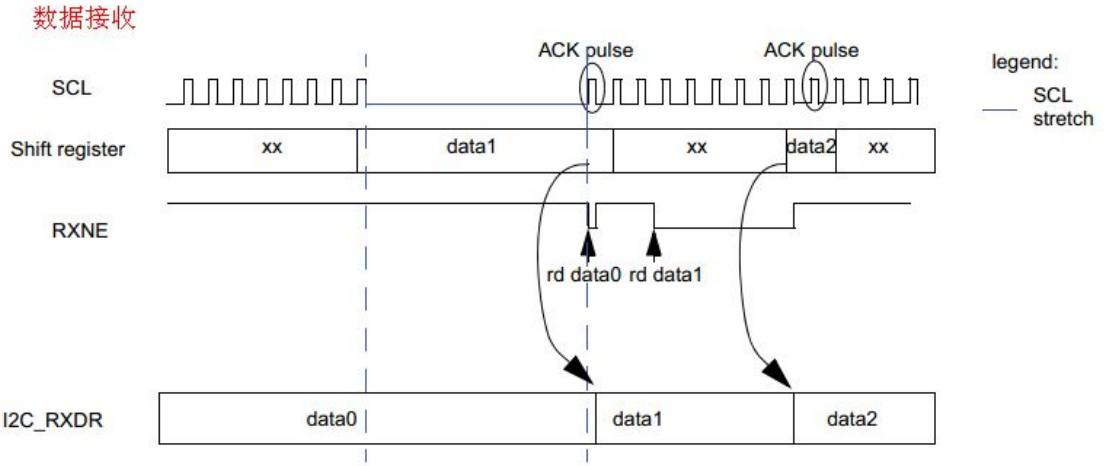
停止信号 Stop::SCL 同样需要处于高电平，SDA 则由低电平跳转到高电平，产生停止信号

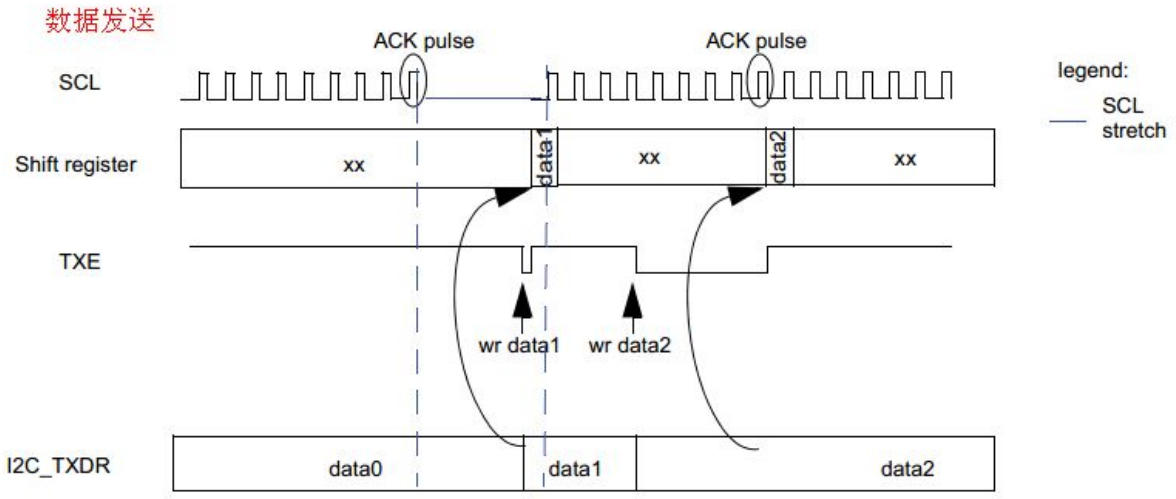


重新开始信号：它本质上是一个开始信号，信号电平和开始信号一样，是在连续传输信号时进行使用。

应答信号：接收数据的 IC 在接收到 8 位数据后，会向主设备发出一个特定的低电平脉冲，这就是应答信号。应答信号一般在第 9 个周期出现，每位信号必须跟一位应答信号，表示数据已经接收。

如下图所示：

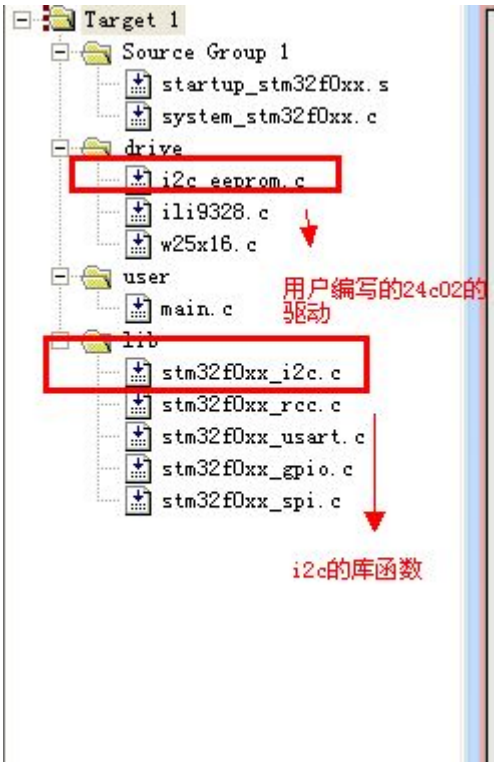




上面两个图表示了数据连续传输的基本格式，那么下面我们就来学习如何通过 I2C 对 24c02 进行读写。

2.12.3 软件准备：

采样库函数编写程序，工程配置如下图所示，用户需要调用 stm32f0xx_i2c.c 库函数。需要编写 i2c_eeprom.c 驱动文件和主函数 main.c。



首先谈谈 24C02 的驱动函数 i2c_eeprom,c 的编写，串行 E2PROM 是基于 I2C-BUS 的存储器件，遵循二线制协议，由于其具有接口方便，体积小，数据掉电不丢失等特点。对于只用一片 24C02 器件的系统，因为不需要分辨不同的地址，只要 WP 保护功

能正常就可以了，这只要断开 WP 与 CPU 连线且保持高电平，再试一下系统数据读写功能是否正常就可以了。ATMEL24C02 为标准的芯片 Twr 实际在 2ms 左右。

首先需要对 stm32f051 的 i2c 接口进行初始化，如下代码，首先对 IO 端口进行配置：

```
01. void GPIO_Configuration(void)
02. {
03.    GPIO_InitTypeDef    GPIO_InitStruct;
04.    RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOB, ENABLE);//使能 GPIO 时钟
05.    RCC_APB1PeriphClockCmd(RCC_APB1Periph_I2C1 , ENABLE); //使能 i2c 时钟
06.
07.    GPIO_InitStruct.GPIO_Pin = GPIO_Pin_8;
08.    GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AF;//配置端口复用
09.    GPIO_InitStruct.GPIO_Speed = GPIO_Speed_Level_3;
10.    GPIO_InitStruct.GPIO_OType = GPIO_OType_OD;
11.    GPIO_Init(GPIOB , &GPIO_InitStruct);
12.    GPIO_InitStruct.GPIO_Pin = GPIO_Pin_9;
13.    GPIO_Init(GPIOB , &GPIO_InitStruct);
14.    GPIO_PinAFConfig( GPIOB , GPIO_PinSource8, GPIO_AF_1);
15.    GPIO_PinAFConfig( GPIOB ,GPIO_PinSource9, GPIO_AF_1); //设置复用通道
16. }
```

IO 端口的复用设置好后，需要配置 I2C 的一些基本参数，这些参数在 stm32f0xx_i2c.h 文档中通过结构体进行了定义，如下面代码：

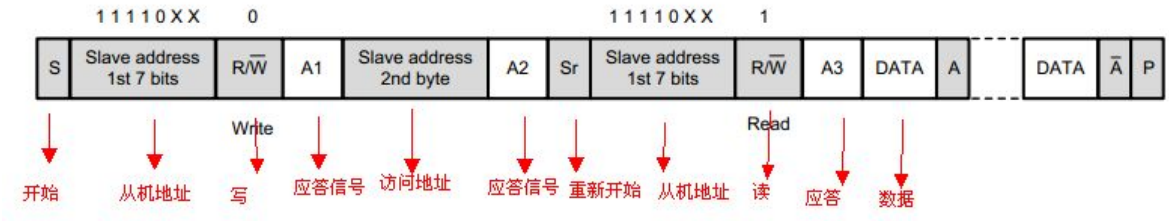
```
17. typedef struct
18. {
19.    uint32_t I2C_Timing;              //i2c 时钟配置
20.    uint32_t I2C_AnalogFilter;        //i2c 模拟滤波
21.    uint32_t I2C_DigitalFilter;       //i2c 数字滤波
22.    uint32_t I2C_Mode;                //i2c 模式配置
23.    uint32_t I2C_OwnAddress1;         //设备地址配置
24.    uint32_t I2C_Ack;                 //i2c 应答
25.    uint32_t I2C_AcknowledgedAddress; //i2c 确定是 7bit 或者 10bit 应答地址
26. }I2C_InitTypeDef;
```

在编写 24c02 的驱动文件时，按照上面的结构体进行配置，可以配置为如下情况：

```
27. void I2C_Configuration(void)//i2c 的初始化
28. {
29.    I2C_InitTypeDef    I2C_InitStruct;
30.
31.    I2C_InitStruct.I2C_Mode = I2C_Mode_I2C;//配置模式
32.    I2C_InitStruct.I2C_AnalogFilter = I2C_AnalogFilter_Enable;//使能模拟噪声滤波
33.    I2C_InitStruct.I2C_DigitalFilter = 0x00;
34.    I2C_InitStruct.I2C_OwnAddress1 =0x00;
35.    I2C_InitStruct.I2C_Ack = I2C_Ack_Enable;//使能应答
```

```
36.    I2C_InitStruct.I2C_AcknowledgedAddress = I2C_AcknowledgedAddress_7bit;  //配置格式
       为 7bit
37.    I2C_InitStruct.I2C_Timing = 0x00210507;//设置时钟寄存器的值
38.    I2C_Cmd(I2C1, ENABLE);//使能 i2c
39.    I2C_Init(I2C1, &I2C_InitStruct);//调入结构体
40. }
```

按照上面的要求配置好了 i2c 的端口后，下一步就是考虑如何通过 i2c 进行读写了，下图是 i2c 主机从从机读数据的过程：



首先是开始信号，然后发送从机地址，写选通，给一个应答信号。然后写入要访问的地址。
再次应答。从新开始信号，发送从机地址，从机地址是外挂总线地址。然后读选通，主机等待从机的应答信号，当接收应答后，开始读取数据。

而 24C02 写和读是按照页进行。而按页写最大页面不超过 8 个字节，超过后会发送覆盖。此时采用缓冲进行处理和分配，下面几个函数就是写缓冲和写页面：

```
41. uint32_t sEE_ReadBuffer(uint8_t* pBuffer, uint16_t ReadAddr, uint16_t* NumByteToRead);
```

读缓冲函数：uint8_t* pBuffer：缓冲指针。
uint16_t ReadAddr：读的地址。
uint16_t* NumByteToRead：读字节数

```
42. uint32_t sEE_WritePage(uint8_t* pBuffer, uint16_t WriteAddr, uint8_t* NumByteToWrite);
```

写页函数：uint8_t* pBuffer, 缓冲指针
uint16_t WriteAddr, 写的地址
uint8_t* NumByteToWrite 写字节数

```
43. void    sEE_WriteBuffer(uint8_t* pBuffer, uint16_t WriteAddr, uint16_t NumByteToWrite);
```

写缓冲函数：uint8_t* pBuffer, 缓冲指针
uint16_t WriteAddr, 写的地址
uint8_t* NumByteToWrite 写字节数

```
44. uint32_t sEE_WaitEepromStandbyState(void);
```

等待 EEPROM 待机状态

主函数首先向 24c02 写入一个数据，然后读出这个数据。对比两个数据是否一样，判断读写是否正确：

```
45. uint8_t temp1[]="qfstm32";
```

```
uint8_t temp2[];

int main(void)

{
    SystemInit();
      LCD_init();           // 液晶显示器初始化

      LCD_Clear(WHITE);    // 全屏显示白色
      POINT_COLOR = BLACK; // 定义笔的颜色为黑色
      BACK_COLOR = WHITE;   // 定义笔的背景色为白色
    I2C_EE_Init();
    SPI_FLASH_Init();
    LCD_ShowString(2,2,"实验十一");
    LCD_ShowString(100,2,"i2c 读 24c02 实验");
      sEE_WriteBuffer(temp1, 0x050, 8);
     LCD_ShowString(2,40,"24c02 写入值：");
      LCD_ShowString(100,40,temp1);
     NumDataRead=8;
     sEE_ReadBuffer(temp2, 0x050,(uint16_t *)(&NumDataRead));
      LCD_ShowString(2,60,"24c02 读出值：");
     LCD_ShowString(100,60,temp2);

           if(*temp1 != *temp2)
           {
            LCD_ShowString(50,80,"读取 EEROR");
           }
      else
      {
        LCD_ShowString(50,80,"读取 sccess");
      }

     while(1)
      {}
}
```

2.12.4 实验现象：

整个操作的显示如下：