第 2-13 讲：模拟 I2C 读写 AT24C02 存储器

1. 学习目的

1. 了解 I2C 总线的特点。
2. 掌握 I2C 地址的定义，对 I2C 地址要有深刻的了解，之后再看到 I2C 接口设备中描述的 7 位地址或 8 位地址，不会再有疑惑。
3. 掌握 IAP15F2K61S2/IAP15W4K61S4 单片机模拟 I2C 的特点以及编程方法。
4. 掌握通过 I2C 读写 eeprom AT24C02。这里的重点是使用按页写入来代替逐个字节写入，从而有效节省写入时间。

2. I2C 总线概述

2.1. 主要特征

典型的 I2C 应用原理如下图所示，I2C 总线通信仅需两根信号线，可以连接多个设备，从设备都有唯一的地址，主设备通过从设备的地址和不同的从设备通信。

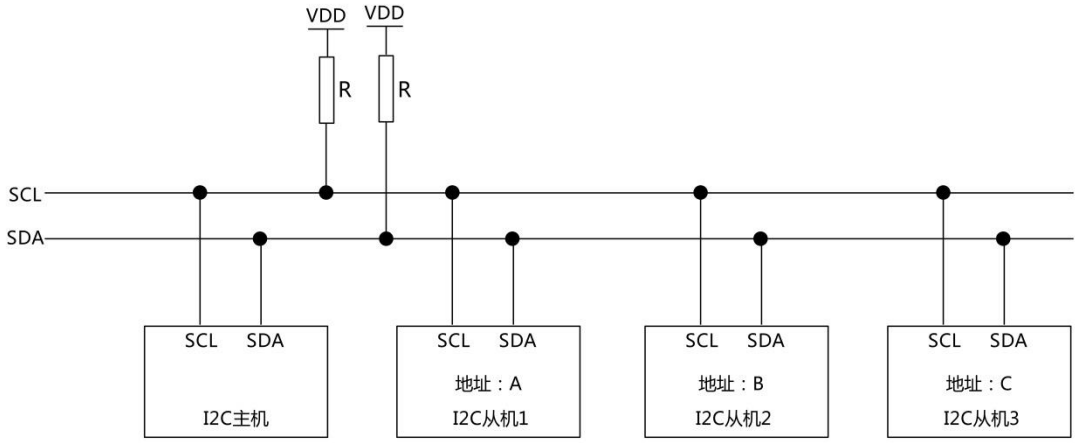

图 1：典型的 I2C 总线应用

1) I2C 总线硬件结构简单，仅需一根时钟线（SCL）、一根数据线（SDA）和两个上拉电阻即可实现通信。I2C 总线的 SCL 和 SDA 均为开漏结构，开漏结构的电路只能输出“逻辑 0”，无法输出“逻辑 1”，因此 SCL 和 SDA 需要连接上拉电阻。上拉电阻的阻值影响传输速率，阻值越大，由于 RC 影响，会带来上升时间的增大，传输的速率慢，阻值小，传输的速率快，但是会增加电流的消耗，一般情况下，我们会选择 4.7K 左右的阻值，在从机数量少，信号线短的情况下，可以适当增加阻值，如使用 10K 的阻值。
2) I2C 总线中的从设备必须有自己的地址，并且该地址在其所处的 I2C 总线中唯一，主设备通过此唯一的地址即可和该从设备进行数据传输。
3) I2C 总线支持多主机，但是同一时刻只允许有一个主机。I2C 总线中存在多个主机时，

为了避免冲突，I2C 总线通过总线仲裁决定由哪一个主机控制总线。

4) I2C 总线只能传输 8 位的数据，数据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。

5) 同时连接到同一个 I2C 总线上的设备数量受总线最大电容（400pF）的限制。

6) I2C 总线电流消耗很低，抗干扰强，适合应用于低功耗的场合。

2.2. I2C 地址

I2C 总线中的设备必须要有唯一的地址，这意味着如果在总线中接入两个相同的设备，该设备必须有配置地址的功能，这也是我们经常用的 I2C 接口的设备会有几个引脚用来配置地址的原因。

对于 I2C 地址，我们经常看到有的 I2C 接口设备在规格书中描述的是 7 位地址，而有的 I2C 接口设备在规格书中描述的是 8 位地址，他们有什么区别？（I2C 也有 10 位地址，但用的较少，这里不做介绍，本章中的内容不涉及到 10 位地址）。

7 位地址和 8 位地址如下图所示，他们结构上是一样的，都是由 7 个地址位加一个用来表示读写的位组成，只是描述上有所区别。

- 规格书中描述 I2C 地址是 7 位地址的设备：给出的是 7 个地址位加 R/W 位，最低位（R/W 位）为 0 时表示为写地址，最低位为 1 时为读地址。如果把 0 和 1 分别带入 R/W 位，得到的地址就和 8 位地址一样了。
- 规格书中描述 I2C 地址是 8 位地址的设备：直接给出写地址和读地址，也就是最低位（R/W 位）为 0 时的地址和最低位为 1 时的地址。

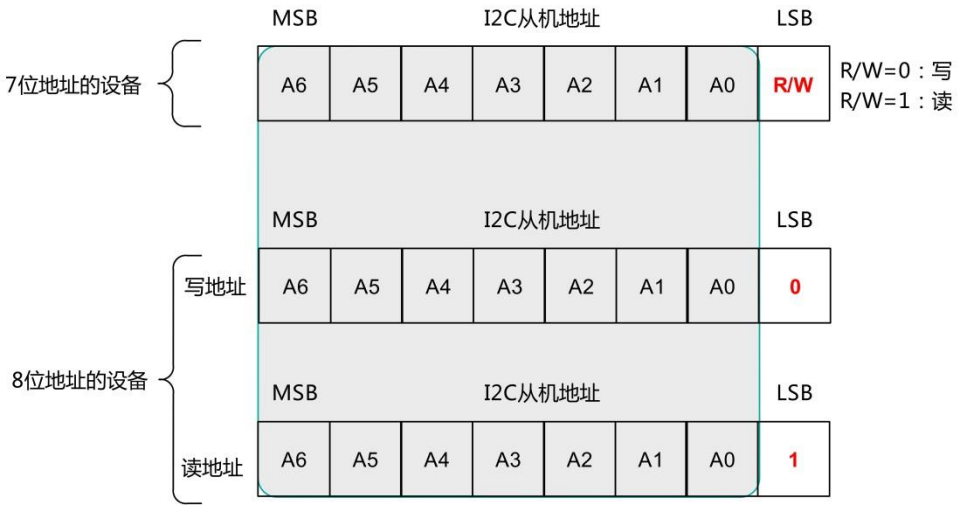

图 2：I2C 地址

由此可见，所谓的 7 位地址和 8 位地址实际上都是 7 位地址加上最低位的读写位，本质上是一样的，只是各个 I2C 接口设备的描述方式不一样。

I2C 保留了如下表所示的两组 I2C 地址，这些地址用于特殊用途。

表 1：保留地址

从机地址	R/ W 位	描述

0000 000	0	广播呼叫地址。
0000 000	1	起始字节。
0000 001	X	CBUS 地址。
0000 010	X	保留给不同的总线格式 。
0000 011	X	保留到将来使用。
0000 1XX	X	Hs 模式主机码。
1111 1XX	X	保留到将来使用。
1111 0XX	X	10 位从机寻址。

2.3. I2C 数据传输

1. 起始和停止条件（START and STOP conditions）

所有的 I2C 事务都是以 START 开始、STOP 结束，起始和停止条件总是由主机产生，如下图所示，当 SCL 为高电平时，SDA 从高电平向低电平转换表示起始条件，当 SCL 是高电平时，SDA 由低电平向高电平转换表示停止条件。如果总线中存在多个主机，先将 SDA 拉低的主机获得总线控制权。

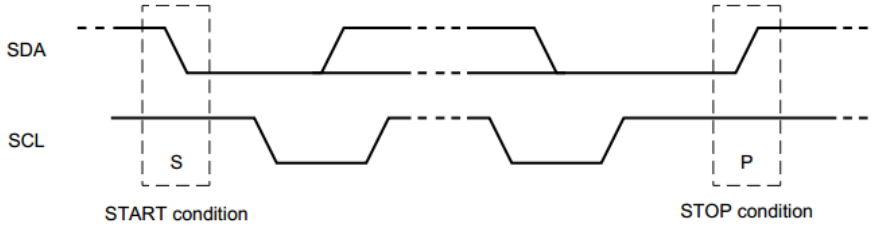

图 3：起始和停止条件

2. 字节格式（Byte format）

I2C 总线发送到 SDA 上的数据必须为 8 位，即一次传输一个字节，每次传输可以发送的字节数量不受限制。每个字节后必须跟一个响应位，首先传输的是数据的最高位 MSB，如果从机要完成一些其他功能后，例如一个内部中断服务程序才能接收或发送下一个完整的数据字节，那么从机可以将时钟线 SCL 保持为低电平强制主机进入等待状态，当从机准备好接收下一个字节数据并释放时钟线 SCL 后数据传输继续。

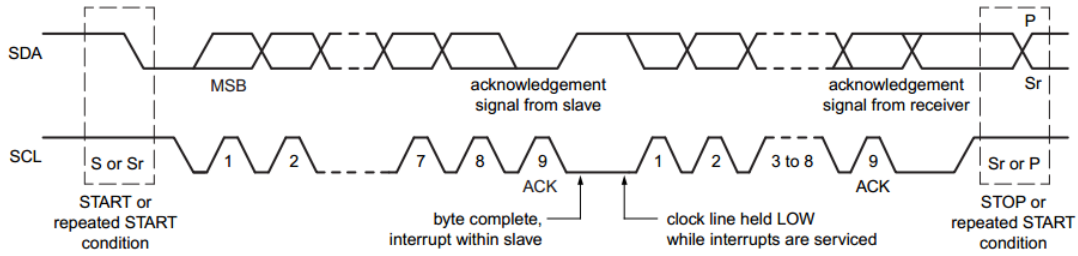

图 4：I2C 总线数据传输

2.4. ACK 和 NACK

每个字节后会跟随一个 ACK 信号。接收者通过 ACK 位告知发送者已经成功接收一字

节数据并准备好接收下一字节数据。所有的时钟脉冲包括 ACK 信号的时钟脉冲都是由主机产生的。

- ACK 信号：发送者发送完 8 位数据后，在 ACK 时钟脉冲期间释放 SDA 线，接收者可以将 SDA 拉低并在时钟信号为高时保持低电平，这样就产生了 ACK 信号，从而使得主机知道从机已成功接收数据并且准备好了接收下一数据。
- NACK 信号：当 SDA 在第 9 个时钟脉冲的时候保持高电平，定义为 NACK 信号。这时，主机要么产生 STOP 条件来放弃这次传输，要么重复 START 条件来启动一个新的传输。

下面的 5 种情况会导致产生 NACK 信号：

1) 发送方寻址的接收方在总线上不存在，因此总线上没有设备应答。
2) 接收方正在处理一些实时的功能，尚未准备好与主机通信，因此接收方不能执行收发。
3) 在传输期间，接收方收到不能识别的数据或者命令。
4) 在传输期间，接收方无法接收更多的数据字节。
5) 主-接收器要通知从-发送器传输的结束。

2.5. 从机地址和 R/W 位

I2C 数据传输如下图所示，在起始条件(S)后，发送从机地址，从机地址是 7 位，从机地址后紧跟着的第 8 位是读写位（R/W），读写位为 0 表示写，读写位为 1 表示读。数据传输一般由主机产生的停止位 P 终止，但是，如果主机仍希望在总线上通信，他可以产生重复起始条件 S 和寻址另一个从机而不是首先产生一个停止条件，在这种传输中可能有不同的读写格式结合。

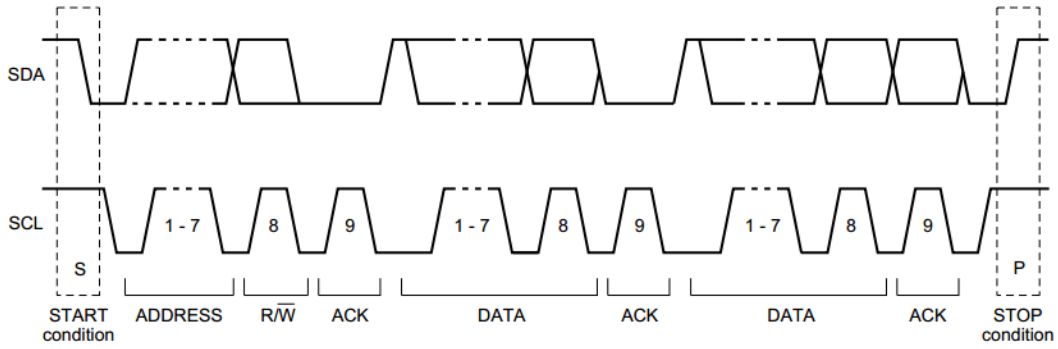

图 5：I2C 总线传输时序

可能的数据传输格式有：

1. 主机发送器发送到从机接收器，传输的方向不会改变，接收器应答每一个字节，如下图所示。

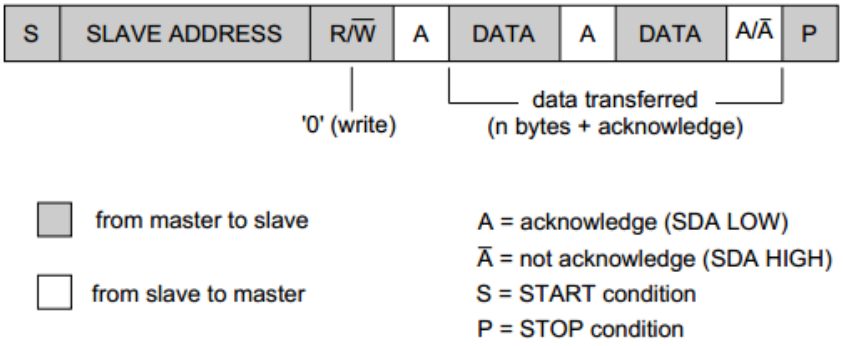

图 6：主机发送器 7 位地址寻址从机接收器（传输方向不改变）

2. 在第一个字节后，主机立即读从机，在第一次应答后，主机发送器变成主机接收器，从机接收器变成从机发送器。第一次应答仍由从机生成，主机生成后续应答。之前发送了一个非应答（A）的主机产生 STOP 条件。

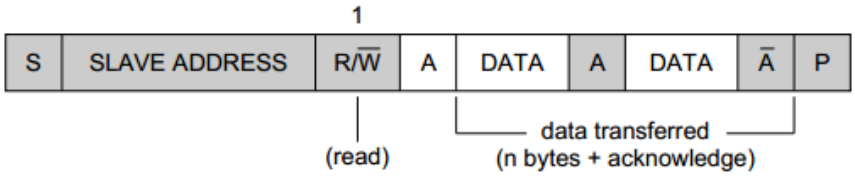

图 7：主机发送第一个字节后立即读取从机

3. 复合格式，如下图所示。传输改变方向的时侯，起始条件和从机地址都会被重复，但 R/W 位取反。如果主接收器发送重复 START 条件，他会在重复 START 条件之前发送一个非应答（A）。

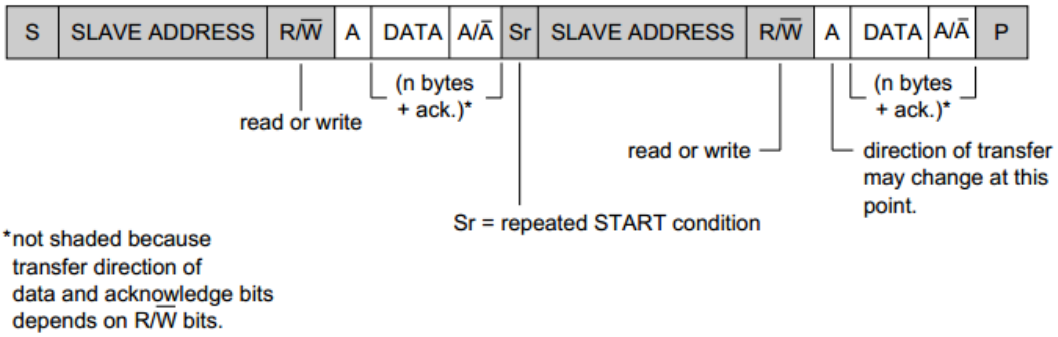

图 8：复合格式

3. 硬件设计

PK107D 开发板上设计了 I2C 接口的 E2PROM 存储器（AT24C02）电路单元，用于我们学习 I2C 的应用。

3.1. AT24C02（E2PROM）电路

EEPROM（全称是 Electrically Erasable Programmable Read-Only Memory）带电可擦除可编程只读存储器，该类存储器是用户可更改的只读存储器（ROM），其可通过高于普通电压的作用来擦除和重编程（重写）。EEPROM（常写成 E2PROM）是一种特殊形式的

闪存，其应用通常是个人电脑中的电压来擦写和重编程。

E2PROM 存储器可分为片内 E2PROM 和片外 E2PROM，片外 E2PROM 和单片机之间通过各种通信接口连接，而最为常见的通信接口即是 I2C 接口。一般情况下，E2PROM 可写入或擦除的次数是 30~100 万次，而读取次数是没有限制的。

AT24C02 是一款串行 CMOS E2PROM，常用来存储一些配置数据，他的存储空间为 2K 位（256 个字节），页面大小为 8 个字节，共 32 个页面。当然如果需要更大存储空间的存储器可以选择其他型号的，比如 24C04 芯片（512 字节）、24C08 芯片（1024 字节）和 24C16 芯片（2048 字节）。

开发板上的 AT24C02 硬件电路如下图所示。AT24C02 的器件地址的低 3 位可以通过引脚 A2 A1 A0 配置，本电路中引脚 A2 A1 A0 均连接到 GND，因此，地址的低 3 位均为 0。

AT24C02 通过 I2C 接口和 IAP15F2K61S2/IAP15W4K61S4 连接，I2C 接口为开漏输出，因此 SCL 和 SDA 线上需要增加上拉电阻，这里我们使用的上拉电阻阻值为 6.8K。

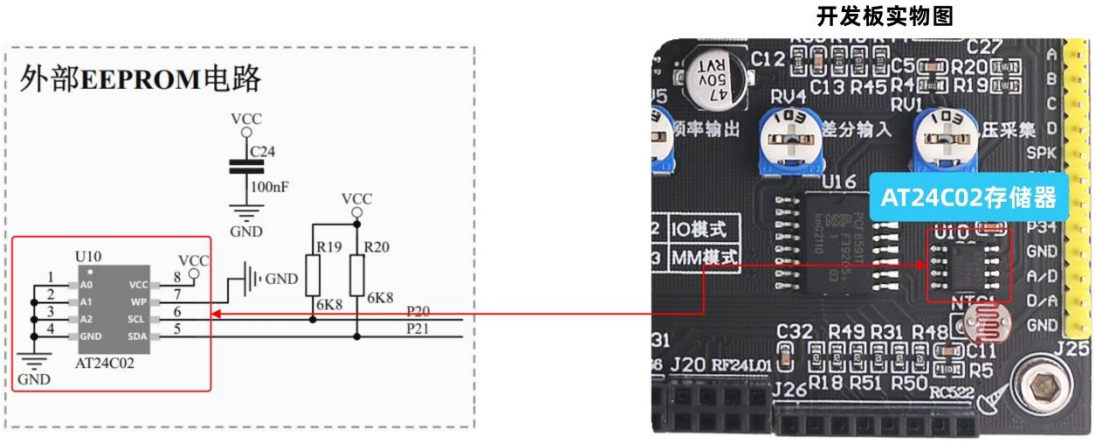

图 9：E2PROM 电路

E2PROM 存储器（AT24C02）占用的 IAP15F2K61S2/IAP15W4K61S4 的引脚如下表：

表 2：I2C 连接 AT24C02 引脚分配

名称	引脚	说明
SCL	P2.0	独立 IO 口
SDA	P2.1	独立 IO 口

✧ 注：为了方便读者理解单片机访问 AT24C02 存储器的编程，AT24C02 的 I2C 通信时序以及读、写、擦除操作将在软件设计部分讲解。

4. 软件设计

4.1. I2C 读写 AT24C02 存储器实验

✧ 注：本节的实验是在“实验 2-5-1：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：模拟 I2C 读写 AT24C02 存储器”。

4.1.1. 实验内容

配置 IAP15F2K61S2/IAP15W4K61S4 单片机的 I2C 引脚，模拟 I2C 时序写函数，通过模拟 I2C 总线访问 AT24C02 存储器，完成以下操作。

- 单个字节写入：向指定地址写入 1 个字节数据。
- 单个字节读出：从指定地址读取 1 个字节数据。
- 批量写：向指定地址连续写入指定长度数据。为了节省写入时间，写入数据时使用按页写入，因此，批量写入要能处理跨页写入的问题。
- 批量读：从指定地址开始顺序读取指定长度数据，并将读取的数据通过串口输出。

4.1.2. 代码编写

1. 新建一个名称为“i2c_hw.c”的文件及其头文件“i2c_hw.h”保存到工程的“Source”文件夹，并将“i2c_hw.c”加入到 Keil 工程中的“SOURCE”组。该文件用于存放 I2C 硬件操作相关的函数。
2. 新建一个名称为“at24c02.c”的文件及其头文件“at24c02.h”保存到工程的“Source”文件夹，并将“at24c02.c”加入到 Keil 工程中的“SOURCE”组。该文件用于存放 EEPROM 存储器 AT24C02 操作相关的函数。
3. 引用头文件

因为在“main.c”文件中使用了“at24c02.c”文件中的函数，所以需要引用下面的头文件“at24c02.h”。

代码清单：引用头文件

```
1.  //引用 AT24C02 的头文件
2.  #include    " at24c02.h"
```

4. AT24C02 地址的确定

查询 AT24C02 数据手册可知其地址高 4 位固定为“1010”，如下图所示，紧跟着的 3 个位由芯片的引脚 A2、A1 和 A0 的电平确定，最低位为读写位。开发板上 AT24C02 的硬件电路中将引脚 A2、A1 和 A0 连接到了 GND，因此 A2、A1 和 A0 均为 0，由此可提取出 7 位地址为 0x50，转换为 8 位地址：0xA0（写）、0xA1（读），如下图所示。

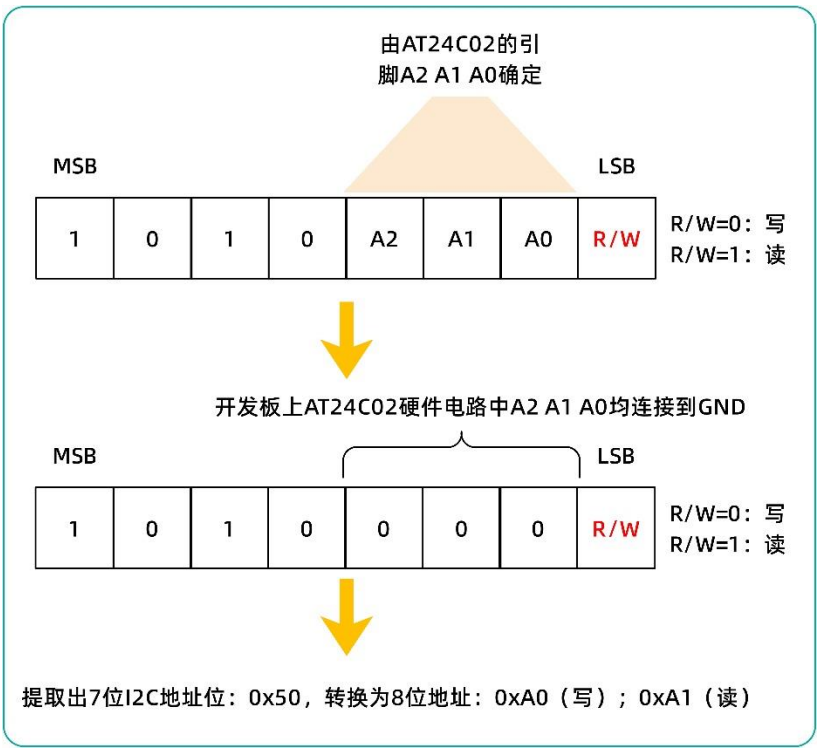

图 10：AT24C02 地址的确定

代码中，AT24C02 地址定义如下：

代码清单：定义 AT24C02 地址

```
1.  #define AT24C02_ADDR_W  0xA0            //I2C 从机写地址
2.  #define AT24C02_ADDR_R  0xA1            //I2C 从机读地址
```

5. 单字节写入：向指定地址写入单个字节数据

AT24C02 将单个字节写入到指定地址的时序如下图所示，首先产生起始条件，紧跟着发送 7 位地址 + 0（写），之后发送写入数据的地址和数据，最后产生停止条件。

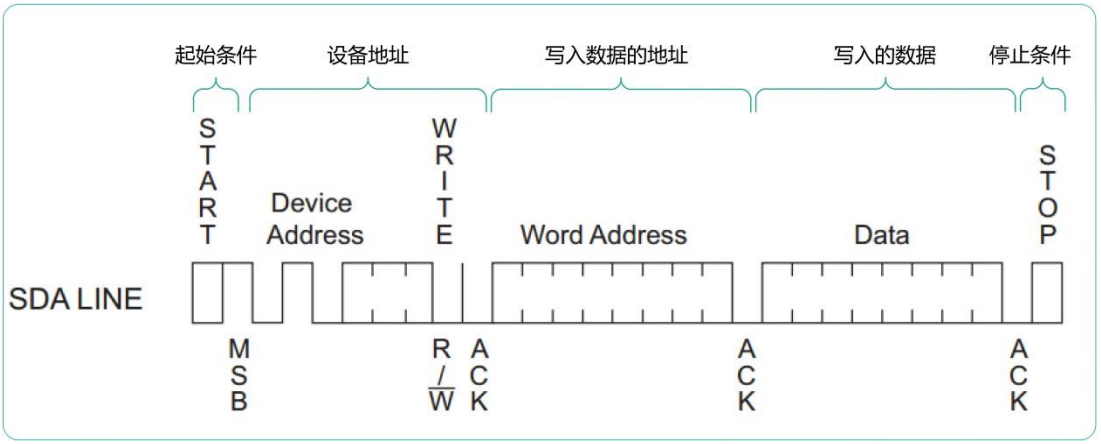

图 11：AT24C02 写入单个字节数据时序

根据时序，即可写出“写入单个字节数据”的函数，代码清单如下。这里面需要特别注意的是：I2C 发送完成后，仅表示 AT24C02 接收到了数据，并不表示 AT24C02 已经完成

了数据的写入。AT24C02 是在停止条件产生后进入写入周期的，AT24C02 的写入周期所需的最大时间是 5ms，因此 I2C 传输完成后要延时 5ms 以确保 AT24C02 正确写入数据。

代码清单：向指定地址写入单个字节数据

```
/*******************************************************************************
 * 描  述 : 向 AT24C02 指定地址写入一个字节数据
 * 参  数 : Addr[in]：写入数据的地址
 *        : dat[in]：待写入的数据
 * 返回值 : 无
 *******************************************************************************/
void AT24C02_write_byte(u8 Addr,u8 Dat)
{
    I2C_Start();                    //发送起始命令，产生起始条件
    I2C_SendData(AT24C02_ADDR_W);   //发送器件地址（写）
    I2C_WaitACK();                  //接收 ACK
    I2C_SendData(Addr);             //发送数据写入地址
    I2C_WaitACK();                  //接收 ACK
    I2C_SendData(Dat);              //发送写入的数据
    I2C_WaitACK();                  //接收 ACK
    I2C_Stop();                     //发送停止命令，产生停止条件
    delay_ms(5);                    // AT24C02 的写入周期所需的最大时间是 5ms，延时 5ms 确保数据正确写入
}
```

6. 单个字节读出：从指定地址读取 1 个字节数据

AT24C02 读取单个字节数据的时序如下图所示，先执行发送操作，将需要读取的数据的地址发送给 AT24C02，之后再执行读取操作，即可读出数据。

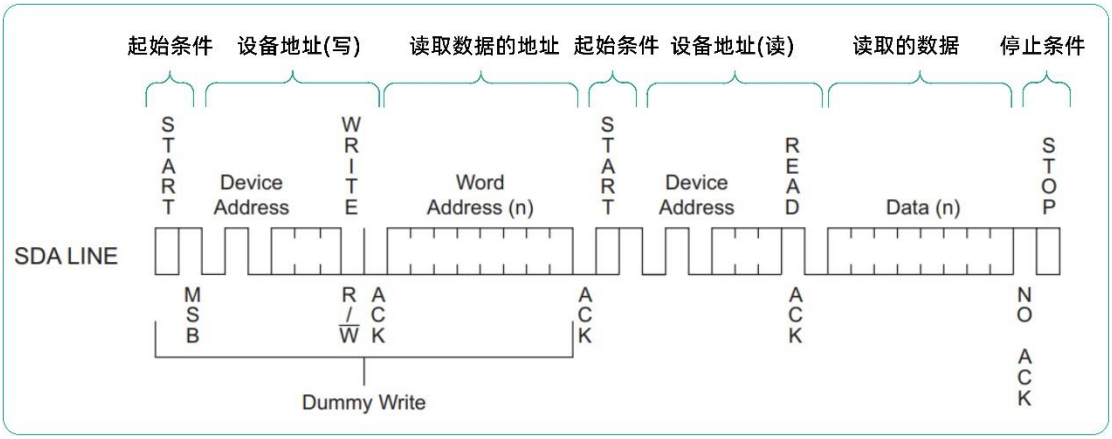

图 12：AT24C02 读取单个字节数据时序

读取单个字节数据的代码清单如下。

代码清单：从指定地址读出单个字节数据

```
/*******************************************************************************
 * 描  述 : 从 AT24C02 指定地址读取一个字节数据
```

```
3.    * 参  数 : Addr[in]：读出数据的地址
4.    * 返回值 : 读取的数据
5.    ********************************************************************************/
6.   u8 AT24C02_read_byte(u8 Addr)
7.   {
8.      u8 temp=0;
9.      I2C_Start();                        //发送起始命令，产生起始条件
10.     I2C_SendData(AT24C02_ADDR_W);       //发送器件地址（写）
11.     I2C_WaitACK();                      //接收 ACK
12.     I2C_SendData(Addr);                 //发送读取数据的地址
13.     I2C_WaitACK();                      //接收 ACK
14.     I2C_Start();                        //发送起始命令，产生起始条件
15.     I2C_SendData(AT24C02_ADDR_R);       //发送器件地址（读）
16.     I2C_WaitACK();                      //接收 ACK
17.     temp=I2C_RecvData();                //读一个字节数据
18.     I2C_Stop();                         //发送停止命令，产生停止条件
19.     return temp;                        //返回读取的数据
20.  }
```

7. 按页写入：向指定页面连续写入不大于页面长度的数据

AT24C02 按页写入的时序如下图所示，首先产生起始条件，紧跟着发送 7 位地址 + 0（写），接着发送写入数据的地址，之后连续发送写入的数据，最后产生停止条件。

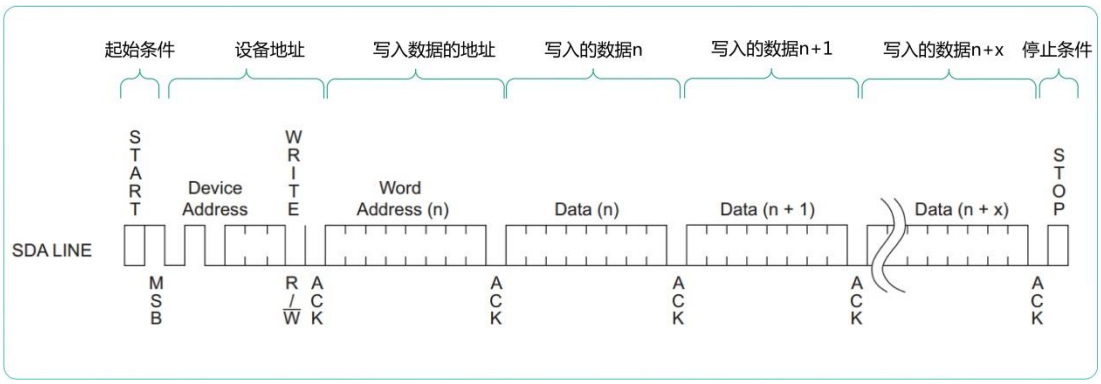

图 13：AT24C02 按页写入时序

按页写入要注意 AT24C02 按页写时的跨页问题，如果地址跨页，则写指针会返回到当前页的起始地址，这一点非常重要，接下来我们来分析按页写的几种情况。

AT24C02 的容量为 256 个字节，页面大小为 8 个字节，因此 AT24C02 被分为 32 个页面，第一个页面地址位 0~7，第二个页面地址位 8~15，以此类推。如下图所示，按页写时如果地址没有超过当前页面，写入正确。

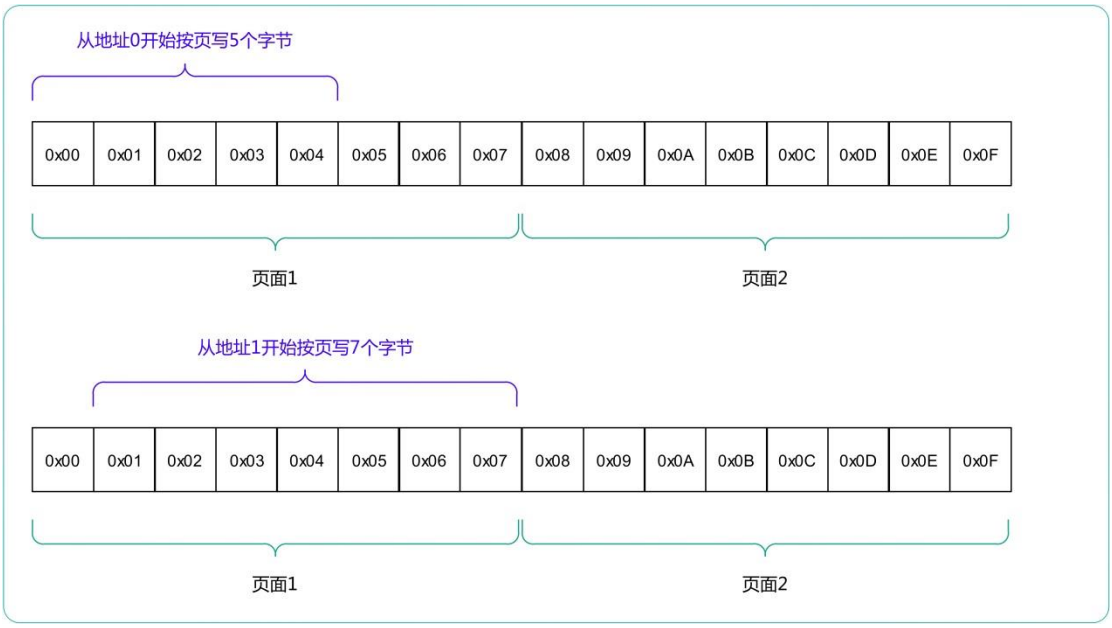

图 14：地址在同一个页面

按页写时如果地址跨页，会出现如下图所示的情形：我们期望从地址 0x04 开始连续写入“A B C D E F”6 个数据，但是实际写时，因为写地址增加到 0x07 后自动复位到 0x00，所以实际写入的地址 0x04~0x07 写入“A B C D”4 个数据，地址 0x00 和 0x01 写入“E”和“ F”2 个数据。

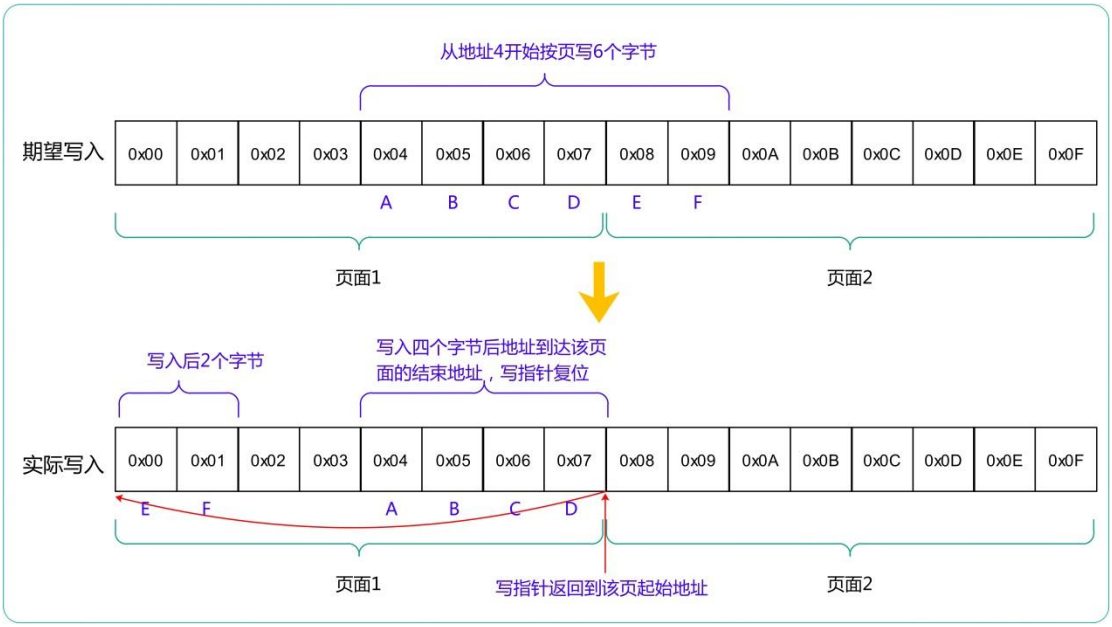

图 15：跨页后写指针复位

综上所述，按页写可实现连续写，不需要每个字节都发起一次写流程，这会有效节省操作时间，但每次只能写一个页面，也就是每次最多只能连续写入 8 个字节。

本例中，我们编写的按页写的函数代码清单如下。这里需要注意一下，按页写的函数本身没有实现跨页的处理，该函数是提供给批量写入函数调用的，批量写入函数中处理了跨页。

代码清单：按页写入

```
/*****************************************************************************
 * 功  能 : 按页写入数据。注意：该函数仅提供给批量写入函数(AT24C02_write_buf)
 *        : 和内存填充函数(AT24C02_fill)调用，目的是为了加快写入速度。
 *        : 其他文件调用该函数时，要确保写入的数据处于同一个页面
 * 参  数 : addr[in]：写入数据的起始地址
 *        : p_data[in]：指向待写入的数据缓存
 *        : len[in]：写入的数据长度，不能超过 1 个页面的大小 8 个字节
 * 返回值 : NRF_SUCCESS:写页面成功
 *****************************************************************************/
static u8 AT24C02_write_page(u8 addr,u8 * p_data,u8 len)
{
    //检查写入的数据长度是否合法
    if (len > AT24C02_PAGESIZE-(addr%AT24C02_PAGESIZE))
    {
       return ERROR_INVALID_LENGTH;
    }

    I2C_Start();                    //发送起始命令，产生起始条件
    I2C_SendData(AT24C02_ADDR_W);   //发送器件地址（写）
    I2C_WaitACK();                  //接收 ACK
    I2C_SendData(addr);             //发送写入数据的地址
    I2C_WaitACK();                  //接收 ACK
    while(len--)
    {
       I2C_SendData(*(p_data++));   //发送写入的数据
       I2C_WaitACK();               //接收 ACK
    }
    I2C_Stop();                     //发送停止命令，产生停止条件
    delay_ms(5);                    //AT24C02 的写入周期所需的最大时间是 5ms，延时 5ms 确保数据正确写入
    return AT24C02_SUCESS;
}
```

8. 批量写入：向指定地址顺序写入指定长度的数据

当写入的数据较多时，可以通过循环执行单个字节写入操作将数据写入到 eeprom，这种方法编写程序简单，但缺点是每个字节都需要执行一次写操作流程（起始条件→ 7 位地址 + 0（写）→写入数据的地址→数据→停止条件→延时等待 AT2402 写完成），这无疑会花费更多的时间。

更好的方法是使用按页写入的方式将数据写入到 eeprom，程序中将处于同一页面的数据通过按页写入的方式一次写入。这种方式的优点是速度快，但编程相对复杂一些，因为要处理地址跨页的问题。

批量写入的代码清单如下，代码中按照写入数据的地址逐个取出待写入的数据，并根据地址判断一个页面的数据是否取完，如果取完，则执行一次按页写入，如此反复，直到数据全部写入。

代码清单：批量写入

```
1.  /**************************************************************************
2.   * 功  能 : 向 AT24C02 指定的起始地址批量顺序写入数据，函数内部实现了跨页写，
3.   *        : 函数会检查 AT24C02 的空间是否能够容纳写入的数据。
4.   * 参  数 : WriteAddr[in]：写入数据的起始地址
5.   *        : p_buf[in]：指向待写入的数据缓存
6.   *        : size[in]：写入的数据长度
7.   * 返回值 : NRF_SUCCESS:写数据成功
8.   **************************************************************************/
9.  u8 AT24C02_write_buf(u8 *p_buf,u8 addr,u16 len)
10. {
11.    u8 addr_ptr = addr;
12.    u8 write_addr;
13.    u8 sendlen = 0;
14.    u8 tx_buf[AT24C02_PAGESIZE];
15.
16.    //AT24C02 剩余空间不够存放需要写入的数据，返回：长度无效的错误代码
17.    if ((AT24C02_SIZE-addr) < len)
18.    {
19.       return ERROR_INVALID_LENGTH;
20.    }
21.    write_addr = addr_ptr;
22.    while(len--)  //连续写入数据
23.    {
24.       if((addr_ptr%AT24C02_PAGESIZE) == 0)  //到达页面的起始地址
25.       {
26.          if(sendlen != 0)  //到达页面起始地址并且发送长度不等于 0，表示即将跨页
27.          {
28.             AT24C02_write_page(write_addr,tx_buf, sendlen);  //执行一次按页写入操作
29.             sendlen = 0;                                     //清零发送长度
30.             write_addr = addr_ptr;
31.          }
32.          tx_buf[sendlen++] = *(p_buf++);                     //数据保存到发送缓存 tx_buf
33.       }
34.       else
35.       {
36.          tx_buf[sendlen++] = *(p_buf++);                     //数据保存到发送缓存 tx_buf
37.          if(len==0)                                          //写入到最后的页面的数据读取完成
38.          {
```

```
            AT24C02_write_page(write_addr,tx_buf, sendlen);  //执行一次按页写入操作
            sendlen = 0;                                     //清零发送长度
          }
       }
       addr_ptr++;  //地址加 1
   }
   return AT24C02_SUCESS;
}
```

9. 批量读取：从指定地址顺序读取指定长度数据

AT24C02 从指定地址顺序读数据的时序如下图所示，顺序读相对于写要简单，因为不用考虑跨页的问题，顺序读时每读出一个字节数据地址指针会自动加 1。

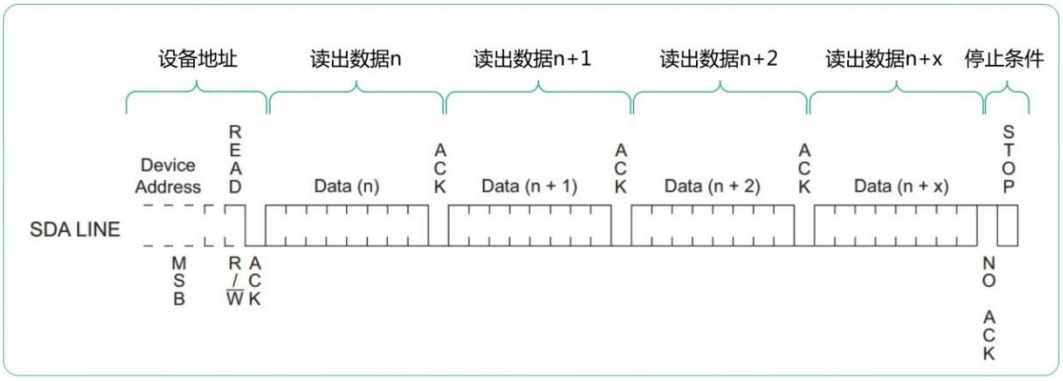

图 16：AT24C02 连续读时序

批量顺序读取的函数的代码清单如下。

代码清单：批量顺序读取

```
/*****************************************************************************
 * 功  能 : 从指定的起始地址顺序读出指定长度的数据
 * 参  数 : WriteAddr[in]：读出数据的起始地址
 *        : p_buf[in]：指向保存读出数据的缓存
 *        : size[in]：读出的数据长度
 * 返回值 : NRF_SUCCESS:读数据成功
 *****************************************************************************/
u8 AT24C02_read_buf(u8 *p_buf,u8 ReadAddr,u16 len)
{
   //读数据的长度已经超出了 AT24C02 的地址范围，返回：长度无效的错误代码
   if ((AT24C02_SIZE-ReadAddr) < len)
   {
      return ERROR_INVALID_LENGTH;
   }
   I2C_Start();                    //发送起始命令，产生起始条件
   I2C_SendData(AT24C02_ADDR_W);   //发送器件地址（写）
   I2C_WaitACK();                  //接收 ACK
   I2C_SendData(ReadAddr);         //发送读取数据的地址
   I2C_WaitACK();                  //接收 ACK
```

```
    I2C_Start();                    //发送起始命令，产生起始条件
    I2C_SendData(AT24C02_ADDR_R);   //发送器件地址（读）
    I2C_WaitACK();                  //接收 ACK
    //顺序读取数据
    while(len)
    {
       len--;
       *p_buf++ = I2C_RecvData();
       if(len == 0)I2C_Stop();      //如果是最后一个字节，发送停止命令，产生停止条件，否则发送 ACK
       else  I2C_SendACK();
    }
    return AT24C02_SUCESS;
}
```

AT24C02 是没有擦除指令的，写入数据前也不需要擦除的，这一点是和 Flash 不一样的。但在应用中，有时我们希望 AT24C02 的数据恢复为某个默认值，以便于区分有没有写入数据，鉴于此点，我们编写了一个全片填充函数，用于将全片填充为指定的数据，如全片填充“0xFF”，模拟 Flash 的擦除操作。

代码清单：全片填充指定的数据

```
/*****************************************************************************
 * 功  能 : AT24C02 全片填充指定的数据
 * 参  数 : dat[in]：填充的数据
 * 返回值 : 无
 *****************************************************************************/
void AT24C02_fill(u8 dat)
{
    u8 i;
    u8 tx_buf[AT24C02_PAGESIZE];
    //拷贝填充的数据
    for (i = 0; i < AT24C02_PAGESIZE; i++)tx_buf[i] = dat;

    //全片填充数据
    for (i = 0; i < AT24C02_PAGENUM; i++)
    {
       AT24C02_write_page(i*8,tx_buf,AT24C02_PAGESIZE);
    }
}
```

编写好读写函数后，我们就可以通过读写函数对 eeprom 进行读和写，下面的程序分别使用了批量写入、批量读取和全片填充访问 AT24C02，具体功能如下。

- 按下 S4 按键：从 AT24C02 地址 0x00 开始连续写入 256 个字节数据。
- 按下 S5 按键：从 AT24C02 地址 0x00 开始顺序读取 256 个字节数据，读出的数据通过串口输出。

- 按下 S8 按键：全片填充数据“0xFF”。

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,test_len;
    u8 j;

    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）
    P2M1 &= 0x1F;   P2M0 |= 0xE0;     //设置 P2.5、P2.6、P2.7 为推挽输出
    P0M1 &= 0x00;   P0M0 |= 0xFF;     //设置 P0.0 ~ P0.7 为推挽输出
    P3M1 &= 0xF3;   P3M0 &= 0xF3;     //设置 P3.2 ~ P3.3 为准双向口
    P4M1 &= 0xEB;   P4M0 &= 0xEB;     //设置 P4.2、P4.4 为准双向口

    SEG_off();          //控制 8 位数码管/点阵不显示
    ULN2003_off();      //控制 ULN2003 输出高电平，关闭蜂鸣器、继电器等
    leds_off();         //熄灭 D1~D8 指示灯
    delay_ms(10);       //延时
    uart1_init();                   //串口 1 初始化
    I2C_init();                     //IIC 初始化
    EA = 1;                         //使能总中断
    delay_ms(10);                   //初始化后延时
    test_len = 256;

    while(1)
    {
        btn_val=keyboard_scan();            //获取开发板矩阵按键检测值
        //按下 S4：从地址 0x0000 开始连续写入 256 个字节数据
        if(btn_val == BUTTON1_PRESSED)
        {
            j = 0;
            for(i=0;i<test_len;i++)my_tx_buf[i] = j++;
            printf("Write data to AT24C02!\r\n");
            //写入 256 个字节数据
            AT24C02_write_buf(my_tx_buf,0,test_len);
            //指示灯 D1 状态翻转，指示操作完成
            led_toggle(LED_1);
```

```
42.         }
43.         //按下 S5：从地址 0x0000 开始连续读出 256 个字节数据
44.         else if(btn_val == BUTTON2_PRESSED)
45.         {
46.             printf("Read data from fram!\r\n");
47.             //读取写入的数据
48.             AT24C02_read_buf(my_rx_buf,0,test_len);
49.             //串口打印读取的数据
50.             for(i=00;i<test_len;i++)printf("%02bX ",my_rx_buf[i]);
51.             printf("\r\n");
52.             led_toggle(LED_2);                  //指示灯 D2 状态翻转，指示操作完成
53.         }
54.         //按下 S8：全片擦除 FRAM，即向 FRAM 全片写入 0x00
55.         else if(btn_val == BUTTON5_PRESSED)
56.         {
57.             led_on(LED_3);                  //点亮指示灯 D3
58.             printf("Clear fram!\r\n");
59.             AT24C02_fill(0xFF);             //AT24C02 全片写入 0xFF
60.             led_off(LED_3);                 //熄灭指示灯 D3
61.         }
62.     }
63.}
```

4.1.3. 硬件连接

本实验程序的编写都是基于 IO 模式，所以 J24 端子需要使用短路帽将该端子第 1 引脚和第 2 引脚短接，即选择为 IO 模式。同时因为触发 AT24C02 存储器执行不同操作用到的是按键 S4、S5 和 S8，所以 J6 端子第 2 引脚和第 3 引脚需跳线帽短接，即选择矩阵按键。

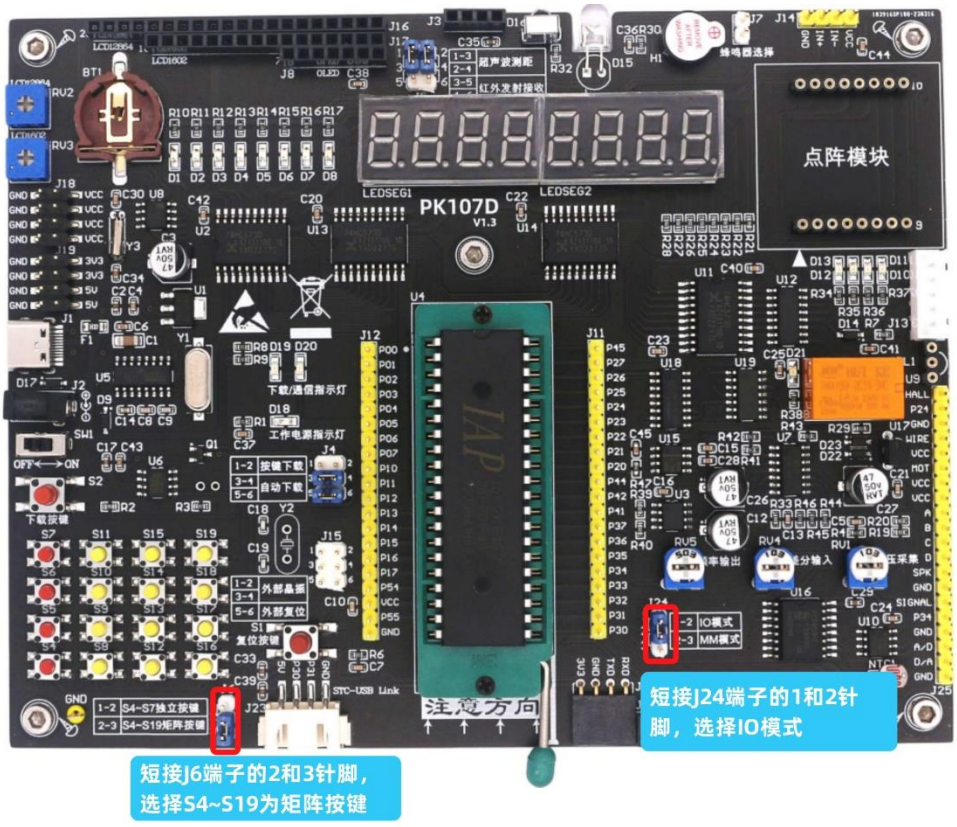

图 17：跳线帽短接

4.1.4. 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：模拟 I2C 读写 AT24C02 存储器”，将解压后得到的文件夹拷贝到合适的目录，如“D\STC15”（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“…\i2c_at24c02\project”目录下的工程文件“i2c_at24c02.uvproj”。
3) 点击编译按钮编译工程，编译成功后生成的 HEX 文件“i2c_at24c02.hex”位于工程的“…\i2c_at24c02\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：12 MHz。
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。
6) 程序运行后，先按下 S4 按键写入 256 个字节数据（写入的数据为 0x00 0x01…0xFF），之后按下 S5 按键读取数据并通过串口输出，可以观察到读取的数据和写入的数据一致。
7) 按下 S8 按键将全片填充数据“0xFF”，之后再按下 KEY2 按键读取数据，可以观察到读取的数据全部为“0xFF”。