stm32之IIC应用实例（AT24C02芯片，硬件和软件方式驱动）

2019年06月22日 09:56:47　脆弱的代码　阅读数 110　更多

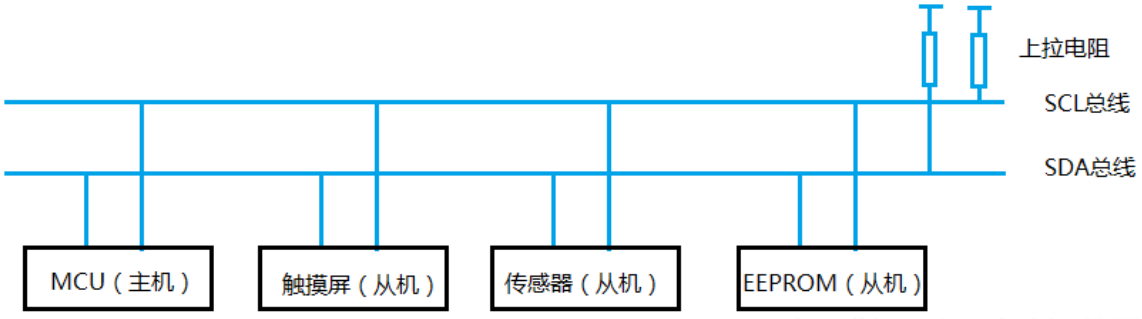
目录

写完回头一看发现字数还不少，如果你觉得文字太枯燥，那么可以跳到后面程序设计，直接动手做实验。如果想仔细了解关于IIC协议的细节，那么希望你能慢慢把看完，看完后一定有所收获。

概述：IIC BUS(Inter Integrated Circuit BUS,内部集成电路总线)是飞利浦公司推出的二线制串行扩展总线；在IIC总线上，只需要两条线——数据线SDA线和时钟线SCL；多个器件可连接到同一个IIC总线上，每一个器件有一个唯一的识别地址，可一对多、多对一、一对一通讯；标准模式下传输速度为100Kb/s，快速模式下为400Kb/s。优点：引脚少，硬件实现简单，成本低。

关于IIC的接口原理有些书可以描述出好几页，但有时候看完了可能未必能懂，虽然看的时候每句话都能理解，但是看完了之后，自己想描述出来却不知道从何说起，脑海里只是一些零散的概念。所以下面讲述一些关于IIC比较核心的东西。

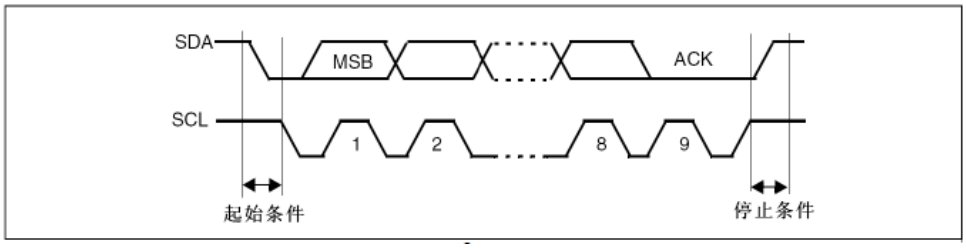
1.物理层：



1. 它是一种支持多个通信主机及多个通信从机的总线。“总线”是指多个设备共用的信号线。
2. 一个I^2C总线只使用两条总线线路；一条双向串行数据线(SDA)，一条串行时钟线（SCL）。
3. 每个连接到总线的设备都有一个独立的地址，总线上任一设备可以利用这个地址对其他设备进行交互。
4. 总线通过上拉电阻接到电源。总线上的任一设备，如果处于空闲状态时。会输出高阻态。所以在总线空闲的时候两条总线都呈现高电平。
5. 具有3种传输模式，标准模式(100kbps)、快速模式(400kbps)、高速模式(3.4Mbps，目前大多数不支持)。、
6. 连接到相同总线的设备数受到总线的最大电容400pF限制。

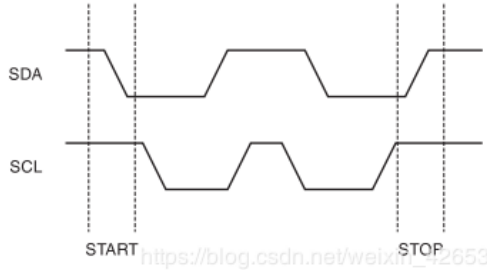
2.协议层：

图241　I^2C总线协议



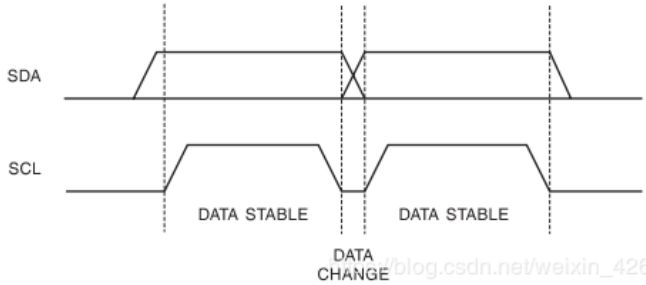
I^2C的协议定义了通信的起始和停止信号、数据有效性、响应、仲裁、时钟同步和地址广播等环节。

- **通讯的起始和停止信号**



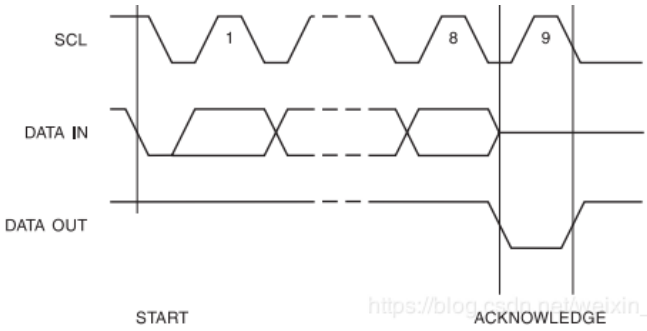
因为设备发送数据的时候是连续字节的，接受端并不能提前知道发送者要发送的总字节，所以接收端是从起始信号开始接收，直到停止信号为止；而且这两种信号要与传输过程中的任何时间点的时序不能有冲突（独特性），iic在数据传输的过程中也保证了这一点。**通讯的起始**：当SCL线是高电平时SDA线从高电平向低电平切换。**通信的停止**：当SCL线是高电平时SDA线由低电平向高电平切换。可看出，这两种信号跟数据发送过程中的差异，在数据发送过程中，SDA线总是在SCL线在低电平的时候才变换。

- **数据的有效性**



iic使用SDA信号线来传输数据，使用SCL时钟线进行数据同步。SDA数据线在SCL时钟线的每一个周期内传输一位，当SCL时钟线为高电平时，SDA数据线有效；即当SCL为高电平，若SDA为高电平表示数据“1”，若SDA为低电平表示数据“0”。当SCL时钟线为低电平时，SDA数据无线，一般这个时候SDA进行电平切换，为下一次表示数据进行准备。

- **应答位**

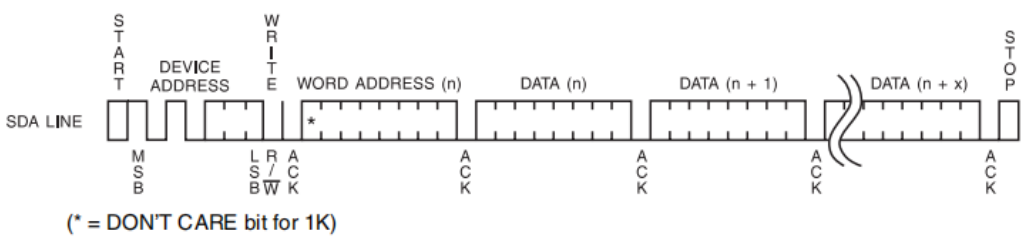


从机每接收到一个字节时，都要作出应答；而主机这时候释放SDA线的控制权，由数据接收端控制SDA，若SDA为高电平，表示非应答信号，低电平表示应答信号即传输成功。

3.数据的传输：

由上面介绍的协议基本理解了IIC通信的传输原理，在实际应用中，主机设备发送到从机设备的数据包括地址和数据。主机通过地址可找到对应的从机设备，而收到广播地址的从机做出判决，若地址与自身地址匹配则做出应答，若不匹配则忽略信息。

- 主机向从机写数据

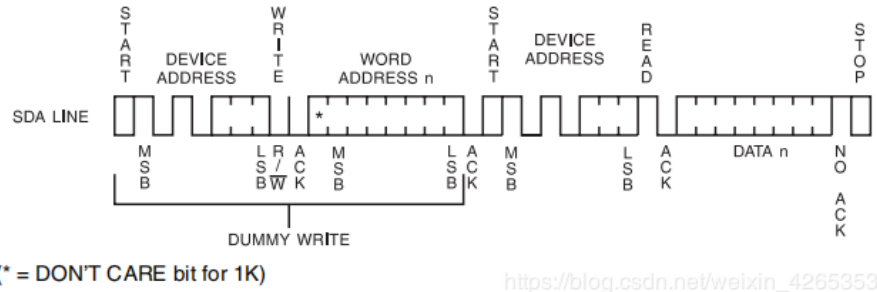


上面是主机对芯片AT24C02存储芯片进行写操作时SDA的数据流，可以很清晰看出所发送的内容。

> 对照着图来说，"DEVICE ADDRESS"设备地址可以是7位或10位，图中所示为7位，加上R/W传输方向位（1为发送，0位接受）组成一个字节；图中所有的ACK是等待从机应答位，主机收到了应答后才继续发送后面的内容，从图中可明显看出，主机每发送一个字节都必须等待从机做出一个应答；

> 图中的"WORD ADDRESS"指的是主机往AT24C02存储芯片写入的首地址，后面DATA就是写入的内容。不是所有的iic从设备都有这个，比如是音频的设备那么就不存在写入地址。具体是要根据iic从设备的数据手册上的功能通信协议来定。

- 主机向从机读数据



上面是主机对芯片AT24C02存储芯片进行读操作时SDA的数据流。

> 对照着图来说，首先是发送从机地址"DEVICE ADDRESS"，接着给从机发送"WORD ADDRESS"是所读取内容的地址，这是告诉AT24C02接下主机需要的内容，设置好地址之后，接着主机再发送一个起始位，紧接着发送"DEVICE ADDRESS"，这时可以注意到传输方向是READ（读），那么主机释放SDA线的控制权，有从机给主机发送数据，可以连续发送多个字节数据，当主机期望停止接收时作出一个非应答，那么从机就停止发送了。
>
> "WORD ADDRESS"不是所有主机向从机进行读操作时都要发送这个，具体是要看对方是什么设备，就如上面所说需要依据设备的数据手册。

个人总结一下，其实IIC是一个非常有意思的通信协议，起始位和停止位之间肯定有一个设备地址。对于一个在总线上的设备，它需要随时监听总线上的起始位和停止位，起始位一旦出现就要进行接收数据，不管地址是不是跟自己匹配，接下来都要关注停止位的出现，因为如果地址跟自己不匹配在停止位出现之前自己是不能占用总线的，虽说空闲的时候SDA和SCL必定都是高电平，但SDA和SCL都是高电平的时候总线未必空闲。

我们可以直接控制任意两个引脚，分别用作SCL和SDA，按照上述信号时序要求，就可以实现IIC通信。直接控制引脚需要CPU控制每个时刻的引脚状态，所以称之为“软件模拟协议”方式。

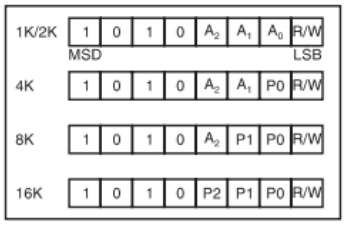
相对地，还有“硬件协议”方式，STM32的IIC片上外设专门负责实现IIC通信协议，只要配置好该外设，它就会自动根据协议要求产生通信信号，收发数据并缓存起来。CPU只要检测该外设的状态和访问数据寄存器，就能完成数据的收发。这种由硬件外设处理的IIC协议的方式减轻了CPU的工作，且是软件设计更简单。

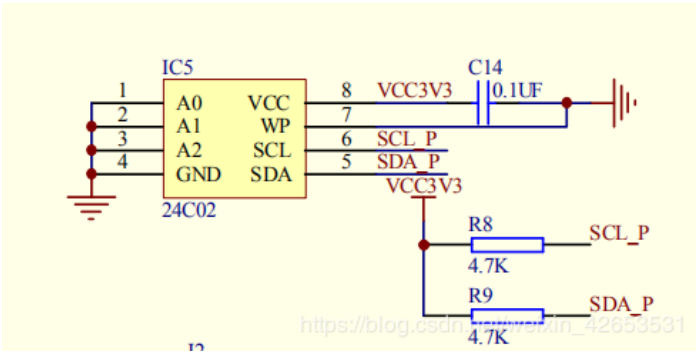
下面分别介绍这两种方式在STM32上的实现。

4.程序设计

通过查阅AT24C02的数据手册，该产品属于2k容量，2k指的是bit，所以等于256字节，需要一个8位数据字进行寻址。另外设备地址可以在数据手册里查到。

Figure 1. Device Address





由上面两图得知该设备地址为0xA0，第0bit是读写方向位。 这里SCL和SDA接了上拉电阻，然后分别接到stm32f103c8t6的PB6、PB7。

- **软件模式方式**

第一步，通过cpu控制io模拟i2c协议，主要实现起始通讯函数，停止通讯函数，发送字节函数和接受字节函数。在控制io是注意加入延时以满足i2c的时序。

创建<i2c.h>

```
#ifndef __I2C_H
#define __I2C_H
#include "stm32f10x.h"

#define IIC_NO_ACK  1
#define IIC_ACK     0

#define SCL_CLR()   GPIOB->BRR = GPIO_Pin_6
#define SCL_SET()   GPIOB->BSRR = GPIO_Pin_6
#define SDA_CLR()   GPIOB->BRR  = GPIO_Pin_7
#define SDA_SET()   GPIOB->BSRR  = GPIO_Pin_7
#define SCL_READ()  GPIOB->IDR  & GPIO_Pin_6
#define SDA_READ()  GPIOB->IDR  & GPIO_Pin_7

/*SCL时钟线*/
#define AT24C02_SCL_PIN  GPIO_Pin_4
#define AT24C02_SCL_PORT GPIOC
#define AR24C02_SCL_CLK  RCC_APB2Periph_GPIOC

/*SDA数据线*/
#define AT24C02_SCL_PIN  GPIO_Pin_4
#define AT24C02_SCL_PORT GPIOC
#define AR24C02_SCL_CLK  RCC_APB2Periph_GPIOC

void I2C_Configuration(void);
extern void I2C_Start(void); //发送开始信号
extern void I2C_Stop(void);  //发送停止信号
extern void I2C_Send_Byte(uint8_t sebyte); // I2C发送一字节数据
extern uint8_t I2C_Recieve_Byte(void);   // I2C接收一字节数据
#endif
```

创建<i2c.c>

```
#include "i2c.h"
#include "stm32f10x_gpio.h"

static void delay(unsigned char us) //大概延时
{
        uint8_t i = 10;
        while(us--)
        {
            while(i--);
        }
}
void I2C_Configuration(void) //i2c初始化引脚
{
  GPIO_InitTypeDef  GPIO_InitStructure;
  /* Configure I2C2 pins: PB6->SCL and PB7->SDA */
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);
  GPIO_InitStructure.GPIO_Pin =  GPIO_Pin_6 | GPIO_Pin_7;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_OD;
  GPIO_Init(GPIOB, &GPIO_InitStructure);
}

void I2C_Stop(void) //产生起始信号
{
        SDA_CLR();      //信号线置低
        delay(1);
        SCL_SET();      //时钟线置高
        delay(1);
        SDA_SET();      //信号线置高
        delay(1);
}
```

```
void I2C_Start(void) //产生停止信号
{
        SCL_SET();      //时钟线置高
        SDA_SET();      //信号线置高
        delay(1);
        SDA_CLR();      //信号线置低
        delay(1);
        SCL_CLR();      //时钟线置低
        delay(1);
}

static unsigned char IIC_Wait_Ack(void) //阻塞等待从机应答
{
        unsigned char ack=0;

        SCL_CLR();      //时钟线置低
        delay(1);
        SDA_SET();      //信号线置高
        delay(1);
        SCL_SET();      //时钟线置高
        delay(1);

        if(SDA_READ()){ //读取SDA的电平
            ack = IIC_NO_ACK;   //如果为1，则从机没有应答
        }
        else{
            ack = IIC_ACK; //如果为0，则从机应答
        }
        SCL_CLR();//时钟线置低
        delay(1);

        return ack;     //返回读取到的应答信息
}

void I2C_Send_Byte(uint8_t IIC_Byte)
{
        unsigned char i;  //定义变量
        for(i=0;i<8;i++) //for循环8次
        {
            SCL_CLR();  //时钟线置低，为传输数据做准备
            delay(1);
            if(IIC_Byte & 0x80) //读取最高位
                SDA_SET();
            else
                    SDA_CLR();
            IIC_Byte <<= 1;  //数据左移1位
            delay(1);
            SCL_SET(); //时钟线置高，产生上升沿，把数据发送出去
            delay(1);
        }
        SCL_CLR();      //时钟线置低
        delay(1);

        while(IIC_Wait_Ack());  //阻塞从机应答
}

uint8_t I2C_Recieve_Byte(void)
{
        uint8_t rebyte=0,i;

        SCL_CLR();      //时钟线置低
        delay(1);
        for(i=0;i<8;i++){
            SCL_SET(); //时钟线置高
            delay(1);
            rebyte=rebyte<<1;
            if(SDA_READ()){
                rebyte |= 0x01;
            }
            SCL_CLR();  //时钟线置低
            delay(2);
        }
```

```
        SDA_SET();
                        delay(1);
        SCL_SET();    //给从机应答
        delay(2);
        SCL_CLR();    //时钟线置低
        return rebyte;
}
```

第二步，AT24C02的数据手册，通过上面的I2C通讯接口跟AT24C02芯片进行数据交互，包括单字节读写，多字节读写函数接口的实现。上面已经提到该芯片大小是256字节，所以地址范围是0x00~0xFF，不可越界。

创建<AT24C02.h>

```
#ifndef  __AT24C02_H
#define  __AT24C02_H
#include "stm32f10x.h"


#define AT24C02_I2Cx I2C1    //AT24C02所用的iic外设
#define AT24C02_ADDR  0xA0   //设备地址

extern void AT24C02_Init(void); //初始化
extern uint8_t AT24C02_ReadOneByte(uint8_t ReadAddr);   //指定地址读取一个字节
extern void AT24C02_WriteOneByte(uint8_t WriteAddr,uint8_t DataToWrite);        //指定地址写入一个字节
extern void AT24C02_Write(uint8_t WriteAddr,uint8_t *Buffer,uint16_t Num);      //从指定地址开始写入指定长度的数据
extern void AT24C02_Read(uint8_t ReadAddr,uint8_t *Buffer,uint16_t Num);        //从指定地址开始读取指定长度的数据

#endif
```

创建<AT24C02.c>

```
#include "AT24c02.h"
#include "i2c.h"

static void delay(unsigned int us) //大概延时
{
    uint8_t i = 10;
    while(us--)
    {
         while(i--);
    }
}

void AT24C02_Init(void)
{
         I2C_Configuration();
}

void AT24C02_WriteOneByte(uint8_t WriteAddr, uint8_t DataToWrite)//向AT24C02指定的地址写入一个字节
{
    I2C_Start(); //发送起始信号
    I2C_Send_Byte(AT24C02_ADDR|0x00);  //设备地址且传输方向位设置为0
    delay(1);
    I2C_Send_Byte(WriteAddr);//发送地址
    I2C_Send_Byte(DataToWrite);     //发送字节
    I2C_Stop();//产生一个停止条件
    delay(100);// 这个延时绝对不能去掉
}
uint8_t AT24C02_ReadOneByte(uint8_t ReadAddr) //从AT24C02指定的地址读取一个字节
{
    uint8_t temp=0;

    I2C_Start();//发送起始信号
    I2C_Send_Byte(AT24C02_ADDR);  //设备地址 且传输方向位设置为0
    delay(1);
    I2C_Send_Byte(ReadAddr);   //发送地址
    I2C_Start();
    I2C_Send_Byte(AT24C02_ADDR|0x01);  //设备地址 且传输方向位设置为1

    delay(1);
```

```
    delay(1);
        temp=I2C_Recieve_Byte();   //接受一个字节
    I2C_Stop();//产生一个停止条件

    return temp;
}

void AT24C02_Read(uint8_t ReadAddr,uint8_t *Buffer,uint16_t Num)//从指定地址连续读取多个字节
{
        while(Num)
        {
            *Buffer++=AT24C02_ReadOneByte(ReadAddr++);
            Num--;
        }
}

void AT24C02_Write(uint8_t WriteAddr,uint8_t *Buffer,uint16_t Num)//向指定地址连续写入过个字节
{
        while(Num--)
        {
            AT24C02_WriteOneByte(WriteAddr,*Buffer);
            WriteAddr++;
            Buffer++;
        }
}
```

第三步，配置一个串口输出，用于打印调试信息，验证实验。

创建USART1.h

```
#ifndef __USART1_INIT_H__
#define __USART1_INIT_H__

#include "stm32f10x.h"
#include <stdio.h>

int fputc(int ch, FILE *f);
void USART1_Configuration(void);//打印输出串口初始化
void UART_send(uint8_t *Buffer, uint32_t Length);
uint8_t UART_recive(void);  //阻塞等待一个数据到来

#endif
```

创建USART1.c

```
#include "USART1.h"

 void USART1_Configuration(void)//打印输出串口初始化
 {
          GPIO_InitTypeDef GPIO_InitStructure;
          USART_InitTypeDef USART_InitStructure;
          NVIC_InitTypeDef NVIC_InitStructure;

          //配置串口1 （USART1） 时钟
          RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1 | RCC_APB2Periph_GPIOA, ENABLE);

          //配置串口1接收终端的优先级
          NVIC_PriorityGroupConfig(NVIC_PriorityGroup_0);
          NVIC_InitStructure.NVIC_IRQChannel = USART1_IRQn;
          NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 1;
          NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
          NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
          NVIC_Init(&NVIC_InitStructure);

          //配置串口1 发送引脚(PA.09)
          GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;
          GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
          GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
          GPIO_Init(GPIOA, &GPIO_InitStructure);
```

```
          //配置串口1 接收引脚 (PA.10)                    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;
          GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
          GPIO_Init(GPIOA, &GPIO_InitStructure);

          //串口1工作模式（USART1 mode）配置
          USART_InitStructure.USART_BaudRate = 115200;//一般设置为9600;
          USART_InitStructure.USART_WordLength = USART_WordLength_8b; //数据位为8个字节
          USART_InitStructure.USART_StopBits = USART_StopBits_1; //一位停止位
          USART_InitStructure.USART_Parity = USART_Parity_No ; //无校验位
          USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None; //不需要流控制
          USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx; //接收发送模式
          USART_Init(USART1, &USART_InitStructure);

          //USART_ITConfig(USART1, USART_IT_RXNE, ENABLE);//开启中断
          USART_Cmd(USART1, ENABLE);//使能串口
          USART_ClearFlag(USART1,0x3FF);
}

int fputc(int ch, FILE *f) //重定向c库里面的fputc到串口，那么使用printf时就能将打印的信息从串口发送出去，在PC上同串口助手接收信息
{
        while( USART_GetFlagStatus(USART1,USART_FLAG_TXE)!= SET);
        //将Printf内容发往串口
        USART_SendData(USART1, (unsigned char) ch);
        while( USART_GetFlagStatus(USART1,USART_FLAG_TC)!= SET);
        return (ch);
}
uint8_t UART_recive(void)
{
        while(USART_GetFlagStatus(USART1, USART_IT_RXNE) == RESET);
                        return USART_ReceiveData(USART1);
}

void UART_send(uint8_t *Buffer, uint32_t Length)
{
        while(Length != 0)
        {
           while( USART_GetFlagStatus(USART1,USART_FLAG_TXE)!= SET);
           USART_SendData(USART1, (unsigned char) *Buffer);
           while( USART_GetFlagStatus(USART1,USART_FLAG_TC)!= SET);
           Buffer++;
            Length--;
        }
}
```

第四步，首先读取AT24C02内从0x00地址开始连续读取11个数据并通过串口打印出来，然后再往该地址写入11个数据，进入while循环后，将从串口接收到的一个字节数据写入到AT24C02芯片内，并读取出来再通过串口打印出来。

创建<main.c>

```
#include "uart.h"
#include "i2c.h"
#include "at24c02.h"

uint8_t data[11],str[12]="hello world";

int main()
{
        uint8_t rece_data;

        UART_Init(9600); // 初始化串口
        I2C_Init(0); // 初始化I2C口

        AT24C02_Read(0x00,data,11);//上电读取AT24C02内地址0~10的数据
        UART_send(data,11);     //把读出的数据返回电脑串口

        AT24C02_Write(0x00,str,11);
        UART_send(data,11);     //把读出的数据返回电脑串口;
        while(1)
        {
                rece_data = UART_recive();// 等待串口发来的数据
```

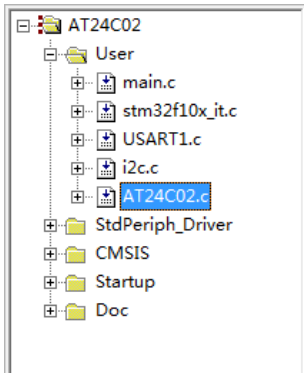
```
                AT24C02_WriteOneByte(0x00, rece_data);// 把串口发来的数据写入AT24C02地址0x00处

            rece_data = 0;// rece_data清零
            rece_data = AT24C02_ReadOneByte(0x00);// 读出AT24C02地址0x00处数据，赋予rece_data
            UART_send_byte(rece_data);// 把读出的数据返回电脑串口          }



}
```

将所有文件加入到工程中：



编译运行，下载到开发板上，连接上AT24C02芯片，接上串口调试转接口，那么就可以观察实验了。AT24C02储存芯片掉电数据不丢失，可以自行适当更改代码验证。

- 硬件协议方式

将调用标准库的函数进行IIC通信，所以将上面的i2c.c、i2c.h移除，只需要更换AT24C02.c文件就行。

创建<AT242C02.c>

```
#include "AT24c02.h"

static void delay(unsigned int us) //大概延时
{
    uint8_t i = 10;
    while(us--)
    {
        while(i--);
    }
}
void AT24C02_Init(void)
{
   I2C_InitTypeDef  I2C_InitStructure;
   GPIO_InitTypeDef  GPIO_InitStructure;

   RCC_APB1PeriphClockCmd(RCC_APB1Periph_I2C1,ENABLE); //使能iic外设时钟
   RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB | RCC_APB2Periph_AFIO , ENABLE); //使能GPIO时钟

   /* Configure I2C1 pins: PB6->SCL and PB7->SDA */
   GPIO_InitStructure.GPIO_Pin =  GPIO_Pin_6 | GPIO_Pin_7;
   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_OD;
   GPIO_Init(GPIOB, &GPIO_InitStructure);

   I2C_DeInit(I2C1);
   I2C_InitStructure.I2C_Mode = I2C_Mode_I2C;
   I2C_InitStructure.I2C_DutyCycle = I2C_DutyCycle_2;
   I2C_InitStructure.I2C_Ack = I2C_Ack_Enable;
   I2C_InitStructure.I2C_AcknowledgedAddress = I2C_AcknowledgedAddress_7bit;
   I2C_InitStructure.I2C_ClockSpeed = 100000;  /* 100K速度 */

   I2C_Cmd(I2C1, ENABLE);
   I2C_Init(I2C1, &I2C_InitStructure);
   /*允许1字节1应答模式*/
   I2C_AcknowledgeConfig(I2C1, ENABLE);
}
```

```
    static void I2C_AcknowledgePolling(I2C_TypeDef *I2Cx,uint8_t I2C_Addr)
{
  vu16 SR1_Tmp;

  do
  {
    I2C_GenerateSTART(AT24C02_I2Cx, ENABLE); /*起始位*/
    /*读SR1*/
    SR1_Tmp = I2C_ReadRegister(AT24C02_I2Cx, I2C_Register_SR1);
    /*器件地址(写)*/
    I2C_Send7bitAddress(AT24C02_I2Cx, I2C_Addr, I2C_Direction_Transmitter);

  }while(!(I2C_ReadRegister(AT24C02_I2Cx, I2C_Register_SR1) & 0x0002));

  I2C_ClearFlag(AT24C02_I2Cx, I2C_FLAG_AF);

  I2C_GenerateSTOP(AT24C02_I2Cx, ENABLE);  /*停止位*/
}



uint8_t AT24C02_ReadOneByte(uint8_t ReadAddr) //从AT24C02指定的地址读取一个字节
{
        uint8_t recvalue;

        while(I2C_GetFlagStatus(AT24C02_I2Cx, I2C_FLAG_BUSY));

        /*允许1字节1应答模式*/
        I2C_AcknowledgeConfig(AT24C02_I2Cx, ENABLE);


        /* 发送起始位 */
      I2C_GenerateSTART(AT24C02_I2Cx, ENABLE);
      while(!I2C_CheckEvent(AT24C02_I2Cx, I2C_EVENT_MASTER_MODE_SELECT));/*EV5,主模式*/
      /*发送器件地址(写)*/
      I2C_Send7bitAddress(AT24C02_I2Cx,  AT24C02_ADDR, I2C_Direction_Transmitter);
      while (!I2C_CheckEvent(AT24C02_I2Cx,                    I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED));

        /*发送地址*/
        I2C_SendData(AT24C02_I2Cx, ReadAddr);
    while (!I2C_CheckEvent(AT24C02_I2Cx, I2C_EVENT_MASTER_BYTE_TRANSMITTED));/*数据已发送*/

        /*起始位*/
        I2C_GenerateSTART(AT24C02_I2Cx, ENABLE);
        while(!I2C_CheckEvent(AT24C02_I2Cx, I2C_EVENT_MASTER_MODE_SELECT));

        /*器件读*/
        I2C_Send7bitAddress(AT24C02_I2Cx, AT24C02_ADDR, I2C_Direction_Receiver);
        while(!I2C_CheckEvent(AT24C02_I2Cx, I2C_EVENT_MASTER_RECEIVER_MODE_SELECTED));

   I2C_AcknowledgeConfig(AT24C02_I2Cx, DISABLE);        /* 最后一位后要关闭应答的 */
   I2C_GenerateSTOP(AT24C02_I2Cx, ENABLE);                      /* 发送停止位 */
         while(!I2C_CheckEvent(AT24C02_I2Cx, I2C_EVENT_MASTER_BYTE_RECEIVED));  /* EV7 */
            recvalue = I2C_ReceiveData(AT24C02_I2Cx);

        /* 再次允许应答模式 */
        I2C_AcknowledgeConfig(AT24C02_I2Cx, ENABLE);

        return recvalue;
}

void AT24C02_WriteOneByte(uint8_t WriteAddr, uint8_t DataToWrite)

{
    /* 起始位 */
        I2C_GenerateSTART(AT24C02_I2Cx, ENABLE);
        while(!I2C_CheckEvent(AT24C02_I2Cx, I2C_EVENT_MASTER_MODE_SELECT));

        /* 发送器件地址(写)*/
        I2C_Send7bitAddress(AT24C02_I2Cx, AT24C02_ADDR, I2C_Direction_Transmitter);
        while(!I2C_CheckEvent(AT24C02_I2Cx,     I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED));
```

```
        /*发送地址*/
        I2C_SendData(AT24C02_I2Cx, WriteAddr);
         while(!I2C_CheckEvent(AT24C02_I2Cx, I2C_EVENT_MASTER_BYTE_TRANSMITTED));

         /* 写一个字节*/
        I2C_SendData(AT24C02_I2Cx, DataToWrite);
        while(!I2C_CheckEvent(AT24C02_I2Cx, I2C_EVENT_MASTER_BYTE_TRANSMITTED));

        /* 停止位*/
        I2C_GenerateSTOP(AT24C02_I2Cx, ENABLE);

        I2C_AcknowledgePolling(AT24C02_I2Cx,AT24C02_ADDR);

          delay(100);

        return ;
}

void AT24C02_Write(uint8_t WriteAddr,uint8_t *Buffer,uint16_t Num)//向指定地址连续写入过个字节
{
        while(Num--)
        {
            AT24C02_WriteOneByte(WriteAddr,*Buffer);
            WriteAddr++;
            Buffer++;
        }
}
void AT24C02_Read(uint8_t ReadAddr,uint8_t *Buffer,uint16_t Num)//从指定地址连续读取多个字节
{
    if(Num==0)
        return ;

        while(I2C_GetFlagStatus(AT24C02_I2Cx, I2C_FLAG_BUSY));

        /*允许1字节1应答模式*/
        I2C_AcknowledgeConfig(AT24C02_I2Cx, ENABLE);


        /* 发送起始位 */
  I2C_GenerateSTART(AT24C02_I2Cx, ENABLE);
  while(!I2C_CheckEvent(AT24C02_I2Cx, I2C_EVENT_MASTER_MODE_SELECT));/*EV5,主模式*/
  /*发送器件地址(写)*/
  I2C_Send7bitAddress(AT24C02_I2Cx,  AT24C02_ADDR, I2C_Direction_Transmitter);
  while (!I2C_CheckEvent(AT24C02_I2Cx, I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED));

        /*发送地址*/
        I2C_SendData(AT24C02_I2Cx, ReadAddr);
    while (!I2C_CheckEvent(AT24C02_I2Cx, I2C_EVENT_MASTER_BYTE_TRANSMITTED));/*数据已发送*/

        /*起始位*/
        I2C_GenerateSTART(AT24C02_I2Cx, ENABLE);
        while(!I2C_CheckEvent(AT24C02_I2Cx, I2C_EVENT_MASTER_MODE_SELECT));

        /*器件读*/
        I2C_Send7bitAddress(AT24C02_I2Cx, AT24C02_ADDR, I2C_Direction_Receiver);
        while(!I2C_CheckEvent(AT24C02_I2Cx, I2C_EVENT_MASTER_RECEIVER_MODE_SELECTED));

        while (Num){
        if(Num==1)
        {
            I2C_AcknowledgeConfig(AT24C02_I2Cx, DISABLE);       /* 最后一位后要关闭应答的 */
            I2C_GenerateSTOP(AT24C02_I2Cx, ENABLE);                     /* 发送停止位 */
        }

        while(!I2C_CheckEvent(AT24C02_I2Cx,I2C_EVENT_MASTER_BYTE_RECEIVED));  /* EV7 */
          *Buffer = I2C_ReceiveData(AT24C02_I2Cx);
          Buffer++;
            /* Decrement the read bytes counter */
          Num--;
        }
        /* 再次允许应答模式 */
```

```
        I2C_AcknowledgeConfig(AT24C02_I2Cx, ENABLE);

        return;
}
```

更改完成下载调试，同样跟软件模拟方式达到一样的效果。

完结。