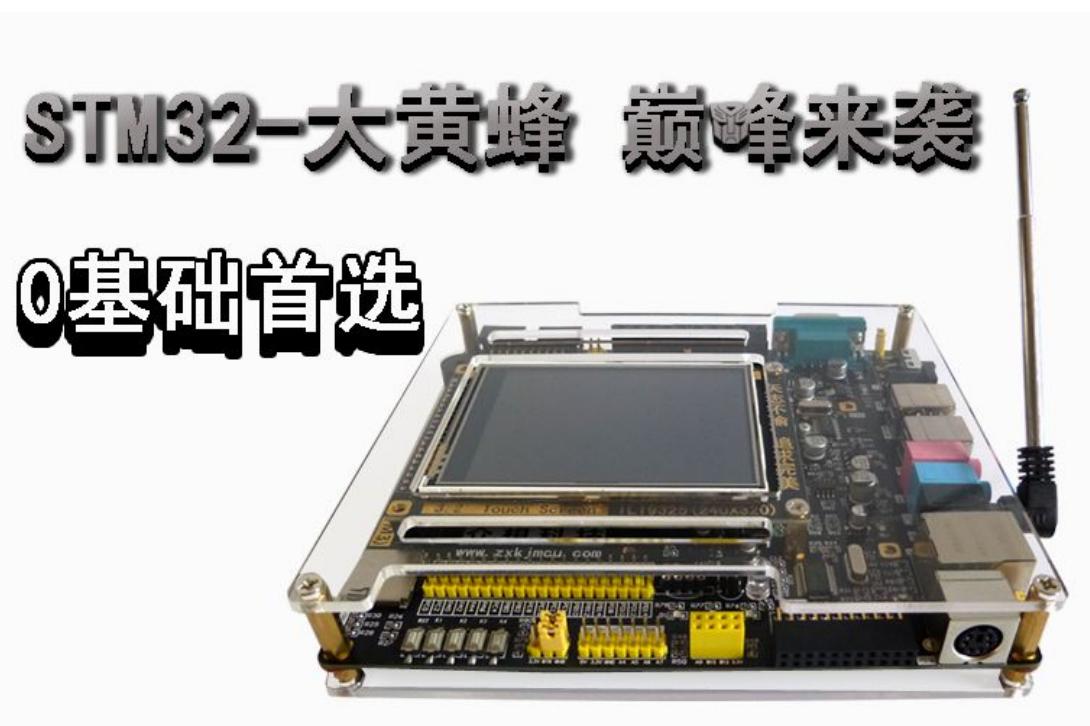


学 ARM 从 STM32 开始

STM32 开发板库函数教程—实战篇



官方网站: <http://www.zxkjmcu.com>

官方店铺: <http://zxkjmcu.taobao.com>

官方论坛: <http://bbs.zxkjmcu.com>

刘洋课堂: <http://school.zxkjmcu.com>

4.21 STM32 I²C 工作原理

4.21.1 概述

I²C(Inter—Integrated Circuit)总线是一种由 PHILIPS 公司开发的两线式串行总线，是具备多主机系统所需要的总线裁决和高低速器件同步功能的高性能串行总线。用于连接微控制器及其外围设备。I²C 总线产生于在 80 年代，最初为音频和视频设备开发，如今主要在服务器管理中使用很多。

4.21.1.1 I²C 总线特点

I²C 总线最主要的优点是其简单性和有效性。由于接口直接在设备之上，因此 I²C 总线占用的空间非常小，减少了电路板的空间和芯片管脚的数量，降低了互联成本。总线的长度可高达 25 英尺，并且能够以 10Kbps 的最大传输速率支持 40 个组件。I²C 总线的另一个优点是，它支持多主控(multimastering)，其中任何能够进行发送和接收的设备都可以成为总线。一个主控能够控制信号的传输和时钟频率。当然，在任何时间点上只能有一个主控。

4.21.1.2 I²C 总线工作原理

2.1 总线的构成及信号类型

I²C 总线是由数据线 SDA 和时钟 SCL 构成的串行总线，可发送和接收数据。在 CPU 与被控 IC 之间、IC 与 IC 之间进行双向传送，最高传送速率 100kbps。各种被控制电路均并联在这条总线上，但就像电话机一样只有拨通各自的号码才能工作，所以每个电路和设备模块都有唯一的地址，在信息的传输过程中，I²C 总线上并接的每一设备模块电路既是主控器(或被控器)，又是发送器(或接收器)，这取决于它所要完成的功能。CPU 发出的控制信

号分为地址码和控制量两部分，地址码用来选址，即接通需要控制的电路，确定控制的种类；控制量决定该调整的类别（如对比度、亮度等）及需要调整的量。这样，各控制电路虽然挂在同一条总线上，却彼此独立，互不相关。

I²C 总线在传送数据过程中共有三种类型信号，它们分别是：起始信号、终止信号和应答信号。

- 起始信号：SCL 为高电平时，SDA 由高电平向低电平跳变，开始传送数据；
- 终止信号：SCL 为低电平时，SDA 由低电平向高电平跳变，结束传送数据；
- 应答信号：接收数据的 IC 在接收到 8bit 数据后，向发送数据的 IC 发出特定的低电平脉冲，表示已收到数据。CPU 向受控单元发出一个信号后，等待受控单元发出一个应答信号，CPU 接收到应答信号后，根据实际情况作出是否继续传递信号的判断。若未收到应答信号，由判断为受控单元出现故障。

目前有很多半导体集成电路上都集成了 I²C 接口。带有 I²C 接口的单片机有：STM32F10XX 系列，PHILIPSP87LPC7XX 系列，MICROCHIP 的 PIC16C6XX 系列等。很多外围器件如存储器、监控芯片等也提供 I²C 接口。

4.21.2 STM32 I²C 总线模式说明

I²C 规程运用主/从双向通讯。器件发送数据到总线上，则定义为发送器，器件接收数据则定义为接收器。主器件和从器件都可以工作于接收和发送状态。总线必须由主器件（通常为微控制器）控制，主器件产生串行时钟（SCL）控制总线的传输方向，并产生起始和终止条件。SDA 线上的数据状态仅在 SCL 为低电平的期间才能改变，SCL 为高电平的期间，SDA 状态的改变被用来表示起始和终止条件。参见图 1 串行总线上的数据传送时序。

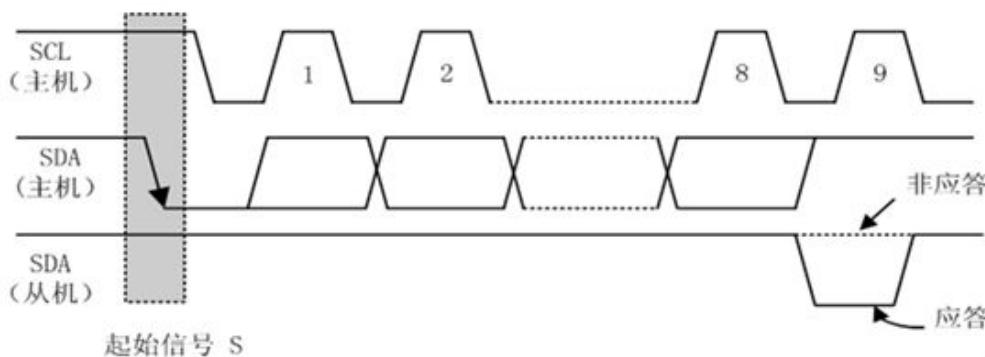


图 1 串行总线上的数据传送时序

4.21.2.1 STM32 I²C 接口可以下述 4 种模式中的一种运行：

- 从发送器模式
- 从接收器模式
- 主发送器模式
- 主接收器模式

该模块默认地工作于从模式。接口在生成起始条件后自动地从从模式切换到主模式；当仲裁丢失或产生停止信号时，则从主模式切换到从模式。允许多主机功能。

4.21.2.2 通信流

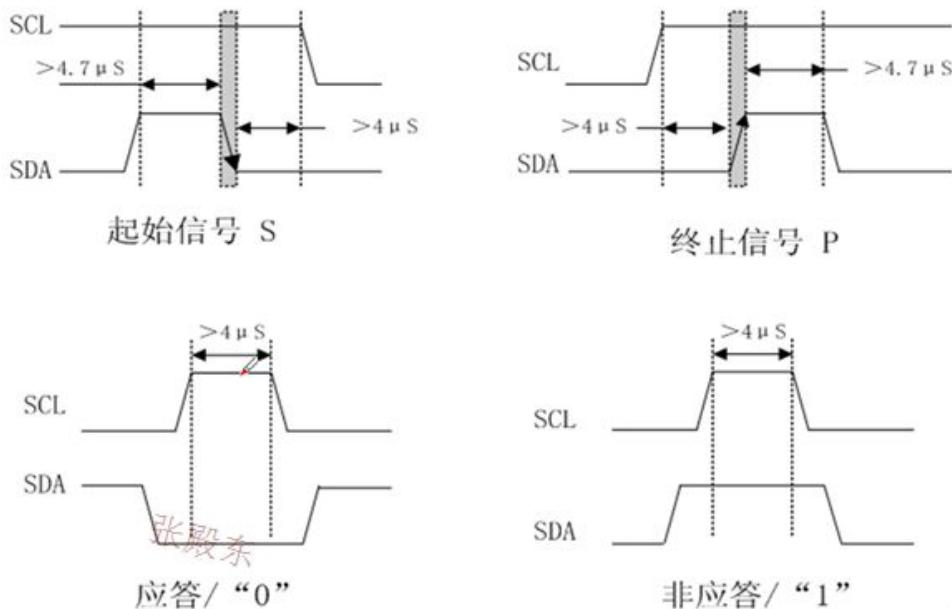
主模式时，I²C 接口启动数据传输并产生时钟信号。串行数据传输总是以起始条件开始并以停止条件结束。起始条件和停止条件都是在主模式下由软件控制产生。

从模式时，I²C 接口能识别它自己的地址(7 位或 10 位)和广播呼叫地址。软件能够控制开启或禁止广播呼叫地址的识别。

数据和地址按 8 位/字节进行传输，高位在前。跟在起始条件后的 1 或 2 个字节是地址(7 位模式为 1 个字节，10 位模式为 2 个字节)。地址只在主模式

发送。

在一个字节传输的 8 个时钟后的第 9 个时钟期间，接收器必须回送一个应答位(ACK)给发送器。参考下图。



图二 I²C 总线协议

4.21.3 实验目的

使用 STM32 芯片模拟 I²C 总线通讯格式，写出程序实际框架，为后续的硬件实验打好坚实的基础。

4.21.4 硬件设计

利用实验板上的 CPU 电路，通过程序设计可以很方便的实现 I²C 通讯功能。

图 4.21.4 CPU 端子图

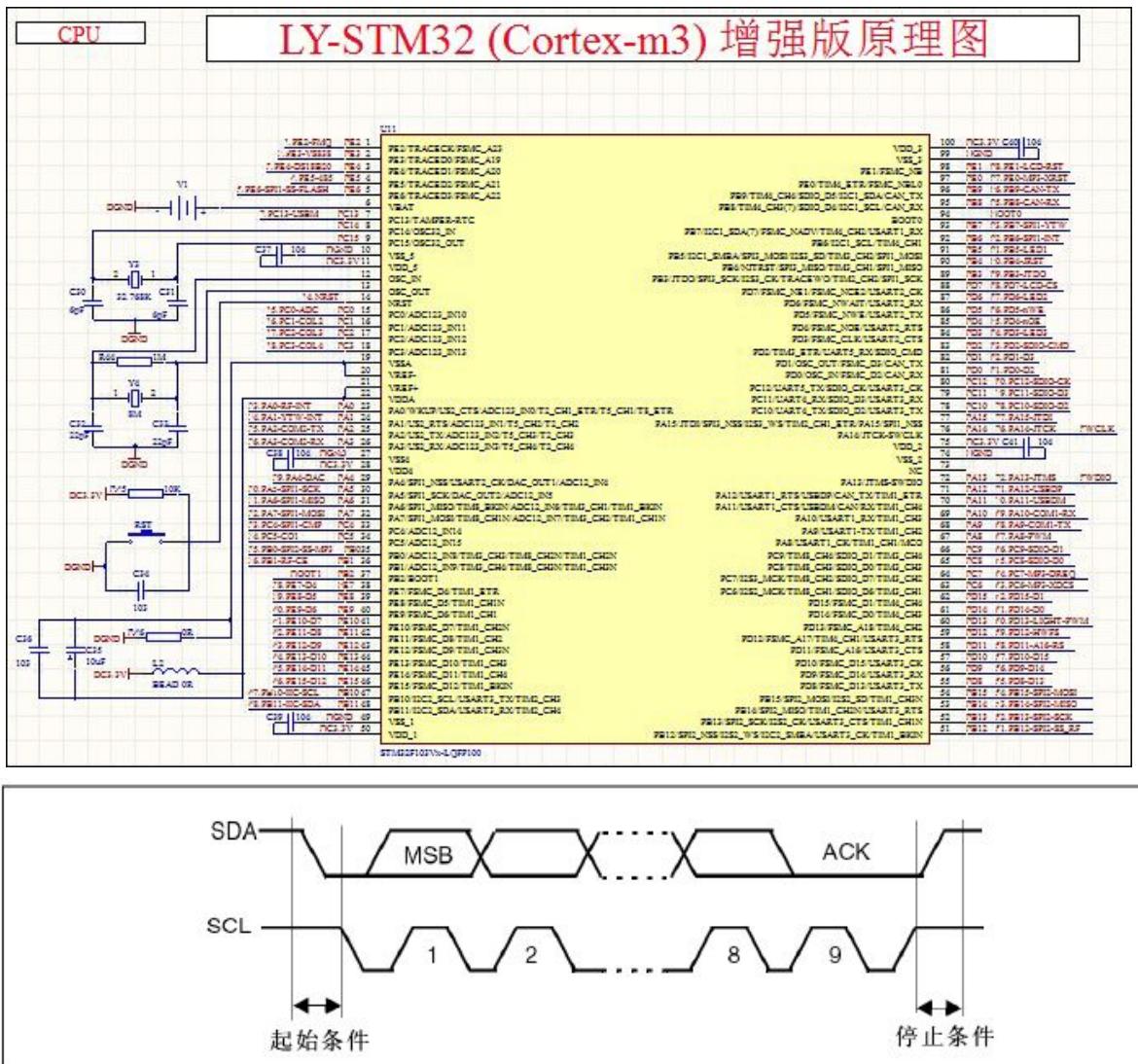
4.21.4.1 I²C 时序图

起始条件：

在 SCL 保持为高电平的条件下，SDA 由高变低，并保持 4 微秒以上。

1、停止条件：

在 SCL 保持为高电平的条件下，SDA 由低变高，并保持 4 微秒以上。

图 4.21.4.1 I²C 时序图

4.21.5 软件设计

4.21.5.1 软件设计说明

这次试验是先把 ID 号读出来，并打印输出，送到显示器显示出来。设计思路是这样的：

- 1、读取芯片中的 ID 号，定义数组，存放读取的 ID 号；
- 2、打印输出到显示器；
- 3、和设置好的 ID 号比较；

4.21.5.2 STM32 库函数文件

```
stm32f10x_gpio.c  
stm32f10x_rcc.c  
Misc.c // 中断控制字（优先级设置）库函数  
stm32f10x_exti.c // 外部中断库处理函数  
stm32f10x_tim.c // 定时器库处理函数  
stm32f10x_usart.c // 串口通讯函数
```

本节实验及以后的实验我们都是用到库文件，其中 `stm32f10x_gpio.h` 头文件包含了 GPIO 端口的定义。`stm32f10x_rcc.h` 头文件包含了系统时钟配置函数以及相关的外设时钟使能函数，所以我们要把这两个头文件对应的 `stm32f10x_gpio.c` 和 `stm32f10x_rcc.c` 加到工程中；`Misc.c` 库函数主要包含了中断优先级的设置，`stm32f10x_exti.c` 库函数主要包含了外部中断设置参数，`stm32f10x_tim.c` 库函数主要包含定时器设置，`stm32f10x_usart.c` 库函数主要包含串行通讯设置，这些函数也要添加到函数库中。以上库文件包含了本次实验所有要用到的函数使用功能。

4.21.5.3 自定义头文件

```
pbdata.h  
pbdata.c
```

同时我们自己也创建了两个公共的文件，这两个文件主要存放我们自定

义的公共函数和全局变量，以方便以后每个功能模块之间传递参数。

4.21.5.4 pbdata.h 文件里的内容是

```
#ifndef _pbdata_H
#define _pbdata_H //定义头文件
#include "stm32f10x.h"
#include "misc.h"
#include "stm32f10x_exti.h"
#include "stm32f10x_tim.h"
#include "stm32f10x_usart.h"
#include "stdio.h"
#include "I2C.h"

extern u8 dt;//定义变量

void RCC_HSE_Configuration(void); //定义函数
void delay(u32 nCount);
void delay_us(u32 nus);
void delay_ms(u16 nms);

#endif
```

语句 `#ifndef`、`#endif` 是为了防止 `pbdata.h` 文件被多个文件调用时出现错误提示。如果不加这两条语句，当两个文件同时调用 `pbdata` 文件时，会提示重复调用错误。

4.21.5.5 pbdata.c 文件里的内容是

```
#include "pbdata.h" //很重要，引用这个头文件

u8 dt=0;

void RCC_HSE_Configuration(void) //HSE 作为 PLL 时钟，PLL 作为 SYSCLK
{
    RCC_DeInit(); /*将外设 RCC 寄存器重设为缺省值 */
    RCC_HSEConfig(RCC_HSE_ON); /*设置外部高速晶振（HSE） HSE 晶振打开(ON)*/

    if(RCC_WaitForHSEStartUp() == SUCCESS) { /*等待 HSE 起振， SUCCESS: HSE 晶振稳定且就绪*/

```

```
RCC_HCLKConfig(RCC_SYSCLK_Div1); /*设置 AHB 时钟(HCLK) RCC_SYSCLK_Div1——
AHB 时钟 = 系统时*/
RCC_PCLK2Config(RCC_HCLK_Div1); /*设置高速 AHB 时钟(PCLK2) RCC_HCLK_Div1——
APB2 时钟 = HCLK*/
RCC_PCLK1Config(RCC_HCLK_Div2); /*设置低速 AHB 时钟(PCLK1) RCC_HCLK_Div2——
APB1 时钟 = HCLK / 2*/
RCC_PLLConfig(RCC_PLLSource_HSE_Div1, RCC_PLLMul_9); /*设置 PLL 时钟源及倍频
系数*/
RCC_PLLCmd(ENABLE); /*使能 PLL */
while(RCC_GetFlagStatus(RCC_FLAG_PLLRDY) == RESET); /*检查指定的 RCC 标志
位(PLL 准备好标志)设置与否*/
RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK); /*设置系统时钟(SYSCLK) */
while(RCC_GetSYSCLKSource() != 0x08); /*0x08: PLL 作为系统时钟 */
}
}
void delay(u32 nCount)
{
    for(;nCount!=0;nCount--);
}
/*****************
* 名 称: delay_us(u32 nus)
* 功 能: 微秒延时函数
* 入口参数: u32 nus
* 出口参数: 无
* 说 明:
* 调用方法: 无
********************/
void delay_us(u32 nus)
{
    u32 temp;
    SysTick->LOAD = 9*nus;
    SysTick->VAL=0X00;//清空计数器
    SysTick->CTRL=0X01;//使能, 减到零是无动作, 采用外部时钟源
    do
    {
        temp=SysTick->CTRL;//读取当前倒计数值
    }while((temp&0x01)&&(! (temp&(1<<16))));//等待时间到达

    SysTick->CTRL=0x00; //关闭计数器
    SysTick->VAL =0X00; //清空计数器
}
```

```
/*
 * 名    称: delay_ms(u16 nms)
 * 功    能: 毫秒延时函数
 * 入口参数: u16 nms
 * 出口参数: 无
 * 说    明:
 * 调用方法: 无
 */
void delay_ms(u16 nms)
{
    u32 temp;
    SysTick->LOAD = 9000*nms;
    SysTick->VAL=0X00;//清空计数器
    SysTick->CTRL=0X01;//使能, 减到零是无动作, 采用外部时钟源
    do
    {
        temp=SysTick->CTRL;//读取当前倒计数值
    } while((temp&0x01)&&(!(temp&(1<<16)));//等待时间到达
    SysTick->CTRL=0x00; //关闭计数器
    SysTick->VAL =0X00; //清空计数器
}
```

4.21.6 STM32 系统用户文件 I2C.c 文件里的内容

在这个工程中我们先建一个独立的用户文件，起名 I2C.c。为了便于以后查找和移植，每种独立功能的外设都按照特定功能建立用户文件。在 I2C.c 文件中要配置函数的工作状态，比如输出形式、震荡频率等。这章节中主要讲述的是模拟出 I2C 总线信号，所以所有的程序书写都要以 I2C 总线的时序图为准，严格遵守 I2C 总线的时序要求，就会很正确的模拟出 I2C 总线工作时序。

```
#include "pbdata.h"

void I2C_Init(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;

    GPIO_InitStructure.GPIO_Pin=I2C_SCL|I2C_SDA;//把 I2C 总线配置成如下形
式
    GPIO_InitStructure.GPIO_Speed=GPIO_Speed_50MHz;
```

```
GPIO_InitStructure.GPIO_Mode=GPIO_Mode_Out_PP;
GPIO_Init(GPIOB, &GPIO_InitStructure);

I2C_SCL_H;//把 I2C_SCL 配置成高电平, 初始化部分
I2C_SDA_H;//把 I2C_SDA 配置成高电平, 初始化部分
}

void I2C_SDA_OUT(void)//把 I2C_SDA 配置成输出方向
{
    GPIO_InitTypeDef GPIO_InitStructure;

    GPIO_InitStructure.GPIO_Pin=I2C_SDA;
    GPIO_InitStructure.GPIO_Speed=GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode=GPIO_Mode_Out_PP;
    GPIO_Init(GPIOB, &GPIO_InitStructure);
}

void I2C_SDA_IN(void)//把 I2C_SDA 配置成输入方向
{
    GPIO_InitTypeDef GPIO_InitStructure;

    GPIO_InitStructure.GPIO_Pin=I2C_SDA;
    GPIO_InitStructure.GPIO_Mode=GPIO_Mode_IPU;
    GPIO_Init(GPIOB, &GPIO_InitStructure);
}

//产生起始信号, 根据时序图编写程序
void I2C_Start(void)
{
    I2C_SDA_OUT();//调用 SDA 输出方向函数

    I2C_SDA_H;
    I2C_SCL_H;
    delay_us(5); //延时 5 微秒
    I2C_SDA_L; //SDA 由高电平变低电平
    delay_us(6); //延时 6 微秒
    I2C_SCL_L; //SCL 由高电平变低电平
}

//产生停止信号
void I2C_Stop(void)
{
    I2C_SDA_OUT();//调用 SDA 输出方向函数
```

```
I2C_SCL_L;
I2C_SDA_L;
I2C_SCL_H; //SCL 由低电平变高电平
delay_us(6); //延时 6 微秒
I2C_SDA_H; //SDA 由低电平变高电平
delay_us(6); //延时 6 微秒
}

//主机产生应答信号 ACK
void I2C_Ack(void)
{
    I2C_SCL_L;
    I2C_SDA_OUT();
    I2C_SDA_L;
    delay_us(2);
    I2C_SCL_H;
    delay_us(5);
    I2C_SCL_L;
}

//主机不产生应答信号 NACK
void I2C_NAck(void)
{
    I2C_SCL_L;
    I2C_SDA_OUT();
    I2C_SDA_H;
    delay_us(2);
    I2C_SCL_H;
    delay_us(5);
    I2C_SCL_L;
}

//等待从机应答信号
//返回值：1 接收应答失败
//        0 接收应答成功
u8 I2C_Wait_Ack(void)
{
    u8 tempTime=0;

    I2C_SDA_IN();

    I2C_SDA_H;
    delay_us(1);
    I2C_SCL_H;
    delay_us(1);
```

```
while(GPIO_ReadInputDataBit(GPIO_I2C, I2C_SDA)) //读取 SDA 信号线状态
{
    tempTime++;
    if(tempTime>250) //如果从机应答，就跳出 while 循环
    {
        I2C_Stop();
        return 1; //如果从机无应答，就循循环，到达设定的时间后强制退出，并返回数据 1
    }
}

I2C_SCL_L; //跳出 while 循环后，把 SCL 线变低
return 0; //返回数据 0
}

//I2C 发送一个字节，通过时序图我们知道，当 SCL 是低电平的时候，我们可以往 SDA 上输出数据，当 SCL 变为高电平的时候，这时数据要保持不变，这时从机就可以读取总线上的数据。
void I2C_Send_Byte(u8 txd)
{
    u8 i=0;

    I2C_SDA_OUT();
    I2C_SCL_L; //拉低时钟开始数据传输

    for(i=0;i<8;i++)
    {
        if((txd&0x80)>0) //0x80 1000 0000 发送高位数据,
            I2C_SDA_H;
        else
            I2C_SDA_L;

        txd<<=1;
        I2C_SCL_H;
        delay_us(2); //发送数据
        I2C_SCL_L;
        delay_us(2);
    }
}

//I2C 读取一个字节

u8 I2C_Read_Byte(u8 ack)
```

```
{  
    u8 i=0, receive=0;  
  
    I2C_SDA_IN();  
    for(i=0;i<8;i++)  
    {  
        I2C_SCL_L;  
        delay_us(2);  
        I2C_SCL_H;  
        receive<<=1;  
        if(GPIO_ReadInputDataBit(GPIO_I2C, I2C_SDA))  
            receive++;  
        delay_us(1);  
    }  
  
    if(ack==0)  
        I2C_NAck();  
    else  
        I2C_Ack();  
  
    return receive;  
}
```

4.21.7 STM32 系统用户文件 I2C.h 文件里的内容

在前面我们先建一个独立的 I2C.c 用户文件，接着还要建立 I2C.h 文件，它两是成对创建的。同样的到了为了便于以后查找和移植，每种独立功能的外设都按照特定功能建立用户文件。

```
#ifndef _I2C_H  
#define _I2C_H  
#include "pbdata.h"  
  
//如果移植程序时只要改一下三个地方就行了  
#define I2C_SCL GPIO_Pin_10  
#define I2C_SDA GPIO_Pin_11  
#define GPIO_I2C GPIOB  
  
#define I2C_SCL_H GPIO_SetBits(GPIO_I2C, I2C_SCL) //这条语句中传递两个参数，  
传递 GPIOB 端口中的 GPIO_Pin_10 管脚，把它置高  
#define I2C_SCL_L GPIO_ResetBits(GPIO_I2C, I2C_SCL) //这条语句中传递两个参数，  
传递 GPIOB 端口中的 GPIO_Pin_10 管脚，把它置低
```

```
#define I2C_SDA_H GPIO_SetBits(GPIO_I2C, I2C_SDA)//这条语句中传递两个参数,  
传递 GPIOB 端口中的 GPIO_Pin_11 管脚, 把它置高  
#define I2C_SDA_L GPIO_ResetBits(GPIO_I2C, I2C_SDA)  
  
void I2C_Init(void);  
void I2C_SDA_OUT(void);  
void I2C_SDA_IN(void);  
void I2C_Start(void);  
void I2C_Stop(void);  
void I2C_Ack(void);  
void I2C_NAck(void);  
u8 I2C_Wait_Ack(void);  
void I2C_Send_Byte(u8 txd);  
u8 I2C_Read_Byte(u8 ack);  
#endif
```

4.21.8 STM32 系统时钟配置 SystemInit()

每个工程都必须在开始时配置并启动 STM32 系统时钟。

4.21.9 GPIO 引脚时钟使能

```
SystemInit()//72m  
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);  
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);  
RCC_APB1PeriphClockCmd(RCC_APB2Periph_USART1, ENABLE);//设置串口 1 时钟使能  
RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO, ENABLE);//功能复用 IO 时钟使能
```

本节实验用到了 PBA 端口，所以要把 PB 端口的时钟打开；串口 1 时钟源是通过 APB2 预分频器得到的，串口 1 时钟初始化；因为要与外部芯片通讯，所以要打开功能复用时钟。

4.21.10 GPIO 管脚电平控制函数

在主程序中采用 while(1) 循环语句，等待串口通讯中断的到来。

```
while(1);
```

4.21.11 stm32f10x_it.c 文件里的内容是

在中断处理 stm32f10x_it.c 文件里中仅串口 1 子函数非空，进入中断处理函数后，只有串口 1 有参数输出。

```
#include "stm32f10x_it.h"
#include "stm32f10x_exti.h"
#include "stm32f10x_rcc.h"
#include "misc.h"
#include "pbdata.h"

void NMI_Handler(void)
{
}

void USART1_IRQHandler(void)
{
    if(USART_GetITStatus(USART1, USART_IT_RXNE) !=RESET)
    {
        USART_SendData(USART1, USART_ReceiveData(USART1));
        while(USART_GetFlagStatus(USART1, USART_FLAG_TXE)==RESET);
    }
}
```

4.21.12 程序下载

在这一章节中要掌握模拟 I²C 总线时序，大家做到熟悉掌握程序就可以，模拟 I²C 总线功能要想在实验板上直观的体现出来还需要其他的外设配合，在下一节中我们要通过写 24C02 存储芯片实验，让初学者体验出 I²C 总线时序的模拟效果。

本节实验的源代码在光盘中：(LY-STM32 光盘资料\1. 课程\2. 外设篇\外设篇 02. STM32 模拟 I2C 通讯\程序)

4.21.13 实验效果图

和下一节课节写入 24C02 存储芯片（外部存储器）配合实现。