目 录

15 通道自校正电容式触摸感应芯片

概述

XW15A 是 15 按键的电容式触摸感应芯片，可替代机械式轻触按键，实现一体式密封美观的外观，

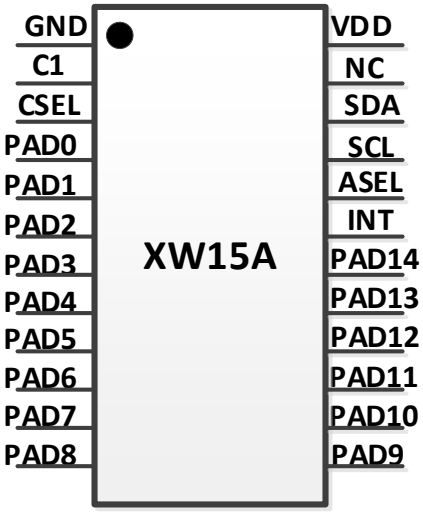
应用

- 用于电视机、音响、显示器、玩具等家电和娱乐设备与工业控制设备

特点

- 极高的灵敏度，可穿透 13mm 的玻璃，感应到手指的触摸
- 超强的抗干扰和 ESD 能力,不加任何器件即可通过人体 8000v 试验
- 内置按键消抖,无需软件再消抖
- 外围寄生电容自动校正
- 多通道公用灵敏度电容
- 工作电压范围：2.9～5.5V
- SSOP24 环保封装

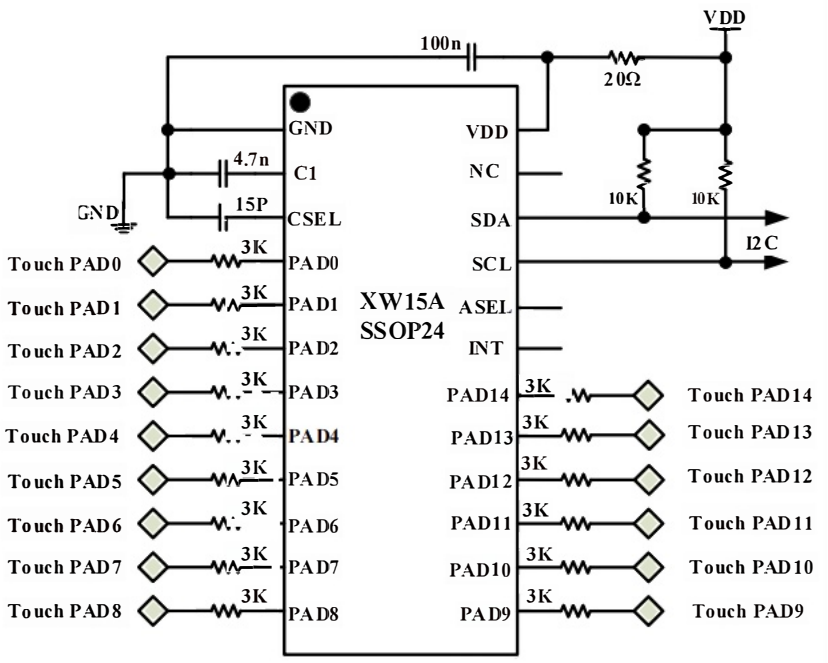
封装



管脚定义

NO.	PADNAME	Descrption		NO.	PADNAME	Descrption
1	GND	电源地		24	VDD	正电源
2	C1	内部平衡电容接口		23	NC	悬空
3	CSEL	灵敏度调节电容接口		22	SDA	I2C 数据输入输出
4	PAD0	触摸按键（不用时悬空）		21	SCL	I2C 时钟输入
5	PAD1			20	ASEL	I2C 地址选择引脚
6	PAD2			19	INT	按键有效输出引脚
7	PAD3			18	PAD14	触摸按键（不用时悬空）
8	PAD4			17	PAD13	
9	PAD5			16	PAD12	
10	PAD6			15	PAD11	
11	PAD7			14	PAD10	
12	PAD8			13	PAD9	

典型应用



1. C1 是内部平衡电容，取值范围是 1nf~10nf 。建议使用 4.7nf 。
2. CSEL 是灵敏度设置电容，电容值最小 10pf,最大 100pF。CSEL 电容的选择，可根据应用的环境，接触感应盘的大小折中选择，

绝对最大值

参数	范围	单位
VDD 电压	-0.3~5.5	V
输入输出电压	-0.3~5.5	V
工作温度范围	-40~85	℃
存储温度范围	-55~150	℃
ESD，HUM	≥8000	V

电气参数特性

(无特殊说明，Ta=25℃)

特性	符号	条件	最小值	典型值	最大值	单位
工作电压	Vcc		2.9		5.5	V
电流消耗	Idd	VCC=5.0V		2.15		mA
		VCC=3.0V		1.12		mA
		VCC=5.0V &SLEEP		42		UA
		VCC=3.0V &SLEEP		5		UA
上电稳定时间	Tini			600		ms
输出阻抗（开漏输出）	Zo	低电平		50		Ohm
		高阻		100M		
输出灌电流	Isk	VCC=5V			10.0	mA
最小检测电容	delta_CX	CSEN=15pf		0.2		pF
采样周期	Tsi	正常工作状态		15		ms
进入空闲时间	Tidle	无按键		18		s

功能描述

初始化

芯片上电复位后，需约 600mS 就可以计算出环境参数和自动校正按键走线长度，按键检测功能开始工作。

自动校正功能

芯片内置自动校正功能，芯片能够根据外部环境的变化，自动调整电容的大小，检测到按键时停止自动校正，进入按键判决过程，从检测到按键开始，经过大约 48 秒左右，芯片重新进入自动校正状态，意味着检测按键有效的时间为 48 秒左右，按键时间超过这个时间，按键无效，感应电容计入外部环境电容。

按键有效指示

芯片 INT 脚位为按键有效指示。 任意按键按下时输出为低电平，无按键按下时为高阻态，需要上拉电阻。

触摸反应时间

每个通道大约每隔 15ms 采样一次。经过按键消抖处理以后，检测到按键按下的反应时间大概是 110 毫秒，检测按键离开的反应时间大概是 70 毫秒。所以检测按键的最快频率大概是每秒 5 次。如果想提高反应速度，可以设置内部寄存器

睡眠模式

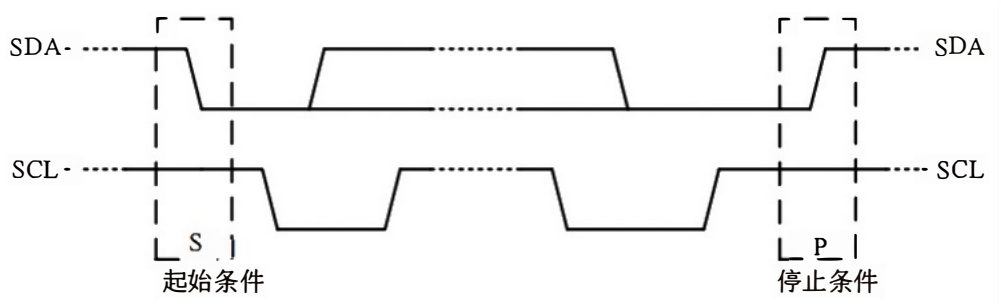
如果在一段时间内（18 秒左右）没有检测到按键并且 SDA 端口一直保持高电平，芯片会自动进入省电模式。只要让 SDA 保持高电平时间不超过 18 秒左右，芯片就不会进入睡眠模式。在睡眠模式中，按键的采样间隔会变长，电流消耗（Idd）会减小。如果检测到按键，芯片会马上离开睡眠模式，进入正常模式，

I2C 接口

XW15A 支持 I²C 总线传输协议。I²C 是一种双向、两线通讯接口，分别是串行数据线 SDA 和串行时钟线 SCL，

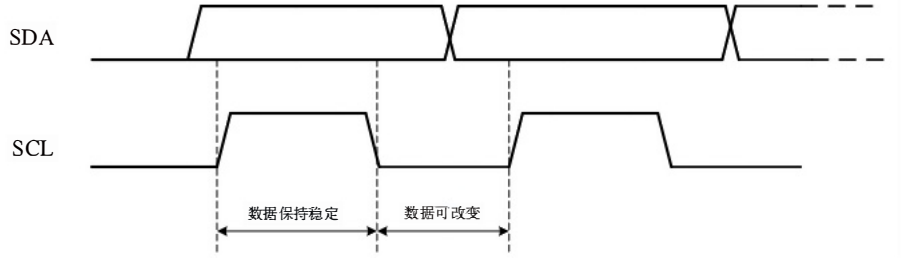
起始和停止条件

数据和时钟线都为高则称总线处在空闲状态。当 SCL 为高电平时 SDA 的下降沿（高到低）叫做起始条件（START，简写为 S）， SDA 的上升沿（低到高）则叫做停止条件（STOP，简写为 P），



位传输

每个时钟脉冲传送一位数据。SCL 为高时 SDA 必须保持稳定，因为此时 SDA 的改变被认为是控制信号。位传输参见图



字节格式

字节由 8 位数据和一个应答信号组成。

器件地址

XW15A 的地址值见下表，

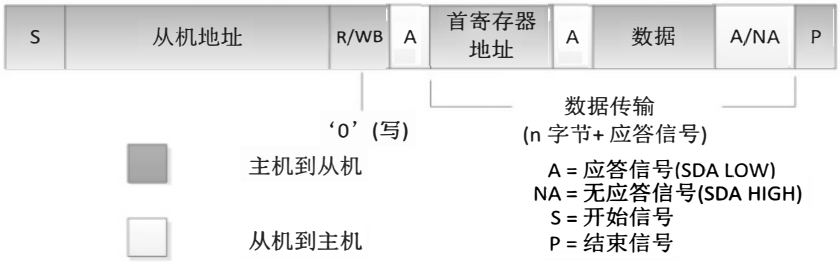
	ASEL 为高电平	ASEL 悬空	ASEL 为低电平
地址 (A[6:0])	44H	40H	42H
读命令 (A[6:0]+RWB)	89H	81H	85H
写命令 (A[6:0]+RWB)	88H	80H	84H

完整通信过程

XW15A 是从器件，支持读写两种操作模式：

（1） 写操作：

- 首字节由 7 位从机地址和一位读写位组成（RWB=0）
- 第二字节是要访问的内部寄存器地址
- 下一个字节是要写入寄存器的内容
- 继续写入下一个寄存器， 直到 接收到主机下达 STOP 信号出现
- 收到数据后 XW15A 会发送应答信号



图： 写操作

（2） 读操作：

读操作的首寄存器地址由不含数据的写操作指定， 由 STOP 信号结束。

然后主机送出开始信号， 和器件地址和读取位(R/WB=1)， 接下来的数据地址，是由首地址开始， 然后地址依次加一，

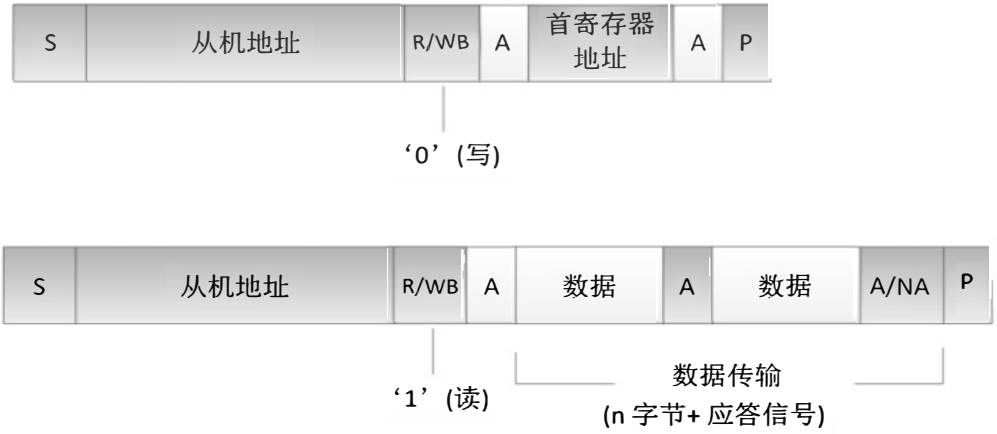


图 ： 读操作

(3) 简化的读操作

XW15A 的默认读寄存器地址为 00H。 所以如果没有写过其它寄存器， 就可以通过下面的时序直接读取按键信息，

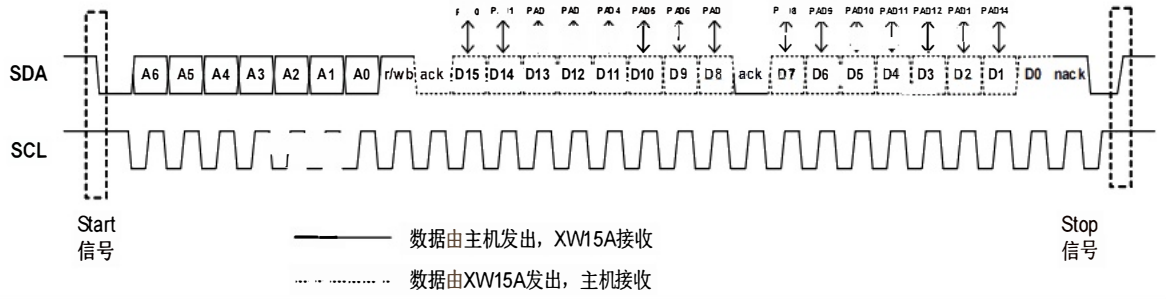


图 ： 简化读操作

寄存器列表

寄存器	地址（HEX）	读写	初始值（BIN）	寄存器功能描述							
				Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
Output1	00H	R	1111 1111	CH0	CH1	CH2	CH3	CH4	CH5	CH6	CH7
Output2	01H	R	1111 1111	CH8	CH9	CH10	CH11	CH12	CH13	CH14	
CTRL0	02H	W	011 1 1001	SLPCYC[2:0]			RTM	SENSETCOM[3:0]			
CTRL1	03H	W	1000 0000	CHEEN	CHCDEN	CHABEN	CH89EN	CH67EN	CH45EN	CH23EN	CH01EN

（1） 按键信息寄存器 Output1 (地址 00H) Output2 (地址 01H)

CH[14:0] 分别对应 PAD[14:0]的按键情况。无按键时为 1，有按键时为零，

（2）控制寄存器 0 CTRL0 (地址 02H)

SENSETCOM[3:0] 通道灵敏度设置

共有 15 档灵敏度可以设置，由低到高为：【F】【E】【D】【C】【B】【A】【9】【8】【7】【6】【5】【4】【3】【2】【1】

RTM **按键反应速度设置**

RTM	0	1（默认值）
按键有效判断	3 个采样周期	6 个采样周期
按键无效判断	1 个采样周期	4 个采样周期

SLPCYC[2:0] 睡眠时，采样周期间隔设置

SLPCYC[2:0]	0	1	2	3（默认值）	4	5	6	7
采样间隔	无穷大	1T	2T	3T	4T	5T	6T	7T

T≈87ms Vdd=3v

（3） 控制寄存器 1 CTRL1(地址 03H)

CH01EN 控制 CH0 和 CH1 通道在睡眠时，是否可以唤醒。0：不能唤醒；
1：可以唤醒

CH23EN 控制 CH2 和 CH3 通道在睡眠时，是否可以唤醒。0：不能唤醒；
1：可以唤醒

CH45EN 控制 CH4 和 CH5 通道在睡眠时，是否可以唤醒。0：不能唤醒；
1：可以唤醒

CH67EN 控制 CH6 和 CH7 通道在睡眠时，是否可以唤醒。0：不能唤醒；
1：可以唤醒

CH89EN 控制 CH8 和 CH9 通道在睡眠时，是否可以唤醒。0：不能唤醒；
1：可以唤醒

CHABEN 控制 CH10 和 CH11 通道在睡眠时，是否可以唤醒。0：不能唤醒；
1：可以唤醒

CHCDEN 控制 CH12 和 CH13 通道在睡眠时，是否可以唤醒。0：不能唤醒；
1：可以唤醒

CHEEN 控制 CH14 通道在睡眠时，是否可以唤醒。0：不能唤醒；
1：可以唤醒

外围电路和注意事项

XW15A 的外围电路很简单，只需少量电容电阻元件， 1.5 是 XW15A 的典型应用电路，

内部平衡电容和灵敏度调节电容

C1 电容和 CSEL 电容建议采用精度 10%的 NPO 材质电容，在 PCB 板 layout 时，请将 C1 电容和 CESL 电容尽量贴近 IC 放置。

灵敏度电容和按键检测 PAD 大小以及介质材料与厚度选择

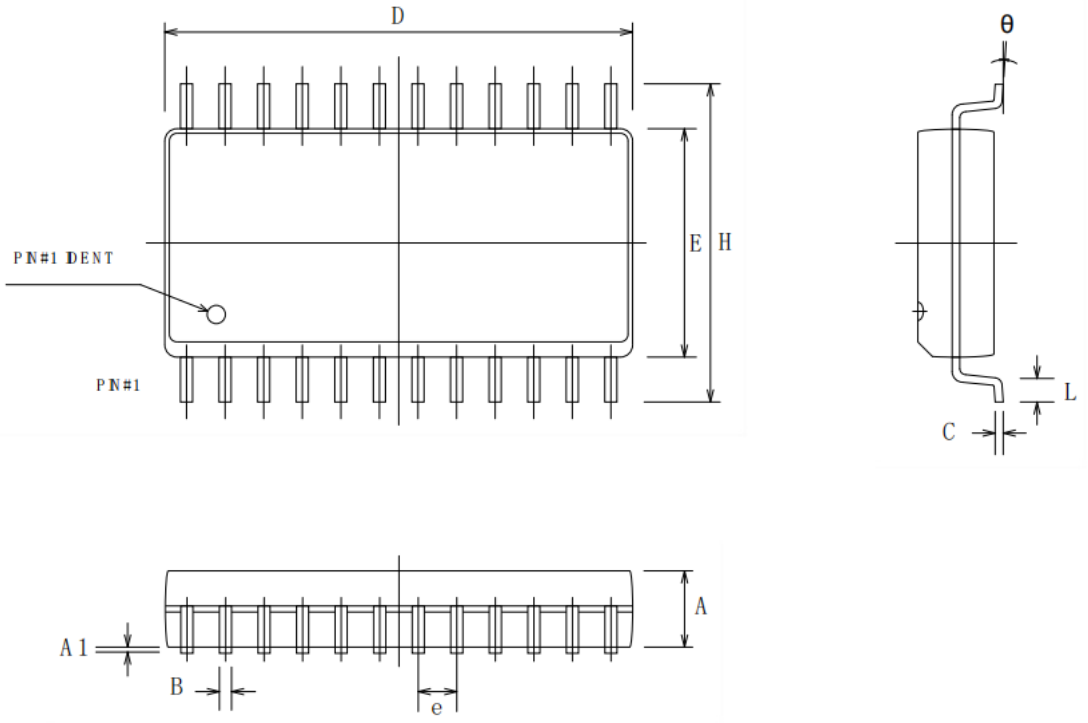
常用的介质有 玻璃、亚克力、塑料、陶瓷等，用户可以根据自己的实际使用情况选择合适的材料及厚度，按照材料的不同和 PCB 板的布局来决定按键 PAD 的大小和电容 CSEL 的值。隔离介质越厚，要求使用的 CSEL 电容越小（增大检测的灵敏度），同时要求适当加大按键检测 PAD 的面积。反之，隔离介质越薄，适当增大 CSEL 电容，增加系统的抗干扰能力，一般建议在 10pf 和 100pF 之间由小到大地选择合适的电容。

一般情况下，按键检测 PAD 面积可以在 3mm*3mm~30mm*30mm 之间，每个感应盘的面积保持相同,以确保灵敏度相同。电容传感器可以是任何形状的导体，建议使用直径大于 10mm 的圆形金属片或边长 10mm 的正方形金属片。常用的感应盘有 PCB 板上的铜箔、平顶圆柱弹簧、金属片和导电橡胶等。

VDD 电源电压注意事项

XW15A 测量的是电容的微小变化，要求电源的纹波和噪声要小，要注意避免由电源串入的外界强干扰。尤其是应用于高噪声环境时，必须能有效隔离外部干扰及电压突变，要求电源有较高稳定度，应尽量远离高压大电流的器件区域或者加屏蔽。如果电源文波幅度较大时，建议对电源做特别处理，比如增加滤波或采用 78L05 组成的稳压线路。在某些特定的应用场合，要尽可能的让触摸电路远离某些功能电路，比如收音机，RF 等。

封装尺寸图 (SSOP-24) （0.635）



Symbol	Dimensions In Millimeters		Dimensions In Inches	
	Min	Max	Min	Max
A	1.25	1.55	0.049	0.061
A1	0.05	0.25	0.002	0.010
B	0.194	0.314	0.008	0.012
C	0.15	0.25	0.006	0.010
D	8.55	8.75	0.337	0.344
E	3.80	4.00	0.015	0.157
e	0.635		0.025	
H	5.70	6.30	0.224	0.248
L	0.30	0.90	0.012	0.035
θ	0°	7°	0°	7°

附录：通过 I2C 接口读写 XW15A 的 C 语言演示程序

1) I2C 写控制寄存器函数

void i2c_write(unsigned dev_addr, unsigned reg_addr, unsigned char * src_buf, unsigned char len)

dev_addr 设置器件地址

reg_addr 设置寄存器地址

scr_buf 写入数据内容的首字节地址

len 写入数据内容的长度

举例：将数据 0x93 写入 XW15A 的 CTRL0 寄存器(地址为 0x03) XW15A 的 ASEL 端口悬空

```
char buffer;
buffer=0x93;
i2c_write(0x40, 3,  &buffer, 1);
```

2) I2C 读取按键信息寄存器函数

void i2c_read_direct(unsigned dev_addr,unsigned char * dest_buf,unsigned char len)

dev_addr 设置器件地址

reg_addr 设置寄存器地址

dest_buf 读出数据的首字节地址

Len 读取数据的长度

举例：读取 XW15A 按键信息 OUTPUT1 OUTPUT2 寄存器(地址为 0x00 0x01) XW15A 的 ASEL 端口悬空

```
char buffer[2]={0,0};
i2c_write(0x40, 0, &buffer, 0);          //先设置读取寄存器地址为 0x00。如果上电后没有写过其它寄存器，这个步骤可以省略。因为芯片默认的读取地址就是 0x00
i2c_read_direct(0x40, &buffer, 2); //读取 0x00 的内容放入 buffer[0]中,0x01 的内容放入 buffer[1]中。有按键时 buffer[2]相应位是 0，无按键时 buffer[2]相应位是 1

//*****************************************************************************
void i2c_delay()//简单延时函数
{
    char i;
    for(i=0;i<2;i++); }
```

```
//*****************************************************************************
void i2c_start()  //开始信号 SCL 在高电平期间，SDA 一个下降沿则表示启动信号
{
    SDA=1; //释放 SDA 总线
    i2c_delay();
    SCL=1;
    i2c_delay();
    SDA=0;
    i2c_delay();
    SCL=0;
    i2c_delay();
//  SDA=1;
}
//*****************************************************************************
void i2c_stop()   //停止 SCL 在高电平期间，SDA 一个上升沿则表示停止信号
{
//  SCL=0;
//    SDA=0;
//    i2c_delay();
    SCL=1;
//    i2c_delay();
//    SDA=1;
//    i2c_delay();
}
//*****************************************************************************
void i2c_checkack()  //应答 SCL 在高电平期间，SDA 被从设备拉为低电平表示应答
{
    bit  bit_temp;
    SCL=1;
    bit_temp=SDA;
/*  if(bit_temp)
        //无 ACK 回应
        ERR=0;
    else
        ERR=1;   */
    SCL=0;
//    i2c_delay();
}
void i2c_sendack()  //应答 SCL 在高电平期间，SDA 被从设备拉为低电平表示应答
{
```

```
    SDA=0;
    i2c_delay();
    SCL=1;
    i2c_delay();
    SCL=0;
    SDA=1;

}
void i2c_sendnack()   //应答 SCL 在高电平期间，SDA 被从设备拉为低电平表示应答
{

    SDA=1;
    i2c_delay();
    SCL=1;
    i2c_delay();
    SCL=0;

}
//*****************************************************************************
//*****************************************************************************
void i2c_write_byte(unsigned char date) //写一个字节
{
    unsigned char i,temp;
    temp=date;


    for(i=0;i<8;i++)
    {
        temp=temp<<1;
//          i2c_delay();
        SDA=CY;
        i2c_delay();
        SCL=1;//拉高 SCL，此时 SDA 上的数据稳定
        i2c_delay();
        SCL=0;//拉低 SCL，因为只有在时钟信号为低电平期间按数据线上的高低电平状态才允许变化；并在此时和上一个循环的 scl=1 一起形成一个上升沿
    }
//    SCL=0;//拉低 SCL，为下次数据传输做好准备
//    i2c_delay();
    SDA=1;//释放 SDA 总线，接下来由从设备控制，比如从设备接收完数据后，在 SCL 为高时，拉低 SDA 作为应答信号
```

```
    i2c_delay();
}
//*****************************************************************************
unsigned char i2c_read_byte()//读一个字节
{
    unsigned char i,k;

    for(i=0;i<8;i++)
    {
        i2c_delay();
        SCL=1;//上升沿时，IIC 设备将数据放在 sda 线上，并在高电平期间数据已经稳定，可以接收啦
 //         i2c_delay();
        k=(k<<1)|SDA;
        SCL=0;
    }
    return k;
}
//*****************************************************************************
void i2c_write(unsigned dev_addr, unsigned reg_addr, unsigned char * src_buf, unsigned char len)
{
    char i;
    i2c_start();
    i2c_write_byte((dev_addr<<1));//发送发送从设备地址 写操作
    i2c_checkack();//等待从设备的响应
    i2c_write_byte(reg_addr);//发送发送寄存器地址 写操作
    i2c_checkack();//等待从设备的响应
    for(i=0;i<len;i++)
    {
        i2c_write_byte(src_buf[i]);//发送缓冲区数据 写操作
        i2c_checkack();//等待从设备的响应
    }
    i2c_stop();//停止
}
void i2c_read_direct(unsigned dev_addr,unsigned char * dest_buf,unsigned char len)
{
    char i;
    i2c_start();//启动
    i2c_write_byte((dev_addr<<1)+1);//发送发送从设备地址 读操作
    i2c_checkack();//等待从设备的响应
```

```
    dest_buf[0]=i2c_read_byte();//获取数据

    for(i=1;i<len;i++)
    {
        i2c_sendack();
        dest_buf[i]=i2c_read_byte();
    }
    i2c_sendnack();
    i2c_stop();//停止
}
```