目录

[Pwm输出 2](#_Toc27735)

[LCD1602显示 10](#_Toc10862)

[（1）清楚液晶显示器，即将DDRAM的内容全部填入空白的ASCII码20H 13](#_Toc13780)

[（2）地址计数器（AC）的值设为0 13](#_Toc23314)

[（3）光标复位到地址00H位置（显示器的左上方） 13](#_Toc3006)

[（4）光标返回地址00H 14](#_Toc29815)

[（5）AC值设为0 14](#_Toc30276)

[（6）DDRAM的内容不变 14](#_Toc30820)

[（7）读取忙信号BF的内容 15](#_Toc26908)

[（8）读取地址计数器（AC）的内容 15](#_Toc18587)

[AT24C002 19](#_Toc21700)

[1、AT24C02简介 19](#_Toc2360)

[2、AT24C02特性 19](#_Toc10452)

[3、 硬件原理图 19](#_Toc3399)

[4、 管脚描述 20](#_Toc26509)

[5、读写操作时序 21](#_Toc3470)

6、[实验如下：储存数据并且在led灯显示 24](#_Toc23043)

[代码： 24](#_Toc25195)

[蓝牙模块 29](#_Toc17487)

[1.蓝牙模块HC05： 29](#_Toc8189)

[2. 蓝牙模块的调试 29](#_Toc28887)

[2.1 两种工作模式： 29](#_Toc23245)

[2.2 进入命令响应工作模式？ 30](#_Toc11985)

[2.3 什么叫做置高一次PIO11？ 30](#_Toc18112)

[2.4 怎么区分进了命令响应工作模式呢？ 30](#_Toc9712)

[2.5 串口调试助手发送AT命令格式 30](#_Toc10907)

[2.6 AT命令 30](#_Toc12374)

[oled显示 35](#_Toc13471)

[1、iic通信 35](#_Toc13636)

[2、SPI通信 35](#_Toc24838)

[3.主要介绍iic协议通信的应用方式： 36](#_Toc4640)

[代码详解 38](#_Toc26982)

[红外遥控 44](#_Toc24883)

[1.什么是红外遥控 44](#_Toc13975)

[2.红外信号是怎么传输的 44](#_Toc20022)

[3.NEC协议 45](#_Toc9254)

[4.程序分析 47](#_Toc24411)

Pwm输出

1. 什么叫呼吸灯？  
   由亮到暗逐渐变化，很有节奏感地一起一伏，感觉好像人在呼吸，当手机收到消息，屏幕上的指示灯会渐变，比较显眼，能起到一个通知提醒的作用，其实这就是一个呼吸灯。

2、什么是PWM?

PWM:Pulse Width Modulation，脉冲宽度调制，是利用微处理器的数字输出来对模拟电路进行控制的一种非常有效的技术。

可以将PWM理解为：对脉冲信号的处理技术，这里的“处理”指就的是改变占空比，从而改变最后呈现的效果。

既然是针对的是脉冲信号，肯定会有以下这些概念（关于脉冲就是不断跳跃的波，而PWM脉冲信号是矩形波）：

PWM的频率：是指1秒钟内，信号从高电平到低电平再回到高电平的次数，也就是说一秒钟PWM有多少个周期。

单位：Hz

一般人眼睛对于80Hz以上刷新频率则完全没有闪烁感（因人而异）。

频率太小的话看起来就会闪烁，那么我们平时见到的LED灯，当它的频率大于50Hz的时候，人眼就会产生视觉暂留效果，基本就看不到闪烁了，而是一个常亮的LED灯。

频率很高时，看不到闪烁，占空比越大，LED越亮（平均电压越大）；频率很低时，可看到闪烁，占空比越大，LED越亮。

所以，在频率一定下，可以用不同占空比改变LED灯的亮度，使其达到一个呼吸灯的效果。

PWM的周期：T=1/f

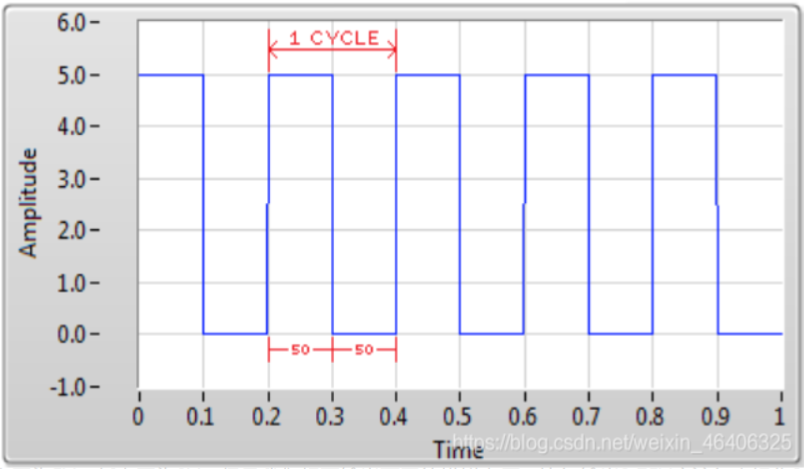
如果频率为50Hz，也就是说一个周期是20ms，那么一秒钟就有50次PWM周期。

占空比：在一个周期内，高电平的时间占整个周期时间

duty=t1/T

单位：% (0%-100%)

脉宽时间：高电平时间。

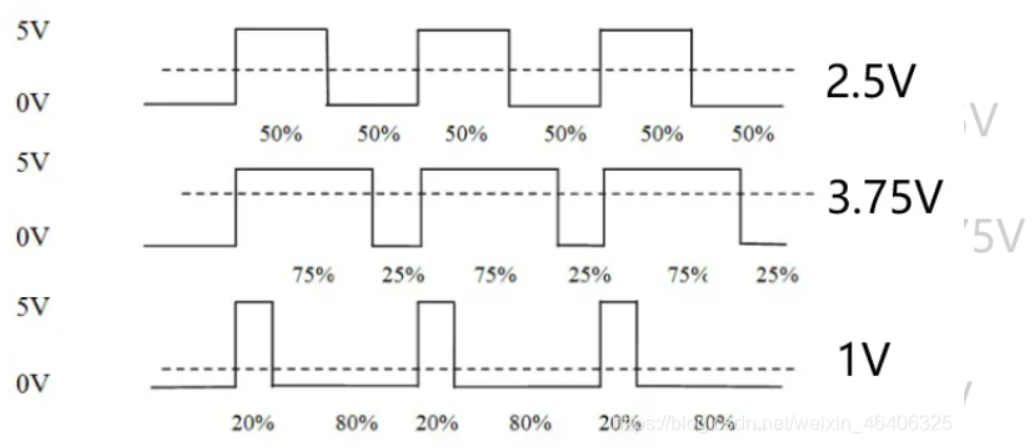


PWM原理：

单片机的IO引脚输出的是数字信号，且只能输出1和0。

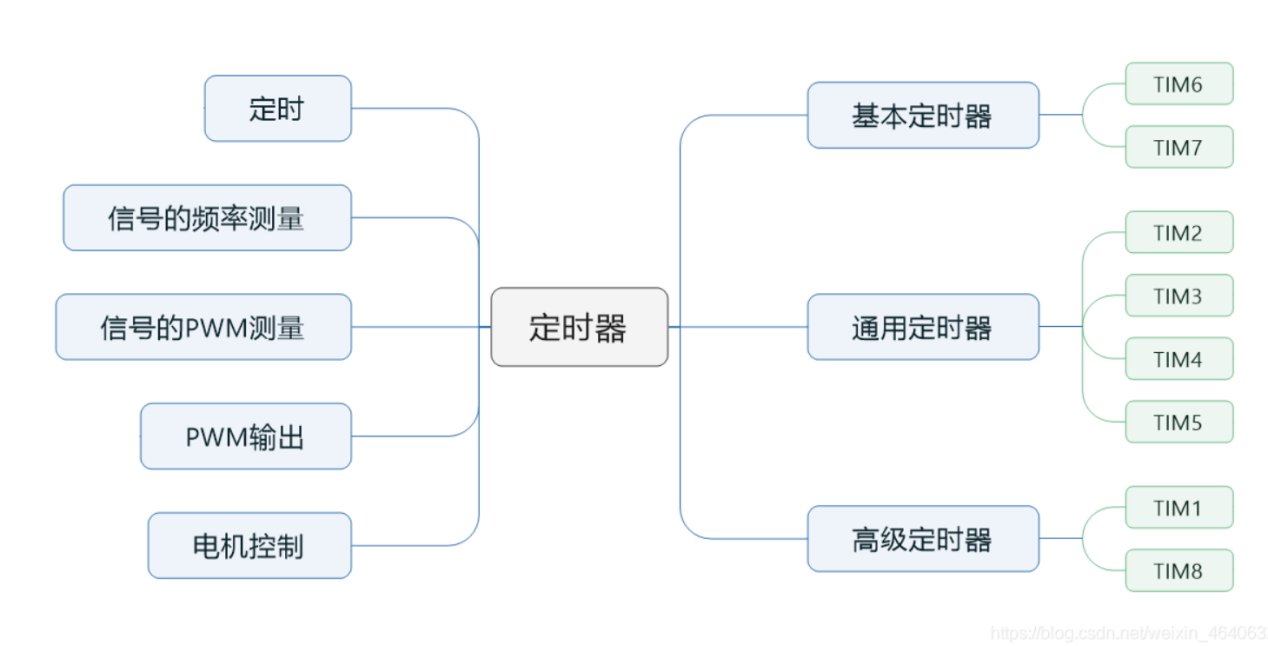
TTL电平中，高电平为5V，低电平为0V，但是我们想要输出不同的模拟电压，比如输出3.75V应该怎么操作？

这里就要用到PWM，通过改变IO引脚输出方波的占空比，从而得到不同的模拟电压值，理论上来讲，可以输出任意不大于最大电压值（即0~5V之间任意大小）的模拟电压。



模拟电压=最大电压\*占空比，这里的模拟电压是平均值，占空比越大，则模拟电压也越大。

如何产生PWM波？  
使用单片机中的定时器I/O口可以产生PWM波。

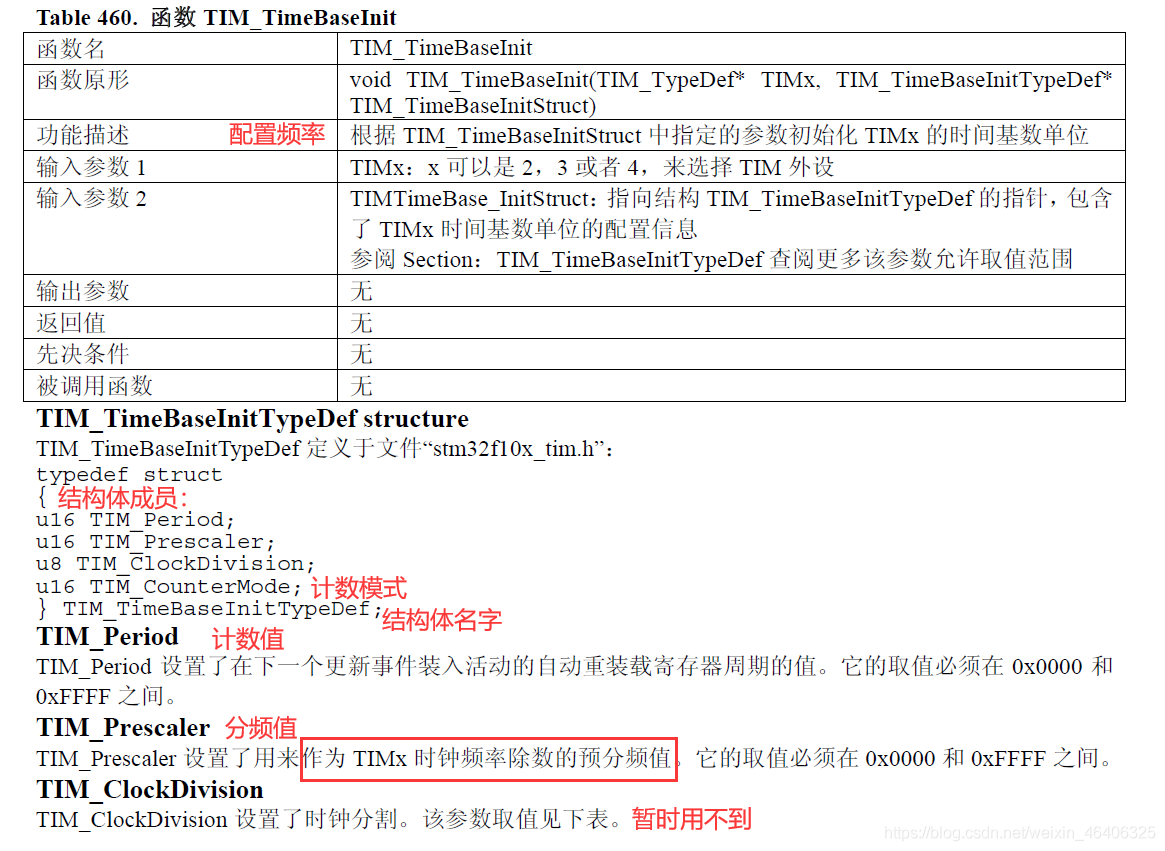


基本定时器只具有定时功能；而高级定时器比通用定时器多了三相六步电机的接口以及刹车功能，而这里我们只需要输出PWM波，使用通用定时器就够了。

如何使用定时器输出PWM波？换个问法就是：如何配置定时器输出PWM波？

通用定时器可以利用GPIO引脚进行脉冲输出，但是并不是任意GPIO 都具有STM32 定时器的输出通道功能，这需要根据芯片的引脚功能选择具有定时器输出通道功能的引脚来控制RGB 灯。

与定时器输出PWM有关的库函数：



实验：通过PWM波实现单色呼吸灯（LED从亮渐灭，再从灭渐亮，以此循环）

讲解代码：

#include "reg52.h"

typedef unsigned char u8;

u8 Tcount,kongbi,fangxiang; //定义pwm周期，占空比，方向参数

sbit led = P2^1;

void Timer0Init();

void Timer0();

void breatheled();

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* 函 数 名 : main

\* 函数功能 : 主函数

\* 输 入 : 无

\* 输 出 : 无

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void main()

{

Timer0Init(); //定时器初始化

while(1)

{

breatheled(); //执行pwm呼吸灯

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* 函 数 名 : Timer0Init

\* 函数功能 : 定时器初始化

\* 输 入 : 无

\* 输 出 : 无

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void Timer0Init(void) //200微秒@11.0592MHz

{

TMOD &= 0xF0; //设置定时器模式

TMOD |= 0x02; //设置定时器模式

TL0 = 0x48; //设置定时初值

TH0 = 0x48; //设置定时重载值

TF0 = 0; //清除TF0标志

ET0 = 1; //打开定时器0中断允许

EA = 1; //打开总中断

TR0 = 1; //定时器0开始计时

}

void Timer0() interrupt 1

{

TL0 = 0x48; //设置定时初值

TH0 = 0x48; //设置定时重载值

Tcount++;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* 函 数 名 : breatheled

\* 函数功能 : pwm呼吸灯

\* 输 入 : 无

\* 输 出 : 无

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void breatheled()

{

if(Tcount>=100) //设置pwm周期

{

Tcount=0; //溢出时归零

if(fangxiang==1) //当方向为1，灯灭，占空比减少

{

kongbi--;

}

else if(fangxiang==0) //当方向为0，灯亮，占空比增加

{

kongbi++;

}

}

if(kongbi<=0) //当占空比小于等于0，方向逆转

{

fangxiang=0;

}

else if(fangxiang>=100) //当占空比大于等于100，方向逆转

{

fangxiang=1;

}

if(Tcount<kongbi)

{

led=0;

}

else

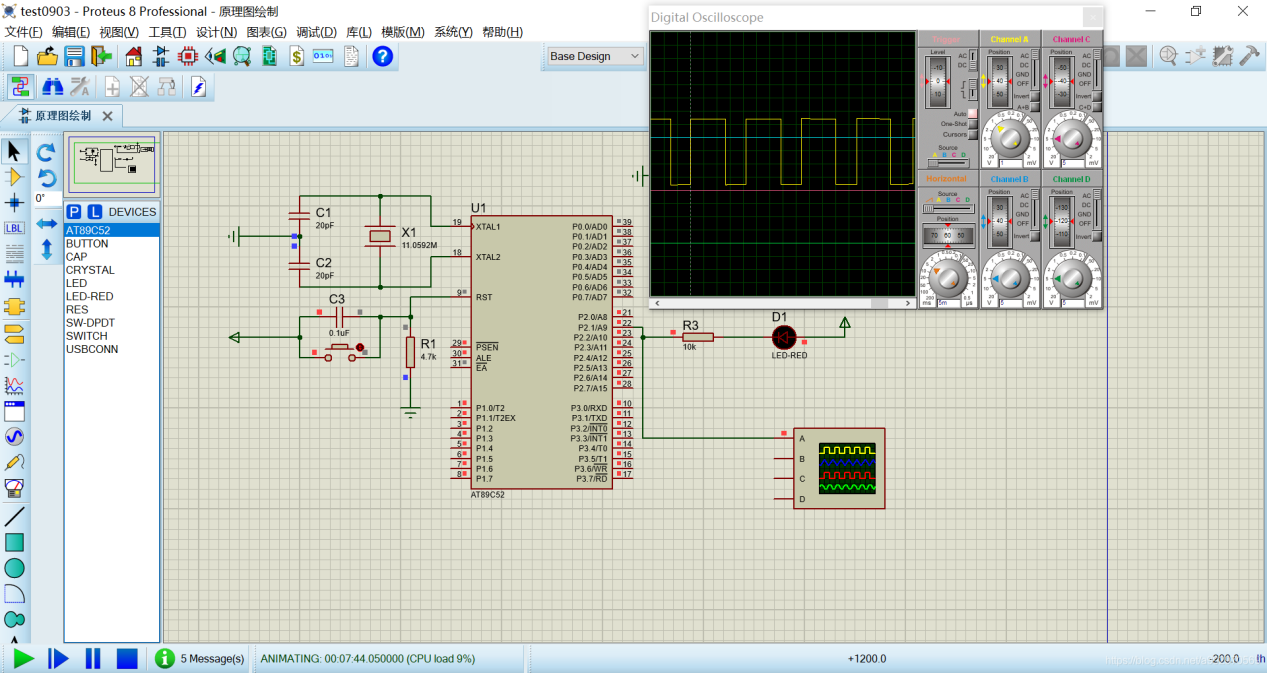
{

led=1;

}

}

Protuse仿真如下：



LCD1602显示

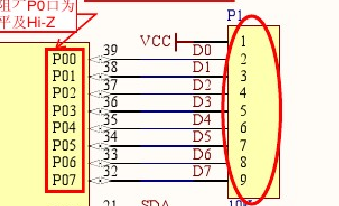
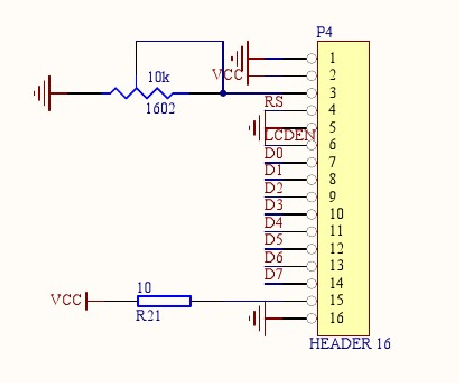
1.1602液晶简介：

1602液晶也叫1602字符型液晶，16x02，每行16个字符显示两行。它是一种专门用来显示字母、数字、符号等的点阵型液晶模块。它由若干个5X7或者5X11等点阵字符位组成，每个点阵字符位都可以显示一个字符，每位之间有一个点距的间隔，每行之间也有间隔，起到了字符间距和行间距的作用，正因为如此所以它不能很好地显示图形（用自定义CGRAM，显示效果也不好）。

引脚接口：



1602液晶上有16个引脚，上面依次标了上述符号，与单片机上的接口依次对应连接即可。



2.指令

指令1：清屏指令。

（1）清楚液晶显示器，即将DDRAM的内容全部填入空白的ASCII码20H

（2）地址计数器（AC）的值设为0

（3）光标复位到地址00H位置（显示器的左上方）

指令2：光标复位指令。

（1）光标返回地址00H

（2）AC值设为0

（3）DDRAM的内容不变

指令3：光标和显示模式设置。设定每次输入1位数据后光标的移位方向，并且设定输入后字符是否移动。

I/D（光标移动方向）： 高：右移 低：左移

S（屏幕上所有文字是否移动）：高 ：有效右移 低：无效

指令4：显示开关控制。控制显示器开/关、光标显示/关闭以及光标是否闪烁

D（控制整体显示的开与关）：高：开显示 低：关显示

C（控制光标的开与关）： 高：有光标 低：无光标

B（控制光标是否闪烁）： 高：闪烁 低：不闪

指令5：光标或显示移位。

指令6：功能设置命令。设定数据总线位数、显示的行数及字型。

DL（数据总线位数）：高：8 低：4

N（显示行数） ：高：2 低：1

F（显示点阵类型） ：高：5\*7点阵 低：5\*10点阵

指令7：设定CGRAM地址指令。

设定下一个要存入数据的CGRAM的地址。

指令8：设定DDRAM地址指令。

定下一个要存入数据的DDRAM的地址。

指令9：读取忙信号或AC地址指令。

（1）读取忙信号BF的内容

BF（液晶显示器是否忙）：高：忙，暂时无法接收单片机送来的数据或指令 低：可接收

（2）读取地址计数器（AC）的内容

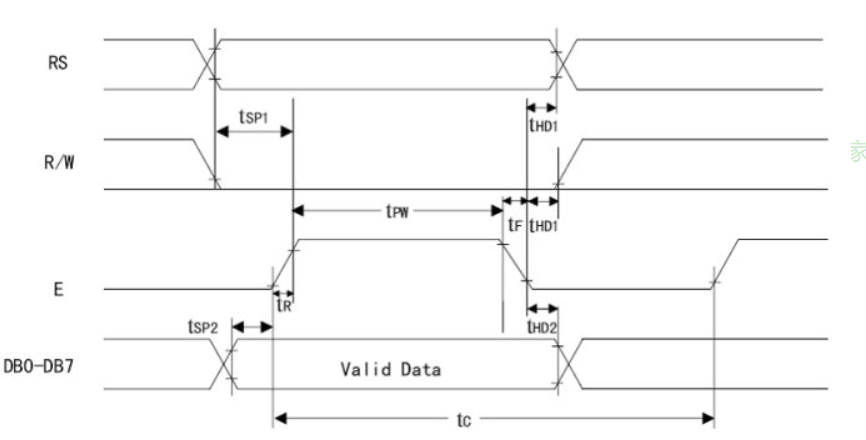
指令10：数据写入DDRAM或CGRAM指令。

（1）将字符码写入DDRAM，以使液晶显示屏显示出相对于的字符

（2）使用者自己设计的图形存入CGRAM

指令11：从CGRAM或DDRAM读出数据的指令。读取DDRAM或CGRAM中的内容。

# ****时序图****



代码如下：

main.c

#include "reg52.h"

#include "LCD1602.h"

typedef unsigned int u16; //对数据类型进行声明定义

typedef unsigned char u8;

u8 xianshi[]=" for freedom ";

void main()

{

u8 i;

LcdInit(); //初始化

for(i=0;i<13;i++)

{

LCDWriteData(xianshi[i]);

}

while(1);

}

Lcd1602：

#include "reg52.h"

sbit LCD\_RS=P2^6;

sbit LCD\_RW=P2^5;

sbit LCD\_E=P2^7;

#define LCDData P0

#define uint unsigned int

#define uchar unsigned char

void Lcd1602\_Delay1ms(uint c)

{

uchar a,b;

for(;c>0;c--)

{

for(b=199;b>0;b--)

{

for(a=1;a>0;a--);

}

}

}

void LcdWriteCom(uchar com)

{

LCD\_E=0;

LCD\_RS=0;

LCD\_RW=0;

LCDData=com;

Lcd1602\_Delay1ms(1);

LCD\_E=1;

Lcd1602\_Delay1ms(5);

LCD\_E=0;

}

void LCDWriteData(uchar dat)

{

LCD\_E=0;

LCD\_RS=1;

LCD\_RW=0;

LCDData=dat;

Lcd1602\_Delay1ms(1);

LCD\_E=1;

Lcd1602\_Delay1ms(5);

LCD\_E=0;

}

void LcdInit() //初始化

{

LcdWriteCom(0x38); //开显示

LcdWriteCom(0x0c); //开显示不显示光标

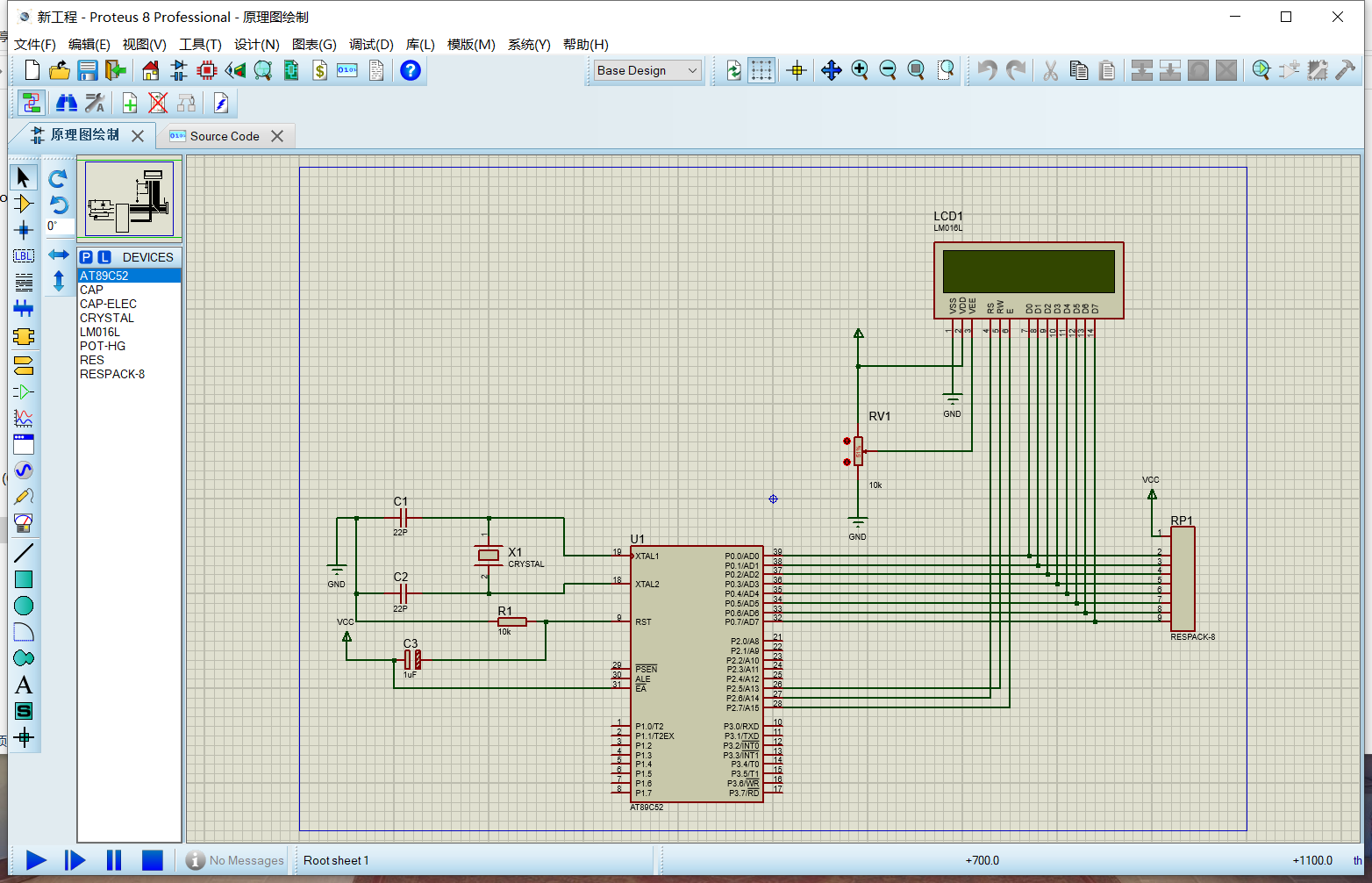
LcdWriteCom(0x06); //写一个指针加1

LcdWriteCom(0x01); //清屏

LcdWriteCom(0x80); //设置数据指针起点

}

Protues仿真：



AT24C002

1、AT24C02简介

  AT24C01/02/04/08/16是一个1K/2K/4K/8K/16K位(AT24C02大小为256字节)串行CMOS E2PROM内部含有128/256/512/1024/2048个8位字节，CATALYST公司的先进CMOS技术实质上减少了器件的功耗AT24C01有一个8字节页写缓冲器 AT24C02/04/08/16有一个16字节页写缓冲器该器件通过I2C总线接口进行操作有一个专门的写保护功能。

2、AT24C02特性

  ①、与 400KHz I2C 总线兼容

  ②、1.8 到 6.0 伏工作电压范围

  ③、低功耗 CMOS 技术

  ④、写保护功能 当 WP 为高电平时进入写保护状态

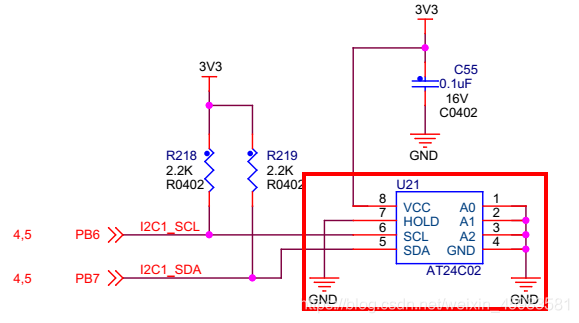
  ⑥、页写缓冲器

  ⑦、自定时擦写周期

  ⑧、1,000,000编程/擦除周期

  ⑨、可保存数据100年

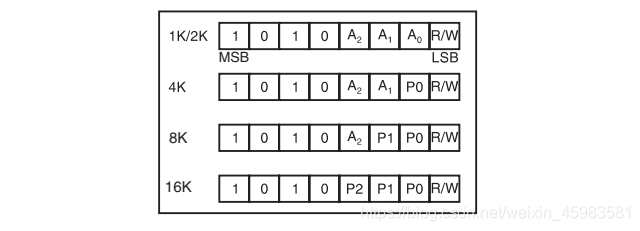
1. **硬件原理图**



1. **管脚描述**



A0、A1、A2、三个脚不同的电平可以形成不同的地址，最多8种。原理图中A0、A1、A2、三个脚全部接地，也就是硬件决定不可更改，默认为A0=0、A1=0、A2=0。同时写保护脚WP(原理图中为HOLD)，也是接地。因此默认也是关闭写保护的。

1. A0、A1和A2对应器件的管脚1、2和3

2. P0、P1、P2对应存储阵列地址字地址。

3. R/W为读写方向位，1为读，0为写。

  对应AT24C02硬件原理图而言，A0、A1、A2、三个脚被硬件接地，即A0=0、A1=0、A2=0。所以AT24C02读地址为1010 0001(0xA1),AT24C02写地址为1010 0000(0xA0)。

  这里再简单的介绍一下P0、P1、P2的含义，对于AT24C02而言大小为256字节，而对于AT24C08而言大小为1024(4\*256)字节。假如P0=0、P1=0,即代表选中第一个256字节的首地址，P0=0、P1=0,即代表选中第一个256的首地址，P0=1、P1=0,即代表选中第二个256的首地址，以此类推。

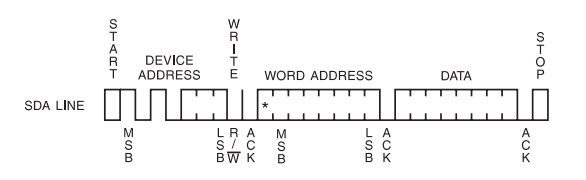
IMG_256

、写周期限制在这里插入图片描述

  写周期时间是指从一个写时序的有效停止信号到内部编程/擦除周期结束的这一段时间。在写周期期间，总线接口电路禁能，SDA保持为高电平，器件不响应外部操作。

  简单的说就是，当向AT24C02执行写操作时，从发出写命令到写完成最多10ms。因此在写程序时，每次写操作后都延迟10ms以保证稳定。

7、读写操作时序



/\*

函数功能：向AT24c02指定地址写一字节数据

\*/

void At24c02\_Write\_OneByte(u8 addr,u8 data)

{

IIC\_Start();

IIC\_Write\_Byte(At24c02\_Write\_Addr);//发送写地址0XA0

if(IIC\_Check\_Ack())return ;

IIC\_Write\_Byte(addr);

if(IIC\_Check\_Ack())return ;

IIC\_Write\_Byte(data);

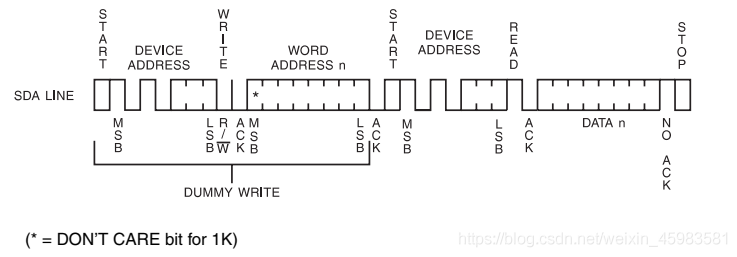
if(IIC\_Check\_Ack())return ;

IIC\_Stop();

delay\_ms(10);//确保写完成

}

**7.2、向指定地址开始读数据**



/\*

函数功能：向AT24c02指定地址开始读数据

\*/

void At24c02\_Read\_Data(u8 addr,u8 \*data,u8 len)

{

u8 i=0;

IIC\_Start();

IIC\_Write\_Byte(At24c02\_Write\_Addr);//发送写地址0XA0

if(IIC\_Check\_Ack())return ;

IIC\_Write\_Byte(addr);//写数据的地址

if(IIC\_Check\_Ack())return ;

IIC\_Start();

IIC\_Write\_Byte(At24c02\_Read\_Addr);//发送读地址0XA1

if(IIC\_Check\_Ack())return ;

for(i=0;i<len;i++)

{

data[i]=IIC\_Read\_Byte();

if(i==len-1)IIC\_Send\_Ack(1);

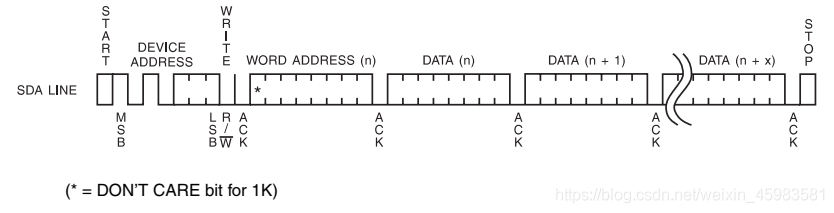
else IIC\_Send\_Ack(0);

}

IIC\_Stop();

}

**7.3、页写**



用页写 AT24C01可一次写入8个字节数据 AT24C02/04/08/16可以一次写入16个字节的数据，页写操作的启动和字节写一样，不同在于传送了一字节数据后并不产生停止信号，主器件被允许发送停止信号P,AT24C01 P=7,AT24C02/04/08/16 P=15个额外的字节,每发送一个字节数据后AT24C01/02/04/08/16 产生一个应答位并将字节地址低位加1,高位保持不变。如果在发送停止信号之前主器件发送超过P+1个字节地址计数器将自动翻转，先前写入的数据被覆盖。

  简单的说，AT24C01一页为8字节，AT24C01/02/04/08/16为16字节。对于AT24C02而言，在一页内，每写一个字节写指针+1，当大于16字节时，又回到第一个字节的位置，先前写入的数据将会被覆盖。

/\*

函数功能：对AT24C02指定地址页内写数据

\*/

void At24c02\_Write\_Page(u8 addr,u8 \*data,u8 len)

{

u8 i=0;

IIC\_Start();

IIC\_Write\_Byte(At24c02\_Write\_Addr);//发送写地址0XA0

if(IIC\_Check\_Ack())return ;

IIC\_Write\_Byte(addr);//写数据的地址

if(IIC\_Check\_Ack())return ;

for(i=0;i<len;i++)

{

IIC\_Write\_Byte(\*data++);/

if(IIC\_Check\_Ack())return ;

}

IIC\_Stop();

delay\_ms(10);

}

**7.4、向指定地址跨页写数据**

**/\***

**函数功能：对AT24C02指定地址跨页写数据**

**\*/**

**void At24c02\_Write\_Data(u8 addr,u8 \*data,u8 len)**

**{**

**u8 write\_len=16-addr%16;//起始页剩下的空间**

**if(write\_len>len)write\_len=len;**

**while(1)**

**{**

**At24c02\_Write\_Page(addr,data,write\_len);**

**if(write\_len==len)break;**

**addr+=write\_len;**

**data+=write\_len;**

**len-=write\_len;**

**if(len>16)write\_len=16;**

**else write\_len=len;**

**}**

**}**

**实验如下：储存数据并且在led灯显示**

**代码：**

**#include "reg51.h" //单片机头文件**

**#include "intrins.h"**

**typedef unsigned int u16; //对数据类型进行声明定义**

**typedef unsigned char u8;**

**#define OP\_READ 0xa1**

**#define OP\_WRITE 0xa0**

**sbit SDA=P3^4;**

**sbit SCL=P3^3;**

**//函数声明**

**//延时函数**

**void delay1ms()**

**{**

**u8 i,j;**

**for(i=0;i<10;i++)**

**for(j=0;j<33;j++);**

**}**

**void delaynms(unsigned char n)**

**{**

**unsigned char i;**

**for(i=0;i<n;i++)**

**{**

**delay1ms();**

**}**

**}**

**//数据传输启动函数**

**void start()**

**{**

**SDA=1; //SDA初始化为高电平“1”**

**SCL=1; //开始数据传输时，要求SCL为高电平**

**\_nop\_();**

**\_nop\_();**

**\_nop\_();**

**\_nop\_();**

**SDA=0;//开始信号**

**\_nop\_();**

**\_nop\_();**

**\_nop\_();**

**\_nop\_();**

**SCL=0; //SCL为低电平时，SDA上数据才允许变化**

**}**

**//数据传送关闭函数**

**void stop()**

**{**

**SDA=0;**

**SCL=1;**

**\_nop\_();**

**\_nop\_();**

**\_nop\_();**

**\_nop\_();**

**SDA=1;**

**\_nop\_();**

**\_nop\_();**

**\_nop\_();**

**\_nop\_();**

**SDA=1;**

**SCL=0;**

**}**

**//AT24C02读取函数**

**unsigned char ReadData()**

**{**

**u8 i,x;**

**for(i=0;i<8;i++)**

**{**

**SCL=1; //SCl置为高电平**

**x<<=1; //将X中的各个二进位向左移位**

**x|=(unsigned char)SDA; //将SDA上的数据通过按位“或”运算存入X中**

**SCL=0;**

**}**

**return(x);**

**}**

**//AT24C02写入数据函数**

**bit WriteCurrent(unsigned char y)**

**{**

**u8 i;**

**bit ack\_bit;**

**for(i=0;i<8;i++)**

**{**

**SDA = (bit)(y&0x80); //通过按位“与”运算将最高位数据送到S,因为传送时高位在前，低位在后**

**\_nop\_();**

**SCL=1;**

**\_nop\_();**

**\_nop\_();**

**SCL=0;**

**y<<=1;**

**}**

**SDA=1;**

**\_nop\_();**

**\_nop\_();**

**SCL=1;**

**\_nop\_();**

**\_nop\_();**

**\_nop\_();**

**\_nop\_();**

**ack\_bit=SDA; //接受设备（AT24Cxx)向SDA送低电平**

**//表示已经接收到一个字节,若送高电平，表示没有接收到，传送异常**

**SCL=0;**

**return ack\_bit; //返回AT24C00应答位**

**}**

**//指定地址写入数据**

**void WriteSet(u8 add,u8 dat)**

**{**

**start(); //开始数据传递**

**WriteCurrent(OP\_WRITE); //选择要操作的AT24C02芯片**

**WriteCurrent(add); //写入地址**

**WriteCurrent(dat); //向当前地址写入数据**

**stop(); //停止数据传递**

**delaynms(4); //一个字节写入周期为1ms，延时**

**}**

**//当前位置读取数据**

**unsigned char ReadCurrent()**

**{**

**u8 x;**

**start(); //开始数据传递**

**WriteCurrent(OP\_READ);**

**x=ReadData(); //将读取的数存入X**

**stop();**

**return x;**

**}**

**//指定地址读取数据**

**unsigned char ReadSet(unsigned char set\_addr)**

**{**

**start();**

**WriteCurrent(OP\_WRITE); //选择要操作的ATC02芯片**

**WriteCurrent(set\_addr); //从指定地址独处数据并且返回**

**return (ReadCurrent());**

**}**

**//主函数**

**void main()**

**{**

**SDA=1; //SDA=1;SCL=1**

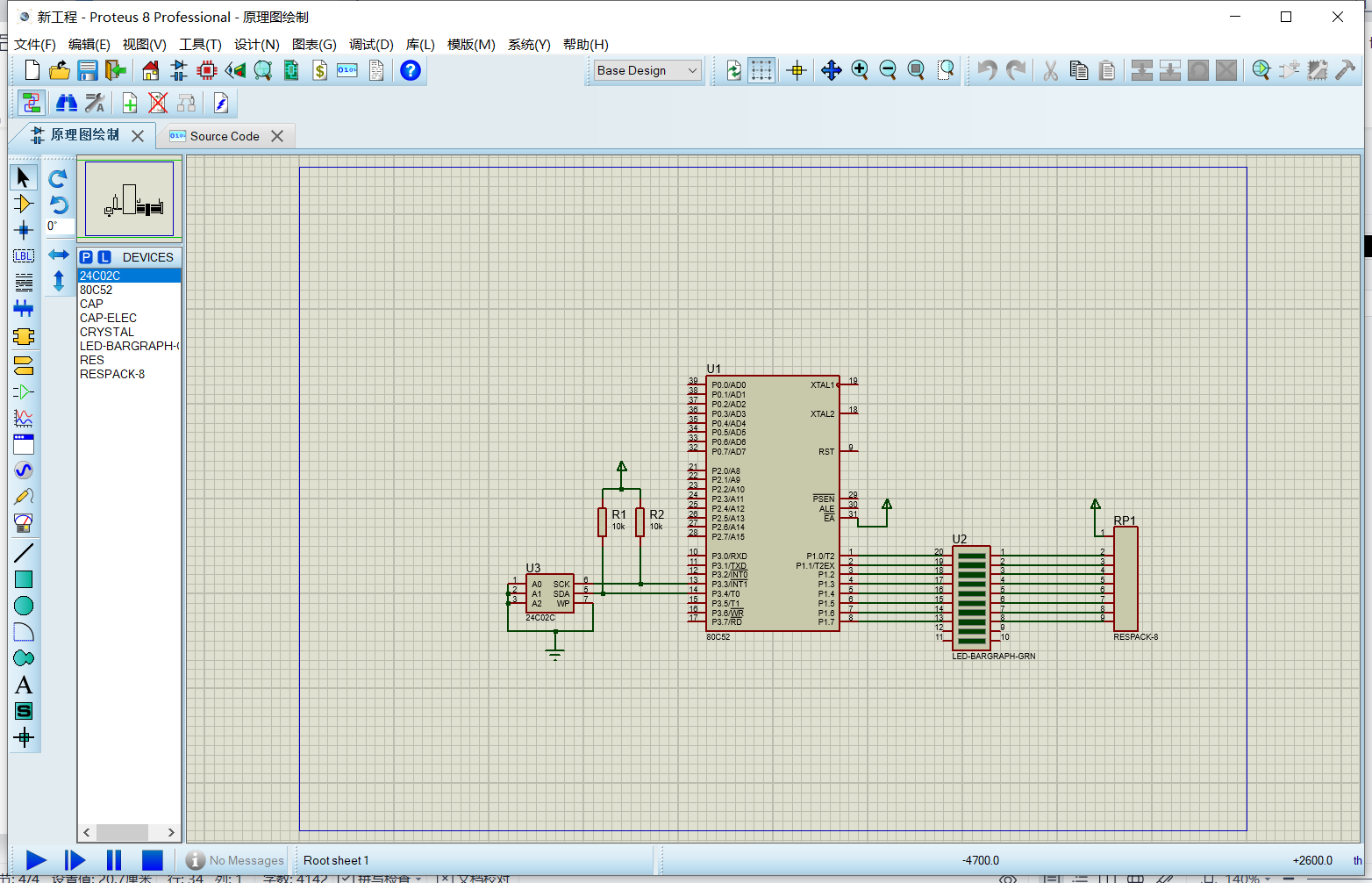
**SCL=1;**

**WriteSet(0x36,0x01); //在指定地址“0x36”中写入数据“0x01”**

**P1=ReadSet(0x36); //从指定地址，“0x36”中读取数据并送回P1显示**

**}**

**Protues仿真：**

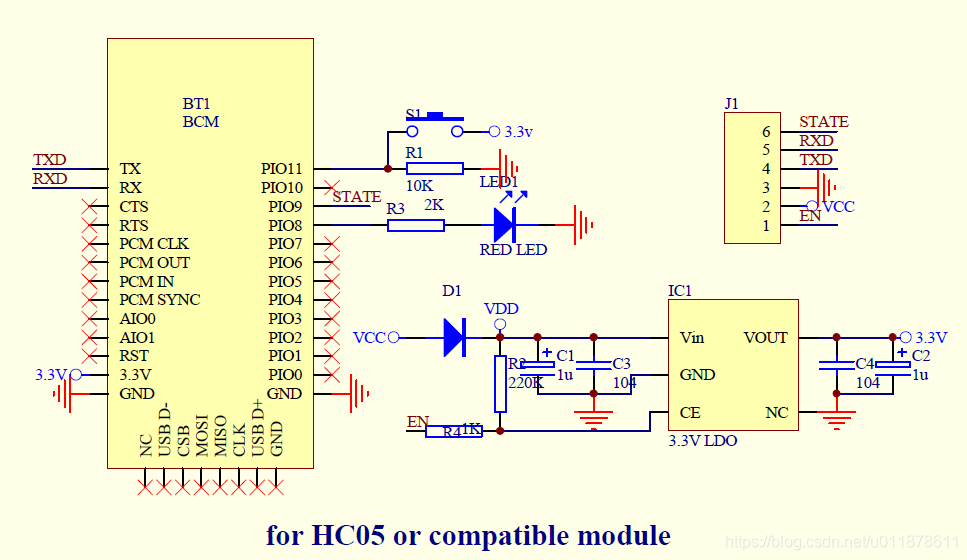


蓝牙模块

1.蓝牙模块HC05：

**HC05蓝牙模块**是主从一体的蓝牙串口模块，简单的说，当蓝牙设备与蓝牙设备配对连接成功后，我们可以忽视蓝牙内部的通信协议，直接将将蓝牙当做串口用。当建立连接，两设备共同使用一通道也就是同一个串口，一个设备发送数据到通道中，另外一个设备便可以接收通道中的数据。

原理图：



2. 蓝牙模块的调试

2.1 两种工作模式：

HC-05蓝牙串口通讯模块具有两种工作模式：命令响应工作模式和自动连接工作模式。在自动连接工作模式下模块又可分为主（Master）、从（Slave）和回环（Loopback）三种工作角色。

当模块处于自动连接工作模式时，将自动根据事先设定的方式连接的数据传输；

当模块处于命令响应工作模式时能执行AT命令，用户可向模块发送各种AT 指令，为模块设定控制参数或发布控制命令。

2.2 进入命令响应工作模式？

进入命令响应工作模式有两种方法：

模块上电，未配对情况下就是AT模式，波特率为模块本身的波特率，默认：9600，发送一次AT指令时需要置高一次PIO11；

PIO11 置高电平后，再给模块上电，此时模块进入AT 模式，波特率固定为：38400，可以直接发送AT指令。

2.3 什么叫做置高一次PIO11？

在蓝牙模块中有一个小按键，按一下就置高一次PIO11。也就是说，第一种方法需要每发送一次AT指令按一次；而第二种方式是长按的过程中上电，之后就无需再管了，直接发送AT命令即可。

需要注意一下，两种进入命令响应工作模式的方式使用的波特率是不一样的，建议使用第二种方式。

2.4 怎么区分进了命令响应工作模式呢？

在蓝牙模块上有灯，当灯快闪的时候，就是自动连接工作模式；当灯慢闪的时候，就是命令响应工作模式。

2.5 串口调试助手发送AT命令格式

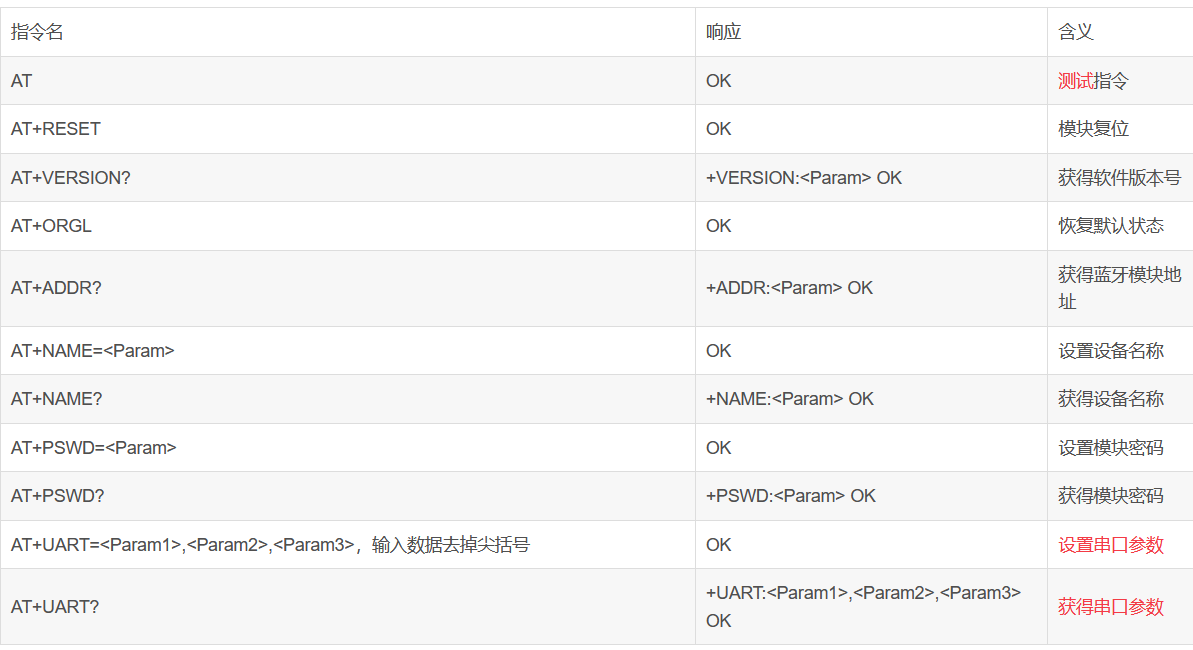
串口软件发送给HC-05蓝牙模块AT指令 ，必须严格按照格式发送才可以（即使看不到格式情况），否则返回错误ERROR或者没有反应，格式如下两种：

1.AT+一个回车；

2.AT命令+勾选发送新行

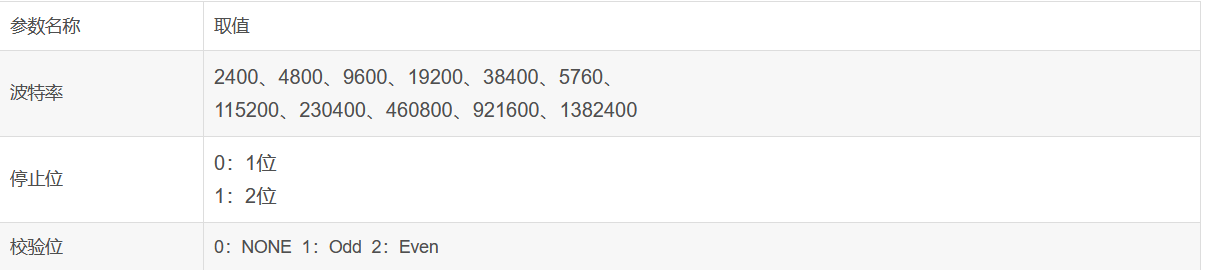
2.6 AT命令

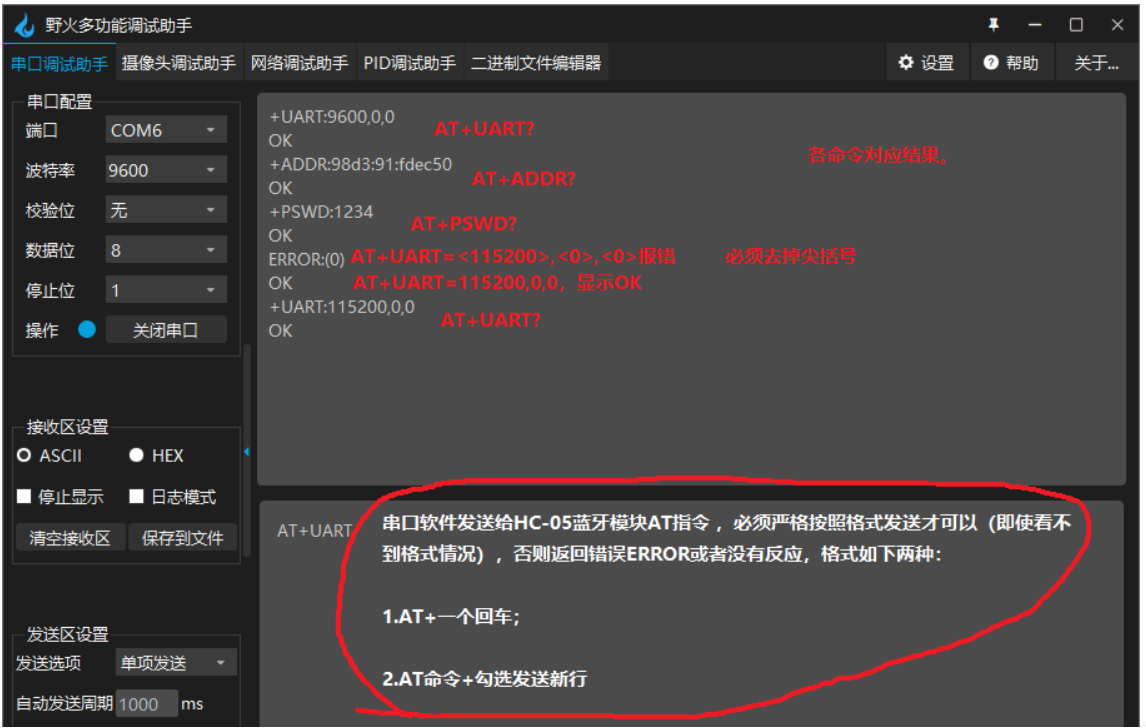
进入到命令响应工作模式之后，就可以使用串口调试助手进行蓝牙调试了。

首先有一点，AT指令不区分大小写，下面介绍常用的AT指令：

### 2.7 AT命令之设置串口

* AT+UART?：获得串口参数，串口的参数一共有三个，波特率、停止位、检验位。其取值如下：



设置蓝牙串口的波特率：115200。之后的内容，就会采用这个波特率来进行通讯。

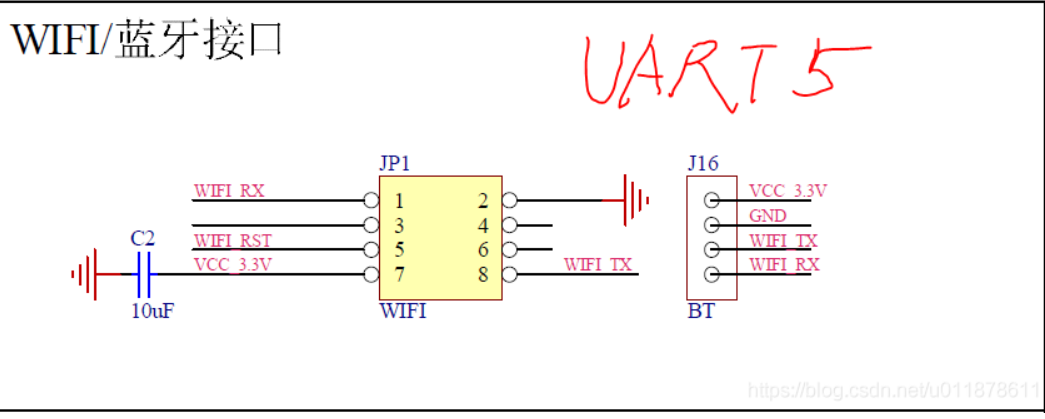
蓝牙模块与手机通讯

实现功能：手机通过蓝牙，向STM32单片机发送消息，STM32接收到消息之后原封不动的返回给手机，同时输入1、2、其它字符控制LED灯。

连线：使用USART5进行试验，也就是说STM32选取PC12(TX)、PD2(RX)来和HC-05进行连接，见下图UART5；同时手机通过蓝牙来和HC-05进行连接。

原理：手机通过蓝牙传输到HC-05上，再通过串口和STM32通信；而之前一般都是电脑上通过USB线转串口的方式，通过串口和STM32通信。本质上没有区别的。

说白了，只是个蓝牙转串口的设备，只要知道串口怎么编程使用，就可以了，实现了所谓的透明传输。蓝牙的相关一切都被封装起来了，都不需要接触到。



代码如下：

/\*串口初始化配置\*/

void UartConfiguration()

{

TMOD=0x20; //设置计数器1的工作方式2

TH1=0xfd; //设置计数器1的初值，决定波特率

TL1=0xfd; //设置计数器1的初值，决定波特率

PCON=0x00; // 波特率倍增0x00不加倍 0x80加倍

SCON=0x50; //设置工作方式1 开启接受允许

EA=1; //开启总中断

ES=1; //开启串口接受中断

TR1=1; //计数器1开始运行

}

/\*发送数据\*/

void sendDate(char date)

{

SBUF=date; //接收到的数据放入发送缓存器发送

while(!TI); //等待发送数据完成

TI=0; //清除发送完成标志位

}

/\*主函数\*/

void main()

{

UartConfiguration();

sendDate('1'); //发送字符1

while(1);

}

/\*中断函数\*/

void Uart() interrupt 4

{

uchar date;

date=SBUF; //取出接受到的数据

RI=0; //清除接受中断标志位

//收到的数据是date

}

其中while(!TI)是等待消息发送成功，如果发送不成功的话会一直等待阻塞程序。而且51单片机的串口中断是一个字节一个字节地接收数据的，比如手机向单片机发送了数据“123”，单片机中的中断函数Uart()会进入三次，把数据“123”分三次接收完，一次接收一个字符，所以要在其中添加自己的逻辑把单个字符组合成一个字符串来进行处理。

oled显示

OLED显示屏是一种由有机分子薄片组成的固态设备，施加电力之后就能发光。OLED能让电子设备产生更明亮、更清晰的图像，其耗电量小于传统的传统的LED显示屏。0.96寸OLED分辨率是128\*64，即OLED显示是128行64列;

OLED技术特点

（1） OLED 器件的核心层厚度很薄，厚度可以小于 1mm，为液晶的 1/3。（2） OLED 器件为全固态机构，无真空，液体物质，抗震性好，可以适应巨大的加速度，振动等恶劣环境。

（3） 主动发光的特性使 OLED 几乎没有视角限制，视角一般可达到 170 度，具有较宽的视角，从侧面也不会失真。

（4） OLED 显示屏的响应时间超过 TFT—LCD 液晶屏。TFT—LCD 的响应时间大约使几十毫秒，现在做得最好的 TFT—LCD 响应时间也只有 12 毫秒。而 OLED 显示屏的响应时间大约是几微秒到几十微秒。

（5） OLED 低温特性好，在零下 40 摄氏度都能正常显示，目前航天服上也使用OLED 作为显示屏。而 TFT—LCD 的响应速度随温度发生变化，低温下，其响应速度变慢，因此，液晶在低温下显示效果不好。（

6） OLED 采用有机发光原理，所需材料很少，制作上比采用液体发光的液晶工序少，液晶显示屏少 3 道工序，成本大幅降低。

（7） OLED 采用的二极管会自行发光，因此不需要背面光源，发光转化效率高，能耗比液晶低，OLED 能够在不同材质的基板上制造，厂家甚至可以将电路印刷在弹性材料上——做成能弯曲的柔软显示器。

（8） 低电压直流驱动，5V 以下，用电池就能点亮。高亮度，可达 300 明流以上。

OLED屏幕的通信方式分为2种：

1、iic通信

它是由数据线 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。

IIC总线进行数据传送时，时钟信号为高电平期间，数据线的数据必须保持稳定，只有在时钟信号为低电平期间，数据线上的高电平或低电平状态才允许变化。

开始信号：SCL 为高电平时，SDA 由高电平向低电平跳变，开始传送数据。

结束信号：SCL 为高电平时，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。

2、SPI通信

SPI通常有SCK时钟，STB片选，DATA数据信号三个信号。SPI总线真正实现了全双工数据传输，SPI 有3线跟4线两种，4线的话，就是多了一条叫SDC的线，用来告知从设备现在传输的是数据还是指令。这个接口较快，可以传输较连续的数据。SPI要想连接多个从设备，就需要给每个从设备配备一根片选信号。如果要可以实现全双工，也是需要多加一根数据线（MOSI MISO）。SPI通信芯片的引脚名称不一定都是这几个名称，可能还有会别的名称，但是意思是一样的，例如MOSI引脚的意思是“主机输出从机输入”，某个SPI接口的芯片就有可能会写成SDI，因为这个SPI器件是作为从机的，所以它的SDI的意思就是“从机数据输入引脚”。

SPI通信过程为：把CS引脚拉低，然后SCK输出时钟，然后就可以在MOSI引脚上输出数据，同时可以在MISO上获得数据了。

(1)SDO/MOSI （（master out slaver in））– 主设备数据输出，从设备数据输入;

(2)SDI/MISO – 主设备数据输入，从设备数据输出;

(3)SCLK – 时钟信号，由主设备产生;

(4)CS/SS – 从设备使能信号，由主设备控制。当有多个从设备的时候，因为每个从设 备上都有一个片选引脚接入到主设备机中，当我们的主设备和某个从设备通信时将需 要将从设备对应的片选引脚电平拉低或者是拉高。

IIC 接口：

GND： 电源地

VCC： 2.2V~5.5V

D/C：数据/命令，在 IIC 通信中 DC的高低电平是用来选择IIC通信地址的；

RES：复位（高电平 2.2V~5.5V）

SCL： CLK 时钟 （高电平 2.2V~5.5V）

SDA： MOSI 数据 （高电平 2.2V~5.5V）

SPI接线方式：

GND：电源地

VCC：2.2V~5.5V

SCL（D0）：CLK 时钟 （高电平 2.2V~5.5V）

SDA(D1)：MOSI 数据（高电平 2.2V~5.5V）

RST：复位（高电平 2.2V~5.5V）

D/C：数据/命令（高电平 2.2V~5.5V）

3.主要介绍iic协议通信的应用方式：

iic的工作模式就是用两根线来模拟iic得到数据，如图：

//-----------------OLED IIC端口定义----------------

#define OLED\_SCLK\_Clr() GPIO\_ResetBits(GPIOA,GPIO\_Pin\_5)//SDA IIC接口的时钟信号

#define OLED\_SCLK\_Set() GPIO\_SetBits(GPIOA,GPIO\_Pin\_5)

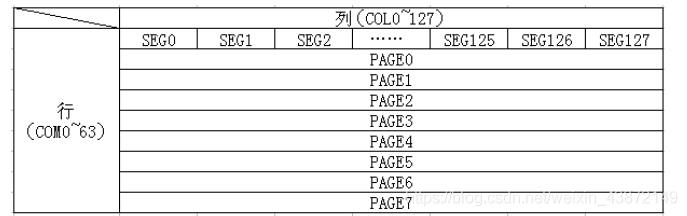
#define OLED\_SDIN\_Clr() GPIO\_ResetBits(GPIOA,GPIO\_Pin\_7)//SCL IIC接口的数据信号

#define OLED\_SDIN\_Set() GPIO\_SetBits(GPIOA,GPIO\_Pin\_7)

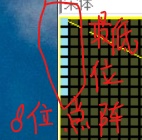
OLED控制器为SSD1306，也就是说：裸屏由SSD1306驱动，这也是一种较为广泛使用的led驱动芯片。

OLED模块显存

OLED本身是没有显存的，它的显存是依赖于SSD1306提供的。SSD1306的显存总共为128 \* 64bit大小，SSD1306将这些显存分为了8页。每页包含了128个字节，总共8页，这样刚好是128\*64的点阵大小。



但是由于OLED不能一次控制一个点阵,只能控制8个点阵;而且是垂直方向扫描控制;如下图;因此垂直方向坐标可选为0~~7;(8\*8=64);水平方向可选坐标0~127.



//OLED的显存

//存放格式如下.

//[0]0 1 2 3 ... 127

//[1]0 1 2 3 ... 127

//[2]0 1 2 3 ... 127

//[3]0 1 2 3 ... 127

//[4]0 1 2 3 ... 127

//[5]0 1 2 3 ... 127

//[6]0 1 2 3 ... 127

//[7]0 1 2 3 ... 127

u16 OLED\_GRAM[128][8];

//更新显存到LCD

void OLED\_Refresh\_Gram(void)

{

u8 i,n;

for(i=0;i<8;i++)

{

OLED\_WR\_Byte (0xb0+i,OLED\_CMD); //设置页地址（0~7）

OLED\_WR\_Byte (0x00,OLED\_CMD); //设置显示位置—列低地址

OLED\_WR\_Byte (0x10,OLED\_CMD); //设置显示位置—列高地址

for(n=0;n<128;n++)OLED\_WR\_Byte(OLED\_GRAM[n][i],OLED\_DATA);

}

}



1：命令0X81：设置对比度。包含两个字节，第一个0X81为命令，随后发送的一个字节为要设置的对比度的值。这个值设置得越大屏幕就越亮。

2：命令0XAE/0XAF：0XAE为关闭显示命令；0XAF为开启显示命令。

3：命令0X8D：包含2个字节，第一个为命令字，第二个为设置值，第二个字节的BIT2表示电荷泵的开关状态，该位为1，则开启电荷泵，为0则关闭。在模块初始化的时候，这个必须要开启，否则是看不到屏幕显示的。

4：命令0XB0~B7：用于设置页地址，其低三位的值对应着GRAM的页地址。

5：命令0X00~0X0F：用于设置显示时的起始列地址低四位。

6：命令0X10~0X1F：用于设置显示时的起始列地址高四位。

## 代码详解

#define OLED\_CMD 0 //写命令

#define OLED\_DATA 1 //写数据

对oled进行初始化

//初始化SSD1306

void OLED\_Init(void)

{

GPIO\_InitTypeDef GPIO\_InitStructure;

RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_GPIOA, ENABLE); //使能A端口时钟

GPIO\_InitStructure.GPIO\_Pin = GPIO\_Pin\_5|GPIO\_Pin\_7;

GPIO\_InitStructure.GPIO\_Mode = GPIO\_Mode\_Out\_PP ; //推挽输出

GPIO\_InitStructure.GPIO\_Speed = GPIO\_Speed\_50MHz;//速度50MHz

GPIO\_Init(GPIOA, &GPIO\_InitStructure); //初始化GPIO

GPIO\_SetBits(GPIOA,GPIO\_Pin\_5|GPIO\_Pin\_7);

delay\_ms(200);

OLED\_WR\_Byte(0xAE,OLED\_CMD);//--display off

OLED\_WR\_Byte(0x00,OLED\_CMD);//---set low column address

OLED\_WR\_Byte(0x10,OLED\_CMD);//---set high column address

OLED\_WR\_Byte(0x40,OLED\_CMD);//--set start line address

OLED\_WR\_Byte(0xB0,OLED\_CMD);//--set page address

OLED\_WR\_Byte(0x81,OLED\_CMD); // contract control

OLED\_WR\_Byte(0xFF,OLED\_CMD);//--128

OLED\_WR\_Byte(0xA1,OLED\_CMD);//set segment remap

OLED\_WR\_Byte(0xA6,OLED\_CMD);//--normal / reverse

OLED\_WR\_Byte(0xA8,OLED\_CMD);//--set multiplex ratio(1 to 64)

OLED\_WR\_Byte(0x3F,OLED\_CMD);//--1/32 duty

OLED\_WR\_Byte(0xC8,OLED\_CMD);//Com scan direction

OLED\_WR\_Byte(0xD3,OLED\_CMD);//-set display offset

OLED\_WR\_Byte(0x00,OLED\_CMD);//

OLED\_WR\_Byte(0xD5,OLED\_CMD);//set osc division

OLED\_WR\_Byte(0x80,OLED\_CMD);//

OLED\_WR\_Byte(0xD8,OLED\_CMD);//set area color mode off

OLED\_WR\_Byte(0x05,OLED\_CMD);//

OLED\_WR\_Byte(0xD9,OLED\_CMD);//Set Pre-Charge Period

OLED\_WR\_Byte(0xF1,OLED\_CMD);//

OLED\_WR\_Byte(0xDA,OLED\_CMD);//set com pin configuartion

OLED\_WR\_Byte(0x12,OLED\_CMD);//

OLED\_WR\_Byte(0xDB,OLED\_CMD);//set Vcomh

OLED\_WR\_Byte(0x30,OLED\_CMD);//

OLED\_WR\_Byte(0x8D,OLED\_CMD);//set charge pump enable

OLED\_WR\_Byte(0x14,OLED\_CMD);//

OLED\_WR\_Byte(0xAF,OLED\_CMD);//--turn on oled panel

}

在我们平时使用的自模中，有两种常见的取模方式，一个是 6 \* 8，另一个则是8 \* 16的，第一个说的是在8行6列的矩形表格中取出我们想要的字符，第二个则是在16行8列的矩形表格中取出字符。正如下面代码注释中写的一样，因为oled中每一页只有8个行，所以就需要使用下一页的空间。所以就有了我们平时使用选择的字体大小，当然，这些都是常用的字体大小，我们也可以自己通过字符取模软件制作自己喜欢的字体大小

/\* 在指定位置显示一个字符,包括部分字符

x:0~127

y:0~63

size:选择字体 16/12\*/

void OLED\_ShowChar(u8 x,u8 y,u8 chr,u8 Char\_Size)

{

unsigned char c=0,i=0;

c=chr-' ';//得到偏移后的值 可从字模中得到,第一个为' ',减去即可得到相应的字符

if(x>Max\_Column-1){x=0;y=y+2;} //Max\_Column：最大列：128; x：设置列数; y:设置页数

if(Char\_Size ==16) //此时需要两页的同一列，8\*16的点阵

{

OLED\_Set\_Pos(x,y); //若 x = y = 2,则设置的为第3页的第3列， 注意：每一页只有八行

for(i=0;i<8;i++)

OLED\_WR\_Byte(F8X16[c\*16+i],OLED\_DATA); //通过i的递增，循环画点，此时将第2页第2列的8行都写入了数据

OLED\_Set\_Pos(x,y+1); //由于画点的数目行数不够，此时需要第3页的第2列来续画点

for(i=0;i<8;i++)

OLED\_WR\_Byte(F8X16[c\*16+i+8],OLED\_DATA); //接着画完，直到第16个点结束

}

else

{

OLED\_Set\_Pos(x,y); //6\*8的点阵，不需要其他的页来续画

for(i=0;i<6;i++)

OLED\_WR\_Byte(F6x8[c][i],OLED\_DATA); //二维数组，c控制第几行，i控制第几列，故不需要其他的操作即可画完

}

}

void OLED\_ShowString(u8 x,u8 y,u8 \*chr,u8 Char\_Size) //显示字符串

{

unsigned char j=0;

while (chr[j]!='\0') //判断字符串是否结束

{

OLED\_ShowChar(x,y,chr[j],Char\_Size); // 一个一个画字符

x+=8; //x 设置的是列，一个字符的大小为8\*16，即行16列8，每次显示为一个后，都需要向高列移8列

if(x>120){x=0;y+=2;} // 最高为128列，超过的话则需要重新从零列开始，由于此时需要别的页数来续画，避免重叠，需要 y += 2。

j++; //循环画字符串

}

}

显示2个数字，具体都在下面的代码中写出，**需要注意的是，下面的 " " 表示的是ASCII值32**

**//m^n函数**

**u32 oled\_pow(u8 m,u8 n)**

**{**

**u32 result=1;**

**while(n--)result\*=m;**

**return result;**

**}**

**//显示2个数字**

**//x,y :起点坐标**

**//len :数字的位数**

**//size:字体大小**

**//mode:模式 0,填充模式;1,叠加模式**

**//num:数值(0~4294967295);**

**void OLED\_ShowNum(u8 x,u8 y,u32 num,u8 len,u8 size2)**

**{**

**u8 t,temp;**

**u8 enshow=0;**

**for(t=0;t<len;t++)**

**{**

**temp=(num/oled\_pow(10,len-t-1))%10;**

**if(enshow==0&&t<(len-1))**

**{**

**if(temp==0)**

**{**

**OLED\_ShowChar(x+(size2/2)\*t,y,' ',size2);**

**continue;**

**}else enshow=1;**

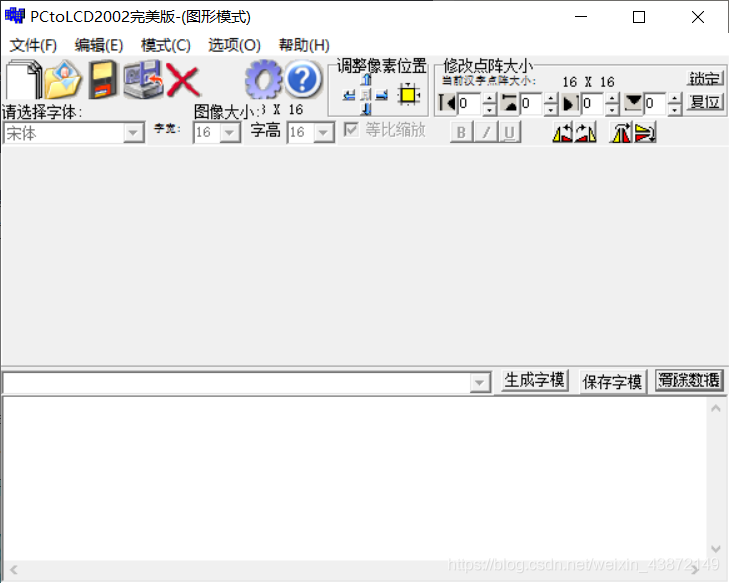
**}**

**OLED\_ShowChar(x+(size2/2)\*t,y,temp+'0',size2);**

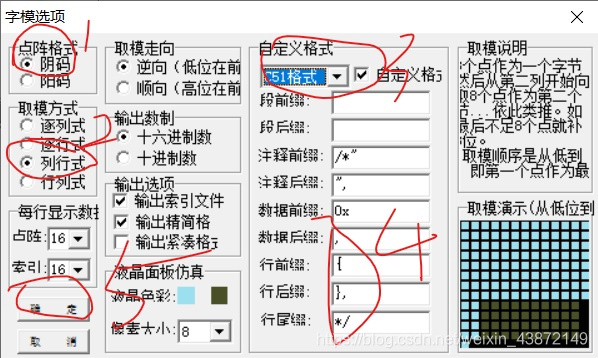
**}**

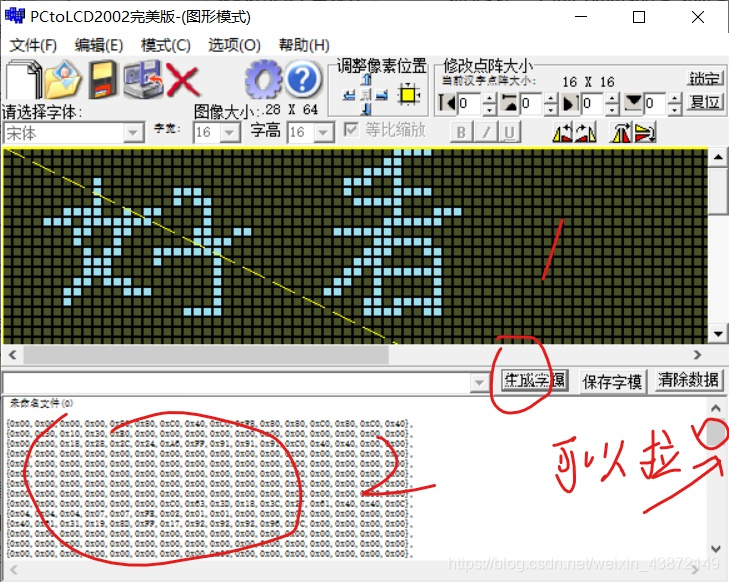
**}**

**取模软件：PCtoLCD2002**



设置方面：





红外遥控

## 1.****什么是红外遥控****

红外遥控系统主要由红外发射装置和红外接收装置构成。  
红外发射装置主要由键盘电路、红外编码芯片、电源和红外发射电路组成（比如遥控器），如图：



红外接收装置主要由红外接收电路、红外解码芯片、电源和应用电路组成，在[单片机](https://so.csdn.net/so/search?q=%E5%8D%95%E7%89%87%E6%9C%BA&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/qq_43743762/article/details/_blank)开发板上有红外接收电路，而且单片机充当解码芯片。

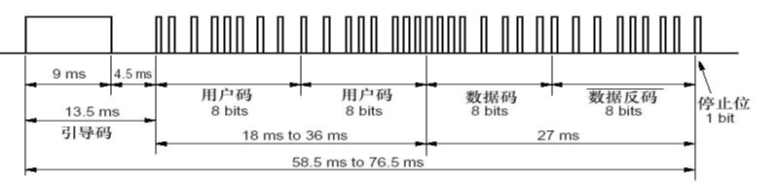
2.红外信号是怎么传输的

人的眼睛能看到的可见光按波长从长到短排列依次为：红、橙、黄、绿、青、蓝、紫。其中红光的波长范围为0.62~0.72微米，红外遥控就是利用波长为0.76至1.5微米之间的近红外线来传送控制信号的。

红外信号的发射由红外发射电路中的红外发光二极管完成，通常情况下为了提高抗干扰能力与降低电源消耗，遥控器将遥控信号（二进制脉冲码）调制在载波（载波是传送信息的物理基础和承载工具）上经放大后发送至红外二极管，再由二极管转换为红外信号发送出去。遥控器上不同的按键有着不一样的键值，按下相对应的键，红外二极管就会发送对应的信号，接收装置接收到信号后会对信号进行信号解调后会得到相应按键的键值，再根据不同的键值执行相应的操作。

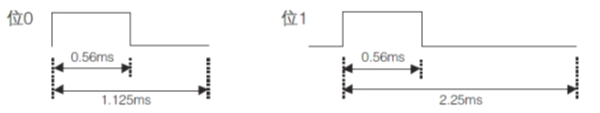
## 3.****NEC协议****

就像串口通信一样，红外通信也有其自己的通信协议，我们一般遵循NEC协议。  
NEC规定红外二极管每次**发送的信号的数据格式**如下：

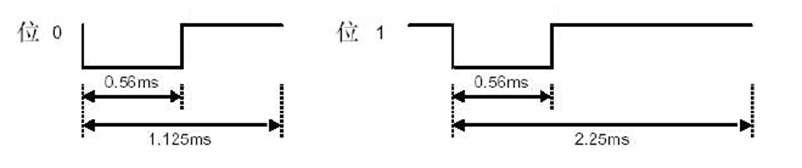


发送的数据主要由引导码、用户码1、用户码2、数据码、数据反码组成。引导码相当于暗号，当接收装置接收到正确的引导码后就开始读取数据，俩组用户码是为了区别其他的红外发射器发射的信号，数据码中的内容就对应着相应的键值，数据反码是为了在接收信号后检验信号的正确性。（引导码是“9ms高电平+4.5ms低电平”）

这样我们虽然清楚了信号的发送形式，但是怎么表达数据发送中的高低位（“0"和"1”）呢？这又涉及到NEC的位定义了，规定：“0.56ms高电平+0.565ms低电平”代表“1”；“0.56ms高电平+1.69ms低电平”代表“0”，就是说“0”与“1”的不同就是他们低电平所持续的时间不同。（PS：发送数据时从最低位开始发送）



**单片机接收到的数据格式与发送时的恰恰相反，就是说接收时的引导码是“9ms低电平+4.5ms高电平”；“0”是“0.56ms低电平+0.565ms高电平”；“1”是“0.56ms低电平+1.69ms高电平”。这一点我们要格外注意**



部分代码：

//外部中断0中断函数，下降沿触发执行

void Int0\_Routine(void) interrupt 0

{

if(IR\_State==0) //状态0，空闲状态

{

Timer0\_SetCounter(0); //定时计数器清0

Timer0\_Run(1); //定时器启动

IR\_State=1; //置状态为1

}

else if(IR\_State==1) //状态1，等待Start信号或Repeat信号

{

IR\_Time=Timer0\_GetCounter(); //获取上一次中断到此次中断的时间

Timer0\_SetCounter(0); //定时计数器清0

//如果计时为13.5ms，则接收到了Start信号（判定值在12MHz晶振下为13500，在11.0592MHz晶振下为12442）

if(IR\_Time>13500-500 && IR\_Time<13500+500)

{

IR\_State=2; //置状态为2

}

//如果计时为11.25ms，则接收到了Repeat信号（判定值在12MHz晶振下为11250，在11.0592MHz晶振下为10368）

else if(IR\_Time>11250-500 && IR\_Time<11250+500)

{

IR\_RepeatFlag=1; //置收到连发帧标志位为1

Timer0\_Run(0); //定时器停止

IR\_State=0; //置状态为0

}

else //接收出错

{

IR\_State=1; //置状态为1

}

}

else if(IR\_State==2) //状态2，接收数据

{

IR\_Time=Timer0\_GetCounter(); //获取上一次中断到此次中断的时间

Timer0\_SetCounter(0); //定时计数器清0

//如果计时为1120us，则接收到了数据0（判定值在12MHz晶振下为1120，在11.0592MHz晶振下为1032）

if(IR\_Time>1120-500 && IR\_Time<1120+500)

{

IR\_Data[IR\_pData/8]&=~(0x01<<(IR\_pData%8)); //数据对应位清0

IR\_pData++; //数据位置指针自增

}

//如果计时为2250us，则接收到了数据1（判定值在12MHz晶振下为2250，在11.0592MHz晶振下为2074）

else if(IR\_Time>2250-500 && IR\_Time<2250+500)

{

IR\_Data[IR\_pData/8]|=(0x01<<(IR\_pData%8)); //数据对应位置1

IR\_pData++; //数据位置指针自增

}

else //接收出错

{

IR\_pData=0; //数据位置指针清0

IR\_State=1; //置状态为1

}

if(IR\_pData>=32) //如果接收到了32位数据

{

IR\_pData=0; //数据位置指针清0

if((IR\_Data[0]==~IR\_Data[1]) && (IR\_Data[2]==~IR\_Data[3])) //数据验证

{

IR\_Address=IR\_Data[0]; //转存数据

IR\_Command=IR\_Data[2];

IR\_DataFlag=1; //置收到连发帧标志位为1

}

Timer0\_Run(0); //定时器停止

IR\_State=0; //置状态为0

}

}

}

4.程序分析

敲代码之前首先要对程序有一个清楚的认识，即明白要做哪几个模块？这些模块之间要怎么联系起来？在下做的一个程序是把从红外发射器中接收到数据经过处理后显示在数码管上，这样看来我的程序就需要如下几个模块：

1.初始化模块：包括初始化红外接收引脚（把P3.2外部中断引脚作为红外接收引脚）、打开外部中断允许位；

2.延时模块：延时模块主要用来分析高低电平持续的时间，从而判断对应的位是“0”还是“1”；

3.解码模块：利用外部中断函数interrupt 0进行数据解码，其任务是检验并接收正确的数据信号，对用户码和数据码进行分析，即通过检测高低电平持续的时间来判断相应的位是“0”还是“1”。然后得出相对应的键值，把键值对应的数反馈到程序中；

4.显示模块：其主要任务是接收解码模块反馈的数值，并将数值显示在数码管上；

5.主函数模块：整合前几个模块。

代码实现思路

使用状态位IR\_State，IR\_State0表示空闲状态，IR\_State1表示等待Start信号或Repeat信号，IR\_State2表示接收信号。

上篇说到为了响应迅速故使用外部中断，由于每个数据位的发送都是以下降沿结束，所以使用下降沿触发的外部中断。因为数据位0或1是由高低电平的时间决定，所以使用定时器定时时间来确定数据位是多少。

红外线接收装置的OUT引脚连接到了外部中断引脚上，当接收装置接收到红外线信号时将会进入外部中断函数。

首次进入到中断函数中，已经产生了外部中断，此时把IR\_State置为1进而判断是Start信号还是Repeat信号。进入中断后可能是开始信号也可能是重复信号，差别是中断时间的不同 ，因此判断时间的多少就可以识别出是开始信号还是结束信号。判断时间时在保证时间不会交集的情况下可以判断时间的范围，因为定时也不一定准确，

若判断的时间长度为开始信号时间的长度，IR\_State置为2，判断的时间长度为重复信号时间的长度，IR\_State置为0，如果这两个都不是，说明接收出错，则把这次接收结果舍去，IR\_State清零，准备下次接收。

当IR\_State2时，说明已接受到开始信号，准备开始接收数据。组成数据的“0”，“1”码是根据高低电平的持续时间确定的，所以只要检测定时器的计数值就能确定数据。

采用了以下算法：

else if(IR\_State==2) //状态2，接收数据

{

IR\_Time=Timer0\_GetCounter(); //获取上一次中断到此次中断的时间

Timer0\_SetCounter(0); //定时计数器清0

//如果计时为1120us，则接收到了数据0（判定值在12MHz晶振下为1120，在11.0592MHz晶振下为1032）

if(IR\_Time>1120-500 && IR\_Time<1120+500)

{

IR\_Data[IR\_pData/8]&=~(0x01<<(IR\_pData%8)); //数据对应位清0

IR\_pData++; //数据位置指针自增

}

//如果计时为2250us，则接收到了数据1（判定值在12MHz晶振下为2250，在11.0592MHz晶振下为2074）

else if(IR\_Time>2250-500 && IR\_Time<2250+500)

{

IR\_Data[IR\_pData/8]|=(0x01<<(IR\_pData%8)); //数据对应位置1

IR\_pData++; //数据位置指针自增

}

else //接收出错

{

IR\_pData=0; //数据位置指针清0

IR\_State=1; //置状态为1

}

if(IR\_pData>=32) //如果接收到了32位数据

{

IR\_pData=0; //数据位置指针清0

if((IR\_Data[0]==~IR\_Data[1]) && (IR\_Data[2]==~IR\_Data[3])) //数据验证

{

IR\_Address=IR\_Data[0]; //转存数据

IR\_Command=IR\_Data[2];

IR\_DataFlag=1; //置收到连发帧标志位为1

}

Timer0\_Run(0); //定时器停止

IR\_State=0; //置状态为0

}

}

IR\_Data[IR\_pData/8] 划分了四个8位数据，IR\_Data[0],IR\_Data[1],IR\_Data[2],IR\_Data[3]分别代表地址码，地址码反码，命令，命令反码

当判断的时间长度为“0”的时间长度时：则把该数据位清0。

代码如下：

if(IR\_Time>1120-500 && IR\_Time<1120+500)

{

IR\_Data[IR\_pData/8]&=~(0x01<<(IR\_pData%8)); //数据对应位清0

IR\_pData++; //数据位置指针自增

}

当判断的时间长度为“0”的时间长度时：则把该数据位置1。

else if(IR\_Time>2250-500 && IR\_Time<2250+500)

{

IR\_Data[IR\_pData/8]|=(0x01<<(IR\_pData%8)); //数据对应位置1

IR\_pData++; //数据位置指针自增

}

当判断的时间长度既不是“0”的时间长度也不是“1”的是时间长度时，说明数据接收错误，数据位置指针清0，重新接收。

else //接收出错

{

IR\_pData=0; //数据位置指针清0

IR\_State=1; //置状态为1

}

因为共需要接收4个8位数据，共32位，所以当地址指针等于32时说明接收完毕，然后就能进入判断：

if(IR\_pData>=32) //如果接收到了32位数据

{

IR\_pData=0; //数据位置指针清0

if((IR\_Data[0]==~IR\_Data[1]) && (IR\_Data[2]==~IR\_Data[3])) //数据验证

{

IR\_Address=IR\_Data[0]; //转存数据

IR\_Command=IR\_Data[2];

IR\_DataFlag=1; //置收到连发帧标志位为1

}

Timer0\_Run(0); //定时器停止

IR\_State=0; //置状态为0

}