HOT-51 增强版开发板用户手册

版本：HOT-51-B-20100718-V2.0

注意事项：

为了安全有效的使用该产品，请您使用前仔细阅读以下信息。

- 做继电器实验时，请务必弄清继电器的工作原理以及接法。为了人身安全，我们建议接32V以下电压，虽然继电器可以控制220V电压。
- 本产品可以通过 USB 供电，供电电流<500mA。外接电源时请注意电源极性和电压参数，最好是购买本店配套电源。电压范围最好在7.5V--12V较合适。
- 请不要在带电时插拔芯片以及相关器件。
- 插单片机和温度DS18B20时，注意方向。
- 自行搭接电路导致不良故障，我们不负任何责任。

质量三包：

一、一个月内出现质量问题，免费维修。
二、终身保修，买家只需要出元件维修成本和来回邮费。

联系方式：

邮箱：ct-315@163.com

QQ : 85536436

网站：www.dlmcu.net

我们真诚的感谢您对我们的支持。

文件更改履历表

编号	日期	版本	说明	备注
1	2009-9-19	V0.6	初始讨论稿	
2	2010-3-15	V1.0	正式稿	
3	2010-7-20	V2.0	HOT-51 增强型初稿	
4				
5				
6				
7				
8				
9				
10				
11				
12				
13				
14				
15				
16				
17				

第一章 HOT-51 单片机开发板系统介绍

1.1 HOT-51 单片机开发板简介

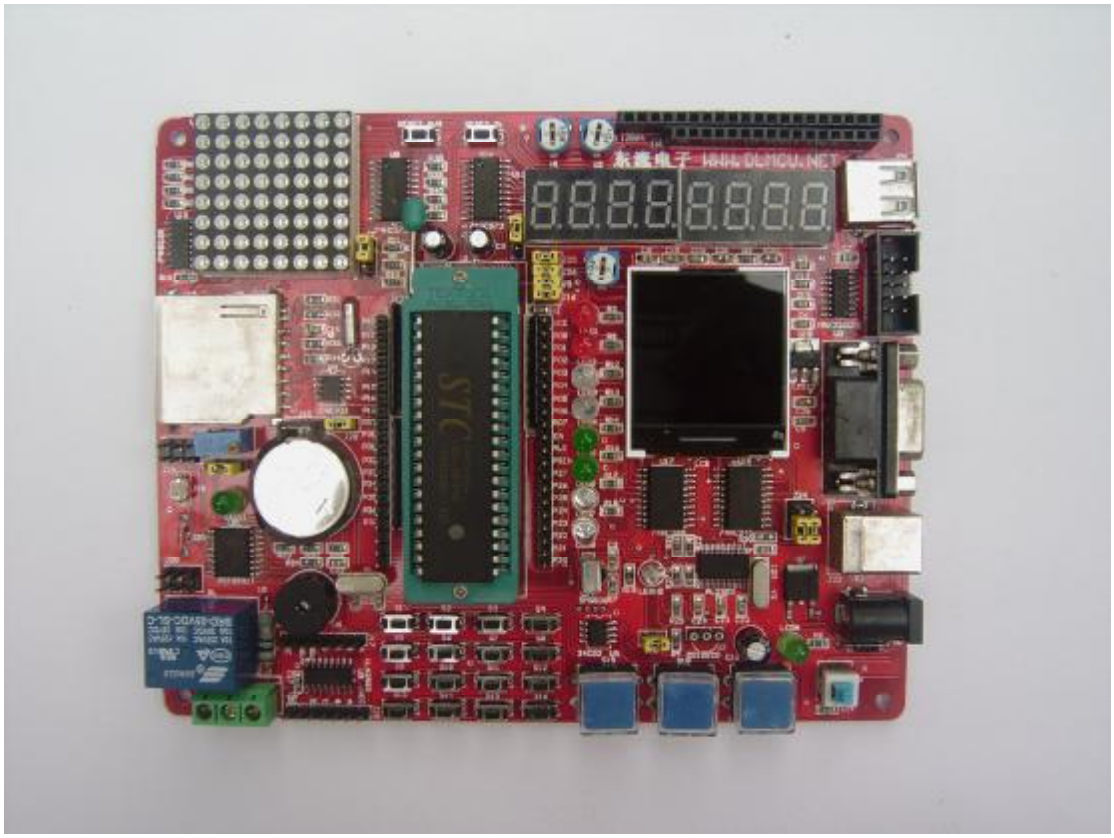
HOT-51 增强型开发板是在 HOT-51 开发板的基础上改进而成的。HOT-51 开发板经过一年的努力，已经为全国数百个高校学子及各地电子爱好者提供了数千套开发板，功能全，质量可靠，得到广泛好评。

HOT-51 增强型开发板在基本保留了 HOT-51 开发板的功能外，增加了彩屏、SD 卡座、光敏电阻、热敏电阻、红外发射、板载一线式下载电路、PS2 接口、时钟芯片电池等功能。

HOT-51 在遵循实用的原则，并且全部为贴片工艺，加了彩屏和铝合金外壳，更美观，更精致。可以说该开发板是单片机开发板行业中的精品。

在这里，东流电子再次感谢大家对我们的支持。我们将提供最优质，最真诚的售后服务。

1.1.1 HOT-51 开发板图片



1.1.2 HOT-51 增强型开发板硬件资源特性（以下器件不全是标配）

芯片类：

- 单片机：我们标配的是 STC12C5A60S2
- AD-DA 芯片：PCF8591T
- 温度传感器：1-WIRE 协议控制芯片 DS18B20
- 红外接头：PC 838(配合遥控器做解码试验）
- 时钟芯片：SPI 协议控制芯片 DS1302
- 储存芯片：I2C 协议控制芯片 AT24C02
- 通讯芯片：MAX232
- USB 转串口芯片：PL2303HX
- 达林顿管：ULN2003(驱动步进电机，直流电机，继电器，蜂鸣器）
- 三态缓冲门电路：74HC245
- 三八译码器：74HC138
- 锁存器：74HC573
- 稳压芯片：7805、HT7130

显示类：

- 彩屏液晶
- 八位 LED 灯
- 八位共阴数码管
- 8*8 点阵
- 1602 液晶
- 12864 液晶（带汉字字库）

其他：

- 精致独立按键
- 4*4 矩阵键盘
- 双复位电路
- 步进电机
- 直流电机
- 继电器
- 蜂鸣器
- 遥控器
- 光敏电阻
- 热敏电阻
- 时钟电池
- SD 卡座
- PS2 接口
- USB 电源接口
- 外接电源接口
- 串口接口
- 标准 JTAG 接口

1.2 硬件说明

1.2.1 IO 口分配说明

- LED：接 P0 口
- 点阵：阳极通过 74HC573 接 P0 口，阴极接 P2 口
- 数码管：段码通过 74HC573 接 P0 口，位选码通过 74HC138 接 P2.0,P2.1,P2.2
- 1602 液晶：RS 接 P2.5，R/W 接 P2.6,E 接 P2.7，数据口接 P0
- 12864 液晶：RS 接 P2.5,R/2 接 P2.6,E 接 P2.7，数据口接 P0，PSB 接 2.4，RST 接 2.2
- 独立按键：分别接 P3.2,P3.3,P3.4
- 矩阵键盘: 接 P1 口
- 时钟芯片 DS1302：SCLK 接 P1.6,RST 接 P1.7，I/O 接 P3.5
- 储存芯片 AT24C02:SCLK 接 P1.5，SDA 接 P3.6
- 通讯芯片 MAX232：接 P3.0,P3.1
- USB 转串口芯片 PL2303：接 P3.0,P3.1(注：这里有跳线帽进行选择)
- 红外解码芯片：接 P3.3
- PS2 接口：连接 P1.6,P1.7
- SD 卡座接口：接 P1.0,P1.1,P1.2,P1.3
- AD/DA 芯片 PCF8591T：接 P1.5,P3.6
- 继电器，蜂鸣器，步进电机，直流电机：最好用 P1 口控制。单片机 IO 控制 ULN2003 芯片去驱动以上器件。

1.3 用户测试开发板

当您拿到开发板时，你可以把“开发板配套资料”中的“程序代码”中编译好的 HEX 文件分别烧到单片机中，测试开发板的各各功能。当然，你需要学习下面 2.1.1 和 2.1.2 的程序下载。

1、LED、数码管和液晶的测试的测试都可以看到明显的现象。

2、测试点阵时需要把 J16（J16 位于数码管左下端）的跳线用短路帽断上。这样才可以启用点阵功能。

3、独立按键测试时，按 P3.2、P3.3、P3.4 时，LED1、LED2、LED3 会相应电亮。测试矩阵键盘时，LED 会有相应的显示。

以下四个测试要借助杜邦线。

4、直流电机测试。P1.0 接 J17 的 DC 输入(蜂鸣器下面一样排针最左边排针)。直流电机接 J18 排针的两个端口。

5、继电器测试。P1.1 接 J17 的 RELAY 端（蜂鸣器下面一样排针左二的排针）。

分别按独立按键 INTO 和 TNT1 时，会听到继电器弹片动作的声音，并且可以用万用表测继电器的输出端。

6、蜂鸣器测试。P1.2 接 BELL 端(蜂鸣器下面一样排针左三的排针）。

7、步进电机测试。P1.3 接 J17 的 D 端、P1.4 接 J17 的 C 端、P1.5 接 J17 的 B 端、P1.6 接 J17 的 A 端。步进电机接右 ULN2003 的右下角的排针。步进电机的红线对应 VCC。

第二章 HOT-51 增强型开发板使用说明

2.1 单片机开发流程

2.1.1 USB 转串口线的驱动安装

HOT-51 增强版开发板配备了两个串口电路。
一个是 MAX232 电路，一个是 PL2303 电路。
PL2303 电路 ： 我们板载了 USB 转串口芯片 PL2303。可以用一条 USB 线就可以下载了。安装 PL2303 的驱动，驱动在“\开发板配套资料\USB 转串口线驱动\PL2303”中。
MAX232 电路 ：如果您电脑没有串口，就需要用 USB 转串口线进行下载。安装驱动 340 的驱动，驱动在“\开发板配套资料\USB 转串口线驱动\CH340”中。
HOT-51 增强型开发板，是利用板上的 J24 对使用哪个串口进行选择。跳线跳到中下两端，PL2303；到中上两端，使用 MAX232。

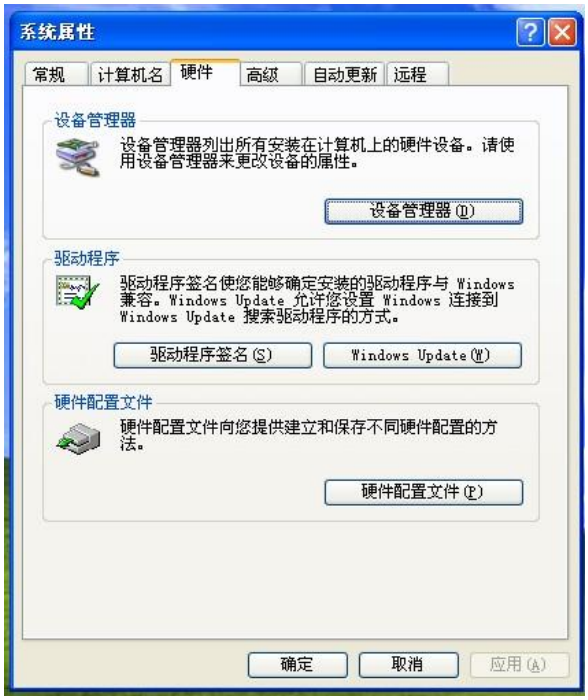
2.1.1.1 PL2303 驱动的安装

1.安装完成之后，按照提示的信息，必须重启计算机。
2.如果你的电脑已经安装此驱动或同类不同版本的驱动时，必须先删除原驱动后，重启计算机，才能再次安装。否则，将提示“无法安装新硬件”。
3.请不要使用同类其它版本的驱动，否则将会影响开发箱使用性能，使用前务必把我们的提供的驱动安装一次。
4、我们在 USB 转串口线驱动下面的 PL2303 目录下面，有两个目录，一个是 XP，一个是 W7，顾客根据自己的需要，选择要安装的驱动。
插上 USB 线，然后根据提示安装即可。
如果，提示已经存在该驱动，则需要先卸载 PL2303 驱动，再重启电脑后，重新安装。

2.1.1.2 CH340 驱动的安装

点击"INSTALL",进行安装。

2.1.2 COM 口的查看



点击“设备管理器”

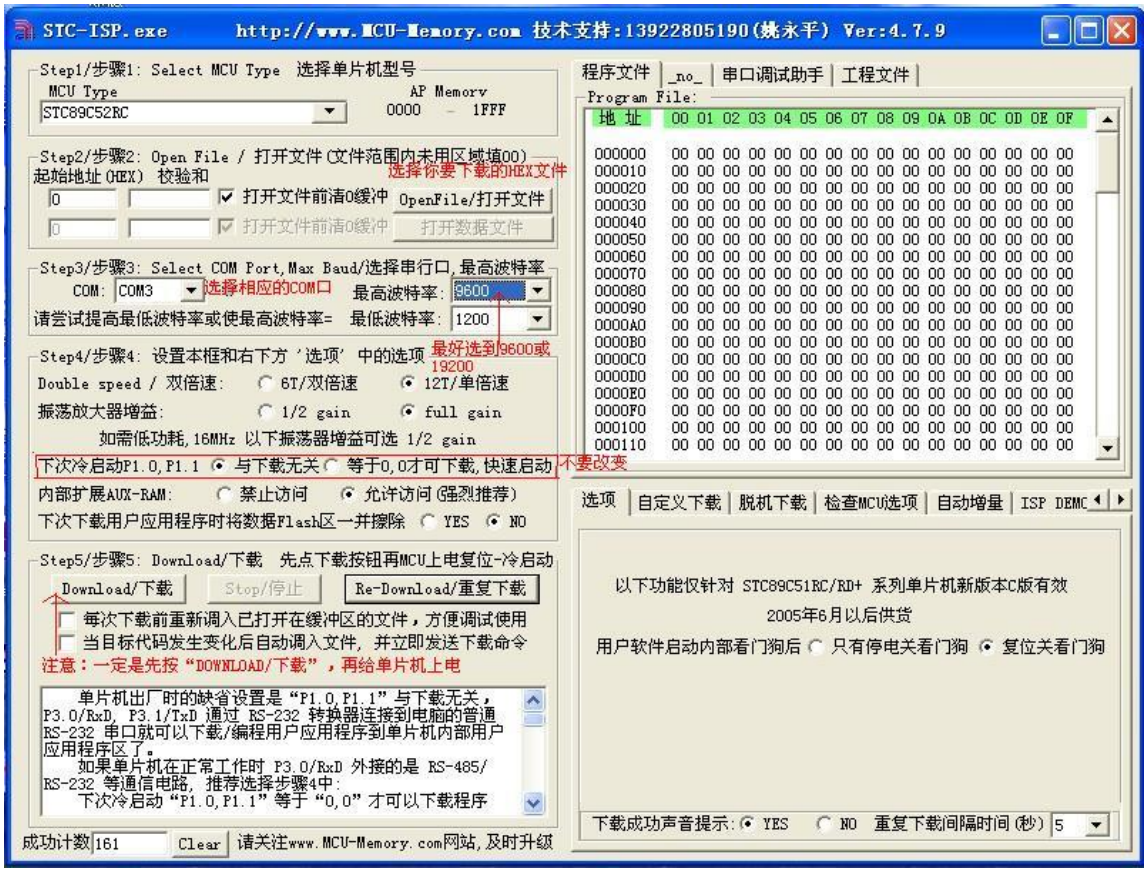
然后插上 USB 转串口线,会看到端口中多了一个“USB-SERIAL CH340（COM6)”，这个就是你的 USB 转串口线在你这台电脑所占用的 COM 口。COM* 不同的电脑会不同。

串口调试助手只有 COM1--COM4，因此，需要更改串口的编号，这在我们光盘中的有相应的资料。

2.1.3 第一个程序的下载

首先：打开 STC-ISP.Exe 软件，位置在“HOT-51\开发板配套资料\开发软件\STC

最新下载软件”下面的 STC-ISP_V480-Setup.Exe,软件为免安装版本，先双击该文件解压，会出现下载软件 STC-ISPV480.EXE 这个文件。



说明：1、首先在 MCU TYPE 中选择芯片，我们这里选择 STC12C5A60S2。

2、选择你要下载的 HEX 文件，在“开发板配套资料\实例程序”文件夹中我们准备了一个流水灯编译好了的 HEX 文件，“流水灯.HEX”。

3、如果用 PL2303 下载，接 USB 电源线。如果用 MAX232 下载，接 USB 电源线和串口线。

4、COM 口选择你插上 USB 线时“设备管理器”中的“端口”中看到的 COM 口。

5、选择波特率。如果出现握手失败的情况，注意调节波特率，最高或最低都调价到 9600 试试。

6、按 STC-ISP 软件上面的“DOWNLOAD/下载”，然后再给单片机上电。(这点非常重要)

2.1.3 第一个程序的编译

2.1.3.1 编译软件的安装

KEIL 安装文件在：HOT-51\开发板配套资料\开发软件 目录中
解压 KeilC51v750 for 51
（注意：很多安装后，用不了。因为只安了 KEIL3，希望跟着下面的步骤进行）

解压缩以后安装，步骤如下：

1.记下安装序列号，进入 setup 目录点击 setup.exe 进行安装；

2.选择 Install Support....全新安装，以前没有安装过或者放弃以前的序列号安装;选择 Update Current Installation 升级安装，将可以保持原来的序列号，不必再次输入

3.选择 Full 安装，Next->Yes(接受版权信息)->选择安装目录->Next->输入序列号、姓名、公司等，除了序列号以外，都随意，可以如实输入你的姓名等。->next->....直到安装完成。

注意：

1.每次安装都必须进行这几步，每次都需要重新写入 AddOn 标识；

2.假如安装过程中存在病毒防火墙，可能会产生 xcopy 错误使安装失败，此时请先

关闭病毒防火墙，然后再安装;

3.安装前必须退出正在运行的 Keil 软件，否则也会产生 xcopy 错误使安装失败；

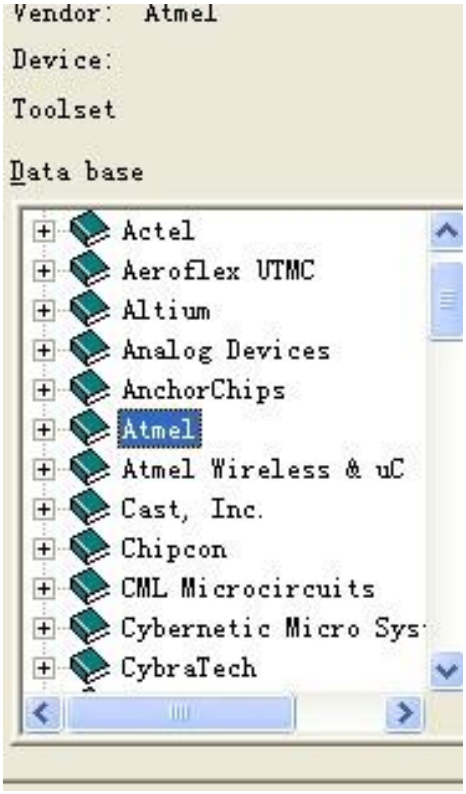
4.安装过程中可能会出现安装 Secrity Key 错误，点击确定即可。

序列号：SN = K1DZP-5IUSH-A01UE

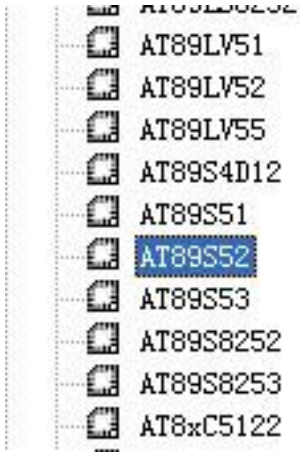
如果需要安装 UV3C51，您也可以点击 UV3C51.Exe 进行安装，安装到 UV2 的目录下就行了。

2.1.3.2 编译第一个文件

先新建一个文件夹，以方便工程的管理。点击 Project,然后选择 New Project,选择保存目录为你刚才新建的目录。

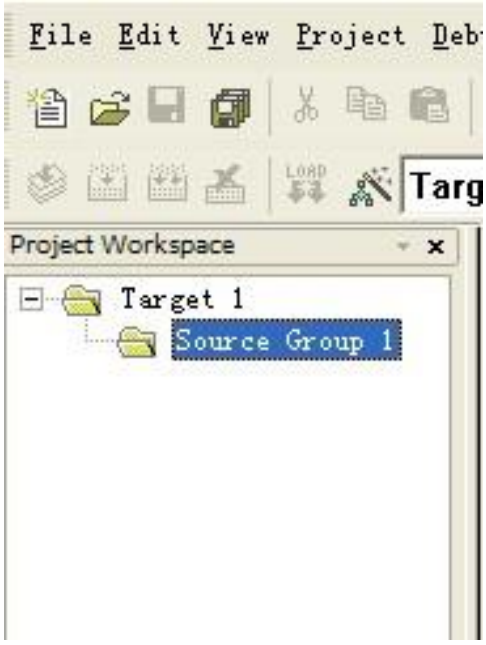


先双击选择 ATMEL 公司

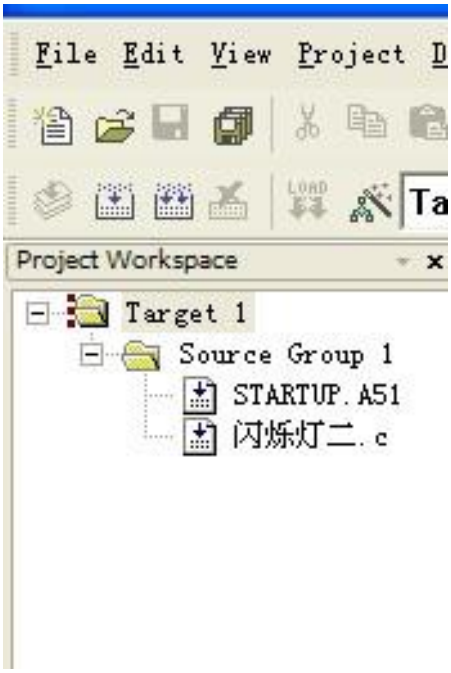


我们这里选择 AT89C52 或者 AT89S52

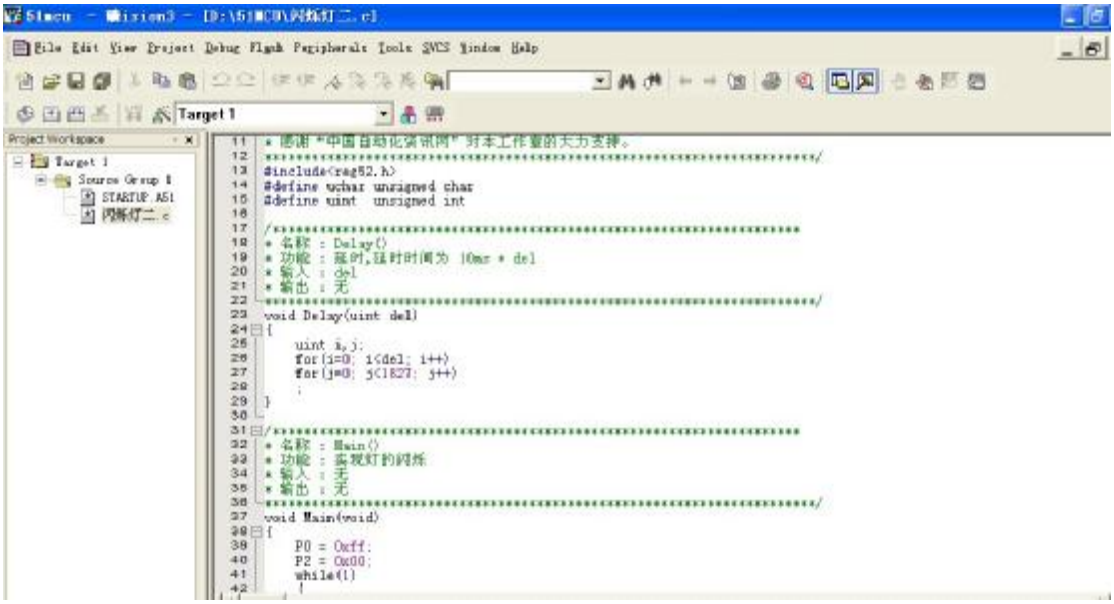
这里选“是”，这是用户上电初始化程序



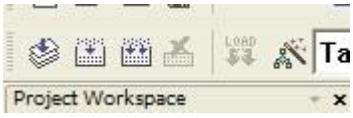
我们在“HOT-51\开发板配套资料\实例程序”下放了一个“闪烁灯二.C”的源文件。你们可以拷到你们新建的文件夹中。在 Source Group 1 上点击鼠标右键，点击 Add Files To Group "Source Group 1"，然后选择需要添加的 C 语言的源文件。



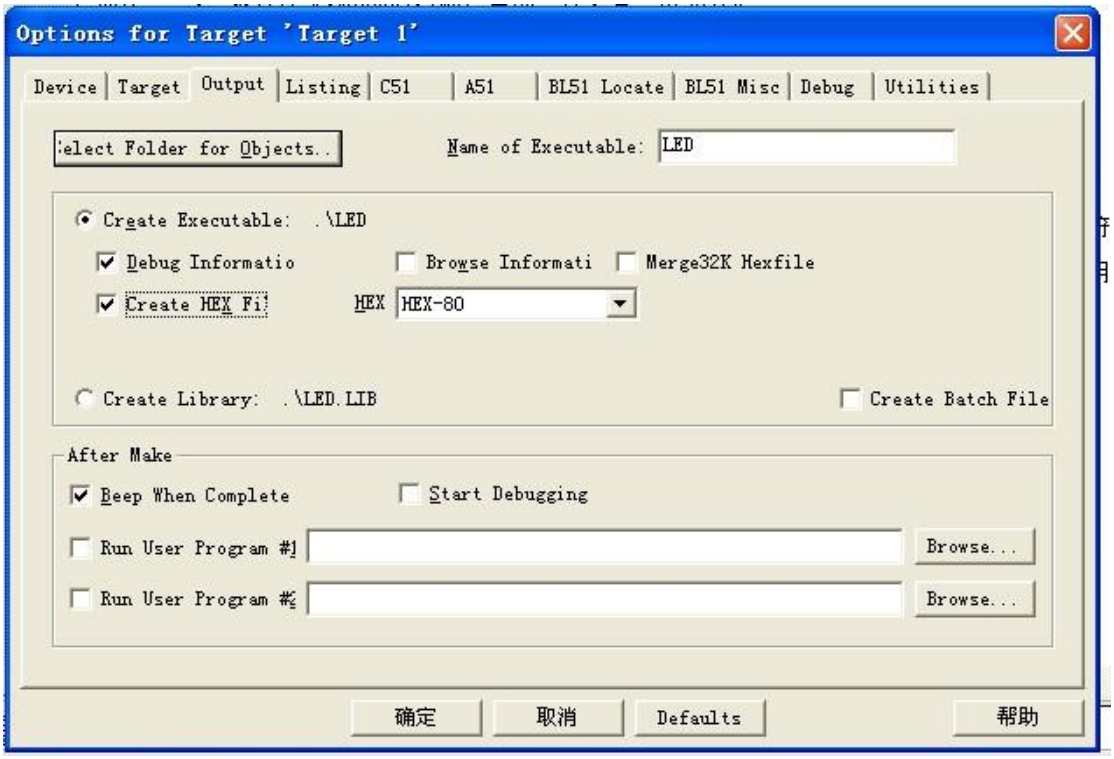
再双击"Source Group 1"。我们可以看到所添加的文件了。



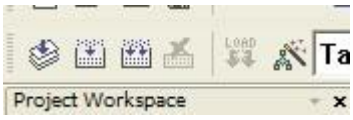
双击"闪烁灯二.C"。可以看到 C 语言的源文件了。



最后边的那个图标，进行设置。



点击“OUTPUT”。注意在“Create HEX "这个选项前打钩。以便生成 HEX 代码。



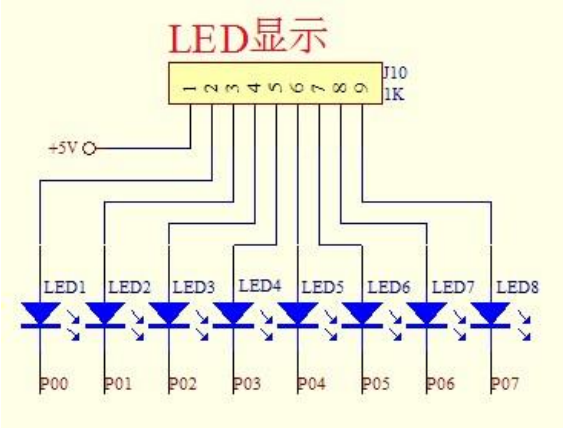
然后先后点击第一个和第三个，这样就可以生成 HEX 文件了。

```
Build target 'Target 1'
compiling 闪烁灯一.c...
linking...
Program Size: data=9.0 xdata=0 code=69
creating hex file from "LED"...
"LED" - 0 Error(s), 0 Warning(s).
```

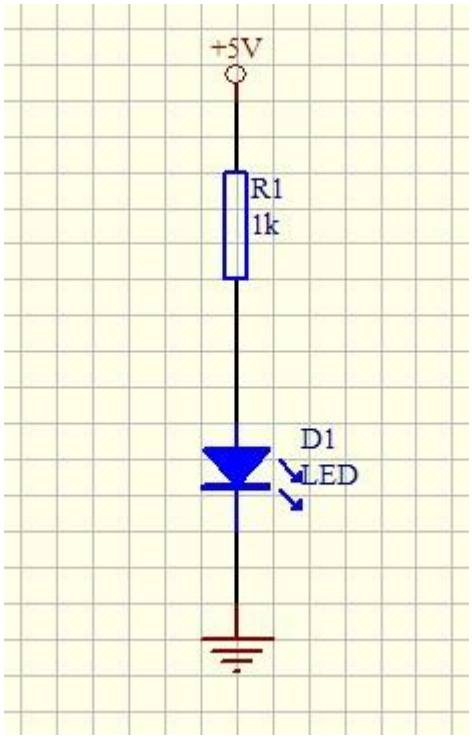
这是编译成功后所显示的。然后就可以在文件夹中找到后缀为 HEX 的文件，可以用 STC-ISP 进行烧写。

2.2 单片机配套试及电路验详解

2.2.1 LED 闪烁



J10 位 1K 的排阻。
当 P0 口某位为高电平时，LED 灭
当 P0 口某位为低电平时，LED 亮



我们单独来分析一个 LED。电路如上，假定 LED 发光时的电压为 1.5V，那么：

5 - 1.5 =470 * I

I = 0.0074A

为 3.5 毫安，我们设计该值也是为了提高 LED 寿命。

这里：我们简单解释一下单片机的 IO“输入与输出”

输入：单片机 IO 作为采集外部信号，在单片机内部处理，现在很多单片机都有高阻或者开漏状态用做输入。

输出：单片机为外界提供信号和电流。这里包括灌电流和拉电流。

灌电流：IO 输出“0”时允许灌入该 IO 口的电流。

拉电流：IO 输出“1”时允许输出该 IO 口的电流。

所以，不要以电流的方向来理解输入和输出，这个是初学者很容易理解错的。

作为单片机的指令的执行的时间是很短，数量为微秒级。因此，我们要求的闪烁时间间隔为 0.2 秒，相对于微秒来说，相差太大，所以我们在执行某一指令时，插入延时程序，来达到我们的要求。

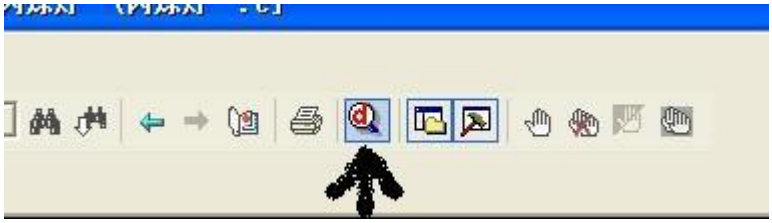
```
/*********************************************************************
* 名称 ：Delay()
* 功能 ： 延时,延时时间为 10ms * del。这是通过软件延时，有一定误差。
* 输入 ：del
* 输出 ： 无
*********************************************************************/
```

```
void Delay(uint del)
{
    uint i,j;
    for(i=0; i<del; i++)
    for(j=0; j<1827; j++)        //这个是通过软件仿真得出的数
    ;
}
```

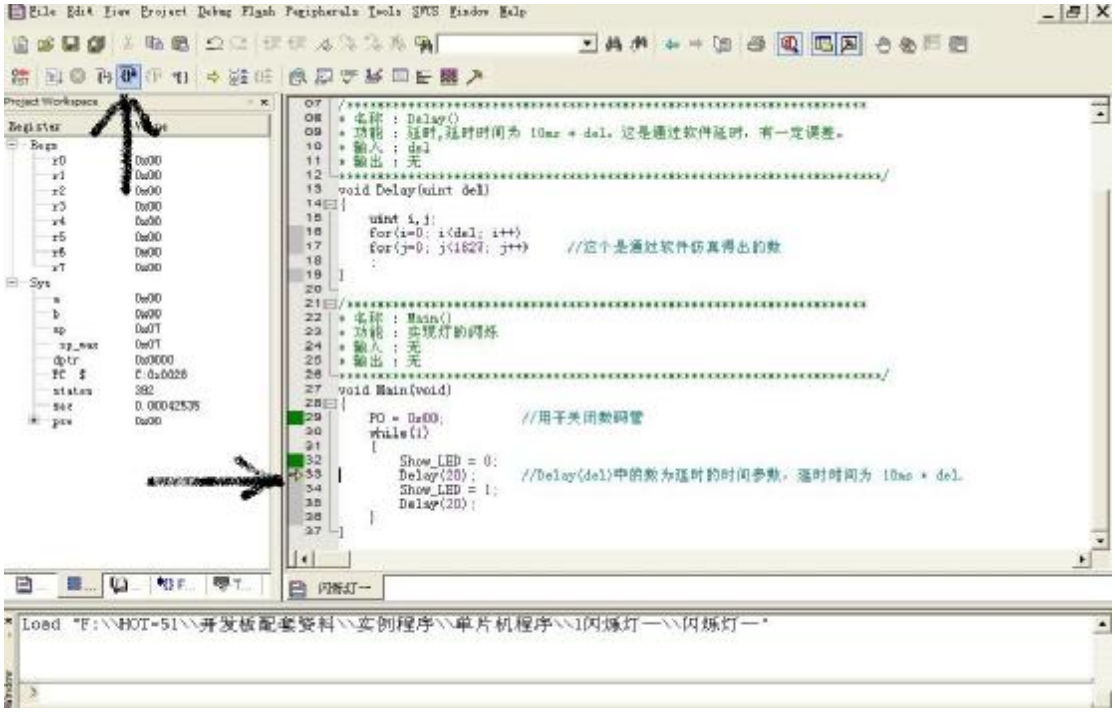
这个就是我们用 KEIL 软件仿真得出的 10 毫秒的延时程序。

这里我们先插入一段，讲下软件仿真，希望大家上网多搜搜这方面的资料。在“HOT-51\开发板配套资料\经典学习资料\单片机相关”下面有 KEIl 软件仿真的教程。

在 Target 中 Xtal(MHZ):设置为 11.0592(也可以根据你选择的晶振选择)，然后点“确定”。

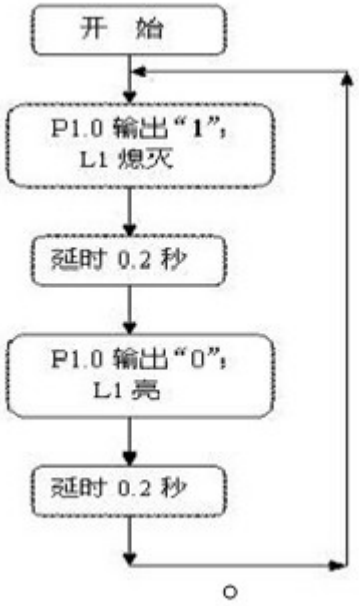


然后点击箭头的地方。进行仿真。



点击左上脚的按键进行单步仿真，观察左边工具栏中的 SEC 的时间在变化。当到达 Delay(20);这行时，记录 SEC 中的数据。然后再点击左上脚的单步仿真，这样，可以看到 SEC 中的数据增加了大概 0.2S。

回到 LED 的学习。下面是流水灯的流程图。

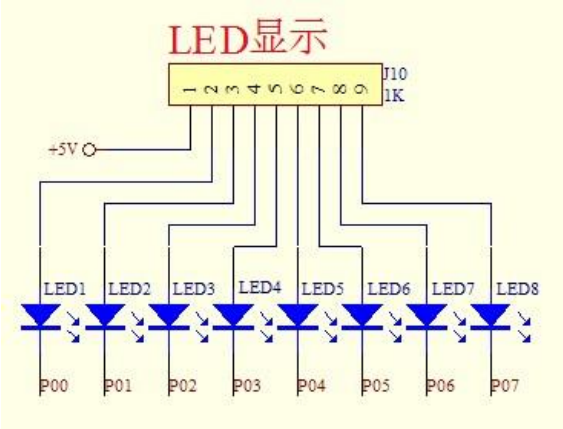


根据主函数，写出主函数如下：

配套试验程序一中部分程序。

```
void Main(void)
{
    while(1)
    {
        Show_LED = 0;
        Delay(20);   //Delay(del)中的数为延时的时间参数，延时时间为 10ms * del.
        Show_LED = 1;
        Delay(20);
    }
}
```

2.2.2 流水灯



八个发光二极管 L1-L8 分别接在单片机的 P0.0-P0.7 接口上，输出“0”时，发光二极管亮,按着 LED1→LED2→LED3→LED4→LED5→LED6→LED7→LED8 的顺序依次点亮。只需要将 P0 口的某位依次变为低电平就行了。我们这里用了头文件 intrins.H 中的_crol_(A,B),_cror(A,B)函数，功能为循环移动。

配套试验程序三部分程序。

```
/*************************************************************************
* 名称 : Main()
* 功能 : 实现灯的闪烁
* 输入 : 无
* 输出 : 无
* 说明 : 也可以使用例 1 的的方法来关闭数码管
*************************************************************************/
void Main(void)
{
```

```
    uchar i;
    P2 = 0x00;
    while(1)
    {
        P0 = 0x7f;                  //P0.7 亮亮
        for(i=0; i<7; i++)          //移位 7 次
        {
            P0 = _cror_(P0, 1);     //_crol_()在 intrins.h 中定义了，用于循环右
移
            Delay(15);
        }
        P0 = 0xfe;
        for(i=0; i<7; i++)
        {
            P0 = _crol_(P0, 1);      //用于循环左移
            Delay(15);
        }
    }
}
```

下面是简单的介绍下 intrins.h 这个 C51的内部函数：

crol 字符循环左移

cror 字符循环右移

irol 整数循环左移

iror 整数循环右移

lrol 长整数循环左移

lror 长整数循环右移

nop 空操作8051 NOP 指令

testbit 测试并清零位8051 JBC 指令

函数名： _crol_，_irol_，_lrol_

原 型： unsigned char _crol_(unsigned char val,unsigned char n);

unsigned int _irol_(unsigned int val,unsigned char n);
unsigned int _lrol_(unsigned int val,unsigned char n);

功 能：_crol_，_irol_，_lrol_以位形式将 val 左移 n 位，该函数与8051“RLA”

指令相关

函数名： _cror_，_iror_，_lror_

原 型： unsigned char _cror_(unsigned char val,unsigned char n);

unsigned int _iror_(unsigned int val,unsigned char n);
unsigned int _lror_(unsigned int val,unsigned char n);

功 能：_cror_，_iror_，_lror_以位形式将 val 右移 n 位，该函数与8051“RRA”

指令相关

比如：P0 = _cror_(P0, 1);就是字符右循环1位。

2.2.3 定时器

- 8051 单片机片内有二个十六位定时器 / 计数器：定时器 0(T0)和定时器 1(T1)。
- 2 个 16 位定时器实际上都是 16 位加 1 计数器。T0 由 2 个 8 位持殊功能寄存器 TH0 和 TL0 构成，T1 由 TH1 和 TL1 构成。
- 每个定时器都可由软件设置为定时工作方式或计数工作方式。
- 这些功能都由特殊功能寄存器 TMOD 设置和 TCON 所控制。
- 设置为定时工作方式时，定时器计数的脉冲是由 8051 片内振荡器输出经 12 分频后产生的。
- 每个机器周期使定时器(T0 或 T1)的数值加 1 直至计计数满产生溢出。

工作模式寄存器 TMOD(89H)

	定时器T1				定时器T0			
TMOD	D7	D6	D5	D4	D3	D2	D1	D0
(89H)	GATE	C/T	M1	M0	GATE	C/T	M1	M0

TMOD 的高 4 位用于 T1，低 4 使用于 T0，4 种符号的含义如下：
GATE：门控制位。GATE 和软件控制位 TR、外部引脚信号 INT 的状态,共同控制定时器 / 计数器的打开或关闭。
C / T：定时器 / 计数器选择位。C/T = 1，为计数器方式；C / T = 0，为定时器方式。
M1M0：工作方式选择位，定时器 / 计数器的 4 种工作方式由 M1M0 设定。

	工作方式	功能描述
0 0	工作方式 0	13 位计数器
0 1	工作方式 1	16 位计数器
1 0	工作方式 2	自动再装入 8 位计数器
1 1	工作方式 3	定时器 0：分成两个 8 位计数器，定时器 1：停止计数

控制寄存器 TCON(88H)

TCON	8FH	8EH	8DH	8CH	8BH	8AH	89H	88H
(88H)	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

TF1：定时器 1 溢出标志位。当字时器 1 计满溢出时，由硬件使 TF1 置“1”，并且申请中断。进入中断服务程序后，由硬件自动清“0”，在查询方式下用软件清“0”。
TR1：定时器 1 运行控制位。由软件清“0”关闭定时器 1。当 GATE=1，且 INT1 为高电平时，TR1 置“1”启动定时器 1；当 GATE=0，TR1 置“1”启动定时器 1。
TF0：定时器 0 溢出标志。其功能及操作情况同 TF1。
TR0：定时器 0 运行控制位。其功能及操作情况同 TR1。
IE1：外部中断 1 请求标志。
IT1：外部中断 1 触发方式选择位。

IE0：外部中断 0 请求标志。
IT0：外部中断 0 触发方式选择位。

定时器/计数器的初始化：

由于定时器/计数器的功能是由软件编程确定的，所以一般在使用定时/计数器前都要对其进行初始化，使其按设定的功能工作。初始货的步骤一般如下：
1、确定工作方式（即对 TMOD 赋值）；
2、预置定时或计数的初值（可直接将初值写入 TH0、TL0 或 TH1、TL1）；
3、根据需要开放定时器/计数器的中断（直接对 IE 位赋值）；
4、启动定时器/计数器（若已规定用软件启动，则可把 TR0 或 TR1 置“1”；若已规定由外中断引脚电平启动，则需给外引脚步加启动电平。当实现了启动要求后，定时器即按规定的工作方式和初值开始计数或定时）。

定时器初值的计算：

在定时器模式下：计数器由单片机主脉冲经 12 分频后计数。因此，定时器定时时间 T 的计算公式为：

$$TC = M - T/T_{计数}$$

这里，M 为模值，和定时器的工作方式有关；方式 0：M = 8192　方式 1：M = 65536;方式 2 和方式 3: M = 256;　$T_{计数}$ 是单片机时钟周期的 12 倍；　TC 为定时器的定时初值，T 为需要定时的时间。

下面，来分析下我们例程的部分程序：

```
/*********************************************************************
* 名称 : Time0_Init()
* 功能 : 定时器的初始化，11.0592MZ 晶振，50ms
* 输入 : 无
* 输出 : 无
*********************************************************************/
void Time0_Init()
{
    TMOD = 0x01;              //初始化定时器 0，工作在方式 1
    IE   = 0x82;              //开定时器中断，开总中断
    TH0  = 0x3c;              //这里是赋初值，初值的计算按上面的公司
    TL0  = 0xb0;              //也可以通过我们给的定时器计算软件计算
    TR0 = 1;                  //启动定时器
}

/*********************************************************************
* 名称 : Time0_Int()
* 功能 : 定时器中断，中断中实现 Count 加一
* 输入 : 无
* 输出 : 无
```

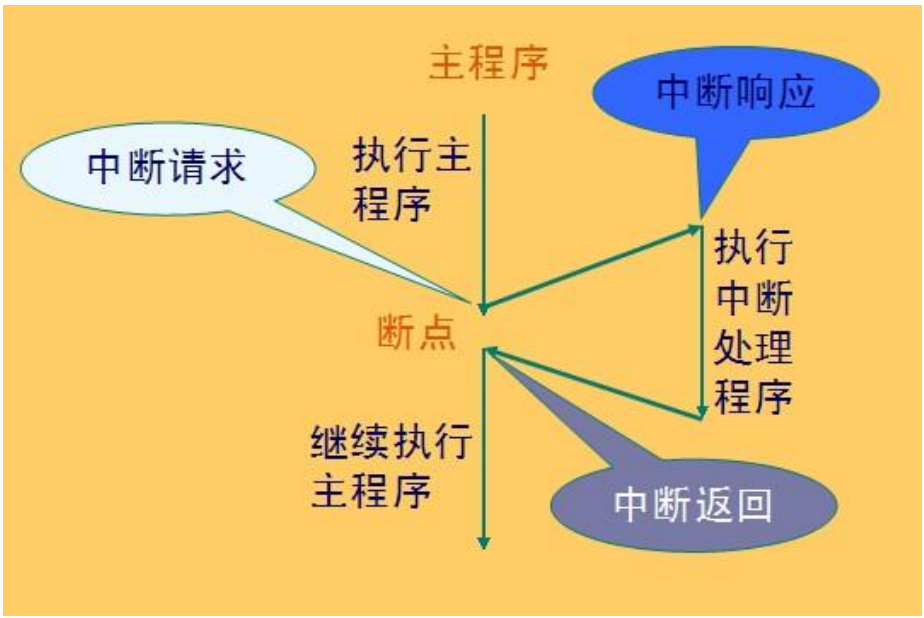
```
* 注意 ：定时器中不能处理进行太多的数据处理，否则会影响定时的准确性
*****************************************************************************/
void Time0_Int() interrupt 1
{
    TH0 = 0x3c;
    TL0 = 0xb0;
    Count++;              //长度加 1
}
```

另外的方式请大家自行学习。

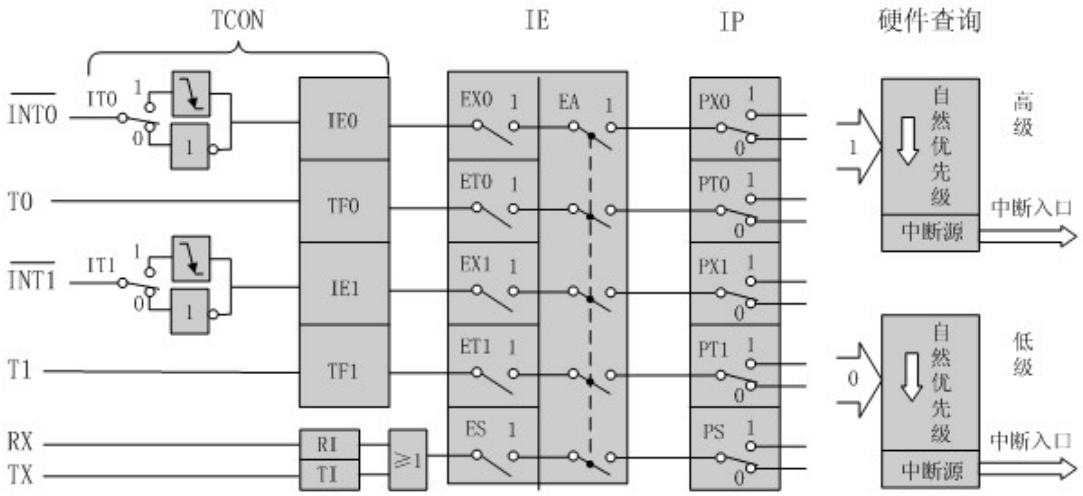
2.2.4 中断

中断的概念

- CPU 在处理某一事件 A 时，发生了另一事件 B 请求 CPU 迅速去处理（中断发生）；
- CPU 暂时中断当前的工作，转去处理事件 B（中断响应和中断服务）；
- 待 CPU 将事件 B 处理完毕后，再回到原来事件 A 被中断的地方继续处理事件 A（中断返回），这一过程称为中断 。



引起 CPU 中断的根源，称为中断源。中断源向 CPU 提出的中断请求。CPU 暂时中断原来的事务 A，转去处理事件 B。对事件 B 处理完毕后，再回到原来被中断的地方（即断点），称为中断返回。实现上述中断功能的部件称为中断系统（中断机构）。



TCON 的中断标志：

位	7	6	5	4	3	2	1	0	
字节地址：88H	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	TCON

IT0（TCON.0），外部中断 0 触发方式控制位。
当 IT0=0 时，为电平触发方式。
当 IT0=1 时，为边沿触发方式（下降沿有效）。
IE0（TCON.1），外部中断 0 中断请求标志位。
IT1（TCON.2），外部中断 1 触发方式控制位。
IE1（TCON.3），外部中断 1 中断请求标志位。
TF0（TCON.5），定时/计数器 T0 溢出中断请求标志位。
TF1（TCON.7），定时/计数器 T1 溢出中断请求标志位。、

SCON 中断标志：

位	7	6	5	4	3	2	1	0	
字节地址：98H							TI	RI	SCON

RI（SCON.0），串行口接收中断标志位。当允许串行口接收数据时，每接收完一个串行帧，由硬件置位 RI。同样，RI 必须由软件清除。

TI（SCON.1），串行口发送中断标志位。当 CPU 将一个发送数据写入串行口发送缓冲器时，就启动了发送过程。每发送完一个串行帧，由硬件置位 TI。CPU 响应中断时，不能自动清除 TI，TI 必须由软件清除。

中断允许控制：IE

位	7	6	5	4	3	2	1	0	
字节地址：A8H	EA			ES	ET1	EX1	ET0	EX0	IE

EX0(IE.0)，外部中断 0 允许位；
ET0(IE.1)，定时/计数器 T0 中断允许位；
EX1(IE.2)，外部中断 0 允许位；
ET1(IE.3)，定时/计数器 T1 中断允许位；
ES（IE.4），串行口中断允许位；
EA (IE.7)， CPU 中断允许（总允许）位。

中断优先级控制：IP

位	7	6	5	4	3	2	1	0	
字节地址：B8H			PT2	PS	PT1	PX1	PT0	PX0	IP

PX0（IP.0），外部中断 0 优先级设定位；
PT0（IP.1），定时/计数器 T0 优先级设定位；
PX1（IP.2），外部中断 0 优先级设定位；
PT1（IP.3），定时/计数器 T1 优先级设定位；
PS （IP.4），串行口优先级设定位；
PT2 (IP.5) ，定时/计数器 T2 优先级设定位。

而 80C52 单片机有四个中断优先级，即可实现四级中断服务嵌套。每个中断源的中断优先级由中断优先级寄存器 IP 和 IPH 中的相应位的状态来规定的 。

位	7	6	5	4	3	2	1	0	
字节地址：B7H			PT2	PS	PT1	PX1	PT0	PX0	IPH

PX0（IPH.0），外部中断 0 优先级设定位；
PT0（IPH.1），定时/计数器 T0 优先级设定位；
PX1（IPH.2），外部中断 0 优先级设定位；
PT1（IPH.3），定时/计数器 T1 优先级设定位；
PS （IPH.4），串行口优先级设定位；
PT2 (IPH.5) ，定时/计数器 T2 优先级设定位。

各中断源响应优先级及中断服务程序入口表

中断源	中断标志	中断服务程序入口	优先级顺序
外部中断 0（$\overline{INT0}$）	IE0	0003H	高
定时/计数器 0（T0）	TF0	000BH	↓
外部中断 1（$\overline{INT1}$）	IE1	0013H	↓
定时/计数器 1（T1）	TF1	001BH	↓
串行口	RI 或 TI	0023H	低

80C51 单片机的中断优先级有三条原则：

- CPU 同时接收到几个中断时，首先响应优先级别最高的中断请求。
- 正在进行的中断过程不能被新的同级或低优先级的中断请求所中断。
- 正在进行的低优先级中断服务，能被高优先级中断请求所中断。

中断系统初始化：

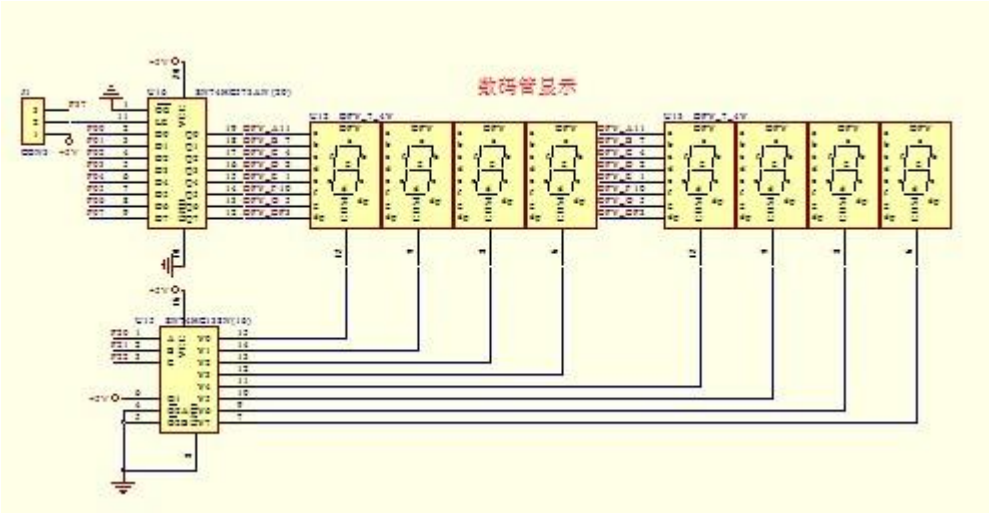
- 开相应中断源的中断；(IE)
- 设定中断优先级；(IP)
- 若为外部中断，设定外部中断的触发方式。

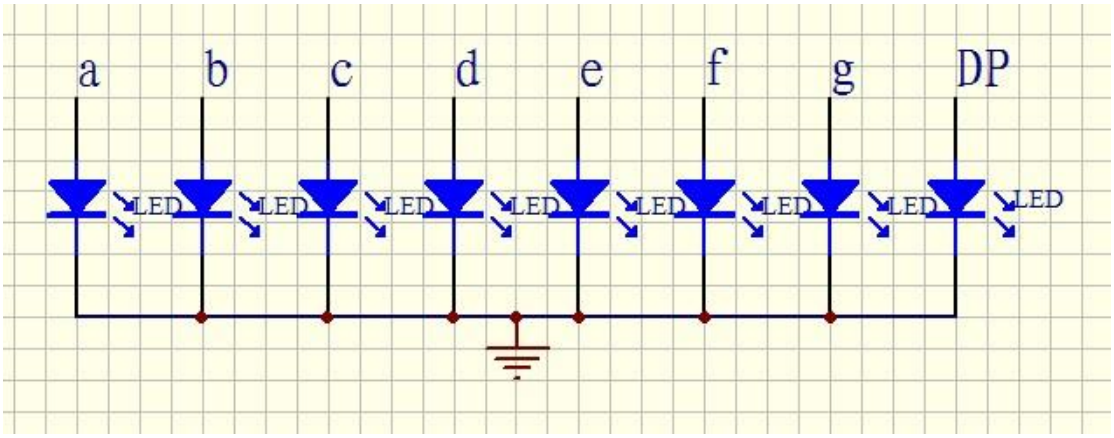
这里，我们通过我们的实例程序来学习：

```
/*********************************************************************
* 名称 : Outside_Init()
* 功能 : 外部中断 0 的初始化
* 输入 : 无
* 输出 : 无
```

```
*************************************************************************/
void Outside_Init(void)
{
    EX0 = 1;   //开外部中断 0
    IT0 = 1;   //负边沿触发
    EA = 1;    //开总中断
}

/*************************************************************************
* 名称 : Outside_Int()
* 功能 : 外部中断 0 的中断处理
* 输入 : 无
* 输出 : 无
*************************************************************************/
void Outside_Int(void) interrupt 0
{
    EX0 = 0;          //关中断
    Delay(3);         //延时 30ms，去掉这行会出现按一下中断几次的情况
    if(KEY == 0)      //对按键进行抗干扰处理
    {
        Count++;
    }
    Delay(30);        //延时 300 毫秒进行下次检测
    EX0 = 1;          //开中断
}
```

2.2.5 数码管静态显示





数码管的基本构造可以看成如上图。他们的一端全连在一起。如果 LED 的阴极相连，那么该数码管被称为共阴数码管；如果 LED 的阳极相连，那么该数码管被称为共阳数码管。上面每个 LED 相当于数码管的段。

数字对应数码管显示控制转换字节（共阴编码）

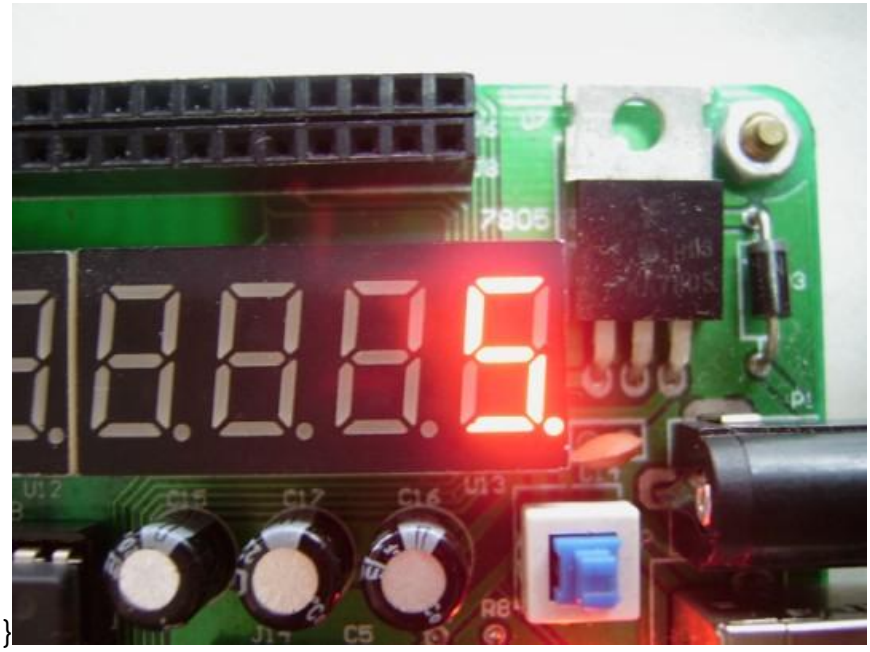
显示	－－	Dp	G	F	E,	D	C	B	A－－编码
0	－－	0	0	1	1,	1	1	1	1－－0x3F;
1	－－	0	0	0	0,	0	1	1	0－－0x06;
2	－－	0	1	0	1,	1	0	1	1－－0x5B;
3	－－	0	1	0	0,	1	1	1	1－－0x4F;
4	－－	0	1	1	0,	0	1	1	0－－0x66;
5	－－	0	1	1	0,	1	1	0	1－－0x6D;
6	－－	0	1	1	1,	1	1	0	1－－0x7D;
7	－－	0	0	0	0,	0	1	1	1－－0x07;
8	－－	0	1	1	1,	1	1	1	1－－0x7F;
9	－－	0	1	1	0,	1	1	1	1－－0x6F;
字符									
A	－－	0	1	1	1,	0	1	1	1－－0x77;
b	－－	0	1	1	1,	1	1	0	0－－0x7C;
C	－－	0	0	1	1,	1	0	0	1－－0x39;
d	－－	0	1	0	1,	1	1	1	0－－0x5E;
E	－－	0	1	1	1,	1	0	0	1－－0x79;
F	－－	0	1	1	1,	0	0	0	1－－0x71;

共阳为编码取反即可。

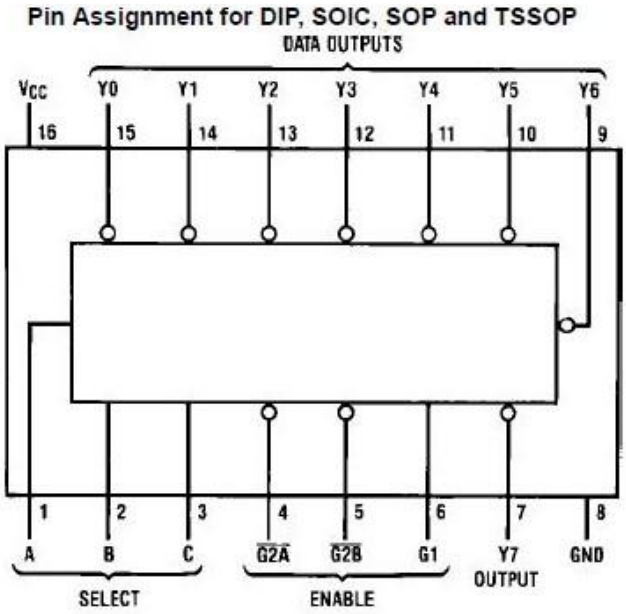
以下是试验程序五的部分程序：

```
/*下一行你可以试着把 code 去掉看看试验结果。 去掉后 table[] 会被存到 RAM 中，因为单片机的 RAM 比 ROM 小的多*/
/*所以，对于不会改变的值应该用 code 或者 #define 去定义，让这些固定值存到 ROM 中去*/
uchar code table[16] =
{0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f,0x77,0x7c,0x39,0x5e,0x79,0x71};
```

```
/*下一行的数组可以显示数值外，还可以显示数码管的点*/
uchar code table_d[16] =
{0xbf,0x86,0xdb,0xcf,0xe6,0xed,0xfd,0x87,0xff,0xef,0xf7,0xfc,0xb9,0xde,0xf9,0xf1
};
/*******************************************************************
* 名称 : Main()
* 功能 : 主函数
* 输入 : 无
* 输出 : 无
*******************************************************************/
void main()
{
    uchar i = 0;
    while(1)
    {
        //P0 = table[i % 16];        //在这里取 i 的个位数，不带点显示
        P0 = table_d[i % 16];        //带点显示
        if(KEY == 0)
        {
            Delay(1);                //软件消抖，试验为20ms
            if(KEY == 0)
            {
                i++;
            }
            Delay(50);               //延时0.5秒后进行下一次的按键检测
        }
    }
}
```



2.2.6 三八译码器（74HC138)



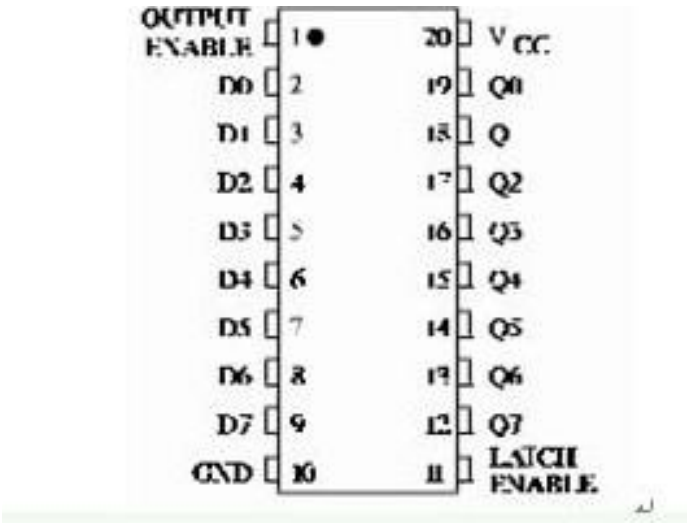
Inputs					Outputs							
Enable		Select										
G1	$\overline{G2}$ (Note 1)	C	B	A	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7
X	H	X	X	X	H	H	H	H	H	H	H	H
L	X	X	X	X	H	H	H	H	H	H	H	H
H	L	L	L	L	L	H	H	H	H	H	H	H
H	L	L	L	H	H	L	H	H	H	H	H	H
H	L	L	H	L	H	H	L	H	H	H	H	H
H	L	L	H	H	H	H	H	L	H	H	H	H
H	L	H	L	L	H	H	H	H	L	H	H	H
H	L	H	L	H	H	H	H	H	H	L	H	H
H	L	H	H	L	H	H	H	H	H	H	L	H
H	L	H	H	H	H	H	H	H	H	H	H	L

由上表可见 74HC138 译码器输出低电平有效。为增加译码器功能，除三个输入端 C、 B、 A 外，还设置了 G1、/G2A，/G2B，使译码器具有较强的抗干扰能力且便于扩展。

你注意下 Y0 到 Y7 的 0 你会发现的,其实他可以用来扩充控制端口,可以实现三个端口入,八个端口出的控制,相当于增加了 MCU 的管教脚。

当 G1 = 0 时，不管其他输入如何，电路输出均为“1”，即无译码输出；只有当 G1 = 1，且/G2A = /G2B = 0 时，译码器才处于允许工作状态，输出与输入二进制码相对应，如 CBA=110 时，Y6 输出低电平。

2.2.7 锁存器（74HC573)



74HC573功能表

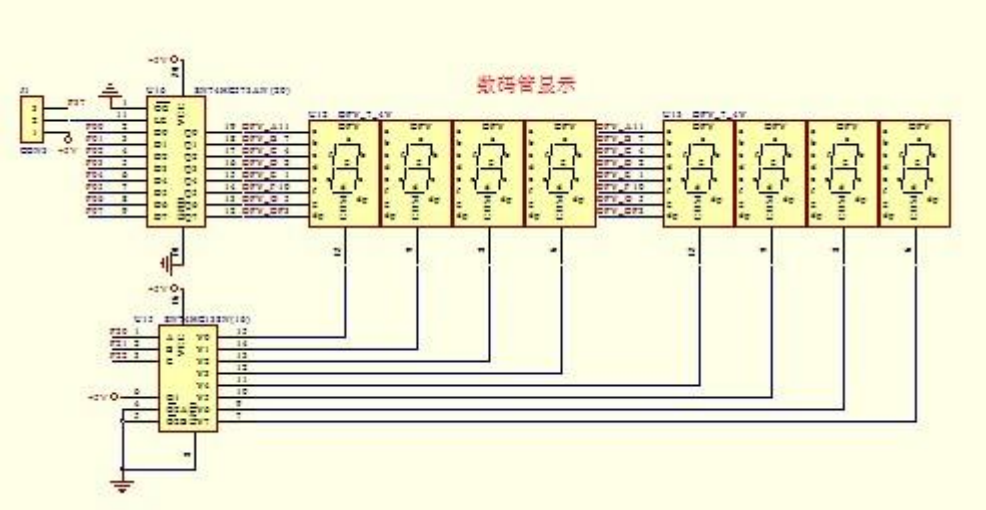
输 入			输出
输出使能	锁存性能	D	Q
L	H	H	H
L	H	L	L
L	L	X	不变
H	X	X	Z

锁存器就是把当前的状态锁存起来，使 CPU 送出的数据在接口电路的输出端保持一段时间锁存后状态不再发生变化，直到解除锁定。

当输出使能为 低 电平时，锁存器锁存功能使能。我们这里，锁存引脚接的 P3.7 口。

如果要使用锁存功能，请把 J1 的跳线跳到下部。

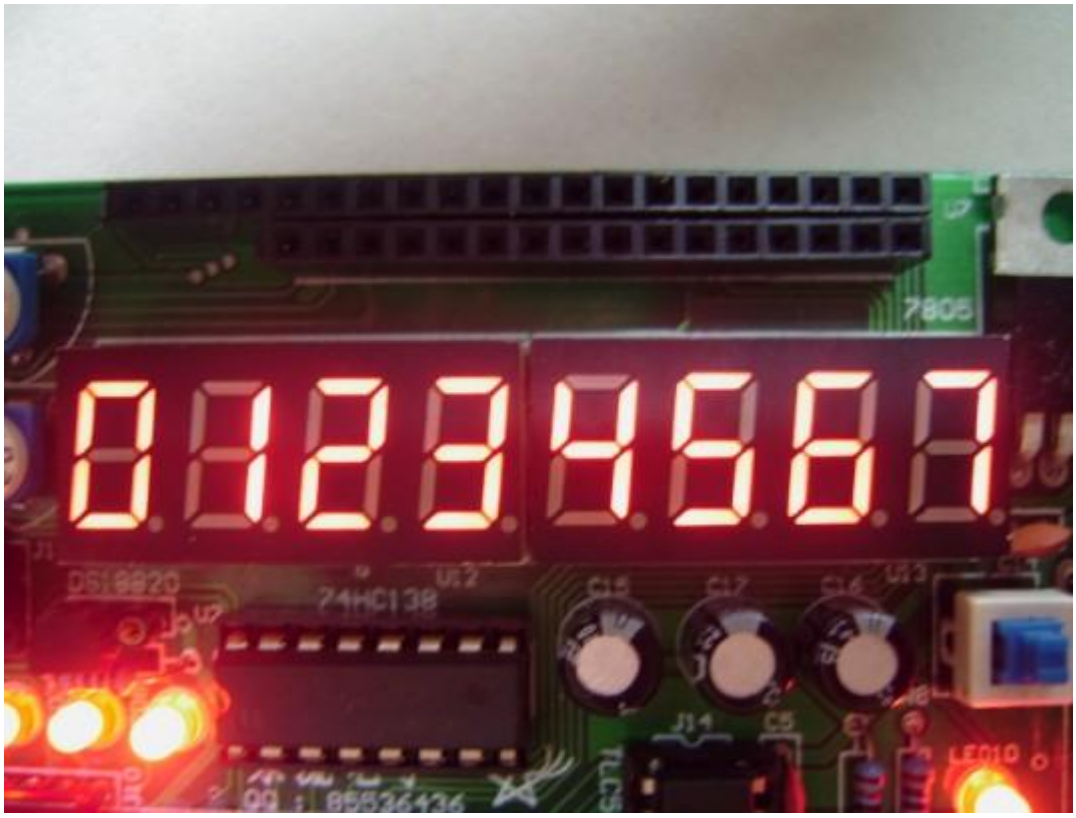
2.2.8 数码管动态显示



我们用的是共阴数码管。数码管的位选是通过74HC138连接的。数码管的段通过74HC573驱动。74HC138和74HC573的使用可以看前面两节。

```
uchar code table[10] = {0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f};
uchar code LED_W[8] = {0,1,2,3,4,5,6,7};     //这里是需要往三八译码器送的数据

/*************************************************************************
* 名称 : Main()
* 功能 : 数码管的显示
* 输入 : 无
* 输出 : 无
*************************************************************************/
void Main(void)
{
    uchar i = 0,j = 0;
    while(1)
    {
        P0 = table[i];                  // 数码管段值
        P2 = LED_W[j];          //点亮某一位数码管
        Delay(2);
        j++;
        i++;
        if(j == 8)                      //每个数码管都点亮了一次。
        {
            j = 0;
```

```
            i = 0;
        }
    }
}
```



数码管移动显示

```
/*****************************************************************************
* 名称 : Time0_Int()
* 功能 : 定时器中断，中断中实现 Count 加一
* 输入 : 无
* 输出 : 无
*****************************************************************************/
void Time0_Int() interrupt 1
{
    TH0 = 0x4c;
    TL0 = 0x00;
    Time_temp++;
    if(Time_temp == 4)  //定时器溢出时间 50 毫秒，隔 0.2 秒，数码管的数据改变一次
    {
        LED_temp++;
        Time_temp = 0;
    }
}
```

```
/*********************************************************************
* 名称 : Main()
* 功能 : 数码管的显示
* 输入 : 无
* 输出 : 无
*********************************************************************/
void Main(void)
{
    uchar i = 0,j = 0;
    Time0_Init();
    while(1)
    {
        P0 = table[(i + LED_temp) % 8];       //数码管段值
        P2 = LED_W[j];                          //点亮某一位数码管
        Delay(2);
        j++;
        i++;
        if(j == 8)                  //每个数码管都点亮了一次。
        {
            j = 0;
            i = 0;
        }
    }
}
```

也许你对这个程序理解起来也许有点困难，这里，我们简单的理清一下思路：

我们其实改动的就一句话： P0 = table[(i + LED_temp) % 8];

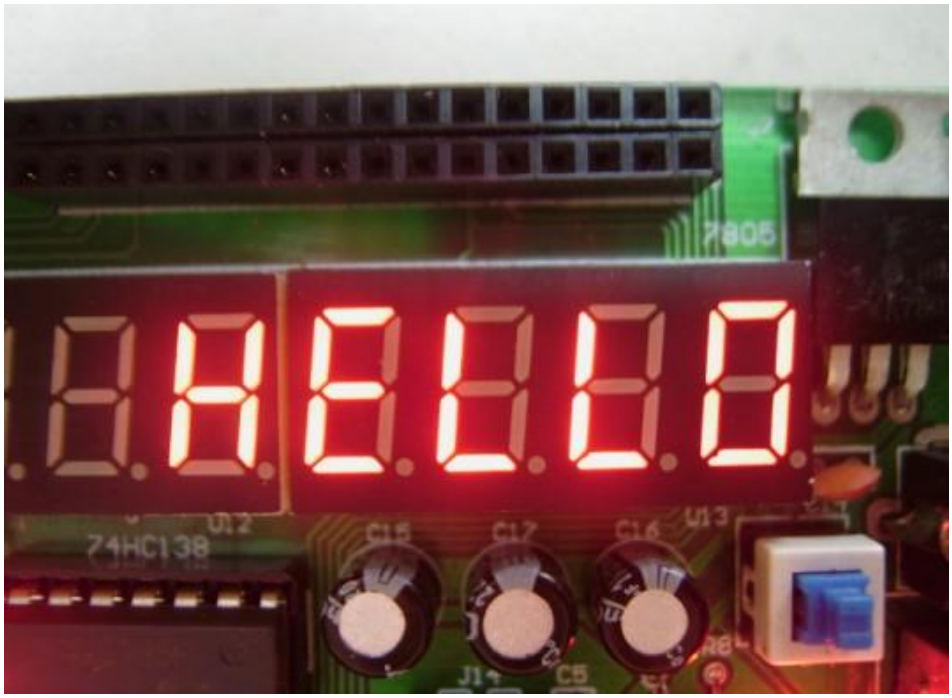
我们在定时器里中断中，让 LED_temp 这个变量每 0.2 秒改变一次。比如 LED_temp 只为 5（假设 i 和 j 这时为 0）：P0 值为 5，P2 值为 0，也就是在数码管最左端显示的值为 5；当 j++,i++后，P0 值为 6，P2 值为 1，最就是在左二的数码管显示为 6，依次类推。我们这里用了“%8”，这个是对 8 取余，也就是 table 中的数最大为 7。

显示 HELLO，PASS 等字符

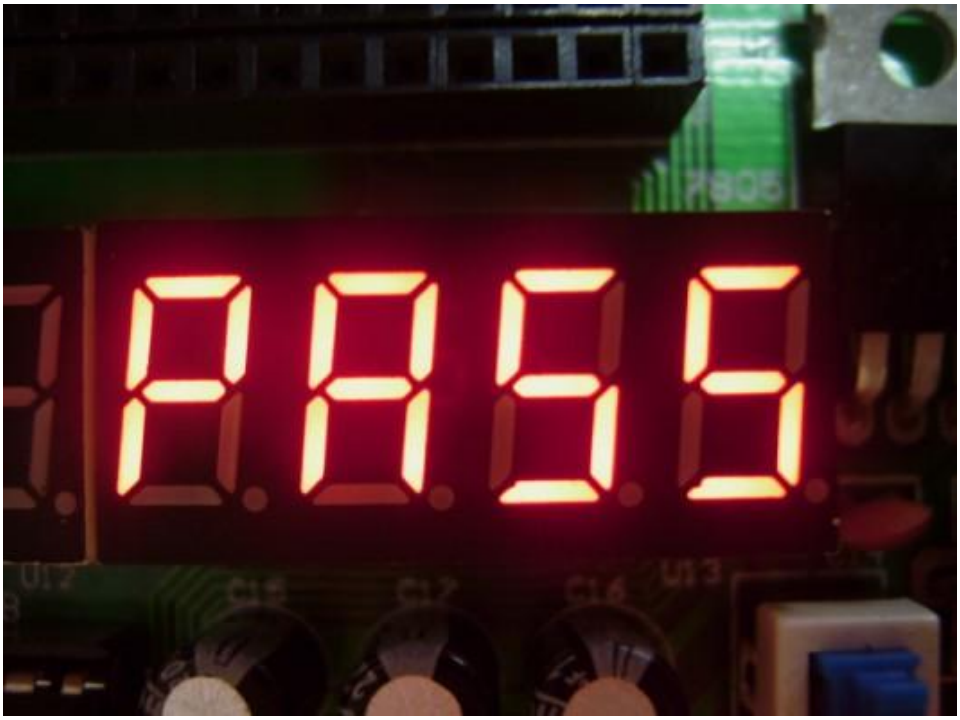
数码管不仅可以显示数字，而且可以显示简单字符。

```
uchar code table[8] = {0x00,0x00,0x00,0x76,0x79,0x06,0x06,0x3f};
```

这段代码就是显示 HELLO 的数码管编码。

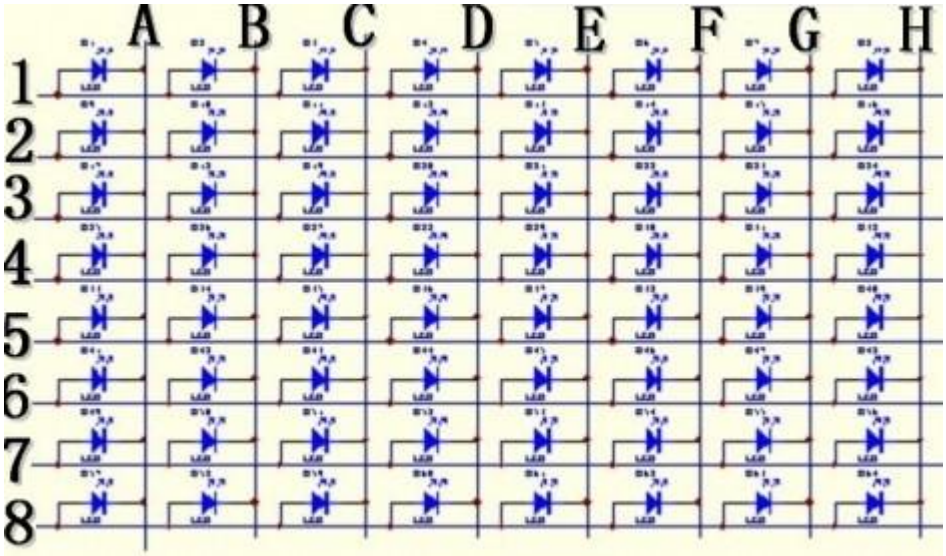


uchar code table[8] = {0x00,0x00,0x00,0x00,0x73,0x77,0x6d,0x6d};
这段代码就是显示 PASS 的数码管编码。



其他部分和数码管显示都差不多。

2.2.9 8 * 8 点阵



8×8点阵LED等效电路

8 X 8 点阵 LED 工作原理说明 :8X8点阵共需要64个发光二极管组成，且每个发光二极管是放置在行线和列线的交叉点上，当对应的某一列置1电平，某一行置0电平，则相应的二极管就亮；因此要实现一根柱形的亮法，如图49所示，对应的一列为一根竖柱，或者对应的一行为一根横柱，因此实现柱的亮的方法如下所述：

一根竖柱：对应的列置1，而行则采用扫描的方法来实现。

一根横柱：对应的行置0，而列则采用扫描的方法来实现。

点阵 LED 扫描法介绍

点阵 LED 一般采用扫描式显示，实际运用分为三种方式：

（1）点扫描；

（2）行扫描；

（3）列扫描。

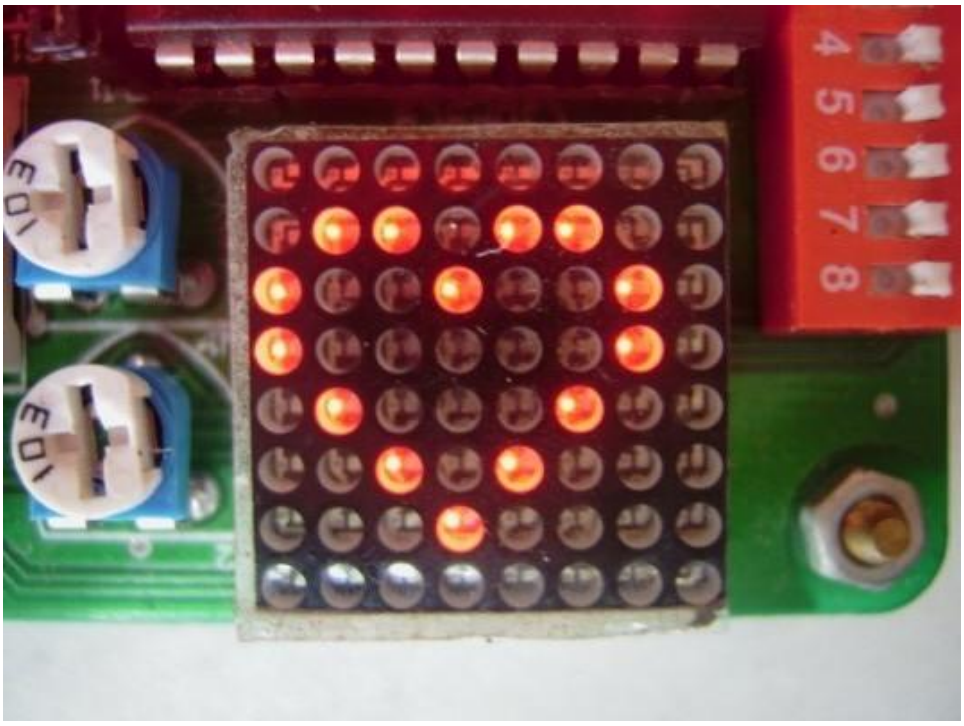
若使用第一种方式，其扫描频率必须大于16×64=1024Hz，周期小于1ms即

可。若使用第二和第三种方式，则频率必须大于16×8=128Hz，周期小于7.8ms即可符合视觉暂留要求。此外一次驱动一列或一行（8颗 LED）时需外加驱动电路提高电流，否则 LED 亮度会不足。

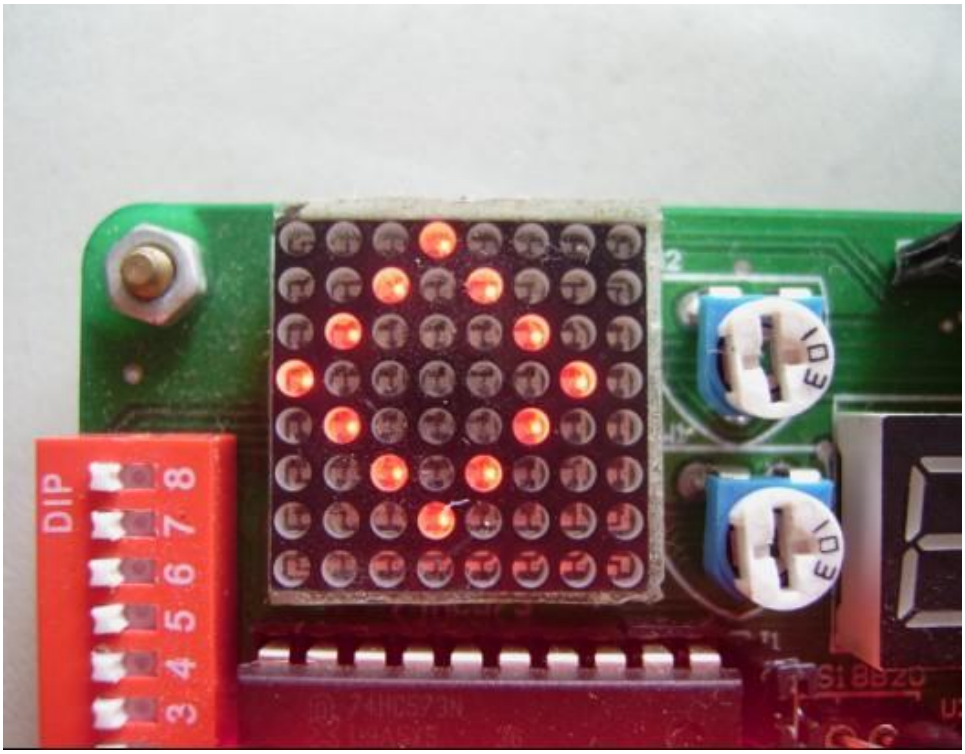
```
/*********************************************************************
* 文件名   :  定时器的使用.c
* 描述     :
* 创建人   :  东流，2009年4月8日
* 版本号   :  2.0
*********************************************************************/
#include <reg52.H>
#include<intrins.h>
```

```
//心形显示
unsigned char code tabP0[]={0x30,0x48,0x44,0x22,0x44,0x48,0x30,0x00};
unsigned char code tabP2[]={0x7F,0xBF,0xDF,0xEF,0xF7,0xFB,0xFD,0xFE};

//圆形显示
//unsigned char code tabP0[]={0x00,0x3c,0x42,0x42,0x42,0x42,0x3c,0x00};
//unsigned char code tabP2[]={0x7F,0xBF,0xDF,0xEF,0xF7,0xFB,0xFD,0xFE};

//菱形显示
//unsigned char code tabP0[]={0x00,0x08,0x14,0x22,0x41,0x22,0x14,0x08};
//unsigned char code tabP2[]={0x7F,0xBF,0xDF,0xEF,0xF7,0xFB,0xFD,0xFE};

//叉形显示
//unsigned char code tabP0[]={0x00,0x41,0x22,0x14,0x08,0x14,0x22,0x41};
//unsigned char code tabP2[]={0x7F,0xBF,0xDF,0xEF,0xF7,0xFB,0xFD,0xFE};

//中字显示
//unsigned char code tabP0[]={0x00,0x1e,0x12,0x12,0x7f,0x12,0x12,0x1e};
//unsigned char code tabP2[]={0x7F,0xBF,0xDF,0xEF,0xF7,0xFB,0xFD,0xFE};

/*********************************************************************
* 名称 : Main()
* 功能 : 主函数
* 输入 : 无
* 输出 : 无
*********************************************************************/
void main()
{
int i;
while(1)
{
    for(i=0;i<8;i++)
    {
        P2=tabP2[i];
        _nop_();
        _nop_();
        _nop_();
        P0=tabP0[i];
        _nop_();
        _nop_();
        _nop_();
    }
}
}
```

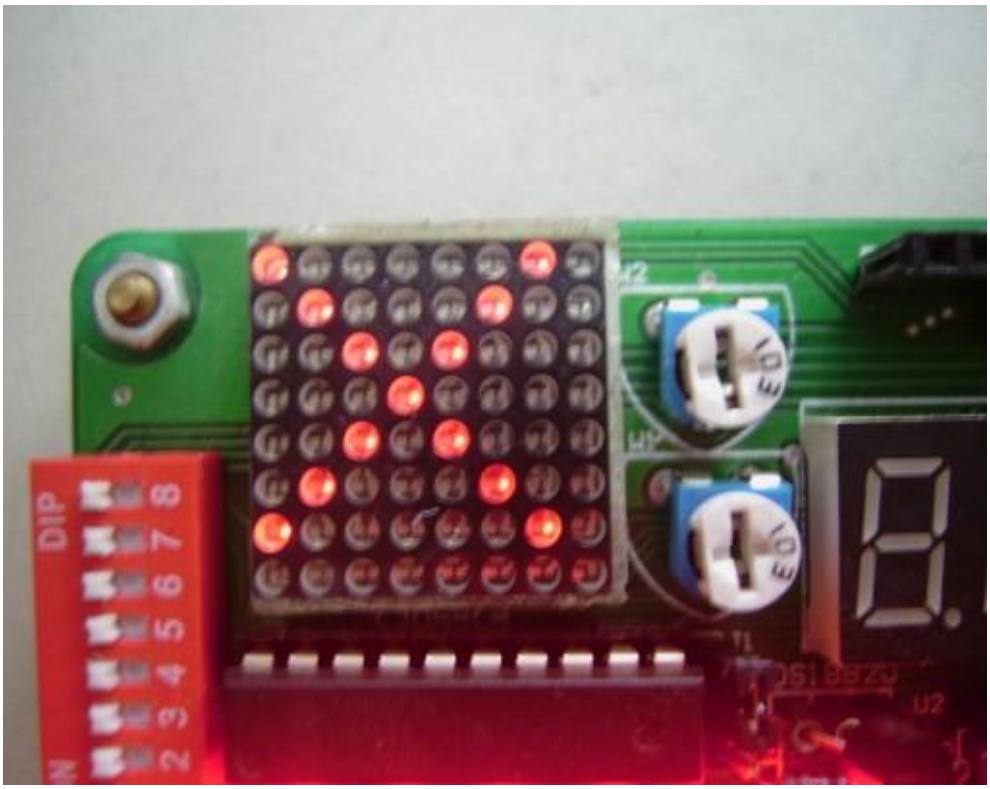
以上的程序我们是通过点扫描编程的。以上编码的点，我是通过最笨的方法一点一点算出的。



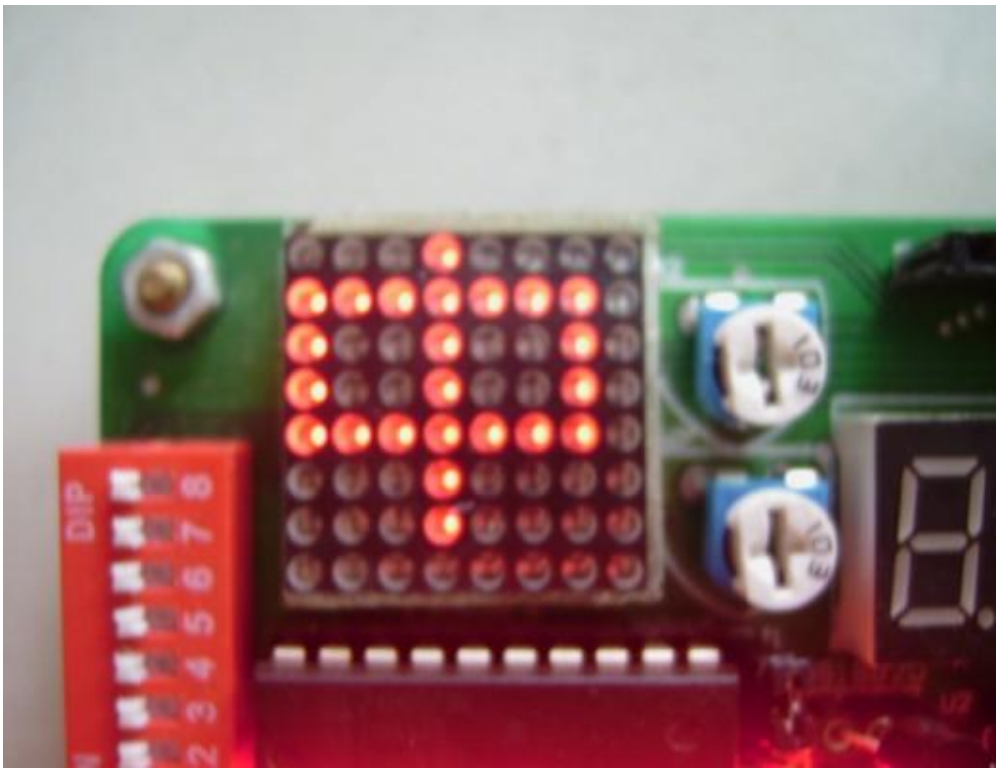
显示心型（但是是倒的，希望读者自行理解程序，然后改为正）



显示菱型



显示叉



显示汉字“中”

2.2.10 1602 显示

◆主要技术参数:

显示容量:	16X2 个字符
芯片工作电压:	4.5~5.5V
工作电流:	2.0mA(5.0V)
模块最佳工作电压:	5.0V
字符尺寸:	2.95X4.35(WXH)mm

◆接口信号说明:

编号	符号	引脚说明	编号	符号	引脚说明
1	VSS	电源地	9	D2	Data I/O
2	VDD	电源正极	10	D3	Data I/O
3	VL	液晶显示偏压信号	11	D4	Data I/O
4	RS	数据/命令选择端（H/L）	12	D5	Data I/O
5	R/W	读/写选择端（H/L）	13	D6	Data I/O
6	E	使能信号	14	D7	Data I/O
7	D0	Data I/O	15	BLA	背光源正极
8	D1	Data I/O	16	BLK	背光源负极

1602 的详细资料在我们的芯片手册中介绍已经很详细了。我们这里就不讨论这方面的内容了。
上面函数实现的功能就是让最高位的数据和最低位的数据交换。子程序请自行理解。

```
/*************************************************************************
* 名称 : L1602_char(uchar hang,uchar lie,char sign)
* 功能 : 改变液晶中某位的值，如果要让第一行，第五个字符显示"b" ，调用该函数如下
          L1602_char(1,5,'b')
* 输入 : 行，列，需要输入 1602 的数据
* 输出 : 无
*************************************************************************/
void L1602_char(uchar hang,uchar lie,char sign)
{
    uchar a;
    if(hang == 1) a = 0x80;
    if(hang == 2) a = 0xc0;
    a = a + lie - 1;
    enable(a);
    write(sign);
}

/*************************************************************************
```

```
* 名称 ：L1602_string(uchar hang,uchar lie,uchar *p)
* 功能 ：改变液晶中某位的值，如果要让第一行，第五个字符开始显示"ab cd ef" ，调用该函数如下
          L1602_string(1,5,"ab cd ef;")
* 输入 ：行，列，需要输入 1602 的数据
* 输出 ：无
*********************************************************************************/
void L1602_string(uchar hang,uchar lie,uchar *p)
{
    uchar a;
    if(hang == 1) a = 0x80;
    if(hang == 2) a = 0xc0;
    a = a + lie - 1;
    enable(a);
    while(1)
    {
        if(*p == '\0') break;
        write(*p);
        p++;
    }
}
```

L1602_char 和 L1602_string 两个函数是我们为液晶显示写的两个小函数。使用说明在功能一行已经有说明。

2.2.11 12864 显示

我们这里主要来讲下 ST920 芯片的 12864 液晶，它是带字库且可以串行和并行编程。

如果你自己有液晶，请注意液晶的电压。常用的屏有 3.3V 和 5V 两种，千万要弄清楚再插到开发板上，以免损坏你的液晶。

一、液晶显示模块概述

12864A-1 汉字图形点阵液晶显示模块，可显示汉字及图形，内置 8192 个中文汉字（16X16 点阵）、128 个字符（8X16 点阵）及 64X256 点阵显示 RAM（GDRAM）。

主要技术参数和显示特性:

电源：VDD 3.3V~+5V(内置升压电路，无需负压)；

显示内容：128 列× 64 行

显示颜色：黄绿

显示角度：6：00 钟直视

LCD 类型：STN

与 MCU 接口：8 位或 4 位并行/3 位串行

配置 LED 背光

多种软件功能：光标显示、画面移位、自定义字符、睡眠模式等。

二、模块引脚说明

128X64 引脚说明

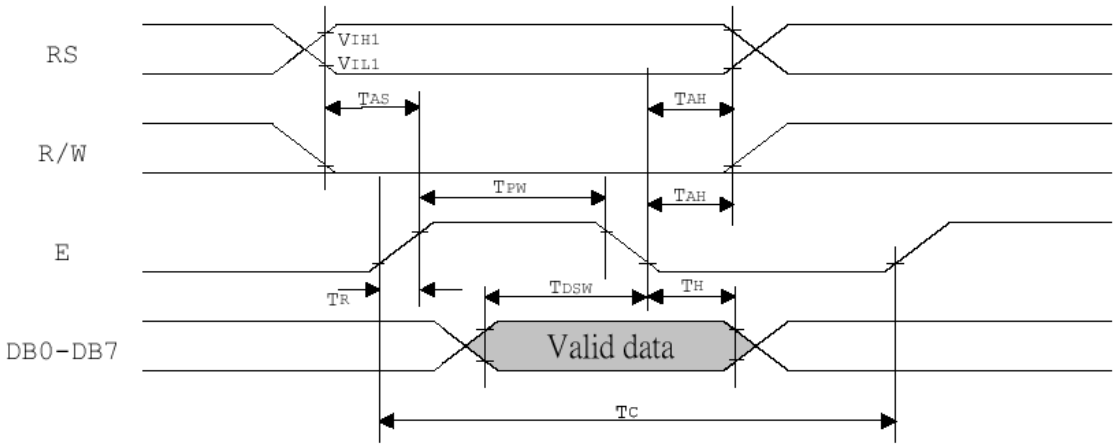
引脚号	引脚名称	方向	功能说明
1	VSS	-	模块的电源地
2	VDD	-	模块的电源正端
3	V0	-	LCD 驱动电压输入端
4	RS(CS)	H/L	并行的指令/数据选择信号；串行的片选信号
5	R/W(SID)	H/L	并行的读写选择信号；串行的数据口
6	E(CLK)	H/L	并行的使能信号；串行的同步时钟
7	DB0	H/L	数据 0
8	DB1	H/L	数据 1
9	DB2	H/L	数据 2
10	DB3	H/L	数据 3
11	DB4	H/L	数据 4
12	DB5	H/L	数据 5
13	DB6	H/L	数据 6
14	DB7	H/L	数据 7
15	PSB	H/L	并/串行接口选择：H-并行；L-串行
16	NC		空脚
17	/RET	H/L	复位 低电平有效

18	NC		空脚
19	LED_A	-	背光源正极（LED+5V）
20	LED_K	-	背光源负极（LED-0V）

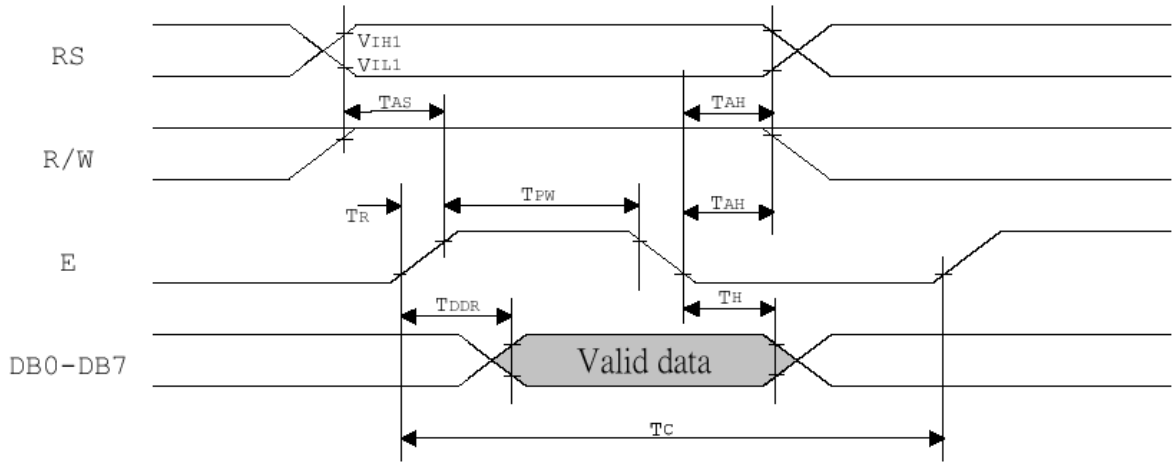
逻辑工作电压(VDD)：4.5～5.5V
电源地(GND)：0V
工作温度(Ta)：0～60℃(常温) / -20～75℃（宽温）

三、接口时序
模块有并行和串行两种连接方法（时序如下）：

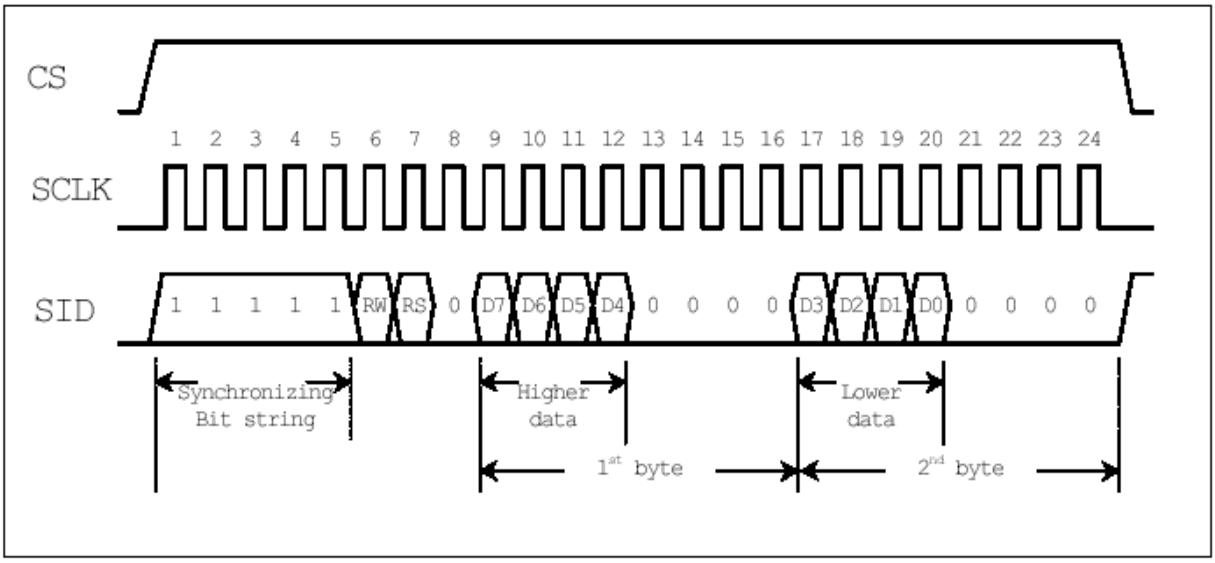
8 位并行连接时序图
MPU 写资料到模块



MPU 从模块读出资料



2、串行连接时序图



串行数据传送共分三个字节完成：

第一字节：串口控制一格式 11111ABC

A 为数据传送方向控制：H 表示数据从 LCD 到 MCU，L 表示数据从 MCU 到 LCD

B 为数据类型选择：H 表示数据是显示数据，L 表示数据是控制指令

C 固定为 0

第二字节：(并行)8 位数据的高 4 位一格式 DDDD0000

第三字节：(并行)8 位数据的低 4 位一格式 0000DDDD

串行接口时序参数：(测试条件：T=25℃ VDD=4.5V)

用户指令集在这里我们不重复介绍了，资料中很详细。

在液晶上显示汉字很简单，这里我们详细说明下如何在液晶上显示图形，网上这方面的资料很少。

一、我们来讲下图像显示的基础知识。

对于 12864 这种二值显示屏来说，其显示状态无外乎显示和不显示一个点这两种状态。而在任意位置画点，是我们随心所欲的画线，画圆，画矩形的等 GUI 函数的基础。

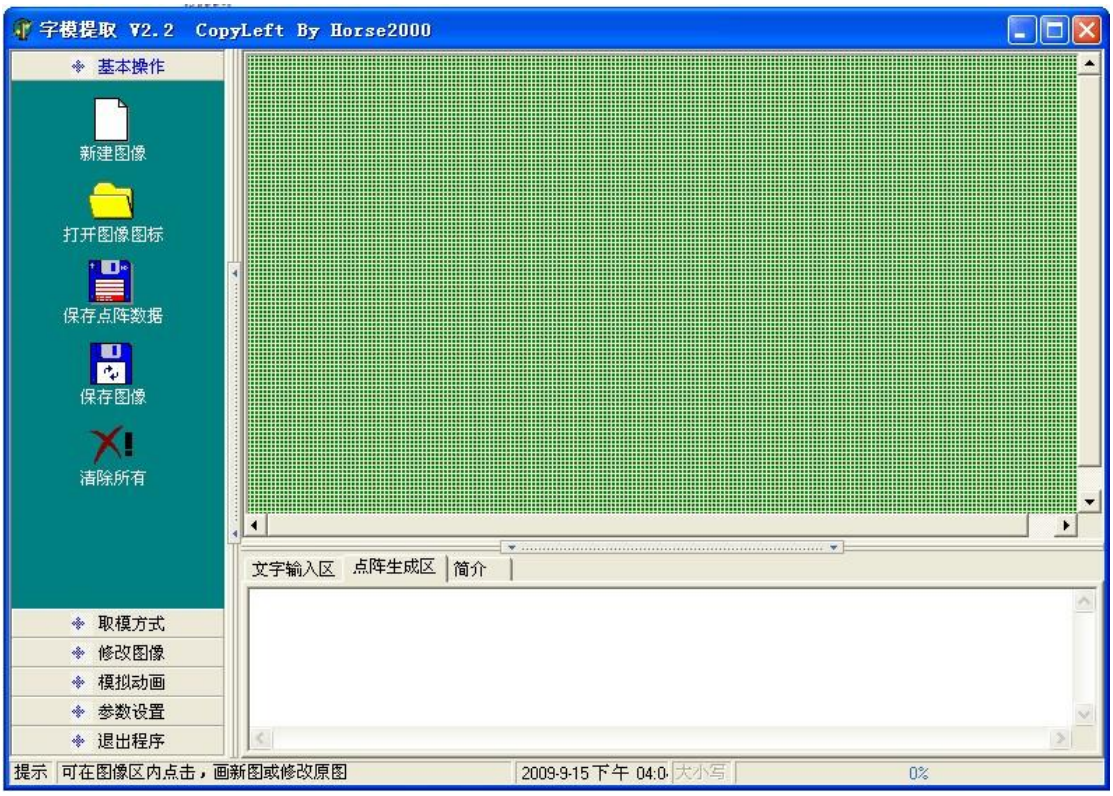
为了让这个位置有一个参考点，我们有必要定义一个坐标系
在这里，我定义的坐标系如下

```
0,0----------------------------------127,0
|                                        |
|                                        |
|                                        |
|                                        |
0,63---------------------------------127,63
```

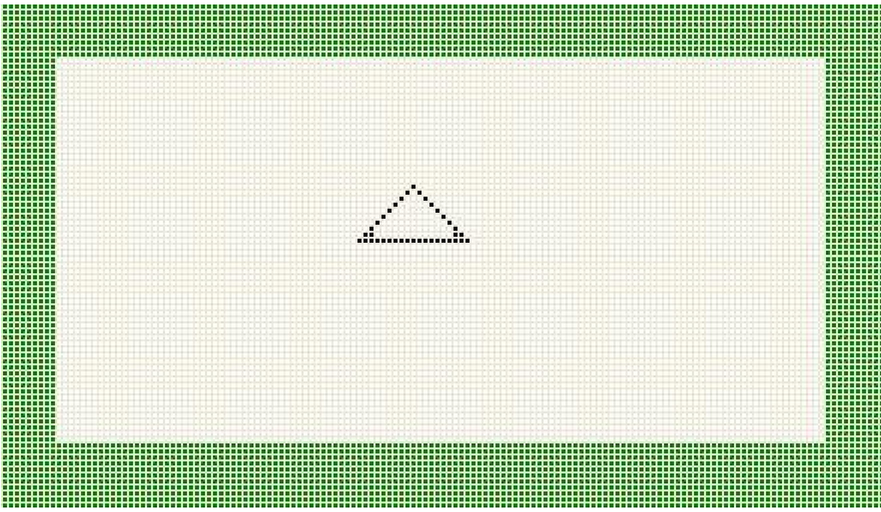
0,0 代表屏幕的左上角，127,63 代表屏幕的右下角。

对于屏幕上面任意一个点，如果我们想要点亮它，必须先读出此点的状态，然后再修改该点，最后送出去，即 读----修改----写。按照这个步骤，然后再运用 C 语言中的位操作运算符 可以很方便的完成画点的函数。

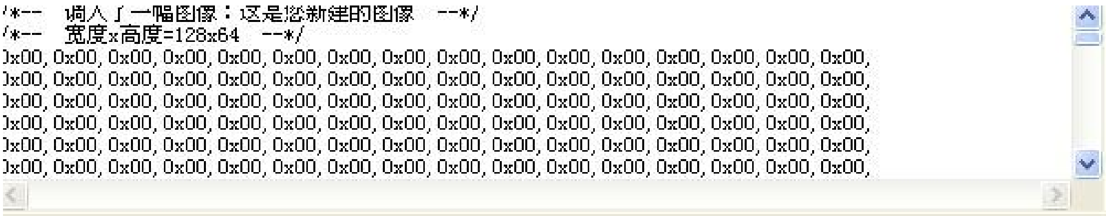
二、我们来介绍下开发板配套资料中的“12864 取模软件”。



建立图形有两种方式：在软件“基本操作”中：一、新建图形 二、打开图像图标



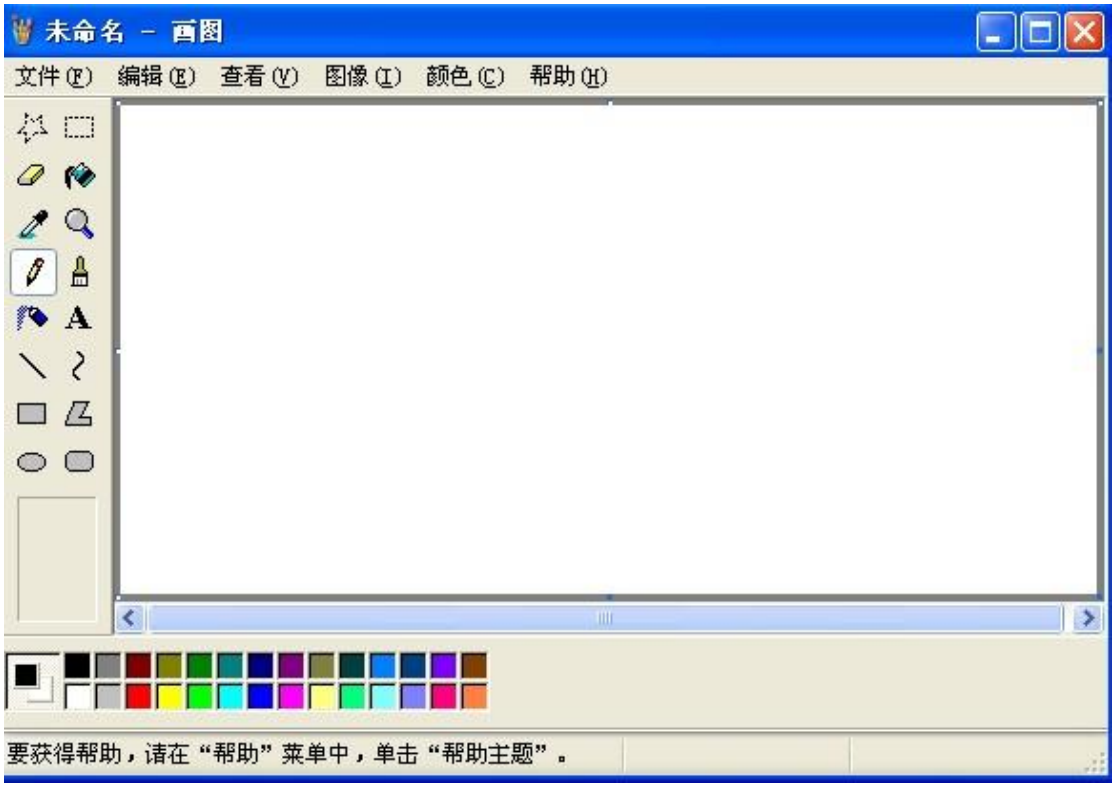
上图是我们用“新建图形”画的，大小设置为宽 128，高 64。
图形画好后，在“取模方式取”下取数据。有 C51 和 A51 两种，看你是用的什么语言编程编程。
我们的配套程序用的 C 语言，所以我们选择 C51 格式。



```
/*--  调入了一幅图像：这是您新建的图像  --*/
/*--  宽度x高度=128x64  --*/
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
```

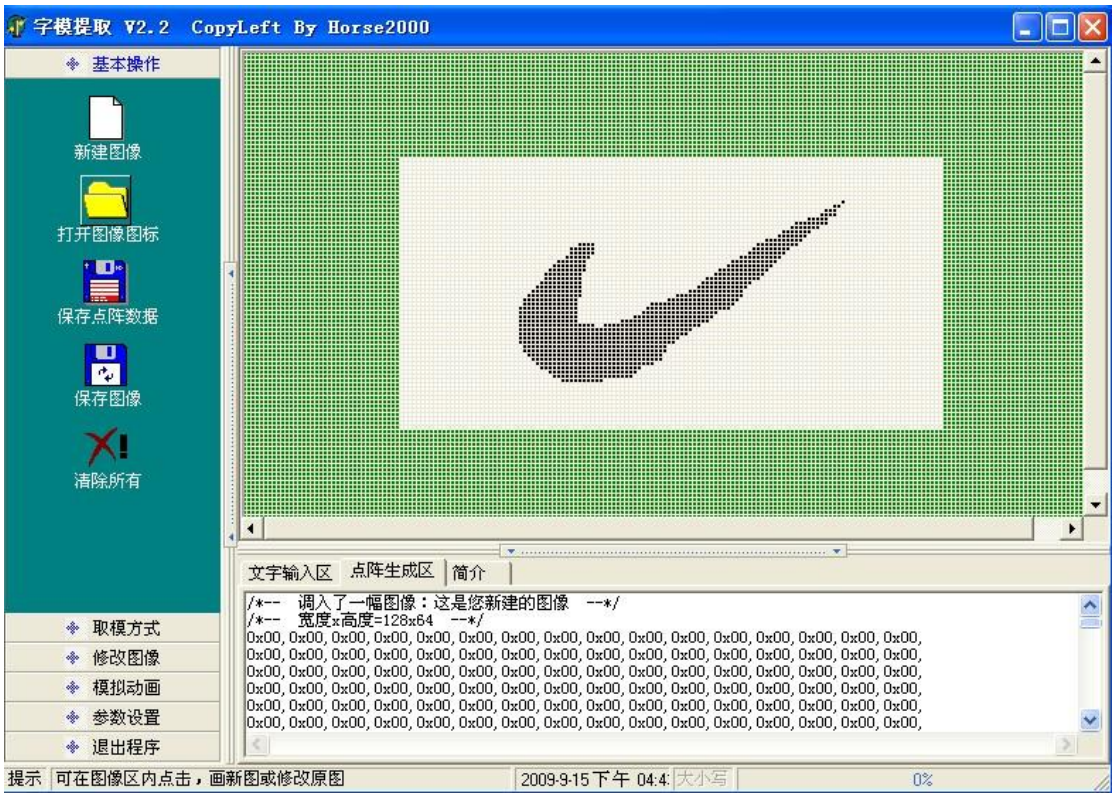
上图就是我们得到的数据，COPY 到我们的程序中就行了。
第二种方法就是通过图形图标来得到数据。
我们可以到入图片，然后再修改，我们也可以用画图板画图片，然后再在 12864 取模软件中修改。

打开画图板，选择“图像”下的“属性”，设置像素的宽度为 128，高度为 64。



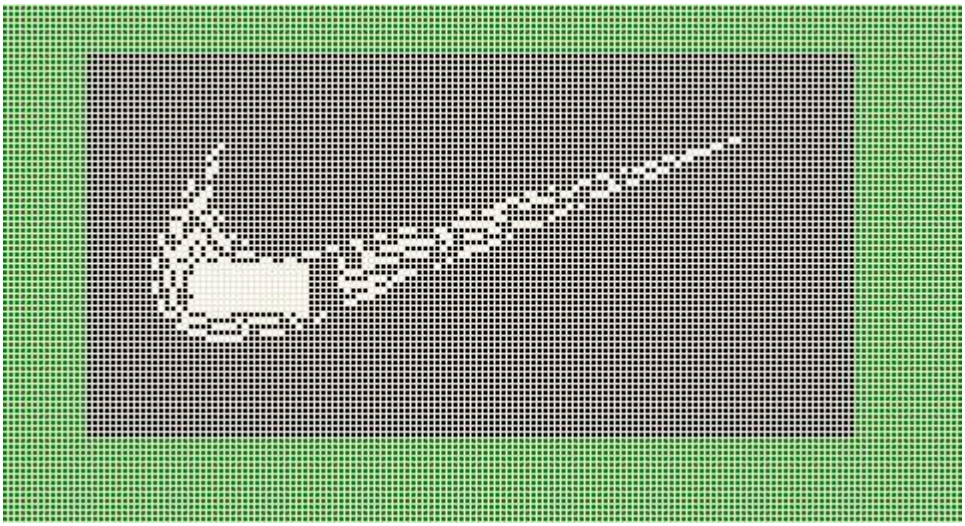
然后选择查看，缩放，大尺寸。得到上图。然后可以进行画图了，比 12864 取模软件必须一个一个点方便多了。

上图就是我们画出的 NIKE 图标，然后保存文件为 BMP 格式。



然后在“12864 取模软件”中选择“打开图像图标”，选择刚才用画图板画的文件。得到了上图图形，然后可以取出数据。

还有，你可以百度搜索你想要的图片，然后再用 QQ 截图工具等截一个 128*64 的图片，当然，这个图片要很小且简单，否则出来时一团糟。然后再导入“12864 取模软件”。



这个就是我们直接用图片导出的。

上图就是在 12864 上的实物照片。

这个是我们用手画的 NIKE 标志。

这个是我截取的芯片的图片。

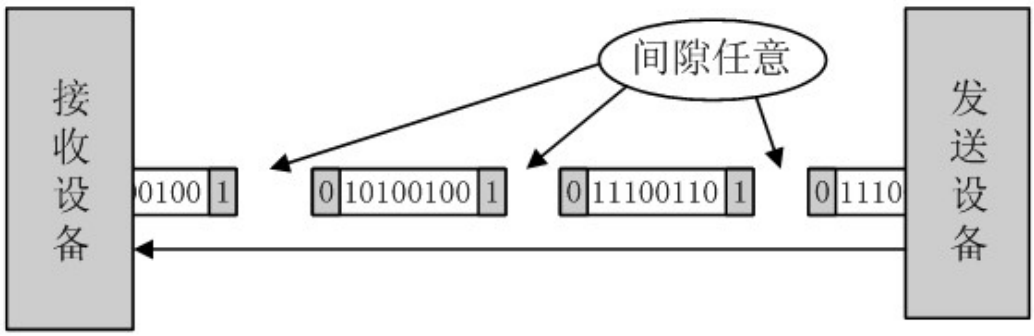
2.2.12 串口通讯

串行通信是将数据字节分成一位一位的形式在一条传输线上逐个地传送。

一、异步通信与同步通信

1、异步通信

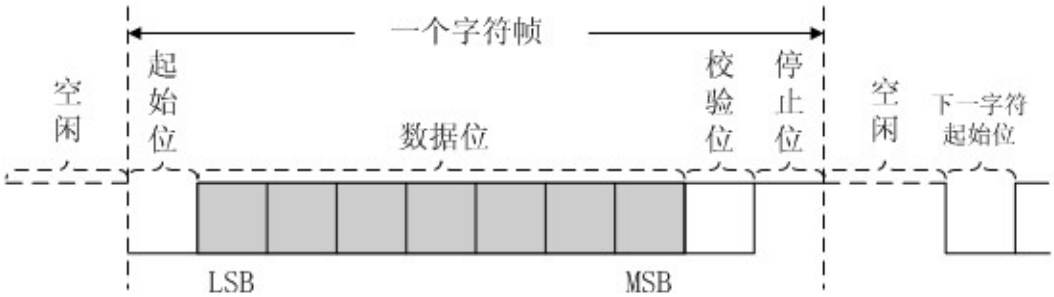
异步通信是指通信的发送与接收设备使用各自的时钟控制数据的发送和接收过程。为使双方的收发协调，要求发送和接收设备的时钟尽可能一致。



异步通信是以字符（构成的帧）为单位进行传输，字符与字符之间的间隙（时间间隔）是任意的，但每个字符中的各位是以固定的时间传送的，即字符之间是异步的（字符之间不一定有“位间隔”的整数倍的关系），但同一字符内的各位是同步的（各位之间的距离均为“位间隔”的整数倍）。

异步通信的数据格式 ：

异步通信的特点：不要求收发双方时钟的严格一致，实现容易，设备开销较小，但每个字符要附加 2～3 位用于起止位，各帧之间还有间隔，因此传输效率不高。



2、同步通信

同步通信时要建立发送方时钟对接收方时钟的直接控制，使双方达到完全同步。此时，传输数据的位之间的距离均为“位间隔”的整数倍，同时传送的字符间不留间隙，即保持位同步关系，也保持字符同步关系。发送方对接收方的同步可以通过两种方法实现。

串行通信的错误校验

1、奇偶校验

在发送数据时，数据位尾随的 1 位为奇偶校验位（1 或 0）。奇校验时，数据中“1”的个数与校验位“1”的个数之和应为奇数；偶校验时，数据中“1”的个数与校验位“1”的个数之和应为偶数。接收字符时，对“1”的个数进行校验，若发现不一致，则说明传输数据过程中出现了差错。

2、代码和校验

代码和校验是发送方将所发数据块求和（或各字节异或），产生一个字节的校验字符（校验和）附加到数据块末尾。接收方接收数据同时对数据块（除校验字节外）求和（或各字节异或），将所得的结果与发送方的“校验和”进行比较，相符则无差错，否则即认为传送过程中出现了差错。

3、循环冗余校验

这种校验是通过某种数学运算实现有效信息与校验位之间的循环校验，常用于对磁盘信息的传输、存储区的完整性校验等。这种校验方法纠错能力强，广泛应用于同步通信中。

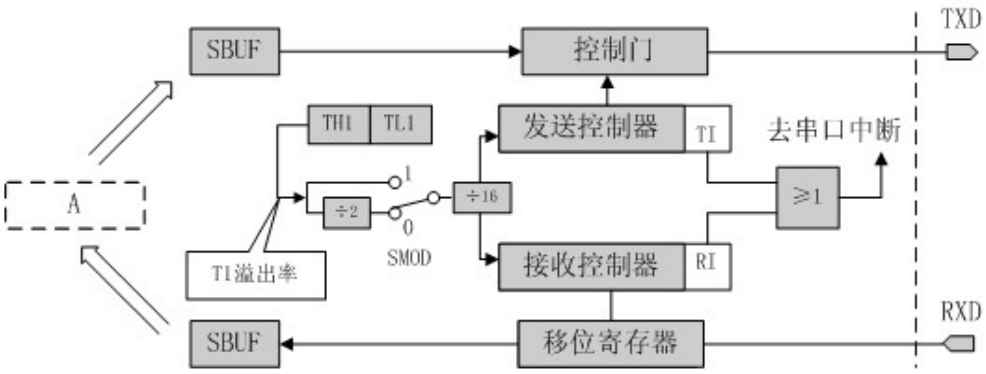
传输速率

比特率是每秒钟传输二进制代码的位数，单位是：位 / 秒（bps）。如每秒钟传送 240 个字符，而每个字符格式包含 10 位(1 个起始位、1 个停止位、8 个数据位)，这时的比特率为：

10 位 × 240 个/秒 = 2400 bps

波特率表示每秒钟调制信号变化的次数，单位是：波特（Baud）。

80C51串行口的结构



有两个物理上独立的接收、发送缓冲器 SBUF，它们占用同一地址99H ；接收器是双缓冲结构 ；发送缓冲器，因为发送时 CPU 是主动的，不会产生重叠错误。

SCON 是一个特殊功能寄存器，用以设定串行口的工作方式、接收/发送控制以及设置状态标志：

位	7	6	5	4	3	2	1	0	
字节地址：98H	SM0	SM1	SM2	REN	TB8	RB8	TI	RI	SCON

SM0和SM1为工作方式选择位，可选择四种工作方式：

串行口的工作方式

SM0	SM1	方 式	说 明	波特率
0	0	0	移位寄存器	$f_{osc}/12$
0	1	1	10 位异步收发器（8 位数据）	可变
1	0	2	11 位异步收发器（9 位数据）	$f_{osc}/64$ 或 $f_{osc}/32$
1	1	3	11 位异步收发器（9 位数据）	可变

●TI，发送中断标志位。在方式0时，当串行发送第8位数据结束时，或在其它方式，串行发送停止位的开始时，由内部硬件使 TI 置1，向 CPU 发中断申请。在中断服务程序中，必须用软件将其清0，取消此中断申请。

●RI，接收中断标志位。在方式0时，当串行接收第8位数据结束时，或在其它方式，串行接收停止位的中间时，由内部硬件使 RI 置1，向 CPU 发中断申请。也必须在中断服务程序中，用软件将其清0，取消此中断申请。

SM2，多机通信控制位，主要用于方式2和方式3。当接收机的 SM2=1时可以利用收到的 RB8来控制是否激活 RI（RB8＝0时不激活 RI，收到的信息丢弃；RB8＝1时收到的数据进入 SBUF，并激活 RI，进而在中断服务中将数据从 SBUF 读走）。当 SM2=0时，不论收到的 RB8为0和1，均可以使收到的数据进入 SBUF，并激活

RI（即此时 RB8不具有控制 RI 激活的功能）。通过控制 SM2，可以实现多机通信。在方式0时，SM2必须是0。在方式1时，若 SM2=1，则只有接收到有效停止位时，RI 才置1。

●REN，允许串行接收位。由软件置 REN=1，则启动串行口接收数据；若软件置 REN=0，则禁止接收。

●TB8，在方式2或方式3中，是发送数据的第九位，可以用软件规定其作用。可以用作数据的奇偶校验位，或在多机通信中，作为地址帧/数据帧的标志位。在方式0和方式1中，该位未用。

●RB8，在方式2或方式3中，是接收到数据的第九位，作为奇偶校验位或地址帧/数据帧的标志位。在方式1时，若 SM2=0，则 RB8是接收到的停止位。

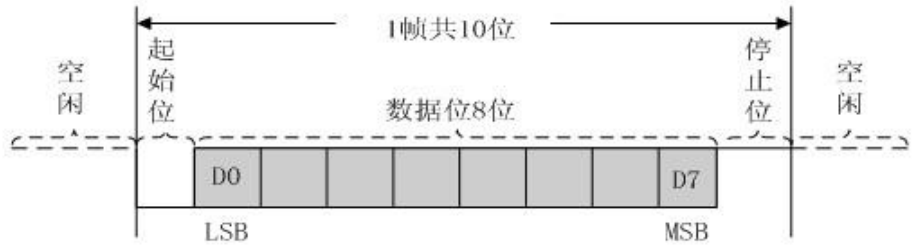
PCON 中只有一位 SMOD 与串行口工作有关 ：

位	7	6	5	4	3	2	1	0	
字节地址：97H	SMOD								PCON

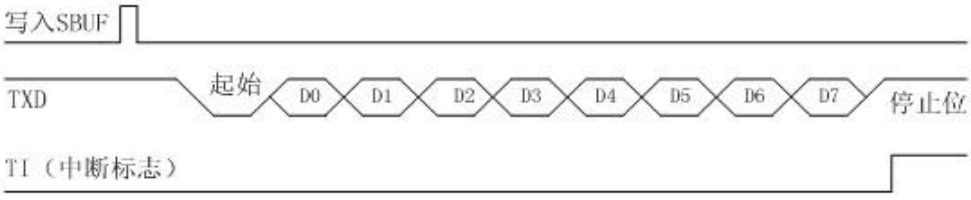
SMOD（PCON.7） 波特率倍增位。在串行口方式1、方式2、方式3时，波特率与 SMOD 有关，当 SMOD=1时，波特率提高一倍。复位时，SMOD=0。

我们这里只简单的介绍下方式一。其他方式请自行查看资料。

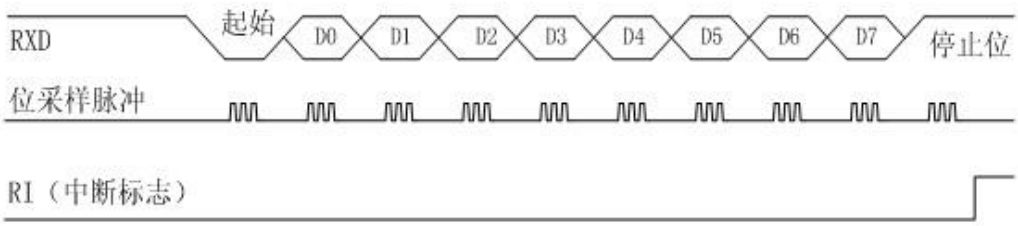
方式1是10位数据的异步通信口。TXD 为数据发送引脚，RXD 为数据接收引脚，传送一帧数据的格式如图所示。其中1位起始位，8位数据位，1位停止位。



1、方式1输出



2、方式1输入



用软件置 REN 为1时，接收器以所选择波特率的16倍速率采样 RXD 引脚电平，检测到 RXD 引脚输入电平发生负跳变时，则说明起始位有效，将其移入输入移位寄存器，并开始接收这一帧信息的其余位。接收过程中，数据从输入移位寄存器右边移入，起始位移至输入移位寄存器最左边时，控制电路进行最后一次移位。当 RI=0，且 SM2=0（或接收到的停止位为1）时，将接收到的9位数据的前8位数　　据装入接收 SBUF，第9位（停止位）进入 RB8，并置 RI=1，向 CPU 请求中断。

波特率方式1的

波特率 =（2SMOD/32）•（T1溢出率）的计算

当 T1作为波特率发生器时，最典型的用法是使 T1工作在自动再装入的8位定时器方式（即方式2，且 TCON 的 TR1=1，以启动定时器）。这时溢出率取决于 TH1中的计数值。

T1 溢出率 = fosc /{12 × [256 − （TH1）]}

在单片机的应用中，常用的晶振频率为：12MHz 和11.0592MHz。所以，选用的波特率也相对固定。常用的串行口波特率以及各参数的关系如表所示。

常用波特率与定时器 1 的参数关系

串口工作方式及波特率/（b/s）		fosc（MHz）	SMOD	定时器 T1		
				C/T	工作方式	初值
方式 1、3	62.5 k	12	1	0	2	FFH
	19.2 k	11.0592	1	0	2	FDH
	9600	11.0592	0	0	2	FDH
	4800	11.0592	0	0	2	FAH
	2400	11.0592	0	0	2	F4H
	1200	11.0592	0	0	2	E8H

串行口工作之前，应对其进行初始化，主要是设置产生波特率的定时器1、串行口控制和中断控制。具体步骤如下：

- 确定 T1的工作方式（编程 TMOD 寄存器）；
- 计算 T1的初值，装载 TH1、TL1；
- 启动 T1（编程 TCON 中的 TR1位）；
- 确定串行口控制（编程 SCON 寄存器）；
- 串行口在中断方式工作时，要进行中断设置（编程 IE、IP 寄存器）。

因为我们收到的是字符型的 ASCII 码数据，如数字“0”的 ASCII 码值是48，所以，我们要显示“0”的话，还需要将其值减去48后才是真正要显示的数据。

串口调试软件中，设置参数如下：串口：COMX（请查看你的开发板是和电脑的哪个串口相连）；波特率：9600；校验位：无；数据位：8位；停止位：1位；发送内容：123456780abcdef

下面是单片机串口接收数据的程序实例：

```
/*********************************************************************
* 名称 : Com_Int()
* 功能 : 串口中断子函数
* 输入 : 无
* 输出 : 无
*********************************************************************/
void Com_Int(void) interrupt 4
{
    EA = 0;
    if(RI == 1)    //当硬件接收到一个数据时，RI 会置位
    {
        LED_Buffer[com_dat] = SBUF;         //把从串口读出的数存到数组
        RI = 0;
        com_dat++;
        if(com_dat == 16) com_dat = 0;          //当 com_dat = 16时，清0，防止数组溢出
    }
    EA = 1;
}

/*********************************************************************
* 名称 : Com_Init()
* 功能 : 串口初始化，晶振11.0592,波特率9600，使能了串口中断
* 输入 : 无
* 输出 : 无
*********************************************************************/
void Com_Init(void)
{
      TMOD = 0x20;    //定时器工作在定时器1的方式2
      PCON = 0x00;    //不倍频
      SCON = 0x50; //串口工作在方式1，并且启动串行接收
      TH1 = 0xFd;      //设置波特率  9600
      TL1 = 0xFd;
      TR1 = 1;        //启动定时器1
      ES = 1;         //开串口中断
      EA = 1;         //开总中断
```

```
}
```

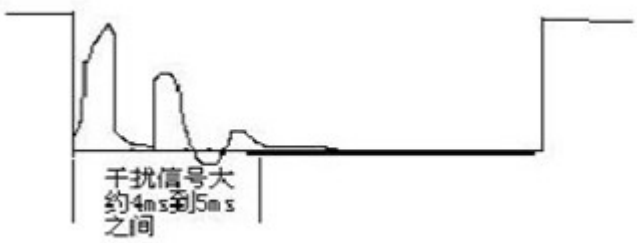
下面是单片机串口发送数据的实例：

```
/*********************************************************************
* 名称 : Com_Init()
* 功能 : 初始化串口程序，晶振11.0592，波特率9600
* 输入 : 无
* 输出 : 无
*********************************************************************/
void Com_Init(void)
{
    TMOD = 0x20;
    PCON = 0x00;
    SCON = 0x50;
    TH1 = 0xFd;
    TL1 = 0xFd;
    TR1 = 1;
}

/*********************************************************************
* 名称 : Main()
* 功能 : 主函数
* 输入 : 无
* 输出 : 无
*********************************************************************/
void Main()
{
  uchar i = 0;
  uchar code Buffer[] = "HOT-51   QQ：1090592530";  //所要发送的数据
  uchar *p;
  Com_Init();
  P2 = 0x00;
  p = Buffer;
  while(1)
  {

    SBUF = *p;
    while(!TI)                    //如果发送完毕，硬件会置位 TI
    {
      _nop_();
    }
    p++;
    if(*p == '\0') break;       //在每个字符串的最后，会有一个'\0'
    TI = 0;                       //TI 清零
```

```
    }
    while(1);
}
```

2.2.13 按键

一般情况下，一个按键按下的时候，总是在按下的时刻存在着一定的干扰信号，按下之后就基本上进入了稳定的状态。具体的一个按键从按下到释放的全过程的信号图如上图所示：



从图中可以看出，我们在程序设计时，从按键被识别按下之后，延时 5ms 以上 ，从而避开了干扰信号区域，我们再来检测一次，看按键是否真得已经按下，若真得已经按下，这时肯定输出为低电平，若这时检测到的是高电平，证明刚才是由于干扰信号引起的误触发，CPU 就认为是误触发信号而舍弃这次的按键识别过程。从而提高了系统的可靠性。

查询方式案件实验（配套试验四部分程序）

```
/************************************************************************
    * 名称 : Delay()
    * 功能 : 实现按键功能，当按键按下时，相应的LED亮灭交替
    * 输入 : 无
    * 输出 : 无
************************************************************************/
    void KEY()
    {
        if(KEY1==0 || KEY2==0 || KEY3==0)
        {
            Delay(2);                //20毫秒软件防抖
            if(KEY1==0 || KEY2==0 || KEY3==0)
            {
                if(KEY1 == 0)
                {
                    LED1 = ~ LED1;      //LED显示取反
                }
                else if(KEY2 == 0)
                {
```

```
                LED2 = ~ LED2;
            }
            else
            {
                LED3 = ~ LED3;
            }
        }
        Delay(50);              //延时0.5秒再进行下次按键的检测
    }
}
```

为了提高响应速度，一般实时性高的事件，我们都中断进行处理。请参考前面 2.2.4 节的中断。

2.2.14 矩阵键盘

独立按键具有编程简单但占用 I/O 口资源的特点，不适合在按键较多的场合应用。在实际应用中经常要用到输入数字、字母等功能，如电子密码锁、电话机键盘等一般都至少有 12 到 16 个按键，在这种情况下如果用独立按键的话显然太浪费 I/O 口资源，为此我们就引入了矩阵键盘的应用。

矩阵键盘简介：

矩阵键盘又称行列键盘，它是用四条 I/O 线作为行线，四条 I/O 线作为列线组成的键盘。在行线和列线的每个交叉点上设置一个按键。这样键盘上按键的个数就为 4*4 个。这种行列式键盘结构能有效地提高单片机系统中 I/O 口的利用率。

矩阵键盘的工作原理：

最常见的键盘布局如图 1 所示。一般由 16 个按键组成，在单片机中正好可以用一个 P 口实现 16 个按键功能，这也是在单片机系统中最常用的形式，4*4 矩阵键盘的内部电路如图 2 所示。

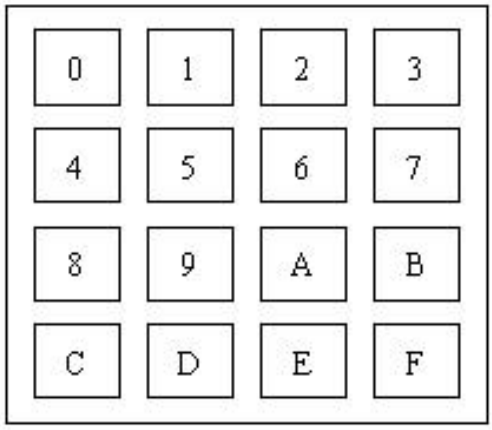


图1 矩阵键盘布局图

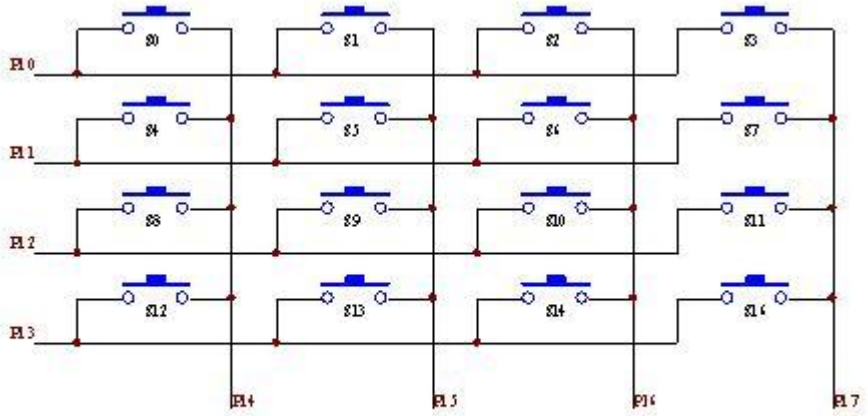


图2 矩阵键盘内部电路图

当无按键闭合时，P10~P13 与 P14~P17 之间开路。当有键闭合时，与闭合键相连的两条 I/O 口线之间短路。判断有无按键按下的方法是：第一步，置列线 P14~P17 为输入状态，从行线 P10~P13 输出低电平，读入列线数据，若某一列线为低电平，则该列线上有键闭合。第二步，行线轮流输出低电平，从列线 P14~P17 读入数据，若有某一列为低电平，则对应行线上有键按下。综合一二两步的结果，可确定按键编号。但是键闭合一次只能进行一次键功能操作，因此须等到按键释放后，再进行键功能操作，否则按一次键，有可能会连续多次进行同样的键操作。

矩阵键盘软硬件设计实例

本期以51单片机综合学习系统为硬件平台，介绍矩阵式键盘的编程方法。实验通过按下相应键后在一位数码管上显示出键值。0到16个键分别对应显示0到F。

本实验可以直接在配套开发板上完成，其硬件原理图如图4所示：
根据4电路原理图，键盘扫描方法是：行线 P10~P13为输出线，列线 P14~P17为输入线。一开始单片机将行线（P10~P13）全部输出低电平，此时读入列线数

据，若列线全为高电平则没有键按下，当列线有出现低电平时调用延时程序以此来去除按键抖动。延时完成后再判断是否有低电平，如果此时读入列线数据还是有低电平，则说明确实有键按下。最后一步确定键值。现在我们以第二行的 S5 键为例，若按下 S5后我们应该怎么得到这个键值呢？当判断确实有键按下之后，行线轮流输出低电平，根据读入列线的数据可以确定键值。首先，单片机将 P10 输出为低电平，其它 P11~P13输出高电平，此时读取列线的数据全为高电平，说明没有在第一行有键按下；其次，单片机将 P11输出低电平，其它 P10、P12、P13仍为高电平，此时再来读取列线数据，发现列线读到的数据有低电平，数值为1011（0x0B），如果我们的键盘布局已经确定，那么0x0B 就代表 S5的值了。转到 S5键功能处理子程序就可以达到目的。

```
/*********************************************************************
    * 名称 : Keyscan()
    * 功能 : 实现按键的读取。下面这个子程序是按处理 矩阵键盘 的基本方法处理的。
              该函数的原理将在“视频及教程”文件夹中介绍。
    * 输入 : 无
    * 输出 : 按键值
*********************************************************************/
    uchar Keyscan(void)
    {
        uchar i,j, temp, Buffer[4] = {0xef, 0xdf, 0xbf, 0x7f};
        for(j=0; j<4; j++)
        {
            P1 = Buffer[j];
            /*以下三个_nop_();作用为让 P1 口的状态稳定*/
            _nop_();
            _nop_();
            _nop_();
            temp = 0x01;
            for(i=0; i<4; i++)
            {
                if(!(P1 & temp))
                {
                    return (i+j*4);             //返回取得的按键值
                }
                temp <<= 1;
            }
        }
    }
```

2.2.15 蜂鸣器唱歌

⑴ 总体原理：
乐曲中不同的音符，实质就是不同频率的声音。通过单片机产生不同的频率的脉冲信号，经过放大电路，由蜂鸣器放出，就产生了美妙和谐的乐曲。
⑵ 单片机产生不同频率脉冲信号的原理：
1）要产生音频脉冲，只要算出某一音频的脉冲（1/频率），然后将此周期除以2，即为半周期的时间，利用定时器计时这个半周期的时间，每当计时到后就将输出脉冲的 I/O 反相，然后重复计时此半周期的时间再对 I/O 反相，就可以在 I/O 脚上得到此频率的脉冲。
2）利用8051的内部定时器使其工作在计数器模式 MODE1下，改变计数值 TH0 及 TL0以产生不同频率的方法如下：
例如，频率为523Hz，其周期天/523 S=1912uS，因此只要令计数器计时956uS/1us=956，在每计数956次时就将 I/O 反接，就可得到中音 DO（532Hz）。
计数脉冲值与频率的关系公式如下：
N=Fi/2/Fr
（N：计数值，Fi：内部计时一次为1uS，故其频率为1MHz，Fr：要产生的频率）
⑶ 其计数值的求法如下：
T=65536-N=65536-Fi/2/Fr
计算举例：
设 K=65536,F=1000000=Fi=1MHz，求低音 DO（261Hz）、中音 DO（523Hz）、高音 DO（1046Hz）的计数值。
T=65536-N=65536-Fi/2/Fr=65536-1000000/2/Fr=65536-500000/Fr
低音 DO 的 T=65536-500000/262=63627
中音 DO 的 T=65536-500000/523=64580
高音 DO 的 T=65536-500000/1047=65059

音符	频率（HZ）	简谱码（T值）	音符	频率（HZ）	简谱码（T值）
低 1 DO	262	63628	# 4 FA#	740	64860
#1 DO#	277	63731	中 5 SO	784	64898
低 2 RE	294	63835	# 5 SO#	831	64934
#2 RE#	311	63928	中 6 LA	880	64968
低 3 M	330	64021	# 6	932	64994
低 4 FA	349	64103	中 7 SI	988	65030
# 4 FA#	370	64185	高 1 DO	1046	65058
低 5 SO	392	64260	# 1 DO#	1109	65085
# 5 SO#	415	64331	高 2 RE	1175	65110
低 6 LA	440	64400	# 2 RE#	1245	65134
# 6	466	64463	高 3 M	1318	65157
低 7 SI	494	64524	高 4 FA	1397	65178
中 1 DO	523	64580	# 4 FA#	1480	65198
# 1 DO#	554	64633	高 5 SO	1568	65217
中 2 RE	587	64684	# 5 SO#	1661	65235
# 2 RE#	622	64732	高 6 LA	1760	65252
中 3 M	659	64777	# 6	1865	65268
中 4 FA	698	64820	高 7 SI	1967	65283

⑸ 每个音符使用1个字节，字节的高4位代表音符的高低，低4位代表音符的节拍，下表为节拍码的对照。但如果1拍为0.4秒，1/4拍是0.1秒，只要设定延迟时间就可求得节拍的时间。假设1/4节拍为1DELAY，则1拍应为4DELAY，以此类推。所以只要求得1/4拍的 DELAY 时间，其余的节拍就是它的倍数，如下表为1/4和1/8节拍的时间设定。

表9.2 节拍码对照表

1/4节拍		1/8节拍	
节拍码	节拍数	节拍码	节拍数
1	1/4拍	1	1/8拍
2	2/4拍	2	1/4拍
3	3/4拍	3	3/8拍
4	1拍	4	1/2拍
5	1又1/4拍	5	5/8拍
6	1又1/2拍	6	3/4拍
7	1又3/4拍	7	7/8拍
8	2拍	8	1拍
9	2又1/4拍	9	1又1/8拍
A	2又1/2拍	A	1又1/4拍
B	2又3/4拍	B	1又3/8拍
C	3拍	C	1又1/2拍
D	3又1/4拍	D	1又5/8拍
E	3又1/2拍	E	1又3/4拍
F	3又3/4拍	F	1又7/8拍

表9.3 各调节拍的时间设定表

1/4节拍 1/8节拍

曲调值	DELAY	曲调值	DELAY
调4/4	125毫秒	调4/4	62毫秒
调3/4	187毫秒	调3/4	94毫秒
调2/4	250毫秒	调2/4	125毫秒

⑹ 建立音乐的步骤：

1）先把吧乐谱的音符找出，然后由上表建立 T 值表的顺序。

2）把 T 值表建立在 TABLE1，构成发音符的计数值放在“TABLE”。

3）简谱码（音符）为高位，节拍为（节拍数）为低4位，音符节拍码放在程序的“TABLE”处。

表9.4 简谱对应的简谱码、T 值、节拍数

简谱	发音	简谱码	T 值	节拍码	节拍数
5	低5SO	1	64260	1	1/4拍
6	低6LA	2	64400	2	2/4拍
7	低7SI	3	64524	3	3/4拍
1	中1DO	4	64580	4	1拍
2	中2RE	5	64684	5	1又1/4拍
3	中3M	6	64777	6	1又2/4拍
4	中4FA	7	64820	7	1又3/4拍
5	中5SO	8	64898	8	2拍
6	中6LA	9	64968	9	2又1/4拍
7	中7SI	A	65030	A	2又2/4拍
1	高1DO	B	65058	B	2又3/4拍
2	高2RE	C	65110	C	3拍
3	高3M	D	65157	D	3又1/4拍
4	高4FA	E	65178	E	3又2/4拍
5	高5SO	F	65217	F	3又3/4拍

不发音 0

```
unsigned char code SONG[] ={        //祝你平安
0x26,0x20,0x20,0x20,0x20,0x20,0x26,0x10,0x20,0x10,0x20,0x80,0x26,0x20,0x3
0,0x20,
0x30,0x20,0x39,0x10,0x30,0x10,0x30,0x80,0x26,0x20,0x20,0x20,0x20,0x20,0x1
c,0x20,
0x20,0x80,0x2b,0x20,0x26,0x20,0x20,0x20,0x2b,0x10,0x26,0x10,0x2b,0x80,0x2
6,0x20,
0x30,0x20,0x30,0x20,0x39,0x10,0x26,0x10,0x26,0x60,0x40,0x10,0x39,0x10,0x2
6,0x20,
0x30,0x20,0x30,0x20,0x39,0x10,0x26,0x10,0x26,0x80,0x26,0x20,0x2b,0x10,0x2
b,0x10,
0x2b,0x20,0x30,0x10,0x39,0x10,0x26,0x10,0x2b,0x10,0x2b,0x20,0x2b,0x40,0x4
0,0x20,
0x20,0x10,0x20,0x10,0x2b,0x10,0x26,0x30,0x30,0x80,0x18,0x20,0x18,0x20,0x2
```

```
6,0x20,
0x20,0x20,0x20,0x40,0x26,0x20,0x2b,0x20,0x30,0x20,0x30,0x20,0x1c,0x20,0x20,0x20,
0x20,0x80,0x1c,0x20,0x1c,0x20,0x1c,0x20,0x30,0x20,0x30,0x60,0x39,0x10,0x30,0x10,
0x20,0x20,0x2b,0x10,0x26,0x10,0x2b,0x10,0x26,0x10,0x26,0x10,0x2b,0x10,0x2b,0x80,
0x18,0x20,0x18,0x20,0x26,0x20,0x20,0x20,0x20,0x60,0x26,0x10,0x2b,0x20,0x30,0x20,
0x30,0x20,0x1c,0x20,0x20,0x20,0x20,0x80,0x26,0x20,0x30,0x10,0x30,0x10,0x30,0x20,
0x39,0x20,0x26,0x10,0x2b,0x10,0x2b,0x20,0x2b,0x40,0x40,0x10,0x40,0x10,0x20,0x10,
0x20,0x10,0x2b,0x10,0x26,0x30,0x30,0x80,0x00,
//路边的野华不要采
0x30,0x1C,0x10,0x20,0x40,0x1C,0x10,0x18,0x10,0x20,0x10,0x1C,0x10,0x18,0x40,0x1C,
0x20,0x20,0x20,0x1C,0x20,0x18,0x20,0x20,0x80,0xFF,0x20,0x30,0x1C,0x10,0x18,0x20,
0x15,0x20,0x1C,0x20,0x20,0x20,0x26,0x40,0x20,0x20,0x2B,0x20,0x26,0x20,0x20,0x20,
0x30,0x80,0xFF,0x20,0x20,0x1C,0x10,0x18,0x10,0x20,0x20,0x26,0x20,0x2B,0x20,0x30,
0x20,0x2B,0x40,0x20,0x20,0x1C,0x10,0x18,0x10,0x20,0x20,0x26,0x20,0x2B,0x20,0x30,
0x20,0x2B,0x40,0x20,0x30,0x1C,0x10,0x18,0x20,0x15,0x20,0x1C,0x20,0x20,0x20,0x26,
0x40,0x20,0x20,0x2B,0x20,0x26,0x20,0x20,0x20,0x30,0x80,0x20,0x30,0x1C,0x10,0x20,
0x10,0x1C,0x10,0x20,0x20,0x26,0x20,0x2B,0x20,0x30,0x20,0x2B,0x40,0x20,0x15,0x1F,
0x05,0x20,0x10,0x1C,0x10,0x20,0x20,0x26,0x20,0x2B,0x20,0x30,0x20,0x2B,0x40,0x20,
0x30,0x1C,0x10,0x18,0x20,0x15,0x20,0x1C,0x20,0x20,0x20,0x26,0x40,0x20,0x20,0x2B,
0x20,0x26,0x20,0x20,0x20,0x30,0x30,0x20,0x30,0x1C,0x10,0x18,0x40,0x1C,0x20,0x20,
0x20,0x26,0x40,0x13,0x60,0x18,0x20,0x15,0x40,0x13,0x40,0x18,0x80,0x00,

};

/*****************************************************************
* 名称 : Time0_Init()
* 功能 : 定时器的初始化，定时时间可用光碟中软件计算，11.0592MZ 晶振，
```

```
10ms
* 输入 : 无
* 输出 : 无
***********************************************************************/
void  Time0_Init()
{
    TMOD  =  0x01;
    IE    =  0x82;
    TH0   =  0xDC;
    TL0   =  0x00;
}

/***********************************************************************
* 名称 : Time0_Int()
* 功能 : 定时器中断，中断中实现 Count 加一
* 输入 : 无
* 输出 : 无
***********************************************************************/
void  Time0_Int() interrupt 1
{
    TH0  =  0xDC;
    TL0  =  0x00;
    Count++;              //长度加1
}
/***********************************************************************
* 名称 : Play_Song()
* 功能 : 播放蜂鸣器控制程序
* 输入 : i (选择播放哪首歌曲，0为“祝你平安”，1为“路边的野花你不要采”
* 输出 : 无
***********************************************************************/
void  Play_Song(uchar  i)
{
    uchar  Temp1,Temp2;
    uint  Addr;
    Count  =  0;                        //中断计数器清0
    Addr  =  i  *  217;
    while(1)
    {
        Temp1  =  SONG[Addr++];
        if  (Temp1  ==  0xFF)             //休止符
        {
            TR0  =  0;
            Delay_xMs(100);
        }
```

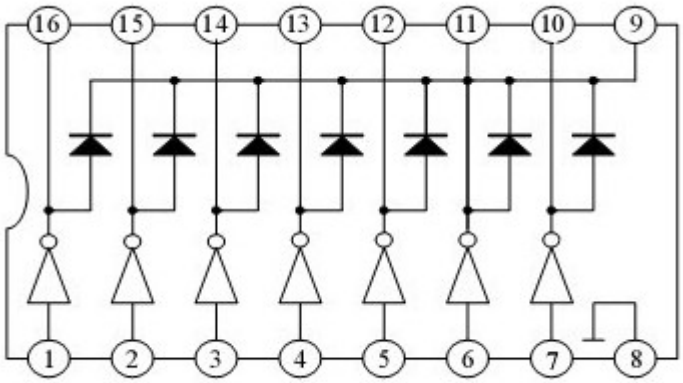
```
        else if (Temp1 == 0x00)     //歌曲结束符
        {
            return;
        }
        else
        {
            Temp2 = SONG[Addr++];
            TR0 = 1;
            while(1)
            {
                Speak = ~Speak;
                Delay_xMs(Temp1);
                if(Temp2 == Count)
                {
                    Count = 0;
                    break;
                }
            }
        }
    }
}
```

2.2.16 ULN2003 基本资料

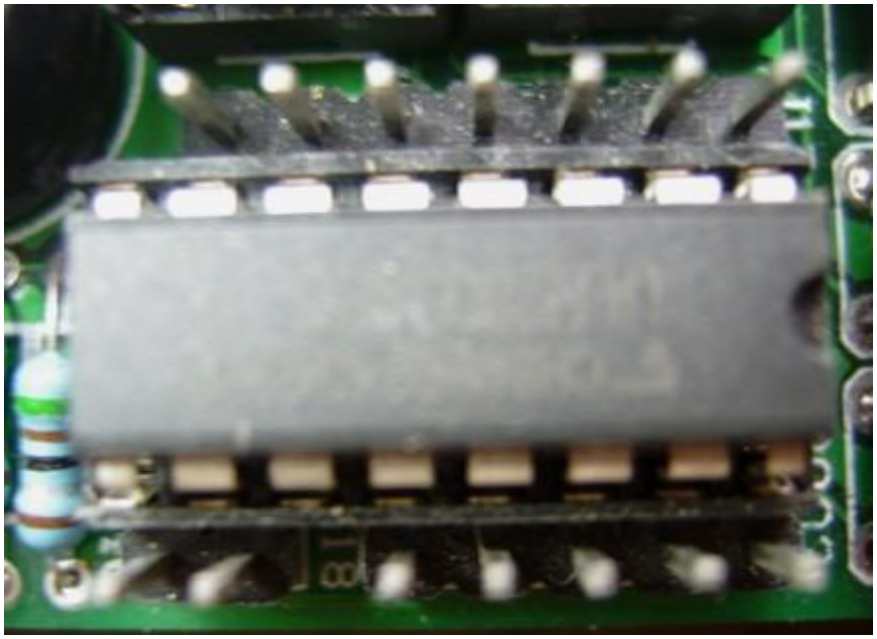
uln 是集成达林顿管 IC，内部还集成了一个消线圈反电动势的二极管，可用来驱动继电器。它是双列 16 脚，NPN 晶体管矩阵，最大驱动电压 50V，电流 500MA，输入电压 5V，适用于 TTL，COMS，由达林顿管组成驱动电路。它的输出端允许通过电流为 200MA，饱和压降 CVE 为 1V 左右，耐压 BVCEO 约为 36V，用户输出口的外接负载可根据以上参数估算。

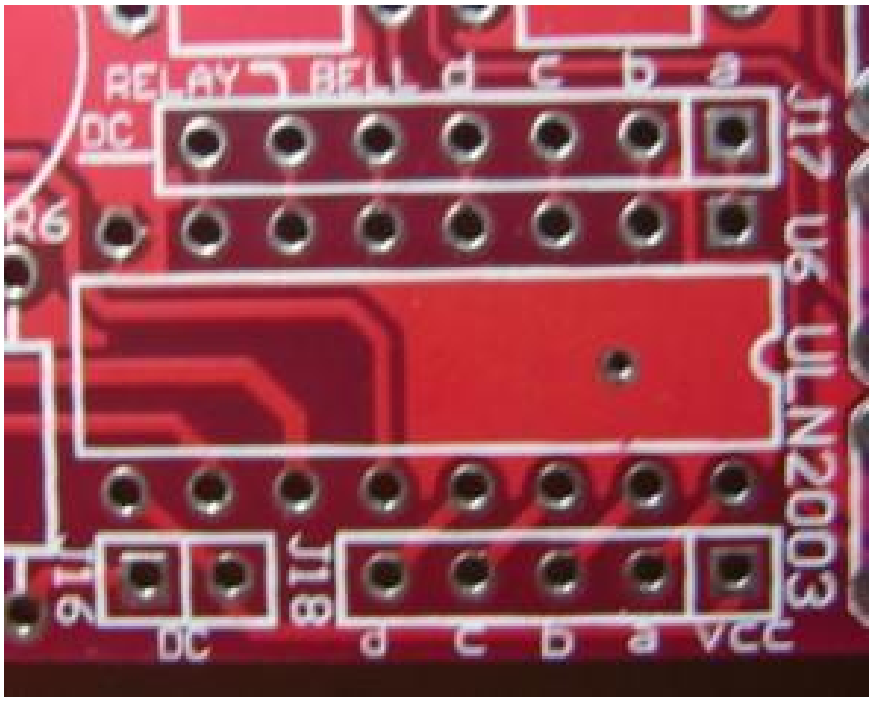
采用极电集开路输出，输出电流大，故可直接驱动继电器或固体继电器，也可直接驱动低压电灯泡，通常单片机驱动 ULN2003 时，上拉 2K 的电阻较为合适，同时，COM 引脚应该悬空或接电源。

ULN2003 是一个非门电路，包含 7 个单元，单独每个单元驱动电流最大可达 350MA，资料的最后有引用电路，9 脚可以悬空。



在本开发板中，我们用 ULN2003 来驱动步进电机，直流电机，继电器，蜂鸣器。这四个试验都需要用杜邦线来连接。





上一行排阵，从左到右分别为 P1.0--P1.6。

- P1.0 连接 J17 的 DC 端，通过 ULN2003 然后去驱动直流电机，直流电机接 J16。
- P1.1 连接 J17 的 RELAY 端，通过 ULN2003 然后去驱动继电器。
- P1.2 连接 J17 的 BELL 端，通过 ULN2003 然后去驱动蜂鸣器。
- P1.3--P.16 连接 J17 的 d,c,b,a，通过 ULN2003 然后去驱动步进电机，步进电机连接 J18。

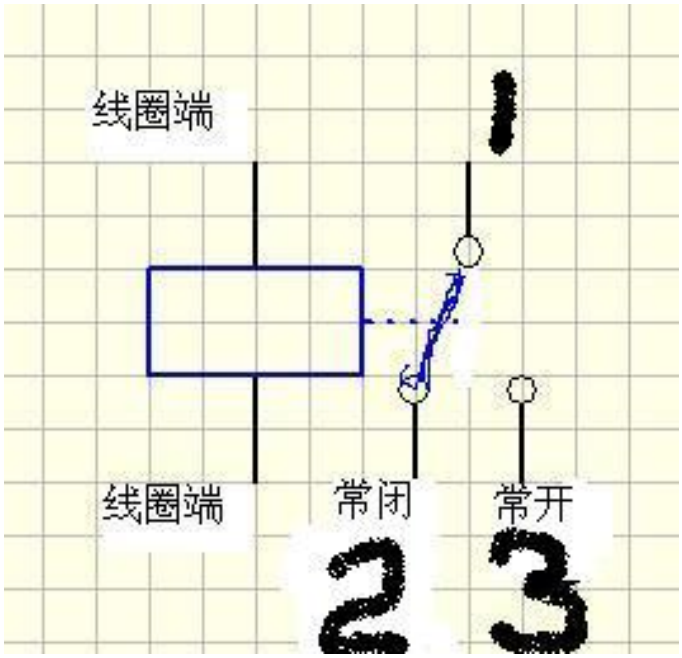
2.2.17 继电器

继电器的控制我们是通过杜邦线连接到 ULN2003，然后通过 ULN2003 来驱动继电器。继电器的工作原理和特性

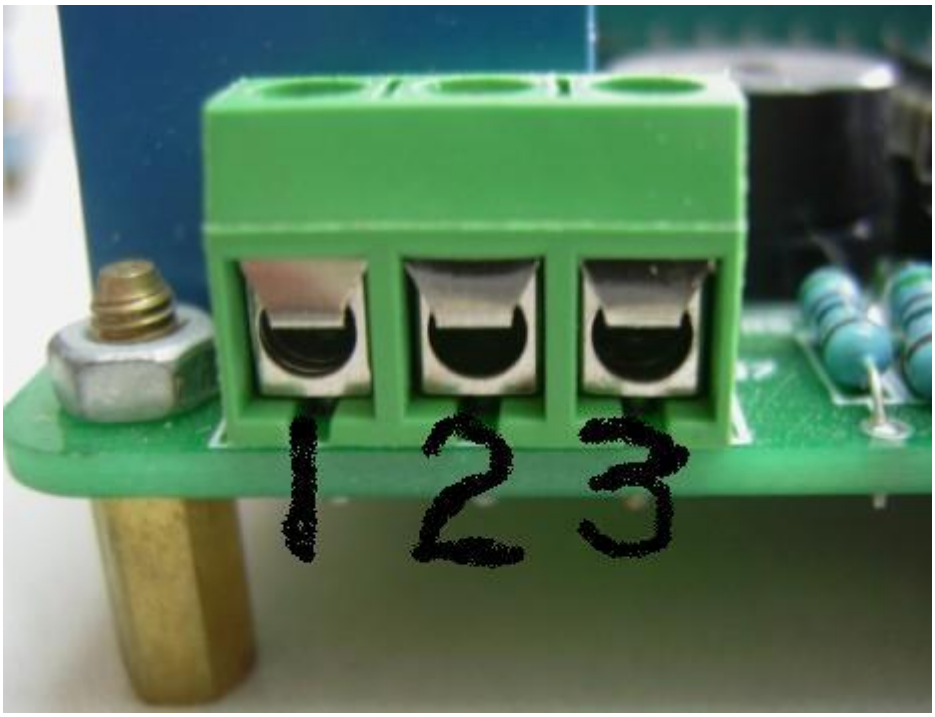
继电器是一种电子控制器件，它具有控制系统（又称输入回路）和被控制系统（又称输出回路），通常应用于自动控制电路中，它实际上是用较小的电流去控制较大电流的一种“自动开关”。故在电路中起着自动调节、安全保护、转换电路等作用。

电磁继电器的工作原理和特性

电磁式继电器一般由铁芯、线圈、衔铁、触点簧片等组成的。只要在线圈两端加上一定的电压，线圈中就会流过一定的电流，从而产生电磁效应，衔铁就会在电磁力吸引的作用下克服返回弹簧的拉力吸向铁芯，从而带动衔铁的动触点与静触点（常开触点）吸合。当线圈断电后，电磁的吸力也随之消失，衔铁就会在弹簧的反作用力返回原来的位置，使动触点与原来的静触点（常闭触点）吸合。这样吸合、释放，从而达到了在电路中的导通、切断的目的。对于继电器的“常开、常闭”触点，可以这样来区分：继电器线圈未通电时处于断开状态的静触点，称为“常开触点”；处于接通状态的静触点称为“常闭触点”。



上图为继电器内部连接图

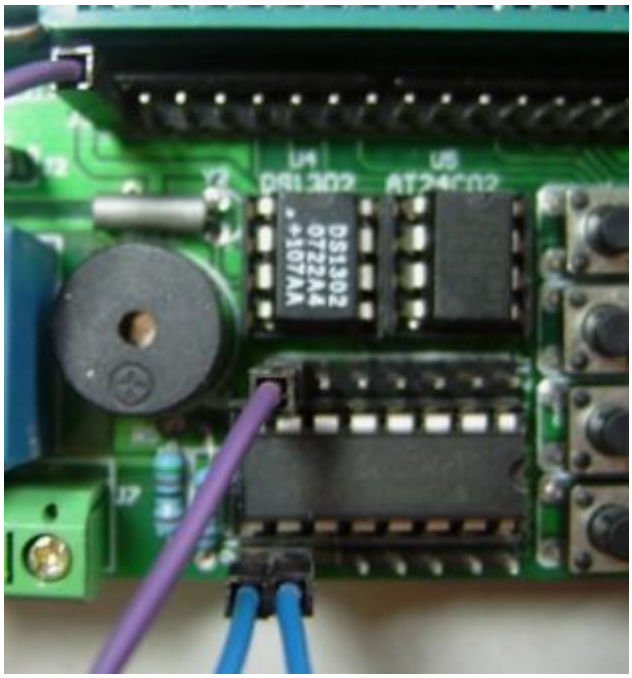


上图为座子与继电器的对应。2 为常闭端口，3 为常开端口。

注意：做该实验时，请务必弄清继电器原理及该如何连接。

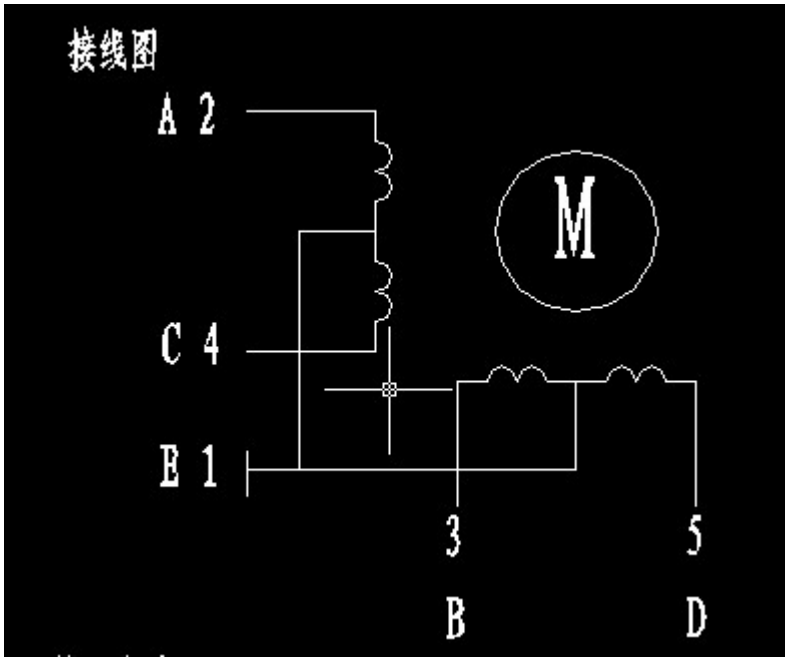
2.2.18 直流电机

直流电机内部阻值很小，电机只有 10 欧电阻，我们设计电路时，用了两个 5 欧电阻。这样，通过直流电机的电流都有 250 毫安左右，电流是比较大的。如果是电脑 USB 供电，不建议长期让直流电机工作。



上两图为直流电机的连接图。
我们用 PWM 来控制直流电机的转速。

2.2.19 步进电机



步进电机的内部连线图。

型号	电压 V DC	相电压 Ω (±5%)	步距角度	减速比	旋转方向	空载起动频率 PPS	起动转矩(at 100PPS) g.cm	绝缘介电强度
2486148-405-B1	5	50±3%	5.625/64	1: 64	CCW	>700	>500	at 600V 1s

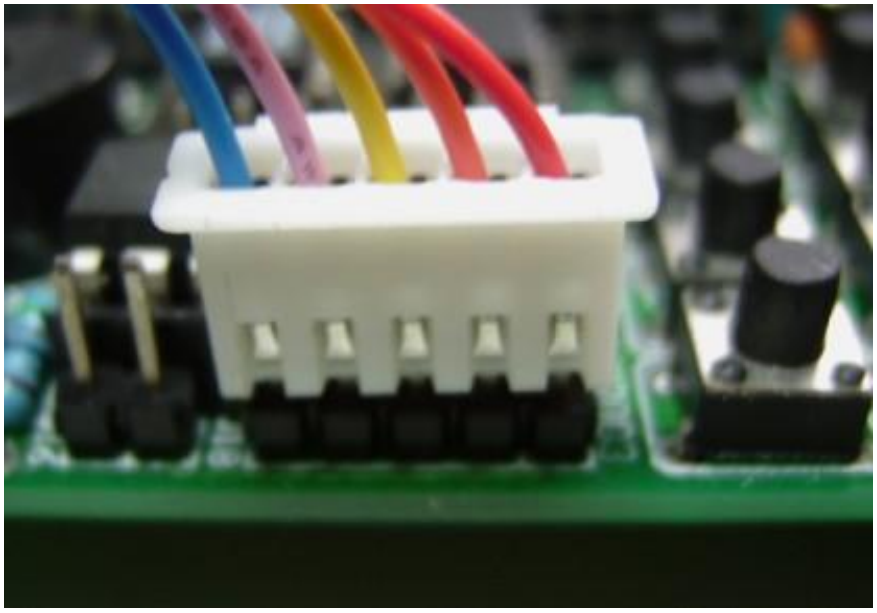
我们使用的步进电机的技术参数。

驱动方式: (4-1-2)

端子位号	导线颜色	1	2	3	4	5	6	7	8
1	红色B	+	+	+	+	+	+	+	+
2	橙色A	-	-						-
3	黄色B		-	-	-				
4	粉色C				-	-	-		
5	蓝色D						-	-	-

CCW方向旋转（轴伸端视）

上图为驱动方式，我们写程序就参考上图时序。



以上两图就是步进电机的连线图。

2.2.20 AT24C02

在很多电子设备中都有要随时存取数据作为历史记录或标志位。目前常用的存储器有24CXX 系列和93CXX 系列，前者是 I2C 总线结构，后者是 SPI 总线结构，本小节先介绍 I2C 结构的 EEPROM（24CXX）作用方法，在后面小节中再介绍 SPI 结构的 EEPROM（93CXX）使用方法。

I2C 总线基本概念

I2C 总线，是 INTERINTEGRATEDCIRCUITBUS 的缩写，即“内部集成电路总线”。I2C 总线是 Philips 公司推出的一种双向二线制总线。目前 Philips 公司和其它集成电路制造商推出了很多基于 I2C 总线的外围器件。I2C 总线包括一条数据线（SDA）和一条时钟线（SCL）。协议允许总线接入多个器件，并支持多主工作。总线中的器件既可以作为主控器也可以作为被控器，既可以是发送器也可以是接收器。总线按照一定的通信协议进行数据交换。在每次数据交换开始，作为主控器的器件需要通过总线竞争获得主控权，并启动一次数据交换。系统中各个器件都具有唯一的地址，各器件之间通过寻址确定数据接收方。

I2C 总线的系统结构

一个典型的 I2C 总线标准的 IC 器件，其内部不仅有 I2C 接口电路，还可将内部各单元电路划分成若干相对独立的模块，它只有二根信号线，一根是双向的数据线 SDA，另一根是时钟线 SCL。CPU 可以通过指令对各功能模块进行控制。。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，所以每个电路和模块都有唯一的地址，在信息的传输过程中，I 2 C 总线上并接的每一模块电路既是主控器（或被控器），又是发送器（或接收器）。CPU 发出的控制信号分为地址码和控制量（数据）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 总线接口电路如下图2所示。

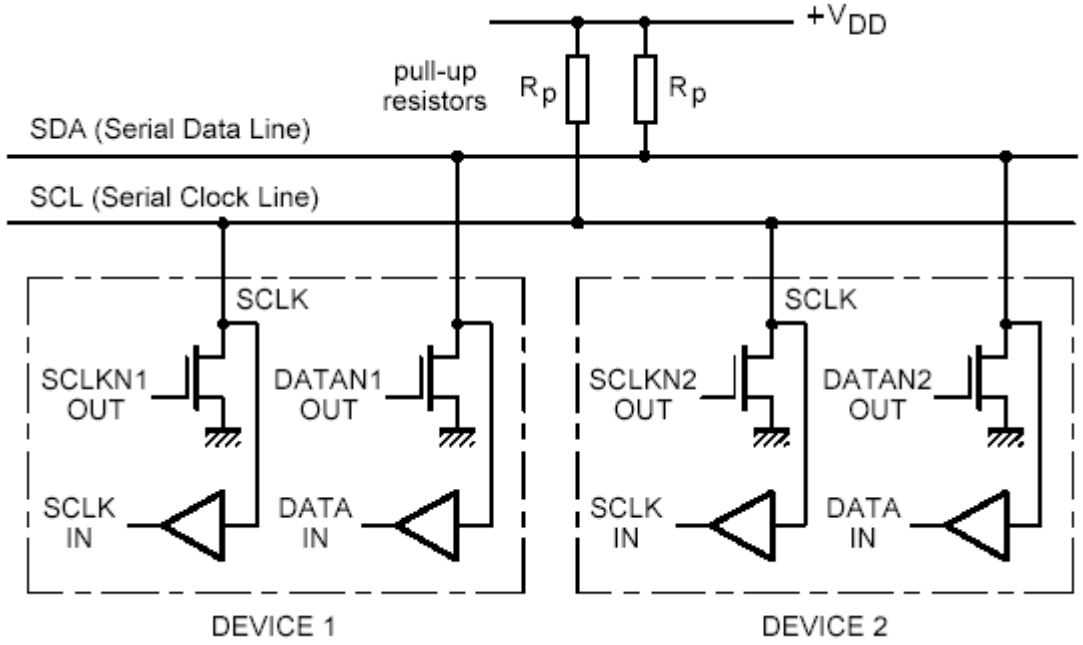


图2 I2C 总线接口电路图

I2C 总线的器件分为主器件和从器件。主器件的功能是启动在总线上传送数据，并产生时钟脉冲，以允许与被寻址的器件进行数据传送。被寻址的器件，称为从器件。一般来讲，任何器件均可以成为从器件，只有微控制器才能称为主器件。主、从器件对偶出现，工作在接收还是发送数据方式，由器件的功能和数据传送方向所决定。

I2C 总线允许连接多个微控制器，显然不能同时存在两个主器件，先控制总线的器件成为主器件，这就是总线竞争。在竞争过程中数据不会被破坏、丢失。数据只能在主、从器件中传送，结束后，主、从器件将释放总线，退出主、从器件角色。

I2C 总线接口特性

传统的单片机串行接口的发送和接收一般都分别各用一条线，如 MCS-51系列的 TXD 和 RXD，而 I2C 总线则根据器件的功能通过软件程序使其工作于发送或接收方式。当某个器件向总线上发送信息时，它就是发送器（也叫主器件），而当其从总线上接收信息时，又成为接收器（也叫从器件）。主器件用于启动总线上传送数据并产生时钟以开放传送的器件，此时任何被寻址的器件均被认为是从器件。I2C 总线的控制完全由挂在总线上的主器件送出的地址和数据决定，在总线上，既没有中心机也没有优先级。

总线上主和从（即发送和接收）的关系取决于此时数据传送的方向。SDA 和 SCL 都是双向线路，都通过一个电流源或上拉电阻连接到电源端。连接总线器件的输出级必须是集电极或漏极开路，以具有线“与”功能，当总线空闲时，两根线都是高电平。I2C 总线上数据的传输速率在标准模式下可达100kbit/s 在快速模式下可达400kbit/s 在高速模式下可达3.4Mbit/s 连接到总线的接口数量只由总线电容是400pF 的限制决定。

I2C 总线器件工作原理及时序

I2C 总线的时钟信号

在 I2C 总线上传送信息时的时钟同步信号是由挂接在 SCL 时钟线上的所有器件的逻辑“与”完成的。SCL 线上由高电平到低电平的跳变将影响到这些器件，一旦某个器件的时钟信号变为低电平，将使 SCL 线上所有器件开始并保护低电平期。此时，低电平周期短的器件的时钟由低至高的跳变并不影响 SCL 线的状态，这些器件将进入高电平等待的状态。

当所有器件的时钟信号都变为高电平时，低电平期结束，SCL 线被释放返回高电平，即所有的器件都同时开始它们的高电平期。其后，第一个结束高电平期的器件又将 SCL 线拉成低电平。这样就在 SCL 线上产生一个同步时钟。可见，时钟低电平时间由时钟低电平期最长的器件决定，而时钟高电平时间由时钟高电平期最短的器件决定。

I2C 总线的传输协议与数据传送

起始和停止条件

在数据传送过程中，必须确认数据传送的开始和结束。在 I2C 总线技术规范中，开始和结束信号（也称启动和停止信号）的定义如图3所示。

开始信号：当时钟总线 SCL 为高电平时，数据线 SDA 由高电平向低电平跳变，开始传送数据。

结束信号：当 SCL 线为高电平时，SDA 线从低电平向高电平跳变，结束传送数据。

开始和结束信号都是由主器件产生。在开始信号以后，总线即被认为处于忙状态，其它器件不能再产生开始信号。主器件在结束信号以后退出主器件角色，经过一段时间过，总线被认为是空闲的。

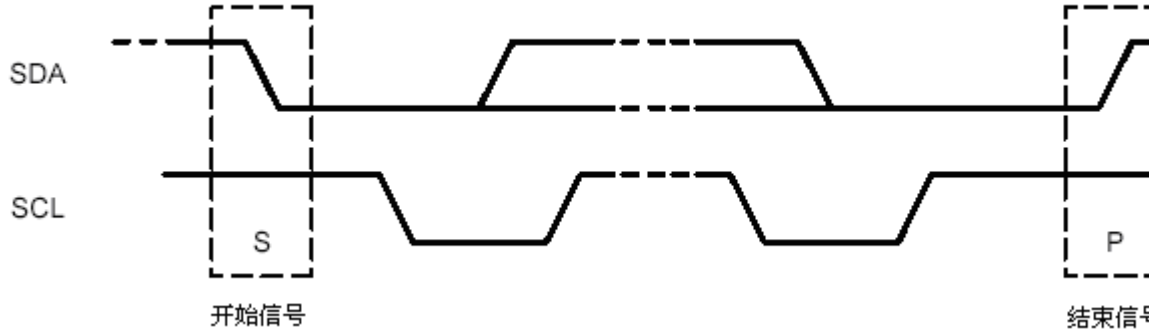


图3超始和停止信号图

数据格式

I2C 总线数据传送采用时钟脉冲逐位串行传送方式，在 SCL 的低电平期间，SDA 线上高、低电平能变化，在高电平期间，SDA 上数据必须保护稳定，以便接收器采样接收，时序如图4所示。

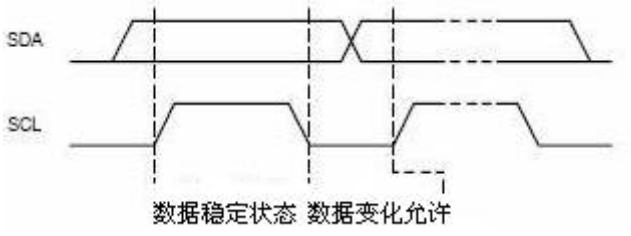


图4 数据传送时序图

I2C 总线发送器送到 SDA 线上的每个字节必须为8位长，传送时高位在前，低位在后。与之对应，主器件在 SCL 线上产生8个脉冲；第9个脉冲低电平期间，发送器释放 SDA 线，接收器把 SDA 线拉低，以给出一个接收确认位；第9个脉冲高电平期间，发送器收到这个确认位然后开始下一字节的传送，下一个字节的第一个脉冲低电平期间接收器释放 SDA。每个字节需要9个脉冲，每次传送的字节数是不受限制的。

I2C 总线的数据传送格式是在 I2C 总线开始信号后，送出的第一字节数据是用来选择从器件地址的，其中前7位为地址码，第8位为方向位（R/W）。方向位为“0”表示发送，即主器件把信息写到所选择的从器件中；方向位为“1”表示主器件将从从器件读信息。格式如下：

1 0 1 0 A2 A1 A0 R/W

注：前四位固定为1010。

开始信号后，系统中的各个器件将自己的地址和主器件送到总线上的地址进行比较，如果与主器件发送到总线上的地址一致，则该器件即被主器件寻址的器件，其接收信息还是发送信息则由第8位（R/W）决定。发送完第一个字节后再开始发数据信号。

响应

数据传输必须带响应。相关的响应时钟脉冲由主机产生，当主器件发送完一字节的数据后，接着发出对应于 SCL 线上的一个时钟（ACK）认可位，此时钟内主器件释放 SDA 线，一字节传送结束，而从器件的响应信号将 SDA 线拉成低电平，使 SDA 在该时钟的高电平期间为稳定的低电平。从器件的响应信号结束后，SDA 线返回高电平，进入下一个传送周期。

通常被寻址的接收器在接收到的每个字节后必须产生一个响应。当从机不能响应从机地址时，从机必须使数据线保持高电平，主机然后产生一个停止条件终止传输或者产生重复起始条件开始新的传输。如果从机接收器响应了从机地址但是在传输了一段时间后不能接收更多数据字节，主机必须再一次终止传输。这个情况用从机在第一个字节后没有产生响应来表示。从机使数据线保持高电平主机产生一个停止或重复起始条件。完整的数据传送过程如图5所示。

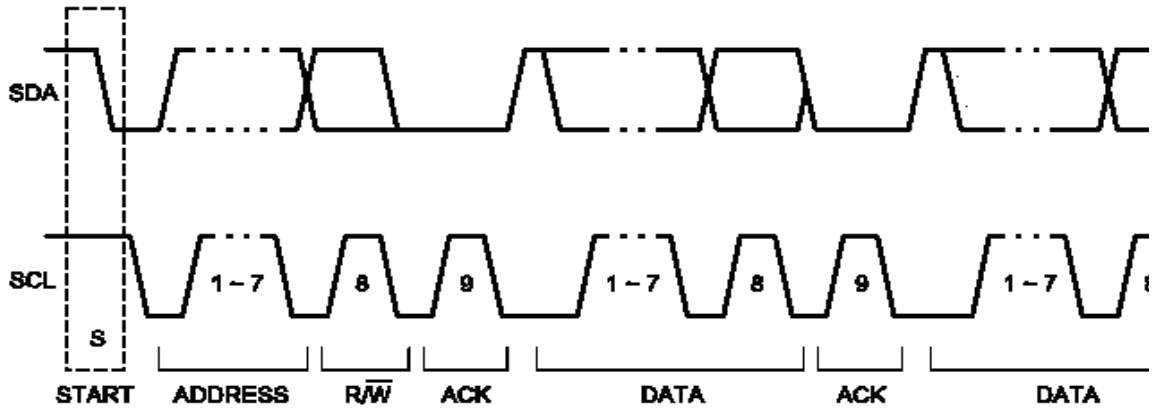


图5 完整的数据传送过程

I2C 总线还具有广播呼叫地址用于寻址总线上所有器件的功能。若一个器件不需要广播呼叫寻址中所提供的任何数据，则可以忽略该地址不作响应。如果该器件需要广播呼叫寻址中按需提供的数据，则应对地址作出响应，其表现为一个接收器。

24C 系列存储器的软硬件设计实例

IIC 总线常用的芯片有24C01、24C02、24C04、24C08等，下面以目前在单片机系统中常用的带 I2C 接口的 EEPROM 芯片 AT24C02为例，介绍 I2C 器件的基本应用，不同型号的24C 芯片，只是容量大小不同，读写方式与基本原理几乎一样。

AT24C02简介

AT24C02是美国 ATMEL 公司的低功耗 CMOS 串行 EEPROM，它是内含256×8位存储空间，具有工作电压宽（2.5～5.5V）、擦写次数多（大于10000次）、写入速度快（小于10ms）等特点。AT24C02中带有片内寻址寄存器。每写入或读出一个数据字节后，该地址寄存器自动加1，以实现对下一个存储单元的操作。所有字节都以单一操作方式读取。为降低总的写入时间，一次操作可写入多达8字节的数据。图6为 AT24C 系列芯片的封装图。各引脚功能如下：

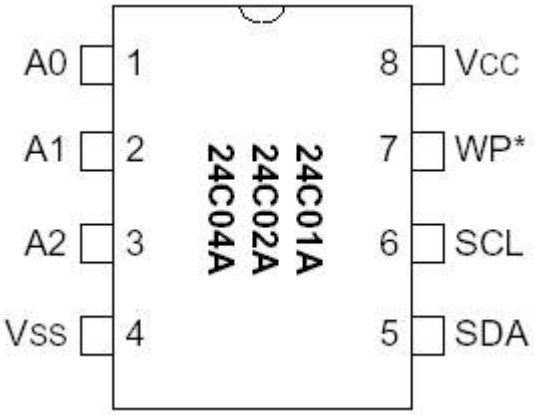


图6 24C 系列芯片封装图

SCL：串行时钟。在该引脚的上升沿时，系统将数据输入到每个 EEPROM 器件，在下降沿时输出。
SDA：串行数据。该引脚为开漏极驱动，可双向传送数据。
A0、A1、A2：器件/页面寻址。为器件地址输入端。
WP：硬件写保护。当该引脚为高电平时禁止写入，当为低电平时可正常读写数据。
VCC：电源。一般输入+5V 电压。
VSS：接地。

```
/*****************************************************************
* 名称 : flash()
* 功能 : 延时,时间为 2 个 NOP，大概为 2US
* 输入 : 无
* 输出 : 无
*****************************************************************/
void flash(void)
{
    _nop_();
    _nop_();
}

/*****************************************************************
* 名称 : x24c02_init()
* 功能 : 24c02 初始化子程序
* 输入 : 无
* 输出 : 无
*****************************************************************/
void x24c02_init(void)
{
    scl = 1;
    flash();
    sda = 1;
    flash();
}

/*****************************************************************
* 名称 : start(void)
* 功能 : 启动 I2C 总线
* 输入 : 无
* 输出 : 无
*****************************************************************/
void start(void)
{
    sda = 1;
    flash();
```

```
    scl = 1;
    flash();
    sda = 0;
    flash();
    scl = 0;
    flash();
}

/*****************************************************************
* 名称 : stop()
* 功能 : 停止 I2C 总线
* 输入 : 无
* 输出 : 无
*****************************************************************/
void stop()
{
    sda = 0;
    flash();
    scl = 1;
    flash();
    sda = 1;
    flash();
}

/*****************************************************************
* 名称 : writex()
* 功能 : 写一个字节
* 输入 : j（需要写入的值）
* 输出 : 无
*****************************************************************/
void writex(uchar j)
{
    uchar i,temp;
    temp = j;
    for(i=0; i<8; i++)
    {
        temp = temp << 1;
        scl = 0;
        flash();
        sda = CY;
        flash();
        scl = 1;
        flash();
    }
```

```
    scl = 0;
    flash();
    sda = 1;
    flash();
}

/*****************************************************************
* 名称 : readx()
* 功能 : 读一个字节
* 输入 : 无
* 输出 : 读出的值
*****************************************************************/
uchar readx(void)
{
    uchar i, j, k = 0;
    scl = 0;
    flash();
    sda = 1;
    for(i=0; i<8; i++)
    {
        flash();
        scl = 1;
        flash();
        if(sda == 1)
        {
            j = 1;
        }
        else j = 0;
        k = (k << 1) | j;
        scl = 0;
    }
    flash();
    return(k);
}

/*****************************************************************
* 名称 : clock()
* 功能 : I2C 总线时钟
* 输入 : 无
* 输出 : 无
*****************************************************************/
void clock(void)
{
    uchar i = 0;
```

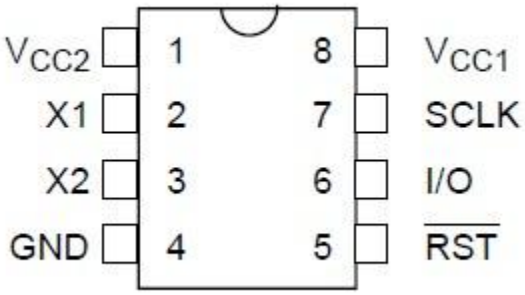
```
    scl = 1;
    flash();
    while((sda == 1) && (i < 255))
    {
        i++;
    }
    scl = 0;
    flash();
}

/*****************************************************************
* 名称 : x24c02_read()
* 功能 : 从 24c02 中读出值
* 输入 : address(要在这个地址读取值)
* 输出 : 从 24c02 中读出的值
*****************************************************************/
uchar x24c02_read(uchar address)
{
    uchar i;
    start();
    writex(0xa0);
    clock();
    writex(address);
    clock();
    start();
    writex(0xa1);
    clock();
    i = readx();
    stop();
    delay1(10);
    return(i);
}

/*****************************************************************
* 名称 : x24c02_write()
* 功能 : 想 24c02 中写入数据
* 输入 : address(地址) , info(值)
* 输出 : 无
*****************************************************************/
void x24c02_write(uchar address, uchar info)
{
    EA = 0;
    start();
    writex(0xa0);
```

```
    clock();
    writex(address);
    clock();
    writex(info);
    clock();
    stop();
    EA = 1;
    delay1(50);
}
```

2.2.21 时钟芯片 DS1302

DS1302 是美国 DALLAS 公司推出的一种高性能、低功耗、带 RAM 的实时时钟电路，它可以对年、月、日、周日、时、分、秒进行计时，具有闰年补偿功能，工作电压为2.5V～5.5V。采用三线接口与 CPU 进行同步通信，并可采用突发方式一次传送多个字节的时钟信号或 RAM 数据。DS1302内部有一个31×8的用于临时性存放数据的 RAM 寄存器。DS1302是 DS1202的升级产品，与 DS1202兼容，但增加了主电源/后背电源双电源引脚，同时提供了对后背电源进行涓细电流充电的能力。

2.1 引脚功能及结构



其中 Vcc1为后备电源，VCC2为主电源。在主电源关闭的情况下，也能保持时钟的连续运行。DS1302由 Vcc1或 Vcc2两者中的较大者供电。当 Vcc2大于 Vcc1＋0.2V 时，Vcc2给 DS1302供电。当 Vcc2小于 Vcc1时，DS1302由 Vcc1供电。X1和 X2是振荡源，外接32.768kHz 晶振。RST 是复位/片选线，通过把 RST 输入驱动置高电平来启动所有的数据传送。RST 输入有两种功能：首先，RST 接通控制逻辑，允许地址/命令序列送入移位寄存器；其次，RST 提供终止单字节或多字节数据的传送手段。当 RST 为高电平时，所有的数据传送被初始化，允许对 DS1302进行操作。如果在传送过程中 RST 置为低电平，则会终止此次数据传送，I/O 引脚变为高阻态。上电运行时，在 Vcc≥2.5V 之前，RST 必须保持低电平。只有在 SCLK 为低电平时，才能将 RST 置为高电平。I/O 为串行数据输入输出端(双向)，后面有详细说明。SCLK 始终是输入端。

```
/*************************************************************************
* 名称 :  v_RTInputByte（）
```

```
* 功能 :  往 DS1302写入1Byte 数据
* 输入 :  ucDa 写入的数据
* 输出 :  无
*****************************************************************************/
void v_RTInputByte(uchar ucDa)
{
    uchar i;
    ACC = ucDa;
    T_RST = 1;
    for(i=8; i>0; i--)
    {
        T_IO = ACC0;
        T_CLK = 1;
        T_CLK = 0;
        ACC = ACC >> 1;
    }
}

/*****************************************************************************
* 名称  :   uc_RTOutputByte ( )
* 功能  :   从 DS1302读取1Byte 数据
* 输入  :  无
* 返回值:   ACC
*****************************************************************************/
uchar uc_RTOutputByte(void)
{
    uchar i;
    T_RST = 1;
    for(i=8; i>0; i--)
    {
        ACC = ACC >>1;
        T_IO=1;
        ACC7 = T_IO;
        T_CLK = 1;
        T_CLK = 0;
    }
    return(ACC);
}

/*****************************************************************************
* 名称   :   v_W1302(uchar ucAddr, uchar ucDa)
* 功能   :   往 DS1302写入数据
* 输入   :   ucAddr: DS1302地址, ucDa:  要写的数据
* 返回值 :   无
```

```
*****************************************************************************/
void v_W1302(uchar ucAddr, uchar ucDa)
{
    T_RST = 0;
    T_CLK = 0;
    T_RST = 1;
    v_RTInputByte(ucAddr);     // 写地址
    _nop_();
    _nop_();
    v_RTInputByte(ucDa);       // 写1Byte 数据
    T_CLK = 1;
    T_RST = 0;
}

/*****************************************************************************
* 名称    :  uc_R1302(uchar ucAddr)
* 功能    :  读取 DS1302某地址的数据
* 输入    :  ucAddr: DS1302地址
* 返回值 :  ucDa :读取的数据
*****************************************************************************/
uchar uc_R1302(uchar ucAddr)
{
    uchar ucDa;
    T_RST = 0;
    T_CLK = 0;
    T_RST = 1;
    v_RTInputByte(ucAddr);      //写地址，命令
    _nop_();
    _nop_();
    ucDa = uc_RTOutputByte(); //读1Byte 数据
    T_CLK = 1;
    T_RST = 0;
    return(ucDa);
}

/*****************************************************************************
* 名称    :  v_BurstW1302T
* 功能    :  往 DS1302写入时钟数据(多字节方式)
* 输入    :  pSecDa: 时钟数据地址 格式为: 秒 分 时 日 月 星期 年 控制
*                          8Byte (BCD 码) 1B 1B 1B 1B 1B 1B   1B   1B
* 返回值 :  无
*****************************************************************************/
void v_BurstW1302T(uchar *pSecDa)
{
```

```
    uchar i;
    v_W1302(0x8e, 0x00);                //控制命令,WP=0,写操作
    T_RST = 0;
    T_CLK = 0;
    T_RST = 1;
    v_RTInputByte(0xbe);                //0xbe:时钟多字节写命令
    for(i=8; i>0; i--)                  //8Byte = 7Byte 时钟数据 + 1Byte 控制
    {
        v_RTInputByte(*pSecDa);     //写1Byte 数据
        pSecDa++;
    }
    T_CLK = 1;
    T_RST = 0;
}

/*********************************************************************
* 名称    :  v_BurstR1302T(uchar *pSecDa)
* 功能    :  读取 DS1302时钟数据
* 输入    :  pSecDa: 时钟数据地址 格式为: 秒 分 时 日 月 星期 年
*                          7Byte (BCD 码) 1B 1B 1B 1B 1B 1B    1B
* 返回值 :  ucDa :读取的数据
*********************************************************************/
void v_BurstR1302T(uchar *pSecDa)
{
    uchar i;
    T_RST = 0;
    T_CLK = 0;
    T_RST = 1;
    v_RTInputByte(0xbf);                    //0xbf:时钟多字节读命令
    for(i=8; i>0; i--)
    {
        *pSecDa = uc_RTOutputByte();    //读1Byte 数据
        pSecDa++;
    }
    T_CLK = 1;
    T_RST = 0;
}

/*********************************************************************
* 名称    :  v_BurstW1302R(uchar *pReDa)
* 功能    :  往 DS1302寄存器数写入数据(多字节方式)
* 输入    :  pReDa: 寄存器数据地址
* 返回值 :  无
*********************************************************************/
```

```
void v_BurstW1302R(uchar *pReDa)
{
    uchar i;
    v_W1302(0x8e,0x00);              //控制命令,WP=0,写操作
    T_RST = 0;
    T_CLK = 0;
    T_RST = 1;
    v_RTInputByte(0xfe);          //0xbe:时钟多字节写命令
    for(i=31; i>0; i--)           //31Byte  寄存器数据
    {
        v_RTInputByte(*pReDa); //写1Byte 数据
        pReDa++;
    }
    T_CLK = 1;
    T_RST = 0;
}

/*********************************************************************
* 名称    :  v_BurstR1302R(uchar *pReDa)
* 功能    :  读取 DS1302寄存器数据
* 输入    :  pReDa:  寄存器数据地址
* 返回值 :  无
*********************************************************************/
void v_BurstR1302R(uchar *pReDa)
{
    uchar i;
    T_RST = 0;
    T_CLK = 0;
    T_RST = 1;
    v_RTInputByte(0xff);                //0xbf:时钟多字节读命令
    for(i=31; i>0; i--)                 //31Byte  寄存器数据
    {
        *pReDa = uc_RTOutputByte(); //读1Byte 数据
        pReDa++;
    }
    T_CLK = 1;
    T_RST = 0;
}

/*********************************************************************
* 名称   :  v_Set1302(uchar *pSecDa)
* 功能   :  设置初始时间
* 输入   :  pSecDa:  初始时间地址。初始时间格式为:  秒  分  时  日  月  星期  年
*                                    7Byte (BCD 码) 1B 1B 1B 1B 1B 1B   1B
```

```
* 返回值: 无
*********************************************************************/
void v_Set1302(uchar *pSecDa)
{
    uchar i;
    uchar ucAddr = 0x80;
    v_W1302(0x8e, 0x00);                //控制命令,WP=0,写操作
    for(i=7; i>0; i--)
    {
        v_W1302(ucAddr, *pSecDa);    // 秒 分 时 日 月 星期 年
        pSecDa++;
        ucAddr += 2;
    }
    v_W1302(0x8e, 0x80);                //控制命令,WP=1,写保护
}

/*********************************************************************
* 名称    : v_Get1302(uchar ucCurtime[])
* 功能    : 读取 DS1302当前时间
* 输入    : ucCurtime: 保存当前时间地址。当前时间格式为: 秒 分 时 日 月
星期 年
*                                                  7Byte (BCD 码) 1B 1B 1B 1B 1B
1B   1B
* 返回值 : 无
*********************************************************************/
void v_Get1302(uchar ucCurtime[])
{
    uchar i;
    uchar ucAddr = 0x81;
    for(i=0; i<7; i++)
    {
        ucCurtime[i] = uc_R1302(ucAddr);      //格式为: 秒 分 时 日 月 星期
年
        ucAddr += 2;
    }
}
```

后面部分是 BCD 码和 DEC 码的相互转换。

```
/*********************************************************************
* 名称 : dectobcd(uchar dec)
* 功能 : DEC 码转换为 BCD 码
* 输入 : dec 码
* 输出 : bcd 码
*********************************************************************/
```

```
uchar dectobcd(uchar dec)
{
    uchar bcd;
    bcd = 0;
    while(dec >= 10)
    {
        dec -= 10;
        bcd++;
    }
    bcd <<= 4;
    bcd |= dec;
    return bcd;
}

/*********************************************************************
* 名称 : bcdtodec(uchar bcd)
* 功能 : BCD 码转换为 DEC 码
* 输入 : bcd 码
* 输出 : dec 码
*********************************************************************/
uchar bcdtodec(uchar bcd)
{
    uchar data1;
    data1 = bcd & 0x0f;       //取 BCD 低4位
    bcd = bcd & 0x70;         //剔除 BCD 的最高位和低4位。
    data1 += bcd >> 1;
    data1 += bcd >> 3;        //用位移代替乘法运算
    return data1;
}
```

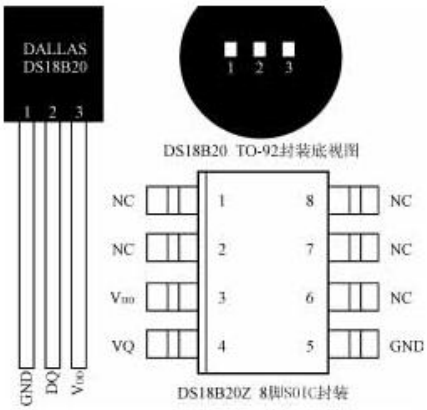
2.2.22 温度传感器 DS18B20

单总线温度传感器 DS18B20 简介

DS18B20 是 DALLAS 公司生产的单总线式数字温度传感器，它具有微型化、低功耗、高性能、搞干扰能力强、易配处理器等优点，特别适用于构成多点温度测控系统，可直接将温度转化成串行数字信号（提供 9 位二进制数字）给单片机处理，且在同一总线上可以挂接多个传感器芯片。它具有 3 引脚 TO－92 小体积封装形式，温度测量范围为－55℃～＋125℃，可编程为 9 位～12 位 A/D 转换精度，测温分辨率可达 0.0625℃，被测温度用符号扩展的 16 位数字量方式串行输出，其工作电源既可在远端引入，也可采用寄生电源方式产生，多个 DS18B20 可以并联到 3 根或 2 根线上，CPU 只需一根端口线就能与多个 DS18B20 通信，占用微处理器的端口较少，可节省大量的引线和逻辑电路。以上特点使 DS18B20 非常适用于远距离多点温度检测系统。

DS18B20 外形及引脚说明

外形及引脚如图 2 所示：



在 TO-92和 SO-8的封装中引脚有所不同，具体差别请查阅 PDF 手册，在 TO-92 封装中引脚分配如下：
1（GND）：地
2（DQ）：单线运用的数据输入输出引脚
3（VDD）：可选的电源引脚

DS18B20工作过程及时序

DS18B20内部的低温度系数振荡器是一个振荡频率随温度变化很小的振荡器，为计数器1提供一频率稳定的计数脉冲。

高温度系数振荡器是一个振荡频率对温度很敏感的振荡器，为计数器2提供一个频率随温度变化的计数脉冲。

初始时，温度寄存器被预置成-55℃，每当计数器1从预置数开始减计数到0时，温度寄存器中寄存的温度值就增加1℃，这个过程重复进行，直到计数器2计数到0时便停止。

初始时，计数器1预置的是与-55℃相对应的一个预置值。以后计数器1每一个循环的预置数都由斜率累加器提供。为了补偿振荡器温度特性的非线性性，斜率累加器提供的预置数也随温度相应变化。计数器1的预置数也就是在给定温度处使温度寄存器寄存值增加1℃计数器所需要的计数个数。

DS18B20内部的比较器以四舍五入的量化方式确定温度寄存器的最低有效位。在计数器2停止计数后，比较器将计数器1中的计数剩余值转换为温度值后与0.25℃进行比较，若低于0.25℃，温度寄存器的最低位就置0；若高于0.25℃，最低位就置1；若高于0.75℃时，温度寄存器的最低位就进位然后置0。这样，经过比较后所得的温度寄存器的值就是最终读取的温度值了，其最后位代表0.5℃，四舍五入最大量化误差为 ± 1/2LSB，即0.25℃。

温度寄存器中的温度值以9位数据格式表示，最高位为符号位，其余8位以二进制补码形式表示温度值。测温结束时，这9位数据转存到暂存存储器的前两个字节中，符号位占用第一字节，8位温度数据占据第二字节。

DS18B20测量温度时使用特有的温度测量技术。DS18B20内部的低温度系数振荡器能产生稳定的频率信号；同样的，高温度系数振荡器则将被测温度转换成

频率信号。当计数门打开时，DS18B20进行计数，计数门开通时间由高温度系数振荡器决定。芯片内部还有斜率累加器，可对频率的非线性度加以补偿。测量结果存入温度寄存器中。一般情况下的温度值应该为9位，但因符号位扩展成高8位，所以最后以16位补码形式读出。

DS18B20工作过程一般遵循以下协议：初始化——ROM 操作命令——存储器操作命令——处理数据

① 初始化

单总线上的所有处理均从初始化序列开始。初始化序列包括总线主机发出一复位脉冲，接着由从属器件送出存在脉冲。存在脉冲让总线控制器知道 DS1820 在总线上且已准备好操作。

② ROM 操作命令

一旦总线主机检测到从属器件的存在，它便可以发出器件 ROM 操作命令之一。所有 ROM 操作命令均为8位长。这些命令列表如下：

Read ROM(读 ROM)[33h]

此命令允许总线主机读 DS18B20的8位产品系列编码，唯一的48位序列号，以及8位的 CRC。此命令只能在总线上仅有一个 DS18B20的情况下可以使用。如果总线上存在多于一个的从属器件，那么当所有从片企图同时发送时将发生数据冲突的现象（漏极开路会产生线与的结果）。

Match ROM(符合 ROM)[55h]

此命令后继以64位的 ROM 数据序列，允许总线主机对多点总线上特定的 DS18B20寻址。只有与64位 ROM 序列严格相符的 DS18B20才能对后继的存贮器操作命令作出响应。所有与64位 ROM 序列不符的从片将等待复位脉冲。此命令在总线上有单个或多个器件的情况下均可使用。

Skip ROM(跳过 ROM)[CCh]

在单点总线系统中，此命令通过允许总线主机不提供64位 ROM 编码而访问存储器操作来节省时间。如果在总线上存在多于一个的从属器件而且在 Skip ROM 命令之后发出读命令，那么由于多个从片同时发送数据，会在总线上发生数据冲突（漏极开路下拉会产生线与的效果）。

Search ROM(搜索 ROM)[F0h]

当系统开始工作时，总线主机可能不知道单线总线上的器件个数或者不知道其64位 ROM 编码。搜索 ROM 命令允许总线控制器用排除法识别总线上的所有从机的64位编码。

Alarm Search(告警搜索)[ECh]

此命令的流程与搜索 ROM 命令相同。但是，仅在最近一次温度测量出现告警的情况下，DS18B20才对此命令作出响应。告警的条件定义为温度高于 TH 或低于 TL。只要 DS18B20一上电，告警条件就保持在设置状态，直到另一次温度测量显示出非告警值或者改变 TH 或 TL 的设置，使得测量值再一次位于允许的范围之内。贮存在 EEPROM 内的触发器值用于告警。

③存储器操作命令

Write Scratchpad（写暂存存储器）[4Eh]

这个命令向 DS18B20的暂存器中写入数据，开始位置在地址2。接下来写入的两个字节将被存到暂存器中的地址位置2和3。可以在任何时刻发出复位命令来中止写入。

Read Scratchpad（读暂存存储器）[BEh]

这个命令读取暂存器的内容。读取将从字节0开始，一直进行下去，直到第9（字节8，CRC）字节读完。如果不想读完所有字节，控制器可以在任何时间发出复位命令来中止读取。

Copy Scratchpad（复制暂存存储器）[48h]

这条命令把暂存器的内容拷贝到 DS18B20的 E^2存储器里，即把温度报警触发字节存入非易失性存储器里。如果总线控制器在这条命令之后跟着发出读时间隙，而 DS18B20又正在忙于把暂存器拷贝到 E^2存储器，DS18B20就会输出一个“0”，如果拷贝结束的话，DS18B20 则输出“1”。如果使用寄生电源，总线控制器必须在这条命令发出后立即起动强上拉并最少保持10ms。

Convert T（温度变换）[44h]

这条命令启动一次温度转换而无需其他数据。温度转换命令被执行，而后DS18B20保持等待状态。如果总线控制器在这条命令之后跟着发出读时间隙，而DS18B20又忙于做时间转换的话，DS18B20将在总线上输出“0”，若温度转换完成，则输出“1”。如果使用寄生电源，总线控制器必须在发出这条命令后立即起动强上拉，并保持500ms。

Recall E^2（重新调整 E^2）[B8h]

这条命令把贮存在 E^2中温度触发器的值重新调至暂存存储器。这种重新调出的操作在对 DS18B20上电时也自动发生，因此只要器件一上电，暂存存储器内就有了有效的数据。在这条命令发出之后，对于所发出的第一个读数据时间片，器件会输出温度转换忙的标识：“0”=忙，“1”=准备就绪。

Read Power Supply（读电源）[B4h]

对于在此命令发送至 DS18B20之后所发出的第一读数据的时间片，器件都会给出其电源方式的信号：“0”=寄生电源供电，“1”=外部电源供电。

④处理数据

DS18B20的高速暂存存储器由9个字节组成，其分配如图3所示。当温度转换命令发布后，经转换所得的温度值以二字节补码形式存放在高速暂存存储器的第0和第1个字节。单片机可通过单线接口读到该数据，读取时低位在前，高位在后。

图3 高速暂存存储器分配图

温度/^{0}C	二进制表示												十六进制表示
	符号位（5位）	数据位（11位）											
+125	0 0 0 0 0	1	1	1	1	1	0	1	0	0	0	0	07D0H
+25.0625	0 0 0 0 0	0	0	1	1	0	0	1	0	0	0	1	0191H
+10.125	0 0 0 0 0	0	0	0	1	0	1	0	0	0	1	0	00A2H
+0.5	0 0 0 0 0	0	0	0	0	0	0	0	1	0	0	0	0008H
0	0 0 0 0	0	0	0	0	0	0	0	0	0	0	0	0000H

	0												
-0.5	1 1 1 1 1	1	1	1	1	1	1	1	1	0	0	0	FFF8H
-10.125	1 1 1 1 1	1	1	1	0	1	0	1	1	1	1	0	FF5EH
-25.625	1 1 1 1 1	1	1	0	0	1	1	0	1	1	1	1	FE6FH
-55	1 1 1 1 1	1	0	0	1	0	0	1	0	0	0	0	FC90H

DS18B20温度数据表

上表是 DS18B20温度采集转化后得到的12位数据，存储在 DS18B20的两个8比特的 RAM 中，二进制中的前面5位是符号位，如果测得的温度大于或等于0，这5位为0，只要将测到的数值乘于0.0625即可得到实际温度；如果温度小于0，这5位为1，测到的数值需要取反加1再乘于0.0625即可得到实际温度。

温度转换计算方法举例：
例如当 DS18B20采集到+125℃的实际温度后，输出为07D0H，则：

实际温度=07D0H╳0.0625=2000╳0.0625=125℃。
例如当 DS18B20采集到-55℃的实际温度后，输出为 FC90H，则应先将11位数据位取反加1得370H（符号位不变，也不作为计算），则：
实际温度=370H╳0.0625=880╳0.0625=55℃。

实例程序分析：

```
/*************************************************************************
* 名称 : Reset()
* 功能 : 复位 DS18B20
* 输入 : 无
* 输出 : 无
*************************************************************************/
uchar Reset(void)
{
    uchar deceive_ready;
    DQ = 0;
    delay(29);
    DQ = 1;
    delay(3);
    deceive_ready = DQ;
    delay(25);
    return(deceive_ready);
}

/*************************************************************************
```

```
* 名称 : read_bit()
* 功能 : 从 DS18B20读一个位值
* 输入 : 无
* 输出 : 从 DS18B20读出的一个位值
*****************************************************************************/
uchar read_bit(void)
{
    uchar i;
    DQ = 0;
    DQ = 1;
    for(i=0; i<3; i++);
    return(DQ);
}

/*****************************************************************************
* 名称 : write_bit()
* 功能 : 向 DS18B20写一位
* 输入 : bitval（要对 DS18B20写入的位值）
* 输出 : 无
*****************************************************************************/
void write_bit(uchar bitval)
{
DQ=0;if(bitval==1)
DQ=1;
delay(5);
DQ=1;
}

/*****************************************************************************
* 名称 : read_byte()
* 功能 : 从 DS18B20读一个字节
* 输入 : 无
* 输出 : 从 DS18B20读到的值
*****************************************************************************/
uchar read_byte(void)
{
    uchar i,m,receive_data;
    m = 1;
    receive_data = 0;
    for(i=0; i<8; i++)
    {
        if(read_bit())
        {
            receive_data = receive_data + (m << i);
```

```
            }
            delay(6);
        }
        return(receive_data);
}

/*****************************************************************
* 名称 : write_byte()
* 功能 : 向 DS18B20写一个字节
* 输入 : val（要对 DS18B20写入的命令值）
* 输出 : 无
*****************************************************************/
void write_byte(uchar val)
{
        uchar i,temp;
        for(i=0; i<8; i++)
        {
            temp = val >> i;
            temp = temp & 0x01;
            write_bit(temp);
            delay(5);
        }
}
```

第三章 结尾

很感谢您使用本开发板，经过上面的学习，相信你已经熟练的掌握了单片机编程。如果真的要精通单片机，需要多年实践工作的积累，加油吧。

联系方式：
Email : ct-315@163.com
QQ：85536436
网站：www.dlmcu.net