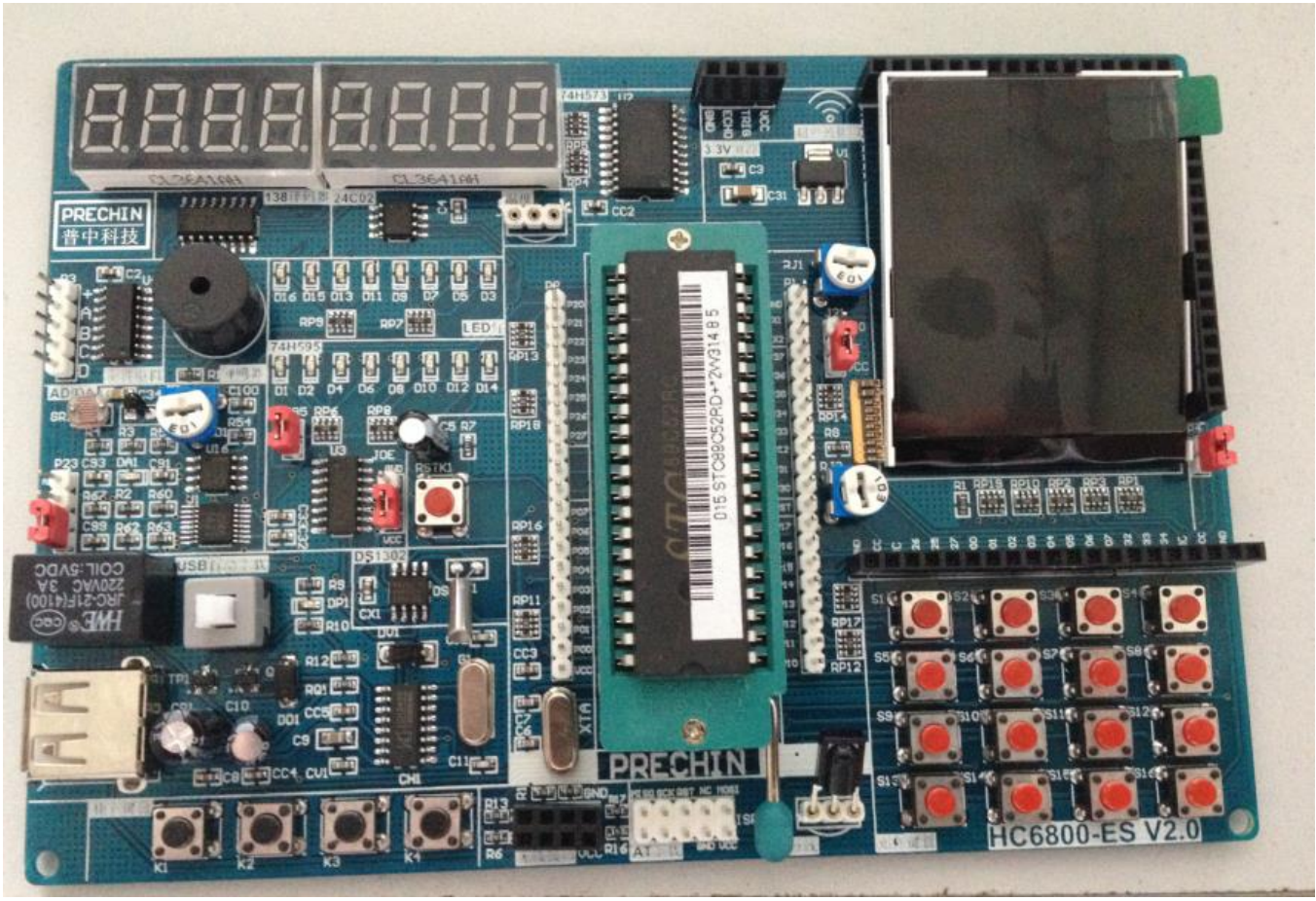
HC6800-ES V2.0 单片机开发板



学习指南

普中科技

目录

第一讲 开发板资源介绍

本开发板相对以往开发板的特点是综合性比较高、把短路冒去掉了 省去接线的麻烦更加方便了初学者、是一款性价比极高的产品，提供USB2.0和串口两种通信方式，USB实现供电、编程、仿真、通信多种功能，另外还提供了Atmel单片机的ISP接口。此板兼容STC、SST、Atmel、Philips等51家族的所有单片机。如果使用ISP编程建议使用开发板自带的单片机,因为每个厂烧录程序的方式不一样。HC6800开发板有着丰富的外部资源，

通过对该开发实验仪的学习，学员不仅可以轻松快速地掌握单片机软件系统的开发（C语言、汇编语言），而且还能快速掌握硬件电路的设计及嵌入系统开发流程。

本套件配有丰富的实例源码、原理图等，特别适合单片机初学者,大中专院校师生，单片机开发工程师选用，也是毕业设计和电子竞赛不可多得的参考板

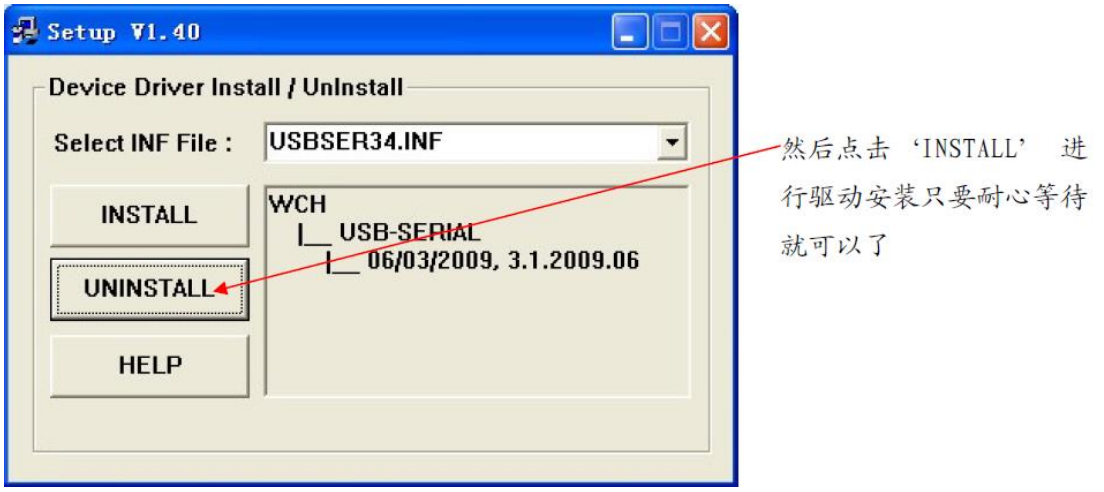
- 单片机采用 STC90c516 1280 SRAM 64K Flash
- 2.1 寸彩色液晶屏
- 超声波接口
- 1602 液晶屏接口
- 12864 液晶屏接口
- 温度传感器 DS18B20
- EEPROM 24C02 存储器
- 8 位动态数码管
- AD/DA 转换 XPT2046
- DS1302 实时时钟
- 4*4 矩阵键盘
- 4 个独立键盘
- 2*8 路 led 灯
- USB 接口，实现下载，供电，串行通信。通过 USB 转串口芯片 CH340T 转换
- IR 红外接收头，红外数据传输
- 板载继电器
- 蜂鸣器

- 步进电机 ULN2003 达林顿管驱动

第二讲 软件安装

1、USB 转串口驱动的安装

双击 USB 驱动 SERIAL 程序的 Setup



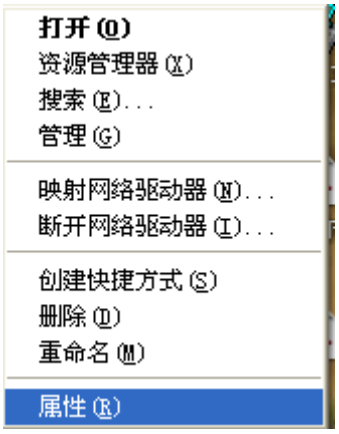
安装成功后,会出现一下提示画面

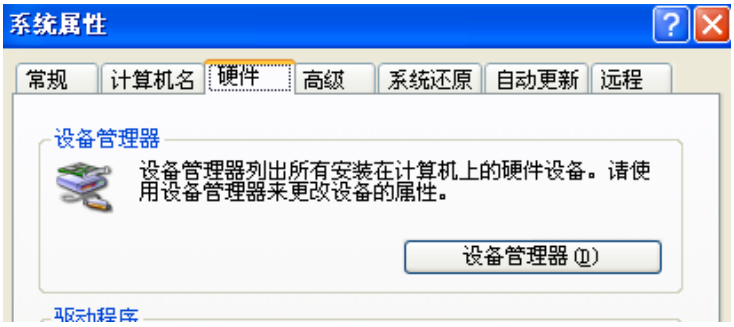
点“确定”即可以结束安装过程。

安装完驱动程序成功后,把开发板与电脑连接可以看到设备管理器中的 COM 口。

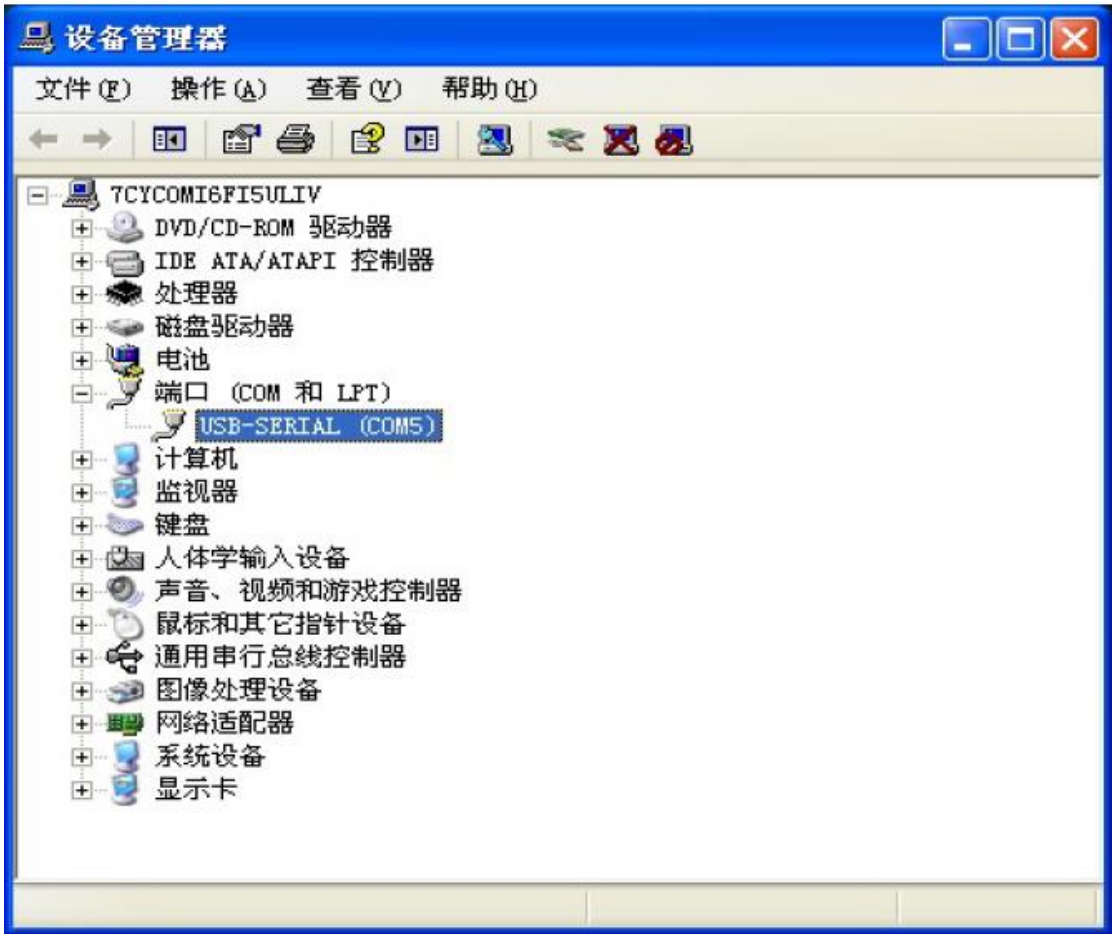
在图标“我的电脑”右键，→属性，→硬件，→设备管理器（这是 XP 系统，WIN

系统与其相似找到设备管理器）





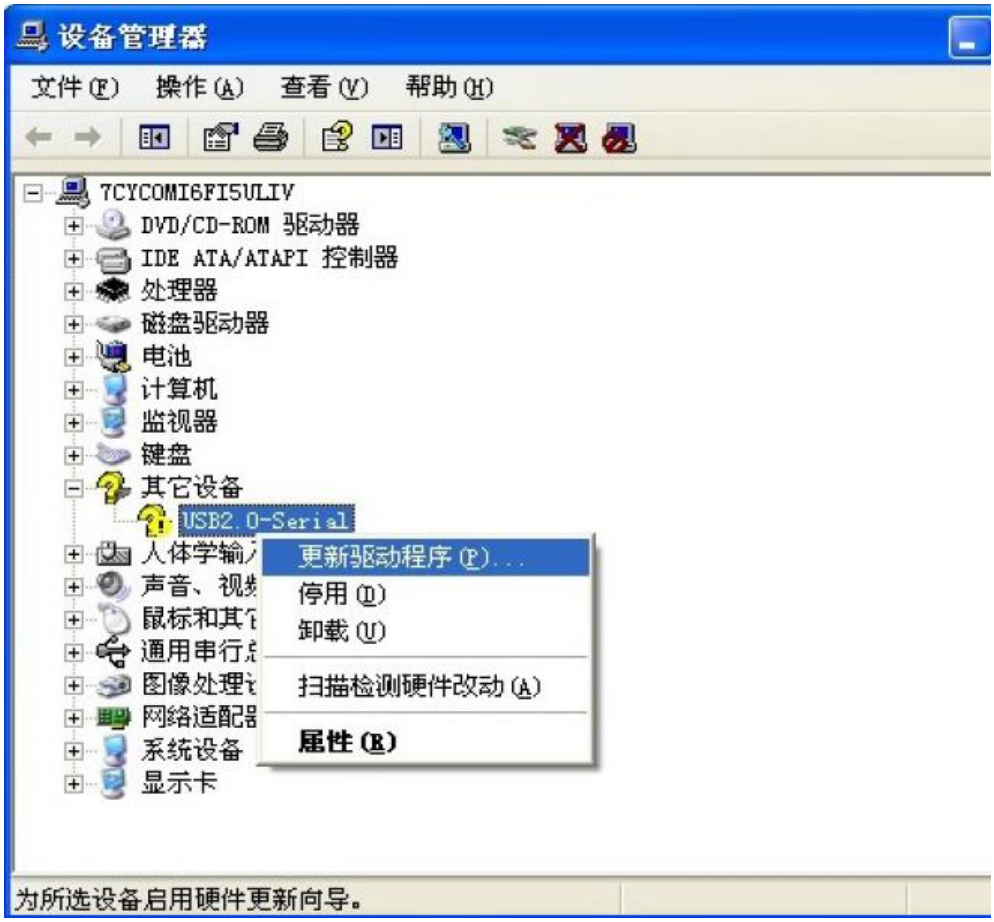
显示成功的 COM 口



安装不成功解决办法

如果发现是下图感叹号， 则没有安装成功或者没有安装、右键点击更新驱动程

序



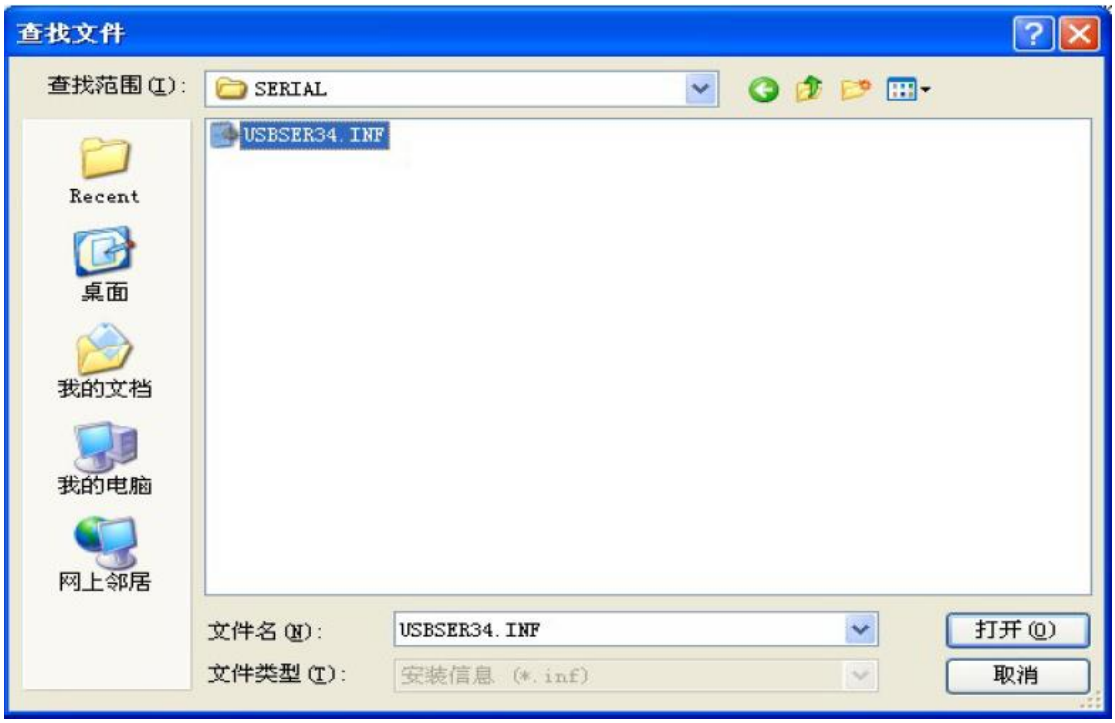
选择“从列表或指定位置安装（推荐）”

出现此对话框,选择不要搜索，自己选择安装 点“下一步”

点击"下一步"

出现此对话框、点击“从磁盘安装”

点击“浏览” 到 USB 驱动文件夹找到安装文件打开



点击“确定”和“下一步”

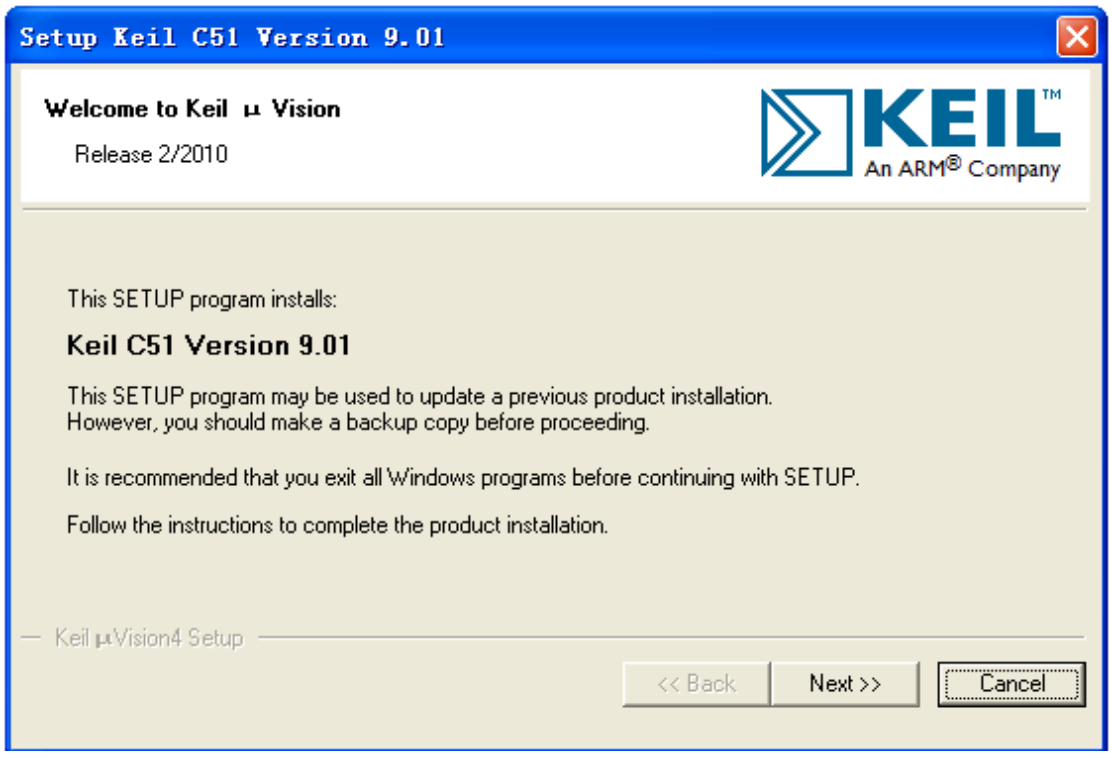
点击"完成"安装结束

KEIL 软件安装

我们用现在新版本的 uVision 4 KEIL 开发环境，使用的版本是破解版，仅供大家学习使用。如做商业开发，请购买正版，可以获得更多 keil 软件原厂技术支持。

即将安装软件如下，一个 keil 安装程序，一个注册机（右侧）。

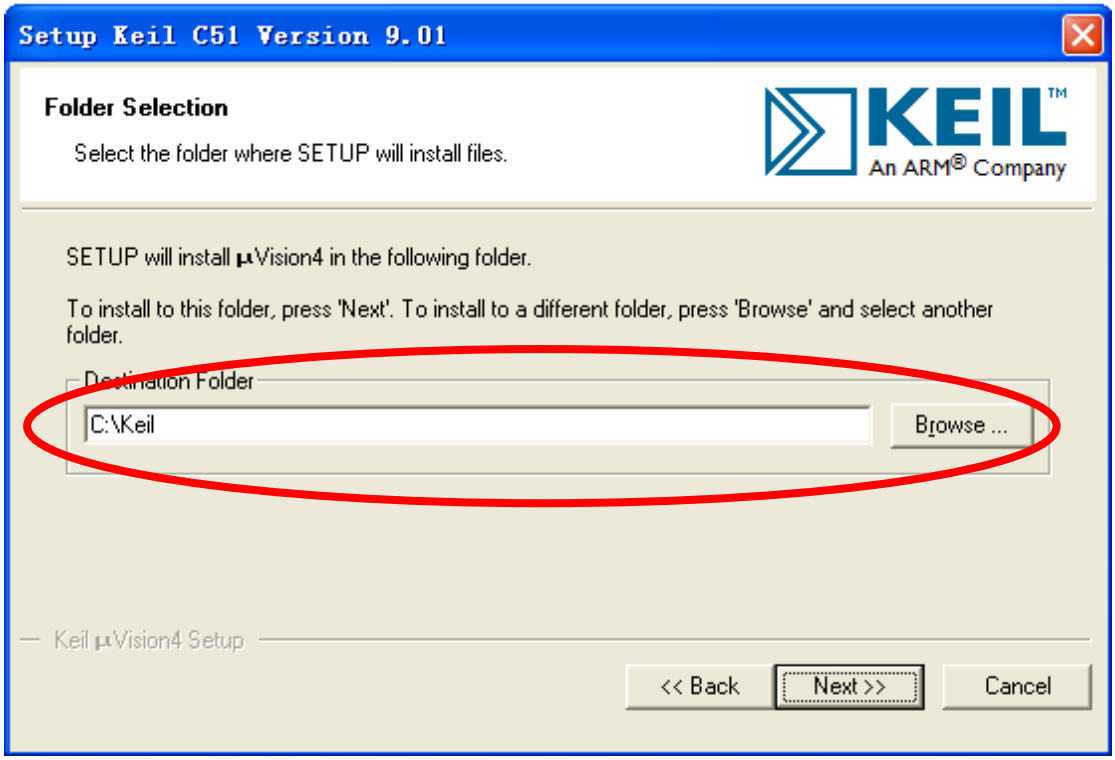
打开 C51V901.EXE 安装程序



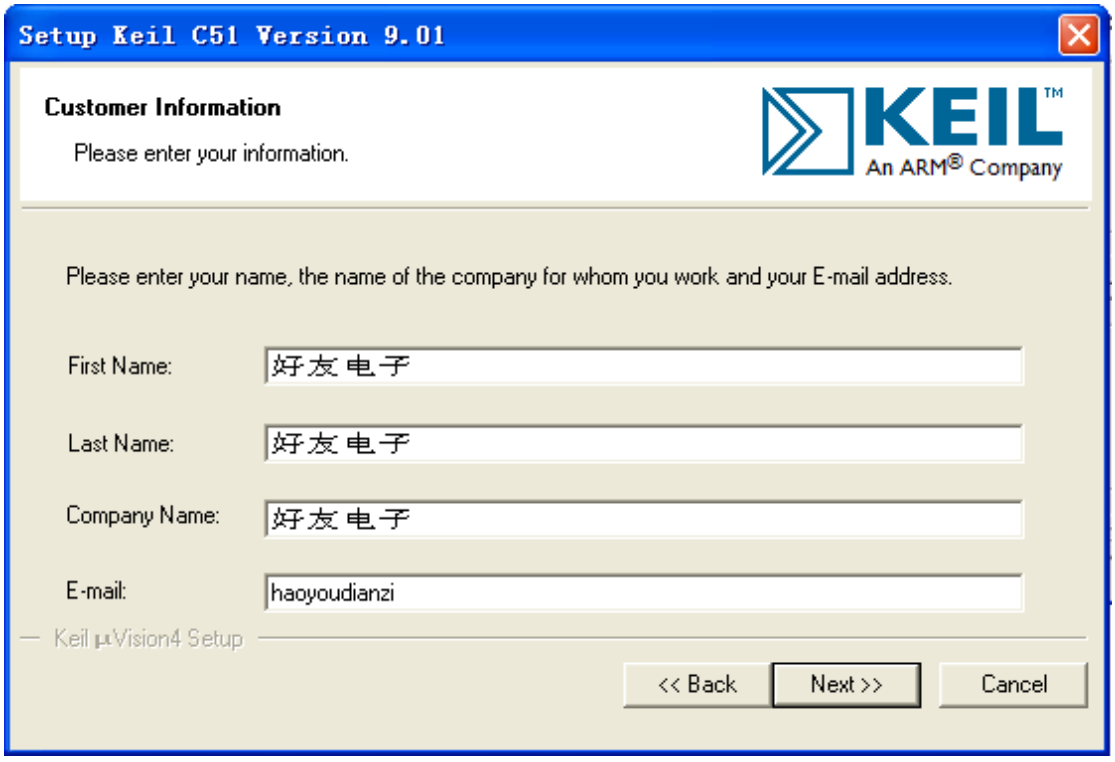
点击 Next >>

I agree all the tems of ……选中

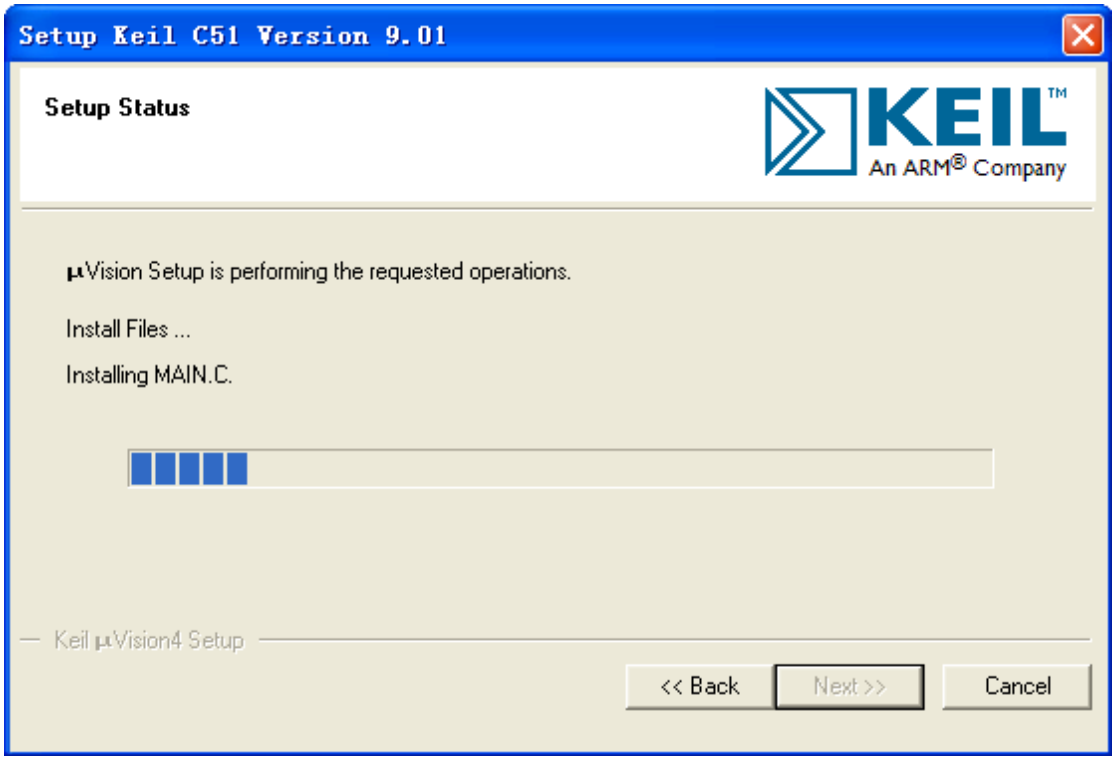
点击 Next >>



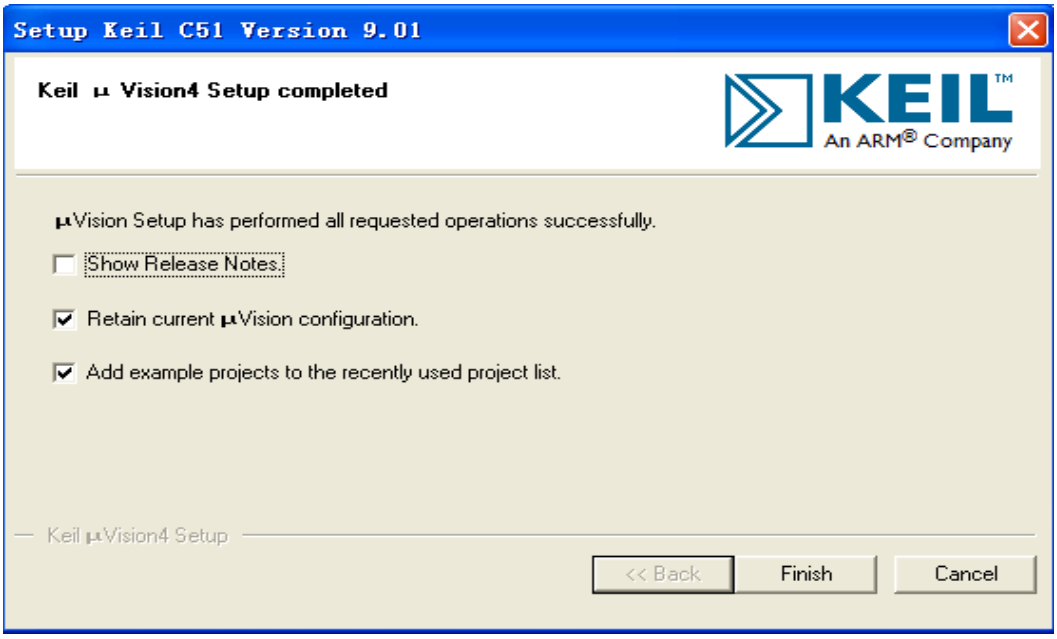
设置安装目录，根据自己的情况选中安装目录，重新设置点击 Browse，这里默认 C 盘，设置好安装目录后　点击 Next>>



输入相关信息（随便输入），输入完毕后点击 Next>>



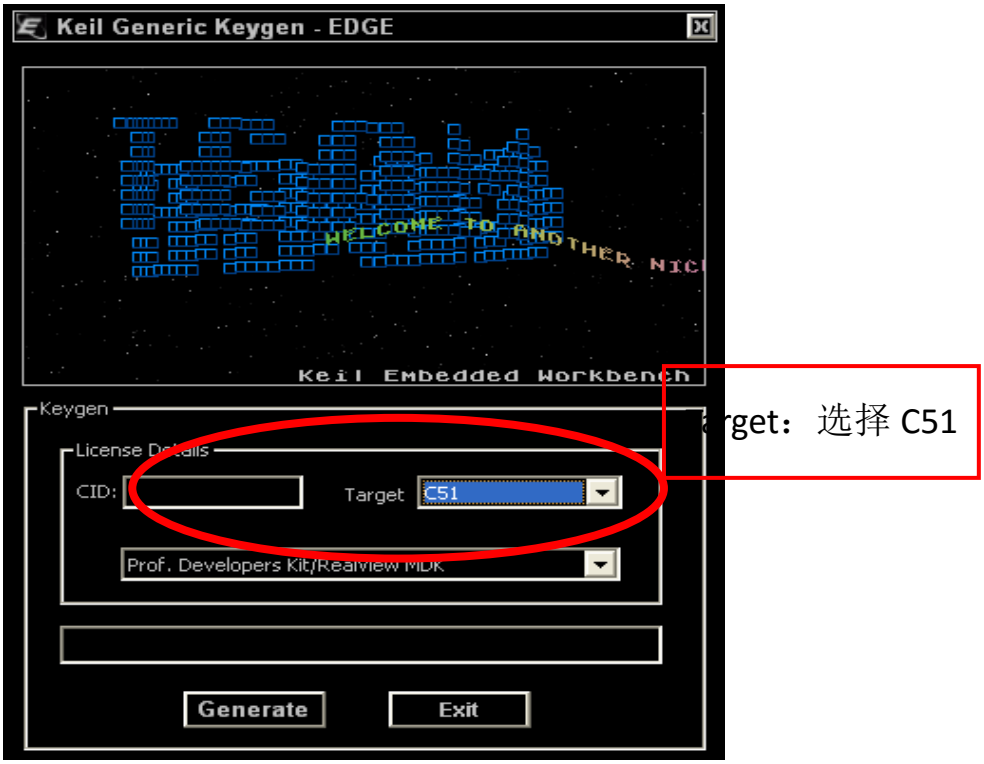
开始安装，安装过程中……..等待安装完成



安装完成，点击 Finish 即可。

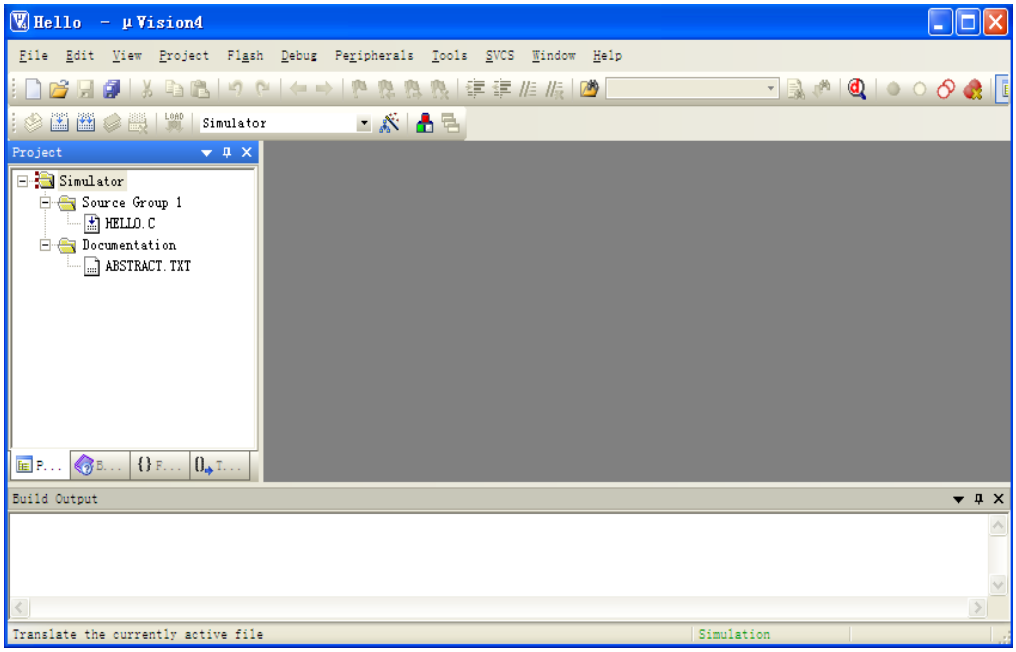
接下来破解软件。 打开注册机软件

KEIL_Lic.exe

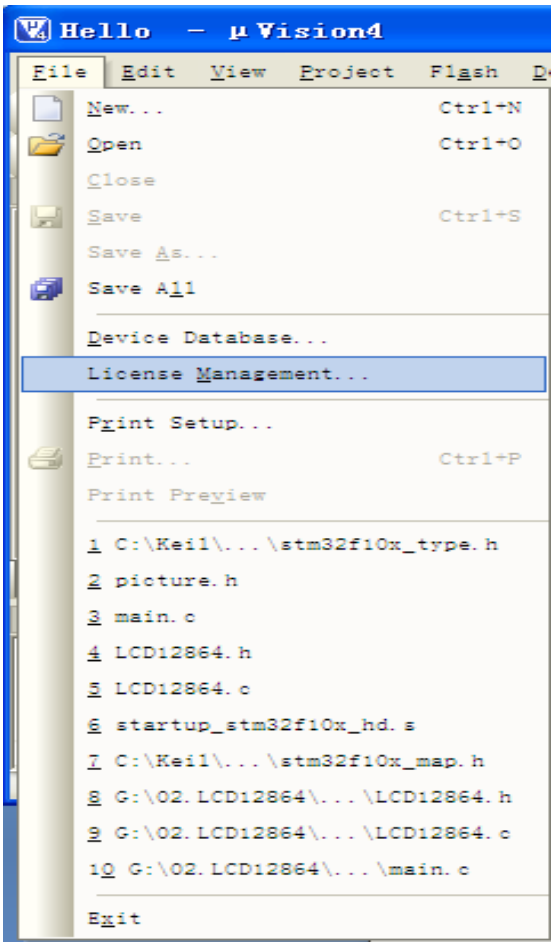


rget：选择 C51

打开刚刚安装好的 keil 软件

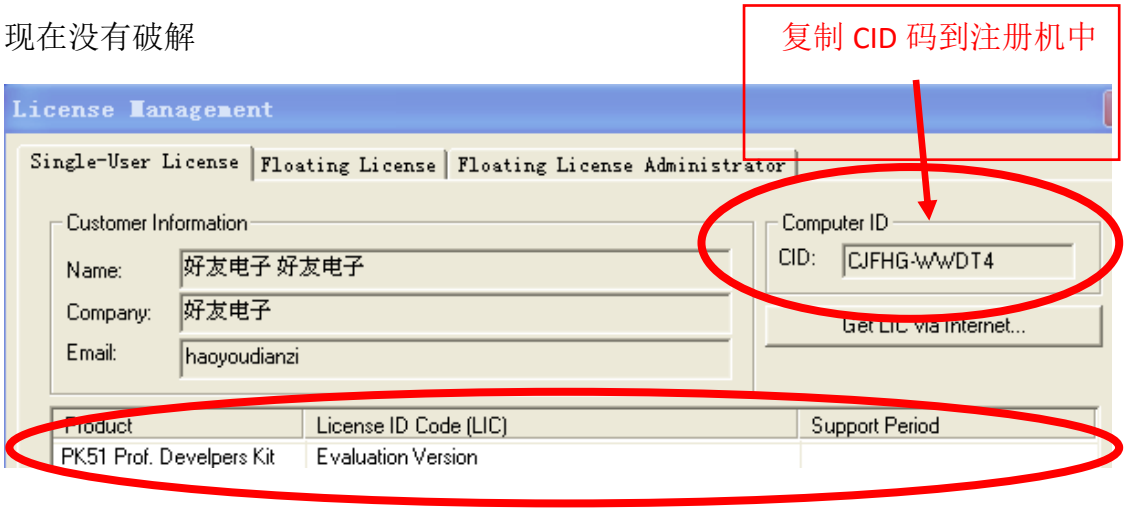


点击 File 菜单

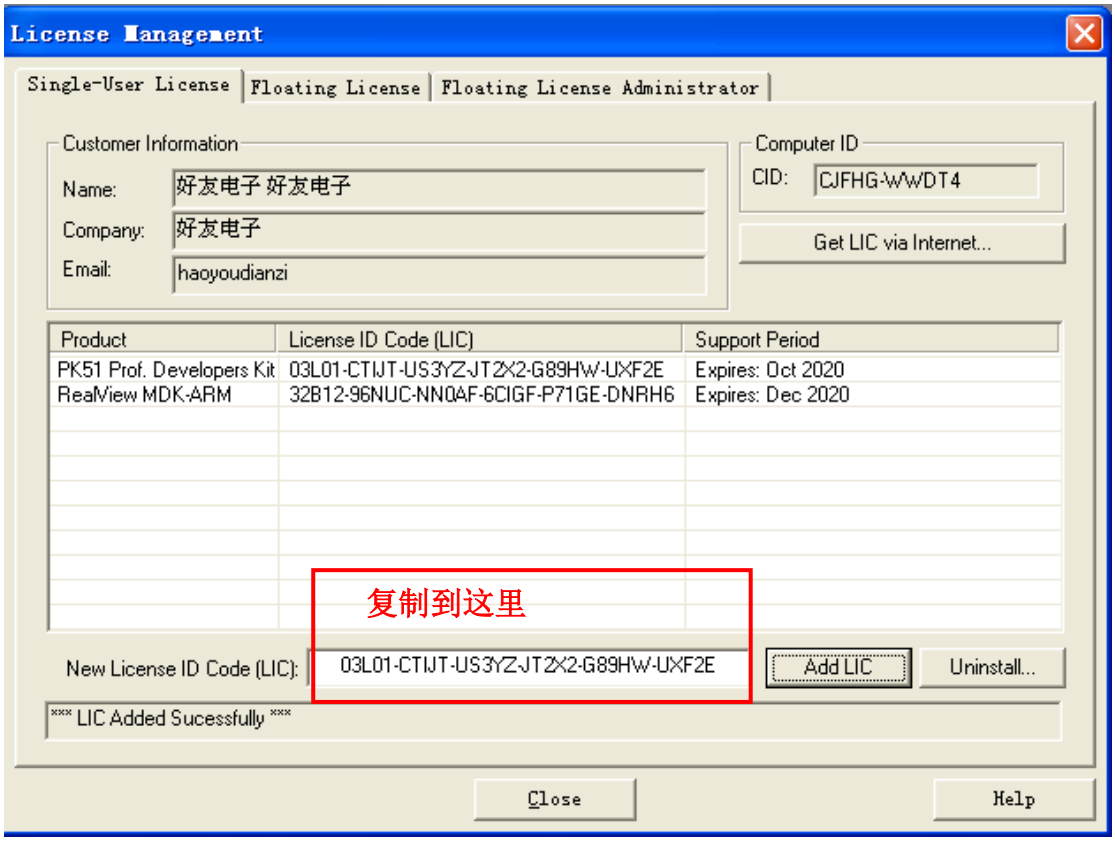


选择 License Management。

现在没有破解

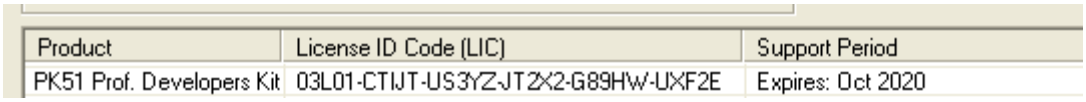


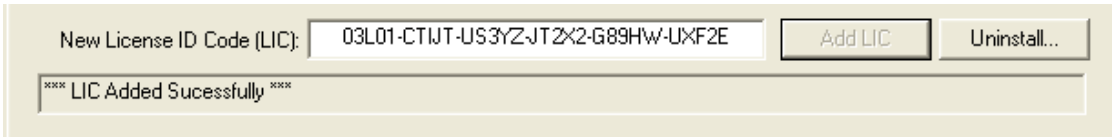
复制 CID 码，之后点击 Generate 生成注册码



复制完注册码后，点击右侧的  ，即可完成破解。提示

如下：





以上软件安装完毕。接下来看程序了。

第三讲 程序下载

我们可以通过 USB 接口方式下载对我们开发板的单片机下载程序。

下载软件又可分为 STC 官方软件 和普中科技自己开发的软件

利用官方软件下载程序需要手动重启单片机，需要给单片机重新上电启动。普中开发的下载软件及所设计的开发板实现了全自动下载功能，省去了手动操作，极大方便了学习及开发人员。接下来就来讲一下下载程序的方式。

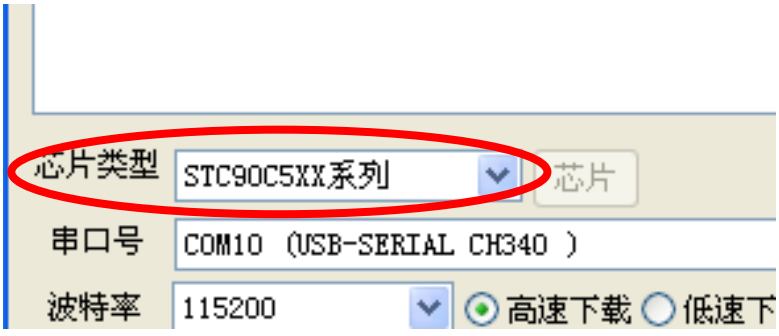
1. USB 口，普中软件

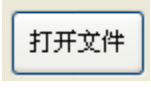
我们插上 USB 口，打开开发板电源开关。然后打开普中 ISP 下载软件，如下图 ：

在 usb 转串口驱动安装成功后，打开软件应该有串口号，如图指示。一般笔记本建议使用低速下载，台式可以使用高速下载，如果笔记本使用高速下载的话那么会报警波特率超时。

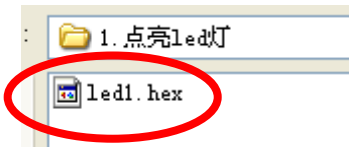
2. 芯片类型选择 STC90C5XX（具体的要根据您板子上使用的单片机型号）

如下图。

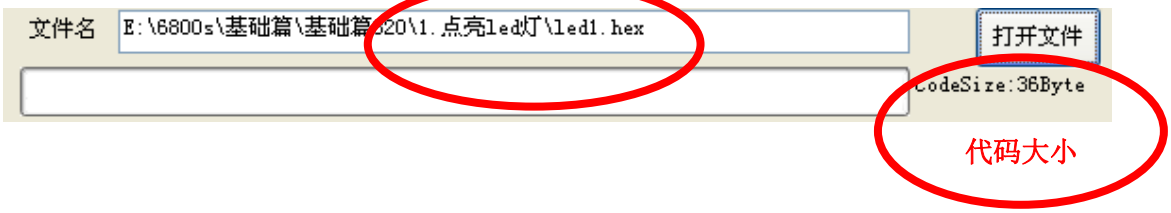


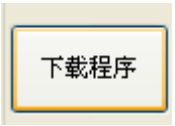
加载我们要下载到单片机里的程序，单击 

，选择下载的后缀为.Hex 的文件，



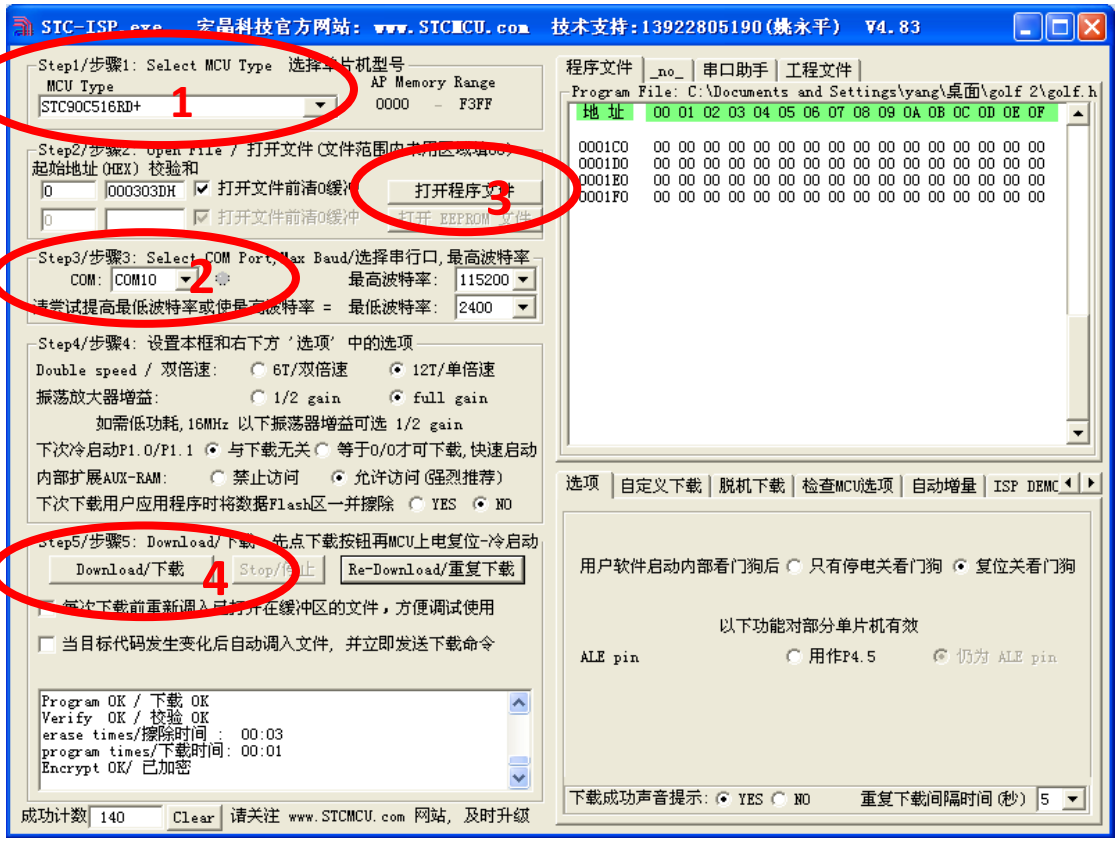
打开后，会在文件名处有提示：文件路径，还有代码大小。



4 最后一步：点击下载程序  ，下载成功如图显示。

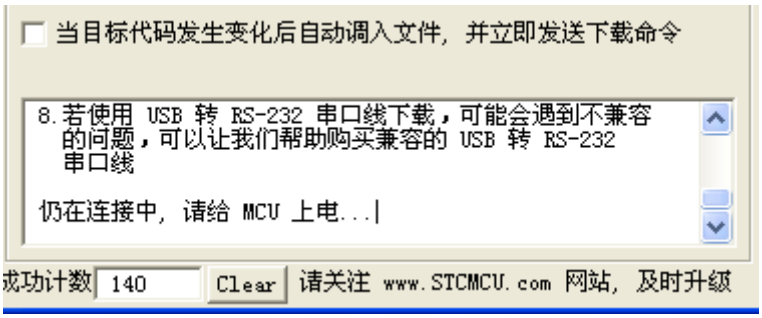
此时继电器工作会有小的啪啪的响声。

接下来我们用官方 STC 下载软件下载



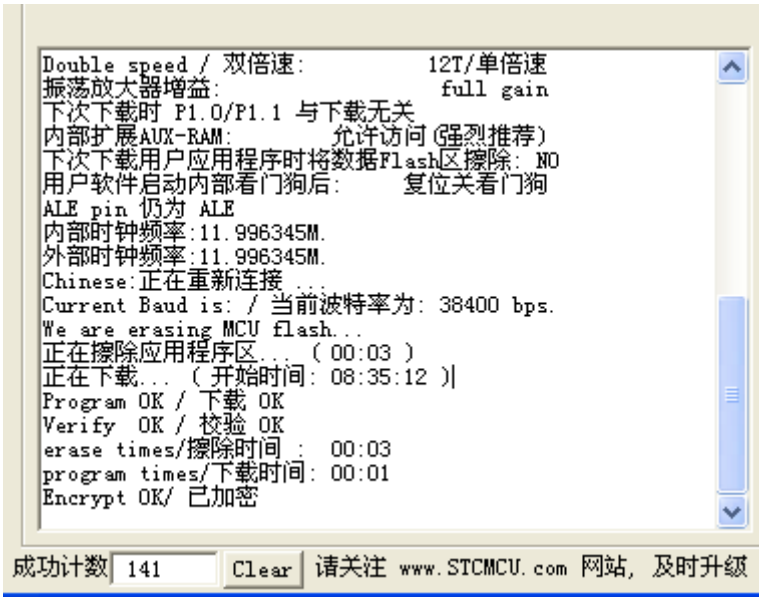
1. 选择开发板上单片机型号，我们选 STC90C516
2. 选择串口，可通过设备管理器查看
3. 打开需要下载到单片机的程序
4. 点击下载

点击下载按钮后会有这样的信息：



这时需要手动按开关键，关闭开关，打开开关这样一个过程，主要是给单片机冷启动。

重新上电后，会出现



开始下载程序，下载完成。上图表示下载成功。

现在用官方软件下载几乎都要手动重新给单片机上电过程。市面的开发板主要也都是采用这种下载方式。 建议使用普中软件自动下载。

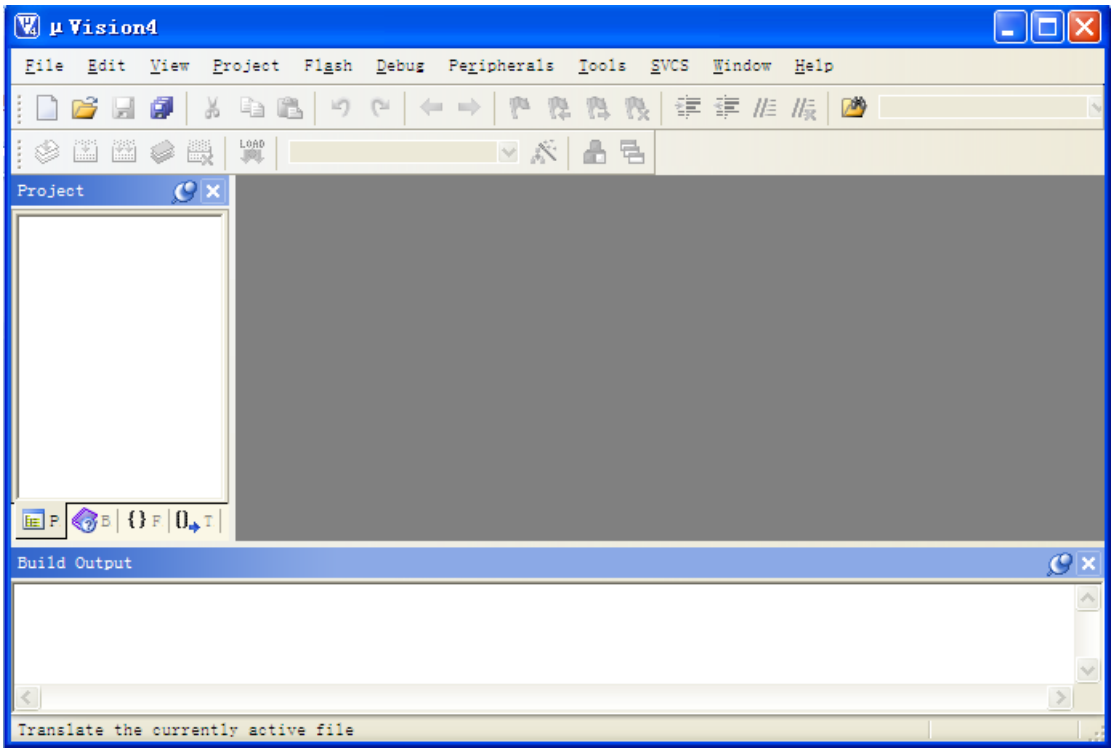
第四讲 KEIL 软件使用及入门 led 灯

单片机开发中除必要的硬件外，同样离不开软件，我们写的汇编语言源程序要变为 CPU 可以执行的机器码有两种方法，一种是手工汇编，另一种是机器汇编，目前已极少使用手工汇编的方法了。机器汇编是通过汇编软件将源程序变为机器码，用于 MCS-51 单片机的汇编软件有早期的 A51，随着单片机开发技术的不断发展，从普遍使用汇编语言到逐渐使用高级语言开发，单片机的开发软件也在不断发展，Keil 软件是目前最流行开发 MCS-51 系列单片机的软件，这从近年来各仿真机厂商纷纷宣布全面支持 Keil 即可看出。Keil 提供了包括 C 编译器、宏汇编、连接器、库管理和一个功能强大的仿真调试器等在内的完整开发方案，通过一个集成开发环境（uVision）将这些部份组合在一起。运行 Keil 软件需要 Pentium 或以上的 CPU，16MB 或更多 RAM、20M 以上空闲的硬盘空间、WIN98、NT、WIN2000、WINXP 等操作系统。掌握这一软件的使用对于使用 51 系列单片机的爱好者来说是十分必要的，如果你使用 C 语言编程，那么 Keil 几乎就是你的不二之选（目前在国内你只能买到该软件、而你买的仿真机也很可能只支持该软件），即使不使用 C 语言而仅用汇编语言编程，其方便易用的集成环境、强大的软件仿真调试工具也会令你事半功倍。

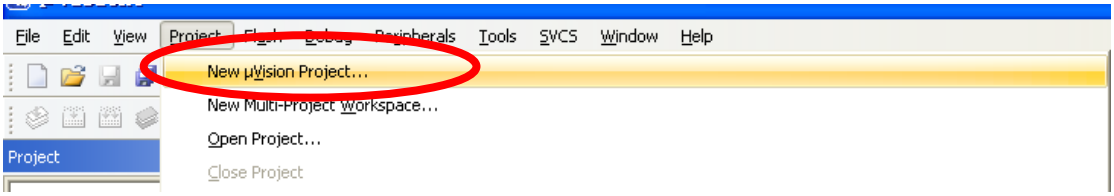
我们将通过一些实例来学习 Keil 软件的使用，在这一部份我们将学习如何输入源程序，建立工程、对工程进行详细的设置，以及如何将源程序变为目标代码。

在这里利用第二个例程，led 闪烁实验来建立 keil 工程。

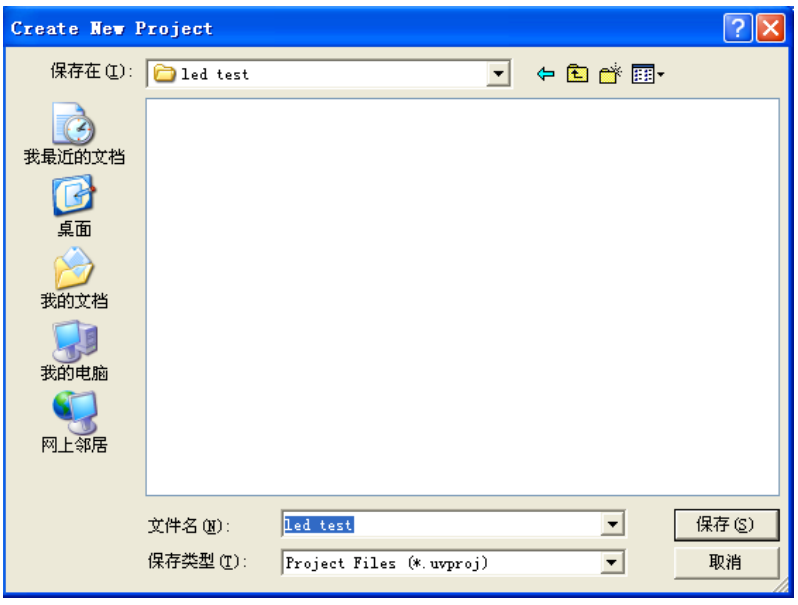
打开 keil 软件，版本μ Vision2 ，μvision3，μvision4 都一样，在这里用μvision4 版本演示，打开之后如下图，有的时候会默认打开上次使用的工程，单击 Project 菜单，选择 Close Project 关闭了默认打开的工程，显示下图界面：



我们要建立新的工程，选择 Project→ new µ vision projiect…

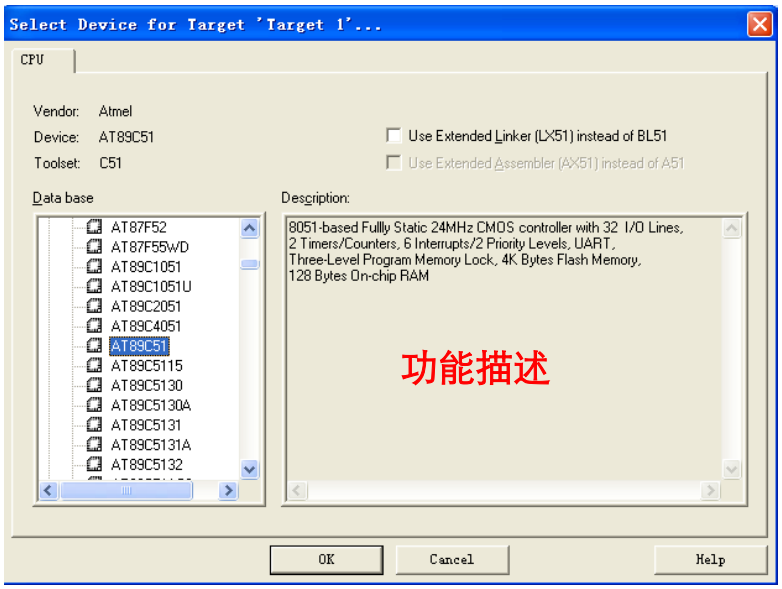


选择工程要保存的路径，输入工程文件名，如图

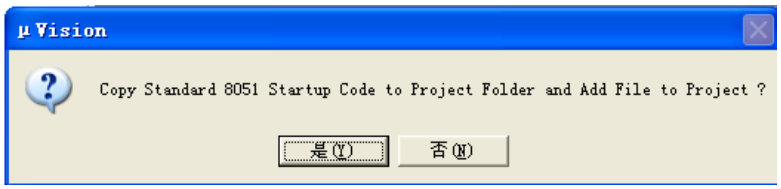


点击保存后会弹出一个对话框，要求用户选择单片机型号，可以根据用户使用的单片机来选择，我们使用的 STC90 单片机是兼容 51 内核的，Keil C51

几乎支持所有的 51 内核单片机，51 内核具有通用型，如果程序用的资源不是太复杂，我们可以选择任意一款 51 单片机内核就行。在稍后我们会继续讲解怎么将 STC 单片机官方的头文件添加进来，在这暂不多介绍。Keil 软件的关键是程序代码的编写，而非是用户选择什么硬件。例程以添加 Atmel 的 AT89C51 来说明。如图，然后，单击 确定（OK）。

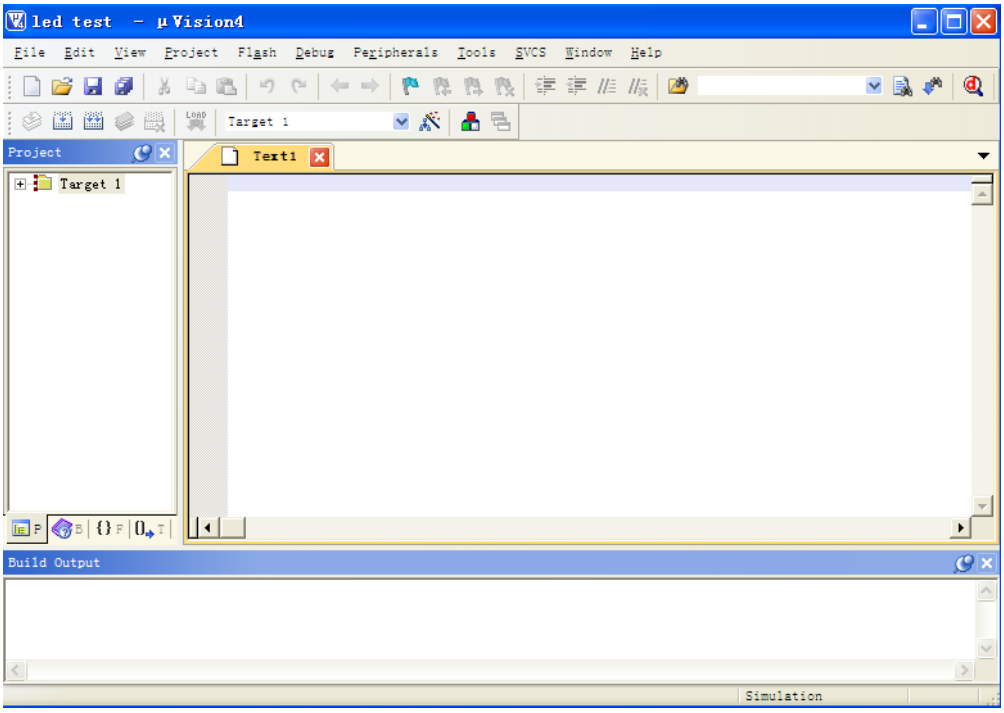


如果出现下面的界面：单击是就可以，意思是将单片机的启动代码添加到工程，我们不用修改。

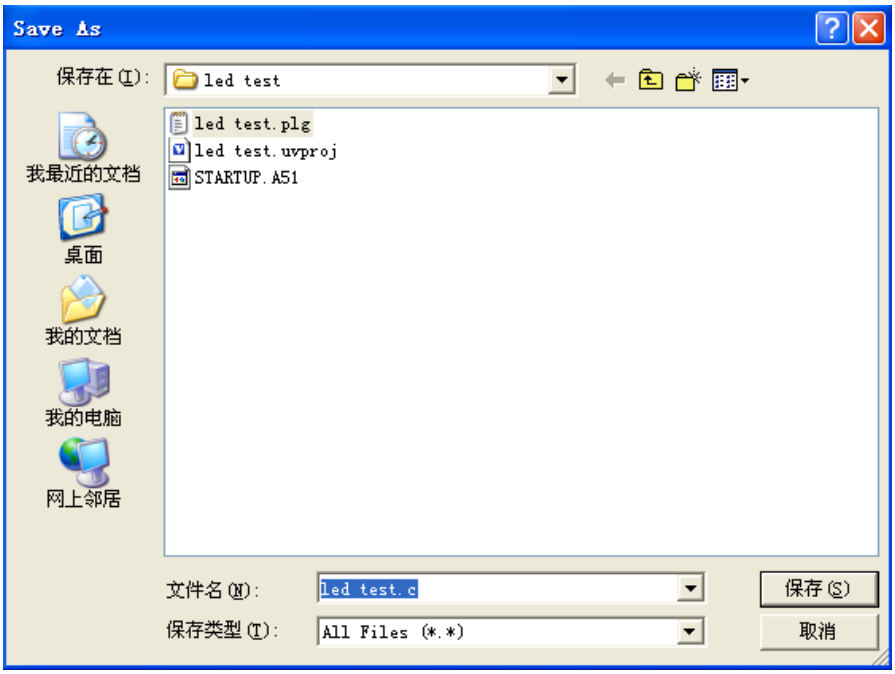


到目前我们还没建立一个完整的工程，只是有工程的名字，框架，工程中还没有任何文件代码，（除了启动代码,有的 keil 版本不显示启动代码），接下来我们添加文件及代码。

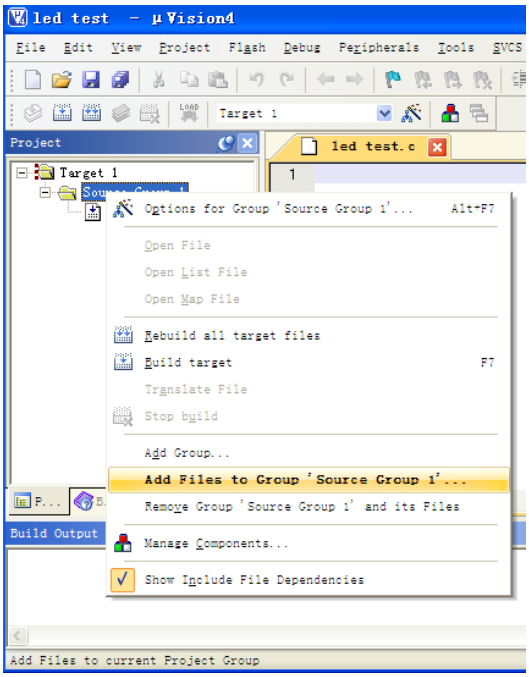
单击菜单 File→New 选项，或者单击界面上的快捷图标  ，新建文件串口如下。

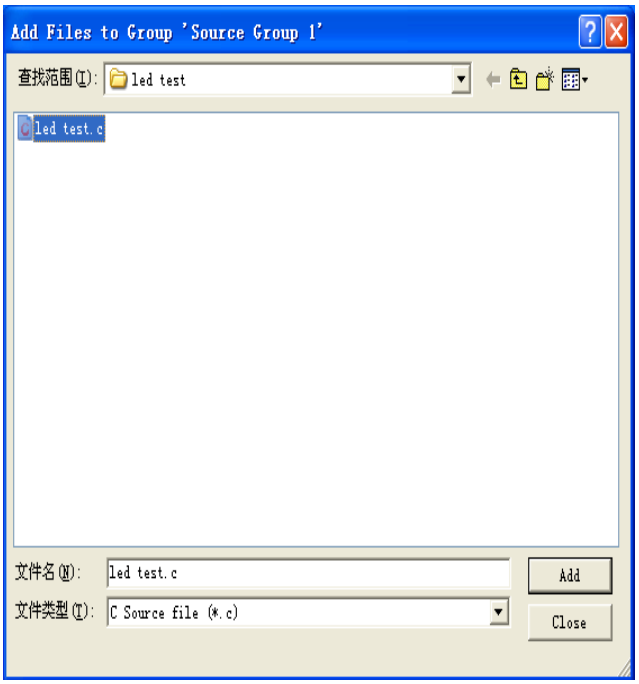


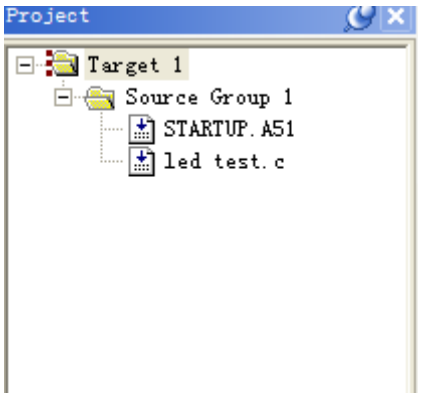
界面显示的 Text1 就是我们刚刚加入的文件，但是这个文件与我们的工程还没有直接联系起来，单击图标  ，保存我们当前的 text1 文件，输入要保存的文件名，同时要输入文件扩展名，扩展名很关键，用 C 语言编写的程序，则必须为**.c,汇编语言必须为**.asm（**为文件名），文件名是用户自己取的。填好文件名后单击保存。



回到编辑界面，单击 Target1 前面的“+”号，然后在 Source Group 1 上右键单击，选择 Add Files to Group Source Group1，选中我们刚才建立的.c 文件，这里是 led test.c，单击 add，只需单击一次。之后单击 Close 就可以。回到主界面后，单击 Source Group 1 前的“+”号，刚刚添加的文件显示在里面。







通过以上步骤我们就建立好了一个工程。接下来就可以写代码了。

在写代码之前介绍介个常用的按钮：


按钮：用来编译我们正在操作的文件。


按钮：用来编译修改过的文件，并生成应用程序共单片机下载。


按钮：用来重新编译当前工程中所有的文件，并生成应用程序共单片机下载。因为很多工程不止有一个文件，当有多个文件时，用它进行编译。

 按钮：用来打开“Option for Target”对话框，对当前工程进行设置。工程的各个参数都可以在这里设置，具体的设置方法用到的时候，再和大家详细讲述。

以上几个按钮是常用的，其他按钮在使用的时候再介绍。

1. 编写程序

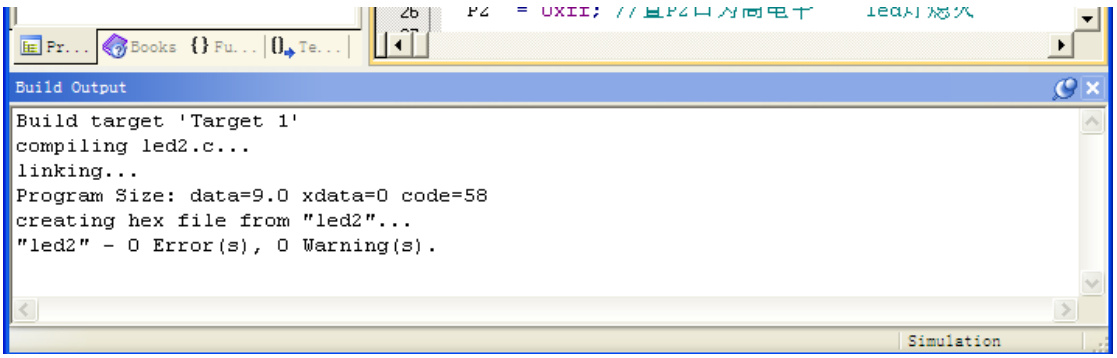
我们以一个程序为例来练习编写：led 闪烁。

这个程序大家一定要弄懂，也算是一个基础，懂了这个程序，也算是进入单片机的领域了接下来我会详细讲解这个例程。另外需要注意的是单片机写程序的时候，一定是英文状态下的字符，尤其注意“;”,往往就因为这个分号，程序出现问题，所以输入时一定要注意是在英文状态下的。

我们在编辑框中输入以下程序：

```
/*********************************************************************
*                          LED 闪烁的简单试验
*
*********************************************************************/
#include <reg51.h>      //此文件中定义了 51 的一些特殊功能寄存器

void delay(unsigned int i);  //声明延时函数

void main()
{
    while(1)
    {
        P2  = 0x00; //置 P0 口为低电平
        delay(600); //调用延时程序
        P2  = 0xff; //置 P0 口为高电平
        delay(600); // 调用延时程序
    }
}
/*******延时函数*************/
void delay(unsigned int i)
{
    unsigned char j;
    for(i; i > 0; i--)    //循环 600*255 次
        for(j = 255; j > 0; j--);
}
```

我们先编译一下：第一次编译我们点击  ，看一下结果



结果的意思：

编译 led2.c…

链接…

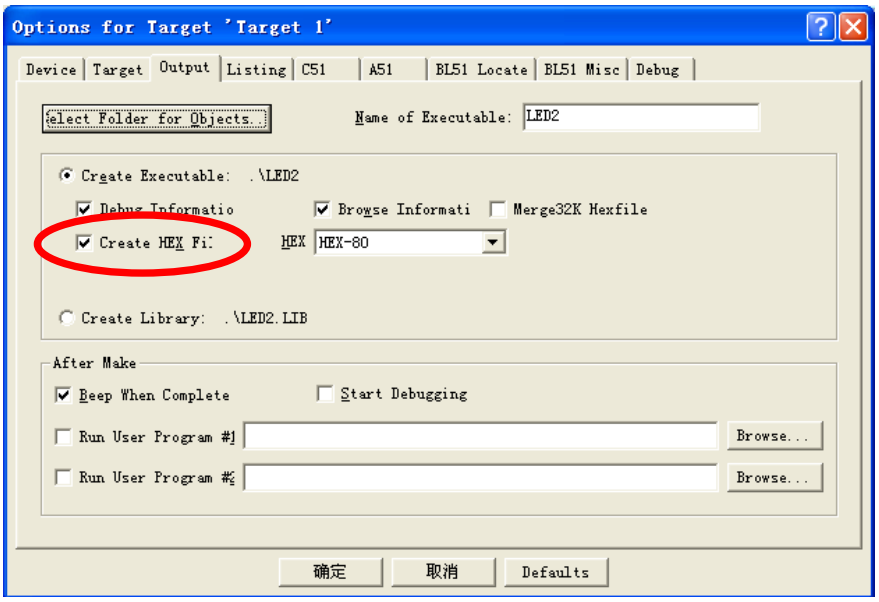
data=9.0--占用内部 RAM9 个字节，xdata=0：外部 RAM 0 字节，

Code=56 代码长度为 56 字节

生成单片机可下载的 HEX（十六进制）文件。

没有错误，没有警告。

在这里说明一点：生成 HEX 文件是我们自己设置的，默认的情况下不会生成 HEX 文件。单击  进行设置，如下图，选择 Output，勾选 Creat hex，后单击确定。

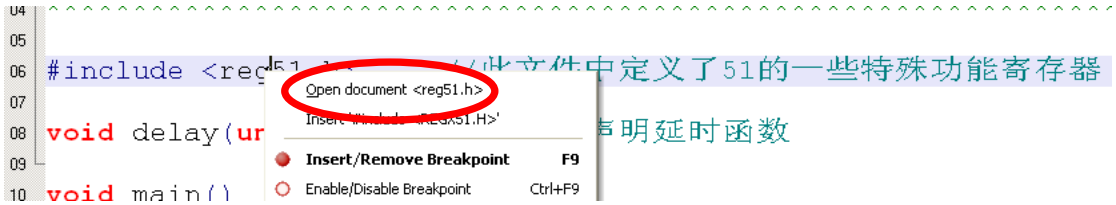


2. **接下来我们分析上面的程序：**

/* */作用，它是用来注释一段内容的，编译器不对其进行编译，只要在/* */直接的内容都被注释掉。

// 是用来注释其后面的内容，只能注释一行。

#include <reg51.h> //这句告诉我们包涵 51 的头文件，那这个头文件里面放的是什么东西呢？放的是 51 单片机对应的操作的寄存器地址，如我们直接用来操作的 P1 口就是代表 0x90 地址，我们可以将光标低位在<reg51.h>上然后右键



打开头文件，

```
#ifndef __REG51_H__
#define __REG51_H__

/*  BYTE Register  */
sfr P0   = 0x80;
sfr P1   = 0x90;
sfr P2   = 0xA0;
sfr P3   = 0xB0;
sfr PSW  = 0xD0;
sfr ACC  = 0xE0;
.....................................
/*  SCON  */
sbit SM0  = 0x9F;
sbit SM1  = 0x9E;
sbit SM2  = 0x9D;
sbit REN  = 0x9C;
sbit TB8  = 0x9B;
sbit RB8  = 0x9A;
sbit TI   = 0x99;
sbit RI   = 0x98;
#endif
```

头文件中定义了 51 系列单片机的所有功能寄存器，我们直接使用其代号就可以，P0，P1 等。

如：sfr P0=0x80，意思是把单片机内部地址 0x80 处的寄存器重新起名字叫 P0，以后我们在程序中直接操作 P0,就相当于对单片机内部 0x80 地址处的寄存

器进行操作。通俗的讲，也就是说，通过 sfr 这个关键字，让 Keil 编译器在单片机与人之间搭建一个桥梁，我们操作 P0 口，单片机对应就去操作内部 0x80 的地址。所以我们写程序要在开始处直接包涵单片机的头文件。

sbit SM0 =0x9F;是定义位操作地址 0x9F 的，这个地址只代表一个位。我们操作 SM0，对应单片机就是操作位地址 0x9F。

总结起来 sfr 与 sbit 区别

sfr 是定义字节的 8 位

sbit 是定义位的 1 位

我们在返回主程序：

```
void delay(unsigned int i)
{
    unsigned char j;
    for(i; i > 0; i--)
        for(j = 100; j > 0; j--);
}
```

定义一个函数 delay();与 c 语言一样，要用一个函数，先定义，我们可以叫它子函数，可以调用的。

```
void main()  // 结构同 c 语言一样，main()函数开始
   {  // 大括号。
    P2  = 0x00; //置 P0 口为低电平
```

从这句开始，你现在控制单片机了，告诉单片机把你的 P2 口都输出 0，如果你不操作 P2 口了，那么 P2 口一直保持这个状态，直到你去改变它。

```
    delay(1000); //调用延时程序
```

调用子函数 delay(),告诉单片机去执行 delay 那个函数，那么 P2 口一直保持 0 这状态。

```
    P2  = 0xff; //置 P0 口为高电平
```

这时告诉 P2 口全部输出 1，(0xff=1111 1111).状态从 0 变成 1 了，对应的灯的输出也由低电平变成高电平了

```
    delay(1000); // 调用延时程序，再进行延时，
}
```

这个程序里执行完了一次又干什么呢，

Keil 编译器会编译成一直重复执行 main()函数里面的代码，整个代码的效果就是：

① P2 输出低电平

② 延时一段时间，目的是 P2 输出的低电平保持一段时间

③ P2 输出高电平

④ 延时一段时间，目的是 P2 输出的高电平保持一段时间

⑤ 重复①到④的过程　　实际效果就是 led 一闪一闪

通过更改 delay（延时时间 i），参数 i，可以改变闪烁频率。

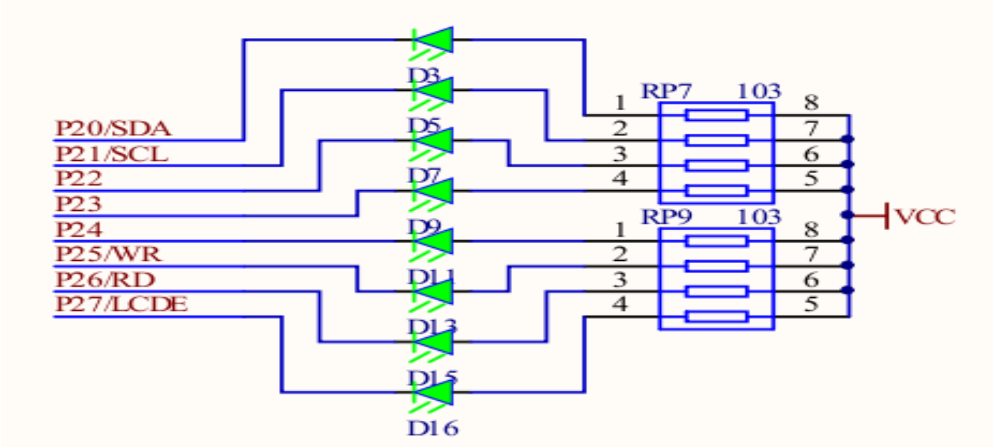
实际操作：

1. 打开软件 

(提前装好驱动)，点击打开文件，打开我们刚刚编译的 led.hex 文件，下载程序，就会看到 led 闪烁。

我们在看一个例程，就是单片机最经典的流水灯例程：

以下是我们的硬件电路,led 一段接单片机 P2 口的 8 个引脚，另外一边接排阻 RP12 和 RP13，然后接到电源



基础知识介绍：

排阻：

一般在排阻上都标有阻值号，其公共端附近也有明显标记。如下下图表示为 472,

表示 47*2=4700 欧姆，还有的标号如 3R0，表示阻值为 3 欧姆，4K7 表示阻值为 4.7kΩ，R002 表示阻值为 0.002 欧姆。

(2)发光二极管。它具有单向导电性，通过 5mA 左右电流即可发光，电流越大，其亮度越强，但若电流过大，会烧毁二极管，一般我们控制在 3 mA-20mA 之间。在这里，给发光二极管串联一个电阻的目的就是为了限制通过发光二极管的电流不要太大，因此这个电阻又称为“限流电阻”。当发光二极管发光时，测量它两端电压约为 1.7V，这个电压又叫做发光二极管的“导通压降”。图 2.2.9 和图 2.2.10 分别为直插式发光二极管和贴片式发光二极管实物图。发光二极管正极又称阳极，负极又称阴极，电流只能从阳极流向阴极。直插式发光二极管长脚为阳极，短脚为阴极。仔细观察贴片式发光二极管正面的一端有彩色标记，通常有标记的一端为阴极。

图 2.2.9

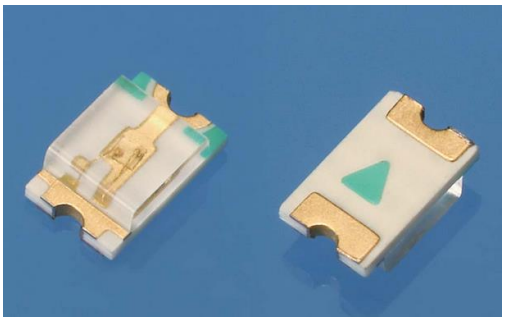


图 2.2.10

关于排阻大小的选择:欧姆定律想必大家都清楚，U=IR，当发光二极管正常导通时，其两端电压约为 1.7V，发光管的阴极为低电平，即 0V，阳极串接一电阻，电阻的另一端为 Vcc ， 为 5V，因此加在电阻两端的电压为 5V-1.7V=3.3V，计算穿过电阻的电流，3.3 V/ 1000Ω=3.3mA。即穿过发光管的电流也为 3.3mA，若想让发光管再亮一些，我们可以适当减小该电阻。

看我们的原理图，可以知道：LED 的正极接在 VCC 上，只要给了低电平，那

么 LED 就会亮，低电平对应到单片机的逻辑就是 0，只要单片机的某一个管脚输出 0，那么对应的发光二极管就会亮。我们看一下源码：

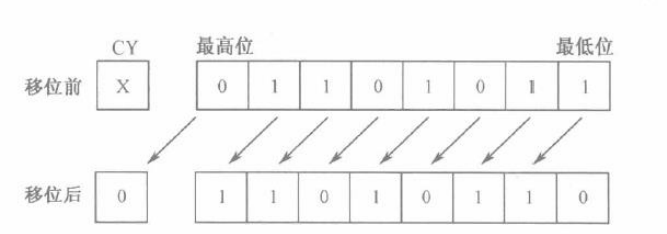
```
/*********************************************************************************
* 实验名                  : 左右流水灯实验
* 使用的 IO       : P2
* 实验效果        : 点亮的 LED 从右边往左边移动，到达左边再往右边移动，依此循环。
*   注意                          :
*********************************************************************************/
#include<reg51.h>
#include<intrins.h>     //因为要用到左右移函数，所以加入这个头文件

#define GPIO_LED P2     //将 P2 口另外取名为 GPIO_LED

void Delay10ms(unsigned int);   //误差 0us
/*********************************************************************************
* 函数名          : main
* 函数功能          : 主函数
* 输入            : 无
* 输出            : 无
*********************************************************************************/
void main(void)
{
    unsigned char n;
    GPIO_LED=0xfe;
    while(1)
    {
        for(n=0;n<7;n++)            //左移 7 次，这样子就会到达最左边
        {
            GPIO_LED=_crol_(GPIO_LED,1); //将 GPIO_LED 左移一位
            Delay10ms(50);                              //延时
        }
        for(n=0;n<7;n++)                        //右移 7 次，这样子就会到达最右边
        {
            GPIO_LED=_cror_(GPIO_LED,1);    //将 GPIO_LED 右移一位
            Delay10ms(50);                                      //延时
        }
    }
}
/*********************************************************************************
* 函数名          : Delay10ms
* 函数功能          : 延时函数，延时 10ms
* 输入            : 无
* 输出            : 无
```

```
*********************************************************************************/
void Delay10ms(unsigned int c)   //误差 0us
{
    unsigned char a,b;
    for(;c>0;c--)
        for(b=38;b>0;b--)
            for(a=130;a>0;a--);
}
```

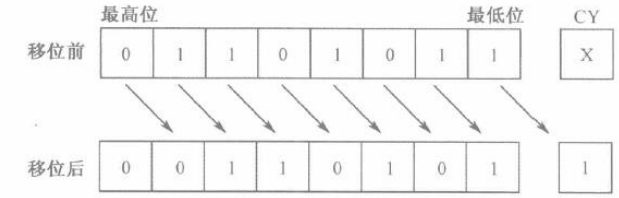
以上程序实现点亮一盏 LED 之后左右移动。

讲解：移位操作‘<<’ ‘>>’

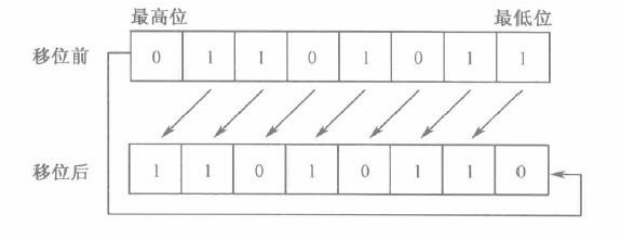
1. **左移**。C51 中操作符为“<<”，每执行一次左移指令，被操作的数将最高位移入单片机 PSW 寄存器的 CY 位，CY 位中原来的数丢弃，最低位补 0,其他位依次向左移动一位，如下图所示：



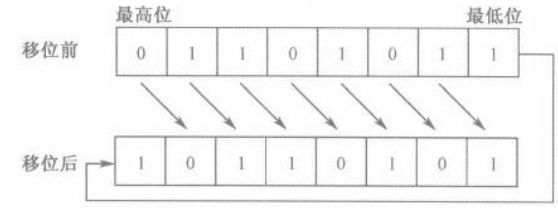
2. **右移**。C51 中操作符为“>>’’，每执行一次右移指令，被操作的数将最低位移入单片机 PSW 寄存器的 CY 位，CY 位中原来的数丢弃，最高位补 0，其他位依次向右移动一位，如下图所示。



3. **循环左移**。最高位移入最低位，其他位依次向左移一位。C 语言中没有专门的指令，通过移位指令与简单逻辑运算可以实现循环左移，或直接利用 C51 库中自带的函数_crol_实现，如下图所示。_crol_函数所在的头文件是<intrins.h>

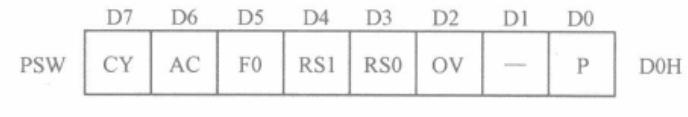


4. **循环右移**。最低位移入最高位，其他位依次向右移一位。C 语言中没有专门的指令，通过移位指令与简单逻辑运算可以实现循环右移，或直接利用 C51 库中自带的函数-cror 实现，如下图所示。_cror_函数所在的头文件是<intrins.h>



5. **讲解：PSW 寄存器**

PSW (Program Status Word)全称为程序状态字标志寄存器，是一个 8 位寄存器，位于单片机片内的特殊功能寄存器区，字节地址 DOH，用来存放运算结果的一些特征，如有无进位、借位等，使用汇编编程时 PSW 寄存器很有用，但在利用 C 语言编程时，编译器会自动控制该寄存器，很少人为操作它，大家只需做简单了解即可.其每位的具体含义如下图所示.



1) CY 一进位标志位，它表示运算是否有进位(或借位).如果操作结果在最高位有进位(加法)或者借位(减法)，则该位为 1，否则为 0.
2) AC 一辅助进位标志，又称半进位标志，它指两个 8 位数运算低四位是否有半进位，即低四位相加(或相减)是否进位(或借位)，如有 AC 为 1，否则为 Oe
3) FO 一由用户使用的一个状态标志位，可用软件来使它置 1 或清 0，也可由软件来测试它，以控制程序的流向.

4) RS1，RSO-4 组工作寄存器区选择控制位，在汇编语言中这两位用来选择 4 组工作寄存器区中的哪一组为当前工作寄存区.

5) OV 一溢出标志位，反映带符号数的运算结果是否有溢出.有溢出时，此位为 1，否则为 0.

6) P 一奇偶标志位，反映累加器 ACC 内容的奇偶性，如果 ACC 中的运算结果有偶数个 I(如 11001100B，其中有 4 个 1)，则 P 为 0，否则 P 为 1.

6. _cror_（）；函数

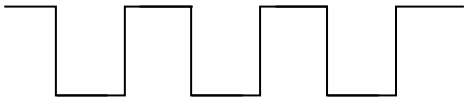
循环右移函数，包含在 intrins.h 的库函数里面。

7. _crol_（）；函数

循环左移移函数，包含在 intrins.h 的库函数里面。

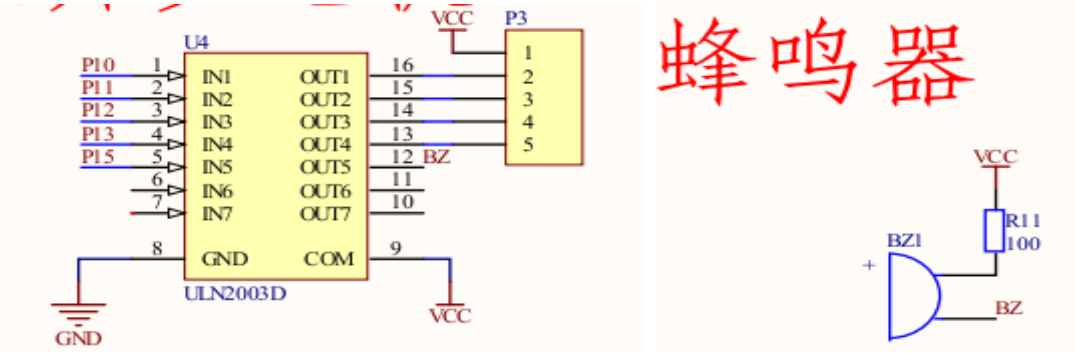
第五讲 蜂鸣器

蜂鸣器是一种一体化结构的电子讯响器，采用直流电压供电，广泛应用于计算机、打印机、复印机、报警器、电子玩具、汽车电子设备、电话机、定时器等电子产品中作发声器件。我们开发板上常用的蜂鸣器就是常常说的交流蜂鸣器或直流蜂鸣器（自激式蜂鸣器）。 直流蜂鸣器是给一定的驱动直流电压就会响。而交流蜂鸣器是需要给蜂鸣器一个脉冲才会响。常见的有 PWM 波控制蜂鸣器的频率。脉冲就是高低电平的切换，如下图：一个方波脉冲



我们用单片机的 IO 口实现一种这样高低电平的方波，驱动蜂鸣器发音。我们板子上配的就是交流蜂鸣器。

接下来我们看一下蜂鸣器的硬件电路：



蜂鸣器通过 ULN2003 驱动。

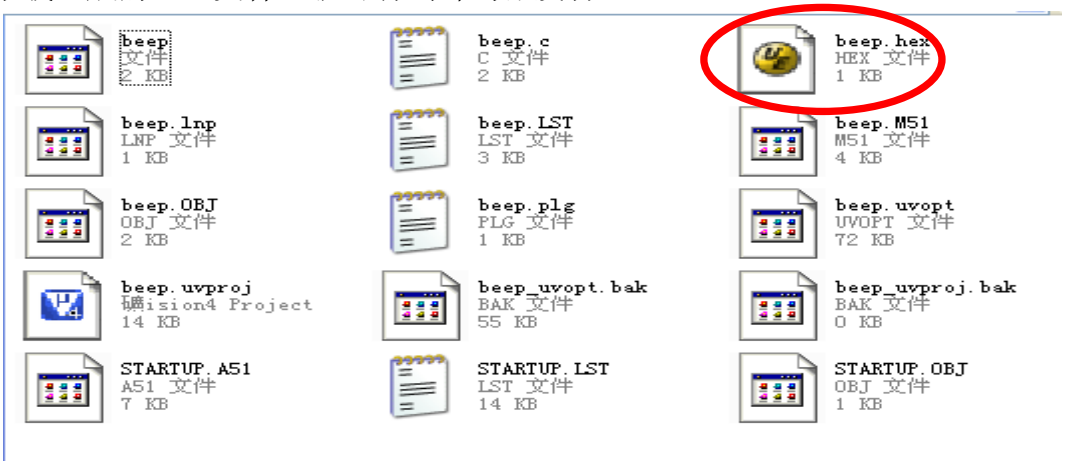
这里 ULN2003 暂不多做介绍，详细介绍可参考步进电机篇。

我们看一下程序源码：

```
/************************************************************************************
*
* 实验名                    : 蜂鸣器实验
* 使用的 IO          : P1^5
* 实验效果           : 蜂鸣器响。
*    注意                          :
************************************************************************************
```

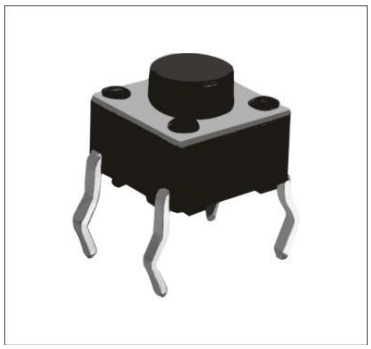
```
/

#include <reg51.h>
sbit Beep =  P1^5 ;
void Delay(unsigned int i) ;
/*******************************************************************************
* 函数名          : main
* 函数功能          : 主函数
* 输入            : 无
* 输出            : 无
*******************************************************************************/

void main()
{
    Beep= 1;
    Delay(5);
    Beep= 0;
    Delay(5);
}
/*******************************************************************************
* 函数名          : Delay()
* 函数功能          : 延时函数
* 输入            : 无
* 输出            : 无
*******************************************************************************/

void Delay(unsigned int i)
{
    char j;
    for(i; i > 0; i--)
        for(j = 200; j > 0; j--);
}
```

下载生成的 hex 文件，就可听到蜂鸣器发音。

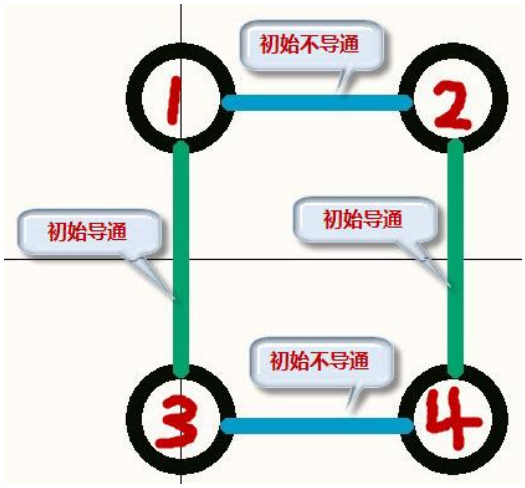


第六讲 独立按键

按键是什么东西，我想这个就不必由我向各位阐述了。嗯，如你所见，按键种类繁多，功能有简有繁，极大的充斥着我们的生活。但是无论如何，所有的按键其实都有一个原型，来源于同一种原理，所有的按键无论多复杂，多华丽，都是从这样一个原型发展而成的。好比你就算长的再帅，你也是只猩猩变来的，呵呵。我们平日所见到的绝大部分的按键，其实都可以归类为一种，叫“接触式按键”。下图为一个典型的接触式按键（又称轻触开关）。



需要特别说明的是，这里说的“接触”，是指机械层面上的接触，而不是感光或者某些特殊涂层（比如触摸屏）一类的接触。所以，按键的工作特性其实是一种机械特性，下文会详细说明。

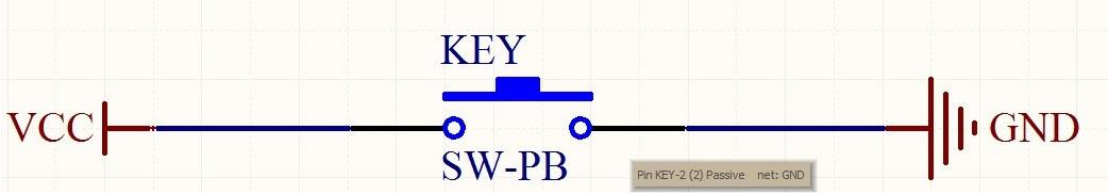


，

如上图，请对照图一想象，1、2、3、4 分别对应按键的四个引脚，其中蓝色的线表示按键未被按下之时的状态，我成为初始状态，它是不导通的；而绿色的线是却永久导通的。各位明白了么，其实是两个相同的结构连在一起了。我们

只要将需要按键开关作用的线路分别接在1、3 和2、4 的任意取一组合，概括起来就是（1，2）、（1，4）、（3，2）、（3，4）四种组合，都可以起到我们预期的开关作用。

相信以上说明使大家对按键的工作原理有了个比较清晰的认识了，现在来说说一个小知识。先看下图（图4）：



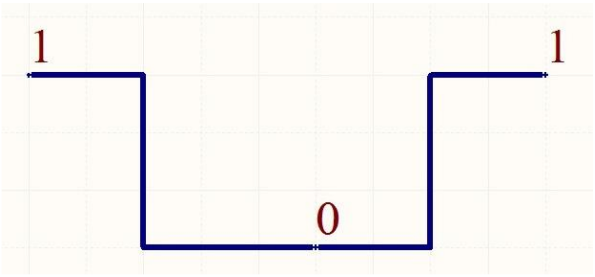
首先说明的是，上图的连法是不允许的，因为当按键按下之后，电源和地短接，会将导线直接烧毁。但是此处用作特例，假设导线不会烧毁。现在来提出一个问题，当按键按下以后，请问如果这时用万用表测量导线上任何一处的电压，得到的结果是VCC 还是GND 的电压？

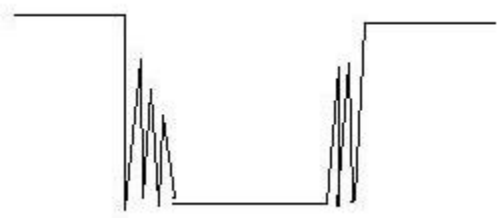
答案是：GND，即表示测出的电压为0V。为什么呢，因为导线上，对于两端的电平是一种类似于程序语言逻辑运算里面的“与”，即对于导线两端：有零即为零，只有全为一是才为一。理解了这点，按键的工作前提就有了。

键盘分为编码键盘和非编码键盘。键盘上闭合键的识别由专用的硬件编码器实现，并产生键编码号或键值的称为编码键盘，如计算机键盘。而靠软件编程来识别的键盘称为非编码键盘，在单片机组成的各种系统中，用的较多的是非编码键盘。非编码键盘又分为独立键盘和行列式键盘（常说的矩阵键盘）。在这一讲中我们介绍一下单片机中键盘使用。

单片机的 IO 口既可作为输出也可作为输入使用，当检测按键时用的是它的输入功能，我们把按键的一端接地，另一端与单片机的某个 I/O 口相连，开始时先给该 IO 口赋一高电平，然后让单片机不断地检测该 I/O 口是杏变为低电平，当按键闭合时，即相当于该 I/O 口通过按键与地相连，变成低电平，程序一旦检测到 I/O 口变为低电平则说明按键被按下，然后执行相应的指令。

我们先来说一下，按键常常遇到的问题—抖动问题。





还以图四为例，按键未按下之前，图四按键左端的导线因为连在VCC 上而显示高电平，右端显示低电平，按键按下后，按键闭合，整个导线都显示低电平，然后按键松开，又回到按键按下之前的点评状态。如果只考察按键左端的电平变化，应该是上图中所显示的一个负脉冲波形。但是，实际上，正确的波形应该是下图。相比于上图，大家都看到了在高低电平直接有一段锯齿一样的波形，这就是所谓的按键抖动。

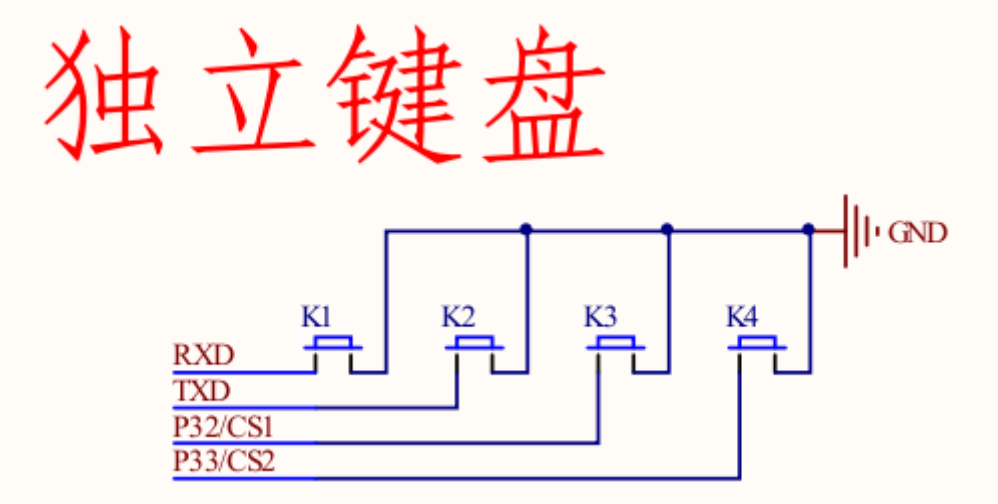
为什么会有按键抖动呢，原因很简单，接触式按键是靠机械的接触来实现开关作用的。这种接触方式就注定了它要经历一个“接触不稳定——正在稳定中——彻底稳定”的一种过程。就好比你用手抓紧一颗石头，即使你一开始就很用力的握紧，也不可能马上就达到最紧的状态，也要经历一个从握住到最紧握的过程。那么在这个过程里，接触式按键就处于一种徘徊在“闭合”与“断开”两者之间的状态。体现在电路中，就是在一小段时间内有非常多的“按下——抬起”动作。而这段抖动的时间，大概是10~20 毫秒，依不同的环境条件而定。

解决这个问题常见的方法有软件去抖动和硬件去抖动。

我们解释一下抖动：关于按键去抖动的解释，我们在手动按键的时候，由于机械抖动或是其它一些非人为的因素很有可能会造成误识别，一般手动按下一次键然后接着释放，按键两片金属膜接触的时间大约为 50ms 左右，在按下瞬间到稳定的时间为 5-10ms,在松开的瞬间到稳定的时间也为 5-10ms，如果我们再首次检测到键被按下后延时 10ms 左右再去检测，这时如果是干扰信号将不会被检测到，如果确实是有键被按下，则可确认，以上为按键识别去抖动的原理。

独立按键：

我们先将一下独立按键的使用方法，开发板独立按键电路图如下：



独立按键一共 5 个，分别连接在单片机的 P3.0 到 P3.4 口。去抖动的方式，我们采用软件延时的方法。过程如下：

1. 先设置 IO 口为高电平（一般上电默认就为高）
2. 读取 IO 口电平确认是否有按键按下
3. 如有 IO 电平为低电平后，延时几个 ms
4. 再读取该 IO 电平，如果任然为低电平，说明对应按键按下
5. 执行相应按键的程序

```
/*********************************************************************************
* 实验名                    : 独立按键实验
* 使用的IO                  : LED使用P2,键盘使用P3.0、P3.1、P3.2、P3.3
* 实验效果                  : 按下K1键，灭掉LED，按下K2键，打开所有的LED，按下K3键，LED左移一位，按下K4键，LED右移一位。
*    注意                                 ：由于P3.2口跟红外线共用，所以做按键实验时为了不让红外线影响实验
*效果，最好把红外线先取下来。
*********************************************************************************/
#include<reg51.h>
#include<intrins.h>

#define GPIO_LED P2
sbit K1=P3^0;
sbit K2=P3^1;
```

```
sbit K3=P3^2;
sbit K4=P3^3;
void Delay10ms( );   //延时10ms

/********************************************************************************
* 函数名         : main
* 函数功能         : 主函数
* 输入           : 无
* 输出           : 无
********************************************************************************/
void main(void)
{
    unsigned int i,j;
    while(1)
    {
        if(K1==0)       //检测按键K1是否按下
        {
            Delay10ms();    //消除抖动
            if(K1==0) //再次检测按键是否按下
                j=0;
            while((i<50)&&(K1==0))   //检测按键松手检测，如果不松手超过延时
也会自动结束等待
            {
                Delay10ms();
                i++;
            }
            i=0;
        }
        if(K2==0)            //检测按键K2是否按下
        {
            Delay10ms();
            if(K2==0)
                j=0xff;
            while((i<50)&&(K2==0))
            {
                Delay10ms();
                i++;
            }
            i=0;
        }
        if(K3==0)               //检测按键K3是否按下
        {
            Delay10ms();
            if(K3==0)
```

```
                {
                    if((j==0)||(j==0xff))        //如果当前状态是全亮的或者全灭
的，就点亮他的第一盏灯
                    {
                        j=0xfe;
                    }
                    else
                        j=_crol_(j,1);           //左移一位
                }
                while((i<50)&&(K3==0))
                {
                    Delay10ms();
                    i++;
                }
                i=0;
            }

            if(K4==0)                    //检测按键K4是否按下
            {
                Delay10ms();
                if(K4==0)
                {
                    if((j==0)||(j==0xff))
                    {
                        j=0xfe;
                    }
                    else
                        j=_cror_(j,1);       //右移一位
                }
                while((i<50)&&(K4==0))
                {
                    Delay10ms();
                    i++;
                }
                i=0;
            }
            GPIO_LED=j;
        }
    }
    /*******************************************************************************
    * 函数名         : Delay10ms
    * 函数功能         : 延时函数，延时10ms
    * 输入           : 无
    * 输出           : 无
```

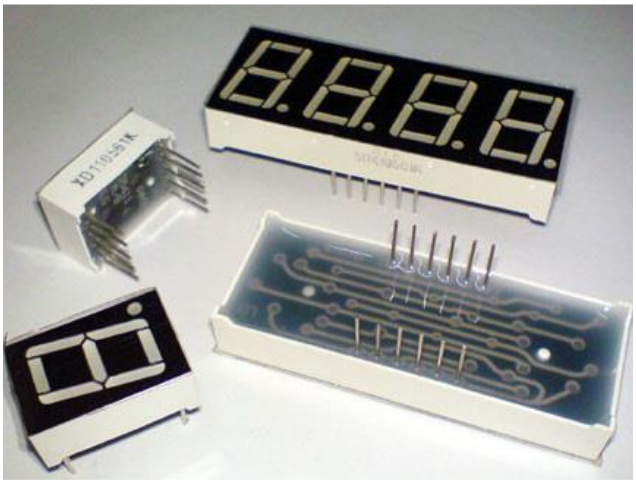
```
*******************************************************************************/
void Delay10ms(void)   //误差 0us
{
    unsigned char a,b,c;
    for(c=1;c>0;c--)
        for(b=38;b>0;b--)
            for(a=130;a>0;a--);
}
```

下载独立按键控制 led 灯.hex，观察实验结果。

实验效果是：按下 K1 键，灭掉 LED，按下 K2 键，打开所有的 LED，按下 K3 键，LED 左移一位，按下 K4 键，LED 右移一位。

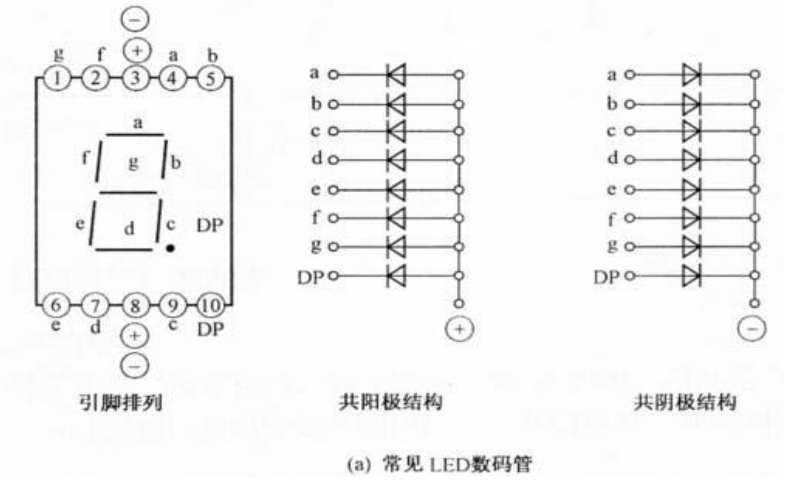
第七讲 静态数码管

我们先看看什么是数码管，



上图就是各种长相各种样子的数码管了，肯定很眼熟了吧。

不管将几位数码管连在一起，数码管的显示原理都是一样的，都是靠点亮内部的发光二极管来发光，下面就来我们讲解一个数码管是如何亮起来的。数码管内部电路如下图所示，从右图可看出，一位数码管的引脚是 10 个，显示一个 8 字需要 7 个小段，另外还有一个小数点，所以其内部一共有 8 个小的发光二极管，最后还有一个公共端，生产商为了封装统一，单位数码管都封装 10 个引脚，其中第 3 和第 8 引脚是连接在一起的。而它们的公共端又可分为共阳极和共阴极，中间图为共阴极内部原理图，右图为共阳极内部原理图。



(a) 常见 LED数码管

上图展出了常用的两种数码管的引脚排列和内部结构。总所周知，点亮发光二极管就是要给予它足够大的正向压降。所以点亮数码管其实也就是给它内部相应的发光二极管正向压降。如上图左（一共a、b、c、d、e、f、g、DP 八段），如果要显示“1”则要点亮b、c 两段LED；显示“A”则点亮a、b、c、e、f、g 这六段LED；我们还知道，既然LED 加载的是正向压降，它的两端电压必然会有高低之分：如果八段LED 电压高的一端为公共端，我们称之为共阳极数码管（如上图中）；如果八段LED 电压低的一段为公共端，则称之为共阴极数码管（上图右）。所以，要点亮共阳极数码管，则要在公共端给予高于非公共端的电平；反之点亮共阴极数码管，则要在非公共端给予较高电平。

对共阴极数码来说，其8个发光二极管的阴极在数码管内部全部连接在一起，所以称“共阴”，而它们的阳极是独立的，通常在设计电路时一般把阴极接地。当我们给数码管的任意一个阳极加一个高电平时，对应的这个发光二极管就点亮了。如果想要显示出一个8字，并且把右下角的小数点也点亮的话，可以给8个阳极全部送高电平，如果想让它显示出一个0字，那么我们可以除了给第“g, dp”这两位送低电平外，其余引脚全部都送高电平，这样它就显示出0字了。想让它显示几，就给相对应的发光二极管送高电平，因此我们在显示数字的时候首先做的就是给0-9十个数字编码，在要它亮什么数字的时候直接把这个编码送到它的阳极就行了。

共阳极数码管其内部8个发光二极管的所有阳极全部连接在一起，电路连接时，公共端接高电平，因此我们要点亮的那个发光管二极管就需要给阴极送低电平，此时显示数字的编码与共阳极编码是相反的关系，数码管内部发光二极管点亮时，也需要5mA以上的电流，而且电流不可过大，否则会烧毁发光二极管。由于单片机的I/O口送不出如此大的电流，所以数码管与单片机连接时需要加驱动电路，可以用上拉电阻的方法或使用专门的数码管驱动芯片，本实验板上使用的是74HC573锁存器，其输出电流较大，电路接口简单，可借鉴使用。

一般共阳极数码管更为常用，为什么呢？这是因为数码管的非公共端往往接在IC 芯片的IO上，而IC 芯片的驱动能力往往是比较小的，如果采用共阴极数码管，它的驱动端在非公共端，就有可能受限于IC芯片输出电流不够而显示昏暗（比如51单片机），要外加上拉电阻或者是三极管加大驱动能力。所以使用共阳数码

管的好处是：将驱动数码管的工作交到公共端（一般接驱动电源），加大驱动电源的功率自然要比加大IC芯片IO口的驱动电流简单许多。另一方面，这样也能减轻MCU的负担。

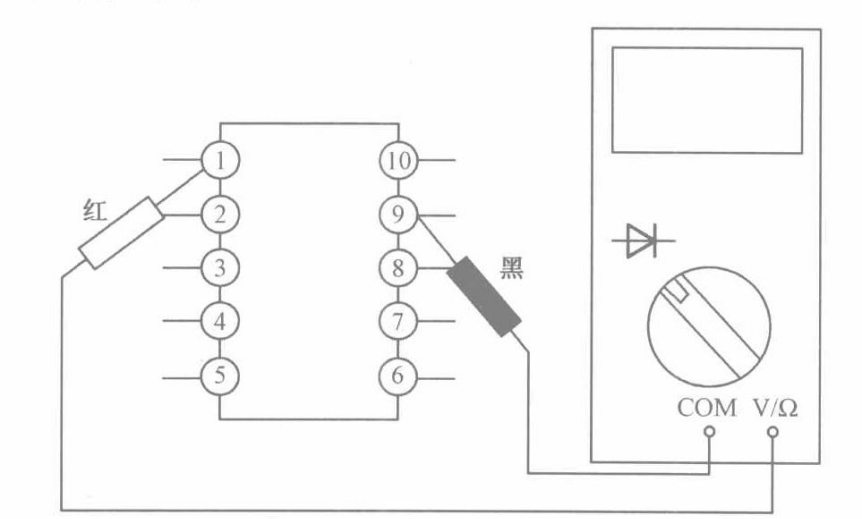
当多位一体时，它们内部的公共端是独立的，而负责显示什么数字的段线全部是连接在一起的，独立的公共端可以控制多位一体中的哪一位数码管点亮，而连接在一起的段线可以控制这个能点亮数码管亮什么数字，通常我们把公共端叫做“位选线”，连接在一起的段线叫做“段选线”，有了这两个线后，通过单片机及外部驱动电路就可以控制任意的数码管显示任意的数字了。

一般单位数码管有10个引脚，二位数码管也是10个引脚，四位数码管是12个引脚，关于具体的引脚及段、位标号大家可以查询相关资料，最简单的办法就是用数字万用表测量，若没有数字万用表也可用5V直流电源串接1k电阻后测量，将测量结果记录，通过统计便可绘制出引脚标号。

知识点:如何用万用表检浏数码管的引脚排列

对数字万用表来说，红色表笔连接表内部电池正极，黑色表笔连接表内部电池负极，当把数字万用表置于二极管档时，其两表笔间开路电压约为1.5V，把两表笔正确加在发光二极管两端时，可以点亮发光二极管。

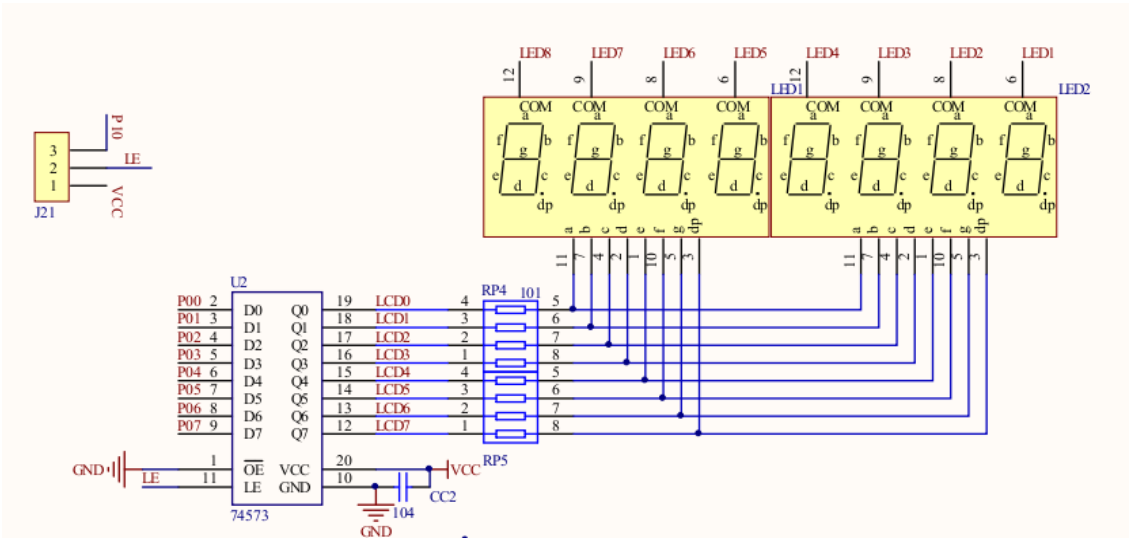
如下图所示，将数字万用表置于二极管挡，红表笔接在①脚，然后用黑表笔去接触其他各引脚，假设只有当接触到⑨脚时，数码管的a段发光，而接触其余引脚时则不发光。由此可知，被测数码管为共阴极结构类型，⑨脚是公共阴极，①脚则是数码管的a段.接下来再检测各段引脚，仍使用数字万用表二极管档，将黑表笔固定接在⑨脚，用红表笔依次接触②、③、④、⑤、⑥、⑦、⑧、⑩引脚时，数码管的其他段先后分别发光，据此便可绘出该数码管的内部结构和引脚排列图。



检测中，若被测数码管为共阳极类型，则需将红、黑表笔对调才能测出上述结果，在判别结构类型时，操作时要灵活掌握，反复试验，直到找出公共端为止，大家只要懂得了原理，检测出各个引脚便不在是问题了。

数码管静态显示

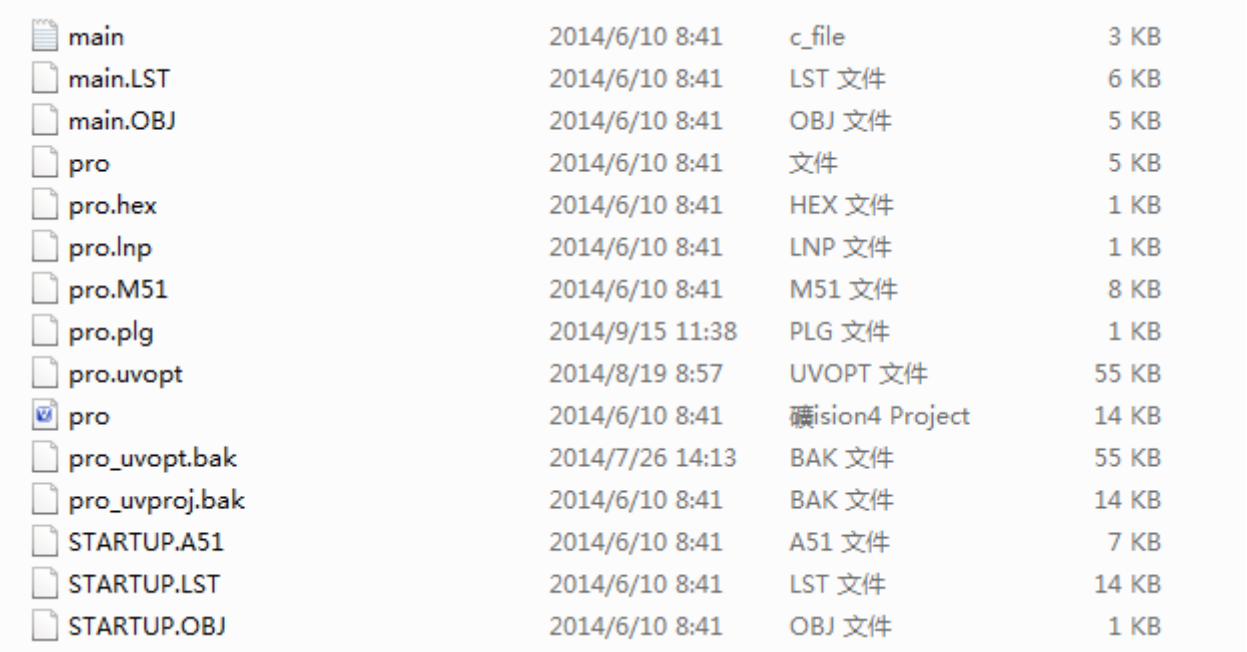
当多位数码管应用于某一系统时，它们的“位选”是可独立控制的，而“段选”是连接在一起的，我们可以通过位选信号控制哪几个数码管亮，而在同一时刻，位选选通的所有数码管上显示的数字始终都是一样的，因为它们的段选是连接在一起的，所以送入所有数码管的段选信号都是相同的，那么它们显示的数字必定一样，数码管的这种显示方法叫做静态显示。



从电路图可以看出，本开发板使用的是共阴极数码管，在每段数码管端加上一个限流电阻。

下面我们来看一下程序

打　　开 



main	2014/6/10 8:41	c_file	3 KB
main.LST	2014/6/10 8:41	LST 文件	6 KB
main.OBJ	2014/6/10 8:41	OBJ 文件	5 KB
pro	2014/6/10 8:41	文件	5 KB
pro.hex	2014/6/10 8:41	HEX 文件	1 KB
pro.lnp	2014/6/10 8:41	LNP 文件	1 KB
pro.M51	2014/6/10 8:41	M51 文件	8 KB
pro.plg	2014/9/15 11:38	PLG 文件	1 KB
pro.uvopt	2014/8/19 8:57	UVOPT 文件	55 KB
pro	2014/6/10 8:41	礦ision4 Project	14 KB
pro_uvopt.bak	2014/7/26 14:13	BAK 文件	55 KB
pro_uvproj.bak	2014/6/10 8:41	BAK 文件	14 KB
STARTUP.A51	2014/6/10 8:41	A51 文件	7 KB
STARTUP.LST	2014/6/10 8:41	LST 文件	14 KB
STARTUP.OBJ	2014/6/10 8:41	OBJ 文件	1 KB

```
/*********************************************************************************
* 实验名                    : 静态数码管实验
* 使用的IO          : 数码管使用P0,键盘使用P3.0、P3.1、P3.2、P3.3
* 实验效果          : 按下K1键，显示1，按下K2键，显示2，按下K3键，显示3，按下K4键，显示4。
*    注意                          ：由于P3.2口跟红外线共用，所以做按键实验时为了不让红外线影响实验
*效果，最好把红外线先取下来。
*********************************************************************************/
#include<reg51.h>
#include<intrins.h>

#define GPIO_DIG P0

sbit LSA=P2^2;
sbit LSB=P2^3;
sbit LSC=P2^4;

sbit K1=P3^0;
sbit K2=P3^1;
sbit K3=P3^2;
sbit K4=P3^3;

DIG_CODE[10]={0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f};// 显示0~9的值
```

```
void Delay10ms();   //延时10ms

/********************************************************************************
* 函数名         : main
* 函数功能         : 主函数
* 输入           : 无
* 输出           : 无
********************************************************************************/
void main(void)
{
    unsigned int i,j;
    LSA=0;
    LSB=0;
    LSC=0;
    while(1)
    {
        if(K1==0)       //检测按键K1是否按下
        {
            Delay10ms();    //消除抖动
            if(K1==0)
            {
                j=1;
            }
            while((i<50)&&(K1==0))   //检测按键是否松开
            {
                Delay10ms();
                i++;
            }
            i=0;
        }
        if(K2==0)            //检测按键K2是否按下
        {
            Delay10ms();
            if(K2==0)
            {
                j=2;
            }
            while((i<50)&&(K2==0))
            {
                Delay10ms();
                i++;
            }
            i=0;
        }
```

```
        if(K3==0)                //检测按键K3是否按下
        {
            Delay10ms();
            if(K3==0)
            {
                j=3;
            }
            while((i<50)&&(K3==0))
            {
                Delay10ms();
                i++;
            }
            i=0;
        }

        if(K4==0)                    //检测按键K4是否按下
        {
            Delay10ms();
            if(K4==0)
            {
                j=4;
            }
            while((i<50)&&(K4==0))
            {
                Delay10ms();
                i++;
            }
            i=0;
        }
        GPIO_DIG=DIG_CODE[j];
    }
}
/********************************************************************************
* 函数名         : Delay10ms
* 函数功能         : 延时函数，延时10ms
* 输入           : 无
* 输出           : 无
********************************************************************************/
void Delay10ms(void)   //误差 0us
{
    unsigned char a,b,c;
    for(c=1;c>0;c--)
        for(b=38;b>0;b--)
            for(a=130;a>0;a--);
```

```
}
```

下载HEX文件，观察实验效果，实验的效果是：按下K1键，显示1，按下K2键，显示2，按下K3键，显示3，按下K4键，显示4。

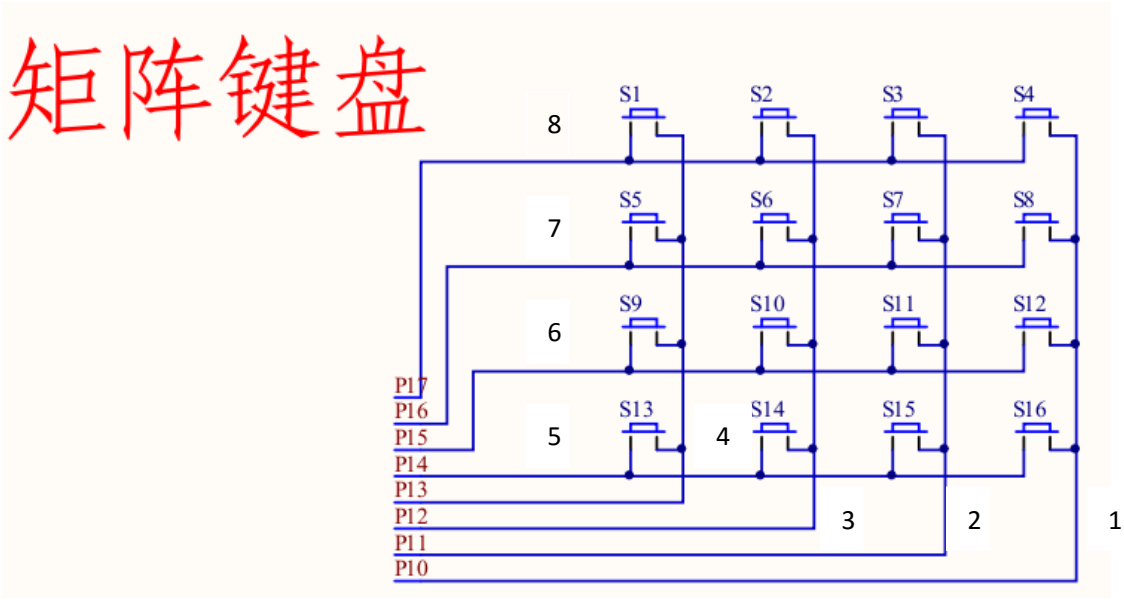
第八讲 矩阵键盘

独立键盘与单片机连接时，每一个按键都需要单片机的一个 I/O 口若某单片机系统需较多按键，如果用独立按键便会占用过多的 I/O 口资源。单片机系统中 I/O 口资源往往比较宝贵，当用到多个按键时为了节省 I/O 口口线，我们引入矩阵键盘。

我们以 4X4 矩阵键盘为例讲解其工作原理和检测方法。将 16 个按键排成 4 行 4 列，第一行将每个按键的一端连接在一起构成行线，第一列将每个按键的另一端连接在一起构成列线，这样便一共有 4 行 4 列共 8 根线，我们将这 8 根线连接到单片机的 8 个 I/O 口上，通过程序扫描键盘就可检测 16 个键。用这种方法我们也可实现 3 行 3 列 9 个键、5 行 5 列 25 个键、6 行 6 列 36 个键等。

无论是独立键盘还是矩阵键盘，单片机检测其是否被按下的依据都是一样的，也就是检测与该键对应的 I/O 口是否为低电平。独立键盘有一端固定为低电平，单片机写程序检测时比较方便。而矩阵键盘两端都与单片机 I/O 口相连，因此在检测时需人为通过单片机 I/O 口送出低电平。检测时，先送一列为低电平，其余几列全为高电平(此时我们确定了列数)，然后立即轮流检测一次各行是否有低电平，若检测到某一行为低电平(这时我们又确定了行数)，则我们便可确认当前被按下的键是哪一行哪一列的，用同样方法轮流送各列一次低电平，再轮流检测一次各行是否变为低电平，这样即可检测完所有的按键，当有键被按下时便可判断出按下的键是哪一个键。当然我们也可以将行线置低电平，扫描列是否有低电平。这就是矩阵键盘检测的原理和方法。

首先看一下电路图

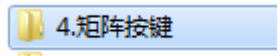


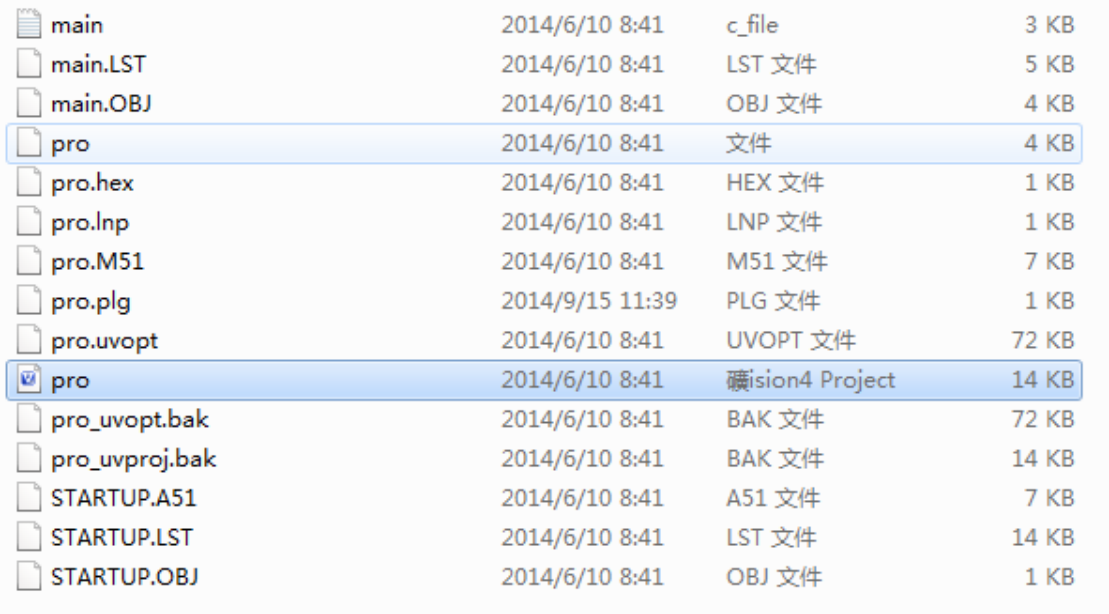
上图是一个4X4 的矩阵键盘，一共是16 个按键。我们照习惯称横为“行”，“竖”为列。那么5、6、7、8 我们称之为“行线”，则1、2、3、4 称为“列线”。要正确记住各个行列线各自对应的IO。注意看，每一个按键的两端，都分别接在某一个列线和行线上，即：“行线和列线是通过某个按键的按下和抬起实现联通和断开的”，和“导线两端上的信号是经过“与”的关系再体现到导线上的。”这两句话便构成了矩阵键盘扫描的全部。要理解好，理解不了就背下来。

现在详细讲述一下矩阵键盘扫描的原理和步骤：

扫描矩阵键盘，即是把某一条（只有一条）行线置为低电平，而列线全部置为输入方向，然后检测列线，如果检测到某一条列线是低电平，那么就表示位于这条列线与输出低电平的行线的交点处的按键被按下了。要扫描16个按键，就依次以这样的方法扫描16次，之后就可以确定哪一个按键被按下了。当然这里也少不了延时消除按键抖动的环节。

下面看一下程序

打开 



main	2014/6/10 8:41	c_file	3 KB
main.LST	2014/6/10 8:41	LST 文件	5 KB
main.OBJ	2014/6/10 8:41	OBJ 文件	4 KB
pro	2014/6/10 8:41	文件	4 KB
pro.hex	2014/6/10 8:41	HEX 文件	1 KB
pro.lnp	2014/6/10 8:41	LNP 文件	1 KB
pro.M51	2014/6/10 8:41	M51 文件	7 KB
pro.plg	2014/9/15 11:39	PLG 文件	1 KB
pro.uvopt	2014/6/10 8:41	UVOPT 文件	72 KB
pro	2014/6/10 8:41	礦ision4 Project	14 KB
pro_uvopt.bak	2014/6/10 8:41	BAK 文件	72 KB
pro_uvproj.bak	2014/6/10 8:41	BAK 文件	14 KB
STARTUP.A51	2014/6/10 8:41	A51 文件	7 KB
STARTUP.LST	2014/6/10 8:41	LST 文件	14 KB
STARTUP.OBJ	2014/6/10 8:41	OBJ 文件	1 KB

```
/*********************************************************************************
* 实验名                    : 矩阵键盘实验
* 使用的IO          : 数码管使用P0,键盘使用P3.0、P3.1、P3.2、P3.3
* 实验效果          : 按矩阵键盘分别显示在数码管上面显示十六进制的0到F。
*    注意                          :
*********************************************************************************/
#include<reg51.h>

#define GPIO_DIG P0
#define GPIO_KEY P1

sbit LSA=P2^2;
sbit LSB=P2^3;
sbit LSC=P2^4;

unsigned char code DIG_CODE[17]={
0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,
0x7f,0x6f,0x77,0x7c,0x39,0x5e,0x79,0x71};
//0、1、2、3、4、5、6、7、8、9、A、b、C、d、E、F的显示码
unsigned char KeyValue;
//用来存放读取到的键值
void Delay10ms();   //延时10ms
void KeyDown();      //检测按键函数

/*********************************************************************************
* 函数名          : main
* 函数功能            : 主函数
* 输入            : 无
* 输出            : 无
```

```
*******************************************************************************/
void main(void)
{
    LSA=0; //给一个数码管提供位选
    LSB=0;
    LSC=0;
    while(1)
    {
        KeyDown();
        GPIO_DIG=DIG_CODE[KeyValue];
    }
}
/*******************************************************************************
* 函数名         : KeyDown
* 函数功能         : 检测有按键按下并读取键值
* 输入           : 无
* 输出           : 无
*******************************************************************************/
void KeyDown(void)
{
    char a;
    GPIO_KEY=0x0f;
    if(GPIO_KEY!=0x0f)
    {
        Delay10ms();
        if(GPIO_KEY!=0x0f)
        {

            //测试列
            GPIO_KEY=0X0F;
            switch(GPIO_KEY)
            {
                case(0X07): KeyValue=0;break;
                case(0X0b): KeyValue=1;break;
                case(0X0d): KeyValue=2;break;
                case(0X0e): KeyValue=3;break;
//              default:    KeyValue=17;    //检测出错回复17意思是把数码管全灭
掉。
            }
            //测试行
            GPIO_KEY=0XF0;
            switch(GPIO_KEY)
            {
                case(0X70): KeyValue=KeyValue;break;
```

```
                case(0Xb0): KeyValue=KeyValue+4;break;
                case(0Xd0): KeyValue=KeyValue+8;break;
                case(0Xe0): KeyValue=KeyValue+12;break;
//              default:    KeyValue=17;
            }
            while((a<50)&&(GPIO_KEY!=0xf0))  //检测按键松手检测
            {
                Delay10ms();
                a++;
            }
            a=0;
        }
    }
}
/*********************************************************************************
* 函数名         : Delay10ms
* 函数功能         : 延时函数，延时10ms
* 输入           : 无
* 输出           : 无
*********************************************************************************/
void Delay10ms(void)   //误差 0us
{
    unsigned char a,b,c;
    for(c=1;c>0;c--)
        for(b=38;b>0;b--)
            for(a=130;a>0;a--);
}
```

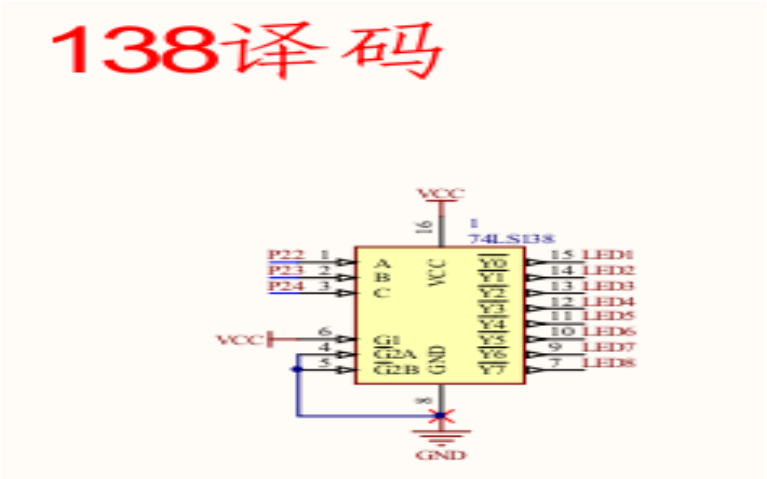
下载HEX文件，观察实验效果，实验的效果是：按矩阵键盘分别显示在数码管上面显示十六进制的0到F。

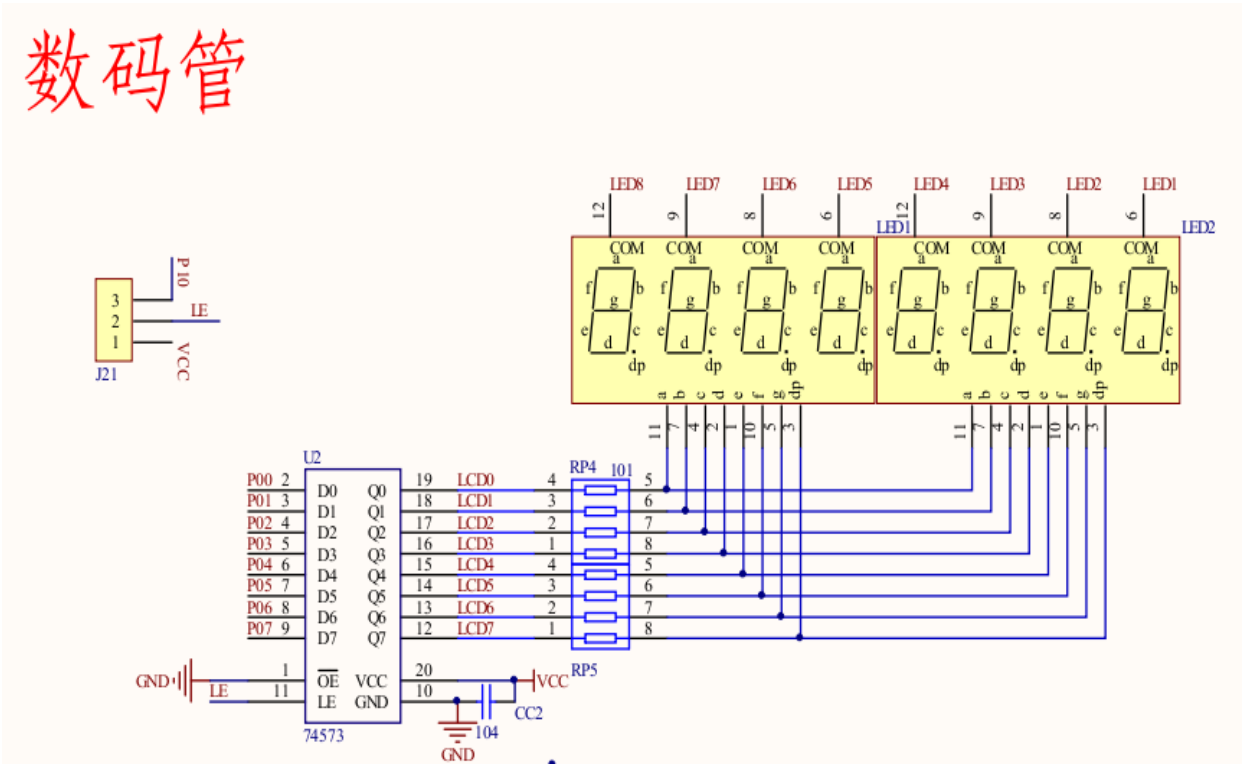
第九讲 动态数码管

1. 动态扫描的原理

在实际的单片机系统中，往往需要多位显示。动态显示是一种最常见的多位显示方法，应用非常广泛。所有数码管段选都连接在一起的时候，怎么让数码管显示不一样的数字呢？动态显示是多个数码管，交替显示，利用人的视觉暂停作用使人看到多个数码管同时显示的效果。

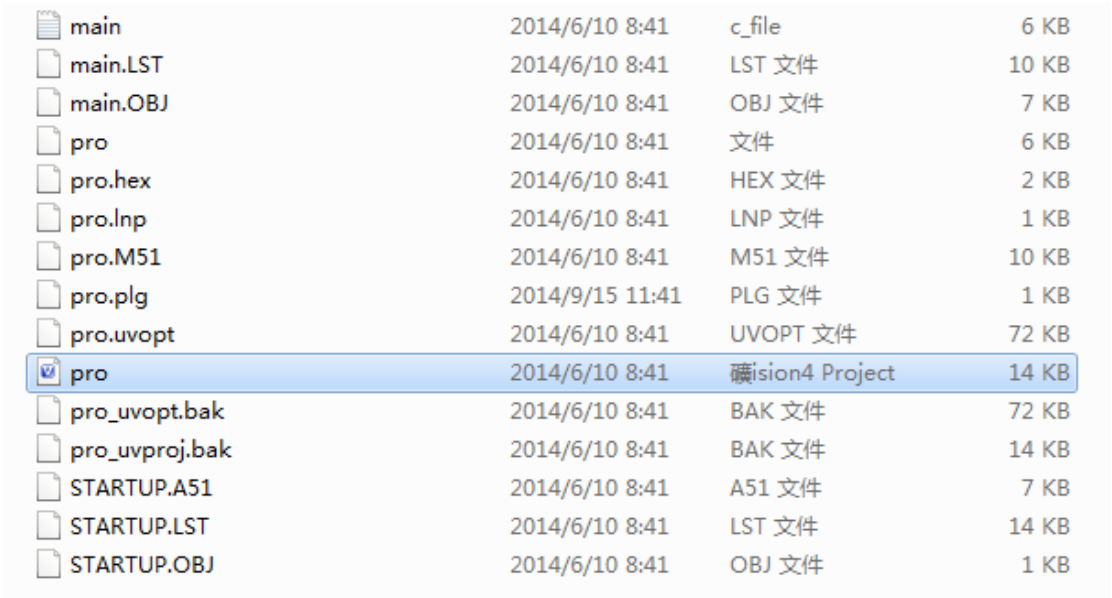
首先我们来看一下开发板上的电路原理图：





本开发板上使用的是，通过P22、P23、P24控制3-8译码器来对数码管进行位选，通过P0口经过573的驱动控制数码管的段选，通过P13控制573的使能端，为低电平时573才会有输出。

打开 5.动态数码管 中断显示



下面看一下程序：

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

```
*****
    * 实 验 名        : 动态显示数码管实验
    * 使用的IO       : 数码管使用P0,P2.2,P2.3,P2.4键盘使用P1
    * 实验效果       : 按矩阵键盘分别显示在数码管上面显示十六进制的0到F。
    * 注    意       :
    ***********************************************************************
****/
    #include<reg51.h>
    //#include<intrins.h>

    #define GPIO_DIG P0
    #define GPIO_KEY P1

    sbit LSA=P2^2;
    sbit LSB=P2^3;
    sbit LSC=P2^4;

    unsigned char code DIG_CODE[17]={
    0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,
    0x7f,0x6f,0x77,0x7c,0x39,0x5e,0x79,0x71};
    //0、1、2、3、4、5、6、7、8、9、A、b、C、d、E、F的显示码
    unsigned char KeyValue;
    //用来存放读取到的键值
    unsigned char KeyState; //记录按键的状态，0没有，1有
    unsigned char DisplayData[8];
    //用来存放要显示的8位数的值
    unsigned char Num;//用来存放中断的时候显示的第位数值
    void Delay50us();   //延时50us
    void KeyDown();      //检测按键函数
    void DigDisplay(); //动态显示函数
```

```
    void TimerConfiguration();//定时器初始化设置
    /*****************************************************************************
*****
    * 函 数 名         : main
    * 函数功能         : 主函数
    * 输    入         : 无
    * 输    出         : 无
    *****************************************************************************
****/
    void main(void)
    {
        TimerConfiguration();
        KeyState=0;  //初始化按键状态
        while(1)
        {
            KeyDown();
            if(KeyState==1)
            {
                DisplayData[7]=DisplayData[6];
                DisplayData[6]=DisplayData[5];
                DisplayData[5]=DisplayData[4];
                DisplayData[4]=DisplayData[3];
                DisplayData[3]=DisplayData[2];
                DisplayData[2]=DisplayData[1];
                DisplayData[1]=DisplayData[0];
                DisplayData[0]=DIG_CODE[KeyValue];
                KeyState=0;
            }
    //      DigDisplay();
        }
```

```
    }
    /*************************************************************************
*****
    * 函 数 名         : TimerConfiguration
    * 函数功能         : 定时器初始化
    * 输    入         : 无
    * 输    出         : 无
    *************************************************************************
****/
    void TimerConfiguration()
    {
        TMOD=0X02;//选择为定时器0模式，工作方式2，仅用TRX打开启动。

        TH0=0X9C;   //给定时器赋初值，定时100us
        TL0=0X9C;
        ET0=1;//打开定时器0中断允许
        EA=1;//打开总中断
        TR0=1;//打开定时器
    }
    /*************************************************************************
*****
    * 函 数 名         : DigDisplay
    * 函数功能         : 使用数码管显示
    * 输    入         : 无
    * 输    出         : 无
    *************************************************************************
****/
    void DigDisplay()
    {
        unsigned char i,j;
```

```
//  for(i=0;i<8;i++)
//  {
        GPIO_DIG=0x00;//消隐
        switch(i)    //位选，选择点亮的数码管，
        {
            case(0):
                LSA=0;LSB=0;LSC=0; break;
            case(1):
                LSA=1;LSB=0;LSC=0; break;
            case(2):
                LSA=0;LSB=1;LSC=0; break;
            case(3):
                LSA=1;LSB=1;LSC=0; break;
            case(4):
                LSA=0;LSB=0;LSC=1; break;
            case(5):
                LSA=1;LSB=0;LSC=1; break;
            case(6):
                LSA=0;LSB=1;LSC=1; break;
            case(7):
                LSA=1;LSB=1;LSC=1; break;
        }
        GPIO_DIG=DisplayData[i];
        i++;
        if(i>7)
            i=0;
//      j=10;                           //扫描间隔时间设定
//      while(j--)
//          Delay50us();
//      GPIO_DIG=0x00;//消隐
```

```
    //  }
    }
    /*************************************************************************
*****
    * 函 数 名         : KeyDown
    * 函数功能         : 检测有按键按下并读取键值
    * 输    入         : 无
    * 输    出         : 无
    *************************************************************************
****/
    void KeyDown(void)
    {
        unsigned int a=0;
        GPIO_KEY=0x0f;
        if(GPIO_KEY!=0x0f)
      {
            Delay50us();
            a++;
            a=0;
            if(GPIO_KEY!=0x0f)
            {
                ET0=0;//关定时器中断
                KeyState=1;//有按键按下
                //测试列
                GPIO_KEY=0X0F;
    //          Delay50us();
                switch(GPIO_KEY)
                {
                    case(0X07): KeyValue=0;break;
                    case(0X0b): KeyValue=1;break;
```

```
                    case(0X0d): KeyValue=2;break;
                    case(0X0e): KeyValue=3;break;
    //              default:    KeyValue=17;    //检测出错回复17意思是把数码管
全灭掉。
                }
                //测试行
                GPIO_KEY=0XF0;
                Delay50us();
                switch(GPIO_KEY)
                {
                    case(0X70): KeyValue=KeyValue;break;
                    case(0Xb0): KeyValue=KeyValue+4;break;
                    case(0Xd0): KeyValue=KeyValue+8;break;
                    case(0Xe0): KeyValue=KeyValue+12;break;
    //              default:    KeyValue=17;
                }
                ET0=1;//开定时器中断
                while((a<5000)&&(GPIO_KEY!=0xf0))    //检测按键松手检测
                {
                    Delay50us();
                    a++;
                }
                a=0;
            }
        }
    }
    /**********************************************************************
*****
    * 函 数 名         : Delay50us
    * 函数功能         : 延时函数，延时50us
```

```
    * 输    入          : 无
    * 输    出          : 无
    ********************************************************************************
****/
    void Delay50us(void)   //延时50us误差 0us
    {
        unsigned char a,b;
        for(b=1;b>0;b--)
            for(a=22;a>0;a--);
    }
    /********************************************************************************
*****
    * 函 数 名          : Delay50us
    * 函数功能          : 延时函数，延时50us
    * 输    入          : 无
    * 输    出          : 无
    ********************************************************************************
****/
    void Timer() interrupt 1
    {
        DigDisplay();
    }
```

在用C语言编程时，编码定义方法如下:

```
unsigned char code DIG_CODE[17]={0x3f,0x06,0x5b,0x4f,0x66,0x6d,
0x7d,0x07,0x7f,0x6f,0x77,0x7c,0x39,0x5e,0x79,0x71};//0、1、2、3、4、5、
6、7、8、9、A、b、C、d、E、F的显示码
```

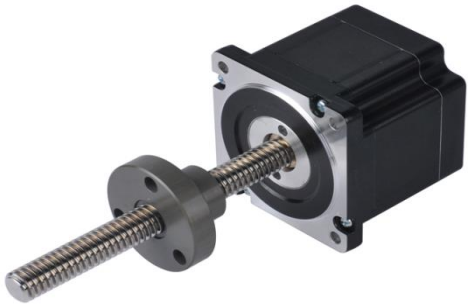
编码定义方法与C语言中的数组定义方法非常相似，不同的地方就是在数组类型后面多了一个code关键字，code即表示编码的意思。需要注意的是，单片机C语言中定义数组时是占用内存空间的，而定义编码时是直接分配到程序空间中，编译后编码占用的是程序存储空间，而非内存空间。

第十讲 电机

1. 步进电机图片

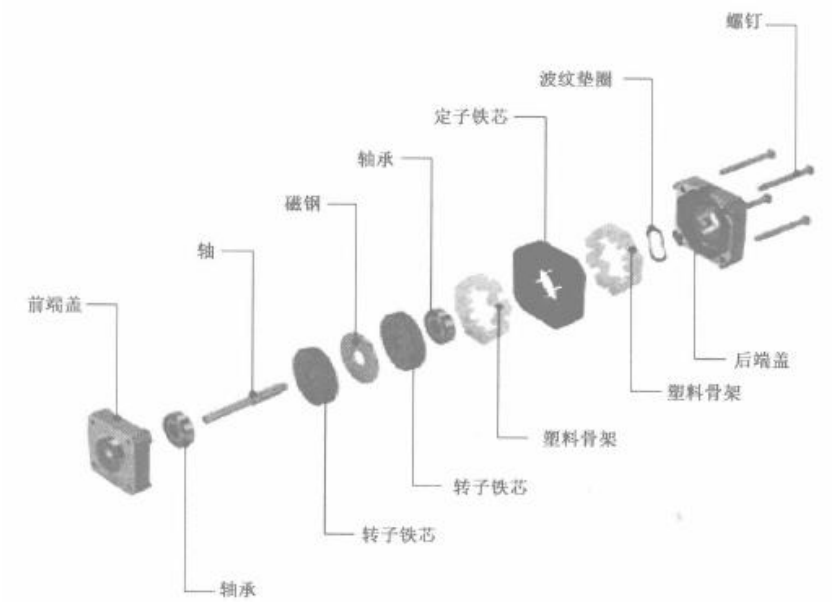
普通步进电机

减速步进电机



直线步进电机

微型步进电机



步进电机剖视图

2.步进电机介绍

步进电机是将电脉冲信号转变为角位移或线位移的开环控制元件。在非超载情况下，电机的转速、停止的位置只取决于脉冲信号的频率和脉冲数，而不受负载变化的影响，即给电机加一个脉冲信号，电机则转过一个步距角。这一线性关系的存在，加上步进电机只有周期性的误差而无累积误差等特点，使得步进电机在速度、位置等控制领域的控制操作非常简单。虽然步进电机应用广泛，但它并不像普通的直流和交流电机那样在常规状态下使用，它必须由双环形脉冲信号、功率驱动电路等组成控制系统方可使用.因此.用好步进电机也非易事，它涉及机械、电机、电子及计算机等许多专业知识。

3.步进电机分类

(1)永磁式(PM)。一般为二相，转矩和体积较小，步距角一般为 7.5° 或 15° 。

(2)反应式(VR)。一般为三相，可实现大转矩输出.步距角一般为 1.5° ，但噪声和振动都很大。在欧美等发达国家 20 世纪 80 年代已经淘汰。

(3)混合式(HB)。指混合了永磁式和反应式的优点。它又分为二相和五相，二相步距角一般为 180° ，而五相步距角一般为 0.72° 。这种步进电机的应用最为广泛。

4.技术指标

(1)步进电机的静态指标

①相数—电机内部的线圈组数。目前常用的有二相、三相、四相、五相步进电机。电机相数不同，其步距角也不同。一般二相电机的步距角为 0.9°/1.8°，三相为 0.75°/1.5°、五相为 0.36°/0.72°。在没有细分驱动器时，用户主要靠选择不同相数的步进电机来满足自己步距角的要求.如果使用细分驱动器，则“相数”将变得没有意义，用户只需在驱动器上改变细分数，就可以改变步距角。

②步距角—表示控制系统每发一个步进脉冲信号，电机所转动的角度。电机出厂时给出了一个步距角的值，如 86BYG250A 型电机的值为 0.9°/1.8°（表示半步工作时为 0.9°、整步工作时为 1.8°），这个步距角可称为“电机固有步距角”，它不一定是电机实际工作时的真正步距角，真正的步距角和驱动器有关。

③拍数—完成一个磁场周期性变化所需脉冲数或导电状态.或指电机转过一个步距角所需脉冲数。以四相电机为例,有四相四拍运行方式,即 AB-BC-CD-DA-AB:四相八拍运行方式，即 A-AB-B-BC-C-CD-D-DA-A

④定位转矩—电机在不通电状态下，转子自身的锁定力矩(由磁场齿形的谐波以及机械误差造成).

⑥保持转矩—步进电机通电但没有转动时，定子锁住转子的力矩。它是步进电机最重要的参数之一，通常步进电机在低速时的力矩接近保持转矩。由于步进电机的输出力矩随速度的增大而不断衰减，输出功率也随速度的增大而变化，所以保持转矩就成了衡量步进电机最重要参数之一。比如，当人们说 2N·m 的步进电机时，在没有特殊说明的情况下，是指保持转矩为 2N·m 的步进电机。

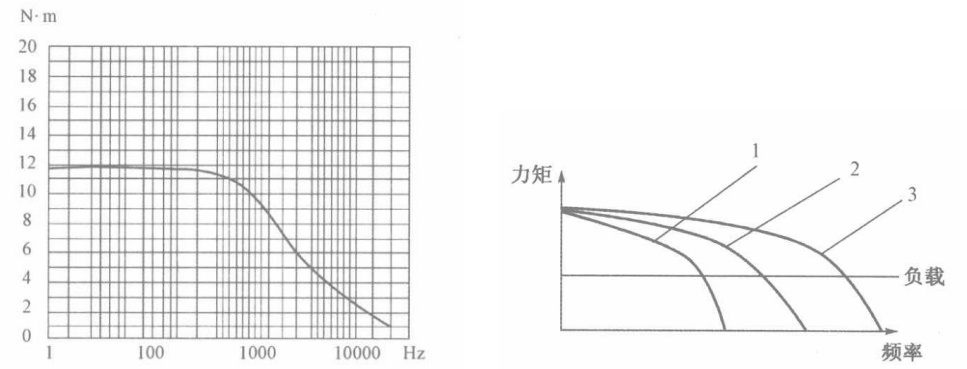
(2)步进电机的动态指标

①步距角精度—步进电机每转过一个步距角的实际值与理论值的误差,用百分比表示:误差/步距角 X100%。不同运行拍数其值不同,四拍运行时应在 5%之内,八拍运行时应在 15%以内。

②失步—电机运转时运转的步数不等于理论上的步数，称为失步。

③失调角—转子齿轴线偏移定子齿轴线的角度.电机运转必存在失调角，由失调角产生的误差，采用细分驱动是不能解决的。最大空载起动频率—电机在某种驱动形式、电压及额定电流下，在不加负载的情况下，能够直接起动的最大频率。最大空载运行频率—电机在某种驱动形式、电压及额定电流下，电机不带负载的最高转速频率。运行矩频特性—电机在某种测试条件下，测得运行中输出力

矩与频率关系的曲线称为运行矩频特性。它是电机诸多动态曲线中最重要的，也是电机选择的根本依据，如图所示。电机一但选定，电机的静力矩确定，而动态力矩却不然.电机的动态力矩取决于电机运行时的平均电流(而非静态电流)，平均电流越大，电机输出力矩越大.即电机的频率特性越硬。



力矩与频率关系曲线

其中.曲线 3 电流最大或电压最高;曲线 I 电流最小或电压最低，曲线与负载的交点为负载的最大速度点。要使平均电流大，尽可能提高驱动电压，或采用小电感大电流的电机。

④电机的共振点—步进电机均有固定的共振区域，其共振区一般在 50r/min-80r/min 或在 180r/min 左右。电机驱动电压越高，电机电流越大，负载越轻，电机体积越小。则共振区向上偏移，反之亦然。为使电机输出电矩大、不失步且整个系统的噪声降低，一般工作点均应偏移共振区较多。因此，在使用步进电机时应避开此共振区。

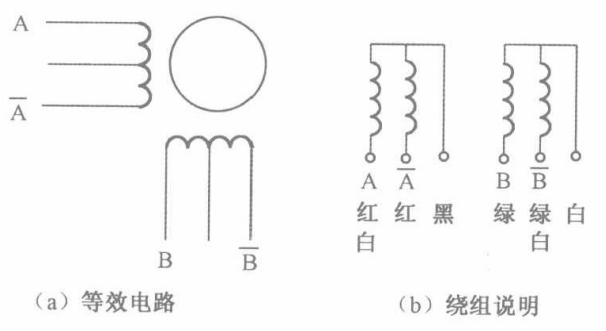
5.步进电机工作原理

步进电机是一种将电脉冲转换成相应角位移或线位移的电磁机械装置。它具有快速启、停能力，在电机的负荷不超过它能提供的动态转矩时，可以通过输入脉冲来控制它在一瞬间的启动或停止。步进电机的步距角和转速只和输入的脉冲频率有关，和环境温度、气压、振动无关，也不受电网电压的波动和负载变化的影响。因此，步进电机多应用在需要精确定位的场合。

(1)工作原理

步进电机有三线式、五线式和六线式，但其控制方式均相同，都要以脉冲信号电流来驱动.假设每旋转一圈需要 200 个脉冲信号来励磁，可以计算出每个励磁信号能使步进电机前进 1.8° 。其旋转角度与脉冲的个数成正比。步进电动机

的正、反转由励磁脉冲产生的顺序来控制。六线式四相步进电机是比较常见的，它的控制等效电路如下图所示。它有 4 条励磁信号引线 A,/A，B,/B 通过控制这 4 条引线上励磁脉冲产生的时刻，即可控制步进电机的转动.每出现一个脉冲信号，步进电机只走一步。因此，只要依序不断送出脉冲信号，步进电机就能实现连续转动。



（a）等效电路　　（b）绕组说明

(2)励磁方式

步进电机的励磁方式分为全步励磁和半步励磁两种。其中全步励磁又有一相励磁和二相励磁之分;半步励磁又称一二相励磁。假设每旋转一圈需要 200 个脉冲信号来励磁，可以计算出每个励磁信号能使步进电动机前进 1.8° 。简要介绍如下。

①一相励磁—在每一瞬间，步进电机只有一个线圈导通.每送一个励磁信号，步进电机旋转 1.8，这是三种励磁方式中最简单的一种。

其特点是:精确度好、消耗电力小，但输出转矩最小，振动较大。如果以该方式控制步进电机正转，对应的励磁顺序如下表所示。若励磁信号反向传送，则步进电机反转。表中的 1 和 0 表示送给电机的高电平和低电平。

一相励磁顺序表

STEP	A	B	$\overline{A}$	$\overline{B}$
1	1	0	0	0
2	0	1	0	0
3	0	0	1	0
4	0	0	0	1

②二相励磁—在每一瞬间，步进电动机有两个线圈同时导通。每送一个励磁信号，步进电机旋转 1.8。

其特点是:输出转矩大，振动小，因而成为目前使用最多的励磁方式。如果以该方式控制步进电机正转，对应的励磁顺序见下表。若励磁信号反向传送，则

步进电机反转。

二相励磁顺序表

STEP	A	B	$\overline{A}$	$\overline{B}$
1	1	1	0	0
2	0	1	1	0
3	0	0	1	1
4	1	0	0	1

③一二相励磁—为一相励磁与二相励磁交替导通的方式。每送一个励磁信号，步进电机旋转 0.9。

其特点是:分辨率高，运转平滑，故应用也很广泛。如果以该方式控制步进电机正转，对应的励磁顺序见下表。若励磁信号反向传送，则步进电机反转。

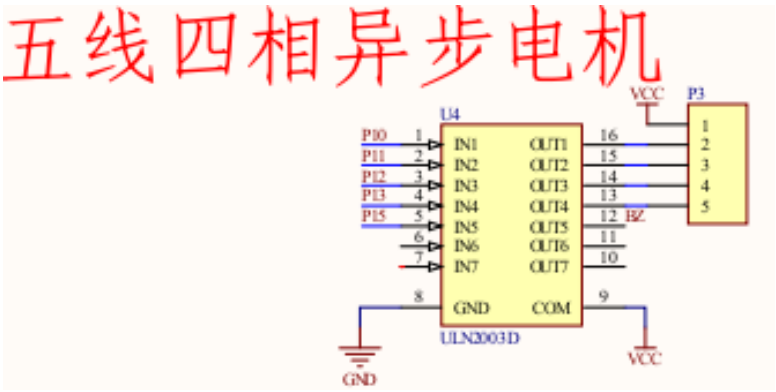
一二相励磁顺序表

STEP	A	B	$\overline{A}$	$\overline{B}$
1	1	0	0	0
2	1	1	0	0
3	0	1	0	0
4	0	1	1	0
5	0	0	1	0
6	0	0	1	1
7	0	0	0	1
8	1	0	0	1

6.步进电机的驱动

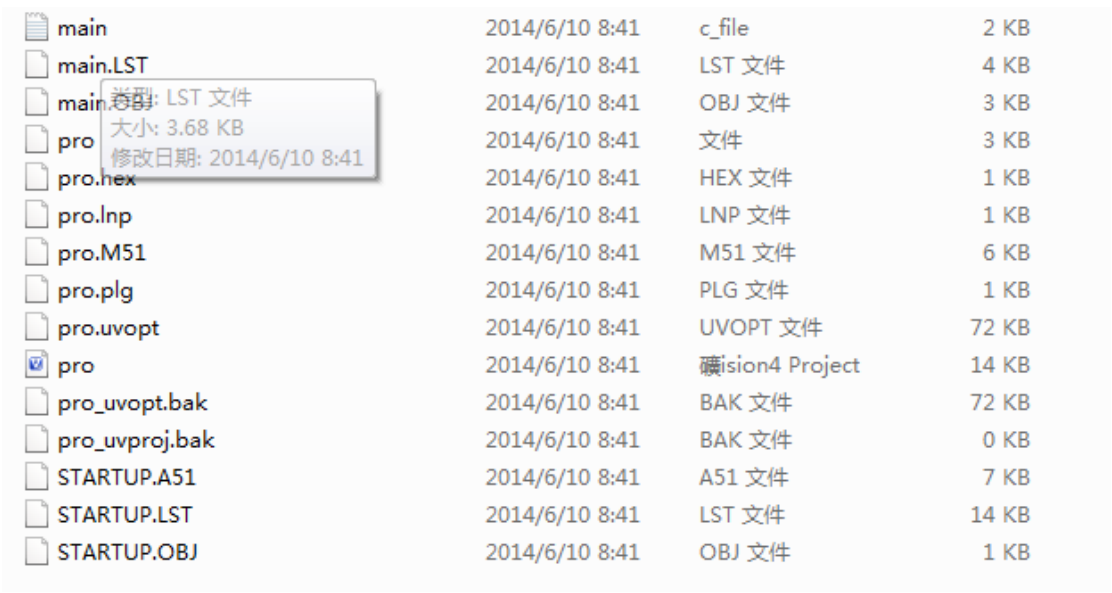
步进电机的驱动可以选用专用的电机驱动模块，如 L298，FF5754 等，这类驱动模块接口简单，操作方便，它们既可驱动步进电机，也可驱动直流电机。除此之外，还可利用三极管自己搭建驭动电路。不过这样会非常麻烦，可靠性也会降低。另外，还有一种方法就是使用达林顿驱动器 ULN2003，该芯片单片最多可一次驱动八线步进电机，当然如果只有四线或六线制的也是没有问题的。

首先我们看一下电路原理图



本实验采用五线制四相步进电机。该电机与 ULN2003 连接如上图所示，P3 为电机五线接口。

打开 6.电机 步进电机转动



下面我们来看程序：

```
/*********************************************************************************
*
* 实 验 名        : 步进电机实验
* 使用的IO        : 电机用P1.0/P1.1/P1.2/P1.3
* 实验效果        :
* 注    意        ：由于P3.2口跟红外线共用，所以做按键实验时为了不让红外线影响实
验
*效果，最好把红外线先取下来。
*********************************************************************************
/
#include "reg52.h"

//电机IO
#define GPIO_MOTOR P1
//sbit F1 = P1^0;
```

```
//sbit F2 = P1^1;
//sbit F3 = P1^2;
//sbit F4 = P1^3;
//按键IO
sbit K1=P3^0;
sbit K2=P3^1;
sbit K3=P3^2;
sbit K4=P3^3;

unsigned char code FFW[8]={0xf1,0xf3,0xf2,0xf6,0xf4,0xfc,0xf8,0xf9}; //反转顺序

void Delay(unsigned int t);
void  Motor();

/*******************************************************************************
*
* 函 数 名         : main
* 函数功能         : 主函数
* 输    入         : 无
* 输    出         : 无
*******************************************************************************
/

void main(void)
{
  while(1)
    {
        Motor();
    }
}
```

```
/*******************************************************************************
*
* 函 数 名         : Motor
* 函数功能         : 电机旋转函数
* 输    入         : 无
* 输    出         : 无
*******************************************************************************
/

void  Motor()
{
    unsigned char i;
    for(i=0;i<8;i++)
    {
        GPIO_MOTOR = FFW[i]&0x1f;  //取数据
        Delay(30);  //调节转速
    }
}
/*******************************************************************************
*
* 函 数 名         : Delay
* 函数功能         : 延时
* 输    入         : t
* 输    出         : 无
*******************************************************************************
/
void Delay(unsigned int t)
{
    unsigned int k;
    while(t--)
```

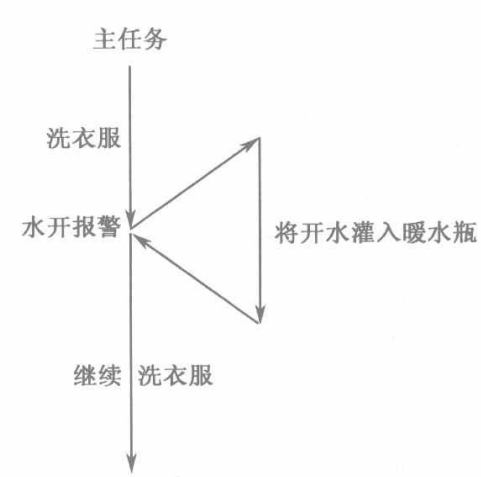
```
    {
        for(k=0; k<80; k++)
        { }
    }
}
```

第十一讲 中断

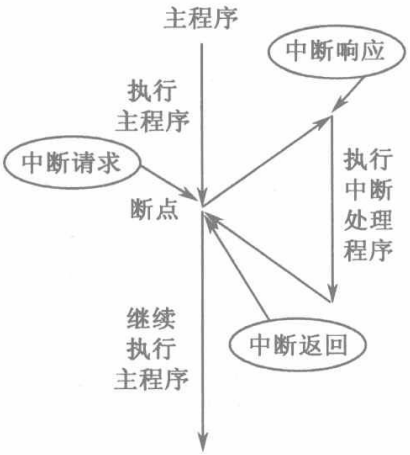
中断是为使单片机具有对外部或内部随机发生的事件实时处理而设置的，中断功能的存在，很大程度上提高了单片机处理外部或内部事件的能力。它也是单片机最重要的功能之一，是我们学习单片机必须要掌握的。很多初学者被困在中断中，学了很久仍然不知道中断究竟是个什么东西，大家千万不要认为它有多难，其实只要掌握正确的学习方法，没有哪个知识点是学不会的。

为了能让大家更容易理解中断概念，我们先来举一个生活事例：你打开火，烧上一壶水。然后去洗衣服，在洗衣服的过程中，突然听到水壶发出水开的报警声，这时，你停止洗衣服动作，立即去关掉火，然后将开水灌入暖水瓶中，灌完开水后，你又回去继续洗衣服。这个过程中实际上就发生了一次中断。

对于单片机来讲，中断是指 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 暂时停止当前的工作(中断响应)，转去处理事件 B(中断服务)；待 CPU 将事件 I.,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。

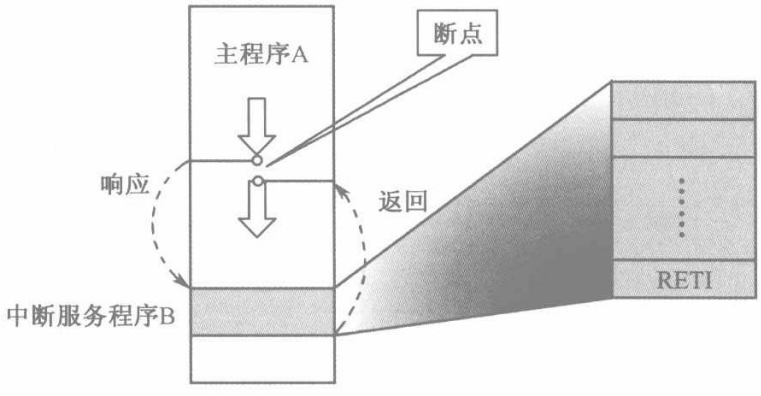


生活中的实例



单片机处理中断的过程

再回来看前面讲的生活事例，与单片机中断结合分析，你的主任务是洗衣服，水开报警这是一个中断请求，这一时刻相当于断点处，你响应中断去关火，然后将开水灌入暖水瓶中，这一动作实际上就是处理中断程序，灌完开水后再回去继续洗衣服，相当于处理完中断程序后再返回主程序继续执行主程序。这里需要注意的是，水开是随时都有可能的，但是无论什么时候开，只要一开你将立即去处理它，处理完后再回来继续接着洗刚才那件衣服。单片机在执行程序时，中断也随时有可能发生，但无论何时发生，只要一旦发生，单片机将立即暂停当前程序，赶去处理中断程序，处理完中断程序后再返回刚才暂停处接着执行原来的程序。单片机在执行程序时其程序流程图。



引起 CPU 中断的根源，称为中断源，中断源向 CPU 提出中断请求，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)。

中断的开启与关闭、设置启用哪一个中断等都是由单片机内部的一些特殊功能寄存器来决定的，在以前的学习中我们仅对单片机内部的特殊功能寄存器 IO 口寄存器设置过，从下节起我们将会设置单片机内部更多的特殊功能寄存器。

当中央处理机 CPU 正在处理某件事的时候外界发生了紧急事件请求，要求 CPU 暂停当前的工作，转而去处理这个紧急事件，处理完以后，再回到原来被中断的地方，继续原来的工作，这样的过程称为中断。实现这种功能的部件称为中断系统，请示 CPU 中断的请求源称为中断源。微型机的中断系统一般允许多个中断源，当几个中断源同时向 CPU 请求中断，要求为它服务的时候，这就存在 CPU 优先响应哪一个中断源请求的问题。通常根据中断源的轻重缓急排队，优先处理最紧急事件的中断请求源，即规定每一个中断源有一个优先级别。CPU 总是先响应优先级别最高的中断请求。

当 CPU 正在处理一个中断源请求的时候(执行相应的中断服务程序)，发生了另外一个优先级比它还高的中断源请求。如果 CPU 能够暂停对原来中断源的服务程序，转而去处理优先级更高的中断请求源，处理完以后，再回到原低级中断服务程序，这样的过程称为中断嵌套。这样的中断系统称为多级中断系统，没有中断嵌套功能的中断系统称为单级中断系统。

STC90C51RC/RD+系列单片机提供了 8 个中断请求源，它们分别是:外部中断 0(INT0),定时器。中断、外部中断 1(INT1)、定时器 1 中断、定时器 2 中断、串口(UART)中断、外部中断 2(INT2)、外部中断 3(INT3)。所有的中断都具有 4 个中断优先级。用户可以用关总中断允许位(EA/IE.7)或相应中断的允许位来屏蔽所有的中断请求，也可以用打开相应的中断允许位来使 CPU 响应相应的中断申请;、每二个中断源可以用软件独立地控制为开中断或关中断状态;每一个中断的优先级别均可用软件设置。高优先级的中断请求可以打断低优先级的中断，反之，低优先级的中断请求不可以打断高优先级及同优先级的中断。当两个相同优先级的中断同时产生时，将由查询次序来决定系统先响应哪个中断。STC90C51RC/RD+系列单片机的各个中断查询次序表如下图所示。

中断源	中断向量地址	相同优先级内的查询次序	中断优先级设置(IPH,IP)	优先级0(最低)	优先级1	优先级2	优先级3(最高)	中断请求标志位	中断允许控制位
$\overline{\text{INT0}}$ (外部中断 0)	0003H	0 (highest)	PX0H, PX0	0, 0	0, 1	1, 0	1, 1	IE0	EX0/EA
Timer 0	000BH	1	PT0H, PT0	0, 0	0, 1	1, 0	1, 1	TF0	ET0/EA
$\overline{\text{INT1}}$ (外部中断1)	0013H	2	PX1H, PX1	0, 0	0, 1	1, 0	1, 1	IE1	EX1/EA
Timer1	001BH	3	PT1H, PT1	0, 0	0, 1	1, 0	1, 1	TF1	ET1/EA
UART	0023H	4	PSH, PS	0, 0	0, 1	1, 0	1, 1	RI+TI	
Timer2	002BH	5	PT2H, PT2	0, 0	0, 1	1, 0	1, 1	TF2 + EXF2	(ET2)/EA
$\overline{\text{INT2}}$ (外部中断2)	0033H	6	PX2H, PX2	0, 0	0, 1	1, 0	1, 1	IE2	EX2/EA
$\overline{\text{INT3}}$ (外部中断3)	003BH	7 (lowest)	PX3H, PX3	0, 0	0, 1	1, 0	1, 1	IE3	EX3/EA

通过设置新增加的特殊功能寄存器 IPH 中的相应位，可将中断优先级设为四级，如果只设置 IP 或 XICON，那么中断优先级就只有两级，与传统 8051 单片机两级中断优先级完全兼容。

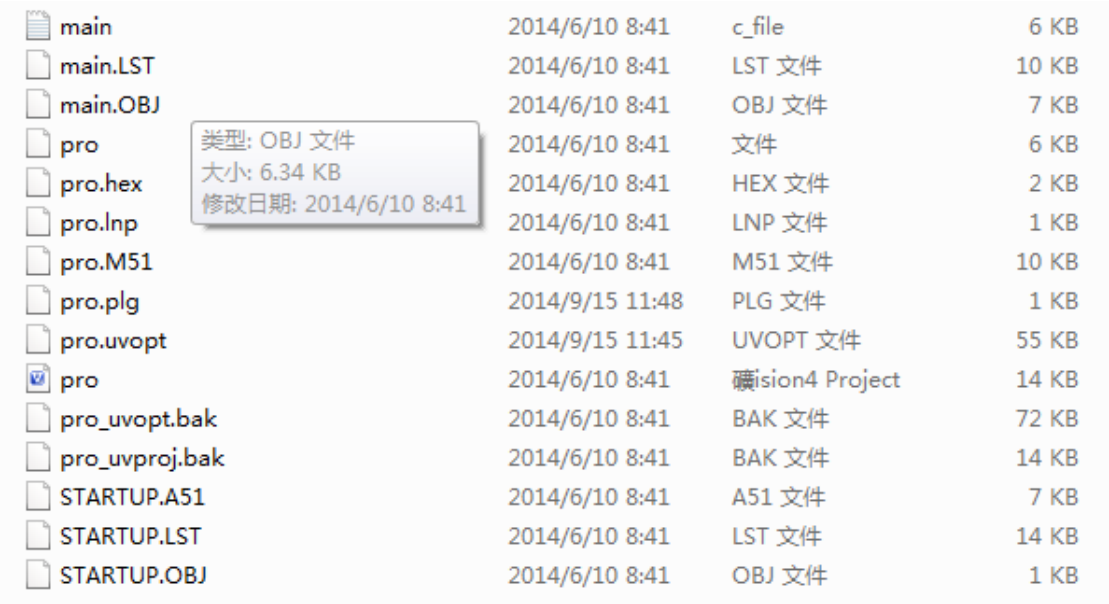
如果使用 C 语言编程，，中断查询次序号就是中断号，例如:

```
void    Into_Routine(void)          interrupt 0;
void    Timer0_Rountine(void)       interrupt 1;
void    Intl_Routine(void)          interrupt 2;
void    Timer1_Rountine(void)       interrupt 3;
void    UART_Routine(void)          interrupt 4;
void    Timer2_Routine(void)        interrupt 5;
void    Int2_Routine(void)          interrupt 6;
void    Int3_Routine(void)          interrupt 7;
```

中断触发表如下图所示

中断源	触发行为
$\overline{\text{INT0}}$ (外部中断0)	(IT0/TCON.0 = 1): 下降沿　(IT0/TCON.0 = 0): 低电平
Timer 0	定时器0溢出
$\overline{\text{INT1}}$ (外部中断1)	(IT1/TCON.2 = 1): 下降沿　(IT1/TCON.2 = 0): 低电平
Timer1	定时器1溢出
UART	发送或接受完成
Timer2	定时器2溢出
$\overline{\text{INT2}}$ (外部中断2)	(IT2/XICON.0 = 1): 下降沿　(IT2/XICON.0 = 0): 低电平
$\overline{\text{INT3}}$ (外部中断3)	(IT3/XICON.4 = 1): 下降沿　(IT3/XICON.4 = 0): 低电平

打开 5.动态数码管 中断显示



下面看一个程序：

```
/*********************************************************************************
*
* 实 验 名        : 动态显示数码管实验
* 使用的IO        : 数码管使用P0,P2.2,P2.3,P2.4键盘使用P1
* 实验效果        : 按矩阵键盘分别显示在数码管上面显示十六进制的0到F。
* 注    意        :
*********************************************************************************
/
#include<reg51.h>
//#include<intrins.h>

#define GPIO_DIG P0
#define GPIO_KEY P1

sbit LSA=P2^2;
sbit LSB=P2^3;
sbit LSC=P2^4;
```

```
unsigned char code DIG_CODE[17]={
0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,
0x7f,0x6f,0x77,0x7c,0x39,0x5e,0x79,0x71};
//0、1、2、3、4、5、6、7、8、9、A、b、C、d、E、F的显示码
unsigned char KeyValue;
//用来存放读取到的键值
unsigned char KeyState; //记录按键的状态，0没有，1有
unsigned char DisplayData[8];
//用来存放要显示的8位数的值
unsigned char Num;//用来存放中断的时候显示的第位数值
void Delay50us();   //延时50us
void KeyDown();      //检测按键函数
void DigDisplay(); //动态显示函数
void TimerConfiguration();//定时器初始化设置
/********************************************************************************
*
* 函 数 名         : main
* 函数功能         : 主函数
* 输    入         : 无
* 输    出         : 无
*******************************************************************************
/
void main(void)
{
    TimerConfiguration();
    KeyState=0;  //初始化按键状态
    while(1)
    {
        KeyDown();
```

```
        if(KeyState==1)
        {
            DisplayData[7]=DisplayData[6];
            DisplayData[6]=DisplayData[5];
            DisplayData[5]=DisplayData[4];
            DisplayData[4]=DisplayData[3];
            DisplayData[3]=DisplayData[2];
            DisplayData[2]=DisplayData[1];
            DisplayData[1]=DisplayData[0];
            DisplayData[0]=DIG_CODE[KeyValue];
            KeyState=0;
        }
//      DigDisplay();
    }
}
/*******************************************************************************
*
* 函 数 名         : TimerConfiguration
* 函数功能         : 定时器初始化
* 输    入         : 无
* 输    出         : 无
*******************************************************************************
/
void TimerConfiguration()
{
    TMOD=0X02;//选择为定时器0模式，工作方式2，仅用TRX打开启动。

    TH0=0X9C;   //给定时器赋初值，定时100us
    TL0=0X9C;
    ET0=1;//打开定时器0中断允许
```

```
    EA=1;//打开总中断
    TR0=1;//打开定时器
}
/*******************************************************************************
*
* 函 数 名         : DigDisplay
* 函数功能         : 使用数码管显示
* 输    入         : 无
* 输    出         : 无
*******************************************************************************
/
void DigDisplay()
{
    unsigned char i,j;
//  for(i=0;i<8;i++)
//  {
        GPIO_DIG=0x00;//消隐
        switch(i)    //位选，选择点亮的数码管，
        {
            case(0):
                LSA=0;LSB=0;LSC=0; break;
            case(1):
                LSA=1;LSB=0;LSC=0; break;
            case(2):
                LSA=0;LSB=1;LSC=0; break;
            case(3):
                LSA=1;LSB=1;LSC=0; break;
            case(4):
                LSA=0;LSB=0;LSC=1; break;
            case(5):
```

```
                LSA=1;LSB=0;LSC=1; break;
            case(6):
                LSA=0;LSB=1;LSC=1; break;
            case(7):
                LSA=1;LSB=1;LSC=1; break;
        }
        GPIO_DIG=DisplayData[i];
        i++;
        if(i>7)
            i=0;
//      j=10;                           //扫描间隔时间设定
//      while(j--)
//          Delay50us();
//      GPIO_DIG=0x00;//消隐
//  }
}
/*********************************************************************************
*
* 函 数 名         : KeyDown
* 函数功能         : 检测有按键按下并读取键值
* 输    入         : 无
* 输    出         : 无
*********************************************************************************
/
void KeyDown(void)
{
    unsigned int a=0;
    GPIO_KEY=0x0f;
    if(GPIO_KEY!=0x0f)
  {
```

```
        Delay50us();
        a++;
        a=0;
        if(GPIO_KEY!=0x0f)
        {
            ET0=0;//关定时器中断
            KeyState=1;//有按键按下
            //测试列
            GPIO_KEY=0X0F;
//          Delay50us();
            switch(GPIO_KEY)
            {
                case(0X07): KeyValue=0;break;
                case(0X0b): KeyValue=1;break;
                case(0X0d): KeyValue=2;break;
                case(0X0e): KeyValue=3;break;
//              default:    KeyValue=17;    //检测出错回复17意思是把数码管全灭
掉。
            }
            //测试行
            GPIO_KEY=0XF0;
            Delay50us();
            switch(GPIO_KEY)
            {
                case(0X70): KeyValue=KeyValue;break;
                case(0Xb0): KeyValue=KeyValue+4;break;
                case(0Xd0): KeyValue=KeyValue+8;break;
                case(0Xe0): KeyValue=KeyValue+12;break;
//              default:    KeyValue=17;
            }
```

```
            ET0=1;//开定时器中断
            while((a<5000)&&(GPIO_KEY!=0xf0))     //检测按键松手检测
            {
                Delay50us();
                a++;
            }
            a=0;
        }
    }
}
/*********************************************************************************
*
* 函 数 名         : Delay50us
* 函数功能         : 延时函数，延时50us
* 输    入         : 无
* 输    出         : 无
*********************************************************************************
/
void Delay50us(void)   //延时50us误差 0us
{
    unsigned char a,b;
    for(b=1;b>0;b--)
        for(a=22;a>0;a--);
}
/*********************************************************************************
*
* 函 数 名         : Delay50us
* 函数功能         : 延时函数，延时50us
* 输    入         : 无
* 输    出         : 无
```

```
*******************************************************************************
/
void Timer() interrupt 1
{
    DigDisplay();
}
```

第十二讲 1602 液晶显示

液晶(Liquid Crystal)是一种高分子材料，因为其特殊的物理、化学、光学特性，20 世纪中叶开始广泛应用在轻薄型显示器上。

液晶显示器(Liquid Crystal Display, LCD)的主要原理是以电流刺激液晶分子产生点、线、面并配合背部灯管构成画面。为叙述简便，通常把各种液晶显示器都直接叫做液晶。

各种型号的液晶通常是按照显示字符的行数或液晶点阵的行、列数来命名的。比如:1602 的意思是每行显示 16 个字符，一共可以显示两行;类似的命名还有 0801, 0802, 1601 等，这类液晶通常都是字符型液晶，即只能显示 ASCII 码字符，如数字、大小写字母、各种符号等。12232 液晶属于图形型液晶，她的意思是液晶由 122 列、32 行组成，即共有 122 X 32 个点来显示各种图形，我们可以通过程序控制这 122 X 32 个点中的任一个点显示或不显示。类似的命名还有 12864, 19264, 192128, 320240 等，根据客户需要，厂家可以设计出任意数组合的点阵液晶。

液晶显示的使用有多广泛我就不多说了，LCD1602 好像10 元左右就可以拿到了的，不算贵。我们来看看现在市面都有哪些样子的1602，下面从网上搜罗了几个：

蓝白屏

黄绿屏

其实显而易，见也就是背光和字体的颜色不一样罢，不过老实说，蓝色背光的1602 看上去显得确实比较亮，也许是人眼视觉的关系。

接下来进入LCD1602 使用的重点：操作时序。操作时序永远使用是任何一片IC 芯片的最主要的内容。一个芯片的所有使用细节都会在它的官方器件手册上包含。所以使用一个器件事情，要充分做好的第一件事就是要把它的器件手册上有用的内容提取，掌握。介于中国目前的芯片设计能力有限，所以大部分的器件都是外国几个IC 巨头比如TI、AT、MAXIM 这些公司生产的，器件资料自然也是英文的多，所以，英文的基础要在阅读这些数据手册时得到提高哦。即便有中文翻译版本，还是建议看英文原版，看不懂时不妨再参考中文版，这样比较利于提高。

我们首先来看1602 的引脚定义,1602 的引脚是很整齐的SIP 单列直插封装，所以器件手册只给出了引脚的功能数据表：

◆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

我们只需要关注以下几个管脚：

3 脚：VL，液晶显示偏压信号，用于调整LCD1602 的显示对比度，一般会外接电位器用以调整偏压信号，注意此脚电压为0 时可以得到最强的对比度。

4 脚：RS，数据/命令选择端，当此脚为高电平时，可以对1602 进行数据字

节的传输操作，而为电平时，则是进行命令字节的传输操作。命令字节，即是用来对LCD1602 的一些工作方式作设置的字节；数据字节，即使用以在1602 上显示的字节。值得一提的是，LCD1602的数据是8 位的。

脚：R/W，读写选择端。当此脚为高电平可对LCD1602 进行读数据操作，反之进行写数据操作。笔者认为，此脚其实用处不大，直接接地永久置为低电平也不会影响其正常工作。但是尚未经过复杂系统验证，保留此意见。

6 脚：E，使能信号，其实是LCD1602 的数据控制时钟信号，利用该信号的上升沿实现对LCD1602 的数据传输。

7~14 脚：8 位并行数据口，使得对LCD1602 的数据读写大为方便。

主要技术参数

1602 液晶主要技术参数表

显示容量	16×2 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	2.95×4.35（W×H）mm

现在来看LCD1602 的操作时序：

1 基本操作时序：

1.1 读状态：输入：RS=L, RW=H, E=H　　输出：D0～D7=状态字

1.2 写指令：输入：RS=L, RW=L, D0～D7=指令码, E=高脉冲　　输出：无

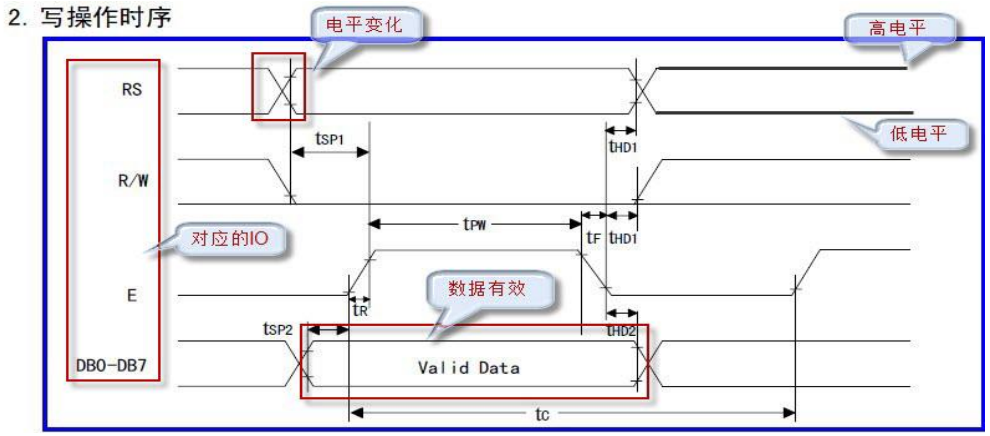
1.3 读数据：输入：RS=H, RW=H, E=H　　输出：D0～D7=数据

1.4 写数据：输入：RS=H, RW=L, D0～D7=数据, E=高脉冲　　输出：无

在此，我们不需要读出它的数据的状态或者数据本身。所以只需要看两个写时序：

① 当我们要写指令字，设置LCD1602 的工作方式时：需要把RS置为低电平，RW置为低电平，然后将数据送到数据口D0~D7，最后E引脚一个高脉冲将数据写入。

② 当我们要写入数据字，在1602 上实现显示时：需要把RS置为高电平，RW置为低电平，然后将数据送到数据口D0~D7，最后E 引脚一个高脉冲将数据写入。发现了么，写指令和写数据，差别仅仅在于RS的电平不一样而已。一下是LCD1602的时序图：



大家要慢慢学会看时序图，要知道操作一个器件的精华便蕴藏在其中，看懂看准了时序，你操控这个芯片就是非常容易的事了。1602 的时序是我见过的一个最简单的时序：

1. 注意时间轴，如果没有标明（其实大部分也都是不标明的），那么从左往右的方向为时间正向轴，即时间在增长。
2. 上图框出并注明了看懂此图的一些常识：
   1) 时序图最左边一般是某一根引脚的标识，表示此行图线体现该引脚的变化，上图分别标明了RS、R/W、E、DB0~DB7 四类引脚的时序变化。
   2) 有线交叉状的部分，表示电平在变化，如上所标注。
   3) 应该比较容易理解，如上图右上角所示，两条平行线分别对应高低电平，也正好吻合(2)中电平变化的说法。
   4) 上图下，密封的菱形部分，注意要密封，表示数据有效，Valid Data 这个词也显示了这点。
3. 需要十分严重注意的是，时序图里各个引脚的电平变化，基于的时间轴是一致的。一定要严格按照时间轴的增长方向来精确地观察时序图。要让器件严格的遵守时序图的变化。在类似于18B20 这样的单总线器件对此要求尤为严格。
4. 以上几点，并不是LCD1602 的时序图所特有的，绝大部分的时序图都遵循着这样的一般规则，所以大家要慢慢的习惯于这样的规则。

也许你还注意到了上面有许多关于时间的标注，这也是个十分重要的信息，这些时间的标注表明了某些状态所要维持的最短或最长时间。因为器件的工作速度也是有限的，一般都跟不上主控芯片的速度，所以它们直接之间要有时序配合。

话说现在各种处理器的主频也是疯狂增长，日后搞不好出现个双核单片机也不一定就是梦话。下面是时序参数表：

3. 时序参数

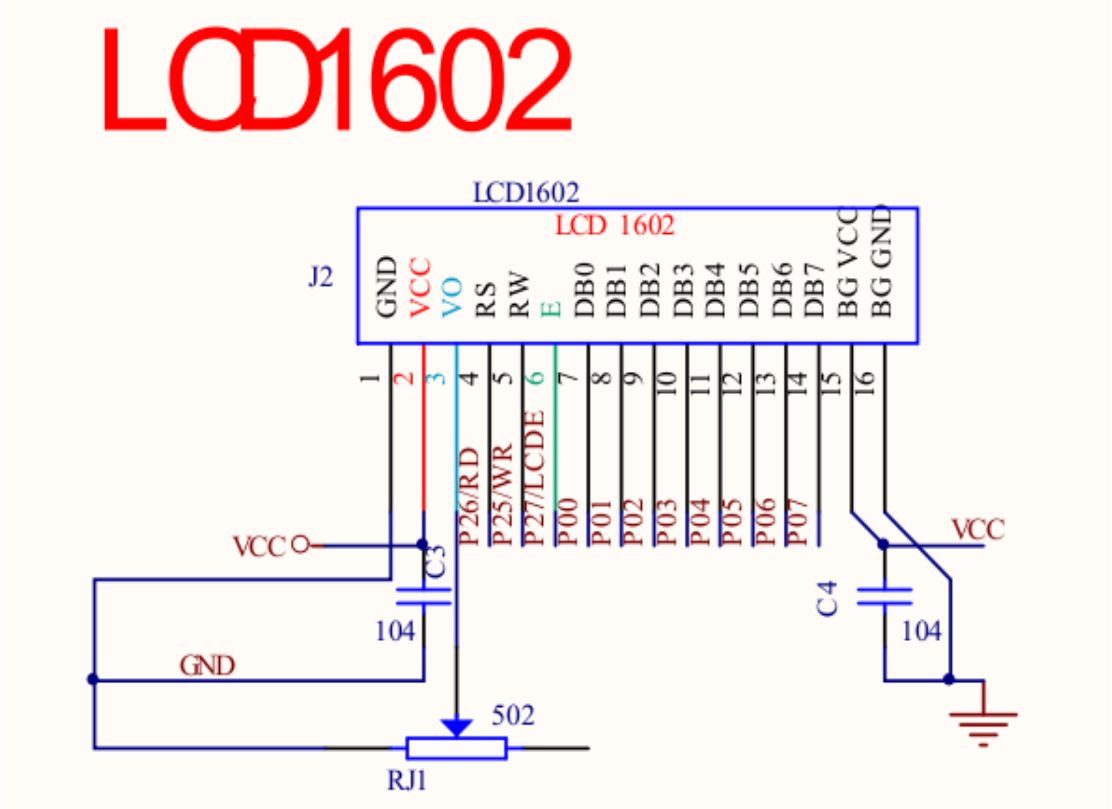
时序参数	符号	极限值			单位	测试条件
		最小值	典型值	最大值		
E 信号周期	t_C	400	–	–	ns	引脚 E
E 脉冲宽度	t_{PW}	150	–	–	ns	
E 上升沿/下降沿时间	t_R, t_F	–	–	25	ns	
地址建立时间	t_{SP1}	30	–	–	ns	引脚 E、RS、R/W
地址保持时间	t_{HD1}	10	–	–	ns	
数据建立时间(读操作)	t_D	–	–	100	ns	引脚 DB0～DB7
数据保持时间(读操作)	t_{HD2}	20	–	–	ns	
数据建立时间(写操作)	t_{SP2}	40	–	–	ns	
数据保持时间(写操作)	t_{HD2}	10	–	–	ns	

大家要懂得估计主控芯片的指令时间，可以在官方数据手册上查到MCU的一些级别参数。比如我们现在主控芯片，外部12MHz 晶振，指令周期就是一个时钟周期为（1/12MHz）us，所以至少确定了它执行一条指令的时间是us级别的。我们看到，以上给的时间参数全部是ns级别的，所以即便我们在程序里不加延时程序，也应该可以很好的配合LCD1602的时序要求了。怎么看这个表呢？很简单，我们在时序图里可以找到TR1，对应时序参数表，可以查到这个是E上升沿/下降沿时间，最大值为25ns，表示E引脚上的电平变化，必须在最大为25ns之内的时间完成。大家看是不是这个意思？

现在我来解读我对这个时序图的理解：

当要写命令字节的时候，时间由左往右，RS 变为低电平，R/W 变为低电平，注意看是RS 的状态先变化完成。然后这时，DB0~DB7 上数据进入有效阶段，接着E引脚有一个整脉冲的跳变，接着要维持时间最小值为tpw=400ns的E脉冲宽度。然后E引脚负跳变，RS电平变化，R/W 电平变化。这样便是一个完整的LCD1602写命令的时序。

我们看一下开发板的电路图



打开 7.LCD1602 LCD160显示字母

lcd._i	2014/6/10 8:41	_I 文件	1 KB
lcd	2014/6/10 8:41	c_file	4 KB
lcd.h	2014/6/10 8:41	H 文件	1 KB
lcd.LST	2014/6/10 8:41	LST 文件	7 KB
lcd.OBJ	2014/6/10 8:41	OBJ 文件	4 KB
main	2014/6/10 8:41	c_file	1 KB
main.LST	2014/6/10 8:41	LST 文件	2 KB
main.OBJ	2014/6/10 8:41	OBJ 文件	2 KB
pro	2014/6/10 8:41	文件	6 KB
pro.hex	2014/6/10 8:41	HEX 文件	1 KB
pro.lnp	2014/6/10 8:41	LNP 文件	1 KB
pro.M51	2014/6/10 8:41	M51 文件	8 KB
pro.plg	2014/6/10 8:41	PLG 文件	1 KB
pro.uvopt	2014/6/10 8:41	UVOPT 文件	141 KB
pro	2014/6/10 8:41	礦ision4 Project	14 KB
pro_uvopt.bak	2014/6/10 8:41	BAK 文件	141 KB
pro_uvproj.bak	2014/6/10 8:41	BAK 文件	14 KB
STARTUP.A51	2014/6/10 8:41	A51 文件	7 KB
STARTUP.LST	2014/6/10 8:41	LST 文件	14 KB
STARTUP.OBJ	2014/6/10 8:41	OBJ 文件	1 KB

下面看一下程序：

/***

```
*****
    * 实 验 名        : 矩阵键盘实验
    * 使用的IO        : 数码管使用P0,键盘使用P3.0、P3.1、P3.2、P3.3
    * 实验效果        : 按矩阵键盘分别显示在数码管上面显示十六进制的0到F。
    * 注    意        :
    ********************************************************************************
****/
    #include<reg51.h>
    #include"lcd.h"

    unsigned char PuZh[]=" Pechin Science ";

    /********************************************************************************
*****
    * 函 数 名         : main
    * 函数功能         : 主函数
    * 输    入         : 无
    * 输    出         : 无
    ********************************************************************************
****/
    void main(void)
    {
        unsigned char i;
        LcdInit();
        for(i=0;i<16;i++)
        {
            LcdWriteData(PuZh[i]);
        }
        while(1)
        {
        }
    }
    #include"lcd.h"

    /********************************************************************************
*****
    * 函 数 名         : Lcd1602_Delay1ms
    * 函数功能         : 延时函数，延时1ms
    * 输    入         : c
    * 输    出         : 无
    * 说    名         : 该函数是在12MHZ晶振下，12分频单片机的延时。
    ********************************************************************************
****/
```

```
    void Lcd1602_Delay1ms(uint c)   //误差 0us
    {
        uchar a,b;
        for (; c>0; c--)
        {
             for (b=199;b>0;b--)
             {
                for(a=1;a>0;a--);
             }
        }

    }

    /*****************************************************************************
*****
    * 函 数 名         : LcdWriteCom
    * 函数功能         : 向LCD写入一个字节的命令
    * 输    入         : com
    * 输    出         : 无
    *****************************************************************************
****/
    #ifndef     LCD1602_4PINS    //当没有定义这个LCD1602_4PINS时
    void LcdWriteCom(uchar com)   //写入命令
    {
        LCD1602_E = 0;     //使能
        LCD1602_RS = 0;    //选择发送命令
        LCD1602_RW = 0;    //选择写入

        LCD1602_DATAPINS = com;     //放入命令
        Lcd1602_Delay1ms(1);        //等待数据稳定

        LCD1602_E = 1;            //写入时序
        Lcd1602_Delay1ms(5);      //保持时间
        LCD1602_E = 0;
    }
    #else
    void LcdWriteCom(uchar com)   //写入命令
    {
        LCD1602_E = 0;   //使能清零
        LCD1602_RS = 0;  //选择写入命令
        LCD1602_RW = 0;  //选择写入

        LCD1602_DATAPINS = com; //由于4位的接线是接到P0口的高四位，所以传送高四
位不用改
```

```
        Lcd1602_Delay1ms(1);

        LCD1602_E = 1;   //写入时序
        Lcd1602_Delay1ms(5);
        LCD1602_E = 0;

    //  Lcd1602_Delay1ms(1);
        LCD1602_DATAPINS = com << 4; //发送低四位
        Lcd1602_Delay1ms(1);

        LCD1602_E = 1;   //写入时序
        Lcd1602_Delay1ms(5);
        LCD1602_E = 0;
    }
    #endif
    /*********************************************************************************
*****
    * 函 数 名         : LcdWriteData
    * 函数功能         : 向LCD写入一个字节的数据
    * 输    入         : dat
    * 输    出         : 无
    *********************************************************************************
****/
    #ifndef     LCD1602_4PINS
    void LcdWriteData(uchar dat)            //写入数据
    {
        LCD1602_E = 0;  //使能清零
        LCD1602_RS = 1; //选择输入数据
        LCD1602_RW = 0; //选择写入

        LCD1602_DATAPINS = dat; //写入数据
        Lcd1602_Delay1ms(1);

        LCD1602_E = 1;   //写入时序
        Lcd1602_Delay1ms(5);   //保持时间
        LCD1602_E = 0;
    }
    #else
    void LcdWriteData(uchar dat)            //写入数据
    {
        LCD1602_E = 0;    //使能清零
        LCD1602_RS = 1;   //选择写入数据
        LCD1602_RW = 0;   //选择写入
```

```
        LCD1602_DATAPINS = dat; //由于4位的接线是接到P0口的高四位，所以传送高四位不用改
        Lcd1602_Delay1ms(1);

        LCD1602_E = 1;    //写入时序
        Lcd1602_Delay1ms(5);
        LCD1602_E = 0;

        LCD1602_DATAPINS = dat << 4; //写入低四位
        Lcd1602_Delay1ms(1);

        LCD1602_E = 1;    //写入时序
        Lcd1602_Delay1ms(5);
        LCD1602_E = 0;
    }
    #endif
    /*******************************************************************************
    * 函 数 名       : LcdInit()
    * 函数功能       : 初始化LCD屏
    * 输    入       : 无
    * 输    出       : 无
    *******************************************************************************/
    #ifndef     LCD1602_4PINS
    void LcdInit()                          //LCD初始化子程序
    {
        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
    }
    #else
    void LcdInit()                          //LCD初始化子程序
    {
        LcdWriteCom(0x32);   //将8位总线转为4位总线
        LcdWriteCom(0x28);   //在四位线下的初始化
        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
    }
    #endif
```

```
#ifndef __LCD_H_
#define __LCD_H_
/**********************************
当使用的是4位数据传输的时候定义，
使用8位取消这个定义
**********************************/
#define LCD1602_4PINS

/**********************************
包含头文件
**********************************/
#include<reg51.h>

//---重定义关键词---//
#ifndef uchar
#define uchar unsigned char
#endif

#ifndef uint
#define uint unsigned int
#endif

/**********************************
PIN口定义
**********************************/
#define LCD1602_DATAPINS P0
sbit LCD1602_E=P2^7;
sbit LCD1602_RW=P2^5;
sbit LCD1602_RS=P2^6;

/**********************************
函数声明
**********************************/
/*在51单片机12MHZ时钟下的延时函数*/
void Lcd1602_Delay1ms(uint c);   //误差 0us
/*LCD1602写入8位命令子函数*/
void LcdWriteCom(uchar com);
/*LCD1602写入8位数据子函数*/
void LcdWriteData(uchar dat)    ;
/*LCD1602初始化子程序*/
void LcdInit();

#endif
```

下载lcd.hex,观察实验结果，实验结果是LCD1602第一行显示Pechin Scinence,第二行显示按键值。

作几点说明：

1. LCD1602 对写进去的数据字节呢是以ASCII 码识别的，所以写进去用以显示的字符数据必须是某一个ADCII 码，当然如果你不想查ASCII 表的话，可以用字符来代替，即用单引号包含的字符常量。所以，也由此推出，’0’和30H（0 的ASCII 码）是等价的。
2. 操作1602，要先对1602 进行初始化，数据手册里写的比较清楚，并且对各个命令字的写入并没有先后要求。
3. 1602 是有自定义字符的功能的，大家熟悉基本操作之后可以尝试自行拓展
4. 在每次写完数据之后，应该要将E 引脚置为低电平，为下一次E 的高脉冲做准备。延伸来说，这叫释放时钟线，要养成释放时钟线的好习惯。对配合时序大有裨益。
5. 可以将所要显示的字符一次定义在一个字符数组里，以调用字符数组的形式调用显示数据，这样程序会变得简洁而高效。
6. 记住时序的要求，往往是对最小时间有要求，在你严格配合时序的情况下仍然的不到理想的结果时，可以尝试插入延时。这个并不违反时序的要求。况且相当多的器件手册并没有详细的讲述最小时间要求。

第十三讲 定时器

1. 定时器的原理

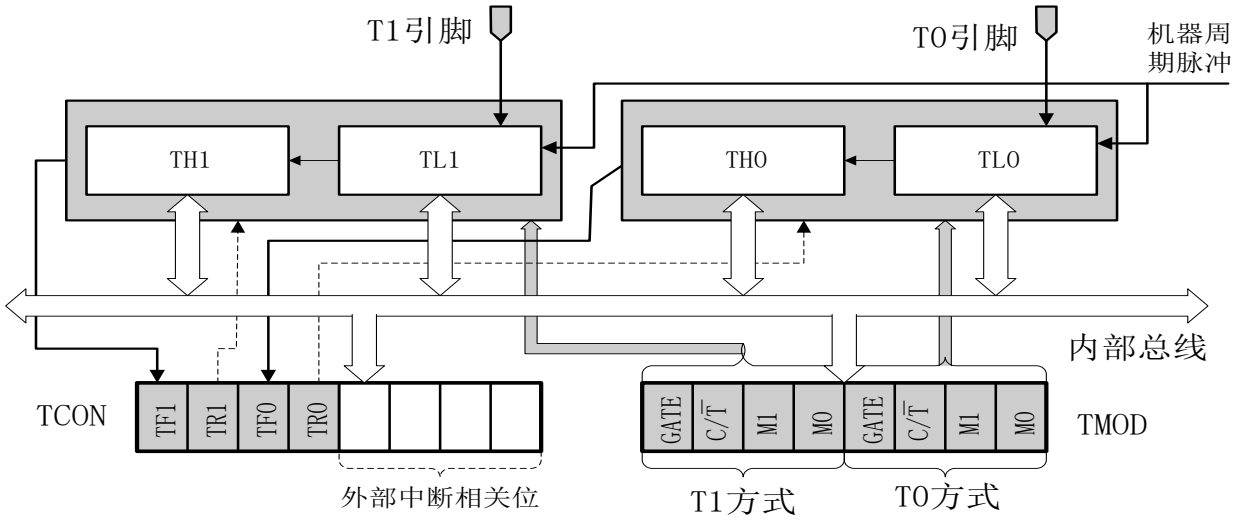
加 1 计数器输入的计数脉冲有两个来源,一个是由系统的时钟振荡器输出脉冲经 12 分频后送来；一个是 T0 或 T1 引脚输入的外部脉冲源。每来一个脉冲计数器加 1，当加到计数器为全 1 时，再输入一个脉冲就使计数器回零，且计数器的溢出使 TCON 中 TF0 或 TF1 置 1，向 CPU 发出中断请求（定时

/计数器中断允许时）。如果定时/计数器工作于定时模式，则表示定时时间已到；如果工作于计数模式，则表示计数值已满。

可见，由溢出时计数器的值减去计数初值才是加 1 计数器的计数值。

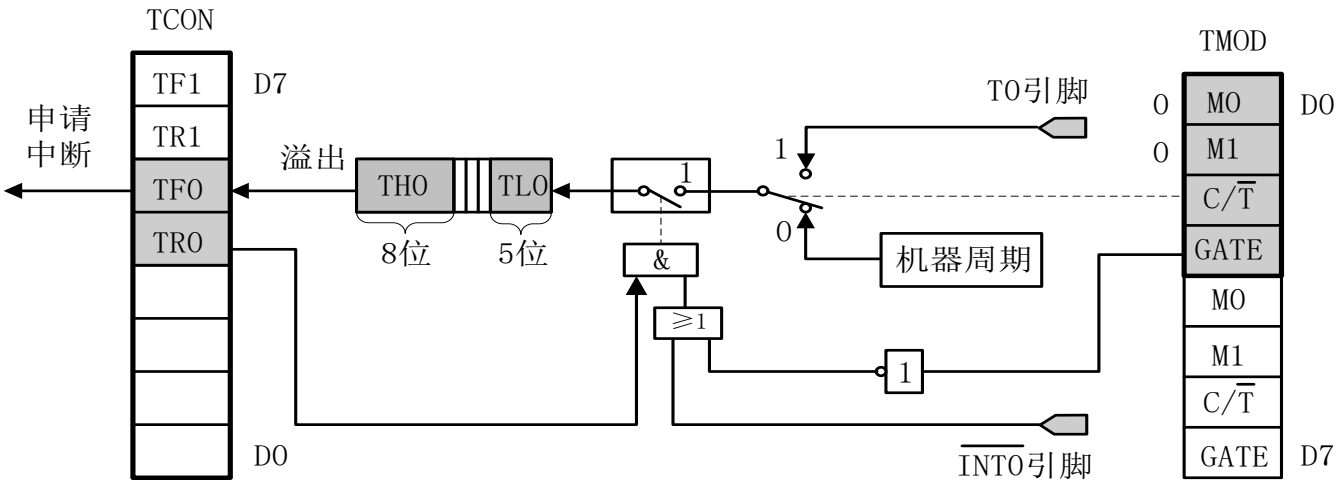
2. 51 定时器

80c51 单片机内有两个可编程的定时/计数器 T0、T1。定时/计数器的实质是加 1 计数器（16 位），由高 8 位和低 8 位两个寄存器组成。TMOD 是定时/计数器的工作方式寄存器，确定工作方式和功能；TCON 是控制寄存器，控制 T0、T1 的启动和停止及设置溢出标志。



3. 工作方式 0

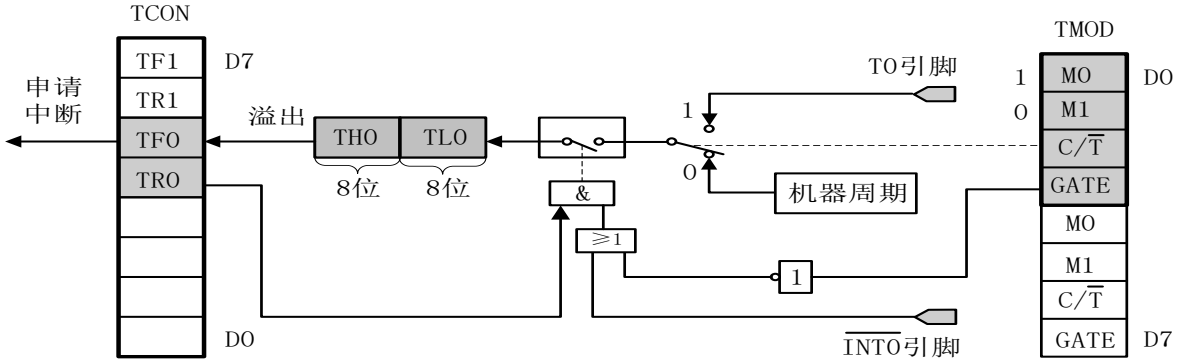
方式 0 为 13 位计数，由 TL0 的低 5 位（高 3 位未用）和 TH0 的 8 位组成。TL0 的低 5 位溢出时向 TH0 进位，TH0 溢出时，置位 TCON 中的 TF0 标志，向



CPU 发出中断请求。

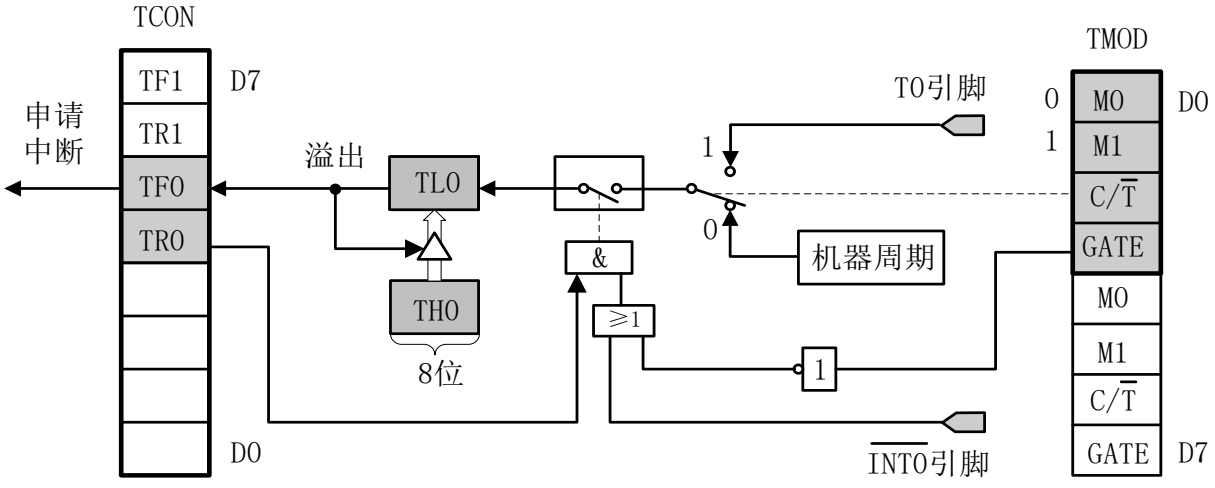
4. 工作方式 1

方式 1 的计数位数是 16 位，由 TL0 作为低 8 位、TH0 作为高 8 位，组成了 16 位加 1 计数器 。



工作方式 2

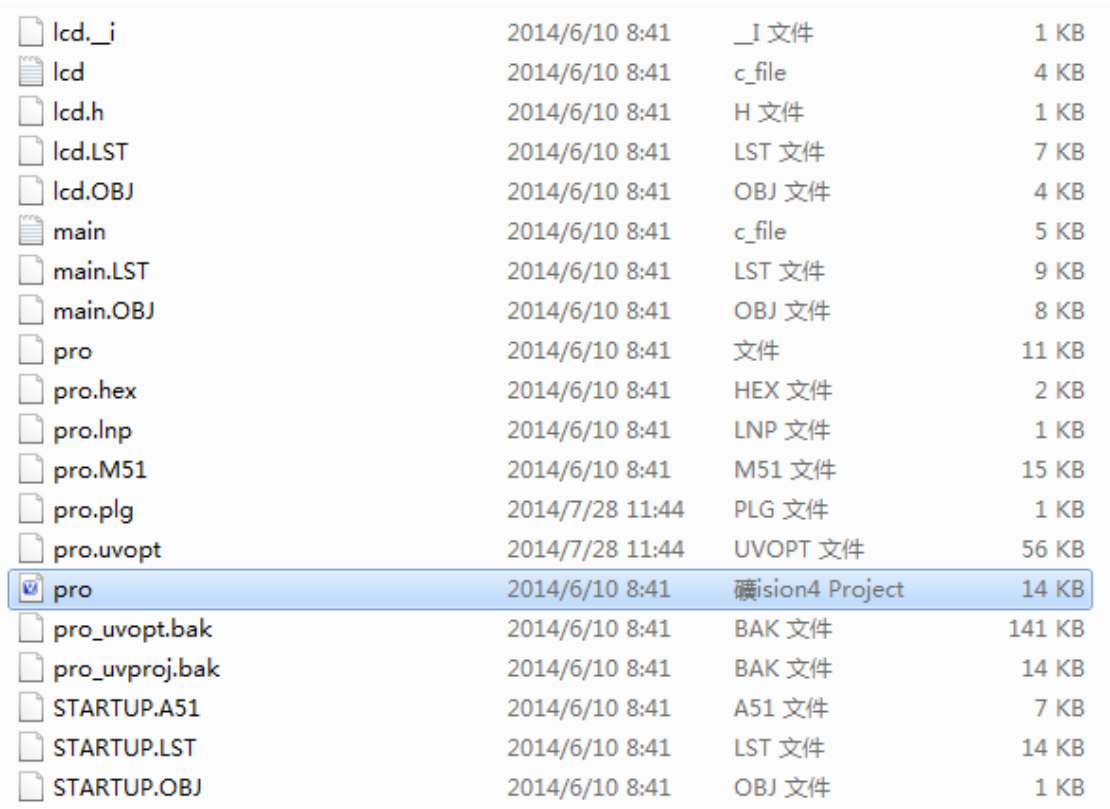
方式 2 为自动重装初值的 8 位计数方式。工作方式 2 特别适合于用作较精确的脉冲信号发生器。



工作方式 3

方式 3 只适用于定时/计数器 T0，定时器 T1 处于方式 3 时相当于 TR1=0，停止计数。工作方式 3 将 T0 分成为两个独立的 8 位计数器 TL0 和 TH0 。

打开 8.时钟（定时器） 定时器时钟



lcd._i	2014/6/10 8:41	_I 文件	1 KB
lcd	2014/6/10 8:41	c_file	4 KB
lcd.h	2014/6/10 8:41	H 文件	1 KB
lcd.LST	2014/6/10 8:41	LST 文件	7 KB
lcd.OBJ	2014/6/10 8:41	OBJ 文件	4 KB
main	2014/6/10 8:41	c_file	5 KB
main.LST	2014/6/10 8:41	LST 文件	9 KB
main.OBJ	2014/6/10 8:41	OBJ 文件	8 KB
pro	2014/6/10 8:41	文件	11 KB
pro.hex	2014/6/10 8:41	HEX 文件	2 KB
pro.lnp	2014/6/10 8:41	LNP 文件	1 KB
pro.M51	2014/6/10 8:41	M51 文件	15 KB
pro.plg	2014/7/28 11:44	PLG 文件	1 KB
pro.uvopt	2014/7/28 11:44	UVOPT 文件	56 KB
pro	2014/6/10 8:41	礦ision4 Project	14 KB
pro_uvopt.bak	2014/6/10 8:41	BAK 文件	141 KB
pro_uvproj.bak	2014/6/10 8:41	BAK 文件	14 KB
STARTUP.A51	2014/6/10 8:41	A51 文件	7 KB
STARTUP.LST	2014/6/10 8:41	LST 文件	14 KB
STARTUP.OBJ	2014/6/10 8:41	OBJ 文件	1 KB

接下来我们看看程序：

```
/*************************************************************************
**************
* 实验名                    : 定时器实验
* 使用的 IO          :
* 实验效果          :1602 显示时钟，按 K3 进入时钟设置，按 K1 选择设置的
时分秒，按 K2 选择
*选择设置加 1。
*    注意                          :
*************************************************************************
*************/
#include<reg51.h>
#include"lcd.h"
sbit K1=P3^0;
sbit K2=P3^1;
sbit K3=P3^2;
sbit K4=P3^3;
```

```
unsigned char Time;
//用来计时间的值

void Delay1ms(unsigned int c);
void TimerConfiguration();
void Int0Configuration();

unsigned char SetPlace;

/*****************************************************************
**************
* 函 数 名            : main
* 函数功能          : 主函数
* 输    入            : 无
* 输    出            : 无
*****************************************************************
*************/
void main(void)
{
    unsigned char hour,minit,second;
    unsigned int i;
    TimerConfiguration();
    Int0Configuration();
    LcdInit();
    hour=12;
    LcdWriteData('0'+hour/10);
    LcdWriteData('0'+hour%10);
    LcdWriteData('-');
    LcdWriteData('0'+minit/10);
    LcdWriteData('0'+minit%10);
    LcdWriteData('-');
    LcdWriteData('0'+second/10);
    LcdWriteData('0'+second%10);
    while(1)
    {
        if(TR0==0)
        {
            if(K1==0)          //检测按键 K2 是否按下
            {
                Delay1ms(10);    //消除抖动
                if(K1==0)
                {
                    SetPlace++;
                    if(SetPlace>=3)
```

```
                SetPlace=0;
            }

            while((i<50)&&(K1==0))     //检测按键是否松开
            {
                Delay1ms(1);
                i++;
            }
            i=0;
        }
        if(K2==0)           //检测按键 K3 是否按下
        {
            Delay1ms(10);   //消除抖动
            if(K2==0)
            {
                if(SetPlace==0)
                {
                    second++;
                    if(second>=60)
                        second=0;
                }
                else if(SetPlace==1)
                {
                    minit++;
                    if(minit>=60)
                        minit=0;
                }
                else
                {
                    hour++;
                    if(hour>=24)
                        hour=0;
                }
            }

            while((i<50)&&(K2==0))     //检测按键是否松开
            {
                Delay1ms(1);
                i++;
            }
            i=0;
        }
    }
    if(Time>=20)  //一秒钟来到改变数值
```

```
        {
            Time=0;
            second++;
            if(second==60)
            {
                second=0;
                minit++;
                if(minit==60)
                {
                    minit=0;
                    hour++;
                    if(hour==24)
                    {
                        hour=0;
                    }
                }
            }
        }
         //--显示时钟--//
        LcdWriteCom(0x80);
        LcdWriteData('0'+hour/10);
        LcdWriteData('0'+hour%10);
        LcdWriteCom(0x83);
        LcdWriteData('0'+minit/10);
        LcdWriteData('0'+minit%10);
        LcdWriteCom(0x86);
        LcdWriteData('0'+second/10);
        LcdWriteData('0'+second%10);

    }
}
/*****************************************************************
**************
* 函 数 名          : Delay1ms()
* 函数功能        : 延时 1ms
* 输    入          : c
* 输    出          : 无
*****************************************************************
*************/

void Delay1ms(unsigned int c)     //误差 0us
{
    unsigned char a,b;
    for (; c>0; c--)
```

```
    {
        for(b=199;b>0;b--)
        {
            for(a=1;a>0;a--);
        }

    }

}

/*************************************************************************
**************
* 函 数 名            : TimerConfiguration()
* 函数功能            : 配置定时器值
* 输    入            : 无
* 输    出            : 无
*************************************************************************
*************/

void TimerConfiguration()
{
    TMOD = 0x01; //选择工作方式 1
    TH0 = 0x3C;  //设置初始值
    TL0 = 0x0B0;
    EA = 1;             //打开总中断
    ET0 = 1;       //打开定时器 0 中断
    TR0 = 1;            //启动定时器 0
}
/*************************************************************************
**************
* 函 数 名            : Timer0()
* 函数功能            : 定时器 0 中断函数
* 输    入            : 无
* 输    出            : 无
*************************************************************************
*************/

void Timer0() interrupt 1
{
      TH0 = 0x3C;     //设置初始值
    TL0 = 0x0B0;
        Time++;
}
/*************************************************************
```

```
**************
* 函 数 名          : Int0Configuration()
* 函数功能          : 配置外部中断 0
* 输    入          : 无
* 输    出          : 无
*******************************************************************
*************/

void Int0Configuration()
{
    //设置 INT0
    IT0=1;//跳变沿出发方式（下降沿）
    EX0=1;//打开 INT0 的中断允许。
    EA=1;//打开总中断
}
/*******************************************************************
**************
* 函 数 名          : Int0() interrupt 0
* 函数功能          : 外部中断 0 的中断函数
* 输    入          : 无
* 输    出          : 无
*******************************************************************
*************/

void Int0() interrupt 0
{
    Delay1ms(10);
    if(K3==0)
    {
        TR0=~TR0;
        SetPlace=0;
    }
}

#include"lcd.h"

/*******************************************************************
**************
* 函 数 名          : Lcd1602_Delay1ms
* 函数功能          : 延时函数，延时 1ms
* 输    入          : c
* 输    出          : 无
* 说    名          : 该函数是在 12MHZ 晶振下，12 分频单片机的延时。
*******************************************************************
```

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

void Lcd1602_Delay1ms(uint c)     //误差  0us
{
      uchar a,b;
     for (; c>0; c--)
     {
           for (b=199;b>0;b--)
           {
               for(a=1;a>0;a--);
           }
     }

}

/*****************************************************************
**************
* 函 数 名              : LcdWriteCom
* 函数功能              : 向 LCD 写入一个字节的命令
* 输    入              : com
* 输    出              : 无
*****************************************************************
*************/
#ifndef  LCD1602_4PINS    //当没有定义这个 LCD1602_4PINS 时
void LcdWriteCom(uchar com)        //写入命令
{
     LCD1602_E = 0;        //使能
     LCD1602_RS = 0;      //选择发送命令
     LCD1602_RW = 0;          //选择写入

     LCD1602_DATAPINS = com;        //放入命令
     Lcd1602_Delay1ms(1);          //等待数据稳定

     LCD1602_E = 1;                 //写入时序
     Lcd1602_Delay1ms(5);        //保持时间
     LCD1602_E = 0;
}
#else
void LcdWriteCom(uchar com)        //写入命令
{
     LCD1602_E = 0;    //使能清零
     LCD1602_RS = 0;  //选择写入命令
     LCD1602_RW = 0;       //选择写入
```

```
    LCD1602_DATAPINS = com;    //由于 4 位的接线是接到 P0 口的高四位，所以传送高四位不用改
    Lcd1602_Delay1ms(1);

    LCD1602_E = 1;    //写入时序
    Lcd1602_Delay1ms(5);
    LCD1602_E = 0;

//  Lcd1602_Delay1ms(1);
    LCD1602_DATAPINS = com << 4; //发送低四位
    Lcd1602_Delay1ms(1);

    LCD1602_E = 1;    //写入时序
    Lcd1602_Delay1ms(5);
    LCD1602_E = 0;
}
#endif
/*****************************************************************************
**************
* 函 数 名             : LcdWriteData
* 函数功能             : 向 LCD 写入一个字节的数据
* 输    入             : dat
* 输    出             : 无
*****************************************************************************
*************/
#ifndef  LCD1602_4PINS
void LcdWriteData(uchar dat)            //写入数据
{
    LCD1602_E = 0;  //使能清零
    LCD1602_RS = 1; //选择输入数据
    LCD1602_RW = 0;     //选择写入

    LCD1602_DATAPINS = dat; //写入数据
    Lcd1602_Delay1ms(1);

    LCD1602_E = 1;     //写入时序
    Lcd1602_Delay1ms(5);     //保持时间
    LCD1602_E = 0;
}
#else
void LcdWriteData(uchar dat)            //写入数据
{
    LCD1602_E = 0;      //使能清零
    LCD1602_RS = 1;    //选择写入数据
```

```
    LCD1602_RW = 0;         //选择写入

    LCD1602_DATAPINS = dat;     //由于 4 位的接线是接到 P0 口的高四位，所以传送高四位不用改
    Lcd1602_Delay1ms(1);

    LCD1602_E = 1;      //写入时序
    Lcd1602_Delay1ms(5);
    LCD1602_E = 0;

    LCD1602_DATAPINS = dat << 4; //写入低四位
    Lcd1602_Delay1ms(1);

    LCD1602_E = 1;      //写入时序
    Lcd1602_Delay1ms(5);
    LCD1602_E = 0;
}
#endif
/*****************************************************************
**************
* 函 数 名          : LcdInit()
* 函数功能        : 初始化 LCD 屏
* 输    入          : 无
* 输    出          : 无
*****************************************************************
*************/
#ifndef       LCD1602_4PINS
void LcdInit()                                  //LCD 初始化子程序
{
    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
}
#else
void LcdInit()                                  //LCD 初始化子程序
{
    LcdWriteCom(0x32);    //将 8 位总线转为 4 位总线
    LcdWriteCom(0x28);    //在四位线下的初始化
    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
```

```
}
#endif

#ifndef __LCD_H_
#define __LCD_H_
/*********************************
当使用的是 4 位数据传输的时候定义，
使用 8 位取消这个定义
*********************************/
#define LCD1602_4PINS

/*********************************
包含头文件
*********************************/
#include<reg51.h>

//---重定义关键词---//
#ifndef uchar
#define uchar unsigned char
#endif

#ifndef uint
#define uint unsigned int
#endif

/*********************************
PIN 口定义
*********************************/
#define LCD1602_DATAPINS P0
sbit LCD1602_E=P2^7;
sbit LCD1602_RW=P2^5;
sbit LCD1602_RS=P2^6;

/*********************************
函数声明
*********************************/
/*在 51 单片机 12MHZ 时钟下的延时函数*/
void Lcd1602_Delay1ms(uint c);     //误差  0us
/*LCD1602 写入 8 位命令子函数*/
void LcdWriteCom(uchar com);
/*LCD1602 写入 8 位数据子函数*/
void LcdWriteData(uchar dat) ;
/*LCD1602 初始化子程序*/
void LcdInit();
```

```
#endif
```

第十四讲 时钟芯片 DS1302

DS1302 是DALLAS 公司推出的涓流充电时钟芯片，内含有一个实时时钟/日历和31 字节静态RAM ，通过简单的串行接口与单片机进行通信。实时时钟/日历电路提供秒、分、时、日、周、月、年的信息，每月的天数和闰年的天数可自动调整。时钟操作可通过AM/PM 指示决定采用24 或12 小时格式。DS1302 与单片机之间能简单地采用同步串行的方式进行通信，仅需用到三个口线：（1）RES 复位（2）I/O 数据线（3）SCLK 串行时钟。时钟/RAM 的读/写数据以一个字节或多达31个字节的字符组方式通信。DS1302 工作时功耗很低保持数据和时钟信息时功率小于1mW。

DS1302由DS1202改进而来增加了以下的特性：双电源管脚用于主电源和备份电源供应，Vcc1为可编程涓流充电电源，附加七个字节存储器。它广泛应用于电话、传真、便携式仪器以及电池供电的仪器仪表等产品领域下面。将主要的性能指标作一综合：

★ 实时时钟具有能计算2100 年之前的秒、分、时、日、星期、月、年的能力，还有闰年调整的能力

★ 31 8位暂存数据存储RAM

★ 串行I/O口方式使得管脚数量最少

★ 宽范围工作电压2.0 5.5V

★ 工作电流2.0V时,小于300nA

★ 读/写时钟或RAM 数据时有两种传送方式单字节传送和多字节传送字符组方式

★ 8 脚DIP封装或可选的8脚SOIC封装根据表面装配

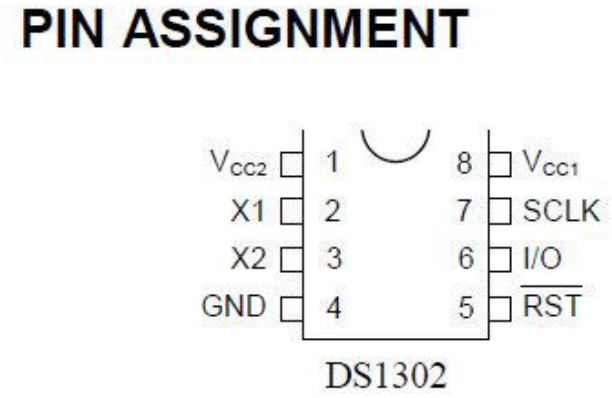
★ 简单3线接口

★ 与TTL兼容Vcc=5V

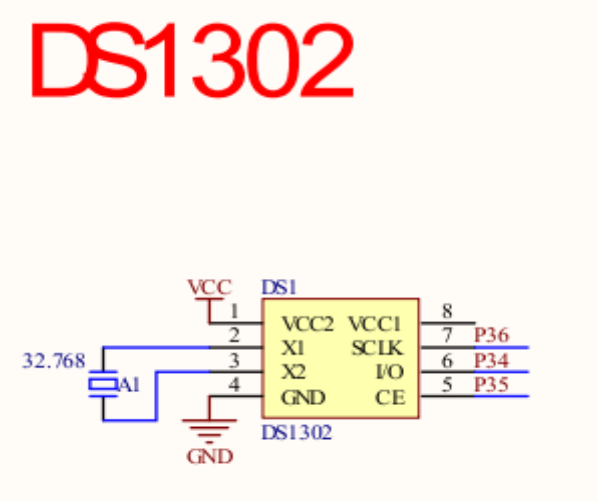
★ 可选工业级温度范围-40---+85

★ 双电源管用于主电源和备份电源供应

以上是DS1302的一些全面的预览，以下为DS1302管脚图：



1) VCC2：主用电源引脚
2) X1、X2：DS1302外部晶振引脚
3) GND：地
4) RST：复位引脚
5) I/O：串行数据引脚，数据输出或者输入都从这个引脚
6) SCLK：串行时钟引脚
7) VCC1：备用电源



1) VCC 为主电源接5V，CX10 为滤波电容
2) 2、 外接32.768K 的晶振
3) 3、 5、6、7 脚分别与控制器相联，注意外部4.7K 上拉电阻
4) 4、 备用电源脚，注意是3.3V，DS1302 要求备用电源电压稍微低于主用电源

下面讲讲DS1302 的具体操作。

操作DS1302的大致过程，就是将各种数据写入DS1302的寄存器，以设置它当前的时间已经格式。然后使DS1302开始运作，DS1302时钟会按照设置情况运转，再用单片机将其寄存器内的数据读出。再用液晶显示，就是我们常说的简易电子钟。所以总的来说DS1302的操作分2步（显示部分属于液晶显示的内容，不属于DS1302本身的内容）但是在讲述操作时序之前，我们要先看看寄存器：

ADDRESS/COMMAND BYTE Figure 2

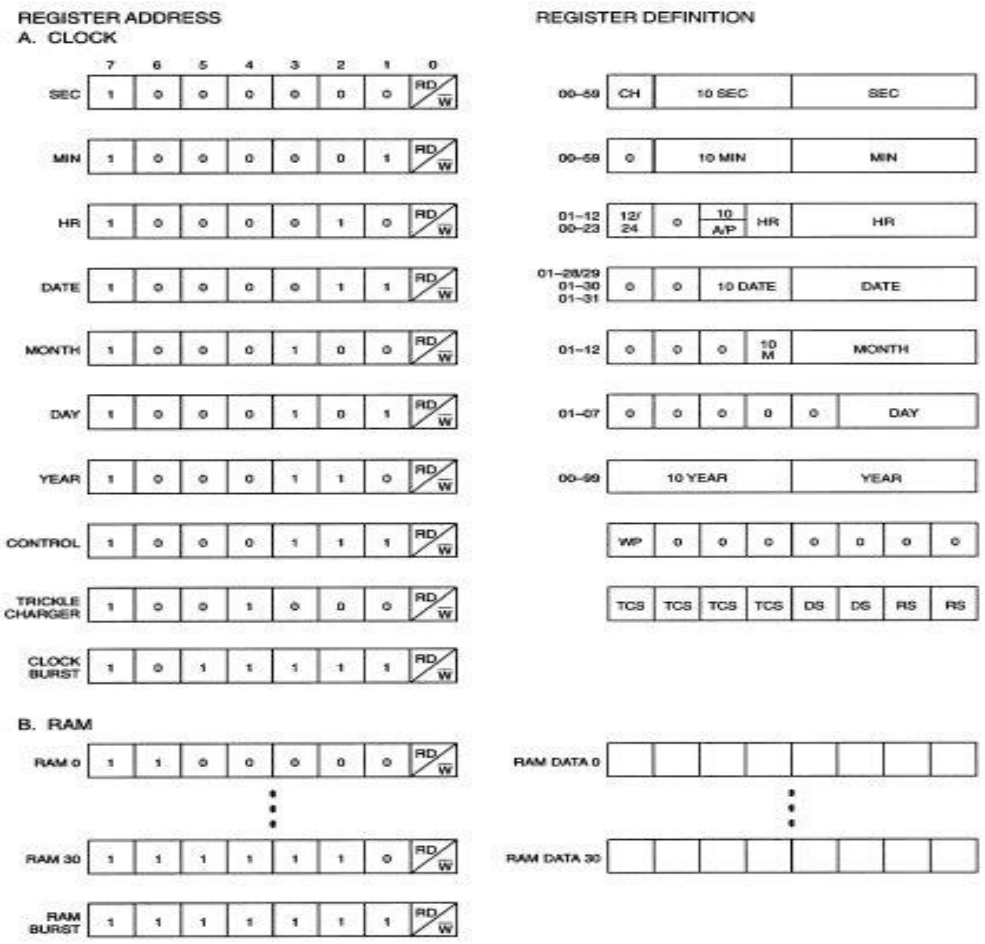
7	6	5	4	3	2	1	0
1	RAM / $\overline{CK}$	A4	A3	A2	A1	A0	RD / $\overline{W}$

上图是DS1302的寄存器样式，我们看到：

1、 第7 位永远都是1

2、 第6 位，1表示RAM，寻址内部存储器地址；0表示CK，寻址内部寄存器；

第5 位到第1 位，为RAM 或者寄存器的地址；最低位，高电平表示RD：即下一步操作将要“读”；低电平表示W：即下一步操作将要“写”。（与AT24C02寄存器类似，这点要理解好）。下面是DS1302的内部寄存器和RAM：



上图左边为寄存器和RAM的地址，右边为具体内容。各个寄存器的最高位都是1，最低位都是“RD/W”，比如要读秒寄存器则命令为10000101，反之写为10000100，要注意其含义。（图片不是很清楚，看不清楚的朋友用软件放大）我们一个一个看：

1、 SEC:秒寄存器，注意具体右边内容：低四位为SEC,高的次三位为10SEC。最高位CH 为

2、 DS1302 的运行标志，当CH=0时，DS1302 内部时钟运行，反之CH=1 时停止；

3、 MIN：分寄存器；

4、 HR:时寄存器，最高位为12/24 小时的格式选择位，该位为1时表示12 小时格式。当设置为12小时显示格式时，第5位的高电平表示下午（PM）；而当设置为24 小时格式时，第5位位具体的时间数据。

5、 DATE：日寄存器；

6、 MONTH：月寄存器；

7、 DAY：周寄存器，注意一周只有7天，所以该寄存器只有低三位有效；

8、 YEAR：年寄存器；

9、 CONTROL：写保护寄存器，当该寄存器最高位WP 为1 时，DS1302 只读不写，所以要在往DS1302 写数据之前确保WP 为0；

10、TRICKLE CHARGE REGISTER：涓细电流充电设置寄存器，我们知道，当DS1302 掉电时，可以马上调用外部电源保护时间数据。该寄存器就是配置备用电源的充电选项的。其中高四位（4个TCS）只有在1010 的情况下才能使用充电选项；低四位的情况，与DS1302内部电路有关，有点意思，下文详细讲述。

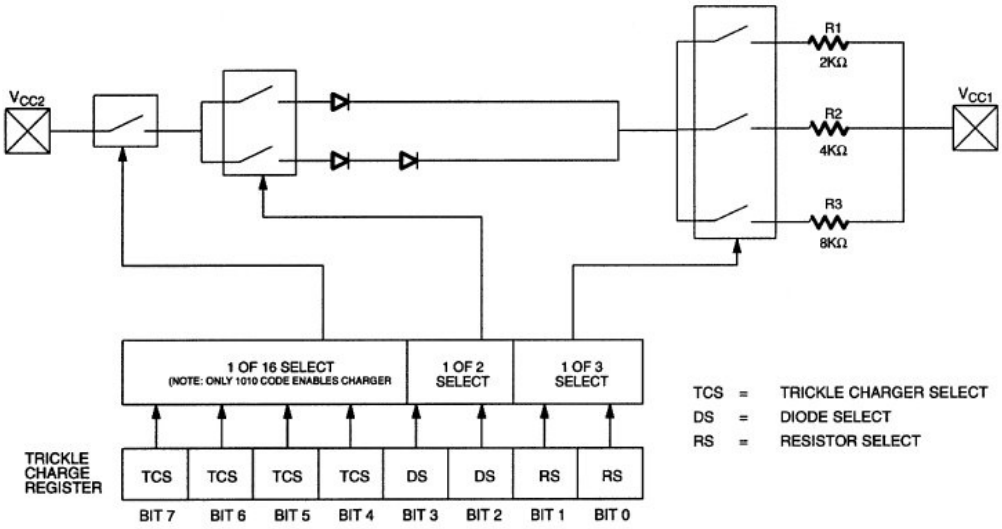
11、CLOCK BURST：批量读写操作设置寄存器，设置该寄存器后，可以对DS1302的各个寄存器进行连续写入。DS1302的另外一种读写方式。笔者还没用过，感兴趣的朋友可以尝试。最后还有一点，前文说过，DS1302有31个字节的存储空间，但是大家要看到的是，这31个存储空间，最后一个是RAM BURST的寄存器，设置该寄存器可以达到对RAM 连续读写的作用。所以DS1302的可用存储空间实际上为30个字节。

现在我们来看看DS1302的涓细电流充电的设置：

以下来自英文原版PDF：

The trickle charge select (TCS) bits (bits4 -7) control the selection of the trickle charger. In order to prevent accidental enabling, only a pattern of 1010 will enable the trickle charger. All other patterns will disable the trickle charger. The DS1302 powers up with the trickle charger disabled. The diode select (DS) bits (bits 2 - 3) select whether onediode or two diodes are connected between VCC2 and VCC1. If DS is 01, one diode is selected or if DS is10, two diodes are selected. If DS is 00 or 11, the trickle charger is disabled independently of TCS. The RS bits (bits 0 -1) select the resistor that is connected between VCC2 and VCC1. The resistor selected by the resistor select (RS) bits is as follows:

DS1302 PROGRAMMABLE TRICKLE CHARGER Figure 5



好，英文水平不好也没关系：

看到这句“The trickle charge select (TCS) bits (bits4 -7) control the selection of the trickle charger. In order to prevent accidental enabling, only a pattern of 1010 will enable the trickle charger”，这句话是说“，TCS为用以控制涓细电流充电功能，为了防止意外产生， 只当TCS位（四位）为1010时涓细电流充电功能才会生效”所以刚才提到“其中高四位（4个TCS）只有在1010的情况下才能使用充电选项”。

那DS呢？“If DS is 01, one diode is selected or if DS is10, two diodes are selected. If DS is 00 or 11, the trickle charger is disabled independently of TCS”，既是说，如果两个DS位为01，则只有1个二极管接入电路，如果DS为10则表示有2个二极管接入，如果DS为00或者11，则充电功能由TCS单独控制”。看到上部电路三个二极管处，DS为01时接入1个二极管，对应上面的开关闭合,为10时表示2个二极管接入,对应下面的开关闭合为00或者11时笔者认为两个开关都不闭合，充电电流不经过二极管。

对应的，RS的设置也相仿：“The RS bits (bits 0 -1) select the resistor that is connected between VCC2 and VCC1. The resistor selected by the resistor select (RS) bits is as follows:

RS Bits	Resistor	Typical Value
00	None	None
01	R1	2 kΩ
10	R2	4 kΩ
11	R3	8 kΩ

意思是：RS位用以选择在VCC1和VCC2直接接入什么样的电阻：

1、 当RS为00时，不接入电阻；

2、 当RS为01时，接入典型值为2K电阻，对应电路图中的R1；

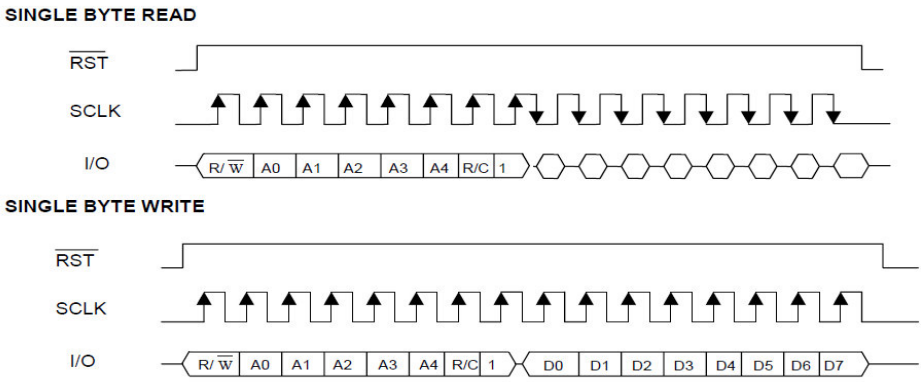
3、 当RS为10时，接入典型值为4K电阻，对应电路图中的R2；

4、 当RS为11时，接入典型值为8K电阻，对应电路图中的R3；

好了，至此我们知道了，DS和RS的作用是配置接入电路中的二极管和电阻，有什么用呢？

笔者认为这些二极管和电阻是分压和限流用的，以调整涓细充电电流的大小。

我们可以看看DS1302的读写时序了：



上图就是DS1302的三个时序：复位时序，单字节写时序，单字节读时序；

RST：复位时序，即在RST引脚产生一个正脉冲，在整个读写器件，RST要保持高电平，一次字节读写完毕之后，要注意把RST返回低电平准备下次读写周期；

SINGLE BYTE READ：单字节读，注意读之前还是要先对寄存器写命令，从最低位开始写；大家细心看可以看到，写数据是在SCLK的上升沿实现，而读数据在SCLK的下降沿实现，所以，在单字节读时序中，写命令的第八个上升沿结束后紧接着的第八个下降沿就将要读寄存器的第一位数据读到数据线上了！这个就是DS1302

操作中最特别的地方。当然读出来的数据也是最低位开始。

SINGLE BUTE WRITE：单字节写，两个字节的数据配合16个上升沿将数据写入即可。

程序注意事项：

★要记得在操作DS1302之前关闭写保护；

★注意用延时来降低单片机的速度以配合器件时序

★DS1302读出来的数据是BCD码形式，要转换成我们习惯的10进制，转换方法在源程序里；

★读取字节之前，将IO设置为输入口，读取完之后，要将其改回输出口；

★在写程序的时候，建议实现开辟数组（内存空间）来集中放置DS1302的一系列数据，方便以后扩展键盘输入；

打开 

ds1302	2014/6/10 8:41	c_file	4 KB
ds1302.h	2014/6/10 8:41	H 文件	1 KB
ds1302.LST	2014/6/10 8:41	LST 文件	7 KB
ds1302.OBJ	2014/6/10 8:41	OBJ 文件	6 KB
lcd	2014/6/10 8:41	c_file	4 KB
lcd.h	2014/6/10 8:41	H 文件	1 KB
lcd.LST	2014/6/10 8:41	LST 文件	7 KB
lcd.OBJ	2014/6/10 8:41	OBJ 文件	4 KB
main	2014/6/10 8:41	c_file	5 KB
main.LST	2014/6/10 8:41	LST 文件	9 KB
main.OBJ	2014/6/10 8:41	OBJ 文件	8 KB
pro	2014/6/10 8:41	文件	16 KB
pro.hex	2014/6/10 8:41	HEX 文件	3 KB
pro.lnp	2014/6/10 8:41	LNP 文件	1 KB
pro.M51	2014/6/10 8:41	M51 文件	19 KB
pro.plg	2014/9/15 11:55	PLG 文件	1 KB
pro.uvopt	2014/6/25 15:13	UVOPT 文件	58 KB
pro	2014/6/10 8:41	礦ision4 Project	14 KB
pro_uvopt.bak	2014/6/14 13:28	BAK 文件	58 KB
pro_uvproj.bak	2014/6/10 8:41	BAK 文件	0 KB
STARTUP.A51	2014/6/10 8:41	A51 文件	7 KB
STARTUP.LST	2014/6/10 8:41	LST 文件	14 KB
STARTUP.OBJ	2014/6/10 8:41	OBJ 文件	1 KB

/**********

```
**********
* 实验名                : 万年历实验
* 使用的IO      :
* 实验效果          :1602显示时钟，按K3进入时钟设置，按K1选择设置的时分秒日月，按K2选择
*选择设置加1。
*   注意                    :
**********************************************************************
**********/

#include<reg51.h>
#include"lcd.h"
#include"ds1302.h"

sbit K1=P3^1;
sbit K2=P3^0;
sbit K3=P3^2;
sbit K4=P3^3;

void Int0Configuration();
void LcdDisplay();
unsigned char SetState,SetPlace;
void Delay10ms(void);   //误差 0us
/**********************************************************************
**********
* 函数名          : main
* 函数功能        : 主函数
* 输入            : 无
* 输出              : 无
```

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

void main()
{
    unsigned char i;
    Int0Configuration();
    LcdInit();
    Ds1302Init();
    while(1)
    {
        if(SetState==0)
        {
            Ds1302ReadTime();
        }
        else
        {
            if(K1==0)      //检测按键K1是否按下
            {
                Delay10ms();  //消除抖动
                if(K1==0)
                {
                    SetPlace++;
                    if(SetPlace>=7)
                        SetPlace=0;
                }

                while((i<50)&&(K1==0))    //检测按键是否松开
                {
```

```
                Delay10ms();
                i++;
            }
            i=0;
        }
        if(K2==0)      //检测按键K2是否按下
        {
            Delay10ms();  //消除抖动
            if(K2==0)
            {
                TIME[SetPlace]++;
                if((TIME[SetPlace]&0x0f)>9)                    //换成
BCD码。
                {
                    TIME[SetPlace]=TIME[SetPlace]+6;
                }
                if((TIME[SetPlace]>=0x60)&&(SetPlace<2))      //分秒
只能到59
                {
                    TIME[SetPlace]=0;
                }
                if((TIME[SetPlace]>=0x24)&&(SetPlace==2))     //小时
只能到23
                {
                    TIME[SetPlace]=0;
                }
                if((TIME[SetPlace]>=0x32)&&(SetPlace==3))     //日只
能到31
                {
```

```
                    TIME[SetPlace]=0;
                }
                if((TIME[SetPlace]>=0x13)&&(SetPlace==4))     //月只
能到12
                {
                    TIME[SetPlace]=0;
                }
                if((TIME[SetPlace]>=0x7)&&(SetPlace==5))      //周只
能到7
                {
                    TIME[SetPlace]=1;
                }
//                  if(SetPlace==5)       //月只能到12
//                  {
//                      TIME[SetPlace]=;
//                  }
            }

            while((i<50)&&(K2==0))    //检测按键是否松开
            {
                Delay10ms();
                i++;
            }
            i=0;

        }
    }
  LcdDisplay();
  }
```

```
}
/*********************************************************************
***********
* 函数名         : LcdDisplay()
* 函数功能       : 显示函数
* 输入           : 无
* 输出           : 无
*********************************************************************
**********/

void LcdDisplay()
{
    LcdWriteCom(0x80+0X40);
    LcdWriteData('0'+TIME[2]/16);             //时
    LcdWriteData('0'+(TIME[2]&0x0f));
    LcdWriteData('-');
    LcdWriteData('0'+TIME[1]/16);             //分
    LcdWriteData('0'+(TIME[1]&0x0f));
    LcdWriteData('-');
    LcdWriteData('0'+TIME[0]/16);             //秒
    LcdWriteData('0'+(TIME[0]&0x0f));

    LcdWriteCom(0x80);
    LcdWriteData('2');
    LcdWriteData('0');
    LcdWriteData('0'+TIME[6]/16);          //年
    LcdWriteData('0'+(TIME[6]&0x0f));
    LcdWriteData('-');
```

```
    LcdWriteData('0'+TIME[4]/16);          //月
    LcdWriteData('0'+(TIME[4]&0x0f));
    LcdWriteData('-');
    LcdWriteData('0'+TIME[3]/16);          //日
    LcdWriteData('0'+(TIME[3]&0x0f));
    LcdWriteCom(0x8D);
    LcdWriteData('0'+(TIME[5]&0x07));  //星期
}
/*************************************************************************
***********
* 函数名          : Int0Configuration()
* 函数功能        : 配置外部中断0
* 输入            : 无
* 输出             : 无
*************************************************************************
**********/

void Int0Configuration()
{
    //设置INT0
    IT0=1;//跳变沿出发方式（下降沿）
    EX0=1;//打开INT0的中断允许。
    EA=1;//打开总中断
}
/*************************************************************************
***********
* 函数名          : Int0()
* 函数功能        : 外部中断0 中断函数
* 输入            : 无
```

```
* 输出             : 无
*****************************************************************************
**********/

void Int0() interrupt 0
{
    Delay10ms();
    if(K3==0)
    {
        SetState=~SetState;
        SetPlace=0;
        Ds1302Init();
    }
}
/*****************************************************************************
***********
* 函数名         : Delay10ms
* 函数功能       : 延时函数，延时10ms
* 输入           : 无
* 输出             : 无
*****************************************************************************
**********/
void Delay10ms(void)   //误差 0us
{
    unsigned char a,b,c;
    for(c=1;c>0;c--)
        for(b=38;b>0;b--)
            for(a=130;a>0;a--);
}
```

```
#include"ds1302.h"

//---DS1302写入和读取时分秒的地址命令---//
//---秒分时日月周年 最低位读写位;-------//
uchar code READ_RTC_ADDR[7] = {0x81, 0x83, 0x85, 0x87, 0x89, 0x8b, 0x8d};
uchar code WRITE_RTC_ADDR[7] = {0x80, 0x82, 0x84, 0x86, 0x88, 0x8a, 0x8c};

//---DS1302时钟初始化2013年1月1日星期二12点00分00秒。---//
//---存储顺序是秒分时日月周年,存储格式是用BCD码---//
uchar TIME[7] = {0, 0, 0x12, 0x01, 0x01, 0x02, 0x13};

/*************************************************************************
***********
* 函 数 名         : Ds1302Write
* 函数功能         : 向DS1302命令（地址+数据）
* 输    入         : addr,dat
* 输    出         : 无
*************************************************************************
**********/

void Ds1302Write(uchar addr, uchar dat)
{
    uchar n;
    RST = 0;
    _nop_();

    SCLK = 0;//先将SCLK置低电平。
    _nop_();
    RST = 1; //然后将RST(CE)置高电平。
```

```
    _nop_();

    for (n=0; n<8; n++)//开始传送八位地址命令
    {
        DSIO = addr & 0x01;//数据从低位开始传送
        addr >>= 1;
        SCLK = 1;//数据在上升沿时，DS1302读取数据
        _nop_();
        SCLK = 0;
        _nop_();
    }
    for (n=0; n<8; n++)//写入8位数据
    {
        DSIO = dat & 0x01;
        dat >>= 1;
        SCLK = 1;//数据在上升沿时，DS1302读取数据
        _nop_();
        SCLK = 0;
        _nop_();
    }

    RST = 0;//传送数据结束
    _nop_();
}

/*************************************************************************
***********
* 函 数 名          : Ds1302Read
* 函数功能          : 读取一个地址的数据
```

```
* 输    入          : addr
* 输    出          : dat
*********************************************************************
**********/

uchar Ds1302Read(uchar addr)
{
    uchar n,dat,dat1;
    RST = 0;
    _nop_();

    SCLK = 0;//先将SCLK置低电平。
    _nop_();
    RST = 1;//然后将RST(CE)置高电平。
    _nop_();

    for(n=0; n<8; n++)//开始传送八位地址命令
    {
        DSIO = addr & 0x01;//数据从低位开始传送
        addr >>= 1;
        SCLK = 1;//数据在上升沿时，DS1302读取数据
        _nop_();
        SCLK = 0;//DS1302下降沿时，放置数据
        _nop_();
    }
    _nop_();
    for(n=0; n<8; n++)//读取8位数据
    {
        dat1 = DSIO;//从最低位开始接收
```

```
        dat = (dat>>1) | (dat1<<7);
        SCLK = 1;
        _nop_();
        SCLK = 0;//DS1302下降沿时，放置数据
        _nop_();
    }

    RST = 0;
    _nop_();  //以下为DS1302复位的稳定时间,必须的。
    SCLK = 1;
    _nop_();
    DSIO = 0;
    _nop_();
    DSIO = 1;
    _nop_();
    return dat;
}

/*************************************************************************
***********
* 函 数 名         : Ds1302Init
* 函数功能       : 初始化DS1302.
* 输    入         : 无
* 输    出         : 无
*************************************************************************
**********/

void Ds1302Init()
{
```

```
    uchar n;
    Ds1302Write(0x8E,0X00);        //禁止写保护，就是关闭写保护功能
    for (n=0; n<7; n++)//写入7个字节的时钟信号：分秒时日月周年
    {
        Ds1302Write(WRITE_RTC_ADDR[n],TIME[n]);
    }
    Ds1302Write(0x8E,0x80);        //打开写保护功能
}

/*************************************************************************
***********
* 函 数 名          : Ds1302ReadTime
* 函数功能          : 读取时钟信息
* 输    入          : 无
* 输    出          : 无
*************************************************************************
**********/

void Ds1302ReadTime()
{
    uchar n;
    for (n=0; n<7; n++)//读取7个字节的时钟信号：分秒时日月周年
    {
        TIME[n] = Ds1302Read(READ_RTC_ADDR[n]);
    }

}
#ifndef __DS1302_H_
#define __DS1302_H_
```

```
//---包含头文件---//
#include<reg51.h>
#include<intrins.h>

//---重定义关键词---//
#ifndef uchar
#define uchar unsigned char
#endif

#ifndef uint
#define uint unsigned int
#endif

//---定义ds1302使用的IO口---//
sbit DSIO=P3^4;
sbit RST=P3^5;
sbit SCLK=P3^6;

//---定义全局函数---//
void Ds1302Write(uchar addr, uchar dat);
uchar Ds1302Read(uchar addr);
void Ds1302Init();
void Ds1302ReadTime();

//---加入全局变量--//
extern uchar TIME[7];    //加入全局变量

#endif
```

```
#include"lcd.h"

/*******************************************************************************
***********
* 函 数 名          : Lcd1602_Delay1ms
* 函数功能          : 延时函数，延时1ms
* 输    入          : c
* 输    出          : 无
* 说    名          : 该函数是在12MHZ晶振下，12分频单片机的延时。
*******************************************************************************
**********/

void Lcd1602_Delay1ms(uint c)   //误差 0us
{
    uchar a,b;
    for (; c>0; c--)
    {
        for (b=199;b>0;b--)
        {
          for(a=1;a>0;a--);
        }
    }

}

/*******************************************************************************
***********
* 函 数 名          : LcdWriteCom
* 函数功能          : 向LCD写入一个字节的命令
```

```
* 输    入          : com
* 输    出          : 无
*****************************************************************************
**********/
#ifndef    LCD1602_4PINS  //当没有定义这个LCD1602_4PINS时
void LcdWriteCom(uchar com)    //写入命令
{
    LCD1602_E = 0;     //使能
    LCD1602_RS = 0;      //选择发送命令
    LCD1602_RW = 0;      //选择写入

    LCD1602_DATAPINS = com;      //放入命令
    Lcd1602_Delay1ms(1);     //等待数据稳定

    LCD1602_E = 1;              //写入时序
    Lcd1602_Delay1ms(5);    //保持时间
    LCD1602_E = 0;
}
#else
void LcdWriteCom(uchar com)    //写入命令
{
    LCD1602_E = 0;     //使能清零
    LCD1602_RS = 0;    //选择写入命令
    LCD1602_RW = 0;    //选择写入

    LCD1602_DATAPINS = com; //由于4位的接线是接到P0口的高四位，所以传
送高四位不用改
    Lcd1602_Delay1ms(1);
```

```
    LCD1602_E = 1;     //写入时序
    Lcd1602_Delay1ms(5);
    LCD1602_E = 0;

//  Lcd1602_Delay1ms(1);
    LCD1602_DATAPINS = com << 4; //发送低四位
    Lcd1602_Delay1ms(1);

    LCD1602_E = 1;     //写入时序
    Lcd1602_Delay1ms(5);
    LCD1602_E = 0;
}
#endif
/*******************************************************************************
***********
* 函 数 名         : LcdWriteData
* 函数功能         : 向LCD写入一个字节的数据
* 输    入         : dat
* 输    出         : 无
*******************************************************************************
**********/
#ifndef    LCD1602_4PINS
void LcdWriteData(uchar dat)           //写入数据
{
    LCD1602_E = 0;    //使能清零
    LCD1602_RS = 1;   //选择输入数据
    LCD1602_RW = 0;   //选择写入

    LCD1602_DATAPINS = dat; //写入数据
```

```
    Lcd1602_Delay1ms(1);

    LCD1602_E = 1;   //写入时序
    Lcd1602_Delay1ms(5);   //保持时间
    LCD1602_E = 0;
}
#else
void LcdWriteData(uchar dat)           //写入数据
{
    LCD1602_E = 0;      //使能清零
    LCD1602_RS = 1;     //选择写入数据
    LCD1602_RW = 0;     //选择写入

    LCD1602_DATAPINS = dat; //由于4位的接线是接到P0口的高四位，所以传送高四位不用改
    Lcd1602_Delay1ms(1);

    LCD1602_E = 1;      //写入时序
    Lcd1602_Delay1ms(5);
    LCD1602_E = 0;

    LCD1602_DATAPINS = dat << 4; //写入低四位
    Lcd1602_Delay1ms(1);

    LCD1602_E = 1;      //写入时序
    Lcd1602_Delay1ms(5);
    LCD1602_E = 0;
}
#endif
```

```
/*********************************************************************
***********
* 函 数 名         : LcdInit()
* 函数功能         : 初始化LCD屏
* 输    入         : 无
* 输    出         : 无
*********************************************************************
**********/
#ifndef       LCD1602_4PINS
void LcdInit()                          //LCD初始化子程序
{
    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
}
#else
void LcdInit()                          //LCD初始化子程序
{
    LcdWriteCom(0x32);     //将8位总线转为4位总线
    LcdWriteCom(0x28);     //在四位线下的初始化
    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
}
#endif
#ifndef __LCD_H_
```

```
#define __LCD_H_
/**********************************
当使用的是4位数据传输的时候定义，
使用8位取消这个定义
**********************************/
#define LCD1602_4PINS

/**********************************
包含头文件
**********************************/
#include<reg51.h>

//---重定义关键词---//
#ifndef uchar
#define uchar unsigned char
#endif

#ifndef uint
#define uint unsigned int
#endif

/**********************************
PIN口定义
**********************************/
#define LCD1602_DATAPINS P0
sbit LCD1602_E=P2^7;
sbit LCD1602_RW=P2^5;
sbit LCD1602_RS=P2^6;
```

```
/**********************************
函数声明
**********************************/
/*在51单片机12MHZ时钟下的延时函数*/
void Lcd1602_Delay1ms(uint c);   //误差 0us
/*LCD1602写入8位命令子函数*/
void LcdWriteCom(uchar com);
/*LCD1602写入8位数据子函数*/
void LcdWriteData(uchar dat)    ;
/*LCD1602初始化子程序*/
void LcdInit();

#endif
```

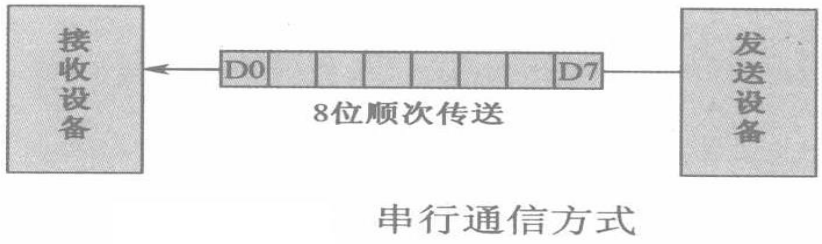
第十五讲 串口通信

随着单片机系统的广泛应用和计算机网络技术的普及，单片机的通信功能愈来愈显得重要。单片机通信是指单片机与计算机或单片机与单片机之间的信息交换，通常单片机与计算机之间的通信我们用的较多。

通信有并行和串行两种方式。在单片机系统以及现代单片机测控系统中，信息的交换多采用串行通信方式。

1. 串行通信方式

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，此时只需要一条数据线，外加一条公共信号地线和若干控制信号线。因为一次只能传送一位，所以对于一个字节的数据，至少要分 S 位才能传送完毕。



串行通信方式

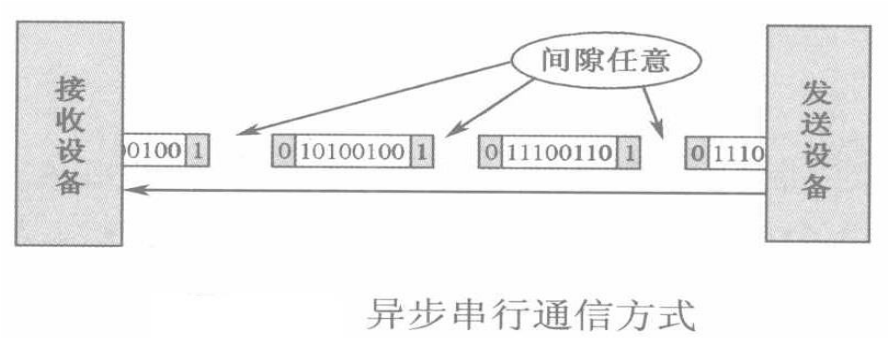
串行通信的必要过程是:发送时，要把并行数据变成串行数据发送到线路上去，接收时，要把串行信号再变成并行数据，这样才能被计算机及其他设备处理。

串行通信传输线少，长距离传送时成本低，且可以利用电话网等现成的设备，但数据的传送控制比并行通信复杂。

串行通信又有两种方式:异步串行通信和同步串行通信。

2. 异步串行通信方式

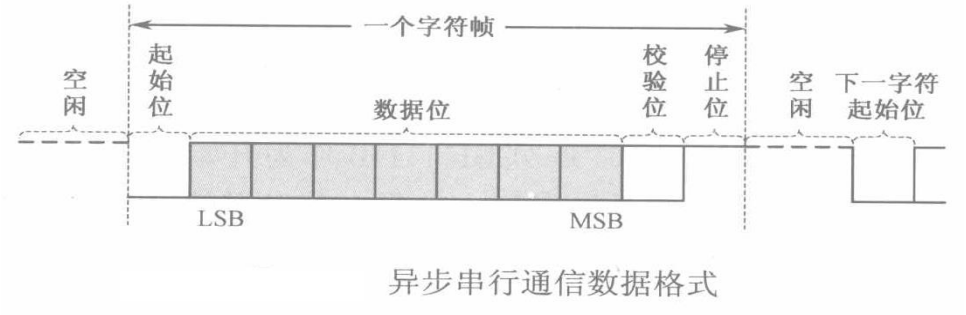
异步串行通信是指通信的发送与接收设备使用各自的时钟控制数据的发送和接收过程。为使双方收、发协调，要求发送和接收设备的时钟尽可能一致，



异步串行通信方式

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，但侮个字符中的各位是以固定的时间传送的，即字符之间不一定有“位间隔”的整数倍关系，但同一字符内的各位之间的距离均为“位间隔”的整数倍。

异步通信一帧字符信息由 4 部分组成:起始位、数据位、奇偶校验位和停止位，如下图所示。有的字符信息也有带空闲位形式，即在字符之间有空闲字符。



异步串行通信数据格式

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、校验位和停止位，各帧之间还有间隔，因此传输效率不高。在单片机与单片机之间，单片机与计算机之间通信时，通常采用异步串行通信方式。

3. 同步串行通信方式

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外同步和自同步两种方法实现。

4. 串行通信的制式

(1)单工。单工是指数据传输仅能沿一个方向，不能实现反向传输。

(2)半双工。半双工是指数据传输可以沿两个方向，但需要分时进行。

(3)全双工。全双工是指数据可以同时进行双向传输。

5. 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。接收方接收数据时同时对数据块(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。

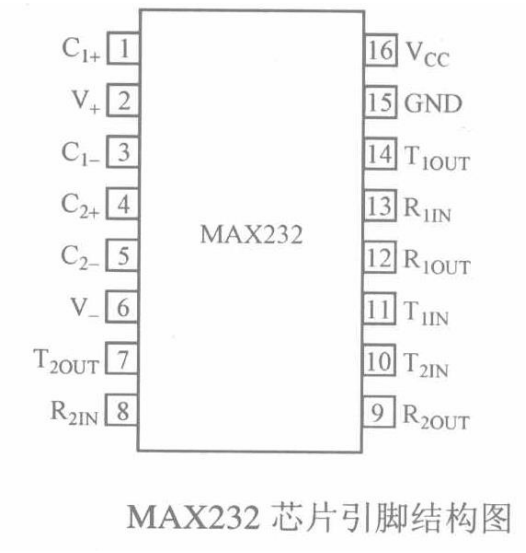
(3)循环冗余校验

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。

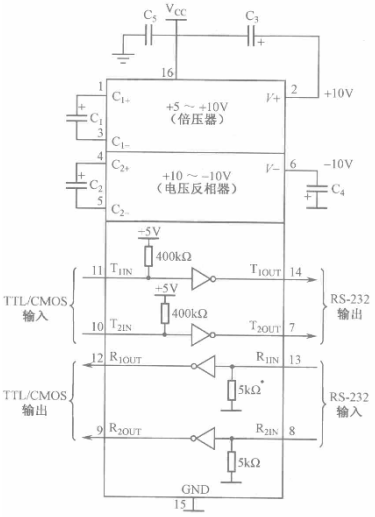
6. MAX232 芯片实现 RS-232 电平与 TTL 电平转换

MAX232 芯片是 MAXIM 公司生产的、包含两路接收器和驱动器的 IC 芯片，它的内部有一个电源电压变换器，可以把输入的+5V 电源电压变换成为 RS-232 输出电平所需的+10V 电压。所以，采用此芯片接口的串行通信系统只需单一的+5V 电源就可以了。对于没有+12V 电源的场合，其适应性更强，加之其价格适中，硬件

接口简单，所以被广泛采用。



MAX232 芯片引脚结构图



外围电路连接图

上右图中上半部分电容 C1，C2，C3，C4 及 V+，V-是电源变换电路部分。在实际应用中，器件对电源噪声很敏感，因此 Vcc 必须要对地加去祸电容 C5，其值为 0.1uF。按芯片手册中介绍，电容 C1，C2，C3,C4 应取 10uF/16V 的电解电容，经大量实验及实际应用，这 4 个电容都可以选用 0.1uF 的非极性瓷片电容代替 10uF/16V 的电解电容，在具体设计电路时，这 4 个电容要尽量靠近 MAX232 芯片，以提高抗干扰能力。

下半部分为发送和接收部分。实际应用中，T1IN,T2IN 可直接连接 TTL/CMOS 电平的 51 单片机串行发送端 TXD; R1OUT，R2OUT 可直接连接 TTL/CMOS 电平的 51 单片机的串行接收端 RXD;T1OUT,T2OUT 可直接连接代机的 RS-232 串口的接收端 RXD;R1IN,R2IN 可直接连接 PC 机的 RS-232 串口的发送端 TXD。

现从 MAX232 芯片中两路发送、接收中任选一路作为接口。要注意其发送、接收的引脚要对应。如使 T1IN 连接单片机的发送端 TXD，则 PC 机的 RS-232 接收端 RXD 一定要对应接 TIOUT 引脚。同时，R1OUT 连接单片机的 RXD 引脚，PC 机的 RS-232 发送端 TXD 对应接 R1IN 引脚。

7. 波特率

单片机或计算机在串口通信时的速率用波特率表示，它定义为每秒传输二进制代码的位数，即 1 波特=1 位/秒，单位是 bps(位/秒)。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)，这时的波特率为 10 位 X 240 个/秒=2400bps.

串行接口或终端直接传送串行信息位流的最大距离与传输速率及传输线的电气特性也有关。当传输线使用每 0.3m(约 1 英尺)有 50pF 电容的非平衡屏蔽双绞线时，传输距离随传输速率的增加而减小。当比特率超过 1000 bps 时，最大传输距离迅速下降，如 9600 bps 时最大距离下降到只有 76m(约 250 英尺)。因此我们在做串口通信实验选择较高速率传输数据时，尽量缩短数据线的长度，为了能使数据安全传输，即使是在较低传输速率下也不要使用太长的数据线。

8. 波特率的计算

在串行通信中，收、发双方对发送或接收数据的速率要有约定。通过编程可对单片机串行口设定为 4 种工作方式，其中方式 0 和方式 2 的波特率是固定的，而方式 1 和方式 3 的波特率是可变的，由定时器 T1 的溢出率来决定。

串行口的 4 种工作方式对应三种波特率。由于输入的移位时钟的来源不同，所以各种方式的波特率计算公式也不相同，以下是 4 种方式波特率的计算公式。

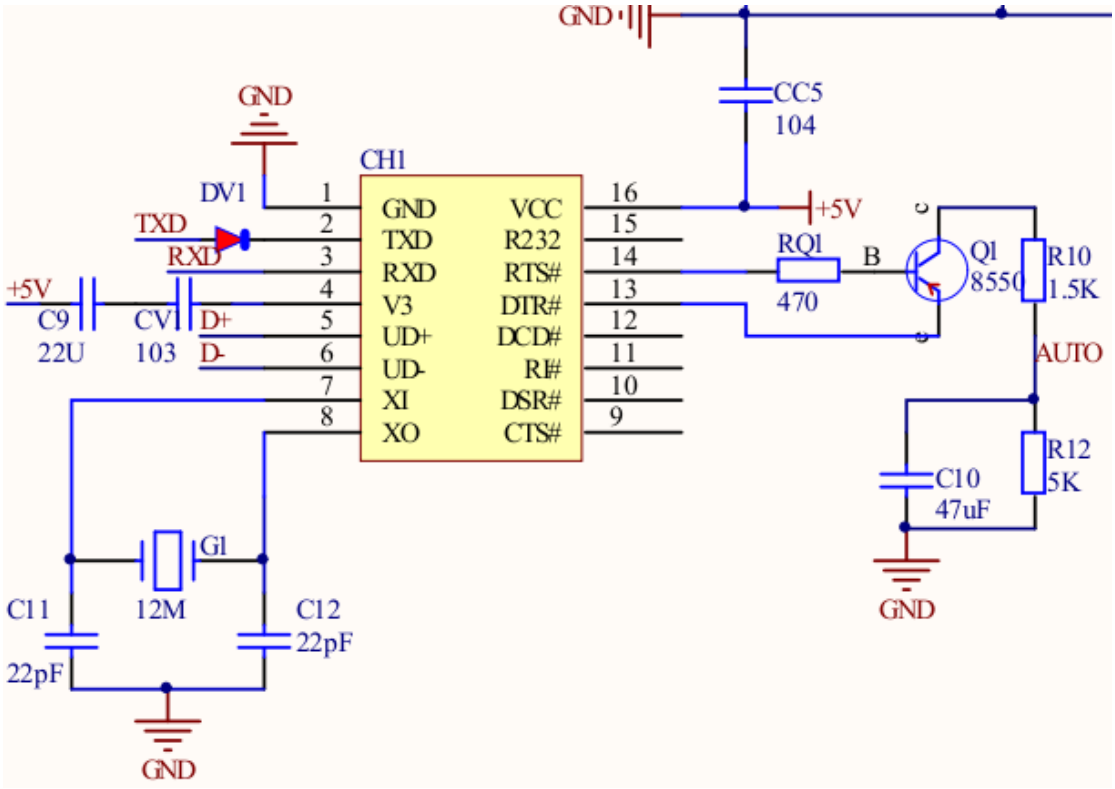
方式 0 的波特率 $=f_{osc}/12$。

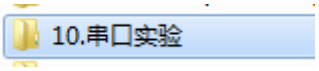
方式 1 的波特率 $=(2^{SMOD}/32)\times$（T1 溢出率）。

方式 2 的波特率 $=(2^{SMOD}/64)\times f_{osc}$。

方式 3 的波特率 $=(2^{SMOD}/32)\times$（T1 溢出率）。

开发板的电路图如下图



打开 

main._i	2014/6/10 8:41	_I 文件	1 KB
main	2014/8/18 16:44	c_file	2 KB
main.LST	2014/6/10 8:41	LST 文件	4 KB
main.OBJ	2014/6/10 8:41	OBJ 文件	3 KB
pro	2014/6/10 8:41	文件	3 KB
pro.hex	2014/6/10 8:41	HEX 文件	1 KB
pro.lnp	2014/6/10 8:41	LNP 文件	1 KB
pro.M51	2014/6/10 8:41	M51 文件	6 KB
pro.plg	2014/8/18 16:46	PLG 文件	1 KB
pro.uvopt	2014/8/18 16:46	UVOPT 文件	55 KB
pro	2014/6/10 8:41	礦ision4 Project	14 KB
pro_uvopt.bak	2014/6/10 8:41	BAK 文件	136 KB
pro_uvproj.bak	2014/6/10 8:41	BAK 文件	0 KB
STARTUP.A51	2014/6/10 8:41	A51 文件	7 KB
STARTUP.LST	2014/6/10 8:41	LST 文件	14 KB
STARTUP.OBJ	2014/6/10 8:41	OBJ 文件	1 KB

```
/*********************************************************************************
******
    * 实验名                  ：串口实验
    * 使用的 IO       ：P2
    * 实验效果         ：将接收到发送回电脑上面。
    *   注意                          ：
    ********************************************************************************
*****/

    #include<reg51.h>

    void UsartConfiguration();
    /*********************************************************************************
******
    * 函数名          ：main
    * 函数功能          ：主函数
    * 输入            ：无
```

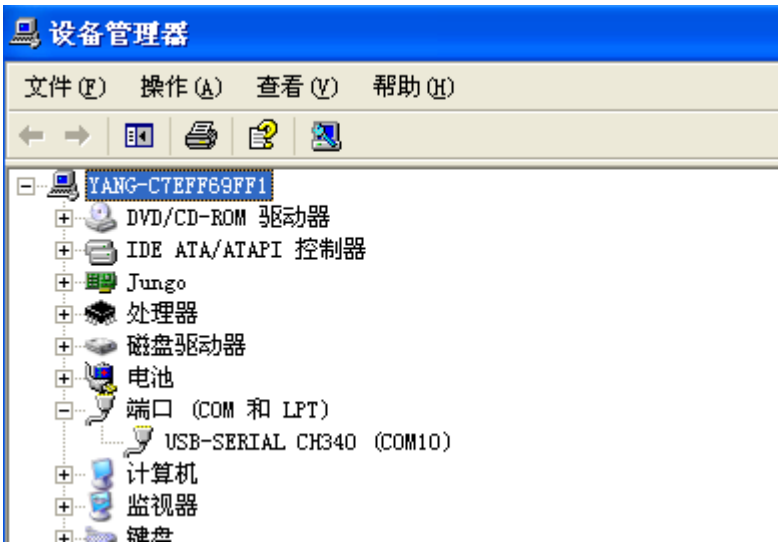
```
    * 输出            : 无
    ********************************************************************************
*****/
    void main()
    {
        UsartConfiguration();
        while(1)
        {
        }
    }
    /********************************************************************************
******
    * 函数名          :UsartConfiguration()
    * 函数功能          :设置串口
    * 输入            : 无
    * 输出            : 无
    ********************************************************************************
*****/

    void UsartConfiguration()
    {
        SCON=0X50;          //设置为工作方式 1
        TMOD=0X20;          //设置计数器工作方式 2
        PCON=0X80;          //波特率加倍
        TH1=0XF3;               //计数器初始值设置，注意波特率是 4800 的
        TL1=0XF3;
        ES=1;                       //打开接收中断
        EA=1;                       //打开总中断
        TR1=1;                  //打开计数器
    }
```

```
    /*************************************************************************
******
    * 函数名         :Delay(unsigned int i)
    * 函数功能         : 延时函数
    * 输入           : 无
    * 输出           : 无
    *************************************************************************
*****/
    void Usart() interrupt 4
    {
        unsigned char receiveData;

        receiveData=SBUF;//出去接收到的数据
        RI = 0;//清除接收中断标志位
        SBUF=receiveData;//将接收到的数据放入到发送寄存器
        while(!TI);           //等待发送数据完成
        TI=0;                          //清除发送完成标志位
    }
```

将程序下载到开发板观察结果，实验结果是你发送到单片机的数据会重新从单片机接收到。

打开串口调试助手

通过设备管理器查看串口号



设置好参数后，打开串口

我们在输入框中写入“普中科技开发板”，择在接收区会显示“普中科技开发板”。单片机程序的作用就是接收到什么内容，就返回什么内容。

第十六讲 温度传感器 18B20

1. 单总线概述

在介绍 DS18B20 之前必须要先介绍 1-Wire 即单总线通信。1-Wire 总线技术是美国 Maxim 全资子公司 Dallas 半导体公司近年推出的新技术。它将地址线、数据线、控制线合为 1 根信号线既传输时钟又传输数据而且数据传输是双向的。允许在这根信号线上挂接多个 1-Wire 总线器件。1-Wire 总线技术具有节省 I/O 资源、结构简单、成本低廉、有广阔的应用空间、便于总线扩展和维护等优点，因此，在分布式测控系统中有着广泛应用。

1-wire 单总线适用于单个主机系统能够控制一个或多个从机设备。当只有一个从机位于总线上时，系统可按照单节点系统操作。而当多个从机位于总线上时，则系统按照多节点系统操作。所有的 1- W ire 总线器件都具有一个共同的特征：在出厂时，每个器件都有一个与其它任何器件互不重复的固定的序列号，通过它自己的序列号可以区分同一总线上的多个器件。

单总线只有一根数据线。设备主机或从机通过一个漏极开路或三态端口，连接至该数据线，这样允许设备在不发送数据时释放数据总线，以便总线被其它设备所使用。单总线端口为漏极开路。单总线要求外接一个约5k 的上拉电阻，这样，单总线的闲置状态为高电平。不管什么原因，如果传输过程需要暂时挂起，且要求传输过程还能够继续的话，则总线必须处于空闲状。态位传输之间的恢复时间没有限制，只要总线在恢复期间处于空闲状态（高电平）。如果总线保持低电平超过480 μs 总线上的所有器件将复位。另外，在寄生方式供电时，为了保证单总线器件在某些工作状态下（如温度转换期间、EPROM 写入等）具有足够的电源电流，必须在总线上提供强上拉。

2. 单总线器件—温度传感器 DS18B20

温度是一种最基本的环境参数，日常生活和工农业生产中经常要检测温度。传统的方式是采用热电偶或热电阻，但是由于模拟温度传感器输出为模拟信号，必须经过 A/D 转换环节获得数字信号后才能与单片机等微处理器接口，使得硬件电路结构复杂，制作成本较高。近年来，美国 DALLAS 公司生产的 DS18B20 为代表的新型单总线数字式温度传感器以其突出优点广泛使用于仓储管理、工农

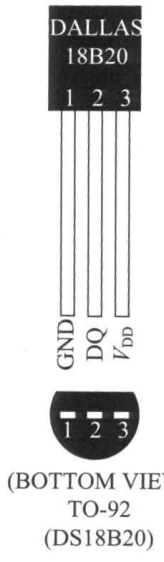
业生产制造、气象观测、科学研究以及日常生活中。DS18B20 集温度测量和 A/D 转换于一体，直接输出数字量，传输距离远，可以很方便地实现多点测量，硬件电路结构简单，与单片机接口几乎不需要外围元件。

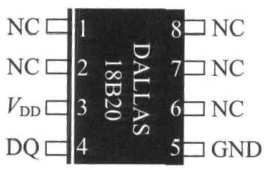
在许多工业场合中都要进行温度检测和温度控制,常用方法是采用温度传感进行检测,配合单片机进行控制。DS18B20 是美国 DALLAS 公司生产的“一线总线”接口的数字化传感器,他具有微型化、低功耗、抗干扰能力强、易与微处理器接口等优点,可直接将温度转化成串行数字信号供微处理器接收处理。利用这种温度传感器构成的温度测量系统电路非常简单、易于实现,并且适用于几乎所有类型的单片机。

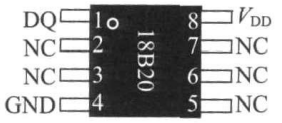
S1820 温度传感器是一种单总线型温度测量器件,具有直接的数字信号,可采用总线供电,在同一根总线上可接多个传感器,构成多点测温网络,是温度场监控系统的理想选择。

美国 DALLAS 半导体公司的 DS18B20 是世界上第一片支持“单总线”接口的数字式温度传感器，能够直接读取被测物的温度值。

如右图 DS18B20 采用 3 脚 TO-92 封装或 8 脚的 SOIC 封装，可以适应不同的环境需求。各引脚的功能：GND 为电压地；DQ 为单数据总线；VDD 为电源电压;NC 为空引脚。







其测量范围在-55～+125℃、-10℃~+85℃之内的测量精度可达± 0.5℃，稳定度为 1%。通过编程可实现 9、10、11、12 位的分辨率读出温度数据，以上都包括一个符号位，因此对应的温度量化值分别为 0.5℃、0.25℃、0.125℃、0.062 5℃，芯片出厂时默认为 12 位的转换精度。读取或写入 DS18B20 仅需要一根总线，要求外接一个约 4. 7k Ω的上拉电阻，当总线闲置时，其状态为高电平。支持多点组网功能，多个 DS18B20 可以并联在唯一的三线上，实现多点测温。此外 DS18B20 是温度- 电流传感器,对于提高系统抗干扰能力有很大的帮助。

DS18B20 工作原理及应用

DS18B20 的温度检测与数字数据输出全集成于一个芯片之上，从而抗干扰力更强。其一个工作周期可分为两个部分，即温度检测和数据处理。在讲解其工作流程之前我们有必要了解 18B20 的内部存储器资源。18B20 共有三种形态的存储器资源，它们分别是：

（1）ROM 只读存储器，用于存放 DS18B20ID 编码，其前 8 位是单线系列编码（DS18B20 的编码是 19H），后面 48 位是芯片唯一的序列号，最后 8 位是以上 56 位的 CRC 码（冗余校验）。数据在出厂时设置不由用户更改。DS18B20 共 64 位 ROM。

（2）RAM 数据暂存器，用于内部计算和数据存取，数据在掉电后丢失，DS18B20 共 9 个字节 RAM，每个字节为 8 位。如表 1-2 所示。第 1、2 个字节是温度转换后的数据值信息，第 3 和第 4 字节是高温触发器 TH 和低温触发器 TL 的易失性拷贝，第 5 个字节为配置寄存器，它的内容用于确定温度值的数字转换分辨率，DS18B20 工作时寄存器中的分辨率转换为相应精度的温度数值。以上字节内容每次上电复位时被刷新。配置寄存器字节各位的定义如表 1-3 所示。低 5 位一直为 1，TM 是工作模式位，用于设置 DS18B20 在工作模式还是在测试模式，DS18B20 出厂时该位被设置为 0，用户不要去改动；R1 和 R0 用来设置分辨率，决定温度转换的精度位数。如下表 所示。

18B20 的 RAM 各字节定义

温度 LSB	温度 MSB	TH 用户字节 1	TL 用户字节 2	配置寄存器	保留	保留	保留	CRC

配置寄存器中的各位定义

TM	R_1	R_0	1	1	1	1	1

为了保证数据可靠地传输，任一时刻 1-Wire 总线上只能有一个控制信号或数据。进行数据通信时应符合 1-Wire 总线协议，访问 DS18B20 的操作顺序遵循以下三步：（详细过程见单总线）

第一步：初始化

第二步：R O M 命令

第三步：DS18B20 功能命令

初始化

基于 1-Wire 总线上的所有传输过程都是以初始化开始的，主机发出复位脉冲，从机响应应答脉冲。应答脉冲使主机知道，总线上有从机设备，且准备就绪。

控制器发送 ROM 指令 DS18B20 及功能命令

双方打完招呼之后就要进行交流，ROM 指令共有 5 条，每一个工作周期只能发一条，ROM 指令分别是读 ROM 数据（33H）、指定匹配芯片（55H）、跳跃 ROM（CCH）、芯片搜索（F0H）、报警芯片搜索（ECH）。ROM 指令为 8 位长度，功能是对片内的 64 位光刻 ROM 进行操作。其主要目的是为了分辨一条总线上挂接的多个器件并作处理。诚然，单总线上可以同时挂接多个器件，并通过每个器件上所独有的 ID 号来区别，一般只挂接单个 DS18B20 芯片时可以跳过 ROM 指令（注意：此处指的跳过 ROM 指令并非不发送 ROM 指令，而是用特有的一条“跳过指令”）。

在 ROM 指令发送给 DS18B20 之后，紧接着（不间断）就是发送存储器操作指令了。操作指令同样为 8 位，共 6 条，存储器操作指令分别是写 RAM 数据（4EH）、读 RAM 数据（BEH）、将 RAM 数据复制到 EEPROM（48H）、温度转换（44H）、将 EEPROM 中的报警值复制到 RAM（B8H）、工作方式切换（B4H）。存储器操作指令的功能是命令 DS18B20 做什么样的工作，是芯片控制的关键。

每次访问单总线器件，必须严格遵守这个命令序列，如果出现序列混乱，则单总线器件不会响应主机。但是，这个准则对于搜索ROM 命令和报警搜索命令例外，在执行两者中任何一条命令之后，主机不能执行其后的功能命令，必须返回至第一步。

R O M 命令

在主机检测到应答脉冲后，就可以发出 ROM 命令。这些命令与各个从机设备的唯一 64 位 ROM 代码相关，允许主机在 1-Wire 总线上连接多个从机设备时，指定操作某个从机设备。这些命令还允许主机能够检测到总线上有多少个从机设备以及其设备类型，或者有没有设备处于报警状态。共有 5 种 ROM 命令，它们分别是：读 ROM、搜索 R O M 、匹配 R O M 、跳过 R O M 、报警搜索。对于只有一个温度传感器的单点系统， 跳过 ROM(SKIP ROM)命令特别有用，控制器不必发送 64 比特序列号，从而节约了大量时间。对于 1-Wire 总线的多点系统，通常先把每一个单总线器件的 64 比特序列号测出，要访问某一个从属节点时，发送

匹配 ROM 命令(MATCH ROM)，然后发送 64 比特序列号，这时可以对指定的从属节点进行操作。

3. 搜索ROM[F0h]

当系统初始上电时，主机必须找出总线上所有从机设备的ROM 代码，这样主机就能够判断出从机的数目和类型。主机通过重复执行搜索ROM 循环（搜索ROM 命令跟随着位数据交换），以找出总线上所有的从机设备。如果总线只有一个从机设备，则可以采用读ROM命令来替代搜索ROM 命令。在每次执行完搜索ROM 循环后，主机必须返回至命令序列的第一步（初始化）。

4. 读ROM[33h] （适合于单节点）

该命令仅适用于总线上只有一个从机设备.允许主机直接读出从机的64 位ROM 代码,无须执行搜索ROM 过程。如果该命令用于多节点系，统则必然发生数据冲突，因为每个从机设备都会响应该命令。

5. 匹配ROM[55h]

匹配ROM 命令跟随64 位ROM 代码，从而允许主机访问多节点系统中某个指定的从机设备。仅当从机完全匹配64 位ROM 代码时，才会响应主机随后发出的功能命令，其它设备将处于等待复位脉冲状态。

6. 跳越ROM[CCh] （仅适合于单节点以18B20为例）

主机能够采用该命令同时访问总线上的所有从机设备， 而无须发出任何ROM 代码信息。例如，主机通过在发出跳越ROM 命令后跟随转换温度命令[44h] ，就可以同时命令总线上所有的DS18B20 开始转换温度，这样大大节省了主机的时间。值得注意，如果跳越ROM命令跟随的是读暂存器[BEh]的命令（包括其它读操作命令），则该命令只能应用于单节点系统，否则将由于多个节点都响应该命令而引起数据冲突。

7. 报警搜索[ECh] （仅少数1-wire 器件支持）

除那些设置了报警标志的从机响应外，该命令的工作方式完全等同于搜索ROM 命令。该命令允许主机设备判断那些从机设备发生了报警（如最近的测量温度过高或过低等）。同搜索ROM 命令一样，在完成报警搜索循环后，主机必须返回至命令序列的第一步。

8. 功能命令（以 DS18B20 为例）

在主机发出ROM 命令，以访问某个指定的DS18B20， 接着就可以发出DS18B20 支持
的某个功能命令。这些命令允许主机写入或读出DS18B20 暂存器、启动温度转换以及判断
从机的供电方式。DS18B20 的功能命令总结于下表

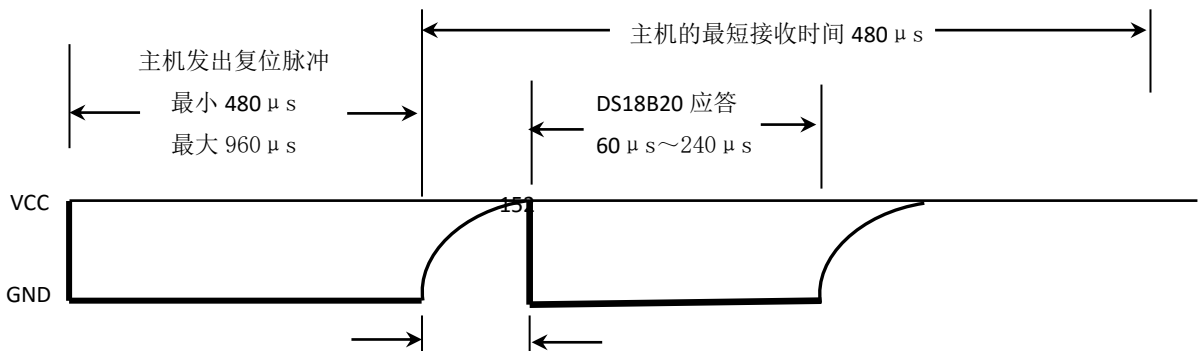
DS18B20 的命令

命令	描述	命令代码	发送命令后，单总线上的响应信息
温度转换命令			
转换温度	启动温度转换	44H	无
存储器命令			
读暂存器	读全部的暂存器内容，包括 CRC 字节	BEH	DS18B20 传输多达 9 个字节至主机
写暂存器	写暂存器第 2、3 和 4 字节的数据（即 TH，TL 和配置寄存器）	4EH	主机传输 3 个字节数据至 18B20
复制暂存器	将暂存器中的 TH，TL 和配置字节复制到 EPPROM 中	48H	无
回读 EEPROM	将 TH，TL 和配置字节从 EPPROM 读回至暂存器中	B8H	18B20

9. 控制器对 DS18B20 操作流程

首先我们必须对 DS18B20 芯片进行复位，复位就是由控制器（单片机）给 DS18B20 单总线至少 480 μs 的低电平信号。当 DS18B20 接到此复位信号后则会在 15 μs~60 μs 后回发一个芯片的存在脉冲。

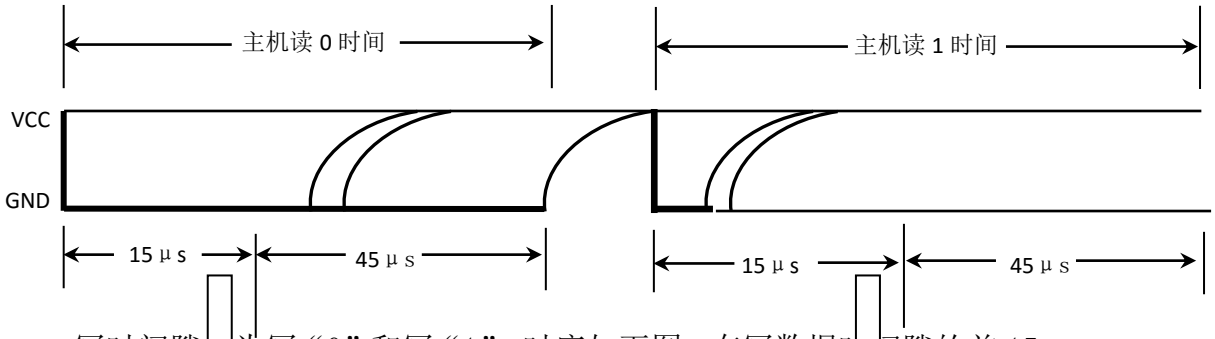
在复位电平结束之后，控制器应该将数据单总线拉高，以便于在 15 μs~60 μs 后接收存在脉冲，存在脉冲为一个 60 μs~240 μs 的低电平信号。至此，通信双方已经达成了基本的协议，接下来将会是控制器与 DS18B20 间的数据通信。如果复位低电平的时间不足或是单总线的电路断路都不会接到存在脉冲，在设计时要注意意外情况的处理。如下图所示。



DS18B20 数据的读写

一个存储器操作指令结束后则将进行指令执行或数据的读写，这个操作要视存储器操作指令而定。如执行温度转换指令则控制器（单片机）必须等待 DS18B20 执行其指令，一般转换时间为 500 μs。如执行数据读写指令则需要严格遵循 DS18B20 的读写时序来操作。

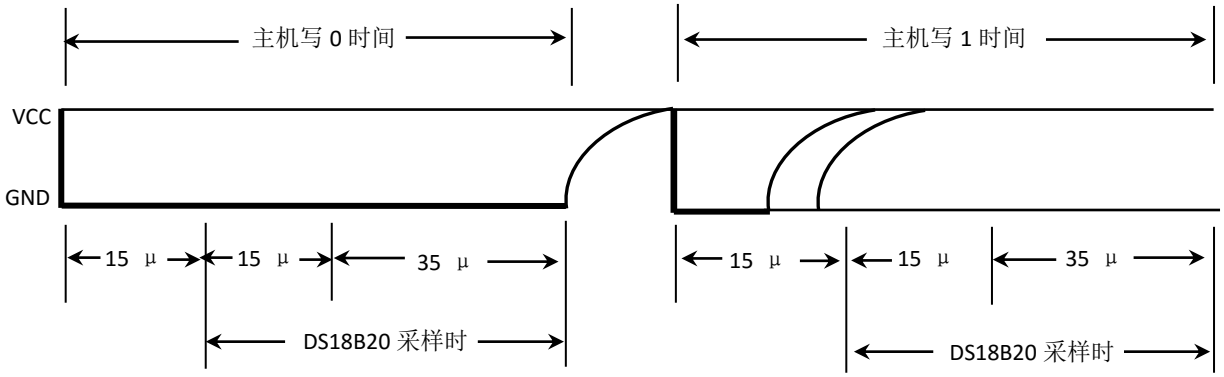
读时间隙控制时的采样时间应该更加精确才行，读时间隙也是必须先由主机产生至少 1 μs 的低电平，表示读时间的起始。随后在总线被释放后的 15 μs 中 DS18B20 会发送内部数据位，这时控制器如果发现总线为高电平表示读出“1”，如果总线为低电平则表示读出数据“0”。每一位的读取之前都由控制器加一个起始信号。注意：如下图所示，必须在读间隙开始的 15 μs 内读取数据位才可以保证通信的正确。



DS18B20 的读 0 读 1 时序

写时间隙：为写“0”和写“1”，时序如下图。在写数据时间隙的前 15 μs 总线需要是被控制器拉置低电平，而后则将是芯片对总线数据的采样时间，采样时间在 15 μs~60 μs，采样时间内如果控制器将总线拉高则表示写“1”，如果控制器将总线拉低则表示写“0”。每一位的发送都应该有一个至少 15 μs 的低电平起始位，随后的数据“0”或“1”应该在 45 μs 内完成。整个位的

发送时间应该保持在 60 μ s~120 μ s，否则不能保证通信的正常。



DS18B20 的写 0 写 1 时序

若要读出当前的温度数据需要执行两次工作周期，第一个周期为复位、跳过 ROM 指令、执行温度转换存储器操作指令、等待 500 μ s 温度转换时间。紧接着执行第二个周期为复位、跳过 ROM 指令、执行读 RAM 的存储器操作指令、读数据（最多为 9 个字节，中途可停止，只读简单温度值则读前 2 个字节即可）。其他的操作流程也大同小异，在此不多介绍。

温度计算的简化处理

当 DS18B20 接收到温度转换命令后，开始启动转换。转换完成后的温度值就以 16 位带符号扩展的二进制补码形式（高 5 位是符号位）存储在高速暂存器的第 1、2 字节。单片机可以通过单线接口读出该数据，读数据时低位在先，高位在后，数据格式以 0.062 5℃/LSB 形式表示。当符号位 *S*=0 时，表示测得的温度值为正值，可以直接将二进制位转换为十进制；当符号位 *S*=1 时，表示测得的温度值为负值，要先将补码变成原码（取反加 1），再计算十进制数值。例如，输出数字量 7D0H，可知温度为正，07D0H 转换成十进制是 2000，所测温度为 2000×0.0625=125℃；输出数字量 0FC90H，可知温度为负，0FC90H 原码为 370H，转换成十进制是 880，所测温度为 880×0.0625=-55℃。若温度显示范围为 0℃~99℃，显示精度为 1℃显示时，可以只用 2 个数码管。由于 12 位转化时每位的精度为 0.062 5℃，可以把转换得到的温度最高四位与最低四位舍去，这样可获得一个新的字节，这个字节就是实际测量的温度值，且无需乘以 0.062 5，简化了计算过程。

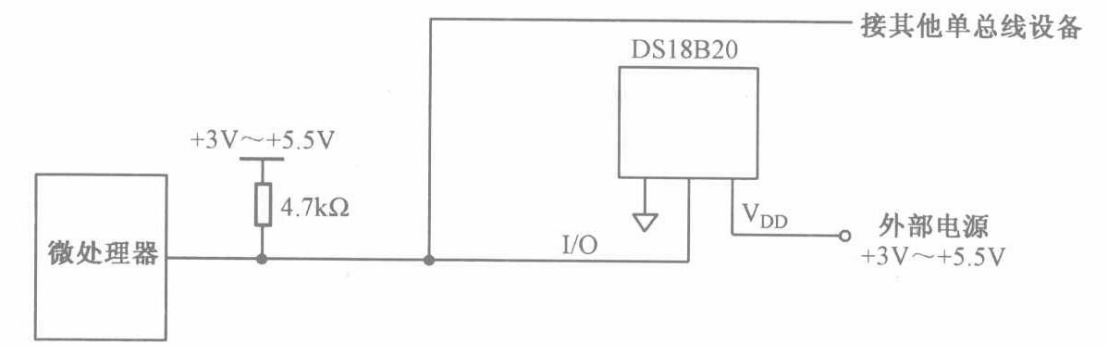
DS18B20 设计中应注意的几个问题

DS18B20 具有测温系统简单、测温精度高、连接方便、占用接口线少等优点，但在实际应用中也应注意以下几方面的问题：

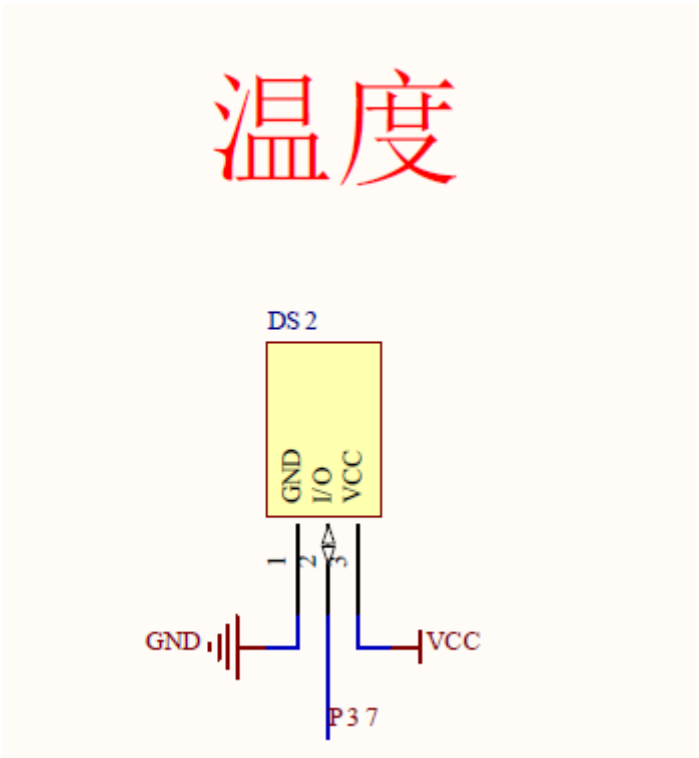
较小的硬件开销需要相对复杂的软件进行补偿，由于DS18B20与微处理器间采用串行数据传送，因此，在对DS18B20 进行读写编程时，必须严格的保证读写时序，否则将无法读取测温结果。

在DS18B20的有关资料中均未提及1-Wire上所挂DS18B20 数量问题，容易使人误认为可以挂任意多个DS18B20，在实际应用中并非如此。当1-Wire 上所挂DS18B20 超过8 个时，就需要考虑微处理器的总线驱动问题，这一点在进行多点测温系统设计时要加以注意。 连接DS18B20 的总线电缆是有长度限制的。试验中，当采用普通信号电缆传输长度超过50 米时，读取的测温数据将发生错误。当将总线电缆改为双绞线带屏蔽电缆时，正常通信距离可达150米。这主要是由于总线分布电容使信号波形产生畸变造成的。因此，在用DS18B20进行长距离测温系统设计时要充分考虑总线分布电容和阻抗匹配问题。实际应用中， 测温电缆线建议采用屏蔽4芯双绞线，其中一对线接地线与信号线，另一组接VCC和地线，屏蔽层在源端单点接地。

18B20 的典型电路连接图



开发板上的电路



打开 11.温度显示(18b20) 数码管显示温度

名称	修改日期	类型	大小
lcd.__i	2014/6/10 8:41	__I 文件	1 KB
lcd	2014/6/10 8:41	c_file	4 KB
lcd.h	2014/6/10 8:41	H 文件	1 KB
lcd.LST	2014/6/10 8:41	LST 文件	7 KB
lcd.OBJ	2014/6/10 8:41	OBJ 文件	4 KB
main.__i	2014/6/10 8:41	__I 文件	1 KB
main	2014/6/10 8:41	c_file	4 KB
main.LST	2014/6/10 8:41	LST 文件	8 KB
main.OBJ	2014/6/10 8:41	OBJ 文件	6 KB
pro	2014/6/10 8:41	文件	12 KB
pro.hex	2014/6/10 8:41	HEX 文件	6 KB
pro.lnp	2014/6/10 8:41	LNP 文件	1 KB
pro.M51	2014/6/10 8:41	M51 文件	16 KB
pro.plg	2014/6/10 8:41	PLG 文件	1 KB
pro.uvgui.Administrator	2014/6/10 8:41	ADMINISTRATO...	67 KB
pro.uvgui_Administrator.bak	2014/6/10 8:41	BAK 文件	67 KB
pro.uvopt	2014/6/10 8:41	UVOPT 文件	8 KB
pro	2014/6/10 8:41	礦ision4 Project	14 KB
pro_uvopt.bak	2014/6/10 8:41	BAK 文件	10 KB
pro_uvproj.bak	2014/6/10 8:41	BAK 文件	14 KB
STARTUP.A51	2014/6/10 8:41	A51 文件	7 KB
STARTUP.LST	2014/6/10 8:41	LST 文件	14 KB
STARTUP.OBJ	2014/6/10 8:41	OBJ 文件	1 KB
temp	2014/6/10 8:41	c_file	4 KB
temp.h	2014/6/10 8:41	H 文件	1 KB

详细的源程序我们可以打开上述目录查看。(程序内已经注释的比较详细)，下载里面的 HEX 文件即可观察现象。

第十七讲 EEPROM 操作 24C02

1. IIC 总线介绍

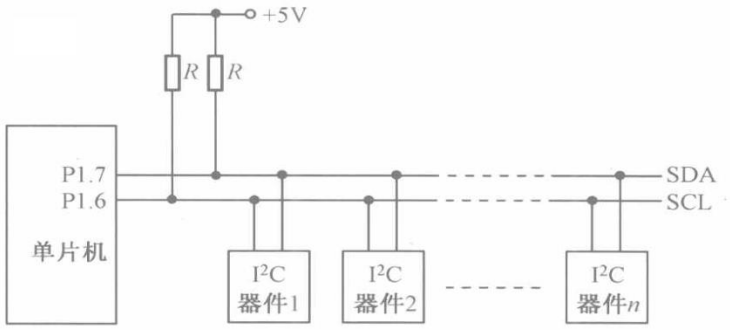
IIC 总线(Inter IC Bus)由 PHILIPS 公司推出，是近年来微电子通信控制领域广泛采用的一种新型总线标准,它是同步通信的一种特殊形式,具有接口线少、控制简单 ,器件封装形式小、通信速率较高等优点.在主从通信中，可以有多个 IIC 总线器件同时接到 IIC 总线上，所有与 IIC 兼容的器件都具有标准的接口，通过地址来识别通信对象，使它们可以经由 IIC 总线互相直接通信。

IIC 总线产生于在80 年代，最初为音频和视频设备开发，如今主要在服务器管理中使用，其中包括单个组件状态的通信。例如管理员可对各个组件进行查询，以管理系统的配置或掌握组件的功能状态，如电源和系统风扇。可随时监控内存、硬盘、网络、系统温度等多个参数，增加了系统的安全性，方便了管理。

II 总线由数据线 SDA 和时钟线 SCL 两条线构成通信线路，既可发送数据，也可接收数据.在 CPU 与被控 IC 之间、IC 与 IC 之间都可进行双向传送，最高传送速率为 400kbps。各种被控器件均并联在总线上,但每个器件都有唯一的地址.在信息传输过程中，IIC 总线上并联的每一个器件既是被控器(或主控器)，又是发送器(或接收器)，这取决于它所要完成的功能。CPU 发出的控制信号分为地址码和数据码两部分:地址码用来选址，即接通需要控制的电路：数据码是通信的内容。这样各 IC 控制电路虽然挂在同一条总线上，却彼此独立。

2. IIC 总线硬件结构图

下图为 IIC 总线系统的硬件结构图，其中，SCL 是时钟线，SDA 是数据线。总线上各器件都采用漏极开路结构与总线相连.因此 SCL 和 SDA 均需接上拉电阻，总线在空闲状态下均保持高电平.连到总线上的任一器件输出的低电平，都将使总线的信号变低，即各器件的 SDA 及 SCL 都是线“与”关系。



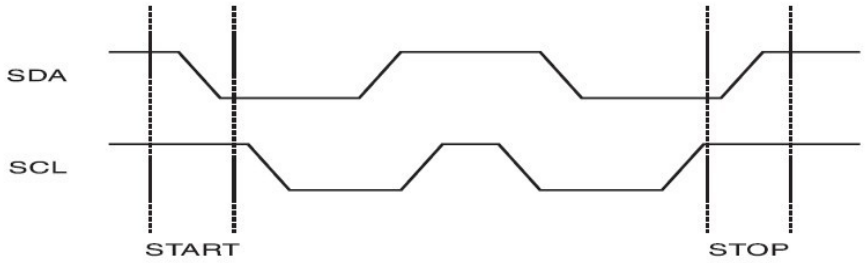
I²C 总线系统硬件结构图

IIC 总线支持多主和主从两种工作方式，通常为主从工作方式.在主从工作方式中，系统中只有一个主器件(单片机)，其他器件都是具有 IIC 总线的外围从器件。在主从工作方式中，主器件启动数据的发送(发出启动信号)，产生时钟信号，发出停止信号。

3. IIC 总线操作方法：

IIC 总线在传送数据过程中共有三种类型信号， 它们分别是：开始信号、结束信号和应答信号。笔者分别配合时序图来说明：

1） 开始信号与结束信号：



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。

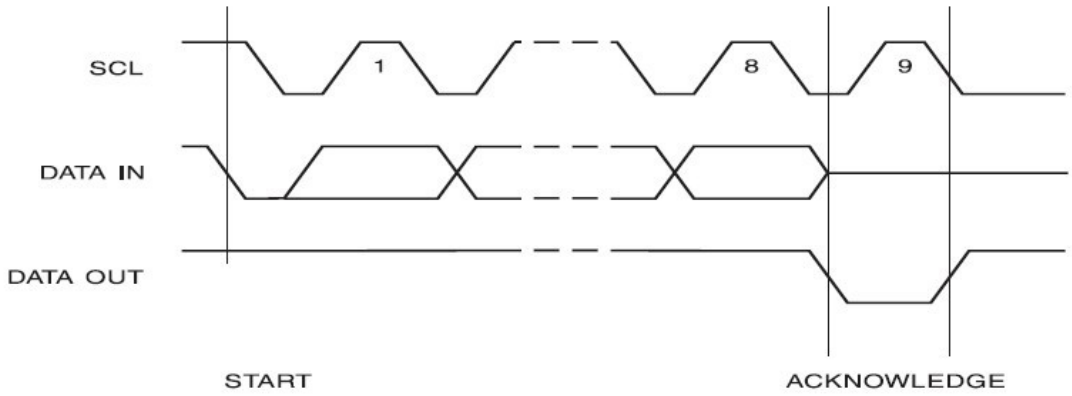
结束信号：SCL为低电平时，SDA由低电平向高电平跳变，结束传送数据。

一定要注意时序的精准性！这两个信号成功的关键是：

① 在SDA的整个变化期间，SCL是一直保持稳定的高电平的。SCL监视SDA的整个变化而不只是开始信号中的负跳变或者结束信号的正跳变。

② 要注意到IIC总线的传输速度为Kbps级，而AVR单片机工作频率为Mbps级，所以必不可少的要用us级延时函数_delay_us()将单片机速度降下来配合IIC时序。

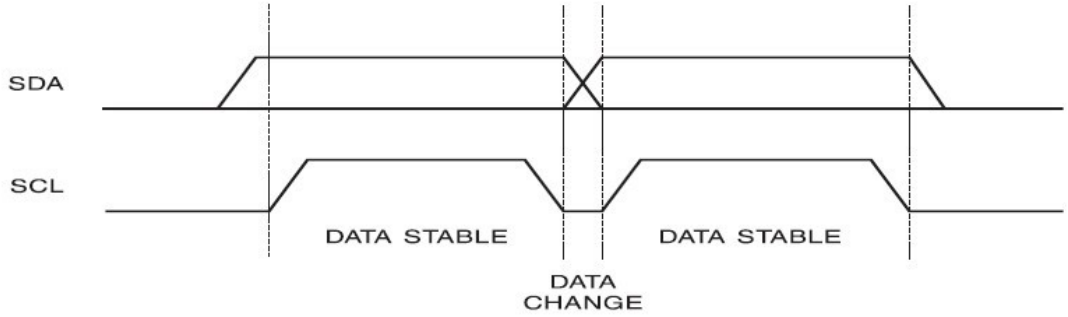
2） 应答信号



首先要知道器件发送数据到总线上，则定义为发送器，器件接收数据则定义为接收器。如上应答信号时序，SCL线为主机（微控制器）控制，DATA IN表示从机接受数据情况，DATA OUT 表示从机发送数据情况。如图可见，在START信号之后，从器件在8个时钟脉冲的控制下接受数据，在此期间从机并未发出任何信号！而在第9个时钟期间，从机将SDA线拉低表示应答。

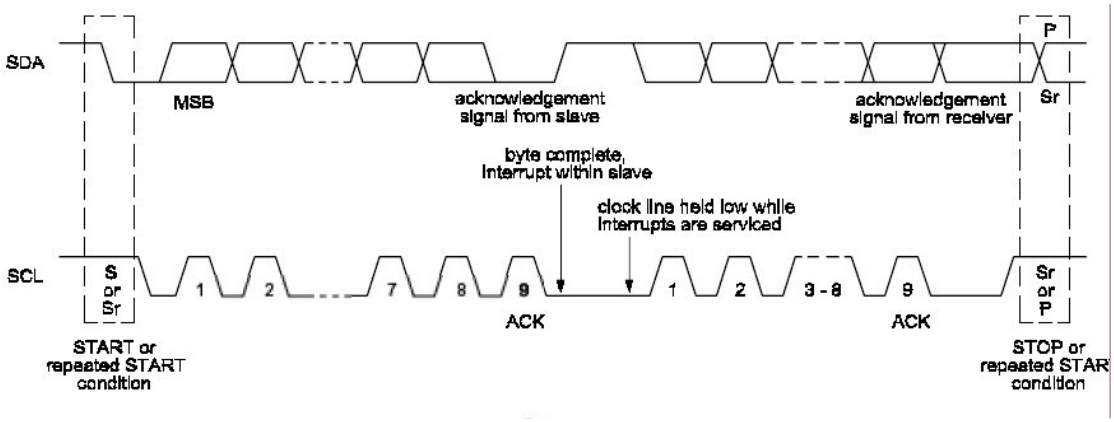
所以，应答信号：接收数据的IC在接收到8bit数据后，向发送数据的IC发出特定的低电平脉冲，表示已收到数据。

关于SDA上数据的变化



如上图所见，当SCL为高电平时要求DATA STABLE 即数据稳定，而在SCL低电平时才可以DATA CHANGE即数据改变。所以IIC总线协议要求只有在SCL为低电平是SDA上的数据才可以改变。因为若SCL处于高电平期间，SDA上的电平无论发生上升沿还是下降沿，都会被识别为一次开始或结束信号。

3） 字节传输时序



上时序是一个个完整的IIC字节写周期，有左往右，起始信号之后：

如果是写字节，前8个时钟周期中的SCL上升沿将数据写入数据接收方，接着在第九个周期数据接收方发出应答信号；

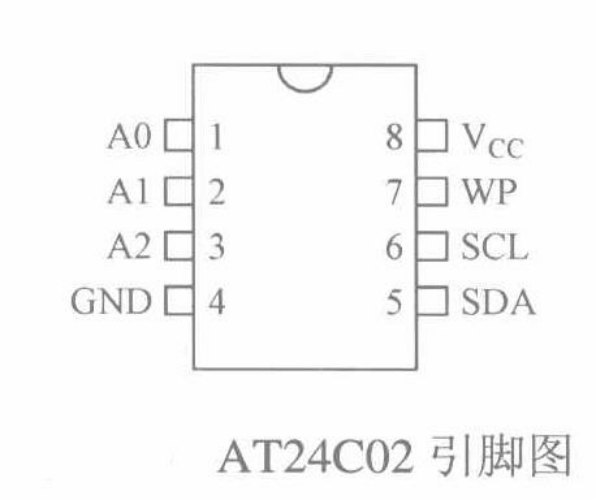
而如果是读字节，前8个时钟周期中的SCL上升沿将数据从数据发送方读出到SDA线上，并且数据接收方在第九个周期选择发送怎样的应答信息到SDA线上；

具有IIC总线接口的EEPROM有多个厂家的多种类型产品。在此仅介绍ATMEL公司生产的AT24C系列EEPROM，主要型号有AT24C01/02/04/08/16等，其对应的存储容量分别为128x8/256x8/512x8/ 1024x8/2048x8。采用这类芯片可解决掉电数据保存问题，可对所存数据保存100年，并可多次擦写，擦写次数可达10万次以上。

在一些应用系统设计中.有时需要对工作数据进行掉电保护，如电子式电能表等智能化产品。若采用普通存储器，在掉电时需要备用电池供电，并需要在硬件上增加掉电检测电路，但存在电池不可靠及扩展存储芯片占用单片机过多口线的缺点。采用具有IIC总线接口的串行EEPROM器件可很好地解决掉电数据保存问题，且硬件电路简单。下面以AT24C02芯片为例，介绍具有IIC总线接口的EEPROM的具体应用。

4. AT24C02引脚配置与引脚功能

AT24C02芯片的常用封装形式有直插(DIP8)式和贴片(SO-8)式两种，无论是直插式还是贴片式.其引脚功能与序号都一样。



AT24C02 引脚图

各引脚功能如下:

1) 1，2，3 (AO，A1、A2)---可编程地址输入端.

2) 4 (GND)---电源地.

3) 5 (SDA)---串行数据输入了输出端。

4) 6 (SCL)---串行时钟输入端。

5) 7 (WP) ---写保护输入端，用于硬件数据保护。当其为低电平时，可以对整个存储器进行正常的读/写操作:当其为高电平时，存储器具有写保护功能.但读操作不受影响。

6) 8 (Vcc) ---电源正端.

5. 存储结构与寻址

AT24CO2的存储容量为2KB，内部分成32页，每页8B，共256B,操作时有两种寻址方式:芯片寻址和片内子地址寻址。

(1)芯片寻址。AT24CO2的芯片地址为1010,其地址控制字格式为1010A2A1AOR/W.其中A2，A1，AO为可编程地址选择位。A2，A1，AO引脚接高、低电平后得到确定的三位编码，与1010形成7位编码，即为该器件的地址码。R/w为芯片读写控制位，该位为0，表示对芯片进行写操作;该位为1,表示对芯片进行读操作。

(2)片内子地址寻址.芯片寻址可对内部256B中的任一个进行读/写操作，其寻址范围为00-FF。共256个寻址单元。

6. 读/写操作时序

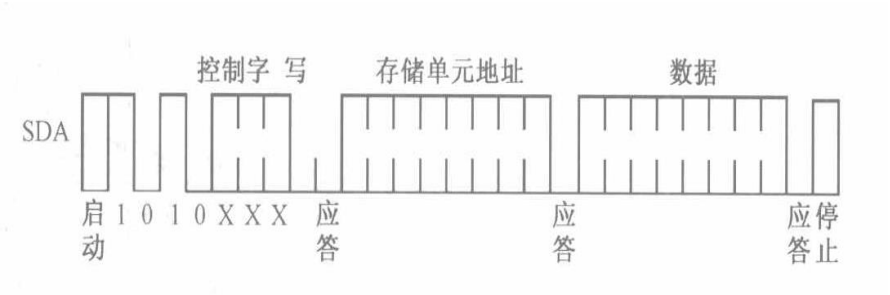
串行EEPROM一般有两种写入方式:一种是字节写入方式，另一种是页写入方

式。页写入方式允许在一个写周期内(10ms左右)对一个字节到一页的若干字节进行编程写入，

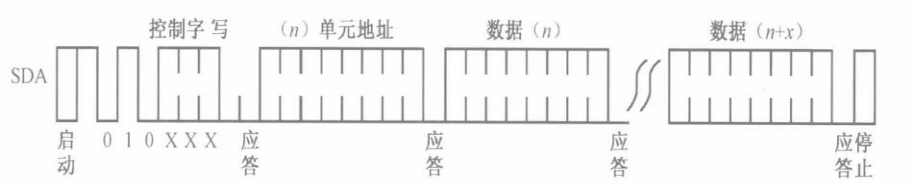
AT24C02的页面大小为8B。采用页写方式可提高写入效率，但也容易发生事故。AT24C系列片内地址在接收到姆一个数据字节后自动加1，故装载一页以内数据字节时，只需输入首地址，如果写到此页的最后一个字节，主器件继续发送数据，数据将重新从该页的首地址写入，进而造成原来的数据丢失，这就是页地址空间的“上卷”现象。

解决“上卷”的方法是:在第8个数据后将地址强制加1，或是将下一页的首地址重新赋给寄存器。

(1)字节写入方式.单片机在一次数据帧中只访问EEPROM一个单元.该方式下，单片机先发送启动信号，然后送一个字节的控制字，再送一个字节的存储器单元子地址，上述几个字节都得到EEPROM响应后，再发送8位数据.最后发送，位停止信号。发送格式如下图所示。

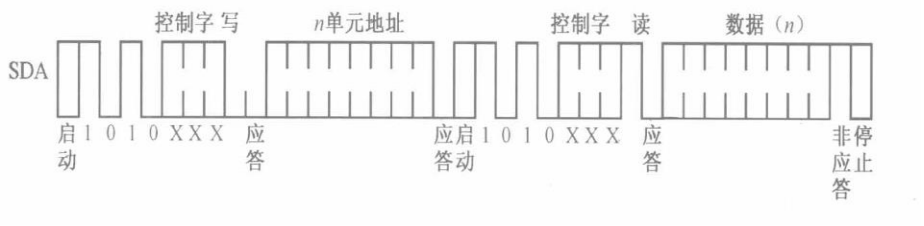


(2)页写入方式。单片机在一个数据写周期内可以连续访问1页(8个)EEPROM存储单元。在该方式中.单片机先发送启动信号，接着送一个字节的控制字.再送1个字节的存储器起始单元地址。上述几个字节都得到EEPROM应答后就可以发送最多1页的数据，并顺序存放在以指定起始地址开始的相继单元中，最后以停止信号结束。页写入帧格式如下图所示。

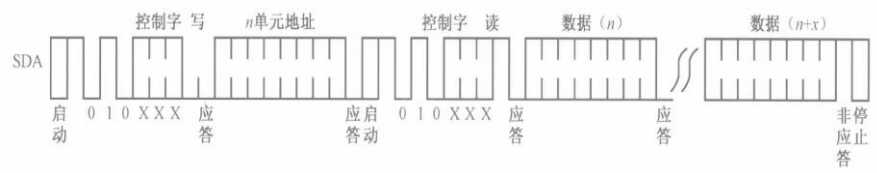


(3)指定地址读操作。读指定地址单元的数据。单片机在启动信号后先发送含有片选地址的写操作控制字.EEPROM应答后再发送1个(2KB以内的

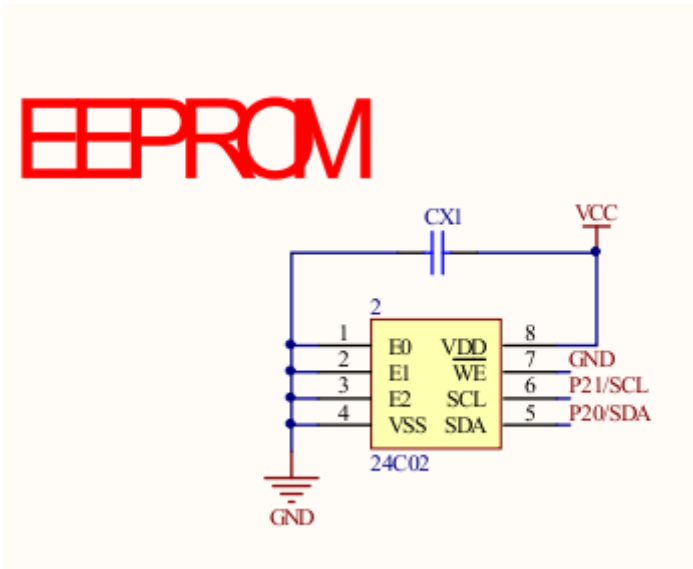
EEPROM)字节的指定单元的地址，EEPROM应答后再发送1个含有片选地址的读操作控制字，此时如果EEPROM做出应答。被访问单元的数据就会按SCL信号同步出现在串行数据/地址线SDA上。这种读操作的数据帧格式如下图所示。



(4)指定地址连续读。此种方式的读地址控制与前面指定地址读相同。单片机接收到每个字节数据后应做出应答，只要EEPROM检测到应答信号，其内部的地址寄存器就自动加I指向下一单元，并顺序将指向的单元的数据送到SDA串行数据线上。当需要结束读操作时，单片机接收到数据后在需要应答的时刻发送一个非应答信号，接着再发送一个停止信号即可。这种读操作的数据帧格式如下图所示。



首先看一下开发板电路图



看上图，

E0、E1、E2三个引脚为AT24C02的硬件地址线，根据引脚上的电平决定当前

器件的硬件地址。

WP为AT24C02的写保护引脚，当该引脚为高电平时，器件只读不写。

SCL、SDA分别为器件的IIC协议接口。

Main.c文件：

```
/*********************************************************************************
* 实验名                  : EEPROM 实验
* 使用的 IO      :
* 实验效果       : 按 K1 保存显示的数据，按 K2 读取上次保存的数据，按 K3 显示数据加一，
*按 K4 显示数据清零。
*    注意                           ：由于 P3.2 口跟红外线共用，所以做按键实验时为了不让红外线影响实验效果，最好把红外线先
*取下来。
*********************************************************************************/
#include<reg51.h>
#include"i2c.h"
//数码管 IO
#define DIG P0
sbit LSA=P2^2;
sbit LSB=P2^3;
sbit LSC=P2^4;
//按键 IO
sbit K1=P3^1;
sbit K2=P3^0;
sbit K3=P3^2;
sbit K4=P3^3;
void At24c02Write(unsigned char ,unsigned char );
unsigned char At24c02Read(unsigned char );
void Delay1ms();
void Timer0Configuration();
unsigned                                char                                code
DIG_CODE[10]={0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f};
unsigned char Num=0;
unsigned int disp[8]={0x3f,0x3f,0x3f,0x3f,0x3f,0x3f,0x3f,0x3f};
/*********************************************************************************
* 函数名          : main
* 函数功能          : 主函数
* 输入            : 无
* 输出            : 无
*********************************************************************************/

void main()
```

```
{
    unsigned int num0=0,num1=0,n;
    Timer0Configuration();
    while(1)
    {
        if(K1==0)
        {
            Delay1ms();
            if(K1==0)
                At24c02Write(2,num0);
            while((n<200)&&(K3==0))
            {
                n++;
                Delay1ms();
            }
            n=0;
            n=0;
        }

        if(K2==0)
        {
            Delay1ms();
            if(K2==0)
                num0=At24c02Read(2);
            while((n<200)&&(K3==0))
            {
                n++;
                Delay1ms();
            }
            n=0;
        }
        if(K3==0)
        {
            Delay1ms();
            if(K3==0)
                num0++;
            while((n<200)&&(K3==0))
            {
                n++;
                Delay1ms();
            }
            n=0;
            if(num0==256)
                num0=0;
```

```
        }

        if(K4==0)
        {
            Delay1ms();
            if(K4==0)
                num0=0;
            while((n<200)&&(K3==0))
            {
                n++;
                Delay1ms();
            }
            n=0;
        }
        disp[0]=DIG_CODE[num1/1000];//千位
        disp[1]=DIG_CODE[num1%1000/100];//百位
        disp[2]=DIG_CODE[num1%1000%100/10];//十位
        disp[3]=DIG_CODE[num1%1000%100%10];//个位
        disp[4]=DIG_CODE[num0/1000];//千位
        disp[5]=DIG_CODE[num0%1000/100];//百位
        disp[6]=DIG_CODE[num0%1000%100/10];//个位
        disp[7]=DIG_CODE[num0%1000%100%10];
    }
}
/*******************************************************************************
* 函数名         : Timer0Configuration()
* 函数功能         : 设置计时器
* 输入           : 无
* 输出           : 无
*******************************************************************************/

void Timer0Configuration()
{
    TMOD=0X02;//选择为定时器模式，工作方式 2，仅用 TRX 打开启动。

    TH0=0X9C;   //给定时器赋初值，定时 100us
    TL0=0X9C;
    ET0=1;//打开定时器 0 中断允许
    EA=1;//打开总中断
    TR0=1;//打开定时器
}
/*******************************************************************************
* 函数名         : Delay1ms()
* 函数功能         : 延时
```

```
* 输入           : 无
* 输出           : 无
*******************************************************************************/

void Delay1ms()   //误差 0us
{
    unsigned char a,b,c;
    for(c=1;c>0;c--)
        for(b=142;b>0;b--)
            for(a=2;a>0;a--);
}
/*******************************************************************************
* 函数名         : void At24c02Write(unsigned char addr,unsigned char dat)
* 函数功能         : 往 24c02 的一个地址写入一个数据
* 输入           : 无
* 输出           : 无
*******************************************************************************/

void At24c02Write(unsigned char addr,unsigned char dat)
{
    I2cStart();
    I2cSendByte(0xa0);
    I2cSendByte(addr);
    I2cSendByte(dat);
    I2cStop();
}
/*******************************************************************************
* 函数名         : unsigned char At24c02Read(unsigned char addr)
* 函数功能         : 读取 24c02 的一个地址的一个数据
* 输入           : 无
* 输出           : 无
*******************************************************************************/

unsigned char At24c02Read(unsigned char addr)
{
    unsigned char num;
    I2cStart();
    I2cSendByte(0xa0);
    I2cSendByte(addr);
    I2cStart();
    I2cSendByte(0xa1);
    num=I2cReadByte();
    I2cStop();
    return num;
```

```
}
/*******************************************************************************
* 函数名         : DigDisplay() interrupt 1
* 函数功能         : 中断数码管显示
* 输入           : 无
* 输出           : 无
*******************************************************************************/

void DigDisplay() interrupt 1
{
//定时器在工作方式二会自动重装初，所以不用在赋值。
//  TH0=0X9c;//给定时器赋初值，定时 1ms
//  TL0=0X00;
    DIG=0; //消隐
    switch(Num)  //位选，选择点亮的数码管，
    {
        case(7):
            LSA=0;LSB=0;LSC=0; break;
        case(6):
            LSA=1;LSB=0;LSC=0; break;
        case(5):
            LSA=0;LSB=1;LSC=0; break;
        case(4):
            LSA=1;LSB=1;LSC=0; break;
        case(3):
            LSA=0;LSB=0;LSC=1; break;
        case(2):
            LSA=1;LSB=0;LSC=1; break;
        case(1):
            LSA=0;LSB=1;LSC=1; break;
        case(0):
            LSA=1;LSB=1;LSC=1; break;
    }
    DIG=disp[Num]; //段选，选择显示的数字。
    Num++;
    if(Num>7)
        Num=0;
}
```

I2C.c 文件：

```
#include"i2c.h"
/*******************************************************************************
* 函数名         : Delay1us()
* 函数功能         : 延时
* 输入           : 无
```

```
* 输出           : 无
*******************************************************************************/

void Delay10us()
{
    unsigned char a,b;
    for(b=1;b>0;b--)
        for(a=2;a>0;a--);

}
/*******************************************************************************
* 函数名         : I2cStart()
* 函数功能         : 起始信号：在 SCL 时钟信号在高电平期间 SDA 信号产生一个下降沿
* 输入           : 无
* 输出           : 无
* 备注           : 起始之后 SDA 和 SCL 都为 0
*******************************************************************************/

void I2cStart()
{
    SDA=1;
    Delay10us();
    SCL=1;
    Delay10us();//建立时间是 SDA 保持时间>4.7us
    SDA=0;
    Delay10us();//保持时间是>4us
    SCL=0;
    Delay10us();
}
/*******************************************************************************
* 函数名         : I2cStop()
* 函数功能         : 终止信号：在 SCL 时钟信号高电平期间 SDA 信号产生一个上升沿
* 输入           : 无
* 输出           : 无
* 备注           : 结束之后保持 SDA 和 SCL 都为 1；表示总线空闲
*******************************************************************************/

void I2cStop()
{
    SDA=0;
    Delay10us();
    SCL=1;
    Delay10us();//建立时间大于 4.7us
    SDA=1;
```

```
    Delay10us();
}
/*****************************************************************************
* 函数名         : I2cSendByte(unsigned char num)
* 函数功能         : 通过 I2C 发送一个字节。在 SCL 时钟信号高电平期间，保持发送信
号 SDA 保持稳定
* 输入           : num
* 输出           : 0 或 1。发送成功返回 1，发送失败返回 0
* 备注           : 发送完一个字节 SCL=0
*****************************************************************************/

unsigned char I2cSendByte(unsigned char dat)
{
    unsigned char a=0,b=0;//最大 255，一个机器周期为 1us，最大延时 255us。

    for(a=0;a<8;a++)//要发送 8 位，从最高位开始
    {
        SDA=dat>>7;  //起始信号之后 SCL=0，所以可以直接改变 SDA 信号
        dat=dat<<1;
        Delay10us();
        SCL=1;
        Delay10us();//建立时间>4.7us
        SCL=0;
        Delay10us();//时间大于 4us
    }
    SDA=1;
    Delay10us();
    SCL=1;
    while(SDA)//等待应答，也就是等待从设备把 SDA 拉低
    {
        b++;
        if(b>200)    //如果超过 200us 没有应答发送失败，或者为非应答，表示接收结
束
        {
            SCL=0;
            Delay10us();
            return 0;
        }
    }
    SCL=0;
    Delay10us();
    return 1;
}
/*****************************************************************************
```

```
* 函数名          : I2cReadByte()
* 函数功能         : 使用 I2c 读取一个字节
* 输入             : 无
* 输出             : dat
* 备注             : 接收完一个字节 SCL=0
*******************************************************************************/

unsigned char I2cReadByte()
{
    unsigned char a=0,dat=0;
    SDA=1;          //起始和发送一个字节之后 SCL 都是 0
    Delay10us();
    for(a=0;a<8;a++)//接收 8 个字节
    {
        SCL=1;
        Delay10us();
        dat<<=1;
        dat|=SDA;
        Delay10us();
        SCL=0;
        Delay10us();
    }
    return dat;
}
```

将程序下载到开发板观察运行结果,实验效果是：按 K1 保存显示的数据，按 K2 读取上次保存的数据，按 K3 显示数据加一，按 K4 显示数据清零。

第十八讲 红外遥控显示

红外线遥控是目前使用最广泛的一种通信和遥控手段。由于红外线遥控装置具有体积小、功耗低、功能强、成本低等特点，因而，继彩电、录像机之后，在录音机、音响设备、空调机以及玩具等其它小型电器装置上也纷纷采用红外线遥控。工业设备中，在高压、辐射、有毒气体、粉尘等环境下，采用红外线遥控不仅完全可靠而且能有效地隔离电气干扰。

1 红外遥控系统

通用红外遥控系统由发射和接收两大部分组成。应用编/解码专用集成电路芯片来进行控制操作，如图1 所示。发射部分包括键盘矩阵、编码调制、LED 红外发送器；接收部分包括光、电转换放大器、解调、解码电路。

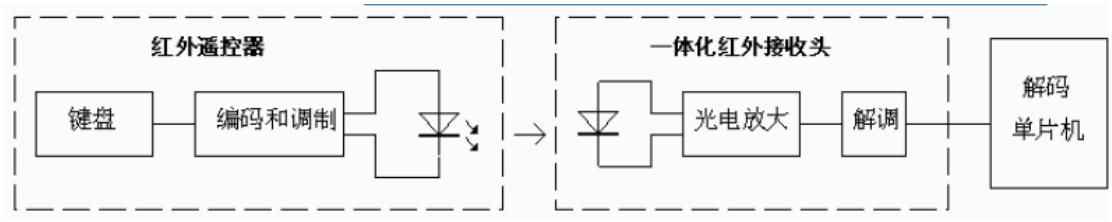


图1 红外线遥控系统框图

2 遥控发射器及其编码

遥控发射器专用芯片很多，根据编码格式可以分成两大类，这里我们以运用比较广泛，解码比较容易的一类来加以说明，现以日本NEC 的uPD6121G 组成发射电路为例说明编码原理（一般家庭用的DVD、VCD、音响都使用这种编码方式）。当发射器按键按下后，即有遥控码发出，所按的键不同遥控编码也不同。这种遥控码具有以下特征：采用脉宽调制的串行码，以脉宽为0.56ms、间隔0.565ms、周期为1.125ms 的组合表示二制的“0”；以脉宽为0.56ms、间隔1.685ms、周期为2.25ms 的组合表示二进制的“1”，其波形如图

2 所示。

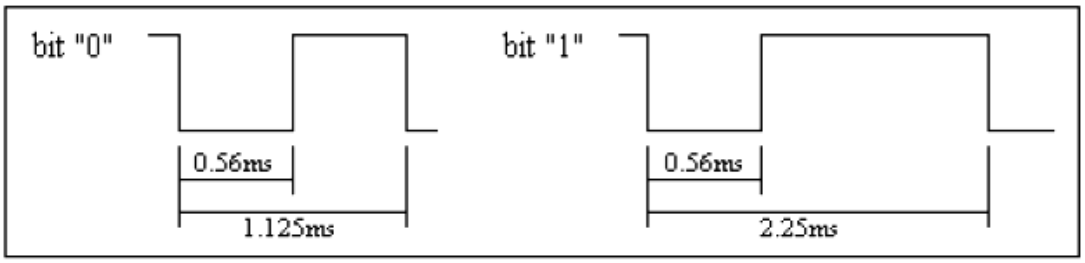


图2 遥控码的“0”和“1” （注：所有波形为接收端的与发射相反）

上述“0”和“1”组成的32 位二进制码经38kHz 的载频进行二次调制以提高发射效率，达到降低电源功耗的目的。然后再通过红外发射二极管产生红外线向空间发射，如图3 所示。

图3 遥控信号编码波形图

UPD6121G 产生的遥控编码是连续的32 位二进制码组，其中前16 位为用户识别码，能区别不同的电器设备，防止不同机种遥控码互相干扰。该芯片的用户识别码固定为十六进制01H；后16位为8 位操作码（功能码）及其反码。UPD6121G 最多额128 种不同组合的编码。

遥控器在按键按下后，周期性地发出同一种32 位二进制码，周期约为108ms。一组码本身的持续时间随它包含的二进制“0”和“1”的个数不同而不同，大约在45～63ms 之间，图4 为发射波形图。

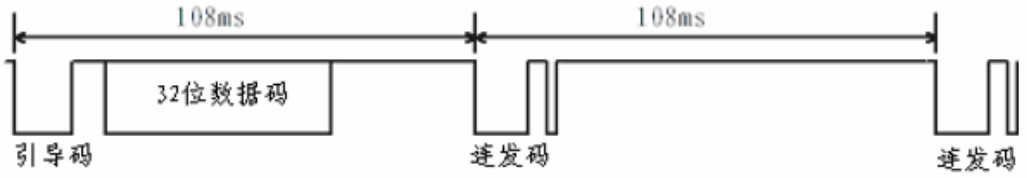


图4 遥控连发信号波形

当一个键按下超过36ms，振荡器使芯片激活，将发射一组108ms 的编码脉冲，这108ms 发射代码由一个引导码（9ms），一个结束码（4.5ms），低8 位地址码（9ms~18ms），高8 位地址码（9ms~18ms），8位数据码（9ms~18ms）和这8 位数据的反码（9ms~18ms）组成。如果键按下超过108ms 仍未松开，接下来发射的代码（连发码）将仅由起始码（9ms）和结束码（2.25ms）组成。

图5 引导码　　图6 连发码

3 遥控信号接收

接收电路可以使用一种集红外线接收和放大于一体的一体化红外线接收器，不需要任何外接元件，就能完成从红外线接收到输出与TTL 电平信号兼容的所有工作，而体积和普通的塑封三极管大小一样，它适合于各种红外线遥控和红外线数据传输。接收器对外只有3个引脚：Out、GND、Vcc 与单片机接口非常方便，如图7 所示。

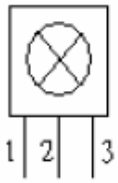


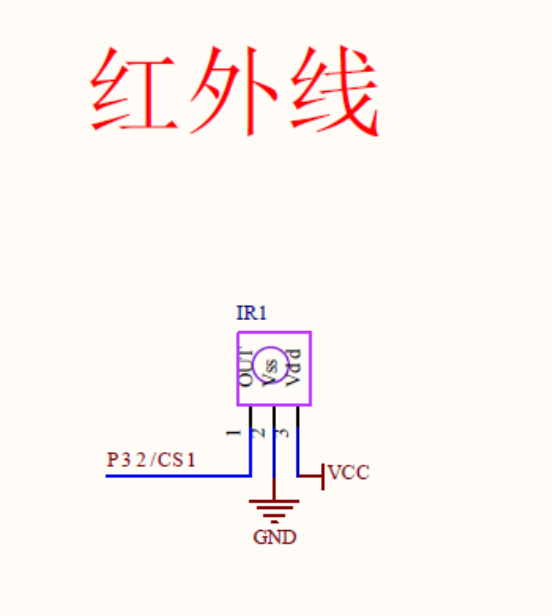
图 7

① 脉冲信号输出接，直接接单片机的IO 口。

② GND 接系统的地线（0V）；

③ Vcc 接系统的电源正极（+5V）；

开发板硬件连接图



开发板上的数据库连接到单片机中断引脚 P3.2，通过触发外部中断，进而使单片机对接收头进行解码。

我们看一下程序。LCD1602 显示：

Main.c 文件：
在这个文件中，主要处理显示内容，红外接收利用外部中断触发，所以整个解码部分都在中断子函数里，故 main()中只有显示部分代码，而 1602 的显示函数由于使用的都是一样的头文件，如果不清楚可以返回 1602 查看详细代码。

```
/*********************************************************************************
* 实验名                  : 1602 显示红外线值实验
* 使用的 IO       : 电机用 P1 口,键盘使用 P3.0、P3.1、P3.2、P3.3
* 实验效果        : LCD1602 显示出读取到的红外线的值
*   注意                          :
*********************************************************************************/

#include<reg51.h>
#include"lcd.h"

sbit IRIN=P3^2;

unsigned char code CDIS1[13]={" Red Control "};
unsigned char code CDIS2[13]={" IR-CODE:--H "};
unsigned char IrValue[6];
unsigned char Time;
void IrInit();
void DelayMs(unsigned int );
/*********************************************************************************
* 函数名          : main
* 函数功能          : 主函数
* 输入            : 无
* 输出            : 无
*********************************************************************************/

void main()
{
    unsigned char i;
    IrInit();
    LcdInit();
    LcdWriteCom(0x80);
    for(i=0;i<13;i++)
    {
        LcdWriteData(CDIS1[i]);
    }
```

```
    LcdWriteCom(0x80+0x40);
    for(i=0;i<13;i++)
    {
        LcdWriteData(CDIS2[i]);
    }
    while(1)
    {
        IrValue[5]=IrValue[2]>>4;           //高位
        IrValue[6]=IrValue[2]&0x0f;     //低位
        if(IrValue[5]>9)
        {
            LcdWriteCom(0xc0+0x09);         //设置显示位置
            LcdWriteData(0x37+IrValue[5]); //将数值转换为该显示的 ASCII 码
        }
        else
        {
            LcdWriteCom(0xc0+0x09);
            LcdWriteData(IrValue[5]+0x30); //将数值转换为该显示的 ASCII 码
        }
        if(IrValue[6]>9)
        {
            LcdWriteCom(0xc0+0x0a);
            LcdWriteData(IrValue[6]+0x37);      //将数值转换为该显示的 ASCII 码
        }
        else
        {
            LcdWriteCom(0xc0+0x0a);
            LcdWriteData(IrValue[6]+0x30);      //将数值转换为该显示的 ASCII 码
        }
    }
}
/*******************************************************************************
* 函数名         : DelayMs()
* 函数功能         : 延时
* 输入           : x
* 输出           : 无
*******************************************************************************/

void DelayMs(unsigned int x)   //0.14ms 误差 0us
{
 unsigned char i;
  while(x--)
 {
  for (i = 0; i<13; i++)
```

```
 {}
 }
}
/*******************************************************************************
* 函数名         : IrInit()
* 函数功能         : 初始化红外线接收
* 输入           : 无
* 输出           : 无
*******************************************************************************/

void IrInit()
{
    IT0=1;//下降沿触发
    EX0=1;//打开中断 0 允许
    EA=1;   //打开总中断

    IRIN=1;//初始化端口
}
/*******************************************************************************
* 函数名         : ReadIr()
* 函数功能         : 读取红外数值的中断函数
* 输入           : 无
* 输出           : 无
*******************************************************************************/

void ReadIr() interrupt 0
{
    unsigned char j,k;
    unsigned int err;
    EX0=0;
    Time=0;
    DelayMs(70);

    if(IRIN==0)     //确认是否真的接收到正确的信号
    {

        err=1000;               //1000*10us=10ms,超过说明接收到错误的信号
        while((IRIN==0)&&(err>0))   //等待前面 9ms 的低电平过去
        {                                                   //当两个条件都为真是循
环，如果有一个条件为假的时候跳出循环

            DelayMs(1);
            err--;
        }
```

```
        if(IRIN==1)          //正确等到 9ms 低电平
        {
            for(k=0;k<4;k++)         //共有 4 组数据
            {
                for(j=0;j<8;j++)     //接收一组数据
                {
                    err=500;
                    while((IRIN==1)&&(err>0))         //等待 4.5ms 的起始高电平过
去
                    {
                        DelayMs(1);
                        err--;
                    }
                    err=60;
                    while((IRIN==0)&&(err>0))//等待信号前面的 560us 低电平过去
                    while (!IRIN)
                    {
                        DelayMs(1);
                        err--;
                    }
                    err=500;
                    while((IRIN==1)&&(err>0))     //计算高电平的时间长度。
                    {
                        DelayMs(1);
                        Time++;
                        err--;
                        if(Time>30)
                        {
                            EX0=1;
                            return;
                        }
                    }
                    IrValue[k]>>=1;  //k 表示第几组数据
                    if(Time>=8)          //如果高电平出现大于 460us，那么是 1
                    {
                        IrValue[k]|=0x80;
                    }
                    Time=0;     //用完时间要重新赋值

                }
            }
        }
        if(IrValue[2]!=~IrValue[3])
        {
```

```
            EX0=1;
            return;
        }
    }
    EX0=1;
}
```

将程序下载到开发板观察运行结果,实验效果是：1602 单片机接收到的红外线键值的十六进制数据。

第十九讲 AD/DA 模数/数模转换

1.XPT2046 是一款 4 线制电阻式触摸屏控制器，内含 12 位分辨率 125KHz 转换速率逐步逼近型 A/D 转换器。XPT2046 支持从 1.5V 到 5.25V 的低电压 I/O 接口。XPT2046 能通过执行两次 A/D 转换查出被按的屏幕位置， 除此之外，还可以测量加在触摸屏上的压力。内部自带 2.5V 参考电压，可以作为辅助输入、温度测量和电池监测之用，电池监测的电压范围可以从 0V 到 6V。XPT2046 片内集成有一个温度传感器。 在 2.7V 的典型工作状态下，关闭参考电压，功耗可小于 0.75mW。XPT2046 采用微小的封装形式：TSSOP-16,QFN-16 和 VFBGA－48。工作温度范围为-40℃～+85℃。与 ADS7846、TSC2046、AK4182A 完全兼容；

2. 典型应用电路

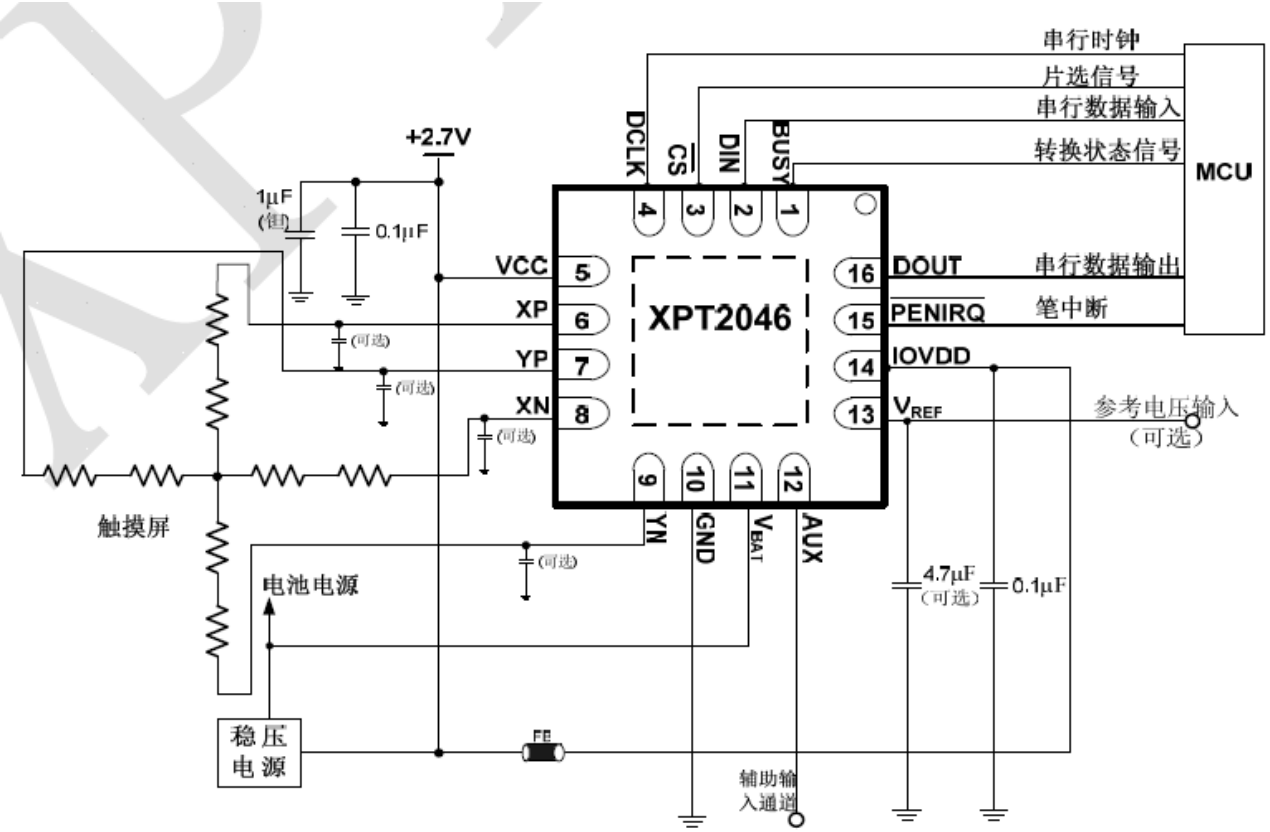


图1 XPT2046 典型应用电路

3. 极限参数

表1 芯片极限参数表

名称	参数
VCC和IOVDD电压	−0.3V至+6V
模拟输入信号电压	−0.3V至+VCC+0.3V
数字输入信号电压	−0.3V至IOVDD+0.3V
功耗	250mW
最大结温	+150℃
工作温度	−40℃～+85℃
贮存温度	-65℃～+150℃
焊接温度（小于10秒）	+300℃

4. 引脚功能描述：

QFN引脚号	TSSOP引脚号	VFBGA引脚号	名称	说明
1	13	A5	BUSY	忙时信号线。当$\overline{CS}$为高电平时为高阻状态
2	14	A4	DIN	串行数据输入端。当$\overline{CS}$为低电平时，数据在DCLK上升沿锁存进来
3	15	A3	$\overline{CS}$	片选信号。控制转换时序和使能串行输入输出寄存器，高电平时ADC掉电
4	16	A2	DCLK	外部时钟信号输入
5	1	B1和C1	VCC	电源输入端
6	2	D1	XP	XP位置输入端
7	3	E1	YP	YP位置输入端
8	4	G2	XN	XN位置输入端
9	5	G3	YN	YN位置输入端
10	6	G4和G5	GND	接地
11	7	G6	V_{BAT}	电池监视输入端
12	8	E7	AUX	ADC辅助输入通道
13	9	D7	V_{REF}	参考电压输入/输出
14	10	C7	IOVDD	数字电源输入端
15	11	B7	$\overline{PENIRQ}$	笔接触中断引脚
16	12	A6	DOUT	串行数据输出端。数据在DCLK的下降沿移出，当$\overline{CS}$高电平时为高阻状态

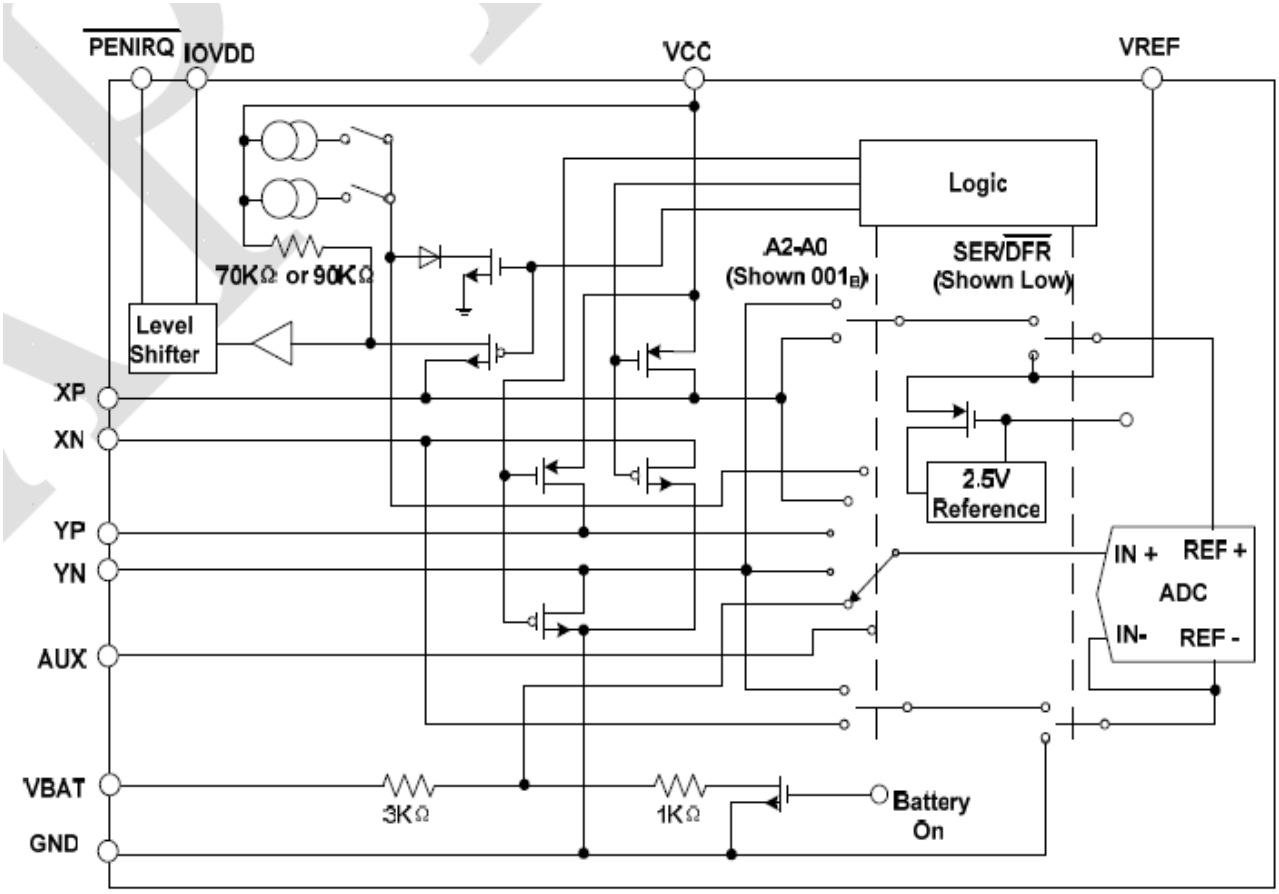
5. 工作原理：

XPT2046 是一种典型的逐次逼近型模数转换器（SAR ADC），包含了采样/保持、模数转换、串口数据 输出等功能。同时芯片集成有一个 2.5V 的内部参考电压源、温度检测电路，工作时使用外部时钟。XPT2046 可以单电源供电，电源电压范围为 2.7V～5.5V。参考电压值直接决定 ADC 的输入范围，参考电压可以使用内部参考电压，也可以从外部直接输入 1V～VCC 范围内的参考电压（要求外部参考电压源输出阻抗低）。X、Y、Z、V_{BAT}、Temp 和 AUX 模拟信号经过片内的控制寄存器选择后进入 ADC，ADC 可以配置为单端或差分模式。选择 V_{BAT}、Temp 和

AUX 时应该配置为单端模式；作为触摸屏应用时，应该配置为差分模式，这可有效消除由于驱动开关的寄生电阻及外部的干扰带来的测量误差，提高转换精度。

6. 模拟输入特性

下图描述了 XPT2046 片内多路选择器、ADC 的模拟差分输入和差分参考电压基准。表 3 和表 4 说明了 A2、A1、A0 和 SER/ DFR控制位与 XPT2046 的配置关系。这些控制位来自 DIN 脚的串行数据（更详细的说明见数字接口部分）。



在这个实验中我们使用的就是单端模式：

通过设置相关寄存器即可实现。

表3 单端模式输入配置（SER/$\overline{DFR}$=1）

A2	A1	A0	V_{BAT}	AUX_{IN}	TEMP	YN	XP	YP	Y-位置	X-位置	Z_1-位置	Z_2-位置	X-驱动	Y-驱动
0	0	0			+IN (TEMP0)								off	off
0	0	1					+IN		测量				Off	On
0	1	0	+IN										Off	Off
0	1	1					+IN				测量		XN, On	YP, On
1	0	0				+IN						测量	XN, On	YP, On
1	0	1						+IN		测量			On	Off
1	1	0		+IN									Off	Off
1	1	1			+IN (TEMP1)								Off	Off

7. 单端工作模式

SER/DFR置为高电平时，XPT2046工作在为单端模式，单端工作模式的应用原理图7所示。

单端模式简单，在采样过程完成后，转换过程中可以关闭驱动开关，降低功耗。但这种模式的缺点是精度直接受参考电压源的精度限制，同时由于内部驱动开关的导通电阻存在，导通电阻与触摸屏电阻的分压作用，也会带来测量误差。

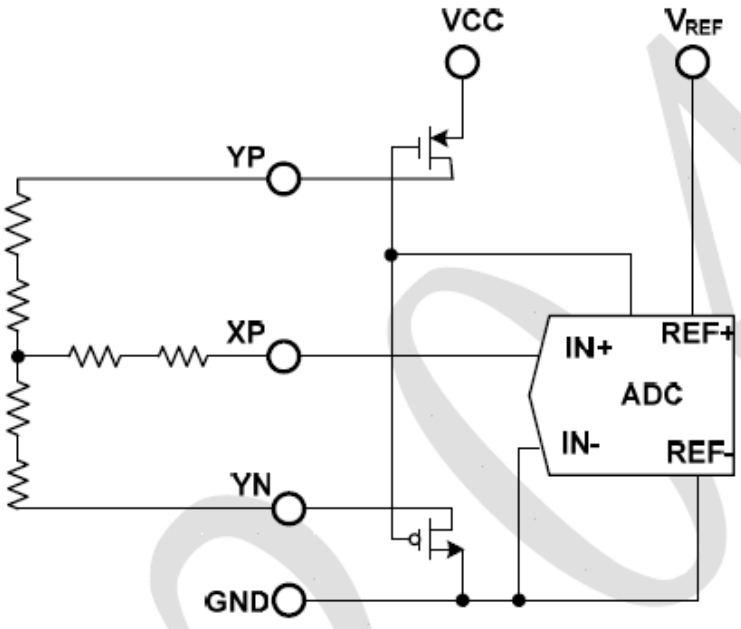


图7 单端模式工作示意图（SER/$\overline{DFR}$＝1，Y开关闭合，XP作为模拟输入）

8. 数字接口

XPT2046数据接口是串行接口，其典型工作时序如图12所示，图中展示的信号来自带有基本串行接口的单片机或数据信号处理器。处理器和转换器之间的的通信需要8个时钟周期，可采用SPI、SSI和Microwire等同步串行接口。一次完整的转换需要24个串行同步时钟（DCLK）来完成。

前8个时钟用来通过DIN引脚输入控制字节。当转换器获取有关下一次转换的足够信息后，接着根据获得的信息设置输入多路选择器和参考源输入，并进入采样模式，如果需要，将启动触摸面板驱动器。3个多时钟周期后，控制字节设置完成，转换器进入转换状态。这时，输入采样－保持器进入保持状态，触摸面板驱动器停止工作（单端工作模式）。接着的12个时钟周期将完成真正的模数转换。

如果是度量比率转换方式（SER/DFR＝0），驱动器在转换过程中将一直工作，第13个时钟将输出转换结果的最后一位。剩下的3个多时钟周期将用来完成被转换器忽略的最后字节（DOUT置低）。

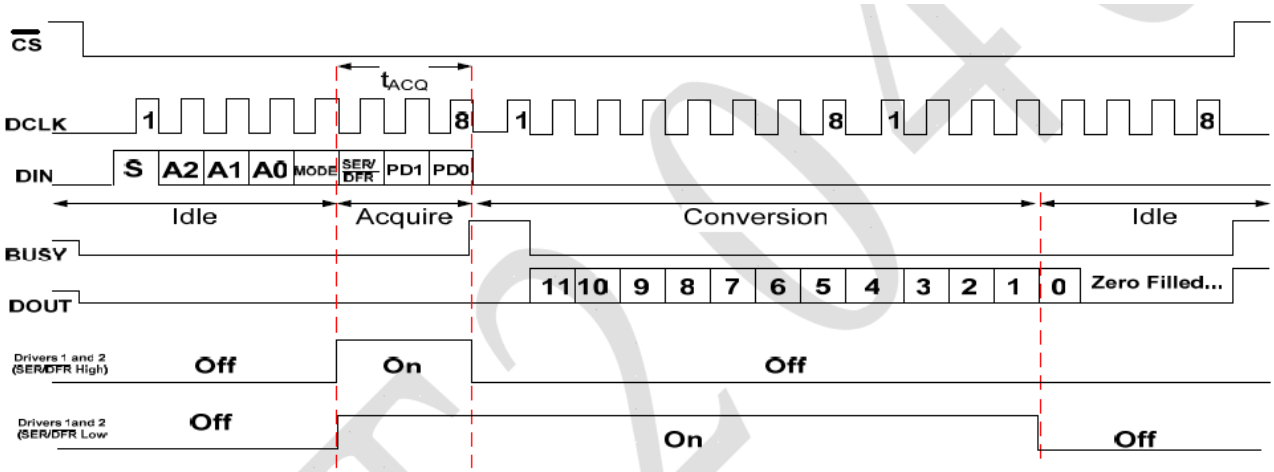
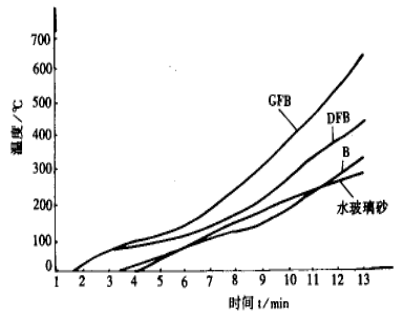


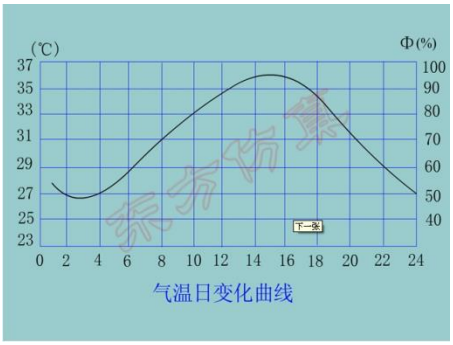
图12 8位总线接口，无DCLK时钟延迟，24时钟周期转换时序

1. AD

模拟量是指变量在一定范围连续变化的量；也就是在一定范围（定义域）内可以取任意值。如温度、压力、位移、图像等都是模拟量，电子线路中模拟量通常包括模拟电压和模拟电流，生活用电 220V 交流正弦波就属于模拟电压，随着负载大小的变化，其电流大小也跟着变化，这里的电流信号也属于

模拟电流，图左和图右所表示的信号就属于模拟量。





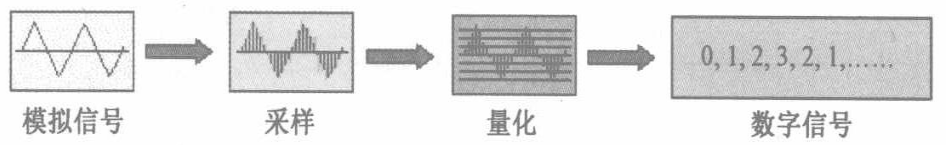
上图所示信号的幅值随着时间变化而连续变化的量就是模拟量，模拟量有可能是标准的正弦波，有可能是不规则的任何波形，也有可能是规则的方波、三角波等，当我们用数值表示其大小时，通常用十进制数表示，如 2.3V，5A，47N 等。

单片机系统内部运算时用的全部是数字量，即 0 和 1，因此对单片机系统而言，我们无法直接操作模拟量，必须将模拟量转换成数字量。所谓数字量，就是用一系列 0 和 1 组成的二进制代码表示某个信号大小的量。用数字量表示同一个模拟量时，数字位数可以多也可以少，位数越多则表示的精度越高，位数越少表示的精度就越低。比如对图 5.1.2 中的正弦波模拟量，我们可以用一个 00000-11111 的 5 位二进制数字量来表示它，5 位二进制数最多只能有 32 种组合形式，因此需把这个正弦波最大值与最小值之间分成犯等分，每一等分用一组 5 位二进制数来表示。很显然，如果用 32 等分数字量表示这个模拟量的话，任意两相邻等分之间的模拟量我们便无法表示出来，唯有增加等分，也就是再增加数字量的位数才可以表示出来。因此，若要用数字量完全表示一个模拟量的话，其数字量位数就为无穷多位。但若要设计出这样的硬件，当今的技术还无法实现。

单片机在采集模拟信号时，通常都需要在前端加上模拟量/数字量转换器，简称模/数转换器.即常说的 A/D (Analog to Digital)芯片。当单片机在输出模拟信号时，通常在输出级要加上数字量/模拟量转换器，简称数/模转换器，即常说的 D/A (Digital to Analog)芯片，下面几节分别讲解 A/D 和 D/A 的原理。

2. A/D 转换原理及参数指标

在 A/D 转换器中，因为输入的模拟信号在时间上是连续的，而输出的数字信号代码是离散的，所以 A/D 转换器在进行转换时，必须在一系列选定的瞬间(时间坐标轴上的一些规定点上)对输入的模拟信号采样，然后再把这些采样值转换为数字量。因此，一般的 A/D 转换过程是通过采样保持、量化和编码这三个步骤完成的，即首先对输入的模拟电压信号采样，采样结束后进入保持时间，在这段时间内将采样的电压量转化为数字量，并按一定的编码形式给出转换结果，然后开始下一次采样。下图给出模拟量到数字量转换过程的框图。



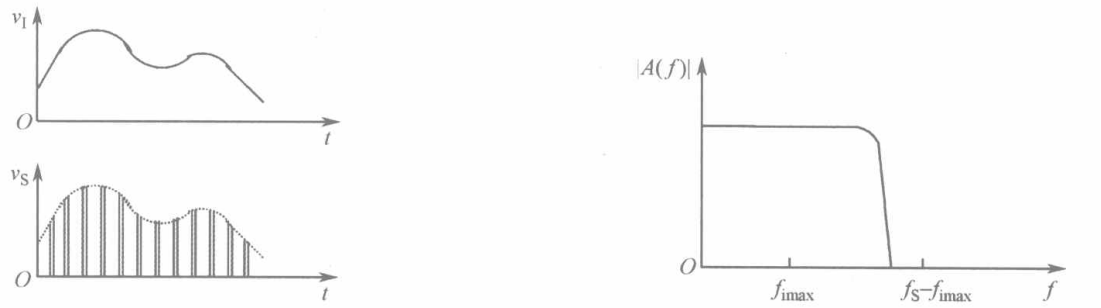
模拟量到数字量转换过程框图

1.采样定理

可以证明，为了正确无误地用下左图中所示的采样信号 Vs 表示模拟信号 Vi，必须满足:

$$f_s \geq 2f_{imax}$$

在满足采样定理的条件下，可以用一个低通滤波器将信号 Vs 还原为 Vi，这个低通滤波器的电压传输系数|A（f）|在低于f_{imax}、的范围内应保持不变，而在,$f_s - f_{imax}$以前应迅速下降为零，如下右图所示。因此，采样定理规定了 A/D 转换的频率下限。



因此，A/D 转换器工作时的采样频率必须高于所规定的频率。采样频率提高以后，留给 A/D 转换器每次进行转换的时间也相应缩短了，这就要求转换电路必须具备更快的工作速度。因此，不能无限制地提高采样频率，通常取

$f_s = (3-5) f_{imax}$已经能够满足要求。

因为每次把采样电压转换为相应的数字量都需要一定的时间，所以在每次采样以后，必须把采样电压保持一段时间。可见，进行 A/D 转换时所用的输入电压，实际上是每次采样结束时的 Vi 值。

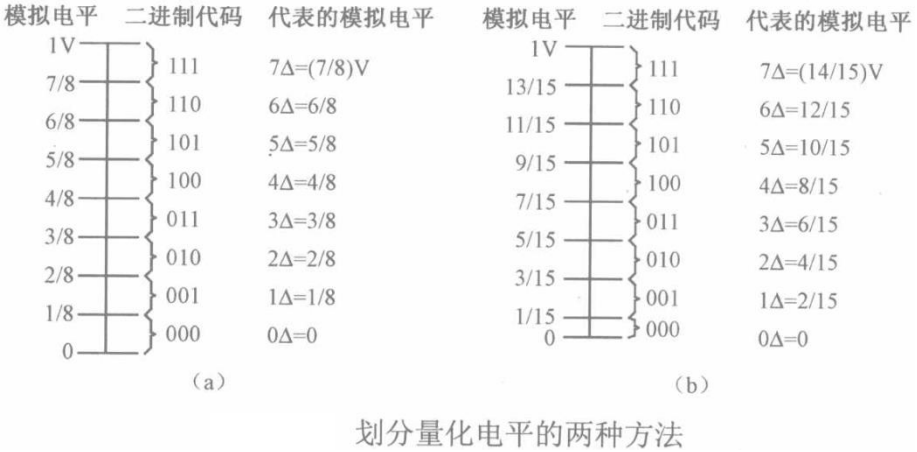
2.量化和编码

我们知道，数字信号不仅在时间上是离散的，而且数值上的变化也不是连续的。这就是说，任何一个数字量的大小，都是以某个最小数量单位的整倍数来表示的。因此，在用数字量表示采样电压时，也必须把它化成这个最小数量单位的整倍数，这个转化过程就叫做量化。

所规定的最小数量单位叫做量化单位，用△表示。显然，数字信号最低有效位中的 1 表示的数量大小，就等于△。把量化的数值用二进制代码表示，称为编码。这个二进制代码就是 A/D 转换的输出信号。

既然模拟电压是连续的，那么它就不一定能被△整除，因而不可避免地会引入误差，我们把这种误差称为量化误差。在把模拟信号划分为不同的量化等级时，用不同的划分方法可以得到不同的量化误差。

假定需要把 0---1V 的模拟电压信号转换成三位二进制代码，这时便可以取△=(1/8)V,并规定凡数值在 0---(1/8)V 之间的模拟电压都当做 0X△看待，用二进制的 000 表示:凡数值在((1/8)~(2/8) V 之间的模拟电压都当做 1x△看待，用二进制的 001 表示，……，如下图所示。不难看出，最大的量化误差可达△，即(1/8)V.



划分量化电平的两种方法

为了减少量化误差，通常采用图（b）所示的划分方法，取量化单位△

=(2/15) V,并将 000 代码所对应模拟电压规定为 0- (1/15) V，即 0---△/2。这时，最大量化误差将减少为△/2= (1/15) V。这个道理不难理解，因为现在把每个二进制代码所代表的模拟电压值规定为它所对应的模拟电压范围的中点，所以最大的量化误差自然就缩小为△/2 了。

A/D 转换器的参数指标

3. 分辨率—它说明 A/D 转换器对输入信号的分辨能力。

A/D 转换器的分辨率以输出二进制数的位数表示。从理论上讲，n 位输出的 A/D 转换器能区分 2n 个不同等级的输入模拟电压，能区分输入电压的最小值为满量程输入的 1/2n。在最大输入电压一定时，输出位数愈多，量化单位愈小，分辨率愈高。常用的有 8, 10, 12, 16,24, 32 位等。例如，A/D 转换器输出为 8 位二进制数，输入信号最大值为 5V，那么这个转换器应能区分输入信号的最小电压为 19.53 mV。再如，某 A/D 转换器输入模拟电压的变化范围为-10V-+10V，转换器为 8 位，若第一位用来表示正、负号，其余 7 位表示信号幅值，则最末一位数字可代表 80mV 模拟电压，即转换器可以分辨的最小模拟电压为 80mV。而同样情况下，用一个 10 位转换器能分辨的最小模拟电压为 20mV。

4. 转换误差—表示 A/D 转换器实际输出的数字量与理论输出数字量之间的差别。

在理想情况下，输入模拟信号所有转换点应当在一条直线上，但实际的特性不能做到输入模拟信号所有转换点在一条直线上。转换误差是指实际的转换点偏离理想特性的误差，一般用最低有效位来表示。例如，给出相对误差<=士 LSB/2，这就表明实际输出的数字量和理论上应得到的输出数字量之间的误差小于最低位的一半。注意，在实际使用中当使用环境发生变化时，转换误差也将发生变化。

5. 转换精度—它是 A/D 转换器的最大量化误差和模拟部分精度的共同体现。

具有某种分辨率的转换器在量化过程中由于采用了四舍五入的方法，因此最大量化误差应为分辨率数值的一半。如上例，8 位转换器最大量化误差应为 40mV (80mVx0.5=40mV)，全量程的相对误差则为 0.4% (40mV / 10Vx100%)。可见，A/D 转换器数字转换的精度由最大量化误差决定。实际上，

许多转换器末位数字并不可靠，实际精度还要低一些。

由于含有 A/D 转换器的模/数转换模块通常包括有模拟处理和数字转换两部分，因此整个转换器的精度还应考虑模拟处理部分(如积分器、比较器等)的误差。一般转换器的模拟处理误差与数字转换误差应尽量处在同一数量级，总误差则是这些误差的累加和。例如，一个 10 位 A/D 转换器用其中 9 位计数时的最大相对量化误差为 29x0.5=0.1%，若模拟部分精度也能达到 0.1%，则转换器总精度可接近 0.2%。

6. 转换时间—指 A/D 转换器从转换控制信号到来开始，到输出端得到稳定的数字信号所经过的时间。

不同类型的转换器转换速度相差甚远。其中并行比较 A/D 转换器转换速度最高，8 位二进制输出的单片集成 A/D 转换器转换时间可达 50ns 以内。逐次比较型 A/D 转换器次之，它们多数转换时间在 10-50us 之间，也有达几百纳秒的。间接 A/D 转换器的速度最慢，如双积分 A/D 转换器的转换时间大都在几十毫秒至几百毫秒之间。在实际应用中，应从系统数据总的位数、精度要求、输入模拟信号的范围及输入信号极性等方面综合考虑 A/D 转换器的选用。

7. DA

我们处在一个数字时代，而我们的视觉、听觉、感觉、嗅觉等所感知的却是一个模拟世界。如何将数字世界与模拟世界联系在一起，正是模拟数字转换器(ADC)和数字模拟转换器(DAC)大显身手之处。任何一个信号链系统，都需要传感器来探测来自模拟世界的电压、电流、温度、压力等信号。这些传感器探测到的信号量被送到放大器中进行放大，然后通过ADC 把模拟信号转化为数字信号，经过处理器、DSP 或FPGA 信号处理后，再经由DAC 还原为模拟信号。所以ADC 和DAC 在信号链的框架中起着桥梁的作用，即模拟世界与数字世界的一个接口。

DAC的原理稍微有点复杂，下面我列出一段话，如果无法理解，可以暂时跳过，不理解原理不一定会影响我们对DAC的简单应用：

DAC主要由数字寄存器、模拟电子开关、位权网络、求和运算放大器和基准电压源（或恒流源）组成。用存于数字寄存器的数字量的各位数码，分别控制对应位的模拟电子开关，使数码为1的位在位权网络上产生与其位权成正比的电流

值，再由运算放大器对各电流值求和，并转换成电压值。

我讲点个人的理解，不准确但是有助于建立一个思维模型：上述的模拟电子开关都分别接着一个分压的器件，比如说电阻。模拟开关的个数取决于DAC 的精度。那么N 个电子开关就把基准电压分为N 份（并不是平均分哦），而这些开关根据输入的二进制每一位数据对应开启或者关闭，把分压的器件上的电压引入输出电路中。

但是如果无论如何你都不明白的话，请保持参考电压的值符合器件要求，并且尽可能的稳定。

8. D/A转换器的参数指标

分辨率—D/A转换器模拟输出电压可能被分离的等级数

输入数字量位数越多，输出电压可分离的等级越多，即分辨率越高。在实际应用中，往往用输入数字量的位数表示D/A转换器的分辨率。此外，D/A转换器也可以用能分辨的最小输出电压(此时输入的数字代码只有最低有效位为1，其余各位都是0)与最大输出电压(此时输入的数字代码各有效位全为1)之比给出。

转换误差—表示D/A转换器实际输出的模拟量与理论输出模拟量之间的差别。

转换误差的来源很多，转换器中各元件参数值的误差，基准电源不够稳定和运算放大器零漂的影响等。D/A转换器的绝对误差(或绝对精度)是指输入端加入最大数字量(全1)时，D/A转换器的理论值与实际值之差。该误差值应低于LSB/2。

建立时间—指输入数字量变化时，输出电压变化到相应稳定电压值所需时间。

一般用D/A转换器输入的数字量从全0变为全1时，输出电压达到规定的误差范围(±LSB/2)时所需时间表示。D/A转换器的建立时间较快，单片集成D/A转换器建立时间最短可达0.1us以内。

转换速率(SR)—大信号工作状态下模拟电压的变化率。

温度系数—指在输入不变的情况下，输出模拟电压随温度变化产生的变化量。一般用满刻度输出条件下温度每升高1° C，输出电压变化的百分数作为温度系数。

- **开发板使用的是 ADDA 转换芯片** XPT2046

10. 光敏电阻

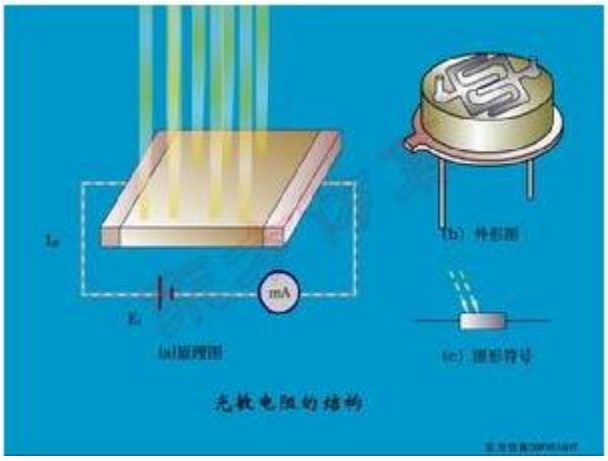
光敏电阻又称光导管，常用的制作材料为硫化镉，另外还有硒、硫化铝、硫化铅和硫化铋等材料。这些制作材料具有在特定波长的光照射下，其阻值迅速减小的特性。这是由于光照产生的载流子都参与导电，在外加电场的作用下作漂移运动，电子奔向电源的正极，空穴奔向电源的负极，从而使光敏电阻器的阻值迅速下降。

光敏电阻器是利用半导体的光电效应制成的一种电阻值随入射光的强弱而改变的电阻器；入射光强，电阻减小，入射光弱，电阻增大。

光敏电阻器一般用于光的测量、光的控制和光电转换（将光的变化转换为电的变化）。常用的光敏电阻器硫化镉光敏电阻器，它是由半导体材料制成的。光敏电阻器的阻值随入射光线（可见光）的强弱变化而变化，在黑暗条件下，它的阻值（暗阻）可达 1~10M 欧,在强光条件（100LX）下，它阻值（亮阻）仅有几百至数千欧姆。光敏电阻器对光的敏感性（即光谱特性）与人眼对可见光（0.4~0.76）μm 的响应很接近，只要人眼可感受的光，都会引起它的阻值变化。设计光控电路时，都用白炽灯泡（小电珠）光线或自然光线作控制光源，使设计大为简化。

光敏电阻的工作原理是基于内光电效应。在半导体光敏材料两端装上电极引线，将其封装在带有透明窗的管壳里就构成光敏电阻，为了增加灵敏度，两电极常做成梳状。用于制造光敏电阻的材料主要是金属的硫化物、硒化物和碲化物等半导体。通常采用涂敷、喷涂、烧结等方法在绝缘衬底上制作很薄的光敏电阻体及梳状欧姆电极，接出引线，封装在具有透光镜的密封壳体内，以免受潮影响其灵敏度。在黑暗环境里，它的电阻值很高，当受到光照时，只要光子能量大于半导体材料的禁带宽度，则价带中的电子吸收一个光子的能量后可跃迁到导带，并在价带中产生一个带正电荷的空穴，这种由光照产生的电子—空穴对了半导体材料中载流子的数目，使

其电阻率变小，从而造成光敏电阻阻值下降。光照愈强，阻值愈低。入射光消失后，由光子激发产生的电子—空穴对将复合，光敏电阻的阻值也就恢复原值。在光敏电阻两端的金属电极加上电压，其中便有电流通过，受到波长的光线照射时，电流就会随光强的而变大，从而实现光电转换。光敏电阻没有极性，纯粹是一个电阻器件，使用时既可加直流电压，也加交流电压。半导体的导电能力取决于半导体导带内载流子数目的多少。



11. 热敏电阻

热敏电阻器是敏感元件的一类，按照温度系数不同分为正温度系数热敏电阻器（PTC）和负温度系数热敏电阻器（NTC）。热敏电阻器的典型特点是对温度敏感，不同的温度下表现出不同的电阻值。正温度系数热敏电阻器（PTC）在温度越高时电阻值越大，负温度系数热敏电阻器（NTC）在温度越高时电阻值越低，它们同属于半导体器件

热敏电阻的主要特点是：①灵敏度较高，其电阻温度系数要比金属大10～100 倍以上，能检测出 10-6℃的温度变化；②工作温度范围宽，常温器件适用于-55℃～315℃，高温器件适用温度高于 315℃（目前最高可达到2000℃），低温器件适用于-273℃～55℃；③体积小，能够测量其他温度计无法测量的空隙、腔体及生物体内血管的温度；④使用方便，电阻值可在0.1～100kΩ 间任意选择；⑤易加工成复杂的形状，可大批量生产；⑥稳定性好、过载能力强。

PTC（Positive Temperature CoeffiCient）是指在某一温度下电阻急

剧增加、具有正温度系数的热敏电阻现象或材料，可专门用作恒定温度传感器．该材料是以 BaTiO3 或 SrTiO3 或 PbTiO3 为主要成分的烧结体，其中掺入微量的 Nb、Ta、 Bi、 Sb、Y、La 等氧化物进行原子价控制而使之半导化，常将这种半导体化的 BaTiO3 等材料简称为半导（体）瓷；同时还添加增大其正电阻温度系数的 Mn、Fe、Cu、Cr 的氧化物和起其他作用的添加物，采用一般陶瓷工艺成形、高温烧结而使钛酸铂等及其固溶体半导化，从而得到正特性的热敏电阻材料．其温度系数及居里点温度随组分及烧结条件（尤其是冷却温度）不同而变化。

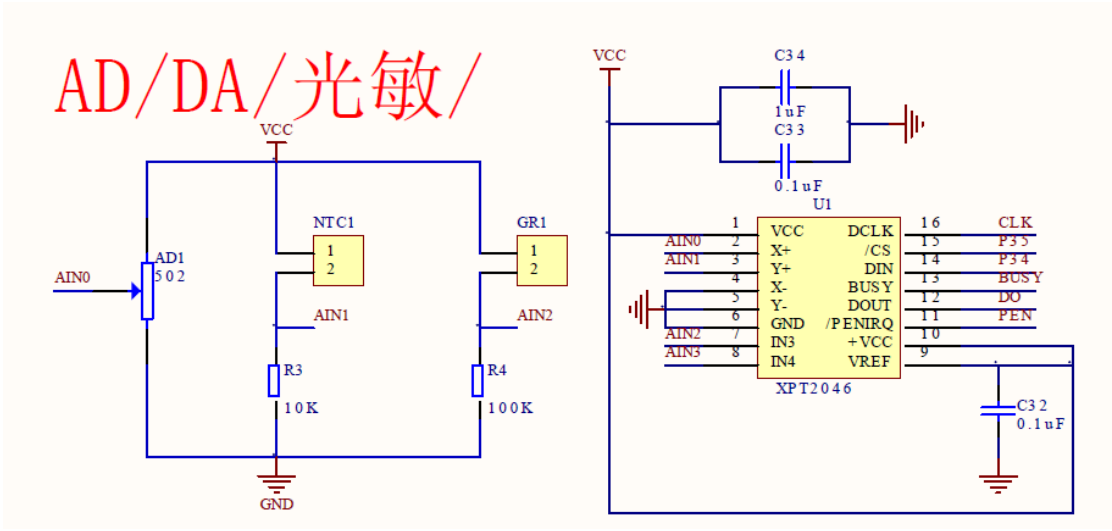
NTC（Negative Temperature CoeffiCient）是指随温度上升电阻呈指数关系减小、具有负温度系数的热敏电阻现象和材料．该材料是利用锰、铜、硅、钴、铁、镍、锌等两种或两种以上的金属氧化物进行充分混合、成型、烧结等工艺而成的半导体陶瓷，可制成具有负温度系数（NTC）的热敏电阻．其电阻率和材料常数随材料成分比例、烧结气氛、烧结温度和结构状态不同而变化．现在还出现了以碳化硅、硒化锡、氮化钽等为代表的非氧化物系 NTC 热敏电阻材料．

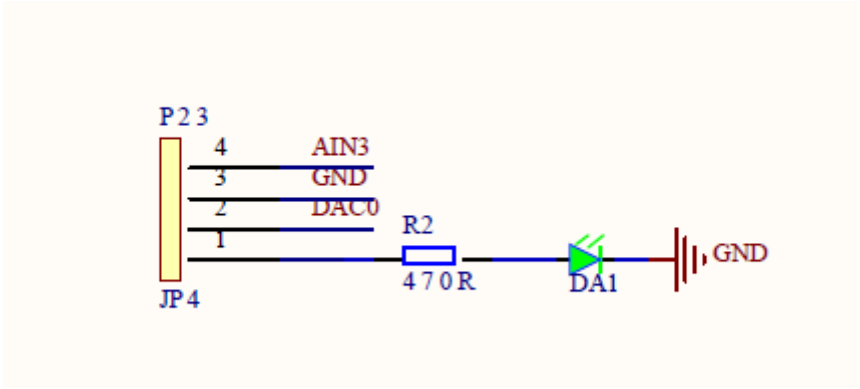
NTC 热敏半导瓷大多是尖晶石结构或其他结构的氧化物陶瓷，具有负的温度系数，电阻值可近似表示为：

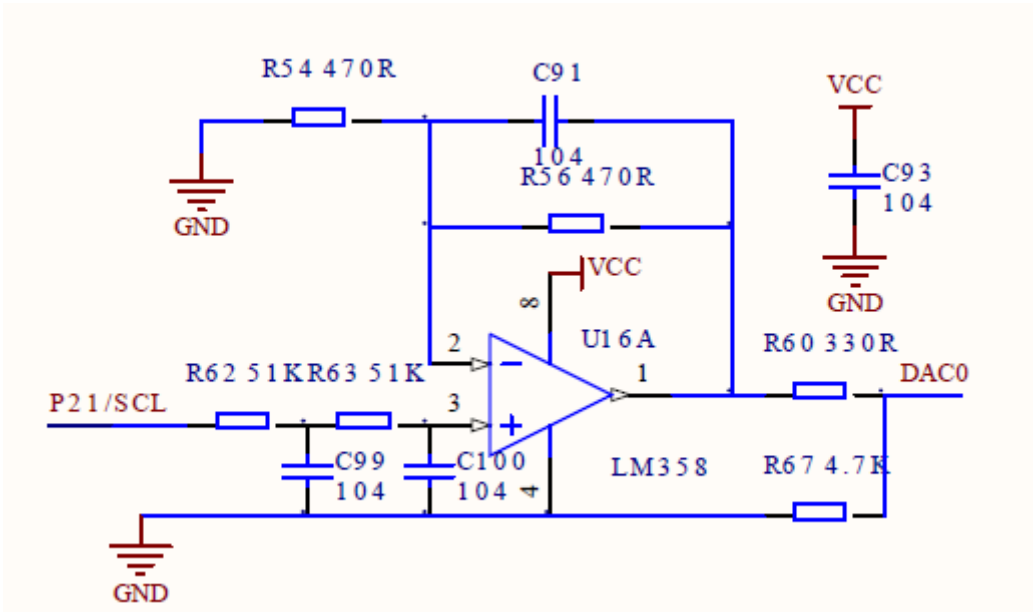
Rt = RT *EXP(Bn*（1/T-1/T0)

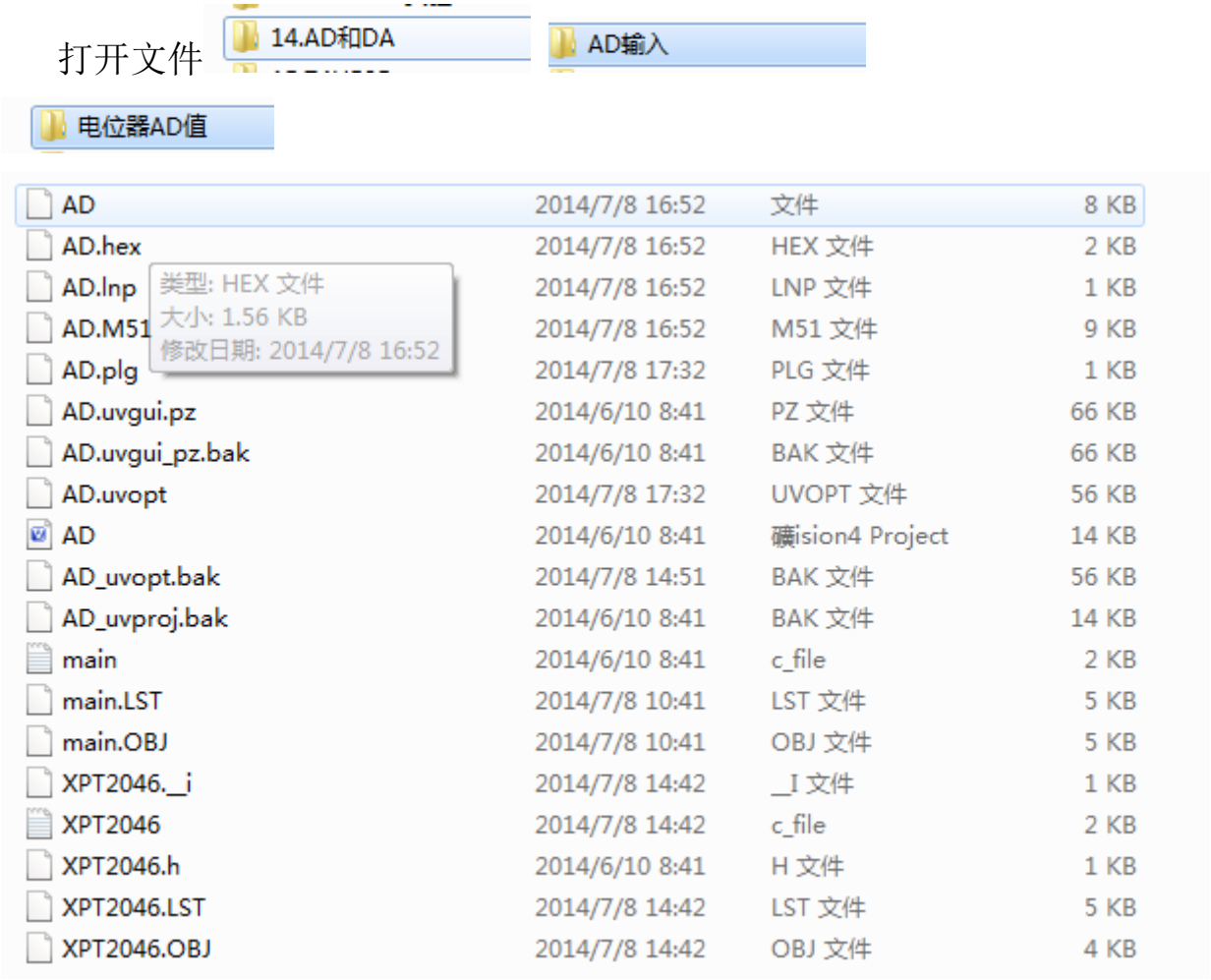
式中 RT、RT0 分别为温度 T、T0 时的电阻值，Bn 为材料常数．陶瓷晶粒本身由于温度变化而使电阻率发生变化，这是由半导体特性决定的．

下面看一下开发板上的电路原理图。


AD/DA/光敏/
VCC
NTC1
GR1
AD1
502
AIN0
AIN1
AIN2
R3
10K
R4
100K
GND
C34
1uF
C33
0.1uF
U1
VCC
X+
Y+
X-
Y-
GND
IN3
IN4
DCLK
/CS
DIN
BUSY
DOUT
/PENIRQ
+VCC
VREF
XPT2046
CLK
P35
P34
BUSY
DO
PEN
C32
0.1uF


P23
AIN3
GND
DAC0
JP4
R2
470R
DA1
GND


R54 470R
C91
104
R56 470R
VCC
C93
104
GND
VCC
U16A
R62 51K
R63 51K
P21/SCL
C99
104
C100
104
LM358
R60 330R
DAC0
R67 4.7K
GND

打开文件 

具体的源程序可以按照上述步骤查看。（程序内已经注释的比较详细），下载 HEX 文件后即可观察实验现象。

第二十讲 液晶屏显示

液晶显示器，或称 LCD（Liquid Crystal Display），为平面超薄的显示设备，它由一定数量的彩色或黑白像素组成，放置于光源或者反射面前方。液晶显示器功耗很低，因此倍受工程师青睐，适用于使用电池的电子设备。

要追溯液晶显示器的来源，必须先从「液晶」的诞生开始讲起。在公元 1888 年,一位奥地利的植物学家,菲德烈.莱尼泽(Friedrich Reinitzer)发现了一种特殊的物质。他从植物中提炼出一种称为螺旋性甲苯酸盐的化合物，在为这种化合物做加热实验时，意外的发现此种化合物具有两个不同温度的熔点。而它的状态介于我们一般所熟知的液态与固态物质之间，有点类似肥皂水的胶状溶液，但它在某一温度范围内却具有液体和结晶双方性质的物质,也由于其独特的状态,后来便把它命名为「Liquid Crystal」,就是液态结晶物质的意思。

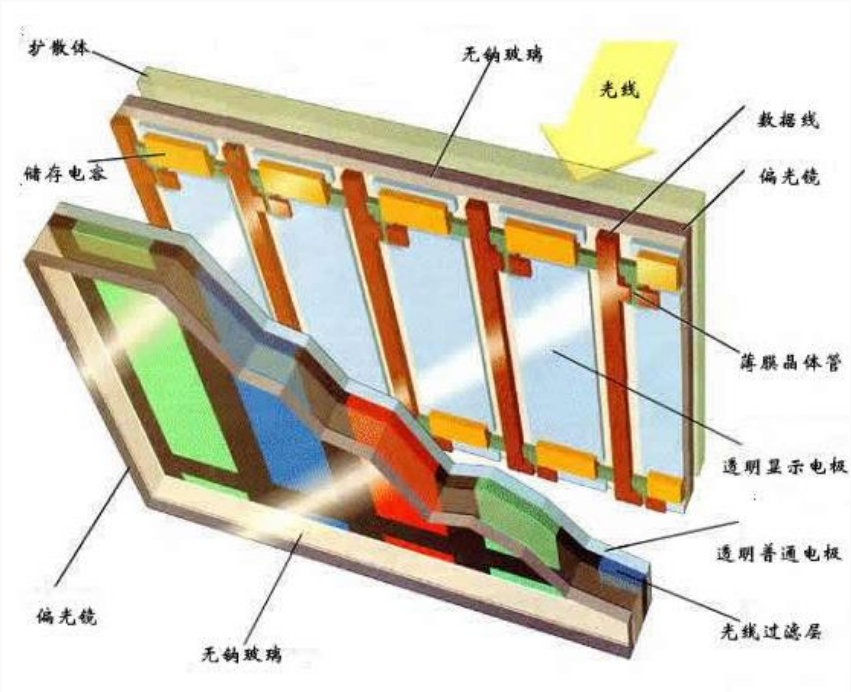
虽然液晶早在 1888 年就被发现，但是真正实用在生活的用品时，却是在 80 年后的事情了。 公元 1968 年，在美国 RCA 公司（收音机与电视的发明公司）的沙诺夫研发中心，工程师们发现液晶分子会受到电压的影响，改变其分子的排列状态，并且可以让射入的光线产生偏转的现象。利用这一原理，RCA 公司发明了世界第一台使用液晶显示的屏幕。而后，液晶显示技术被广泛的用在一般的电子产品中，举凡计算器、电子表、手机屏幕、医院所使用的仪器（因为有辐射计量的考虑）或是数字相机上面的屏幕等等。 令人玩味的是，液晶的发现比真空管或是阴极射线管还早，但世人了解此一现象的并不多，直到 1962 年才有第一次，由 RCA 研究小组的化学家乔．卡司特雷诺（Joe Castellano）先生所出版的书籍来描述。而与映像管相同的，这两项技术虽然都是由美国的 RCA 公司所发明的，却分别被日本的索尼（Sony）与夏普（Sharp）两家公司发扬光大。

液晶显示器是以液晶材料为基本组件，由于液晶是介于固态和液态之间，不但具有固态晶体光学特性，又具有液态流动特性，所以已经可以说是一个中间相。而要了解液晶的所产生的光电效应，我们必须来解释液晶的物理特性，包括它的黏性（visco-sity）与弹性（elasticity）和其极化性（polarizalility)。液晶的黏性和弹性从流体力学的观点来看，可说是一个具有排列性质的液体，依照作用力量不同的方向，应该有不同的效果。就好像是将一把短木棍扔进流动的河水中，短木棍随着河水流着，起初显得凌乱，过了一会儿，所有短木棍的长轴都自然的变成与河水流动的方向一致，这表示着次黏性最低的流动方式，也是流动自由能最低的一个

物理模型。 此外，液晶除了有黏性的反应外，还具有弹性的反应，它们都是对于外加的力量，呈现了方向性的效果。也因此光线射入液晶物质中，必然会按照液晶分子的排列方式行进，产生了自然的偏转现像。至于液晶分子中的电子结构，都具备着很强的电子共轭运动能力，所以当液晶分子受到外加电场的作用，便很容易的被极化产生感应偶极性（induced dipolar），这也是液晶分子之间互相作用力量的来源。而一般电子产品中所用的液晶显示器，就是是利用液晶的光电效应，藉由外部的电压控制，再透过液晶分子的折射特性，以及对光线的旋转能力来获得亮暗情况（或著称为可视光学的对比），进而达到显像的目的。

简单的来说，屏幕能显示的基本原理就是在两块平行板之间填充液晶材料，通过电压来改变液晶材料内部分子的排列状况，以达到遮光和透光的目的来显示深浅不一，错落有致的图象，而且只要在两块平板间再加上三元色的滤光层，就可实现显示彩色图象。

认识了它的结构和原理，了解了它的技术和工艺特点，才能在选购时有的放矢，在应用和维护时更加科学合理。液晶是一种有机复合物，由长棒状的分子构成。在自然状态下，这些棒状分子的长轴大致平行。



LCD 第一个特点是必须将液晶灌入两个列有细槽的平面之间才能正常工作。这两个平面上的槽互相垂直(90 度相交)，也就是说，若一个平面上

的分子南北向排列，则另一平面上的分子东西向排列，而位于两个平面之间的分子被强迫进入一种 90 度扭转的状态。由于光线顺着分子的排列方向传播，所以光线经过液晶时也被扭转 90 度。但当液晶上加一个电压时，分子便会重新垂直排列，使光线能直射出去，而不发生任何扭转。

LCD 的第二个特点是它依赖极化滤光片和光线本身，自然光线是朝四面八方随机发散的，极化滤光片实际是一系列越来越细的平行线。这些线形成一张网，阻断不与这些线平行的所有光线，极化滤光片的线正好与第一个垂直，所以能完全阻断那些已经极化的光线。 只有两个滤光片的线完全平行，或者光线本身已扭转到与第二个极化滤光片相匹配，光线才得以穿透。**一方面**，LCD 正是由这样两个相互垂直的极化滤光片构成，所以在正常情况下应该阻断所有试图穿透的光线。但是，由于两个滤光片之间充满了扭曲液晶，所以在光线穿出第一个滤光片后，会被液晶分子扭转 90 度，最后从第二个滤光片中穿出。**另一方面**，若为液晶加一个电压，分子又会重新排列并完全平行，使光线不再扭转，所以正好被第二个滤光片挡住。总之，加电将光线阻断，不加电则使光线射出。当然，也可以改变 LCD 中的液晶排列，使光线在加电时射出，而不加电时被阻断。但由于液晶屏幕几乎总是亮着的，所以只有"加电将光线阻断"的方案才能达到最省电的目的。

液晶屏常见分类

1. STN 液晶屏

STN 是“Super Teisted Nematic”的缩写，它属于无源被动矩阵式 LCD，几乎所有黑白屏手机的液晶屏都是这种材料。彩色 STN 液晶屏就是在单色的 STN 液晶屏基础上加个彩色滤光片，并将单色显示矩阵中的每个像素分成三个子像素，分别通过彩色滤光片显示红、绿、蓝三种颜色，从而实现彩色画面。由于技术的限制，目前 STN 液晶屏最高只有 65536 种色彩，市场上见到的大多数都是 4096 色的 STN 产品，所以 STN 也被称为“伪彩”。

2. GF 液晶屏

GF 是“Glass Fine Color”的缩写，或许大家对 GF 液晶屏较为陌生，因为现在市面上采用 GF 液晶屏数码产品非常少，其实 GF 属于 STN 的一种，

GF 的主要特点是：在保证功耗较小的前提下亮度有所提高，但 GF 液晶屏有些偏色。

3. TFT 液晶屏

TFT 是“Thin Film Transistor”的缩写，又称为“真彩”，它属于有源矩阵液晶屏，它是由薄膜晶体管组成的屏幕，它的每个液晶像素点都是由薄膜晶体管来驱动，每个像素点后面都有四个相互独立的薄膜晶体管驱动像素点发出彩色光，可显示 24bit 色深的真彩色。在分辨率上，TFT 液晶屏最大可以达到 UXGA（1600×1200）。

TFT 的排列方式具有记忆性，所以电流消失后不会马上恢复原状，从而改善了 STN 液晶屏闪烁和模糊的缺点，有效地提高了液晶屏显示动态画面的效果，在显示静态画面方面的能力也更加突出，TFT 液晶屏的优点是响应时间比效短，并且色彩艳丽，所以它被广泛使用于笔记本电脑和 DV、DC 上。而 TFT 液晶屏的缺点就是比较耗电，并且成本也比较高。

4. TFD 液晶屏

TFD 是“Thin Film Diode”的缩写，由于 TFT 液晶屏耗电量较高，而且成本较高，从而大大增加了产品的成本，所以 EPSON 专门为手机屏幕开发出了 TFD 技术，它同样属于有源矩阵液晶屏，LCD 上的每一个像素都配备了一颗单独的二极管，可以对每个像素进行单独控制，使每个像素之间不会互相影响，这样可以明显提高分辨率，可以无拖尾地显示动态画面和绚丽的色彩。

在性能方面，TFD 液晶屏兼顾了 TFT 液晶屏和 STN 液晶屏的优点，TFD 液晶屏比 STN 液晶屏的亮度更高，并且色彩也更鲜艳，同时比 TFT 液晶屏更省电，不过在色彩和亮度上还是比 TFT 液晶逊色一些。

5. OLED 液晶屏

OLED 是“Organic Light Emitting Display”的缩写，也称有机发光显示屏，它采用了有机发光技术，这是目前最新的显示技术，OLED 显示技术与传统的液晶显示方式不同，它不需要背光灯，而是采用非常薄的有机材料涂层和玻璃基板，当有电流通过时，这些有机材料就会发光，所以它

的视角很大，从各个方向都可以看清楚屏幕上的内容，并且可以做得很薄，而且 OLED 显示屏能够显著节省电能，被誉为“梦幻显示器”。

严格来讲，OLED 不是液晶屏，是 LED 的技术。 液晶屏的技术核心在日本；而 OLED 的技术核心在美国，并且现在已经大规模在韩国批量生产。就使用在韩国三星的最选进的手机上，售价奇高，甚至比一台智能手机的价格都高。

但是 OLED 也并非没有缺点，由于它还属于一种未成熟的技术，所以现阶段它的使用寿命还比较短，屏幕面积也比较小。

相关参数

1. 分辨率

分辨率是一个非常重要的性能指标。它指的是屏幕上水平和垂直方向所能够显示的点数（屏幕上显示的线和面都是由点构成的）的多少，分辨率越高，同一屏幕内能够容纳的信息就越多。对于一台能够支持 1280x1024 分辨率的 CRT 来说，无论是 320x240 还是 1280x1024 分辨率，都能够比较完美地表现出来（因为电子束可以做弹性调整）。但它的最大分辨率未必是最合适的分辨率，因为如果 17 寸显示器上到 1280x1024 分辨率的话，WINDOWS 的字体会很小，时间一长眼睛就容易疲劳，所以 17 寸显示器的最佳分辨率应为 1024x768。

但对 LCD 来说则不然。LCD 的最大分辨率就是它的真实分辨率，也就是最佳分辨率。一旦所设定的分辨率小于真实分辨率（比如说 15 寸 LCD，其真实分辨率为 1024x768，而 WINDOWS 中设定分辨率为 800x600）的话，将有两种显示方式。一是居中显示，只有 LCD 中间的 800x600 个点会显示图象，其他没有用到的点不会发光，保持黑暗背景，看起来画面是居中缩小的。另一种是扩展显示，这种方式会使用到屏幕上每一个像素，但由于像素很容易发生扭曲，所以会对显示效果造成一定影响。所以说无论如何在选择 LCD 时要注意分辨率不是越大越好而是适当好用。

2. 视角

目前大多数纯平显示器的视角都能达到 180 度，也就是说，从屏幕前的任意一个方向都能清楚地看到所显示的内容。而 LCD 则不同，它的可视

角度根据工艺先进与否而有所不同。市场上一线品牌，如华硕、三星、LG 等产品的可是角度大部分都能达到 170 度这一水平，而部分采用广视角的显示器则能够达到 178 度，跟 CRT 的 180 度已经非常接近。用户在使用过程中一旦视角超出其实际可视范围，画面的颜色就会减退、变暗，甚至出现正像变成负像的情况。很可能大家为飞利浦的广告所迷惑其实 LCD 的视角并不是很大，反而比 CRT 的小许多，是一个明显比 CRT 弱的地方。

3. 可视面积

可视面积指的是在实际应用中，可以用来显示图像的那部分屏幕的面积。因为 CRT 显示器的尺寸实际上是其显像管的尺寸，可以用来显示图像的部分根本达不到这个尺寸，因为显像管的边框占了一部分空间。一般来讲，17 寸 CRT 显示器的可视面积约在 15.8-16 英寸左右，而 15 寸显示器的可视面积则只有 13.8 英寸左右。但对于 LCD 来说，标称的尺寸大小基本上就是可视面积的大小，被边框占用的空间非常小，15 寸 LCD 的可视面积大约有 14.5 英寸左右，这也是为什么 LCD 看起来要比同样尺寸 CRT 更大一些的原因。所以选购 LCD 的时候 15 英寸就基本上够了.

4. 亮度与对比度

液晶显示器的显示功能主要是有一个背光的光源，这个光源的亮度决定整台 LCD 的画面亮度及色彩的饱和度。理论上来说，液晶显示器的亮度是越高越好，亮度的测量单位为 cd/m2（每公尺平方烛光），也叫 NIT 流明。目前 TFT 屏幕的亮度大部分都是从 150Nits 开始起步，通常情况下 200Nits 才能表现出比较好的画面。对比度也就是黑与白两种色彩不同层次的对比测量度。对比度 120:1 时就可以显示生动、丰富的色彩（因为人眼可分辨的对比度约在 100:1 左右),对比率达到 300:1 时便可以支持各阶度的颜色。目前大多数 LCD 显示器的对比度都在 500:1～800:1 左右。而如华硕、三星、LG 等一线品牌的液晶显示器产品对比度则普遍达到了 1000:1 左右。目前还没有一套公正的标准值来衡量亮度与对比的反差值，所以购买 LCD 全靠一双锐利的眼睛。所以在选购 LCD 时要注意这个指标，它也是 LCD 产品上性能差异最大的一环估计选购上有些难度。

5. 反应速度

测量反应速度的时间单位是毫秒（ms），指的是象素由亮转暗并由暗转亮所需的时间。这个数值越小越好，数值越小，说明反应速度越快。目前主流 LCD 的反应速度都在 25ms 以上，在一般商业用途中（例如字处理或文本处理）没有什么太大关系，因为此类用途不必太在意 LCD 的反应时间。而如果是用来玩游戏、观看 VCD/DVD 等全屏高速动态影象时，反应时间就尤其重要了，如果反应时间较长的话，画面就会出现拖尾、残影等现象。举个简单的例子，现在市场上绝大多数 LCD 显示器在玩 QUAKE3 时都会有不同程度的拖尾现象，在画面高速更新时尤其明显。而 CRT 则完全没有这个问题，因为 CRT 的反应时间只有 1ms，是绝对不会出现拖尾现象的。

6. 色彩

说到色彩，LCD 也比不上 CRT，从理论上讲，CRT 可显示的色彩跟电视机一样为无限。而 LCD 只能显示大约 26 万种颜色，绝大部分产品都宣称能够显示 1677 万色（16777216 色，32 位），但实际上都是通过抖动算法（dithering）来实现的，与真正的 32 位色相比还是有很大差距，所以在色彩的表现力和过渡方面仍然不及传统 CRT。同样的道理，LCD 在表现灰度方面的能力也不如 CRT。大家有条件的话可以自己比较一下：找一台 17 英寸特丽珑显像管的显示器，再摆一台 15 寸 LCD，同时显示一幅 1677 万色的图象。CRT 显示出来的画面十分鲜艳，而 LCD 则显得有些"假"，虽然说不上来哪里不对，但看着就是没有那台珑管的 CRT 舒服。

液晶面板主要由以下八大部分组成：

1. 背光源（或背光模组）；

由于液晶分子自身是无法发光的，因此若想出现画面，液晶屏需要专门的发光源来提供光线，然后经过液晶分子的偏转来产生不同的颜色。而背光源起到的就是提供光能的作用。之前液晶屏采用的都是名叫 CCFL 的冷阴极射线管，其发光原理与日光灯几乎完全相同，而现在新品液晶屏都采用了更加节能、长寿面的 LED 背光源。灯管（或 LED）发光后藉由导光板将光线分布到各处，通过背面的反射板将所有的光线的方向集中朝向液晶分子。最后光线通过 prism sheet 以及扩散板将光线均匀的散发出去，避免出现中央亮度过高、四周亮度过低的情况。

2. 上下层两个偏光片；

偏光片的作用是让光线从单方向通过。

3. 上层和下层两块玻璃基板；

玻璃基板不仅仅是两块玻璃那么简单，其内侧具有沟槽结构，并附着配向膜，可以让液晶分子沿着沟槽整齐的排列。在上、下两层玻璃两侧会贴有 TFT 薄膜晶体管和彩色滤光片。

4. ITO 透明导电层；

其作用是提供导电通路，分为像素电极（P 级）和公共电极（M 级）。在下一页中我们为大家讲解液晶面板结构更多的内容。

下面我们来详细看看另外四组主要的结构。

5. 薄膜晶体管（就是我们经常所说的 TFT）；

我们经常说 TFT-LCD，其实际上指的就是这个薄膜晶体管，它的作用类似于开关，TFT 能够控制 IC 控制电路上的信号电压，并将其输送到液晶分子中，决定液晶分子偏转的角度大小，因此其是非常重要的一个部件。

6. 液晶分子层；

这个不用过多解释，其是改变光线偏光状态最重要的元素，通过电力和弹性力共同决定其排列和偏光状态。

7. 彩色滤光片；

通过液晶分子偏转的光线只能显示不同的灰阶，但是不能提供红、绿、蓝（RGB）三原色，而彩色滤光片则由 RGB 三种过滤片组成，通过三者混喝调节各个颜色与亮度。液晶面板中每一个像素由红、绿、蓝 3 个点构成，每种颜色的点各自拥有不同的灰阶变化。

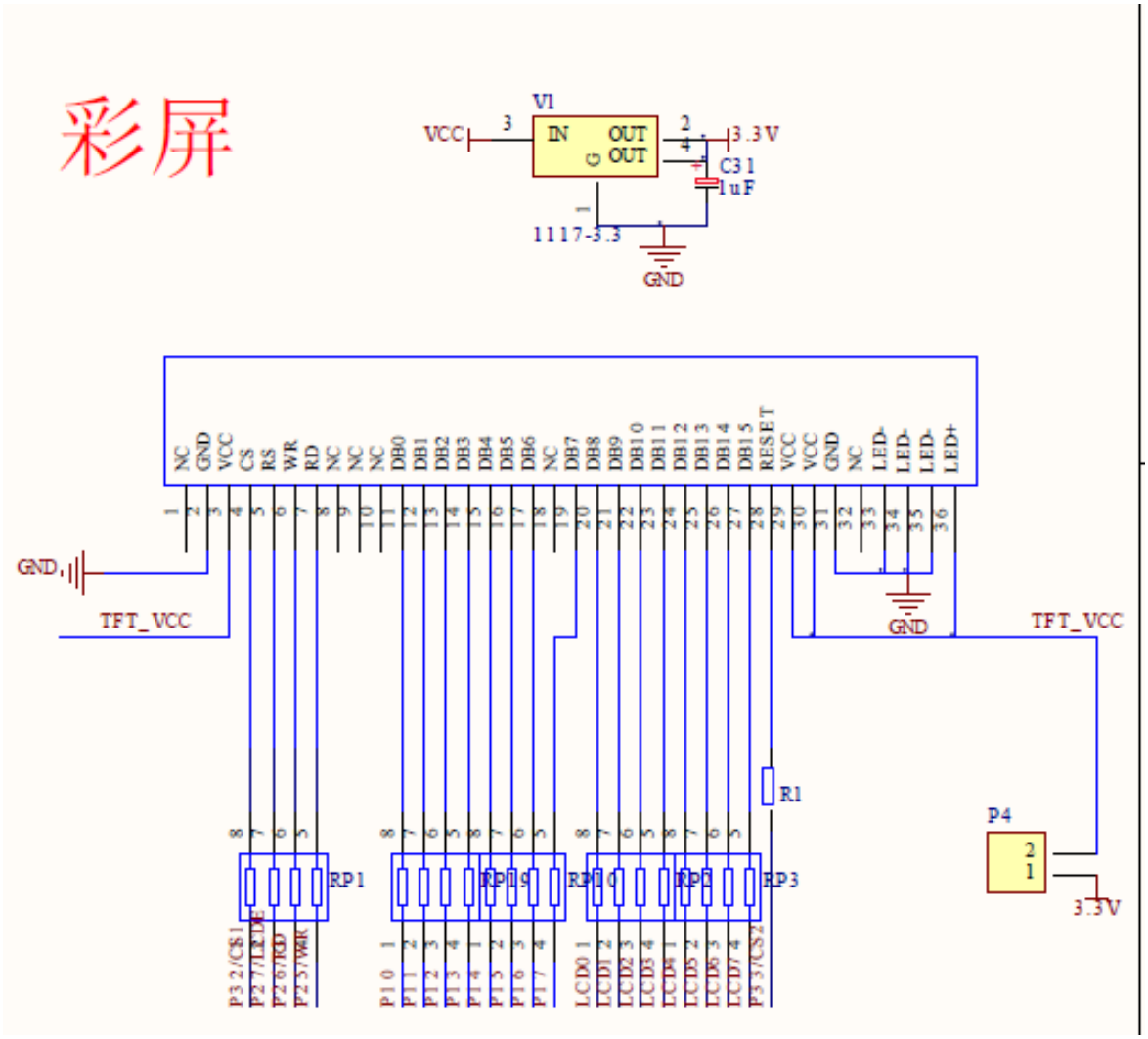
8. 框胶；

相信大家都能够猜到这个部件的作用，其就是让液晶面板中上下两层玻璃基板能够牢固的黏在一起，并将整个内部系统与外接“隔绝”，防止灰尘进入影响色彩效果。

当然，上面所描述的结构为市面上大多数液晶电视和显示器所使用面板的结构，一些特殊的产品可能会使用不同的配件，但整体架构和工作原理基本不会有太大差异。而我们通常所说的“LED 背光”指的则是其在第一

部分背光源所做出的改进，将 CCFL 冷阴极射线管更换成了 LED，而其他部分几乎没有任何变化（外部的供电部分会有相应小幅度的调节）。此外液晶屏色域的大小，也主要由液晶屏背光源来决定。

首先我们看一下开发板的电路原理图



附录 A 单片机 C 语言介绍

单片机 C 语言设计指导

在实际工程应用中，51 单片机的程序设计一般都是采用 C 语言编写，通过相应的编译器，得到可执行代码，以提高程序开发效率。但由于单片机的内部资源有限，并结合单片机的特点（如位操作），与标准 C 语言相比，有所不同，称为 C51 程序。

C 语言的特点

- ✧ 语言简洁、紧凑，使用方便、灵活。
- ✧ 运算符丰富。
- ✧ 数据结构丰富。具有现代化语言的各种数据结构。
- ✧ 可进行结构化程序设计。
- ✧ 可以直接对计算机硬件进行操作。
- ✧ 生成的目标代码质量高，程序执行效率高。
- ✧ 可移植性好。

C 语言程序采用函数结构，每个 C 语言程序由一个或多个函数组成，在这些函数中至少应包含一个主函数 main()，也可以包含一个 main()函数和若干个其它的功能函数。不管 main()函数放于何处，程序总是从 main()函数开始执行，执行到 main()函数结束则结束。在 main()函数中调用其它函数，其它函数也可以相互调用，但 main()函数只能调用其它的功能函数，而不能被其它的函数所调用。

功能函数可以是 C 语言编译器提供的库函数，也可以是由用户定义的自定义函数。在编制 C 程序时，程序的开始部分一般是预处理命令、函数说明和变量定义等。

用 C 语言编写 51 单片机程序与用汇编语言编写 51 单片机程序不同，汇编语言必须要考虑其存储器结构，尤其必须考虑其片内数据存储器与特殊功能寄存器的使用以及按实际地址处理端口数据。

用 C 语言编写的 51 单片机应用程序，则不用像汇编语言那样须具体组织、分配存储器资源和处理端口数据，但在 C 语言编程中，对数据类型与变量的定义，必须要与单片机的存储结构相关联，否则编译器不能正确地映射定位。

用 C 语言编写单片机应用程序与标准的 C 语言程序也有相应的区别：C 语言编写单片机应用程序时，需根据单片机存储结构及内部资源定义相应的数据类型和变量，而标准的 C 语言程序不需要考虑这些问题。

C51 包含的数据类型、变量存储模式、输入输出处理、函数等方面与标准的 C 语言有一定的区别。其它的语法规则、程序结构及程序设计方法等与标准的 C 语言程序设计相同。

现在支持 51 系列单片机的 C 语言编译器有很多种，如 American Automation、Avocet、BSO/TASKING、DUNFIELD SHAREWARE、KEIL/Franklin 等。各种编译器的基本情况相同，但具体处理时有一定的区别，其中 KEIL/Franklin 以它的代码紧凑和使用方便等特点优于其它编译器，使用特别广泛。

本章主要以 KEIL 编译器介绍 51 单片机 C 语言程序设计。

C51 程序结构

C51 的语法规定、程序结构及程序设计方法都与标准的 C 语言程序设计相同，但 C51 程序与标准的 C 程序在以下几个方面不一样：

（1） C51 中定义的库函数和标准 C 语言定义的库函数不同。标准的 C 语言定义的库函数是按通用微型计算机来定义的，而 C51 中的库函数是按 51 单片机相应情况来定义的；

（2） C51 中的数据类型与标准 C 的数据类型也有一定的区别，在 C51 中还增加了几种针对 51 单片机特有的数据类型；

（3） C51 变量的存储模式与标准 C 中变量的存储模式不一样，C51 中变量的存储模式是与 51 单片机的存储器紧密相关；

（4） C51 与标准 C 的输入输出处理不一样，C51 中的输入输出是通过 51 串行口来完成的，输入输出指令执行前必须要对串行口进行初始化；

（5） C51 与标准 C 在函数使用方面也有一定的区别，C51 中有专门的中断函数。

C51 的数据类型

C51 的数据类型分为基本数据类型和组合数据类型，情况与标准 C 中的数据类型基本相同，但其中 char 型与 short 型相同，float 型与 double 型相同，另外，C51 中还有专门针对于 51 单片机的特殊功能寄存器型和位类型。

2. 字符型 char

有 signed char 和 unsigned char 之分，默认为 signed char。它们的长度均为一个字节，用于存放一个单字节的数据。

对于 signed char，它用于定义带符号字节数据，其字节的最高位为符号位，“0”表示正数，“1”表示负数，补码表示，所能表示的数值范围是-128~+127；

对于 unsigned char，它用于定义无符号字节数据或字符，可以存放一个字节的无符号数，其取值范围为 0~255。unsigned char 可以用来存放无符号数，也可以存放西文字符，一个西文字符占一个字节，在计算机内部用 ASCII 码存放。

3. int 整型

分 singed int 和 unsigned int。默认为 signed int。它们的长度均为两个字节，用于存放一个双字节数据。对于 signed int，用于存放两字节带符号数，补码表示，数的范畴为-32768~+32767。对于 unsigned int，用于存放两字节无符号数，数的范围为 0~65535。

4. long 长整型

分 singed long 和 unsigned long。默认为 signed long。它们的长度均为四个字节，用于存放一个四字节数据。对于 signed long，用于存放四字节带符号数，补码表示，数的范畴为-2147483648~+2147483647。对于 unsigned long，用于存放四字节无符号数，数的范围为 0~4294967295。

5. float 浮点型

float 型数据的长度为四个字节，格式符合 IEEE-754 标准的单精度浮点型数据，包含指数和尾数两部分，最高位为符号位，“1”表示负数，“0”表示正数，其次的 8 位为阶码，最后的 23 位为尾数的有效数位，由于尾数的整数部分隐含为“1”，所以尾数的精度为 24 位。

6. * 指针型

指针型本身就是一个变量，在这个变量中存放的指向另一个数据的地址。这个指针变量要占用一定的内存单元，对不同的处理器其长度不一样，在 C51 中它

的长度一般为 1~3 个字节。

7. 特殊功能寄存器型

这是 C51 扩充的数据类型，用于访问 51 单片机中的特殊功能寄存器数据，它分 sfr 和 sfr16 两种类型。其中：

sfr 为字节型特殊功能寄存器类型，占一个内存单元，利用它可以访问 51 内部的所有特殊功能寄存器；

sfr16 为双字节型特殊功能寄存器类型，占用两个字节单元，利用它可以访问 51 内部的所有两个字节的特殊功能寄存器。

在 C51 中对特殊功能寄存器的访问必须先用 sfr 或 sfr16 进行声明。

8. 位类型

这也是 C51 中扩充的数据类型，用于访问 51 单片机中的可寻址的位单元。在 C51 中，支持两种位类型：bit 型和 sbit 型。它们在内存中都只占一个二进制位，其值可以是“1”或“0”。

其中：用 bit 定义的位变量在 C51 编译器编译时，在不同的时候位地址是可以变化的，而用 sbit 定义的位变量必须与 51 单片机的一个可以寻址位单元或可位寻址的字节单元中的某一位联系在一起，在 C51 编译器编译时，其对应的位地址是不可变化的。

KEIL C51 编译器能够识别的基本数据类型：

基本数据类型	长度	取值范围
unsigned char	1 字节	0~255
signed char	1 字节	-128~+127
unsigned int	2 字节	0~65535
signed int	2 字节	-32768~+32767
unsigned long	4 字节	0~4294967295
signed long	4 字节	-2147483648~+2147483647
float	4 字节	±1.175494E-38~±3.402823E+38
bit	1 位	0 或 1
sbit	1 位	0 或 1
sfr	1 字节	0~255
sfr16	2 字节	0~65535

在 C51 语言程序中，有可能会出现在运算中数据类型不一致的情况。C51 允许任何标准数据类型的隐式转换，隐式转换的优先级顺序如下：

Bit→char→int→long→float→signed→unsigned

也就是说，当 char 型与 int 型进行运算时，先自动对 char 型扩展为 int 型，然后与 int 型进行运算，运算结果为 int 型。C51 除了支持隐式类型转换外，还可以通过强制类型转换符“()”对数据类型进行人为的强制转换。

C51 编译器除了能支持以上这些基本数据类型之外，还能支持一些复杂的组合型数据类型，如数组类型、指针类型、结构类型、联合类型等这些复杂的数据类型，在后面将相继介绍。

C51 的运算量

A. 常量

常量是指在程序执行过程中其值不能改变的量。在 C51 中支持整型常量、浮点型常量、字符型常量和字符串型常量。

一. 整型常量

整型常量也就是整型常数，根据其值范围在计算机中分配不同的字节数来存放。在 C51 中它可以表示成以下几种形式：

十进制整数。如 234、-56、0 等。

十六进制整数。以 0x 开头表示，如 0x12 表示十六进制数 12H。

长整数。在 C51 中当一个整数的值达到长整型的范围，则该数按长整型存放，在存储器中占四个字节，另外，如一个整数后面加一个字母 L，这个数在存储器中也按长整型存放。如 123L 在存储器中占四个字节。

二. 浮点型常量

浮点型常量也就是实型常数。有十进制表示形式和指数表示形式。

十进制表示形式又称定点表示形式，由数字和小数点组成。如 0.123、34.645 等都是十进制数表示形式的浮点型常量。

指数表示形式为： [ ] 数字 [.数字] e [ ]数字

例如：123.456e-3、-3.123e2 等都是指数形式的浮点型常量。

三. 字符型常量

字符型常量是用单引号引起的字符，如‘a’、‘1’、‘F’等。可以是可显示的 ASCII 字符，也可以是不可显示的控制字符。对不可显示的控制字符须在前面加上反斜杠“\”组成转义字符。利用它可以完成一些特殊功能和输出时的格式控制。常用的转义字符如下表所示。

转义字符	含 义	ASCII 码（十六进制数）
\ o	空字符（null）	OOH
\ n	换行符（LF）	OAH
\ r	回车符（CR）	ODH
\ t	水平制表符（HT）	O9H
\ b	退格符（BS）	O8H
\ f	换页符（FF）	OCH
\ ‘	单引号	27H
\ ”	双引号	22H
\ \	反斜杠	5CH

四. 字符串型常量

字符串型常量由双引号“”括起的字符组成。如“D”、“1234”、“ABCD”等。注意字符串常量与字符常量是不一样，一个字符常量在计算机内只用一个字节存放，而一个字符串常量在内存中存放时不仅双引号内的字符一个占一个字节，而且系统会自动的在后面加一个转义字符“\o”作为字符串结束符。因此不要将字符常量和字符串常量混淆，如字符常量‘A’和字符串常量“A”是不一样的。

B. 变量

变量是在程序运行过程中其值可以改变的量。一个变量由两部分组成：变量名和变量值。

在 C51 中，变量在使用前必须对变量进行定义，指出变量的数据类型和存储模式。以便编译系统为它分配相应的存储单元。定义的格式如下：

[存储种类] 数据类型说明符 [存储器类型] 变量名 1[=初值]，变量名 2[初值]…;

一. 数据类型说明符

在定义变量时，必须通过数据类型说明符指明变量的数据类型，指明变量在存储器中占用的字节数。可以是基本数据类型说明符，也可以是组合数据类型说明符，还可以是用 typedef 定义的类型别名。

在 C51 中，为了增加程序的可读性，允许用户为系统固有的数据类型说明符用 typedef 起别名，格式如下：

typedef c51 固有的数据类型说明符 别名;

定义别名后，就可以用别名代替数据类型说明符对变量进行定义。别名可以用大写，也可以用小写，为了区别一般用大写字母表示。

【例】 typedef 的使用。

```
typedef  unsigned  int   WORD;
typedef  unsigned  char  BYTE;
BYTE  a1=0x12;
WORD  a2=0x1234;
```

二. 变量名

变量名是 C51 区分不同变量，为不同变量取的名称。在 C51 中规定变量名可以由字母、数字和下划线三种字符组成，且第一个字母必须为字母或下划线。变量名有两种：普通变量名和指针变量名。它们的区别是指针变量名前面要带“*”号。

三. 存储种类

存储种类是指变量在程序执行过程中的作用范围。C51 变量的存储种类有四种，分别是自动(auto)、外部(extern)、静态(static)和寄存器(register)。

1. auto:

使用 auto 定义的变量称为自动变量，其作用范围在定义它的函数体或复合语句内部，当定义它的函数体或复合语句执行时，C51 才为该变量分配内存空间，结束时占用的内存空间释放。自动变量一般分配在内存的堆栈空间中。定义变量时，如果省略存储种类，则该变量默认为自动(auto)变量。

2. extern：

使用 extern 定义的变量称为外部变量。在一个函数体内，要使用一个已在该函数体外或别的程序中定义过的外部变量时，该变量在该函数体内要用 extern 说明。外部变量被定义后分配固定的内存空间，在程序整个执行时间内都有效，直到程序结束才释放。

3. static：

使用 static 定义的变量称为静态变量。它又分为内部静态变量和外部静态变量。在函数体内部定义的静态变量为内部静态变量，它在对应的函数体内有效，一直存在，但在函数体外不可见，这样不仅使变量在定义它的函数体外被保护，还可以实现当离开函数时值不被改变。外部静态变量上在函数外部定义的静态变量。它在程序中一直存在，但在定义的范围之外是不可见的。如在多文件或多模块处理中，外部静态变量只在文件内部或模块内部有效。

4. register：

使用 register 定义的变量称为寄存器变量。它定义的变量存放在 CPU 内部的寄存器中，处理速度快，但数目少。C51 编译器编译时能自动识别程序中使用频率最高的变量，并自动将其作为寄存器变量，用户可以无需专门声明。

四. 存储器类型

存储器类型是用于指明变量所处的单片机的存储器区域情况。存储器类型与存储种类完全不同。C51 编译器能识别的存储器类型有以下几种，见表所示。

存储器类型	描　述
data	直接寻址的片内 RAM 低 128B，访问速度快
bdata	片内 RAM 的可位寻址区（20H~2FH），允许字节和位混合访问
idata	间接寻址访问的片内 RAM，允许访问全部片内 RAM
pdata	用 Ri 间接访问的片外 RAM 的低 256B
xdata	用 DPTR 间接访问的片外 RAM，允许访问全部 64k 片外 RAM
code	程序存储器 ROM 64k 空间

定义变量时也可以省“存储器类型”，省时 C51 编译器将按编译模式默认存储器类型，具体编译模式的情况在后面介绍。

【例】变量定义存储种类和存储器类型相关情况。

char data var1; /*在片内 RAM 低 128B 定义用直接寻址方式访问的字符型变量 var1*/

int idata var2; /*在片内 RAM256B 定义用间接寻址方式访问的整型变量 var2*/

auto unsigned long data var3; /*在片内 RAM128B 定义用直接寻址方式访问的自动无符号长整型变量 var3*/

extern float xdata var4; /*在片外 RAM64KB 空间定义用间接寻址方式访问的外部实型变量 var4*/

int code var5; /*在 ROM 空间定义整型变量 var5*/

unsign char bdata var6; /*在片内 RAM 位寻址区 20H~2FH 单元定义可字节处理和位处理的无符号字符型变量 var6*/

五. 特殊功能寄存器变量

51 系列单片机片内有许多特殊功能寄存器，通过这些特殊功能寄存器可以控制 51 系列单片机的定时器、计数器、串口、I/O 及其它功能部件，每一个特殊功能寄存器在片内 RAM 中都对应于一个字节单元或两个字节单元。

在 C51 中,允许用户对这些特殊功能寄存器进行访问,访问时须通过 sfr 或 sfr16 类型说明符进行定义，定义时须指明它们所对应的片内 RAM 单元的地址。格式如下：

fr 或 sfr16 特殊功能寄存器名=地址；

sfr 用于对 51 单片机中单字节的特殊功能寄存器进行定义，sfr16 用于对双字节特殊功能寄存器进行定义。特殊功能寄存器名一般用大写字母表示。地址一般用直接地址形式，具体特殊功能寄存器地址见前面内容。

【例】特殊功能寄存器的定义。

```
sfr  PSW=0xd0;
sfr  SCON=0x98;
sfr  TMOD=0x89;
sfr  P1=0x90;
sfr16  DPTR=0x82;
sfr16  T1=0X8A;
```

六. 位变量

在 C51 中，允许用户通过位类型符定义位变量。位类型符有两个：bit 和 sbit。可以定义两种位变量。

bit 位类型符用于定义一般的可位处理位变量。它的格式如下：

bit　位变量名；

在格式中可以加上各种修饰，但注意存储器类型只能是 bdata、data、idata。只能是片内 RAM 的可位寻址区，严格来说只能是 bdata。

【例】 bit 型变量的定义。

```
bit  data  a1;        /*正确*/
bit  bdata  a2;      /*正确*/
bit  pdata  a3;      /*错误*/
bit  xdata  a4;      /*错误*/
```

sbit 位类型符用于定义在可位寻址字节或特殊功能寄存器中的位，定义时须指明其位地址，可以是位直接地址，可以是可位寻址变量带位号，也可以是特殊功能寄存器名带位号。格式如下：

sbit　位变量名=位地址；

如位地址为位直接地址，其取值范围为 0x00~0xff；如位地址是可位寻址变量带位号或特殊功能寄存器名带位号，则在它前面须对可位寻址变量或特殊功能寄存器进行定义。字节地址与位号之间、特殊功能寄存器与位号之间一般用“^”作间隔。

【例】sbit 型变量的定义：

```
sbit  OV=0xd2;
sbit  CY=oxd7;
unsigned  char  bdata  flag;
sbit  flag0=flag^0;
sfr  P1=0x90;
sbit  P1_0=P1^0;
sbit  P1_1=P1^1;
sbit  P1_2=P1^2;
sbit  P1_3=P1^3;
sbit  P1_4=P1^4;
sbit  P1_5=P1^5;
sbit  P1_6=P1^6;
sbit  P1_7=P1^7;
```

在 C51 中，为了用户处理方便，C51 编译器把 51 单片机的常用的特殊功能寄存器和特殊位进行了定义，放在一个“reg51.h”或“reg52.h”的头文件中，当

用户要使用时，只须要在使用之前用一条预处理命令#include <reg52.h>把这个头文件包含到程序中，然后就可使用殊功能寄存器名和特殊位名称。

C. 存储模式

C51 编译器支持三种存储模式：SMALL 模式、COMPACT 模式和 LARGE 模式。不同的存储模式对变量默认的存储器类型不一样。

（1） SMALL 模式。SMALL 模式称为小编译模式，在 SMALL 模式下，编译时，函数参数和变量被默认在片内 RAM 中，存储器类型为 data。

（2） 2）COMPACT 模式。COMPACT 模式称为紧凑编译模式，在 COMPACT 模式下，编译时，函数参数和变量被默认在片外 RAM 的低 256 字节空间，存储器类型为 pdata。

（3） LARGE 模式。LARGE 模式称为大编译模式，在 LARGE 模式下，编译时函数参数和变量被默认在片外 RAM 的 64K 字节空间，存储器类型为 xdata。

在程序中变量的存储模式的指定通过#pragma 预处理命令来实现。函数的存储模式可通过在函数定义时后面带存储模式说明。如果没有指定，则系统都隐含为 SMALL 模式。

【例】变量的存储模式。

```
#pragma  small                    /*变量的存储模式为 SMALL*/
char  k1;
int  xdata  m1;
#pragma  compact              /*变量的存储模式为 SMALL*/
char  k2;
int  xdata  m2;
int  func1(int  x1,int  y1)  large  /*函数的存储模式为 LARGE*/
{   return(x1+y1);}
int  func2(int  x2,int  y2)       /*函数的存储模式隐含为 SMALL*/
{   return(x2-y2);}
```

程序编译时，k1 变量存储器类型为 data，k2 变量存储器类型为 pdata，而 m1 和 m2 由于定义时带了存储器类型 xdata，因而它们为 xdata 型；函数 func1 的形参 x1 和 y1 的存储器类型为 xdata 型，而函数 func2 由于没有指明存储模式，隐含为 SMALL 模式，形参 x2 和 y2 的存储器类型为 data。

D. 绝对地址的访问

一. 使用 C51 运行库中预定义宏

51编译器提供了一组宏定义来对51系列单片机的code、data、pdata和xdata空间进行绝对寻址。规定只能以无符号数方式访问，定义了 8 个宏定义，其函数原型如下：

```
#define  CBYTE((unsigned char volatile*)0x50000L)
#define  DBYTE((unsigned char volatile*)0x40000L)
#define  PBYTE((unsigned char volatile*)0x30000L)
#define  XBYTE((unsigned char volatile*)0x20000L)
#define  CWORD((unsigned int volatile*)0x50000L)
#define  DWORD((unsigned int volatile*)0x40000L)
#define  PWORD((unsigned int volatile*)0x30000L)
#define  XWORD((unsigned int volatile*)0x20000L)
```

这些函数原型放在 absacc.h 文件中。使用时须用预处理命令把该头文件包含到文件中，形式为：#include <absacc.h>。

其中：CBYTE 以字节形式对 code 区寻址，DBYTE 以字节形式对 data 区寻址，PBYTE 以字节形式对 pdata 区寻址，XBYTE 以字节形式对 xdata 区寻址，CWORD 以字形式对code区寻址,DWORD以字形式对data区寻址,PWORD以字形式对pdata区寻址，XWORD 以字形式对 xdata 区寻址。

【例】绝对地址对存储单元的访问

```
#include  <absacc.h>            /*将绝对地址头文件包含在文件中*/
#include  <reg52.h>                 /*将寄存器头文件包含在文件中*/
#define  uchar  unsigned  char      /*定义符号 uchar 为数据类型符 unsigned
char*/
#define  uint  unsigned  int        /*定义符号 uint 为数据类型符 unsigned
int*/
void  main(void)
{   uchar  var1;
    uint  var2;
var1=XBYTE[0x0005];             /*XBYTE[0x0005]访问片外RAM的0005字节单元*/
var2=XWORD[0x0002];          /*XWORD[0x0002]访问片外 RAM 的 000 字单元*/
......
while(1);
}
```

在上面程序中，其中 XBYTE[0x0005]就是以绝对地址方式访问的片外 RAM 0005 字节单元；XWORD[0x0002]就是以绝对地址方式访问的片外 RAM 0002 字单

元。

二．通过指针访问

采用指针的方法，可以实现在 C51 程序中对任意指定的存储器单元进行访问。

【例】 通过指针实现绝对地址的访问。

```
#define  uchar  unsigned char          /*定义符号 uchar 为数据类型符
unsigned char*/
#define  uint  unsigned int            /*定义符号uint为数据类型符unsigned
int*/
void  func(void)
{
uchar  data  var1;
uchar  pdata  *dp1;              /*定义一个指向 pdata 区的指针 dp1*/
uint     xdata  *dp2;          /*定义一个指向 xdata 区的指针 dp2*/
uchar  data    *dp3;          /*定义一个指向 data 区的指针 dp3*/
dp1=0x30;                     /*dp1 指针赋值，指向 pdata 区的 30H 单元*/
dp2=0x1000;                   /*dp2 指针赋值，指向 xdata 区的 1000H 单元*/
*dp1=0xff;                        /*将数据 0xff 送到片外 RAM30H 单元*/
*dp2=0x1234;                  /*将数据 0x1234 送到片外 RAM1000H 单元*/
dp3=&var1;                    /*dp3 指针指向 data 区的 var1 变量*/
*dp3=0x20;                    /*给变量 var1 赋值 0x20*/
}
```

三．使用 C51 扩展关键字_at_

使用_at_对指定的存储器空间的绝对地址进行访问，一般格式如下：

[存储器类型] 数据类型说明符 变量名 _at_ 地址常数；

其中，存储器类型为 data、bdata、idata、pdata 等 C51 能识别的数据类型，如省略则按存储模式规定的默认存储器类型确定变量的存储器区域；数据类型为 C51 支持的数据类型。地址常数用于指定变量的绝对地址，必须位于有效的存储器空间之内；使用_at_定义的变量必须为全局变量。

【例】通过_at_实现绝对地址的访问。

```
#define  uchar  unsigned char          /*定义符号 uchar 为数据类型符
unsigned char*/
#define  uint  unsigned int            /*定义符号uint为数据类型符unsigned
int*/
void  main(void)
{
data  uchar  x1  _at_  0x40;          /*在data区中定义字节变量x1,它的地址为
40H*/
xdata  uint  x2  _at_  0x2000;         /*在xdata区中定义字变量x2,它的地址为
2000H*/
```

```
x1=0xff;
x2=0x1234;
......
while(1);
}
```

E. C51 的运算符及表达式

1. 赋值运算符

赋值运算符“=”，在 C51 中，它的功能是将一个数据的值赋给一个变量，如 x=10。利用赋值运算符将一个变量与一个表达式连接起来的式子称为赋值表达式，在赋值表达式的后面加一个分号“;”就构成了赋值语句，一个赋值语句的格式如下：

变量=表达式;

执行时先计算出右边表达式的值，然后赋给左边的变量。例如：

```
x=8+9;      /*将 8+9 的值赋给变量 x*/
x=y=5;      /*将常数 5 同时赋给变量 x 和 y*/
```

在 C51 中，允许在一个语句中同时给多个变量赋值，赋值顺序自右向左。

2. 算术运算符

C51 中支持的算术运算符有：

+ 加或取正值运算符

- 减或取负值运算符

* 乘运算符

/ 除运算符

% 取余运算符

加、减、乘运算相对比较简单，而对于除运算，如相除的两个数为浮点数，则运算的结果也为浮点数，如相除的两个数为整数，则运算的结果也为整数，即为整除。如 25.0/20.0 结果为 1.25，而 25/20 结果为 1。

对于取余运算，则要求参加运算的两个数必须为整数，运算结果为它们的余

数。例如：x=5%3，结果 x 的值为 2。

3. 关系运算符

C51 中有 6 种关系运算符：

> 大于

< 小于

>= 大于等于

<= 小于等于

= = 等于

!= 不等于

关系运算用于比较两个数的大小，用关系运算符将两个表达式连接起来形成的式子称为关系表达式。关系表达式通常用来作为判别条件构造分支或循环程序。关系表达式的一般形式如下：

表达式 1 关系运算符 表达式 2

关系运算的结果为逻辑量，成立为真（1），不成立为假（0）。其结果可以作为一个逻辑量参与逻辑运算。例如：5>3，结果为真（1），而 10= =100，结果为假（0）。

注意：关系运算符等于“= =”是由两个“=”组成。

4. 逻辑运算符

C51 有 3 种逻辑运算符：

|| 逻辑或

&& 逻辑与

! 逻辑非

关系运算符用于反映两个表达式之间的大小关系，逻辑运算符则用于求条件式的逻辑值，用逻辑运算符将关系表达式或逻辑量连接起来的式子就是逻辑表达式。

逻辑与，格式：

条件式 1 && 条件式 2

当条件式 1 与条件式 2 都为真时结果为真（非 0 值），否则为假（0 值）。

逻辑或，格式：

条件式 1 || 条件式 2

当条件式 1 与条件式 2 都为假时结果为假（0 值），否则为真（非 0 值）。

逻辑非，格式：

！条件式

当条件式原来为真（非 0 值），逻辑非后结果为假（0 值）。当条件式原来为假（0 值），逻辑非后结果为真（非 0 值）。

例如：若 a=8，b=3，c=0，则！a 为假，a && b 为真，b && c 为假。

C51 语言能对运算对象按位进行操作，它与汇编语言使用一样方便。位运算是按位对变量进行运算，但并不改变参与运算的变量的值。如果要求按位改变变量的值，则要利用相应的赋值运算。C51 中位运算符只能对整数进行操作，不能对浮点数进行操作。C51 中的位运算符有：

5. 位运算符

& 按位与

| 按位或

^ 按位异或

~ 按位取反

<< 左移

>> 右移

【例】设 a=0x45=01010100B，b=0x3b=00111011B，则 a&b、a|b、a^b、~a、a<<2、b>>2 分别为多少？

a&b=00010000b=0x10。
a|b=01111111B=0x7f。
a^b=01101111B=0x6f。
~a=10101011B=0xab。
a<<2=01010000B=0x50。
b>>2=00001110B=0x0e。

6. 复合赋值运算符

C51 语言中支持在赋值运算符“=”的前面加上其它运算符，组成复合赋值运算符。下面是 C51 中支持的复合赋值运算符

+=	加法赋值	+	减法赋值
*=	乘法赋值	/=	除法赋值

%=　取模赋值　　&=　逻辑与赋值

|=　逻辑或赋值　　^=　逻辑异或赋值

~=　逻辑非赋值　　>>=　右移位赋值

<<=　左移位赋值

复合赋值运算的一般格式如下：

变量　复合运算赋值符　表达式

它的处理过程：先把变量与后面的表达式进行某种运算，然后将运算的结果赋给前面的变量。其实这是 C51 语言中简化程序的一种方法，大多数二目运算都可以用复合赋值运算符简化表示。例如：a+=6 相当于 a=a+6；a*=5 相当于 a=a*5；b&=0x55 相当于 b=b&0x55；x>>=2 相当于 x=x>>2。

7. 逗号运算符

在 C51 语言中，逗号“,”是一个特殊的运算符，可以用它将两个或两个以上的表达式连接起来，称为逗号表达式。逗号表达式的一般格式为：

表达式 1，表达式 2，……，表达式 n

程序执行时对逗号表达式的处理：按从左至右的顺序依次计算出各个表达式的值，而整个逗号表达式的值是最右边的表达式（表达式 n）的值。例如：x=(a=3,6*3)结果 x 的值为 18。

8. 条件运算符

条件运算符“？：”是 C51 语言中唯一的一个三目运算符，它要求有三个运算对象，用它可以将三个表达式连接在一起构成一个条件表达式。条件表达式的一般格式为：

逻辑表达式？表达式 1：表达式 2

功能是先计算逻辑表达式的值，当逻辑表达式的值为真（非 0 值）时，将计算的表达式 1 的值作为整个条件表达式的值；当逻辑表达式的值为假（0 值）时，将计算的表达式 2 的值作为整个条件表达式的值。例如：条件表达式 max=(a>b)?a:b 的执行结果是将 a 和 b 中较大的数赋值给变量 max。

9. 指针与地址运算符

指针是 C51 语言中的一个十分重要的概念，在 C51 中的数据类型中专门有一种指针类型。指针为变量的访问提供了另一种方式，变量的指针就是该变量的地

址，还可以定义一个专门指向某个变量的地址的指针变量。

为了表示指针变量和它所指向的变量地址之间的关系，C51 中提供了两个专门的运算符：

*　　指针运算符

&　　取地址运算符

指针运算符“*”放在指针变量前面，通过它实现访问以指针变量的内容为地址所指向的存储单元。例如：指针变量 p 中的地址为 2000H，则*p 所访问的是地址为 2000H 的存储单元，x=*p，实现把地址为 2000H 的存储单元的内容送给变量 x。

取地址运算符“&”放在变量的前面，通过它取得变量的地址，变量的地址通常送给指针变量。例如：设变量 x 的内容为 12H，地址为 2000H，则&x 的值为 2000H，如有一指针变量 p，则通常用 p=&x，实现将 x 变量的地址送给指针变量 p，指针变量 p 指向变量 x，以后可以通过*p 访问变量 x。

表达式语句及复合语句

A.　表达式语句

在表达式的后边加一个分号“;”就构成了表达式语句 ,如:

```
a=++b*9;
x=8; y=7;
++k;
```

可以一行放一个表达式形成表达式语句，也可以一行放多个表达式形成表达式语句，这时每个表达式后面都必须带“;”号，另外，还可以仅由一个分号“;”占一行形成一个表达式语句，这种语句称为空语句。

空语句在程序设计中通常用于两种情况：

（1）　在程序中为有关语句提供标号，用以标记程序执行的位置。例如采用下面的语句可以构成一个循环。

```
repeat:;
 :
 goto  repeat;
```

（2）在用 while 语句构成的循环语句后面加一个分号，形成一个不执行其它操作的空循环体。这种结构通常用于对某位进行判断，当不满足条件则等待，满足条件则执行。

【例】下面这段子程序用于读取 8051 单片机的串行口的数据，当没有接收到则等待，当接收到，接收数据后返回，返回值为接收的数据。

```
#include  <reg51.h>
char  getchar()
{
char  c;
while(!RI);                  //当接收中断标志位 RI 为 0 则等待，当接收中断标志位为 1 则;等待结束
c=SBUF;
RI=0;
return(c);
}
```

B. 复合语句

复合语句是由若干条语句组合而成的一种语句，在 C51 中，用一个大括号“{ }”将若干条语句括在一起就形成了一个复合语句，复合语句最后不需要以分号“;”结束，但它内部的各条语句仍需以分号“;”结束。复合语句的一般形式为：

```
{
局部变量定义;
语句 1;
语句 2;
}
```

复合语句在执行时，其中的各条单语句按顺序依次执行，整个复合语句在语

法上等价于一条单语句，因此在 C51 中可以将复合语句视为一条单语句。通常复合语句出现在函数中，实际上，函数的执行部分（即函数体）就是一个复合语句；复合语句中的单语句一般是可执行语句，此外还可以是变量的定义语句（说明变量的数据类型）。在复合语句内部语句所定义的变量，称为该复合语句中的局部变量，它仅在当前这个复合语句中有效。利用复合语句将多条单语句组合在一起，以及在复合语句中进行局部变量定义是 C51 语言的一个重要特征。

C51 的输入输出

在 C51 语言中，它本身不提供输入和输出语句，输入和输出操作是由函数来实现的。在 C51 的标准函数库中提供了一个名为“stdio.h”的一般 I/O 函数库，它当中定义了 C51 中的输入和输出函数。当对输入和输出函数使用时，须先用预处理命令“#include <stdio.h>”将该函数库包含到文件中。

在 C51 的一般 I/O 函数库中定义的 I/O 函数都是通过串行接口实现，在使用 I/O 函数之前，应先对 51 单片机的串行接口进行初始化。选择串口工作于方式 2（8 位自动重载方式），波特率由定时器/计数器 1 溢出率决定。例如，设系统时钟为 12MHZ，波特率为 2400，则初始化程序如下：

```
SCON=0x52;
TMOD=0X20;
TH1=0xf3;
TR1=1;
```

A. 格式输出函数 printf()

printf()函数的的作用是通过串行接口输出若干任意类型的数据，它的格式如下：

printf(格式控制，输出参数表)

格式控制是用双引号括起来的字符串，也称转换控制字符串，它包括三种信息：格式说明符、普通字符和转义字符。

（1） 格式说明符，由“%”和格式字符组成，它的作用是用于指明输出的数据的格式输出，如%d、%f 等，它们的具体情况见下表。

（2） 普通字符，这些字符按原样输出，用来输出某些提示信息。

（3） 转义字符，就是前面介绍的转义字符（下表），用来输出特定的控制符，如输出转义字符\n 就是使输出换一行。

输出参数表是需要输出的一组数据，可以是表达式。

格式字符	数据类型	输出格式
d	int	带符号十进制数
u	int	无符号十进制数
o	int	无符号八进制数
x	int	无符号十六进制数，用“a~f”表示
X	int	无符号十六进制数，用“A~F”表示
f	float	带符号十进制数浮点数，形式为[-]dddd.dddd
e，E	float	带符号十进制数浮点数，形式为[-]d.ddddE±dd
g，G	float	自动选择 e 或 f 格式中更紧凑的一种输出格式
c	char	单个字符
s	指针	指向一个带结束符的字符串
p	指针	带存储器批示符和偏移量的指针，形式为 M：aaaa。其中，M 可分别为：C(code)，D(data)，I(idata)，P(pdata)，如 M 为 a，则表示的是指针偏移量

B. 格式输入函数 scanf（）

scanf（）函数的作用是通过串行接口实现数据输入，它的使用方法与 printf（）类似，scanf（）的格式如下：

scanf（格式控制，地址列表）

格式控制与 printf（）函数的情况类似，也是用双引号括起来的一些字符，可以包括以下三种信息：空白字符、普通字符和格式说明。

（1） 空白字符，包含空格、制表符、换行符等，这些字符在输出时被忽略。

（2） 普通字符，除了以百分号“%”开头的格式说明符而外的所有非空白字符，在输入时要求原样输入。

（3） 格式说明，由百分号“%”和格式说明符组成，用于指明输入数据的格式，它的基本情况与 printf（）相同，具体情况见表 4-5。

地址列表是由若干个地址组成，它可以是指针变量、取地址运算符“&”加变量（变量的地址）或字符串名（表示字符串的首地址）。

格式字符	数据类型	输出格式

d	int 指针	带符号十进制数
u	int 指针	无符号十进制数
o	int 指针	无符号八进制数
x	int 指针	无符号十六进制数
f，e，E	float 指针	浮点数
c	char 指针	字符
s	string 指针	字符串

【例】 使用格式输入输出函数的例子

```
#include  <reg52.h>      //包含特殊功能寄存器库
#include  <stdio.h>       //包含 I/O 函数库
void main(void)             //主函数
{
int  x,y;                                 //定义整型变量 x 和 y
SCON=0x52;                    //串口初始化
TMOD=0x20;
TH1=0XF3;
TR1=1;
printf(“input  x,y:\n”);                      //输出提示信息
scanf(“%d%d”,&x,&y);                   //输入 x 和 y 的值
printf(“\n”);                                          //输出换行
printf(“%d+%d=%d”,x,y,x+y);     //按十进制形式输出
printf(“\n”);                                          //输出换行
printf(“%xH+%xH=%XH”,x,y,x+y);     //按十六进制形式输出
while(1);       //结束
}
```

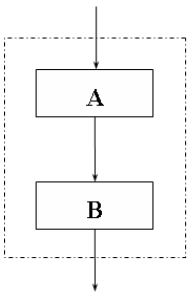
C51 程序基本结构与相关语句

A. C51 的基本结构

1) 顺序结构

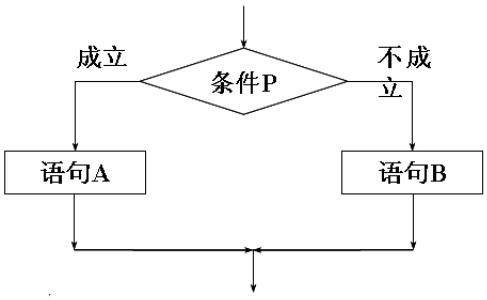
顺序结构是最基本、最简单的结构，在这种结构中，程序由低地址到高地址

依次执行，如图给出顺序结构流程图，程序先执行 A 操作，然后再执行 B 操作。



二．选择结构

择结构可使程序根据不同的情况，选择执行不同的分支，在选择结构中，程序先都对一个条件进行判断。当条件成立，即条件语句为“真”时，执行一个分支，当条件不成立时，即条件语句为“假”时，执行另一个分支。如图：当条件 S 成立时，执行分支 A，当条件 P 不成立时，执行分支 B。



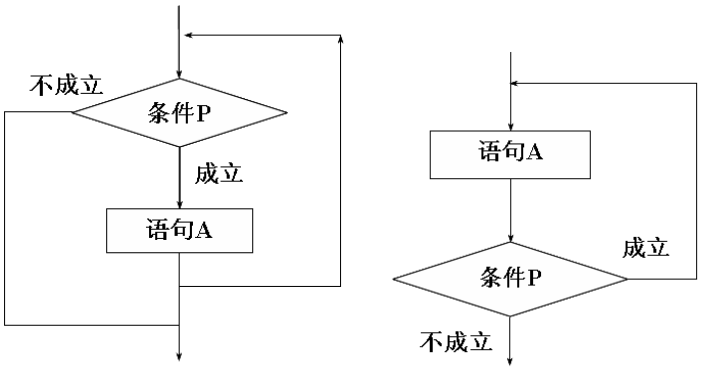
在 C51 中，实现选择结构的语句为 if/else，if/else　if 语句。另外在 C51 中还支持多分支结构，多分支结构既可以通过 if 和 else　if 语句嵌套实现，可用 swith/case 语句实现。

三．循环结构

在程序处理过程中，有时需要某一段程序重复执行多次，这时就需要循环结构来实现，循环结构就是能够使程序段重复执行的结构。循环结构又分为两种：当（while）型循环结构和直到（do...while）型循环结构。

（1） 当型循环结构

当型循环结构如图：当条件 P 成立（为“真”）时，重复执行语句 A，当条件不成立（为“假”）时才停止重复，执行后面的程序。



（2） 直到型循环结构

直到型循环结构，先执行语句 A，再判断条件 P，当条件成立（为“真”）时，再重复执行语句 A，直到条件不成立（为“假”）时才停止重复，执行后面的程序。

构成循环结构的语句主要有：while、do while、for、goto

B. if 语句

if 语句是 C51 中的一个基本条件选择语句，它通常有三种格式：

（1）if （表达式）｛语句；｝

（2）if （表达式）｛语句 1；｝ else ｛语句 2；｝

（3）if （表达式 1）｛语句 1；｝

else if （表达式 2） （语句 2；）

else if （表达式 3） （语句 3；）

……

else if （表达式 n-1） （语句 n-1；）

else ｛语句 n｝

【例】 if 语句的用法。

（1）if (x!=y) printf(“x=%d,y=%d\n”,x,y);

执行上面语句时，如果 x 不等于 y，则输出 x 的值和 y 的值。

（2）if (x>y) max=x;

else max=y;

执行上面语句时，如 x 大于 y 成立，则把 x 送给最大值变量 max，如 x 大于 y 不成立，则把 y 送给最大值变量 max。使 max 变量得到 x、y 中的大数。

（3）if (score>=90) printf(“Your result is an A\n”);

```
else  if  (score>=80)  printf(“Your result is an B\n”);
else  if  (score>=70)  printf(“Your result is an C\n”);
else  if  (score>=60)  printf(“Your result is an D\n”);
else  printf(“Your result is an E\n”);
```

执行上面语句后，能够根据分数 score 分别打出 A、B、C、D、E 五个等级。

C. switch/case 语句

if 语句通过嵌套可以实现多分支结构，但结构复杂。switch 是 C51 中提供的专门处理多分支结构的多分支选择语句。它的格式如下：

```
switch （表达式）
{case  常量表达式 1：{语句 1；}break；
case  常量表达式 2：{语句 2；}break；
……
case  常量表达式 n：{语句 n；}break；
default：{语句 n+1；}
```

说明如下：

（1） switch 后面括号内的表达式，可以是整型或字符型表达式。

（2） 当该表达式的值与某一“case”后面的常量表达式的值相等时，就执行该“case”后面的语句，然后遇到 break 语句退出 switch 语句。若表达式的值与所有 case 后的常量表达式的值都不相同，则执行 default 后面的语句，然后退出 switch 结构。

（3） 每一个 case 常量表达式的值必须不同否则会出现自相矛盾的现象。

（4） case 语句和 default 语句的出现次序对执行过程没有影响。

（5） 每个 case 语句后面可以有“break”，也可以没有。有 break 语句，执行到 break 则退出 switch 结构，若没有，则会顺次执行后面的语句，直到遇到 break 或结束。

（6） 每一个 case 语句后面可以带一个语句，也可以带多个语句，还可以不带。语句可以用花括号括起，也可以不括。

（7） 多个 case 可以共用一组执行语句。

【例】 switch/case 语句的用法。

对学生成绩划分为 A~D，对应不同的百分制分数，要求根据不同的等级打印出它的对应百分数。可以通过下面的 switch/case 语句实现。

```
……
switch（grade）
{
    case   ‘A’; printf（”90~100\n”); break;
    case   ‘B’; printf（”80~90\n”); break;
    case   ‘C’; printf（”70~80\n”); break;
    case   ‘D’; printf（”60~70\n”); break;
    case   ‘E’; printf（”<60\n”); break;
    default; printf（”error”\n）
}
```

D. while 语句

while 语句在 C51 中用于实现当型循环结构，它的格式如下：

```
while（表达式）
    {语句;}  /*循环体*/
```

while 语句后面的表达式是能否循环的条件，后面的语句是循环体。当达式为非 0（真）时，就重复执行循环体内的语句；当表达式为 0（假），则中止 while 循环，程序将执行循环结构之外的下一条语句。它的特点是：先判断条件，后执行循环体。在循环体中对条件进行改变，然后再判断条件，如条件成立，则再执行循环体，如条件不成立，则退出循环。如条件第一次就不成立，则循环体一次也不执行。

【例】 下面程序是通过 while 语句实现计算并输出 1~100 的累加和。

```
#include  <reg52.h>      //包含特殊功能寄存器库
#include  <stdio.h>      //包含 I/O 函数库
void main(void)       //主函数
{
    int  i,s=0;          //定义整型变量 x 和 y
    i=1;
```

```
    SCON=0x52;          //串口初始化
    TMOD=0x20;
    TH1=0XF3;
    TR1=1;
    while  (i<=100)        //累加 1~100 之和在 s 中
    {
        s=s+i;
        i++;
    }
    printf(“1+2+3……+100=%d\n”,s);
    while(1);
}
```

程序执行的结果：

1+2+3……+100=5050

E. do while 语句

do while 语句在 C51 中用于实现直到型循环结构，它的格式如下：

```
do
    {语句；}              /*循环体*/
while（表达式）;
```

它的特点是：先执行循环体中的语句，后判断表达式。如表达式成立（真），则再执行循环体，然后又判断，直到有表达式不成立（假）时，退出循环，执行 do while 结构的下一条语句。do while 语句在执行时，循环体内的语句至少会被执行一次。

【例】 通过 do while 语句实现计算并输出 1~100 的累加和。

```
#include  <reg52.h>  //包含特殊功能寄存器库
#include  <stdio.h>   //包含 I/O 函数库
void main(void)        //主函数
{
    i,s=0;             //定义整型变量 x 和 y
```

```
    i=1;
    SCON=0x52;          //串口初始化
    TMOD=0x20;
    TH1=0XF3;
    TR1=1;
    do                          //累加 1~100 之和在 s 中
    {
        s=s+i;
        i++;
    }
    while  (i<=100);
    printf(“1+2+3……+100=%d\n”,s);
    while(1);
}
```

F. for 语句

for（表达式 1；表达式 2；表达式 3）

{语句；} /*循环体*/

for 语句后面带三个表达式，它的执行过程如下：

（1） 先求解表达式 1 的值。

（2） 求解表达式 2 的值，如表达式 2 的值为真，则执行循环休中的语句，然后执行下一步（3）的操作，如表达式 2 的值为假，则结束 for 循环，转到最后一步。

（3） 若表达式 2 的值为真，则执行完循环体中的语句后，求解表达式 3，然后转到第四步。

（4） 转到（2）继续执行。

（5） 退出 for 循环，执行下面的一条语句。

在 for 循环中，一般表达式 1 为初值表达式，用于给循环变量赋初值；表达式 2 为条件表达式，对循环变量进行判断；表达式 3 为循环变量更新表达式，用于对循环变量的值进行更新，使循环变量能不满足条件而退出循环。

【例】 用 for 语句实现计算并输出 1~100 的累加和。

```
#include  <reg52.h>     //包含特殊功能寄存器库
#include  <stdio.h>      //包含 I/O 函数库
void main(void)            //主函数
{
    int  i,s=0;                          //定义整型变量 x 和 y
    SCON=0x52;                  //串口初始化
    TMOD=0x20;
    TH1=0XF3;
    TR1=1;
    for (i=1;i<=100;i++)  s=s+i;      //累加 1~100 之和在 s 中
    printf(“1+2+3……+100=%d\n”,s);
    while(1);
}
```

G. 循环的嵌套

在一个循环的循环体中允许又包含一个完整的循环结构，这种结构称为循环的嵌套。外面的循环称为外循环，里面的循环称为内循环，如果在内循环的循环体内又包含循环结构，就构成了多重循环。在 C51 中，允许三种循环结构相互嵌套。

【例】用嵌套结构构造一个延时程序。

```
void  delay(unsigned  int  x)
{
    unsigned  char j;
    while(x--)
    {for (j=0;j<125;j++);}
}
```

这里，用内循环构造一个基准的延时，调用时通过参数设置外循环的次数，这样就可以形成各种延时关系。

H. break 和 continue 语句

break 和 continue 语句通常用于循环结构中，用来跳出循环结构。但是二者又有所不同，下面分别介绍。

1. break 语句

前面已介绍过用 break 语句可以跳出 switch 结构，使程序继续执行 switch 结构后面的一个语句。使用 break 语句还可以从循环体中跳出循环，提前结束循环而接着执行循环结构下面的语句。它不能用在除了循环语句和 switch 语句之外的任何其它语句中。

【例 19】下面一段程序用于计算圆的面积，当计算到面积大于 100 时，由 break 语句跳出循环。

```
for (r=1; r<=10; r++)
{
    area=pi*r*r;
    if (area>100) break;
    printf("%f\n", area);
}
```

2. continue 语句

continue 语句用在循环结构中，用于结束本次循环，跳过循环体中 continue 下面尚未执行的语句，直接进行下一次是否执行循环的判定。

continue 语句和 break 语句的区别在于：continue 语句只是结束本次循环而不是终止整个循环；break 语句则是结束循环，不再进行条件判断。

【例 20】 输出 100~200 间不能被 3 整除的数。

```
for (i=100; i<=200; i++)
{
    if  (i%3= =0)  continue;
    printf("%d  "; i);
}
```

在程序中，当 i 能被 3 整除时，执行 continue 语句，结束本次循环，跳过 printf（）函数，只有能被 3 整除时才执行 printf（）函数。

I. return 语句

return 语句一般放在函数的最后位置，用于终止函数的执行，并控制程序返回调用该函数时所处的位置。返回时还可以通过 return 语句带回返回值。return 语句格式有两种：

（1）return;

（2）return (表达式);

如果 return 语句后面带有表达式，则要计算表达式的值，并将表达式的值作为函数的返回值。若不带表达式，则函数返回时将返回一个不确定的值。通常我们用 return 语句把调用函数取得的值返回给主调用函数。

函　数

A.　函数的定义

函数定义的一般格式如下：

函数类型　函数名（形式参数表）　[reentrant][interrupt　m][using　n]

形式参数说明

```
{
    局部变量定义
    函数体
}
```

前面部件称为函数的首部，后面称为函数的尾部，格式说明：

1．函数类型

函数类型说明了函数返回值的类型。

2．函数名

函数名是用户为自定义函数取的名字以便调用函数时使用。

3．形式参数表

形式参数表用于列录在主调函数与被调用函数之间进行数据传递的形式参数。

【例】定义一个返回两个整数的最大值的函数 max()。

```
int  max(int  x,int  y)
{
    int  z;
    z=x>y?x：y;
```

```
    return（z）;
}
```

也可以用成这样：

```
int  max(x,y)
int  x,y;
{
  int  z;
  z=x>y?x：y;
  return（z）;
}
```

4. reentrant 修饰符

这个修饰符用于把函数定义为可重入函数。所谓可重入函数就是允许被递归调用的函数。函数的递归调用是指当一个函数正被调用尚未返回时，又直接或间接调用函数本身。一般的函数不能做到这样，只有重入函数才允许递归调用。关于重入函数，注意以下几点：

（1） 用 reentrant 修饰的重入函数被调用时，实参表内不允许使用 bit 类型的参数。函数体内也不允许存在任何关于位变量的操作，更不能返回 bit 类型的值。

（2） 编译时，系统为重入函数在内部或外部存储器中建立一个模拟堆栈区，称为重入栈。重入函数的局部变量及参数被放在重入栈中，使重入函数可以实现递归调用。

（3） 在参数的传递上，实际参数可以传递给间接调用的重入函数。无重入属性的间接调用函数不能包含调用参数，但是可以使用定义的全局变量来进行参数传递。

5. interrupt m 修饰符

interrupt m 是 C51 函数中非常重要的一个修饰符，这是因为中断函数必须通过它进行修饰。在 C51 程序设计中，当函数定义时用了 interrupt m 修饰符，系统编译时把对应函数转化为中断函数，自动加上程序头段和尾段，并按 51 系统中断的处理方式自动把它安排在程序存储器中的相应位置。

在该修饰符中，m 的取值为 0~31，对应的中断情况如下：

0——外部中断 0

1——定时/计数器 T0

2——外部中断 1

3——定时/计数器 T1

4——串行口中断

5——定时/计数器 T2

其它值预留。

编写 51 中断函数注意如下：

（1） 中断函数不能进行参数传递，如果中断函数中包含任何参数声明都将导致编译出错。

（2） 中断函数没有返回值，如果企图定义一个返回值将得不到正确的结果，建议在定义中断函数时将其定义为 void 类型，以明确说明没有返回值。

（3） 在任何情况下都不能直接调用中断函数，否则会产生编译错误。因为中断函数的返回是由 8051 单片机的 RETI 指令完成的，RETI 指令影响 8051 单片机的硬件中断系统。如果在没有实际中断情况下直接调用中断函数，RETI 指令的操作结果会产生一个致命的错误。

（4） 如果在中断函数中调用了其它函数，则被调用函数所使用的寄存器必须与中断函数相同。否则会产生不正确的结果。

（5） C51 编译器对中断函数编译时会自动在程序开始和结束处加上相应的内容，具体如下：在程序开始处对 ACC、B、DPH、DPL 和 PSW 入栈，结束时出栈。中断函数未加 using n 修饰符的，开始时还要将 R0~R1 入栈，结束时出栈。如中断函数加 using n 修饰符，则在开始将 PSW 入栈后还要修改 PSW 中的工作寄存器组选择位。

（6） C51 编译器从绝对地址 8m+3 处产生一个中断向量，其中 m 为中断号，也即 interrupt 后面的数字。该向量包含一个到中断函数入口地址的绝对跳转。

（7） 中断函数最好写在文件的尾部，并且禁止使用 extern 存储类型说明。防止其它程序调用。

【例】编写一个用于统计外中断 0 的中断次数的中断服务程序

```
extern  int  x;
void  int0()  interrupt 0  using 1
{
  x++;
}
```

6. using n 修饰符

修饰符 using n 用于指定本函数内部使用的工作寄存器组，其中 n 的取值为 0~3，表示寄存器组号。

对于 using n 修饰符的使用，注意以下几点：

（1） 加入 using n 后，C51 在编译时自动的在函数的开始处和结束处加入以下指令。

```
{
PUSH  PSW       ；标志寄存器入栈
MOV  PSW，#与寄存器组号相关的常量
……
POP  PSW      ；标志寄存器出栈
}
```

（2） using n 修饰符不能用于有返回值的函数，因为 C51 函数的返回值是放在寄存器中的。如寄存器组改变了，返回值就会出错。

B. 函数的调用与声明

2) 函数的调用

函数调用的一般形式如下：

函数名（实参列表）;

对于有参数的函数调用，若实参列表包含多个实参，则各个实参之间用逗号隔开。

按照函数调用在主调函数中出现的位置，函数调用方式有以下三种：

（1） 函数语句。把被调用函数作为主调用函数的一个语句。

（2） 函数表达式。函数被放在一个表达式中，以一个运算对象的方式出现。这时的被调用函数要求带有返回语句，以返回一个明确的数值参加表达式的运算。

（3） 函数参数。被调用函数作为另一个函数的参数。

二．自定义函数的声明

在 C51 中，函数原型一般形式如下：

[extern] 函数类型 函数名（形式参数表）；

函数的声明是把函数的名字、函数类型以及形参的类型、个数和顺序通知编译系统，以便调用函数时系统进行对照检查。函数的声明后面要加分号。

如果声明的函数在文件内部，则声明时不用 extern，如果声明的函数不在文件内部，而在另一个文件中，声明时须带 extern，指明使用的函数在另一个文件中。

【例】函数的使用

```
#include  <reg52.h>        //包含特殊功能寄存器库
#include  <stdio.h>         //包含 I/O 函数库
int  max(int  x,int  y);       //对 max 函数进行声明
void main(void)               //主函数
{
    int  a,b;
    SCON=0x52;                    //串口初始化
    TMOD=0x20;
    TH1=0XF3;
    TR1=1;
    scanf(“please input a,b:%d,%d”,&a,&b);
    printf(“\n”);
    printf(“max is:%d\n”,max(a,b));
    while(1);
}
int  max(int  x,int  y)
{
    int  z;
    z=(x>=y?x:y);
    return(z);
}
```

【例 24】 外部函数的使用

程序 serial_initial.c

```
#include  <reg52.h>  //包含特殊功能寄存器库
#include  <stdio.h>  //包含 I/O 函数库
void serial_initial(void)    //主函数
{
    SCON=0x52;      //串口初始化
    TMOD=0x20;
    TH1=0XF3;
    TR1=1;
}
```

程序 y1.c

```
#include  <reg52.h>  //包含特殊功能寄存器库
#include  <stdio.h>  //包含 I/O 函数库
extern  serial_initial();
void  main(void)
{
    int  a,b;
    serial_initial();
    scanf(“please input a,b:%d,%d”,&a,&b);
    printf(“\n”);
    printf(“max is:%d\n”,a>=b?a:b);
    while(1);
}
```

C. 函数的嵌套与递归

3) 函数的嵌套

在一个函数的调用过程中调用另一个函数。C51 编译器通常依靠堆栈来进行参数传递，堆栈设在片内 RAM 中，而片内 RAM 的空间有限，因而嵌套的深度比较有限，一般在几层以内。如果层数过多，就会导致堆栈空间不够而出错。

【例】 函数的嵌套调用

```
#include  <reg52.h>  //包含特殊功能寄存器库
#include  <stdio.h>  //包含 I/O 函数库
extern  serial_initial();
int  max(int  a,int  b)
{
    int  z;
    z=a>=b?a:b;
    return(z);
}
int  add(int  c,int  d,int  e,int  f)
{
    int  result;
    result=max(c,d)+max(e,f);     //调用函数 max
```

```
    return(result);
}
main()
{
    int  final;
    serial_initial();
    final=add(7,5,2,8);
    printf("%d",final);
    while(1);
}
```

二. 函数的递归

递归调用是嵌套调用的一个特殊情况。如果在调用一个函数过程中又出现了直接或间接调用该函数本身，则称为函数的递归调用。

在函数的递归调用中要避免出现无终止的自身调用，应通过条件控制结束递归调用，使得递归的次数有限。

下面是一个利用递归调用求 n!的例子。

【例】递归求数的阶乘 n!。

在数学计算中，一个数 n 的阶乘等于该数本身乘以数 n-1 的阶乘，即 n!=n(n-1)!，用 n-1 的阶乘来表示 n 的阶乘就是一种递归表示方法。在程序设计中通过函数递归调用来实现。

程序如下：

```
#include  <reg52.h>  //包含特殊功能寄存器库
#include  <stdio.h>  //包含 I/O 函数库
extern  serial_initial();
int  fac(int  n)  reentrant
{
    int  result;
    if  (n= =0)
        result=1;
    else
        result=n*fac(n-1);
    return(result);
}

main()
{
    int  fac_result;
    serial_initial();
    fac_result=fac(11);
```

```
    printf(“%d\n”,fac_result);
}
```

C51 构造数据类型

A. 数组

一. 一维数组

一维数组只有一个下标，定义的形式如下：

数据类型说明符　数组名[常量表达式][={初值，初值……}]

各部分说明如下：

1) “数据类型说明符”说明了数组中各个元素存储的数据的类型。
2) （2）“数组名”是整个数组的标识符，它的取名方法与变量的取名方法相同。
3) （3）“常量表达式”，常量表达式要求取值要为整型常量，必须用方括号“[]”括起来。用于说明该数组的长度，即该数组元素的个数。
4) （4）“初值部分”用于给数组元素赋初值，这部分在数组定义时属于可选项。对数组元素赋值，可以在定义时赋值，也可以定义之后赋值。在定义时赋值，后面须带等号，初值须用花括号括起来，括号内的初值两两之间用逗号间隔，可以对数组的全部元素赋值，也可以只对部分元素赋值。初值为 0 的元素可以只用逗号占位而不写初值 0。

例如：下面是定义数组的两个例子。

```
unsigned  char  x[5];
unsigned  int  y[3]={1,2,3};
```

第一句定义了一个无符号字符数组，数组名为 x，数组中的元素个数为 5。

第二句定义了一个无符号整型数组，数组名为 y，数组中元素个数为 3，定义的同时给数组中的三个元素赋初值，赋初值分别为 1、2、3。

需要注意的是，C51 语言中数组的下标是从 0 开始的，因此上面第一句定义的 5 个元素分别是：x[0]、x[1]、x[2]、x[3]、x[4]。第二句定义的 3 个元素分

别是：y[0]、y[1]、y[2]。赋值情况为：y[0]=1；y[1]=2；y[2]=3。

C51 规定在引用数组时，只能逐个引用数组中的各个元素，而不能一次引用整个数组。但如果是字符数组则可以一次引用整个数组。

【例】用数组计算并输出 Fibonacci 数列的前 20 项。

Fibonacci 数列在数学和计算机算法中十分有用。Fibonacci 数列是这样的一组数：第一个数字为 0，第二个数字为 1，之后每一个数字都是前两个数字之和。设计时通过数组存放 Fibonacci 数列，从第三项开始可通过累加的方法计算得到。

程序如下：

```
#include  <reg52.h>  //包含特殊功能寄存器库
#include  <stdio.h>  //包含 I/O 函数库
extern  serial_initial();
main()
{
    int  fib[20],i;
    fib[0]=0;
    fib[1]=1;
    serial_initial();
    for (i=2;i<20;i++)  fib[i]=fib[i-2]+fib[i-1];
    for (i=0;i<20;i++)
    {
        if (i%5= =0) printf(“\n”);
        printf(“%6d”,fib[i]);
    }
    while(1);
}
```

程序执行结果：

```
0   1   1   2    3
5     8   13    21       34
55      89  144     233      377
610 987     1597    2584     4148
```

二．字符数组

用来存放字符数据的数组称为字符数组，它是 C 语言中常用的一种数组。字符数组中的每一个元素都用来存放一个字符，也可用字符数组来存放字符串。字符数组的定义下一般数组相同，只是在定义时把数据类型定义为 char 型。

例如：char string1[10];

char string2[20];

上面定义了两个字符数组，分别定义了 10 个元素和 20 个元素。

在 C51 语言中，字符数组用于存放一组字符或字符串，字符串以“\0”作为结束符，只存放一般字符的字符数组的赋值与使用和一般的数组完全相同。对于存放字符串的字符数组。既可以对字符数组的元素逐个进行访问，也可以对整个数组按字符串的方式进行处理。

【例】对字符数组进行输入和输出。

```
#include  <reg52.h>  //包含特殊功能寄存器库
#include  <stdio.h>  //包含 I/O 函数库
extern  serial_initial();
main()
{
    char  string[20];
    serial_initial();
    printf(“please  type  any  character:”);
    scanf(“%s”,string);
    printf(“%s\n”,string);
    while(1);
}
```

B. 指针

指针是 C 语言中的一个重要概念。指针类型数据在 C 语言程序中使用十分普遍，正确地使用指针类型数据，可以有效地表示复杂的数据结构；可以动态地分配存储器，直接处理内存地址。

一. 指针的概念

了解指针的基本概念，先要了解数据在内存中的存储和读取方法。

在汇编语言中，对内存单元数据的访问是通过指明内存单元的地址。访问时有两种方式：直接寻址方式和间接寻址方式。直接寻址是通过在指令中直接给出数据所在单元的地址而访问该单元的数据。例如：MOV A，20H。在指令中直接给出所访问的内存单元地址 20H，访问的是地址为 20H 的单元的数据，该指令把地址为 20H 的片内 RAM 单元的内容送累加器 A；间接寻址是指所操作的数据所在的内存单元地址不是通过指令中直接提供，该地址是存放在寄存器中或其它的内存单元中，指令中指明存放地址的寄存器或内存单元来访问相应的数据。

在 C 语言中，可以通过地址方式来访问内存单元的数据，但 C 语言作为一种高级程序设计语言，数据通常是以变量的形式进行存放和访问的。对于变量，在

一个程序中定义了一个变量，编译器在编译时就在内存中给这个变量分配一定的字节单元进行存储。如对整型变量（int）分配 2 个字节单元，对于浮点型变量（float）分配 4 个字节单元，对于字符型变量分配 1 个字节单元等。变量在使用时分清两个概念：变量名和变量的值。前一个是数据的标识，后一个是数据的内容。变量名相当于内存单元的地址，变量的值相当于内存单元的内容。对于内存单元的数据访问方式有两种，对于变量也有两种访问方式：直接访问方式和间接访问方式。

直接访问方式。对于变量的访问，我们大多数时候是直接给出变量名。例如：printf(“%d”,a)，直接给出变量 a 的变量名来输出变量 a 的内容。在执行时，根据变量名得到内存单元的地址，然后从内存单元中取出数据按指定的格式输出。这就是直接访问方式。

间接访问方式。例如要存取变量 a 中的值时，可以先将变量 a 的地址放在另一个变量 b 中，访问时先找到变量 b，从变量 b 中取出变量 a 的地址，然后根据这个地址从内存单元中取出变量 a 的值。这就是间接访问。在这里，从变量 b 中取出的不是所访问的数据，而是访问的数据（变量 a 的值）的地址，这就是指针，变量 b 称为指针变量。

关于指针，注意两个基本概念：变量的指针和指向变量的指针变量。变量的指针就是变量的地址。对于变量 a，如果它所对应的内存单元地址为 2000H，它的指针就是 2000H。指针变量是指一个专门用来存放另一个变量地址的变量，它的值是指针。上面变量 b 中存放的是变量 a 的地址，变量 b 中的值是变量 a 的指针，变量 b 就是一个指向变量 a 的指针变量。

如上所述，指针实质上就是各种数据在内存单元的地址，在 C51 语言中，不仅有指向一般类型变量的指针，还有指向各种组合类型变量的指针。在本书中我们只讨论指向一般变量的指针的定义与引用，对于指向组合类型的指针，大家可以参考其它书籍学习它的使用。

二．指针变量的定义

指针变量的定义与一般变量的定义类似，定义的一般形式为：

数据类型说明符 ［存储器类型］ *指针变量名；

其中：

“ 数据类型说明符”说明了该指针变量所指向的变量的类型。

存储器类型”是可选项，它是 C51 编译器的一种扩展．如果带有此选项．指针被定义为基于存储器的指针。无此选项时，被定义为一般指针，这两种指针的区别在于它们占的存储字节不同。

下面是几个指针变量定义的例子：

```
int  * p1;          /*定义一个指向整型变量的指针变量 p1*/
char  * p2;         /*定义一个指向字符变量的指针变量 p2*/
char  data  * p3;       /*定义一个指向字符变量的指针变量 p3，该指针访问的数据在片内数据存储器中，该指针在内存中占一个字节*/
float  xdata  * p4;       /*定义一个指向字符变量的指针变量 p4，该指针访问的数据在片外数据存储器中，该指针在内存中占两个字节*/
```

三．指针变量的引用

指针变量是存放另一变量地址的特殊变量，指针变量只能存放地址。指针变量使用时注意两个运算符：&和*。这两个运算符在前面已经介绍，其中：“&”是取地址运算符，“*”是指针运算符。通过“&”取地址运算符可以把一个变量的地址送给指针变量，使指针变量指向该变量；通过“*”指针运算符可以实现通过指针变量访问它所指向的变量的值。

指针变量经过定义之后可以象其他基本类型变量一样引用。例如：

```
int  x，* px，* py;        /*变量及指针变量定义*/
px=&x;          /*将变量 x 的地址赋给指针变量 px，使 px 指向变量 x*/
* px=5;          /*等价于 x=5*/
py=px;          /*将指针变量 px 中的地址赋给指针变量 py，使指针变量 py 也指向 x*/
```

例】输入两个整数 x 与 y，经比较后按大小顺序输出。

程序如下：

```
#include  <reg52.h>  //包含特殊功能寄存器库
#include  <stdio.h>  //包含 I/O 函数库
extern  serial_initial();
main()
{
    int  x,y;
    int  * p,* p1,* p2;
```

```
    serial_initial();
    printf(“input  x  and  y:\n”);
    scanf(“%d%d”,&x,&y);
    p1=&x;p2=&y;
    if (x<y) {p=p1;p1=p2;p2=p;}
    printf(“max=%d,min=%d\n”,*p1,*p2);
    while(1);
}
```

程序执行结果：

```
input  x  and  y:
4  8
max=8,min=4
```

C. 结构

结构是一种组合数据类型，它是将若干个不同类型的变量结合在一起而形成的一种数据的集合体。组成该集合体的各个变量称为结构元素或成员。整个集合体使用一个单独的结构变量名。

一．结构与结构变量的定义

结构与结构变量是两个不同的概念，结构是一种组合数据类型，结构变量是取值为结构这种组合数据类型的变量，相当于整型数据类型与整型变量的关系。对于结构与结构变量的定义有两种方法。

1．先定义结构类型再定义结构变量

结构的定义形式如下：

struct 结构名

{结构元素表}；

结构变量的定义如下：

struct 结构名 结构变量名 1，结构变量名 2，……；

其中，“结构元素表”为结构中的各个成员，它可以由不同的数据类型组成。在定义时须指明各个成员的数据类型。

例如，定义一个日期结构类型 date，它由三个结构元素 year、month、day 组成，定义结构变量 d1 和 d2，定义如下：

```
struct  date
{
    int  year;
```

```
    char  month,day;
}
struct  date  d1,d2;
```

2．定义结构类型的同时定义结构变量名

这种方法是将两个步骤合在一起，格式如下：

```
struct  结构名
{结构元素表} 结构变量名 1，结构变量名 2，……;
```

例如对于上面的日期结构变量 d1 和 d2 可以按以下格式定义：

```
struct  date
{
    int  year;
    char  month,day;
}d1,d2;
```

对于结构的定义说明如下：

（1） 结构中的成员可以是基本数据类型，也可以是指针或数组，还可以是另一结构类型变量，形成结构的结构，即结构的嵌套。结构的嵌套可以是多层次的，但这种嵌套不能包含其自己。

（2） 定义的一个结构是一个相对独立的集合体，结构中的元素只在该结构中起作用，因而一个结构中的结构元素的名字可以与程序中的其它变量的名称相同，它们两者代表不同的对象，在使用时互相不影响。

（3） 结构变量在定义时也可以像其它变量在定义时加各种修饰符对它进行说明。

（4） 在 C51 中允许将具有相同结构类型的一组结构变量定义成结构数组，定义时与一般数组的定义相同，结构数组与一般变量数组的不同就在于结构数组的每一个元素都是具有同一结构的结构变量。

二．结构变量的引用

结构元素的引用一般格式如下：

结构变量名.结构元素名

或

结构变量名->结构元素名

其中，“.”是结构的成员运算符，例如：d1.year 表示结构变量 d1 中的元素 year，d2.day 表示结构变量 d2 中的元素 day 等。如果一个结构变量中结构元素又是另一个结构变量，即结构的嵌套，则需要用到若干个成员运算符，一级一级找到最低一级的结构元素，而且只能对这个最低级的结构元素进行引用，形如 d1.time.hour 的形式。

例】输入 3 个学生的语文、数学、英语的成绩，分别统计他们的总成绩并输出。

程序如下：

```
#include  <reg52.h>  //包含特殊功能寄存器库
#include  <stdio.h>  //包含 I/O 函数库
extern  serial_initial();
struct  student
{
    unsigned  char  name[10];
    unsigned  int  chinese;
    unsigned  int  math;
    unsigned  int  english;
    unsigned  int  total;
}p1[3];
main()
{
    unsigned  char  i;
    serial_initial();
    printf(“input  3  studend  name  and  result:\n”);
    for  (i=0;i<3;i++)
{
printf(“input  name:\n”);
scanf(“%s”,p1[i].name);
printf(“input  result:\n”);
scanf(“%d,%d,%d”,&p1[i].chinese,&p1[i].math,&p1[i].english);
}
    for  (i=0;i<3;i++)
    {
    p1[i].total=p1[i].chinese+p1[i].math+p1[i].english;
    }
    for  (i=0;i<3;i++)
    {
    printf(“%s total is %d”,p1[i].name,p1[i].total);
    printf(“\n”);
```

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

程序执行结果：

```
input  3  studend  name  and  result:
input  name:
wang
input  result:
76,87,69
input  name:
yang
input  result:
75,77,89
input  name:
zhang
input  result:
72,81,79
wang total is 232
yang total is 241
zhang total is 232
```

D. 联合

前面介绍的结构能够把不同类型的数据组合在一起使用，另外，在 C51 语言中，还提供一种组合类型－－联合，也能够把不同类型的数据组合在一起使用，但它与结构又不一样，结构中定义的各个变量在内存中占用不同的内存单元，在位置上是分开的，而联合中定义的各个变量在内存中都是从同一个地址开始存放，即采用了所谓的“覆盖技术”。这种技术可使不同的变量分时使用同一内存空间，提高内存的利用效率。

一．联合的定义

1．先定义联合类型再定义联合变量

定义联合类型，格式如下：

union 联合类型名

{成员列表}；

定义联合变量，格式如下：

union 联合类型名 变量列表；

例如：

union data

```
{
    float  i;
    int  j;
    char  k;
}
nion  data  a,b,c;
```

2. 定义联合类型的同时定义联合变量

格式如下：

```
union  联合类型名
{成员列表}变量列表;
```

例如：

```
union  data
{
    float  i;
    int  j;
    char  k;
}data  a,b,c;
```

可以看出，定义时，结构与联合的区别只是将关键字由 struct 换成 union，但在内存的分配上两者完全不同。结构变量占用的内存长度是其中各个元素所占用的内存长度的总和；而联合变量所占用的内存长度是其中各元素的长度的最大值。结构变量中的各个元素可以同时进行访问，联合变量中的各个元素在一个时刻只能对一个进行访问。

二. 联合变量的引用

联合变量中元素的引用与结构变量中元素的引用格式相同，形式如下：

```
联合变量名.联合元素
```

或

```
联合变量名->联合元素
```

例如：对于前面定义的联合变量 a、b、c 中的元素可以通过下面形式引用。

```
a.i;
```

b.j;

c.k;

分别引用联合变量 a 中的 float 型元素 i，联合变量 b 中的 int 型元素 j，联合变量 c 中的 char 型元素 k。

E. 枚举

枚举数据类型是一个有名字的某些整型常量的集合。这些整型常量是该类型变量可取的所有的合法值。枚举定义时应当列出该类型变量的所有可取值。

枚举定义的格式与结构和联合基本相同，也有两种方法。

先定义枚举类型，再定义枚举变量，格式如下：

enum 枚举名 {枚举值列表};

enum 枚举名 枚举变量列表;

或在定义枚举类型的同时定义枚举变量，格式如下：

enum 枚举名 {枚举值列表}枚举变量列表;

例如：定义一个取值为星期几的枚举变量 d1。

enum week {Sun,Mon,Tue,Wed,Thu,Fri,Sat};

enum week d1;

或

enum week {Sun,Mon,Tue,Wed,Thu,Fri,Sat} d1;

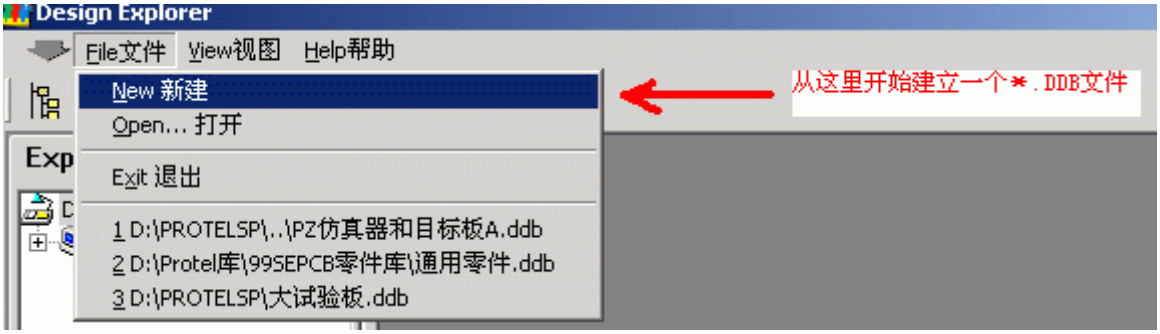
以后就可以把枚举值列表中各个值赋值给枚举变量 d1 进行使用了。

附录 B 电路板绘制软件 PROTEL 介绍

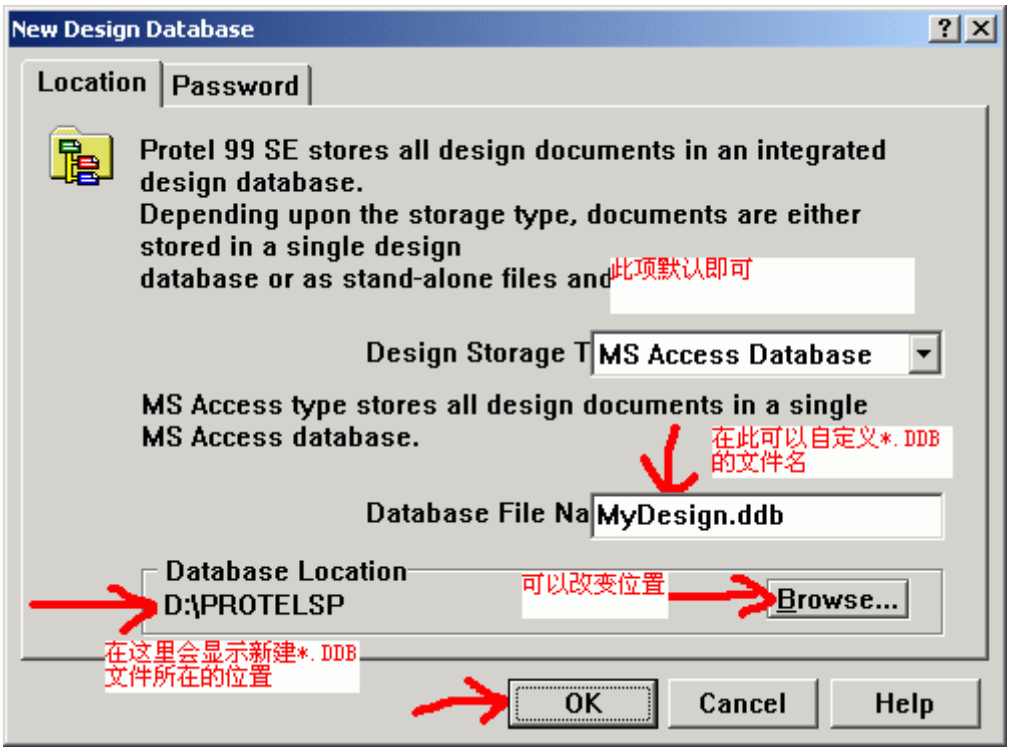
很多电子爱好者都有过学习 PROTEL 的经历，本人也是一样，摸索的学习，耐心的体会，充分的体会什么是成功之母。不希望大家把不必要的时间浪费在学习 PROTEL 的初期操作上，在这里做这个教程是为了给渴望快速了解和操作 PROTEL 的初学者们一个走捷径的机会，教程大家都可以看到，可以省走很多不必要的弯路及快速建立信心，网络的魅力之一就在于学习的效率很高。由于本人的水平很有限，所以教程做的比较浅，就是教大家：1.画画简单的原理图（SCH）2.学会创建 SCH 零件 2.把原理图转换成电路板（PCB）3.对 PCB 进行自动布线 4.学会创建 PCB 零件库 5.学会一些常用的 PCB 高级技巧。鉴于此，如果您这方面已经是水平很高的专业人士，无需看此教程。 同时也愿这些简单的图片教程可以使大家在今后的电子电路设计之路上所向披靡。

关于教程涉及软件版本：此教程采用的样板软件是 PROTEL99SE 汉化版，99SE 是 PROTEL 家族中目前最稳定的版本，功能强大。采用了*.DDB 数据库格式保存文件，所有同一工程相关的 SCH、PCB 等文件都可以在同一*.DDB 数据库中并存，非常科学，利于集体开发和文件的有效管理。还有一个优点就是自动布线引擎很强大。在双面板的前提下，可以在很短的时间内自动布通任何的超复杂线路！

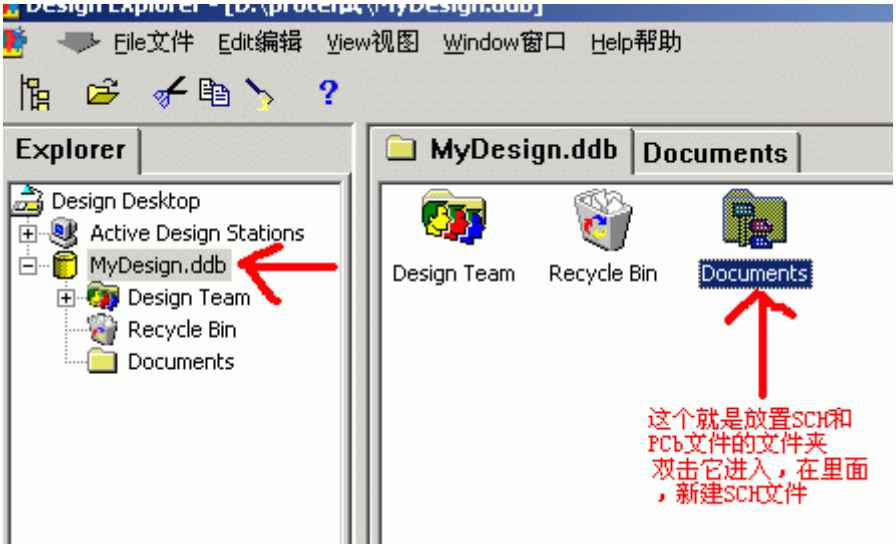
1.万事开头难，从建立一个 *.DDB 文件开始



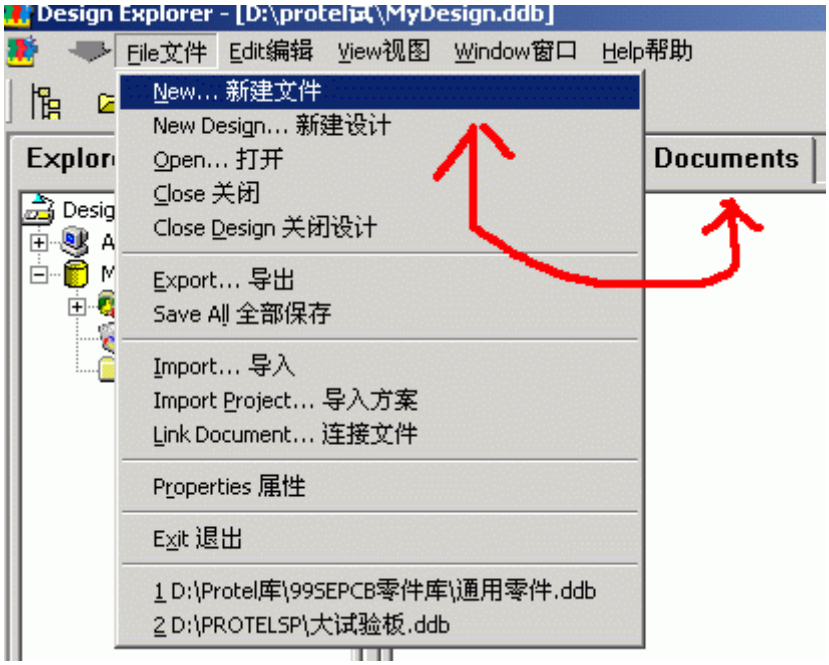
2.定义新建*.DDB 的选项.GIF



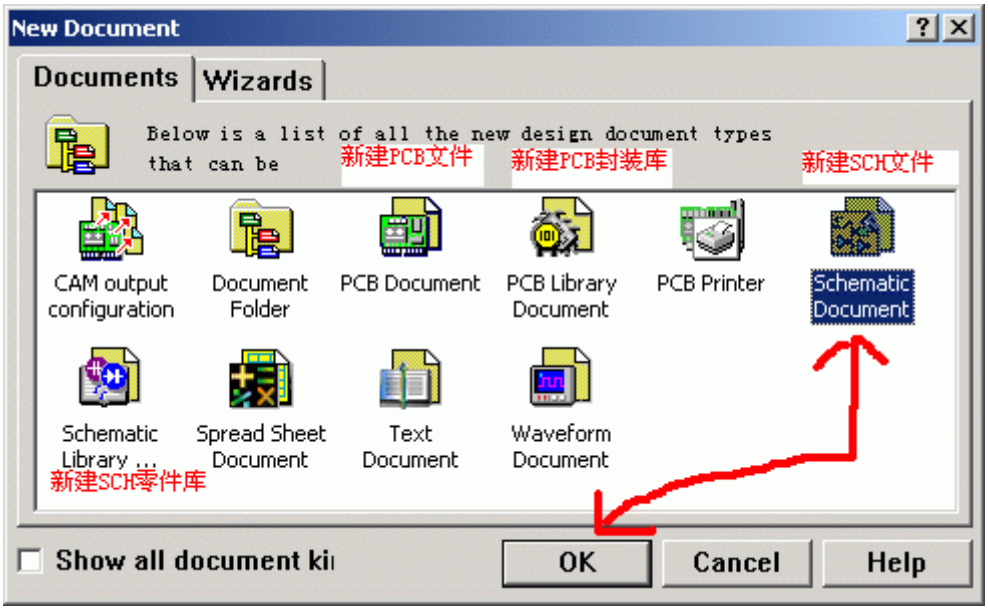
3.所有新建的文件一般放置在主文件夹中



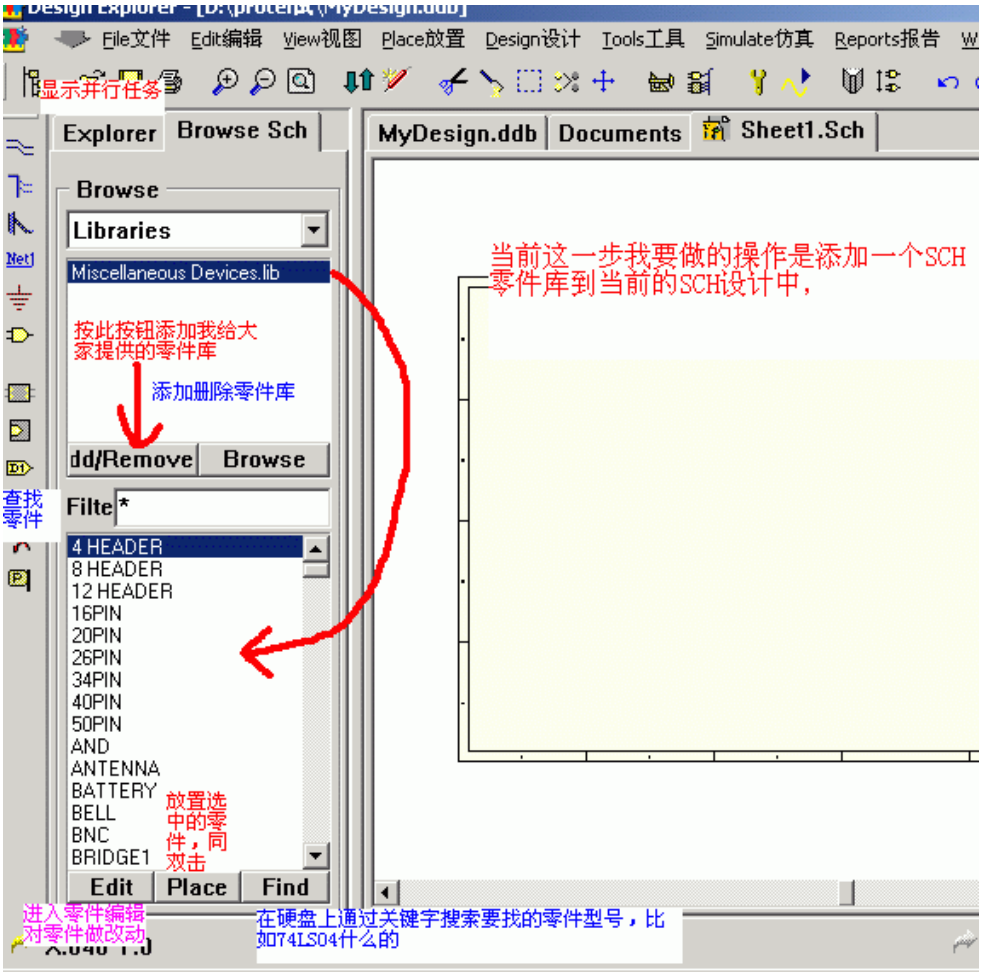
4.进入并新建*.SCH



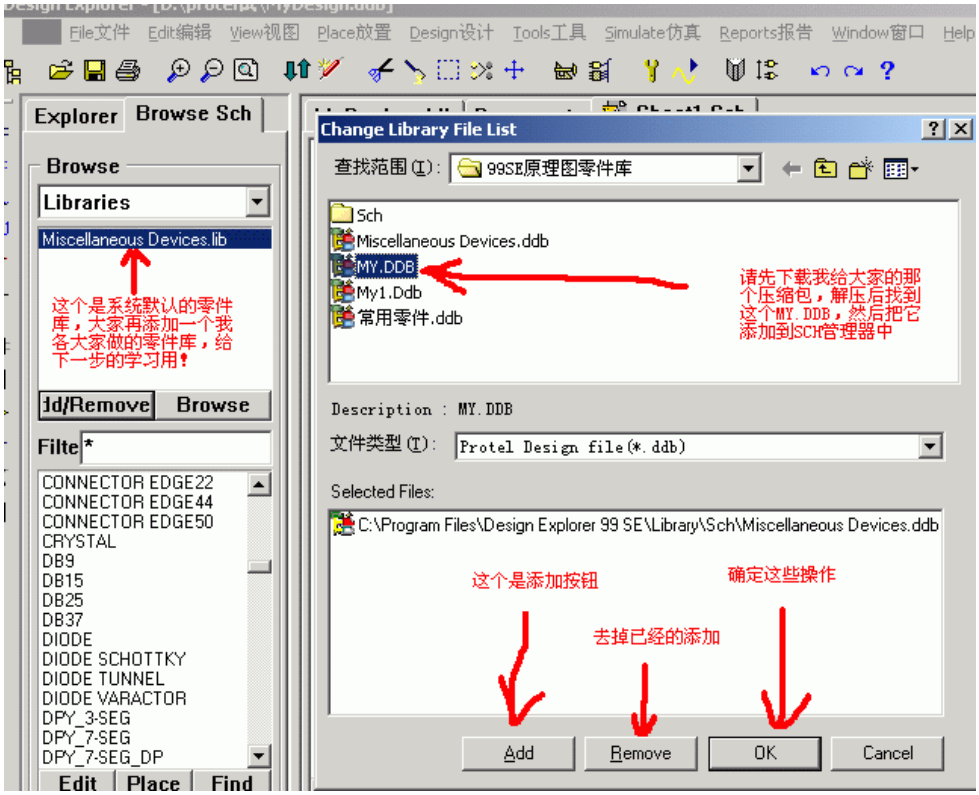
5. 新建一个*. SCH 文件（注意看一下那些中文的文字注释）



6. 添加新的零件库. GIF

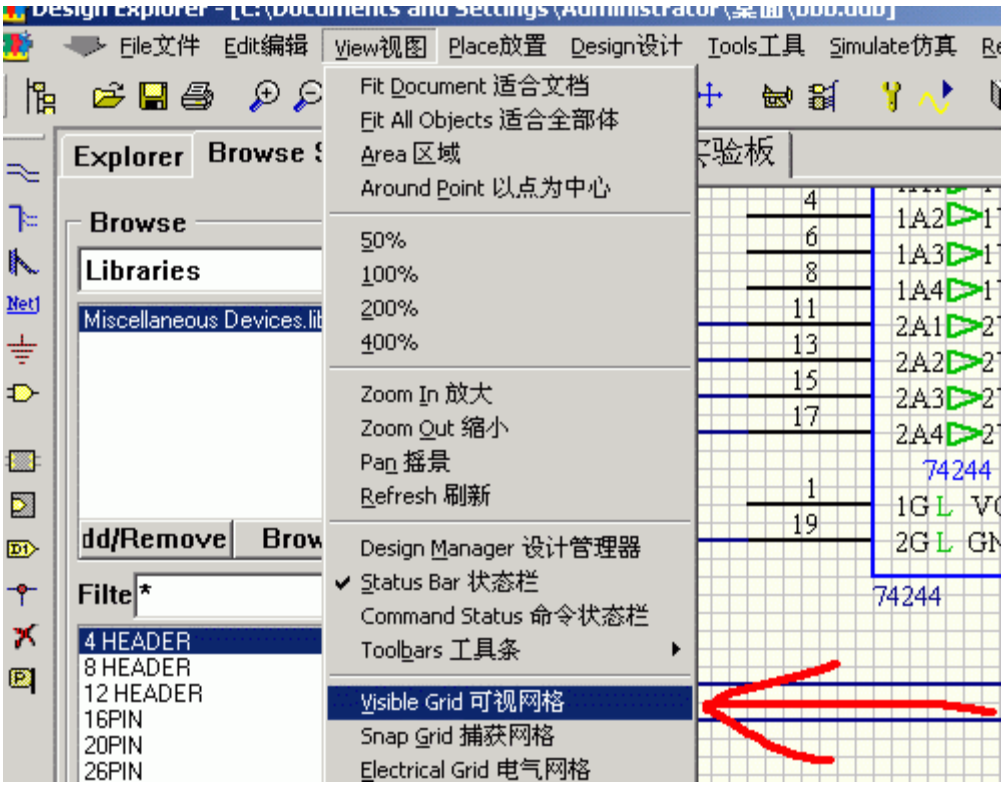


7. 下面这个就是具体的添加方法了



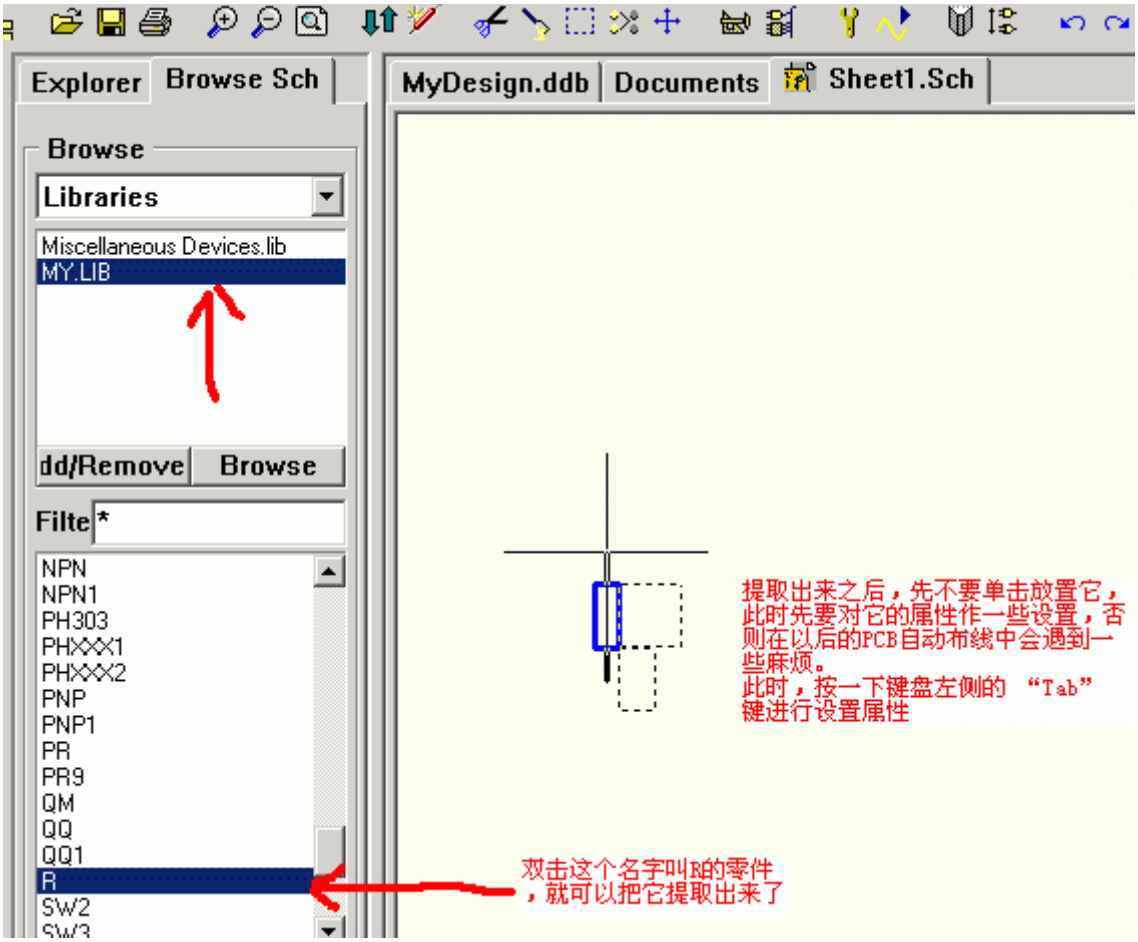
开始画原理图啦！

首先要先设置一下，去掉讨厌的网格显示



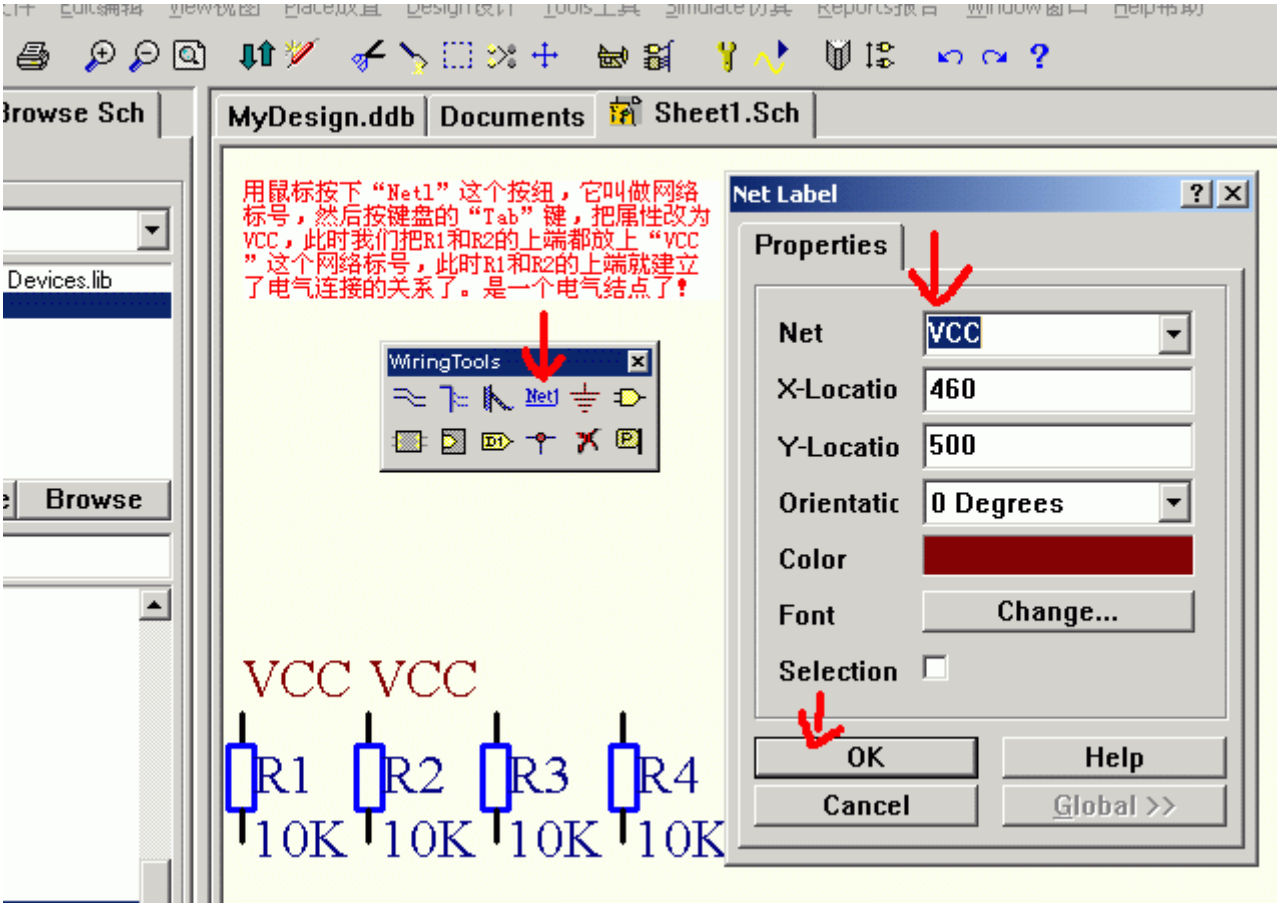
在这里我们利用先前添加好的 SCH 零件库做一个简单的 SCH 格式原理图，然后进行自动布线

1. 如何调出 SCH 零件进行并且进行属性设置

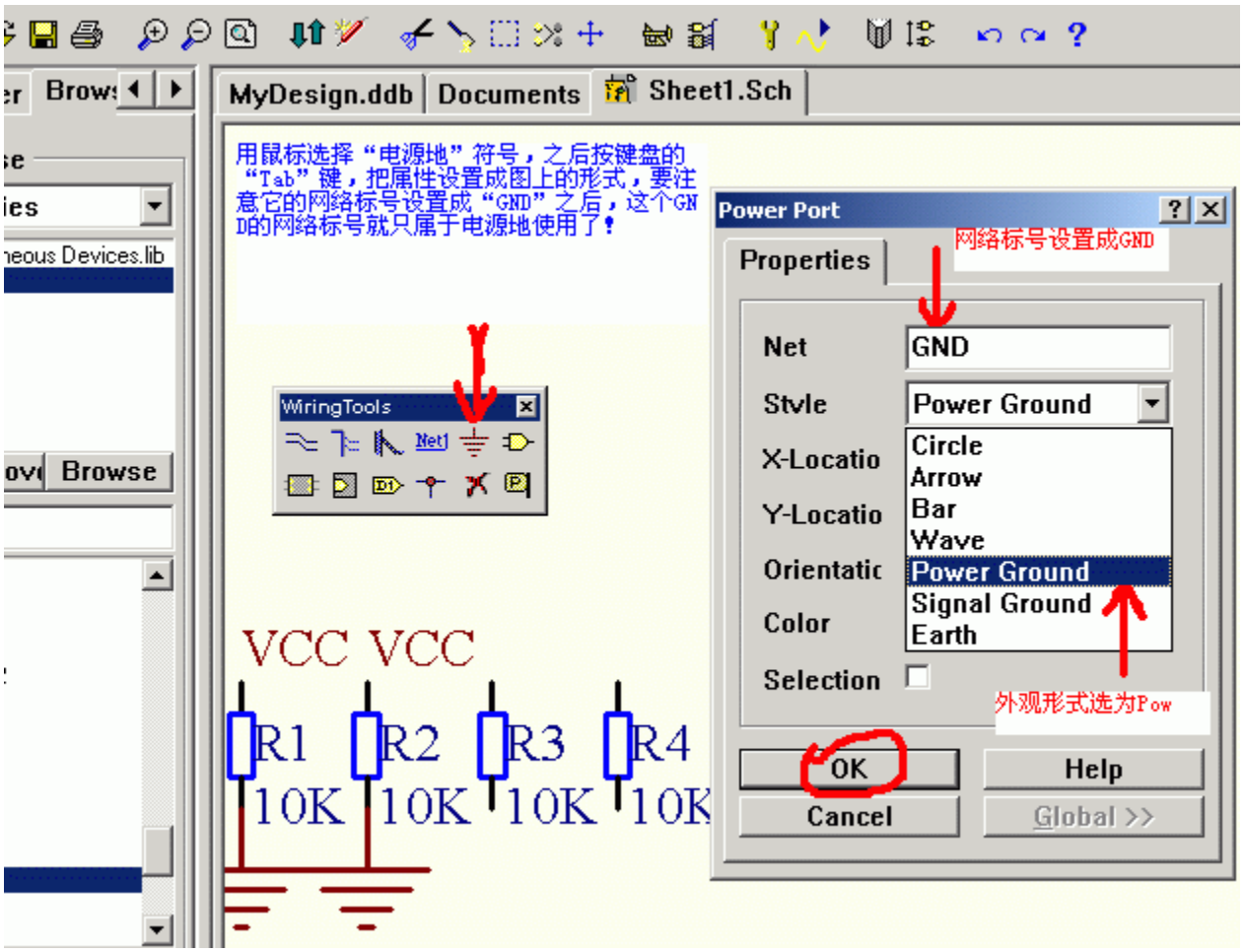


2. 如何正确的设置 SCH 零件的属性

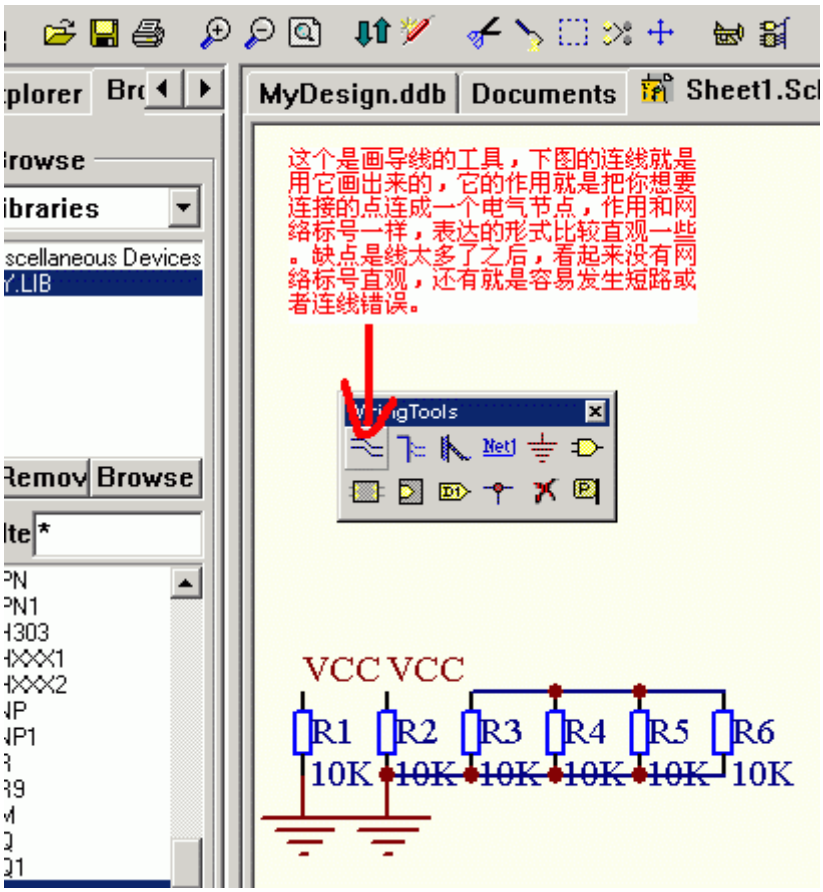
3.一个必须学会的操作，那就是网络标号的使用，SCH 可不是单纯的画图板



4. 电源地的设置，这可是整个电气系统的半边天了

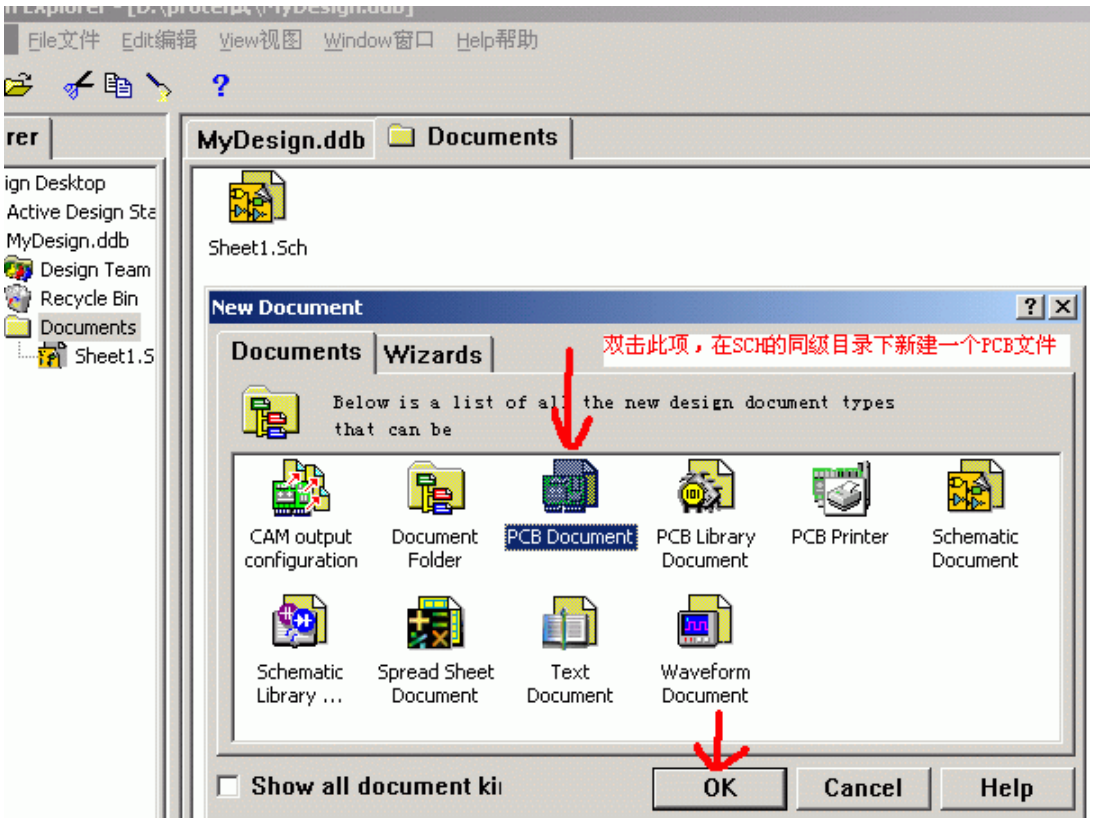


5. 连线工具，和网络标号的和网络标号左右一样，更直观一些，属于使用频率很高的工具

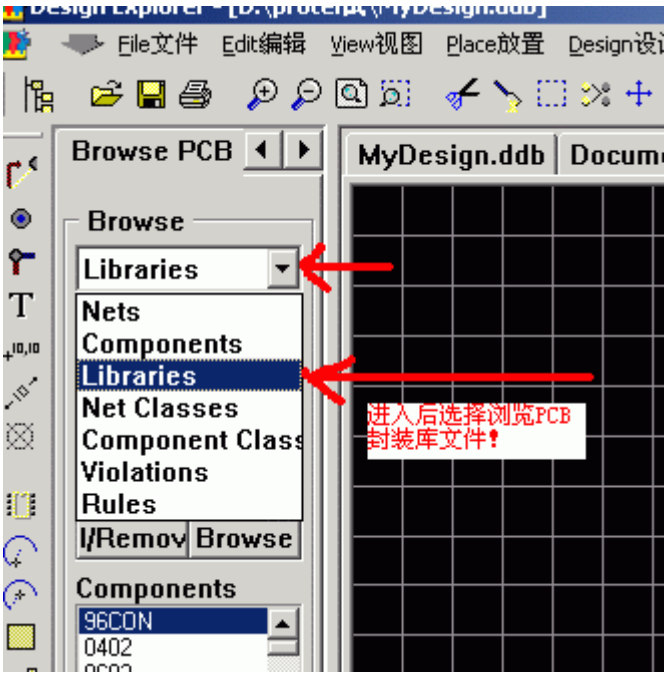


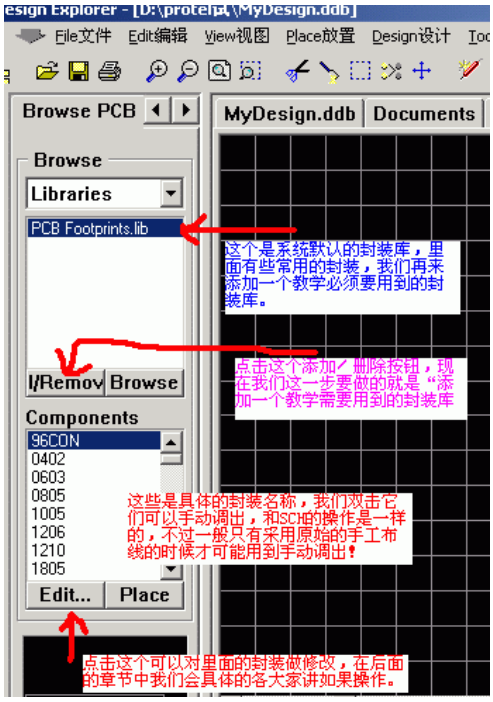
到这里我们就已经画好一个简单的电子电路原理图了，它具备了电源、负载、电气连接关系，三个最基本的要素，下面我们来看看如何把它快速的变成 PCB 电路板！>>

1.在 Documents 目录下新建一个*.PCB 文件，这样做的目的是要让*.SCH 和*.PCB 在同一目录下

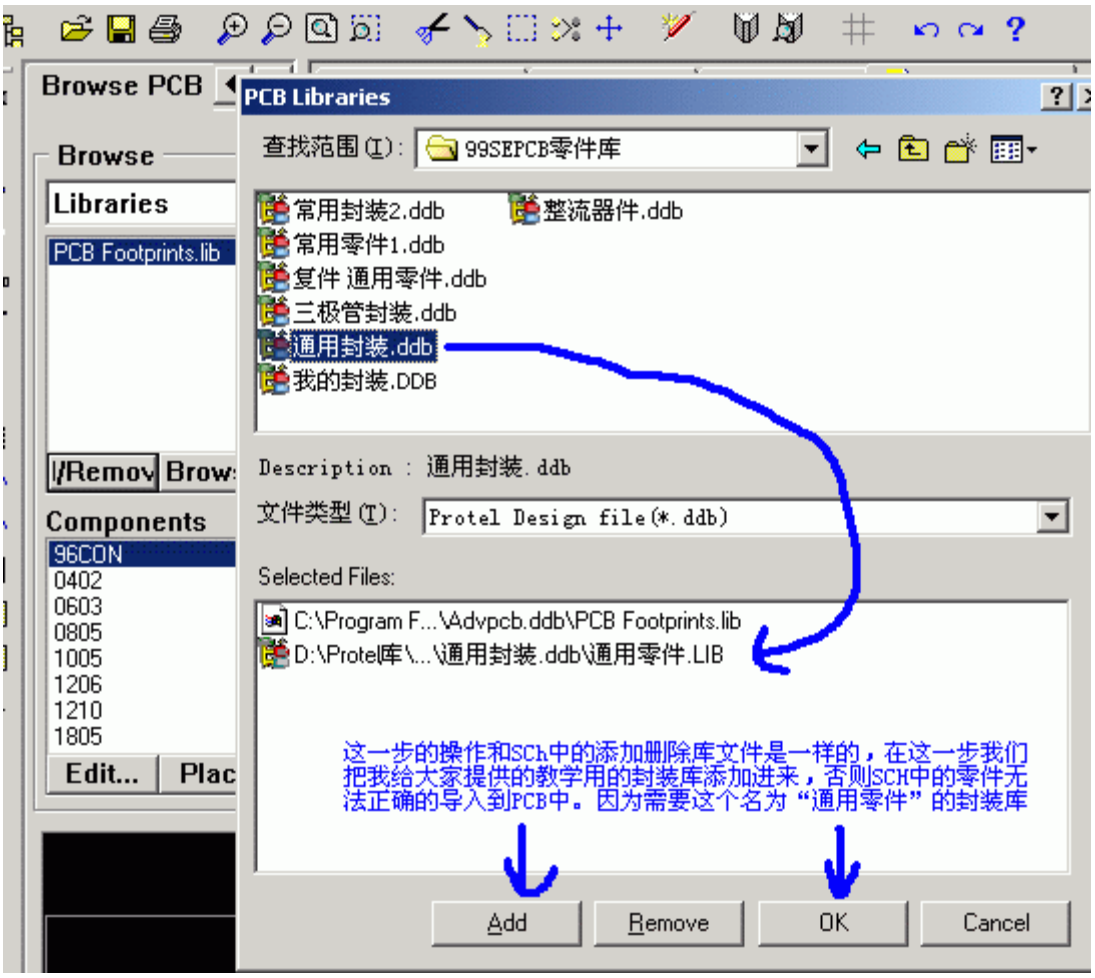


2. 添加自动布线要用到的封装库

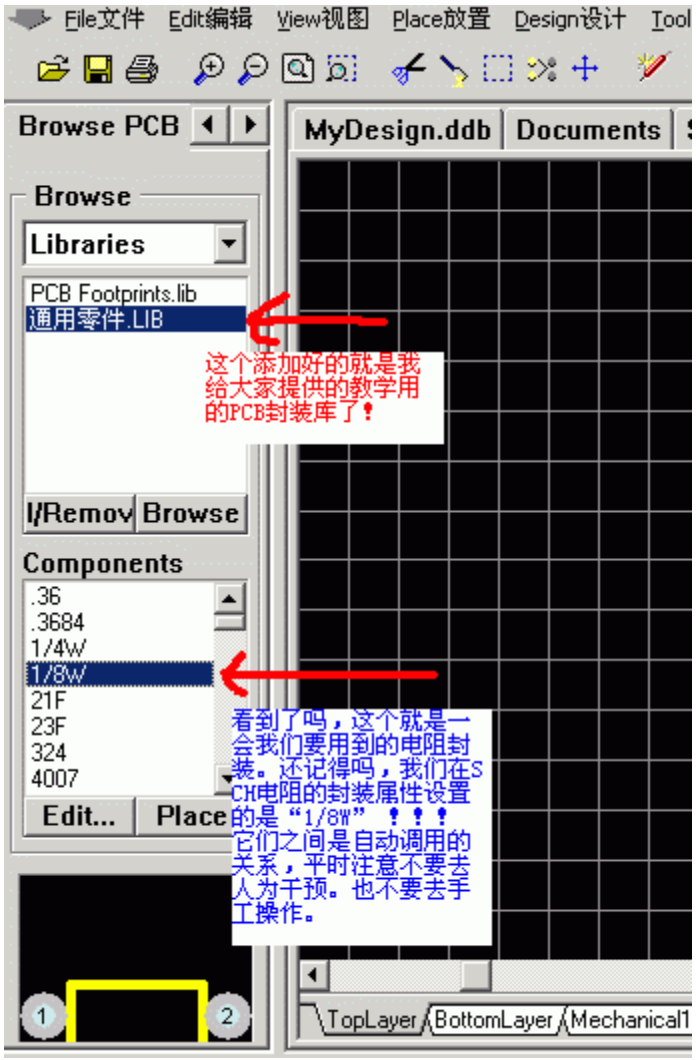




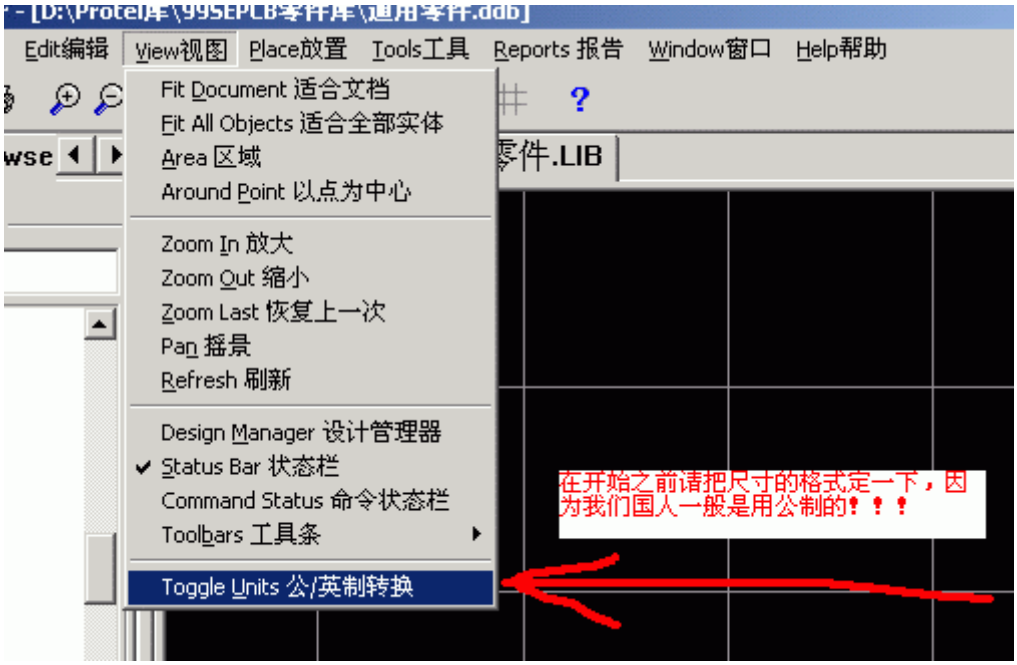
3. 添加我给大家准备的用于教学的封装库，点击下载它，下载后用 WINRAR 解压


Browse PCB
Browse
Libraries
PCB Footprints.lib
Components
96CON
0402
0603
0805
1005
1206
1210
1805
Edit...
PCB Libraries
查找范围(I):
99SEPCB零件库
常用封装2.ddb
整流器件.ddb
常用零件1.ddb
复件 通用零件.ddb
三极管封装.ddb
通用封装.ddb
我的封装.DDB
Description : 通用封装.ddb
文件类型(T):
Protel Design file(*.ddb)
Selected Files:
C:\Program F...\Advpcb.ddb\PCB Footprints.lib
D:\Protel库\...\通用封装.ddb\通用零件.LIB
这一步的操作和SCh中的添加删除库文件是一样的，在这一步我们把我给大家提供的教学用的封装库添加进来，否则SCH中的零件无法正确的导入到PCB中。因为需要这个名为“通用零件”的封装库
Add
Remove
OK
Cancel

4.这是添加好后的效果

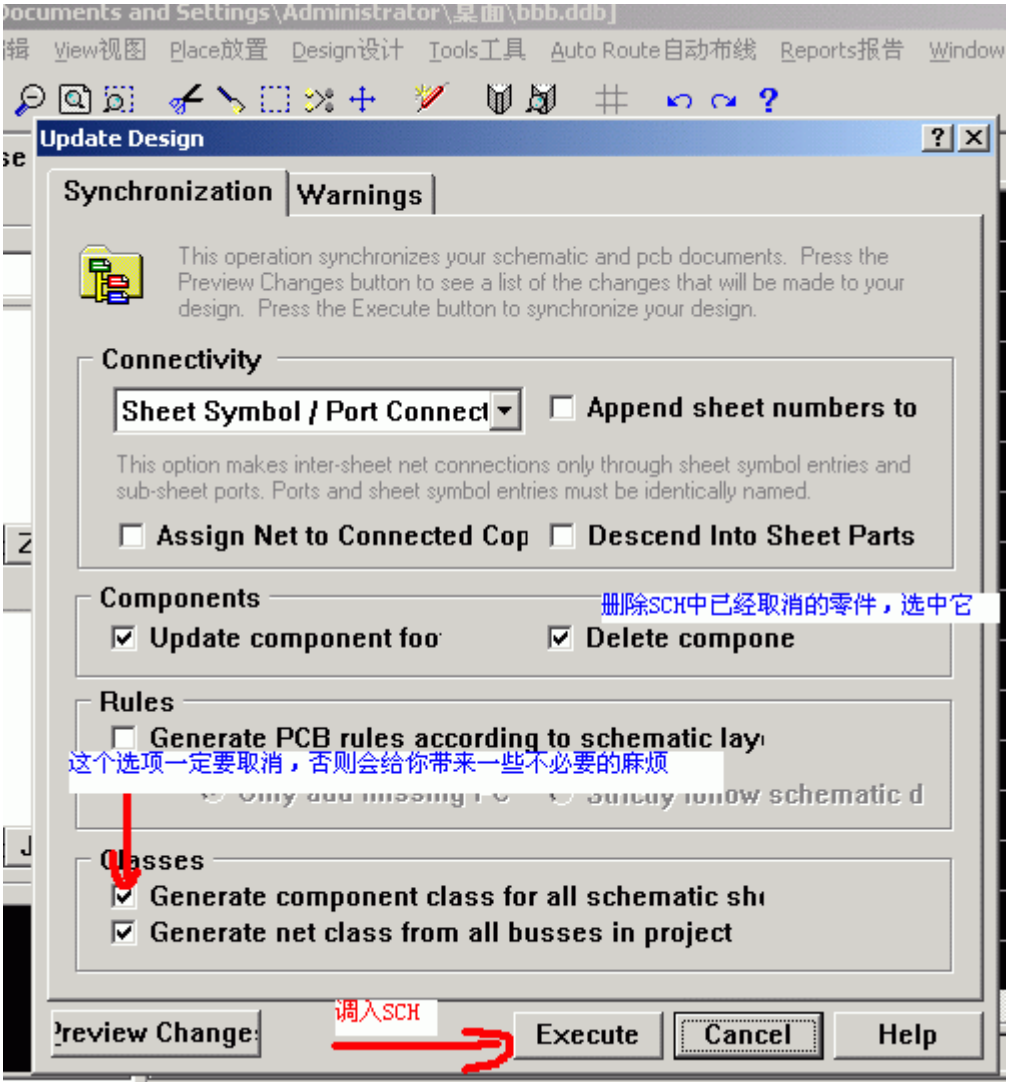


5. 一定要把尺寸单位转换一下, 英制实在是不爽

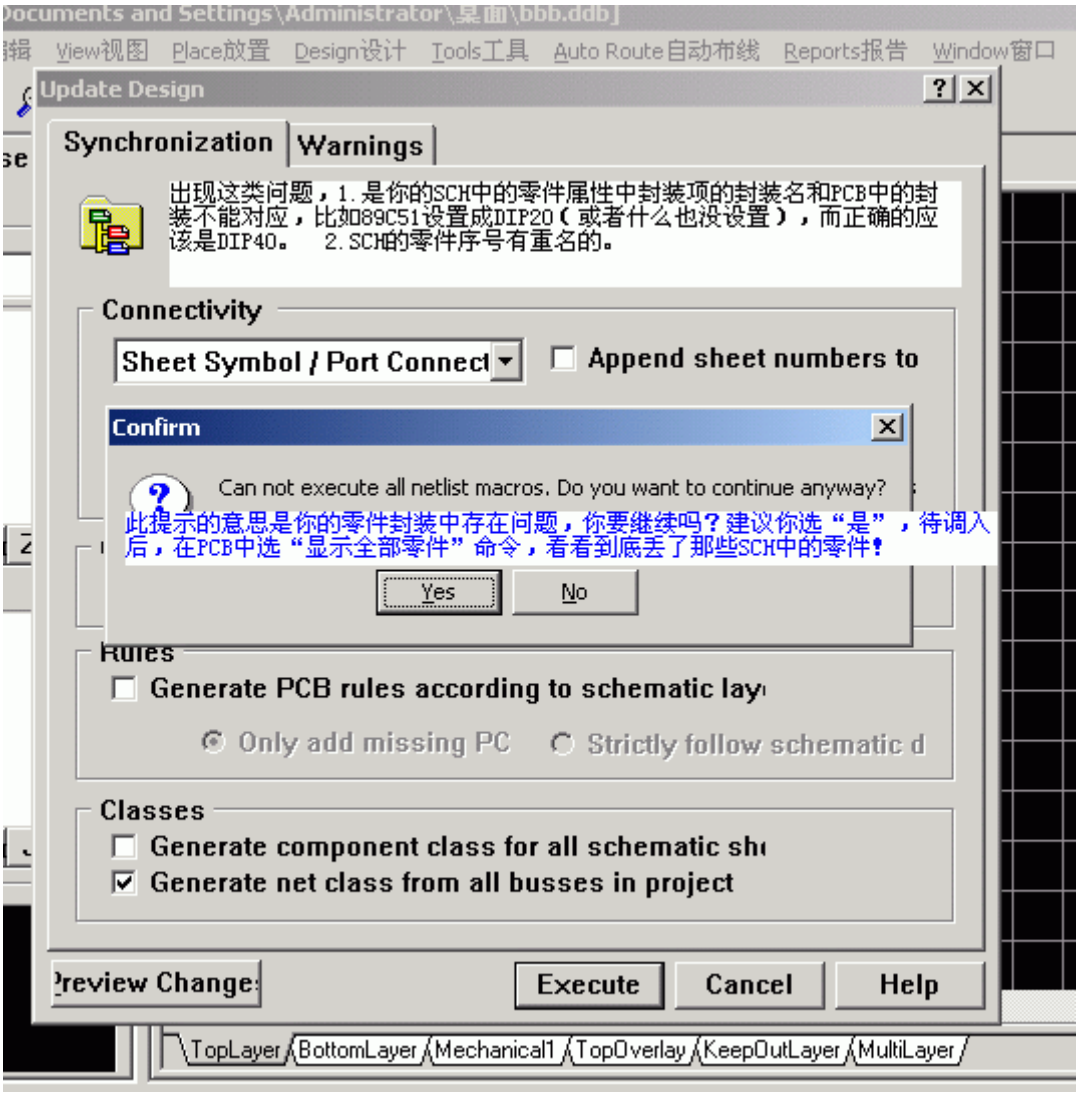


到这里准备工作就做好了,进入 SCH 到 PCB 的转变>>

3.注意看一下中文注释

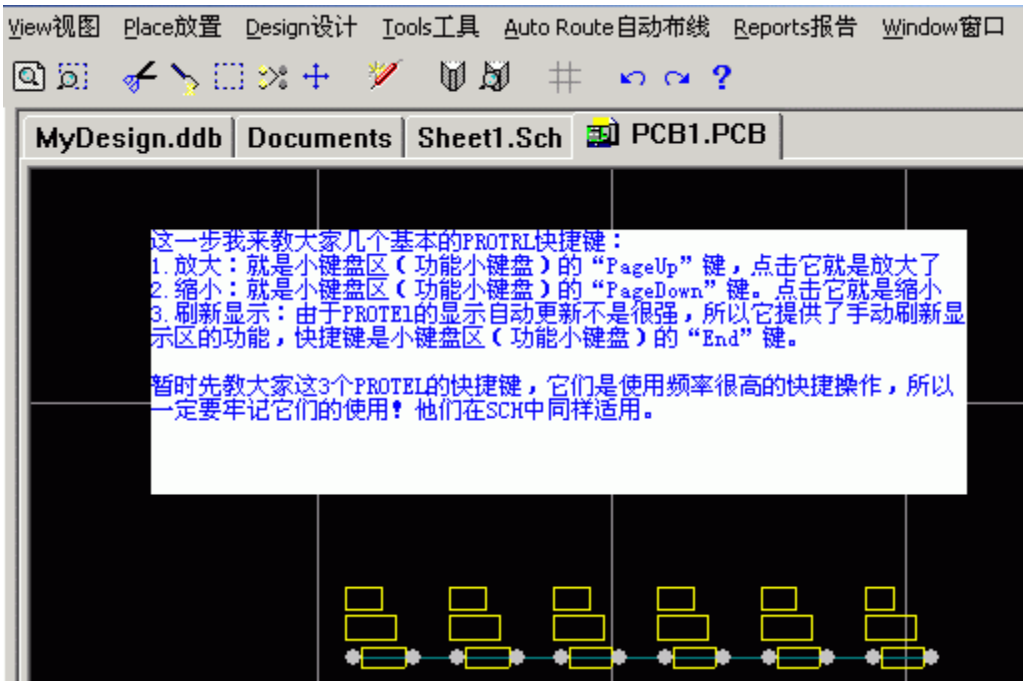


4. 如果遇到这个问题，说明 SCH 的里面还存在小的问题，注意看中文注释

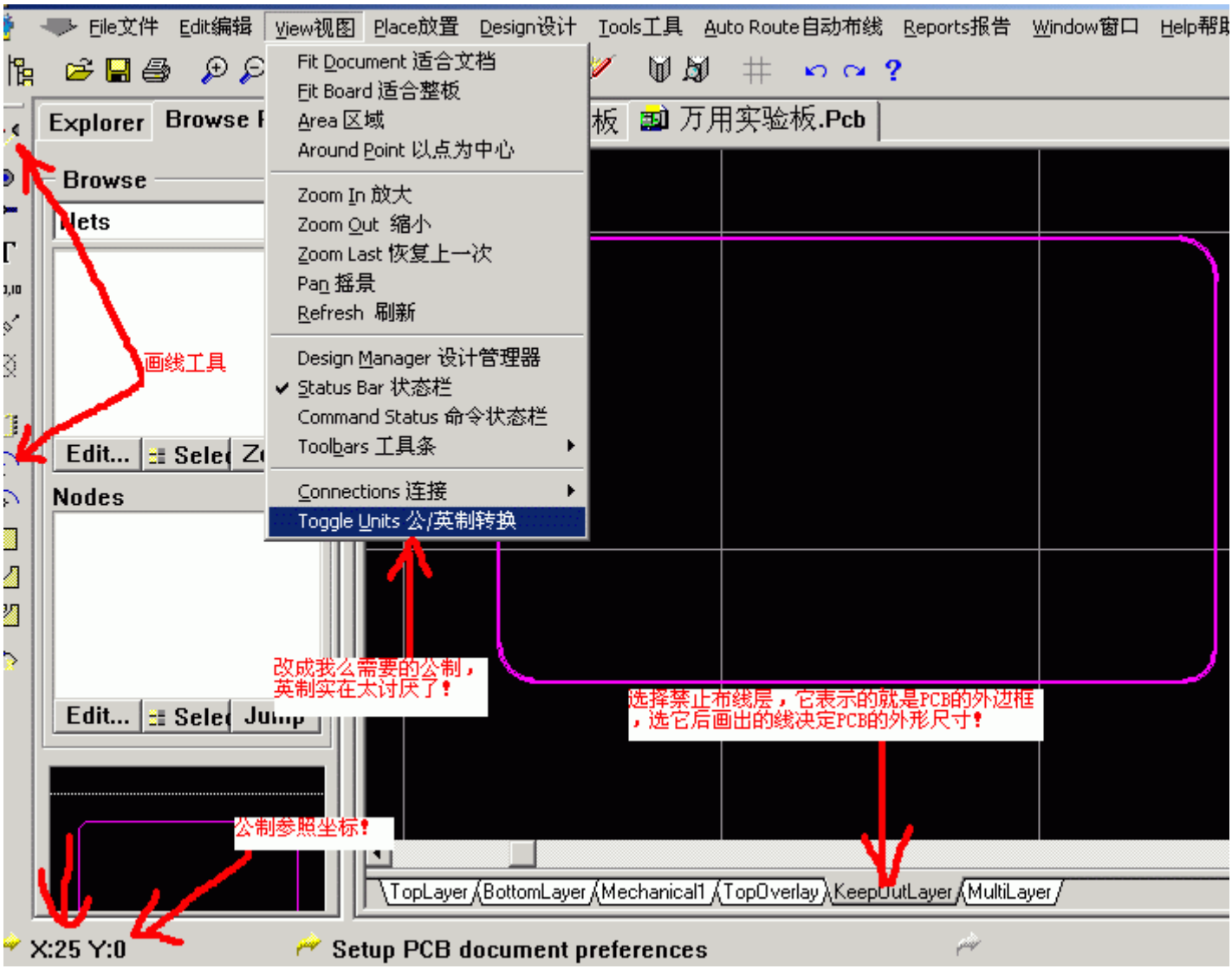


5. 这是成功导入后的显示方面的一些技巧

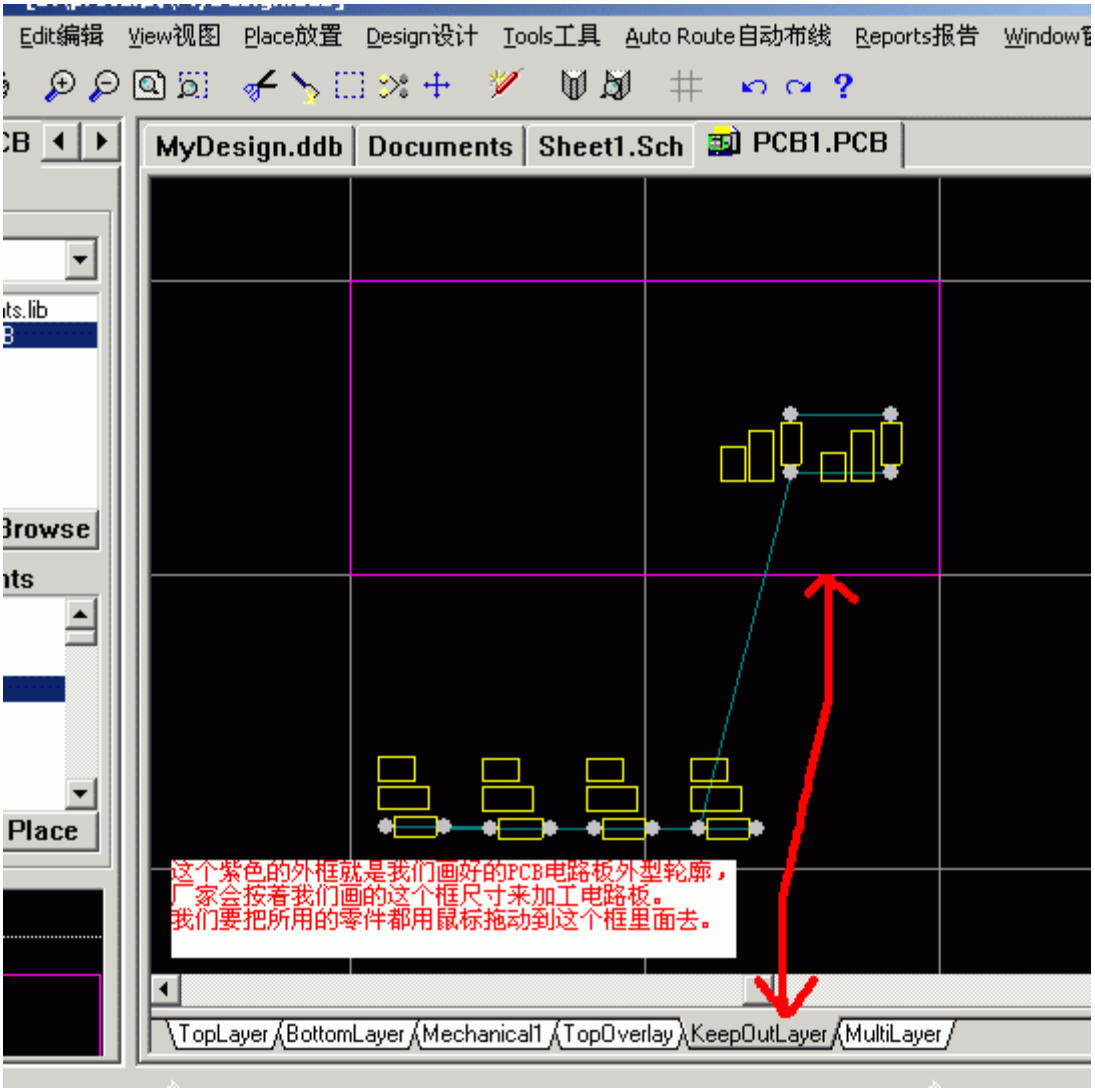
6. 一些常用的技巧，补充一点，如要旋转元件的话，只要用鼠标按住元件然后按压键盘“空格键”即可



7. 画一个 PCB 的外型框

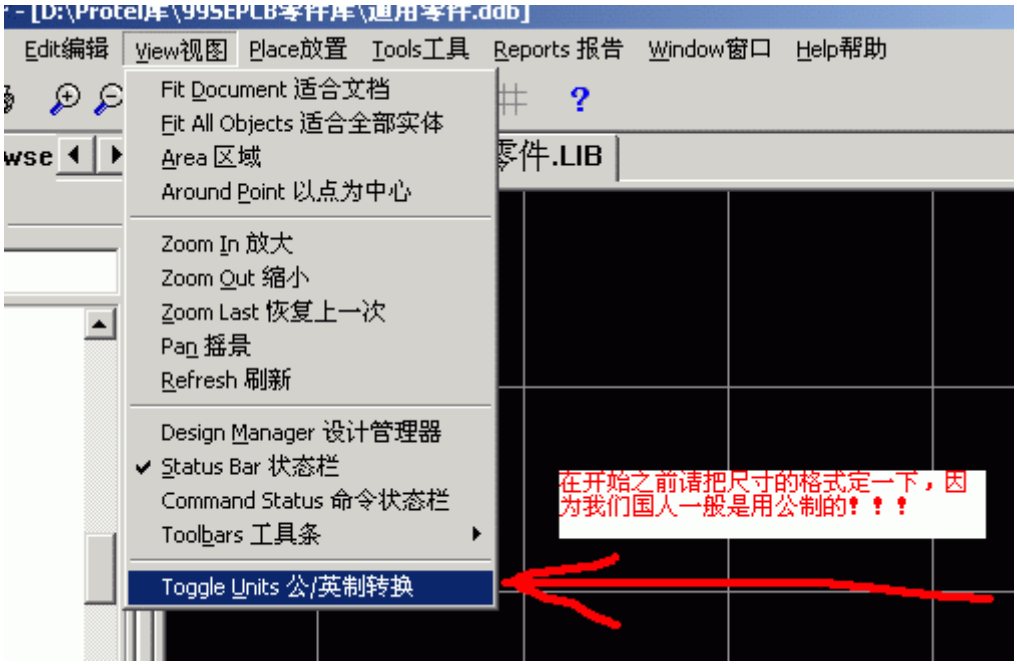


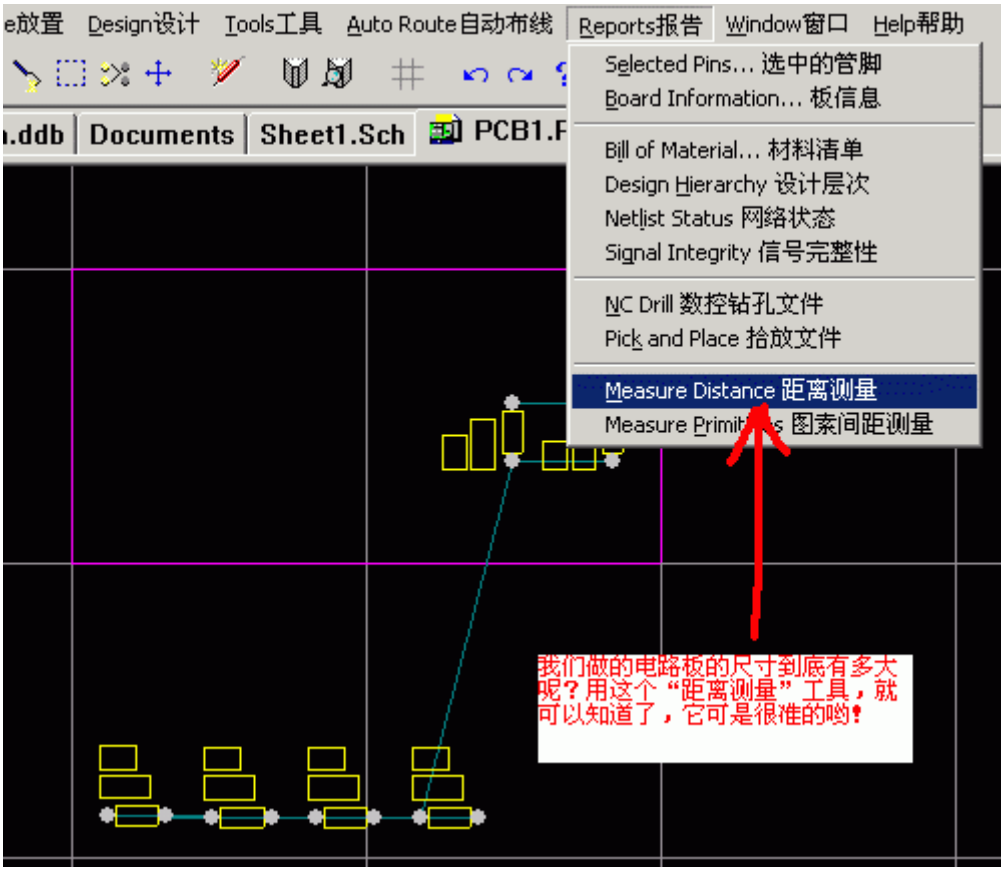
8. 做一个自己要的外型框，然后把 PCB 零件封装移动到里面去



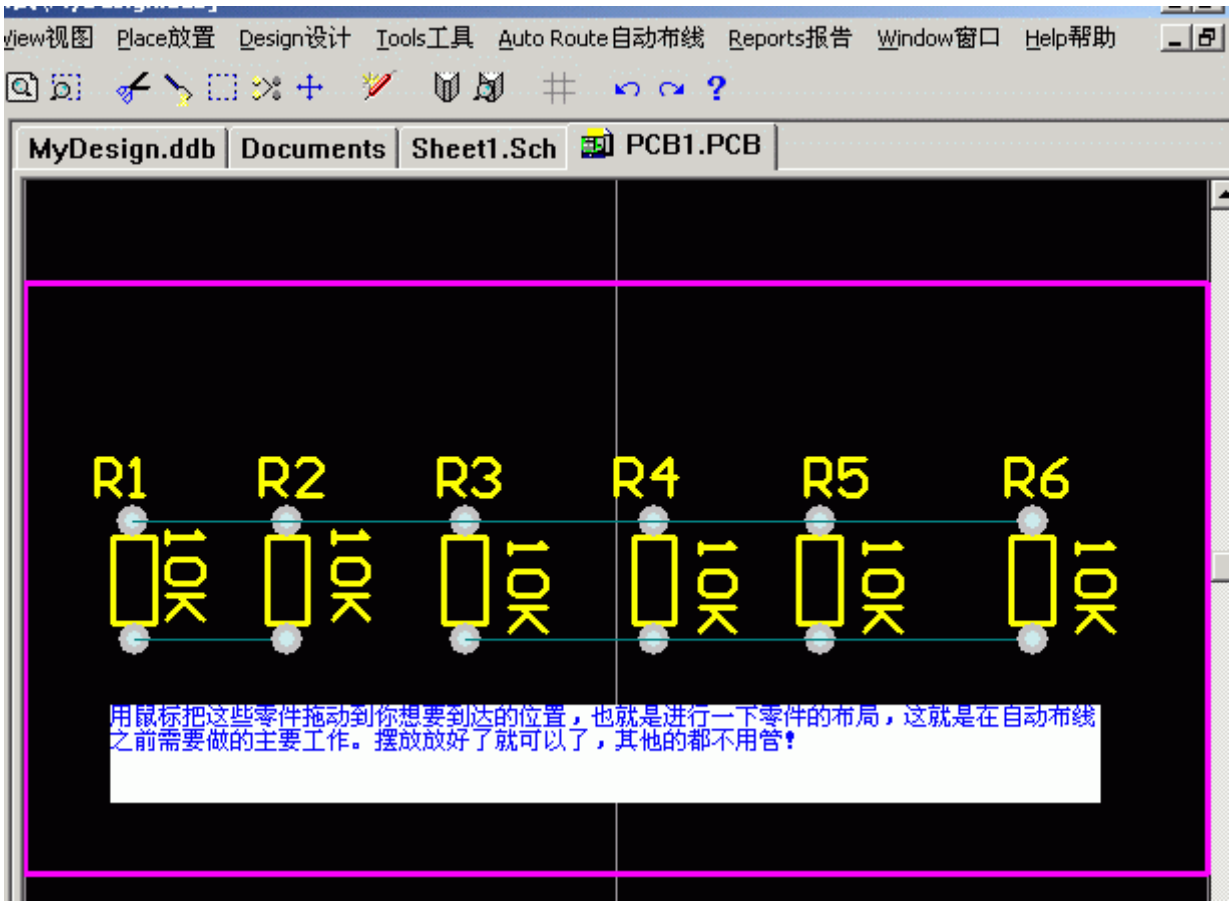
到了这里，我们就可以进入自动布线的操作了

1. 可以先看看尺寸是不是合适

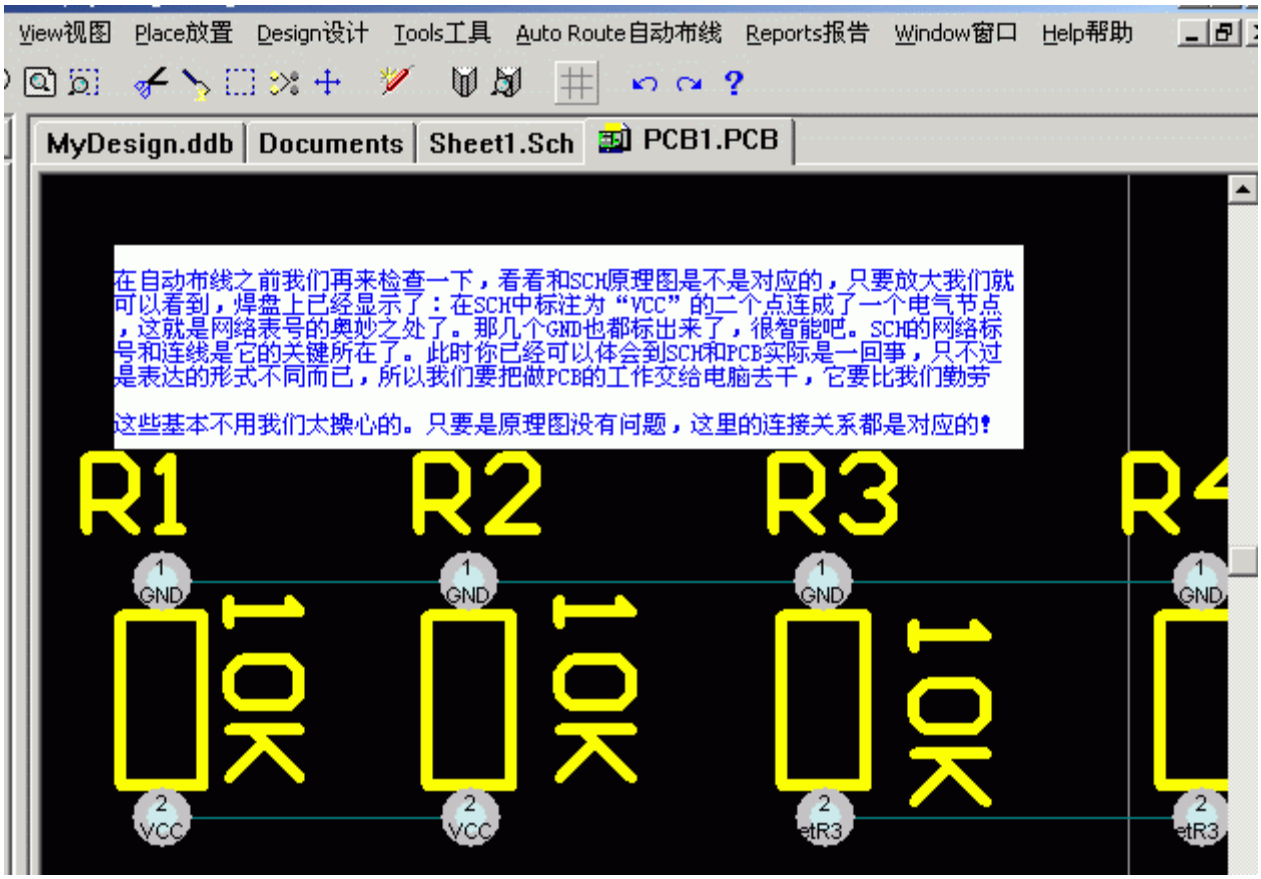




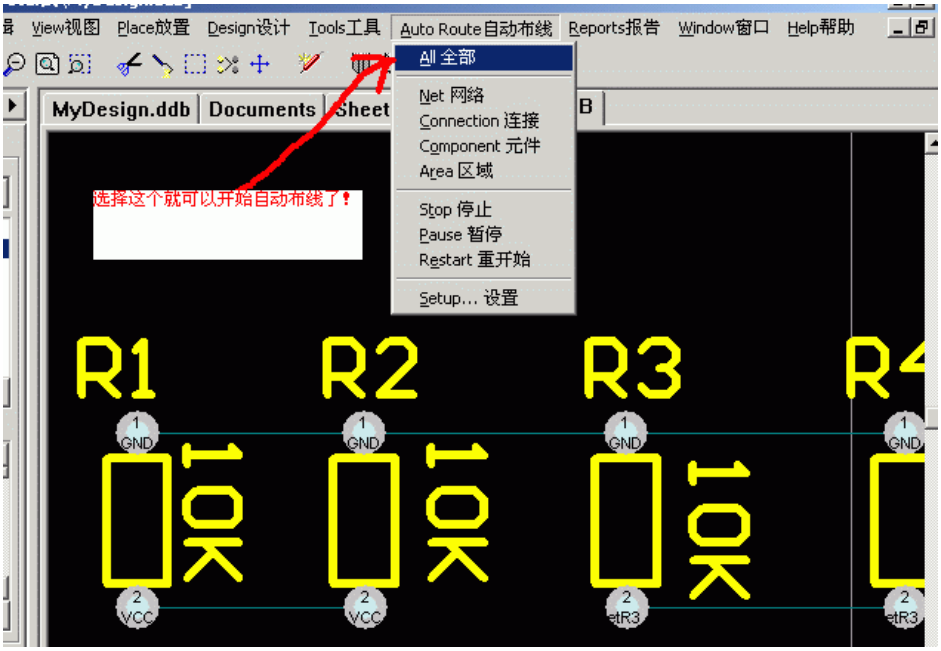
2.对元件进行一下布局,就是用鼠标拖动元件而已,键盘的“空格键”负责翻转元件



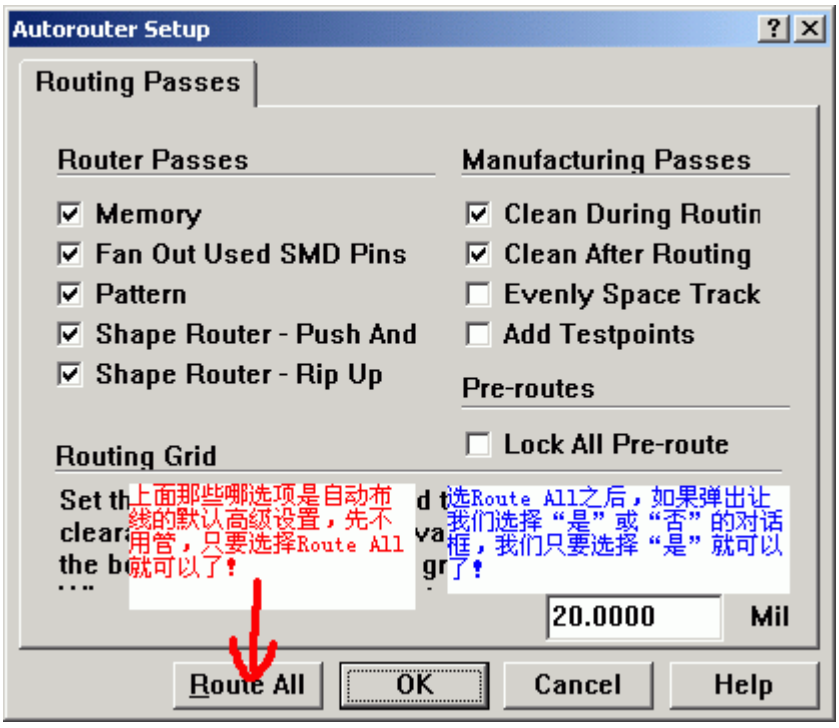
3.自动布线之前要校验一下，看看是不是有错误！



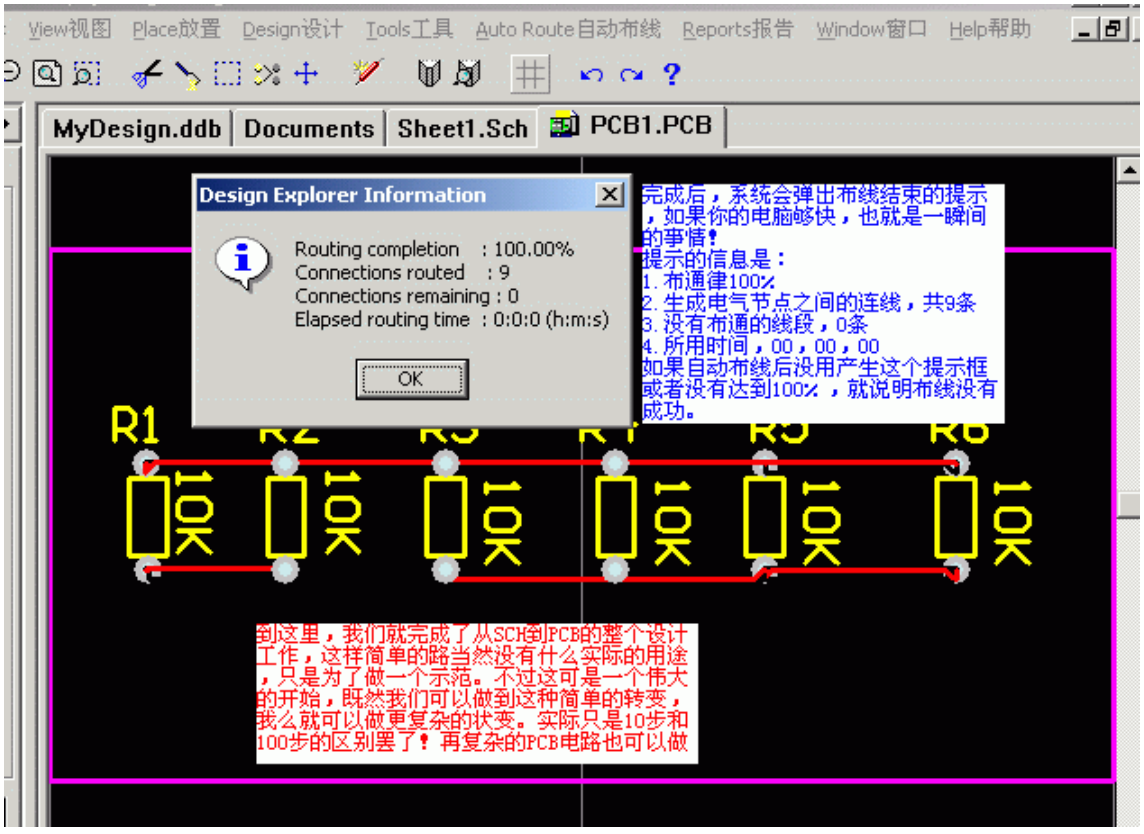
4. 可以开始自动布线了



5. 这是自动布线之前的最后一步

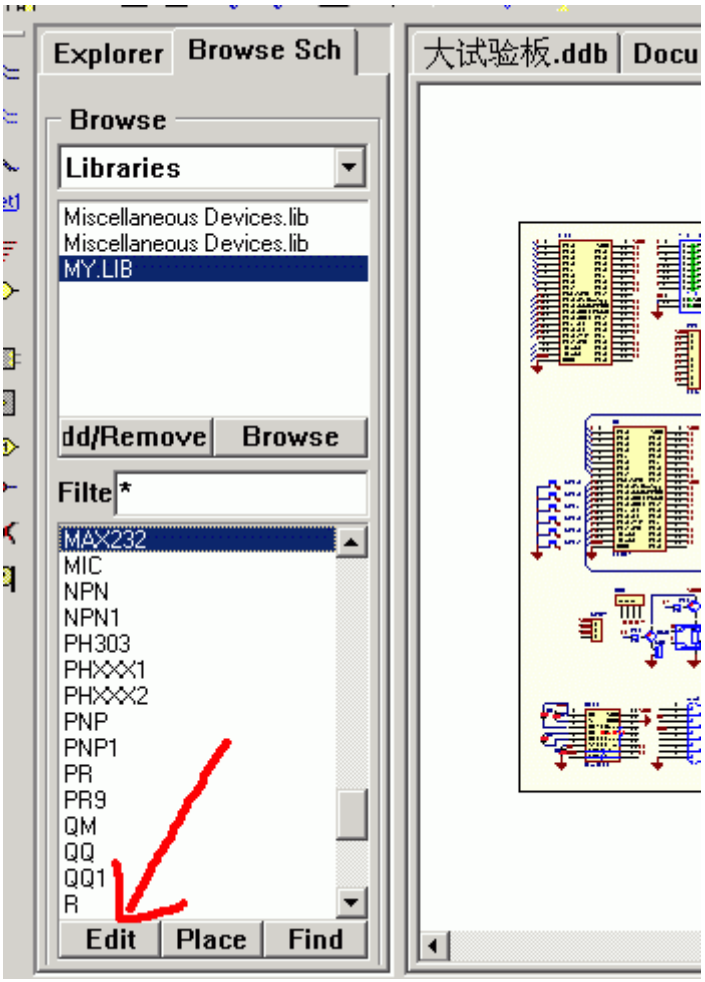


6. 自动布线完成了，到这里你已经完成了对 PROTEL99SE 的一次快速穿越！

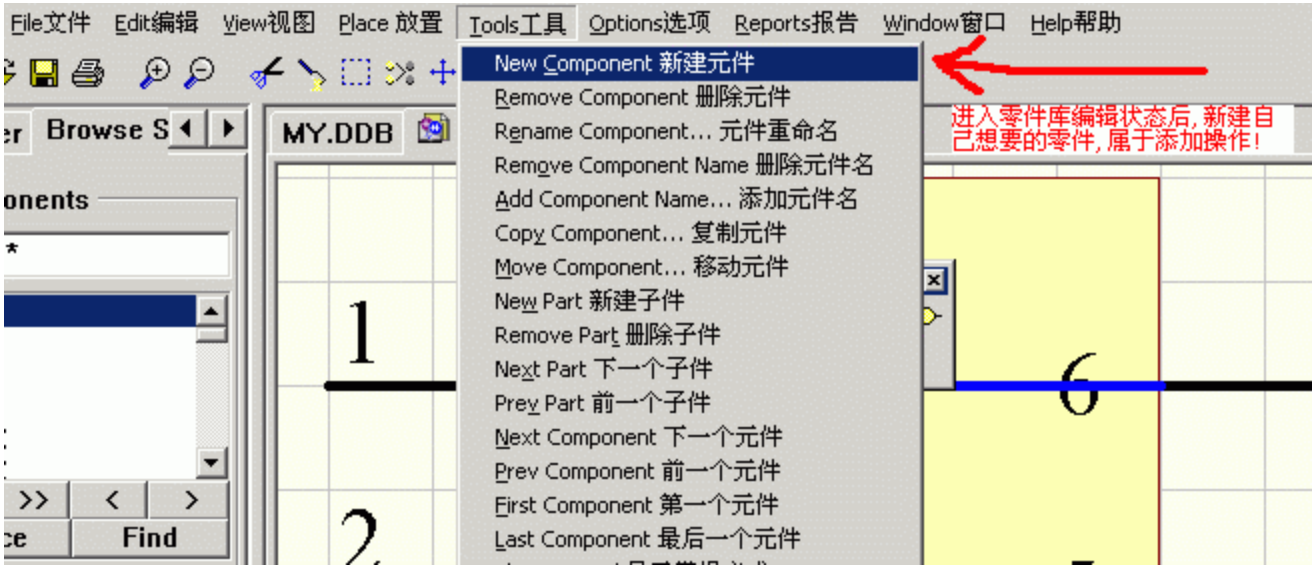


下一步我们来看看 SCh 零件库中零件是如何创建出来的，
做 SCH 零件了，很关键的一步，自力更生才可能走得更远！

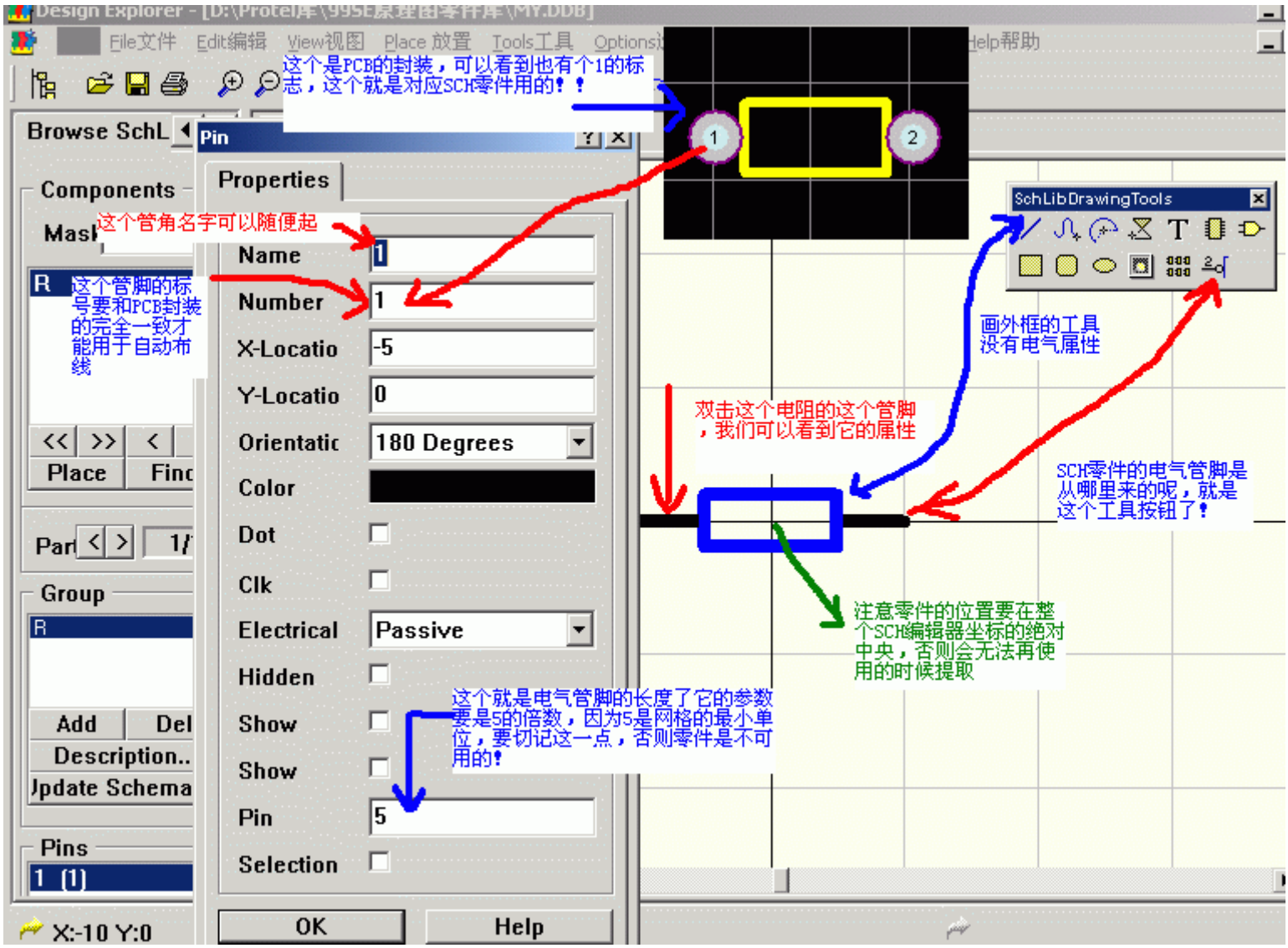
1. 先来打开 SCH 文件，选 SCH 零件库，然后选编辑，进入 SCH 零件编辑器



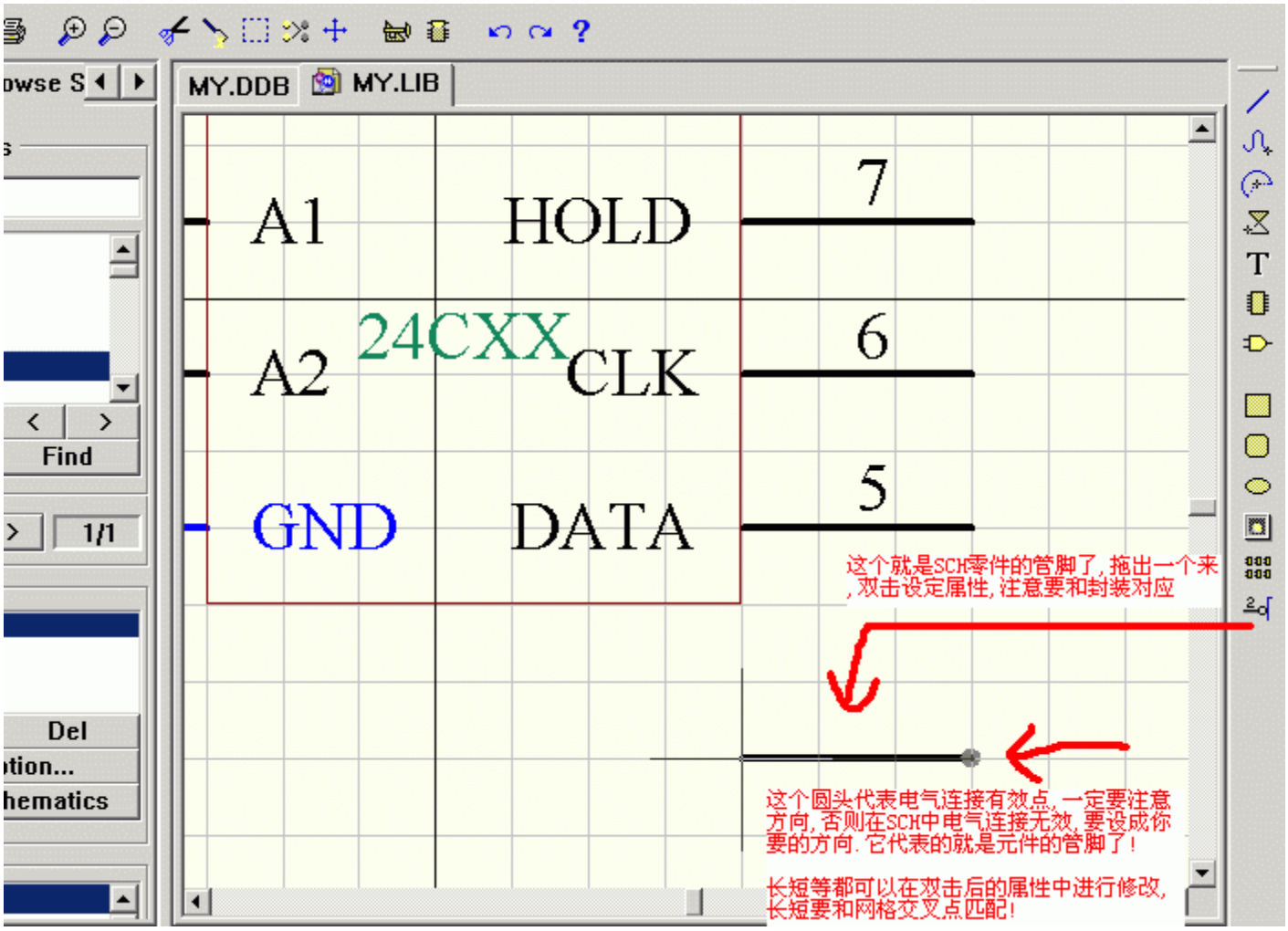
2. 在这个现有的库中新建一个 SCH 零件



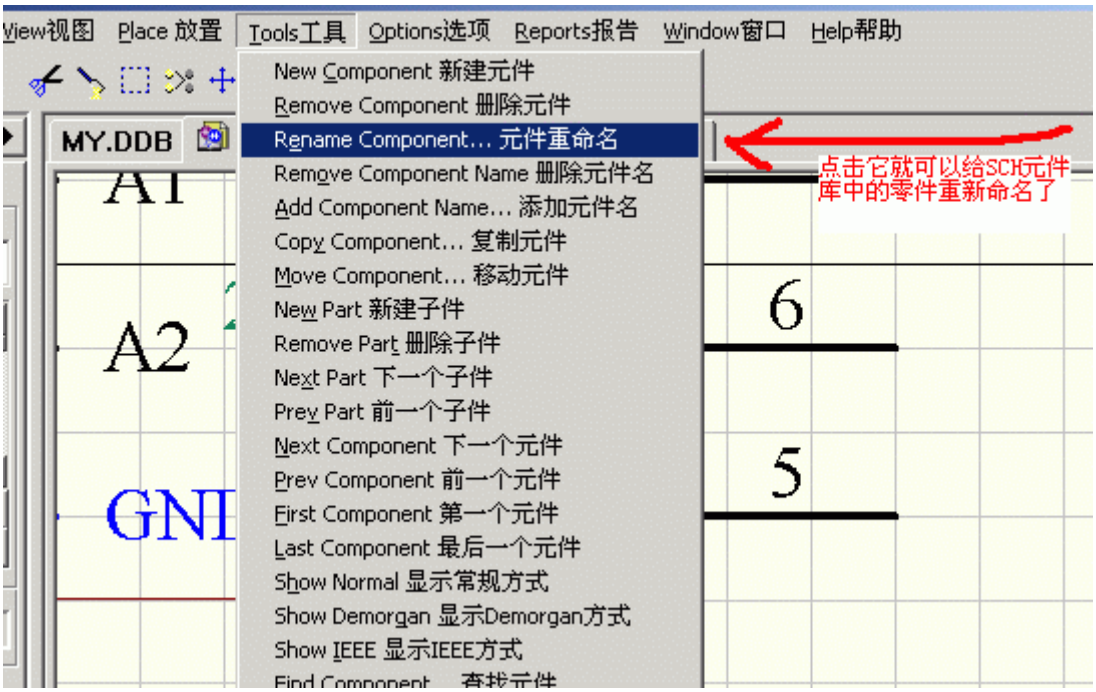
3. 先以做一个 SCH 电阻零件为例子说明一下，请注意看图中所有的中文注释！



4.要注意 SCH 零件的管脚的电气连接有效点是有讲究的！仔细看一下下面这个图，注意看中文注释！



5.用这个方法可以给零件库中的零件改名字

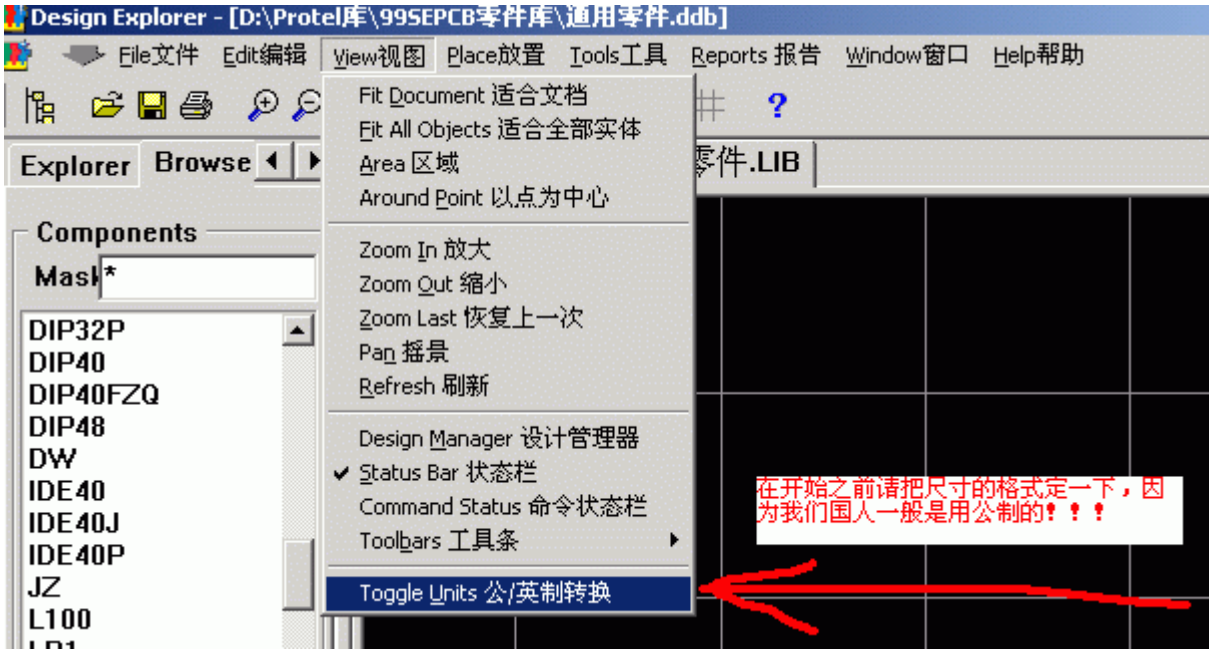


6. 最后是保存你的所有劳动成果，要提取的你的新零件需要重新启动 PROTEL99SE

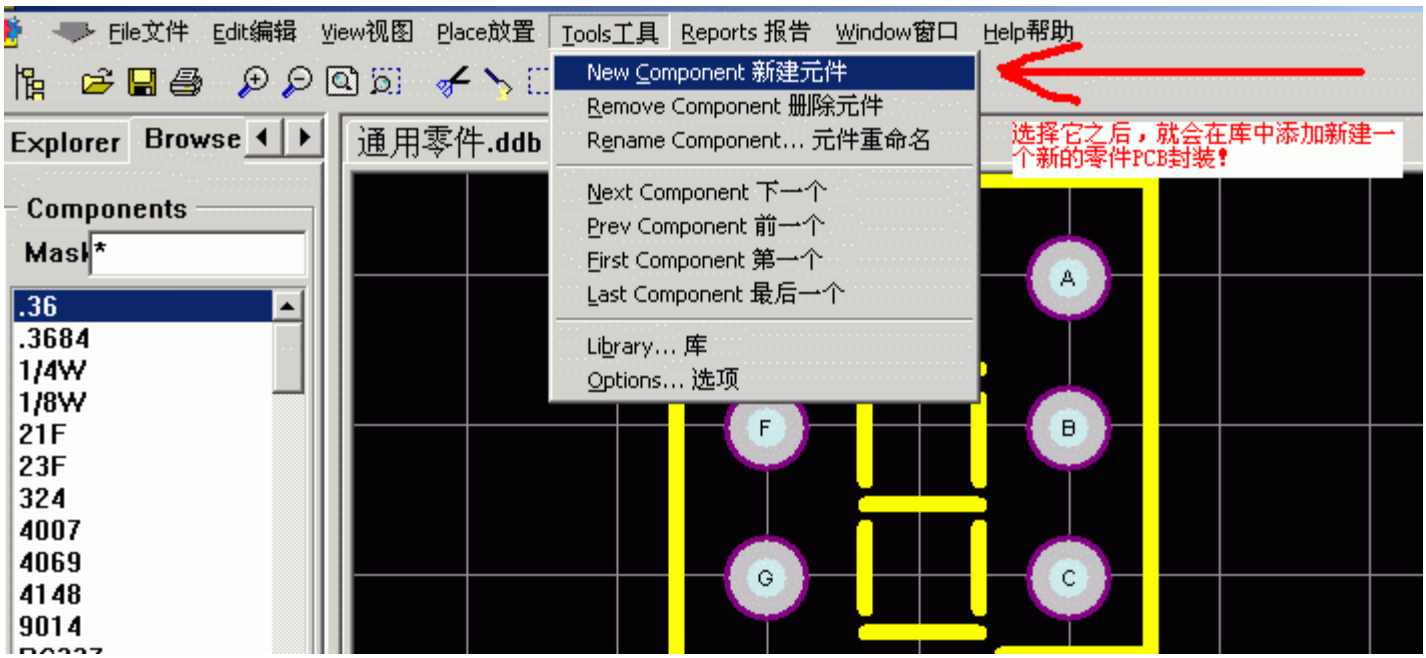
到这里，我们已经学会了如何来做 SCH 零件了，下面我们再来看看如何做一个 PCB 封装

1. 打开在前几课已经做过的 PCB，选择那个我给大家提供的封装库，
然后按着图选择编辑按钮就进入了 PCB 封装编辑器

2. 先把制式转换一下，改为公制

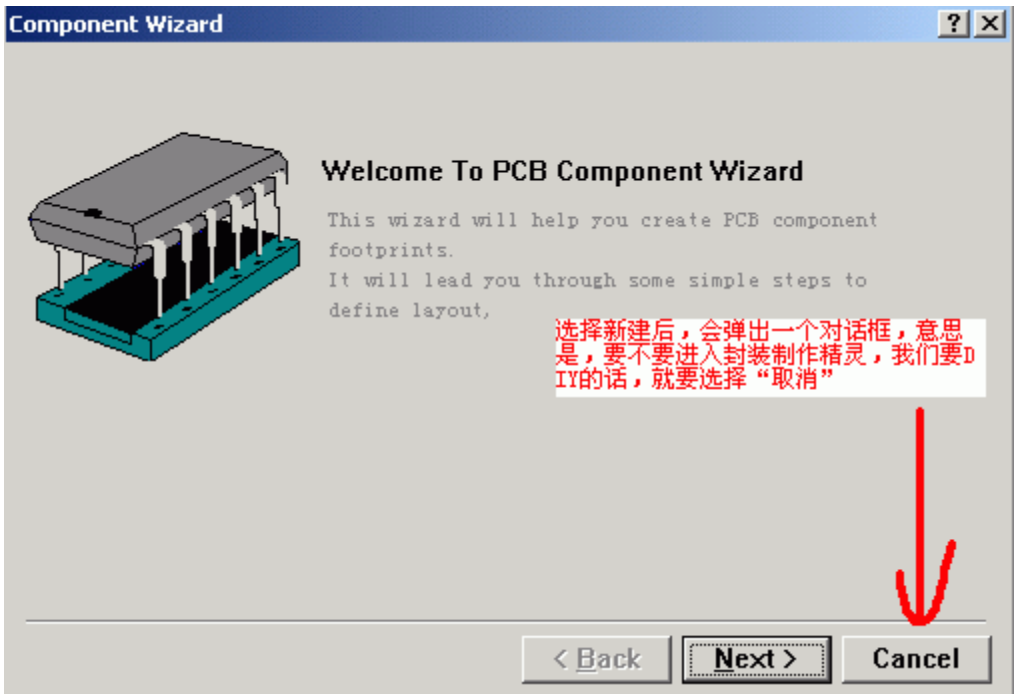


3. 新建一个 PCB 封装

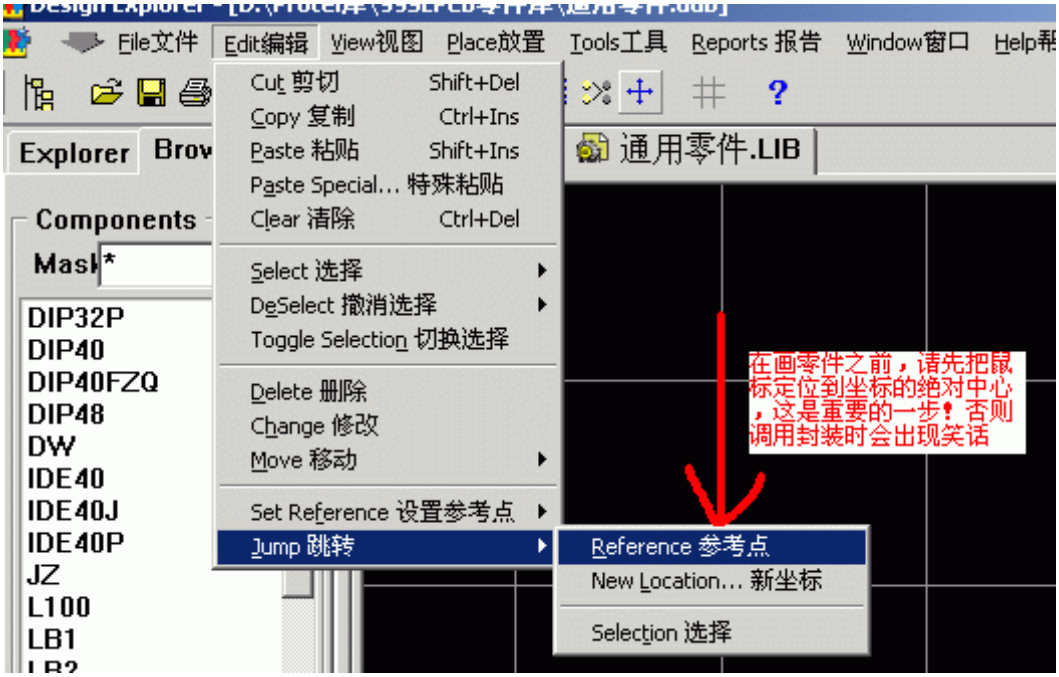


2.

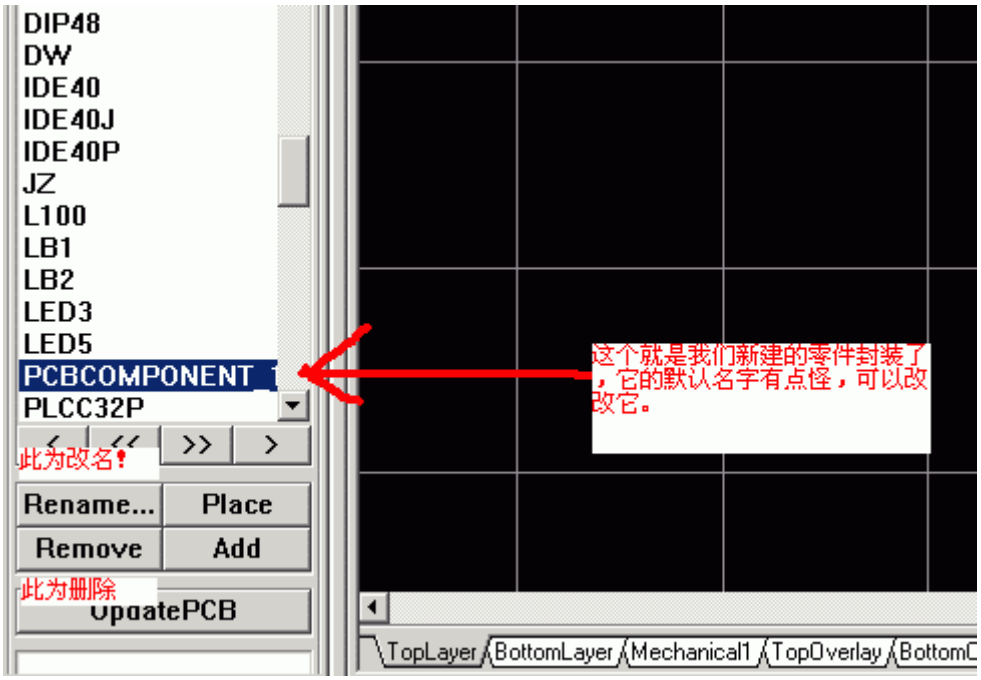
4. 之后会出现这个对话框，是一个傻瓜精灵，选择取消，因为我们要做一个完全属于自己的封装



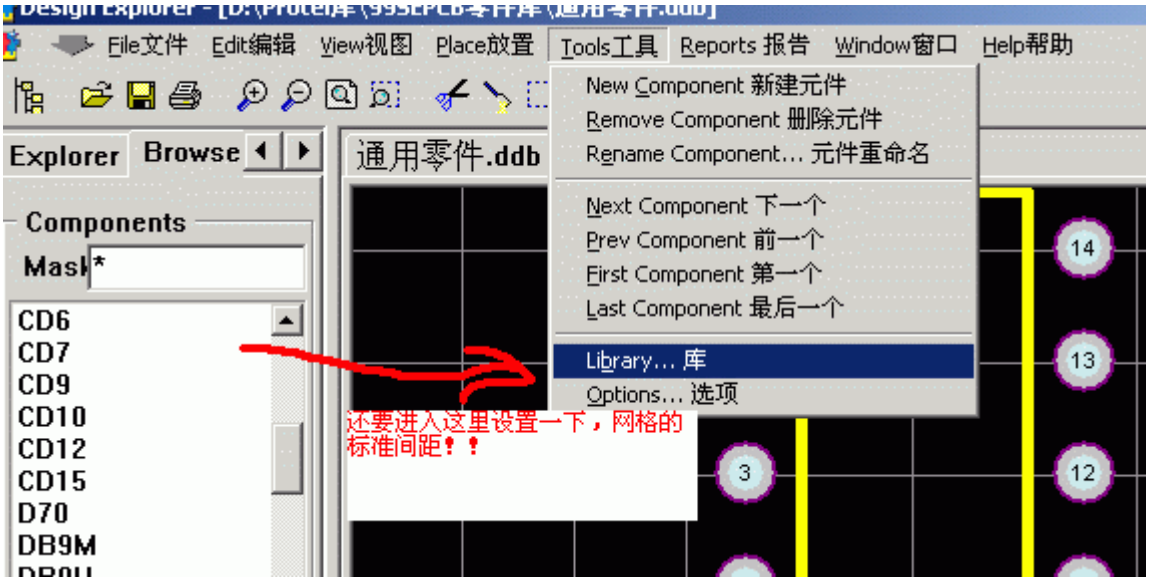
5. 注意：在做之前一定要把封装的起始位置定位成绝对中心，否者做好后的封装无法正常调用！



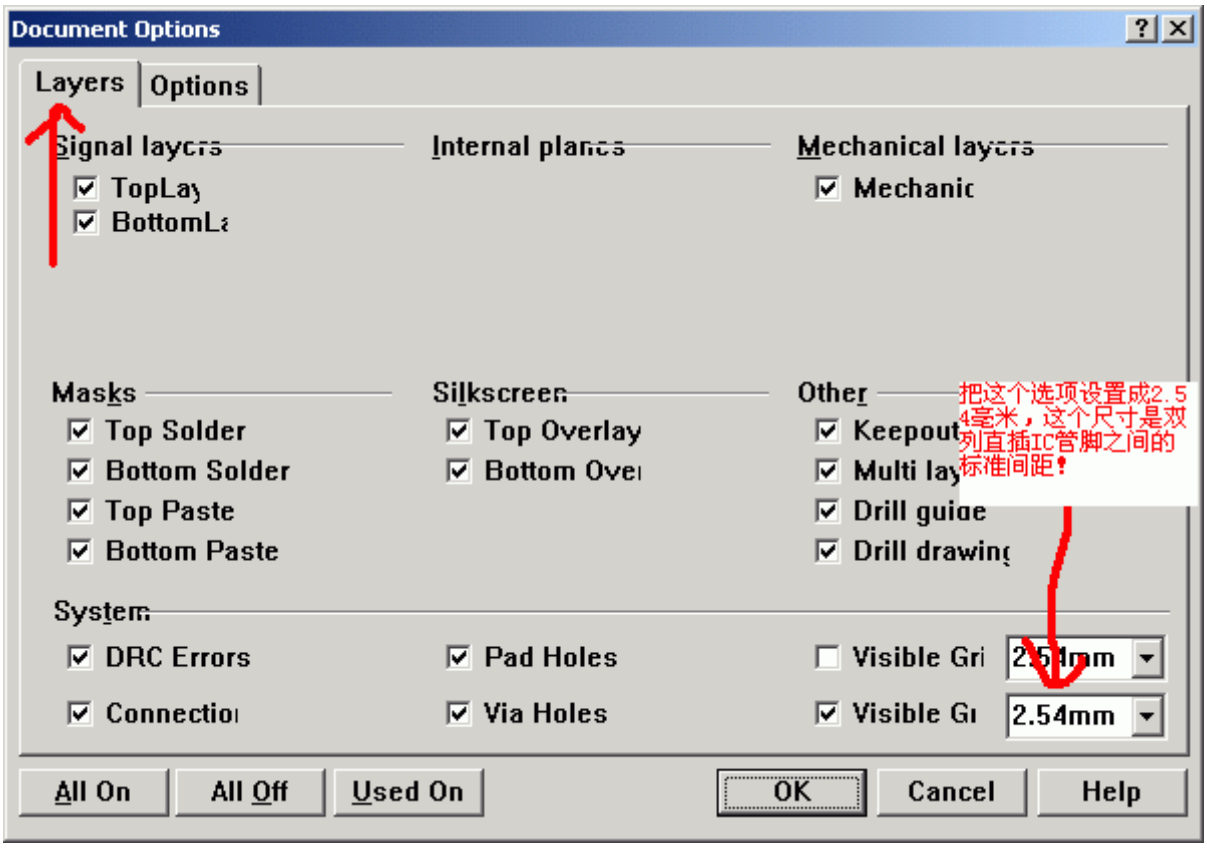
6. 如果对默认的封装名不满意，就需要改一个自己喜欢的

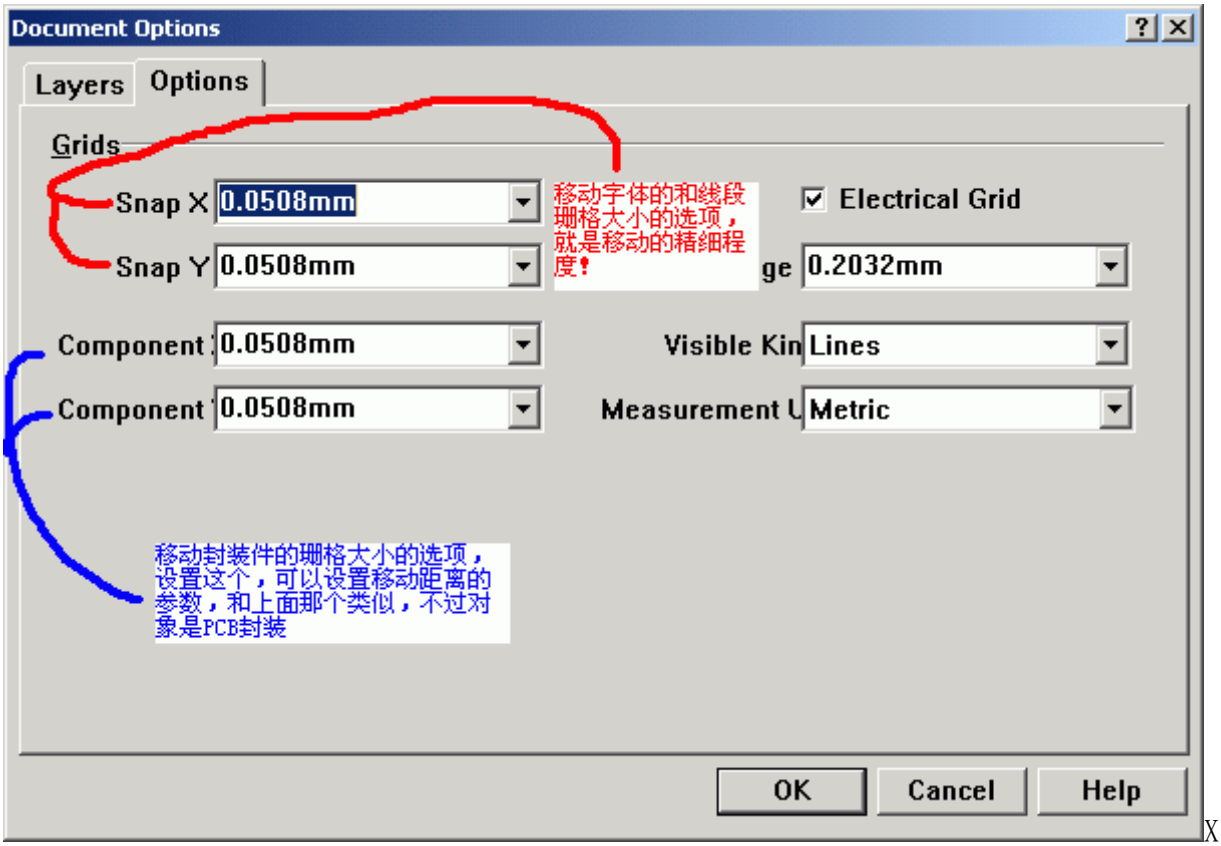


7. 这个是用来设置网格的标准，属于是个人喜好问题



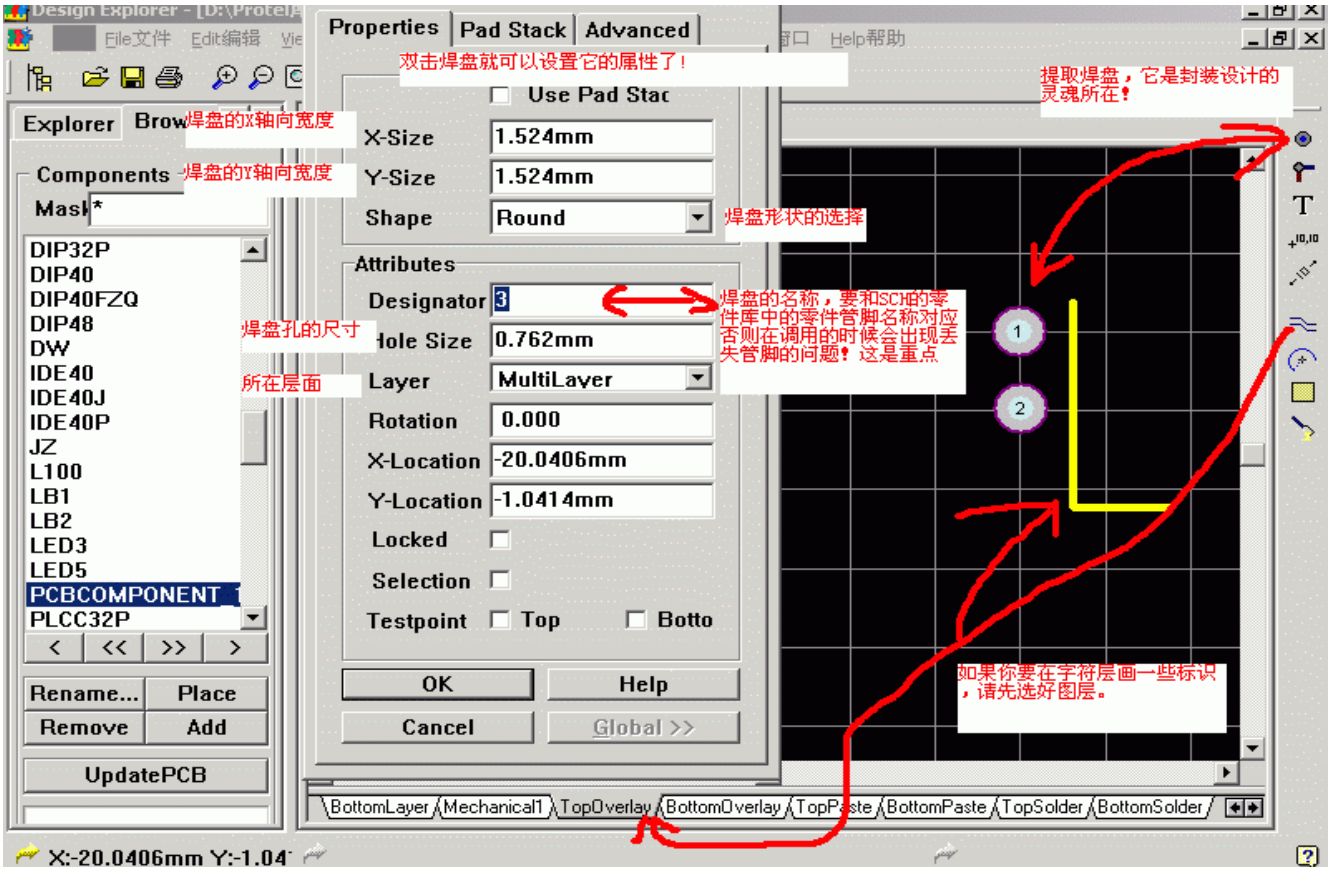
8. 这个就是执行上一步后的对话框


Document Options
Layers
Options
Signal layers
Internal planes
Mechanical layers
TopLay
BottomLa
Mechanic
Masks
Silkscreen
Other
把这个选项设置成2.54毫米，这个尺寸是双列直插IC管脚之间的标准间距！
Top Solder
Top Overlay
Keepout
Bottom Solder
Bottom Over
Multi lay
Top Paste
Drill guide
Bottom Paste
Drill drawing
System
DRC Errors
Pad Holes
Visible Gri
2.54mm
Connectio
Via Holes
Visible Gr
2.54mm
All On
All Off
Used On
OK
Cancel
Help

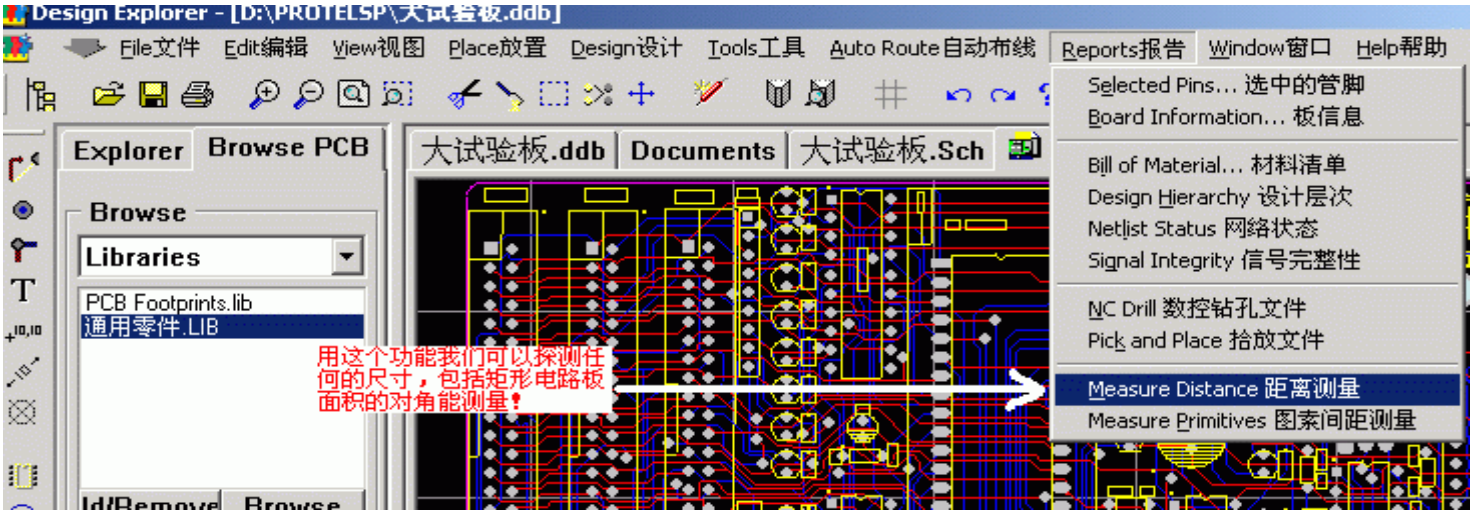

Document Options
Layers
Options
Grids
Snap X 0.0508mm
移动字体的和线段栅格大小的选项，就是移动的精细程度！
Electrical Grid
Snap Y 0.0508mm
ge 0.2032mm
Component 0.0508mm
Visible Kin Lines
Component 0.0508mm
Measurement U Metric
移动封装件的栅格大小的选项，设置这个，可以设置移动距离的参数，和上面那个类似，不过对象是PCB封装
OK
Cancel
Help

X

9.可以开始做封装了，注意哪些中文注释，核心问题就是焊盘的名称。



10.用这个功能可以知道我们做的封装的尺寸是不是精确的



恭喜恭喜！到这里， 你已经学会了 PROTEL 的基本操作.