

我爱萝莉的

单片机课堂

我爱萝莉爱萝莉·著

【我爱梦丽的单片机课堂】

第一课：准备入门4

项目一：《七色光芒》 彩色灯光控制器

第二课：欲学编程 先动烙铁.....8

第三课：人生第一个程序.....19

第四课：听话的单片机.....32

第五课：进军！神奇的程序世界.....37

第六课：高大上的 PWM 是怎么回事.....49

第七课：扩展课！用掉你的单片机.....59

项目二：《红外遥控》 智能家居新起点

第八课：新的玩具.....68

第九课：看不见的信使——红外线.....71

第十课：扩展课！ 好想遥控点什么83

项目三：《电量显示器》 学习 AD 转换

第十一课：自制电压表.....89

第十二课：最便宜的显示屏.....90

第十三课：扩展课！电压变成数字.....100

项目四：《舵机测试仪》 玩转航模电子

第十四课：制作舵机测试仪.....111

项目五：《电子开关》 控制夜航灯带

第十五课：制作电子开关.....123

第十六课：扩展课！ 万能的电子开关.....124

项目六：《单向有刷电调》 驱动高速直流电机

第十七课：电子开关的华丽转身——有刷电调.....133

第十八课：小弟出马——定时器.....135

第一课：准备入门

单片机是个奇妙的东西，几行代码，就能实现很多神奇的功能。第一课主要是做一些材料准备和环境搭建。

为什么要学习单片机？

什么改变了世界？是单片机改变了世界。环顾你的四周，哪一个电子设备里面没有单片机？一部汽车里面，甚至有上百块单片机。每个人都有很多脑洞大开的想法，创意是无限的，学会单片机，能让你亲自去改变世界。

网上已经有那么多单片机教程，为什么还要再开？

已有的很多教程在楼主看来，都存在着明显的不足，有的教程过于知识化、课本化，学习过程非常枯燥，自学者很难搞懂；有的教程虽然生动形象一些，但学习的东西还是传统的案例，什么流水灯呀抢答器呀万年历呀这些，根本没什么意思，学生也很难有兴趣去学。大一的学弟初学单片机，向我请教单片机有什么经验？我说，多动手多做一些好玩的小项目，做一两个以后自然就学会单片机了。然而翻开教程书的项目列表，哪个好玩？我沉默了，没一个好玩的，书上的那些项目我一个也没做过。这几年做 DIY 教程也让我发现，其实我挺适合做一个老师的。正好论坛里也有很多对单片机感兴趣却无从下手的朋友，那么就开一个与众不同的单片机学习课堂吧！

如何学会单片机？

关键字：兴趣。 回顾楼主个人的学习之旅，兴趣才是你学习单片机最根本的动力。曾有另一个学弟问我，学长我学多少算是学会单片机了？学到什么程度以后就可以不学了？学习只是为了完成“学会”的任务，学了两年他也没学会。

关键字：投入。投入是广义上的投入，包括时间、精力、金钱等很多的投入，没有投入就没有收获，亘古不变的真理。很多人只看到别人表面上的光鲜，却看不到别人背后的投入。想要跟着本课堂学习的人，先自问一下，能不能有这些投入？

如果愿意跟进，请回帖，以示决心！也让我能看到有多少人学习。

学单片机要不要买一个开发板？

很多学生买开发板时都是这样一个想法：开发板上什么都有，买一个用很久，也不用焊线，用着多方便。表面上这确实没错，然而实际上呢？很多人学了一两年，还是只有一个开发板，一个单片机，怎么用单片机做一个实际的项目？还是不会。

这里要说一下，学单片机，不只是学怎么写程序，更重要的是制作、调试电路的动手能力！

本课堂将主要从兴趣出发，边玩边学，强调实际的动手能力，制作出有实际意义的东西，在成就感中学会单片机！

1.1 先来制作一个下载器吧

无论是搞 DIY，还是学习单片机，一个下载器是必备的！

材料准备：

1、CH340G USB 转 TTL 模块一个



USB转TTL CH340模块 转串口

价格 ¥3.20

淘宝价 **¥2.70** 首件

配送 广东深圳 至 福建泉州

数量



2、杜邦线 4 根 买一排是 40 根，可以用很久啦



40P杜邦线 双头1P胶壳 长31CM 元

价格 **¥3.40**

配送 广东深圳 至 福建泉州丰泽区

数量 件(库存432)


立即购买

加入购物车

承诺 7天无理由

支付 模友之吧

1.2 学习单片机需要的软件

1、CH340G 模块驱动： [CH341SER.rar](#)

将 CH340G 插到 USB 口，解压并安装成功后，在设备管理器中会出现相应的 COM 口。不一定是 COM4，也可能是别的



2、STC 单片机下载软件： [stc-isp-15xx-v6.85.rar](#)

3、Keil4 C51 用于编写程序，文件 60M，到度盘下载：

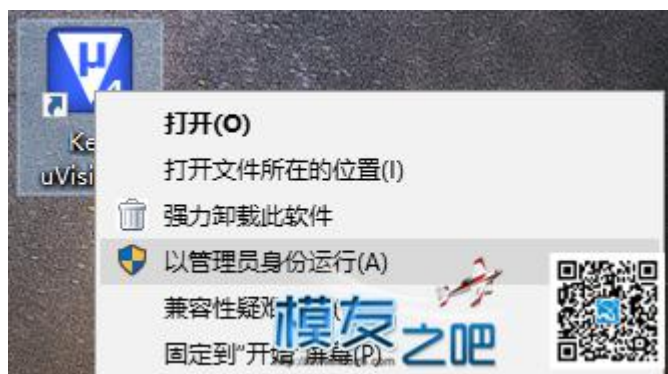
<https://pan.baidu.com/s/1dETNFrV>

注意：Keil 安装后即可使用，但如果写的程序大于 2K，就需要使用注册机破解：

网上的破解教程：

<http://jingyan.baidu.com/article/48b558e34c7ed77f38c09acc.html>

破解的时候必须管理员身份运行。如果没有成功，可以换一个注册码试试



（注：现在网络这么发达，有什么不会的都可以百度）

第一课大概就是这么多，如果有不足可以向楼主提意见

第二课：欲学编程 先动烙铁

单片机到底该怎么学？现在学校学生的普遍学习方法是买一个功能齐全的开发板，然后看着例程就慢慢搞了。这样是在学单片机吗？不，这只是学会了 C 语言编程。使用开发板好处虽然显而易见：能够快速上手，但坏处也是贻害颇深：动手能力为 0。

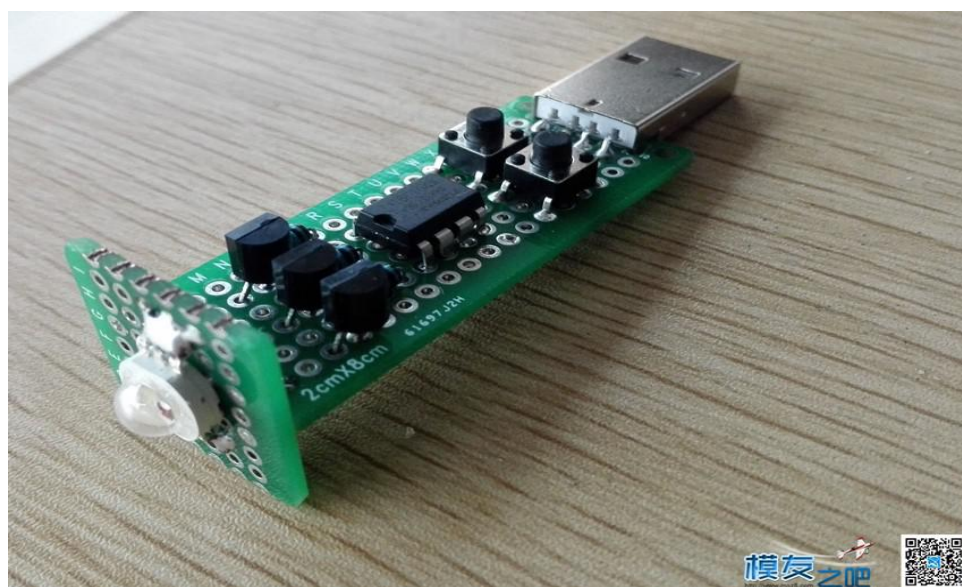
单片机解决的是实际问题，更多的是在和外部电路打交道，所以欲练神功，必先……必先挥动烙铁！

第一个项目搞什么？

前面楼主一直在强调，兴趣主导，和解决问题。那么咱们的课堂第一个项目搞什么？换做别的单片机教程，第一个程序肯定是流水灯……如果我们也这样搞，那前面吹那么多牛皮都白吹了。作为入学的第一个项目，既要简单，易学，又要有逼格，不丢份儿，楼主灵机一动：就做《七色光芒》！

《七色光芒》

《七色光芒》是楼主《光幻系列作品》之一。结构简单，好学易做，在平时，它只是一个普通的小手电，但在单片机的作用下，能够化成一团绚丽的色彩，营造氛围、送女友的神器！接下来楼主就带你边玩边学，自己添加单片机程序，让它变得更加炫目吧！



第二课的主要任务是采购元件、焊接电路、学习下载程序。

1.1 材料准备

1、STC15F104W 单片机一个。别看是个只有 8 只脚的小单片机，它能做的事情可不少呢！

（注：15F104W 只支持 5V，有条件最好换成 15W104 单片机，支持 3~5V，可用于 1S 电池。两个单片机程序兼容）



直插 STC15F104 单片机

价格 **¥1.70**

配送 广东深圳 至 福建泉州

数量

立即购买

承诺 7天无理由

2、8P IC 座 单片机的爱心小窝！



IC座 圆孔IC插座 6P 8P 宽体

价格 **¥0.05**

配送 广东深圳 至 福建泉州

颜色分类



3、RGB 灯珠。学过三原色都知道，红绿蓝三色可以组成所有的色彩。



4、SS8050 三极管。灯珠电流很大，而且需要恒流，用三极管是最佳的驱动方式。



5、驱动电流有多大？全靠电阻决定！推荐选择 10K 电阻，不够亮可以减小电阻。而且电阻全身都是宝哦，等下你就知道了。



6、单排针 40P



7、洞洞板一小块，由于需要垂直安装，最好买双面喷锡的板。

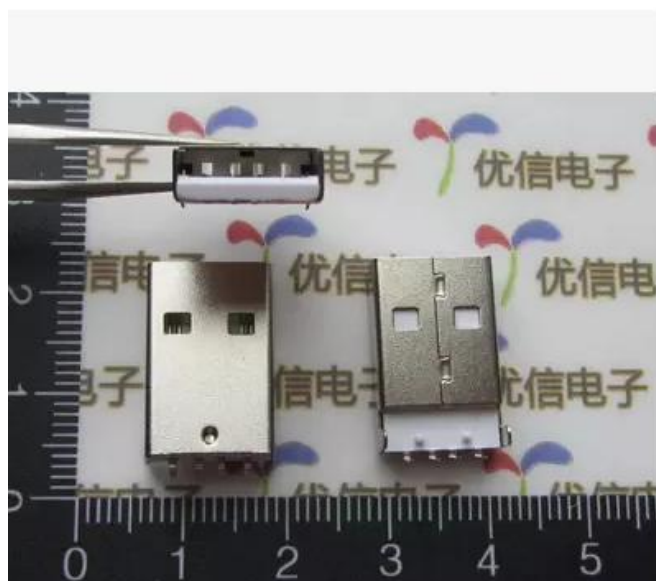


8、按键两个，用来控制单片机改变色彩、模式等



9、USB 公头。现在充电宝这么流行，USB 供电是最佳选择！自己配电池还要考虑充电和过放等。

注意买弯针的 USB 头，这样可以插入洞洞板。



白色USB A公头 USB弯针

价格 ¥0.12

淘宝价 **¥0.07** 首件优

配送 广东深圳 至 福建泉州

数量 件

立即购买



1.2 简易下载器制作

在第一课中，我们已经购买了一个下载模块，但为了让它更加好用，还要做一些必要处理：

1、将模块 VCC 和 5V 焊接在一起。为什么要这样做？这个模块 3V 不稳定，很容易下载失败，只能用 5V。



2、找四根线杜邦线，按图线序编排。四根线的顺序：GND、5V、RX、TX。以后所有的电路下载口默认都会是这个线序！

（为什么是这个线序：这个线序是防反插的。很多人设计的下载口都是 5V、GND 在两边，殊不知一旦插反，可能会烧电路！）

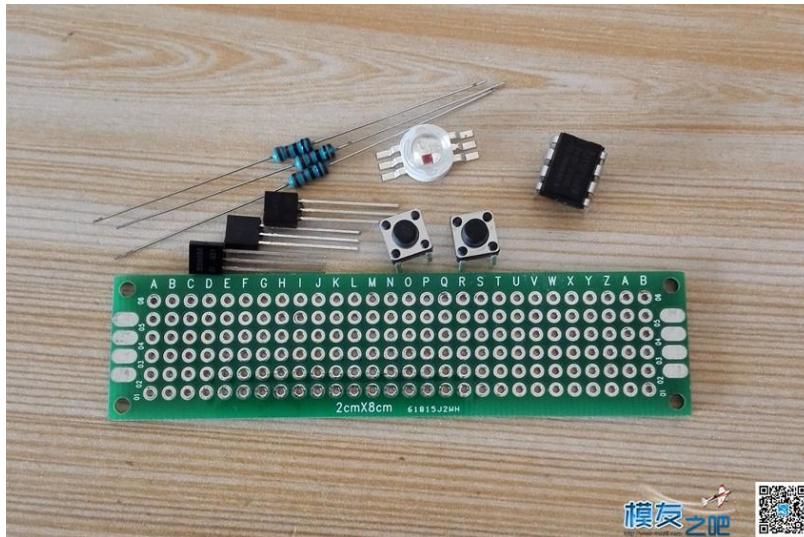


1.3 亲手焊接一个小电路吧

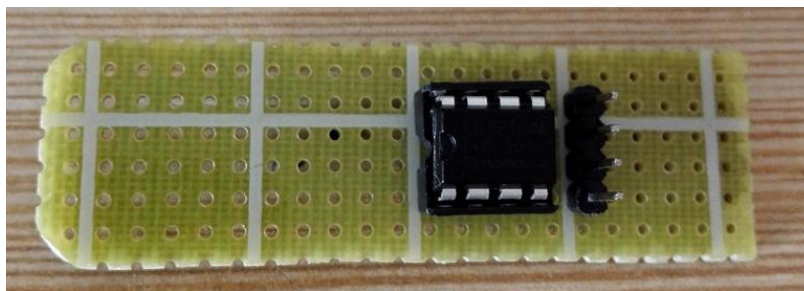
1、洞洞板裁剪技巧：用小刀在板子上反复刻画一条划痕，反面相同位置也刻画一道，用力用力一掰就开啦



2、裁出合适大小的板子，准备上元件全家福：



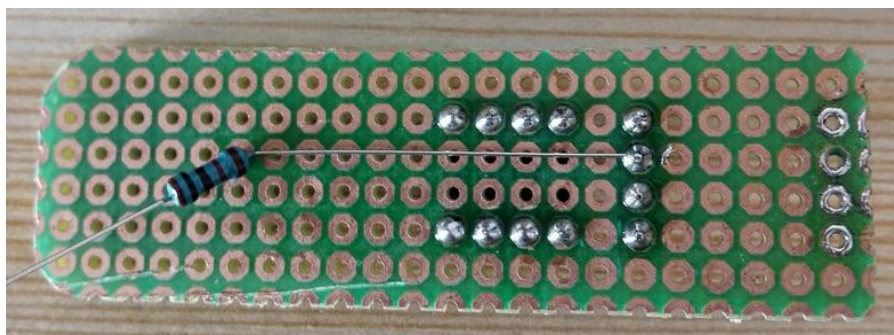
3、焊上单片机座及四针下载口

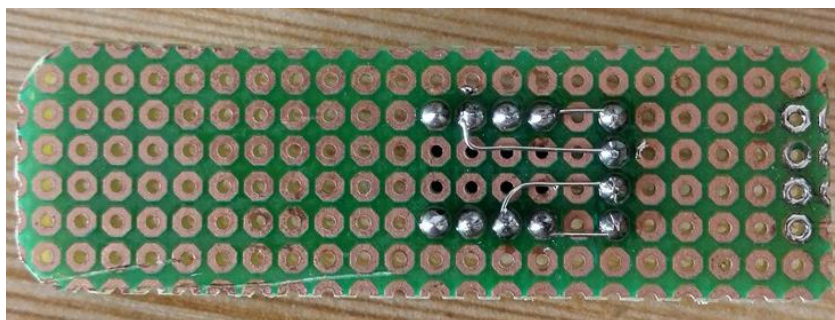


4、洞洞板布线祖传技巧：电阻腿大法！

在之前出的 DIY 教程中很多人问我电路板上整齐的布线、发亮的“银丝”是哪里的来的，现在我就不在吝啬告诉你啦：那是电阻腿\(^o^)/~

快使用电阻腿大法！不用剥线头，不用镀锡，想用多长就接多长，一个电阻两个腿，两个电阻四条腿，三个电阻.....一块钱 100 个电阻，多实惠！O(∩_∩)O~~





1.4 学习烧录下载程序（固件）

1、下载程序其实很简单，把下载器插电脑 2、打开软件按下图设置一下，点击下载，

3、插上四针下载口，自动开始下载。

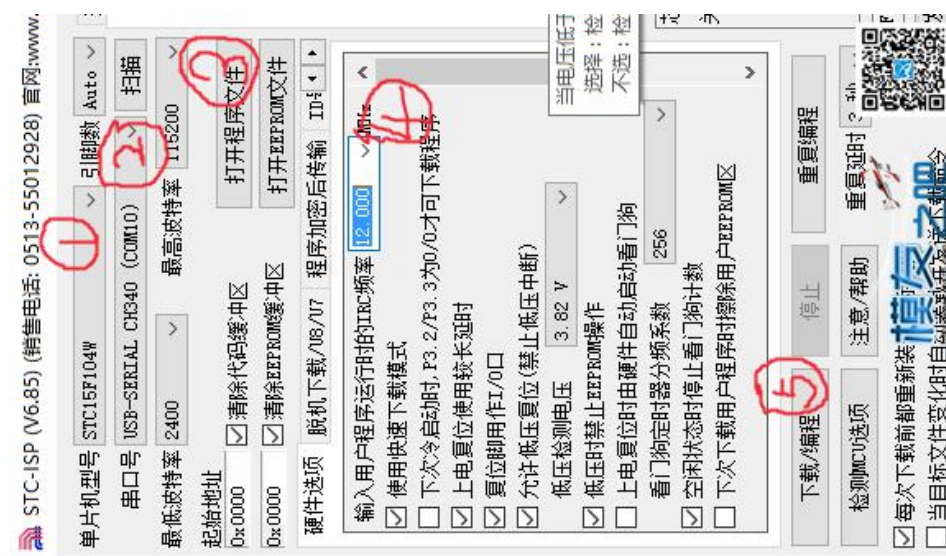
下载程序是最基础的知识，必须学会，不然是没法向下进行的。

测试固件，焊完以后先自己烧录一下试试。



项目 01【七色光芒】呼吸灯效果测试固件.rar

下载设置图：



点击下载然后再插下载口



下载这么简单，我觉得认真一点，几秒就能学会。

下面是给那些千方百计还下载不进去的人看的！

下载程序对会的人来说像喝凉水一样简单，对不会的人来说简直像魔法，其实这个真没什么难的，注意一下注意事项就可以了。

- 1、前提条件：买下载器、安装驱动、制作下载线
- 2、①打开 STC-ISP 软件，选择单片机型号；如果出现超出文件大小是单片机型号选错了
 - ②选择 COM 号；如果下载失败时发现 COM 号闪烁，是供电不足下载器掉了线了
 - ③打开程序文件，加载 HEX 固件；
 - ④正确勾选每个选项；多数异常现象都是由于没有正确勾选导致的
 - ⑤点击下载，插上四针下载线，等待下载完成。必须是点了下载后再给单片机供电

3、如果一直在检测，试着交叉一下 RXD 和 TXD，看是不是你接反了。或者重新插拔一下 USB。

USB 注意插到电脑原生的 USB 口 ,大部分失败都是电脑 USB 供电不足导致 ,
用 USB3.0 1A 的供电能力 ,就算你电路上有 2000uf 的电容也能成功下载。

如果始终在 “开始调节频率” ,看你是否接触不良供电不稳。

自己电脑不行的可以用别人电脑试试。

1.6 继续完成焊接

1、插上三极管和电阻

2、焊接 LED 灯珠 ,看灯珠里面的 “品” 字形 ,注意正反 ,反了是不会亮的。

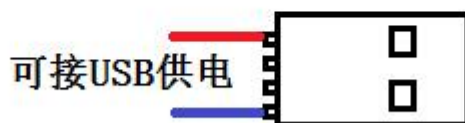
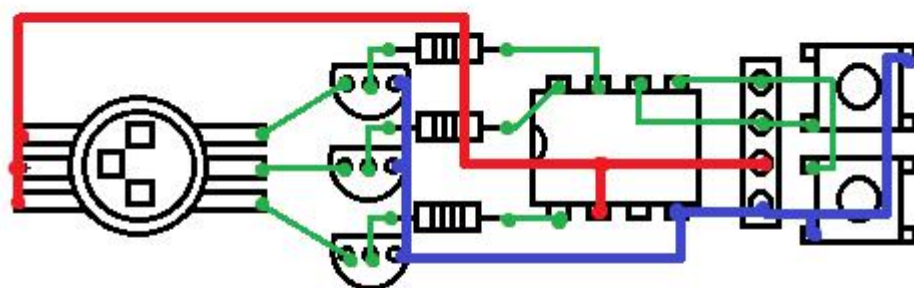
如果你已经下载好测试程序 ,现在插上电就能亮呢。

(如果是新单片机 ,出厂内置有流水灯程序 ,也会闪烁)

3、焊上两个按键。有这些就可以调试程序了 ,USB 供电头可以等以后再焊。

4、实物电路图。注意：为了统一规范，以后实物电路图全都按正面布局、绘图，避免引起误会。

仍习惯于背面焊接图的，只需要自己镜像一下即可！



第三课：人生第一个程序

人都说：师傅领进门，修行靠个人，其实很多人缺的不是修行的能力，而是领进门的师傅！我相信，一定有很多业余爱好者喜欢单片机，想要学习却无从下手，网上的资料虽多，却没有一个顾及业余爱好者、0基础受众的课程！所以我在教学时会都会从最简单的开始讲起，如果有没考虑到的，欢迎补充提问。而且课程进度也不会很快，每天抽出几十分钟时间练习一下即可。

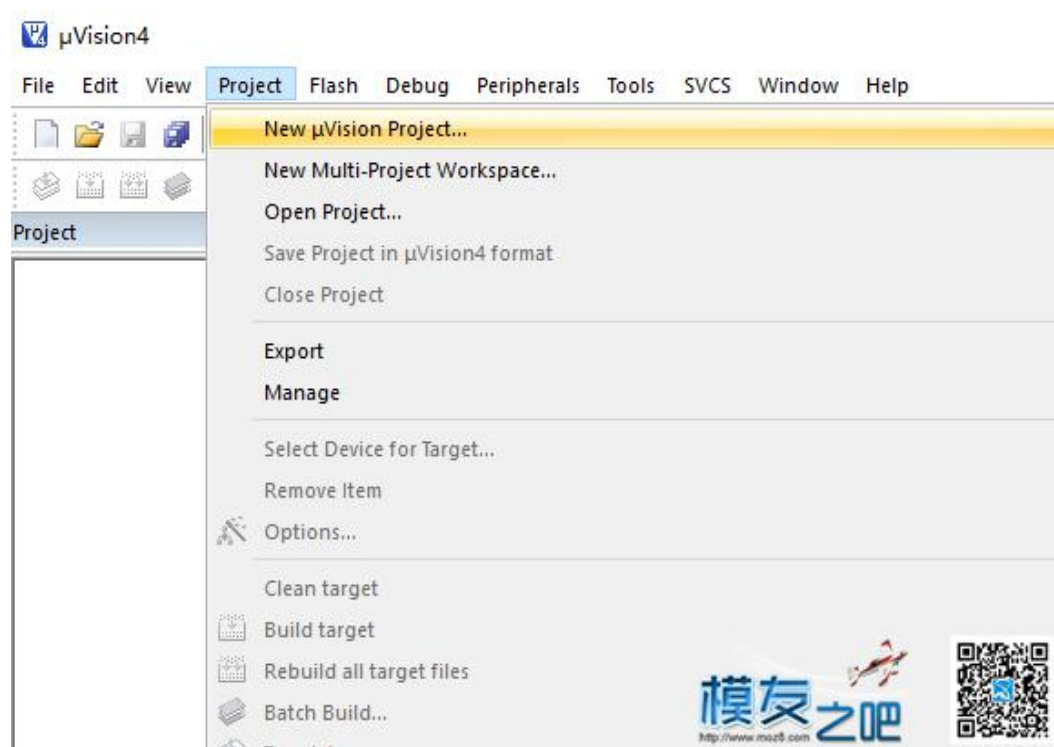
第三课的主要任务是学习建立 Keil C51 工程，和编写最简单的程序。

1.1 新建单片机项目工程

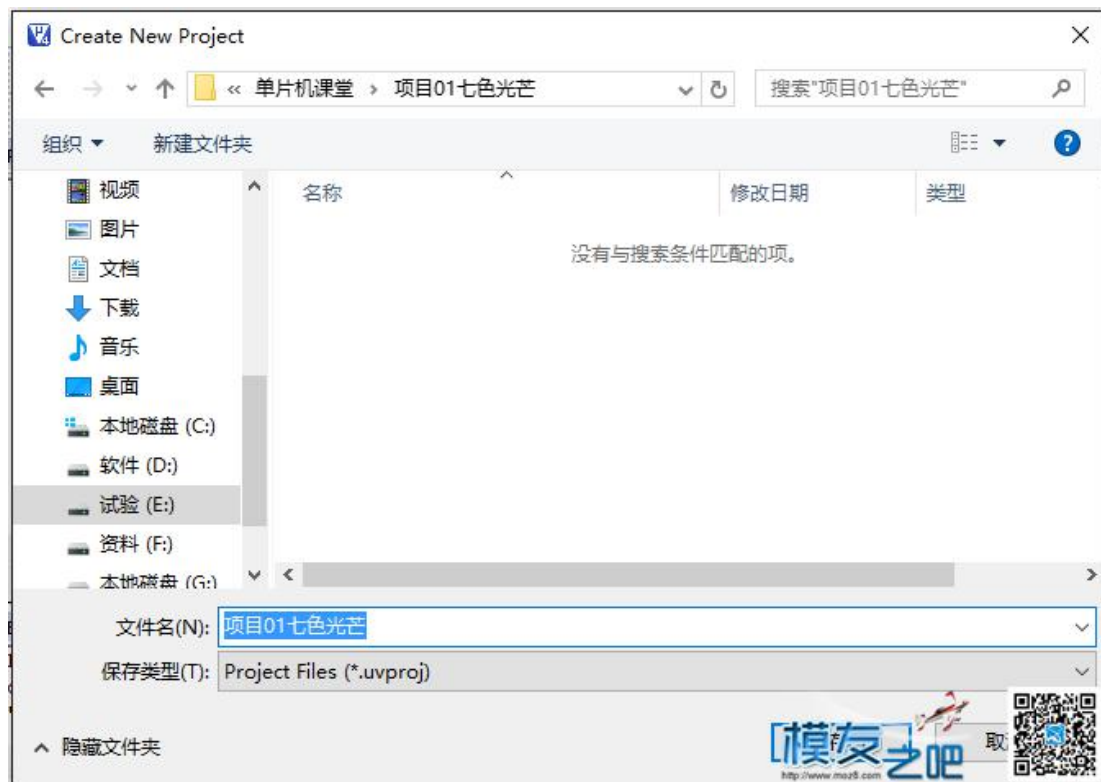
如果你已经在第一课下载并安装了编程软件，那么就可以愉快的按照下面的步骤学习了：

（如果有英语不好的童鞋，可以百度把那几个单词记下来，总共也没几个单词）

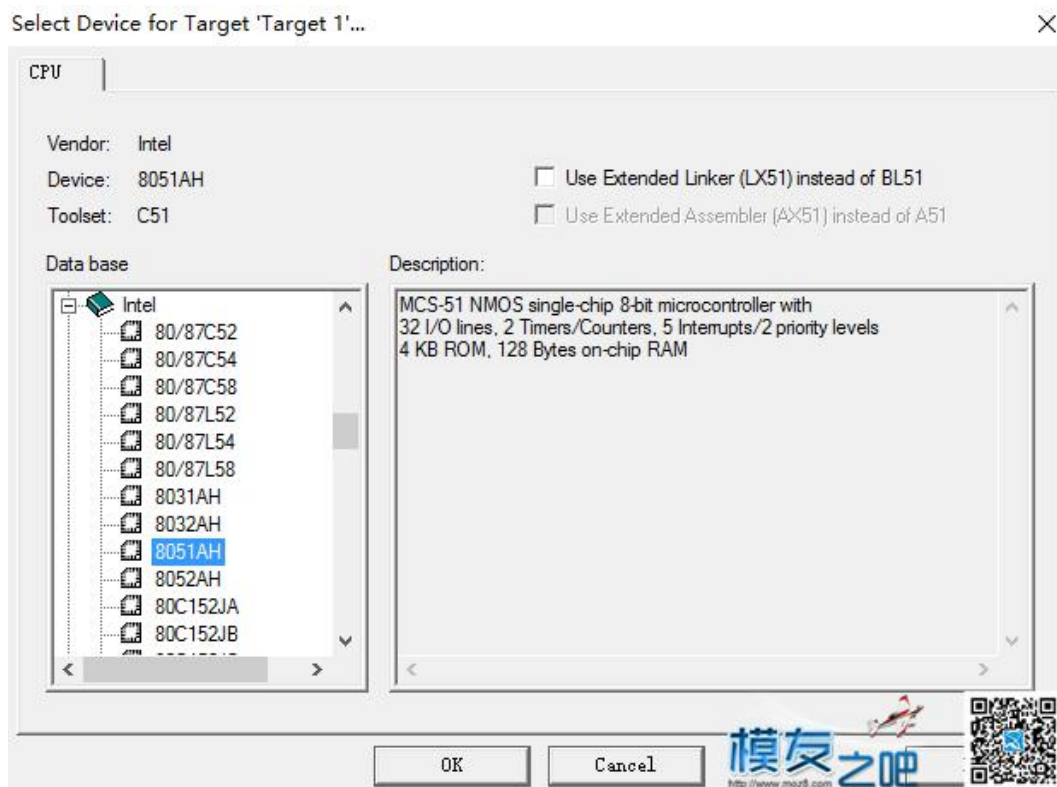
1、新建工程



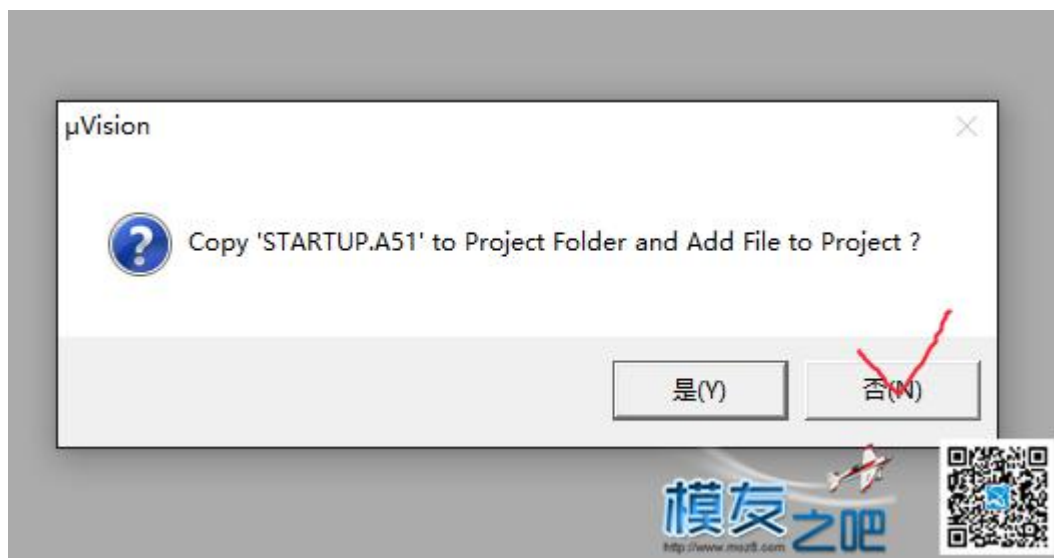
2、新建规整的文件夹方便管理，保存工程



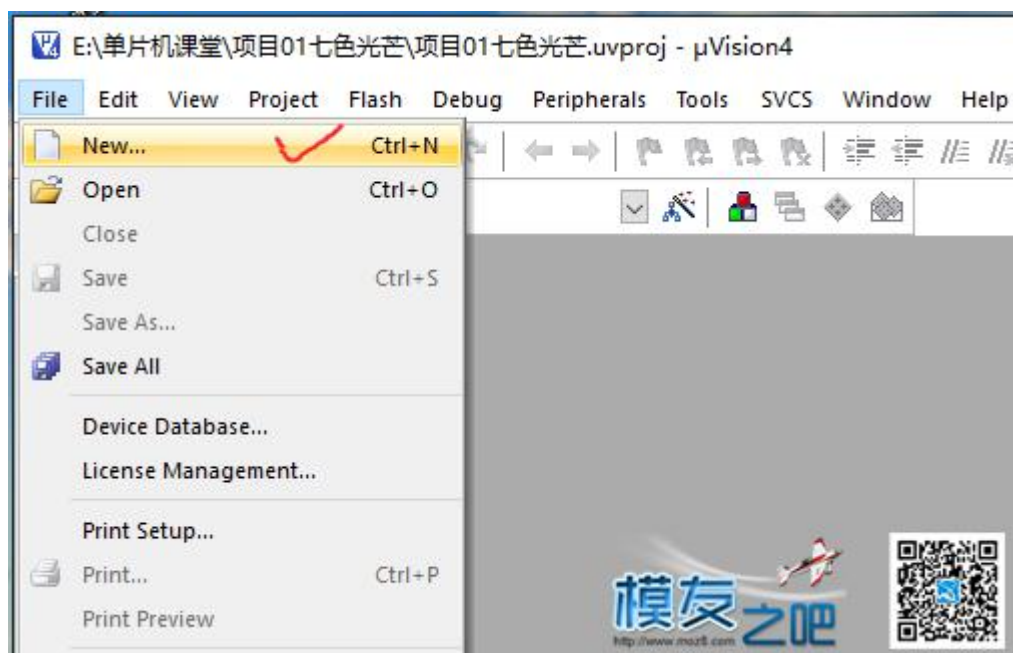
3、弹出芯片型号选择窗口，STC 单片机是开源的 Intel 8051 构架



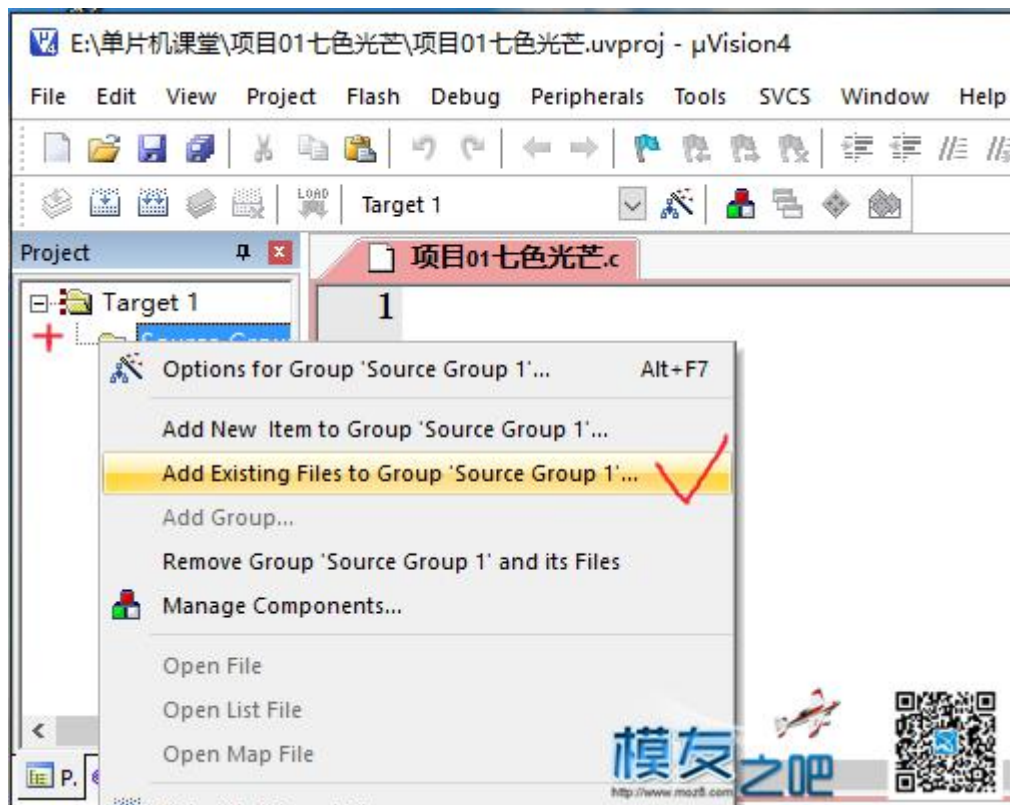
4、弹出是否自动添加头文件，选择否



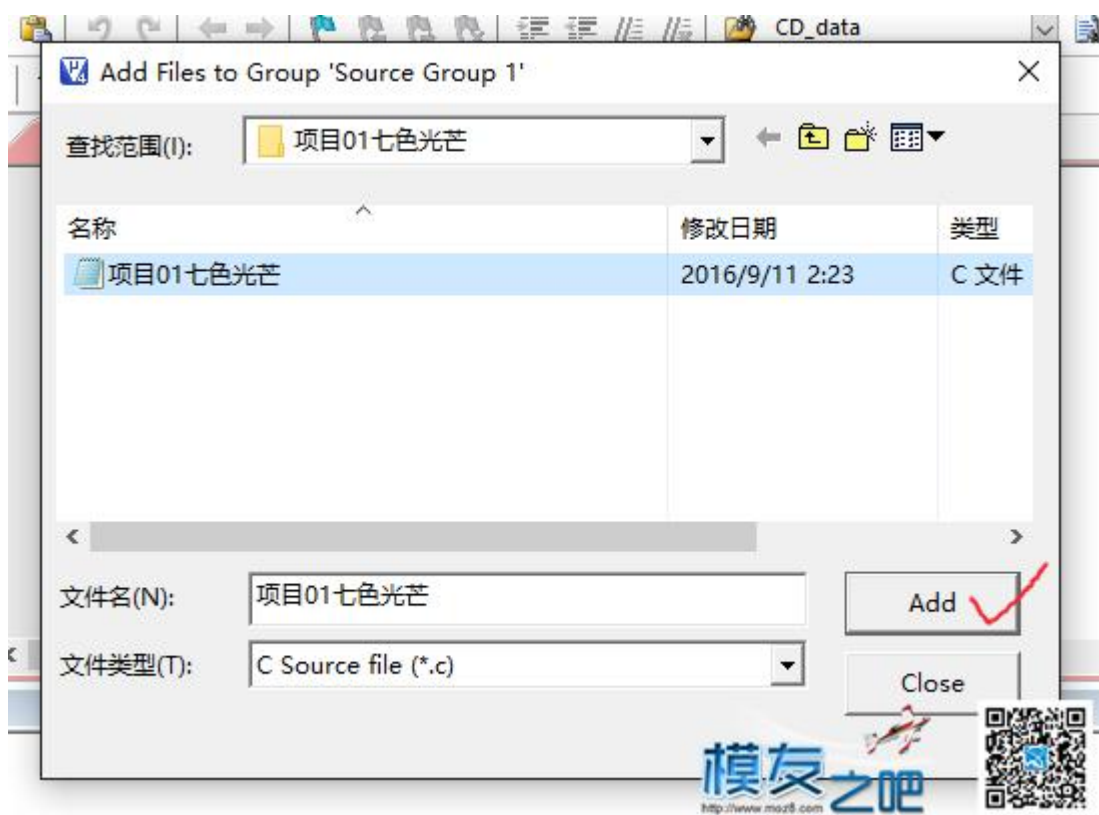
5、新建一个文件



6、保存



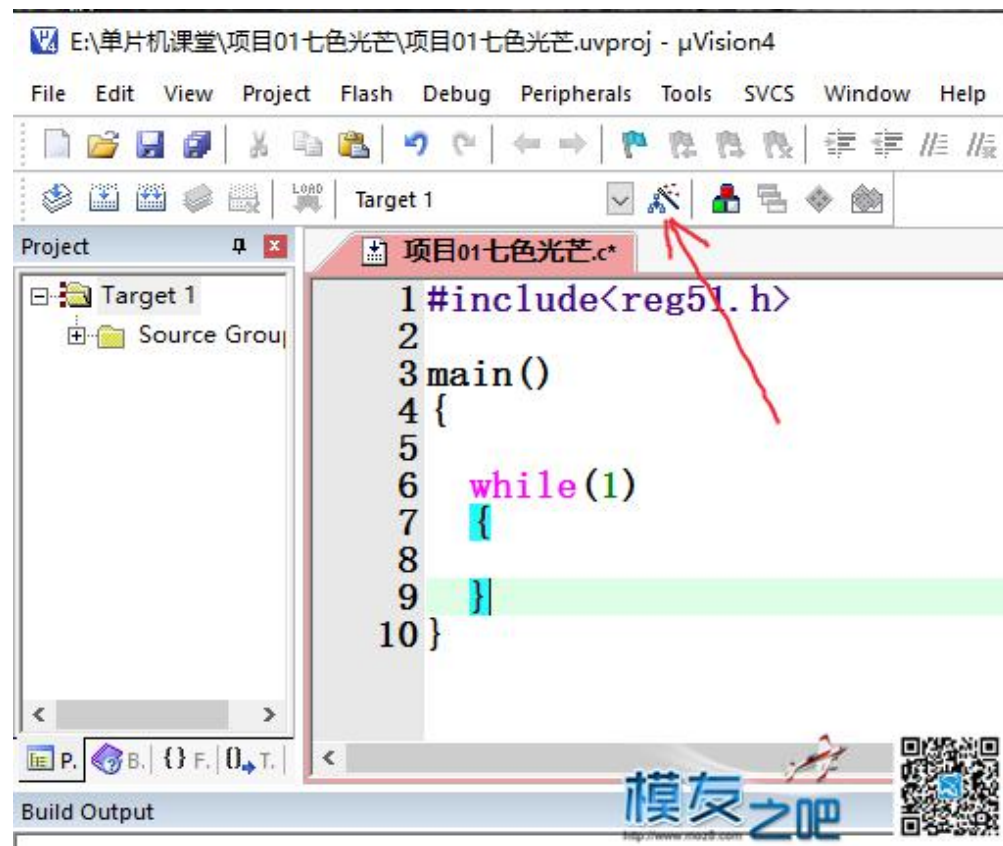
9、把.c 文件添加到工程



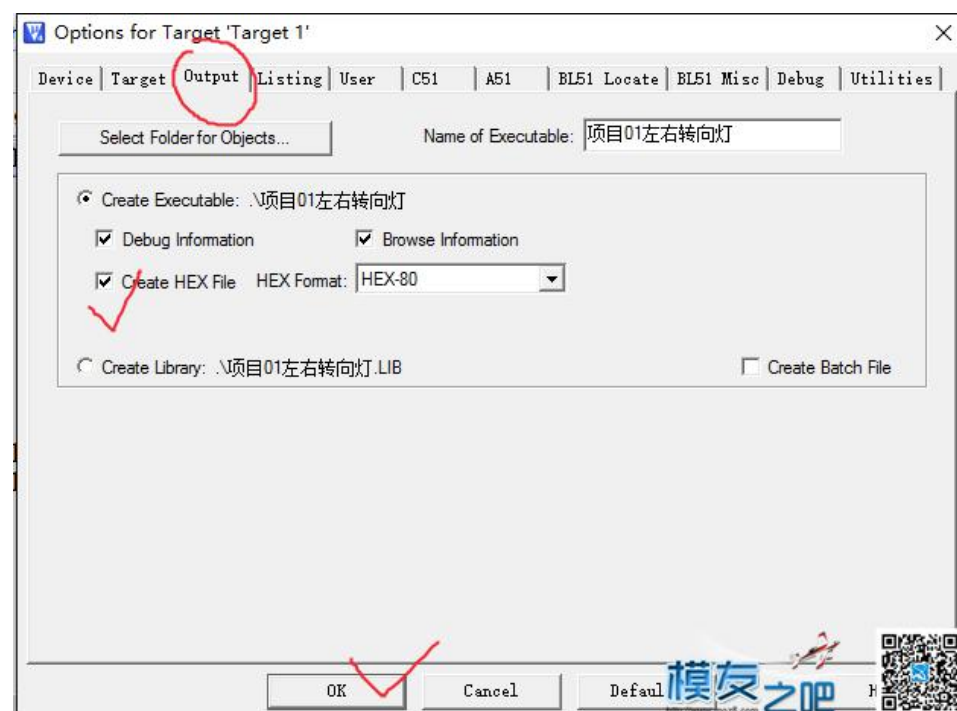
10、切换输入法到英文，亲自手打敲出你的第一个代码吧。这算是最简单的一

个代码了，有不能理解的随后会讲。敲完后点击锤子按钮进行设置。

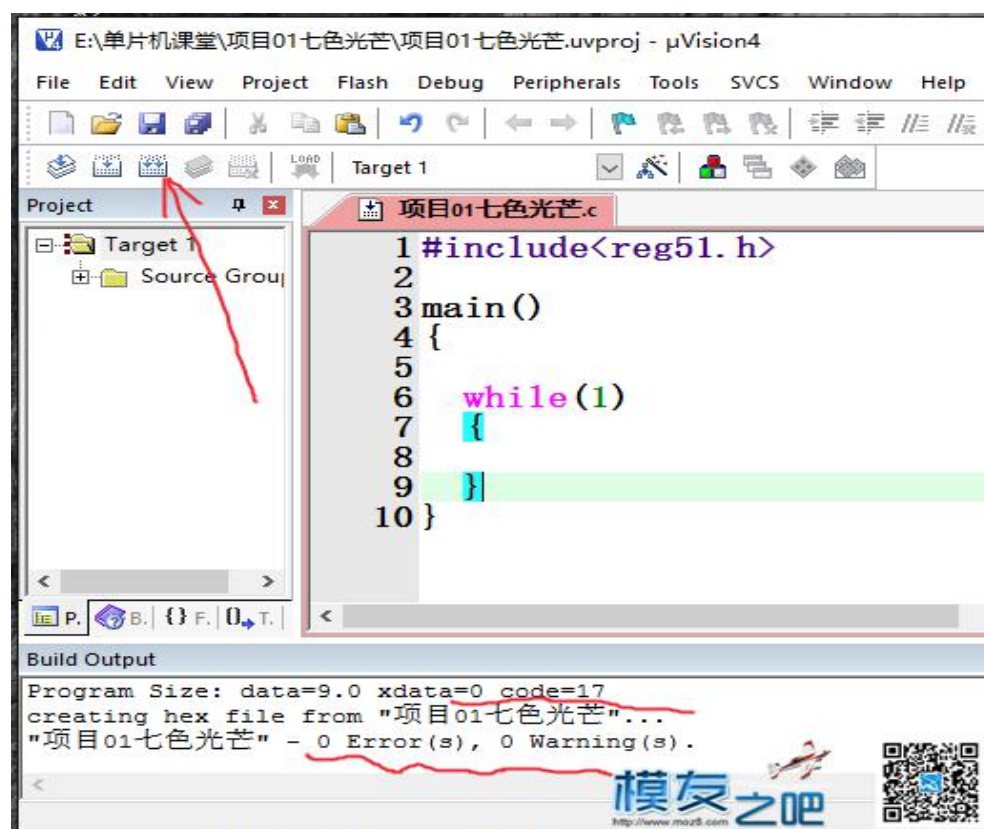
（中文输入法会敲出不能识别的乱码，切记！）



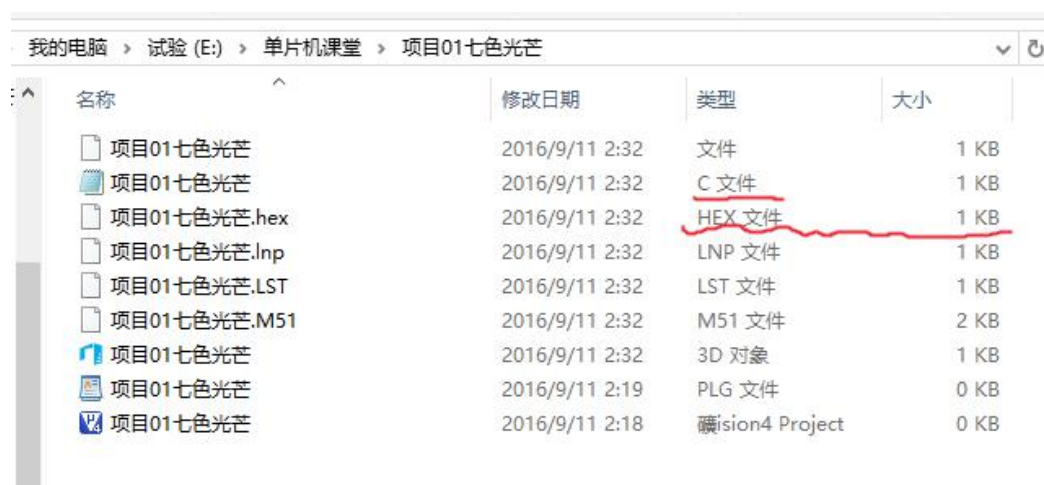
output 输出选项，勾选创建 HEX 文件



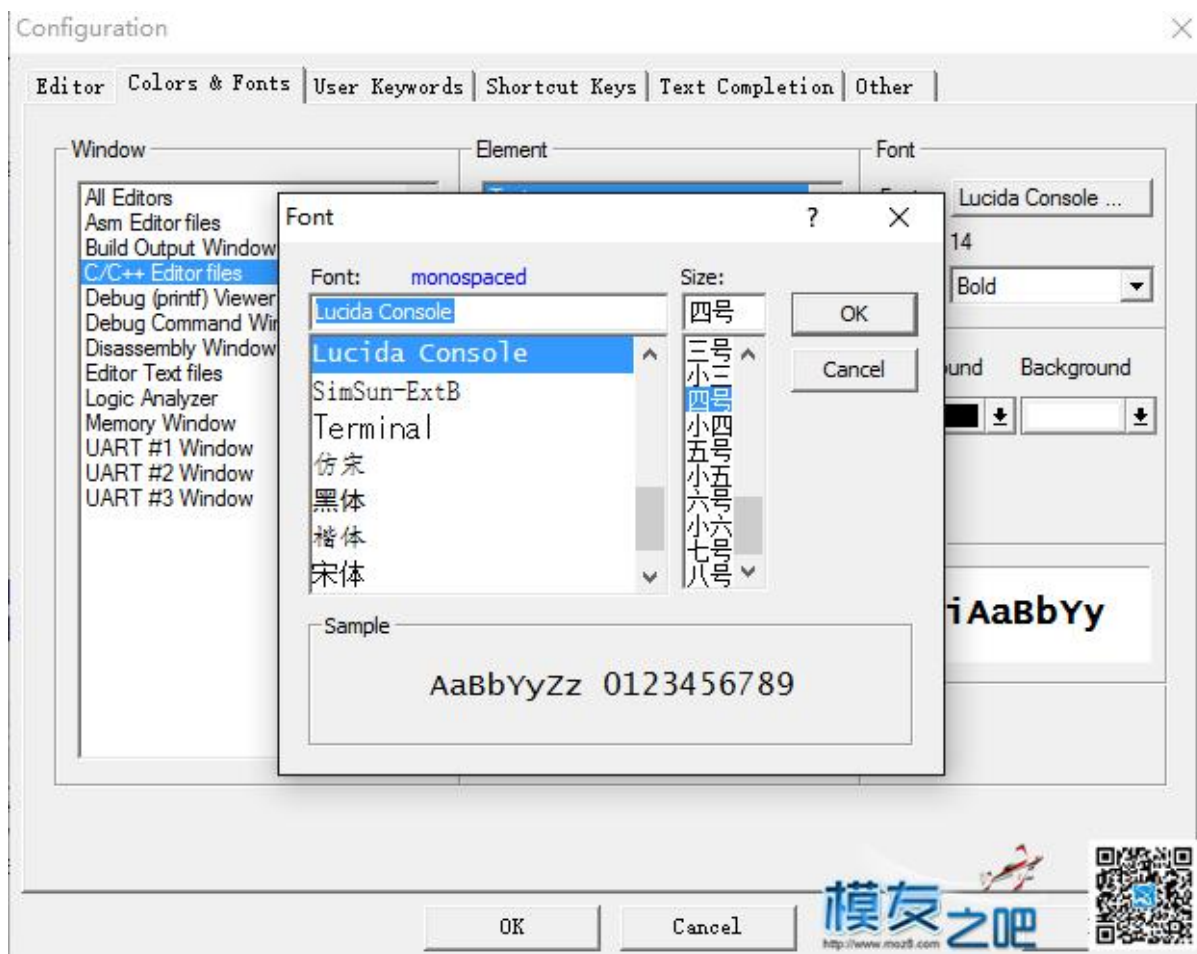
12、点击编译按钮，叮咚一声清脆的声音，0 错误，0 警告，撒花~一个完美的新工程诞生啦



13、打开工程文件夹，可以看到你新建工程的所有文件。其中 HEX 就是可以下载到单片机里面的东西哦。



14、如果对程序的字体不满意。可以百度“Keil 字体设置”，设置自己喜欢的字体和颜色。楼主自己喜欢稍大圆滑的字体，配置如下：



1.2 必备小知识

我们先来学习一下这个最简单的单片机代码

```

01.  #include<reg51.h>
02.
03.  main()
04.  {
05.
06.      while(1)
07.      {
08.
09.      }
10.
11. }
    
```

复制代码

知识点 (1) #include<reg51.h>

#include<> 是包涵、调用的意思。reg51.h 是 51 单片机的寄存器声明文件。

合起来的意思就是调用 51 单片机的头文件。我们初学单片机，对这个不用深究，只要知道在程序第一行这样写就行了。

知识点（2）main 函数

包括 main（）和 main 下的大括号。就是主函数的意思。

当单片机一通电，程序就会从主函数里面的第一行开始，一行一行的执行，走完了以后重新开始

知识点（3）while（1）循环

上面说了，如果程序走完了，单片机就会复位重新走。我们如果不希望程序被走完呢？

那就用 while（1），把程序卡住。

while（1）就代表一个无限循环，它的括号里面的程序会被反复执行，程序永远不会走完。

那单片机一直这样走，会不会把它累坏呢？哈哈，其实楼主在初学时也一直担心这个问题，现在看来担心是多余的。

1.3 延时函数

我们要想让单片机正确的做事，必须要知道什么时间去做。比如现在客户有一个要求，按下按钮以后，灯泡亮一秒自动熄灭。

怎样知道时间过去了一秒呢？最简单的方法：软件延时。

STC 单片机有一个好处，就是下载软件自带程序生成器，我们可以用这个功能方便的创建不同时间的延时函数。

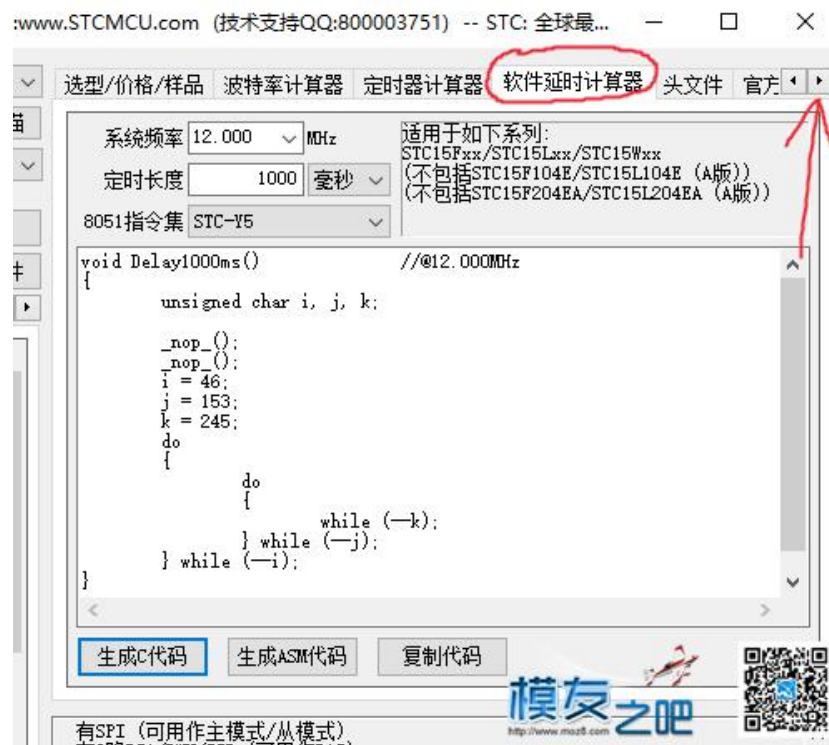
打开 STC-ISP 软件，找到软件延时计算器；

系统频率选择 12Mhz，我们以后写程序都默认这个频率；

定时时间选择 1000 毫秒 (ms)，就是 1 秒 (s)。1 毫秒 (ms) 等于 1000 微妙 (us)；

指令集选择 Y5，代表 STC15 系列单片机；

点击生成代码，然后复制到你自己的程序里就可以了。



```

01.  #include<reg51.h>
02.
03.
04.  void Delay1000ms()                //@12.000MHz
05.  {
06.      unsigned char i, j, k;
07.
08.      //_nop_();
09.      //_nop_(); 这两个用不到，直接删掉
10.      i = 46;
11.      j = 153;
12.      k = 245;
13.      do
14.      {
15.          do
16.          {
17.              while (--k);
18.          } while (--j);
19.      } while (--i);
20.  }
21.
22.  //此为分界线，函数的具体内容写在上面；
23.  //下面是具体的程序，写函数名字就可以调用了。
24.
25.  main()
26.  {
27.
28.      while(1)
29.      {
30.          Delay1000ms(); //每一行程序结尾要加分号
31.      }
32.
33.  }
复制代码

```

知识点（4） 延时函数原理

可以看到自动生成的延时函数里有很多密密麻麻的东西，不要感到恐惧，其实它就是靠反复执行那些无用的东西浪费时间，才有了延时效果。

现在先不用管是怎么写的，现在只需会用就行了。

知识点（5） 声明函数

在函数调用之前，要先声明，所以要放在主函数之前。

知识点（6） 调用函数

只有在程序中调用了函数，它才会实际生效。

知识点（7） 程序注释

这段函数什么意思？如果怕自己忘了或别人看不懂，可以在函数中添加注释，注释可以用汉字。

格式：

```
//注释内容
```

具体原理是：“//” 符号后面的东西 Keil 软件就当不存在，只有人类才能看见哦。

现在的程序就是在反复执行延时 1 秒的函数，赶快编译试一下吧！

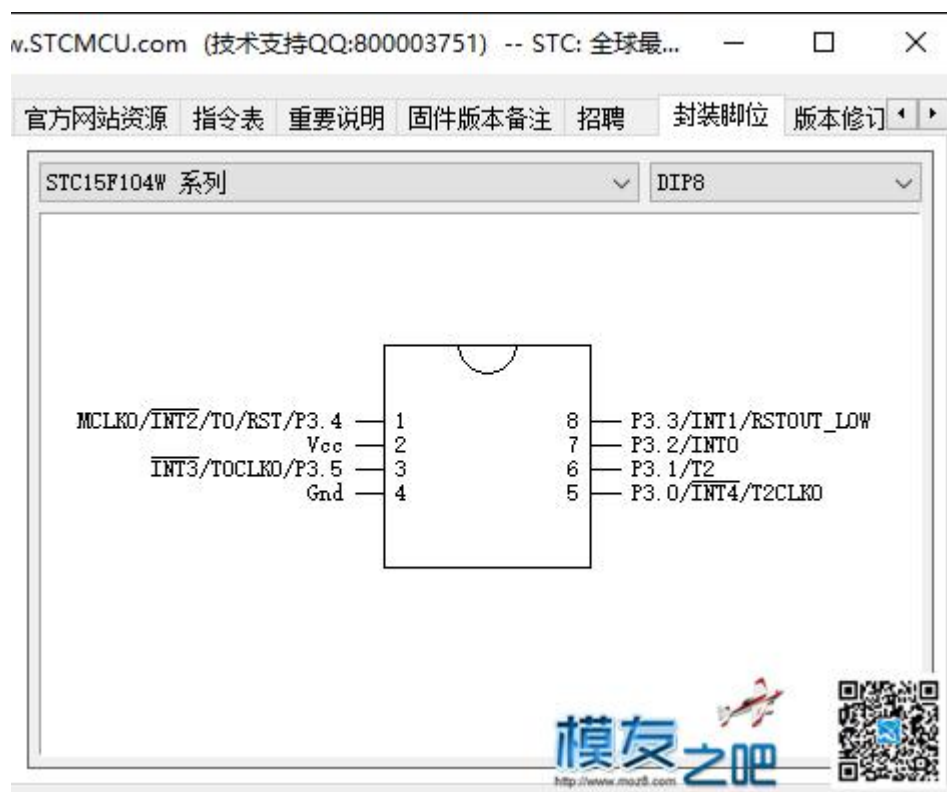
1.4 单片机的引脚

当当当当~~~请出我们本次课程的主角：STC 最简单的 8 脚单片机！



这个单片机的各个引脚是什么功能呢？

打开 STC-ISP 软件，找到封装脚位选项，可以看到具体描述：



知识点（8）VCC、GND

GND 是地，代表电源的负极，VCC 就是电源正极了。根据不同的芯片型号，可以接 3~5V。

知识点（9）P3.0、P3.1、.....

P 代表端口的意思，单片机有多个端口，如 P0、P1、P2、P3 等。

每个端口有 8 只引脚，对于端口 3，有 P3.0、P3.1、P3.2、P3.3、P3.4、P3.5、P3.6、P3.7。

这个单片机只有可怜的几个引脚，所以就只能用其中一部分啦。

这些引脚还有一些别的名称，那是额外的功能，暂时无需知道。

在第一课中，我们已经制作出了试验电路，三色灯珠接在单片机的引脚上，如何控制它们？下节课自见分晓！

第四课：听话的单片机

把单片机想象成一个八只爪的小傀儡，每次下载程序就是给它赋予新的灵魂，听话的执行你的命令，这样是不是感觉有意思多了？

再高傲的单片机也要听从程序猿的命令，从现在开始驯服你的小傀儡吧！

第四课的主要内容是学习如何控制单片机的引脚输出。

（注意：准备学习单片机却毫无基础的童鞋，请拿出一个小本本，把每个知识点全部记下来。以后知识点会有很多，只看一遍就关掉网页，没有认真理解到后面是学不会的！）

1.1 引脚定义和输出

我们要想让单片机听话的为我们做事，首先要明白单片机能做什么。很多人可能会想当然：让单片机输出不同的电压，来控制电机转的快慢。事实上单片机能直接做的事只有两件：输出高电平和低电平。

知识点（10） 高电平 低电平

在单片机术语中，只有真和假、1 和 0 的概念，对应于引脚的输出，也只有高电平和低电平两种。

如果我们在程序中是 1，单片机实际的输出就是高电平，接近单片机的电源电压 5V；

如果我们在程序中是 0，单片机实际的输出就是低电平，大概 0V；

那可不可以用单片机输出的 5V 来直接驱动电机、灯泡呢？

知识点（11） 引脚驱动能力

单片机输出低电平时，能流入的电流大概 20ma。

单片机在输出高电平时，输出电流就很小了，不到 1ma。（也有别的模式，这个以后会讲）

（单片机在通电后，引脚的默认输出都是高电平）

也就是说，单片机直接驱动的话，20ma 点亮个发光二极管还行，想驱动上百 ma 电流的大功率灯珠，就要加放大器了。

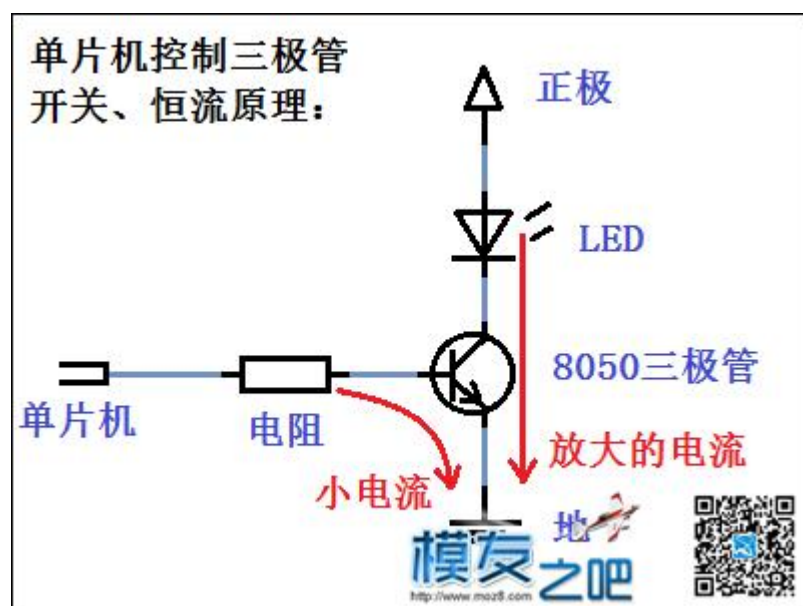
知识点（12） 单片机控制三极管

如果你高中的物理知识还没有全部还给体育老师，那么你应该知道，三极管可以放大电流。

看下面这个电路图，如果单片机输出低电平 0V，没有电流流过，自然也不能放大，LED 不发光，我们可以定义一下：低电平关闭三极管。

如果单片机输出高电平 5V，经过电阻以后，会有一个很小的电流流过，三极管放大电流，LED 发光，同样可以定义：高电平打开三极管。

简单的估算：假设经过电阻的电流是 1ma，三极管放大 150 倍，大概就是 150ma 电流。也就是说通过调节电阻的大小，可以控制灯珠的最大亮度。



知识点 (13) 引脚定义 `sbit LED1=P3^2;`

引脚定义格式：`sbit 名称=引脚标号`；

想要控制一个引脚，必然要先定义一个引脚，这句话的意思就是将单片机 P3.2 引脚取了一个别名 LED1，以后你只要在程序中写上 LED1，代表的就是这个引脚。

知识点 (14) 引脚控制 `LED1=0;`

如此只要在程序中写上 LED1 等于 0、等于 1，就能随心控制引脚输出的电压啦，快来试试吧。

```

01.  #include<reg51.h>
02.
03.  sbit LED1=P3^2;//先定义，才能使用
04.
05.
06.  void Delay1000ms()                //@12.000MHz
07.  {
08.      unsigned char i, j, k;
09.
10.      //_nop_();
11.      //_nop_(); 这两个用不到，直接删掉
12.      i = 46;
13.      j = 153;
14.      k = 245;
15.      do
16.      {
17.          do
18.          {
19.              while (--k);
20.          } while (--j);
21.      } while (--i);
22.  }
23.
24.  //此为分界线，函数的具体内容写在上面；

```

```

25. //下面是具体的程序，写函数名字就可以调用了。
26.
27. main()
28. {
29.
30.     while(1)
31.     {
32.         LED1=0; //低电平灭
33.         Delay1000ms(); //每一行程序结尾要加分号
34.         LED1=1; //高电平亮
35.         Delay1000ms(); //这个程序作用就是1秒亮1秒灭
36.     }
37.
38. }

```

[复制代码](#)

1.2 闪出七种色彩！

在学会了怎么控制 LED 的亮灭后，你有什么想法呢？是不是已经按捺不住想要写点更复杂的程序了？

我们先把每个 LED 的控制引脚都定义好：

```
sbit LED_R=P3^4; //定义红灯
```

```
sbit LED_G=P3^3; //定义绿灯
```

```
sbit LED_B=P3^2; //定义蓝灯
```

然后按照三原色的混合原理，控制每一种色彩吧：

```

01. #include<reg51.h>
02.
03.
04. sbit LED_R=P3^4; //定义红灯
05. sbit LED_G=P3^3; //定义绿灯
06. sbit LED_B=P3^2; //定义蓝灯
07.
08.
09. void Delay1000ms() // @12.000MHz
10. {
11.     unsigned char i, j, k;

```

```
12.
13.      //_nop_();删去此处
14.      //_nop_();
15.      i = 46;
16.      j = 153;
17.      k = 245;
18.      do
19.      {
20.          do
21.          {
22.              while (--k);
23.          } while (--j);
24.      } while (--i);
25.  }
26.
27.
28.  main()
29.  {
30.
31.      while(1)
32.      {
33.          LED_R=1;
34.          LED_G=0;
35.          LED_B=0;
36.          Delay1000ms(); //红色亮1秒
37.          //空一行为了让程序更清晰
38.          LED_R=1;
39.          LED_G=1;
40.          LED_B=0;
41.          Delay1000ms(); //黄色亮1秒
42.
43.          LED_R=0;
44.          LED_G=1;
45.          LED_B=0;
46.          Delay1000ms(); //绿色亮1秒
```

```
47.  
48.          LED_R=0;  
49.          LED_G=1;  
50.          LED_B=1;  
51.          Delay1000ms(); //青色亮1秒  
52.  
53.          LED_R=0;  
54.          LED_G=0;  
55.          LED_B=1;  
56.          Delay1000ms(); //蓝色亮1秒  
57.  
58.          LED_R=1;  
59.          LED_G=0;  
60.          LED_B=1;  
61.          Delay1000ms(); //紫色亮1秒  
62.  
63.          LED_R=1;  
64.          LED_G=1;  
65.          LED_B=1;  
66.          Delay1000ms(); //白色亮1秒  
67.      }  
68.  
69.  }  
复制代码
```

第五课：进军！神奇的程序世界

几天开课后，发现一些问题，，想法也有改变。既然开课，就希望能做成一套成功的课程，几年后能够传为经典的课程，所以课程需要更加系统化。

现在已经调整第一个项目为《七色光芒》，前四课均已重新编辑！

从第五课开始，将真正开启程序世界的大门！

本课主要内容：学习程序的流程、调用等。

1.1 循环控制

在上一课中我们使用 STC 软件自动生成了一个延时 1 秒的函数，如果以后需要延时 2 秒、延时半秒呢，要是每次都生成一个这个的函数，那将不可想象。

幸好有一个巧妙的办法：函数调用。

如果我们有一个延时 1ms 的函数，执行 1000 次不就是 1 秒了么，执行 500 次不就是半秒？多么机智的想法。

调用一个函数两次，我们可以写两行，调用三次，可以写三行，那调用 1000 次呢.....

幸好有“while”。在我们的主函数中，所有的程序都在一个无限循环的 while（1）里，其实它循环的次数是可以控制的。

知识点（15）while（？）循环

while（？）{ 当？的值是“真”的，括号里的程序会一直执行；当？的值是“假”的，括号里的程序不再执行。

什么是真？大于 0 的数都是真的，1 是真，1000 也是真。什么是假？0 就是假。

如果我们想要循环 1000 次，可以 while（1000）循环一次，变成 while（999）.....

到最后，变成 while（0），自然循环就停止了。

知识点（16）unsigned int n； 变量定义

想要一个数从 1000 减到 0，就先要定义这个数。

unsigned int 定义变量的关键字，意思是定义一个数，范围是 0 到 65535。

unsigned int n；定义了 n 这个数。n 只是一个名字，你也可以写成 abc 等都可以。

知识点（17） 1000 次循环的程序实例

```
n=1000;

while(n)

{

    //被循环的程序

    n=n-1;//每循环一次 n 减小 1

}
```

把前面所讲的组合起来，自己写一个函数吧！

知识点（18） 定义延时 n 毫秒函数实例

void delay_n_ms(unsigned int n)//括号里面有值代表有参数的函数，在调用这个函数时，可以顺便写 n 的大小。

```
{

    while(n)          //注意什么地方有分号，什么地方没分号

    {

        Delay1ms();

        n=n-1;//每循环一次 n 减小 1

    }

}
```

知识点（19） 调用带参数的函数

```
delay_n_ms(1000);    //在调用时，1000 这个值就传递给了 n
```

有了能随意控制延时时间的函数，就能让彩灯变化更加灵活，写一个每种色彩亮

0.1 秒，白色亮 1 秒的程序吧！

是不是绚丽多了？你还能想到别的变化组合吗？

```
#include<reg51.h>
```

```
sbit LED_R=P3^4;//定义红灯
```

```
sbit LED_G=P3^3;//定义绿灯
```

```
sbit LED_B=P3^2;//定义蓝灯
```

```
void Delay1ms()          //@12.000MHz
```

```
{
```

```
    unsigned char i, j;
```

```
    i = 12;
```

```
    j = 169;
```

```
    do
```

```
    {
```

```
        while (--j);
```

```
    } while (--i);
```

```
}
```

```
void delay_n_ms(unsigned int n) //自己定义的延时 n 毫秒函数
```

```
{
```

```
    while(n)
```

```
    {
```

```
        Delay1ms();
```

```
        n=n-1;//每循环一次 n 减小 1
```

```
    }
```

```
}
```

```
main()
```

```
{
```

```
while(1)
{
    LED_R=1;
    LED_G=0;
    LED_B=0;
    delay_n_ms(100);//红色亮 0.1 秒

    LED_R=1;
    LED_G=1;
    LED_B=0;
    delay_n_ms(100);//黄色亮 0.1 秒

    LED_R=0;
    LED_G=1;
    LED_B=0;
    delay_n_ms(100);//绿色亮 0.1 秒

    LED_R=0;
    LED_G=1;
    LED_B=1;
    delay_n_ms(100);//青色亮 0.1 秒

    LED_R=0;
    LED_G=0;
    LED_B=1;
    delay_n_ms(100);//蓝色亮 0.1 秒

    LED_R=1;
    LED_G=0;
    LED_B=1;
    delay_n_ms(100);//紫色亮 0.1 秒

    LED_R=1;
    LED_G=1;
    LED_B=1;
    delay_n_ms(1000);//白色亮 1.0 秒
}
}
```

1.2 按键输入

单片机的三要素是什么？我认为是输入、处理、输出。上一节已经学了简单的处理和输出，这一节再学一下输入你就可以毕业啦 $\geq \nabla \leq$

单片机最普遍的输入就是按键。

知识点 (20) 按键的工作原理：

单片机引脚设置为高电平，通过按键接到地。

按键没有按下：单片机引脚还是高电平；

按键被按下：单片机引脚接地，变成低电平。程序读一下引脚电平就可以知道有没有按下。

知识点 (21) 为什么通常按键接地

单片机的引脚在默认模式下，引脚内部是串联一个电阻接到正极的，当你的按键导通时，电流很小，不会有什么影响。

但是单片机的低电平是没有串联电阻的，如果你用按键连到正极，相当于将正短路，可能会损坏引脚。

程序如何读引脚的状态？同样，要想先读引脚，也要先定义才能用，在我们焊接的电路中，一个按键接在 P3.0 引脚上：

`sbit Key1=P3^0;`定义之后，不仅可以控制，还可以读。

知识点 (22) if 判断语句

`if(?) { 1 }` 如果？是真的，执行括号 1 里面的程序

`else { 2 }` 否则就执行 2 里面的程序

单片机有哪些语句？循环和判断，无非就这两种，所有复杂的程序都是这两个组合起来的。

现在你已经都学了，是不是感觉单片机其实也没那么复杂？

知识点 (23) =和==

等于 和 是否等于。

单片机语言中，有两种等于号，这点务必分清。

`n=1000;` `n` 等于 1000，是向 `n` 写入 1000

`n==1000;` `n` 是否等于 1000？是对 `n` 进行判断。

知识点 (24) 判断引脚状态的程序实例

`if(Key1==0)` //如果按键按下，引脚读到低电平

{

`LED_R=1;`

`LED_G=0;`

`LED_B=0;` //亮红灯

}

else

{

`LED_R=0;`

`LED_G=1;`

`LED_B=0;` //否则亮绿灯

}

赶快写程序试一下吧。

```

#include<reg51.h>

sbit LED_R=P3^4;//定义红灯
sbit LED_G=P3^3;//定义绿灯
sbit LED_B=P3^2;//定义蓝灯

sbit Key1=P3^0;//定义按键

void Delay1ms()                //@12.000MHz
{
    unsigned char i, j;

    i = 12;
    j = 169;
    do
    {
        while (--j);
    } while (--i);
}

void delay_n_ms(unsigned int n) //自己定义的延时 n 毫秒函数
{
    while(n)
    {
        Delay1ms();
        n=n-1;//每循环一次 n 减小 1
    }
}

main()
{
    while(1)
    {

        if(Key1==0) //如果按键按下，引脚读到低电平
        {
            LED_R=1;
            LED_G=0;
            LED_B=0; //亮红灯
        }
    }
}

```

```

        else
        {
            LED_R=0;
            LED_G=1;
            LED_B=0; //否则亮绿灯
        }

        delay_n_ms(100); //延时 0.1 秒 程序每秒检测 10 次按键
    }
}

```

1.3 多种模式切换

前面学习的按键检测只能简单的切换两种状态 ,如果想要通过按键改变七种色彩 该怎么做的呢 ?

通常一般的单片机教材在讲按键检测时 ,都会教你用 delay 延时等待按键弹起、等待按键稳定.....我不会这样给你讲。这样有什么问题吗 ?

我的导师在喝酒时都会大骂教材中的各种错误 , “等待按键弹起 ? 按键坏了弹不起来怎么办 ? 程序死机 ? ? ! ”

而在我看来 ,教材中的各种错误 ,影响远不止这么点。很多学生学了 delay 检测按键 ,一辈子都只会用 delay ,容易养成错误的程序思维 ,走向歧途 ,缺乏对单片机的 “时间片” 的概念 ,日后难以写出复杂、实时性的程序。所以 ,我要在一开始 ,就把 “时间片” 的概念深印在你心中。

知识点 (25) 单片机的运行与 “时间片”

首先要知道 ,单片机能干什么 ? 我们的电脑 CPU 实际上也可以看成是一个速度

超快的单片机，它能一边挂 QQ、一边看网页、一边打游戏，它是不是可以同时处理很多任务？NO，事实上，单片机同一时间，只能做一件事，CPU 只是聪明的划分出了很多很小的“时间片”，一个很短的时间内，只执行 QQ，另一个很短的时间内，只执行游戏，一秒钟内“时间片”切换几十次，看起来就像是同时在做多种事了。这种方式是单片机最正确的处理方式，以后在考虑程序时，先从时间片角度出发！

我们现在的任务很简单，检测按键，然后改变颜色，现在还用不到时间片这么高大上的东西，但对检测周期还要有一个概念：人类按一次按键大概需要 0.3 秒，我们大概每 0.15 秒检测一下按键就能满足需要。

知识点（26） 如何区分多种状态

我们的七色光芒有七种颜色，如果算上熄灭，至少有 8 种状态，怎么用程序对应 8 种状态呢？

你需要先定义一个数据：比如 mode；如果 mode 等于 0，代表熄灭，mode 等于 1，代表红色.....mode 不同的值就能代表不同状态。

知识点（27） unsigned char mode；

unsigned char 是定义数据的关键字，代表定义一个大小是 0~255 的数。255 种模式，够用了吧。

知识点（28） 全局变量

我们定义的 mode 在整个程序中都会用到，所以需要定义在程序开头，它就叫做全局变量。

定义在程序开头的变量默认都是 0。

想到怎么做了吗？我们每检测一次按键，就改变一下 mode 的值，指示颜色改

变了，再用 if 判断不同的值显示不同颜色不就可以了么？

本课的主要内容就是这些了，更多精彩下一课再见

```
#include<reg51.h>
```

```
sbit LED_R=P3^4;//定义红灯
```

```
sbit LED_G=P3^3;//定义绿灯
```

```
sbit LED_B=P3^2;//定义蓝灯
```

```
sbit Key1=P3^0;//定义按键
```

```
unsigned char mode; //定义一个数据，指示不同状态。定义后 mode 的值是 0
```

```
void Delay1ms()           //@12.000MHz
```

```
{
    unsigned char i, j;
```

```
    i = 12;
```

```
    j = 169;
```

```
    do
```

```
    {
```

```
        while (--j);
```

```
    } while (--i);
```

```
}
```

```
void delay_n_ms(unsigned int n) //自己定义的延时 n 毫秒函数
```

```
{
```

```
    while(n)
```

```
    {
```

```
        Delay1ms();
```

```
        n=n-1;//每循环一次 n 减小 1
```

```
    }
```

```
}
```

```
main()
```

```
{
```

```
    while(1)
```



```

{

if(Key1==0) //如果按键按下，引脚读到低电平
{
    mode=mode+1; //每按一次按键，颜色模式改变
    if(mode>7)
    {
        mode=0; //七种颜色切换完，回到 0 重新开始
    }
}

if(mode==0) //模式 0，熄灭
{
    LED_R=0;
    LED_G=0;
    LED_B=0;
} //没有 else 可以不写

if(mode==1) //模式 1，红色
{
    LED_R=1;
    LED_G=0;
    LED_B=0;
}

if(mode==2) //模式 2，黄色
{
    LED_R=1;
    LED_G=1;
    LED_B=0;
}

if(mode==3) //模式 3，绿色
{
    LED_R=0;
    LED_G=1;
    LED_B=0;
}

if(mode==4) //模式 4，青色
{
    LED_R=0;
    LED_G=1;
    LED_B=1;
}

```

```

    }

    if(mode==5) //模式 5 , 蓝色
    {
        LED_R=0;
        LED_G=0;
        LED_B=1;
    }

    if(mode==6) //模式 6 , 紫色
    {
        LED_R=1;
        LED_G=0;
        LED_B=1;
    }

    if(mode==7) //模式 7 , 白色
    {
        LED_R=1;
        LED_G=1;
        LED_B=1;
    }

    delay_n_ms(150); //程序每 0.15 秒检测一次按键

}

}

```

第六课：高大上的 PWM 是怎么回事

单片机深奥复杂吗？在没接触过的人看来，单片机神乎其神，但对于已经学到第六课的你，一定已经看穿它的真面目，不再恐惧。单片机有什么？很少的几个程序关键字、循环语句和判断语句，重要的东西就这些。所有复杂的程序都是由这些组成。

学到这里，基础的知识已经差不多都学了，剩下的就是培养正确的

程序思维，多动手，多在实际项目中练习。

单片机作为电路中的“司令官”，是发号施令的。单片机只有高电平、低电平，能够发出开、关的命令，控制 LED 点亮、熄灭。但在实际的需求中，只能控制开关是远远不够的，我们还需要控制灯泡的亮和暗、电机转速的快慢等，单片机能不能实现呢？好像听说过一个叫做 PWM 的东西？那是什么？

PWM 听起来高大上，实际上原理很简单，这节课就教你数字电路中最灵活最通用的技术：PWM

1.1 PWM 是什么

玩过航模的人，对 PWM 再熟悉不过：接收机是 PWM，有刷、无刷电调是 PWM，舵机是 PWM，充电器开关电源是 PWM，连 BB 响都是 PWM！航模只是一个缩影，不止航模，几乎所有的电子设备都在使用 PWM，那 PWM 到底是什么？

知识点（29）PWM 的简单解释

单片机只能输出高电平、低电平，5V 供电时高电平就是 5V，有没有办法输出 2.5V 呢？虽然不能直接输出 2.5V，但可以用等效的概念：

假设在 1 秒钟内，前半秒是高电平（5V），后半秒是低电平（0V），那一秒内的平均电压是多少？平均电压就可以当成 2.5V。

LED 如果半秒亮半秒灭肯定会感到明显的闪烁，那如果加快时间呢？

假设在 10ms 内，前 5ms 亮，后 5ms 灭。一秒钟有 100 个 10ms，相当于显示器的 100 帧，人眼只能看到 30 帧的画面，这样就在你没有发现闪烁的情况下，LED 亮度变成了原来的一半！PWM 的真相就是这样，非常快的高低电平切换，欺骗了你的眼睛。

知识点 (30) 用 PWM 调节亮度

PWM 有两个关键词：周期和占空比。

周期：前面举的例子，10ms 一个循环，10ms 就是 PWM 周期。对应的频率是 100Hz。

占空比：一个周期内，高电平时间的比例。10ms 内高电平 5ms，占空比就是 50%。对应的 LED 亮度就是 50%。

显然，调节占空比就可以改变亮度：10ms 内高电平 2ms，低电平 8ms，占空比 20%，亮度变成 20%。

已经明白原理的你有没有想到怎么用程序实现？

思考时间，自己思考.....

前面已经学了循环程序，加上可以控制时间的延时函数，我想你应该可以轻松构建一个 10ms 周期的 PWM 了：

```
#include<reg51.h>

sbit LED_R=P3^4;//定义红灯
sbit LED_G=P3^3;//定义绿灯
sbit LED_B=P3^2;//定义蓝灯

sbit Key1=P3^0;//定义按键

unsigned char count;    //定义一个数据，记录循环次数

void Delay1ms()          //@12.000MHz
{
    unsigned char i,j;

    i = 12;
    j = 169;
```

```

do
{
    while (--j);
} while (--i);
}

void delay_n_ms(unsigned int n) //自己定义的延时 n 毫秒函数
{
    while(n)
    {
        Delay1ms();
        n=n-1;//每循环一次 n 减小 1
    }
}

main()
{

    while(1)
    {
        count=count+1;    //每过 1ms 计数+1
        if(count>9)        //计数大于 9 重新开始计数，0~9 十毫秒周期
        {
            count=0;
        }

        if(2>count)        //在 count 0~9 变化中，2 大于 count 的次数是 2 次。改变 2 就可
以改变占空比
        {
            LED_R=1;
            LED_G=1;
            LED_B=1;        //2 次高电平，LED 亮白色
        }
        else
        {
            LED_R=0;
            LED_G=0;
            LED_B=0;        //8 次低电平，LED 熄灭，亮度 20%
        }
        delay_n_ms(1);    //每循环一次 1ms
    }
}

```



```
}
```

1.2 学一个正确的按键检测！

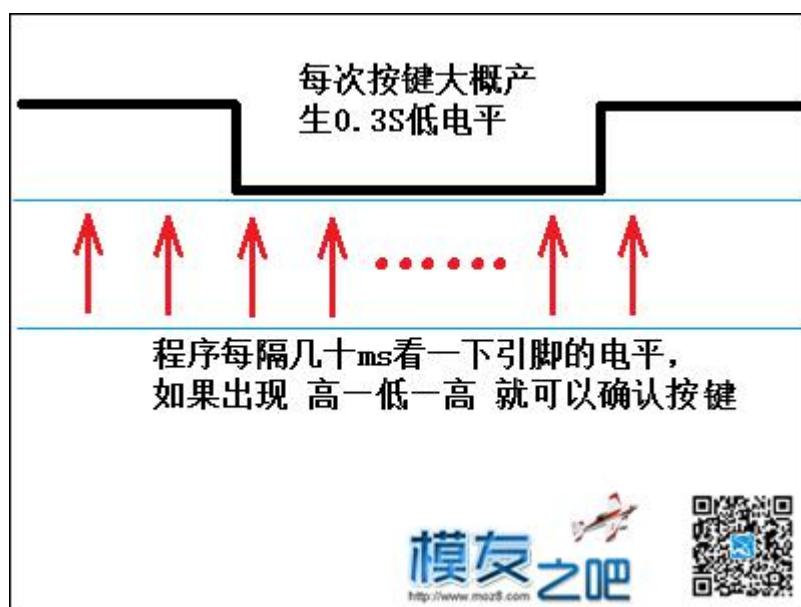
虽然学会了怎么用 PWM，但修改程序才能改亮度也太死板了点，你需要灵活运用按键来控制亮度。

前面已经狠狠的批斗了教材中错误的按键检测，现在要接着批斗：

假设人按一次按键 0.3 秒，如果用等待按键弹起的方法，虽然也能检测到，但是 CPU 负荷却被占用 30%！这么长时间我们 10ms 周期的 PWM 已经跑了 30 次了！如果有人按着按键不放，直接 CPU 负荷 100%，程序都死了，PWM 函数也不能走了！

所以你需要换个思维，学习更好的按键检测方法：

知识点（31） 基于跳变的按键检测



跳变式的按键检测不用死等，不浪费程序运行时间，每隔一段时间读一下引脚电平即可，间隔 10ms~100ms 都行。有人可能会有疑问：不处理按键抖动吗？

由于检测周期大于按键抖动时间，是不受按键抖动影响的！。

知识点（32） flag 标志位

程序读引脚只能读到现在的电平，如何同时读到过去和现在，比较低—高的跳变？

回到过去肯定是不可能的，所以你需要把过去的电平保存下来。

你可以定义一个数：比如 old，如果出现低电平，让 old=0。这里 old 就是一个指示过去电平的标志位。

简单的语句之所以能够组成复杂的程序，完全是各种标志位在起作用，控制着程序走向，学会用标志位才是你走向高手的第一步。

可能你还没有完全理解，那就用程序去验证你的想法吧：

```
#include<reg51.h>

sbit LED_R=P3^4;//定义红灯
sbit LED_G=P3^3;//定义绿灯
sbit LED_B=P3^2;//定义蓝灯

sbit Key1=P3^0;//定义按键

unsigned char Key1_old;    //定义一个数据，记录过去出现低电平

void Delay1ms()            //@12.000MHz
{
    unsigned char i,j;

    i = 12;
    j = 169;
    do
    {
        while (--j);
    } while (--i);
}
```

```

void delay_n_ms(unsigned int n) //自己定义的延时 n 毫秒函数
{
    while(n)
    {
        Delay1ms();
        n=n-1;//每循环一次 n 减小 1
    }
}

main()
{

    Key1_old=1;    //进入循环前，先初始化标志位。按键还没按下，old 是高电平
    LED_G=0;
    LED_B=0; //关闭两个 LED，只用红色 LED

    while(1)
    {

        if(Key1==1)
        {
            if(Key1_old==0) //如果本次是高电平，上次是低电平,检测到按键
            {

                if(LED_R==1)//如果 LED 亮让它灭，灭的让它亮，指示每次按键
                {
                    LED_R=0;
                }
                else
                {
                    LED_R=1;
                }
            }
        }

        Key1_old=Key1;    //保存高低电平

        delay_n_ms(10); //10ms 检测一次按键

    }
}

```

1.3 组合按键程序和 PWM 程序

学习单个的知识是很简单的,但把单独的程序组合成复杂的程序你会吗?学会组合程序就是你走向高手的第二步!

组合多个程序考验的是你对程序全局的把握能力。这里,我教你用“时间片”的角度去规划全局。

知识点 (33) 求最小时间片

多个程序对时间的需求是不一样的,要想组合两个程序,就要找到他们的最小时间片。

现在我们有二个程序,按键程序每 10ms 执行一次,PWM 程序每 1ms 执行一次,所以最小时间片就是 1ms!

知识点 (34) 确认系统循环周期

while (1) 每循环一次,所有程序执行一遍,相当于系统的一个循环,循环周期就是最小时间片 1ms。

知识点 (35) 多程序共用最小时间片

很简单的数学问题,按键程序 10ms 一次,PWM 1ms 一次,只要每执行 10 次 PWM 后执行一次按键检测即可。

```
#include<reg51.h>
```

```
sbit LED_R=P3^4;//定义红灯
sbit LED_G=P3^3;//定义绿灯
sbit LED_B=P3^2;//定义蓝灯
```

```
sbit Key1=P3^0;//定义按键
```

```
unsigned char Key1_old;    //定义一个数据,记录过去出现低电平
unsigned char time;        //定义一个数据,记录时间
```

```

unsigned char duty;           //定义一个数据，代表占空比
unsigned char count;         //定义一个数据，PWM 循环次数

void Delay1ms()               //@12.000MHz
{
    unsigned char i, j;

    i = 12;
    j = 169;
    do
    {
        while (--j);
    } while (--i);
}

void delay_n_ms(unsigned int n) //自己定义的延时 n 毫秒函数
{
    while(n)
    {
        Delay1ms();
        n=n-1; //每循环一次 n 减小 1
    }
}

main()
{
    Key1_old=1; //进入循环前，先初始化标志位。按键还没按下，old 是高电平

    while(1)
    {
        delay_n_ms(1); //系统周期，每循环一次 1ms

        count=count+1; //每过 1ms 计数+1
        if(count>9)
        {
            count=0; //计数大于 9 后又变成 0！，0~9 十毫秒 PWM 周期
        }

        if(duty>count) //在 count 0~9 变化中，duty 大于 count 的次数就是占空比
        {
            LED_R=1;

```

```

        LED_G=1;
        LED_B=1;    //改变 duty 改变高电平时间，亮度变化
    }
    else
    {
        LED_R=0;
        LED_G=0;
        LED_B=0;
    }

    time=time+1; //1ms 计数
    if(time>9)    //time 加到 10ms，从 0 开始
    {
        time=0;

        if(Key1==1)
        {
            if(Key1_old==0) //如果本次是高电平，上次是低电平,检测到按键
            {

                duty=duty+1; //每次按键改变一次亮度
                if(duty>9)
                {
                    duty=0;    //10 级亮度调节
                }

            }
        }

        Key1_old=Key1;    //保存高低电平
    }

}

}

```


第七课：扩展课！用掉你的单片机

如果你认真跟进学到第七课，你可以回头看看你都收获了什么？

学会了单片机焊接和下载；

学会了 Keil 编程软件的使用；

学会了 C 语言的基础语法，循环判断等基础语句；

学会了单片机引脚控制，输入输出；

学会了多种模式切换的程序写法；

学会了新型的按键检测方法；

学会了 PWM 控制亮度；

.....

是不是突然感觉自己会了好多 $O(n_n)O\sim\sim$

想要做到无基础的人快速入门单片机，首先就要抛弃那些深奥复杂的东西，直接说重点。很多教材上来就讲二进制、十进制、十六进制互相换算，左移右移等，掰着手指头数都绕晕，这里我可以负责的告诉你，只要会加减乘除四则运算，就可以学单片机！大部分情况用的都是十进制。

但是，只学了这些表层的知识是远远不够的，有很多单片机教材都声称

“七天学会单片机”，如果真是这样，楼主也可以说“七课学会单片机”。

然而实际上只是会背了乘法口诀，应用题还是不会做。

要想真正学会单片机，深入的做项目才是王道。在做项目的过程中，发现问题，解决问题，你才能真正进步。

所以我要提出一句话：“用掉你的单片机！”

这节课作为扩展课，就是要让你潜下心来，把第一个项目深入做好。这

也是我的课程的特点：以实际项目为单位进行，而不是以不同知识为单位进行。

1.1 联系实际，规划出一个有用的产品

跟做过就忘的单片机实验不同，我的目的是让你的学习历程中能够实际的做出一些东西，留下自豪的回忆，以后用单片机处理类似的问题也能快速应用。

你焊好了《七色光芒》，它能做什么？

最低要求：

大功率的灯珠有强烈的光芒，所以你至少可以做一个小手电筒；

色彩的控制，让你的手电筒与众不同，随时可以从白色光芒变成彩色，回头率瞬间+1；

中等要求：

不能调节亮度的手电不是好用的手电！

扩展要求：

为了能够引起远方人的注意力，你还需要爆闪模式！

模仿一下警车的闪烁模式，是不是更拉风？

还记不记得晚会上的彩色舞台灯，让你到哪都能嗨起来；

和女友浪漫时，打开彩色的呼吸灯功能，更加温馨.....

.....

RGB 灯光控制器现在还没有普及，这对很多人来说仍然是个新奇的玩意儿，只要发挥你的想象，它还能做更多的事。

1.2 程序模块化思维

看到上面罗列的各种功能，是不是没有想到，原本以为只是学了一个彩色流水

灯，实际上却有那么多用途！

那你想不想把那些功能亲自都实现呢？

其实所有功能的程序，在前六课都已经教给你了，你缺的只是灵活的组合，熟练的运用。

我们在第五课学了控制多种色彩模式，第六课学了 PWM 控制亮度，把它们组合起来，就能实现一个中等要求的产品。

在我们的电路中，有两个按键，正好分配一下，一个切换模式，一个控制亮度。

那么组合三个功能的程序如何写呢？很多人简单的程序还能写，但复杂一点就懵逼，不是不会，是因为没有学到一套正确的程序思维：

知识点（36） 程序模块化

为了让程序思路更清晰、以后复制写好的程序更方便，你的程序需要模块化。

什么是模块化？就是实现某个功能的程序尽可能的和其它程序独立，不要杂糅在一起。很多人一想到用按键控制色彩、亮度，自然而然的想到在按键程序里改变引脚、改变 PWM 等，这样做虽然也能实现，但 3 个程序被你杂糅成一个复杂的程序，不仅难以理解，改动也困难。

知识点（37） 消息传递机制

如果程序之间独立、模块化了，那怎么发生联系呢？你需要用“消息”喊话，把命令从一个程序传到另一个程序。

“消息”就是你定义的一个变量。比如你定义一个 Key1_action，如果检测到按键，就把 Key1_action=1，之后所有的程序都可以读一下 Key1_action，读到 1 就知道按键情况了，而不必每个程序都来检测一下按键。

先来模块化两个程序练练手吧，再自己试试把 PWM 程序也加入

```

#include<reg51.h>

sbit LED_R=P3^4;//定义红灯
sbit LED_G=P3^3;//定义绿灯
sbit LED_B=P3^2;//定义蓝灯

sbit Key1=P3^0;//定义按键 1
sbit Key2=P3^1;//定义按键 2

unsigned char Key1_old;    //定义一个数据，记录 Key1 过去出现低电平
unsigned char Key1_action; //定义一个数据，传递 Key1 是否动作的消息
unsigned char Key2_old;    //定义一个数据，记录 Key2 过去出现低电平
unsigned char Key2_action; //定义一个数据，传递 Key2 是否动作的消息

unsigned char mode;        //定义一个数据，代表不同模式
unsigned char light;       //定义一个数据，代表亮度等级

unsigned char time;        //定义一个数据，记录时间

unsigned char count;       //定义一个数据，PWM 循环次数
unsigned char duty_R;      //定义一个数据，传递红色亮度占空比
unsigned char duty_G;      //定义一个数据，传递绿色亮度占空比
unsigned char duty_B;      //定义一个数据，传递蓝色亮度占空比

void Delay1ms()             //@12.000MHz
{
    unsigned char i, j;

    i = 12;
    j = 169;
    do
    {
        while (--j);
    } while (--i);
}

void delay_n_ms(unsigned int n) //自己定义的延时 n 毫秒函数
{
    while(n)
    {
        Delay1ms();
    }
}

```

```

        n=n-1;//每循环一次 n 减小 1
    }
}

main()
{

    Key1_old=1;  //进入循环前，先初始化标志位。按键还没按下，old 是高电平
    Key2_old=1;

    while(1)
    {

        delay_n_ms(1);  //系统周期，每循环一次 1ms

        time=time+1;  //1ms 计数
        if(time>9)    //time 加到 10ms，从 0 开始,每 10ms 检测一次按键
        {
            time=0;

            if(Key1==1)
            {
                if(Key1_old==0) //如果本次是高电平，上次是低电平,检测到按键
                {
                    Key1_action=1;
                }
            }

            Key1_old=Key1;    //保存高低电平

            if(Key2==1)
            {
                if(Key2_old==0) //如果本次是高电平，上次是低电平,检测到按键
                {
                    Key2_action=1;
                }
            }

            Key2_old=Key2;    //保存高低电平
        }

        //上面是完全模块化的按键检测程序，只传递出两个 action 消息
        //////////////////////////////////////

```

```

if(Key1_action==1) //有按键消息，处理按键功能
{
    Key1_action=0; //处理过按键后要清 0

    mode++;
    if(mode>6) //按键 1 调节七种色彩
    {
        mode=0;
    }
}

if(Key2_action==1) //有按键消息，处理按键功能
{
    Key2_action=0; //处理过按键后要清 0

    light++;
    if(light>10) //按键 2 控制 10 级亮度
    {
        light=0;
    }
}

//上面是模块化的按键执行程序，只传递出 mode 模式、light 亮度两个消息
////////////////////////////////////

if(mode==0) //模式 0 红色
{
    duty_R=light; //红色占空比=亮度
    duty_G=0; //绿色、蓝色占空比为 0，熄灭，只有红色亮
    duty_B=0;
}
if(mode==1) //模式 1 橙色
{
    duty_R=light;
    duty_G=light;
    duty_B=0;
}
if(mode==2) //模式 2 绿色
{
    duty_R=0;
    duty_G=light;
    duty_B=0;
}

```

```

if(mode==3)  //模式 3 青色
{
    duty_R=0;
    duty_G=light;
    duty_B=light;
}
if(mode==4)  //模式 4 蓝色
{
    duty_R=0;
    duty_G=0;
    duty_B=light;
}
if(mode==5)  //模式 5 紫色
{
    duty_R=light;
    duty_G=0;
    duty_B=light;
}
if(mode==6)  //模式 6 白色
{
    duty_R=light;
    duty_G=light;
    duty_B=light;
}

```

//经过 mode 模式程序的处理，只剩下三个消息：每个 LED 的 duty 占空比

//最后只需要按照 duty 改变 PWM，就能实现亮度和色彩控制了

//这样看起来，程序的思路是不是非常清晰？

//你的发挥时间到了，自己在下面添加一个 PWM 程序，不然按键不能控制灯光

//.....

//.....

}

}

1.3 添加 PWM 程序

上一小节中，留了一个小坑让大家自己填。要求实现的功能是：用每个 LED 的占空比独立的控制三种颜色的亮度。

一共有三位童鞋提交了自己的答案：

第一位：只能亮白色或熄灭，达不到要求；

第二位：能够实现控制亮度、颜色的目的，但逻辑不清晰，引脚只能是 0 或 1，将 0~10 的 duty 值赋给引脚没有意义；

第三位：能够实现要求。

为什么要使用三个 PWM？独立控制每个 LED 亮度才能实现更多的色彩，方便接下来更复杂的程序编写。

```
count=count+1;  //每过 1ms 计数+1
    if(count>9)
    {
        count=0;      //计数大于 9 后又变成 0！，0~9 十毫秒 PWM 周期
    }

    if(duty_R>count)  //在 count 0~9 变化中，duty 大于 count 的次数就是占空比
    {
        LED_R=1;      //改变 duty 改变高电平时间，亮度变化
    }
    else
    {
        LED_R=0;
    }

    if(duty_G>count)  //绿色 LED PWM
    {
        LED_G=1;
    }
    else
    {
        LED_G=0;
```

```

    }

    if(duty_B>count)  //蓝色 LED PWM
    {
        LED_B=1;
    }
    else
    {
        LED_B=0;
    }
}

```

1.4 更多模式怎么加

首先你应该注意到，无论什么模式，都是有它的周期性的。

举例说最简单的爆闪模式：亮半秒灭半秒，两个状态不断循环；

双闪模式：亮半秒、灭半秒、亮半秒、灭三秒，四种状态循环；

七色舞台灯的模式：每种颜色亮 1 秒，7 个状态不断循环；

.....

举例程序如何实现爆闪模式，学到以后你可以试试自己实现其他功能。

循环程序必定有有一个数据计算时间，所以你要先定义一个数据：

```
unsigned int delay_time;  //定义一个数据，记录时间
```

注意数据大于 255 时，必须使用 unsigned int

增加了模式 7，所以按键改变模式的最大值也要相应更改。

添加模式 7 的代码：

更多更复杂的模式程序，就要看每个人的能力自己深入练习了。第一个项目的学习算是结束。

```

if(mode==7)  //模式 7 爆闪
{
    delay_time=delay_time+1;
    if(delay_time>1000)  //爆闪周期 1000ms
    {

```

```

        delay_time=0;
    }

    if(delay_time<400)    //前 0.4 秒全亮
    {
        duty_R=10;
        duty_G=10;
        duty_B=10;
    }
    else                  //后 0.6 秒熄灭
    {
        duty_R=0;
        duty_G=0;
        duty_B=0;
    }
}

```

第八课：新的玩具

学习单片机的精髓不在于你懂了多少知识，而在于你有没有学到方法。在第一个项目中，C 语言的语法/关键词等只讲了一点，但你可以看到，这并不影响你写出功能复杂的程序。可能你会渴望学到更多的 C 语言语法/关键词，以为学了更多就能写更复杂的程序，事实上并非如此，其它的语法/关键词，只能让你在实现同样功能时程序短两行而已，而决定你能不能写出实现某种功能的程序的，还是看你有没有相应的程序思维，程序方法。

程序思维和人平时的思维不太一样，如何锻炼出这种程序思维？还是要靠你多练习，多“悟”。在第八课，单片机课堂将会有新的“玩具”。

1.1 准备材料

如果说 DIY 的东西什么用处最广，那一定是遥控器！有了遥控器，你就能改装很多很多东西，很多麻烦的事情轻轻按一个按键就能搞定。现在我们初学单片机，基础还不是很牢固，先做一个简单的红外遥控器。



直插 STC15F104 单片机

价格 **¥1.70**

配送 广东深圳 至 福建泉州

数量

立即购买

承诺 ☒ 7天无理由 ☐

0038 一体化万能接收头 塑封红外接收头

价格 **¥0.32**淘宝价 **¥0.22** 首件优惠

配送 广东深圳 至 福建泉州丰泽区 快递 ¥3.0

数量 件(限购9998件)

立即购买

加入购物车



6*6*6MM微动开关

价格 **¥0.60**

配送 广东深圳 至 福建泉州

数量

立即购买

承诺 ☒ 7天无理由

LED灯珠 5MM 长脚

价格 **¥0.05**

颜色分类

数量

子

全新原装 松乐继电器 SRD-05VDC-SL-C T73 5V 蓝色
正品松乐 继电器价格 **¥1.25**淘宝价 **¥0.88** 首件优惠991
累计评论

配送 广东深圳 至 福建泉州丰泽区 快递 ¥3.00

数量 件(限购9998件)

立即购买

加入购物车

承诺 ☒ 7天无理由模友之吧
http://www.moz8.com



三极管 SS8550 双S大电流 TO-18 (50只)

价格 **¥2.85**

配送 广东深圳 至 福建泉州丰泽区 快递 ¥3.00

数量 件(库存9885件)

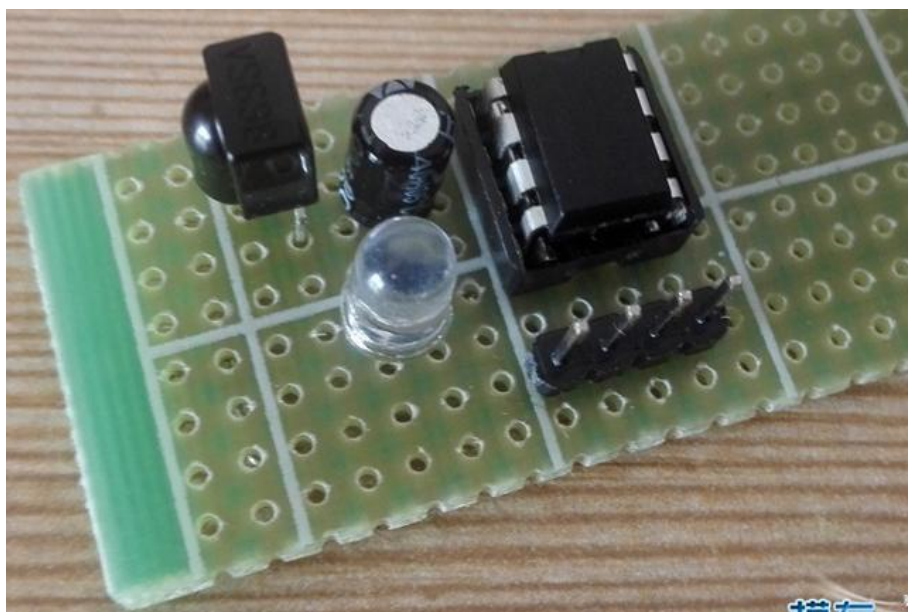
立即购买

 加入购物车

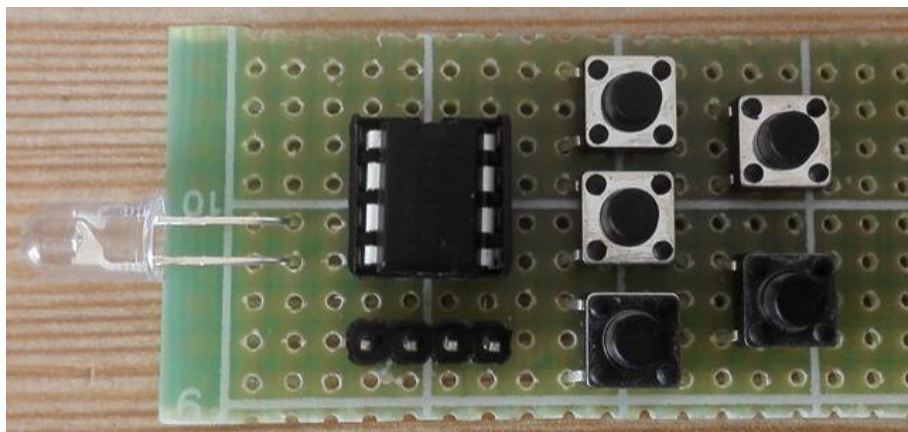
1.2 开工

先制作一只发射机。发射机非常简单，一个单片机，一个红外发射管，几个按键。再制作接收机。接收机也很简单，一个单片机，一个红外接收头，一个发光 LED。当然，这只是最简单的配置，如果你需要控制大电流的东西，就需要三极管或 MOS 管；如果要控制 220V 的家用电器，需要小心使用继电器。

先做一个简单的红外接收机：



再做一只红外遥控器：



第九课：看不见的信使——红外线

从这一课开始学习第二个项目：红外遥控器。遥控器无论是搞智能家居还是搞 DIY 都是不可缺少的东西，远距离遥控加上单片机智能处理，让你轻轻抬一抬手指就能实现以往不可想象的事情。

通过练习第一个项目，你已经跨过了单片机的“门槛”，但是还没有真正入门，想要真正走进单片机的大堂，你还需要很多努力。

本课主要内容是构建一套简易的遥控器、接收机。

1.1 还记得程序怎么写吗

第一个项目之后，不知道你有没有自己做练习？三日不做而手生，如果忘记怎么新建工程了，赶紧回去复习吧。

电路焊接好以后先做什么？要先写一个小程序试试你的电路工作了没。

红外接收头在收到信号时会输出低电平，没有信号时会输出高电平。这个电平人眼不能看到，所以你需要用一个 LED 发光来指示是否收到信号。

你需要新建一个项目文件夹、新建一个工程，新写一个程序：

可以先拿现成的空调、电视遥控器来测试下，按下按键时 LED 会闪，说明我们的接收机已经成功收到信号了！


```

项目02红外接收机.c
1 #include<reg51.h>
2
3 sbit LED=P3^5;    //定义指示灯
4 sbit IR_IN=P3^4;  //定义红外接收头引脚
5
6
7 main()
8 {
9
10  while(1)
11  {
12      if(IR_IN==0)  //收到红外数据引脚变成低电平
13      {
14          LED=1;    //如果收到数据让LED亮
15      }
16      else
17      {
18          LED=0;
19      }
20  }
21 }

```




1.2 发射 38kHz 红外信号

什么是红外线？红外线也是光的一种，只是人眼看不见而已。所有的物体都会发射红外线，电灯也会发射红外线，为什么开了灯红外接收机的灯不亮，只有按下电视遥控器灯才会亮？

这是因为在红外接收头里，已经秘密做了协议，只有收到闪烁频率为 38KHz 的红外线，才会认为有正确信号。

也就是说，如果你能把一个手电筒一秒内开关 3 万 8 千次，就可以被红外接收机识别。

人没有那么快，但是神奇的单片机可以啊，一秒开关一百万次都轻松，你需要动脑筋想想怎么去写程序实现？

很简单，算一下 38KHz 的周期大概是 26 微秒，让遥控器发射 13 微秒红外线，

再关闭 13 微秒，循环发射即可。

按下按键就能让接收机有反应的程序：

```

项目02红外遥控器.c*
1 #include<reg51.h>
2
3 sbit IR_LED=P3^5; //定义红外LED引脚
4 sbit Key1=P3^0; //定义按键1
5
6 void Delay13us() //@12.000MHz
7 {
8     unsigned char i;
9
10    i = 36;
11    while (--i);
12}
13
14 main()
15 {
16
17    while(1)
18    {
19        if(Key1==0) //按键按下变成低电平
20        {
21            IR_LED=1; //发射38KHz方波红外信号
22            Delay13us();
23            IR_LED=0;
24            Delay13us();
25        }
26        else
27        {
28            IR_LED=0;
29        }
30    }
31}

```



1.3 引脚推挽输出模式

做完 1.2，你应该会发现我们的遥控器距离没有电视的遥控器距离远，几米外就没反应了。为什么？

复习知识点（11），这是因为现在单片机的高电平输出电流只有不到 1ma，发

射的红外线太小。

除了像第一个项目一样用三极管放大电流，只驱动一个 LED 的话，还可以直接使用单片机 20ma 电流的推挽输出模式。

知识点 (38) 声明特殊寄存器 sfr P3M0=0xb2;

sfr 是声明寄存器的关键词；

P3M0 是设置 P3 口输出模式的寄存器；

0xb2 是 P3M0 这个寄存器的地址。

就跟 sbit 声明单片机引脚一样，声明之后程序里就可以使用。

知识点 (39) <reg51.h>里装了什么

有足够好奇心的话你可以右键 reg51、open 打开看一看。其实里面都是寄存器声明哦，以后学习会逐个用到。

至于 P3M0 寄存器，是新型 STC 单片机新加入的，所以你要自己声明。

知识点 (40) P3M0 寄存器是什么

什么是寄存器？你可以想象为一个寄存器里有 8 个小房间，每个房间控制一个引脚，可以装 1 或者 0，1 代表打开推挽模式，0 代表关。

P3M0 register

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
P3M0	B2H	name	P3M0.7	P3M0.6	P3M0.5	P3M0.4	P3M0.3	P3M0.2	P3M0.1	P3M0.0

P3.5引脚

0	0	1	0	0	0	0	0
---	---	---	---	---	---	---	---



知识点（41） 设置寄存器

怎么设置？直接写 $P3M0=0010\ 0000$ 吗？

当然不是，我们人类用的是 10 进制，计算机用的是二进制，需要先换算才能正确写。

我知道很多人一想到进制之间的换算就会头晕，所以我推荐你使用电脑自带的强大工具“计算器”

打开 Windows 计算器工具，选择程序员选项，点击 BIN 二进制选项，输入 0010 0000，（最前面的 0 不起作用）

你会看到 DEC 十进制选项里已经自动转换成 10 进制 32，所以正确的设置应该是 $P3M0=32$;

同时 HEX 十六进制选项里也显示了 20，写成 $P3M0=0x20$;也是可以的，加上 0x 代表 16 进制。



改了程序以后遥控距离瞬间提高十米：

```
#include<reg51.h>
```

```
sfr P3M0=0xb2;    //声明寄存器
```

```
sbit IR_LED=P3^5; //定义红外 LED 引脚
```

```

sbit Key1=P3^0;    //定义按键 1

void Delay13us()    //@12.000MHz
{
    unsigned char i;

    i = 36;
    while (--i);
}

main()
{
    P3M0=32;        //设置 P3.5 引脚为大电流模式

    while(1)
    {
        if(Key1==0)    //按键按下变成低电平
        {
            IR_LED=1;    //发射 38KHz 方波红外信号
            Delay13us();
            IR_LED=0;
            Delay13us();
        }
        else
        {
            IR_LED=0;
        }
    }
}

```

1.4 制定一个简单的通信协议

实验到这里，还有一个很不爽的问题需要解决：别的遥控器也能控制我们的接收机！

怎样让电视的遥控器只能控制电视，我们的遥控器只控制我们的呢？

这就跟二战时的电台谍战一样了，我们的电台可以收到敌人的信息，敌人的电台也能收到我们的，当时是怎么保密的呢？

欲保密，先加密。制定一套跟别人不同的通信协议，就能让别的遥控器对你失灵！

电视遥控器的通信协议很复杂，我们不必学他，给自己添烦恼，而是另辟蹊径，自己定义自己的协议。

这里先举个例子，定义一个超简单的协议：

发射 20ms 红外信号，代表按键 1；

发射 30ms 红外信号，代表按键 2；

发射 40ms 红外信号，代表按键 3.....

是不是特别简单(∩_∩)

在上一小节，我们发射红外的程序，发射一次是 26 微秒，发射 100 次才 2.6ms，20ms 需要 769 次。

用程序控制循环的时间，是不是跟以前的学的 delay_n_ms()类似？如果忘记可以回去复习一下知识点（18）。

void TX_IR(unsigned int n)//发射 N 个红外信号 时间是 26us*N

```
{
    while(n)
    {
        IR_LED=1;    //发射一个 38KHz 方波红外信号
        Delay13us();
        IR_LED=0;
        Delay13us();

        n=n-1;
    }
}
```

```
}
```

遥控器用旧知识可以搞定，接收机要测量时间需要新技能：

知识点（42） 用 while 测量低电平时间

```
while(IR_IN==0) //如果引脚一直为低电平，一直循环
{
    Delay1ms();

    time=time+1; //每循环一次，就记录了 1ms 时间
}
```

循环程序不仅可以用来延时，还可以用来计时。当引脚变成高电平 1 时，

IR_IN==0 就不是“真”的了，

循环自动结束，这时看 time 的值就知道时间过去多少毫秒了。

当遥控器发射了 20ms，由于环境的干扰，接收机可能收到 20ms，也可能只收到 18ms，有没有办法判断一个范围呢？

知识点（43） && “并且” 运算符

```
if(time>15 && time<25)
```

两个&&代表“并且”的意思，time 大于 15，并且小于 25，两个都正确才行。

用这样的方式，可以同时判断两个条件。

接收机参考程序，遥控器可以控制接收机的亮灭，更多功能就要看大家自己的练习了。甚至可以和第一个项目结合起来，遥控控制七色光芒。大家可以把的程序上传上来相互交流。

```
#include<reg51.h>
```

```
sbit LED=P3^5;    //定义指示灯
sbit IR_IN=P3^4;  //定义红外接收头引脚
```

```
unsigned char time;    //定义一个数据，记录时间
```

```
void Delay1ms()        //@@12.000MHz
{
    unsigned char i, j;

    i = 12;
    j = 169;
    do
    {
        while (--j);
    } while (--i);
}
```

```
main()
{

    while(1)
    {

        if(IR_IN==0)        //收到红外数据引脚变成低电平,开始测量
        {
            time=0; //先把时间清零
            while(IR_IN==0) //如果引脚一直为低电平，一直循环
            {
                Delay1ms();
                time=time+1; //每循环一次，就记录了 1ms 时间
            }
            //引脚变成高电平，循环结束

            if(time>15&&time<25)//16ms~24ms 低电平，按键 1 按下
            {
                LED=1;
            }

            if(time>25&&time<35)//26ms~34ms 低电平，按键 2 按下
            {
                LED=0;
            }
        }
    }
}
```



```

        }
    }
}

```

遥控器参考程序：

```

#include<reg51.h>

sfr P3M0=0xb2;    //声明寄存器

sbit IR_LED=P3^5; //定义红外 LED 引脚
sbit Key1=P3^0;   //定义按键 1
sbit Key2=P3^1;   //定义按键 2

unsigned char Key1_old;    //定义一个数据，记录 Key1 过去出现低电平
unsigned char Key1_action; //定义一个数据，传递 Key1 是否动作的消息
unsigned char Key2_old;    //定义一个数据，记录 Key2 过去出现低电平
unsigned char Key2_action; //定义一个数据，传递 Key2 是否动作的消息

void Delay1ms()           // @12.000MHz
{
    unsigned char i, j;

    i = 12;
    j = 169;
    do
    {
        while (--j);
    } while (--i);
}

void delay_n_ms(unsigned int n) //自己定义的延时 n 毫秒函数
{
    while(n)
    {
        Delay1ms();
        n=n-1; //每循环一次 n 减小 1
    }
}

```

```

    }
}

void Delay13us()           //@12.000MHz
{
    unsigned char i;

    i = 36;
    while (--i);
}

void TX_IR(unsigned int n)//发射 N 个红外信号 时间是 26us*N
{
    while(n)
    {
        IR_LED=1;         //发射一个 38KHz 方波红外信号
        Delay13us();
        IR_LED=0;
        Delay13us();

        n=n-1;
    }
}

main()
{
    IR_LED=0;//关闭红外 LED
    P3M0=32;      //设置 P3.5 引脚为大电流模式

    Key1_old=1;   //进入循环前，先初始化标志位。按键还没按下，old 是高电平
    Key2_old=1;

    while(1)
    {

        delay_n_ms(10);  //系统周期，每循环一次 10ms

        if(Key1==1)
        {
            if(Key1_old==0) //如果本次是高电平，上次是低电平,检测到按键
            {

```

```

        Key1_action=1;
    }
}

Key1_old=Key1;    //保存高低电平

if(Key2==1)
{
    if(Key2_old==0) //如果本次是高电平，上次是低电平,检测到按键
    {
        Key2_action=1;
    }
}

Key2_old=Key2;    //保存高低电平

//上面是完全模块化的按键检测程序，只传递出两个 action 消息
////////////////////////////////////

if(Key1_action==1) //有按键消息，处理按键功能
{
    Key1_action=0;    //处理过按键后要清 0

    TX_IR(769); //按键 1 发射 20ms 红外
}

if(Key2_action==1) //有按键消息，处理按键功能
{
    Key2_action=0;    //处理过按键后要清 0

    TX_IR(1153); //按键 2 发射 30ms 红外
}

}

}

```

第十课：扩展课！好想遥控点什么

照例，在每节课开头说点什么，以显得有意境.....

回到开设单片机课堂的初衷，为什么要开课？楼主想到的是两个字：传承。

一个人纵有再辉煌，也会有熄灭的一天。一个人纵有再努力，也不可能实现所有的目标。

楼主现在所做的事情，就是为了让楼主之后，能够有千千万万个“楼主”。

不知道你能否接下楼主的传承？

本课的主要内容是对红外遥控进行升级和更详细的讲解。

以后课程结构基本上像这样，一课制作，一课基础，一课扩展。

1.1 掌控时间

在第九课，有一个问题没有讲：如果对时间精度要求达到 1 微秒，用软件延时时就要注意了，需要微调：

以红外发射为例：

while(n)//发射 n 个 38KHz 方波红外信号

```
{
```

```
    IR_LED=1;
```

```
    Delay13us();
```

```
    IR_LED=0;
```

```
    Delay13us();//本意是延时 13us
```

```
    n=n-1; //实际上 n-1、while ( n ) 也会耗费一点时间
```

```
}
```

由于 n-1 这些程序也耗费时间，实际上一次循环的时间不是 26us，而是 27us 多，发射的是 36KHz 方波。虽然 36KHz 也能用，但最好还是调准确，以获得最好的效果。

知识点（44）空操作 _nop_();

delay 延时函数中常见 Nop（），Nop 就是空的意思，也是一行程序，只是单片机什么也没干而已。

当单片机时钟是 12MHz 时，一个 Nop 的时间是 1/12 us。所以使用 Nop 可以精确延时。

想要使用 Nop，必须在程序开头写上 #include<intrins.h>，否则会报错。

所以软件延时要想精确，不要盯着单个延时函数，而要从整个循环周期来出发。

显然，要想让总的时间是准确的 26 微秒，Delay13us()这个延时就要稍微短一些。

```
void Delay13us()           //@12.000MHz
{
    unsigned char i;

    _nop_();

    _nop_();

    i = 34;    //调整 Nop 和 i 可以微调时间

    while (--i);
}
```

使用微调后的新函数就可以准确的发射 38KHz 了。

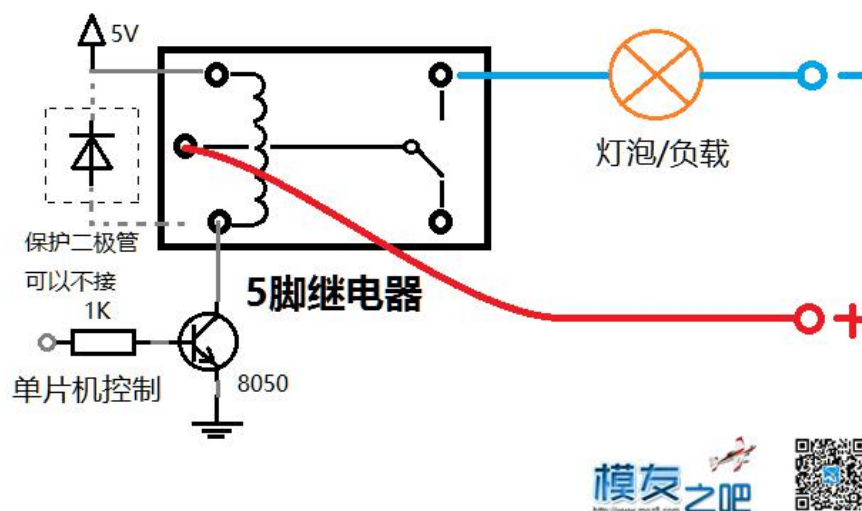
1.2 电动开关——继电器

接收机要想控制另一个电路的通断，就需要一个特殊的设备：继电器。

知识点（45） 继电器使用

继电器的线圈吸合需要几十 ma 电流，单片机的引脚电流是不够的，需要加三极管放大驱动。

使用继电器不仅可以控制低压设备，还可以直接控制 220V 家电。但高压操作很危险，没有相关经验禁止魔改。



驱动继电器的程序很简单，就跟点亮一个 LED 一样：

注意驱动三极管的引脚要设置为大电流推挽模式，如果忘了就回去看第九课知识点（41）

```
#include<reg51.h>
```

```
sfr P3M0=0xb2;    //声明寄存器
```

```
sbit Relay1=P3^3;    //定义继电器 1 引脚
```

```

void Delay1ms()                //@12.000MHz
{
    unsigned char i, j;

    i = 12;
    j = 169;
    do
    {
        while (--j);
    } while (--i);
}

void delay_n_ms(unsigned int n) //自己定义的延时 n 毫秒函数
{
    while(n)
    {
        Delay1ms();
        n=n-1;//每循环一次 n 减小 1
    }
}

main()
{
    P3M0=8;//P3.3 引脚设为大电流模式，寄存器实际值 0000 1000
            //复习知识点（41），使用电脑计算器工具，BIN 二进制输入 1000，得到 DEC 十进制 8
    while(1)
    {

        Relay1=0;
        delay_n_ms(1000);
        Relay1=1;           //让继电器每隔一秒自动打开、关闭
        delay_n_ms(1000);   //接上负载就能看到效果了
    }
}

```

1.3 做一个实用的红外遥控开关

将上一课的红外接收与本节课的继电器结合起来，你就有了一个可以遥控任何东西的遥控开关哦。

这里对整个项目进行一下规划：

考虑到大家对单片机熟练度的不同，你起码要实现的最低要求：

遥控器——使用两个按键，按键 1 控制打开，按键 2 控制关闭；

接收机——驱动一个继电器，接收命令打开、关闭。

中等要求：

使用多个按键、多个继电器，遥控多路开关。

扩展要求：

增加一些人性化功能，比如延时模式，某路继电器打开 10 秒后自动关闭；

增加循环模式，某路继电器自动的打开一段时间、关闭一段时间，，，

这里提供一个最低要求的接收机例程，按动遥控器，就可以听到继电器开关的啪啪声。

至于其他的功能，就要看各位自己的发挥和练习了！

```
#include<reg51.h>
```

```
sfr P3M0=0xb2;    //声明 引脚输出模式寄存器
```

```
sbit LED=P3^5;    //定义指示灯
```

```
sbit IR_IN=P3^4;  //定义红外接收头引脚
```

```
sbit Relay1=P3^3;    //定义继电器 1 引脚
```

```
unsigned char time;    //定义一个数据，记录时间
```

```
void Delay1ms()        //@12.000MHz
```

```
{
    unsigned char i,j;

    i = 12;
    j = 169;
    do
```

```

    {
        while (--j);
    } while (--i);
}

main()
{
    LED=0;
    Relay1=0; //先关闭继电器和指示灯
    P3M0=8;    //P3.3 引脚大电流模式

    while(1)
    {

        if(IR_IN==0)    //收到红外数据引脚变成低电平,开始测量
        {
            time=0; //先把时间清零
            while(IR_IN==0) //如果引脚一直为低电平，一直循环
            {
                Delay1ms();
                time=time+1; //每循环一次，就记录了 1ms 时间
            }
            //引脚变成高电平，循环结束

            if(time>15&&time<25)//16ms~24ms 低电平，按键 1 按下
            {
                LED=1;
                Relay1=1; //打开继电器
            }

            if(time>25&&time<35)//26ms~34ms 低电平，按键 2 按下
            {
                LED=0;
                Relay1=0; //关闭继电器
            }
        }

    }
}

```

第十一课：自制电压表

前面我们学习了很小的 8 脚单片机，小单片机有小的优点，简单便宜，但是引脚太少，做一些复杂的东西就无能为力。

今天我们的课程将迎来一个更厉害的新主角儿：STC15W408AS

15W408AS 这款单片机，资源丰富，价位适中，又有 16 脚、20 脚、28 脚三种封装可以选择，会是以后 DIY 教程的主力型号。

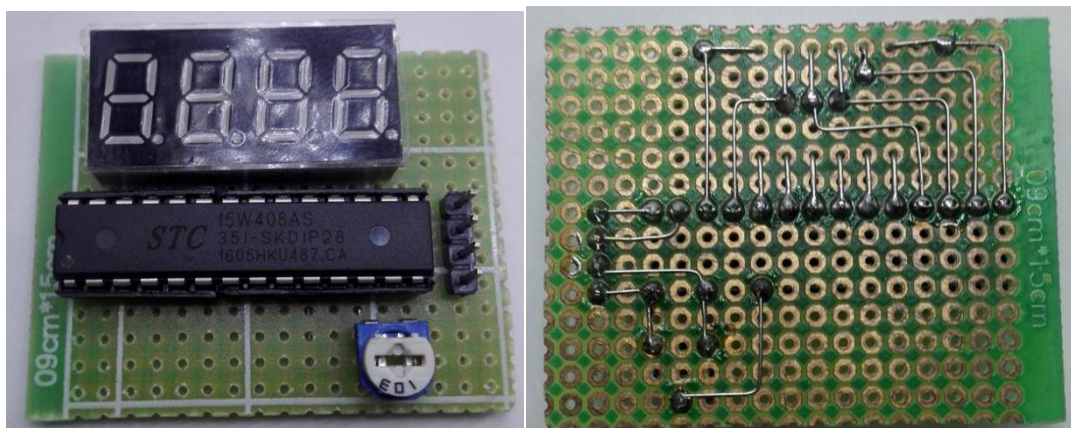
在本课中将使用 DIP28 直插 28 脚的单片机。

本课的主要内容：做一个电量显示器

1.2 材料准备

- 1、STC15W408AS DIP28 单片机
- 2、数码管 四位共阴
- 3、可调电阻 10K

1.3 焊接



第十二课：最便宜的显示屏

单片机为什么叫单片机？

看单片机的全称：单片微型计算机。

看看单片机里面有什么：有一个 12MHz 的处理器（CPU），有 0.5K 的 RAM（内存条），有 8K 的 FLASH（硬盘），，.....简直就是一台小电脑！

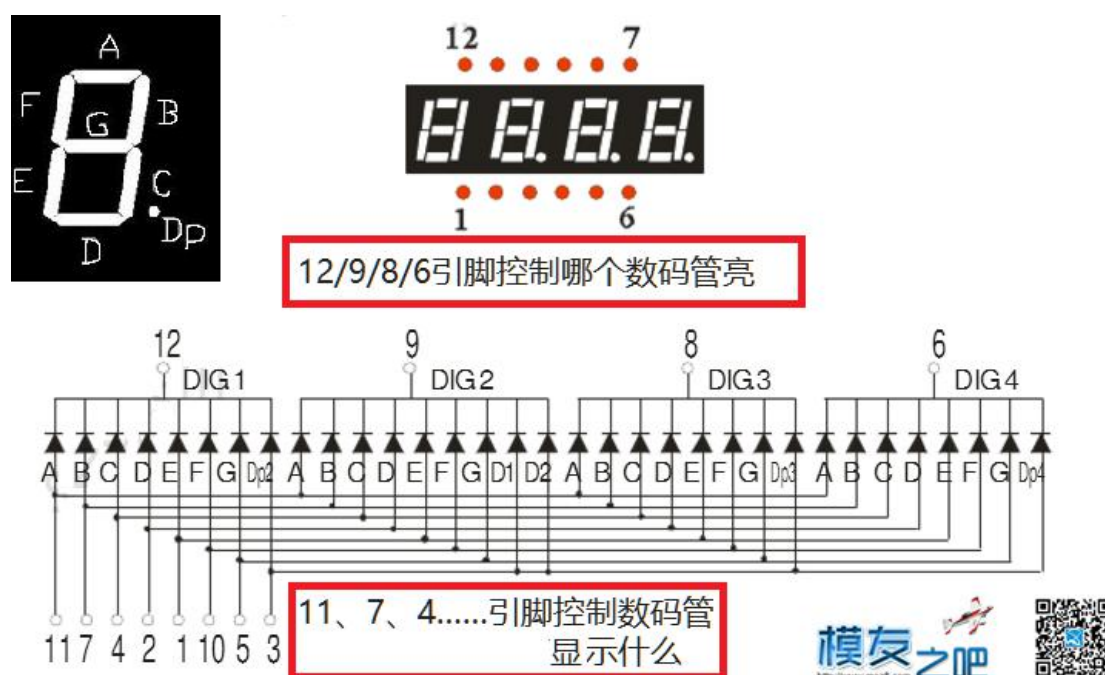
之前我们学了按键检测（键盘），现在还缺一个显示器就能凑成一个完整的电脑啦。

电压表自然需要一个显示器来显示，**本课的主要内容：驱动最便宜的显示器：数码管**

1.1 什么是数码管？

数码管其实就是很多个发光 LED，排列成一个“8”字形，控制数码管跟我们之前学习的点亮 LED 完全一样，只是原来是点亮一个，现在要点亮 $4 \times 8 = 32$ 个！

知识点（46）数码管结构



数码管的结构很简单，8 个 LED 构成一个数字。如果想要显示“3”，只需要让 A、B、C、D、G 为高电平（引脚 11、7、4、2、5），其余引脚为低电平即可。

使用第三课提到的方法，获得单片机的引脚图：



按照引脚，写出一个让数码管显示“3333”的小程序：

你也可以按照这个方法，显示出 1、2、3、4.....等

```
#include<reg51.h>
```

```
sbit DA=P2^4;           //数码管每个引脚
sbit DB=P3^2;
sbit DC=P3^6;
sbit DD=P2^0;
sbit DE=P2^1;
sbit DF=P2^3;
sbit DG=P3^5;
sbit DP=P3^7;
```

```
sbit B1=P2^5;           //四个数码管的引脚定义
sbit B2=P2^2;
sbit B3=P3^3;
sbit B4=P3^4;
```

```

main()
{
    DA=1;
    DB=1;
    DC=1;
    DD=1;
    DE=0;
    DF=0;
    DG=1;
    DP=0;    //A、B、C、D、G 引脚为高电平，显示 “3”

    B1=0;
    B2=0;
    B3=0;
    B4=0;    //四个数码管都显示

    while(1)
    {

    }
}

```

1.2 让数字动起来！

学会了显示 3，自己再显示出来 0~9 对你来说应该小意思！

只是好像有些不对劲，灯光很暗？

那是你忘记开启引脚的大电流模式了。这是个很重要的技能，要熟练掌握。开了之后立马高亮！

写一个小程序，让数字从 0 到 9 自动变化起来吧：

```

#include<reg51.h>

sfr P2M0=0x96;    //声明 P2 引脚模式寄存器
sfr P3M0=0xb2;    //声明 P3 引脚模式寄存器

sbit DA=P2^4;      //数码管每个引脚
sbit DB=P3^2;

```

```

sbit DC=P3^6;
sbit DD=P2^0;
sbit DE=P2^1;
sbit DF=P2^3;
sbit DG=P3^5;
sbit DP=P3^7;

sbit B1=P2^5;           //四个数码管的引脚定义
sbit B2=P2^2;
sbit B3=P3^3;
sbit B4=P3^4;

void Delay1ms()          //@12.000MHz
{
    unsigned char i, j;

    i = 12;
    j = 169;
    do
    {
        while (--j);
    } while (--i);
}

void delay_n_ms(unsigned int n) //延时 n 毫秒函数
{
    while(n)
    {
        Delay1ms();
        n=n-1;//每循环一次 n 减小 1
    }
}

main()
{
    P2M0=0x1B;    //需要高电平大电流的引脚 2.4、2.3、2.1、2.0，填入数值 0001 1011
    P3M0=0xE4;    //需要高电平大电流的引脚 3.7、3.6、3.5、3.2，填入数值 1110 0100

    B1=0;
    B2=0;
    B3=0;
    B4=0;//四个数码管都显示

```



```

while(1)
{
    DA=1;DB=1;DC=1;DD=1;DE=1;DF=1;DG=0;DP=0;//显示 "0"
    delay_n_ms(500);                //延时 0.5 秒

    DA=0;DB=1;DC=1;DD=0;DE=0;DF=0;DG=0;DP=0;//显示 "1"
    delay_n_ms(500);

    DA=1;DB=1;DC=0;DD=1;DE=1;DF=0;DG=1;DP=0;//显示 "2"
    delay_n_ms(500);

    DA=1;DB=1;DC=1;DD=1;DE=0;DF=0;DG=1;DP=0;//显示 "3"
    delay_n_ms(500);

    //.....
    //.....试着自己添加显示其他数字的程序

}
}

```

1.3 唯快不破

能不能让每个数码管显示的数字不一样呢？

显示数字的时候，同时也可以控制四个数码管中哪个显示，这样你就可以显示出来“0123”了。

在屏幕上滚动显示“0123”的小程序：

只能滚动显示，不能同时显示吗？你可以试着把延时 500ms 改成 100ms、20ms.....试试，自然就明白这一小节为什么叫这个标题。

```
#include<reg51.h>
```

```

sfr P2M0=0x96;    //声明 P2 引脚模式寄存器
sfr P3M0=0xb2;    //声明 P3 引脚模式寄存器

```

```

sbit DA=P2^4;           //数码管每个引脚
sbit DB=P3^2;
sbit DC=P3^6;
sbit DD=P2^0;
sbit DE=P2^1;
sbit DF=P2^3;
sbit DG=P3^5;
sbit DP=P3^7;

sbit B1=P2^5;           //四个数码管的引脚定义
sbit B2=P2^2;
sbit B3=P3^3;
sbit B4=P3^4;

void Delay1ms()          //@12.000MHz
{
    unsigned char i, j;

    i = 12;
    j = 169;
    do
    {
        while (--j);
    } while (--i);
}

void delay_n_ms(unsigned int n) //延时 n 毫秒函数
{
    while(n)
    {
        Delay1ms();
        n=n-1; //每循环一次 n 减小 1
    }
}

main()
{
    P2M0=0x1B;   //需要高电平大电流的引脚 2.4、2.3、2.1、2.0，填入数值 0001 1011
    P3M0=0xE4;   //需要高电平大电流的引脚 3.7、3.6、3.5、3.2，填入数值 1110 0100

    while(1) //显示 "0123"

```

```

{
    B1=0;B2=1;B3=1;B4=1;           //第一个数码管显示，二三四数码管关闭
    DA=1;DB=1;DC=1;DD=1;DE=1;DF=1;DG=0;DP=0;//显示 "0"
    delay_n_ms(500);                 //延时 0.5 秒

    B1=1;B2=0;B3=1;B4=1;           //第二个数码管显示
    DA=0;DB=1;DC=1;DD=0;DE=0;DF=0;DG=0;DP=0;//显示 "1"
    delay_n_ms(500);

    B1=1;B2=1;B3=0;B4=1;           //第三个数码管显示
    DA=1;DB=1;DC=0;DD=1;DE=1;DF=0;DG=1;DP=0;//显示 "2"
    delay_n_ms(500);

    B1=1;B2=1;B3=1;B4=0;           //第四个数码管显示
    DA=1;DB=1;DC=1;DD=1;DE=0;DF=0;DG=1;DP=0;//显示 "3"
    delay_n_ms(500);

}
}

```

1.4 程序模块化

做过前三小节的练习后，相信你已经学会了数码管的显示。

实际上四位数码管同一时间只能显示一位，因此想要让四位“同时显示”，就要利用单片机的“快”，在人眼察觉到之前就把四个依次显示一遍。这样，由于视觉暂留，人眼就会看到四位数码管同时在亮。这种显示方法就是“动态扫描法”。那么人眼能够察觉到闪烁的时间是多少呢？

我们在看视频的时候都会关心一个参数：屏幕刷新率（每秒帧数）。一般刷新率达到 50Hz 以上，人眼就不能感觉到了。

也就是说四个数码管显示一遍的时间小于 20ms，每个显示小于 5ms，就能够稳定显示。

当然只学会如何显示是远远不够的，显示程序是给别的程序服务的，为了方便以后的使用，你要学会把程序模块化起来：就像我们一开始写的函数 `delay_n_ms`

(n) 一样，会重复使用到的小程序可以封装成函数，随时调用。

这里我们封装了一个新函数：

display (x) //显示数字 x

```
#include<reg51.h>
```

```
sfr P2M0=0x96;    //声明 P2 引脚模式寄存器
```

```
sfr P3M0=0xb2;    //声明 P3 引脚模式寄存器
```

```
sbit DA=P2^4;      //数码管每个引脚
```

```
sbit DB=P3^2;
```

```
sbit DC=P3^6;
```

```
sbit DD=P2^0;
```

```
sbit DE=P2^1;
```

```
sbit DF=P2^3;
```

```
sbit DG=P3^5;
```

```
sbit DP=P3^7;
```

```
sbit B1=P2^5;      //四个数码管的引脚定义
```

```
sbit B2=P2^2;
```

```
sbit B3=P3^3;
```

```
sbit B4=P3^4;
```

```
unsigned char display_time;    //定义变量用于显示次数
```

```
unsigned char Data1;          //第一个数码管要显示的数据
```

```
unsigned char Data2;          //第二个数码管要显示的数据
```

```
unsigned char Data3;          //第三个数码管要显示的数据
```

```
unsigned char Data4;          //第四个数码管要显示的数据
```

```
void Delay1ms()                //@12.000MHz
```

```
{
```

```
    unsigned char i, j;
```

```
    i = 12;
```

```
    j = 169;
```

```
    do
```

```

    {
        while (--j);
    } while (--i);
}

void delay_n_ms(unsigned int n) //延时 n 毫秒函数
{
    while(n)
    {
        Delay1ms();
        n=n-1; //每循环一次 n 减小 1
    }
}

void display(unsigned char x) //控制数码管显示内容的函数
{
    //判断 x 的值来决定显示什么
    if(x==0){DA=1;DB=1;DC=1;DD=1;DE=1;DF=1;DG=0;DP=0;} //显示 "0"

    if(x==1){DA=0;DB=1;DC=1;DD=0;DE=0;DF=0;DG=0;DP=0;} //显示 "1"

    if(x==2){DA=1;DB=1;DC=0;DD=1;DE=1;DF=0;DG=1;DP=0;} //显示 "2"

    if(x==3){DA=1;DB=1;DC=1;DD=1;DE=0;DF=0;DG=1;DP=0;} //显示 "3"

    if(x==4){DA=0;DB=1;DC=1;DD=0;DE=0;DF=1;DG=1;DP=0;} //显示 "4"

    if(x==5){DA=1;DB=0;DC=1;DD=1;DE=0;DF=1;DG=1;DP=0;} //显示 "5"

    if(x==6){DA=1;DB=0;DC=1;DD=1;DE=1;DF=1;DG=1;DP=0;} //显示 "6"

    if(x==7){DA=1;DB=1;DC=1;DD=0;DE=0;DF=0;DG=0;DP=0;} //显示 "7"

    if(x==8){DA=1;DB=1;DC=1;DD=1;DE=1;DF=1;DG=1;DP=0;} //显示 "8"

    if(x==9){DA=1;DB=1;DC=1;DD=1;DE=0;DF=1;DG=1;DP=0;} //显示 "9"

}

main()
{
    P2M0=0x1B; //需要高电平大电流的引脚 2.4、2.3、2.1、2.0，填入数值 0001 1011
}

```

```
P3M0=0xE4;    //需要高电平大电流的引脚 3.7、3.6、3.5、3.2，填入数值 1110 0100
```

```
while(1)
{
    delay_n_ms(4); //系统周期 4ms

    Data1=4; //四个数码管显示的数据
    Data2=5;
    Data3=6;
    Data4=7;

    //////////////////////////////////////
    //显示程序模块化，只需要改变 Data1/2/3/4 的值就可以控制显示内容

    display_time=display_time+1;
    if(display_time>3)
    {
        display_time=0; //0、1、2、3 四个循环
    }

    if(display_time==0) //第 0 次显示第一个数码管
    {
        B1=0;B2=1;B3=1;B4=1;
        display(Data1);           //显示第 1 位数据
    }

    if(display_time==1) //第 1 次显示第二个数码管
    {
        B1=1;B2=0;B3=1;B4=1;
        display(Data2);           //显示第 2 位数据
    }

    if(display_time==2) //第 2 次显示第三个数码管
    {
        B1=1;B2=1;B3=0;B4=1;
        display(Data3);           //显示第 3 位数据
    }

    if(display_time==3) //第 3 次显示第四个数码管
    {
        B1=1;B2=1;B3=1;B4=0;
        display(Data4);           //显示第 4 位数据
    }
}
```

```
    }
}
```

第十三课：扩展课！电压变成数字

单片机中运行的是 0、1、0、1 的数字信号，只能知道引脚是高电平还是低电平，怎么能测量电压呢？

在 15W408AS 单片机中，集成了一个神奇的东西：模拟/数字转换器，简称 AD 转换，能够将引脚上的电压换算成数字。

有了它，你不仅能测电压，还能测电流，加个热敏电阻单片机就能感知到温度，加个光敏电阻单片机就能知道是白天还是黑夜.....学会 AD 转换，你的单片机将大有可为。

本课的主要内容：学会单片机测量电压，做一个小电压表

1.1 模数转换原理

首先要知道的一点是：模数转换并不能直接告诉你电压是多少，只会告诉你一个分数。简单的模数转换，精度是 255 个等级，如果单片机电源电压是 5V，相当于 5V 是满分 255。如果你的引脚上电压是 1V，你只能得 51 分。

当然，相反的，如果你得了 51 分，说明引脚上是 1V。

AD 转换的流程很简单，三步走：

- 1、设置相应的引脚为模拟输入功能；
- 2、启动一次转换；
- 3、等待转换完成，读取转换结果。

学习新的功能必然会学到新的寄存器。先说一个最简单的：

知识点（47） sfr ADC_RES=0xBD; 声明 转换结果寄存器

转换出来的分数保存在这个寄存器里，范围 0~255。

假如写程序 `a=ADC_RES`; `a` 就获得转换结果了。

如果你已经熟练掌握了之前学的引脚模式寄存器设置，那么设置引脚的模拟功能也是非常轻松的了：

STC 单片机的 P1 口引脚有模拟功能，本次实验用的电位器接在了 P1.7 引脚上。

因此需先设置 P1.7 引脚为模拟输入：

知识点（48） `sfr P1ASF=0x9D` 声明 P1 口模拟功能寄存器

P1ASF：P1口模拟功能控制寄存器

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
P1ASF	9DH	name	P17ASF	P16ASF	P15ASF	P14ASF	P13ASF	P12ASF	P11ASF	P10ASF

寄存器名 寄存器地址

P1.7引脚

1	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

1：设置为模拟功能

0：不设置

模友之吧
http://www.mcu8.com



设置 P1.7 引脚对应的程序是：`P1ASF=0x80`;

本节的难点在于最后这个寄存器，稍微复杂：

知识点（49） `sfr ADC_CONTR=0xBC` 声明 模数转换寄存器

ADC_CONTR：ADC控制寄存器

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
ADC_CONTR	BCH	name	ADC_POWER	SPEED1	SPEED0	ADC_FLAG	ADC_START	CHS2	CHS1	CHS0

AD转换的开关

开始AD转换

引脚选择

1	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---

这个寄存器里面参数较多，不过我们只需要用该用到的就行了，别的暂时不用理会。

寄存器的后三位是引脚选择，000~111，0 到 7 代表 P1.0~P1.7 引脚。

开始 P1.7 引脚 AD 转换，你可以直接设置为 1000 1111，也可以分开再相加
1000 1000+111

对应程序，可以写作 ADC_CONTR=0x8F; 也可以写作
ADC_CONTR=0x88+7;

知识点（50）AD 转换例程

```
P1ASF=0x80; //设置 P1.7 引脚
```

```
ADC_CONTR=0x88+7; //开始 P1.7 引脚转换
```

```
Delay50us(); //等待 AD 转换完成
```

```
a=ADC_RES; //读取转换结果
```

单片机时钟为 12M 时，一次 AD 转换最慢也是 45us，所以等待 50us 即可保证转换完成。

1.2 显示测量结果

上一课学了数码管的显示，加上这节课的 AD 转换，你就能测量出电位器上的电压啦。

不过每个数码管都只能独立的显示 0~9，假如你有一个数据是 123，你需要写个小程序把它拆开成百位、十位、个位才行。

用我们小学数学的知识来解答：

$123/100=1$ 余数 23，百位显示 1； $23/10=2$ 余数 3，十位显示 2，个位显示 3。

单片机由于没有小数点，它的除法跟小学生的水平是差不多的，你可千万别嫌它笨哈哈：

知识点（51）单片机的除法和余数

/：整除符号。单片机里面，29 除以 10 结果不是 2.9，真正的结果没有小数也

没有四舍五入，是 2；

%：求余数符号。29 除 10 的余数自然是 9。

求 123 的十位的例程：a=123%100/10; //先求 100 的余数是 23，再除以 10 得 2。

显示电位器 ADC 结果的程序，试着思考下怎么把 AD 结果换算成电压？

```
#include<reg51.h>
```

```
sfr P2M0=0x96;    //声明 P2 引脚模式寄存器
sfr P3M0=0xb2;    //声明 P3 引脚模式寄存器
sfr P1ASF=0x9D;    //声明 P1 口模拟功能寄存器
sfr ADC_RES=0xBD;  //声明 ADC 转换结果寄存器
sfr ADC_CONTR=0xBC; //声明 ADC 控制寄存器
```

```
sbit DA=P2^4;          //数码管每个引脚
sbit DB=P3^2;
sbit DC=P3^6;
sbit DD=P2^0;
sbit DE=P2^1;
sbit DF=P2^3;
sbit DG=P3^5;
sbit DP=P3^7;
```

```
sbit B1=P2^5;          //四个数码管的引脚定义
sbit B2=P2^2;
sbit B3=P3^3;
sbit B4=P3^4;
```

```
unsigned char display_time;    //定义变量用于显示次数
```

```
unsigned char Data1;    //第一个数码管要显示的数据
unsigned char Data2;    //第二个数码管要显示的数据
unsigned char Data3;    //第三个数码管要显示的数据
unsigned char Data4;    //第四个数码管要显示的数据
```

```
unsigned char adc;    //保存 AD 转换结果
```

```
void Delay1ms()    // @12.000MHz
```

```

{
    unsigned char i, j;

    i = 12;
    j = 169;
    do
    {
        while (--j);
    } while (--i);
}

void delay_n_ms(unsigned int n) //延时 n 毫秒函数
{
    while(n)
    {
        Delay1ms();
        n=n-1; //每循环一次 n 减小 1
    }
}

void Delay50us() // @12.000MHz
{
    unsigned char i, j;

    i = 1;
    j = 146;
    do
    {
        while (--j);
    } while (--i);
}

void display(unsigned char x) //控制数码管显示内容的函数
{
    //判断 x 的值来决定显示什么
    if(x==0){DA=1;DB=1;DC=1;DD=1;DE=1;DF=1;DG=0;DP=0;} //显示 "0"

    if(x==1){DA=0;DB=1;DC=1;DD=0;DE=0;DF=0;DG=0;DP=0;} //显示 "1"

    if(x==2){DA=1;DB=1;DC=0;DD=1;DE=1;DF=0;DG=1;DP=0;} //显示 "2"

    if(x==3){DA=1;DB=1;DC=1;DD=1;DE=0;DF=0;DG=1;DP=0;} //显示 "3"

    if(x==4){DA=0;DB=1;DC=1;DD=0;DE=0;DF=1;DG=1;DP=0;} //显示 "4"
}

```

```

if(x==5){DA=1;DB=0;DC=1;DD=1;DE=0;DF=1;DG=1;DP=0;} //显示 "5"

if(x==6){DA=1;DB=0;DC=1;DD=1;DE=1;DF=1;DG=1;DP=0;} //显示 "6"

if(x==7){DA=1;DB=1;DC=1;DD=0;DE=0;DF=0;DG=0;DP=0;} //显示 "7"

if(x==8){DA=1;DB=1;DC=1;DD=1;DE=1;DF=1;DG=1;DP=0;} //显示 "8"

if(x==9){DA=1;DB=1;DC=1;DD=1;DE=0;DF=1;DG=1;DP=0;} //显示 "9"

}

main()
{
    P2M0=0x1B; //需要高电平大电流的引脚 2.4、2.3、2.1、2.0，填入数值 0001 1011
    P3M0=0xE4; //需要高电平大电流的引脚 3.7、3.6、3.5、3.2，填入数值 1110 0100

    while(1)
    {
        delay_n_ms(4); //系统周期 4ms

        P1ASF=0x80; //设置 P1.7 引脚
        ADC_CONTR=0x88+7; //开始 P1.7 引脚转换
        Delay50us(); //等待 AD 转换完成
        adc=ADC_RES; //读取转换结果

        Data1=0;
        Data2=adc/100; //百位数据
        Data3=adc%100/10; //十位数据
        Data4=adc%10; //个位数据

        //////////////////////////////////////
        //显示程序模块化，只需要改变 Data1/2/3/4 的值就可以控制显示内容

        display_time=display_time+1;
        if(display_time>3)
        {
            display_time=0; //0、1、2、3 四个循环
        }

        if(display_time==0) //第 0 次显示第一个数码管

```

```

{
    B1=0;B2=1;B3=1;B4=1;
    display(Data1);           //显示第 1 位数据
}

if(display_time==1)//第 1 次显示第二个数码管
{
    B1=1;B2=0;B3=1;B4=1;
    display(Data2);           //显示第 2 位数据
}

if(display_time==2)//第 2 次显示第三个数码管
{
    B1=1;B2=1;B3=0;B4=1;
    display(Data3);           //显示第 3 位数据
}

if(display_time==3)//第 3 次显示第四个数码管
{
    B1=1;B2=1;B3=1;B4=0;
    display(Data4);           //显示第 4 位数据
}

}
}

```

1.3 项目规划

上次说到如何把 ADC 转换结果转换为电压，这个算法应该中学生都会：（前提是单片机 5V 要准）

电压 x adc

_____ = _____

5V 255

所以电压 $x = \text{adc} / 255 * 5 = \text{adc} / 51$ 。假设 $\text{adc} = 128$, $x = 2.5$ 。

不过要注意，由于单片机没有小数点，你要是想精确到小数点后，需要把 adc 扩大 10 倍， $x = 25$ ，显示时再补上小数点。

思考题： $a = 29 * 10 / 10$; 与 $b = 29 / 10 * 10$; 有什么区别？

表面上好像一样，实际上大不相同！

$a = 29 * 10 / 10 = 290 / 10 = 29$;

$b = 29 / 10 * 10 = 2 * 10 = 20$;

由于单片机的除法会损失精度，所以必须先乘后除。

学了新知识，就要学以致用。本期的项目就是要做一个实用的小电压表。

根据已学到的知识，你起码可以完成的最低要求：

单片机使用稳压 5V 或 3.3V 供电，测量一路 0~5V 或 0~3.3V 电压。

中等要求：

单片机稳压供电，用 10k、10k 电阻分压，量程增加 2 倍。

增加数码管显示小数点功能。

扩展要求：

多路不同量程电压测量，自动切换显示或按键切换.....

使用 10 位 ADC 或多次测量求平均值等方法提高精度.....

一个实现中等要求的小程序：

```
#include<reg51.h>
```

```
sfr P2M0=0x96;    //声明 P2 引脚模式寄存器
sfr P3M0=0xb2;    //声明 P3 引脚模式寄存器
sfr P1ASF=0x9D;    //声明 P1 口模拟功能寄存器
sfr ADC_RES=0xBD;  //声明 ADC 转换结果寄存器
```



```
sfr ADC_CONTR=0xBC;//声明 ADC 控制寄存器
```

```
sbit DA=P2^4; //数码管每个引脚
```

```
sbit DB=P3^2;
```

```
sbit DC=P3^6;
```

```
sbit DD=P2^0;
```

```
sbit DE=P2^1;
```

```
sbit DF=P2^3;
```

```
sbit DG=P3^5;
```

```
sbit DP=P3^7;
```

```
sbit B1=P2^5; //四个数码管的引脚定义
```

```
sbit B2=P2^2;
```

```
sbit B3=P3^3;
```

```
sbit B4=P3^4;
```

```
unsigned char display_time; //定义变量用于显示次数
```

```
unsigned char Data1; //第一个数码管要显示的数据
```

```
unsigned char Data2; //第二个数码管要显示的数据
```

```
unsigned char Data3; //第三个数码管要显示的数据
```

```
unsigned char Data4; //第四个数码管要显示的数据
```

```
unsigned char count; //计时
```

```
unsigned int adc; //AD 转换计算
```

```
void Delay1ms() // @12.000MHz
```

```
{
    unsigned char i, j;
```

```
    i = 12;
```

```
    j = 169;
```

```
    do
```

```
    {
```

```
        while (--j);
```

```
    } while (--i);
```

```
}
```

```
void delay_n_ms(unsigned int n) //延时 n 毫秒函数
```

```
{
```

```
    while(n)
```

```

    {
        Delay1ms();
        n=n-1;//每循环一次 n 减小 1
    }
}

void Delay50us()                //@12.000MHz
{
    unsigned char i, j;

    i = 1;
    j = 146;
    do
    {
        while (--j);
    } while (--i);
}

void display(unsigned char x)//控制数码管显示内容的函数
{
    //判断 x 的值来决定显示什么
    if(x==0){DA=1;DB=1;DC=1;DD=1;DE=1;DF=1;DG=0;DP=0;} //显示 "0"
    if(x==1){DA=0;DB=1;DC=1;DD=0;DE=0;DF=0;DG=0;DP=0;} //显示 "1"
    if(x==2){DA=1;DB=1;DC=0;DD=1;DE=1;DF=0;DG=1;DP=0;} //显示 "2"
    if(x==3){DA=1;DB=1;DC=1;DD=1;DE=0;DF=0;DG=1;DP=0;} //显示 "3"
    if(x==4){DA=0;DB=1;DC=1;DD=0;DE=0;DF=1;DG=1;DP=0;} //显示 "4"
    if(x==5){DA=1;DB=0;DC=1;DD=1;DE=0;DF=1;DG=1;DP=0;} //显示 "5"
    if(x==6){DA=1;DB=0;DC=1;DD=1;DE=1;DF=1;DG=1;DP=0;} //显示 "6"
    if(x==7){DA=1;DB=1;DC=1;DD=0;DE=0;DF=0;DG=0;DP=0;} //显示 "7"
    if(x==8){DA=1;DB=1;DC=1;DD=1;DE=1;DF=1;DG=1;DP=0;} //显示 "8"
    if(x==9){DA=1;DB=1;DC=1;DD=1;DE=0;DF=1;DG=1;DP=0;} //显示 "9"

    if(x==10){DA=1;DB=1;DC=1;DD=1;DE=1;DF=1;DG=0;DP=1;} //显示 "0" 带小数点
    if(x==11){DA=0;DB=1;DC=1;DD=0;DE=0;DF=0;DG=0;DP=1;} //显示 "1" 带小数点
    if(x==12){DA=1;DB=1;DC=0;DD=1;DE=1;DF=0;DG=1;DP=1;} //显示 "2" 带小数点
    if(x==13){DA=1;DB=1;DC=1;DD=1;DE=0;DF=0;DG=1;DP=1;} //显示 "3" 带小数点
    if(x==14){DA=0;DB=1;DC=1;DD=0;DE=0;DF=1;DG=1;DP=1;} //显示 "4" 带小数点
    if(x==15){DA=1;DB=0;DC=1;DD=1;DE=0;DF=1;DG=1;DP=1;} //显示 "5" 带小数点
    if(x==16){DA=1;DB=0;DC=1;DD=1;DE=1;DF=1;DG=1;DP=1;} //显示 "6" 带小数点
    if(x==17){DA=1;DB=1;DC=1;DD=0;DE=0;DF=0;DG=0;DP=1;} //显示 "7" 带小数点
    if(x==18){DA=1;DB=1;DC=1;DD=1;DE=1;DF=1;DG=1;DP=1;} //显示 "8" 带小数点
    if(x==19){DA=1;DB=1;DC=1;DD=1;DE=0;DF=1;DG=1;DP=1;} //显示 "9" 带小数点
}

```

```

main()
{
    P2M0=0x1B; //需要高电平大电流的引脚 2.4、2.3、2.1、2.0，填入数值 0001 1011
    P3M0=0xE4; //需要高电平大电流的引脚 3.7、3.6、3.5、3.2，填入数值 1110 0100

    P1ASF=0x80; //设置 P1.7 引脚为模拟功能

    while(1)
    {
        delay_n_ms(4); //系统周期 4ms

        count=count+1;
        if(count>125) //0.5 秒测量一次电压
        {
            count=0;

            ADC_CONTR=0x88+7; //开始 P1.7 引脚转换
            Delay50us(); //等待 AD 转换完成
            adc=ADC_RES; //读取转换结果

            adc=adc*2; //量程增大倍数
            adc=adc*10/51; //计算电压*10 倍，消除小数点

            Data1=0;
            Data2=adc/100; //百位数据
            Data3=adc%100/10; //十位数据
            Data3=Data3+10; //加 10，display 函数显示的数字会带小数点

            Data4=adc%10; //个位数据
        }

        //////////////////////////////////////
        //显示程序模块化，只需要改变 Data1/2/3/4 的值就可以控制显示内容

        display_time=display_time+1;
        if(display_time>3)
        {
            display_time=0; //0、1、2、3 四个循环
        }

        if(display_time==0) //第 0 次显示第一个数码管
        {

```

```

        B1=0;B2=1;B3=1;B4=1;
        display(Data1);           //显示第 1 位数据
    }

    if(display_time==1)//第 1 次显示第二个数码管
    {
        B1=1;B2=0;B3=1;B4=1;
        display(Data2);           //显示第 2 位数据
    }

    if(display_time==2)//第 2 次显示第三个数码管
    {
        B1=1;B2=1;B3=0;B4=1;
        display(Data3);           //显示第 3 位数据
    }

    if(display_time==3)//第 3 次显示第四个数码管
    {
        B1=1;B2=1;B3=1;B4=0;
        display(Data4);           //显示第 4 位数据
    }

    }
}

```

第十四课：制作舵机测试仪

积少而成多，量变成质变！

回顾已经过去的十三课，坚持下来的人又学会了多少新知识！

很快，你就会发现，你已经不知不觉中能够写程序制作出以前看起来很高大上的、“大神”们才能捣鼓东西了！

在新的一课，你还将学会怎么控制舵机、电调这些航模设备，在以后的

课程中，还将亲手做一个实用的四通道航模遥控器，想想是不是很激动人心？

最近又有不少人问我：“我没有基础/学历不高/没学过编程.....可以学会吗？”

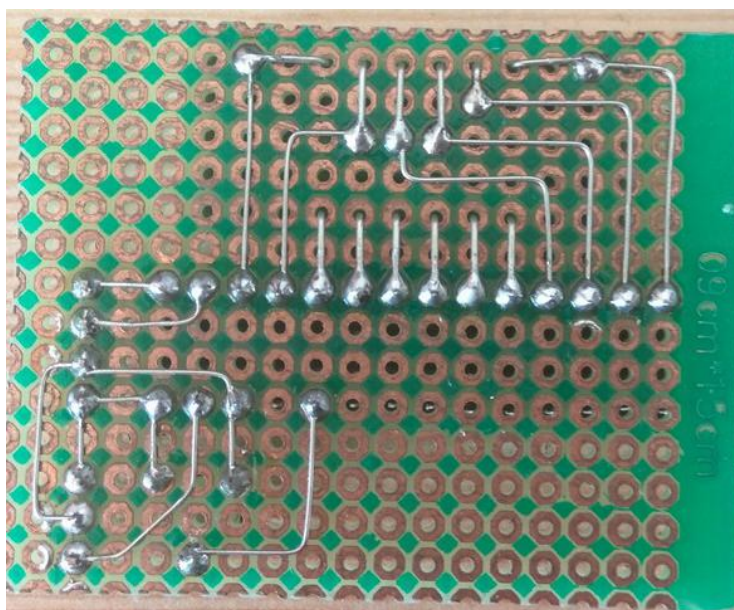
这里要强调一下，学东西切不可急功近利，已经错过课程开始的人，更要认真一步一个脚印的从第一课慢慢学，每一个练习都是要认真做的。不做练习我敢肯定还是学不会的。

当然，还是那句话，只要你会 ABCD、加减乘除，没有理由学不会。课程相对来说已经是非常简单。（有感到不懂和难的还是要多问，不问别人怎么知道）

本课的主意内容：学习用 PWM 信号控制舵机

1.1 焊接电路

电路与上一个项目《电量显示》基本相同，数码管+单片机。因此直接在原电路上扩展，增加一个 3 针的排针，焊上地、5V、信号线即可。

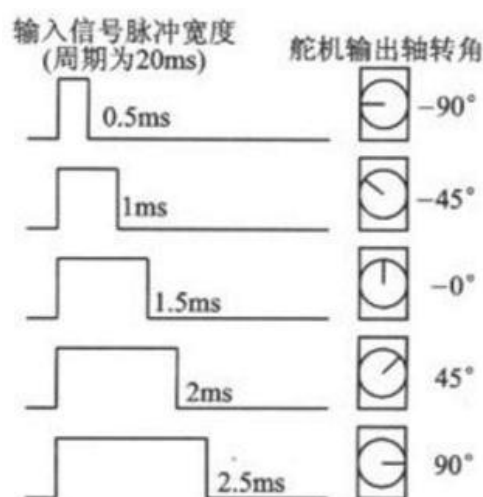


1.2 舵机控制原理

控制舵机的信号其实也是 PWM，跟我们在之前课程讲到的 PWM 原理一样。比较一下异同：

控制 LED 亮度的 PWM：周期 10ms，高电平 0~10ms 可以连续变化；

控制舵机的 PWM：周期 20ms，高电平时间 0.5ms~2.5ms 变化。



在航模舵面的实际控制中，不可能有 180 度的转动，所以通用的高电平宽度其实是 1ms~2ms。

看到这，如果你已经熟练掌握 delay 延时的方法，我想你心中已经有主意怎么实现了~~

知识点 (52) `sfr P5=0xC8;` 声明 P5 引脚

单片机常用的是 P0、P1、P2、P3 引脚，系统默认已经声明，新增的 P5 引脚需要先声明才能用

让舵机左右摆动的小程序：

```
#include<reg51.h>
```

```
sfr P5=0xC8; //声明 P5 引脚寄存器
```

```
sbit OUT=P5^5; //定义一个输出引脚
```

```

unsigned int L; //定义一个数据

void Delay10us()           //@12.000MHz
{
    unsigned char i;

    i = 27;
    while (--i);
}

void delay_n_10us(unsigned int n)//延时 n 个 10us 的函数
{
    while(n)
    {
        Delay10us();
        n=n-1;//每循环一次 n 减小 1
    }
}

void main()
{
    L=150;//1.5ms 高电平，舵机摆到中间

    while(1)
    {
        OUT=1;
        delay_n_10us(L);//输出高电平
        OUT=0;
        delay_n_10us(1900);//19ms 左右低电平

        L=L+1;//舵机反复左右摆动
        if(L>200)//如果时间大于 2ms，从 1ms 重新开始
        {
            L=100;
        }
    }
}

```

1.3 旋钮控制舵机

学到的知识更重要的是能灵活运用。在第十三课中学习的 AD 转换功能不仅可以

测量电压，还能测出电位器旋转的角度。

旋转电位器时输出的电压会改变，用 AD 转换结果会 0~255 变化，正好可以用来控制 PWM 高电平的时间。

用电位器控制舵机的程序：

```
#include<reg51.h>

sfr P5=0xC8; //声明 P5 引脚寄存器
sfr P1ASF=0x9D; //声明 P1 口模拟功能寄存器
sfr ADC_RES=0xBD; //声明 ADC 转换结果寄存器
sfr ADC_CONTR=0xBC; //声明 ADC 控制寄存器

sbit OUT=P5^5; //定义一个输出引脚

unsigned int L; //定义一个数据
unsigned int adc; //AD 转换计算

void Delay10us() // @12.000MHz
{
    unsigned char i;

    i = 27;
    while (--i);
}

void delay_n_10us(unsigned int n) //延时 n 个 10us 的函数
{
    while(n)
    {
        Delay10us();
        n=n-1; //每循环一次 n 减小 1
    }
}

void main()
{
    L=150; //1.5ms 高电平，舵机摆到中间
    P1ASF=0x80; //设置 P1.7 引脚为模拟功能

    while(1)
```



```

{
    OUT=1;
    delay_n_10us(L); //输出高电平
    OUT=0;
    delay_n_10us(1900); //19ms 左右低电平

    ADC_CONTR=0x88+7; //开始 P1.7 引脚转换
    delay_n_10us(5); //等待 50us, AD 转换完成
    adc=ADC_RES; //读取转换结果

    L=adc; //转动旋钮, adc 的值 0~255 变化
    //L 的变化范围是 100~200, 所以进行一下限制
    if(L>200)L=200; //高电平时间不能超过 2ms
    if(L<100)L=100; //高电平时间不能低于 1ms
}
}

```

1.4 组合数码管程序与舵机程序

如果只有一个旋钮, 相当于你只 DIY 了一个市面价值 6 元的简易测试仪:



但如果你加了显示屏, 那立马就上了档次, 相当于 30 元的高端测试仪了



组合不同的程序功能，在每个项目里都会练习到。现在，你应该已经学会“系统周期法”。用这个方法，你应该马上想到：

数码管程序周期：4ms；

舵机程序周期：10us；

用最小的 10us 做为新程序的周期，你需要：

每 400 次（4ms）执行一次数码管；

每 2000 次（20ms）中，前 100~200 次（1~2ms）高电平，其余低电平。

这样，很轻松的就将两个简单的程序组合成一个复杂的程序。

学会这个方法，相当于你学会了一个“套路”，大部分程序你都可以这样套。赶快自己练习试试吧。

.....

正如一道数学题有多种解法一样，“系统周期”也可以更灵活的分配。其实舵机只受高电平时间控制，至于低电平时间则并不关心。利用这个特性，可以将程序流程简化一下：

系统周期：4ms；

每 4ms 执行一次数码管；

每 20ms 执行一次输出高电平。1ms 的时间很短，对显示来说没有影响。

```
#include<reg51.h>
```

```
sfr P5=0xC8; //声明 P5 引脚寄存器
sfr P2M0=0x96; //声明 P2 引脚模式寄存器
sfr P3M0=0xb2; //声明 P3 引脚模式寄存器
sfr P1ASF=0x9D; //声明 P1 口模拟功能寄存器
sfr ADC_RES=0xBD; //声明 ADC 转换结果寄存器
sfr ADC_CONTR=0xBC; //声明 ADC 控制寄存器
```

```
sbit DA=P2^4; //数码管每个引脚
```

```

sbit DB=P3^2;
sbit DC=P3^6;
sbit DD=P2^0;
sbit DE=P2^1;
sbit DF=P2^3;
sbit DG=P3^5;
sbit DP=P3^7;

sbit B1=P2^5;           //四个数码管的引脚定义
sbit B2=P2^2;
sbit B3=P3^3;
sbit B4=P3^4;

sbit OUT=P5^5; //定义一个输出引脚

unsigned char display_time; //定义变量用于显示次数

unsigned char Data1; //第一个数码管要显示的数据
unsigned char Data2; //第二个数码管要显示的数据
unsigned char Data3; //第三个数码管要显示的数据
unsigned char Data4; //第四个数码管要显示的数据

unsigned char count; //计时
unsigned int adc; //AD 转换计算
unsigned int L; //高电平时间

void Delay10us() // @12.000MHz
{
    unsigned char i;

    i = 27;
    while (--i);
}

void delay_n_10us(unsigned int n) //延时 n 个 10us 的函数
{
    while(n)
    {
        Delay10us();
        n=n-1; //每循环一次 n 减小 1
    }
}

```

```
void display(unsigned char x)//控制数码管显示内容的函数
```

```
{
    //判断 x 的值来决定显示什么
    if(x==0){DA=1;DB=1;DC=1;DD=1;DE=1;DF=1;DG=0;DP=0;} //显示 "0"
    if(x==1){DA=0;DB=1;DC=1;DD=0;DE=0;DF=0;DG=0;DP=0;} //显示 "1"
    if(x==2){DA=1;DB=1;DC=0;DD=1;DE=1;DF=0;DG=1;DP=0;} //显示 "2"
    if(x==3){DA=1;DB=1;DC=1;DD=1;DE=0;DF=0;DG=1;DP=0;} //显示 "3"
    if(x==4){DA=0;DB=1;DC=1;DD=0;DE=0;DF=1;DG=1;DP=0;} //显示 "4"
    if(x==5){DA=1;DB=0;DC=1;DD=1;DE=0;DF=1;DG=1;DP=0;} //显示 "5"
    if(x==6){DA=1;DB=0;DC=1;DD=1;DE=1;DF=1;DG=1;DP=0;} //显示 "6"
    if(x==7){DA=1;DB=1;DC=1;DD=0;DE=0;DF=0;DG=0;DP=0;} //显示 "7"
    if(x==8){DA=1;DB=1;DC=1;DD=1;DE=1;DF=1;DG=1;DP=0;} //显示 "8"
    if(x==9){DA=1;DB=1;DC=1;DD=1;DE=0;DF=1;DG=1;DP=0;} //显示 "9"

    if(x==10){DA=1;DB=1;DC=1;DD=1;DE=1;DF=1;DG=0;DP=1;} //显示 "0" 带小数点
    if(x==11){DA=0;DB=1;DC=1;DD=0;DE=0;DF=0;DG=0;DP=1;} //显示 "1" 带小数点
    if(x==12){DA=1;DB=1;DC=0;DD=1;DE=1;DF=0;DG=1;DP=1;} //显示 "2" 带小数点
    if(x==13){DA=1;DB=1;DC=1;DD=1;DE=0;DF=0;DG=1;DP=1;} //显示 "3" 带小数点
    if(x==14){DA=0;DB=1;DC=1;DD=0;DE=0;DF=1;DG=1;DP=1;} //显示 "4" 带小数点
    if(x==15){DA=1;DB=0;DC=1;DD=1;DE=0;DF=1;DG=1;DP=1;} //显示 "5" 带小数点
    if(x==16){DA=1;DB=0;DC=1;DD=1;DE=1;DF=1;DG=1;DP=1;} //显示 "6" 带小数点
    if(x==17){DA=1;DB=1;DC=1;DD=0;DE=0;DF=0;DG=0;DP=1;} //显示 "7" 带小数点
    if(x==18){DA=1;DB=1;DC=1;DD=1;DE=1;DF=1;DG=1;DP=1;} //显示 "8" 带小数点
    if(x==19){DA=1;DB=1;DC=1;DD=1;DE=0;DF=1;DG=1;DP=1;} //显示 "9" 带小数点
}
```

```
void main()
```

```
{
    P2M0=0x1B; //需要高电平大电流的引脚 2.4、2.3、2.1、2.0，填入数值 0001 1011
    P3M0=0xE4; //需要高电平大电流的引脚 3.7、3.6、3.5、3.2，填入数值 1110 0100
    P1ASF=0x80; //设置 P1.7 引脚为模拟功能

    while(1)
    {
        delay_n_10us(400);//系统周期 4ms

        count=count+1;
        if(count>4) //每 20ms 秒插入一次高电平
        {
            count=0;
            B1=1;B2=1;B3=1;B4=1;//暂时关闭数码管，避免闪烁

            ADC_CONTR=0x88+7; //开始 P1.7 引脚转换
        }
    }
}
```

```

delay_n_10us(5); //等待 50us,AD 转换完成
adc=ADC_RES; //读取转换结果

L=adc; //转动旋钮, adc 的值 0~255 变化
//L 的变化范围是 100~200, 所以进行一下限制
if(L>200)L=200; //高电平时间不能超过 2ms
if(L<100)L=100; //高电平时间不能低于 1ms

OUT=1;
delay_n_10us(L); //输出 1~2ms 高电平
OUT=0; //其他时间为低电平

//显示 L 的大小
Data1=0;
Data2=L/100; //百位数据
Data2=Data2+10; //加 10, display 函数显示的数字会带小数点

Data3=L%100/10; //十位数据
Data4=L%10; //个位数据
}

////////////////////////////////////
//显示程序模块化, 只需要改变 Data1/2/3/4 的值就可以控制显示内容

display_time=display_time+1;
if(display_time>3)
{
    display_time=0; //0、1、2、3 四个循环
}

if(display_time==0)//第 0 次显示第一个数码管
{
    B1=0;B2=1;B3=1;B4=1;
    display(Data1); //显示第 1 位数据
}

if(display_time==1)//第 1 次显示第二个数码管
{
    B1=1;B2=0;B3=1;B4=1;
    display(Data2); //显示第 2 位数据
}

```

```

        if(display_time==2)//第 2 次显示第三个数码管
        {
            B1=1;B2=1;B3=0;B4=1;
            display(Data3);           //显示第 3 位数据
        }

        if(display_time==3)//第 3 次显示第四个数码管
        {
            B1=1;B2=1;B3=1;B4=0;
            display(Data4);           //显示第 4 位数据
        }
    }
}

```

1.5 项目扩展

作为一个实用的项目，自然不能随便做一做就行。在这个项目中，还存在一个问题：输出的高电平时间不是准确的 1ms，而是 1.1ms。

关于精确延时，在《红外遥控》章节中有过讲述。当延时时间达到微秒级时，就会不太准确，需要手动调整。

由于循环程序也会占用时间，明显执行一次 delay10us 不是 10us，而是 11us。

当然，精确调整需要示波器、逻辑分析仪才行，普通学习者没有条件，这里给出一个调整好的：

```

void Delay10us()           //@12.000MHz
{
    unsigned char i;

    _nop_();

    _nop_();//调整过
}

```

```
    i = 24;

    while (--i);
}

void delay_n_10us(unsigned int n)//延时 n 个 10us 的函数
{
    while(n) //while 会占用时间，所以调小 delay10us
    {
        Delay10us();

        n=n-1;//每循环一次 n 减小 1
    }
}
```

除了使信号输出更精确，作为一款有“显示屏”的产品，搭配上按键，你就可以切换多种模式，实现丰富的功能！这样它就不单单是一个舵机测试仪了，而是一个多功能的小助手。

在你学会了更多知识以后，它可以变成电压电流表、转速表、温度湿度表、电机调速器等等。

现在，你就可以先试着增加一个按键，集成电压测量的功能？

第十五课：制作电子开关

有细心的童鞋就会发现，现在进行的单片机课堂，其实就是将楼主之前发的所有 DIY 教程按难易程度串起来再讲一遍。

不同在于，原来你只能照猫画虎，并不知道原理，

而现在你却能一程序一程序亲手自己写出一模一样的功能！

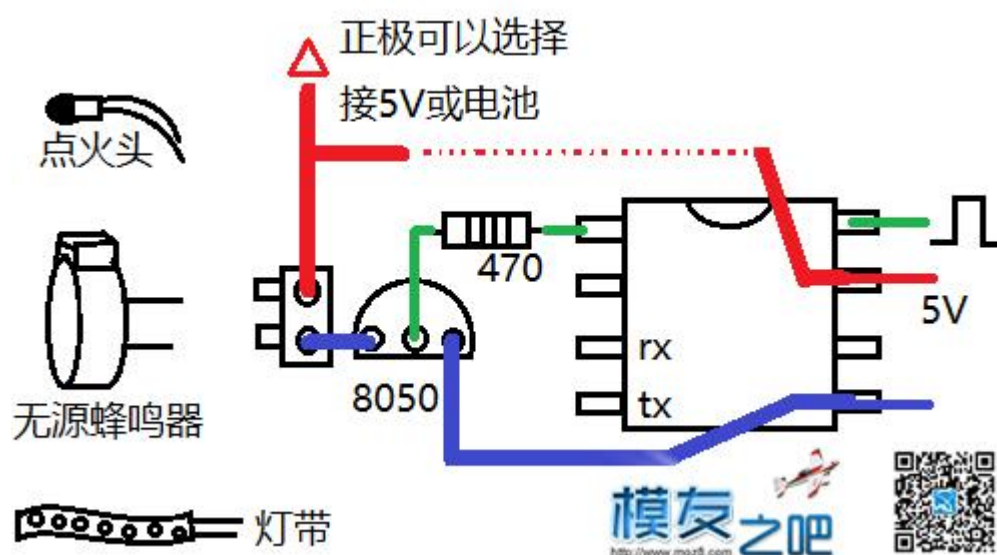
本课的主要内容：制作一个电子开关

1.1 材料准备

- 1、STC15F104W（5V）或者 STC15W104（3V~5V）
- 2、SS8050 三极管
- 3、470 欧电阻

1.2 焊接

焊接制作过程与楼主以前的电子开关教程一样：



第十六课：扩展课！万能的电子开关

航模用的电子开关主要用途就是检测接收机的 PWM 信号，控制其他设备的供电。因此，它的用途实际上非常广泛：控制灯带、控制点火器、控制喇叭.....

而实际上，它的程序非常简单！

马上你就会恍然大悟，原来只需要几行代码，就可以实现曾经遥不可及的功能。

随着你们学习的越来越多，其实现在已经有能力去独立做一些简单的小东西了。

本次课程完全没有涉及到任何新知识，都是过去知识点的组合。

本课的主要内容：自主发挥，项目练手

1.1 测量 PWM 信号

前面学到，舵机测试仪控制舵机的信号，是用 PWM 高电平时间的长度，1ms~2ms；航模接收机的输出信号，也是一样的。

电子开关的原理，就是把自己“伪装”成一个“舵机”，如果收到 1ms 的高电平信号，就关闭开关；如果收到 2ms 的高电平信号，就打开开关。

现在如果还没有航模接收机，可以先用舵机测试仪代替。

如何测量高电平的时间？你是不是有些似曾相识。对的，在《红外接收机》课程中，已经学过了检测低电平时间。

只需要把红外接收的程序稍微修改下，就是一个万能的电子开关！

检测高电平时间：

```
while(IN==1) //如果引脚一直为高电平，一直循环
```

```

{

    Delay10us();

    time=time+1; //每循环一次，就记录了 10us 时间

}

```

判断高电平时间：

if(time>90&&time<120)OUT=0;//允许一定的误差 ,0.9ms~1.2ms 高电平信号都可以关闭开关

if(time>180&&time<210)OUT=1;//允许一定的误差，1.8ms~2.1ms 高电平信号都可以打开开关

简单例程：

```

#include<reg51.h>
#include<intrins.h>

sfr P3M0=0xb2;    //声明 引脚输出模式寄存器

sbit IN=P3^4;    //定义 PWM 输入引脚
sbit OUT=P3^3;    //定义输出引脚

unsigned int time;    //记录时间

void Delay10us()    //@12.000MHz
{
    unsigned char i;

    _nop_();//使用《舵机测试仪》课程中调整过的延时程序
    _nop_();
    i = 24;
    while (--i);
}

```

```

main()
{
    OUT=0;          //先关闭输出
    P3M0=8;         //P3.3 引脚大电流模式

    while(1)
    {

        if(IN==1)    //收到 PWM 高电平,开始测量
        {
            time=0; //先把时间清零
            while(IN==1) //如果引脚一直为高电平，一直循环
            {
                Delay10us();
                time=time+1; //每循环一次，就记录了 10us 时间
            }
            //引脚变成低电平，循环结束

            if(time>90&&time<120)//0.9ms~1.2ms 高电平
            {
                OUT=0; //关闭开关
            }

            if(time>180&&time<210)//1.8ms~2.1ms 高电平
            {
                OUT=1; //打开开关
            }
        }

    }
}

```

1.2 项目扩展

电子开关的强大就在于它灵活的扩展性：

扩展方向 1：夜航灯控

不仅能控制灯带的亮灭，通过编程，还能够实现多种模式：开关变化一次，

灯带切换一个模式，如常亮、爆闪、双闪……等；

扩展方向 2：点火器

开关动作后，点火两秒钟自动关闭；

扩展方向 3：左右转向灯

1.5ms 高电平舵机在中间，小于 1.4 说明左转弯亮左灯，大于 1.6 说明右转弯亮右灯；

扩展方向 4：多路开关

控制四路，开关变化一次切换一路开关；

5：.....

6：.....

各种各样电子开关能实现的功能，只有你想不到，没有做不到。学会单片机，有什么需求直接自己动手实现！

在这里，给出一个实现一路灯控的例程，更多的功能，就要靠你自己的发挥练习了：

顺便学点小知识

知识点（53） #define 声明替换

写程序中我们经常遇到要写“unsigned char”这么一长串英文，有没有什么办法能够更简便点？在 C 程序中，还提供了一种替换机制：

#define A B //如果声明了这句话，那么以后在程序中写 A，编译时会自动被系统替换成 B。

利用这个机制，可以简写，比如：

```
#define u8 unsigned char //“unsigned char”是 0~255 的 8 位二进制数，我们以后可以简写为“u8”
#define u16 unsigned int //“unsigned int”是 0~65535 的 16 位二进制数，我们以后可以简写为“u16”
```

知识点（54） n++;自己加一

同样是为了简写程序，以后遇到 “n=n+1;” 这样自己增加 1 的，可以用两个加号代替：“n++;”

同样的，“n=n-1;” 也可以简写为 “n--;”

```
#include<reg51.h>

#include<intrins.h> //使用 Nop ( ) 需要先声明

#define u8 unsigned char //“unsigned char” 是 0~255 的 8 位二进制数，我们以后可以简写为 “u8”
#define u16 unsigned int //“unsigned int” 是 0~65535 的 16 位二进制数 我们以后可以简写为 “u16”

sfr P3M0=0xb2; //声明 引脚输出模式寄存器

sbit IN=P3^4; //定义 PWM 输入引脚
sbit OUT=P3^3; //定义输出引脚

u16 time; //记录时间
u16 delay_time; //记录时间

u8 Key1; //把高电平时间变化当做学过的按键处理
u8 Key1_old; //记录 Key1 过去出现低电平

u8 mode; //模式
u8 step; //步骤

void Delay10us() // @12.000MHz
```

```

{

    unsigned char i;

    _nop_();//使用《舵机测试仪》课程中调整过的延时程序

    _nop_();

    i = 24;

    while (--i);

}

```

```

main()

{

    OUT=0;          //先关闭输出

    P3M0=8;         //P3.3 引脚大电流模式

    Key1_old=1;

    while(1)

    {

        Delay10us();//系统周期 10us

        //模式处理与《七色光芒》几乎相同

        if(mode==0)//模式 0 , 关闭

        {

            OUT=0;

        }

    }

}

```

```
if(mode==1)//模式 1 , 打开
```

```
{
```

```
    OUT=1;
```

```
}
```

```
if(mode==2)//模式 2 , 闪烁
```

```
{
```

```
    delay_time++;
```

```
    if(delay_time>20000)//每过 0.2 秒
```

```
    {
```

```
        delay_time=0;
```

```
        if(OUT==1)OUT=0;//变化一次
```

```
        else OUT=1;
```

```
    }
```

```
}
```

```
if(mode==3)//模式 3 , 双闪
```

```
{
```

```
    delay_time++;
```

```
    if(delay_time>20000)//每过 0.2 秒
```

```
    {
```

```
        delay_time=0;
```

```
        step++;
```

```
        if(step>5)step=0;//双闪变化较多 , 分为 6 个步骤
```

```
}

if(step==0)//步骤 0 , 亮
{
    OUT=1;
}

if(step==1)//步骤 1 , 灭
{
    OUT=0;
}

if(step==2)//步骤 2 , 亮
{
    OUT=1;
}

if(step==3)//步骤 3 , 灭
{
    OUT=0;
}

if(step==4)//步骤 4 , 灭
{
    OUT=0;
}

if(step==5)//步骤 5 , 灭
{
    OUT=0;
```



```

    }

}

if(IN==1)      //收到 PWM 高电平,开始测量
{
    time=0; //先把时间清零

    while(IN==1) //如果引脚一直为高电平，一直循环
    {
        Delay10us();

        time=time+1; //每循环一次，就记录了 10us 时间
    }

    //引脚变成低电平，循环结束

    if(time>90&&time<120)//0.9ms~1.2ms 高电平
    {
        Key1=1; //相当于按键还没按下
    }

    if(time>180&&time<210)//1.8ms~2.1ms 高电平
    {
        Key1=0; //相当于按键按下
    }

    if(Key1==1&&Key1_old==0)//如果本次是 1 并且上次是 0,检测到 0~1 的变
    {

```

化

```

        mode++;

        if(mode>3)mode=0;//检测到信号的变化，切换模式
    }

    Key1_old=Key1;    //保存

}

}

}

```

第十七课：电子开关的华丽转身——有刷电调

可能快过年了，大家都忙了，也可能随着课程逐渐变难，能跟上的越来越少了。

但是为了星辰大海，我们的课程还是要继续下去的！

对以前的课程有感到不懂的，还是可以随时提问。

在扩展电子开关功能的时候，有没有想过用开关控制电机呢？如果还能结合 PWM 控制电机的转速.....等等，能控制电机转速的电子开关还叫开关吗，应该改名叫有刷电调了！如果控制转速的同时还能控制正反转，那就是双向有刷电调了！相似的电路，不一样的程序却有着截然不同的

功能，这就是电子开关的华丽转身。

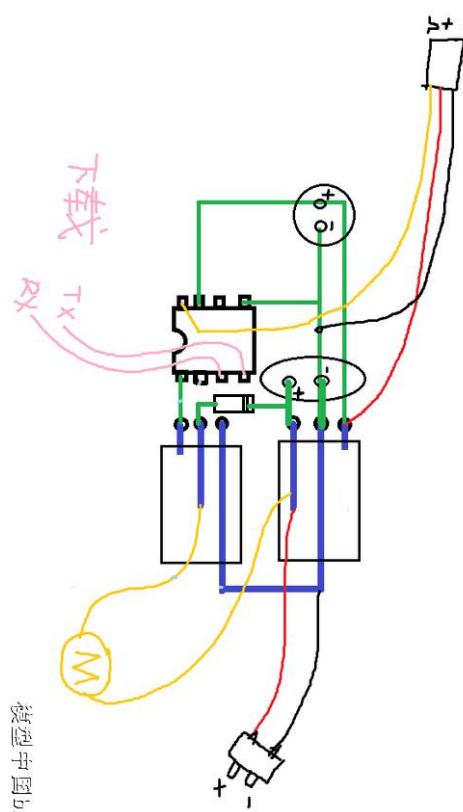
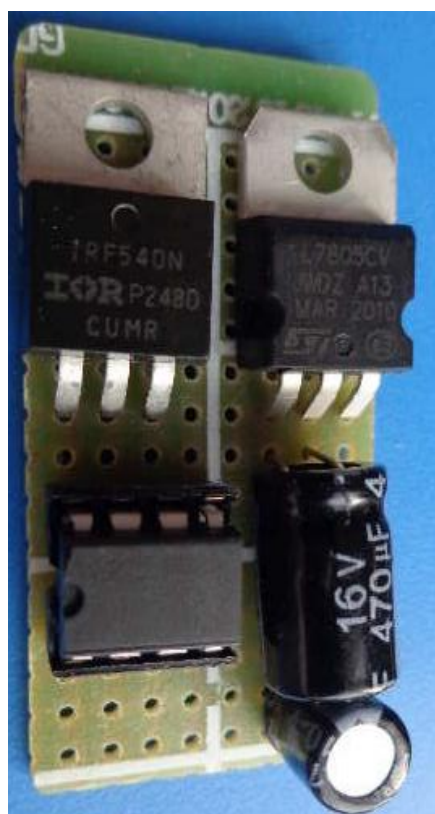
本课的主要内容：制作一个单向有刷电调

1.1 材料准备

- 1、STC15W104 或 STC15F104W 单片机
- 2、7805 5V 稳压
- 3、IRF3205 N 沟道 MOS 管
- 4、SS34 二极管
- 5、470UF 电容 16V
- 6、线材、3P 杜邦头

1.2 焊接制作

由于现在很忙，所以制作过程就直接用以前的教程先充数了.....以后会补上



模型中国站

第十八课：小弟出马——定时器

在单片机中，除了有 CPU，还集成了很多功能丰富的模块，定时器就是其中之一。

如果说 CPU 是单片机中的老大，那么定时器就是小弟。在前面的课程中，我们主要学习简单好理解的“系统周期法”。周期法就是所有的活都是老大一个人干，小弟们全歇着。

周期法虽然简单，但是时间精度低，要想高精度的测量高电平时间，就要定时器小弟来出马。

本课的主要内容：学习定时器

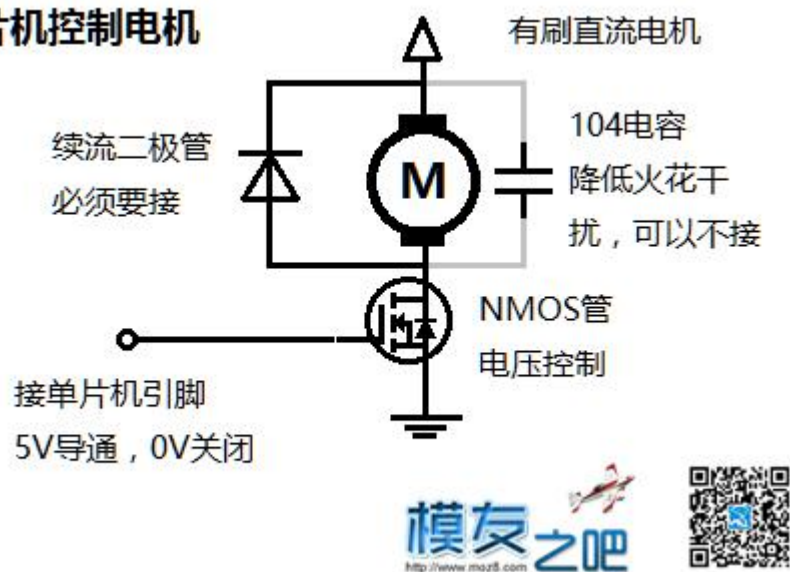
1.1 系统周期法实现简易电调

周期法虽然时间精度低，却是最万金油的程序写法，无论你的程序有多复杂，都可以拆分成一个一个小片来执行。我们可以先用周期法来实现简单电调的功能，方便以后使用定时器时的理解。

电调的主要功能就是产生 PWM，控制电机转速，和《七色光芒》中的 PWM 控制 LED 亮度原理是一样的。把《电子开关》和《七色光芒》两个程序组合起来，就是一个简易电调。

知识点 (55) MOS 管控制直流电机

单片机控制电机



控制 N 沟道 MOS 管与控制 8050 三极管类似，都是高电平打开。区别是 MOS 管是电压控制，不需要接电阻了。

系统周期：10us

PWM 周期：1ms，1000Hz

PWM 精度：100 级，步进 10us，步进 100 次为一个周期 1ms

根据以上参数写一个简易电调的程序：

