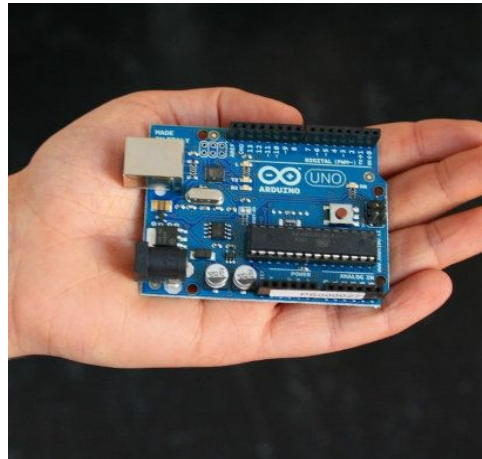


## 例程1、Hello World!

首先先来练习一个不需要其他辅助元件，只需要一块Arduino 和一根下载线的简单实验，让我们的Arduino 说出“Hello World! ”，这是一个让Arduino 和PC 机通信的实验，这也是一个入门试验，希望可以带领大家进入Arduino 的世界。

这个实验我们需要用到的实验硬件有：



Arduino 控制器



### USB 下载线

我们按照上面所讲的将Arduino 的驱动安装好后，我们打开Arduino 的软件，编写一段程序让Arduino 接受到我们发的指令就显示“Hello World! ”字符串，当然您也可以让Arduino 不用接受任何指令就直接不断回显“Hello World! ”，其实很简单，一条if ( ) 语句就可以让你的Arduino 听从你的指令了，我们再借用一下Arduino 自带的数字13口LED，让Arduino 接受到指令时LED 闪烁一下，再显示“Hello World! ”

**下面给大家一段参考程序。**

```
int val;//定义变量val
int ledpin=13;//定义数字接口13
void setup()
{
  Serial.begin(9600);//设置波特率为9600,这里要跟软件设置相一致。当接入特定设备(如:
  蓝牙)时,我们也要跟其他设备的波特率达到一致。
  pinMode(ledpin, OUTPUT);//设置数字13 口为输出接口,Arduino 上我们用到的I/O 口都要
  进行类似这样的定义。
}
void loop()
{
  val=Serial.read();//读取PC 机发送给Arduino 的指令或字符,并将该指令或字符赋给val
```

```
if(val=='R')//判断接收到的指令或字符是否是“R”。  
{//如果接收到的是“R”字符  
digitalWrite(ledpin,HIGH);//点亮数字13 口LED。  
delay(500);  
digitalWrite(ledpin,LOW);//熄灭数字13 口LED  
delay(500);  
Serial.println("Hello World!");//显示“Hello World!”字符串  
}  
}
```





## 例程2、LED 闪烁实验

LED 小灯实验是比较基础的实验之一，上一个“Hello World”实验里已经利用到了Arduino自带的LED，这次我们利用其他I/O口和外接直插LED灯来完成这个实验，我们需要的实验器材除了每个实验都必须的Arduino控制器和USB下载线以外的其它器件如下：

红色M5 直插LED\*1

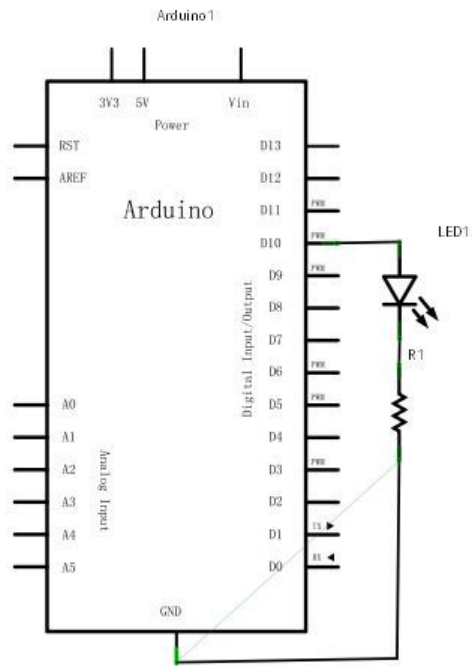
220  $\Omega$  直插电阻\*1

面包板\*1

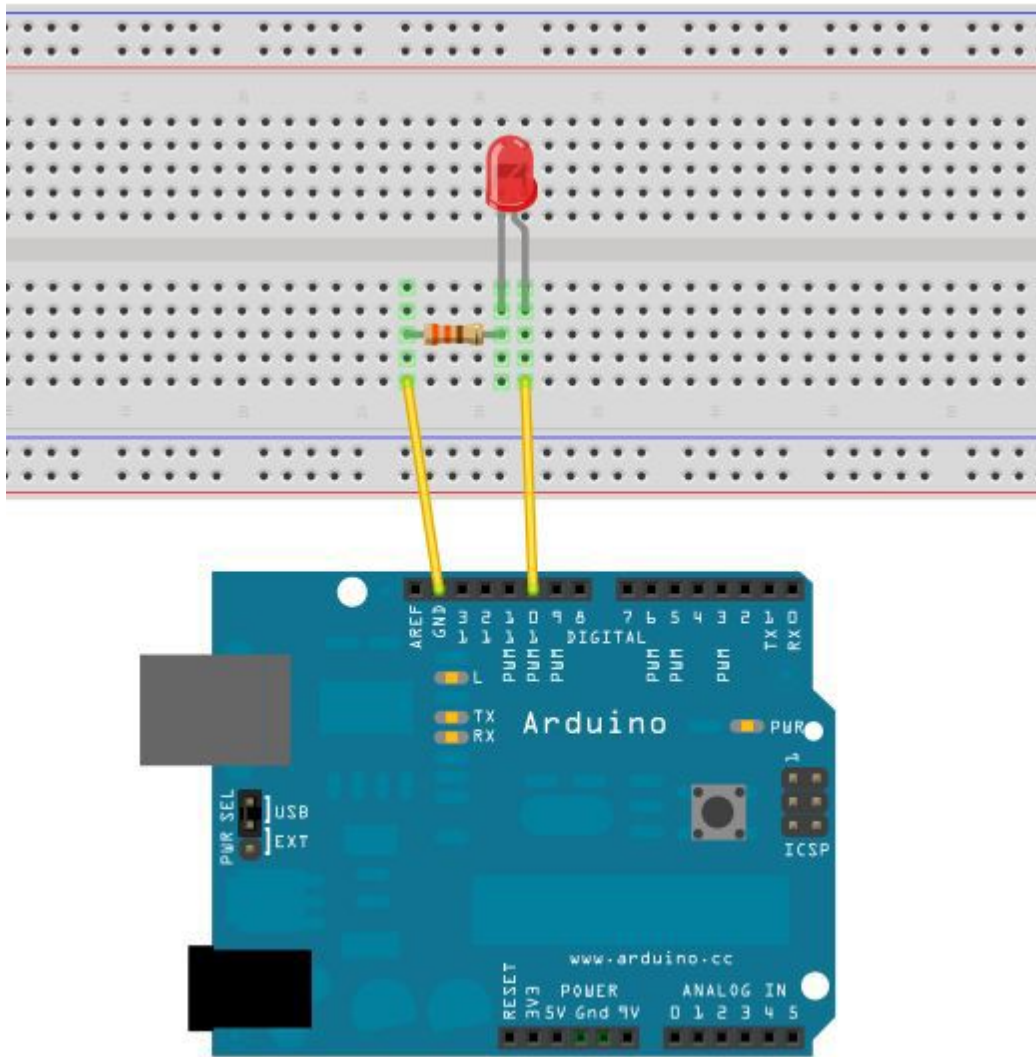
面包板跳线\*1 扎

下一步我们按照下面的小灯实验原理图链接实物图，这里我们使用数字10接口。使用发光二极管LED时，要连接限流电阻，这里为220  $\Omega$  电阻，否则电流过大会烧毁发光二极管。

小灯实验原理图



实物图



按照上图链接好电路后，就可以开始编写程序了，我们还是让LED 小灯闪烁，点亮1 秒熄灭1 秒。这个程序很简单与Arduino 自带的例程里的Blink 相似只是将13 数字接口换做10 数字接口。

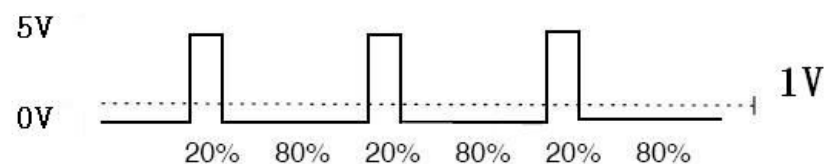
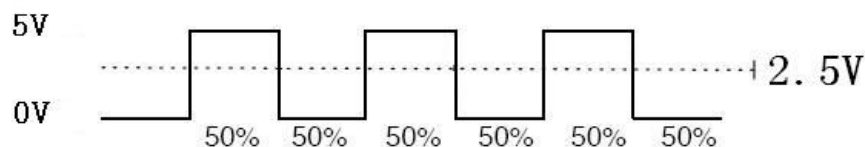
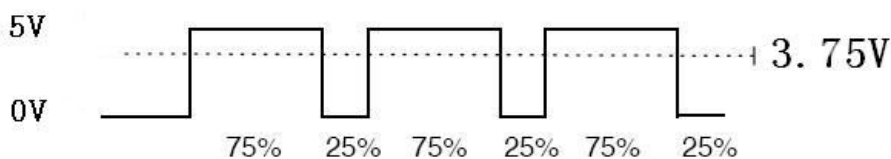
参考程序如下：

```
int ledPin = 10; //定义数字10 接口
void setup()
{
  pinMode(ledPin, OUTPUT); //定义小灯接口为输出接口
}
void loop()
{
  digitalWrite(ledPin, HIGH); //点亮小灯
  delay(1000); //延时1 秒
  digitalWrite(ledPin, LOW); //熄灭小灯
  delay(1000); // 延时1 秒
}
```

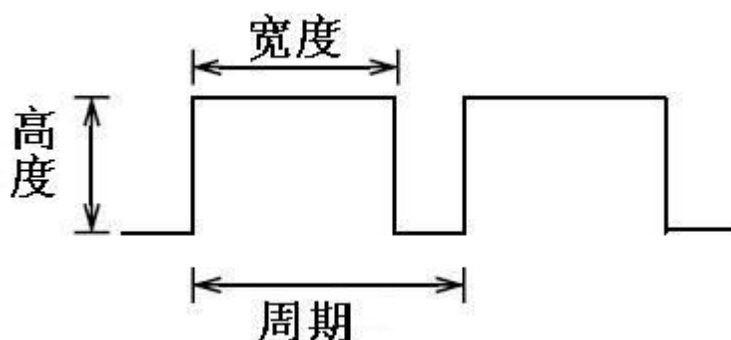
下载完程序就可以看到我们的10 口外接小灯在闪烁了，这样我们的小灯闪烁实验就完成了。

### 例程3、PWM 调控灯光亮度实验

Pulse Width Modulation 就是通常所说的PWM，译为脉冲宽度调制，简称脉宽调制。脉冲宽度调制（PWM）是一种对模拟信号电平进行数字编码的方法，由于计算机不能输出模拟电压，只能输出0 或5V 的数字电压值，我们就通过使用高分辨率计数器，利用方波的占空比被调制的方法来对一个具体模拟信号的电平进行编码。PWM 信号仍然是数字的，因为在给定的任何时刻，满幅值的直流供电要么是5V (ON)，要么是0V (OFF)。电压或电流源是以一种通(ON)或断(OFF)的重复脉冲序列被加到模拟负载上去的。通的时候即是直流供电被加到负载上的时候，断的时候即是供电被断开的时候。只要带宽足够，任何模拟值都可以使用PWM 进行编码。输出的电压值是通过通和断的时间进行计算的。输出电压=（接通时间/脉冲时间）\*最大电压值



PWM 被用在许多地方，调光灯具、电机调速、声音的制作等等。  
下面介绍一下PWM 的三个基本参数：



- 1、脉冲宽度变化幅度（最小值/最大值）
- 2、脉冲周期（1 秒内脉冲频率个数的倒数）
- 3、电压高度（例如：0V-5V）

Arduino 控制器有6 个PWM 接口分别是数字接口3、5、6、9、10、11，前面我们已经做了按

键控制小灯的实验，那是数字信号控制数字接口的实验，我们也做过电位计的实验，这次我们就来完成一个用电位计控制小灯的实验。

需要的元器件有：

电位计模块\*1

红色M5 直插LED\*1

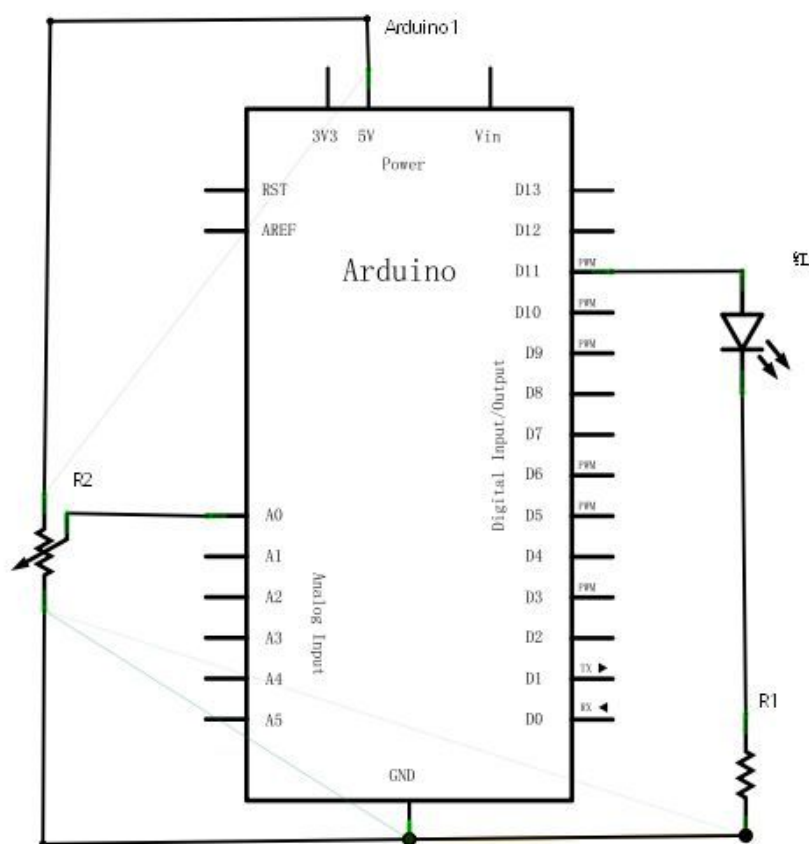
220  $\Omega$  直插电阻

面包板\*1

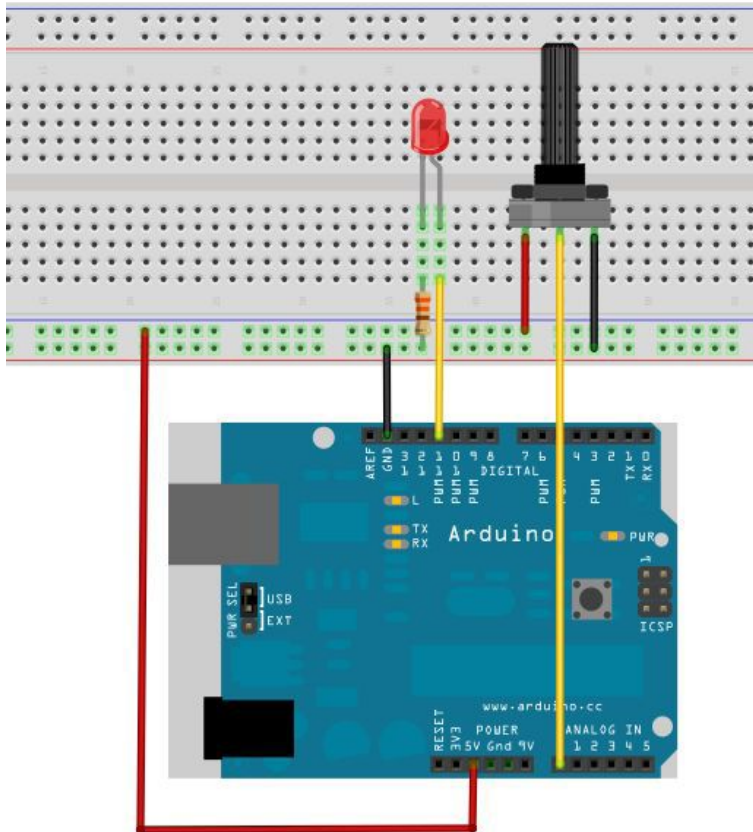
面包板跳线\*1 扎

电位计即为模拟值输入我们接到模拟口，小灯我们接到PWM 接口上，这样通过产生不同的PWM 信号就可以让小灯有亮度不同的变化。

我们先按照下面的原理图连接实物图。







在编写程序的过程中，我们会用到模拟写入`analogWrite(PWM 接口, 模拟值)`函数，对于模拟写入`analogWrite()`函数，此函数用法也很简单，我们在本实验中读取电位计的模拟值信号并将其赋给PWM 接口使小灯产生相应的亮度变化，再在屏幕上显示出读取的模拟值，大家可以理解为此程序是在模拟值读取的实验程序中多加了将模拟值赋给PWM 接口这一部分，下面给大家提供一段参考源程序。

参考源程序：

```
int potpin=0;//定义模拟接口0
int ledpin=11;//定义数字接口11（PWM 输出）
int val=0;// 暂存来自传感器的变量数值
void setup()
{
  pinMode(ledpin, OUTPUT);//定义数字接口11 为输出
  Serial.begin(9600);//设置波特率为9600
  //注意：模拟接口自动设置为输入
}
void loop()
{
  val=analogRead(potpin);// 读取传感器的模拟值并赋值给val
  Serial.println(val);//显示val 变量
  analogWrite(ledpin, val/4);// 打开LED 并设置亮度（PWM 输出最大值255）
  delay(10);//延时0.01 秒
}
```





下载完程序，我们旋转电位计的旋钮不但可以看到屏幕上数值的变化还可以清楚的看到我们面包板上的LED 小灯的亮度也在随之变化。

## 例程4广告灯效果实验

### 1) 实验器件

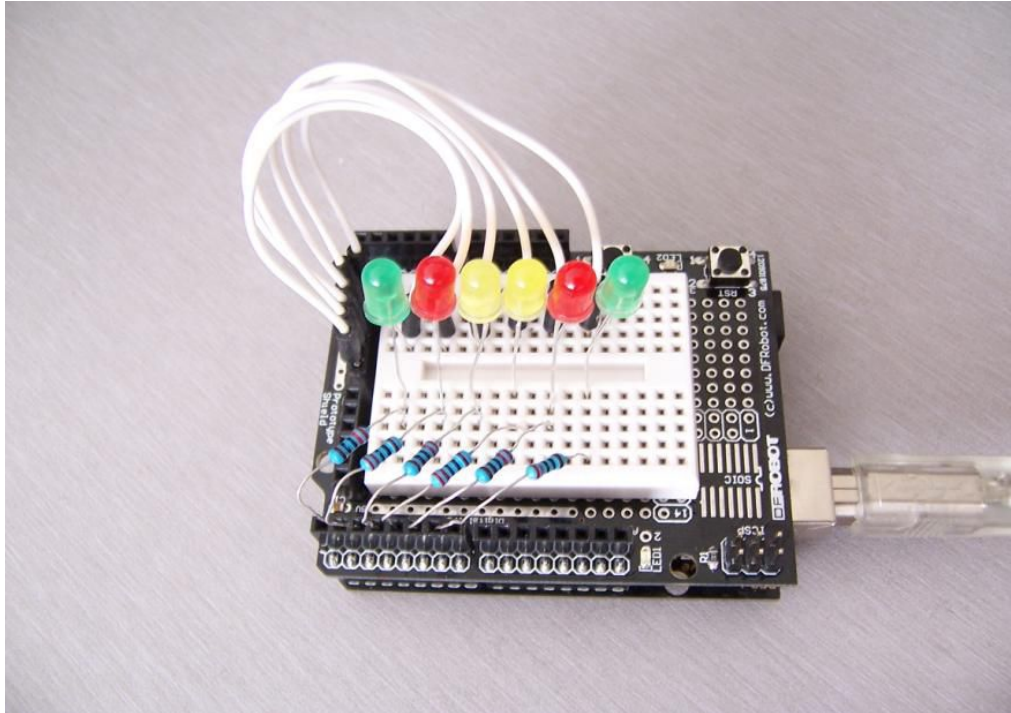
Led灯：6个

220  $\Omega$  的电阻：6个

多彩面包板实验跳线：若干

### 2) 实验连线

按照二级管的接线方法，将六个LED灯依次接到数字1~6引脚上。如图：  
广告灯实验的接线



### 3) 实验原理

在生活中我们经常会看到一些由各种颜色的led灯组成的广告牌, 广告牌上各个位置上led灯不断的变话, 形成各种效果。本节实验就是利用led灯编程模拟广告灯效果。

程式参考:

```
int BASE = 2 ;    //第一顆 LED 接的 I/O 腳
int NUM = 6;      //LED 的總數
```

```
void setup()
{
    for (int i = BASE; i < BASE + NUM; i++)
    {
        pinMode(i, OUTPUT);    //設定數字I/O腳為輸出
    }
}
```

```
void loop()
{
    for (int i = BASE; i < BASE + NUM; i++)
    {
        digitalWrite(i, LOW);    //設定數字I/O腳輸出為“低”，即逐漸關燈
        delay(200);              //延遲
    }
    for (int i = BASE; i < BASE + NUM; i++)
    {
        digitalWrite(i, HIGH);   //設定數字I/O腳輸出為“高”，即逐漸開燈
    }
}
```

```

        delay(200);          //延迟
    }
}

```

## 例程5. 交通灯设计实验

上面我们已经完成了单个小灯的控制实验，接下来我们就来做一個稍微复杂一点的交通灯实验，其实聪明的朋友们可以看出来这个实验就是将上面单个小灯的实验扩展成3 个颜色的小灯，就可以实现我们模拟交通灯的实验了。我们完成这个实验所需的元件除了Arduino 控制器和下载线还需要的硬件如下：

红色M5 直插LED\*1

黄色M5 直插LED\*1

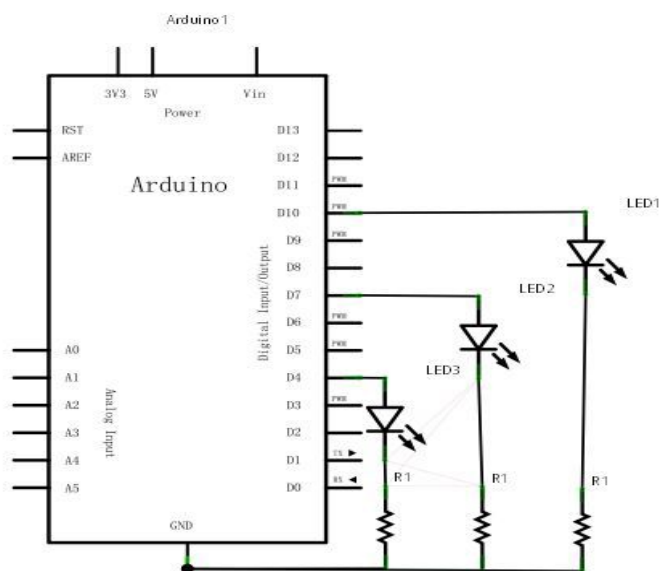
绿色M5 直插LED\*1

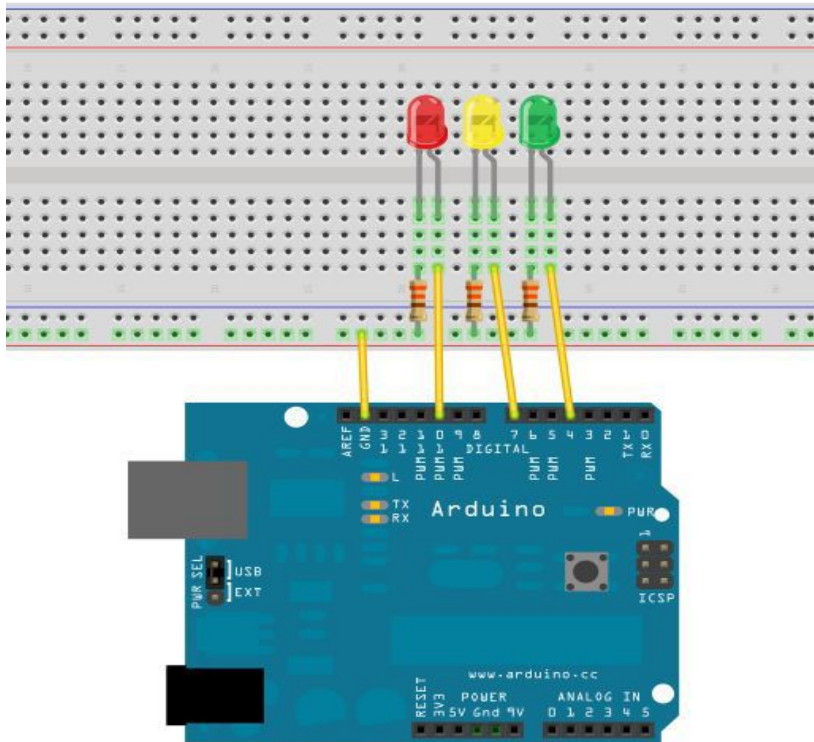
220  $\Omega$  电阻\*3

面包板\*1

面包板跳线\*1 扎

准备好上述元件我们就可以开工了，我们可以按照上面小灯闪烁的实验举一反三，下面是我们提供参考的原理图，我们使用的分别是数字10、7、4、接口。

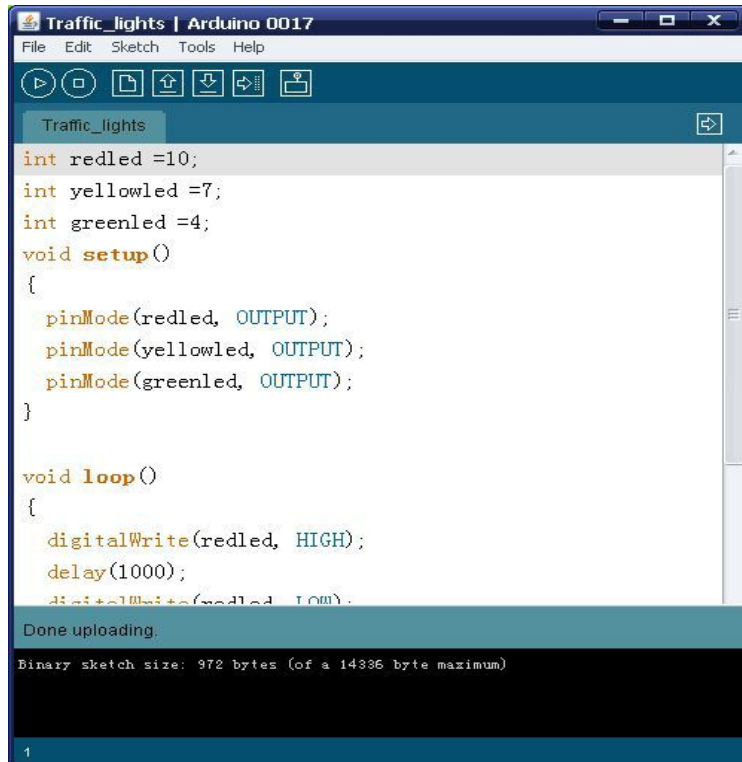




既然是交通灯模拟实验，红黄绿三色小灯闪烁时间就要模拟真实的交通灯，我们使用Arduino的delay（）函数来控制延时时间，相对于C 语言就要简单许多了。

下面是一段参考程序：

```
int redled =10; //定义数字10 接口
int yellowled =7; //定义数字7 接口
int greenled =4; //定义数字4 接口
void setup()
{
  pinMode(redled, OUTPUT); //定义红色小灯接口为输出接口
  pinMode(yellowled, OUTPUT); //定义黄色小灯接口为输出接口
  pinMode(greenled, OUTPUT); //定义绿色小灯接口为输出接口
}
void loop()
{
  digitalWrite(redled, HIGH); //点亮红色小灯
  delay(1000); //延时1 秒
  digitalWrite(redled, LOW); //熄灭红色小灯
  digitalWrite(yellowled, HIGH); //点亮黄色小灯
  delay(200); //延时0.2 秒
  digitalWrite(yellowled, LOW); //熄灭黄色小灯
  digitalWrite(greenled, HIGH); //点亮绿色小灯
  delay(1000); //延时1 秒
  digitalWrite(greenled, LOW); //熄灭绿色小灯
}
```



下载程序完成后就可以看到我们自己设计控制的交通灯了。

## 例程6按键控制LED实验

I/O 口的意思即为INPUT 接口和OUTPUT 接口，到目前为止我们设计的小灯实验都还只是应用到Arduino 的I/O 口的输出功能，这个实验我们来尝试一下使用Arduino的I/O 口的输入功能即为读取外接设备的输出值，我们用一个按键和一个LED 小灯完成一个输入输出结合使用的实验，让大家能简单了解I/O 的作用。按键开关大家都应该比较了解，属于开关量（数字量）元件，按下时为闭合（导通）状态。完成本实验要

用到的元件如下：

按键开关\*1

红色M5 直插LED\*1

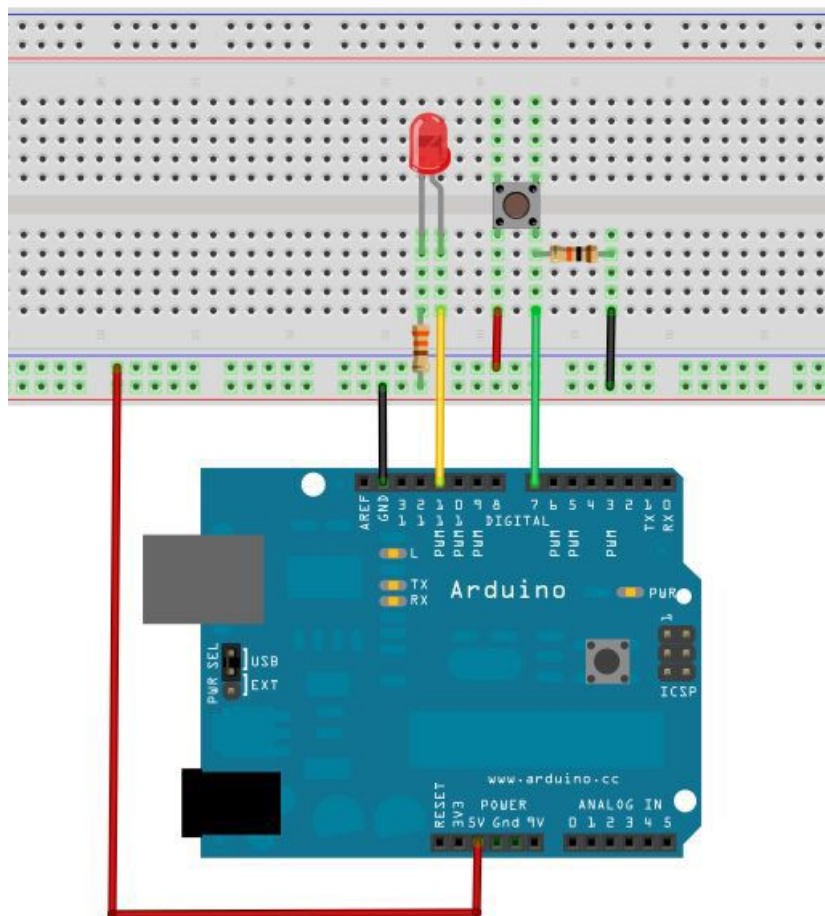
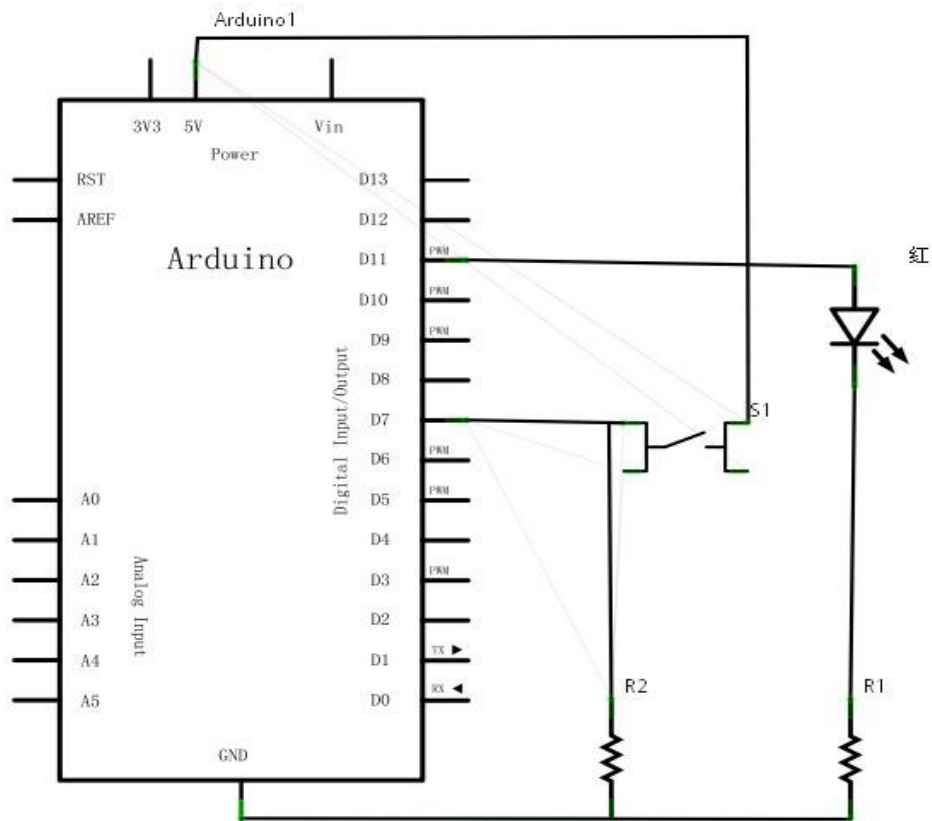
220  $\Omega$  电阻\*1

10K  $\Omega$  电阻\*1

面包板\*1

面包板跳线\*1 扎

我们将按键接到数字7 接口，红色小灯接到数字11 接口（Arduino 控制器0-13 数字I/O 接口都可以用来接按键和小灯，但是尽量不选择0 和1 接口，0 和1 接口为接口功能复用，除I/O 口功能外也是串口通信接口，下载程序时属于与PC 机通信故应保持0 和1 接口悬空，所以为避免插拔线的麻烦尽量不选用0 和1 接口），按下面的原理图连接好电路。





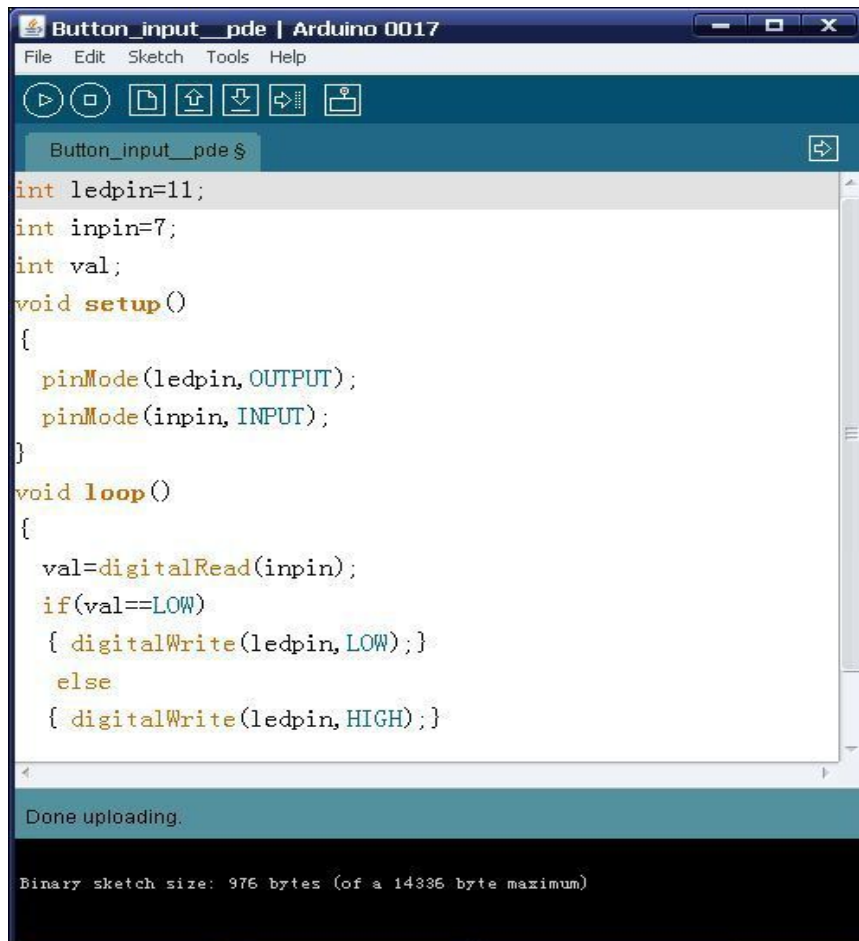
下面开始编写程序，我们就让按键按下时小灯亮起，根据前面的学习相信这个程序很容易就能编写出来，相对于前面几个实验这个实验的程序中多加了一条条件判断语句，这里我们使用if 语句，Arduino 的程序便写语句是基于C 语言的，所以C 的条件判断语句自然也适用于Arduino，像while、swich 等等。这里根据个人喜好我们习惯于使用简单易于理解的if 语句给大家做演示例程。

我们分析电路可知当按键按下时，数字7 接口可读出为高电平，这时我们使数字11 口输出高电平可使小灯亮起，程序中我们判断数字7 口是否为低电平，要为低电平使数字11 口输出也为低电平小灯不亮，原理同上。

参考源程序：

```
int ledpin=11;//定义数字11 接口
int inpin=7;//定义数字7 接口
int val;//定义变量val
void setup()
{
  pinMode(ledpin, OUTPUT);//定义小灯接口为输出接口
  pinMode(inpin, INPUT);//定义按键接口为输入接口
}
void loop()
{
  val=digitalRead(inpin);//读取数字7 口电平值赋给val
  if(val==LOW)//检测按键是否按下，按键按下时小灯亮起
  { digitalWrite(ledpin, LOW);}
  else
  { digitalWrite(ledpin, HIGH);}
}
```





```
Button_input_pde | Arduino 0017
File Edit Sketch Tools Help

Button_input_pde $

int ledpin=11;
int inpin=7;
int val;
void setup()
{
  pinMode(ledpin, OUTPUT);
  pinMode(inpin, INPUT);
}
void loop()
{
  val=digitalRead(inpin);
  if(val==LOW)
  { digitalWrite(ledpin, LOW); }
  else
  { digitalWrite(ledpin, HIGH); }
}

Done uploading.

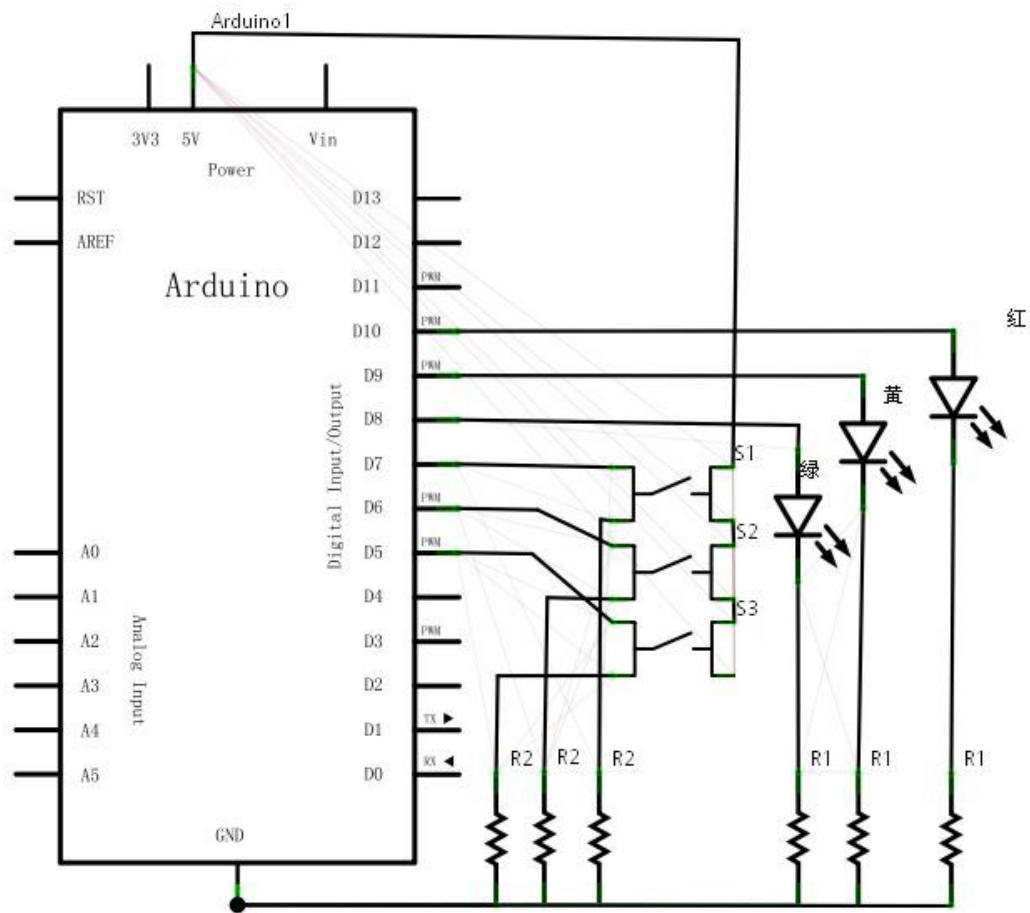
Binary sketch size: 976 bytes (of a 14336 byte maximum)
```

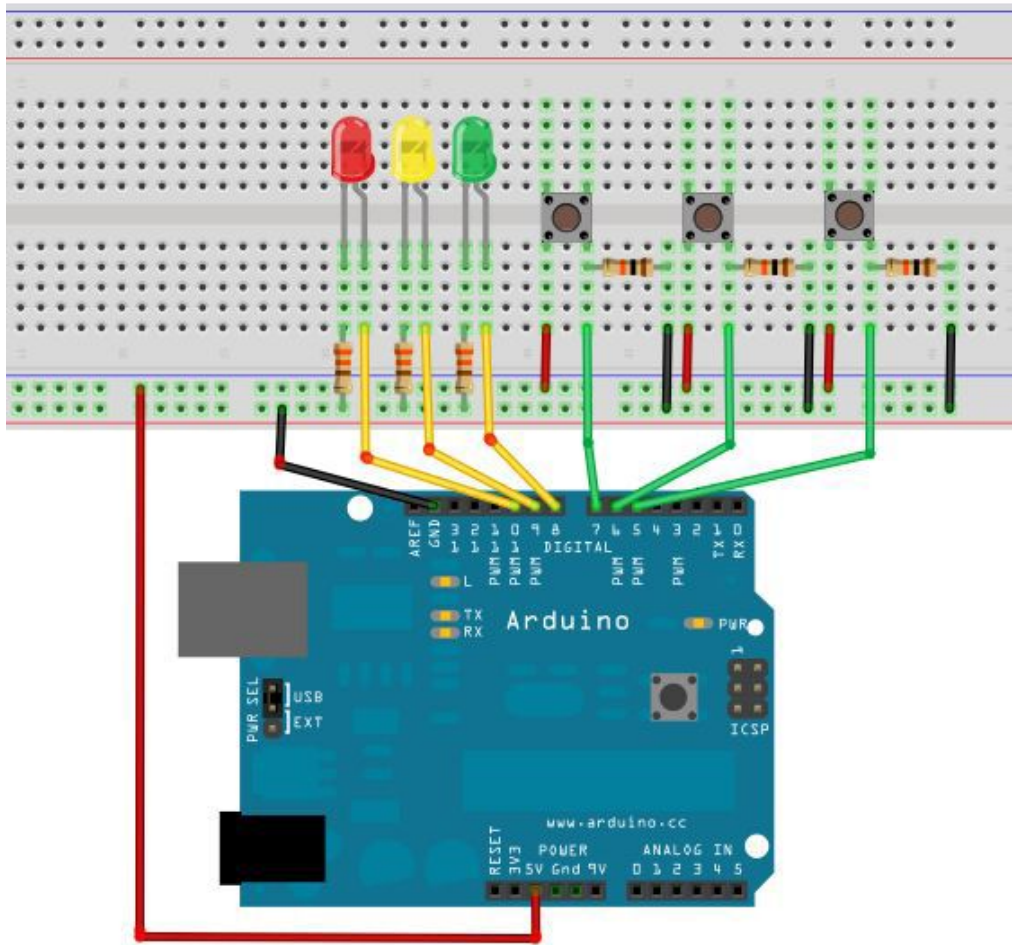
下载完程序我们本次的小灯配合按键的实验就完成了，本实验的原理很简单，广泛被用于各种电路和电器中，实际生活中大家也不难在各种设备上发现，例如大家的手机当按下任一按键时背光灯就会亮起，这就是典型应用了，下面一个实验就是一个最简单的生活中应用实例-----抢答器。

## 例程7抢答器设计实验

完成上面的实验以后相信已经有很多朋友可以独立完成这个实验了，本实验就是将上面的按键控制小灯的实验扩展成3 个按键对应3 个小灯，占用6 个数字I/O 接口。

原理这里就不多说了同上面实验，下面附上参考原理图和实物连接图。





参考源程序如下：

```
int redled=10;
int yellowled=9;
int greenled=8;
int redpin=7;
int yellowpin=6;
int greenpin=5;
int red;
int yellow;
int green;
void setup()
{
  pinMode(redled, OUTPUT);
  pinMode(yellowled, OUTPUT);
  pinMode(greenled, OUTPUT);
  pinMode(redpin, INPUT);
  pinMode(yellowpin, INPUT);
  pinMode(greenpin, INPUT);
}
void loop()
```

```
{  
red=digitalRead(redpin);  
if(red==LOW)  
{ digitalWrite(redled, LOW);}  
else  
{ digitalWrite(redled, HIGH);}  
yellow=digitalRead(yellowpin);  
if(yellow==LOW)  
{ digitalWrite(yellowled, LOW);}  
else  
{ digitalWrite(yellowled, HIGH);}  
green=digitalRead(greenpin);  
if(green==LOW)  
{ digitalWrite(greenled, LOW);}  
else  
{ digitalWrite(greenled, HIGH);}  
}
```



此程序与前面程序除接口增多以外并无异处，因此不做程序注解分析。  
下载完程序，我们自己制作的简易抢答器就完成了。

## 例程 8 蜂鸣器发声实验

用Arduino 可以完成的互动作品有很多，最常见也最常用的就是声光展示了，前面一直都是在用LED 小灯在做实验，本个实验就让大家电路发出声音，能够发出声音的最常见的元器件就是蜂鸣器和喇叭了，两者相比较蜂鸣器更简单和易用所以我们本实验采用蜂鸣器。

以下是要准备的元件：

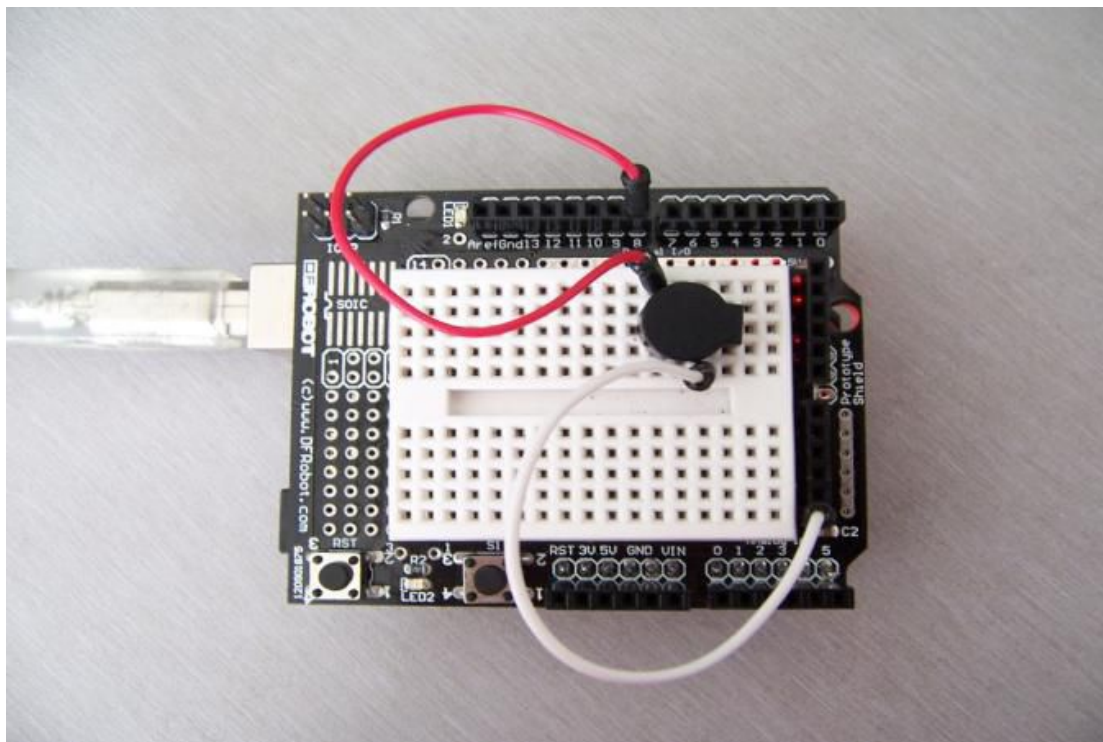
蜂鸣器\*1

按键\*1

面包板\*1

面包板跳线\*1 扎

照下面的原理图连接电路，



连接电路时要注意一点就是蜂鸣器有正负极之分，下面右侧实物图可看到蜂鸣器有红黑两种接线。连接好电路程序这方面就很简单了，与前面按键控制小灯是实验程序类似，因为蜂鸣器的控制接口也是数字接口输出高低电平就可以控制蜂鸣器的鸣响。

参考源程序：

```
int buzzer=8;//设置控制蜂鸣器的数字IO脚
void setup()
{
  pinMode(buzzer, OUTPUT);//设置数字IO脚模式，OUTPUT为输出
}
void loop()
{
  unsigned char i,j;//定义变量
  while(1)
```

```
{  
for(i=0;i<80;i++)//输出一个频率的声音  
{  
digitalWrite(buzzer,HIGH);//发声音  
delay(1);//延时1ms  
digitalWrite(buzzer,LOW);//不发声音  
delay(1);//延时ms  
}  
for(i=0;i<100;i++)//输出另一个频率声音  
{  
digitalWrite(buzzer,HIGH);//发声音  
delay(2);//延时2ms  
digitalWrite(buzzer,LOW);//不发声音  
delay(2);//延时2ms  
}  
}  
}
```

下载完程序，蜂鸣器实验就完成了。

## 例程 9 模拟值读取实验

本个实验我们就来开始学习一下模拟I/O 接口的使用，Arduino 有模拟0—模拟5 共计6 个模拟接口，这6 个接口也可以算作为接口功能复用，除模拟接口功能以外，这6 个接口可作为数字接口使用，编号为数字14—数字19，简单了解以后，下面就来开始我们的实验。电位计是大家比较熟悉的典型的模拟值输出元件，本实验就用它来完成。

所需元器件有：

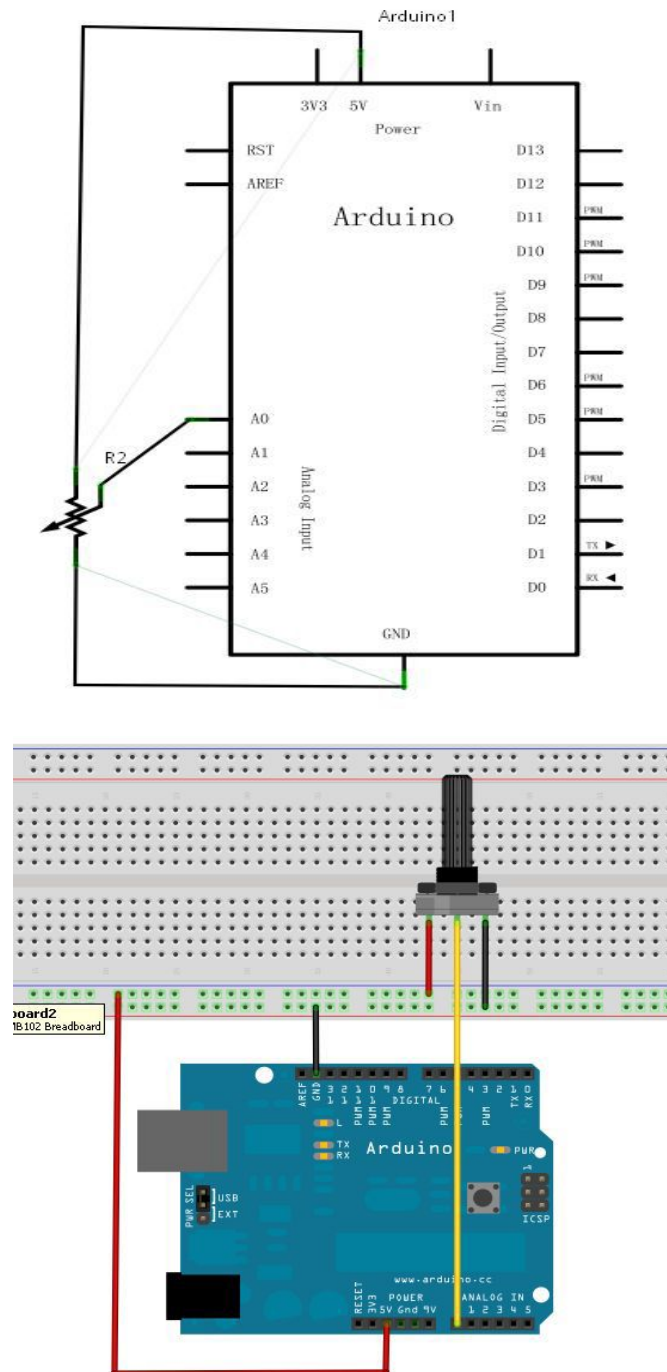
电位计\*1

面包板\*1

面包板跳线\*1 扎

本实验我们将电位计的阻值转化为模拟值读取出来，然后显示到屏幕上，这也是我们以后完成自己所需的实验功能所必须掌握的实例应用。我们先要按照以下电路图连接实物图





我们使用的是模拟0 接口。

程序的编写也很简单，一个`analogRead()` ;语句就可以读出模拟口的值，Arduino 328是10 位的A/D 采集，所以读取的模拟值范围是0-1023，本个实验的程序里还有一个难点就是显示数值在屏幕这一问题，学习起来也是很简单的。首先我们要在`void setup()` 里面设置波特率，显示数值属于Arduino 与PC 机通信，所以Arduino 的波特率应与PC 机软件设置的相同才能显示出正确的数值，否则将会显示乱码或是不显示，在Arduino 软件的监视窗口右下角有一个可以设置波特率的按钮，这里设置的波特率需要跟程序里`void setup()` 里面设置波特率相同，程序设置波特率的语句为`Serial.begin()` ;括号中为波特率的值。其次就是显示数值的语句了，`Serial.print()` ;或者`Serial.println()` ;



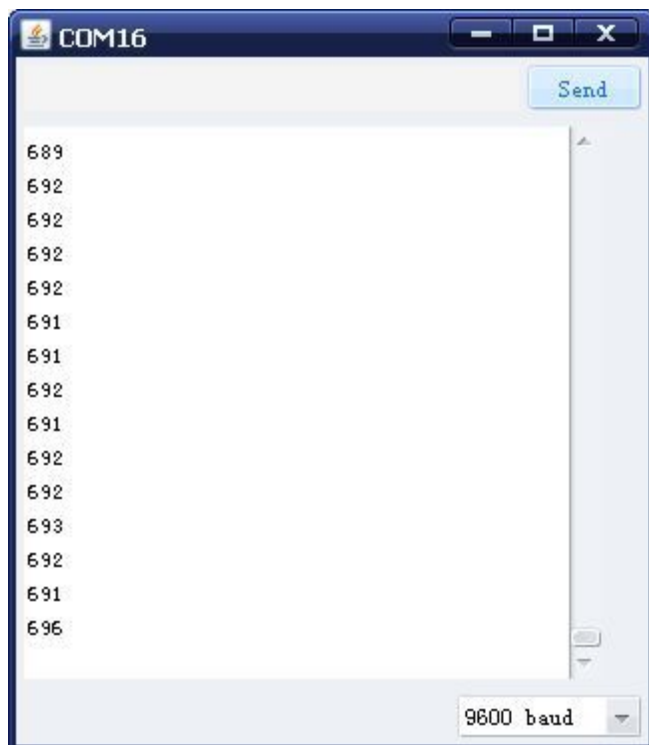
都可以，不同的是后者显示完数值后自动回车，前者不是，更多的关于语句的讲解前面有介绍这里就不再多说了。

下面是参考源程序：

```
int potpin=0;//定义模拟接口0
int ledpin=13;//定义数字接口13
int val=0;//将定义变量val, 并赋初值0
void setup()
{
  pinMode(ledpin, OUTPUT);//定义数字接口为输出接口
  Serial.begin(9600);//设置波特率为9600
}
void loop()
{
  digitalWrite(ledpin,HIGH);//点亮数字接口13 的LED
  delay(50);//延时0.05 秒
  digitalWrite(ledpin,LOW);//熄灭数字接口13 的LED
  delay(50);//延时0.05 秒
  val=analogRead(potpin);//读取模拟接口0 的值，并将其赋给val
  Serial.println(val);//显示出val 的值
}
```

参考程序借用了Arduino 数字13 口自带的LED 小灯，每读一次值小灯就会闪烁一下。

下面就是读出的模拟值。



本实验到这里就完成了，当您旋转电位计旋钮的时候就可以看到屏幕上数值的变化了，读取模拟值这个方法将一直陪伴我们，模拟值读取是我们很常用的功能，因为很多传感器都是模拟值输出，我们读出模拟值后再进行相应的算法处理，就可以应用到我们需要实现的功能里了。

## 例程 10 光控声音实验

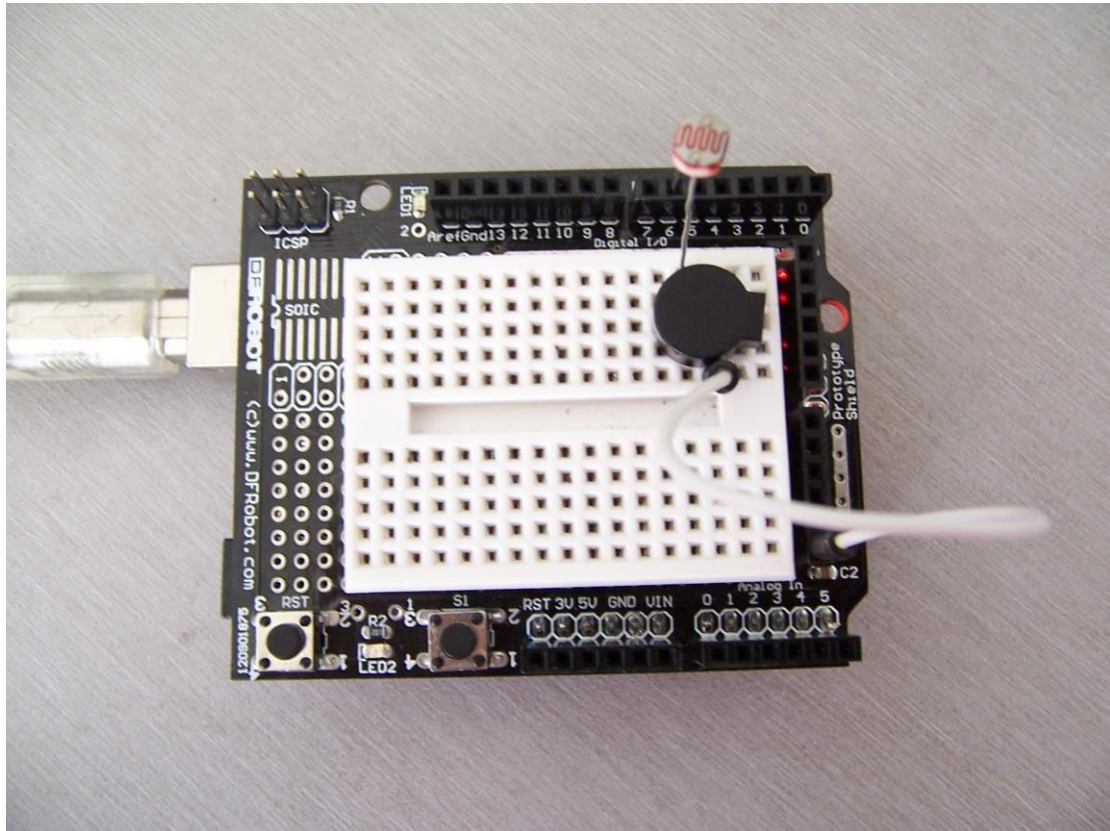
### 1、实验器件

光敏电阻：1个

蜂鸣器：1个

多彩面包板实验跳线：若干

### 2、实验连线



按照Arduino教程将控制板、prototype板子、面包板连接好，下载线接好。光敏电阻的一端接在数字6口，另一端与蜂名起正极相连,蜂明器的负极和GND相连。

### 3、实验原理

本程序应用前面几节读取模拟口电压值的方法，直接将光敏电阻接在数字口。程序类似第二节蜂鸣器发声的程序，没有光照时，正常发出声音，但声音特别的小；当有光照时，光敏电阻的阻值减小，所以蜂鸣器两端的电压就会增大，蜂鸣器声音发大。光照越强，电阻越小，蜂鸣器越响。

程序说明：

```
void setup()
{
  pinMode(6, OUTPUT);
}
void loop()
```

```
{
while(1)
{
char i, j;
while(1)
{
for(i=0;i<80;i++) //输出一个频率乱声音
{
digitalWrite(6, HIGH);
delay(1);
digitalWrite(6, LOW);
delay(1);
}
for(i=0;i<100;i++) //输出另一个频率乱声音
{
digitalWrite(6, HIGH);
delay(2);
digitalWrite(6, LOW);
delay(2);
}
}
}
}
```

将程序下载到实验板后，可以用手电筒或其他发光物体照射光敏电阻，可以听到有光照时蜂鸣器声音更大。

**掌握**本程序后，大家可以自己动手设计实验，也可以用光敏电阻控制 led 灯亮度。

## 例程 11 感光灯实验

完成以上的各种实验后，我们对Arduino 的应用也应该有一些认识和了解了，在基本的数字量输入输出和模拟量输入以及PWM 的产生都掌握以后，我们就可以开始进行一些传感器的应用了。

光敏电阻器（photoresistor）又叫光敏电阻，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）。

光敏电阻可广泛应用于各种光控电路，如对灯光的控制、调节等场合，也可用于光控开关。本次实验我们先进行一个较为简单的光敏电阻的使用实验。光敏电阻既然是可以根据光强改变阻值的元件，自然也需要模拟口读取模拟值了，本实验可以借鉴PWM 接口实验，将电位计换做光敏电阻实现当光强不同时LED 小灯的亮度也会有相应的变化。

下面是所需要的元器件：

光敏电阻\*1

红色M5 直插LED\*1

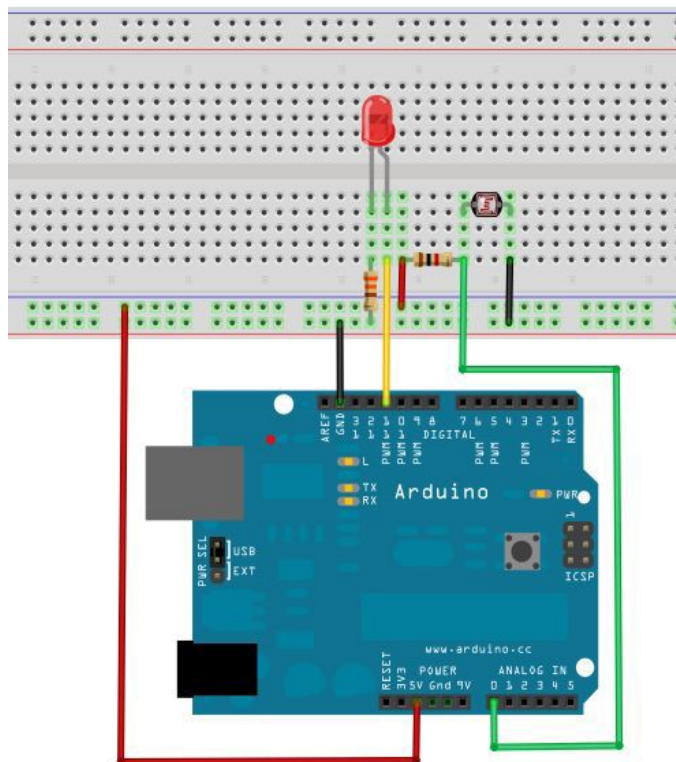
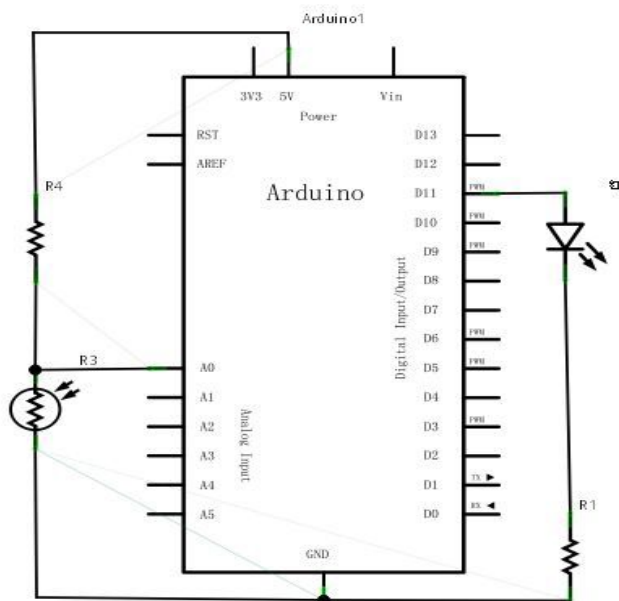
10K $\Omega$  直插电阻\*1

220 $\Omega$  直插电阻\*1

面包板\*1

面包板跳线\*1 扎

按照以下原理图连接电路。



连接好就可以编写程序了，本实验程序与PWM 实验程序相类似只是在PWM 值赋值时根据我们现在的电路稍有修改（修改部分见参考源程序）。

参考源程序：

```
int potpin=0;//定义模拟接口0 连接光敏电阻
```

```
int ledpin=11;//定义数字接口11 输出PWM 调节LED 亮度
int val=0;//定义变量val
void setup()
{
  pinMode(ledpin, OUTPUT);//定义数字接口11 为输出
  Serial.begin(9600);//设置波特率为9600
}
void loop()
{
  val=analogRead(potpin);//读取传感器的模拟值并赋值给val
  Serial.println(val);//显示val 变量数值
  analogWrite(ledpin, val);// 打开LED 并设置亮度（PWM 输出最大值255）
  delay(10);//延时0.01 秒
}
```

这里我们将传感器返回值除以4，原因是模拟输入analogRead（）函数的返回值范围是0 到1023，而模拟输出analogWrite（）函数的输出值范围是0 到255。下载完程序再试着改变光敏电阻所在的环境的光强度就可以看到我们的小灯有相应的变化了。在日常生活中光敏电阻的应用是很广泛的，用法也是很多，大家可以根据这个实验举一反三，做出更好的互动作品。

## 例程 12LM35 温度传感器实验

LM35 是很常用且易用的温度传感器元件，在元器件的应用上也只需要一个LM35元件，只利用一个模拟接口就可以，难点在于算法上的将读取的模拟值转换为实际的温度。

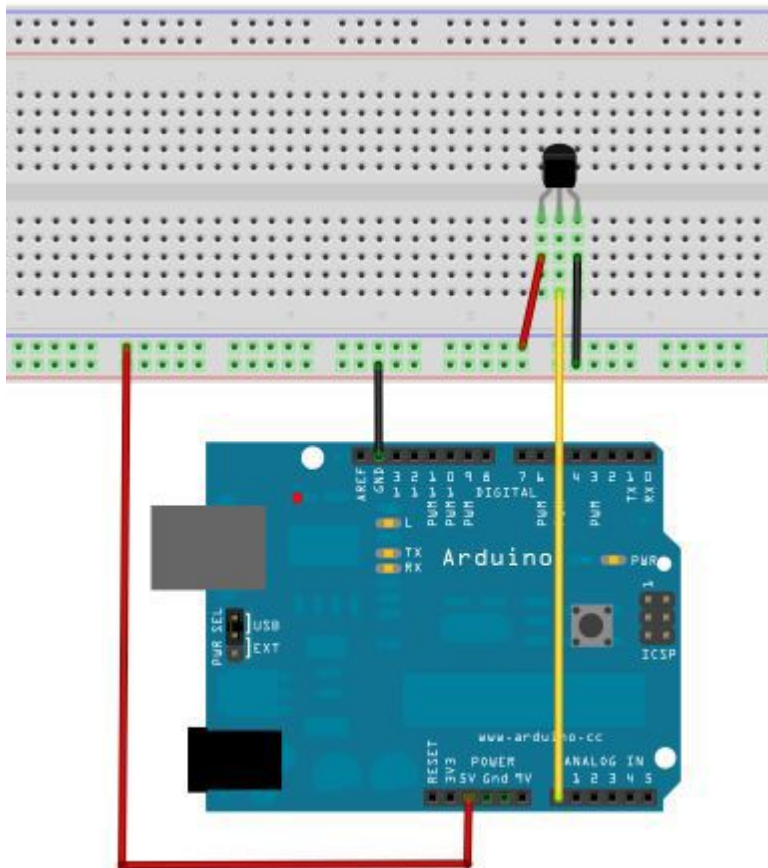
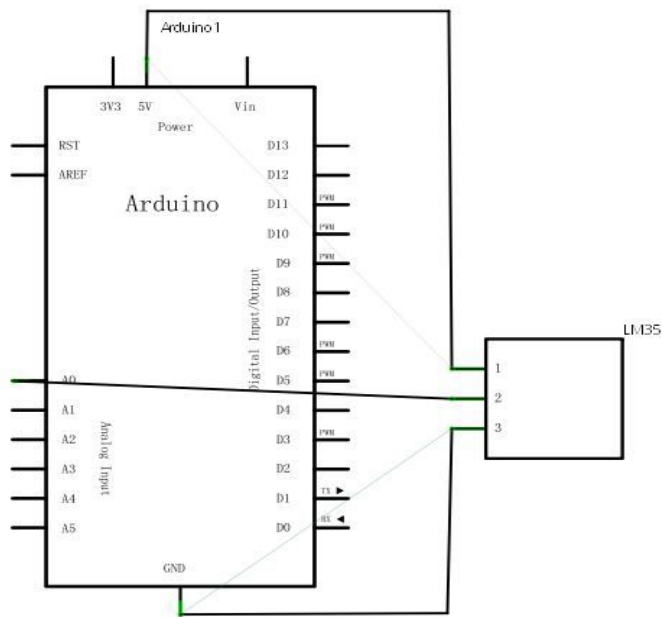
所需的元器件如下。

直插LM35\*1

面包板\*1

面包板跳线\*1 扎

按照下面原理图连接电路。



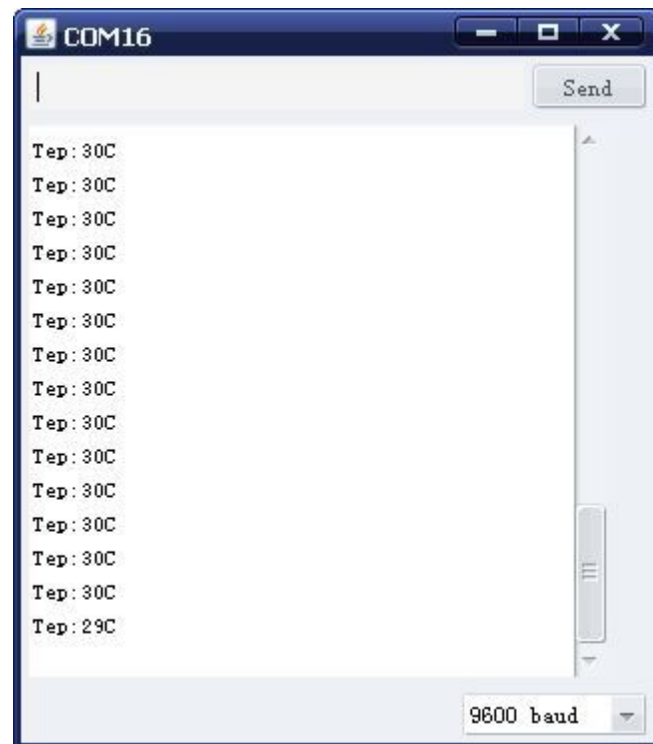
参考源程序:

```
int potPin = 0; //定义模拟接口0 连接LM35 温度传感器
```



```
void setup()
{
  Serial.begin(9600); //设置波特率
}
void loop()
{
  int val; //定义变量
  int dat; //定义变量
  val = analogRead(0); // 读取传感器的模拟值并赋值给val
  dat = (125 * val) >> 8; //温度计算公式
  Serial.print("Tep:"); //原样输出显示Tep 字符串代表温度
  Serial.print(dat); //输出显示dat 的值
  Serial.println("C"); //原样输出显示C 字符串
  delay(500); //延时0.5 秒
}
```

下载完程序打开监视窗就可以看见当前的温度了。



## 例程 13 倾斜开关实验

倾斜开关控制led灯的亮灭

实验器件

滚珠开关

10k电阻

多彩面包板实验跳线：若干

### 2、实验连线

按照Arduino教程将控制板、扩展板子、面包板连接好，下载线接好。滚珠开关一端接VCC另一段串联电阻接地并用模拟5引脚采样。



3、实验原理 当开关一端低于水平位置倾斜，开关寻通，模拟口电压值为5V左右（数字二进制表示为1023），点亮led灯。当另一端低于水平位置倾斜，开关停止，模拟口电压值为0V左右（数字二进制表示为0），熄灭led灯。在程序中模拟口电压值是否大于2.5V左右（数字二进制表示为512），即可知道是否倾斜开关寻通了。

#### 4、程序代码

```
void setup()
{
  pinMode(13, OUTPUT); //设置 13 引脚为输出模式
}

void loop()
{
  if(analogRead(5)>512) //如果大于 512 (2.5V)
    digitalWrite(13, HIGH); //点亮 led 灯
  else //否则
    digitalWrite(13, LOW); //熄灭 led 灯
}
```

将程序下载到实验板后大家可以将板子倾斜观察led灯状态。当金色一端低于水平位置倾斜，开关寻通，点亮led灯；当银色一端低于水平位置倾斜，开关截止，模拟口电压值为0V左右（数字二进制表示为0），熄灭led灯。

**掌握**本程序后，大家可以按照自己的想法实验，还可以控制其他器件例如蜂鸣器等

## 例程 14 火焰报警实验

### 一、火焰传感器介绍

#### 1、认识火焰传感器



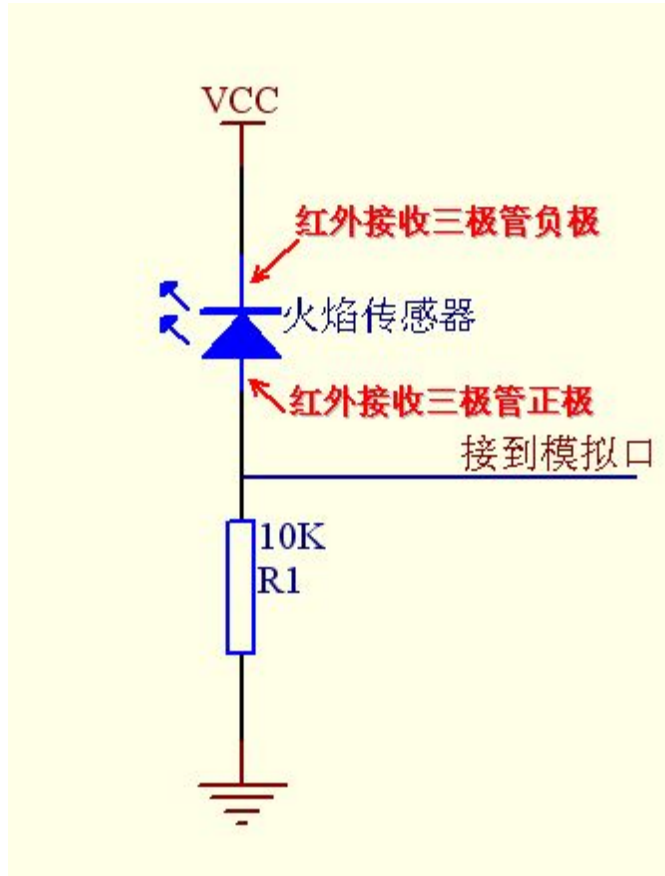
火焰传感器（即红外接收三极管）是机器人专门用来搜寻火源的传感器，本传感器对火焰特别灵敏。实物如图：

#### 2、工作原理

火焰传感器利用红外线对火焰非常敏感的特点，使用特制的红外线接收管来检测火焰，然后把火焰的亮度转化为高低变化的电平信号，输入到中央处理器，中央处理器根据信号的变化做出相应的程序处理。

#### 3、火焰传感器的连线

红外接收三极管的短引线端为负极，长引线端为正极。按照下图将负极接到5V接口中，然后将正极和10K电阻相连，电阻的另一端接到GND接口中，最后从火焰传感器的正极端所在列接入一根跳线，跳线的另一端接在模拟口中。如图



## 二、火焰报警实验

### 1、实验器件

火焰传感器：1个

蜂鸣器：1个

10K电阻：1个

多彩面包板实验跳线：若干

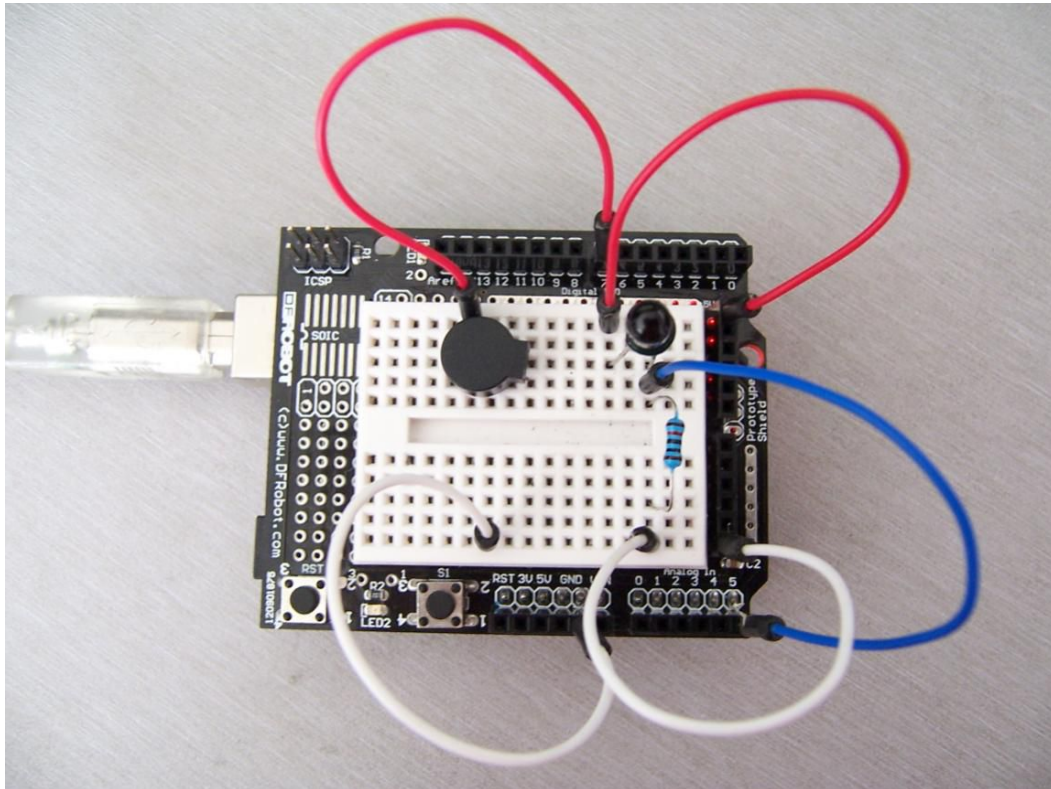
### 2、实验连线

#### 1) 蜂鸣器的连接

首先 按照Arduino教程将控制板、prototype板子、面包板连接好，下载线接好。从实验盒中取出蜂鸣器，按照第二节实验蜂鸣器的连接方法，将蜂鸣器连接到数字第八口。完成蜂鸣器的连接。

#### 2) 火焰传感器的连接

从实验盒中取出火焰传感器，按照本节所讲述的火焰传感器的接线方法，将火焰传感器接到模拟5口。完成整个实验的连线。



### 3、实验原理

在有火焰靠近和没有火焰靠近两种情况下，模拟口读到的电压值是有变化的。实际用万用表测量可知，在没有火焰靠近时，模拟口读到电压值为0.3V左右；当有火焰靠近时，模拟口读到电压值为1.0V左右，火焰靠近距离越近电压值越大。

所以在程序一开始，我大可以先存储一个没有火焰时模拟口的电压值*i*。接着不断的循环读取模拟口电压值*j*、同存储的值做差值*k=j-i*、差值*k*不0.6v做比较。差值*k*如果大于0.6V（数字二进制值为123），则判断有火焰靠近让蜂鸣器发出声音以作报警；如果差值小于0.6v则蜂鸣器不响。

### 4、程序代码

```
int flame=A5;//定义火焰接口为模拟5 接口
int Beep=8;//定义蜂鸣器接口为数字8 接口
int val=0;//定义数字变量
void setup()
{
    pinMode(Beep, OUTPUT);//定义Beep 为输出接口
    pinMode(flame, INPUT);//定义flame为输入接口
    Serial.begin(9600);//设定波特率为9600
    val=analogRead(flame);
}

void loop() {
    Serial.println(analogRead(flame));//输出模拟值，并将其打印出来
    if((analogRead(flame)-val)>=600)//当模拟值大于600 时蜂鸣器鸣响
        digitalWrite(Beep, HIGH); }
```

## 5、下载程序

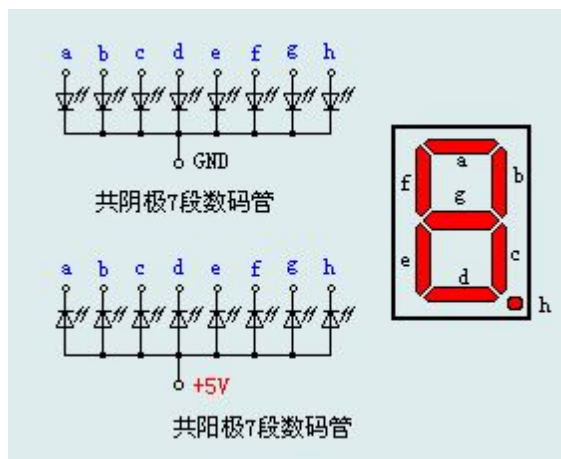
按照arduino教程中程序下载方法将本程序下载到实验板中。

## 6、程序功能

本程序可以模拟在有火焰时报警乱情呀，在没有火焰时一切正常，当有火焰时立刻报警做出提示

## 例程 15 数码管显示实验

数码管是一种半导体发光器件，其基本单元是发光二极管。数码管按段数分为七段数码管和八段数码管，八段数码管比七段数码管多一个发光二极管单元（多一个小数点显示），本实验所使用的是八段数码管。按发光二极管单元连接方式分为共阳极数码管和共阴极数码管。共阳数码管是指将所有发光二极管的阳极接到一起形成公共阳极 (COM) 的数码管。共阳数码管在应用时应将公共极COM 接到+5V，当某一字段发光二极管的阴极为低电平时，相应字段就点亮。当某一字段的阴极为高电平时，相应字段就不亮。共阴数码管是指将所有发光二极管的阴极接到一起形成公共阴极 (COM) 的数码管。共阴数码管在应用时应将公共极COM 接到地线GND 上，当某一字段发光二极管的阳极为高电平时，相应字段就点亮。当某一字段的阳极为低电平时，相应字段就不亮。



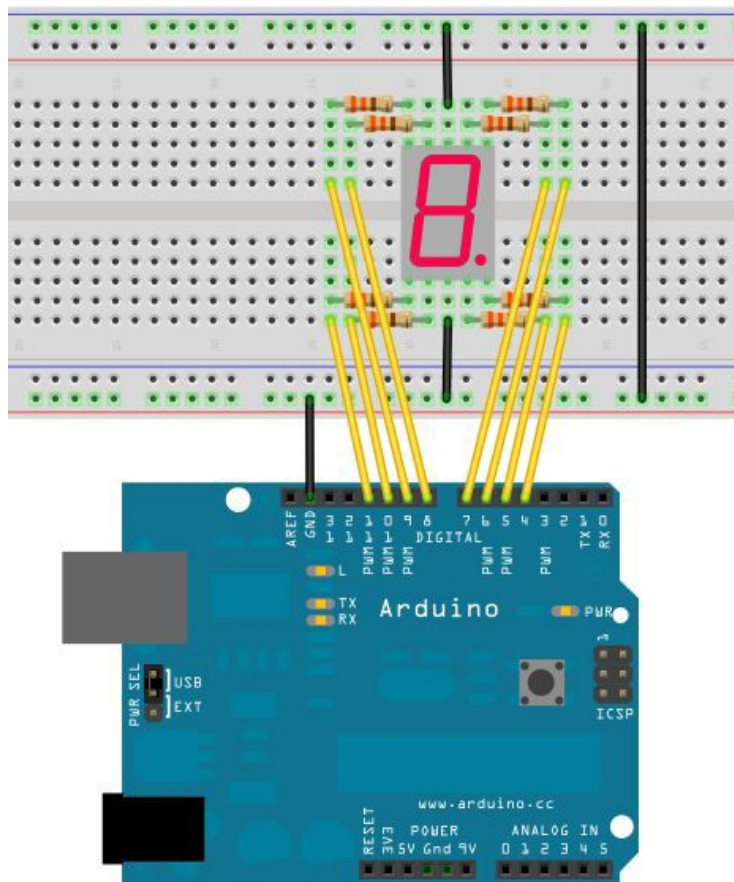
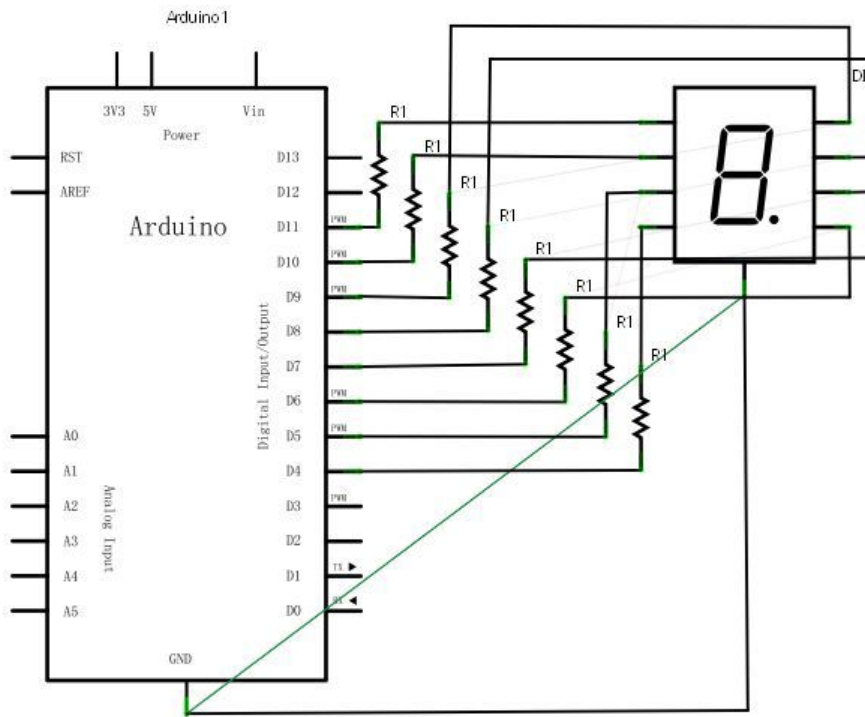
数码管的每一段是由发光二极管组成，所以在使用时跟发光二极管一样，也要连接限流电阻，否则电流过大会烧毁发光二极管的。本实验用的是共阴极的数码管，共阴数码管在应用时应将公共极接到GND，当某一字段发光二极管的阳极为低电平时，相应字段就点熄灭。当某一字段的阳极为高电平时，相应字段就点亮。介绍完原理，我们开始准备实验用元器件。

八段数码管\*1

220  $\Omega$  直插电阻\*8

面包板\*1 面包板跳线\*1 扎

我们参考实物连接图按原理图连接好电路。



数码管共有七段显示数字的段，还有一个显示小数点的段。当让数码管显示数字时，只要将



相应的段点亮即可。例如：让数码管显示数字1，则将b、c 段点亮即可。将每个数字写成一个子程序。在主程序中每隔2s 显示一个数字，让数码管循环显示1~8 数字。每一个数字显示的时间由延时时间来决定，时间设置的大些，显示的时间就长些，时间设置的小些，显示的时间就短。

参考程序源代码：

```
//设置控制各段的数字IO 脚
int a=7;//定义数字接口7 连接a 段数码管
int b=6;// 定义数字接口6 连接b 段数码管
int c=5;// 定义数字接口5 连接c 段数码管
int d=11;// 定义数字接口11 连接d 段数码管
int e=10;// 定义数字接口10 连接e 段数码管
int f=8;// 定义数字接口8 连接f 段数码管
int g=9;// 定义数字接口9 连接g 段数码管
int dp=4;// 定义数字接口4 连接dp 段数码管
void digital_1(void) //显示数字1
{
    unsigned char j;
    digitalWrite(c,HIGH);//给数字接口5 引脚高电平，点亮c 段
    digitalWrite(b,HIGH);//点亮b 段
    for(j=7;j<=11;j++)//熄灭其余段
        digitalWrite(j,LOW);
    digitalWrite(dp,LOW);//熄灭小数点DP 段
}
void digital_2(void) //显示数字2
{
    unsigned char j;
    digitalWrite(b,HIGH);
    digitalWrite(a,HIGH);
    for(j=9;j<=11;j++)
        digitalWrite(j,HIGH);
    digitalWrite(dp,LOW);
    digitalWrite(c,LOW);
    digitalWrite(f,LOW);
}
void digital_3(void) //显示数字3
{
    unsigned char j;
    digitalWrite(g,HIGH);
    digitalWrite(d,HIGH);
    for(j=5;j<=7;j++)
        digitalWrite(j,HIGH);
    digitalWrite(dp,LOW);
    digitalWrite(f,LOW);
    digitalWrite(e,LOW);
}
```

```
}  
void digital_4(void) //显示数字4  
{  
    digitalWrite(c, HIGH);  
    digitalWrite(b, HIGH);  
    digitalWrite(f, HIGH);  
    digitalWrite(g, HIGH);  
    digitalWrite(dp, LOW);  
    digitalWrite(a, LOW);  
    digitalWrite(e, LOW);  
    digitalWrite(d, LOW);  
}  
void digital_5(void) //显示数字5  
{  
    unsigned char j;  
    for(j=7; j<=9; j++)  
        digitalWrite(j, HIGH);  
    digitalWrite(c, HIGH);  
    digitalWrite(d, HIGH);  
    digitalWrite(dp, LOW);  
    digitalWrite(b, LOW);  
    digitalWrite(e, LOW);  
}  
void digital_6(void) //显示数字6  
{  
    unsigned char j;  
    for(j=7; j<=11; j++)  
        digitalWrite(j, HIGH);  
    digitalWrite(c, HIGH);  
    digitalWrite(dp, LOW);  
    digitalWrite(b, LOW);  
}  
void digital_7(void) //显示数字7  
{  
    unsigned char j;  
    for(j=5; j<=7; j++)  
        digitalWrite(j, HIGH);  
    digitalWrite(dp, LOW);  
    for(j=8; j<=11; j++)  
        digitalWrite(j, LOW);  
}  
void digital_8(void) //显示数字8  
{  
    unsigned char j;
```



```
for(j=5;j<=11;j++)
digitalWrite(j,HIGH);
digitalWrite(dp,LOW);
}
void setup()
{
int i;//定义变量
for(i=4;i<=11;i++)
pinMode(i,OUTPUT);//设置4~11 引脚为输出模式
}
void loop()
{
while(1)
{
digital_1();//显示数字1
delay(2000);//延时2s
digital_2();//显示数字2
delay(1000); //延时1s
digital_3();//显示数字3
delay(1000); //延时1s
digital_4();//显示数字4
delay(1000); //延时1s
digital_5();//显示数字5
delay(1000); //延时1s
digital_6();//显示数字6
delay(1000); //延时1s
digital_7();//显示数字7
delay(1000); //延时1s
digital_8();//显示数字8
delay(1000); //延时1s
}
}
```

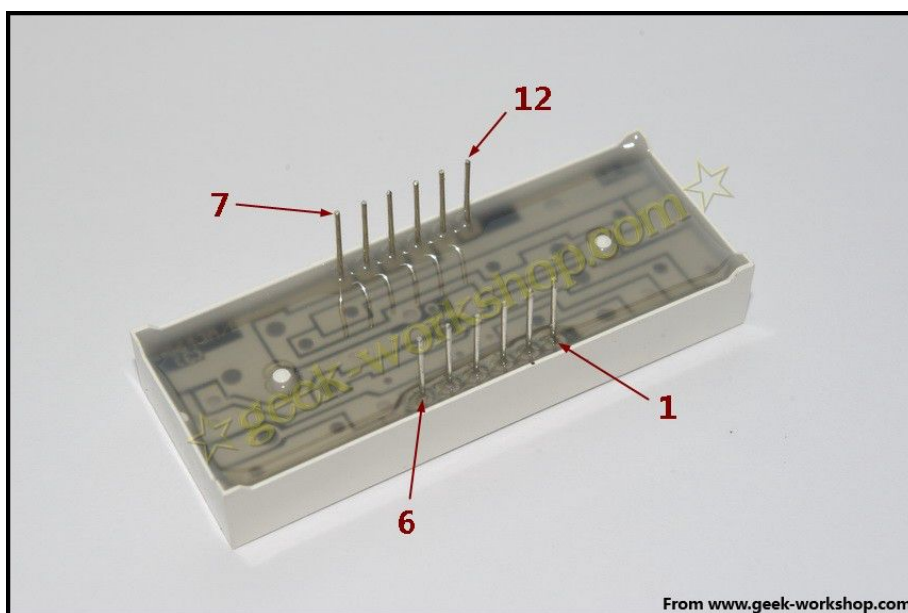
在setup()前面定义了一系列的数字显示子程序，这些子程序的定义可以方便在loop()中使用，使用时只需将子程序的名写上即可。

## 例程 16 共阴四位数码管

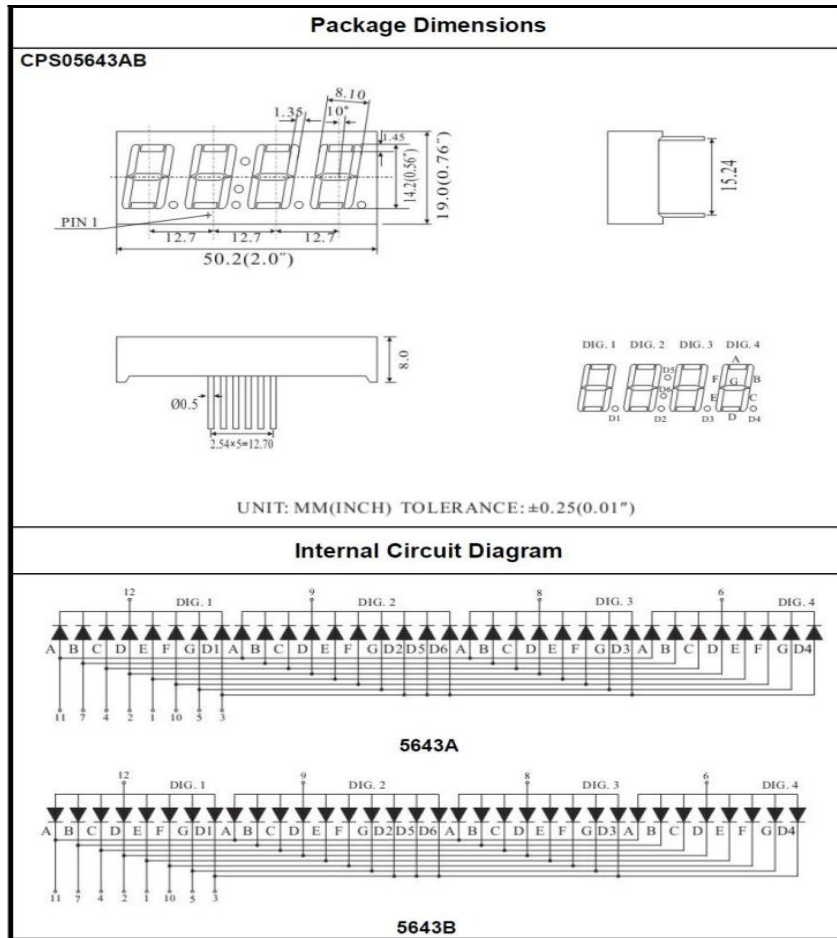
这次我们进行的实验是使用 arduino 驱动一块共阴四位数码管。驱动数码管限流电阻肯定是必不可少的，限流电阻有两种接法，一种是在 d1-d4 阳极接，总共接 4 颗。这种接法好处是需求电阻比较少，但是会产生每一位上显示不同数字亮度会不一样，1 最亮，8 最暗。另外一种接法就是在其他 8 个引脚上接，这种接法亮度显示均匀，但是用电阻较多。本次实验使用 8 颗 220Ω 电阻（因为没有 100Ω 电阻，所以使用 220Ω 的代替，100 欧姆亮度会比较高）。



4 位数码管总共有 12 个引脚，小数点朝下正放在面前时，左下角为 1, 其他管脚顺序为逆时针旋转。左上角为最大的 12 号管脚。

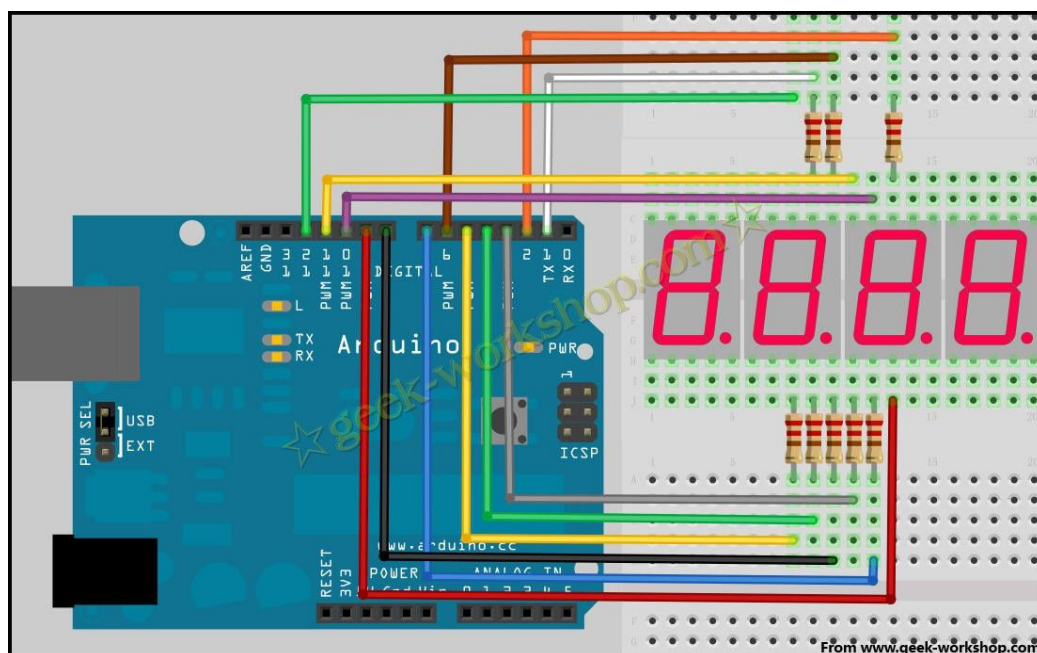


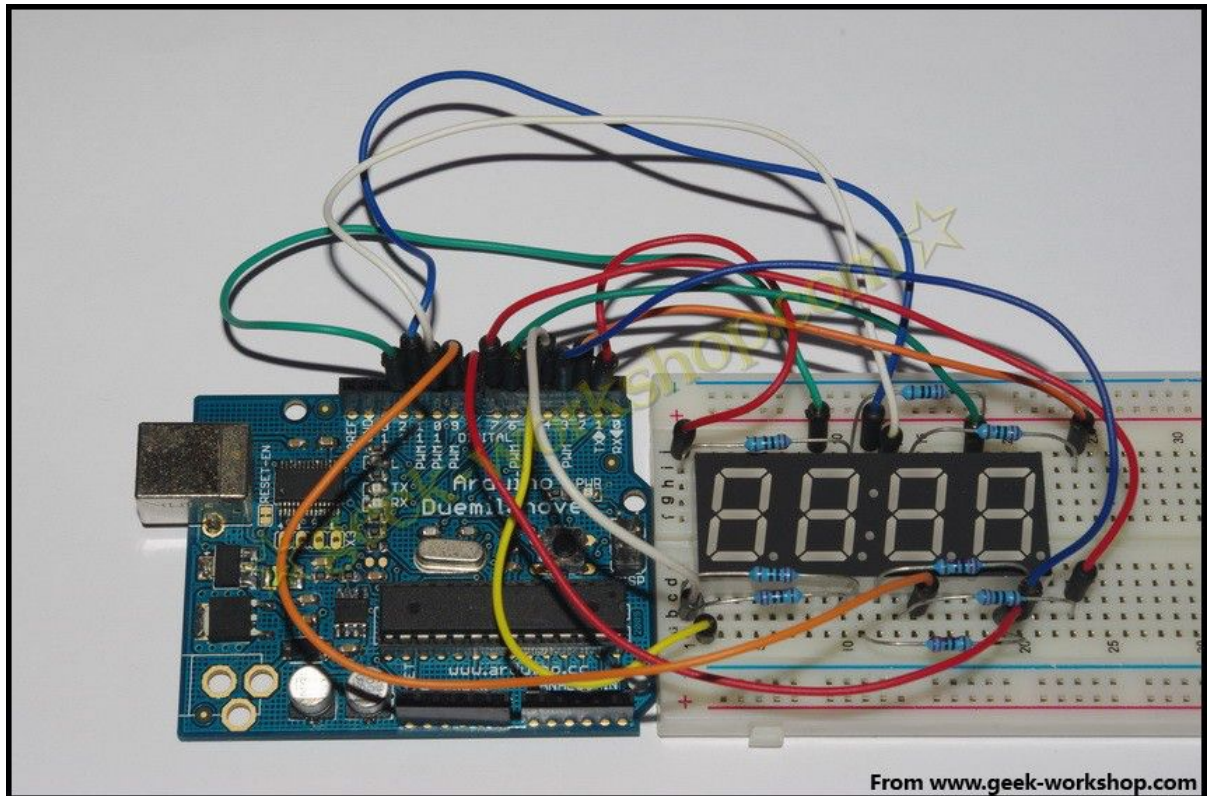
下图为数码管的说明手册



Four Digits Displays Series

下面是硬件连接图





把下面代码复制下载到控制板中，看看效果。

```
#define SEG_A 2
#define SEG_B 3
#define SEG_C 4
#define SEG_D 5
#define SEG_E 6
#define SEG_F 7
#define SEG_G 8
#define SEG_H 9

#define COM1 10
#define COM2 11
#define COM3 12
#define COM4 13

unsigned char table[10][8] =
{
    {0, 0, 1, 1, 1, 1, 1, 1},          //0
    {0, 0, 0, 0, 0, 1, 1, 0},          //1
    {0, 1, 0, 1, 1, 0, 1, 1},          //2
    {0, 1, 0, 0, 1, 1, 1, 1},          //3
    {0, 1, 1, 0, 0, 1, 1, 0},          //4
    {0, 1, 1, 0, 1, 1, 0, 1},          //5
```

```
{0, 1, 1, 1, 1, 1, 0, 1},          //6
{0, 0, 0, 0, 0, 1, 1, 1},          //7
{0, 1, 1, 1, 1, 1, 1, 1},          //8
{0, 1, 1, 0, 1, 1, 1, 1}           //9
};

void setup()
{
    pinMode(SEG_A, OUTPUT);          //设置为输出引脚
    pinMode(SEG_B, OUTPUT);
    pinMode(SEG_C, OUTPUT);
    pinMode(SEG_D, OUTPUT);
    pinMode(SEG_E, OUTPUT);
    pinMode(SEG_F, OUTPUT);
    pinMode(SEG_G, OUTPUT);
    pinMode(SEG_H, OUTPUT);

    pinMode(COM1, OUTPUT);
    pinMode(COM2, OUTPUT);
    pinMode(COM3, OUTPUT);
    pinMode(COM4, OUTPUT);
}

void loop()
{
    Display(1, 1);                    //第 1 位显示 1
    delay(500);
    Display(2, 2);                    //第 2 位显示 2
    delay(500);
    Display(3, 3);                    //第 3 位显示 3
    delay(500);
    Display(4, 4);                    //第 4 位显示 4
    delay(500);
}

void Display(unsigned char com, unsigned char num)
{
    digitalWrite(SEG_A, LOW);         //去除余晖
    digitalWrite(SEG_B, LOW);
    digitalWrite(SEG_C, LOW);
    digitalWrite(SEG_D, LOW);
    digitalWrite(SEG_E, LOW);
    digitalWrite(SEG_F, LOW);
    digitalWrite(SEG_G, LOW);
```

```
digitalWrite(SEG_H, LOW);

switch(com)                                //选通位选
{
    case 1:
        digitalWrite(COM1, LOW);           //选择位 1
        digitalWrite(COM2, HIGH);
        digitalWrite(COM3, HIGH);
        digitalWrite(COM4, HIGH);
        break;
    case 2:
        digitalWrite(COM1, HIGH);
        digitalWrite(COM2, LOW);           //选择位 2
        digitalWrite(COM3, HIGH);
        digitalWrite(COM4, HIGH);
        break;
    case 3:
        digitalWrite(COM1, HIGH);
        digitalWrite(COM2, HIGH);
        digitalWrite(COM3, LOW);          //选择位 3
        digitalWrite(COM4, HIGH);
        break;
    case 4:
        digitalWrite(COM1, HIGH);
        digitalWrite(COM2, HIGH);
        digitalWrite(COM3, HIGH);
        digitalWrite(COM4, LOW);          //选择位 4
        break;
    default:break;
}

digitalWrite(SEG_A, table[num][7]);        //a 查询码值表
digitalWrite(SEG_B, table[num][6]);
digitalWrite(SEG_C, table[num][5]);
digitalWrite(SEG_D, table[num][4]);
digitalWrite(SEG_E, table[num][3]);
digitalWrite(SEG_F, table[num][2]);
digitalWrite(SEG_G, table[num][1]);
digitalWrite(SEG_H, table[num][0]);
}
```



## 例程 17 74HC595 实验

74HC595 简单说来就是具有8 位移位寄存器和一个存储器，以及三态输出功能。 这里我们用它来控制8 个LED 小灯。我们为什么要用74HC595 来控制小灯呢？一定会有很多朋友会问这个问题，我想问的是我们要是单纯的用Arduino 控制8 个小灯的话要占用多少个I/O 呢？答案是8 个，但是我们的Arduino 168 有几个I/O 口呢？加上模拟接口也就20 个吧，这8 个小灯占用了太多的资源了，我们用74HC595 的目的就是减少I/O 口的使用数量。用74HC595 以后我们可以用3 个数字I/O 口控制8 个LED 小灯岂不美哉。下面是我们要准备的元器件。

74HC595 直插芯片\*1

红色M5 直插LED\*4

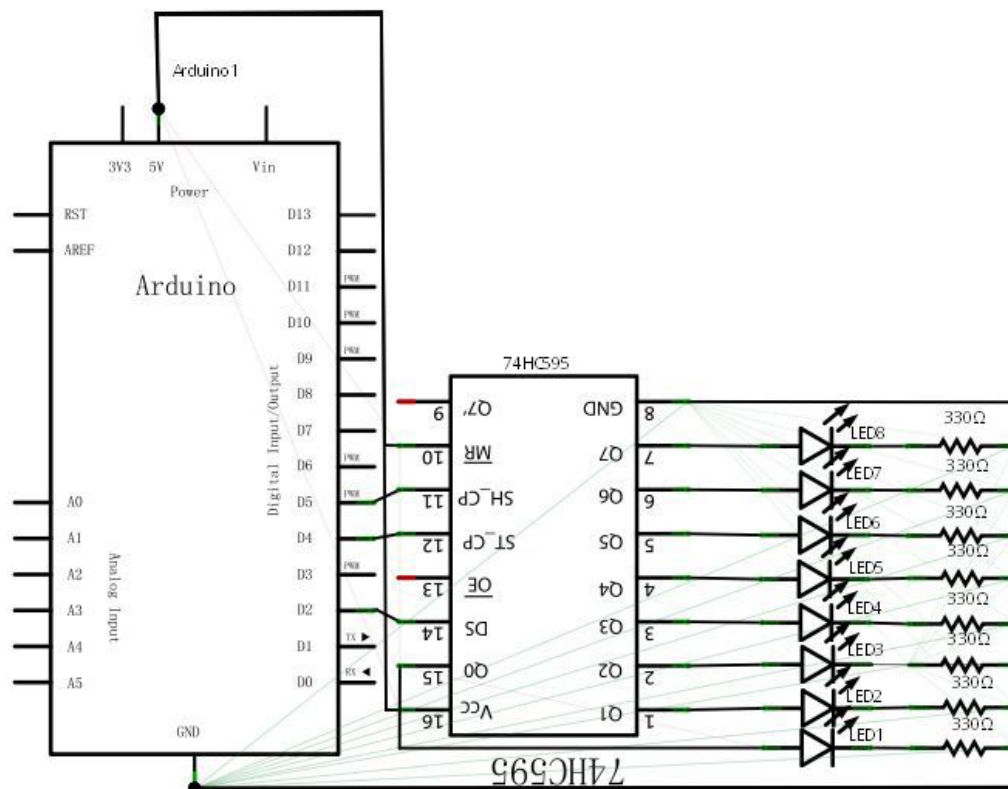
绿色M5 直插LED\*4

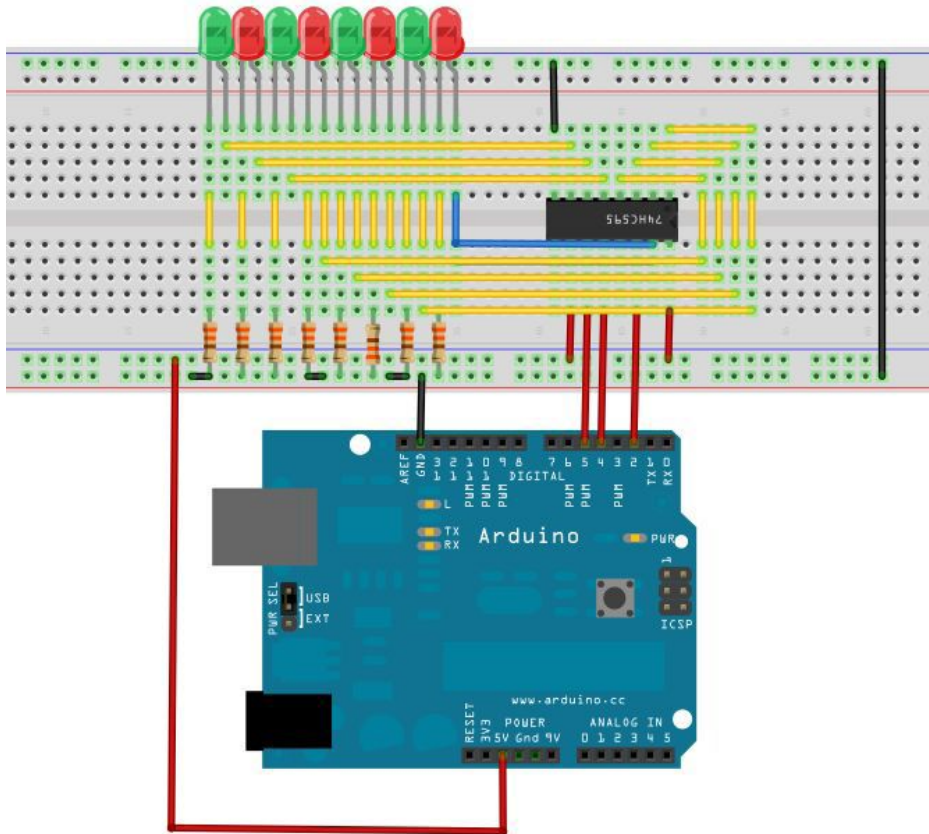
220  $\Omega$  直插电阻\*8

面包板\*1

面包板跳线\*1 扎

准备好元件我们就按下面的原理图连接电路。





此电路图看似复杂，我们仔细分析以后再结合参考实物就会发现很简单。

下面是参考源程序:

```
int latchPin = 5;
int clockPin = 4;
int dataPin = 2; //这里定义了那三个脚
void setup ()
{
    pinMode(latchPin, OUTPUT);
    pinMode(clockPin, OUTPUT);
    pinMode(dataPin, OUTPUT); //让三个脚都是输出状态
}
void loop()
{
    for(int a=0; a<256; a++)
        //这个循环的意思是让 a 这个变量+1 一直加到到 256，每次循环都进行下面的活动
    {
        digitalWrite(latchPin, LOW); //将 ST_CP 口上面加低电平让芯片准备好接收数据
        shiftOut(dataPin, clockPin, MSBFIRST, a);
        //这个就是用 MSBFIRST 参数让 0-7 个针脚以高电平输出(LSBFIRST 低电平)是 dataPin
        的参数，
        //clockPin 的参数是变量 a，前面我们说了这个变量会一次从 1+1 到 256，是个十进制数，
        // 输入到芯片后会产生 8 个二进制数，达到开关的作用
    }
}
```

```

digitalWrite(latchPin, HIGH); //将 ST_CP 这个针脚恢复到高电平
delay(1000); //暂停 1 秒钟让你看到效果
}
}

```

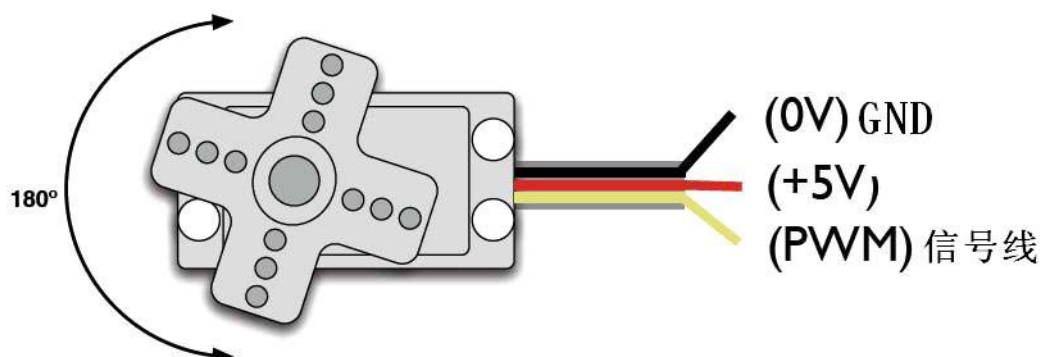
下载完程序大家就可以看到 8 个小灯闪烁的美妙场景了。

## 例程 18 舵机控制实验

舵机是一种位置伺服的驱动器，主要是由外壳、电路板、无核心马达、齿轮与位置检测器所构成。其工作原理是由接收机或者单片机发出信号给舵机，其内部有一个基准电路，产生周期为20ms，宽度为1.5ms 的基准信号，将获得的直流偏置电压与电位器的电压比较，获得电压差输出。经由电路板上的IC 判断转动方向，再驱动无核心马达开始转动，透过减速齿轮将动力传至摆臂，同时由位置检测器送回信号，判断是否已经到达定位。适用于那些需要角度不断变化并可以保持的控制系统。当电机转速一定时，通过级联减速齿轮带动电位器旋转，使得电压差为0，电机停止转动。一般舵机旋转的角度范围是0 度到180 度。

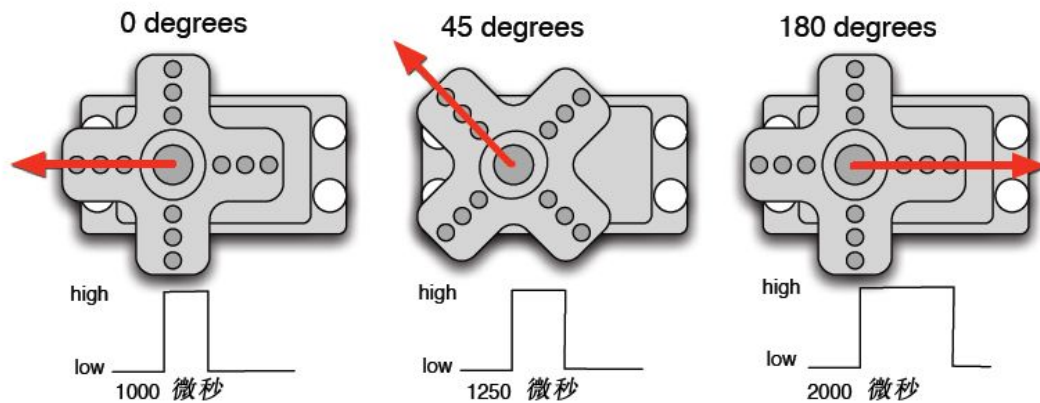


舵机有很多规格，但所有的舵机都有外接三根线，分别用棕、红、橙三种颜色进行区分，由于舵机品牌不同，颜色也会有所差异，棕色为接地线，红色为电源正极线，橙色为信号线。



舵机的转动的角度是通过调节PWM（脉冲宽度调制）信号的占空比来实现的，标准PWM（脉冲宽度调制）信号的周期固定为20ms（50Hz），理论上脉宽分布应在1ms到2ms 之间，但是，事实上脉宽可由0.5ms 到2.5ms 之间，脉宽和舵机的转角0° ~180° 相对应。有一点值得注

意的地方，由于舵机牌子不同，对于同一信号，不同牌子的舵机旋转的角度也会有所不同。



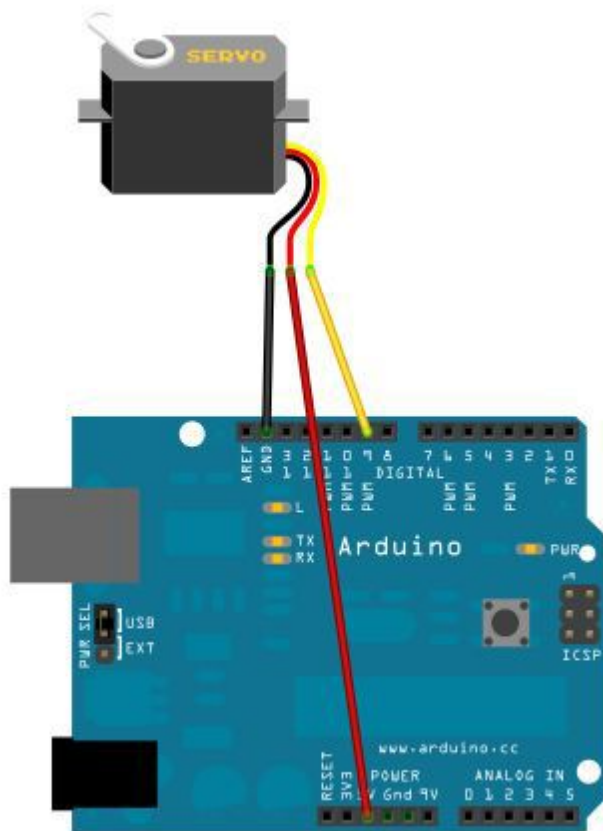
了解了基础知识以后我们就可以来学习控制一个舵机了，本实验所需要的元器件很少只需要舵机一个、跳线一扎就可以了。

RB—412 舵机\*1

面包板跳线\*1 扎

用Arduino 控制舵机的方法有两种，一种是通过Arduino 的普通数字传感器接口产生占空比不同的方波，模拟产生PWM 信号进行舵机定位，第二种是直接利用Arduino 自带的Servo 函数进行舵机的控制，这种控制方法的优点在于程序编写，缺点是只能控制2 路舵机，因为Arduino 自带函数只能利用数字9、10 接口。Arduino 的驱动能力有限，所以当需要控制1 个以上的舵机时需要外接电源。

方法一



将舵机接数字 9 接口上。

编写一个程序让舵机转动到用户输入数字所对应的角度数的位置，并将角度打印显示到屏幕上。

参考源程序A:

```
int servopin=9;//定义数字接口9 连接伺服舵机信号线
int myangle;//定义角度变量
int pulsewidth;//定义脉宽变量
int val;
void servopulse(int servopin,int myangle)//定义一个脉冲函数
{
pulsewidth=(myangle*11)+500;//将角度转化为500-2480 的脉宽值
digitalWrite(servopin,HIGH);//将舵机接口电平至高
delayMicroseconds(pulsewidth);//延时脉宽值的微秒数
digitalWrite(servopin,LOW);//将舵机接口电平至低
delay(20-pulsewidth/1000);
}
void setup()
{
pinMode(servopin, OUTPUT);//设定舵机接口为输出接口
Serial.begin(9600);//连接到串行端口，波特率为9600
```

```
Serial.println("servo=o_seral_simple ready" );
}
void loop()//将0 到9 的数转化为0 到180 角度，并让LED 闪烁相应数的次数
{
val=Serial.read();//读取串行端口的值
if(val>'0' &&val<='9')
{
val=val-'0' ;//将特征量转化为数值变量
val=val*(180/9) ;//将数字转化为角度
Serial.print("moving servo to ");
Serial.print(val, DEC);
Serial.println();
for(int i=0;i<=50;i++) //给予舵机足够的时间让它转到指定角度
{
servopulse(servopin, val);//引用脉冲函数
}
}
}
```

#### 方法二

先具体分析一下 Arduino 自带的Servo 函数及其语句，来介绍一下舵机函数的几个常用语句吧。

- 1、attach（接口）——设定舵机的接口，只有数字9 或10 接口可利用。
- 2、write（角度）——用于设定舵机旋转角度的语句，可设定的角度范围是0° 到180° 。
- 3、read（）——用于读取舵机角度的语句，可理解为读取最后一条write() 命令中的值。
- 4、attached（）——判断舵机参数是否已发送到舵机所在接口。
- 5、detach（）——使舵机与其接口分离，该接口（数字9 或10 接口）可继续被用作PWM 接口。

注：以上语句的书写格式均为“舵机变量名.具体语句（）”例如：myservo.attach(9)。  
仍然将舵机接在数字9 接口上即可。

参考源程序B：

```
#include <Servo.h>//定义头文件，这里有一点要注意，可以直接在Arduino 软件菜单栏单击Sketch>Importlibrary>Servo,调用Servo 函数，也可以直接输入#include <Servo.h>，但是在输入时要注意在#include 与<Servo.h>之间要有空格，否则编译时会报错。
Servo myservo;//定义舵机变量名
void setup()
{
myservo.attach(9);//定义舵机接口（9、10 都可以，缺点只能控制2 个）
}
void loop()
{
myservo.write(90);//设置舵机旋转的角度
}
```

以上就是控制舵机的两种方法，各有优缺点大家根据自己的喜好和需要进行选择。



## 例程 19 红外遥控

### 1、红外接收头介绍

#### 一、什么是红外接收头？

红外遥控器发出的信号是一连串的二进制脉冲码。为了使其在无线传输过程中免受其他红外信号的干扰,通常都是先将其调制在特定的载波频率上,然后再经红外发射二极管发射出去,而红外线接收装置则要滤除其他杂波,另接收该特定频率的信号并将其还原成二进制脉冲码,也就是解调。

#### 二、工作原理

内置接收管将红外发射管发射出来光信号转换为微弱的电信号,此信号经由IC内部放大器进行放大,然后通过自动增益控制、带通滤波、解调变、波形整形后还原为遥控器发射出的原始编码,经由接收头的信号输出脚输入到电器上的编码识别电路。

#### 三、红外接收头的引脚与连线



红外接收头有三个引脚如下图:

用的时候将VOUT接到模拟口, GND接到实验板上的GND, VCC接到实验板上的+5v。

### 红外遥控实验

#### 1、实验器件

- 红外遥控器: 1个
- 红外接收头: 1个
- LED 灯: 6 个
- 220  $\Omega$  电阻: 6个
- 多彩面包线: 若干

#### 2、实验连线

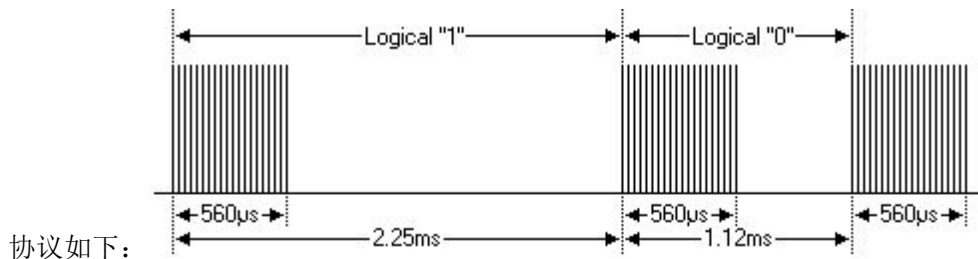
首先将板子连接好;接着将红外接收头按照上述方法接好,将VOUT接到数字11口引脚,将LED灯通过电阻接到数字引脚2, 3, 4, 5, 6, 7。返样就完成了电路部分的连接。

#### 3、实验原理

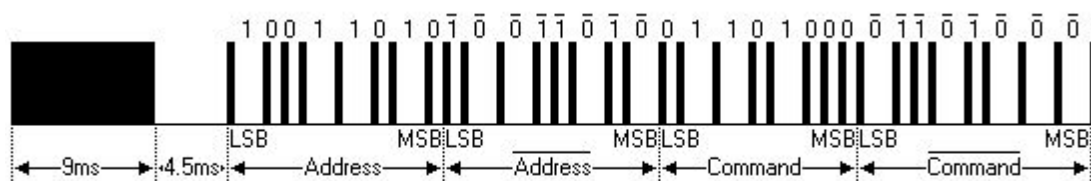
要想对某一遥控器进行解码必须要了解该遥控器的编码方式。本产品使用的控器的码方式为: NEC协议。下面就介绍一下NEC协议:

- NEC协议介绍: 特点: (1) 8位地址位, 8位命令位
- (2) 为了可靠性地址位和命令位被传输两次

- (3) 脉冲位置调制
- (4) 载波频率38khz
- (5) 每一位乂时间为1.125ms戒2.25ms
- 逻辑 0和1的定义如下图

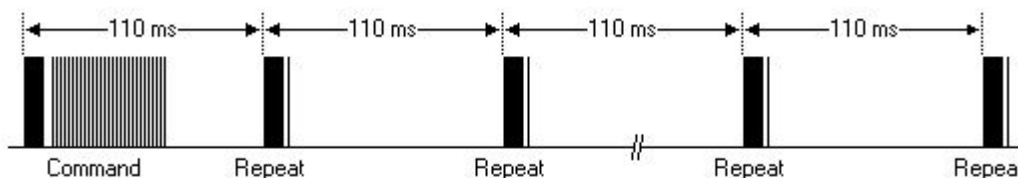


- 按键按下立刻松开的发射脉冲：



上面图片显示了NEC的协议典型的脉冲序列。注意：这首先发送LSB（最低位）的协议。在上面乂脉冲传输的地址为0x59命令为0x16。一个消息是由一个9ms的高电平开始，随后有一个4.5ms的低电平，（返两段电平组成寻码）然后由地址码和命令码。地址和命令传输两次。第二次所有位都取反，可用于对所收到的消息中的确认使用。总传输时间是恒定的，因为每一点与它取反长度重复。如果你不感兴趣，你可以忽略这个可靠性取反，也可以扩大地址和命令，以每16位！

按键按下一段时间才松开的发射脉冲：



一个命令发送一次，即使在遥控器上的按键仍然按下。当按键一直按下时，第一个110ms乂脉冲与上图一样，之后每110ms重复代码传输一次。返个重复代码是由一个9ms的高电平脉冲和一个2.25ms低电平和560µs乂高电平组成。

- 重复脉冲



注意：脉冲波形进入一体化接收头以后，因为一体化接收头里要迓解码、信号放大和整形，故要注意：在没有红外信号时，其输出端为高电平，有信号时为低电平，故其输出信号电平正好和发射端相反。接收端脉冲大家可以通过示波器看到，结合看到的波形理解程序。

遥控器键值:

一排一 = 0x00FFA25D; 一排二 = 0x00FFE01F; 一排三 = 0x00FF629D;  
二排一 = 0x00FFA857; 二排二 = 0x00FFE21D; 二排三 = 0x00FF906F;  
三排一 = 0x00FF22DD; 三排二 = 0x00FF6897; 三排三 = 0x00FF02FD;  
四排一 = 0x00FF9867; 四排二 = 0x00FFC23D; 四排三 = 0x00FFB047;

#### 程序代码

```
#include <IRremote.h>
int RECV_PIN = 11;
int LED1 = 2;
int LED2 = 3;
int LED3 = 4;
int LED4 = 5;
int LED5 = 6;
int LED6 = 7;
long on1 = 0x00FFA25D;
long off1 = 0x00FFE01F;
long on2 = 0x00FF629D;
long off2 = 0x00FFA857;
long on3 = 0x00FFE21D;
long off3 = 0x00FF906F;
long on4 = 0x00FF22DD;
long off4 = 0x00FF6897;
long on5 = 0x00FF02FD;
long off5 = 0x00FF9867;
long on6 = 0x00FFC23D;
long off6 = 0x00FFB047;
IRrecv irrecv(RECV_PIN);
decode_results results;
// Dumps out the decode_results structure.
// Call this after IRrecv::decode()
// void * to work around compiler issue
//void dump(void *v) {
//  decode_results *results = (decode_results *)v
void dump(decode_results *results) {
  int count = results->rawlen;
  if (results->decode_type == UNKNOWN)
  {
    Serial.println("Could not decode message");
  }
  else
  {
    if (results->decode_type == NEC)
    {
```

```
        Serial.print("Decoded NEC: ");
    }
    else if (results->decode_type == SONY)
    {
        Serial.print("Decoded SONY: ");
    }
    else if (results->decode_type == RC5)
    {
        Serial.print("Decoded RC5: ");
    }
    else if (results->decode_type == RC6)
    {
        Serial.print("Decoded RC6: ");
    }
    Serial.print(results->value, HEX);
    Serial.print(" ");
    Serial.print(results->bits, DEC);
    Serial.println(" bits");
}
Serial.print("Raw ");
Serial.print(count, DEC);
Serial.print("): ");

for (int i = 0; i < count; i++)
{
    if ((i % 2) == 1) {
        Serial.print(results->rawbuf[i]*USECPERTICK, DEC);
    }
    else
    {
        Serial.print(-(int)results->rawbuf[i]*USECPERTICK, DEC);
    }
    Serial.print(" ");
}
Serial.println("");
}

void setup()
{
    pinMode(RECV_PIN, INPUT);
    pinMode(LED1, OUTPUT);
    pinMode(LED2, OUTPUT);
    pinMode(LED3, OUTPUT);
    pinMode(LED4, OUTPUT);
}
```

```
pinMode(LED5, OUTPUT);
pinMode(LED6, OUTPUT);
pinMode(13, OUTPUT);
Serial.begin(9600);

irrecv.enableIRIn(); // Start the receiver
}

int on = 0;
unsigned long last = millis();

void loop()
{
  if (irrecv.decode(&results))
  {
    // If it's been at least 1/4 second since the last
    // IR received, toggle the relay
    if (millis() - last > 250)
    {
      on = !on;
      //      digitalWrite(8, on ? HIGH : LOW);
      digitalWrite(13, on ? HIGH : LOW);
      dump(&results);
    }
    if (results.value == on1 )
      digitalWrite(LED1, HIGH);
    if (results.value == off1 )
      digitalWrite(LED1, LOW);
    if (results.value == on2 )
      digitalWrite(LED2, HIGH);
    if (results.value == off2 )
      digitalWrite(LED2, LOW);
    if (results.value == on3 )
      digitalWrite(LED3, HIGH);
    if (results.value == off3 )
      digitalWrite(LED3, LOW);
    if (results.value == on4 )
      digitalWrite(LED4, HIGH);
    if (results.value == off4 )
      digitalWrite(LED4, LOW);
    if (results.value == on5 )
      digitalWrite(LED5, HIGH);
    if (results.value == off5 )
      digitalWrite(LED5, LOW);
  }
}
```

```

    if (results.value == on6 )
        digitalWrite(LED6, HIGH);
    if (results.value == off6 )
        digitalWrite(LED6, LOW);
    last = millis();
    irrecv.resume(); // Receive the next value
}
}

```

## 五、程序功能

对遥控器发射出来的编码脉冲进行解码，根据解码结果执行相应的动作。返样大家就可以用遥控器遥控你的器件了，让它听你的指挥。

## 例程 20 1602 液晶实验

本次试验使用 arduino 直接驱动 1602 液晶显示文字

1602 液晶在应用中非常广泛，最初的 1602 液晶使用的是 HD44780 控制器，现在各个厂家的 1602 模块基本上都是采用了与之兼容的 IC，所以特性上基本都是一致的。

1602LCD 主要技术参数

显示容量为 16×2 个字符；

芯片工作电压为 4.5~5.5V；

工作电流为 2.0mA (5.0V)；

模块最佳工作电压为 5.0V；

字符尺寸为 2.95×4.35 (W×H) mm。

1602 液晶接口引脚定义

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

From [www.geek-workshop.com](http://www.geek-workshop.com)



- 接口说明：
- 1、两组电源 一组是模块的电源 一组是背光板的电源 一般均使用 5V 供电。本次试验背光使用 3.3V 供电也可以工作。
  - 2、VL 是调节对比度的引脚，串联不大于 5KΩ 的电位器进行调节。本次实验使用 1KΩ 的电阻来设定对比度。其连接分高电位与低电位接法，本次使用低电位接法，串联 1KΩ 电阻后接 GND。
  - 3、RS 是很多液晶上都有的引脚 是命令/数据选择引脚 该脚电平为高时表示将进行数据操作；为低时表示进行命令操作。
  - 4、RW 也是很多液晶上都有的引脚 是读写选择端 该脚电平为高是表示要对液晶进行读操作；为低时表示要进行写操作。
  - 5、E 同样很多液晶模块有此引脚 通常在总线上信号稳定后给一正脉冲通知把数据读走，在此脚为高电平的时候总线不允许变化。
  - 6、D0—D7 8 位双向并行总线，用来传送命令和数据。
  - 7、BLA 是背光源正极，BLK 是背光源负极。

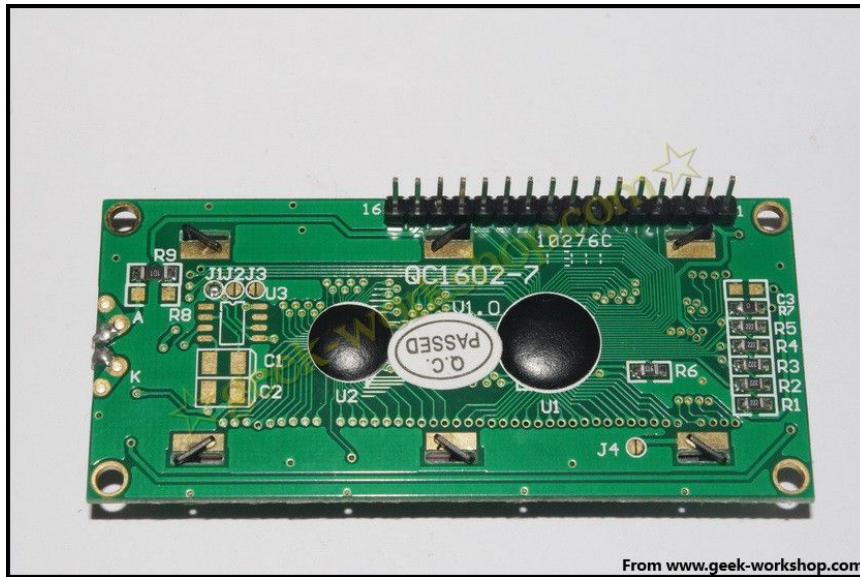
1602 液晶的基本操作分以下四种：

读状态	输入	RS=L, R/W=H, E=H	输出	D0~D7=状态字
写指令	输入	RS=L, R/W=L, D0~D7=指令码, E=高脉冲	输出	无
读数据	输入	RS=H, R/W=H, E=H	输出	D0~D7=数据
写数据	输入	RS=H, R/W=L, D0~D7=数据, E=高脉冲	输出	无

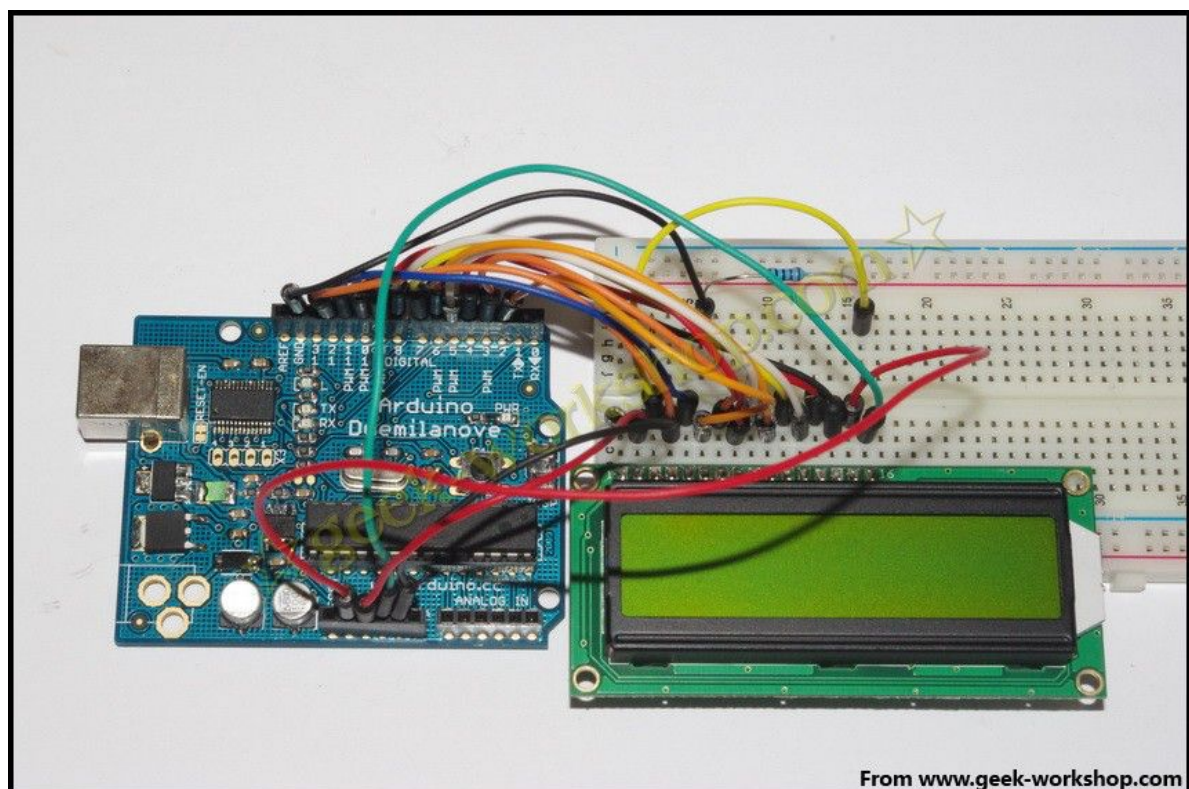
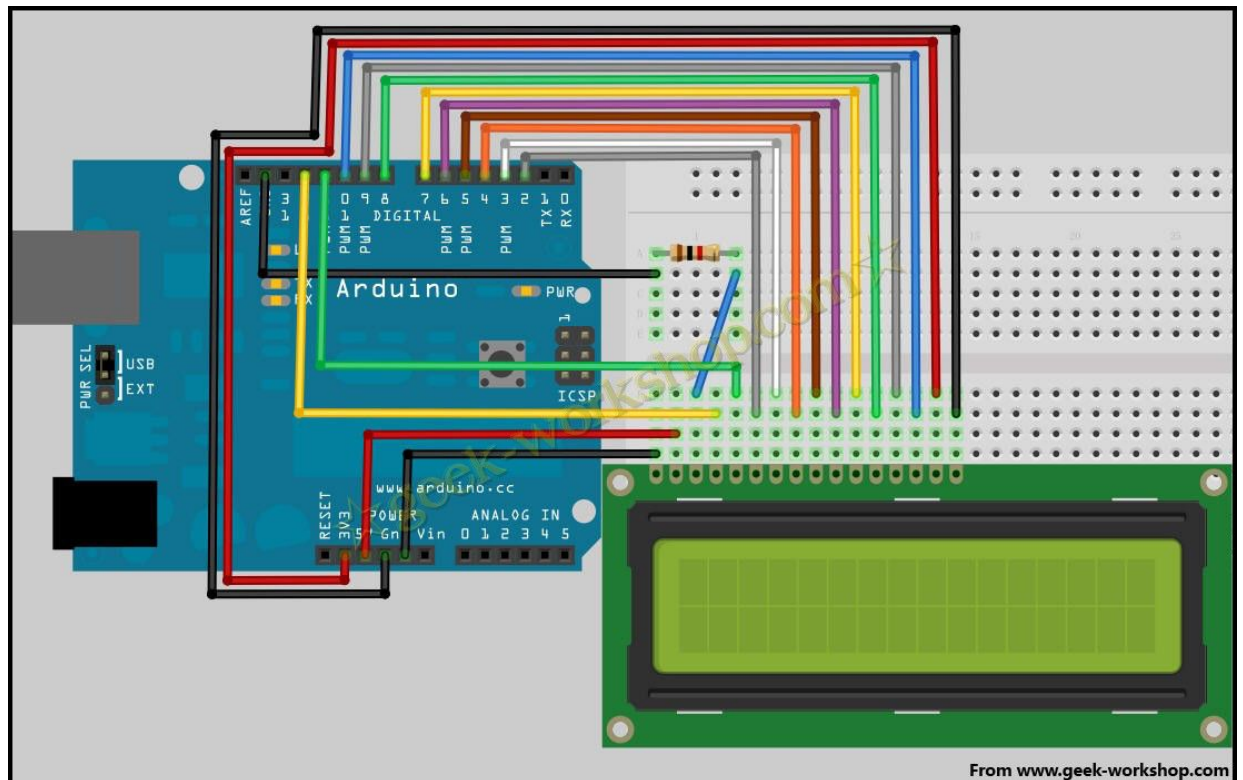
From www.geek-workshop.com

下图就是 1602 液晶实物图





1602 直接与 arduino 通信，根据产品手册描述，分 8 位连接法与 4 位连接法，咱们先使用 8 位连接法进行实验。硬件连接方式如下图



Arduino 确实不是省油的灯，早就帮你准备好了一切，1602 液晶有专门的函数库，即 LiquidCrystal，这个函数库相关资讯，可以从官网了解到，<http://arduino.cc/en/Tutorial/HomePage>。

LiquidCrystal 函数库针对 1602 液晶的数据传送有两种模式，一种是 8bit 模式，一种是 4bit 模式。8bit 的传送速度快，是因为显示的字符都是 ASCII 码，ASCII 码是 8 位二进制数组成，所以 8bit 刚好一次就把字符的二进制码一次传完，而 4bit 则是需要将字符拆成两半，一次只传送 4bit，两倍时间才可以把数据传完，不过 4bit 模式的好处是需要的数据引脚少了一半，方便硬件连线。

8bit 模式需要 D0~D7 引脚，4bit 只需后四个引脚 D4~D7。不管是哪种模式控制引脚都有 3 个，分别为：RS、RW、Enable。

4bit 模式的 LiquidCrystal 申明函数为：LiquidCrystal(RS, RW, Enable, D4, D5, D6, D7);

8bit 模式的 LiquidCrystal 申明函数为：LiquidCrystal(RS, RW, Enable, D0, D1, D2, D3, D4, D5, D6, D7);

这篇文章介绍的仿真项目中 1602 液晶显示采用的是 8bit 模式，用到的 3 个控制引脚和 8 个数据引脚，如下图所示，它们分别连到了 Arduino 单片机 11 个数字端口上，根据 8bit 模式的 LiquidCrystal 申明函数的参数格式和 Arduino 单片机与 1602 液晶的引脚连线关系，1602 液晶引脚与 Arduino 数字端口对应关系，可申明为：LiquidCrystal lcd(12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2)。

代码如下

```
#include <LiquidCrystal.h> //申明 1602 液晶的函数库
//申明 1602 液晶的 11 个引脚所连接的 Arduino 数字端口
LiquidCrystal lcd(12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2);

void setup()
{
    lcd.begin(16, 2);          //初始化 1602 液晶工作模式
                                //定义 1602 液晶显示范围为 2 行 16 列字符
    lcd.setCursor(0, 0);      //把光标定位在第 0 行，第 0 列
    lcd.print("Hello World"); //显示
    lcd.setCursor(0, 2);      //把光标定位在第 0 行，第 15 列
    lcd.print("Arduino is fun"); //显示
}

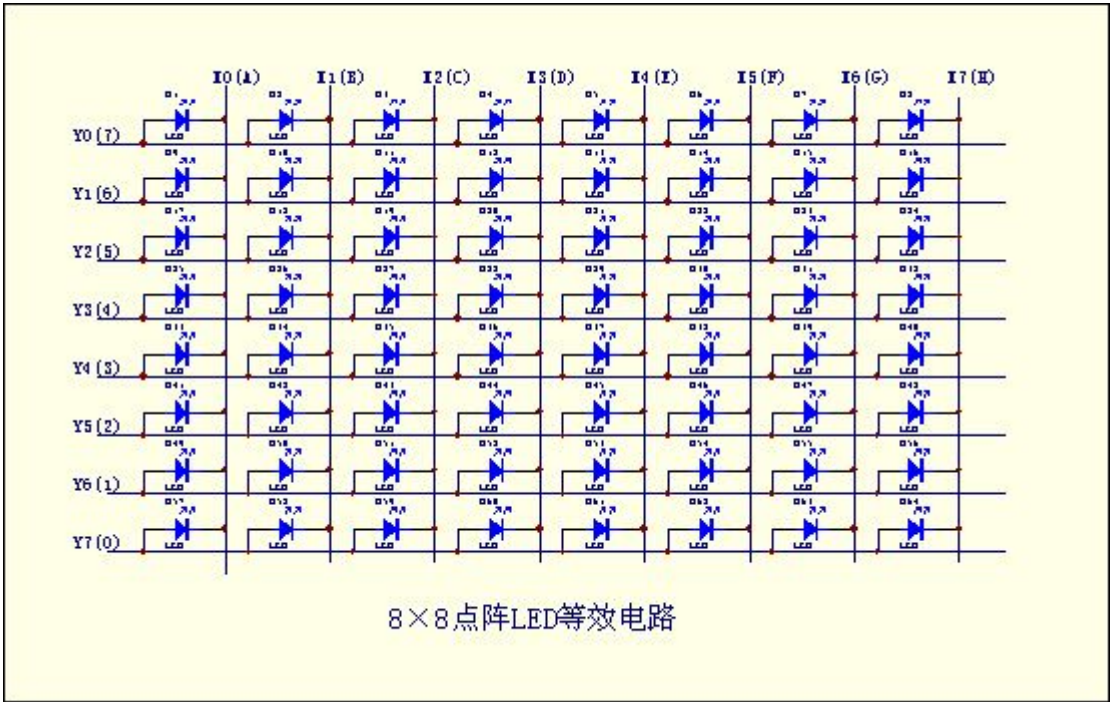
void loop()
{
}
```

## 例程 21 8x8 点阵实验

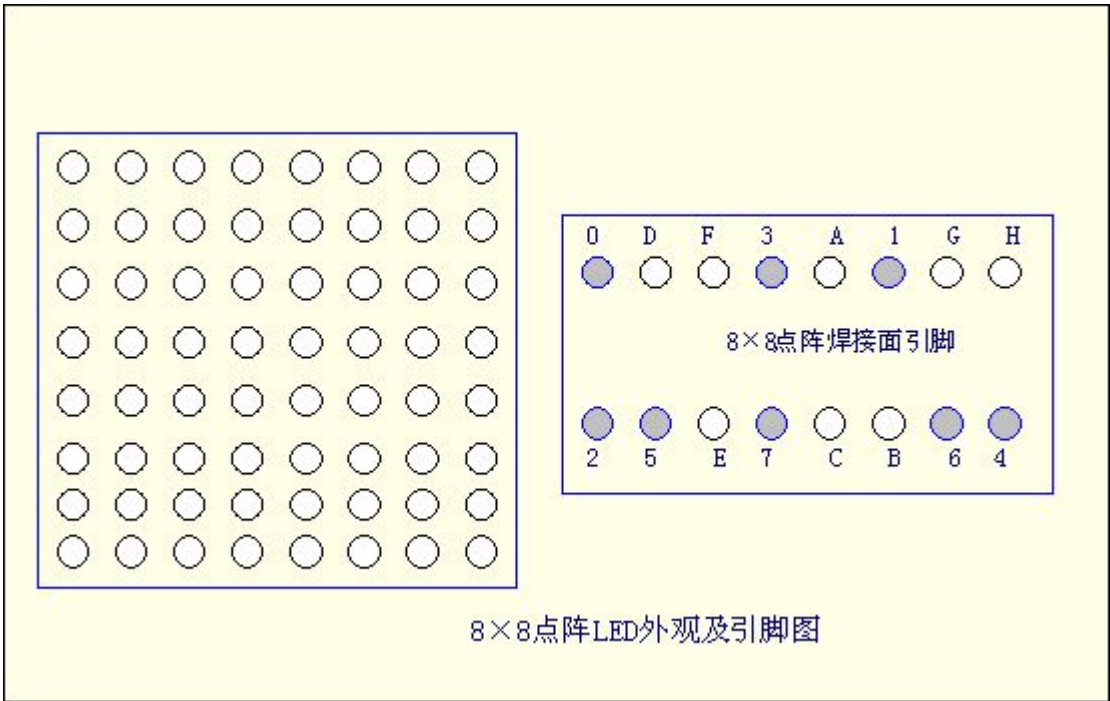


概述

1. 8\*8 点阵原理图



2. 8\*8 点阵实物图



图为8×8 点阵 LED 外观及引脚图，其等效电路如图（2）所示，只要其对应的 X、Y 轴顺向偏压，即可使 LED 发亮。例如如果想使左上角 LED 点亮，则 Y0=1，X0=0 即可。应用时

限流电阻可以放在 X 轴或 Y 轴

### 3. 8\*8 点阵扫描方式

LED 一般采用扫描式显示，实际运用分为三种方式

(1) 点扫描

(2) 行列扫描

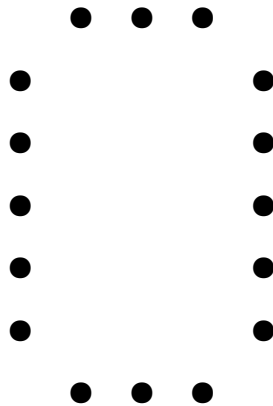
$16 \times 64 = 1024\text{Hz}$ ，周期小于 1ms 即可。若使用第二和第三种方式，则频率必须大于  $16 \times 8 = 128\text{Hz}$ ，周期小于 7.8ms 即可符合视觉暂留要求。此外一次驱动一列或一行（8 颗 LED）时需外加驱动电路提高电流，否则 LED 亮度会不足。

### 3. 8\*8 点阵应用举例

点阵内部结构及外形如下，8X8 点阵共由 64 个发光二极管组成，且每个发光二极管是放置在行线和列线的交叉点上，当对应的某一行置 1 电平，某一列置 0 电平，则相应的二极管就亮；如要将第一个点点亮，则 9 脚接高电平 13 脚接低电平，则第一个点就亮了；如要将第一行点亮，则第 9 脚要接高电平，而（13、3、4、10、6、11、15、16）这些引脚接低电平，那么第一行就会点亮；如要将第一列点亮，则第 13 脚接低电平，而（9、14、8、12、1、7、2、5）接高电平，那么第一列就会点亮。

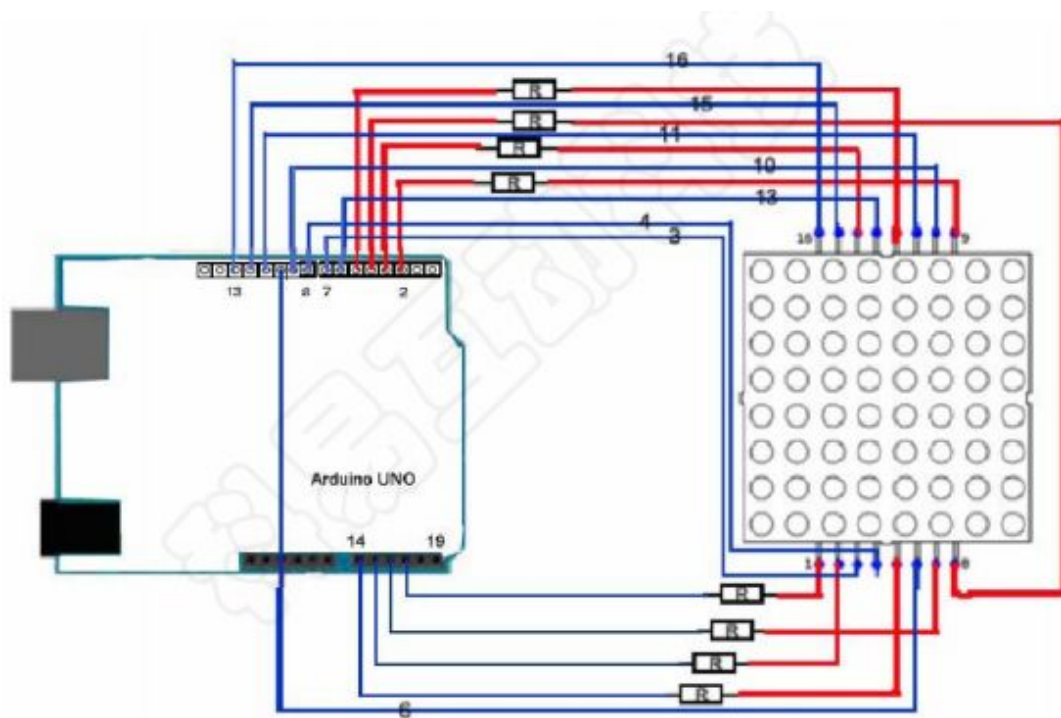
一般我们使用点阵显示汉字是用的 16\*16 的点阵宋体字库，所谓 16\*16，是每一个汉字在纵、横各 16 点的区域内显示的。也就是说得用四个 8\*8 点阵组合成一个 16\*16 的点阵。如下图所示，要显示“你”则相应的点就要点亮，由于我们的点阵在列线上是低电平有效，而在行线上是高电平有效，所以要显示“你”字的话，它的位代码信息要取反，即所有列（13~16 脚）送(1111011101111111, 0xF7, 0x7F)，而第一行（9 脚）送 1 信号，然后第一行送 0。再送第二行要显示的数据（13~16 脚）送(1111011101111111, 0xF7, 0x7F)，而第二行（14 脚）送 1 信号。依此类推，只要每行数据显示时间间隔够短，利用人眼的视觉暂停作用，这样送 16 次数据扫描完 16 行后就会看到一个“你”字；第二种送数据的方法是字模信号送到行线上再扫描列线也是同样的道理。同样以“你”字来说明，16 行（9、14、8、12、1、7、2、5）上送(0000000000000000, 0x00, 0x00)而第一列（13 脚）送、“0”。同理扫描第二列。当行线上送了 16 次数据而列线扫描了 16 次后一个“你”字也就显示出来了。



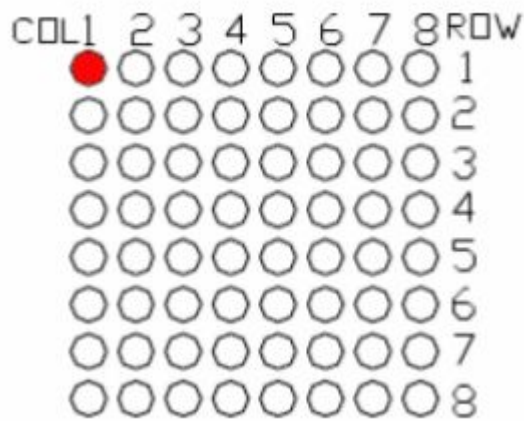


因此，形成的列代码为 00H，00H，3EH，41H，41H，3EH，00H，00H；只要把这些代码分别依次送到相应的列线上面，即可实现“0”的数字显示。

本实验的连线图。



点亮 8X8 点阵 LED 的一个 LED 如下：



\*\*\*\*\*

实例代码：

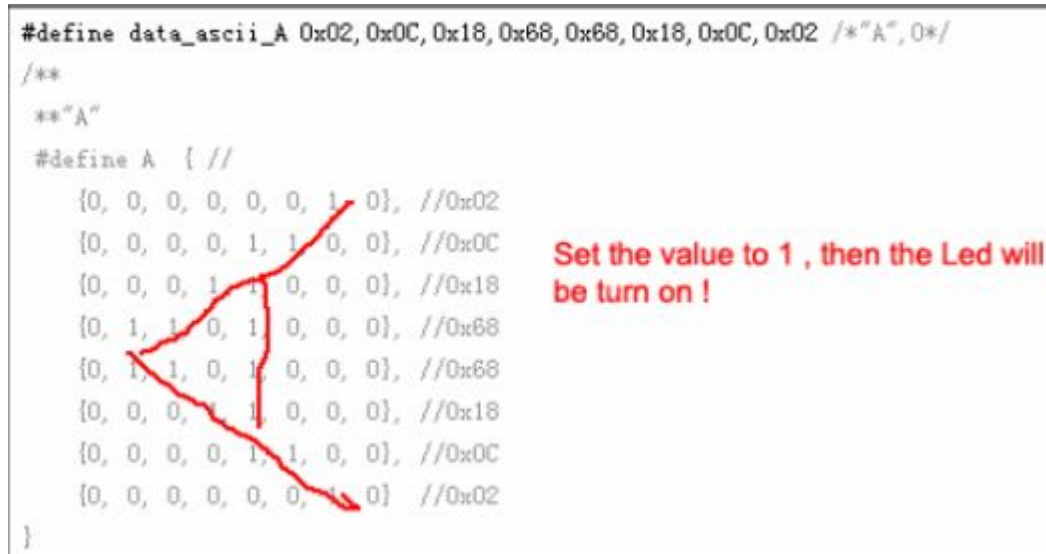
```
//the pin to control ROW
const int row1 = 2; // the number of the row pin 9
const int row2 = 3; // the number of the row pin 14
const int row3 = 4; // the number of the row pin 8
const int row4 = 5; // the number of the row pin 12
const int row5 = 17; // the number of the row pin 1
const int row6 = 16; // the number of the row pin 7
const int row7 = 15; // the number of the row pin 2
const int row8 = 14; // the number of the row pin 5
//the pin to control COL
const int col1 = 6; // the number of the col pin 13
const int col2 = 7; // the number of the col pin 3
const int col3 = 8; // the number of the col pin 4
const int col4 = 9; // the number of the col pin 10
const int col5 = 10; // the number of the col pin 6
const int col6 = 11; // the number of the col pin 11
const int col7 = 12; // the number of the col pin 15
const int col8 = 13; // the number of the col pin 16
void setup() {
  int i = 0 ;
  for(i=2;i<18;i++)
  {
    pinMode(i, OUTPUT);
  }
  pinMode(row5, OUTPUT);
  pinMode(row6, OUTPUT);
  pinMode(row7, OUTPUT);
  pinMode(row8, OUTPUT);
  for(i=2;i<18;i++) {
```

```
digitalWrite(i, LOW);
}
digitalWrite(row5, LOW);
digitalWrite(row6, LOW);
digitalWrite(row7, LOW);
digitalWrite(row8, LOW);
}
void loop() {
int i;
//the row # 1 and col # 1 of the LEDs turn on
digitalWrite(row1, HIGH);
digitalWrite(row2, LOW);
digitalWrite(row3, LOW);
digitalWrite(row4, LOW);
digitalWrite(row5, LOW);
digitalWrite(row6, LOW);
digitalWrite(row7, LOW);
digitalWrite(row8, LOW);
digitalWrite(col1, LOW);
digitalWrite(col2, HIGH);
digitalWrite(col3, HIGH);
digitalWrite(col4, HIGH);
digitalWrite(col5, HIGH);
digitalWrite(col6, HIGH);
digitalWrite(col7, HIGH);
digitalWrite(col8, HIGH);
delay(1000);
//turn off all
for(i=2;i<18;i++) {
digitalWrite(i, LOW);
}
delay(1000);
}
```

\*\*\*\*\*

另外的实验代码如下：

显示 A 这个字母， 则在点阵中的位置 置 1. 通过动态扫描显示 。



代码:

```

*****
****
#define display_array_size 8
// ascii 8x8 dot font
#define data_null 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 // null char
#define data_ascii_A 0x02, 0x0C, 0x18, 0x68, 0x68, 0x18, 0x0C, 0x02 /*"A", 0*/
/**
**"A"
#define A { //
    {0, 0, 0, 0, 0, 0, 1, 0}, //0x02
    {0, 0, 0, 0, 1, 1, 0, 0}, //0x0C
    {0, 0, 0, 1, 1, 0, 0, 0}, //0x18
    {0, 1, 1, 0, 1, 0, 0, 0}, //0x68
    {0, 1, 1, 0, 1, 0, 0, 0}, //0x68
    {0, 0, 0, 1, 1, 0, 0, 0}, //0x18
    {0, 0, 0, 0, 1, 1, 0, 0}, //0x0C
    {0, 0, 0, 0, 0, 0, 1, 0} //0x02
}
**/
#define data_ascii_B 0x00, 0x7E, 0x52, 0x52, 0x52, 0x52, 0x2C, 0x00 /*"B", 1*/
#define data_ascii_C 0x00, 0x3C, 0x66, 0x42, 0x42, 0x42, 0x2C, 0x00 /*"C", 2*/
#define data_ascii_D 0x00, 0x7E, 0x42, 0x42, 0x42, 0x66, 0x3C, 0x00 /*"D", 3*/
#define data_ascii_E 0x00, 0x7E, 0x52, 0x52, 0x52, 0x52, 0x52, 0x42 /*"E", 4*/
#define data_ascii_F 0x00, 0x7E, 0x50, 0x50, 0x50, 0x50, 0x50, 0x40 /*"F", 5*/
#define data_ascii_G 0x00, 0x3C, 0x66, 0x42, 0x42, 0x52, 0x16, 0x1E /*"G", 6*/
#define data_ascii_H 0x00, 0x7E, 0x10, 0x10, 0x10, 0x10, 0x7E, 0x00 /*"H", 7*/
#define data_ascii_I 0x00, 0x00, 0x00, 0x7E, 0x00, 0x00, 0x00, 0x00 /*"I", 8*/
// display array

```

```
byte data_ascii[][display_array_size] = {
    data_null,
    data_ascii_A, data_ascii_B,
    data_ascii_C,
    data_ascii_D,
    data_ascii_E,
    data_ascii_F,
    data_ascii_G,
    data_ascii_H,
    data_ascii_I,
};
//the pin to control ROW
const int row1 = 2; // the number of the row pin 24
const int row2 = 3; // the number of the row pin 23
const int row3 = 4; // the number of the row pin 22
const int row4 = 5; // the number of the row pin 21
const int row5 = 17; // the number of the row pin 4
const int row6 = 16; // the number of the row pin 3
const int row7 = 15; // the number of the row pin 2
const int row8 = 14; // the number of the row pin 1
//the pin to control COL
const int col1 = 6; // the number of the col pin 20
const int col2 = 7; // the number of the col pin 19
const int col3 = 8; // the number of the col pin 18
const int col4 = 9; // the number of the col pin 17
const int col5 = 10; // the number of the col pin 16
const int col6 = 11; // the number of the col pin 15
const int col7 = 12; // the number of the col pin 14
const int col8 = 13; // the number of the col pin 13

void displayNum(byte rowNum,int colNum)
{
    int j;
    byte temp = rowNum;
    for(j=2;j<6;j++)
    {
        digitalWrite(j, LOW);
    }
    digitalWrite(row5, LOW);
    digitalWrite(row6, LOW);
    digitalWrite(row7, LOW);
    digitalWrite(row8, LOW);
    for(j=6;j<14;j++)
    {
```

```
digitalWrite(j, HIGH); }
switch(colNum)
{
    case 1: digitalWrite(col1, LOW); break;
    case 2: digitalWrite(col2, LOW); break;
    case 3: digitalWrite(col3, LOW); break;
    case 4: digitalWrite(col4, LOW); break;
    case 5: digitalWrite(col5, LOW); break;
    case 6: digitalWrite(col6, LOW); break;
    case 7: digitalWrite(col7, LOW); break;
    case 8: digitalWrite(col8, LOW); break;
    default: break;
}
for(j = 1 ;j < 9; j++)
{
    temp = (0x80)&(temp) ;
    if(temp==0)
    {
        temp = rowNum<<j;
        continue;
    }
    switch(j)
    {
        case 1: digitalWrite(row1, HIGH); break;
        case 2: digitalWrite(row2, HIGH); break;
        case 3: digitalWrite(row3, HIGH); break;
        case 4: digitalWrite(row4, HIGH); break;
        case 5: digitalWrite(row5, HIGH); break;
        case 6: digitalWrite(row6, HIGH); break;
        case 7: digitalWrite(row7, HIGH); break;
        case 8: digitalWrite(row8, HIGH); break;
        default: break;
    }
    temp = rowNum<<j;
}
}

void setup() {
    int i = 0 ;
    for(i=2;i<18;i++)
    {
        pinMode(i, OUTPUT);
    }
}
```

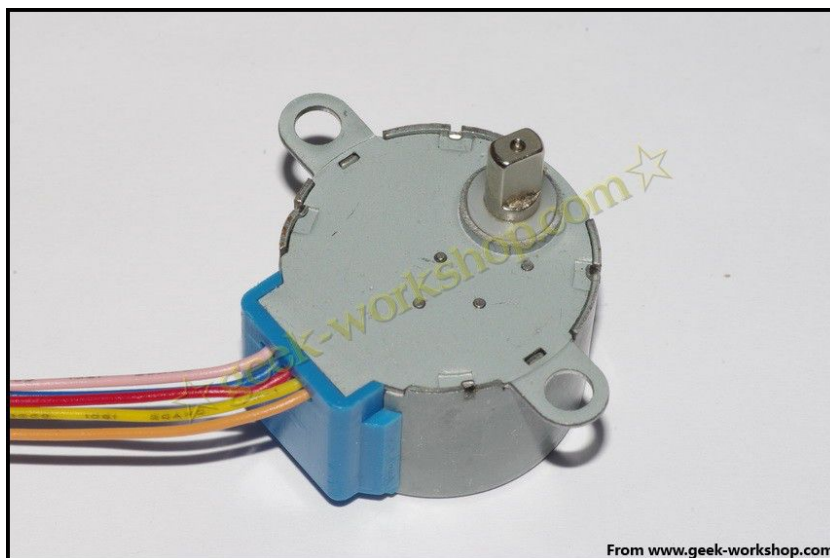


```
for(i=2;i<18;i++) {  
    digitalWrite(i, LOW);  
}  
}  
void loop() {  
    int t1;  
    int l;  
    int arrage;  
    for(arrage=0;arrage<10;arrage++)  
    {  
        for(l=0;l<512;l++)  
        {  
            for(t1=0;t1<8;t1++)  
            {  
                displayNum(data_ascii[arrage][t1], (t1+1));  
            }  
        }  
    }  
}
```

## 例程 22- 步进电机试验

步进电机是一种将电脉冲转化为角位移的执行机构。通俗一点讲：当步进驱动器接收到一个脉冲信号，它就驱动步进电机按设定的方向转动一个固定的角度（及步进角）。你可以通过控制脉冲个数来控制角位移量，从而达到准确定位的目的；同时你也可以通过控制脉冲频率来控制电机转动的速度和加速度，从而达到调速的目的。

下面这个就是本次实验使用的步进电机



使用步进电机前一定要仔细查看说明书，确认是四相还是两相，各个线怎样连接，本次实验

使用的步进电机是四相的，不同颜色的线定义如下图：

驱动方式：〈4-1-2相驱动〉								
导线颜色	1	2	3	4	5	6	7	8
5 红	+	+	+	+	+	+	+	+
4 橙	-	-						-
3 黄		-	-	-				
2 粉				-	-	-		
1 蓝						-	-	-

→ CCW 方向旋转（轴伸端视）

From www.geek-workshop.com

减速步进电机

直径：28mm

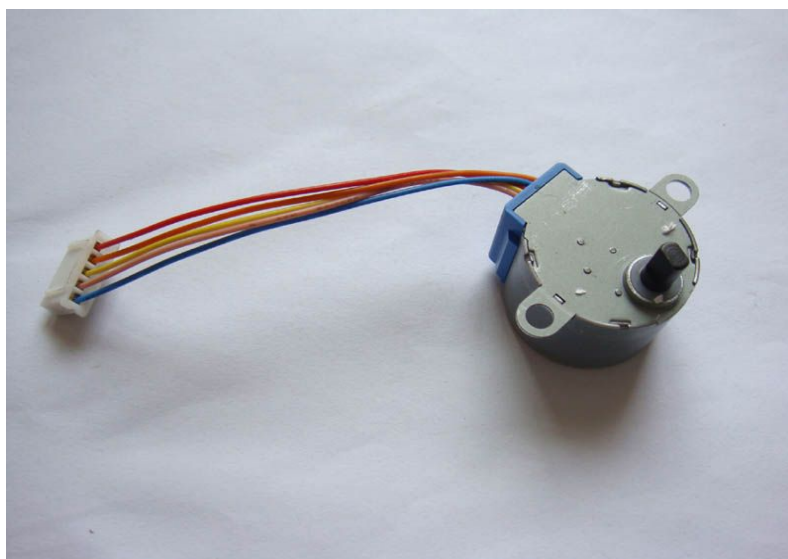
电压：5V

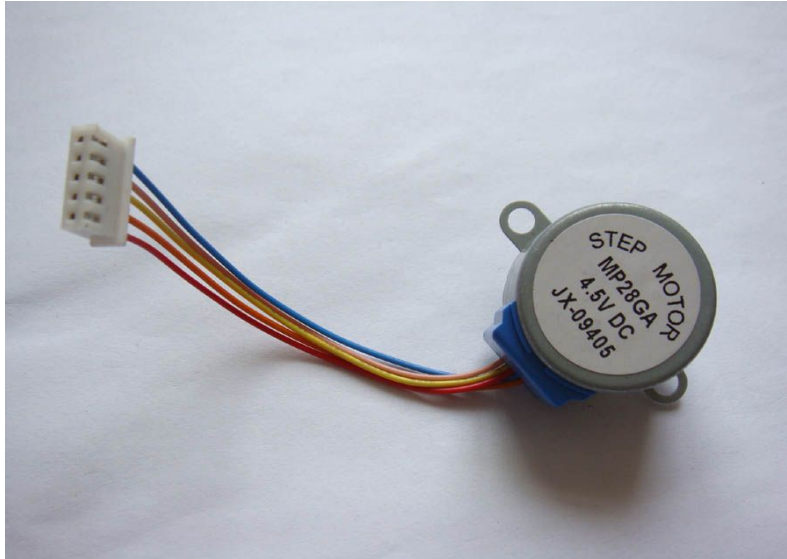
步进角度：5.625 x 1/64

减速比：1/64

5 线 4 相 可以用普通 uln2003 芯片驱动，也可以接成 2 相使用

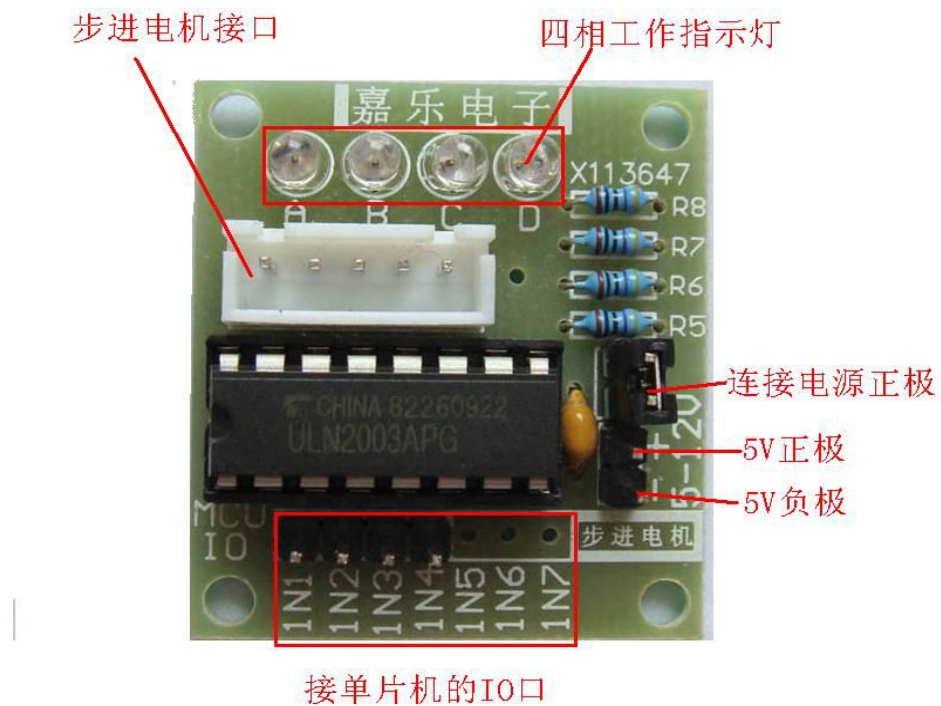
该步进电机空载耗电在 50mA 以下，带 64 倍减速器，输出力矩比较大，可以驱动重负载，极适合开发板使用。注意：此款步进电机带有 64 倍减速器，与不带减速器的步进电机相比，转速显得较慢，为方便观察，可在输出轴处粘上一片小纸板。





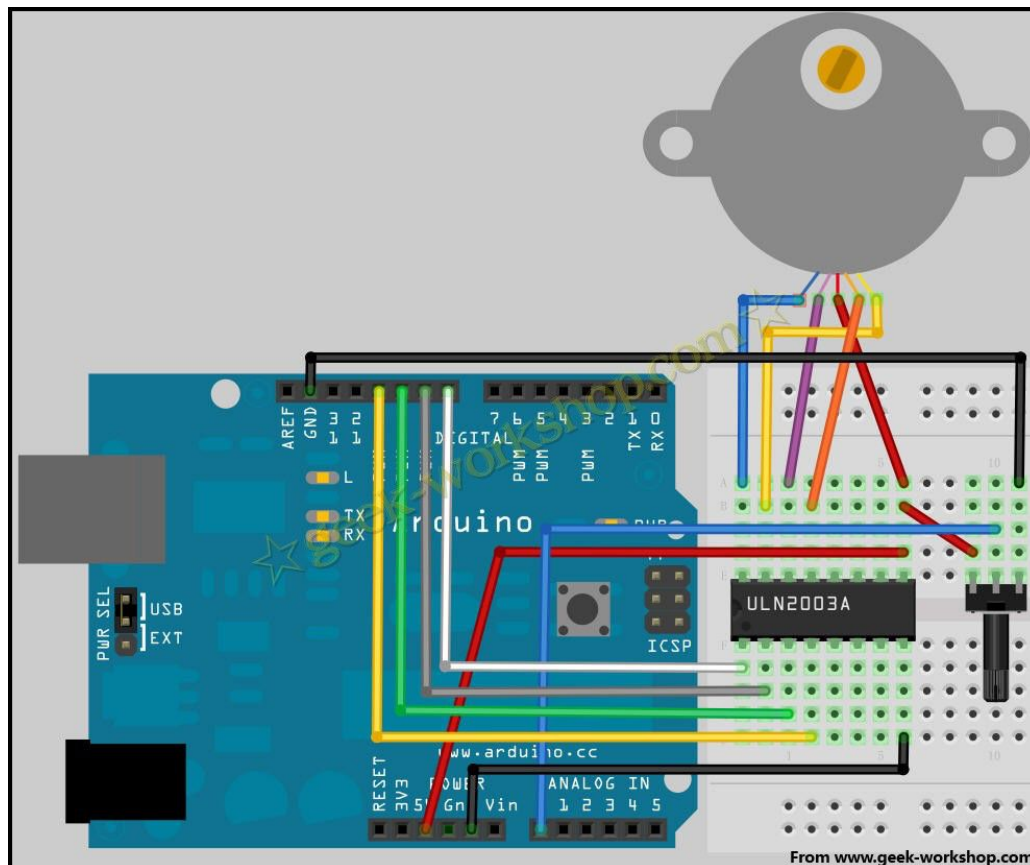
步进电机(五线四相) 驱动板(UL2003) 试验板

步进电机驱动板(UL2003) 试验板



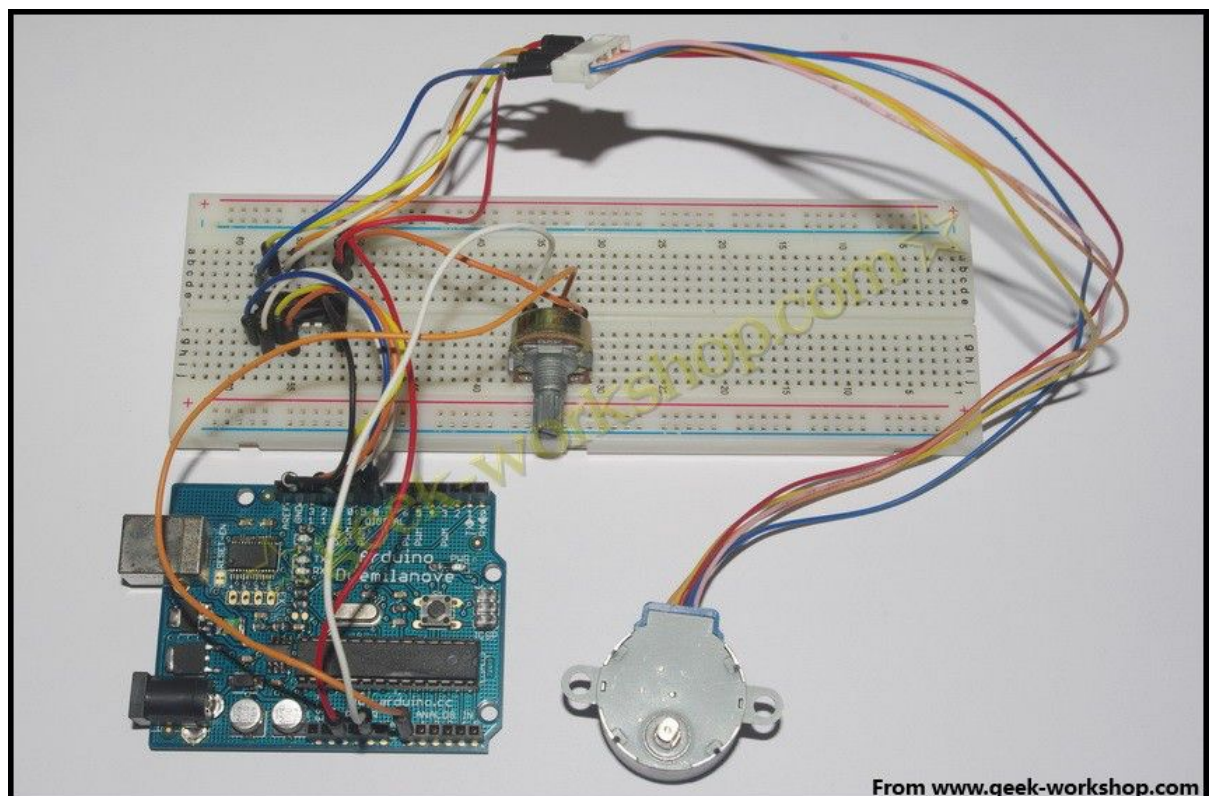
外形尺寸: 31×35mm

硬件连接图如下



2011-9-5 23:06:50 上传

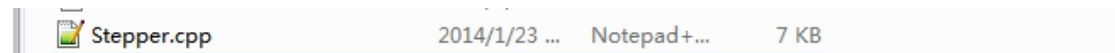
下载附件 (126.87 KB)



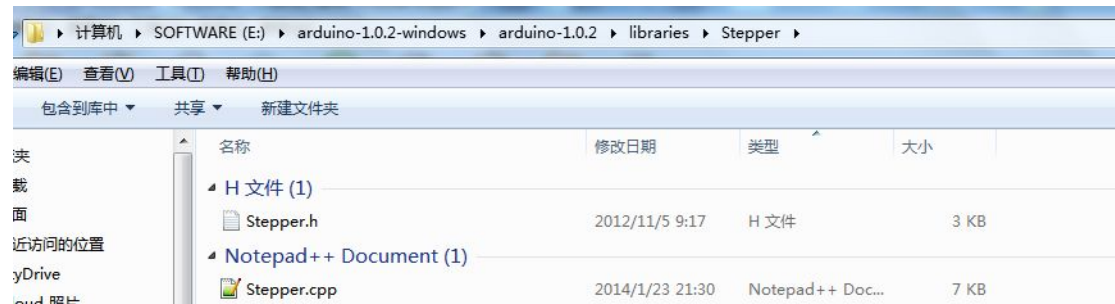


因为我们用的电机和 arduino 库中使用的不太一样，所以我们这个实验需要替换 arduino 的库！

找到我们教程里带的 stepper.cpp 文件



然后复制黏贴到 arduino 安装文件夹的 libraries/Stepper 文件夹下替换



把代码下载到 arduino 控制板中看看效果

```
/*
 * 步进电机跟随电位器旋转
 * (或者其他传感器)使用0号模拟口输入
 * 使用arduino IDE自带的Stepper.h库文件
 */
#include <Stepper.h>
#define STEPS 100 // 这里设置步进电机旋转一圈是多少步

Stepper stepper(STEPS, 8, 9, 10, 11); // attached to设置步进电机的步数和引脚
int previous = 0; // 定义变量用来存储历史读数

void setup()
{
    stepper.setSpeed(90); // 设置电机每分钟的转速为90步
}

void loop()
{
    int val = analogRead(0); // 获取传感器读数
    delay(200);
    if(abs(analogRead(0)-val)>=5)
    {
        stepper.step(val - previous); // 移动步数为当前读数减去历史读数
        previous = val; // 保存历史读数
    }
}
```