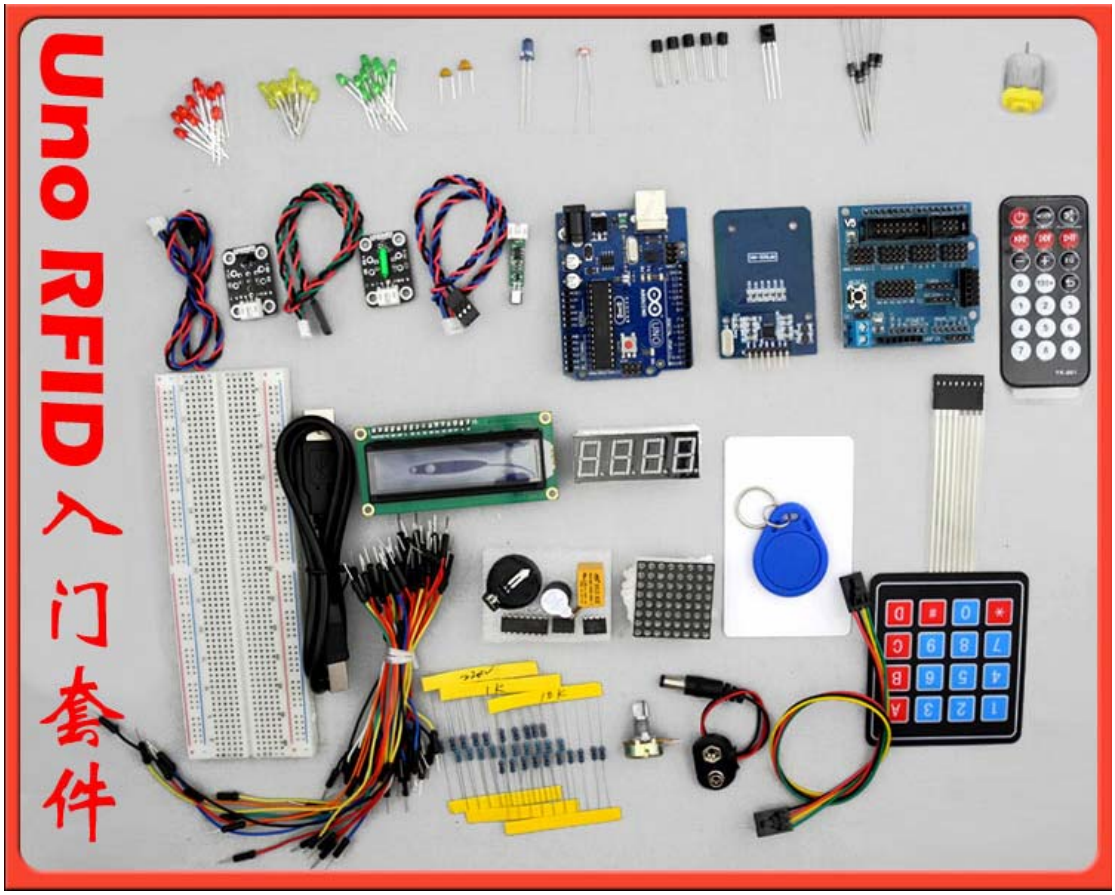
# Arduino UNO RFID 入门到精通套件




Arduino UNO RFID 教程 V1.0 版
2012 年 6 月 29 日星期五



腾博科技有限公司 http://shop68507353.taobao.com/

# 目录

# Arduino介绍篇

## 概 述

### 什么是 Arduino？

Arduino 是一块基于开放原始代码的 Simple i/o 平台，并且具有开发语言和开发环境都很简单、易理解的特点。让您可以快速使用 Arduino 做出有趣的东西。

它是一个能够用来感应和控制现实物理世界的一套工具。 它由一个基于单片机并且开放源码的硬件平台，和一套为 Arduino 板编写程序 的开发环境组成。

Arduino 可以用来开发交互产品，比如它可以读取大量的开关和传感器信号，并且可以控制各式各样的电灯、电机和其他物理设备。Arduino 项目可以是单独的，也可以在运行时和你电脑中运行的程序（例如：Flash，Processing，MaxMSP）进行通讯。Arduino 开源的 IDE 可以免费下载得到。

### 特色描述

- 开放原始码的电路图设计，开发界面免费下载，也可依需求自己修改!!
- 下载程序简单、方便。
- 可简单地与传感器、各式各样的电子元件连接（如：LED 灯、蜂鸣器、按键、光敏电阻等等），做出各种各样有趣的东西。
- 使用高速的微处理控制器(ATMEGA328)。
- 开发语言和开发环境都非常的简单、易理解，非常适合初学者学习。

### 性能描述

- Digital I/O 数字输入/输出端口 0—13。
- Analog I/O 模拟输入/输出端口 0-5。
- 支持 ISP 下载功能。
- 输入电压：接上 USB 时无须外部供电或外部 5V~9V 直流电压输入。
- 输出电压：5V 直流电压输出和 3.3V 直流电压输出和外部电源输入。
- 采用 Atmel Atmega328 微处理控制器。因其支持者众多，已有公司开发出来 32 位的 MCU 平台支持 arduino。
- Arduino 大小尺寸：宽 70mm X 高 54mm。

### 几个比较特殊的端口说明：

**VIN 端口：**VIN 是 input voltage 的缩写，表示有外部电源时的输入端口。如果不使用 USB 供电时，外接电源可以通过此引脚提供电压。（如电池供电，电池正构接 VIN 端口，负构接 GND 端口）。

**AREF:** Reference voltage for the analog inputs (模拟输入的基准电压)。使用 analogReference() 命令调用。

**ICSP：** 也有称为 ISP（In System Programmer)，就是一种线上即时烧录，目前比较新的芯片都支持这种烧录模式，包括大家常听说的 8051 系列的芯片，也都慢慢采用这种简便的烧录方式。我们都知道传统的烧录方式，都是将被烧录的芯片，从线路板上拔起，有的焊死在线路板上的芯片，还得先把芯片焊接下来才能烧录。为了解决这种问题，发明了 ICSP 线上即时烧录方式。只需要准备一条 R232 线（连接烧录器），以及一条连接烧录器与烧录芯片针脚的连接线就可以。电源的+5V，GND，两条负责传输烧录信息的针脚，再加上一个烧录电压针脚，这样就可以烧录了。

# Arduino C语法介绍

Arduino 语法是建立在 C/C++基础上的，其实也就是基础的 C 语法，Arduino 语法只不过把相关的一些参数设置都函数化，不用我们去了解他的底层，让我们去了解 AVR 单片机（微控制器）的朋友也能轻松上手。那么这里我就简单的注释一下 Arduino 语法。

## 关键字：

- **if**
- **if...else**
- **for**
- **switch case**
- **while**
- **do... while**
- **break**
- **continue**
- **return**
- **goto**

## 语法符号：

- **;**
- **{}**
- **//**
- **/* */**

## 运算符：

- **=**
- **+**
- **-**
- ***
- **/**
- **%**
- **==**
- **!=**
- **<**
- **>**
- **<=**
- **>=**
- **&&**
- **||**
- **!**

- ++
- --
- +=
- -=
- *=
- /=

## 数据类型：

- **boolean 布尔类型**
- **char 字符类型**
- **byte 字节类型**
- **int 整数类型**
- **unsigned int 无符号整型**
- **long 长整型**
- **unsigned long 无符号长整型**
- **float 实数类型**
- **double**
- **string**
- **array**
- **void**

## 常量：

- **HIGH | LOW 表示数字 IO 口的电平，HIGH 表示高电平（1），LOW 表示低电平（0）。**
- **INPUT | OUTPUT 表示数字 IO 口的方向，INPUT 表示输入（高阻态），OUTPUT 表示**

## 输出（AVR 能提供 5V 电压 40mA 电流）。

- true | false true 表示真（1），false 表示假（0）。

以上为基础 c 语法的关键字和符号，大家可以了解，具体使用可以结合实验的程序。

# 结构

void setup() 初始化发量，管脚模式，调用库函数等
void loop() 连续执行函数内的语句

# 功能

**数字 I/O**

- **pinMode(pin, mode)** 数字 IO 口输入输出模式定义函数，pin 表示为 0～13，mode 表示为 INPUT 或 OUTPUT。
- **digitalWrite(pin, value)** 数字 IO 口输出电平定义函数，pin 表示为 0～13，value 表示为 HIGH 或 LOW。比如定义 HIGH 可以驱动 LED。
- **int digitalRead(pin)** 数字 IO 口读输入电平函数，pin 表示为 0～13，value 表示为 HIGH 或 LOW。比如可以读数字传感器。

**模拟 I/O**

- **int analogRead(pin)** 模拟 IO 口读函数，pin 表示为 0～5（Arduino Diecimila 为 0～5，Arduino nano 为 0～7）。比如可以读模拟传感器（10 位 AD，0～5V 表示为 0～1023）。
- **analogWrite(pin, value)** PWM 数字 IO 口 PWM 输出函数，Arduino 数字 IO 口标注了 PWM 的 IO 口可使用该函数，pin 表示 3, 5, 6, 9, 10, 11，value 表示为 0～255。比如可用于电机 PWM 调速或音乐播放。

**时间函数**

- **delay(ms)** 延时函数（单位 ms）。
- **delayMicroseconds(us)** 延时函数（单位 us）。

**数学函数**

- **min(x, y)** 求最小值
- **max(x, y)** 求最大值
- **abs(x)** 计算绝对值
- **constrain(x, a, b)** 约束函数，下限 a，上限 b，x 必须在 ab 之间才能返回。
- **map(value, fromLow, fromHigh, toLow, toHigh)** 约束函数，value 必须在 fromLow 与 toLow 之间和 fromHigh 与 toHigh 之间。
- **pow(base, exponent)** 开方函数，base 的 exponent 次方。
- **sq(x)** 平方

- **sqrt(x)** 开根号

# 第一章 Hello World!

这一章， 我们简单学习一下利用 Arduino IDE 的串口工具，在电脑中显示我们想要显示的内容。

**实例代码：**

```
void setup()
{
  Serial.begin(9600);// opens serial port, sets data rate to 9600 bps
  Serial.println("Hello World!");
}

void loop()
{

}
```

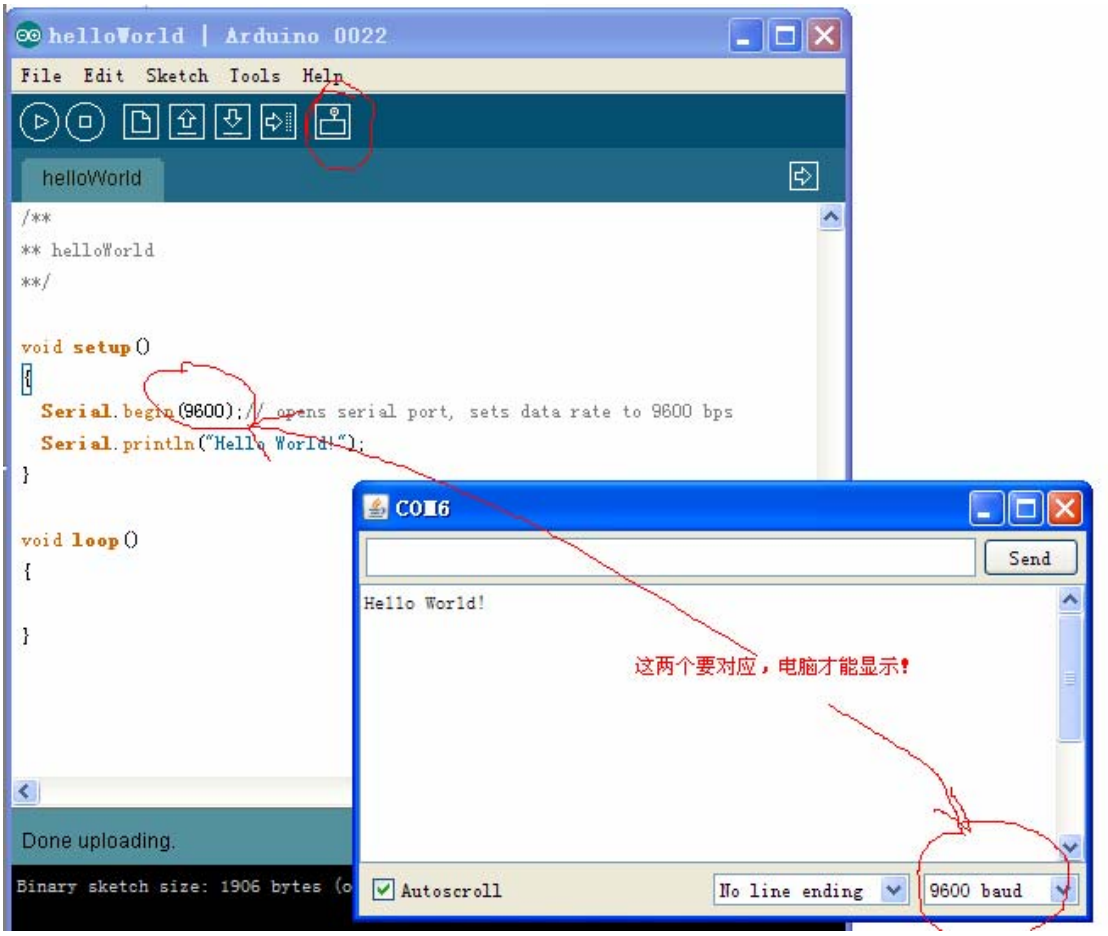
**说明：**

Serial.begin(9600); 这个函数是为串口数据传输设置每秒数据传输速率，每秒多少位数（波特率）。为了能与计算机进行通信，可选择使用以下这些波特率：“ **300，1200，2400，4800，9600，14400，19200，28800，38400，57600 或 115200** ” 。

实验结果与操作：

1) 把代码下载到 arduino 控制板。

2）下载成功后， “”先从选项“tool” ， 选择相应的 arduino 控制板， 和对应的“com”口。 打开串口工具，在新打开的串口工具窗口的“右下角” 选择相应的波特率 。


helloWorld | Arduino 0022
File Edit Sketch Tools Help
helloWorld
/**
** helloWorld
**/
void setup()
{
Serial.begin(9600);// opens serial port, sets data rate to 9600 bps
Serial.println("Hello World!");
}
void loop()
{
}
Done uploading.
Binary sketch size: 1906 bytes (o
COM6
Send
Hello World!
这两个要对应，电脑才能显示！
Autoscroll
No line ending
9600 baud

# 第二章 LED闪烁实验

## 一、发光二极管介绍

### 1、什么是发光二极管

发光二极管简称为 LED。由镓（Ga）与砷（AS）、磷（P）的化合物制成的二极管，当电子与空穴复合时能辐射出可见光，因而可以用来制成发光二极管，在电路及仪器中作为指示灯，或者组成文字或数字显示。磷砷化镓二极管发红光，磷化镓二极管发绿光，碳化硅二极管发黄光。

它是半导体二极管的一种，可以把电能转化成光能；常简写为 LED。发光二极管与普通二极管一样是由一个 PN 结组成，也具有单向导电性。当给发光二极管加上正向电压后，从 P 区注入到 N 区的空穴和由 N 区注入到 P 区的电子，在 PN 结附近数微米内分别与 N 区的电子和 P 区的空穴复合，产生自发辐射的荧光。不同的半导体材料中电子和空穴所处的能量状态不同。当电子和空穴复合时释放出的能量多少不同，释放出的能量越多，则发出的光的波长越短。常用的是发红光、绿光或黄光的二极管。

### 2、工作原理

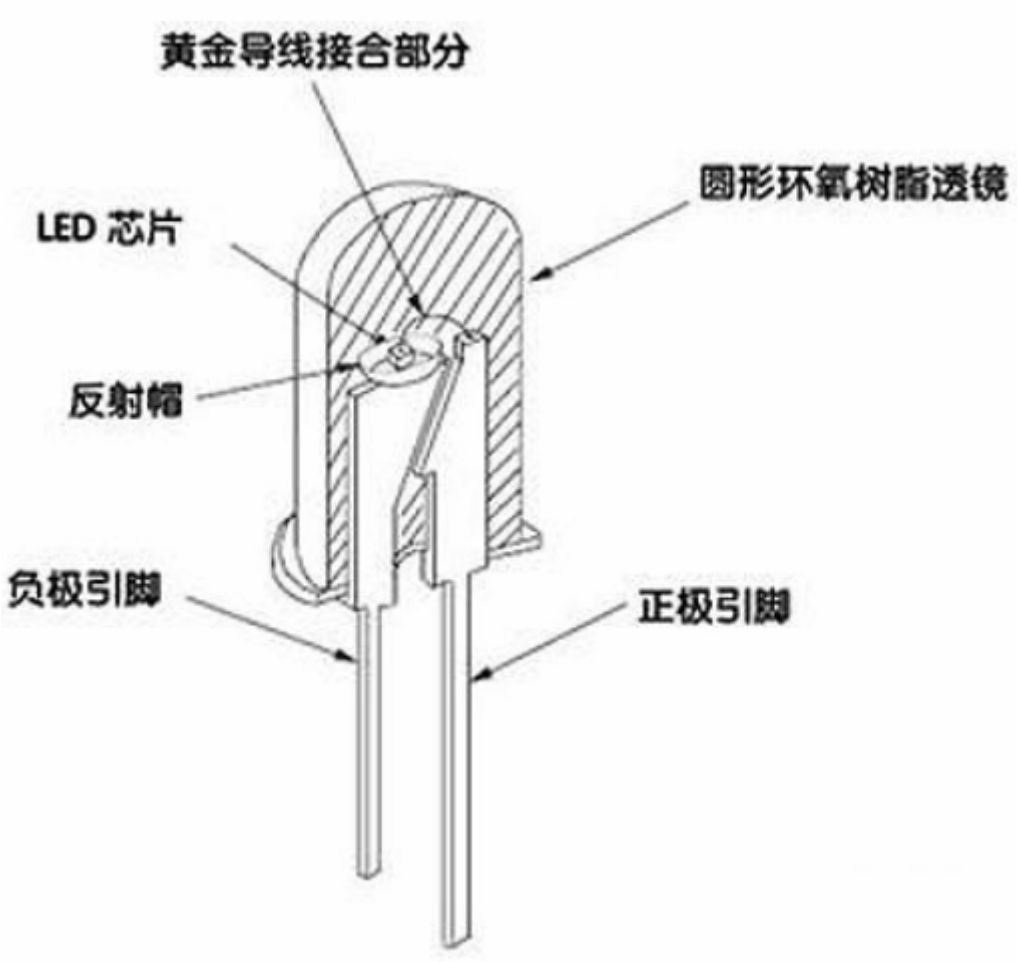
发光二极管的反向击穿电压约 5 伏。它的正向伏安特性曲线很陡，使用时必须串联限流电阻以控制通过管子的电流。限流电阻 R 可用下式计算：

**R＝（E－VF）／I ；**

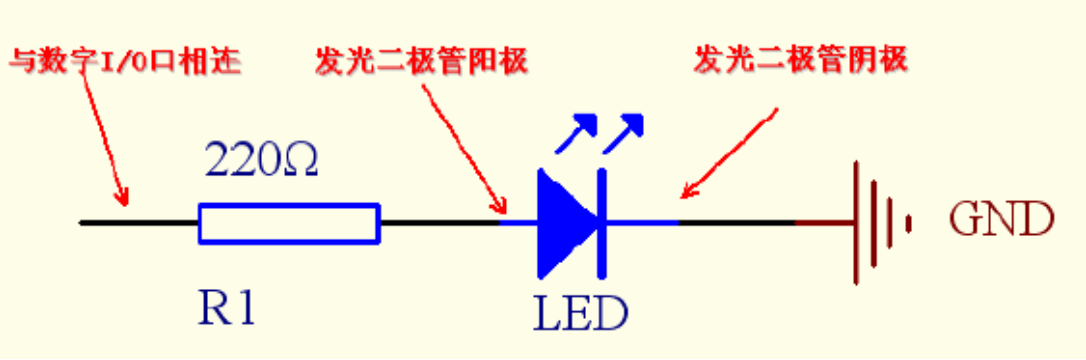
式中 E 为电源电压，VF 为 LED 的正向压降，I 为 LED 的一般工作电流。发光二极管的工作电压一般为 1.5～2.0V，其工作电流一般为 10～20mA。所以在 5v 的数字逻辑电路中，可使用 220Ω 的电阻作为限流电阻。

### 3、Led灯的内部结构与连线

发光二极管的两根引线中较长的一根为正极，应连接电源正极。有的发光二极管的两根引线一样长，但管壳上有一凸起的小舌，靠近小舌的引线是正极。如下图所示：

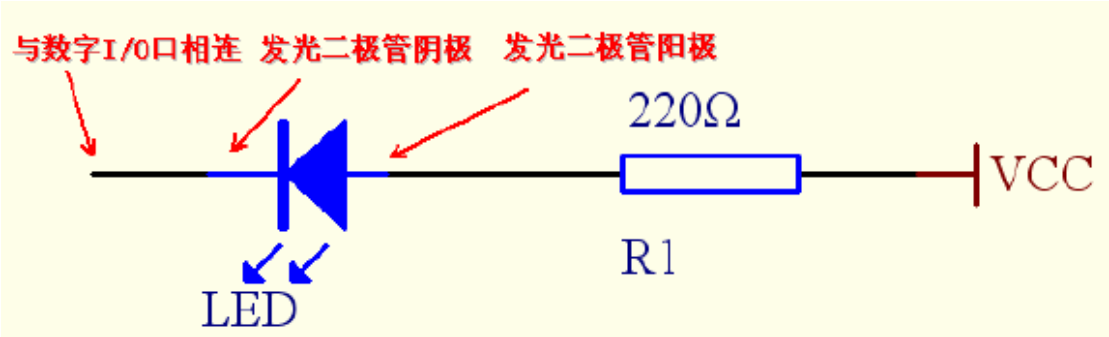



Led 灯有两种连线方法：当 led 灯的阳极通过限流电阻与板子上的数字 I/O 口相连，数字口输出高电平时，led 导通，发光二极管发出亮光；数字口输出低电平时，led 截止，发光二极管熄灭。如图：

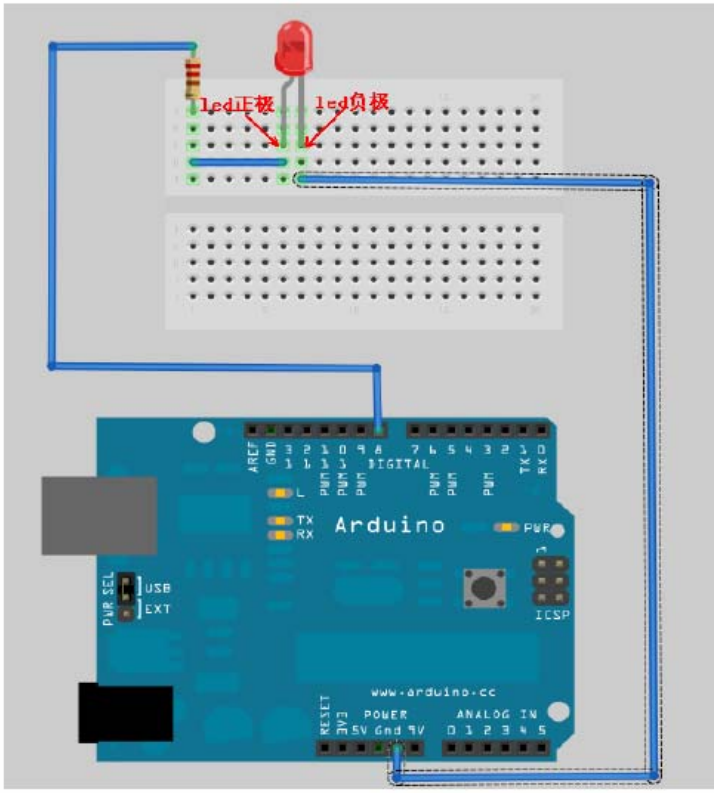



当 led 灯的阴极与板子上的数字 I/O 口相连时，数字口输出高电平，led 截止，

发光二极管熄灭；数字口输出低电平，led 灯导通，发光二极管点亮。




本实验选择了接线方法 1 连接发光二极管，将 220Ω 电阻的一端插在 Prototype Shield 扩展板上的第 8 个 digital I/O 口，电阻的另一端插在面包板上，电阻和发光二极管通过导线相连，发光二极管的负端插在面包板上与 GND 相连。具体连接如图：




# 简单的控制一个led灯的闪烁实验

## 1) 实验器件

Led 灯：1 个
220Ω 的电阻：1 个
多彩面包板实验跳线：若干

## 2) 实验连线

按照 Arduino 使用介绍将控制板、Prototype Shied 板子、面包板连接好，下载线插好。最后，按照图将发光二级管连接到数字的第 8 引脚。这样我们就完成了实验的连线部分。

### 3）实验原理

先设置数字 8 引脚为高电平点亮 led 灯，然后延时 1s，接着设置数字 8 引脚为低电平熄灭 led 灯，再延时 1s。这样使 led 灯亮 1s、灭 1s，在规视上就形成闪烁状态。如果想让 led 快速闪烁，可以将延时时间设置的小一些，但不能过小，过小的话人眼就识别不出来了，看上去就像 led 灯一直在亮着；如果想让 led 慢一点闪烁，可以将延时时间设置的大一些，但也不能过大，过大的话就没有闪烁的效果了。

### 4）程序代码

程序代码在简单 led 程序文件夹中，双击打开后有一个 led1 文件夹，接着双击打开后可以看见有一个 led1.pde 文件，双击图标即可打开。打开后我们可以看到这是 arduino 编程软件窗口，上面有本实验的程序代码。

程序代码如下：

```
int ledPin=8; //设定控制 LED 的数字 IO 脚
void setup()
{
    pinMode(ledPin,OUTPUT);//设定数字 IO 口的模式，OUTPUT 为输出
}
void loop()
{
    digitalWrite(ledPin,HIGH); //设定 PIN8 脚为 HIGH = 5V 左右
    delay(1000); //设定延时时间，1000 = 1 秒
    digitalWrite(ledPin,LOW); //设定 PIN8 脚为 LOW = 0V
    delay(1000); //设定延时时间，1000 = 1 秒
}
```

从 Arduino 教程中我们可以知道，Arduino 语法是以 setup()开头，loop()作为主体的一个程序极架。setup()是用来初始化发量，管脚模式，调用库函数等等，此函数只运行一次。本程序在 setup()中用数字 IO 口输入输出模式定义函数 pinMode（pin，mode），将数字的第 8 引脚设置为输出模式。

loop()函数是一个循环函数，函数内的语句周而复始的循环执行，本程序在 loop()中先用 数字 IO 口输出电平定义函数 digitalWrite(pin, value)，将数字 8 口定义为高电平，点亮 led 灯；接着调用延时函数 delay(ms)（单位 ms）延时 1000ms，让发光二极管亮 1s；再用数字 IO 口输出电平定义函数 digitalWrite(pin, value)，将数字 8 口定义为低电平，熄灭 led 灯；接着再调用延时函数 delay(ms)（单位 ms）延时 1000ms，让发光二极管熄灭 1s。因为 loop()函数是一个循环函数，所以这个过程会不断的循环。

### 5）下载程序

按照 arduino 教程中的程序下载方法将本程序下载到实验板中。

### 6）程序功能

将程序下载到实验板后我们可以观察到，发光二极管以 1s 的时间间隔不断的闪烁。

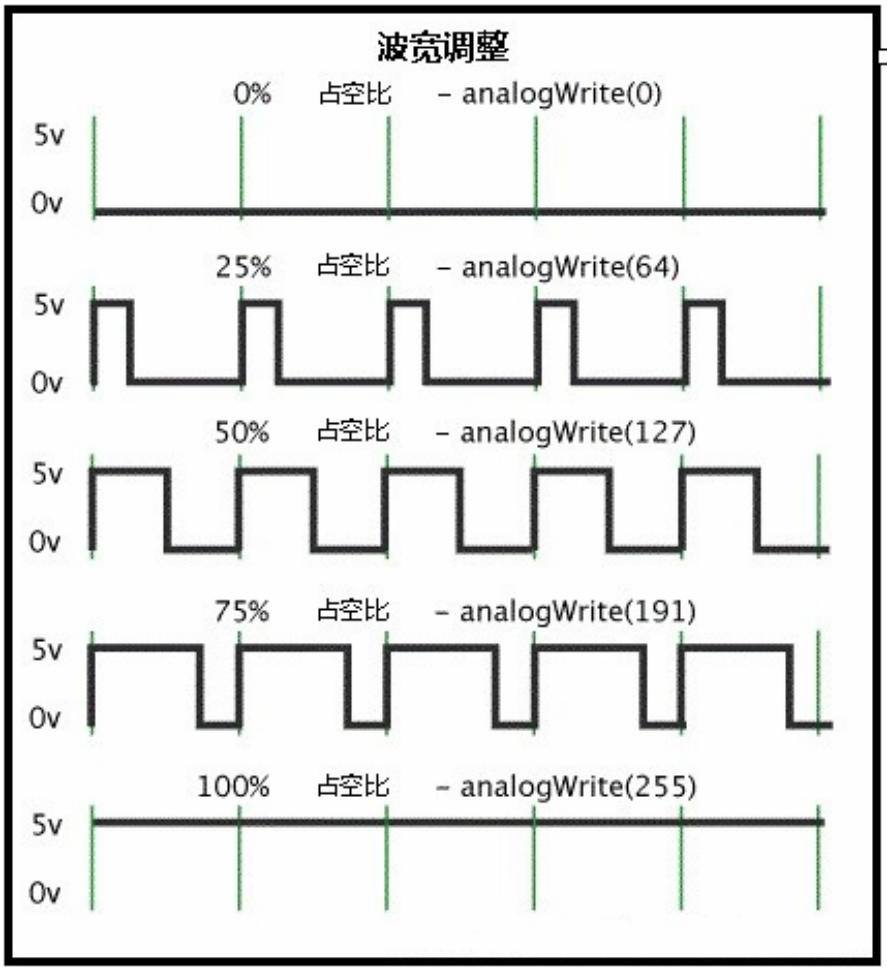
# 第三章 PWM调控灯光亮度实验

### 1、工作原理

PWM 是使用数字手段来控制模拟输出的一种手段。使用数字控制产生占空比不同的方波（一个不停在开与关之间切换的信号)来控制模拟输出。

占空比越高，端口得到的实际电压越接近 5V ， 灯的亮度越亮。

请看下图：




Arduino 端口的输入电压只有两个 0V 与 5V。如我想要 3V 的输出电压怎么办… ，也许你会说串联电阻 ？ OK，这个方法是正确的。但是如果我想 1V,3V,3.5V 等等之间来回变动怎么办呢？不可能不停地切换电阻吧。这种情况下… ， 就需要使用 PWM 了。他是怎么控制的呢，对于 arduino 的数字端口电压输出只有 LOW 与 HIGH 两个开关，对应的就是 0V 与 5V 的电压输出，我们本把 LOW 定义为 0，HIGH 定义为 1 。 一秒内让 arduino 输出 500 个 0 或者 1 的信号。如果这 500 个全部为 1，那就是完整的 5V，如果全部为 0，那就是 0V。如果 010101010101 这样输出，刚好一半一半，这样输出端口实际的输出电压是 2.5V。这个类似我们放映电影，我们所看的电影并不是完全连续的，它其实是每秒输出 25 张图片，在这种情况下人的肉眼是分辨不出来的，看上去就是连续的了。PWM 也是同样的道理，如果想要不同的电压，就控制 0 与 1 的输出比例就 ok 。 当然… 这和真实的连续输出还是有差别的，单位时间内输出的 0,1 信号越多，控制的就越精确。

在上图中，绿线之间代表一个周期，其值也是 PWM 频率的倒数。换句话说，如果 arduino PWM 的频率是 500Hz，那么两绿线之间的周期就是 2 毫秒。 analogWrite() 命令中可以操控的范围为 0-255， analogWrite(255)表示 100%占空比（常开）， analogWrite(127)占空比大约为 50%（一半的时间）。

在 Arduino 语法中， 我们使用函数： **“ analogWrite() “**

**analogWrite() ：**作用是给端口写入一个模拟值(PWM 波)。可以用来控制 LED 灯的亮度变化，或者以不同的速度驱动马达。当执行 analogWrite()命令后，端口会输出一个稳定的占空比的方波。除非有下一个命令来改变它。PWM 信号的频率大约为 490Hz.

在使用 ATmega168、 ATmega328 与 UNO 的 arduino 控制板上，其工作在 3,5,6,9,10,11 端口。 Arduino Mega ，2560 控制板，可以工作于 2-13 号端口。在更古老的基于 ATmega8 的 arduino 控制板上，analogWrite()命令只能工作于 9,10,11 号端口。在使用 analogWrite()命令前，可以不使用 pinMode()命令把端口定义为输出端口，当然如果定义了更好，这样利于程序语言规范。

**语法**

**analogWrite(pin, value)**

**参数**

**Pin：写入的端口**

**value:占空比:在 0-255 之间。**

**注释与已知问题:**

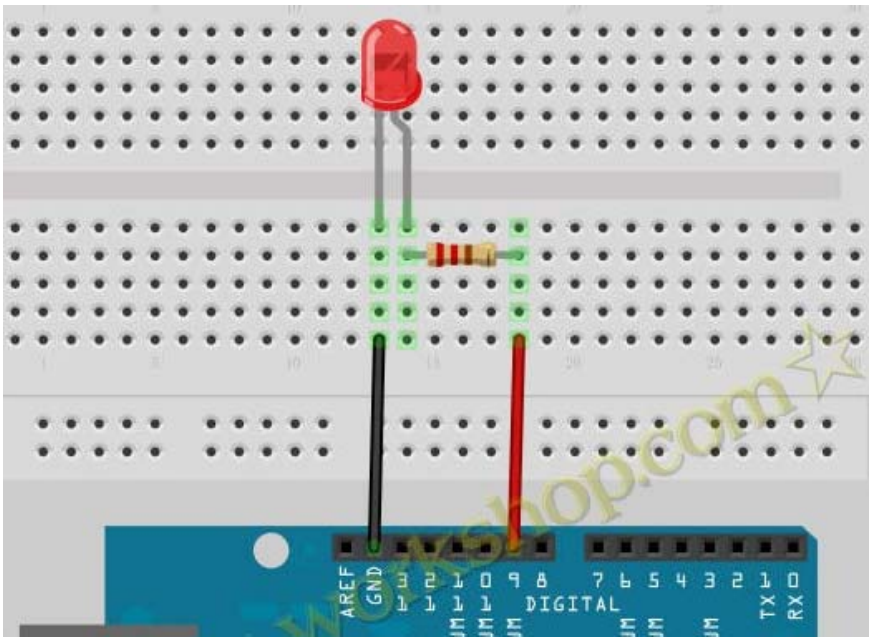
当 PWM 输出与 5,6 号端口的时候，会产生比预期更高的占空比。原因是 PWM 输出所使用的内部时钟，millis()与 delay()两函数也在使用。所以要注意使用 5,6 号端口时，空占比要设置的稍微低一些，或者会产生 5,6 号端口无法输出完全关闭的信号。

## 2) 实验器件

Led 灯：1 个
220Ω 的电阻：1 个
多彩面包板实验跳线：若干

## 3) 实验连线

按照 Arduino 使用介绍将控制板、面包板连接好 最后，按照图将发光二级管通过 220 欧姆电阻连接到数字的第 9 引脚。这样我们就完成了实验的连线部分。




## 5）程序代码

程序代码在“PWM 调控灯光亮度实验”文件夹中。

程序代码如下：

```
int brightness = 0;   //定义整数型变量 brightness 与其初始值，此变量用来表示 LED 的亮度。
int fadeAmount = 5;   //定义整数型变量 fadeAmount，此变量用来做亮度变化的增减量。

void setup()   {
  pinMode(9, OUTPUT);//  设置 9 号口为输出端口:
}

void loop()   {

  analogWrite(9, brightness);//把 brightness 的值写入 9 号端口

  brightness = brightness + fadeAmount;//改变 brightness 值，使亮度在下一次循环发生改变

  if (brightness == 0 || brightness == 255) {
    fadeAmount = -fadeAmount ; //在亮度最高与最低时进行翻转
  }

  delay(30); //延时 30 毫秒
}
```

## 5）下载程序

按照 arduino 教程中的程序下载方法将本程序下载到实验板中。

## 6）程序功能

将程序下载到实验板后我们可以观察到，通过 PWM 来控制一盏 LED 灯，让它慢慢变亮再慢慢变暗，如此循环。

***PWM***

Pulse Width Modulation 脉冲宽度调制，简称脉宽调制。是利用微处理器的数字输出来对模拟电路进行控制的一种非常有效的技术，广泛应用在从测量、通信到功率控制与变换的许多领域中。

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，而只能输出 0V 或 5V 的数字电压值，（0V 为 0；5V 为 1）所以通过高分辨率计数器，利用方波的占空比被调制的方法对一个具体模拟信号的电平进行编码。

但 PWM 信号仍然是数字的，因为在给定的任意时刻，直流供电要么是 5V(数字值为 1)，要么是 0V（数字值为 0）。电压或电流源以一种通(ON)、断(OFF)的重复脉冲序列加到模拟负载上，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，计算公式为：

**输出电压 =（ 接通时间 / 脉冲时间 ） * 最大电压值**

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 控制器上有 6 个 PWM 接口分别是数字接口 3、5、6、9、10、11

# 第四章　　广告流水灯实验

## 1）实验器件

- Led 灯：6 个
- 220Ω的电阻：6 个
- 多彩面包板实验跳线：若干

## 2）实验连线

按照上述方法将板子和数据线连好。然后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：

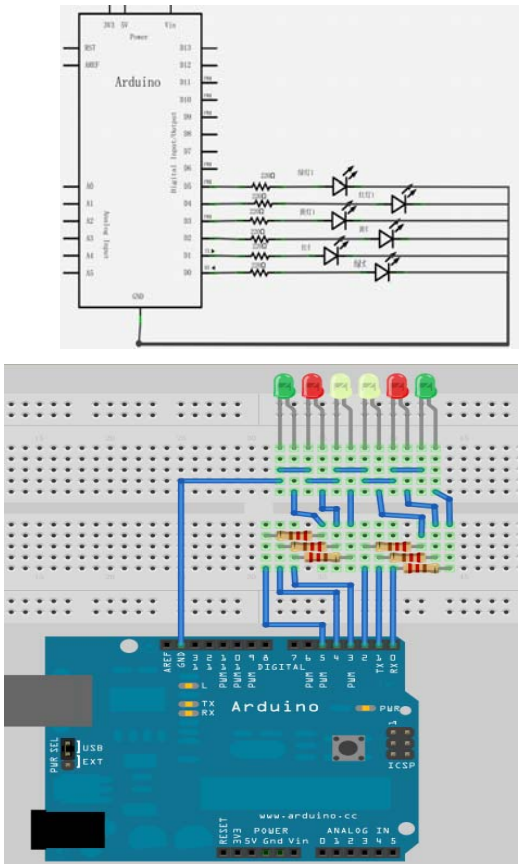



图 1.6 广告灯实验的接线

## 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 灯组成的广告灯，广告灯上各个位置上的 led 灯不断的亮灭变化，就形成各种不同的效果。本节实验就是利用 led 灯编程模拟广告灯的效果。

在程序中我们设置 led 灯亮灭的次序和时间，这样就可以组成不同的效果。样式一子程

序：led 首先从左边的绿灯开始间隔 200ms 依次点亮六个 led 灯，如图 1.6，接着从右边的绿灯开始间隔 200ms 依次熄灭六个 led 灯。灯闪烁子程序：六个 led 灯首先全部点亮，接着延时 200ms，最后六个 led 灯全部熄灭，这个过程循环两次就实现了闪烁的效果。样式二子程序设置 k 和 j 的值让中间的两个黄灯亮先亮，接着让挨着两个黄灯两边的红灯亮，最后让两边的绿灯亮；执行一遍后改发 k 和 j 的值让让两边的绿灯先熄灭，接着两边的红灯熄灭，最后中间的两个黄灯熄灭。样式三子程序：设置 k 和 j 的值，让两边的绿灯亮 400ms 后再熄灭，接着让两边的红灯亮 400ms 后再熄灭，最后让中间的两个黄灯亮 400ms 后再熄灭；执行一遍后改发 k 和 j 的值让两个红灯亮 400ms 后熄灭，接着让两边的绿灯亮 400ms 后熄灭。

## 4）程序代码

程序代码在广告灯程序文件夹中，双击打开后有一个 led2 文件夹，接着双击打开后可以看见有一个 led2.pde 文件，双击图标即可打开。打开后我们可以看到这是 arduino 编程软件窗口，上面有本实验的程序代码。

程序代码如下：

```
//设置控制 Led 的数字 IO 脚
int Led1 = 1;
int Led2 = 2;
int Led3 = 3;
int Led4 = 4;
int Led5 = 5;
int Led6 = 6;
//led 灯花样显示样式 1 子程序
void style_1(void)
{
  unsigned char j;
  for(j=1;j<=6;j++)//每隔 200ms 依次点亮 1~6 引脚相连的 led 灯
  {
    digitalWrite(j,HIGH);//点亮 j 引脚相连的 led 灯
    delay(200);//延时 200ms
  }
  for(j=6;j>=1;j--)//每隔 200ms 依次熄灭 6~1 引脚相连的 led 灯
  {
    digitalWrite(j,LOW);//熄灭 j 引脚相连的 led 灯
    delay(200);//延时 200ms
  }
}
//灯闪烁子程序
void flash(void)
{
  unsigned char j,k;
```

```
  for(k=0;k<=1;k++)//闪烁两次
  {
    for(j=1;j<=6;j++)//点亮 1~6 引脚相连的 led 灯
      digitalWrite(j,HIGH);//点亮与 j 引脚相连的 led 灯
    delay(200);//延时 200ms
    for(j=1;j<=6;j++)//熄灭 1~6 引脚相连的 led 灯
      digitalWrite(j,LOW);//熄灭与 j 引脚相连的 led 灯
    delay(200);//延时 200ms
  }
}
//led 灯花样显示样式 2 子程序
void style_2(void)
{
  unsigned char j,k;
  k=1;//设置 k 的初值为 1
  for(j=3;j>=1;j--)
  {
    digitalWrite(j,HIGH);//点亮灯
    digitalWrite(j+k,HIGH);//点亮灯
    delay(400);//延时 400ms
    k +=2;//k 值加 2
  }
  k=5;//设置 k 值为 5
  for(j=1;j<=3;j++)
  {
    digitalWrite(j,LOW);//熄灭灯
    digitalWrite(j+k,LOW);//熄灭灯
    delay(400);//延时 400ms
    k -=2;//k 值减 2
  }
}
//led 灯花样显示样式 3 子程序
void style_3(void)
{
  unsigned char j,k;//led 灯花样显示样式 3 子程序
  k=5;//设置 k 值为 5
  for(j=1;j<=3;j++)
  {
    digitalWrite(j,HIGH);//点亮灯
    digitalWrite(j+k,HIGH);//点亮灯
    delay(400);//延时 400ms
    digitalWrite(j,LOW);//熄灭灯
    digitalWrite(j+k,LOW);//熄灭灯
    k -=2;//k 值减 2
```

```
  }
  k=3;//设置 k 值为 3
  for(j=2;j>=1;j--)
  {
    digitalWrite(j,HIGH);//点亮灯
    digitalWrite(j+k,HIGH);//点亮灯
    delay(400);//延时 400ms
    digitalWrite(j,LOW);//熄灭灯
    digitalWrite(j+k,LOW);//熄灭灯
    k +=2;//k 值加 2
  }
}
void setup()
{
  unsigned char i;
  for(i=1;i<=6;i++)//依次设置 1~6 个数字引脚为输出模式
    pinMode(i,OUTPUT);//设置第 i 个引脚为输出模式
}
void loop()
{
  style_1();//样式 1
  flash();//闪烁
  style_2();//样式 2
  flash();//闪烁
  style_3();//样式 3
  flash();//闪烁
}
```

程序代码中用到的：

**for(i=1;i<=6;i++)//依次设置 1~6 个数字引脚为输出模式**
**pinMode(i,OUTPUT);//设置第 i 个引脚为输出模式**

这是一个 for 循环。它的一般形式为: for(<初始化>; <条件表达式>; <增量>) 语句; 初始化总是一个赋值语句, 它用来给循环控制发量赋初值; 条件表达式是一个关系表达式, 它决定什么时候退出循环; 增量定义循环控制发量每循环一次后 按什么方式变化。这三个部分之间用";"分开。 例如: for(i=1; i<=10; i++) 语句; 上例中先给 " i " 赋初值 1, 判断 " i " 是否小于等于 10, 若是则执行语句, 之后值增 加 1。再重新判断, 直到条件为假, 即 i>10 时, 结束循环。

## 5）下载程序

按照 arduino 教程中的程序下载方法将本程序下载到实验板中。

## 6）程序功能

将程序下载到实验板后我们可以观察到，六个 led 不断的循环执行样式一子程序—>闪烁子程序—>样式二子程序—>闪烁子程序—>样式三子程序—>闪烁子程序。

在掌握了以上两个程序后，大家可以充分发挥自己的想象，编写出自己想要的 led 灯效果，玩转多彩 led 灯。

# 第五章　交通灯设计实验

## 1）实验器件

- 红、绿、黄 Led 灯：3 个
- 220Ω的电阻：3 个
- 多彩面包板实验跳线：若干
- 面包板 ： 1 个

## 2）实验连线

按照下图原理图和实物图 ， 将 3 个 LED 灯依次接到数字 10 ， 7 ，4 脚上。如图：

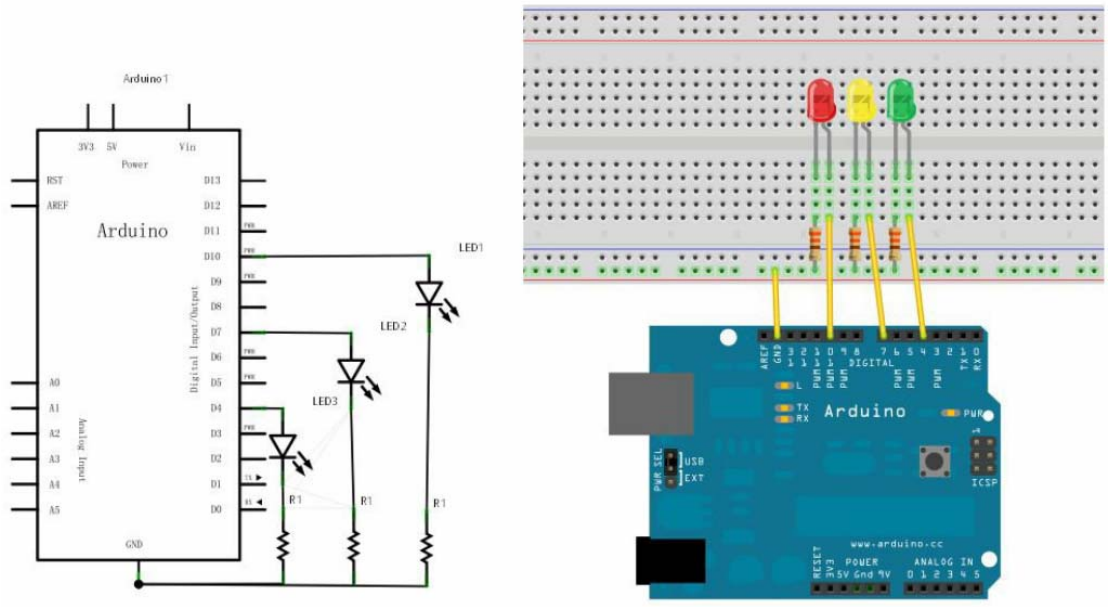



图 1.6 广告灯实验的接线

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

## 4）程序代码

程序代码在“交通灯设计实验”文件夹中，接着双击打开后可以看见有一个 trafficLed.pde 文件，双击图标即可打开。打开后我们可以看到这是 arduino 编程软件窗口，上面有本实验的程序代码。

**程序代码如下：**

```
int ledred=10;     //定义数字接口 10  红灯
int ledyellow=7;   //定义数字接口 7  黄灯
```

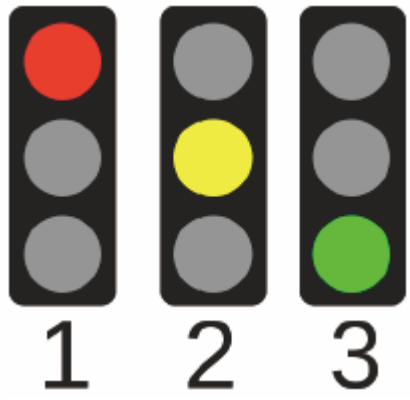
```
int ledgreen=4;     //定义数字接口 4  绿灯
void setup()
{
    pinMode(ledred,OUTPUT);//设置红灯接口为输出接口
    pinMode(ledyellow,OUTPUT);//设置黄灯接口为输出接口
    pinMode(ledgreen,OUTPUT);//设置绿灯接口为输出接口
}
void loop()
{
    digitalWrite(ledred,HIGH);//点亮红灯
    delay(1000);//延时 1000 毫秒 = 1 秒
    digitalWrite(ledred,LOW);//熄灭红灯
    digitalWrite(ledyellow,HIGH);//点亮黄灯
    delay(200);//延时 200 毫秒//
    digitalWrite(ledyellow,LOW);//熄灭黄灯
    digitalWrite(ledgreen,HIGH);//点亮绿灯
    delay(1000);//延时 1000 毫秒
    digitalWrite(ledgreen,LOW);//熄灭绿灯
}
```

## 5）下载程序

按照 arduino 教程中的程序下载方法将本程序下载到实验板中。

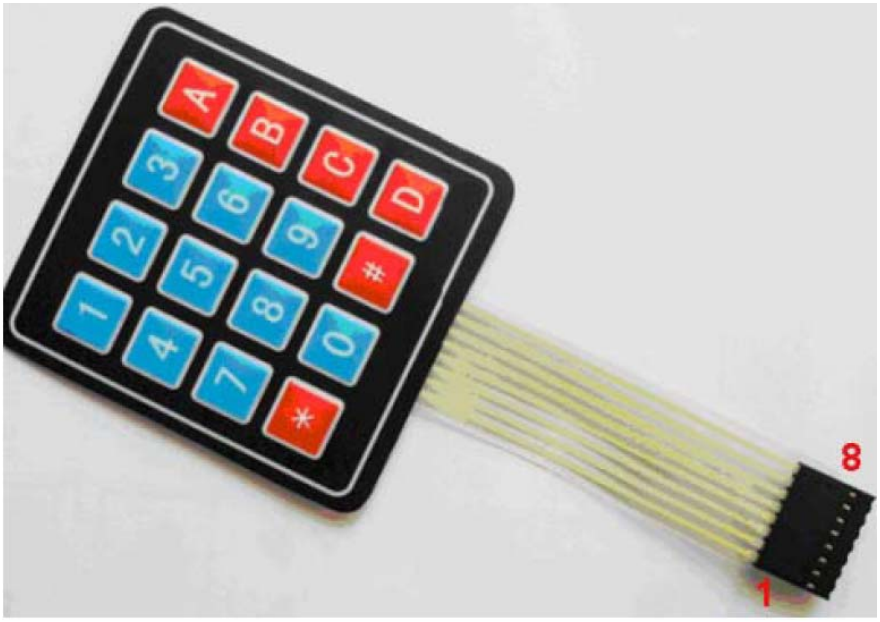
## 6）程序功能

将程序下载到实验板后，我们可以看到我们自己，设计控制的交通灯了。

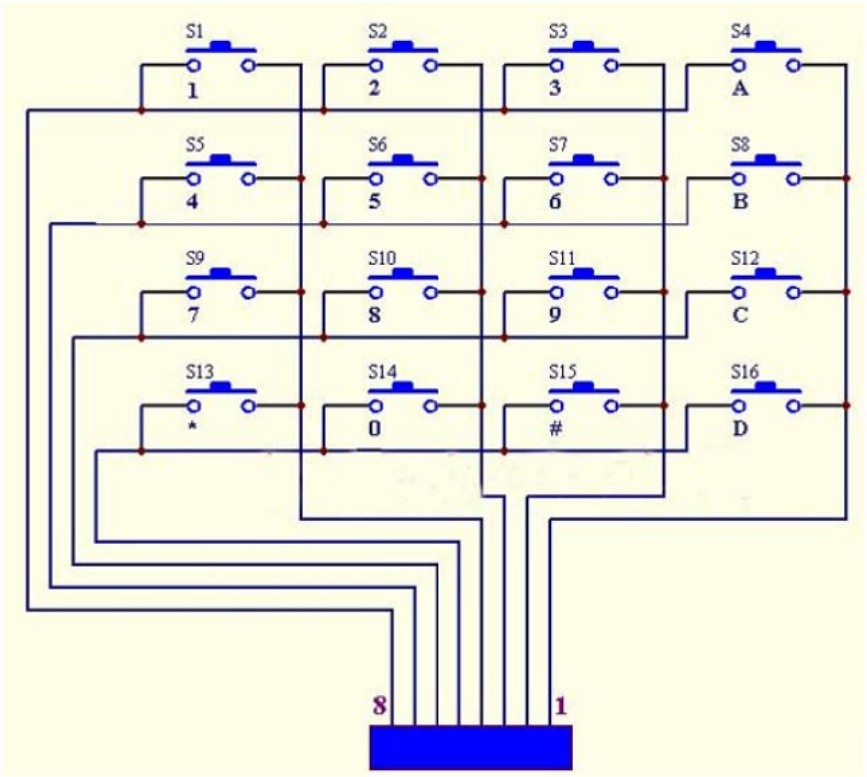


# 第六章　4x4 按键显示实验

## 1）概述




4*4 薄膜按键脚位，请看上图。 其原理图如下：




## 实验器件

- 4*4 薄膜按键：1 个
- 多彩面包板实验跳线：若干
- 面包板 ： 1 个

## 2）实验连线

按照下图原理图， 将 4*4 薄膜按键的 1-8 依次接到数字 2-9 脚上。如图：

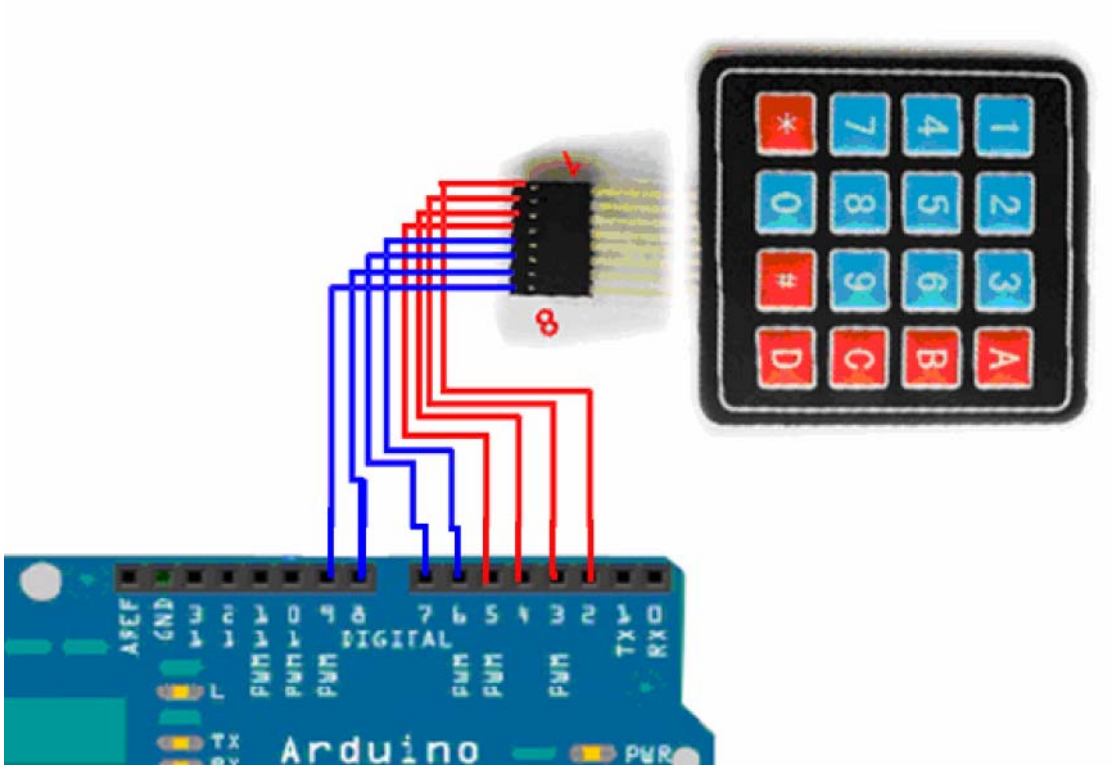



图 1.6 广告灯实验的接线

## 4）程序代码

程序代码在“4x4 按键显示实验”文件夹中。首先要将按键类库文件“Keypad.zip”，解压到 arduino IDE 的安装文件夹下的“libraries” 文件夹。

**程序代码如下：**

```
#include <Keypad.h>

const byte ROWS = 4; //定义 4 行
const byte COLS = 4; //定义 4 列
char keys[ROWS][COLS] = {
  {'1','2','3','A'},
  {'4','5','6','B'},
  {'7','8','9','C'},
```

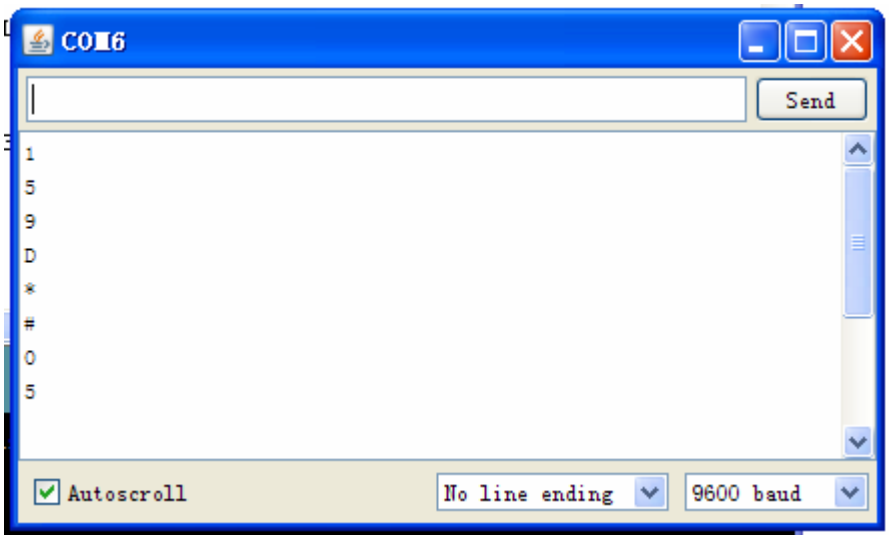
```
    {'*','0','#','D'}
};
//连接 4*4 按键的行位端口，相应控制板的数字 IO 口
byte rowPins[ROWS] = {2,3,4,5};
//连接 4*4 按键的列位端口，相应控制板的数字 IO 口
byte colPins[COLS] = {6,7,8,9};

//调用 Keypad 类库功能函数
Keypad keypad = Keypad( makeKeymap(keys), rowPins, colPins, ROWS, COLS );

void setup(){
   Serial.begin(9600);
}

void loop(){
   char key = keypad.getKey();

   if (key != NO_KEY){
      Serial.println(key);
   }
}
```

## 5）下载程序

按照 arduino 教程中的程序下载方法将本程序下载到实验板中。

## 6）程序功能

将程序下载到实验板后，打开串口工具， 此时按下键盘上的某个键， 在串口工具上显示该按键的值 。如图，我们按下“#”，则显示如下：



# 第七章　4x4 按键控制灯实验

1)　实验连线：

使用上一章的连线图。 并在这里， 我们借用控制板上的 13 脚连接的小灯。

2）实验代码：

```
#include <Keypad.h>

const byte ROWS = 4; //定义 4 行
const byte COLS = 4; //定义 4 列
char keys[ROWS][COLS] = {
  {'1','2','3','A'},
  {'4','5','6','B'},
  {'7','8','9','C'},
  {'*','0','#','D'}
};
//连接 4*4 按键的行位端口，相应控制板的数字 IO 口
byte rowPins[ROWS] = {2,3,4,5};
//连接 4*4 按键的列位端口，相应控制板的数字 IO 口
byte colPins[COLS] = {6,7,8,9};

Keypad keypad = Keypad( makeKeymap(keys), rowPins, colPins, ROWS, COLS );
byte ledPin = 13;

boolean blink = false;

void setup(){
  Serial.begin(9600);
  pinMode(ledPin, OUTPUT);          // sets the digital pin as output
  digitalWrite(ledPin, HIGH);     // sets the LED on
  keypad.addEventListener(keypadEvent); //add an event listener for this keypad
}

void loop(){
  char key = keypad.getKey();

  if (key != NO_KEY) {
    Serial.println(key);
  }
  if (blink){
    digitalWrite(ledPin,!digitalRead(ledPin));
```

```
    delay(100);
  }
}

//take care of some special events
void keypadEvent(KeypadEvent key){
  switch (keypad.getState()){
    case PRESSED:
      switch (key){
        case '#': digitalWrite(ledPin,!digitalRead(ledPin)); break;
        case '*':
          digitalWrite(ledPin,!digitalRead(ledPin));
        break;
      }
    break;
    case RELEASED:
      switch (key){
        case '*':
          digitalWrite(ledPin,!digitalRead(ledPin));
          blink = false;
        break;
      }
    break;
    case HOLD:
      switch (key){
        case '*': blink = true; break;
      }
    break;
  }
}
```

3）实验结果

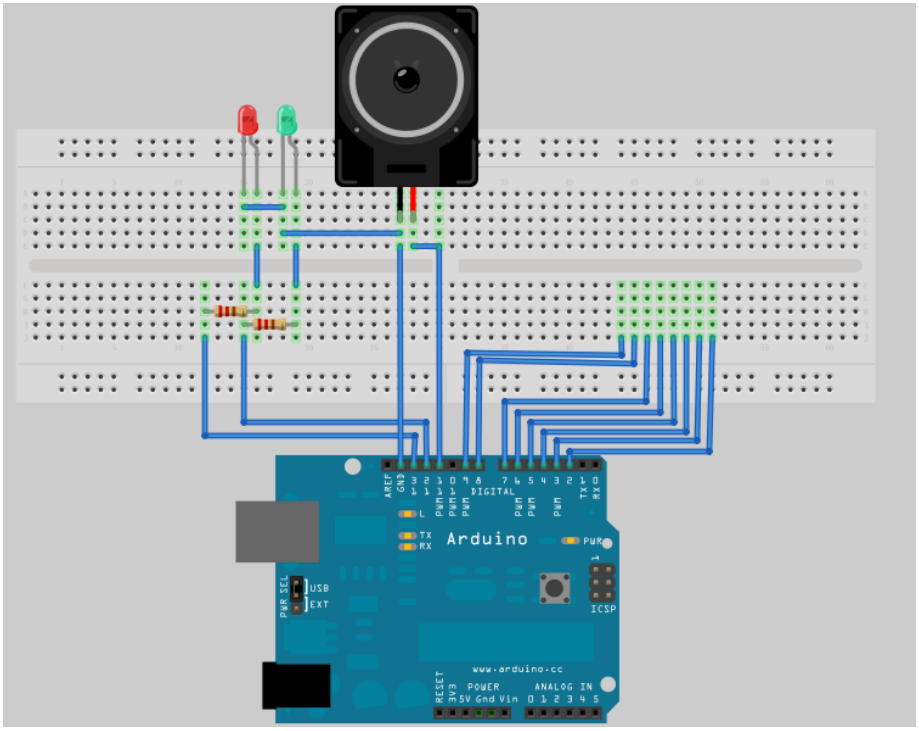
该程序，当按下按键“*”不放时，控制板自带的 13 脚 LED 灯将一直亮， 直到释放按键“*” ； 当按下按键“#”，然后就释放时，13 脚小灯将一直亮，再按一下“#”时，小灯熄灭！

# 第八章　抢答器实验

1） 实验连线：

使用上两章的按键连线图连接 到 控制板的数字端口 2-9，再按照下图连接剩下的蜂鸣器和两盏小灯。

小灯抢答器实验。




2）实验代码：

```
#include <Keypad.h>

const byte ROWS = 4; //定义 4 行
const byte COLS = 4; //定义 4 列
char keys[ROWS][COLS] = {
  {'1','2','3','A'},
  {'4','5','6','B'},
  {'7','8','9','C'},
  {'*','0','#','D'}
};
//连接 4*4 按键的行位端口，相应控制板的数字 IO 口
byte rowPins[ROWS] = {2,3,4,5};
//连接 4*4 按键的列位端口，相应控制板的数字 IO 口
byte colPins[COLS] = {6,7,8,9};

Keypad keypad = Keypad( makeKeymap(keys), rowPins, colPins, ROWS, COLS );
byte redLed = 13;
```

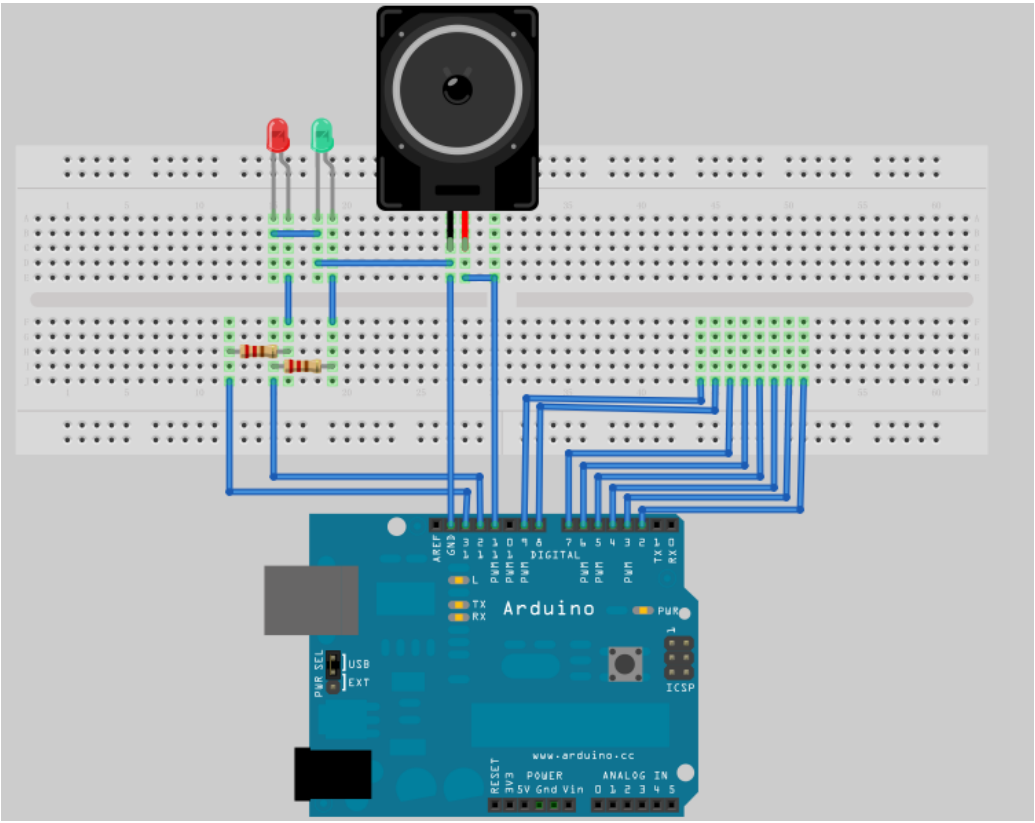
```
byte greenLed = 12;
byte buzzerPin = 11;

boolean blink = false;

void buzzer()//蜂鸣器发出“嘀”声音子程序
{
        for(int i=0;i<80;i++)
        {
        digitalWrite(buzzerPin,HIGH);//发声音
        delay(1);//延时 1ms
        digitalWrite(buzzerPin,LOW);//不发声音
        delay(1);//延时 ms
        }
}
void setup(){
  pinMode(redLed, OUTPUT);
  pinMode(greenLed, OUTPUT);
  pinMode(buzzerPin, OUTPUT);
  digitalWrite(redLed, LOW);
  digitalWrite(greenLed, LOW);
  digitalWrite(buzzerPin, LOW);
  keypad.addEventListener(keypadEvent); //add an event listener for this keypad
}

void loop(){
  char key = keypad.getKey();
}

//take care of some special events
void keypadEvent(KeypadEvent key){
  switch (keypad.getState()){
    case PRESSED:
      switch (key){
        case '1': //按键 1 确实被按下
        {
            buzzer();//蜂鸣器发出声音
            digitalWrite(redLed,HIGH);//红灯亮
            digitalWrite(greenLed,LOW);//绿灯灭
        } break;
        case '2': //按键 2 确实被按下
        {
            buzzer();//蜂鸣器发出声音
            digitalWrite(redLed,LOW);//红灯灭
```

```
                digitalWrite(greenLed,HIGH);//绿灯亮
            } break;
            case '3'://按键 3 确实被按下
            {
                buzzer();//蜂鸣器发出声音
                digitalWrite(redLed,LOW);//红灯灭
                digitalWrite(greenLed,LOW);//绿灯灭
            } break;
            default: break;
        }
      break;
   }
}
```

3）实验结果

按键 1 和 2 是抢答按键，按键 3 是清除按键。 如果按键 1 先被按下，蜂鸣器发出提示音，红灯亮，绿灯灭；如果按键 2 先被按下，蜂鸣器发出提示音，绿灯亮，红灯灭；如果按键 3 被按下，蜂鸣器发出提示音，将红灯和绿灯都熄灭。

两名选手各选一个按键，当比赛开始后进行抢答，谁先按下按键对应的灯就会亮起来。裁判可根据亮灯情况提示参赛选手答题，本次结束后，裁判按下按键 3 清除现在亮灯情况（即将亮灯都熄灭。）



# 第八章　蜂鸣器实验

## 一、蜂鸣器介绍

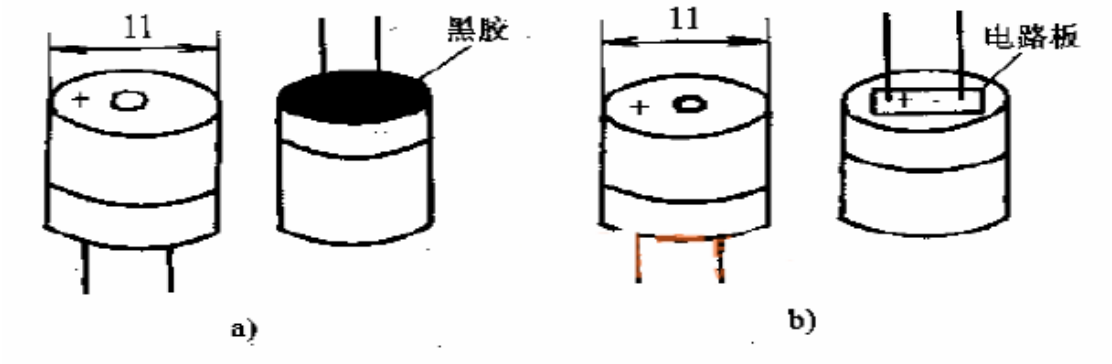
### 1、认识蜂鸣器

蜂鸣器是一种一体化结构的电子讯响器，采用直流电压供电，广泛应用于计算机、打印机、复印机、报警器、电子玩具、汽车电子设备、电话机、定时器等电子产品中作发声器件。

图 2.1 蜂鸣器

按其驱动方式的不同，可分为：有源蜂鸣器（内含驱动线路）和无源蜂鸣器（外部驱动）

教你区分有源蜂鸣器和无源蜂鸣器,现在市场上出售的一种小型蜂鸣器因其体积小(直径只有 llmm)、重量轻、价格低、结构牢靠，而广泛地应用在各种需要发声的电器设备、电子制作和单片机等电路中。有源蜂鸣器和无源蜂鸣器的外观如图 a、b 所示。a)有源 b)无源。



从图 a、b 外观上看，两种蜂鸣器好像一样，但仔细看，两者的高度略有区别，有源蜂鸣器 a，高度为 9mm，而无源蜂鸣器 b 的高度为 8mm。如将两种蜂鸣器的引脚郡朝上放置时，可以看出有绿色电路板的一种是无源蜂鸣器，没有电路板而用黑胶封闭的一种是有源蜂鸣器。进一步判断有源蜂鸣器和无源蜂鸣器，还可以用万用表电阻档 Rxl 档测试:用黑表笔接蜂鸣器 "+"引脚，红表笔在另一引脚上来回碰觉，如果觉发出咔、咔声的且电阻只有 8Ω(或 16Ω)的是无源蜂鸣器;如果能发出持续声音的，且电阻在几百欧以上的，是有源蜂鸣器。有源蜂鸣器直接接上额定电源(新的蜂鸣器在标签上都有注明)就可连续发声;而无源蜂鸣器则和电磁扬声器一样，需要接在音频输出电路中才能发声。

按构造方式的不同，可分为：电磁式蜂鸣器和压电式蜂鸣器；

压电式蜂鸣器 压电式蜂鸣器主要由多谐振荡器、压电蜂鸣片、阻抗匹配器及共鸣箱、外壳等组成。有的压电式蜂鸣器外壳上还装有发光二极管。多谐振荡器由晶体管或集成电路构成。当接通电源后（1.5~15V 直流工作电压）,多谐振荡器起振,输出 1.5~2.5kHZ 的音频信号，阻抗匹配器推动压电蜂鸣片发声压电蜂鸣片由锆钛酸铅或铌镁酸铅压电陶瓷材料制成。在陶瓷片的两面镀上银电极，经极化和老化处理后，再不黄铜片或不锈钢片粘在一起。

电磁式蜂鸣器由振荡器、电磁线圈、磁铁、振动膜片及外壳等组成。接通电源后，振荡器产生的音频信号电流通过电磁线圈，使电磁线圈产生磁场。振动膜片在电磁线圈和磁铁的相互作用下，周期性地振动发声。

## 2、工作原理

蜂鸣器发声原理是电流通过电磁线圈，使电磁线圈产生磁场来驱动振动膜发声的，因此需要一定的电流才能驱动它，本实验用的蜂鸣器内部带有驱动电路，所以可以直接使用。当不蜂鸣器连接的引脚为高电平时，内部驱动电路导通，蜂鸣器发出声音；当不蜂鸣器连接的引脚为低电平，内部驱动电路截止，蜂鸣器不发出声音。

## 3、蜂鸣器的连线

本实验用的蜂鸣器内部带有驱动电路，所以可以直接将蜂鸣器的正极连接到数字口，蜂鸣器的负极连接到 GND 插口中。如下图：

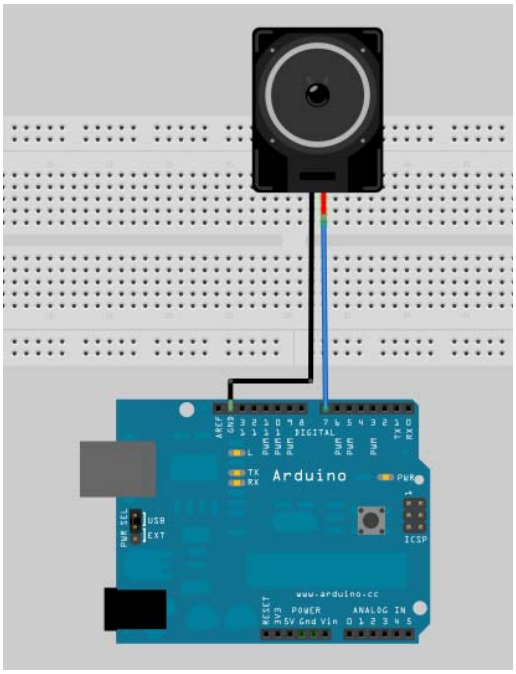



图 2.1 蜂鸣器接线图

# 二、蜂鸣器模拟救护车警笛声音实验

## 1、 实验器件

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、面包板连接好，下载线插好。然后按照蜂鸣器的接法将蜂鸣器连接到数字 7 口上，连线完毕。

## 3、 实验原理

蜂鸣器发出声音的时间间隔不同，频率就不同，所以发出的声音就不同。根据返一原理我们通过改发蜂鸣器发出声音的时间间隔，来发出不同种声音，来模拟各种声音。
本程序首先让蜂鸣器间隔 1ms 发出一种频率的声音，循环 80 次；接着让蜂鸣器间隔 2ms 发出另一种频率的声音，循环 100 次。

## 4、 程序代码

程序代码在 “蜂鸣器实验” 文件夹中，双击打开后有一个 buzzer 文件夹，接着双击打开后可以看见有一个 buzzer.pde 文件，双击图标即可打开。打开后我们可以看到返是 arduino 编程软件窗口，上面有本实验的程序代码。

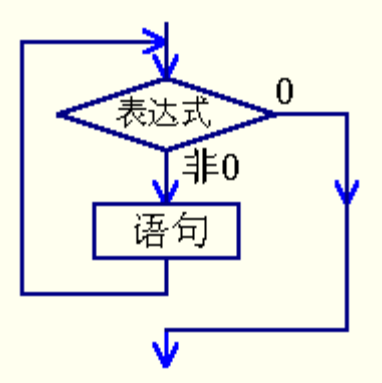
**程序代码如下：**

```
int buzzer=7;//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++)//输出另一个频率的声音
    {
      digitalWrite(buzzer,HIGH);//发声音
      delay(2);//延时 2ms
      digitalWrite(buzzer,LOW);//不发声音
      delay(2);//延时 2ms
    }
  }
}
```

在 loop（）中用的 while 也是一个循环语句，一般形式：

**while(表达式)**
**语句**

表达式是循环条件，语句是循环体。语义是：计算表达式的值，当值为真(非 0)时， 执行循环体语句。其执行过程可用下图表 ：




作用：实现“当型”循环。当“表达式”非 0（真）时，执行“语句”。“语句”是被循环执行的程序，称为“循环体”。

## 5、 下载程序

按照 arduino 教程中的程序下载方法将本程序下载到实验板中。

## 6、 程序功能

将程序下载到实验板后我们可以听到，蜂鸣器发出救护车警笛声。
掌握本程序后，大家可以在程序中自己改发时间间隔，调试出各种频率的声音。

# 第十章 倾斜开关实验

## 一、倾斜开关介绍

### 1、了解倾斜开关

本节实验所使用的倾斜开关是内部带有一个金属滚珠的滚珠倾斜开关，如图所示：

图 5.1 SW—200D 倾斜开关

滚珠开关：也叫碰珠开关、摇珠开关、钢珠开关、倾斜开关，倒顺开关、角度传感器。它主要是利用滚珠在开关内随不同倾斜角度的变化，达到触发电路的目的。 目前滚珠开关在市场上使用的常用型号有 SW-200D、SW-460、SW-300DA 等，本节使用的是 SW-200D 型号的。这类开关不象传统的水银开关，它功效同水银开关，但没有水银开关的环保及安全等问题。

### 2、工作原理

观察倾斜开关我们可以发现，倾斜开关的一端为金色寻针，另一端为银色寻针。金色一端为<ON>导通触发端银色一端为<OFF>开路端当受到外力摇晃而达到适当晃动力时或金色一端设置角度低于水平适当角度时导电接脚电气特性会产生短时间导通或持续导通<ON>状态 。而当电气特性要恢复开路状态<OFF>时开关设置环境必须为静止，且银色一端设置角度需低于水平 10 度。

### 3、倾斜开关连线

将倾斜开关银色的一端连接到 5V 插口，金色一端连接到模拟口。

# 二、倾斜开关控制led灯的亮灭

## 1、实验器件

- 倾斜开关模块：1 个
- IO 传感器扩展板：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、传感器扩展板、面包板连接好，下载线插好。然后将倾斜开关连接到模拟 5 引脚。如图：

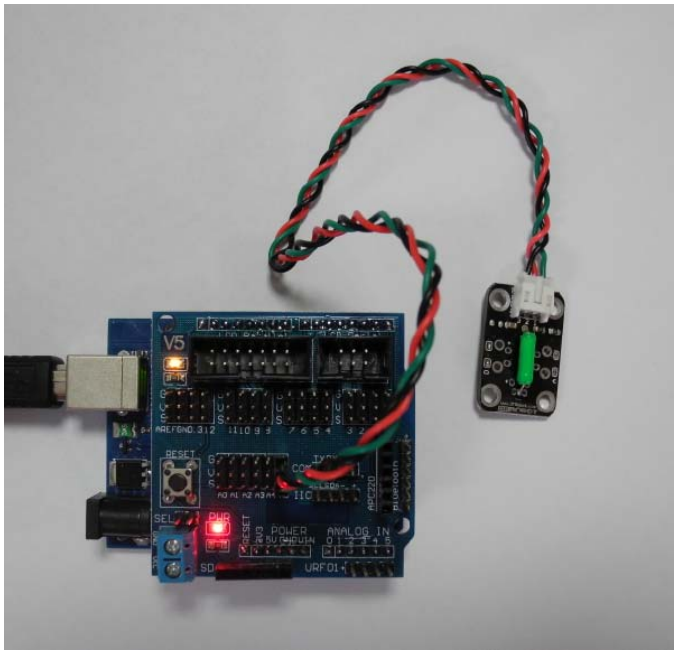


图　倾斜开关实验接线图

## 3、实验原理

当金色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。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

程序代码在“倾斜开关实验”程序文件夹中，双击打开后有一个 tilt.pde 文件，双击图标即可打开。打开后我们可以看到这是 arduino 编程软件窗口，上面有本实验的程序代码。大家可以参考 arduino 教程，浏览 arduino 语法中的各语句的功能。结合 arduino 教程，理解

程序代码。

程序代码如下：

```
void setup()
{
    pinMode(8,OUTPUT);//设置数字 8 引脚为输出模式
}
void loop()
{
    int i;//定义变量 i
    while(1)
    {
        i=analogRead(5);//读取模拟 5 口电压值
        if(i>200)//如果大于 512（2.5V）
        {
            digitalWrite(8,HIGH);//点亮 led 灯
        }
        else//否则
        {
            digitalWrite(8,LOW);//熄灭 led 灯
        }
    }
}
```

## 5、下载程序

按照 arduino 教程中的程序下载方法将本程序下载到实验板中。

## 6、程序功能

将程序下载到实验板后大家可以将板子倾斜观察 led 灯的状态。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。

# 第十一章 模拟值读取实验

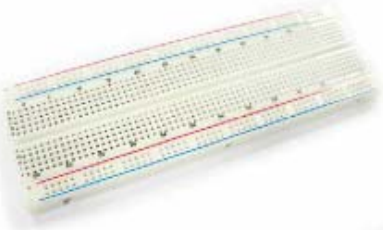
(1)简述

本章我们将使用模拟 I/O 口，Arduino 有模拟口—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 。

下面我们开始做实验了。

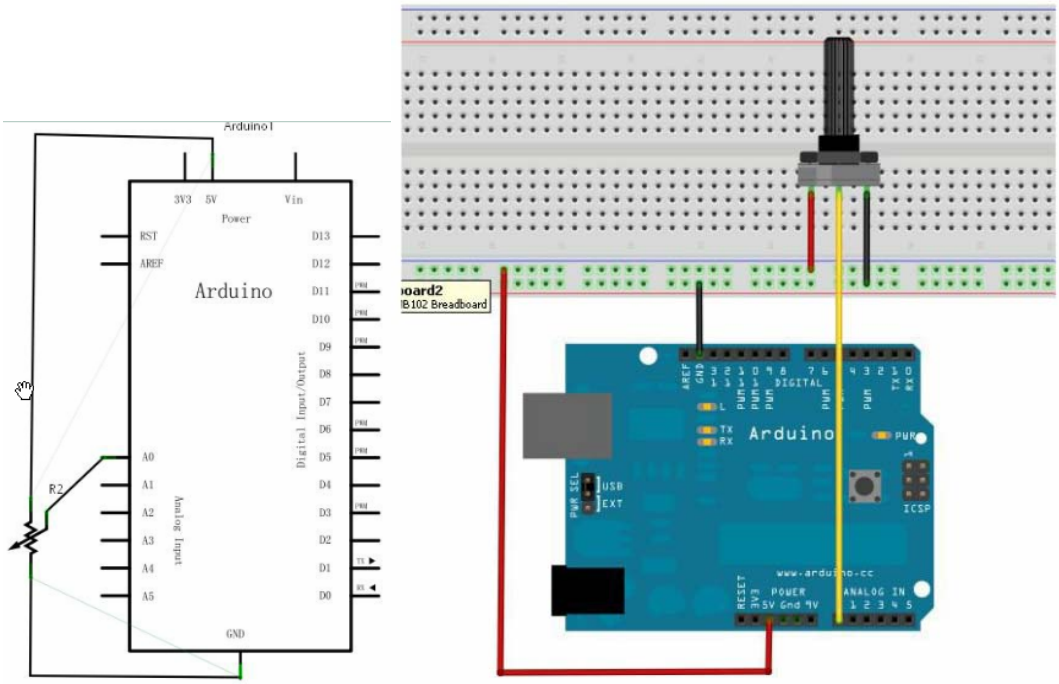
电位器（或者叫滑动电阻）是大家比较熟悉的典型的模拟值输出元件，本实验就用它来完成。

电位计*1



面包板*1

面包板跳线*1 扎




analogRead();语句就可以读出模拟口的值，Arduino UNO 控制器是 10 位的 A/D 采集，所以读取的模拟值范围是 0-1023 。 首先我们在 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正确的数值，否则将会显示乱码或者不显示。在 Arduino 软件的串口工具监视窗口右下角有一个

可以设置波特率的按钮，选中与程序中设置的波特率语句相同的波特率，Serial.begin();括号中为波特率的值。

实例程序：

```
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);//设置数字 13 引脚为输出模式
      Serial.begin(9600);//设置波特率 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
}
```

**程序结果：**

每读取一次值，arduino 自带的 LED 小灯就会闪烁一下，下图为读出的值（注意只做参考）：


本实验中，当您旋转电位计旋钮的时候就可以看到屏幕上的数值变化了 。这种模拟值读取是我们很常用的功能。因因为在很多的传感器，都是模拟值输出出，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。

# 第几章 光控声音实验

## 一、光敏电阻介绍

### 1、认识光敏电阻

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

实物如图：

图 6.1 光敏电阻

### 2、工作原理

光敏电阻的工作原理是基于内光电效应。在半寻体光敏材料两端装上电极引线，将其封装在带有透明窗的管壳里就极成光敏电阻，为了增加灵敏度，两电极常做成梳状。在有光照射时入射光强，电阻减小，入射光弱，电阻增大。

## 二、光控实验

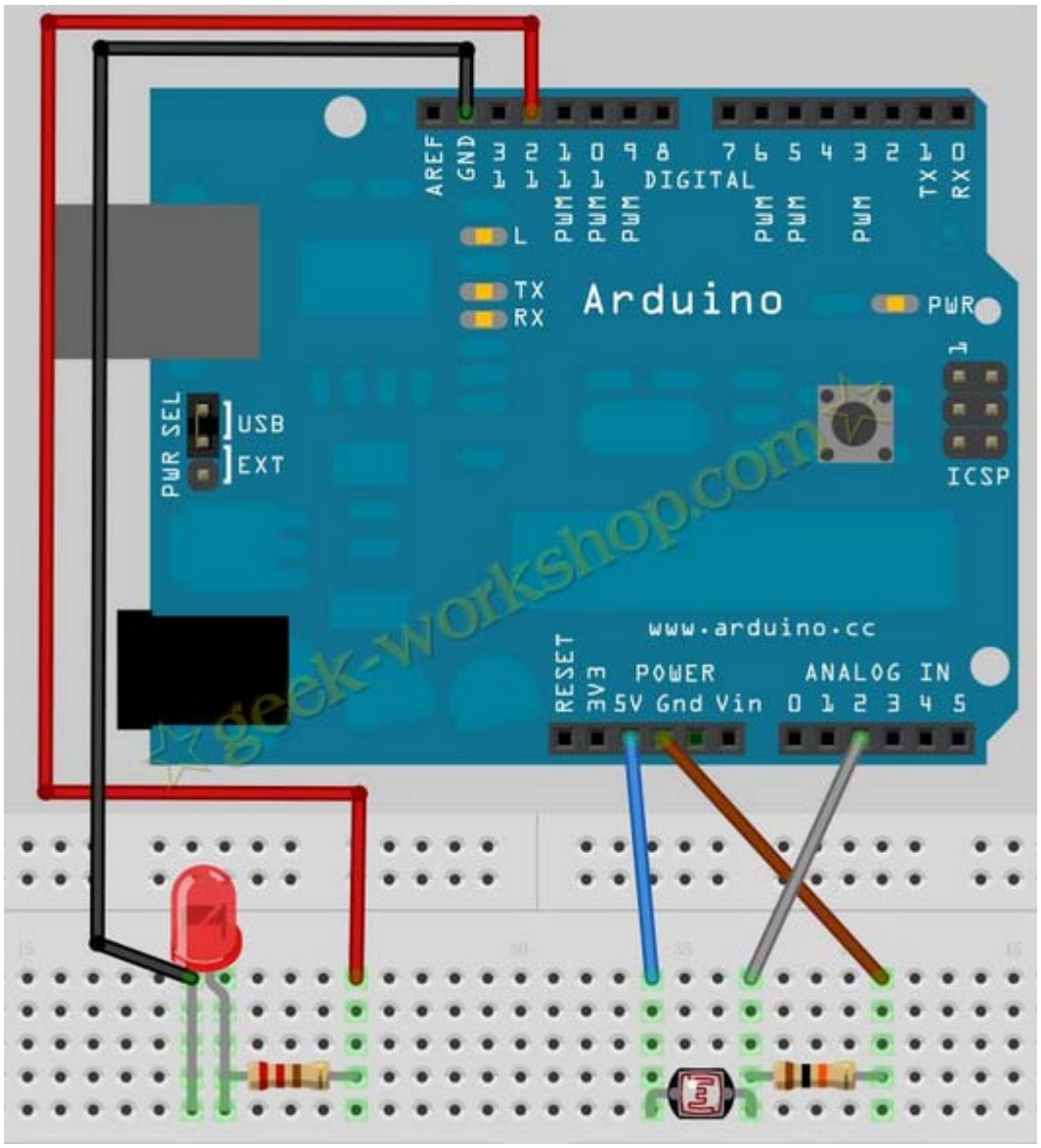
### 1、实验器件

- 光敏电阻：1 个

- 蜂鸣器：1 个
- 10 K 欧电阻 ： 1 个
- 220 欧电阻 ： 1 个
- 多彩面包板实验跳线：若干

## 2、光敏电阻的连线

光敏电阻在使用中可以直接与所控制器件相连。




光敏电阻是根据光强度改变阻值的元件，本章实验我们接着上一章学过的知识，使用模拟口读取模拟值。

## 3、程序代码

程序代码在“光控声音实验”文件夹中.

```
int photocellPin = 2;       //定义变量 photocellsh=2，为电压读取端口。
int ledPin = 12;     //定义变量 ledPin=12，为 led 电平输出端口
int val = 0;            //定义 val 变量的起始值


void setup() {
   pinMode(ledPin, OUTPUT);   //使 ledPin 为输出模式
}

void loop() {
   val = analogRead(photocellPin);       //从传感器读取值
   if(val<=512){
//512=2.5V，想让传感器敏感一些的时候，把数值调高，想让传感器迟钝的时候把数值调低。
      digitalWrite(ledPin, HIGH); //当 val 小于 512(2.5V)的时候，led 亮。
   }
   else{
      digitalWrite(ledPin, LOW);
   }
}
```

## 4、下载程序

按照 arduino 教程中的程序下载方法将本程序下载到实验板中。

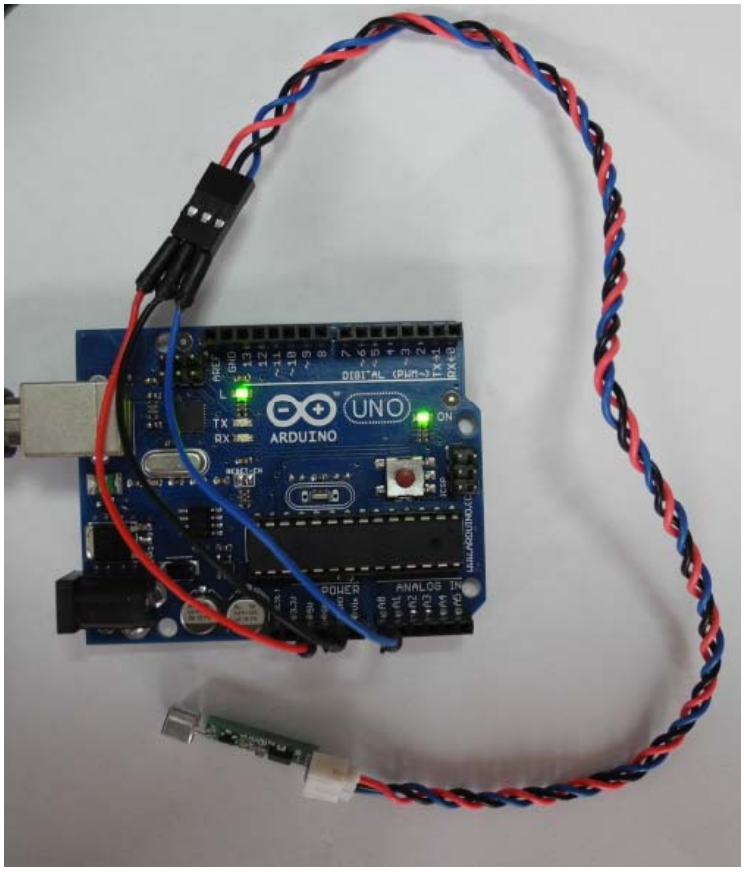
## 5、程序功能

本次实验设计的效果是，当光照正常的时候 led 灯是灭的，当周围变暗时 led 灯变亮。

因为光敏电阻受不同光照影响变化很大，所以本次实验的参数是在 60W 三基色节能灯照射下实验（无日光照射），同样亮度的日光下光敏电阻的阻值会比日光灯下低不少，估计和不同光的波段有关系。不同环境下实验使用的参数不同，大家根据原理进行调整。

# 第几章　声音传感器实验

本章我们将用到声音传感器模块。 并使用模拟口 A0，对声音的大小进行测量。

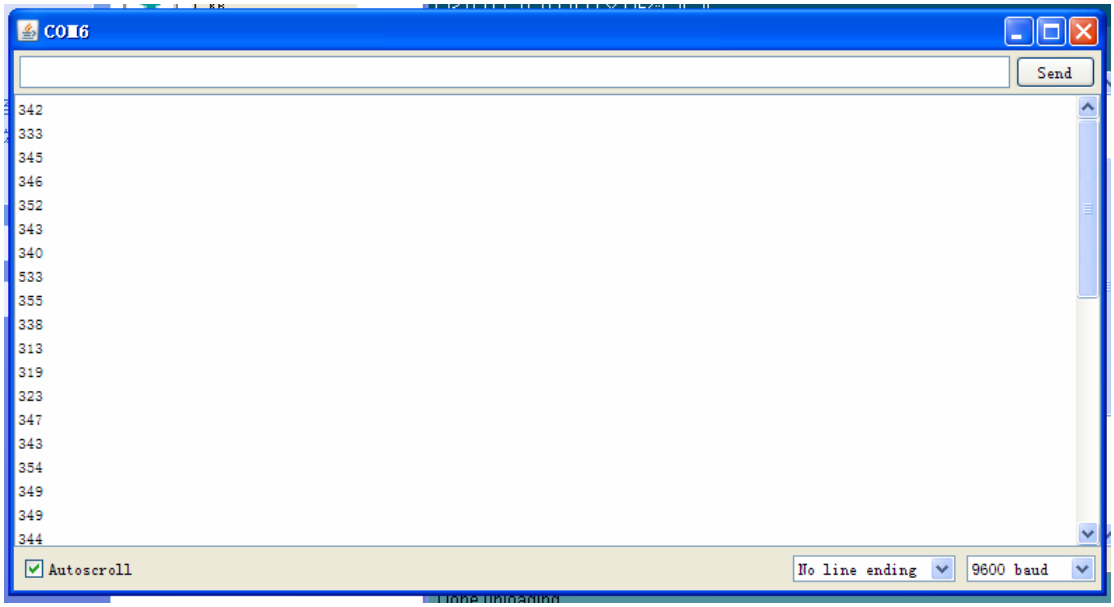



程序代码：

```
int SensorLED=13;

void setup()
{
   pinMode(SensorLED,OUTPUT); //设置数字 IO  引脚为输出模式
   Serial.begin(9600);
}
void loop()
{
   int i;//定义变量
   while(1)
   {
      i = analogRead(0); //读取模拟端口 0  电压值
      Serial.println(i,DEC);
      if(i>400)
      {
          digitalWrite(SensorLED,HIGH);//点亮小灯
```

```
        delay(1000);//延时 1 秒钟
    }
    else
    {
        digitalWrite(SensorLED,LOW);
        delay(500);//延时 0.5 秒钟
    }
  }

}
```

对着声音模块大声说话，声音越大，得出的数值越大。 当超过 400 时，小灯会亮 1 秒钟。

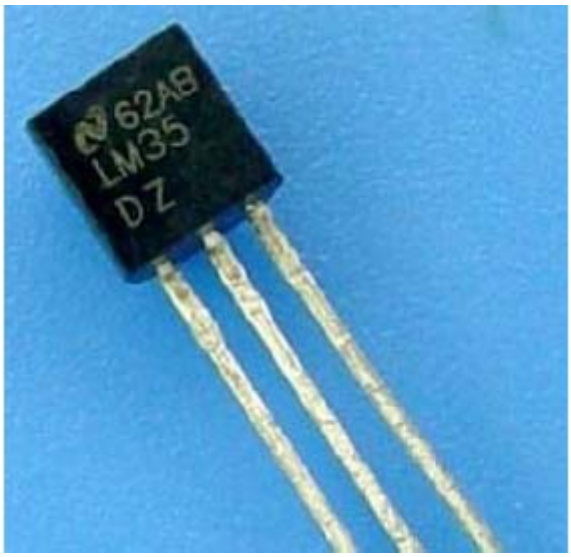


# 第十四章　LM35 温度传感器实验

## 一、温度传感器介绍

### 1、什么是温度传感器？

温度传感器就是利用物质各种物理性质随温度变化的规律把温度转换为电量的传感器。返些呈现规律性变化的物理性质主要有体。温度传感器是温度测量仪表的核心部分，品种繁多。按测量方式可分为接触式和非接触式两大类，按照传感器材料及电子元件特性分为热电阻和热电偶两类。本实验使用的是 LM35 温度传感器。如下图：



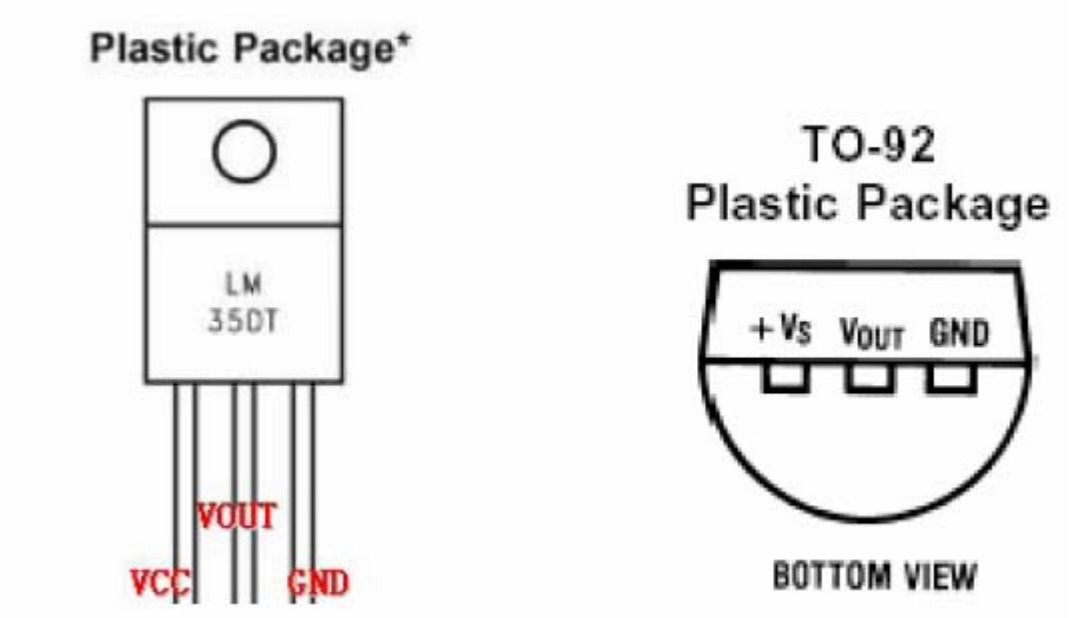

### 2、工作原理

LM35 温度传感器的输出电压不摄氏温标呈线性关系，0℃，时输出为 0V，每升高 1℃，输出电压增加 10mV。转换公式如下：

$$V_{out\_LM35}(T) = 10\,\mathrm{mV}/^{\circ}\mathrm{C} \times T^{\circ}\mathrm{C}$$

### 3、LM35 的连线

LM35 的引脚示意图如下：




从实验盒中将温度传感器拿出来可以看到，温度传感器的一面是平的，另一面是半圆的。将平面对着自己，最左边的是 VCC 引脚（接+5v），中间的为 VOUT（电压值输出引脚，接板子上的模拟引脚），最右边的引脚为 GND 引脚（接板子上的 GND）。三个引脚分别接好就可以用了。

# 二、温度报警实验

## 1、实验器件

- LM35 温度传感器模块：1 个
- 多彩面包板实验跳线：若干

## 2、实验连线

首先将实验板连接好；接着按照 LM35 温度传感器连线方法将其连好，将 VOUT 连接到模拟 0 口。这样温度报警实验的电路就连接好了。

## 3、实验原理

从LM35温度传感器的工作原理可知，温度每升高1℃，vout口输出的电压就增加10MV。根据这一原理程序中实时读出模拟 0 口的电压值，由于模拟口读出的电压值使用 0~1023 表示的，即 0V 对应 0、5V 对应 1023。

应用中， 我们只需一个 LM35 模块，利用模拟接口，将读取的模拟值转换为实际的温度。

## 4、程序代码

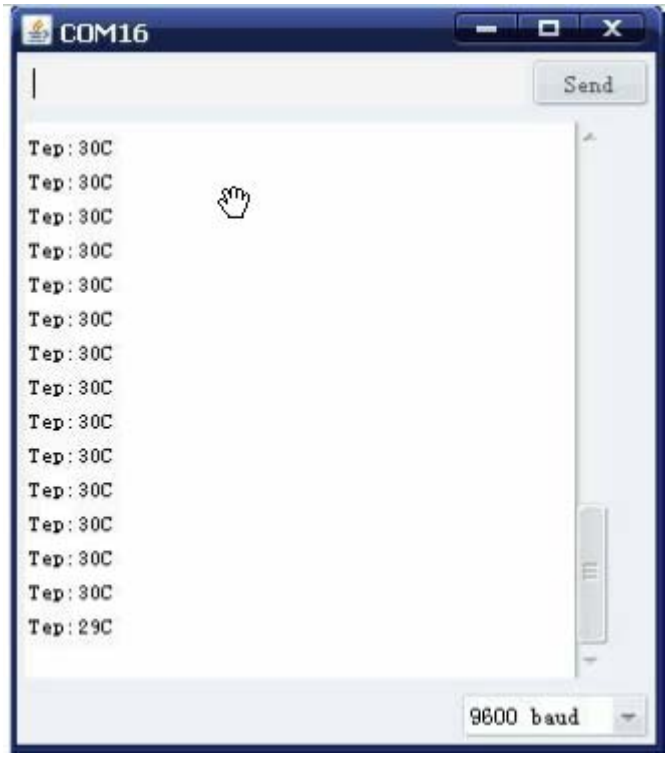
程序代码在“温度传感器”文件夹中也可以找到。.

程序代码：

```
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(potPin);//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
}
```

## 5、程序功能

将程序下载到实验板，打开监视器，就可以看到当前的环境温度了。（ 实际上，温度值有一点点偏差，要根据自己的环境温度修改一下程序，使其完全与自己的环境一致。）



# 第十五章 数码管实验

## 一、数码管介绍

### 1、认识数码管

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）；

按能显示多少个“8”可分为 1 位、2 位、4 位等等数码管；

图 15.2 各种数码管

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。

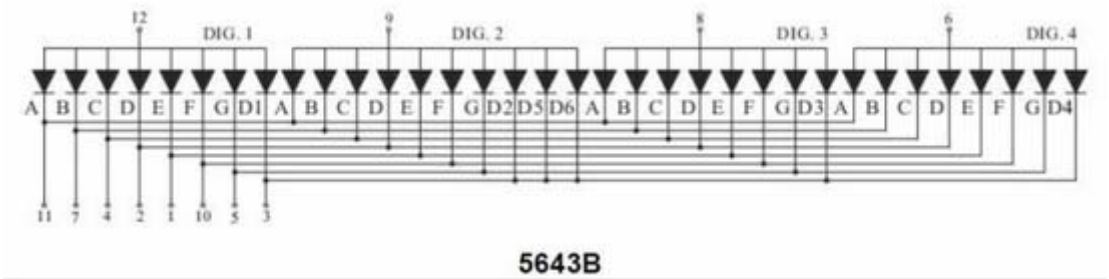


图 15.3 共阳极数码管内部结极

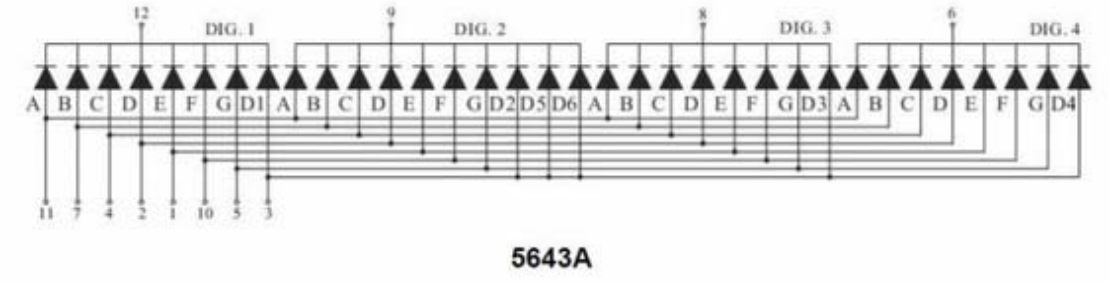



图 15.4 共阴极数码管内部结极

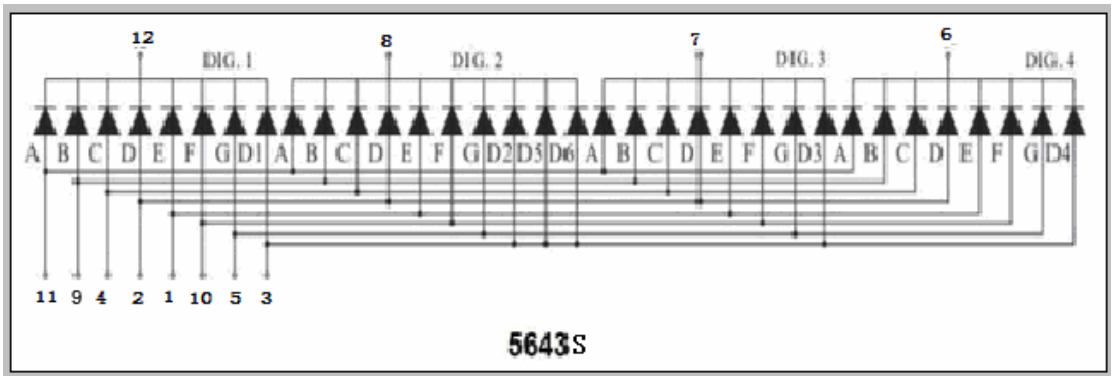



图 15.5 共阴极数码管内部结极

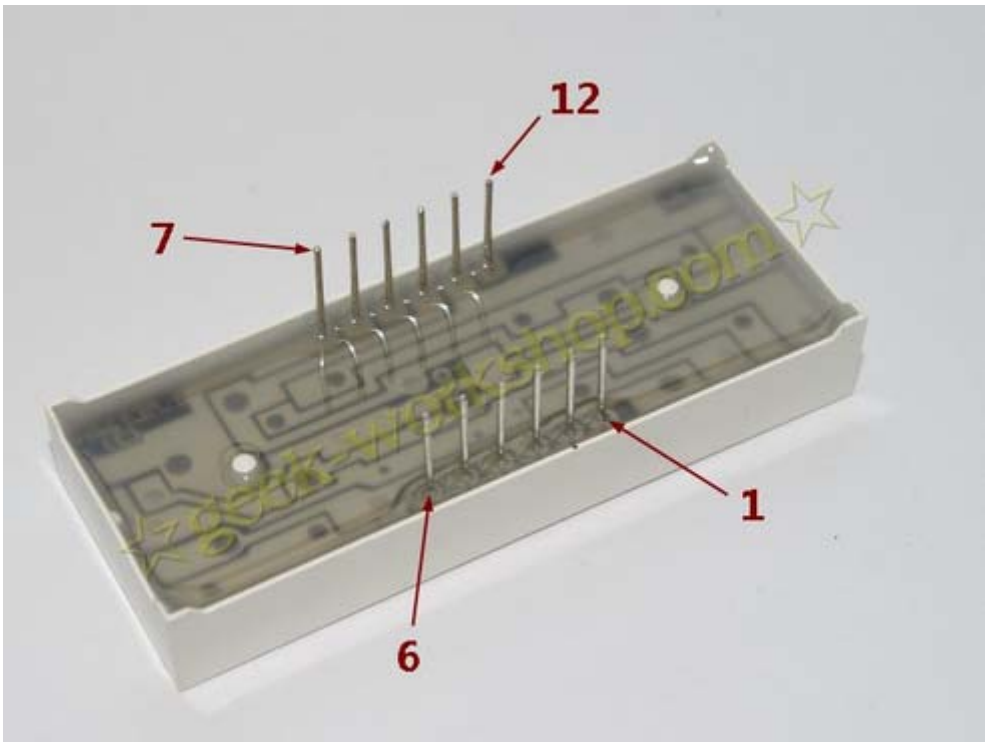
## 2、工作原理

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。本实验用的是共阳极的数码管，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

## 3、数码管的连线

将限流电阻的一端插到数字 I/O 中，另一端不数码管的字段引脚相连，剩下的六个字段和一个小数点依次按照返种方法接。将公共极 COM 如果是共阳极的就接到+5V，如果是共阴极的就接到 GND。

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。




## 二、数码管显示数字的实验

### 1、实验器件

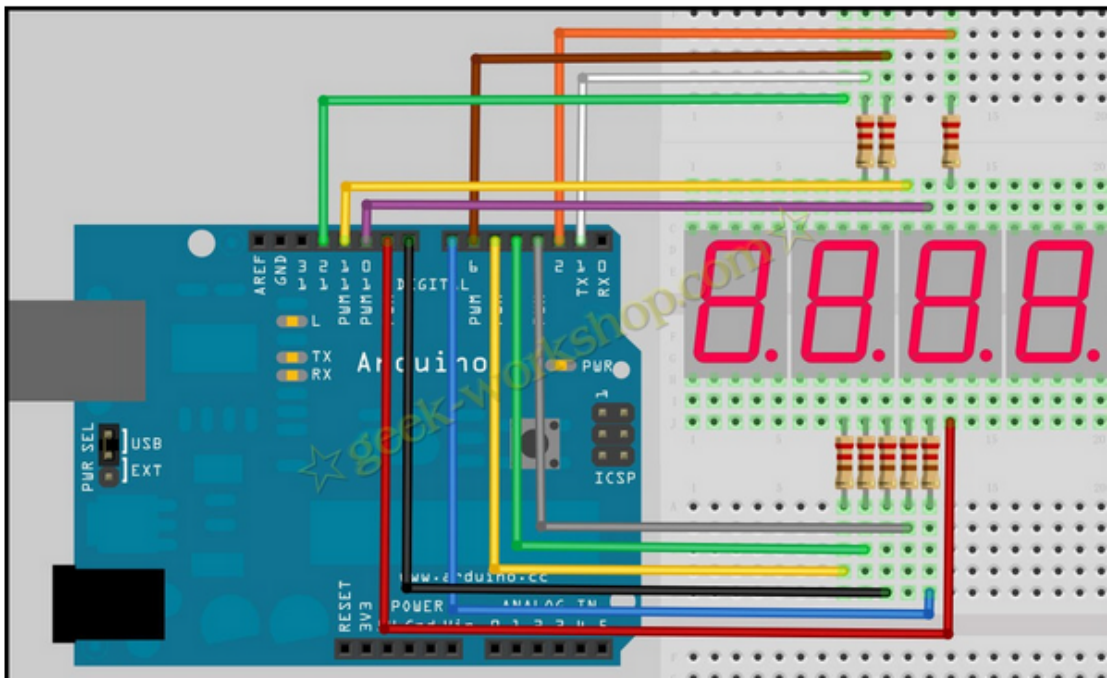
- 4 位数码管：1 个
- 220Ω 的电阻：8 个
- 多彩面包板实验跳线：若干

### 2、实验连线

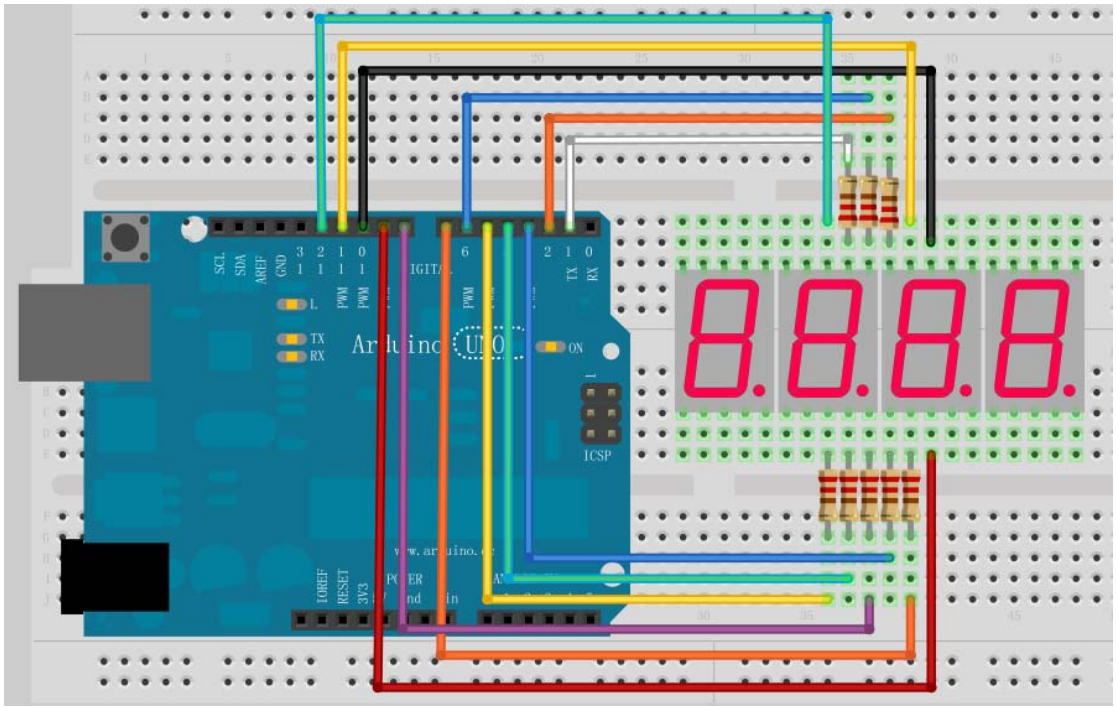
驱动数码管限流电阻肯定是必不可少的，限流电阻有两种接法，一种是在 d1-d4 阳极接，总共接 4 颗。这种接法好处是需求电阻比较少，但是会产生每一位上显示不同数字亮度会不一样，1 最亮，8 最暗。另外一种接法就是在其他 8 个引脚上接，这种接法亮度显示均匀，但是用电阻较多。本次实验使用 8 颗 220Ω 电阻。

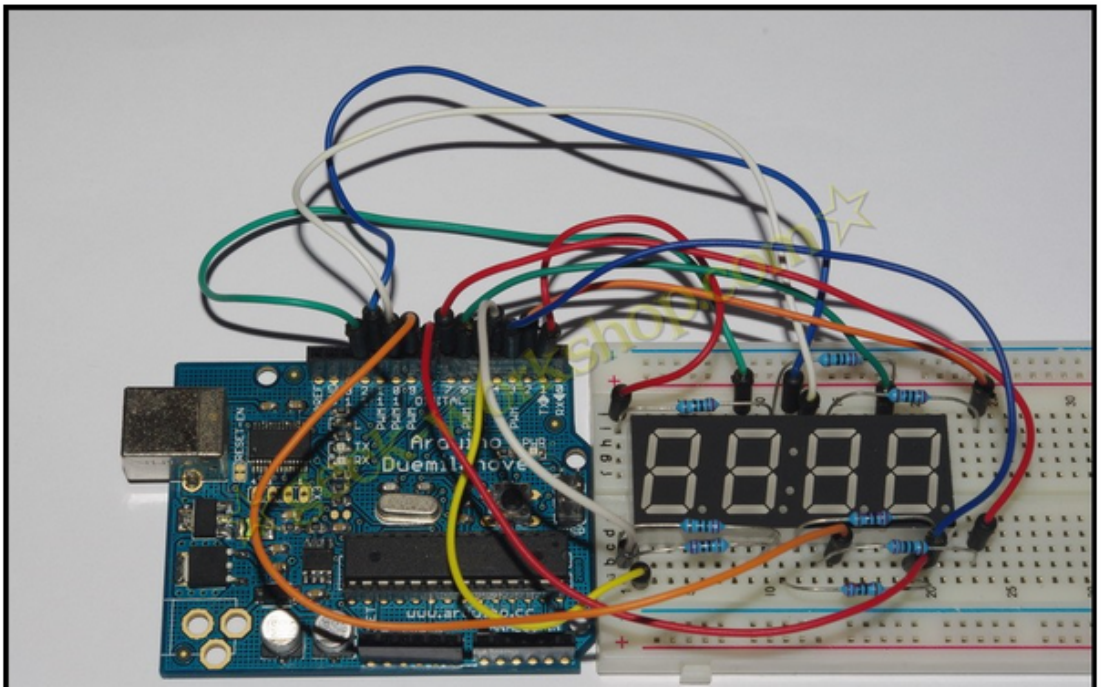
### 硬件连接图

**对于 5643A， 请参照下图接线 。**




对于 5643S, 请参照下图接线 。





## 3、程序代码

程序代码在数码管显示数字程序文件夹中，双击打开后有一个 digital_tube1 文件夹，接着双击打开后可以看见有一个 digital_tube1.pde 文件，双击图标即可打开。打开后我们可以看到返是 arduino 编程软件窗口，上面有本实验的程序代码

此代码为简易的秒表，效果如下，精准度不是很高，需要大家微调参数。

```
//设置阳极接口
int a = 1;
int b = 2;
int c = 3;
int d = 4;
int e = 5;
int f = 6;
int g = 7;
int p = 8;
//设置阴极接口
int d4 = 9;
int d3 = 10;
int d2 = 11;
int d1 = 12;
//设置变量
long n = 0;
int x = 100;
int del = 55;   //此处数值对时钟进行微调
```

```
void setup()
{
  pinMode(d1, OUTPUT);
  pinMode(d2, OUTPUT);
  pinMode(d3, OUTPUT);
  pinMode(d4, OUTPUT);
  pinMode(a, OUTPUT);
  pinMode(b, OUTPUT);
  pinMode(c, OUTPUT);
  pinMode(d, OUTPUT);
  pinMode(e, OUTPUT);
  pinMode(f, OUTPUT);
  pinMode(g, OUTPUT);
  pinMode(p, OUTPUT);
}

void loop()
{
  clearLEDs();
  pickDigit(1);
  pickNumber((n/x/1000)%10);
  delayMicroseconds(del);

  clearLEDs();
  pickDigit(2);
  pickNumber((n/x/100)%10);
  delayMicroseconds(del);

  clearLEDs();
  pickDigit(3);
  dispDec(3);
  pickNumber((n/x/10)%10);
  delayMicroseconds(del);

  clearLEDs();
  pickDigit(4);
  pickNumber(n/x%10);
  delayMicroseconds(del);

  n++;

  if (digitalRead(13) == LOW)
  {
    n = 0;
```

```
    }
}

void pickDigit(int x)    //定义 pickDigit(x),其作用是开启 dx 端口
{
    digitalWrite(d1, HIGH);
    digitalWrite(d2, HIGH);
    digitalWrite(d3, HIGH);
    digitalWrite(d4, HIGH);

    switch(x)
    {
    case 1:
        digitalWrite(d1, LOW);
        break;
    case 2:
        digitalWrite(d2, LOW);
        break;
    case 3:
        digitalWrite(d3, LOW);
        break;
    default:
        digitalWrite(d4, LOW);
        break;
    }
}

void pickNumber(int x)      //定义 pickNumber(x),其作用是显示数字 x
{
    switch(x)
    {
    default:
        zero();
        break;
    case 1:
        one();
        break;
    case 2:
        two();
        break;
    case 3:
        three();
        break;
    case 4:
```

```
      four();
      break;
   case 5:
      five();
      break;
   case 6:
      six();
      break;
   case 7:
      seven();
      break;
   case 8:
      eight();
      break;
   case 9:
      nine();
      break;
   }
}

void dispDec(int x)   //设定开启小数点
{
   digitalWrite(p, LOW);
}

void clearLEDs()   //清屏
{
   digitalWrite(a, LOW);
   digitalWrite(b, LOW);
   digitalWrite(c, LOW);
   digitalWrite(d, LOW);
   digitalWrite(e, LOW);
   digitalWrite(f, LOW);
   digitalWrite(g, LOW);
   digitalWrite(p, LOW);
}

void zero()   //定义数字 0 时阴极那些管脚开关
{
   digitalWrite(a, HIGH);
   digitalWrite(b, HIGH);
   digitalWrite(c, HIGH);
   digitalWrite(d, HIGH);
   digitalWrite(e, HIGH);
```

```
  digitalWrite(f, HIGH);
  digitalWrite(g, LOW);
}

void one()   //定义数字 1 时阴极那些管脚开关
{
  digitalWrite(a, LOW);
  digitalWrite(b, HIGH);
  digitalWrite(c, HIGH);
  digitalWrite(d, LOW);
  digitalWrite(e, LOW);
  digitalWrite(f, LOW);
  digitalWrite(g, LOW);
}

void two()   //定义数字 2 时阴极那些管脚开关
{
  digitalWrite(a, HIGH);
  digitalWrite(b, HIGH);
  digitalWrite(c, LOW);
  digitalWrite(d, HIGH);
  digitalWrite(e, HIGH);
  digitalWrite(f, LOW);
  digitalWrite(g, HIGH);
}

void three()   //定义数字 3 时阴极那些管脚开关
{
  digitalWrite(a, HIGH);
  digitalWrite(b, HIGH);
  digitalWrite(c, HIGH);
  digitalWrite(d, HIGH);
  digitalWrite(e, LOW);
  digitalWrite(f, LOW);
  digitalWrite(g, HIGH);
}

void four()   //定义数字 4 时阴极那些管脚开关
{
  digitalWrite(a, LOW);
  digitalWrite(b, HIGH);
  digitalWrite(c, HIGH);
  digitalWrite(d, LOW);
  digitalWrite(e, LOW);
```

```
  digitalWrite(f, HIGH);
  digitalWrite(g, HIGH);
}

void five()   //定义数字 5 时阴极那些管脚开关
{
  digitalWrite(a, HIGH);
  digitalWrite(b, LOW);
  digitalWrite(c, HIGH);
  digitalWrite(d, HIGH);
  digitalWrite(e, LOW);
  digitalWrite(f, HIGH);
  digitalWrite(g, HIGH);
}

void six()   //定义数字 6 时阴极那些管脚开关
{
  digitalWrite(a, HIGH);
  digitalWrite(b, LOW);
  digitalWrite(c, HIGH);
  digitalWrite(d, HIGH);
  digitalWrite(e, HIGH);
  digitalWrite(f, HIGH);
  digitalWrite(g, HIGH);
}

void seven()   //定义数字 7 时阴极那些管脚开关
{
  digitalWrite(a, HIGH);
  digitalWrite(b, HIGH);
  digitalWrite(c, HIGH);
  digitalWrite(d, LOW);
  digitalWrite(e, LOW);
  digitalWrite(f, LOW);
  digitalWrite(g, LOW);
}

void eight()   //定义数字 8 时阴极那些管脚开关
{
  digitalWrite(a, HIGH);
  digitalWrite(b, HIGH);
  digitalWrite(c, HIGH);
  digitalWrite(d, HIGH);
  digitalWrite(e, HIGH);
```

```
  digitalWrite(f, HIGH);
  digitalWrite(g, HIGH);
}

void nine()   //定义数字 9 时阴极那些管脚开关
{
  digitalWrite(a, HIGH);
  digitalWrite(b, HIGH);
  digitalWrite(c, HIGH);
  digitalWrite(d, HIGH);
  digitalWrite(e, LOW);
  digitalWrite(f, HIGH);
  digitalWrite(g, HIGH);
}
```

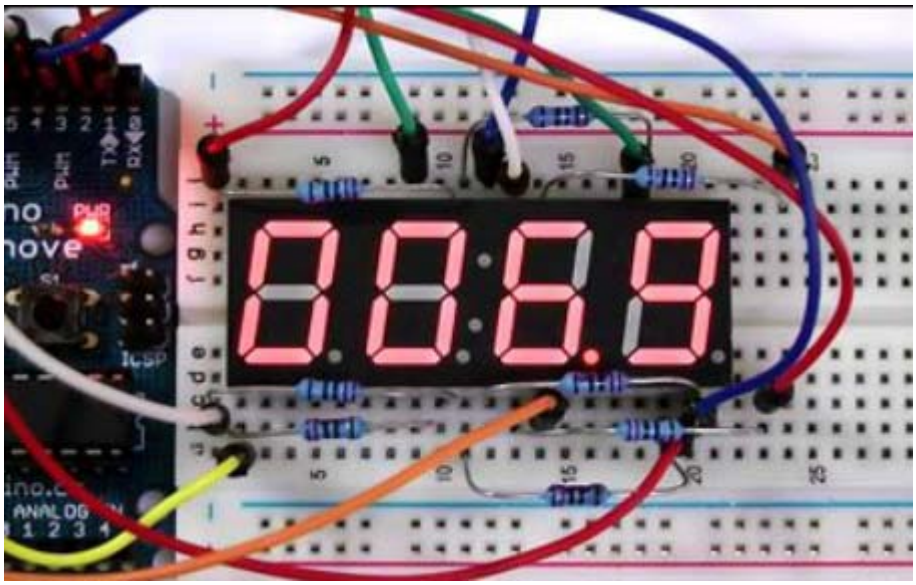
在 setup()前面定义了一系列的数字显示子程序，返些子程序的定义可以方便在 loop()中使用，使用时只需将子程序的名写上即可。

## 5、下载程序

按照 arduino 教程中的程序下载方法将本程序下载到实验板中。

## 6、程序功能

简易的秒表

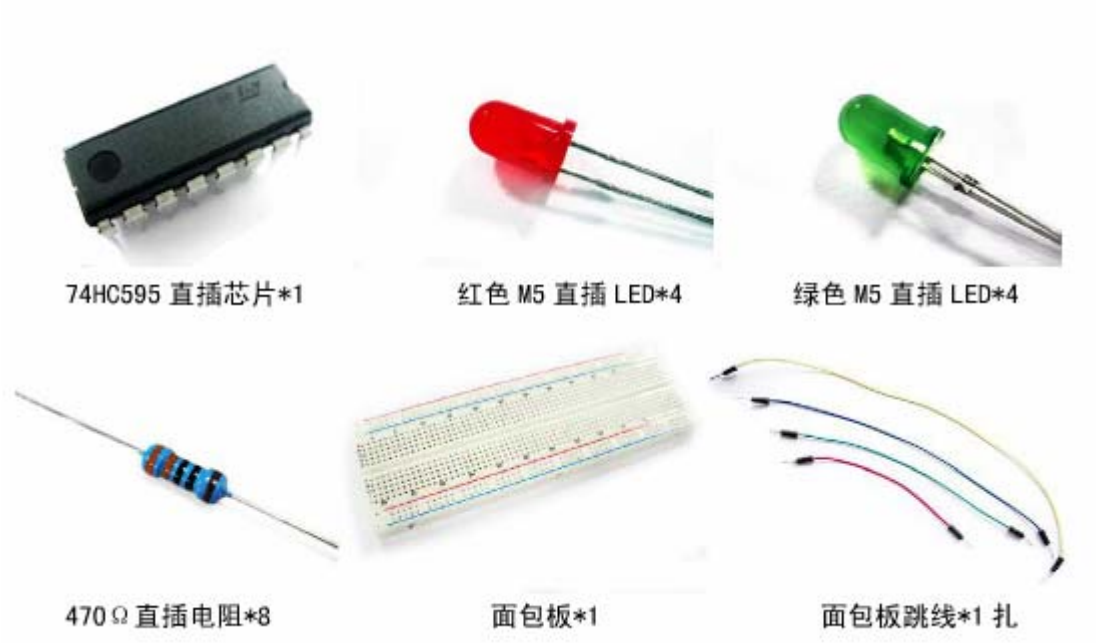


# 第十六章 74HC595 应用实验

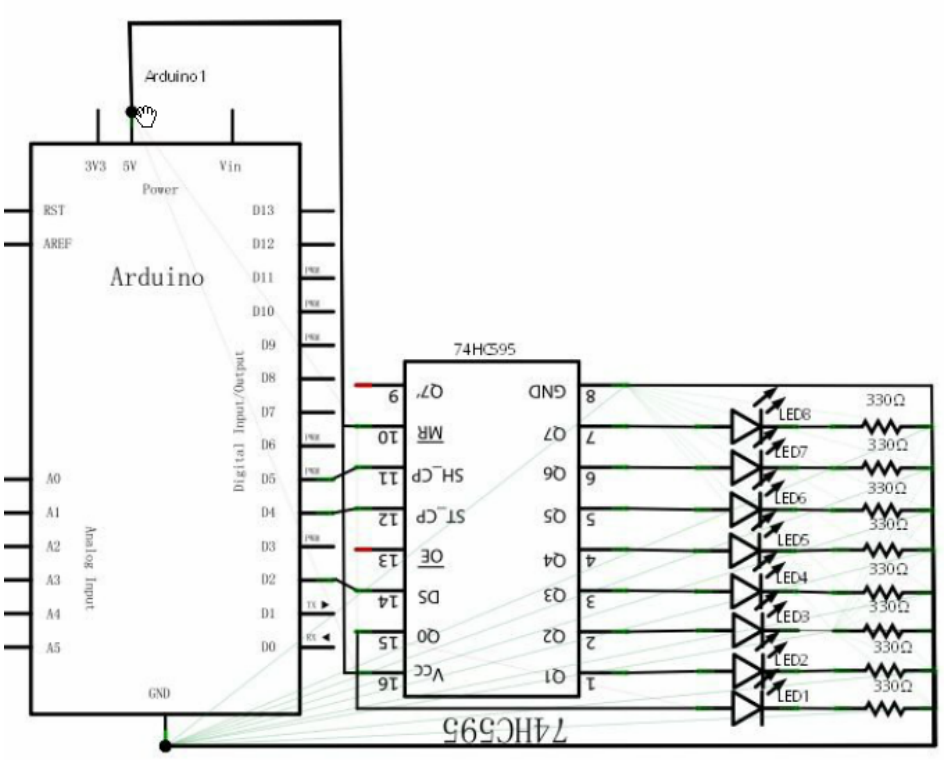
1）概述

74HC595 是具有 8 位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？ 一定会有很多朋友问这个问题。 回答这个问题， 我想问朋友们，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 uno 有几个 I/O 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。用了 74HC595 芯片以后，我们可以用 3 个数字 I/O 口控制 8 个 LED 小灯， 岂不美哉！

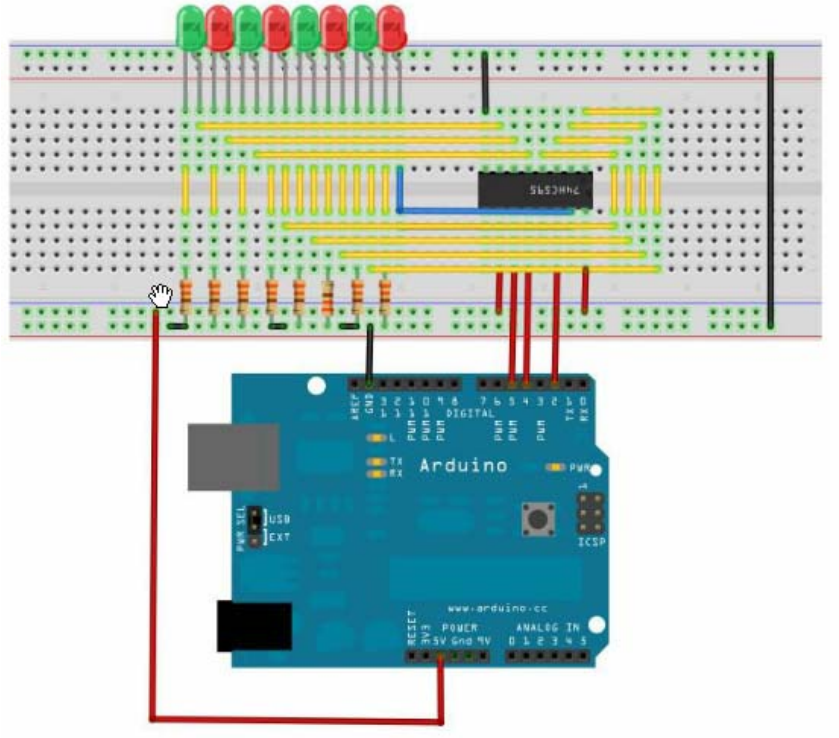
下面准备实验元件。




准备好元件，我们按下图的原理图连接电路。




此电路图看似复杂，我们仔细分析以后结合参考实物就会发现很简单。




实例程序：

```
const int ON =    HIGH ;
const int OFF = LOW ;
```

```
int latchPin = 5;                 //接 595 的脚位 12
int clockPin = 4;                 //5 接 595 的脚位 11
int dataPin = 2;                  //接 595 的脚位 14
//595 的脚位 16 接 5VDC
//595 的脚位 8 接 GND

int ledState = 0;

void setup() {
   pinMode(latchPin, OUTPUT);
   pinMode(clockPin, OUTPUT);
   pinMode(dataPin, OUTPUT);
}
void loop() {
   int delayTime = 100 ;
   for(int i=0;i<256;i++)
   {
       updateLEDs(i);
       delay(delayTime);
   }
}
void updateLEDs(int value)
{
   digitalWrite(latchPin,LOW);
   shiftOut(dataPin, clockPin, MSBFIRST, value);
   digitalWrite(latchPin,HIGH);
}

void updateLEDsLong(int value)
{
   digitalWrite(latchPin,LOW);
   for(int i=0;i<8;i++)
   {
      int bit = value&B10000000;
      value = value<<1;
      if(bit==128)
      {
            digitalWrite(dataPin,HIGH);
      }
      else
      {
            digitalWrite(dataPin,LOW);
      }
      digitalWrite(clockPin,HIGH);
```

```
      delay(1);
      digitalWrite(clockPin,LOW);
   }
   digitalWrite(latchPin,HIGH);
}

int bits[] = {B00000001,B00000010,B00000100,B00001000,B00010000,B00100000,
B01000000,B10000000};
int masks[] = {B11111110,B11111101,B11111011,B11110111,B11101111,B11011111,
B10111111,B01111111};
void changeLED(int led,int state)
{
   ledState = ledState & masks[led];
   if(state == ON){ ledState = ledState|bits[led]; }
   updateLEDs(ledState);
}
```

下载完程序到控制板后， 我们可以看到 8 个小灯闪烁的美妙场景。

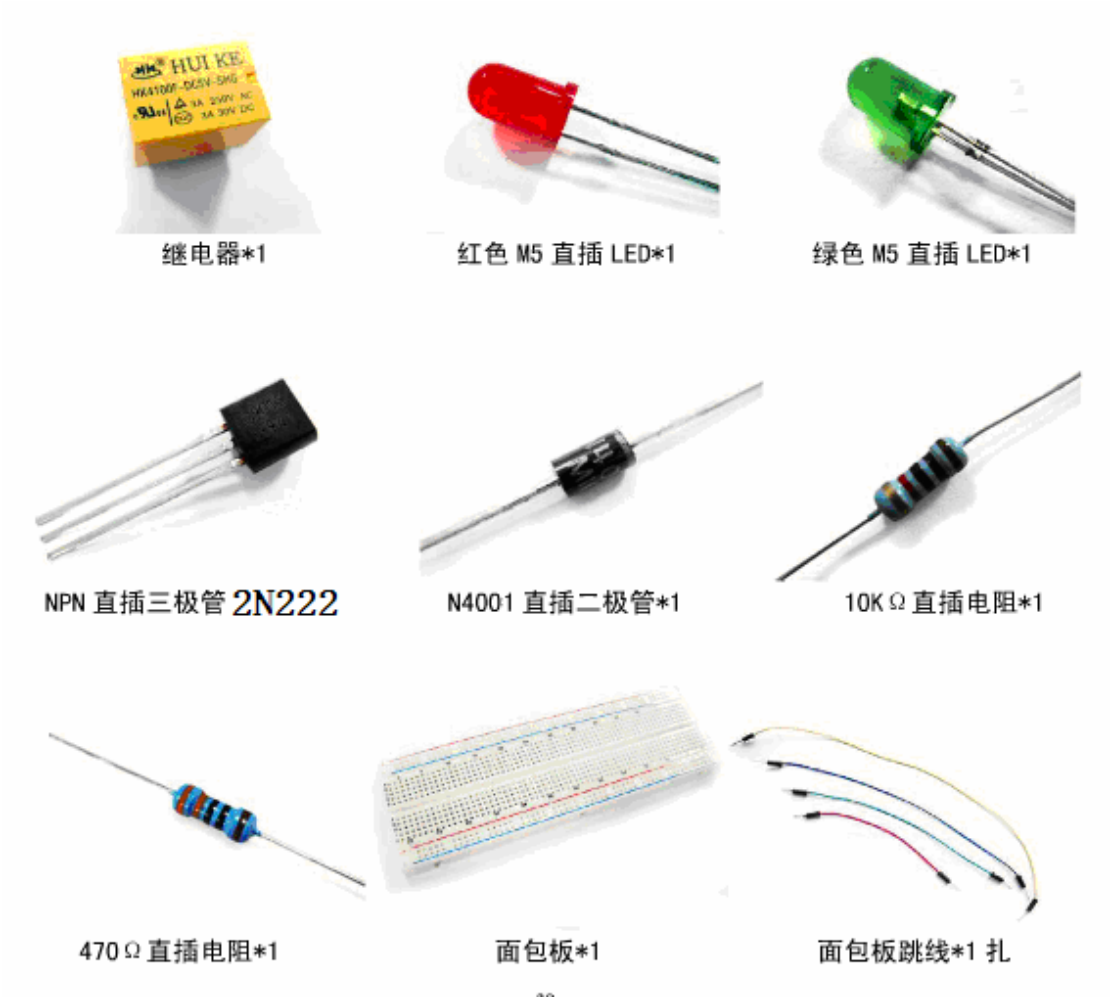
# 第十七章 继电器控制实验

1）概述

继电器是一种当输入量（电、磁、声、光、热）达到一定值时，输出量将发生跳跃式变化的自动控制器件。在生活中我们常需要用弱电控制强电的情况，也就是常说的小电流控制大电流问题。 比喻说用 Arduino 控制器控制风扇之类的大功率电器时我们就要用到继电器了。继电器的工作原理这里就不多说了。

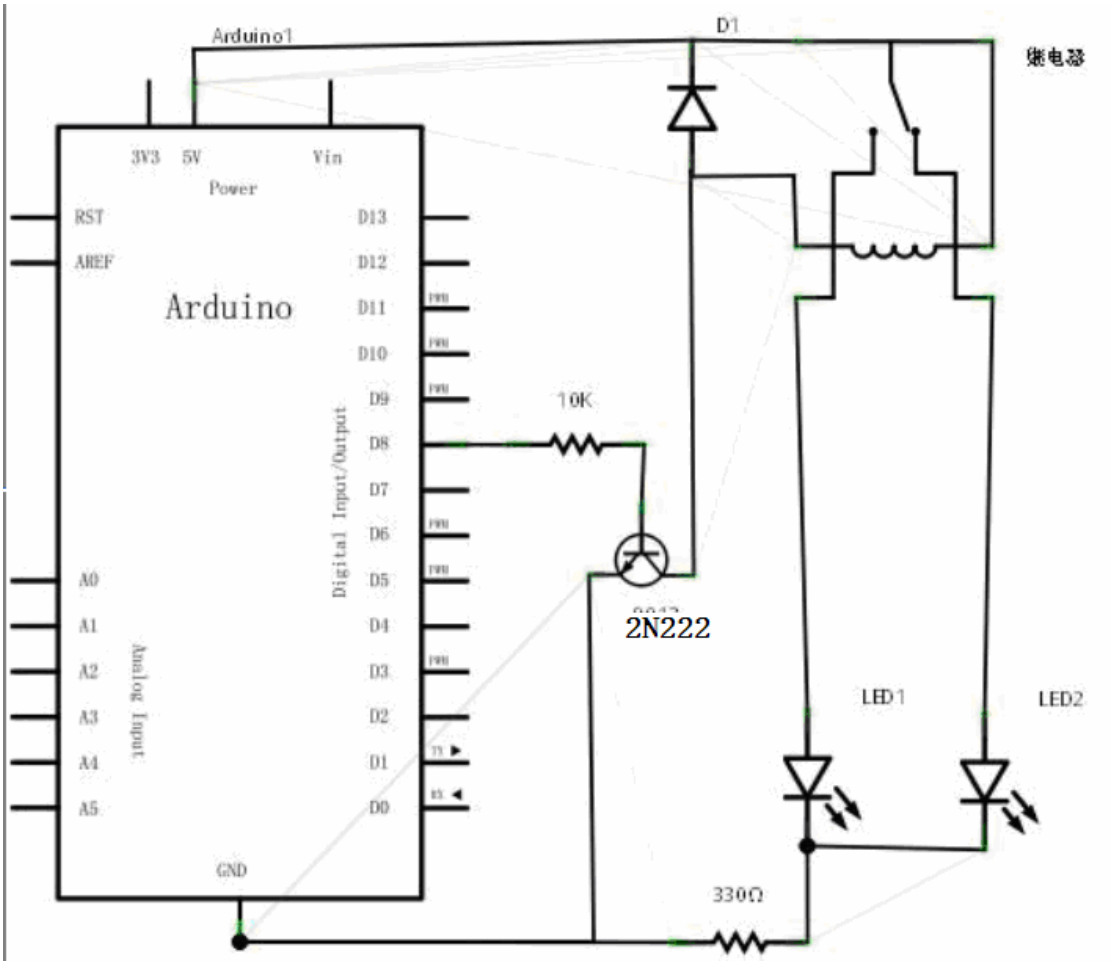
对于初学者，为安全起见，本实验我们不动用大功率电器了。 以小见大， 我们采用 LED 小灯来完成实验演示。

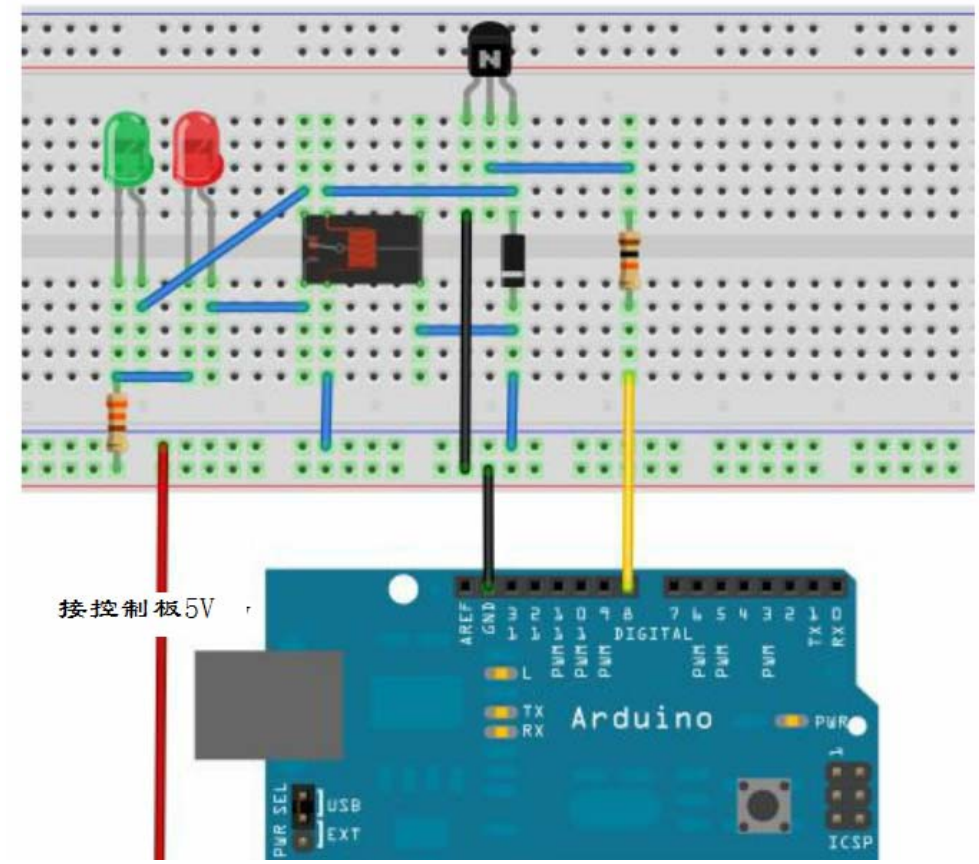
2） 实验准备




实验用到的元件增加，所以搭建的电路也相对较复杂。 继电器有两路开关：一路常闭，一路常开，实际应用中我们通常只使用一路。

我们将用红绿两个 LED 小灯来表示两路开关的开合状态，电阻使用 220 欧姆。按照以下原理图连接电路。


Arduino1
D1
继电器
3V3
5V
Vin
Power
RST
AREF
Arduino
D13
D12
D11
D10
D9
D8
D7
D6
D5
D4
D3
D2
D1
D0
Digital Input/Output
A0
A1
A2
A3
A4
A5
Analog Input
GND
10K
2N222
LED1
LED2
330Ω


接控制板5V
AREF
GND
DIGITAL
L
TX
RX
Arduino
PWR
USB
EXT
PWR SEL
ICSP

在连接电路过程中，我们要注意的是明确继电器的引脚位置，其次是 IN4001 二极管是有正负极之分的。别看继电器电路稍有复杂，kiev 程序却是很简单的。继电器属于数字信号模块，我们通过给三极管数字信号使继电器开合来控制大功率设备，这里我们用 LED 小灯当作大功率设备吧。

程序中，我们使用数字端口 8，输出高电平并延时 1 秒后，输出低电平 1 秒，即为开关断开 1 秒再接通 1 秒。

3）程序代码：

```
int relayPin = 8 ;//定义数字接口 8，连接三极管基级
void setup()
{
    pinMode(relayPin,OUTPUT);//设置 relayPin 接口为输出模式
}
void loop()
{
  digitalWrite(relayPin,HIGH);//驱动继电器闭合导通
  delay(1000);//延时 1 秒钟
  digitalWrite(relayPin,LOW);//驱动继电器断开
  delay(1000);//延时 1 秒钟
}
```

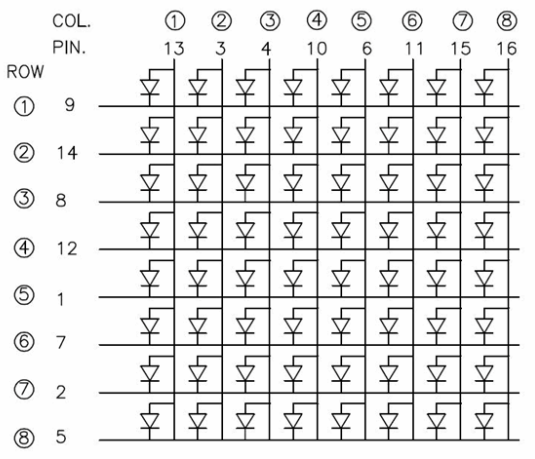
4） 程序结果：

我们将看到 红色小灯 和 绿色小灯轮番闪烁。本章实验到此结束，希望大家多多开动脑筋应用到自己的互动作品中去。

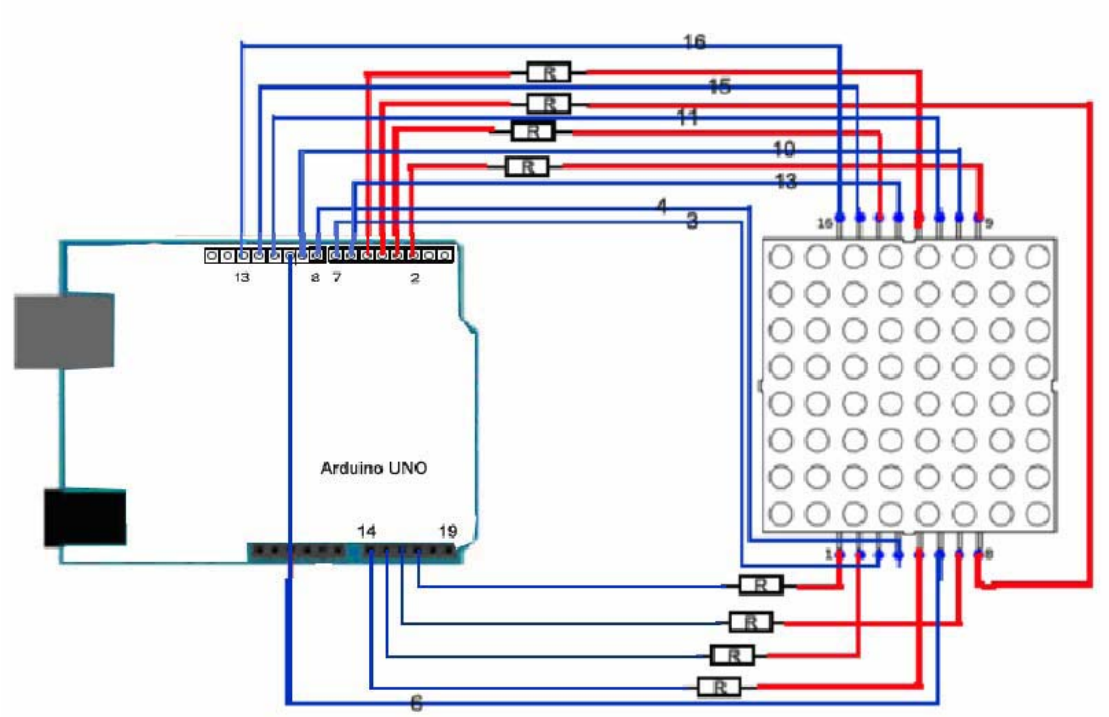
# 第十八章 8x8 矩阵LEDs实验

1）概述

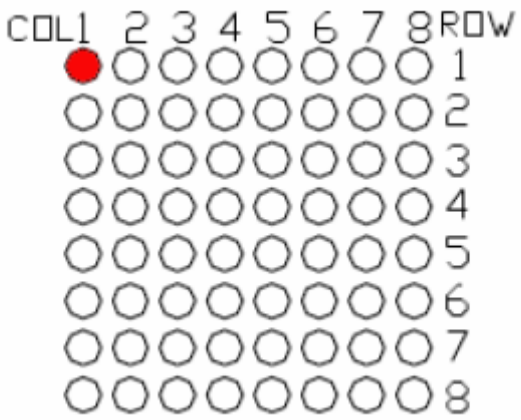
下图是矩阵 LED 的内部原理图。




本实验的连线图。




点亮 8X8 点阵 LED 的一个 LED 如下：

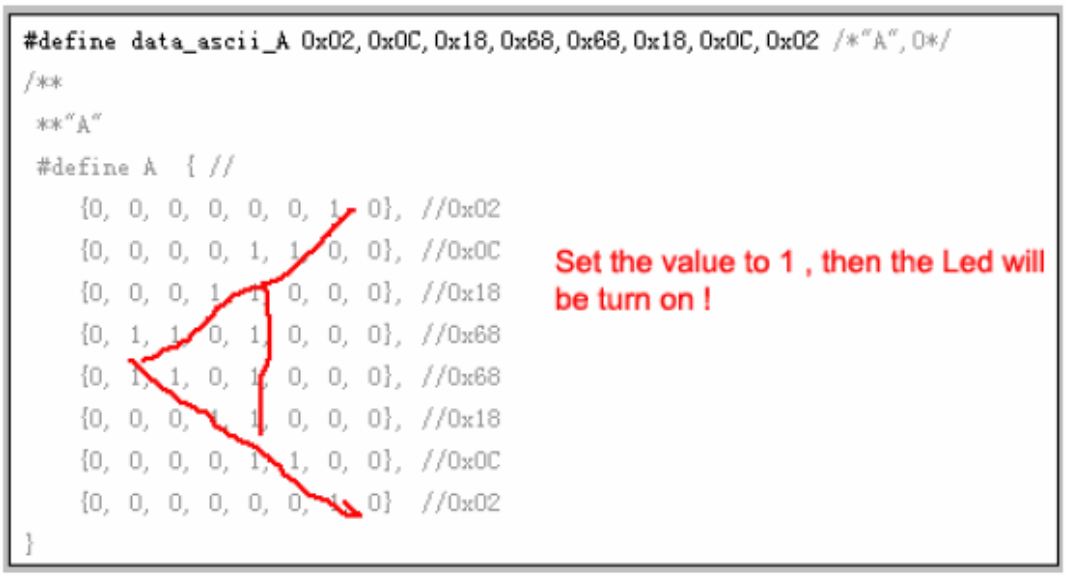



实例代码：

```
//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 。




代码在文件夹—“8x8 矩阵 LEDs 实验” ，大家可以以此为参看，做出更多多彩的实验。

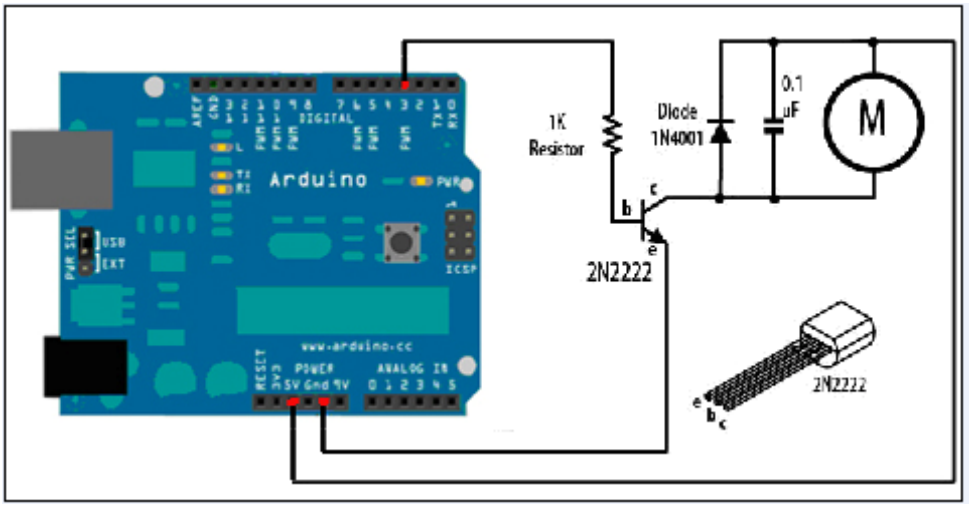
# 第十九章　PWM电机控制实验

1）概述

本章 使用 三极管驱动直流电机 并使用 PWM, 控制电机的速度快慢。

2） 实验原理图

实验中使用数字端口 3，控制电机。




2) 实验代码

```
const int motorPin = 3; // vibration motor transistor is connected to pin 3

int shuDu = 0;//定义整数型变量 shuDu 与其初始值
int addNum = 50;

void setup()
{
pinMode(motorPin, OUTPUT);
}
void loop()
{
   analogWrite(motorPin, shuDu);//把 shuDu 的值写入 3 号端口

   shuDu = shuDu + addNum;//改变 shuDu 值，使速度在下一次循环发生改变

   if (shuDu == 0 || shuDu == 255) {
      addNum = -addNum ; //在速度最高与最低时进行翻转
   }

   delay(500); //延时 500 毫秒
}
```

这个实验，我们将看到电机每 500 毫秒将加速一次，到了最大速度后，又变为每 500 毫秒减速一次。

# 第二十章　I2C 1602 液晶显示屏实验

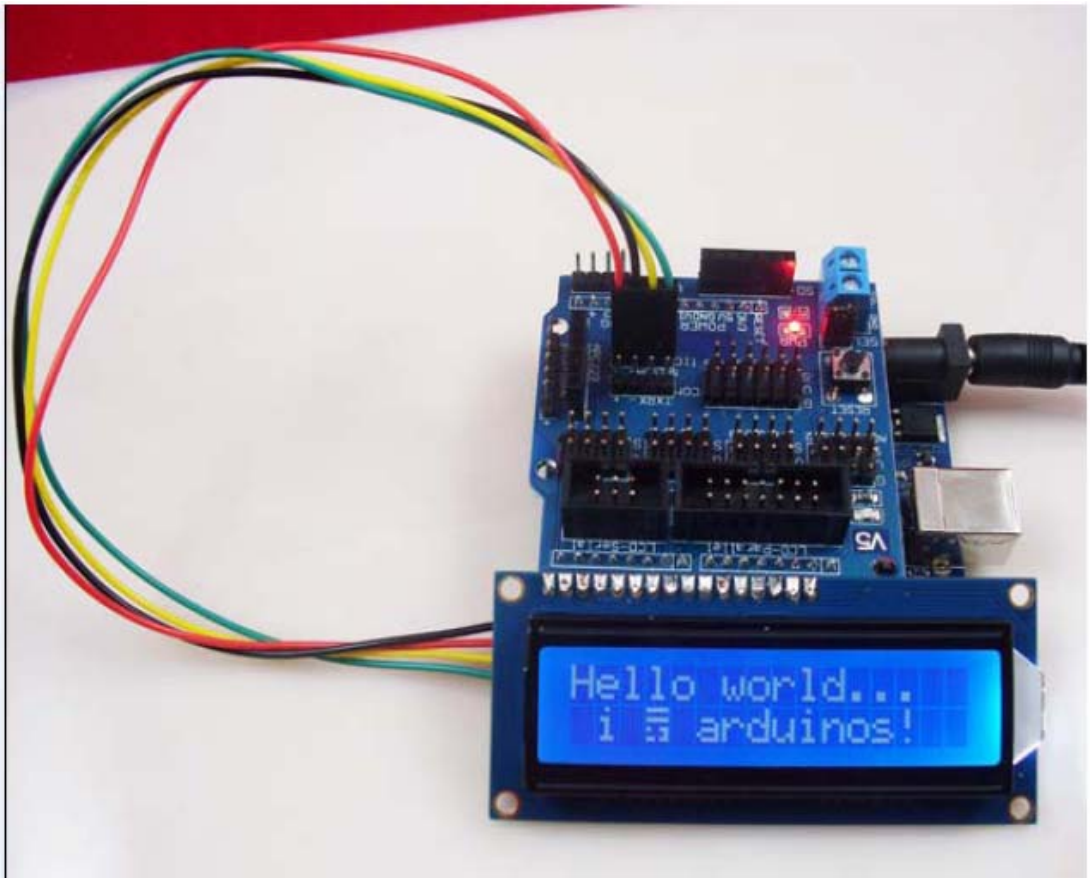
1)概述

串口的端口为 ：　SCL 接 Arduino(A4), SDA 接 Arduino（A5）。

IIC LCD1602 液晶模块,可以显示 2 行,每行 16 个字符。对于 Arduino 初学者来说,不必为繁琐复杂液晶驱动电路连线而头疼了，,I2C 只需两根线就可以实现数据显示。

在这里只需使用扩展板，连接就可以了。

2）实验连线。



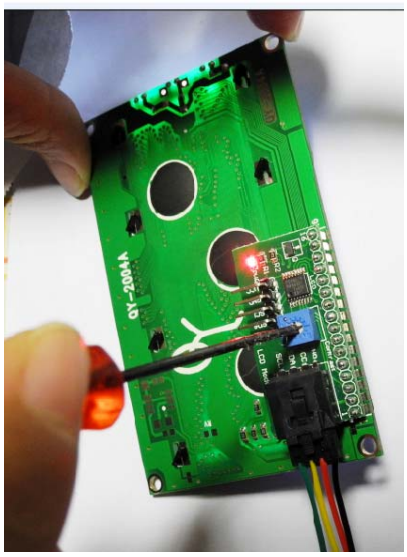

编译程序之前， 要把“LiquidCrystal_I2C”压缩包解压到 Arduino 安装目录下的类库文件夹里才能使用。

3） 实验代码

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

LiquidCrystal_I2C lcd(0x27,16,2);  // set the LCD address to 0x27 for a 16 chars and 2 line display
```

```
void setup()
{
   lcd.init();                                  // initialize the lcd
   // Print a message to the LCD.
   lcd.backlight();
   lcd.setCursor(1, 0);
   lcd.print("Welcome to ");
   lcd.setCursor(0, 1);
   lcd.print("Teng Bo Ke Ji");
}

void loop()
{
}
```

下载完程序后，按一下 reset 键，然后调节对比度。 模块后面有个蓝色的旋钮电阻，调节该电阻，可以清晰看到显示的字符 。

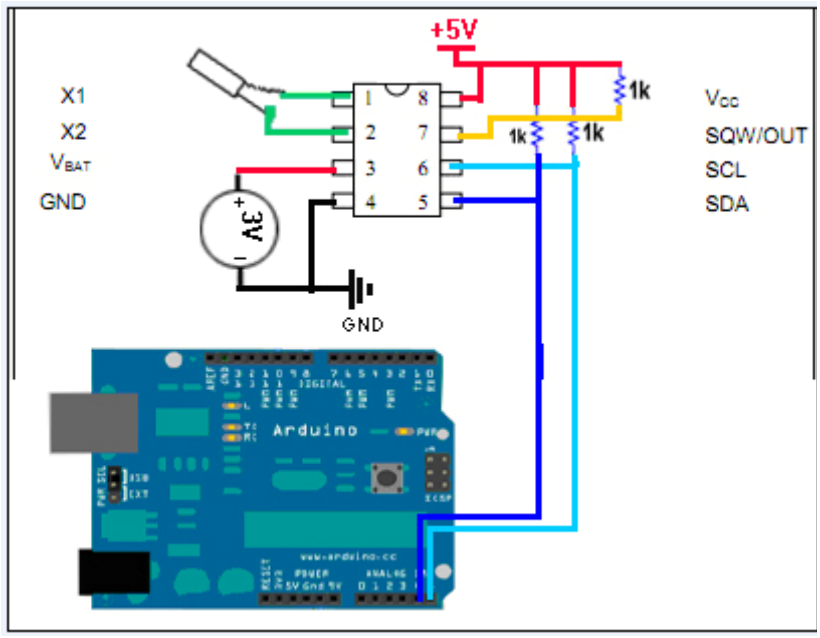


# 第二十一章　DS1307 时间实验

1）概述

是一款低功耗，具有 56 字节非失性 RAM 的全 BCD 码时钟日历实时时钟芯片，地址和数据通过两线双向的串行总线的传输，芯片可以提供秒，分，小时等信息，每一个月的天数能自动调整。并且有闰年补偿功能。 AM/PM 标志位决定时钟工作于 24 小时或 12 小时模式，芯片有一个内置的电源感应电路，具有掉电检测和电池切换功能

2）实验连线：




**实验中，如果我们可以不用纽扣电池，使用 1K 欧姆电阻，直接将 DS1307 的 3 脚连接到 Arduino 控制板的+3.3V 接口，DS1307 芯片的 4 脚接地。**

4） 程序代码：

解压“DS1307.rar”文件到 Arduino 安装文件夹下的类库文件夹 。

```
#include <DS1307.h>

//初始化 DS1307 环境
DS1307 rtc(4, 5);

void setup()
{
  Serial.begin(9600);
  // Set the clock to run-mode
  rtc.halt(false);
```

```
    //设置 DS1307 参数
   rtc.setDOW(SUNDAY);           // 设置星期参数（Day-of-Week），这里是星期日
   rtc.setTime(12, 0, 0);        // 设置时间（24 小时制），这里是中午 12 点正
   rtc.setDate(23, 6, 2012);   // 设置日期参数，这里是 2012 年 6 月 23 日

   // Set SQW/Out rate to 1Hz, and enable SQW
   rtc.setSQWRate(SQW_RATE_1);
   rtc.enableSQW(true);
}

void loop()
{
   // 显示时间
   Serial.println(rtc.getTimeStr());

   // 显示星期 和 日期
   Serial.print(rtc.getDOWStr(FORMAT_SHORT));
   Serial.print("    ");
   Serial.println(rtc.getDateStr());

   //一秒之后读取数据并显示
   delay (1000);
}
```

下载程序到控制板成功后，打开监控器， 我们将会看到， 每一秒钟， 屏幕显示一次:

**时间**

**星期 和 日期**

# 第二十二章　红外遥控实验

## 一、红外接收头介绍

### 1、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,只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.

### 2、工作原理

内置接收管将红外发射管发射出来的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、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。

### 3、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。

# 二、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
- 蜂鸣器：1 个
- 220Ω 电阻：1 个
- 多彩面包线：若干

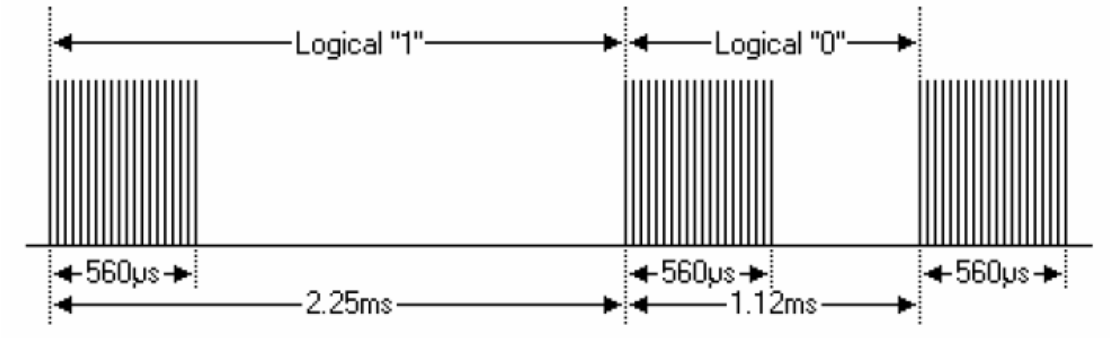
## 3、实验原理

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，这就叫知己知彼，百战不殆 。本产品使用的遥控器的编码方式为：NEC 协议。下面就介绍一下 NEC 协议：

**• NEC 协议介绍：**

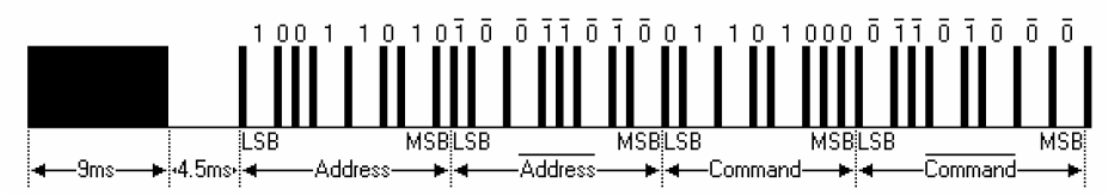
特点：
（1）8 位地址位，8 位命令位
（2）为了可靠性地址位和命令位被传输两次
（3）脉冲位置调制
（4）载波频率 38khz
（5）每一位的时间为 1.125ms 或 2.25ms

**• 逻辑 0 和 1 的定义如下图：**



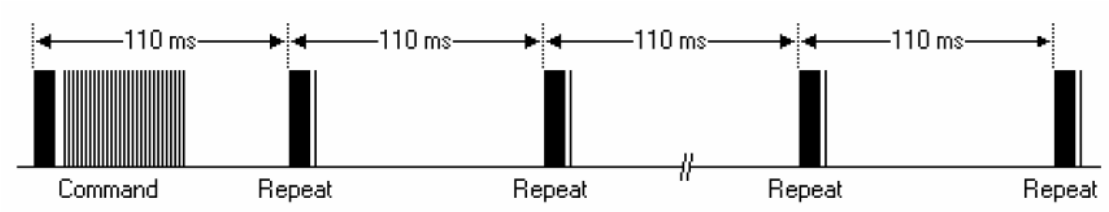

**协议如下：**

**• 按键按下立刻松开的发射脉冲：**



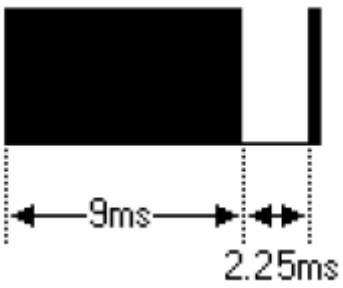
上面的图片显示了 NEC 的协议典型的脉冲序列。注意：这是首先发送 LSB（最低位）的协议。在上面的脉冲传输的地址为 0x59 命令为 0x16。一个消息是由一个 9ms 的高电平开始，随后有一个 4.5ms 的低电平，（这两段电平组成引导码）然后由地址码和命令码。地址和命令传输两次。第二次所有位都取反，可用于对所收到的消息中的确认使用。总传输时间是恒定的，因为每一点不它取反长度重复。如果你不感兴趣，你可以忽略这个可靠性取反，也可以扩大地址和命令，以每 16 位！

**• 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 的脉冲不上图一样，之后每 110ms 重复代码传输一次。这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 的高电平组成。

**• 重复脉冲**



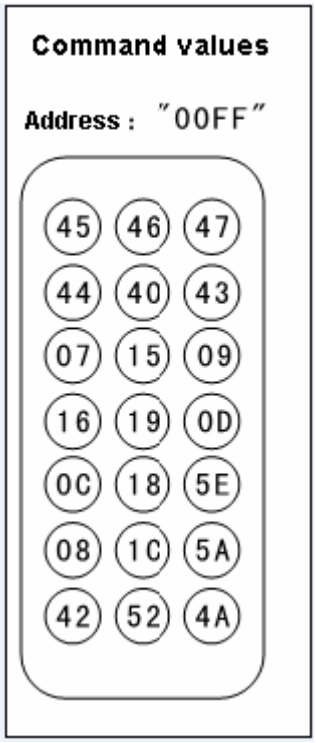

本介绍是参考http://www.sbprojects.com/knowledge/ir/nec.htm

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进行解码、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。

**本实验编程思想**

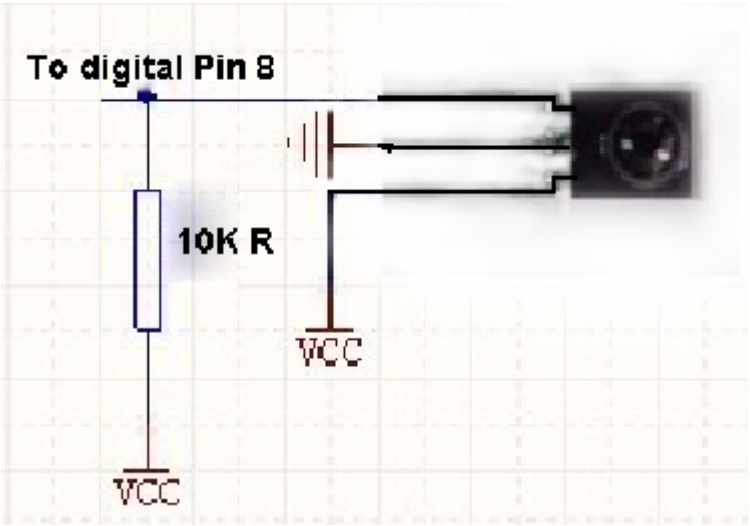
根据 NEC 编码的特点和接收端的波形，本实验将接收端的波形分成四部分：引寻码（9ms 和 4.5ms 的脉冲）、地址码 16 位（包括 8 位的地址位和 8 位的地址的取反）、命令码 16 位（包括 8 位命令位和 8 位命令位的取反）、重复码（9ms、2.25ms、560us 脉冲组成）。利用定时器对接收到的波形的高电平段和低电平段进行测量，根据测量到的时间来区分：逻辑“0”、逻辑“1”、引寻脉冲、重复脉冲。引导码和地址码只要判断是正确的脉冲即可，不用存储，但是命令码必须存储，因为每个按键的命令码都不同，

键码值 ：




## 4 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 接到数字端口 8 口，参照以下图：




与 LCD 1602 的连线图， 如下图。

**实验例程图：**

## 5、程序代码

```
#define IR_IN    8    //红外接收

int Pulse_Width = 0;//存储脉宽
int ir_code = 0x00;//  用户编码值
char    adrL_code = 0x00;//命令码
char    adrH_code = 0x00;//命令码反码

void timer1_init(void)//定时器初始化函数
{
   TCCR1A = 0X00;
   TCCR1B = 0X05;//给定时器时钟源
   TCCR1C = 0X00;
   TCNT1 = 0X00;
   TIMSK1 = 0X00;       //禁止定时器溢出中断
}
void remote_deal(void)//执行译码结果函数
{     //数据显示
       Serial.println(ir_code,HEX);//16 进制显示
       Serial.println(adrL_code,HEX);//16 进制显示
}
char logic_value()//判断逻辑值“0”和“1”子函数
{
   TCNT1 = 0X00;
   while(!(digitalRead(IR_IN))); //低等待
   Pulse_Width=TCNT1;
   TCNT1=0;
   if(Pulse_Width>=7&&Pulse_Width<=10)//低电平 560us
   {
      while(digitalRead(IR_IN));//是高就等待
      Pulse_Width=TCNT1;
      TCNT1=0;
      if(Pulse_Width>=7&&Pulse_Width<=10)//接着高电平 560us
         return 0;
      else if(Pulse_Width>=25&&Pulse_Width<=27) //接着高电平 1.7ms
         return 1;
   }
   return -1;
}
```

```
void pulse_deal()//接收地址码和命令码脉冲函数
{
  int i;
  int j;
  ir_code=0x00;// 清零
  adrL_code=0x00;// 清零
  adrH_code=0x00;// 清零

  //解析遥控器编码中的用户编码值
  for(i = 0 ; i < 16; i++)
  {
    if(logic_value() == 1) //是 1
        ir_code |= (1<<i);//保存键值
  }
  //解析遥控器编码中的命令码
  for(i = 0 ; i < 8; i++)
  {
    if(logic_value() == 1) //是 1
      adrL_code |= (1<<i);//保存键值
  }
  //解析遥控器编码中的命令码反码
  for(j = 0 ; j < 8; j++)
  {
    if(logic_value() == 1) //是 1
        adrH_code |= (1<<j);//保存键值
  }
}
void remote_decode(void)//译码函数
{
  TCNT1=0X00;
  while(digitalRead(IR_IN))//是高就等待
  {
    if(TCNT1>=1563)  //当高电平持续时间超过 100ms，表明此时没有按键按下
    {
      ir_code=0x00ff;// 用户编码值
      adrL_code=0x00;//键码前一个字节值
      adrH_code=0x00;//键码后一个字节值
      return;
    }
  }

  //如果高电平持续时间不超过 100ms
  TCNT1=0X00;
```

```
  while(!(digitalRead(IR_IN))); //低等待
  Pulse_Width=TCNT1;
  TCNT1=0;
  if(Pulse_Width>=140&&Pulse_Width<=141)//9ms
  {

    while(digitalRead(IR_IN));//是高就等待
    Pulse_Width=TCNT1;
    TCNT1=0;
    if(Pulse_Width>=68&&Pulse_Width<=72)//4.5ms
    {
      pulse_deal();
      return;
    }
    else if(Pulse_Width>=34&&Pulse_Width<=36)//2.25ms
    {
      while(!(digitalRead(IR_IN)));//低等待
      Pulse_Width=TCNT1;
      TCNT1=0;
      if(Pulse_Width>=7&&Pulse_Width<=10)//560us
      {
        return;
      }
    }
  }
}
void setup()
{
  Serial.begin(9600);
  pinMode(IR_IN,INPUT);//设置红外接收引脚为输入
Serial.flush();
}
void loop()
{
  timer1_init();//定时器初始化
  while(1)
  {
    remote_decode();   //译码
    remote_deal();     //执行译码结果
  }
}
```

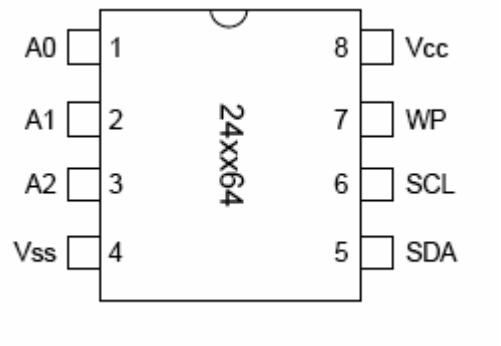
## 五、程序功能

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。按下相应的键，串口工具的监控窗口将显示地址码（即相应的遥控器的用户码） 和 键码命令值 。

# 第二十三章　EEPROM储存实验

1)概述：

储存芯片脚位图如下。




2）连线如下：

Arduino 的模拟口 4 连接到 EEPROM 的 5 脚；Arduino 的模拟口 5 连接到 EEPROM 的 6 脚；Arduino 的电源+5V 连接到 EEPROM 的 8 脚；Arduino 的 GND 连接到 EEPROM 的 1、2、3、4 脚。

3)　实验代码：

```
/*
   *    Use the I2C bus with EEPROM 24LC64
   *    Sketch:       eeprom.pde
   *
   *    Author: hkhijhe
   *    Date: 01/10/2010
   *
   *
***/

#include <Wire.h> //I2C library
void i2c_eeprom_write_byte( int deviceaddress, unsigned int eeaddress, byte data ) {
      int rdata = data;
      Wire.beginTransmission(deviceaddress);
      Wire.send((int)(eeaddress >> 8)); // MSB
      Wire.send((int)(eeaddress & 0xFF)); // LSB
      Wire.send(rdata);
      Wire.endTransmission();
}
   // WARNING: address is a page address, 6-bit end will wrap around
   // also, data can be maximum of about 30 bytes, because the Wire library has a buffer of 32 bytes
void i2c_eeprom_write_page( int deviceaddress, unsigned int eeaddresspage, byte* data, byte
```

```
length ) {
      Wire.beginTransmission(deviceaddress);
      Wire.send((int)(eeaddresspage >> 8)); // MSB
      Wire.send((int)(eeaddresspage & 0xFF)); // LSB
      byte c;
      for ( c = 0; c < length; c++)
         Wire.send(data[c]);
      Wire.endTransmission();
}
byte i2c_eeprom_read_byte( int deviceaddress, unsigned int eeaddress ) {
      byte rdata = 0xFF;
      Wire.beginTransmission(deviceaddress);
      Wire.send((int)(eeaddress >> 8)); // MSB
      Wire.send((int)(eeaddress & 0xFF)); // LSB
      Wire.endTransmission();
      Wire.requestFrom(deviceaddress,1);
      if (Wire.available()) rdata = Wire.receive();
      return rdata;
}
   // maybe let's not read more than 30 or 32 bytes at a time!
void i2c_eeprom_read_buffer( int deviceaddress, unsigned int eeaddress, byte *buffer, int length )
{
      Wire.beginTransmission(deviceaddress);
      Wire.send((int)(eeaddress >> 8)); // MSB
      Wire.send((int)(eeaddress & 0xFF)); // LSB
      Wire.endTransmission();
      Wire.requestFrom(deviceaddress,length);
      int c = 0;
      for ( c = 0; c < length; c++ )
         if (Wire.available()) buffer[c] = Wire.receive();
}
void setup()
{
      char somedata[] = "this is data from the eeprom"; // data to write
      Wire.begin(); // initialise the connection
      Serial.begin(9600);
      i2c_eeprom_write_page(0x50, 0, (byte *)somedata, sizeof(somedata)); // write to EEPROM

      delay(10); //add a small delay

      Serial.println("Memory written");
}

void loop()
```

```
{
    int addr=0; //first address
    byte b = i2c_eeprom_read_byte(0x50, 0); // access the first address from the memory

    while (b!=0)
    {
      Serial.print((char)b); //print content to serial port
      addr++; //increase address
      b = i2c_eeprom_read_byte(0x50, addr); //access an address from the memory
    }
    Serial.println(" ");
    delay(2000);
}
```

# 第二十四章 RFID读卡器实验

1）概述

射频技术也简称 RFID,RFID 是英文 radio frequency identification”的缩写，叫做射频识别技术，简称射频技术。

RFID 工作原理

射频识别系统的基本模型如图所示。

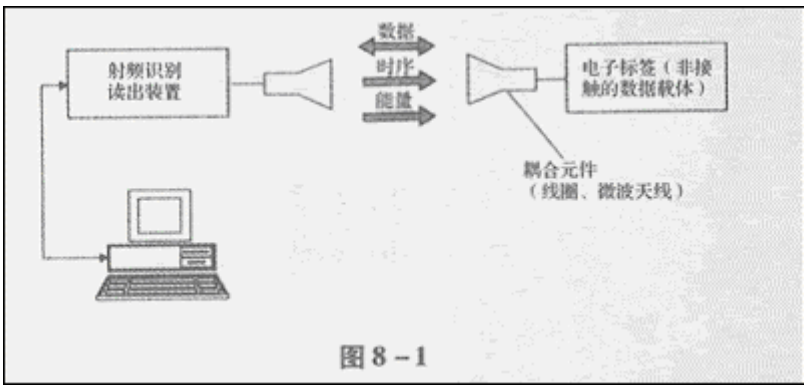
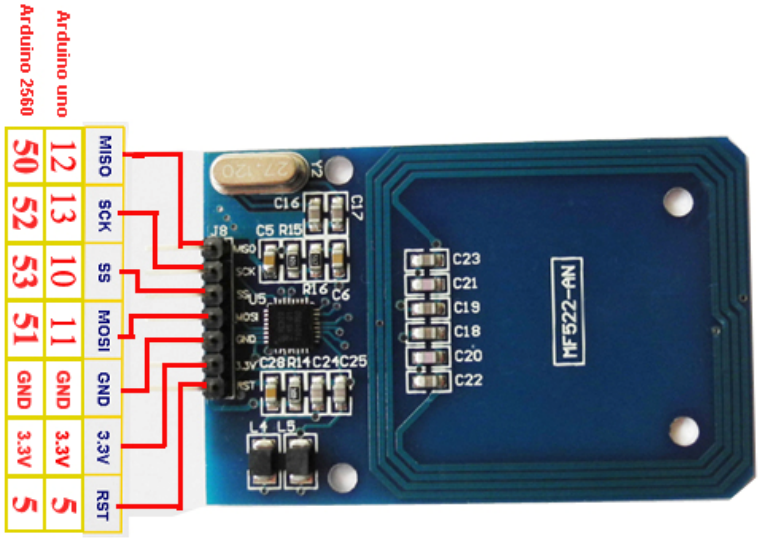


图 8－1

其中，电子标签又称为射频标签、应答器、数据载体；阅读器又称为读出装置，扫描器、通讯器、读写器(取决于电子标签是否可以无线改写数据)。电子标签与阅读器之间通过耦合元件实现射频信号的空间(无接触)耦合、在耦合通道内，根据时序关系，实现能量的传递、数据的交换。

**本模块，大家一定要使用+3.3V 供电，否则会烧掉模块 。**

2）具体的连线图如下。




4) 程序代码：

```
#include <SPI.h>

#define   uchar      unsigned char
#define   uint  unsigned int

//数组最大长度
#define MAX_LEN 16

//////////////////////////////////////////////////////////////////////
//set the pin
//////////////////////////////////////////////////////////////////////
const int chipSelectPin = 10; //如果控制板为 UNO,328,168
const int chipSelectPin = 53; //如果控制板为 mega 2560,1280
const int NRSTPD = 5;

//MF522 命令字
#define PCD_IDLE              0x00               //NO action;取消当前命令
#define PCD_AUTHENT           0x0E               //验证密钥
#define PCD_RECEIVE           0x08               //接收数据
#define PCD_TRANSMIT          0x04               //发送数据
#define PCD_TRANSCEIVE        0x0C               //发送并接收数据
#define PCD_RESETPHASE        0x0F               //复位
#define PCD_CALCCRC           0x03               //CRC 计算

//Mifare_One 卡片命令字
#define PICC_REQIDL           0x26               //寻天线区内未进入休眠状态
#define PICC_REQALL           0x52               //寻天线区内全部卡
#define PICC_ANTICOLL         0x93               //防冲撞
#define PICC_SElECTTAG        0x93               //选卡
#define PICC_AUTHENT1A        0x60               //验证 A 密钥
#define PICC_AUTHENT1B        0x61               //验证 B 密钥
#define PICC_READ             0x30               //读块
#define PICC_WRITE            0xA0               //写块
#define PICC_DECREMENT        0xC0               //
#define PICC_INCREMENT        0xC1               //
#define PICC_RESTORE          0xC2               //调块数据到缓冲区
#define PICC_TRANSFER         0xB0               //保存缓冲区中数据
#define PICC_HALT             0x50               //休眠


//和 MF522 通讯时返回的错误代码
#define MI_OK                 0
#define MI_NOTAGERR           1
#define MI_ERR                2
```

```
//------------------MFRC522 寄存器---------------
//Page 0:Command and Status
#define     Reserved00            0x00
#define     CommandReg            0x01
#define     CommIEnReg            0x02
#define     DivlEnReg             0x03
#define     CommIrqReg            0x04
#define     DivIrqReg             0x05
#define     ErrorReg              0x06
#define     Status1Reg            0x07
#define     Status2Reg            0x08
#define     FIFODataReg           0x09
#define     FIFOLevelReg          0x0A
#define     WaterLevelReg         0x0B
#define     ControlReg            0x0C
#define     BitFramingReg         0x0D
#define     CollReg               0x0E
#define     Reserved01            0x0F
//Page 1:Command
#define     Reserved10            0x10
#define     ModeReg               0x11
#define     TxModeReg             0x12
#define     RxModeReg             0x13
#define     TxControlReg          0x14
#define     TxAutoReg             0x15
#define     TxSelReg              0x16
#define     RxSelReg              0x17
#define     RxThresholdReg        0x18
#define     DemodReg              0x19
#define     Reserved11            0x1A
#define     Reserved12            0x1B
#define     MifareReg             0x1C
#define     Reserved13            0x1D
#define     Reserved14            0x1E
#define     SerialSpeedReg        0x1F
//Page 2:CFG
#define     Reserved20            0x20
#define     CRCResultRegM         0x21
#define     CRCResultRegL         0x22
#define     Reserved21            0x23
#define     ModWidthReg           0x24
#define     Reserved22            0x25
```

```
#define     RFCfgReg            0x26
#define     GsNReg              0x27
#define     CWGsPReg            0x28
#define     ModGsPReg           0x29
#define     TModeReg            0x2A
#define     TPrescalerReg       0x2B
#define     TReloadRegH         0x2C
#define     TReloadRegL         0x2D
#define     TCounterValueRegH   0x2E
#define     TCounterValueRegL   0x2F
//Page 3:TestRegister
#define     Reserved30          0x30
#define     TestSel1Reg         0x31
#define     TestSel2Reg         0x32
#define     TestPinEnReg        0x33
#define     TestPinValueReg     0x34
#define     TestBusReg          0x35
#define     AutoTestReg         0x36
#define     VersionReg          0x37
#define     AnalogTestReg       0x38
#define     TestDAC1Reg         0x39
#define     TestDAC2Reg         0x3A
#define     TestADCReg          0x3B
#define     Reserved31          0x3C
#define     Reserved32          0x3D
#define     Reserved33          0x3E
#define     Reserved34          0x3F
//-----------------------------------------------

//4 字节卡序列号，第 5 字节为校验字节
uchar serNum[5];
uchar    writeDate[16] ={'T', 'e', 'n', 'g', ' ', 'B', 'o', 0, 0, 0, 0, 0, 0, 0, 0,0};
//扇区 A 密码，16 个扇区，每个扇区密码 6Byte
 uchar sectorKeyA[16][16] = {{0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF},
                             {0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF},
                             {0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF},
                            };
 uchar sectorNewKeyA[16][16] = {{0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF},
                                {0xFF,  0xFF,  0xFF,  0xFF,  0xFF,  0xFF,
0xff,0x07,0x80,0x69, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF},
                                {0xFF,  0xFF,  0xFF,  0xFF,  0xFF,  0xFF,
0xff,0x07,0x80,0x69, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF},
                               };
```

```
void setup() {
    Serial.begin(9600);                                  // RFID reader SOUT pin connected to Serial
RX pin at 2400bps
 // start the SPI library:
   SPI.begin();

   pinMode(chipSelectPin,OUTPUT);                        // Set digital pin 10 as OUTPUT to connect
it to the RFID /ENABLE pin
      digitalWrite(chipSelectPin, LOW);                  // Activate the RFID reader
   pinMode(NRSTPD,OUTPUT);                               // Set digital pin 10 , Not Reset and
Power-down
      digitalWrite(NRSTPD, HIGH);

   MFRC522_Init();
}

void loop()
{
      uchar i,tmp;
      uchar status;
            uchar str[MAX_LEN];
            uchar RC_size;
            uchar blockAddr;   //选择操作的块地址 0～63


            //寻卡，返回卡类型
            status = MFRC522_Request(PICC_REQIDL, str);
            if (status == MI_OK)
            {
            }

            //防冲撞，返回卡的序列号 4 字节
            status = MFRC522_Anticoll(str);
            memcpy(serNum, str, 5);
            if (status == MI_OK)
            {
                              Serial.println("The card's number is    : ");
                  Serial.print(serNum[0],BIN);
                  Serial.print(serNum[1],BIN);
                  Serial.print(serNum[2],BIN);
                  Serial.print(serNum[3],BIN);
                  Serial.print(serNum[4],BIN);
                              Serial.println(" ");
            }
```

```
        //选卡，返回卡容量
        RC_size = MFRC522_SelectTag(serNum);
        if (RC_size != 0)
        {}

        //写数据卡
        blockAddr = 7;            //数据块 7
        status = MFRC522_Auth(PICC_AUTHENT1A, blockAddr, sectorKeyA[blockAddr/4],
serNum); //认证
        if (status == MI_OK)
        {
            //写数据
            status = MFRC522_Write(blockAddr, sectorNewKeyA[blockAddr/4]);
                        Serial.print("set the new card password, and can modify the data of
the Sector: ");
                        Serial.print(blockAddr/4,DEC);

                        //写数据
                        blockAddr = blockAddr - 3 ;
                        status = MFRC522_Write(blockAddr, writeDate);
                        if(status == MI_OK)
                        {
                            Serial.println("OK!");
                        }
        }

        //读卡
        blockAddr = 7;            //数据块 7
        status          =          MFRC522_Auth(PICC_AUTHENT1A,          blockAddr,
sectorNewKeyA[blockAddr/4], serNum);    //认证
        if (status == MI_OK)
        {
            //读数据
                        blockAddr = blockAddr - 3 ;
                        status = MFRC522_Read(blockAddr, str);
            if (status == MI_OK)
            {
                                Serial.println("Read from the card ,the data is : ");
                for (i=0; i<16; i++)
                {
                                Serial.print(str[i]);
                }
                                Serial.println(" ");
```

```
                }
            }
                    Serial.println(" ");
            MFRC522_Halt();                 //命令卡片进入休眠状态

}

/*
 * 函 数 名：Write_MFRC5200
 * 功能描述：向 MFRC522 的某一寄存器写一个字节数据
 * 输入参数：addr--寄存器地址；val--要写入的值
 * 返 回 值：无
 */

……
```

因数据边幅很长，所以代码，在这里不全部显示了。

本实验，当 IC 卡靠近后，RFID 模块将写入数据到 IC 卡，然后 RFID 模块再从 IC 卡读出数据，并显示在监控窗口中。

# 第二十五章　门禁系统实验

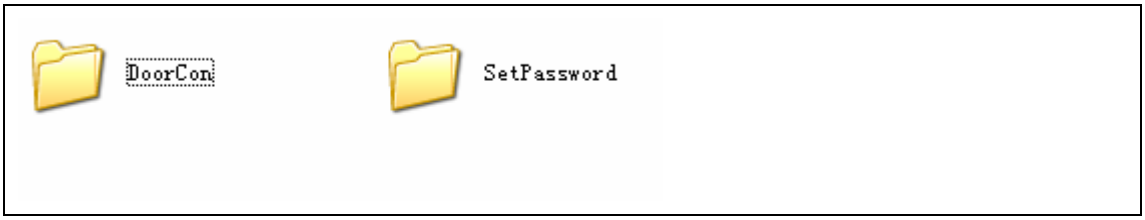
1）概述

本章实验中， 我们将使用 RFID 模块，和继电器，以及红、绿 LED 小灯 开发一个门禁系统。

当我们拿着预先设置了密码的 IC 卡，向 RFID 模块刷卡时，如果密码不正确，则继电器常开，红灯亮，表示门不开 ； 如果密码正确，则继电器闭合，绿灯亮起来，表示开门 。实际应用中，只需将接两小灯的电路稍作修改，连到门的控制器上就可以了。

连线图，请大家结合上一章“RFID 读卡器实验” 和 17 章的“继电器控制实验”连线图。

2） 程序代码：




请参考“门禁系统实验” 程序文件夹下。

使用方法： 1） 先下载“SetPassword.pde ”代码到控制板， 连接好 RFID 模块之后，打开串口的监控窗口，将 IC 卡靠近模块后， 当显示“The password has been changed!”（ 密码已经修改完毕）。

2） 再下载“DoorCon.pde”代码到控制板，同时把 继电器控制实验连接线路也接上来。则当 IC 卡靠近模块后，RFID 模块检测该卡，如果密码不一致，则继电器常开，红灯亮，门紧闭 ； 如果密码一致，则继电器闭合，绿灯亮，门被打开。

通过这个实验后，大家也可以发挥创意，将键盘操作也融入进去，实现更多丰富的项目。接下来，就靠大家自由发挥啦！