

Arduino

从基础到实践

【美】Michael McRoberts 著 杨继志 郭敬 译

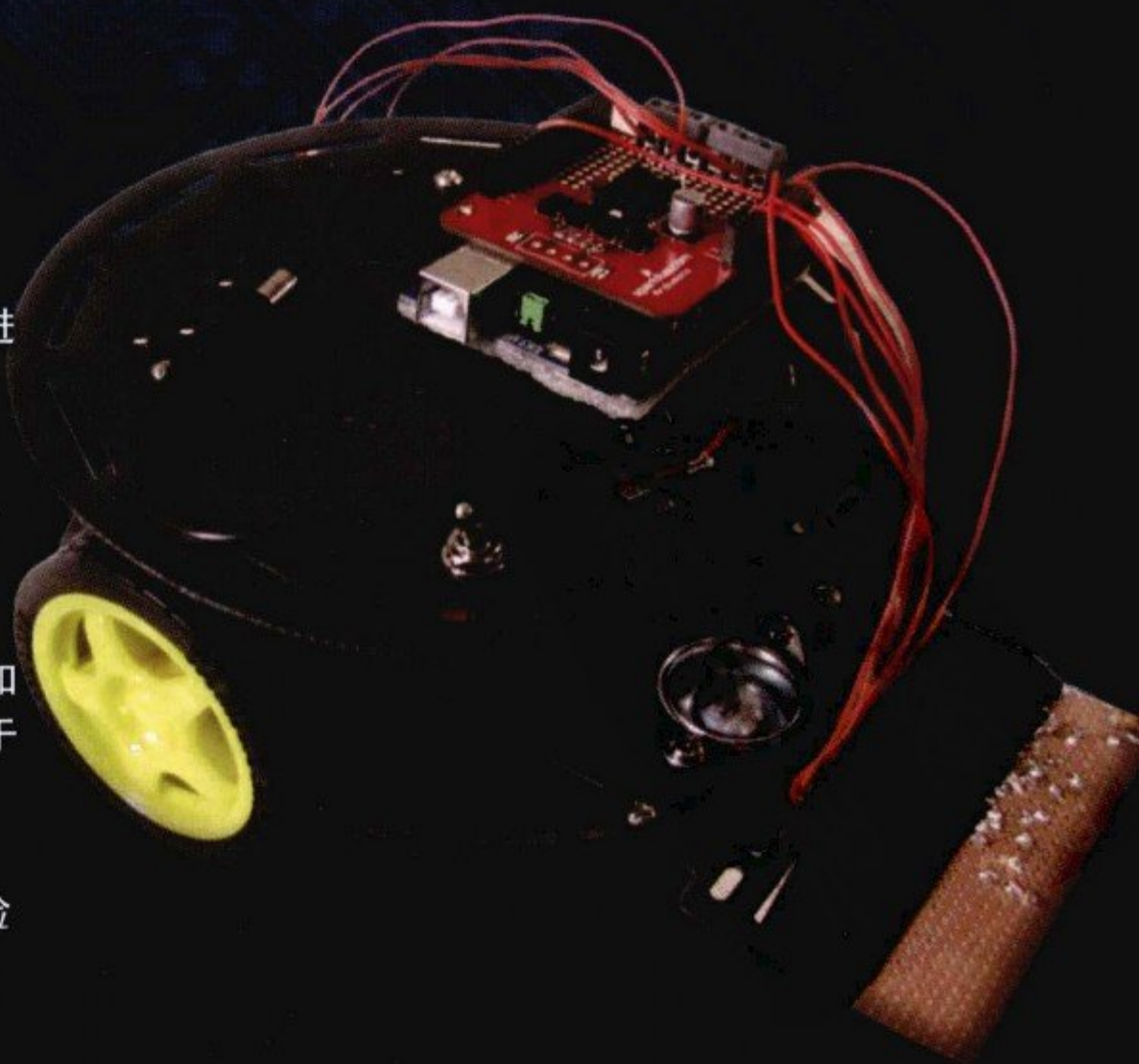
Beginning Arduino

深入浅出、图文并茂、循序渐进地讲授**50个**Arduino项目

学习如何使用**小电机、传感器、显示器和网络**

完成简单但实用的项目，如**测距仪、RFID读卡器、基于互联网的天气显示器**

无须具备编程或电子操作经验



Arduino 从基础到实践

本书通过50个非常酷的项目讲授当前流行的Arduino知识，帮你从一个初学者，转变成能够建立自己的Arduino项目的能手，而且之前绝对不需要你具备任何编程或电子知识。

本书采用手把手教学方法，从开始你就会沉浸在项目制作中，学习如何使用各种电子元件，学习如何对Arduino编程以实现控制和沟通各种元件。每一个项目都建立在之前项目的知识基础上，你可以稳步提升编程知识和电子技术技能。

本书采用逐步详细讲解的方法，列出了全部源代码，你可以学到下面这些非常有趣的东西：

- ◆ 控制LED
- ◆ 在LCD上显示字符和图形
- ◆ 制作巡线机器人
- ◆ 使用触摸屏
- ◆ 使用数字压力传感器
- ◆ 从SD卡中读写数据
- ◆ 连接Arduino到Internet

学习完这本书你就可以非常自信并有创造力地制作属于自己的项目了，一起开动吧！

Apress
www.apress.com



策划编辑：林瑞和
责任编辑：董英
封面设计：李玲

上架建议：单片计算机

ISBN 978-7-121-19201-2



9 787121 192012 >

定价：79.00元

Arduino

从基础到实践

【美】Michael McRoberts 著 杨继志 郭敬 译

Beginning Arduino

电子工业出版社
Publishing House of Electronics Industry
北京•BEIJING



内 容 简 介

采用 Arduino 进行电子制作越来越流行，在网络上可以找到很多用 Arduino 制作机器人、媒体互动产品、电子创意项目的案例。本书用 50 个项目来介绍 Arduino 的应用，从最基础的输入输出项目逐渐进入较高级的话题，比如 Arduino 与互联网的连接等。每一个项目都由完整的硬件方案和软件程序组成，读者无须再翻阅其他书籍即可完成本书中的各种电子制作项目。

本书对读者的基础知识要求非常低，非常适合学生进行课外电子制作项目使用，同时本书也介绍了一些相当有难度和实用性很强的项目，对于有一定基础的电子爱好者也有很好的参考价值。

Beginning Arduino

By Michael McRoberts, ISBN 978-1-4302-3240-7

Original English language edition published by Apress Media.

Copyright © 2011 by Apress Media

Simplified Chinese-language edition copyright © 2013 by Publishing House of Electronics Industry. All rights reserved.

本书简体中文版专有出版权由 Apress Media 授予电子工业出版社。专有出版权受法律保护。

版权贸易合同登记号 图字：01-2012-7689

图书在版编目（CIP）数据

Arduino 从基础到实践 / （美）麦克罗伯茨（McRoberts,M.）著；杨继志，郭敬译。

北京：电子工业出版社，2013.3

书名原文：Beginning Arduino

ISBN 978-7-121-19201-2

I. ①A… II. ①麦… ②杨… ③郭… III. ①单片微型计算机—程序设计 IV. ①TP368.1

中国版本图书馆 CIP 数据核字（2012）第 295357 号

策划编辑：林瑞和

责任编辑：董 英

印 刷：北京天宇星印刷厂

装 订：三河市鹏成印业有限公司

出版发行：电子工业出版社

北京市海淀区万寿路 173 信箱 邮编 100036

开 本：787×1092 1/16 印张：29.25 字数：468 千字 彩插：8

印 次：2013 年 3 月第 1 次印刷

印 数：4000 册 定价：79.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

服务热线：（010）88258888。

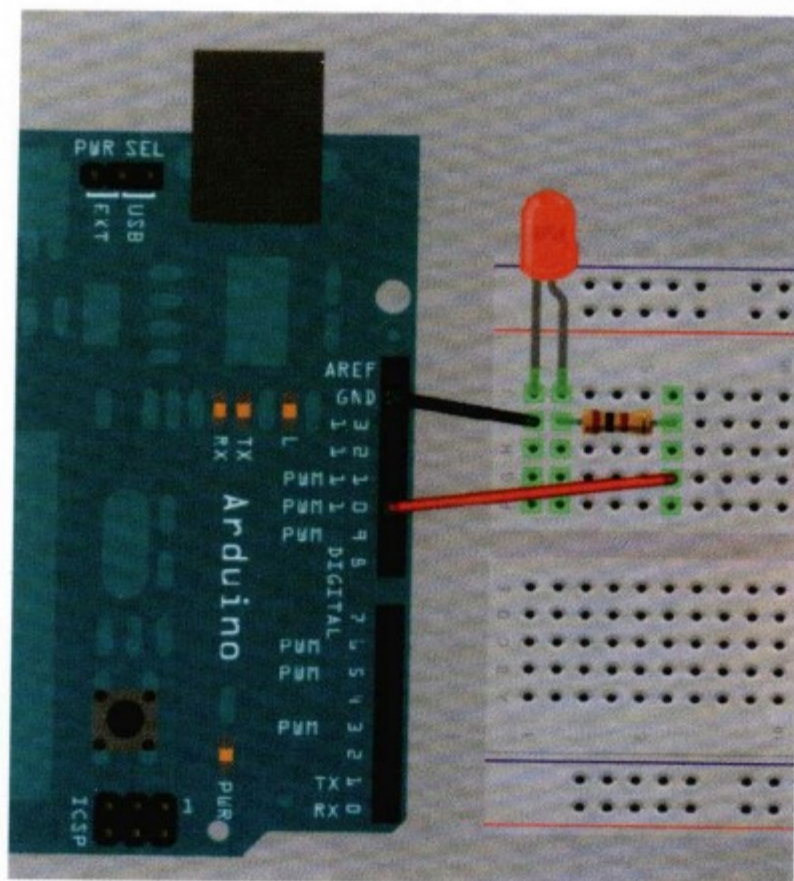


图 2-1 项目 1——LED 闪灯器电路图

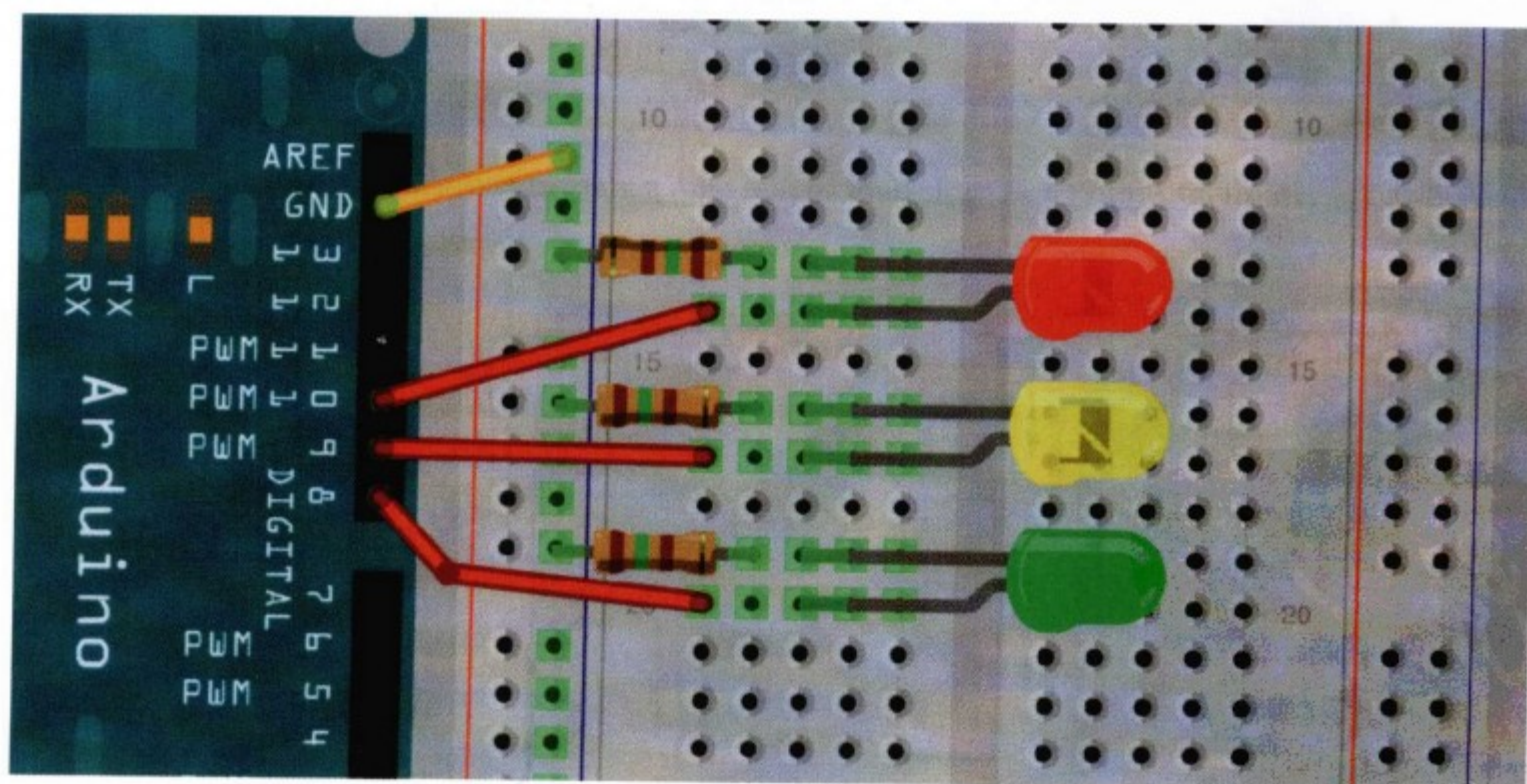


图 2-6 项目 3——交通信号灯电路图

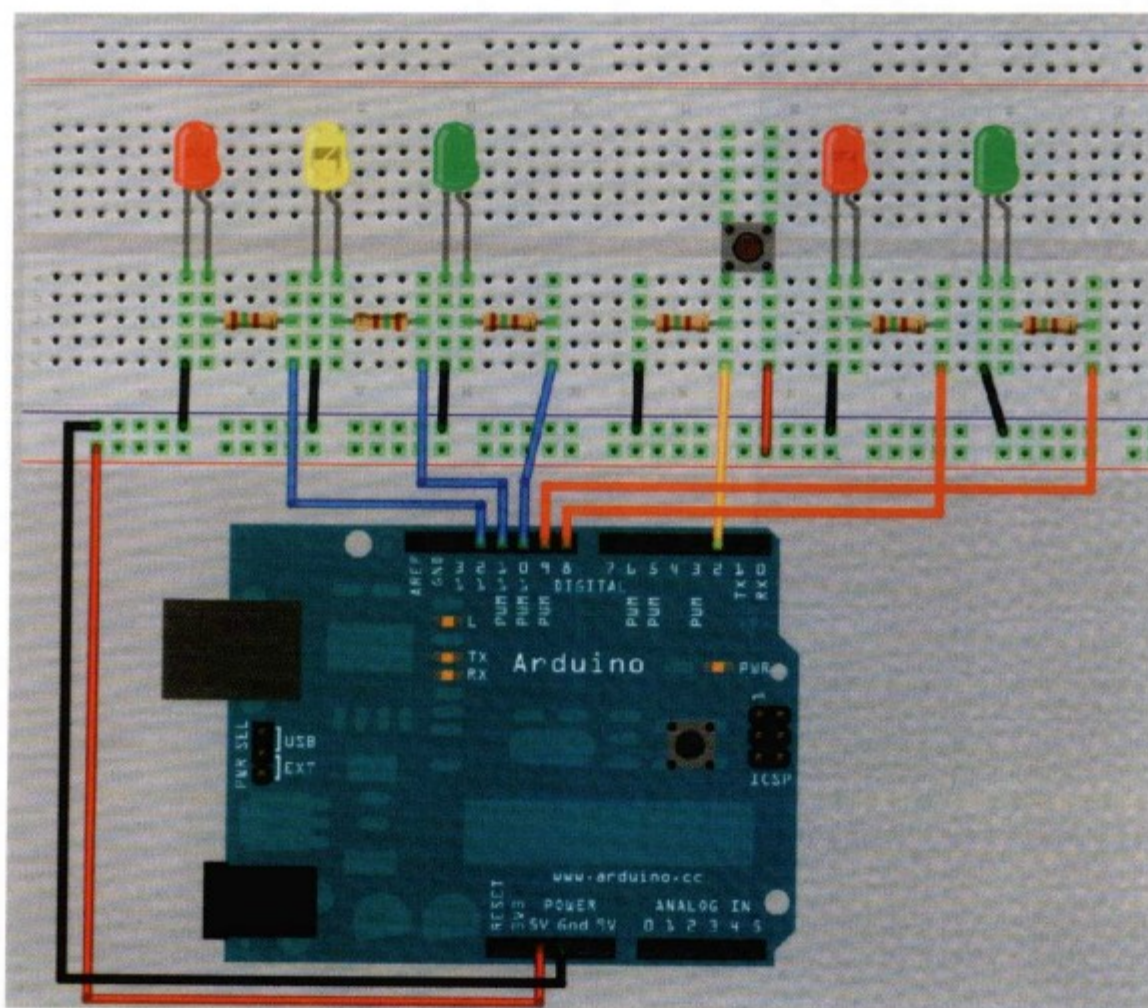


图 2-8 项目 4——有行人通过灯和请求按钮的交通信号灯系统电路图

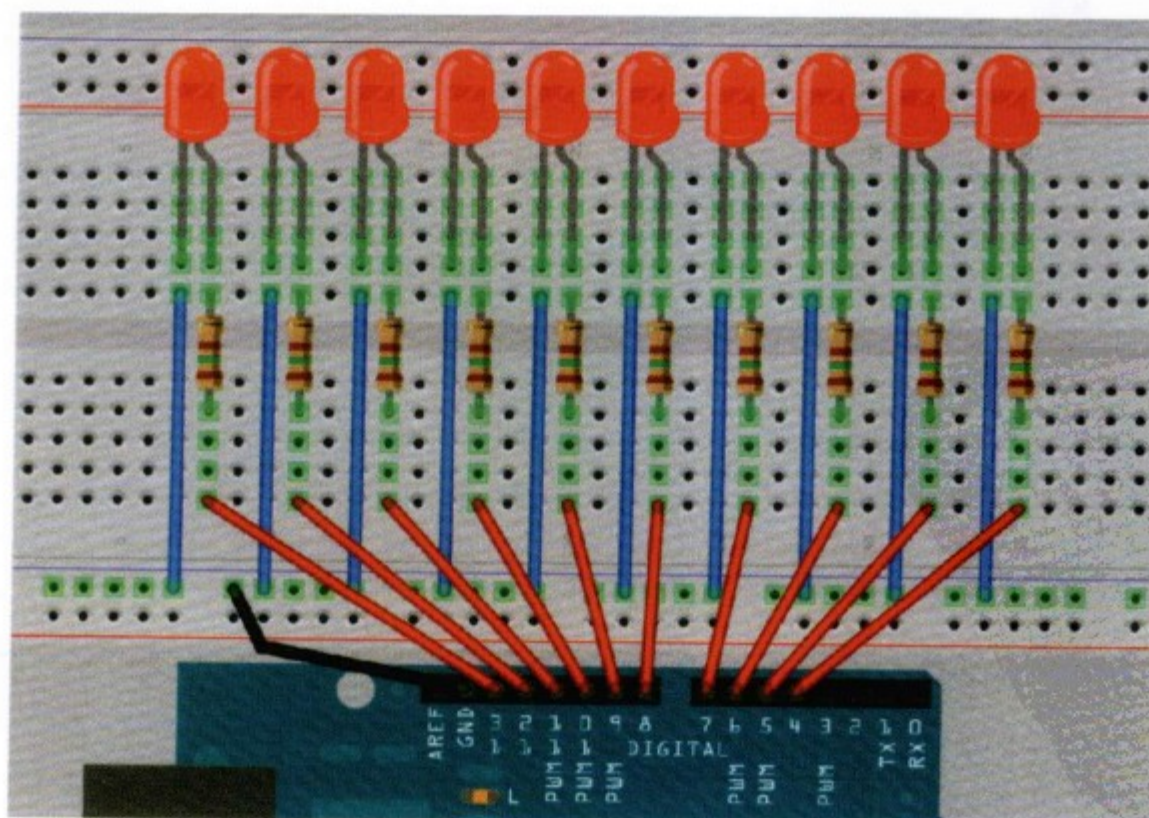


图 3-1 项目 5——LED 跑马灯效果电路图

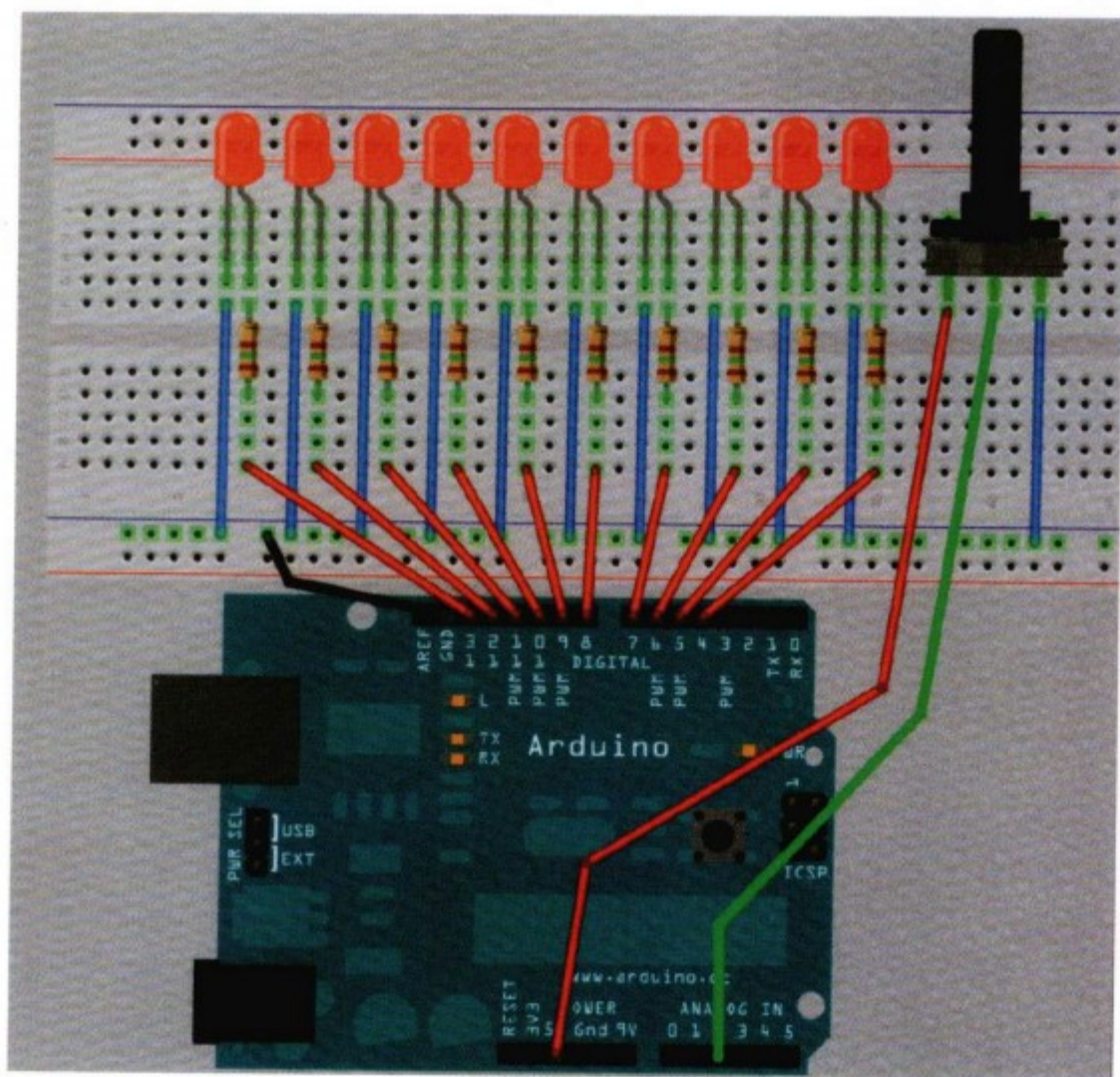


图 3-2 项目 6——互动 LED 跑马灯效果电路图

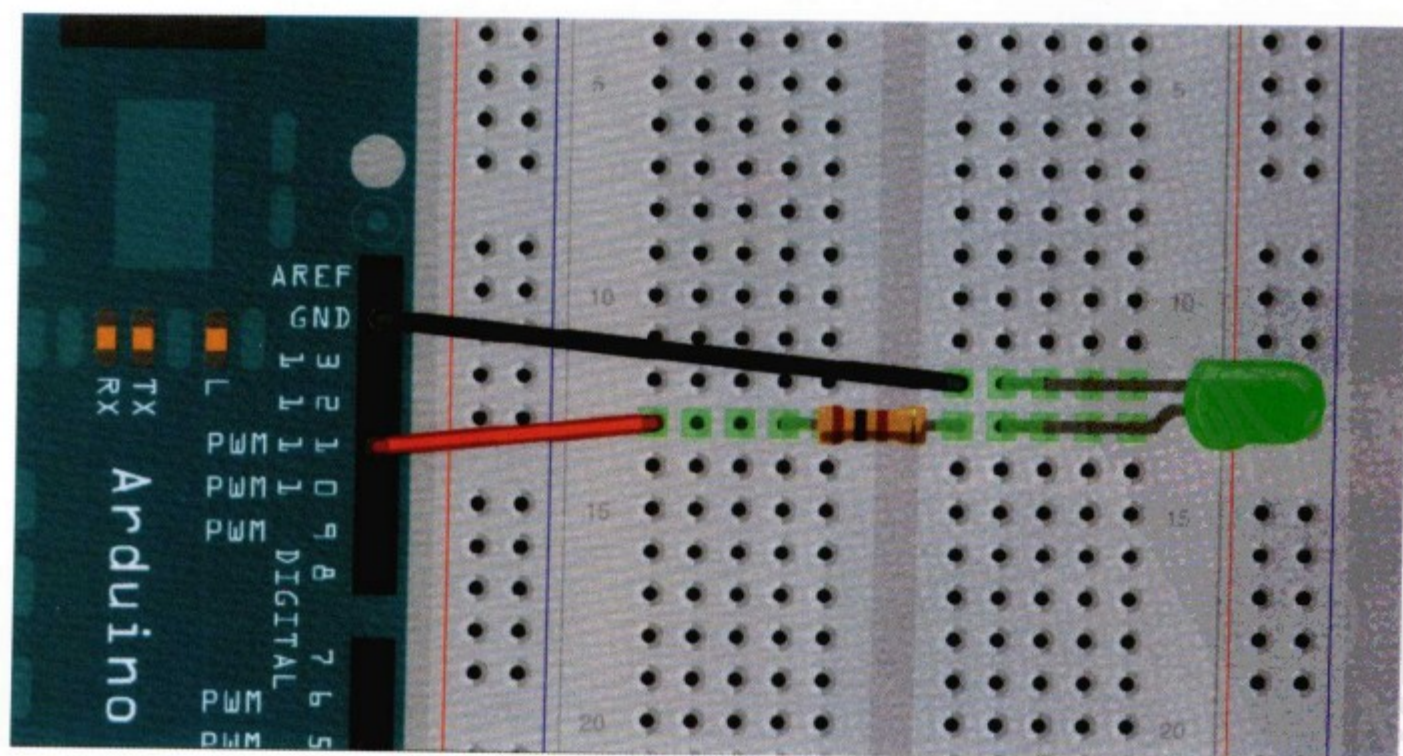


图 3-3 项目 7——闪烁灯电路图

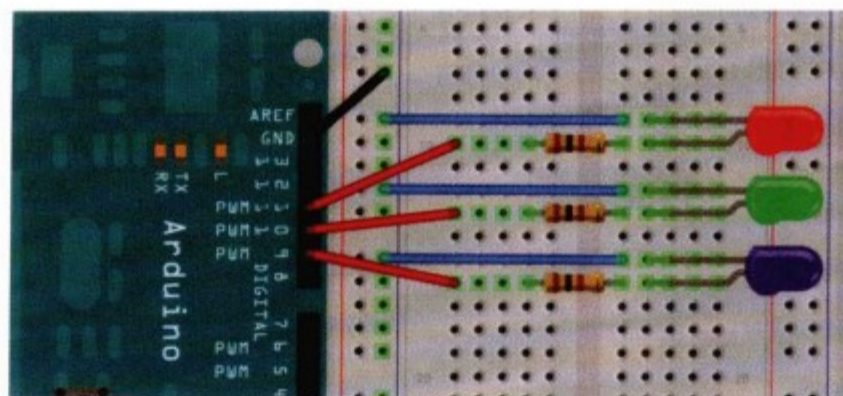


图 3-4 项目 8——RGB 彩灯电路图



图 3-5 混合 R、G、B 获得不同颜色

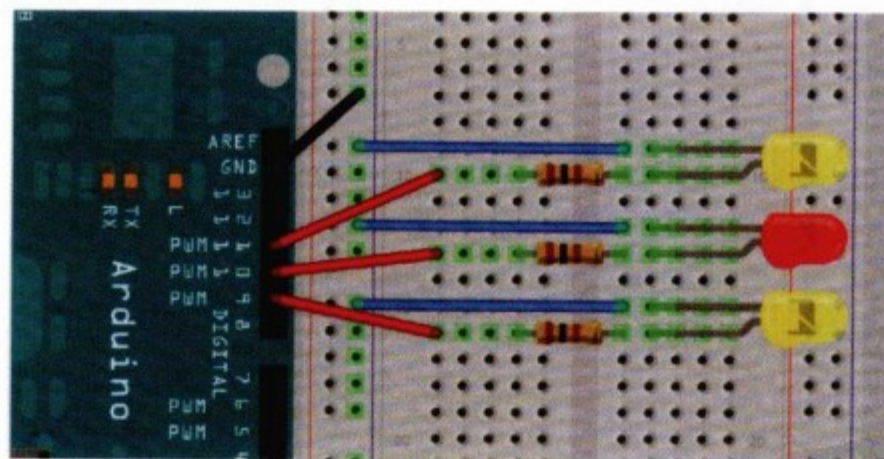


图 3-6 项目 9——LED 火焰效果电路图

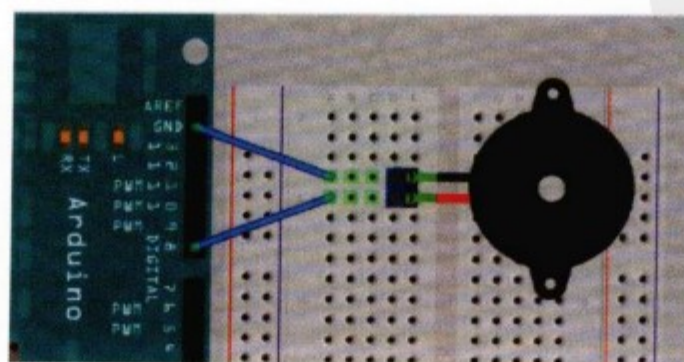


图 4-1 项目 11——压电声音报警器电路图

新华书店
PDG

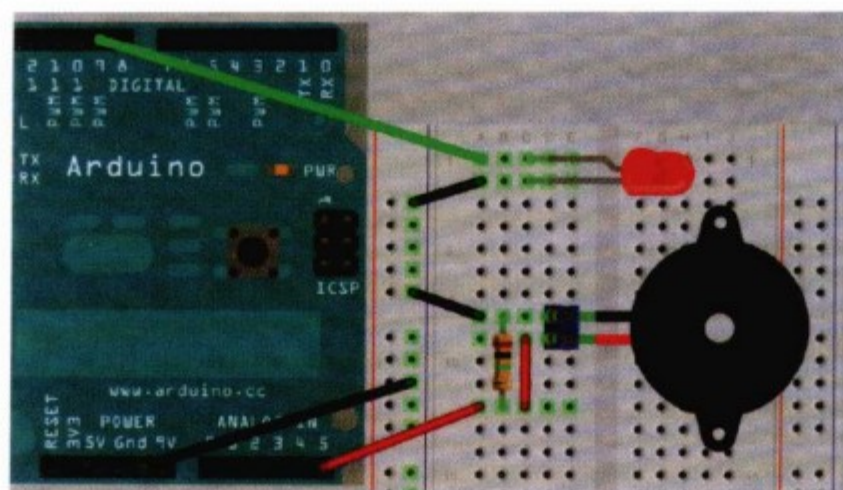


图 4-3 项目 13——压电震动传感器电路图

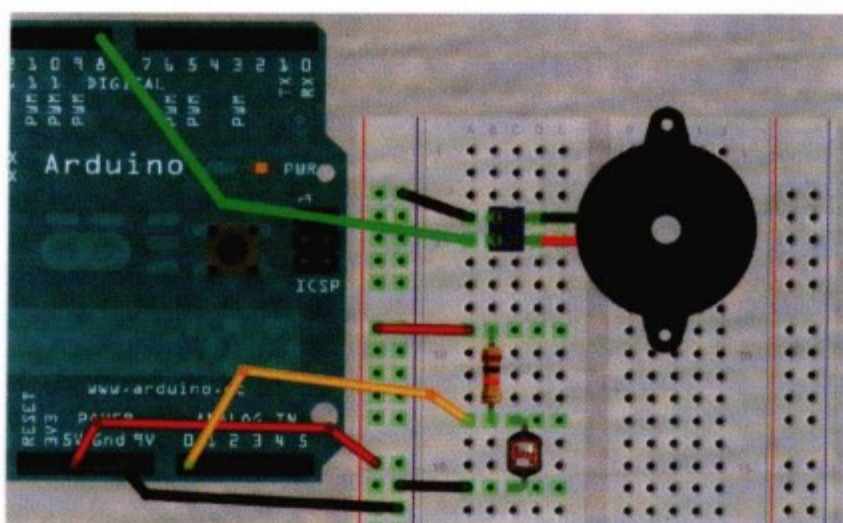


图 4-4 项目 14——光敏元件电路图

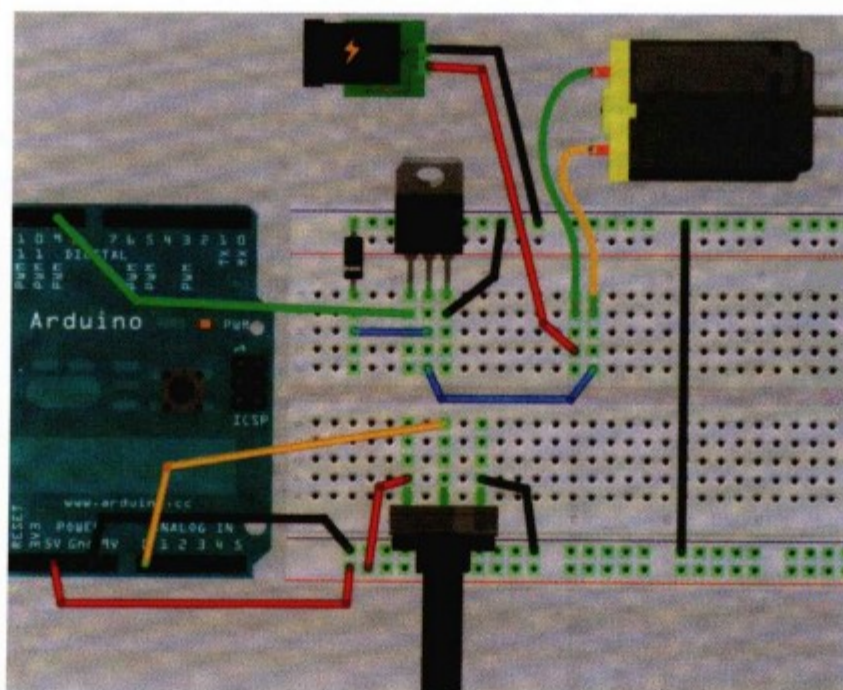


图 5-1 项目 15——简单的电机控制系统电路图

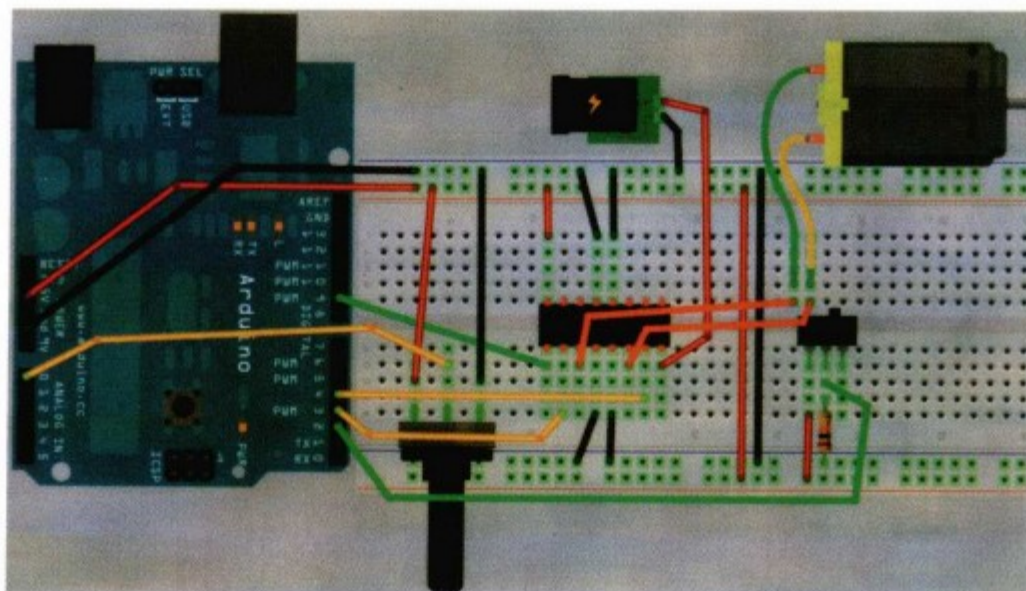


图 5-3 项目 16 电路图

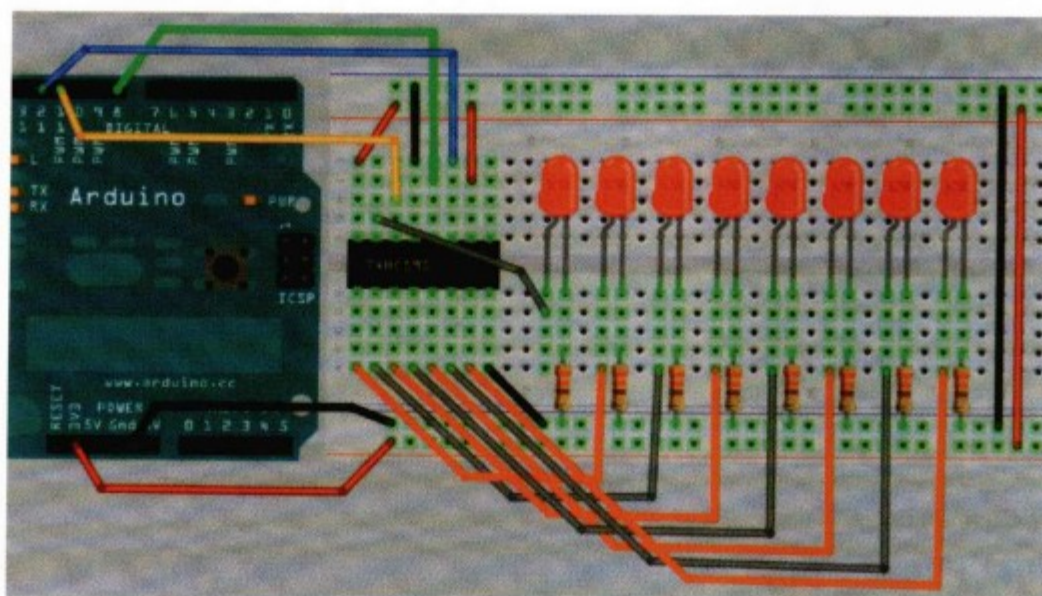


图 6-1 项目 17——移位寄存器 8 位计数器电路图

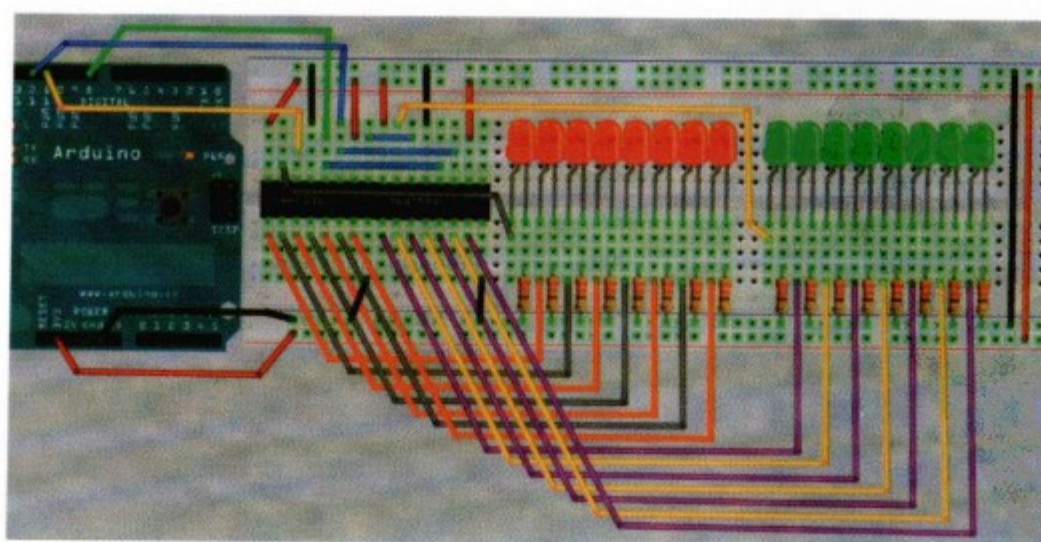


图 6-4 项目 18 电路图

知
道
PDG

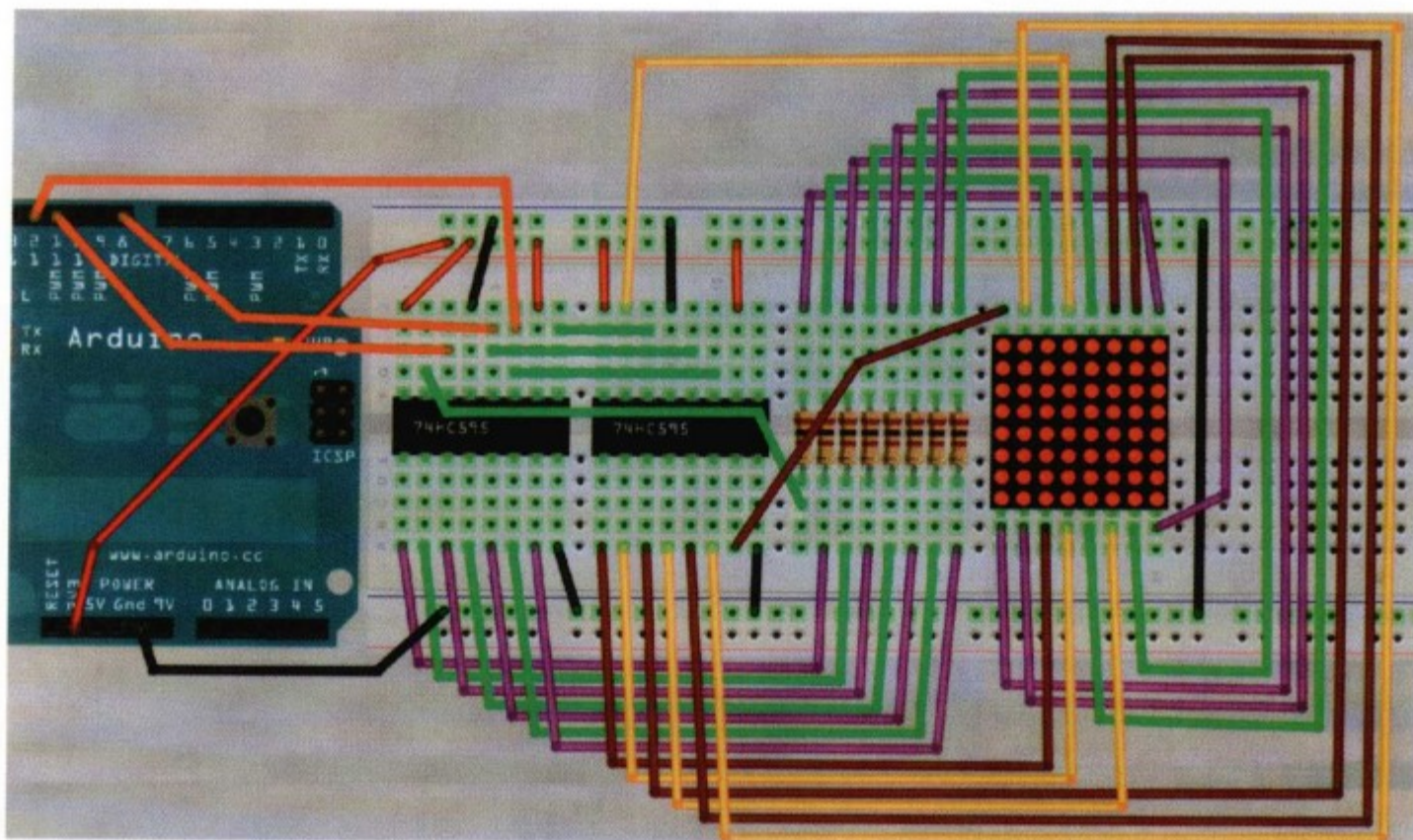


图 7-1 项目 19—LED 点阵显示器—基本动画电路图

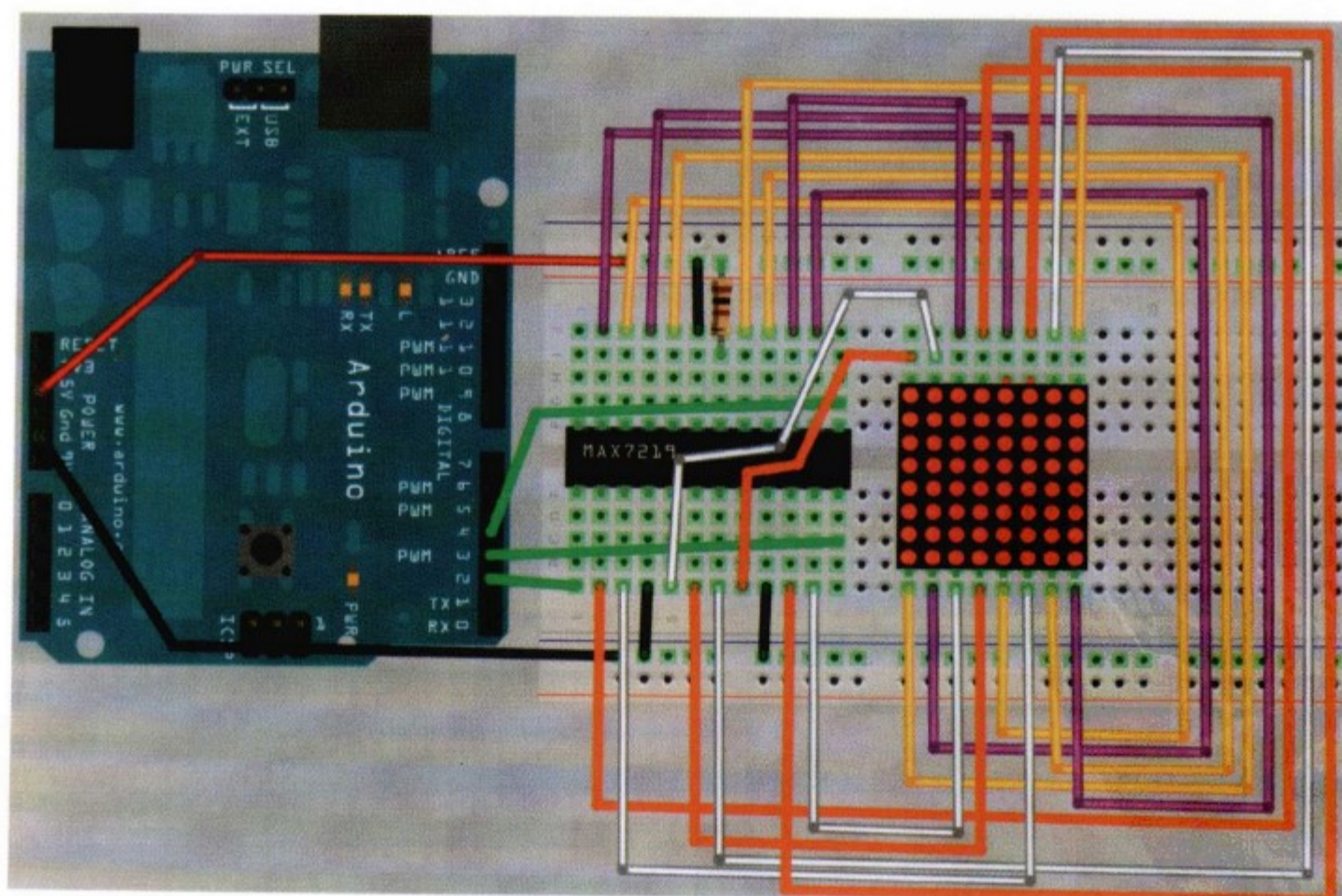


图 7-4 项目 21 电路图

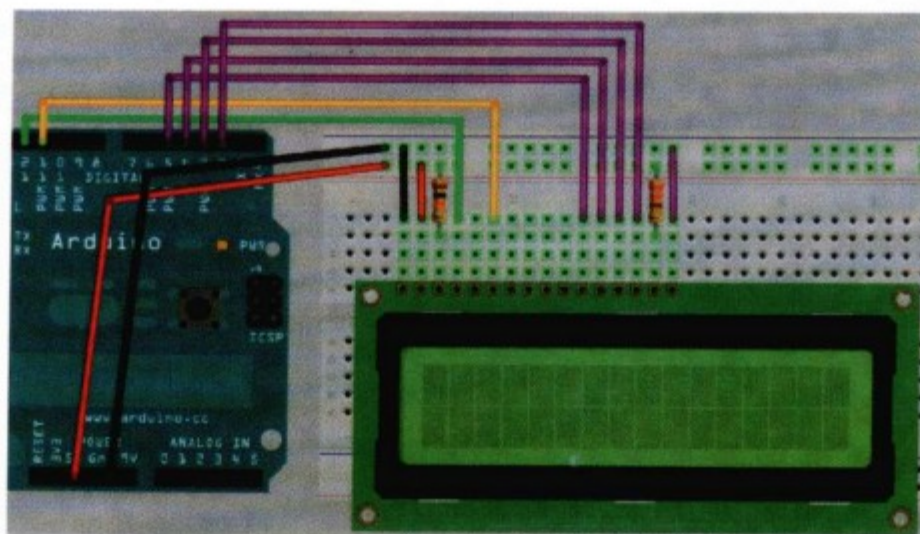


图 8-1 项目 23——基本 LCD 控制电路图

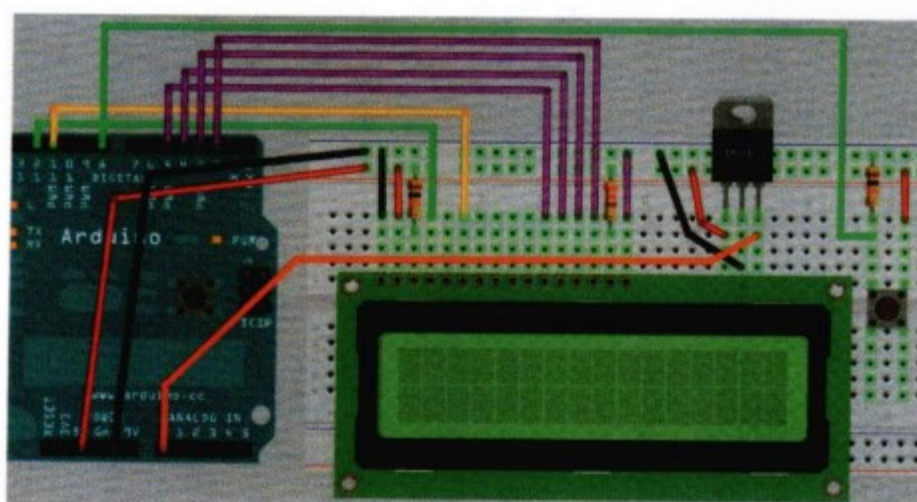


图 8-2 项目 24——LCD 温度显示器电路图

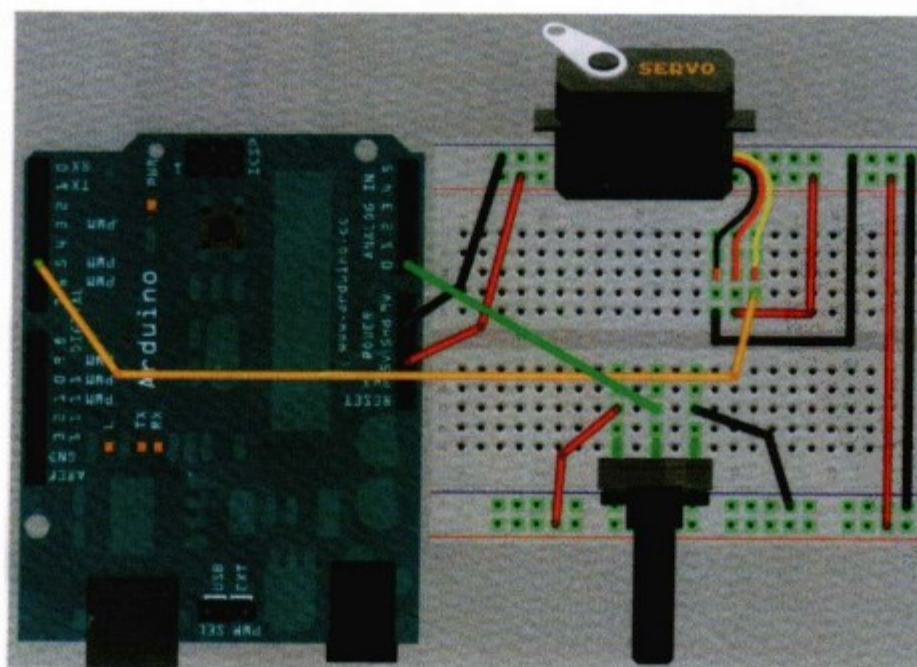


图 9-3 项目 25——舵机控制电路图

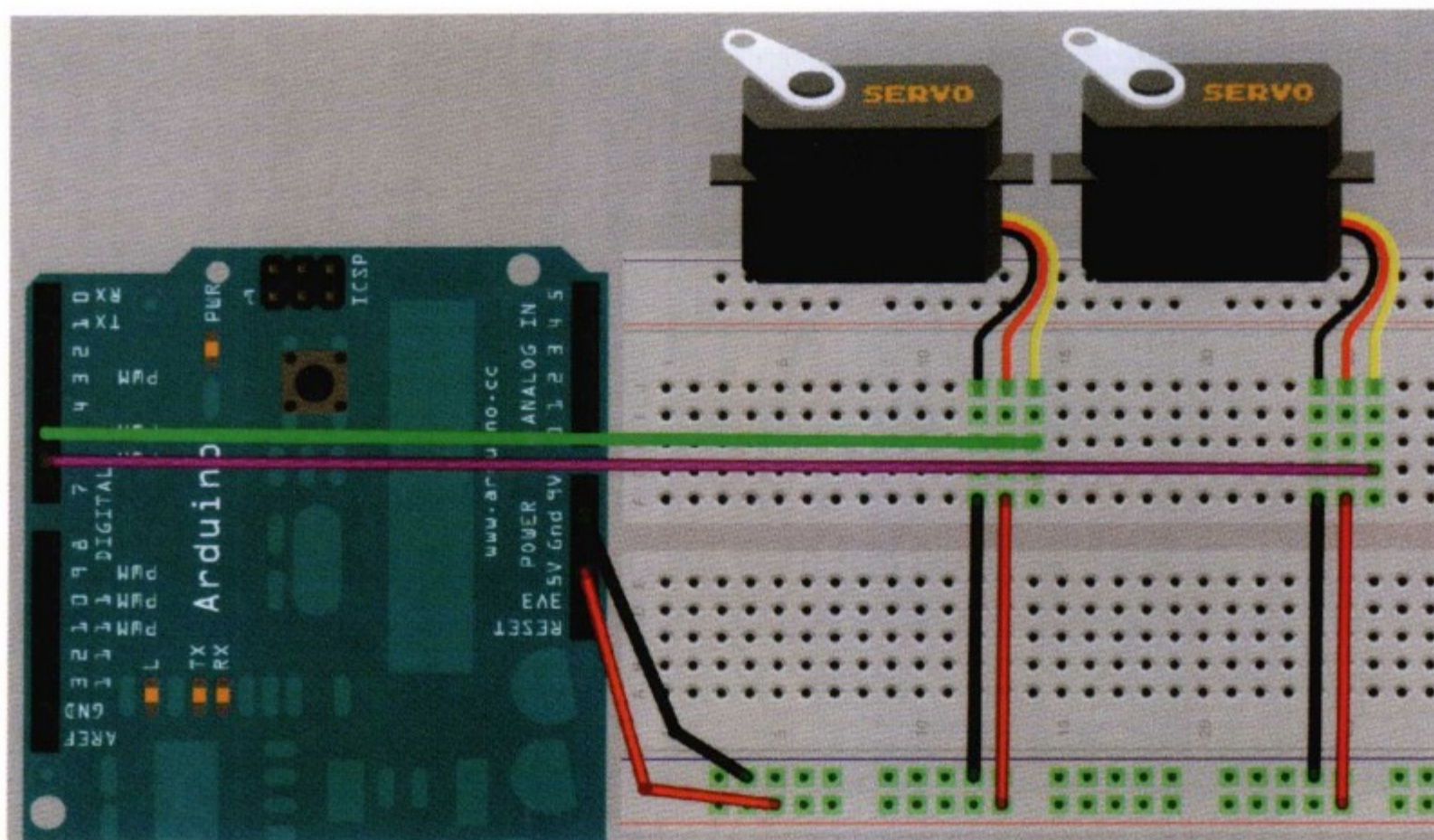


图 9-5 项目 26——两个舵机控制系统电路图

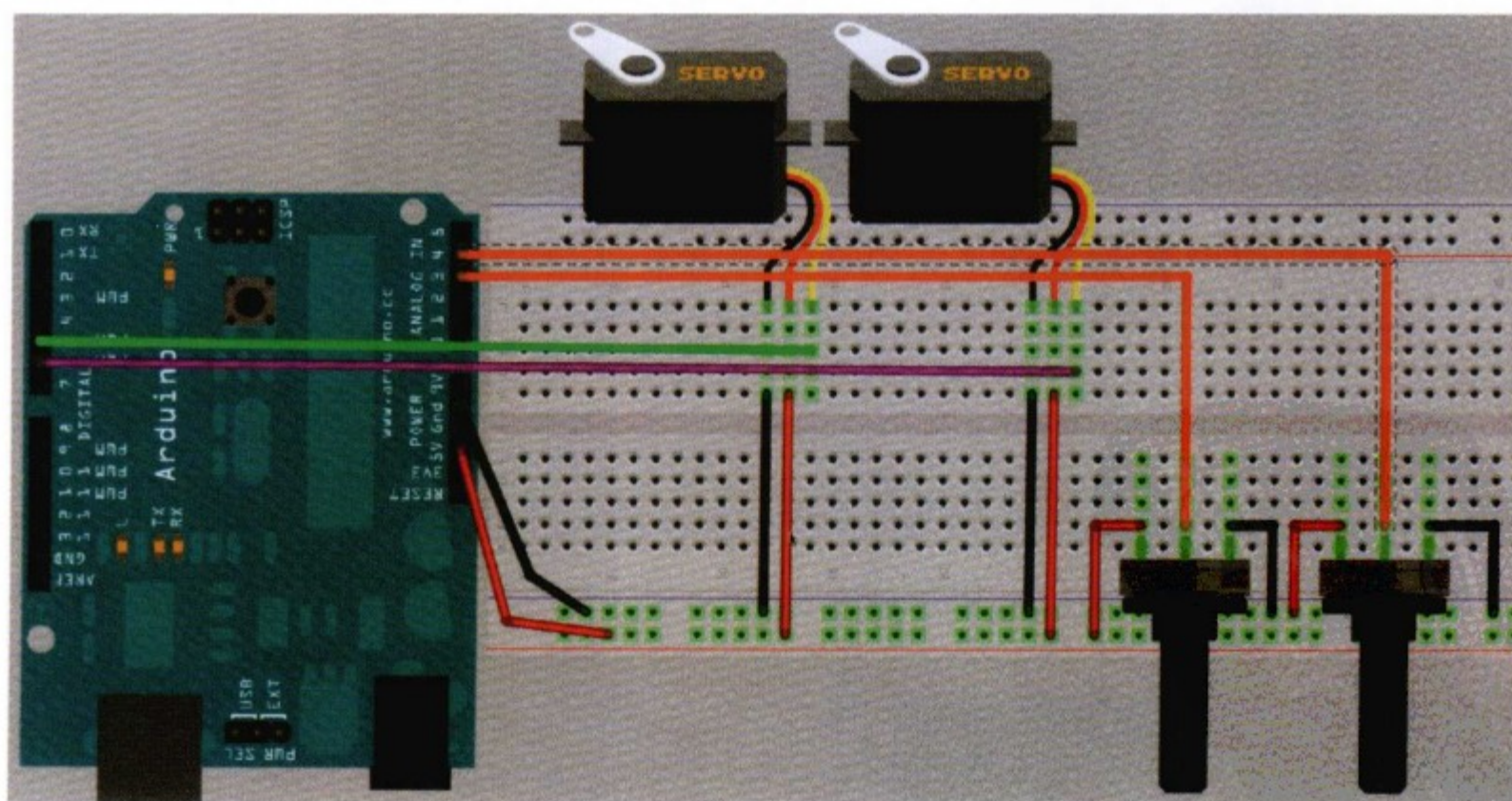


图 9-6 项目 27——操纵杆控制舵机电路图

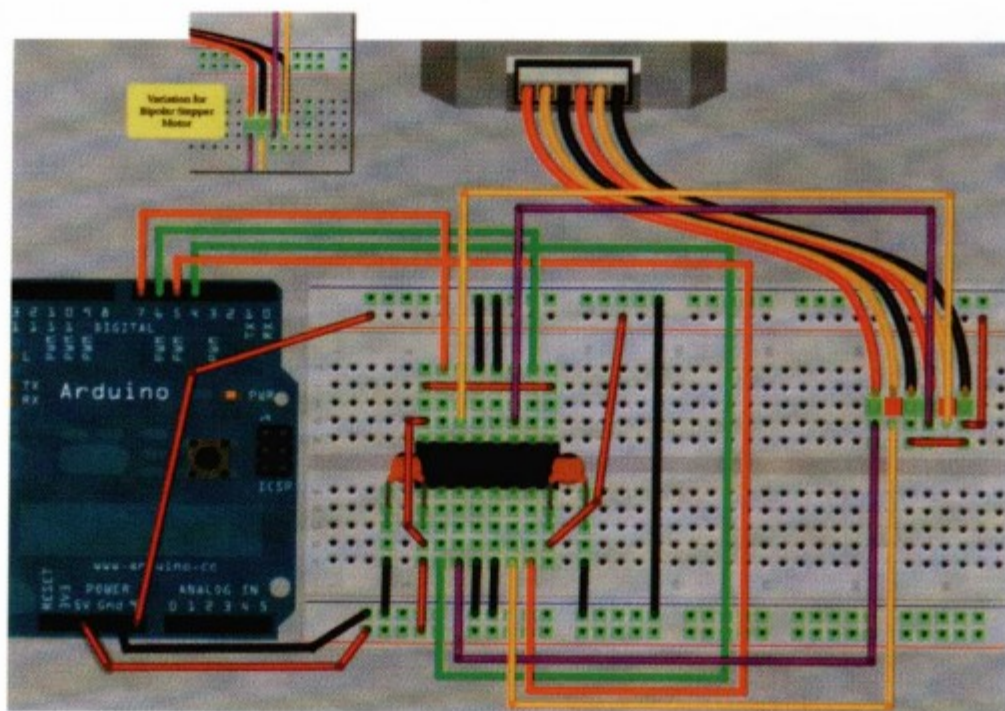


图 10-1 项目 28——基本步进电机控制电路图

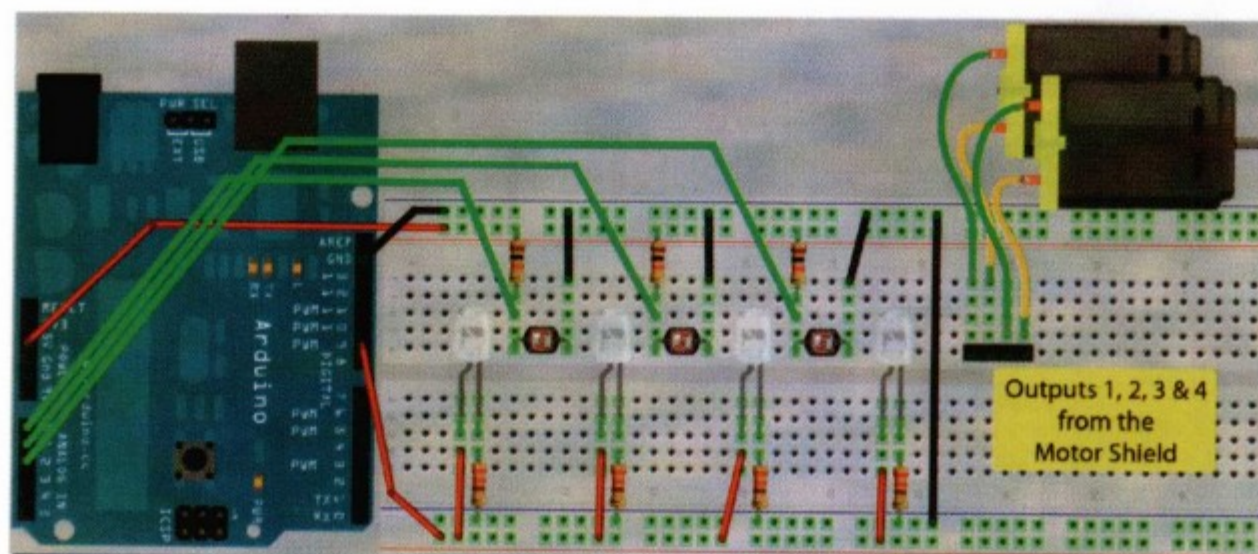


图 10-9 项目 30——巡线机器人电路图

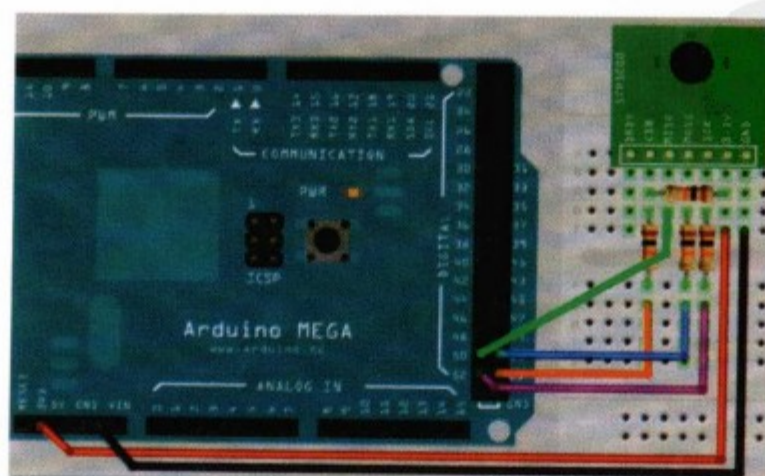


图 11-2 项目 31——数字压力传感器电路图

数字压力传感器
PDG

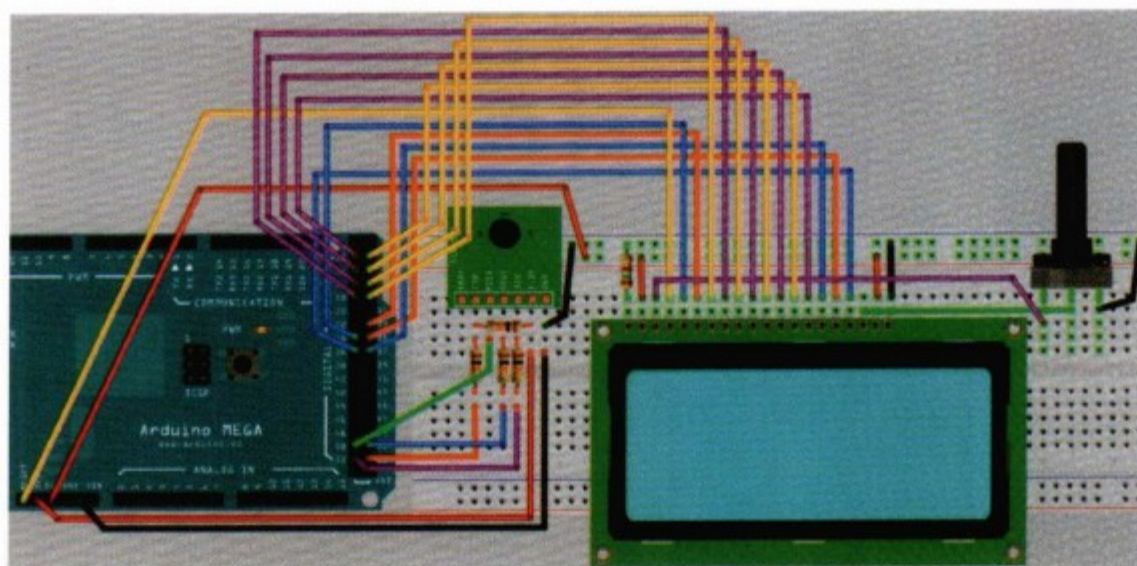


图 11-5 项目 32——数字气压表电路图

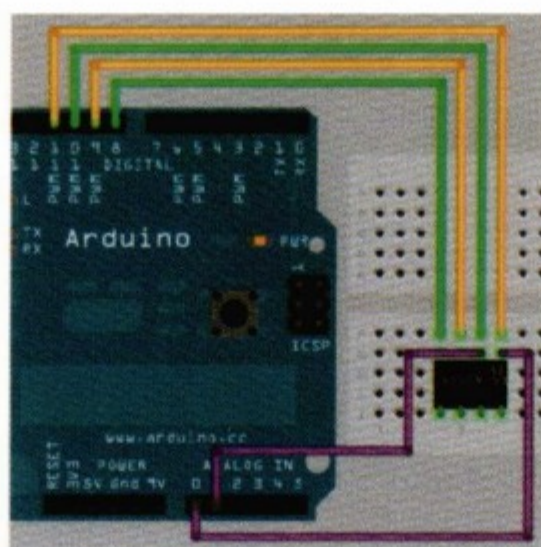


图 12-1 项目 33——基本的触摸屏电路图

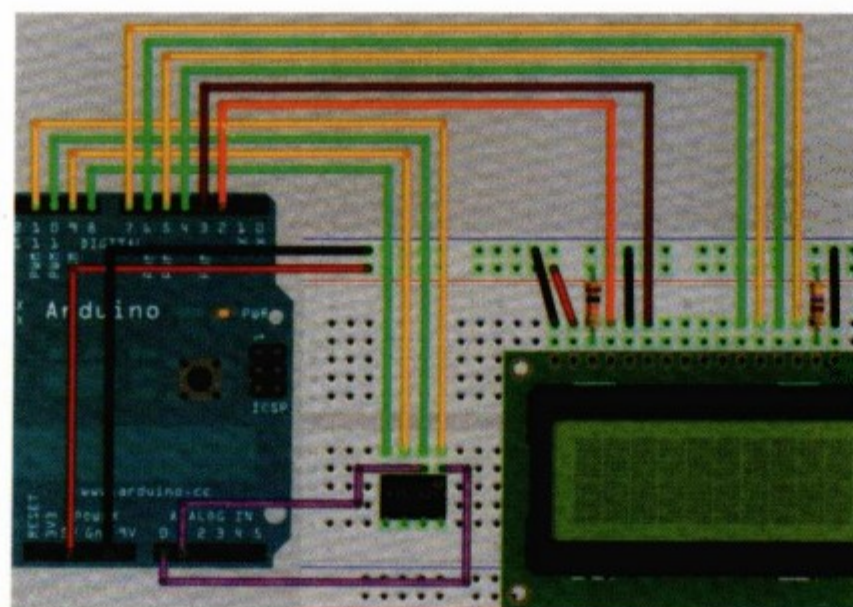


图 12-4 项目 34——触摸屏键盘电路图

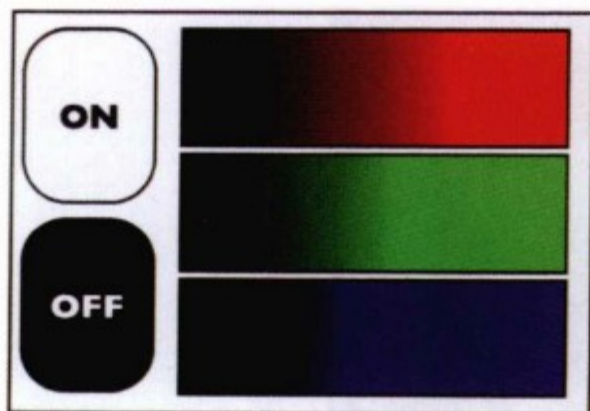


图 12-5 项目 35 的键盘图

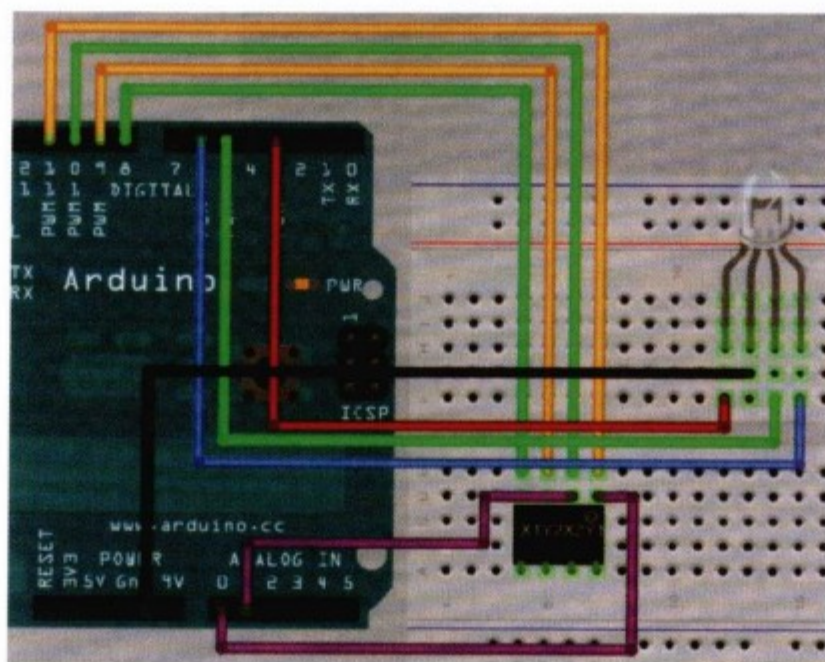


图 12-6 项目 35——触摸屏灯控制电路图

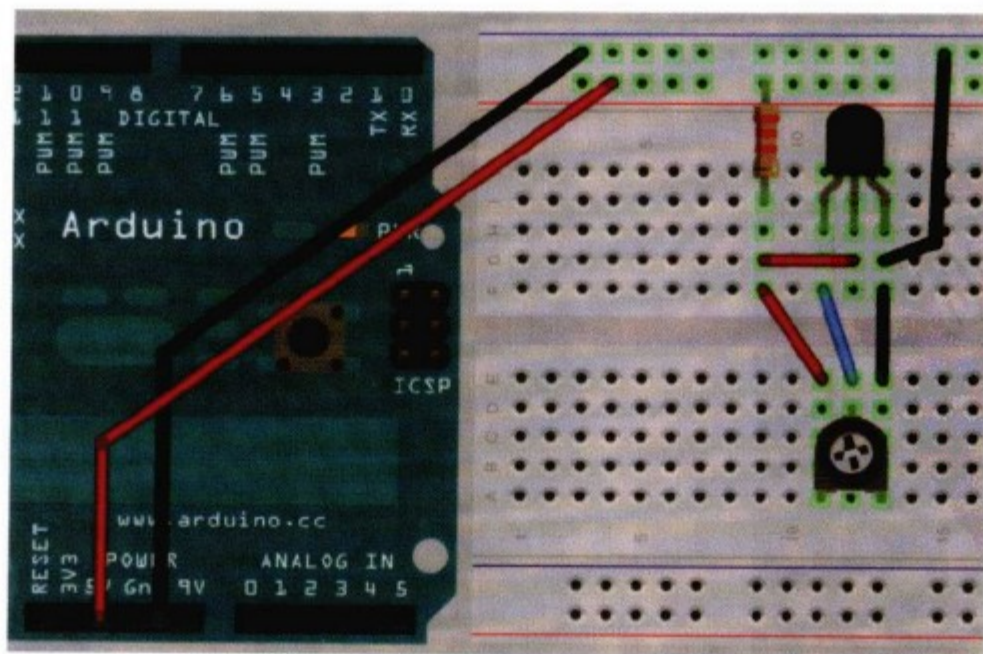


图 13-1 项目 36——串口温度传感器电路图

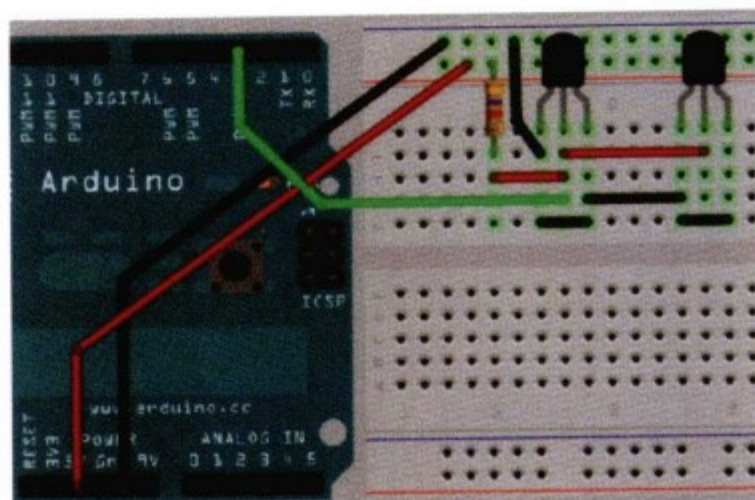


图 13-3 项目 37——单线数字温度传感器电路图

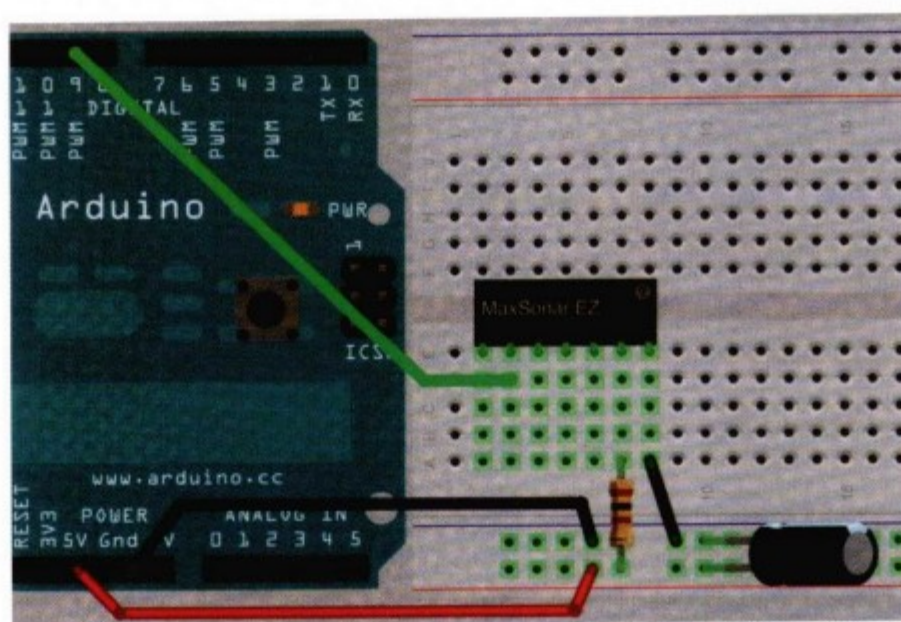


图 14-1 项目 38——简单的超声测距仪电路图

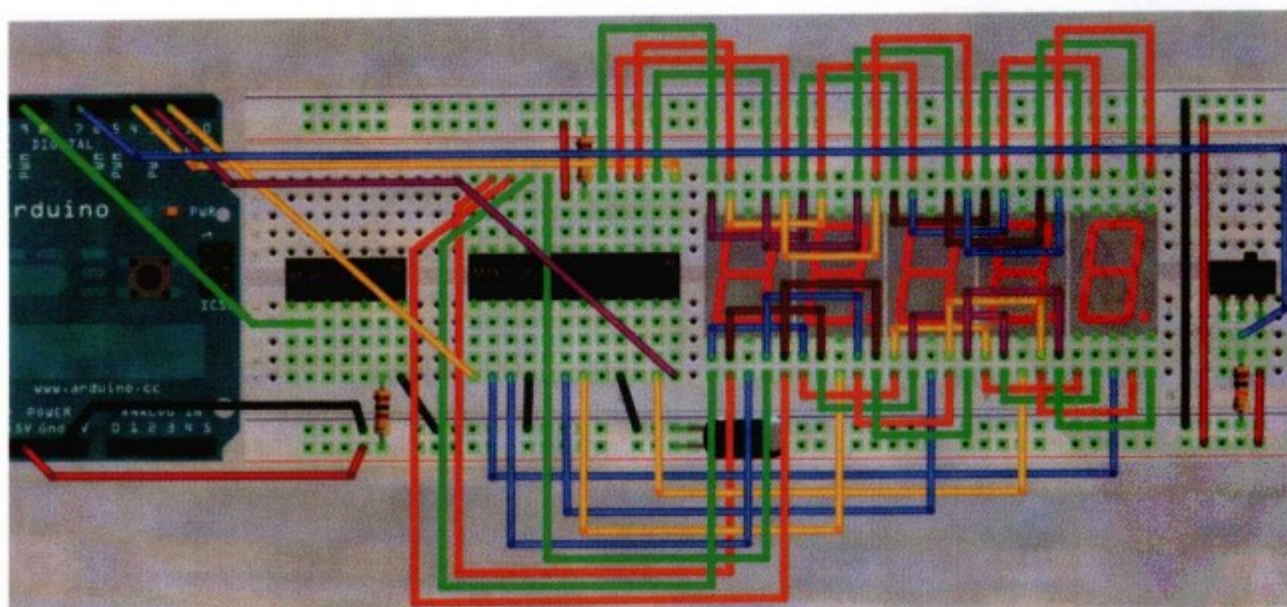


图 14-3 项目 39——超声测距显示仪电路图

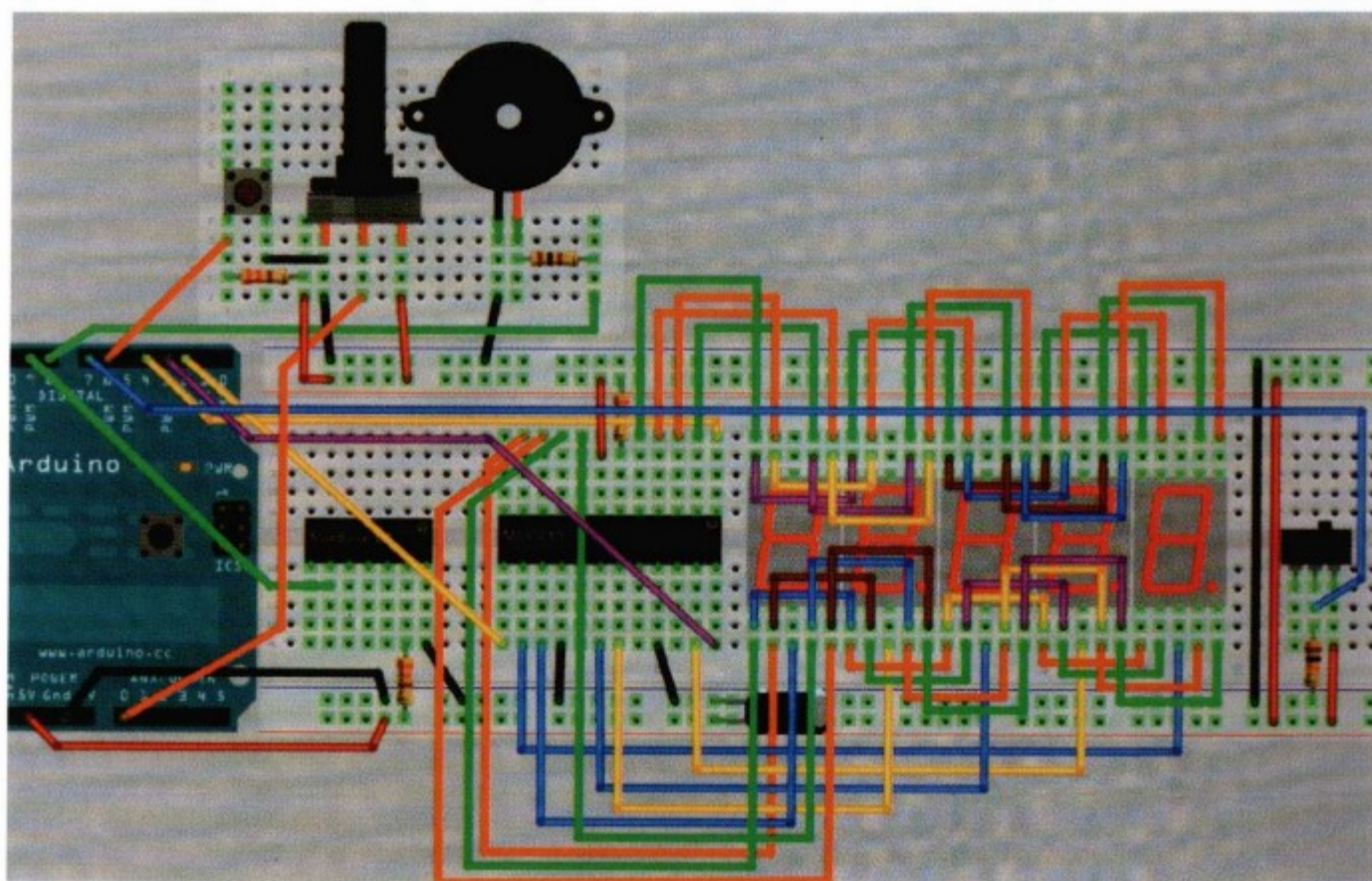


图 14-5 项目 40——超声报警电路图

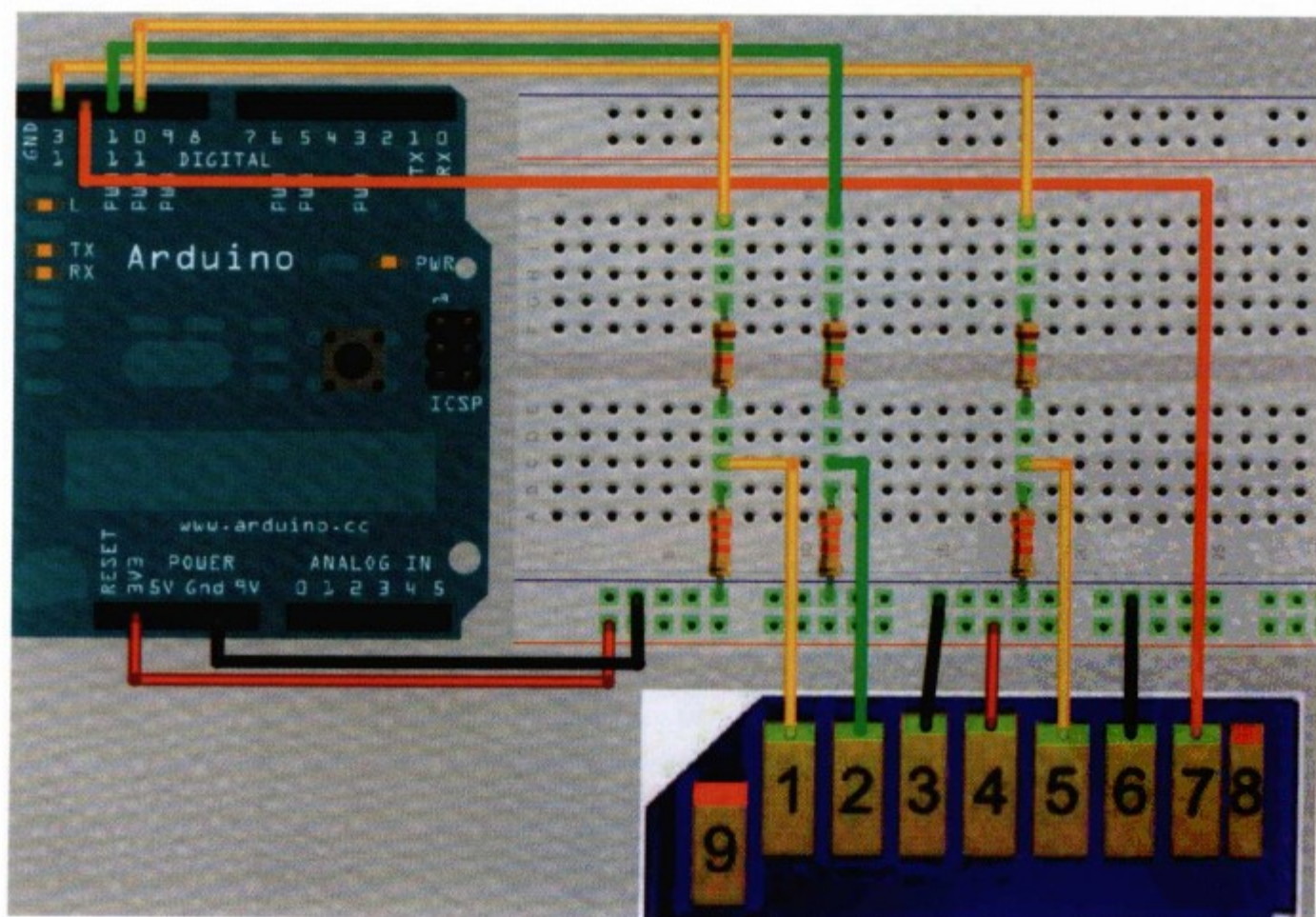


图 15-1 项目 42——简单的 SD 卡读写电路图

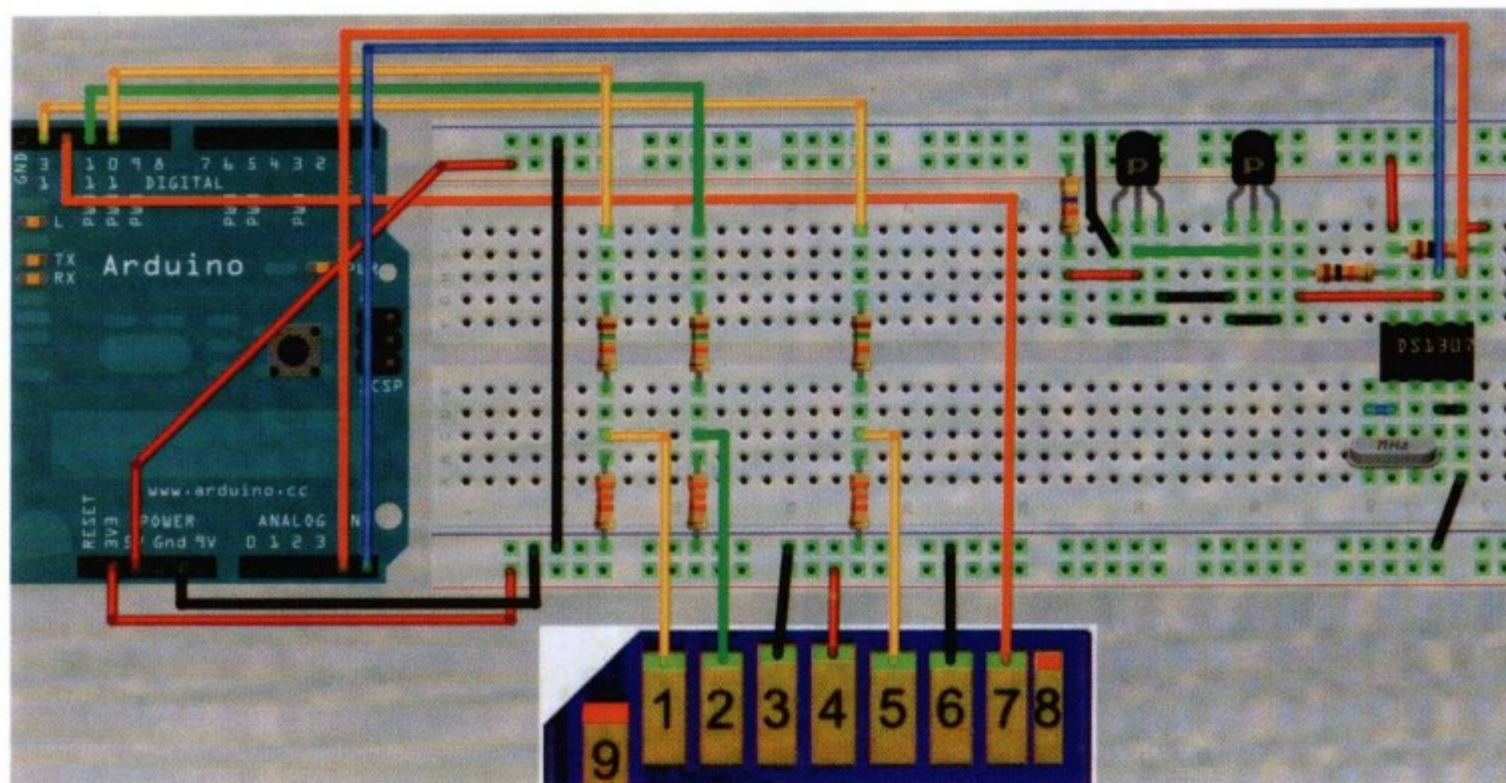


图 15-2 项目 43——用 SD 卡记录温度数据电路图

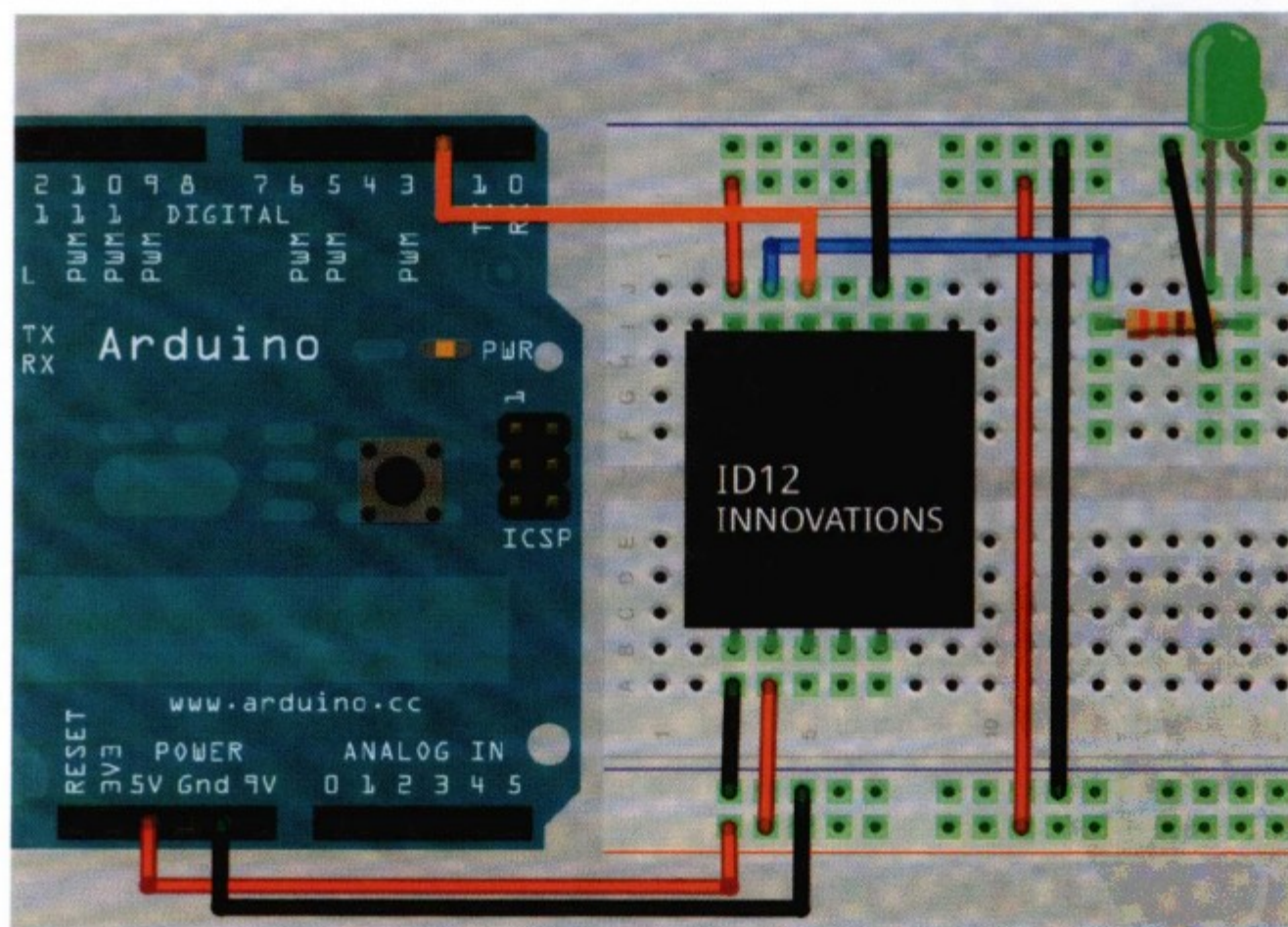


图 16-1 项目 44——简单的 RFID 读卡器电路图

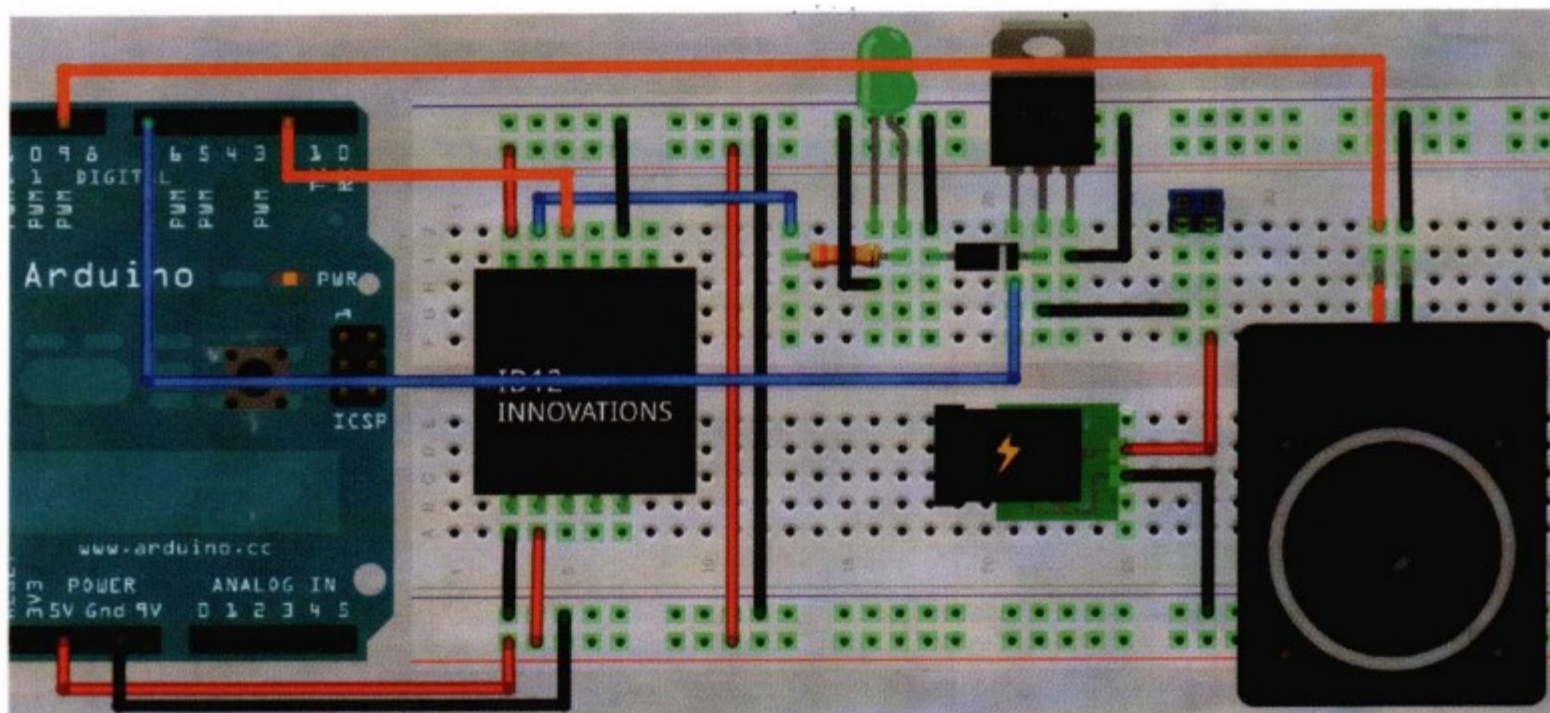


图 16-3 项目 45——门禁控制系统电路图

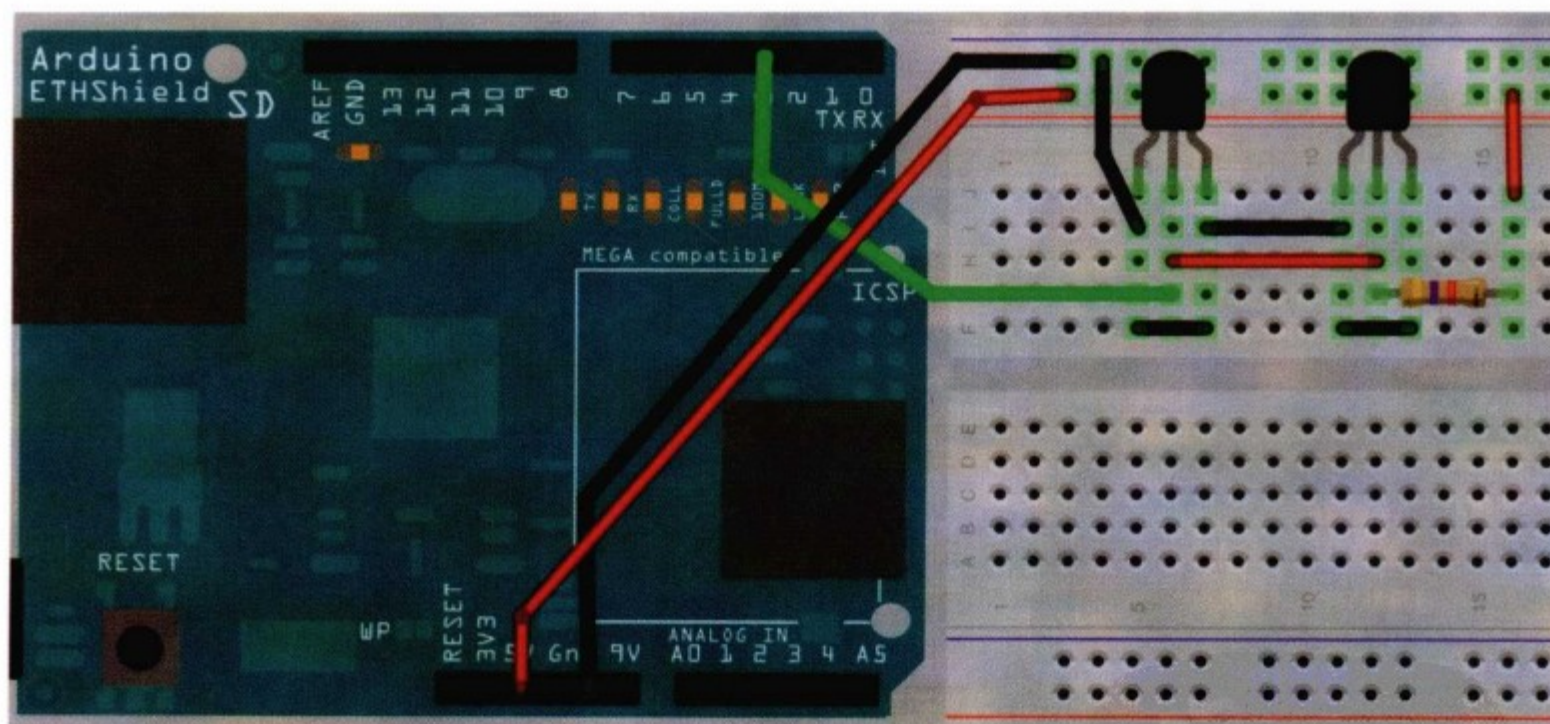


图 17-1 项目 46——Ethernet 板电路图

作者简介

米歇尔·麦克罗伯茨 (Michael McRoberts) 2008 年在天体摄影的小项目中制作云探测器，寻找将温度传感器连接到 PC 的方法时发现了 Arduino。经过研究，发现 Arduino 是解决这个问题的理想选择。最后很快成功地制作出了云探测器，而且价格便宜。米歇尔对 Arduino 的兴趣由此开始。自那以后，他采用 Arduino 完成了大量的工程项目。他同时在网上发现了被称为“地球之光电子” (Earthshine Electronics) 的 Arduino 入门套件及组件。他采用 Arduino 制作的下一个项目是在 UKHAS 和 CUSF 的搭档们的帮助下，将一个高海拔气球送到边缘空间进行拍照和录像。他做这个项目仅仅是为了好玩。



当米歇尔还是一个孩子的时候，就对电子产生了浓厚的兴趣。那时，无线器材公司还在利用一百合一电子套件制作圣诞礼物单。在他青少年时，得到一台辛克莱 81 计算机，开始对编程感兴趣。从那以后，他再也离不开计算机。最近，他又成为了 Mac 平台的粉丝。

他是伦敦黑客联盟 (London Hackspace) 和奥尔平顿天文协会的会员，他还经常为 Arduino 论坛写一些东西。他还喜欢以“Arduino 爱好者” (TheArduinoGuy) 为网名，潜伏在 Arduino、高海拔探测项目和伦敦黑客联盟 (一个位于伦敦的非盈利性的技术人员分享工具和知识的社区) 等论坛上，或在 Twitter 上与网民闲聊。闲暇之余，他也喜欢研究天文学、天体摄像，以及骑摩托车兜风和航海。

校订者简介

约翰·亚当斯（John Adams）是一名程序员和设计师，具有超过 9 年的产品质量检测软件和管理项目设计经验。他曾经为高校科研项目做过能够射出 27 英寸电火花的特斯拉线圈。作为 Isotope 11 网站的首席设计师，他负责审查架构决策，及将用户需求转换成工作软件。约翰毕业于阿拉巴马大学伯明翰分校，获得数学和哲学学位。约翰业余时间利用他的 Arduino 微处理器核对了本书的程序。不工作的时候，约翰喜欢和他的家人在一起。



致 谢

首先，我要感谢我的编辑 Michelle Lowman 和 Jennifer Blackwell，他们来自 Apress 出版社，没有他们，本书就无从谈起；我的技术校订 Josh Adams 详细地检查了我的代码和电路图，保证它们正确无误；Nancy Wright 找出了所有瑕疵。

非常感谢来自 Flickr 和 Wikimedia Commons 公司的人们，他们把自己的图片以创作公用许可方式给我使用，并允许我永久使用这些图片，他们包括 Bruno Soares、Richard V. Gilbank、Inductiveload、Snorpey、Iain Fergusson、Patrick H. Lauke、cultured_society2nd、Cyril Buttay、Tony Jewell、Tod E. Kurt、Adam Grieg、David Stokes、Mike Prevette、David Mitchell、Aki Korhonen、Alex43223、Sparkfun、DFRobot、Adafruit Industries、Colin M. L. Burnett、David Batley、Jan-Piet Mens、Mercury13、Georg Wiora 和 Timo Arnall。

感谢所有允许我使用或更改他们的代码或 Arduino 支持库的人，以及那些给我技术支持和建议的人，他们包括 Michael Margolis、来自 Pachube 的 Usman Haque、Georg Kaindl、Tom Pollard、Jim Studt、Miles Burton、Robin James、Paul Stoffregen、Cornor、Tom Igoe、Tim Newsome、James Whiddon、Bill Greiman、Matt Joyce、D.Sjunnesson、David A. Mellis、Bob S.(Xtalker)、Ian Baker 和 NeoCat。

感谢 Sparkfun 和 Adafruit 公司给我提供它们的元器件，并允许我使用它们的图片。同时感谢 Arduino 核心团队，没有他们，神奇的 Arduino 和它的社区将不会存在，他们包括 Massimo Banzi、Tom Igoe、David Cuartielles、Gianluca Martino、David Mellis 和 Nicholas Zambetti。

最后，感谢所有 Arduino 论坛、Arduino IRC 频道和在微博上给我提供帮助和建议的人，还有那些在这本书的整个出版过程中给予我鼓励的人，以及 London Hackspace（一个位于伦敦的非赢利性的技术人员分享工具和知识的社区）给我机会试验一些项目来写作最后一章。

如果我遗漏了某些人，我表示抱歉并同样感谢你们。

前言

我第一次接触 Arduino 是在 2008 年，当时我正在为云探测计划的温度传感器与计算机的连接方法发愁。我想实现一个气象论坛上的云探测想法，因为这是个实习性质的项目，所以我不想因项目失败而损失太多的钱。当时有许多解决方案，但是采用 Arduino 的方案对我来说有点特别，因为它不仅是容易使用、价格便宜的连接传感器的方法，而且它还可以用于其他很酷的事情。在博客、视频网站和论坛上有成千上万的用 Arduino 做得很炫的项目。看上去很有团队精神，每个人都试图帮助其他人。

显而易见，我可以从 Arduino 上得到很多乐趣。然而，我不想在网站上漫无目的地寻找需要的信息。我想要买一本关于 Arduino 的书籍，我想手边有些资料在工作中随时去翻阅。一番寻找后，我发现了一本书。但是，它太基础了，信息量小，几乎不能给我使用 Arduino 提供一点实际的帮助，而且我不喜欢那本书的讲授方式。我希望得到的是一本手册式的书来指导编程和学习电子知识，当我做一些东西时不想首先阅读一大串理论知识。这种简洁明了的书当时还不存在。

当我开始封装基于 Arduino 的“地球之光”项目工具的时候，为帮助使用这个工具，我写了一个小指导书指导其他人使用这个工具。这本小书后来非常流行，并且我从人们的提问中得到了几百条反馈意见，包括什么时候增加更多的项目或是否卖印刷版的指导手册。实际上，我已经想到了写一本初学者容易理解的书要采用这种项目讲授方式，即简单指导动手做的编写方式。这就是这本书的来源。

我写这本书假定读者之前从来没有编写过计算机程序，也没有电子知识，并且假定读者在实际动手用 Arduino 做一些事情前没有兴趣学习大量的理论知识。因此，当你使用这本书时就从做一些小项目开始，接下来你将要完成整整 50 个项目，直到你对 Arduino 开发精通。我相信学一项技术最好的方法是通过直接动手做些事情来学。

本书是这样编排的，第一个项目介绍一些关于 Arduino 编程和电子学知识的基本概念。

接下来的项目再增加一点，每一个项目都建立在之前项目的基础上，当你完成 50 个项目时，你就可以有信心并且专业地做你自己的项目了。用新的技巧和知识去连接 Arduino 的外围元件，根据你的兴趣用 Arduino 做电子设计项目。

每一个项目开始就列出了项目元件需求单。我选择的都是通用元件，很容易买到。我也提供了一个电路图直观演示如何使用跳线和面包板连接 Arduino 和元件。为给这本书生成元件图和面包板电路图，我用了一款优秀的开源软件 Fritzing。这个程序用逼真的方法演示面包板电路图及其他电路。读者可在 <http://fritzing.org> 查找这个软件。

当你连接电路之后，根据书中提供的代码，把它输入 Arduino 的开发环境（IDE）就可以上传到你的 Arduino 中，并使项目工作。很快就可以得到一个真正的项目运行结果。当项目工作并看到它的运行结果之后，我解释代码是如何工作的。硬件也通过这种方式向你解释它是如何工作的，以及如何正确地把它们连接到 Arduino。之后为你一步一步地解释代码，因此你会非常清楚代码每一部分的作用。通过分解电路和代码，你就能理解整个项目，之后这些技巧和知识可用在后面的项目及今后你自己的项目中。

这种讲授方式非常容易动手操作。即使你对编程和电子设计一点经验也没有，你也可以非常容易地以这种方式理解相关概念。更重要的是，你不会因大量枯燥的理论而丧失兴趣。Arduino 是强大、有趣、开源的产品，在这本书的帮助下，你会发现通过单片机使你自己的设备与环境互动是多么简单。

Mike McRoberts



博文视点诚邀精锐作者加盟

九载耕耘奠定专业地位

以书为证彰显卓越品质

《代码大全》、《Windows内核情景分析》、《加密与解密》、《编程之美》、《VC++深入详解》、《SEO实战密码》、《PPT演义》……

“圣经”级图书光耀夺目,被无数读者朋友奉为案头手册传世经典。

潘爱民、毛德操、张亚勤、张宏江、咎辉Zac、李刚、曹江华……

“明星”级作者济济一堂,他们的名字熠熠生辉,与IT业的蓬勃发展紧密相连。

九年的开拓、探索和励精图治,成就博古通今、文圆质方、视角独特、点石成金之计算机图书的风向标杆:博文视点。

“凤翱翔于千仞兮,非梧不栖”,博文视点欢迎更多才华横溢、锐意创新的作者朋友加盟,与大师并列于IT专业出版之巔。

英雄帖

江湖风云起,代有才人出。

IT界群雄并起,逐鹿中原。

博文视点诚邀天下技术英豪加入,

指点江山,激扬文字

传播信息技术,分享IT心得

专业的作者服务

博文视点自成立以来一直专注于IT专业技术图书的出版,拥有丰富的与技术图书作者合作的经验,并参照IT技术图书的特点,打造了一支高效运转、富有服务意识的编辑出版团队。我们始终坚持:

善待作者——我们会把出版流程整理得清晰简明,为作者提供优厚的稿酬服务,解除作者的顾虑,安心写作,展现出最好的作品。

尊重作者——我们尊重每一位作者的技术实力和生活习惯,并会参照作者实际的工作、生活节奏,量身制定写作计划,确保合作顺利进行。

提升作者——我们打造精品图书,更要打造知名作者。博文视点致力于通过图书提升作者的个人品牌和技术影响力,为作者的事业开拓带来更多的机会。



联系我们



北航

C1640652

博文视点官网: <http://www.broadview.com.cn>

CSDN官方博客: <http://blog.csdn.net/broadview2006/>

新浪官方微博: <http://weibo.com/broadviewbj>

腾讯官方微博: <http://t.qq.com/bowenshidian>

目 录

作者简介	iii
校订者简介	iv
致谢	v
前言	vi

■ 第 1 章 引言 1

如何使用本书	2
你需要的东西	2
Arduino 到底是什么?	3
可以开始了	6
在 Windows XP 上安装	7
在 Windows 7 或 Vista 上安装	8
在 Mac OSX 上安装	8
板子和接口的选择	9
加载第一个程序	11
Arduino 的 IDE	12

■ 第 2 章 让我们开始吧 19

项目 1——LED 闪灯器	19
需要的元件	19
连接所有的东西	20
输入代码	21
代码回顾	21
硬件回顾	26
项目 2——S.O.S 莫尔斯码信号源	31
代码回顾	32

项目 3——交通信号灯	35
需要的元件	35
把元件连起来	35
输入代码	36
项目 4——互动交通灯	37
需要的元件	38
把元件连接起来	38
输入代码	38
代码回顾	41
硬件回顾	45
逻辑状态	45
下拉电阻	46
上拉电阻	47
Arduino 的内部上拉电阻	48
小结	49
本章的主题和概念	49

第 3 章 LED 效果 51

项目 5——LED 跑马灯效果	51
需要的元件	51
把元件连接起来	51
代码回顾	53
项目 6——互动 LED 跑马灯效果	55
需要的元件	55
把元件连接起来	55
输入代码	56
代码回顾	57
硬件回顾	57
项目 7——闪烁灯	58
需要的元件	58
把元件连起来	59
输入代码	59
代码回顾	60

项目 8——RGB 彩灯	61
需要的元件	61
把元件连接起来	62
输入代码	62
代码回顾	63
项目 9——LED 火焰效果	67
需要的元件	67
把元件连接起来	68
输入代码	68
代码回顾	69
项目 10——串口控制彩灯	70
输入代码	70
代码回顾	73
小结	83
本章的主题和概念	83
第 4 章 简单的发声器和传感器	85
项目 11——压电声音报警器	85
需要的元件	85
把元件连接起来	85
输入代码	86
代码回顾	87
硬件回顾	88
项目 12——压电扬声器音乐演奏	89
输入代码	90
代码回顾	91
项目 13——压电震动传感器	95
需要的元件	95
把元件连接起来	95
输入代码	96
代码回顾	97
项目 14——光敏元件	98
需要的元件	98

把元件连接起来	99
输入代码	99
硬件回顾	100
小结	102
本章的主题和概念	102

■ 第 5 章 驱动直流电机 104

项目 15——简单的电机控制系统	104
需要的元件	104
把元件连接起来	105
输入代码	106
代码回顾	106
硬件回顾	107
项目 16——使用 L293D 电机驱动芯片	109
需要的元件	109
把元件连接起来	110
输入代码	111
代码回顾	112
硬件回顾	113
小结	115
本章的主题和概念	115

■ 第 6 章 二进制计数器 116

项目 17——移位寄存器 8 位二进制计数器	116
需要的元件	116
把元件连接起来	117
输入代码	117
二进制数制	119
硬件回顾	120
代码回顾	123
按位操作	125
代码回顾（继续）	128
项目 18——16 位二进制计数器	130

需要的元件	130
把元件连接起来	130
输入代码	131
代码和硬件回顾	133
小结	134
本章的主题和概念	134
第 7 章 LED 显示器	135
项目 19——LED 点阵显示器——基本动画	135
需要的元件	135
把元件连接起来	136
输入代码	137
硬件回顾	140
代码回顾	143
项目 20——LED 点阵显示器——滚动画面	146
输入代码	146
代码回顾	148
项目 21——LED 点阵显示器——滚动信息	151
需要的元件	152
把元件连接起来	152
输入代码	154
硬件回顾	159
代码回顾	163
项目 22——LED 点阵显示器——Pong 游戏	173
需要的元件	173
把元件连接起来	173
上传代码	174
代码回顾	175
小结	180
本章的主题和概念	180
第 8 章 液晶显示器	182
项目 23——基本的 LCD 控制	182

需要的元件·····	182
把元件连接起来·····	183
输入代码·····	184
代码回顾·····	188
硬件回顾·····	193
项目 24——LCD 温度显示器·····	194
需要的元件·····	194
把元件连接起来·····	194
输入代码·····	195
代码回顾·····	197
小结·····	201
本章的主题和概念·····	202
第 9 章 舵机 ·····	203
项目 25——舵机控制·····	204
需要的元件·····	204
把元件连接起来·····	204
输入代码·····	205
代码回顾·····	206
硬件回顾·····	207
项目 26——两个舵机控制系统·····	208
需要的元件·····	208
把元件连接起来·····	209
输入代码·····	209
代码回顾·····	211
项目 27——操纵杆控制舵机·····	214
需要的元件·····	214
把元件连接起来·····	214
输入代码·····	216
代码回顾·····	217
小结·····	218
本章的主题和概念·····	218

■ 第 10 章 步进电机和机器人	220
项目 28——基本步进电机控制	220
需要的元件	220
把元件连接起来	221
输入代码	222
代码回顾	223
硬件回顾	224
项目 29——使用电机模板	226
需要的元件	226
把元件连接起来	227
输入代码	228
代码回顾	230
硬件回顾	232
项目 30——巡线机器人	233
需要的元件	233
把元件连接起来	234
输入代码	236
代码回顾	239
小结	244
本章的主题和概念	244
■ 第 11 章 压力传感器	246
项目 31——数字压力传感器	246
需要的元件	246
把元件连接起来	247
输入代码	248
代码回顾	251
代码回顾（继续）	257
项目 32——数字气压表	262
需要的元件	262
把元件连接起来	263
输入代码	264
代码回顾	269

小结.....	275
本章的主题和概念.....	276

■ 第 12 章 触摸屏277

项目 33——基本的触摸屏	277
需要的元件.....	277
把元件连接起来.....	278
输入代码.....	279
硬件回顾.....	281
代码回顾.....	282
项目 34——触摸屏键盘	284
需要的元件.....	285
把元件连接起来.....	285
输入代码.....	286
代码回顾.....	289
项目 35——触摸屏灯控制	291
需要的元件.....	291
把元件连接起来.....	292
输入代码.....	293
代码回顾.....	295
小结.....	297
本章的主题和概念.....	297

■ 第 13 章 温度传感器.....298

项目 36——串口温度传感器.....	298
需要的元件.....	298
把元件连接起来.....	299
输入代码.....	300
代码回顾.....	301
项目 37——单线数字温度传感器.....	303
需要的元件.....	303
把元件连接起来.....	303
输入代码.....	304

代码回顾.....	309
小结.....	312
本章的主题和概念.....	312

■ 第 14 章 超声测距.....313

项目 38——简单的超声测距仪.....	313
需要的元件.....	313
把元件连起来.....	314
输入代码.....	314
代码回顾.....	315
硬件回顾.....	317
项目 39——超声测距显示仪.....	318
需要的元件.....	318
把元件连接起来.....	319
输入代码.....	321
代码回顾.....	322
项目 40——超声报警.....	326
需要的元件.....	326
把元件连接起来.....	327
输入代码.....	327
代码回顾.....	330
项目 41——超声电子音乐.....	333
输入代码.....	333
代码回顾.....	334
小结.....	335
本章的主题和概念.....	336

■ 第 15 章 读写 SD 卡.....337

项目 42——简单的 SD 卡读写.....	337
需要的元件.....	337
把元件连接起来.....	338
输入代码.....	338
代码回顾.....	342

项目 43——用 SD 卡记录温度数据	348
需要的元件	348
把元件连接起来	349
输入代码	350
代码回顾	356
硬件回顾	361
小结	363
本章的主题和概念	363

■ 第 16 章 RFID 读卡器

项目 44——简单的 RFID 读卡器	365
需要的元件	366
把元件连接起来	366
输入代码	367
硬件回顾	367
项目 45——门禁控制系统	368
需要的元件	369
把元件连接起来	369
输入代码	370
代码回顾	373
小结	380
本章的主题和概念	380

■ 第 17 章 连接到 Internet

项目 46——Ethernet 板	381
需要的元件	381
把元件连接起来	382
输入代码	382
代码回顾	386
项目 47——Internet 天气显示	391
输入代码	394
代码回顾	400

项目 48——电子邮件提醒系统	410
输入代码	410
代码回顾	414
项目 49——微博机器人	421
输入代码	421
代码回顾	425
项目 50——RSS 读取气象信息	431
输入代码	431
代码回顾	436
小结	446
本章的主题和概念	446



第1章

引言

自从 2005 年 Arduino 项目诞生到今天，全世界范围内已经卖出了超过 15 万块 Arduino 开发板。非官方的克隆开发板的数量无疑要多于官方的数量，所以很可能已经有超过 50 万块 Arduino 开发板和它的变种开发板正在被使用。Arduino 得到如此广泛的应用，是因为越来越多的人认识到采用 Arduino 开源项目无须太多的培训就能简便快捷地开发出微控制器应用。

相对于其他微控制器开发平台，Arduino 最大的优点是它的易用性。业余玩家也可以轻松掌握它的基础，并在相当短的时间里开发出自己的项目。特别是艺术家，他们发现 Arduino 是一个在没有专门的电子知识的情况下快速开发互动电子艺术品的最理想的工具。有大量使用 Arduino 的人在分享他们的代码和电路图，以便其他人复制和修改。这个群体中的大部分人都十分愿意去帮助其他人。你可以在 Arduino 论坛中快速找到你所需要的答案。

尽管互联网上有大量的信息供初学者使用，但这些信息大部分通过各式各样的资源来传播，这使得获得完整的信息变得很难。本书主要为你解决这个困难。本书专门设计了 50 个项目，带领读者一步一步通向 Arduino 应用之路。当你第一次拿到一块 Arduino（或者其他诸如此类的小玩意儿）时，你肯定想插上电源，连上一个 LED，使它闪烁起来，你绝不希望先阅读大量的通篇理论才能做以上事情。本书作者理解那种立即动手就做的兴奋感，这也是为什么你乐意马上把东西连接到 Arduino 上，上传代码，开始动手试验。我相信即学即用是学习一个项目最好的办法，特别是计算机硬件项目，这也是 Arduino 所关注的。

如何使用本书

本书从 Arduino 的简要介绍开始，接着是如何准备搭建硬件、安装软件、上传第一个项目并确保你的 Arduino 开发板和软件运行正常。之后介绍 Arduino 的编程环境，指导你在完成逐渐变得复杂的项目之前如何使用编程环境。每一个项目都是从介绍如何搭建硬件和使硬件工作起来所需的代码开始的，然后我将阐述一些硬件和代码如何工作的细节。所有论述都简捷明了，并使用大量图表和照片使你更容易正确地跟着这些项目进行练习。

在本书中你开始可能会遇到一些不懂的名词和概念，不用着急，这些名词和概念会随着项目进一步的深入而逐渐变得清晰。

你需要的东西

为了能够做本书中的项目，你需要各种各样的元件，这可能要花很多钱，所以我建议你开始时只购买头几个项目中所需要的元件（元件清单在项目开始处已经列明）。随着学习的深入，你再不断添置后面项目需要的元件。

另外还有大量需要或有用的东西。你需要有一块 Arduino 开发板，或者市场上任意一种克隆板，如 Freeduino、Seeeduino（没错，就是这样拼写的）、Boarduino、Sanguino、Roboduino，或者任何一种其他各种各样的“duino”，所有这些都与 Arduino 的 IDE、Arduino 的外围板及其他所有的官方 Arduino 板相关的东西兼容。记住，Arduino 是一个开源项目，因此任何人都可以自由免费地制作 Arduino 的克隆板或者其他变种板。然而，如果你想要支持 Arduino 原始开发团队，就从一个注册分销商那里购买一块官方板。截至 2010 年 10 月，Arduino 的最新版本是 Arduino Uno。

你需要通过 Internet 下载 Arduino 的 IDE（集成开发环境——这个软件是用来编写 Arduino 程序的），也可以下载本书的示例代码（如果你不想自己把代码输入计算机）及其他各种代码库，这些库文件有可能是你的程序运行所必需的。

你还需要一张光线充足的桌子或其他工作台来布置你的各种元件。桌子或工作台最好靠近你的台式计算机或笔记本电脑以便于上传代码到 Arduino 中。记住，你是在与电器打交道（虽然是低压直流电），你在布置元件时，一些金属面板需要用非导体材料（如桌布或纸张）覆盖。如果有可能，可以准备一些电线钳、扁嘴钳和剖线钳。手边再放一打便笺纸和铅笔，以便画图或记下概念和设计灵感等。

最重要的是，你需要充满热情和求知欲地去学习。Arduino 是作为一个简单廉价的通向微电子控制器世界的工具而被设计的。如果你准备放手一试，它没有特别难学的理论。这本书将帮助你踏上这个旅程并指导你养成这个充满激情和创意的业余爱好。

Arduino 到底是什么？

维基百科描述：“Arduino 是一块单板的微控制器和一整套的开发软件，它的硬件包含一个以 Atmel AVR 单片机为核心的开发板和其他各种 I/O 板。软件包括一个标准编程语言开发环境和在开发板上运行的烧录程序。”

考虑到 Arduino 主要是为业余爱好者的使用而设计的，所以它被设计成一个小计算机的形式，它使你可以给连接到 Arduino 开发板上的外部输入输出器件编程（见图 1-1）。Arduino 就是所谓的嵌入式计算机平台，这意味着它是一个交互式系统，可以通过硬件和软件与它的环境进行互动。例如，一个简单的 Arduino 应用项目是在按下一个按钮时，点亮一盏小灯并保持一段时间，比如说 30 秒。这个例子中必须有一盏小灯和一个按钮与 Arduino 相连。Arduino 可以一直处于等待按钮被按下的状态，一旦按钮被按下，Arduino 就点亮那盏小灯并开始计时，当它计时到 30 秒时，Arduino 熄灭小灯并且等待下一次按钮被按下。你可以将这个小程序应用在一个小壁橱中。

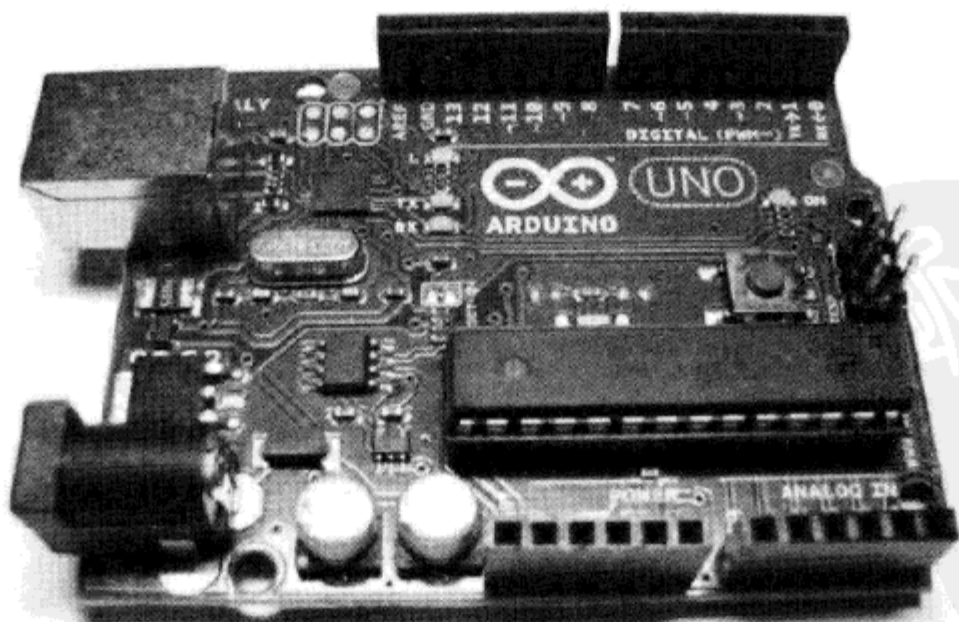


图 1-1 一块 Arduino Uno 开发板

你可以把以上项目通过连上其他传感器进行扩展，如一个红外探头，当红外探头被触发时点亮那盏小灯。以上是一些使用 Arduino 的简单例子。

Arduino 可以用于开发孤立的互动项目，也可以将它与计算机相连，形成一个网络，甚至可以与互联网相连，用来接收或发送数据并按指令做出相应的动作。换句话说，它可以发送一些来自于传感器的数据到网络，这些数据可以以图表的形式显示出来。

Arduino 可以与 LED、点阵显示器（见图 1-2）、按钮、旋钮、小电机、温度传感器、压力传感器、距离传感器、GPS 接收机、以太网卡或者其他能够输出数据或被控制的任何东西相连接。随便到互联网上浏览一下，就会获得用 Arduino 读取数据或者控制各类电子原件的丰富资源。

Arduino 开发板由一块 Atmel AVR 单片机、一个晶振或振荡器（一种天然的时钟，可以以特定的频率发出时间脉冲使 Arduino 以正确的速度运行）和一个 5V 的直流电源组成。根据你所使用 Arduino 类型的不同，可能需要一条 USB 线来连接你的 PC 或 Mac 计算机，用来下载程序或接收数据。Arduino 开发板引出了 Atmel AVR 单片机的所有 I/O（输入、输出）引脚，你可以连接这些引脚到其他电路或传感器。

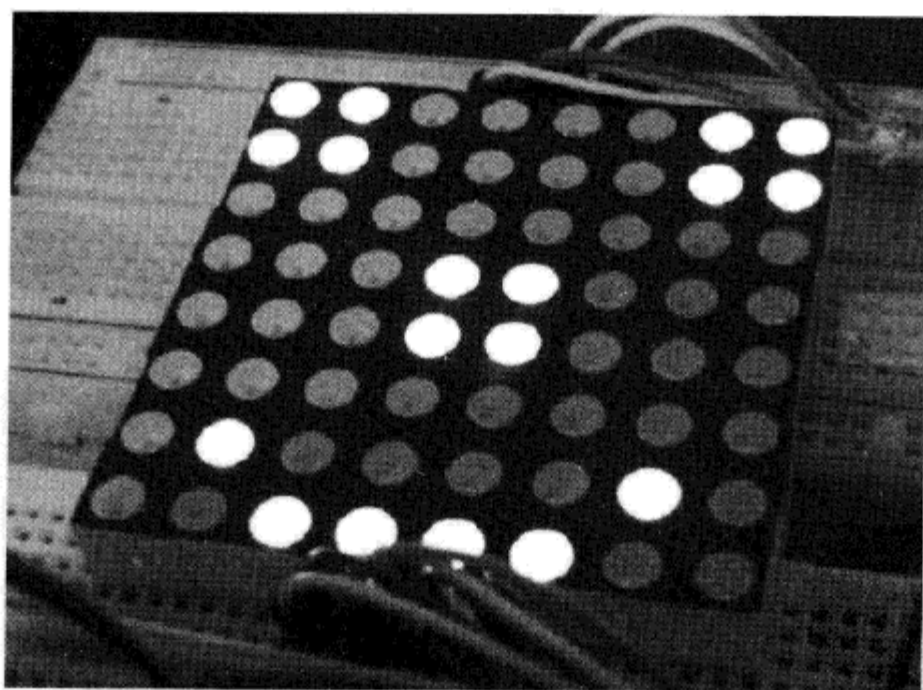


图 1-2 Arduino 控制一个点阵显示器

最新的 Arduino 开发板——Uno，不同于以前的各种 Arduino 开发板，它不再使用 FTDI 的 USB 到串口驱动芯片，而是把 Atmega8U2 编程为一个 USB 到串口转换器，这给它带来了一系列不同于其上代 Duemilank 开发板的好处。首先，Atmega 芯片比 FTDI 芯片便宜得多，这使得整个板子的价格有所下降。第二，也是最重要的一点，它可以使 USB 芯片刷写开发板的固件引导程序，使 Arduino 在 PC 端显示为一个 USB 设备，就像一个鼠标或游戏操作杆那样。这开辟了一系列 Arduino 的新用途。不幸的是，使用这种 USB 芯片使得克隆

板制造者克隆 Arduino Uno 的难度大大增加了。

给 Arduino 编程（让它做你希望它做的事情）要使用 IDE（集成开发环境），它是一款免费的软件，使用它你可以用 Arduino 可理解的语言（类似于 C 语言的编程语言）开发 Arduino 代码。IDE 使你可以编写计算机程序。这是一种一步一步解释性的语言，你可以下载编好的程序到 Arduino，你的 Arduino 就开始解释执行这些程序，并且与连接设备互动。在 Arduino 世界里，程序也叫做架构。

Arduino 的硬件和软件都是开放的资源，这意味着代码、图表、设计等可以被任何人自由地获得，并用它们做任何他们喜欢做的事情。因此，可以买到大量的克隆板或基于 Arduino 的开发板，或者可使用同一个原理图自己做一个。实际上，你可以随意购买合适的元件在面包板上或者你自己的 PCB（印制电路板）上制作属于自己的 Arduino，唯一需要注意的是你不能使用“Arduino”这个词，这个名字是保留给官方版 Arduino 的。克隆板可以以诸如 Freeduino、Roboduino 等名字命名。

因为设计方案是公开的资源，任何克隆板都 100%兼容官方版 Arduino，所以任何软件、硬件、图表等也 100%兼容正宗的 Arduino。

Arduino 可通过其他板进行扩展，它们是一些电路板，包含了其他元件（如 GPS 接收机、LCD 显示器、网络模块等），你可以很方便地把它们连接到 Arduino 开发板上以获得拓展功能。拓展板能把 Arduino 的引脚引到它自己的电路板上，因此在拓展板上你可以使用 Arduino 的每一个引脚。如果你不想用拓展板，也可以不用。你可以使用面包板、针孔板、万用板或者你自己的 PCB 制作完全相同的电路。本书中的大多数项目都是在面包板上制作的。

Arduino 有各种各样的版本。最新版本是 Arduino Uno。之前的版本中最流行的是 Duenilanove（2009 年意大利产），也就是你从互联网上看到的大多数项目所使用的板子。你也可以得到 Arduino 小型的、微型的、基于蓝牙的变种开发板。另外一个新增产品线是 Arduino Mega 2560，它提供了更大的存储空间和更多的 I/O 引脚，这种开发板使用了一个新的烧录器，叫做 Optiboot，它释放了大约 1.5KB Flash 存储空间，并能更快速地启动。

Arduino 系列中最通用、最受欢迎的板子是 Uno 或它之前的版本 Duenilanove。这是因为它采用一个标准的 28 针 IC（一个完整的电路）插座来安装 Arduino 上的单片机。这样做的优点在于，如果要用 Arduino 做点事情，并想把它转化成永久的东西，不必使用相对较贵的 Arduino 开发板，你可以把芯片从 IC 插座上拿下来，并把它安装到你自已制作的电路

板上。这样，你可以做一个个性的嵌入式系统。这是相当方便的。

然后多花几元钱，你可以用一个新的 AVR 芯片替换你的 Arduino 上的芯片。记得这个芯片一定要事先编好 Arduino 下载程序（一种软件，运行在芯片上，使它能使用 Arduino IDE）。你也可以购买一块 AVR 芯片自己去烧写启动程序，或直接买一块已经烧好的。大多数 Arduino 元件供应商提供这些支持。也可通过另外一个 Arduino 来烧写一块芯片。关于这些事情的说明可在线找到。

如果你以“Arduino”为检索词在网上做一次搜索，你会吃惊地发现大量的网页是关于 Arduino 的，很多很棒的项目是用它开发的。Arduino 是一个让人着迷的东西，它能使你做任何东西，从艺术互动产品（见图 1-3）到机器人。有一点学习如何进行 Arduino 编程及如何使它与其他元件互动的激情，再加上一点想象力，你就可以做任何你能够想到的东西了。



图 1-3 Richard v. Gilbank 制作的用 Arduino 控制的心理学艺术装置

本书告诉你如何开始这项有吸引力、有创造力的爱好的必要技能。现在你已经知道 Arduino 是什么了，让我们坐到计算机前使用它吧！

可以开始了

本节首先告诉你在 Windows 和 Mac（使用 OSX 10.3.9 或以后的版本）操作系统上如何安装 Arduino 及 IDE。如果你使用 Linux，安装方法参见 Arduino 网站，网址是 www.arduino.cc/playground/Learning/Linux，此处假定你使用的板子是 Arduino Uno，如果你

使用其他板子，如 Duemilanove（见图 1-4），可参考 Arduino 网站上入门指导中的相应内容。

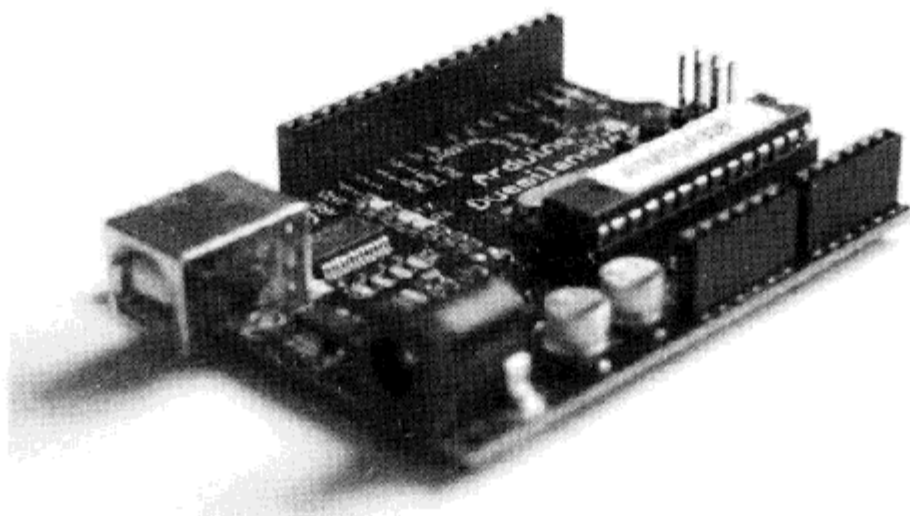


图 1-4 Arduino Duemilanove（感谢 Snorpey 供图）

你需要准备一根 USB 电缆（A 到 B 插口形式的），就是大多数 USB 打印机用的那种 USB 线。如果你有一块 Arduino Nano，你就需要一根 A 到 mini-B 接口形式的 USB 电缆。先不要动手把线接到 Arduino 上，需要时我会告诉你。

下一步，下载 Arduino 的 IDE，它是用来编写程序（或架构）并上传程序到你的开发板上的软件。到 <http://arduino.cc/en/main/software> 下载最新版本的 IDE，注意所下载版本要与你所用的操作系统匹配。

在 Windows XP 上安装

如果你下载了最新版本的 IDE，解压文件后双击已解压的文件夹打开它。你在里边能看到 Arduino 的文件和子文件夹。现在将 USB 电缆插入你的 Arduino 板，确保绿色电源 LED（标志为 PWR）灯亮。Windows 显示“Found new hardware: Arduino Uno”，然后新硬件对话框向导出现。单击“Next”按钮，Windows 尝试安装驱动程序，这个过程可能会失败，别着急，这是正常的。

然后右击桌面上的“我的电脑”图标，选择“管理”选项，打开“计算机管理”对话框，找到系统工具列表中的“Event Manager”，打开它。在右侧的窗格中将看到设备列表。Arduino Uno 就在列表内，并且带有一个黄色问号图标，这表示该设备没有正确安装。右击图标选择升级驱动程序，在第一个界面上选择“No, not this time”，然后单击“Next”按

钮，选择“Install from a list or specific location (Advanced)”再单击“Next”按钮。现在单击“Include this location in the search”并浏览。找到解压后的 Arduino IDE 文件夹中的 Drivers 文件夹，之后单击“Next”按钮，Windows 将安装驱动程序，最后单击“Finish”按钮。

这时 Arduino Uno 将出现在下方的设备列表中，并且显示你给它注册的端口号（如 COM6）。双击文件图标可打开 IDE。

在 Windows 7 或 Vista 上安装

如果你下载了最新的 IDE，解压文件，双击已解压文件夹打开它，你将会看到 Arduino 文件和子文件夹在里边。然后，用 USB 电缆连接你的 Arduino 开发板，并确保绿色电源 LED（标志为 PWR）灯亮。Windows 将尝试给 Arduino Uno 自动安装驱动程序，这个过程可能会失败，别着急，这是正常的。

单击 Windows 的“开始”按钮，然后单击“控制面板”，之后单击“系统和安全”按钮，接着单击“系统”按钮。在左侧的列表中单击“设备管理”选项。Arduino 出现在设备列表中，在它上面有黄色的问号图标，表明它还没有正确安装，右击“Arduino Uno”选项，然后选择“Update Driver Software”选项。

之后选择“Browse my computer for driver software”选项，在下一个界面单击“浏览”按钮。在你之前已解压的文件夹中找到 Arduino 文件夹，单击“OK”按钮后单击“Next”按钮。Windows 将尝试安装驱动程序，单击“Install this driver software anyway”选项，安装驱动程序软件现在开始做它自己的事情了。如果一切正常，将出现一个写着“Windows has successfully updated your driver software”的对话框，单击“Close”按钮。双击文件夹内的 Arduino 图标可打开 IDE。

在 Mac OSX 上安装

下载最新的 IDE 光盘镜像文件（后缀为.dmg），打开后缀为.dmg 的文件，将出现一个如图 1-5 所示的界面。

拖动 Arduino 图标到 Application 文件夹图标上并放在那里。如果使用一个旧版的 Arduino，如 Dvemilannive，你需要安装 FTDI USB 串口驱动。双击打包的图标，按照说明去做，对于 Uno 和 Mega 2560 不需要安装任何驱动。

进入应用文件夹，双击 Arduino 图标就能打开 IDE。

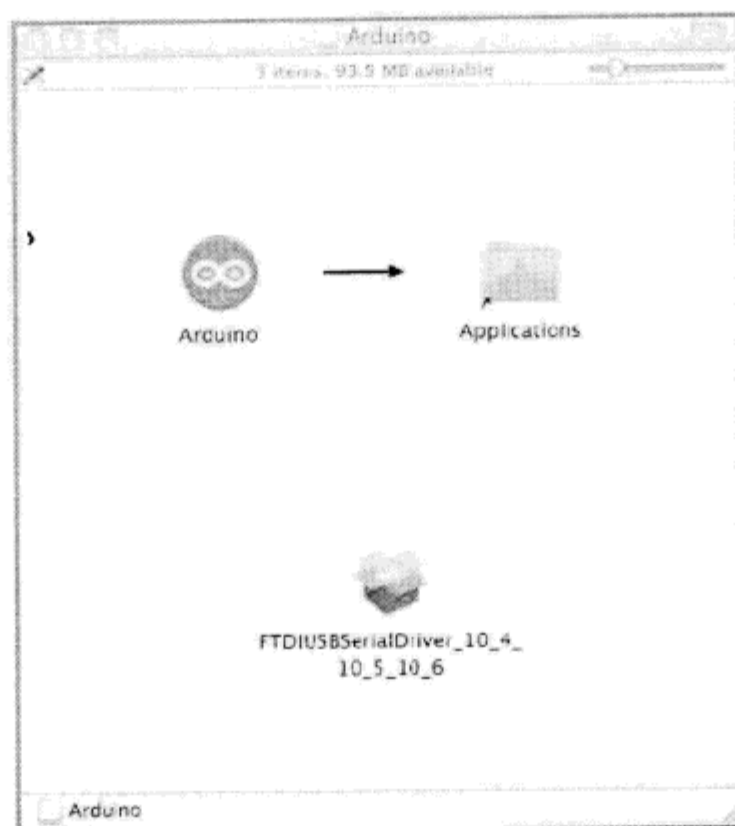


图 1-5 在 OSX 上打开 Arduino.dmg 文件

板子和接口的选择

当你打开 IDE 时你将看到如图 1-6 所示的窗口。



图 1-6 第一次打开 Arduino IDE

现在单击菜单上的 Tools，之后单击 Board（见图 1-7）。



图 1-7 Arduino 的 Tools 菜单

这时会看到一个开发板的列表（见图 1-8），如果你有一块 Uno，选择它，如果你有 Duemilanove 或其他 Arduino 开发板，从列表中选择相应的开发板。

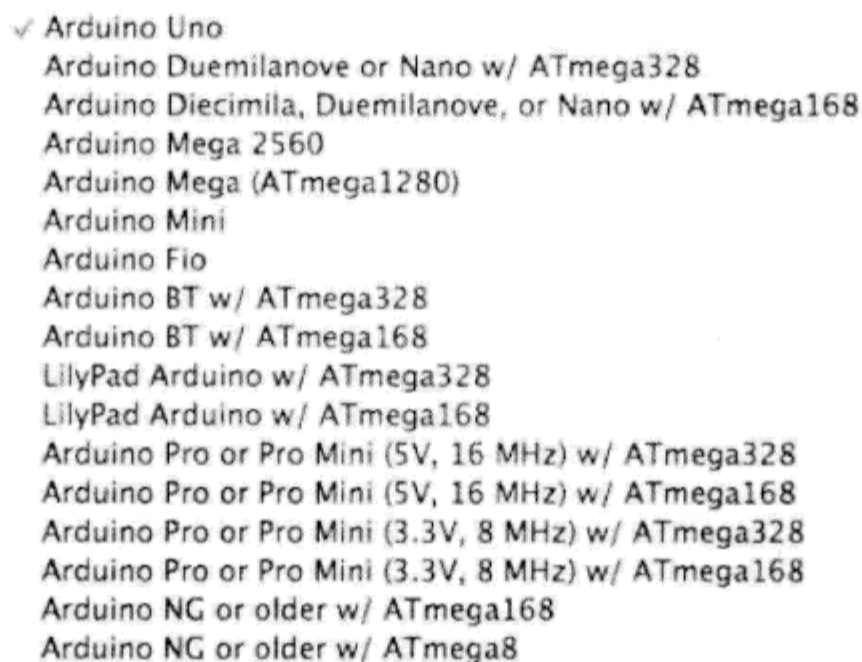


图 1-8 Arduino 的 Board 菜单

之后再单击 Tools 菜单，单击 Serial Port，在出现的列表中选择你的 Arduino 使用的接口（见图 1-9）。现在你已准备好上传一个示例程序去检验安装是否正确了。

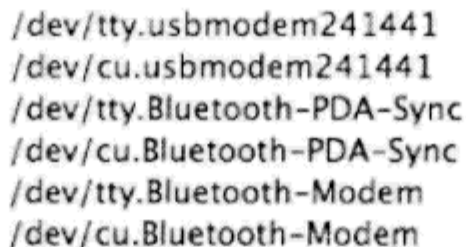


图 1-9 串行接口列表

加载第一个程序

现在你已经安装了驱动程序和 IDE，并且已经选择了相应的开发板和串口。在进行你的第一个项目之前，可以加载一个程序到 Arduino 来检验各种准备是否已经做好了。

首先，单击 File 菜单（见图 1-10），之后单击 Examples。



图 1-10 File 菜单

你可以看到一大串例子程序，我们先试一个简单的，单击 Basics，之后单击 Blink（见图 1-11），这时 Blink 程序将出现在 IDE 中。

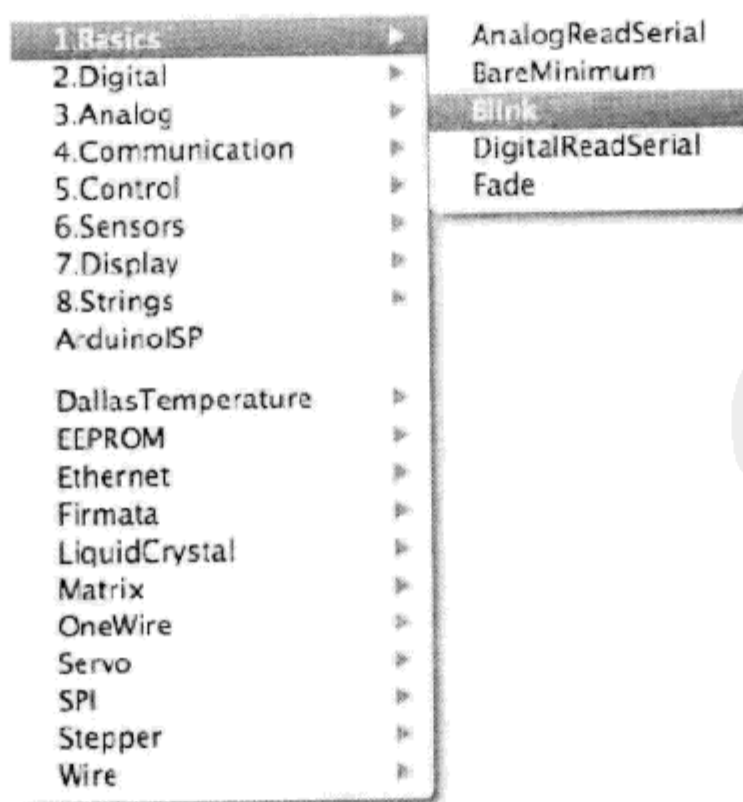


图 1-11 Examples 菜单

之后，单击 Upload 按钮（从左数第六个按钮），然后注意你的 Arduino 开发板（如果

你的 Arduino 开发板是 Arduino Mini、NG 或者其他型号，你需要在单击 Upload 按钮前先单击重启按钮），RX 和 TX 灯开始闪烁，表明你的计算机和开发板之间正在交换数据。如果程序已经成功加载，“Done Uploading” 出现在 IDE 状态条中，并且 RX 和 TX 灯停止闪烁。

几秒之后，你会看到引脚 13 的 LED（小的 LED 灯，靠着 RX 和 TX LED 灯）开始每隔 1 秒闪烁一次。如果结果是这样的，表明你已经成功地连接到 Arduino 开发板，安装了驱动程序和软件，并且成功地加载了示例程序。Blink 程序是一个非常简单的程序，它的功能是使 LED 闪烁，如图 1-12 所示。13 号 LED 是一个焊接在板上的、小的、绿色（或者橘色）的 LED（与单片机的数字引脚 13 连接）。在进入项目 1 之前，我们先看一下 Arduino 的 IDE，下面将详细解释这个软件的每一部分。

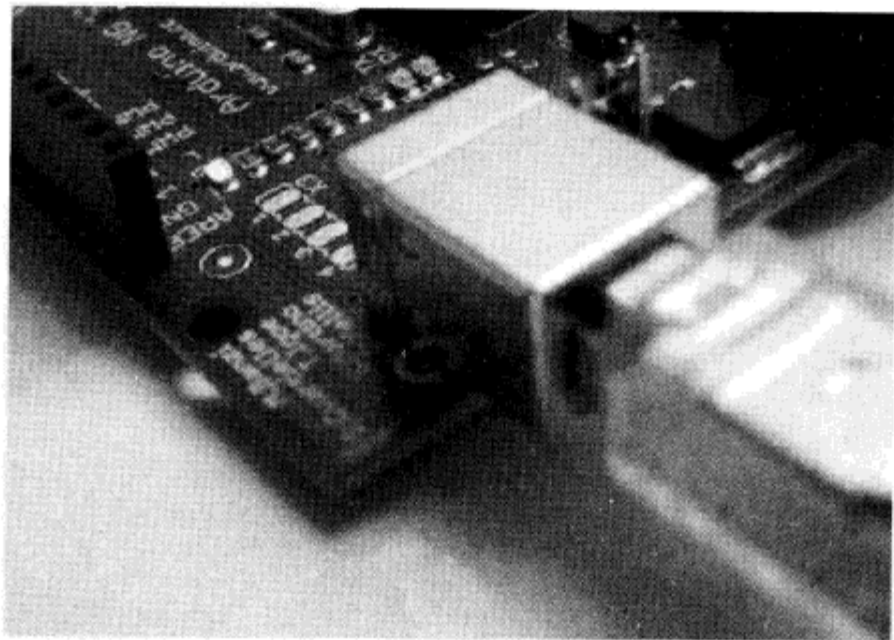


图 1-12 13 号 LED 闪烁

Arduino 的 IDE

当你打开 Arduino 的 IDE 时，它看起来和图 1-13 非常相似，如果你使用 Windows 或者 Linux，可能会有一点小小的不同，但是不管怎样，IDE 在任何操作系统上看起来都差不多。

IDE 界面分成三个部分。顶部是工具栏，代码窗口在中间，消息窗口在底部。工具栏中包含 7 个按钮。在工具栏下边是一个或一系列标签，标签上有程序的文件名，在右端还有一个按钮。

上边是文件菜单，有下拉菜单，包括 File、Edit、Sketch、Tool 和 Help。为了方便使用，最常用功能的按钮放置在菜单工具栏上（见图 1-14）。

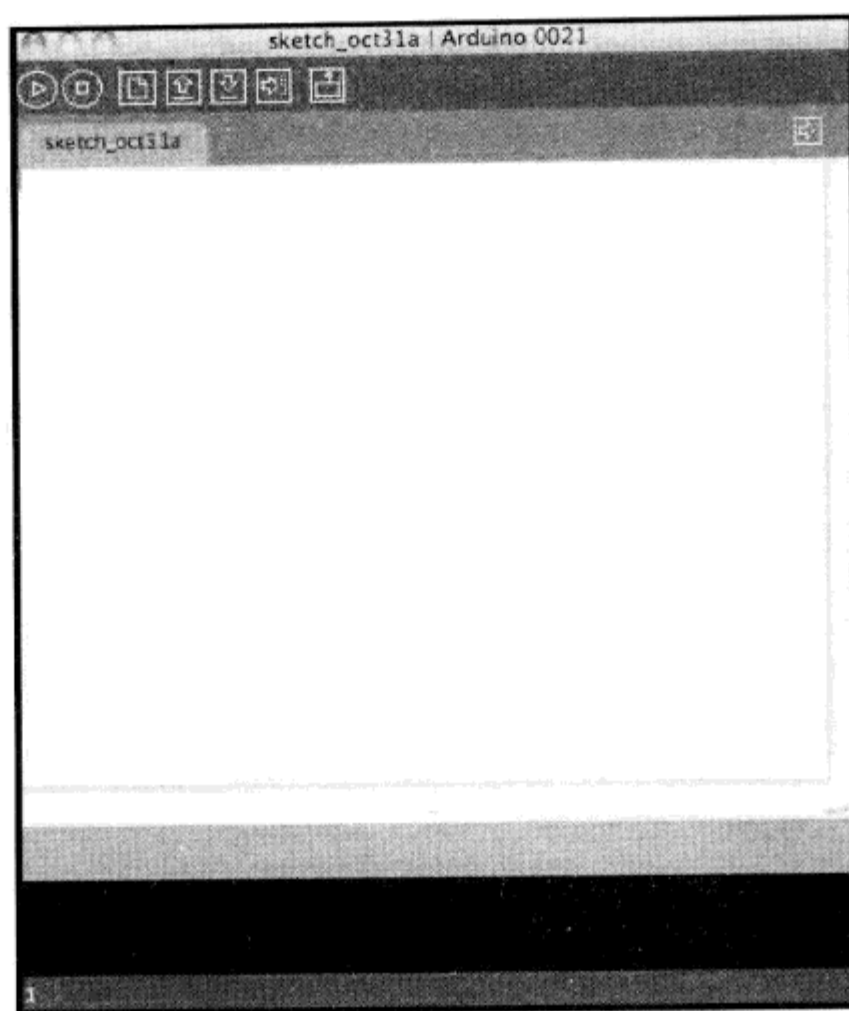


图 1-13 当打开应用程序时 Arduino IDE 的样子

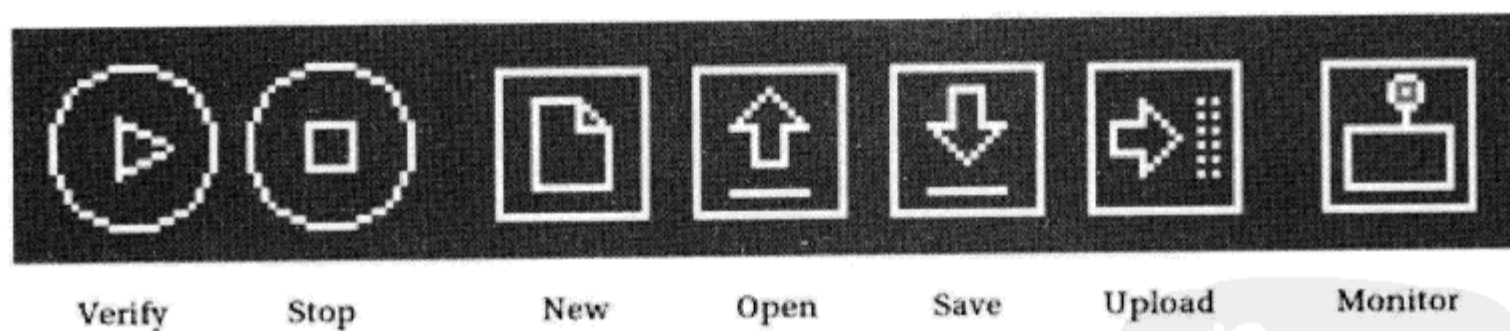


图 1-14 工具栏

表 1-1 列出了工具栏按钮和它们的功能。

表 1-1 工具栏按钮及功能

Verify/Compile	检查代码是否有错误
Stop	停止串口监视或其他高亮状态的按钮的高亮
New	生成一个新的空白框架
Open	打开架构库中的一个架构
Save	存储当前架构到架构库
Upload	上传当前代码到 Arduino
Serial Monitor	显示从 Arduino 发来的串口数据

Verify/Compile 按钮用来在加载代码到 Arduino 之前检查所编代码是否正确。

Stop 按钮用于停止串口监视操作，它也可以停止其他高亮状态按钮的高亮。当串口监视正在操作时单击 Stop 按钮，将在串口数据中获得一个中断点以便进行更详细的检查。如果你正发送数据到串口，速度太快以至于你已来不及读这些数据，此时这项功能是特别有用的。

New 按钮用于生成一个新的空白架构，你可以在里边输入代码，IDE 会提示你输入文件名和文件存储位置（如果可能尝试用默认的位置），然后给你一个空白的构架去写代码。顶部的标签显示你已经输入的文件名。

Open 按钮用于在构架单中显示已存在的构架的列表，就像例子架构那样。你可以使用不同的外围设备运行这些程序。例子构架对初学者非常有用，是编制自己的架构的基础。根据你的设备打开相应的程序，你可以按自己的需要连接或修改这些代码。

Save 按钮用于存储架构窗口中的代码到文件中。一旦完成存储，你在当前代码窗口的底部会看到一个“Done Saving”信息。

Upload to I/O Board 按钮用于加载当前架构窗口中的代码到你的 Arduino 中。在上传之前一定要确保已经选择了正确的开发板型号和端口。在加载架构到 Arduino 开发板之前一定要保存你的程序，防止意外错误引起系统死机或 IDE 崩溃。在加载之前单击 Verify/Compile 按钮也是必要的，确保程序首先没有错误需要调试和排除。

Serial Monitor（串口监视器）是非常有用的工具，特别是在调试程序时。串口监视器显示从你的 Arduino 开发板（USB 口或串口）上输出的串口数据，你也可以通过串口监视器向 Arduino 传送数据。单击串口监视器按钮后将出现类似于如图 1-15 所示的窗口。

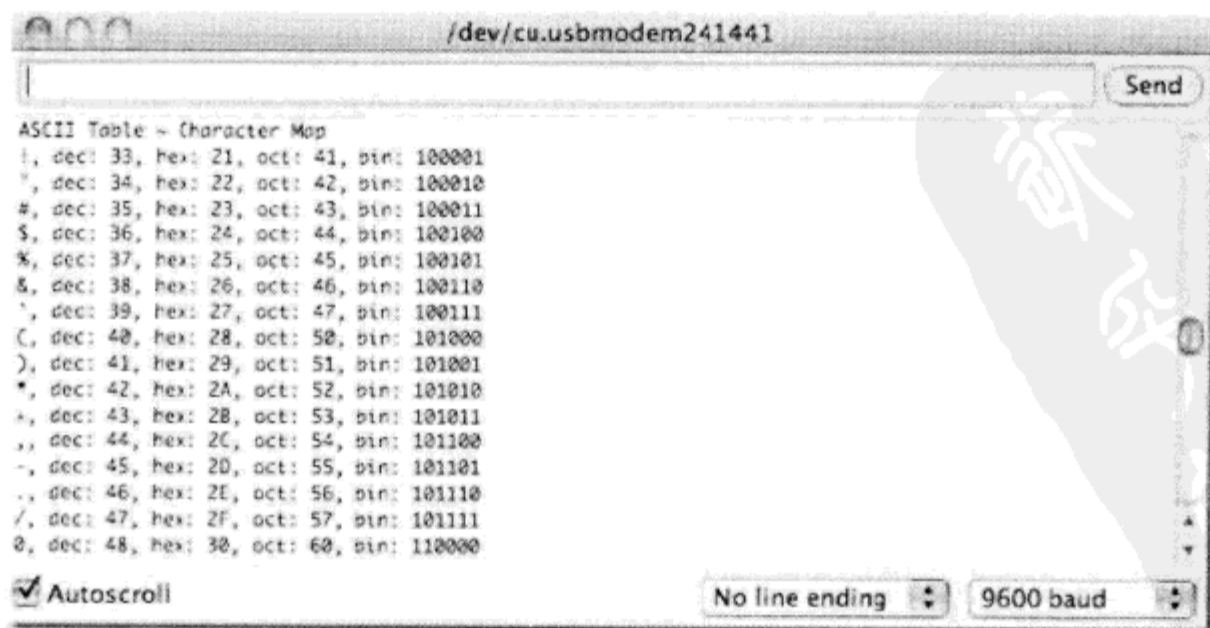


图 1-15 使用中的串口监视器窗口

在右下方你可以选择从 Arduino 发送或接收数据的波特率。波特率是每秒从 Arduino 开发板发送或接收状态的（或比特数据）变化率。默认的波特率是 9600，这意味着如果你要通过串口连接线（此处指的是 USB 电缆）发送一个字符记录，那么将每秒发送记录中的 1200 个字母或符号（ $9600\text{bit}/8\text{bit}$ 每字符=1200 字节或字符），注意比特和字节的概念将在后边讲到。

在串口监视器窗口顶部是一个空的文本框，你可单击 Send 按钮把其中的字符传送给 Arduino。注意，如果没有在代码中编写串口通信程序，串口监视器就不会接收串行数据。同样地，Arduino 也会不接受任何串口发送的数据，除非你已经在上传到 Arduino 的程序中编写了串口通信代码。

最后，空白处是串行数据显示的地方。在图 1-15 中，Arduino 运行在 ASCII 码状态（互相通信的例子程序）。在这个程序里，Arduino 通过串口（USB 电缆）输出 ASCII 码字符到 PC，用串行监视器显示这些字符。

单击 Serial Monitor 按钮可启动串行监视器，单击 Stop 按钮可结束串行监视器。在 Mac 或 Linux 操作系统中，当你单击 Serial Monitor 按钮时，Arduino 板会自己重启（从程序开始处重新执行）。

如果你已经精通通过串口从 Arduino 接收数据或发送数据的通信方法，可以用其他程序，如 Processing、Flash、MaxMsp 等实现 Arduino 与 PC 之间的通信。当你用 Arduino 从传感器中读数据、通过串口把数据发送到 PC 并需要以人能理解的形式显示时，你就要用到串行监视器。

在 IDE 窗口的底部，你可以看到出错信息（以红色的字符显示）。出错信息将在尝试与板子相连、下载代码或改变代码时出现。在 IDE 底部的左侧，你可以看到一个数字，这是目前光标在程序中所在的位置。如果你已经在你的窗口中写下代码，向下浏览程序（在键盘上使用 ↓ 键）你会看到这个数字增加为你向下浏览到的行代号。这对于发现错误是非常有用的，错误将用高亮的错误信息表示出来。

IDE 窗口的顶部（如果你使用 MAC 操作系统，就是横穿屏幕的顶端），你可以看到各种菜单，单击一个菜单可以看到更多的子菜单（见图 1-16）。

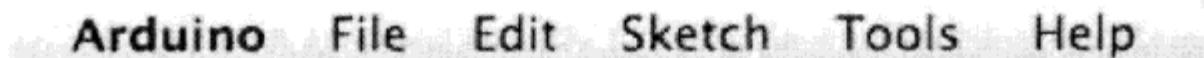


图 1-16 IDE 的菜单

第一个菜单是 Arduino 菜单（见图 1-17），About Arduino 选项显示目前 IDE 的版本号、一个致力于制作这个有吸引力东西的人员名单及其他的信息。

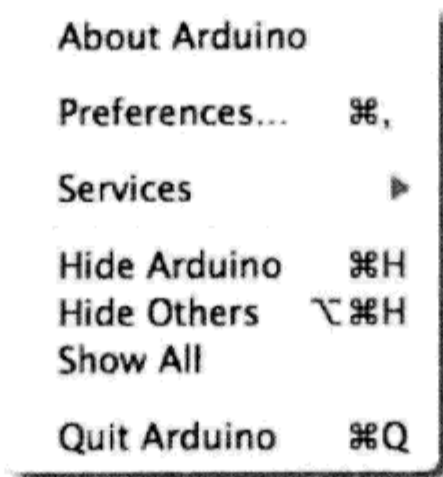


图 1-17 Arduino 菜单

紧接下来是 Preferences 选项，它显示专业配置窗口，在那里你可以改变 IDE 的各种配置，如默认架构库的位置。Quit 选项表示关闭当前程序。

File 菜单（见图 1-18）包含的功能有生成新的架构、查看架构在架构库中存储的情况（如例子架构）、存储架构及使用 Save As 选项给架构重新命名、上传新建架构到 I/O 板（Arduino）、把代码打印出来。

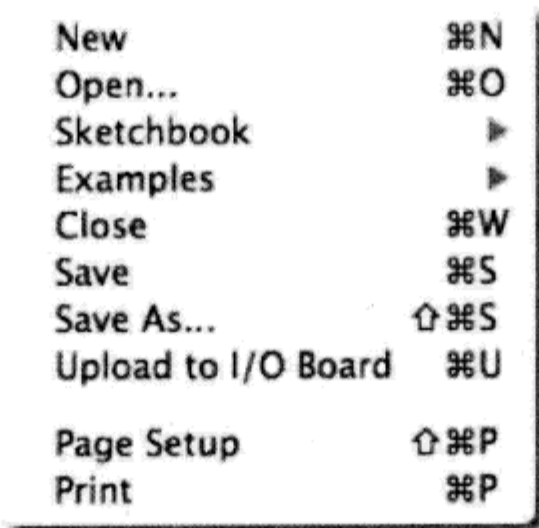


图 1-18 File 菜单

Edit 菜单提供了剪切、复制、粘贴部分代码的功能。你也可以在代码中选择全部或查找词或词组。Undo 和 Redo 的在编辑出错时会派上用场。

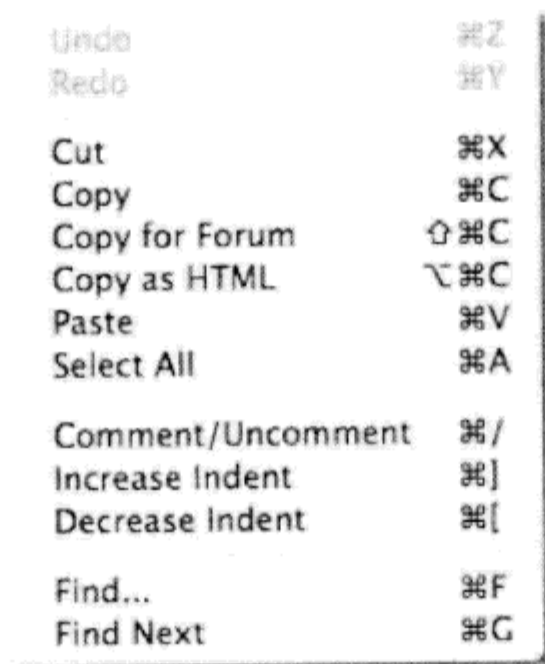


图 1-19 Edit 菜单

Sketch 菜单（见图 1-20）包含更改/编辑和其他有用功能，包括引入库功能。它将存储在库文件夹中的库用列表的形式显示出来。

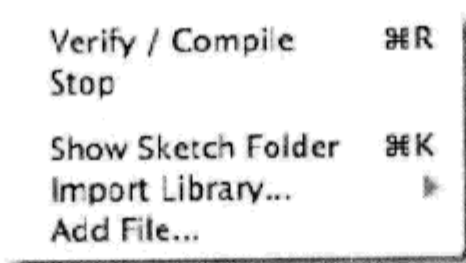


图 1-20 Sketch 菜单

库是一个代码的集合，可插入架构以提高项目的功能。这种方法避免了大量重复劳动，你可以使用别人已经为各种相同的硬件写好的代码。例如，Stepper 库是控制步进电机的一系列功能库。热心人已经做好了控制步进电机所有必要的功能，因此通过插入 Stepper 库到架构中，就可以使用这些功能去控制电机。通过存储公用代码到库中，可以在项目中一遍一遍地重复使用这些代码。你也可以将代码最复杂的部分隐藏起来。后面将更详细地说明如何使用库。

Show Sketch Folder 选项用于显示架构存储文件夹。Add File 选项用于增加资源文件到当前架构中，这样你可以把一个大的架构分成几个小的文件，然后把它们加到这个主要的架构中。

Tools 菜单（见图 1-21）提供多项操作功能。通过这个菜单你可以在第一次启动 Arduino

开发板时选择开发板型号和串口。Auto Format 功能格式化所编代码，使代码看上去更整齐。Copy for Forum 选项用于用统一的格式来复制架构窗口中的代码，当用这项功能把代码粘贴到 Arduino 论坛中（或其他论坛）时，代码看起来的样子与在 IDE 中的一样，包括语法、颜色等。Arduino Sketch 选项用于将架构压缩成 Zip 格式的文件，并提示你输入存储文件夹。最后，Burn Bootloader 选项用于烧写 Arduino 的引导程序（芯片上的一小段代码，使它与 Arduino IDE 能兼容）到芯片上。这个操作只能在以下情况下使用：你有一个 AVR 编程器且已放置在你的 Arduino 中或在你自己的嵌入式项目中采用一个空白芯片。除非你想烧写许多芯片，否则只买一个已经预编程了 Bootloader 的 Atmega 芯片（见图 1-22）通常是比较便宜和容易的方法。许多网上商店出售便宜的预编程芯片。



图 1-21 Tools 菜单



图 1-22 一块 Atmega168P 芯片，Arduino 开发板的核心（感谢 Earthshine Electronics 供图）

菜单的最后一项是 Help，在这里你可以找到许多关于 IDE、连接到 Arduino 的参考网址，或其他有用的网址信息。Arduino 的 IDE 是非常简单的，你可以非常快速、容易地学会如何在项目过程中使用它。当你非常专业地使用 Arduino 并采用 C 语言（Arduino 中写代码的一种编程语言）编程时，你会发现，Arduino 的 IDE 是非常基础的。如果你需要使用更高级的功能，你可以尝试专业的 IDE 程序（它们中的一些是免费的），如 Eclipse、ArduIDE、GNU/Emacs、AVR-GCC、AVR studio，甚至苹果的 Apple's X Code。

现在你已经安装了 Arduino 软件，开发板已经连好，并开始工作了。你已经掌握了 IDE 的一些基本用法，下面我们进入项目 1——LED 闪灯器。

第2章

第2章

让我们开始吧

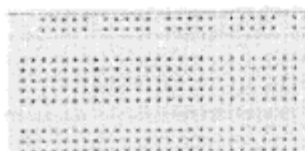
现在从4个项目开始你的工作。这些项目都是用不同的方法控制LED。你将学会像控制按钮输入一样控制Arduino的各种输出。在硬件方面，你将学习到有关LED、按钮和电阻的内容，包括上拉和下拉电阻的知识，这对于掌握正确的输入数据和读出数据方法是非常重要的。在这个过程中，你将接触到Arduino编程的概念。让我们从一个最基本的项目“Hello World”开始学习，这个项目的内容是使用Arduino控制一个外部LED的闪烁。

项目1——LED 闪灯器

在第一个项目中你将重复使用试验阶段用过的LED blink 程序。所不同的是，这次你要连接外部LED 到一个数字引脚，而不是使用焊在开发板上的LED 13。你也会详细地了解这个项目中硬件和软件是如何工作的。然后再学一点电子知识，同时学点用Arduino 语言（这是一种变种的C语言）编写代码的知识。

需要的元件

面包板



5mm LED



100Ω电阻*



跳线



*这个阻值根据你使用的 LED 的不同而不同，后面会说明如何计算这个阻值。

本书中涉及的大多数项目最好用 840 针的面包板。还有大量其他标准的面包板，840 针面包板尺寸是 $16.5\text{cm} \times 5.5\text{cm}$ ，在板上有 840 个孔或针。通常面包板边上有个小的燕尾槽，通过它你能够把几个同样的面包板连接起来形成一个更大的面包板。这对于一些复杂的项目是非常有用的。然而对于本项目，任何尺寸的面包板都行。

可采用 5mm 直径的 LED，颜色不限。你需要知道 LED 的电流和电压（有时也叫最大电流、最高电压），根据这些数据你可以计算所需的电阻值。我们将在本项目的后面计算电阻值。

跳线既可以从市场上购买（通常线终端都按统一标准制作，以便于插入面包板），也可以使用自己制作的。将一小段单芯信号线端部剥掉 6mm 左右的绝缘层即可做成跳线。

连接所有的东西

首先，拔掉 USB 电缆保证 Arduino 电源关闭。现在，取出你的 Arduino 开发板、LED、电阻、跳线，并把所有的东西按照图 2-1 的形式连接起来。

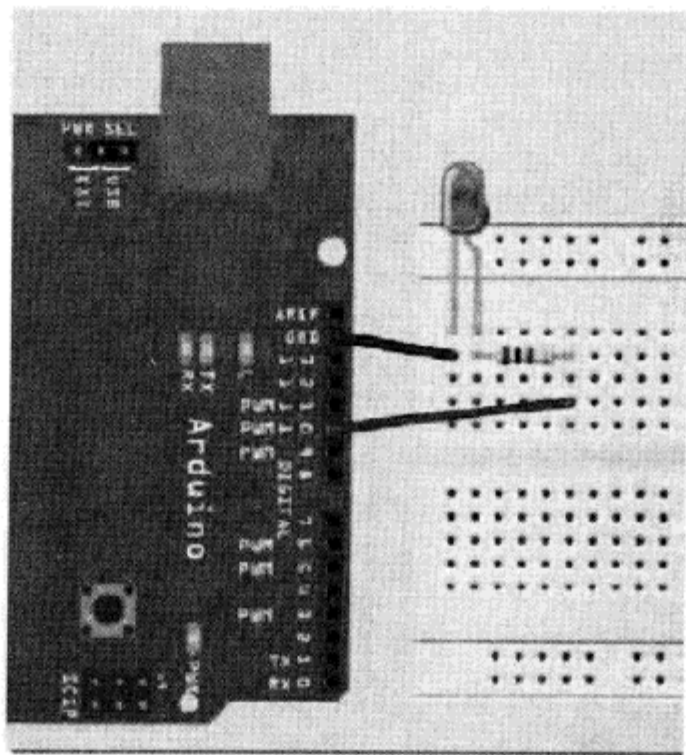


图 2-1 项目 1——LED 闪灯器电路图（参见彩色版插图）

也可以使用其他颜色的线，使用面包板上其他的孔也没有关系，只要元件和线的连接顺序与上图一样即可。把元件插入面包板时一定要小心。如果面包板是新的，孔内的夹子会有点紧，插放元件时不小心可能会发生危险。

确保 LED 连接是正确的。长脚要连接到数字引脚 10 上，LED 的长脚是它的正极，必须与正 5V 电源相连（在这里，从数字引脚 10 引出 5V 电源），短脚是阴极，必须要连接 GND（地）。当你确定所有的连接都正确后，给 Arduino 上电接上 USB 电缆。

输入代码

打开你的 Arduino IDE，输入如清单 2-1 所示的代码。

清单 2-1 项目 1 的代码

```
//项目 1 —— LED 闪灯器
int ledPin = 10;
void setup() {
    pinMode(ledPin, OUTPUT);
}
void loop() {
    digitalWrite(ledPin, HIGH);
    delay(1000);
    digitalWrite(ledPin, LOW);
    delay(1000);
}
```

按下 IDE 上部的 Verify/Compile 按钮，确保输入的代码没有错误。如果成功，单击 Upload 按钮上传代码到你的 Arduino 中，如果正确地做了每件事，你可以看到面包板上的红色 LED 每隔 1 秒交替亮灭 1 次。

现在让我们回顾一下代码和硬件，看看它们是如何工作的。

代码回顾

代码的第一行如下所示：

```
//项目 1 —— LED 闪灯器
```

这是你的代码中的说明文字。可以叫它们注释，因为它是以“//”开始的，这个符号后

面所有的文字编译器都将忽略。注释在代码中是非常有用的，它帮助你理解代码是如何工作的。如果你的项目非常复杂，代码超过了几百行，甚至几千行，注释是非常重要的，它使你很容易知道每一段代码的作用。你可能会写出一段迷人的代码，但是，你不可能永远记得它是如何工作的，而注释可以使你回忆起代码的功能。同样，如果你要给其他人看你编写的代码，注释将帮助别人理解你的代码。Arduino 的灵魂即它的开放性，是分享代码和原理图。因此我希望当你用 Arduino 开始做你自己的项目时，也要将它与其他人分享。还有另外一种写注释的方式，它是一个说明块，用/*和*/括起来，例如以下文字：

```
/* All of the text within  
the slash and the asterisks  
is a comment and will be  
ignored by the compiler */
```

IDE 将自动把注释文字的颜色转化成灰色。程序接下来的一行是这样的：

```
int ledPin = 10;
```

这就是所谓的变量，变量是用来存储数据的。在上面的例子里定义了一个变量，类型是 int 或者说整型。整型表示一个数，范围在-32768 到 32767 之间，接下来指定了这个整型数的名字是 ledPin，并且给它赋了一个值 10。（可以不叫它 ledPin，你可以叫它任何你想叫的东西，但是希望你的变量名是有一定意义的，叫它 ledPin 说明这个变量表示将要去与 LED 相连的 Arduino 的引脚。）在这里，使用数字引脚 10。在这个声明的结尾是一个分号，它告诉编译器这个声明到此结束。

虽然可以给变量设置任何名字，但在 C 语言中，变量名必须以一个字母开头，之后可以包含字母、数字和下画线。注意 C 语言认为大小写字母是不同的。最后，不能使用 C 语言的任何保留字作为变量名，如 main、while、switch 等。保留字是常量、变量和函数名，这些都被定义为 Arduino 语言的一部分。为了避免使用保留字作为变量名，所有保留字在程序中都显示为红色。

想象变量是一个小盒子，在那里存储着东西，在这个程序中，变量在内存中开辟了一个小空间用来存储一个整数，以上的定义表示在变量开辟的存储空间内存了一个数字 10。

最后，变量之所以叫变量是因为可以改变它的值。稍后，将对变量进行数学计算以使程序能够做更复杂的事情。

接下来是 `setup()` 函数：

```
void setup() {
    pinMode(ledPin, OUTPUT);
}
```

Arduino 程序必须包含 `setup()` 和 `loop()` 两个函数，否则它将不能工作。`setup` 函数只在程序的开头运行一次。在这个函数里可以在主循环开始前为程序设定一些通用的规则，如设置引脚形式、设置波特率等。一般情况下，函数是一组集合在一个程序块中的代码。例如，如果生成一个函数来完成一系列复杂的数学计算，它有好几行代码，这些代码可以运行很多次，每次使用这些代码时只要简单地调用函数名即可，而无须每次都把这些代码重新再写一遍。当你开始动手做自己的项目时，你将对函数有更深入的了解。在本程序里，`setup()` 函数只有一行声明。函数从以下形式开始：

```
void setup()
```

它告诉编译器函数叫 `setup`，它不返回数据（`void`），并且不传递参数给它（空括号），如果函数返回整型值，并且需要给它传递一个整型数（如让函数处理）作为参数，它可以写成如下形式：

```
int myFunc (int x, int y)
```

在这里，所生成的函数（或一段代码）叫做 `myFunc`。需要给这个函数传递两个整数作为参数，叫做 `x` 和 `y`，如果函数运行完成，它将在程序调用函数处返回一个整数值。

函数中的所有代码在两个花括号之间，“{”符号在代码块的开头，“}”符号在代码块的结尾。这两个符号之间的任何东西都是属于这个函数的。（我将在后面进一步详细地说明函数，所以现在请不必着急。）

在这个程序里有两个函数，一个函数叫做 `setup`，它的目的是在主要的 `loop` 函数运行之前为程序做必要的设置。

```
void setup() {
    pinMode(ledPin, OUTPUT);
}
```

`setup` 函数内只有一条语句，那就是 `pinMode` 函数，这个函数告诉 Arduino 设置引脚的

模式为输出模式，而不是输入模式。在函数后面的括号内，设定了引脚号和模式（OUTPUT 或 INPUT），引脚号是 `ledPin`，它在这之前已经被设置为数值 10。因此，这条语句只是简单地告诉 Arduino 数字引脚 10 被设置为 OUTPUT 模式。因为 `setup()` 函数只运行一次，现在程序移动到主函数 `loop`：

```
void loop() {
    digitalWrite(ledPin, HIGH);
    delay(1000);
    digitalWrite(ledPin, LOW);
    delay(1000);
}
```

`loop()` 函数是主要的过程函数，只要 Arduino 打开就一直运行。每一条 `loop()` 函数（在花括号内的代码）中的代码都要执行，并按顺序逐个执行，直到函数的最后。然后 `loop` 函数再次开始，从函数顶部开始运行，一直这样循环下去，直到关闭 Arduino 或者按下重启按钮。

在这个项目中，希望 LED 灯亮，保持 1 秒，然后关闭，保持 1 秒，并且重复以上动作。因为你希望反复地做以上动作，所以告诉 Arduino 要这样做的命令设置在 `loop()` 函数内。函数内第一个语句是：

```
digitalwrite (ledPin, HIGH);
```

在这个语句中，写一个 HIGH 或 LOW 值到引脚（在这个例子里，引脚是数字引脚 10），设置一个 HIGH 到引脚中，将输出一个 5V 电压到那个引脚，当设置引脚为 LOW 时，这个引脚变成 0V，或者地，因此上面的声明表示输出一个 5V 电压到引脚 10，这就点亮了 LED。之后的代码是：

```
delay (1000);
```

这条语句只是告诉 Arduino 在执行下一条语句之前等待 1000 毫秒（1000 毫秒是 1 秒）。下一条语句是：

```
digitalwrite (ledPin, LOW);
```

该语句将关闭数字引脚 10 的电源，因此会熄灭 LED。之后是另外一个延时 1000 毫秒的语

句，然后函数结束。然而，因为这个函数是主 `loop()` 函数，所以这个函数将重新从头开始执行。

再一步一步看以上程序，你会发现程序是非常简单的。

```
//项目1 —— LED 闪灯器
int ledPin = 10;
void setup() {
    pinMode(ledPin, OUTPUT);
}
void loop() {
    digitalWrite(ledPin, HIGH);
    delay(1000);
    digitalWrite(ledPin, LOW);
    delay(1000);
}
```

从指定一个名为 `ledPin` 的变量开始，向这个变量赋值 10。之后，执行 `setup()` 函数，此处设置数字引脚 10 为输出模式。在主程序循环里设置数字引脚 10 为 HIGH，输出为 5V，之后等待 1 秒，关闭数字引脚 10 的 5V 电源，等待 1 秒，之后，`loop` 重新开始执行。只要 Arduino 上电，LED 将持续地交替开关。

现在你已经知道代码是如何工作的了，你可以通过改变代码去打开 LED 并保持一段不同的时间，关闭 LED 并保持一段不同的时间。例如，想要持续打开 2 秒，之后关闭 0.5 秒，可以按照下面这样做：

```
void loop() {
    digitalWrite(ledPin, HIGH);
    delay(2000);
    digitalWrite(ledPin, LOW);
    delay(500);
}
```

如果想要 LED 保持关闭 5 秒，然后短暂地闪烁一下（250 毫秒），如同汽车报警器上的 LED 指示灯那样，可以这样做：

```
void loop() {
    digitalWrite(ledPin, HIGH);
    delay(250);
```

```
    digitalWrite(ledPin, LOW);  
    delay(5000);  
}
```

如果想让 LED 快速开关闪烁，尝试以下方式：

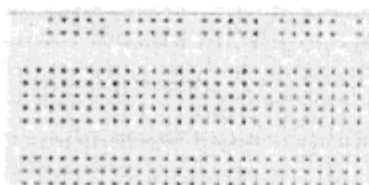
```
void loop() {  
    digitalWrite(ledPin, HIGH);  
    delay(50);  
    digitalWrite(ledPin, LOW);  
    delay(50);  
}
```

通过改变 LED 开和关的时间，可以产生任何你希望看到的效果（当然是在单个 LED 开和关范围内的效果）。在做一些更有意思的东西之前，让我们看一下硬件，看看硬件是如何工作的。

硬件回顾

项目 1 中用到的硬件有：

面包板



5mm LED



100Ω电阻*



跳线



*或其他适合你的 LED 的数值。

面包板是一个可重复使用的非焊接元件，用于制作一个电子线路原型或者线路设计试验。这个板在一个栅格中有一系列的孔，在板子背面，这些孔通过两条导电金属条相连，

这些金属条通常如图 2-2 所示那样排列。

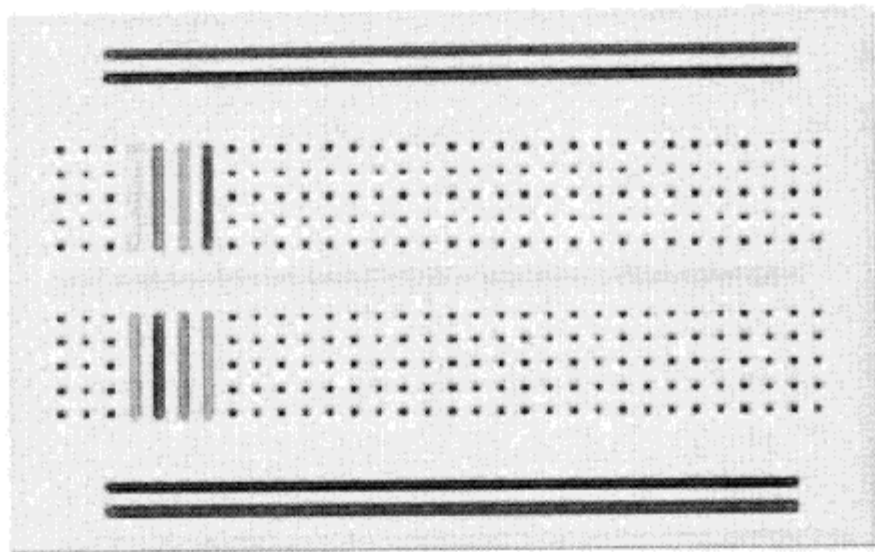


图 2-2 金属条在面包板上如何放置

有些金属条沿着顶部和底部平行贯穿板子，可以用做电源线和地线。板子上中间的元件很方便地连接 5V（或其他所使用的电压）电源和地。有些面包板有红色和黑色线平行贯穿板子，用颜色表明这些孔哪个是电源（红）、哪个是地（黑）。在更大的面包板上电源金属条有时是有断点的，通过断开红线来说明，这使得给板子上不同部分提供不同的电压成为可能。如果你只使用一种电压，可使用一小段跳线短接这些间隔使得同样的电压沿着电源金属条供应。

中间这部分格栅与电源、地金属条形成 90° 角。在中间部分有一个断点。你可以穿过这些断点放置一个集成电路，使得芯片上的每一个引脚进入不同的孔组，也就是与不同的金属条相连（见图 2-3）。

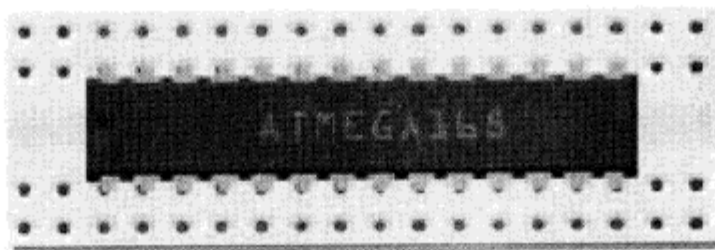


图 2-3 一块集成电路沿着中间断点插入面包板

下一个要说的元件是电阻。电阻会对电流产生一定的阻力，引起它两端电压的下降。可以将电阻想象成一个水管，它比连接它的管子细一点，当水（电流）进入电阻，因管子变细，水流（电流）虽然从另一端出来但是水流减少了。所以可以使用电阻给其他元件减压或减流。

电阻的单位是 Ω ，这是一个大写的 Omega 字母，在这个例子里，数字引脚 10 输出 5V（根据 Atmega 数据手册）、40mA（毫安）直流电，而你的 LED 需要（根据它的数据手册）2V 电压和 35mA 的电流。因此，如果想以 LED 的最大亮度点亮它，需要一个电阻将电压从 5V 降到 2V，电流从 40mA 减到 35mA。如果想让 LED 暗一些，可以使用更高阻值的电阻。

■ **注意** 千万不要用比实际需要小的电阻，如果你给 LED 输入了太大的电流，不但会造成 LED 的永久性损坏，也可能损坏电路中的其他元件。

计算你所需要的电阻阻值的公式是：

$$R = (V_S - V_L) / I$$

这里 V_S 是电源电压， V_L 是 LED 需要的电压， I 是 LED 的电流。该例中 LED 的电压是 2V，电流是 35mA，其所连接到的 Arduino 的数字引脚输出电压是 5V，所以需要的电阻值是：

$$R = (5 - 2) / 0.035$$

算出来的阻值是 87.71 Ω 。

电阻来自一个标准系列，与之最接近的电阻值是 100 Ω 。需要选择标准阻值的电阻，并且要大于需要的阻值。如果你选择小的阻值，将产生太大的电流，导致损坏其他元件。

那么怎样找到 100 Ω 的电阻呢？一个电阻太小了，所以不能写下容易认出来的标签，所以电阻使用色环代码表示阻值的大小。沿着电阻可以发现一个典型的 4 色环，使用表 2-1 列出的色环代码可找到对应的电阻阻值。同样，可以为指定阻值的电阻找到色环代码。

表 2-1 电阻色环代码

颜 色	第 1 个色环	第 2 个色环	第 3 个色环（幂）	第 4 个色环（误差）
黑色	0	0	$\times 10^0$	
棕色	1	1	$\times 10^1$	$\pm 1\%$
红色	2	2	$\times 10^2$	$\pm 2\%$
橘黄色	3	3	$\times 10^3$	
黄色	4	4	$\times 10^4$	
绿色	5	5	$\times 10^5$	$\pm 0.5\%$
蓝色	6	6	$\times 10^6$	$\pm 0.25\%$
紫色	7	7	$\times 10^7$	$\pm 0.1\%$

续表

颜 色	第 1 个色环	第 2 个色环	第 3 个色环 (幂)	第 4 个色环 (误差)
灰色	8	8	$\times 10^8$	$\pm 0.05\%$
白色	9	9	$\times 10^9$	
金色			$\times 10^{-1}$	$\pm 5\%$
银色			$\times 10^{-2}$	$\pm 10\%$
无				$\pm 20\%$

根据这个表格, 对于 100Ω 电阻, 第 1 个色环是 1, 用灰色表示, 接下来一个环是 0, 用黑色表示, 之后需要的幂是 10^1 (换句话说就是 1 个 0), 因此第 3 个环用棕色表示, 最后一个环是电阻的误差, 如果电阻有一个金色的环, 它的误差是 $\pm 5\%$, 这意味着实际的电阻值在 95Ω 到 105Ω 之间。因此如果有一个 LED 需要 2V、35mA, 那么限流电阻用一个灰色的、黑色的、棕色的环组合表示。

$10k\Omega$ (10 千欧) 的电阻需要一个灰色、黑色、橘色的色环组合 (1, 0, +3 个 0)。750k Ω 电阻的色环是蓝色、空白和黄色的组合。

同样地, 如果想要知道一个电阻的阻值, 需要反过来查表。例如图 2-4 中的电阻, 想要知道它的阻值, 然后将它存放在标注明晰的电阻盒里, 通过查表获知它的阻值应是 220Ω 。

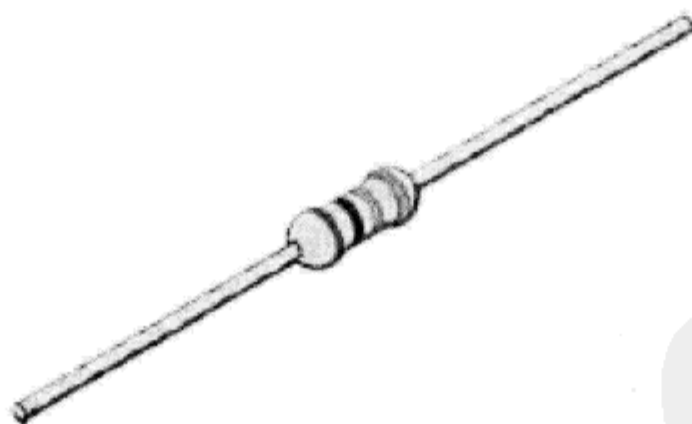


图 2-4 有 5% 误差的 $10k\Omega$ 电阻

现在, 你知道了色环的意思, 那么请为所购买的 LED 选择正确的电阻以完成这个项目。

最后要谈的元件 (不是跳线, 你可以自己弄明白它们是如何工作的) 是 LED, 这是标准的发光二极管。二极管是一种器件, 只允许电流从一个方向流进。它就像一个水流系统中的阀门, 但是只允许电流从一个方向通过。如果电流试图改变方向从相反的方向通过, 二极管会阻止它这样做。二极管用来防止在电路中意外地将电源和地连接以至于损坏其他元件。

LED 也是一个二极管，但是它还能发光，LED 能发出不同的颜色和亮度的光线，包括光谱中紫外线和红外线部分的光线（像电视遥控器上的 LED）。

如果仔细观察 LED，你会注意到两件事：LED 引脚的长度不同，LED 外壳的一边有一个小平面，而不是圆柱形的（见图 2-5）。有一个小提示表明哪个引脚是阳极（正）、哪个引脚是阴极（负）：长腿引脚（阳极）连接电源正极（3.3V），在平面一边的引脚（阴极）接地。

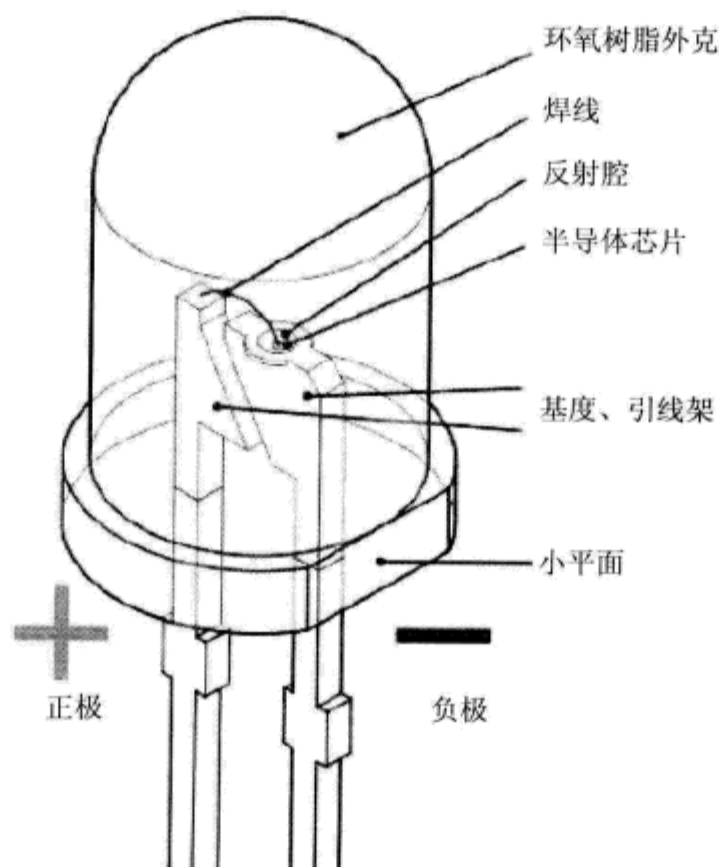


图 2-5 LED 的一部分（维基共享图库供图）

如果 LED 的接线方式出错，通常不会损坏（除非给它接了特别高的电压）。然而，总是给 LED 串联一个电阻是必要的，以确保给 LED 提供正确的电流。如果不这样做，LED 可能受到永久性损坏。

注意，还有一种两色或三色 LED。这种 LED 有几个引脚。RGB LED 有一个红色、一个蓝色和一个绿色 LED，这三色 LED 放在同一个外壳里（因此被称为 RGB）。这种 LED 有 4 个引脚，其中一个是共同的阳极或者共同的阴极（常见的全三色 LED），其他引脚是独立的 LED 的阴极或阳极。通过调整 RGB LED 的 R、G、B 的亮度值，你能得到任何你想要的颜色（你可以通过三个独立的红、绿、蓝 LED 获得相同的效果）。

现在你知道了各元件的功能及项目中代码是如何工作的，让我们尝试做一些更有趣的东西。

项目2——S.O.S 莫尔斯码信号源

在本项目中你将重复使用项目1中建立起来的电路（所以不需要再回顾硬件了），但是你将使用不同的代码使LED发出S.O.S信号。这是国际莫尔斯码求救信号。莫尔斯码是一种字符编码，它使用开关模式传递字母和数字。它非常适合数字系统，因为你可以通过开关LED来拼出一个词或一系列的字母。在本例中S.O.S模式是三个点（短闪烁），紧接着三个横杠（长闪烁），然后再接三个点。

为了使LED闪烁以这个模式发出S.O.S信号，使用的代码如清单2-2所示。

清单2-2 项目2的代码

```
//LED 连接到数字引脚10
int ledPin = 10;

//当程序开始时只运行一次
void setup()
{
    //设置数字引脚模式为输出
    pinMode(ledPin, OUTPUT);
}

//一遍一遍地运行以下代码
void loop()
{
    //3个点
    for (int x=0; x<3; x++) {
        digitalWrite(ledPin, HIGH);    //设置LED为开
        delay(150);                    //延时150毫秒
        digitalWrite(ledPin, LOW);     //设置LED为关
        delay(100);                    //延时100毫秒
    }

    //100毫秒延时产生字母之间的间隔
    delay(100);
    //3个横杠
    for (int x=0; x<3; x++) {
        digitalWrite(ledPin, HIGH);    //设置LED为开
        delay(400);                    //延时400毫秒
    }
}
```

```

        digitalWrite(ledPin, LOW);        //设置 LED 为关
        delay(100);                        //延时 100 毫秒
    }

    //100 毫秒延时产生字母之间的间隔
    delay(100);

    //又是 3 个点
    for (int x=0; x<3; x++) {
        digitalWrite(ledPin, HIGH);        //设置 LED 为开
        delay(150);                        //延时 150 毫秒
        digitalWrite(ledPin, LOW);        //设置 LED 为关
        delay(100);                        //延时 100 毫秒
    }

    //在重复 S.O.S 信号前等待 5 秒
    delay(5000);
}

```

创建一个新的架构，并且输入清单 2-2 中的程序，确认代码中没有错误，把代码上传到 Arduino，如果一切顺利，你将看到 LED 闪烁出莫尔斯码 S.O.S 信号，等待 5 秒，它会重复闪烁。

如果给 Arduino 装备一个电池，用 Arduino 控制一个非常亮的灯，并把它们全部装在一个防水的手提盒子里，就可用它发出 S.O.S 信号。这种信号装置可用在船上或在登山时使用。

让我们研究一下代码是如何工作的。

代码回顾

代码的第一部分与上一个项目完全一样。在这里初始化一个变量，并且设置数字引脚 10 的模式为输出模式。在主要代码 loop 里，你可以看到与上一个项目中类似的语句用于去控制 LED 的开和关，并保持一段时间。然而，这次语句中包含了三个独立的代码段。

第一个代码段是输出三个点：

```

for (int x=0; x<3; x++) {
    digitalWrite(ledPin, HIGH);
    delay(150);
}

```

```
    digitalWrite(ledPin, LOW);  
    delay(100);  
}
```

你可以看到 LED 开 150 毫秒之后关闭 100 毫秒。这些语句是括在一对花括号内的，因此，是一段独立的代码段。但是，当程序运行时，可以看到，灯闪了 3 次而不是只闪 1 次。

产生这样的结果是因为使用了 for 循环：

```
for (int x=0; x<3; x++) {
```

这个语句使代码段中的代码执行 3 次。这里需要传递 3 个参数给 for 循环，它们是初始化变量、条件和增量。初始化变量在 for 循环的最开始，并且只执行一次。每一次循环都要检测条件，如果条件为真，执行语句块且对变量进行增加，之后，再次判断条件，当条件为假时，循环终止。

所以，首先需要初始化一个变量，作为循环开始的计数。在这个例子里，设置变量为 x，并设置它的数值为 0：

```
int x=0;
```

之后，设置一个条件来决定在循环中的代码要循环多少次：

```
x<3;
```

在这里，如果 x 小于 (<) 3，代码将循环。不管你设置的条件是什么，for 循环中的代码总是先执行一次。

<叫做比较操作符。比较操作符用在代码中做判断，用于比较两个值。程序中可能用到的比较操作符有：

- == (等于)
- != (不等于)
- < (小于)
- > (大于)
- <= (小于等于)

- \geq (大于等于)

在代码中, 比较 x 和数值 3, 判断 x 是否小于 3, 如果 x 小于 3, 代码段中的代码要再重复执行一次。

最后的语句是:

```
x++
```

表示把 x 的值增加 1, 也可写为 $x=x+1$, 这意味着把 x 的值换成 $x+1$, 注意 for 循环中最后的语句不需要加分号。

可以用符号 +、-、*、/ (加、减、乘、除) 做简单的数学计算。例如:

- $1+1=2$
- $3-2=1$
- $2*4=8$
- $8/2=4$

将 for 循环初始化 x 的值设为 0, 运行代码块中的代码 (花括号内), 之后, 变量递增 (在这个例子里 x 加 1)。在代码块的最后检查是否符合条件, 条件是 x 小于 3。如果条件符合, 则重复执行。

现在你知道 for 循环是如何工作的了, 可以看到在代码里有 3 个 for 循环: 第一个循环 3 次, 输出 3 个点; 第二个循环 3 次输出 3 个横杠; 最后一个循环 3 次, 再一次输出 3 个点。

必须要注意, x 是一个本地变量, 这意味着它只能在它自己的代码段内起作用。除非在 `setup()` 函数之前初始化它, 在这种情况下, 它是一个全局变量, 可以被整个程序使用。这里如果试图在 for 循环外使用 x , 将发生错误。

在每一个 for 循环之间是一小段延时, 使 S.O.S 字母之间产生一个可见的暂停。最后, 在主程序 loop 重新执行一遍之前, 代码等候 5 秒。

接下来让我们使用多个 LED 创建一个项目。

项目 3——交通信号灯

现在做一套交通信号灯，信号灯要从绿灯亮变成黄灯亮，再变成红灯亮，然后再重新开始。时间间隔遵守英国交通信号灯标准。这个项目可用于一个实用的铁路交通灯模型或孩子的玩具城市。如果你不是英国人，你可以改变 LED 的颜色，使它们像你自己国家的交通信号灯那样工作。但是你首先要按照我们规定的方式做这个项目，当你知道项目是如何工作的之后，你再按照自己的想法去修改它。

需要的元件

面包板



红色漫反射 LED



黄色漫反射 LED



绿色漫反射 LED



3 个 150Ω 电阻*



跳线若干



*或者其他与你所用的 LED 适合的阻值。

把元件连起来

像图 2-6 那样连接电路，连接 3 个 LED，将 LED 的正极通过 150Ω 电阻（或者其他合适的阻值）连接到数字引脚 8、9 和 10。

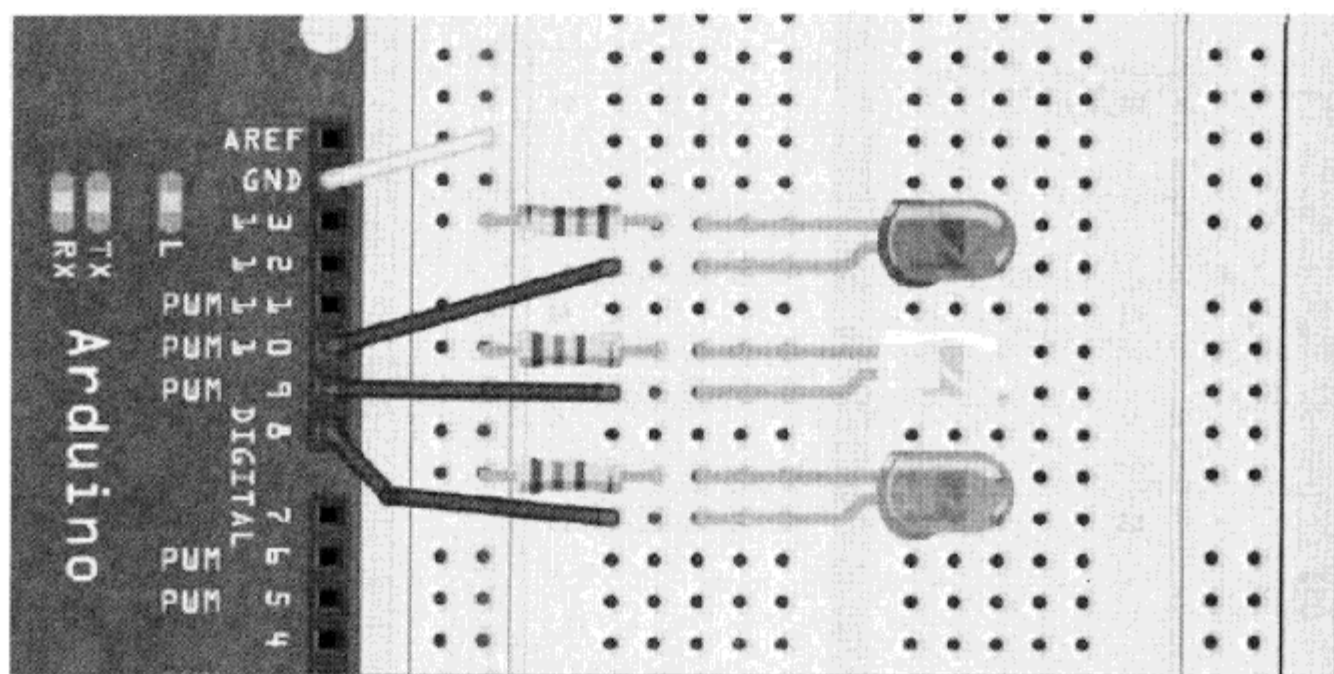


图 2-6 项目 3——交通信号灯电路图（参见彩色版插图）

使用一根跳线将 Arduino 的地端连接到面包板底部的地线上。使用一根地线将每一个 LED 的阴极引脚通过一个电阻连到公共地上。这次把电阻连接到 LED 的阴极上（对于这个简单的电路，电阻连在阴极上还是阳极上都没有关系）。

输入代码

输入如清单 2-3 所示的代码并检查，如无错误，把代码上传到你的 Arduino 中。LED 现在模拟英国交通信号灯系统的 4 个状态，如图 2-7 所示。如果你已经学习了项目 1 和项目 2，那么项目 3 中的代码和硬件就不言自明了。请你自己检查代码并弄清楚它是如何工作的。

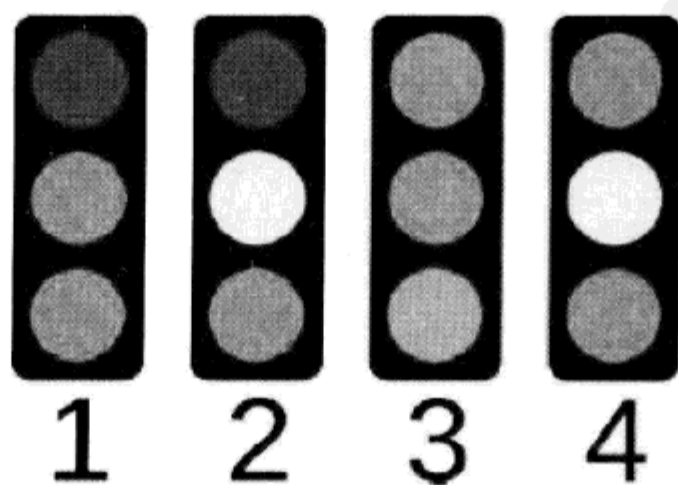


图 2-7 英国交通信号灯的四个状态（维基图库 Alex43223 供图）

清单 2-3 项目 3 的代码

```

//项目 3——交通信号灯
int ledDelay = 10000; //变灯之间的延时
int redPin = 10;
int yellowPin = 9;
int greenPin = 8;

void setup() {
    pinMode(redPin, OUTPUT);
    pinMode(yellowPin, OUTPUT);
    pinMode(greenPin, OUTPUT);
}

void loop() {

    digitalWrite(redPin, HIGH); //红灯亮
    delay(ledDelay); //等待 5 秒

    digitalWrite(yellowPin, HIGH); //黄灯亮
    delay(2000); //等待 2 秒
    digitalWrite(greenPin, HIGH); //绿灯亮
    digitalWrite(redPin, LOW); //红灯灭
    digitalWrite(yellowPin, LOW); //黄灯灭
    delay(ledDelay); //等待 ledDelay 毫秒数

    digitalWrite(yellowPin, HIGH); //黄灯亮
    digitalWrite(greenPin, LOW); //绿灯灭
    delay(2000); //等待 2 秒
    digitalWrite(yellowPin, LOW); //黄灯灭
    //重新开始循环
}

```

项目 4——互动交通灯

这次将要对之前的项目做一些拓展，增加一套行人灯和一个行人请求通过马路的按钮。当按钮被按下时，Arduino 将有反应，它改变交通灯的状态，使汽车停下来，允许行人安全通过。

这是你第一次与 Arduino 互动。当 Arduino 处于等待状态时，改变按钮的状态，使它产生相应的动作。在这个项目中你将学习如何在代码里创建你自己的函数。

从现在开始，在需要的元件列表里我不再列出面包板和跳线，但是记住，项目还是需要这些基本元件的。

需要的元件

2 个红色漫射 LED



黄色漫射 LED



2 个绿色漫射 LED



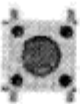
150Ω电阻



4 个电阻



按钮



给项目中使用的 LED 选择限流电阻。150Ω电阻是给按钮用的，它叫做下拉电阻（将在后面解释这个概念），有时供应商称按钮为微动开关，微动开关在面包板上使用是十分理想的。

把元件连接起来

像图 2-8 那样连接你的电路。给 Arduino 上电前再检查一遍你的连线。记住，当你连接电路时，要把 Arduino 电源断开。

输入代码

输入清单 2-4 中的代码，确定无语法错误并上传代码。运行这个程序，开始时汽车通行灯为绿色允许汽车通过，行人通行灯为红色。

当你按下按钮时，程序判断距最后一次行人通过灯的状态发生变化是否已经超过 5 秒（允许交通运行）。如果是这样，把代码转移到你建立的叫做 `changeLights()` 的函数中执行，在这个函数中汽车通行灯从绿色变为黄色再变为红色，行人通行灯变绿。经过设置在变量 `crossTime` 中的时间之后（这段时间应当允许行人安全通过），绿色行人通行灯闪烁，警告行人抓紧时间通过，因为灯要变为红色了。之后行人通行灯变为红色，机动车通行灯从红色变为黄色再变为绿色。机动车继续通行。

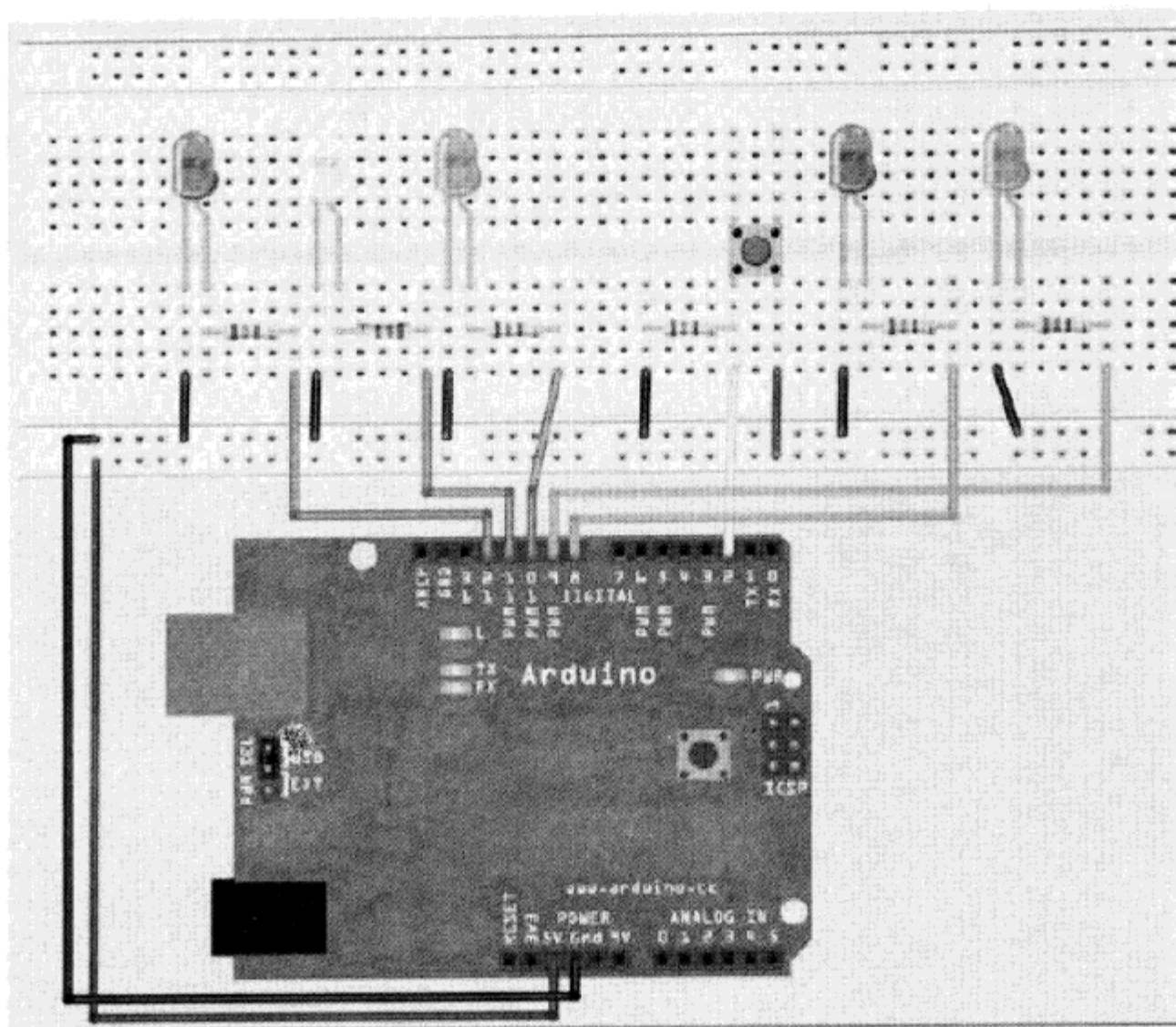


图 2-8 项目 4——有行人通过灯和请求按钮的交通信号灯系统电路图（参见彩色版插图）

清单 2-4 项目 4 代码

//项目 4 —— 互动交通信号灯

```
int carRed = 12; //指定机动车灯
int carYellow = 11;
int carGreen = 10;
int pedRed = 9; //指定行人灯
int pedGreen = 8;
int button = 2; //按钮引脚
int crossTime = 5000; //允许行人通过的时间
unsigned long changeTime; //按钮按下的时间

void setup() {
    pinMode(carRed, OUTPUT);
    pinMode(carYellow, OUTPUT);
```



```

    pinMode(carGreen, OUTPUT);
    pinMode(pedRed, OUTPUT);
    pinMode(pedGreen, OUTPUT);
    pinMode(button, INPUT); //按钮引脚模式
    //变成绿色灯
    digitalWrite(carGreen, HIGH);
    digitalWrite(pedRed, HIGH);
}

void loop() {
    int state = digitalRead(button);
    /* 检查按钮是否按下, 并且是否距上次按下已经超过 5 秒 */
    if (state == HIGH && (millis() - changeTime) > 5000) {
        //调用变灯函数
        changeLights();
    }
}

void changeLights() {
    digitalWrite(carGreen, LOW); //绿灯灭
    digitalWrite(carYellow, HIGH); //黄灯亮
    delay(2000); //等待 2 秒

    digitalWrite(carYellow, LOW); //黄灯灭
    digitalWrite(carRed, HIGH); //红灯亮
    delay(1000); //为安全再等待 1 秒

    digitalWrite(pedRed, LOW); //行人红灯灭
    digitalWrite(pedGreen, HIGH); //行人绿灯亮
    delay(crossTime); //等待一个通过时间

    //闪烁行人绿灯
    for (int x=0; x<10; x++) {
        digitalWrite(pedGreen, HIGH);
        delay(250);
        digitalWrite(pedGreen, LOW);
        delay(250);
    }
    //行人红灯亮
    digitalWrite(pedRed, HIGH);
    delay(500);

    digitalWrite(carYellow, HIGH); //黄灯亮

```

```
digitalWrite(carRed, LOW); //红灯灭
delay(1000);
digitalWrite(carGreen, HIGH);
digitalWrite(carYellow, LOW); //黄灯灭

//记录自上一次变灯的时间
changeTime = millis();
//返回到主循环当中
```

代码回顾

通过前面项目的学习，你应该能够理解这个项目中的大部分代码。我只解释一下新的关键词和概念：

```
unsigned long changeTime;
```

这是一个新的变量数据类型。之前，你曾创建整型数，它可以存储一个-32 768 到 32 767 之间的整数。这次你创建了一个数据类型为 `long` 的数，它可以存储一个从-2 147 483 648 到 2 147 483 647 之间的整数。然而，对于指定的 `unsigned long` 数据类型，它不存储负数，所以它的存储范围是从 0 到 4 294 967 295。如果你使用一个整型变量存储信号灯状态变化的间隔时间，它能存储的最大值为 32 秒，这是整型变量能存储数据的上限。

行人通过马路的请求不可能每次间隔都在 32 秒内，你也不希望变量存储超过数据类型范围的数据，因为这会使变量溢出而造成系统崩溃。所以你使用一个 `unsigned long` 数据类型的变量来存储行人通过按钮两次被按下之间最大的时间长度：

- $4294967295 * 1 \text{ 毫秒} = 4294967 \text{ 秒}$
- $4294967 \text{ 秒} = 71582 \text{ 分钟}$
- $71582 \text{ 分钟} = 1193 \text{ 小时}$
- $1193 \text{ 小时} = 49 \text{ 天}$

在 49 天内按钮不可避免地会被行人按下一次，所以你使用这个数据类型没有问题。

那么，为什么只有一种数据类型可以存储大数？因为变量的内容要占用一定的空间。数越大需要越多的空间存储变量。在 PC 或笔记本电脑上，这没有什么问题，但是对于一个小的微控制器，像 Arduino 的 Atmega 32，使用能够满足程序需要的最小的数据类型是必

要的。

表 2-2 列出了在程序中可能用到的变量数据类型。

表 2-2 数据类型

数据类型	RAM	范围
void keyword	N/A	N/A
boolean	1byte	0 到 1 (True 或 False)
byte	1byte	0 到 255
char	1byte	-128 到 127
unsigned char	1byte	0 到 255
int	2byte	-32 768 到 32 767
unsigned int	2byte	0 到 65 535
word	2byte	0 到 65 535
long	4byte	-2 147 483 648 到 2 147 483 647
unsigned long	4byte	0 到 4 294 967 295
float	4byte	-3.4028235E38 到 3.4028235E38
double	4byte	-3.4028235E38 到 3.4028235E38
string	1byte+x	字符数组
array	1byte+x	变量集合

每一种数据类型使用确定的内存。有些变量只使用 1 Byte 的存储空间，而其他的变量可能用到 4 Byte 或更多（不用着急知道什么是 1 Byte，我将在后面讨论）。注意，你不可以复制一个数字从一种数据类型到另一种数据类型。如果 x 是一个整型数，y 是一个字符串，程序语句 x=y 不会工作，因为 x 和 y 是两种不同数据类型的数据。

Atmega 168 有 1kB（1000 Byte），Atmega 328 有 2kB（2000 Byte）的内存，这不是一个很大的存储空间。在大的程序里会有很多变量，如果在程序里不优先使用正确的数据类型，容易超出最大存储空间。如果使用整型（这种数据类型使用 2 Byte 存储空间，可存储的数字最大为 32 767）存储引脚号码，Arduino 的引脚号最大才到 13（Arduino Mega 到 54），你使用了比实际需要更多的存储空间。你完全可以使用 Byte 类型存储引脚号码到内存中，它的存储范围在 0 到 255 之间，足够用来存储 I/O 引脚号码。

接下来给出以下代码：

```
pinMode(button, INPUT);
```

本语句告诉 Arduino 使用数字引脚 2（button=2）作为 Input 模式，你要使用数字引脚 2 来

接收按钮的按下信号，因此它的模式需要设置为输入模式。

在程序主循环里用如下语句检查引脚 2 的状态：

```
int state = digitalRead(button);
```

初始化一个整数（是的，这有些浪费，你应该使用一个布尔型变量）`state`，并且设置 `state` 的数值为数字引脚 2 的状态。`digitalRead` 语句是读括号内的引脚状态，并且返回给已经指定的整数，因此可以通过检查 `state` 中的数值来判断按钮是否被按下：

```
if (state == HIGH && (millis() - changeTime) > 5000) {
    //调用函数变换灯的状态
    changeLights();
}
```

`if` 语句是程序控制结构的一种，它的目的是检查一个确定的条件是否达到。如果达到，执行代码块中的代码。例如，如果你想实现当变量 `x` 大于数值 500 时点亮一个 LED，可以用如下的代码实现：

```
if(x>500){digitalWrite(ledPin, HIGH);
```

当用 `digitalRead` 函数读引脚状态时，引脚状态要么是 HIGH，要么是 LOW，所以在程序中 `if` 语句可写成如下形式：

```
if (state == HIGH && (millis() - changeTime) > 5000)
```

在这里需要做的是检查两个条件是否符合。第一个条件是 `state` 变量为 HIGH。如果按钮被按下，`state` 将是 HIGH，因为已经设置从数字引脚 2 中读取它的数值。第二个条件是 `millis()` 的值减 `changeTime` 大于 5000，然后两个条件进行与运算（使用逻辑与操作符 `&&`）。`millis()` 函数是 Arduino 语言中自有的函数，它返回以毫秒为单位的从 Arduino 开始执行到运行到当前的时间。`changeTime` 变量初始化不存储任何数值，除非在 `changeLights` 函数运行之后设置 `changeTime` 为这个函数结束时的 `millis()` 函数返回的数值。

通过从当前的 `millis()` 数值中减去 `changeTime` 中的数值，你可以判断自从 `changeTime` 最后一次设置以来是否已经过去了至少 5 秒。`millis()-changeTime` 计算放在括号里，保证比较的是 `state` 与本次计算结果，而不是 `millis()` 中的数值。

在 `state==HIGH` 与后面的计算之间有符号 `&&`，`&&` 符号是一个布尔计算符号，这个符号的意思是进行逻辑与运算。要想更清楚地了解逻辑与运算，先看一下全部的布尔运算符：

- `&&`——逻辑与
- `||`——逻辑或
- `!`——逻辑非

这些用于逻辑计算，并且可以检查 if 语句中的各种条件。

`&&` 的意思是当两个操作数都是真时，计算结果为真，因此当 `x` 为 5、`y` 为 10 时下面这个 if 语句内的代码将运行：

```
if (x==5&&y==10) {...
```

`||` 表示两个操作数中只要一个为真，值为真，例如，当 `x` 为 5 或 `y` 为 10 时下面这个 if 语句中的代码将运行：

```
if (x==5||y==10) {...
```

`!` 是非运算符，如果操作数为假，运算后的值为真，因此如果 `x` 是假，下面这个 if 语句内的代码将运行：

```
if (!x) {...
```

可以将操作符放在括号中使用，例如：

```
if (x==5&&(y==10||z==25)) {...
```

在这个例子中，括号中的条件独立处理，当成一个单独的条件对待，并且与第二个条件比较。因此，如果你为这个语句画一个真值表（见表 2-3），就可以看到条件是如何工作的。

表 2-3 条件 `(x==5&&(y==10||z==25))` 的真值表

x	y	z	真/假
4	9	25	假
5	10	24	真
7	10	25	假
5	10	25	真

if 语句内的函数是:

```
changeLights();
```

这是一个函数调用的例子。函数是一段有名称的独立代码。可以给函数传递参数，函数也可以返回数值。但在这个例子里，你没有传递给函数任何参数，也没有让函数返回任何数值。我将在后面更进一步详细地讲解函数的参数传递和返回值的内容。

当 `changeLights()` 被调用时，程序的执行从当前行跳转到函数中，执行函数内的代码，然后返回到程序中调用函数的下一行代码。

在这里，如果满足 if 语句中的条件，程序执行函数中的代码，函数运行完成之后，返回 if 语句中 `changeLights()` 下一行代码继续执行。

函数中的代码仅仅改变交通灯的颜色，从红色变为黄色，变为绿色，之后点亮绿色行人通过灯。在变量 `crossTime` 中设置的时间过后，灯开始闪烁一小段时间以提示行人，允许他通过的时间要结束了。之后，行人通行灯变红，汽车通行灯从红色变为黄色，再变为绿色，这样返回交通灯的正常工作状态。

主程序 `loop` 只是不断地检查行人按钮是否被按下。如果被按下，并且 (`&&`) 距最后一次按下时间大于 5 秒，则再次调用 `changeLights()`。

在这个程序里，把代码放入它自己的函数中除了使代码看得更清楚并解释函数的概念外没有什么好处。只有当一个函数传递了参数，并且/或者返回数值，它的优点才能真正凸显出来。当你再次使用函数时，会看到这一点。

硬件回顾

项目 4 中介绍的新硬件是按钮，或者叫开关。通过电路你可以发现，按钮不是直接连在电源线和输入引脚之间的，在按钮和地之间有一个电阻，这就是下拉电阻。下拉电阻对于保证按钮正常工作是必要的。我要花一点时间解释下拉电阻和上拉电阻。

逻辑状态

逻辑电路是一种只有开、关两种状态的电路，这两种状态用布尔数 1 和 0 表示。电路处于关（或 0）状态时输出端的电压接近 0V。电路处于开状态（或 1）时用高电平表示，输出端接近于电源供电电压。对逻辑电路的简单描述是开关（见图 2-9）。

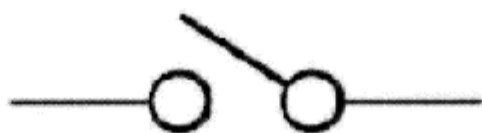


图 2-9 开关的电学符号

当开关处于打开状态时，电路中没有电流通过，在输出端不能测量到电压。当你合上开关时，电流将流过，所以在输出端可以测量到一个电压。因此在逻辑电路中，开关的开状态可以表示为 0，关状态可以表示为 1。

在逻辑电路里，若想要描述 1 状态所需要的电压是 5V，当输出电路状态为 1 时，电压尽可能接近 5V，这是非常重要的。同样，当输出状态是 0（或关）时，电压尽可能接近 0V，这也是非常重要的。如果不能确保状态接近所需要的电压，这部分电路可认为是浮动的（既不是高电平也不是低电平的状态），这种浮动也被叫做电子噪声，噪声在数字电路中可能随机地解释为 1 或者 0。

上拉电阻或下拉电阻可用来保证状态确定为高或低。使用数字电路时不希望电路中的某个点浮动，即随机地被解释成 0 或 1，最好强制电路指向一个希望的状态。

下拉电阻

图 2-10 显示了一个使用下拉电阻的原理图。如果按钮被按下，电流以电阻最小的路径在 5V 端与输入引脚之间流过（100Ω电阻连在输入引脚上，10kΩ电阻接地）。然而，当按钮没有被按下时，输入引脚通过 100kΩ电阻接地。如果没有这个电阻，当按钮没有被按下时，这个引脚将不连接任何东西，因此它的电压将在 0V 和 5V 之间浮动。在这个电路中，当按钮没有被按下时，输入将总是接地的，或者是 0V，当按钮被按下时它将指向 5V 端，也就是说，你能保证引脚不在两个值之间浮动。

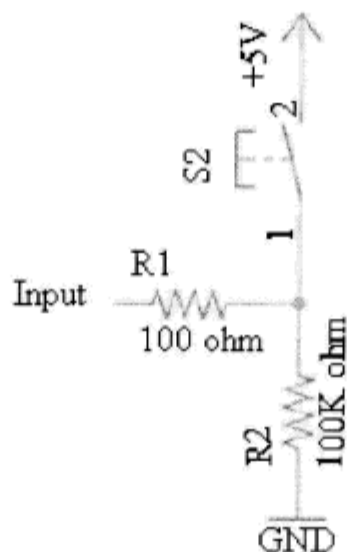


图 2-10 下拉电阻电路

上拉电阻

现在看图 2-11。在这个电路里，交换了下拉电阻和开关的位置。现在电阻变成了上拉电阻。如你所看到的那样，当按钮没有被按下时，输入引脚通过上拉电阻接到 5V 端，所以引脚上总是高电平。当按钮被按下，通过限流电阻的路径引脚接地，所以引脚被拉向地或低电平的状态。如果没有 5V 端和地之间的电阻，电路将被短路，这将损坏电路或电源。正是有了这个电阻，电路不再短路，因为该电阻限制了电流大小。上拉电阻在数字电路中应用得更广泛。

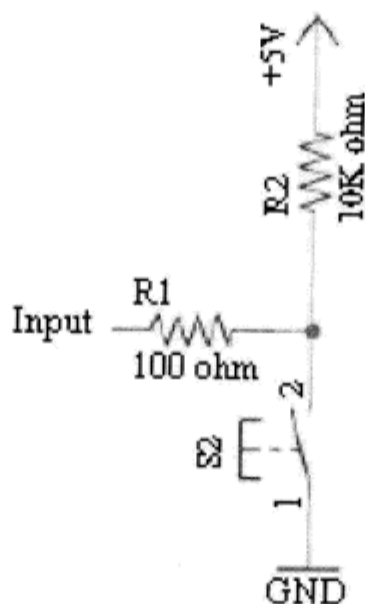


图 2-11 上拉电阻电路

通过使用简单的上拉电阻你可以根据需要确保输入引脚的状态要么是高电平，要么是低电平。

项目 4 中使用下拉电阻确保开关状态能被 Arduino 正确记录。让我们重新回顾一下电路中的下拉电阻（见图 2-12）。

这个电路中包含了一个按钮。按钮的一个引脚直接连接 5V 电压，另外一个引脚直接与数字引脚 2 相连，同时这个引脚通过一个下拉电阻与地相连。这意味着，当按钮没有被按下时，引脚被拉向地，因此读出的是一个 0 或低电平。当按钮被按下时，5V 电压接入引脚，引脚读出的是 1 或者高电平。通过检查输入是高电平还是低电平就可以判断按钮是否被按下。如果不存在电阻，输入引脚就不连接任何东西（悬空），因此电压是浮动的。Arduino 既可能读这个值为高电平也可能读这个值为低电平，这可能导致错误地记录按钮的动作。

上拉电阻在数字电路中经常用来保证输入保持高电平。例如，在本书后面会用到集成

电路移位寄存器 74HC595，它有一个主复位引脚，这个引脚处于低电平时重置芯片。因此，这个引脚必须总是保持为高电平，除非想去复位。一般通过使用上拉电阻保持这个引脚状态为高电平。当想要复位时，可以通过设置一个数字输出为低把它拉向低电平。否则，引脚将保持高电平状态。许多其他的 IC 也有这样的引脚，在大多数时间里保持高电平，拉到低电平仅仅是为了触发各种功能。

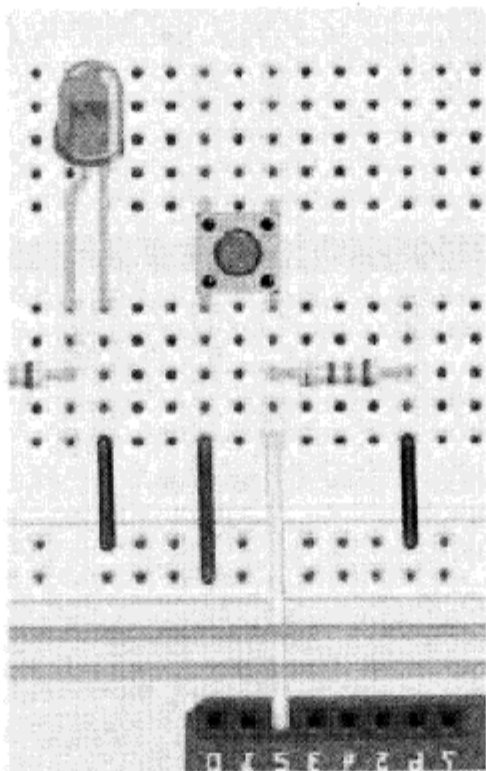


图 2-12 项目 4 中的下拉电阻（参见彩色版插图）

Arduino 的内部上拉电阻

Arduino 内部包含了上拉电阻，使用非常便捷。它连接在引脚上（模拟引脚上也有上拉电阻），阻值为 $20k\Omega$ ，使用时需要通过软件激活。为了激活引脚内部的上拉电阻，首先需要用 `pinMode` 函数设置引脚模式为 `INPUT`，之后用 `digitalWrite` 函数将这个引脚写为 `HIGH`：

```
pinMode(pin, INPUT);  
digitalWrite(pin, HIGH);
```

激活内部上拉电阻后，如果将 `pinMode` 从 `INPUT` 改为 `OUTPUT`，引脚将保持 `HIGH` 状态。反之亦然：当一个输出脚为 `HIGH` 时，转换这个引脚到 `INPUT` 模式，那么内部上拉电阻将激活。

小结

本书开始这4个项目涵盖了许多基本内容。读者现在已经了解基本的读输入和开、关LED的方法。读者通过了解LED和电阻是如何工作的、电阻怎样用来限流及如何根据需要来使用电阻设置引脚为高或低电平等建立了自己的电子学知识。同时也知道了如何选择电阻，并通过观察色环的方式找到电阻所对应的阻值。读者还理解了Arduino程序语言并逐步了解该语言。本章还介绍了一些函数和概念。

第2章所介绍的知识是将来做非常复杂的Arduino项目的基础。在第3章中我们将继续使用LED产生各种效果，并了解大量的函数和概念。这些知识将有助于你学习本书后面的许多高级项目。

本章的主题和概念

- 代码注释的重要性
- 变量和它们的类型
- setup()和loop()函数的目的
- 函数概念及如何创建函数
- 设置一个数字引脚的类型
- 往引脚上写HIGH或LOW值
- 如何产生一个给定毫秒值的延时
- 面包板及其使用方法
- 什么是电阻，阻值如何计算，如何用它来限流
- 如何算出LED需要的电阻的阻值
- 如何通过色环看出电阻的阻值
- LED是什么，以及它是如何工作的
- 如何使用for循环使代码重复执行
- 比较运算符

- 代码中简单的数学计算
- 全局和本地变量的不同
- 上拉电阻、下拉电阻及使用方法
- 如何读取按钮的状态
- 使用 if 语句进行判断
- 在 OUTPUT 和 INPUT 之间改变引脚模式
- millis()函数及如何使用它
- 布尔运算符及如何用它们做逻辑判断



第3章

LED 效果

在第2章中，你学习了基本的输入输出方法、一些基础的电子学知识及一整套编程概念。在本章中，你将继续使用LED，让LED产生一些非常迷人的效果。本章不把重点放在学习更多的电子元件上，而是介绍一些重要的编程概念，如数组、数学函数，以及用于处理本书后面更复杂项目的一些必需的编程技巧。

项目5——LED跑马灯效果

你要使用一串LED（总共10个）产生LED跑马灯效果，就像《霹雳游侠》中汽车基特或《太空堡垒卡拉狄加》中赛昂面板上的效果一样。在这个项目中会介绍数组的概念。

需要的元件

10个5mm红色LED



10个限流电阻



把元件连接起来

首先，把USB电缆拔下来，确保Arduino没有上电。现在将面包板、LED、电阻和导线连成如图3-1中所示的那样。在给Arduino连上电源之前，请仔细检查你的电路。

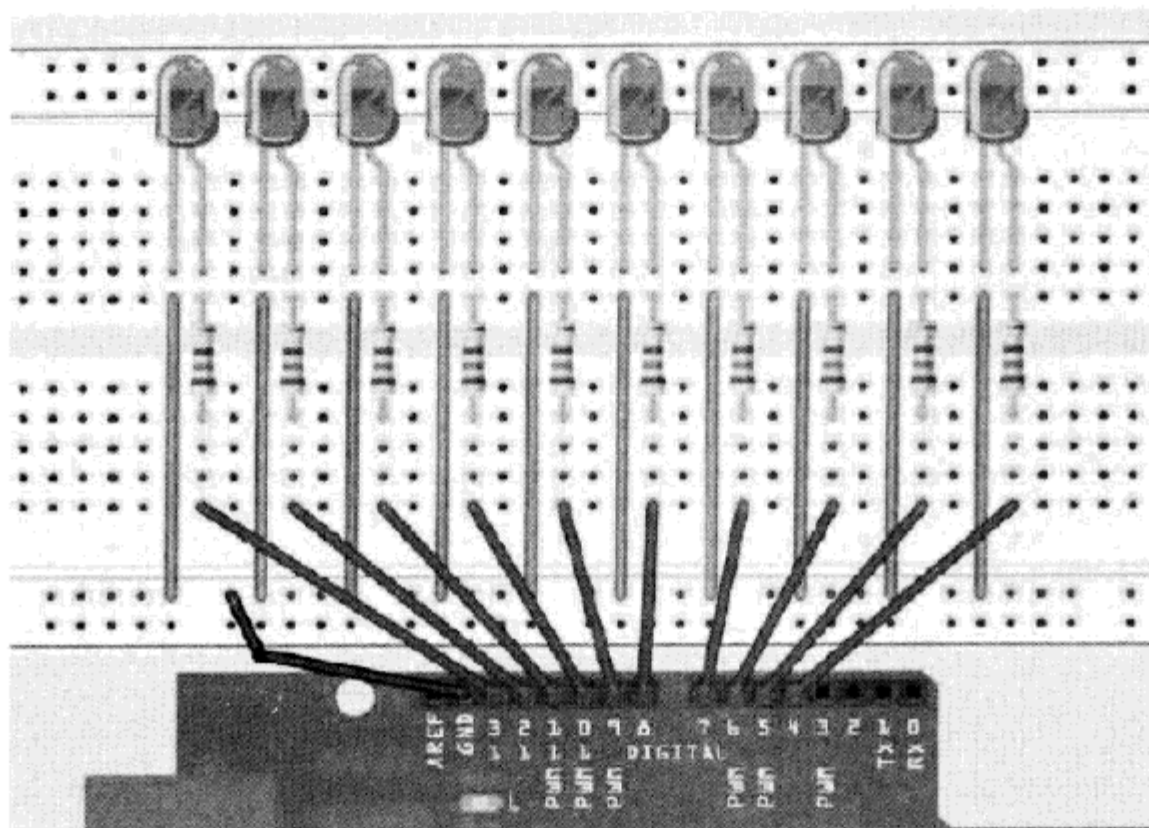


图 3-1 项目 5——LED 跑马灯效果电路图（参见彩色版插图）

打开 Arduino IDE，输入清单 3-1 中的代码。

清单 3-1 项目的 5 代码

```
//项目 5——LED 跑马灯效果
byte ledPin[] = {4, 5, 6, 7, 8, 9, 10, 11, 12, 13}; //为 LED 引脚创建数组
int ledDelay(65); //变化之间的延时
int direction = 1;
int currentLED = 0;
unsigned long changeTime;
void setup() {
    for (int x=0; x<10; x++) { //设置所有引脚为输出模式
        pinMode(ledPin[x], OUTPUT); }
    changeTime = millis();
}

void loop() {
    //如果距最后一次变灯 ledDelay 毫秒已经过去
    if ((millis() - changeTime) > ledDelay) {
        changeLED();
        changeTime = millis();
    }
}
```

```

void changeLED() {
    for (int x=0; x<10; x++) {                //关闭所有 LED
        digitalWrite(ledPin[x], LOW);
    }
    digitalWrite(ledPin[currentLED], HIGH); //点亮当前 LED
    currentLED += direction; //当前值增加 direction 表示的数值
    //如果到达 LED 末端改变 direction 值
    if (currentLED == 9) {direction = -1;}
    if (currentLED == 0) {direction = 1;}
}

```

单击 IDE 顶部的 **Verify/compile** 按钮，验证代码中是否有错误。如果没有错误，单击 **Upload** 按钮。如果操作正确无误，LED 将依次点亮，之后返回到第一个点亮的 LED。

我在这个项目中没有介绍任何新硬件，因此这里不需要硬件回顾。然而，我要在这个项目中介绍数组这个新概念，让我们看一下项目 5 的代码，看看它是如何工作的。

代码回顾

该程序中的第一行为：

```
byte ledPin[] = {4, 5, 6, 7, 8, 9, 10, 11, 12, 13};
```

这是定义一个 **byte** 类型的数组变量。数组是一个变量的集合，可以通过索引号来使用数组中的元素。在你的程序中声明了一个数据类型为 **byte** 的数组，并取名为 **ledPin**。之后用 10 个数值初始化这个数组（数字引脚 4 到 13）。为了获得数组中的一个元素，你只需简单地指出这个元素的索引号。数组是从 0 开始索引的，这意味着数组中第一个元素的索引是 0 而不是 1，因此数组中的 10 个元素的索引号是 0 到 9。在这里元素 3 (**ledPin[2]**) 的值是 6，元素 7 (**ledPin[6]**) 的值是 10。

如果你事先没用数值初始化数组，就必须设定数组的大小。在你的程序中没有明确说明数组大小，因为编译器可以自己计算数组的大小。你已经在给数组初始化过程中指定了这个数组的大小是 10。如果你已经声明了数组，但是没有用数值初始化它，那就需要声明数组的大小。例如，可这样做：

```
byte ledPin[10];
```

载入数据到数组元素中之后，可采用如下方式从数组中获得数值：

```
x = ledPin[5];
```

在这个例子中把索引号为 5（第 6 个元素）的数组元素赋值给 x，这个数值是 8。

返回到程序，你已经在开始处声明和初始化一个数组来存储 10 个数值。这 10 个数值是数字引脚的编号，这是 10 个数字引脚，用来控制 10 个 LED。

在主循环里，要判断距最后一次 LED 的状态改变是否已经过去了一段确定的时间，这个时间以毫秒为单位存储在变量 ledDelay 中。如果这个时间已经超过，程序将控制权交给函数 changeLED()。主程序采用这种判断时间的方式转换到控制函数 changeLED()，而不是使用 delay() 函数，原因是：在需要的情况下允许其他代码在主程序循环中运行（只要代码运行时间比 ledDelay 中的时间短）。

所创建的函数是：

```
void changeLED() {
    for (int x=0; x<10; x++) {                //关闭所有 LED
        digitalWrite(ledPin[x], LOW);
    }
    digitalWrite(ledPin[currentLED], HIGH); //打开当前 LED
    currentLED += direction; //当前值增加 direction 表示的数值
    //如果到达 LED 灯末端，改变 direction 值
    if (currentLED == 9) {direction = -1;}
    if (currentLED == 0) {direction = 1;}
}
```

这个函数做的工作是关闭所有 LED，然后打开当前 LED（这个转换如此之快，以至于你感觉不到关闭这个动作的发生）。当前 LED 编号存储在变量 currentLED 中。

变量 currentLED 与增量常数相加，因为增量常数不是 1 就是 -1，所以 currentLED 要么加 1，要么减 1（currentLED+（-1））。

之后是一个 if 语句，判断是否已经达到了末端的 LED。如果是，将对增量取反。

通过改变 ledDelay 的数值，你可以设定 LED 向前或向后依次点亮的速度。试试不同的数值，看看将发生什么变化。

注意，停止程序运行后，手动改变 ledDelay 的数值，之后上传修改后的程序，看看会有什么不同。当程序运行时能够调整 LED 依次点亮的速度是不是更好？答案是肯定的。我们将在下一个项目中详细说明如何实现在程序运行时改变 LED 点亮的速度。你将学到如何

通过一个电位计与程序互动来调整 LED 点亮的速度。

项目 6——互动 LED 跑马灯效果

将电路完全保留在项目 5 的状态。你只需要给电路加一个电位计，它将在代码运行时改变 LED 灯点亮的速度。

需要的元件

在项目 5 的基础上增加一个 $4.7\text{k}\Omega$ 的旋转电位计。

$4.7\text{k}\Omega$ 旋转电位计



Iain Fergusson 供图

把元件连接起来

首先拔下 USB 线缆，确认 Arduino 没有上电。现在如图 3-2 所示在电路中增加电位计。把电位计左边引脚连接到 Arduino 的 5V 电源上去，中间引脚接到 Arduino 的模拟引脚 2 上，右端引脚接地。

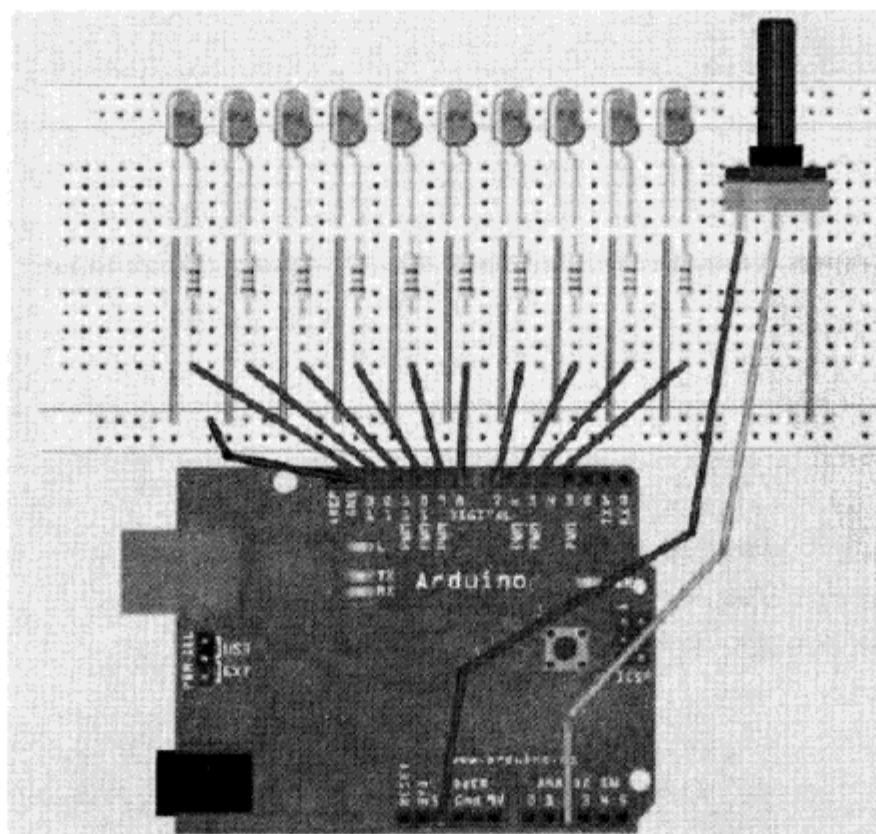


图 3-2 项目 6——互动 LED 跑马灯效果电路图（参见彩色版插图）

输入代码

打开 Arduino IDE，输入清单 3-2 中的代码。

清单 3-2 项目 6 的代码

```
byte ledPin[] = {4, 5, 6, 7, 8, 9, 10, 11, 12, 13};    //为 LED 引脚生成数组
int ledDelay; //变灯之间的延时
int direction = 1;
int currentLED = 0;
unsigned long changeTime;
int potPin = 2;                                     //选择给电位计的输入引脚

void setup() {
  for (int x=0; x<10; x++) {                        //设置所有引脚模式为输出模式
    pinMode(ledPin[x], OUTPUT); }
  changeTime = millis();
}

void loop() {
  ledDelay = analogRead(potPin);                    //从电位计中读值
  //如果自最后一次变灯起已经过去了 ledDelay 毫秒的时间
  if ((millis() - changeTime) > ledDelay) {
    changeLED();
    changeTime = millis();
  }
}

void changeLED() {
  for (int x=0; x<10; x++) { //关闭所有 LED
    digitalWrite(ledPin[x], LOW);
  }
  digitalWrite(ledPin[currentLED], HIGH); //打开当前 LED
  currentLED += direction; //当前 LED 号增加 direction 表示的数值
  //如果到达末端 LED 灯，改变 direction 值
  if (currentLED == 9) {direction = -1;}
  if (currentLED == 0) {direction = 1;}
}
```

检查程序，如果没有错误则上载代码，你将看到 LED 灯依次点亮出现如前面所述的跑马灯效果。通过旋转电位计的旋钮就可以改变 ledDelay 的数值，使 LED 灯依次点亮的速度

变快或变慢。

让我们看一下程序是如何工作的，以及什么是电位计。

代码回顾

这个项目的代码几乎与前一个项目的完全一样，只是在硬件中增加了一个旋转电位计和从电位计中读数值的代码，通过它调整 LED 跑马灯效果的速度。

首先给电位计引脚声明一个变量：

```
int potPin = 2;
```

由于变阻器连接到模拟引脚 2 上，为了从模拟引脚读数值，需要用到读取模拟量的函数。Arduino 有 6 个模拟输入/输出引脚，每个引脚带有一个 10 位模/数转换器（我将在后面讨论这个概念）。这意味着模拟引脚能够读 0 到 5V 之间的电压，用 0 到 1023 之间的整数代表 0V 到 5V 之间的电压。这个转换的结果是每个分度表示 $5V/1024$ 电压，即每个分度是 0.0049V (4.9mV)。

用电位计设置延时使你可以直接利用从引脚读出的数值在 0 到 1023 毫秒(或刚好 1 秒)之间调整延时时间。通过直接读取电位计引脚数值到 ledDelay 变量中实现这个功能。注意，不需要设置一个模拟引脚为输入或输出模式（不像数字引脚需要的那样）：

```
ledDelay = analogRead(potPin);
```

以上功能是在主循环中实现的，因此程序总是在不断地读模拟量和调整时间间隔数值。通过旋转电位计旋钮你可以在 0 到 1023 毫秒之间（或者将近 1 秒）调整间隔时间，因此完全可以控制跑马灯的速度。

下面阐述什么是电位计及它是如何工作的。

硬件回顾

该项目中唯一增加的硬件是 $4.7k\Omega$ 的电位计。

你应当明白电阻是如何工作的。电位计就是一个可调电阻，调节范围从 0 到一个设定值（写在它的外壳上面）。本项目所采用的 $4.7k\Omega$ (4700Ω) 电位计指的是电位计的调节范围是 0 到 4700Ω 。

电位计有 3 个引脚。若只连接两个引脚，电位计可变为一个可变电阻。通过连接 3 个引脚，并为其提供一个电压，它将成为一个分压器。后面的项目将继续介绍在电路中如何使用电位计。电位计一端引脚接地，一端引脚连到 5V 电压，中间引脚接模拟引脚。你可从模拟引脚 2 读取电压值，并且用它改变跑马灯效果的延时时间。

电位计是非常有用的，它提供了一种在 0 到设定的最大值之间调整数值的方法，例如收音机的音量调整或灯的亮度调节，实际上，调节你家里灯光明暗所用的元件也是一种电位计。

练 习

你已经有足够的知识调整代码，请完成以下练习。

练习 1：使 LED 灯带两端变亮，并向对方 LED 运动，直到互相碰到，之后返回到两端。

练习 2：通过点亮 LED 制作一个球弹跳的效果。LED 是垂直排列的。使在底部的 LED 先点亮，然后逐渐向上点亮直到顶部 LED（第 10 个 LED），再返回底部 LED。之后再向上逐渐亮到第 9 个 LED，随之返回底部 LED，最后向上到第 8 个 LED。通过这样做来模拟跳跃起来的球每次反弹的高度距顶部少一个灯的效果。

项目 7——闪烁灯

现在试着更进一步调整 LED。之前你已经学会简单地开关 LED，现在你愿意调整 LED 的亮度吗？你能用 Arduino 实现吗？其实这些效果都能用 Arduino 实现。

返回到基础问题。

需要的元件

5mm 漫射绿色 LED



限流电阻



把元件连起来

这个项目的电路只是简单地连接一个绿色的 LED，在地和数字引脚 1 之间设置一个限流电阻（见图 3-3）。

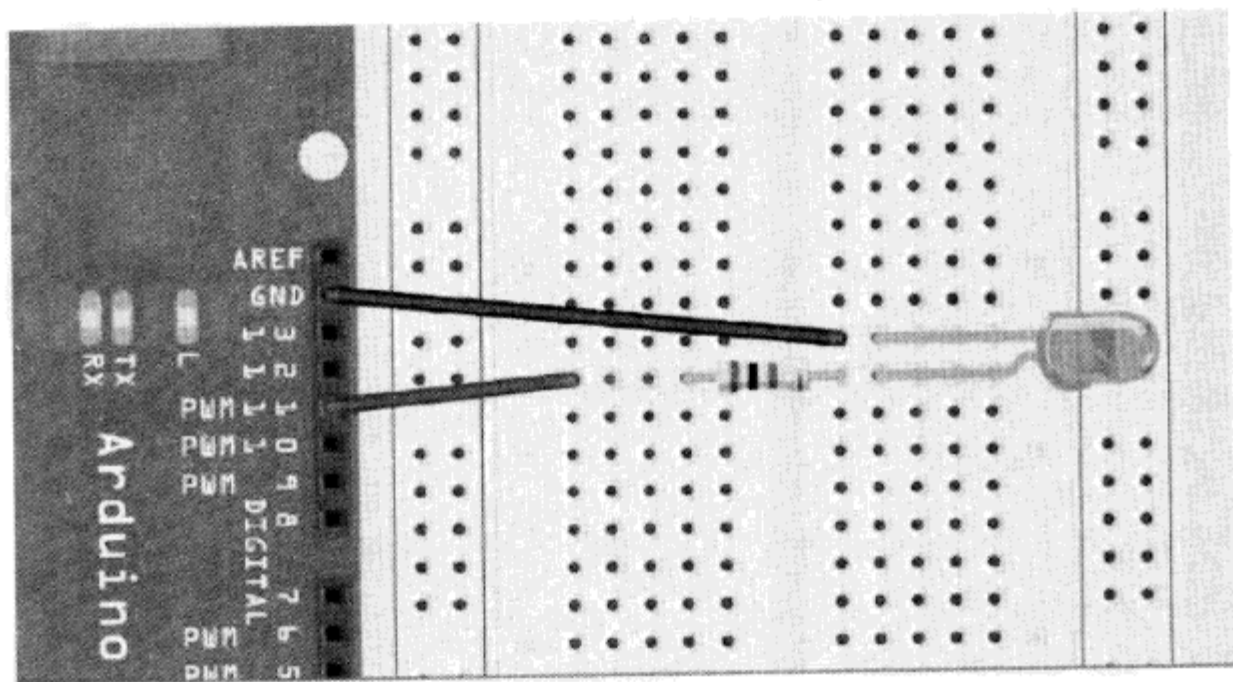


图 3-3 项目 7——闪烁灯电路图（参见彩色版插图）

输入代码

打开 Arduino IDE，输入清单 3-3 中的代码。

清单 3-3 项目 7 的代码

```
//项目 7——闪烁灯
int ledPin = 11;
float sinVal;
int ledVal;

void setup() {
    pinMode(ledPin, OUTPUT);
}

void loop() {
    for (int x=0; x<180; x++) {
        //当用 sin 函数时转化角度单位到弧度单位
        sinVal = (sin(x*(3.1412/180)));
        ledVal = int(sinVal*255);
```



```
        analogWrite(ledPin, ledVal);  
        delay(25);  
    }  
}
```

验证代码没有错误并上传代码。你将看到 LED 平稳地由开过渡到关或相反，并且亮度可调，而不是简单地开关。让我们看看代码是如何工作的。

代码回顾

这个项目的代码非常简单，但还需要做必要的说明。

首先，你给 LED 引脚设置变量，一个正弦波浮点数值（浮点数据类型），ledVal 将整型值送到数字 PWM 引脚 11。

这样做的目的是生成一个正弦波，并且让 LED 的亮度随着正弦函数渐变，而不仅仅是由全亮直接变为全暗。

此处采用 sin() 函数，这是一个数学函数，可算出一个角度的正弦值。该函数采用弧度单位。你不希望函数进入负半轴，因为这将产生负值亮度，而亮度值只能是从 0 到 255 的正值，所以设置 for 循环运行在 0~179 之间。

函数 sin() 采用弧度单位，而不是角度单位。可通过公式 $(3.1412/180)$ 将角度转化成弧度，再将结果乘以 255 之后转换到 ledVal。sin() 函数的结果是在 -1 到 1 之间的一个数，因此乘以 255 获得最大亮度。可采用如下语句将浮点值 sinVal 转换成整型值：

```
ledVal = int(sinVal*255);
```

之后，使用下面的语句将这个数值送到数字 PWM 引脚 11：

```
analogWrite(ledPin, ledVal);
```

类型转换意味着将浮点值转变为整型值（通过省略掉小数点以后的数）。但是，如何发送一个模拟值到一个数字引脚？观察一下 Arduino 开发板，如果查看数字引脚，你会发现其中 6 个引脚（3、5、6、9、10 和 11）旁边标注着 PWM，这些引脚不同于其他引脚，因为它们可以输出 PWM 信号。

PWM 是 Pulse Width Modulation 的缩写，这是一项用数字方法获得模拟量的技术。

Arduino 通过快速地转换引脚的开关送出一系列脉冲到这些引脚。快速地开关引脚并通过改变输出高电平与低电平所持续的时间来实现模拟一个 0 到 5V 之间变化的电压。高电平所占用的时间叫做脉冲宽度。

例如，如果你要使用 `analogWrite()` 输出 0 到数字 PWM 引脚 11，ON 的时间是 0，或者说它的占空比是 0%。如果你想输出数值 64（最大值 255 的 25%，即占空比为 25%），那么这个引脚 25% 的时间为 ON 状态，75% 的时间为 OFF 状态。数值 191 的占空比是 75%，数值 255 的占空比是 100%，这些脉冲运行的速度大约是 500Hz，或者说每 2 毫秒一个周期。

所以，在你编写的程序中，LED 迅速开关。如果占空比是 50%（数值 127），LED 将以 500Hz 的频率连续开关，所呈现的亮度为最大亮度的一半。你可以更进一步地采用这个基本原理使数字引脚输出一个模拟值到 LED。

注意，虽然只有 6 个引脚有 PWM 功能，但是如果你愿意也可以编写一个软件使所有的数字引脚都具有 PWM 功能。

稍后我们将回过头来用 PWM 和蜂鸣器产生一个可以听到的声音。

项目 8——RGB 彩灯

在上一个项目里，你学习了如何使用有 PWM 功能的 Atmega 芯片调整 LED 的亮度。现在用这种器件控制红、绿、蓝 LED 混合成任何我们希望的顏色。通过这种方式产生像在商店里看到的彩灯一样的效果。

需要的元件

这次使用 3 个 LED，一红、一绿、一蓝。

红色漫射 5mm LED



绿色漫射 5mm LED



蓝色漫射 5mm LED



3 个限流电阻



把元件连接起来

像图 3-4 那样连接 3 个 LED，拿一小块信封大小的纸，把它卷成一个圆柱状并粘牢。用圆筒罩住 3 个 LED。这将使每个 LED 中发出的光发生漫射，并把这些光混合成一种颜色。

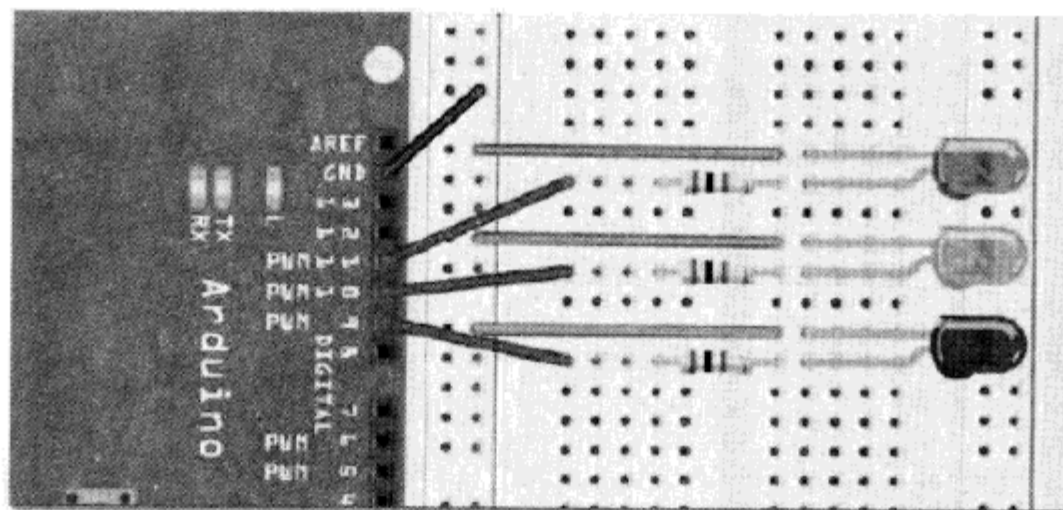


图 3-4 项目 8——RGB 彩灯电路图（参见彩色版插图）

输入代码

打开 Arduino IDE，输入清单 3-4 中的代码。

清单 3-4 项目 8 的代码

```
//项目 8 —— 彩灯
float RGB1[3];
float RGB2[3];
float INC[3];

int red, green, blue;

int RedPin = 11;
int GreenPin = 10;
int BluePin = 9;

void setup()
{
    randomSeed(analogRead(0));

    RGB1[0] = 0;
    RGB1[1] = 0;
    RGB1[2] = 0;
```

```

    RGB2[0] = random(256);
    RGB2[1] = random(256);
    RGB2[2] = random(256);
}

void loop()
{
    randomSeed(analogRead(0));

    for (int x=0; x<3; x++) {
        INC[x] = (RGB1[x] - RGB2[x]) / 256; }

    for (int x=0; x<256; x++) {
        red = int(RGB1[0]);
        green = int(RGB1[1]);
        blue = int(RGB1[2]);

        analogWrite (RedPin, red);
        analogWrite (GreenPin, green);
        analogWrite (BluePin, blue);
        delay(100);

        RGB1[0] -= INC[0];
        RGB1[1] -= INC[1];
        RGB1[2] -= INC[2];
    }
    for (int x=0; x<3; x++) {
        RGB2[x] = random(556)-300;
        RGB2[x] = constrain(RGB2[x], 0, 255);
        delay(1000);
    }
}

```

当运行这个程序时，将看到颜色慢慢改变。你已经实现了属于自己的彩灯！

代码回顾

构成彩灯的 LED 包括红色、绿色和蓝色 LED。同样道理，计算机显示器也是由小的红、绿、蓝点组成的。可以通过调整三个 LED 中每个灯的亮度产生不同的颜色，用这种办法给出不同颜色的 RGB 数值。

当然可以用一个 RGB LED 来代替。这是一个单一的 5mm LED，有 4 个引脚（有些有更多的引脚），一个引脚是共用的正极（阳）或者共用的阴极（负），其他三个引脚连接到灯内的 LED 的另外一个终端。它只是简单地把三个颜色的 LED 封装在同一个 5mm LED 中。还有更紧凑的产品，但是价格也更贵。

RGB 数值 255,0,0 是纯红色，值 0,255,0 是纯绿色，值 0,0,255 是纯蓝色。通过混合这三种色彩可以得到任何颜色。这就是三原色原理（见表 3-1）。注意如果你仅开或关 LED（不尝试其他的亮度），只能得到几种不同的颜色。

表 3-1 不同的 LED 开、关组合所能产生的颜色

红 色	绿 色	蓝 色	颜 色
255	0	0	红色
0	255	0	绿色
0	0	255	蓝色
255	255	0	黄色
0	255	255	蓝绿色
255	0	255	紫红色
255	255	255	白色

用纸筒漫射光混合颜色的方法非常好，LED 可放置在能够漫射光的任何物体里，其中的一种方法是把光源放在一个漫反射塑料壳里。可尝试把光源放在一个乒乓球中，或一个小的白塑料瓶中（塑料越薄越好）。

通过使用 PWM 调整亮度，你可以得到任何颜色。通过把 LED 聚集在一起，将色彩混合，三种颜色的光谱加在一起形成一个单一的颜色（见图 3-5）。使用 PWM 可以产生 0~255 之间的全部颜色，共 16 777 216 种颜色（ $256 \times 256 \times 256$ ）。

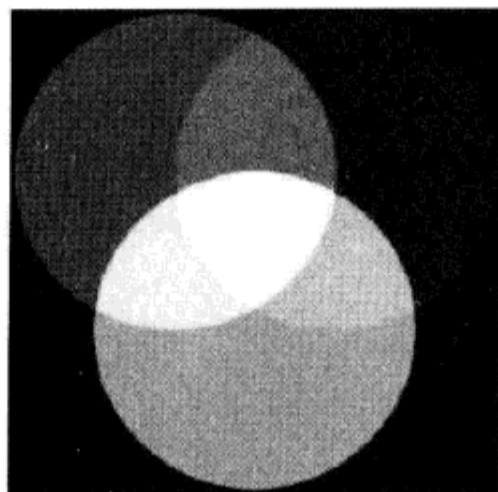


图 3-5 混合 R、G、B 获得不同颜色（参见彩色版插图）

在代码中，开始定义了一些浮点类型的数组和整型变量来存储 RGB 值，如下：

```
float RGB1[3];
float RGB2[3];
float INC[3];

int red, green, blue;
```

在 `setup` 函数中，有下列语句：

```
randomSeed(analogRead(0));
```

`randomSeed` 函数产生一个随机数（实际上是伪随机数）。计算机芯片实际上不可能产生真正的随机数，但是它可以通过查看不同的存储器或者通过查找有不同值的表形成一个伪随机数。通过设置一个种子，告诉计算机从哪块存储器或表中的哪个数开始计算。在这里所指定的随机数种子是从模拟引脚 0 读到的数。因为模拟引脚 0 是悬空的，因此将读到由模拟噪声引起的随机数。

一旦给随机数设置一个种子，就可以用 `random()` 函数生成一个随机数。需要两组 RGB 值，存储在三个元素的数组里。RGB1 存储你希望灯开始时的 RGB 值（在这里，都是零或全关闭状态）。

```
RGB1[0] = 0;
RGB1[1] = 0;
RGB1[2] = 0;
```

RGB2 数组是一组随机 RGB 值，表示你所希望转换成的颜色。

```
RGB2[0] = random(256);
RGB2[1] = random(256);
RGB2[2] = random(256);
```

这是用 `random(256)` 给数组元素用随机数赋值。`Random(256)` 产生一个在 0 到 255（含 255）之间的随机数（因此所产生的数总是大于 0）。

如果传递给 `random()` 函数一个数，它将返回 0 到这个数之间的一个数。比如，`random(1000)` 将返回一个 0 到 999 之间的数。如果提供两个数作为参数，它将返回一个从较小数（含）到较大数减 1 之间的随机数。例如，`random(10,100)` 将返回 10 到 99 之间的

一个随机数。

在主程序循环中，你首先查看开始和结束时的 RGB 值，并计算出需要什么值作为增量以在 256 步内（因为 PWM 值只能在 0 到 255 之间）从一个值转换到另外一个值，使用下面的代码实现以上动作：

```
for (int x=0; x<3; x++) {
    INC[x] = (RGB1[x] - RGB2[x]) / 256; }
```

这个 for 循环通过计算两个亮度值之差，并除以 256 实现为 R、G、B 设置 INC 增量。

还需要另外一个 for 循环：

```
for (int x=0; x<256; x++) {

    red = int(RGB1[0]);
    green = int(RGB1[1]);
    blue = int(RGB1[2]);

    analogWrite (RedPin, red);
    analogWrite (GreenPin, green);
    analogWrite (BluePin, blue);
    delay(100);

    RGB1[0] -= INC[0];
    RGB1[1] -= INC[1];
    RGB1[2] -= INC[2];
}
```

它给 RGB1 数组中的元素设置一个红、绿、蓝值；将这些数值写到数字引脚 9、10 和 11 中；减去增量值；重复这个过程 256 次，慢慢将一种随机颜色变成下一种颜色。在每一步中延时 100 毫秒，保证这个过程缓慢并且稳步地进行。可以通过调整这个数值让颜色变化的速度变慢或变快。也可以增加一个电位计让使用者自己设置颜色变化的速度。

现在已经通过 256 步缓慢地从一个随机颜色变到另外一个，RGB1 数组变得与 RGB2 数组的数值相同。下面需要确定另外 3 个随机值，为下一次变化做好准备。用另一个 for 循环实现。

```
for (int x=0; x<3; x++) {
```

```

    RGB2[x] = random(556)-300;
    RGB2[x] = constrain(RGB2[x], 0, 255);
    delay(1000);
}

```

随机数是通过在 0 到 556 之间选择一个随机数之后减去 300 得到的。用这种方法，每次强制用不同的颜色初始化，以保证不总是从大青灰阴影颜色开始。你有 300 次机会在 556 中得到一个负数，因此强制有一个或多个颜色以减少的方向在初始和结束两个颜色之间变化，而不总是向增强方向变化。下一句代码通过使用 `constrain()` 函数确保这个数被送往 PWM 引脚时不会出现负数。

`constrain()` 函数需要 3 个参数：x、a 和 b。这里 x 是一个被约束的数，a 是最小值，b 是最大值。因此 `constrain()` 函数查看 x 的值，确定它是否在 a 到 b 范围内，如果它小于 a，则设置为 a。如果高于 b，则设置为 b。在你的例子里，可在 0 到 255 之间设置，这是 PWM 的输出范围。

因为使用 `random(556)-300` 作为 RGB 值，这些值中有些可能小于 0，`constrain` 函数可以确保送到 PWM 中的值不小于 0。

练 习

看是否能改变代码使颜色沿着彩虹色变化，而不是在两个随机颜色之间变化。

项目 9——LED 火焰效果

项目 9 将再通过 PWM 使 LED 产生随机的亮度变化去模拟火焰闪烁的效果。例如，如果你把这个 LED 放在一个模拟铁路设施的小房子里，可使这个房子看起来像正在着火。或者你可以把它放在壁炉里代替原木火焰。这是一个说明如何用 LED 为电影、舞台表演、透视图、铁路设施模型产生特殊效果的简单例子。

需要的元件

这次我们要用 3 个 LED，1 红、2 黄。

红色 5mm 漫射 LED



2 个黄色 5mm 漫射 LED



3 个限流电阻



把元件连接起来

给 Arduino 断电，如图 3-6 所示连接这 3 个 LED。它本质上与项目 8 的电路相同，但是用了一个红色 LED 和两个黄色 LED，而不是一红、一绿、一蓝。还有如果用一个圆筒纸散射光线或使光线从一个白纸板或反射器的表面散射到另一个表面时，效果看起来会更好。

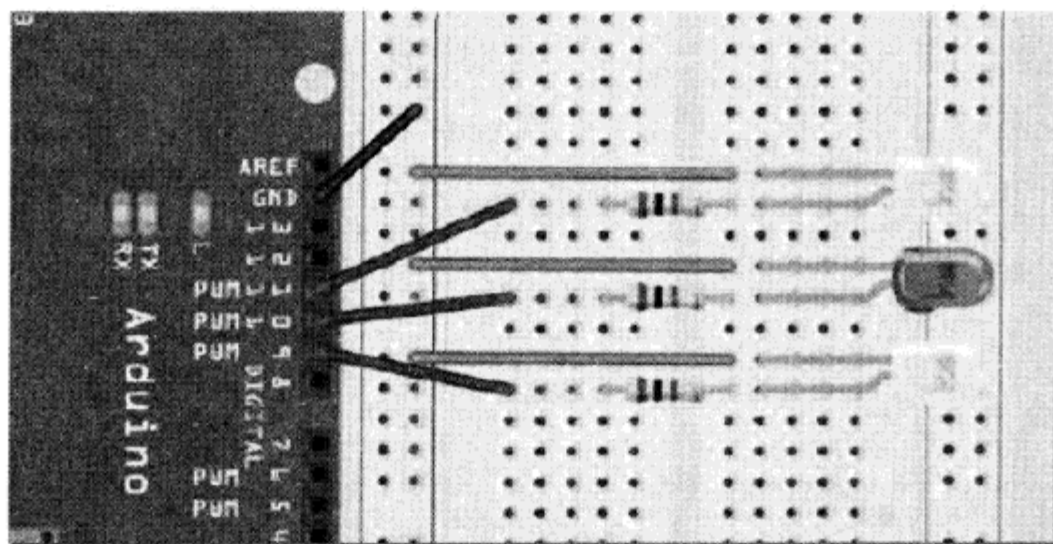


图 3-6 项目 9——LED 火焰效果电路图（参见彩色版插图）

输入代码

打开 Arduino IDE 并且输入清单 3-5 中的代码。

清单 3-5 项目 9 的代码

```
// 项目 9——LED 火焰效果
int ledPin1 = 9;
int ledPin2 = 10;
int ledPin3 = 11;

void setup()
{
    pinMode(ledPin1, OUTPUT);
    pinMode(ledPin2, OUTPUT);
    pinMode(ledPin3, OUTPUT);
}
```

```

}

void loop()
{
    analogWrite(ledPin1, random(120)+135);
    analogWrite(ledPin2, random(120)+135);
    analogWrite(ledPin3, random(120)+135);
    delay(random(100));
}

```

单击 IDE 顶部的 **Verify/Compile** 按钮确定代码中没有错误。如果代码内没有语法错误，单击 **Upload** 按钮。

如果一切顺利，LED 将以一种随机方式闪烁，模拟火花或者火焰的效果。

代码回顾

让我们看一下这个项目的代码。首先声明并初始化一些整型变量，它们用来存储连接到 LED 上的数字引脚值。

```

int ledPin1 = 9;
int ledPin2 = 10;
int ledPin3 = 11;

```

然后设置这些引脚模式为输出：

```

pinMode(ledPin1, OUTPUT);
pinMode(ledPin2, OUTPUT);
pinMode(ledPin3, OUTPUT);

```

在主程序循环中算出一个 0 到 120 之间的随机数与 135 相加后的数值，这样可获得 LED 的最大亮度并赋值给 PWM 引脚 9、10 和 11：

```

analogWrite(ledPin1, random(120)+135);
analogWrite(ledPin2, random(120)+135);
analogWrite(ledPin3, random(120)+135);

```

最后在 0 和 100 毫秒之间随机延时：

```

delay(random(100));

```

主循环重新开始，产生闪烁效果。把灯罩上一个白纸板或用镜子反射到墙上，你可以看到一个非常真实的火焰效果。

这个项目的硬件非常简单，你现在应该已知道它是怎么工作的了，所以让我们进入项目 10。

练 习

尝试完成以下两个练习。

练习 1：使用一个蓝色的和/或一个或两个白色的 LED，看看你是否能产生氩弧焊效果的闪光。

练习 2：使用蓝色和红色 LED 产生消防车灯效果。

项目 10——串口控制彩灯

对于项目 10，需要回顾项目 8 中——RGB 彩灯的电路，但是现在将步入串口通信世界。要通过 Arduino IDE 的串口监视器从 PC 向 Arduino 发送命令控制彩灯。串口通信是每次通过串口线传送一比特数据的通信方法。

这个项目中也会介绍如何操作文本字符串。因此，现在先像项目 8 那样搭建硬件，然后输入新的代码。

输入代码

打开 Arduino IDE，输入清单 3-6 中的代码。

清单 3-6 项目 10 的代码

```
//项目 10——串口控制彩灯
char buffer[18];
int red, green, blue;

int RedPin = 11;
int GreenPin = 10;
int BluePin = 9;
```

```

void setup()
{
    Serial.begin(9600);
    Serial.flush();
    pinMode(RedPin, OUTPUT);
    pinMode(GreenPin, OUTPUT);
    pinMode(BluePin, OUTPUT);
}

void loop()
{
    if (Serial.available() > 0) {
        int index=0;
        delay(100); //等待缓冲区填满
        int numChar = Serial.available();
        if (numChar>15) {
            numChar=15;
        }
        while (numChar-->0) {
            buffer[index++] = Serial.read();
        }
        splitString(buffer);
    }
}

void splitString(char* data) {
    Serial.print("Data entered: ");
    Serial.println(data);
    char* parameter;
    parameter = strtok (data, " ,");
    while (parameter != NULL) {
        setLED(parameter);
        parameter = strtok (NULL, " ,");
    }

    //清除串口缓冲区中的文本
    for (int x=0; x<16; x++) {
        buffer[x]='\0';
    }
    Serial.flush();
}

```



```
void setLED(char* data) {
    if ((data[0] == 'r') || (data[0] == 'R')) {
        int Ans = strtol(data+1, NULL, 10);
        Ans = constrain(Ans,0,255);
        analogWrite(RedPin, Ans);
        Serial.print("Red is set to: ");
        Serial.println(Ans);
    }
    if ((data[0] == 'g') || (data[0] == 'G')) {
        int Ans = strtol(data+1, NULL, 10);
        Ans = constrain(Ans,0,255);
        analogWrite(GreenPin, Ans);
        Serial.print("Green is set to: ");
        Serial.println(Ans);
    }
    if ((data[0] == 'b') || (data[0] == 'B')) {
        int Ans = strtol(data+1, NULL, 10);
        Ans = constrain(Ans,0,255);
        analogWrite(BluePin, Ans);
        Serial.print("Blue is set to: ");
        Serial.println(Ans);
    }
}
```

确认程序没有错误，然后把它上传到 Arduino 中。

注意上传程序之后看起来像什么也没有发生，这是因为程序正在等待你的输入。单击 Arduino IDE 工具栏上的 SerialMonitor 按钮，开始串口监控。在串口监控文本窗口中可以手动输入 3 个 LED 的 R、G、B 值。LED 将转变成你所输入的颜色。

如果输入 R255，红色 LED 以最大亮度显示。如果输入 R255、G255，红色和绿色 LED 都以最大亮度显示。现在输入 R127、G100、B255，得到一个漂亮的略带有紫色的颜色。输入 r0、g0、b0 将关闭所有 LED。

输入字符可接受小写或大写的 R、G、B 字母，后面的数字可从 0 到 255。所有超过 255 的值自动降为 255，你可以在各参数之间输入一个逗号或空格，而且可以一次输入 1 个、2 个或 3 个 LED 值，例如：

```
r255 b100
r127 b127 g127
```

```
G255,B0
B127,R0,G255
```

代码回顾

这个项目介绍了几个新的概念，包括串口通信、指针、字符串等。请注意，我们要花大量时间去解释这些概念。

首先，设置一个字符型数组存储文本字符串，它有 18 个字符长。这比实际需要的最大长度 16 个字符长，保证不会发生“缓冲器溢出”错误。

```
char buffer[18];
```

之后声明了整型变量存储红、绿和蓝颜色数值和数字引脚号：

```
int red, green, blue;

int RedPin = 11;
int GreenPin = 10;
int BluePin = 9;
```

在 `setup` 函数中设置 3 个数字引脚为输出模式。但是在这之前，使用了 `serial.begin` 函数。

```
void setup()
{
    Serial.begin(9600);
    Serial.flush();
    pinMode(RedPin, OUTPUT);
    pinMode(GreenPin, OUTPUT);
    pinMode(BluePin, OUTPUT);
}
```

`Serial.begin` 告诉 Arduino 开始串口通信，括号内的数字（在这里是 9600）是波特率（每秒钟内的信号或脉冲数）。串口以这个波特率通信。

`Serial.flush` 命令清空串口中残存的字符，因此串口是空的，为输入/输出做好准备。

串口是 Arduino 和外界进行通信的一个简单的方法。在本例中，串口用于在 PC 和 Arduino IDE 中的串口监视器之间通信。在主循环中，有一个 if 语句：

```
if (Serial.available() > 0) {
```

它使用 `Serial.available` 函数检查是否有字符发送到串口。如果已经收到字符，表示条件满足，那么执行 if 语句中的代码：

```
if (Serial.available() > 0) {
    int index=0;
    delay(100); //等待缓冲区充满
    int numChar = Serial.available();
    if (numChar>15) {
        numChar=15;
    }
    while (numChar-->0) {
        buffer[index++] = Serial.read();
    }
    splitString(buffer);
}
```

声明一个叫 `index` 的整型数，并且初始化为 0，这个整型数用于存储字符数组中字符指针的位置。

之后延时 100 毫秒，延时的目的是保证在你处理这些数据之前串口缓冲区（内存中存储接收到的串口数据的地方，存储在这儿的数据优先处理）是满的。如果不延时，在接收到全部数据之前可能函数已经执行，并且开始处理所收到的文本串，因为串口通信相对于复位代码的运行速度是很慢的。当你传送一个字符串时，`Serial.available` 函数马上产生一个大于 0 的值，if 语句将开始执行。如果没有 `delay(100)` 语句，在接收到所有的字符串之前，if 语句中的代码将开始执行，所以在串口上出现的字符会仅仅是开头的几个字符。

等待 100 毫秒使送来的数据将缓冲区填满。之后声明并初始化 `numChar` 整型变量，记录字符串中的字符数。

因此，如果在串口监视器中输入的字符串是：

```
R255,G255,B255
```

`numChar` 的值将是 17，而不是 16，因为在文本的每行末端都有一个看不见的字符，叫做 NULL 字符，它告诉 Arduino 什么时候到达文本行的末端。

接下来的 if 语句检查 `numChar` 的值是否大于 15，如果是，设置它为 15，以保证不会

溢出字符数组 `buffer[18]`。

下一个是 `while` 语句。这是之前你还没有遇到过语句，我来解释一下。

你已经用过 `for` 循环，`While` 也是一个循环，但是，只有在条件是真时才执行。

语法如下：

```
while(expression){
    //语句
}
```

本项目代码中，`while` 循环是：

```
while (numChar--){
    buffer[index++] = Serial.read();
}
```

循环的条件是检查 `numChar`。换句话说，它检查存储在 `numChar` 中的整数值是否为 0，注意 `numChar` 后有“—”符号，它表示使用后递减：也就是变量值使用后递减。如果已使用 `numChar`，在求值之前，`numChar` 中的值将递减（每次减 1）。在例子中，`while` 循环检查 `numChar` 值，然后从中减 1，如果 `numChar` 递减前不为 0，程序执行 `while` 语句中的代码。

`numChar` 设置为输入串口监视器窗口中的文本串长度。因此，`while` 语句中的代码将要循环很多次。

在 `while` 循环中的代码是：

```
buffer[index++] = Serial.read();
```

这使缓冲区数组中的每一个元素为从串口中读入的一个字符。换句话说，这个语句用你输入串口监视器窗口中的字符填满缓冲区数组。

`Serial.read()` 函数读串口输入数据，每次一个字节。因此现在字符数组已经用你输入串口监视器的字符填满，当 `numChar` 变为 0（如到字符串结尾）时 `while` 循环就结束了。

在 `while` 循环之后是：

```
splitString (buffer);
```

这里调用了你所定义的两个函数中的一个，叫做 `splitString()`，这个函数如下：


```
void splitString(char* data) {
    Serial.print("Data entered: ");
    Serial.println(data);
    char* parameter;
    parameter = strtok (data, " ,");
    while (parameter != NULL) {
        setLED(parameter);
        parameter = strtok (NULL, " ,");
    }

    //清除缓冲区内的文本
    for (int x=0; x<16; x++) {
        buffer[x]='\0';
    }
    Serial.flush();
}
```

这个函数不返回数据，因此它的数据类型设置为 `void`。给这个函数传递一个名为 `data` 的字符型参数，但在 C++ 程序设计语言里，不允许给函数传送一个字符型数组，可通过使用指针绕过这个限制。在变量名之前加一个星号，如 `*data`，表明这是一个指针类型的变量。

在 C 语言里指针是一个高级概念，我要深入详细地说明它。如果需要了解更多，可参考讲 C 语言编程的书。现在所需要知道的是，通过声明一个数为指针，可使它变成一个指向另外一个变量的变量。

你既可以通过使用 `&` 符号指定后面的变量为内存中存储变量的一个地址，也可以像例子中那样，通过使用 `*` 符号获得内存中的数值。你用它来欺骗系统，因为如同已经提到的，你不能传送字符数组给函数，然而允许给函数传送一个指向字符数组的指针。因此，声明一个叫 `data` 的字符型数据变量，但是在它前面加一个 `*` 符号，这意味着它是一个指向存储在变量存储器中的字符型数据的指针。

当调用 `splitString()` 函数时，传给函数 `buffer` 中的内容（就像你在上面看到的那样，实际上是一个指向它的指针）：

```
splitString(buffer)
```

这样你调用这个函数并传递给它 `buffer` 字符数组中的全部内容。

第一个函数是：

```
Serial.print("Data entered: ");
```

这是从 Arduino 传送数据给 PC 的方法。在这个例子里，`print` 命令通过 USB 电缆传送括号内的内容给 PC，可在 PC 串口监视器窗口读出这些信息。在这个例子里，你传送的词是“Data entered:”。注意，这些文本必须括在双引号内。下一行与之类似：

```
Serial.println(data);
```

再次传送数据给 PC。这次传送了一个字符变量叫 `data`，这是传送给函数的 `buffer` 字符数组内容的一个副本。如果文本字符串输入是：

```
R255 G127 B56
```

然后

```
Serial.println(data);
```

函数将这个文本字符串送入 PC，并把它显示在串口监视器窗口上（要保证首先已经打开了串口监视器）。

这次 `print` 结尾有 `ln` 两个字母，使函数名称为 `println`。它的意思是“用一整行打印”。

当使用 `print` 命令打印时，光标（下一个字符将要出现的位置）仍保持在已打印字符的结尾。当使用 `println` 命令打印时，换行命令激活，因此打印文本并且光标跳到下一行。

```
Serial.print("Data entered: ");
Serial.println(data);
```

所以，第一个命令打印“Data entered:”之后光标保持在这个文本的末尾；第二个命令打印数据（就是名为 `buffer` 的数组的内容）之后激活换行，光标跳到下一行。如果在它之后发出另外一个 `print` 或 `println` 语句，不管在串口监视器窗口打印的是什么，都出现在下一行。

之后创建一个新的变量，叫做 `parameter`：

```
char* parameter;
```

当用变量获得数组中的元素时，它必须与数组是相同的类型。因此使用 `*` 符号。不能在两个不同的数据类型之间传递数据，如果确实要在不同的数据类型变量之间传递数据，首先必须进行数据类型转换。这个变量是本地变量的一个例子，本地变量只能被这个函数内部的代码看到。如果你试图在 `splitString()` 函数外引用 `parameter` 变量，将导致错误。

之后用到 `strtok` 函数。这是一个在操作字符串时非常有用的函数。`strtok` 这个函数是取 `string` 和 `token` 这两个单词的词头命名的。`string` 是字符串，而 `token` 表明这个函数的目的是分割这个字符串。在本例中，是在字符串中查找一个空格或逗号。这个函数用来将一个字符串分割成更小的字符串。

传递字符串数据给 `strtok` 函数作为第一个参数，分割符（必须在双引号内）作为第二个参数，因此有以下语句：

```
parameter = strtok (data, " ,");
```

函数把字符串在空格或逗号处分开。

因此，如果你的字符串是：

```
R127 G56 B98
```

那么执行这个语句之后 `parameter` 的值将是：

```
R127
```

因为 `strtok` 函数在第一个空格或逗号出现处分开字符串。

将你希望分开的那部分字符串（例子中第一个空格或逗号处）赋值给 `parameter` 之后，进入 `while` 循环，条件是 `parameter` 非空（也就是还没有到达字符串的尾部）：

```
while (parameter != NULL) {
```

在这个循环中，我们调用第二个函数：

```
setLED(parameter);
```

（我们将在后面仔细研究它）之后，设置 `parameter` 为下一个空格或逗号之前的字符串。通过传递一个 `NULL` 参数给 `strtok` 来实现，语句如下：

```
parameter = strtok (NULL, " ,");
```

这个语句告诉 `strtok` 函数从逗号处断开字符串。

因此函数的整个部分如下；

```

char* parameter;
parameter = strtok (data, " ,");
while (parameter != NULL) {
    setLED(parameter);
    parameter = strtok (NULL, " ,");
}

```

它只是简单地把用空格或逗号分开的文本字符串分成不同部分，并且把部分字符传给一个叫 `setLED()` 的函数。

这个函数的最后部分只是用 `NULL` 字符填充 `buffer` 数组，使用 “\0” 符号实现，之后清除串口缓冲区内的串口数据，从而准备好下一次数据的输入：

```

//清除缓冲区内的文本
for (int x=0; x<16; x++) {
    buffer[x]='\0';
}
Serial.flush();

```

`setLED()` 函数获取字符串内容，并且设置相应的 LED 颜色为你选定的颜色，因此如果输入的字符串是：

```
G125 B55
```

`splitString()` 函数把它分成两个独立的部分：

```
G125
B55
```

并将这两个短字符串传给 `setLED()` 函数，由这个函数读短字符串，确定选择哪个 LED 及设置 LED 相应的亮度值。

让我们看看称为 `setLED()` 的第二个函数：

```

void setLED(char* data) {
    if ((data[0] == 'r') || (data[0] == 'R')) {
        int Ans = strtol(data+1, NULL, 10);
        Ans = constrain(Ans,0,255);
        analogWrite(RedPin, Ans);
        Serial.print("Red is set to: ");
    }
}

```



```

        Serial.println(Ans);
    }
    if ((data[0] == 'g') || (data[0] == 'G')) {
        int Ans = strtol(data+1, NULL, 10);
        Ans = constrain(Ans,0,255);
        analogWrite(GreenPin, Ans);
        Serial.print("Green is set to: ");
        Serial.println(Ans);
    }
    if ((data[0] == 'b') || (data[0] == 'B')) {
        int Ans = strtol(data+1, NULL, 10);
        Ans = constrain(Ans,0,255);
        analogWrite(BluePin, Ans);
        Serial.print("Blue is set to: ");
        Serial.println(Ans);
    }
}

```

这个函数包含了 3 个相似的 if 语句，因此只选其中一个做例子：

```

    if ((data[0] == 'r') || (data[0] == 'R')) {
        int Ans = strtol(data+1, NULL, 10);
        Ans = constrain(Ans,0,255);
        analogWrite(RedPin, Ans);
        Serial.print("Red is set to: ");
        Serial.println(Ans);
    }

```

if 语句检查字符串 data[0] 中的第一个字符是否是 r 或者 R（在 C 语言定义中大写和小写字母完全不同）。用逻辑命令 OR（符号||）检查字符是否是 r 或 R，再决定如何做。

如果字符是 r 或 R，if 语句知道你想改变红色 LED 的亮度，函数中的代码将被执行。首先声明一个整型数 Ans（它的作用范围在 setLED 函数内），并用 strtol（字符型转换成长整型）函数转化字母 R 后的字符串为一个整型。strtol 函数有 3 个参数：一个传递给它的字符串类型的参数、一个指向整数后下一个字符的指针参数（这个参数在这里用不到，因为已经用 strtok 函数拆分了字符串，因此传递一个 NULL 符号）和一个基数（在这个例子里，基数是 10，因为使用的是普通的十进制数而不是二进制、八进制、十六进制，它们相应的基数是 2、8、16）。简要地说，你声明了一个整型，并设置它的值为字母 R 之后的字符串所表示的数值（或者位数）。

下一步，用 constrain 函数确保 Ans 在 0~255 之间。之后执行 analogWrite 函数写红色 LED 的引脚，并将 Ans 的值传到引脚。然后程序输出“Red is set to:”。紧跟着 Ans 值返回串口监视器。另外两个 if 语句的原理基本相同，只是用于绿色和蓝色 LED 的设置。

在这个项目中，你已经学到了大部分基础知识和许多新概念。为了使你精确地理解代码中什么操作正在进行，我在项目代码（记住是用 C 语言）旁边用伪代码解释（本质上，计算机语言能更精确地描述用词和思想）。两者比较如表 3-2 所示。

表 3-2 项目 10 中的伪代码解释

C 语言代码	伪 代 码
<pre>// Project 10 - Serial controlled RGB lamp char buffer[18]; int red, green, blue; int RedPin = 11; int GreenPin = 10; int BluePin = 9; void setup() { Serial.begin(9600); Serial.flush(); pinMode(RedPin, OUTPUT); pinMode(GreenPin, OUTPUT); pinMode(BluePin, OUTPUT); } void loop() { if (Serial.available() > 0) { int index=0; delay(100); // let the buffer fill up int numChar = Serial.available(); if (numChar>15) { numChar=15; } while (numChar-->0) { buffer[index++] = Serial.read(); } splitString(buffer); } }</pre>	<p>项目号和名称的说明</p> <p>声明一个 18 个字符的字符数组</p> <p>声明 3 个整型数 red、green 和 blue</p> <p>一个整数表明接红色 LED 的引脚</p> <p>一个整数表明接绿色 LED 的引脚</p> <p>一个整数表明接蓝色 LED 的引脚</p> <p>setup 函数</p> <p>设置串口通信在 9600 波特率下运行</p> <p>清空串口</p> <p>设置红色 LED 引脚为输出引脚</p> <p>设置绿色 LED 引脚为输出引脚</p> <p>设置蓝色 LED 引脚为输出引脚</p> <p>主循环</p> <p>如果数据送入串口</p> <p>声明整数 index，初始化为 0</p> <p>等待 100 毫秒</p> <p>设置 numChar 为串口输入数据</p> <p>如果 numChar 大于 15 个字符或更多，设置它为 15 个字符</p> <p>当 numChar 不是 0 时（从它中减 1）</p> <p>设置 element[index]值读 1（加 1）</p> <p>调用 splitString 函数把 buffer 中的数据送入其中</p>

续表

C 语言代码	伪 代 码
<pre> void splitString(char* data) { Serial.print("Data entered: "); Serial.println(data); char* parameter; parameter = strtok (data, " ,"); while (parameter != NULL) { setLED(parameter); parameter = strtok (NULL, " ,"); } // Clear the text and serial buffers for (int x=0; x<16; x++) { buffer[x]='\0'; } Serial.flush(); } void setLED(char* data) { if ((data[0] == 'r') (data[0] == 'R')) { int Ans = strtol(data+1, NULL, 10); Ans = constrain(Ans,0,255); analogWrite(RedPin, Ans); Serial.print("Red is set to: "); Serial.println(Ans); } if ((data[0] == 'g') (data[0] == 'G')) { int Ans = strtol(data+1, NULL, 10); Ans = constrain(Ans,0,255); analogWrite(GreenPin, Ans); Serial.print("Green is set to: "); Serial.println(Ans); } if ((data[0] == 'b') (data[0] == 'B')) { int Ans = strtol(data+1, NULL, 10); Ans = constrain(Ans,0,255); analogWrite(BluePin, Ans); Serial.print("Blue is set to: "); Serial.println(Ans); } } </pre>	<p>splitstring 函数的参考字符 打印 "Data entered:" 打印 data 值之后换行 声明 char 指针数据类型 parameter 设置它为到第一个逗号的位置处的字符串 当 parameter 的内容不是空时 调用 setLED 函数 设置 parameter 为字符串下一个部分</p> <p>另外一个逗号 运行下一行 16 次 设置 buffer 元素为 NULL (空)</p> <p>溢出逗号</p> <p>调用 setLED 传递 buffer 参数 如果第一个字符是 r 或 R 设置整数 Ans 为字符串下一个位置的数 确认它在 0~255 之间 写数值到红色 LED 引脚 打印 "Red is set to:" 之后是 Ans 的值</p> <p>如果第一个字符是 g 或 G 设置整数 Ans 为字符串下一个位置的数 确认它在 0~255 之间 写数值到绿色 LED 引脚 打印 "Green is set to:" 之后是 Ans 的值</p> <p>如果第一个字符是 b 或 B 设置整数 Ans 为字符串下一个位置的数 确认它在 0~255 之间 写数值到蓝色 LED 引脚 打印 "Blue is set to:" 之后是 Ans 的值</p>

希望伪代码能够帮助读者弄明白程序的含义。

小结

第3章介绍了许多新的命令和概念。你已经学习了数组及如何使用数组，如何从引脚读模拟值，如何使用 PWM 引脚及串口通信的基础。知道了如何通过串口读和发送数据，意味着你可以使用 Arduino 与各种串口设备或者其他有简单串口协议的设备通信了。你将在本书的后面用到串口通信。

本章的主题和概念

- 数组及如何使用它们
- 什么是电位计（或可调电阻）及如何使用它
- 从一个模拟输入引脚读电压值
- 如何使用数学函数 \sin
- 转换角度值到弧度值
- 转换不同类型的变量
- 脉宽调制（PWM）和如何利用 `analogWrite()` 函数使用它们
- 用不同的 RGB 值产生彩色光
- 使用 `random()` 和 `randomSeed()` 函数产生随机数
- 如何使用相同的电路、不同的代码产生各种灯光效果
- 串口通信的概念
- 使用 `Serial.begin()` 函数设置串口波特率
- 使用串口监视器发送命令
- 使用数组生成字符串
- 使用 `Serial.flush` 清空串口缓冲区
- 使用 `Serial.available` 检查数据是否发送到串口

- 当条件满足时用 `while()` 语句产生一个循环
- 用 `Serial.read()` 函数从串口中读数据
- 指针的基本概念
- 用 `Serial.print()` 或 `Serial.println()` 函数发送数据到串口监视器
- 用 `strtok()` 函数操作字符串
- 用 `strtol()` 函数转变一个字符串为长整型数
- 用 `constrain()` 函数限制一个变量值



第4章

简单的发声器和传感器

本章将介绍如何产生声音。通过 Arduino 控制一个压电扬声器，给你的设备上增加报警、蜂鸣、声音提示等功能。对于 Arduino IDE 的 0018 版本，由于增加了一个新函数，使得在 Arduino 项目上增加声音功能非常容易。你也将学到如何使用压电元件作为传感器，学习如何从其中读出电压值。最后，你将学到光电传感器的知识。

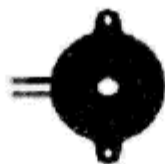
让我们从一个简单的汽车报警器项目开始学习使用 `tone()` 函数在 Arduino 中产生声音。

项目 11——压电声音报警器

连接压电扬声器到 Arduino 数字输出引脚，配合相应软件就可以产生尖啸报警的声音。其原理与项目 7 中使用正弦波产生脉冲灯光一样，只不过这次把 LED 换成了一个扬声器或压电盘。

需要的元件

压电扬声器（或压电盘）



两路螺钉式接线端子



把元件连接起来

首先，把 USB 线拔掉确保 Arduino 没有上电。然后拿起压电扬声器，它的两条引线

拧到螺钉式接线端子上。连接螺钉式接线端子到面包板之后，把接线端子连接到 Arduino 上，如图 4-1 那样。现在把 Arduino 连回到 USB 电缆给它上电。

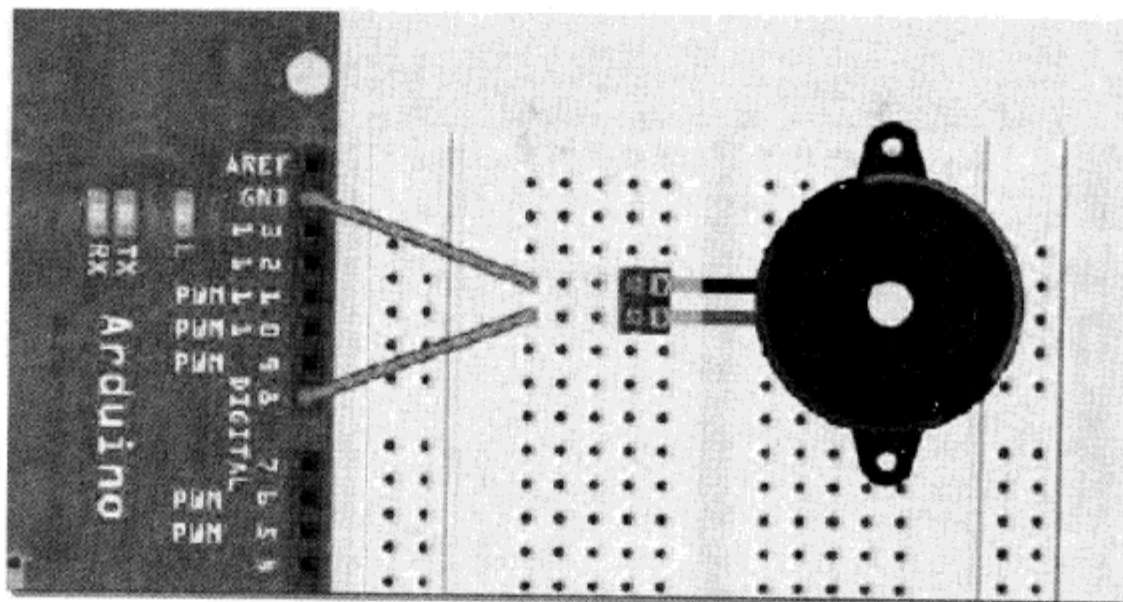


图 4-1 项目 11——压电声音报警器电路图（参见彩色版插图）

输入代码

打开 Arduino IDE 之后，输入清单 4-1 中的代码。

清单 4-1 项目 11 的代码

```
// 项目 11——压电声音报警器

float sinVal;
int toneVal;

void setup() {
    pinMode(8, OUTPUT);
}

void loop() {
    for (int x=0; x<180; x++) {
        //当使用 sin 函数时转化角度到弧度
        sinVal = (sin(x*(3.1412/180)));
        //用 sin 函数值产生声音频率
        toneVal = 2000+(int(sinVal*1000));
        tone(8, toneVal);
        delay(2);
    }
}
```

```
}
```

上传代码之后，将有一点小小的延时，之后压电扬声器开始发出声音。如果每一步骤都正确地完成了，你会听到一高一低的警报声，如同汽车警报声一样。项目 11 的代码与项目 7 的代码几乎完全一样，让我们看看程序是如何工作的。

代码回顾

首先，定义两个变量：

```
float sinVal;
int toneVal;
```

`sinVal` 浮点变量存储正弦值，使声音产生高低变化，其原理如同项目 7 中的闪烁灯一样。`toneVal` 从 `sinVal` 变量中获得数值，并把它转化成所需要的频率。

在 `setup` 函数里，设置数字引脚 8 的模式为输出模式：

```
void setup(){
    pinMode(8, OUTPUT);
}
```

在主循环中设置一个从 0 到 179 的 `for` 循环，保证正弦值不会产生负值（像项目 7 中做的那样）：

```
for (int x=0; x<180; x++) {
```

将 `x` 值转换成弧度（又与项目 7 相同）：

```
sinVal = (sin(x*(3.1412/180)));
```

之后，将这个值转变成相应的报警声音的频率：

```
toneVal = 2000+(int(sinVal*1000));
```

把 `sinVal` 乘以 1000，转化该结果为整型数后再加上 2000 赋值给变量 `toneVal`，现在 `toneVal` 就是一个合适的按正弦波高低变化的声音的频率值。

之后，用 `tone()` 函数产生输出给压电扬声器的频率：


```
tone(8, toneVal);
```

tone()函数需要两个或三个参数，如下：

```
tone(pin, frequency)
tone(pin, frequency, duration)
```

pin 是连接到压电扬声器的数字引脚，frequency 是以 Hz 为单位的频率值。这里有一个可选的 duration 参数，它是以毫秒为单位的表示声音长度的参数。如果没有指定 duration，声音将一直持续直到输出一个不同的声音或使用 noTone(pin)函数结束在指定引脚上产生的声音。

最后，在频率变化的中间延时 2ms，保证正弦波以需要的速度变化：

```
delay(2);
```

你可能有疑问，为什么不把 2ms 放在 tone()函数的 duration 参数中，像下面这样：

```
tone(8, toneVal, 2);
```

这是因为 for 循环运行时间很短，它将在少于 2ms 的时间内改变频率值，因此使用 duration 参数是没有用的。所以，把延时 2ms 放在产生声音之后，保证这个声音在 for 循环重复并再次改变声音之前持续至少 2ms。

在本书的后面当你学习如何连接传感器到 Arduino 上时你还要使用这个产生声音的原理。那时你可以当传感器设定的报警条件满足时激活一个警报声，如当一个物体接近了超声测距仪或温度过高时。

如果改变 toneVal 计算中的 2000 和 1000 或延时时间，可以产生不同的报警声音。看到自己能在 Arduino 项目中创造声音是多么有趣的一件事啊！

硬件回顾

在这个项目中有两个新元件：螺钉式接线端子和压电扬声器。因为从压电扬声器或压电片中引出的导线太细太软，不能插入面包板中，所以要使用螺钉式接线端子。螺钉式接线端子上有硬的金属引脚，使得可以把它插入面包板中。

压电扬声器或压电盘（见图 4-2）是一个由粘到金属盘上的压电陶瓷薄片组成的元件。

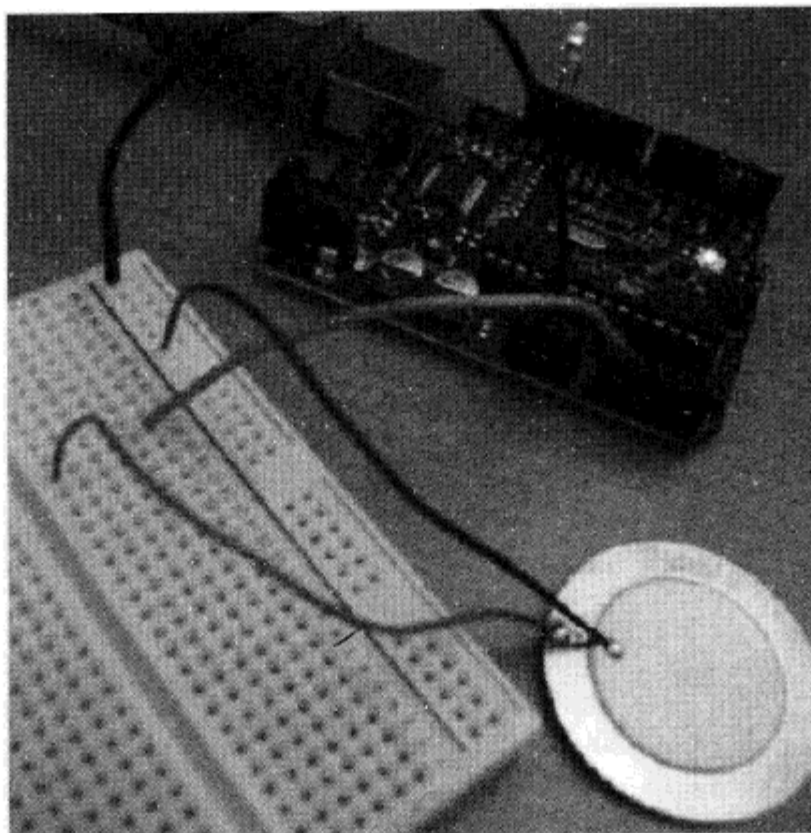


图 4-2 压电扬声器和 Arduino（图片由 Patrick H.Lauke/splintered.co.uk 友情提供）

压电材料是由晶体或压电陶瓷组成的，在它上面施加机械压力能够产生电场。这个效应非常有用，如产生或检测声音、产生高电压、产生电子频率、制作微天平或在光学装置中非常精确地聚焦等。

这个效应是可逆的。如果电场穿过压电材料，将引起材料形状的改变（一般情况下变化是 0.1%）。

为了从压电盘上产生声音，电场迅速开、关使材料变形；当压电盘鼓起来和返回原状时（像一个微型的鼓）能产生听得见的敲击声。通过改变脉冲的频率，压电盘每秒将变形几百或上千次，引起嗡嗡声。改变敲击的频率和间隔时间可产生特定的声音。

你也可以使用压电效应产生一个电场来测量运动和振动。实际上压电片可用做吉他和架子鼓的接触式扬声器。在项目 13 中将使用这个特性制造一个敲击传感器。

项目 12——压电扬声器音乐演奏

与其用压电扬声器产生一个烦人的报警声音，为什么不用它来演奏音乐呢？下面将用 Arduino 演奏“神奇龙”乐曲。电路与项目 11 相同，只需要改变代码即可。

输入代码

打开 Arduino IDE 并输入清单 4-2 中的代码。

清单 4-2 项目 12 的代码

//项目 12——压电扬声器音乐演奏

```
#define NOTE_C3      131
#define NOTE_CS3     139
#define NOTE_D3      147
#define NOTE_DS3     156
#define NOTE_E3      165
#define NOTE_F3      175
#define NOTE_FS3     185
#define NOTE_G3      196
#define NOTE_GS3     208
#define NOTE_A3      220
#define NOTE_AS3     233
#define NOTE_B3      247
#define NOTE_C4      262
#define NOTE_CS4     277
#define NOTE_D4      294
#define NOTE_DS4     311
#define NOTE_E4      330
#define NOTE_F4      349
#define NOTE_FS4     370
#define NOTE_G4      392
#define NOTE_GS4     415
#define NOTE_A4      440
#define NOTE_AS4     466
#define NOTE_B4      494

#define WHOLE 1
#define HALF 0.5
#define QUARTER 0.25
#define EIGHTH 0.125
#define SIXTEENTH 0.0625
```

```
int tune[] = { NOTE_C4, NOTE_C4, NOTE_C4, NOTE_C4, NOTE_C4, NOTE_B3, NOTE_G3,
NOTE_A3, NOTE_C4, NOTE_C4, NOTE_G3, NOTE_G3, NOTE_F3, NOTE_F3, NOTE_G3, NOTE_F3,
NOTE_E3, NOTE_G3, NOTE_C4, NOTE_C4, NOTE_C4, NOTE_C4, NOTE_A3, NOTE_B3, NOTE_C4,
```

```
NOTE_D4};
```

```
float duration[] = { EIGHTH, QUARTER+EIGHTH, SIXTEENTH, QUARTER, QUARTER, HALF,
HALF, HALF, QUARTER, QUARTER, HALF+QUARTER, QUARTER, QUARTER, QUARTER,
QUARTER+EIGHTH, EIGHTH, QUARTER, QUARTER, QUARTER, EIGHTH, EIGHTH, QUARTER,
QUARTER, QUARTER, QUARTER, HALF+QUARTER};
```

```
int length;
```

```
void setup() {
    pinMode(8, OUTPUT);
    length = sizeof(tune) / sizeof(tune[0]);
}
```

```
void loop() {
    for (int x=0; x<length; x++) {
        tone(8, tune[x]);
        delay(1500 * duration[x]);
        noTone(8);
    }
    delay(5000);
}
```

上传代码之后，稍等片刻，压电扬声器开始演奏音乐。希望你能听出它是“神奇龙”乐曲的一部分。现在让我们看一下这个项目中的新概念。

代码回顾

阅读项目 12 的代码时，你所看到的第一部分是一长串的 **define** 指令。**define** 指令是非常简单却有用的。**#define** 仅仅是定义一个值和它的替代符号，例如：

```
#define PI 3.14159265358979323846264338327950288419716939937510
```

允许你在任何计算中使用字符“PI”代替圆周率，而不用输入圆周率的 50 位有效数字。另一个例子是：

```
#define TRUE 1
#define FALSE 0
```

其意义是可以在代码中输入 **TRUE** 或者 **FALSE**，而不是 0 或 1，这使得逻辑语句非常容易阅读。

假设你在编制 LED 点阵显示器中显示图形的代码, 这个显示器的分辨率是 8×32 像素, 你可以用 `define` 指令设置显示器的高和宽如下:

```
#define DISPLAY_HEIGHT 8
#define DISPLAY_WIDTH 32
```

现在, 不管什么时候在代码中引用显示器的高和宽, 你都可以输入 `SISPLAY_HEIGHT` 和 `DISPLAY_WIDTH` 而不用写数字 8 和 32。

这样在代码中就不是简单地使用数字, 有两个主要优点。首先, 代码变得非常容易理解。当你将显示器的高和宽值用符号表示时, 符号使第三者非常容易理解其含义。第二, 如果将来换成大分辨率的显示器, 比如 16×64 像素的显示器, 你所需要做的仅仅是改变 `define` 指令处的两个值, 而不是改变也许有几百行的代码中的数字。在程序开头改变 `define` 指令中的数值, 后面的代码就可以自动使用更改后的新值。

在项目 12 里, 你创建一系列的 `define` 指令, 它代替的是从 C3 到 B4 这些产生声音所需要的频率值。你所要演奏的音乐的第一个音符是 C4, 它对应的频率是 262Hz, 这是音阶的中音 C。(不是所有定义的音符在你所演奏的音乐中都用到了, 但是我定义了所有的音符, 因为这样你可以写你自己的音乐。)

下面 5 个 `define` 指令定义音符的长度。音符按长度分为整拍、半拍、1/4 拍、1/8 拍和 1/16 拍。我们将用这些数字乘以拍的长度 (以毫秒为单位) 去获得每个音符的长度。例如, 1/4 拍音符的数字是 0.25 (1/4), 因此 0.25 乘以拍长度获得 1/4 音符:

$$1500 \times 0.25 = 375\text{ms}$$

`define` 指令也可以用来定义宏, 在以后的章节中会有许多涉及宏的地方。

接下来定义了一个叫 `tune[]` 的整型数组, 并用“神奇龙”的音调初始化该数组, 如下:

```
int tune[] = { NOTE_C4, NOTE_C4, NOTE_C4, NOTE_C4, NOTE_C4, NOTE_B3, NOTE_G3,
NOTE_A3, NOTE_C4, NOTE_C4, NOTE_G3, NOTE_G3, NOTE_F3, NOTE_F3, NOTE_G3, NOTE_F3,
NOTE_E3, NOTE_G3, NOTE_C4, NOTE_C4, NOTE_C4, NOTE_C4, NOTE_A3, NOTE_B3, NOTE_C4,
NOTE_D4};
```

之后定义另外一个数组和一个浮点数, 它存储每个音符演奏时所持续的时间:

```
float duration[] = { EIGHTH, QUARTER+EIGHTH, SIXTEENTH, QUARTER, QUARTER, HALF,
HALF, HALF, QUARTER, QUARTER, HALF+QUARTER, QUARTER, QUARTER, QUARTER,
QUARTER+EIGHTH, EIGHTH, QUARTER, QUARTER, QUARTER, EIGHTH, EIGHTH, QUARTER,
QUARTER, QUARTER, QUARTER, HALF+QUARTER};
```

通过观察这些数组可以看到，使用 `define` 指令定义音调和音调长度比直接用数字表示更容易阅读和理解。之后定义一个叫 `length` 的整型变量：

```
int length;
```

它用来计算和存储数组的长度（如音乐中的音符数）。

在 `setup` 函数中设置引脚 8 为输出模式：

```
pinMode(8, OUTPUT);
```

之后，使用 `sizeof()` 函数获得数组的音符数。用数组中的音符数初始化整型数 `length`：

```
length = sizeof(tune) / sizeof(tune[0]);
```

`sizeof` 函数返回传递给它的参数的字节数。在 `Arduino` 中，一个整数由两个字节组成，一个字节由 8 个比特组成（这涉及二进制数，在本项目中，你不需要着急弄清比特和字节的含义，在本书的后面还会遇到它们，到时候再解释）。你的音调中有 26 个音符，因此 `tunes[]` 数组有 26 个元素。通过以下计算得到整个数组的大小（以字节为单位）：

```
sizeof(tune)
```

并且除以一个元素的字节数

```
sizeof(tune[0])
```

在这里，它相当于

```
26/2=13
```

如果你用自己的音调数组代替这个项目里的音调数组，`length` 的值则等于你的音调数组中音符的数目。

`sizeof()` 函数在计算不同类型的数据长度方面是非常有用的，尤其是需要输出代码

给另一个设备而这个设备数据类型的长度可能与 Arduino 中的不同时。

在主循环中，设置了一个 for 循环，它的循环次数为音乐中的音符数：

```
for (int x=0; x<length; x++) {
```

之后通过数字引脚 8 演奏 tune[] 数组中的下一个音符：

```
tone(8, tune[x]);
```

最后延时适当的时间，让音乐继续演奏：

```
delay(1500 * duration[x]);
```

延时时间是 1500ms 乘以音符的长度（1/4 拍是 0.25，1/8 拍是 0.125 等）。

在演奏下一个声符之前，停止引脚 8 上产生的声音：

```
noTone(8);
```

这可以保证两个同样的音符连续演奏时能区分每一个。没有 noTone() 函数，两个相同的音符就会混合成一个拉长的音符。

最后，for 循环完成之后，在再次演奏音乐之前延时 5s：

```
delay(5000);
```

为了产生这首音乐的音符，我在网站上找到一些公开的“神奇龙”乐谱，并将音符输入 tune[] 数组中，音符长度存入 duration[] 数组中。注意，我已添加音符长度以获得四分音符（例如，1/4 音符+1/8 音符）。同样的，你可以产生任何你想要的音乐。

如果你想加快或放慢曲调节奏，只要改变延时函数中 1500 这个数值，将它变大或变小即可。

你也可以把电路中的压电盘换成一个扬声器或耳机，只需串联一个电阻，确保扬声器中的最大电流不过流。

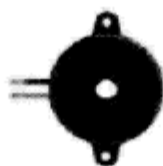
也可将压电盘用于其他类型的项目——压电盘能够在它受到震动或敲击时产生电流。利用这个特性，我们将在项目 13 中制作一个震动传感器。

项目 13——压电震动传感器

当电流穿过压电盘里的压电陶瓷材料时，压电盘开始工作。电流使它变形，因此压电盘发出声音（敲击声）。压电盘也可以反过来工作。当压电盘被敲击或震动时，作用在材料上的力产生电流，可以使用 Arduino 读出这个电流。利用这个原理你现在马上就要做一个震动传感器。

需要的元件

压电扬声器（或压电片）



两路螺钉式接线端子



5mm LED（任何颜色）



1MΩ电阻



把元件连接起来

首先，通过把 USB 线拔下来确保 Arduino 没有上电。之后连接元件，形成如图 4-3 所示的电路。注意，在这个项目中，压电盘工作得比压电扬声器更好。

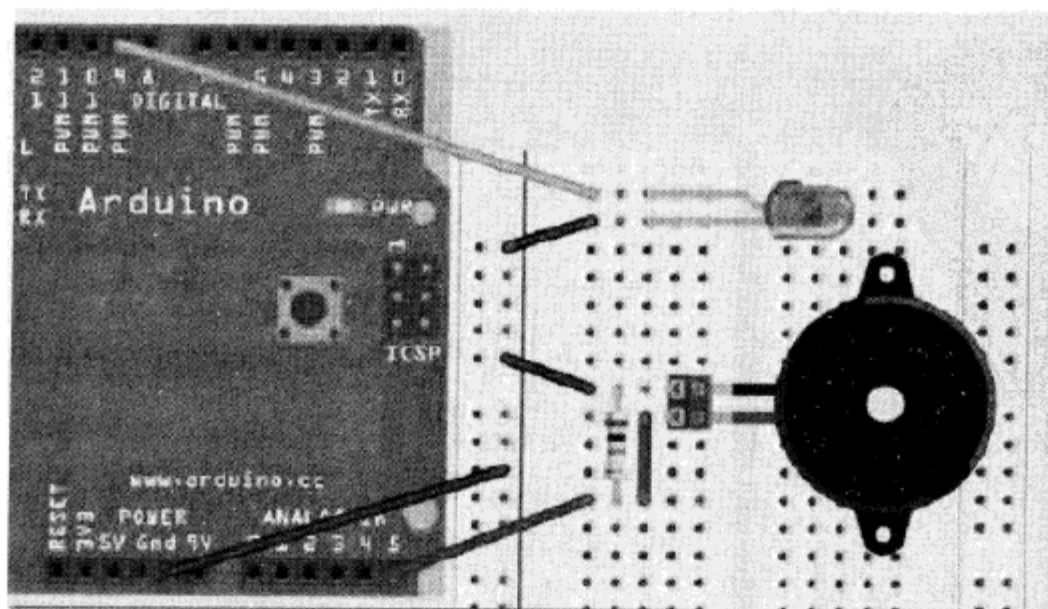


图 4-3 项目 13 电路——压电震动传感器（参见彩色版插图）

输入代码

打开 Arduino IDE 并输入清单 4-3 中的代码。

清单 4-3 项目 13 的代码

```
//项目 13——震动传感器

int ledPin = 9; //LED 接在引脚 9 上
int piezoPin = 5; //压电盘接在引脚 5 上
int threshold = 120; //传感器产生动作的阈值
int sensorValue = 0; //存储从传感器读出的值的变量
float ledValue = 0; //LED 的亮度

void setup() {
    pinMode(ledPin, OUTPUT); //设置引脚为输出模式
    //闪烁 LED 两次显示程序已经开始
    digitalWrite(ledPin, HIGH); delay(150); digitalWrite(ledPin, LOW);
    delay(150);
    digitalWrite(ledPin, HIGH); delay(150); digitalWrite(ledPin, LOW);
    delay(150);
}

void loop() {
    sensorValue = analogRead(piezoPin); //从传感器读值
    if (sensorValue >= threshold) { //如果检测到敲击，设置亮度为最大值
        ledValue = 255;
    }
    analogWrite(ledPin, int(ledValue)); //写亮度值到 LED
    ledValue = ledValue - 0.05; //慢慢使 LED 变暗
    if (ledValue <= 0) { ledValue = 0; } //确保值没有低于 0
}
```

上传代码之后，LED 将快速闪烁两下，表明程序已经开始。现在可以敲击传感器（首先把它平放在一个平面上），或者用手指挤压它。每一次当 Arduino 检测到敲击或挤压时，LED 变亮之后将慢慢变暗直至关闭（注意代码中阈值是根据本项目我所使用的特定压电片设置的。你可以根据你项目中所使用的压电片的形式和尺寸设定合适的值。阈值低会更敏感，阈值高敏感性会差一点）。

代码回顾

本项目中没有涉及新的编写代码的知识，但是我要说明一下本项目的代码是如何工作的。

首先，给程序设置必要的变量，代码后有注释：

```
int ledPin = 9; //LED 接在引脚 9 上
int piezoPin = 5; //压电盘接在引脚 5 上
int threshold = 120; //传感器产生动作的阈值
int sensorValue = 0; //存储从传感器读出的值的变量
float ledValue = 0; //LED 的亮度
```

在 `setup` 函数中，`ledPin` 设置为输出，如前所述，LED 快速闪烁两次作为程序开始工作的可视标志：

```
void setup() {
    pinMode(ledPin, OUTPUT);
    digitalWrite(ledPin, HIGH); delay(150);
    digitalWrite(ledPin, LOW); delay(150);
    digitalWrite(ledPin, HIGH); delay(150);
    digitalWrite(ledPin, LOW); delay(150);
}
```

在主循环中，首先从模拟引脚 5 读一个模拟值。模拟引脚 5 是连接压电盘的那个引脚。

```
sensorValue = analogRead(piezoPin);
```

之后，代码检查这个模拟值是否大于或等于已设定的阈值，例如，它是否被敲击或挤压（如果阈值设定为一个非常低的值，你会看到压电盘变得非常敏感）。如果满足条件，设置 `ledValue` 为 255，这是数字 PWM 引脚 9 输出的最大电压值：

```
if (sensorValue >= threshold) {
    ledValue = 255;
}
```

之后将该值写入 PWM 引脚 9。因为 `ledValue` 是一个浮点值，需要将它转换成整型。作为模拟量写函数，`analogWrite` 只能接受整型而不是浮点型数值。

```
analogWrite(ledPin, int(ledValue));
```

之后，将 ledValue 的值减去浮点数 0.05：

```
ledValue = ledValue - 0.05;
```

因为希望 LED 慢慢变暗，因此这里用浮点型而不是整型来存储 LED 的亮度值。这样，可以用较小的增量逐渐减小它的值（这里是 0.05），它通过主程序循环逐渐减小，直至变成 0。如果希望 LED 变暗的速度放慢或加快，可减小或增大这个增量。

最后，不希望 ledValue 的值小于零，因为数字 PWM 引脚 9 只能输出 0~255 之间的值，因此需要检查它是否小于等于 0，如果是，将它置零：

```
if(ledValue <= 0) { ledValue = 0; }
```

然后主循环重复，每次运行时 LED 慢慢变暗直到 LED 关闭，或者检测到另一次敲击，亮度变回最大值。

现在，让我们介绍新的传感器——光敏电阻器也叫做 LDR。

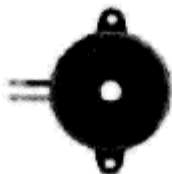
项目 14——光敏元件

这个项目中我介绍一个新的元件叫做光敏电阻或 LDR。从名字可以看出，这个器件是一个依赖于光的电阻。在黑暗环境中，光敏电阻是一个具有非常高阻值的电阻。当光子撞击到光检测器时，电阻值降低。光线越强，电阻值越低。通过从传感器中读取这个电阻值，就可以检查光线是亮还是暗，或者是介于两者之间的某个值。在这个项目中，使用 LDR 检测光，用压电扬声器给出所检测到的光亮度的声音反馈。

这套装置可用做检测门被打开时的报警器。或者你可以用它做一个特雷门琴式的电子乐器。

需要的元件

压电扬声器（或压电片）



两路螺钉式接线端子



光敏电阻器



10kΩ电阻



把元件连接起来

首先，拔掉 USB 电缆，确保 Arduino 断电。之后把元件连起来，获得如图 4-4 所示的电路图。在给 Arduino 上电前，再检查一下所有连接是否正确。

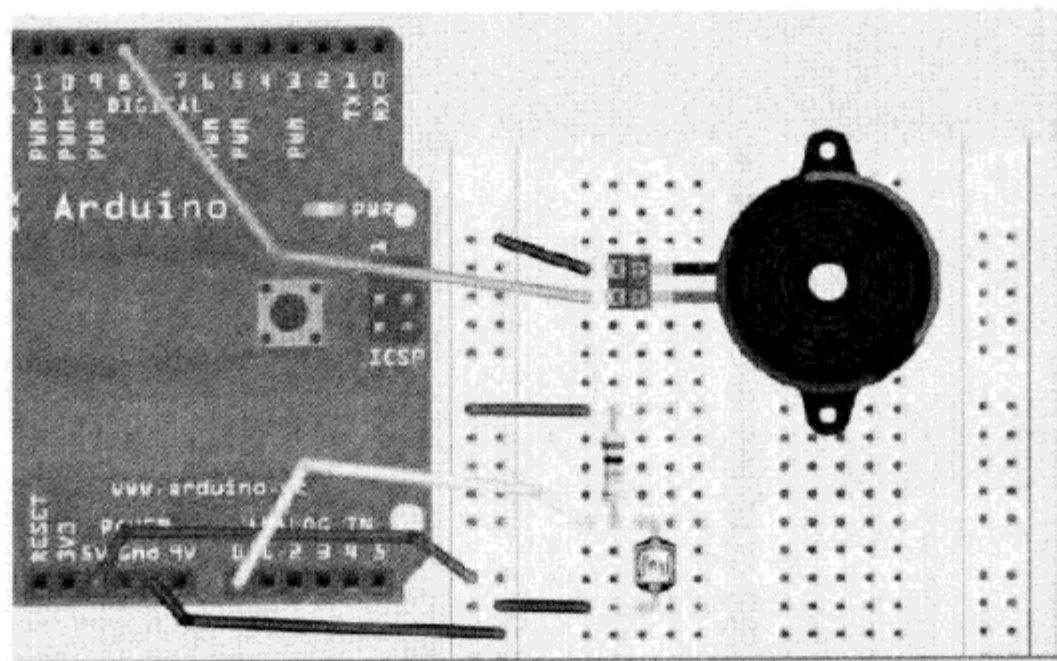


图 4-4 项目 14——光敏元件电路图（参见彩色版插图）

LDR 可以以任何方式插入电路中，因为它没有极性。10kΩ 的电阻对于我的 LDR 是适合的。但是你需要尝试不同的电阻，找到一个适合你的 LDR 的电阻。一般来说 1kΩ 到 10kΩ 之间阻值的电阻能达到目的。手边准备一个元件箱，以便随时选择不同的电阻。

输入代码

现在打开 Arduino IDE，输入清单 4-4 中所示的一段代码。

清单 4-4 项目 13 的代码

//项目 14——光敏元件

```
int piezoPin = 8; //压电盘引脚
int ldrPin = 0;   //LDR 模拟量引脚 0
int ldrValue = 0; //从 LDR 中读到的值
```



```
void setup() {
    //什么也不做
}

void loop() {
    ldrValue = analogRead(ldrPin); //从 LDR 中读数值
    tone(piezoPin, 1000); //用压电盘发出 1000Hz 的声音
    delay(25); //等上一小会儿
    noTone(piezoPin); //停止声音
    delay(ldrValue); //等上 ldrValue 表示的毫秒数
}
```

当把代码上载到 Arduino 中后，Arduino 发出短促的提示音。如果 LDR 在阴影中，提示音之间的间隔时间较长。如果光线照在 LDR 上，提示音的间隔时间较短，类似盖革计数器的效果。焊接一个长导线到 LDR 上是非常有用的。它使你可以把面包板和 Arduino 放在桌子上，把 LDR 放在阴暗或明亮处。另外一种方法是用手电筒的光打在传感器上或在传感器周围来回移动。

项目 14 的代码是非常简单的，在没有任何提示的情况下你也能看出它是如何工作的。我将告诉你 LDR 的工作原理及为什么附加电阻是十分重要的。

硬件回顾

这个项目的一个新概念是光敏电阻器（LDR），也叫 CdS（硫化镉）或光电阻器。LDR 有不同的形状和尺寸（参见图 4-5）和不同的阻值范围。

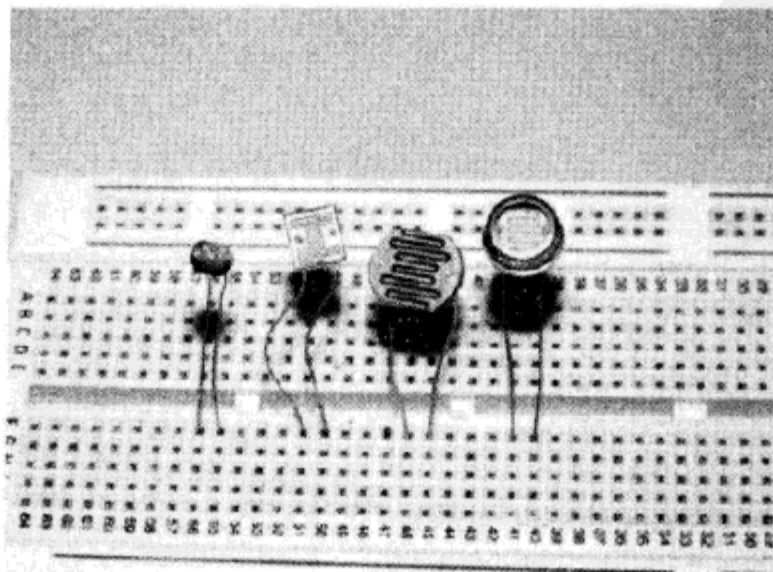


图 4-5 不同类型的 LDR（图片来自 cultured_society 2nd）

每个 LDR 的引脚接一个电极。在深色材料上，电极之间的一串弯弯曲曲的线是光导材料。这个元件用塑料或玻璃封装。光线照射到光导材料上，它的电阻变小，允许更大的电流流过两个电极之间。LDR 可以用于各种有趣的项目中。例如，让一束激光照射到 LDR 中，检测到有人挡住光柱时，触发报警器或按动相机的快门。

项目中另外一个新的概念是分压器（也叫电位分配器）。分压器是用电阻构成的。使用两个电阻串联，从其中一个取出电压，这样可以减小进入电路的电压。在这个例子里，用一个合适阻值的电阻（10kΩ或差不多大小的）和一个 LDR 形式的可变电阻组成分压器。让我们看一个典型的使用电阻的分压电路，看看它是如何工作的。图 4-6 显示了一个使用两个电阻的分压电路。

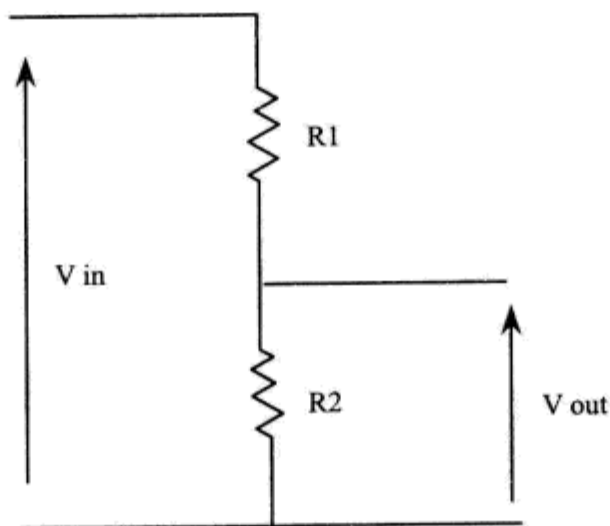


图 4-6 分压器

输入电压（ V_{in} ）连在两个电阻上。当测量通过一个电阻的电压（ V_{out} ）时，电压将小于输入电压（分压）。计算 R_2 两端的 V_{out} 电压公式如下：

$$V_{out} = \frac{R_2}{R_2 + R_1} \times V_{in}$$

因此，如果 R_1 和 R_2 的电阻值是 100Ω （ $0.1k\Omega$ ）， V_{in} 电压为 $5V$ ，公式是：

$$\frac{0.1}{0.1 + 0.1} \times 5 = 2.5 V$$

让我们用 470Ω 电阻再计算一遍：

$$\frac{0.47}{0.47 + 0.47} \times 5 = 2.5 V$$

再次得到 $2.5V$ ，这说明阻值不重要，重要的是它们之间的比值。让我们试一个 $1k\Omega$ 和

500Ω的电阻:

$$\frac{0.5}{0.5+1} \times 5 = 1.66 \text{ V}$$

下面的电阻值是上面电阻值的一半, 得到 1.66V, 是输入电压的 1/3。如果下面的电阻值是上面的两倍, 即 2kΩ:

$$\frac{2}{2+1} \times 5 = 3.33 \text{ V}$$

得到 2/3 的输入电压。因此, 将这个方法应用到 LDR, 假定 LDR 电阻在暗处的阻值是 10kΩ 左右, 在强光下阻值是 1kΩ。表 4-1 显示出当电阻发生变化时电路输出电压值的大小。

表 4-1 V in 5V 时 LDR 的 V out 的值

R1	R2 (LDR)	V out	亮 度
10kΩ	100 kΩ	4.54V 黑	est
10kΩ	73 kΩ	4.39V25%	
10kΩ	45 kΩ	4.09V50%	
10kΩ	28 kΩ	3.68V75%	
10kΩ	10 kΩ	2.5V 亮	ghtest

可以看到, 当亮度增加时, V out 的电压减小。因此, 在传感器上读到的电压值变小, 即提示音间歇时间变短, 导致提示音发生得更频繁。如果转换电阻和 LDR 的位置, 亮度增加时, 读出的电压将增大。哪一种方法都可以正常运行, 这只取决于你想让传感器如何工作。

小结

在第 4 章中, 学习了如何利用 Arduino 生成音乐、报警声、敲击声等。这些声音有许多有用的用途, 例如, 可以制作闹钟。通过反向使用压电扬声器, 可以测量到它所产生的电压。利用这个原理检测压电盘上的敲击或压力, 可以做一个乐器。最后, 通过使用 LDR 检测光线, 可以实现当环境光线暗到一定程度时打开夜灯的功能。

本章的主题和概念

- 什么是压电传感器, 它是如何工作的

- 如何用 `tone()` 函数发出声音
- 如何用 `noTone()` 函数停止 `tone()` 函数产生的声音
- `#define` 命令和它是如何使代码容易调试和理解的
- 用 `sizeof()` 函数获得一个数组的大小
- 什么是 LDR（光敏电阻器），它是如何工作的，如何从它中读出数值
- 分压器的概念和如何使用它们



第5章

驱动直流电机

现在是来控制直流电机的时候了。如果你要制作一个机器人或其他可动的东西，控制电机的技术是必须要学习的。驱动电机需要的电流高于 Arduino 的输出引脚所能安全提供的电流，因此需要用三极管保证有足够的电流驱动电机。在硬件电路中还要用二极管保护 Arduino。在硬件回顾中将解释它们是如何工作的。

在第一个项目中，你将学到如何用非常简单的方法控制一台电机。之后，会进一步学习使用非常流行的 L293D 电机控制芯片。在本书的后面你将学习如何使用这个芯片控制步进电机。

项目 15——简单的电机控制系统

首先，你要学习使用功率三极管、二极管、外电源供电（用于驱动电机）和一个变阻器（用于控制电机转速）来控制单向转动电机的转速。所有为大电流负载设计的 NPN 功率三极管都可以代替 TIP120 三极管。

外电源供电可以采用电池组或“插墙式”直流电源。电源必须能提供足够大的电压和电流来驱动电机。电压不能超过电机额定电压。本项目使用 5V、500mA 的直流电源，这足够驱动项目所用的 5V 直流电机。注意，如果你使用的电源的电压超过电机所能承受的最大电压，电机将受到永久性的损坏。

需要的元件

直流电机



10k Ω 变阻器

TIP120 三极管*



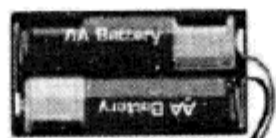
1N4001 二极管



电源插头



外电源



*其他合适的元件。

把元件连接起来

首先，通过把 Arduino 从 USB 电缆上拔下来确保它没有上电。现将以上元件连接成如图 5-1 所示的电路。在给电路上电之前检查并再次确认所有连接与图 5-1 是否完全一样是十分必要的。如果操作不慎可能会损坏某些元件或 Arduino。二极管扮演一个重要的角色，它保护 Arduino 在电路中发生 EMF（反电动势）现象时不致损坏。我将在后面解释。

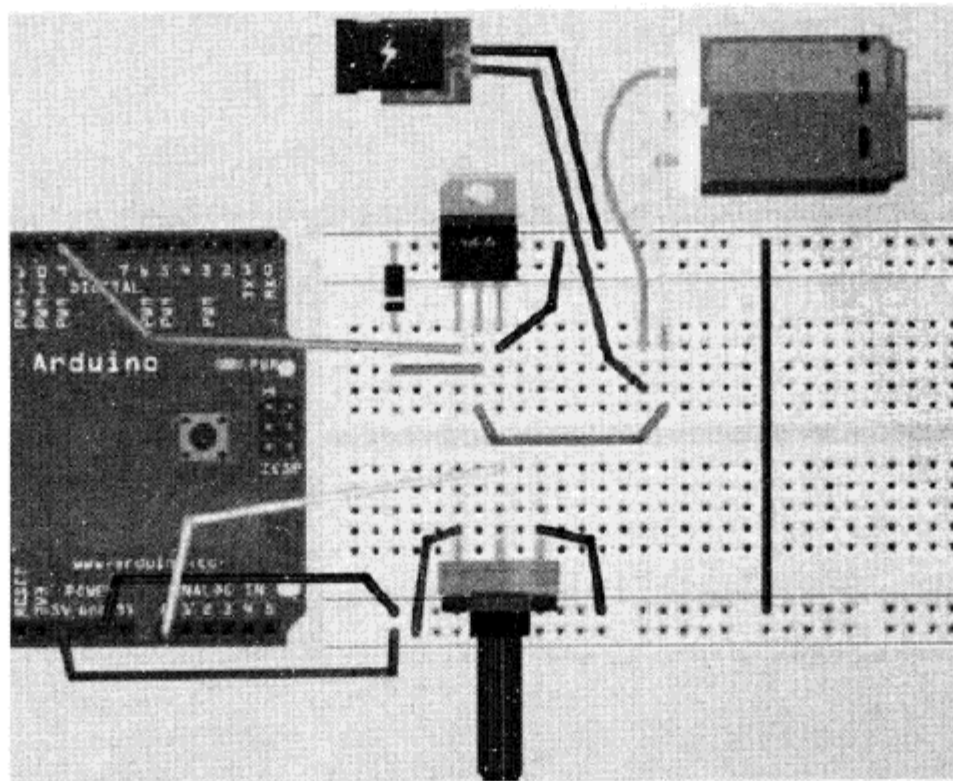


图 5-1 项目 15——简单的电机控制系统电路图（参见彩色版插图）

输入代码

打开 Arduino IDE，输入清单 5-1 中的程序代码。在上传代码之前，断开给电机供电的外部电源，确保变阻器旋钮能顺时针旋转。然后上传代码到 Arduino 中。

清单 5-1 项目 15 的代码

```
//项目 15——简单的电机控制

int potPin = 0;           //模拟引脚 0 连接到变阻器
int transistorPin = 9;    //PWM 引脚 9 连接到三极管
int potValue = 0;        //从变阻器中读出的模拟值

void setup() {
    //设置连接到三极管上的引脚模式为输出
    pinMode(transistorPin, OUTPUT);
}

void loop() {
    //读变阻器值并转化成 0 ~ 255
    potValue = analogRead(potPin) / 4;
    //使用这个值控制三极管
    analogWrite(transistorPin, potValue);
}
```

上传代码之后，接通外部供电电源。现在可以转动变阻器来控制电机的速度了。

代码回顾

首先，声明了三个变量，分别用来存储连接到变阻器上的模拟输入引脚值、连接到三极管基极上的 PWM 引脚值、从模拟引脚 0 中读出的变阻器的返回值：

```
int potPin = 0;           //模拟引脚 0 连接到变阻器
int transistorPin = 9;    //PWM 引脚 9 连接到三极管
int potValue = 0;        //从变阻器中读出的模拟值
```

在 setup() 函数中，设置连接到三极管上的引脚的模式为输出：

```
void setup() {
    //设置连接到三极管上的引脚模式为输出
    pinMode(transistorPin, OUTPUT);
}
```

在主循环中，变量 `potValue` 的值设置为从模拟引脚 0 (`potPin`) 读到的数据除以 4 的结果：

```
potValue = analogRead(potPin) / 4;
```

需要把读到的值除以 4 是因为读到的模拟量值的范围是 0 对应 0V 电压，1023 对应 5V 电压，而要写给三极管引脚的值的范围是 0~255，所以需要 4 除这个模拟引脚 0 的值（最大值是 1023），得出最大值是 255，用于设定模拟输出引脚 9（当使用 PWM 功能时需调用 `analogWrite` 函数）。

之后的代码是把 `potValue` 的值写入三极管引脚：

```
analogWrite(transistorPin, potValue);
```

换句话说，当旋转变阻器时，读入一个 0~1023 之间的变化值，之后把它转化成范围在 0~255 之间的一个值，再把这个值写到（通过 PWM 功能）数字引脚 11 上，用它改变直流电机的速度。变阻器旋钮旋到最右侧，电机停止。向左旋，电机加速，直到旋到最左端，电机获得最大速度。这个代码非常简单，没有包含任何新知识。

现在看一下项目中用到的硬件，看它们是如何工作的。

硬件回顾

电路可以分成两部分。第一部分是变阻器。它连接在 5V 电源和地之间，中间引脚连接到模拟引脚 0。当变阻器转动时，电阻发生变化，给中间引脚输入一个 0~5V 的电压，然后用 Arduino 模拟引脚 0 读出电压值。

第二部分是用于控制电机的电源，Arduino 数字引脚最大能输出 40mA（毫安）的电流。一个直流电机以最大速度运行时需要约 500mA 的电流。这明显大于 Arduino 的输出能力。如果试图用 Arduino 的引脚直接驱动电机，会对 Arduino 造成严重的永久性损坏。

因此，需要找到一种给直流电机提供更大电流的方法。这个方法是使用能给电机提供足够电力的外部电源。你可以使用 Arduino 开发板的 5V 输出电源。当连接到外部电源时，它可以提供高达 800mA 的电流。但是，Arduino 开发板比较贵，而且当把它连接到大电流设备时，如连接直流电机时，Arduino 开发板特别容易损坏。因此，为了保证板子的安全，也是为了当电机需要使用超过了 Arduino 板能提供的电压（如 9V、12V 电压）或更大的安培数的电源时，应该使用一个外部供电电源。

注意，这个项目是要控制电机的速度，因此需要一种控制电压的方法使电机加速、减速。这就是使用 TIP120 三极管的原因。

三极管

三极管本质上是一个数字开关，它也可用做功率放大器。在这个电路中，把它作为开关用。三极管的电路符号如图 5-2 所示。

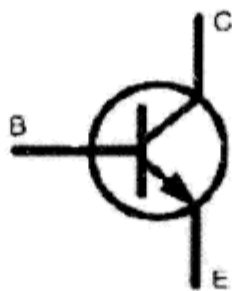


图 5-2 NPN 三极管符号

三极管有 3 个引脚：基极、集电极和发射极。在图上的标志为 B、C 和 E。在本项目电路里，通过数字引脚 9 连接 5V 到基极，集电极连接到电机的接线端子，发射极连接到地。当通过数字引脚 9 输出电压到基极时，三极管打开，允许电流从集电极和发射极之间流过，给电机供电。在电路中电机和三极管串联，通过在基极上施加一个小电流，可以控制发射极和集电极之间的大电流。

在现代电子设备里，三极管是一种重要的电子元件。很多人认为三极管是 20 世纪最伟大的发明。台式 PC 和笔记本电脑中有大约 300 万到 1 亿个三极管。

在这个例子里，使用三极管作为开关来打开或关闭高电压大电流。当电流施加到基极时，供应在集电极上的电源打开，允许电流流过集电极和发射极。当输入脉冲信号时，三极管每秒开关许多次。用集电极与发射极之间的脉冲电流控制电机速度。

电机

电机是一种电磁元件，在它两端施加电压时，电机内的线圈产生磁场。当断开电源后，磁场产生突变。这个突变的磁场产生反电动势，施加在电机线圈上。这个反电动势可能严重损坏 Arduino 开发板，这就是为什么电路中需要放置二极管来防止这种反向电流。二极管上白色环一般接地。电流从正极流向负极。如果方向接反，根本就不会产生电流。然而，如果电机产生“back EMF”（反电动势），从线圈反向送出电流，二极管将作为一个阀门阻止它这样做。因此二极管在这个电路里是用来保护 Arduino 的。

如果将直流电机直接连接一个电流表，而不连接其他元件。之后转动电机轴，你会发现电机产生了电流。这就是风力发电机的工作原理。当给电机上电时，让它旋转起来，然后，突然断开电源，电机在自身惯性下继续转动，直到停下来。在这段时间内，电机在没有任何电源供电的情况下旋转，它会产生电流。这就是前面提到的“back EMF”（反电动势）。这时，也是二极管作为一个阀门使电流不会反向流到电路中，保护其他元件不受损坏。

二极管

二极管是一个单向阀。二极管允许电流向一个方向流，而不允许反向流动。这与单向阀在水管中只允许水朝一个方向流动但不能反向流动的原理完全相同。当你使用 LED 时，已接触过二极管。LED 有极性。连接电源正极到 LED 上的长脚上，使 LED 点亮。转变 LED 引脚的接法，不但 LED 不会发光，也阻止了电流通过 LED 的引脚。二极管上靠近负极导线一边有一个白色的环。可以将这个白色环想象成一个关卡。电流流过二极管时，是从没有关卡的一侧流过的。当反向施加电压试图使电流流过白色环的一侧时，电流会被阻止。

在保护电路不承受反向电压时需要使用二极管，例如，防止在把电源接反或有反向电压时电路发生“back EMF”（反电动势）时保护电路中的元件就需要用到二极管。因此，在电路中存在由于使用者操作不当或 EMF 现象造成的电源接反的危险时，我们总是试着在电路里使用二极管以防止损坏电路。

项目 16——使用 L293D 电机驱动芯片

在前面的项目中，我们使用三极管控制电机。在这个项目里，我要用一个非常流行的电机驱动芯片，叫做 L293D。使用这个芯片的优点是可以同时控制两台电机，并且同时控制它们的转向。这个芯片也可以用来控制步进电机，就像你将在项目 28 中看到的那样。（也可以使用引脚可编程的芯片，叫做 SN754410，它可以输出更大的电流。）注意到没有，元件列表中缺了什么元件？二极管吗？没错，这个芯片内部有自己的二极管，所以不需要为本项目准备二极管。

需要的元件

直流电机

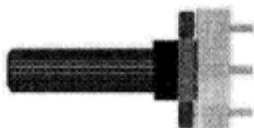


L293D 或 SN754410

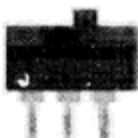
电机驱动芯片



10k Ω 变阻器



扳钮开关



10k Ω 电阻



散热片



把元件连接起来

首先通过把 Arduino 从 USB 线上拔下来确保它没有上电。现在用以上元件把它们连成如图 5-3 所示的电路。在上电之前再次全面检查电路。L293D 在使用时会变得非常热。因此，必须使用散热片给 L293D 散热。用环氧树脂把散热片粘到芯片顶部。散热片越大越好。注意，运行时 L293D 的温度足以熔化面包板的塑料或连在其上的导线。不要碰触散热片，会造成烫伤。当芯片过热时不要给电路上电，也不要碰触它。在项目中使用万用板而不使用面包板是对的，这样做可以防止芯片过热损坏面包板。

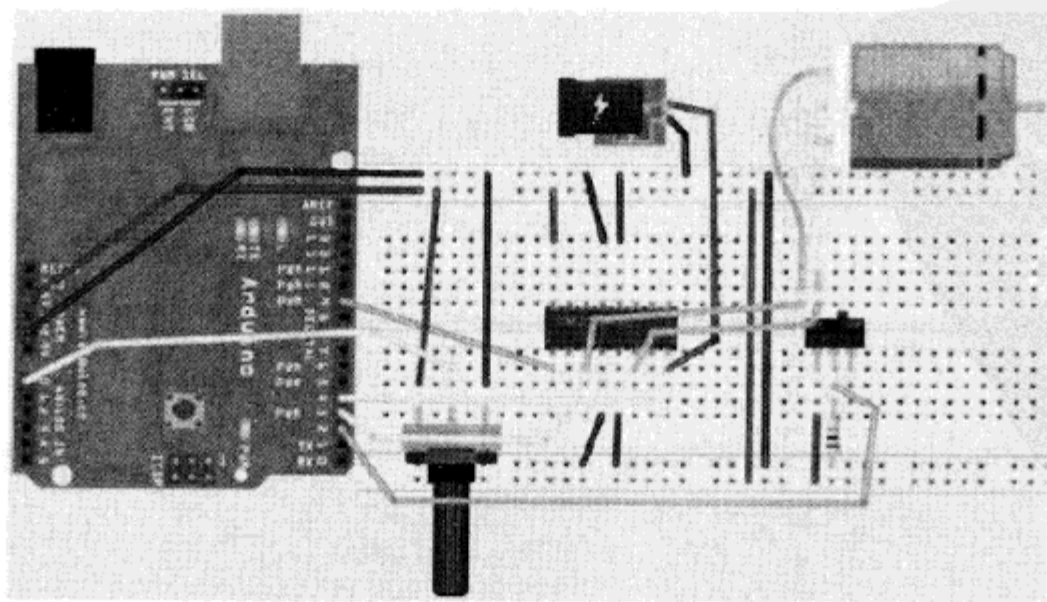


图 5-3 项目 16 电路图（参见彩色版插图）

输入代码

一旦确认电路连接正确，上传清单 5-2 中的代码。在这一步中不要连接外部电源。

清单 5-2 项目 16 的代码

```
// 项目 16——使用 L293D 电机驱动芯片
#define switchPin 2 //扳钮开关输入
#define motorPin1 3 //L293D 输入 1
#define motorPin2 4 //L293D 输入 2
#define speedPin 9 //L293D 使能引脚 1
#define potPin 0 //连接变阻器的模拟引脚
int Mspeed = 0; //变量存储当前速度值
void setup() {
  //设置扳钮开关引脚为 INPUT
  pinMode(switchPin, INPUT);

  //设置其他引脚为 OUTPUT
  pinMode(motorPin1, OUTPUT);
  pinMode(motorPin2, OUTPUT);
  pinMode(speedPin, OUTPUT);
}

void loop() {
  Mspeed = analogRead(potPin)/4; //从变阻器中读速度值
  analogWrite(speedPin, Mspeed); //写速度值到使能引脚
  if (digitalRead(switchPin)) { //如果扳钮开关是 HIGH，电机顺时针旋转
    digitalWrite(motorPin1, LOW); //设置 L293D 输入 1 为 LOW
    digitalWrite(motorPin2, HIGH); //设置 L293D 输入 2 为 HIGH
  }
  else { //如果扳钮开关是 LOW，电机逆时针旋转
    digitalWrite(motorPin1, HIGH); //设置 L293D 输入 1 为 HIGH
    digitalWrite(motorPin2, LOW); //设置 L293D 输入 2 为 LOW
  }
}
```

代码上传完毕，把变阻器旋钮旋到中间位置，插入外部电源。电机开始旋转，可以通过变阻器调整电机的速度。要改变电机的转向，首先要把速度设置为最小，按扳钮开关，电机将反方向旋转。还是要注意一旦上电 L293D 芯片将变得非常热。

代码回顾

这个项目的代码非常简单，首先定义要使用的 Arduino 引脚：

```
#define switchPin 2 //扳钮开关输入
#define motorPin1 3 //L293D 输入 1
#define motorPin2 4 //L293D 输入 2
#define speedPin 9 //L293D 使能引脚 1
#define potPin 0 //连接变阻器的模拟量引脚
```

之后，设置一个整数存储从变阻器中读出的速度值：

```
int Mspeed = 0; //变量存储当前速度值
```

在 `setup` 函数中，设定相关的引脚模式为输入或者输出：

```
pinMode(switchPin, INPUT);

pinMode(motorPin1, OUTPUT);
pinMode(motorPin2, OUTPUT);
pinMode(speedPin, OUTPUT);
```

在主循环中，首先从连接到模拟引脚 0 上的变阻器读值，并把它存储在 `Mspeed` 中：

```
Mspeed = analogRead(potPin)/4; //从变阻器中读速度值
```

之后给 PWM 引脚 9 设置相应的速度 PWM 值：

```
analogWrite(speedPin, Mspeed); //写速度值到使能引脚
```

接着用 `if` 语句检查从扳钮开关引脚中读出的值是高还是低。如果是高，将 L293D 上的输出引脚 1 设置为 LOW，输出 2 设置为 HIGH。这等于输出 2 为正电压，输出 1 为地，使电机向一个方向旋转：

```
if (digitalRead(switchPin)) { //如果扳钮开关是 HIGH，电机顺时针旋转
    digitalWrite(motorPin1, LOW); //设置 L293D 输入 1 为 LOW
    digitalWrite(motorPin2, HIGH); //设置 L293D 输入 2 为 HIGH
}
```

如果扳钮开关引脚是 LOW，输出 1 设为 HIGH，输出 2 设为 LOW，电机向反方向

旋转：

```
else { //如果按钮开关引脚是 LOW，逆时针转动电机
    digitalWrite(motorPin1, HIGH); //设置 L293D 输入 1 为 HIGH
    digitalWrite(motorPin2, LOW); //设置 L293D 输入 2 为 LOW
}
```

主循环重复运行，检查一个新的速度值和一个新的方向，之后，设置相应的速度和方向引脚。就像你看到的那样，使用电机驱动 IC 没有开始想象得那么难。事实上，它提供了很多方便。如果不使用电机驱动芯片实现如上功能，电路和代码会变得非常复杂。不要被 IC 吓倒，仔细阅读它的说明书，就能够发现它的秘密。现在看一下本项目所介绍的新元件是如何工作的。

硬件回顾

项目 16 中的新元件是电机驱动芯片，它可以是 L293D 也可以是 SN754410，这取决于你的选择（还有其他的芯片可用，在互联网上搜索一下，可以找到其他兼容的电机驱动芯片）。

L293D 也叫双 H 桥。H 桥是有用而简单的电子概念（见图 5-4）。

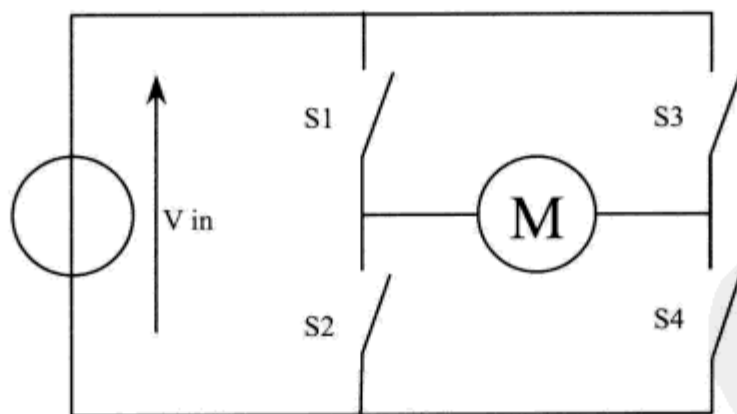


图 5-4 由开关组成的 H 桥（由 Cyril Buttay 制图）

在图 5-4 中，电机连接 4 个开关，这种情况叫 H 桥，因为它们的组合像字母 H，负载在桥的中间。现在看一下图 5-5。

在图 5-5 的左上、右下开关闭合时，电流从左向右流过电机，电机将正向旋转。如果把这两个开关断开，闭合右上和左下开关，电流以相反的方向流过电机，因此电机向相反方向旋转。这就是 H 桥的工作原理。电机驱动芯片由两个 H 桥组成，不过它用的不是开关，而是三极管，就像项目 15 中的三极管被用做开关一样。H 桥内的三极管设置方式如图 5-5

所示，通过控制三极管开关使电机能够双向转动。这种芯片叫做双 H 桥，因为它的内部有两个 H 桥，因此可同时控制两台电机的速度和转向。

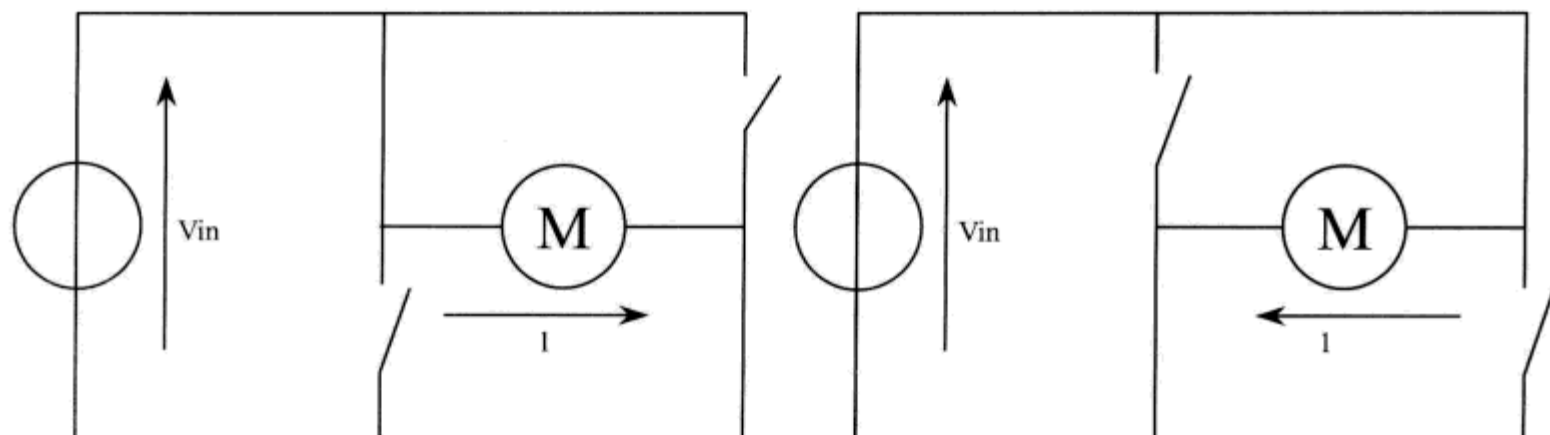


图 5-5 用 H 桥改变电机转动方向（由 Cyril Buttay 制图）

当然你也可以使用三极管、二极管制作自己的 H 桥，实现与 L293D 同样的功能。L293D 的好处是体积小，它把所有的三极管和二极管封装到一个小集成电路里了。

练 习

现在尝试以下练习。

练习 1：这次不是驱动一台电机，尝试驱动两台直流电机，用两个扳钮开关和两个变阻器分别控制两个电机的转动方向和速度。图 5-6 显示了芯片的引脚。

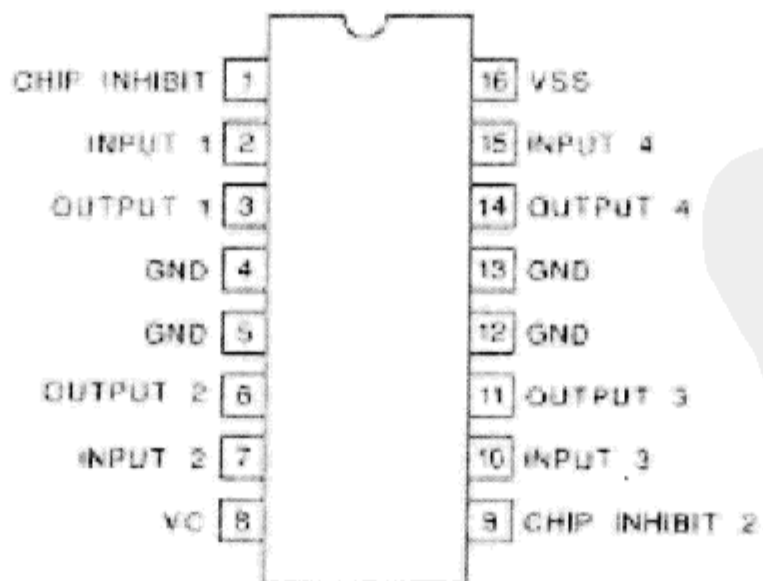


图 5-6 L293D（或 SN754410）的引脚

小结

第5章告诉你在项目中如何使用电机。同时，也介绍了关于三极管和二极管的重要概念。以后在项目中要大量使用这些元件，特别是在所控制的设备需要输入高于 Arduino 能提供的最高电压或需要大电流的情况下。

本章也介绍了如何组成一个 H 桥和如何用它改变电机的转向。同时，教你使用流行的 L293D 电机控制芯片。你将在本书的后面接触另一种形式的电机时继续使用这个芯片。如果你想制造一个使用 Arduino 控制的机器人或无线电小车，掌握驱动电机的技巧是十分重要的。

本章的主题和概念

- 使用三极管给高电压、大电流元件供电
- 使用 PWM 控制电机的速度
- 使用 Arduino 数字引脚输出电流
- 高电压、大电流元件使用外部电源是必要的
- 三极管的概念和它们是如何工作的
- 电机以及它们是如何工作的
- 使用二极管防止出现 EMF（反电动势）时损坏元件
- 二极管是如何工作的
- 电机也可以做发电机用
- 如何用 L293D 电机驱动芯片驱动电机
- 为什么用散热片给 IC 散热是必要的
- H 桥的概念及如何用它改变电机的转向

第6章

二进制计数器

现在我们回到控制 LED 的项目，但是这次不直接从 Arduino 驱动 LED，而是使用叫做移位寄存器的小芯片来驱动 LED。这个 IC（集成电路）使你只用 3 个 Arduino 引脚就能控制 8 个独立的 LED。但是我们还能控制更多的 LED：在项目 18 中，仅使用 3 个 Arduino 引脚控制了整整 16 个 LED。

为了说明使用这些芯片的方法，我要演示两个二进制计数器项目，首先只使用一个移位寄存器芯片，之后拓展到把两个移位寄存器芯片串联（将在第 18 个项目中学习串联的概念）起来。第 6 章要深入学习相关知识，因此在学习新知识之前，你要稍微整理一下思路。

项目 17——移位寄存器 8 位二进制计数器

在这个项目中，我们将要使用移位寄存器驱动 LED 进行二进制计数（后面会解释二进制的概念）。具体地说，我们仅使用 Arduino 的 3 个输出引脚驱动 8 个独立的 LED。

需要的元件

1 个 74HC595 移位寄存器芯片



8 个 220Ω电阻*



8 个 5mm LED



*其他合适的元件。

把元件连接起来

仔细地按图 6-1 连线。连接 3.3V 电源到面包板的顶部，地线连到底部。在芯片一端有一个小凹坑，把凹坑朝向左侧。这时芯片引脚 1 在凹坑的下边，引脚 8 在芯片右端下侧，引脚 9 在芯片顶部右侧，引脚 16 在芯片顶部左侧。

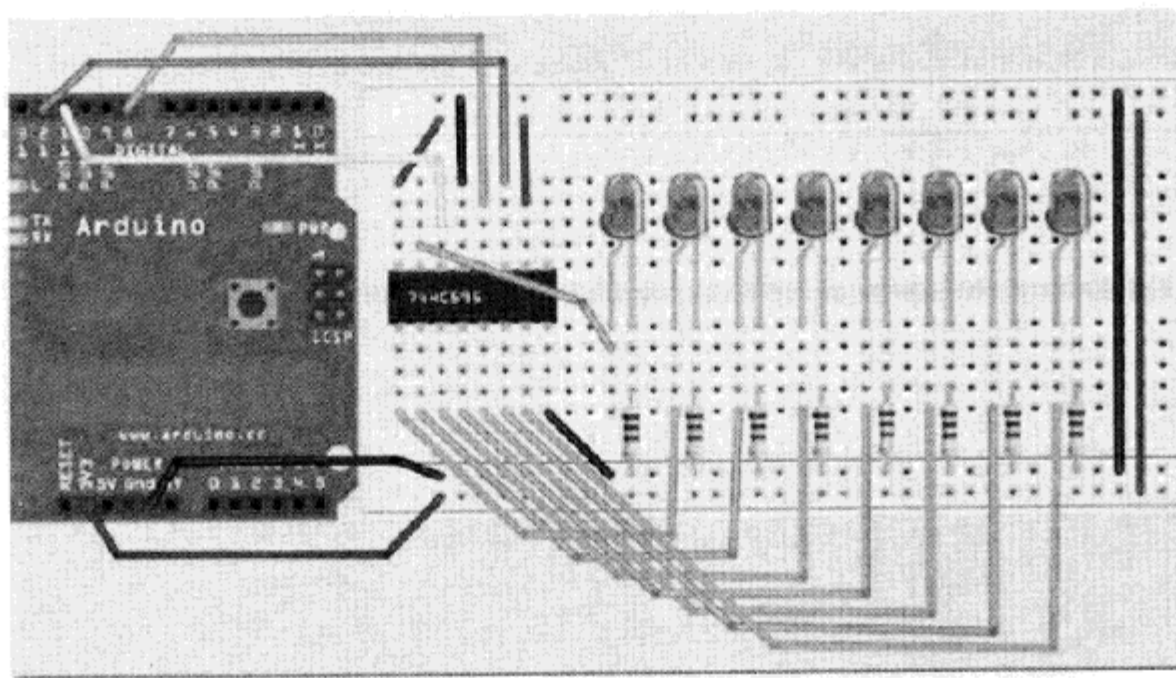


图 6-1 项目 17——用移位寄存器实现八位二进制计数器的电路图（参见彩色版插图）

现在用导线把 3.3V 电源与 74HC595 芯片的引脚 10 和 16 连起来，把地与 74HC595 芯片的引脚 8 和 13 连起来。然后用一根导线把 Arduino 引脚 8 和 74HC595 芯片的引脚 12 连起来，再用一根线把 Arduino 数字引脚 12 和 74HC595 芯片的引脚 14 连接起来，最后把 Arduino 数字引脚 11 和 74HC595 芯片的引脚 11 连起来。

8 个 LED 的负极分别通过一个 220Ω 的电阻与地连接起来。之后将 LED1 的正极连接到芯片引脚 15 上，LED2 到 LED8 的正极依次连接到芯片的引脚 1 到引脚 7 上。

所有元件连接完毕之后，再次检查连线是否正确。特别是要检查 IC 和 LED 的极性连接是否正确。

输入代码

输入清单 6-1 的代码，之后上传到 Arduino 板上，运行代码。你会看到每个 LED 独立地点亮和熄灭，它们每秒变化一次，以二进制方式从 0 加到 255，之后再重新开始。

清单 6-1 项目 17 的代码

//项目 17

```
int latchPin = 8; //Arduino 连接到 74HC595 的引脚 12 (Latch)
int clockPin = 12; //Arduino 连接到 74HC595 的引脚 11 (Clock)
int dataPin = 11; //Arduino 连接到 74HC595 的引脚 14 (Data)
void setup() {
    //设置引脚为输出模式
    pinMode(latchPin, OUTPUT);
    pinMode(clockPin, OUTPUT);
    pinMode(dataPin, OUTPUT);
}

void loop() {
    //从 0 到 255 计数
    for (int i = 0; i < 256; i++) {
        //设置 latchPin 引脚为 LOW, 允许数据输入芯片
        digitalWrite(latchPin, LOW);
        shiftOut(i);
        //设置 latchPin 引脚为 HIGH, 锁存数据并送出数据
        digitalWrite(latchPin, HIGH);
        delay(1000);
    }
}

void shiftOut(byte dataOut) {
    //在时钟上升沿送出 8 位数据
    boolean pinState;
    digitalWrite(dataPin, LOW); //清除移位寄存器, 为送数据做准备
    digitalWrite(clockPin, LOW);

    for (int i=0; i<=7; i++) { //送出数据的每一位
        digitalWrite(clockPin, LOW); //在输出数据前设置 clockPin 引脚为 LOW

        //如果 dataOut 与位掩码进行逻辑或运算的结果是 true, 设置 pinState 为 HIGH
        if ( dataOut & (1<<i) ) {
            pinState = HIGH;
        }
        else {
            pinState = LOW;
        }
    }
}
```

```

        //根据 pinState 设置 dataPin 为 HIGH 或 LOW
        digitalWrite(dataPin, pinState); //在时钟上升沿送出数据
        digitalWrite(clockPin, HIGH);
    }
    digitalWrite(clockPin, LOW); //停止移位输出数据
}

```

二进制数制

在剖析项目 17 的代码和硬件之前,让我们先看一下二进制数制。理解二进制数制对于给微控制器编程是十分必要的。

人类使用以 10 为基数的十进制数制系统,是因为我们手上有 10 根手指。计算机没有手指,因此对计算机来说最好的办法就是使用相当于手指的东西,这就是状态开或关(1 或 0)。一个逻辑设备,如计算机,能检查一个电压是存在(1)还是不存在(0),因此用二进制或基数为 2 的数字系统。这种数字系统能很容易地在一个电路中用高或低电压状态来实现。

我们所用的数字系统,基数是 10,有 10 个数字,范围从 0 到 9。当我们数到 9 以后的下一个数时,这个数返回为 0,但是在它的左侧的十位上加 1。一旦十位到了 9,再加 1 将使十位清零,但是十位左侧的百位上加 1,依此类推:

```

000, 001, 002, 003, 004, 005, 006, 007, 008, 009
010, 011, 012, 013, 014, 015, 016, 017, 018, 019
020, 021, 022, 023...

```

在二进制系统中,与此完全相同,只是最大数为 1,因此给 1 加 1,这个数位清零,而左边进位加 1:

```

000, 001
010, 011
100, 101...

```

表 6-1 表示一个 8 位二进制数(或一个字节)。

表 6-1 一个 8 位二进制数

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1
0	1	0	0	1	0	1	1

表 6-1 中的数字用二进制表示是 1001011，用十进制表示是 75。

它是这样计算出来的：

$$1 \times 1 = 1$$

$$1 \times 2 = 2$$

$$1 \times 8 = 8$$

$$1 \times 64 = 64$$

把这些都加起来，得到 75，表 6-2 显示了其他一些例子。

表 6-2 其他一些例子

Dec	2^7 128	2^6 64	2^5 32	2^4 16	2^3 8	2^2 4	2^1 2	2^0 1
75	0	1	0	0	1	0	1	1
1	0	0	0	0	0	0	0	1
2	0	0	0	0	0	0	1	0
3	0	0	0	0	0	0	1	1
4	0	0	0	0	0	1	0	0
12	0	0	0	0	1	1	0	0
27	0	0	0	1	1	0	1	1
100	0	1	1	0	0	1	0	0
127	0	1	1	1	1	1	1	1
255	1	1	1	1	1	1	1	1

……依此类推

提示 你可以使用 Google 将十进制数转换成二进制数，反之亦然。

因此要将十进制数 171 转换成二进制数，可输入 **171 in binary** 到 Google 搜索框，返回 **171=0b10101011**。

开头的 0b 表示这个数是一个二进制数而不是十进制数，所以答案是 10101011。转换二进制数到十进制数，输入数字 **0b10101011** 到搜索框，返回 **0b1001100=204**。

现在你已经理解了二进制的概念，让我们在看代码之前先回顾一下硬件。

硬件回顾

该项目使用了一个移位寄存器，具体地说是 74HC595 芯片。这种移位寄存器是一种带

输出锁存，具有 8 位串行输入、串行或并行输出的移位寄存器。这意味着可以用串行方式输入移位寄存器数据，用并行方式输出数据。串行的意思是每个动作只操作一位数据，并行的意思是每个动作同时操作多位（在这里是 8 位）数据。

当 74HC595 芯片的 LATCH 引脚设置为 LOW 时，数据输入（也就是允许数据进入芯片），当 LATCH 引脚设置为 HIGH 时，数据输出。因此给移位寄存器输入数据时（以 1 或 0 的方式），每个动作只输入一位数据，然后一个输出动作同时输出 8 位数据。当输入下一位时，已经输入的二进制数据一同向左移动。如果第 9 位数据在 LATCH 设为 HIGH 之前输入，则第一个输入的数据将超出输入序列的最左端而永久丢失。

移位寄存器一般用来实现从串行到并行数据的转换。在这种情况下，输出的数据是 1 或 0（也就是 0V 或 3.3V），因此可以用它来开关一组中的 8 个 LED。

这个项目中的移位寄存器只需要使用 Arduino 的 3 个输出引脚。

Arduino 的输出和 595 的输入见表 6-3。

表 6-3 使用的引脚

Arduino 引脚	595 引脚	描 述
8	12	存储寄存器时钟输入
11	14	串行数据输入
12	11	移位寄存器时钟输入

74HC595 芯片引脚 12 为时钟引脚，引脚 14 为数据引脚，引脚 11 为锁存器引脚。

可以把锁存器想象成一个门，当门的设置是低（LOW）时它允许数据进入 595 中，不允许 595 中的数据输出，但是数据可以输入。当门的设置是高（HIGH）时，不允许数据输入，但是移位寄存器内的数据已释放到 8 个引脚上（QA-QH 或 Q0-Q7，引脚号以说明书为准；见图 6-2）。Clock 只是 0、1 脉冲。数据引脚是指从 Arduino 发送数据到 595 的引脚。

为了使用移位寄存器，Latch 引脚和 Clock 引脚必须设置为 LOW。Latch 引脚必须保持为 LOW，直到 8 位全部设置完毕。设置 Latch 引脚为 LOW 可允许数据进入寄存器（寄存器只是 IC 内部存储 1 或 0 的地方）。在数据引脚出现 HIGH 或 LOW 信号之后，设置 Clock 引脚为 HIGH。设置 Clock 引脚为 HIGH 可以把出现在数据引脚上的数据存入寄存器。做好这些之后，再次设置 Clock 为 LOW，将第二位数据送入数字引脚。重复以上动作 8 次，可将 8 位数据输入 595。之后 Latch 引脚变为 HIGH，从寄存器传送数据到移位寄存器，并通过引脚 Q0 到 Q7 输出（引脚 15 和引脚 1 到 7）。

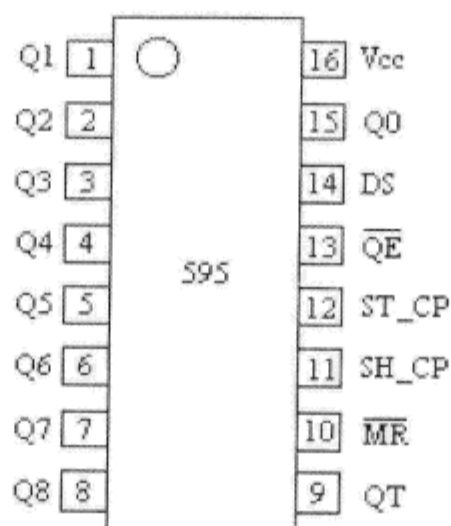


图 6-2 595 芯片的引脚图

这些动作发生的顺序在表 6-4 中进行了描述。

表 6-4 动作序列

引 脚	状 态	说 明
Latch	LOW	Latch 引脚为 LOW 允许数据进入
Data	HIGH	数据的第一位 (1)
Clock	HIGH	Clock 引脚设置为 HIGH, 数据存储
Clock	LOW	为下一位输入做好准备
Data	HIGH	数据的第二位 (1)
Clock	HIGH	第二位存储
.....
Data	LOW	数据第 8 位 (0)
Clock	HIGH	存储数据
Clock	LOW	禁止任何新数据存储
Latch	HIGH	并行送出 8 位数

当这个程序运行时, 连接一个逻辑分析仪 (一种让你看到从数字元件中输出的 1 和 0 的设备) 到 595, 图 6-3 显示了它的输出图像。从图中可以看到二进制数 0011011 (从右往左读) 或十进制数 55 已经输入 IC。

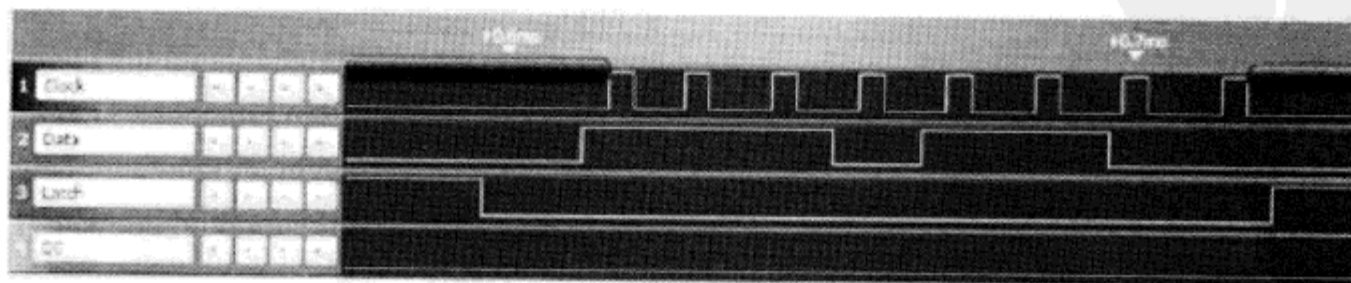


图 6-3 在逻辑分析仪上显示的 595 的输出

为了在本项目中全面描述单个移位寄存器的使用方法，将 8 个 LED 连接到寄存器的 8 个输出上。Latch 设置为 LOW 时，数据可以输入。数据被送入 Data 引脚，每次输送一位，但是只有在 Clock 引脚设置为 HIGH 时，才会存储数据。之后 Clock 引脚状态变为 LOW，准备接收数据的下一位。当所有 8 位数据输入完之后，Latch 引脚设置为 HIGH，阻止更多的数据输入，并且根据寄存器的状态设置 8 个输出引脚为 HIGH (3.3V) 或 LOW (0V)。

如果想阅读更多有关移位寄存器的说明，可查看芯片上的序列号（如 74HC595N 或 SN74HC595N 等），之后用该序列号进行搜索，找到你所用芯片的说明书。

我非常喜欢 595 芯片，因为它的用途非常广泛。当然，它也能在你的配置中增加数字输出引脚的数量。标准的 Arduino 有 19 个数字输出引脚（6 个模拟引脚也可用做数字输出引脚，引脚号从 14 到 19）。使用 8 位移位寄存器，可以把输出引脚数量扩展到 49 个（6 乘以 595 的引脚数加上一个左溢出引脚）。芯片的运行速度非常快，通常情况下是 100MHz，即在需要的情况下，大约每秒送出数据 100 万次（如果 Arduino 有能力这样做的话）。这意味着，可以通过软件给 IC 送出 PWM 信号，使得 IC 也有能力控制 LED 的亮度。

因为所输出的只是电压的开和关，所以它也可以用来开关其他低电压设备（通过使用三极管或继电器甚至可以开关高电压设备），或者送出数字信号到设备（如老式的点阵打印机或其他串行设备）。

注意，各制造厂商生产的 595 移位寄存器基本一样。你也能买到更大的移位寄存器，它们有 16 个输出引脚甚至更多。一些 IC 也在说明书中被宣传为 LED 驱动芯片，然而它只是更大的移位寄存器而已（如 STMicroelectronic 公司的 M5450 和 M5451）。

代码回顾

项目 17 的代码初看有点难，但是当你把它分成几个部分时，会发现它没有看上去那样复杂。

首先，定义并初始化 3 个要用到的引脚：

```
int latchPin = 8;
int clockPin = 12;
int dataPin = 11;
```

之后，在 `setup` 函数中，把引脚设置为输出模式：

```
pinMode(latchPin, OUTPUT);
pinMode(clockPin, OUTPUT);
pinMode(dataPin, OUTPUT);
```

主循环只是运行一个从 0 到 255 的 `for` 循环。在每一次循环过程中，`Latch` 引脚设置为 `LOW`，允许数据输入，之后调用 `shiftOut` 函数，将 `for` 循环中的 `i` 值传递给这个函数。然后 `Latch` 引脚设置为 `HIGH`，禁止数据输入，设置 8 个引脚的输出。最后在执行下一个循环之前，延时 500ms：

```
void loop() {
    //从 0 to 255 计数
    for (int i = 0; i < 256; i++) {
        //设置 Latch 引脚为 LOW，允许数据输入
        digitalWrite(latchPin, LOW);
        shiftOut(i);
        //设置 Latch 引脚为 HIGH，锁存并送出数据
        digitalWrite(latchPin, HIGH);
        delay(500);
    }
}
```

`shiftOut` 函数接受的参数是一个字节（8 位元数字），表示一个在 0 到 255 之间的数。在这个项目中你选择一个字节为参数，它的长度正好为 8 位，你只需要给移位寄存器送一个 8 位数：

```
void shiftOut (byte dataOut) {
```

之后初始化一个叫做 `pinState` 的布尔变量。`pinState` 存储输出数据时（1 或 0）引脚的状态：

```
boolean pinState;
```

数据和时钟引脚设置为 `LOW`，以便于重置数据和时钟引脚，准备更新数据：

```
digitalWrite(dataPin, LOW);
digitalWrite(clockPin, LOW);
```

之后，准备以串行方式向 595 芯片送出 8 位数字，每次一位。用一个循环 8 次的 for 语句实现：

```
for (int i=0; i<=7; i++) {
```

在送出一位数据前，将时钟引脚设为 LOW：

```
digitalWrite(clockPin, LOW);
```

现在，使用一个 if/else 语句决定 pinState 的值是 1 还是 0：

```
if ( dataOut & (1<<i) ) {
    pinState = HIGH;
}
else {
    pinState = LOW;
}
```

if 语句的条件是：

```
dataout & (1<<i)
```

这是一个位屏蔽的例子。if 语句的条件使用了按位操作符。这些逻辑操作符与之前用到的布尔运算相似。然而，按位操作符是在位级别上的运算。

在这个例子里，使用按位与(&)操作执行两个数之间的逻辑运算。第一个数是 dataOut，第二个数是 (1<<i) 运算的结果。在学习其他的知识之前，先看一下按位操作。

按位操作

按位操作对变量在位级别上进行计算，有 6 个常用的按位操作：

- & 按位与
- | 按位或
- ^ 按位异或
- ~ 按位非

- << 按位左移
- >> 按位右移

按位操作只能在两个整型数之间执行。每个操作执行相应的逻辑规则计算。首先让我们详细看一下按位与运算符，之后再介绍其他运算符。

按位与 (&)

按位与操作根据如下规则进行计算：

如果两个操作数相同位上都是 1，则输出结果的该位上也是 1，否则为 0。

这个规则的另一种描述是：

0011 操作数 1

0101 操作数 2

.....

0001 (操作数 1 & 操作数 2)

典型的整型数是 16 位数值。因此，使用 & 在两个整型数之间展开计算产生 16 个并行的与操作，代码片段如下：

```
int x = 77;    //二进制: 0000000001001101
int y = 121;   //二进制: 0000000001111001
int z = x&y;   //二进制: 0000000001001001
```

即 77&121=73。

按位或 (|)

如果两个操作数相同位上至少有一个是 1，则输出结果的该位上也是 1，否则是 0。

0011 操作数 1

0101 操作数 2

.....

0111 (操作数 1 | 操作数 2)

按位异或 (^)

如果两个操作数对应位上的数相同结果为 0, 不同结果为 1。

0011 操作数 1

0101 操作数 2

.....

0110 (操作数 1 ^ 操作数 2)

按位非 (~)

按位非操作符只有一个操作数在操作符的右侧。

输出变成输入的取反。

0011 操作数 1

.....

1100 ~操作数 1

按位左移 (<<)、按位右移 (>>)

按位操作符向左或向右移动一个整数表示的位数, 移动的位数由操作符右侧的数指定。

变量<<左移位数

例如:

```
byte x = 9;           //二进制: 00001001
byte y = x << 3;       //二进制: 01001000 (或者十进制 72)
```

任何移出行端部的数据都将永久丢失。可以使用按位左移实现用 2 乘以一个数的运算, 按位右移相当于用 2 去除这个数。

现在已经了解了按位移操作符, 让我们重新回到代码。

代码回顾（继续）

if/else 语句的条件是：

```
dataOut & (1<<i)
```

你现在知道了，这是一个按位与（&）操作符。右侧括号内的操作是一个左移操作，这是一个位掩码。74HC595 一次只能接收数据的一位。因此需要将 dataOut 中的 8 位数转化为单个比特，也就是要返回一个字节 8 位中的每一位。位掩码确保 pinState 变量根据位屏蔽计算的结果设置为 1 或 0。右侧的操作数是把 1 按位移动 i 位，因为 for 循环使 i 值从 0 递增到 7，可以看到数字 1 通过每次循环位移动 i 位得到如下的二进制数结果（表 6-5）。

表 6-5 1<<i 的结果

i 的值	(1<<i) 的二进制结果
0	00000001
1	00000010
2	00000100
3	00001000
4	00010000
5	00100000
6	01000000
7	10000000

因此，可以看到这个操作的使 1 从右向左移动的结果。

现在，与操作的规则是：

如果操作数相应位上都是 1，则结果是 1，否则输出是 0。

所以对于条件

```
dataOut & (1<<i)
```

如果操作数与位掩码相同位置处的数都是 1，则逻辑与运算的结果是 1，否则是 0。例如，dataOut 是十进制数 139 或二进制数 10001011，那么每次循环的计算结果如表 6-6 所示。

表 6-6 $b10001011 \ll i$ 的结果

I 的值	$(1 \ll i)$ 的二进制结果
0	00000001
1	00000010
2	00000000
3	00001000
4	00000000
5	00000000
6	00000000
7	10000000

因此，每次第 I 位是 1（从右向左读）时，运算结果大于 1（或 TRUE）。每次第 I 位是 0 时，结果是 0（或 FALSE）。

如果条件运算结果大于 0（换句话说，如果相应位上的数是 1），执行 if 语句中的代码，否则执行 else 语句中的代码（如果该位的值是 0）。

因此再看一遍 if/else 语句：

```
if ( dataOut & (1<<i) ) {
    pinState = HIGH;
}
else {
    pinState = LOW;
}
```

通过参考表 6-6 中的真值表，可以看到在 dataOut 的值中比特值为 1 时，pinState 将设置为 HIGH，为 0 时，pinState 将设置为 LOW。

下一部分代码用来将高或低状态写到数据引脚，然后设置时钟引脚为高，把这个位写入存储寄存器：

```
digitalWrite(dataPin, pinState);
digitalWrite(clockPin, HIGH);
```

最后，时钟引脚设置为低，确保不再有比特数据写入：

```
digitalWrite(clockPin, LOW);
```

因此，简单地说，这段代码逐个查找 dataOut 值的 8 个比特，依据结果设置数据引脚的状态为高或低。之后把值写入存储寄存器中。

这种方法把一个 8bit 的数每次一比特地送入 595 芯片中，之后主循环设置 Latch 引脚为 HIGH，把这 8 个比特同时送到移位寄存器的引脚 15 和 1~7（QA~QH）上。所得到的结果是 8 个 LED 显示存到它的移位寄存器内的二进制数。

在项目 17 中你接触了太多的新知识，因此要休息一下，伸个懒腰，喝点什么，然后再进入项目 18。在项目 18 中你要学习如何把两个移位寄存器级联在一起。

项目 18——16 位二进制计数器

在项目 18 中，将级联（层叠）另外一个 74HC595 芯片到项目 17 中所用的芯片上，生成一个 16 位的二进制计数器。

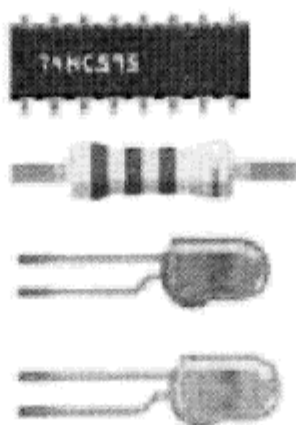
需要的元件

2 个 74HC595 移位寄存器芯片

16 个限流电阻

8 个红色 LED

8 个绿色 LED



把元件连接起来

第一个 595 芯片与项目 17 中的连接线完全相同。第二个 595 芯片的 +5V 和地连到第一个 595 芯片相同的引脚上。之后增加一根连线，将第一个 595 芯片的引脚 9 连到第二个 595 芯片的引脚 14 上。增加另外一根连线，将第一个 595 芯片的引脚 11 连到第二个 595 芯片的引脚 11 上。最后，将第一个 595 芯片的引脚 12 与第二个 595 芯片的引脚 12 连起来。

第二个 595 芯片的输出引脚和第二组 LED 之间的连线与第一个芯片的输出引脚与第一组 LED 之间的连线完全相同。

按图 6-4 和图 6-5 仔细检查电路。

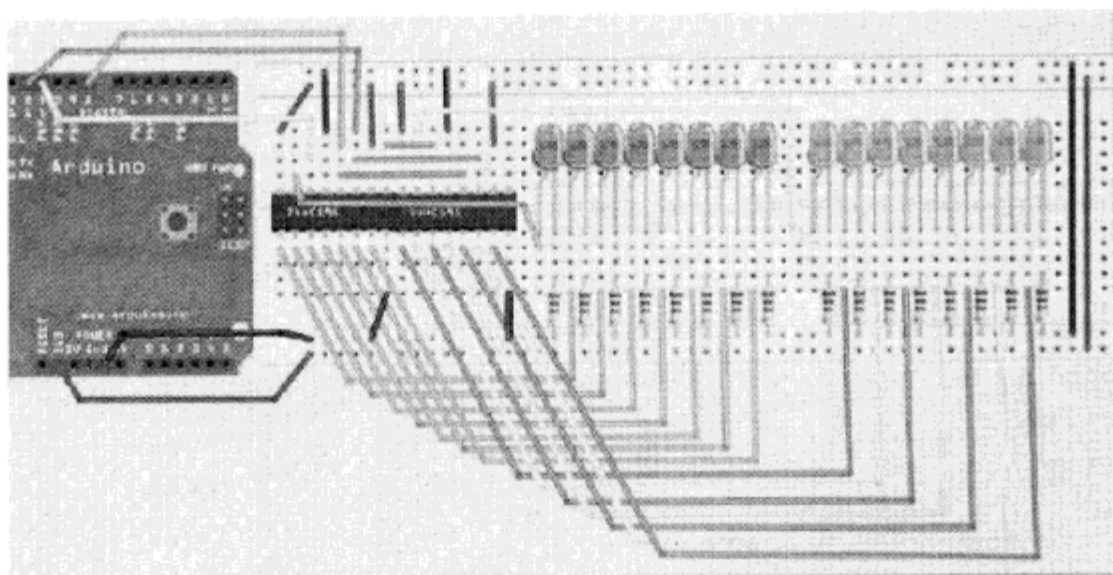


图 6-4 项目 18 电路图（参见彩色版插图）

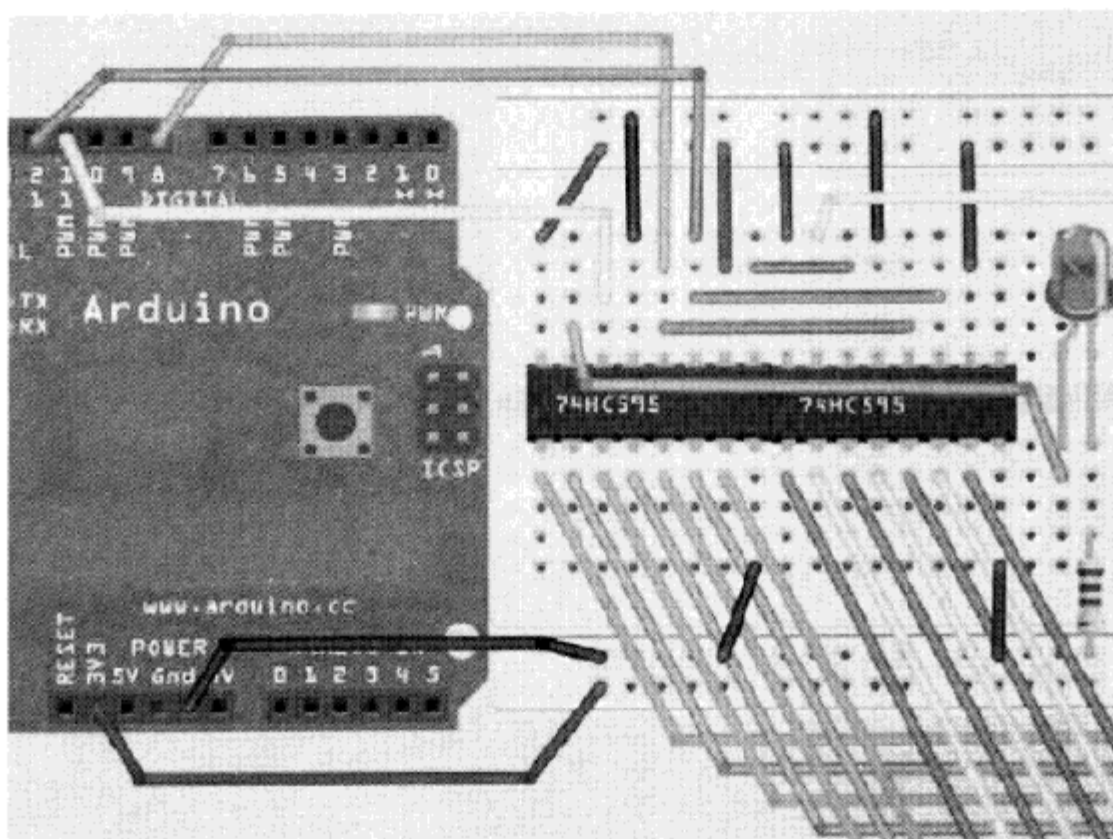


图 6-5 项目 18 中 IC 互相靠紧的连线图

输入代码

输入清单 6-2 中的代码，上传到 Arduino。当运行这个代码时，绿色组的 LED 从 0 到 255 计数（二进制），与此同时红色组的 LED 从 255 到 0 进行倒数。

清单 6-2 项目 18 的代码

// 项目 18 的代码

```

int latchPin = 8; //Arduino 连接到 74HC595 (Latch) 引脚 12 上的引脚
int clockPin = 12; //Arduino 连接到 74HC595 (Clock) 引脚 11 上的引脚
int dataPin = 11; //Arduino 连接到 74HC595 (Data) 引脚 14 上的引脚

void setup() {
    //设置引脚为输出模式
    pinMode(latchPin, OUTPUT);
    pinMode(clockPin, OUTPUT);
    pinMode(dataPin, OUTPUT);
}

void loop() {
    for (int i = 0; i < 256; i++) { //从 0 到 255 计数
        //设置 latchPin 引脚为 LOW, 允许数据输入
        digitalWrite(latchPin, LOW);
        shiftOut(i);
        shiftOut(255-i);
        //设置 latchPin 引脚为 HIGH, 允许输入数据
        digitalWrite(latchPin, HIGH);
        delay(250 );
    }
}

void shiftOut(byte dataOut) {
    boolean pinState; //在时钟上升沿低位优先输出 8 位数据

    digitalWrite(dataPin, LOW); //清除移位寄存器准备好送数据
    digitalWrite(clockPin, LOW);

    for (int i=0; i<=7; i++) { //一位一位地送出 dataOut 中的数据
        digitalWrite(clockPin, LOW); //在送比特数据前设置 clockPin 引脚为 LOW

        //如果 DataOut 与 bitmask 的逻辑或运算结果是 true, 设置 pinState 为 HIGH
        if ( dataOut & (1<<i) ) {
            pinState = HIGH;
        }
        else {
            pinState = LOW;
        }

        //根据 pinState 设置 dataPin 为 HIGH 或 LOW
        digitalWrite(dataPin, pinState);
    }
}

```

```

        digitalWrite(clockPin, HIGH); //在时钟上升沿送出比特数据
        digitalWrite(dataPin, LOW);
    }

    digitalWrite(clockPin, LOW); //停止送数据
}

```

代码和硬件回顾

除了在主循环中增加下面的语句：

```
shiftOut(255-i);
```

项目 18 中的代码与项目 17 中的完全相同。shiftOut 函数送出 8bit 到 595 芯片。在主循环中，调用两次 shiftOut 函数。一个以 i 为参数，另一个以 255-i 为参数。也就是在设置 latch 引脚为 HIGH 之前调用了两次 shiftOut 函数，送出两组 8 比特数据或总共 16bit 数据给 595 芯片。将 Latch 设置为 HIGH 可阻止更多的数据写入寄存器，并输出寄存器的内容到输出引脚，使 LED 点亮或熄灭。

第二个 595 芯片的连线与第一个完全一样。Clock 引脚和 Latch 引脚连接到第一个 595 芯片相应的引脚上，但是要把芯片 1 的引脚 9 与芯片 2 的引脚 14 用导线连起来。引脚 9 是数据输出引脚，引脚 14 是数据输入引脚。

数据从 Arduino 输入到第一个芯片的引脚 14。第二个 595 芯片通过第一个芯片的引脚 9 级联到第一个芯片上。第一个芯片上的引脚 9 输出数据给第二个芯片的引脚 14，引脚 14 是第二个芯片的输入引脚。

如果输入 9 个或 9 个比特以上的数据，数据将从第一个芯片引脚 9 流出，进入第二个芯片的数据输入引脚。因此如果 16bit 数据全部从 Arduino 中输出，前 8 个比特将从第一个芯片流入第二个芯片。第二个 595 芯片将保持前 8 个送出的比特数据，第一个 595 芯片保存第 9 个到第 16 个比特的数据。

用这种方法级联 595 芯片几乎没有数量限制。

练 习

使用与项目 18 同样的电路和 16 个 LED，产生跑马灯效果，使 16 个 LED 在首尾之间来回依次点亮。

小结

第 6 章论述了使用外围芯片增加 Arduino 输出引脚的方法。尽管可以不使用外围芯片完成这些项目，但是移位寄存器使增加 Arduino 输出引脚数量变得非常简单。如果不使用移位寄存器，这些项目的代码就会非常复杂。使用这种芯片把串行数据用并行形式输出是控制 LED 的理想方法。

不要被外围芯片吓倒。认真系统地研读它们的说明书，你就会明白它们是如何工作的。说明书乍一看非常复杂，但是一旦将有用信息从不相关的信息中提取出来，你就完全可以理解芯片是如何工作的。

第 7 章将继续使用移位寄存器，但是要用移位寄存器控制 LED 点阵显示器，LED 点阵显示器每一个单元至少包含 64 个 LED。我将演示如何使用多路复用技术同时控制大量的 LED。

本章的主题和概念

- 二进制数系统及如何实现十进制与二进制的相互转化
- 如何使用移位寄存器输入串行数据、输出并行数据
- 使用外围芯片降低项目的复杂度
- 给函数调用传递参数
- 使用布尔类型变量
- 位运算符的概念和使用方法
- 使用位运算符产生位掩码
- 如何级联两个或更多的移位寄存器



第7章



LED 显示器

到目前为止，你已经学习了如何使用单个 5mm LED。也可以把 LED 封装成点阵显示器。最流行的点阵显示器是 8×8 LED 矩阵，总共 64 个 LED。你也可以组成双色点阵显示器（如红、绿）甚至是 RGB 点阵显示器，RGB 点阵显示器把 192 个 LED 封装在同一个外壳中，可显示任何颜色。本章要使用标准单色 8×8 点阵显示器演示如何显示图形和文字。本章从在一个 8×8 显示器上显示动画的项目开始，然后进入更复杂的项目。在学习这些项目时，你会接触到一个非常重要的概念——多路复用技术。

项目 19——LED 点阵显示器——基本动画

在这个项目中，再次使用两组移位寄存器。这两组移位寄存器连接到点阵显示器的行和列上。之后在显示器上显示一个简单物体或小动画，并让它们动起来。这个项目的主要目的是演示点阵显示器的工作原理及多路复用的概念。多路复用技术是一种非常有用的技术。

需要的元件

需要准备 2 个移位寄存器（74HC595）和 8 个限流电阻。同时还需要一个共阳极点阵显示器及其说明书，以便于查看说明书来了解点阵显示器的引脚与哪一行或哪一列的 LED 连接。

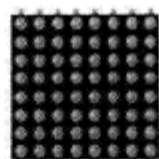
2 个 74HC595 移位寄存器芯片



8 个限流电阻



8×8 点阵显示器（共阳极）



把元件连接起来

仔细检查电路。在电路连完之前，不要给 Arduino 上电。这一点非常重要，否则有可能会损坏移位寄存器或点阵显示器。这个电路非常复杂，因此接线时要十分小心。连接电路时要慢一点，有条不紊地进行。

图 7-1 的连线图是基于我所用的一个小型 8×8 红色点阵显示模块的。但是，你使用的显示器可能（非常可能）与我所用的引脚完全不同。你必须仔细研读所用器件的说明书，保证把移位寄存器引脚正确地连接到点阵显示器相应的引脚上。www.egr.msu.edu/classes/ece480/goodman/read_datasheet.pdf 为你提供了如何阅读器件说明书的 PDF 格式的教程。

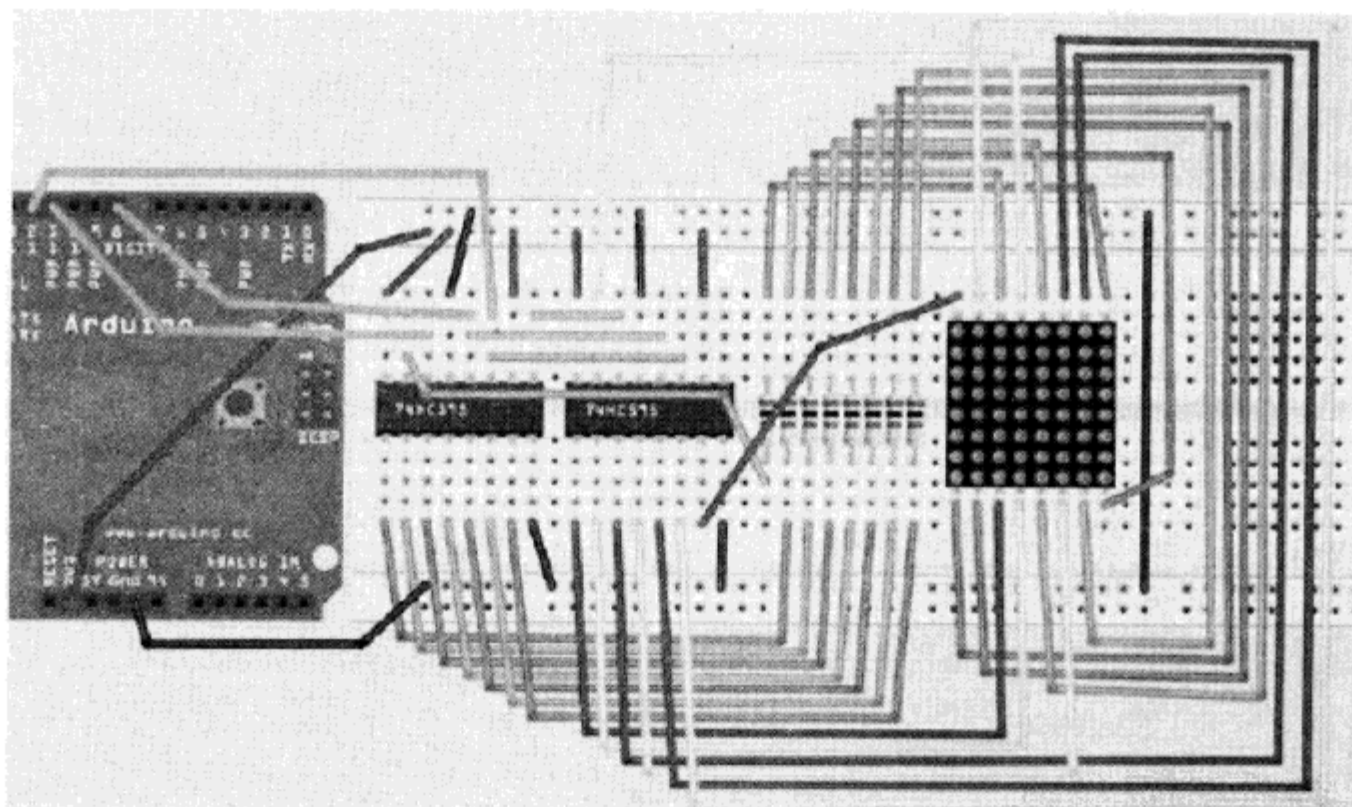


图 7-1 项目 19——LED 点阵显示器——基本动画项目的电路图

为了使你学起来更轻松，表 7-1 给出了移位寄存器引脚与所连接的点阵显示器引脚的对应关系。根据这个表调整电路，确保你所连接的电路是正确的。

表 7-1 点阵显示器需要的引脚

	移位寄存器 1	移位寄存器 2
行 1	引脚 15	
行 2	引脚 1	
行 3	引脚 2	
行 4	引脚 3	
行 5	引脚 4	

续表

	移位寄存器 1	移位寄存器 2
行 6	引脚 5	
行 7	引脚 6	
行 8	引脚 7	
列 1		引脚 15
列 2		引脚 1
列 3		引脚 2
列 4		引脚 3
列 5		引脚 4
列 6		引脚 5
列 7		引脚 6
列 8		引脚 7

本项目所用的点阵显示器的原理图如图 7-2 所示。可以看到行和列（正极和负极）号不是按逻辑顺序排列的。根据表 7-1 和概略图 7-2 可以看到，移位寄存器 1 的引脚 15 需要与点阵显示器的第一行相连，因此要连到显示器的引脚 9 上。移位寄存器的引脚 1 需要与第二行相连，因此要连到显示器的引脚 14 上，依此类推。

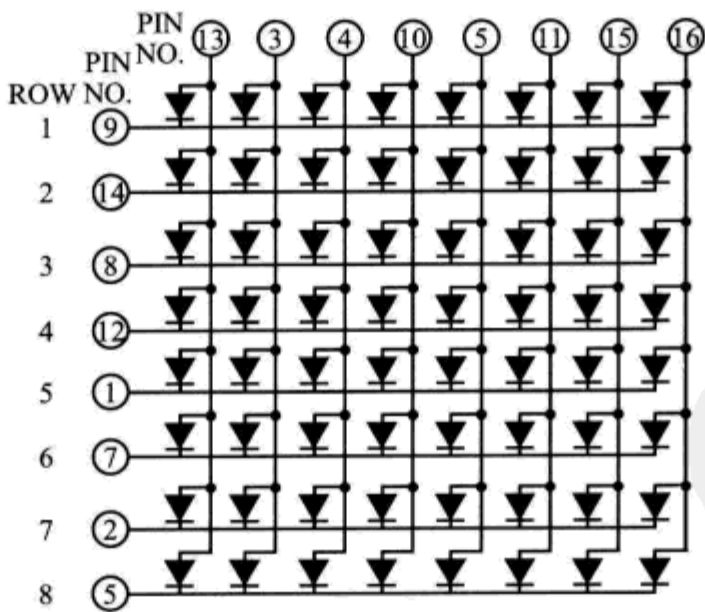


图 7-2 一个典型的 8×8 LED 点阵显示器原理图

你要阅读所用显示器的说明书，通过以上的步骤去确定移位寄存器引脚和 LED 显示器引脚的连接关系。

输入代码

确认连线正确之后，输入清单 7-1 中的代码，上传到 Arduino。你需要下载 TimerOne

库。你可以从 Arduino 的网站 www.arduino.cc/palyground/Code/Timer1 下载。下载完该库之后，解压，并把 TimerOne 的完整文件夹放到 Arduino 安装目录中的 hardware/libraries 文件夹内。这是一个外部库的例子，Arduino 的 IDE 预安装了许多库，如 Ethernet、LiquidCrystal、Servo 等，TimerOne 库是一个外部库，你仅需要下载并将它的文件夹放在 Arduino IDE 的库文件夹中（需要重启 IDE 之后它才能被注册）。

库只是其他人已经编写完并且别人也可使用的代码的集合。使用库避免了重新编写代码的麻烦。这就是代码复用，它可以加快项目开发的速度。毕竟事事从头开始没什么好处。如果某些人已经编写了一套代码去执行特定的任务，而且把代码放在公共领域与大家分享，那么你就可以直接使用这些代码。

一旦代码运行，你将会在显示器上看到一个心形图案。大约每隔半秒，显示器上原来亮的 LED 关闭，原来关闭的 LED 点亮，这种明暗转换使图像出现简单的动画效果。

清单 7-1 项目 19 的代码

```
//项目 19
#include <TimerOne.h>

int latchPin = 8; //Arduino 连接到 74HC595 (Latch)引脚 12 上的引脚
int clockPin = 12; //Arduino 连接到 74HC595 (Clock) 引脚 11 上的引脚
int dataPin = 11; //Arduino 连接到 74HC595 (Data)引脚 14 上的引脚

byte led[8]; //存储子画面的 8 元素无符号整型数组
void setup() {
    pinMode(latchPin, OUTPUT); //设置数字引脚 3 为输出模式
    pinMode(clockPin, OUTPUT);
    pinMode(dataPin, OUTPUT);
    led[0] = B11111111; //输入二进制表示的图像
    led[1] = B10000001; //输入到数组
    led[2] = B10111101;
    led[3] = B10100101;
    led[4] = B10100101;
    led[5] = B10111101;
    led[6] = B10000001;
    led[7] = B11111111;
    //设置定时长度为 10000ms (1/100s) 的定时器
    Timer1.initialize(10000);
    //连接定时中断服务程序到 screenUpdate 函数
    Timer1.attachInterrupt(screenUpdate);
}
```

```

void loop() {
    for (int i=0; i<8; i++) {
        led[i]= ~led[i]; //转换每行的二进制图像
    }
    delay(500);
}

void screenUpdate() { //显示图像的函数
    byte row = B10000000; // row 1
    for (byte k = 0; k < 9; k++) {
        digitalWrite(latchPin, LOW); //打开 Latch 引脚准备接收数据
        shiftIt(~led[k] ); //送出翻转后的 LED 数组
        shiftIt(row ); //送出二进制数

        //关闭 Latch 引脚从寄存器送出数据控制点阵显示器
        digitalWrite(latchPin, HIGH);
        row = row << 1; //向左移位
    }
}

void shiftIt(byte dataOut) { //在时钟上升沿低位优先送出 8 为数字

    boolean pinState;
    digitalWrite(dataPin, LOW); //清除移位寄存器准备送数据

    for (int i=0; i<8; i++) { //送出 dataOut 的每一位
        //在送数据前设置 clockPin 引脚为 LOW
        digitalWrite(clockPin, LOW);

        //如果 DataOut 与掩码进行与运算的结果是 true, 设置 pinState 为 1 (HIGH)
        if ( dataOut & (1<<i) ) {
            pinState = HIGH;
        }
        else {
            pinState = LOW;
        }
        //根据 pinState 设置 dataPin 引脚为 HIGH 或 LOW
        digitalWrite(dataPin, pinState);
        digitalWrite(clockPin, HIGH); //在时钟上升沿送出数据
        digitalWrite(dataPin, LOW);
    }
    digitalWrite(clockPin, LOW); //停止对移位寄存器操作
}

```

硬件回顾

对于这个项目，在弄明白代码是如何工作的之前先看一下硬件，这会使代码变得更容易理解。

在前面的项目中已经学习了如何使用 74HC595，这次在电路中只是增加了 8×8 点阵 LED 模块。

典型的点阵显示模块有 5×7 或 8×8 两种点阵 LED。每一行中的 LED 的正极或者负极连在一块组成矩阵。换句话说，在一个共阳极 LED 点阵模块里，把每一行中 LED 的阳极连在一起，每一列中 LED 的阴极连在一起。下面揭示这样做的原因。

一个典型的 8×8 单色点阵模块有 16 个引脚，8 行 8 列。你也可以使用双色模块（如红色和绿色）甚至全色 RGB（红色、绿色、蓝色）模块——用于大型电视墙上。双色或全色模块在每个像素点上有两个或三个 LED，它们非常小并且距离非常近。

通过改变每个像素点上红、绿、蓝的组合模式及亮度可以得到任何颜色。

所有行或列的引线连在一起的原因是最小化所需要引脚的数量。如果不采用这种方法，一个单色 8×8 点阵模块就要用到 65 个引脚，每一个 LED 需要一个引脚加上一个共阴极或共阳极的引脚。而使用把行和列连起来的方法只需要 16 个引脚。

然而，问题是如果你想点亮一个特定位置上的特定的 LED，例如，有一个共阳极模块，希望点亮 X 和 Y 位置为 5 和 3（第 5 列、第 3 行）的 LED，那么需要给第 3 行的阳极通电，第 5 列的阴极接地。

第 5 列第 3 行的 LED 将被点亮（如下图）。

	1	2	3	4	5	6	7	8
1								
2								
3								
4								
5								
6								
7								
8								

现在让我们想象一下，如果你还想同时点亮第 3 列、第 6 行的 LED，因此要给第 6 行施加电流，第 3 列引脚连到地。第 3 列、第 6 行的 LED 将被点亮，但是因为第 3 行、第 5 列也施加了电流，所以第 3 列、第 6 行和第 5 列、第 6 行的 LED 也将被点亮（如下图）。

	1	2	3	4	5	6	7	8
1								
2								
3								
4								
5								
6								
7								
8								

也就是你给第3行和第6行LED供电，第3列和第5列接地。所以在不关掉你希望点亮的LED的前提下，不能关闭不希望点亮的LED。很明显，没有办法只点亮所需要的LED而不点亮不希望点亮的LED，因为它们的行列线连在一起。要想能分别点亮每一个LED，唯一的办法是每一个LED都引出一个独立的引脚，这意味着引脚数会从16增加到64。一个64个引脚的点阵模块将非常难以接线和控制，因为所需要的微控制器至少要有64个数字输出引脚。

有办法解决这个问题吗？是的，有，它叫做多路复用技术（或 muxing）。

多路复用技术

多路复用是一种在同一时间只开一行显示的技术。选择包含你想点亮的LED的行和列，给该行上电（或者可用于共阳极LED的其他方法），行中的LED将点亮。之后关闭该行，打开下一行。对所选择的列执行同样的操作，第二行中的LED将被点亮。重复以上操作，分别打开每一行直到最后一行。之后，重新从第一行开始。

如果运行得足够快（大约100Hz，或者每秒100次），那么视觉暂留现象（已经消失的图像在视网膜上停留1/25 s的现象）将静态地显示图像，尽管每行是按顺序交替开关的。

通过使用这项技术，我们点亮了单个LED而没有使在同一行或一列中的其他LED也被点亮。

例如，要在显示器上显示如下的图形：

	1	2	3	4	5	6	7	8
1								
2								
3								
4								
5								
6								
7								
8								

那么每一行将像下图一样被点亮：

	1	2	3	4	5	6	7	8
1								
2								
3								
4								
5								
6								
7								
8								

	1	2	3	4	5	6	7	8
1								
2								
3								
4								
5								
6								
7								
8								

	1	2	3	4	5	6	7	8
1								
2								
3								
4								
5								
6								
7								
8								

	1	2	3	4	5	6	7	8
1								
2								
3								
4								
5								
6								
7								
8								

	1	2	3	4	5	6	7	8
1								
2								
3								
4								
5								
6								
7								
8								

	1	2	3	4	5	6	7	8
1								
2								
3								
4								
5								
6								
7								
8								

	1	2	3	4	5	6	7	8
1								
2								
3								
4								
5								
6								
7								
8								

	1	2	3	4	5	6	7	8
1								
2								
3								
4								
5								
6								
7								
8								

通过以非常快的速度（大于 100Hz）向下扫描每一行点亮这行相应的列中的 LED，人类的眼睛将以静态的方式感知这个图像，因此在 LED 显示器上看到一个心形图像。

在这个项目的代码中使用了多路复用技术。这就是如何显示心形图像而没有显示不相干的 LED 的方法。

代码回顾

这个项目的代码使用了 Atmega 芯片的一个叫硬件时钟中断的功能。它是一个芯片上自带的计时器，可用于触发一个事件。在这个例子里设置每 10ms 触发一次 ISR（中断服务程序），即每秒触发 100 次 ISR。

该项目使用了一个叫做 TimerOne 的函数库，它简化了使用中断的过程。TimerOne 产生中断服务程序十分容易。只要简单地告诉函数中断间隔时间（此处是 10ms）和当中断触发时你希望激活的函数的名称（在这个例子中，它是 screenUpdate() 函数）即可。

TimerOne 是一个外部库，因此需要加载到你编写的代码中，这需要使用 include 命令：

```
#include <TimerOne.h>
```

之后，声明移位寄存器的接口引脚：

```
int latchPin = 8; //连到 74HC595 (Latch) 引脚 12 上的引脚
int clockPin = 12; //连到 74HC595 (Clock) 引脚 11 上的引脚
int dataPin = 11; //连到 74HC595 (Data) 引脚 14 上的引脚
```

之后，定义一个有 8 个元素的 byte 型数组，led[8] 数组用来存储要在点阵显示器上显示的图像：

```
byte led[8]; //8 个元素的无符号整形数组用来存储图像
```

在 Setup 函数中，设置 latch、clock、data 引脚为输出：

```
void setup() {
    pinMode(latchPin, OUTPUT); //设置 3 个数字引脚模式为 OUTPUT
    pinMode(clockPin, OUTPUT);
    pinMode(dataPin, OUTPUT);
}
```

引脚设置为输出之后，将 8 位二进制图像存储在 led 数组中。图像将显示在 8×8 点阵

显示器上。

```
led[0] = B11111111; //进入二进制形式表现的图像
led[1] = B10000001; //进入数组
led[2] = B10111101;
led[3] = B10100101;
led[4] = B10100101;
led[5] = B10111101;
led[6] = B10000001;
led[7] = B11111111;
```

通过观察以上数组，就能看出将要显示的图像是什么样子的。它是一个“回”字形图像。数组元素中位为 1 表明相应位置的 LED 点亮，0 表示相应位置的 LED 熄灭。当然你也可以自己调整相应的 1 和 0，使 LED 显示任何 8×8 图形。

随后，使用了 Timer1 类。首先，这个类需要初始化激活它的频率。在这个例子里，所设置的周期是 10000us，即 1/100s。一旦中断被初始化，需要连接这个中断到一个函数。当每个中断到来之时，执行这个函数。在这里连接的是 screenUpdate()函数，所以该函数将每 1/100s 执行一次：

```
//设置 timer 的长度是 10000us (1s 的 1/100)
Timer1.initialize(10000);
//连接 screenUpdate 函数到中断定时器
Timer1.attachInterrupt(screenUpdate);
```

在主循环中，通过 for 循环逐个访问 LED 数组中的 8 个元素，并用~或按位非操作符转换这 8 个元素的内容。通过将所有 1 转换成 0 或将 0 转换成 1 将二进制图像翻转。等待 500ms 之后重复运行。

```
for (int i=0; i<8; i++) {
    led[i]= ~led[i]; //翻转每一行的二进制图像
}
delay(500);
```

接下来是 screenUpdate()函数。这个函数每 1/100s 被中断激活一次。这个函数是非常重要的，因为它的任务是保证点阵数组中的 LED 显示正确。这是一个非常简单但是有效的函数。

```

void screenUpdate() { //显示图像函数
byte row = B10000000; //第一行
for (byte k = 0; k < 9; k++) {
    digitalWrite(latchPin, LOW); //开锁寄存器准备接收数据

    shiftIt(~led[k] ); //移出 LED 数组（反向的）
    shiftIt(row ); //移出行二进制数
    //关闭锁存器，将寄存器中的数据送到点阵显示器
    digitalWrite(latchPin, HIGH);
    row = row << 1; //向左移动一位
}
}

```

声明一个叫 row 的字节变量并且用值 B10000000 初始化：

```
Byte row=B10000000; //第一行
```

现在从 led 数组开始循环，将数据送到移位寄存器（用按位非~处理，使希望显示的列关闭或接地）。之后的程序是：

```

for(byte k=0;k<9;k++){
    digitalWrite(latchPin , LOW); //开锁寄存器准备接受数据
    shiftIt(~led[k]); //led 数组（反向的）
    shiftIt(row); //二进制数行
}

```

一旦送出当前行的 8 个比特，行中比特值转换到下一个位置，因此显示下一行（因为 shiftIt 仅仅显示有 1 的那一行）。在第 6 章中已经学习了 shiftIt 函数。

```
row=row>>1; //按位右移
```

在硬件回顾中提到，多路复用技术的程序每一次仅显示一行，关闭这一行之后显示下一行。这是在 100 Hz 的频率下进行的，对于人类的眼睛来说看这种闪烁太快了。

最后，使用 shiftIt 函数。与之前的移位寄存器项目相同，该函数也是将数据送到 74HC595 芯片中：

```
void shiftIt(byte dataOut)
```

因此，本项目最重要的概念是每 1/100s 中断一次。在程序中，你只是使用屏上的寄存器数组的内容（在这里是 led[]），并且每次在点阵显示单元上显示一行，但是这样做的速度

非常快，所以人看到的是各行同时点亮的效果。

程序的主循环只是改变屏幕缓冲区寄存器数组的内容，之后让中断处理函数做剩下的事情。

这个项目中的动画非常简单，但是通过改变寄存器数组中的 1 和 0 数据，可以在点阵单元上显示从静态图像到滚动文字的任何图形。我们试着在下一个项目中做一些小改动，产生一个动态的画面。

项目 20——LED 点阵显示器——滚动画面

使用同样的电路，只是在代码上做一点小小的改变，就可以实现多帧动画、从右到左滚动图像的效果。在本项目中，我将要介绍多维数组和按位循环（或循环移动）的概念。本项目使用与项目 19 完全相同的电路。

输入代码

输入并上传清单 7-2 中的代码。

清单 7-2 项目 20 的代码

```
//项目 20
#include <TimerOne.h>

int latchPin = 8; //Arduino 连接到 74HC595 (Latch)引脚 12 上的引脚
int clockPin = 12; //Arduino 连接到 74HC595 (Clock)引脚 11 上的引脚
int dataPin = 11; //Arduino 连接到 74HC595 (Data)引脚 14 上的引脚
byte frame = 0; //存储正在显示的当前帧的变量

byte led[8][8] = { {0, 56, 92, 158, 158, 130, 68, 56}, //动画的 8 帧画面
                   {0, 56, 124, 186, 146, 130, 68, 56},
                   {0, 56, 116, 242, 242, 130, 68, 56},
                   {0, 56, 68, 226, 242, 226, 68, 56},
                   {0, 56, 68, 130, 242, 242, 116, 56},
                   {0, 56, 68, 130, 146, 186, 124, 56},
                   {0, 56, 68, 130, 158, 158, 92, 56},
                   {0, 56, 68, 142, 158, 142, 68, 56} };

void setup() {
```

```

pinMode(latchPin, OUTPUT); //设置数字引脚 3 为输出模式
pinMode(clockPin, OUTPUT);
pinMode(dataPin, OUTPUT);

Timer1.initialize(10000); //设置一个定时长度为 10000 微秒的定时器
Timer1.attachInterrupt(screenUpdate); //连接定时中断到 screenUpdate 函数
}

void loop() {
    for (int i=0; i<8; i++) { //循环通过动画的 8 帧
        for (int j=0; j<8; j++) { //循环通过每帧的 8 行
            led[i][j]= led[i][j] << 1 | led[i][j] >> 7; //按位循环
        }
    }
    frame++; //到动画的下一帧
    if (frame>7) { frame = 0; } //超过第 7 帧后返回到第 0 帧
    delay(100); //在每帧中间延时一会儿
}

void screenUpdate() { //显示图像的函数
    byte row = B10000000; // row 1
    for (byte k = 0; k < 9; k++) {
        digitalWrite(latchPin, LOW); //开锁存引脚准备接收数据

        shiftIt(~led[frame][k] ); //led 数组 (取反)
        shiftIt(row); //行二进制数

        // 关锁存, 从寄存器送出数据控制点阵显示器
        digitalWrite(latchPin, HIGH);
        row = row >> 1; //向右比特移位
    }
}

void shiftIt(byte dataOut) {
    //在时钟上升沿低位优先送出 8 位数据

    boolean pinState;
    //清空移位寄存器
    digitalWrite(dataPin, LOW);

    //送出 dataOut 的每一位
    for (int i=0; i<8; i++) {
        //在送出数据前设置 clockPin 引脚为 LOW

```

```

        digitalWrite(clockPin, LOW);

        // 如果dataOut和掩码按位与计算结果是true, 设置pinState为1 (HIGH)
        if ( dataOut & (1<<i) ) {
            pinState = HIGH;
        }
        else {
            pinState = LOW;
        }

        //根据 pinState 设置 dataPin 为 HIGH 或 LOW
        digitalWrite(dataPin, pinState);
        //在时钟上升沿送出数据
        digitalWrite(clockPin, HIGH);
        digitalWrite(dataPin, LOW);
    }

    digitalWrite(clockPin, LOW); //停止操作移位寄存器
}

```

当运行项目 20 时, 你会看到一个旋转轮子的动画。因为本项目的硬件没有发生任何变化, 所以在这里不需要讨论硬件问题, 我们直接看代码是如何工作的。

代码回顾

还是加载 **TimerOne** 库, 并且设置 3 个引脚来控制移位寄存器:

```

#include <TimerOne.h>
int latchPin = 8; //Arduino 连接到 74HC595 (Latch)引脚 12 的引脚
int clockPin = 12; //Arduino 连接到 74HC595 (Clock)引脚 11 的引脚
int dataPin = 11; //Arduino 连接到 74HC595 (Data)引脚 14 的引脚

```

之后声明一个字节型变量并且初始化为 0, 它用来存储 8 帧动画中的当前帧的编号:

```
byte frame = 0; //用来存储当前显示帧的变量
```

之后设置字节型的二维数组:

```

byte led[8][8] = { {0, 56, 92, 158, 158, 130, 68, 56}, //动画的 8 个帧
                   {0, 56, 124, 186, 146, 130, 68, 56},
                   {0, 56, 116, 242, 242, 130, 68, 56},

```

```
{0, 56, 68, 226, 242, 226, 68, 56},
{0, 56, 68, 130, 242, 242, 116, 56},
{0, 56, 68, 130, 146, 186, 124, 56},
{0, 56, 68, 130, 158, 158, 92, 56},
{0, 56, 68, 142, 158, 142, 68, 56} };
```

数组的概念在第3章中已经介绍过了，数组是变量的集合，通过一个索引号来访问其中的变量。这个数组有些不同，因为它有两个元素索引号。在第3章中，声明了一个一维数组：

```
byte ledPin[] = {4,5,6,7,8,9,10,11,12,13};
```

此处生成了一个有两个索引号的二维数组。在这个例子中，数组是 8×8 的，总共 64 个元素。二维数组非常像一个二维数据表，因为可以通过相应的行、列号来获得一个单元的内容。表 7-2 显示了如何得到数组中的元素。

表 7-2 数组中的元素

	0	1	2	3	4	5	6	7
0	0	56	92 158		158	13 068		56
1	0	56	124 186	146		130	68	56
2	0	56	116 242	242		130	68	56
3	0	56	68 226		242	22 668		56
4	0	56	68	130 242		242 116		56
5	0	56	68	130 146		186 124		56
6	0	56	68 130		158	15 892		56
7	0	56	68 142		158	14 268		56

数组索引中的第一个数字代表行，如 `byte led[8][..]`。数组索引中的第二个数字代表列，如 `byte led[..][8]`。为了获得编号为 6 的行、编号为 4 的列中的数字 158，应该使用 `byte led[6][4]`。

注意，当声明数组时，应同时用数字初始化它。为了初始化二维数组，在一个花括号内输入全部数据，具有相同的第二个索引值的数放入它自己的花括号内，并用逗号隔开，如下：

```
byte led[8][8] = { {0, 56, 92, 158, 158, 130, 68, 56},
                    {0, 56, 124, 186, 146, 130, 68, 56},
                    {0, 56, 116, 242, 242, 130, 68, 56}, //依此类推
```

这个二维数组要存储动画的 8 个帧，数组的第一个索引号指向动画的帧，第二个索引号下的 8 个字节组成 LED 的开关组合。为了节省代码空间，数字已经从二进制转换成十进制。如果要查看二进制数格式的数组，将做出如图 7-3 所示的动画。

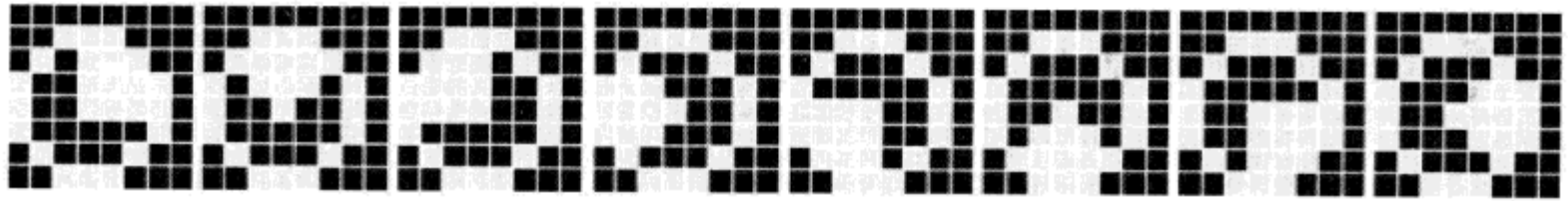


图 7-3 转动轮子的动画

当然，你可以将这个动画变成任何你想变成的东西，也可以增加或减少帧的数量。在坐标纸上画出动画，之后将每行转化成 8 比特二进制数并把它们输入数组中。

在 `setup` 函数中设置 3 个引脚为输出，生成一个 `timer` 对象。用长度为 10000 微秒初始化，之后将 `screenUpdate()` 与中断连接：

```
void setup() {
    pinMode(latchPin, OUTPUT); //设置 3 个数字引脚模式为输出
    pinMode(clockPin, OUTPUT);
    pinMode(dataPin, OUTPUT);

    Timer1.initialize(10000); //设置一个时间为 10000 微秒的中断
    Timer1.attachInterrupt(screenUpdate); //将 screenUpdate 函数送入中断中
}
```

在主循环中，与项目 19 相同，对子画面的 8 个行执行循环操作。然而，这个循环在另外一个循环的内部，它重复 8 次。这个外部的循环控制显示哪一帧：

```
void loop() {
    for (int i=0; i<8; i++) { //对动画的 8 个帧执行循环操作
        for (int j=0; j<8; j++) { //对每个帧的 8 个元素执行循环操作
```

之后，将数组中的每一个元素向左移动一位。使用一个相当简洁的逻辑运算就可以保证移出左端的数出现在右端，用下面这个逻辑运算实现：

```
led[i][j]= led[i][j] << 1 | led[i][j] >> 7; //按位循环
```

以上程序表示由整数 `i` 和 `j` 决定的数据的当前元素向左移动一位，之后与向右移 7 位的该元素运行按位逻辑或运算。让我们看一下这会发生什么。

先假定 `led[i][j]` 的值是 156，可用二进制数 10011100 表示。将这个数左移一位，得到的结果是 00111000。同样将 156 这个数右移 7 位，得到 00000001，换句话说，你已经把最左端的二进制位移动到最右端，现在对这两个数执行按位逻辑或操作。按位逻辑或计算将产生如下 8 位二进制数：

```
00111000|
00000001=
-----
00111001
```

因此，你把这个数字向左移动一位，之后用同一个数字右移 7 位，最后对两次移位结果进行按位或操作。就像在上面看到的一样，这样做的结果等同于将这个数字左移 1 位，左移溢出那位回到右端，这叫按位循环或者循环移位。这个技术在数字系统中经常用到。可以用如下方法对任何长度的二进制数字进行按位循环操作：

```
i << n | i >> (a - n);
```

这里 `n` 是一个希望循环移位的位数长度，`a` 是该二进制数字的位长度。

接下来，帧值加一，检查它是否大于 7，如果是，将该值置零。这将对动画的 8 个帧逐个显示，直到到达最后一帧，之后再从第一帧开始重复同样的显示操作。最后，延时 100ms。

```
frame++; //到动画中的下一个帧
if (frame>7) { frame =0;} //确保当帧超过 7 时返回到第 0 帧
delay(100); //在每个帧之间延时
```

之后运行 `screenUpdata()` 和 `shiftIt` 移位函数，像之前移位寄存器基础项目中讲的那样。在下一个项目中，再次使用 LED 点阵显示器，但是这次我们不使用移位寄存器，而是使用流行的 MAX7219 芯片。

项目 21——LED 点阵显示器——滚动信息

有许多驱动 LED 的方法，它们各自有各自的优点。使用移位寄存器是驱动 LED 方法中的一种。然而，还有许多专门为驱动 LED 显示器设计的芯片，会使你的工作变得更简单。对 Arduino 来说，最流行的 LED 驱动芯片是 MAX7219 串行接口芯片。它是 Maxim 公司制

造的一种 8 位 LED 驱动芯片。这种芯片可实现 8 位段码 LED 显示、柱状图显示或 8×8 点阵显示。Arduino IDE 有一个库叫 Matrix Plus，它提供了一些用于 MAX7219 芯片的示例代码，这个库使得使用这些芯片变得非常简单。然而，在这个项目中，你将不借助于任何外部库，而是自己写代码完成这些工作，自己编写代码中的每一部分。这样，你会了解 MAX7219 芯片的工作细节，并且可以把这些技巧转换到你希望用的其他芯片上。

需要的元件

该项目要使用一个 MAX7219 LED 驱动芯片，也可以用澳大利亚 Micorsystems 公司的 AS1107 代替。它与 MAX7219 几乎完全一样，不需要改变你的代码或电路就可以使用 AS1107 芯片。这次用的 8×8 点阵显示器需要是共阴极的，因为 MAX 芯片与共阳极显示器不兼容。

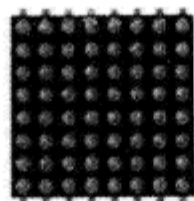
MAX7219（或 AS1107）



限流电阻



8×8 点阵显示器（共阴极）



把元件连接起来

按图 7-4 仔细检查电路。确保当你连接电路时 Arduino 没有上电。在图 7-4 中，从 MAX7219 到点阵显示器的连线是基于我在本项目中使用的显示元件的，你所用的显示器引脚可能跟我的不同。这没有关系，只要从 MAX7219 引脚出来的连线正确地连接到显示器的行列引脚上就可以了（见表 7-3）。从表的行中可看到各个元件之间引脚连接关系。在显示器上，列是阳极，行是阴极。在我的显示器上，行 1 引脚在底部，行 8 引脚在顶部。如果你发现你的显示器上字符上下翻转或前后倒置，应该改变你所编写的代码。连接 Arduino 的 5V 端到面包板的正极上，地端连接到面包板的地端。

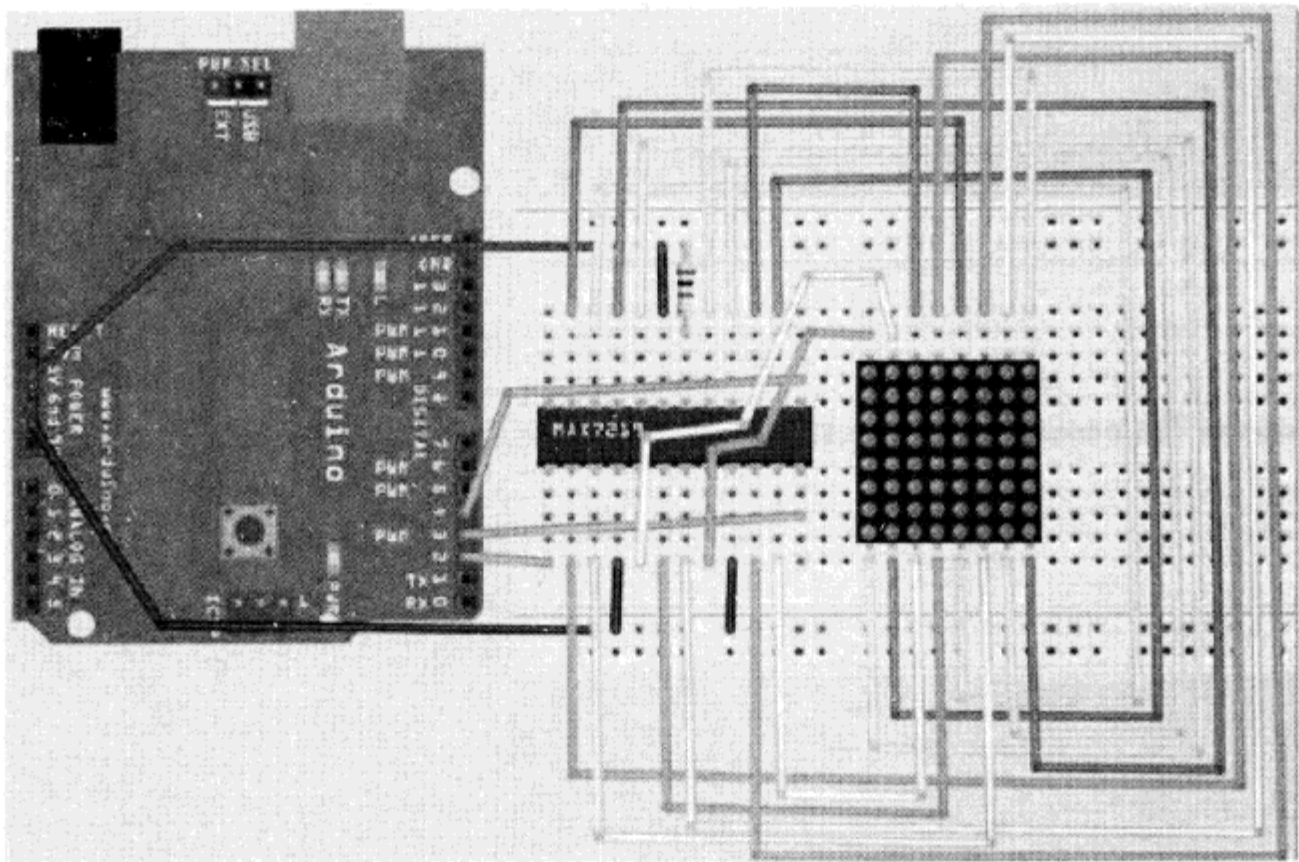


图 7-4 项目 21 的电路图（参见彩色版插图）

表 7-3 Arduino、IC 和点阵显示器之间的引脚连线

Arduino	MAX7219	显 示 器	其 他
数字引脚 2	1 (DIN)		
数字引脚 3	12 (LOAD)		
数字引脚 4	13 (CLK)		
4	9		Gnd
19			+5V
	18 (ISET)		电阻到+5V
	2 (DIG 0)	列 1	
	11 (DIG 1)	列 2	
	6 (DIG 2)	列 3	
	7 (DIG 3)	列 4	
	3 (DIG 4)	列 5	
	10 (DIG 5)	列 6	
	5 (DIG 6)	列 7	
	8 (DIG 7)	列 8	
	22 (SEG DP)	行 1	
	14 (SEG A)	行 2	
	16 (SEG B)	行 3	
	20 (SEG C)	行 4	

续表

Arduino	MAX7219	显 示 器	其 他
	23 (SEG D)	行 5	
	21 (SEG E)	行 6	
	15 (SEG F)	行 7	
	17 (SEG G)	行 8	

在给 Arduino 上电之前，检查电路是否连接正确。

输入代码

输入并上载清单 7-3 中的代码。

清单 7-3 项目 21 的代码

```
#include <avr/pgmspace.h>
#include <TimerOne.h>

int DataPin = 2; //MAX 上的引脚 1
int LoadPin = 3; //MAX 上的引脚 12
int ClockPin = 4; //MAX 上的引脚 13
byte buffer[8];

static byte font[][8] PROGMEM = {
//32~126 之间的可打印的 ASCII 字符
{B00000000, B00000000, B00000000, B00000000, B00000000, B00000000, B00000000, B00000000},
{B00000100, B00000100, B00000100, B00000100, B00000100, B00000100, B00000000, B00000100},
{B00001010, B00001010, B00001010, B00000000, B00000000, B00000000, B00000000, B00000000},
{B00000000, B00001010, B00011111, B00001010, B00011111, B00001010, B00011111, B00001010},
{B00000111, B00001100, B00010100, B00001100, B00000110, B00000101, B00000110, B00011100},
{B00011001, B00011010, B00000010, B00000100, B00000100, B00001000, B00001011, B00010011},
{B00000110, B00001010, B00010010, B00010100, B00001001, B00010110, B00010110, B00001001},
{B00000100, B00000100, B00000100, B00000000, B00000000, B00000000, B00000000, B00000000},
{B00000010, B00000100, B00001000, B00001000, B00001000, B00001000, B00000100, B00000010},
{B00001000, B00000100, B00000010, B00000010, B00000010, B00000010, B00000100, B00001000},
{B00010101, B00001110, B00011111, B00001110, B00010101, B00000000, B00000000, B00000000},
{B00000000, B00000000, B00000100, B00000100, B00011111, B00000100, B00000100, B00000000},
{B00000000, B00000000, B00000000, B00000000, B00000000, B00000110, B00000100, B00001000},
{B00000000, B00000000, B00000000, B00000000, B00001110, B00000000, B00000000, B00000000},
{B00000000, B00000000, B00000000, B00000000, B00000000, B00000000, B00000000, B00000100},
{B00000001, B00000010, B00000010, B00000100, B00000100, B00001000, B00001000, B00010000},
```

```

{B00001110, B00010001, B00010011, B00010001, B00010101, B00010001, B00011001, B00001110},
{B00000100, B00001100, B00010100, B00000100, B00000100, B00000100, B00000100, B00011111},
{B00001110, B00010001, B00010001, B00000010, B00000100, B00001000, B00010000, B00011111},
{B00001110, B00010001, B00000001, B00001110, B00000001, B00000001, B00010001, B00001110},
{B00010000, B00010000, B00010100, B00010100, B00011111, B00000100, B00000100, B00000100},
{B00011111, B00010000, B00010000, B00011110, B00000001, B00000001, B00000001, B00011110},
{B00000111, B00001000, B00010000, B00011110, B00010001, B00010001, B00010001, B00001110},
{B00011111, B00000001, B00000001, B00000001, B00000010, B00000100, B00001000, B00010000},
{B00001110, B00010001, B00010001, B00001110, B00010001, B00010001, B00010001, B00001110},
{B00001110, B00010001, B00010001, B00001111, B00000001, B00000001, B00000001, B00000001},
{B00000000, B00000100, B00000100, B00000000, B00000000, B00000100, B00000100, B00000000},
{B00000000, B00000100, B00000100, B00000000, B00000000, B00000100, B00000100, B00001000},
{B00000001, B00000010, B00000100, B00001000, B00001000, B00000100, B00000010, B00000001},
{B00000000, B00000000, B00000000, B00011110, B00000000, B00011110, B00000000, B00000000},
{B00010000, B00001000, B00000100, B00000010, B00000010, B00000100, B00001000, B00010000},
{B00001110, B00010001, B00010001, B00000010, B00000100, B00000100, B00000000, B00000100},
{B00001110, B00010001, B00010001, B00010101, B00010101, B00010001, B00010001, B00011110},
{B00001110, B00010001, B00010001, B00010001, B00011111, B00010001, B00010001, B00010001},
{B00011110, B00010001, B00010001, B00011110, B00010001, B00010001, B00010001, B00011110},
{B00000111, B00001000, B00010000, B00010000, B00010000, B00010000, B00001000, B00000111},
{B00011100, B00010010, B00010001, B00010001, B00010001, B00010001, B00010010, B00011100},
{B00011111, B00010000, B00010000, B00011110, B00010000, B00010000, B00010000, B00011111},
{B00011111, B00010000, B00010000, B00011110, B00010000, B00010000, B00010000, B00010000},
{B00001110, B00010001, B00010000, B00010000, B00010111, B00010001, B00010001, B00001110},
{B00010001, B00010001, B00010001, B00011111, B00010001, B00010001, B00010001, B00010001},
{B00011111, B00000100, B00000100, B00000100, B00000100, B00000100, B00000100, B00011111},
{B00011111, B00000100, B00000100, B00000100, B00000100, B00000100, B00010100, B00001000},
{B00010001, B00010010, B00010100, B00011000, B00010100, B00010010, B00010001, B00010001},
{B00010000, B00010000, B00010000, B00010000, B00010000, B00010000, B00010000, B00011111},
{B00010001, B00011011, B00011111, B00010101, B00010001, B00010001, B00010001, B00010001},
{B00010001, B00011001, B00011001, B00010101, B00010101, B00010011, B00010011, B00010001},
{B00001110, B00010001, B00010001, B00010001, B00010001, B00010001, B00010001, B00001110},
{B00011110, B00010001, B00010001, B00011110, B00010000, B00010000, B00010000, B00010000},
{B00001110, B00010001, B00010001, B00010001, B00010001, B00010101, B00010011, B00001111},
{B00011110, B00010001, B00010001, B00011110, B00010100, B00010010, B00010001, B00010001},
{B00001110, B00010001, B00010000, B00001000, B00000110, B00000001, B00010001, B00001110},
{B00011111, B00000100, B00000100, B00000100, B00000100, B00000100, B00000100, B00000100},
{B00010001, B00010001, B00010001, B00010001, B00010001, B00010001, B00010001, B00001110},
{B00010001, B00010001, B00010001, B00010001, B00010001, B00010001, B00001010, B00000100},
{B00010001, B00010001, B00010001, B00010001, B00010001, B00010101, B00010101, B00001010},
{B00010001, B00010001, B00001010, B00000100, B00000100, B00001010, B00010001, B00010001},
{B00010001, B00010001, B00001010, B00000100, B00000100, B00000100, B00000100, B00000100},

```



```
{B00011111, B00000001, B00000010, B00000100, B00001000, B00010000, B00010000, B00011111},
{B00001110, B00001000, B00001000, B00001000, B00001000, B00001000, B00001000, B00001110},
{B00010000, B00001000, B00001000, B00000100, B00000100, B00000010, B00000010, B00000001},
{B00001110, B00000010, B00000010, B00000010, B00000010, B00000010, B00000010, B00001110},
{B00000100, B00001010, B00010001, B00000000, B00000000, B00000000, B00000000, B00000000},
{B00000000, B00000000, B00000000, B00000000, B00000000, B00000000, B00000000, B00011111},
{B00001000, B00000100, B00000000, B00000000, B00000000, B00000000, B00000000, B00000000},
{B00000000, B00000000, B00000000, B00001110, B00010010, B00010010, B00010010, B00001111},
{B00000000, B00010000, B00010000, B00010000, B00011100, B00010010, B00010010, B00011100},
{B00000000, B00000000, B00000000, B00001110, B00010000, B00010000, B00010000, B00001110},
{B00000000, B00000001, B00000001, B00000001, B00000111, B00001001, B00001001, B00000111},
{B00000000, B00000000, B00000000, B00011100, B00010010, B00011110, B00010000, B00001110},
{B00000000, B00000011, B00000100, B00000100, B00000110, B00000100, B00000100, B00000100},
{B00000000, B00001110, B00001010, B00001010, B00001110, B00000010, B00000010, B00001100},
{B00000000, B00010000, B00010000, B00010000, B00011100, B00010010, B00010010, B00010010},
{B00000000, B00000000, B00000100, B00000000, B00000100, B00000100, B00000100, B00000100},
{B00000000, B00000010, B00000000, B00000010, B00000010, B00000010, B00000010, B00001100},
{B00000000, B00010000, B00010000, B00010100, B00011000, B00011000, B00010100, B00010000},
{B00000000, B00010000, B00010000, B00010000, B00010000, B00010000, B00010000, B00001100},
{B00000000, B00000000, B00000100, B00000000, B00000100, B00000100, B00000100, B00000100},
{B00000000, B00000010, B00000000, B00000010, B00000010, B00000010, B00000010, B00001100},
{B00000000, B00010000, B00010000, B00010000, B00010000, B00010000, B00010000, B00001100},
{B00000000, B00000000, B00000000, B00001010, B00010101, B00010001, B00010001, B00010001},
{B00000000, B00000000, B00000000, B00010100, B00011010, B00010010, B00010010, B00010010},
{B00000000, B00000000, B00000000, B00001100, B00010010, B00010010, B00010010, B00001100},
{B00000000, B00011100, B00010010, B00010010, B00011100, B00010000, B00010000, B00010000},
{B00000000, B00001110, B00010010, B00010010, B00001110, B00000010, B00000010, B00000001},
{B00000000, B00000000, B00000000, B00001010, B00001100, B00001000, B00001000, B00001000},
{B00000000, B00000000, B00001110, B00010000, B00001000, B00000100, B00000010, B00011110},
{B00000000, B00010000, B00010000, B00011100, B00010000, B00010000, B00010000, B00001100},
{B00000000, B00000000, B00000000, B00010010, B00010010, B00010010, B00010010, B00001100},
{B00000000, B00000000, B00000000, B00010001, B00010001, B00010001, B00001010, B00000100},
{B00000000, B00000000, B00000000, B00010001, B00010001, B00010001, B00010101, B00001010},
{B00000000, B00000000, B00000000, B00010001, B00001010, B00000100, B00001010, B00010001},
{B00000000, B00000000, B00010001, B00001010, B00000100, B00001000, B00001000, B00010000},
{B00000000, B00000000, B00000000, B00011111, B00000010, B00000100, B00001000, B00011111},
{B00000010, B00000100, B00000100, B00000100, B00001000, B00000100, B00000100, B00000010},
{B00000100, B00000100, B00000100, B00000100, B00000100, B00000100, B00000100, B00000100},
{B00001000, B00000100, B00000100, B00000100, B00000010, B00000100, B00000100, B00001000},
{B00000000, B00000000, B00000000, B00001010, B00011110, B00010100, B00000000, B00000000}
};
```

```
void clearDisplay() {
    for (byte x=0; x<8; x++) {
        buffer[x] = B00000000;
    }
}
```

```

    }
    screenUpdate();
}

void initMAX7219() {
    pinMode(DataPin, OUTPUT);
    pinMode(LoadPin, OUTPUT);
    pinMode(ClockPin, OUTPUT);
    clearDisplay();
    writeData(B00001011, B00000111); //扫描极限设置为 0 到 7
    writeData(B00001001, B00000000); //小数点模式关闭
    writeData(B00001100, B00000001); //设置 shutdown 寄存器模式为正常状态
    intensity(15); //值只能在 0 到 15 之间 (4 位)
}

void intensity(int intensity) {
    writeData(B00001010, intensity); //B0001010 是亮度寄存器
}

void writeData(byte MSB, byte LSB) {
    byte mask;
    digitalWrite(LoadPin, LOW); //设置 LoadPin 引脚准备接收数据
    //送出 MSB
    for (mask = B10000000; mask>0; mask >>= 1) { //通过比特掩码循环
        digitalWrite(ClockPin, LOW);
        if (MSB & mask){ //如果按位与的结果是 true
            digitalWrite(DataPin, HIGH); //送出 1
        }
        else{ //如果按位与的结果是 false
            digitalWrite(DataPin, LOW); //送出 0
        }
        digitalWrite(ClockPin, HIGH); //时钟设置为 HIGH, 数据引脚获得输入
    }
    //送出 LSB 数据
    for (mask = B10000000; mask>0; mask >>= 1) { //通过比特掩码循环
        digitalWrite(ClockPin, LOW);
        if (LSB & mask){ //如果按位与的结果是 true
            digitalWrite(DataPin, HIGH); //送出 1
        }
        else{ //如果按位与的结果是 false
            digitalWrite(DataPin, LOW); //送出 0
        }
    }
}

```

```

        digitalWrite(ClockPin, HIGH); //时钟设置为 HIGH, 数据输入
    }
    digitalWrite(LoadPin, HIGH); //锁存数据
    digitalWrite(ClockPin, LOW);
}

void scroll(char myString[], int speed) {
    byte firstChrRow, secondChrRow;
    byte ledOutput;
    byte chrPointer = 0; //初始化字符串位置指针
    byte Char1, Char2; //这两个字符将要显示
    byte scrollBit = 0;
    byte strLength = 0;
    unsigned long time;
    unsigned long counter;

    //增加计数器直到到达字符串结尾
    while (myString[strLength]) {strLength++;}

    counter = millis();

    while (chrPointer < (strLength-1)) {
        time = millis();
        if (time > (counter + speed)) {
            Char1 = myString[chrPointer];
            Char2 = myString[chrPointer+1];
            for (byte y= 0; y<8; y++) {
                firstChrRow = pgm_read_byte(&font[Char1 -
32][y]);
                secondChrRow = (pgm_read_byte(&font[Char2 -
32][y])) << 1;
                ledOutput = (firstChrRow << scrollBit) |
(secondChrRow >> (8 - scrollBit) );
                buffer[y] = ledOutput;
            }
            scrollBit++;
            if (scrollBit > 6) {
                scrollBit = 0;
                chrPointer++;
            }
            counter = millis();
        }
    }
}

```

```

    }
}

void screenUpdate() {
    for (byte row = 0; row < 8; row++) {
        writeData(row+1, buffer[row]);
    }
}

void setup() {
    initMAX7219();
    Timer1.initialize(10000);          //初始化 Timer1 设置中断周期
    Timer1.attachInterrupt(screenUpdate);
}

void loop() {
    clearDisplay();
    scroll(" BEGINNING ARDUINO ", 45);
    scroll(" Chapter 7 - LED Displays ", 45);
    scroll(" HELLO WORLD!!! :) ", 45);
}

```

当上传代码后，将能看到信息在显示器上滚动。

硬件回顾

为了更容易理解代码，首先需要知道 MAX7219 芯片是如何工作的，因此在看代码之前让我们先看一下硬件。

MAX7219 操作起来比移位寄存器简单，使用移位寄存器必须以串行的方式逐个输入数据，一个完整的 16 位数据必须一次下载到芯片。这个芯片使用起来很简单，它只需要连接 Arduino 的 3 个引脚。Arduino 数字引脚 2 连到 MAX7219 的引脚 1 上，这是数据输入引脚。Arduino 数字引脚 3 连到 MAX7219 的引脚 12 上，结束传输，完成数据加载。Arduino 数字引脚 4 连到 MAX7219 的引脚 13，这是时钟引脚。MAX7219 引脚图见图 7-5。

LOAD 引脚设置为低电平，第一个为高或低的比特数据送到 DIN 引脚。CLK 引脚设置在 LOW 和 HIGH 之间振荡。在时钟脉冲的上升沿 DIN 引脚上的数据移入内部寄存器。之后时钟脉冲变成低，为下一个数据进入 DIN 引脚做好准备，再后这个输入数据的过程重复。全部 16 位比特数据已经输入寄存器之后，时钟下降上升 16 次，LOAD 引脚最后设置为

HIGH，将数据锁存到寄存器中，图 7-6 是 MAX7219 数据手册中的时序图。时序图显示把数据 D0 到 D15 送入芯片时 3 个引脚是如何动作的。DOUT 引脚，也就是引脚 24，在本项目中不用，但是如果有两个及两个以上的 MAX7219 芯片级联在一起，第一个芯片的 DOUT 引脚要连到第二个芯片的 DIN 引脚上，依此类推。数据在时钟脉冲的下降沿从 DOUT 引脚上输出。

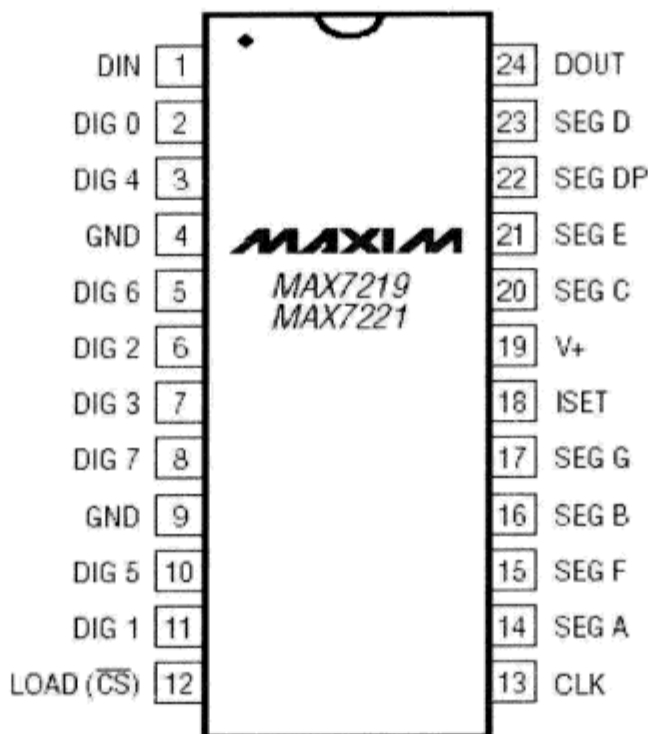


图 7-5 MAX7219 的引脚图

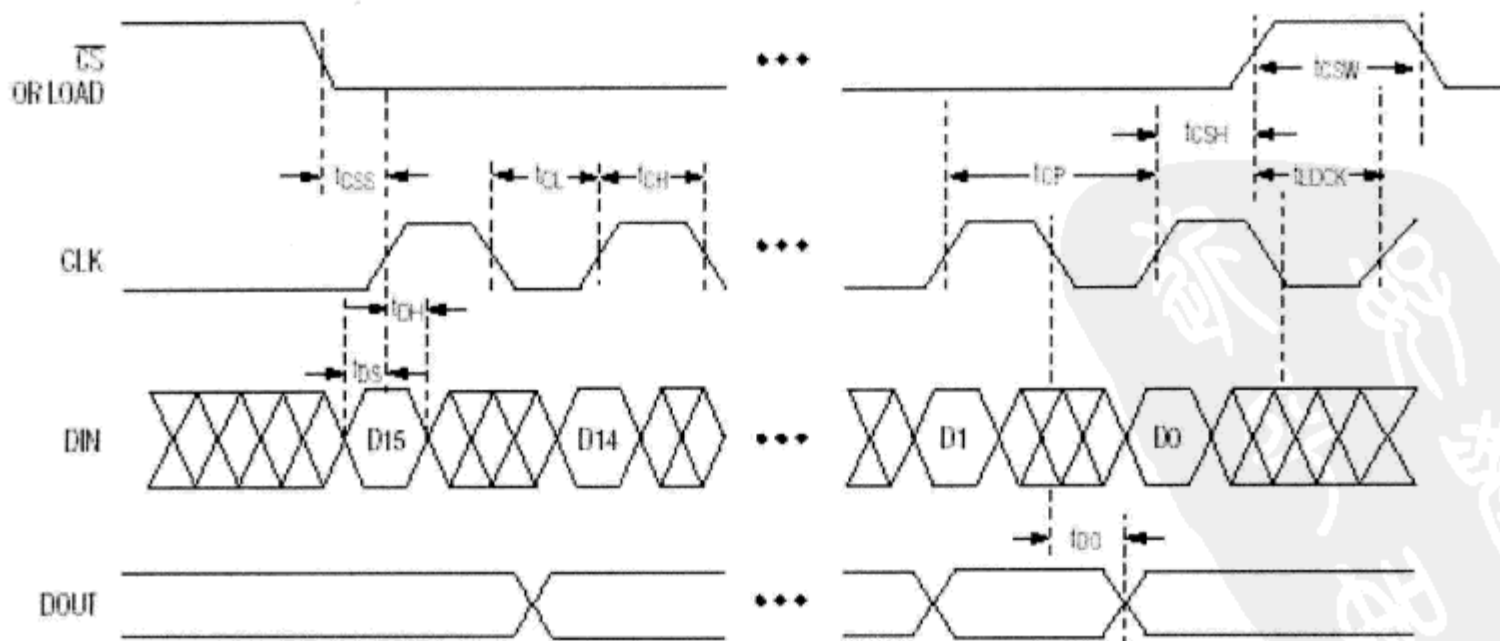


图 7-6 MAX7219 的时序图

为了使 Arduino 能正确地送数据到芯片中，需要在代码中按这个时间序列设置语句。

这个芯片可以接收高达 100mA 的电流，这对于大多数点阵显示器是足够的。如果希望阅读 MAX7219 的数据说明书，可以从 Maxim 公司网站 <http://datasheets.maxim-ic.com/en/ds/MAX7219-MAX7221.pdf> 下载。

这个芯片以 16 位形式接收数据。D15 或 MSB（最高有效位）首先送入，因此送入的数据是从 D15 到 D0，前 4 个比特是不用去关心的，它们没有被芯片使用，因此可以是任意值。接下来的 4 个比特组成寄存器地址，最后 8 个比特组成数据。表 7-4 显示了串行数据形式，表 7-5 显示了寄存器地址。

表 7-4 MAX7219 的串行数据形式（16 位）

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
X X	X X			地址				MSB		DATA			LSB		
										X					
										X					
										X					
										X					
										X					
										X					
										X					

表 7-5 MAX7219 的寄存器地址图

寄 存 器	地 址					HEX 码
	D15~D12	D11	D10	D9	D8	
No-Op	X	00		00		0xX0
Digit0	X	00		01		0xX1
Digit1	X	00		10		0xX2
Digit2	X	00		11		0xX3
Digit3	X	01		00		0xX4
Digit4	X	01		01		0xX5
Digit5	X	01		10		0xX6
Digit6	X	01		11		0xX7
Digit7	X	10		00		0xX8
Decode Mode(小数点模式寄存器)	X	10		01		0xX9
Intnesity(亮度寄存器)	X	10		10		0xXA
Scan Limit(扫描长度寄存器)	X	10		11		0xXB
Shutdown(关闭模式寄存器)	X	11		00		0xXC
DisplayTest(显示测试寄存器)	X	11		11		0xFF

例如，可以从表 7-5 的寄存器地址表中看到，亮度寄存器的地址是二进制 1010，亮度寄存器设置显示的亮度，从最暗 0 到最亮 15（即 B0000 到 B1111）。要设置亮度为 15（最大值），应该送出如下的 16 比特，标志位（最左端的）要首先送出，标志位（最右端的）最后送出（例如，用二进制表示的命令码如下）：

```
0000101000001111
```

低 8 位数据中的低 4 位是值 B1111，它是内部寄存器的地址。前 8 比特中的高 4 位是无关紧要的，因此送入 B0000。LSB 是第二个送入寄存器的 8 比特数据。在这个例子里，希望把值 B1111 送入亮度寄存器，前 4 比特仍然是无关紧要的，因此送入 B0000，通过送出这 16 比特到芯片，可以设置显示器亮度为最大。要送出的全部 16 比特数值是 B00010100001111，但是首先需要给 MAX7219 送入 MSB（最高有效位），之后是 LSB（最低有效位），所以要送出的数用二进制表示是 B111100000101000。

另外一个地址是扫描长度，记住 MAX7219 是给 7 段码显示器工作设计的（见图 7-7）。

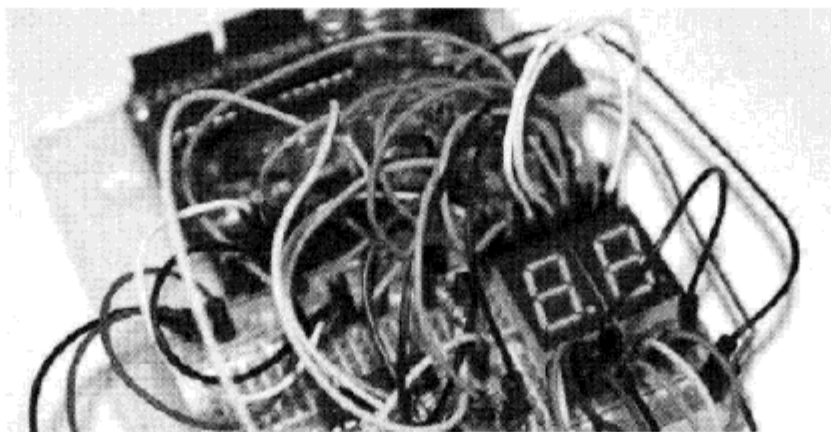


图 7-7 7 段码 LED 显示器（图片由 Tony Jewell 提供）

扫描长度定义有多少 7 段码显示器连接在 MAX7219 上，在你的例子里，不使用 7 段码显示器，而是 8×8 点阵显示器。扫描长度这个数字反应在你的显示器的列上。你希望所有 8 列都能用，因此扫描长度寄存器应送入 B00000111（第 0 号到第 7 号，7 是二进制中的 B111）。

小数点模式寄存器只有在使用 7 段码显示器时才是有用的。因此把小数点模式寄存器设置为 B00000000，关闭小数点。

最后，将设置关闭模式寄存器为 B00000001，确保它在正常模式下工作，不是关闭模式。如果设置关闭模式寄存器为 B00000000，那么，当前 LED 全部接地，这使显示器变黑。

关于 MAX7219 芯片的进一步信息可以阅读它的数据说明书，只需要阅读和你的项目

有关的那部分内容，你会发现相比第一次见到它时它其实是非常容易理解的。

现在已经理解 MAX7219 是如何工作的（希望如此），下面让我们看一下代码，看它是如何使显示器显示滚动字符的。

代码回顾

打开集成开发环境后首先要做的是加载两个将在代码中使用的头文件：

```
#include <avr/pgmspace.h>
#include <TimerOne.h>
```

第一个链接库是 `pgmspace` 或程序空间实用工具库。这个链接库中的函数允许程序使用程序存储区或 Flash 存储区中的数据。采用 Atmega328 芯片的 Arduino 有 32KB 的 Flash 存储器（2KB 用做引导程序存储区，因此有 30KB 存储区给用户使用）。Arduino Mega 有 128KB 的 Flash 存储器，4KB 用做引导程序存储区，程序存储空间精确地说是用户程序将要存储的空间。通过 `pgmspace` 库可以使用 Flash 中没有用到的自由空间，在程序存储空间中你要存储大量的二维数组，这些数组存储字符的二进制显示样式。

第二个链接库是 `TimerOne` 链接库，它在项目 19 中第一次被使用。之后，声明连接到 MAX7219 的三个数字引脚：

```
int DataPin = 2; //MAX 上的引脚 1
int LoadPin = 3; //MAX 上的引脚 12
int ClockPin = 4; //MAX 上的引脚 13
```

之后，生成一个有 8 个元素的字节类型的数组 `buffer`：

```
byte buffer[8];
```

这个数组存储二进制形式的帧，每个帧决定当显示器激活时哪个 LED 开或关。

之后是一个大的字节型的二维数组：

```
static byte font[][8] PROGMEM = {
//32~126 之间的可打印的 ASCII 字符
{B00000000, B00000000, B00000000, B00000000, B00000000, B00000000, B00000000, B00000000},
{B00000100, B00000100, B00000100, B00000100, B00000100, B00000100, B00000000, B00000100},
...依此类推
```

这个数组存储二进制格式的在显示器上显示的字符样式。它是一个 `static byte` 类型的二维数组。在数组后面增加了 `PROGMEM` 声明。这是 `Program Space Utilities` 库中的一个功能，它告诉编译器，存储这个数组到 `Flash` 存储器内而不是 `SRAM` 存储器（静态随机存储器）。

`SRAM` 是 `Atmega` 芯片上的一个存储空间，它一般用来存储项目中用到的变量和字符串。当使用时它们被从程序空间复制到 `SRAM` 中。然而，`font` 数组用来存储 96 个字符的帧，每个帧由 8 个字节组成。数组有 96×8 个元素，即总共是 768 个元素，并且每个元素是一个字节（8 比特），那么这个数组大约总共用 768 字节。`Atmega328` 只有 2K 字节或约 2000 字节的存储空间给变量使用。一旦增加其他变量或在程序中用的字符串，就会遇到溢出存储空间的风险。

`Arduino` 没有办法警告溢出存储空间，遇到这种情况它会崩溃。为了阻止这种事情发生，要存储这个数组到 `Flash` 存储器而不是 `SRAM` 存储器。因为这里有更多的存储空间给用户使用。这个项目大约有 2800 个字节，而这个数组大小在 800 字节以下，因此使用大约 3.6K 字节，而 `Flash` 存储器有 30K 字节的存储空间给用户使用。

之后，开始生成程序中需要的各种各样的函数，第一个函数是清除显示器数据，清除将要显示在点阵显示器上的存储在缓冲队列中的任何数据。`clearDisplay()` 函数只是循环地将矩阵的 8 个元素设置为 0，因此没有 LED 点亮，所以显示器是黑屏状态的。之后调用 `screenUpdate()` 函数，它在点阵显示器上显示存储在 `buffer[]` 数组中的图样。在这个例子中，缓冲区 `buffer` 中除了存储 0 外什么也没有，因此什么也不会显示。

```
void clearDisplay() {
    for (byte x=0; x<8; x++) {
        buffer[x] = B00000000;
    }
    screenUpdate();
}
```

下一个函数 `initMAX7219()`，做使用 `MAX7219` 芯片前的准备工作。首先是将 3 个引脚设置为 `OUTPUT` 状态：

```
void initMAX7219() {
    pinMode(DataPin, OUTPUT);
    pinMode(LoadPin, OUTPUT);
}
```

```
pinMode(ClockPin, OUTPUT);
```

之后显示器被清空:

```
clearDisplay();
```

扫描长度设置为 7, 小数点模式关闭, 关闭寄存器设置为正常操作状态:

```
writeData(B00001011, B00000111); //扫描极限设置为 0 到 7
writeData(B00001001, B00000000); //小数点模式关闭
writeData(B00001100, B00000001); //设置 shutdown 寄存器模式为正常状态
```

之后通过调用 `intensity()` 函数把亮度设置为最大:

```
intensity(15); //值只能在 0 到 15 之间 (4 位)
```

下一个是 `intensity()` 函数本身, 它只是把传给它的值通过调用 `writeData` 函数写到内部寄存器:

```
void intensity(int intensity) {
    writeData(B00001010, intensity); //B0001010 是亮度寄存器
}
```

下一个函数做了许多艰难的工作。它的工作是每次写一个比特到 MAX7219 芯片。这个函数需要两个参数, 都是字节型的, 它们组成了 16 位数的高 8 位 (不是比特) 和低 8 位:

```
void writeData(byte MSB, byte LSB) {
```

声明一个叫做 `mask` 的字节型变量:

```
byte mask;
```

它作为掩码用来选择需要送出的比特 (参见项目 17)。

之后, `LoadPin` 引脚设置为 LOW。这没有锁定数据到芯片寄存器中, 做好接收新数据的准备:

```
digitalWrite(LoadPin, LOW); //设置 LoadPin 引脚准备接收数据
```


现在需要送出 16 位数的高 8 位给芯片，最左端的比特首先送出。为了做到这个，使用两个 for 循环。一个送出高 8 位，另一个送出低 8 位。这个循环使用比特掩码获得所有 8 个比特数据。使用按位与功能决定当前比特是 1 还是 0，相应地来设置数字引脚为 HIGH 或 LOW。时钟引脚设置为 LOW，之后，将 HIGH 或 LOW 值写入数据引脚：

```
//送出 MSB
for (mask = B10000000; mask>0; mask >>= 1) { //通过比特掩码循环
    digitalWrite(ClockPin, LOW);
    if (MSB & mask){ //如果按位与的结果是 true
        digitalWrite(DataPin,HIGH); //送出 1
    }
    else{ //如果按位与的结果是 false
        digitalWrite(DataPin,LOW); //送出 0
    }
    digitalWrite(ClockPin, HIGH); //时钟设置为 HIGH，数据引脚获得输入
}
//送出 LSB 数据
for (mask = B10000000; mask>0; mask >>= 1) { //通过比特掩码循环
    digitalWrite(ClockPin, LOW);
    if (LSB & mask){ //如果按位与的结果是 true
        digitalWrite(DataPin,HIGH); //送出 1
    }
    else{ //如果按位与的结果是 false
        digitalWrite(DataPin,LOW); //送出 0
    }
    digitalWrite(ClockPin, HIGH); //时钟设置为 HIGH，数据输入
}
```

最后，LoadPin 引脚设为 HIGH，保证 16 比特数据锁进芯片寄存器，ClockPin 引脚设置为 LOW，因为最后一个脉冲是高电平（时钟必须振荡在 HIGH 和 LOW 来确保数据成功输入）：

```
digitalWrite(LoadPin, HIGH); //锁存数据
digitalWrite(ClockPin, LOW);
```

之后是 scroll()函数，这个函数显示字符串在显示器上，它需要两个参数，第一个是要显示的字符串，第二个是字符刷新动作发生的速度，用毫秒表示：

```
void scroll(char myString[], int speed) {
```

之后, 设置两个 `byte` 变量, 它们存储构成特定的字符显示的 8 行二进制模式中的一个:

```
byte firstChrRow, secondChrRow;
```

另外一个字节被声明叫做 `ledOutput`, 这个变量存储在第一个字符的比特帧和第二个字符的比特帧之间的计算结果。它将决定哪个 LED 开或关 (这将在后面简要说明):

```
byte ledOutput;
```

另外, 又声明了一个 `byte` 型的变量, 叫做 `chrPointer`, 并且初始化为 0, `chrPointer` 将存储当前显示的字符在字符串中的位置。变量的值开始为 0, 之后增加到字符串的长度:

```
byte chrPointer = 0; //初始化字符串位置指针
```

声明另外两个字节变量, 它们将存储当前字符和字符串中的下一个字符:

```
byte Char1, Char2; //这两个字符将被显示
```

这两个变量不同于 `firstchrRow` 和 `secondchrRow`, 因为它们存储的是要显示的字符的 ASCII 码 (美国信息传送标准代码) 值, 和字符串中的下一个字符的 ASCII 码。`FirstchrRow` 和 `SecondChrRow` 存储组成要显示的字符的二进制形式的帧。

所有可以显示在计算机屏幕上或通过串行口发送的字符、数字、符号等都有 ASCII 码值。ASCII 码值是一个简单的索引标记, 表明字符在 ASCII 码表中的位置。字符 0 到 31 是控制字符, 用户不会用到这些控制字符因为它们不能显示在点阵显示器上。要用 ASCII 字符 32 到 196, 这是 95 个可打印字符。这些字符的起始位置是 32, 它是一个空格, 直到 126, 它是波浪线 (~) 符号。可打印字符如表 7-6 所示。

表 7-6 可打印 ASCII 字符

!	"	#	\$	%	&	'	()	*	+	,	-	.	/	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?	@
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_	'
a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	{		}	~		

声明另外一个字节并且初始化为 0, 它要存储当前设置的字母有多少需要移位给从右向左卷动的二进制帧:

```
byte scrollBit = 0;
```

另外一个字节存储字符串长度，它初始化为 0:

```
byte strLength = 0;
```

之后，声明两个无符号长整型字符变量，它们将要存储以毫秒为单位的从 Arduino 芯片启动或重启开始计算的当前时间，另外一个存储同样的值，但是这个时间是在一个 while 循环运行之后。联合这两个数保证比特移位只发生在一个以毫秒计算的特定的时间之后，因此保证字符滚动以一个可以读的速度进行：

```
unsigned long time;
unsigned long counter;
```

现在需要知道字符串中有多少个字符。有几种方法。但是在这个例子里，只要设置一个 while 循环来检查当前数组索引中是否是一个数据。如果是数据，strLength（初始化为 0）加 1，之后重复这个循环，直到条件 myString[strlength]为假，如已经没有字符在字符串中。之后，已经在每个循环中加 1 的变量 strlength 将存储字符串的长度：

```
while (myString[strLength]) {strLength++;}
```

接下来，定义一个变量 counter 等于 millis()的返回值。在项目 4 中遇到过 millis()函数，它以毫秒形式存储自 Arduino 运行或重启以来的时间：

```
counter = millis();
```

接下来是一个 while 循环，它的循环条件是当前字符位置小于字符串长度减 1:

```
while (chrPointer < (strLength-1)) {
```

变量 time 设置为 millis()函数的返回值：

```
time = millis();
```

之后一个 if 语句检查当前时间是否大于存储在 speed 中的最后一次发出脉冲以来的时间，如 45 毫秒，如果是，运行它中的代码块：

```
if (time > (counter + speed)) {
```

Char1 装载的是 mystring 数组中 chrPointer 指定的字符的 ASCII 码值, Char2 存储随后的那一个字符的 ASCII 码值:

```
Char1 = myString[chrPointer];
Char2 = myString[chrPointer+1];
```

一个 for 循环重复 LED 显示器 8 个行中的每一行:

```
for (byte y= 0; y<8; y++) {
```

现在读帧数组并且存放 8 个帧中的当前帧到 firstchrRow, 第二个帧放到 secondchrRow 中。注意, 帧数组存储二进制格式的帧, 它组成 ASCII 码表中的字符, 但只是可打印字符——从 ASCII 码 32 到 ASCII 码 126。数组的第一个元素是字符的 ASCII 码 (最小是 32, 因为不使用字符 0 到 31), 数组的第二个元素存储 8 行比特帧组成的字符。例如, 字母 A 和 Z 相应的 ASCII 码是 65 和 90, ASCII 码减去 32 就得到相应字符的帧数组索引。

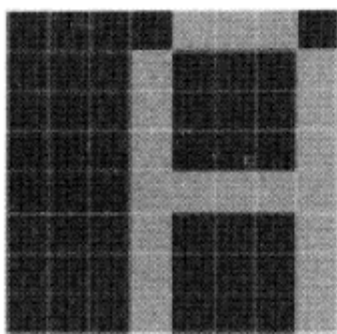
因此字母 A 的 ASCII 代码是 65, 存储在数组索引值为 33 的元素中, 数组的这个索引值下的第二维存储组成的这个字母的 8 比特帧, 字母 Z 的 ASCII 码是 90, 所以它在数组中的索引值是 58。font[33][0]到 font[33][7]组成字母 A 的帧结构, 如下:

```
{B00001110, B00010001, B00010001, B00010001, B00011111, B00010001, B00010001, B00010001},
```

把这些数据竖直排列起来, 可以看得更清楚一些, 如下:

```
B00001110
B00010001
B00010001
B00010001
B00011111
B00010001
B00010001
B00010001
```

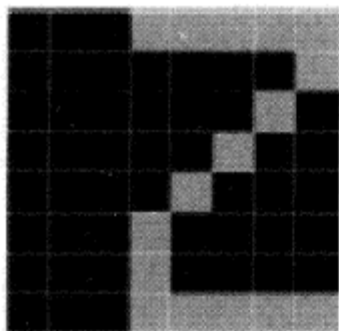
如果仔细看, 将看到如下的帧组成字母 A:



对于字母 Z，数组的数据是

```
B00011111
B00000001
B00000010
B00000100
B00001000
B00010000
B00010000
B00010000
B00011111
```

相应的 LED 图案如下：



为了读二进制形式的图案，需要获得 font 数组，它存储在程序空间中而不是平常的 SRAM 空间中。为了获得程序空间中的数据，需要使用 pgmspace 链接库中的一个函数 `pgm_read_byte`。

```
firstChrRow = pgm_read_byte(&font[Char1 - 32][y]);
secondChrRow = (pgm_read_byte(&font[Char2 - 32][y])) << 1;
```

当进入程序空间后，就可以获得存储在 Flash 存储器中的数据。为了获得 Flash 存储器中的数据，还需要知道数据存储在存储器中的地址（在存储器中的每一个存储位置都有一个独一无二的地址）。

为了获得地址，要在变量的前面使用“&”符号。当在变量前面加“&”符号时，表示不读变量中的数，而是读变量的地址。`pgm_read_byte` 函数需要知道要读的数据在 Flash 存

存储器中的地址，因此，放“&”符号在 `font[Char1-32][y]` 前，成为 `pgm_read_byte (&font[Char1-32][y])`，它的意思就是读在程序空间中存储的 `font[Char1-32][y]` 的地址。

`secondChrRow` 的值向左移动 1 位使在两个字母之间的空格变小，使得更容易在显示器上分辨出两个字母。这是因为在所有字母的左边有 3 个比特空格没有使用。可以把字母的帧结构按位向左移动两位，之后关闭帧，但是这样做字母会变得难以阅读。

下面一行载入相关行的比特帧模式，之后用 `ledOutput` 函数向引脚输出：

```
ledOutput = (firstChrRow << scrollBit) | (secondChrRow >> (8 - scrollBit));
```

因为想让字母从右向左滚动，所以把 `firstChrRow` 中存储的字母的帧结构按位左移 `scrollBit` 表示的次数，把 `secondChrRow` 中存储的字母的帧结构向右移动 `8-scrollBit` 次。之后用逻辑或把这两个结果合在一起形成在显示器上显示需要的 8 比特帧。例如，如果要显示的字母是 A 和 Z，那么两个的帧是：

```
B00001110 B00011111
B00010001 B00000001
B00010001 B00000010
B00010001 B00000100
B00011111 B00001000
B00010001 B00010000
B00010001 B00010000
B00010001 B00011111
```

因此，当 `scrollBit` 为 5 时对以上数据行的第一行进行计算。例如，字母 A 的第一行向左滚动 5 位，字母 Z 的第一行向右滚动 (8-5) 位，结果是：

```
B11000000 B00000011
```

也就是这个字母 A 顶行左移 5 次，字母 Z 顶行左移 3 次 (8-5)，你可以看到左侧的数值表示字母 A 左移 5 位，右侧的数值表示字母 Z 右移 3 次。之后用逻辑或，也就是“|”，将两个模式合在一起生成：

```
B11000011
```

这就是你将获得的字母 A 和 Z 互相连接并左移 5 位的效果。

下一行代码载入运算后的屏幕缓冲器的行：

```
buffer[y]=ledOutput;
```

scrollBit 增加 1：

```
scrollBit++;
```

之后一个 if 语句检查 **scrollBit** 的值是否到 7，如果是，它将返回 0 并使 **chrPointer** 增加 1，因此在下一次调用函数时，它将显示下两组字母：

```
if (scrollBit > 6) {
    scrollBit = 0;
    chrPointer++;
}
```

最后，**counter** 的值刷新到 **millis()** 函数的返回值：

```
counter = millis();
```

screenUpdate() 函数只是把已经装载到缓冲数组 8 个元素中的 8 行帧数据写到芯片上，芯片转而把它显示在点阵显示器上：

```
void screenUpdate() {
    for (byte row = 0; row < 8; row++) {
        writeData(row+1, buffer[row]);
    }
}
```

设置这 6 个函数之后，最后是 **setup()** 函数和 **loop()** 函数。在 **setup()** 函数中，通过调用 **initMax7219()** 函数初始化芯片，设置并启动一个定时器，定时器中断时间为 10000 微秒，中断时激活 **screenUpdate()** 函数。如前所述，通过硬件中断保证 **screenUpdate()** 函数每 10000 微秒激活一次，不管什么程序正在运行。

```
void setup() {
    initMAX7219();
    Timer1.initialize(10000); //初始化 Timer1 设置中断周期
    Timer1.attachInterrupt(screenUpdate);
}
```

最后，程序的主循环只有 4 行，第一行是清空显示器，之后的三行调用 `scroll` 函数显示三行字符串，并且设置字符串滚动通过显示器。

```
void loop() {
    clearDisplay();
    scroll(" BEGINNING ARDUINO ", 45);
    scroll(" Chapter 7 - LED Displays ", 45);
    scroll(" HELLO WORLD!!! :) ", 45);
}
```

当然可以改变代码中的字符串，用来显示你希望表达的任何东西。项目 21 代码运行有点复杂。就像我在开始时说的那样，其实如果使用一个 LED 点阵显示链接库，这些艰难的工作都可以避免，这些库可以随意使用。但是使用外部链接库，你不可能学到 MAX7219 芯片的工作原理。这些技能可以转移到操作相关的任何其他外围芯片上，因为这些规律是非常相似的。

在下一个项目中要使用这些链接库，可以看到链接库使工作变得容易多了，让我们用你的显示器找一点小乐子。

项目 22——LED 点阵显示器——Pong 游戏

项目 21 很难，你可能在其中投入了很多精力。项目 22 使用一个点阵显示器、一个电位器和一些简单的代码做一个游戏。这次要使用许多可用的控制点阵显示器的链接库中的一个。可以看到使用这些链接库会使编写程序的工作变得很简单。

需要的元件

需要的元件与项目 21 的完全相同，只是增加了一个 $10\text{k}\Omega$ 的变阻器。

与项目 21 的元件相同……

$10\text{k}\Omega$ 变阻器



把元件连接起来

保持电路与项目 21 相同，并增加一个变阻器。变阻器的最左和最右引脚分别连接到地

和+5V，中间引脚连接到 Arduino 模拟引脚 5。

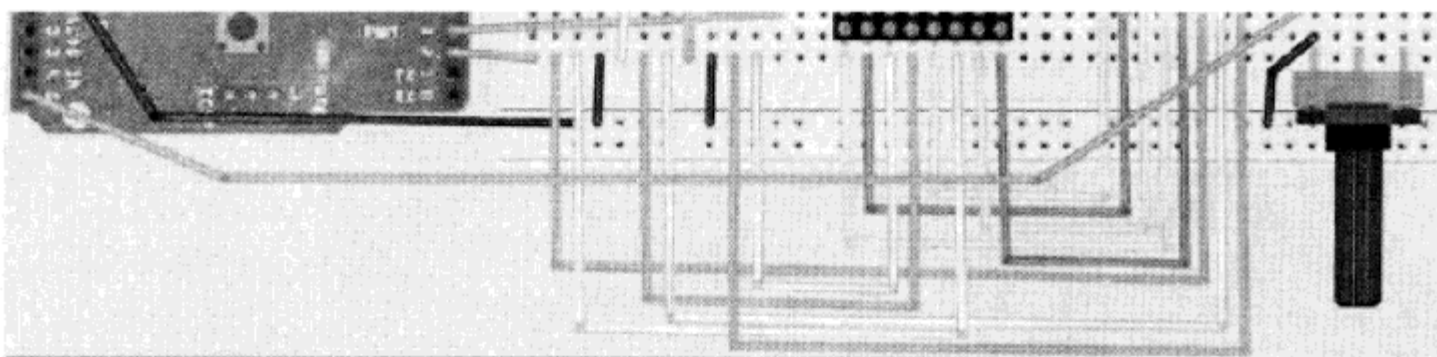


图 7-8 给项目 21 的电路增加一个变阻器

上传代码

从清单 7-4 中上传代码，当程序运行时，可以看到一个球从左边随机位置向右移动，使用变阻器控制拍子接到球，使它返回到墙。当球再次返回时，它的速度变得越来越快，直到你不能接住球。

如果没有用拍子接到球，屏幕闪烁，游戏重新开始。看看你在程序重新开始前能坚持多久。

清单 7-4 项目 22 的代码

```
//项目 22
#include "LedControl.h"

LedControl myMatrix = LedControl(2, 4, 3, 1); //建立一个点阵显示器实例

int column = 1, row = random(8)+1; //决定小球从哪里开始
int directionX = 1, directionY = 1; //确定小球移动的方向
int paddle1 = 5, paddle1Val; //拍子的引脚和数值
int speed = 300;
int counter = 0, mult = 10;

void setup()
{
    myMatrix.shutdown(0, false); //使能显示器
    myMatrix.setIntensity(0, 8); //设置显示器亮度为中间值
    myMatrix.clearDisplay(0); //清空显示器
    randomSeed(analogRead(0));
}
```

```

void loop()
{
    paddle1Val = analogRead(paddle1);
    paddle1Val = map(paddle1Val, 200, 1024, 1,6);
    column += directionX;
    row += directionY;
    if (column == 6 && directionX == 1 && (paddle1Val == row || paddle1Val+1
== row || paddle1Val+2 == row)) {directionX = -1;}
    if (column == 0 && directionX == -1 ) {directionX = 1;}
    if (row == 7 && directionY == 1 ) {directionY = -1;}
    if (row == 0 && directionY == -1 ) {directionY = 1;}
    if (column == 7) { oops();}
    myMatrix.clearDisplay(0); //清空显示器, 为动画的下一个帧做好准备
    myMatrix.setLed(0, column, row, HIGH);
    myMatrix.setLed(0, 7, paddle1Val, HIGH);
    myMatrix.setLed(0, 7, paddle1Val+1, HIGH);
    myMatrix.setLed(0, 7, paddle1Val+2, HIGH);
    if (!(counter % mult)) {speed -= 5; mult * mult;}
    delay(speed);
    counter++;
}

void oops() {
    for (int x=0; x<3; x++) {
        myMatrix.clearDisplay(0);
        delay(250);
        for (int y=0; y<8; y++) {
            myMatrix.setRow(0, y, 255);
        }
        delay(250);
    }
    counter=0; //重置所有值
    speed=300;
    column=1;
    row = random(8)+1; //选择新的开始位置
}

```

代码回顾

项目 22 的代码是非常简单的, 在项目 21 中做了相当艰难的工作后终于得到了一点休息。

首先，将 `LedControl.h` 链接库插入项目中。需要下载这个链接库并且把它放到库文件夹中，像以前做的那样。链接库文件，以及更进一步的信息可以在 www.arduino.cc/playground/Main/LedControl 网址中找到。

```
#include "LedControl.h"
```

之后，可以建立一个 `LedControl` 类的实例，如下：

```
LedControl myMatrix = LedControl(2, 4, 3, 1); //建立一个点阵显示器实例
```

这里建立的 `LedControl` 对象叫 `myMatrix`。`LedControl` 对象需要 4 个参数。头 3 个是 `MAX7219` 连接到 `Arduino` 上的引脚号，代号是 `DataIn`、`Clock` 和 `Load`。最后的参数是芯片的数量（可控制多于 1 个的 `MAX7219` 芯片和显示器）。

之后决定小球从哪一行开始移动，使用一个随机数确定行数。

```
int column = 1, row = random(8)+1; //决定小球从哪里开始
```

下面声明了两个整型数，决定小球前进的方向，如果数是正数，球将从左向右、从下向上运动。相反，如果是负数，球将向上面所述相反的方向运动。

```
int directionX = 1, directionY = 1; //确定小球移动的方向
```

确定哪一个引脚给拍子使用（变阻器），并且声明一个整型数来存储从模拟量引脚中读到的数值：

```
int paddle1 = 5, paddle1Val; //拍子的引脚和数值
```

声明用毫秒来表示的球速度变量：

```
int speed = 300;
```

之后，声明 `counter` 变量，并用 0 初始化变量，同样声明变量 `multiplier`，用 10 初始化变量：

```
int counter = 0, mult = 10;
```

`setup()`函数通过设置省电模式为 `false` 使显示器可用，显示器亮度设置在中间亮度处，

之后显示器清空，为游戏做好准备。在游戏开始之前，用一个随机值设置 `randomSeed`，这个值是从一个不用的模拟量引脚中读入的。

```
void setup()
{
    myMatrix.shutdown(0, false); //使能显示器
    myMatrix.setIntensity(0, 8); //设置显示器亮度为中间值
    myMatrix.clearDisplay(0); //清空显示器
    randomSeed(analogRead(0));
}
```

主循环从读拍子的模拟值开始：

```
paddle1Val = analogRead(paddle1);
```

之后，把这个值映射到 1 到 6 之间：

```
paddle1Val = map(paddle1Val, 200, 1024, 1, 6);
```

`map` 函数需要 5 个参数。第一个是需要映射的数，之后的是这个数的最小、最大值，再后是需要映射到的最小和最大值。在这个例子里，获得 `paddle1Val` 的值，它是从模拟引脚 5 中读到的电压，这个值在 0V 时为 0，5V 时为 1024，希望把这个数映射到 1 到 6 之间，因为这是拍子将显示在显示器上的行数。

行列坐标用 `directionX` 和 `directionY` 中的值进行加计算：

```
column += directionX;
row += directionY;
```

现在需要确定球是撞在了墙上还是撞在了拍子上。如果是撞到了这两样东西上，球弹回（例外是它穿过了 `paddle0`）。第一个 `if` 语句检查球是否撞到了拍子上，它通过检查球坐标的列是否在列 6 内并且（逻辑与 `&&`）是否已撞到了最左端、最右端的墙来做到这一点。

```
if (column == 6 && directionX == 1 && (paddle1Val == row || paddle1Val+1 == row || paddle1Val+2 == row)) {directionX = -1;}
```

这里三个条件决定球的方向的改变。第一个条件是球坐标的列是 6，第二个条件是球的运动方向是正（例如，左到右），第三个是球在 3 个点组成的拍子所在的行中的任意一

行。判断球是否碰到拍子是通过括号内的 3 个或（逻辑或||）运算完成的，首先计算这个逻辑结果，之后与其他两个条件进行逻辑与计算。

之后的三个 if 语句检查球是否撞到了顶、底或左边的墙，如果是则改变球的方向：

```
if (column == 0 && directionX == -1 ) {directionX = 1;}
if (row == 7 && directionY == 1 ) {directionY = -1;}
if (row == 0 && directionY == -1 ) {directionY = 1;}
```

如果球在第 7 列，很显然它没有碰到拍子，而是越过了拍子，如果是这种情况，调用 `oops()` 函数闪烁屏幕重新设置数值：

```
if (column == 7) { oops();}
```

之后，清空显示器，擦除之前的任何点：

```
myMatrix.clearDisplay(0); //清空显示器，为动画下一个帧做好准备
```

调用 `LedControl` 链接库中的 `.setLed` 函数把球画在相应的行和列位置。

```
myMatrix.setLed(0, column, row, HIGH);
```

`.setLed` 函数需要 4 个参数，第一个数是显示的地址，后面就是 x 和 y 坐标，最后的 `HIGH` 或 `LOW` 表示开或关，这个过程重复三次，在第 7 列和 `paddle1Val` 表示的行中（每次增加 1 个位）画 3 个点组成的拍子。

```
myMatrix.setLed(0, 7, paddle1Val, HIGH);
myMatrix.setLed(0, 7, paddle1Val+1, HIGH);
myMatrix.setLed(0, 7, paddle1Val+2, HIGH);
```

之后检查 `counter` 模除 `mult` 的结果是否为“否”（逻辑 NOT），如果是，时间间隔减 5，`mult` 变量乘以自己。模除是当用一个整数除以另外一个整数时得到的余数。在这个例子里用 `mult` 模除 `counter`，检查是否有余数。这保证只是在增加整数的时间后才加快球的速度。

```
if (!(counter % mult)) {speed -= 5; mult * mult;}
```

用毫秒表示的延时，延时时间是 `speed`，之后 `counter` 值增加 1：

```
delay(speed);
```

```
counter++;
}
```

最后，oops()函数中有一个 for 循环，在 for 循环中清空显示器，之后再点亮所有行，在两者之间延时 250 毫秒，这使所有的 LED 开关闪烁来表明球没被接到，游戏将重启。counter、speed 和 column 变量重置为它们的初始值，给行选择一个新的随机数。

```
void oops() {
  for (int x=0; x<3; x++) {
    myMatrix.clearDisplay();
    delay(250);

    for (int y=0; y<8; y++) {
      myMatrix.setRow(0, y, 255);
    }
    delay(250);
  }
  counter=0; //重置所有值
  speed=300;
  column=1;
  row = random(8)+1; //选择新的开始位置
}
```

.setRow 函数的工作是通过传递显示器的地址、行值和二进制 LED 帧来开关显示器。在这个例子里，需要它们全部开，这个二进制数值是 1111111，十进制数值是 255。

项目 22 的目的是演示如果使用预先为这个芯片设计好的代码库控制一个 LED 驱动芯片是多么容易。在项目 21 实践了使用锁存器控制显示器的方式，其代码是非常难编写的。在项目 22 中，这些艰难的工作已在幕后做了，这里有其他点阵显示器控制代码库可用。事实上，Arduino 的 IDE 包含了一个叫 maxtix 的库。我只是因为喜欢用 LedControl.h 库，所以选择使用这个库。实际上可以使用任何符合你的需要的链接库。

在下一章将看到一个不同形式的点阵显示器，叫 LCD。

练 习

使用项目 21 和项目 22 的概念，并把它们联系起来。制作 Pong 游戏，但是代码要显示开始时刻的倒计时（在游戏开始前用毫秒表示），当游戏结束时使用滚动字符的功能显示刚玩过的游戏的得分（玩家存活的毫秒数）和最高分值数字。

小结

第 7 章指导你学习一些有点复杂的主题，包括使用外围芯片。通过这个项目你已经成功一半了，而且你已经知道如何使用移位寄存器或专门的 LED 驱动芯片控制一个点阵显示器。同时，也学到了如何用难的方式编写代码和用简单的方式编写代码，简单的方式编写代码就是使用一个为 LED 驱动芯片预先做好的链接库。

你已经学了一些分时操作的概念——多路复用。这是一个同用在点阵显示器上一样可用于许多其他东西上的技巧。

本章的主题和概念

- 点阵显示器是如何连线的
- 如何连接外部链接库
- 多路复用的概念
- 如何只用 16 个输出引脚独立地开关 64 个 LED
- 定时器的基本概念
- 如何使用 TimerOne 链接库启动中断服务程序
- 如何在代码里使用 `#include` 加载外部链接库
- 如何使用二进制数存储 LED 图像
- 如何用按位非操作转换二进制数
- 如何使用视觉暂留的特性欺骗眼睛
- 如何在多维数组中存储动画
- 如何声明和初始化多维数组
- 从多维数组中获得特定的元素
- 如何做到按位循环
- 如何使用移位寄存器控制点阵显示器

- 如何使用 MAX7219 芯片控制点阵显示器
- 如何使用正确时钟脉冲来向外部芯片装入和输出数据
- 如何在二维数组中存储字符的帧
- 如何读说明文件中的时序图
- 如何使用 MAX7219 的寄存器
- 如何突破 SRAM 的限制把数据存储在程序空间内
- 如何翻转信号
- 如何使字符和其他符号滚动通过点阵显示器
- ASCII 码字母表的概念
- 从 ASCII 代码中选择一个字母
- 如何算出字符串的长度
- 如何从程序空间中写和读
- 如何使用&符号获得变量在存储器中的地址
- 如何使用 LedControl.h 链接库控制独立的 LED 列和 LED 行
- 如何使用逻辑操作符
- 如何使用代码库使工作变得简单和提高代码开发速度



第8章

液晶显示器

让我们研究另外一种流行的显示字符和符号的元件——LCD（液晶显示器）。液晶显示器的典型应用是作为计算器和闹钟的显示器。许多 Arduino 项目中包含了液晶显示器，因此有必要知道如何使用它们。LCD 显示器需要由驱动芯片控制，控制芯片已经集成在显示器中了。最流行的驱动芯片是 Hitachi 公司的 HD44780（或其他相应的芯片）。

做一个基于 LCD 显示器的项目是非常有用并且容易的，因为有一组实用的 LCD 代码库。Arduino IDE 中有一个叫 LiquidCrystal.h 的链接库，里边有大量应用示例，在你的项目中可以使用其中的一个示例。

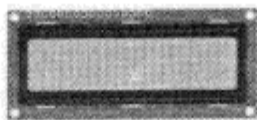
项目 23——基本的 LCD 控制

作为开始，先建立一个演示项目，它将演示如何使用 LiquidCrystal.h 库中的各种功能。项目需要使用一个封装为 16×2 的 LCD 显示器。

需要的元件

需要获得一个使用 HD44780 驱动芯片的 LCD 显示器。这类 LCD 显示器有许多种，有各种各样颜色的。作为一个业余天文爱好者，我特别喜欢黑底上显示红色（红色的字在黑色背景上），因为如果是用在一个天文项目中这会让人想起夜晚的景色。也可以选择其他颜色字符和背景的 LCD，但是显示器必须有背景灯光，并且可以显示 16 列 2 行字符（经常叫做 16×2 LCD 显示器）。

16×2 背光 LCD 显示器



限流电阻（背景灯）



限流电阻（对比度）



把元件连接起来

项目 23 的电路很简单，找到所使用的 LCD 的说明书，用表 8-1 中的 Arduino 引脚，+5V 和地连接到 LCD 相应引脚上。

表 8-1 LCD 用到的引脚

Arduino	其 他	液晶点阵显示器
数字引脚 11		Enable 使能
数字引脚 12		RS（寄存器选择）
数字引脚 15		DB4（数据引脚 4）
数字引脚 14		DB5（数据引脚 5）
数字引脚 13		DB6（数据引脚 6）
数字引脚 12		DB7（数据引脚 7）
地		Vss（电源地）
地		R/W（读/写）
+5V		Vdd（电源正极）
	+5V 通过电阻	Vo（对比度）
	+5V 通过电阻	A/Vee（LED 电源）
	地	LED 地

LCD 显示器的数字引脚 0 到 3 不需要使用，因为本项目用的是 LCD 显示器的 4 比特工作模式。对于一个典型的 LCD 显示器，图 8-1 的电路是正确的。

LCD 上的对比度调节引脚必须连接一个限流电阻以调节对比度到希望的水平，一个大约 10kΩ 的电阻正合适。如果发现找到合适的电阻较难，那么就连接一个变阻器（阻值在 4kΩ 到 10kΩ 之间），将左边的引脚接到 +5V，右边引脚接到地，中间引脚连接到对比度调整引脚（对于我使用的 LCD 对比度调节引脚是引脚 3），现在可以使用旋钮调整对比度直到可以看清楚显示器为止。

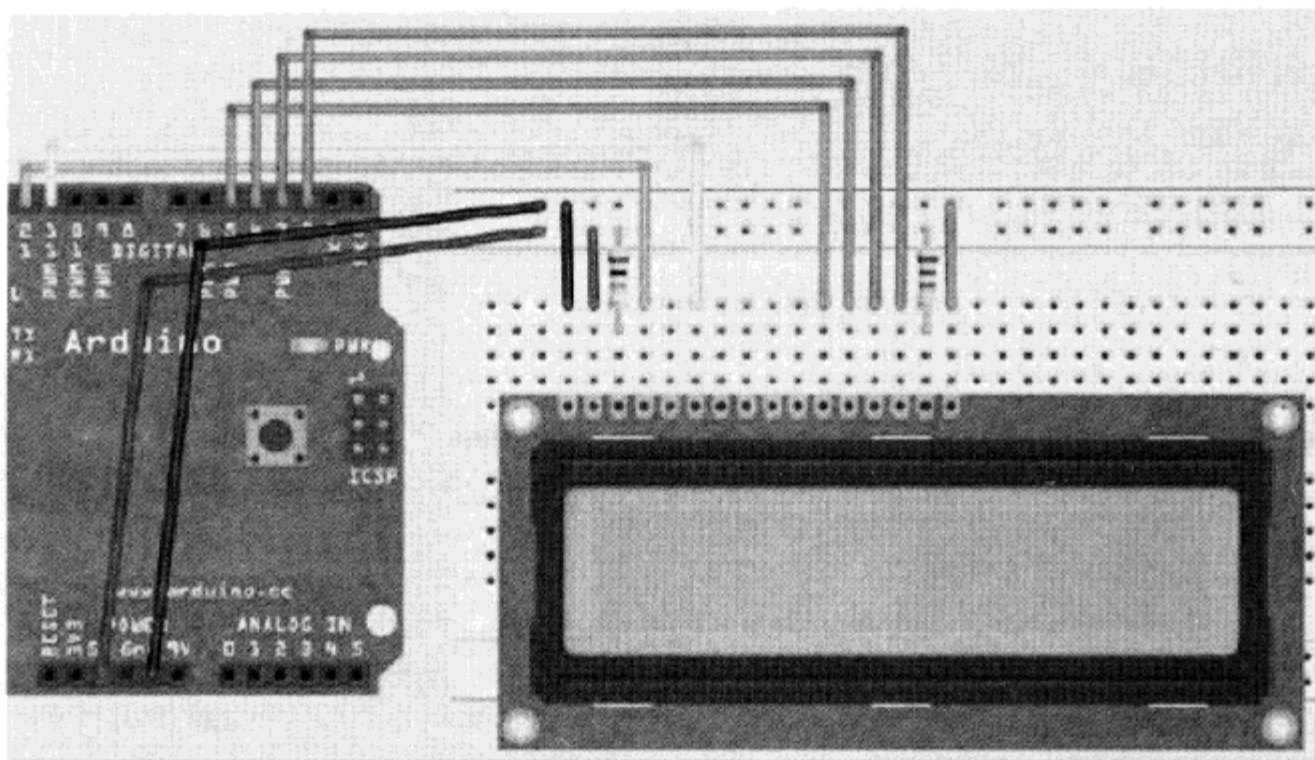


图 8-1 项目 23——基本 LCD 控制电路图（参见彩色版插图）

我试验的 LCD 背景灯光需要 4.2V，因此我加合适的限流电阻在+5V 和 LCD 电源供应引脚之间（在我的 LCD 上是引脚 15），可以连接 LCD 电源引脚到 Arduino 的 PWM 引脚，并且使用 PWM 输出控制背景亮度。但是为了简单起见，在这个项目中就不用这种方法了。已经正确地连接 Arduino 与显示器的引脚、+5V 和地（根据 LCD 的说明书）后，就可以输入代码了。

输入代码

检查你的接线，之后上传清单 8-1 中的代码。

清单 8-1 项目 23 的代码

```
//项目 23
#include <LiquidCrystal.h>
//用接口引脚初始化液晶显示库
LiquidCrystal lcd(12, 11, 5, 4, 3, 2); //用相应的引脚建立一个 lcd 对象
void setup() {
    lcd.begin(16, 2); //设置显示器为 16 列 2 行
}

void loop() {
    //运行 7 个演示程序
```

```
    basicPrintDemo();
    displayOnOffDemo();
    setCursorDemo();
    scrollLeftDemo();
    scrollRightDemo();
    cursorDemo();
    createGlyphDemo();
}

void basicPrintDemo() {
    lcd.clear(); //清空显示器
    lcd.print("Basic Print"); //打印一些字符
    delay(2000);
}

void displayOnOffDemo() {
    lcd.clear(); //清空显示器
    lcd.print("Display On/Off"); //打印一些字符
    for(int x=0; x < 3; x++) { //循环3次
        lcd.noDisplay(); //关闭显示器
        delay(1000);
        lcd.display(); //再开显示器
        delay(1000);
    }
}

void setCursorDemo() {
    lcd.clear(); //清空显示器
    lcd.print("SetCursor Demo"); //打印一些字符
    delay(1000);
    lcd.clear(); //清空显示器
    lcd.setCursor(5,0); //把光标设置在第5列第0行
    lcd.print("5,0");
    delay(2000);
    lcd.setCursor(10,1); //把光标设置在第10列第1行
    lcd.print("10,1");
    delay(2000);
    lcd.setCursor(3,1); //把光标设置在第3列第1行
    lcd.print("3,1");
    delay(2000);
}
```



```

void scrollLeftDemo() {
    lcd.clear(); //清空显示器
    lcd.print("Scroll Left Demo");
    delay(1000);
    lcd.clear(); //清空显示器
    lcd.setCursor(7,0);
    lcd.print("Beginning");
    lcd.setCursor(9,1);
    lcd.print("Arduino");
    delay(1000);
    for(int x=0; x<16; x++) {
        lcd.scrollDisplayLeft(); //向左卷动显示 16 次
        delay(250);
    }
}

void scrollRightDemo() {
    lcd.clear(); //清空显示器
    lcd.print("Scroll Right");
    lcd.setCursor(0,1);
    lcd.print("Demo");
    delay(1000);
    lcd.clear(); //清空显示器
    lcd.print("Beginning");
    lcd.setCursor(0,1);
    lcd.print("Arduino");
    delay(1000);
    for(int x=0; x<16; x++) {
        lcd.scrollDisplayRight(); //向右卷动 16 次
        delay(250);
    }
}

void cursorDemo() {
    lcd.clear(); //清空显示器
    lcd.cursor(); //使光标可见
    lcd.print("Cursor On");
    delay(3000);
    lcd.clear(); //清空显示器
    lcd.noCursor(); //光标不可见
    lcd.print("Cursor Off");
    delay(3000);
}

```

```

    lcd.clear(); //清屏显示器
    lcd.cursor(); //光标可见
    lcd.blink(); //光标闪烁
    lcd.print("Cursor Blink On");
    delay(3000);
    lcd.noCursor(); //光标不可见
    lcd.noBlink(); //关闭闪烁模式
}

void createGlyphDemo() {
    lcd.clear();

    byte happy[8] = { //生成一个笑脸的字节型数组
        B00000,
        B00000,
        B10001,
        B00000,
        B10001,
        B01110,
        B00000,
        B00000};

    byte sad[8] = { //生成一个哭脸字节型数组
        B00000,
        B00000,
        B10001,
        B00000,
        B01110,
        B10001,
        B00000,
        B00000};

    lcd.createChar(0, happy); //生成用户字符 0
    lcd.createChar(1, sad); //生成用户字符 1

    for(int x=0; x<5; x++) { //循环动画 5 次
        lcd.setCursor(8,0);
        lcd.write(0); //写用户字符 0
        delay(1000);
        lcd.setCursor(8,0);
        lcd.write(1); //写用户字符 1
        delay(1000);
    }
}

```

代码回顾

首先载入链接库，要使用这个库控制 LCD。对于不同形式的 LCD 有许多链接库和代码样例可用。在 Arduino 基地网址 www.arduino.cc/playground/code/LCD 上可以找到这些代码。然而，Arduino 的 IDE 自带一个库叫 LiquidCrystal.h，它是非常容易理解和使用的：

```
#include <LiquidCrystal.h>
```

现在你需要创建 LiquidCrystal 对象并设置液晶显示器使用的引脚：

```
LiquidCrystal lcd(12, 11, 5, 4, 3, 2); //用相应的引脚建立一个 lcd 对象
```

这样就创建了一个 LiquidCrystal 对象，叫做 lcd。头两个参数是连接 RS（存储器选择）的引脚和使能引脚。后四个参数是数字引脚 D4 到 D7，因为使用 4-bit 工作模式，所以只需要用 LCD 显示器上 8 个数据引脚中的 4 个。

4-bit 工作模式和 8-bit 工作模式之间的区别是，在 8-bit 工作模式中一次向 LCD 中送一个字节的数，而在 4-bit 工作模式中 8 位数据要分成两个 4 比特数（叫做半字节），这使得代码更长、更复杂。但是在项目中使用了代码库，因此不需要担心使代码变长的问题。如果编写运行时间或存储空间要求苛刻的代码，应该直接使用 LCD 的 8-bit 工作模式。使用 4-bit 模式的优点是只使用了 4 个引脚，节省了引脚数量，这在需要同时连接其他设备时是有用的。

在 setup() 循环中，初始化显示器到需要的大小，它是 16 列 2 行。

```
lcd.begin(16, 2); //设置显示器位 16 列 2 行
```

主循环只是顺序进行 7 个不同的演示例程。在重启之前一个接一个地执行。每一个演示例程显示一个 LiquidCrystal.h 链接库中相关的例程。

```
void loop() {
    //运行 7 个演示程序
    basicPrintDemo();
    displayOnOffDemo();
    setCursorDemo();
    scrollLeftDemo();
    scrollRightDemo();
    cursorDemo();
}
```

```

        createGlyphDemo();
    }

```

第一个函数是 `basicPrintDemo()`，它演示 `print()` 函数的使用。这个例程只是用 `lcd.clear()` 函数清空显示器之后在显示器上用 `lcd.print()` 函数打印，注意如果已经初始化液晶显示器对象并调用它，例如 `LCD1602`，那么这些函数相应地是 `LCD1602.clear()` 和 `LCD1602.print()`。换句话说，函数名紧挨在对象名之后，在它们之间有个点。

`print()` 函数将在光标位置处打印括号内的所有东西。默认的光标位置总是在第 0 行第 0 列。这是顶端左侧那个角，清空显示器之后，光标将设置在这个默认位置或者叫初始位置。

```

void basicPrintDemo() {
    lcd.clear(); //清空显示器
    lcd.print("Basic Print"); //打印一些字符
    delay(2000);
}

```

第二个函数用来演示 `display()` 和 `noDisplay()` 命令。这些函数只是使能或非使能显示器。例程任务是打印 “Display On/Off”，之后运行一个循环 3 次，开关显示器，等待 1 秒，把它打开，再等待 1 秒，重复。不管什么时候关显示器，不管显示器上之前显示了什么，当显示器重启时它都将被保存下来。

```

void displayOnOffDemo() {
    lcd.clear(); //清空显示器
    lcd.print("Display On/Off"); //打印一些字符
    for(int x=0; x < 3; x++) { //循环 3 次
        lcd.noDisplay(); //关闭显示器
        delay(1000);
        lcd.display(); //再开显示器
        delay(1000);
    }
}

```

下一个是 `setCursor()` 函数。它设置光标到括号中的行列位置。这个例程设置光标到 3 个位置，并在显示器的光标位置上打印字符。`setCursor()` 在控制字符的布局时是有用的，它保证字符可输出到显示器的任意位置上。

```

void setCursorDemo() {

```

```

    lcd.clear(); //清空显示器
    lcd.print("SetCursor Demo"); //打印一些字符
    delay(1000);
    lcd.clear(); //清空显示器
    lcd.setCursor(5,0); //把光标设置在第 5 列第 0 行
    lcd.print("5,0");
    delay(2000);
    lcd.setCursor(10,1); //把光标设置在第 10 列第 1 行
    lcd.print("10,1");
    delay(2000);
    lcd.setCursor(3,1); //把光标设置在第 3 列第 1 行
    lcd.print("3,1");
    delay(2000);
}

```

链接库中有两个命令来滚动字符：`scrollDisplayLeft()`和 `scrollDisplayRight()`。两个演示例程显示了这些命令，一个在显示器的右侧并向左滚动 16 个字符的位置打印“Beginning Arduino”，这将使文本滚动出屏幕范围：

```

void scrollLeftDemo() {
    lcd.clear(); //清空显示器
    lcd.print("Scroll Left Demo");
    delay(1000);
    lcd.clear(); //清空显示器
    lcd.setCursor(7,0);
    lcd.print("Beginning");
    lcd.setCursor(9,1);
    lcd.print("Arduino");
    delay(1000);
    for(int x=0; x<16; x++) {
        lcd.scrollDisplayLeft(); //向左卷动显示 16 次
        delay(250);
    }
}

```

下一个函数动作也很简单，文本在左侧向右侧滚动 16 个字符位置，直到文本滚动出屏幕范围：

```

void scrollRightDemo() {

```



```

    lcd.clear(); //清空显示器
    lcd.print("Scroll Right");
    lcd.setCursor(0,1);
    lcd.print("Demo");
    delay(1000);
    lcd.clear(); //清空显示器
    lcd.print("Beginning");
    lcd.setCursor(0,1);
    lcd.print("Arduino");
    delay(1000);
    for(int x=0; x<16; x++) {
        lcd.scrollDisplayRight(); //向右卷动16次
        delay(250);
    }
}

```

光标还不能被看到，它其实一直在那里，只是看不见。不管什么时候用户清空显示，光标都返回顶部左侧角落（列0，行0），打印了一些字符之后，光标在最后打印的字符后。下一个函数清空显示器。之后用 `cursor()` 函数把光标打开，并打印一些字符，光标就可见了，就在这个字符之后，以一个下画线（`_`）符号表示。

```

void cursorDemo() {
    lcd.clear(); //清空显示器
    lcd.cursor(); //使光标可见
    lcd.print("Cursor On");
    delay(3000);
}

```

显示器再次清空，这次光标关闭，这是默认的模式，现在使用 `noCursor()` 使光标不可见。

```

    lcd.clear(); //清空显示器
    lcd.noCursor(); //光标不可见
    lcd.print("Cursor Off");
    delay(3000);
}

```

下一步，光标再次使能，用 `blink()` 函数使能光标的闪烁模式：

```

    lcd.clear(); //清空显示器
    lcd.cursor(); //光标可见
    lcd.blink(); //光标闪烁
}

```

```
    lcd.print("Cursor Blink On");  
    delay(3000);
```

这次光标不仅可见，而且还一开一关地闪烁。这种模式在等待用户输入一些字符时是有用的，闪烁的光标可作为输入字符的提示。

最终，关闭闪烁的光标返回到默认模式：

```
    lcd.noCursor(); //光标不可见  
    lcd.noBlink(); //关闭闪烁模式  
}
```

最后的函数叫做 `createGlyphDemo()`，产生一个用户字符。许多 LCD 允许用户给它们编写自己的字符。标准的 16×2 LCD 可在存储器内存储 8 个用户字符，字符有 5 像素宽、8 像素高（一个像素是一个图像基本单元，例如组成显示图像的一个独立的点）。显示器清空之后用一个笑脸符号和哭脸符号的二进制帧初始化两个字节型的数组。二进制帧有 5 比特宽。

```
void createGlyphDemo() {  
    lcd.clear();  
  
    byte happy[8] = { //生成一个笑脸的字节型数组  
        B00000,  
        B00000,  
        B10001,  
        B00000,  
        B10001,  
        B01110,  
        B00000,  
        B00000};  
  
    byte sad[8] = { //生成一个哭脸字节型数组  
        B00000,  
        B00000,  
        B10001,  
        B00000,  
        B01110,  
        B10001,  
        B00000,  
        B00000,  
    }  
}
```

```
B00000};
```

之后，使用 `createChar()` 生成两个用户定义的字符，这需要两个参数。第一个参数是用户字符号（这个例子中的 LCD 是 0 和 1，LCD 能最多存 8 个用户字符）。第二个参数是在 LCD 存储器中用来生成和存储字符的二进制帧的数组名称：

```
lcd.createChar(0, happy); //生成用户字符 0
lcd.createChar(1, sad); //生成用户字符 1
```

下面一个 `for` 循环循环本身 5 次，在每一次循环的内部把光标设置在第 8 列第 0 行，在这个位置使用 `write()` 命令写第一个用户字符。这个函数把在括号中的用户字符写在光标位置。第一个用户字符——一个笑脸符写在光标位置，之后延时 1 秒，然后第二个用户字符——一个哭脸符写在相同位置处。这个过程重复 5 次生成一个简单的动画。

```
for(int x=0; x<5; x++) { //循环动画 5 次
    lcd.setCursor(8,0);
    lcd.write(0); //写用户字符 0
    delay(1000);
    lcd.setCursor(8,0);
    lcd.write(1); //写用户字符 1
    delay(1000);
}
}
```

项目 23 包含了许多常用的 `LiquidCrystal.h` 链接库中的函数。还有几个其他的函数需要去研究，在 Arduino 参考库 www.arduino.cc/en/Reference/LiquidCrystal 中有关于它们的资料。

硬件回顾

这个项目里的新概念显而易见是 LCD。一个液晶显示器的工作原理是液晶的光膜效应。显示器是由像素组成的。每一个像素由液晶填满。像素以矩阵形式排列在背景光源或反射面上。液晶散布在偏振滤镜之间。两个偏振板互相成 90° ，这使得光通不过。第一个偏振滤镜把光波偏振，因此光波都只向一个方向振动。第二个滤镜与第一个滤镜成 90° ，这将使已经偏振的光通不过。换句话说，图像滤镜由一个方向的非常细的缝组成。振动在与缝相同的方向的光线可以通过缝。但当光线到达第二个滤镜时，这个滤镜的缝是另一个方向，光线不能通过。通过一个电流在各层的行和列液晶处可产生电场，用这个电场可以使液晶

改变方向并排列起来。这引起光线转动 90° ，因此使得光可以通过第二个滤镜，所以一些显示器也叫做“超级扭转器”。

LCD 由灰色的像素组成。排列得非常小的灰点组成字符。一个典型的 16×2 LCD 可在两行中显示 16 个字符，每个字符由 5 个像素宽、8 个像素高组成。如果在显示器上把对比度设得很高，32 组 5×7 像素将变得可见。

这些实际上就是你需要知道的关于 LCD 的工作原理。现在让我们使用 LCD 组成一个温度显示器。

项目 24——LCD 温度显示器

这个项目是一个使用 LCD 给用户提供显示信息的演示，在这个例子里，温度值来自一个模拟温度传感器。这里需要一个按钮用于在显示摄氏度和华氏度之间进行切换。温度最大值和最小值将显示在第二行中。

需要的元件

需要的元件与项目 23 相同，增加一个按钮和一个模拟温度传感器即可。确认温度传感器只输出正值。

16×2 背光 LCD



背光限流电阻



对比度限流电阻



按钮



模拟温度传感器



把元件连接起来

使用在项目 23 中建立的完全相同的电路，增加一个按钮和一个温度传感器，如图 8-2 所示。


```

void loop() {
    lcd.setCursor(0,0); //设置光标到它的初始位置
    int sensor = analogRead(0); //从传感器中读值
    int buttonState = digitalRead(buttonPin); //检查按钮是否被按下
    switch (buttonState) { //如果按钮被按下改变单位状态
        case HIGH:
            scale=-scale; //改变单位
            lcd.clear();
        }

    delay(250);
    switch (scale) { //确定是摄氏度还是华氏度
        case 1:
            celsius(sensor);
            break;
        case -1:
            fahrenheit(sensor);
    }
}

void celsius(int sensor) {
    lcd.setCursor(0,0);
    int temp = sensor * 0.09765625; //转化到摄氏度
    lcd.print(temp);
    lcd.write(B11011111); //温度符号
    lcd.print("C ");
    if (temp>maxC) {maxC=temp;}
    if (temp<minC) {minC=temp;}
    lcd.setCursor(0,1);
    lcd.print("H=");
    lcd.print(maxC);
    lcd.write(B11011111);
    lcd.print("C L=");
    lcd.print(minC);
    lcd.write(B11011111);
    lcd.print("C ");
}

void fahrenheit(int sensor) {
    lcd.setCursor(0,0);
    float temp = ((sensor * 0.09765625) * 1.8)+32; //转化为华氏度
}

```

```

    lcd.print(int(temp));
    lcd.write(B11011111); //打印温度符号
    lcd.print("F ");
    if (temp>maxF) {maxF=temp;}
    if (temp<minF) {minF=temp;}
    lcd.setCursor(0,1);
    lcd.print("H=");
    lcd.print(maxF);
    lcd.write(B11011111);
    lcd.print("F L=");
    lcd.print(minF);
    lcd.write(B11011111);
    lcd.print("F ");
}

```

当运行代码时，当前温度将显示在 LCD 的顶行上。下行显示自从 Arduino 运行或程序重新启动以来的最高、最低温度记录。通过按按钮，可以在摄氏度和华氏度之间改变温度单位。

代码回顾

如前，将 LiquidCrystal 链接库装载到你的项目中：

```
#include <LiquidCrystal.h>
```

初始化一个 LiquidCrystal 对象，设置相应的引脚：

```
LiquidCrystal lcd(12, 11, 5, 4, 3, 2); //建立一个 lcd 对象，设置相应的引脚
```

用摄氏度和华氏度存储最高、最低温度值的一些整数。并且用不可能出现的最大、最小温度值初始化。这些值当程序第一次运行时马上改变。

```
int maxC=0, minC=100, maxF=0, minF=212;
```

定义一个整型变量 `scale`，并用 1 初始化它。`scale` 变量将确定使用摄氏度还是华氏度作为温度单位，默认设置为 1，表示用摄氏度作为温度单位，可以把它改成-1，表示以华氏度作为温度单位。

```
int scale = 1;
```

声明并初始化给按钮用的引脚为一个整型变量:

```
int buttonPin=8;
```

在 `setup()` 函数中, 设置显示器为 16 列 2 行:

```
lcd.begin(16, 2); //设置显示器位 16 列 2 行
```

模拟引脚的参考电压模式设置为 `INTERNAL`:

```
analogReference(INTERNAL);
```

这给 Arduino 的 ADC (模拟量到数字量转化) 一个更精确的参考电压。LM35DT 的输出电压在 100℃ 时是 1V。如果你使用 5V 作为默认参考电压, 那么在 50℃ 时, 也就是在传感器量程一半处的温度时, ADC 上的读数将是 $0.5V = (0.5/5) \times 1023 = 102$, 这仅仅是 ADC 量程的 10%。当使用内部参考电压 1.1V 时, 在 50℃ 时模拟量引脚上现在是 $0.5V = (0.5/1.1) \times 1023 = 465$ 。

如你所见, 这几乎是模拟引脚可读取的值的范围的一半, 因此阅读的分辨率和精度提高了, 所以这种电路更有敏感性。

按钮的引脚现在设置为输入:

```
pinMode(buttonPin, INPUT);
```

清空 LCD 显示器:

```
lcd.clear();
```

在主循环中, 程序开始于设置光标到它初始的位置:

```
lcd.setCursor(0,0); //设置光标到它的初始位置
```

之后, 在模拟引脚 0 上从温度传感器中读温度值:

```
int sensor = analogRead(0); //从传感器中读值
```

之后读按钮的状态, 并把它存储在变量 `buttonState` 中:

```
int buttonState = digitalRead(buttonPin); //检查按钮是否按下
```

现在，需要知道按钮是否已经被按下，如果是，把单位从摄氏度改为华氏度，或者相反，使用 **switch/case** 语句完成以上动作：

```
switch (buttonState) { //如果按钮被按下改变单位状态
    case HIGH:
        scale=-scale; //改变单位
        lcd.clear();
}
```

switch/case 语句是一个新概念：它判断一个条件是否达到以控制程序的流向，确定什么代码应该被执行。**switch/case** 语句用条件语句中的值对比一个变量的值，结果如果是 **true**，运行 **case** 后的代码。

例如，如果有一个变量叫 **var**，当它的值是 1、2 或 3 时希望不同的事情发生，那么你可以这样编程来决定做什么：

```
switch(var) {
    case 1:
        //如果 var 是 1 运行这里的一些代码
        break;
    case 2:
        //如果 var 是 2 运行这里的一些代码
        break;
    case 3:
        //如果 var 是 3 运行这里的一些代码
        break;
    default:
        //如果没有符合的条件运行这里的代码
}
```

switch/case 语句将检查 **var** 的值，如果它是 1，它将运行在 **case 1** 后的代码块，直到 **break** 语句。**break** 语句用来跳出 **switch/case** 语句。没有它，代码将一直执行，直到遇到一个 **break** 语句或到达 **switch/case** 语句的结尾。如果没有检查到一致的值，那么运行在 **default** 中的代码块。注意最好有 **default** 程序块，但是 **default** 程序块不是必需的。

在这个例子里，仅检查一种情况：**buttonState** 是否为 **HIGH**，如果是，值的单位就转换（从摄氏度到华氏度或相反）并且清空显示器。

之后是一个短延时：

```
delay(250);
```

之后是另外一个 `switch/case` 语句，检查如果 `scale` 的值是 1，那么用摄氏度为单位，如果是 -1 用华氏度为单位，并运行相应的函数：

```
switch (scale) { //确定是摄氏度还是华氏度
    case 1:
        celsius(sensor);
        break;
    case -1:
        fahrenheit(sensor);
}
}
```

之后，用两个函数显示温度到 LCD 上。一个以摄氏度为单位，一个以华氏度为单位。这个函数有一个单独参数，传递给它一个整型值，它是从温度传感器中读出的值：

```
void celsius(int sensor) {
```

光标设置在初始位置：

```
lcd.setCursor(0,0);
```

之后，读传感器的值并通过乘以 0.09765625 把它转化成摄氏度：

```
int temp = sensor * 0.09765625; //转化到摄氏度
```

这个常数是用 100（对应传感器的量程）除以 ADC 的转化范围（1024）得到的：

```
100/1024=0.09765625
```

如果传感器量程是 -40 到 150 摄氏度。这个计算应该如下（假定传感器不输出负电压）：

```
190/1024=0.185546875
```

之后打印这种转化后的值到 LCD，接着打印一个字符 `B110111111`，这是一个温度符号，之后打印字符 `C` 表明显示的温度值是摄氏度：


```

lcd.print(temp);
lcd.write(B11011111); //温度符号
lcd.print("C ");

```

之后检查当前的温度值，看它是否大于或小于当前存储在 `max` 和 `min` 中的值。如果是，`max` 或 `min` 中的值变成当前温度值。这将保持自从 Arduino 运行之后读到的最高和最低温度值。

```

if (temp>maxC) {maxC=temp;}
if (temp<minC) {minC=temp;}

```

在 LCD 的第二行，打印 H（表示最高温度）和 `maxC` 的值并跟随温度符号和字母“C”。之后打印 L（表示最低温度）和 `minC` 的值并跟随温度符号和字母“C”。

```

lcd.setCursor(0,1);
lcd.print("H=");
lcd.print(maxC);
lcd.write(B11011111);
lcd.print("C L=");
lcd.print(minC);
lcd.write(B11011111);
lcd.print("C ");

```

打印华氏温度的函数基本差不多（一样的），它需要把温度值乘以 1.8 再加上 32，这样就把摄氏度表示的温度值转化为华氏度表示的温度值：

```

float temp = ((sensor * 0.09765625) * 1.8)+32; //转化为华氏温度

```

现在你应该知道如何使用 LCD 显示有用的信息了，可以做一个自己的项目来显示传感器的数或生成简单的用户界面。

小结

在第 8 章中，你已浏览了在 `LiquidCrystal.h` 库中的最常用的函数。清空屏幕、在显示器上指定的位置打印字符，使光标可见、不可见或闪烁，甚至如何使字符滚动到左边或右边，项目 24 是这些函数在温度传感器上的一个简单应用，是一个如何使用 LCD 在真正的项目上显示数字的样例。

本章的主题和概念

- 如何装载 LiquidCrystal.h 链接库
- 如何把一个 LCD 连接到 Arduino 上
- 如何使用不同的电阻值调整背景灯光亮度和显示对比度
- 如何从 PWM 引脚控制背景灯光亮度
- 如何声明和初始化液晶显示器对象
- 如何在显示器上设置行列值
- 如何用 clear()清空显示器
- 如何用 print()在光标位置打印
- 如何用 display()或 noDisplay()设置显示器开关
- 如何用 setCursor(x, y)设置光标位置
- 如何用 scrollDisplayLeft()向左滚动显示
- 如何用 scrollDisplayRight()向右滚动显示
- 如何用 cursor()和 noCursor()使能或非使能光标
- 如何用 blink()使一个可见光标闪烁
- 如何用 creatChar()生成用户自己的字符
- 如何用 write()写一个字符到光标位置
- LCD 显示器是如何工作的
- 如何从一个模拟量温度传感器中读温度值
- 如何通过使用内部参考电压提高 ADC 的灵敏度
- 使用 switch/case 语句实现选择
- 如何转变 ADC 的值到温度值并用摄氏度或华氏度作为单位
- 如何根据不同量程的温度传感器改变程序代码

舵机

本章要接触到舵机和舵机机构。舵机是一种电机，它使用一个反馈系统来控制电机的位置。尽管你也能买到连续旋转的舵机，甚至也可以把舵机改造成一个标准的连续旋转电机，但是舵机通常情况下只能旋转 180° 。如果你玩过用无线电操纵的飞机，就会接触到舵机。它们用来控制飞机的飞行轨迹。遥控汽车用舵机控制转向机构。模型船用舵机控制船舵。同样地，舵机经常用来控制小机器人手臂的关节转动。也许在本章的结尾，读者会心血来潮地把舵机放到玩具熊或其他玩具内使它们运动起来。图 9-1 和图 9-2 显示了舵机的其他用途。

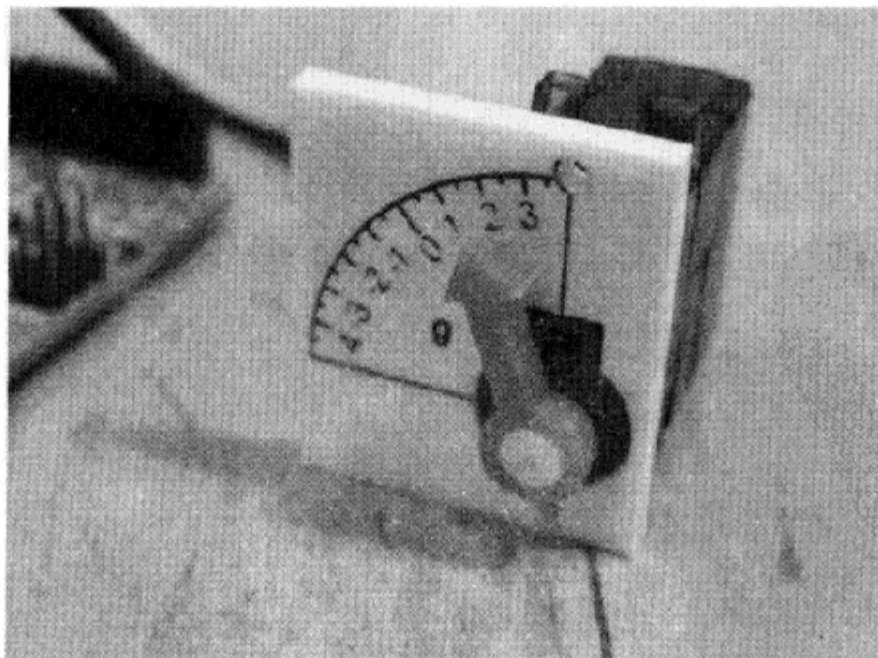


图 9-1 用来控制速度表的舵机（Tod E.Kurt 提供）

Servo.h 库使舵机变得非常容易控制，它集成在 Arduino IDE 中。相对于本书中的其他项目，本章涉及的三个项目非常简单，但是十分有用。让我们从简单入手，先控制一个舵机，

之后进入控制两个舵机的问题。最后用玩具控制手柄控制两个舵机。

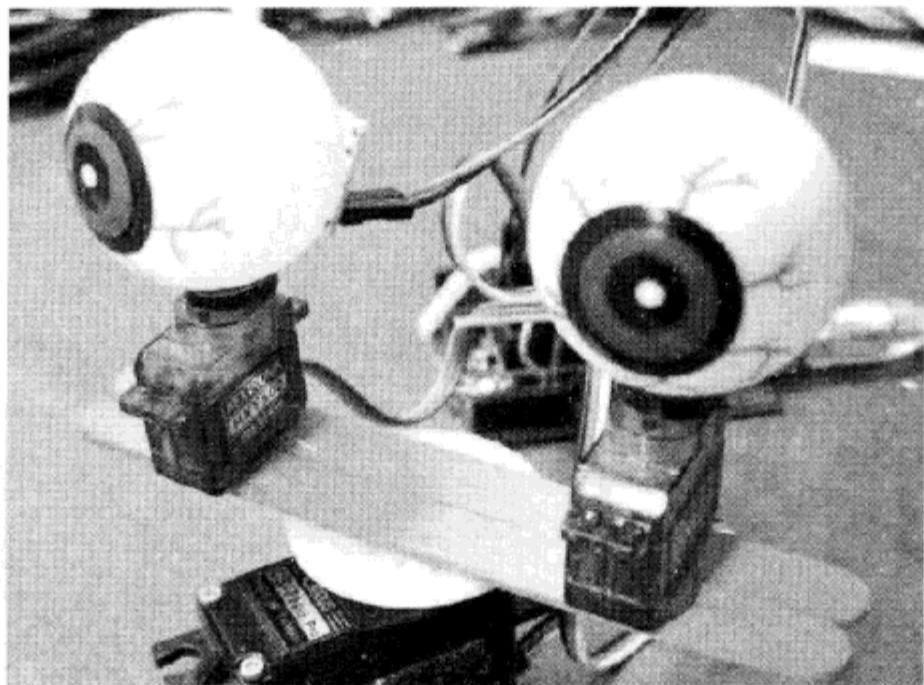


图 9-2 三个舵机控制一个机器人的头部和眼睛 (Tod E.Kurt 供图)

项目 25——舵机控制

在这个简单的项目里读者将用一个变阻器控制一个舵机。

需要的元件

你需要准备一个标准的 RC 舵机，小的或中等尺寸的舵机就行。不推荐使用大的舵机，因为它们需要大电流，所以要单独供电。同时还需要一个变阻器，任何阻值的变阻器都可以。我使用的是一个 $4.7\text{k}\Omega$ 的变阻器。注意，最好把你的 Arduino 连到一个外部 DC 电源上。

标准 RC 舵机



旋转变阻器



把元件连接起来

项目 25 的电路非常简单，按图 9-3 所示的电路把元件连接起来即可。

舵机引出三根线。一根是红色的，要连到+5V 上。一根是黑色或棕色的，要连接到地。第三根可能是白色、黄色或橘色的，要连接到数字量引脚 5 上。

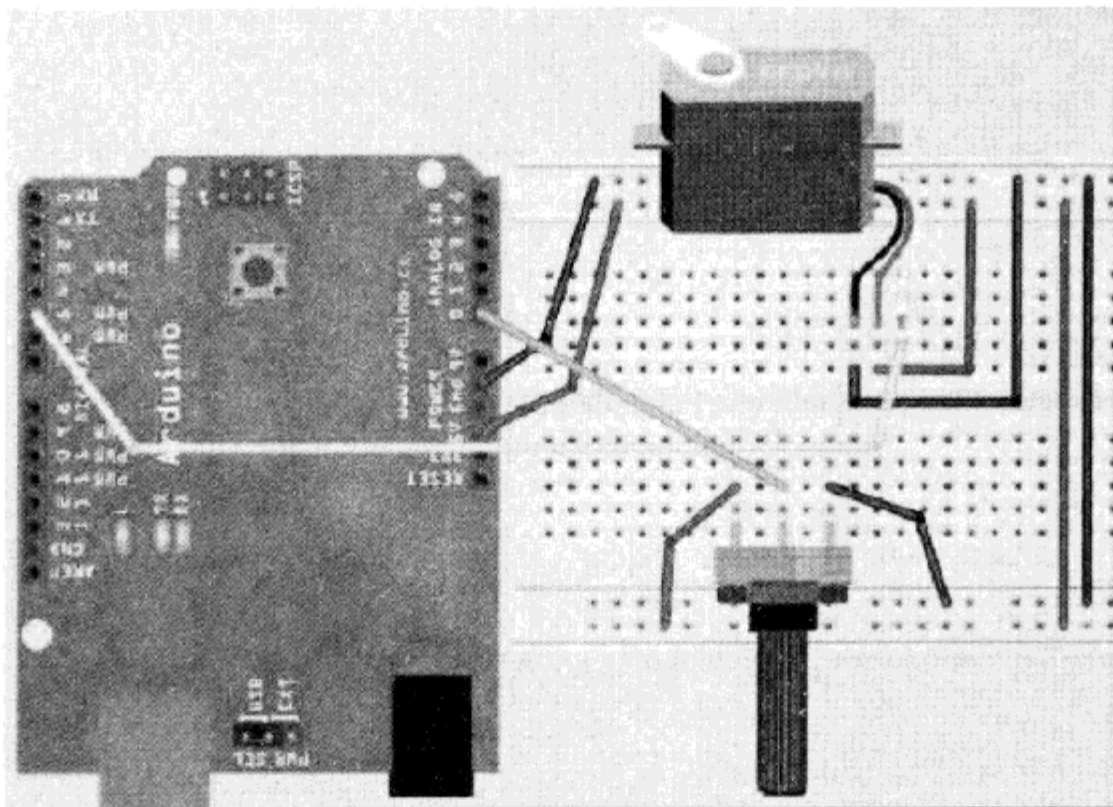


图 9-3 项目 25——舵机控制电路图（参见彩色版插图）

将旋转变阻器两端的引脚连到+5V 和地上，中间引脚连到模拟引脚 0 上。

将每个元件连接好之后，输入以下的代码。

输入代码

这是本书最短的一个程序，看清单 9-1。

清单 9-1 项目 25 的代码

```
//项目 25
#include <Servo.h>

Servo servol; //建立一个舵机对象
void setup()
{
    servol.attach(5); //将引脚 5 上的舵机与舵机对象连接起来
}
```



```
void loop()
{
    int angle = analogRead(0); //读模拟量值
    angle=map(angle, 0, 1023, 0, 180); //映射模拟量值到 0~180°之间
    servol.write(angle); //写角度到舵机
    delay(15); //延时 15 毫秒让舵机转到指定位置
}
```

代码回顾

首先，载入 Servo.h 链接库：

```
#include <Servo.h>
```

之后声明一个叫做 servol 的 Servo 对象：

```
Servo servol; //建立一个舵机对象
```

在 setup 循环中将舵机连接到引脚 5 上：

```
servol.attach(5); //将引脚 5 上的舵机与舵机对象连接起来
```

attach 函数连接一个舵机对象到指定的引脚上。attach 函数可以有 1 或 3 个参数。如果使用 3 个参数，第一个参数表示引脚，第二个参数表示最小角度（0°）的脉冲宽度，单位是毫秒（默认是 544），第三个参数表示最大角度（180°）的脉冲宽度，单位毫秒（默认是 2400）。这将在硬件回顾中解释。在通常情况下，我们只需设置舵机引脚，忽略第二和第三个参数。

Arduino Duemilanove 最多可以连接 12 个舵机，而专门为机器人控制应用设计的 ArduinoMega 可连接多达 48 个舵机。

注意，这个库不使用引脚 9 和引脚 10 的模拟写（PWM）功能。在 Mega 上，使用 12 个以内舵机不需要使用内部 PWM 功能。使用 12 到 23 个舵机将屏蔽引脚 11 和引脚 12 上的 PWM 功能。

在主循环中，从连在引脚 0 上的变阻器中读模拟量值：

```
int angle = analogRead(0); //读模拟量值
```

之后，这个值被映射到 0~180 之间，与舵机臂的转动角度一一对应：

```
angle=map(angle, 0, 1023, 0, 180); //映射模拟量值到 0~180°之间
```

之后，为舵机对象写入正确的角度，用角度做单位（角度必须在 0~180 之间）：

```
servo1.write(angle); //写角度到舵机
```

之后，延时 15 毫秒，让舵机转到指定位置：

```
delay(15); //延时 15 毫秒让舵机转到指定位置
```

你也可以取消引脚和舵机的连接，用它做其他事情。而且，可以用 `read()` 函数读取舵机当前的角度（即最后传给 `write()` 命令的值）。

在 Arduino 网页 <http://arduino.cc/en/Reference/Servo> 上，可以获得更多关于 Servo 链接库的资料。

硬件回顾

舵机是一个小盒子，里面包括一个直流电机，一套装在电机和输出轴之间的齿轮系统，一个位置传感机构和一个控制电路。位置传感机构检测舵机的位置并提供给控制系统，控制系统使用电机控制舵机臂运动到指定位置。

有很多不同尺寸、速度、强度和精度的舵机，它们中有些是非常贵的。功率越大、精度越高的舵机，价格越高。使用无线电控制飞机、汽车或船时，舵机是最常用的一种方式。

通过提供一系列脉冲 PWM 信号实现舵机位置控制。这种方法在前面已经碰到过。脉冲的宽度用毫秒计量。脉冲的发送频率不是特别重要。脉冲的宽度对控制电路非常重要。典型的脉冲频率在 400Hz 到 50Hz 之间。

对典型的舵机来说，到达中间位置所需要的脉冲间隔是 1.5 毫秒，到达 -45° 位置所需要的脉冲间隔是 0.6 毫秒，到达 +45° 位置所需要的脉冲间隔是 2.4 毫秒。根据舵机的数据文件确定对于不同的角度所需要的脉冲宽度。但是，在这个项目中我们使用了 Servo.h 库，所以不用着急，该库提供了舵机需要的 PWM 信号。当需要送出不同角度值给舵机对象时，库中的代码会送出正确的 PWM 信号到舵机中。

一些舵机本身具有连续旋转功能。或者，你可以对一个标准舵机进行改造，使其提供

连续旋转功能。

通过提供一个 $0\sim 150$ 的角度值，可采用同样的方式控制连续旋转的舵机。但是， 0° 值提供单方向的全速旋转， 90° 表示静止， 180° 值提供反方向的全速旋转。中间的值使舵机在正反方向以不同的速度旋转。连续旋转的舵机对制造小机器人非常重要（如图 9-4 所示）。它们能安装到轮子上，为每一个轮子提供相应的速度和方向控制。

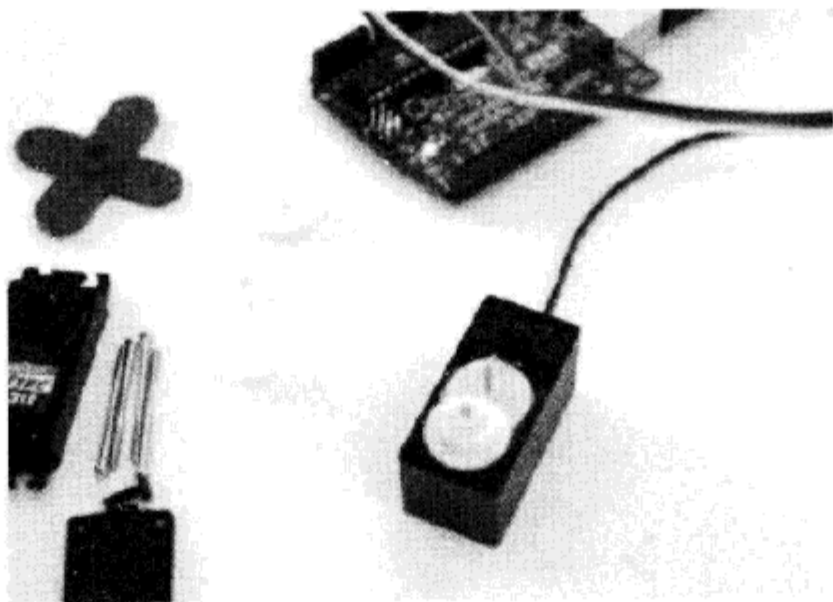


图 9-4 改变舵机使其提供连续旋转功能（Adam Grieg 供图）

还有另外一种舵机，叫做直线电机。它们可驱动滑块运动到指定位置，能够推或拉连接在滑块端部的物体。电视节目“留言终结者”的常驻机器人专家 Grant Imahara 经常使用这种舵机。

项目 26——两个舵机控制系统

现在要建立另外一个简单项目，从串口监视器发送命令来控制两个舵机。在项目 10 中已经学了许多关于串口控制的知识，在那个项目里用串口命令改变 RGB 灯的颜色。这里我们先借用项目 10 中的代码来完成本项目。

需要的元件

本项目需要两个舵机，不需要变阻器。

标准 RC 舵机 2 个



把元件连接起来

项目 26 的电路也非常简单，按照图 9-5 将它们连起来。与上个项目的电路相比，本项目只是拿走了变阻器，将第二个舵机连接到数字引脚 6 上。

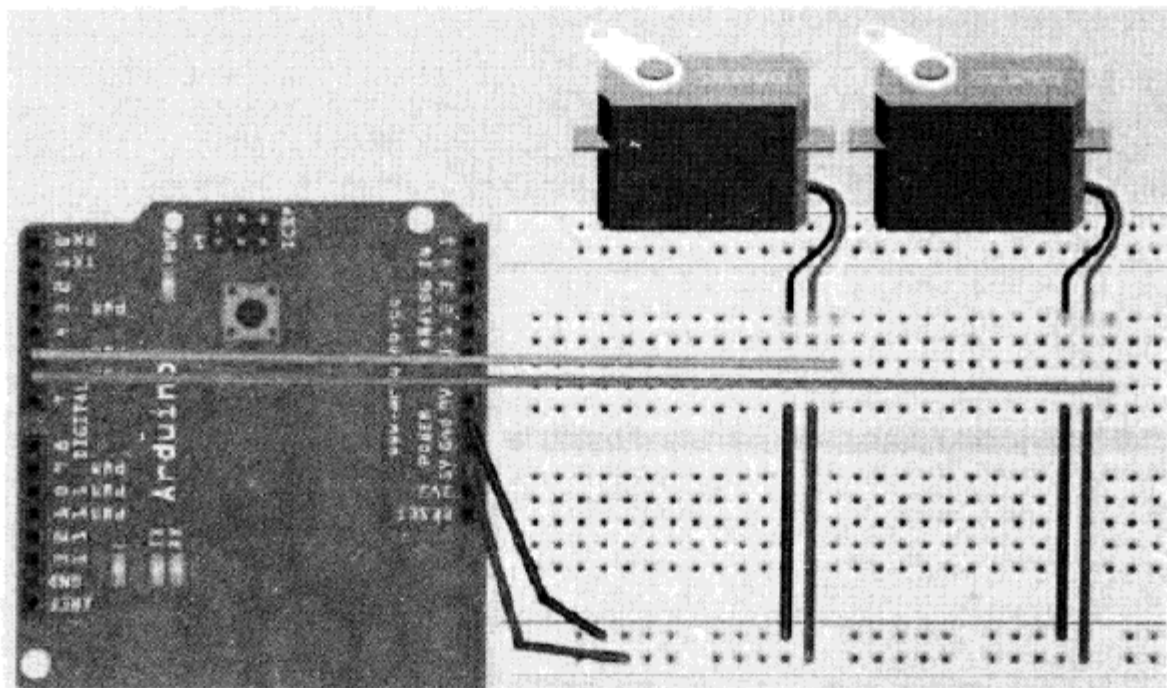


图 9-5 项目 26——两个舵机系统控制电路图（参见彩色版插图）

输入代码

输入清单 9-2 中的代码。

清单 9-2 项目 26 的代码

```
//项目 26
#include <Servo.h>

char buffer[10];
Servo servo1; //声明第一个 Servo 对象
Servo servo2; //声明第二个 Servo 对象

void setup()
{
    servo1.attach(5); //引脚 5 上的舵机连接到 servo1 对象
    servo2.attach(6); //引脚 6 上的舵机连接到 servo2 对象
    Serial.begin(9600);
    Serial.flush();
    servo1.write(90); //设置舵机 1 到初始位置
    servo2.write(90); //设置舵机 2 到初始位置
}
```

```

        Serial.println("STARTING...");
    }

    void loop()
    {
        if (Serial.available() > 0) { //检查是否有数据送到串口
            int index=0;
            delay(100); //延时使缓存填满
            int numChar = Serial.available(); //确定字符串长度
            if (numChar>10) {
                numChar=10;
            }
            while (numChar-->0) {
                //用字符串填满缓存
                buffer[index++] = Serial.read();
            }
            splitString(buffer); //运行 splitString 函数
        }
    }

    void splitString(char* data) {
        Serial.print("Data entered: ");
        Serial.println(data);
        char* parameter;
        parameter = strtok (data, " ,"); //到逗号的字符串
        while (parameter != NULL) { //如果还没有到达字符串结尾
            setServo(parameter); //运行 setServo 函数
            parameter = strtok (NULL, " ,");
        }
        //清除串口缓冲器的文本
        for (int x=0; x<9; x++) {
            buffer[x]='\0';
        }
        Serial.flush();
    }

    void setServo(char* data) {
        if ((data[0] == 'L') || (data[0] == 'l')) {
            int firstVal = strtol(data+1, NULL, 10); //字符串转换成整型
            firstVal = constrain(firstVal,0,180); //数值约束
            servol.write(firstVal);
            Serial.print("Servol is set to: ");
            Serial.println(firstVal);
        }
    }
}

```



```

    }
    if ((data[0] == 'R') || (data[0] == 'r')) {
        int secondVal = strtol(data+1, NULL, 10); //字符串转换成长整型
        secondVal = constrain(secondVal,0,255); //数值约束
        servo2.write(secondVal);
        Serial.print("Servo2 is set to: ");
        Serial.println(secondVal);
    }
}

```

运行代码，打开串口监视器窗口，Arduino 将重启，舵机将转到中间位置。现在可以使用串口监视器发送命令给 Arduino。

我们通过送出一个“L”字符控制左边的舵机，接下来输入 0~180 之间的数字表示转动角度。右侧舵机控制也通过输入 R 和一个数字实现。你可以单独给每一个舵机发送命令，也可同时给两个电机发送命令，中间用空格或逗号隔开。命令如下：

```

L180
L45 R135
L180, R90
R77
R25 L175

```

这个简单的例子告诉你如何通过串口线将命令发送到 Arduino 控制的机器人手臂或电子玩具中。注意串口命令不仅可以从 Arduino 串口监视器输出，还可通过任何使用串口线通信的设备输出。读者也可以用 Python 或 C++ 语言编写程序实现。

代码回顾

这个项目的代码与项目 10 相比，基本上没有太大变化，因此不需要介绍每一个命令的细节。在这里，我只对代码进行简单叙述。仔细研读项目 10，回顾一下字符串处理命令是如何工作的。

首先将 Servo.h 库包含到代码中：

```
#include <Servo.h>
```

之后声明一个 char 类型的数组，用于存储你在串口监视器中输入的命令字符串：

```
char buffer[10];
```

声明两个 Servo 对象:

```
Servo servo1; //声明第一个 Servo 对象
Servo servo2; //声明第二个 Servo 对象
```

在 setup 函数中连接 Servo 对象到引脚 5 和引脚 6:

```
servo1.attach(5); //引脚 5 上的舵机连接到 servo1 对象
servo2.attach(6); //引脚 6 上的舵机连接到 servo2 对象
```

之后, 开始串口通信, 并且执行 serial.flush() 函数。该函数的作用是清除串口缓冲器中的字符, 确保串口缓冲器是空的, 为接收发送给舵机的命令做好准备:

```
Serial.begin(9600);
Serial.flush();
```

给两个舵机都写入值 90, 这是舵机的中间位置, 以使舵机从中间位置开始运动:

```
servo1.write(90); //设置舵机 1 到初始位置
servo2.write(90); //设置舵机 2 到初始位置
```

然后, 字符串 “STARTING...” 显示在串口监视窗口中, 所以用户知道设备已准备好开始接收命令:

```
Serial.println("STARTING...");
```

在主循环中, 检查是否有数据已经送到串口中:

```
if (Serial.available() > 0) { //检查是否有数据送到串口
```

如果数据已送到, 让缓冲器填满并获得字符串长度, 保证字符串长度不超过 10 个字符。一旦缓存满了, 调用 splitString 函数将缓存数组送到函数中:

```
int index=0;
delay(100); //延时使缓存填满
int numChar = Serial.available(); //确定字符串长度
if (numChar>10) {
    numChar=10;
}
while (numChar-->0) {
```

```

//用字符串填满缓存
buffer[index++] = Serial.read();
}
splitString(buffer); //运行 splitString 函数

```

`splitString` 函数接收缓存数组，如果输入多个命令，该函数把它们分成单独的命令。之后调用 `setServo` 函数，用从串口线中接收到的命令字符串作为参数：

```

void splitString(char* data) {
    Serial.print("Data entered: ");
    Serial.println(data);
    char* parameter;
    parameter = strtok (data, " ,");
    while (parameter != NULL) { //如果还没有到达字符串结尾
        setServo(parameter); //运行 setServo 函数
        parameter = strtok (NULL, " ,");
    }
    //清除串口缓冲器中的文本
    for (int x=0; x<9; x++) {
        buffer[x]='\0';
    }
    Serial.flush();
}

```

`setServo` 函数接收从 `splitString` 函数送出小字符串并检查是否有 L 或 R，如果有，根据字符串中的特定字符运行左、右舵机：

```

void setServo(char* data) {
    if ((data[0] == 'L') || (data[0] == 'l')) {
        int firstVal = strtol(data+1, NULL, 10); //字符串转换成长整型
        firstVal = constrain(firstVal,0,180); //数值约束
        servo1.write(firstVal);
        Serial.print("Servo1 is set to: ");
        Serial.println(firstVal);
    }
    if ((data[0] == 'R') || (data[0] == 'r')) {
        int secondVal = strtol(data+1, NULL, 10); //字符串转换成长整型
        secondVal = constrain(secondVal,0,255); //数值约束
        servo2.write(secondVal);
        Serial.print("Servo2 is set to: ");
        Serial.println(secondVal);
    }
}

```

最后两个函数不再赘述，因为它们与项目 10 几乎完全一样，你如果想不起来了，可回头查看项目 10 的解释。

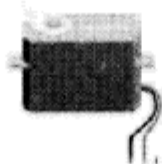
项目 27——操纵杆控制舵机

现在开始另一个简单的项目，用游戏手柄控制两个舵机。你可以控制舵机实现云台摄像功能，就像电视台或机器人所用的摄像头那样。

需要的元件

在项目 26 电路的基础上，增加两个变阻器和 2 轴变阻器游戏手柄。

标准 RC 舵机



2 轴变阻器游戏手柄
(或两个变阻器)



把元件连接起来

项目 27 的电路与项目 26 相同，只是增加了一个游戏手柄，如图 9-6 所示。

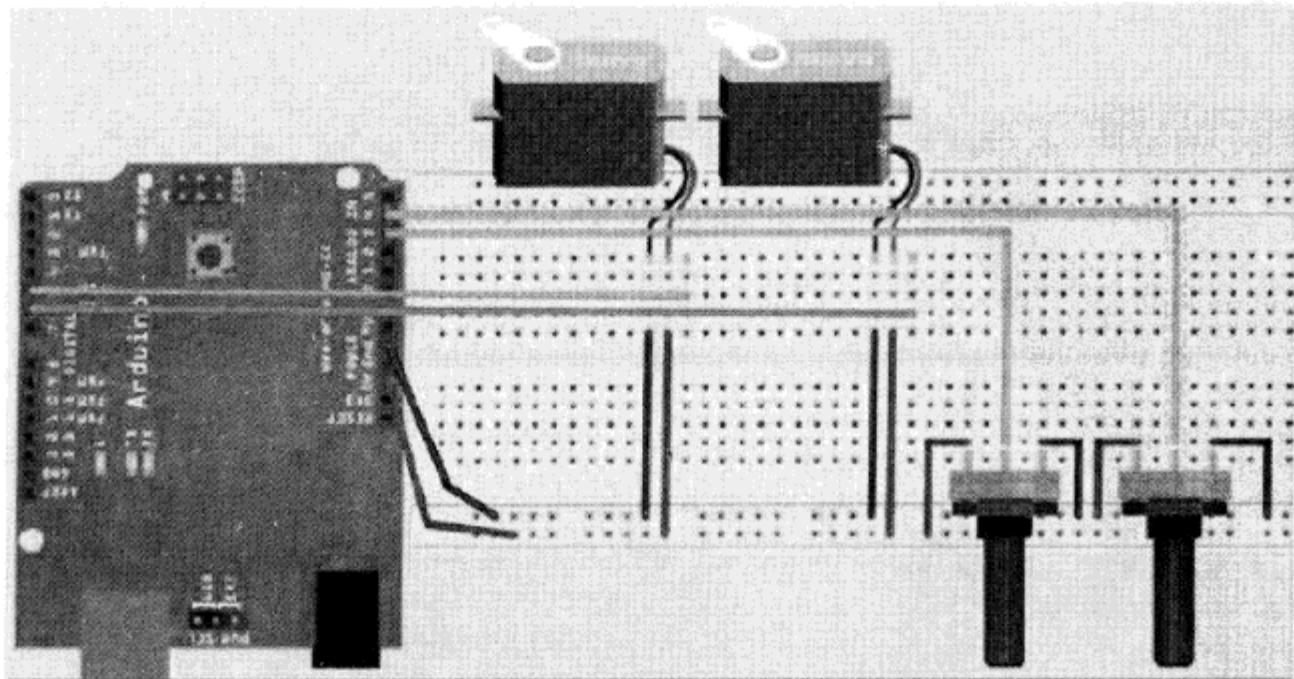


图 9-6 项目 27——操纵杆控制舵机电路图（参见彩色版插图）

变阻器游戏手柄就是由两个互相垂直的变阻器组成的手柄。变阻器的旋钮连接到一个能够来回摇动的摇杆上，并且能够通过一套弹簧系统返回中间位置。

电路连接非常简单。两个变阻器的输出引脚分别连接到+5V 和地，中间引脚连到模拟引脚 3 和模拟引脚 4 上。如果你没有游戏手柄，使用两个互相垂直安装的变阻器也行。

连接两个舵机。一个舵机的轴是垂直的，另一个舵机的轴是水平的，与第一个舵机轴成 90° ，并连接到第一个舵机的电枢侧面。采用图 9-7 所示的方式连接舵机。试验时可用热熔胶，调试好之后使用强力胶做永久连接。

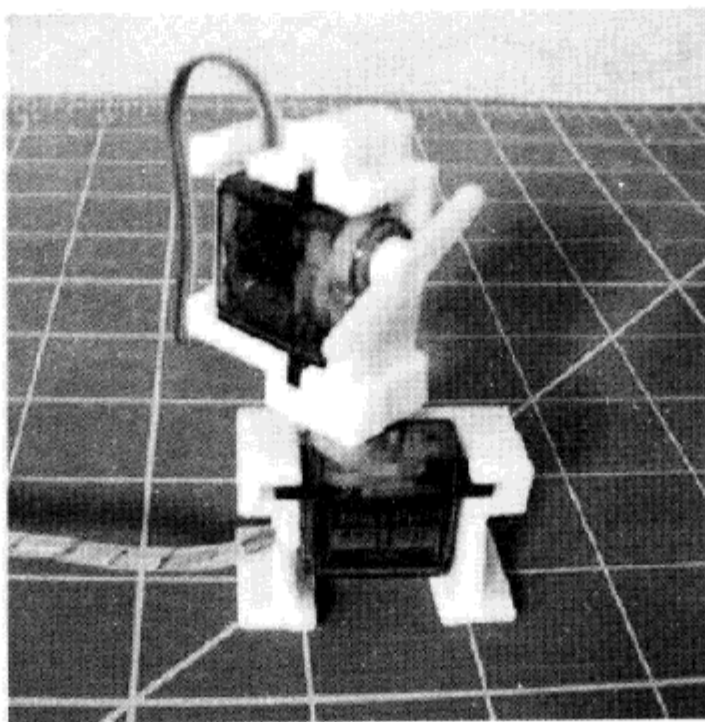


图 9-7 将一个舵机安装在另一个舵机的顶部 (David Stokes 供图)

你也可以用机器人使用的摇摆镜头和倾斜伺服装置代替，这些器件在 eBuy 上能淘到很便宜的。

底部舵机的运动带动顶部舵机旋转，顶部舵机的运动带动它的手臂来回摇动。你可以将网络摄像头或超声传感器安装在手臂上。

游戏手柄可以从 eBuy 上或电子供应商处购买。你也可以使用旧的 C64 或 Atari 的游戏手柄。然而，还有更便宜的替代品，叫做 PS2 控制器。它包含两个两轴变阻器游戏手柄、一套振动电机和其他按钮，如图 9-8 所示。这些器件在 eBuy 上购买非常便宜，而且很容易拆开获得内部的元件。如果你不想把手柄拆成散件，你可以从 PS2 控制的电缆中获得数字代码。事实上，Arduino 提供的链接库可以实现这个功能，这将使你可以控制游戏手柄上所有的元件和按钮。

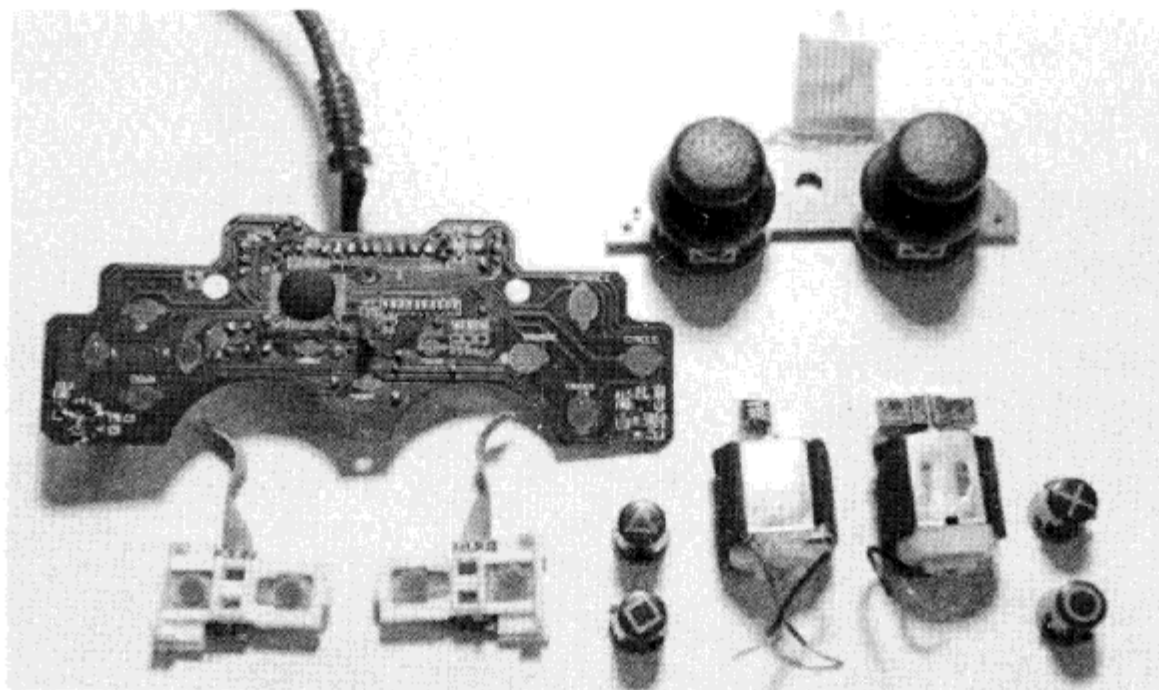


图 9-8 PS2 游戏手柄内部的元件 (Mike Prevette 供图)

输入代码

输入清单 9-3 中的代码。

清单 9-3 项目 27 的代码

```
//项目 27
#include <Servo.h>
Servo servo1; //生成一个 Servo 对象
Servo servo2; //生成另一个 Servo 对象
int pot1, pot2;
void setup()
{
    servo1.attach(5); //引脚 5 上的舵机连接到 servo1 对象
    servo2.attach(6); //引脚 6 上的舵机连接到 servo2 对象
    servo1.write(90); //置舵机 1 到初始位置
    servo2.write(90); //置舵机 2 到初始位置
}

void loop()
{
    pot1 = analogRead(3); //读 X 轴
    pot2 = analogRead(4); //读 Y 轴
    pot1 = map(pot1,0,1023,0,180);
```

```

    pot2=map(pot2,0,1023,0,180);
    servol.write(pot1);
    servo2.write(pot2);
    delay(15);
}

```

当运行这个程序时，你可以使用舵机做一个摇摆摄像头。向前向后摇动游戏手柄，将引起上部舵机电枢前后摆动。左右摇动游戏手柄将控制底部舵机的转动。

如果你发现舵机的运动方向与你期望的相反，说明你把舵机外引脚接反了，只要把接线交换一下就行了。

代码回顾

这是一个简单的项目，但是能够非常好地控制两个舵机的运动。

载入舵机链接库：

```
#include <Servo.h>
```

生成两个 Servo 对象和两个整型数用来存储从游戏手柄变阻器读出的值：

```

Servo servol; //生成一个 Servo 对象
Servo servo2; //生成另一个 Servo 对象
int pot1, pot2;

```

setup 函数连接两个 Servo 对象到引脚 5 和引脚 6，并且转动舵机到中间位置：

```

servol.attach(5); //引脚 5 上的舵机连接到 servol 对象
servo2.attach(6); //引脚 6 上的舵机连接到 servo2 对象
servol.write(90); //置舵机 1 到初始位置
servo2.write(90); //置舵机 2 到初始位置

```

在主循环中，从游戏手柄中的 X 轴和 Y 轴读出模拟值：

```

pot1 = analogRead(3); //读 X 轴
pot2 = analogRead(4); //读 Y 轴

```

将这些值映射到 0 到 180：

```
pot1 = map(pot1,0,1023,0,180);
```

```
pot2 = map(pot2, 0, 1023, 0, 180);
```

之后送到两个舵机中:

```
servo1.write(pot1);  
servo2.write(pot2);
```

这种云台装置的运动非常棒，你可以控制这个装置，使它像人一样运动。这种舵机装置经常用来控制航拍照相机（见图 9-9）。

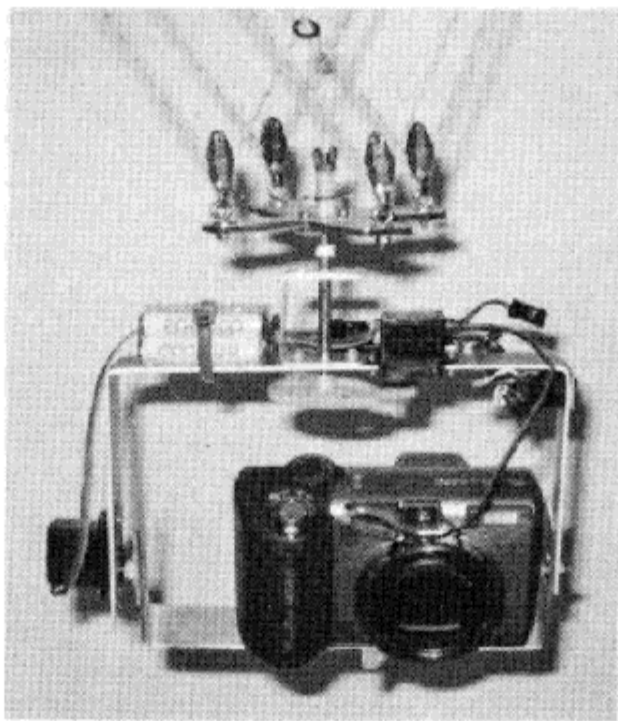


图 9-9 使用两个舵机给相机做一个拍照云台（David Mitchell 供图）

小结

在第 9 章中，通过三个非常简单的项目说明使用 Servo.h 链接库可多么容易地控制舵机。你可以对项目进行改造，增加到 12 个舵机，做一个玩具恐龙，使其像真恐龙一样运动。

舵机对于制作 RC 运输器或机器人非常有用。而且，如同我们在图 9-9 中看到的那样，你可做一个相机云台控制系统，甚至可以使用第三个舵机按下相机上的快门。

本章的主题和概念

- 舵机的潜在用途
- 如何使用 Servo.h 链接库控制 1~12 个舵机

- 舵机是如何工作的
- 如何改造一个舵机使其能够连续旋转
- 如何使用串口命令控制一套舵机
- 如何使用一个模拟游戏手柄控制两轴舵机
- 如何使用两个舵机搭建一个云台装置
- 如何用一个 PS2 控制器控制游戏手柄和按钮



步进电机和机器人

现在要介绍一种新的电机，叫步进电机。项目 28 指导你如何控制步进电机，使步进电机转动一定的圈数，并改变电机的转动速度和方向。步进电机不同于标准电机，因为它们的旋转是以固定的角度一步一步运行的，通过调整电机转动的步数来控制电机速度和转动精度。步进电机有不同的形式和尺寸，引出线有 4、5 或 6 相。

步进电机有许多用途。它们可以用在扫描仪上，控制扫描头的位置。可以用在喷墨打印机上，控制喷头和纸的位置。

本章中另外一个项目使用一个带减速机的直流电机控制一个机器底盘。最后一个项目用来控制一个机器小车沿着画在地上的一条黑线运动。

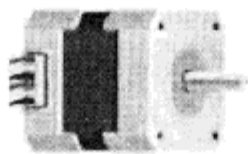
项目 28——基本步进电机控制

这是个非常简单的项目。连接一个步进电机，用 Arduino 控制它以不同的方向和不同的速度转动。

需要的元件

该项目需要一个双极或多级直流步进电机。我在这个项目中用的是 Sanyo 103H546-0440 单极步进电机。

步进电机



L293D 或 SN754410



电机驱动芯片

2 个 $0.01\mu\text{f}$ 陶瓷电容

限流电阻



把元件连接起来

像图 10-1 那样连接每一个元件，注意双极步进电机的接线方式。

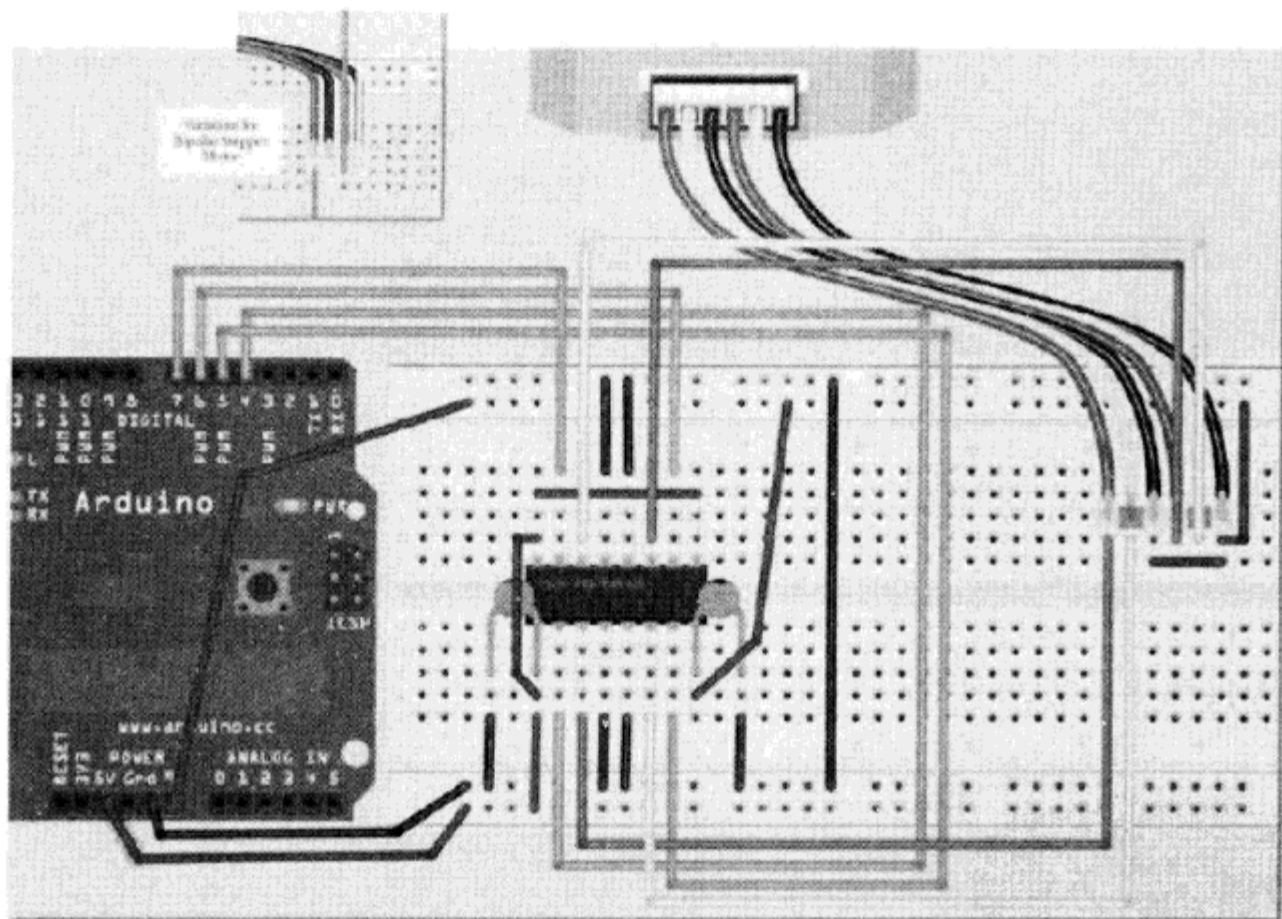


图 10-1 项目 28——基本步进电机控制电路图（参见彩色版插图）

把 Arduino 连接到一个外部直流电源，注意不要使它超载。电容的作用是平滑输出电流，防止对 Arduino 产生干扰。这两个电容分别连在+5V 电源和地之间、电机供电电源和地之间。你也可以使用便宜的铝电解电容器，但是一定要确保连接正确，因为铝电解电容器是有极性的。如果没有这两个电容，这个电路是不能工作的。

电路可能也需要一个限流电阻，将它连在 Arduino 的 Vin 引脚和 SN754410 或 L293D 芯片的电源线之间。Vin 引脚依靠外部电源提供电压，因此需要将电压降低以满足电机的

需要。不这样做，可能损坏电机。

Arduino 的数字引脚 4、5、6 和 7 连接到电机驱动芯片的 1、2、3 和 4 引脚上（见图 10-2）。电机驱动芯片的输出引脚 1 和 2 连接到电机的线圈 1 上，输出引脚 3 和 4 连接到线圈 2 上。需要查看你所用电机的说明书，看哪种颜色的线连接到线圈 1 上，哪种颜色的线连接到线圈 2 上。单级步进电机还可能有第 5 条和第 6 条引出线，这两条线是连到电源地上的。

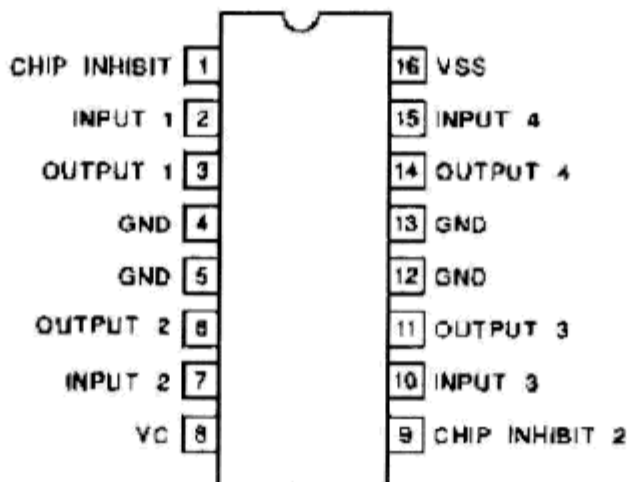


图 10-2 L293D 或 SN754410 电机驱动芯片引脚图

Arduino 上的 5V 引脚连到电机驱动器的引脚 16 上，两个芯片的内部引脚（1 和 9）要连接到 3.3V 线上，使它们为逻辑高。

Arduino 上的 Vin 引脚连到驱动芯片的引脚 8 上，引脚 4、5、12 和 13 连接到地。

如果所有的元件连接正确，那么输入下面的代码。

输入代码

输入清单 10-1 中的代码。

清单 10-1 项目 28 的代码

```
//项目 28
#include <Stepper.h>

//步数等于 360/电机的步距角
#define STEPS 200

//建立一个步进电机对象，叫做 stepper，并指定连接到两个线圈的引脚
Stepper stepper(STEPS, 4, 5, 6, 7);
```

```

void setup()
{
}

void loop()
{
    stepper.setSpeed(60);
    stepper.step(200);
    delay(100);
    stepper.setSpeed(20);
    stepper.step(-50);
    delay(100);
}

```

在运行代码前，确认 Arduino 由一个外部直流电源供电。当项目运行时，你会看到步进电机转动一整圈，停一小会儿，之后向相反方向转回 1/4 圈，停一小会儿，之后重复以上动作。在电机伸出轴上放一个小纸条有助于更清楚地看到电机的转动。

代码回顾

这个项目的代码是非常简单的，因为 Stepper.h 库为你做了所有困难的工作。首先在程序中调用这个库：

```
#include <Stepper.h>
```

之后，定义电机转一整圈需要多少步。典型的步进电机每步转过的角度要么是 7.5° ，要么是 1.8° 。但是你所使用的步进电机也可能有不同的步距角。步进电机的步数等于 360° 除以步距角。在这个项目中我用的步进电机步距角是 1.8° ，意味着转动 360° 需要 200 步：

```
#define STEPS 200
```

之后，建立一个步进电机对象，叫做 stepper，并指定连接到两个线圈的引脚。

```
Stepper stepper(STEPS, 4, 5, 6, 7);
```

setup 函数什么也不做，但是必须包含到代码中：

```
void setup()
```

```
{  
}
```

在主循环中，首先用 `rpm`（每分钟的转动圈数）设置电机转速，使用 `.setSpeed` 函数把以 `rpm` 为单位的速度值放在括号内作为参数：

```
stepper.setSpeed(60);
```

之后，告诉步进电机它要执行多少步。这里设置为以 `60rpm` 的转速转动 `200` 步，意味着电机用一秒钟转一整圈：

```
stepper.step(200);
```

当转动结束时，延时 `1/10` 秒：

```
delay(100);
```

将速度降低到 `20` 转/分钟：

```
stepper.setSpeed(20);
```

之后使电机向相反方向运转（因此速度是负值）`50` 步，对于 `200` 步一圈的步进电机，这是 `1/4` 圈：

```
stepper.step(-50);
```

最后执行另一个 `100` 毫秒延时，然后程序重复循环。

就像你所看到的，控制一个步进电机是非常容易的。你已经控制了它的转动速度、转动方向和转动角度，因此可以精确地控制安装在电机轴上的物体到达指定位置。使用步进电机可以精确地控制自动车轮的运动。使用步进电机可以转动机器人的头部或相机，让它们到达你所期望的位置。

硬件回顾

在这个项目中使用的新硬件是步进电机。一个步进电机是一个直流电机，它的转动可分成一定数量的小步。能够实现这一点是因为电机轴上有齿轮状的转子。在电机外壳内有带齿的电磁铁。当电磁铁的第一个线圈通电时，转子上的齿与电磁铁上的齿对齐，第二个

电磁铁上的齿位于偏离第一个电磁铁的方向转动一个齿的角度。当第一个电磁铁的线圈断电、第二个电磁铁的线圈通电时，转子上的齿与这个电磁铁上的齿对齐，使转子向相同方向转动一个齿的角度。这个过程重复进行，通过给第一个电磁铁通电，转子上的下一个齿再次与第一个电磁铁上的齿对齐，电机持续向一个方向转动。

每一次电磁铁通电，电机转子向一个方向转动一步。通过改变电磁铁的通电顺序可使步进电机朝相反的方向转动。

Arduino 的任务是以正确的顺序和速率给线圈输入合适的高电平和低电平驱动电机轴转动。这些工作由 Stepper.h 库完成。

一个单极电机有 5 个或 6 个引线，连到电机内部的 4 个线圈上。线圈上的中间引脚连在一起，接到电源上。双极电机通常有 4 个线头，它们没有中间连接点。图 10-3 给出了不同形式和尺寸的步进电机。

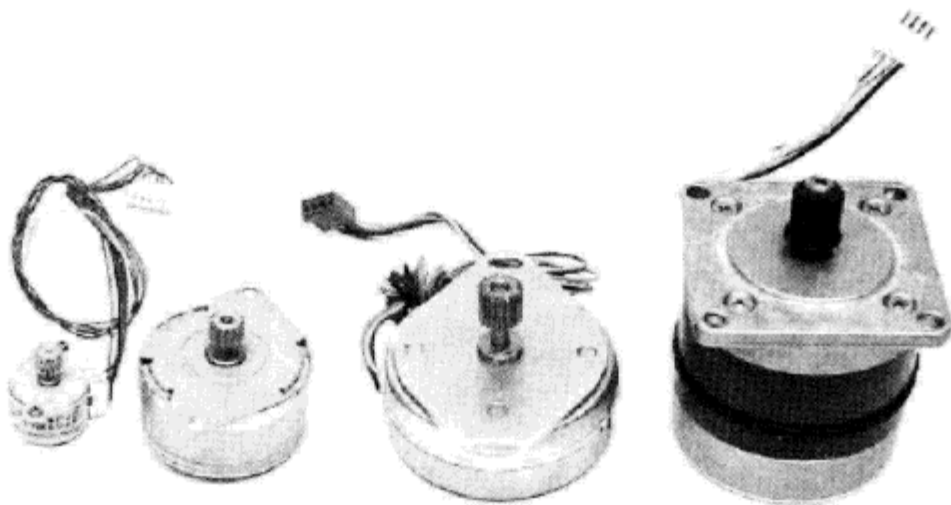


图 10-3 不同类型的步进电机（Aki Korhonen 供图）

步进电机的控制顺序可见表 10-1。

表 10-1 步进电机的控制顺序

步	线头 1	线头 2	线头 4	线头 5
1	HIGH	LOW	HIGH	LOW
2	LOW	HIGH	HIGH	LOW
3	LOW	HIGH	LOW	HIGH
4	HIGH	LOW	LOW	HIGH

图 10-4 显示了一个单极步进电机简图。可以看到，这里有 4 个线圈（实际上有两个，但是中间连线将它们又分成两个小线圈）。中间线接到电源，其他两条线接到外部 H 桥驱

动 IC 上，第二个线圈的另外两条线接到外部 H 桥上。

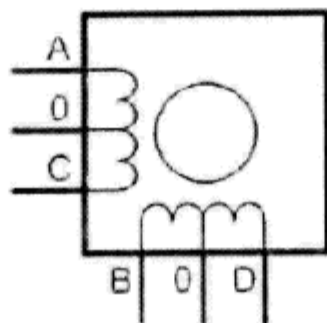


图 10-4 单极步进电机电路图

图 10-5 显示的是双极步进电机的原理图。这种电机只有两个线圈没有中间引脚。双极步进电机的步序与单极步进电机相同。然而，这次将通过线圈的电流反向，例如，HIGH 电平相当于正电压，LOW 电平相当于接地（或将电压拉低连接）。

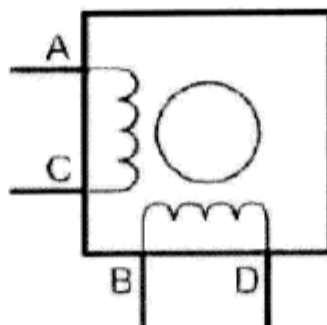


图 10-5 双极步进电机电路图

使用特殊半步细分技术有助于提高步进电机的精度。这项技术虽然可以提高电机精度，但是降低了电机的位置稳定性。利用齿轮系统是提高电机精度的最好方法。

你已经学习了所有控制步进电机的知识，接下来利用这些知识对电机进行控制。这次要使用一个电机模板控制两个安装在机器人底盘轮子上的电机。

项目 29——使用电机模板

现在使用一个电机模板分别控制两个电机。你将要学到如何控制每个电机的速度和方向，然后应用这些知识控制机器人的两个车轮。项目 30 要使用这些技能做一个巡线机器人。

需要的元件

光使用两个直流电机也能完成本项目。但是，这没什么新鲜的，之前的项目已经控制过直流电机。因此，在本项目中要么使用一个两轮机器底盘，要么做出一个带有轮子的齿

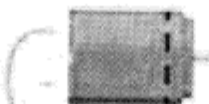
轮减速电机。电机以半速运行时，每秒钟可以前进大约 20cm 或 6 英寸。

本项目需要一个电池盒，给 Arduino 板和电机提供足够的电压。我使用 6XAA 电池组成的一个电池盒，盒内有一个开关。电池盒连接到步进电机引线上。

电机控制板



2 个直流电机或……



1 个两轮机器人底盘



电池盒



把元件连接起来

因为这个项目只需要把一个电机模板插入 Arduino 中，把两个步进电机连接到电机模板上的 4 个输出上，所以不需要电路图。图 10-6 显示的是电机模板插入 Arduino 上的图片。

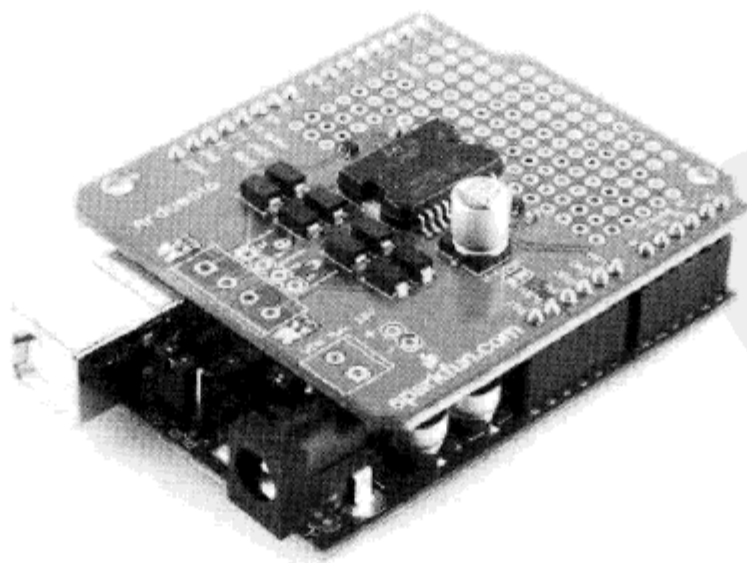


图 10-6 电机模板插入 Arduino 上（感谢 Sparkfun 供图）

为了测试这个项目，我使用 Sparkfun 的 Ardumoto 电机模板，这个模板是为控制两台电机设计的。Adafruit 也提供一种电机模板，可以同时控制 4 台直流电机、两个步进电机或

两个舵机。Adafruit 模板和一个外部 AFMotor 链接库配合使用可以非常容易地控制电机、步进电机或舵机。你也可以根据自己的情况选择使用哪个模板和控制哪种电机。然而，这个项目的代码是针对 Ardumoto 模板的，因此如果你使用不同的模板，需要相应地修改代码。

每台电机需要通过 A 和 B 口连到模板上。

我使用来自 DFRobot 的两轮机器人底盘（见图 10-7）。这是一个不错的机器人底盘，价格也便宜。底盘上有配件和安装孔可以固定传感器。它是巡线机器人项目最理想的选择。然而，任何机器人底盘都可用于这个项目。你甚至可以使用 4 轮机器人底盘，只要你记得去控制每一侧的两台电机，而不仅仅控制一台电机。当然，如果你不想用机器人底盘，也可以只使用两个直流电机来做试验。

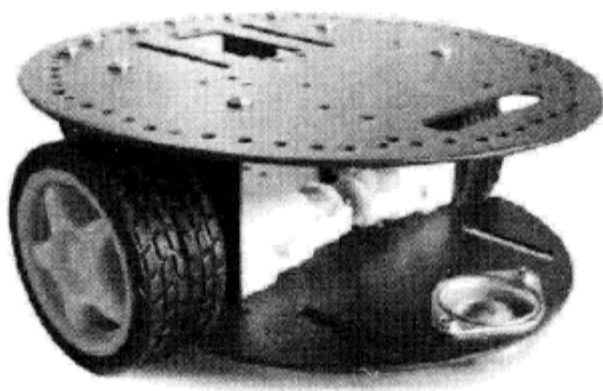


图 10-7 两轮机器人底盘（感谢 DFRobot 供图）

输入代码

从清单 10-2 中输入代码。

清单 10-2 使用电机模板

//项目 29——使用电机模板

//设置每台电机的速度和方向引脚

```
int speed1 = 3;
int speed2 = 11;
int direction1 = 12;
int direction2 = 13;
```

```
void stopMotor() {
```

```
//关闭电机
analogWrite(speed1, 0);
analogWrite(speed2, 0);
}

void setup()
{
    //设置所有引脚为输出模式
    pinMode(speed1, OUTPUT);
    pinMode(speed2, OUTPUT);
    pinMode(direction1, OUTPUT);
    pinMode(direction2, OUTPUT);
}

void loop()
{
    //所有电机以 50%的速度向前运动 2 秒
    digitalWrite(direction1, HIGH);
    digitalWrite(direction2, HIGH);
    analogWrite(speed1, 128);
    analogWrite(speed2, 128);
    delay(2000);

    stopMotor(); delay(1000); //停止

    //向左转向 1 秒
    digitalWrite(direction1, LOW);
    digitalWrite(direction2, HIGH);
    analogWrite(speed1, 128);
    analogWrite(speed2, 128);
    delay(1000);

    stopMotor(); delay(1000); //停止

    //两台电机以 50%速度运行 2 秒
    digitalWrite(direction1, HIGH);
    digitalWrite(direction2, HIGH);
    analogWrite(speed1, 128);
    analogWrite(speed2, 128);
    delay(2000);
```

```
stopMotor(); delay(1000); //停止

//以 25%速度向右转
digitalWrite(direction1, HIGH);
digitalWrite(direction2, LOW);
analogWrite(speed1, 64);
analogWrite(speed2, 64);
delay(2000);

stopMotor(); delay(1000); //停止

}
```

代码回顾

本项目所要做的第一件事情是分配控制电机的速度和方向的引脚。在 Arduino 模板上，引脚 3、11、12 和 13 比较合适：

```
int speed1 = 3;
int speed2 = 11;
int direction1 = 12;
int direction2 = 13;
```

之后，生成一个函数，用于关闭电机。在代码中，电机要关闭 4 次，因此有必要生成一个函数来做这些事情：

```
void stopMotor() {
    //关闭电机
    analogWrite(speed1, 0);
    analogWrite(speed2, 0);
}
```

为了关闭电机，只需要设置电机的速度为 0 即可。因此，写一个 0 值到 speed1（左侧电机）引脚和 speed2（右侧电机）引脚中。

在 setup 函数中，这 4 个引脚设置为输出模式：

```
pinMode(speed1, OUTPUT);
pinMode(speed2, OUTPUT);
```



```
pinMode(direction1, OUTPUT);
pinMode(direction2, OUTPUT);
```

在主循环中，执行 4 个单独的运动任务，第一步，使电机向前以 50% 的速度运行 2 秒：

```
digitalWrite(direction1, HIGH);
digitalWrite(direction2, HIGH);
analogWrite(speed1, 128);
analogWrite(speed2, 128);
delay(2000);
```

这里首先要设置方向引脚为 HIGH，这等于使底盘向前运动（假定电机连线正确）。每个电机的速度引脚设置为 128。PWM 值的范围是 0~255，因此 128 为中间值，表示最高速度的一半，即每个电机以半速运行。不管你所设定的方向和速度如何，电机将连续运行直到你改变它们的运行状态。因此，延时 2 秒将保证电机向前以半速运行 2 秒，大约运行 1 米或 3 英尺的距离。

下一步，停止电机运动，等待 1 秒：

```
stopMotor(); delay(1000); //停止
```

接下来将左侧电机的运动方向改为向后，而右侧的电机向前。为了使电机向前转，设置方向引脚为 HIGH，为了使电机向后转，设置方向引脚为 LOW。速度保持在最高速度的 50%，因此将值 128 写到两个电机的速度引脚上（右侧轮向前运动状态和速度是不必要的，但是我还是留着它们，以便于对这些参数做适当修改来获得所期望的运动状态）。

右侧的轮子向前，左侧的轮子向后转动使机器人逆时针转弯（向左），持续运动 1 秒，使底盘向左转 90° 左右。执行完这段代码之后，电机再次停止运行，保持 1 秒：

```
digitalWrite(direction1, LOW);
digitalWrite(direction2, HIGH);
analogWrite(speed1, 128);
analogWrite(speed2, 128);
delay(1000);
```

下一个运动使机器人再次向前以半速运行 2 秒。然后，电机停止运行，保持 1 秒：

```
digitalWrite(direction1, HIGH);
```

```
digitalWrite(direction2, HIGH);  
analogWrite(speed1, 128);  
analogWrite(speed2, 128);  
delay(2000);
```

最后所做的运动是改变车轮转动方向，使左侧轮向前、右侧轮向后运动。这将使机器人顺时针转动（向右）。这次设置速度为 64，这是最高速度的 25%，使车轮转动得比以前慢。这个动作持续 2 秒，足以使机器人转动 180°：

```
digitalWrite(direction1, HIGH);  
digitalWrite(direction2, LOW);  
analogWrite(speed1, 64);  
analogWrite(speed2, 64);
```

这 4 个部分联合在一起将使机器人向前运动 2 秒，停止，向左转 90°，停止，再向前运动 2 秒，之后以较慢的速度转 180°。之后重复以上动作。

要根据你所用的机器人调整时间和速度，因为你可能使用不同的电压等级的电机、不同减速比的齿轮。采用不同的运动组合，使机器人以不同的速度和不同的转向角度运动。

项目 29 教你使用不同的命令以不同的方向和不同的速度控制两个电机。如果你只有电机而没有机器人底盘，在电机轴上粘上一些纸带，可以看到电机的速度和转动的方向。

硬件回顾

这个项目中使用的新元件是电机模板。模板是一个简单的电路板，能够方便地安装在 Arduino 上。模板上的引脚用于连接到 Arduino 的引脚上。Arduino 顶部有引脚插孔，模板可以连到这些引脚上。当然，不同类型的模板使用不同的 Arduino 引脚，因此在代码中也要根据不同的模板修改相应的设置。例如 Ardumoto 模板使用了 Arduino 的数字引脚 10、11、12 和 13。

通过使用简单的插入模板和装载必要的代码可以给 Arduino 增加功能。可以采用很多种模板延伸 Arduino 的功能，包括实现以太网连接、GPS、继电器控制、SD 和 MicroSD 卡使用、LCD 显示器、电视连接、无线通信、触摸屏、点阵显示器和 DMX 灯控制。图 10-8 给出了不同类型的模板。

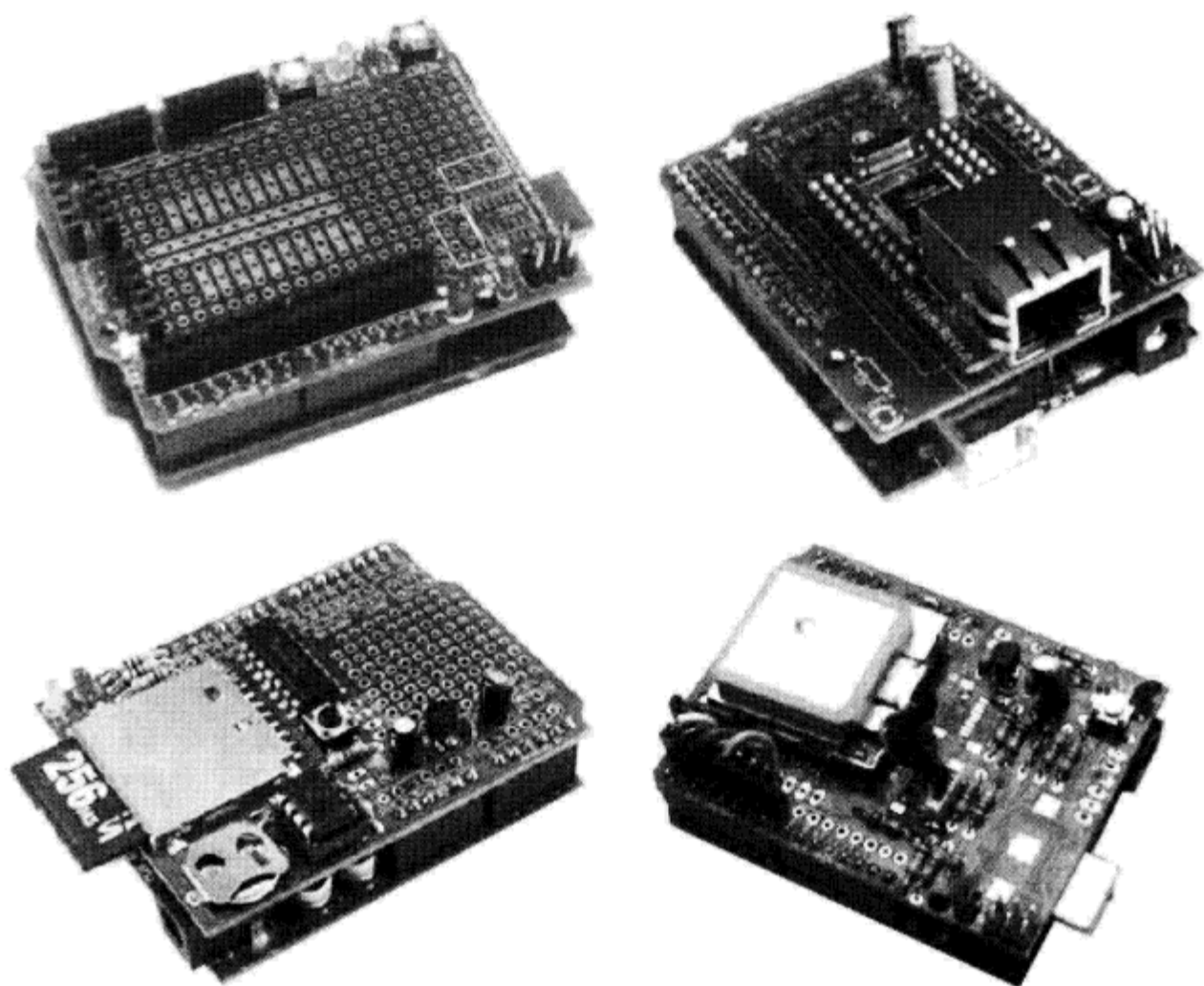


图 10-8 样机模板、以太网模板、时间日期模板、GPS 模板(感谢 Adafruit 公司, 网址为 www.adafruit.com)

项目 30——巡线机器人

现在你知道了如何使用一个电机模板控制两个电机或机器人底盘了。你也可以用这些技能做更多的事情。在这个项目中, 制作一个机器人, 这个机器人可以检测并追随一条画在地上的黑色线。如今, 在工厂中也使用这类机器人, 它们能够自动地在工厂地面上运动, 被称为自导引运输机 (AGVs)。

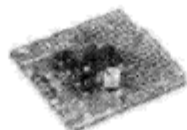
需要的元件

这个项目需要一个小的轻型机器人底盘。你可以购买现成的或用组件拼装, 甚至按图纸制作一个 (这更有意思)。轮子要选用经齿轮减速的, 保证它转动得比电机慢, 因为机器人运动得必须足够慢, 才能沿着导航线运动。

本项目中, 机器人自主运行, 不需外部供电, 所以它需要有自己的电池组。我发现 6-AA 电池组盒能够给 Arduino、传感器、LED、底盘和电机提供足够电力。更大的电池组可以使

机器人运行更长的时间，但是要更贵、更重，所以采用哪种电池组依赖于机器人的负载能力。

电机模板



4 个限流电阻



3 个 1kΩ电阻



4 个白色 LED



3 个光电阻



2 个直流电机



……2 轮机器人底座



电池组



把元件连接起来

像图 10-9 那样连接电路。电机模板没有显示出来，但是它有 4 个输出直接连到电机 A 和电机 B 上。

4 个 LED 可以是任何颜色的。只要它们有足够的亮度照亮地面，使得在地面和黑线之间的对比度能够被传感器检测到即可。白色 LED 是最好的，它们能够以最低的电流输出最大的亮度。这 4 个 LED 连成一条直线，由引脚 9 驱动。要使用正确的限流电阻给 LED 限流。

最后，连接 3 个光电阻到模拟引脚 0、1 和 2。光电阻的一个引脚连接到地，另一个引脚通过 1kΩ电阻连接到+5V 上。模拟引脚线连到光电阻引脚和电阻之间（像项目 14 那样使用 LDR）。

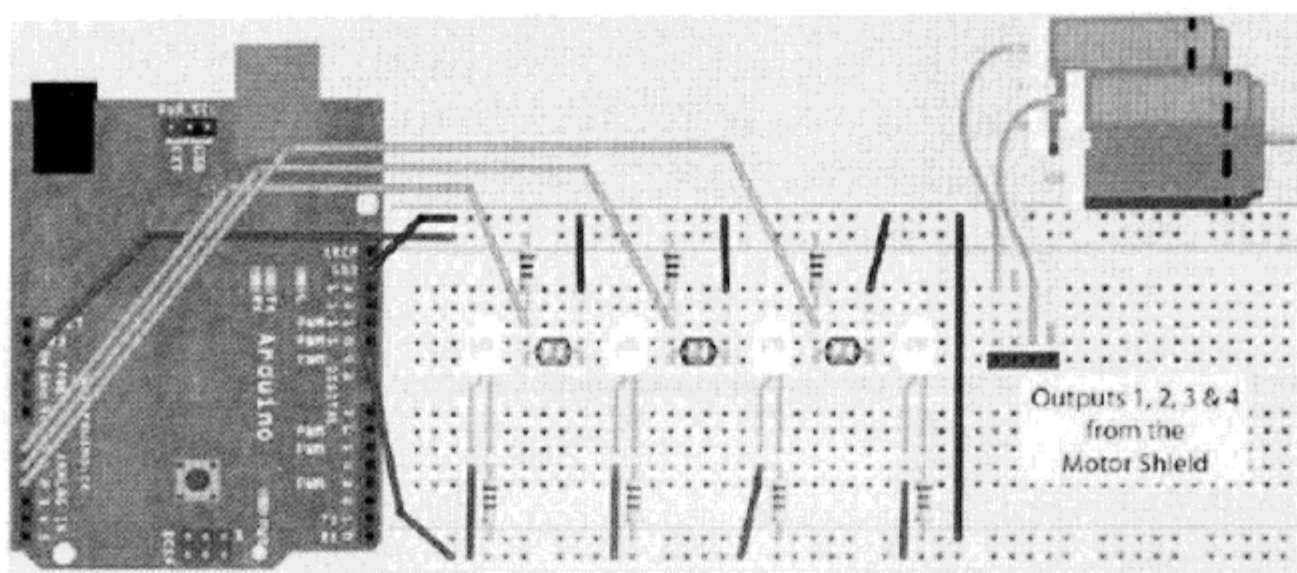


图 10-9 项目 30——巡线机器人电路图（参见彩色版插图）

如果只为达到试验目的，可以使用上面的面包板接线方式。但是为了制作机器人，需要制作一个传感器电路板，它包括 4 个 LED 和 3 个 LDR。这些元件必须集中放置，因此将它们靠近并排成直线放置。我将 LED 和传感器焊接在一块小电路板上。电路板上放了 3 个 LDR 和 4 个 LED，中间的 LDR 比 LED 高一点，因此从 LED 出来的光不会直接照到传感器上，图 10-10 显示了我所做的东西。

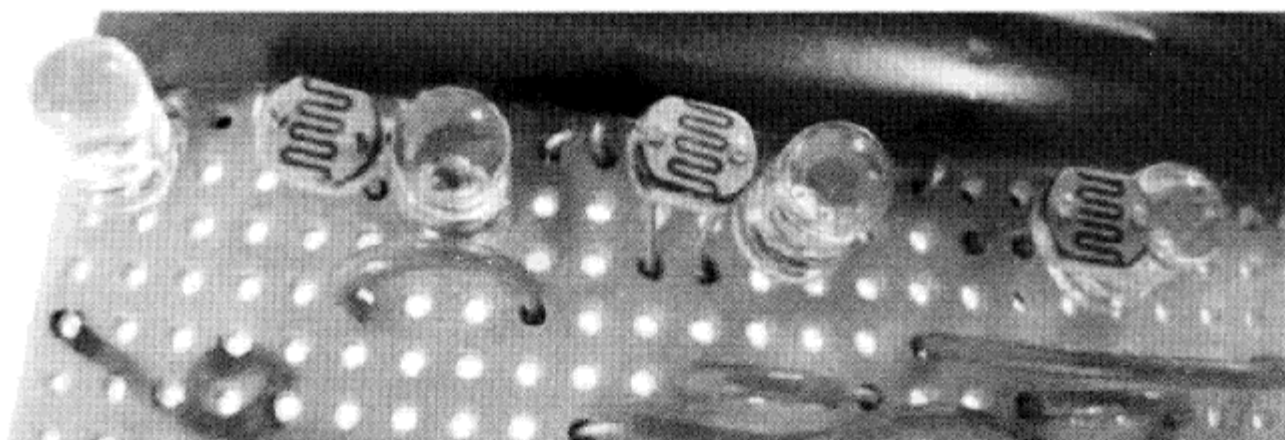


图 10-10 包含 4 个 LED 和 3 个 LDR 的传感器板

之后把传感器板安装到机器人的前端，这样 LED 和 LDR 距地面大概有 1cm 或 0.5 英寸高（见图 10-11）。这里我简单地把传感器板插接在机器人前端，但是作为一个永久解决方案，传感器应用螺栓连在底板上。

电池盒应该放在机器人内部的中间位置，这样不会破坏两轮之间的平衡，避免使机器人翻倒。用胶水把电池盒固定在底板上，保证在机器人运动时它不会左右晃动。

就像你在图 10-11 中看到的那样，有导线连接到电机盒和两个电机之间，也有一些导线连到传感器板上。一根线用来提供 5V 电源，一根线接地，一根线将数字引脚 9 连接到

5V 上，另外 3 根线连到模拟传感器 0、1 和 2 上。0 是左侧传感器（从机器人的前方看），1 是中间传感器，2 是右侧传感器。这是非常重要的，你应当把顺序搞对，否则机器人将不能正常工作。图 10-12 是我所做的机器人在厨房地板上运行的图片（这里在 YouTube 和 Vimeo 上也有视频，可以在这两个网站上找到）。

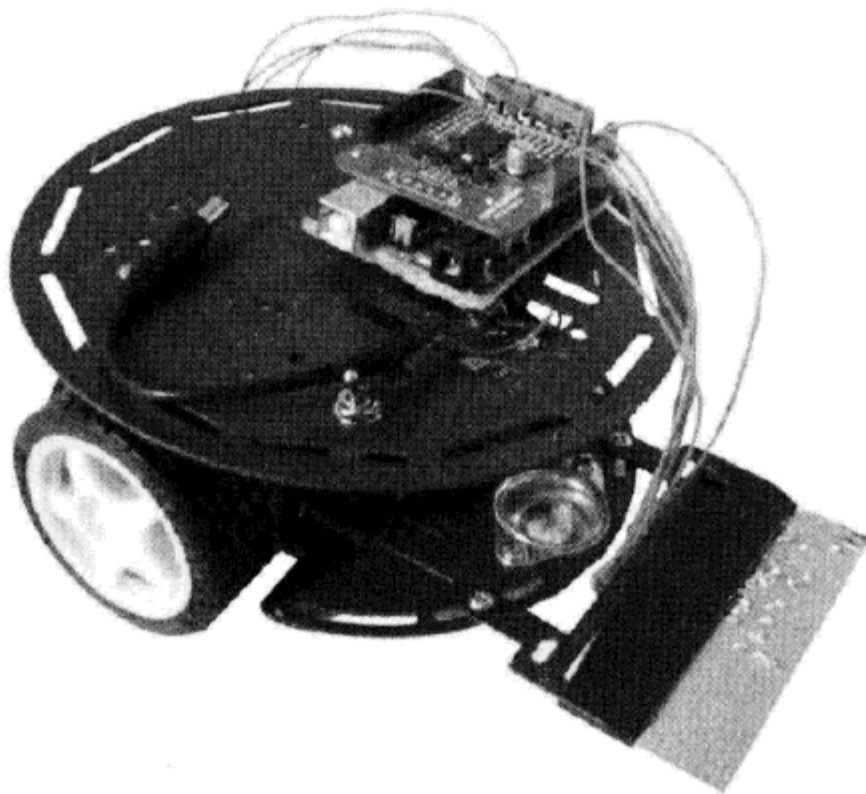


图 10-11 连接了传感器板的机器人底板

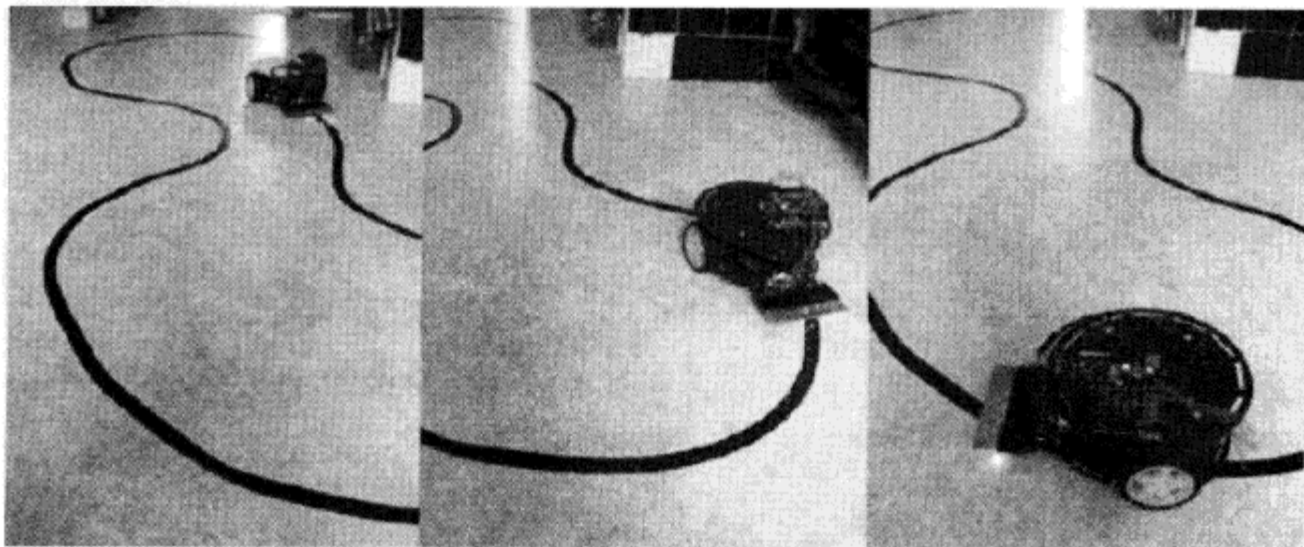


图 10-12 我自己的巡线机器人在运行

输入代码

将机器人做好之后，再检查一遍，然后输入清单 10-3 中的代码。

清单 10-3 项目 30 的代码

```

//项目 30——巡线机器人

#define lights 9
int LDR1, LDR2, LDR3; //传感器值

//计算补偿值
int leftOffset = 0, rightOffset = 0, centre = 0;
//电机速度和方向引脚
int speed1 = 3, speed2 = 11, direction1 = 12, direction2 = 13;
//电机的初始速度和速度偏离值
int startSpeed = 70, rotate = 30;
//传感器临界值
int threshold = 5;
//左右电机的初始速度
int left = startSpeed, right = startSpeed;

//传感器计算函数
void calibrate() {

    for (int x=0; x<10; x++) { //运行 10 次获得平均值
        digitalWrite(lights, HIGH); //开 led 灯
        delay(100);
        LDR1 = analogRead(0); //读 3 个传感器
        LDR2 = analogRead(1);
        LDR3 = analogRead(2);
        leftOffset = leftOffset + LDR1; //加入左侧传感器值中
        centre = centre + LDR2; //加入中间传感器值中
        rightOffset = rightOffset + LDR3; //加入右侧传感器值中

        delay(100);
        digitalWrite(lights, LOW); //关灯
        delay(100);
    }
    //获得每个传感器的平均值
    leftOffset = leftOffset / 10;
    rightOffset = rightOffset / 10;
    centre = centre / 10;
    //计算左右传感器的偏离量
    leftOffset = centre - leftOffset;
    rightOffset = centre - rightOffset;
}

```

```

}

void setup()
{
    //设置电机引脚模式为输出
    pinMode(lights, OUTPUT); //灯的引脚模式
    pinMode(speed1, OUTPUT);
    pinMode(speed2, OUTPUT);
    pinMode(direction1, OUTPUT);
    pinMode(direction2, OUTPUT);
    //运行 calibrate 函数
    calibrate();
    delay(3000);

    digitalWrite(lights, HIGH); //点亮发光二极管
    delay(100);

    //设置电机转动方向为向前
    digitalWrite(direction1, HIGH);
    digitalWrite(direction2, HIGH);
    //设定两台电机的速度
    analogWrite(speed1, left);
    analogWrite(speed2, right);
}

void loop() {

    //使两台电机具有相同的速度
    left = startSpeed;
    right = startSpeed;

    //读传感器的值并加上补偿值
    LDR1 = analogRead(0) + leftOffset;
    LDR2 = analogRead(1);
    LDR3 = analogRead(2) + rightOffset;

    //如果 LDR1 值大于中间传感器值+阈值, 则向右转
    if (LDR1 > (LDR2+threshold)) {
        left = startSpeed + rotate;
        right = startSpeed - rotate;
    }
}

```

```

//如果 LDR3 值大于中间传感器值+阈值，则向左转
if (LDR3 > (LDR2+threshold)) {
    left = startSpeed - rotate;
    right = startSpeed + rotate;
}
//将速度值送到电机
analogWrite(speed1,left);
analogWrite(speed2,right);
}

```

在平整地面上为机器人画出一条曲线。我在厨房地板上铺上油毡纸，用黑色的电工胶带粘出曲线。打开机器人，将它放在所画的黑线的附近，但是不要放在黑线上。机器人路径检测程序启动，这时 LED 连续闪烁 10 次。一旦 LED 闪烁停止，你要在 3 秒内拿起机器人并把它放在线上。如果一切顺利，机器人就会跟随黑线运动。不要让机器人急拐弯，因为快速转动时只用 3 个 LDR 不能检测到黑线的位置。图 10-12 是一个用黑电工胶带粘成曲线路径的例子。

代码回顾

首先，定义宏 lights 为 9，之后声明 3 个整型数，它们将存储从 3 个光电阻中读出的值：

```

#define lights 9
int LDR1, LDR2, LDR3; //传感器值

```

之后，声明另外 3 个整型值，它们将存储 3 个传感器计算出的补偿值（这将在后面解释）：

```

int leftOffset = 0, rightOffset = 0, centre = 0;

```

随后，定义控制两个电机速度和方向的引脚：

```

int speed1 = 3, speed2 = 11, direction1 = 12, direction2 = 13;

```

接下来，定义两个整型数，存储电机的初始速度和速度偏离值，速度偏离值指的是为使机器人转弯而使每个轮子运行速度增减的值：

```

int startSpeed = 70, rotate = 30;

```

程序默认的速度设置为 70，这大约是最高速的 27%。可能要根据你所使用的机器人调整这个值。太高的速度值将使机器人越过黑线而不能及时调整方向。如果摩擦力不够，机器人可能会打滑。

`rotate` 值是指为使机器人转弯轮子的速度上升或下降的值。在我的例子中，推荐值是 30，因此当向左转时，右侧轮子的速度是 100，左侧轮子的速度是 40 ($70+30$ 和 $70-30$)。你要根据自己的需要设置 `rotate` 值。

另外定义一个整数存储传感器的临界值。

```
int threshold = 5;
```

这是一个中间传感器和的左右传感器检测值的对比度的阈值，超过这个阈值，机器就开始转弯。在我的例子中，该值设置为 5 时，机器人工作得很好。这意味着机器人在转向动作之前，需要判断左右两侧的传感器检测到的值是否高于从中间传感器读出的值加 `threshold`。换句话说，如果中间传感器的读出值是 600，左边传感器读出值是 603，那么机器人保持直线行进。但是，如果左边传感器的值是 612（高于中间传感器读到的值加 `threshold`），意味着黑线已经偏离左侧传感器，需要返回到中线，也就是机器人的运行偏左侧了，需要电机做相应的调整使机器人向右转，返回到黑线上。

`threshold` 值在不同的情况下差异很大，取决于你的地板和黑线之间的对比度。这个值需要根据实际情况调整，以保证机器人不会太偏离中线运行。

最后设置一个变量，存储左侧或右侧电机的速度值。这个值存储在 `startSpeed` 中：

```
int left = startSpeed, right = startSpeed;
```

将这些变量全部声明并初始化。接下来是 `calibrate` 函数：

```
void calibrate() {
```

这个函数的目的有两个。首先，它对每个传感器连续测到的 10 次数据取平均值。其次，它用 LED 闪烁 10 次（当从 3 个传感器中读值时），表示机器人正在计算，并且马上就要开始运行了。

传感器需要校准，因为这 3 个传感器的读数都不相同。每个 LDR 所测到的数值略微不同，这是由制造误差、分压电阻的误差和线路中的电阻造成的。你希望 3 个传感器的读数

相同，因此让每个传感器测量 10 次，计算出它们的平均值，然后计算出左侧和右侧值传感器与中间传感器的差值（中间传感器的读数作为基准）。

程序从一个 for 循环开始，它内部的代码运行 10 次：

```
for (int x=0; x<10; x++) {
```

连接到引脚 9 的 LED 打开，之后延时 100 毫秒：

```
digitalWrite(9, HIGH);  
delay(100);
```

之后，将 3 个传感器中读出的值分别存储在 LDR1、LDR2 和 LDR3 中：

```
LDR1 = analogRead(0);  
LDR2 = analogRead(1);  
LDR3 = analogRead(2);
```

现在，把这些值赋给变量 leftOffset、center 和 rightOffset。由于这些变量的初始值为 0，因此循环 10 次后，变量的值为每个传感器 10 次读数的总和：

```
leftOffset = leftOffset + LDR1;  
centre = centre + LDR2;  
rightOffset = rightOffset + LDR3;
```

之后，等待 100 毫秒，关闭 LED，等 100 毫秒。重复这个过程：

```
delay(100);  
digitalWrite(9, LOW); //关灯  
delay(100);
```

这个过程重复 10 次后，退出 for 循环。之后用 10 除每一个和值，得到 3 个 LDR 传感器 10 次读数的平均值：

```
leftOffset = leftOffset / 10;  
rightOffset = rightOffset / 10;  
centre = centre / 10;
```

之后计算偏离量，用中间的传感器的测量值分别减去左、右传感器的测量值：

```
leftOffset = centre - leftOffset;
rightOffset = centre - rightOffset;
```

这些值将分别加到左、右传感器读出的值上，因此这 3 个传感器给出近似相同的数值。这个数值表明 3 个读数中一个由传感器制造误差等原因造成的确定的不同，经过用这个数值补偿后使得检查地板和黑线之间的对比度更容易。

之后，设置引脚模式。首先设置与 LED 连接的引脚模式为 OUTPUT，之后设置连接电机速度、转向的引脚模式为 OUTPUT：

```
pinMode(9, OUTPUT);           //点亮
pinMode(speed1, OUTPUT);
pinMode(speed2, OUTPUT);
pinMode(direction1, OUTPUT);
pinMode(direction2, OUTPUT);
```

现在运行 `calibrate` 函数，保证 3 个传感器有相同的读数。之后延时 3 秒。在执行 `calibrate` 函数时 LED 闪烁 10 次。你有 3 秒时间将机器人放置到它要跟随的路线上：

```
calibrate();
delay(3000);
```

发光二极管点亮，之后延时一小段时间：

```
digitalWrite(9, HIGH); //点亮发光二极管
delay(100);
```

两个电机的方向都设置为向前，电机的速度设置为 `right` 和 `left` 变量的值（初始值存储在 `startSpeed` 中）：

```
digitalWrite(direction1, HIGH);
digitalWrite(direction2, HIGH);
analogWrite(speed1, left);
analogWrite(speed2, right);
```

接下来是主循环程序。程序开始用 `startSpeed` 中的值设置左右电机的速度：

```
left = startSpeed;
right = startSpeed;
```

在每一次循环开始时，电机速度重新设置为以上数值。因此机器人总是向前运动，除非在循环中改变这些值而使机器人转向。

现在读取三个传感器的检测值，并且把值存储到 LDR1、LDR2 和 LDR3 整型变量中。将补偿值加到左、右传感器的读数上，这样，当检测同一地面时，这 3 个传感器读到的数值大致相同：

```
LDR1 = analogRead(0) + leftOffset;
LDR2 = analogRead(1);
LDR3 = analogRead(2) + rightOffset;
```

接下来需要检查传感器的读数，检测机器人运行轨迹与黑线的距离是否超过阈值。也就是检查左、右传感器检测到的数值是否比从中间 LDR 读到的值加上 threshold 所得到的数值还要大。

如果从左侧传感器中读出来的值大于中间传感器的读数与偏差之和，则黑线已经向右侧偏离中间位置太多了。之后，对电机的速度进行调整，使左轮旋转速度比右轮旋转速度快，因此机器人向右偏转，这将使黑线返回到中间位置。

```
if (LDR1 > (LDR2+threshold)) {
    left = startSpeed + rotate;
    right = startSpeed - rotate;
}
```

当右侧传感器读数大于中间传感器读数加上偏差值时，重复类似操作，使机器人向左转：

```
if (LDR3 > (LDR2+threshold)) {
    left = startSpeed - rotate;
    right = startSpeed + rotate;
}
```

将调整后的速度值送到电机：

```
analogWrite(speed1, left);
analogWrite(speed2, right);
```

整个过程每秒重复多次，形成反馈控制，使机器人跟随黑线运行。

如果机器人偏离黑色中线，需要将 `startSpeed` 中的速度值调小一点。如果机器人调整方向时转角过小或过大，需要调整相应的 `rotate` 值。LDR 的灵敏度可以通过调整 `threshold` 变量来改变。不断调整这些参数，直到机器人巡线成功。

小结

第 10 章从一个简单的例子开始，介绍了如何使用 IC 控制步进电机和控制电机的速度和方向的相关知识。本章也指出了单极和双极步进电机的不同和它们的工作原理。随后给出一个使用电机模板单独地控制两个独立电机的例子。本章最后将这两台电机安装到机器人底盘上做了一个两轮巡线机器人。

你已经学习了电机模板的相关知识和它们的使用方式，并将它应用到一个实际的项目中；你也学会了如何像控制舵机（第 9 章）那样控制步进电机和标准电机：这意味着你已经具备了自己制作机器人或互动设备的能力。

本章的主题和概念

- 单极和双极步进电机的区别
- 如何将步进电机连接到电机驱动模块上
- 使用电容平滑信号，减少干扰
- 在代码中如何建立一个步进电机对象
- 设置电机的速度和步数
- 使用正数和负数控制电机的方向
- 步进电机是如何工作的
- 驱动步进电机的线圈激励顺序
- 使用半步法或细分法提高步进电机的精度
- 电机模板的使用方法
- 使用电机模板控制几个电机的速度和方向

- Arduino 拓展板及其种类
- 如何用不同的 LDR 检测光线对比度
- 如何获得传感器读数的平均值
- 如何使用偏差值平衡传感器读数
- 如何做一个实用的巡线机器人



第 11 章

压力传感器

现在看一下压力传感器，特别是 VTI 公司的 SCP1000 绝对数字压力传感器。这是一个很好的传感器，它可以很容易地实现与 Arduino 接口并且提供压力和温度读数。这个元件需要连接到 Arduino 的 SPI（串行通信接口）总线上进行数据交换。SPI 对你来说是一个新的概念。尽管本章没有非常详细地介绍它的信息（SPI 本身需要一到两章进行讲解），但是你会学到 SPI 的基本概念及如何利用它从 SCP1000 传感器中获得数据。

绝对压力传感器是制作天气预报系统的一个理想的选择。从本章开始你将学习如何将 SCP1000 连接到 Arduino 上，并且使用串口监视器从中读取数值。还要使用传感器和一个 LCD 显示 24 小时的气压曲线。在做这个项目的过程中你也会学到如何控制 LCD（图形 LCD）显示器。

项目 31——数字压力传感器

在本项目中要使用 Arduino Mega，因为项目 32 所用的 GLCD 需要 Mega 提供的额外的引脚。如果没有 Mega，也可以使用标准 Arduino——只要调整 SPI 的引脚，使其与 Duemilanove 的引脚匹配就可以。

需要的元件

可以从 Sparkfun 上或它们的分销商处购买 SCP1000 传感器。该传感器预先焊接在一块 PCB 上（见图 11-1），使它容易与 Arduino（或其他的微控制器）接口。如果你希望把它插到面包板上，需要事先焊一些针型引脚到板上。不然，将一些导线焊到它上面，使它能连接到 Mega 上。

Arduino Mega



SCP1000 压力传感器



3 个 10kΩ电阻



1 个 1kΩ电阻

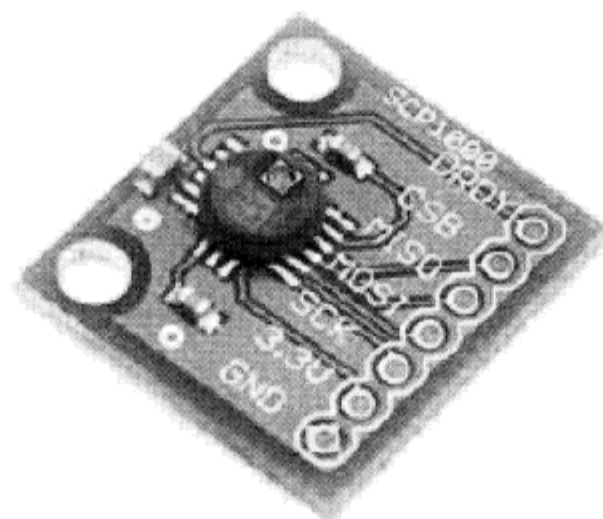


图 11-1 Sparkfun 上的焊好的 SCP1000 传感器

把元件连接起来

像图 11-2 那样连接每个元件。

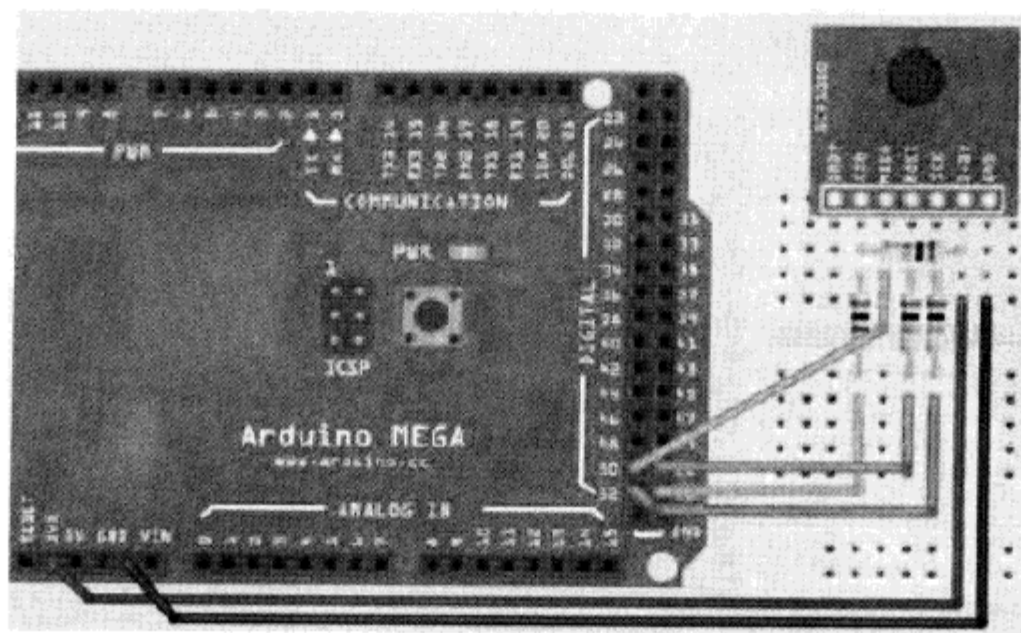


图 11-2 项目 31——数字压力传感器电路图（参见彩色版插图）

传感器有 TRIG 和 PD 引脚，它们都需要连到地上。确保每根连线连接正确，特别是电阻，因为电阻起到保护 SCP1000 的作用，防止过电压。现在输入代码。

输入代码

输入清单 11-1 中的代码。

清单 11-1 项目 31 的代码

```

/*
SCP1000          Mega
DRDY             N/A
CSB              53 via Logic Level Convertor
MISO             50 (straight through)
MOSI             51 via Logic Level Convertor
SCK              52 via Logic Level Convertor
3.3v            3.3v
GND              GND
TRIG             GND
PD              GND
*/

//SPI 引脚
#define SLAVESELECT 53
#define SPICLOCK 52
#define DATAOUT 51 //主器件数据输出，从器件数据输入
#define DATAIN 50  //主器件数据输入，从器件数据输出
#define UBLB(a,b) ( ( (a) << 8 ) | (b) )
#define UBLB19(a,b) ( ( (a) << 16 ) | (b) )

//寄存器地址
#define PRESSURE 0x1F //压力数据的高 3 位
#define PRESSURE_LSB 0x20 //压力数据的低 16 位
#define TEMP 0x21 //16 位温度数据

char rev_in_byte;
int temp_in;
unsigned long pressure_lsb;
unsigned long pressure_msb;
unsigned long temp_pressure;
unsigned long pressure;

```

```

void setup()
{
    byte clr;
    pinMode(DATAOUT, OUTPUT);
    pinMode(DATAIN, INPUT);
    pinMode(SPICLOCK, OUTPUT);
    pinMode(SLAVESELECT, OUTPUT);
    digitalWrite(SLAVESELECT, HIGH); //将传感器挂起
    SPCR = B01010011; //SPI 控制寄存器
    //MPIE=0, SPE=1 (on), DORD=0 (MSB first), MSTR=1 (master), CPOL=0 (clock
idle when low), CPHA=0 (samples MOSI on rising edge), SPR1=0 & SPR0=0 (500kHz)
    clr=SPSR; // SPI 状态寄存器
    clr=SPDR; // SPI 数据寄存器
    delay(10);
    Serial.begin(38400);
    delay(500);

    write_register(0x03, 0x09); //高速读模式
    write_register(0x03, 0x0A); //高精度测量模式
}

void loop()
{
    pressure_msb = read_register(PRESSURE);
    pressure_msb &= B00000111;
    pressure_lsb = read_register16(PRESSURE_LSB);
    pressure_lsb &= 0x0000FFFF;
    pressure = UBLB19(pressure_msb, pressure_lsb);
    pressure /= 4;
    Serial.print("Pressure (hPa): ");
    float hPa = float(pressure)/100;
    Serial.println(hPa);

    Serial.print("Pressure (Atm): ");
    float pAtm = float(pressure)/101325.0;
    Serial.println(pAtm, 3);

    temp_in = read_register16(TEMP);
    float tempC = float(temp_in)/20.0;
    Serial.print("Temp. C: ");
    Serial.println(tempC);
}

```

```

    float tempF = (tempC*1.8) + 32;
    Serial.print("Temp. F: ");
    Serial.println(tempF);
    Serial.println();
    delay(1000);
}

char spi_transfer(char data)
{
    SPDR = data;          //开始传送
    while (!(SPSR & (1<<SPIF))) { }; //等待传送结束
    return SPDR;          //它返回寄存器 SPDR 中的值
}

char read_register(char register_name)
{
    char in_byte;
    register_name <<= 2;
    register_name &= B11111100; //读命令

    digitalWrite(SLAVESELECT, LOW); //使能 SPI 元件
    spi_transfer(register_name); //写字节到元件
    in_byte = spi_transfer(0x00); //什么也不送出, 但是得到寄存器的返回值
    digitalWrite(SLAVESELECT, HIGH); //关闭 SPI 元件
    delay(10);
    return(in_byte); //返回值
}

unsigned long read_register16(char register_name)
{
    byte in_byte1;
    byte in_byte2;
    float in_word;

    register_name <<= 2;
    register_name &= B11111100; //读命令

    digitalWrite(SLAVESELECT, LOW); //启动 SPI 元件
    spi_transfer(register_name); //向元件中写字节
    in_byte1 = spi_transfer(0x00);
    in_byte2 = spi_transfer(0x00);
    digitalWrite(SLAVESELECT, HIGH); //关闭 SPI 元件

```

```

        in_word = UBLB(in_byte1, in_byte2);
        return(in_word); //返回值
    }

void write_register(char register_name, char register_value)
{
    register_name <=& 2;
    register_name |= B00000010; //写命令

    digitalWrite(SLAVESELECT, LOW); //选择 SPI 元件
    spi_transfer(register_name); //送寄存器地址
    spi_transfer(register_value); //送记录在寄存器中的值
    digitalWrite(SLAVESELECT, HIGH);
}

```

本段代码是建立在 Conor 和 Arduino 论坛上其他人所做的工作的基础上的，在这里对他们表示感谢。而且，Arduino IDE 的 0019 版本有一个为 SCP1000 写的 SPI 例程，所用的 SPI.h 库是由 TomIgoe 编写的。

上面的代码较难读懂，因此你需要仔细阅读 SCP1000 与 Arduino 通信部分。一旦理解了这部分代码，你就可以通过使用 SPI 库简化工作。

上载了代码之后，打开串口监视器，将串口波特率设置为 38400，你会从显示器上看到一串数字，显示以 hPa 为单位的大气压力值。hPa 是在气象行业中通用的压力单位。你还可以看到用摄氏度或华氏度表示的温度。

代码回顾

代码开始时定义一系列的 Mega 引脚：

```

#define SLAVESELECT 53
#define SPICLOCK 52
#define DATAOUT 51 //主器件数据输出，从器件数据输入
#define DATAIN 50 //主器件数据输入，从器件数据输出

```

如果你使用的是 Duemilanove，引脚将定义为：

```

#define SLAVESELECT 10
#define SPICLOCK 13
#define DATAOUT 11 //主器件数据输出，从器件数据输入

```



```
#define DATAIN 12 //主器件数据输入，从器件数据输出
```

之后定义两个宏：

```
#define UBLB(a,b) ( ( (a) << 8) | (b) )
#define UBLB19(a,b) ( ( (a) << 16 ) | (b) )
```

第一个宏用两个 8 比特数字量合成一个 16 比特数字量，将其中一个作为最低有效字节 (LSB)，另一个作为最高有效字节 (MSB)。两个宏主要使用移位和按位操作指令。

例如，如果你有两个 8 比特数 (10010101 和 00111001)，把它们带入公式：

```
UBLB (a, b) (((a) <<8) | (b))
```

进行如下计算：

```
((B10010101) <<8) | (B00111001))
```

因此，第一个比特数左移 8 位，得到：

```
B1001010100000000
```

这个数与第二个数字进行按位或操作，生成如下数字：

```
B1001010100111001
```

通过以上操作，它把两个 8 位数结合为一个 16 位数，使第一个数为高 8 位，另一个数为低 8 位。

第二个宏做类似的事情：

```
#define UBLB19(a,b) ( ( (a) << 16 ) | (b) )
```

这次它将生成一个 19 位数，这是 SCP1000 压力传感器的分辨率，将数据 a 的开头 3 个比特左移 16 位，并将后 16 位与数据 b 相加。

之后，定义三个存储器为传感器的内部寄存器，以存储压力和温度数据：

```
#define PRESSURE 0x1F //压力数据的高 3 位
#define PRESSURE_LSB 0x20 //压力数据的低 16 位
```

```
#define TEMP 0x21           //16 位温度数据
```

PRESSURE 寄存器的地址是 0x1F, 存储表示压力的 19 比特数值中 3 个最重要的比特。PRESSURE-LSB 寄存器的是地址 0x20, 存储其余 16 位比特数。通过使用 USLB19 (a, b) 定义的计算, 16 比特数和 3 比特数将结合成一个 19 比特数。

之后, 声明了一些变量, 用来存储从压力传感器和温度传感器中读到的值:

```
char rev_in_byte;
int temp_in;
unsigned long pressure_lsb;
unsigned long pressure_msb;
unsigned long temp_pressure;
unsigned long pressure;
```

接下来是 setup 函数, 首先声明一个 byte 型的本地变量 (后面说明如何使用它):

```
byte clr;
```

之后, 设置组成 SPI 总线的 4 个引脚, 将它们相应地设置为 INPUT 和 OUTPUT 状态:

```
pinMode(DATAOUT, OUTPUT);
pinMode(DATAIN, INPUT);
pinMode(SPICLOCK, OUTPUT);
pinMode(SLAVESELECT, OUTPUT);
```

之后, 设置从元件选择引脚为高电平状态。在调试过程中, 不希望与元件交换数据时, 采用该方式将该元件挂起:

```
digitalWrite(SLAVESELECT, HIGH); //将传感器挂起
```

之后, 用二进制值 B01010011 设置 SPCR:

```
SPCR = B01010011; //SPI 控制寄存器
```

你可能已经注意到这个变量还没有声明, 但是代码却可以正常工作, 为什么呢? 原来 SPCR 专指 SPI 控制寄存器, 它是 SPI 使用的 3 个寄存器之一。SPI 通信代码已经做到 Arduino IDE 中。另外两个是 SPSR (SPI 状态寄存器) 和 SPDR (SPI 数据寄存器)。无论什么时候给这 3 个寄存器中的任何一个赋值, 都将改变 SPI 总线上正在进行的工作。

让我们先回过来看一下什么是 SPI 及它是如何工作的。

SPI—串行外围芯片接口

在你弄懂上述代码之前，需要理解什么是 SPI 及它是如何工作的。SPI 是一个复杂的主题，因此我不准备对它进行深入讨论，毕竟这只是一本入门书籍。我将给出足够的知识使读者理解什么是 SPI 总线 and 它是如何工作的，以及与项目 31 和 32 有关的 SCP1000 传感器元件所用到的 SPI 总线知识。你将能够理解元件如何通过 SPI 总线与其他元件进行通信。

SPI 是两个元件交换信息的方法。它的优点是仅使用 Arduino 中的 4 个引脚，并且通信速度快。SPI 是一个同步通信协议，允许主设备与从设备同时进行通信。数据传递用时钟信号控制，时钟信号决定了什么时候进行数据传递，什么时候读取数据。SPI 总线的时钟频率是可以变化的，不像其他的一些协议，时钟信号必须是非常精确的。这对于微控制器是非常理想的，因为微控制器运行速度不是特别快，并且它的内部振荡器计时不是特别精确。

SPI 是主-从协议（见图 11-3），这意味着一个主元件控制时钟信号。只有在接收到时钟脉冲信号时，才可以进行数据传递。SPI 也是一个数据交换协议，其意思是总线送出数据的同时接受新的数据，元件不仅仅是简单地传送或接收数据，它们之间必须互相交换数据，即使这些数据对其中一方可能是没有用的。

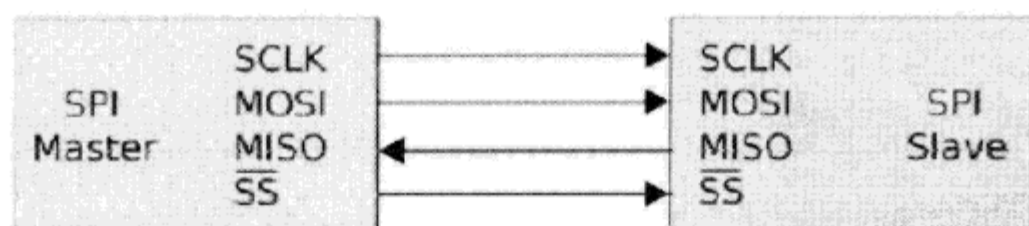


图 11-3 SPI 主-从协议（感谢 Colin M.L.Burnett 供图）

当主元件连接多个从元件时（见图 1-4），从元件的选择引脚（SS 引脚）控制哪个从元件什么时候可用。当只有一个从元件时，例如，在我的例子中，SS 引脚（在 SCP1000 上叫做 CSB 引脚）是可选的。然而，通常情况下，应该使用它，而不管它是否被用做这个元件接收下一个字节的复位键。从元件转发主元件送出的信号，告诉从元件，主元件想要开始通信。当这个引脚是低电平状态时，SPI 数据交换是激活的，因此当这个引脚保持高电平时，该从元件没有被激活。

数据仅仅在 SCK 上的时钟信号处于上升或下降沿时才能输出，与之对应的，数据在 SCK 时钟信号处于下降沿或上升沿时锁存，这就是时钟的极性。时钟的极性由主元件根据

SPCR 寄存器中的一个标志位设置。SPI 有两个数据线分别叫做 MOSI（主输出、从接收）和 MISO（主接收、从输出）。因此，如果元件设置为时钟脉冲处于上升沿时，主芯片送出数据，则时钟脉冲处于下降沿时，数据必须由从元件送回。因此主元件在一个脉冲周期内同时送出和接收数据。

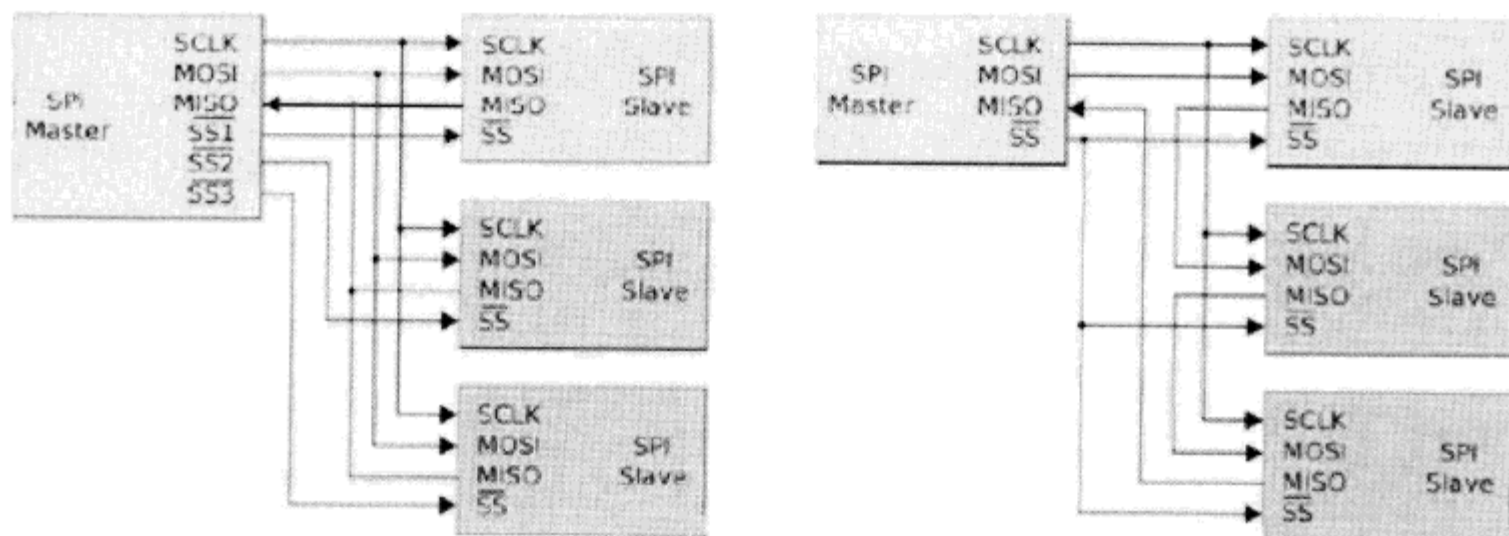


图 11-4 左：有 3 个独立从元件的总线。右：菊花链方式（感谢 Colin M.L.Burnett 供图）

记住，如果你仅仅是想从元件中读取数据（就像你用 SCP1000 做的那样），在通信时仍然需要同步发送数据。

SPI 总线使用的 3 个寄存器是：

- SPCR-SPI 控制寄存器
- SPDR-SPI 数据寄存器
- SRSR-SPI 状态寄存器

控制寄存器是 8 位的，每一位控制一个特别的 SPI 设置，如表 11-1 所示。

表 11-1 SPI 控制寄存器设置

7	6	5	4	3	2	1	0
SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0

- SPIE——SPI 中断使能——如果该位设置为 1，开启 SPI 中断。
- SPE——SPI 使能——如果该位设置为 1，开启 SPI。
- DORD 数据请求——如果该位是 1，首先传递最低有效字节；如果该位是 0，则传递最高有效字节。

- **MSTR**——主/从选择——如果该位是 1，设置 Arduino 为主模式；当它是 0 时，设置 Arduino 为从模式。
- **CPOL**——时钟极性——当高电平设置为 1 或低电平设置为 0 时，时钟暂停。
- **CPHA**——时钟相位——确定当时钟处于上升沿还是下降沿时采样。
- **SPR1/0**——SPI 时钟选择 1 或 0——控制主设备的时钟频率。

需要改变这些设置的原因是，不同的 SPI 设备所期望时钟频率、时钟相位、数据命令、速度等是不同的。这主要是因为现在没有统一的 SPI 标准，制造商生产的元件都有微小差别。在代码中，你可以这样设置 SPCR：

```
SPCR = B01010011;
```

这样的设置表明不开启中断（SPIE=0），但允许使用 SPI（SPE=1），设置数据请求为 MSB 优先（DORD=0），设置 Arduino 为主元件（MSTR=1），设置时钟极性为低电平时闲置（CPOL=0），设置时钟相位为在上升沿时采样（CPHA=0），设置速度为 250kHz（SPR1/2=11），这是 1/64 的 Arduino 晶振频率。

SPI 状态寄存器（SPSR）使用 3 个比特设置 SPI 传递的状态。这里只关心第 7 个比特（SPIF-SPI 中断标志），它告诉你数据传送是否已经完成。如果传送已经完成，它被设置为 1。当读到 SPSR 7 个位中的第一个时这个位被置零（设为 0），之后用 SPI 数据寄存器（SPDR）接收数据。

SPDR 只存储一个将要在 MOSI 线上送出的字节，并且存储从 MISO 线上接收到的字节。

以上过程听起来有点复杂，但是其中的许多细节你不需要知道（就是这样）。SPI 总线可用通俗语言解释为：一个主元件，一个从元件，它们之间要互相交流。时钟脉冲保证在时钟脉冲上升（或下降，依赖于你在 control 寄存器中如何设置）时，将数据从主设备送到从设备，保证数据在时钟下降（或上升）时由从设备传送到主设备。在数据传送完成后，SPIF 设置为 1。

现在让我们返回到代码。

代码回顾（继续）

现在从 SPSR 和 SPDR 寄存器中读取数据，并赋值给 `clr`。在读取和存储数据之前，要先清除寄存器中的数据，为下一步工作做好准备。

```
clr=SPSR; //SPI 状态寄存器
clr=SPDR; //SPI 数据寄存器
```

之后延时一小段时间，设置波特率，接下来又是 500 毫秒延时，使串行线完成设置。

```
delay(10);
Serial.begin(38400);
delay(500);
```

现在使用 `write_register` 函数（之后解释）将两个值写入传感器的操作寄存器中，寄存器的十六进制地址是 `0x03`。将值 `0x09` 写入这个寄存器中，设置传感器为高速读模式。并且写入值 `0x0A`，设置传感器为高分辨率模式，保证从传感器中得到更精确的值。

```
write_register(0x03,0x09); //高速读模式
write_register(0x03,0x0A); //高精度测量模式
```

接下来是主循环程序。开始时，从 `PRESSORE` 寄存器中读取数值，（地址是 `0x1F`），并把它存储在 `pressure_msb` 中，这个值就是 19 位值中的最高有效位（也就是你需要的 3 比特）。

```
pressure_msb = read_register(PRESSURE);
```

之后对这个值进行按位运算。对 `pressure_msb` 和一个二进制数 `B00000111` 进行按位与操作 `AND (&)`。记住，这是按位与操作符，如果两个操作位都是 1，操作结果是 1，如果有一个是 0，操作结果是 0。将前三个比特设置为 1 意味着仅取前三个比特，这正是我们所希望的。

```
pressure_msb &= B00000111;
```

你现在想要获得 16 比特的压力值，因此读存储在 `PRESSURE_LSB` 寄存器中的值，寄存器的地址是（`address 0x20`）：

```
pressure_lsb = read_register16(PRESSURE_LSB);
```


之后用这个值和 0x0000FFFF 执行按位与操作 AND。0x0000FFFF 用二进制数表示为 B1111111111111111，意思是取 16 比特中的全部值：

```
pressure_lsb &= 0x0000FFFF;
```

之后使用在 UBLB19 中的宏定义，把 16 比特数和 3 比特数合成一个 19 比特数：

```
pressure = UBLB19(pressure_msb, pressure_lsb);
```

SCP1000 的说明书告诉我们，压力值是一个整数，将它从一个十进制整数转为以 Pa（压力的测量单位）为单位的值，需要用 4 除这个数：

```
pressure /= 4;
```

现在，已经获得了压力数据，需要将它送到串口监视器中，使你可以读到这个数据，因此要把压力值打印出来。首先通过除以 100 将它转化成以百帕为单位的数值，并且把它存储在一个叫 hPa 的浮点变量中，必须打印成浮点数确保小数点后有两位有效数字。

```
Serial.print("Pressure (hPa): ");
float hPa = float(pressure)/100;
Serial.println(hPa);
```

接下来，你再打印出压力数据，压力的单位是大气压。通过除以值 101325 获得一大气压为单位的压力数据，一大气压等于 101325Pa。下面的代码第三行中的 pAtm 后面的数字“3”告诉串口监视器打印到小数点后 3 位：

```
Serial.print("Pressure (Atm): ");
float pAtm = float(pressure)/101325.0;
Serial.println(pAtm, 3);
```

接下来，将 TEMR 寄存器（地址 0x21）里的数据传送给该函数，通过调用 read_register 函数读温度数据。从这个函数返回的值存储在 temp_in 变量中，温度是一个 14 比特的数值。

```
temp_in = read_register16(TEMP);
```

这个值转化成浮点数之后，除以 20 得到摄氏温度值：

```
float tempC = float(temp_in)/20.0;
```

```
Serial.print("Temp. C: ");
Serial.println(tempC);
```

之后，把这个值乘以 1.8 加 32 得到华氏温度值，并且把它输出：

```
float tempF = (tempC*1.8) + 32;
Serial.print("Temp. F: ");
Serial.println(tempF);
```

现在有 4 个函数允许你通过 SPI 总线从传感器中读取数据。第一个是 `spi_transfer` 函数。要传递一个字符给该函数，并返回一个字符，因此这个函数的形参是字符型，函数的返回值也是字符型。

```
char spi_transfer(char data)
```

给这个函数传递的参数要传递给 SPI 数据寄存器：

```
SPDR = data;
```

之后等待，直到 SPIF 标志表明数据已成功送出：

```
while (!(SPSR & (1<<SPIF))) { };
```

在代码块中不执行任何操作，直到 SPIF 被设置。通过对 SPSR 和 SPIF 标志进行按位操作实现这个功能：

```
!(SPSR & (1<<SPIF))
```

因此，我们将以上步骤分成两个部分，首先执行：

```
(1<<SPIF)
```

所有这些工作是为了要在 7 个比特位中移动 1 比特的位置，准备用它来做比特掩码。SPIF 定义在宏中，如 7（对于 Atmega 芯片）。

之后将这个比特掩码与 SPSR 中的值进行按位与操作。

```
(SPSR & (1<<SPIF))
```

如果 SPIF 标志设置为 1，以上结果是非零值。因此，如果 SPSR 为 B01010101，并且 SPIF 标志是 1，那么计算公式是 $(B01010101 \& (1 \ll 7))$ ，所得到的计算结果是 B00000000。因为如果两个操作数相应的位都是 1，进行按位与操作之后的结果还是 1。

但是如果 SPSR 是 B11010101，并且 SPIF 标志是 1，那么 $B11010101 \& B10000000$ 的结果将是 B10000000。这是一个非零值，整个计算结果与逻辑非 ‘!’ 操作的结果相同。

```
!(SPSR & (1<<SPIF))
```

换句话说，如果 SPIF 标志没有设置为 1，程序将不做任何事情，并保持这种状态直到 SPIF 标志设置为 1。然后跳出 while 循环，执行下一条命令：

```
return SPDR;
```

它返回寄存器 SPDR 中的值，因此，整个函数从主元件送出一个字节到从元件，等待通信完成，然后再通过从元件向主元件返回 1 个字节的值。

下一个函数的作用是从 SCP1000 中的一个寄存器中读取 8 比特数值。它的参数和返回值都是字符型。

```
char read_register(char register_name)
```

声明另一个叫做 in_byte 的字符型变量。它将要存储从压力寄存器中读到数值：

```
char in_byte;
```

之后，寄存器地址向左移动两个比特的位置：

```
register_name <<= 2;
```

现在通过把选择引脚设置为低电平状态，启动 SPI 设备：

```
digitalWrite(SLAVESELECT, LOW); //使能 SPI 元件
```

之后，将寄存器的名称送到元件中：

```
spi_transfer(register_name); //写字节到元件
```

之后，送出另外一个字节，但是这次送出的不是任何有意义的值，但是得到寄存器的返回值：

```
in_byte = spi_transfer(0x00); //什么也不送出，但是得到寄存器的返回值
```

int_tye 存储从压力传感器中读出的 MSB 值。之后，SPI 线不启动，并且返回 MSB 的压力传感器读值到主循环中：

```
digitalWrite(SLAVESELECT, HIGH); //关闭 SPI 元件
delay(10);
return(in_byte); //返回值
```

下一个函数是 read_register16()，除了返回的不是 8 比特字节外它的作用同上面的函数类似。这个函数返回 16 比特的数据。寄存器的地址再次传递给函数。

```
unsigned long read_register16(char register_name)
```

之后，声明两个字节和一个浮点数：

```
byte in_byte1;
byte in_byte2;
float in_word;
```

这两个字节分别存储 16 比特数中的两个 8 比特，浮点数将存储从函数返回的 16 位数。这个函数的其他部分与前一个完全相似，只是这次返回的是两个 8 比特数，而不是一个 8 比特数。

```
register_name <= 2;
digitalWrite(SLAVESELECT, LOW); //启动 SPI 元件
spi_transfer(register_name); //向元件中写字节
in_byte1 = spi_transfer(0x00);
in_byte2 = spi_transfer(0x00);
digitalWrite(SLAVESELECT, HIGH); //关闭 SPI 元件
```

UBLB 命令将采用按位操作命令将这两个字节合成一个 16 位数，之后作为结果从函数返回：

```
in_word = UBLB(in_byte1,in_byte2);
```

```
return(in_word); //返回值
```

函数最后所做的工作是写一个值到 SCP1000 的寄存器中，这个函数不返回任何值，因此它的数据类型是 void。该函数有两个参数——寄存器地址和要写入的值：

```
void write_register(char register_name, char register_value)
```

寄存器写命令是把寄存器地址的比特移动两位之后，与二进制数 B00000010 进行按位或操作所生成的命令：

```
register_name <<= 2;
register_name |= B00000010; //写命令
```

SPI 线打开，寄存器地址和它所储存的数据通过 SPI 线传递给元件。然后，SPI 线在函数结束之前再次关闭。

```
digitalWrite(SLAVESELECT, LOW); //选择 SPI 元件
spi_transfer(register_name); //送寄存器地址
spi_transfer(register_value); //送记录在寄存器中的值
digitalWrite(SLAVESELECT, HIGH);
```

这个项目简单地读取（即使使用 SPI 不是那么简单的，但是幸运的是现在你理解它了）温度和压力，并发送它们到串口监视器。因为 SCP1000 是封装的元件，我们并没有对硬件做详细说明。你只需知道它是压力和温度传感器，并且通过 SPI 线传递数据。

下面让我们使用压力传感器读取技术做一些有用的东西。

项目 32——数字气压表

现在你可以连接上 SCP1000，利用它获得数据，做一些应用项目。这个项目是要做一个数字气压表，能够显示随时间变化的压力图，为了显示这个图，要使用 GLCD（图形 LCD）。通过这个项目你将学习到如何像显示文字那样显示图形。

需要的元件

元件表与项目 31 相同，所增加的是一个 128×64 点的 GLCD、一个电阻和一个变阻器。该项目需要使用 glcd.h 头文件，因为 GLCD 必须用 KS0108（或相当的）驱动芯片。在购

买 GLCD 之前，要仔细查看说明书。

Arduino Mega



SCP1000 压力传感器



3×10kΩ电阻



1×1kΩ电阻



150Ω电阻



10kΩ变阻器



128×64 图形 GLCD



把元件连接起来

如图 11-5 那样将每个元件连接起来。

SCP1000 元件部分的电路与项目 31 完全相同，这里只是额外加了一点元件到电路中。你使用的 GLCD 与本项目所使用的元件可能有不同的引脚。仔细阅读说明书，确保你的 GLCD 引脚与表 11-2 所列的引脚相匹配。

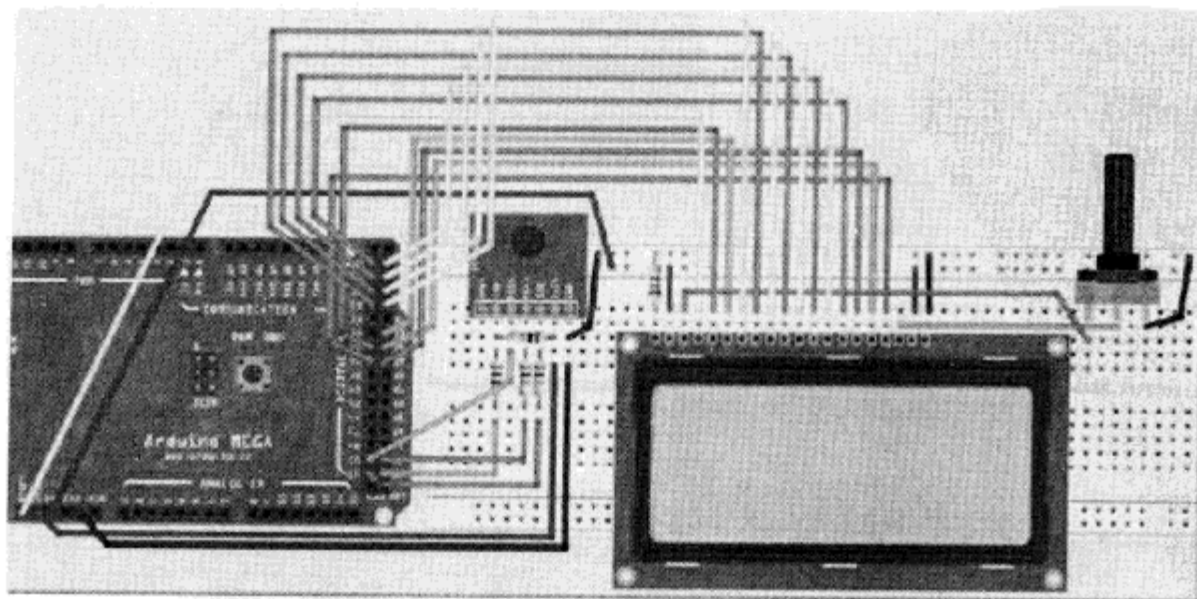


图 11-5 项目 32——数字气压表电路图（参见彩色版插图）

表 11-2 Mega 和 GLCD 之间的引脚

Arduino 开发板	图形 LCD	其它元件
地	地	
+5V	+5V	
	LCD 对比度	中间引脚
36D/I (输入/输出)		
35R/W (读/写)		
37Enab (使能)		
22DB0 (数字引脚 0)		
23DB1 (数字引脚 1)		
24DB2 (数字引脚 2)		
25DB3 (数字引脚 3)		
26DB4 (数字引脚 4)		
27DB5 (数字引脚 5)		
28DB6 (数字引脚 6)		
29DB7 (数字引脚 7)		
33C (引脚 C)	S1 引脚	
34C (引脚 C)	S2 引脚	
Reset Res (复位引脚)	Et 引脚	
	电源引脚	正极引脚
	背景灯+5V	+5V
	背景灯+0V	通过 150 Ω 电阻连接地

在将电路连接好并仔细检查各个连线之前，不要给电路上电。如果接线不对，很容易损坏 GLCD。特别是，电位计地端要接地，中间引脚连到 LCD 的对比度调整引脚上，另外一个引脚连到 VEE (LCD 电压)。150 Ω 电阻是用来限制流入 LCD 背光灯中的电流的。你需要估算它的阻值以给出合适的亮度，并确保没有超过 LCD 说明书中的电压值。变阻器用来调整显示器的对比度，使你能够很容易地看清楚显示器。

一旦你确信每件东西都连接正确，给 Arduino 上电后，你应该能看到 GLCD 亮起来，准备接收数据。现在输入代码。

输入代码

在输入代码之前，需要下载并安装 glcd.h 库。这个库是 Michael Margolis 写的，目前已发展到第三个版本。它是一个大文件，显示连接不同的 GLCD 的方法和对库中的全部命令的完整说明。查找 LCDs 下的 Arduino Playground 可以找到它。如果已经下载了库，将文件

解压。并且把整个文件夹放在 Aruidno 安装目录下的“libraries”文件夹里。下次启动 Arduino IDE 时它将加载并备用。

输入清单 11-2 中的代码。

清单 11-2 项目 32 的代码

```

/*
SCP1000          Mega
DRDY             N/A
CSB              53 via Logic Level Convertor
MISO             50 (straight through)
MOSI             51 via Logic Level Convertor
SCK              52 via Logic Level Convertor
3.3v             3.3v
GND              GND
TRIG             GND
PD              GND
*/

#include <glcd.h>

#include "fonts/allFonts.h"

//SPI 引脚
#define SLAVESELECT 53
#define SPICLOCK 52
#define DATAOUT 51      //主器件数据输出，从器件数据输入
#define DATAIN 50       //主器件数据输入，从器件数据输出
#define UBLB(a,b) ( ( (a) << 8) | (b) )
#define UBLB19(a,b) ( ( (a) << 16 ) | (b) )

//地址
#define PRESSURE 0x1F    //压力数据的高三位
#define PRESSURE_LSB 0x20 //压力数据的低 16 位
#define TEMP 0x21        //16 位温度数据
#define INTERVAL 900     //用秒表示的间隔时间（大概的）

int dots[124], dotCursor = 0, counter = 0;
char rev_in_byte;
int temp_in;
float hPa;

```

```

unsigned long pressure_lsb;
unsigned long pressure_msb;
unsigned long temp_pressure;
unsigned long pressure;

void setup()
{
    GLCD.Init(); //初始化 GLCD 设备
    GLCD.ClearScreen();
    GLCD.SelectFont(System5x7, BLACK); //加载字体

    byte clr;
    pinMode(DATAOUT, OUTPUT);
    pinMode(DATAIN, INPUT);
    pinMode(SPICLOCK, OUTPUT);
    pinMode(SLAVESELECT, OUTPUT);
    digitalWrite(SLAVESELECT, HIGH); //非使能从元件
    SPCR = B01010011; //SPI 控制寄存器
    //MPIE=0, SPE=1 (on), DORD=0 (MSB first), MSTR=1 (master), CPOL=0 (clock idle
when low), CPHA=0 (samples MOSI on rising edge), SPR1=1 & SPR0=1 (250kHz)
    clr=SPSR; //SPI 状态寄存器
    clr=SPDR; // SPI 数据寄存器
    delay(10);

    write_register(0x03, 0x09); //高速读模式
    write_register(0x03, 0x0A); //高分辨率模式

    GLCD.DrawRect(1, 1, 125, 44); //画一个矩形
    for (int x=0; x<46; x+=11) { //画水平分度
        GLCD.SetDot(0, 1+x, BLACK);
        GLCD.SetDot(127, 1+x, BLACK);
    }
    for (int x=0; x<128; x+=5) { //画垂直分度
        GLCD.SetDot(1+x, 0, BLACK);
    }

    for (int x; x<124; x++) {dots[x]=1023;} //清除数组
    getPressure();
    drawPoints(dotCursor);
}

void loop()

```

```

{
    getPressure();

    GLCD.CursorToXY(0, 49); //打印压力值
    GLCD.print("hPa:");
    GLCD.CursorToXY(24, 49);
    GLCD.print(hPa);

    temp_in = read_register16(TEMP);
    float tempC = float(temp_in)/20.0;
    float tempF = (tempC*1.8) + 32;

    GLCD.CursorToXY(0, 57); //打印温度值
    GLCD.print("Temp:");
    GLCD.CursorToXY(28, 57);
    GLCD.print(tempC); //打印温度值

    delay(1000);

    GLCD.CursorToXY(84, 49); //打印曲线
    GLCD.print("TREND:");
    GLCD.CursorToXY(84, 57);
    printTrend();

    counter++;
    if (counter==INTERVAL) {drawPoints(dotCursor);}
}

void drawPoints(int position) {
    counter=0;
    dots[dotCursor] = int(hPa);
    GLCD.FillRect(2, 2, 123, 40, WHITE); //清除图形区
    for (int x=0; x<124; x++) {
        GLCD.SetDot(125-x, 44-((dots[position]-980)), BLACK);
        position--;
        if (position<0) {position=123;}
    }
    dotCursor++;
    if (dotCursor>123) {dotCursor=0;}
}

```

```

void getPressure() {
    pressure_msb = read_register(PRESSURE);
    pressure_msb &= B00000111;
    pressure_lsb = read_register16(PRESSURE_LSB);
    pressure_lsb &= 0x0000FFFF;
    pressure = UBLB19(pressure_msb, pressure_lsb);
    pressure /= 4;
    hPa = float(pressure)/100;
}

void printTrend() { //计算自最后一次数据更新后的曲线
    int dotCursor2=dotCursor-1;
    if (dotCursor2<0) {dotCursor2=123;}
    int val1=dots[dotCursor2];
    int dotCursor3=dotCursor2-1;
    if (dotCursor3<0) {dotCursor3=123;}
    int val2=dots[dotCursor3];
    if (val1>val2) {GLCD.print("RISING ");}
    if (val1==val2) {GLCD.print("STEADY ");}
    if (val1<val2) {GLCD.print("FALLING");}
}

char spi_transfer(char data)
{
    SPDR = data;                                //开始传送数据
    while (!(SPSR & (1<<SPIF)))                //等待传送结束
    {
    };
    return SPDR;    //返回数值
}

char read_register(char register_name)
{
    char in_byte;
    register_name <= 2;

    digitalWrite(SLAVESELECT, LOW); //使能 SPI 元件
    spi_transfer(register_name); //写字节到元件
    in_byte = spi_transfer(0x00); //获得寄存器返回值
    digitalWrite(SLAVESELECT, HIGH); //非使能 SPI 元件
    delay(10);
    return(in_byte); //返回值
}

```

```

}

unsigned long read_register16(char register_name)
{
    byte in_byte1;
    byte in_byte2;
    float in_word;

    register_name <=& 2;

    digitalWrite(SLAVESELECT, LOW); //使能 SPI 元件
    spi_transfer(register_name); //写数据到元件
    in_byte1 = spi_transfer(0x00);
    in_byte2 = spi_transfer(0x00);
    digitalWrite(SLAVESELECT, HIGH); //非使能 SPI 元件
    in_word = UBLB(in_byte1, in_byte2);
    return(in_word); //返回值
}

void write_register(char register_name, char register_value)
{
    register_name <=& 2;
    register_name |= B00000010; //写命令

    digitalWrite(SLAVESELECT, LOW); //选择 SPI 元件
    spi_transfer(register_name); //送寄存器地址
    spi_transfer(register_value); //送记录中的值到寄存器
    digitalWrite(SLAVESELECT, HIGH);
}

```

当运行代码时，你将看到一条随时间变化的压力曲线，以及当前压力和温度值。开始时不是很激动人心，它需要满 24 小时才能显示压力的变化。每点代表 15 分钟，水平轴的刻度单位是小时。如果你想使曲线变化快一点，改变 INTERVAL 的定义使值变小一点即可。

代码回顾

项目 32 中的许多代码与项目 31 相同。所不同的是，这里需要控制新的元件 GLCD，并且送数据到串口监视器部分的代码去掉了，因此我将略过与项目 31 重复代码的讲解，将重点放在增加的部分。

首先，需要包含 glcd.h 库及所要用到的字体到代码中。


```
#include <glcd.h>
#include "fonts/allFonts.h"
```

在定义部分，定义名为 INTERVAL 的时间间隔，以秒为单位，用于存储和显示图上两个数据点之间的间隔。间隔时间是 900 秒，也就是两个相邻的时间点之间间隔 15 分钟。

```
#define INTERVAL 900 //用秒表示的间隔时间
```

声明 3 个新的整型变量并初始化。一个是 dot[] 数组，它用来存储在不同的时间读取到的压力测量值。下一个是 dotCursor 变量，它存储正在使用的数组的索引。在每一次主循环重复时 Counter 变量都要递增，它用来确定自最后一个数据存储之后有多少秒过去了（大概的）。

```
int dots[124], dotCursor = 0, counter = 0;
```

在 setup 函数中，首先初始化 GLCD 设备：

```
GLCD.Init();
```

控制 GLCD 的命令在初始化之后，就像下面的清空显示器命令：

```
GLCD.ClearScreen();
```

之后，选择使用哪一种字体及它是要显示成阴文还是阳文：

```
GLCD.SelectFont(System5x7, BLACK); //加载字体
```

这里有许多不同形式和大小的字体可用。而且，这个库包含了一个非常棒的功能，提供了一个叫做 FontCreator2 的免费软件，它可以在你的项目中生成字体头文件，这个程序可以转换 PC 的字体供库使用。

之后，显示要画图的空间，这使用 DrawRect 命令：

```
GLCD.DrawRect(1,1,125,44);
```

采用 128×64 点显示的坐标系，128 点宽，64 点高，两个轴的 0 点都在左上角。X 轴的刻度从 0~127，Y 轴的刻度相应地从 0 到 63。DrawRect 函数的作用是画一个矩形。函数的前两个参数表示矩形起点的 X 轴和 Y 轴坐标，也就是矩形的左上角，后面两个参数是沿着

X 轴和 Y 轴的高和宽的延伸。程序中的矩形从点 (1, 1) 开始画, 125 像素点宽, 44 像素点高。

你也可以用 `FillRect()` 命令生成一个实心矩形, 使用

```
GLCD.FillRect(1,1,125,44,BLACK);
```

产生一个实心、黑色、同样尺寸的矩形。下一步你需要产生垂直和水平刻度。使用 `for` 循环, 从 0 像素点处和 127 像素点处每隔 11 个像素点生成一个点:

```
for (int x=0; x<46; x+=11) { //画垂直分度
    GLCD.SetDot(0,1+x, BLACK);
    GLCD.SetDot(127,1+x, BLACK);
}
```

以上代码所用的 `SetDot` 命令以黑色或白色在 X 轴和 Y 坐标上简单地放置一个像素点。白色像素点只是擦除任何已经显示的黑色像素点。

之后, 从像素点 1 延水平轴 (小时) 向右侧画, 每隔 5 个像素点画一个:

```
for (int x=0; x<128; x+=5) { //画水平刻度
    GLCD.SetDot(1+x,0, BLACK);
}
```

然后, 用值 1023 初始化压力的测量数组, 这是因为这个图像将显示从开始存储的所有点。因为你不想显示全部的点, 而只显示那些从 Arduino 上电起所存储的点。值 1023 保证这些测量值与矩形的顶线重合, 因此可能隐藏起来而不被看到。我将在 `drawPoints()` 函数中进一步解释。

```
for (int x; x<124; x++) {dots[x]=1023;}
```

接下来, 调用 `getPressure()` 函数。这里只是把项目 31 中主循环代码移到 `getPressure()` 函数中, 因为在代码中它要被调用几次。该项目中去掉了项目 31 中送数据到串口监视器部分的代码, 因为这些数据实际已经显示在 GLCD 上了。

```
getPressure();
```

现在调用 `drawPoints()` 函数:

```
drawPoints(dotCursor);
```

这个函数的作用是在图形上画点。需要简单地解释一下。在运行主循环之前，已经获得压力数据并画出图形，因此在 `dots[]` 数组第一个索引中已经有一个测量值了。确保 `printTrend()` 函数工作正常（以后再解释）是非常重要的。

在主程序循环中，程序获得当前压力值：

```
getPressure();
```

之后，在显示器上打印压力值。为了打印字符，光标首先需要移动位置。通过在 `CursorToXY(x,y)` 函数中设置 *X* 坐标和 *Y* 坐标实现光标位置移动，之后打印字符。这样做两次，打印出“hPa:”字符，之后显示压力值。

```
GLCD.CursorToXY(0, 49);
GLCD.print("hPa:");
GLCD.CursorToXY(24, 49);
GLCD.print(hPa);
```

采用同样的方式，在压力值的下面打印温度值：

```
GLCD.CursorToXY(0, 57);
GLCD.print("Temp:");
GLCD.CursorToXY(28, 57);
GLCD.print(tempC);
```

如果你想显示华氏度而不是摄氏度，需要把摄氏单位转化成华氏单位。

之后，要延时 1000 毫秒，这意味着测量到的压力大约每秒钟显示一次：

```
delay(1000);
```

之后，打印压力曲线，通过调用 `PrintTrend()` 函数获得曲线值（稍后再解释）：

```
GLCD.CursorToXY(84, 49);
GLCD.print("TREND:");
GLCD.CursorToXY(84, 57);
printTrend();
```

你要存储每个间隔时间内所测到的压力值，因此 1000 毫秒延时之后，计数器加 1。

```
counter++;
```

之后，检查计数器的值是否已经达到 INTERVAL 所设定的值。如果达到，调用 drawPoints()函数：

```
if (counter==INTERVAL) {drawPoints(dotCursor);}
```

之后，要增加两个新的函数去画图，并且打印出当前的压力曲线。第一个是 drawPoints()函数，将 dotCursor 的值作为一个参数传递给该函数：

```
void drawPoints(int position) {
```

计数器的值已经到了 INTERVAL 所设定的值，因此现在将它置零：

```
counter=0;
```

读到的压力值存储在 dots[]数组的当前位置。因为低像素显示就能满足使用，所以你不需小数点后的任何数字，因此只取 hPa 的整数部分，这样可以节省存储空间，因为一个整型数组比一个浮点型数组占用的空间小。

```
dots[dotCursor] = int(hPa);
```

现在你需要给新数据清除图形区。使用 FillRect 命令实现，在矩形边框内生成一个白色矩形：

```
GLCD.FillRect(2, 2, 123, 40, WHITE); //清除图形区
```

用一个 for 循环遍历数组的 124 个元素：

```
for (int x=0; x<124; x++) {
```

之后使用 SetDot()命令将一个点放置在图像中合适的位置上：

```
GLCD.SetDot(125-x, 44-((dots[position]-980)), BLACK);
```

你想要从右向左画图，所以最近读到的数显示在右侧，图形向左卷动，因此在 X 轴坐标 125-x 处开始画第一个点。当 x 值增加时，该点将向左移动。Y 轴的坐标由当前压力值（以 hPa 为单位）减去 980 确定。图形的纵坐标有 40 像素高，压力范围从 980~1020hPa。

这是许多地方典型的气压值。如果你所在地区的大气压比这个高或低，你可以相应地调整 980 的值。然后用 44 减去所得到的数值，将数组中读出的压力值转换成所需要的 Y 轴坐标。

变量 `position` 的初始值设置为 `dotCursor` 的当前值。接下来 `position` 的值递减：

```
position--;
```

当它递减到 0 时，重新设置为 123：

```
if (position<0) {position=123;}
```

当 124 个点画完时，`dotCursor` 值递增，准备在数组中存储下一个压力测量值：

```
dotCursor++;
```

之后，在它的值大于 123 时（数组的最大元素），把它设回 0：

```
if (dotCursor>123) {dotCursor=0;}
```

下一个新的函数是 `printTrend()` 函数，这个函数的作用只是发现当前测得的压力值与上个值相比是高、低还是相同，随后相应地显示 `RISING`、`STEADY` 和 `FALLING`。

先将 `dotCursor` 最后的值存储在 `dotCursor2` 中，然后将该值减去 1，因为当测量值存储到 `drawPoints()` 函数中之后，`dotCursor` 的值加 1。

```
int dotCursor2=dotCursor-1;
```

检查 `dotCursor2` 的值是否小于 0，如果是，将它设置为 123：

```
if (dotCursor2<0) {dotCursor2=123;}
```

`dotCursor2` 存储当前测量值在数组中位置的索引号。现在声明一个叫做 `val1` 的整型变量，存储最后测量到的压力值：

```
int val1=dots[dotCursor2];
```

接下来需要获得倒数第二位测量值，因此创建另外一个变量 `dotCursor3`，存储数组中

倒数第二个位置的索引号：

```
int dotCursor3=dotCursor2-1;
if (dotCursor3<0) {dotCursor3=123;}
int val2=dots[dotCursor3];
```

现在利用 val1 存储最后一次测量值，val2 存储倒数第二次的测量值。剩下的工作是判断与前一次读到的值相比，最后读到的压力值是高、低还是相同，然后打印相应的曲线。

```
if (val1>val2) {GLCD.print("RISING ");}
if (val1==val2) {GLCD.print("STEADY ");}
if (val1<val2) {GLCD.print("FALLING");}
```

剩下的代码与项目 31 相同。当运行这个程序时，你将得到与图 11-6 相同的曲线。如果在 24 个小时内大气压力发生剧烈变化，你将看到变化幅度较大的曲线。

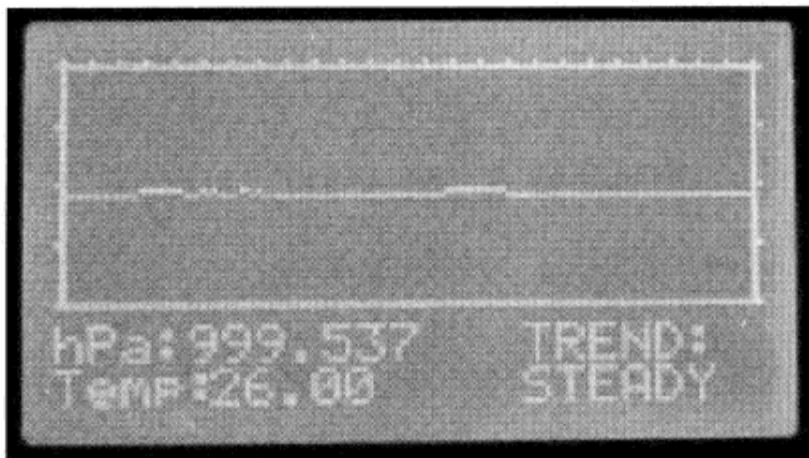


图 11-6 项目 32——数字气压计的显示

项目 32 的主要目的是提供一个使用 SCP1000 传感器的例子，以及如何使用 GLCD 显示器。本项目所采用的 GLCD 命令只是教你尝试使用 GLCD。你可以显示矩形、圆和线，设置点，甚至画一个 BMP 图像。屏幕可以分成字符区域和图形区域，你可以独立使用和打印它们。该库里有 Michael Margolis 所写的非常好的文档，这使得你可以非常容易地使用该库。仔细阅读文档，它将帮助你做很多事情。该库里也有一整套例子程序，包括 Conway 生命游戏和非常漂亮的火箭游戏。

小结

第 11 章介绍了如何使用数字压力传感器及如何使用串行外设接口与它进行通信，还介

绍了 SPI 的基本概念和它的工作原理。现在你大体上知道了 SPI 是如何工作的，可以使用 Arduino IDE 0019 版本的 SPI 库的帮助文件获得更多知识。当你通过 SPI 与另一个设备通信时，它将为你完成最困难的那部分工作。你已经学习了如何使用 SPI 从小型 SCP1000 压力传感器中读取数据。如果你想要做自己的气象站，这种便宜的元件是个理想的选择。我使用这种传感器做过一个大气压海拔球项目，因为它小巧、精确并且容易使用。它也可以精确地测量低于说明书给定范围的压力，因此它是 HAB（大气压海拔球）项目理想的测量元件。

你已经了解了如何将一个图形 LCD 连接到 Arduino 上，以及当使用 GLCD 库时如何使用它显示字符和基本图形。通过详细阅读帮助文档，你可以学着制作很酷的东西，如显示 BMP 图像或制作自己的游戏。Arduino 加一个 GLCD 可以很容易地封装在一个小盒子里，用来制作你自己的游戏手柄。

在下一章中，你将学习如何使用触摸屏。

本章的主题和概念

- 如何连接 SCP1000 压力传感器到一个 Arduino
- 在一串数字中如何使用 `#define` 生成位操作
- 如何连接两个短比特长度的数字生成一个长比特数字
- SPI 接口是如何工作的
- SPI 元件可以单独控制或菊花链连接元件
- SPI 元件的主、从设备
- 如何通过时钟脉冲控制 SPI 设备的数据输入和输出
- SPI 三个总线寄存器的作用和使用方法
- 把压力值从以帕斯卡为单位转换到以百帕和大气压为单位
- 使用位操作检查位是否已经置位
- 如何连接图形 LCD 到 Arduino
- 使用 `glcd.h` 库中的基本命令画线、点和打印字符

触 摸 屏

本章我们学习如何使用触摸屏，使用 Arduino 控制触摸屏是非常简单的。自从出现手机和掌上游戏机，触摸屏就变得不再那么贵了并且很容易买到。触摸屏使你非常容易给设备制作一个触摸界面或者把它覆盖在一个 LCD 上得到一个触摸界面。一些公司（如 Sparkfun）出品了一些连接器元件，这些连接器使得很容易实现与触摸屏互动。连接元件一般是小的连接插口或非标连接器，它把一些接线分开使接口更友好，比如连接器可能有一些插针可以直接插入面包板中。在这个项目中，你要使用的是已封装好的 Nintendo DS 触摸屏和 Sparkfun 公司的接口元件。在正式使用它之前，我们从一个简单的演示工程开始，演示如何从触摸屏获得读数。

项目 33——基本的触摸屏

这个项目需要一个 Nintendo DS 触摸屏和一个接口模块。后者是必需的，因为从触摸屏输出的是非常细小易断的连接线，如果没有附加元件，它不可能与 Arduino 连接。

需要的元件

Nintendo DS 触摸屏价格便宜，可以在许多供应商处得到。XL 版大概是标准屏的两倍大小，推荐使用这个元件，如果你能找到一块。连接器元件可以从 Sparkfun 公司或它的分销商处获得。

Nintendo DS 触摸屏



连接器



把元件连接起来

参照图 12-1 连接每一个元件。

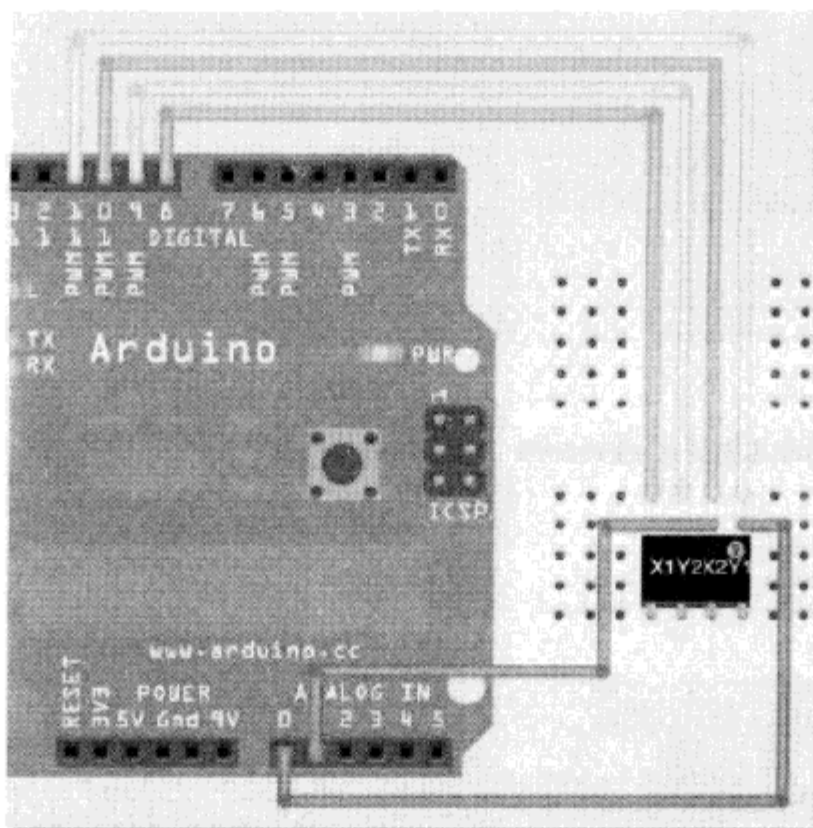


图 12-1 项目 33——基本的触摸屏电路图（参见彩色版插图）

连接器有标志为 X1、Y2、X2 和 Y1 的引脚，按照表 12-1 连接这些引脚。

表 12-1 项目 33 的连接引脚

Arduino	连接器
Digital Pin 8	X1
Digital Pin 9	Y2
Digital Pin 10	X2
Digital Pin 11	Y1
Analog Pin 0	Y1
Analog Pin 1	X2

需要焊接一些引脚到连接器元件上。在焊接这些引脚时要注意，Sparkfun 的标志要朝上。屏通过小的连接扣连接到连接器元件上。把按钮按回，将带状电缆推到连接器内，之后合上锁扣将它固定在适当的位置上。连接好后，软带电路板位于屏右上部。从现在开始

对待这个元件要十分小心，它非常容易打碎。在试验中，我打碎了三个触摸屏和两个连接器。你最好能找到一种将面包板、连接器和触摸屏固定在一起的办法，防止它们互相移动。

输入代码

输入清单 12-1 的代码。

清单 12-1 项目 33 的代码

```
//项目 33

//电源连接
#define Left 8      //左端(X1) 连到数字引脚 8
#define Bottom 9    //底部(Y2) 连到数字引脚 9
#define Right 10    //右侧(X2) 连到数字引脚 10
#define Top 11      //顶部(Y1) 连到数字引脚 11

//模拟量连接
#define topInput 0   //顶部(Y1) 连到模拟引脚 0
#define rightInput 1 //右侧(X2) 连到模拟引脚 1

int coordX = 0, coordY = 0;

void setup()
{
    Serial.begin(38400);
}

void loop()
{
    if (touch()) //如果屏被按下，打印坐标
    {
        Serial.print(coordX);
        Serial.print(" ");
        Serial.println(coordY);
        delay(250);
    }
}

//如果有触摸动作返回 TRUE 并设置坐标值给 touchX 和 touchY
boolean touch()
```

```

{
    boolean touch = false;

    //获得水平坐标
    pinMode(Left, OUTPUT);
    digitalWrite(Left, LOW); //将左引脚连接地

    pinMode(Right, OUTPUT); //将右引脚连接+5V
    digitalWrite(Right, HIGH);

    pinMode(Top, INPUT); //顶部和底部数字引脚设置为高阻抗
    pinMode(Bottom, INPUT);

    delay(3);
    coordX = analogRead(topInput);

    //获得垂直坐标
    pinMode(Bottom, OUTPUT); //设置底部引脚为地
    digitalWrite(Bottom, LOW);

    pinMode(Top, OUTPUT); //设置顶部为引脚+5V
    digitalWrite(Top, HIGH);

    pinMode(Right, INPUT); //左端右端引脚设为高阻抗
    pinMode(Left, INPUT);

    delay(3);
    coordY = analogRead(rightInput);

    //如果坐标值小于1000 大于0, 那么屏被触摸
    if(coordX < 1000 && coordX > 0 && coordY < 1000 && coordY > 0) {touch
= true;}

    return touch;
}

```

输入这些代码并且上载到 **Arduino** 中。一旦程序运行, 打开串口监视器, 之后接触触摸屏。不管什么时候接触触摸屏, 手指的坐标都将显示在串口监视器上。 X 坐标从左到右水平穿过平面, Y 坐标是垂直方向从上到下。

在看这个代码之前, 知道触摸屏的工作原理是有帮助的。因此在讲解验证软件之前, 先回顾一下硬件。

硬件回顾

你现在使用的触摸屏来自 Nintendo DS，叫做电阻触摸屏。它是由不同的层面板组成的相对简单的结构。屏的底层由玻璃制成，表面镀上了透明的金属氧化物材料。这使得镀层的一部分导电，另一部分不导电。电压通过镀膜会产生一个梯度。在这层坚硬的玻璃层顶部是一个覆盖了导电镀膜的柔软层。这两层被非常薄的带分开，薄带是由网格状的绝缘点组成的。这些点使触摸屏在不被触摸时保持两层不接触。

如果仔细检查触摸屏，你会发现带状电缆上有 4 个连接点，这些连接点通向屏边缘的 4 条金属带。如果揭开屏看，你会发现有两个金属触点在屏的顶和底部，另外两个触点分别在第二层的左右两侧。

当一个手指或硬物按在柔性层上时，这一层被压弯，碰到硬层，关闭电流产生一个开关（见图 12-2）。

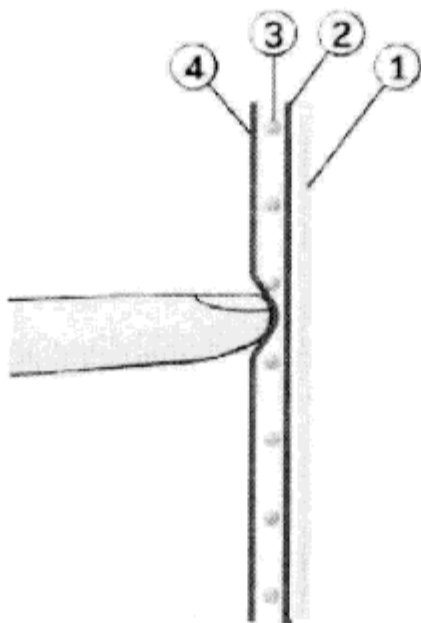


图 12-2 触摸屏是如何工作的（感谢 Wikimedia Commons 的 Mercury13）。

① 硬质层 ② 金属氧化物层 ③ 绝缘点 ④ 金属氧化物柔性胶片层

为了算出触摸点的坐标，首先在一个层上加载电压，产生从左到右的电压梯度。电路的一端接地，而另一端接在 5V 电源上。之后，使用模拟量输入去测量相对层上的金属带的电压。当触摸点的位置靠近 5V 那侧时，将测到接近 5V 的电压。同样，当触摸点接近地的一侧时，测得的电压接近于 0V。

之后，在另外一层加载电压，从未加电压层读电压值。这种交换通电层和读值层的速度非常快，每秒几百次。所以首先读 X 轴，之后读 Y 轴。我们通过这种方法获得触摸点的

X 轴和 Y 轴坐标。如果同时触碰屏上的两个点，你将获得两个触摸点中部某个点的坐标。

触摸屏也可以采用其他的技术，但是电阻式的制作便宜，并且它容易实现与 Arduino 的连接口，而不需要其他电路就能正常工作。现在你知道了触摸屏是如何工作的，让我们看一下代码，看看如何测量电压和获得坐标。

代码回顾

读触摸屏的代码实际上非常简单，首先定义 4 个数字引脚和两个模拟引脚。数字引脚用来给各层提供电压，模拟引脚用来测量电压：

```
//连接电源
#define Left 8      //左端(X1)连到数字引脚 8
#define Bottom 9    //底部(Y2)连到数字引脚 9
#define Right 10    //右侧(X2)连到数字引脚 10
#define Top 11      //顶部(Y1)连到数字引脚 11

//模拟量连接
#define topInput 0   //顶部(Y1)连到模拟引脚 0
#define rightInput 1 //右侧(X2)连到模拟引脚 1
```

之后声明并初始化整型数 X 和 Y ，它们用来存储坐标值。它们都初始化为 0：

```
int coordX = 0, coordY = 0;
```

因为要使用串口监视器读取坐标值，在 `setup` 函数中需要做的工作是开始串行通信，并设置波特率。在这个例子中，所使用的波特率是 38400：

```
Serial.begin(38400);
```

主程序循环只包含一个 `if` 语句，判断屏是否被按下：

```
if (touch()) //如果屏被按下，打印坐标
```

`Touch()`函数后面会解释。如果屏已经被按下，使用 `Serial.print` 命令在串口监视器上打印出 X 和 Y 的坐标，中间用空格隔开：

```
Serial.print(coordX);
Serial.print(" ");
```

```
Serial.println(coordY);
```

打印完坐标之后，等 1/4 秒。如果将手指一直按在屏幕上，坐标可以连续读出来：

```
delay(250);
```

下面的函数用于处理所有复杂工作。这个函数将返回一个布尔值，**true** 或 **false**，因此它的数据类型是 **boolean**，你不用给这个函数传递任何参数，因此参数表是空的。

```
boolean touch ()
```

声明一个布尔变量，并且初始化为 **false**。根据是否触摸到屏幕决定其值是 **true** 还是 **false**：

```
boolean touch = false;
```

之后，从左-右层设置一个电压，使用第二层上面的输入引脚读取电压值。为做这些，设置左、右引脚为输出，然后设置左引脚为低电平，使其成为地，右引脚为高电平，因此将它连到 5V 电压上：

```
pinMode(Left, OUTPUT);  
digitalWrite(Left, LOW); //将左引脚连接地
```

```
pinMode(Right, OUTPUT); //将右引脚连接+5V  
digitalWrite(Right, HIGH);
```

将顶部和底部的数字引脚设置为 **INPUT**，因此它们变成高阻抗：

```
pinMode(Top, INPUT);  
pinMode(Bottom, INPUT);
```

高阻抗意味着这个引脚不会被电流驱动，因此不会浮动，如它们既不是高电平也不是低电平。你不希望这些引脚有电压加载或者成为地，所以阻抗状态是完美的，因为你想要使用这些引脚中的任意一个读取模拟电压值。

之后，延时一小段时间，允许上面的状态发生改变之后，从上端输入引脚读取模拟值。这个值存储在 **coordX** 中，给出 *X* 坐标值。

```
delay(3);
coordX = analogRead(topInput);
```

你已经获得了 X 坐标值。接下来做几乎完全相同的事情，只不过这次你通过顶部-底部层设置电压，然后使用在相对的层上的 `rightInput` 引脚读取电压值：

```
pinMode(Bottom, OUTPUT); //将底层引脚连接地
digitalWrite(Bottom, LOW);

pinMode(Top, OUTPUT); //将顶层连接到+5V
digitalWrite(Top, HIGH);

pinMode(Right, INPUT); //左右输入模式
pinMode(Left, INPUT);

delay(3);
coordY = analogRead(rightInput);
```

如果读值大于 0 并且小于 1000，设置 Boolean 变量 `touch` 为 `true`。这保证如果读到的值在可接受范围内，只返回一个 `true` 值。

```
if(coordX < 1000 && coordX > 0 && coordY < 1000 && coordY > 0) {touch = true;}
```

你会发现，读值的范围在 100 到 900 之间变动。最后返回 `touch` 的值，如果屏没有被按下，返回值是 `false`；如果被按下，返回值将是 `true`：

```
return touch;
```

正如你所看到的，从触摸屏读值是非常简单的，并且有各种各样的应用。你可以在屏的下面放一个图片或者图表，并与按钮或其他的控制元件相关联，或者将屏覆盖在一个 LCD 显示器上，如 Nintendo DS 触摸屏，它允许你根据需要改变屏下的用户接口。

下面看一个简单的示例，打印出放在触摸屏下面的按键，并且读出被按下键的值。

项目 34——触摸屏键盘

在触摸屏下放置一个用户接口。用户接口是一个打印键盘，然后根据触摸位置检测哪一个键被按下。一旦你明白这种方法的原理，就可以进一步用 LCD 或 OLCD 显示器上的

图形来代替打印键盘。

你将在 LCD 显示器上输出所按下的键。下面先看一下本项目所用的元件列表。

需要的元件

与项目 33 需要完全相同的元件和电路，只是增加了一个 16×2 LCD 显示器。

Nintendo DS 触摸屏



触摸屏接口



16×2 LCD 显示器



另外一个不同之处是你要制作和打印一个键盘放置在触摸屏下，标准的 DS 触摸屏大小是 70mm×55mm (2.75 英寸×2.16 英寸)，因此需要生成这个尺寸的一个模板，可以使用文字处理程序来制作键盘字块。之后把这些字块均匀地放置在一个矩形框中，用它组成一个电话键盘。图 12-3 显示了我制作的键盘，你可以随意使用它。

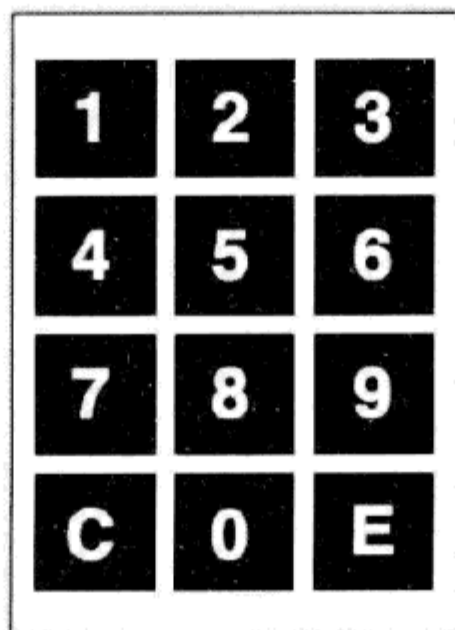


图 12-3 项目 34 的键盘图

把元件连接起来

按照图 12-4 连接每个元件。

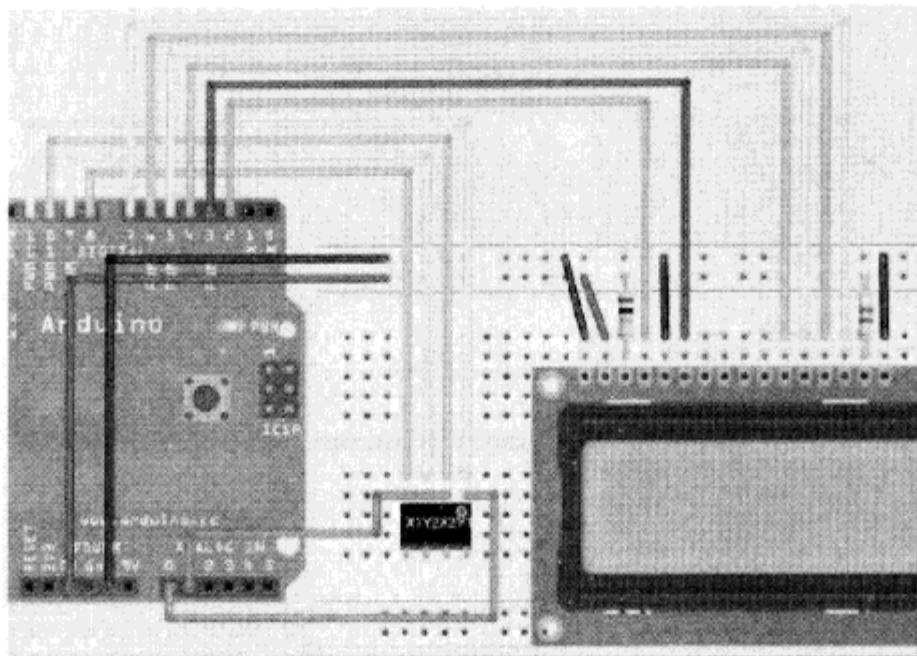


图 12-4 项目 34——触摸屏键盘电路图（参见彩色版插图）

参考表 12-2 的 LCD 引脚。

表 12-2 项目 34 的 LCD 输出引脚

Arduino 开发板	其它元件	液晶屏
数字引脚 2		Enable（使能）
数字引脚 2		RS（选择寄存器）
数字引脚 2		DB4（数字引脚 4）
数字引脚 2		DB5（数字引脚 5）
数字引脚 2		DB6（数字引脚 64）
数字引脚 2		DB7（数字引脚 7）
地		Vss（地）
地		R/W（读/写）
+5v		Vdd（电源）
	通过电阻连接+5v	Vo（对比度）
	通过电阻连接+5v	A/Vee（LED 电源）
	地	LED 的地

输入代码

输入清单 12-2 中的代码。

清单 12-2 项目 34 的代码

```
//项目 34
```

```

#include <LiquidCrystal.h>

LiquidCrystal lcd(2, 3, 4, 5, 6, 7); //生成 lcd 对象, 指定引脚

//电源连接
#define Left 8    //左侧(X1) 连到引脚 8
#define Bottom 9  //底部(Y2) 连接引脚 9
#define Right 10  //右侧(X2) 连接引脚 10
#define Top 11    //顶部(Y1) 连接引脚 11

//模拟量连接
#define topInput 0 //顶部(Y1) 连接引脚 0
#define rightInput 1 //右侧(X2) 连接引脚 1

int coordX = 0, coordY = 0;
char buffer[16];

void setup()
{
    lcd.begin(16, 2); //显示设置为 16 列 2 行
    lcd.clear();
}

void loop()
{
    if (touch())
    {
        if ((coordX>110 && coordX<300) && (coordY>170 && coordY<360)) {lcd.print("3");}
        if ((coordX>110 && coordX<300) && (coordY>410 && coordY<610)) {lcd.print("2");}
        if ((coordX>110 && coordX<300) && (coordY>640 && coordY<860)) {lcd.print("1");}
        if ((coordX>330 && coordX<470) && (coordY>170 && coordY<360)) {lcd.print("6");}
        if ((coordX>330 && coordX<470) && (coordY>410 && coordY<610)) {lcd.print("5");}
        if ((coordX>330 && coordX<470) && (coordY>640 && coordY<860)) {lcd.print("4");}
        if ((coordX>490 && coordX<710) && (coordY>170 && coordY<360)) {lcd.print("9");}
        if ((coordX>490 && coordX<710) && (coordY>410 && coordY<610)) {lcd.print("8");}
        if ((coordX>490 && coordX<710) && (coordY>640 && coordY<860)) {lcd.print("7");}
        if ((coordX>760 && coordX<940) && (coordY>170 && coordY<360)) {scrollLCD();}
        if ((coordX>760 && coordX<940) && (coordY>410 && coordY<610)) {lcd.print("0");}
        if ((coordX>760 && coordX<940) && (coordY>640 && coordY<860)) {lcd.clear();}
        delay(250);
    }
}

```



```

}

//如果有触摸动作返回 true 并设置坐标值给 touchX 和 touchY
boolean touch()
{
    boolean touch = false;

    //获得水平坐标
    pinMode(Left, OUTPUT);
    digitalWrite(Left, LOW); //将左引脚连接地

    pinMode(Right, OUTPUT); //将右引脚连接+5V
    digitalWrite(Right, HIGH);

    pinMode(Top, INPUT); //顶部和底部数字引脚设置为高阻抗
    pinMode(Bottom, INPUT);

    delay(3); //短暂延时
    coordX = analogRead(topInput);

    //获得垂直坐标
    pinMode(Bottom, OUTPUT); //设置底部引脚为地
    digitalWrite(Bottom, LOW);

    pinMode(Top, OUTPUT); //设置顶部为引脚+5V
    digitalWrite(Top, HIGH);

    pinMode(Right, INPUT); //左端右端引脚设为高阻抗
    pinMode(Left, INPUT);

    delay(3); //短暂延时
    coordY = analogRead(rightInput);

    //如果坐标值小于 1000 大于 0, 那么屏被触摸
    if(coordX < 1000 && coordX > 0 && coordY < 1000 && coordY > 0) {touch = true;}

    return touch;
}

void scrollLCD() {
    for (int scrollNum=0; scrollNum<16; scrollNum++) {
        lcd.scrollDisplayLeft();
    }
}

```

```

    delay(100);
}
lcd.clear();
}

```

输入代码并上载到你的 Arduino 中，把打印好的键盘轻轻地放到触摸屏下面，带状电缆在底部右侧（挨着 E 键）。你现在可以在触摸屏上按键了，你按下的键将显示在 LCD 上。当你按 C（清屏）键时，屏幕显示将清除。当你按下 E（输入）键时，显示的数字将向左滚动直到它们消失。

你已经知道 LCD 屏和触摸屏是如何工作的了，因此我将略过这个项目的硬件回顾，只看代码。

代码回顾

开始包含 LiquidCrystal 库，生成一个 lcd 对象：

```

#include <LiquidCrystal.h>

LiquidCrystal lcd(2, 3, 4, 5, 6, 7); //生成 lcd 对象，指定引脚

```

这次，使用引脚 2 和 3 连接 LCD 上的 RS 和使能引脚，使用引脚 4 到 7 连接数据线。之后，定义触摸屏的引脚，初始化 X 和 Y 变量：

```

//电源连接
#define Left 8    //左侧(X1)连到引脚 8
#define Bottom 9  //底部(Y2)连接引脚 9
#define Right 10  //右侧(X2) 连接引脚 10
#define Top 11    //顶部(Y1) 连接引脚 11

//模拟量连接
#define topInput 0 //顶部(Y1) 连接引脚 0
#define rightInput 1 //右侧(X2) 连接引脚 1
int coordX = 0, coordY = 0;

```

在 setup 函数中将 LCD 对象设置为 16 列 2 行，之后清空显示器，做好开始的准备：

```

lcd.begin(16, 2); //显示设置为 16 列 2 行
lcd.clear();

```

在主循环中，如同项目 33 一样，有一个 if 语句。但是这次你需要检查触碰的坐标在哪个键的边界矩形中。如果坐标在相应的键边界内，相应的数字被显示在 LCD 上。如果按下 C 键，清除显示器的显示。如果按下 E 键，调用 scrollLCD 函数。

```
if (touch())
{
    if ((coordX>110 && coordX<300) && (coordY>170 && coordY<360)) {lcd.print("3");}
    if ((coordX>110 && coordX<300) && (coordY>410 && coordY<610)) {lcd.print("2");}
    if ((coordX>110 && coordX<300) && (coordY>640 && coordY<860)) {lcd.print("1");}
    if ((coordX>330 && coordX<470) && (coordY>170 && coordY<360)) {lcd.print("6");}
    if ((coordX>330 && coordX<470) && (coordY>410 && coordY<610)) {lcd.print("5");}
    if ((coordX>330 && coordX<470) && (coordY>640 && coordY<860)) {lcd.print("4");}
    if ((coordX>490 && coordX<710) && (coordY>170 && coordY<360)) {lcd.print("9");}
    if ((coordX>490 && coordX<710) && (coordY>410 && coordY<610)) {lcd.print("8");}
    if ((coordX>490 && coordX<710) && (coordY>640 && coordY<860)) {lcd.print("7");}
    if ((coordX>760 && coordX<940) && (coordY>170 && coordY<360)) {scrollLCD();}
    if ((coordX>760 && coordX<940) && (coordY>410 && coordY<610)) {lcd.print("0");}
    if ((coordX>760 && coordX<940) && (coordY>640 && coordY<860)) {lcd.clear();}
    delay(250);
}
```

每个 if 语句是一套条件和逻辑操作，看一下按下键盘 3 时的语句：

```
if ((coordX>110 && coordX<300) && (coordY>170 && coordY<360)) {lcd.print("3");}
```

你会发现第一个逻辑与条件用来检查触碰的位置是否在以下位置之内：X 坐标在 110 和 300 之间（从左边开始），Y 坐标在 170 和 360 之间（从顶端开始），所有的条件必须与按下的键相对应，因此使用逻辑与操作符。

为了找到每个按钮的坐标，使用一个点状的笔尖轻轻地压在按钮的左和右侧来获得 X 坐标。之后在按钮的上和下部重复相同的动作获得 Y 坐标。如果你需要修改代码或使用自己的按钮布局，可利用项目 33 的代码将坐标打印到串口监视器上，采用以上方式确定每个按钮的精确坐标。

接下来是触摸函数，你已经知道它的工作原理。最后，是 scrollLCD 函数，它既不返回任何数据，也不需要传递任何参数，因此它的类型是 void：

```
void scrollLCD () {
```

之后是一个 for 函数，它循环 16 次，这是所能输入和显示的字符的最大长度。

```
For cint scrollNum=0;scrollNum<16;scrollNum++7{
```

在 for 循环内部，使用 LiquidCrystal 库中的 scrollDisplayLeft() 函数使字符向左滚动一位。接下来是延时 100 毫秒。

```
lcd.scrollDisplayLeft();  
delay(100);
```

一旦这个函数运行 16 次，输入的数字将全部滚动到左侧，显示它们已经进入系统的效果。当数据输入完成时，你可以编写代码做你想做的事情。

最后，清空屏幕。在退出这个函数返回到主循环时，使它为接收新的数字做好准备。

这个项目给你提供了一个理想的确定触摸屏范围的方法，因此你可以选每个按钮的大小。可以用图形 LCD 或 OLED 代替纸张，你可以在上面画按钮。这样做的好处是根据使用者的选择，可以画上不同的按钮或菜单。使用这个技术，你可以为项目创建一个非常棒的触摸屏人机界面。

接下来我们控制一个 RGB LED，通过滑动方式，而不是按压方式控制 LED 的颜色。

项目 35——触摸屏灯控制

在这个项目中，你使用滑动方式开关一个 RGB LED 灯，控制 LED 的颜色。

需要的元件

使用与项目 33 完全相同的元件和电路，只是增加一个 RGB LED。RGB LED 需要用共阴极类型的，这意味着有一个共同的引脚连接在地上（阴极），其他三个引脚独立地接到控制引脚，为红色、绿色和蓝色 LED 提供电压。

Nintendo DS 触摸屏



触摸屏插座



RGBLED



限流电阻



*其他需要的元件。

当然也需要一个如项目 34 中的键盘，这次它需要为颜色滑动条和一个 on/off 按钮提供足够空间，可以采用如图 12-5 所示的方式。

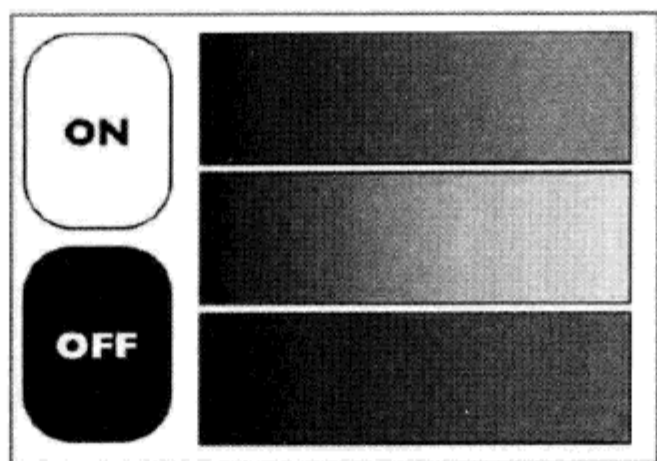


图 12-5 项目 35 的键盘图（参见彩色版插图）

把元件连接起来

像图 12-6 那样把每个元件连接起来。

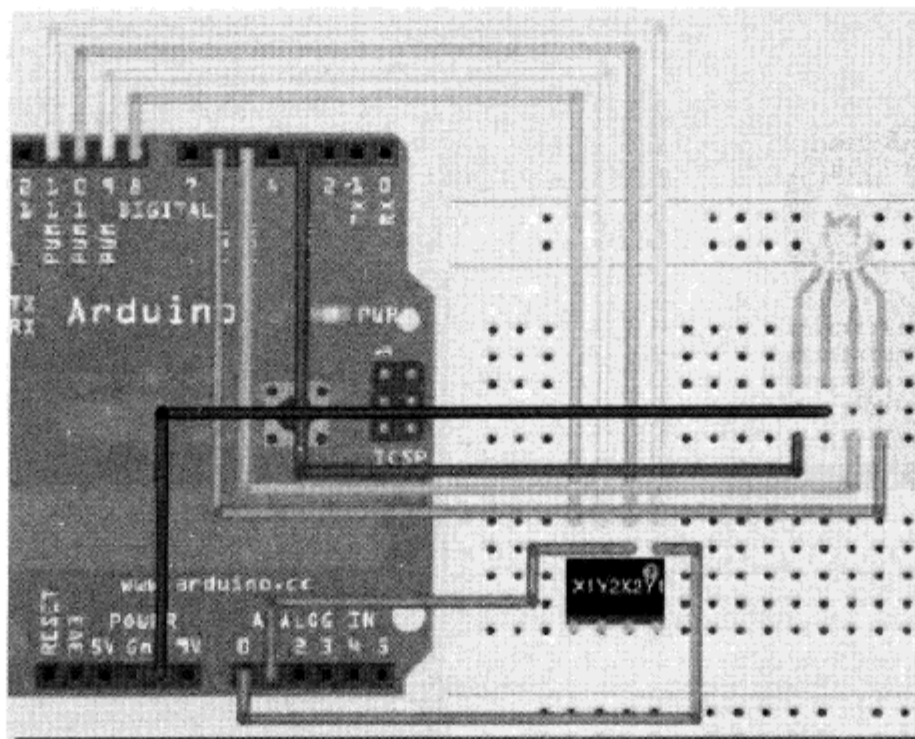


图 12-6 项目 35——触摸屏灯控制电路图（参见彩色版插图）

地线连到 LED 的共阴极引脚。PWM 引脚 3 连到红二极管, PWM 引脚 5 连到绿二极管, PWM 引脚 6 连到蓝二极管。如果需要, 在颜色引脚上设置限流电阻。我使用的是 Piranha RGB LED 的额定电压是 5V, 因此我认为它不需要限流电阻。然而, 这是不好的习惯, 使用 LED 不用限流电阻将降低它们的寿命。

输入代码

输入清单 12-3 中的代码。

清单 12-3 项目 35 的代码

```
//项目 35

//电源连接
#define Left 8      //左侧(X1)连到引脚 8
#define Bottom 9    //底部(Y2)连接引脚 9
#define Right 10    //右侧(X2)连接引脚 10
#define Top 11      //顶部(Y1)连接引脚 11

//模拟量连接
#define topInput 0   //顶部(Y1)连接引脚 0
#define rightInput 1 //右侧(X2)连接引脚 1

//RGB 引脚
#define pinR 3
#define pinG 5
#define pinB 6

int coordX = 0, coordY = 0;
boolean ledState = true;
int red = 100, green = 100, blue = 100;

void setup()
{
    pinMode(pinR, OUTPUT);
    pinMode(pinG, OUTPUT);
    pinMode(pinB, OUTPUT);
}

void loop()
```



```

{
    if (touch()) {
        if ((coordX>0 && coordX<270) && (coordY>0 && coordY<460)) {ledState =
true; delay(50);}
        if ((coordX>0 && coordX<270) && (coordY>510 && coordY< 880)) {ledState =
false; delay(50);}
        if ((coordX>380 && coordX<930) && (coordY>0 && coordY<300)) {red=
map(coordX, 380, 930, 0, 255);}
        if ((coordX>380 && coordX<930) && (coordY>350 && coordY<590))
{green=map(coordX, 380, 930, 0, 255);}
        if ((coordX>380 && coordX<930) && (coordY>640 && coordY<880))
{blue=map(coordX, 380, 930, 0, 255);}
        delay(10);
    }

    if (ledState) {
        analogWrite(pinR, red);
        analogWrite(pinG, green);
        analogWrite(pinB, blue);
    }
    else {
        analogWrite(pinR, 0);
        analogWrite(pinG, 0);
        analogWrite(pinB, 0);
    }
}

//如果有触摸动作返回 TRUE 并设置坐标值给 touchX 和 touchY
boolean touch()
{
    boolean touch = false;

    //获得水平坐标
    pinMode(Left, OUTPUT);
    digitalWrite(Left, LOW); //将左引脚连接地

    pinMode(Right, OUTPUT); //将右引脚连接+5V
    digitalWrite(Right, HIGH);

    pinMode(Top, INPUT); //顶部和底部数字引脚设置为高阻抗
    pinMode(Bottom, INPUT);

```

```

delay(3); //短暂延时
coordX = analogRead(topInput);

//获得垂直坐标
pinMode(Bottom, OUTPUT); //设置底部引脚为地
digitalWrite(Bottom, LOW);

pinMode(Top, OUTPUT); //设置顶部为引脚+5V
digitalWrite(Top, HIGH);

pinMode(Right, INPUT); //左端右端引脚设为高阻抗
pinMode(Left, INPUT);

delay(3); //短暂延时
coordY = analogRead(rightInput);

//如果坐标值小于 1000 大于 0, 那么屏被触摸
if(coordX < 1000 && coordX > 0 && coordY < 1000 && coordY > 0) {touch = true;}

    return touch;
}

```

代码回顾

代码的开始部分与项目 33 和项目 34 完全相同, 只是增加了 3 个 PWM 引脚的定义, 用来控制 RGB LED 内部的 R、G 和 B 元件:

```

//RGB 引脚
#define pinR 3
#define pinG 5
#define pinB 6

```

增加一个叫做 ledState 的布尔变量, 并将它初始化为 true。这个布尔变量将存储 LED 的状态, 例如 ture=开, false=关。

```
boolean ledState = true;
```

声明 3 个整型变量, 并用值 100 初始化每一个变量:

```
int red = 100, green = 100, blue = 100;
```

这三个整数将存储 LED 独立的颜色值，它们等于从引脚 3、5 和 6 输出的 PWM 值。

在 `setup` 函数中，将你所定义的 3 个 LED 引脚全部设置为输出：

```
pinMode(pinR, OUTPUT);
pinMode(pinG, OUTPUT);
pinMode(pinB, OUTPUT);
```

在主循环中，再次使用 `if` 语句来检查 `touch()` 函数返回的值是否为 `true`。只是这里用更多的 `if` 语句来确定触摸屏的哪个部分被按动。前两个定义 ON 和 OFF 按钮的界限范围。如果检查到触摸事件发生在 ON 按钮的界限范围内，设置 `ledState` 为 `true`。如果检查到触摸事件发生在 OFF 按钮的界限范围内，设置 `ledState` 为 `false`。之后延时一小段时间，防止从按钮中获得错误读数。

```
if ((coordX>0 && coordX<270) && (coordY>0 && coordY<460)) {ledState =
true; delay(50);}
if ((coordX>0 && coordX<270) && (coordY>510 && coordY< 880)) {ledState =
false; delay(50);}
```

之后检测控制红色、绿色和蓝色的滑条是否被触碰。如果检测到有碰触发生，那么 `red`、`green` 和 `blue` 中的值将发生相应的改变。

```
if ((coordX>380 && coordX<930) && (coordY>0 && coordY<300)) {red=map(coordX,
380, 930, 0, 255);}
if ((coordX>380 && coordX<930) && (coordY>350 && coordY<590))
{green=map(coordX, 380, 930, 0, 255);}
if ((coordX>380 && coordX<930) && (coordY>640 && coordY<880))
{blue=map(coordX, 380, 930, 0, 255);}
```

你使用 `map()` 函数完成这个功能，它有 5 个参数，第一个参数是所检查的值，接下来的参数表示值的高低范围（其他的忽略），最后两个参数表示所映射到的值的上限和下限。换句话说，你把在滑条范围内 `X` 的坐标映射为 0~255 之间的值，0 和 255 分别表示滑条的最左端和最右端。当你从左到右滑动手指时，可以使相应颜色的元件发生改变，从 0 变到 255 即从最暗变到最亮。

最后，用另一个 `if` 语句将 PWM 的 R、G 和 B 引脚值设置为 `red`、`green` 和 `blue` 的值，但是只有 `ledState` 为 `true` 时才这样做。如果 `ledState` 是 `false`，执行 `else` 语句，设置 PWM 的值为 0 或关闭状态。

```
if (ledState) {  
    analogWrite(pinR, red);  
    analogWrite(pinG, green);  
    analogWrite(pinB, blue);  
}  
else {  
    analogWrite(pinR, 0);  
    analogWrite(pinG, 0);  
    analogWrite(pinB, 0);  
}
```

剩下的程序是 `touch()` 函数，前面已经讲解，此处不再赘述。

小结

项目 35 介绍了触摸屏控制按钮和滑条的概念，再次使用 GLCD 或 OLED 显示器对控制灯系统进行控制。项目 35 比较简单，你可以对该项目进行拓展，用它来控制房子周围的 RGB 灯，把开关换成彩色 OLED 显示器和触摸屏以实现各种各样的光控制。

第 12 章说明电阻触摸屏非常容易实现与 Arduino 的连接，供我们使用。利用简短的程序，触摸屏和 Arduino 就可以为用户控制提供灵活的接口方法。与图形显示器配合使用，触摸屏将变成非常有用的系统控制工具。

本章的主题和概念

- 如何使用接口卡使得非标准连接变得更容易
- 电阻触摸屏是如何工作的
- 可获得 X 和 Y 坐标的正确的电源和电压测量循环方法
- 高阻抗的含义
- 将触摸屏盖在图形显示器上生成互动按钮
- 如何使用坐标和逻辑与操作定义一个按钮区域
- 如何在触摸屏上划分按钮或滑条区域

调阻计或微调变阻器只是一个小型的可调变阻器，用来调整或调试部分电路。调试完成之后，就不再动了。

把元件连接起来

像图 13-1 那样连接每个元件。

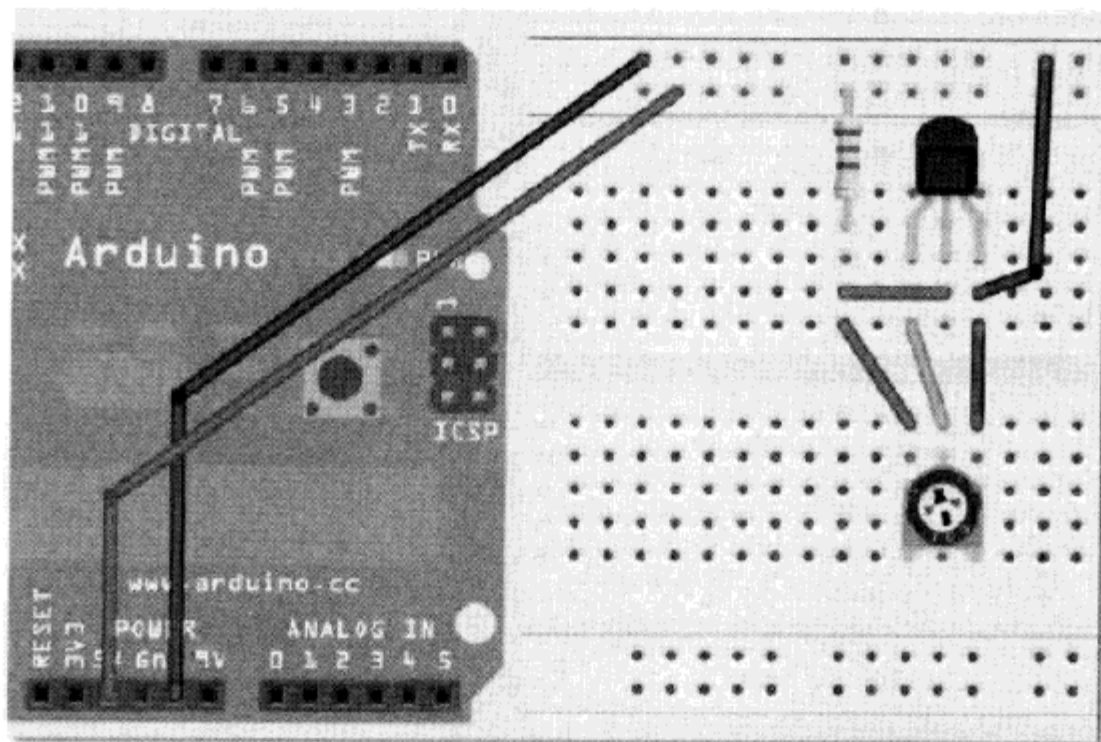
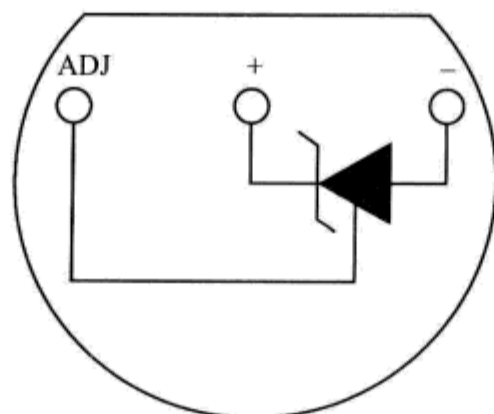


图 13-1 项目 36——串口温度传感器电路图（参见彩色版插图）

如果你把 LM335 平面那侧朝向你，那么左侧的引脚是调整脚，它连接内部元件的调整部分，中间的引脚是电源正引脚，右侧引脚是地引脚，图 13-2 是从 National Semiconductor 公司数据说明书上复制的图。



仰视图

图 13-2 LM335 温度传感器引脚图

输入代码

输入清单 13-1 中的代码。

清单 13-1 项目 36 的代码

```
//项目 36

#define sensorPin 0

float Celsius, Fahrenheit, Kelvin;
int sensorValue;

void setup() {
  Serial.begin(9600);
  Serial.println("Initialising.....");
}

void loop() {

  GetTemp();
  Serial.print("Celsius: ");
  Serial.println(Celsius);
  Serial.print("Fahrenheit: ");
  Serial.println(Fahrenheit);
  Serial.println();

  delay(2000);
}

void GetTemp()
{
  sensorValue = analogRead(sensorPin); //读传感器
  Kelvin = (((float(sensorValue) / 1023) * 5) * 100); //转换到开氏温度
  Celsius = Kelvin - 273.15; //转换到摄氏度
  Fahrenheit = (Celsius * 1.8) + 32; //转换到华氏度
}
```

输入代码后上载到 **Arduino**，一旦代码运行，打开串口监视器并且确认波特率设置为 **9600**，你将看到用摄氏和华氏单位表示的温度显示出来。温度值看起来可能有点不对，这就要调阻计发挥作用了。你首先必须校对传感器。最简单的方法是用冰。拿一个冰块，把

传感器放入一个薄塑料袋中，或者把传感器放在一段热缩管中，将它封装起来，就可以防水了，就可以直接将它放在一块冰上了。将冰块放在传感器上，保持大约 30 秒时间，使它降低到 0℃（或 32°F），然后旋转调阻计，直到从串口中读到正确温度，传感器就校正好了。

把调阻计从电路中拿走，电路也会工作得很好，然而，温度将产生一个 1℃ 以内的误差。传感器是如何工作的不重要（实际上原理是相当复杂的），因此我将简单地看一下在这个项目中代码是如何工作的。如果你想多了解这类传感器的工作原理，可以读读《电子艺术》这本书。这本书是 Horowitz 和 Hill 写的，经常被认为是“电子技术的圣经”。

代码回顾

这个项目的代码是短小简单的。先定义传感器的引脚，在这里使用模拟引脚 0。

```
#define sensorPin 0
```

之后，你需要一些变量来存储用摄氏度、华氏度和开氏度为单位的温度值。因为希望数值比较精确，所以变量类型是 float。

```
float Celsius, Fahrenheit, Kelvin;
```

之后，生成一个整型变量来存储从模拟引脚中读到的值：

```
int sensorValue;
```

在 setup 循环中设置波特率 9600，开始串口通信：

```
Serial.begin(9600);
```

之后，显示“Initializing.....”，表示程序就要开始：

```
Serial.println("Initialising.....");
```

在主程序循环中调用 GetTemp() 函数，它从传感器获得当前温度值，然后把它转化成摄氏和华氏度，之后在串口监视器中打印温度值。

```
GetTemp();  
Serial.print("Celsius: ");  
Serial.println(Celsius);
```

```
Serial.print("Fahrenheit: ");
Serial.println(Fahrenheit);
Serial.println();
```

现在生成 `GetTemp()` 函数:

```
void GetTemp()
```

首先读传感器的值, 并把值存储在 `sensorValue` 中:

```
sensorValue = analogRead(sensorPin); //读传感器
```

从传感器中输出的是开氏温度, 每 k 是 $10mV$, 开氏温度的 0 度是温度的绝对零度, 或者说是世界上可能获得的最低温度。因此温度为绝对零度时, 传感器将输出 $0V$ 。根据说明书, 传感器在 $25^{\circ}C$ 时的电压是 $2.98V$, 通过这个值可以计算当前温度。对于将开氏温度转换到摄氏温度, 只需要从开氏温度减去 273.15 就可转换到摄氏温度, 因此 25 摄氏度对应 298.15 开氏度。如果每度是 $10mV$, 那么, 你只要简单地将小数点向左移动两位, 就可得到与这个温度对应的电压值, 该电压是 $2.98V$ 。

因此, 用开氏表示从传感器中读到温度的电压值, 它的范围是从 $0 \sim 1023$ 。将该值分成 1023 份, 再把结果乘以 5 , 映射到 0 到 $5V$, 因为每度是 $10mV$, 你需要把结果乘以 100 得到浮点型的开氏温度。

```
Kelvin = (((float(sensorValue) / 1023) * 5) * 100); //转换到开氏温度
```

因为 `sensorValue` 的值是整型, 因此要转化到浮点型, 以保证整个计算的结果是浮点数。

现在你要读到的是以 K 为单位的温度, 这是很容易转换成摄氏度或华氏度的。为了转换成摄氏度, 从开氏温度中减去 273.15 :

```
Celsius = Kelvin - 273.15; //转换到摄氏度
```

要转换成华氏度, 需要摄氏度值乘以 1.8 加上 32 :

```
Fahrenheit = (Celsius * 1.8) + 32; //转换到华氏度
```

LM135 传感器非常好用, 因为它非常容易校正, 所以能保证每次都得到一个精确的读

数。它也非常便宜，因此你可以多买几个，利用它从房子中不同的地方获得温度值或者将它用在高度球项目中。

其他的模拟传感器也可以用，你会发现在一些传感器上有第三个引脚。在 LM335 中它是一个 adj（调整）引脚，而在其他温度传感器中这个引脚是温度输出引脚，因此，你应该使用第三个引脚读温度值而不是用它提供电压。这种形式的传感器校正可以非常容易地在软件中实现。

下面你将接触到一种数字温度传感器，最常用的类型是 Maxim 公司的 DS18B20。

项目 37——单线数字温度传感器

现在要看一下 DS18B20 数字温度传感器，这种温度传感器通过单线以串行数据串的方式送出温度值，这是为什么这个协议叫单线协议的原因。每一个传感器都有一个独一无二的序列号，允许你使用 ID 号查询不同的传感器。这样，你可以用同一个数据线连接许多传感器，它在 Arduino 应用中非常流行，因为传感器可以以菊花链的方式连在一起，几乎没有数量的限制，而所有的传感器只占用 Arduino 的一个引脚。这种温度传感器的测量范围也非常宽，在 $-55^{\circ}\text{C} \sim +125^{\circ}\text{C}$ 之间。

在这个项目中，你会用到两个传感器。你要学的不仅是如何连接和使用这种传感器，更重要的是学会如何使用菊花链将两个或更多个传感器连在一起。

需要的元件

你需要 2 个 DS18B20 传感器，TO-90 封装形式（这只是意味着它有 3 个引脚，可以很容易地插入面包板或焊在 PCB 上）。有些带有 BS18B20+ 标志，表示它们是无铅的。

2 个 DS18B20 温度传感器



4.7kΩ 电阻



把元件连接起来

如图 13-3 所示那样连接每个元件。

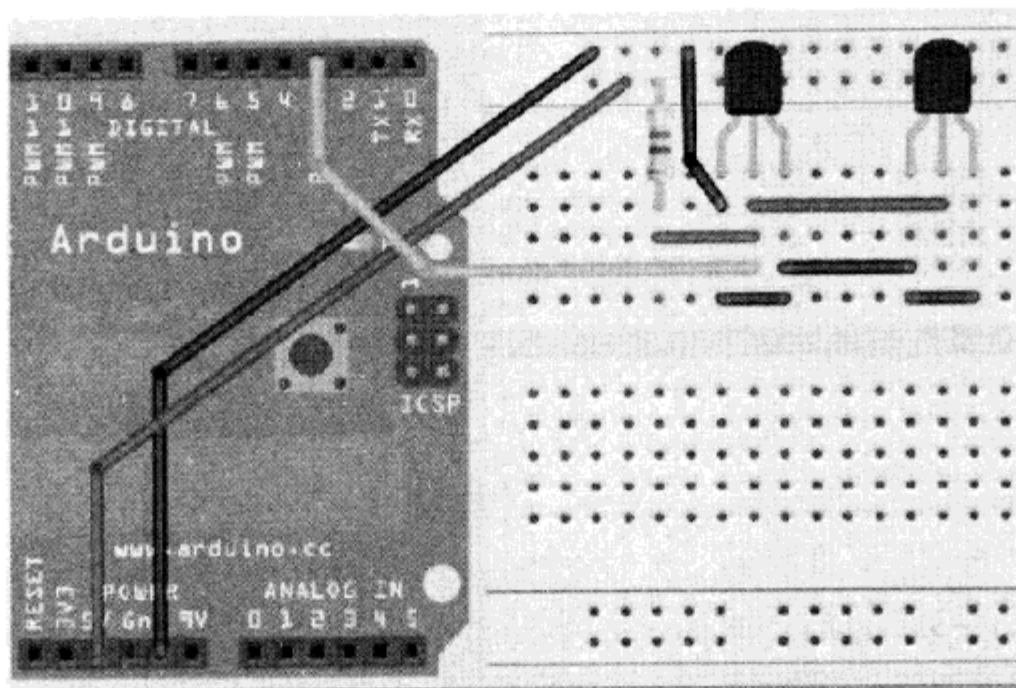


图 13-3 项目 37——单线数字温度传感器电路图（参见彩色版插图）

我把代码分成两部分，第一部分要找到两个传感器的地址。一旦你知道了它们的地址，就可进行第二部分操作。第二部分根据地址直接从传感器中获得温度。

输入代码

在你输入代码之前，需要下载并安装两个库。第一个是 OneWire 库，从 www.pjrc.com/teensy/td_libs_OneWire.html 下载并解压。OneWire 库最初是由 Jim Studt 写的，之后由 Robin James、Paul Stoffregen 和 Tom Pollard 改进。这个库可用来与单线传感器通信，把它放在 Arduino 安装目录中的“libraries”文件夹内。

之后，从网址 http://milesburton.com/index.php?title=Dallas_Temperature_Contror_Library 下载并安装 Dallas Temperature 库，再次把它安装到“libraries”文件夹内。这个库是 OneWire 的衍生库，是由 Miles Burton 开发、由 Tim Newsome 和 James Whiddon 改进的，这个项目就是这个库给出的例子程序。

安装这两个库，重启 Arduino IDE 之后，输入清单 13-2 中的代码。

清单 13-2 项目 37 的代码（第 1 部分）

//项目 37——第 1 部分

```
#include <OneWire.h>
#include <DallasTemperature.h>
```

```

//定义要用来从传感器中读取数据的数字引脚
#define ONE_WIRE_BUS 3

//生成一个 oneWire 实例连接任意单线设备（不仅仅是 Maxim/Dallas 温度传感器）
OneWire oneWire(ONE_WIRE_BUS);

//传递 oneWire 引用给 Dallas 温度传感器
DallasTemperature sensors(&oneWire);

//存储元件地址的数组
DeviceAddress insideThermometer, outsideThermometer;

void setup()
{
    //开始串口通信
    Serial.begin(9600);

    //开始使用库函数
    sensors.begin();

    //确定总线上的传感器
    Serial.print("Locating devices...");
    Serial.print("Found ");
    Serial.print(sensors.getDeviceCount(), DEC);
    Serial.println(" devices.");

    if (!sensors.getAddress(insideThermometer, 0)) Serial.println("Unable to
find address for Device 0");
    if (!sensors.getAddress(outsideThermometer, 1)) Serial.println("Unable to
find address for Device 1");
    //打印两个传感器的地址
    Serial.print("Device 0 Address: ");
    printAddress(insideThermometer);
    Serial.println();

    Serial.print("Device 1 Address: ");
    printAddress(outsideThermometer);
    Serial.println();
    Serial.println();
}

```



```
//打印地址的函数
void printAddress(DeviceAddress deviceAddress)
{
    for (int i = 0; i < 8; i++)
    {
        //必要时用 0 填补地址
        if (deviceAddress[i] < 16) Serial.print("0");
        Serial.print(deviceAddress[i], HEX);
    }
}

//给传感器打印温度的函数
void printTemperature(DeviceAddress deviceAddress)
{
    float tempC = sensors.getTempC(deviceAddress);
    Serial.print("Temp C: ");
    Serial.print(tempC);
    Serial.print(" Temp F: ");
    Serial.print(DallasTemperature::toFahrenheit(tempC));
}

//打印传感器信息的主函数
void printData(DeviceAddress deviceAddress)
{
    Serial.print("Device Address: ");
    printAddress(deviceAddress);
    Serial.print(" ");
    printTemperature(deviceAddress);
    Serial.println();
}

void loop()
{
    //调用 sensors.requestTemperatures() 函数查询全部的传感器
    //请求总线上的所有设备
    Serial.print("Requesting temperatures...");
    sensors.requestTemperatures();
    Serial.println("DONE");

    //打印设备信息
    printData(insideThermometer);
    printData(outsideThermometer);
}
```

```

    Serial.println();
    delay(1000);
}

```

一旦代码已经上传，打开串口监视器，你将看到如下的显示：

```
Locating devices...Found 2 devices.
```

```
Device 0 Address:28CA90C202000088
```

```
Device 1 Address:283B40C202000093
```

```
Requesting temperatures...DONE
```

```
Device Address: 28CA90C202000088 Temp C:31.00 Temp F:87.80
```

```
Device Address: 283B40C202000093 Temp C:25.31 Temp F:77.56
```

这个程序给出你所使用的两个 DS18B20 传感器 ID 号，它们是独一无二的。可以通过改变这两个传感器的温度找到哪一个 ID 号对应哪个传感器。我用手握住右边的传感器，看到其中一个温度上升很快。这告诉我右侧传感器的地址是 28CA90C202000088，左边的那个是 283B40C202000093。你所用的传感器的地址显然跟我所用的是不同的。抄下它们的 ID 号或者复制粘贴它们到你的文本编辑器上。

现在你知道了两个元件的 ID 号，可以进行第二部分了，输入清单 13-3 中的代码。

清单 13-3 项目 37 的代码（第 2 部分）

```

//项目 37——第 2 部分

#include <OneWire.h>
#include <DallasTemperature.h>

//数据总线连接到 Arduino 的数字引脚 3 上
#define ONE_WIRE_BUS 3
#define TEMPERATURE_PRECISION 12

//建立 oneWire 实例连接任意的 OneWire 设备（不仅仅是 Maxim/Dallas 温度传感器）
OneWire oneWire(ONE_WIRE_BUS);

//传递 oneWire 引用给 Dallas 温度传感器
DallasTemperature sensors(&oneWire);

```

```
//存储元件地址的数组——用你自己的传感器地址代替
DeviceAddress insideThermometer = { 0x28, 0xCA, 0x90, 0xC2, 0x2, 0x00, 0x00, 0x88 };
DeviceAddress outsideThermometer = { 0x28, 0x3B, 0x40, 0xC2, 0x02, 0x00, 0x00, 0x93 };

void setup()
{
    //开始串口通信
    Serial.begin(9600);

    //开始使用库函数
    sensors.begin();

    Serial.println("Initialising...");
    Serial.println();

    //设置精度
    sensors.setResolution(insideThermometer, TEMPERATURE_PRECISION);
    sensors.setResolution(outsideThermometer, TEMPERATURE_PRECISION);
}

//打印传感器检测到的温度
void printTemperature(DeviceAddress deviceAddress)
{
    float tempC = sensors.getTempC(deviceAddress);
    Serial.print(" Temp C: ");
    Serial.print(tempC);
    Serial.print(" Temp F: ");
    Serial.println(DallasTemperature::toFahrenheit(tempC));
}

void loop()
{
    //打印温度
    Serial.print("Inside Temp:");
    printTemperature(insideThermometer);
    Serial.print("Outside Temp:");
    printTemperature(outsideThermometer);
    Serial.println();
    delay(3000);
}
```

用在第 1 部分中你得到的传感器 ID 代替上文中的 ID，之后上传代码，打开串口监视

器，你将看到如下信息：

```
Initialising...

Inside Temp: Temp C:24.25 Temp F:75.65
Outside Temp: Temp C:19.50 Temp F:67.10

Inside Temp: Temp C:24.37 Temp F:75.87
Outside Temp: Temp C:19.44 Temp F:66.99

Inside Temp: Temp C:24.44 Temp F:75.99
Outside Temp: Temp C:19.37 Temp F:66.87
```

如果你将外边的两个传感器焊接到长双绞线上（把引脚 1 和引脚 3 焊接在一起用一根线，引脚 2 用第二根线），之后用热缩管封上进行防水处理。将它放置在外边来获得室外温度，第二个传感器获得室内温度。

代码回顾

首先是包含两个库：

```
#include <OneWire.h>
#include <DallasTemperature.h>
```

之后定义要用来从传感器中读取数据的数字引脚：

```
#define ONE_WIRE_BUS 3
```

接下来用比特形式定义需要的精度：

```
#define TEMPERATURE_PRECISION 12
```

精度可以设置在 9~12 比特分辨率之间。相应的分辨率是 0.5℃、0.25℃、0.125℃。默认分辨率是 12 比特。最大的分辨率 12 比特给出最小的温度增量，但是降低了速度。采用最大分辨率时，传感器要用 750 毫秒时间转换成温度。采用 11 比特时，转换时间只有最大分辨率时的一半——385 毫秒。10 比特又降低一半时间，是 187.5 毫秒，最后 9 比特用时 93.7 毫秒。750 毫秒对大多数应用来说是足够快的了。然而，如果由于某种原因，你需要在一秒内读取几个温度值，那么采用 9 比特分辨率将给出最快的转换时间。

之后，生成一个 `OneWire` 对象，叫做 `oneWire`：

```
OneWire oneWire(ONE_WIRE_BUS);
```

生成一个 `DallasTemperature` 对象实例，叫做 `sensors`，并且传递一个叫 `oneWire` 的参数给这个对象。

```
DallasTemperature sensors(&oneWire);
```

之后，你需要生成一个数组来存储传感器地址。`DallasTemperature` 库定义各种形式的 `DeviceAddress`（它只是字节型 8 元素数组），我们生成两个 `DeviceAddress` 型变量，分别称为 `insideThermometer` 和 `outsidethermometer`。之后用在第 1 部分中获得的 ID 号给两个数组赋值。

把你在第 1 部分中得到的地址分成两个 16 进制数，并且在前面加上 `0x`（告诉编译器它们是 16 进制数而不是标准的十进制数），每个数之间用逗号分开。地址将分成两个 8 位二进制数。

```
DeviceAddress insideThermometer = { 0x28, 0xCA, 0x90, 0xC2, 0x2, 0x00, 0x00, 0x88 };
DeviceAddress outsideThermometer = { 0x28, 0x3B, 0x40, 0xC2, 0x02, 0x00, 0x00, 0x93 };
```

在 `setup` 循环中开始用 9600 波特率进行串行通信：

```
Serial.begin(9600);
```

之后，与传感器对象的通信用 `.begin()` 函数开始：

```
sensors.begin();
```

打印字符 “`Initializing...`”，显示程序已经开始，接下来是一个空行：

```
Serial.println("Initialising...");
Serial.println();
```

之后用 `.setResolution` 函数设置每个传感器的精度。这个函数需要两个参数，第一个是元件的地址，第二个是精度。你已经在程序的开始设置了精度为 12 比特。

```
sensors.setResolution(insideThermometer, TEMPERATURE_PRECISION);
```

```
sensors.setResolution(outsideThermometer, TEMPERATURE_PRECISION);
```

生成一个叫做 `printTemperature()` 的函数，它用摄氏度和华氏度为单位打印温度，函数的参数是传感器的地址，这是该函数唯一的参数。

```
void printTemperature(DeviceAddress deviceAddress)
```

使用 `getTemp()` 函数从指定地址的元件获得温度，以摄氏度为单位。将结果存储到一个叫做 `tempC` 的浮点型变量中。

```
float tempC = sensors.getTempC(deviceAddress);
```

之后打印出这个温度

```
Serial.print(" Temp C: ");
Serial.print(tempC);
```

接着打印以华氏度为单位的温度：

```
Serial.print(" Temp F: ");
Serial.println(DallasTemperature::toFahrenheit(tempC));
```

使用 `::` 来获得 `toFahrenheit` 函数，它在 `DallasTemperature` 库里。它将摄氏度值转换成华氏度值。

在主循环中，你只调用 `printTemperature()` 函数两次传递内部传感器的地址，之后传递外部传感器的地址，每一次跟随 3 秒的延时。

```
Serial.print("Inside Temp:");
printTemperature(insideThermometer);
Serial.print("Outside Temp:");
printTemperature(outsideThermometer);
Serial.println();
delay(3000);
```

我建议你尝试 `DallasTemperature` 库中的各种例子，因为这些例子可以使你更好地理解库中的各种功能。我也建议你阅读 `DS18B20` 的说明书。这个传感器也可以在它的内部设置报警，在满足一定条件的情况下触发，这对于判断传感器的过热或过冷条件是有用的。

DS18B20 是功能强大的传感器，它有很宽的温度测量范围，有优于模拟传感器的优点——许多个传感器可以采用菊花链方式连在同一条数据线上，因此不管你用多少个传感器，仅需要一个引脚就可以。

在本书后面，我们要接触一种完全不同的传感器，它使用声波测量。

小结

在本章中，你做了两个简单的项目，演示了如何连接模拟和数字温度传感器到 Arduino 上。项目演示了如何从每个传感器中读数据并且把它显示在串口监视器中。一旦你知道了如何实现这些功能，就可以非常容易地使数据显示在 LCD 或 LED 点阵显示器上。

本章告诉你如何从传感器中获得温度读数，增加了你对 Arduino 的热情。在本书的后面，在第 17 章进入实际应用时，会再次用到温度传感器。

本章的主题和概念

- 如何连接一个模拟温度传感器到 Arduino
- 如何使用调阻计校正 LM135 系列传感器
- 如何将从传感器获得的电压转换成开氏温度
- 如何实现开氏温度与摄氏温度的相互转换
- 如何通过使用热缩管使传感器具有防水功能
- 如何连接单线传感器到 Arduino 上
- 单线传感器可以实现菊花链连接
- 单线传感器具有独一无二的 ID 号
- 如何设置 DS18B20 传感器精度
- 高精度对应低转换速度

第 14 章

传感器

超声测距

你现在要接触一种不同的传感器，这种传感器在机器人和工业中用得很多。超声测距通过向被测物体发射一系列超声脉冲，测量脉冲返回时间以检测与被测物体之间的距离。本项目用现在流行的超声测距传感器 MAXbotix LV-MaxSonar。首先学习将传感器连接到 Arduino 的基本知识，之后把这个传感器用于实践。

项目 38——简单的超声测距仪

LV-MaxSonar 超声测距仪有 EZ1、EZ2、EZ3 和 EZ4 型。所有型号都有同样的测量范围，但是它们的发射角逐渐变窄，你可以根据需要选用相应的传感器。在本章的项目中，我使用一个 EZ3 型的超声测距仪，但是你可以选择其他型号的。

需要的元件

LV-MaxSonar EZ3*



100uF 电容



100Ω电阻



*或其他型号（感谢 Sparkfun 供图）。

把元件连起来

如图 14-1 所示那样连接每个元件。

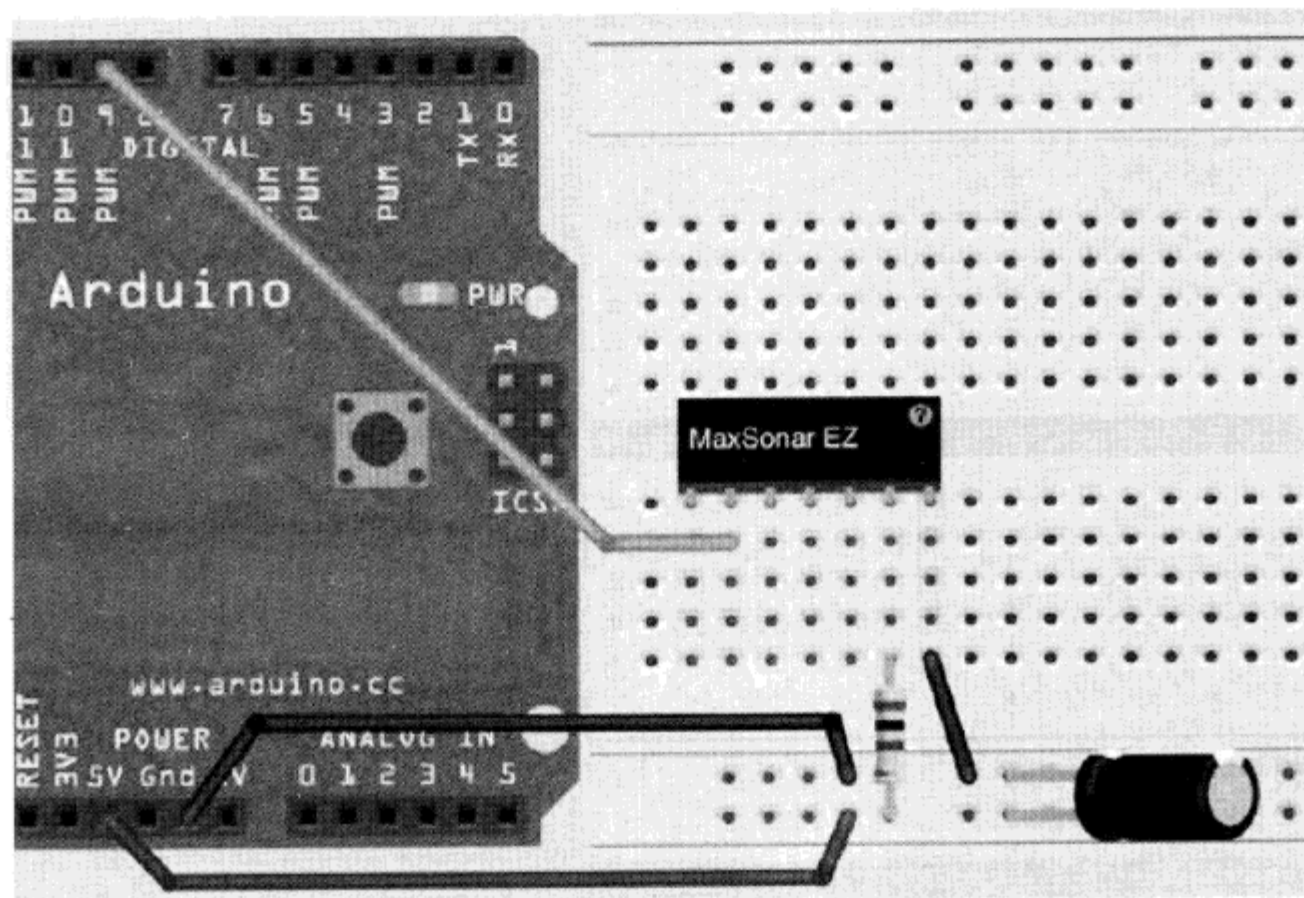


图 14-1 项目 38——简单的超声测距仪电路图（参见彩色版插图）

因为在 Fritzing（用来生成本书中面包板图片的软件）的元件库中没有 LV-MaxSonar，所以我使用万能元件替代。连接 5V 和地到面包板电源母线上。在电源母线之间放置 100uF 的电容。确认元件的长脚连接 5V，短脚（也就是有白环标志穿过的脚）接地线。之后在地和传感器的 Gnd 引脚之间连接一个跳线。一定要把极性搞对，因为如果连反，电容可能爆炸。之后在 5V 母线和传感器的+5V 引脚之间连接一个 100Ω 的电阻。最后在传感器上的 PW 引脚和数字引脚 9 之间连接一条线。

输入代码

检查电路，确保连线正确，输入清单 14-1 中的代码，上载到你的 Arduino 中。

清单 14-1 项目 38 的代码

```
//项目 38
#define sensorPin 9
```

```

long pwmRange, inch, cm;

void setup() {
    //开始串口通信
    Serial.begin(115200);
    pinMode(sensorPin, INPUT);
}

void loop() {
    pwmRange = pulseIn(sensorPin, HIGH);

    //根据说明书 147  $\mu$ s 相当于一英寸
    inch = pwmRange / 147;
    //把英寸转化成厘米
    cm = inch * 2.54;

    Serial.print(inch);
    Serial.print(" inches  ");
    Serial.print(cm);
    Serial.println(" cm");
}

```

上载代码后，给 Arduino 断电 1 秒。之后确认超声传感器指向静止的物体。最好把它平放在桌子上，让它指向天花板。当你重新给 Arduino 上电时，确认在传感器附近没有任何东西。当元件第一次上电时，在读数据之前先运行一个调校程序。当进行调校时，确保在它周围没有任何物体，否则你将得到不精确的读数。这个信息将用于检测物体到传感器的直线距离。接下来要测量传感器和天花板之间的距离。当你打开串口监视器时，这个距离（大概的）将输出到串口监视器上。如果这个距离不精确，给 Arduino 下电，之后重新上电，让元件在没有障碍物的情况下进行调校。把你的手放在传感器的上面上下摆动或到处移动传感器，你就可以在串口监视器上看传感器与它所检测物体之间的距离。

代码回顾

使用超声传感器的代码是非常短小、简单的。首先，定义用来检测脉冲的引脚，在这里使用数字引脚 9：

```
#define sensorPin 9
```

之后声明 3 个长整型变量：

```
long pwmRange, inch, cm;
```

这些将用来存储从传感器读到的返回距离。距离的单位是英寸，之后将转化成厘米。

在 `setup` 函数里，用 115200 波特率开始串口通信，设置传感器引脚为输入：

```
Serial.begin(115200);  
pinMode(sensorPin, INPUT);
```

在主循环中，先从传感器引脚中读脉冲，之后把它存储在 `pwmRange` 中：

```
pwmRange = pulseIn(sensorPin, HIGH);
```

为了完成这个任务，你使用了新的函数 `pulseIn`，这个函数用于满足测量脉冲长度的需要，用毫秒为单位，测量一个引脚上的脉冲长度。当超声脉冲从元件送出时，传感器上的 PW 引脚送出 HIGH 信号。当脉冲返回时，PW 引脚送出 LOW 信号。获得引脚在高-低电平之间的转换时间，之后换算成声音传播的距离。`pulseIn` 命令需要两个参数，第一个参数是接收回音的引脚，第二个参数是电平的高或低状态，用于定义在什么状态下 `pulseIn` 命令开始给脉冲计时。在这个例子里，你已经设置第二个参数为 HIGH，因此，传感器的引脚已变成 HIGH，`pulseIn` 命令将开始计时，一旦它变为 LOW，它停止计时，之后返回以毫秒为单位的时间。根据 LV-MaxSonar 传感器的范围说明书，这个元件测距范围为 0~254 英寸（6.45 米），在低于 6 英寸时输出为 6 英寸。每 147 μ s（ μ 秒）相当于一英寸。因此要将从 `pulseIn` 返回的值转换成英寸，只需要把它除以 147。这个值将存储在 `inch` 中。

```
inch = pwmRange / 147;
```

之后，这个值乘以 2.54 给出以厘米为单位的距离：

```
cm = inch * 2.54;
```

最后，该值以英寸和厘米为单位打印到串口监视器上：

```
Serial.print(inch);  
Serial.print(" inches  ");  
Serial.print(cm);  
Serial.println(" cm");
```

硬件回顾

这个项目中介绍的新元件是超声测距仪。这个元件使用超声波，超声波是频率很高的声音，其频率超出人类可以听到的声音的频率范围。MaxSonar 传感器发出的超声波脉冲频率是 42kHz，人类平均可听到的频率上限是 20kHz，因此这个传感器工作在人类可以听到的声音范围之外。一个超声脉冲通过一个元件的转换器发出，当超声波碰到物体时发生反射，通过同一个转换器得到回声。通过计算脉冲的往返时间，你可以算出传感器到反射物体的距离（见图 14-2）。声波以声速传播，声波在干空气内 20℃（68°F）时传播的速度是每秒 343 米，或每秒 1125 英寸。知道这个参数和以 μ 秒为单位的声波返回传感器的时间，你可以算出距离。根据说明书，脉冲返回时每英寸距离花 147 μ s 的时间。因此，用得到的 μ 秒时间除以 147 可以算出以英寸为单位的距离。如果需要，你可以把它转化成厘米。

这个测量方法也叫 SONAR（声音导航或测距）。该方法用于潜艇中，可以用声纳检查到其他军船或周围沉船的距离。它也是蝙蝠测量自身与猎物距离的方法。

从 MaxSonar 传感器中读取数据有三种方法。一种是模拟量输入方法，第二种是 PWM 输入方法，最后一种是串行接口方法。PWM 输入可能是最简单的方法，所测得的数据也最准确，因此我们在这里使用 PWM 方法。如果你有兴趣，可以自己研究和使用的另外两种方法。尽管这样做也没有什么实际的好处，除非你指定使用模拟量或串行数据方法获得传感器输出的数据。

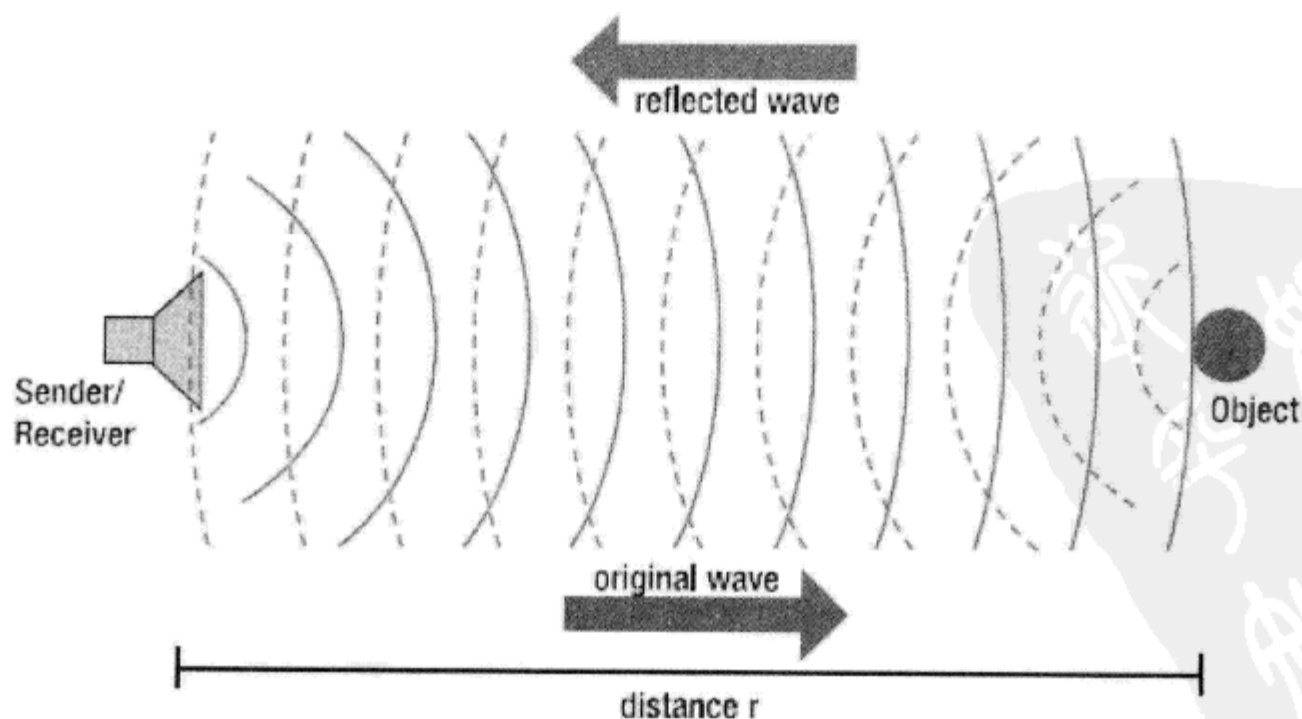


图 14-2 声纳或雷达测距原理图（Georg Wiora 供图）

现在你知道了传感器的工作原理，让我们将它用到实际项目中，做一个超声波尺子或测距仪。

项目 39——超声测距显示器

我们用超声传感器制作一个相当精确的距离显示器。采用第 7 章用过的 Max7219 LED 驱动芯片显示测量到的距离。这次我们不再使用点阵显示器，而是使用专门为 MAX7219 设计的一个 7 段 LED 显示器。

需要的元件

LV-MaxSonar EZ3*



100uF 电容



2 个 100Ω电阻



100kΩ电阻



开关



5 个 7 段 LED 显示器



MAX7219 LED 驱动芯片



*或其他需要的元件（感谢 Sparkfun 供图）。

本项目所用的开关必须是单极的双投开关（SPDT），这种开关是一种滑动开关，它能保持在两个位置中的某一个位置上。你将使用它转换显示英寸和厘米。7 段 LED 显示器必须是共阴极形式。一定要找到你购买的元件的说明书，这样你才能知道如何连接它们，因为你买到的元件可能与我的不同。

把元件连接起来

如图 14-3 所示那样连接各元件。

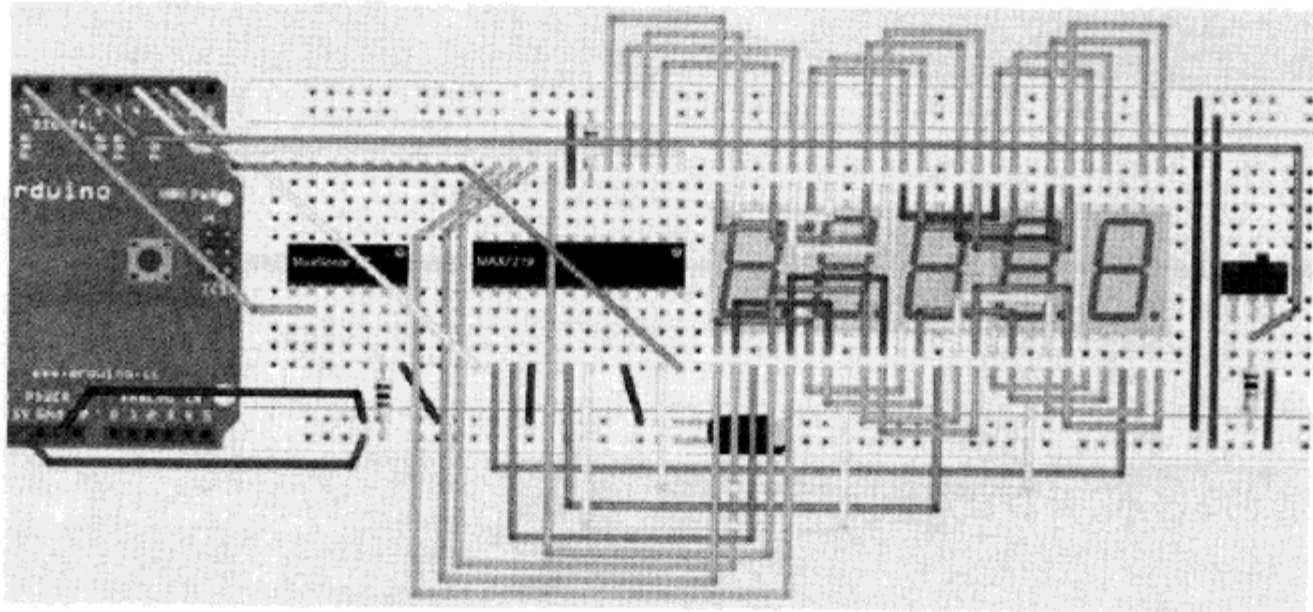


图 14-3 项目 39——超声测距显示仪电路图（参见彩色版插图）

这个电路有点复杂，因此下面我将为 Arduino、Max7219 和 7 段显示器提供一个引脚表（表 14-1），你可以把它和电路图进行对照。我用的显示器型号是 5101AB，任何共阴极 7 段显示器都是可以用的。确认引脚是从顶部和底部而不是从两侧伸出的。若从两侧引出引脚就不能把 7 段显示器插入面包板中。

表 14-1 项目 39 需要的输出引脚

Arduino	MaxSonar	MAX7219	7 段显示器	其 他
数字引脚 2		Pin1 (DIN)		
数字引脚 3		Pin12 (LOAD)		
数字引脚 4		Pin13 (CLK)		
数字引脚 7				开关
数字引脚 9	PW	Pin4 (Gnd)		Gnd
		Pin9 (Gnd)		Gnd
		Pin18 (ISET)		Gnd 通过 10kΩ电阻
		Pin19 (VDD)		+5V
		Pin2 (DIG 0)	0 号 7 段管的 Gnd	
		Pin11 (DIG 1)	1 号 7 段管的 Gnd	
		Pin6 (DIG 2)	2 号 7 段管的 Gnd	
		Pin2 (DIG 3)	3 号 7 段管的 Gnd	
		Pin2 (DIG 4)	4 号 7 段管的 Gnd	
		Pin14	SEG A	

续表

Arduino	MaxSonar	MAX7219	7 段显示器	其 他
		Pin16	SEG B	
		Pin20	SEG C	
		Pin23	SEG D	
		Pin21	SEG E	
		Pin15	SEG F	
		Pin17	SEG G	
		Pin22	SEG DP	

先连接 MAX7219 到第一个 7 段显示器的 SEG A-G 和 DP 引脚上，如最近的芯片，见图 14-4。将第一个显示器上的 SEG 引脚连接到第二个，第二个连到第三个，依此类推。所有显示器上的 SEG 引脚都连在一起，ground 引脚单独地连到 MAX7219 的相应 DIG 引脚上。一定要阅读你的 7 段显示器说明书，因为它可能与我所使用的不一样。

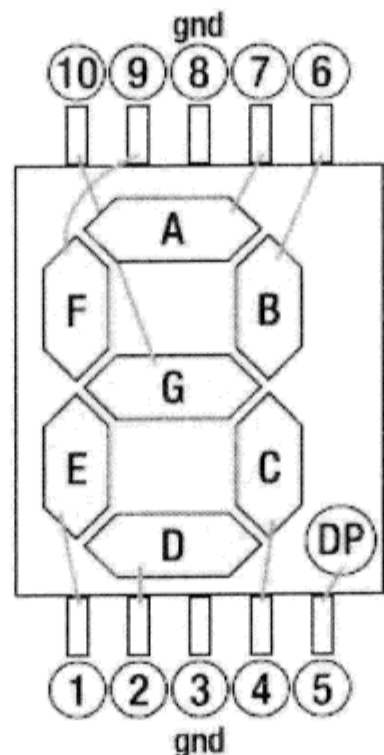


图 14-4 一个典型的共阴极 7 段 LED 显示引脚图（感谢 Jan-Piet Mens 供图）

MaxSonar 的连接与前面的相同，只是 PW 引脚要连到数字引脚 9 上而不是引脚 3 上。最后，数字引脚 T 连到乒乓开关上。

注意，在这个项目中，如果你发现有什么不对头——7 段显示器拉低了 USB 总线的电压，使电压很低，你可能就需要使用额外的电源。

输入代码

将连线连接正确之后, 给 Arduino 上电, 输入清单 14-2 中的代码, 之后上载到 Arduino, 确保在 Arduino 的库文件夹中已有 LedControl.h 库 (见第 7 章的说明)。

清单 14-2 项目 39 的代码

```
//项目 39

#include "LedControl.h"

#define sensorPin 9
#define switchPin 7
#define DataIn 2
#define CLK 4
#define LOAD 3
#define NumChips 1
#define samples 5.0

float pwmRange, averageReading, inch, cm;
LedControl lc=LedControl(DataIn,CLK,LOAD,NumChips);

void setup() {
    //唤醒 MAX7219
    lc.shutdown(0,false);
    //设置亮度为中间亮度
    lc.setIntensity(0,8);
    //清除显示器显示
    lc.clearDisplay(0);
    pinMode(sensorPin, INPUT);
    pinMode(switchPin, INPUT);
}

void loop() {
    averageReading = 0;
    for (int i = 0; i<samples; i++) {
        pwmRange = pulseIn(sensorPin, HIGH);
        averageReading += pwmRange;
    }

    averageReading /= samples;
    //根据说明书 147μS 对应 1 英寸
```

```

    inch = averageReading / 147;
    //转化英寸到厘米
    cm = inch * 2.54;

    if (digitalRead(switchPin)) {
        displayDigit(inch);
    }
    else {
        displayDigit(cm);
    }
}

void displayDigit(float value) {
    int number = value*100;
    lc.setDigit(0,4,number/10000,false); //百位数
    lc.setDigit(0,3,(number%10000)/1000,false); //十位数
    lc.setDigit(0,2,(number%1000)/100,true); //个位数带小数点
    lc.setDigit(0,1,(number%100)/10,false); //十分位数
    lc.setDigit(0,0,number%10,false); //百分位数
}

```

代码回顾

这个项目开始包含 LedControl 库：

```
#include "LedControl.h"
```

之后，定义传感器和 MAX7219 芯片需要的引脚：

```

#define sensorPin 9
#define switchPin 7
#define DataIn 2
#define CLK 4
#define LOAD 3
#define NumChips 1

```

传感器的读数需要做平滑处理，使用简单的取平均值法就可以，因此你需要定义求平均值所用的样本数量：

```
#define samples, 5.0
```

之后，将这个数作为浮点形式使用。为了避免错误，这个数字定义为 5.0 而不是 5，强制它为浮点数而不是整数：

```
#define samples 5.0
```

之后，像项目 38 那样定义浮点数，这里增加了 `averageReading` 变量，它将在程序后面使用：

```
float pwmRange, averageReading, inch, cm;
```

生成 `LedControl` 对象并且设置用到的引脚和芯片数量：

```
LedControl lc=LedControl(DataIn,CLK,LOAD,NumChips);
```

像项目 21 一样，确保显示器可用，光强设置为中间值，清除显示器显示，为使用做好准备：

```
lc.shutdown(0,false);
lc.setIntensity(0,8);
lc.clearDisplay(0);
```

将传感器和乒乓开关用的引脚都设置为输入：

```
pinMode(sensorPin, INPUT);
pinMode(switchPin, INPUT);
```

接下来是主循环代码，首先把 `averageReading` 变量设置为 0：

```
averageReading = 0;
```

之后，程序采用一个 `for` 循环从传感器中采集样本。传感器的值像前面代码一样读入 `pwmRange` 中，但是每次循环运行时要加到 `averageReading` 中。`for` 循环将重复的次数定义在程序开始时的 `sample` 中。

```
for (int i = 0; i<samples; i++) {
    pwmRange = pulseIn(sensorPin, HIGH);
    averageReading += pwmRange;
}
```


之后用 `averageReading` 中的值除以 `samples` 中的值。在这个例子里，样本数设置为 5，因此获得没 5 个样本，加到 `averageReading` 中，它的初始值是 0，之后除以 5，算出这 5 个样本的平均值。这么做可以获得更精确的读数，并且可以将读数中的噪声或者在计时时混入的温度和气压变化引起的误差处理掉。

```
averageReading /= samples;
```

如前，将脉冲时间转化成英寸和厘米：

```
inch = averageReading / 147;
cm = inch * 2.54;
```

之后，使用 `if` 语句检查乒乓开关是高电平还是低电平，如果是高电平，那么执行 `displayDigit()` 函数（在后面解释），将以英寸为单位的参数传递给它。如果乒乓开关是低电平，`else` 语句运行，同样运行 `displayDigit()` 函数，但是使用以厘米做单位的参数。

```
if (digitalRead(switchPin)) {
    displayDigit(inch);
}
else {
    displayDigit(cm);
}
```

这个 `if-else` 语句根据乒乓开关的位置决定显示以英寸还是以厘米做单位的数值。

最后，定义 `displayDigit()` 函数。这个函数只是将传送给它的数字打印到 7 段显示器上。传送给这个函数的参数必须是一个浮点数，参数可以用英寸或厘米做单位。

```
void displayDigit(float value) {
```

传递给这个函数的值是浮点值，小数点后有数字。你只对小数点后两位感兴趣，因此用它乘以 100，使小数点向后移动两位：

```
int number = value*100;
```

这是因为你要使用模除操作符号 `%`，它需要整型数作为操作数，因此必须将浮点数转换成整型数。将它乘以 100 保证小数点后保留两位有效数字而其他数字舍去。现在获得了初始值，但是没有小数点。这没有关系，因为你知道小数点后有两位数。

之后，你需要将这个数显示在一个 7 段显示器上。用 `setDigit` 函数显示每个数字。该函数需要 4 个参数：

```
setDigit (int addr, int digit, byte value, boolean dp);
```

其中 `addr` 是 MAX7219 芯片的地址。项目中只用到一个芯片，因此这个值是 0。如果增加第二个芯片，它的地址应是 1，依此类推。`digit` 是被控 7 段显示器的索引，在这个例子里，右手一侧的显示器是数字 0，与它相邻的左边一个是 1，依此类推。`value` 是你希望显示在 7 段 LED 显示器上的实际的数字，从 0 到 9。最后，用布尔值 `false` 或 `true` 决定小数点是否显示。

因此，使用 `setDigit` 命令把值存储在叫 `number` 的整型数中，之后对它进行模除操作，得到一个单独的数字，把它显示在 LED 上：

```
lc.setDigit(0,4,number/10000,false); //百位数
lc.setDigit(0,3,(number%10000)/1000,false); //十位数
lc.setDigit(0,2,(number%1000)/100,true); //个位数带小数点
lc.setDigit(0,1,(number%100)/10,false); //十分位数
lc.setDigit(0,0,number%10,false); //百分位数
```

数字 2 要显示小数点，因为你希望小数点后有两位数字，因此 DP 标志为 `true`。

下面举例说明以上程序是如何工作的。假设我们要显示的数字是 543.21。先将这个数乘以 100，得到 54321。为了获得这个数字的个位上的数字，对这个数进行模 10 操作，得到个位上的数字（最右边的）是 1。

```
54321*100=54321
54321%10=1
```

记住，模除操作%是用它后边的数除一个整数，但是只能留下余数。54321 除以 10 是 5432.1，余数是 1，给出需要显示的第一个数字（最右侧的）。

第 2 个数字（十位）模除 100 之后除以 10，给出所要显示的第二位数字。

```
54321%100=21
21/10=2（记住，这是整型数计算，因此小数点后的数字都丢失了）
```

依此类推。

如果按这个计算方法，用 543.21 作为初始值，在一系列模除和除操作之后，得到初始数字的每一位独立的数字。增加的小数点（右数第三个）保证数字显示为小数点后带两位数字。

这种超声测量方法是相当精确的，其精度能达到 1/100 英寸或厘米。增加小数点后的位数不能使之更精确，因为超声传播得快和慢取决于不同的温度或大气压力。而且，声波从不同的表面返回的时间也是不同的。相当平的、垂直于传感器的平面反射声波效果最好，能够给出最精确的测量值。凸凹不平的表面、吸声的表面或与声波有一定角度的表面所得到的测量值不那么精确。可以用实际的卷尺测量距离的真实值，比较不同的表面返回的读数。

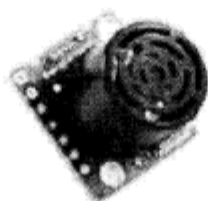
下面，让我们使用超声传感器进行另一个项目。

项目 40——超声报警

本项目在上个项目基础上建立一个电路实现超声报警系统。

需要的元件

LV-MaxSonar EZ3*



100uF 电容



2 个 100Ω电阻



2 个 10kΩ电阻



乒乓开关



5 个 7 段显示器（共阴极的）



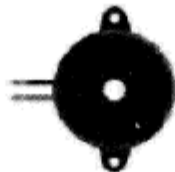
MAX7219 LED 驱动芯片



5~10k Ω 变阻器



嗡鸣器或 8 Ω 扬声器



*或其他元件（感谢 Sparkfun 供图）。

把元件连接起来

如图 14-5 所示那样连接每一个元件。

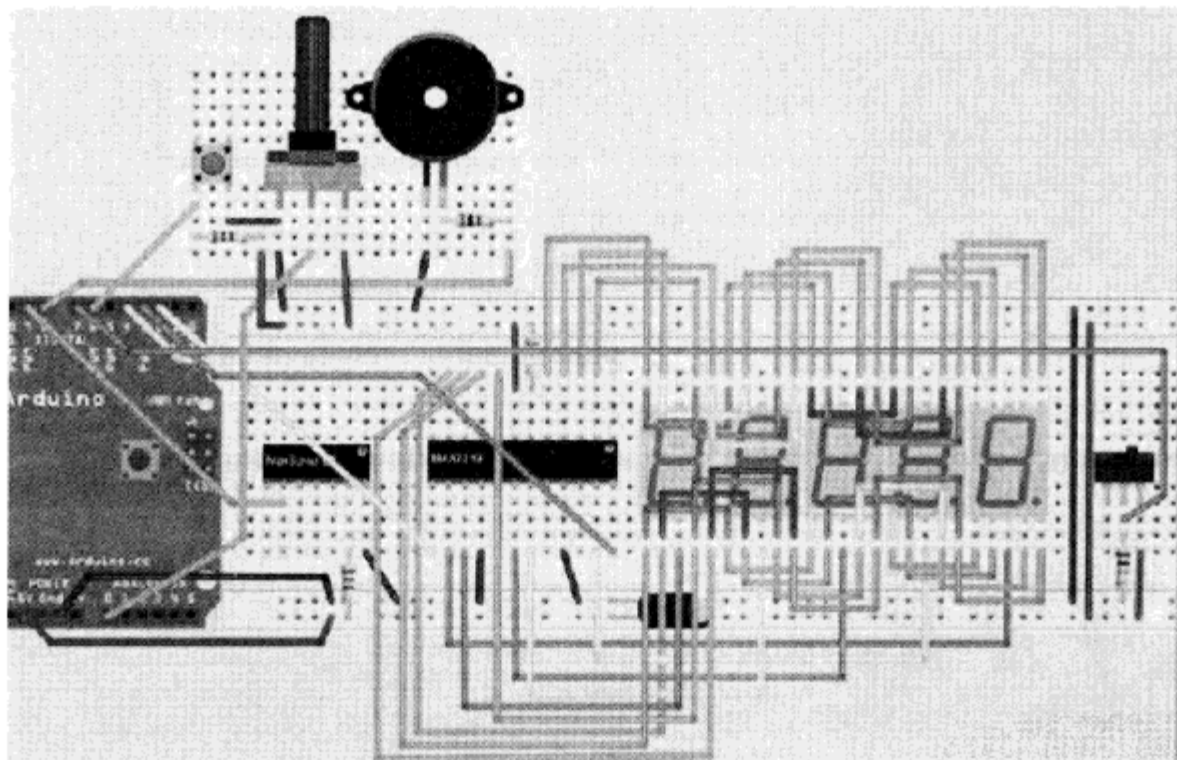


图 14-5 项目 40——超声报警电路图（参见彩色版插图）

电路图与项目 39 完全相同，但是增加了一个按钮、一个变阻器和一个嗡鸣器（或扬声器）。按钮的两端连在+5V 和地之间，+5V 引脚通过一个 10k Ω 电阻连到 5V 上。从同一个引脚引出一条接线连到模拟引脚 0 上。扬声器负极接地，正极通过一个 100 Ω 电阻连到数字引脚 8 上。变阻器用来调整报警传感器的范围。按钮用于警报拉响后复位。嗡鸣器用来提供报警声。

输入代码

检查一遍电路，确保连线正确，给 Arduino 上电。之后上载清单 14-3 中的代码。

清单 14-3 项目 40 的代码

```
//项目 40

#include "LedControl.h"

#define sensorPin 9
#define switchPin 7
#define buttonPin 6
#define potPin 0
#define DataIn 2
#define CLK 4
#define LOAD 3
#define NumChips 1
#define samples 5.0

float pwmRange, averageReading, inch, cm, alarmRange;
LedControl lc=LedControl(DataIn,CLK,LOAD,NumChips);

void setup() {

    //唤醒 MAX7219
    lc.shutdown(0,false);
    //设置亮度为中间亮度
    lc.setIntensity(0,8);
    //清空显示器
    lc.clearDisplay(0);
    pinMode(sensorPin, INPUT);
    pinMode(switchPin, INPUT);
}

void loop() {
    readPot();
    averageReading = 0;
    for (int i = 0; i<samples; i++) {
        pwmRange = pulseIn(sensorPin, HIGH);
        averageReading += pwmRange;
    }

    averageReading /= samples;
    //根据说明书 147μS 相当于 1 英寸
    inch = averageReading / 147;
    //转化英寸到厘米
    cm = inch * 2.54;
}
```

```

    if (digitalRead(switchPin)) {
        displayDigit(inch);
    }
    else {
        displayDigit(cm);
    }

    //当前的距离是否小于 alarmRange 设置的数值, 若是则报警
    if (inch<=alarmRange) {startAlarm();}
}

void displayDigit(float value) {
    int number = value*100;
    lc.setDigit(0,4,number/10000,false); //百位数字
    lc.setDigit(0,3,(number%10000)/1000,false); //十位数字
    lc.setDigit(0,2,(number%1000)/100,true); //个位数字
    lc.setDigit(0,1,(number%100)/10,false); //十分位数字
    lc.setDigit(0,0,number%10,false); //百分位数字
}

//读变阻器
float readPot() {
    float potValue = analogRead(potPin);
    alarmRange = 254 * (potValue/1024);
    return alarmRange;
}

//直到复位按下关闭报警声音
void startAlarm() {
    while(1) {
        for (int freq=800; freq<2500;freq++) {
            tone(8, freq);
            if (digitalRead(buttonPin)) {
                noTone(8);
                return;
            }
        }
    }
}

```

输入代码, 上载代码到你的 Arduino 中。之后, 给系统断电。重新上电, 确认传感器可以正确校正。现在你可以旋转变阻器调整报警范围。把手放在超声波中, 慢慢地向传感

器移动直到发生报警，一旦报警，读数激活并保持显示最后测量到的距离，这是你设置的报警范围。按复位按钮给报警消声，复位系统。之后，调整报警范围，直到你满意为止。现在不要让任何东西接近报警器，任何东西进入传感器的报警范围都将激活一个报警，直到重新复位。

代码回顾

本项目中许多代码与项目 39 完全一样，因此我将略过解释这些片段。

加载 LedControl 库：

```
#include "LedControl.h"
```

下面的代码定义所使用的引脚及芯片数量等，与上个项目相同：

```
#define sensorPin 9
#define switchPin 7
#define buttonPin 6
#define potPin 0
#define DataIn 2
#define CLK 4
#define LOAD 3
#define NumChips 1
#define samples 5.0
```

本项目增加了 buttonPin 和 potPin 的定义。这些变量声明中包含了一个名为 alarmRange 的新变量，它用来存储距离报警门限。如果有人进入报警范围之内，系统将发出报警声：

```
float pwmRange, averageReading, inch, cm, alarmRange;
```

生成一个叫做 lc 的 LedControl 对象并定义引脚：

```
LedControl lc=LedControl(DataIn,CLK,LOAD,NumChips);
```

setup()函数与前面的项目相同，增加了引脚模式设置，buttonPin 引脚模式为 INPUT：

```
lc.shutdown(0,false);
lc.setIntensity(0,8);
lc.clearDisplay(0);
```

```
pinMode(sensorPin, INPUT);
pinMode(switchPin, INPUT);
pinMode(buttonPin, INPUT);
```

主循环调用一个新的函数，叫做 `readPot()`。这个函数从变阻器中读数，用来调整报警范围（稍后讨论）：

```
readPot();
```

主循环中其他的代码与项目 39 相同：

```
averageReading = 0;
for (int i = 0; i<samples; i++) {
    pwmRange = pulseIn(sensorPin, HIGH);
    averageReading += pwmRange;
}
averageReading /= samples;
inch = averageReading / 147;
cm = inch * 2.54;

if (digitalRead(switchPin)) {
    displayDigit(inch);
}
else {
    displayDigit(cm);
}
```

直到另外一个 if 语句：

```
if (inch<=alarmRange) {startAlarm();}
```

它的作用是检查当前传感器的测量值是否小于或等于 `alarmRange` 中的值，如果是，调用 `startAlarm()` 函数，这个函数是用户自己设定的。

`displayDigit()` 函数与项目 39 中的完全一样：

```
void displayDigit(float value) {
    int number = value*100;
    lc.setDigit(0,4,number/10000,false); //百位数字
    lc.setDigit(0,3,(number%10000)/1000,false); //十位数字
```

```

    lc.setDigit(0,2,(number%1000)/100,true); //个位数字
    lc.setDigit(0,1,(number%100)/10,false); //十分位数字
    lc.setDigit(0,0,number%10,false); //百分位数字
}

```

接下来是两个新函数。第一个函数用来读取变阻器的值，把变阻器的值转化为以英寸为单位的值，用来设置报警范围。这个函数没有参数，但是返回值为浮点型，因此它要返回一个浮点值到 `alarmRange` 中。

```
float readPot()
```

之后，从 `potPin` 中读模拟值，并把它存储在 `potValue` 中：

```
float potValue = analogRead(potPin);
```

然后，对这个值做一个运算，把从变阻器中读出的 0~1023 之间的值转化成传感器测量的最大和最小作用范围，例如 0~254 英寸。

```
alarmRange = 254 * (potValue/1024);
```

之后把这个值返回到调用本函数的那一点：

```
return alarmRange;
```

下一个函数用于设置报警声音，也就是 `startAlarm()` 函数：

```
void startAlarm() {
```

之后是一个 `while` 循环，你在第 3 章中见到过 `while` 循环。当括号中的条件是 `true` 时，这个循环将运行花括号中的语句。这个 `while` 循环的参数是 1。这意味着当这个值是 `true` 时，这个循环将运行。在这个例子里要检测的值是个常数，因此这个循环总是运行，你要使用一个 `return` 语句退出这个循环。

```
while(1) {
```

接下来是一个 `for` 循环，它将扫描从 800Hz 到 2500Hz 之间的频率：

```
for (int freq=800; freq<2500;freq++) {
```

从引脚 8 上发出声响，这是嗡鸣器的声音，所要发出声音的频率存储在 `freq` 中：

```
tone(8, freq);
```

现在使用 `digitalRead` 函数检查 `buttonPin`，看这个按钮是否被按下：

```
if (digitalRead(buttonPin)) {
```

如果按钮已经被按下，运行花括号内的代码。它先使用 `noTone()` 函数停止报警声，之后退出这个函数，返回到主循环：

```
noTone(8);  
return;
```

在下一个项目中将使用同样的电路，但是上载不同的代码将这个传感器用于不同的目的。

项目 41——超声电子音乐

这个项目要使用同样的电路。但是本项目用不到变阻器、乒乓开关和复位按钮。但我仍要保留它们，你可以利用这些元件修改本项目，增加项目的灵活性。如果你之后想要再做一遍项目 40，你可以跳回到项目 40 中。

这次你要使用传感器生成泰勒明电子琴，这个项目使用的传感器不是真正的泰勒明电子琴使用的电子元件。如果你不知道什么是泰勒明电子琴，可以查查维基百科。它是一种电子乐器，用它演奏时不需要敲击，只需把你的手放入电场中再把手从电场中移走。设备感觉到电场的变化，演奏一个与手到线圈距离有关的声音。这很难解释清楚，你到 YouTube 上看看泰勒明电子琴演奏的视频就明白了。因为电路与前几个项目的电路相同，我直接跳过去解释代码。

输入代码

输入清单 14-4 中的代码。

清单 14-4 项目 41 的代码

```
//项目 41
```

```
#define sensorPin 9
```

```

#define lowerFreq 123 //C3
#define upperFreq 2093 //C7
#define playHeight 36

float pwmRange, inch, cm, note;

void setup() {
    pinMode(sensorPin, INPUT);
}

void loop() {
    pwmRange = pulseIn(sensorPin, HIGH);

    inch = pwmRange / 147;
    //转化英寸到厘米
    cm = inch * 2.54;

    //把距离映射到高频数和低频数之间的值作为声音频率
    note = map(inch, 0, playHeight, lowerFreq, upperFreq);
    if (inch<playHeight) {tone(8, note); }
    else {noTone(8);}
}

```

将代码上传到 **Arduino** 之后，你就可以把手放在传感器作用的范围内，它将根据手到传感器的距离演奏相应的声音。在传感器的作用范围内上下移动你的手，所演奏的声音将按比例地升高或降低。如果你希望改变声音高低转换的频率范围，可以调整相应的代码。

代码回顾

这个项目的代码是项目 40 的简化版本，其中有一些代码把传感器的测量范围转化成在嗡鸣器或扬声器上演奏的音调。程序开始时定义传感器的引脚：

```
#define sensorPin9
```

之后，定义给演奏的声调的高低及 **playHeight** 变量，**playHeight** 表示以英寸为单位的传感器检测到的作用范围，也就是你的手要在这个范围内乐器才会演奏。你可以根据需要调整这个范围。

```

#define lowerFreq 123 // C3
#define upperFreq 2093 // C7

```

```
#define playHeight 36
```

声明 **note** 变量，它表示通过扬声器演奏的音调：

```
float pwmRange, inch, cm, note;
```

setup 函数设置传感器的引脚为输入：

```
pinMode(sensorPin, INPUT);
```

在主循环中从传感器中读取数值并把它转化成英寸：

```
pwmRange = pulseIn(sensorPin, HIGH);
inch = pwmRange / 147;
```

之后，把从 0 到存储在 **playHeight** 中的英寸值映射到定义在程序开始处的高、低频率：

```
note = map(inch, 0, playHeight, lowerFreq, upperFreq);
```

你只是希望当你的手在传感器的作用范围内才演奏声音，因此检查传感器的所测到的值是否小于或等于作用范围。如果是，手一定是放到演奏区域里了，因此可以演奏出声音：

```
if (inch<playHeight) {tone(8, note); }
```

如果手不在作用范围内或从作用范围内移走，那么就不会有声音发出：

```
else {noTone(8);}
```

改变 **playHeight**、**upperFreq** 和 **LowerFreq** 值，以得到你想要的声音。

小结

在这章里，你学到了如何与一个超声传感器实现连接。我也介绍了一点传感器的使用知识，并应用于项目中，使你对传感器有了直观的认识。这样的传感器经常用在玩具机器人项目中，帮助机器人检测它是否接近了一面墙或者障碍物。它也被用在陀螺仪项目中，保证飞行器不会撞上一面墙或人群。另一个常用的用途是检测桶或罐中的液体的高度。我相信你还会找到这种传感器的其他用途。

本章的主题和概念

- 超声传感器是如何工作的
- 如何从 MaxSonar 元件中读取 PWM 的输出值
- 使用电容来平滑电源电压
- 如何使用 pulseIn 命令测量 pulse 宽度
- 超声测距仪的潜在应用
- 如何使用 Max7219 控制 7 段显示器
- 如何连接一个共阴极 7 段显示器
- 使用通过取平均值法来平滑读入数据
- 如何使用 setDigit()函数显示 7 段 LED
- 使用模除操作从常数中取出每一位数字
- 如何用 while()语句编写永久循环代码



第 15 章

读写 SD 卡

现在你要学一点从 SD 卡读写数据的基础知识。SD 卡体积小价格低，是比较好的存储数据的元件，而且 Arduino 可以相当容易地通过 SD 卡的 SPI 接口与其通信。你将学习如何建立一个文件、向一个已存在的文件中添加数据、给文件打上时间戳、写数据到文件等相关知识。这些知识使你可以用 SD 卡和 Arduino 作为数据记录设备来存储任何想要存储的数据。一旦你了解了这些技术，就可以用 SD 卡做一个带有时间戳的温度记录仪。

项目 42——简单的 SD 卡读写

本项目需要一个 SD 卡和一些连接 SD 卡到 Arduino 上的元件。最简单的办法是使用从各类电子元件供应商处买的 SD/MMC 接口卡。我使用一个 Sparkfun 的 SD/MMC 接口卡。

需要的元件

SD 卡接口板*



3×3.3kΩ电阻



3×1.8kΩ电阻



*感谢 Sparkfun 供图。

电阻起到分压的作用，它把 5V 逻辑电压降到 3.3V（注意，尽管使用电阻很容易，但是更安全的办法是使用逻辑电平转换器）

把元件连接起来

如图 15-1 所示那样连接每一件东西。

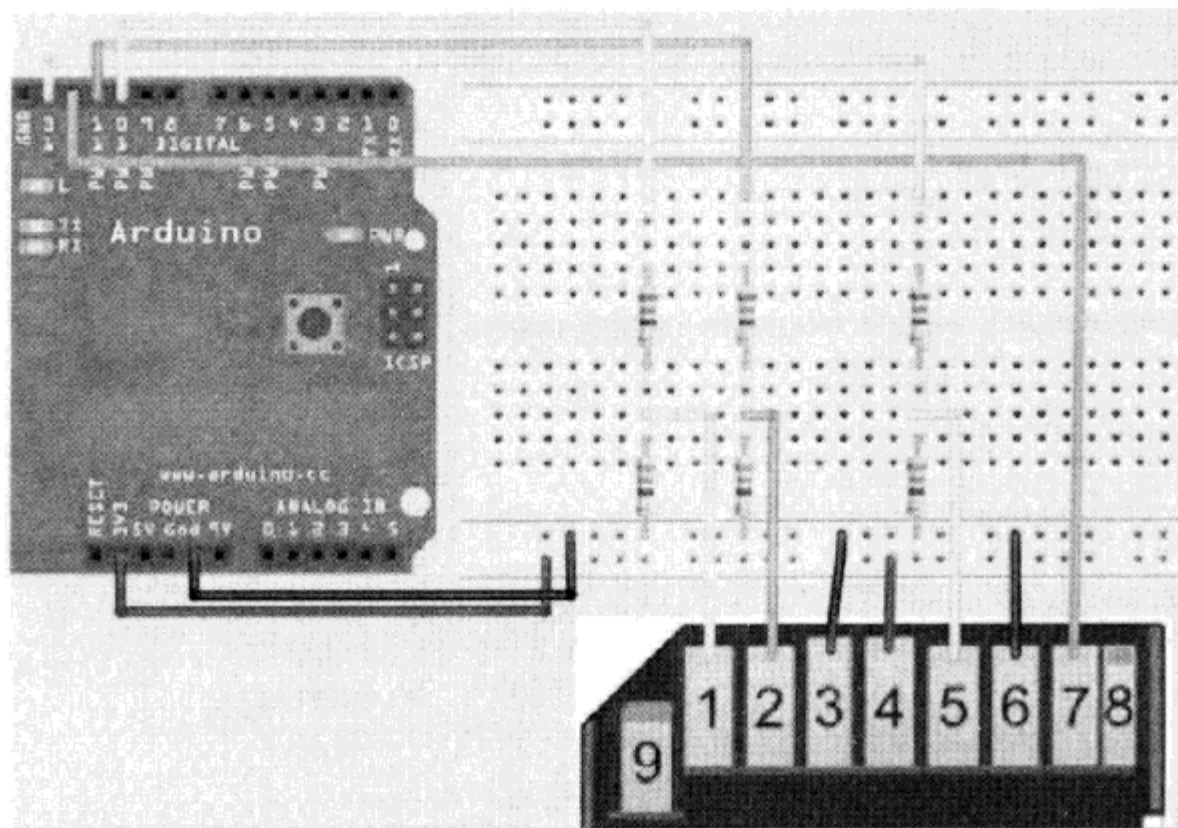


图 15-1 项目 42——简单的 SD 卡读写电路图（参见彩色版插图）

参考表 15-1 连接各元件的输出引脚，Arduino 上的数字引脚 12 直接连接到 SD 卡的引脚 7，Arduino 数字引脚 13、11 和 10 通过电阻分压至逻辑电平 3.3V 后连接到相应的 SD 卡引脚上。

表 15-1 Arduino 和 SD 卡之间的引脚连接

Arduino	SD 卡
+3.3V	Pin4 (VCC)
Gnd	Pin3&6 (GND)
Digital Pin 13 (SCK)	Pin5 (CLK)
Digital Pin 12 (MISO)	Pin7 (DO)
Digital Pin 11 (MOSI)	Pin2 (DI)
Digital Pin 10 (SS)	Pin1 (CS)

输入代码

首先你需要安装 Bill Breiman 写的 SdFat.h 和 SdFatUtil.h 库，两个库可以在 <http://code.google.com/p/sdfatlib/> 网址中找到，下载库，解压它，安装 sdat 文件夹到

你的 Arduino 库文件夹中。安装完链接库后，再检查连线是否正确，若正确，输入清单 15-1 中的代码并上传到 Arduino 中。

清单 15-1 项目 42 的代码

```
//项目 42
//sdfatlib 的例子, Bill Greiman 写的 SD 读写实例

#include <SdFat.h>
#include <SdFatUtil.h>

Sd2Card card;
SdVolume volume;
SdFile root;
SdFile file;

//在 Flash 存储器中存储错误字符串, 节省 RAM 空间
#define error(s) error_P(PSTR(s))

void error_P(const char* str) {
  PgmPrint("error: ");
  SerialPrintln_P(str);
  if (card.errorCode()) {
    PgmPrint("SD error: ");
    Serial.print(card.errorCode(), HEX);
    Serial.print(',');
    Serial.println(card.errorData(), HEX);
  }
  while(1);
}

//向文件写一个回车或换行符号
void writeCRLF(SdFile& f) {
  f.write((uint8_t*)"r\n", 2);
}

//写一个无符号数到文件
void writeNumber(SdFile& f, uint32_t n) {
  uint8_t buf[10];
  uint8_t i = 0;
  do {
```

```

        i++;
        buf[sizeof(buf) - i] = n%10 + '0';
        n /= 10;
    } while (n);
    f.write(&buf[sizeof(buf) - i], i);
}

//写字符串到文件
void writeString(SdFile& f, char *str) {
    uint8_t n;
    for (n = 0; str[n]; n++);
    f.write((uint8_t *)str, n);
}

void setup() {
    Serial.begin(9600);
    Serial.println();
    Serial.println("Type any character to start");
    while (!Serial.available());

    //初始化 SD 卡在一半的通信速度避免使用面包板造成总线错误
    //如果你的卡可以使用全速执行
    if (!card.init(SPI_HALF_SPEED)) error("card.init failed");

    //初始化 FAT 卷标
    if (!volume.init(&card)) error("volume.init failed");

    //打开根目录
    if (!root.openRoot(&volume)) error("openRoot failed");

    //生成一个新文件
    char name[] = "TESTFILE.TXT";

    file.open(&root, name, O_CREAT | O_EXCL | O_WRITE);
    //在这里放当天的日期时间
    file.timestamp(2, 2010, 12, 25, 12, 34, 56);

    //写 10 行数据到文件
    for (uint8_t i = 0; i < 10; i++) {
        writeString(file, "Line: ");
        writeNumber(file, i);
        writeString(file, " Write test.");
    }
}

```

```

    writeCRLF(file);
}

//关闭文件并强制写所有数据到SD卡
file.close();
Serial.println("File Created");

//打开文件
if (file.open(&root, name, O_READ)) {
    Serial.println(name);
}
else{
    error("file.open failed");
}
Serial.println();

int16_t character;
while ((character = file.read()) > 0) Serial.print((char)character);

Serial.println("\nDone");
}

void loop() { }

```

确保你的SD卡已经格式化为FAT格式。运行这个程序，打开串口监视器。你可以马上输入一个字符，然后按SEND按钮。这个程序要写一个文件到SD卡上，之后读回文件名和文件内容，并显示在串口监视器窗口中。如果每件事都做得很好。串口监视器窗口显示的东西如下：

```

Type any character to start
File Created
TESTFILE.TXT
    Line:0 Write test.
    Line:1 Write test.
    Line:2 Write test.
    Line:3 Write test.
    Line:4 Write test.
    Line:5 Write test.
    Line:6 Write test.
    Line:7 Write test.
    Line:8 Write test.

```



```
Line:9 Write test.
```

```
Done
```

要说明的是，SD 卡可能在你的 PC 或 Mac 计算机上工作得很好，但是在 Arduino 上可能不能工作。在我找到一个可用的（SD4/16Gb kingston）卡之前我已经试过了 6 个卡。因此你可能要自己试一下，其他用户报告成功使用了 SD 卡。

当这个程序运行结束后，把你的 SD 卡插入 PC 或 Mac 计算机，你会发现 SD 卡上有一个叫 TESTFILE.TXT 的文件，如果打开这个文件，文件内容是如上的文本。让我们看看代码是如何工作的。

代码回顾

程序开始于包含 sdfatlib 库中的两个库文件，保证代码能够工作：

```
#include <SdFat.h>
#include <SdFatUtil.h>
```

之后，你要生成 Sd2Card、SdVolume、SdFile 类的实例，并给它们起名字：

```
Sd2Card card;
SdVolume volume;
SdFile root;
SdFile file;
```

Sd2Card 类使你可以连接到标准的 SD 卡和 SDHC 卡。SdVolume 类支持 FAT16 和 FAT32 文件格式。Sdfile 类给出文件接口函数，如 open()、read()、rename()、write()、close()和 sync()，这个类给出到根目录和它的子目录的入口。

之后定义错误捕捉。定义 error(s)指向第一个函数叫 error_P:

```
#define error(s) error_P(PSTR(s))
```

生成一个函数，叫做 error_P，这个函数的目的只是打印出错信息，需要传递给函数的参数是相关的错误代码，这个参数是一个字符串。字符串定义之前有一个 const 字符，这叫做变量修饰符，它改变变量的行为。在这个例子里，const 修饰符使得变量为只读的。这样定义之后的变量可以作为一般变量使用，但是值不能改变。

```
void error_P(const char* str) {
```

之后是 `PgmPrint()` 函数，这是 `sdfatlib` 库中的函数，它存储括号内的字符串到 SD 卡。

```
PgmPrint("error: ");
```

之后是 `SerialPrintln_P` 函数，这个函数也是来自库的，作用是打印在 SD 卡中的字符串到串口，跟随着 CR/LF（回车/换行）字符。

```
SerialPrintln_P(str);
```

之后，使用 `errorCode()` 函数检查是否有错误。库文档有一个错误代码的列表。

```
if (card.errorCode()) {
```

如果产生了一个错误，花括号内的代码被执行，显示错误代码和错误数据：

```
PgmPrint("SD error: ");
Serial.print(card.errorCode(), HEX);
Serial.print(',');
Serial.println(card.errorData(), HEX);
```

最后，如果产生了一个错误，`while(1)` 语句进行无限循环，停止做其他任何事情：

```
while(1);
```

下一个函数叫做 `writeCRLF()`，作用是写一个回车或换行符号到文件，要给这个函数传递一个文件指针作为参数。

```
void writeCRLF(SdFile& f) {
```

大括号中的代码使用 `write()` 函数去写两个字节到文件，这里有 `\r` 和 `\n`，即回车和换行符号。`write()` 函数有时叫做多态函数，它的意思是一个函数被定义几次以接受不同数据类型的参数。这里函数的参数是 `uint8`，告诉函数你希望调用无符号 8 位整型版本的 `write()` 函数。

```
f.write((uint8_t*)"\r\n", 2);
```

下一个函数用来写数到文件，它接受一个文件指针和一个无符号 32 比特整数为参数。

```
void writeNumber(SdFile& f, uint32_t n) {
```

定义一个有 10 个元素的数组，它的数据类型是无符号 8 位整数；定义一个叫 i 的无符号 8 位整数变量，将它初始化为 0：

```
uint8_t buf[10];
uint8_t i = 0;
```

之后是一个 do-while 循环，它的作用是把一个整型数转化为字符串，每次循环转化一个数字：

```
do {
    i++;
    buf[sizeof(buf) - i] = n%10 + '0';
    n /= 10;
} while (n);
```

do-while 循环是你在之前还没接触到的东西。它的作用与 while 循环相似，但是 do-while 循环在循环的结尾处检查条件而不是在开始处检查条件。如果开始处的条件没有达到 while 循环将不会运行，而 do-while 循环总是要运行括号中的代码至少一次，如果条件符合，循环将重复。

循环中把 i 增加 1，之后使用 sizeof 函数获得数组的大小，它返回数组中的字节数，在这里是 10。i 开始时是 1，因此循环将首先通过 10-1 或数组的第 8 个元素，也就是数组的最后一个元素。之后它将变成 10-2 或第 8 个元素，依此类推，从右向左工作。之后，存储 n%10 的结果，这将是每个数字最右侧的数字。之后，用 10 除以 n，结果使 n 中的数字向右侧移动了。这个循环重复，可以起到获得整数 n 最右侧每一个数字的作用，并存储它到字符数组最后一个元素中，丢掉最右侧的数字后，再次重复。这样，这个数字被拆分成单独的数字之后存储在字符数组中成为一个字符串。最后在数字尾部加 0 字符把数字转化成相应的 ASCII 码。

```
buf[sizeof(buf) - i] = n%10 + '0';
```

这有把 ASCII 码把数字 '0' 值增加 n%10 后送入字符数组的作用。换句话说，如果数

字是 123，那么 $123\%10=3$ ，“0”的 ASCII 码是 48 增加 3 是 51，这恰恰是“3”的 ASCII 码。这样，这个数字转化成它相应的 ASCII 码。循环退出后使用以下代码把 buf[] 数组中的内容写到文件中：

```
f.write(&buf[sizeof(buf) - i], i);
```

该函数的参数是一个指针，表示要写的字符在字符数组中的位置。跟着是要写的字符数组的字节数。还有另外一个版本的 write() 函数。之后到达一个函数，作用是写一个字符串到文件，参数是一个文件指针和一个字符串指针。

```
void writeString(SdFile& f, char *str) {
```

这一次你还是用 for 循环查看字符串的每一个元素，一个字符一个字符地查看并写到文件中。

```
uint8_t n;
for (n = 0; str[n]; n++);
f.write((uint8_t *)str, n);
```

之后到达 setup 函数，它做了所有的事情，因为你只需要程序运行一次，因此全部代码都在 setup() 中，loop() 循环什么也不做。

接着开始初始化串口通信，之后提示使用者输入字符开始运行程序：

```
Serial.begin(9600);
Serial.println();
Serial.println("Type any character to start");
```

现在程序等待，直到一些东西输入串口监视器。如果在串口监视器中没有输入任何东西，使用一个空的 while 循环不做任何事情。

```
while (!Serial.available());
```

之后，如果在初始化卡、卷标或打开根目录的时候有任何错误发生，那么运行 3 个 if 语句进行错误处理：

```
if (!card.init(SPI_HALF_SPEED)) error("card.init failed");
if (!volume.init(&card)) error("volume.init failed");
```

```
if (!root.openRoot(&volume)) error("openRoot failed");
```

你可以改变 `SPI_FULL_SPEED` 定义的通信速度，如果你的卡可以接受的话。当我试图把我用的卡运行在最高速度时发生了错误，因此我让它半速运行。你也许能在高速运行时取得成功。

现在需要给生成的新文件定义一个名称，因此把文件名放入一个字符数组内：

```
char name[] = "TESTFILE.TXT";
```

之后，打开一个在根目录下的文件：

```
file.open(&root, name, O_CREAT | O_EXCL | O_WRITE);
```

要打开的是名字存储在 `name` 中的文件，这就是你初始化为 `TESTFILE.TXT` 的文件。因为这个文件不存在，函数同时创建了具有同样名字的文件。在函数中有三个标志，标志表明要做什么，三个标志分别是 `O_CREAT`、`O_EXCL` 和 `O_WRITE`。

`O_CREAT` 标志告诉 `open` 函数如果文件不存在，生成文件。`O_EXCL` 标志使 `O_CREAT` 无效（因为文件是排他性的，因此如果文件已经存在，不要再生成同样的文件）。`O_WRITE` 使文件打开做写的准备。因此，这个函数将打开文件，如果文件还不存在会生成文件，如果这个文件已经存在要确保文件不会被重写，最后打开这个文件使它做好写的准备。

之后，使用时间戳函数确认文件生成的日期和时间。这个函数接收 7 个参数，它们是标志、年、月、日、小时、分钟和秒。

```
file.timestamp(2, 2010, 12, 25, 12, 34, 56);
```

在这个例子里，你传递给函数的是不一定准的数据，但是在理想状态下，你可以从一个时钟源如 `RTC`（真实时钟）芯片或一个 `GPS` 模块获得时间，之后用这个数据作为文件的时间戳。组成第一个参数的标志如下：

```
T_ACCESS = 1
T_CREATE = 2
T_WRITE = 4
```

- `T_ACCESS`——设置文件最后访问的时间。

- `T_CREATE`——设置文件生成的时间和日期。
- `T_WRITE`——设置文件最后写/修改的时间和日期。

在你的例子里用的是 2，它的意思是设置文件生成日期和时间。如果你想设置全部的 3 个时间，值应该是 7 (4+2+1)。

之后，运行一个 for 循环程序块，循环 10 次，写数字和一些检测数据到文件，跟着是一个回车或换行符号。调用这三个函数来完成写数字、字符串和换行符。

```
for (uint8_t i = 0; i < 10; i++) {
  writeString(file, "Line: ");
  writeNumber(file, i);
  writeString(file, " Write test.");
  writeCRLF(file);
}
```

任何对文件执行的操作都不会被真正执行，直到文件关闭时才会一起执行。使用 `close()` 函数关闭文件，它将关闭文件并写你在代码中生成的全部数据。

```
file.close();
```

之后，让使用者知道文件已经生成：

```
Serial.println(File.Created);
```

现在，你已经生成一个新的文件并且写了数据到它中。接下来是打开 SD 卡并从中读取数据的程序。使用 `open()` 函数，它需要 3 个参数：文件所在的目录、文件名、打开文件的标识。使用 `&root` 表明文件在根目录下。标识是 `O_READ` 表示用读方式打开一个文件。这个命令是 if 语句的一个条件，如果文件成功打开可以打印文件名，如果文件没有成功打开运行错误处理程序。

```
if (file.open(&root, name, O_READ)) {
  Serial.println(name);
}
else{
  error("file.open failed");
}
Serial.println();
```


之后，使用 `while` 循环从文件中一次读出一个字符，并且打印结果到串口监视器。

```
int16_t character;
while ((character = file.read()) > 0) Serial.print((char)character);
```

最后如果全部成功，打印一个 “Done”：

```
Serial.println("\nDone");
```

主循环根本就不包含代码。你只是想运行这个程序一次，因此把全部语句放在 `setup()` 函数中，没有任何代码在 `loop` 中。`Setup()` 函数只执行一次。之后执行 `loop`，因为 `loop` 中没有包含任何代码，所以程序执行一次后什么也不会再发生，直到 `Arduino` 掉电或复位。

```
void loop() {}
```

以上的例子表明了生成一个文件、写数字和字符串到文件、关闭文件、读文件的基本方法。下面将延伸这些知识把它用于实践，使用 `SD` 卡记录一些传感器数据。

项目 43——用 `SD` 卡记录温度数据

现在你要增加 `DS18B20` 温度传感器和一个 `DS1307RTC`（实时时钟）芯片到电路中。从温度传感器中读出要写入 `SD` 卡中的数据，读 `RTC` 芯片内的实时时间数据，用这个实时时间数据作为读传感器和更改文件的时间戳。

需要的元件

`SD` 卡和接口*



3 个 `3.3kΩ` 电阻



3 个 `1.8 kΩ` 电阻



`4.7 kΩ` 电阻



2 个 `1 kΩ` 电阻



DS1307TTC 芯片



32.768kHz 12.5pF 晶振



2 个 SD18B20 温度传感器



*电池固定器**



*感谢 Sparkfun 供图。

**其他可选。

电池固定器是不错的选择，有一个电池作为后备电源，使得即使在项目断电的时候也会保持 TRC 芯片中的时间和日期数据。

把元件连接起来

如图 15-2 所示连接每一个东西。

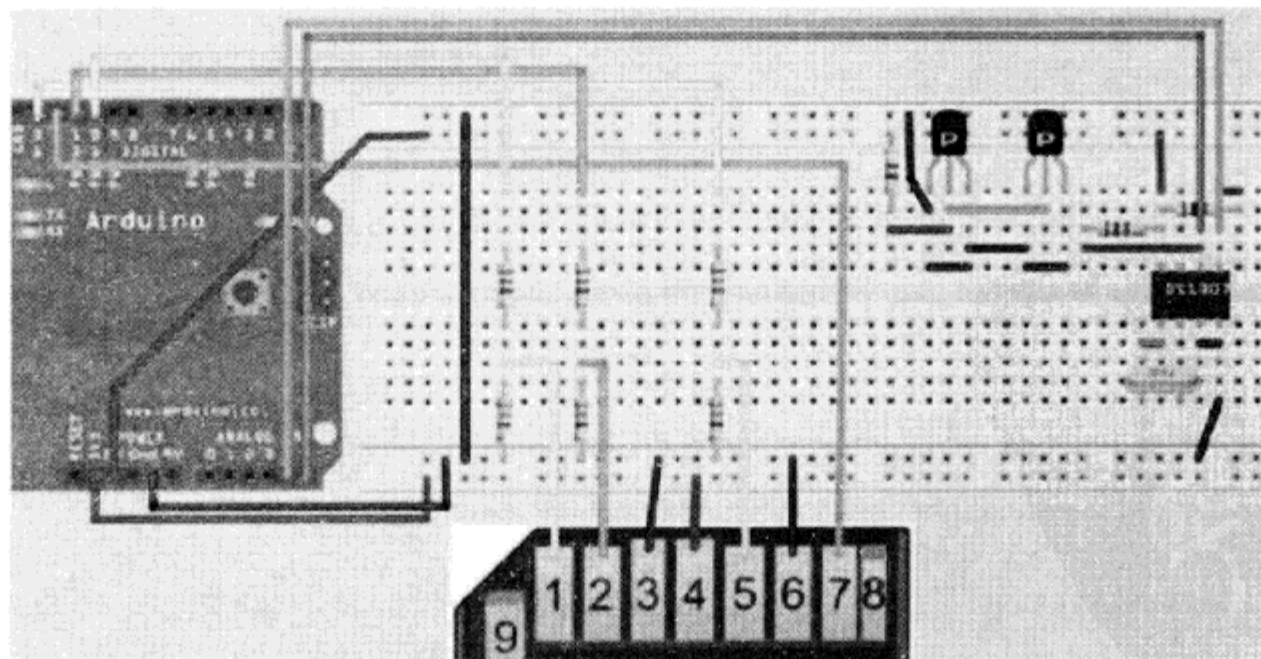


图 15-2 项目 43——用 SD 卡记录温度数据电路图（参见彩色版插图）

参考表 15-2 检查连接是否正确，与项目 37 完全相同地连接 DS18B20，如果你用电池盒作为后备电池，不要连接图上 RTC 引脚 3 和 4 的线，而是连接到电池的正极到芯片的引

脚 3，负极到芯片的引脚 4。

表 15-2 Arduino、SD 卡和 RTC 之间的引脚连接

Arduino	SD 卡	RTC
+5V		Pin 5
+3.3V	Pin 4 (VCC)	
Gnd	Pin 3&6 (GND)	Pin 3 & 4
Digital Pin 13 (SCK)	Pin 5 (CLK)	
Digital Pin 12 (MISO)	Pin 7 (DO)	
Digital Pin 11 (MOSI)	Pin 2 (DI)	
Digital Pin 10 (SS)	Pin 1 (CS)	
Analog Pin 4		Pin 5
Analog Pin 5		Pin 6

连接 1K Ω 电阻到 RTC 上的引脚 8、引脚 5 和引脚 6。

输入代码

确保在项目 37 之中使用的 OneWire.h 库和 DallasTemperature.h 库安装在这个项目中。你还要使用 Matt Joyce 和 D.Sjunnesson 写的控制 SD1307 芯片的 DS1307.h 库。

清单 15-2 项目 43 的代码

```
//项目 43
//sdfatlib 的例子, Bill Greiman 写的 SD 读写实例
//Matt Joyce 写的 DS1307 库, D.Sjunnesson 进一步改写

#include <SdFat.h>
#include <SdFatUtil.h>
#include <OneWire.h>
#include <DallasTemperature.h>
#include <WProgram.h>
#include <Wire.h>
#include <DS1307.h> //mathtt 在 Arduino 论坛上写的库, D.Sjunnesson 改进

//在 Flash 存储器中存储错误字符串, 节省 RAM 空间
#define error(s) error_P(PSTR(s))
//连接 Arduino 引脚 3 的数据线
#define ONE_WIRE_BUS 3
#define TEMPERATURE_PRECISION 12
```

```

Sd2Card card;
SdVolume volume;
SdFile root;
SdFile file;

//建立一个 oneWire 实例, 连接任何 OneWire 元件 (不仅仅是 Maxim/Dallas 温度传感器)
OneWire oneWire(ONE_WIRE_BUS);
//传递 oneWire 引用给温度传感器实例
DallasTemperature sensors(&oneWire);

//存储温度传感器地址的数组
DeviceAddress insideThermometer = { 0x10, 0x20, 0x2C, 0xA9, 0x01, 0x08, 0x00,
0x73 };
DeviceAddress outsideThermometer = { 0x10, 0x22, 0x5B, 0xA9, 0x01, 0x08, 0x00,
0x21 };

float tempC, tempF;
int hour, minute, seconds, day, month, year;

//生成新文件的文件名
char name[] = "TEMPLOG.TXT";

void error_P(const char* str) {
  PgmPrint("error: ");
  SerialPrintln_P(str);
  if (card.errorCode()) {
    PgmPrint("SD error: ");
    Serial.print(card.errorCode(), HEX);
    Serial.print(',');
    Serial.println(card.errorData(), HEX);
  }
  while(1);
}

void writeCRLF(SdFile& f) {
  f.write((uint8_t*)"r\n", 2);
}

//写无符号数到文件
void writeNumber(SdFile& f, uint32_t n) {
  uint8_t buf[10];

```

```

uint8_t i = 0;
do {
    i++;
    buf[sizeof(buf) - i] = n%10 + '0';
    n /= 10;
} while (n);
f.write(&buf[sizeof(buf) - i], i);
}

//写字符串到文件
void writeString(SdFile& f, char *str) {
    uint8_t n;
    for (n = 0; str[n]; n++);
    f.write((uint8_t *)str, n);
}

void getTemperature(DeviceAddress deviceAddress)
{
    sensors.requestTemperatures();
    tempC = sensors.getTempC(deviceAddress);
    tempF = DallasTemperature::toFahrenheit(tempC);
}

void getTimeDate() {
    hour = RTC.get(DS1307_HR,true); //读小时数据, 第二个参数为 true, 更新所有数据
    minute = RTC.get(DS1307_MIN,false); //读分钟数据, 不更新数据
    seconds = RTC.get(DS1307_SEC,false); //读秒数据
    day = RTC.get(DS1307_DATE,false); //读日期
    month = RTC.get(DS1307_MTH,false); //读月份
    year = RTC.get(DS1307_YR,false); //读年份
}

void setup() {
    Serial.begin(9600);
    Serial.println("Type any character to start");
    while (!Serial.available());
    Serial.println();

    //开始使用传感器库函数
    sensors.begin();
    Serial.println("Initialising Sensors.");
}

```

```

//设置精度
sensors.setResolution(insideThermometer, TEMPERATURE_PRECISION);
sensors.setResolution(outsideThermometer, TEMPERATURE_PRECISION);
delay(100);

//设置 RTC 中的时间
//注释掉这部分, 如果已经设置了时间和有一个备份电池
RTC.stop();
RTC.set(DS1307_SEC, 0);      //设置秒
RTC.set(DS1307_MIN, 15);     //设置分钟
RTC.set(DS1307_HR, 14);      //设置小时
RTC.set(DS1307_DOW, 7);      //设置星期
RTC.set(DS1307_DATE, 3);     //设置日期
RTC.set(DS1307_MTH, 10);     //设置月份
RTC.set(DS1307_YR, 10);      //设置年份
RTC.start();

Serial.println("Initialising SD Card...");

//初始化 SD 卡运行在 SPI_HALF_SPEED 下避免电路错误
//使用 SPI_FULL_SPEED 优化执行结果, 如果卡可以承受
if (!card.init(SPI_HALF_SPEED)) error("card.init failed");

//初始化 FAT 卷标
if (!volume.init(&card)) error("volume.init failed");

//打开根目录
if (!root.openRoot(&volume)) error("openRoot failed");
Serial.println("SD Card initialised successfully.");
Serial.println();
}

void loop() {

    Serial.println("File Opened.");
    file.open(&root, name, O_CREAT | O_APPEND | O_WRITE);
    getTimeDate();
    file.timestamp(7, year, month, day, hour, minute, seconds);

    getTemperature(insideThermometer);
    Serial.print("Inside: ");
    Serial.print(tempC);
    Serial.print(" C ");
}

```



```

Serial.print(tempF);
Serial.println(" F");
writeNumber(file, year);
writeString(file, "/");
writeNumber(file, month);
writeString(file, "/");
writeNumber(file, day);
writeString(file, " ");
writeNumber(file, hour);
writeString(file, ":");
writeNumber(file, minute);
writeString(file, ":");
writeNumber(file, seconds);
writeCRLF(file);
writeString(file, "Internal Sensor: ");
writeNumber(file, tempC);
writeString(file, " C ");
writeNumber(file, tempF);
writeString(file, " F");
writeCRLF(file);

getTemperature(outsideThermometer);
Serial.print("Outside: ");
Serial.print(tempC);
Serial.print(" C ");
Serial.print(tempF);
Serial.println(" F");
writeString(file, "External Sensor: ");
writeNumber(file, tempC);
writeString(file, " C ");
writeNumber(file, tempF);
writeString(file, " F");
writeCRLF(file);
writeCRLF(file);

Serial.println("Data written.");
//关闭文件强制写所有的数据到 SD 卡
file.close();
Serial.println("File Closed.");
Serial.println();
delay(10000);
}

```

打开串口监视器，提示你输入一个字符。之后，在串口监视器上显示如下的输出。

```
Type any character to start
```

```
Initialising Sensors.  
Initialising SD Card...  
SD Card initialised successfully.
```

```
File Opened.  
Inside:27.25 C 81.05 F  
Outside:15.19 C 59.34 F  
Data writen.  
File Closed.
```

```
File Opened.  
Inside:28.31 C 82.96 F  
Outside:15.25 C 59.45 F  
Data writen.  
File Closed.
```

```
File Opened.  
Inside:28.62 C 83.52 F  
Outside:15.31 C 59.56 F  
Data written.  
File Closed.
```

如果给 Arduino 断电并拔出 SD 卡，之后把 SD 卡插入你的 PC 或 Mac 计算机上，将看到一个叫 TEMPLOG.TXT 的文件，用文件编辑器打开这个文件，会看到文件的内容是从传感器中读到的数据及时间：

```
2010/10/3 17: 29: 9  
Internal Sensor: 27 C 81 F  
External Sensor: 15 C 59 F
```

```
2010/10/3 17: 29: 21  
Internal Sensor: 28 C 82 F  
External Sensor: 15 C 59 F
```

```
2010/10/3 17: 29: 32  
Internal Sensor: 28 C 83 F  
External Sensor: 15 C 59 F
```

让我们看一下程序是如何工作的。

代码回顾

像项目 37 和 42 中的许多代码一样，我主要关心代码中新增加的部分。

首先包含如下的库：

```
#include <SdFat.h>
#include <SdFatUtil.h>
#include <OneWire.h>
#include <DallasTemperature.h>
#include <WProgram.h>
#include <Wire.h>
#include <DS1307.h>
```

最后 3 个库是运行 DS1307 需要的库，WProgram.h 和 Wire.h 库是 Arduino 程序的核心部分。

定义错误捕获函数并生成一个单线总线：

```
#define error(s) error_P(PSTR(s))
#define ONE_WIRE_BUS 3
#define TEMPERATURE_PRECISION 12
```

生成 SD 卡实例：

```
Sd2Card card;
SdVolume volume;
SdFile root;
SdFile file;
```

生成单线和 Dallas 温度传感器实例：

```
OneWire oneWire(ONE_WIRE_BUS);
DallasTemperature sensors(&oneWire);
```

之后定义 DS18B20 传感器地址。使用项目 37 的第一部分代码，找到传感器的地址。

```
DeviceAddress insideThermometer = { 0x10, 0x20, 0x2C, 0xA9, 0x01, 0x08, 0x00,
0x73 };
```

```
DeviceAddress outsideThermometer = { 0x10, 0x22, 0x5B, 0xA9, 0x01, 0x08, 0x00,
0x21 };
```

你现在定义一些变量来存储温度读数和从 RTC 中读来的时间数据。定义文件名数组，这个文件就是你想要存储数据的文件。

```
float tempC, tempF;
int hour, minute, seconds, day, month, year;
char name[] = "TEMPLOG.TXT";
```

之后到错误捕获函数，写字符 CR 和 LF 到文件，写数字和字符串。

```
void error_P(const char* str) {
    PgmPrint("error: ");
    SerialPrintln_P(str);
    if (card.errorCode()) {
        PgmPrint("SD error: ");
        Serial.print(card.errorCode(), HEX);
        Serial.print(',');
        Serial.println(card.errorData(), HEX);
    }
    while(1);
}
```

```
void writeCRLF(SdFile& f) {
    f.write((uint8_t*)"r\n", 2);
}
```

//写一个无符号数到文件

```
void writeNumber(SdFile& f, uint32_t n) {
    uint8_t buf[10];
    uint8_t i = 0;
    do {
        i++;
        buf[sizeof(buf) - i] = n%10 + '0';
        n /= 10;
    } while (n);
    f.write(&buf[sizeof(buf) - i], i);
}
```

//写一个字符串到文件

```
void writeString(SdFile& f, char *str) {
    uint8_t n;
```

```
    for (n = 0; str[n]; n++);
    f.write((uint8_t *)str, n);
}
```

现在定义一个新函数从传感器中获得温度。芯片的地址作为一个参数传递给这个函数。

```
void getTemperature(DeviceAddress deviceAddress)
{
    sensors.requestTemperatures();
    tempC = sensors.getTempC(deviceAddress);
    tempF = DallasTemperature::toFahrenheit(tempC);
}
```

之后，定义另外一个新函数从实时时钟芯片 DS1307 中获得时间和日期，这个函数叫做 `getTimeDate()`。

```
void getTimeDate() {
```

为了从 TRC 中获得小时、分钟、秒、日期、月、年数据，使用 `get()` 函数，首先从芯片中获得小时数据，把它存储在 `hour` 变量中。

```
hour = RTC.get(DS1307_HR,true); //读小时数据并且用 true 参数更新全部值
```

这个函数需要两个参数。第一个是希望获得哪一个数据的状态标志。参数的名称显而易见地表明了它们的功能。第二个是 `false` 或 `true`。如果是 `true`，芯片中的数据（DS1307HR、DS1307YR 等）更新到当前时间。如果是 `false`，只是读最后更新的时间。因为只想在读开始时间日期时更新一次芯片中存储的数据，所以用一个 `true` 做参数。随后的 `get()` 函数用 `false` 做参数。

```
minute = RTC.get(DS1307_MIN,false); //读分钟不需更新 (false)
seconds = RTC.get(DS1307_SEC,false); //读秒
day = RTC.get(DS1307_DATE,false); //读日期
month = RTC.get(DS1307_MTH,false); //读月份
year = RTC.get(DS1307_YR,false); //读年份
```

之后到 `setup` 函数：

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

```
Serial.println("Type any character to start");
while (!Serial.available());
Serial.println();
```

初始化 DS18B20 传感器并设置传感器的精度:

```
sensors.begin();
Serial.println("Initialising Sensors.");
sensors.setResolution(insideThermometer, TEMPERATURE_PRECISION);
sensors.setResolution(outsideThermometer, TEMPERATURE_PRECISION);
```

之后,到达设置 RTC 中的日期和时间代码。使用.set()函数,它实际上是与.get()函数相反的。在设置芯片前,首先需要使用.stop()函数来停止 RTC 芯片运行。一旦设置了时间,使用.start()命令开始用新的时间和日期运行 RTC 芯片。

```
RTC.stop();
RTC.set(DS1307_SEC,0);           //设置秒
RTC.set(DS1307_MIN,15);          //设置分钟
RTC.set(DS1307_HR,14);           //设置小时
RTC.set(DS1307_DOW,7);           //设置星期
RTC.set(DS1307_DATE,3);          //设置日期
RTC.set(DS1307_MTH,10);          //设置月份
RTC.set(DS1307_YR,10);           //设置年
RTC.start();
```

初始化 SD 卡:

```
Serial.println("Initialising SD Card...");
//初始化 SD 卡运行在 SPI_HALF_SPEED 下避免电路错误
//使用 SPI_FULL_SPEED 优化执行结果,如果卡可以承受的话
if (!card.init(SPI_HALF_SPEED)) error("card.init failed");

//初始化 FAT 卷标
if (!volume.init(&card)) error("volume.init failed");

//打开根目录
if (!root.openRoot(&volume)) error("openRoot failed");
Serial.println("SD Card initialised successfully.");
Serial.println();
```

之后到主循环,这次使用 O_APPEND 标志打开文件。


```
file.open(&root, name, O_CREAT | O_APPEND | O_WRITE);
```

调用 `getTimeDate()` 函数从 RTC 中获得时间和日期:

```
getTimeDate();
```

这些数值用于文件的时间戳:

```
file.timestamp(7, year, month, day, hour, minute, seconds);
```

调用 `getTemperature()` 函数从第一个温度传感器芯片获得温度值:

```
getTemperature(insideThermometer);
```

温度值打印在串口监视器上, 之后是时间戳, 并把温度值写在文件中:

```
Serial.print("Inside: ");  
Serial.print(tempC);  
Serial.print(" C ");  
Serial.print(tempF);  
Serial.println(" F");  
writeNumber(file, year);  
writeString(file, "/");  
writeNumber(file, month);  
writeString(file, "/");  
writeNumber(file, day);  
writeString(file, " ");  
writeNumber(file, hour);  
writeString(file, ":");  
writeNumber(file, minute);  
writeString(file, ":");  
writeNumber(file, seconds);  
writeCRLF(file);  
writeString(file, "Internal Sensor: ");  
writeNumber(file, tempC);  
writeString(file, " C ");  
writeNumber(file, tempF);  
writeString(file, " F");  
writeCRLF(file);
```

从第二个温度传感器芯片获得温度并做同样的事情:

```

getTemperature(outsideThermometer);
Serial.print("Outside: ");
Serial.print(tempC);
Serial.print(" C ");
Serial.print(tempF);
Serial.println(" F");
writeString(file, "External Sensor: ");
writeNumber(file, tempC);
writeString(file, " C ");
writeNumber(file, tempF);
writeString(file, " F");
writeCRLF(file);
writeCRLF(file);

```

最后，关闭文件，通知用户完成，在下一次读取数据前延时 10 秒：

```

Serial.println("Data written.");
//关闭文件强制写所有的数据到SD卡
file.close();
Serial.println("File Closed.");
Serial.println();
delay(10000);

```

现在你已经有了如何写传感器数据或其他数据到一个SD卡的基本概念。对于更进一步的函数，阅读来自SDfat库的文档。下面在进入下一章之前快速阅读一下RTC芯片的资料。

硬件回顾

在项目43中使用了一个新的IC——DS1307实时时钟芯片。这是一个小芯片，使你可以轻松地增加时钟到项目中，增加一个电池做后备电源，可以在Arduino掉电时芯片自动转换到后备电源保证它的时间日期使用电池更新。如果使用一个好的晶振，芯片可以保持一个相当好的时间精度。芯片甚至可以自己调节闰年和少于31天的月份。它可以被设置成24小时或12小时模式。可以通过I²C接口写这个芯片，我稍后再解释。

有意思的是芯片的引脚7上有一个方波输出，它可以是1Hz、4.096kHz、8.19kHz或32.768kHz，因此可以用它产生声音或用在其他需要脉冲的地方。如果引脚脉冲设置成1Hz，可以非常容易地增加一个LED给这个引脚来指示过去的秒数。

与这个芯片通信是通过 I²C 总线实现的，你已经见过 one-wire 和 SPI 总线，但是这是一个新的总线，为了使用 I²C 总线，你需要在你的代码中包含 Wire.h 库。

I²C 总线(或 Inter-I 总线)(有时也叫 TWI 或 Two Wire 接口)是由 Philips Semiconductors (现在叫 NXP) 开发的一个简单的片内总线，用来连接两个 inter-IC 元件。这个总线只用两根线：串行数据线和串行时钟线。在 Arduino 中，这两个总线引脚是模拟引脚 4 (SDA) 和模拟引脚 5 (SCL)，外部仅需要芯片在每一条总线上放置一个上拉电阻。可以连接 128 个 I²C 芯片(或节点)在这两个总线上。一些 I²C 芯片使用+5V 也有使用+3.3V 的，因此在使用 I²C 芯片时要考虑各元件的电压等级。在把 I²C 芯片连接起来时你要阅读数据说明书以确保使用正确的电压。如果想在两个 Arduino 之间互相通信，采用 I²C 总线是一个好的选择。

I²C 总线与 SPI 总线相似，因为它也有主从芯片之分。Arduino 是主元件，I²C 元件是从元件，每个元件有它自己的 I²C 地址，每个时钟脉冲送一位数据。当主芯片在总线上要求开始时通信开始，当主芯片要求停止时通信结束。

要开始在 Arduino 上进行 I²C 通信，要用 Wire.begin()函数。它将初始化总线，并且 Arduino 作为主 I²C 元件。它也重新定义模拟引脚 4 和 5 为 I²C 总线引脚。要开始进行芯片间通信(如两个 Arduino 之间通过 I²C 连接通信)，从元件的地址必须包含在参数中，换句话说：

```
Wire.begin(5)
```

将使 Arduino 连到 I²C 总线上地址是 5 的从元件。使用以下函数可以接收从 I²C 元件中传送的一个字节的数据：

```
Int x=Wire.Receive();
```

在接收数据前，你必须使用如下函数确定通信的字节数：

```
Wire.requestForm (address,quantity)
```

因此

```
Wire.requestForm(5,10);
```

它将从设备 5 返回 10 字节。送出数据到元件很简单，只是用

```
Wire.beginTransaction(5);
```

它将送数据到元件 5，之后

```
Wire.send(x);
```

送出 1 字节，或

```
Wire.send("Wire test.");
```

送出 10 字节。

你可以在 www.arduino.cc/en/Reference/Wire 上学到更多关于总线链接库和 I²C 总线的知识，也可以通过阅读 Atmel 网站上的 I²C 扩展资料来了解更多的相关信息。

小结

在第 15 章中你已经学习了读写 SD 卡的基础知识。通过阅读 SDFat 库的随机文档可以了解更多关于使用 SD 卡的信息。在这个项目里你只是学到了 SDFat 库的一些基本知识。接下来介绍了 I²C 协议。随后又介绍了如何连接 DS1307 实时时钟芯片。今后如果你要做自己的基于时钟的项目，那么实时时钟芯片是非常有用的。另外一个获得精确时间信号的方法是使用便宜的用串口输出的 GPS 元件。

如何读和写 SD 卡是做数据记录的重要知识，特别是采用电池供电的项目。许多基于 Arduino 或 AVR 芯片的高度球项目中使用 SD 卡记录 GPS 和传感器数据，当气球返回地面时再获得这些数据。

本章的主题和概念

- 如何连接 SD 卡到 Arduino
- 使用电阻分压电路降低电压等级，电压从 5V 降到 3.3V
- 如何使用 SDFat 库
- 写字符串和数字到文件
- 打开和关闭文件

- 给文件命名
- 生成文件时间戳
- 捕获文件错误
- do-while 循环的概念
- 使用模除从一个长数中分出单个数字
- 如何连接 DS1307 实时时钟芯片到 Arduino
- 如何使用 DS1307.h 库设置实时时钟芯片的时间和日期
- 当系统掉电时使用后备电池保持 DS1307 上的数据
- I²C 协议的介绍



第 16 章

RFID 读卡器

RFID（无线射频识别）卡越来越流行了，它们可以用做进入办公室的门卡或公共交通付费卡。小的 RFID 标签可以注入动物体内，如果动物丢失了可用动物体内的 RFID 标签来识别它们。在交通工具上可以采用 RFID 标签来实现收费。RFID 甚至用在医院手术室给一次性器件做标签，保证手术后没有外物留在人体内。这些年这项技术的价格在大幅下降。目前，读者可以用不到 10 美元获得 RFID 读卡器，它们很容易与 Arduino 连接使用，所以许多有意思的项目可以用 RFID 去做。

下面你将使用容易得到而且很便宜的 Innovations 公司的 ID-12 读卡芯片。这种读卡芯片使用 125kHz 技术，可以从距芯片 180mm 外读 RFID 卡。还有其他读卡器有同样的或更大的读卡范围。通过增加额外的天线，可以增加到更大的读卡范围。

本章开始于连接一个读卡器到 Arduino 上，学习从它中得到串口数据，你会发现这是非常容易的，之后将做一个简单的门禁系统。

项目 44——简单的 RFID 读卡器

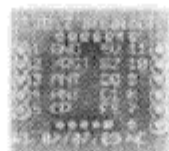
ID12 上的引脚不是标准的，因此它不能插入面包板。你需要从供应商（如 Sparkfun）处获得一个接口板。卡或标签可以从各种渠道购买，而且非常便宜。我在 eBay 上买了一包 keybob 型标签，只花了几美元。确保标签和卡是 125kHz 技术的，否则它将不能工作在这个读卡器上。

需要的元件

ID-12RFID 读卡器



ID-1 接口板



限流电阻



5mm LED



125kHz RFID 标签或卡*

(最少 4 个)



*感谢 Sparkfun 公司供图

把元件连接起来

如图 16-1 所示连接每一件东西。

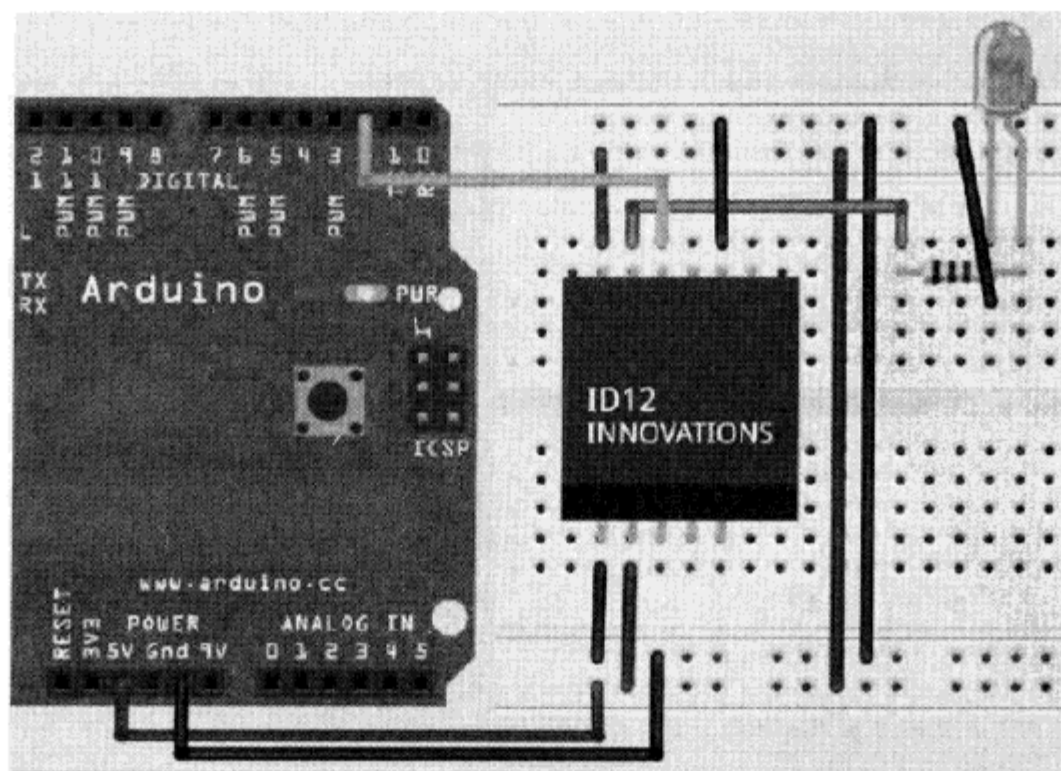


图 16-1 项目 44——简单的 RFID 读卡器电路图（参见彩色版插图）

在读卡器的引脚 10 (BZ) 上通过一个限流电阻连接 LED。

输入代码

输入清单 16-1 中的代码。

清单 16-1 项目 44 的代码

```
//项目 44

char val = 0; //从串口读到的数据

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

void loop () {

    if(Serial.available() > 0) {
        val = Serial.read(); //从串口读数据
        Serial.print(val, BYTE); //打印数据到串口监视器
    }
}
```

运行代码，打开串口监视器，把你的 RFID 标签或卡放在读卡器上。LED 将闪烁显示它已经读到卡，串口监视器窗口将显示 12 个数字，它组成了卡独一无二的 ID 号。把卡的 ID 号记下来，在下一个项目中需要卡的 ID 号。

这个程序的作用是读 RFID 卡数据并把它显示在串口上，你应该足以知道这个程序是如何工作的，因此我将跳过代码回顾直接看硬件。

硬件回顾

RFID 到处可见，从你的公交卡到让你进入办公室或大学等的门卡都是基于 RFID 技术的。卡有各种各样的形式和大小（见图 16-2），卡可以做得很小，科学家甚至可以把 RFID 标签放到蚂蚁上来记录蚂蚁的运动。RFID 是一个简单的元件，不做任何事情，只是通过无线电传递一个独一无二的 ID 号到读卡器。在大多数情况下，RFID 卡和标签是无源的、被动的，这意味着它们没有电源也不需要从外部供电。还有其他的选择，包括主动 RFID 卡，它们有自己的电源和电池，这种卡等待外部源唤醒之后使用自己的电源传送数据，可以给

出更大的感应范围。

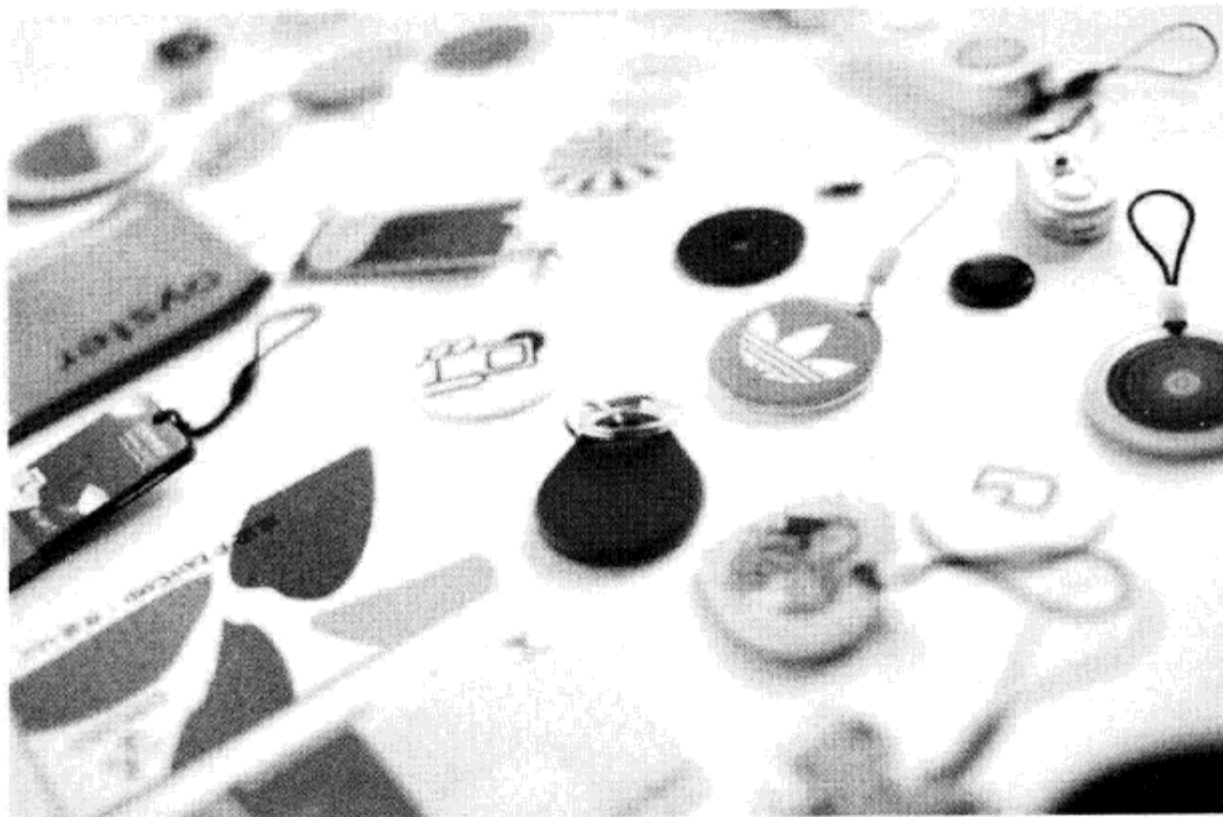


图 16-2 各种各样形式和大小的 RFID 标签和卡（感谢 Timo Arnall 供图）

本项目中使用的是被动式没有电池的 RFID 卡，这种卡从读卡器发出的一个电磁场中得到电能。因为标签进入电磁场，在标签内部的线圈中产生了电流。这个电流用于唤醒一个小芯片。它传送一个串行数据。之后读卡器以串口形式传送这些数据到 PC 或与它连接的微控制器。从 ID12 读卡器中送出的数据形式如下：

STX (02h)	DATA (10ASCII)	CHECKSUM(2 ASCII)	CR	LF	EXT (03H)
-----------	----------------	-------------------	----	----	-----------

首先送出 (ASCII02) STX，也就是传送开始字符，随后是由 10 个字节组成的独立 HEX（十六进制）数字。下两个十六进制数是校验和数字（在下一个项目中解释），之后有一个回车 (CR) 和换行符号。最后是 ETX，也就是传送结束符。

只有 12 个 ASCII 数字要显示在串口监视器上，因为其他的是不可打印字符。在下一个项目中，你将使用校验和保证接收到的字符串正确，STX 字符告诉你一串字符将要送出。

项目 45——门禁控制系统

下面要建立一个门禁系统，要使用 RFID 读卡器读标签并实现通过确认标签来允许门

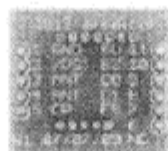
打开。Arduino 通过三极管操纵一个电子门锁。

需要的元件

ID-12 RFID 阅读器



ID-12 接口板*



限流电阻



5mm LED



125kHz RFID 标签或卡*
(至少 4 个)



1N4001 二极管



TIP-120 NPN 三极管



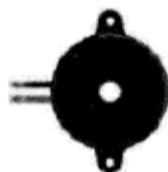
2.1mm 电源接口



12V DC 直流电源



8Ω扬声器或嗡鸣器



12V 电子门锁



*感谢 Sparkfun 供图。

把元件连接起来

如图 16-3 所示连接每一件东西。

如果你使用的不是 TIP120 三极管，需要参考你使用的三极管的数据，保证接线正确。在 TIP120 上从左到右依次是基极、集电极和发射极。基极连到数字引脚 7。集电极通过二极管和嗡鸣器或扬声器的负极端连接地。发射极直接接地。如果你有一个 8Ω 扬声器，可以使用扬声器，它会发出比喻鸣器更好听、音量也更大的声音。

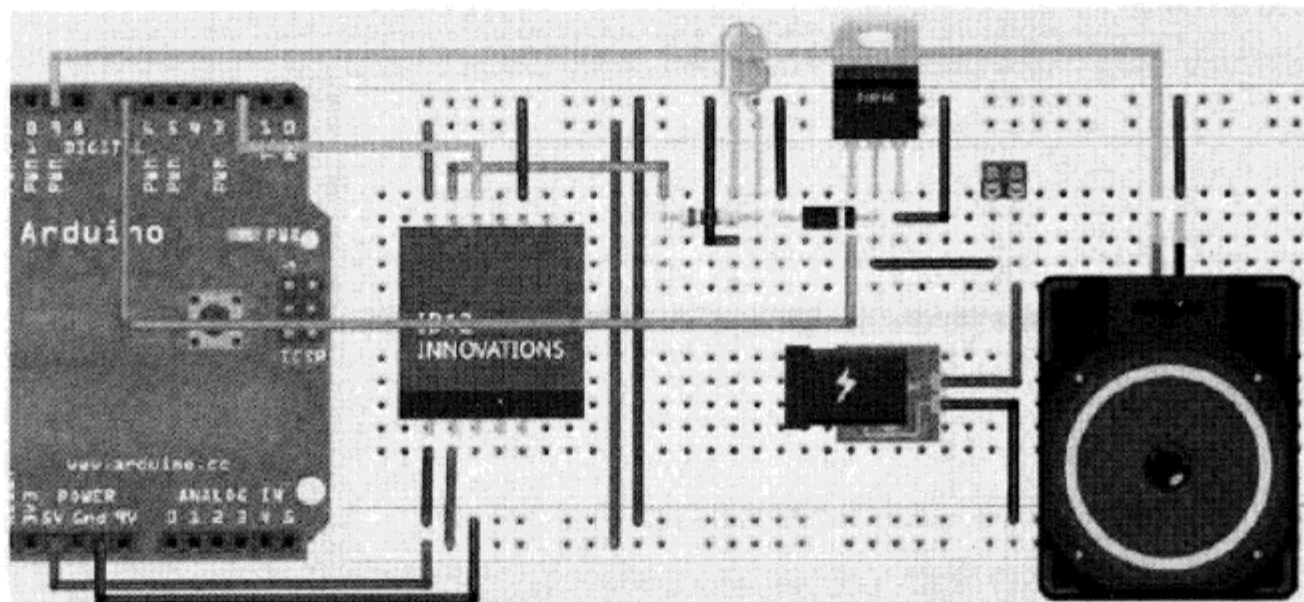


图 16-3 项目 45——门禁控制系统电路图（参见彩色版插图）

电子门锁的电源必须从一个至少有 500mA 电流输出的 12V DC 外部电源接出。

输入代码

输入清单 16-2 中的代码。

清单 16-2 项目 45 的代码

```
//项目 45

#define lockPin 7
#define speakerPin 9
#define tx 3
#define rx 2
#define unlockLength 2000

#include <SoftwareSerial.h>

SoftwareSerial rfidReader = SoftwareSerial(rx, tx);

int users = 3;
```

```

char* cards[] = { //有效卡
    "3D00768B53",
    "3D00251C27",
    "3D0029E6BF",
};

char* names[] = { //持卡人姓名
    "Tom Smith",
    "Dick Jones",
    "Harry Roberts"
};

void setup() {
    pinMode (lockPin, OUTPUT);
    pinMode (speakerPin, OUTPUT);
    digitalWrite(lockPin, LOW);
    Serial.begin(9600);
    rfidReader.begin(9600);
}

void loop() {
    char cardNum[10]; //存储卡号的数组
    byte cardBytes[6]; //字节形式的卡号+校验和
    int index=0; //索引号
    byte byteIn=0; //从RFID卡中读字节
    byte lastByte=0; //最后读的字节
    byte checksum = 0; //校验和结果存在这里

    if (rfidReader.read()==2) { //读 RFID 阅读器
        while(index<12) { //串口中的 12 个数字
            byteIn = rfidReader.read(); //存储值到byteIn 中
            //如果是 STX、ETX、CR 或 LF，退出
            if ((byteIn==1) || (byteIn==2) || (byteIn==10) || (byteIn==13)) {return;}
            //只存储头 10 个十六进制数（最后两个是校验和）
            if (index<10) {cardNum[index]=byteIn;}
            //ASCII 字符转换为相应的十六进制数
            if ((byteIn>='0') && (byteIn<='9')) {
                byteIn -= '0';
            }
            else if ((byteIn>='A') && (byteIn<='F')) {
                byteIn = (byteIn+10) - 'A';
            }
            //如果是奇数把两个十六进制数（占 4 比特）合成一个 8 比特的字节

```



```

        if ((index & 1) == 1) {
            //把上一次得到的数字左移 4 位并添加新数字
            cardBytes[index/2]= (byteIn | (lastByte<<4));
            if (index<10) {checksum ^= cardBytes[index/2];} //获得校验和
        }
        lastByte=byteIn; //存储最后阅读的字节
        index++; //索引增加
        //如果已经到达所有数字的结尾, 在结尾处增加 null 符号
        if (index==12) {cardNum[10] = '\0';}
    }

    Serial.println(cardNum); //打印卡号
    int cardIndex =checkCard(cardNum); //检查卡是否有效并返回索引号
    if(cardIndex>=0 && (cardBytes[5]==checksum)) { //如果卡号有效并且校验和正确
        Serial.println("Card Validated");
        Serial.print("User: ");
        Serial.println(names[cardIndex]); //打印持卡人姓名
        unlock(); //开门锁
        Serial.println();
    }
    else {
        Serial.println("Card INVALID");
        tone(speakerPin, 250, 250);
        delay(250);
        tone(speakerPin, 150, 250);
        Serial.println();
    }
}
}

int checkCard(char cardNum[10]) {
    for (int x=0; x<=users; x++) { //检查所有有效卡
        if(strcmp(cardNum, cards[x])==0) { //用最后读到的卡号进行对比
            return (x); //返回卡号索引
        }
    }
    return (-1); //如果不匹配, 返回-1
}

void unlock() {
    tone(speakerPin, 1000, 500);
    digitalWrite(lockPin, HIGH);
    delay(unlockLength);
}

```

```
digitalWrite(lockPin, LOW);
}
```

在程序的开始处把三个标签 ID 号码输入 `cards[]` 数组中。如果有必要可使用项目 44 的程序去找到标签的 ID 号码（运行代码，并打开串口监视器）。现在在阅读器上刷 4 张卡，阅读器将闪烁 LED 表示它已经读到卡，你将得到如下的输出：

```
3D00251C27
Card Validated
User: Dick Jones

3D0029E6BF
Card Validated
User: Harry Roberts

3D002A7C6C
Card INVALID

3D00768B53
Card Validated
User: Tom Smith
```

卡的号码将显示在屏上，之后是字符串“Card INVALID”或“Card Validated”，接下来是使用者的名字。如果卡是有效的，蜂鸣器发出一个高音，然后锁打开两秒钟。如果卡不是有效的，嗡鸣器发出两声低音，门锁不打开。12V 电子门锁使用三极管提供高电压，比 Arduino 能够提供电流更大。在第 5 章中当驱动一个直流电机时使用了同样的三极管。下面让我们看项目是如何工作的。

代码回顾

首先，定义要使用的锁和扬声器的引脚。要使用 `SoftwareSerial.h` 库而不是一般的在数字引脚 0 和 1 上的串口引脚，因此必须定义 `rx` 和 `tx` 引脚。还需要定义一个用毫秒作为单位的时间长度，表示电子门锁保持开启的时间。

```
#define lockPin 7
#define speakerPin 9
#define tx 3
#define rx 2
#define unlockLength 2000
```

使用 `SoftwareSerial.h` 库（现在这个库是 Arduino IDE 核心的一部分）是为了方便。如果你使用标准 rx 和 tx 引脚，每当你想上传代码到 Arduino 中时，不得不取下连接在这些引脚上的连线。通过使用 `SoftwareSerial.h` 库，可以使用任何你想要的引脚作为串行通信引脚。

```
#include <SoftwareSerial.h>
```

之后，生成一个 `SoftwareSerial` 实例，叫它 `rfidReader`，传递给它已经定义的 rx 和 tx 引脚：

```
SoftwareSerial rfidReader = SoftwareSerial(rx, tx);
```

之后，定义一个变量，用来存储数据库中使用者的数量：

```
int users = 3;
```

之后是两个数组存储卡的 ID 号和持卡人的姓名，改变数组中的卡号为你自己的卡号（仅开头十个数）：

```
char* cards[] = { //有效卡
  "3D00768B53",
  "3D00251C27",
  "3D0029E6BF",
};

char* names[] = { //持卡人姓名
  "Tom Smith",
  "Dick Jones",
  "Harry Roberts"
};
```

setup 函数设置锁和扬声器引脚为输出模式：

```
pinMode (lockPin, OUTPUT);
pinMode (speakerPin, OUTPUT);
```

设置锁引脚为 **LOW**，保证项目开始时锁不会打开：

```
digitalWrite(lockPin, LOW);
```



之后开始串口通信:

```
Serial.begin(9600);
rfidReader.begin(9600);
```

之后是主循环, 开始于定义要在主循环中使用的变量:

```
char cardNum[10]; //存储卡号的数组
byte cardBytes[6]; //字节形式的卡号+校验和
int index=0; //索引号
byte byteIn=0; //从 RFID 卡中读字节
byte lastByte=0; //最后读的字节
byte checksum = 0; //校验和结果存在这里
```

下面检查是否有数据进入 RFID 阅读器串口, 如果有, 而且第一个字符是 2 号 ASCII 码, 它是传递开始的代码, 你就可以知道一个 ID 字符串要传送, 而且可以开始读数字了。

```
if (rfidReader.read()==2) { //读 RFID 阅读器
```

之后是一个循环, 当索引小于 12 时运行这个循环:

```
while(index<12) { //串口中的 12 个数字
```

索引值将存储读入数字的当前位置。因为要读一个长度为 12 个字符的数字, 所以仅需要读头 12 个数字。

之后, 从串口读入值并存储在 `byteIn` 中:

```
byteIn = rfidReader.read(); //存储值到 byteIn 中
```

在某些时候一些字符由于某些原因丢失了, 你会再次检测传送开始、传送结束、回车或换行符。如果检测到这些符号, 循环退出。

```
//如果是 STX、ETX、CR 或 LF, 退出
if ((byteIn==1) || (byteIn==2) || (byteIn==10) || (byteIn==13)) {return;}
```

12 个数字的字符串最后两个数字是校验和, 不希望存储校验和到卡号数组中, 因此仅存储头 10 个数字。

```
if (index<10) {cardNum[index]=byteIn;}//只存储头 10 个十六进制数(最后两个是校验和)
```

之后，需要把读到的 ASCII 字符转换为相应的十六进制数，因此运行 if-else 语句去检查字符是否在 0~9 和 A~F 之间，如果是，转换它们为相应的十六进制数。

```
if ((byteIn>='0') && (byteIn<='9')) {
    byteIn -= '0';
}
else if ((byteIn>='A') && (byteIn<='F')) {
    byteIn = (byteIn+10)-'A';
}
```

使用逻辑与(&&)操作保证字符在 0~9 之间或 A~F 之间。之后，转化最后两个十六进制数为一个字节。一个十六进制数的基是 16，我们常用的进制是十进制，有数字 0~9，在十六进制中，数字是从 0~15，使用字母 A~F 表示 10~15，因此十六进制数 FF 表示十进制数 255：

```
F=15
(F*16)+F=(15*16)+15=255
```

你需要把最近两次读到的两个 ASCII 数字合成一个字节，变成相应的十进制数。你已经声明一个变量叫做 lastByte，用来存储 while 循环上一次处理的数字。这个数字初始化为 0。你只需要每两个数字合成一个字节，因为两个十六进制数组成一个字节。因此判断索引值是否为一个奇数，通过把索引中存储的值与 1 进行按位与操作，看结果是否也是 1。记住索引开始为 0，因此第二个数字的索引是 1。

```
if ((index & 1) == 1) {//如果是奇数把两个十六进制数(占 4 比特)合成一个 8 比特的字节
```

任何奇数和 1 进行按位与结果还是 1，任何偶数和 1 进行按位与结果是 0。

```
12 (偶数) & 1 = 0
00001100
00000001 &
= 00000000

11 (奇数) & 1 = 1
00001011
00000001 &
```

```
= 00000001
```

如果语句决定在第二、四、六、八、十个字符处对字符进行以下计算，把结果存储在 cardBytes 中：

```
//把上一次得到的数字左移 4 位并添加新数字
cardBytes[index/2]= (byteIn | (lastByte<<4));
```

使用 index/2 判定索引值，索引是一个整数，除法运算结果中只有小数点前的值才保留。因此每处理两个数字，CardBytes 的索引值将加 1。

这样操作的结果是用上一次收到的数字值向左位移 4 个位置，之后用这个数和当前读到的值执行一个按位或操作。这样可以把上一次收到的数字左移 4 位后与当前收到的数字的 4 位合成一个 8 位十六进制数。向左移位 4 个位置后上一次读到的数组成前 4 个比特。之后用第二个十六进制数与之混合成一个完整的字节。因此，如果第一个数是 9，第二个数是 E，这个计算结果将是：

```
Lastbyte = 9 = 0001001
00001001 << = 10010000
E = 14 = 00001110
          10010000 OR
= 10011110
```

校验和是一个数字，保证全部的字符串阅读正确。校验和在数据传送中使用得很多，它们只是每个字符全部进行异或的结果。

```
if (index<10) {checksum ^= cardBytes[index/2];} //获得校验和
```

你的一个卡的 ID 是：

3D00768B53

因此，它的校验和是：

```
3D XOR 00 XOR 76 XOR 8B XOR 53
3D = 00111101
00 = 00000000
76 = 01110110
8B = 10001011
```



```
53 = 01010011
```

如果你对每一个数字执行 XOR（异或），结果是 93，因此 93 是卡 ID 号的校验和。如果由于干扰造成任何一个数字传递错误，算出的校验和将不再是 93，你就可以知道从卡中读到的数不正确，需要丢掉已经读到的数据。

因为下一次循环要给 lastByte 赋新值，所以在循环外设置它为当前值。

```
lastByte=byteIn; //存储最后阅读的字节
```

索引号增加：

```
index++; //索引号增加
```

如果已经到字符串尾部，必须在 cardNum 的第十位数字设置 ASCII 码值为 \0 或者空。当你需要判断是否已到字符串的结尾时这是必需的。 \0 表示已到字符串的结尾。

```
if (index==12) {cardNum[10] = '\0';}
```

之后打印从 RFID 读卡器中读到的卡号：

```
Serial.println(cardNum); //打印卡号
```

之后生成一个叫做 cardIndex 的整数，赋值为 checkCard()函数的返回值（简单说明一下）。如果卡号是一个有效的 checkCard()函数将返回一个正值，否则如果卡号不在有效卡数据库中，函数返回一个负值。

```
int cardIndex =checkCard(cardNum); //检查卡是否有效并返回索引号
```

之后判断是否满足返回数字是正并且校验和正确这一条件，若满足条件表明卡有效而且读卡过程正确，所以可以打开电子门锁。

```
if(cardIndex>=0 && (cardBytes[5]==checksum)) { // 如果卡号有效并且校验和正确
    Serial.println("Card Validated");
    Serial.print("User: ");
    Serial.println(names[cardIndex]); //打印持卡人姓名
    unlock(); //开门锁
    Serial.println();
}
```

如果卡是无效的或校验和不正确，也就是卡确定为无效或使用者是不允许进入的。

```
else {
    Serial.println("Card INVALID");
    tone(speakerPin, 250, 250);
    delay(250);
    tone(speakerPin, 150, 250);
    Serial.println();
}
```

之后到达 `checkCard()` 函数，它将返回一个整数，因此它的类型是整型，你传递给它的参数是卡号。

```
int checkCard(char cardNum[10]) {
```

之后，你循环检查在数据库中的每一个卡号，看是否有与刚读入的卡号相同的。

```
for (int x=0; x<=users; x++) { //检查所有有效卡
```

使用 `strcmp` 函数或字符串匹配函数来确定你传递给 `checkCard()` 函数的卡号是否与数据库目前的卡号相匹配。这就是为什么你需要一个卡号数据库，因为 `strcmp` 函数需要它。

```
if(strcmp(cardNum, cards[x])==0) { //用最后读到的卡号进行对比
```

`strcmp` 函数需要两个参数，它们是希望对比的两个字符串。如果每个字符匹配，则从函数返回的数是 0。如果返回非零值，则表示两个字符串没有匹配。如果匹配，返回 `x` 的值，它是数据库内有效卡的索引。

```
return (x); //返回卡号索引
```

如果卡没有匹配，返回-1。

```
return (-1); //如果不匹配，返回-1
```

最后的函数是 `unlock()` 发出一个高音声响，开门锁，等待一个确定的时间之后重新锁上门：

```
void unlock() {
    tone(speakerPin, 1000, 500);
```

```
digitalWrite(lockPin, HIGH);  
delay(unlockLength);  
digitalWrite(lockPin, LOW);  
}
```

这个项目的下一步是增加更多的读卡器和电子门锁来满足你的家门入口授权需求，使用者可带一个卡或标签以允许他们进入相关的房间。可以给每一个使用者一个个人授权，因此他们可以进入建筑物的不同部分，通过不同的入口，这样可以只允许使用者进入特定区域。

下面到这本书的最后一章了，在那里你将连接 Arduino 到因特网！

小结

在本章，你看到了从 RFID 卡或标签中读数是多么简单，之后使用这些数去开一个电子门锁或做其他动作。我见过一个项目使用 RFID 卡开公交车的门。也有的项目把 RFID 读卡器做成一个碗，当使用者到家时把他们的卡放到碗里，之后房子对到家的人有所反应，例如，打开空调调节室温到主人设定的温度，调整灯光等级，给主人放音乐，播放电影等。当使用一个 RFID 卡时，受限制的只是你自己的想象力。

本章的主题和概念

- RFID 技术是如何工作的
- 如何连接 ID12 RFID 读卡器到 Arduino
- 从 RFID 读卡器中阅读串行数据
- 使用三极管控制高电压元件
- 使用软件串行库
- 转换 ASCII 码到十六进制值
- 使用按位与判断一个数是否为奇数
- 使用位移和按位或连接两个十六进制数为一个字节
- 使用 XOR 生成校验和
- 使用 strcmp 对比两个字符串

第 17 章

连接到 Internet

在本书的最后一章中，你将学习如何把 Arduino 连接到路由器上，使数据可以通过 Internet 送出。如果把 Arduino 连接到 Internet，就可以从网络上的任何地方读到 Arduino 的数据，也可以用 Arduino 送出数据到 Internet 上，之后通过浏览器看到这些数据。这个项目是使用官方的 Arduino Ethernet 板完成的。

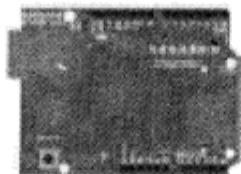
连接 Arduino 到一般网络上或 Internet 上打开了一个完整的潜在的项目方向。Arduino 可以送数据到网站，如发布微博。也可以通过 Internet 控制 Arduino 或使用 Arduino 作为网络服务器，通过简单的网页连接就可以发布传感器数据等。这章给你基本的知识来创建你自己的基于 Arduino 的 Ethernet 或 Internet 项目。

项目 46——Ethernet 板

本项目使用 Ethernet 板和一个温度传感器来说明如何连接 Arduino 到 Ethernet。

需要的元件

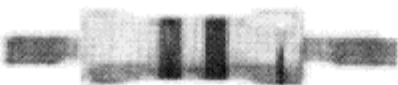
Arduino Ethernet 板



2 个 DS18B20 温度传感器



4.7k Ω 电阻



把元件连接起来

插入 Ethernet 板到 Arduino 板上,之后如图 17-1 所示连接每件东西,在 Arduino Ethernet 板上接线就像直接接到 Arduino 上相同的位置上一样。

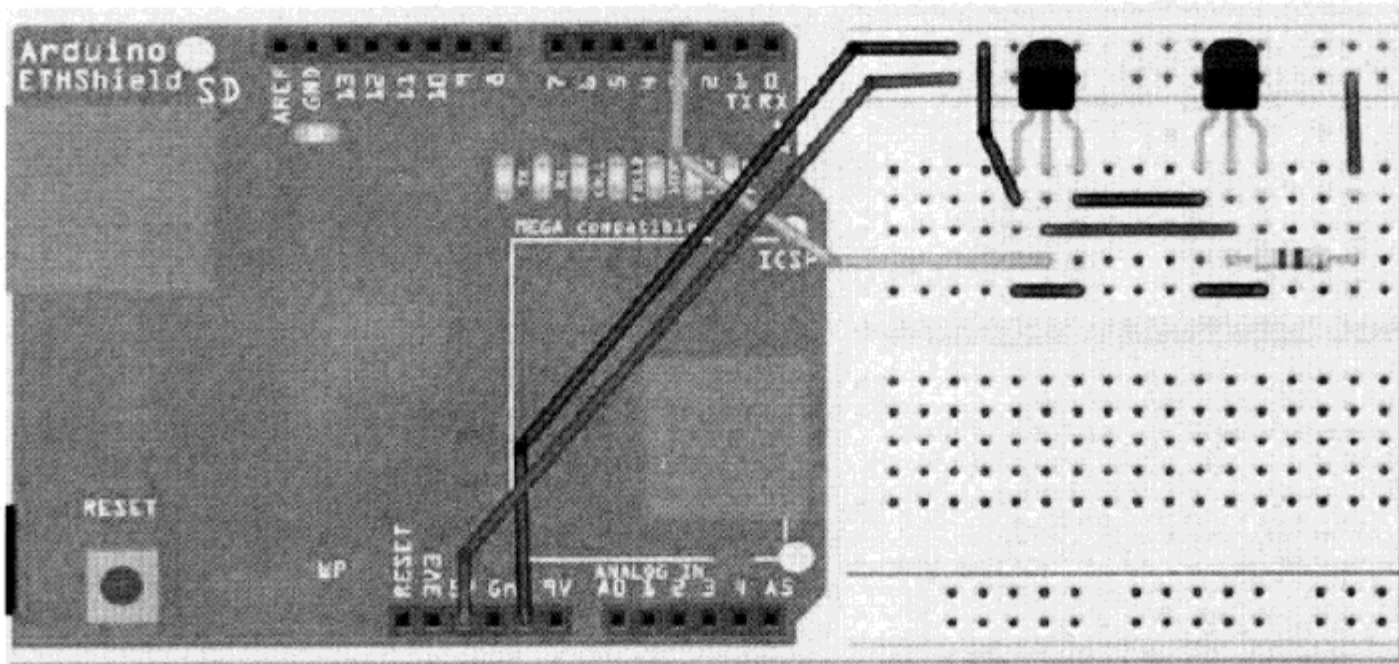


图 17-1 项目 46——Ethernet 板电路图 (参见彩色版插图)

输入代码

输入清单 17-1 中的代码。

清单 17-1 项目 46 的代码

//项目 46——David A. Mellis 和 Tom Igoe 编写的基于 Arduino 的网页服务器例子

```
#include <SPI.h>
#include <Ethernet.h>
#include <OneWire.h>
#include <DallasTemperature.h>

//定义要用来从传感器中读取数据的数字引脚
#define ONE_WIRE_BUS 3
#define TEMPERATURE_PRECISION 12

float tempC, tempF;

//生成一个 oneWire 实例连接任意单线设备 (不仅仅是 Maxim/Dallas 温度传感器)
```

```

OneWire oneWire(ONE_WIRE_BUS);

//传递 oneWire 引用给 Dallas 温度传感器
DallasTemperature sensors(&oneWire);

//存储元件地址的数组
DeviceAddress insideThermometer = { 0x10, 0x7A, 0x3B, 0xA9, 0x01, 0x08, 0x00,
0xBF };
DeviceAddress outsideThermometer = { 0x10, 0xCD, 0x39, 0xA9, 0x01, 0x08, 0x00,
0xBE};

byte mac[] = { 0x48, 0xC2, 0xA1, 0xF3, 0x8D, 0xB7 };
byte ip[] = { 192,168,0, 104 };

//建立 80 端口服务器
Server server(80);

void setup()
{
    //开始 Ethernet 通信服务
    Ethernet.begin(mac, ip);
    server.begin();
    //使用传感器
    sensors.begin();
    //设置精度
    sensors.setResolution(insideThermometer, TEMPERATURE_PRECISION);
    sensors.setResolution(outsideThermometer, TEMPERATURE_PRECISION);
}

//从传感器中获得温度的函数
void getTemperature(DeviceAddress deviceAddress)
{
    tempC = sensors.getTempC(deviceAddress);
    tempF = DallasTemperature::toFahrenheit(tempC);
}

void loop()
{
    sensors.requestTemperatures();

    //监听客户机的输入
    Client client = server.available();
    if (client) {

```



```

//从客户机发出的一个 HTTP 请求要结束于一个空行符号
boolean BlankLine = true;
while (client.connected()) {
  if (client.available()) {
    char c = client.read();

    //如果行是空的并且行结尾是换行符 '\n', 说明到了 HTTP 请求的结尾
    if (c == '\n' && BlankLine) {
      getTemperature(insideThermometer);
      client.println("HTTP/1.1 200 OK"); //标准 HTTP 响应
      client.println("Content-Type: text/html\n");
      client.println("<html><head><META HTTP-EQUIV=\"\"refresh\"\"CONTENT=
""5\"\">\n");
      client.println("<title>Arduino Web Server</title></head>");
      client.println("<body>\n");
      client.println("<h1>Arduino Web Server</h1>");
      client.println("<h3>Internal Temperature</h3>");
      client.println("Temp C:");
      client.println(tempC);
      client.println("<br/>");
      client.println("Temp F:");
      client.println(tempF);
      client.println("<br/>");
      getTemperature(outsideThermometer);
      client.println("<h3>External Temperature</h3>");
      client.println("Temp C:");
      client.println(tempC);
      client.println("<br/>");
      client.println("Temp F:");
      client.println(tempF);
      client.println("<br/>");

      break;
    }
    if (c == '\n') {
      //开始一个新行
      BlankLine = true;
    }
    else if (c != '\r') {
      //有一个字符在当前行中
      BlankLine = false;
    }
  }
}

```

```

    }
  }
  //允许浏览器有时间接收数据
  delay(10);
  //停止客户机
  client.stop();
}
}

```

你需要在这一行中输入 IP 地址：

```
byte ip[] = { 192,168,0, 104 };
```

你要根据自己的系统改变以上的 IP 地址。需要从路由器中找到计算机设置的 IP 地址范围来更改 IP 地址。通常，地址将开始于 192.168.0 或 192.168.1，之后你通常只需要把它的最后一位设置成高于 100，保证地址没有被其他设备占用了而发生冲突。可能需要进入路由器设置确认任何到 80 端口的 HTTP 请求指向网卡板的 IP 地址。见你的路由器菜单下的“Port Forwarding”菜单项，要在防火墙设置中打开端口 80。

现在打开你的网络浏览器，输入 IP 地址和端口如下：

```
192.168.0.104:80
```

如果每件事情都做对，在你的浏览器中可以看到如图 17-2 所示的网页。

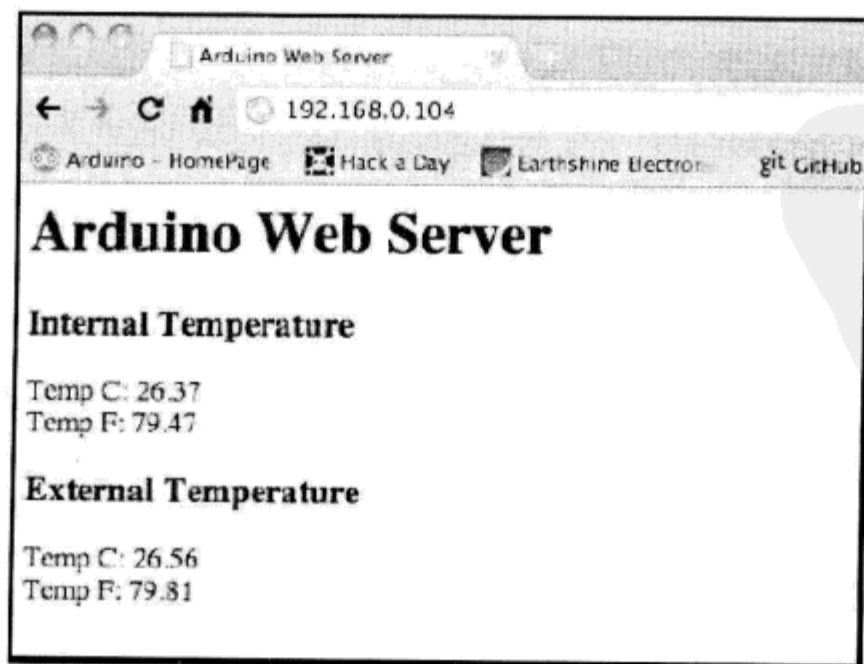


图 17-2 Arduino 网页服务的输出

这个页面每 5 秒自动刷新一次来显示温度的变化。如果你已经在路由器中正确设置了前面的端口和防火墙，那么在任何地方只要连上局域网就可以得到这个页面。你要知道路由器的 IP 地址，这可以在路由器的 **Administration** 页面中找到。在任何浏览器中输入以下的 IP 地址，冒号后是端口号：

```
95.121.118.204:80
```

上面的网址的网页现在将显示在浏览器中，你可以从以太网的任何入口查看温度读数。

代码回顾

上面代码的一些部分与项目 37 中的一样，因此我会复制那些部分，并且把主要精力集中于与 **Ethernet** 板相关的代码。首先，要加载库文件，确保在库文件夹内有这个给温度传感器的库。要注意的是版本 0019 的 **Arduino IDE** 在每个 **Ethernet.h** 库的项目中需要包含 **SPI.h** 库（见项目 37）。

```
#include <SPI.h>
#include <Ethernet.h>
#include <OneWire.h>
#include <DallasTemperature.h>
```

之后是引脚和传感器精度设置：

```
#define ONE_WIRE_BUS 3
#define TEMPERATURE_PRECISION 12
```

接下来是两个浮点数，用其存储摄氏度或华氏度单位的温度值。

```
float tempC, tempF;
```

生成一个 **OneWire** 对象的实例，作为参数传递到 **DallasTemperature** 库：

```
OneWire oneWire(ONE_WIRE_BUS);
DallasTemperature sensors(&oneWire);
```

为这两个温度传感器设置地址，有必要用项目 37 中的代码查出传感器的 ID 号。

```
DeviceAddress insideThermometer = { 0x10, 0x7A, 0x3B, 0xA9, 0x01, 0x08, 0x00,
```

```
0xBF };
DeviceAddress outsideThermometer = { 0x10, 0xCD, 0x39, 0xA9, 0x01, 0x08, 0x00,
0xBE};
```

之后需要定义元件的 MAC 和 IP 地址：

```
byte mac[] = { 0x48, 0xC2, 0xA1, 0xF3, 0x8D, 0xB7 };
byte ip[] = { 192,168,0, 104 };
```

Mac 地址（物理地址）是一个独一无二的给网络接口的记号，你的 PC 或 Mac 上的网卡有它们自己的由制造商确定的 Mac 地址。在这个例子里，你自己决定 Mac 地址，它只是一个 48 位数。因此只要设置任何十六进制数到地址中，甚至让它们像代码中的那样也行。IP 地址需要手工设置，它必须在路由器的允许范围内。

之后，以设备的端口号为参数生成一个 Server 类实例：

```
Server server(80);
```

服务器可以监听到这个特定的端口上的连接。一个端口号只是一个数据的通路。网卡只是接入设备，端口号决定数据的去向。想象 Mac 地址是一个建筑物地址，建筑物有很多单元，每个单元有自己的房间，每个房间有自己的编号一样。

之后到达 setup 函数。开始于初始化 Ethernet 通信，传递 Mac 地址和 IP 地址给设备的实例：

```
Ethernet.begin(mac, ip);
```

现在，需要使用 begin() 函数告诉你的服务器开始监听连接的输入：

```
server.begin();
```

程序开始于使能传感器和设置它们的精度：

```
sensors.begin();
sensors.setResolution(insideThermometer, TEMPERATURE_PRECISION);
sensors.setResolution(outsideThermometer, TEMPERATURE_PRECISION);
```

之后，建立一个函数从传感器中获得温度（像项目 37 中做的那样）：

```
void getTemperature(DeviceAddress deviceAddress)
{
    tempC = sensors.getTempC(deviceAddress);
    tempF = DallasTemperature::toFahrenheit(tempC);
}
```

之后到程序主循环，首先，从两个传感器获得温度值：

```
sensors.requestTemperatures();
```

需要监听任何客户输入，例如，网页浏览器要求通过 Arduino 查看网页服务，需要生成一个 Client 实例，使用它去检查服务器内是否有数据可读。客户机是将要连接到 Arduino 的 Web 浏览器，服务器是 Arduino。

```
Client client = server.available();
```

之后，检查客户机是否连上、是否有数据可以给客户机，如果有，if 语句执行。

```
if (client) {
```

首先 if 语句生成一个布尔变量 BlankLine，设置它为 true：

```
boolean BlankLine = true;
```

从客户机发出的一个 HTTP 请求将结束于一个空行符号，中断用一个换行字符。因此使用 BlankLine 变量去决定是否已经到达数据尾部。

之后，检查是否客户机还连着，如果是，运行在 while 循环中的代码：

```
while (client.connected()) {
```

之后，检查给客户机的数据是否准备好了。如果数据准备好了，下一个 if 语句中的代码被执行。available() 函数返回已经向连接到服务器上的客户机中写入的字节数。如果值是零以上，那么 if 语句被执行。

```
if (client.available()) {
```

之后，定义一个字节变量，存储从服务器收到的下一个字节，使用 client.read() 函数获

得字节。

```
char c = client.read();
```

如果读到的字符是换行符 (“\n”), 需要检查 BlankLine 是 true 还是 false。如果是 true, 说明已经到 HTTP 的尾部, 因此可以返回 HTML 代码到客户机 (用户的网络浏览器)。

```
if (c == '\n' && BlankLine) {
```

之后, 要从你的服务器送出的数据。开始从内部传感器获得温度:

```
getTemperature(insideThermometer);
```

之后, 返回给客户机的 HTML 代码。每个页面组成代码叫做 HTML (或 Hyper Text Markup Language)。解释 HTML 语言超出了本书的范围, 因此我将仅仅给出一些基本的信息。如果你想获得更多关于 HTML 的信息, 在 <http://en.wikipedia.org/wiki/HTML> 网址上查看维基百科中的 HTML 词条。在 Internet 上有许多 HTML 教程可用。使用 client.println() 函数去给客户机发布数据, 送出代码生成网页。在许多浏览器中如果在网页中单击鼠标右键, 会有一个弹出菜单查看网页源代码。试一下, 可以看到你刚刚看到的组成网页的 HTML 代码。这个代码告诉浏览器要显示什么和如何显示。

首先, 告诉客户机使用的 HTTP 版本是 1.1, 这是一个用于网页的标准协议, 它包含了要送出的 HTML 文件:

```
client.println("HTTP/1.1 200 OK"); //标准 HTTP 响应
client.println("Content-Type: text/html\n");
```

之后, 有 HTML 标签表明从现在开始每行的内容是 HTML 代码和 HTML 代码的头标签。头标签在希望浏览器去执行的任何命令和脚本等代码主体之前。第一个命令告诉浏览器让网页每 5 秒自动刷新一次。

```
client.println("<html><head><METAHTTP-EQUIV=\"\"refresh\"\"CONTENT=\"\"5\"\">\n");
```

之后, 给出页面标题。它将出现在浏览器的顶部, 你可以给这个页面随便添加一个标题。

```
client.println("<title>Arduino Server</title></head>");
```


通过插入一个</head>标签结束头部分。之后是 HTML 的程序体，这是使用者可以见到的部分。

```
client.println("<body>\n");
```

显示一个<h1>头标志 “Arduino Web Server”，h1 是最开始的头部，跟着是 h2、h3 等。

```
client.println("<h1>Arduino Web Server</h1>");
```

随后是下一部分的标题，它是 “InternalTmeperature”，头部为 h3。

```
client.println("<h3>Internal Temperature</h3>");
```

之后，输入用摄氏度和华氏度为单位的温度，接着是换行符号：

```
client.println("Temp C:");
client.println(tempC);
client.println("<br/>");
client.println("Temp F:");
client.println(tempF);
client.println("<br/>");
```

之后，获得外部温度和显示外部温度：

```
getTemperature(outsideThermometer);
client.println("<h3>External Temperature</h3>");
client.println("Temp C:");
client.println(tempC);
client.println("<br/>");
client.println("Temp F:");
client.println(tempF);
client.println("<br/>");
```

之后，用一个 break 语句退出 while 循环：

```
break;
```

现在如果读到\n（换行符）符，设置 BlankLine 为 true，如果不是\r 符号（回车符）等，

设置 BlankLine 为 false，这表示还有字符要从服务器读。

```
if (c == '\n') {
    // 开始一个新行
    BlankLine = true;
}
else if (c != '\r') {
    //有一个字符在当前行中
    BlankLine = false;
}
```

等待一个短的延时，允许浏览器有时间接收数据。之后用 stop()函数停止客户机，这个函数从服务器上断开客户机的连接。

```
delay(10);
client.stop();
```

这个项目介绍通过网络提供一个网页的基本知识，在网页中包含传感器的数据。其实还有其他更好的通过 Internet 提供数据的方法，你将在后面看到这些方法中的一个。

项目 47——Internet 天气显示

使用同样的元件和电路，但是这次将送出两个传感器的温度数据到 Pachube 网站。Pachube 是一个在线数据服务器，允许使用者连接传感器到网页（见图 17-3），数据可以用各种各样的形式送出并在网址上用网摘的形式显示一个标签。网摘是时时更新的并且可以嵌入网站中。通过拖曳一个网摘也可以看到历史数据，可以送出报警控制脚本、设备等。网站上有一系列的专门为 Arduino 写的教程。

要使用 Pachube 服务器，你必须注册一个 Pachtube 用户。它是显示传感器中的数据、家庭能源监控、建筑物监视系统等的一个理想方法。有几个等级可以选择，从免费服务（10 数据流，每分钟 10 次更新，30 条历史记录）到 4 个付费服务级别，付费服务允许更多的数据流、刷新率和历史数据长度。

在上传数据前必须先注册一个账户，因此首先进入网址 www.pachtube.com，单击“SIGN UP”按钮，选择一个服务（你可以尝试有 7 天限制的任何服务）。对于这个项目，使用免费的服务即可。如果你喜欢这项服务，需要更多数据流，必须在免费期结束前升级为付费

服务。



图 17-3 Pachube 网站

下一页面需要你提供一个用户名、E-mail 地址和口令，填好这些后，单击“SIGN UP”按钮。一旦成功注册，可以返回到主页登录。现在你要生成一个数据源，单击“Register a Feed”按钮，出现一个数据源表格（见图 17-4）。

选择手工数据源形式，并输入数据源标题。如果你愿意也可以选择数据源的地址并在“Description”方框中输入一些描述。所有免费的 Pachube 数据源服务都是公开的，所以如果你不想让任何人看到更详细的信息就不要再数据源注册页输入任何东西。

之后，增加 4 个数据源。这些数据源是从温度传感器获得的用摄氏度和华氏度表示的室内温度和室外温度。保持 ID 不变，输入数据源的名字和使用的符号，如图 17-4 所示。

一旦成功输入全部数据，单击“Save Feed”按钮，进入刚刚生成数据源的页面，在那里你将看到在注册页面上输入的数据源的名字。

Feed type * ☐ automatic ☒ manual

Feed title *

Description

Tags

Website

Contact email
Note - this email address will be publicly available on the site, so that non-members can contact you. Please don't supply an email address you wish to keep private.

Datasources

ID (required)	Tags	Units	Symbol	Type	
0	Internal Temp.	Celsius	C	---	<input type="button" value="add"/> <input type="button" value="remove"/>
1	Internal Temp.	Fahrenheit	F	---	<input type="button" value="add"/> <input type="button" value="remove"/>
2	External Temp.	Celsius	C	---	<input type="button" value="add"/> <input type="button" value="remove"/>
3	External Temp.	Fahrenheit	F	---	<input type="button" value="add"/> <input type="button" value="remove"/>

☒ Save Feed ☐ Cancel

Location name

Elevation (m)

Exposure ☐ indoor ☐ outdoor

Disposition ☐ fixed ☐ mobile

Domain ☒ physical ☐ virtual

Double click to set the feed location, drag-and-drop marker to reposition, right click for additional options.

图 17-4 Pachube 数据源注册页

现在需要从页面中获得一些信息。首先需要数据源号。看数据源页面左上部，在标题下面以.XML 结尾的 URL，URL 结尾处的数是你的数据源号（见图 17-5），把它抄下来。

之后，需要获得主键。为了获得主键，单击页面顶部的“My Settings”按钮。在数字标题页将显示“Your Master API Key”（见图 17-6）。复制粘贴这个主键到代码内。

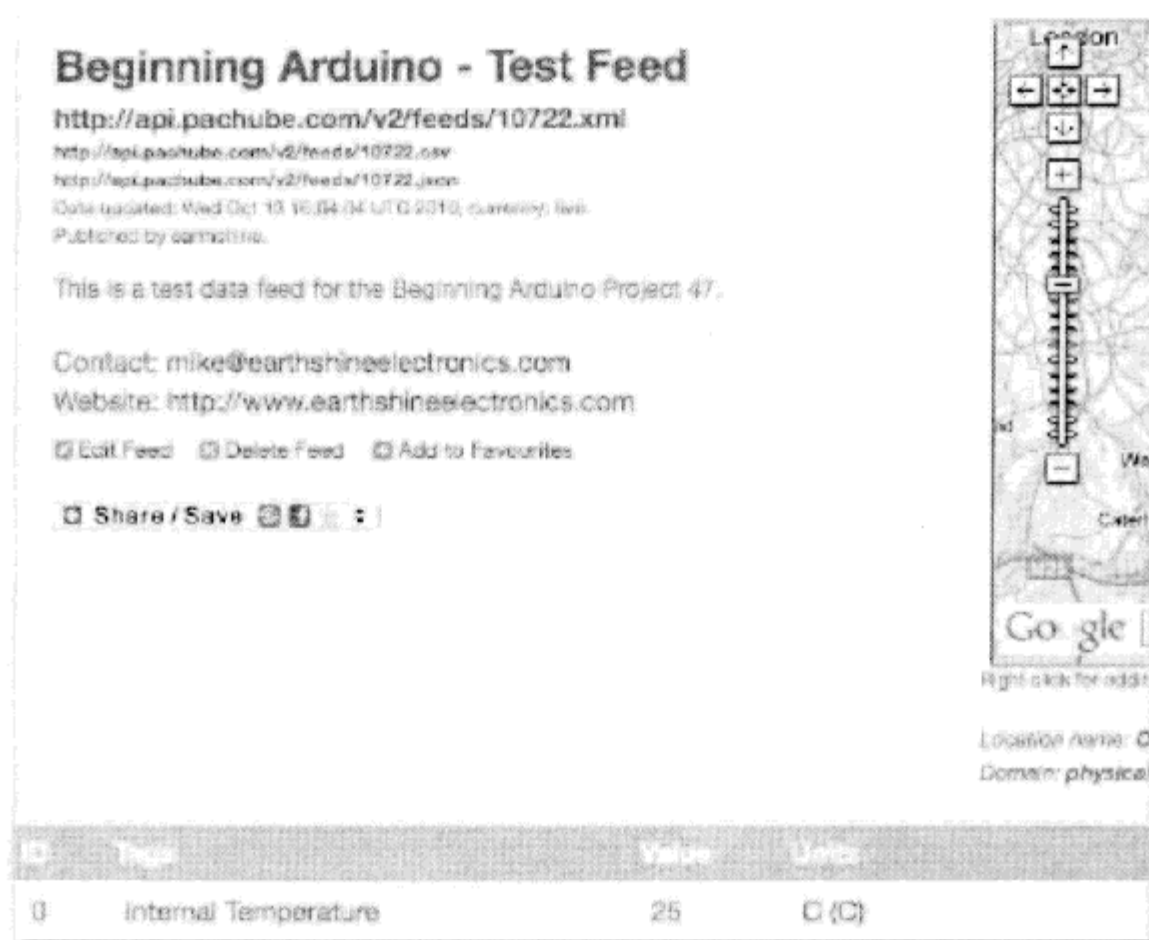


图 17-5 数据源号在标题下边

Your Master API Key

This is your master API key, for accessing Pachube feed data and updating manual feeds (keep this private):

9cd7a134fca4038bc103458b9337654dac3088a0d6ee862b8381cd03fb3913d

[Regenerate API key](#)

图 17-6 主 API 键

现在，你已经有了那些重要的信息，可以输入代码了。

输入代码

输入清单 17-2 中的代码，感谢 Usman Hague 在 Pachube 对这个项目的帮助。

清单 17-2 项目 47 的代码

```
//项目 47——基于 Pachube 网站的 Arduino 项目
#include <SPI.h>
```

```

#include <Ethernet.h>
#include <OneWire.h>
#include <DallasTemperature.h>

#define SHARE_FEED_ID 10722 // 这是 Pachube 数据源 ID
#define UPDATE_INTERVAL 10000 // 如果连接正常, 在下一次更新前等待 10 秒, 不要少于 5 秒
// 如果连接失败或重启, 在重新尝试连接前等待 10 秒, 不要少于 5 秒
#define RESET_INTERVAL 10000
#define PACHUBE_API_KEY "066ed6eald1073600e5b44b35e8a399697d66532c3e736c77dc-11123dfbfel2f" // 填上你自己的 API 号

// 定义要用来从传感器中读取数据的数字引脚
#define ONE_WIRE_BUS 3
#define TEMPERATURE_PRECISION 12

// 生成一个 oneWire 实例连接任意单线设备 (不仅仅是 Maxim/Dallas 温度传感器)
OneWire oneWire(ONE_WIRE_BUS);

// 传递 oneWire 引用给 Dallas 温度传感器
DallasTemperature sensors(&oneWire);

// 存储元件地址的数组
DeviceAddress insideThermometer = { 0x10, 0x7A, 0x3B, 0xA9, 0x01, 0x08, 0x00, 0xBF };
DeviceAddress outsideThermometer = { 0x10, 0xCD, 0x39, 0xA9, 0x01, 0x08, 0x00, 0xBE };

byte mac[] = { 0xCC, 0xAC, 0xBE, 0xEF, 0xFE, 0x91 }; // 确保这在你的网络上没有重复
byte ip[] = { 192, 168, 0, 104 }; // 没有 DHCP 所以你设置自己的 IP 地址
byte remoteServer[] = { 173, 203, 98, 29 }; // pachube.com 的 IP 地址

Client localClient(remoteServer, 80);

unsigned int interval;
char buff[64];
int pointer = 0;
char pachube_data[70];
char *found;
boolean ready_to_update = true;
boolean reading_pachube = false;
boolean request_pause = false;
boolean found_content = false;
unsigned long last_connect;
int content_length;
int itempC, itempF, etempC, etempF;

```



```

void setupEthernet() {
    resetEthernetShield();
    delay(500);
    interval = UPDATE_INTERVAL;
    Serial.println("setup complete");
}

void clean_buffer() {
    pointer = 0;
    memset(buff, 0, sizeof(buff));
}

void resetEthernetShield() {
    Serial.println("reset ethernet");
    Ethernet.begin(mac, ip);
}

void pachube_out() {
    getTemperatures();
    if (millis() < last_connect) last_connect = millis();

    if (request_pause) {
        if ((millis() - last_connect) > interval) {
            ready_to_update = true;
            reading_pachube = false;
            request_pause = false;
        }
    }

    if (ready_to_update) {
        Serial.println("Connecting...");
        if (localClient.connect()) {

            sprintf(pachube_data, "%d,%d,%d,%d", itempC, itempF, etempC, etempF);
            Serial.print("Sending: ");
            Serial.println(pachube_data);
            content_length = strlen(pachube_data);

            Serial.println("Updating.");
            localClient.print("PUT /v1/feeds/");
            localClient.print(SHARE_FEED_ID);
            localClient.print(".csv HTTP/1.1\nHost: api.pachube.com\nX-Pachube ApiKey: ");
            localClient.print(PACHUBE_API_KEY);
            localClient.print("\nUser-Agent: Beginning Arduino - Project 47");
        }
    }
}

```

```

    localClient.print("\nContent-Type: text/csv\nContent-Length: ");
    localClient.print(content_length);
    localClient.print("\nConnection: close\n\n");
    localClient.print(pachube_data);
    localClient.print("\n");

    ready_to_update = false;
    reading_pachube = true;
    request_pause = false;
    interval = UPDATE_INTERVAL;

}
else {
    Serial.print("connection failed!");
    ready_to_update = false;
    reading_pachube = false;
    request_pause = true;
    last_connect = millis();
    interval = RESET_INTERVAL;
    setupEthernet();
}
}

while (reading_pachube){
    while (localClient.available()) {
        checkForResponse();
    }
    if (!localClient.connected()) {
        disconnect_pachube();
    }
}

}

void disconnect_pachube(){
    Serial.println("disconnecting.\n=====\n\n");
    localClient.stop();
    ready_to_update = false;
    reading_pachube = false;
    request_pause = true;
    last_connect = millis();
    resetEthernetShield();
}

void checkForResponse(){

```

```

    char c = localClient.read();
    buff[pointer] = c;
    if (pointer < 64) pointer++;
    if (c == '\n') {
        found = strstr(buff, "200 OK");
        buff[pointer]=0;
        clean_buffer();
    }
}

//从 DS18B20 中获得温度
void getTemperatures()
{
    sensors.requestTemperatures();
    itempC = sensors.getTempC(insideThermometer);
    itempF = DallasTemperature::toFahrenheit(itempC);
    etempC = sensors.getTempC(outsideThermometer);
    etempF = DallasTemperature::toFahrenheit(etempC);
}

void setup()
{
    Serial.begin(57600);
    setupEthernet();
    //开始使用传感器
    sensors.begin();
    //设定传感器的精度
    sensors.setResolution(insideThermometer, TEMPERATURE_PRECISION);
    sensors.setResolution(outsideThermometer, TEMPERATURE_PRECISION);
}

void loop()
{
    pachube_out();
}

```

上传代码，之后打开串口监视器，如果每件事情都做得正确的话你将成功地向 Pachube 传送数据，输出看起来如下：

```

Reset ethernet
Setup complete
Connecting...
Sending:25,77,25,77

```

```
Updating.
Disconnecting.
=====
```

现在打开你的浏览器，到 www.Pachube.com 导航到你的数据源页面，网摘的最后更新日期和时间显示在一个绿色的“live”按钮下方，按钮显示网摘是活跃的并且可以接收数据（见图 17-7）。图 17-7 可以显示整个时间过程的温度。如果你运行这个项目一个相当长的时间，应该可以看到整天的温度变化。



图 17-7 一个活跃的 Pachube 网摘页

单击上面图形上的按钮，将给出最后 1 小时、最近 24 小时、最近 4 天和最近 3 个月的数据。如果你单击图形，将打开一个新的页面显示全部的数据。这些图形可以包含在你自己的网页中来提供更新的数据源数据。在 Pachube 程序库中还有其他图形形式，这些程序允许你从手机上看到这数据，可以自己定义这个图形的大小、颜色、灰度形式等。

你可以改变代码，增加其他传感器，例如，我们在项目 31 中使用的压力传感器、测量周围光线的光传感器、湿度传感器和风速传感器，使用你自己的在 Internet 上的 Pachube 网站获得的网摘去做成一个完整成熟的气象站。

现在让我们看一下这个项目的代码，看看它是如何工作的。

代码回顾

代码开始于包含 Ethernet 板和单线温度传感器：

```
#include <SPI.h>
#include <Ethernet.h>
#include <OneWire.h>
#include <DallasTemperature.h>
```

之后是数据源号。你需要改变这个数使它与你自己的数据源 ID 相同。

```
#define SHARE_FEED_ID 10722
```

之后定义的是用毫秒表示的数据上传到 Pachube 的时间间隔，以及当发生连接错误时等待的时间。要保证这两个时间不少于 5 秒，否则会出现“API 速率限制报警”。如果你需要更高的更新速度，可以购买一个付费的页面。

```
#define UPDATE_INTERVAL 10000
#define RESET_INTERVAL 10000
```

之后你需要定义 API 主键，这个键就是“my setting”页面中的那个，从页面复制并把它粘贴到代码中：

```
#define PACHUBE_API_KEY "066ed6ea1d1073600e5b44b35e8a399697d66532c3e736c77d  
c11123dfbfe12f"
```

之后两个定义分别是温度传感器引脚和精度设置：

```
#define ONE_WIRE_BUS 3
#define TEMPERATURE_PRECISION 12
```

接下来生成一个单线总线实例和一个 DallasTemperature 对象:

```
OneWire oneWire(ONE_WIRE_BUS);
DallasTemperature sensors(&oneWire);
```

之后两个 DS17B20 传感器的序列号设置:

```
DeviceAddress insideThermometer = { 0x10, 0x7A, 0x3B, 0xA9, 0x01, 0x08, 0x00,
0xBF };
DeviceAddress outsideThermometer = { 0x10, 0xCD, 0x39, 0xA9, 0x01, 0x08, 0x00,
0xBE};
```

之后需要输入你的 Ethernet 板的 Mac 和 IP 地址:

```
byte mac[] = { 0xCC, 0xAC, 0xBE, 0xEF, 0xFE, 0x91 }; //确保这在你的网络上没有重复
byte ip[] = { 192, 168, 0, 104 }; //没有 DHCP 所以你设置自己的 IP 地址
```

之后是远程服务器的 IP 地址 (这里 IP 地址是 pachube.com 的 IP 地址):

```
byte remoteServer[] = { 173, 203, 98, 29 };
```

之后生成一个客户机实例传递 Pachube 的 IP 地址和端口 80 给它:

```
Client localClient(remoteServer, 80);
```

之后是你将用于这个程序的全部变量。开始是 interval, 用毫秒表示的在更新数据和尝试连接之间的时间间隔:

```
unsigned int interval;
```

一个数组存储从 Pachube 中返回的字符串:

```
char buff[64];
```

给上面数组的一个指针或索引:

```
int pointer = 0;
```


一个数组存储要发送到 Pachube 的字符串（如果你想送出更多的字符可以把该字符串定义得更长一些）：

```
char pachube_data[70];
```

一个存储字符串对比函数结果的变量，也就是后面你检查数据是否被数据源正确接收时的结果：

```
char *found
```

一系列从变量名就能看出含义的布尔变量：

```
boolean ready_to_update = true;
boolean reading_pachube = false;
boolean request_pause = false;
boolean found_content = false;
```

用毫秒表示的最后一次连接时间。用它与你已经设置的内部时间和 millis() 的值进行对比来决定是否需要做一次推送数据：

```
unsigned long last_connect;
```

要送到 Pachube 的数据串的长度：

```
int content_length;
```

之后是用摄氏度和华氏度表示的来自于传感器的温度。

```
int itempC, itempF, etempC, etempF;
```

接下来是在整个解决方案中用到的一系列函数中的第一个，setupEthernet()，它重置 Ethernet 连接和更新数据的中断设置：

```
void setupEthernet() {
  resetEthernetShield();
  delay(500);
  interval = UPDATE_INTERVAL;
  Serial.println("setup complete");
}
```

之后是 `clean_buffer()` 函数，清除并用 0 填充缓冲区：

```
void clean_buffer() {
```

首先，指针（数组的索引）设置为 0：

```
pointer = 0;
```

之后，`memset` 函数用数字 0 来填充缓冲区：

```
memset(buff, 0, sizeof(buff));
```

`memset` 函数是个新函数，它的作用是设置在存储空间内一定字节的存储区为一个指定值。函数需要三个参数，一个是指向要填充的空白存储空间的指针，一个是要设定的值，另外一个是要设置字节的数量。

在这个例子里，你传递给它 `buffer` 数组名作为第一个参数，因此它指向在存储空间内第一个字节，在那里存储 `buffer` 数组。之后写 0 到存储空间块中的每个字节，块的数量是 `sizeof(buff)`。`sizeof` 函数返回数组的字节数。

`memset` 函数的作用是用 0 填充 `buffer` 数组空间，因此它清空之前 `buffer` 内所有的数据。

之后到达 `resetEthernetShield()` 函数，调用这个函数重置以太网模板。

```
void resetEthernetShield() {
  Serial.println("reset ethernet");
  Ethernet.begin(mac, ip);
}
```

之后到达一个重要的函数，它的作用是传送传感器数据到 Internet 的 Pachube 数据源页面：

```
void pachube_out() {
```

开始于调用 `getTemperatures()` 函数，因此得到两个传感器的读数：

```
getTemperatures();
```

之后，检查在 `millis()` 中的当前时间是否小于存储在 `last_connect` 中的值，如果是，

`last_connect` 变量更新到 `millis()` 的当前值。每次与 Pachube 断开连接，在 `last_connect` 中的值将更新为 `millis()` 中的值。用这个值去表示从最后一次连接起多少毫秒的时间已经过去。

```
if (millis() < last_connect) last_connect = millis();
```

之后运行一个 `if` 语句，条件是 `request_pause` 是否为 `true`，如果连接失败或已经从 Pachube 断开，这个变量被设置为 `true`。

```
if (request_pause) {
```

在 `if` 语句的内部检查 `last_connect` 当前值减去 `millis()` 中的值是否大于 `interval` 的值，如果是，那么间隔时间已经过去，相应地设置三个标志为 `true` 或 `false`。

```
if ((millis() - last_connect) > interval) {
    ready_to_update = true;
    reading_pachube = false;
    request_pause = false;
}
```

这些标志告诉 `pachube_out()` 函数，自从你最后一次连接或尝试连接起规定的间隔时间已经过去，因此准备尝试下一次更新，如果已准备好更新，下一个 `if` 语句运行。

```
if (ready_to_update) {
```

它开始于通知用户已经连接：

```
Serial.println("Connecting...");
```

之后检查是否已经成功连接到客户机：

```
if (localClient.connect()) {
```

如果是，运行其余的代码来更新数据源。首先使用 `sprintf` 函数去打印一个字符串格式的数据。`sprintf` 函数（字符串打印格式）是打包大量不同格式的信息到字符串的一个好方法。

```
sprintf(pachube_data, "%d,%d,%d,%d", itempC, itempF, etempC, etempF);
```

它需要三个参数：变量，你将在这个变量中存储格式化数据（在这个例子里，变量是 `pachube_data`）；指定的字符串连接格式；之后是原变量。函数所做的是插入第一个变量为字符串，在第二个 `%d` 处插入第二个变量等。4 个格式符号用逗号分隔，因此在字符串中这 4 个变量表示的数将用逗号分隔，如果变量的值是：

```

itempC      25
itempF      77
etempC      14
etempF      52

```

那么，运行 `sprintf` 函数后，`pachube_data` 中的内容将是：

```
"25, 77, 14, 52"
```

如果 `sprintf` 函数是：

```

sprintf(pachube_data, "Internal Temps:%d,%d External Temps:%d,%d", itempC,
itempF, etempC, etempF);

```

那么 `pachube_data` 将是：

```
"Internal Temps:25,77 External Temps:14,52"
```

就像你看到的，`sprintf` 函数是转化长的混合字符串和把数字混合到一个字符串中有力的工具。

之后，通知用户你要送出什么数据：

```

Serial.print("Sending: ");
Serial.println(pachube_data);

```

使用 `strlen()` 函数算出 `pachube_data` 中字符串的长度：

```
content_length = strlen(pachube_data);
```

通知用户你要更新数据源：

```
Serial.println("Updating.");
```

之后，通过建立一个将要送到数据源 URL 的字符串实现打印数字到 localClient。字符串的第一个部分是数据源的 URL，包含数据源的 ID，接下来是 API 键。

```
localClient.print("PUT /v1/feeds/");
localClient.print(SHARE_FEED_ID);
localClient.print(".csv HTTP/1.1\nHost: api.pachube.com\nX-PachubeApiKey: ");
localClient.print(PACHUBE_API_KEY);
```

接下来是用户名称，以及你要送出的数据的长度，接下来是把传感器字符串数据转到 csv（逗号分隔）文件。

之后，送出用户代理命令。它是一个 HTTP 命令，通常用来确定正在作为客户机使用的是什软件，这个命令是客户机-服务器通信的一部分。客户机-服务器需要头文件包含关于用户代理组织要求的信息。这个信息不是必要的，但是为了在编写客户代码时保持好的习惯最好写这个信息。

```
localClient.print("\nUser-Agent: Beginning Arduino - Project 47");
```

之后，告诉客户机连接是什么形式的和它的长度。在你的例子里，它是文本文件，csv 文件格式。

```
localClient.print("\nContent-Type:text/csv\nContent-Length: ");
localClient.print(content_length);
localClient.print("\nConnection: close\n\n");
```

最后，用逗号分隔传感器数据字符串：

```
localClient.print(pachube_data);
localClient.print("\n");
```

标志全部设置回它们的默认值：

```
ready_to_update = false;
reading_pachube = true;
request_pause = false;
interval = UPDATE_INTERVAL;
}
```



如果你不能够成功连接，告诉用户标志设置为与以上相同，之后重启网络连接。

```
else {
    Serial.print("connection failed!");
    ready_to_update = false;
    reading_pachube = false;
    request_pause = true;
    last_connect = millis();
    interval = RESET_INTERVAL;
    setupEthernet();
}
```

下一步是一个 while 语句检查 reading_pachube 是否为 true，如果是，它检查 localClient 值是否有效，例如，从 Pachube 获得了反馈，while 语句调用 checkForResponse()、disconnect_pachube() 函数。

```
while (reading_pachube) {
    while (localClient.available()) {
        checkForResponse();
    }
    if (!localClient.connected()) {
        disconnect_pachube();
    }
}
```

disconnect_pachube() 函数通知用户没有连接，停止本地客户机，设置标志到它们的默认值，重置网络。

```
void disconnect_pachube() {
    Serial.println("disconnecting.\n=====\n\n");
    localClient.stop();
    ready_to_update = false;
    reading_pachube = false;
    request_pause = true;
    last_connect = millis();
    resetEthernetShield();
}
```


`checkForResponse()`函数的作用是检查客户机 (Pachube) 是否已经送回一个“200 OK”命令。它告诉你传送数据已经成功。之后它检查你在字符串结尾是否已经送出一个换行 (`\n`)，如果是，为下一次传递数据做好准备，清除缓冲区。

```
void checkForResponse() {
```

`read()`函数用于从客户机中接收一个字节并存储它到变量 `c` 中：

```
char c = localClient.read();
```

接收到的字节存储在 `buff` 数组中：

```
buff[pointer] = c;
```

如果还没有超过 64 字节长度 (数组的大小)，那么索引值增加：

```
if (pointer < 64) pointer++;
```

检查 `c` 中的值看是否已经收到字符串的尾部。

```
if (c == '\n') {
```

如果是，使用 `strstr` 函数确定“200 OK”是否出现在字符串里，返回它的位置指针，把指针存储在 `found` 中。

```
found = strstr(buff, "200 OK");
```

`strstr` 函数找出在另一个字符串中的子字符串，它需要两个参数。第一个是你要检查的字符串，第二个是你希望定位的子字符串。如果发现子字符串，它返回子字符串的位置，如果没有发现，函数返回一个 `null` 指针。

之后重置并清除缓冲区：

```
buff[pointer]=0;
clean_buffer();
```

下面是最后的函数，从 DS18B20 中获得温度：

```

void getTemperatures()
{
    sensors.requestTemperatures();
    itempC = sensors.getTempC(insideThermometer);
    itempF = DallasTemperature::toFahrenheit(itempC);
    etempC = sensors.getTempC(outsideThermometer);
    etempF = DallasTemperature::toFahrenheit(etempC);
}

```

在你已经定义全部函数后，到达 `setup` 函数：

```
void setup()
```

它用 57600 波特率开始串行通信：

```
Serial.begin(57600);
```

之后调用 `setupEthernet()` 函数：

```
setupEthernet();
```

开始获得单线传感器数据和设定传感器的精度：

```

sensors.begin();
sensors.setResolution(insideThermometer, TEMPERATURE_PRECISION);
sensors.setResolution(outsideThermometer, TEMPERATURE_PRECISION);

```

剩下的是主循环，它只是一遍一遍地调用 `pachube_out` 这个函数。

```

void loop()
{
    pachube_out();
}

```

现在你知道了送传感器数据到 Pachube 存储和浏览的基本方法，改变代码去增加传感器和其他数字对你来说是相当简单的任务。

到目前为止，你已经学习了如何通过 Ethernet 送数据到网络上的一个网页，和到 Pachube 网站上获得数据存储和表格显示服务。之后，你将学习如何使用 Arduino 发送 E-mail 来通知你温度过高或过低。

练 习

在 C 语言中，使用 %f 来格式化打印浮点数。然而，在 Arduino 版本的 C 语言中没有这个功能，只有整数可以打印。改变送温度值的代码，送出浮点数。提示：在这本书中你已学习了如何人工转变数字为字符串。

项目 48——电子邮件提醒系统

现在再介绍一个不同的数据传送方法。在这个项目中，当温度太低或太高时你用带网卡板的 Arduino 发送一个 E-mail 来报警。设计这个项目是为了演示通过网卡板传送 E-mail 的基本知识。本项目使用与上一个项目相同的电路，但是只用一个温度传感器。

输入代码

从清单 17-3 中输入代码，需要获得你的 SMTP 电子邮件服务器的 IP 地址。为了得到 IP 地址，打开你的 window 终端（window 命令，控制面板，或其他你知道的在你系统上的程序），输入 ping 命令，跟着是你希望获得的 IP 地址的网址，换句话说，如果你想知道在 smtp.live.com 网站上的 Hotmail SMTP 邮件服务器的 IP 地址，你应输入：

```
ping smtp.live.com
```

命令返回如下：

```
PING smtp.hot.glbdns.microsoft.com(65.55.162.200):56data bytes
```

这表明 IP 地址是 65.55.162.200，把它作为你的 E-mail 服务器的 SMTP 服务器地址，输入到服务器代码块中。如果你的 SMTP 服务器需要身份认证。你需要获得 Base-64 版本的用户名和口令。有许多网站可以为你做这个译码工作。例如：

```
www.motobit.com/util/base64-decoder-encoder.asp
```

输入你的用户名将它译成 base-64 密文形式，之后对你的口令做同样的事情。复制粘贴译码结果到相应的代码段。也要改变 FROM 和 TO 代码段中的代码，换成你自己的 E-mail 地址或电子邮件接收地址。

清单 17-3 项目 48 的代码

```

#include <Ethernet.h>
#include <SPI.h>
#include <OneWire.h>
#include <DallasTemperature.h>
#define time 1000
#define emailInterval 60
#define HighThreshold 40 //高温告警门限
#define LowThreshold 10 //低温告警门限

//定义要用来从传感器中读取数据的数字引脚
#define ONE_WIRE_BUS 3
#define TEMPERATURE_PRECISION 12

float tempC, tempF;
char message1[35], message2[35];
char subject[] = "ARDUINO: TEMPERATURE ALERT!!\0";
unsigned long lastMessage;

//生成一个 oneWire 实例连接任意单线设备
Setup a oneWire instance to communicate with any OneWire devices
OneWire oneWire(ONE_WIRE_BUS);

//传递 oneWire 引用给 Dallas 温度传感器
DallasTemperature sensors(&oneWire);

//存储元件地址的数组
DeviceAddress insideThermometer = { 0x10, 0x7A, 0x3B, 0xA9, 0x01, 0x08, 0x00,
0xBF };

byte mac[] = { 0x64, 0xB9, 0xE8, 0xC3, 0xC7, 0xE2 };
byte ip[] = { 192, 168, 0, 105 };
//电子邮件服务器的 IP 地址, 必须改为你自己的邮件服务器的 IP 地址
byte server[] = { 62, 234, 219, 95 };
Client client(server, 25);

void sendEmail(char subject[], char message1[], char message2[], float temp) {
    Serial.println("connecting...");

    if (client.connect()) {
        Serial.println("connected");
        client.println("EHLO MYSERVER"); delay(time); // 登录

```

```

    client.println("AUTH LOGIN"); delay(time); //授权
    //在这里输入你自己的用户名
    client.println("caFzLmNvbQaWNZXGluZWVsZWNOcm9uNAZW2FsydGhzd3"); delay(time);
    //在这里输入密码
    client.println("ZnZJh4TYZ2ds"); delay(time);
    client.println("MAIL FROM:<alert@bobsmith.org>"); delay(time);
    client.println("RCPT TO:<fred@bloggs.com>"); delay(time);
    client.println("DATA"); delay(time);
    client.println("From: <alert@bobsmith.org>"); delay(time);
    client.println("To: <fred@bloggs.com>"); delay(time);
    client.print("SUBJECT: ");
    client.println(subject); delay(time);
    client.println(); delay(time);
    client.println(message1); delay(time);
    client.println(message2); delay(time);
    client.print("Temperature: ");
    client.println(temp); delay(time);
    client.println("."); delay(time);
    client.println("QUIT"); delay(time);
    Serial.println("Email sent.");
    lastMessage=millis();
} else {
    Serial.println("connection failed");
}

}

void checkEmail() { //看是否有来自客户机的数据可用
    while (client.available()) {
        char c = client.read();
        Serial.print(c);
    }

    if (!client.connected()) {
        Serial.println();
        Serial.println("disconnecting.");
        client.stop();
    }
}

//从传感器中获得温度数据的函数
void getTemperature(DeviceAddress deviceAddress)

```

```

{
    tempC = sensors.getTempC(deviceAddress);
    tempF = DallasTemperature::toFahrenheit(tempC);
}

void setup()
{
    lastMessage = 0;
    Ethernet.begin(mac, ip);
    Serial.begin(9600);

    //初始化传感器
    sensors.begin();
    //设置精度
    sensors.setResolution(insideThermometer, TEMPERATURE_PRECISION);

    delay(1000);
}

void loop()
{
    sensors.requestTemperatures();
    getTemperature(insideThermometer);
    Serial.println(tempC);
    //温度是否过高
    if (tempC >= HighThreshold && (millis() > (lastMessage + (emailInterval * 1000)))) {
        Serial.println("High Threshold Exceeded");
        char message1[] = "Temperature Sensor\0";
        char message2[] = "High Threshold Exceeded\0";
        sendEmail(subject, message1, message2, tempC);
    } //温度过低
    else if (tempC <= LowThreshold && (millis() > (lastMessage + (emailInterval * 1000))))
        Serial.println("Low Threshold Exceeded");
        char message1[] = "Temperature Sensor\0";
        char message2[] = "Low Threshold Exceeded\0";
        sendEmail(subject, message1, message2, tempC);
    }

    if (client.available()) {checkEmail();}
}

```

上传代码，然后打开串口监视器窗口。串口监视器将一遍一遍地显示来自第一个传感

器的温度。如果温度降低到 LowThreshold 值以下，串口监视器将显示“Low Threshold Exceeded”，之后发送相关的电子邮件告警。如果温度值高于 HighThreshold，它将显示“High Threshd Excoded”，送出高温状态电子邮件告警。

可以人工检查这个动作是否正确，设置 highthreslode 为稍高于环境温度之后用手握住传感器直到温度上升到门限之上，设置的告警系统就会工作。

注意系统头 60 秒将忽略任何温度告警。它仅在 60 秒过后才会开始传送告警。如果温度已经超过了门限，告警系统将一直发送电子邮件直到温度下降到可接受的水平。当设定的报警门限已经超过时，每隔 emailInterval 秒送出一封告警电子邮件，你可以根据自己的需要调整时间间隔。

在一封电子邮件已送出后，系统将等待直到成功接收返回的信息，之后将显示反馈信息。如果系统没有正确工作，你可以使用这个数据来调试系统。

代码回顾

首先包含库：

```
#include <Ethernet.h>
#include <SPI.h>
#include <OneWire.h>
#include <DallasTemperature.h>
```

之后，定义以毫秒为单位的送出数据到服务器的延时时间：

```
#define time 1000
```

接下来是以秒为单位的时间——发送电子邮件的时间间隔。

```
#define emailInterval 60
```

之后，需要设定将引起告警的温度高低门限：

```
#define HighThreshold 40 //高温告警门限
#define LowThreshold 10 //低温告警门限
```

之后，设置传感器的引脚和精度：

```
#define ONE_WIRE_BUS 3
#define TEMPERATURE_PRECISION 12
```

之后，是存储温度的浮点数：

```
float tempC, tempF;
```

之后，是两个字符数组，存储电子邮件中的信息：

```
char message1[35], message2[35];
```

还有其他的字符数组存储电子邮件的标题，这个字符数据在定义时就被初始化：

```
char subject[] = "ARDUINO: TEMPERATURE ALERT!!\0";
```

因为当门限超过时你不希望受到电子邮件信息轰炸，所以需要存储最后一封邮件送出的时间。它将存储在一个无符号整数中，叫做 `lastMessage`：

```
unsigned long lastMessage;
```

建立传感器实例和传感器的地址：

```
OneWire oneWire(ONE_WIRE_BUS);
DallasTemperature sensors(&oneWire);
DeviceAddress insideThermometer = { 0x10, 0x7A, 0x3B, 0xA9, 0x01, 0x08, 0x00,
0xBF };
```

`mac` 和 `ip` 是指 Ethernet 板的 Mac 地址和 IP 地址：

```
byte mac[] = { 0x64, 0xB9, 0xE8, 0xC3, 0xC7, 0xE2 };
byte ip[] = { 192, 168, 0, 105 };
```

之后，设置你的电子邮件服务器的 IP 地址。它必须要设成你自己的，否则代码无法正常工作。

```
byte server[] = { 62, 234, 219, 95 };
```

生成一个客户机实例，传递给它服务器地址和端口号 25，如果你的 SMTP 服务器使用不同的端口号，需要改变端口号为你的服务器使用的端口号。

```
Client client(server, 25);
```

接下来是你自己定义的第一个函数。这个函数做的工作是送出电子邮件到服务器，这个函数需要 4 个参数：电子邮件主题、信息的第一行、信息的第二行、最后的温度。

```
void sendEmail(char subject[], char message1[], char message2[], float temp) {
```

通知用户正在尝试连接：

```
Serial.println("connecting...");
```

之后，检查客户机是否已连接，如果是，执行在 if 语句块中的代码。

```
if (client.connect()) {
```

首先，用户得到已经连接到客户机的通知。客户机在这里指的是你的邮件服务器。

```
Serial.println("connected");
```

现在送出命令到服务器，与项目 46 中使用的方法十分相似。首先你必须对邮件接收服务器介绍自己，用 EHLO 命令和其他一些服务器的细节技术命令实现。每个命令发出之后，你必须等一会儿允许命令执行。我发现我的服务器要 1000 毫秒。你需要相应地增加或减少这个数值。

```
client.println("EHLO MYSERVER"); delay(time); //登录
```

这像是一个在服务器和客户机之间的握手过程。它们互相之间介绍自己。之后你要授权连接，如果你的 SMTP 服务器不需要授权，你可以注释掉这一行和用户名、口令行。

```
client.println("AUTH LOGIN"); delay(time); //授权
```

有时服务器需要一个非密登录，在这种情况下，你应该用明码符号形式送出 AUTH PLAN 和用户名、口令。

之后需要送到服务器的是 Base-64 加密用户名和口令：

```
client.println("caFzLmNvbQaWNZXGluZWVsZWNOcm9uNAZW2FsydGhzd3"); delay(time);
client.println("ZnZJh4TYZ2ds"); delay(time);
```

之后你需要告诉服务器这个邮件来自于谁，这个邮件要送给谁：

```
client.println("MAIL FROM:<alert@bobsmith.org>");    delay(time);
client.println("RCPT TO:<fred@bloggs.com>");        delay(time);
```

这次必须改成你自己的电子邮件地址和接收地址。许多 SMTP 服务器只允许你发送电子邮件时使用它自己的电子邮件地址或从它自己的域中发送（例如，你不能使用 Yahoo 服务器从 Hotmail 账号发送邮件）。

DATA 命令告诉服务器之后到来的是电子邮件的数据，也就是邮件接收者可以看到的東西。

```
client.println("DATA");        delay(time);
```

你希望接收者看到这是谁的电子邮件和电子邮件来自哪里，为了接收者更好地阅读邮件，再次包含以下这些信息。

```
client.println("From: <alert@bobsmith.org>");    delay(time);
client.println("To: <fred@bloggs.com>");        delay(time);
```

之后，发送电子邮件的标题，使用词“**SUBJECT:**”，跟着是送出主题的函数：

```
client.print("SUBJECT: ");
client.println(subject);    delay(time);
```

在发送邮件内容之前，你必须从一个空白行开始：

```
client.println();    delay(time);
```

之后是两行信息传递函数：

```
client.println(message1);    delay(time);
client.println(message2);    delay(time);
```

之后包含温度数据：

```
client.print("Temperature: ");
client.println(temp);    delay(time);
```

所有电子邮件必须结束于一个单独占一行的点符号，告诉服务器邮件已经结束：

```
client.println(".");    delay(time);
```

之后，送出一个 QUIT 命令取消服务器的连接：

```
client.println("QUIT");    delay(time);
```

通知使用者电子邮件已发送：

```
Serial.println("Email sent.");
```

之后，存储 `millis()` 中的当前值到 `lastMessage` 变量，因为你在后面要用它判断指定的时间间隔是否已经过去或正在发送信息。

```
lastMessage=millis();
```

如果与客户机连接失败，电子邮件不会被送出并通知使用者：

```
} else {  
    Serial.println("connection failed");  
}
```

之后是从客户机读返回信息的函数：

```
void checkEmail() { //看是否有来自客户机的数据可用
```

如果从客户机返回信息中读到可用数据：

```
while (client.available()) {
```

你存储这个字节到变量 `c` 中：

```
char c = client.read();
```

之后打印它到串口监视器上：

```
Serial.print(c);
```

如果客户机没有连接:

```
if (!client.connected()) {
```

那么通知使用者, 系统断开连接, 停止与客户机连接:

```
Serial.println();
Serial.println("disconnecting.");
client.stop();
```

下面的函数已经在前面用到过, 是从单线传感器中获得温度数据的函数:

```
void getTemperature(DeviceAddress deviceAddress)
{
    tempC = sensors.getTempC(deviceAddress);
    tempF = DallasTemperature::toFahrenheit(tempC);
}
```

接下来是 `setup` 函数, 它仅仅初始化网络和传感器:

```
void setup()
{
    Ethernet.begin(mac, ip);
    Serial.begin(9600);
    //初始化传感器
    sensors.begin();
    //设置精度
    sensors.setResolution(insideThermometer, TEMPERATURE_PRECISION);
    delay(1000);
}
```

最后, 是主程序循环:

```
void loop()
```

主循环开始于使用 `DallasTemperture` 库中的方法从传感器获得温度数据:

```
sensors.requestTemperatures();
```

之后, 以传感器地址为参数调用 `getTemperature` 函数:


```
getTemperature(insideThermometer);
```

之后，把温度显示在串口显示器上：

```
Serial.println(tempC);
```

之后，检查温度值看它是否已经达到或超过设定的高门限。如果是，送出电子邮件。但是只是每（`emailInterval*1000`）秒送出一封邮件，因此检查 `millis()` 是否大于电子邮件已送出的最后时间加上间隔时间。如果是，代码执行。

```
if (tempC >= HighThreshold && (millis() > (lastMessage + (emailInterval * 1000)))) {
```

通知使用者，之后两行数组为电子邮件要送出的信息：

```
Serial.println("High Threshold Exceeded");  
char message1[] = "Temperature Sensor\0";  
char message2[] = "High Threshold Exceeded\0";
```

调用 `sendEmail` 函数，传递给它的参数是标题、第一和第二行信息和当前的温度：

```
sendEmail(subject, message1, message2, tempC);
```

如果高温度门限还没有到，传递给它的参数是标题、第一和第二信息行和当前的温度。

```
else if (tempC <= LowThreshold && (millis() > (lastMessage + (emailInterval *  
1000))))  
    Serial.println("Low Threshold Exceeded");  
    char message1[] = "Temperature Sensor\0";  
    char message2[] = "Low Threshold Exceeded\0";  
    sendEmail(subject, message1, message2, tempC);  
}
```

最后，检查是否有任何从客户机返回（在一封电子邮件已送出）的数据已准备好接收，若有显示结果。

```
if (client.available()) {checkEmail();}
```

这个数据对于调试程序是有用的。

这个项目已经介绍了基本的从 Arduino 网卡板送出一封电子邮件的知识。可以使用这些技术在事件发生时发送告警或报告，例如，检查到有人进入房间或盒子被打开等事件发生时报警。这个系统也可以用于产生其他动作，例如，如果室内的温度太高则打开窗户，或者如果鱼缸中的水位太低则向鱼缸中加水。

下面你将学习如何从 Arduino 中发送数据到微博。

项目 49——微博机器人

在这个项目中还是需要使用两个温度传感器的电路。这次，你要有规律地在微博上更新两个传感器的状态。用发送微博的办法使你可以了解已经连接到 Arduino 上的任何传感器的状态。

Twitter 是一条微博服务器，允许你发送长度在 140 个字符以内的微博或短消息。任何人只要进行一下搜索，或那些已选择了关注你的微博的人（你的粉丝），都可以看到你发布的微博的内容。微博是非常流行的网络应用，可以从任何网络浏览器或许多可用的微博客户机中浏览微博内容，甚至电话终端也可以发送微博。这使得微博成为发送简单消息的理想选择。你也可以通过移动终端查看微博内容。

你需要连到 Twitter.com 注册一个新用户。我推荐创建一个只被你的 Arduino 使用的账号。

因为 2010 年 8 月 31 日以后，Twitter 改变了它的第三方案入口注册策略，使用了一种叫做 OAuth 的认证方法，使得直接从 Arduino 发送微博非常困难。在这个改变之前，向 Twitter 发送微博是一个简单的过程，现在只能通过第三方实现。换句话说，你发布微博到一个网站或代理服务器，代理服务器使用 OAuth 协议（授权代码）代替你发布微博，Arduino 当前的微博库就使用这种方法。

如果你已有账号，输入下面的代码。

输入代码

在写这本书的时候，Ethernet 库的 Twitter 库仅仅可以在 0018 版本的 ArduinoIDE 上使用，因此你需要访问 Arduino 网站，导航到下载页面获得 0018 版本 IDE，把你当前的版本的 IDE 放在一边，确保你只是从这个版本的 IDE 运行和上传代码。Twitter 库需要使用的 EthernetDNS 和 EthernetDHCP 库可以在网址 <http://gkaindl.com/software/arduino-ethernet> 下

找到。这个库已经升级，适合最新的 IDE 使用，你可以使用这个新的库和 IDE。

在你上传代码之前，需要注册一条微博账号。你使用的这个库已经由 NeaCat 完成，可以使用它的网址作为代理服务器来发送微博。这意味着你必须首先获得一个口令，它是一个你自己的要进入微博网站的用户名和口令的加密版本。要得到这个版本可以去网址 <http://arduino-tweet.appspot.com>，并单击“step1”链接获得口令，复制并粘贴结果到口令代码段。

注意，因为你使用代理服务器而且必须给出你的微博用户名和口令来获得 OAuth 令牌，所以推荐创建一个新的微博账号保持它为匿名。（例如，不要增加任何名字或电子邮件地址到这个账号的微博说明中）。我相信你用自己的账号使用这个库也是安全的，但是相比较而言匿名更安全。

之后，单击“step2”链接，获得两套相关的库代码。安装这些库文件夹到之前你下载和安装的 0018 版本的 Arduino IDE 内。在使用这些库之前需要重启 IDE。Twitter 库也有一些例子可以试一下。如果你想要阅读关于 Twitter 库的资料可以在 Arduino 的开发者基地网址 www.arduino.cc/playground/Code/TwitterLibrary 找到。

如果已经安装了这个库，输入清单 17-4 中的代码，并上传代码。

清单 17-4 项目 49 的代码

//项目 49——微博机器人

```
#include <Ethernet.h>
#include <EthernetDHCP.h>
#include <EthernetDNS.h>
#include <Twitter.h>
#include <OneWire.h>
#include <DallasTemperature.h>

//定义要用来从传感器中读取数据的数字引脚
#define ONE_WIRE_BUS 3
#define TEMPERATURE_PRECISION 12

float itempC, itempF, etempC, etempF;
boolean firstTweet = true;
//生成一个 oneWire 实例连接任意单线设备（不仅仅是 Maxim/Dallas 温度传感器）
OneWire oneWire(ONE_WIRE_BUS);
```

```

//传递 oneWire 引用给 Dallas 温度传感器
DallasTemperature sensors(&oneWire);

//存储元件地址的数组
DeviceAddress insideThermometer = { 0x10, 0x7A, 0x3B, 0xA9, 0x01, 0x08, 0x00,
0xBF };
DeviceAddress outsideThermometer = { 0x10, 0xCD, 0x39, 0xA9, 0x01, 0x08, 0x00,
0xBE};

byte mac[] = { 0x64, 0xB9, 0xE8, 0xC3, 0xC7, 0xE2 };

//连接到微博的 Token 协议 (从 http://arduino-tweet.appspot.com/ 获得)
Twitter twitter("608048201-CxY1yQi8ezhvz60ZVfPHVdzIHbMOD1h2gvoaAIx");

unsigned long interval = 600000; // 10 分钟
unsigned long lastTime; //最后一次发出微博的时间
//发送的信息
char message[140], serialString[60];
//获得温度的函数
void getTemperatures()
{
    itempC = sensors.getTempC(insideThermometer);
    itempF = DallasTemperature::toFahrenheit(itempC);
    etempC = sensors.getTempC(outsideThermometer);
    etempF = DallasTemperature::toFahrenheit(etempC);
}

void tweet(char msg[]) {
    Serial.println("connecting ...");
    if (twitter.post(msg)) {
        int status = twitter.wait();
        if (status == 200) {
            Serial.println("OK. Tweet sent.");
            Serial.println();
            lastTime = millis();
            firstTweet = false;
        } else {
            Serial.print("failed : code ");
            Serial.println(status);
        }
    } else {

```

```

        Serial.println("connection failed.");
    }
}

void setup()
{
    EthernetDHCP.begin(mac);
    Serial.begin(9600);
    sensors.begin();
    //设置精度
    sensors.setResolution(insideThermometer, TEMPERATURE_PRECISION);
    sensors.setResolution(outsideThermometer, TEMPERATURE_PRECISION);

    sensors.requestTemperatures()
    //合成要发布的微博信息字符串
    getTemperatures();
    while (firstTweet) {
        sprintf(message, "Int. Temp: %d C (%d F) Ext. Temp: %d C (%d F). Tweeted from
Arduino. %ld", int(itempC), int(itempF), int(etempC), int(etempF), millis());
        tweet(message);
    }
}

void loop()
{
    EthernetDHCP.maintain();
    sensors.requestTemperatures();
    //合成打印在串口上的字符串
    sprintf(serialString, "Internal Temp: %d C %d F. External Temp: %d C %d F",
int(itempC), int(itempF), int(etempC), int(etempF));
    delay(500);
    Serial.println(serialString);
    Serial.println();

    if (millis() >= (lastTime + interval)) {
        //合成要发布微博信息的字符串
        sprintf(message, "Int. Temp: %d C (%d F) Ext. Temp: %d C (%d F). Tweeted
from Arduino. %ld", int(itempC), int(itempF), int(etempC), int(etempF), millis());
        tweet(message);
    }
    delay(10000); //10 秒
}

```

上传代码到 **Arduino** 之后，打开串口监视器，**Arduino** 尝试连接到 **Twitter**（实际是 **Neocat's** 的网址）并发送微博。如果第一条微博发送成功，在串口监视器上的输出信息如下：

```
Connectiong...
OK.Tweet sent.

Internal Temp:26 C 79 F .External Temp:26 C 79 F

Internal Temp:26 C 79 F .External Temp:26 C 79 F

Internal Temp:26 C 79 F .External Temp:26 C 79 F
```

程序第一次运行时，在进入主循环之前它将在 **setup** 函数中获得温度之后保持尝试连接到微博。尝试连接不会停止，直到成功连接。如果程序连接失败，返回一个 **failed: code 403** 或 **connection failed** 信息。如果发送微博成功，程序将不再发送直到间隔时间已过，这个时间的默认设置是 10 分钟。虽然你可以改变这个时间间隔，但是 **Twitter** 限制每小时最多发送 350 次，因此要保证不要超过这个数。你现在可以进入 **Twitter** 网站，从任何位置看账号内的信息，查看温度读数。

下面让我们看一下代码是如何工作的。

代码回顾

这个程序开始于包含相应的库：

```
#include <Ethernet.h>
#include <EthernetDHCP.h>
#include <EthernetDNS.h>
#include <Twitter.h>
#include <OneWire.h>
#include <DallasTemperature.h>
```

Twitter 库需要 3 个 **Ethernet** 库才能工作，因此你要全部包含这 3 个库，之后定义传感器：

```
#define ONE_WIRE_BUS 3
#define TEMPERATURE_PRECISION 12
```

给温度生成 4 个浮点数，这次内部和外部温度都用摄氏度和华氏度：


```
float itempC, itempF, etempC, etempF;
```

程序第一次试图生成一条微博，你希望程序保持尝试连接，直到成功连接并送出信息。因此生成一个布尔数，设置为 **true**，使你知道本次发送的是否为第一条微博：

```
boolean firstTweet = true;
```

如前所述，生成一个单线温度传感器实例和两个传感器的地址实例：

```
OneWire oneWire(ONE_WIRE_BUS);
DallasTemperature sensors(&oneWire);
```

```
DeviceAddress insideThermometer = { 0x10, 0x7A, 0x3B, 0xA9, 0x01, 0x08, 0x00,
0xBF };
DeviceAddress outsideThermometer = { 0x10, 0xCD, 0x39, 0xA9, 0x01, 0x08, 0x00,
0xBE };
```

设定网卡的 Mac 地址：

```
byte mac[] = { 0x64, 0xB9, 0xE8, 0xC3, 0xC7, 0xE2 };
```

之后，生成一个 **Twitter** 实例传递给它你的账户的口令作为参数：

```
Twitter twitter("608048201-CxYlyQi8ezhvjz60ZVfPHVdzIHbMODlh2gvoaAIx");
```

设置发送两条微博之间的时间间隔：

```
unsigned long interval = 600000; //10 分钟
```

生成一个变量存储最后一条微博发出的时间：

```
unsigned long lastTime; //最后一次发出微博的时间
```

生成两个字符串数组，它们将存储微博的信息和要在你的串口监视器上显示的字符。

```
char message[140], serialString[60]
```

现在生成一些函数，第一个函数是从两个传感器中获得温度，存储温度值到变量：

```
void getTemperatures()
```

```

{
    itempC = sensors.getTempC(insideThermometer);
    itempF = DallasTemperature::toFahrenheit(itempC);
    etempC = sensors.getTempC(outsideThermometer);
    etempF = DallasTemperature::toFahrenheit(etempC);
}

```

之后，这个函数发送一条微博，它需要一个参数，这个参数是一个字符数组，微博信息存储在其中。

```
void tweet(char msg[]) {
```

通知用户正尝试连接：

```
Serial.println("connecting ...");
```

之后，使用 `Twitter` 对象的 `post()` 方法去发送信息。如果发送成功，这个函数返回 `true`，如果连接失败，它返回 `false`。

```
if (twitter.post(msg)) {
```

如果连接成功，那么使用 `wait()` 方法检查发送的状态，这个方法返回 `Twitter` 网站的 HTTP 状态代码。

```
int status = twitter.wait();
```

如果状态代码是 200，这是一个 HTTP 代码，它意味着每件事情都做对了，换句话说，如果微博成功发送出去，那么将执行括号中的代码。

```
if (status == 200) {
```

如果成功，通知使用者：

```
Serial.println("OK. Tweet sent.");
Serial.println();
```

之后设置 `lastTime` 为 `millis()` 中的当前值。这使你可以判断自从最后一次发送微博多长时间已经过去了。

```
lastTime = millis
```

你第一次成功发送微博后希望程序退出 `setup` 函数中的 `while` 循环，然后继续运行主循环，因此设置 `firstTweet` 标志位为 `false`。

```
firstTweet = false;
```

如果状态不是 200，表明传送失败，那么通知使用者返回的代码作为调试信息用：

```
} else {
    Serial.print("failed : code ");
    Serial.println(status);
}
```

如果在第一部分就没有连接成功，通知用户如下：

```
} else {
    Serial.println("connection failed.");
}
```

定义用户函数后，到 `setup` 函数：

```
void setup()
```

首先你开始给 `EthernetDHCP` 类传递 Mac 地址：

```
EthernetDHCP.begin(mac);
```

DHCP（动态主机协议）是在网上自动注册 IP 地址的协议。协议允许网卡自动地注册从路由器获得的一个 IP 地址。在之前的项目中手工设置了 IP 地址，这次用 `EthernetDHCP` 类去自动注册一个 IP 地址。这虽然方便了用户，但是会使得代码变得非常长。

之后，用 9600 波特率开始串口通信，与前面的项目一样建立传感器：

```
Serial.begin(9600);
sensors.begin();
sensors.setResolution(insideThermometer, TEMPERATURE_PRECISION);
sensors.setResolution(outsideThermometer, TEMPERATURE_PRECISION);
```

获得温度，因为你要使用温度数值：

```
sensors.requestTemperatures()
getTemperatures();
```

现在尝试送出第一条微博，`while` 循环使这个动作保持运行直到第一条微博成功送出。

```
while (firstTweet) {
```

之后，使用 `sprintf` 函数把微博内容合成到 `Message[]` 数组中。给 `sprintf` 函数传递 4 个温度值和 `millis()` 值。因为 `millis` 是一个无符号长整型数，在 `sprintf` 中要使用 `%ld` 描述符来打印长整型数。

```
sprintf(message, "Int. Temp: %d C (%d F) Ext. Temp: %d C (%d F). Tweeted from
Arduino. %ld", int(itempC), int(itempF), int(etempC), int(etempF), millis());
tweet(message);
```

加 `millis()` 值到最后发送的微博内的原因是，Twitter 将不发布与上一次一样的信息。如果当前温度与上一次发送微博时相比没有发生变化，信息将是相同的，这样 Twitter 将返回错误代码。因为你想在时间间隔内有规律地更新每个数据，所以通过加 `millis()` 的值到信息的尾部，可以保证信息与上一次发送的不同。确保你的微博的长度总共不超过 140 字符，否则在你的微博上会出现不完整的信息。

现在你已经完成了微博信息的合成，进入 `tweet()` 函数：

```
tweet (message);
```

之后到主循环，只有在 `setup` 函数中第一次发送成功才能进入：

```
void loop()
```

首先，运行一个 `EthernetDHCP` 类中的 `maintain` 函数，保持自动注册 IP 地址激活和有效。

```
EthernetDHCP.maintain();
```

更新温度：

```
sensors.requestTemperatures();
```

之后，使用 `sprintf` 函数去编译串口监视器的输出。这个函数比整行打印 `serial.print()` 函数更方便，因此你可以很好地使用 `sprintf` 函数，尽管它增加了代码的长度。

```
sprintf(serialString, "Internal Temp: %d C %d F. External Temp: %d C %d F",
int(itempC), int(itempF), int(etempC), int(etempF));
```

之后，在一个短的延时后把字符串输出到串口监视器：

```
delay(500);
Serial.println(serialString);
Serial.println();
```

之后，确定自从最后一次发送微博是否间隔时间已经过去。如果是，发送另外一条微博。计算 `lastTime+interval` 的值看是否 `millis()` 中的当前值大于它（如间隔时间已经过去），如果是，编译新的信息再次发送微博信息。

```
if (millis() >= (lastTime + interval)) {
    //合成要发布微博信息的字符串
    sprintf(message, "Int. Temp: %d C (%d F) Ext. Temp: %d C (%d F).
Tweeted from Arduino. %ld", int(itempC), int(itempF), int(etempC), int(etempF),
millis()); tweet(message);
}
```

最后，在更新串口监视器前延时 10 秒，以保证不会用过多的信息轰炸用户。

```
delay(10000); //延时 10 秒
```

现在你知道了如何从 **Arduino** 中发送微博，可以使用这个技术做各种项目。微博可以告诉你一个盆景是否需要浇水了，或在一个房子周围加上传感器，当有人进入房间时发布信息，或当有人在门口时发送一个门铃声，或者告诉你你的猫已经离开或进入房间。可能性是无限的。

下面到达你的旅程的最后一个项目了。在最后一个项目中，你将使用网卡从 **Internet** 上读一些数据而不是送出一些数据。

项目 50——RSS 读取气象信息

最后一个项目将再次使用 Ethernet 板，不是传送一些数据到网络服务器，而是使用 Arduino 和 Ethernet 板从互联网上获得数据，之后显示这些数据到串口显示器。你将使用的的数据是采用 RSS（简易信息聚合）从 www.weather.gov 由你选择的美国一个地区的天气数据。这个代码非常适合读 RSS 天气数据，并且如果你在美国之外，也可以从任何其他地方读天气数据。

RSS 是一个网站形式，发布有规律更新的信息，例如，天气信息、新闻等。数据以 XML（可扩展标记语言）格式发布，这是一套以机器可读的形式编码文档的规则。XML 是一个简单的格式，没有必要去理解它是如何工作的。Arduino 只是查看在 XML 中的标签，那里存储着温度、湿度和气压数据，Arduino 分离出这些信息并显示。

你将使用 XML 返回加利福尼亚的爱德华地区的气压。如果你想使用其他地区的数据，到 http://www.weather.gov/xml/current_obs/ 上选择地区，之后找到 XML 数据返回位置的完整地址，然后根据这个地址调整代码即可。

对于硬件，这次你不使用什么新的东西，只是把 Ethernet 板插入 Arduino 中。

输入代码

插入 Ethernet 板到 Arduino 中（如果还没有插上），之后从清单 17-5 中输入代码，感谢 Bobs（Xtalker）在 Arduino 论坛上给出的代码。

清单 17-5 项目 50 的代码

```
//项目 50
//感谢 Bob S. 的原始代码
//从 weather.gov 网站以 XML 格式获得爱德华地区当前天气信息

#include <Ethernet.h>
#include <SPI.h>

//定义数据字符串的最大长度
#define MAX_STRING_LEN 20
```



```
//各种变量
char tagStr[MAX_STRING_LEN] = "";
char dataStr[MAX_STRING_LEN] = "";
char tmpStr[MAX_STRING_LEN] = "";
char endTag[3] = {'<', '/', '\0'};
int len;
//存储在 XML 返回信息中包含的结尾标签的数组
boolean tagFlag = false;
boolean dataFlag = false;

//网络变量
byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
byte ip[] = {172,31,24,232};
byte server[] = { 140, 90, 113, 200 }; // www.weather.gov

//建立网络客户机
Client client(server, 80);

void setup()
{
  Serial.begin(9600);
  Serial.println("Starting Weather RSS Reader");
  Serial.println("connecting...");
  Ethernet.begin(mac, ip);
  delay(1000);

  if (client.connect()) {
    Serial.println("connected");
    client.println("GET /xml/current_obs/KEDW.xml HTTP/1.0");
    client.println();
    delay(2000);
  } else {
    Serial.println("connection failed");
  }
}

void loop() {

  //从 Web 页面读数据
  while (client.available()) {
    serialEvent();
  }
}
```

```

if (!client.connected()) {

    client.stop();

    if (int t=0; t<15; t++) { //数据源每 15 分钟更新一次
        delay(60000); // 1 分钟
    }
    if (client.connect()) {
        client.println("GET /xml/current_obs/KEDW.xml HTTP/1.0");
        client.println();
        delay(2000);
    } else {
        Serial.println("Reconnection failed");
    }
}

//处理 Web 页面的每一个字符
void serialEvent() {

    //读一个字符
    char inChar = client.read();

    if (inChar == '<') {
        addChar(inChar, tmpStr);
        tagFlag = true;
        dataFlag = false;
    } else if (inChar == '>') {
        addChar(inChar, tmpStr);

        if (tagFlag) {
            strncpy(tagStr, tmpStr, strlen(tmpStr)+1);
        }

        //清除临时变量
        clearStr(tmpStr);

        tagFlag = false;
        dataFlag = true;
    }
}

```

```

} else if (inChar != 10) {
    if (tagFlag) {
        //在字符串内增加标志
        addChar(inChar, tmpStr);

        //检查</XML> 结尾标签, 并忽略它
        if ( tagFlag && strcmp(tmpStr, endTag) == 0 ) {
            clearStr(tmpStr);
            tagFlag = false;
            dataFlag = false;
        }
    }

    if (dataFlag) {
        //增加数据到字符串
        addChar(inChar, dataStr);
    }
}

//如果是换行符号, 换行
if (inChar == 10 ) {

    //找到指定标签, 打印数据
    if (matchTag("<temp_f>")) {
        Serial.print("Temp: ");
        Serial.print(dataStr);
    }
    if (matchTag("<temp_c>")) {
        Serial.print(", TempC: ");
        Serial.print(dataStr);
    }
    if (matchTag("<relative_humidity>")) {
        Serial.print(", Humidity: ");
        Serial.print(dataStr);
    }
    if (matchTag("<pressure_in>")) {
        Serial.print(", Pressure: ");
        Serial.print(dataStr);
        Serial.println("");
    }

    //清除所有字符串

```

```

    clearStr(tmpStr);
    clearStr(tagStr);
    clearStr(dataStr);

    //清除标志
    tagFlag = false;
    dataFlag = false;
}
}

//清除字符串的函数
void clearStr (char* str) {
    int len = strlen(str);
    for (int c = 0; c < len; c++) {
        str[c] = 0;
    }
}

//在字符串中增加字符和检查长度的函数
void addChar (char ch, char* str) {
    char *tagMsg = "<TRUNCATED_TAG>";
    char *dataMsg = "-TRUNCATED_DATA-";

    //检查字符串的最大长度，确保它不会超长
    //如果字符串长度超过 MAX_STRING_LEN 指定的长度，用提醒信息替换它
    if (strlen(str) > MAX_STRING_LEN - 2) {
        if (tagFlag) {
            clearStr(tagStr);
            strcpy(tagStr, tagMsg);
        }
        if (dataFlag) {
            clearStr(dataStr);
            strcpy(dataStr, dataMsg);
        }

        //清理临时字符串和标签，停止当前过程
        clearStr(tmpStr);
        tagFlag = false;
        dataFlag = false;
    } else {
        //增加字符到字符串

```

```

        str[strlen(str)] = ch;
    }
}

//检查当前标签中是否包含特定字符的函数
boolean matchTag (char* searchTag) {
    if ( strcmp(tagStr, searchTag) == 0 ) {
        return true;
    } else {
        return false;
    }
}

```

上传代码，打开串口监视器。如果每件事情都做对了，你将得到与以下内容类似的输出：

```

Starting Weather RSS Reader
connecting...
connected
TempF:60.0,TempC:15.4,Humidity:100,Pressure:29.96

```

它表示每 60 秒用最新的数据更新一次。让我们看看代码是如何工作的。

代码回顾

这个程序开始于包含要使用的 Ethernet 库及相关的库：

```

#include <Ethernet.h>
#include <SPI.h>

```

之后定义数据字符串的最大长度：

```

#define MAX_STRING_LEN 20

```

如果你需要更多的返回信息，你可能需要增加这个长度。之后，生成三个数组，它们将存储要处理的（所有这些都由 Length 定义）字符串变量。

```

char tagStr[MAX_STRING_LEN] = "";
char dataStr[MAX_STRING_LEN] = "";
char tmpStr[MAX_STRING_LEN] = "";

```

之后，生成另外一个数组存储在 XML 返回信息中包含的结尾标签：

```
char endTag[3] = {'<', '/', '\0'};
```

之后，声明变量存储在相关的代码段中要处理的字符长度：

```
int len;
```

之后，声明两个标志。这些将用来区分需要分析的 XML 代码标签和信息。

```
boolean tagFlag = false;
boolean dataFlag = false;
```

之后，设置网卡的 Mac 和 IP 地址：

```
byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
byte ip[] = {172, 31, 24, 232};
```

这次 IP 地址是 www.weather.gov 网站的 IP 地址：

```
byte server[] = { 140, 90, 113, 200 }; //www.weather.gov 网站的 IP 地址
```

如果你使用其他网站获得天气信息，更改以上 IP 地址为你使用的 URL。之后，生成客户机对象，传递给它使用的服务器地址和端口：

```
Client client(server, 80);
```

之后，到 `setup` 函数：

```
void setup()
```

它开始于用 9600 波特率进行串口通信，准备打印数据到串口监视器：

```
Serial.begin(9600);
```

程序告知使用者这个例程的名称和正在尝试连接：

```
Serial.println("Starting Weather RSS Reader");
Serial.println("connecting...");
```


Ethernet 连接开始，传递元件的 Mac 和 IP 地址，随后是一个短延时让 Arduino 尝试连接：

```
Ethernet.begin(mac, ip);  
delay(1000)
```

之后，检查是否已经连接到客户机（www.weather.gov.websit）：

```
if (client.connect()) {
```

如果连接成功则通知使用者：

```
Serial.println("connected");
```

之后，执行 HTML GET 命令去连接 XML，从存储相关的数据源子目录中获得数据，后面跟随一个延时保证连接成功。

```
client.println("GET/xml/current_obs/KEDW.xmlHTTP/1.0");  
client.println();  
delay(2000);
```

如果连接没有成功，通知使用者连接失败：

```
    } else {  
        Serial.println("connection failed");  
    }  
}
```

之后是主循环：

```
void loop() {
```

因为在 setup 循环中执行了一个 GET 命令，所以串口缓冲区应该含有 XML 服务器的反馈信息，也就是那里有数据可用。

```
while (client.available()) {
```

调用 serialEvent()函数：

```
serialEvent();
```

要对这个函数进行简要解释。如果已进行连接：

```
if (!client.connected()) {  
  
    //到客户机的连接停止  
  
    client.stop();
```

那么在尝试另外一个连接前等待 15 分钟。在许多时候数据源每 15 分钟才更新一次，因此更新信息时间短于这个时间没有意义：

```
if (int t=0; t<15; t++) { //数据源每 15 分钟更新一次  
    delay(60000); //延时 1 分钟  
}
```

如果已经成功连接到客户机：

```
if (client.connect())
```

那么执行另外一个 GET 命令去获得最后的 XML 数据源：

```
client.println("GET/xml/current_obs/KEDW.xmlHTTP/1.0");  
client.println();  
delay(2000);
```

如果连接失败，通知使用者：

```
} else {  
    Serial.println("Reconnection failed");  
}
```

之后到 serialEvent() 函数，这个函数的目的是从 XML 中读数据，然后根据读数据的结果执行一些动作。

```
void serialEvent() {
```

这个函数开始于读第一个字符并存储到 inChar 中：

```
char inChar = client.read();
```

现在你需要分析这个字符确定它是一个标签还是数据。如果是标签，那么设置标志 `tagFlag` 为 `true`。如果是数据，设置标志 `dataFlag` 为 `true`。每次确定一个标签为 `true` 后另外一个标签设置为 `false`。

返回的每行数据看起来像：

```
<current_observation version="1.0"
  Xmlns:xsd=http://www.w3.org/2001/XMLSchema
  Xmls:xsi=http://www.w3.org/2001/XMLSchema-instance
  Xsi:noNamespaceSchemaLocation=http://www.weather.gov/view/current
_observation.xsd>
  <credit>NOAA's National Weather.gov/</credit_URL.
  <credit_URL>http://weather.gov/</credit_URL>
  <image><url>http://weather.gov/images/xml_logo.gif</url><title>NOAA
's National
Weather Service</title><link>http://weather.gov</link></image>
  <suggested_pickup>15 minutes after the hour</suggested_pickup>
  <suggested_pickup_period>60</suggested_pickup_period>
  <location>Edwards AFB,CA</location>
  <station_id>KEDW</station_id>
  <latitude>34.91</latitude>
  <longitude>=117.87</longitude>
  <observation_time>Last Updated on Oct 19 2010 ,9:55 am
PDT</observation_time>
  <observation_time_rfc822>Tue,19 Oct 2010 09:55:00
=0700</observation_time_rfc822>
  <weather>Mostly Cloudy</weather>
  <temperature_string>62.0 F (16.4 C)</temperature_string>
  <temp_f>62.0</temp_f>
  <temp_c>16.4</temp_c>
  <relative_humidity>100</relative_humidity>
```

正如你看到的，每一段信息都包含在一个标签内，例如，华氏温度用 `<temp_f>` 标签作为开始，用 `</temp_f>` 结束，在标签中的内容是数据。

首先，检查字符是否是一个“<”符号，如果是，它是一个标签的开始。

```
if (inChar == '<') {
```

如果是这样，调用 `addChar` 函数，它将检查字符串长度是否在 `MAXSTRING_LIN` 限制内，如果是，加字符到 `tmpStr` 字符串中，在后面的函数中检查这个字符串。

```
addChar(inChar, tmpStr);
```

因为你已经发现一个标签，所以 `tagFlag` 设置为 `true`，`dataFlag` 设置为 `false`：

```
tagFlag = true;
dataFlag = false;
```

通过查找 “>” 符号确认是否到标签的尾部：

```
} else if (inChar == '>') {
```

那么字符增加到 `stmpStr` 字符串中：

```
addChar(inChar, tmpStr);
```

如果你正在处理一个标签并且到达标签的尾部，可以从 `tmpStr`（临时字符串）中复制完整标签到标签字符串（`tgsStr`），使用 `strncpy` 函数进行字符串复制。

```
if (tagFlag) {
    strncpy(tagStr, tmpStr, strlen(tmpStr)+1);
}
```

`strncpy` 函数复制字符串的一部分到另一个字符串中，它需要 3 个参数：要复制的字符串、从哪个字符串复制和复制字符的数量。例如：

```
strncpy(firstString, secondString, 10)
```

它表示 `secondString` 字符串中头 10 个字符将被复制到 `firstString` 字符串中。在你的例子中复制全部内容，所以用 `temporary` 字符串长度+1 作为复制标签字符串中字符的数量。

一旦临时字符串已被复制，需要清除它准备接收下一部分的数据，调用 `clearstr` 函数实现清空字符串，传递给这个函数想要清除的字符串做参数。

```
clearStr(tmpStr);
```

这两个标志设为 `false`，为接收下一部分的信息做好准备。

```
tagFlag = false;
dataFlag = true;
```

如果读到的字符是换行符（ASCII10）：

```
} else if (inChar != 10) {
```

那么如果你正在处理一个标签，加字符到字符串中：

```
if (tagFlag) {
    addChar(inChar, tmpStr);
```

要忽略标签的结尾，因此检查是否正在处理一个标签并且已到达标签的尾部（通过对比标签结束符）：

```
if ( tagFlag && strcmp(tmpStr, endTag) == 0 ) {
```

之后，标签被忽略，清除字符串，设置标签标志为它们的默认值：

```
clearStr(tmpStr);
tagFlag = false;
dataFlag = false;
```

`strcmp` 函数的作用是对比两个字符串，在这个例子里，它对比临时字符串（`tmpstr`）和 `endTag` 数组中的字符串：

```
strcmp(tmpStr, endTag)
```

如果字符串相符合，函数返回值将是 0，如果不符合，函数返回其他值。通过与 `endTag` 数组进行对比，检查当前是否为三个结束符中的任意一个。

如果当前字符是数字：

```
if (dataFlag) {
```

增加当前字符到数据字符串（`dataStr`）：

```
addChar(inChar, dataStr);
```

上面的代码只是判断你是否正在处理标签，如果是，存储字符到标签字符串（tagStr），如果是数据，存储它到数据字符串（dataStr），直到到达这个标签尾部，然后把数据单独存储。

如果已经到了行尾字符，清除当前字符串的结尾字符。现在需要检查标签看它们是否是你希望的温度、湿度或压力数据：

```
if (inChar == 10 ) {
```

使用 `matchTag` 函数（一会儿就会遇到这个函数）分析数据，函数检查在标签字符串中是否有特定的标签，如果有返回 `true` 值。开始在华氏标签中查找温度：

```
<temp_f>
if (matchTag("<temp_f>")) {
```

如果是，打印出字符串。如果标签是 `<temp_f>`，数据就是华氏的温度数据。

```
Serial.print("Temp: ");
Serial.print(dataStr);
```

之后，查找摄氏的温度：

```
if (matchTag("<temp_c>")) {
    Serial.print(", TempC: ");
    Serial.print(dataStr);
}
```

湿度：

```
if (matchTag("<relative_humidity>")) {
    Serial.print(", Humidity: ");
    Serial.print(dataStr);
}
```

压力：

```
if (matchTag("<pressure_in>")) {
    Serial.print(", Pressure: ");
    Serial.print(dataStr);
```



```
        Serial.println("");  
    }
```

之后清除全部字符串，为处理下一行做准备：

```
clearStr(tmpStr);  
clearStr(tagStr);  
clearStr(dataStr);
```

清除标志：

```
tagFlag = false;  
dataFlag = false
```

之后是你自己的函数，开始于字符串清除函数（clearStr）：

```
void clearStr (char* str) {
```

它只是使用 `strlen()` 函数找出传递给函数的字符串长度：

```
int len = strlen(str);
```

之后使用 `for` 循环来用 ASCII0（NULL）字符填充数组的每一个元素：

```
for (int c = 0; c < len; c++) {  
    str[c] = 0;  
}
```

下一个函数是 `addChar` 函数，传递读到的当前字符和当前字符串作为它的参数：

```
void addChar (char ch, char* str) {
```

定义两个新的字符数组，在它们中存储错误信息：

```
char *tagMsg = "<TRUNCATED_TAG>";  
char *dataMsg = "-TRUNCATED_DATA-";
```

如果你发现字符串长度超过了 `MAX_STRING_LEN`，那么用出错信息替换它们。

现在检查字符串长度看它是否到达最大长度：

```
if (strlen(str) > MAX_STRING_LEN - 2) {
```

如果你正在处理的是标签:

```
if (tagFlag) {
```

那么标签字符串清除, 复制错误信息到标签字符串中:

```
clearStr(tagStr);
strcpy(tagStr, tagMsg);
```

如果处理的是数据, 那么数据字符串清除, 复制数据出错信息到数据字符串中:

```
if (dataFlag) {
    clearStr(dataStr);
    strcpy(dataStr, dataMsg);
}
```

清除临时字符串和标签:

```
clearStr(tmpStr);
tagFlag = false;
dataFlag = false;
```

如果字符串长度没有超过最大长度, 增加已经读进的当前字符到字符串中。使用字符串的长度去找出最后的字符, 你就可以把字符增加到字符串中的下一个位置内。

```
} else {
    //增加字符到字符串
    str[strlen(str)] = ch;
}
```

最后到 `matchTag` 函数, 它用来检查是否发现作为参数传递给它的标签, 相应地返回 `true` 或 `false`:

```
boolean matchTag (char* searchTag) {
```

这个函数是 `Boolean` 型的, 因为它返回布尔值, 需要一个字符数组作为参数。

```
if ( strcmp(tagStr, searchTag) == 0 ) {
```

```

        return true;
    } else {
        return false;
    }
}

```

通过改变 XML 的数据源的 URL 和数据源中的标签,可以使用这个代码在任何 RSS 数据源中查找到一段数据。例如,你可以使用在 <http://weather.yahoo.com> 上的 Yahoo 天气数据源,之后导航到你要查找的地区,并单击 RSS 按钮。把数据源的 URL 输入到代码中,通过右击和选择正确的菜单项你可以看到数据源的原始来源,可以查看标签并更改代码去找到相应的信息段。

最后一个项目演示如何使用网卡板从 Internet 上获得信息。之前学习的是从网卡板发送数据到外部网络,这个项目是从 Internet 读回数据。不仅仅是在串口监视器中显示天气数据,还可以使用在前面的项目中已经学到的技巧去把数据显示到 LCD 屏或 LED 点阵显示器上。

小结

最后一章介绍了如何将你的 Arduino 连到 Internet,目的有为了送出格式化的数据到服务器网页、发送微博到 Twitter、发电子邮件或传感器数据到邮件服务器,或从网页上获得格式化数据为你自己使用。有了连接你的 Arduino 到 LAN 或 Internet 的知识,打开了一个潜在项目的新空间。数据可以从你的房子的任何地方或办公室里获得,只要那里有可用 Internet 接口,或数据可以从 Internet 上读出给 Arduino 处理并进行一些动作等。

例如,你可以使用当前的天气数据去判断是否要下雨,或提醒你要带两件衣服、关上天窗等。用可以联网的 Arduino 做什么东西,受限制的只是你的想象力。

本章的主题和概念

- 如何手工指定 MAC 和 IP 地址给你的元件
- 客户机和服务器的概念
- 如何用 `client.connect()` 函数监听客户连接
- 通过打印信息到客户机的方法送出 HTML 代码
- 使用你的 Arduino 作为网络服务器

- 从网络浏览器中连接到 Arduino
- 用 `client.available()` 函数检查数据是否可用
- 用 `client.read()` 函数读数据
- 把数据送到 Pachube, 以图表形式查看数据
- 通过代理服务器从 Arduino 中发布微博到 Twitter
- 用 Arduino 发送电子邮件
- 从 RSS 数据源获得数据并显示
- 用 `strcpy` 和 `strncpy` 函数复制字符串
- 用 `strcmp` 函数对比字符串
- 用 `memset` 函数填充内存区
- 用 `sizeof` 函数获得数组大小
- 用 `sprintf` 函数处理字符串
- 用 `strlen` 函数获得字符串的长度
- 用 `strstr` 函数查找子字符串
- 用 `ping` 命令找到 IP 地址
- 用 `client.Connect()` 函数检查是否连接到客户机
- 用 `Ethernet.begin (mac.IP)` 函数生成 Ethernet 连接
- 用网站资源加密用户名和口令为 Base-64 加密版本
- 使用 `EthernetDHCP.h` 库自动注册 IP 地址
- 使用 `Twitter` 库发送和等待命令
- 查看 XML RSS 返回信息中的一个标签