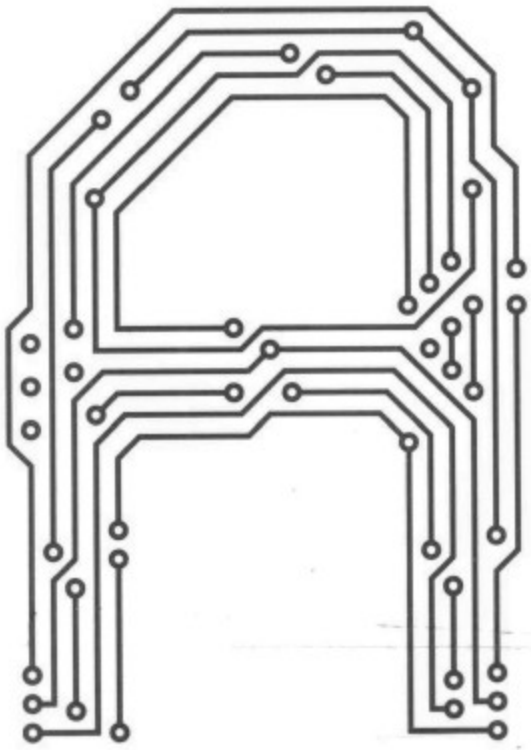
HZ BOOKS
华章科技

国内首本Arduino著作，全面而系统地讲解Arduino平台的功能特性和开发方法。
注重实战，包含大量实战案例，不懂硬件的工程师也能迅速搭建产品原型。
资深Arduino玩家、硬件工程师主笔，创客联盟与学术专家联袂推荐，权威性毋庸置疑。

单片机与嵌入式



AVR篇

Arduino开发实战指南

程晨 著

机械工业出版社
China Machine Press

这是一本全面讲述Arduino应用开发的书，它从Arduino的I/O板和最基本的C语言入门开始，详细介绍了Arduino库和10多个第三方开源库及其对应的外围设备的使用与编程。这本书第三部分通过详细介绍两个实例展示了Arduino作品开发的过程，也体现了Arduino的强大潜力。本书适合Arduino的初学者作为循序渐进的教材，也适合希望深入学习Arduino的开发者作为参考手册。

——浙江大学计算机学院Arduino创新教学实践者 翁恺老师

Arduino是一个开源硬件平台，电子专业的学生完全可以通过查资料、买元件、做PCB、焊电路，制作自己的Arduino硬件模块；同时，很多厂商也开发了各种各样的Arduino外围功能电路供学生选择，无论是电机驱动、无线通信、音乐播放，还是各种传感器（压力、速度、倾角、方向等），这些均为学生在学习和设计与自动控制、物联网、无线传感网相关的知识提供了不同的学习途径，并且使得学习电子知识变得相对容易。另外，Arduino的代码语法简单易懂，对于学过C语言程序设计甚至没有任何编程经验的读者来说，Arduino程序也是简单易读的。因此，这本书非常适合作为Arduino爱好者的参考教材。同时，通过全面系统介绍Arduino及其开发方法，本书也为电子和计算机类专业低年级学生打开了一扇兴趣之门，书中丰富的实例更是增强学生动手能力不可多得的素材。

——西安邮电大学计算机学院周立功“3+1”创新教育实验班班主任 马博老师

这是一本关于Arduino及其开发方法的书。本书内容涵盖广泛，但又不失重点。通过系统的理论知识介绍和精彩的实例讲解，将Arduino这个开源硬件平台阐释得淋漓尽致。这本书的内容包括Arduino的来龙去脉、C语言基础、Arduino开发平台的使用以及实战项目。与之前看过的几本直接翻译的国外Arduino书籍相比，这本书更加生动。读者即使原来没有在嵌入式平台上编写过软件，通过这本书也可以学会Arduino的开发方法，实现自己的产品创意，这也是Arduino的魔力所在。本书作者有非常丰富的Arduino项目开发经验，后面几个章节的项目记录了作者使用Arduino的开发心得。通过阅读此书你也可以DIY出充满创意的产品原型！

——DFRobot创始人、Arduino首批引入者之一 庄明波

上架指导：电子与电气工程
ISBN 978-7-111-37005-5
9 787111 370055
定价：59.00元

客服热线：(010) 88378991, 88361066
购书热线：(010) 68326294, 88379649, 68995259
投稿热线：(010) 88379604
读者信箱：hzjsj@hzbook.com

华章网站 http://www.hzbook.com
网上购书：www.china-pub.com

单片机与嵌入式

程晨 著

AVR篇

Arduino 开发实战指南

机械工业出版社
China Machine Press

Arduino是一个注重实际动手操作的产品，所以本书以实际应用为纽带将各个章节联系起来。本书首先介绍Arduino的一些基础知识，接着针对具体应用介绍了一些扩展板以及Arduino扩展库，最后应用之前的内容完成了具有视频监控功能的履带车、遥控机械臂以及双足机器人的制作。

本书内容循序渐进，图文并茂，可以带领读者走入Arduino的精彩世界。本书适合电子专业、交互设计专业、新媒体技术专业学生阅读，也可以作为所有电子爱好者开展Arduino制作项目的参考手册。

图书在版编目（CIP）数据

Arduino开发实战指南：AVR篇 / 程晨著. —北京：机械工业出版社，2012.2

ISBN 978-7-111-37005-5

Ⅰ. A…　Ⅱ. 程…　Ⅲ. 单片微型计算机－指南　Ⅳ. TP368.1-62

中国版本图书馆CIP数据核字（2011）第281978号

机械工业出版社（北京市西城区百万庄大街22号　邮政编码　100037）

责任编辑：秦　健

北京京师印务有限公司印刷

2012年3月第1版第1次印刷

186mm × 240mm · 21印张

标准书号：ISBN 978-7-111-37005-5

定价：59.00元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换

客服热线：（010）88378991；88361066

购书热线：（010）68326294；88379649；68995259

投稿热线：（010）88379604

读者信箱：hzjsj@hzbook.com

前　言

在2011年举行的Google I/O开发者大会上，Google发布了基于Arduino的Android Open Accessory标准和ADK工具，这使得大家对Arduino的前景十分看好。Phillip Torrone大胆地预测Google将用Android+Arduino的形式掀起自己的“Kinect模式”浪潮。目前，国内关注Arduino的人越来越多，但介绍Arduino的书籍却很少。笔者由于工作的关系，接触Arduino较早，所以希望通过自己的努力让更多的人了解Arduino，在近一年的时间里，通过不断学习、查阅Arduino相关知识，终于完成了书稿的撰写工作。但在书稿完成之后，心中却一直忐忑不安，Arduino是一个介于软件与硬件之间的产品，系统性不是很强，加上笔者水平有限，拙著中一定存在不少的缺点与漏洞，为此，笔者先为书中的不足之处致以真诚的歉意，同时诚挚地欢迎广大读者提出宝贵的意见并不吝赐教。

本书的内容及面向的读者

Arduino是一个注重实际动手操作的产品，所以本书以实际应用为纽带将各个章节联系起来。本书共9章，首先介绍Arduino的一些基础知识，接着针对具体应用介绍了一些扩展板以及Arduino扩展库，最后应用之前的内容完成了具有视频监控功能的履带车、遥控机械臂以及双足机器人的制作。

因为Arduino本身具有简单易用的特点，所以本书面向的读者是所有有兴趣使用Arduino进行项目开发的人。

当然，根据读者的情况不同，本书的阅读方式也不同。

如果读者是一个之前没有进行过单片机开发也没有进行过软件开发的人，现在想使用Arduino来实现自己的一些想法，那么首先要阅读本书的前两章，了解一些简单的编程思想以及程序结构。接下来阅读第3章的目录，了解Arduino都有什么基本函数，具体内容可以先不用看，当你之后使用这些函数遇到问题时再回过头来看一看相应的函数说明。然后将Arduino接到你的电脑上，翻开第4章，根据书中的内容，边学习边实践，4.5节可以跳过不

看。第5～7章介绍了Arduino周边的资源，以便帮助你尽快地实现想法，这3章的内容也可以采用跳跃式的阅读方式。第8、9章会告诉你前3章的内容是如何结合起来的。建议按照书中的内容至少动手完成一个项目的制作。

如果读者之前进行过AVR单片机的开发，想了解Arduino一些底层的知识，那么第2章的知识就可以跳过了，在简单地翻阅第3章的内容后，直接进入第4章，把Arduino连到电脑上实践一下，再回过头阅读第3章中关于Arduino的基本函数，结合自身已有的AVR单片机的知识了解Arduino底层的工作机制。需要说明的是，这里需要读者自己花一些精力，可能还需要学习一些C++方面的知识。第5章对Arduino硬件原理进行了详细介绍，若读者之前学习过，这一章可以选择性学习。第6章介绍的是Arduino的扩展库，如果读者也想开发一些Arduino扩展板，并以库的形式提供扩展板的软件资源，那么建议先学习最后一节，再从6.1节开始学习，深入地了解这些扩展库是如何与Arduino结合在一起的。至于剩下几章的内容，如果用开发单片机的思路来完成也是不难的，所以阅读的重点是看看如何用Arduino的思路进行项目的制作。

如果读者之前是做纯电脑软件开发工作的，即使用C++非常熟练，那么在阅读完第1章后，可以直接跳到第4章，感受一下Arduino给纯软件开发人员带来的那种完成硬件制作的感觉，然后仔细阅读第5章，看看目前都有哪些扩展板可以为自己所用，控制电机、控制液晶之类的，硬件知识哪怕我们不用，也还是要了解一些的。接下来，对于第6章，可以仔细阅读一下与硬件关系不太大的扩展库以及如何创建自己的库，在今后底层硬件库不断丰富完善的情况下，开发一些注重应用、与底层关系不是太紧密的库时，这就是我们的用武之地。第7～9章的内容会告诉我们前面的知识是如何结合起来的——用纯软件的思路。同样建议按照书中的内容至少动手完成一个项目的制作，做纯软件开发工作的人开发硬件也是很容易的。

致谢

首先要感谢本书的策划张国强先生，是他对Arduino的关注促成了本书的出版，同时在笔者撰写书稿时他也对本书提出了宝贵的写作建议，并对书稿进行了仔细审阅。

其次要感谢让我了解Arduino的庄明波先生，他不但在技术上给予了我很多的指导，同时也无私地提供了大量的Arduino扩展板的资料以及实物，供我在Arduino的程序调试中使用，同时与我共同探讨技术上遇到的问题。

最后要感谢现在正捧着这本书的您，感谢您肯花费时间和精力阅读本书，由于时间有限，书中难免存在疏漏与错误，诚恳地希望您批评指正，您的意见和建议将是我巨大的财富。希望在Arduino的领域结识更多的朋友。

目　录

前言

第一篇　基础篇

第1章　初识Arduino ……2

1.1　Arduino的历史 ……2
1.2　Arduino的家族 ……3
1.3　Arduino的资源 ……6
1.4　Arduino的开发环境 ……9
1.5　添加新硬件及设置开发环境 ……9
1.6　Arduino开发环境的应用 ……14

第2章　编写Arduino程序 ……16

2.1　绘制流程图 ……16
2.1.1　流程图基本符号 ……16
2.1.2　流程图的三种基本结构 ……17
2.2　C语言的标识符与关键字 ……18
2.2.1　标识符 ……18
2.2.2　关键字 ……18
2.2.3　运算符 ……19
2.2.4　分隔符 ……21
2.2.5　常量 ……21
2.2.6　注释符 ……21
2.3　控制语句 ……21
2.3.1　if语句 ……21
2.3.2　switch语句 ……22
2.3.3　while语句 ……23
2.3.4　do-while语句 ……24
2.3.5　for语句 ……25
2.3.6　break语句 ……26
2.3.7　continue语句 ……26
2.3.8　goto语句 ……26
2.4　程序结构 ……27

第3章　Arduino的基本函数 ……29

3.1　数字I/O ……30
3.1.1　pinMode(pin,mode) ……30
3.1.2　digitalWrite(pin,value) ……31
3.1.3　digitalRead(pin) ……32
3.2　模拟I/O ……33
3.2.1　analogReference(type) ……33
3.2.2　analogRead(pin) ……33
3.2.3　analogWrite(pin, value) ……34
3.3　高级I/O ……37
3.3.1　shiftOut(dataPin,clockPin, bitOrder,val) ……37
3.3.2　pulseIn(pin,state,timeout) ……38
3.4　时间函数 ……39
3.4.1　millis( ) ……39
3.4.2　delay(ms) ……40
3.4.3　delayMicroseconds(us) ……40
3.5　数学库 ……41
3.5.1　min(x,y) ……41
3.5.2　max(x,y) ……41

3.5.3 abs(x) ……41
3.5.4 constrain(amt,low,high) ……41
3.5.5 map(x,in_min,in_max,out_min, out_max) ……41
3.5.6 三角函数 ……42
3.6 随机数 ……42
3.6.1 randomSeed(seed) ……42
3.6.2 random(howsmall,howbig) …42
3.7 位操作 ……43
3.8 中断函数 ……43
3.8.1 interrupts( )和noInterrupts( )…43
3.8.2 attachInterrupt(interrupt, function,mode) ……43
3.9 串口通信 ……45
3.10 SPI接口 ……48
3.10.1 SPI接口概述 ……48
3.10.2 SPI接口数据传输 ……48
3.10.3 SPI类及其成员函数 ……49

第4章 Arduino硬件平台 ……52
4.1 Arduino的原理图 ……52
4.2 串行通信口的使用 ……55
4.2.1 实例功能 ……56
4.2.2 硬件电路 ……56
4.2.3 程序设计 ……56
4.3 数字I/O口的使用 ……58
4.3.1 实例功能 ……59
4.3.2 硬件电路 ……59
4.3.3 程序设计 ……59
4.4 模拟I/O口的使用 ……61
4.4.1 实例功能 ……61
4.4.2 硬件电路 ……61
4.4.3 程序设计 ……62
4.5 烧写引导程序 ……62
4.5.1 下载器AVRISP ……63
4.5.2 AVR Studio ……64
4.5.3 烧写引导程序 ……65

第二篇 模块篇

第5章 Arduino基本扩展模块 ……68
5.1 L293 Motor Shield ……68
5.1.1 直流电机的工作原理 ……68
5.1.2 H桥驱动电路 ……70
5.1.3 线性放大调速原理 ……71
5.1.4 PWM调速原理 ……72
5.1.5 L293 Motor Shield的原理 …72
5.1.6 L293 Motor Shield的应用 …74
5.1.7 程序设计 ……75
5.1.8 程序分析 ……76
5.1.9 程序的精练 ……77
5.2 Input Shield ……78
5.2.1 Input Shield原理图 ……79
5.2.2 Input Shield的实例 ……79
5.2.3 程序设计 ……80
5.2.4 程序分析 ……81
5.2.5 使用摇杆控制直流电机转速…81
5.3 LCD Keypad Shield ……83
5.3.1 液晶显示原理 ……83
5.3.2 标准1602液晶模块 ……83
5.3.3 1602液晶模块控制方式 ……84
5.3.4 LCD Keypad Shield原理图 …87
5.3.5 LCD Keypad Shield应用实例…89
5.3.6 程序设计 ……89
5.3.7 程序分析 ……92
5.3.8 Arduino的液晶控制方式 ……93
5.3.9 “hello Arduino!” ……94
5.4 Ethernet Shield ……97

5.4.1 Ethernet Shield原理图 ……97
5.4.2 W5100芯片介绍 ……97
5.4.3 W5100芯片的寄存器 ……101
5.4.4 W5100芯片的使用 ……105
5.4.5 Ethernet Shield应用实例 …105
5.4.6 程序设计 ……106
5.5 I/O扩展板 ……109
5.5.1 Xbee传感器扩展板V5 ……109
5.5.2 伺服电机控制 ……110
5.5.3 伺服电机应用实例 ……111
5.5.4 Interface shield ……114
5.5.5 RGB LED Module ……114
5.5.6 RGB LED Module应用实例…118
5.5.7 程序的精练 ……123

第6章 Arduino的扩展库 ……126

6.1 Arduino扩展库介绍 ……126
6.1.1 Arduino扩展库的作用 ……126
6.1.2 Arduino扩展库的应用 ……126
6.2 对象和类 ……130
6.2.1 类的定义 ……130
6.2.2 对象的创建及成员函数的调用 ……131
6.2.3 对象的初始化和构造函数 …132
6.2.4 函数的重载 ……133
6.2.5 析构函数 ……133
6.3 LiquidCrystal库 ……134
6.3.1 构造函数 ……136
6.3.2 command( )和write( ) ……139
6.3.3 begin( ) ……140
6.3.4 clear( ) ……142
6.3.5 home( ) ……142
6.3.6 setCursor( ) ……142
6.3.7 noDisplay( )和display( ) …143
6.3.8 cursor( )和noCursor( ) ……143
6.3.9 blink( )和noBlink( ) ……143
6.3.10 autoscroll( )和noAutoscroll( ) ……144
6.3.11 scrollDisplayLeft( )和scrollDisplayRight( ) ……144
6.3.12 print( ) ……145
6.4 Ethernet库 ……146
6.4.1 EthernetClass类定义 ……146
6.4.2 Server类定义 ……148
6.4.3 Server类构造函数 ……148
6.4.4 Server类成员函数 ……148
6.4.5 Client类定义 ……152
6.4.6 Client类构造函数 ……152
6.4.7 Client类成员函数 ……153
6.5 SoftwareSerial库 ……158
6.5.1 构造函数 ……159
6.5.2 begin( ) ……160
6.5.3 read( ) ……160
6.5.4 print( )和println( ) ……161
6.5.5 使用限制 ……164
6.6 EEPROM库 ……165
6.6.1 read( ) ……165
6.6.2 write( ) ……166
6.7 Wire库 ……166
6.7.1 IIC总线概述 ……166
6.7.2 TwoWire类定义 ……167
6.7.3 begin( ) ……168
6.7.4 requestFrom( ) ……168
6.7.5 available ( ) ……169
6.7.6 receive( ) ……169
6.7.7 beginTransmission( ) ……170
6.7.8 endTransmission( ) ……170
6.7.9 send( ) ……171
6.7.10 onReceive( ) ……172
6.7.11 onRequest( ) ……173

6.8 Servo库 ……174
6.8.1 构造函数 ……175
6.8.2 attach()……176
6.8.3 write( )……177
6.8.4 writeMicroseconds( )……177
6.8.5 read( )……178
6.8.6 readMicroseconds( )……178
6.8.7 attached( )……178
6.8.8 detach( )……179
6.9 Stepper库 ……179
6.9.1 步进电机概述……179
6.9.2 步进电机的基本参数……180
6.9.3 步进电机的优缺点……181
6.9.4 步进电机的工作原理……181
6.9.5 步进电机的控制电路……183
6.9.6 Stepper类定义……185
6.9.7 构造函数……186
6.9.8 setSpeed( )……188
6.9.9 step( )……188
6.10 TLC5940库 ……189
6.10.1 Tlc5940类的定义……190
6.10.2 init( )……191
6.10.3 update( )……192
6.10.4 set( )……193
6.10.5 get( )……194
6.10.6 setAll( )……194
6.10.7 clear( )……195
6.11 OneWire库 ……195
6.11.1 单总线的结构……195
6.11.2 单总线控制方式……195
6.11.3 单总线信号形式……196
6.11.4 OneWire类……198
6.11.5 构造函数……200
6.11.6 reset( )……200
6.11.7 write_bit( )……201
6.11.8 read_bit( )……202
6.11.9 write( )……202
6.11.10 read( )……203
6.11.11 select( )……203
6.11.12 skip( )……204
6.12 XBee库 ……204
6.12.1 XBee类定义……204
6.12.2 构造函数……205
6.12.3 begin( )……206
6.12.4 readPacket( )……206
6.12.5 send( )……209
6.13 创建自己的库 ……210
6.13.1 库的功能——Morse……210
6.13.2 MorseCode类的定义……213
6.13.3 MorseCode类的成员函数…214
6.13.4 MorseCode库的使用……222
6.13.5 关键字的定义……223

第7章 无线模块的应用 ……224

7.1 APC220 ……224
7.1.1 APC220性能指标……224
7.1.2 模块引脚定义……226
7.1.3 模块的使用……226
7.1.4 注意事项……227
7.2 DFduino wireless ……228
7.2.1 DFduino wireless性能指标…228
7.2.2 模块引脚定义……229
7.2.3 模块的使用……229
7.3 Bluetooth V3 ……231
7.3.1 Bluetooth V3性能指标……231
7.3.2 模块引脚定义……232
7.3.3 模块的使用……232
7.4 XBee和XBee PRO ……234

7.4.1 XBee及XBee PRO性能指标…235
7.4.2 模块引脚定义…235
7.4.3 模块的使用…236
7.4.4 程序设计…236

第三篇 应用篇

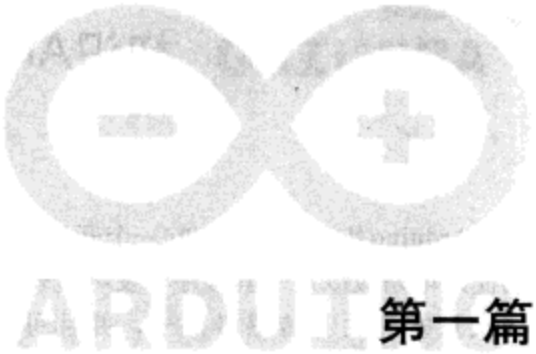
第8章 打造自己的遥控履带车…242
8.1 履带车的驱动…242
8.1.1 实现功能…242
8.1.2 所需器材…242
8.1.3 硬件连接…243
8.1.4 程序设计…246
8.1.5 MotorCar类…251
8.1.6 类的应用…255
8.2 添加感知器件…257
8.2.1 实现功能…257
8.2.2 所需器材…257
8.2.3 器材介绍…257
8.2.4 硬件连接…258
8.2.5 程序设计…258
8.3 添加无线模块…261
8.3.1 实现功能…261
8.3.2 所需器材…261
8.3.3 硬件连接…261
8.3.4 程序设计…262
8.4 制作遥控器…264
8.4.1 实现功能…264
8.4.2 所需器材…265
8.4.3 硬件连接…265
8.4.4 程序设计…265
8.5 履带车遥控调速…267
8.5.1 实现功能…267
8.5.2 程序设计…267
8.6 添加无线摄像头…272
8.6.1 实现功能…272
8.6.2 所需器材…272
8.6.3 器材介绍…272
8.6.4 硬件连接…273
8.6.5 程序设计…277
8.7 环境信息获取器件…283
8.7.1 实现功能…283
8.7.2 所需器材…284
8.7.3 器材介绍…284
8.7.4 硬件连接…285
8.7.5 程序设计…286

第9章 仿生机器人…295
9.1 遥控机械臂…295
9.1.1 实例功能…295
9.1.2 器材列表…295
9.1.3 搭建硬件环境…296
9.1.4 安装控制部分…298
9.1.5 Wii游戏手柄…298
9.1.6 机械臂程序设计…300
9.2 双足机器人…304
9.2.1 实例功能…304
9.2.2 器材列表…304
9.2.3 搭建硬件环境…305
9.2.4 双足机器人程序设计…307
9.2.5 PC调试软件编写…310
9.2.6 双足机器人的调试…317

附录A Arduino引脚与AVR单片机管脚对应关系…319

附录B Arduino扩展板…320

附录C 其他可扩展模块…322



第一篇

基 础 篇

第1章　初识Arduino

第2章　编写Arduino程序

第3章　Arduino的基本函数

第4章　Arduino硬件平台

第1章 初识Arduino

Arduino是源自意大利的一个开放源代码的硬件项目平台，该平台包括一块具备简单I/O功能的电路板以及一套程序开发环境软件。Arduino可以用来开发交互产品，比如它可以读取大量的开关和传感器信号，并且可以控制电灯、电机和其他各式各样的物理设备；Arduino也可以开发出与PC相连的周边装置，能在运行时与PC上的软件进行通信。Arduino的硬件电路板可以自行焊接组装，也可以购买已经组装好的模块，而程序开发环境的软件则可以从网上免费下载与使用。

1.1 Arduino的历史

说到Arduino的起源似乎有点令人感觉无心插柳柳成荫。Massimo Banzi是意大利米兰互动设计学院的教师，他的学生常常抱怨不能找到一块价格便宜且功能强大的控制主板来设计他们的机器人。2005年的冬天，Banzi和David Cuartielles讨论到这个问题，David Cuartielles是西班牙的微处理器设计工程师，当时在这所学校做访问研究。他们决定自己设计一块控制主板。他们找来了Banzi的学生David Mellis，让他来编写代码程序。David Mellis只花了两天时间就完成了代码的编写，然后又过了3天，板子就设计出来了，取名为Arduino。很快，这块板子受到了广大学生的欢迎。这些学生当中那些甚至完全不懂计算机编程的人，都用Arduino做出了“很炫”的东西：有人用它控制和处理传感器，有人用它控制灯闪烁，有人用它制作机器人……之后Banzi、Cuartielles和Mellis将设计图上传到网上，然后花了3000欧元加工出第一批板子。

Banzi等人当时加工了200块板子，卖给学校50块，起初还担心剩下的150块怎么卖出去，但是几个月后，他们的设计作品在网上得到了快速传播，接着他们收到了几个上百块板子的订单。这时他们明白Arduino是很有市场价值的，所以，他们决定开始Arduino的事业，但是有个原则——开源。他们规定任何人都可以复制、重设计甚至出售Arduino板子。人们不用花钱购买版权，连申请许可权都不用。但是，如果你加工出售Arduino原板，版权还是归Arduino团队所有。如果你是在基于Arduino的设计上修改，你的设计必须也和Arduino一样开源。

Arduino设计者们唯一所有的就是“Arduino”这个商标。如果你的设计也想用Arduino命

名，那么你就得支付费用。这样做是为了保护“Arduino”这个商标不被低劣的作品损坏。

对于最初决定硬件开源，几位设计者也有不同的动机。Cuartielles认为自己是个“左倾学术主义者”，不喜欢因为赚钱而限制大家的创造力，从而导致自己的作品得不到广泛使用。“如果有人要复制它，没问题。复制只会让它更出名。”Cuartielles在某次演讲中甚至说：“请你们复制它吧！”Banzi则恰恰相反，他更像一个精明的商人。他现在已经退休了，不再教书，开了一家科技设计公司。他猜想，如果Arduino开源，相比那些不开源的作品，会激发更多人的兴趣，从而得到更广泛的使用。还有一点就是，一些电子疯狂爱好者会去寻找Arduino的设计缺陷，然后要求Arduino团队做出改进。利用这种免费的劳动力，他们可以开发出更好的新产品。

实际情况也正如他所料，在接下来的几个月内，很多人提出重新布线、改进编程语言等建议。后来曾有销售商要求代理Arduino产品。2006年，Arduino方案获得了Prix Art Electronica电子通信类方面的荣誉奖。那一年，他们销售了5000块板子。第二年，他们销售了30 000块。Arduino被电子疯狂爱好者用来设计机器人、调试汽车引擎、制作无人飞机模型等。

1.2 Arduino的家族

Arduino设计之初的目的是希望让设计师和艺术家们能够很快地通过它学习电子和传感器的基础知识，并应用到他们的设计当中。设计中所要表现的想法和创意才是最主要的，至于单片机如何工作，硬件的电路是如何构成的，设计师和艺术家们并不需要考虑。

Arduino的出现，大大降低了互动设计的门槛，没有学过电子知识的人也能够使用它制作出各种充满创意的作品。越来越多的艺术家、设计师开始使用Arduino制作交互艺术品。为了针对不同的应用领域，目前Arduino已设计出很多不同的型号以满足不同使用者的需要，在这里简单介绍一下几类主要产品，详细信息可登录Arduino的主页http://www.arduino.cc查阅。

1. Arduino Duemilanove

这是一款基本的Arduino产品，控制器采用ATmega168或ATmega328，支持直流电源供电和USB端口供电，如图1.1所示。后续的很多产品都是在这款产品的基础上发展起来的。

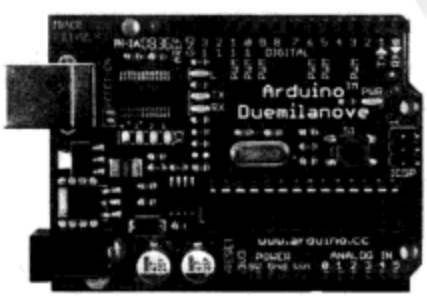


图1.1 Arduino Duemilanove

2. Arduino Nano

Arduino Nano在设计中去掉了直流电源接口，采用了Mini-B标准的USB接口来连接电脑，除了外观变了，其他接口及功能保持不变，控制器同样采用ATmega168或ATmega328，是一款缩小版的Arduino Duemilanove，如图1.2所示。

3. Arduino mini

考虑到存在一些对空间要求十分严格的使用者，Arduino mini（见图1.3）在设计时甚至去掉了USB接口和复位开关，这样能减小Arduino的尺寸。唯一的问题是连接电脑或烧写程序时需要一个USB或RS232转换成TTL的适配座，Arduino官方也有相应的适配座——Mini USB Adapter（http://www.arduino.cc/en/Main/MiniUSB上有相关的资料）。

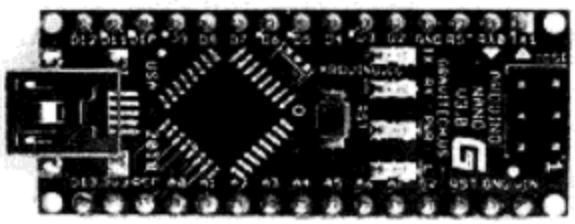


图1.2 Arduino Nano

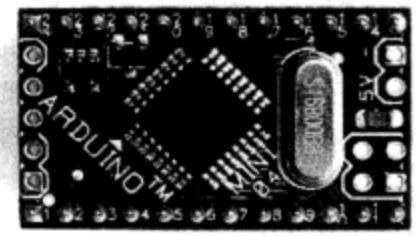


图1.3 Arduino mini

4. Arduino BT

Arduino BT（见图1.4）本身包含了一个Bluegiga WT11蓝牙模块，支持蓝牙无线串行通信，但不支持蓝牙音频设备。若没有USB接口，连接电脑或烧写程序可以通过蓝牙适配器与Arduino BT连接实现无线程序下载与控制。

5. LilyPad Arduino

这是一款真正有艺术气质的产品，面向的主要使用者是从事服装设计之类工作的设计师，它可以使用导电线或普通线缝在衣服或布料上，LilyPad Arduino每个引脚上的小洞大到足够缝纫针轻松穿过，如图1.5所示。如果用导电线缝纫的话，既可以起到固定的作用，又可以起到传导的作用。比起普通的Arduino板，LilyPad Arduino相对比较脆弱，比较容易损坏，但它的功能基本都保留了下来，除了一点，即它没有USB接口，所以LilyPad Arduino连接电脑或烧写程序时同Arduino mini一样需要一个USB或RS232转换成TTL的适配座。

6. Arduino Pro和Arduino Pro Mini

设计Arduino Pro的目的是为了那些需要便利性和低成本的高级用户。为了降低成本，它省去了USB接口、直流电源接口和引脚排针，连接电脑或烧写程序时需要一个USB或RS232转换成TTL的适配座。Arduino Pro更像是一个大号的Arduino mini，如图1.6所示。需要注意的是，Arduino Pro有3.3V/8MHz和5V/16MHz两个版本，使用的时候要留心点。另外Arduino Pro同样有一个Arduino Pro Mini的版本，如图1.7所示。

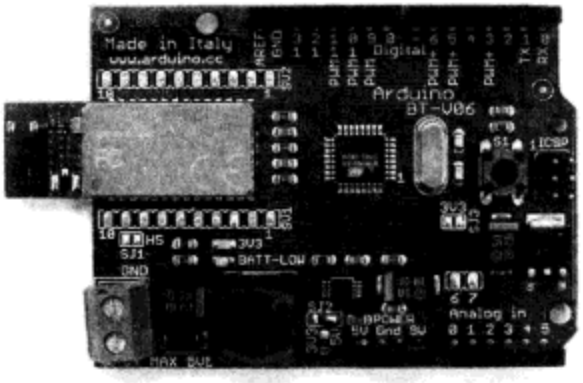


图1.4 Arduino BT

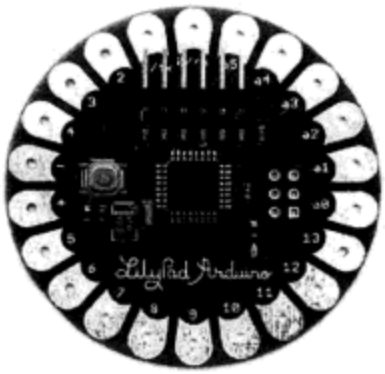


图1.5 LilyPad Arduino

图1.6 Arduino Pro

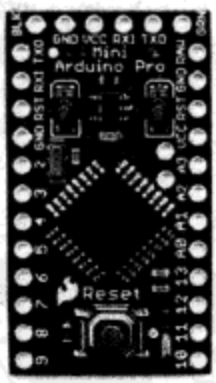


图1.7 Arduino Pro Mini

7. Arduino Fio

Arduino Fio（见图1.8）的工作电压是3.3V，控制器的工作频率是8MHz，采用了Mini-B标准的USB接口，提供一个锂聚合物电池接口，底部预留了一个XBee模块插座（美国DIGI的zigbee模块，本书的第7章有XBee模块的相关介绍，也可登录http://www.digi.com.cn了解XBee模块的更多信息），XBee模块可使Arduino方便地应用于无线网络。

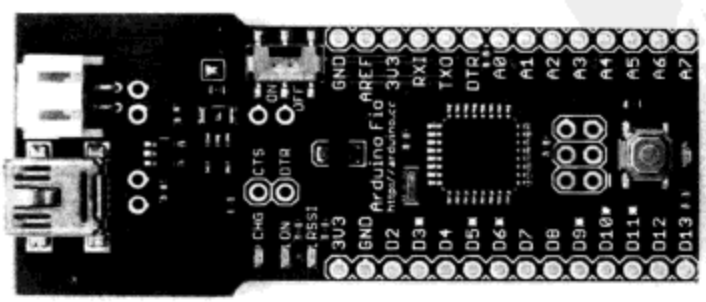


图1.8 Arduino Fio

8. Arduino Uno

Arduino Uno是最新的Arduino产品系列，如图1.9所示，它与之前的Arduino板最大的不同在于它不是使用FTDI USB-to-serial串行驱动器芯片，而是采用Atmega8U2芯片进行USB到串行数据的转换。目前Arduino Uno已成为Arduino主推的产品。

9. Arduino Mega2560

Arduino Mega2560（见图1.10）的控制器采用的是ATMega2560，它的资源要比之前的Arduino产品丰富很多，用于满足需使用较多资源进行产品设计与开发的用户需求，具体资源会在下一节描述。同时，Arduino Mega2560也兼容之前基于Arduino Duemilanove的设计。

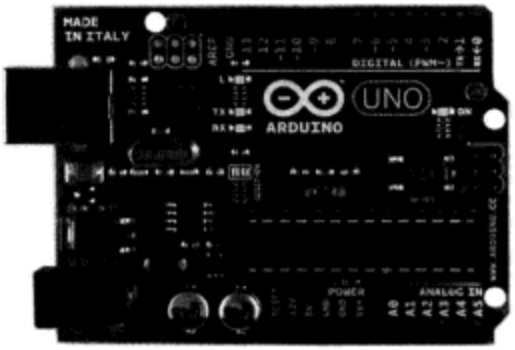


图1.9 Arduino Uno

图1.10 Arduino Mega2560

1.3 Arduino的资源

Arduino的硬件电路设计以创作公用约定（creative commons）的形式提供授权。相应的原理图和电路图都可以从Arduino网站上免费获得。Arduino Duemilanove具有14个数字I/O口（其中6个可提供PWM输出），6个模拟I/O口，一个复位开关，一个ICSP下载口，支持USB接口，可通过USB接口供电，也可以使用单独的7～12V电源供电。Arduino的资源在板子上已经明确标注，使用者可以很方便地了解具体的资源分配，DIGITAL一边有14个数字I/O口0～13，ANALOG IN一边有6个模拟I/O口0～5，其他还有POWER、TX、RX、PWM等标识，如图1.11所示。

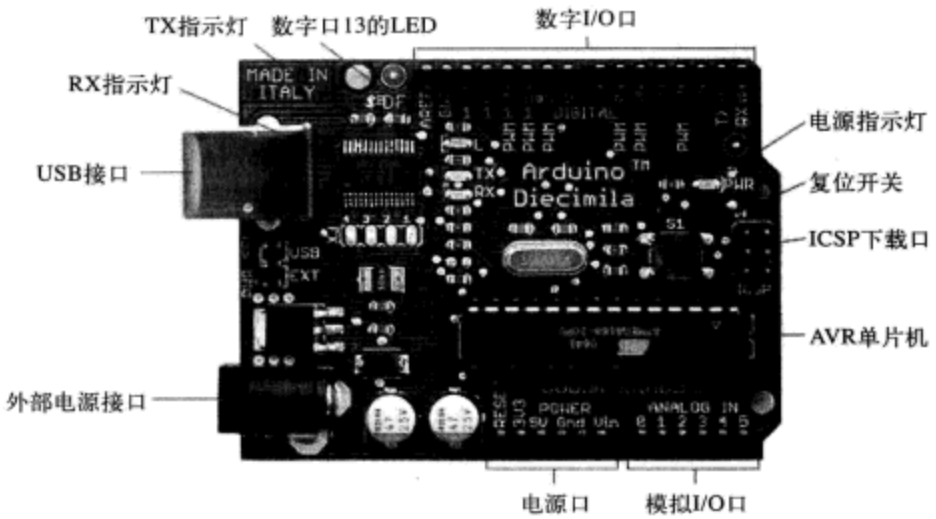


图1.11 Arduino的资源

Arduino Duemilanove总体参数如表1.1所示。

表1.1 Arduino总体参数

名 称	参 数
微控制器	ATmega168/ATmega328
操作电压	5V
推荐输入电压	7～12V
极限输入电压	6～20V
数字I/O脚数	14，其中6个提供PWM输出
模拟输入脚数	6
I/O脚直流电流	40 mA
3.3伏脚的电流	50 mA
闪存	16 KB (ATmega168)或32 KB (ATmega328)，其中2KB用于引导程序
SRAM	1 KB (ATmega168)或2 KB (ATmega328)
EEPROM	512 byte (ATmega168)或1 KB (ATmega328)
时钟频率	16 MHz
尺寸	6.0cm × 5.33cm

各引脚定义如下：

- ❑ 数字引脚：0～13
- ❑ 模拟引脚：A0～A5（为区分数字引脚，在引脚号前加A）
- ❑ 串行通信：0，1（0作为RX，接收数据；1作为TX，发送数据）
- ❑ 外部中断：2，3
- ❑ PWM输出：3，5，6，9，10，11
- ❑ SPI通信：10（SS），11（MOSI），12（MISO），13（SCK）

- 板上LED：13
- TWI通信：A4（SDA），A5（SCL）

另外，Arduino有一个可复位的熔断器来保护计算机的USB口，防止短路或者过流。虽然大部分计算机都有它们内置的保护措施，但是熔断器还是可以提供额外的保护，如果连到USB口的电源超过500mA，它将自动断开，直到短路或者过流消除。

Arduino Uno的硬件资源与Arduino Duemilanove相同，Arduino Mega2560的资源就要丰富多了，它具有54个数字I/O口（其中14个可提供PWM输出），16个模拟I/O口，4对串行数据通信口（UART），一个复位开关，一个ICSP下载口，支持USB接口和直流电源供电。Arduino Mega2560总体参数如表1.2所示。

表1.2 Arduino总体参数

名 称	参 数
微控制器	ATmega2560
操作电压	5V
推荐输入电压	7～12V
极限输入电压	6～20V
数字I/O脚数	54，其中14个提供PWM输出
模拟输入脚数	16
I/O脚直流电流	40 mA
3.3伏脚的电流	50 mA
闪存	256KB，其中8KB用于引导程序
SRAM	8 KB
EEPROM	4 KB
时钟频率	16 MHz

各引脚定义如下：

- 数字引脚：0～53
- 模拟引脚：A0～A15（为区分数字引脚，在引脚号前加A）
- 串行通信：Arduino Mega2560提供4组串行通信端口，分别是0（RX）和1（TX）用作串口1，19（RX）和18（TX）用作串口2，17（RX）和16（TX）用作串口2，15（RX）和14（TX）用作串口3
- 外部中断：Arduino Mega2560提供6个引脚作为外部中断，分别是2（外部中断0），3（外部中断1），21（外部中断2），20（外部中断3），19（外部中断4），18（外部中断5）
- PWM输出：0～13
- SPI通信：53（SS），51（MOSI），50（MISO），52（SCK）
- 板上LED：13
- TWI通信：20（SDA），21（SCL）

1.4 Arduino的开发环境

Arduino的开发环境是以AVR-GCC和其他一些开源软件为基础，采用Java编写的，软件无需安装，下载完成解压缩后就可以直接打开使用了。软件可以在Arduino的网站http://www.arduino.cc上免费下载，目前最新版本是0022。本书中使用的版本是0021。Arduino开发环境使用的语法与C/C++相似，非常容易使用。图1.12所示的就是Arduino开发环境的主界面，中间的白色区域就是程序编辑区，下方的黑色区域为信息提示区。

图1.12 Arduino的开发环境

提示： 目前的软件界面中无法输入中文字符，可在其他软件内输入中文字符再拷贝至软件界面内。

1.5 添加新硬件及设置开发环境

在应用Arduino开发环境之前，先要在电脑端添加新硬件Arduino控制板，本节介绍Arduino Duemilanove和Arduino Uno两块控制板的安装。

先看看Arduino Duemilanove控制板的添加。准备一块Arduino Duemilanove板和一条USB连接线。第一次将Arduino板连接到电脑上，会出现Found New Hardware Wizard（发现

新硬件向导）的提示，依照提示完成驱动安装，步骤如图1.13～图1.18所示。

图1.13 选择从指定目录安装

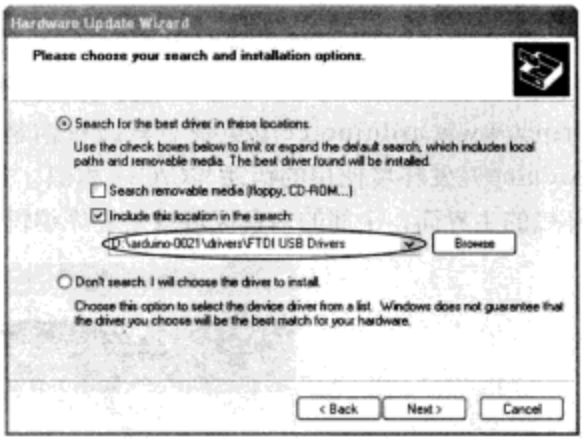


图1.14 可在Arduino开发环境目录下的drivers文件夹中找到驱动程序

图1.15 选择Finish完成驱动安装

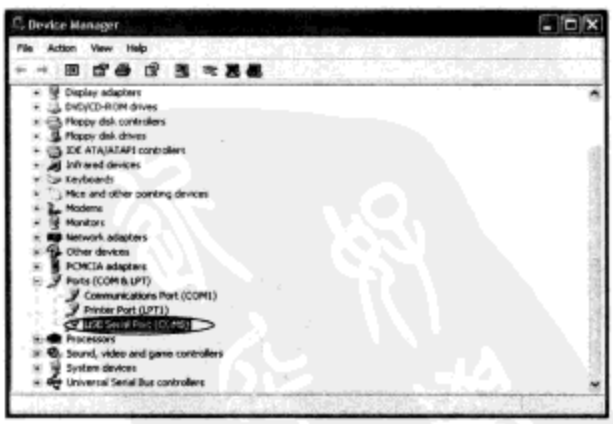


图1.16 在Device Manager（设备管理器）里查看新安装的USB转串口设备所对应的串口号

正确设置串口和Arduino板系列型号后，就可以开始Arduino之旅了。接下来看一下Arduino Uno控制板的添加。准备一块Arduino Uno板和一条USB连接线。第一次连接到电脑，会出现“发现新硬件向导”的提示，依照提示完成驱动安装。若错过了“发现新硬件向导”的提示，也可以在设备管理器中找到未安装的新硬件，如图1.19所示。

右键选中未识别的硬件Arduino Uno，从弹出的如图1.20所示的快捷菜单中选择Update Driver（更新驱动程序），将会出现如图1.21所示的Hardware Update Wizard（更新硬件驱动向导）的提示，依照提示完成Arduino Uno驱动的安装。

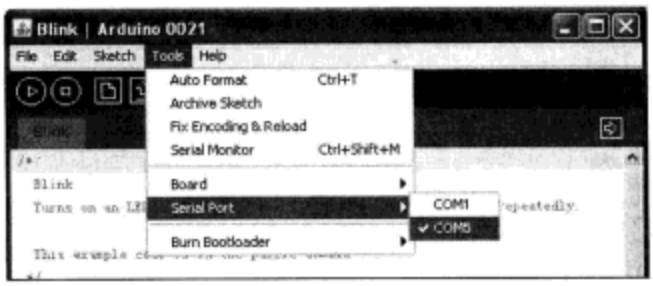


图1.17 在Arduino的开发环境中设置串口（Tools→Serial Port→COM5）

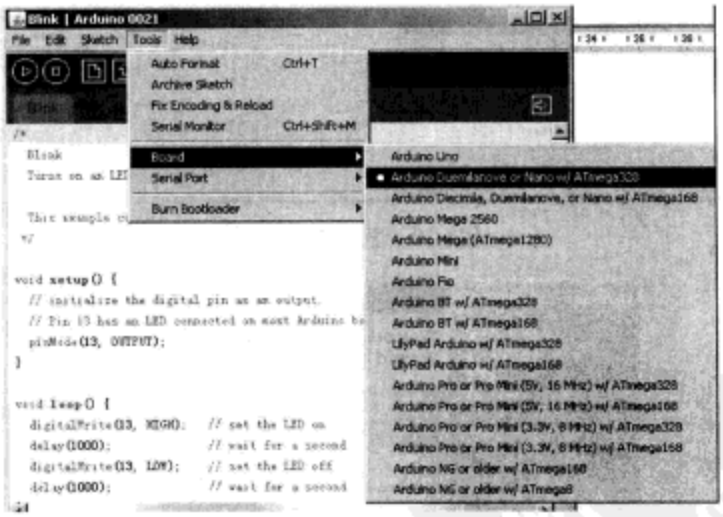


图1.18 选择正确的Arduino板系列型号

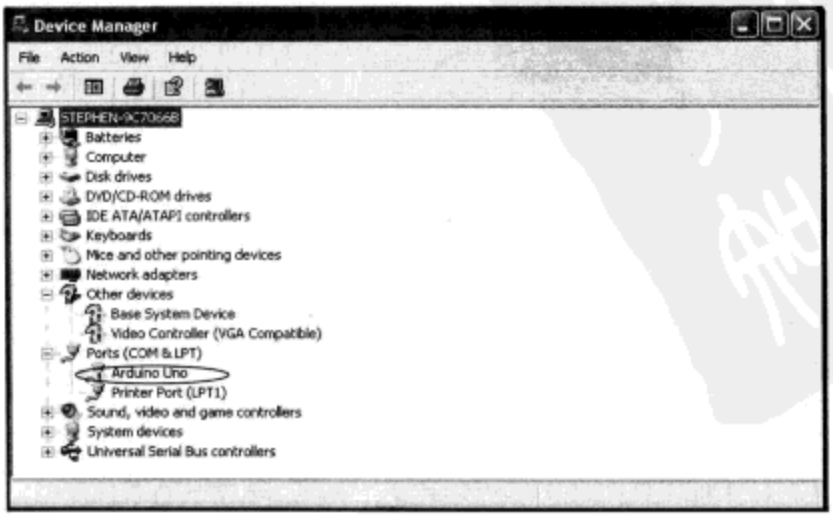


图1.19 设备管理器中未安装的新硬件

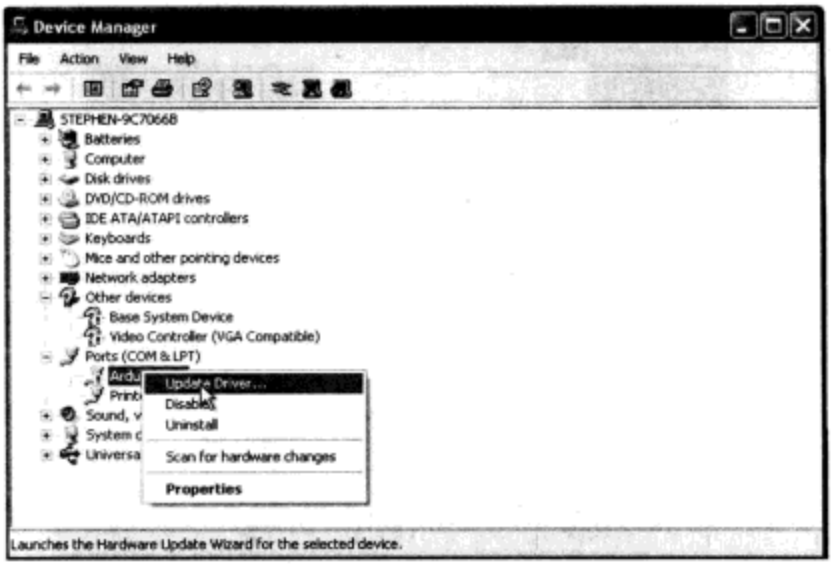


图1.20 更新驱动程序

图1.21 更新硬件驱动向导

如图1.13、图1.14所示选择正确的驱动程序，此时电脑会出现驱动程序不被Windows信任的提示，选择Contine Anyway（仍然继续），如图1.22所示。

最后选择Finish完成硬件驱动的更新，如图1.23所示。

此时设备管理器中的新硬件Arduino Uno就被识别成串口设备，如图1.24所示。

最后要在Arduino开发环境中设置相应的串口号以及Arduino板型号，如图1.17和图1.18所示，注意Arduino板型号的选择，要选择Arduino Uno，如图1.25所示。

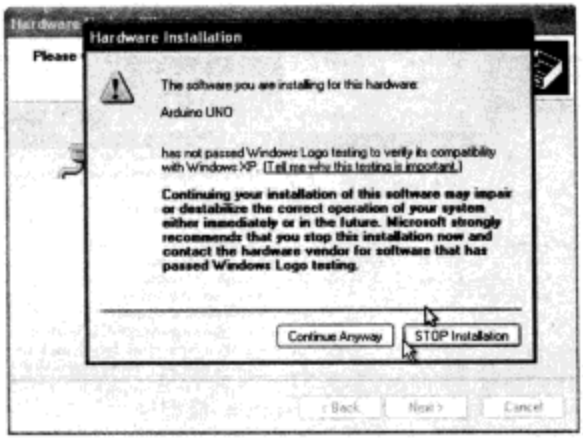


图1.22 驱动程序不被信任

图1.23 完成硬件驱动的更新

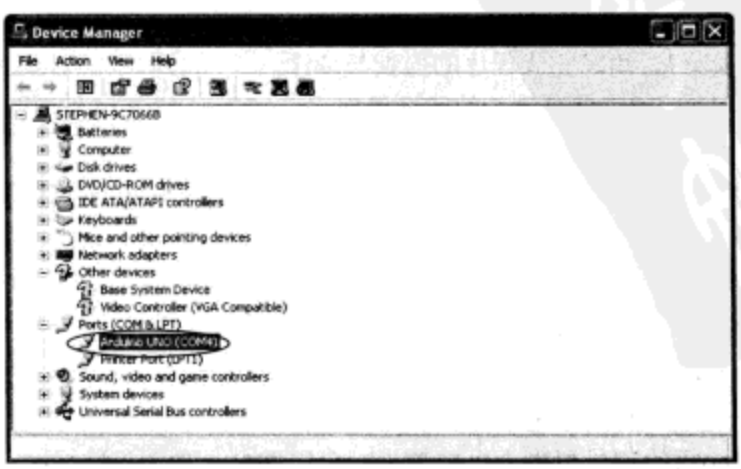


图1.24 设备管理器中的Arduino Uno

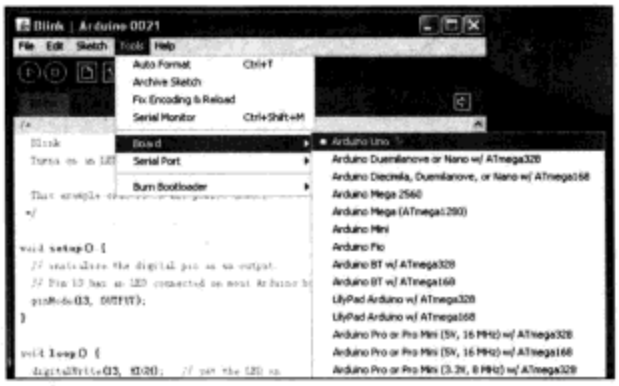


图1.25 选择Arduino板型号

提示：本书后面的章节使用的控制板均为Arduino Uno。

1.6 Arduino开发环境的应用

Arduino开发环境中菜单栏下方的7个按钮必须先了解一下，它们依次是Verify（校验）、Stop（停止）、New（新建）、Open（打开）、Save（保存）、Upload（上传）、Serial Monitor（串口监视窗），如图1.26所示。

图1.26 Arduino开发环境中菜单栏下方的7个按钮

各按钮的具体功能如下：

- Verify（校验），用以完成程序的检查与编译。
- Stop（停止），用以停止进行的编译操作。
- New（新建），可新建一个程序文件。
- Open（打开），打开一个存在的程序文件，Arduino开发环境下的程序文件后缀名为.pde。
- Save（保存），保存当前的程序文件。
- Upload（上传），将编译后的程序文件上传到Arduino板中。
- Serial Monitor（串口监视窗），可监视开发环境使用的串口收发的数据。

接下来通过一个Arduino开发环境中LED灯闪烁的例子（Blink）来简单应用一下这些按钮。在Arduino Uno板的13号引脚上已经带了一个LED灯，如图1.27所示。Blink程序就是控制这个LED灯闪烁，我们可以在Arduino控制板上明确地看到。

图1.27 13号引脚及LED灯位置示意

点击Open按钮后，依次选择1.Basics→Blink，就可以看到Blink程序已经加载到程序编辑区，如图1.28所示。

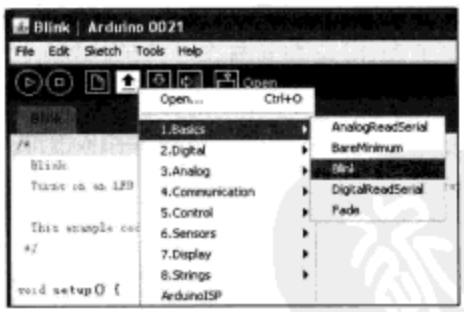


图1.28 加载Blink程序

点击“校验”按钮实现程序的编译，等待一会儿后状态栏会提示Done compiling（程序编译完成），信息提示区内会显示程序编译完成后的大小，如图1.29所示，此例中Blink程序编译后的大小为1010bytes。

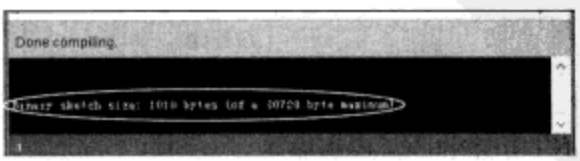


图1.29 Blink程序编译后的大小

编译完成后点击“上传”按钮，上传一般需要等待几秒钟时间，在上传的时候串口的指示灯（RX和TX）会不停地闪烁。上传完成后状态栏会有上传成功的提示：Done uploading。观察Arduino控制板上LED灯是否在不停地闪烁，Arduino控制板通过LED闪烁的方式告诉你，你已经会使用Arduino了，就是这么简单。

第2章 编写Arduino程序

Arduino项目很注重动手能力，只有通过动手才会发现Arduino的优点。本章介绍如何编写Arduino程序。

写程序就像盖房子。盖房子首先要有图纸，然后依照图纸搭建主体，最后内外部装修。写程序大致也需要这几步，首先绘制流程图，然后使用控制语句搭建主体程序，最后查错、调试、修改、添加注释完成程序编写。

本章内容就按照这样一个次序展开，让我们共同来搭建Arduino“大厦”。

2.1 绘制流程图

2.1.1 流程图基本符号

美国国家标准化协会（American National Standards Institute, ANSI）规定了一些常用的流程图符号，目前已被世界各国的多个领域普遍采用，如图2.1所示。

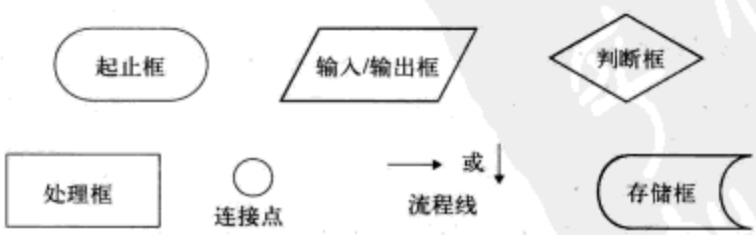


图2.1 流程图基本符号

第1章的Blink程序可以用图2.2的流程图描述。

通过Blink程序的流程图可以看出，一个流程图应该包括以下几部分：

- 表示相应操作的框。
- 带箭头的流程线。
- 框内外必要的文字说明。

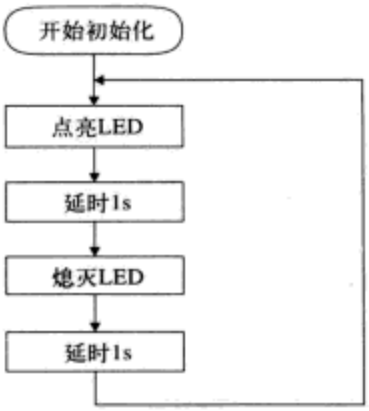


图2.2 Blink程序流程图

2.1.2 流程图的三种基本结构

起初流程图用流程线指出各框的执行顺序，对流程线的使用没有严格的规定，流程线在程序中随意地连接，流程图也就没有实现使程序直观形象、简单清晰的目的，同样，编写的程序也是逻辑混乱、难以理解。

为了提高流程图及程序的逻辑性，使其更容易理解、更方便阅读，必须限制流程线的使用，不允许流程线无规律地连接，而是按照一定顺序和条件进行连接。于是，1966年，Bohra和Jacopini提出了三种基本结构，用这三种基本结构作为表示一个良好算法的基本单元。

1. 顺序结构

如图2.3所示，虚线框内是一个顺序结构。其中A和B两个框是顺序执行的。顺序结构是最简单的一种基本结构。

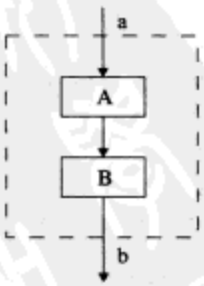


图2.3 顺序结构

2. 选择结构

如图2.4所示，虚线框内是一个选择结构。此结构中包含一个判断框，根据条件是否成立而选择执行A还是B，执行完成后，经过b点脱离选择结构。

3. 循环结构

如图2.5所示，虚线框内是一个循环结构。此结构中也有一个判断框用来决定是否跳出循环结构。有两种循环结构：判断框成立跳出循环的称为until型循环；判断框不成立跳出循环的称为while型循环。

以上三种基本结构有如下特点：

- 结构内的每一部分都有机会被执行到。
- 结构内不存在无法跳出的循环。
- 只有一个入口，即图中的a点。

❑ 只有一个出口，即图中的b点。

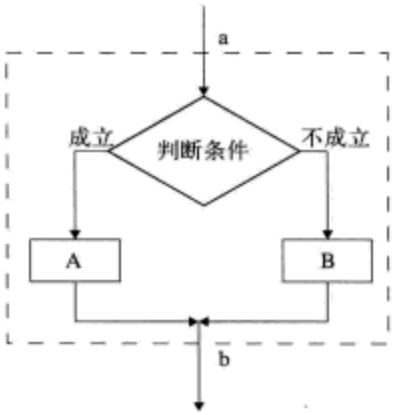


图2.4 选择结构

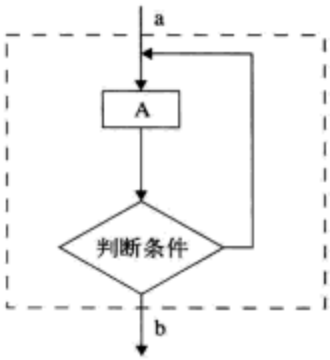


图2.5 循环结构

注意：一个判断框有两个出口，而一个选择结构只有一个出口，不要将二者混为一谈。

这三种基本结构可解决任何复杂的问题，由基本结构所构成的程序流程不存在无规律的转向，只在本基本结构内才允许存在分支和跳转。

2.2 C语言的标识符与关键字

Arduino的开发使用的是C语言，在C语言中使用的词汇大致可分为6类：标识符、关键字、运算符、分隔符、常量和注释符。

2.2.1 标识符

标识符用来标识源程序中某个对象的名字，这些对象可以是语句、数据类型、函数、变量、常量等。一个标识符由字符串、数字和下划线等组成，第一个字符必须是字母或下划线，通常以下划线开头的标识符是编译系统专用的，因此在编写C语言源程序时一般不要使用以下划线开头的标识符，而将下划线用作分段符。

2.2.2 关键字

关键字是编程语言保留的特殊标识符，它们具有固定的名称和含义，ANSI C标准一共规定了32个关键字，如表2.1所示。

表2.1 ANSI C标准规定的32个关键字

关 键 字	用 途	说 明
auto	存储种类说明	用于说明局部变量，为默认值
break	程序语句	退出最内层循环体
case	程序语句	switch语句中的选择项
char	数据类型说明	字符型数据
const	存储种类说明	程序中不可更改的常量值
continue	程序语句	转向下一次循环
default	程序语句	switch语句中的失败选择项
do	程序语句	构成do-while循环结构
double	数据类型说明	双精度浮点数
else	程序语句	构成if-else选择结构
enum	数据类型说明	枚举
extern	存储种类说明	在其他程序模块中说明了的全局变量
float	数据类型说明	单精度浮点数
for	程序语句	构成for循环结构
goto	程序语句	构成goto转移结构
if	程序语句	构成if-else选择结构
int	数据类型说明	整型数
long	数据类型说明	长整型数
register	存储种类说明	使用CPU内部寄存器的变量
return	程序语句	函数返回
short	数据类型说明	短整型数
signed	数据类型说明	有符号数，二进制数据中最高位为符号位
sizeof	运算符	计算表达式或数据类型的字节数
static	存储种类说明	静态变量
struct	数据类型说明	结构类型数据
switch	程序语句	构成switch选择结构
typedef	数据类型说明	重新进行数据类型定义
union	数据类型说明	联合类型数据
unsigned	数据类型说明	无符号数据
void	数据类型说明	无类型数据
volatile	数据类型说明	该变量在程序执行中可被隐含地改变
while	程序语句	构成while和do-while循环结构

2.2.3 运算符

C语言中含有相当丰富的运算符。运算符与变量、函数一起组成表达式，表示各种运算功能，在任意一个表达式的后面加一个分号“；”就构成了一个表达式语句。表2.2列出了C语言常用的运算符。

表2.2 C语言常用的运算符

类 型	运算符	说 明
算术运算符	+	加或取正值运算符
	−	减或取负值运算符
	*	乘运算符
	/	除运算符
	%	模运算符
关系运算符	>	大于
	<	小于
	>=	大于或等于
	<=	小于或等于
	==	测试等于
	!=	测试不等于
逻辑运算符	||	逻辑或
	&&	逻辑与
	!	逻辑非
赋值运算符	+=	加法赋值运算符
	−=	减法赋值运算符
	*=	乘法赋值运算符
	/=	除法赋值运算符
	%=	取模赋值运算符
	>>=	右移位赋值运算符
	<<=	左移位赋值运算符
	&=	逻辑与赋值运算符
	|=	逻辑或赋值运算符
	~=	逻辑非赋值运算符
	^=	逻辑异或赋值运算符
自增和自减运算符	++	自增运算符
	−−	自减运算符
逗号运算符	,	将多个表达式连接起来，依次执行
条件运算符	? :	
位运算符	~	取反
	<<	左移
	>>	右移
	&	与
	^	异或
	|	或
求字节运算符	sizeof	求取数据类型、变量以及表达式的字节数的运算符

注意：sizeof是一种特殊的运算符，它不是一个函数。实际上，字节数的计算在编译时就完成了，而不是在程序执行的过程中才计算出来的。

2.2.4 分隔符

C语言中采用的分隔符有逗号和空格两种。逗号主要用在类型说明和函数参数表中，用于分隔各个变量。空格多用于语句各单词之间作间隔符。在关键字、标识符之间必须要有一个以上的空格符作间隔。

2.2.5 常量

常量就是在程序运行过程中，其值不能改变的数据。有时候也可以用一些有意义的符号来代替常量的值，称为符号常量。符号常量在使用之前必须先定义，其一般形式如下：

```
#define   标识符   常量
```

2.2.6 注释符

C语言的注释符包括两种：

- 以“/*”开头并以“*/”结尾的字符串。在“/*”与“*/”之间的内容即为注释。
- “//”后面的字符串。

程序在编译时，不对注释作任何处理。注释可出现在程序的任何位置。编程时添加适当的注释对于程序员读懂该段程序非常有用。

2.3 控制语句

在程序的执行过程中，往往需要根据某些条件来决定执行哪些语句，这就需要选择型控制语句if和switch来实现选择结构程序，某些情况下还会不断地重复执行某些语句，这就需要循环型控制语句for和while来完成循环结构程序。

2.3.1 if语句

用if语句可以实现选择结构。它根据给定的条件进行判断，以决定执行某个分支程序段。if语句有3种基本形式。

1. 第一种基本形式

```
if（表达式）
      语句
```

功能描述：如果表达式的值为真，则执行其后的语句；否则，跳过该语句。

2. 第二种基本形式

```
if (表达式)
        语句1
else
        语句2
```

功能描述：如果表达式的值为真，则执行语句1；如果表达式的值为假，则执行语句2。

3. 第三种基本形式

```
if (表达式1)
        语句1
else if (表达式2)
        语句2
else if (表达式3)
        语句3
……
else if (表达式n)
        语句n
else
        语句m
```

功能描述：如果表达式1的结果为真，则执行语句1，然后退出if选择语句，不执行下面的语句；否则，判断表达式2，如果表达式2的结果为真，则执行语句2，然后退出if选择语句，不执行下面的语句，同样如果表达式2的结果为假则判断表达式3，依次类推，最后，如果表达式n不成立，则执行else后面的语句m。

在使用if语句时还要注意以下问题：

- 在3种基本形式中，if关键字后面均为表达式。该表达式通常是逻辑表达式或关系表达式，也可以是一个变量。
- 在if语句中，条件判断表达式必须用括号括起来。在语句之后必须加分号，如果是多行语句组成的程序段，则要用花括号括起来。

2.3.2 switch语句

switch语句可实现多分支的选择结构，在这种情况下，判断条件表达式的值是由几段组成或不是一个连续的值，每一段或每一个值对应一段分支程序。switch语句的一般形式为：

```
switch (表达式)
{
    case 常量表达式1:
            语句1
    case 常量表达式2:
            语句2
```

```
    ……
    case 常量表达式n:
            语句n
    default:
            语句m
}
```

switch语句的流程图如图2.6所示。

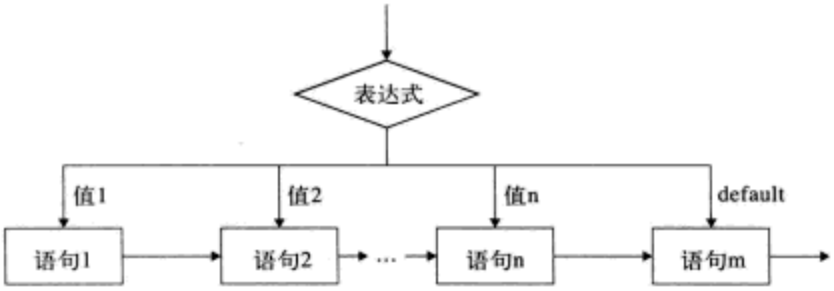


图2.6 switch语句的流程图

功能描述：计算表达式的值，并逐个与其后的常量表达式的值进行比较。当表达式的值与某个常量表达式的值相等时，即执行其后的语句，然后不再进行判断，继续执行所有case后面的语句。如果表达式的值与所有case后的常量表达式均不相等，则执行default后的语句。

在使用switch语句时要注意以下问题：

- 表达式的计算结果必须是整型或者字符型，也就是常量表达式1到常量表达式n必须是整型或字符型常量。
- 每个case的常量表达式必须互不相同，但各个case出现的次序没有顺序。case语句标号后面的语句可以省略不写，在关键字case和常量表达式之间一定要有空格。
- 当表达式的值与某个常量表达式的值相等并执行完其后的语句时，如果不想继续执行所有case后面的语句，则要在语句后面加上“break”，以跳出switch结构。

2.3.3 while语句

while语句能够实现“当型”循环结构，其一般形式为：

```
while (表达式) 语句
```

功能描述：计算表达式的值，当值为真时，执行循环体语句；当表达式的值为假时，跳出循环体，结束循环。其中，表达式是循环条件，语句是循环体。

while语句的流程图如图2.7所示。

在使用while语句时要注意以下几点：

- 不要混淆while语句构成的循环结构与if语句构成的选择结构。while的条件表达式为

真时，其后的循环体将被重复执行；而if的条件表达式为真时，其后的语句只执行一次。

- 在循环体中应有使循环趋于结束的语句。如果没有，则会进入死循环。在编写嵌入式应用程序时，我们经常会用到死循环。
- 循环体若包含一个以上的语句，应使用大括号括起来。

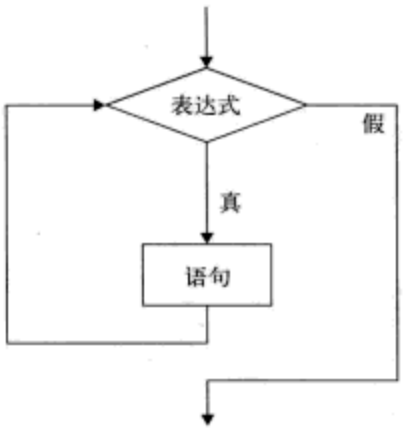


图2.7 while语句的流程图

2.3.4 do-while语句

do-while语句用来实现“直到型”循环，其特点是先执行循环体，然后判断循环条件是否成立，其一般形式为：

```
do
        语句
while（表达式）；
```

do-while语句流程图如图2.8所示。

功能描述：由于do-while循环为“直到型”循环，它先执行循环体中的语句，然后再判断表达式是否为真，如果为真则继续循环；如果为假则终止循环。因此，do-while循环至少执行一次循环体语句。

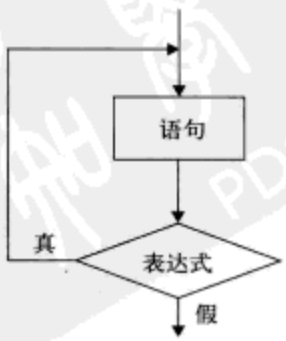


图2.8 do-while语句流程图

注意：do-while语句在使用时除了要注意循环体至少执行一次的问题外，在使用时还要注意它是以do开始，以while结束，while（表达式）后的分号不能丢。

2.3.5 for语句

for语句的使用极为灵活，可以完全取代while语句。它既可以用于循环次数确定的情况，又可以用于循环次数不确定而只是给出循环条件的情况。for语句的一般形式为：

```
for（表达式1；表达式2；表达式3）语句
```

for语句的流程图如图2.9所示。

功能描述：先求解表达式1，一般情况下，表达式1为循环结构的初始化语句，给循环计数器赋初值。然后求解表达式2，若其值为假，则终止循环；若其值为真，则执行for语句中的内嵌语句。内嵌语句执行完后，求解表达式3。最后继续求解表达式2，根据求解值进行判断，直到表达式2的值为假。

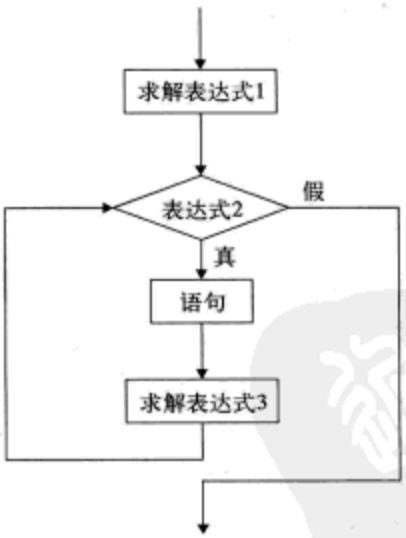


图2.9 for语句的流程图

for语句最简单也是最典型的形式如下：

```
for（循环变量赋初值；循环条件；循环变量增量）语句
```

循环变量赋初值总是一个赋值语句，用来给循环控制变量赋初值。循环条件是一个关系表达式，决定什么时候退出循环。循环变量的增量用来定义循环控制变量每次循环后按什么方式变化。这3个部分之间用分号分开。

for循环语句的一般形式可用while语句进行解释，如下：

```
表达式1；
while（表达式2）
{
        语句
```

```
        表达式3;
}
```

或者用do-while语句解释，如下：

```
表达式1;
do{
        语句
        表达式3;
}while (表达式2) ;
```

在使用for语句时要注意以下几点：

- for循环中的表达式1、表达式2和表达式3都是选择项，但是分号不能省略。
- 若3个表达式都省略，则for循环变成for（；；），相当于while（1）死循环。
- 表达式2一般是关系表达式或逻辑表达式，但也可以是数值表达式或字符表达式，只要其值非零，就执行循环体。

2.3.6 break语句

break语句通常用在循环语句和switch语句中。当break用在switch语句中时，可使程序跳出switch而执行switch以后的语句。

当break语句用在do-while、for、while循环语句中时，可使程序终止循环而执行循环后面的语句。通常break语句总是与if语句连在一起的，即满足条件时便跳出循环。

2.3.7 continue语句

continue语句的作用是跳过循环体中剩余的语句而强行执行下一次循环。continue语句只用在for、while、do-while等循环体中，常与if条件语句一起使用，用来加速循环。

注意：continue语句与break语句的区别是，break语句结束整个循环过程，而continue语句只结束本次循环，不终止整个循环。

2.3.8 goto语句

goto语句是一个无条件转向语句，它的一般形式为：

```
goto 语句标号;
```

功能描述：语句标号是一个带冒号“:”的标识符，用于标识语句的地址。当执行跳转语句时，使程序跳转到标识符指向的位置继续执行。将goto语句和if语句一起使用，可以构成一个循环结构。一般常见的是采用goto语句来跳出多重循环。注意，只能用goto语句从内层循环跳到外层循环，而不允许从外层循环跳到内层循环。

注意：由于goto语句容易使程序层次不清，所以在结构化程序设计中不主张使用goto语句。

2.4 程序结构

一般情况下在C语言中要求一个源程序不论由多少个文件组成，都必须有一个主函数，即main函数，且只能有一个主函数，C语言程序执行是从主函数开始的。但在Arduino中，主函数main在内部定义了，使用者只需要完成以下两个函数就能够完成Arduino程序的编写，这两个函数分别负责Arduino程序的初始化部分和执行部分。

- void setup( )
- void loop( )

两个函数均为无返回值的函数，setup( )函数用于初始化，一般放在程序开头，主要工作是用于设置一些引脚的输出/输入模式、初始化串口等，该函数只在上电或重启时执行一次；loop( )函数用于执行程序，它是一个死循环，其中的代码将被循环执行，用于完成程序的功能，如读入引脚状态、设置引脚状态等。

结合第1章的Blink程序及2.1节的程序流程图，来看看这两个函数的作用。

```
void setup( )
{
      //注释：初始化Arduino的引脚13为输出，Arduino板上自带的LED连接在引脚13上
      pinMode(13, OUTPUT);
}

void loop( )
{
      digitalWrite(13, HIGH);          //引脚13置高，输出+5V电压，LED点亮
      delay(1000);                     //等待1000ms
      digitalWrite(13, LOW);           //引脚13置低，输出0V电压，LED熄灭
      delay(1000);                     //等待1000ms
}
```

在setup函数中设置连接LED的引脚13为输出，以控制LED的亮或灭，这个操作只在上电或重启时执行一次，之后就没有必要执行了。

在loop函数中有4条语句，分别执行的操作是：

1）设置引脚13输出高，LED点亮；

2）等待1s；

3）设置引脚13输出低，LED熄灭；

4）等待1s。

由于loop( )函数中的代码将被循环执行，所以在第4步执行完成后，将回到第1步继续执行，程序不断循环，我们就看到了LED闪烁的效果。

setup函数和loop函数与流程图的对应关系如图2.10所示。

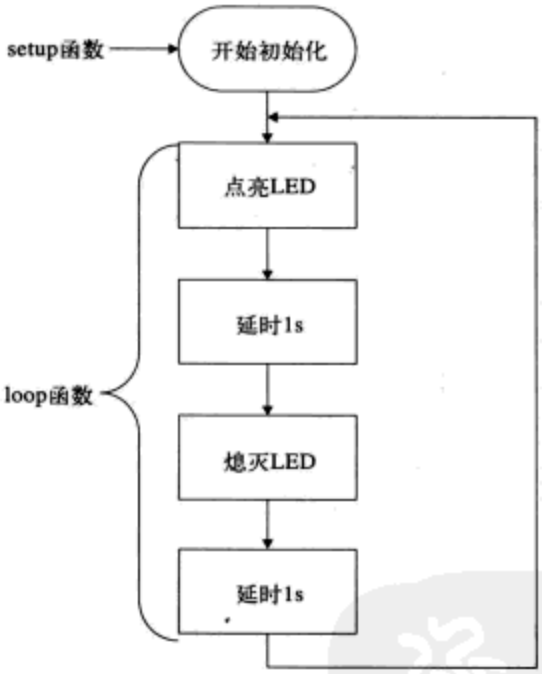


图2.10 setup函数和loop函数与流程图的对应关系

第3章 Arduino的基本函数

第2章已经介绍了如何编写Arduino程序，是不是感觉很容易上手，使用很方便。但Arduino真正的低门槛、硬件无关性的优点是因为在Arduino开发环境下提供了大量的基础函数，这些基础函数涉及I/O控制、时间函数、数学函数、三角函数等，使用者可以很方便地对板上的资源进行控制。同时，在Arduino开发环境下还提供了许多的实例程序来使用这些基础函数，这更加快了使用者的上手速度，这些实例程序可以在开发环境的File→Examples菜单下找到，其中包括之前提到的Blink程序，如图3.1所示。

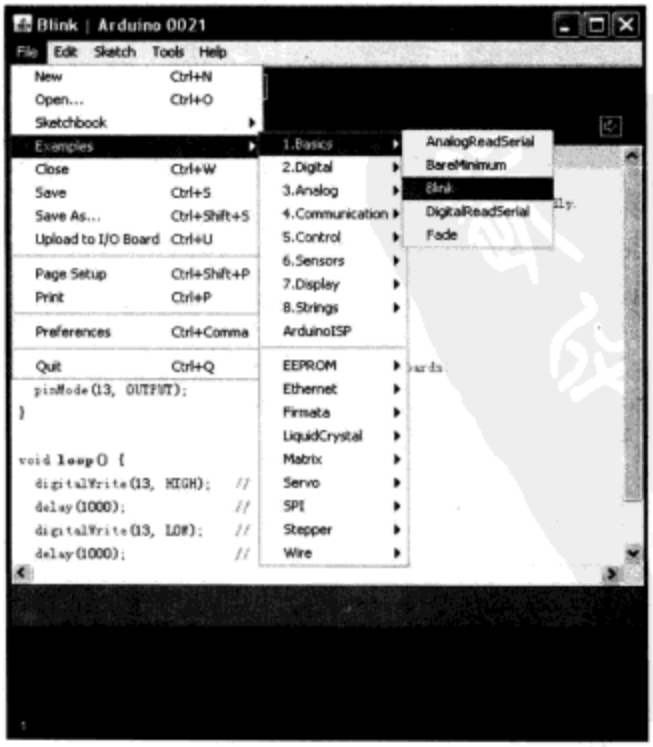


图3.1 实例程序可以在开发环境下找到

提示：函数原型部分的内容建议在单独深入学习过AVR单片机的基础上进行，跳过函数原型的部分不会影响对Arduino的应用。

3.1 数字I/O

3.1.1 pinMode(pin, mode)

pinMode函数在第2章中已经出现过了，用以配置引脚为输出或输出模式，它是一个无返回值函数，函数有两个参数pin和mode，pin参数表示所要配置的引脚，mode参数表示设置的模式——INPUT（输入）或OUTPUT（输出）。

注意：Arduino板上的模拟引脚也可以当做数字引脚使用，编号为14（对应模拟引脚0）到19（对应模拟引脚5）。

由于Arduino项目是完全开源的，所以pinMode(pin, mode)函数原型可直接在Arduino开发环境目录下的hardware\arduino\cores\arduino文件夹里的wiring_digital.c文件中查看。

函数原型有助于我们深入了解Ardnino的基本函数的底层实现方式，但这部分的内容需要在单独深入学习AVR单片机的基础上进行，本书将这些函数原型从文件中提取出来，有兴趣的读者可以参考一下。一般只要能够熟练地使用这些Arduino基本函数就可以了，本书对函数原型没有进行过多讲解。

pinMode(pin, mode)函数原型：

```
void pinMode(uint8_t pin, uint8_t mode)
{
      uint8_t bit = digitalPinToBitMask(pin);
      uint8_t port = digitalPinToPort(pin);
      volatile uint8_t *reg;

      if (port == NOT_A_PIN)
      return;

      reg = portModeRegister(port);

      if (mode == INPUT)
      {
            uint8_t oldSREG = SREG;
            cli();
            *reg &= ~bit;
```

```
            SREG = oldSREG;
        }
        else
        {
            uint8_t oldSREG = SREG;
            cli();
            *reg |= bit;
            SREG = oldSREG;
        }
}
```

可以在开发环境中的下列实例程序中找到pinMode函数的应用：

ADXL3xx.pde、AnalogInput.pde、Blink.pde、BlinkWithoutDelay.pde、Button.pde、Calibration.pde、Debounce.pde、Dimmer.pde、Knock.pde、Loop.pde、Melody.pde、Memsic2125.pde、PhysicalPixel.pde、Ping.pde

3.1.2 digitalWrite(pin,value)

digitalWrite函数也是在Blink程序中见到过的，它的作用是设置引脚的输出的电压为高电平或低电平。该函数也是一个无返回值的函数，函数有两个参数pin和value，pin参数表示所要设置的引脚，value参数表示输出的电压——HIGH（高电平）或LOW（低电平）。

注意：在使用digitalWrite(pin, value)函数设置引脚之前，需要将引脚设置为OUTPUT模式。

digitalWrite(pin, value)函数原型同样也可以在wiring_digital.c文件中找到，函数原型如下：

```
void digitalWrite(uint8_t pin, uint8_t val)
{
    uint8_t timer = digitalPinToTimer(pin);
    uint8_t bit = digitalPinToBitMask(pin);
    uint8_t port = digitalPinToPort(pin);
    volatile uint8_t *out;

    if (port == NOT_A_PIN) return;

    // If the pin that support PWM output, we need to turn it off
    // before doing a digital write.
    if (timer != NOT_ON_TIMER) turnOffPWM(timer);

    out = portOutputRegister(port);

    if (val == LOW)
```

```
        {
                uint8_t oldSREG = SREG;
                cli();
                *out &= ~bit;
                SREG = oldSREG;
        }
        else
        {
                uint8_t oldSREG = SREG;
                cli();
                *out |= bit;
                SREG = oldSREG;
        }
}
```

可以在开发环境的下列实例程序中找到digitalWrite函数的应用：

ADXL3xx.pde、AnalogInput.pde、Blink.pde、BlinkWithoutDelay.pde、Button.pde、Calibration.pde、Debounce.pde、Knock.pde、Loop.pde、Melody.pde、PhysicalPixel.pde、Ping.pde

3.1.3 digitalRead(pin)

digitalRead函数用在引脚为输入的情况下，可以获取引脚的电压情况——HIGH（高电平）或LOW（低电平），参数pin表示所要获取电压值的引脚，该函数返回值为int型，表示引脚的电压情况。函数原型如下：

```
int digitalRead(uint8_t pin)
{
        uint8_t timer = digitalPinToTimer(pin);
        uint8_t bit = digitalPinToBitMask(pin);
        uint8_t port = digitalPinToPort(pin);

        if (port == NOT_A_PIN) return LOW;

        // If the pin that support PWM output, we need to turn it off
        // before getting a digital reading.
        if (timer != NOT_ON_TIMER) turnOffPWM(timer);

        if (*portInputRegister(port) & bit) return HIGH;
        return LOW;
}
```

注意：如果引脚没有链接到任何地方，那么将随机返回HIGH或LOW。

可以在开发环境的下列实例程序中找到digitalRead函数的应用：

Button.pde、Debounce.pde

3.2 模拟I/O

3.2.1 analogReference(type)

analogReference函数的作用是配置模拟引脚的参考电压。在嵌入式应用中引脚获取模拟电压值之后，将根据参考电压将模拟值转换到0～1023。该函数为无返回值函数，参数为type类型，有3种类型（DEFAULT/INTERNAL/EXTERNAL），具体含义如下：

DEFAULT：默认值，参考电压为5V。

INTERNAL：低电压模式，使用片内基准电压源。

EXTERNAL：扩展模式，通过AREF引脚获取参考电压，AREF引脚位置见图3.2。

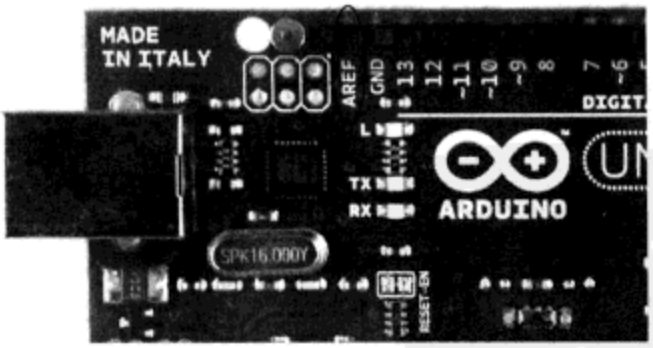


图3.2 AREF引脚位置

注意：如果在AREF引脚加载外部参考电压，需要使用一个5KΩ的上拉电阻，这会避免由于设置不当造成控制芯片的损坏。

3.2.2 analogRead(pin)

analogRead函数用于读取引脚的模拟量电压值，每读一次需要花100μs的时间。参数pin表示所要获取模拟量电压值的引脚，该函数返回值为int型，表示引脚的模拟量电压值，范围在0～1023。函数原型可在wiring_analog.c文件中查看，如下：

```
int analogRead(uint8_t pin)
{
        uint8_t low, high;
```

```
    // set the analog reference (high two bits of ADMUX) and select
    // the channel (low 4 bits).  this also sets ADLAR (left-adjust
    // result) to 0 (the default).
    ADMUX = (analog_reference << 6) | (pin & 0x07);

    // start the conversion
    sbi(ADCSRA, ADSC);

    // ADSC is cleared when the conversion finishes
    while (bit_is_set(ADCSRA, ADSC));

    // we have to read ADCL first; doing so locks both ADCL
    // and ADCH until ADCH is read.  reading ADCL second would
    // cause the results of each conversion to be discarded,
    // as ADCL and ADCH would be locked when it completed.
    low = ADCL;
    high = ADCH;

    // combine the two bytes
    return (high << 8) | low;
}
```

注意：函数的参数pin范围是0～5，表示6个模拟量I/O口中的一个。

可以在开发环境中的下列实例程序中找到analogRead函数的应用：

ADXL3xx.pde、AnalogInput.pde、Calibration.pde、Graph.pde、Knock.pde、Smoothing.pde、VirtualColorMixer.pde

3.2.3 analogWrite(pin, value)

analogWrite函数通过PWM的方式在引脚上输出一个模拟量，较多的应用在LED亮度控制、电机转速控制等方面。

PWM（Pulse Width Modulation，脉冲宽度调制）方式是通过对一系列脉冲的宽度进行调制，来等效地获得所需要的波形或电压。脉冲宽度调制是一种模拟控制方式，其根据相应载荷的变化调制晶体管栅极或基极的偏置，来实现开关稳压电源输出晶体管或晶体管导通时间的改变，这种方式能使电源的输出电压在工作条件变化时保持恒定，是利用微处理器的数字输出来对模拟电路进行控制的一种非常有效的技术。图3.3是一个简单的PWM波示意图。

其中，VCC是高电平值，T是PWM波的周期，D是高电平的宽度，D/T是PWM波的占空比，当上述PWM波通过一个低通滤波器后，波形中高频的部分被滤掉得到所需的波形，其平均电压为VCC × D/T。因此，可通过调节D的大小来改变占空比，产生不同的平均电压；

同样，调节PWM波的周期T也可以改变占空比，从而得到不同的平均电压值。

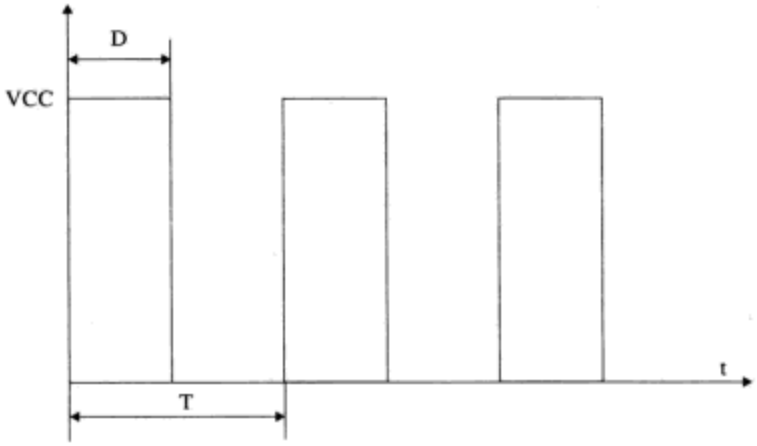


图3.3 PWM波示意图

在Arduino中执行该操作后，应该等待一定时间后才能对该引脚进行下一次操作。Arduino中的PWM的频率大约为490Hz。该函数支持以下引脚：3、5、6、9、10、11。在Arduino控制板上引脚号旁边标注～的就是可用作PWM的引脚，如图3.4所示。

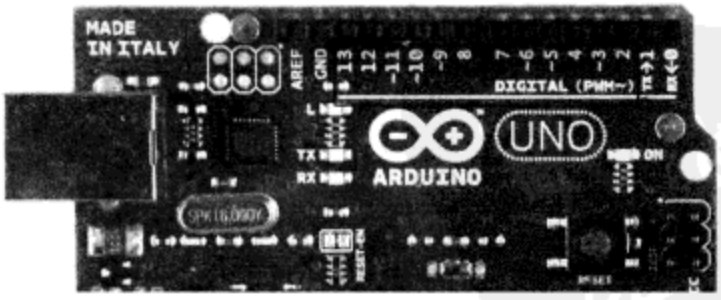


图3.4 Arduino板上PWM引脚的标识

analogWrite函数为无返回值函数，有两个参数pin和value，参数pin表示所要设置的引脚，只能选择函数支持的引脚；参数value表示PWM输出的占空比，范围在0～255的区间，对应的占空比为0%～100%，函数原型如下：

```
void analogWrite(uint8_t pin, int val)
{
      if (digitalPinToTimer(pin) == TIMER1A)
      {
            // connect pwm to pin on timer 1, channel A
            sbi(TCCR1A, COM1A1);
            // set pwm duty
            OCR1A = val;
```

```
    }
    else if (digitalPinToTimer(pin) == TIMER1B)
    {
          // connect pwm to pin on timer 1, channel B
          sbi(TCCR1A, COM1B1);
          // set pwm duty
          OCR1B = val;
    }
    else if (digitalPinToTimer(pin) == TIMER0A)
    {
          if (val == 0)
          {
                digitalWrite(pin, LOW);
          }
          else
          {
                // connect pwm to pin on timer 0, channel A
                sbi(TCCR0A, COM0A1);
                // set pwm duty
                OCR0A = val;
          }
    }
    else if (digitalPinToTimer(pin) == TIMER0B)
    {
          if (val == 0)
          {
                digitalWrite(pin, LOW);
          }
          else
          {
                // connect pwm to pin on timer 0, channel B
                sbi(TCCR0A, COM0B1);
                // set pwm duty
                OCR0B = val;
          }
    }
    else if (digitalPinToTimer(pin) == TIMER2A)
    {
          // connect pwm to pin on timer 2, channel A
          sbi(TCCR2A, COM2A1);
          // set pwm duty
          OCR2A = val;
    }
```

```
        else if (digitalPinToTimer(pin) == TIMER2B)
        {
                // connect pwm to pin on timer 2, channel B
                sbi(TCCR2A, COM2B1);
                // set pwm duty
                OCR2B = val;
        }
        else if (val < 128)
                digitalWrite(pin, LOW);
        else
                digitalWrite(pin, HIGH);
}
```

可以在开发环境的下列实例程序中找到analogWrite函数的应用：

Calibration.pde、Dimmer.pde、Fading.pde

3.3 高级I/O

3.3.1 shiftOut(dataPin,clockPin,bitOrder,val)

shiftOut函数能够将数据通过串行的方式在引脚上输出，相当于一般意义上的同步串行通信，这是控制器与控制器、控制器与传感器之间常用的一种通信方式。

shiftOut函数无返回值，有4个参数：dataPin、clockPin、bitOrder、val，具体说明如下：

dataPin：数据输出引脚，数据的每一位将逐次输出。引脚模式需要设置成输出。

clockPin：时钟输出引脚，为数据输出提供时钟，引脚模式需要设置成输出。

bitOrder：数据位移顺序选择位，该参数为byte类型，有两种类型可选择，分别是高位先入MSBFIRST和低位先入LSBFIRST。

val：所要输出的数据值。

函数原型在wiring_shift.c文件中，如下：

```
void shiftOut(uint8_t dataPin, uint8_t clockPin, uint8_t bitOrder, uint8_t val)
{
        uint8_t i;

        for (i = 0; i < 8; i++)
        {
                if (bitOrder == LSBFIRST)
                        digitalWrite(dataPin, !!(val & (1 << i)));
                else
                        digitalWrite(dataPin, !!(val & (1 << (7 - i))));
```

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

❑ 另外还有shiftIn函数用于通过串行的方式从引脚上读入数据，其函数定义如下：

```
uint8_t shiftIn(uint8_t dataPin, uint8_t clockPin, uint8_t bitOrder)
{
        uint8_t value = 0;
        uint8_t i;

        for (i = 0; i < 8; ++i)
        {
                digitalWrite(clockPin, HIGH);
                if (bitOrder == LSBFIRST)
                        value |= digitalRead(dataPin) << i;
                else
                        value |= digitalRead(dataPin) << (7 - i);
                digitalWrite(clockPin, LOW);
        }
        return value;
}
```

3.3.2 pulseIn(pin,state,timeout)

pulseIn函数用于读取引脚脉冲的时间长度，脉冲可以是HIGH或LOW。如果是HIGH，函数将先等引脚变为高电平，然后开始计时，一直到变为低电平为止。返回脉冲持续的时间长短，单位为ms。如果超时还没有读到的话，将返回0。

pulseIn函数返回值类型为无符号长整型（unsigned long），3个参数分别表示脉冲输入的引脚、脉冲响应的状态（高脉冲或低脉冲）和超时时间。函数原型在wiring_pulse.c中，如下：

```
unsigned long pulseIn(uint8_t pin, uint8_t state, unsigned long timeout)
{
        uint8_t bit = digitalPinToBitMask(pin);
        uint8_t port = digitalPinToPort(pin);
        uint8_t stateMask = (state ? bit : 0);
        unsigned long width = 0;
        // keep initialization out of time critical area

        unsigned long numloops = 0;
        unsigned long maxloops = microsecondsToClockCycles(timeout) / 16;

        // wait for any previous pulse to end
```

```
        while ((*portInputRegister(port) & bit) == stateMask)
                if (numloops++ == maxloops)
                return 0;
        // wait for the pulse to start
        while ((*portInputRegister(port) & bit) != stateMask)
                if (numloops++ == maxloops)
                return 0;
        // wait for the pulse to stop
        while ((*portInputRegister(port) & bit) == stateMask)
                width++;

        return clockCyclesToMicroseconds(width * 10 + 16);
}
```

可以在开发环境的下列实例程序中找到pulseIn函数的应用：

Memsic2125.pde、Ping.pde

3.4 时间函数

3.4.1 millis( )

应用millis函数可获取机器运行的时间长度，单位ms。系统最长的记录时间为9小时22分，如果超出时间将从0开始。函数返回值为unsigned long型，无参数。函数原型如下：

```
unsigned long millis()
{
        unsigned long m;
        uint8_t oldSREG = SREG;

        cli();
        m = timer0_millis;
        SREG = oldSREG;

        return m;
}
```

注意：函数返回值为unsigned long型，如果用int型保存时间将得到错误结果。

可以在开发环境的下列实例程序中找到millis函数的应用：

BlinkWithoutDelay.pde、Calibration.pde、Debounce.pde

3.4.2 delay(ms)

delay函数是一个延时函数，在Blink程序中用到过，参数表示延时时长，单位是ms。函数无返回值，原型如下：

```
void delay(unsigned long ms)
{
      uint16_t start = (uint16_t)micros();

      while (ms > 0)
      {
            if (((uint16_t)micros() - start) >= 1000)
            {
                  ms--;
                  start += 1000;
            }
      }
}
```

可以在开发环境的下列实例程序中找到delay函数的应用：

ADXL3xx.pde、AnalogInput.pde、Blink.pde、Fading.pde、Graph.pde、Knock.pde、Loop.pde、Melody.pde、Memsic2125.pde、Ping.pde

3.4.3 delayMicroseconds(us)

delayMicroseconds函数同样是延时函数，所不同的是其参数单位是μs（1ms=1000μs）。函数原型如下：

```
void delayMicroseconds(unsigned int us)
{
      // for a one-microsecond delay, simply return.
      //the overhead of the function call yields a delay of
      //approximately 1 1/8 us.
      if (--us == 0)
      return;

      // the following loop takes a quarter of a microsecond (4 cycles)
      // per iteration, so execute it four times for each
      //microsecond of delay requested.
      us <<= 2;

      // account for the time taken in the preceeding commands.
      us -= 2;
```

```
    // busy wait
    __asm__ __volatile__ (
            "1: sbiw %0,1" "\n\t" // 2 cycles
            "brne 1b" : "=w" (us) : "0" (us) // 2 cycles
    );
}
```

可以在开发环境中的下列实例程序中找到delayMicroseconds函数的应用：

Melody.pde、Ping.pde

3.5 数学库

3.5.1 min(x,y)

min(x, y)函数的作用是返回x、y两者中较小的，函数原型为：

```
#define min(a,b)    ((a)<(b)?(a):(b))
```

3.5.2 max(x,y)

max(x, y)函数的作用是返回x、y两者中较大的，函数原型为：

```
#define max(a,b)    ((a)>(b)?(a):(b))
```

3.5.3 abs(x)

abs(x)函数的作用是获取x的绝对值，函数原型为：

```
#define abs(x)    ((x)>0?(x):-(x))
```

3.5.4 constrain(amt,low,high)

constrain(amt，low，high)函数的工作过程是，如果值amt小于low，则返回low；如果amt大于high，则返回high；否则，返回amt。该函数一般可以用于将值归一化到某个区间内。函数原型为：

```
#define constrain(amt,low,high)
                ((amt)<(low)?(low):((amt)>(high)?(high):(amt)))
```

3.5.5 map(x,in_min,in_max,out_min,out_max)

map(x, in_min, in_max, out_min, out_max)函数的作用是将 [in_min, in_max]范围内的x等比映射到[out_min, out_max]范围内。函数返回值为long型，原型为：

```
long map(long x, long in_min, long in_max, long out_min, long out_max)
{
    return (x - in_min)*(out_max - out_min) / (in_max - in_min)+out_min;
}
```

3.5.6 三角函数

三角函数包括sin(rad)、cos(rad)、tan(rad)，分别得到rad的正弦值、余弦值和正切值。返回值都为double型。

3.6 随机数

3.6.1 randomSeed(seed)

randomSeed( )函数用来设置随机数种子，随机种子的设置对产生的随机序列有影响。函数无返回值，原型如下：

```
void randomSeed(unsigned int seed)
{
      if (seed != 0)
      {
            srandom(seed);
      }
}
```

3.6.2 random(howsmall,howbig)

应用random函数可生成一个随机数，两个参数howsmall和howbig决定了随机数的范围，函数的参数及返回值均为long型，原型如下：

```
long random(long howsmall, long howbig)
{
      if (howsmall >= howbig)
      {
            return howsmall;
      }
      long diff = howbig - howsmall;
      return random(diff) + howsmall;
}
```

3.7 位操作

位操作用于设置或读取字节中某一位或几位，包括bitRead()、bitSet()、bitClear()等，具体定义及功能可以参考文件wiring.h。

```
#define lowByte(w)  ((uint8_t) ((w) & 0xff))                    //低字节
#define highByte(w)  ((uint8_t) ((w) >> 8))                     //高字节

//读bit位的值，即保留bit位，其他位均清零
#define bitRead(value, bit)  (((value) >> (bit)) & 0x01)

//置bit位的值，即bit位置1
#define bitSet(value, bit)  ((value) |= (1UL << (bit)))

//清除bit位，即bit位置0
#define bitClear(value, bit)  ((value) &= ~(1UL << (bit)))

//写bit位的值，1或者0
#define bitWrite(value, bit, bitvalue) (bitvalue ? bitSet(value, bit) : bitClear(value, bit))
```

3.8 中断函数

3.8.1 interrupts( )和noInterrupts( )

interrupts和noInterrupts函数在Arduino中负责打开和关闭总中断，函数无返回值，无参数，同样可在文件wiring.h中查看到函数原型，如下：

```
#define interrupts() sei()
#define noInterrupts() cli()
```

3.8.2 attachInterrupt(interrupt,function,mode)

attachInterrupt函数用于设置外部中断，函数有3个参数：interrupt、function和mode，分别表示中断源、中断处理函数、触发模式。参数中断源可选值0或1，在Arduino中一般对应2号和3号数字引脚；参数中断处理函数用来指定中断的处理函数，参数值为函数的指针，触发模式有4种类型：LOW（低电平触发）、CHANGE（变化时触发）、RISING（低电平变为高电平触发）、FALLING（高电平变为低电平触发）。

下面的例子是通过外部引脚触发中断函数。然后控制13号引脚的LED的闪烁。

```
int pin = 13;
volatile int state = LOW;
```

```
void setup()
{
    pinMode(pin, OUTPUT);
    attachInterrupt(0, blink, CHANGE);     //中断源：1
                                           //中断处理函数：blink ()
                                           //触发模式：CHANGE（变化时触发）
}

void loop()
{
    digitalWrite(pin, state);
}

//中断处理函数
void blink()
{
    state = !state;
}
```

在使用attachInterrupt函数时要注意以下几点：

- 在中断函数中delay函数不能使用。
- 使用millis函数始终返回进入中断前的值。
- 读取串口数据的话，可能会丢失。
- 中断函数中使用的变量需要定义为volatile型。

attachInterrupt函数的函数原型可在文件WInterrupts.c中找到，如下所示：

```
void attachInterrupt(uint8_t interruptNum, void (*userFunc)(void), int mode)
{
    if(interruptNum < EXTERNAL_NUM_INTERRUPTS)
    {
        intFunc[interruptNum] = userFunc;

        switch (interruptNum)
        {
            case 0:
                EICRA = (EICRA & ~((1 << ISC00)
                    |(1 << ISC01)))|(mode << ISC00);
                EIMSK |= (1 << INT0);
                break;
            case 1:
                EICRA = (EICRA & ~((1 << ISC10)
                    |(1 << ISC11)))|(mode << ISC10);
                EIMSK |= (1 << INT1);
```

```
                                    break;
                }
        }
}
```

另外，还有detachInterrupt函数用于取消中断，参数interrupt表示所要取消的中断源，函数的定义如下：

```
void detachInterrupt(uint8_t interruptNum)
{
        if(interruptNum < EXTERNAL_NUM_INTERRUPTS)
        {
                switch (interruptNum)
                {
                        case 0:
                                EIMSK &= ~(1 << INT0);
                                break;
                        case 1:
                                EIMSK &= ~(1 << INT1);
                                break;
                }
                intFunc[interruptNum] = 0;
        }
}
```

3.9 串口通信

Arduino中串口通信是通过HardwareSerial类来实现的，在头文件HardwareSerial.h中定义了一个HardwareSerial类的对象Serial，直接使用类的成员函数就可简单地实现串口通信。对象和类的概念与应用可参阅6.1节内容。下面以一个串口调光器的程序为例介绍HardwareSerial类几个较常用的公有成员函数，程序清单如下：

```
/*
  Dimmer (调光器)
  通过计算机发送数据控制LED灯的亮度，单字节数据发送，数据范围0～255
  使用具有pwm功能的9号引脚

 created 2006
 by David A. Mellis
 modified 14 Apr 2009
 by Tom Igoe and Scott Fitzgerald

 This example code is in the public domain.
```

```
 http://www.arduino.cc/en/Tutorial/Dimmer
 */

const int ledPin = 9;       // the pin that the LED is attached to

void setup()
{
      // 设置串口波特率:
      Serial.begin(9600);                                          //1
      // 设置LED控制引脚:
      pinMode(ledPin, OUTPUT);
}

void loop()
{
      byte brightness;
      // 查询串口是否收到数据:
      if (Serial.available())                                      //2
      {
                // 获取数据
                brightness = Serial.read();                        //3
                //控制LED亮度
                analogWrite(ledPin, brightness);
      }
}
```

串口通信相关语句分析解释。

1. Serial.begin(9600);

该语句的功能是设置串口通信波特率为9600bps，其函数原型如下：

```
void HardwareSerial::begin(long baud)
{
      uint16_t baud_setting;
      bool use_u2x;

      // U2X mode is needed for baud rates higher than (CPU Hz / 16)
      if (baud > F_CPU / 16)
      {
                use_u2x = true;
      }
      else
      {
                // figure out if U2X mode would allow for a better connection
```

```
                // calculate the percent difference between the baud-rate specified and
                // the real baud rate for both U2X and non-U2X mode
                uint8_t nonu2x_baud_error = abs((int)(255-
                          ((F_CPU/(16*(((F_CPU/8/baud-1)/2)+1))*255)/baud)));
                uint8_t u2x_baud_error = abs((int)(255-
                          ((F_CPU/(8*(((F_CPU/4/baud-1)/2)+1))*255)/baud)));

                // prefer non-U2X mode because it handles clock skew better
                use_u2x = (nonu2x_baud_error > u2x_baud_error);
        }

        if (use_u2x)
        {
                *_ucsra = 1 << _u2x;
                baud_setting = (F_CPU / 4 / baud - 1) / 2;
        }
        else
        {
                *_ucsra = 0;
                baud_setting = (F_CPU / 8 / baud - 1) / 2;
        }

        // assign the baud_setting, a.k.a. ubbr (USART Baud Rate Register)
        *_ubrrh = baud_setting >> 8;
        *_ubrrl = baud_setting;

        sbi(*_ucsrb, _rxen);
        sbi(*_ucsrb, _txen);
        sbi(*_ucsrb, _rxcie);
}
```

2. if (Serial.available())

该语句用来判断Arduino串口是否收到数据，函数Serial.available( )返回值为int型，不带参数。函数原型如下：

```
int HardwareSerial::available(void)
{
  return (RX_BUFFER_SIZE + _rx_buffer->head - _rx_buffer->tail) % RX_BUFFER_SIZE;
}
```

3. brightness = Serial.read();

该语句的功能是将串口数据读入到变量brightness中，函数Serial.read( )也不带参数，返

回值为串口数据，int型。函数原型如下：

```
int HardwareSerial::read(void)
{
        // if the head isn't ahead of the tail, we don't have any characters
        if (_rx_buffer->head == _rx_buffer->tail)
        {
                return -1;
        }
        else
        {
                unsigned char c = _rx_buffer->buffer[_rx_buffer->tail];
                _rx_buffer->tail=(_rx_buffer->tail + 1) % RX_BUFFER_SIZE;
                return c;
        }
}
```

3.10 SPI接口

3.10.1 SPI接口概述

SPI（Serial Peripheral Interface）是由摩托罗拉公司提出的一种同步串行外设接口总线，它可以使MCU与各种外围设备以串行方式进行通信以及交换信息，总线采用3根或4根数据线进行数据传输，常用的是4根线，即两条控制线（芯片选择CS和时钟SCLK）以及两条数据信号线SDI和SDO。

SPI是一种高速、全双工、同步的通信总线。在摩托罗拉公司的SPI技术规范中，数据信号线SDI称为MISO（Master-In-Slave-Out，主入从出），数据信号线SDO称为MOSI（Master-Out-Slave-In，主出从入），控制信号线CS称为SS（Slave-Select，从属选择），将SCLK称为SCK（Serial-Clock，串行时钟）。在SPI通信中，数据是同步进行发送和接收的。数据传输的时钟基于来自主处理器产生的时钟脉冲，摩托罗拉公司没有定义任何通用的SPI时钟规范。

3.10.2 SPI接口数据传输

SPI是以主从方式工作的，其允许一个主设备和多个从设备进行通信，主设备通过不同的SS信号线选择不同的从设备进行通信。其典型应用示意图如图3.5所示。

当主设备选中某一个从设备后，MISO和MOSI用于串行数据的接收和发送，SCK提供串行通信时钟，上升沿发送，下降沿接收。在实际应用中，未选中的从设备的MOSI信号线需处于高阻状态，否则会影响主设备与选中从设备间的正常通信。

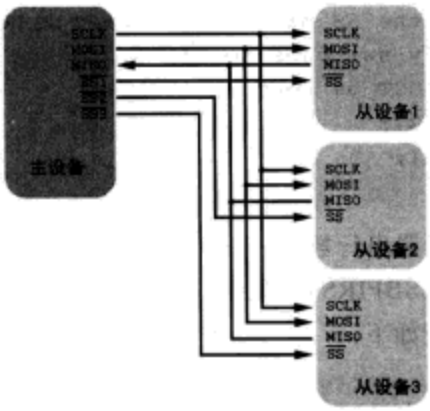


图3.5 SPI总线应用示意图

3.10.3 SPI类及其成员函数

Arduino中的SPI通信是通过SPIClass类来实现的，使用SPIClass类能够方便地将Arduino作为主设备与其他从设备通信。SPIClass类提供了6个成员函数供使用者调用，如下：

- begin()
- setBitOrder ()
- setClockDivider ()
- setDataMode ()
- transfer ()
- end ()

begin函数用于初始化SPI总线，函数原型如下：

```
void SPIClass::begin()
{
      // Set direction register for SCK and MOSI pin.
      // MISO pin automatically overrides to INPUT.
      // When the SS pin is set as OUTPUT, it can be used as
      // a general purpose output port (it doesn't influence
      // SPI operations).

      pinMode(SCK, OUTPUT);
      pinMode(MOSI, OUTPUT);
      pinMode(SS, OUTPUT);

      digitalWrite(SCK, LOW);
      digitalWrite(MOSI, LOW);
      digitalWrite(SS, HIGH);
```

```
        // Warning: if the SS pin ever becomes a LOW INPUT then SPI
        // automatically switches to Slave, so the data direction of
        // the SS pin MUST be kept as OUTPUT.
        SPCR |= _BV(MSTR);
        SPCR |= _BV(SPE);
}
```

setBitOrder的作用是在设置串行数据传输时是先传输低位还是先传输高位，函数有一个type类型的参数bitOrder，有LSBFIRST（最低位在前）和MSBFIRST（最高位在前）两种类型可选。函数无返回值，原型如下：

```
void SPIClass::setBitOrder(uint8_t bitOrder)
{
        if(bitOrder == LSBFIRST)
        {
                SPCR |= _BV(DORD);
        }
        else
        {
                SPCR &= ~(_BV(DORD));
        }
}
```

setClockDivider 函数的作用是设置SPI串行通信的时钟，通信时钟是由系统时钟分频而得到，分频值可选2、4、8、16、32、64及128，有一个type类型的参数rate，有7种类型，对应7个分频值分别为SPI_CLOCK_DIV2、SPI_CLOCK_DIV4、SPI_CLOCK_DIV8、SPI_CLOCK_DIV16、SPI_CLOCK_DIV32、SPI_CLOCK_DIV64和SPI_CLOCK_DIV128。函数默认参数设置是SPI_CLOCK_DIV4，设置SPI串行通信时钟为系统时钟的1/4。函数原型如下：

```
void SPIClass::setClockDivider(uint8_t rate)
{
        SPCR = (SPCR & ~SPI_CLOCK_MASK) | (rate & SPI_CLOCK_MASK);
        SPSR = (SPSR & ~SPI_2XCLOCK_MASK) | (rate & SPI_2XCLOCK_MASK);
}
```

setDataMode函数的作用是设置SPI数据模式，由于在SPI通信中没有定义任何通用的时钟规范，所以在具体应用中有的在上升沿采样，有的在下降沿采样，由此SPI存在4种数据模式，如表3.1所示。

表3.1 SPI通信数据模式

模 式	说 明		
模式0	上升沿采样	下降沿置位	SCK闲置时为0
模式1	上升沿置位	下降沿采样	
模式2	下降沿采样	上升沿置位	SCK闲置时为1
模式3	下降沿置位	上升沿采样	

setDataMode函数的type类型的参数mode有4种类型可选，分别是SPI_MODE0、SPI_MODE1、SPI_MODE2和SPI_MODE3。函数原型如下：

```
void SPIClass::setDataMode(uint8_t mode)
{
      SPCR = (SPCR & ~SPI_MODE_MASK) | mode;
}
```

transfer函数用来传输一个数据，由于SPI是一种全双工、同步的通信总线。所以传输一个数据实际上会发送一个数据，同时接收一个数据。函数的参数为发送的数据值，返回的参数为接收的数据值。函数原型如下：

```
byte SPIClass::transfer(byte _data)
{
      SPDR = _data;
      while (!(SPSR & _BV(SPIF)));
      return SPDR;
}
```

end函数停止SPI总线的使用，函数原型如下：

```
void SPIClass::end()
{
      SPCR &= ~_BV(SPE);
}
```

第4章 Arduino硬件平台

通过前几章的介绍，大家对Arduino的软件资源和环境应该有了一定的了解，但Arduino毕竟是一个硬件项目，要想真正用好Arduino，还需要一些必要的硬件知识。

本章通过原理分析、资源使用等方面介绍Arduino的硬件平台。

4.1 Arduino的原理图

Arduino Duemilanove的硬件布局如图4.1所示。“Duemilanove”在意大利语中是2009的意思，名字就取自它的发布年份。可以在网站：http://www.arduino.cc上获得该控制板的原理图，如图4.2所示。Arduino Uno与Arduino Duemilanove的硬件布局相似，所不同的是Arduino Uno使用的不是FTDI USB-to-serial串行驱动器芯片，而是采用ATmega8U2芯片进行USB到串行数据的转换。Uno在西班牙语和意大利语中的意思为“一”，代表着Arduino 1.0，其原理图如图4.3所示。

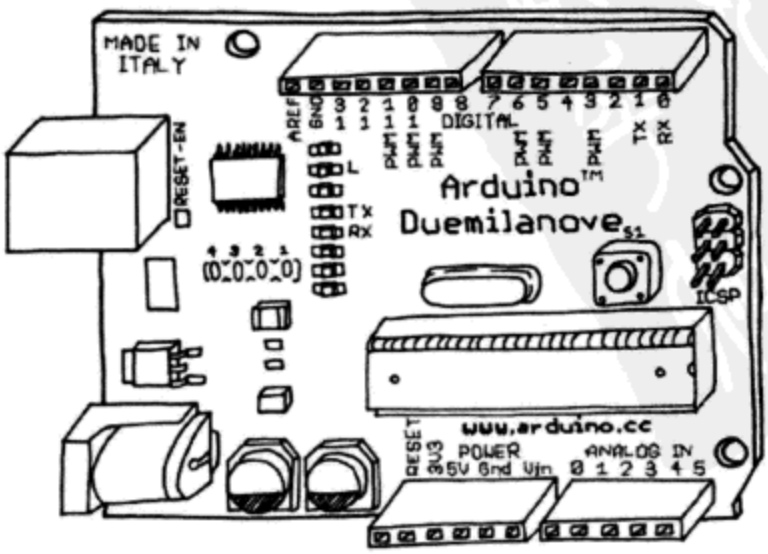


图4.1 Arduino Duemilanove的硬件布局

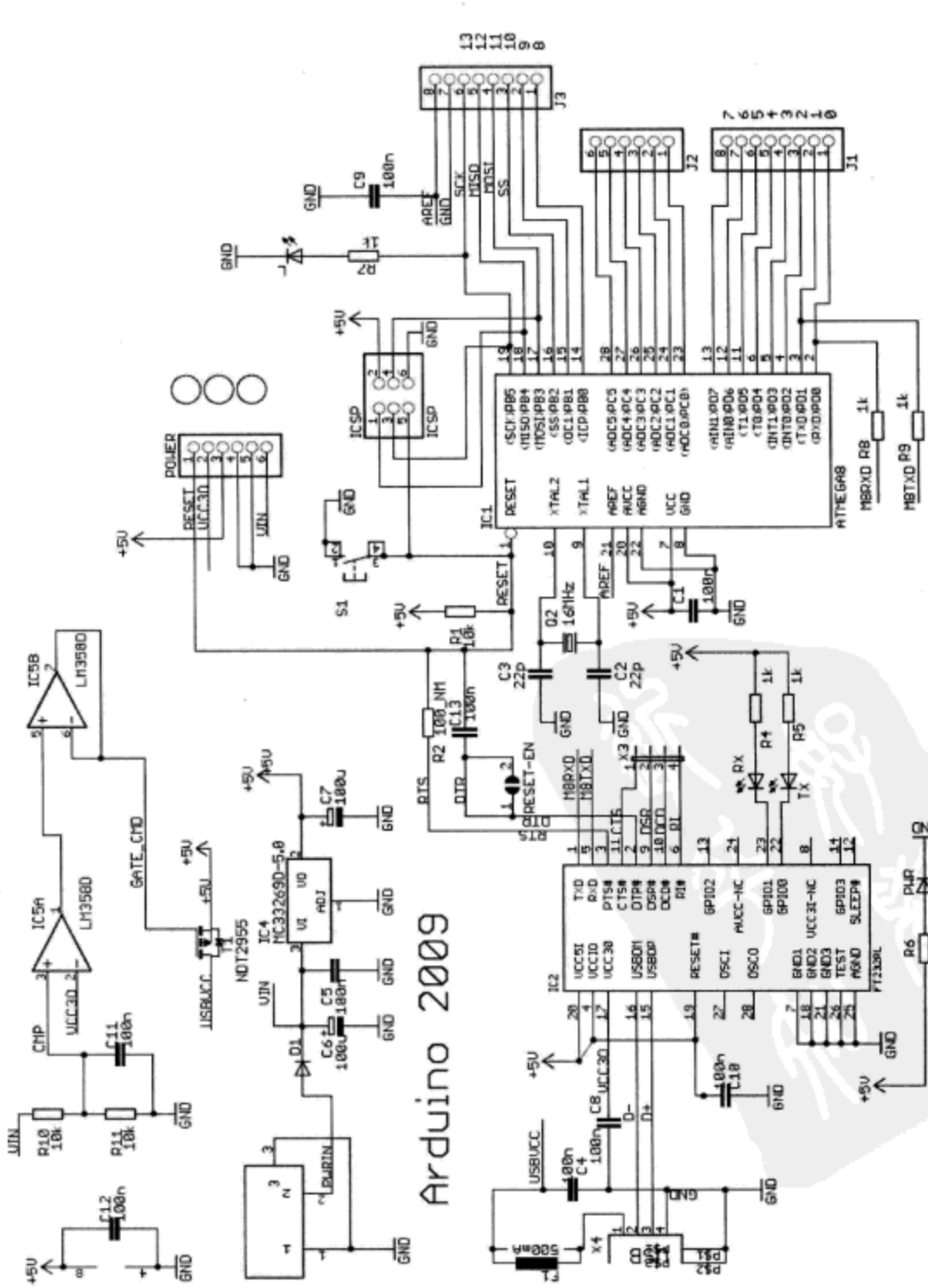


图4.2 Arduino Duemilanove原理图

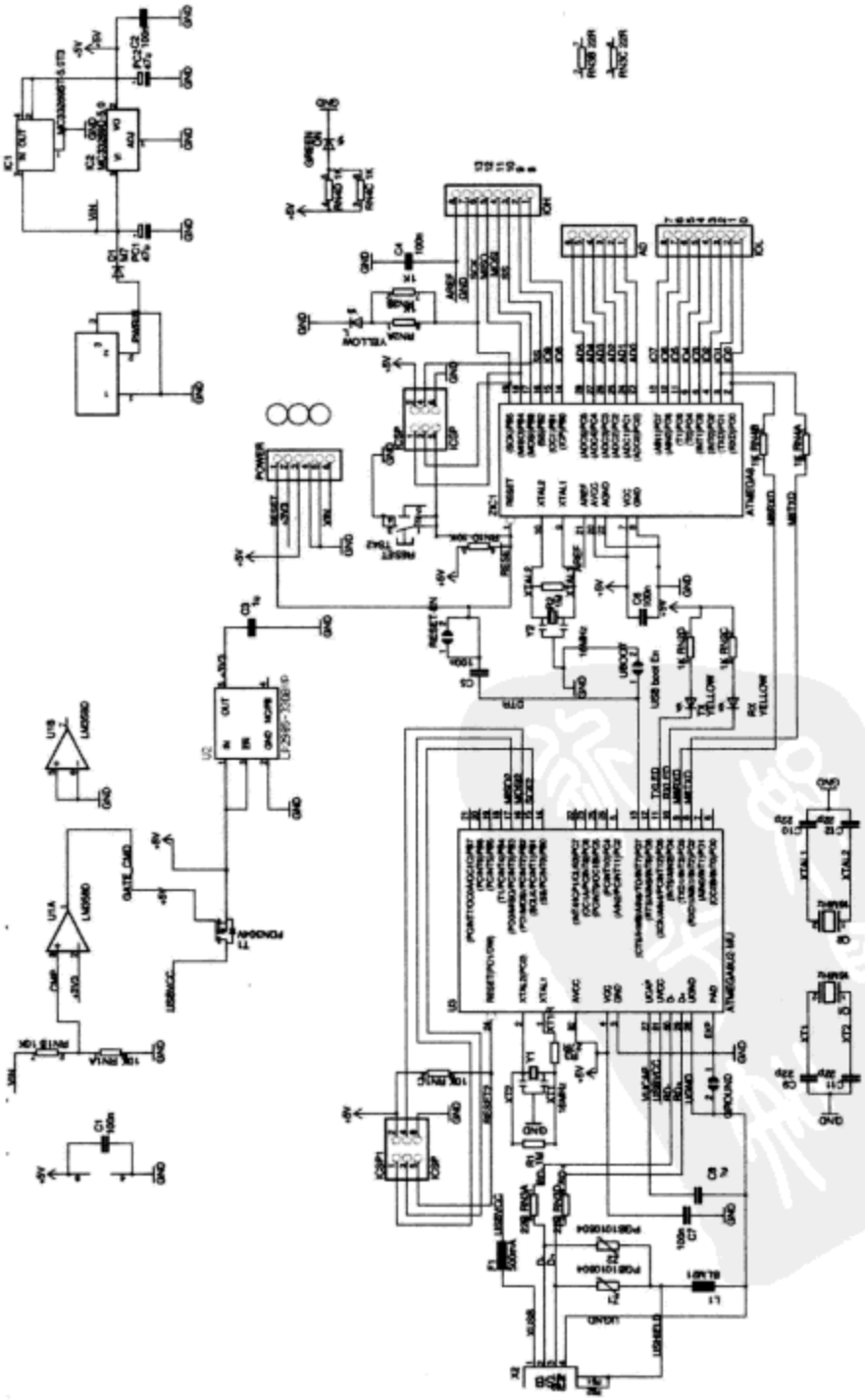


图4.3 Arduino Uno原理图

由原理图可以发现，Arduino就是一个带USB转串口芯片的AVR板，其核心就是AVR单片机。AVR单片机具有以下优点：

- 采用RISC（精简指令集）指令系统，32个通用工作寄存器，避免了传统8051单片机指令系统的指令长度不够长、指令数多、CPU利用率低和执行速度慢的缺点。
- 采用哈佛结构的流水线技术，提高指令执行速度。
- 除支持ISP在线编程外，还支持IAP在应用编程。
- 片内集成了可擦写1000次的FLASH程序存储器，同时还有大容量的可擦写100 000次的EEPROM。
- 集成了丰富的外设，包括IIC、SPI、WTD、ADC、PWM和片内振荡器等。
- 具有较宽的工作电压范围，工作电压在1.8～6V，电源抗干扰能力强。
- I/O驱动能力强，可直接驱动LED。
- USART使用独特的波特率发生器，波特率可达到576Kbit/s。

Arduino的I/O引脚直接连接单片机的I/O引脚，利用AVR单片机引脚本身具有的功能实现PWM、ADC、IIC、驱动LED等应用。AVR单片机硬件连接16MHz晶振采用外部振荡方式工作。

Arduino的编程利用AVR单片机的IAP在应用编程技术，让使用者能够直接通过USB接口将程序上传到芯片中。每一块Arduino板在出厂前都在单片机中烧写了引导程序（bootloader），它能够引导芯片通过串口完成与计算机的通信，实现在应用编程；否则Arduino是无法通过USB口更新程序的。我们会在4.5节详细介绍这一点。

4.2 串行通信口的使用

由原理图可以看出，AVR单片机的串行通信口（RxD和TxD）除了接在USB到串行数据的转换芯片上之外，还作为Arduino的0号和1号引脚，在板上有明确的标识，如图4.4所示。

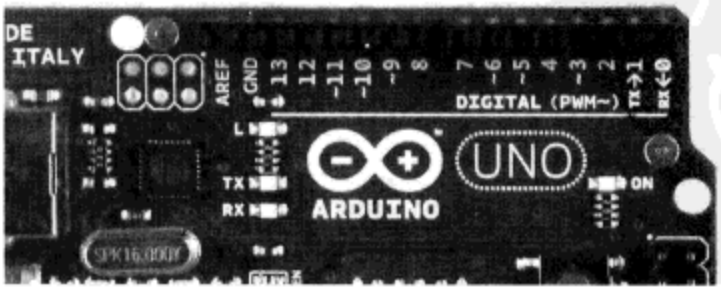


图4.4 Arduino板上的串行通信口

4.2.1 实例功能

本节实例采用两条杜邦线将两块Arduino板的通信口交叉连起来，其中一块Arduino每隔1s发送一个值为0x55的字节，另一块Arduino收到该字节后控制13号引脚的LED转换状态。

4.2.2 硬件电路

串口通信实例电路参考图4.5。

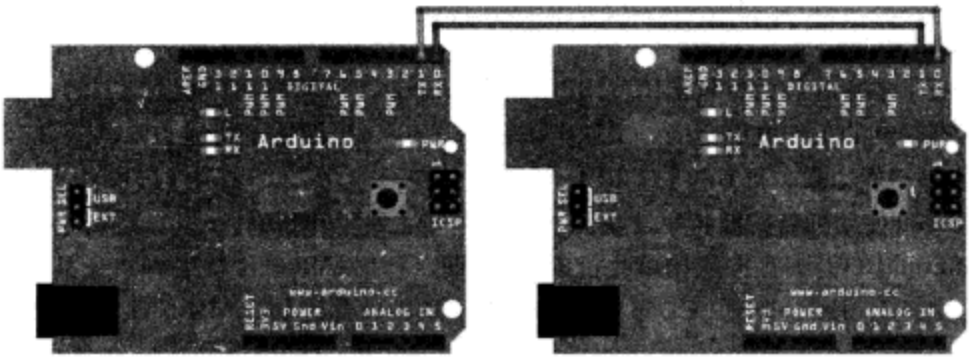


图4.5 串口通信实例电路

4.2.3 程序设计

由于实例用到了两块Arduino，因此程序设计分为发送端和接收端两部分。先来看看发送端的程序，实例要求发送端每隔1s发送一个值为0x55的字节。

```
/*************************************************
  串行通信实例程序——发送端
  每隔1s发送一个值为0x55的字节
  created 2011
  by Nille
  Email: chenille@126.com

  This example code is in the public domain.
 *************************************************/

/*************************************************
                初始化部分——setup()函数
*************************************************/
void setup()
{
      //设置串口波特率为9600bps
      Serial.begin(9600);
```

```
}

/************************************************
                执行部分——loop()函数
************************************************/
void loop()
{
      //延时1s，delay函数见3.4.2
      delay(1000);
      //输出0x55
      Serial.print(0x55, BYTE);
}
```

接收端的程序的任务是收到0x55后转换13号引脚的状态，代码如下：

```
/************************************************
  串行通信实例程序——接收端
  收到0x55后转换13号引脚的状态
  created 2011
  by Nille
  Email: chenille@126.com

  This example code is in the public domain.
 ************************************************/

int ledFlag;      //LED状态
/************************************************
                初始化部分——setup()函数
************************************************/
void setup()
{
      //设置串口波特率为9600bps
      Serial.begin(9600);
      //设置13号引脚为输出
      pinMode(13, OUTPUT);
}

/************************************************
                执行部分——loop()函数
************************************************/
void loop()
{
      byte RXData;
```

```
        if (Serial.available())
        {
                    //获取收到的数据
                    RXData = Serial.read();
                    //判断收到的数据是否是0x55
                    if(RXData == 0x55)
                    {
                              if(ledFlag == 0)//判断LED状态
                              {
                                        ledFlag = 1;
                                        digitalWrite(13,HIGH);
                              }
                              else
                              {
                                        ledFlag = 0;
                                        digitalWrite(13,LOW);
                              }
                    }
        }
}
```

读者在开始学习Arduino串口通信时，可以就用计算机与Arduino进行通信，原理上是一样的。在Arduino开发环境下带有Serial Monitor（串口监视窗）功能，可方便地进行串口通信调试，界面如图4.6所示。

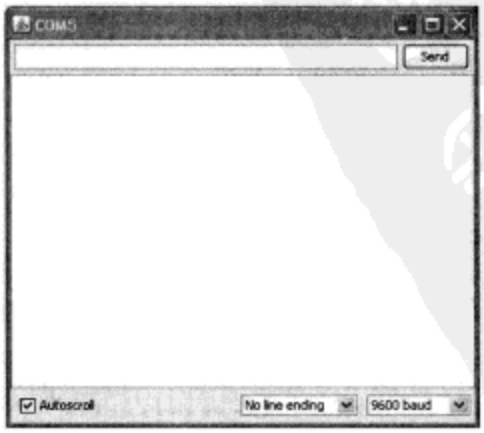


图4.6 Serial Monitor（串口监视窗）界面

4.3 数字I/O口的使用

13号引脚的LED控制就属于数字I/O输出控制，前几章已经应用过多次了，本节介绍数

字I/O输入的使用。

4.3.1 实例功能

本节实例是在2号引脚加一个按键开关，当按下按键时，13号引脚的LED点亮，同时通过串口发送合计的按键次数，当松开按键时，LED熄灭。

4.3.2 硬件电路

将按键的一端接2号引脚，另一端接地。同时为保持2号引脚输入状态稳定，加1个5.1KΩ的上拉电阻在2号引脚。另外不要忘了把Arduino连接在电脑上，实例电路如图4.7所示。

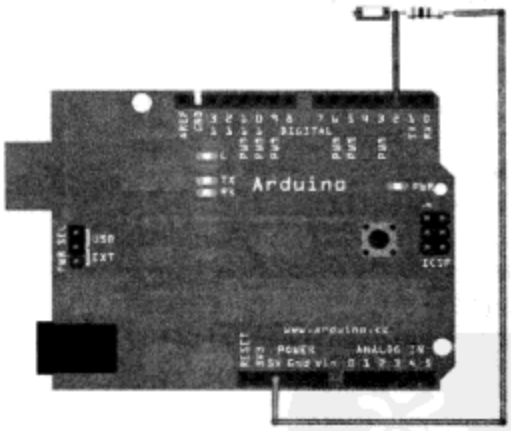


图4.7 数字I/O口实例电路

4.3.3 程序设计

```
/************************************************
  数字I/O口实例程序

  按键在2号引脚
  按下按键时，13号引脚的LED点亮，同时通过串口发送合计的按键次数，当松开按键时，LED熄灭

  created 2011
  by Nille
  Email: chenille@126.com
  This example code is in the public domain.
************************************************/

int keySum = 0;                                    //按键次数
/************************************************
                初始化部分——setup()函数
```

```
**********************************************/
void setup()
{
      //设置串口波特率为9600bps
      Serial.begin(9600);
      //设置13号引脚为输出
      pinMode(13, OUTPUT);
      //设置2号引脚为输入
      pinMode(2, INPUT);
}

/**********************************************
            执行部分——loop()函数
**********************************************/
void loop()
{
      //判断按键是否按下
      if(LOW == digitalRead(2))
      {
            //延时去抖
            delay(50);
            if(LOW == digitalRead(2))
            {
                  //点亮LED
                  digitalWrite(13,HIGH);
                  keySum++;
                  //发送按键次数
                  Serial.print(keySum, DEC);
                  while(1)
                  {
                        //判断是否松开按键
                        if(HIGH == digitalRead(2))
                        {
                              //延时去抖
                              delay(50);
                              if(HIGH == digitalRead(2))
                              break;
                        }
                  }
                  //熄灭LED
                  digitalWrite(13,LOW);
            }
      }
}
```

4.4 模拟I/O口的使用

由原理图可以看出Arduino的6个模拟量引脚实际上是连接的AVR单片机的6个具有ADC功能的管脚，所以Arduino的模拟量输入功能就是通过单片机的ADC接口实现的，该ADC接口具有以下特点：

- 10位采样精度
- 0.5 LSB的非线性度
- ±2 LSB的绝对精度
- 13～260μs的转换时间
- 最高分辨率时采样率高达15ksps
- 0～5V的ADC输入电压范围

本节就利用ADC接口实现模拟量的输入。

4.4.1 实例功能

本节实例在0号模拟口连接电位器，通过调整电位器改变输入模拟量的大小。Arduino板每1s进行一次A/D转换，并将结果传给计算机。

4.4.2 硬件电路

将Arduino的0号模拟口接至电位器的中点，电位器另外两端分别连接+5V和地，USB口连接至计算机用于传送数据，模拟I/O实例电路如图4.8所示。

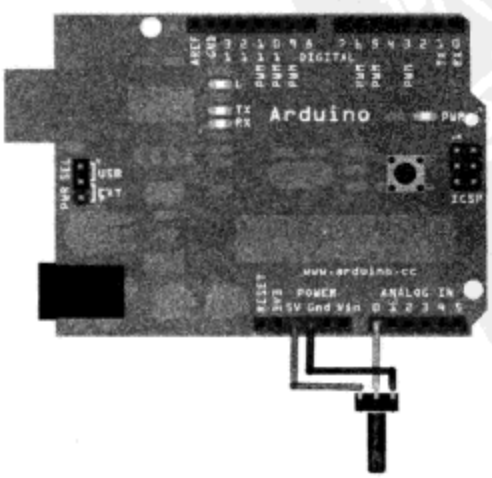


图4.8 模拟I/O口实例电路

4.4.3 程序设计

```
/*********************************************
  模拟I/O口实例程序

  在0号模拟口连接电位器
  每1s进行一次A/D转换并将结果发送给计算机

  created 2011
  by Nille
  Email: chenille@126.com

  This example code is in the public domain.
 *********************************************/

/*********************************************
                初始化部分——setup()函数
*********************************************/
void setup()
{
      //设置串口波特率为9600bps
      Serial.begin(9600);
}

/*********************************************
                执行部分——loop()函数
*********************************************/
void loop()
{
      //延时1s
      delay(1000);
      //进行A/D转换并传输数据
      Serial.print(analogRead(0), DEC);
}
```

4.5 烧写引导程序

在4.1节说过每一块Arduino板在出厂前都在单片机中烧写了引导程序，它可以让Arduino通过USB口进行程序更新。如果我们手上有一块空的AVR单片机想用到Arduino中，则只能自己烧写引导程序了。在Arduino这个完全开源的硬件项目中，AVR单片机的引导程序就在开发环境的文件夹里。本节就使用AVR的开发环境AVR Studio和ISP下载器AVRISP烧写引导

程序。

4.5.1 下载器AVRISP

给一块空芯片烧写程序需要使用烧写器、下载器之类的工具，AVRISP就是一款AVR单片机专用的ISP下载器。如图4.9所示即为一个AVRISP，它采用USB接口与电脑相连，通过一条10芯的扁平线连接目标板，并利用目标板供电，图4.10即为AVRISP的连接示意图。为方便使用某些6芯ISP下载口（比如这里的Arduino），AVRISP配有一个10转6的转换头将10芯ISP下载口转换成Arduino上的6芯ISP下载口。10芯与6芯下载口的关系如表4.1所示。

图4.9 AVRISP

图4.10 AVRISP连接示意图

表4.1 ISP接口定义

信号定义	6芯下载口	10芯下载口	输入/输出	描 述
VCC	2	2	—	目标板上的电源
GND	6	3,4,6,8,10	—	地
MOSI	4	1	输出	AVRISP到目标芯片的指令或数据
MISO	1	9	输入	目标芯片到AVRISP的数据
SCK	3	7	输出	串行时钟，由AVRISP控制
RESET	5	5	输出	复位，由AVRISP控制

4.5.2 AVR Studio

AVR Studio集成开发环境是Atmel公司官方发布的免费软件，专门用于开发AVR单片机的软件平台，使用者可以到Atmel的网站（http://www.atmel.com）下载最新版的AVR Studio。

AVR Studio的安装十分简单，依照软件的提示就能很顺利完成，图4.11所示即为软件的使用界面。这里我们要使用AVR Studio完成引导程序hex文件的烧写。

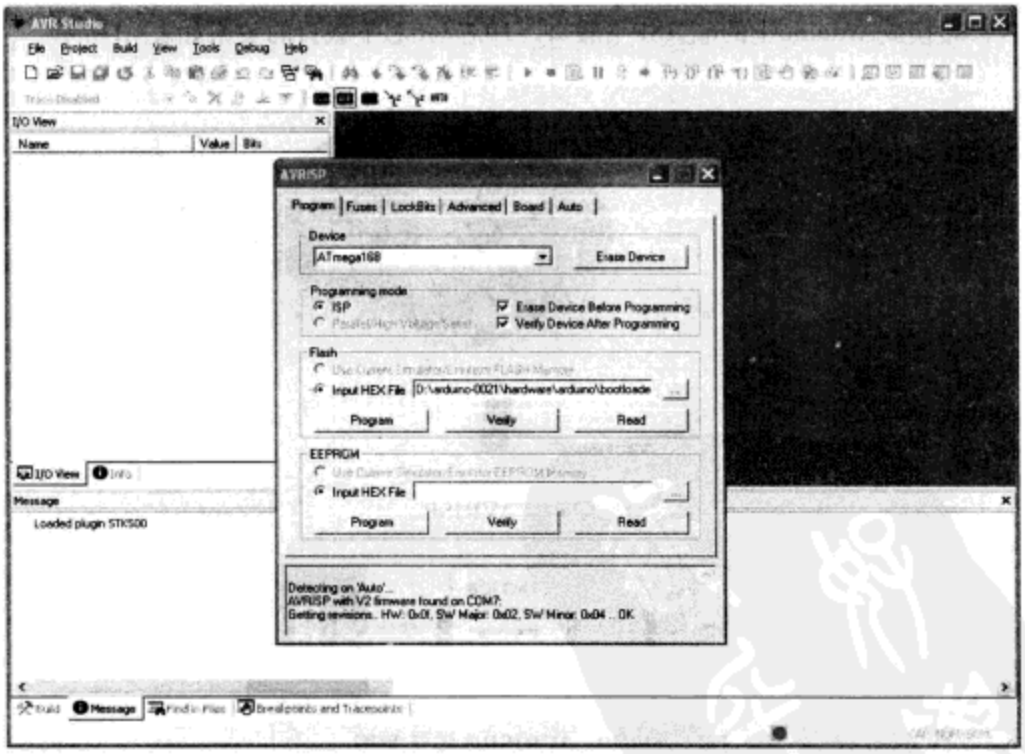


图4.11 AVR Studio使用界面

打开软件后，将AVRISP连接到电脑上，如图4.12所示，在AVR Studio界面中，选择Tools→Program AVR→Connect菜单项，弹出Select AVR Programmer对话框，如图4.13所示。

在Select AVR Programmer对话框中，在Platform列表中选择STK 500 or AVRISP，在Port列表中选择Auto，然后点击Connect按钮，将AVRISP连上AVR Studio。

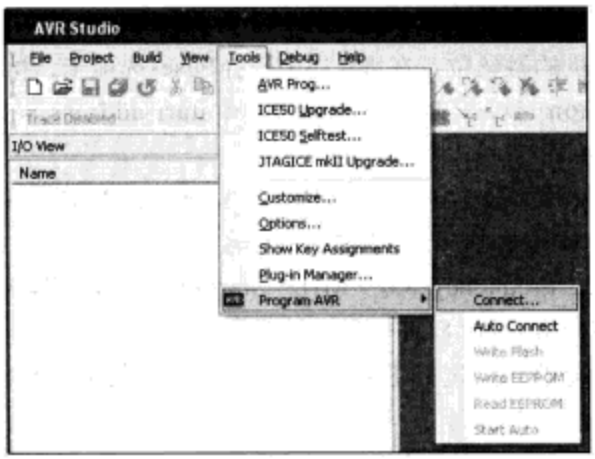


图4.12 下载器连接

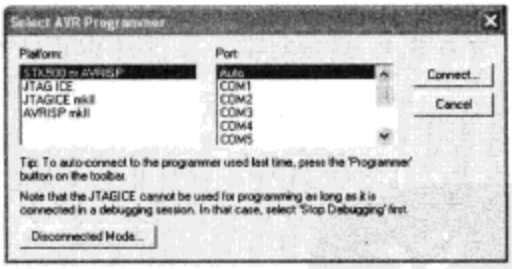


图4.13 Select AVR Programmer对话框

4.5.3 烧写引导程序

如果下载器连接成功，AVR Studio中就会出现AVRISP对话框，如图4.14所示。

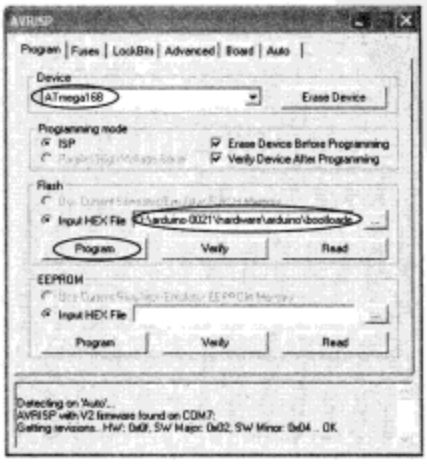


图4.14 AVRISP对话框

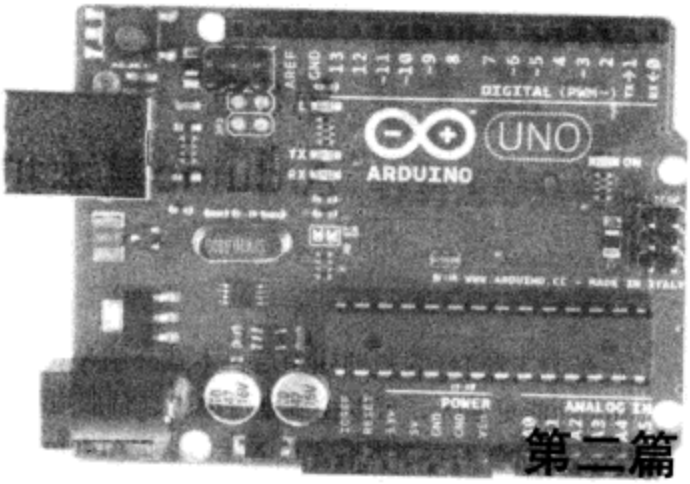
在AVRISP对话框中就可以完成引导程序hex文件的烧写。

首先配置AVR单片机的融丝位，在图4.14中选择Fuses选项卡，按以下内容配置融丝位：

- ❑ Boot Flash section size = 1024 words Boot start address = $1C00; [BOOTSZ=00]; default value
- ❑ Boot Reset Vector Enabled (default address = $0000); [BOOTRST=0]
- ❑ Brown-out detection disabled; [BODLEVEL=111]
- ❑ Ext. Crystal Osc.; Frequency 8.0 - MHz; Startup time PWRDWN/RESET: 16K CK/14 CK + 64ms; [CKSEL=1111 SUT=0]

然后回到Program选项卡，选择芯片型号及引导文件的hex文件后，点击Flash框内的Program按钮完成引导程序的烧写。

提示：引导文件在Arduino开发环境文件夹下的\hardware\arduino\bootloaders子文件夹中。



第二篇

模 块 篇

第5章　Arduino基本扩展模块
第6章　Arduino的扩展库
第7章　无线模块的应用

第5章 Arduino基本扩展模块

在学习了前4章的基础知识后，本章介绍Arduino的一些扩展模块，这些扩展模块使得Arduino功能更强大，可以控制直流电机、伺服电机、网络通信、液晶显示，获取温度、湿度值等。通过学习本章内容，读者能够掌握Arduino的各个功能，为后面的项目应用作铺垫。

5.1 L293 Motor Shield

L293 Motor Shield是一款基于L293B直流电机驱动芯片的Arduino扩展模块，模块可驱动两个电机，采用了可堆叠的设计，可直接插到Arduino控制板上，如图5.1所示，4个LED分别表示两个电机的正反转。

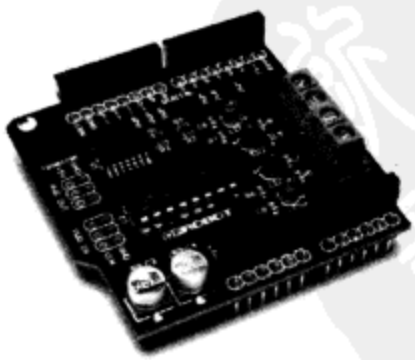


图5.1 电机驱动模块L293 Motor Shield

5.1.1 直流电机的工作原理

大家都知道在磁场中放入通有电流的导体就会产生磁感应效应。直流电机是应用磁感应原理将电能转换为机械能的装置，其转子和定子分别由绕组和永久磁铁组成。直流电机具有调速性能较好和启动转矩较大等特点。

如图5.2所示为一个直流电机的模型，在一对静止的磁极N和S之间，安装一个可绕Z轴旋转的矩形线圈abcd，这个转动的部分通常叫做电枢，线圈的两端a和b分别接到叫做换向片的

两个半圆形铜环上，两个换向片之间彼此是绝缘的，它们和电枢装在同一根轴上，随电枢一起转动。A和B是两个固定不动的碳质铜刷，它们和换向片间是滑动接触的。来自直流电源的电流就是通过电刷和换向片流入线圈的。

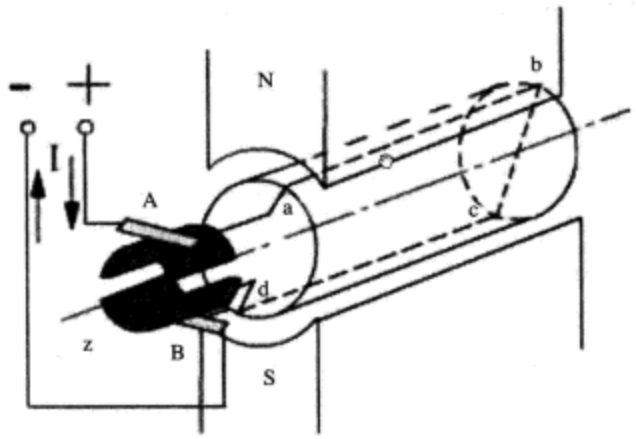


图5.2　直流电机模型图

当电刷A和B分别与直流电源的正极和负极接通时，电流从电刷A流入，而从电刷B流出。这时线圈中的电流方向是从a流向b，从c流向d。由于磁感应效应，当电枢在如图5.3a所示的位置时，线圈ab边的电流从a流向b，用圆圈中的一个加号表示，会受到一个向左下方的力；线圈cd边的电流从c流向d，用圆圈里的一个点表示，会受到一个向右上方的力。这样，电枢上就产生了一个逆时针方向的转矩，电枢就沿着逆时针方向转动起来。

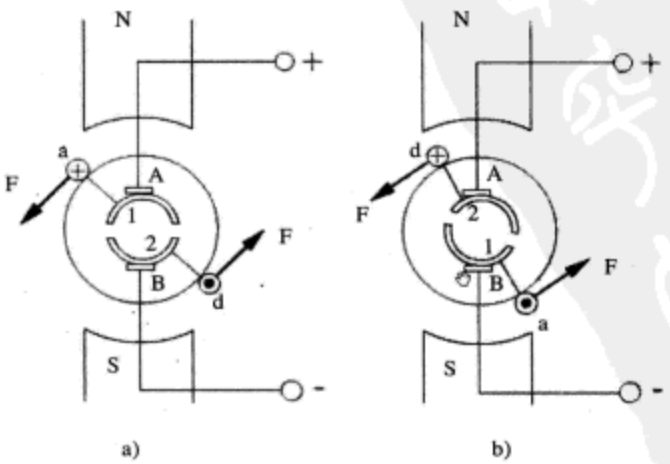


图5.3　电机工作原理

当电枢转到使线圈ab边从N极进入S极，而cd边从S极进入N极时，与线圈a端连接的换向片跟B接触，而与线圈d端接触的换向片跟A接触，如图5.3b所示。这样，线圈内的电流方向

变为从d流向c，再从b流向a，从而保证在N极下方的导体中电流方向不变，因此转矩的方向也不变，电枢依然按照原来的方向旋转。这样，再通过传动装置，直流电机就可以带动其他部件转动了。

由原理分析可以发现，线圈中的电流越大，线圈在磁场中所受到的力就越大，电机的转速就会越快；反之电流越小，电机转速就越慢。

5.1.2 H桥驱动电路

直流电机的正反转实际上是通过变换直流电压的正负极实现的，对于图5.2所示的直流电机模型，当电刷A和B分别与直流电源的正极和负极接通时，电枢逆时针旋转，若A端接直流电源负极，B端接直流电源正极，则直流电机顺时针旋转。

在实际应用中常采用图5.4所示的电路驱动直流电机。电路名为“H桥驱动电路”，这是因为它的形状酷似字母H。如图5.4所示，H桥驱动电路包括4个三极管和一个电机。要使电机运转，必须导通对角线上的一对三极管。根据不同三极管对的导通情况，电流可能会从左至右或从右至左流过电机，从而控制电机的转向。

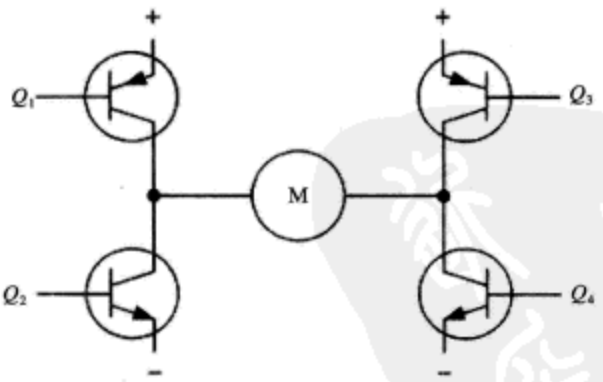


图5.4 H桥驱动电路示意图

如图5.5a所示，当Q1管和Q4管导通时，电流从电源正极经Q1从左至右穿过直流电机，然后再经Q4回到电源负极，从而驱动直流电机沿一个方向旋转（假设电流由左向右穿过电机时，直流电机顺时针旋转）；在图5.5b中另一对三极管Q2和Q3导通的情况下，电流将从右至左流过直流电机，从而驱动直流电机沿另一方向转动。

注意：在实际应用中驱动直流电机时，保证H桥上两个同侧的三极管不会同时导通非常重要。如果三极管Q1和Q2同时导通，那么电流就会从正极穿过两个三极管直接回到负极。此时，电路中除了三极管外没有其他任何负载，电路上的电流就可能达到最大值（该电流仅受电源性能限制），甚至烧坏三极管。

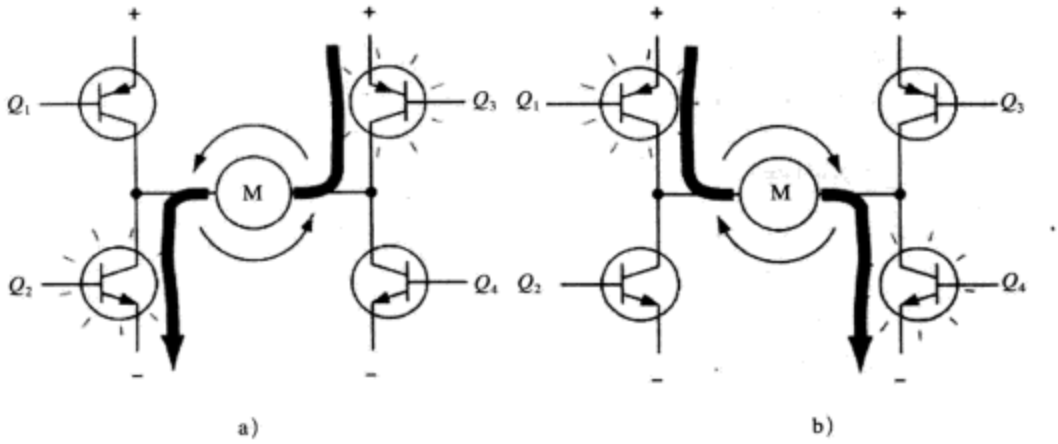


图5.5　H桥驱动电路控制直流电机示意图

5.1.3　线性放大调速原理

直流电机的转速计算公式如下：

$$n=(U-IR)/K\Phi$$

其中U为电枢端电压；I为电枢电流；R为电枢电路总电阻；Φ为每极磁通量；K为电动机结构参数。

由公式可以发现对一个各参数已经确定的直流电机调速的可控量就是电压。通过调整电压就能够调整直流电机的转速，电压高直流电机转速就快；反之，电压低，直流电机转速就慢。

通过线性放大电路就能够方便地实现调节直流电机转速的效果。线性放大电路调速原理示意图如图5.6所示，调节电位器的阻值大小，就会调整电位器的分压值，从而改变电机两端的电压值。

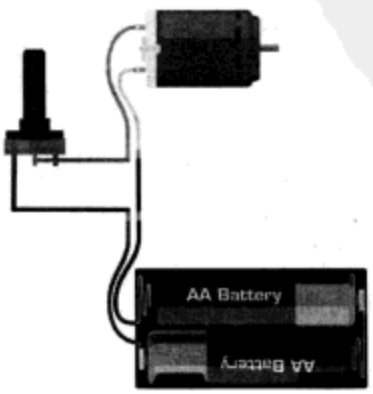


图5.6　线性放大电路调速原理示意图

线性放大电路调速的特点是：

- 与直流电机的转矩无关、消耗大量的电流，效率低。
- 电位器分压产生大量的热量。

5.1.4 PWM调速原理

在第3章的analogWrite函数里已经介绍过通过调节PWM的占空比在引脚上输出不同的电压值。脉冲宽度调制（PWM）是一种对模拟信号电平进行数字编码的方法。通过高分辨率计数器的使用，调制方波的占空比用来对一个具体模拟信号的电平进行编码。PWM信号仍然是数字的，因为在给定的任何时刻，满幅值的直流供电要么完全有（ON），要么完全无（OFF）。电压或电流源是以一种通（ON）或断（OFF）的重复脉冲序列加到模拟负载上去的。通的时候即是直流供电加到负载上的时候，断的时候即是供电断开的时候。只要带宽足够，任何模拟值都可以使用PWM进行编码。

采样控制理论中有一个重要结论：当冲量相等而形状不同的窄脉冲加在具有惯性的环节上时，其效果基本相同。PWM控制技术就是以该结论为理论基础，对半导体开关器件的导通和关断进行控制，使输出端得到一系列幅值相等而宽度不相等的脉冲，用这些脉冲来代替正弦波或其他所需要的波形。按一定的规则对各脉冲的宽度进行调制，既可改变逆变电路输出电压的大小，也可改变输出频率。

PWM的一个优点是从处理器到被控系统信号都是数字形式的，无须进行数模转换。让信号保持为数字形式可将噪声影响降到最小。噪声只有在强到足以将逻辑1改变为逻辑0或将逻辑0改变为逻辑1时，才能对数字信号产生影响。对噪声抵抗能力的增强是PWM相对于模拟控制的另外一个优点，而且这也是在某些时候将PWM用于通信的主要原因。从模拟信号转化为PWM可以极大地延长通信距离。在接收端，通过适当的RC或LC网络可以滤除调制高频方波并将信号还原为模拟形式。

5.1.5 L293 Motor Shield的原理

电机驱动模块L293 Motor Shield驱动直流电机就采用PWM方式，其核心是一块直流电机驱动芯片L293B，驱动电机的电源取自Arduino的Vin引脚，模块原理图如图5.7所示。

L293B直流电机驱动芯片允许的电压范围是4.5～36V，内有四重推挽（双重H桥集成功放电路）驱动电路，两个通道可以向各自的电机提供1 A的驱动电流，并且如果芯片过热，芯片能够自动关断，保障系统不受损坏。L293B内部结构图及典型应用如图5.8所示。

在图5.7中，当DIRA、DIRB为高电平时，两个电机的电流分别由3引脚流向6引脚和11引脚流向14引脚，假设此时电机正转；反之，当DIRA、DIRB为低电平时，电机电流分别由6引脚流向3引脚和14引脚流向11引脚，电机反转。此时，可以用PWM控制芯片上电机使能脚的通断时间比来对电机进行调速。

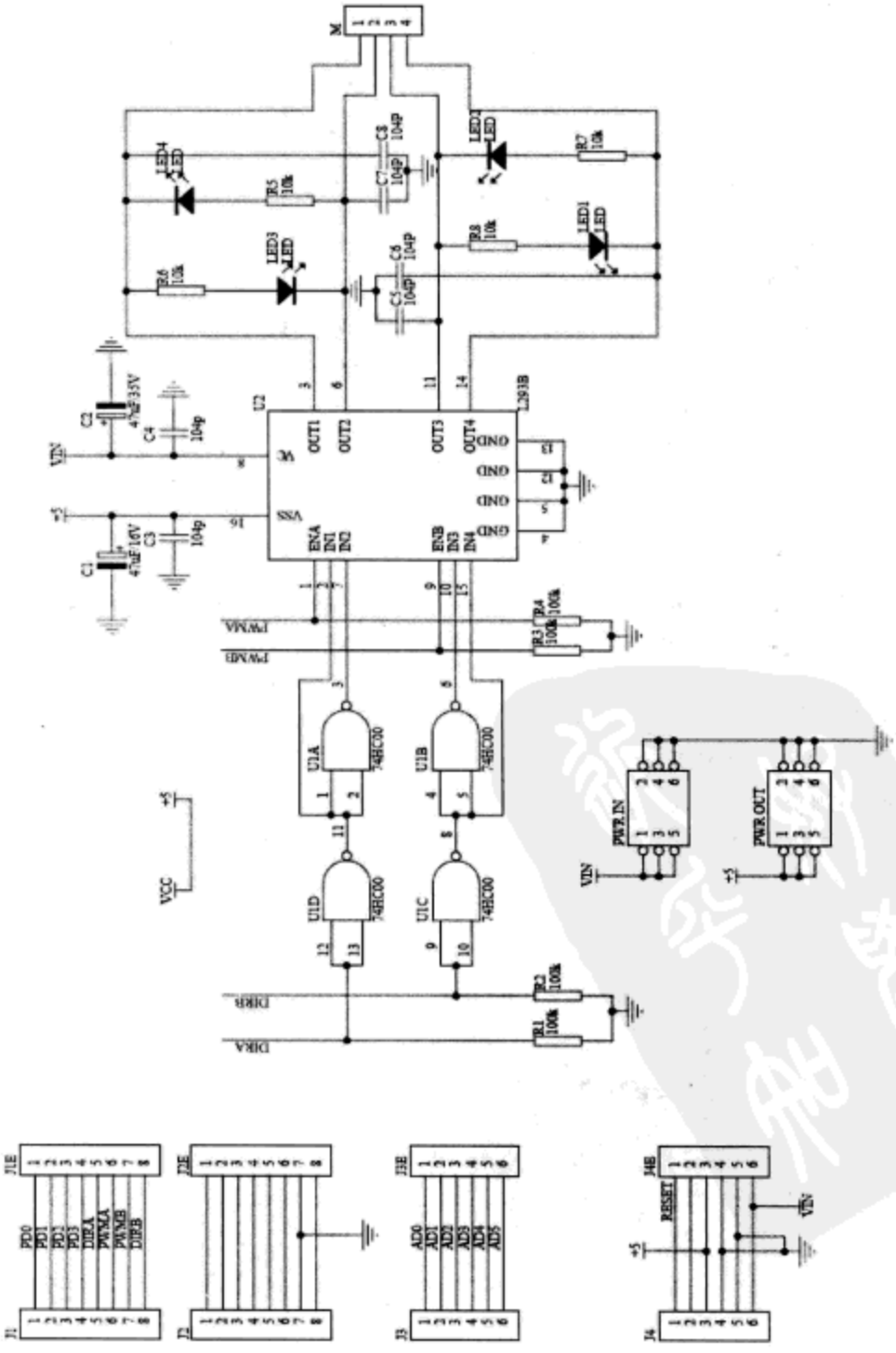


图5.7　L293 Motor Shield原理图

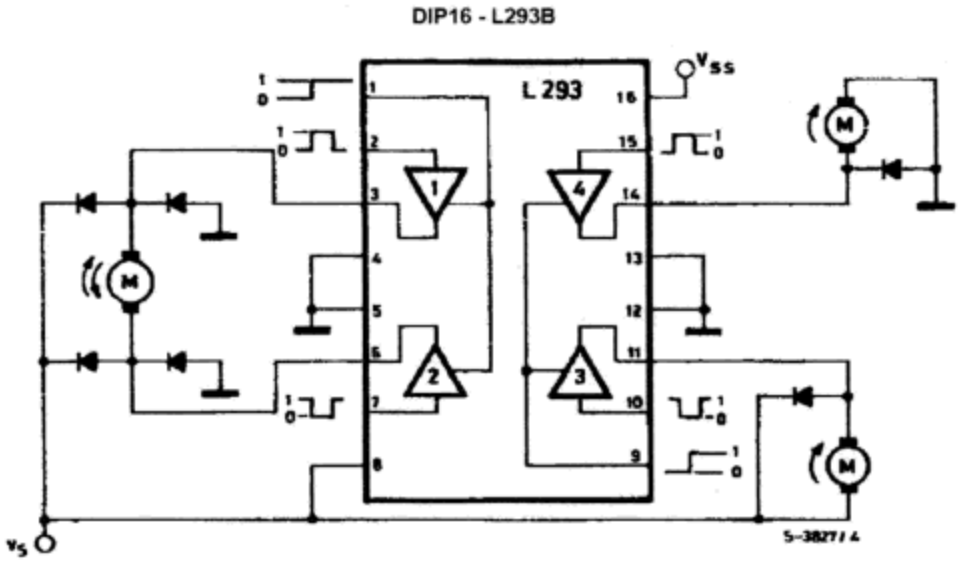


图5.8 L293B典型应用

电机驱动模块L293 Motor Shield占用Arduino控制板的数字引脚4、5、6、7，分别对应DIRA、PWMA、PWMB和DIRB，DIRA和PWMA共同控制电机A，DIRB和PWMB共同控制电机B。

5.1.6 L293 Motor Shield的应用

由于L293 Motor Shield采用可堆叠设计，因此它的应用非常方便，只要将模块对应地插到Arduino控制板上就可以了。在本节中，在L293 Motor Shield上连接一个直流电机，用Arduino控制直流电机由静止逐渐变换到快速，再逐渐变回到静止，反方向同样变换一次。从静止到快速及从快速到静止的变化时间均为25.5s。直流电机的连接示意图如图5.9所示（在图5.9中未画出Arduino控制板），对应Arduino控制板的引脚为4和5。

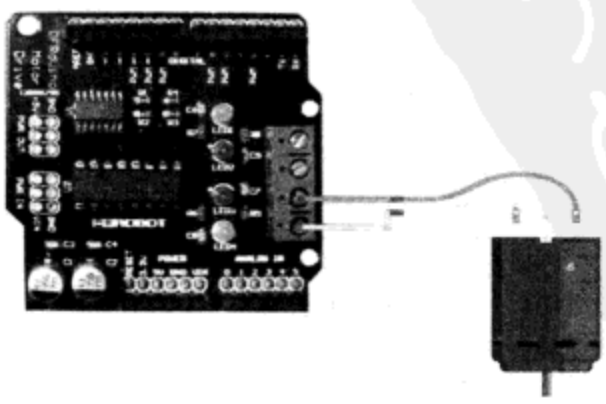


图5.9 直流电机连接示意图

注意：由于电机驱动的电流较大，因此Arduino须采用单独的7～12V电源供电。

5.1.7 程序设计

```
/*************************************************
直流电机驱动实例程序

占用引脚4、引脚5
控制直流电机由静止逐渐变换到快速，再逐渐变回到静止，反方向同样变换一次
从静止到快速及从快速到静止的变化时间均为25.5s

created 2011
by Nille
Email: chenille@126.com

This example code is in the public domain.
*************************************************/


/************************************************
                初始化部分——setup()函数
************************************************/
void setup()
{
      //设置4号引脚为输出，控制直流电机正反转
      pinMode(4, OUTPUT);
      //设置5号引脚为输出，控制直流电机转速
      pinMode(5, OUTPUT);
}

/************************************************
                执行部分——loop()函数
************************************************/
void loop()
{
      //引脚4输出高电平，直流电机正转
      digitalWrite(4, HIGH);

      //直流电机由静止逐渐变换为快速
      for(int i = 0; i<255 ; i++)
  {
                  //控制电机转动，i表示转速
```

```
                analogWrite(5,i);
                //延时0.1s，i由0变化到255需25.5s
                delay(100);
        }

        //直流电机由快速逐渐变换为静止
        for( i = 255; i>0 ; i--)
        {
                //控制电机转动，i表示转速
                analogWrite(5,i);
                //延时0.1s，i由255变化到0需25.5s
                delay(100);
        }

        //引脚4输出低电平，直流电机反转
        digitalWrite(4, LOW);

        //直流电机由静止逐渐变换为快速
            for( i = 0; i<255 ; i++)
        {
                //控制电机转动，i表示转速
                analogWrite(5,i);
                //延时0.1s，i由0变化到255需25.5s
                delay(100);
        }

        //直流电机由快速逐渐变换为静止
        for( i = 255; i>0 ; i--)
    {
                //控制电机转动，i表示转速
                analogWrite(5,i);
                //延时0.1s，i由255变化到0需25.5s
                delay(100);
    }
}
```

5.1.8 程序分析

首先在setup( )函数中设置直流电机驱动的两个引脚4和5为输出，然后在loop( )函数中对引脚4、5进行控制。

引脚4控制直流电机的正反转，使用数字I/O控制函数digitalWrite( )，函数描述见3.1节。程序中两次使用该函数对引脚4的输出电平进行控制，用于在直流电机由静止变为快速，再

变回静止后对直流电机旋转方向的改变。

引脚5控制直流电机的转速，使用模拟I/O控制函数analogWrite( )，函数描述见3.2节。在程序的for循环内不断改变PWM的占空比，调整直流电机的转速。调速周期为每0.1s一次，在程序中通过delay( )函数实现。

注意：在实际应用中，由于直流电机存在一个启动电压，一旦小于启动电压，直流电机就处于静止状态，因此直流电机静止的时间就显得较长。可根据实际情况改变程序中的i值来缩短直流电机静止的时长。

5.1.9 程序的精练

通过对程序的分析会发现在5.1.7节的程序代码中对直流电机速度控制的代码段都是十分类似的，通过引脚4调整直流电机正反转后的调速代码甚至是一样的。5.1.7节中的代码流程固然清晰，但代码的集成度不高，本节通过在程序中加入两条判断语句来实现程序的精练。程序代码如下：

```
/*************************************************
  直流电机驱动实例程序——精练程序

     占用引脚4、引脚5
  控制直流电机由静止逐渐变换到快速，再逐渐变回到静止，反方向同样变换一次
     从静止到快速及从快速到静止的变化时间均为25.5s

created 2011
  by Nille
  Email: chenille@126.com

  This example code is in the public domain.
 *************************************************/

/************************************************
                    初始化部分——setup()函数
************************************************/
void setup()
{
          //设置4号引脚为输出，控制直流电机正反转
          pinMode(4, OUTPUT);
          //设置5号引脚为输出，控制直流电机转速
          pinMode(5, OUTPUT);
}
```

```
/*************************************************
                执行部分——loop()函数
*************************************************/
void loop()
{
  //通过判断引脚4的电压值来对引脚4输出进行控制
        if(LOW == digitalRead(4))
        {
                //引脚4输出高电平，直流电机正转
                digitalWrite(4, HIGH);
        }
        else
        {
                //引脚4输出低电平，直流电机反转
                digitalWrite(4, LOW);
        }

        //对直流电机调速
        for(int i = 0; i<511 ; i++)
        {
                //控制电机转动
                if(i<255)
                {
                        //如果i小于255，则加速
                        analogWrite(5,i);
                }
                else
                {
                        //如果i大于255，则减速
                        analogWrite(5,511-i);
                }
                //延时0.1s
                delay(100);
        }
}
```

在上述程序中，通过if(LOW == digitalRead(4))和if(i<255)两条判断语句，实现了程序的精练，一条语句用在控制引脚4的输出电压之前，用来在直流电机调速完成后对正转、反转的切换；另一条语句用在控制直流电机转速的函数之前，用来决定是加速还是减速。

5.2 Input Shield

Input Shield是一款带有摇杆和按键的输入扩展板，同时预留了无线通信模块接口，采用

堆叠设计，可用于Arduino的交互设计当中。模块如图5.10所示。

5.2.1 Input Shield原理图

图5.10 Input Shield

如图5.11所示，Input Shield扩展板具有一个摇杆（含一个按键）、两个大圆帽按键和1个复位按键，同时预留一个无线通信模块接口。摇杆可以理解为一个两轴的可调电位器，它能输出两个模拟信号，以实现上下左右的控制，分别占用Arduino控制板的模拟口0和模拟口1，摇杆按键占用Arduino控制板的数字引脚5；B圆帽按键（即Input Shield上的红色按键）占用数字引脚3；C圆帽按键（即Input Shield上的蓝色按键）占用数字引脚4；无线通信模块接口占用数字引脚0和1。

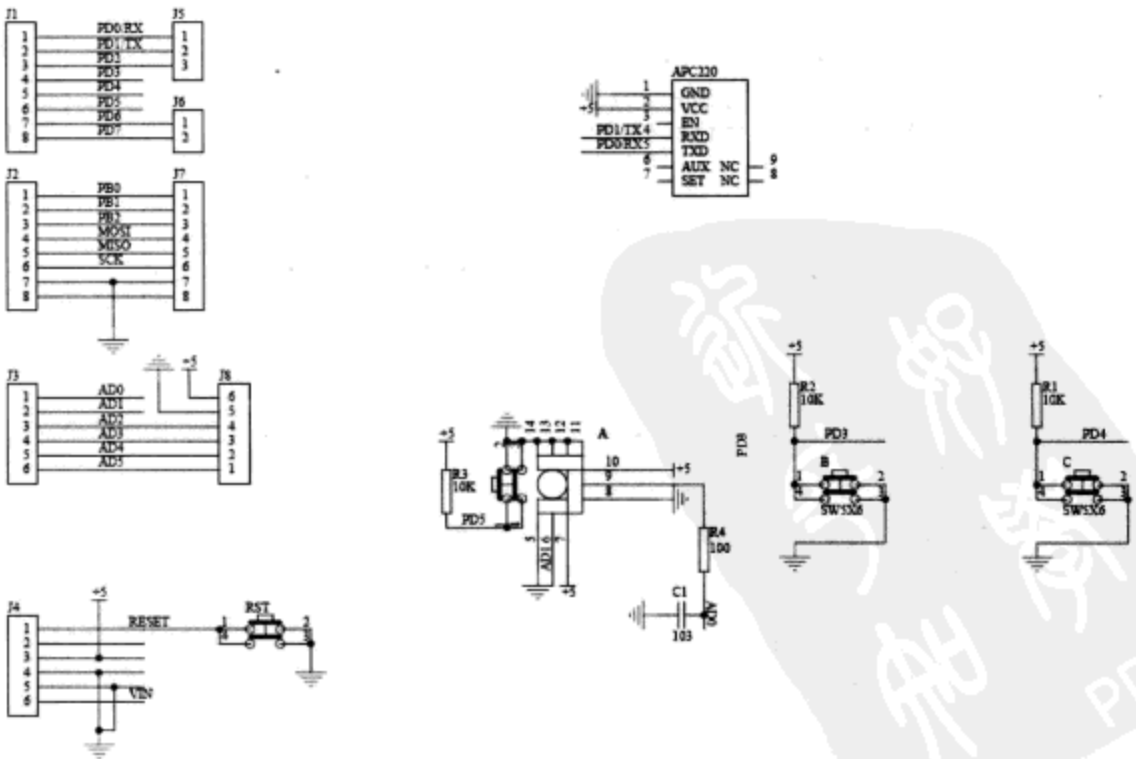


图5.11 Input Shield原理图

5.2.2 Input Shield的实例

本节中将Input Shield扩展板对应插到Arduino控制板上，每1s获取一次摇杆上下方向和左右方向的模拟量数据，通过串口通信传送到电脑上，在Arduino开发环境下的 Serial

Monitor（串口监视窗）中查看。

5.2.3 程序设计

```
/*********************************************
获取摇杆模拟量实例程序

摇杆上下方向和左右方向分别占用Arduino的模拟口0和模拟口1
每1s进行一次A/D转换并将结果发送给计算机

created 2011
by Nille
Email: chenille@126.com
This example code is in the public domain.
*********************************************/

/*************************************************
                初始化部分——setup()函数
*************************************************/
void setup()
{
        //设置串口波特率为9600bps
        Serial.begin(9600);
}

/*************************************************
                执行部分——loop()函数
*************************************************/
void loop()
{
        //延时1s
        delay(1000);
        //显示水平模拟量值，
        Serial.print("horizontal: ");
        Serial.print(analogRead(1), DEC);

        //显示垂直模拟量
        Serial.print(", vertical: ");
        Serial.println (analogRead(0), DEC);
}
```

5.2.4 程序分析

在程序中应用analogRead( )函数获取模拟口0和模拟口1的值，传送到电脑端，效果如图5.12所示。通过操纵摇杆可看到模拟量数据的变化，由于Arduino采用10位的ADC，因此模拟量的变化范围是0～1023。在水平方向上，越向右，模拟量的值越小；反之模拟量的值越大。在垂直方向上，越向上模拟量的值越小，反之模拟量的值越大。

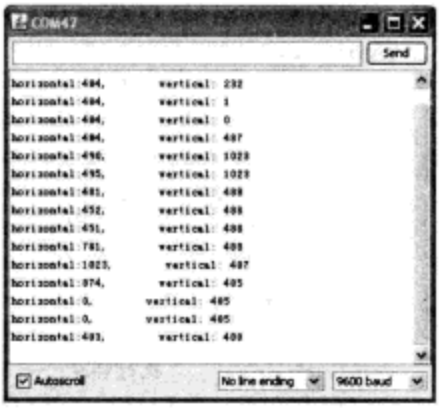


图5.12 摇杆操纵显示效果

5.2.5 使用摇杆控制直流电机转速

将Input Shield扩展板对应插到5.1节L293 Motor Shield应用的实例中，可实现直流电机转速的交互式控制，人为地用摇杆控制直流电机的转速。直流电机的连接位置依然采用5.1.6节中的连接方式，当摇杆位于中位时，直流电机静止不动；当向上推动摇杆时，直流电机逐渐加速正转；当向下拉动摇杆时，直流电机逐渐加速反转。程序清单如下：

```
/*************************************************
  摇杆控制直流电机转速实例程序

直流电机控制占用引脚4、5
  引脚4控制直流电机的正反转——数字量输出
  引脚5控制直流电机的转速——模拟量输出

  摇杆垂直位置模拟量获取占用模拟量引脚0——模拟量输入
  向上推摇杆模拟量值越来越小
  向下拉摇杆模拟量值越来越大
  模拟量范围0～1023

当摇杆位于中位时，直流电机静止不动
当向上推动摇杆时，直流电机逐渐加速正转
当向下拉动摇杆时，直流电机逐渐加速反转

  摇杆位置模拟量获取频率为10Hz（每秒10次，即0.1s一次）

created 2011
  by Nille
  Email: chenille@126.com

  This example code is in the public domain.
 *************************************************/
```

```
int  verticalVal;                    //定义整型变量verticalVal存储AD值
/*************************************************
              初始化部分——setup()函数
*************************************************/
void setup()
{
      //设置4号引脚为输出，控制直流电机正反转
      pinMode(4, OUTPUT);
      //设置5号引脚为输出，控制直流电机转速
      pinMode(5, OUTPUT);
}

/*************************************************
              执行部分——loop()函数
*************************************************/
void loop()
{
       //延时0.1s
       delay(100);
       //获取模拟量AD值
       verticalVal = analogRead(0);

       //判断电机正转还是反转
       if(verticalVal < 512)
       {
                //如果模拟量值小于512，则直流电机正转
                //引脚4输出高电平，直流电机正转
                digitalWrite(4, HIGH);
                //设置直流电机转速
                analogWrite(5, map(512- verticalVal, 0, 511, 0, 255));
       }
       else
       {
                //否则直流电机反转
                //引脚4输出低电平，直流电机反转
                digitalWrite(4, LOW);
                //设置直流电机转速
                analogWrite(5, map(verticalVal, 512, 1023, 0, 255));
       }
}
```

在程序中对摇杆输入模拟量的值转换成PWM输出的值时使用了map( )函数，该函数是将数据进行等比映射，可参考3.5节。这是由于在摇杆处于中间位置时，采集到的模拟量的值

为512（0～1023的中间值），而此时直流电机是静止的；当摇杆向上推时，模拟量的值在0～512区间，此时需要控制直流电机正转，转速范围在0～255区间；当摇杆向下拉时，模拟量的值在512～1023区间，此时需要控制直流电机反转，转速范围同样在0～255区间。

5.3 LCD Keypad Shield

LCD Keypad Shield是一块液晶显示扩展板，使用的是标准1602液晶，该液晶可显示两行内容，每行可显示16个英文字符，具有对比度调节电位器，同时提供5个按键输入，1个复位键，扩展板如图5.13所示。

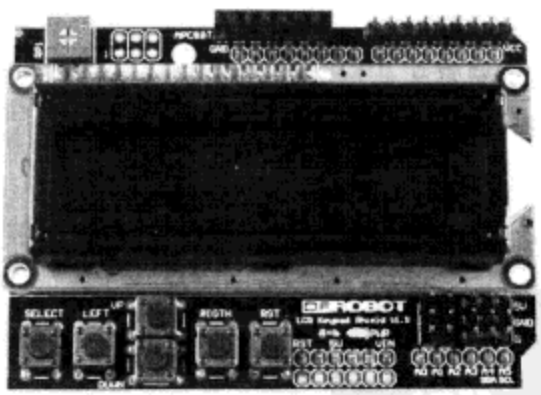


图5.13 LCD Keypad Shield

5.3.1 液晶显示原理

液晶显示（Liquid Crystal Display，LCD）的应用在我们的生活中已经随处可见，如手机、MP3、仪器仪表等。液晶显示的原理是利用液晶的物理特性，通过电压对其显示区域进行控制：通电时液晶排列变得有序，光线容易通过；不通电时排列混乱，阻止光线通过。

液晶显示按其显示方式可分为线段式、字符式、点阵式等。除了黑白显示外，还有多灰度显示、彩色显示等。LCD Keypad Shield扩展板使用的是字符式液晶，这是一种专门用于显示字母、数字、符号的液晶模块。

提示：Arduino提供了类库，可方便地实现液晶显示模块的控制，而不需要知道液晶显示的原理、控制方式等基础知识，希望尽快进行液晶显示模块应用的读者可直接跳到5.3.8阅读。

5.3.2 标准1602液晶模块

标准1602液晶模块除了液晶显示屏外，还包括行列驱动器、控制器及连接附件等，它

是一种将液晶显示器件、控制集成电路、PCB板、背光源、结构件装配在一起的集合。提供与控制芯片的简单接口，简化了液晶显示应用的控制方式。标准1602液晶模块实物如图5.14所示。

图5.14 标准1602液晶模块

1602液晶模块主要技术参数如下：

- 显示容量：16×2个字符
- 芯片工作电压：4.5～5.5V
- 工作电流：2.0mA（5V）
- 字符尺寸：2.95mm×4.35mm

采用单排16芯接口，引脚定义如下表所示。

表5.1 1602液晶模块引脚定义

编号	符号	引脚定义	功能描述
1	VSS	电源地	电源地
2	VDD	电源正	接+5V电源
3	VL	液晶显示偏压	对比度调节 接正电源时对比度最弱，接电源地时对比度最高，一般通过一个电位器调节
4	RS	数据/命令寄存器选择	高电平：选择数据寄存器 低电平：选择命令寄存器
5	R/W	读/写选择	高电平：读操作 低电平：写操作
6	E	使能	电平下降沿触发模块工作
7	D0	数据0	双向数据口
8	D1	数据1	
9	D2	数据2	
10	D3	数据3	
11	D4	数据4	
12	D5	数据5	
13	D6	数据6	
14	D7	数据7	
15	BLA	背光正极	背光灯阳极
16	BLK	背光负极	背光灯阴极

5.3.3 1602液晶模块控制方式

在1602液晶模块的内部显示存储器中预先保存好的字符图形符号，其显示工作流程就是通过控制器向1602写入指定的显示存储器地址，相应地址对应的字符图形符号就分别显示在液晶屏幕上。1602内部显示存储器保存的字符图形符号包括阿拉伯数字、英文字母大小写、常用符号、日文平假名和片假名等，共160个，每一个字符图形符号都有一个固定的地址。

1602液晶模块的控制指令共有11条，如表5.2所示。

表5.2 1602液晶模块的控制指令

序号	指 令	RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
1	清屏	0	0	0	0	0	0	0	0	0	1
2	光标复位	0	0	0	0	0	0	0	0	1	*
3	光标和显示模式设置	0	0	0	0	0	0	0	1	I/D	S
4	显示开/关控制	0	0	0	0	0	0	1	D	C	B
5	光标或字符移位	0	0	0	0	0	1	S/C	R/L	*	*
6	功能设置	0	0	0	0	1	DL	N	F	*	*
7	显示存储器地址设置	0	0	0	1	显示存储器地址					
8	数据存储器地址设置	0	0	1	数据存储器地址						
9	读忙标志和光标地址	0	1	BF	计数器地址						
10	写数据到CGRAM或DDRAM	1	0	要写的数据内容							
11	从CGRAM或DDRAM读数据	1	1	读出的数据内容							

1602液晶模块的读/写操作、屏幕和光标的操作都是通过控制指令实现的，表中1表示高电平，0表示低电平，*表示无效。

指令1：清屏，指令码01H，光标复位到地址00H。

指令2：光标复位，光标返回到地址00H。

指令3：光标和显示模式设置。

I/D：光标移动方向，高电平右移，低电平左移。

S：屏幕上所有文字是否左移或右移，高电平表示有效，低电平表示无效。

指令4：显示开/关控制。

D：控制整体显示的开或者关，高电平表示开显示，低电平表示关显示。

C：控制光标的开或者关，高电平表示有光标，低电平表示无光标。

B：控制光标是否闪烁，高电平闪烁，低电平不闪烁。

指令5：光标或字符移位。

S/C：高电平时移动显示的文字，低电平时移动光标。

R/L：高电平时向右滚动，低电平时向左滚动。

指令6：功能设置。

DL：高电平时为8位总线，低电平时为4位总线。

N：低电平时为单行显示，高电平时为双行显示。

F：低电平时显示5×7的点阵字符，高电平时显示5×10的点阵字符。

指令7：显示存储器RAM地址设置。

指令8：数据存储器地址设置。

指令9：读忙信号和光标地址。

BF：忙标志位，高电平表示忙，此时模块不接收命令或者数据；低电平表示不忙。液晶显示模块是一个慢显示器件，所以执行每条指令之前一定要确定模块的忙标志位为低电平。

指令10：写数据。

指令11：写数据。

在1602液晶模块上显示字符前必须指定字符显示的位置，所以必须知道1602的内部显示地址，表5.3为1602液晶模块的内部显示地址分配。

表5.3 1602液晶模块的内部显示地址分配

第1行															
00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
第2行															
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F

注意：由表5.2中指令8可以看到，设置显示地址时，数据最高位D7始终为高电平，所以实际写入的数据应为显示地址+10000000B（80H）。比如写入00H地址实际应写入：00H + 80H = 80H。

对1602液晶模块的读/写操作时序如图5.15和图5.16所示。

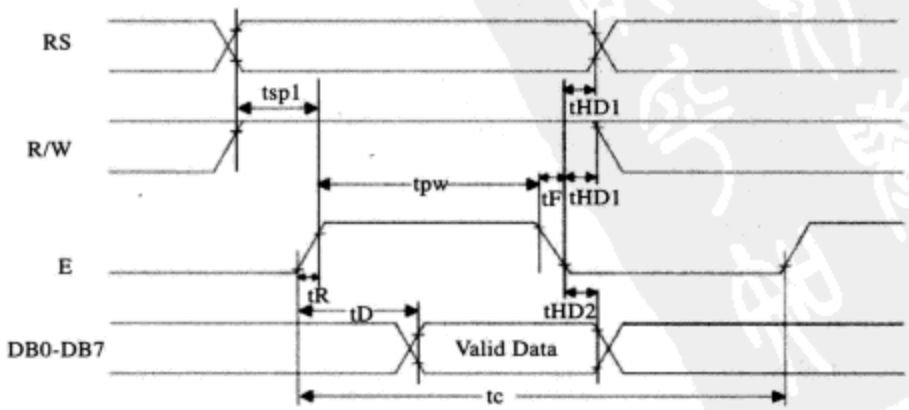


图5.15 读操作时序

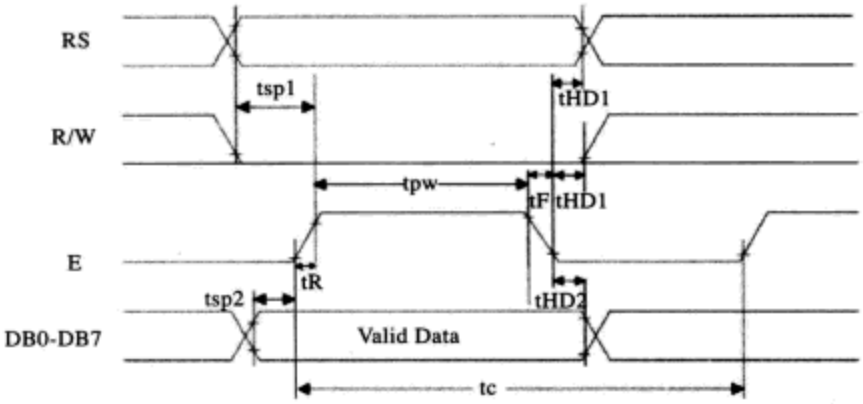


图5.16 写操作时序

1602液晶模块推荐的初始化过程如下：

(1) 延时15ms

(2) 写指令28H（不检测忙信号）

(3) 延时5ms

(4) 写指令28H（不检测忙信号）

(5) 延时5ms

(6) 写指令28H（不检测忙信号）

(7) 延时5ms

(8) 写指令28H：功能设置

(9) 写指令01H：清屏

(10) 写指令06H：光标和显示模式设置

(11) 写指令0CH：显示开

5.3.4 LCD Keypad Shield原理图

由LCD Keypad Shield原理图（见图5.17）可以看出，扩展板占用数字口4、5、6、7、8、9和10采用4位总线方式控制1602液晶模块，其中4、5、6、7对应液晶模块的D4、D5、D6、D7；8对应RS；9对应E；而10用于控制液晶的背光。

另外，LCD Keypad Shield扩展板使用了1个模拟口获取5个按键的值，当没有按下按键时，模拟口只接了一个2KΩ的上拉电阻，此时模拟口的电压值为5V；当“RIGHT”键按下时，模拟口直接接地，此时模拟口的电压值为0V；当按下“UP”键时，模拟口通过一个330Ω的电阻接地，此时模拟口的电压值为2kΩ和330Ω电阻的分压值，约为0.71V；其他按键同理。通过模拟接口获取多个按键的值。

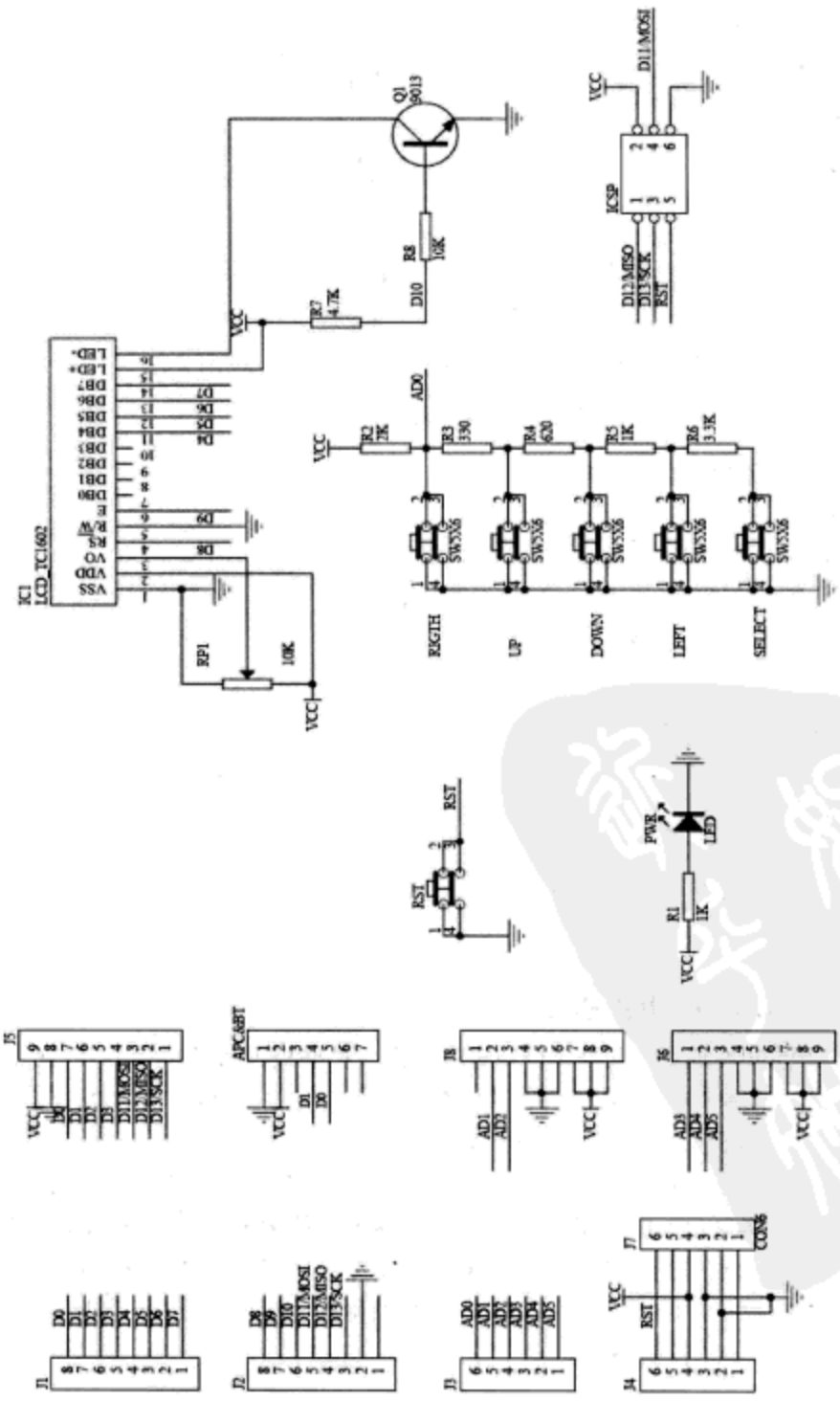


图5.17 LCD Keypad Shield原理图

5.3.5 LCD Keypad Shield应用实例

本节中将LCD Keypad Shield扩展板对应插到Arduino控制板上，在液晶左上角的位置循环显示0～9 10个数字，循环周期为5s变化一次，单数背光灯亮，双数背光灯熄灭。

5.3.6 程序设计

```
/*************************************************
  LCD循环显示0～9实例程序
  每5s变化一个数字，单数背光灯亮，双数背光灯熄灭
  显示位置：液晶左上角

4位总线
* LCD RS 连接到数字I/O PIN8
* LCD Enable连接到数字I/O PIN9
* LCD D4连接到数字I/O PIN4
* LCD D5连接到数字I/O PIN5
* LCD D6连接到数字I/O PIN6
* LCD D7连接到数字I/O PIN7
* LCD 背光灯控制使用数字I/O PIN10

created 2011
  by Nille
  Email: chenille@126.com

  This example code is in the public domain.
 *************************************************/

/*************************************************/
#define PortEn   9              //定义使能引脚
#define PortRS   8              //定义数据/命令选择引脚
#define PortBL   10             //定义背光控制引脚


/*************************************************
 写数据函数
函数功能：向1602液晶模块发送数据
入口参数：value——要发送的值
出口参数：无
*************************************************/
void lcdDat(uint8_t value)
{
        digitalWrite(PortRS,  HIGH); //RS置高，发送的是数据
        digitalWrite(PortEn,  LOW);  //使能位拉低
```

```
        //先传送高4位
        for(int i = 4 ;i < 8 ; i++)
        digitalWrite( i, (value >> i) & 0x01);

        //产生使能位下降沿
        digitalWrite(PortEn, HIGH);
        digitalWrite(PortEn, LOW);

        //再传送低4位
        for(int i = 4 ;i < 8 ; i++)
        digitalWrite( i, (value >> i - 4) & 0x01);

        //产生使能位下降沿
        digitalWrite(PortEn, HIGH);
        digitalWrite(PortEn, LOW);
}

/************************************************
写命令函数
函数功能：向1602液晶模块发送命令
入口参数：value——要发送的命令
出口参数：无
************************************************/
void lcdCmd(uint8_t value)
{
        digitalWrite(PortRS, LOW);   //RS置低，发送的是命令
        digitalWrite(PortEn, LOW);   //使能位拉低

        //先传送高4位
        for(int i = 4 ;i < 8 ; i++)
        digitalWrite( i, (value >> i) & 0x01);

        //产生使能位下降沿
        digitalWrite(PortEn, HIGH);
        digitalWrite(PortEn, LOW);

        //再传送低4位
        for(int i = 4 ;i < 8 ; i++)
        digitalWrite( i, (value >> i - 4) & 0x01);

        //产生使能位下降沿
        digitalWrite(PortEn, HIGH);
```

```
        digitalWrite(PortEn, LOW);
}

/*************************************************
LCD初始化函数
函数功能：初始化1602液晶模块
入口参数：无
出口参数：无
*************************************************/
void lcdInit()
{
        delay(15);
        lcdCmd(0x28);    //功能设置，4位总线，双行显示
        delay(10);
        lcdCmd(0x28);    //功能设置，4位总线，双行显示
        delay(10);
        lcdCmd(0x28);    //功能设置，4位总线，双行显示
        delay(10);
        lcdCmd(0x01);    //清屏
        delay(10);
        lcdCmd(0x06);    //光标和显示模式设置
        delay(10);
        lcdCmd(0x0C);    //显示开，无光标
        delay(10);

}

/*************************************************
字符显示函数
函数功能：在1602液晶模块上显示字符
入口参数：pos——字符显示的位置
data——显示的字符
出口参数：无
*************************************************/
void lcdDisplay(uint8_t pos, uint8_t data)
{
        lcdCmd(pos | 0x80);     //设置显示字符的地址
        delay(10);
        lcdDat(data);           //发送要显示的数据
        delay(10);
}

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

```
                    初始化部分——setup函数
*********************************************/
void setup()
{
        //设置引脚4～10为输出，控制1602液晶显示模块
        pinMode(4, OUTPUT);
        pinMode(5, OUTPUT);
        pinMode(6, OUTPUT);
        pinMode(7, OUTPUT);
        pinMode(8, OUTPUT);
        pinMode(9, OUTPUT);
        pinMode(10, OUTPUT);
        //初始化1602液晶显示模块
        lcdInit();
}

/*********************************************
                    执行部分——loop函数
*********************************************/
void loop()
{
        for(int i='0';i<='9';i++)
        {
                    //判断数字是单数还是偶数，决定背光灯点亮
                    //单数背光灯亮，双数背光灯熄灭
                    if(i%2 == 0)
                    digitalWrite(PortBL, LOW);
                    else
                    digitalWrite(PortBL, HIGH);

                    //在左上角（地址0x00）显示数字
                    lcdDisplay( 0 , i );
                    //延时5s
                    delay(5000);
        }

}
```

5.3.7 程序分析

程序中定义了4个子函数，用来完成不同情况下所要发送的数据或命令。lcdDat和lcdCmd是两个基本的函数，通过控制数字I/O口置高或置低的操作完成单条数据或命令的发送，操作时序参见图5.15和图5.16。lcdInit函数用于完成1602液晶模块的初始化，此函数只

执行一次，放在setup函数中。lcdDisplay用于完成在液晶屏的指定位置显示字符，本实例中始终在左上角位置显示数字字符。

5.3.8 Arduino的液晶控制方式

在Arduino中当然不能像上面介绍的那样控制液晶模块显示，否则就不能称作与硬件无关性的程序设计了。Arduino提供了一个LiquidCrystal类，用于字符型液晶模块的控制，在程序中只要声明一个LiquidCrystal类的对象lcd就能够通过调用该类的公有成员函数简单地实现液晶模块的应用。关于LiquidCrystal类会在第6章中进行详细介绍。现在来看看5.3.5节中的应用实例是如何使用LiquidCrystal类实现的。

```
/*************************************************
LCD循环显示0～9实例程序——LiquidCrystal类
  每5s变化一个数字，单数背光灯亮，双数背光灯熄灭
  显示位置：液晶左上角

4位总线
* LCD RS 连接到数字I/O PIN8
* LCD Enable连接到数字I/O PIN9
* LCD D4连接到数字I/O PIN4
* LCD D5连接到数字I/O PIN5
* LCD D6连接到数字I/O PIN6
* LCD D7连接到数字I/O PIN7
* LCD 背光灯控制使用数字I/O PIN10

created 2011
  by Nille
  Email: chenille@126.com

  This example code is in the public domain.
*************************************************/

// 包含LiquidCrystal库头文件
#include <LiquidCrystal.h>

//声明LiquidCrystal类的对象lcd，同时定义使用的I/O口
LiquidCrystal lcd(8,9,4,5,6,7);

/*************************************************
                初始化部分——setup函数
*************************************************/
void setup()
{
```

```
    pinMode(10, OUTPUT);
        // 设置lcd显示的行数和列数
    lcd.begin(16, 2);
}

/*********************************************
        执行部分——loop函数
*********************************************/
void loop()
{
    for(int i=0;i<=9;i++)
    {
        //判断数字是单数还是偶数，决定背光灯点亮
        //单数背光灯亮，双数背光灯熄灭
        if(i%2 == 0)
        digitalWrite(10, LOW);
        else
        digitalWrite(10, HIGH);

        //在左上角（地址0x00）显示数字
        lcd.setCursor(0,0);
        lcd.print(i);
        //延时5s
        delay(5000);
    }
}
```

通过程序可以看出，使用LiquidCrystal类后，显示数字字符的操作只需两步操作：

（1）lcd.setCursor(0,0);

（2）lcd.print(i);

第1步设置光标位置，因为应用实例中数字字符始终显示在左上角，所以函数参数为0,0。第2步负责在光标的位置显示数字字符，此处直接调用print函数就可完成。

提示：*在声明LiquidCrystal类的对象时要设置液晶模块所占用的引脚，参数按先后顺序依次对应液晶模块的RS、E、D4、D5、D6、D7。*

5.3.9 “hello Arduino！”

在LCD Keypad Shield扩展板上还有5个按键，Arduino使用了1个模拟口获取这5个按键的值。本节就使用LiquidCrystal类函数在1602液晶模块的第一行显示字符串“hello Arduino！”，同时响应LCD Keypad Shield扩展模块上的“UP”和“DOWN”键：当字符串在第一行显示时，按下“DOWN”键，字符串会移动到第二行；当字符串在第二行显示时，

按下“UP”键，字符串会移动到第一行。

程序代码如下：

```
/*********************************************
  hello Arduino实例程序

  在1602液晶模块上第一行显示字符串“hello Arduino! ”
同时响应LCD Keypad Shield扩展模块上的“UP”和“DOWN”键：
当字符串在第一行显示时，按下“DOWN”键，字符串会移动到第二行；
当字符串在第二行显示时，按下“UP”键，字符串会移动到第一行

 4位总线
 * LCD RS 连接到数字I/O PIN8
 * LCD Enable连接到数字I/O PIN9
 * LCD D4连接到数字I/O PIN4
 * LCD D5连接到数字I/O PIN5
 * LCD D6连接到数字I/O PIN6
 * LCD D7连接到数字I/O PIN7
 * LCD 背光灯控制使用数字I/O PIN10

       created 2011
  by Nille
  Email: chenille@126.com

  This example code is in the public domain.
 *********************************************/

// 包含LiquidCrystal库头文件
#include <LiquidCrystal.h>

//声明LiquidCrystal类的对象lcd，同时定义使用的I/O口
LiquidCrystal lcd(8,9,4,5,6,7);
//设置变量keyVal用于保存按键模拟量采集值
int keyVal;

/*********************************************
                初始化部分——setup函数
*********************************************/
void setup()
{
  // 设置lcd显示的行数和列数
  lcd.begin(16, 2);
  // 在第一行显示字符串 “hello, Arduino! ”
  lcd.print("hello, Arduino!");
```

```
}

/*************************************************
                执行部分——loop函数
*************************************************/
void loop()
{
      keyVal = analogRead(0);
      //判断按键是否为UP，按下UP按键时，模拟量采样值在120～140内
      if(( 120 < keyVal ) && ( keyVal < 140))
  {
                //延时10ms去抖
                delay(10);
                //再进行一次模拟量采集
                keyVal = analogRead(0);
                if(( 120 < keyVal ) && ( keyVal < 140))
                {
                        //如果二次采用值仍在120～140内
                        //则确定按键为UP
                        //清除屏幕
                        lcd.clear();
                        //设定显示位置为第1行第1列
                        lcd.setCursor(0,0);
                        //重新显示字符串
                        lcd.print("hello, Arduino!");
                }
      }
      else if(( 300 < keyVal ) && ( keyVal < 310))
  {
                //判断按键是否为DOWN
                //按下DOWN按键时，模拟量采样值在300～310内

                //延时10ms去抖
                delay(10);
                //再进行一次模拟量采集
                keyVal = analogRead(0);
                if(( 300 < keyVal ) && ( keyVal < 310))
                {
                        //如果二次采用值仍在300～310内
                        //则确定按键为DOWN
                        //清除屏幕
                        lcd.clear();
                        //设定显示位置为第2行第1列
```

```
                lcd.setCursor(0,1);
                //重新显示字符串
                lcd.print("hello, Arduino!");
            }
        }
        //延时0.5s，0.5s后在进行按键模拟量采集
        delay(500);
    }
```

5.4　Ethernet Shield

Ethernet Shield是一块以W5100为核心的网络扩展模块。EthernetShield可以使Arduino成为简单的Web服务器或者通过网络控制Arduino读写数字I/O和模拟I/O等。扩展板采用了可堆叠的设计，还同时支持SD卡读/写，实物如图5.18所示。

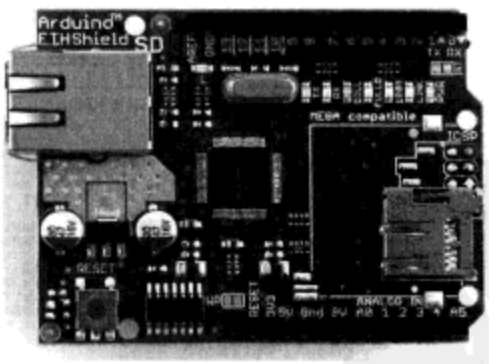


图5.18　Ethernet Shield

5.4.1　Ethernet Shield原理图

通过Ethernet Shield原理图（见图5.19）可以看出Ethernet和SD卡共用一个SPI接口，占用Arduino的引脚10（SPI接口的/SS）、引脚11（SPI接口的MOSI）、引脚12（SPI接口的MISO）、引脚13（SPI接口的SCK）、引脚2（W5100的外部中断）、引脚3（W5100的片选）和引脚4（SD卡的片选），同一时间内Ethernet Shield扩展板只能启用Ethernet和SD卡中的一个功能。本节重点介绍网络的控制和应用。

5.4.2　W5100芯片介绍

W5100是一款多功能的单片网络接口芯片，内部集成有 10/100以太网控制器，主要应用于高集成、高稳定、高性能和低成本的嵌入式系统中。使用W5100可以实现没有操作系统的Internet连接，与IEEE802.3 10Base-T以及802.3u 100BASE-TX兼容。

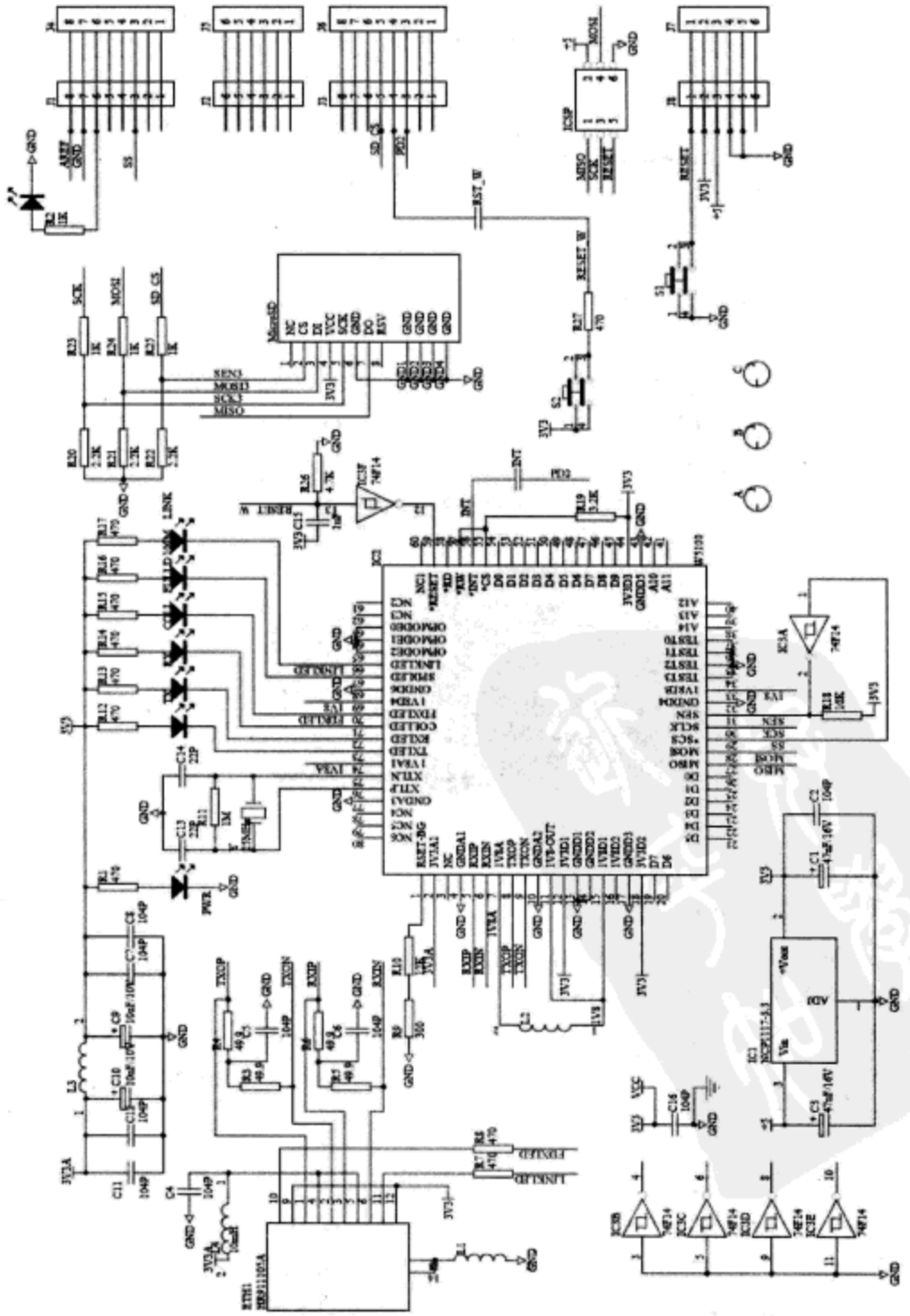


图5.19 Ethernet Shield原理图

W5100内部集成了全硬件的且经过多年市场验证的TCP/IP协议栈、以太网介质传输层（MAC）和物理层（PHY）。硬件TCP/IP协议栈支持TCP、UDP、IPv4、ICMP、ARP、IGMP和PPPoE。使用W5100不需要考虑以太网的控制，只需要进行简单的端口控制。W5100主要有以下特点：

- ❑ 支持硬件TCP/IP协议。
- ❑ 内嵌10BaseT/100BaseTX以太网物理层。
- ❑ 支持自动通信握手（全双工和半双工）。
- ❑ 支持自动MDI/MDX，自动校正信号极性。
- ❑ 支持ADSL连接。
- ❑ 支持4个独立端口同时运行。
- ❑ 不支持IP的分片处理。
- ❑ 内部16KB存储器用于数据发送/接收缓存。
- ❑ 0.18μm CMOS工艺。
- ❑ 3.3V工作电压，I/O可承受5V电压。
- ❑ 环保无铅封装。
- ❑ 支持SPI接口（SPI模式0）。
- ❑ 多功能LED信号输出（TX、RX、全双工/半双工、地址冲突、连接、速度等）。

W5100管脚定义见图5.20，接口信号分类见表5.4～表5.8。

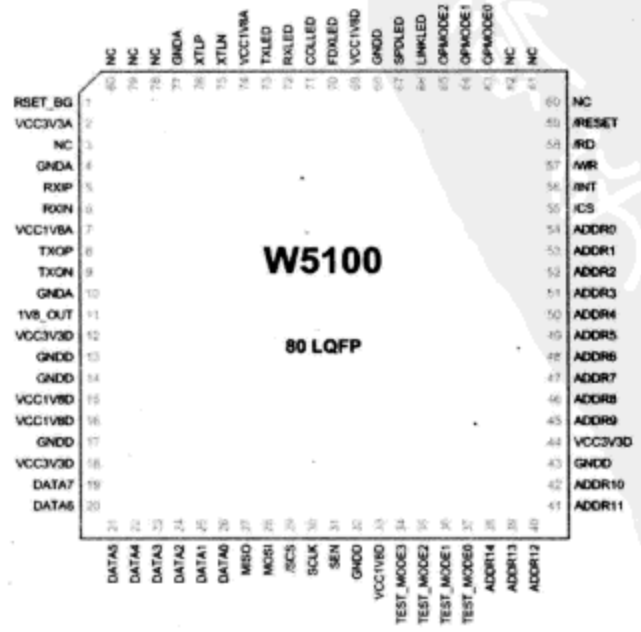


图5.20 W5100管脚定义

表5.4 W5100控制脚接口信号

符 号	管 脚	I/O	说 明
/RESET	59	I	复位，低电平有效 低电平持续时间不小于2μs
ADDR[14～0]	38，39，40，41，42，45，46，47，48，49，50，51，52，53，54	I	地址总线 地址总线内部下拉为低电平
DATA[7～0]	19，20，21，22，23，24，25，26	I/O	数据总线
/CS	55	I	片选，低电平有效
/INT	56	O	中断输出，低电平有效 当W5100在端口产生连接、断开、接收数据、数据发送完成以及通信超时等条件下，输出信号给控制器
/WR	57	I	写使能，低电平有效 控制器发出信号写W5100内部寄存器或存储器，访问地址由ADDR[14～0]选择，数据在该信号的上升沿锁存到W5100
/RD	58	I	读使能，低电平有效 访问地址由ADDR[14～0]选择
SEN	31	I	SPI接口使能 高电平使用SPI模式
SCLK	30	I	SPI时钟 该引脚用于SPI时钟输入
/SCS	29	I	SPI从模式选择，低电平有效
MOSI	28	I	SPI的MOSI信号
MISO	27	O	SPI的MISO信号

表5.5 W5100以太网物理层信号

符 号	管 脚	I/O	说 明
RXIP	5	I	RXIP/RXIN信号组
RXIN	6	I	在RXIP/RXIN信号组接收到从介质传输来的差分数据信号
TXOP	8	O	TXOP/TXON信号组
TXON	9	O	通过TXOP/TXON信号组向介质传输差分数据信号
RSET_BG	1	O	物理层片外电阻 连接一个12.3kΩ的电阻到地
OPMODE[2－0]	63、64、65	I	运行控制模式 [2:0]——描述 000——自动握手 001——100BASE-TX FDX/HDX自动握手 010——10BASE-T FDX/HDX自动握手

（续）

符 号	管 脚	I/O	说 明
			011——保留 100——手动选择 100BASE-TX FDX 101——手动选择 100BASE-TX HDX 110——手动选择 10BASE-T FDX 111——手动选择 10BASE-T HDX

表5.6 W5100电源接口

符 号	管 脚	I/O	说 明
VCC3V3A	2	P	3.3V模拟系统电源
VCC3V3D	12，18，44	P	3.3V数字系统电源
VCC1V8A	7，74	P	1.8V模拟系统电源
VCC1V8D	15，16，33，69	P	1.8V数字系统电源
GNDA	4，10，77	P	模拟电源地
GNDD	13，14，17，32	P	数字电源地
V18	11	O	1.8V电压输出

表5.7 W5100时钟信号

符 号	管 脚	I/O	说 明
XTLP	76	I	外接25MHz晶振以稳定内部振荡电路，如果使用外部振
XTLN	75	I	荡信号，信号连接到XTLN，而XTLP断开

表5.8 W5100 LED信号

符 号	管 脚	I/O	说 明
LINKLED	66	O	低电平表示10/100M连接状态正常在TX/RX状态时闪烁
SPDLED	67	O	低电平表示连接速度为100Mbps
FDXLED	70	O	低电平表示全双工模式
COLLED	71	O	低电平表示网络IP地址冲突
RXLED	72	O	低电平表示当前接收数据
TXLED	73	O	低电平表示当前发送数据

5.4.3 W5100芯片的寄存器

W5100芯片的控制实质上是对片内的寄存器和存储器进行设置和读写。W5100内含公共寄存器、端口寄存器、发送存储器以及接收存储器，如图5.21所示。

图5.21 W5100内的寄存器、存储器

公共寄存器详细信息如表5.9所示：

表5.9 公共寄存器

地 址	寄 存 器	说 明
0x0000	MR（模式）	该寄存器用于软件复位、Ping关闭模式、PPPoE模式以及间接总线接口 bit7：RST，软件复位，置1芯片复位 bit4：PB，Ping阻止模式，置1阻止Ping bit3：PPPoE模式，置1打开PPPoE模式 bit1：AI，间接总线接口模式下地址自动增加 bit0：IND，间接总线接口模式，置1启动
0x0001	GAR0（网关地址0）	
0x0002	GAR1（网关地址1）	网关地址为： GAR0. GAR1. GAR2. GAR3
0x0003	GAR2（网关地址2）	
0x0004	GAR3（网关地址3）	
0x0005	SUBR0（子网掩码地址0）	
0x0006	SUBR1（子网掩码地址1）	子网掩码地址为： SUBR0. SUBR1. SUBR2. SUBR3
0x0007	SUBR2（子网掩码地址2）	
0x0008	SUBR3（子网掩码地址3）	
0x0009	SHAR0（本机硬件地址0）	
0x000A	SHAR1（本机硬件地址1）	
0x000B	SHAR2（本机硬件地址2）	MAC地址为： SHAR0-SHAR1-SHAR2-SHAR3-SHAR4-SHAR5
0x000C	SHAR3（本机硬件地址3）	
0x000D	SHAR4（本机硬件地址4）	
0x000E	SHAR5（本机硬件地址5）	

（续）

地 址	寄 存 器	说 明
0x000F	SIPR0（本机IP地址0）	
0x0010	SIPR1（本机IP地址1）	本机IP地址为：
0x0011	SIPR2（本机IP地址2）	SIPR0.SIPR1.SIPR2.SIPR3
0x0012	SIPR3（本机IP地址3）	
0x0015	IR（中断）	bit7：CONFLICT，IP地址冲突 bit6：UNREACH，无法到达地址 bit5：PPPoE，PPPoE连接关闭 bit3：S3_INT，端口3中断 Bit2：S2_INT，端口2中断 Bit1：S1_INT，端口1中断 Bit0：S0_INT，端口0中断
0x0016	IMR（中断屏蔽）	该寄存器用来屏蔽中断源，每个中断屏蔽位对应中断寄存器（IR）中的一个位，置0屏蔽中断
0x0017	RTR0（重发时间0）	该寄存器用来设置溢出的时间，每一个单位数值为100μs。RTR0为高位，RTR1为低位
0x0018	RTR1（重发时间1）	
0x0019	RCR（重发计数）	该寄存器内的数值设定可重发的次数
0x001A	RMSR（接收存储器大小）	该寄存器配置全部8KB的RX存储空间到各指定端口 bit7、bit6配置端口3 00(1KB)，01(2KB)，10(4KB)，11(8KB) Bit5、bit4配置端口2 Bit3、bit2配置端口1 Bit1、bit0配置端口0
0x001B	TMSR（发送存储器大小）	寄存器用来将8KB的发送存储区分配给每个端口
0x001C	PATR0（PPPoE认证类型0）	在与PPPoE服务器连接时，该寄存器指示已通过的安全认证方法。W5100只支持两种安全认证类型：0xC023（PAP）和0xC223（CHAP）
0x001D	PATR1（PPPoE认证类型1）	
0x0028	PTIMER（PPP LCP请求定时器）	该寄存器表示发出LCP Echo所需要的时间间隔
0x0029	PMAGIC（PPP LCP魔数值）	该寄存器用于LCP握手时采用的魔数选项
0x002A	UIPR0（不能达到IP地址0）	在UDP数据传输时，如果目的IP地址不存在，将会收到一个ICMP数据包。这种情况下，无法到达的IP地址及端口号将存储在UIPR和UPORT中
0x002B	UIPR1（不能达到IP地址1）	
0x002C	UIPR2（不能达到IP地址2）	
0x002D	UIPR3（不能达到IP地址3）	
0x002E	UPORT0（不能达到端口地址0）	
0x002F	UPORT1（不能达到端口地址1）	

端口寄存器详细信息如表5.10（以端口0位例）所示。

表5.10 端口寄存器详细信息

地 址	寄 存 器	说 明
0x0400	S0_MR（端口0模式）	该寄存器设置相应端口的选项或协议类型
0x0401	S0_CR（端口0命令）	该寄存器用来设置端口的初始化、关闭、建立连接、断开连接、数据传输以及命令接收等。命令执行后，寄存器的值自动清零
0x0402	S0_IR（端口0中断）	该寄存器指示建立和终止连接、接收数据、发送完成以及时间溢出等信息
0x0403	S0_SR（端口0状态）	该寄存器指示端口的状态数值
0x0404 0x0405	S0_PORT0（端口0端口号0） S0_PORT1（端口0端口号1）	该寄存器在TCP或UDP模式下设定对应端口的端口号，这些端口号必须在进行OPEN指令之前完成
0x0406	S0_DHAR0（端口0目的物理地址0）	该寄存器设置端口的目的物理地址
0x0407	S0_DHAR1（端口0目的物理地址1）	
0x0408	S0_DHAR2（端口0目的物理地址2）	
0x0409	S0_DHAR3（端口0目的物理地址3）	
0x040A	S0_DHAR4（端口0目的物理地址4）	
0x040B	S0_DHAR5（端口0目的物理地址5）	
0x040C	S0_DIPR0（端口0目的IP地址0）	该寄存器设置端口的目的IP地址
0x040D	S0_DIPR1（端口0目的IP地址1）	
0x040E	S0_DIPR2（端口0目的IP地址2）	
0x040F	S0_DIPR3（端口0目的IP地址3）	
0x0410	S0_DPORT0（端口0目的端口号0）	该寄存器设置端口的目的端口号
0x0411	S0_DPORT1（端口0目的端口号1）	
0x0412	S0_MSSR0（端口0最大分片字节数0）	该寄存器设置端口的最大分片字节数
0x0413	S0_MSSR1（端口0最大分片字节数1）	
0x0414	S0_PROTO（IP RAW模式下端口0的协议）	该寄存器设置端口的IP RAW模式下的协议
0x0415	S0_TOS（端口0的IP TOS）	该寄存器设置端口的IP TOS
0x0416	S0_TTL（端口0的数据包生存期）	该寄存器设置端口的数据包生存期
0x0420	S0_TX_FSR0（端口0的发送存储器剩余空间0）	该寄存器指示端口发送数据缓存区大小
0x0421	S0_TX_FSR1(端口0的发送存储器剩余空间1)	
0x0422	S0_TX_RD0（端口0的发送存储器读指针0）	该寄存器指示端口发送过程完成后的读地址信息
0x0423	S0_TX_RD1（端口0的发送存储器读指针1）	
0x0424	S0_TX_WR0（端口0的发送存储器写指针0）	该寄存器指示端口发送地址信息
0x0425	S0_TX_WR1（端口0的发送存储器写指针1）	
0x0426	S0_RX_RSR0（端口0的接收数据大小0）	该寄存器指示端口接收数据缓存区中接收数据的字节数
0x0427	S0_RX_RSR1（端口0的接收数据大小1）	
0x0428	S0_RX_RD0（端口0的接收存储器读指针0）	该寄存器指示端口接收过程完成后的读地址信息
0x0429	S0_RX_RD1（端口0的接收存储器读指针1）	

5.4.4 W5100芯片的使用

Ethernet Shield扩展板通过SPI接口实现对W5100的控制。SPI协议定义了4种操作模式，W5100使用的是模式0。W5100使用读和写两种操作代码，其他的操作代码均不响应。

在SPI模式下，W5100使用“完整32位数据流”，包括1个字节的操作码、2个字节的地址和1个字节的数据。操作码、地址和数据字节传输都是高位在前，低位在后。数据格式如表5.11所示，时序图如图5.22所示。

表5.11 W5100 SPI数据格式

命 令	操 作 码	地 址	数 据
写操作	0xF0	2字节	1字节
读操作	0x0F	2字节	1字节

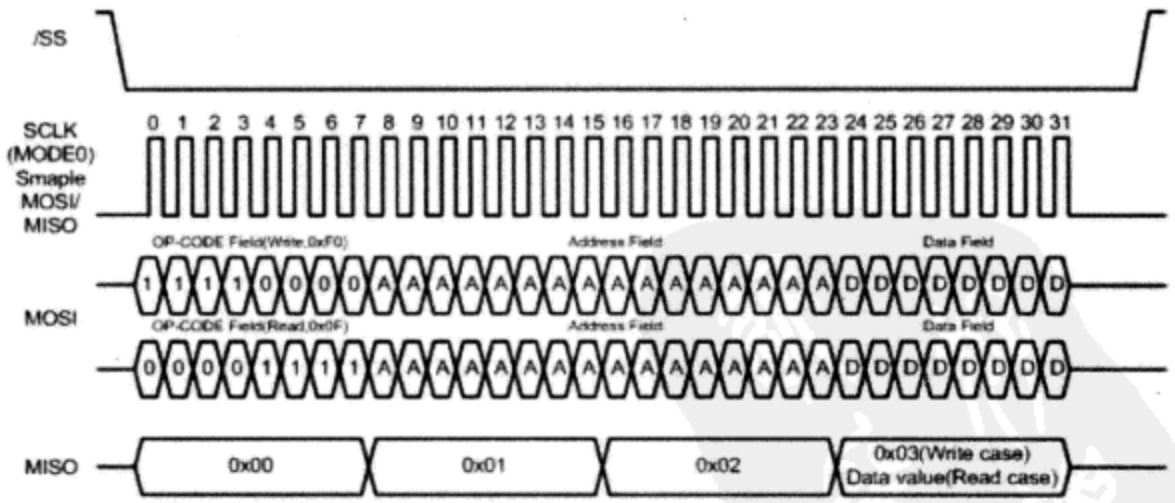


图5.22 完整32位数据流时序图

5.4.5 Ethernet Shield应用实例

本节将Arduino控制板、Ethernet Shield扩展板、Input Shield扩展板（见5.2节）按照对应关系，并且利用扩展板堆叠设计结构插接在一起。其中Input Shield扩展板在最上层，Arduino控制板在最下层。程序的通信模式设置为服务器模式，硬件IP地址为192.168.0.123，通过浏览器访问http://192.168.0.123，以获取Input Shield扩展板上摇杆的水平方向、垂直方向的模拟量数值。

W5100作为服务器模式的工作流程如图5.23所示。

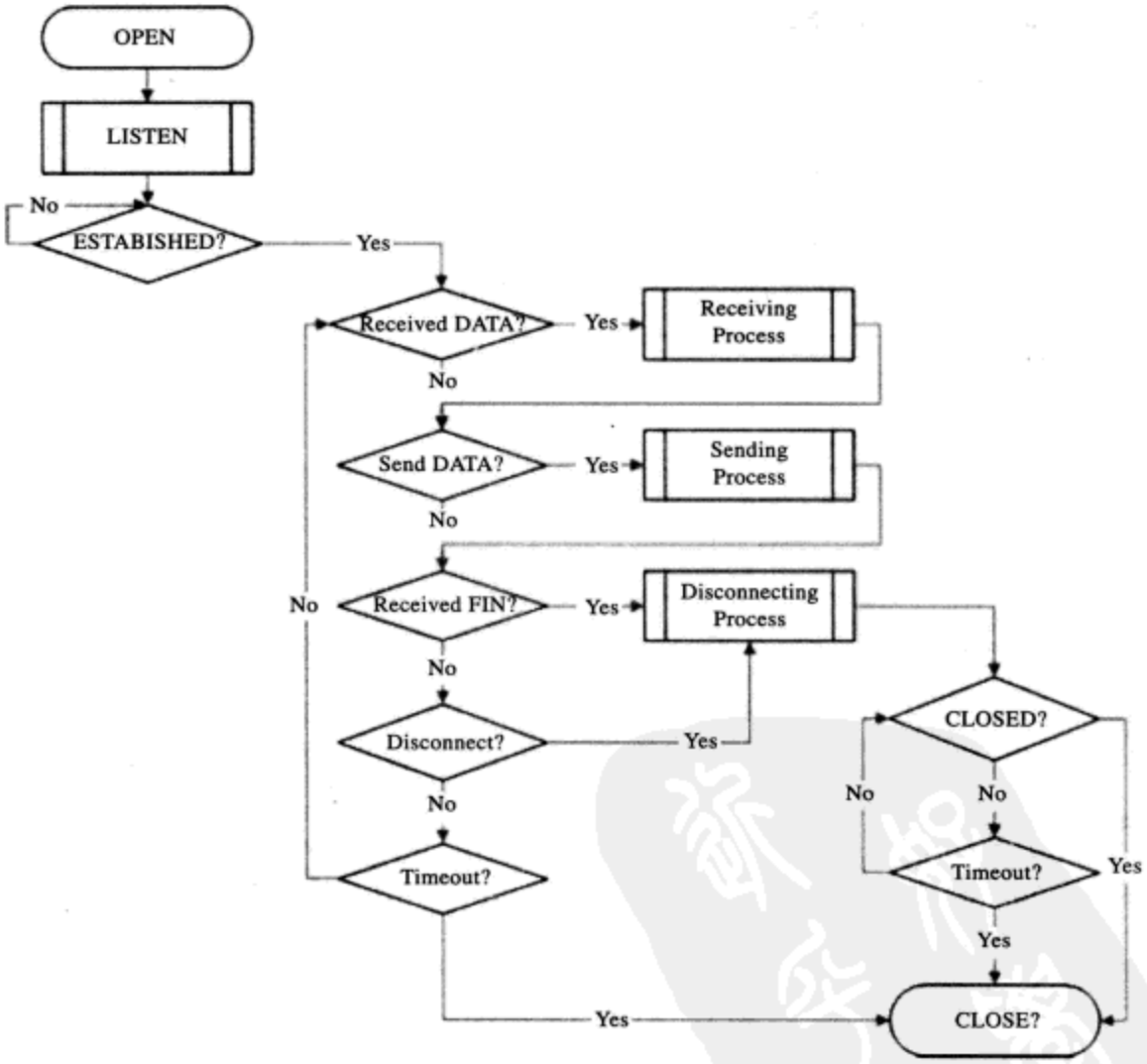


图5.23 W5100作为服务器模式的工作流程

5.4.6 程序设计

同样，在Arduino中网络的应用相对也简单得多。Arduino提供了一个Ethernet库，用于网络的应用。关于Ethernet库会在第6章中详细介绍，这里我们直接应用Ethernet类实现5.4.5节的实例，程序代码如下所示：

```
/*********************************************
  网页获取摇杆模拟量值实例程序——Ethernet库

  使用Arduino控制板、Ethernet Shield扩展板、Input Shield扩展板
```

```
  浏览器形式获取摇杆水平方向、垂直方向模拟量数值
  IP地址：192.168.0.123
  MAC地址：0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED
  网关：192.168.0.1
子网掩码：255.255.255.0

      created 2011
  by Nille
  Email: chenille@126.com

  This example code is in the public domain.
 ************************************************/

// 包含SPI库头文件和Ethernet库头文件
#include <SPI.h>
#include <Ethernet.h>

// 定义两个数组用来存储MAC地址和IP地址.
byte mac[] = {0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
byte ip[] = {192,168,0, 123 };
byte gateway[] = {192,168,0, 1};
byte subnet[] = {255, 255, 255, 0};

// 初始化Ethernet server 库(port 80 is default for HTTP):
Server server(80);

/************************************************
              初始化部分——setup函数
************************************************/
void setup()
{
  //SPI接口初始化
      SPI.begin();
  SPI.setBitOrder(MSBFIRST);
  SPI.setDataMode(SPI_MODE0);

      //由于W5100的外部中断使用引脚3，此处将引脚3置为输出，同时输出高电平
      pinMode(3, OUTPUT);
  digitalWrite(3, HIGH);

      // 初始化网络端口
  Ethernet.begin(mac, ip, gateway, subnet);
```

```
  server.begin();
}

/************************************************
                执行部分——loop函数
************************************************/
void loop()
{
  //侦听
Client client = server.available();
  if (client)
{
    // an http request ends with a blank line
    boolean currentLineIsBlank = true;
    while (client.connected()) {
      if (client.available()) {
         char c = client.read();

         if (c == '\n' && currentLineIsBlank)
                {
                // send a standard http response header
                client.println("HTTP/1.1 200 OK");
                client.println("Content-Type: text/html");
                client.println();

                //输出水平方向模拟量数值
                client.print("horizontal:");
                //水平方向使用的是模拟口1
                client.print(analogRead(1));
           //输出一个网页格式中的回车
                client.println("<br />");

           //输出垂直方向模拟量数值
                client.print("vertical:");
                //垂直方向使用的是模拟口0
           client.print(analogRead(0));

                break;
           }

           if (c == '\n')
           {
           // you're starting a new line
```

```
            currentLineIsBlank = true;
          }
          else if (c != '\r')
          {
            // you've gotten a character on the current line
            currentLineIsBlank = false;
          }
        }
      }
      // 延时等待浏览器接收数据
      delay(1);
      // 关闭连接
      client.stop();
    }
}
```

5.5 I/O扩展板

Arduino的I/O扩展板本身没有什么附加的功能，只是将原有的引脚在形式上加以变换，使其可以更灵活地与更多的功能模块直接连接，帮助开发人员减少了在Arduino上添加电路的麻烦。本节介绍两款I/O扩展板——Xbee传感器扩展板V5和Interface shield扩展板，另外针对两款I/O扩展板分别实现对伺服电机（舵机）和TLC全彩LED控制器模块的应用。

5.5.1 Xbee传感器扩展板V5

Xbee传感器扩展板V5主要侧重于一些传感器、无线通信模块的接口扩展，扩展板如图5.24所示，具体接口如下：

- 扩展14个数字I/O口（12个舵机接口）及电源，接口使用VCC、GND以及相应的数字引脚（引脚0和引脚1不能作为舵机控制接口）。
- 6个模拟I/O口及电源，接口使用+5V、GND以及相应的模拟引脚。
- 1个数字端口外接电源接线柱，数字端口外部供电和板载电源自动切换。
- 1个外接电源输入接线柱和1个输入插针。
- RS485接口，使用VCC、GND、数字引脚0、数字引脚1和数字引脚2。
- XBee/Bluetooh Bee蓝牙无线数传接口，使用+3.3V、GND、数字引脚0和数字引脚1。
- APC220/Bluetooh V3蓝牙无线数传接口，使用+5V、GND、数字引脚0和数字引脚1。
- I IC/TWI接口，使用VCC、GND、模拟引脚4和模拟引脚5。

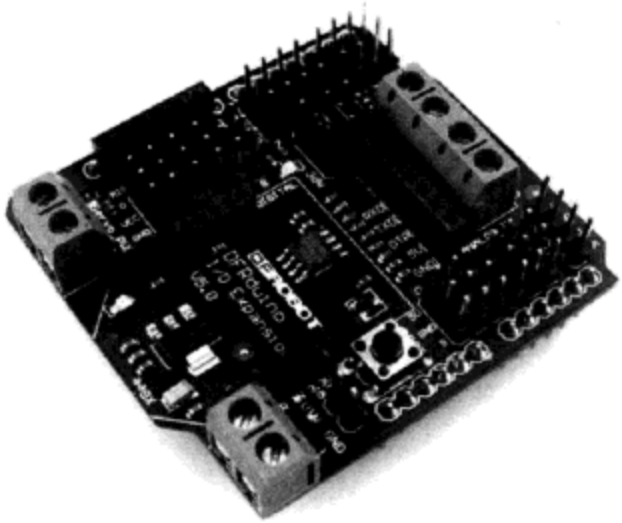


图5.24 XBee传感器扩展板V5

5.5.2 伺服电机控制

伺服电机（舵机）最早出现在航模运动中，主要用于遥控模型的运动姿态，是控制模型动作的动力来源，其工作过程是把所收到的电信号转换成电动机轴上的角位移或角速度输出。目前广泛应用在机器人控制领域中，舵机的输入信号是脉宽变化在1～2ms的PWM信号，而舵机本身也有一个自身的信号源，它产生的脉宽也是1～2ms，但是极性是和输入的1～2ms信号相反。把这两个信号比对，就会出现正差或者是负差，这个差就是左右舵机电机正反转的依据。电机本身还联动一个电位器，这个电位器的变化就改变了自身信号源的脉宽，电机的转动最终会使输入和输出信号等宽，这个时候舵机进入平衡位置，停转。

通用舵机的结构如图5.25所示。

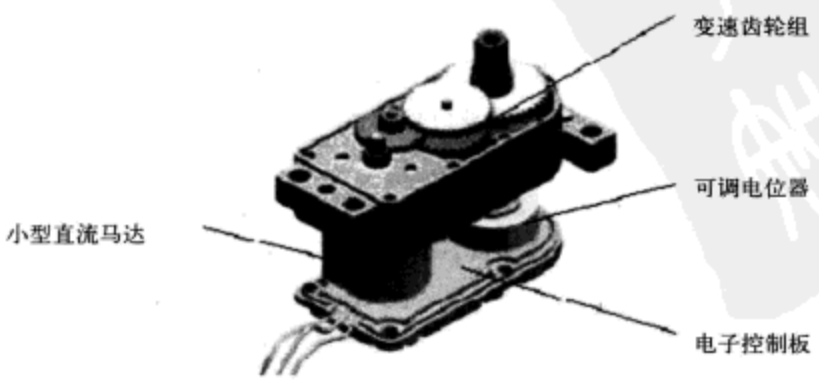


图5.25 舵机的结构及控制信号

标准的舵机有3条控制线，分别是电源、地和信号线。电源和地用于提供内部的直流马达及控制线路所需的能量，电压通常介于4～6V，由于舵机内的马达会产生噪声，所以该电源应尽可能与处理系统的电源隔离。信号线是一个PWM信号，高电平时间通常在1～2ms，低电平时间应在5～20ms，舵机每20ms必须接收到高电平信号，否则舵机将断电，并且不能维持在原来的位置。表5.12展示了一个典型的PWM信号与舵机位置的关系。

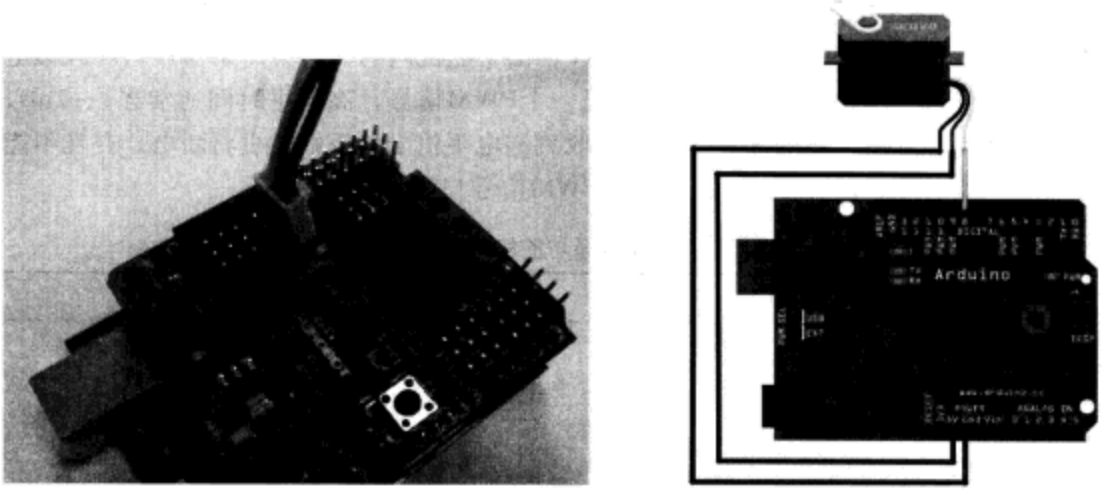
表5.12 PWM信号与舵机位置关系

高电平脉宽	舵机位置
0.5ms	≈−90°
1.0ms	≈−45°
1.5ms	≈0°
2.0ms	≈45°
2.5ms	≈90°

提示：在实际应用中，脉宽与低电平时间要求并不严格，需要根据实际情况调整。

5.5.3 伺服电机应用实例

本节将用Arduino实现计算机对舵机角度的控制，舵机初始位置在0，高电平脉宽为1.5ms，当计算机发送“W”时，舵机角度增加，高电平脉宽增加10μs，同时回复当前高电平脉宽，达到最大角度时，舵机角度不再增加；当计算机发送“S”时，舵机角度减少，高电平脉宽减少10μs，同时回复当前高电平脉宽，达到最小角度时，舵机角度不再减少。舵机占用引脚8。硬件上可以使用XBee传感器扩展板V5将舵机直接插到引脚8的扩展接口上，如图5.26a所示，也可以如图5.26b所示将舵机连接到Arduino控制板上。



a) b)

图5.26 舵机连接示意图

程序代码如下。

```
/*************************************************
伺服电机应用实例程序

舵机初始位置在0，高电平脉宽为1.5ms

当计算机发送“W”时，高电平脉宽增加10μs，同时回复当前高电平脉宽
达到最大角度时，舵机角度不再增加

当计算机发送“S”时，高电平脉宽减少10μs，同时回复当前高电平脉宽
达到最小角度时，舵机角度不再减少

舵机占用引脚8

created 2011
by Nille
Email: chenille@126.com

This example code is in the public domain.
*************************************************/

uint16_t  angle=1500;          //定义初始脉宽值

/*************************************************
                初始化部分———setup函数
```

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

void setup()
{
  //初始化串口，设备通信波特率为9600bps
  Serial.begin(9600);
       //舵机占用引脚8，设为输出
  pinMode(8, OUTPUT);

}

/*************************************************
                执行部分——loop函数
*************************************************/
void loop()
{
        //判断是否收到计算机发送的数据
        if ( Serial.available())
        {
            switch(Serial.read())
            {
                 //收到“W”，高电平脉宽增加10μs
               case 'W':
                  angle += 10;
                  if(angle >2500)
                  angle =2500;// 达到最大角度时，舵机角度不再增加
                  break;

        //收到“S”，高电平脉宽减少10μs
                case 'S':
                  angle -= 10;
                  if(angle <500)
                  angle =500;// 达到最小角度时，舵机角度不再减少
                  break;

                default:
                  break;
             }
             //回复当前高电平脉宽
             Serial.println(angle,DEC);
          }

   //输出高电平
```

```
        digitalWrite(8, HIGH);
        //脉宽为angle
        delayMicroseconds(angle);
        //输出低电平
        digitalWrite(8, LOW);
        //低电平持续15ms
        delay(15);
}
```

5.5.4 Interface shield

Interface shield扩展板侧重于总线接口的扩展，SPI接口、IIC接口、Micro SD卡接口(可直接插Micro SD卡)、SD卡存储模块接口、TLC5940接口等。扩展板如图5.27所示，所占用的引脚在扩展板均有明确的标注。

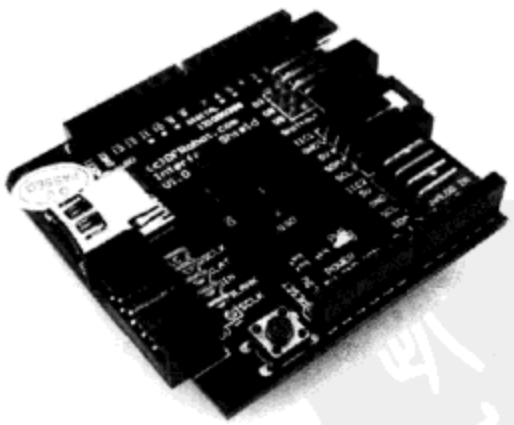


图5.27 Interface shield扩展板

5.5.5 RGB LED Module

RGB LED Module是应用TLC5940接口实现4×4全彩点阵显示的模块。TLC5940接口主要用于控制TI公司的TLC5940NT PWM LED驱动芯片，该芯片支持级联和串口数据传输，除了可应用在全彩LED的控制上，还可用于大量使用舵机的情况下。

RGB LED Module使用3只TI公司的TLC5940NT PWM LED驱动芯片控制，每只芯片控制LED 1种颜色，通过程序控制每只芯片每个引脚PWM的占空比，即可实现4×4全彩点阵显示。该模块可以使用Interface shield扩展板直接连接到Arduino控制板上，同时使用IDC10连接线还可以实现多级级联。模块实物及连接效果图如图5.28所示。

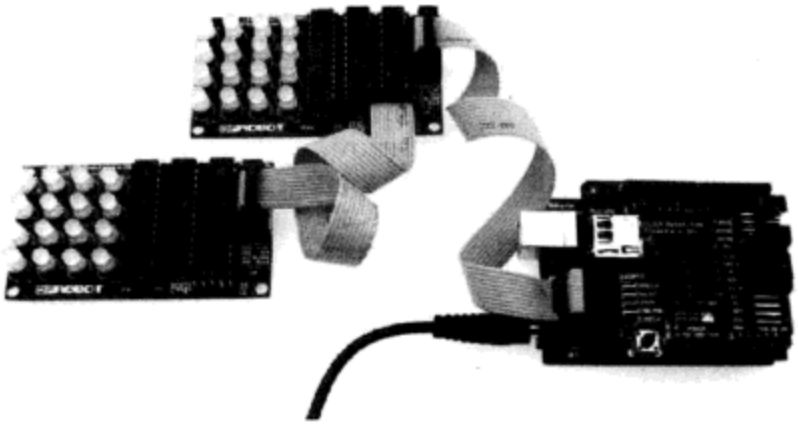


图5.28　TLC全彩LED控制器模块连接效果图

TLC5940NT PWM LED驱动芯片外观示意如图5.29所示，可提供16路PWM输出，最大30MHz数据传输率，芯片引脚定义如表5.13所示，其典型应用电路如图5.30所示。

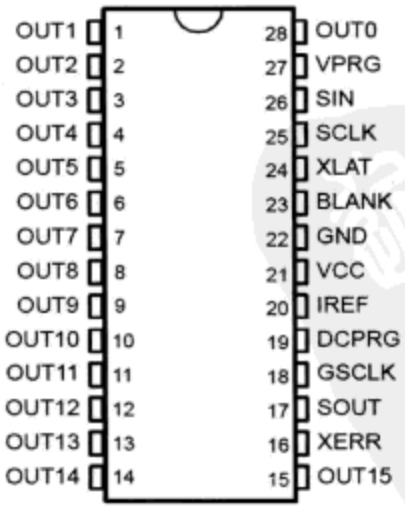


图5.29　TLC5940NT芯片外观

表5.13　TLC5940NT芯片引脚定义

引　脚	符　号	输入/输出	功能描述
28	OUT0	O	16路PWM输出
1～15	OUT1～OUT15		
16	XERR	O	错误输出，可不接
17	SOUT	O	串行数据输出
18	GSCLK	I	PWM控制器参考时钟

（续）

引 脚	符 号	输入/输出	功 能 描 述
19	DCPRG	I	DC数据输入切换，当DCPRG为低时，DC连接到EEPROM；当DCPRG为高时，DC连接到DC寄存器
20	IREF	I	参考电流
21	VCC	P	电源正
22	GND	G	电源地
23	BLANK	I	当BLANK为高时，清空所有PWM输出引脚
24	XLAT	I	锁存信号，当XLAT由高变低时，保存信号
25	SCLK	I	串行数据时钟
26	SIN	I	串行数据输入
27	VPRG	I	芯片功能选择 当VPRG为高时，芯片为DC模式（Dot Correction，点校正模式） 当VPRG为低时，芯片为GS模式（Grayscale PWM Control，PWM调节控制功能）

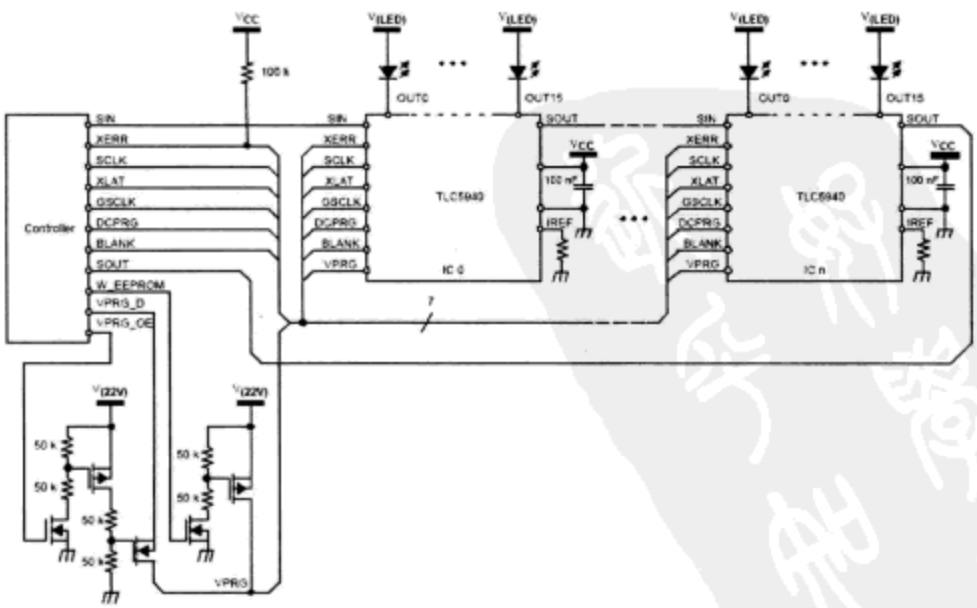


图5.30 TLC5940NT芯片典型应用电路

TLC5940接口占用了Arduino的引脚3（对应GSCLK）、引脚9（对应XLAT）、引脚10（对应BLANK）、引脚11（对应SIN）和引脚13（对应SCLK）。Arduino就通过这些引脚实现对RGB LED Module的控制，具体的接口定义及RGB LED Module电路图如图5.31所示。模块内部将DCPRG拉高、VPRG接地，使芯片应用在PWM调节控制功能。

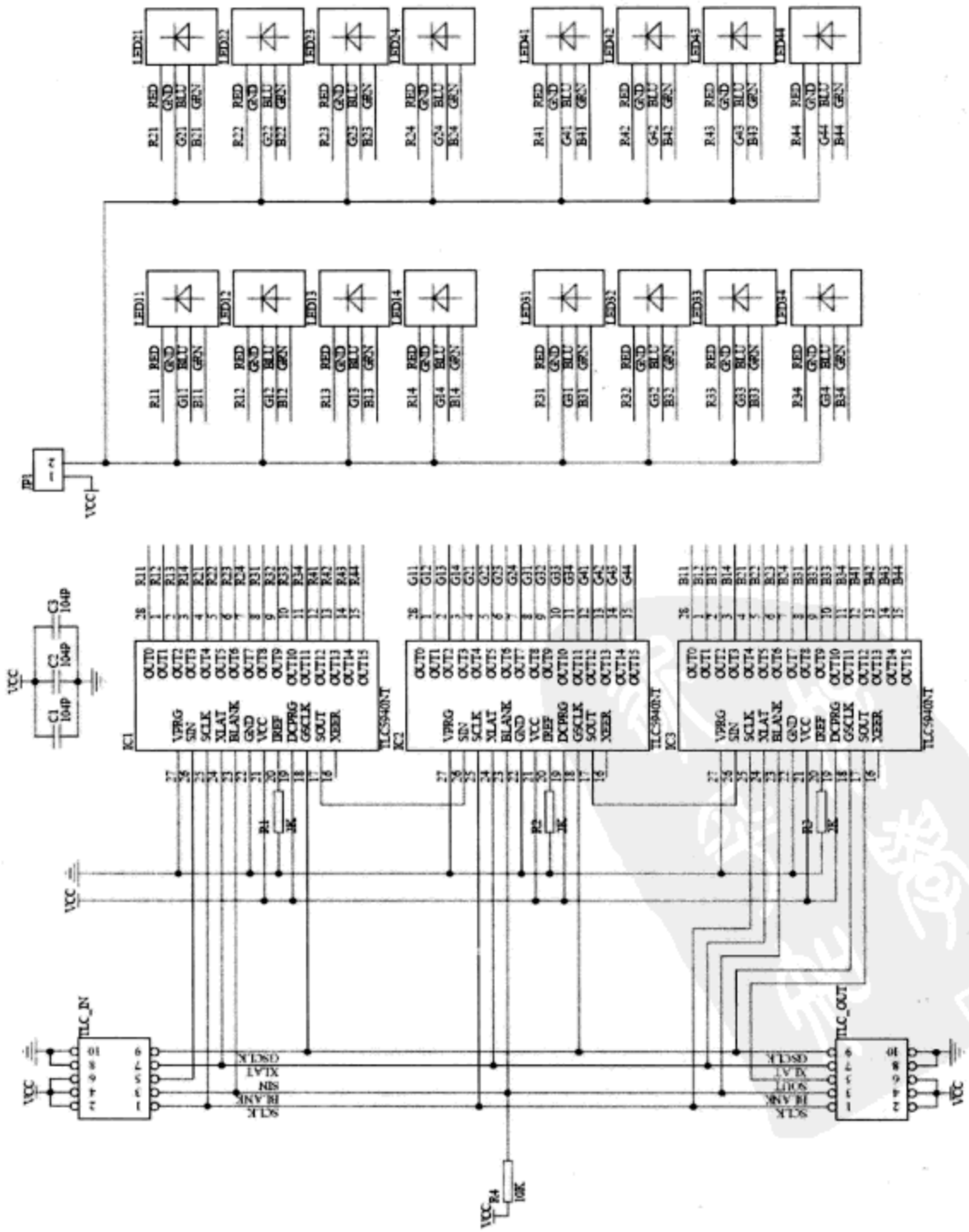


图5.31　RGB LED Module原理图

5.5.6 RGB LED Module应用实例

本节将控制RGB LED Module上16个全彩LED逐个点亮或熄灭，每个LED灯每次点亮时间0.5s。先后显示蓝、绿、红3种颜色。TLC5940NT PWM LED驱动芯片的控制时序如图5.32所示。

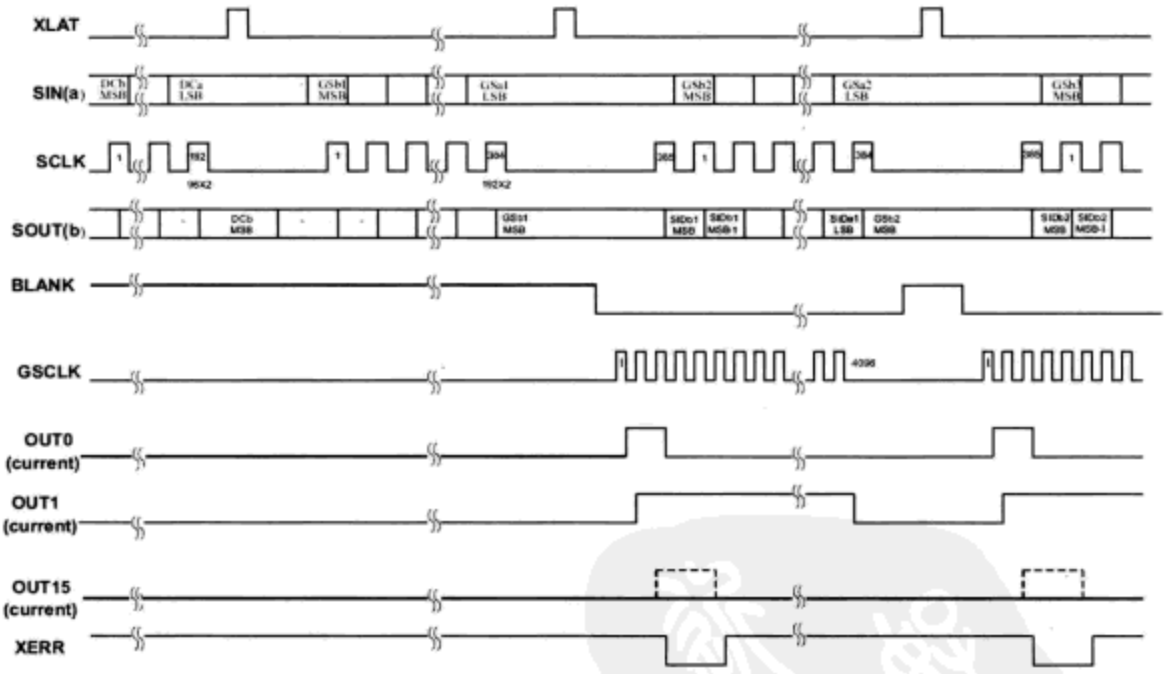


图5.32 TLC5940NT PWM LED驱动芯片的控制时序

TLC5940芯片在BLANK置低时启动数据传输功能，SIN上的信号在SCLK信号的上升沿写入，当所有信号都写入后，通过XLAT的一个高脉冲将数据锁存到内部寄存器，最后拉高BLANK。所有数据均是MSB在前。Arduino依据时序图的控制逻辑对芯片进行操作，实现16个全彩LED逐个点亮或熄灭的效果，程序代码如下：

```
/*********************************************
RGB LED Module上的LED逐个点亮或熄灭实例程序

每个LED灯每次点亮时间0.5s。先后显示蓝、绿、红3种颜色

引脚使用情况：
------------------------
 ARDUINO          13     |-> SCLK (pin 25)
                  12     |
                  11     |-> SIN (pin 26)
```

```
                10    |-> BLANK (pin 23)
                 9    |-> XLAT (pin 24)
                 8    |
                      |
                 7    |
                 6    |
                 5    |
                 4    |
                 3    |-> GSCLK (pin 18)
                 2    |
                 1    |
                 0    |
--------------------

TLC5940引脚定义:
        ----------TLC5940---------
   OUT1    |1                28    | OUT0
   OUT2    |2                27    |-> VPRG (GND)
   OUT3    |3                26    |-> SIN (pin 11)
   OUT4    |4                25    |-> SCLK (pin 13)
   OUT5    |5                24    |-> XLAT (pin 9)
   OUT6    |6                23    |-> BLANK (pin 10)
   OUT7    |7                22    |-> GND
   OUT8    |8                21    |-> VCC (+5V)
   OUT9    |9                20    |-> IREF
  OUT10    |10               19    |-> DCPRG (+5V)
  OUT11    |11               18    |-> GSCLK (pin 3)
  OUT12    |12               17    |-> SOUT
  OUT13    |13               16    |-> XERR
  OUT14    |14               15    | OUT15
        ---------------------------

created 2011
by Nille
Email: chenille@126.com
*******************************************/

#define     GSCLK     3          //GSCLK占用引脚3
#define     XLAT      9          //XLAT占用引脚9
#define     BLANK     10         //BLANK占用引脚10
#define     SCLK      13         //SCLK占用引脚13
#define     SIN       11         //SIN占用引脚11
```

```
/*************************************************
 XLAT产生一个脉冲
函数功能：XLAT产生一个脉冲，用于锁存一组数据
入口参数：无
出口参数：无
*************************************************/
void XLATPulses()
{
        digitalWrite(XLAT, HIGH);        //XLAT置高
        delayMicroseconds(10);
        digitalWrite(XLAT, LOW);         //XLAT置低
        delayMicroseconds(10);
}


/*************************************************
 SCLK产生一个脉冲
函数功能：SCLK产生一个脉冲，用于传输1bit数据
入口参数：无
出口参数：无
*************************************************/
void SCLKPulses()
{
        digitalWrite(SCLK, HIGH);        //SCLK置高
        delayMicroseconds(10);
        digitalWrite(SCLK, LOW);         //SCLK置低
        delayMicroseconds(10);
}

/*************************************************
 点亮一个通道的LED灯
函数功能：点亮指定通道的LED灯
入口参数：channel——通道值，由于一片TLC5940有16个通道
          通道值为0～15，所以参数范围值为0～16
          参数为16时表示熄灭所有通道的LED灯
出口参数：无
*************************************************/
void SetChannel(int channel)
{
    if( channel < 16)          //如果通道值小于16，则点亮指定通道的LED
    {
        //TLC5940的每一个通道为12位的PWM，故总的数据量是16×12bit = 192bit
        //每一个通道含12bit数据
```

```
        for(int i = 0 ; i<192 ; i++)
        {
                    //根据通道值决定每bit数据置高或置低
                    if ( (i>(channel*12) )&&( i < ( (channel+1)*12-1) ) )
                    digitalWrite(SIN, HIGH);
                    else
                    digitalWrite(SIN, LOW);
                    delayMicroseconds(10);

                    //每1bit置位完成后输出一个SCLK脉冲
                    SCLKPulses ();
        }
    }
    else                          //否则熄灭所有通道的LED灯
    {
        for(int i = 0 ; i<192 ; i++)
        {
                    digitalWrite(SIN, LOW);
                    delayMicroseconds(10);

                    SCLKPulses ();
        }
    }
}

/*************************************************
            初始化部分——setup函数
*************************************************/
void setup()
{
  //将控制引脚置位输出
        pinMode(SCLK, OUTPUT);
        pinMode(SIN, OUTPUT);
        pinMode(BLANK, OUTPUT);
        pinMode(XLAT, OUTPUT);

    //依据图5.32 TLC5940NT PWM LED驱动芯片的控制时序图设置引脚初始值
       digitalWrite(BLANK,LOW);
       digitalWrite(XLAT,LOW);
       digitalWrite(SCLK,LOW);

    //GSCLK输出PWM脉冲，用于TLC5940芯片输出PWM信号的参考时钟
        analogWrite(GSCLK,127);
```

```
}


/************************************************
              执行部分——loop函数
************************************************/
void loop()
{
    for(int j = 0 ; j < 16 ; j ++)          //依次点亮16个LED灯
    {
        digitalWrite(BLANK,HIGH);           //BLANK置高，开始传输数据

        // RGB LED Module使用3只TI公司的TLC5940NT芯片
        //每只芯片控制LED 1种颜色，3个芯片的同一个通道共同控制1个全彩LED
        //可以理解为总共有16×3 = 48个通道
        //前16个通道为蓝色
        //中间16个通道为绿色
        //最后16个通道为红色

        SetChannel(j);      //循环点亮蓝色
        SetChannel(16);     //不点亮绿色
        SetChannel(16);     //不点亮红色
        XLATPulses();       //XLAT脉冲

        digitalWrite(BLANK,LOW);            //BLANK置低，数据传输结束
        delay(500);                         //每个LED灯每次点亮时间0.5s
    }
    for(int j = 0 ; j < 16 ; j ++)          //依次点亮16个LED灯
    {
        digitalWrite(BLANK,HIGH);           //BLANK置高，开始传输数据

        SetChannel(16);     //不点亮蓝色
        SetChannel(j);      //循环点亮绿色
        SetChannel(16);     //不点亮红色
        XLATPulses();       //XLAT脉冲

        digitalWrite(BLANK,LOW);            //BLANK置低，数据传输结束
        delay(500);                         //每个LED灯每次点亮时间0.5s
    }
    for(int j = 0 ; j < 16 ; j ++)          //依次点亮16个LED灯
    {
        digitalWrite(BLANK,HIGH);           //BLANK置高，开始传输数据
```

```
        SetChannel(16);                    //不点亮蓝色
        SetChannel(16);                    //不点亮绿色
        SetChannel(j);                     //循环点亮红色
        XLATPulses();                      //XLAT脉冲

        digitalWrite(BLANK,LOW);           //BLANK置低，数据传输结束
        delay(500);                        //每个LED灯每次点亮时间0.5s
    }
}
```

5.5.7 程序的精练

在5.5.6节中提到了可以将RGB LED Module看做一个48通道的LED显示模块，16个LED依次点亮或熄灭，先后显示蓝、绿、红的颜色的过程可以看做48个通道依次点亮或熄灭的过程，依据这个想法对5.5.6节的应用实例的程序进行适当调整，程序代码如下：

```
/*********************************************
RGB LED Module上的LED逐个点亮或熄灭实例程序2

每个LED灯每次点亮时间0.5s。先后显示蓝、绿、红3种颜色

created 2011
by Nille
Email: chenille@126.com
*********************************************/

#define     GSCLK     3          //GSCLK占用引脚3
#define     XLAT      9          //XLAT占用引脚9
#define     BLANK     10         //BLANK占用引脚10
#define     SCLK      13         //SCLK占用引脚13
#define     SIN       11         //SIN占用引脚11


/*************************************************
 XLAT产生一个脉冲
函数功能：XLAT产生一个脉冲，用于锁存一组数据
入口参数：无
出口参数：无
*************************************************/
void XLATPulses()
{
      digitalWrite(XLAT, HIGH);     //XLAT置高
      delayMicroseconds(10);
```

```
    digitalWrite(XLAT, LOW);     //XLAT置低
    delayMicroseconds(10);
}


/***********************************************
 SCLK产生一个脉冲
函数功能：SCLK产生一个脉冲，用于传输1bit数据
入口参数：无
出口参数：无
***********************************************/
void SCLKPulses()
{
        digitalWrite(SCLK, HIGH);     //SCLK置高
        delayMicroseconds(10);
        digitalWrite(SCLK, LOW);        //SCLK置低
        delayMicroseconds(10);
}

/***********************************************
 点亮一个通道的LED灯
函数功能：点亮指定通道的LED灯
入口参数：channel--通道值，参数范围值为0～47
出口参数：无
***********************************************/
void SetChannel(int channel)
{
         //TLC5940的每一个通道为12位的PWM，48个通道的数据量是48×12bit = 576bit
         //每一个通道含12bit数据
         for(int i = 0 ; i<576 ; i++)
         {
             //根据通道值决定每bit数据置高或置低
             if ( (i> (channel*12) ) && ( i < ( (channel+1)*12-1) ) )
             digitalWrite(SIN, HIGH);
             else
             digitalWrite(SIN, LOW);
             delayMicroseconds(10);

             //每1bit置位完成后输出一个SCLK脉冲
             SCLKPulses ();
         }
}
```

```
/*************************************************
            初始化部分——setup函数
*************************************************/
void setup()
{
      //将控制引脚置位输出
      pinMode(SCLK, OUTPUT);
      pinMode(SIN, OUTPUT);
      pinMode(BLANK, OUTPUT);
      pinMode(XLAT, OUTPUT);

      //依据图5.32 TLC5940NT PWM LED驱动芯片的控制时序图设置引脚初始值
      digitalWrite(BLANK,LOW);
      digitalWrite(XLAT,LOW);
      digitalWrite(SCLK,LOW);

      //GSCLK输出PWM脉冲，用于TLC5940芯片输出PWM信号的参考时钟
      analogWrite(GSCLK,127);
}


/*************************************************
            执行部分——loop函数
*************************************************/
void loop()
{
    for(int j = 0 ; j < 48 ; j ++)          //依次点亮48个通道的LED灯
    {
        digitalWrite(BLANK,HIGH);           //BLANK置高，开始传输数据
        SetChannel(j);                      //循环点亮LED
        XLATPulses();                       //XLAT脉冲

        digitalWrite(BLANK,LOW);            //BLANK置低，数据传输结束
        delay(500);                         //每个LED灯每次点亮时间0.5s
    }
}
```

第6章 Arduino的扩展库

从第5章的介绍中我们可以看到Arduino提供了许多扩展库，这些扩展库可以使相关的应用变得十分简单，本章就来介绍一下这些扩展库，看看Arduino还使开发什么硬件变得简单。本章会提供每个函数的原型，这有助于读者了解底层硬件的控制方式，但跳过函数原型并不影响Arduino扩展库的应用，只要掌握每个函数的用法就可以进行Arduino的应用了。

6.1 Arduino扩展库介绍

6.1.1 Arduino扩展库的作用

Arduino的扩展库有很多都是Arduino的爱好者编写的，大家在完成某一功能的扩展库之后本着开源的思想将这些资源放在网络上共享，同时根据其他爱好者的使用情况进行完善。

Arduino的扩展库涉及各个方面，极大地扩展了Arduino的应用领域，本章介绍了一些较常用的扩展库，用在液晶显示、舵机控制、以太网络、步进电机控制等方面，如图6.1所示。其中有些扩展库是建立在其他扩展库的基础上，如Ethernet库、TLC5940库等，这些扩展库会更贴近C++的风格，单片机程序的风格越来越少，让使用者对掌握硬件知识的依赖性越来越小。

总体来说，Arduino的扩展库主要的作用有两点：一是将硬件底层的应用封装起来，比如SPI库、Wire库、EEPROM库；二是针对具体的应用编写相应的库，比如Ethernet库、TLC5940库、XBee库。当然有些库同时体现了这两个作用，比如LiquidCrystal库、Servo库、Stepper库。

6.1.2 Arduino扩展库的应用

由于Arduino完全开源的特点导致了Arduino的扩展库并没有一个统一的规划，许多爱好者根据自己的需要或者外围元器件的特点编写Arduino的扩展库并在网络上交流共享，因此Arduino的开发工具中已包含了一些扩展库，同时也可以在网络上下载其他扩展库使用。下面使用Servo扩展库（伺服电机扩展库）实现对伺服电机的控制，通过这个实例简单说明一

下Arduino扩展库的应用，其他扩展库的应用过程与此相类似。

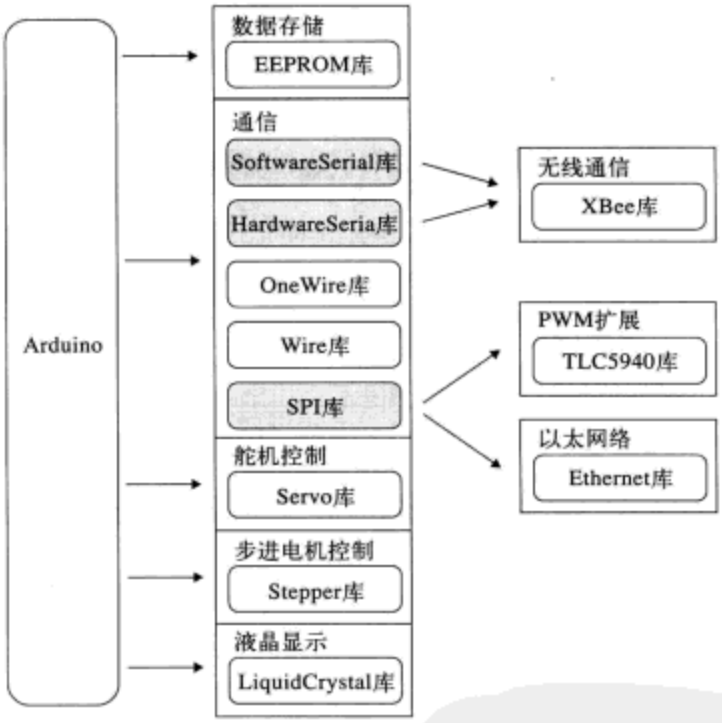


图6.1 本章介绍的Arduino扩展库

首先要确认Arduino开发环境目录下的libraries文件夹包含Servo扩展库，如果是从网络上下载的其他扩展库，也要将扩展库复制到libraries文件夹内，如图6.2所示，在libraries文件夹内包含Servo文件夹。

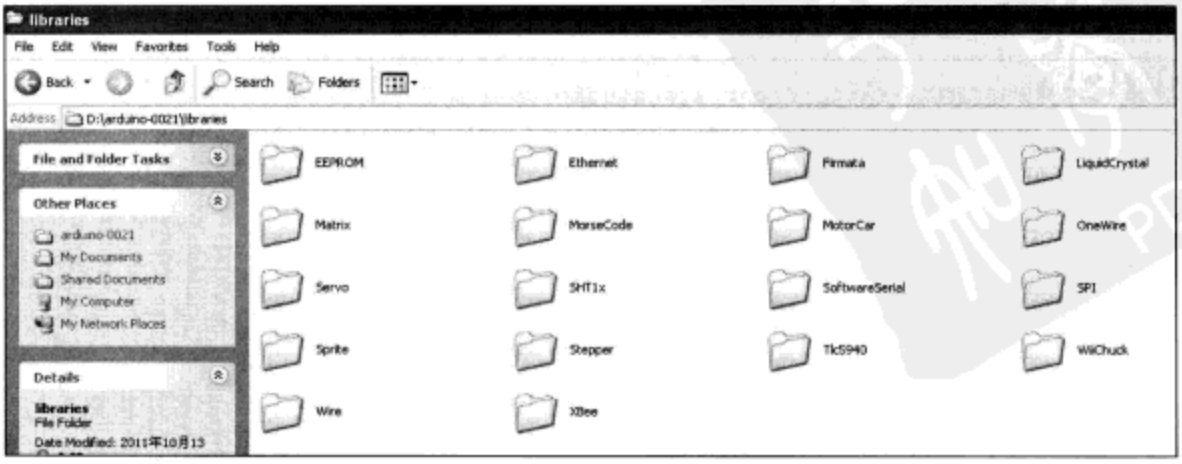


图6.2 Arduino开发环境目录下的libraries文件夹

其次了解扩展库都有哪些成员函数可以使用。Servo扩展库包含attch、write、read、detach等成员函数用于实现对伺服电机的控制功能，具体的成员函数功能及参数信息可以参考6.8节。一般在扩展函数的文件夹内都会有示例程序帮助我们掌握扩展库的成员函数，如图6.3和图6.4所示。

图6.3 Servo扩展库中包含examples文件夹

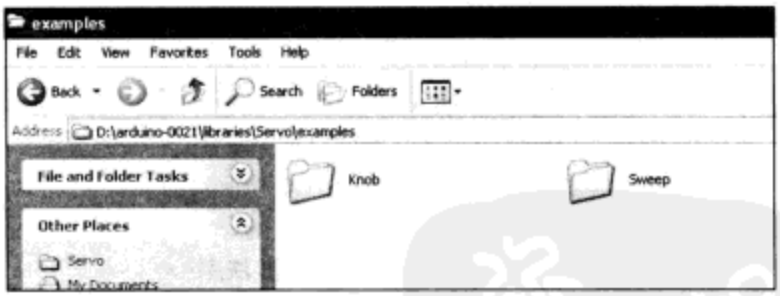


图6.4 examples文件夹下包含两个示例程序

示例程序可以在Arduino的开发环境下打开，这里打开Sweep示例程序进行说明，如图6.5所示。应用扩展库编写程序可以参考扩展库提供的示例程序。

Sweep示例程序代码如下：

```
// Sweep
// by BARRAGAN <http://barraganstudio.com>
// This example code is in the public domain.


#include <Servo.h>

Servo myservo;  // create servo object to control a servo
                // a maximum of eight servo objects can be created

int pos = 0;    // variable to store the servo position

void setup()
```

```
{
  myservo.attach(9);  // attaches the servo on pin 9 to the servo object
}


void loop()
{
  for(pos = 0; pos < 180; pos += 1)// goes from 0 degrees to 180 degrees
  {                                // in steps of 1 degree
    myservo.write(pos);            // tell servo to go to position in variable 'pos'
    delay(15);                     // waits 15ms for the servo to reach the position
  }

  for(pos = 180; pos>=1; pos-=1)// goes from 180 degrees to 0 degrees
  {
    myservo.write(pos);     // tell servo to go to position in variable 'pos'
    delay(15);              // waits 15ms for the servo to reach the position
  }
}
```

图6.5 打开Sweep示例程序

Sweep示例程序主要的功能是控制一个伺服电机的转动角度从0°转到180°，然后再回到0°。在程序的开头使用#include <Servo.h>语句将Servo库包含到程序中，只有在程序中包含了Servo库的头文件（即.h文件）才可以在程序中使用该库。

接下来定义了Servo类的一个对象myservo，在程序中只有通过对象才能够使用库中的成员函数。同时还定义了一个int类型的变量pos用来保存伺服电机的角度值。

在setup()函数中使用了成员函数attach来指定控制伺服电机的引脚，在程序中使用引脚9来控制伺服电机，那么在硬件的连接上就要将伺服电机的信号线连接在Arduino的引脚9上。对象和类的概念以及成员函数的使用方法、格式等内容将在下一节详细介绍。

最后在loop()函数中通过两个for循环以及成员函数write()实现了让一个伺服电机的转动角度从0°转到180°，然后再回到0°的控制。成员函数write()用于设定伺服电机的角度值。

注意：在Arduino的扩展库的使用中，实质上其实只需要注意两点，一是需要在程序开头包含库的头文件；二是需要定义一个类的对象。

可以看出Arduino的这些扩展库都用到了C++中的类，类的学习对于扩展库的应用至关重要，下面我们就从对象和类开始了解Arduino的部分扩展库，最后还会介绍如何创建自己的扩展库。

6.2 对象和类

Arduino的这些库文件都使用了类（class）的概念，类实际上是对现实生活中一类具有共同特征的事物的抽象，是对某种类型的对象定义的变量和方法的原型。类是对某个对象的定义。它包含有关对象动作方式的信息，包括它的名称、方法、属性和事件。但类并不是对象，因为它不存在于内存中。当引用类的代码运行时，就会创建类的一个实例，即对象。一个类可以创建多个对象。

我们可以把类看做“理论上”的对象，它是对象的蓝图，但它在内存中并不存在。从这个蓝图可以创建任何数量的对象。从类创建的所有对象都有相同的成员：属性、方法和事件。但是，每个对象都像一个独立的实体一样动作。类就好比一张楼房的建筑图纸，可以依照一张图纸建造多栋楼房，每栋楼房都是单独的实体，即对象。

6.2.1 类的定义

程序中的对象都具有相对的独立性，其在程序执行过程中生成和删除，还可以一起形成数组、表等结构。为了在程序中创建对象，必须先定义类。类是一组性质相同的对象的描述，它包括一组性质相同的数据和函数，程序运行时用类作为模板来建立对象。这个过程称为类的实例化。类定义的一般格式如下：

```
class 类名
{
      private:
                私有数据成员和成员函数;
      public:
                公有数据成员和成员函数;
      protected:
                保护数据成员和成员函数;
};
```

类定义的关键字是class，类名是一个有效的标识符。类所说明的内容用花括号括起来，以分号作为类说明语句的结束标志。在括号内的内容称为类体。在类体内定义类的成员（包括数据成员和成员函数）。类体内的关键字private、public和protected用于定义类成员的访问权限，在关键字private后的成员称为私有成员，在关键字public后的成员称为公有成员，在关键字protected后的成员称为保护成员。访问权限用于控制对象的某个成员在程序中的可见性，数据成员通常是私有的，成员函数通常有一部分是公有的，另一部分是私有的。

例如，定义关于车（car）的一个类。

```
class car
{
      private:
                int 排量;
                char 品牌;
                int 重量;
                int 经度位置;
                int 纬度位置;
      public:
                void infor () ; //输出车辆信息函数
                void move (int dir) ; //车辆移动函数
};
```

从这个定义的car类中可以看出，它除了包含数据部分以外，还包括对这些数据的操作。类将数据以及数据的操作封装在一起，如infor()函数可输出车辆的信息。除了封装性外，类还具有隐蔽性原则，使类中的成员与外界的联系减少到最低的限度。如car类中的数据成员都是私有的（private），外界不能调用它们，只有通过公有函数infor()函数访问类中的数据。

类成员的访问权限控制实际上就是类的隐藏和封装，访问控制的目的是为了减少出错的可能性。

6.2.2 对象的创建及成员函数的调用

所谓对象就是类的实体，用类来创建对象称为对象的实体化。

如果我们想创建car类的一个对象bmw，操作如下所示：

```
car bmw;
```

对象的创建有点类似于变量的定义，car就像一个特定的数据类型，bmw就相当于变量名。创建了对象bmw后，bmw就可以使用类的成员函数了。成员函数的应用格式是在对象名后加点再加函数名，如下所示：

```
bmw.move(left);
```

6.2.3 对象的初始化和构造函数

在创建一个对象时，常常需要进行一些初始化工作，类中有一个特殊的成员函数——构造函数。构造函数是用于定义的，因此它必须与类名相同，以便系统能够识别它并把它作为构造函数。构造函数可以在对象创建时自动调用，以完成对象的初始化任务。

构造函数是类的特殊成员函数，它的作用是在创建对象时，使用确定的值将所创建的对象初始化，并为新创建的对象分配存储空间。如果类中没有定义构造函数，系统会自动定义默认的构造函数。下例中在car类中加入构造函数，在创建对象时定义位置信息。

```
class car
{
      private:
                int 排量;
                char 品牌;
                int 重量;
                int 经度位置;
                int 纬度位置;
      public:
                car (int posi_x, int posi_y)
                {
                          经度位置 = posi_x;
                          纬度位置 = posi_y;
                }
                void infor () ; //输出车辆信息函数
                void move (int dir) ; //车辆移动函数
};
```

构造函数有如下特点：

- 构造函数的名字必须与类名相同。
- 构造函数无返回类型，包括void类型。
- 构造函数在创建对象时自动调用。
- 构造函数一般为公有函数，其函数体可以在类体内，也可以在类体外。
- 构造函数与普通函数一样，可以有一个或多个参数。
- 对于类，如果没有定义构造函数，系统会自动定义默认的构造函数。

6.2.4 函数的重载

所谓函数重载是指同一个函数名可以对应多个函数的实现。每种实现对应一个函数体，这些函数的名字相同，但是函数的参数类型不同。这就是函数重载的概念。函数重载在类和对象的应用中尤其重要。一般一个类中会定义多个构造函数，这些构造函数的参数类型不同，使类能够在多种情况下创建对象。

下例中在car类中再加入一个构造函数，用于含有重量信息的对象的创建。

```
class car
{
      private:
                int 排量;
                char 品牌;
                int 重量;
                int 经度位置;
                int 纬度位置;
      public:
                car (int posi_x, int posi_y)
                {
                          经度位置 = posi_x;
                          纬度位置 = posi_y;
                }
                car (int posi_x, int posi_y, int weight)
                {
                          经度位置 = posi_x;
                          纬度位置 = posi_y;
                          重量 = weight;
                }
                void infor () ; //输出车辆信息函数
                void move (int dir) ; //车辆移动函数
};
```

6.2.5 析构函数

析构函数也是类的一个特殊的成员函数，当对象撤销释放一个对象时，在对象删除之前，系统自动执行析构函数来做一些系统清理工作。析构函数的功能与构造函数正好相反。

析构函数的名字也同类名一致，但前面要加一个“~”符号，以区别于构造函数。析构函数同样无参数，无返回类型。如果类中没有定义析构函数，系统会自动提供默认的析构函数。

析构函数具有以下特点：

❑ 析构函数的名字必须与类名相同，但要在前面加一个“~”符号，以区别于构造函数。

- 析构函数一般也为公有函数，其函数体可以在类体内，也可以在类体外。
- 一个类只能有一个析构函数。
- 析构函数可以被程序调用，也可以被系统自动调用。
- 如果类中没有定义析构函数，系统会自动提供默认的析构函数。

注意：类中还有许多复杂的概念，如继承性、多态性，这里不再赘述，有兴趣的读者可以参考相关的书籍。

6.3 LiquidCrystal库

LiquidCrystal库是针对日立HD44780芯片组（或兼容的芯片组）驱动的液晶显示器而推出的扩展库，第5章中的LCD Keypad Shield液晶显示扩展板上使用的就是与日立HD44780兼容的芯片组驱动的液晶显示器，LiquidCrystal扩展库可支持4位或8位总线控制方式，库中定义了一个LiquidCrystal类。LiquidCrystal类的具体定义可以参看Arduino开发环境目录下libraries文件夹内的LiquidCrystal.h文件，该文件包含一些常数的宏定义和类的定义，内容如下：

```
/********************************************************
                    宏定义部分
********************************************************/
// commands
#define LCD_CLEARDISPLAY 0x01
#define LCD_RETURNHOME 0x02
#define LCD_ENTRYMODESET 0x04
#define LCD_DISPLAYCONTROL 0x08
#define LCD_CURSORSHIFT 0x10
#define LCD_FUNCTIONSET 0x20
#define LCD_SETCGRAMADDR 0x40
#define LCD_SETDDRAMADDR 0x80

// flags for display entry mode
#define LCD_ENTRYRIGHT 0x00
#define LCD_ENTRYLEFT 0x02
#define LCD_ENTRYSHIFTINCREMENT 0x01
#define LCD_ENTRYSHIFTDECREMENT 0x00

// flags for display on/off control
#define LCD_DISPLAYON 0x04
#define LCD_DISPLAYOFF 0x00
#define LCD_CURSORON 0x02
```

```
#define LCD_CURSOROFF 0x00
#define LCD_BLINKON 0x01
#define LCD_BLINKOFF 0x00

// flags for display/cursor shift
#define LCD_DISPLAYMOVE 0x08
#define LCD_CURSORMOVE 0x00
#define LCD_MOVERIGHT 0x04
#define LCD_MOVELEFT 0x00

// flags for function set
#define LCD_8BITMODE 0x10
#define LCD_4BITMODE 0x00
#define LCD_2LINE 0x08
#define LCD_1LINE 0x00
#define LCD_5x10DOTS 0x04
#define LCD_5x8DOTS 0x00


/***********************************************
                类定义部分
***********************************************/
class LiquidCrystal
{
public:
  LiquidCrystal(uint8_t rs, uint8_t enable,
                uint8_t d0, uint8_t d1, uint8_t d2, uint8_t d3,
                uint8_t d4, uint8_t d5, uint8_t d6, uint8_t d7);
  LiquidCrystal(uint8_t rs, uint8_t rw, uint8_t enable,
                uint8_t d0, uint8_t d1, uint8_t d2, uint8_t d3,
                uint8_t d4, uint8_t d5, uint8_t d6, uint8_t d7);
  LiquidCrystal(uint8_t rs, uint8_t rw, uint8_t enable,
                uint8_t d0, uint8_t d1, uint8_t d2, uint8_t d3);
  LiquidCrystal(uint8_t rs, uint8_t enable,
                uint8_t d0, uint8_t d1, uint8_t d2, uint8_t d3);

  void init(uint8_t fourbitmode, uint8_t rs, uint8_t rw, uint8_t enable,
            uint8_t d0, uint8_t d1, uint8_t d2, uint8_t d3,
            uint8_t d4, uint8_t d5, uint8_t d6, uint8_t d7);

  void begin(uint8_t cols, uint8_t rows, uint8_t charsize = LCD_5x8DOTS);

  void clear();
```

```
    void home();

    void noDisplay();
    void display();
    void noBlink();
    void blink();
    void noCursor();
    void cursor();
    void scrollDisplayLeft();
    void scrollDisplayRight();
    void leftToRight();
    void rightToLeft();
    void autoscroll();
    void noAutoscroll();

    void createChar(uint8_t, uint8_t[]);
    void setCursor(uint8_t, uint8_t);
    virtual void write(uint8_t);
    void command(uint8_t);

  private:
    void send(uint8_t, uint8_t);
    void write4bits(uint8_t);
    void write8bits(uint8_t);
    void pulseEnable();

    uint8_t _rs_pin; // LOW: command.  HIGH: character.
    uint8_t _rw_pin; // LOW: write to LCD.  HIGH: read from LCD.
    uint8_t _enable_pin; // activated by a HIGH pulse.
    uint8_t _data_pins[8];

    uint8_t _displayfunction;
    uint8_t _displaycontrol;
    uint8_t _displaymode;
    uint8_t _initialized;
    uint8_t _numlines,_currline;
};
```

这里只介绍一下公有成员函数（类定义中的public部分）。液晶模块的控制方式及控制时序可参考5.3节。

6.3.1 构造函数

由于LiquidCrystal类支持多种控制方式，包括总线的差别（4位或8位）、是否控制RW引

脚等，因此LiquidCrystal类有4个构造函数，如下：

```
        LiquidCrystal(uint8_t rs, uint8_t enable,
                      uint8_t d0, uint8_t d1, uint8_t d2, uint8_t d3,
                      uint8_t d4, uint8_t d5, uint8_t d6, uint8_t d7);

LiquidCrystal(uint8_t rs, uint8_t rw, uint8_t enable,
                      uint8_t d0, uint8_t d1, uint8_t d2, uint8_t d3,
                      uint8_t d4, uint8_t d5, uint8_t d6, uint8_t d7);

LiquidCrystal(uint8_t rs, uint8_t rw, uint8_t enable,
                      uint8_t d0, uint8_t d1, uint8_t d2, uint8_t d3);

LiquidCrystal(uint8_t rs, uint8_t enable,
                      uint8_t d0, uint8_t d1, uint8_t d2, uint8_t d3);
```

❑ 参数：

rs：指Arduino连接液晶模块RS（数据/命令寄存器选择）的引脚；

enable：指Arduino连接液晶模块E（使能）的引脚；

rw：指Arduino连接液晶模块R/W的引脚；

d0～d7：指Arduino连接数据口的引脚，在4位总线的控制方式下，不需要连接液晶模块的D0～D3引脚。

构造函数的原型如下：

```
LiquidCrystal::LiquidCrystal(uint8_t rs, uint8_t rw, uint8_t enable,
                             uint8_t d0, uint8_t d1, uint8_t d2, uint8_t d3,
                             uint8_t d4, uint8_t d5, uint8_t d6, uint8_t d7)
{
  init(0, rs, rw, enable, d0, d1, d2, d3, d4, d5, d6, d7);
}

LiquidCrystal::LiquidCrystal(uint8_t rs, uint8_t enable,
                             uint8_t d0, uint8_t d1, uint8_t d2, uint8_t d3,
                             uint8_t d4, uint8_t d5, uint8_t d6, uint8_t d7)
{
  init(0, rs, 255, enable, d0, d1, d2, d3, d4, d5, d6, d7);
}

LiquidCrystal::LiquidCrystal(uint8_t rs, uint8_t rw, uint8_t enable,
                             uint8_t d0, uint8_t d1, uint8_t d2, uint8_t d3)
{
  init(1, rs, rw, enable, d0, d1, d2, d3, 0, 0, 0, 0);
}
```

```
LiquidCrystal::LiquidCrystal(uint8_t rs,  uint8_t enable,
                             uint8_t d0, uint8_t d1, uint8_t d2, uint8_t d3)
{
  init(1, rs, 255, enable, d0, d1, d2, d3, 0, 0, 0, 0);
}
```

通过分析代码可以发现4个构造函数实际上只进行了一个参数传递的工作，其内部均使用同一个函数init()完成初始化，其原型如下：

```
void LiquidCrystal::init(uint8_t fourbitmode, uint8_t rs, uint8_t rw, uint8_t enable,
                         uint8_t d0, uint8_t d1, uint8_t d2, uint8_t d3,
                         uint8_t d4, uint8_t d5, uint8_t d6, uint8_t d7)
{
  _rs_pin = rs;
  _rw_pin = rw;
  _enable_pin = enable;

  _data_pins[0] = d0;
  _data_pins[1] = d1;
  _data_pins[2] = d2;
  _data_pins[3] = d3;
  _data_pins[4] = d4;
  _data_pins[5] = d5;
  _data_pins[6] = d6;
  _data_pins[7] = d7;

  pinMode(_rs_pin, OUTPUT);
  // we can save 1 pin by not using RW. Indicate by passing 255 instead of pin#
  if (_rw_pin != 255)
      {
    pinMode(_rw_pin, OUTPUT);
  }
  pinMode(_enable_pin, OUTPUT);

  if (fourbitmode)
    _displayfunction = LCD_4BITMODE | LCD_1LINE | LCD_5x8DOTS;
  else
    _displayfunction = LCD_8BITMODE | LCD_1LINE | LCD_5x8DOTS;

  begin(16, 1);
}
```

构造函数在创建LiquidCrystal类的对象时会自动调用，在程序中的应用如下（假设采用4位总线控制方式，RS连接Arduino的引脚8，E连接引脚9，D4～D7分别连接引脚4～7）：

```
LiquidCrystal lcd(8,9,4,5,6,7);
```

6.3.2 command( )和write( )

command()函数和write()函数是两个较为底层的函数，用于发送命令和数据给液晶模块，这两个函数用于最基本的液晶模块控制，一般不需要使用者调用。

- 返回值：无。
- 参数：value，表示需要发送的命令或数据。

两个函数的原型如下：

```
//command()函数
void LiquidCrystal::command(uint8_t value)
{
        digitalWrite(_rs_pin, LOW);

        // if there is a RW pin indicated, set it low to Write
        if (_rw_pin != 255)
        {
        digitalWrite(_rw_pin, LOW);
        }

        if (_displayfunction & LCD_8BITMODE)
        {
        write8bits(value);
        }
        else
        {
        write4bits(value>>4);
        write4bits(value);
        }
}

//write()函数
void LiquidCrystal::write(uint8_t value)
{
        digitalWrite(_rs_pin, HIGH);

        // if there is a RW pin indicated, set it low to Write
        if (_rw_pin != 255)
        {
        digitalWrite(_rw_pin, LOW);
        }
```

```
        if (_displayfunction & LCD_8BITMODE)
        {
        write8bits(value);
        }
        else
        {
        write4bits(value>>4);
        write4bits(value);
        }

}
```

6.3.3 begin( )

begin()函数用于设置LCD液晶显示器的显示尺寸，即显示区域的行数和列数。

- 返回值：无。
- 参数：列数（cols）、行数（lines）和字符大小（dotsize，可省略，默认5×8个像素点）。

函数原型如下：

```
void LiquidCrystal::begin(uint8_t cols, uint8_t lines, uint8_t dotsize)
{
      if (lines > 1)
      {
                  _displayfunction |= LCD_2LINE;
      }
      numlines= lines;
      _currline = 0;

      //LCD液晶模块上电后至少等待40ms后开始发送数据
      delayMicroseconds(50000);
      //将lcd的RS和R/W置位
      digitalWrite(_rs_pin, LOW);
      digitalWrite(_enable_pin, LOW);
      if (_rw_pin != 255)
      {
      digitalWrite(_rw_pin, LOW);
      }

      //设置LCD的工作方式是4位总线还是8位总线
      if (! (_displayfunction & LCD_8BITMODE))
      {
                  //LCD模块初始为8位总线形式，将其设置为4位总线形式
```

```
        write4bits(0x03);
        delayMicroseconds(4500);

                    // second try
                    write4bits(0x03);
                    delayMicroseconds(4500);

                    // third go!
                    write4bits(0x03);
                    delayMicroseconds(150);

                    // finally, set to 8-bit interface
                    write4bits(0x02);
        }
        else
        {
        // Send function set command sequence
        command(LCD_FUNCTIONSET | _displayfunction);
        delayMicroseconds(4500);  // wait more than 4.1ms

        // second try
        command(LCD_FUNCTIONSET | _displayfunction);
        delayMicroseconds(150);

        // third go
        command(LCD_FUNCTIONSET | _displayfunction);
        }

        // finally, set # lines, font size, etc.
        command(LCD_FUNCTIONSET | _displayfunction);

        // turn the display on with no cursor or blinking default
        _displaycontrol = LCD_DISPLAYON | LCD_CURSOROFF | LCD_BLINKOFF;
        display();

        // clear it off
        clear();

        // Initialize to default text direction (for romance languages)
        _displaymode = LCD_ENTRYLEFT | LCD_ENTRYSHIFTDECREMENT;
        // set the entry mode
        command(LCD_ENTRYMODESET | _displaymode);
}
```

6.3.4 clear( )

clear()函数的作用是清除液晶显示屏上的内容，同时将光标移动到初始位置左上角。

❑ 返回值：无。

❑ 参数：无。

函数原型如下：

```
void LiquidCrystal::clear()
{
  command(LCD_CLEARDISPLAY);// clear display, set cursor position to zero
  delayMicroseconds(2000);  // this command takes a long time!
}
```

6.3.5 home( )

home()函数的作用是将光标移动到初始值左上角，但不清除液晶屏上的内容。

❑ 返回值：无。

❑ 参数：无。

函数原型如下：

```
void LiquidCrystal::home()
{
  command(LCD_RETURNHOME);  // set cursor position to zero
  delayMicroseconds(2000);  // this command takes a long time!
}
```

6.3.6 setCursor( )

setCursor()函数的作用是设置光标的位置。

❑ 返回值：无。

❑ 参数：函数带两个参数，分别是列数（col）和行数（row）。

函数原型如下：

```
void LiquidCrystal::setCursor(uint8_t col, uint8_t row)
{
        int row_offsets[] = { 0x00, 0x40, 0x14, 0x54 };
        if ( row > _numlines )
        {
        row = _numlines-1;
        }

        command(LCD_SETDDRAMADDR | (col + row_offsets[row]));
}
```

6.3.7 noDisplay( )和display( )

noDisplay()函数和display()函数用于关闭或打开液晶显示屏。

- 返回值：无。
- 参数：无。

两个函数的原型如下：

```
void LiquidCrystal::noDisplay()
{
  _displaycontrol &= ~LCD_DISPLAYON;
  command(LCD_DISPLAYCONTROL | _displaycontrol);
}

void LiquidCrystal::display()
{
  _displaycontrol |= LCD_DISPLAYON;
  command(LCD_DISPLAYCONTROL | _displaycontrol);
}
```

6.3.8 cursor( )和noCursor( )

cursor()函数和noCursor()函数用于在液晶模块上显示或不显示光标。

- 返回值：无。
- 参数：无。

两个函数的原型如下：

```
void LiquidCrystal::cursor()
{
  _displaycontrol |= LCD_CURSORON;
  command(LCD_DISPLAYCONTROL | _displaycontrol);
}

void LiquidCrystal::noCursor()
{
  _displaycontrol &= ~LCD_CURSORON;
  command(LCD_DISPLAYCONTROL | _displaycontrol);
}
```

6.3.9 blink( )和noBlink( )

blink()函数和noBlink()函数用于控制显示屏上的光标是否闪烁。

- 返回值：无。
- 参数：无。

两个函数的原型如下：

```
void LiquidCrystal::blink()
{
  _displaycontrol |= LCD_BLINKON;
  command(LCD_DISPLAYCONTROL | _displaycontrol);
}

void LiquidCrystal::noBlink()
{
  _displaycontrol &= ~LCD_BLINKON;
  command(LCD_DISPLAYCONTROL | _displaycontrol);
}
```

6.3.10 autoscroll( )和noAutoscroll( )

autoscroll()函数和noAutoscroll()函数用于打开或关闭液晶显示屏的字符自动滚动功能。

❑ 返回值：无。

❑ 参数：无。

两个函数的原型如下：

```
void LiquidCrystal::autoscroll(void)
{
  _displaymode |= LCD_ENTRYSHIFTINCREMENT;
  command(LCD_ENTRYMODESET | _displaymode);
}

void LiquidCrystal::noAutoscroll(void)
{
  _displaymode &= ~LCD_ENTRYSHIFTINCREMENT;
  command(LCD_ENTRYMODESET | _displaymode);
}
```

6.3.11 scrollDisplayLeft( )和scrollDisplayRight( )

scrollDisplayLeft函数和scrollDisplayRight函数用于控制液晶显示字符的滚动方向（向左或向右）。

❑ 返回值：无。

❑ 参数：无。

两个函数的原型如下：

```
void LiquidCrystal::scrollDisplayLeft(void)
{
```

```
  command(LCD_CURSORSHIFT | LCD_DISPLAYMOVE | LCD_MOVELEFT);
}

void LiquidCrystal::scrollDisplayRight(void)
{
  command(LCD_CURSORSHIFT | LCD_DISPLAYMOVE | LCD_MOVERIGHT);
}
```

6.3.12 print( )

print函数的作用是控制液晶屏显示相应的字符，print函数有两种形态print(data, BASE)和print(data)，用于不同情况下显示字符的需求。

❑ 返回值：无。

❑ 参数：

data：表示要显示的字符或数据；

BASE：表示现实数据的形式（二进制BIN、八进制OCT、十进制DEC或十六进制HEX）。

函数原型如下：

```
void print(const String &s)
{
  for (int i = 0; i < s.length(); i++) {
    write(s[i]);
  }
}

void printNumber(unsigned long n, uint8_t base)
{
        unsigned char buf[8 * sizeof(long)]; // Assumes 8-bit chars.
        unsigned long i = 0;

        if (n == 0)
        {
        print('0');
        return;
        }

        while (n > 0)
        {
        buf[i++] = n % base;
        n /= base;
        }
```

```
        for (; i > 0; i--)
          print((char) (buf[i - 1] < 10 ?
                    '0' + buf[i - 1] :
                    'A' + buf[i - 1] - 10));
}
/**********************************************/
void print(long n, int base)
{
        if (base == 0)
        {
        write(n);
        }
        else if (base == 10)
        {
        if (n < 0)
        {
        print('-');
        n = -n;
        }
        printNumber(n, 10);
        }
        else
        {
        printNumber(n, base);
        }
}
```

6.4 Ethernet库

Ethernet库在第5章已经接触过了，该库文件针对Ethernet Shield扩展板，将帮助Arduino控制板连接到互联网上，最多支持4路双向连接。既可以将Arduino作为客户端使用，也可以作为服务器端使用，为此，库文件中定义了3个类：EthernetClass类、Server类和Client类。

6.4.1 EthernetClass类定义

EthernetClass类的具体定义可以参看Arduino开发环境目录下libraries文件夹内的Ethernet.h文件，该文件包含一些常数的宏定义和类的定义，内容如下：

```
#define MAX_SOCK_NUM 4                    //最多支持4路

class EthernetClass
{
```

```
    private:
    public:
            static uint8_t _state[MAX_SOCK_NUM];
            static uint16_t _server_port[MAX_SOCK_NUM];
            void begin(uint8_t *mac, uint8_t *ip);
            void begin(uint8_t * mac, uint8_t * ip, uint8_t *gateway);
            void begin(uint8_t * mac, uint8_t * ip, uint8_t * gateway,
            uint8_t *subnet);
            friend class Client;
            friend class Server;
};
```

由类的定义可以看出EthernetClass类没有构造函数，只有一个相关的begin函数，该函数有3种形态，用来初始化类的对象并设置网络参数，包括mac（本机MAC地址）、ip（本机IP地址）、gateway（本机网关）和subnet（本机子网掩码）。

提示：网络参数设置内容可参考5.4节相关内容。

begin函数的原型如下：

```
void EthernetClass::begin(uint8_t *mac, uint8_t *ip)
{
  uint8_t gateway[4];
  gateway[0] = ip[0];
  gateway[1] = ip[1];
  gateway[2] = ip[2];
  gateway[3] = 1;
  begin(mac, ip, gateway);
}

void EthernetClass::begin(uint8_t *mac, uint8_t *ip, uint8_t *gateway)
{
  uint8_t subnet[] = {
    255, 255, 255, 0   };
  begin(mac, ip, gateway, subnet);
}

void EthernetClass::begin(uint8_t *mac, uint8_t *ip, uint8_t *gateway, uint8_t *subnet)
{
  W5100.init();
  W5100.setMACAddress(mac);
  W5100.setIPAddress(ip);
  W5100.setGatewayIp(gateway);
  W5100.setSubnetMask(subnet);
}
```

6.4.2 Server类定义

Server类的具体定义可以参看Arduino开发环境目录下libraries文件夹内的Server.h文件，内容如下：

```
class Server
{
private:
  uint16_t _port;
  void accept();
public:
  Server(uint16_t);
  Client available();
  void begin();
  virtual void write(uint8_t);
  virtual void write(const char *str);
  virtual void write(const uint8_t *buf, size_t size);
};
```

6.4.3 Server类构造函数

构造函数用于创建一个Server类，在参数port指定的端口侦听输入的连接请求。

❑ 参数：port表示端口号。

构造函数原型如下:

```
Server::Server(uint16_t port)
{
  _port = port;
}
```

6.4.4 Server类成员函数

1. begin( )

begin函数的作用是使Server类的对象开始侦听端口的连接请求。

❑ 返回值：无。

❑ 参数：无。

函数原型如下：

```
void Server::begin()
{
  for (int sock = 0; sock < MAX_SOCK_NUM; sock++) {
    Client client(sock);
    if (client.status() == SnSR::CLOSED) {
```

```
        socket(sock, SnMR::TCP, _port, 0);
        listen(sock);
        EthernetClass::_server_port[sock] = _port;
        break;
      }
    }
}
```

2. available( )

available函数的作用是一个可用的客户端连接到服务器，当连接存在时返回客户端对象。

❑ 返回值：

Client类型，表示可用的客户端对象。

❑ 参数：无。

函数原型如下：

```
Client Server::available()
{
        accept();

        for (int sock = 0; sock < MAX_SOCK_NUM; sock++)
        {
        Client client(sock);
        if (EthernetClass::_server_port[sock] == _port &&
                            (client.status() == SnSR::ESTABLISHED ||
                            client.status() == SnSR::CLOSE_WAIT))
                    {
                    if (client.available())
                            {
                                        return client;
                    }
        }
        }

        return Client(MAX_SOCK_NUM);
}
```

3. write( )

write函数的作用是给所有连接到服务器的客户端发送数据。

❑ 返回值：无。

❑ 参数：write函数有3种形态，参数说明如下：

b：表示所要发送的1个字节的数据；

*str：表示所要发送的字符串的指针；

*buffer：表示所要发送数据的缓存器的指针；

size：表示所要发送数据的大小。

函数原型如下：

```
void Server::write(uint8_t b)
{
  write(&b, 1);
}

void Server::write(const char *str)
{
  write((const uint8_t *)str, strlen(str));
}

void Server::write(const uint8_t *buffer, size_t size)
{
  accept();

  for (int sock = 0; sock < MAX_SOCK_NUM; sock++) {
    Client client(sock);

    if (EthernetClass::_server_port[sock] == _port &&
      client.status() == SnSR::ESTABLISHED) {
      client.write(buffer, size);
    }
  }
}
```

4. print( )

print函数的作用是给所有连接到服务器的客户端发送数据，print函数有两种形态：print(data, BASE)和print(data)，用于不同情况下显示字符的需求。

❑ 返回值：无。

❑ 参数：

data：表示要显示的字符或数据；

BASE：表示现实数据的形式（二进制BIN、八进制OCT、十进制DEC或十六进制HEX）。

函数原型如下：

```
void print(const String &s)
{
  for (int i = 0; i < s.length(); i++) {
    write(s[i]);
```

```
  }
}

void printNumber(unsigned long n, uint8_t base)
{
      unsigned char buf[8 * sizeof(long)]; // Assumes 8-bit chars.
 unsigned long i = 0;

      if (n == 0)
      {
      print('0');
      return;
      }

      while (n > 0)
      {
      buf[i++] = n % base;
      n /= base;
      }

      for (; i > 0; i--)
      print((char) (buf[i - 1] < 10 ?
                '0' + buf[i - 1] :
                'A' + buf[i - 1] - 10));
}
/*********************************************/
void print(long n, int base)
{
      if (base == 0)
      {
      write(n);
      }
      else if (base == 10)
      {
      if (n < 0)
                {
                print('-');
                n = -n;
                }
      printNumber(n, 10);
      }
      else
      {
```

```
        printNumber(n, base);
        }
}
```

6.4.5 Client类定义

Client类的具体定义可以参看Arduino开发环境目录下libraries文件夹内的Client.h文件，内容如下：

```
class Client
{
public:
  Client(uint8_t *ip, uint16_t port);

  uint8_t status();
  uint8_t connect();
  virtual void write(uint8_t);
  virtual void write(const char *str);
  virtual void write(const uint8_t *buf, size_t size);
  virtual int available();
  virtual int read();
  virtual int peek();
  virtual void flush();
  void stop();
  uint8_t connected();
  uint8_t operator==(int);
  uint8_t operator!=(int);
  operator bool();

  friend class Server;

private:
  static uint16_t _srcport;
  uint8_t _sock;
  uint8_t *_ip;
  uint16_t _port;
};
```

6.4.6 Client类构造函数

构造函数用于创建一个可以连接到指定的IP地址和端口的客户端。

❑ 参数：

*ip：客户端需连接到的IP地址数组的指针；

port：客户端需连接到的端口号。

构造函数原型如下：

```
Client::Client(uint8_t*ip,uint16_t port):_ip(ip),_port(port), _sock(MAX_SOCK_NUM){}
```

6.4.7 Client类成员函数

1. status( )

status函数用于获取模块的状态。

❑ 返回值：

uint8类型，表示模块的状态。

❑ 参数：无。

函数原型如下：

```
uint8_t Client::status()
{
      if (_sock == MAX_SOCK_NUM) return SnSR::CLOSED;
      return W5100.readSnSR(_sock);
}
```

2. connect( )

connect函数会将模块连接到构造函数中指定的IP地址和端口上。

❑ 返回值：

uint8类型，表示连接成功（返回true）或失败（返回false）。

❑ 参数：无。

函数原型如下：

```
uint8_t Client::connect()
{
      if (_sock != MAX_SOCK_NUM)
      return 0;

      for (int i = 0; i < MAX_SOCK_NUM; i++)
      {
      uint8_t s = W5100.readSnSR(i);
      if (s == SnSR::CLOSED || s == SnSR::FIN_WAIT)
                {
                _sock = i;
                break;
      }
}
```

```
    if (_sock == MAX_SOCK_NUM)
    return 0;

    _srcport++;
    if (_srcport == 0) _srcport = 1024;
    socket(_sock, SnMR::TCP, _srcport, 0);

    if (!::connect(_sock, _ip, _port))
    {
    _sock = MAX_SOCK_NUM;
    return 0;
    }

    while (status() != SnSR::ESTABLISHED)
    {
    delay(1);
    if (status() == SnSR::CLOSED)
                {
                _sock = MAX_SOCK_NUM;
                return 0;
    }
    }

    return 1;
}
```

3. write( )

write函数的作用是给客户端连接的服务器发送数据。

❑ 返回值：无。

❑ 参数：write函数有3种形态，参数说明如下：

b：表示所要发送的1个字节的数据；

*str：表示所要发送的字符串的指针；

*buffer：表示所要发送数据的缓存器的指针；

size：表示所要发送数据的大小。

函数原型如下：

```
void Client::write(uint8_t b)
{
     if (_sock != MAX_SOCK_NUM)
     send(_sock, &b, 1);
}
```

```
void Client::write(const char *str)
{
        if (_sock != MAX_SOCK_NUM)
           send(_sock, (const uint8_t *)str, strlen(str));
}

void Client::write(const uint8_t *buf, size_t size)
{
        if (_sock != MAX_SOCK_NUM)
           send(_sock, buf, size);
}
```

4. available( )

available函数的作用是返回可读取的字节数。

❑ 返回值：

int类型，表示可读取的字节数。

❑ 参数：无。

函数原型如下：

```
int Client::available()
{
        if (_sock != MAX_SOCK_NUM)
        return W5100.getRXReceivedSize(_sock);
        return 0;
}
```

5. read( )

read函数的作用是读取下一个接收到的字节。

❑ 返回值：

int类型，表示下一个接收到的字节信息。

❑ 参数：无。

函数原型如下：

```
int Client::read() {
      uint8_t b;
      if (!available())
         return -1;
      recv(_sock, &b, 1);
      return b;
}
```

6. flush( )

flush函数的作用是丢弃所有已写入客户端但尚未读取的字节。

❑ 返回值：无。

❑ 参数：无。

函数原型如下：

```
void Client::flush()
{
    while (available())
    read();
}
```

7. connected( )

connected函数用于获知客户端是否已连接。

❑ 返回值：

uint8类型，表示客户端是否连接，返回true表示客户端连接，返回false表示客户端未连接。

❑ 参数：无。

函数原型如下：

```
uint8_t Client::connected()
{
      if (_sock == MAX_SOCK_NUM)
                return 0;

      uint8_t s = status();
      return !(s==SnSR::LISTEN||s == SnSR::CLOSED||s == SnSR::FIN_WAIT ||
    (s == SnSR::CLOSE_WAIT && !available()));
}
```

8. print( )

print函数的作用是给客户端连接的服务器发送数据，print函数有两种形态：print(data, BASE)和print(data)，用于不同情况下显示字符的需求。

❑ 返回值：无。

❑ 参数：

data：表示要显示的字符或数据；

BASE：表示现实数据的形式（二进制BIN、八进制OCT、十进制DEC或十六进制HEX）。

函数原型如下：

```
void print(const String &s)
```

```
{
  for (int i = 0; i < s.length(); i++) {
    write(s[i]);
  }
}

void printNumber(unsigned long n, uint8_t base)
{
      unsigned char buf[8 * sizeof(long)]; // Assumes 8-bit chars.
  unsigned long i = 0;

      if (n == 0)
      {
      print('0');
      return;
      }

      while (n > 0)
      {
      buf[i++] = n % base;
      n /= base;
      }

      for (; i > 0; i--)
      print((char) (buf[i - 1] < 10 ?
                '0' + buf[i - 1] :
                'A' + buf[i - 1] - 10));
}
/*********************************************/
void print(long n, int base)
{
      if (base == 0)
      {
      write(n);
      }
      else if (base == 10)
      {
      if (n < 0)
                {
                print('-');
                n = -n;
                }
      printNumber(n, 10);
```

```
    }
        else
        {
        printNumber(n, base);
        }
}
```

9. stop( )

stop函数的作用是断开与服务器的连接。

❑ 返回值：无。

❑ 参数：无。

函数原型如下：

```
void Client::stop()
{
      if (_sock == MAX_SOCK_NUM)
      return;

      // attempt to close the connection gracefully (send a FIN to other side)
      disconnect(_sock);
      unsigned long start = millis();

      // wait a second for the connection to close
      while (status() != SnSR::CLOSED && millis() - start < 1000)
      delay(1);

      // if it hasn't closed, close it forcefully
      if (status() != SnSR::CLOSED)
      close(_sock);

      EthernetClass::_server_port[_sock] = 0;
      _sock = MAX_SOCK_NUM;
}
```

注意：客户端与服务器的连接断开后，可能仍会有未读出的数据。

6.5 SoftwareSerial库

一般来说，Arduino的串行通信通过占用引脚0和1的硬件资源来实现。但使用SoftwareSerial库可通过软件模拟的方式利用任意两个I/O实现串口通信。库中定义了一个

SoftwareSerial类，类的具体定义可以参看Arduino开发环境目录下libraries文件夹内的SoftwareSerial.h文件，内容如下：

```
class SoftwareSerial
{
  private:
    uint8_t _receivePin;
    uint8_t _transmitPin;
    long _baudRate;
    int _bitPeriod;
    void printNumber(unsigned long, uint8_t);
  public:
    SoftwareSerial(uint8_t, uint8_t);
    void begin(long);
    int read();
    void print(char);
    void print(const char[]);
    void print(uint8_t);
    void print(int);
    void print(unsigned int);
    void print(long);
    void print(unsigned long);
    void print(long, int);
    void println(void);
    void println(char);
    void println(const char[]);
    void println(uint8_t);
    void println(int);
    void println(long);
    void println(unsigned long);
    void println(long, int);
};
```

6.5.1 构造函数

SoftwareSerial类的构造函数用于在创建类的对象时指定用于串口通信的数据发送引脚和数据接收引脚。

❑ 参数：

receivePin：表示接收数据的引脚号；

transmitPin：表示发送数据的引脚号。

构造函数原型如下：

```
SoftwareSerial::SoftwareSerial(uint8_t receivePin, uint8_t transmitPin)
```

```
{
        _receivePin = receivePin;
        _transmitPin = transmitPin;
        _baudRate = 0;
}
```

6.5.2 begin( )

begin函数用于设置串行通信的速率。

❑ 返回值：无。

❑ 参数：

speed：串行通信速率，最大传输速率不能超过9600bps。

函数原型如下：

```
void SoftwareSerial::begin(long speed)
{
      _baudRate = speed;
      _bitPeriod = 1000000 / _baudRate;

      digitalWrite(_transmitPin, HIGH);
      delayMicroseconds( _bitPeriod); // if we were low this establishes the end
}
```

6.5.3 read( )

read函数用于读取串行通信中接收到的字符。

❑ 返回值：

int类型，表示接收到的数据。

❑ 参数：无。

函数原型如下：

```
int SoftwareSerial::read()
{
       int val = 0;
       int bitDelay = _bitPeriod - clockCyclesToMicroseconds(50);

       // one byte of serial data (LSB first)
       // ...---\    /--\/--\/--\/--\/--\/--\/--\/--...
       //         \--/\--/\--/\--/\--/\--/\--/\--/
       //       start  0   1   2   3   4   5   6   7 stop

       while (digitalRead(_receivePin));
```

```
        // confirm that this is a real start bit, not line noise
        if (digitalRead(_receivePin) == LOW) {
          // frame start indicated by a falling edge and low start bit
          // jump to the middle of the low start bit
          delayMicroseconds(bitDelay / 2 - clockCyclesToMicroseconds(50));

          // offset of the bit in the byte: from 0 (LSB) to 7 (MSB)
          for (int offset = 0; offset < 8; offset++) {
              // jump to middle of next bit
              delayMicroseconds(bitDelay);

              // read bit
              val |= digitalRead(_receivePin) << offset;
          }

          delayMicroseconds(_bitPeriod);

          return val;
        }

        return -1;
}
```

6.5.4 print( )和println( )

print函数和println函数用于在数据输出引脚串行输出数据，println函数在数据输出完成后会输出一个回车换行符。函数均无返回值。由于所发送数据的类型不同，所以print函数和println函数有多种形态。原型如下：

```
void SoftwareSerial::print(uint8_t b)
{
        if (_baudRate == 0)
          return;

        int bitDelay = _bitPeriod - clockCyclesToMicroseconds(50); // a
        digitalWrite is about 50 cycles
        byte mask;

        digitalWrite(_transmitPin, LOW);
        delayMicroseconds(bitDelay);

        for (mask = 0x01; mask; mask <<= 1) {
          if (b & mask){ // choose bit
```

```
            digitalWrite(_transmitPin,HIGH); // send 1
          }
          else{
            digitalWrite(_transmitPin,LOW); // send 1
          }
          delayMicroseconds(bitDelay);
        }

        digitalWrite(_transmitPin, HIGH);
        delayMicroseconds(bitDelay);
}

void SoftwareSerial::print(const char *s)
{
        while (*s)
          print(*s++);
}

void SoftwareSerial::print(char c)
{
        print((uint8_t) c);
}

void SoftwareSerial::print(int n)
{
        print((long) n);
}

void SoftwareSerial::print(unsigned int n)
{
        print((unsigned long) n);
}

void SoftwareSerial::print(long n)
{
        if (n < 0) {
          print('-');
          n = -n;
        }
        printNumber(n, 10);
}

void SoftwareSerial::print(unsigned long n)
```

```
{
        printNumber(n, 10);
}

void SoftwareSerial::print(long n, int base)
{
        if (base == 0)
          print((char) n);
        else if (base == 10)
          print(n);
        else
          printNumber(n, base);
}

void SoftwareSerial::println(void)
{
        print('\r');
        print('\n');
}

void SoftwareSerial::println(char c)
{
        print(c);
        println();
}

void SoftwareSerial::println(const char c[])
{
        print(c);
        println();
}

void SoftwareSerial::println(uint8_t b)
{
        print(b);
        println();
}

void SoftwareSerial::println(int n)
{
        print(n);
        println();
}
```

```
void SoftwareSerial::println(long n)
{
        print(n);
        println();
}

void SoftwareSerial::println(unsigned long n)
{
        print(n);
        println();
}

void SoftwareSerial::println(long n, int base)
{
        print(n, base);
        println();
}

// Private Methods
void SoftwareSerial::printNumber(unsigned long n, uint8_t base)
{
          unsigned char buf[8 * sizeof(long)]; // Assumes 8-bit chars.
          unsigned long i = 0;

          if (n == 0)
        {
            print( '0' );
            return;
          }

          while (n > 0)
        {
            buf[i++] = n % base;
            n /= base;
          }

          for (; i > 0; i--)
            print((char) (buf[i-1]<10 ? '0' + buf[i-1]:'A' + buf[i-1]- 10));
}
```

6.5.5 使用限制

由于这种串行通信是由软件方式实现的，所以在使用上有一些限制。

- 最大的数据传输速率只能达到9600bps。
- SoftwareSerial.read( )函数会一直等待，直到接收到数据。
- 只有在SoftwareSerial.read( )函数调用时，接收到的数据才会接收，其他时间接收到的数据将会丢失。

6.6 EEPROM库

通过EEPROM库可以方便地使用Arduino控制板芯片内部的EEPROM来存储数据。控制芯片内部的EEPROM作为一个独立的数据空间而存在，可以按字节读/写，其擦写次数在100 000次以上，这部分数据空间在掉电后依然保持数据内容，常用来记录状态或保存工作参数等。

注意：不同的Arduino控制板由于控制芯片的区别，其内部的EEPROM大小也不同，控制芯片是ATmega328的其内部EEPROM大小为1024字节；控制芯片是ATmega168和ATmega8的大小是512字节；控制芯片是ATmega1280和ATmega2560的大小是4 KB（4096字节）。

EEPROM库中定义了一个EEPROMClass类，类的具体定义可以参看Arduino开发环境目录下libraries文件夹内的EEPROM.h文件，内容如下：

```
class EEPROMClass
{
  public:
    uint8_t read(int);
    void write(int, uint8_t);
};
```

6.6.1 read( )

read函数的作用是从内部EEPROM中读取一个字节。

- 返回值：

uint8类型，表示读取的数据，未写过的地址返回值是255。

- 参数：

address：表示数据的地址。

函数原型如下：

```
uint8_t EEPROMClass::read(int address)
{
      return eeprom_read_byte((unsigned char *) address);
}
```

6.6.2 write( )

write函数用于在指定的地址写入一个字节的数据。

❑ 返回值：无。

❑ 参数：

address：写入的地址；

value：写入的数据。

函数原型如下：

```
void EEPROMClass::write(int address, uint8_t value)
{
      eeprom_write_byte((unsigned char *) address, value);
}
```

6.7 Wire库

Wire库可使我们的Arduino通过IIC/TWI总线连接其他设备，在大部分Arduino控制板中使用Wire库要占用引脚4作为数据线，占用引脚5作为时钟线。在Arduino Mega板上，占用的是引脚20和引脚21。

6.7.1 IIC总线概述

IIC（Inter-Integrated Circuit）总线是由Philips公司开发的两线式串行总线，是具有多主机系统所需的包括总线仲裁和高低速器件同步功能的高性能串行总线。

IIC总线只有两根双向信号线，一根是数据线SDA，另一根是时钟线SCL。所有连接到IIC总线上器件的数据线都连接到SDA上，各器件的时钟线都连接到SCL上。IIC总线是一个多主机总线，总线上可以有一个或多个主机，总线运行由主机控制。每一个接到总线上的器件都有一个唯一的地址识别，且都可以作为一个发送器和接收器。

IIC总线上的SDA和SCL是双向的，均通过上拉电阻接正电源。当总线空闲时，两根信号线均为高电平。总线上器件的输出级必须都是漏级或集电极开路的，以避免对总线上的数据造成干扰，任意器件输出的低电平都将使总线的信号变低。

注意：IIC总线上连接的器件数目受总电容量的限制，总线上连接的器件越多，电容值越大，最大不能超过400pF。在IIC总线模式下，总线速度可达到100kbps，快速模式下可达到400kbps。

6.7.2 TwoWire类定义

Wire库中定义了一个TwoWire类，类的具体定义可以参看Arduino开发环境目录下libraries文件夹内的Wire.h文件，内容如下：

```
#define BUFFER_LENGTH 32

class TwoWire
{
  private:
    static uint8_t rxBuffer[];
    static uint8_t rxBufferIndex;
    static uint8_t rxBufferLength;

    static uint8_t txAddress;
    static uint8_t txBuffer[];
    static uint8_t txBufferIndex;
    static uint8_t txBufferLength;

    static uint8_t transmitting;
    static void (*user_onRequest)(void);
    static void (*user_onReceive)(int);
    static void onRequestService(void);
    static void onReceiveService(uint8_t*, int);
  public:
    void begin();
    void begin(uint8_t address);
    void begin(int address);
    void beginTransmission(uint8_t address);
    void beginTransmission(int address);
    uint8_t endTransmission(void);
    uint8_t requestFrom(uint8_t address, uint8_t quantity);
    uint8_t requestFrom(int address, int quantity);
    void send(uint8_t data);
    void send(uint8_t * data, uint8_t quantity);
    void send(int data);
    void send(char* data);
    uint8_t available(void);
    uint8_t receive(void);
    void onReceive( void (*)(int) );
    void onRequest( void (*)(void) );
};
```

6.7.3 begin( )

begin函数用于设置I IC通信时本机的地址，函数有3种形态。

❑ 返回值：无。

❑ 参数：

address：表示本机的地址，地址为7bit形式。即范围在0～127。

函数原型如下：

```
void TwoWire::begin(void)
{
        rxBufferIndex = 0;
        rxBufferLength = 0;

        txBufferIndex = 0;
        txBufferLength = 0;

        twi_init();
}

void TwoWire::begin(uint8_t address)
{
        twi_setAddress(address);
        twi_attachSlaveTxEvent(onRequestService);
        twi_attachSlaveRxEvent(onReceiveService);
        begin();
}

void TwoWire::begin(int address)
{
        begin((uint8_t)address);
}
```

6.7.4 requestFrom( )

requestFrom函数在主机模式下设置来自于从机的字节量。在调用该函数后必须调用available函数和receive函数（见6.6.5节和6.6.6节）。

❑ 返回值：无。

❑ 参数：

address：从机地址；

quantity：字节数的要求。

函数原型为：

```
uint8_t TwoWire::requestFrom(uint8_t address, uint8_t quantity)
{
        // clamp to buffer length
        if(quantity > BUFFER_LENGTH){
          quantity = BUFFER_LENGTH;
        }
        // perform blocking read into buffer
        uint8_t read = twi_readFrom(address, rxBuffer, quantity);
        // set rx buffer iterator vars
        rxBufferIndex = 0;
        rxBufferLength = read;

        return read;
}
```

6.7.5 available ( )

available函数用于返回接收的字节数。

❑ 返回值：

uint8类型，表示接收的字节数。

❑ 参数：无。

函数原型如下：

```
uint8_t TwoWire::available(void)
{
        return rxBufferLength - rxBufferIndex;
}
```

6.7.6 receive( )

receive函数的作用是获取下一个接收的字节。

❑ 返回值：

uint8类型，表示下一个接收的字节。

❑ 参数：无。

函数原型如下：

```
uint8_t TwoWire::receive(void)
{
        // default to returning null char
        // for people using with char strings
        uint8_t value = '\0';
```

```
        // get each successive byte on each call
        if(rxBufferIndex < rxBufferLength){
          value = rxBuffer[rxBufferIndex];
          ++rxBufferIndex;
        }

        return value;
}
```

6.7.7 beginTransmission( )

beginTransmission函数用于启动IIC通信，调用此函数后，再调用send函数向指定的从机发送数据或接收数据。

❑ 返回值：无。

❑ 参数：

address：从机地址。

函数原型如下：

```
void TwoWire::beginTransmission(uint8_t address)
{
        // indicate that we are transmitting
        transmitting = 1;
        // set address of targeted slave
        txAddress = address;
        // reset tx buffer iterator vars
        txBufferIndex = 0;
        txBufferLength = 0;
}
```

6.7.8 endTransmission( )

endTransmission函数用于结束IIC通信。

❑ 返回值：

uint8类型，表示传输的状态。返回值含义如下：

0——数据传输成功

1——数据太长

2——发送地址时收到NACK

3——发送数据时收到NACK

4——其他错误

❑ 参数：无。

函数原型如下：

```
uint8_t TwoWire::endTransmission(void)
{
        // transmit buffer (blocking)
        int8_t ret = twi_writeTo(txAddress, txBuffer, txBufferLength, 1);
        // reset tx buffer iterator vars
        txBufferIndex = 0;
        txBufferLength = 0;
        // indicate that we are done transmitting
        transmitting = 0;
        return ret;
}
```

6.7.9 send( )

send函数用于发送数据或字符串，函数有4种形态，如下：

```
void send(uint8_t data);
void send(uint8_t * data, uint8_t quantity);
void send(int data);
void send(char* data);
```

❑ 返回值：无。

❑ 参数：

data：表示要发送的数据或字符串；

quantity：表示发送的数据大小。

函数原型如下：

```
void TwoWire::send(uint8_t data)
{
        if(transmitting){
        // in master transmitter mode
          // don't bother if buffer is full
          if(txBufferLength >= BUFFER_LENGTH){
            return;
          }
          // put byte in tx buffer
          txBuffer[txBufferIndex] = data;
          ++txBufferIndex;
          // update amount in buffer
          txBufferLength = txBufferIndex;
        }else{
        // in slave send mode
```

```
            // reply to master
            twi_transmit(&data, 1);
          }
}

void TwoWire::send(uint8_t* data, uint8_t quantity)
{
        if(transmitting){
        // in master transmitter mode
          for(uint8_t i = 0; i < quantity; ++i){
            send(data[i]);
          }
        }else{
        // in slave send mode
          // reply to master
          twi_transmit(data, quantity);
        }
}

void TwoWire::send(char* data)
{
        send((uint8_t*)data, strlen(data));
}


void TwoWire::send(int data)
{
        send((uint8_t)data);
}
```

6.7.10 onReceive( )

onReceive函数的作用是在从机模式下注册一个处理接收数据的函数。

❑ 返回值：无。

❑ 参数：处理函数句柄。

函数原型如下：

```
void TwoWire::onReceive( void (*function)(int) )
{
        user_onReceive = function;
}
```

注册的这个处理接收数据的函数将会被类的私有成员函数onReceiveService()调用，用来

处理接收的数据，调用情况如下：

```
void TwoWire::onReceiveService(uint8_t* inBytes, int numBytes)
{
        // don't bother if user hasn't registered a callback
        if(!user_onReceive){
          return;
        }
        // don't bother if rx buffer is in use by a master requestFrom() op
        // i know this drops data, but it allows for slight stupidity
        // meaning, they may not have read all the master requestFrom() data yet
        if(rxBufferIndex < rxBufferLength){
          return;
        }
        // copy twi rx buffer into local read buffer
        // this enables new reads to happen in parallel
        for(uint8_t i = 0; i < numBytes; ++i){
          rxBuffer[i] = inBytes[i];
        }
        // set rx iterator vars
        rxBufferIndex = 0;
        rxBufferLength = numBytes;
        // alert user program
        user_onReceive(numBytes);
}
```

6.7.11 onRequest( )

onRequest函数的作用是在从机模式下注册一个处理主机请求数据的函数。

- 返回值：无。
- 参数：处理函数句柄。

函数原型如下：

```
void TwoWire::onRequest( void (*function)(void) )
{
        user_onRequest = function;
}
```

注册的这个函数将会被类的私有成员函数onRequestService()调用，调用情况如下：

```
void TwoWire::onRequestService(void)
{
        // don't bother if user hasn't registered a callback
        if(!user_onRequest){
          return;
```

```
        }
        // reset tx buffer iterator vars
        // !!! this will kill any pending pre-master sendTo() activity
        txBufferIndex = 0;
        txBufferLength = 0;
        // alert user program
        user_onRequest();
}
```

6.8 Servo库

Servo库主要针对Arduino控制伺服电机的应用。伺服电机在5.5.2节中已介绍过，是一种用于遥控模型的运动姿态、控制机器手臂动作的角度可控的特殊电机。Servo库最多支持12路伺服电机控制。库中定义了一个Servo类，类的具体定义可以参看Arduino开发环境目录下libraries文件夹内的Servo.h文件，文件包含一些常数的宏定义和类的定义，内容如下：

```
#define Servo_VERSION           2   // software version of this library

#define MIN_PULSE_WIDTH       544   // the shortest pulse sent to a servo
#define MAX_PULSE_WIDTH      2400   // the longest pulse sent to a servo
#define DEFAULT_PULSE_WIDTH 1500   // default pulse width when servo is attached
#define REFRESH_INTERVAL   20000 // minumim time to refresh servos in microseconds

#define SERVOS_PER_TIMER 12 // the maximum number of servos controlled by one timer
#define MAX_SERVOS   (_Nbr_16timers  * SERVOS_PER_TIMER)

#define INVALID_SERVO      255  // flag indicating an invalid servo index

typedef struct
{
          uint8_t nbr       :6 ; // a pin number from 0 to 63
          uint8_t isActive  :1 ; // true if this channel is enabled, pin not
                                   pulsed if false
} ServoPin_t;

typedef struct
{
          ServoPin_t Pin;
          unsigned int ticks;
} servo_t;
```

```
class Servo
{
public:
  Servo();
  uint8_t attach(int pin); // attach the given pin to the next free channel,
  sets pinMode, returns channel number or 0 if failure
  uint8_t attach(int pin, int min, int max); // as above but also sets
  min and max values for writes.
  void detach();
  void write(int value);              // if value is < 200 its treated as
  an angle, otherwise as pulse width in microseconds
  void writeMicroseconds(int value); // Write pulse width in microseconds
  int read();                         // returns current pulse width as an
  angle between 0 and 180 degrees
  int readMicroseconds();             // returns current pulse width in
  microseconds for this servo (was read_us() in first release)
  bool attached();                    // return true if this servo is
  attached, otherwise false
private:
   uint8_t servoIndex;      // index into the channel data for this servo
   int8_t min; // minimum is this value times 4 added to MIN_PULSE_WIDTH
   int8_t max; // maximum is this value times 4 added to MAX_PULSE_WIDTH
};
```

注意：在Arduino开发环境0016版或更早的版本中，Servo库只支持9和10两个引脚。

6.8.1 构造函数

Servo类的构造函数用于初始化类的对象。

❑ 参数：无。

构造函数原型如下：

```
Servo::Servo()
{
     if( ServoCount < MAX_SERVOS)
     {
               // assign a servo index to this instance
     this->servoIndex = ServoCount++;

               // store default values
               servos[this->servoIndex].ticks = usToTicks(DEFAULT_PULSE_WIDTH);
  }
```

```
  else
     this->servoIndex = INVALID_SERVO ;
}
```

6.8.2 attach( )

attach函数用于为伺服电机指定一个引脚。函数有两种形态，如下：

```
uint8_t attach(int pin);
uint8_t attach(int pin, int min, int max);
```

❑ 返回值：

uint8类型，表示通道号，若返回0则表示指定引脚未成功。

❑ 参数：

pin：指定的引脚号；

min：最小角度的脉宽值，单位μs，默认最小值为544，对应最小伺服角度0；

max：最大角度的脉宽值，单位μs，默认最大值2400，对应最大伺服角度180°。

函数原型如下：

```
uint8_t Servo::attach(int pin)
{
      return this->attach(pin, MIN_PULSE_WIDTH, MAX_PULSE_WIDTH);
}

uint8_t Servo::attach(int pin, int min, int max)
{
      if(this->servoIndex < MAX_SERVOS )
      {
      pinMode( pin, OUTPUT) ;                    // set servo pin to output
      servos[this->servoIndex].Pin.nbr = pin;
      // todo min/max check: abs(min - MIN_PULSE_WIDTH) /4 < 128
      this->min = (MIN_PULSE_WIDTH - min)/4;//resolution of min/max is 4 uS
      this->max  = (MAX_PULSE_WIDTH - max)/4;
      // initialize the timer if it has not already been initialized
      timer16_Sequence_t timer = SERVO_INDEX_TO_TIMER(servoIndex);
      if(isTimerActive(timer) == false)
                initISR(timer);
      // this must be set after the check for isTimerActive
                servos[this->servoIndex].Pin.isActive = true;
  }
  return this->servoIndex ;
}
```

6.8.3 write( )

write函数用于设定伺服电机的角度值。

❑ 返回值：无。

❑ 参数：

value：表示设定的角度值，参数值范围在0～180° 。

函数原型如下：

```
void Servo::write(int value)
{
      if(value < MIN_PULSE_WIDTH)
      {
                // treat values less than 544 as angles in degrees
                //(valid values in microseconds are handled as microseconds)
      if(value < 0) value = 0;
      if(value > 180) value = 180;
      value = map(value, 0, 180, SERVO_MIN(),  SERVO_MAX());
      }
      this->writeMicroseconds(value);
}
```

6.8.4 writeMicroseconds( )

writeMicroseconds函数用于设定控制伺服电机的脉冲宽度。

❑ 返回值：无。

❑ 参数：

value：表示设定的脉宽值，单位μs，参数值应在最大脉冲宽度值与最小脉冲宽度值之间。

函数原型如下：

```
void Servo::writeMicroseconds(int value)
{
      // calculate and store the values for the given channel
      byte channel = this->servoIndex;
      if( (channel >= 0) && (channel < MAX_SERVOS)) // ensure channel is valid
      {
      if( value < SERVO_MIN() )          // ensure pulse width is valid
                value = SERVO_MIN();
      else if( value > SERVO_MAX() )
                value = SERVO_MAX();

                value = value - TRIM_DURATION;
      // convert to ticks after compensating for interrupt overhead
                value = usToTicks(value);
```

```
        uint8_t oldSREG = SREG;
        cli();
        servos[channel].ticks = value;
        SREG = oldSREG;
    }
}
```

6.8.5 read( )

read函数用于获取伺服电机的角度值。

❑ 返回值：

int类型，表示伺服电机的角度值，范围在0～180°。

❑ 参数：无。

函数原型如下：

```
int Servo::read() // return the value as degrees
{
    return map( this->readMicroseconds()+1, SERVO_MIN(), SERVO_MAX(), 0, 180);
}
```

6.8.6 readMicroseconds( )

❑ 返回值：无。

int类型，表示伺服电机的脉宽值，单位μs，范围在最大脉冲宽度值与最小脉冲宽度值之间。

❑ 参数：无。

函数原型如下：

```
int Servo::readMicroseconds()
{
    unsigned int pulsewidth;
    if( this->servoIndex != INVALID_SERVO )
    pulsewidth = ticksToUs(servos[this->servoIndex].ticks)  + TRIM_DURATION ;
    else
    pulsewidth  = 0;

    return pulsewidth;
}
```

6.8.7 attached( )

attached函数用于检查伺服电机是否指定了引脚。

❑ 返回值：

布尔类型，若为true则表示伺服电机指定了引脚。

❑ 参数：无。

函数原型如下：

```
bool Servo::attached()
{
      return servos[this->servoIndex].Pin.isActive ;
}
```

6.8.8 detach( )

detach函数用于将伺服电机与指定的引脚分离。

❑ 返回值：无。

❑ 参数：无。

函数原型如下：

```
void Servo::detach()
{
        servos[this->servoIndex].Pin.isActive = false;
        timer16_Sequence_t timer = SERVO_INDEX_TO_TIMER(servoIndex);
        if(isTimerActive(timer) == false) {
          finISR(timer);
        }
}
```

注意：具有PWM功能的引脚在指定为Servo类对象的引脚后丧失PWM输出功能，只有用detach函数将伺服电机与指定的引脚分离，才会恢复PWM输出功能。

6.9 Stepper库

Stepper库主要针对单极性（unipolar）和双极性（bipolar）步进电机。步进电机是一种将电脉冲信号转变为角位移或线位移的开环控制机构，广泛应用在各种自动化控制系统中。Stepper库的应用除了相应的步进电机外，还需要一些元器件来搭建驱动电路。

6.9.1 步进电机概述

通俗地来讲，步进电机就是一个按照一个固定角度一步一步转动的电机，一旦步进驱动器接收到一个脉冲信号，就会驱动步进电机按设定的方向转动一个固定的角度。在非超载的情况下，电机的转速、停止的位置只取决于脉冲信号的频率和脉冲数，而不受负载变化的影

响。当步进驱动器接收到一个脉冲信号，它就驱动步进电机按设定的方向转动一个固定的角度，称为“步距角”，它的旋转是以固定的角度一步一步运行的。可以通过控制脉冲个数来控制角位移量，从而达到准确定位的目的；同时可以通过控制脉冲频率来控制电机转动的速度和加速度，从而达到调速的目的。

6.9.2 步进电机的基本参数

一般步进电机主要有以下参数：步距角、相数、保持转矩、定位转矩、静转矩、拍数等。分别介绍如下。

1. 步距角

步距角是指控制系统每发出一个步进脉冲信号后电机所转动的角度，步距角的用θ表示，θ=360°/（转子齿数×运行拍数）。以常规二、四相，转子齿为50齿的电动机为例，四拍运行时步距角为θ=360°/（50×4）=1.8°（俗称整步），八拍运行时步距角为θ=360°/（50×8）=0.9°（俗称半步）。

2. 相数

步进电机的相数是指电机内部产生不同对极N、S磁场的激磁线圈对数，常用m表示。目前常用的有二相、三相、四相、五相步进电机。电机相数不同，其步距角也不相同。

3. 保持转矩

保持转矩（holding torque）是指步进电机通电但没有转动时，定子锁住转子的力矩。它是步进电机最重要的参数之一，通常步进电机在低转速时的力矩接近保持转矩。由于步进电机的输出力矩随速度的增大而不断衰减，输出功率也随速度的增大而变化，所以保持转矩就成为衡量步进电机最重要的参数之一。

4. 定位转矩

定位转矩（detent torque）是指步进电机在没有通电的情况下，电动机转子自身的锁定力矩。

5. 静转矩

静转矩表示电动机在额定静态电作用下，电动机不作旋转运动时，转轴的锁定力矩。此力矩是衡量电动机体积的标准。

6. 拍数

步进电动机拍数是指完成一个磁场周期性变化所需的脉冲数，用n表示，或者说是电动机转过一个步距角所需的脉冲数。以三相电机为例，三相三拍的工作方式为AB-BC-CA-AB，三相六拍的工作方式为A-AB-B-BC-C-CA-A。

此外，步进电机还有步距角精度、失步、失调角等参数，此处不作介绍。

6.9.3 步进电机的优缺点

优点：

- 电机旋转的角度正比于脉冲数。
- 电机停转的时候具有最大的转矩（当绕组激磁时）。
- 由于每步的精度在3%~5%，而且不会将一步的误差积累到下一步，因而有较好的位置精度和运动的重复性。
- 优秀的起停和反转响应。
- 由于没有电刷，可靠性较高，因此电机的寿命仅仅取决于轴承的寿命。
- 电机的响应仅由数字输入脉冲确定，因而可以采用开环控制，这使得电机的结构比较简单。
- 仅仅将负载直接连接到电机的转轴上也可以极低的速度同步旋转。
- 由于速度正比于脉冲频率，因而有比较宽的转速范围。

缺点：

- 如果控制不当容易产生共振。
- 难以运转到较高的转速。
- 难以获得较大的转矩。
- 在体积重量方面没有优势，能源利用率低。
- 超过负载时会破坏同步，高速工作时会发出振动和噪声。

6.9.4 步进电机的工作原理

这里以三相反应式步进电机为例，介绍一下步进电机的工作原理。一个普通的三相反应式步进电机结构如图6.6所示，由转子和定子两部分组成。

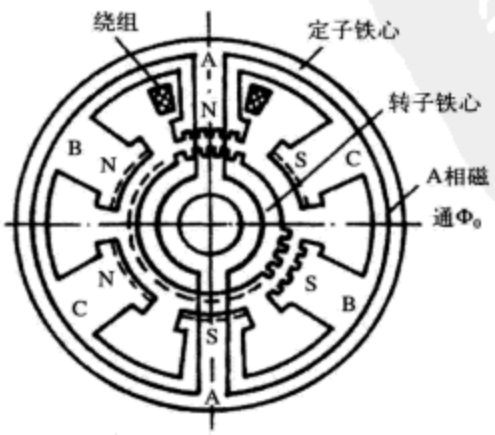


图6.6 三相反应式步进电机结构图

步进电机的定子由硅钢片叠成，上面有6个磁极，每2个相对的磁极组成一对，共3对。每对磁极都绕有同一绕组，形成一相，这样3对磁极就形成三相。类似地，四相步进电机有4对磁极、四相绕组……每个磁极的内表面都分布着多个小齿，它们大小相同，间距相等。

转子的外表面也均匀分布着小齿，这些小齿与定子磁极上的小齿齿距相同，形状相似。所不同的是，转子的小齿是圆周均匀分布的，而定子的小齿只分布在磁极上。当某相上的小齿与转子上的小齿完全对齐时（称为对齿），其他相上的小齿一定与转子上的小齿不对齐（称为错齿），错齿的存在是步进电机旋转的前提条件。

如果给处于错齿状态的相通电，则转子在电磁力的作用下向磁导率最大（或磁阻最小）的位置转动，即向趋于对齿的状态转动。图6.7所示为三相六拍工作方式示意图。

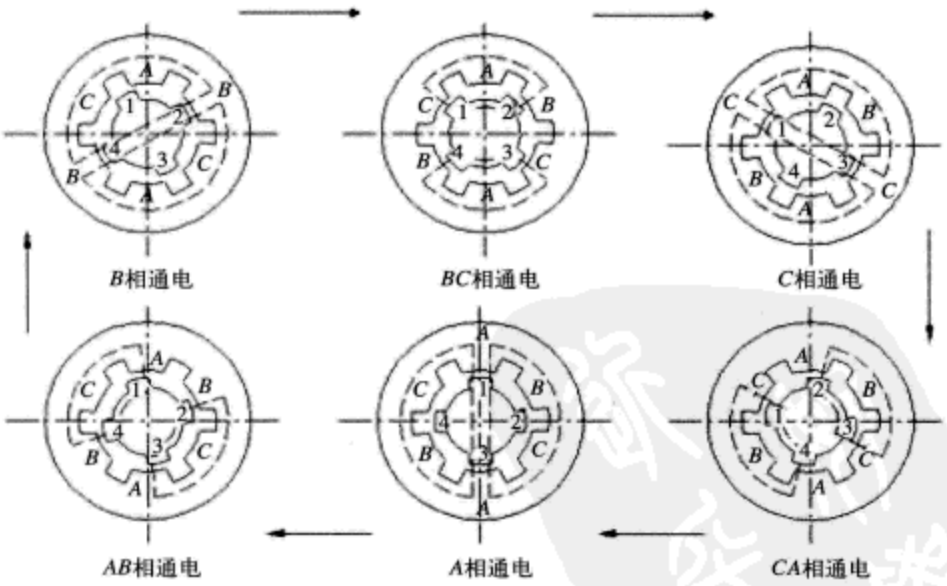


图6.7 三相六拍工作方式示意图

当*A*相通电时，建立*A*相磁场。*A*相上的小齿与转子上的小齿处于对齿状态，*B*、*C*相上小齿与转子小齿处于错齿状态。

当*B*相也通电时，此时磁导率最大的情况是*A*、*B*相趋于对齿状态但都未达到对齿状态，转子转动了一定的角度。

当*A*相断电后，此时只有*B*相通电，转子会接着转动到*B*相小齿与转子小齿处于对齿的状态。

同理，当只有*C*相通电时，转子会转动到*C*相小齿与转子小齿处于对齿的状态。如此不断地变化3相的电压，就可以控制步进电机转动了。三相六拍工作方式时的时序图如图6.8所示。

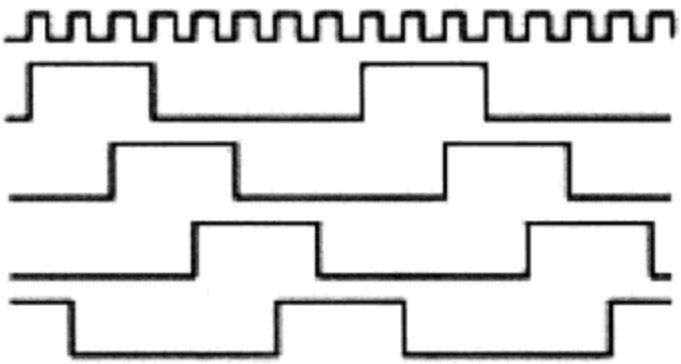


图6.8 三相六拍工作方式时的时序图

6.9.5 步进电机的控制电路

Stepper库适用于单极性和双极性步进电机。单极性步进电机是给每个绕组的中心抽头通电，而绕组的末端接地；双极性步进电机是由H桥回路提供电压，接绕组末端，又根据驱动电路的不同可采用2线制或4线制的控制方式。典型应用电路如图6.9～图6.12所示。

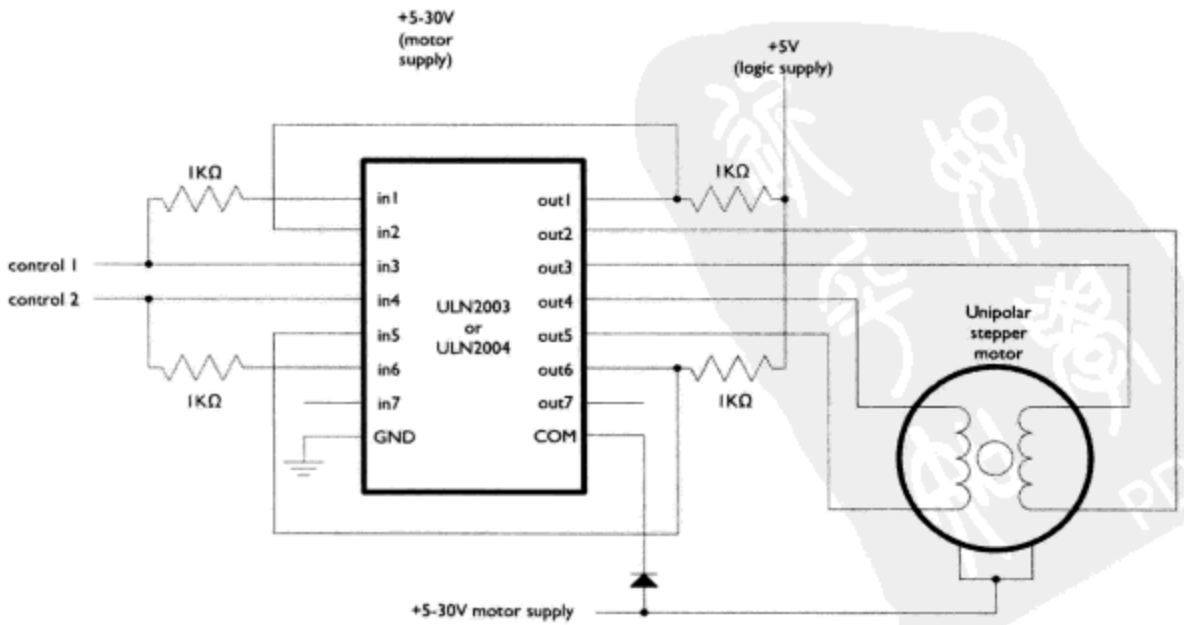


图6.9 单极性步进电机（2线制控制）

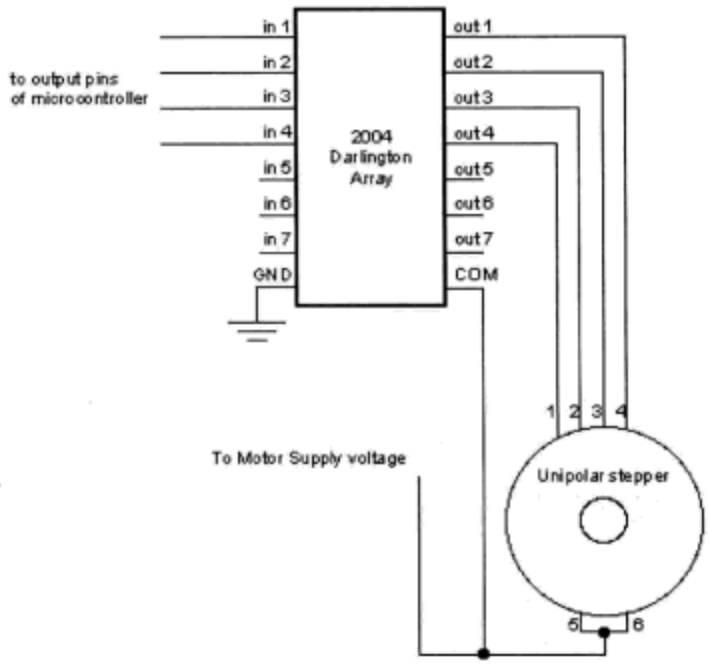


图6.10 单极性步进电机（4线制控制）

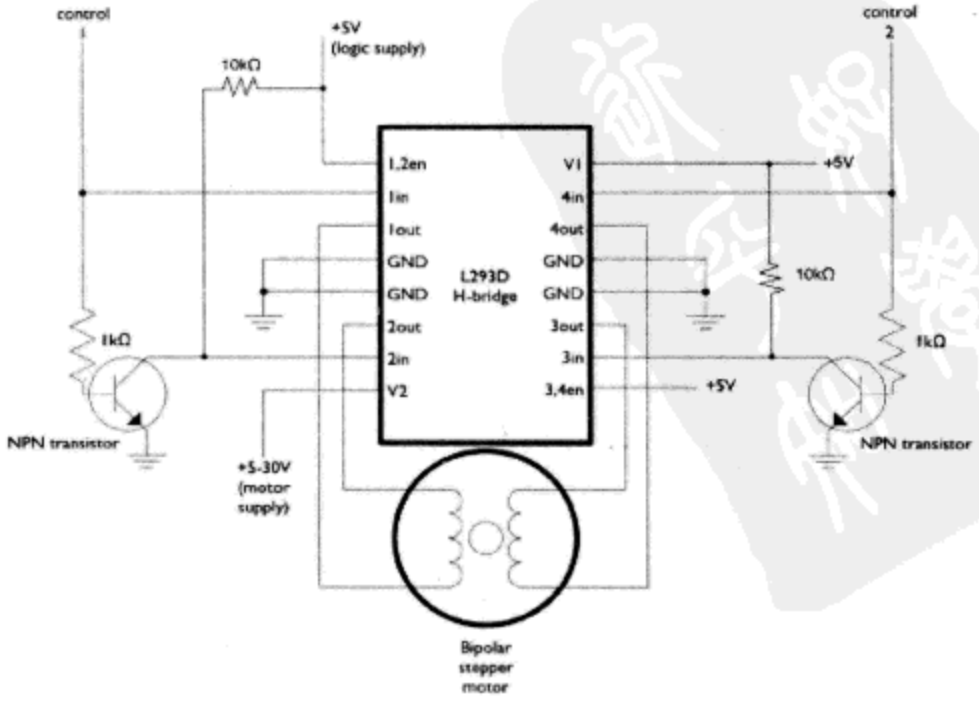


图6.11 双极性步进电机（2线制控制）

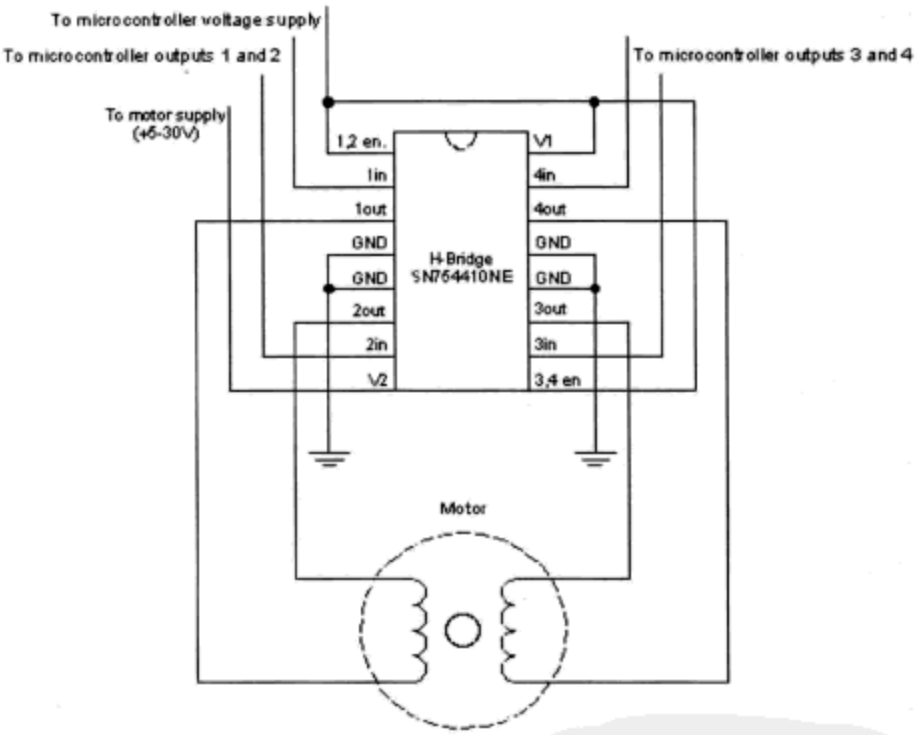


图6.12 双极性步进电机（4线制控制）

提示：通过原理图可以看出，所谓的2线制控制方式只是将4线控制方式中的两个控制脚增加反相电路后合并到另外两个控制引脚上，其本质的控制方式依然是4线制的控制方式。

6.9.6 Stepper类定义

Stepper库中定义了一个Stepper类，类的具体定义可以参看Arduino开发环境目录下libraries文件夹内的Stepper.h文件，内容如下：

```
/*
  Stepper.h - - Stepper library for Wiring/Arduino - Version 0.4

  Drives a unipolar or bipolar stepper motor using  2 wires or 4 wires
*/

// library interface description
class Stepper {
  public:
    // constructors:
```

```
    Stepper(int number_of_steps, int motor_pin_1, int motor_pin_2);
    Stepper(int number_of_steps,
                  int motor_pin_1,
                  int motor_pin_2,
                  int motor_pin_3,
                  int motor_pin_4);

    // speed setter method:
    void setSpeed(long whatSpeed);

    // mover method:
    void step(int number_of_steps);

    int version(void);

  private:
    void stepMotor(int this_step);

    int direction;      // Direction of rotation
    int speed;          // Speed in RPMs
    unsigned long step_delay;// delay between steps, in ms, based on speed
    int number_of_steps; // total number of steps this motor can take
    int pin_count;    // whether you're driving the motor with 2 or 4 pins
    int step_number;    // which step the motor is on

    // motor pin numbers:
    int motor_pin_1;
    int motor_pin_2;
    int motor_pin_3;
    int motor_pin_4;

    long last_step_time;// time stamp in ms of when the last step was taken
};
```

6.9.7 构造函数

Stepper类的构造函数用于在创建对象时指定控制用的引脚。由于有2线制和4线制两种控制方式，所以构造函数有两种形态，如下：

```
Stepper(int number_of_steps, int motor_pin_1, int motor_pin_2);

Stepper(int number_of_steps,int motor_pin_1,int motor_pin_2,
                                    int motor_pin_3,int motor_pin_4);
```

❑ 参数：

number_of_steps：表示步进电机旋转一圈的步数；

motor_pin_1、motor_pin_2：表示两个用于控制步进电机的引脚；

motor_pin_3、motor_pin_4：表示在4线制控制模式下剩余的两个用于控制步进电机的引脚。

构造函数原型如下：

```
/*
 * two-wire constructor.
 * Sets which wires should control the motor.
 */
Stepper::Stepper(int number_of_steps, int motor_pin_1, int motor_pin_2)
{
      this->step_number = 0;    // which step the motor is on
      this->speed = 0;          // the motor speed, in revolutions per minute
      this->direction = 0;      // motor direction
      this->last_step_time = 0;// time stamp in ms of the last step taken
      this->number_of_steps = number_of_steps;// total number of steps for this motor

      // Arduino pins for the motor control connection:
      this->motor_pin_1 = motor_pin_1;
     this->motor_pin_2 = motor_pin_2;

      // setup the pins on the microcontroller:
      pinMode(this->motor_pin_1, OUTPUT);
      pinMode(this->motor_pin_2, OUTPUT);

      // When there are only 2 pins, set the other two to 0:
      this->motor_pin_3 = 0;
      this->motor_pin_4 = 0;

      // pin_count is used by the stepMotor() method:
      this->pin_count = 2;
}

/*
 *   constructor for four-pin version
 *   Sets which wires should control the motor.
 */

Stepper::Stepper(int number_of_steps, int motor_pin_1,
                 int motor_pin_2, int motor_pin_3, int motor_pin_4)
```

```
{
        this->step_number = 0;   // which step the motor is on
        this->speed = 0;         // the motor speed, in revolutions per minute
        this->direction = 0;     // motor direction
        this->last_step_time = 0; // time stamp in ms of the last step taken
        this->number_of_steps = number_of_steps;// total number of steps for this motor

        // Arduino pins for the motor control connection:
        this->motor_pin_1 = motor_pin_1;
        this->motor_pin_2 = motor_pin_2;
        this->motor_pin_3 = motor_pin_3;
        this->motor_pin_4 = motor_pin_4;

        // setup the pins on the microcontroller:
        pinMode(this->motor_pin_1, OUTPUT);
        pinMode(this->motor_pin_2, OUTPUT);
        pinMode(this->motor_pin_3, OUTPUT);
        pinMode(this->motor_pin_4, OUTPUT);

        // pin_count is used by the stepMotor() method:
        this->pin_count = 4;
}
```

6.9.8 setSpeed( )

setSpeed函数用于设定步进电机每分钟旋转的速度。此函数不会使步进电机转动。

❑ 返回值：无。

❑ 参数：

whatSpeed：表示步进电机每分钟旋转的速度。

函数原型如下：

```
void Stepper::setSpeed(long whatSpeed)
{
        this->step_delay = 60L *1000L / this->number_of_steps / whatSpeed;
}
```

6.9.9 step( )

step函数的作用是控制步进电机按照setSpeed函数设置的速度转动一定的步距角。

❑ 返回值：无。

❑ 参数：

steps_to_move：表示转动的步数，若参数大于0则表示正向旋转，若参数小于0则表示反

向旋转。

函数原型如下：

```
void Stepper::step(int steps_to_move)
{
        int steps_left = abs(steps_to_move);  // how many steps to take

        // determine direction based on whether steps_to_mode is + or -:
        if (steps_to_move > 0) {this->direction = 1;}
        if (steps_to_move < 0) {this->direction = 0;}


        // decrement the number of steps, moving one step each time:
        while(steps_left > 0) {
        // move only if the appropriate delay has passed:
        if (millis() - this->last_step_time >= this->step_delay) {
            // get the timeStamp of when you stepped:
            this->last_step_time = millis();
            // increment or decrement the step number,
            // depending on direction:
            if (this->direction == 1) {
              this->step_number++;
              if (this->step_number == this->number_of_steps) {
                this->step_number = 0;
              }
            }
            else {
              if (this->step_number == 0) {
                this->step_number = this->number_of_steps;
              }
              this->step_number--;
            }
            // decrement the steps left:
            steps_left--;
            // step the motor to step number 0, 1, 2, or 3:
            stepMotor(this->step_number % 4);
          }
        }
}
```

6.10 TLC5940库

TLC5940库是针对TI公司的TLC5940NT PWM LED驱动芯片控制而推出的扩展库。该芯

片可提供16路PWM输出，最大30MHz数据传输率，多用于全彩LED驱动或伺服电机控制。关于TLC5940芯片的控制方式可参考5.5.5节。

6.10.1 Tlc5940类的定义

TLC5940库不属于Arduino的基本库，需要在网站http://arduino.cc/en/Reference/Libraries上单独下载，使用TLC5940库需占用Arduino的引脚3（对应GSCLK）、9（对应XLAT）、10（对应BLANK）、11（对应SIN）和13（对应SCLK）。库中定义了一个Tlc5940类，类的定义如下：

```
class Tlc5940
{
  public:
    void init(uint16_t initialValue = 0);
    void clear(void);
    uint8_t update(void);
    void set(TLC_CHANNEL_TYPE channel, uint16_t value);
    uint16_t get(TLC_CHANNEL_TYPE channel);
    void setAll(uint16_t value);
};
```

这里要特别说明一下，在Tlc5940类中有16个通道模拟量数据的存储形式，由于TLC5940驱动芯片的每一个通道为12bit的PWM，而一个字节只有8bit，所以一个通道的PWM值必须放在两个字节中，这就需要32个字节（16×2）来存储16个通道的数据。这样的存储方式相当于是每两个字节就有4bit是浪费的，所以在Tlc5940类中对这种存储方式进行了改进，让每两个通道使用3个字节，这样就只需要24个字节（16÷2×3）来存储16个通道的数据。又因为在数据的传输过程中所有数据均是MSB在前，所以数据的存储也是从高到低的。以通道0和通道1为例，使用的是最后3个字节来存储数据，最后一个字节存储的是通道0的低8位，倒数第二个字节存储的是通道1的低4位和通道0的高4位，倒数第3个字节存储的是通道1的高8位。Tlc5940类中定义了一个数组GSData来存储这些数据。

```
/** Packed grayscale data, 24 bytes (16 * 12 bits) per TLC.

    Format: Lets assume we have 2 TLCs, A and B, daisy-chained with the SOUT of
    A going into the SIN of B.
    - byte 0: upper 8 bits of B.15
    - byte 1: lower 4 bits of B.15 and upper 4 bits of B.14
    - byte 2: lower 8 bits of B.14
    - ...
    - byte 23: lower 8 bits of B.0

    \note Normally packing data like this is bad practice.  But in this
```

```
        situation, shifting the data out is really fast because the format of
        the array is the same as the format of the TLC's serial interface.
*/
uint8_t tlc_GSData[NUM_TLCS * 24];
```

注意：由于TLC5940驱动芯片支持级联，所以数组GSData定义中的宏定义NUM_TLCS表示实际应用中使用的芯片个数。

6.10.2 init( )

init函数用于对类的对象初始化。

❑ 返回值：无。

❑ 参数：

initialValue：表示16路输出的初始值。

函数原型如下：

```
void Tlc5940::init(uint16_t initialValue)
{
    /**************************************************
                                        Pin Setup
    **************************************************/
    XLAT_DDR |= _BV(XLAT_PIN);
    BLANK_DDR |= _BV(BLANK_PIN);
    GSCLK_DDR |= _BV(GSCLK_PIN);

    BLANK_PORT |= _BV(BLANK_PIN);

    tlc_shift8_init();

    setAll(initialValue);
    update();
    disable_XLAT_pulses();
    clear_XLAT_interrupt();
    tlc_needXLAT = 0;
    pulse_pin(XLAT_PORT, XLAT_PIN);

   /**************************************************
                                        Timer Setup
    **************************************************/

    /* Timer 1 - BLANK / XLAT */
    TCCR1A = _BV(COM1B1);  // non inverting, output on OC1B, BLANK
```

```
    TCCR1B = _BV(WGM13);// Phase/freq correct PWM, ICR1 top
    OCR1A = 1;          // duty factor on OC1A, XLAT is inside BLANK
    OCR1B = 2;          // duty factor on BLANK (larger than OCR1A (XLAT))
    ICR1 = TLC_PWM_PERIOD; // see tlc_config.h

    /* Timer 2 - GSCLK */
#ifdef TLC_ATMEGA_8_H
    TCCR2  = _BV(COM20) // set on BOTTOM, clear on OCR2A (non-inverting),
           | _BV(WGM21);  // output on OC2B, CTC mode with OCR2 top
    OCR2   = TLC_GSCLK_PERIOD / 2; // see tlc_config.h
    TCCR2 |= _BV(CS20);        // no prescale, (start pwm output)
#else
    TCCR2A = _BV(COM2B1)// set on BOTTOM, clear on OCR2A (non-inverting),
                        // output on OC2B
           | _BV(WGM21)        // Fast pwm with OCR2A top
           | _BV(WGM20);       // Fast pwm with OCR2A top
    TCCR2B = _BV(WGM22);       // Fast pwm with OCR2A top
    OCR2B = 0;              // duty factor (as short a pulse as possible)
    OCR2A = TLC_GSCLK_PERIOD; // see tlc_config.h
    TCCR2B |= _BV(CS20);       // no prescale, (start pwm output)
#endif
    TCCR1B |= _BV(CS10);       // no prescale, (start pwm output)
    update();
}
```

6.10.3 update( )

update函数的作用是把数组GSData中的数据发送出去，更新TLC5940驱动芯片的输出。

❑ 返回值：

uint8类型，表示数据是否成功发送。

❑ 参数：无。

函数原型如下：

```
uint8_t Tlc5940::update(void)
{
    if (tlc_needXLAT) {
        return 1;
    }
    disable_XLAT_pulses();
    if (firstGSInput) {
        // adds an extra SCLK pulse unless we've just set dot-correction data
        firstGSInput = 0;
```

```
    } else {
        pulse_pin(SCLK_PORT, SCLK_PIN);
    }
    uint8_t *p = tlc_GSData;
    while (p < tlc_GSData + NUM_TLCS * 24) {
        tlc_shift8(*p++);
        tlc_shift8(*p++);
        tlc_shift8(*p++);
    }
    tlc_needXLAT = 1;
    enable_XLAT_pulses();
    set_XLAT_interrupt();
    return 0;
}
```

6.10.4 set( )

set函数的作用是将需要设定的通道的模拟量数值存储到数组GSData中，等待发送数据。

❑ 返回值：无。

❑ 参数：

channel：通道号，级联的情况下通道号是累加的；

value：需写入的通道模拟量数值。

函数原型如下：

```
void Tlc5940::set(TLC_CHANNEL_TYPE channel, uint16_t value)
{
    TLC_CHANNEL_TYPE index8 = (NUM_TLCS * 16 - 1) - channel;
    uint8_t *index12p = tlc_GSData + ((((uint16_t)index8) * 3) >> 1);
    if (index8 & 1) { // starts in the middle
                      // first 4 bits intact | 4 top bits of value
        *index12p = (*index12p & 0xF0) | (value >> 8);
                      // 8 lower bits of value
        *(++index12p) = value & 0xFF;
    } else { // starts clean
                      // 8 upper bits of value
        *(index12p++) = value >> 4;
                      // 4 lower bits of value | last 4 bits intact
        *index12p = ((uint8_t)(value << 4)) | (*index12p & 0xF);
    }
}
```

6.10.5 get( )

get函数的作用是获取指定通道的模拟量数值。

❑ 返回值：

uint16类型，表示所获取的通道的模拟量数值，由于数据是12bit的，所以返回值是uint16类型。

❑ 参数：

channel：需要获取模拟量数值的通道号。

函数原型如下：

```
uint16_t Tlc5940::get(TLC_CHANNEL_TYPE channel)
{
    TLC_CHANNEL_TYPE index8 = (NUM_TLCS * 16 - 1) - channel;
    uint8_t *index12p = tlc_GSData + ((((uint16_t)index8) * 3) >> 1);
    return (index8 & 1)? // starts in the middle
            (((uint16_t)(*index12p & 15)) << 8) | // upper 4 bits
            *(index12p + 1)                       // lower 8 bits
        : // starts clean
            (((uint16_t)(*index12p)) << 4) | // upper 8 bits
            ((*(index12p + 1) & 0xF0) >> 4); // lower 4 bits
    // that's probably the ugliest ternary operator I've ever created.
}
```

6.10.6 setAll( )

setAll函数的作用是把所有通道的模拟量数值都设置为同一个数值，等待发送数据。

❑ 返回值：无。

❑ 参数：

value：需写入的通道模拟量数值。

函数原型如下：

```
void Tlc5940::setAll(uint16_t value)
{
    uint8_t firstByte = value >> 4;
    uint8_t secondByte = (value << 4) | (value >> 8);
    uint8_t *p = tlc_GSData;
    while (p < tlc_GSData + NUM_TLCS * 24) {
        *p++ = firstByte;
        *p++ = secondByte;
        *p++ = (uint8_t)value;
    }
}
```

6.10.7 clear( )

clear函数的作用是把所有通道的模拟量数值都设置为0。

❑ 返回值：无。

❑ 参数：无。

函数原型如下：

```
void Tlc5940::clear(void)
{
    setAll(0);
}
```

提示：set函数、setAll函数以及clear函数都只是对数组GSData的数据进行了更新，要达到更新TLC5940驱动芯片输出的目的，在调用以上3个函数后，均需要调用update函数。

6.11 OneWire库

单总线（One-Wire）是Dallas公司的一项特有的总线技术，它采用单根信号线实现数据的双向传输，具有节省I/O口资源、结构简单、便于扩展和维护等优点。

One-Wire适用于单个主机的系统，能够控制一个或多个从机设备。OneWire库就是针对单总线推出的扩展库。

6.11.1 单总线的结构

由于单总线系统中只使用一根信号线进行双向的数据传输，所以总线上的每个节点都必须是漏级或集电极开路的，这样设备在不发送数据时将释放数据总线，以便其他设备使用总线。单总线要求外接一个约5kΩ的上拉电阻以保证总线在闲置状态为高电平。无论是什么原因，如果在传输过程中需要暂时挂起，且要求传输过程还能够继续，则总线必须处于空闲状态。位传输之间的恢复时间没有限制，只要总线在恢复期间处于空闲状态。如果总线保持低电平超过480μs，总线上所有的器件将复位。

6.11.2 单总线控制方式

在单总线系统中，主机对从机的控制操作简单来说可以分为3步。

第一步，初始化。初始化的过程是由主机发出的复位脉冲和从机响应的应答脉冲组成。应答脉冲使主机知道总线上有从机设备，且准备就绪。

第二步，ROM指令。单总线器件内部都有一个64bit的只读存储区（ROM），ROM中的数据是唯一的，用以在总线中区分不同的节点。其中前8bit是单线系列编码（比如：

DS18B20的编码是19H），接下来的48bit是器件唯一的序列号，最后8bit是以上56bit数据的CRC校验码。ROM指令的目的是使主机指定某个从机设备或获取总线上有多少个从机设备及其设备类型。

提示：ROM指令共有5条，每条命令的长度为8位，它们分别是读ROM数据、指定匹配芯片、跳跃ROM、芯片搜索、报警芯片搜索。具体的ROM指令功能及作用这里就不介绍了。

第三步，操作指令。在主机指定某个从机后，接着就发送操作指令对器件进行控制。不同类型的单总线器件其操作指令可能会有一些差异。

注意：每次访问单总线器件，必须严格遵守这3步操作，否则单总线器件不会响应主机。但芯片搜索命令和报警芯片搜索命令例外，在执行两者中的任意一条命令之后，主机不能执行其后的操作命令，必须返回第一步重新执行。

6.11.3 单总线信号形式

单总线系统中定义了几种信号形式，所有单总线器件必须严格遵守这些信号形式，以保证数据的完整性。这些信号包括：复位脉冲、应答脉冲、写0、写1、读0和读1。除了应答脉冲以外，都是由主机发出同步信号。单总线中发送的所有的命令和数据都是低位在前。

1. 复位脉冲和应答脉冲

单总线上的所有通信都是以初始化序列开始，包括主机发出的复位脉冲和从机的应答脉冲，如图6.13所示。

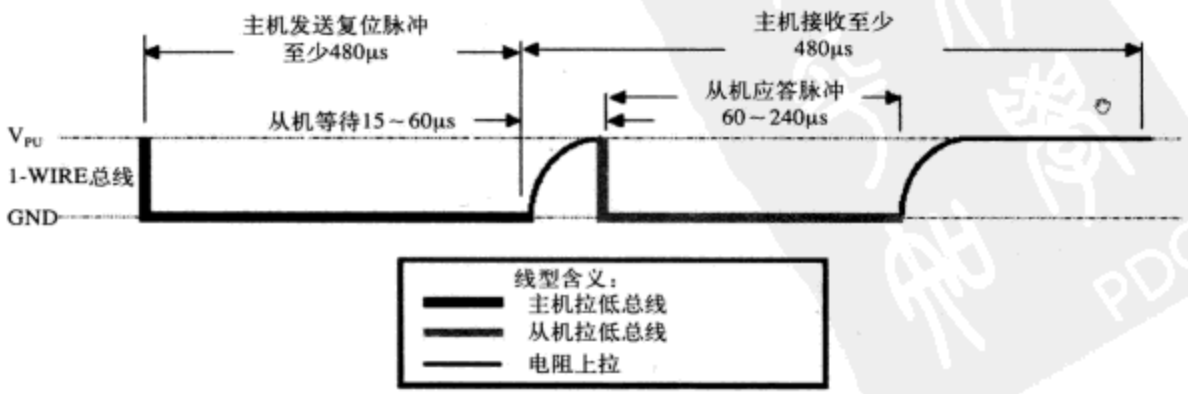


图6.13 复位脉冲和应答脉冲

当从机发出响应主机的应答脉冲时，即向主机表明它处在总线上，且准备就绪。在主机初始化过程中，主机通过拉低单总线至少480μs，以产生复位脉冲。接着主机释放总线，并进入接收模式。当释放总线后，5kΩ上拉电阻将单总线拉高，在单总线器件检测到上升沿后，

延时15～60μs，接着通过拉低总线60～240μs以产生应答脉冲。

2. 写时隙

在写时隙期间，主机向单总线器件写入数据，一个时隙总线只能传输1bit数据。单总线中存在两种写时隙：写1和写0。主机采用写1时隙向从机写入1，而采用写0时隙向从机写入0。所有写时隙至少需要60μs，且在两次独立的写时隙之间至少需要1μs的恢复时间。两种写时隙均起始于主机拉低总线。产生写1时隙的方式，主机在拉低总线后，接着必须在15μs之内释放总线，由5kΩ上拉电阻将总线拉至高电平；而产生写0时隙的方式，在主机拉低总线后，只需在整个时隙期间保持低电平即可（至少60μs）。写1和写0时隙如图6.14所示。

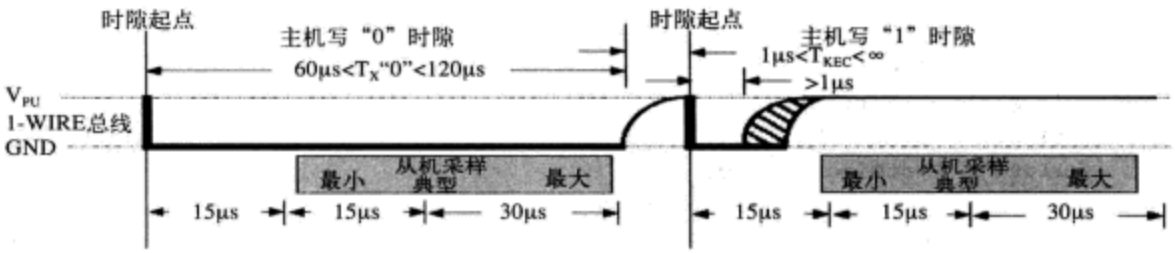


图6.14 单总线写时隙

在写时隙起始后15～60μs期间，单总线器件采样总线电平状态。如果在此期间采样为高电平，则将逻辑1写入该器件；如果为0，则写入逻辑0。

3. 读时隙

在读时隙期间，主机读入来自从机的数据，一个时隙总线只能传输1bit数据。单总线器件仅在主机发出读时隙时，才向主机传输数据，所有在主机发出读数据命令后，必须马上产生读时隙，以便从机能够传输数据。所有读时隙至少需要60μs，且在两次独立的读时隙之间至少需要1μs的恢复时间。每个读时隙都由主机发起，至少拉低总线1μs。在主机发起读时隙之后，单总线器件才开始在总线上发送0或1。若从机发送1，则保持总线为高电平；若发送0，则拉低总线，如图6.15所示。

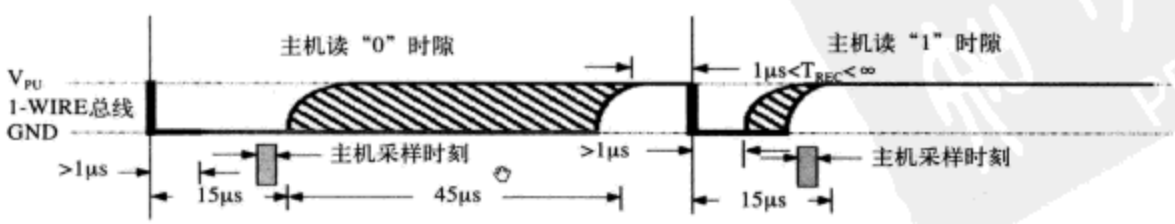


图6.15 单总线读时隙

当发送0时，从机在该时隙结束后释放总线，由5kΩ上拉电阻将总线拉回至空闲高电平状态。从机发出的数据在起始时隙之后，保持有效时间15μs，因而，主机在读时隙期间必须

释放总线，并且在时隙起始后的15μs之内采样总线状态。

6.11.4 OneWire类

OneWire库不属于Arduino的基本库，需要在网站http://www.arduino.cc/en/Reference/Libraries上单独下载，库中定义了一个OneWire类，类的定义如下：

```
class OneWire
{
  private:
    uint8_t bitmask;
    volatile uint8_t *baseReg;

#if ONEWIRE_SEARCH
    // global search state
    unsigned char ROM_NO[8];
    uint8_t LastDiscrepancy;
    uint8_t LastFamilyDiscrepancy;
    uint8_t LastDeviceFlag;
#endif

  public:
    OneWire( uint8_t pin);

    // Perform a 1-Wire reset cycle. Returns 1 if a device responds
    // with a presence pulse.  Returns 0 if there is no device or the
    // bus is shorted or otherwise held low for more than 250uS
    uint8_t reset(void);

    // Issue a 1-Wire rom select command, you do the reset first.
    void select( uint8_t rom[8]);

    // Issue a 1-Wire rom skip command, to address all on bus.
    void skip(void);

    // Write a byte. If 'power' is one then the wire is held high at
    // the end for parasitically powered devices. You are responsible
    // for eventually depowering it by calling depower() or doing
    // another read or write.
    void write(uint8_t v, uint8_t power = 0);

    // Read a byte.
    uint8_t read(void);
```

```
    // Write a bit. The bus is always left powered at the end, see
    // note in write() about that.
    void write_bit(uint8_t v);

    // Read a bit.
    uint8_t read_bit(void);

    // Stop forcing power onto the bus. You only need to do this if
    // you used the 'power' flag to write() or used a write_bit() call
    // and aren't about to do another read or write. You would rather
    // not leave this powered if you don't have to, just in case
    // someone shorts your bus.
    void depower(void);

#if ONEWIRE_SEARCH
    // Clear the search state so that if will start from the beginning again.
    void reset_search();

    // Look for the next device. Returns 1 if a new address has been
    // returned. A zero might mean that the bus is shorted, there are
    // no devices, or you have already retrieved all of them.  It
    // might be a good idea to check the CRC to make sure you didn't
    // get garbage.  The order is deterministic. You will always get
    // the same devices in the same order.
    uint8_t search(uint8_t *newAddr);
#endif

#if ONEWIRE_CRC
    // Compute a Dallas Semiconductor 8 bit CRC, these are used in the
    // ROM and scratchpad registers.
    static uint8_t crc8( uint8_t *addr, uint8_t len);

    #if ONEWIRE_CRC16
    // Compute a Dallas Semiconductor 16 bit CRC. Maybe. I don't have
    // any devices that use this so this might be wrong. I just copied
    // it from their sample code.
    static unsigned short crc16(unsigned short *data, unsigned short len);
    #endif
#endif
};
```

6.11.5 构造函数

OneWire类的构造函数用于指定Arduino应用于单总线的引脚。

❑ 参数：

pin：表示应用于单总线的引脚。

构造函数原型如下：

```
OneWire::OneWire(uint8_t pin)
{
     bitmask =  digitalPinToBitMask(pin);
     baseReg = portInputRegister(digitalPinToPort(pin));
     #if ONEWIRE_SEARCH
     reset_search();
     #endif
}
```

6.11.6 reset( )

reset函数的作用是初始化单总线。

❑ 返回值：

uint8类型，表示初始化结构。返回1则表示单总线上有从机设备，且准备就绪；否则返回0。

❑ 参数：无。

函数原型如下：

```
uint8_t OneWire::reset(void)
{
     uint8_t mask=bitmask;
     volatile uint8_t *reg asm("r30") = baseReg;
     uint8_t r;
     uint8_t retries = 125;

     cli();
     DIRECT_MODE_INPUT(reg, mask);
     sei();
     // wait until the wire is high... just in case
     do {
          if (--retries == 0) return 0;
          delayMicroseconds(2);
     } while ( !DIRECT_READ(reg, mask));

     cli();
```

```
    DIRECT_WRITE_LOW(reg, mask);
    DIRECT_MODE_OUTPUT(reg, mask);      // drive output low
    sei();
    delayMicroseconds(500);
    cli();
    DIRECT_MODE_INPUT(reg, mask);      // allow it to float
    delayMicroseconds(80);
    r = !DIRECT_READ(reg, mask);
    sei();
    delayMicroseconds(420);
    return r;
}
```

6.11.7 write_bit( )

write_bit函数的功能是写时隙，即写1或写0。

❑ 返回值：无。

❑ 参数：

v：写入的数据。参数的bit0为0则为写0，bit1则为写1。

函数原型如下：

```
void OneWire::write_bit(uint8_t v)
{
    uint8_t mask=bitmask;
    volatile uint8_t *reg asm("r30") = baseReg;

    if (v & 1) {
        cli();
        DIRECT_WRITE_LOW(reg, mask);
        DIRECT_MODE_OUTPUT(reg, mask);      // drive output low
        delayMicroseconds(10);
        DIRECT_WRITE_HIGH(reg, mask);      // drive output high
        sei();
        delayMicroseconds(55);
    }
    else
    {
        cli();
        DIRECT_WRITE_LOW(reg, mask);
        DIRECT_MODE_OUTPUT(reg, mask);      // drive output low
        delayMicroseconds(65);
        DIRECT_WRITE_HIGH(reg, mask);      // drive output high
```

```
            sei();
            delayMicroseconds(5);
        }
    }
```

6.11.8 read_bit( )

read_bit函数的功能是读时隙，即读1或读0。

❑ 返回值：

uint8类型，表示读出的数据。

❑ 参数：无。

函数原型如下：

```
uint8_t OneWire::read_bit(void)
{
    uint8_t mask=bitmask;
    volatile uint8_t *reg asm("r30") = baseReg;
    uint8_t r;

    cli();
    DIRECT_MODE_OUTPUT(reg, mask);
    DIRECT_WRITE_LOW(reg, mask);
    delayMicroseconds(3);
    DIRECT_MODE_INPUT(reg, mask);   // let pin float, pull up will raise
    delayMicroseconds(9);
    r = DIRECT_READ(reg, mask);
    sei();
    delayMicroseconds(53);
    return r;
}
```

6.11.9 write( )

write函数的功能是发送1 byte的数据。

❑ 返回值：无。

❑ 参数：

v：表示要发送的数据；

power：可选参数，默认值为0。该参数表示在空闲状态引脚是否置高。若参数值为0则空闲状态为漏级或集电极开路；若为1则表示空闲状态引脚置高，以保证寄生性单总线器件的正常工作。

函数原型如下：

```
void OneWire::write(uint8_t v, uint8_t power /* = 0 */) {
    uint8_t bitMask;

    for (bitMask = 0x01; bitMask; bitMask <<= 1) {
     OneWire::write_bit( (bitMask & v)?1:0);
    }
    if ( !power) {
     cli();
     DIRECT_MODE_INPUT(baseReg, bitmask);
     DIRECT_WRITE_LOW(baseReg, bitmask);
     sei();
    }
}
```

6.11.10 read( )

read函数的功能是读取1byte的数据。

❑ 返回值：

uint8类型，表示读出的数据。

❑ 参数：无。

函数原型如下：

```
uint8_t OneWire::read() {
    uint8_t bitMask;
    uint8_t r = 0;

    for (bitMask = 0x01; bitMask; bitMask <<= 1) {
       if ( OneWire::read_bit()) r |= bitMask;
    }
    return r;
}
```

6.11.11 select( )

select函数的作用是让主机指定某一个从机。

❑ 返回值：无。

❑ 参数：

rom[8]：表示将指定从机的8byte的ROM数据。

函数原型如下：

```
void OneWire::select( uint8_t rom[8])
```

```
{
    int i;

    write(0x55);            // Choose ROM

    for( i = 0; i < 8; i++) write(rom[i]);
}
```

6.11.12 skip( )

skip函数的作用是执行跳跃ROM指令。

❑ 返回值：无。

❑ 参数：无。

函数原型如下：

```
void OneWire::skip()
{
    write(0xCC);            // Skip ROM
}
```

提示：OneWire类中的其他成员函数多是在以上的函数中进行调用，包括CRC校验、芯片搜索等，这里就不一一介绍了。

6.12 XBee库

XBee库是针对美国DIGI公司的zigbee模块XBee而推出的扩展库，XBee是一种远距离低功耗的数传模块，频段有2.4G、900M、868M 3种，同时可兼容802.15.4协议。模块内置协议栈，可组mesh网络，每个模块都可以作为路由节点、协调器以及终端节点。模块通过串行数据传输接口控制，Arduino通过引脚0（RX）和引脚1（TX）实现对XBee模块的控制。

6.12.1 XBee类定义

XBee库不属于Arduino的基本库，需要在网站http://www.arduino.cc/en/Reference/Libraries上单独下载，库中定义了一个XBee类，类的定义如下：

```
class XBee
{
public:
    XBee();
    void setSerial(HardwareSerial serial);
```

```
        void readPacket();
    bool readPacket(int timeout);
    void readPacketUntilAvailable();

    void begin(long baud);

    void getResponse(XBeeResponse &response);
    XBeeResponse& getResponse();

    void send(XBeeRequest &request);

    uint8_t getNextFrameId();

private:
    void sendByte(uint8_t b, bool escape);
    void resetResponse();
    XBeeResponse _response;
    bool _escape;
    // current packet position for response.
    //just a state variable for packet parsing and has no relevance for the response otherwise
    uint8_t _pos;
    // last byte read
    uint8_t b;
    uint8_t _checksumTotal;
    uint8_t _nextFrameId;
    // buffer for incoming RX packets. holds only the api specific frame data,
    //starting after the api id byte and prior to checksum
    uint8_t _responseFrameData[MAX_FRAME_DATA_SIZE];
};
```

XBee模块的配置方式有两种，分别是AP和ATI命令，可通过X-CTU以及Zigbee Operator这两款软件进行调试。AP和ATI命令就不介绍了，读者可以参考一下XBee模块的数据手册，这里本着应用为本的原则介绍一下几个常用的成员函数。

6.12.2 构造函数

XBee的构造函数用于初始化类的对象。

❑ 参数：无。

构造函数原型如下：

```
XBee::XBee()
{
    _pos = 0;
```

```
    _escape = false;
    _checksumTotal = 0;
    _nextFrameId = 0;

    _response.init();
    _response.setFrameData(_responseFrameData);
}
```

6.12.3 begin( )

begin函数的作用是设置串行数据接口的波特率，以实现对模块的控制。

❑ 返回值：无。

❑ 参数：

baud：表示设置的串行数据接口波特率。

函数原型如下：

```
void XBee::begin(long baud)
{
    Serial.begin(baud);
}
```

6.12.4 readPacket( )

readPacket函数的作用是从模块中获取数据包。

❑ 返回值：无。

❑ 参数：无。

函数原型如下：

```
void XBee::readPacket()
{
    // reset previous response
    if (_response.isAvailable() || _response.isError())
    {
            // discard previous packet and start over
            resetResponse();
    }

    while (Serial.available())
    {

            b = Serial.read();
```

```
        if (_pos > 0 && b == START_BYTE && ATAP == 2)
        {
        // new packet start before previous packeted completed
        _response.setErrorCode(UNEXPECTED_START_BYTE);
        return;
}

        (_pos > 0 && b == ESCAPE)
        {
                if (Serial.available())
                {
                        b = Serial.read();
                        b = 0x20 ^ b;
                }
                else
                {
                        // escape byte.  next byte will be
                        _escape = true;
                        continue;
                }
        }

        if (_escape == true)
        {
                b = 0x20 ^ b;
                _escape = false;
        }

        // checksum includes all bytes starting with api id
        if (_pos >= API_ID_INDEX)
        {
                _checksumTotal+= b;
        }

        switch(_pos)
        {
                case 0:
                        if (b == START_BYTE)
                        {
                                _pos++;
                        }
                        break;
```

```
case 1:
        // length msb
        _response.setMsbLength(b);
        _pos++;

        break;
case 2:
        // length lsb
        _response.setLsbLength(b);
        _pos++;

        break;
case 3:
        _response.setApiId(b);
        _pos++;

        break;
default:
        // starts at fifth byte

        if (_pos > MAX_FRAME_DATA_SIZE)
        {
                // exceed max size.  should never occur
                _response.setErrorCode(
                PACKET_EXCEEDS_BYTE_ARRAY_LENGTH);
                return;
        }

        // check if we're at the end of the packet
        // packet length does not include start, length,
        //or checksum bytes, so add 3
        if (_pos == (_response.getPacketLength() + 3))
        {
                // verify checksum

                if ((_checksumTotal & 0xff) == 0xff)
                {
                        _response.setChecksum(b);
                        _response.setAvailable(true);

                        _response.setErrorCode(NO_ERROR);
                }
                else
```

```
                                {
                                // checksum failed
                                _response.setErrorCode(CHECKSUM_FAILURE);
                                }
                                _response.setFrameLength(_pos - 4);

                                // reset state vars
                                _pos = 0;

                                _checksumTotal = 0;

                                return;
                        }
                        else
                        {
                                _response.getFrameData()[_pos - 4] = b;
                                _pos++;
                        }
                }
        }
}
```

6.12.5 send( )

send函数的作用是给模块发送一个请求。

❑ 返回值：无。

❑ 参数：

request：XBeeRequest类的对象。

函数原型如下：

```
void XBee::send(XBeeRequest &request) {
    // the new new deal

    sendByte(START_BYTE, false);

    // send length
    uint8_t msbLen = ((request.getFrameDataLength() + 2) >> 8) & 0xff;
    uint8_t lsbLen = (request.getFrameDataLength() + 2) & 0xff;

    sendByte(msbLen, true);
    sendByte(lsbLen, true);

    // api id
```

```
    sendByte(request.getApiId(), true);
    sendByte(request.getFrameId(), true);

    uint8_t checksum = 0;

    // compute checksum, start at api id
    checksum+= request.getApiId();
    checksum+= request.getFrameId();


    for (int i = 0; i < request.getFrameDataLength(); i++)
    {
        sendByte(request.getFrameData(i), true);
        checksum+= request.getFrameData(i);
    }

    // perform 2s complement
    checksum = 0xff - checksum;

    // send checksum
    sendByte(checksum, true);

    // send packet
    Serial.flush();
}
```

6.13 创建自己的库

除了前面介绍的Arduino扩展库外，在网站http://www.arduino.cc/en/Reference/Libraries上还能下载到更多的库，内容涉及方方面面，包括PS2键盘、GLCD、消息系统、音色声调等。这些库当中很多是Arduino爱好者们自己编写并放在网络上共享的。

同样我们也能够创建自己的库，如果觉得不错也可以放在网络上与更多的人分享。Arduino以及关于Arduino的一切都是开源的。

6.13.1 库的功能——Morse

在本节中，我们将建立一个自己的库，实现摩尔斯译码器的功能。摩尔斯码（Morse）是一种时通时断的信号代码，这种信号代码通过不同的排列顺序来表达不同的英文字母、数字和标点符号等。用一个电键就可以敲击出点、划以及中间的停顿。这种代码可以是电报电线里的电子脉冲，也可以是一种机械的或视觉的信号（比如闪光）。

摩尔斯码用两种“符号”用来表示字元：划（-）和点（·），或分别叫嗒（dah）和滴（dit）或长和短。点的长度决定了发报的速度，并且被当做发报时间参考。划一般是3个点的长度；点划之间的间隔是一个点的长度；字元之间的间隔是3个点的长度；单词之间的间隔是7个点的长度。摩尔斯电码表如图6.16所示。

字符	电码符号	字符	电码符号	字符	电码符号
A	• —	N	— •	1	• — — — —
B	— • • •	O	— — —	2	• • — — —
C	— • — •	P	• — — •	3	• • • — —
D	— • •	Q	— — • —	4	• • • • —
E	•	R	• — •	5	• • • • •
F	• • — •	S	• • •	6	— • • • •
G	— — •	T	—	7	— — • • •
H	• • • •	U	• • —	8	— — — • •
I	• •	V	• • • —	9	— — — — •
J	• — — —	W	• — —	0	— — — — —
K	— • —	X	— • • —	?	• • — — • •
L	• — • •	Y	— • — —	/	— • • — •
M	— —	Z	— — • •	( )	— • — — • —
				—	— • • • • —
				•	• — • — • —

图6.16 摩尔斯电码表

我们首先看看如何在Arduino的环境下用程序实现发送摩尔斯码的功能。假定摩尔斯码通过引脚13的LED发出，发送的内容是国际摩尔斯码救难信号SOS，由图6.11可以查到所发送的符号是 · · · – – – · · · 。程序内容如下：

```
/*************************************************
     摩尔斯码例程——发送SOS

     摩尔斯码通过引脚13的LED发出
     所发送的符号是 · · · – – – · · ·

     created 2011
       by Nille
       Email: chenille@126.com

  This example code is in the public domain.
 *************************************************/

/*************************************************
                初始化部分——setup函数
*************************************************/
void setup()
{
```

```
    pinMode(13, OUTPUT);        //将引脚13设为输出
}

/*************************************************
          执行部分——loop函数
*************************************************/
void loop()
{
   //发送S
       dot();
       dot();
       dot();

       //发送O
       dash();
       dash();
       dash();

       //发送S
       dot();
       dot();
       dot();
       .
       //延时3s
       delay(3000);
 }

/*************************************************
滴（dit）发送子函数
函数功能：发出一个滴，时长0.25s
入口参数：无
出口参数：无
*************************************************/
void dot()
{
      digitalWrite(13, HIGH);
      delay(250);
      digitalWrite(13, LOW);
      delay(250);
}
/*************************************************
嗒（dah）发送子函数
```

```
函数功能：发出一个嗒，时长1s
入口参数：无
出口参数：无
*************************************************/
void dash()
{
      digitalWrite(13, HIGH);
      delay(1000);
      digitalWrite(13, LOW);
      delay(250);
}
```

运行程序后，会在Arduino上的LED看到闪烁的SOS信号，如果在引脚13连接一个蜂鸣器还可以听到发出的SOS信号。接下来我们把它封装成一个类，叫做MorseCode。

6.13.2 MorseCode类的定义

MorseCode类的定义需要包含变量和成员函数两部分。变量定义为私有的（private），包括codePin。成员函数定义为公有的（public），包括dash()、dot()、构造函数以及字符转译函数。由此得到MorseCode类的定义，如下：

```
class MorseCode
{
      private:                                    //定义为私有
              int     _codePin;                   //定义用作输出的引脚
      public:                                     //定义为公有
      Morse(int _pin);                            //构造函数
         void dot();                              //滴（dit）发送子函数
         void dash();                             //嗒（dah）发送子函数
         void transfor(char _code);               //字符转译函数
};
```

类的定义一般放在一个后缀名为.h的文件中，称为头文件，头文件作为一种包含功能函数、数据接口声明的载体文件，用于保存程序的声明（declaration）。头文件的主要作用在于调用库功能，对各个被调用函数给出一个描述，其本身不包含程序的逻辑实现代码，它只起描述性作用，告诉应用程序通过相应途径寻找相应功能函数的真正逻辑实现代码。用户程序只需要按照头文件中的接口声明来调用库功能，编译器会从库中提取相应的代码。

文件除了包括类的定义之外，还要包括预处理块，用于在编译器在编译代码时进行判断与选择，最后在头文件最前端添加一些库文件的信息注释。完整的头文件内容如下，这里将其命名为MorseCode.h。

```
/*********************************************
     文件名：MorseCode.h
     摩尔斯码库

     created 2011
       by Nille
       Email: chenille@126.com

  This example code is in the public domain.
 *********************************************/
#ifndef MorseCode_h
#define MorseCode_h

class MorseCode
{
      private:                                    //定义为私有
              int     _codePin;                   //定义用作输出的引脚
      public:                                     //定义为公有
      MorseCode (int _pin);                       //构造函数
         void dot();                              //滴（dit）发送子函数
         void dash();                             //嗒（dah）发送子函数
         void transfor(int _code);                //字符转译函数
};

#endif
```

其中预处理块部分是为了防止头文件被重复引用，结构如下：

```
#ifndef xxxx_h
#define xxxx_h
......
#endif
```

6.13.3 MorseCode类的成员函数

程序的实现放在文件名为MorseCode（与头文件名相同）、后缀名为.cpp的文件中，包括构造函数以及其他成员函数。

MorseCode类的构造函数主要实现初始化摩尔斯码信号输出引脚。完整构造函数如下：

```
/*********************************************
MorseCode类构造函数
函数功能：定义摩尔斯码输出引脚为输出
入口参数：_pin,表示摩尔斯码输出引脚
*********************************************/
```

```
MorseCode::MorseCode (int _pin)
{
    pinMode( _pin, OUTPUT);
    _codePin = _pin;
}
```

dot()和dash()函数的实现如下：

```
/*************************************************
滴（dit）发送子函数
函数功能：发出一个滴，时长0.25s
入口参数：无
出口参数：无
*************************************************/
void MorseCode::dot()
{
    digitalWrite(_codePin, HIGH);
    delay(250);
    digitalWrite(_codePin, LOW);
    delay(250);
}

/*************************************************
嗒（dah）发送子函数
函数功能：发出一个嗒，时长1s
入口参数：无
出口参数：无
*************************************************/
void MorseCode::dash()
{
    digitalWrite(_codePin, HIGH);
    delay(1000);
    digitalWrite(_codePin, LOW);
    delay(250);
}
```

transfor函数的作用是将字符转换为摩尔斯码，程序用switch语句实现，代码如下：

```
/*************************************************
transfor子函数
函数功能：字符转译函数，将字符转换为摩尔斯码
入口参数：需转译的字符
出口参数：无
*************************************************/
void MorseCode::transfor(char _code)
```

```
{
    //判断需转译的字符，转换为摩尔斯码，参见图6.11
    switch( _code )
    {
        case 'A':
        case 'a':
            dot();
            dash();
            break;
        case 'B':
        case 'b':
            dash();
            dot();
            dot();
            dot();
            break;
        case 'C':
        case 'c':
            dash();
            dot();
            dash();
            dot();
            break;
        case 'D':
        case 'd':
            dash();
            dot();
            dot();
            break;
        case 'E':
        case 'e':
            dot();
            break;
        case 'F':
        case 'f':
            dot();
            dot();
            dash();
            dot();
            break;
        case 'G':
        case 'g':
            dash();
```

```
        dash();
        dot();
        break;
    case 'H':
    case 'h':
        dot();
        dot();
        dot();
        dot();
        break;
    case 'I':
    case 'i':
        dot();
        dot();
        break;
    case 'J':
    case 'j':
        dot();
        dash();
        dash();
        dash();
        break;
    case 'K':
    case 'k':
        dash();
        dot();
        dash();
        break;
    case 'L':
    case 'l':
        dot();
        dash();
        dot();
        dot();
        break;
    case 'M':
    case 'm':
        dash();
        dash();
        break;
    case 'N':
    case 'n':
        dash();
```

```
            dot();
            break;
        case 'O':
        case 'o':
            dash();
            dash();
            dash();
            break;
        case 'P':
        case 'p':
            dot();
            dash();
            dash();
            dot();
            break;
        case 'Q':
        case 'q':
            dash();
            dash();
            dot();
            dash();
            break;
        case 'R':
        case 'r':
            dot();
            dash();
            dot();
            break;
        case 'S':
        case 's':
            dot();
            dot();
            dot();
            break;
        case 'T':
        case 't':
            dash();
            break;
        case 'U':
        case 'u':
            dot();
            dot();
            dash();
```

```
        break;
    case 'V':
    case 'v':
        dot();
        dot();
        dot();
        dash();
        break;
    case 'W':
    case 'w':
        dot();
        dash();
        dash();
        break;
    case 'X':
    case 'x':
        dash();
        dot();
        dot();
        dash();
        break;
    case 'Y':
    case 'y':
        dash();
        dash();
        dot();
        dot();
        break;
    case '1':
        dot();
        dash();
        dash();
        dash();
        dash();
        break;
    case '2':
        dot();
        dot();
        dash();
        dash();
        dash();
        break;
    case '3':
```

```
            dot();
            dot();
            dot();
            dash();
            dash();
            break;
        case '4':
            dot();
            dot();
            dot();
            dot();
            dash();
            break;
        case '5':
            dot();
            dot();
            dot();
            dot();
            dot();
            break;
        case '6':
            dash();
            dot();
            dot();
            dot();
            dot();
            break;
        case '7':
            dash();
            dash();
            dot();
            dot();
            dot();
            break;
        case '8':
            dash();
            dash();
            dash();
            dot();
            dot();
            break;
        case '9':
            dash();
```

```
            dash();
            dash();
            dash();
            dot();
            break;
        case '0':
            dash();
            dash();
            dash();
            dash();
            dash();
            break;
        default:
            break;
    }
}
```

另外，在MorseCode.cpp的文件中还应包含头文件MorseCode.h、WProgram.h和string.h以及一些文件的信息注释，如下所示：

```
/*************************************************
    文件名：MorseCode.cpp
    摩尔斯码库

    created 2011
    by Nille
    Email: chenille@126.com

  This example code is in the public domain.
 *************************************************/
#include <WProgram.h>
#include "MorseCode.h"
#include "string.h"
```

编写了MorseCode.h和MorseCode.cpp两个文件后，为了能够在Arduino的开发环境中使用MorseCode库，需要在Arduino开发环境目录下的libraries文件夹中创建一个MorseCode文件夹，再将MorseCode.h和MorseCode.cpp两个文件放入MorseCode文件夹中，如图6.17所示。

到这里就完成了一个MorseCode库的建立，下一节就用这个MorseCode库来实现6.13.1节中发送国际摩尔斯码救难信号SOS的功能。

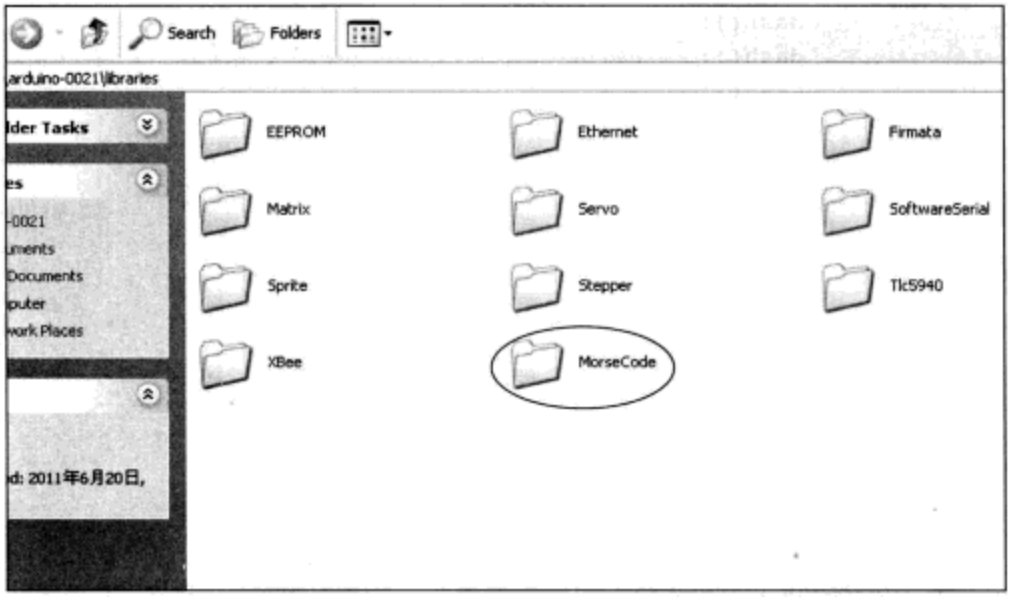


图6.17 创建MorseCode文件夹

6.13.4 MorseCode库的使用

要使用MorseCode库首先需要包含头文件MorseCode.h，同时生成一个MorseCode类的对象，接着就可以使用类的成员函数了。程序代码如下：

```
/*********************************************
     摩尔斯码例程——通过MorseCode库发送SOS

     摩尔斯码通过引脚13的LED发出
     所发送的符号是 · · · - - - · · ·

     created 2011
       by Nille
       Email: chenille@126.com

   This example code is in the public domain.
 *********************************************/

#include "MorseCode.h"          //包含头文件MorseCode.h


//定义MorseCode类的对象Morse，使用引脚13
MorseCode Morse(13);
```

```
/*************************************************
                初始化部分——setup函数
*************************************************/
void setup()
{
}

/*************************************************
                执行部分———loop函数
*************************************************/
void loop()
{

        Morse.transfor('S');        //发送S的摩尔斯码
        Morse.transfor('O');        //发送O的摩尔斯码
        Morse.transfor('S');        //发送S的摩尔斯码
        //延时3s
        delay(3000);

}
```

6.13.5 关键字的定义

如果在Arduino的开发环境中写入了上面的程序，就会发现setup、loop等关键字是橙色的，而我们的类的成员函数没有什么变化，依然是黑色的。其实Arduino开发环境提供了让类的成员函数也能用不同颜色显示以示区别的功能，不过这需要编写一个小的txt文件来完成。

这个txt文件的文件名是keywords.txt，也要放在MorseCode文件夹中。文件内容的格式如下所示：

```
dot   KEYWORD2
```

每行的开始是需要变换颜色的关键字名称，其后跟着一个tab键，最后是关键字所显示的颜色——KEYWORD1表示橙色，KEYWORD2表示褐色。MorseCode文件夹中的keywords.txt文件内容如下：

```
dot     KEYWORD2
dash    KEYWORD2
transfor     KEYWORD1
```

完成了keywords.txt文件的编写后，需要重新启动Arduino的开发环境才能识别这些关键字。

注意：关键字名称后面跟随的是一个tab键，而不是空格。

第7章 无线模块的应用

在第1章中说过在Arduino的家族中有一位Arduino BT，其本身包含了一个Bluegiga WT11蓝牙模块，支持蓝牙无线串行通信，没有USB接口，连接电脑或烧写程序均通过蓝牙适配器与Arduino BT连接实现无线程序下载与控制。无线控制可以让Arduino应用到更广泛的领域，本章将介绍4种应用于Arduino的无线模块，使Arduino Uno也具有无线数据传输功能。

7.1 APC220

APC220模块是一款半双工微功耗无线数据传输模块，其内部嵌入高速单片机和高性能射频芯片，创新地采用高效的循环交织纠检错编码，抗干扰和灵敏度大大提高。目前市场使用较多的无线门铃、无线键盘、工业控制、煤矿系统都能见到它的身影，很多电子爱好者也使用该模块进行产品的制作。该模块提供多个频道的选择，丰富便捷的软件编程设置功能，可设置7种速率和3种接口校验方式，对外为UART/TTL接口。该模块能够透明传输任何大小的数据，传输距离可达到1000m（开阔地可视距离）。模块一般成对使用，实物如图7.1所示（最下方为模块连接电脑的适配座）。

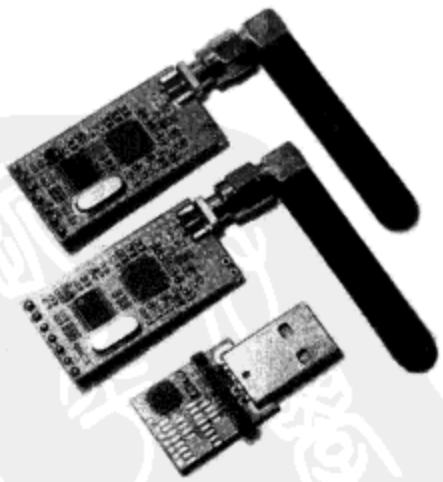


图7.1 APC220模块

7.1.1 APC220性能指标

模块主要的性能指标如下：

- ❑ 工作频率：415～455MHz（1KHz步进）
- ❑ 调制方式：GFSK
- ❑ 频率间隔：200KHz
- ❑ 发射功率：20mW（10级可调）
- ❑ 接收灵敏度：−117dBm@1200bps

- 空中传输速率：1200～19 200bps
- 接口速率：1200～57 600bps
- 接口校验方式：8E1/8N1/8O1
- 接口缓冲空间：512byte
- 工作湿度：10%～90%（无冷凝）
- 工作温度：−20～70℃
- 工作电压：3.3～5.5V
- 发射电流：≤35mA
- 接收电流：≤30mA
- 休眠电流：≤5μA
- 传输距离：1000m（开阔地可视距离）
- 尺寸大小：37mm×17mm×6.5mm（不含天线座和引脚插头）

APC220模块是一款多通道嵌入式无线数传模块，发射功率高达20mW，而仍然具有较低的功耗，体积37mm×17mm×6.5mm（不含天线座和引脚插头），为业内目前最小体积，非常方便客户嵌入系统内使用。

APC220模块创新地采用了高效的循环交织纠检错编码，最大可以纠24bit连续突发错误，其编码增益高达近3dBm，纠错能力和编码效率均达到业内的领先水平，远远高与一般的前向纠错编码，抗突发干扰和灵敏度都较大的改善。同时编码也包含可靠检错能力，能够自动滤除错误及虚假信息，真正实现了透明连接。所以APC220模块特别适合与在工业领域等强干扰的恶劣环境中使用。

512byte超大容量缓冲区，意味着用户在任何状态下都可以1次传输512byte的数据。当设置空中波特率大于串口波特率时，可看做传输数据长度不受限制。模块提供标准的UART/TTL接口，7种传输速率可选，分别是：1200/2400/4800/9600/19 200/38 400/57 600bps，同时具有3种接口校验方式。

传统无线模块使用跳线设置如串口速率、校验方式、频点等参数，这会带来接触不良、选项较少、不宜设置等诸多不便。APC220模块采用串口设置模块参数，具有丰富便捷的软件编程设置选项，包括频点、空中速率、调制频偏、地址码，以及串口速率、校验方式、串口类型等。

在数据传输方式上，APC220模块有两种数据传输方式，第一种是透明数据传输，透明数据传输能适应任何标准或非标准的用户协议，所接收的数据就是所发送的数据；第二种是分地址数据传输，此时所传内容的前两个字节为地址，后为数据，若接收端接收到地址匹配的数据包，即将地址、数据传给终端设备，否则将丢弃，分地址数据传输主要用于组网以及中继的需求，使用这种方式可以减轻上位机的软件开销。

7.1.2 模块引脚定义

APC220模块的外观如图7.2所示。

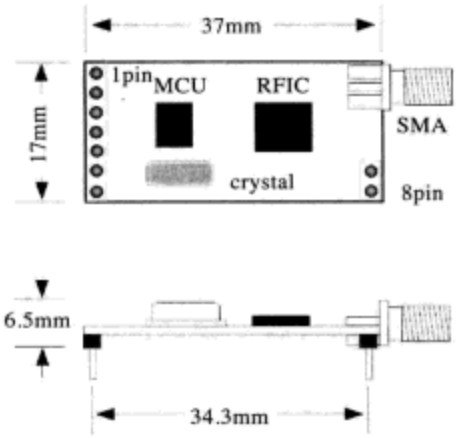


图7.2 APC220模块外观尺寸

模块引脚定义如表7.1所示。

表7.1 APC220模块引脚定义

引 脚	定 义	说 明
1	GND	地
2	VCC	3.3－5V
3	EN	电源使能端，≥1.6V或悬空使能，≤0.5V休眠
4	RXD	URAT输入口，TTL电平
5	TXD	URAT输出口，TTL电平
6	AUX	URAT信号，接收为低，发送为高
7	SET	设置参数，低有效

7.1.3 模块的使用

APC220模块可直接应用于Input Shield扩展板、XBee传感器扩展板V5和Interface shield扩展板，占用Arduino的引脚0（RX）和引脚1（TX），使用方法与Arduino中串口通信使用方法相同。使用前先使用模块的设置软件RF-ANET对模块的参数进行设置，软件界面如图7.3所示，所设置的参数如表7.2所示。

注意：APC220模块连接电脑需使用图7.1最下方的USB适配座。对于一般的客户，软件设置的选项选择默认即可（出厂时为默认值），除非有特别的用途，选项中空中速率、调制频偏、输出功率是不需要调整的。

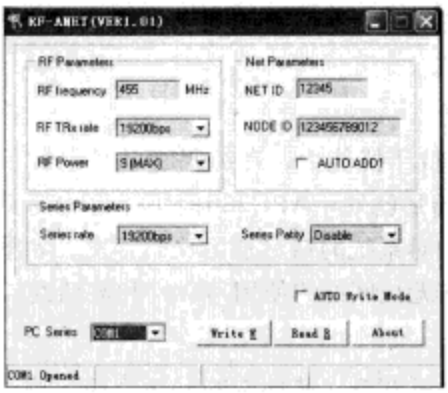


图7.3　RF-ANET设置软件

表7.2　APC220模块参数设置说明

设　置	选　项	默　认
收发频率（RF frequency）	415～455MHz（步进1KHz，精度±100Hz）	434MHz
空中速率（RF TRx Rate）	1200，2400，4800，9600，19200bps	9600bps
输出功率（RF Power）	0～9	9
串口速率（Series Rate）	1200/2400/4800/9600/19 200/38 400/57 600bps	9600bps
网络地址码（NET ID）	0～65 535（16位）	12 345
节点地址码（NODE ID）	123456789012	
串口效验（Series Patity）	Disable、Odd Patity、Even Patity	disable

7.1.4　注意事项

考虑到空中传输的复杂性，无线数据传输方式固有的一些特点，在模块的使用中需考虑以下几个问题。

1. 无线通信中数据的延迟

由于无线通信发射端是从终端设备接收到一定数量的数据后，或等待一定的时间没有新的数据才开始发射，无线通信发射端到无线通信接收端存在着几十到几百ms延迟（具体延迟是由串口速率，空中速率以及数据包的大小决定），另外从无线通信接收端到终端设备也需要一定的时间，但同样的条件下延迟时间是固定的。

2. 数据流量的控制

APC220模块虽然有512byte大容量缓冲区，但若串口速率大于等于空中速率，则存在数据流量的问题，可能会出现数据溢出而导致的数据丢失的现象。在这种情况下，终端设备要保证串口平均速率不大于60%空中速率，如串口速率为9600bps，空中速率为4800bps，终端设备每次向串口发送100byte，那么终端设备每次向串口发送的时间约104ms，

(104ms/0.6)*(9600/4800)=347ms，所以终端设备每次向串口发送100byte每次间隔不小于347ms，以上问题则不会出现。

3. 差错控制

APC220模块具有较强的抗干扰能力，在编码时已经包含了强大的纠检错能力。但在极端恶劣的条件下或接收地的场强已处于APC220模块接收的临界状态，难免出现接收不到或丢包的状况。此时客户可增加对系统的链路层协议的开发，如增加类似TCP/IP中滑动窗口及丢包重发等功能，可大大提高无线网络的使用可靠性和灵活性。

4. 天线的选择

天线是通信系统的重要组成部分，其性能的好坏直接影响通信系统的指标，用户在选择天线时必须首先注重其性能。一般有两个方面：第一选择天线类型；第二选择天线的电气性能。选择天线类型的意义是：所选天线的方向图是否符合系统设计中电波覆盖的要求；选择天线电气性能的要求是：选择天线的频率带宽、增益、额定功率等电气指标是否符合系统设计要求。因此，用户在选择天线时最好向厂家联系咨询，APC220要求的天线阻抗为50Ω。

7.2 DFduino wireless

DFduino wireless无线模块非常适合与Arduino配合使用，除了可以像APC220模块一样进行无线数据传输外，其独有的无线编程模式非常适用于Arduino不便连接或根本就无法连接USB线缆的情况下。模块上有两个拨码开关，分别用于模块的设置模式和出厂编程模式。DFduino wireless模块可设置10种速率，但其传输距离较短，室外理论传输距离为20m。模块实物如图7.4所示。

图7.4 DFduino wireless无线模块

7.2.1 DFduino wireless性能指标

- 工作电源：直流电2.7～3.6V，推荐值3.3V
- 工作电流：10mA（供电电压3.3V）
- 工作温度范围：−30～＋70℃
- 接口类型：UART
- 支持波特率：1200/2400/4800/9600/14 400/19 200/28 800/38 400/57 600/115 200bps
- 理论传输距离：室外20m

7.2.2　模块引脚定义

在模块的底面印有模块的引脚定义，如图7.5所示。

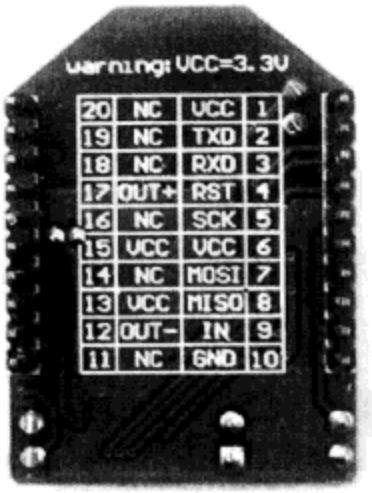


图7.5　DFduino wireless无线模块引脚定义

提示：模块在使用中实际只用到了TXD、RXD、VCC、GND及OUT-，其他引脚均用于厂家内部使用。

7.2.3　模块的使用

DFduino wireless模块可直接应用于XBee传感器扩展板V5，占用Arduino的引脚0（RX）和引脚1（TX），使用方法与APC220模块类似。

模块的设置是通过写模块内部的寄存器实现的，模块内部寄存器信息如表7.3所示。

表7.3　DFduino wireless模块内部寄存器

寄存器序号	寄存器名称
1	波特率设置寄存器
2	
3	
4	器件地址寄存器
5	
6	
7	
8	
9	载波频率寄存器
10	空中波特率寄存器
11	时限设置寄存器

1. 波特率设置寄存器

V1版本支持的波特率为：1200/2400/4800/9600/14 400/19 200/28 800/38 400/57 600/115 200bps。现以波特率设置方式来阐述所有寄存器和发送字符的关系，单片机实际工作时的波特率由3个寄存器来确定，由于所有的指令操作为字符型，为了方便记忆和操作，将波特率拆成6个字符，

如115 200，拆成“11”“52”“00”（设置时写“0”亦可，以下类同）；1200拆成“00”“12”“00”，每两个字符代表对应寄存器中的实际数值，这样做的原因是用户可以不用去记忆实际波特率与参数的对应关系，记住自己常用的波特率，拆成6个字符发送即可。

2. 器件地址寄存器

器件地址由5个字节的数据组成，可以通过地址的区分来实现多模块间下载不受干扰（这种方法我们实际测试过），如果在实际的环境中这种区分模块的方式不是很可靠，可以再对不同的模块设置不同的载波频率。

3. 载波频率寄存器

此无线模块载波频率可以在2.4～2.5GHz范围内调节，载波频率寄存器的最大值为125，单位调节频率为0.000 8GHz。

4. 空中波特率

空中波特率只能设置为3种模式：250K/1M/2M(bps)。

5. 时限设置

在Arduino系列产品无线程序下载应用中，时限设置非常讲究，目前，引导程序（boot loader）中波特率设置为19 200的，时限设置为10；波特率设置为57 600和115 200的，时限设置为20。

为方便操作，此模块仅有4条指令，对寄存器的操作，只能一一配置，不能连续对寄存器一次配置。

❑ 指令1：ATENTER\r\n

指令功能：查看模块配置信息。

指令应用实例如下所示：

```
ATENTER
Thank you for using the 2.4G wireless module form DFRobot
----------------------------www.dirobot.com-------------------------------

---------------------------Enter the setting mode-----------------------
---------------------------Show the settings----------------------------
Version:V1
Baudrate(Register:1-3):57600
ID(Register:4-8):192.168.1.1.1
Operating Frequency(0-125)(Register:9):40
Air Date Rate(1M/2M/250K)(Register:10):250Kbps
Timeout(Register:11):20
```

❑ 指令2：ATEXIT\r\n

指令功能：软件复位模块，使能之前所有设置。

指令应用实例如下所示：

```
ATEXIT
---------------------------------EXIT---------------------------------
```

❑ 指令3：ATEEPW[],[]\r\n

指令功能：对指定寄存器进行写操作。

指令应用实例如下所示（将模块的波特率设置为9600bps）：

```
ATEEPW1,0\r\n
ATEEPW2,96\r\n
ATEEPW3,0\r\n
```

❑ 指令4：ATEEPR[]\r\n

指令功能：对指定寄存器进行读操作。

指令应用实例如下所示（读取寄存器1的值）：

```
ATEEPW1\r\n
```

所有不正确的参数设置将在输入“ATENTER\r\n”指令查看模块配置信息时，在对应项中将显示“ERROR”。其中，波特率最为特殊，一旦输入的参数有误，对应项显示“ERROR”外，系统波特率恢复为默认值，即57 600。

注意：模块设置需使用针对DFduino wireless的USB to Serial适配模块。

7.3　Bluetooth V3

Bluetooth V3无线模块基于蓝牙通信协议研制，采用独特双层板设计，既美观又防止静电损坏模块，设计2个电源输入口，宽电压供电（3.5～8V）和3.3V供电，可适用于各种场合。STATE和LINK指示灯用于显示模块工作状态和连接状态。

自带高效板载天线，信号质量好发射距离更远，透明串口，可与各种蓝牙适配器、蓝牙手机配对使用，人性化的设计为二次开发提供便利。模块实物如图7.6所示。

图7.6　Bluetooth V3

7.3.1　Bluetooth V3性能指标

❑ 蓝牙芯片：CSR BC417143

❑ USB协议：USB 1.1/2.0

❑ 工作频率：在无需授权的ISM波段下为2.4～2.48GHz

❑ 调制方式：GFSK（Gaussian Frequency Shift Keying，高斯频移键控）

- 发射功率：小于4dBm
- 传输距离：在空旷无遮挡的情况下为20～30m
- 传输速率：异步传输：2.1Mbps(Max) / 160 Kbps；同步传输：1Mbps/1Mbps
- 数据传输形式：蓝牙串行端口
- 串口波特率：4800～1 382 400/无校验位/8位数据位/1位停止位，默认为9600/无校验位/8位数据位/1位停止位
- 输入电压：直流电+3.5～+8V或3.3V
- 工作温度：−20～+55°C
- 模块尺寸：40mm × 20mm × 13mm

7.3.2 模块引脚定义

表7.4 Bluetooth V3模块引脚定义

引 脚	定 义	说 明
1	GND	地
2	VCC	直流电3.5～8V输入，当有VCC输入时，3V3端口可当做3.3V电源输出
3	NC	空脚
4	RXD	URAT输入口（TTL电平）接单片机TXD
5	TXD	URAT输出口（TTL电平）接单片机RXD
6	GND	地
7	3V3	直流电3.3V，当有3.3V输入时，VCC端口不能接输入电源

7.3.3 模块的使用

Bluetooth V3蓝牙模块可直接应用于Input Shield扩展板、XBee传感器扩展板V5和Interface shield扩展板，占用Arduino的引脚0（RX）和引脚1（TX）。

Bluetooth V3蓝牙模块支持AT指令设置波特率和主从机模式。模块有一个2位拨码开关，1号开关LED Off是LINK灯的开关，可以关闭LINK省电，拨到ON为开，拨到1端为关；2号开关AT Mode是AT命令模式开关，拨到ON进入AT命令模式，拨到2端退出AT命令模式。

注意：模块设置需使用针对Bluetooth V3的USB to Serial适配模块。

将模块通过USB to Serial适配模块连接到电脑上后，打开串口调试工具，在发送栏中输入AT（不分大小写），然后点击发送，可看见模块返回OK，这表示AT指令通信正常。当AT指令设置完毕后，将2号开关AT Mode拨到2端退出AT命令模式，重新上电后设置才生效。

AT指令集如下：

- 测试指令：

指令	响应	参数
AT	OK	无

❑ 模块重启指令：

指令	响应	参数
AT+RESET	OK	无

❑ 设置和查询模块角色：

指令	响应	参数
AT+ROLE=< Param > AT+ROLE?	OK +ROLE: < Param > OK	Param：参数取值如下： 0——从角色（Slave） 1——主角色（Master） 2——回环角色（Slave-Loop） 默认值：0

模块角色说明：

Slave（从角色）——被动连接，可以和任意蓝牙适配器配对使用。

Master（主角色）——查询周围从设备，并主动发起连接，从而建立主从蓝牙设备间的透明数据传输通道。

Slave-Loop（回环角色）——被动连接，接收远程蓝牙主设备数据并将数据原样返回给远程蓝牙主设备。

❑ 设置和查询配对码：

指令	响应	参数
AT+PSWD=< Param > AT+PSWD?	OK + PSWD: < Param > OK	Param：配对码 默认值："1234"

❑ 设置和查询串口参数：

指　令	响　应	参　数
AT+ UART=< Param1 >, < Param2 >, < Param3 >	OK	Param1：波特率（bps）
AT+ UART?	+ UART :< Param1 >, < Param2 >, < Param3> OK	取值如下（十进制）： 4800 9600 19 200 38 400 57 600

（续）

指　令	响　应	参　数
AT+ UART?	+ UART :< Param1 >, < Param2 >, < Param3> OK	115 200 230 400 460 800 921 600 1 382 400 Param2：停止位 0——1 位 1——2 位 Param3：校验位 0——None 1——Odd 2——Even 默认设置：9600,0,0

2个模块成对使用时必须是一主一从，可使用AT指令将2个模块分别设置为主机和从机。主从成对使用不需要驱动程序，两个模块上电就能传输。

模块上的STATE灯频闪状态时表示正在配对，LINK灯常亮表示配对完毕，此时串口功能已经启动。

一主一从成对正常使用时灯不会灭。如主机和从机距离太远而断线，则主机和从机的灯一直闪，如果它们距离再靠近，则又会找到而自动连上。主机会记忆它配好的从机，一上电给主机就会找它记忆的从机地址。

7.4 XBee和XBee PRO

XBee和XBee PRO无线模块基于Zigbee通信协议开发，是一种低功耗的无线组网通信模块。多个模块可以形成一个多跳的自组织Zigbee网络（基于Zigbee协议——IEEE 802.15.4）是它的优势所在，在网络中的各个节点均可移动，网络的拓扑结构也会随着节点的移动而不断地动态变化。XBee和XBee PRO两个模块均工作在ISM 2.4GHz频段，且引脚互相兼容。模块实物图如图7.7所示。

图7.7 XBee和XBee PRO

7.4.1　XBee及XBee PRO性能指标

	XBee	XBee PRO
高性能指标	室内传输距离：30m	室内传输距离：100m
	户外传输距离（开阔地）：100m	户外传输距离（开阔地）：1500m
	发射功率：1mW（0dBm时）	发射功率：100mW（20dBm）
	接收灵敏度：−92dBm	接收灵敏度：−100dBm
低功率指标	发送电流：45mA（3.3V）	发送电流：215mA（3.3V）
	接收电流：50 mA（3.3V）	接收电流：55 mA（3.3V）
	掉电电流：<10mA	掉电电流：<10mA

7.4.2　模块引脚定义

XBee及XBee PRO模块顶部视图如图7.8所示：

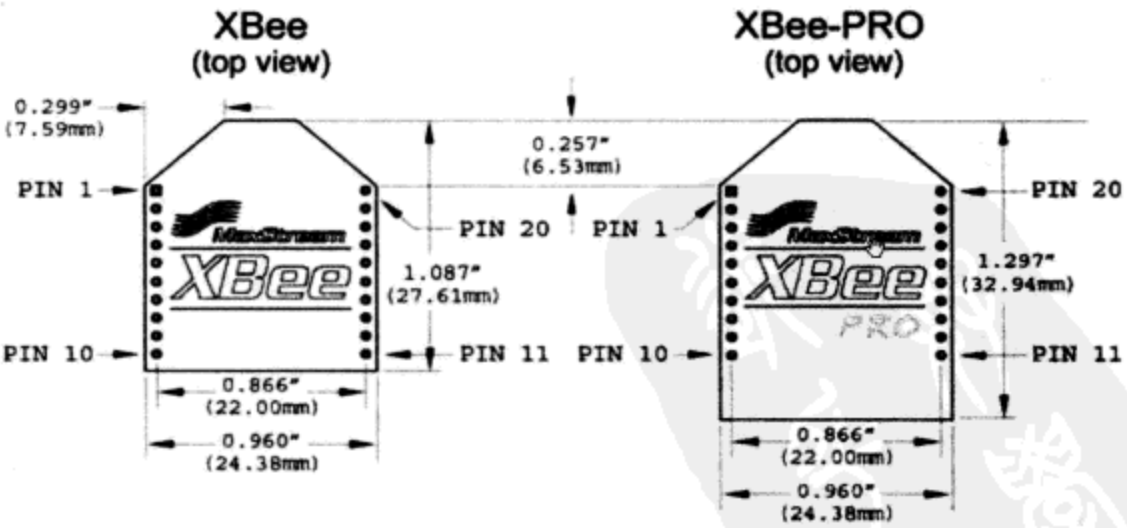


图7.8　XBee及XBee PRO模块顶部视图

引脚定义如下表所示：

表7.5　XBee及XBee PRO引脚定义

引　脚	定　义	说　明
1	VCC	电源（3.3V）
2	DOUT	UART数据输出
3	DIN/CONFIG	UART数据输入
4	DO8	数据输出8
5	RESET	模块复位（复位脉冲必须至少200ns）

（续）

引 脚	定 义	说 明
6	PWM0/RSSI	PWM输出0/RX信号强度指示器
7	NC	未用
8	NC	未用
9	DTR/SLEEP/DI8	睡眠引脚控制或数字输入8
10	GND	地
11	AD4/DIO4	模拟输入4或数字I/O 4
12	CTS/DIO7	CTS（Clear-to-Send）或数字I/O 7
13	ON/SLEEP	模块状态指示
14	VREF	A/D参考电压输入
15	Associate/AD5/DIO5	模拟输入5或数字I/O 5
16	RTS/AD6/DIO6	RTS（Request-to-Send）、模拟输入6或数字I/O 6
17	AD3/DIO3	模拟输入3或数字I/O 3
18	AD2/DIO2	模拟输入2或数字I/O 2
19	AD1/DIO1	模拟输入1或数字I/O 1
20	AD0/DIO0	模拟输入0或数字I/O 0

7.4.3 模块的使用

XBee和XBee PRO模块可直接应用于XBee传感器扩展板V5，扩展板上应用了模块的最小连接方式（VCC、GND、DOUT、DIN），占用Arduino的引脚0（RX）和引脚1（TX）。模块可以在ZigBee网络中用作协调器或终端设备。

XBee和XBee PRO模块的设置支持AT和API指令模式，也可使用模块的设置软件X-CTU对模块的参数进行设置，软件界面如图7.9所示。

多个XBee和XBee PRO模块可以形成一个多跳的自组织Zigbee网络（基于Zigbee协议：IEEE 802.15.4），在网络中的各个节点均可移动，网络的拓扑结构也会随着节点的移动而不断动态变化，XBee和XBee PRO模块的具体使用方式以及ZigBee的内容这里就不详细介绍了，有兴趣的读者可以查阅相关的文章和资料。

在Arduino上使用XBee和XBee PRO模块简单得多，在第6章中已经介绍过一个XBee库，这里通过使用其中的成员函数实现一个无线数据收发的功能来说明演示一下模块的使用。

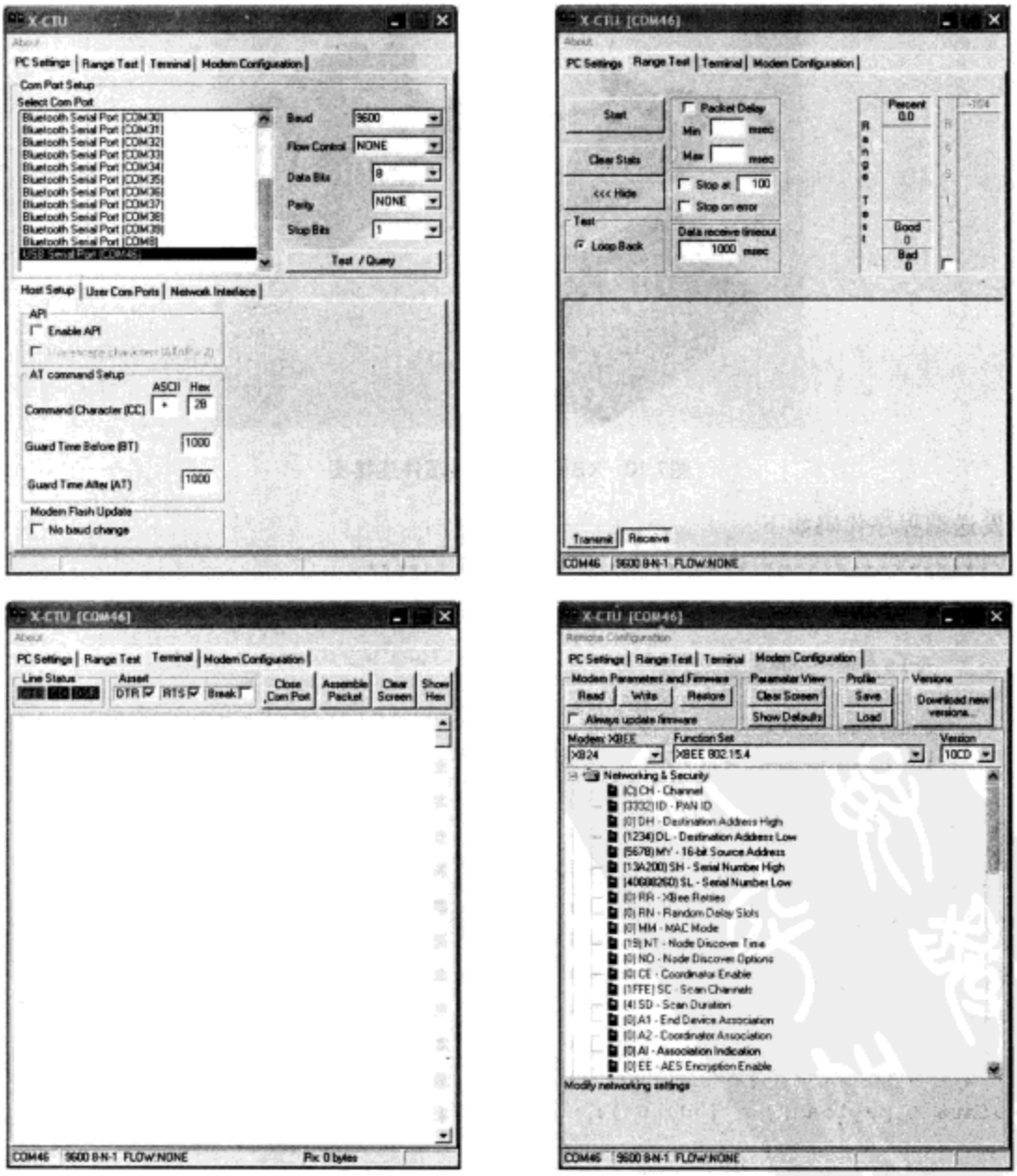


图7.9　X-CTU软件界面

7.4.4　程序设计

本实例需要两块连接好XBee模块的Arduino板，一块用来定时发送数据，另一块在收到数据后改变伺服电机的角度，伺服电机占用Arduino的引脚8。硬件连接如图7.10所示。

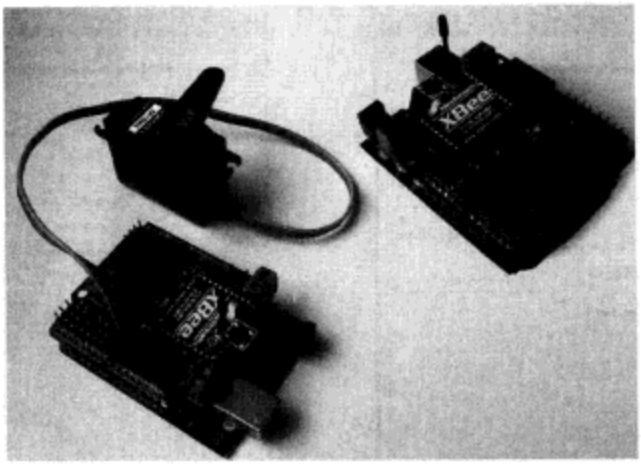


图7.10 XBee应用实例硬件连接图

发送端程序代码如下:

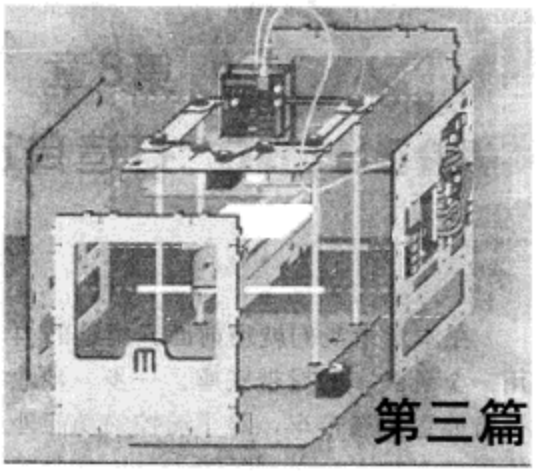
```
/*********************************************
     XBee应用实例——发送端

     每1s发送一个模拟口5的采样值

     created 2011
     by Nille
     Email: chenille@126.com

This example code is in the public domain.
*********************************************/
#include <XBee.h>

// 建立一个XBee的对象
XBee xbee = XBee();

//定义一个数组用于存储要发送的数据
uint8_t payload[] = { 0, 0 };

// 设置接收端的SH + SL地址
XBeeAddress64 addr64 = XBeeAddress64(0x0013a200, 0x403e0f30);
ZBTxRequest zbTx = ZBTxRequest(addr64, payload, sizeof(payload));

//定义变量pin5用于保存采样数据
int pin5 = 0;
```

```
/*************************************************
            初始化部分——setup函数
*************************************************/
void setup()
{
  xbee.begin(9600);
}

/*************************************************
            执行部分——loop函数
*************************************************/
void loop()
{
    pin5 = analogRead(5);          //获取模拟口5的模拟量值，并进行AD转换
    payload[0] = pin5 >> 8 & 0x03;//由于Arduino的AD是10位的，所以将结果放入两个字节
    payload[1] = pin5 & 0xff;

    xbee.send(zbTx);               //发送数据

    delay(1000);                   //延时1s
}
```

接收端程序代码如下：

```
/*************************************************
     XBee应用实例——接收端

收到数据后改变伺服电机的角度

     created 2011
     by Nille
     Email: chenille@126.com

This example code is in the public domain.
*************************************************/
#include <XBee.h>

// 建立一个XBee的对象
XBee xbee = XBee();
ZBRxResponse rx = ZBRxResponse();

uint16_t  angle=1500;          //定义初始脉宽值
/*************************************************
            初始化部分——setup函数
```

```
*************************************************/
void setup()
{
  //舵机占用引脚8，设为输出
  pinMode(8, OUTPUT);

  xbee.begin(9600);
}

/*************************************************
          执行部分——loop函数
*************************************************/
void loop()
{
    xbee.readPacket();                    //提取XBee接收到的数据包

        if (xbee.getResponse().isAvailable())       //如果收到数据包
         {
              if (xbee.getResponse().getApiId() == ZB_RX_RESPONSE)
             {
                      xbee.getResponse().getZBRxResponse(rx);
                        //提取模拟量值，并将数值转化到1000～2024
                    angle = rx.getData(0)*256 +  rx.getData(1) + 1000;
              }
        }
         //输出高电平
         digitalWrite(8, HIGH);
         //脉宽为angle
         delayMicroseconds(angle);
         //输出低电平
         digitalWrite(8, LOW);
         //低电平持续15ms
         delay(15);
}
```



第三篇

应 用 篇

第8章　打造自己的遥控履带车
第9章　仿生机器人

第8章 打造自己的遥控履带车

在本章中我们就用前面介绍过的内容，利用一个玩具车的履带底座，一步一步地打造一个智能遥控履带车，除了遥控的功能外，还要给履带车加上一点判断能力，比如会判断前方是否有障碍，是否有陡坡或台阶，同时还具有一个带云台的摄像头用来观察周围的环境。

先来看看本书选择的玩具车履带底座，实物如图8.1所示，读者可以选择自己的小车底座，或者利用一些报废的玩具车，但有一点要注意，底座的左右两个轮子必须是用两个电机驱动的，像《四驱兄弟》里的那种四驱车就不满足要求。

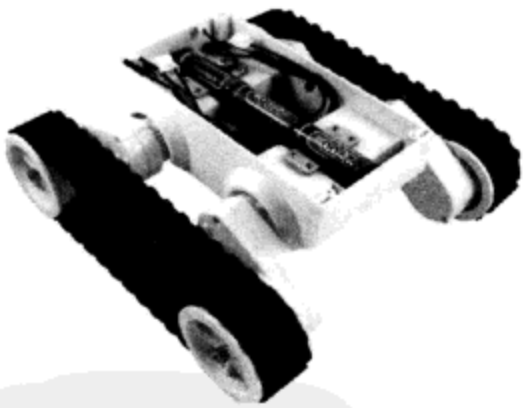


图8.1 玩具车履带底座

8.1 履带车的驱动

8.1.1 实现功能

通过这一节的调试，应该能够使用Arduino让履带车完成基本的动作，包括左前、右前、前进、后退、左转及右转，同时对了解履带车的最大速度并调整实际的速度。

8.1.2 所需器材

本节实质上的主要工作是完成直流电机的控制，参考5.1节的内容，这里所需的器材如下：

- ❑ L293 Motor Shield扩展板
- ❑ 大容量5号电池（6节）
- ❑ 电池盒
- ❑ Arduino控制板
- ❑ 履带车底座（带两个直流电机）

- ❑ 一块用来固定Arduino的安装板（可用面包板制作）
- ❑ M3的螺柱及螺丝若干
- ❑ 导线若干

8.1.3　硬件连接

首先把安装板固定在履带车底座上，再装上Arduino控制板，L293 Motor Shield扩展板插接在Arduino控制板上，最后连接直流电机以及电池盒的引线。这里假设由直流电机直接驱动的两个轮子为后轮，由履带带动的两个轮子为前轮；将前轮置于前方，定义左边的后轮为左轮，右边的后轮为右轮。左右轮的直流电机连接示意图如图8.2所示。

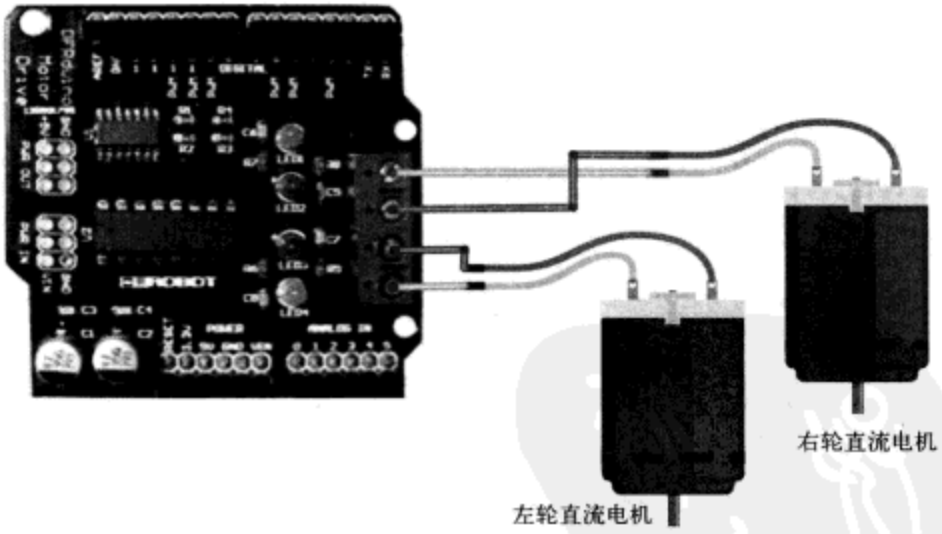


图8.2　履带车直流电机连接示意图

提示：在直流电机正转的情况下，当用于控制左轮时，是控制左轮向前转；但当用于控制右轮时，则是控制右轮向后转，所以在连接直流电机的引线时，通常左右轮直流电机的连接方式是相反的。

编写两个调试程序，分别对左轮和右轮的控制进行测试，L293 Motor Shield扩展板占用Arduino的引脚4、5、6、7。本书中达到的效果是：当引脚5输出高电平时，引脚4输出高电平，左轮直流电机正转（控制履带车向右前方前进），引脚4输出低电平，左轮直流电机反转（控制履带车向右后方运动）；当引脚7输出高电平时，引脚6输出高电平，右轮直流电机反转（控制履带车向左前方前进），引脚6输出低电平，左轮直流电机正转（控制履带车向左后方运动）。两个调试程序代码如下：

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

```
左轮直流电机调试程序

占用引脚4、5
当引脚5输出高电平时
引脚4输出高电平，左轮直流电机正转（控制履带车向右前方前进）
引脚4输出低电平，左轮直流电机反转（控制履带车向右后方运动）

created 2011
by Nille
Email: chenille@126.com

This example code is in the public domain.
************************************************/


/************************************************
          初始化部分——setup函数
************************************************/
void setup()
{
        //设置4号引脚为输出，控制直流电机正反转
        pinMode(4, OUTPUT);
        //设置5号引脚为输出，控制直流电机转速
        pinMode(5, OUTPUT);
}

/************************************************
        执行部分——loop函数
************************************************/
void loop()
{
        //引脚4输出高电平，直流电机正转
        digitalWrite(4, HIGH);
        //引脚5输出高电平
        digitalWrite(5, HIGH);
        delay(5000);

        //停止直流电机的转动
        digitalWrite(5, LOW);


        //引脚4输出低电平，直流电机反转
        digitalWrite(4, LOW);
```

```
        //引脚5输出高电平
        digitalWrite(5, HIGH);
        delay(5000);

        //停止直流电机的转动
        digitalWrite(5, LOW);
}

/*********************************************
右轮直流电机调试程序

占用引脚7、引脚6
当引脚7输出高电平时
引脚6输出高电平，右轮直流电机反转（控制履带车向左前方前进）
引脚6输出低电平，左轮直流电机正转（控制履带车向左后方运动）

created 2011
by Nille
Email: chenille@126.com

This example code is in the public domain.
*********************************************/


/*********************************************
        初始化部分——setup函数
*********************************************/
void setup()
{
        //设置7号引脚为输出，控制直流电机正反转
        pinMode(7, OUTPUT);
        //设置6号引脚为输出，控制直流电机转速
        pinMode(6, OUTPUT);
}
/*********************************************
        执行部分——loop函数
*********************************************/
void loop()
{
        //引脚7输出高电平，直流电机反转
        digitalWrite(7, HIGH);
        //引脚6输出高电平
        digitalWrite(6, HIGH);
```

```
        delay(5000);

        //停止直流电机的转动
        digitalWrite(6, LOW);


        //引脚7输出低电平，直流电机正转
        digitalWrite(7, LOW);
        //引脚6输出高电平
        digitalWrite(6, HIGH);
        delay(5000);

        //停止直流电机的转动
        digitalWrite(6, LOW);
}
```

提示：若直流电机未按照预想的方式动作，转动方向相反时，可对调一下直流电机的两根引线。

8.1.4 程序设计

在调整好履带车的硬件连接之后，可以构建几个驱动子函数，以方便后面使用。子函数包括turnLeft、turnRight、forward、back、turnLeftOrigin、turnRightOrigin以及stop，分别用于实现履带车的左转、右转、前进、后退、原地左转、原地右转和停止。

前进子函数代码如下：

```
/*************************************************
forward子函数——前进子函数
函数功能：控制履带车前进
入口参数：_speed——前进速度，范围0～255
*************************************************/
void forward(int _speed)
{
        //引脚7输出高电平，右轮前进
        digitalWrite(7, HIGH);

        //引脚4输出高电平，左轮前进
        digitalWrite(4, HIGH);

        //引脚6输出PWM，PWM值由_speed决定
        analogWrite(6,_speed);
```

```
        //引脚5输出PWM，PWM值由_speed决定
        analogWrite(5,_speed);
}
```

后退子函数代码如下：

```
/*************************************************
back子函数——后退子函数
函数功能：控制履带车后退
入口参数：_speed——后退速度，范围0～255
*************************************************/
void back(int _speed)
{
        //引脚7输出高电平，右轮后退
        digitalWrite(7, LOW);

        //引脚4输出高电平，左轮后退
        digitalWrite(4, LOW);

        //引脚6输出PWM，PWM值由_speed决定
        analogWrite(6,_speed);

        //引脚5输出PWM，PWM值由_speed决定
        analogWrite(5,_speed);
}
```

左转子函数代码如下：

```
/*************************************************
turnLeft子函数——左转子函数
函数功能：控制履带车左转
入口参数：_speed——速度，范围0～255
*************************************************/
void turnLeft (int _speed)
{
        //引脚7输出高电平，右轮前进
        digitalWrite(7, HIGH);

        //引脚6输出PWM，PWM值由_speed决定
        analogWrite(6,_speed);

        //引脚5输出PWM，PWM值为0，表示左轮静止不转
        analogWrite(5,0);
}
```

右转子函数代码如下：

```
/*************************************************
turnRight子函数——右转子函数
函数功能：控制履带车右转
入口参数：_speed——速度，范围0～255
*************************************************/
void turnRight (int _speed)
{
        //引脚4输出高电平，左轮前进
        digitalWrite(4, HIGH);

        //引脚6输出PWM，PWM值为0，表示右轮静止不转
        analogWrite(6,0);

        //引脚5输出PWM，PWM值由_speed决定
        analogWrite(5, _speed);
}
```

原地左转子函数代码如下：

```
/*************************************************
turnLeftOrigin子函数——原地左转子函数
函数功能：控制履带车原地左转
入口参数：_speed——速度，范围0～255
*************************************************/
void turnLeftOrigin (int _speed)
{
        //引脚7输出高电平，右轮前进
        digitalWrite(7, HIGH);

        //引脚4输出高电平，左轮后退
        digitalWrite(4, LOW);

         //引脚6输出PWM，PWM值由_speed决定
        analogWrite(6,_speed);
        //引脚5输出PWM，PWM值由_speed决定
        analogWrite(5, _speed);
}
```

原地右转子函数代码如下：

```
/*************************************************
turnRightOrigin子函数——原地右转子函数
函数功能：控制履带车原地右转
入口参数：_speed——速度，范围0～255
```

```
*************************************************/
void turnRightOrigin (int _speed)
{
        //引脚4输出高电平，左轮前进
        digitalWrite(4, HIGH);

        //引脚7输出低电平，右轮后退
        digitalWrite(7, LOW);

        //引脚6输出PWM，PWM值由_speed决定
        analogWrite(6, _speed);

        //引脚5输出PWM，PWM值由_speed决定
        analogWrite(5, _speed);
}
```

停止函数代码如下：

```
/*************************************************
stop子函数——停止子函数
函数功能：控制履带车停止
入口参数：无
*************************************************/
void stop()
{
        //引脚6输出低
        analogWrite(6, 0);

        //引脚5输出低
        analogWrite(5,0);
}
```

编写setup函数和loop函数对以上的子函数进行测试，看履带车能否按预想的方式运动。程序代码如下，将以上的子函数放在setup函数和loop函数之前：

```
/*************************************************
直流电机控制子函数测试程序

占用引脚4、5、6、7
履带车前进2s
履带车后退2s
左转2s
右转2s
原地左转2s
原地右转2s
```

```
停止2s

created 2011
by Nille
Email: chenille@126.com

This example code is in the public domain.
************************************************/

//添加直流电机驱动子函数

/************************************************
              初始化部分——setup函数
************************************************/
void setup()
{
          pinMode(4, OUTPUT);
          pinMode(5, OUTPUT);
          pinMode(6, OUTPUT);
          pinMode(7, OUTPUT);
}
/************************************************
              执行部分——loop函数
************************************************/
void loop()
{
          forward(250);//前进子函数
          delay(2000);

          back(250);//后退子函数
          delay(2000);

          turnLeft(250);//左转子函数
          delay(2000);
          turnRight(250);//右转子函数
          delay(2000);

          turnLeftOrigin(250);//原地左转子函数
          delay(2000);

          turnRightOrigin(250);//原地右转子函数
          delay(2000);
```

```
        stop();//停止子函数
        delay(2000);
}
```

8.1.5　MotorCar类

为了更加方便地对履带车进行控制，可以将以上的驱动子函数封装成一个类——MotorCar类。MotorCar类的定义包含变量和成员函数两部分。变量定义为私有的（private），成员函数定义为公有的（public），包括forward()前进、back()后退、turnLeft()左转、turnRight()右转、turnLeftOrigin()原地左转、turnRightOrigin()原地右转、stop()停止以及构造函数。由此得到MotorCar类的定义如下：

```
/*********************************************
     文件名：MotorCar.h
履带车控制类库

     created 2011
     by Nille
     Email：chenille@126.com

This example code is in the public domain.
*********************************************/
#ifndef MotorCar_h
#define MotorCar_h

class MotorCar
{
 private:                    //定义为私有
          int     _speedLeftPin;      //定义控制左轮速度的引脚
          int     _speedRightPin;     //定义控制右轮速度的引脚
          int     _dirLeftPin;        //定义控制左轮方向的引脚
          int     _dirRightPin;       //定义控制右轮方向的引脚

     public:                          //定义为公有
     MotorCar (int _slpin, int _dlpin, int _srpin, int _drpin);  //构造函数
     void forward(int _speed);                          //前进函数
          void back(int _speed);                        //后退函数
          void turnLeft(int _speed);                    //左转函数
          void turnRight(int _speed);                   //右转函数
          void turnLeftOrigin(int _speed);              //原地左转函数
          void turnRightOrigin(int _speed);             //原地右转函数
          void stop();                                  //停止函数
```

```
};

#endif
```

各成员函数的定义如下。

MotorCar类的构造函数：

```
/*************************************************
MotorCar类构造函数
函数功能：定义用于控制直流电机的引脚
入口参数：                    _slpin，表示控制左轮速度的引脚
                              _dlpin，表示控制左轮方向的引脚
                              _srpin，表示控制右轮速度的引脚
                              _drpin，表示控制右轮方向的引脚
*************************************************/
MotorCar::MotorCar (int _slpin , int _dlpin , int _srpin , int _drpin)
{
     _speedLeftPin = _slpin;
     _speedRightPin = _srpin;
     _dirLeftPin = _dlpin;
     _dirRightPin = _drpin;
     pinMode(_speedLeftPin, OUTPUT);
     pinMode(_speedRightPin, OUTPUT);
     pinMode(_dirLeftPin , OUTPUT);
     pinMode(_dirRightPin, OUTPUT);
}
```

前进子函数代码如下：

```
/*************************************************
forward子函数——前进子函数
函数功能：控制履带车前进
入口参数：_speed——前进速度，范围0～255
*************************************************/
void MotorCar::forward(int _speed)
{
          //引脚_dirRightPin输出高电平，右轮前进
          digitalWrite(_dirRightPin, HIGH);

          //引脚_dirLeftPin输出高电平，左轮前进
          digitalWrite(_dirLeftPin, HIGH);

          //引脚_speedRightPin输出PWM，PWM值由_speed决定
          analogWrite(_speedRightPin,_speed);
```

```
        //引脚_speedLeftPin输出PWM，PWM值由_speed决定
        analogWrite(_speedLeftPin,_speed);
}
```

后退子函数代码如下：

```
/*************************************************
back子函数——后退子函数
函数功能：控制履带车后退
入口参数：_speed——后退速度，范围0～255
*************************************************/
void MotorCar::back(int _speed)
{
        //引脚_dirRightPin输出高电平，右轮后退
        digitalWrite(_dirRightPin, LOW);

        //引脚_dirLeftPin输出高电平，左轮后退
        digitalWrite(_dirLeftPin, LOW);

        //引脚_speedRightPin输出PWM，PWM值由_speed决定
        analogWrite(_speedRightPin,_speed);

        //引脚_speedLeftPin输出PWM，PWM值由_speed决定
        analogWrite(_speedLeftPin,_speed);
}
```

左转子函数代码如下：

```
/*************************************************
turnLeft子函数——左转子函数
函数功能：控制履带车左转
入口参数：_speed——速度，范围0～255
*************************************************/
void MotorCar::turnLeft (int _speed)
{
        //引脚_dirRightPin输出高电平,右轮前进
        digitalWrite(_dirRightPin, HIGH);
        //引脚_speedRightPin输出PWM，PWM值由_speed决定
        analogWrite(_speedRightPin,_speed);

        //引脚_speedLeftPin输出PWM，PWM值为0，表示左轮静止不转
        analogWrite(_speedLeftPin,0);
}
```

右转子函数代码如下：

```
/*************************************************
turnRight子函数——右转子函数
函数功能：控制履带车右转
入口参数：_speed——速度，范围0～255
*************************************************/
void MotorCar::turnRight (int _speed)
{
          //引脚_dirLeftPin输出高电平，左轮前进
          digitalWrite(_dirLeftPin, HIGH);

          //引脚_speedRightPin输出PWM，PWM值为0，表示右轮静止不转
          analogWrite(_speedRightPin,0);

          //引脚_speedLeftPin输出PWM，PWM值由_speed决定
          analogWrite(_speedLeftPin, _speed);
}
```

原地左转子函数代码如下：

```
/*************************************************
turnLeftOrigin子函数——原地左转子函数
函数功能：控制履带车原地左转
入口参数：_speed——速度，范围0～255
*************************************************/
void MotorCar::turnLeftOrigin (int _speed)
{
          //引脚_dirRightPin输出高电平，右轮前进
          digitalWrite(_dirRightPin, HIGH);

          //引脚_dirLeftPin输出高电平，左轮后退
          digitalWrite(_dirLeftPin, LOW);

          //引脚_speedRightPin输出PWM，PWM值由_speed决定
          analogWrite(_speedRightPin,_speed);

          //引脚_speedLeftPin输出PWM，PWM值由_speed决定
          analogWrite(_speedLeftPin, _speed);
}
```

原地右转子函数代码如下：

```
/*************************************************
turnRightOrigin子函数——原地右转子函数
函数功能：控制履带车原地右转
入口参数：_speed——速度，范围0～255
```

```
*********************************************/
void MotorCar::turnRightOrigin (int _speed)
{
          //引脚_dirLeftPin输出高电平，左轮前进
          digitalWrite(_dirLeftPin, HIGH);

          //引脚_dirRightPin输出低电平，右轮后退
          digitalWrite(_dirRightPin, LOW);

          //引脚_speedRightPin输出PWM，PWM值由_speed决定
          analogWrite(_speedRightPin, _speed);

          //引脚_speedLeftPin输出PWM，PWM值由_speed决定
          analogWrite(_speedLeftPin, _speed);
}
```

停止函数代码如下：

```
/*********************************************
stop子函数——停止子函数
函数功能：控制履带车停止
入口参数：无
*********************************************/
void MotorCar::stop()
{
          //引脚_speedRightPin输出低
          analogWrite(_speedRightPin, 0);

          //引脚_speedLeftPin输出低
          analogWrite(_speedLeftPin,0);
}
```

编写类的代码后将MotorCar库的文件夹放在Arduino开发环境目录下的libraries文件夹中。

8.1.6　类的应用

生成一个MotorCar类的对象Motor，分别对成员函数左转、右转、前进、后退、原地左转、原地右转和停止进行测试。

```
/*********************************************
直流电机控制子函数测试程序

占用引脚4、5、6、7
履带车前进2s
履带车后退2s
```

```
左转2s
右转2s
原地左转2s
原地右转2s
停止2s

created 2011
by Nille
Email: chenille@126.com

This example code is in the public domain.
************************************************/

#include "MotorCar.h"            //包含头文件MotorCar.h

//定义MotorCar类的对象Motor
MotorCar Motor(5,4,6,7);
/************************************************
          初始化部分——setup函数
************************************************/
void setup()
{

}
/************************************************
          执行部分——loop函数
************************************************/
void loop()
{
        Motor.forward(250);//前进子函数
        delay(2000);

        Motor.back(250);//后退子函数
        delay(2000);

        Motor.turnLeft(250);//左转子函数
        delay(2000);

        Motor.turnRight(250);//右转子函数
        delay(2000);

        Motor.turnLeftOrigin(250);//原地左转子函数
        delay(2000);
```

```
        Motor.turnRightOrigin(250);//原地右转子函数
        delay(2000);

        Motor.stop();//停止子函数
        delay(2000);
}
```

8.2　添加感知器件

8.2.1　实现功能

履带车一直盲目运动是不行的，在本节里，通过添加红外接近开关使履带车能够自主运行，实现避障、寻线、边沿检测等功能。

8.2.2　所需器材

- ❑ 红外接近开关×2
- ❑ 红外接近开关安装支架
- ❑ XBee传感器扩展板V5

8.2.3　器材介绍

红外接近开关是一种集发射与接收于一体的光电开关传感器。检测距离可以根据要求进行调节。传感器输出的信号是开关信号，无障碍物时输出高电平，有障碍物时输出低电平，并且探头后面指示灯亮，探测范围3～80cm。传感器实物图如图8.3所示，其性能指标如下：

图8.3　红外接近开关

- ❑ 电源：5V
- ❑ 电流：<100mA
- ❑ 探测距离：3～80cm
- ❑ 探头直径：18 mm
- ❑ 探头长度：45 mm
- ❑ 电缆长度：45 cm
- ❑ 接口定义：3线制，可直接插在XBee传感器扩展板V5上。其中红色为电源，绿色为地，黄色为信号输出

8.2.4 硬件连接

在8.1节的硬件基础上再插接上XBee传感器扩展板V5，同时将红外接近开关通过安装支架固定在安装板的左前方和右前方，并将红外接近开关插头连接在XBee传感器扩展板V5上，其中左前方红外接近开关占用引脚11，右前方的红外接近开关占用引脚12。硬件连接如图8.4所示。

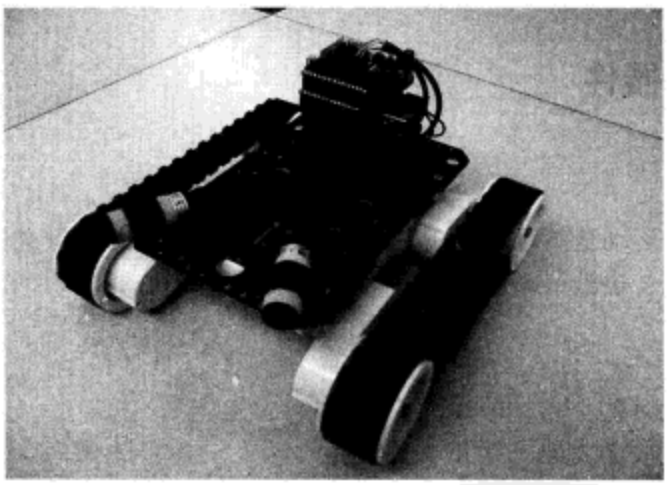


图8.4 红外接近开关连接效果图

8.2.5 程序设计

在连接好红外接近开关后就可以编写一个程序让履带车自主自由地运动了，不用担心它会碰到“南墙”也不回头。程序设计左边的红外接近开关检测到有障碍物时，履带车原地右转；右边的红外接近开关检测到有障碍物时，履带车原地左转。代码如下：

```
/*********************************************
加装红外接近开关后履带车的避障程序

左边的红外接近开关检测到有障碍物时（占用引脚11），履带车原地右转
右边的红外接近开关检测到有障碍物时（占用引脚12），履带车原地左转

created 2011
by Nille
Email: chenille@126.com

This example code is in the public domain.
*********************************************/
```

```
#include "MotorCar.h"              //包含头文件MotorCar.h

//定义MotorCar类的对象Motor
MotorCar Motor(5,4,6,7);

/*********************************************
              初始化部分——setup函数
*********************************************/
void setup()
{
          pinMode(11, INPUT);
          pinMode(12, INPUT);
}
/*********************************************
              执行部分——loop函数
*********************************************/
void loop()
{
     Motor.forward(250);                              //前进子函数
          if(LOW == digitalRead(11))                  //左前方检测到有障碍物
          {
                Motor.stop();                         //停止子函数
                Motor.turnRightOrigin(250);           //原地右转子函数
          delay(1000);
          }

          if(LOW == digitalRead(12))                  //右前方检测到有障碍物
          {
                Motor.stop();                         //停止子函数
                Motor.turnLeftOrigin(250);            //原地左转子函数
          delay(1000);
          }
}
```

履带车避障程序也可以设计成左边的红外接近开关检测到有障碍物时，履带车后退一段距离，再右转；右边的红外接近开关检测到有障碍物时，履带车后退一段距离，再左转。程序代码如下：

```
/*********************************************
加装红外接近开关后履带车的避障程序

左边的红外接近开关检测到有障碍物时，履带车后退一段距离，再右转；
右边的红外接近开关检测到有障碍物时，履带车后退一段距离，再左转
```

```
created 2011
by Nille
Email: chenille@126.com

This example code is in the public domain.
*********************************************/

#include "MotorCar.h"            //包含头文件MotorCar.h

//定义MotorCar类的对象Motor
MotorCar Motor(5,4,6,7);

/***********************************************
              初始化部分——setup函数
***********************************************/
void setup()
{
              pinMode(11, INPUT);
              pinMode(12, INPUT);
}
/***********************************************
                执行部分——loop函数
***********************************************/
void loop()
{
      Motor.forward(250);                        //前进子函数
              if(LOW == digitalRead(11))         //左前方检测到有障碍物
              {
                      Motor.back(250);           //后退子函数
                      delay(1000);
                      Motor.stop();              //停止子函数
                      Motor.turnRight(250);      //原地右转子函数
              delay(1000);
              }

              if(LOW == digitalRead(12))         //右前方检测到有障碍物
              {
                      Motor.back(250);           //后退子函数
                      delay(1000);
                      Motor.stop();              //停止子函数
                      Motor.turnLeft(250);       //原地左转子函数
                      delay(1000);
```

```
            }
    }
```

8.3　添加无线模块

8.3.1　实现功能

为了实现遥控履带车的目的，本节通过添加无线模块使履带车能够接收到电脑发送的控制报文，使履带车能够在遥控状态下运动。这里选用之前介绍的APC220模块进行无线通信。

8.3.2　所需器材

- ❑ APC220模块 × 2
- ❑ USB TO UART/TTL接口转换适配座

8.3.3　硬件连接

使用USB TO UART/TTL接口转换适配座将模块连到电脑上，如图8.5所示，分别将两个APC220模块的参数设置为：收发频率434MHz、空中速率19 200bps、输出功率9级、串口速率9600bps、串口效验Disable。设置界面如图8.6所示。

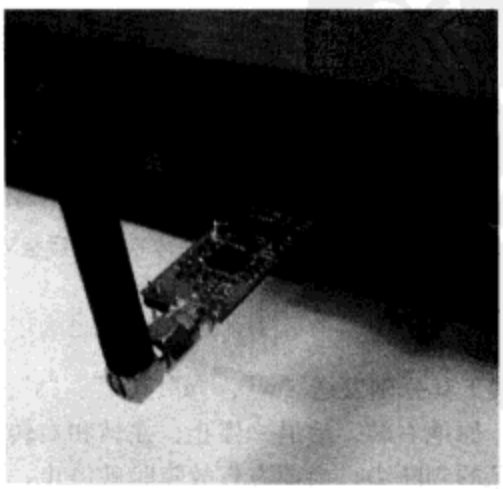


图8.5　APC220模块连接电脑

将一个APC220模块插接在XBee传感器扩展板V5上，如图8.7所示，另一个依然连在电脑上。电脑和履带车之间的无线通信通道就接通了。

图8.6 APC220模块设置

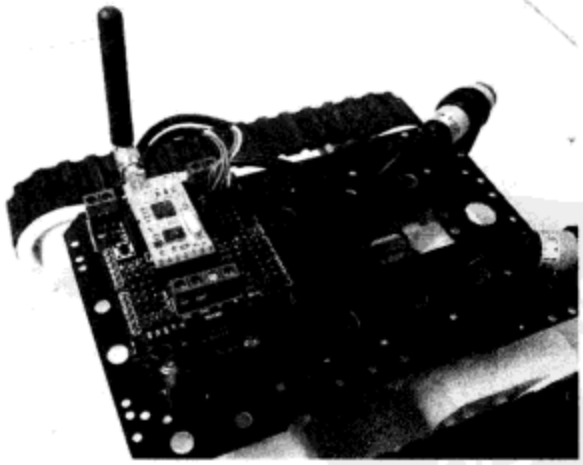


图8.7 APC220模块插接在XBee传感器扩展板V5上

8.3.4 程序设计

在电脑端使用串口调试工具分别发送“w”、“a”、“d”、“s”、“t”、“q”、“e”7个字母，分别表示前进、原地左转、原地右转、后退、停止、左转和右转。履带车要对这7个字母做出响应，同时还要有它自己的判断力，当前方有故障时就停止，无论是左前方还是右前方。程序代码如下：

```
/*********************************************
加装红外接近开关和APC220无线模块后履带车的程序

在电脑端使用串口调试工具分别发送“w”、“a”、“d”、“s”、“t”、“q”、“e”7个字母
```

```
分别表示前进、原地左转、原地右转、后退、停止、左转和右转
当前方有故障时履带车就停止

created 2011
by Nille
Email: chenille@126.com

This example code is in the public domain.
*********************************************/

#include "MotorCar.h"            //包含头文件MotorCar.h

//定义MotorCar类的对象Motor
MotorCar Motor(5,4,6,7);

int      comtemp;                //定义一个变量来存储串口收到的数据

/***********************************************
          初始化部分——setup函数
***********************************************/
void setup()
{

            //红外接近开关传感器使用的引脚
            pinMode(11, INPUT);
            pinMode(12, INPUT);

            Serial.begin(9600);              // 波特率9600 bps
}
/***********************************************
          执行部分——loop函数
***********************************************/
void loop()
{
      if ( Serial.available())
      {
                        comtemp= Serial.read();
            switch(comtemp)
            {
                        case 'w':
                                    Motor.forward(250);     //前进子函数
                                    break;
```

```
                case 's':
                    Motor.back(250);                    //后退子函数
                    break;
                case 'a':
                    Motor.turnLeftOrigin(250);  //原地左转子函数
                    break;
                case 'd':
                    Motor.turnRightOrigin(250); //原地右转子函数
                    break;
                case 'q':
                    Motor.turnLeft(250);                //左转子函数
                    break;
                case 'e':
                    Motor.turnRight(250);               //右转子函数
                    break;
                case 't':
                    Motor.stop();                       //停止子函数
                    break;
                default:
                    Motor.stop();                       //停止子函数
                    break;
            }
        }

        if((LOW == digitalRead(11))&&('w' == comtemp))//左前方检测到有障碍物
        {
            Motor.stop();                               //停止子函数
        }

        if((LOW == digitalRead(12)) &&('w' == comtemp))//右前方检测到有障碍物
        {
            Motor.stop();                               //停止子函数
        }
}
```

程序下载完成后，就可以通过电脑端使用串口调试工具发送控制命令控制履带车动作。

8.4 制作遥控器

8.4.1 实现功能

8.3节实现了使用电脑无线遥控履带车，本节就来用Input Shield扩展板制作一个遥控器

控制履带车，这样履带车的控制就不需要依托电脑了。

8.4.2　所需器材

- ❑ Arduino控制板（另外一块，不是履带车上那个了）
- ❑ Input Shield扩展板
- ❑ APC220模块（可使用8.3节中电脑端连接的APC220模块）
- ❑ 方形电池盒（装4节5号电池）
- ❑ 大容量5号电池（4节）

8.4.3　硬件连接

将Input Shield扩展板插接在Arduino控制板上，再将APC220模块插接在Input Shield扩展板上，最后给Arduino通电。硬件连接完成后效果如图8.8所示。

图8.8　Arduino遥控器

8.4.4　程序设计

使用Arduino采集摇杆的上下方向和左右方向的模拟量数值，根据数值的范围确定发送给履带车的控制命令。程序代码如下：

```
/*********************************************
遥控器实例程序

采集摇杆的上下方向和左右方向的模拟量数值，根据数值的范围确定发送给履带车的控制命令

摇杆上下方向和左右方向分别占用Arduino的模拟口0和模拟口1


created 2011
by Nille
Email: chenille@126.com

This example code is in the public domain.
*********************************************/

int      valueHori;          //左右方向模拟量数值
int      valueVerti;         //上下方向模拟量数值
int      comTemp;
int      oldComTemp;

/*********************************************
          初始化部分——setup函数
```

```
*********************************************/
void setup()
{
      //设置串口波特率为9600bps
              Serial.begin(9600);
}

/*********************************************
              执行部分——loop函数
*********************************************/
void loop()
{
              valueHori = analogRead(1);          //保存左右方向的模拟量数值
              valueVerti = analogRead(0);         //保存上下方向的模拟量数值

              if(valueVerti <230)                 //摇杆向上
              {
                        if(valueHori<230)         //摇杆向右
                        {
                                  comTemp ='e';   //右转
                        }
                        else if(valueHori>800)    //摇杆向左
                        {
                                  comTemp ='q';   //左转
                        }
                        else
                        {
                                  comTemp ='w';   //前进
                        }
              }
              else if(valueVerti >800)            //摇杆向下
              {
                        comTemp ='s';             //后退
              }
              else                                //摇杆在中间位置
              {
                        if(valueHori<230)         //摇杆向右
                        {
                                  comTemp ='d';   //原地右转子函数
                        }
                        else if(valueHori>800)    //摇杆向左
                        {
                                  comTemp ='a';   //原地左转子函数
```

```
                }
                else
                {
                        comTemp ='t';       //停止子函数
                }
        }

        if(comTemp != oldComTemp)                    //当命令不同时才发送命令
        {
                Serial.print(comTemp , BYTE); //发送命令
                oldComTemp= comTemp;
        }
}
```

程序下载完成后，将履带车和遥控器都通电，控制摇杆就能够实现履带车的前进后退等动作。

8.5　履带车遥控调速

8.5.1　实现功能

在8.1节中编写的直流电机驱动部分子函数中用到了一个参数_speed，用以控制直流电机转动的速度，但是在之前的应用中没有发挥这个参数的功能。本节中我们编写履带车和遥控器的程序，根据采集到的摇杆的模拟量值来改变履带车的运行速度。

8.5.2　程序设计

程序设计包括履带车和遥控器两部分，遥控器要将模拟量的数据发送给履带车，而不像之前的只是发送前进“w”、后退“s”等命令信息。履带车在收到模拟量值后要控制直流电机的转速。发送的数据格式如下：

```
'C'+ 命令字（q、w、e、a、s、d、t） + [_speed参数值]
```

遥控器的程序代码如下：

```
/*********************************************
遥控器实例程序——发送速度参数

采集摇杆的上下方向和左右方向的模拟量数值，根据采集到的摇杆的模拟量值来改变履带车的运行速度
数据格式：'C'+ 命令字（q、w、e、a、s、d、t） + _speed参数值
```

```
摇杆上下方向和左右方向分别占用Arduino的模拟口0和模拟口1

created 2011
by Nille
Email: chenille@126.com

This example code is in the public domain.
*********************************************/

int   valueHori;       //左右方向模拟量数值
int   valueVerti;      //上下方向模拟量数值
int   comTemp;
int   oldComTemp;

/*************************************************
              初始化部分——setup函数
*************************************************/
void setup()
{
      //设置串口波特率为9600bps
              Serial.begin(9600);
}

/*************************************************
              执行部分——loop函数
*************************************************/
void loop()
{
            valueHori = analogRead(1);            //保存左右方向的模拟量数值
            valueVerti = analogRead(0);           //保存上下方向的模拟量数值

            if(valueVerti <230)                   //摇杆向上
            {
                    if(valueHori<230)             //摇杆向右
                    {
                            comTemp ='e';         //右转
                    }
                    else if(valueHori>800)        //摇杆向左
                    {
                            comTemp ='q';         //左转
                    }
                    else
                    {
```

```
                    comTemp ='w';          //前进
            }
    }
    else if(valueVerti >800)               //摇杆向下
    {
            comTemp ='s';                  //后退
    }
    else                                   //摇杆在中间位置
    {
            if(valueHori<230)              //摇杆向右
            {
                    comTemp ='d';          //原地右转子函数
            }
            else if(valueHori>800)         //摇杆向左
            {
                    comTemp ='a';          //原地左转子函数
            }
            else
            {
                    comTemp ='t';          //停止子函数
            }
    }


    //当comTemp和oldComTemp的值都为t时不发送命令
    if(!((comTemp == 't')&&(oldComTemp==' t' )))
    {
            Serial.print('C');             //发送命令起始标记C
            Serial.print(comTemp, BYTE);//发送命令
            if(comTemp=='w')
            {
                    Serial.print((255-(valueVerti/2)),BYTE);//发送参数
                    //摇杆向上时采样模拟量值在0－512，越向上值越小
                    //通过公式变化为速度参数值
            }
            else if( comTemp=='s')
            {
                    Serial.print(((valueVerti-512)/2),BYTE);//发送参数
                    //摇杆向下时采样模拟量值在512～1024，越向下值越大
                    //通过公式变化为速度参数值
            }
            else if(( comTemp=='q')|| ( comTemp=='a'))
            {
```

```
                    Serial.print(((valueHori -512)/2),BYTE);//发送参数
                    //摇杆向左时采样模拟量值在512－1024，越向左值越大
                    //通过公式变化为速度参数值
                }
                else if(( comTemp=='e')|| ( comTemp=='d'))
                {
                    Serial.print((255-( valueHori /2)), BYTE);//发送参数
                    //摇杆向右时采样模拟量值在0～512，越向右值越小
                    //通过公式变化为速度参数值
                }
                else
                {Serial.print(0, BYTE);}
                oldComTemp= comTemp;
            }
            delay(200);//延时200ms之后再采样
}
```

履带车的程序代码如下：

```
/*********************************************
加装红外接近开关和APC220无线模块后履带车的程序——可调速

采集摇杆的上下方向和左右方向的模拟量数值，根据采集到的摇杆的模拟量值来改变履带车的运行速度
数据格式：'C'+ 命令字（q、w、e、a、s、d、t） +   _speed参数值

当前方有故障时履带车就停止

created 2011
by Nille
Email: chenille@126.com

This example code is in the public domain.
*********************************************/

#include "MotorCar.h"              //包含头文件MotorCar.h

//定义MotorCar类的对象Motor
MotorCar Motor(5,4,6,7);

int     comtemp;                   //定义一个变量用来存储串口收到的数据
int     speedtemp;

/*********************************************
              初始化部分——setup函数
```

```
*************************************************/
void setup()
{
        //红外接近开关传感器使用的引脚
        pinMode(11, INPUT);
        pinMode(12, INPUT);

        Serial.begin(9600);              // 波特率9600 bps
}
/*************************************************
              执行部分——loop函数
*************************************************/
void loop()
{
        if ( Serial.available())
      {
            if('C'== Serial.read())
            {
                while(!Serial.available());      //提取数据中的命令字
                comtemp= Serial.read();

                while(!Serial.available());      //提取数据中的_speed参数
                speedtemp= Serial.read();
               switch(comtemp)
                {
                        case 'w':
                              Motor.forward(speedtemp);  //前进子函数
                              break;
                    case 's':
                        Motor.back(speedtemp);       //后退子函数
                        break;
                    case 'a':
                        //原地左转子函数
                        Motor.turnLeftOrigin(speedtemp);
                        break;
                    case 'd':
                        //原地右转子函数
                        Motor.turnRightOrigin(speedtemp);
                        break;
                    case 'q':
                        Motor.turnLeft(speedtemp);       //左转子函数
                        break;
                    case 'e':
```

```
                    Motor.turnRight(speedtemp);//右转子函数
                    break;
                case 't':
                    Motor.stop();           //停止子函数
                    break;
                default:
                        Motor.stop();           //停止子函数
                        break;
            }
        }
    }

    if((LOW == digitalRead(11))&&('w' == comtemp))//左前方检测到有障碍物
    {
        Motor.stop();           //停止子函数
    }

    if((LOW == digitalRead(12)) &&('w' == comtemp))//右前方检测到有障碍物
    {
        Motor.stop();           //停止子函数
    }
}
```

8.6 添加无线摄像头

8.6.1 实现功能

本节会在履带车前端添加一个无线摄像头，这样在看不见履带车的情况下，我们也能够通过摄像头的信息完成履带车的遥控。为保证摄像头具有一定的灵活性，同时添加了一个舵机可使摄像头在前方180° 的范围内左右转动。

8.6.2 所需器材

- ❑ 无线摄像头wiCam
- ❑ 无线摄像头安装框架
- ❑ 伺服电机（舵机）

8.6.3 器材介绍

我们选用的是一款WIFI摄像头——wiCam，wiCam是一款集WIFI视频流、无线控制为一体的超小型无线视频模块，如图8.9所示。模块通过WIFI网络与外界进行数据传输，通过

手机端、平板电脑端以及PC端均可显示wiCam模块发送的视频流。模块同时提供一路串口，用户可通过网络socket访问串口。模块提供一套Android手机应用程序，可在手机端实现视频显示以及串行数据的双向传输。这里我们仅使用模块的无线视频功能。

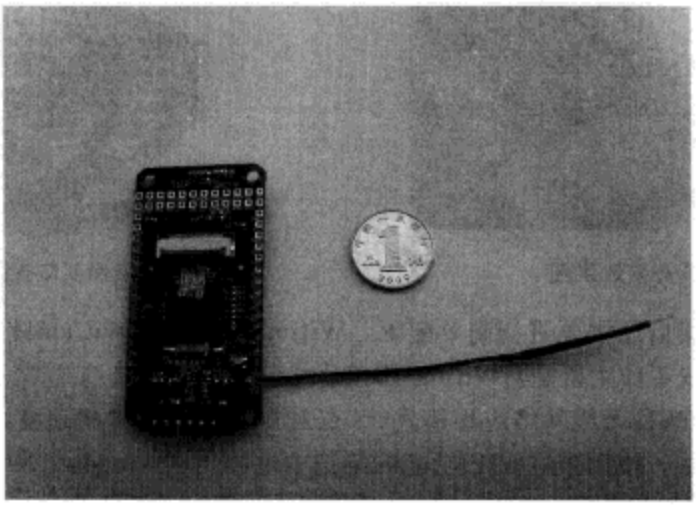


图8.9　wiCam无线摄像头模块

wiCam无线摄像头模块技术规格如下：

- 视频参数：H264 QCIF（176×144）、QVGA（320×240）、VGA（640×480）可选
- 无线参数：802.11bg INFRA/ADHOC
- 尺寸参数：3cm×6cm
- 供电电压：5V
- 工作电流：静态260mA（1.3W），拍摄传输 340mA（1.7W）
- 采集延时：拍摄图像、传输至播放器<0.3s（手机端处理可能随机型有不同延时，1G+512M机型总体时延在0.3s）。
- 传输距离：视距约50m
- 串口规格：2400～115200 UART（UDP模式）

8.6.4　硬件连接

首先如图8.10所示将舵机固定在履带车前端。舵机的控制线直接插接在XBee传感器扩展板V5上，占用Arduino的数字引脚8。然后将无线摄像头安装在框架中固定于舵机上，连接wiCam模块仅需要将电源的两条线连接在XBee传感器扩展板V5的任意对电源上。安装时注意在舵机处于中间位置时，摄像头是朝前的。安装完摄像头后的效果图如图8.11所示。

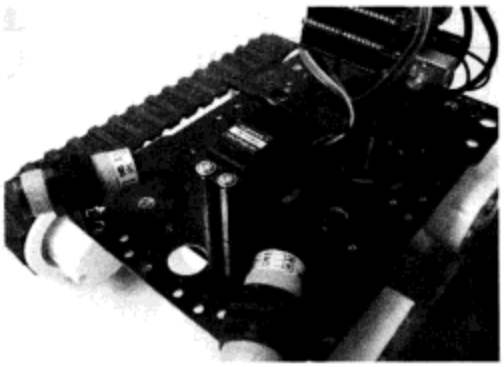


图8.10 舵机安装效果图

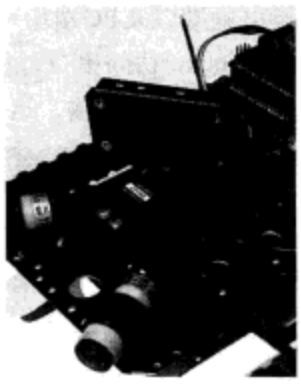


图8.11 摄像头安装后的效果图

wiCam模块连接后还需要进行简单配置。WifimodII模块是wiCam模块底层实现WIFI功能的部分，所以配置工作主要是对WifimodII模块的配置。

WifimodII支持两种无线网络应用模式，“点对点连接”和“基础模式连接”，模块初始的模式是点对点连接。使用带有WIFI功能的笔记本电脑，在Windows XP系统下，可以使用windows自带的无线扫描功能，找到wifimodII节点，如图8.12所示。

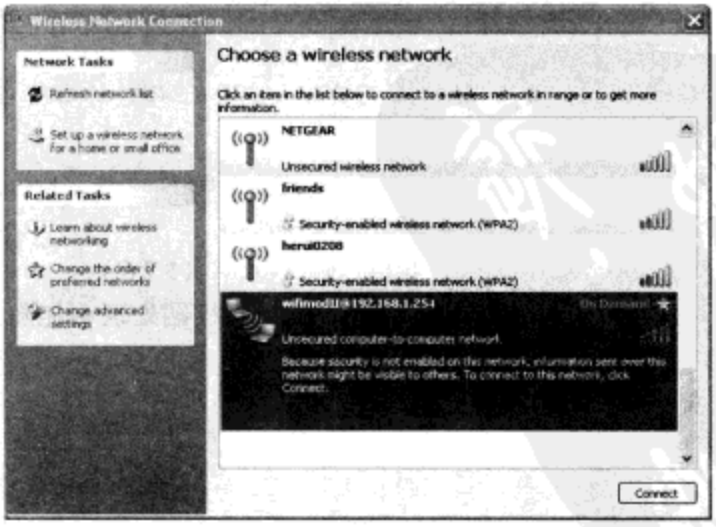


图8.12 寻找wifimodII节点

选中图中的节点，点击“Connect”按钮便可以连接上模块了，计算机会提示您已连接。由节点的名称可以看出模块缺省的IP地址为“192.168.1.254”。此时，需要改动笔记本的IP地址，使其能够与wifimodII处在同一个网段中，建议您设置为192.168.1.105（该IP为wifimodII 出厂设置的默认目标IP），配置窗口如图8.13所示。

图8.13　配置IP地址

配置IP完毕后，打开浏览器，在地址栏输入："192.168.1.254"，回车，输入缺省用户名admin 与密码admin即可登录wifimodII的配置界面了（见图8.14）。

图8.14　wifimodII的配置界面

点击界面中的参数设置。将wiCam模块接入当前的WIFI网络需要对以下几个参数进行设置。

- 将无线网络应用模式改为"基础模式"，无线网络名称及加密设置改为当前的WIFI网络名称及密码（作者使用的无线网络名称为"dfrobot"，加密方式为"WEP"），如图8.15所示。

❑ 将IP获取方式改为DHCP，如图8.16所示。

图8.15 无线网络设置

图8.16 获取IP方式

设置完成后，选择“设置保存”，模块重新上电就会直接连接到当前的WIFI无线网络中。

然后选择一款安装Android操作系统的手机，安装模块提供的手机应用程序“wifimodII管理列表”，软件图标如图8.17所示。

软件运行后，选择菜单中的“扫描模块”，如图8.18所示，会找到当前WIFI网络中的无线摄像头模块（确切地说是WifimodII模块），如图8.19所示。

图8.17 wifimodII管理列表软件

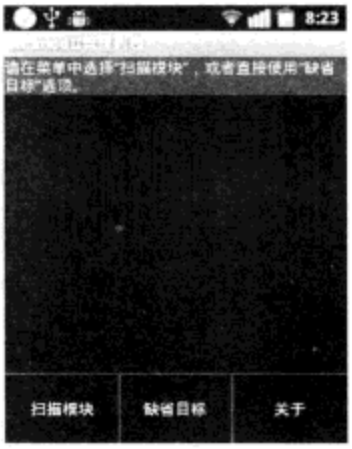


图8.18　选择菜单中的“扫描模块”

图8.19　软件寻找当前WIFI网络中的无线摄像头模块

选择正确的模块后，会出现如图8.20所示的界面，提示用户进一步操作，包括IO控制DEMO、收发测试、wiCam控制台等。这里直接选择Wicam控制台。

进入wiCam控制台界面后，就可以看到无线摄像头的图像了，同时软件界面中还会显示模块在WIFI网络中分配的IP地址。

图8.20　操作选择界面

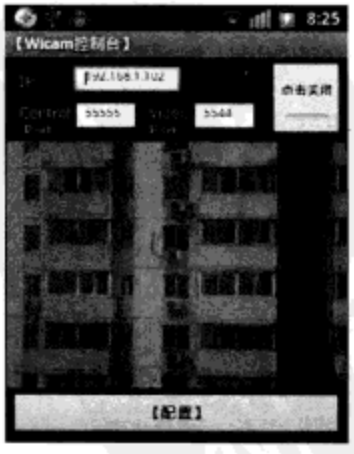


图8.21　显示无线摄像头图像界面

无线摄像头模块调试完成后，接下来就可以进行履带车和遥控器的Arduino程序设计了。

8.6.5　程序设计

基于8.5节的程序进行设计程序，可以使用遥控器来遥控摄像头的转动，在履带车的程序上添加一段处理控制摄像头舵机的代码，同样是接收遥控器的命令数据，数据格式如下为：

‘C’ + 命令字（m） + [舵机角度参数]；

当按下遥控器的蓝色按钮的情况下，左右转动摇杆，则发送命令字为‘m’的命令数据。履带车的程序代码如下：

```
/*********************************************
加装无线摄像头后履带车的程序——可调速

采集摇杆的上下方向和左右方向的模拟量数值，根据采集到的摇杆的模拟量值来改变履带车的运行速度
数据格式：'C'+ 命令字（q、w、e、a、s、d、t） +   _speed参数值

当前方有故障时履带车就停止

控制摄像头舵机的数据格式为：'C'+ 命令字（m） +  [舵机角度参数]

created 2011
by Nille
Email: chenille@126.com

This example code is in the public domain.
*********************************************/

#include "MotorCar.h"                    //包含头文件MotorCar.h
#include <Servo.h>                       //包含头文件Servo.h

//定义MotorCar类的对象Motor
MotorCar Motor(5,4,6,7);

//定义Servo类的对象myservo
Servo myservo;

int   comtemp;                  //定义一个变量用来存储串口收到的数据
int   speedtemp;

/*********************************************
              初始化部分——setup函数
*********************************************/
void setup()
{
            //红外接近开关传感器使用的引脚
            pinMode(11, INPUT);
            pinMode(12, INPUT);

            Serial.begin(9600);                  // 波特率9600 bps

            myservo.attach(8);                   //定义摄像头舵机使用引脚8
```

```
        myservo.write(90);              //设定舵机初始角度为90°
}
/*********************************************
          执行部分——loop函数
*********************************************/
void loop()
{
      if ( Serial.available())
      {
                  if('C'== Serial.read())
                  {
                     while(!Serial.available()); //提取数据中的命令字
                     comtemp= Serial.read();

                     while(!Serial.available()); //提取数据中的_speed参数
                     speedtemp= Serial.read();
                  switch(comtemp)
                  {
                        case 'w':
                              Motor.forward(speedtemp);//前进子函数
                              break;
                              case 's':
                              Motor.back(speedtemp);  //后退子函数
                              break;
                         case 'a':
                              //原地左转子函数
                              Motor.turnLeftOrigin(speedtemp);
                              break;
                         case 'd' :
                              //原地右转子函数
                              Motor.turnRightOrigin(speedtemp);
                              break;
                         case 'q':
                              Motor.turnLeft(speedtemp); //左转子函数
                              break;
                         case 'e':
                              Motor.turnRight(speedtemp);//右转子函数
                              break;
                         case 't':
                              Motor.stop();              //停止子函数
                              break;
                         case 'm':
                              //收到的是0～255的模拟量数值
```

```
                                    //转换为0～180° 的角度值
                                    speedtemp
                                          = map(speedtemp,0,255,0,179);
                                            myservo.write(speedtemp);
                                    break;
                              default:
                                          Motor.stop();          //停止子函数
                                          break;
                        }
                  }
            }

            if((LOW == digitalRead(11))&&('w' == comtemp)) //左前方检测到有障碍物
            {
                  Motor.stop();              //停止子函数
            }

            if((LOW == digitalRead(12)) &&('w' == comtemp))//右前方检测到有障碍物
            {
                  Motor.stop();              //停止子函数
            }
}
```

遥控器的程序代码如下：

```
/*********************************************
遥控器实例程序——发送速度参数及摄像头舵机角度值

采集摇杆的上下方向和左右方向的模拟量数值，根据采集到的摇杆的模拟量值来改变履带车的运行速度
数据格式：'C'+ 命令字（q、w、e、a、s、d、t） +    _speed参数值

摇杆上下方向和左右方向分别占用Arduino的模拟口0和模拟口1

控制摄像头舵机的数据格式为：'C'+ 命令字（m） +  [舵机角度参数]

蓝色按钮占用Arduino的数字口4

created 2011
by Nille
Email: chenille@126.com

This example code is in the public domain.
*********************************************/
```

```
int       valueHori;              //左右方向模拟量数值
int       valueVerti;             //上下方向模拟量数值
int       comTemp;
int       oldComTemp;

/*********************************************
            初始化部分——setup函数
*********************************************/
void setup()
{
      //设置串口波特率为9600bps
            Serial.begin(9600);

            //设置数字口4为输入
            pinMode(4, INPUT);
}

/*********************************************
                执行部分——loop函数
*********************************************/
void loop()
{
          valueHori = analogRead(1);       //保存左右方向的模拟量数值
          valueVerti = analogRead(0);      //保存上下方向的模拟量数值

          if(HIGH == digitalRead(4))
          {
                  if(valueVerti <230)                    //摇杆向上
                  {
                          if(valueHori<230)              //摇杆向右
                          {
                                  comTemp ='e';          //右转
                          }
                          else if(valueHori>800)         //摇杆向左
                          {
                                  comTemp ='q';          //左转
                          }
                          else
                          {
                                  comTemp ='w';          //前进
                          }
                  }
                  else if(valueVerti >800)               //摇杆向下
```

```
{
        comTemp ='s';                    //后退
}
else                                     //摇杆在中间位置
{
        if(valueHori<230)                //摇杆向右
        {
                comTemp ='d';            //原地右转子函数
        }
        else if(valueHori>800)           //摇杆向左
        {
                comTemp ='a';            //原地左转子函数
        }
        else
        {
                comTemp ='t';            //停止子函数
        }
}

//当comTemp和oldComTemp的值都为t时不发送命令
if(!((comTemp == 't')&&(oldComTemp=='t')))
{
        Serial.print('C');               //发送命令起始标记C
        Serial.print(comTemp , BYTE);    //发送命令
        if(comTemp=='w')
        {
           Serial.print((255-(valueVerti/2)),BYTE);
           //发送参数
           //摇杆向上时采样模拟量值在0～512，越向上值越小
           //通过公式变化为速度参数值
        }
        else if( comTemp=='s')
        {
           Serial.print(((valueVerti-512)/2),BYTE);
           //发送参数
           //摇杆向下时采样模拟量值在512～1024，越向下值越大
           //通过公式变化为速度参数值
        }
        else if(( comTemp=='q')|| ( comTemp=='a'))
        {
           Serial.print(((valueHori -512)/2),BYTE);
           //发送参数
```

```
                    //摇杆向左时采样模拟量值在512～1024，越向左值越大
                    //通过公式变化为速度参数值
                }
                else if(( comTemp=='e')|| ( comTemp=='d'))
                {
                    Serial.print((255-( valueHori /2)), BYTE);//发送参数
                    //摇杆向右时采样模拟量值在0～512，越向右值越小
                    //通过公式变化为速度参数值
                }
                else
                { Serial.print(0, BYTE);}
                oldComTemp= comTemp;
            }
        }
        else
        {
            delay(50);
            if(LOW == digitalRead(4))
            {
                Serial.print('C');              //发送命令起始标记C
                Serial.print('t');              //发送停止命令
                Serial.print(0,BYTE );

                Serial.print('C');              //发送命令起始标记C
                Serial.print('m');              //发送m命令字

                //发送参数，valueHori值的范围为0～1023
                Serial.print(valueHori/4,BYTE);

            }
        }
        delay(200);//延时200ms之后再采样
}
```

提示：也可不使用APC220模块，直接使用wiCam模块提供的串口实现手机对履带车的控制。

8.7　环境信息获取器件

8.7.1　实现功能

本节再给履带车添加两个环境信息获取模块，分别是模拟气体传感器模块和温湿度传感

器模块，使履带车能够采集所处环境的信息，并通过无线方式传送出来。

8.7.2 所需器材

- ❑ 模拟气体传感器
- ❑ SHT1x温湿度传感器

8.7.3 器材介绍

1. 模拟气体传感器

图8.22为本节添加的模拟气体传感器，传感器基于气敏元件研制，可以很灵敏地检测到空气中的烟雾、甲烷、煤气、二氧化碳等气体。其技术规格如下：

- ❑ 电压：+5V
- ❑ 电流：100mA
- ❑ 引脚定义：1—输出 2—地 3—电源
- ❑ 尺寸：42mm × 20mm
- ❑ 重量：8g

传感器输出电压模拟量，气体及烟雾浓度越大，输出的电压值越高，可使用传感器配制的连接线直接连接在XBee传感器扩展板V5上。

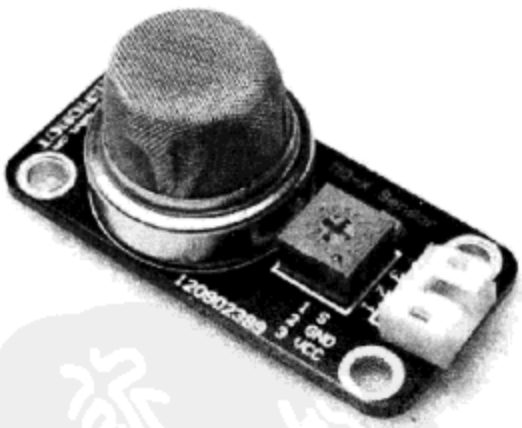


图8.22 模拟气体传感器

2. SHT1x温湿度传感器

图8.23所示的SHT1x温湿度传感器是瑞士Sensirion公司推出的单片数字温湿度集成传感器。采用CMOS过程微加工专利技术（CMOSens technology），确保产品具有极高的可靠性和出色的长期稳定性。该传感器由1个电容式聚合体测湿元件和1个能隙式测温元件组成，并与1个14位A/D转换器以及1个2-wire数字接口在单芯片中无缝结合，使得该产品具有功耗低、反应快、抗干扰能力强等优点。在对环境温度与湿度测量要求高的情况下使用，该产品具有极高的可靠性和出色的稳定性。其技术规

图8.23 SHT1x温湿度传感器

格如下：

- ❑ 全部校准，数字输出
- ❑ 接口简单（2-wire），响应速度快
- ❑ 超低功耗，自动休眠
- ❑ 出色的长期稳定性
- ❑ 超小体积（表面贴装）
- ❑ 湿度范围0～100%RH，温度范围−40～128.8℃
- ❑ 测湿精度±4.5%RH，测温精度±0.5℃（25℃）
- ❑ 模块尺寸：32mm×17mm

传感器采用2-wire接口，数字输出，可使用传感器配制的连接线直接连接在XBee传感器扩展板V5上，需使用两个数字口。

8.7.4　硬件连接

模拟气体传感器和温湿度传感器均可使用配套的连接线直接连接到XBee传感器扩展板V5上，安装完成后的效果图如图8.24所示，其中模拟气体传感器占用Arduino的模拟口0，温湿度传感器占用Arduino的数字口9（接SHT1x温湿度传感器的SCK）和数字口10（接SHT1x温湿度传感器的DATA）。

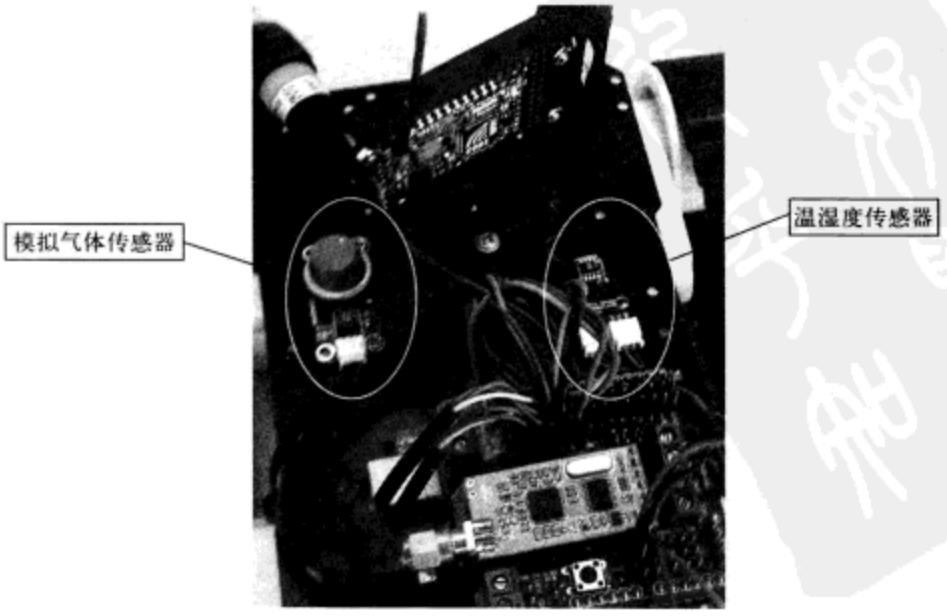


图8.24　模拟气体传感器和温湿度传感器安装效果图

遥控履带车完成后整体效果如图8.25所示。

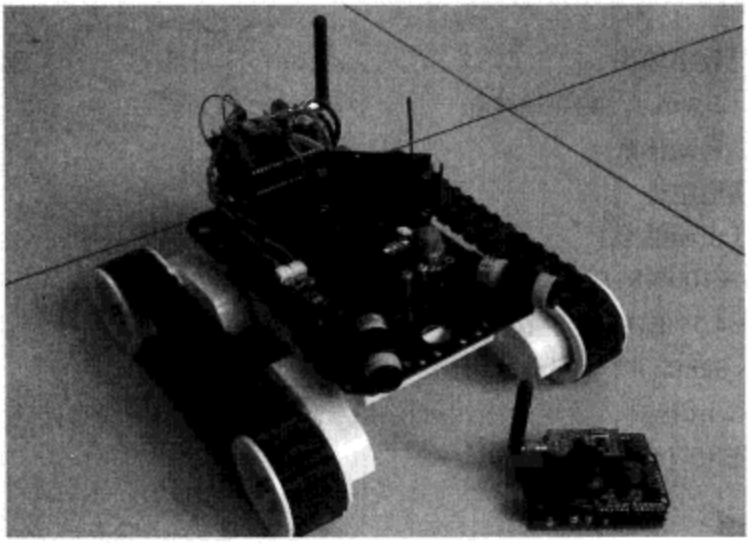


图8.25 遥控履带车整体效果

8.7.5 程序设计

硬件连接后分别编写两个测试程序对模拟气体传感器和温湿度传感器进行测试，确定硬件连接的正确性。测试程序控制Arduino控制板每2s获取一次温湿度及模拟气体传感器值，并通过串口发送给电脑，可在Arduino开发环境下Serial Monitor（串口监视窗）中看到测试结果。

模拟气体传感器测试程序如下：

```
/*********************************************
模拟气体传感器测试程序

模拟气体传感器占用Arduino的模拟口0

created 2011
by Nille
Email: chenille@126.com

This example code is in the public domain.
*********************************************/

/*************************************************
        初始化部分——setup函数
*************************************************/
void setup()
{
            //使用串口传输模拟气体传感器数值
```

```
        Serial.begin(9600);                    // 波特率9600 bps
}
/*********************************************
        执行部分——loop函数
*********************************************/
void loop()
{
        //传输模拟气体传感器采集值
        //模拟气体传感器占用Arduino的模拟口0
        Serial.println(analogRead(0));
        //2s采样一次
        delay(2000);
}
```

温湿度传感器需使用SHT1x库文件，可以在Arduino网站上下载，测试程序如下：

```
/*********************************************
温湿度传感器测试程序

数字口9（接SHT1x温湿度传感器的SCK）
数字口10（接SHT1x温湿度传感器的DATA）

created 2011
by Nille
Email: chenille@126.com

This example code is in the public domain.
*********************************************/

#include <SHT1x.h>

#define dataPin  10
#define clockPin 9

//定义SHT1x类的对象sht1x
SHT1x sht1x(dataPin, clockPin);

/*********************************************
       初始化部分——setup函数
*********************************************/
void setup()
{
        //使用串口传输温湿度传感器数值
        Serial.begin(9600);                    // 波特率9600 bps
```

```
}

/*********************************************
        执行部分——loop函数
*********************************************/
void loop()
{
        float temp_c;                   //定义温度值变量
        float humidity;                 //定义湿度值变量

        // 读取温湿度值
        temp_c = sht1x.readTemperatureC();
        humidity = sht1x.readHumidity();

        //通过串口输出温度值
        Serial.print("Temperature: ");
        Serial.print(temp_c, DEC);

        //通过串口输出湿度值
        Serial.print("      Humidity: ");
        Serial.print(humidity);
        Serial.println("%");

        //2s采样一次
        delay(2000);
}
```

完成硬件的测试后，就要把环境信息传输的代码加到8.6节的程序中，设定程序当收到Ch的命令数据后，会采集环境的温度、湿度以及烟雾浓度，并通过串口发送。程序代码如下：

```
/*********************************************
加装无线摄像头、环境信息获取器件后履带车的程序——可调速

created 2011
by Nille
Email: chenille@126.com

This example code is in the public domain.
*********************************************/

#include "MotorCar.h"           //包含头文件MotorCar.h
#include <Servo.h>              //包含头文件Servo.h
#include <SHT1x.h>              //包含头文件SHT1x.h

//定义MotorCar类的对象Motor
```

```
MotorCar Motor(5,4,6,7);

//定义Servo类的对象myservo
Servo myservo;

//定义SHT1x类的对象sht1x
SHT1x sht1x(10, 9);

int       comtemp;                    //定义一个变量用来存储串口收到的数据
int       speedtemp;

/*********************************************
          初始化部分——setup函数
*********************************************/
void setup()
{
            //红外接近开关传感器使用的引脚
            pinMode(11, INPUT);
            pinMode(12, INPUT);

            Serial.begin(9600);     // 波特率9600 bps

            myservo.attach(8);      //定义摄像头舵机使用引脚8
      myservo.write(90);            //设定舵机初始角度为90°
}
/*********************************************
            执行部分——loop函数
*********************************************/
void loop()
{
          if ( Serial.available())
        {
                if('C'== Serial.read())
                {
                      while(!Serial.available()); //提取数据中的命令字
                      comtemp= Serial.read();

                      while(!Serial.available()); //提取数据中的_speed参数
                      speedtemp= Serial.read();
                switch(comtemp)
                {
                       case 'w':
                                  Motor.forward(speedtemp);//前进子函数
                                  break;
```

```
        case 's':
              Motor.back(speedtemp);//后退子函数
              break;
        case 'a':
              //原地左转子函数
              Motor.turnLeftOrigin(speedtemp);
             break;
        case 'd':
              //原地右转子函数
              Motor.turnRightOrigin(speedtemp);
              break;
        case 'q':
              Motor.turnLeft(speedtemp);//左转子函数
              break;
        case 'e':
         Motor.turnRight(speedtemp);//右转子函数
         break;
        case 't':
         Motor.stop();              //停止子函数
         break;
        case 'm':
         //收到的是0－255的模拟量数值
         //转换为0～180° 的角度值
         speedtemp
           = map(speedtemp, 0, 255, 0, 179);
                  myservo.write(speedtemp);
         break;
        case 'h':
         //通过串口输出温度值
         Serial.print("Temperature: ");
      Serial.print(sht1x.readTemperatureC());

         //通过串口输出湿度值
      Serial.print("      Humidity: ");
      Serial.print(sht1x.readHumidity());
      Serial.print("%");

         //通过串口输出烟雾浓度
         Serial.print("      smog: ");
         Serial.print(analogRead(0));
         Serial.println("(0-1023)");
         break;
    default:
              Motor.stop();        //停止子函数
```

```
                                        break;
                    }
                }
            }

            if((LOW == digitalRead(11))&&('w == comtemp))//左前方检测到有障碍物
            {
                Motor.stop();                   //停止子函数
            }

            if((LOW == digitalRead(12)) &&('w' == comtemp))//右前方检测到有障碍物
            {
                Motor.stop();                   //停止子函数
            }
}
```

遥控器的程序代码也要进行相应调整，当按下红色按钮时，则发送“Ch”命令字。

```
/*********************************************
遥控器实例程序——发送速度参数及摄像头舵机角度值

采集摇杆的上下方向和左右方向的模拟量数值，根据采集到的摇杆的模拟量值来改变履带车的运行速度
数据格式：'C'+ 命令字（q、w、e、a、s、d、t） +   _speed参数值

摇杆上下方向和左右方向分别占用Arduino的模拟口0和模拟口1

控制摄像头舵机的数据格式为：'C'+ 命令字（m） +  [舵机角度参数]

获取环境信息的报文为：'Ch'

蓝色按钮占用Arduino的数字口4
红色按钮占用Arduino的数字口3

created 2011
by Nille
Email: chenille@126.com

This example code is in the public domain.
*********************************************/

int      valueHori;              //左右方向模拟量数值
int      valueVerti;             //上下方向模拟量数值
int      comTemp;
int      oldComTemp;
```

```
/*************************************************
              初始化部分——setup函数
*************************************************/
void setup()
{
      //设置串口波特率为9600bps
      Serial.begin(9600);

      //设置数字口4为输入
      pinMode(4, INPUT);

      //设置数字口3为输入
      pinMode(3, INPUT);
}

/*************************************************
                  执行部分——loop函数
*************************************************/
void loop()
{
            valueHori = analogRead(1);        //保存左右方向的模拟量数值
            valueVerti = analogRead(0);       //保存上下方向的模拟量数值

            if(LOW == digitalRead(3))
            {
                  delay(50);
                  if(LOW == digitalRead(3))
                  {
                        Serial.print('C');              //发送命令起始标记C
                        Serial.print('t');              //发送停止命令
                        Serial.print(0,BYTE );

                        Serial.println ("Ch");          //发送命令Ch
                  }
            }

            if(HIGH == digitalRead(4))
            {
                  if(valueVerti <230)               //摇杆向上
                  {
                        if(valueHori<230)           //摇杆向右
                        {
                              comTemp ='e';         //右转
```

```
        }
        else if(valueHori>800)    //摇杆向左
        {
              comTemp ='q';       //左转
        }
        else
        {
              comTemp ='w';       //前进
        }
  }
  else if(valueVerti >800)        //摇杆向下
  {
        comTemp ='s';             //后退
  }
  else                            //摇杆在中间位置
  {
        if(valueHori<230)         //摇杆向右
        {
              comTemp ='d';       //原地右转子函数
        }
        else if(valueHori>800)    //摇杆向左
        {
              comTemp ='a';       //原地左转子函数
        }
        else
        {
              comTemp ='t';       //停止子函数
        }
  }


  //当comTemp和oldComTemp的值都为t时不发送命令
  if(!((comTemp == 't')&&(oldComTemp=='t')))
  {
        Serial.print('C');                  //发送命令起始标记C
        Serial.print(comTemp , BYTE); //发送命令
        if(comTemp=='w')
        {
              Serial.print((255-(valueVerti/2)), BYTE);
              //发送参数
              //摇杆向上时采样模拟量值在0～512，越向上值越小
              //通过公式变化为速度参数值
        }
```

```
                else if( comTemp=='s')
                {
                      Serial.print(((valueVerti-512)/2), BYTE);
                      //发送参数
                      //摇杆向下时采样模拟量值在512～1024，越向下值越大
                      //通过公式变化为速度参数值
                }
                else if(( comTemp=='q')|| ( comTemp=='a'))
                {
                      Serial.print(((valueHori -512)/2), BYTE);
                      //发送参数
                      //摇杆向左时采样模拟量值在512～1024，越向左值越大
                      //通过公式变化为速度参数值
                }
                else if(( comTemp=='e')|| ( comTemp=='d'))
                {
                      Serial.print((255-(valueHori /2)),BYTE);//发送参数
                      //摇杆向右时采样模拟量值在0～512，越向右值越小
                      //通过公式变化为速度参数值
                }
                else
                { Serial.print(0, BYTE);}
                oldComTemp= comTemp;
          }
    }
    else
    {
          delay(50);
          if(LOW == digitalRead(4))
          {
                Serial.print('C');                    //发送命令起始标记C
                Serial.print('t' );                   //发送停止命令
                Serial.print(0,BYTE );

                Serial.print('C');                    //发送命令起始标记C
                Serial.print('m' );                   //发送m命令字
                //发送参数，valueHori值的范围为0～1023
                Serial.print(valueHori/4,BYTE );

          }
    }
    delay(200);//延时200ms之后再采样
}
```

第9章 仿生机器人

机器人主要用于在人类不宜、不便或不能进入的地域进行探测，比如在2011年日本福岛由于地震引起核电站泄漏的区域。根据行动方式机器人可以分为两种：一种是由轮子驱动的轮行机器人，例如第8章中的遥控履带车就属于这个范畴，第8章中的履带车可以监测环境的温度、湿度以及烟雾浓度，轮行机器人的不足之处在于对复杂的自然地形适应能力差；另一种是基于仿生学的仿生机器人，仿生机器人可以在复杂的自然地形中完成探测任务。

仿生机器人是指模仿生物关节动作的机器人，在小型机器人领域多用舵机实现关节的动作，目前常见的是6足爬虫机器人、8足蜘蛛机器人、人形机器人等。本书通过遥控机械臂和双足机器人两个仿生机器人局部功能的制作让读者对仿生机器人有一个初步的认识，了解Arduino对关节的控制方式。

9.1 遥控机械臂

9.1.1 实例功能

本节制作一个6自由度机械臂，然后用一个Wii Nunchuck游戏手柄控制机械臂的动作。自由度是指机构具有确定运动时所必须给定的独立运动参数的数目，简单地说，一个自由度可以理解为一个活动关节，用一个舵机来控制。6自由度就使用了6个舵机，对应臂、肘、腕(2个自由度)、张合5个关节和1个旋转底座，每个关节可在一定范围内运动，底座可以实现左右90°旋转，当让机械臂具有确定运动时，我们必须同时给出6个舵机独立运动的参数，从而实现机械臂在空间中的精确作业，所以称为6自由度机械臂。

9.1.2 器材列表

- ❑ Arduino控制板
- ❑ XBee传感器扩展板V5
- ❑ 舵机×6
- ❑ 大容量5号电池（6节）

- ❑ 电池盒
- ❑ 舵机支架
- ❑ 旋转底座
- ❑ 小型机械手夹持器
- ❑ 铝合金长U型支架
- ❑ 螺钉螺母若干

9.1.3 搭建硬件环境

在组装6自由度机械臂之前，为保证安装的正确性，首先要将舵机的角度调整到90°，然后安装码盘，使码盘的4个圆孔成正十字，如图9.1所示。

将XBee传感器扩展板V5插接在Arduino控制板上，连上电源，编写一个控制舵机的程序，通过引脚8将舵机的角度调整到90°。硬件连接如图9.2所示。

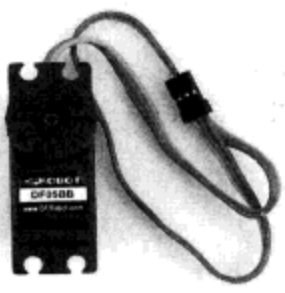


图9.1 调整舵机

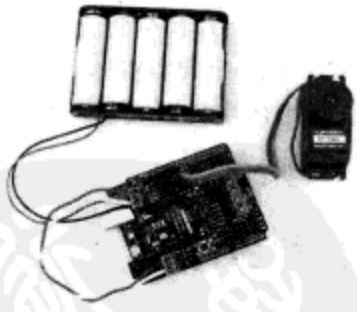


图9.2 搭建舵机调整环境

舵机调整程序代码如下：

```
/*************************************************
舵机调整程序

通过引脚8将舵机的角度调整到90°

created 2011
by Nille
Email: chenille@126.com

This example code is in the public domain.
*************************************************/

#include <Servo.h>                    //包含头文件Servo.h

Servo myservo;                        //定义一个Servo类的对象myservo
```

```
/************************************************
            初始化部分——setup函数
************************************************/
void setup()
{
        myservo.attach(8);                    //使用引脚8控制舵机
        myservo.write(90);                    //将舵机的角度调整到90°
}

/************************************************
                执行部分——loop函数
************************************************/
void loop()
{
        //loop函数内无需执行程序
}
```

舵机调整完后，就可以进行结构件的安装了，按以下步骤进行。

1）在旋转底座上安装一个舵机，能够实现左右90°的旋转，安装效果如图9.3所示。

2）在底座的舵机上安装一个舵机支架，用以安装臂关节的舵机，安装效果如图9.4所示。

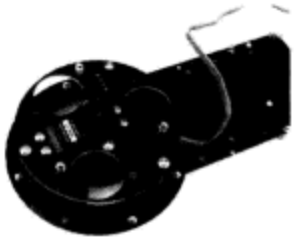


图9.3 旋转底座的安装

图9.4 底座的舵机上安装一个舵机支架

3）安装上臂关节的舵机，并在舵机上连接两个铝合金长U型支架，安装效果如图9.5所示。

4）在臂关节铝合金长U型支架的另一端安装肘关节，安装方式类似于臂关节的安装，效果如图9.6所示。

图9.5 安装臂关节的舵机

图9.6 肘关节的安装

5）将两个舵机安装支架背向连接好，并安装上舵机，构成腕关节安装在肘关节的另一端，效果如图9.7所示。

6）最后安装小型机械手夹持器及控制其张合的舵机，效果如图9.8所示。至此，机械臂的结构就安装完了。

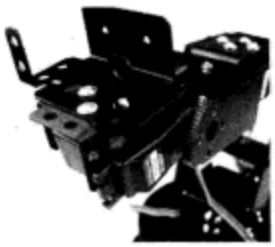


图9.7 腕关节的安装

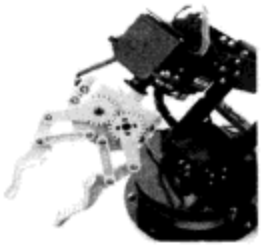


图9.8 小型机械手夹持器的安装

9.1.4 安装控制部分

相对于结构件的安装，控制部分的安装要简单得多。将插接了XBee传感器扩展板V5的Arduino控制板及电池盒依次安装在旋转底座上，再将电源正负极线连接在扩展板Servo_PWR接线柱上，电源连接线从Servo_PWR接线柱连接到VIN接线柱上。安装效果如图9.9所示。

将机械臂舵机从下到上编为1~6号，连接在控制部分，分别占用Arduino的数字口8~13。所有的舵机线都插好后，准备一些黑塑螺旋管，将舵机线缠好。安装效果如图9.10所示。

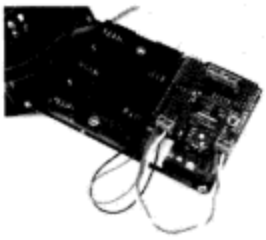


图9.9 控制部分的安装

图9.10 舵机控制线的连接

9.1.5 Wii游戏手柄

Wii Nunchuck游戏手柄集成了1个3轴加速度传感器，可以提供I^2C接口，实物如图9.11所示。Wii Nunchuck游戏手柄和Arduino配合会让我们的机械臂作品更炫。Wii Nunchuck游戏手柄的接口不能直接插在Arduino板上，所以需要选择一个WiiChuck适配器，方便Arduino添加Wii Nunchuck游戏手柄。

将WiiChuck适配器上的各个I^2C引脚和扩展板上的I^2C引脚一一对应连接好，注意两者的

线序是不同的，再将适配器和Wii Nunchuck游戏手柄连接好。这样我们的遥控机械臂就安装完成了。整体效果如图9.12所示。

图9.11 WiiNunchuck游戏手柄

图9.12 遥控机械臂

在使用Wii Nunchuck游戏手柄对机械臂进行控制之前，先编一段程序对Wii Nunchuck游戏手柄的硬件进行测试。测试程序需要引用Wire和WiiChuck两个类库函数，程序功能是获取游戏手柄内3轴加速度传感器、摇杆及两个按键的数值，并通过串口发送给计算机。代码如下：

```
/**********************************************
Wii Nunchuck游戏手柄测试程序

获取游戏手柄内3轴加速度传感器、摇杆及两个按键的数值

created 2011
by Nille
Email: chenille@126.com

This example code is in the public domain.
**********************************************/
#include <math.h>
#include <stdlib.h>
#include "Wire.h"
#include "WiiChuck.h"

WiiChuck wii = WiiChuck();

/**********************************************
              初始化部分——setup函数
**********************************************/
void setup()
{
      wii.init();
      Serial.begin(9600);
}

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

```
                执行部分——loop函数
*********************************************/
void loop()
{
        static int count = 0;

        if (true == wii.read())
                    {
                    Serial.print(count, DEC);
                    Serial.print("\t");                    //Tab键

                    //显示摇杆数值
                    Serial.print("joystick:");
                    Serial.print(wii.getJoyAxisX(), DEC);
                    Serial.print(",");
                    Serial.print(wii.getJoyAxisY(), DEC);
                    Serial.print(" \t");                   //Tab键

                    //显示3轴加速度传感器数值
                    Serial.print("accel:");
                    Serial.print(wii.getAccelAxisX(), DEC);
                    Serial.print(",");
                    Serial.print(wii.getAccelAxisY(), DEC);
                    Serial.print(",");
                    Serial.print(wii.getAccelAxisZ(), DEC);
                    Serial.print(" \t");                   //Tab键

                    //显示两个按键的数值
                    Serial.print("button:");
                    Serial.print(wii.getButtonZ(), DEC);
                    Serial.print(",");
                    Serial.print(wii.getButtonC(), DEC);

                    Serial.println(" ");
                    count ++;
        }
        delay(1000);
}
```

9.1.6 机械臂程序设计

Wii Nunchuck游戏手柄调试完成后，就可以用在机械臂的遥控上了。这里使用摇杆和两

个按键来选择所要控制的舵机，加速度传感器用来控制相应舵机的转动。使用时右手握住手柄，拇指向正前方拨动手柄上的摇杆不松手，则选中1号舵机，此时向右侧、左侧转动手柄，旋转底盘就会随之旋转；如果从左侧逐渐向右侧转动手柄，旋转底盘也会从左侧逐渐转动到右侧，所以当想让机械臂的某个部位转动到某个方向时，就可以利用手柄先选中这个部位对应的舵机，然后旋转手柄控制舵机旋转到设定的位置。控制2号舵机，拇指向正后方拨动摇杆，此时向上、向下转动手柄，2号舵机处的关节会随之上下运动；摇杆向正左方拨动则控制3号舵机；摇杆向正右方拨动则控制4号舵机；松开摇杆按下标着字母C的键则控制5号舵机；按下标着字母Z的按键则控制6号舵机。

相应的程序代码如下：

```
/*********************************************
Wii Nunchuck游戏手柄控制机械臂

使用摇杆和两个按键来选择所要控制的舵机，加速度传感器用来控制相应舵机的转动

使用时右手握住手柄

摇杆向正前方拨动控制1号舵机
摇杆向正后方拨动控制2号舵机
摇杆向正左方拨动控制3号舵机
摇杆向正右方拨动控制4号舵机
按下C键控制5号舵机
按下Z键控制6号舵机

created 2011
by Nille
Email: chenille@126.com

This example code is in the public domain.
*********************************************/
#include <Servo.h>
#include <math.h>
#include <stdlib.h>
#include "Wire.h"
#include "WiiChuck.h"

//定义6个舵机对象
Servo servo1;
Servo servo2;
Servo servo3;
Servo servo4;
Servo servo5;
```

```
Servo servo6;

WiiChuck wii = WiiChuck();
int x,y,xa,ya,c,z;

/*********************************************
                初始化部分——setup函数
*********************************************/
void setup()
{
      //定义舵机控制口
      servo1.attach(8);
      servo2.attach(9);
      servo3.attach(10);
      servo4.attach(11);
      servo5.attach(12);
      servo6.attach(13);

      //定义舵机的初始角度
      servo1.write(90);
      servo2.write(70);
      servo3.write(30);
      servo4.write(90);
      servo5.write(90);
      servo6.write(30);
      wii.init();
}

/*********************************************
                执行部分——loop函数
*********************************************/
void loop()
{
      if(true == wii.read())
      {
            //读取手柄的值
            x=wii.getJoyAxisX();          //摇杆x轴的值
            y=wii.getJoyAxisY();          //摇杆y轴的值
            xa=wii.getAccelAxisX();       //手柄x方向摆动的值
            ya=wii.getAccelAxisY();       //手柄y方向摆动的值
            c=wii.getButtonC();           //手柄按键C的值
            z=wii.getButtonZ();           //手柄按键Z的值
            xa = map(xa,0,1023,0,179);//将数值转换为0－180°的角度值
```

```
                    ya = map(ya,0,1023,0,179);//将数值转换为0～180°的角度值
}

          if(x>100 && x<150 && y==255)       //摇杆位置为向前
{
          if(xa<30)                 //限制最小值
          xa=30;
          if(xa>150)                //限制最大值
          xa=150;
          servo1.write(xa);         //给舵机旋转的度数
          delay(10);
}
else if(x>100 && x<150 &&y==0) //摇杆位置向后
{
          if(ya<30)                 //限制最小值
          xa=30;
          if(ya>150)                //限制最大值
          xa=150;
          servo2.write(ya);         //给舵机旋转的度数
          delay(10);

}
else if(x==0 && y>100 && y<150)//摇杆的位置向左
{
          ya=175-ya;                //为了让手柄在Y轴向前的时候，舵机3也向前
          if(ya<20)
          ya=20;
          if(ya>150)
          ya=150;
          servo3.write(ya);         //给舵机旋转的度数
          delay(10);
}
else if(x==255 && y>100 && y<150)     //摇杆的位置向右
{
          ya=175-ya;
          if(ya<30)
          ya=30;
          if(ya>80)
          ya=80;
          servo4.write(ya);              //给舵机旋转的度数
          delay(10);
}
```

```
        else if(c==0)                          //手柄按键C按下
        {
                if(xa<30)
                xa=30;
                if(xa>150)
                xa=150;
                servo5.write(xa);              //给舵机旋转的度数
                delay(10);
        }
        else if(z==0)                          //手柄按键Z按下
        {
                if(ya<40)
                ya=40;
                if(ya>110)
                ya=110;
                servo6.write(ya);              //给舵机旋转的度数
                delay(10);
        }
}
```

9.2 双足机器人

9.2.1 实例功能

本节制作一个简单的双足机器人，之所以说简单是因为实际上我们只制作了两条腿。作品也是6自由度，分别对应两条腿的髋关节、膝关节和腕关节。双足机器人在运动中要不断调整以保证平衡性，本书中没有介绍什么高深的算法来让双足机器人自身保持平衡，只是使用LabWindow/CVI制作了一个PC调试软件，方便我们对双足机器人进行调试。

9.2.2 器材列表

本实例使用到的器材如下所示：

- ❑ Arduino控制板
- ❑ XBee传感器扩展板V5
- ❑ 舵机×6
- ❑ 大容量5号电池（6节）
- ❑ 电池盒
- ❑ APC220模块
- ❑ 舵机支架

- ❑ 铝合金U型支架
- ❑ 铝合金L型支架
- ❑ 铝合金脚板结构件
- ❑ 铝合金U型梁
- ❑ 螺丝螺母若干
- ❑ 电路板安装板

9.2.3 搭建硬件环境

同样，在组装双足机器人之前，要先将舵机的角度调整到90°，使码盘的4个圆孔成正十字，如图9.1所示。

组装顺序是由下向上的，先在铝合金脚板结构件上安装一个舵机支架，如图9.13所示。

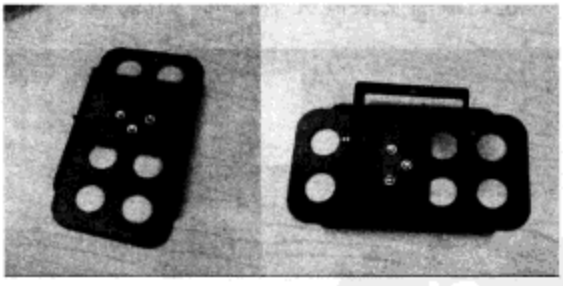


图9.13 脚板上安装一个舵机支架

接着在脚板上装上腕关节的舵机，同时用铝合金L型支架、铝合金U型支架及舵机支架组装膝关节，并将膝关节安装在腕关节上。组装效果如图9.14所示。

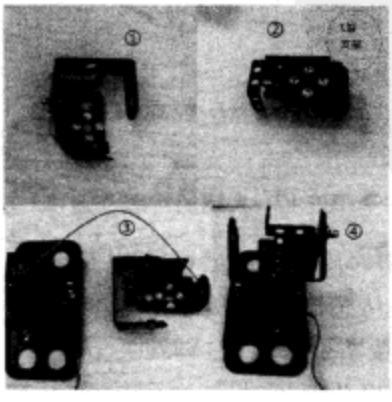


图9.14 膝关节、腕关节的组装

在膝关节的舵机支架上装上控制舵机，再装上一个铝合金U型支架，双足机器人的小腿就完成了。组装效果如图9.15所示。

按照相同的方法再组装一条小腿，组装的时候注意区分左右腿（见图9.16）。

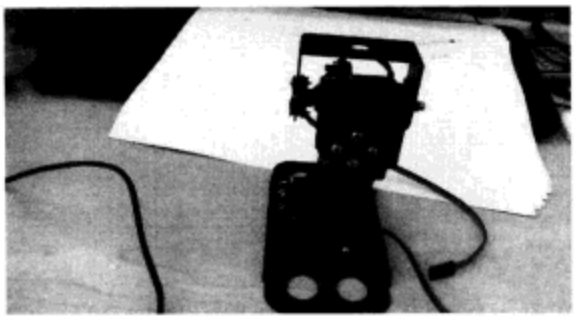


图9.15 小腿组装效果图

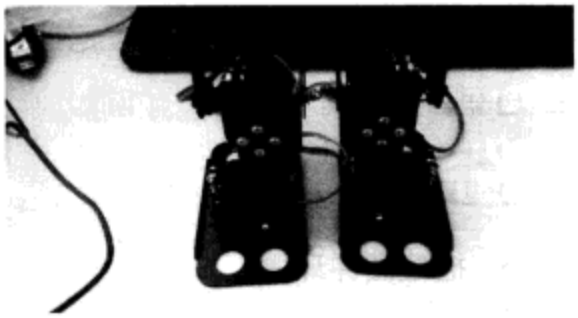


图9.16 再组装一条小腿

接下来就是大腿了，还是先把舵机支架和U型支架连起来再把它连接到小腿，同时用铝合金U型梁将两条腿连在一起，最后安装髋关节舵机。组装效果如图9.17所示。

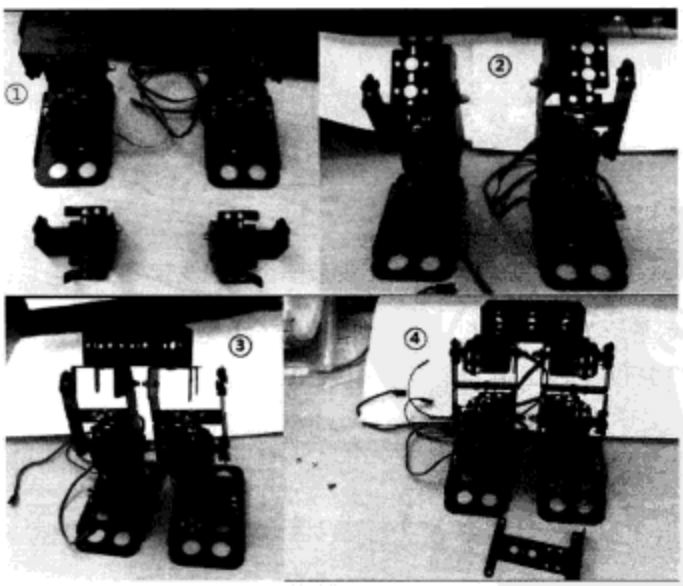


图9.17 大腿的组装及连接

最后将Arduino控制板、XBee传感器扩展板V5以及电池包安装在铝合金U型梁上，并将各个舵机的控制线插接在XBee传感器扩展板V5上。本书中双足机器人的左脚腕关节舵机定义为0号舵机，占用Arduino的数字引脚4；右脚腕关节舵机定义为1号舵机，占用Arduino的数字引脚5；左腿膝关节舵机定义为2号舵机，占用Arduino的数字引脚6；右腿膝关节舵机定义为3号舵机，占用Arduino的数字引脚7；左腿髋关节舵机定义为4号舵机，占用Arduino的数字引脚8；右腿髋关节舵机定义为5号舵机，占用Arduino的数字引脚9。至此，整个双足机器人的组装就完成了。效果如图9.18所示。

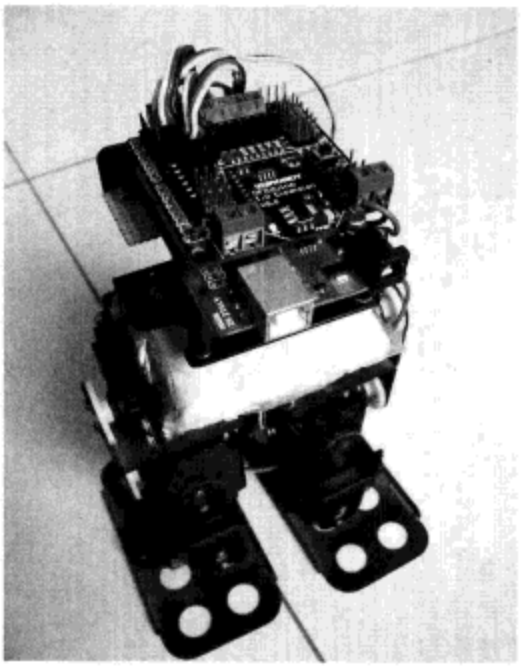


图9.18 双足机器人组装完成

9.2.4 双足机器人程序设计

本书中没有介绍什么高深的算法来让双足机器人在运动中自身保持平衡，而是通过LabWindow制作了一个PC调试软件对双足机器人进行调试，Arduino中的程序的功能就是与PC调试软件进行通信，然后控制相应的舵机动作从而实现双足机器人的运动。通信中发送的是ASCII码，以字符#开头，之后的一个字符表示所控制的舵机，编号范围0～5，最后的3个字符表示舵机的角度值，若角度值超过180°则判定舵机角度为90°。程序代码如下：

```
/*********************************************
双足机器人程序调试程序

Arduino与PC调试软件进行通信，然后控制相应的舵机动作
左脚腕关节舵机定义为0号舵机，占用Arduino的数字引脚4
右脚腕关节舵机定义为1号舵机，占用Arduino的数字引脚5
左腿膝关节舵机定义为2号舵机，占用Arduino的数字引脚6
右腿膝关节舵机定义为3号舵机，占用Arduino的数字引脚7
左腿髋关节舵机定义为4号舵机，占用Arduino的数字引脚8
右腿髋关节舵机定义为5号舵机，占用Arduino的数字引脚9
通信中发送的是ASCII码，以字符#开头
之后的一个字符表示所控制的舵机，编号范围0～5
最后的3个字符表示舵机的角度值，若角度值超过180°则判定舵机角度为90°
```

```
created 2011
by Nille
Email: chenille@126.com

This example code is in the public domain.
*************************************************/

#include <Servo.h>

// 定义6个舵机的对象
Servo myservo0 , myservo1 , myservo2 , myservo3 , myservo4 , myservo5;
int channel;              //定义保存舵机号的变量
int angle;                //定义保存角度值的变量

void setup()
{
      //为每个舵机对象分配管脚
      myservo0.attach(4);
      myservo1.attach(5);
      myservo2.attach(6);
      myservo3.attach(7);
      myservo4.attach(8);
      myservo5.attach(9);

      //初始化各个舵机的角度
      myservo0.write(90);
      myservo1.write(90);
      myservo2.write(90);
      myservo3.write(90);
      myservo4.write(90);
      myservo5.write(90);

      //定义串口波特率为9600bps
      Serial.begin(9600);
}

void loop()
{
      if ( Serial.available())
      {
                              if('#'== Serial.read())
                              {
                                        while(!Serial.available());
```

```
                                    //提取数据中的舵机号
                                    channel= Serial.read() ;

                                    while(!Serial.available());
                                    //提取数据中的角度值
                                    angle= Serial.read() - 0x30;

                          while(!Serial.available());
                                    angle= (Serial.read() - 0x30) + (angle*10);

                          while(!Serial.available());
                                    angle= (Serial.read() - 0x30) + (angle*10);

                          //如果角度值大于180° 则判定舵机角度为90°
                                    if(angle > 180)
                          angle = 90;

                          }

                          //根据通信数据控制相应的舵机动作
switch(channel)
{
                          case '0':
                                    myservo0.write(angle);
                                    break;
                                    case '1':
                                              myservo1.write(angle);
                                              break;
                                    case '2':
                                              myservo2.write(angle);
                                              break;
                                    case '3':
                                              myservo3.write(angle);
                                              break;
                                    case '4':
                                              myservo4.write(angle);
                                              break;
                                    case '5':
                                              myservo5.write(angle);
                                              break;
                                    default:
                                              break;
}
```

```
        }
    }
```

9.2.5 PC调试软件编写

双足机器人的PC调试软件采用LabWindows/CVI开放环境。LabWindows/CVI是一个完全的ANSIC开发环境，用于仪器控制、自动检测、数据处理的应用软件。它把C语言的有力和柔性同虚拟仪器的软件工具库结合起来，包含了各种总线、数据采集和分析库。编程采用事件驱动与回调函数方式，可大大地提高编程效率。双足机器人的PC调试软件界面如图9.19所示。

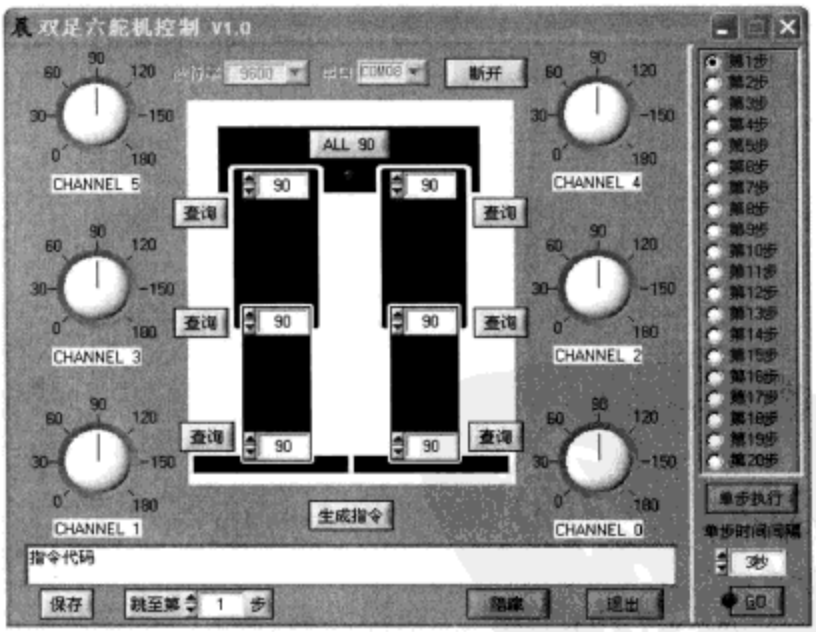


图9.19 双足六舵机控制软件界面

软件打开后首先要选择正确的波特率和串口号，在软件界面中可以通过六个旋钮分别调整每个舵机的角度值，角度值的范围在0～180°，当一个动作调整完成后，可以通过“生成指令”按钮生成发送的控制命令数据，指令代码会显示在“指令代码”文本框中，然后可以通过“保存”按钮将命令保存在右方的逐步运行条中。当通过软件保存了很多步的操作后，可以通过“单步执行”观察双足机器人的运动效果，也可以通过“GO”按钮自动执行每一步操作，自动执行每一步的时间间隔可以通过“单步时间间隔”数字框调整。

本节附上双足机器人的PC调试软件的部分代码。了解LabWindows/CVI的读者可以参考一下。

```
/*************************************************
双足机器人PC调试软件程序
```

```
开发环境LabWindows/CVI


通信中发送的是ASCII码，以字符#开头
之后的一个字符表示所控制的舵机，编号范围0~5
最后的3个字符表示舵机的角度值，若角度值超过180°则判定舵机角度为90°

created 2011
by Nille
Email: chenille@126.com

This example code is in the public domain.
*********************************************/
#include "toolbox.h"
#include <formatio.h>
#include <utility.h>
#include <rs232.h>
#include <cvirte.h>
#include <userint.h>
#include "DBLeg.h"

static int panelHandle;                    //面板句柄
static void *callbackdata;

//自定义变量
int ComPort;                               //使用的端口号
unsigned int SendBuffer[10];               //发送缓存区
char DictBuffer[100];                      //指令缓存区

int PreAngle;                              //之前的角度值
int PreChannel;                            //之前的通道值
int itemindex;
int CheckChannel;

void comcallback(int COMport,int eventMask,void *callbackdata);//串口回调函数

//初始化函数
void init(void)
{
                int i;
                //扩展鼠标响应事件
                EnableExtendedMouseEvents (PANEL, PANEL_NUMERICDIAL5, 0.1);
```

```
                EnableExtendedMouseEvents (PANEL, PANEL_NUMERICDIAL4, 0.1);
                EnableExtendedMouseEvents (PANEL, PANEL_NUMERICDIAL3, 0.1);
                EnableExtendedMouseEvents (PANEL, PANEL_NUMERICDIAL2, 0.1);
                EnableExtendedMouseEvents (PANEL, PANEL_NUMERICDIAL1, 0.1);
                EnableExtendedMouseEvents (PANEL, PANEL_NUMERICDIAL0, 0.1);

                //设置程序运行时属性，使其不在任务栏显示按钮
                SetPanelAttribute (panelHandle, ATTR_HAS_TASKBAR_BUTTON, 0);
                SetSystemAttribute (ATTR_TASKBAR_BUTTON_VISIBLE, 0);
    }

//单步执行子函数——使用文件管理功能
void StepByStep(void)
{
                int fileHandle;
                int     fileLeng;                                   //数据长度
                char fileName[14]="CODE\\01.nille";                 //文件名
                int optionIndex,checked,i;

                int tempAngle=0;
                int tempChannel=0;
                int flagAngle=0;
                int flagChannel=0;

                //从文本中获取单步指令
                for(optionIndex=0;optionIndex<20;optionIndex++)//获取序列号,得到步伐数
                {
                IsListItemChecked(panelHandle,PANEL_RADIOGROUP,optionIndex,&checked);
                if(checked==1)
                {break;}
                }

                //更改文件名
                fileName[5]=(optionIndex/10)+48;
                fileName[6]=(optionIndex%10)+48;

                //打开文件
                fileHandle=OpenFile(fileName,VAL_READ_WRITE,VAL_OPEN_AS_IS,VAL_ASCII);
                //读文件
                ReadLine(fileHandle,DictBuffer,90);
                //关闭文件
                CloseFile(fileHandle);
```

```
            //将指令代码显示在文本区
             ResetTextBox (panelHandle,PANEL_TEXTBOX, DictBuffer);

            if(DictBuffer[1]=='R')//判断是不是返回指令
            {optionIndex=((DictBuffer[2]-48)*10)+DictBuffer[3]-49;}
            else            //发送单步指令
            {
                    //判断文本框是否为空
                    GetNumTextBoxLines(panelHandle,PANEL_TEXTBOX,&checked);
                    if(checked>0)
                    {
                    //获取文本框的数据长度
                    GetTextBoxLineLength(panelHandle,PANEL_TEXTBOX,0,&fileLeng);

                    //将6个角度值从指令中分出来
                    for(i=0;i<fileLeng;i++)
                    {
                            if(DictBuffer[i]=='#')//遇到#号后面跟着通道
                            {
                                    //两个标记均为1说明读取了一次角度值
                                    if(flagChannel>0 && flagAngle>0)
                                    {
                                            flagChannel=0;
                                            flagAngle=0;
                                    }
                                    flagChannel=1;
                                    tempChannel=0;
                            }

                            //如果遇到的是数据
                            if(DictBuffer[i]>=48 && DictBuffer[i]<=57)
                            {
                                    if(flagChannel>0)//数据为通道数据
                                    {
                                    if(flagAngle>0)    //数据为角度数据
                                    {tempAngle=(tempAngle*10)+DictBuffer[i]-48;}
                                    else
{tempChannel=(tempChannel*10)+DictBuffer[i]-48;}
                                    }
                            }
                    }

                    for(i=0;i<fileLeng+1;i++)
```

```
                    {
                        ComWrtByte(ComPort,DictBuffer[i]);
                    }

                }
                optionIndex++;
                if(optionIndex>19)
                {optionIndex=0;}

                SetCtrlIndex (panelHandle, PANEL_RADIOGROUP,optionIndex);
                CheckListItem (panelHandle,PANEL_RADIOGROUP,optionIndex,1);
}


//主函数
int main (int argc, char *argv[])
{
                if (InitCVIRTE (0, argv, 0) == 0)
                return -1;  /* out of memory */
                if ((panelHandle = LoadPanel (0, "DBLeg.uir", PANEL)) < 0)
                return -1;

                DisplayPanel (panelHandle);
                init();

                RunUserInterface ();
                DiscardPanel (panelHandle);

                return 0;
}

//退出按钮回调函数
int CVICALLBACK QUITCBK (int panel, int control, int event,
                void *callbackData, int eventData1, int eventData2)
{
                switch (event)
                {
                        case EVENT_COMMIT:
                                GetCtrlVal (panelHandle,PANEL_RING, &ComPort);
                                CloseCom(ComPort);

                                QuitUserInterface (0);
                                break;
```

```
                }
                return 0;
}

//串口回调函数
void comcallback(int COMport,int eventMask,void *callbackdata)
{
                unsigned char buff[10];
                int inputdataleng,i;

                inputdataleng=GetInQLen(COMport);
                if(inputdataleng>0)                          //接收到数据
                {
                        buff[0]='\0';
                        ComRd(COMport,buff,inputdataleng);//读数据
                        buff[inputdataleng]='\0';
                        FlushInQ(COMport);

                        if(buff[0]==0)
                        {buff[0]=150;}
                }
}

////////////////////////////////////////////////////////////////////////////
//旋钮回调函数，此函数为旋钮5的回调函数，其他旋钮的回调函数类似
int CVICALLBACK Tune5CBK (int panel, int control, int event,
                void *callbackData, int eventData1, int eventData2)
{
                switch (event)
                {
                        case EVENT_LEFT_CLICK:
                                //相应鼠标左键按下事件
                                //定时器5开始工作
    SetCtrlAttribute(panelHandle,PANEL_TIMER5,ATTR_ENABLED,1);
break;
                        case EVENT_LEFT_MOUSE_UP:
                                //相应鼠标左键抬起事件
                                //定时器5停止工作
    SetCtrlAttribute(panelHandle,PANEL_TIMER5,ATTR_ENABLED,0);
break;
                }
                return 0;
}
```

```
//////////////////////////////////////////////////////////////////////////////
//软件中使用一个定时器用于当调整旋钮时发送数据，6个旋钮使用了6个定时器
//此函数为channel 5的定时器的回调函数，其他定时器的回调函数类似
int CVICALLBACK Timer5CBK (int panel, int control, int event,
                  void *callbackData, int eventData1, int eventData2)
{
                  int tempAngle;
                  int i;
                  switch (event)
                  {
                        case EVENT_TIMER_TICK:
                              //获取旋钮的值
                              GetCtrlVal (panelHandle,PANEL_NUMERICDIAL5,&tempAngle);
                              if(tempAngle!=PreAngle)
                              {
                                PreAngle=tempAngle;

                                SendBuffer[0]=0x23;          //#
                                SendBuffer[1]=0x35;          //舵机号
                                SendBuffer[2]=(tempAngle/100)+48; //角度值
                                SendBuffer[3]=((tempAngle/10)%10)+48;
                                SendBuffer[4]=(tempAngle%10)+48;

                                for(i=0;i<5;i++)
                                {
                                      ComWrtByte(ComPort,SendBuffer[i]);
                                }
                              }
                              break;
                  }
                  return 0;
}

//复位所有舵机子函数，使所有舵机回到90° 的位置
int CVICALLBACK AllMCBK (int panel, int control, int event,
                  void *callbackData, int eventData1, int eventData2)
{
                  int i,j;
                  switch (event)
                  {
                        case EVENT_COMMIT:
                              SendBuffer[0]=0x23;          //#
                              SendBuffer[2]=48;            //角度值
```

```
                    SendBuffer[3]=57;
                    SendBuffer[4]=48;

                    for(j=0;j<6;j++)
                    {
                      SendBuffer[1]=48+j;

                      for(i=0;i<5;i++)
                      {
                            ComWrtByte(ComPort,SendBuffer[i]);
                      }
                    }
                    //同时将6个旋钮的值复位
                    SetCtrlVal (panelHandle,PANEL_NUMERICDIAL0,90);
                    SetCtrlVal (panelHandle,PANEL_NUMERICDIAL1,90);
                    SetCtrlVal (panelHandle,PANEL_NUMERICDIAL2,90);
                    SetCtrlVal (panelHandle,PANEL_NUMERICDIAL3,90);
                    SetCtrlVal (panelHandle,PANEL_NUMERICDIAL4,90);
                    SetCtrlVal (panelHandle,PANEL_NUMERICDIAL5,90);
                    break;
            }
            return 0;
}

//单步执行回调函数
int CVICALLBACK StepCBK (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{
            switch (event)
            {
                  case EVENT_COMMIT:
                        StepByStep();
                        break;
            }
            return 0;
}
```

9.2.6 双足机器人的调试

PC调试软件编写完成后，就可以控制双足机器人运动了，这时把APC220模块插接在双足机器人上，就能实现对双足机器人的无线控制了，整体效果如图9.20所示。双足机器人的无线控制同样可以选用第7章介绍的其他无线模块，如果手上的电脑本身就带有蓝牙功能的

话，使用Bluetooth V3无线模块会更加方便。

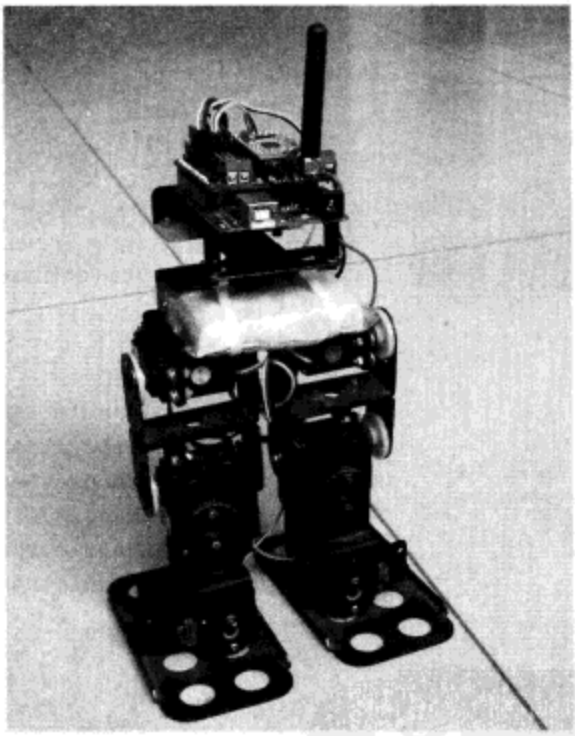


图9.20 遥控调试双足机器人

注意：由于左右腿的安装是相反的关系，所以在调试时要注意在执行相同的动作时，左边3个舵机与右边3个舵机的角度值也是相反，即左边角度值应等于180°减去右边的角度值。

附录A　Arduino引脚与AVR单片机管脚对应关系

表中列出了Arduino UNO的引脚与AVR单片机管脚的对应关系

Arduino UNO		AVR单片机管脚	扩展功能
数字引脚	模拟引脚		
0	/	PD 0	USART RX
1	/	PD 1	USART TX
2	/	PD 2	Ext Int 0
3	/	PD 3	PWM T2B, Ext Int 1
4	/	PD 4	
5	/	PD 5	PWM T0B
6	/	PD 6	PWM T0A
7	/	PD 7	
8	/	PB 0	Input capture
9	/	PB 1	PWM T1A
10	/	PB 2	PWM T1B, SS
11	/	PB 3	PWM T2A, MOSI
12	/	PB 4	SPI MISO
13	/	PB 5	SPI SCK
14	0	PC 0	
15	1	PC 1	
16	2	PC 2	
17	3	PC 3	
18	4	PC 4	I2C SDA
19	5	PC 5	I2C SCL

附录B　Arduino扩展板

L293 Motor Shield

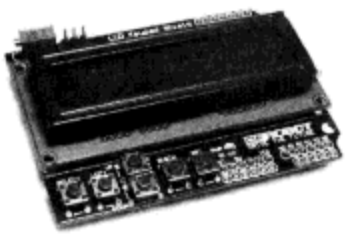
电机驱动板，可直接驱动直流电机、二相、四相步进电机，驱动电流达1A

L298P Motor Shield

L298P Shield直流电机驱动器采用LGS公司优秀大功率电机专用驱动芯片L298P，可直接驱动直流电机、二相、四相步进电机，驱动电流达2A，电机输出端采用8只高速肖特基二极管作为保护

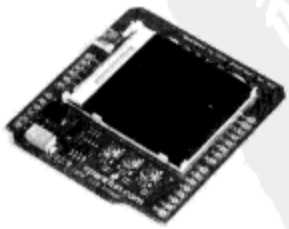
USB Host Shield

USB Host Shield支持Arduino UNO等，还支持MEGA1280、2560。可使Arduino非常方便地与一个USB从设备通信，例如USB键盘、鼠标、U盘、相机、安装Arduino操作系统的手机等



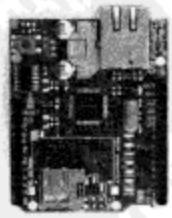
LCD Keypad Shield

使用2行16个字符液晶，具有对比度调节，使用1个模拟口模拟5个按键输入，1个复位按键，未使用的I/O口都扩展出来备用，充分利用I/O口



Color LCD Shield

使用Nokia 6100 彩色LCD作为显示的彩色LCD扩展板，可实现128x128像素的彩色图像显示



Ethernet Shield

以W5100为核心的网络扩展模块，可以使Arduino成为简单的Web服务器或者通过网络控制读写Arduino的数字和模拟接口等网络应用。可直接使用IDE中的Ethernet库文件实现一个简单Web服务器

Input Shield

DFduino输入扩展板，叠层设计可以方便地插到Arduino上，可配合APC220和Arduino Duemilanove作为无线遥控手柄

XBee传感器扩展板V5

XBee传感器扩展板V5主要侧重于一些传感器、无线通信模块的接口扩展，使这些模块能够方便地和Arduino连接

Interface shield

Interface shield扩展板侧重于总线接口的扩展，SPI接口、I IC接口、Micro SD卡接口（可直接插Micro SD卡）、SD卡存储模块接口、TLC5940接口等

WiFi Shield

基于WizFi210的WIFI无线模块提供TTL电平串口到IEEE802.11 b/g/n无线通信的桥接。任何具有TTL串口的设备都可以很容易地建立起无线网络，实现远程管理和控制。内部集成多种通信协议及其加密算法。只需要简单的设置即可实现多种场合的应用

MIDI Shield

MIDI shield可以让Arduino与MIDI设备连接，通过MIDI协议可以控制电子合成器、电子鼓、序列器以及其他MIDI音乐设备。MIDI协议使用标准异步串行通信接口，通信波特率为31.25×(1±0.01) KBaud，可以用Arduino通过串口的发送和接收与MIDI进行通信

GPS shield

本模块使用u-blox公司的GPS模块，定位精度是2.5米，定位性能优异准确，整体参数真实可靠，绝对没有其余修饰水分，在防漫反射及抗干扰能力十分强劲，是工程人员得以信赖的高精度模块之一，在韩国及瑞士拥有95%以上的GPS模块市场

可访问网址http://www.dfrobot.com.cn获取以上Arduino扩展板的详细资料。

附录C 其他可扩展模块

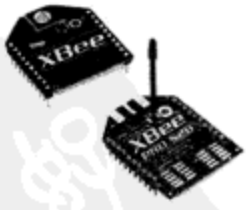
RGB LED Module
4×4全彩LED显示矩阵

IIC LCD2004
基于IIC总线控制的液晶模块

APC220
无线数据传输模块

DFduino wireless
无线数据传输模块

Bluetooth V3
蓝牙数据传输模块



XBee/XBee PRO
Zigbee数据传输模块

SPI LED Module
基于SPI的8位LED显示模块

URM37 V3.2超声波测距模块

IR Receiver Module
红外发射接收套件

SHT1x温湿度传感器

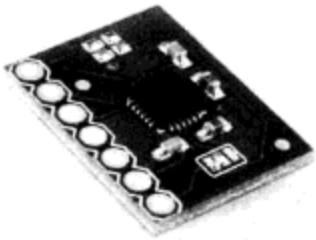
模拟气体传感器

磁感应传感器

继电器模块

模拟灰度传感器

触摸开关Touch



ITG-3200 3轴数字陀螺仪

MMA7260 3轴加速度

红外接近开关

可访问网址http://shop67239743.taobao.com获取以上扩展模块的详细资料。