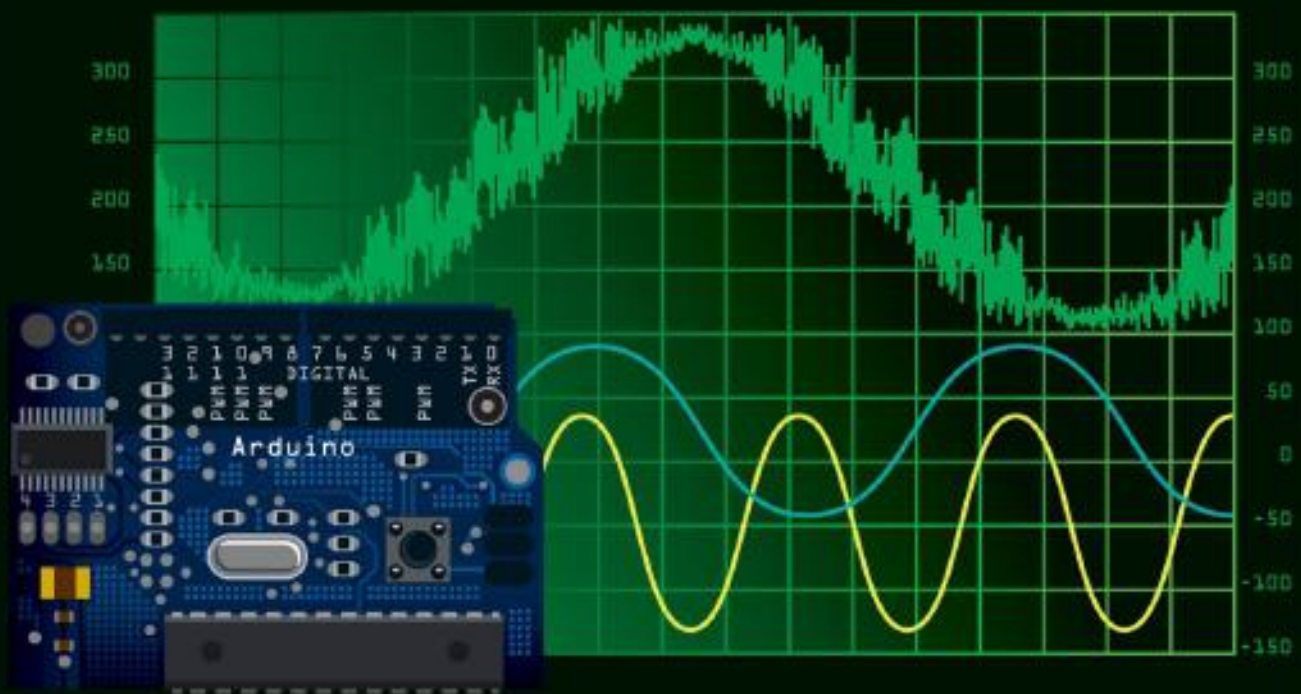
30 ARDUINO™ PROJECTS FOR THE EVIL GENIUS™

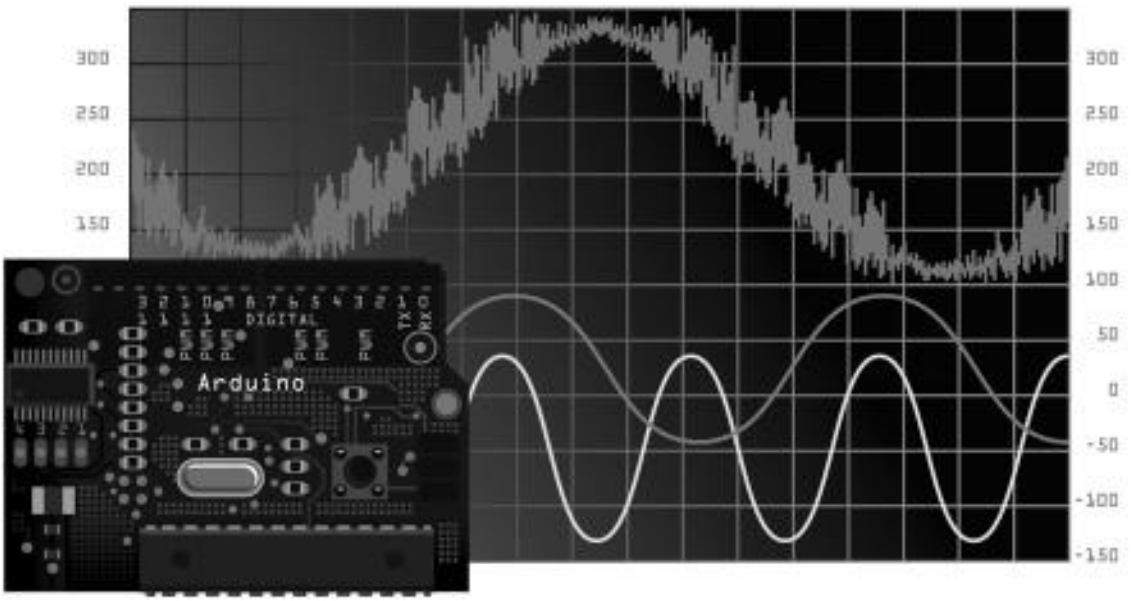


- Shows you how to program and build your own Arduino projects
- Filled with step-by-step instructions, illustrations, photos, and diagrams
- Covers Windows, Mac, and Linux platforms

TAB™

SIMON MONK

30 Arduino™ Projects for the Evil Genius™



Evil Genius™ Series

Bike, Scooter, and Chopper Projects for the Evil Genius
Bionics for the Evil Genius: 25 Build-it-Yourself Projects
Electronic Circuits for the Evil Genius, Second Edition: 64 Lessons with Projects
Electronic Gadgets for the Evil Genius: 28 Build-it-Yourself Projects
Electronic Sensors for the Evil Genius: 54 Electrifying Projects
50 Awesome Auto Projects for the Evil Genius
50 Green Projects for the Evil Genius
50 Model Rocket Projects for the Evil Genius
51 High-Tech Practical Jokes for the Evil Genius
46 Science Fair Projects for the Evil Genius
Fuel Cell Projects for the Evil Genius
Holography Projects for the Evil Genius
Mechatronics for the Evil Genius: 25 Build-it-Yourself Projects
Mind Performance Projects for the Evil Genius: 19 Brain-Bending Bio Hacks
MORE Electronic Gadgets for the Evil Genius: 40 NEW Build-it-Yourself Projects
101 Spy Gadgets for the Evil Genius
101 Outer Space Projects for the Evil Genius
123 PIC® Microcontroller Experiments for the Evil Genius
123 Robotics Experiments for the Evil Genius
125 Physics Projects for the Evil Genius
PC Mods for the Evil Genius: 25 Custom Builds to Turbocharge Your Computer
PICAXE Microcontroller Projects for the Evil Genius
Programming Video Games for the Evil Genius
Recycling Projects for the Evil Genius
Solar Energy Projects for the Evil Genius
Telephone Projects for the Evil Genius
30 Arduino Projects for the Evil Genius
22 Radio and Receiver Projects for the Evil Genius
25 Home Automation Projects for the Evil Genius

30 Arduino™ Projects for the Evil Genius™

Simon Monk

New York Chicago San Francisco Lisbon London Madrid
Mexico City Milan New Delhi San Juan Seoul
Singapore Sydney Toronto

ISBN: 978-0-07-174134-7

MHID: 0-07-174134-8

The material in this eBook also appears in the print version of this title: ISBN: 978-0-07-174133-0,
MHID: 0-07-174133-X.

To my late father, Hugh Monk, from whom I inherited a love for electronics.
He would have had so much fun with all this.

About the Author

Simon Monk has a bachelor's degree in cybernetics and computer science and a doctorate in software engineering. He has been an active electronics hobbyist since his school days, and is an occasional author in hobby electronics magazines.

Contents

Acknowledgments

I WOULD LIKE to thank my sons, Stephen and Matthew Monk, for their interest and encouragement in the writing of this book, their helpful suggestions, and their field testing of projects. Also, I could not have written this book without Linda's patience and support.

I am grateful to Chris Fitzer for the loan of his oscilloscope, and his good grace after I broke it! I also thank all the "techies" at Momote for taking an interest in the project and humoring me.

Finally, I would like to thank Roger Stewart and Joya Anthony at McGraw-Hill, who have been extremely supportive and enthusiastic, and have been a pleasure to work with.

This page intentionally left blank

Introduction

Arduino interface boards provide the Evil Genius with a low-cost, easy-to-use technology to create their evil projects. A whole new breed of projects can now be built that can be controlled from a computer. Before long, the computer-controlled, servo-driven laser will be complete and the world will be at the mercy of the Evil Genius!

This book will show the Evil Genius how to attach an Arduino board to their computer, to program it, and to connect all manner of electronics to it to create projects, including the computer-controlled, servo-driven laser mentioned earlier, a USB-controlled fan, a light harp, a USB temperature logger, a sound oscilloscope, and many more.

Full schematic and construction details are provided for every project, and most can be built without the need for soldering or special tools. However, the more advanced Evil Genius may wish to transfer the projects from a plug-in breadboard to something more permanent, and instructions for this are also provided.

So, What Is Arduino?

Well, Arduino is a small microcontroller board with a USB plug to connect to your computer and a number of connection sockets that can be wired up to external electronics, such as motors, relays, light sensors, laser diodes, loudspeakers, microphones, etc. They can either be powered through the USB connection from the computer or from a 9V battery. They can be controlled from the computer or programmed by the computer and then disconnected and allowed to work independently.

At this point, the Evil Genius might be wondering which top secret government organization they need to break into in order to acquire one. Well, disappointingly, no evil deeds at all are required to obtain one of these devices. The Evil Genius needs to go no further than their favorite online auction site or search engine. Since the Arduino is an open-source hardware design, anyone is free to take the designs and create their own clones of the Arduino and sell them, so the market for the boards is competitive. An official Arduino costs about $30, and a clone often less than $20.

The name “Arduino” is reserved by the original makers. However, clone Arduino designs often have the letters “duino” on the end of their name, for example, Freeduino or DFRduino.

The software for programming your Arduino is easy to use and also freely available for Windows, Mac, and LINUX computers at no cost.

Arduino

Although Arduino is an open-source design for a microcontroller interface board, it is actually rather more than that, as it encompasses the software development tools that you need to program an Arduino board, as well as the board itself. There is a large community of construction, programming, electronics, and even art enthusiasts willing to share their expertise and experience on the Internet.

To begin using Arduino, first go to the Arduino site (www.arduino.cc) and download the software for Mac, PC, or LINUX. You can then either buy an official Arduino by clicking the Buy An

Arduino button or spend some time with your favorite search engine or an online auction site to find lower-cost alternatives. In the next chapter, step-by-step instructions are provided for installing the software on all three platforms.

There are, in fact, several different designs of Arduino board. These are intended for different types of applications. They can all be programmed from the same Arduino development software, and in general, programs that work on one board will work on all.

In this book we mostly use the Arduino Duemilanove, sometimes called Arduino 2009, which is an update of the popular board, the Diecimila. Duemilanove is Italian for 2009, the year of its release. The older Diecimila name means 10,000 in Italian, and was named that after 10,000 boards had been manufactured. Most compatible boards such as the Freeduino are based on the Diecimila and Duemilanove designs.

Most of the projects in this book will work with a Diecimila, Duemilanove, or their clone designs, apart from one project that uses the Arduino Lilypad.

When you are making a project with an Arduino, you will need to download programs onto the board using a USB lead between your computer and the Arduino. This is one of the most convenient things about using an Arduino. Many microcontroller boards use separate programming hardware to get programs into the microcontroller. With Arduino, it's all contained on the board itself. This also has the advantage that you can use the USB connection to pass data back and forth between an Arduino board and your computer. For instance, you could connect a temperature sensor to the Arduino and have it repeatedly tell your computer the temperature.

On the older Diecimila boards, you will find a jumper switch immediately below the USB socket. With the jumper fitted over the top two pins, the board will receive its power from the USB connection. When over the middle and bottom pins, the board will be powered from an external power supply plugged into the socket below. On the newer Duemilanove boards, there is no such jumper and the supply switches automatically from USB to the 9V socket.

The power supply can be any voltage between 7 and 12 volts. So a small 9V battery will work just fine for portable applications. Typically, while you are making your project, you will probably power it from USB for convenience. When you are ready to cut the umbilical cord (disconnect the USB lead), you will want to power the board independently. This may be with an external power adaptor or simply with a 9V battery connected to a plug to fit the power socket.

There are two rows of connectors on the edges of the board. The row at the top of the diagram is mostly digital (on/off) pins, although any marked with "PWM" can be used as analog outputs. The bottom row of connectors has useful power connections on the left and analog inputs on the right.

These connectors are arranged like this so that so-called "shield" boards can be plugged on to the main board in a piggyback fashion. It is possible to buy ready-made shields for many different purposes, including:

- Connection to Ethernet networks
- LCD displays and touch screens
- XBee (wireless data communications)
- Sound
- Motor control
- GPS tracking
- And many more

You can also use prototyping shields to create your own shield designs. We will use these Protoshields in some of our projects. Shields usually have through connectors on their pins, which means that you can stack them on top of

each other. So a design might have three layers: an Arduino board on the bottom, a GPS shield on it, and then an LCD display shield on top of that.

The Projects

The projects in this book are quite diverse. We begin with some simple examples using standard LEDs and also the ultra high-brightness Luxeon LEDs.

In Chapter 5, we look at various sensor projects for logging temperature and measuring light and pressure. The USB connection to the Arduino makes it possible to take the sensor readings in these projects and pass them back to the computer, where they can be imported into a spreadsheet and charts drawn.

We then look at projects using various types of display technology, including an alphanumeric LCD message board (again using USB to get messages from your computer), as well as seven-segment and multicolor LEDs.

Chapter 7 contains four projects that use sound as well as a simple oscilloscope. We have a simple project to play tunes from a loudspeaker, and build up to a light harp that changes the pitch and volume of the sound by waving your hand over light sensors. This produces an effect rather like the famous Theremin synthesizer. The final project in this chapter uses sound input from a microphone. It is a VU meter that displays the intensity of the sound on an LED display.

The final chapters contain a mixture of projects. Among others, there is, as we have already mentioned, an unfathomable binary clock using an Arduino Lilypad board that indicates the time in an obscure binary manner only readable by an Evil Genius, a lie detector, a motor-controlled swirling hypnotizer disk, and, of course, the computer-controlled-servo-guided laser.

Most of the projects in this book can be constructed without the need for soldering; instead we use a breadboard. A breadboard is a plastic block with holes in it with sprung metal connections behind. Electronic components are pushed through the holes at the front. These are not expensive, and a suitable breadboard is also listed in the appendix. However, if you wish to make your designs more permanent, the book shows you how to do that, too, using the prototyping board.

Sources for all the components are listed in the appendix, along with some useful suppliers. The only things you will need in addition to these components are an Arduino board, a computer, some wire, and a piece of breadboard. The software for all the projects is available for download from www.arduinoevilgenius.com.

Without Further Ado

The Evil Genius is not noted for their patience, so in the next chapter we will show you how to get started with Arduino as quickly as possible. This chapter contains all the instructions for installing the software and programming your Arduino board, including downloading the software for the projects, so you will need to read it before you embark on your projects.

In Chapter 2 we take a look at some of the essential theory that will help you build the projects described in this book, and go on to design projects of your own. Most of the theory is contained in this chapter, so if you are the kind of Evil Genius who prefers to just make the projects and find out how they work afterwards, you may prefer, after reading Chapter 1, to just to pick a project and start building. Then if you get stuck, you can use the index or read some of the early chapters.

This page intentionally left blank

CHAPTER 1

Quickstart

This is a chapter for the impatient Evil Genius. Your new Arduino board has arrived and you are eager to have it do something.

So, without further ado...

Powering Up

When you buy an Arduino Diecimila or Duemilanove board, it is usually preinstalled with a sample Blink program that will make the little built-in LED flash. Figure 1-1 shows an Arduino-compatible board with the LED lit.

The light-emitting diode (LED) marked L is wired up to one of the digital input-output sockets on the board. It is connected to digital pin 13. This really limits pin 13 to being used as an output, but the LED only uses a small amount of current, so you can still connect other things to that connector.

All you need to do to get your Arduino up and running is supply it with some power. The easiest way to do this is to plug in it into the Universal Serial Bus (USB) port on your computer. You will need a type A-to-type B USB lead. This is the same type of lead that is normally used to connect a computer to a printer.

If you are using the older Arduino Diecimila board, make sure that the power jumper is in the USB position (see Figure 1-1). The jumper should connect together the two top pins to allow the board to be powered from the USB. The newer Arduino Duemilanove boards do not have this jumper and select the power source automatically.

If everything is working okay, the LED should blink once every two seconds. The reason that new Arduino boards have this Blink sketch already installed is to verify that the board works. If your board does not start to blink when connected, check the position of the power jumper (if it has one) and try a different USB socket, possibly on a different computer, as some USB sockets are capable of supplying more power than others. Also, clicking the Reset button should cause the LED to flicker momentarily. If this is the case, but the LED does not flash, then it may just be that the board has not been programmed with the Flash sketch; but do not despair, as once everything is installed, we are going to modify and install that script anyway as our first project.

Installing the Software

Now we have our Arduino working, let's get the software installed so that we can alter the Blink program and send it down to the board. The exact procedure depends on what operating system you use on your computer. But the basic principle is the same for all.

Install the USB driver that allows the computer to talk to the Arduino's USB port. It uses this for programming and sending messages.

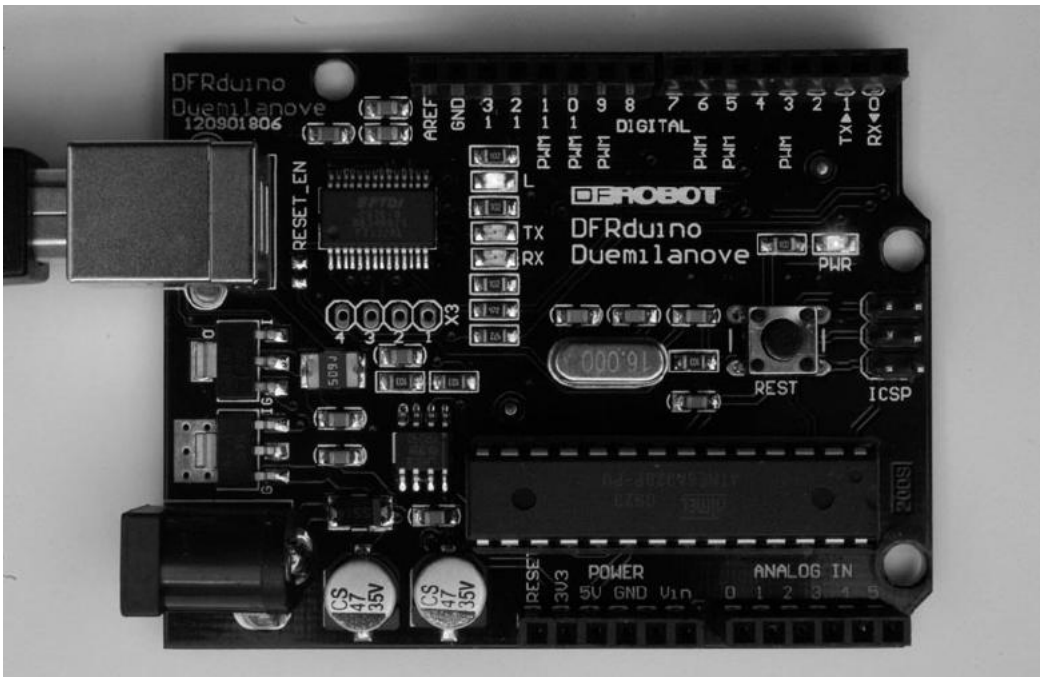


Figure 1-1 A powered-up Arduino board with LED lit.

Install the Arduino development environment, which is the program that you run on your computer that enables you to write sketches and download them to the Arduino board.

The Arduino website (www.arduino.cc) contains the latest version of the software.

Installation on Windows

Follow the download link on the Arduino home page (www.arduino.cc) and select the download for Windows. This will start the download of the Zip archive containing the Arduino software, as shown in Figure 1-2. You may well be downloading a more recent version of the software than the version 17 shown. This should not matter, but if you experience any problems, refer back to the instructions on the Arduino home page.

The Arduino software does not distinguish between different versions of Windows. The download should work for all versions, from Windows XP onwards. The following instructions are for Windows XP.

Select the Save option from the dialog, and save the Zip file onto your desktop. The folder contained in the Zip file will become your main Arduino directory, so now unzip it into C:\Program Files\Arduino.

You can do this in Windows XP by right-clicking the Zip file to show the menu in Figure 1-3 and selecting the Extract All option. This will open the Extraction Wizard, shown in Figure 1-4.

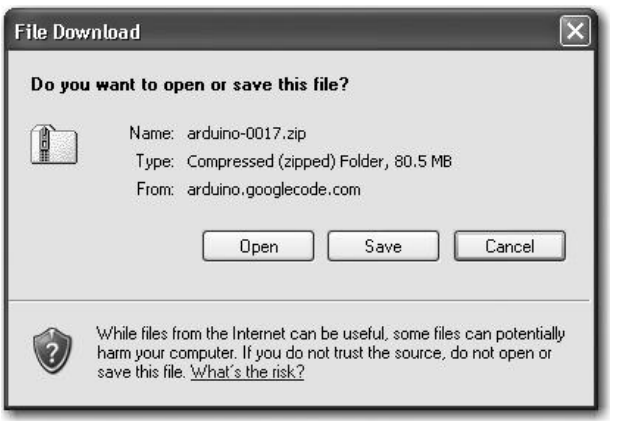


Figure 1-2 Downloading the Arduino software for Windows.

Click Next and then modify the folder to extract files to C:\Program Files\Arduino as shown in Figure 1-5. Then click Next again.

This will create a new directory for this version of Arduino (in this case, 17) in the folder C:\Program Files\Arduino. This allows you to have multiple versions of Arduino installed at the same time, each in its own folder. Updates of Arduino are fairly infrequent and historically have always kept compatibility with earlier versions of the software. So unless there is a new feature of the software that you want to use, or you have been having problems, it is by no means essential to keep up with the latest version.

Now that we have got the Arduino folder in the right place, we need to install the USB drivers. We let Windows do this for us by plugging in the Arduino board to trigger the Windows Found New Hardware Wizard shown in Figure 1-6.

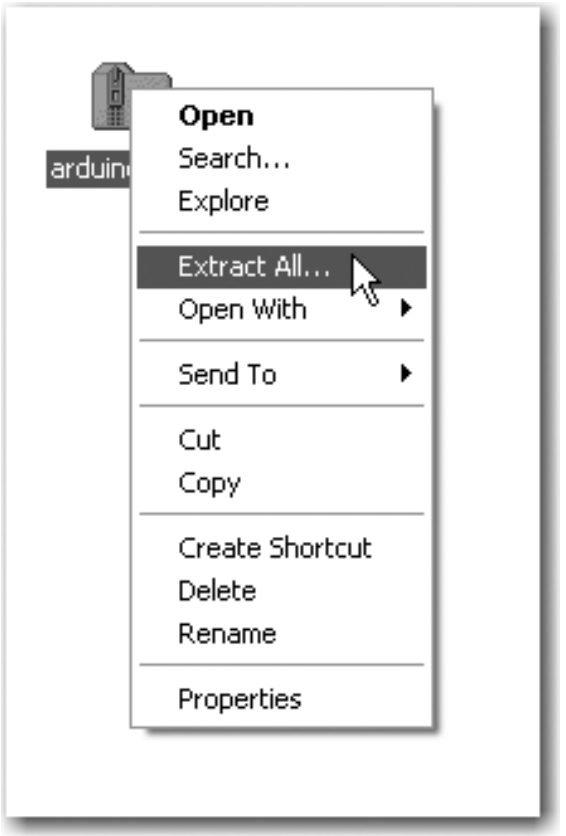


Figure 1-3 The Extract All menu option in Windows.

Figure 1-4 Extracting the Arduino file in Windows.

Figure 1-5 Setting the directory for extraction.

Select the option No, Not This Time, and then click Next.

On the next screen (Figure 1-7), click the option to install from a specified location, enter or browse to the location C:\Program Files\Arduino\arduino-0017\drivers\FTDI USB Drivers, and then click Next. Note that you will have to change 0017 in the path noted if you download a different version.

The installation will then complete and you are ready to start up the Arduino software itself. To do this, go to My Computer, navigate to C:\Program

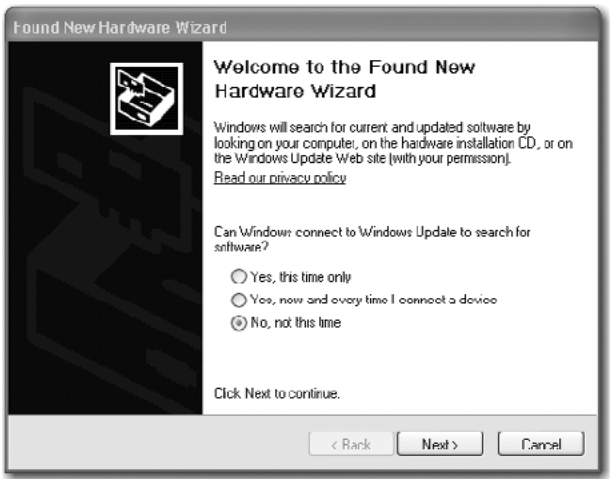


Figure 1-6 Windows Found New Hardware Wizard.

Figure 1-7 Setting the location of the USB drivers.

Files\Arduino\arduino-0017, and click the Arduino icon, as shown in Figure 1-8. The Arduino software will now start.

Note that there is no shortcut created for the Arduino program, so you may wish to select the Arduino program icon, right-click, and create a shortcut that you can then drag to your desktop.

The next two sections describe this same procedure for installing on Mac and LINUX computers, so if you are a Windows user, you can skip these sections.

Installation on Mac OS X

The process for installing the Arduino software on the Mac is a lot easier than on the PC.

Figure 1-8 Starting the Arduino software from Windows.

As before, the first step is to download the file. In the case of the Mac, it is a disk image file. Once downloaded, it will mount the disk image and open a Finder window, as shown in Figure 1-9. The Arduino application itself is installed in the usual Mac way by dragging it from the disk image to your Applications folder.

The disk image also contains two installer packages for the USB drivers (see Figure 1-10). Be sure to choose the package for your system architecture. Unless you are using a Mac built before March 2006, you will need to use the Intel version rather than the PPC version.

When you run the installer, you can simply click Continue until you come to the Select Disk screen, where you must select the hard disk before clicking Continue. As this software installs a kernel extension, it will prompt you to enter your password before completing the installation.

You can now find and launch the Arduino software in your Applications folder. As you are going to use it frequently, you may wish to right-click its icon in the dock and set it to Keep In Dock.

Figure 1-9 Installing the Arduino software on Mac OS X.

You can now skip the next subsection, which is for installation on LINUX.

Installation on LINUX

There are many different LINUX distributions, and for the latest information, refer to the Arduino home page. However, for most versions of LINUX, installation is straightforward. Your LINUX will

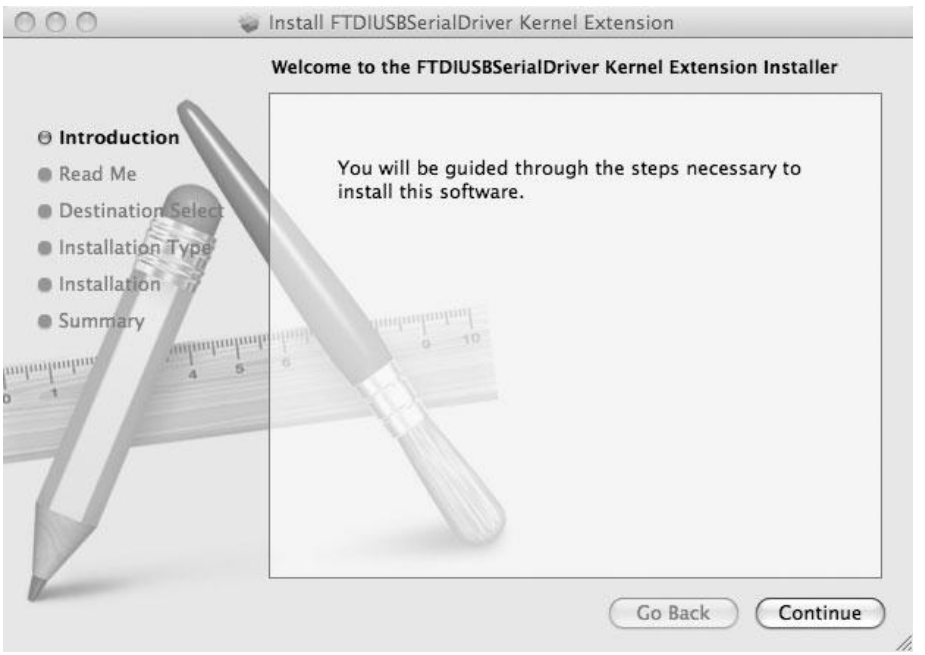


Figure 1-10 Installing the USB drivers on Mac OS X.

probably already have the USB drivers installed, the AVR-GCC libraries, and the Java environment that the Arduino software needs.

So, if you are lucky, all you will need to do is download the TGZ file for the Arduino software from the Arduino home page (www.arduino.cc), extract it, and that is your working Arduino directory.

If, on the other hand, you are unlucky, then as a LINUX user, you are probably already adept at finding support from the LINUX community for setting up your system. The pre-requisites that you will need to install are Java runtime 5 or later and the latest AVR-GCC libraries.

Entering into Google the phrase "Installing Arduino on SUSE LINUX," or whatever your distribution of LINUX is, will, no doubt, find you lots of helpful material.

Configuring Your Arduino Environment

Whatever type of computer you use, you should now have the Arduino software installed on it. We now need to make a few settings. We need to specify the operating system name for the port that is connected to the USB port for communicating with the Arduino board, and we need to specify the type of Arduino board that we are using. But first, you need to connect your Arduino to your computer using the USB port or you will not be able to select the serial port.

The serial port is set from the Tools menu, as shown in Figure 1-11 for the Mac and in Figure 1-12 for Windows—the list of ports for LINUX is similar to the Mac.

If you use many USB or Bluetooth devices with your Mac, you are likely to have quite a few options in this list. Select the item in the list that begins with "dev/tty.usbserial."

On Windows, the serial port can just be set to COM3.

From the Tools menu, we can now select the board that we are going to use, as shown in Figure 1-13. If you are using the newer Duemilanove, choose the first option. However, if you are using the older Diecimila board, select the second option.

Downloading the Project Software

The software for all of these sketches is available for download. The whole download is less than a megabyte, so it makes sense to download the software for all of the projects, even if you only intend to use a few. To download them, browse to www.arduinoevilgenius.com and click Downloads at the top of the screen.

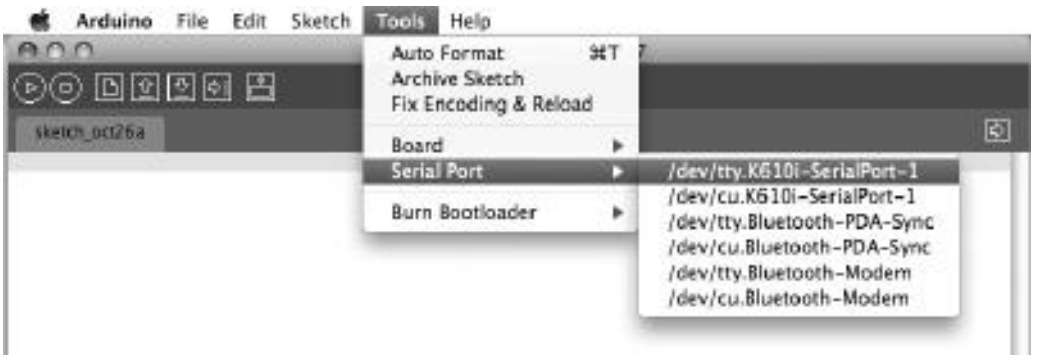


Figure 1-11 Setting the serial port on the Mac.

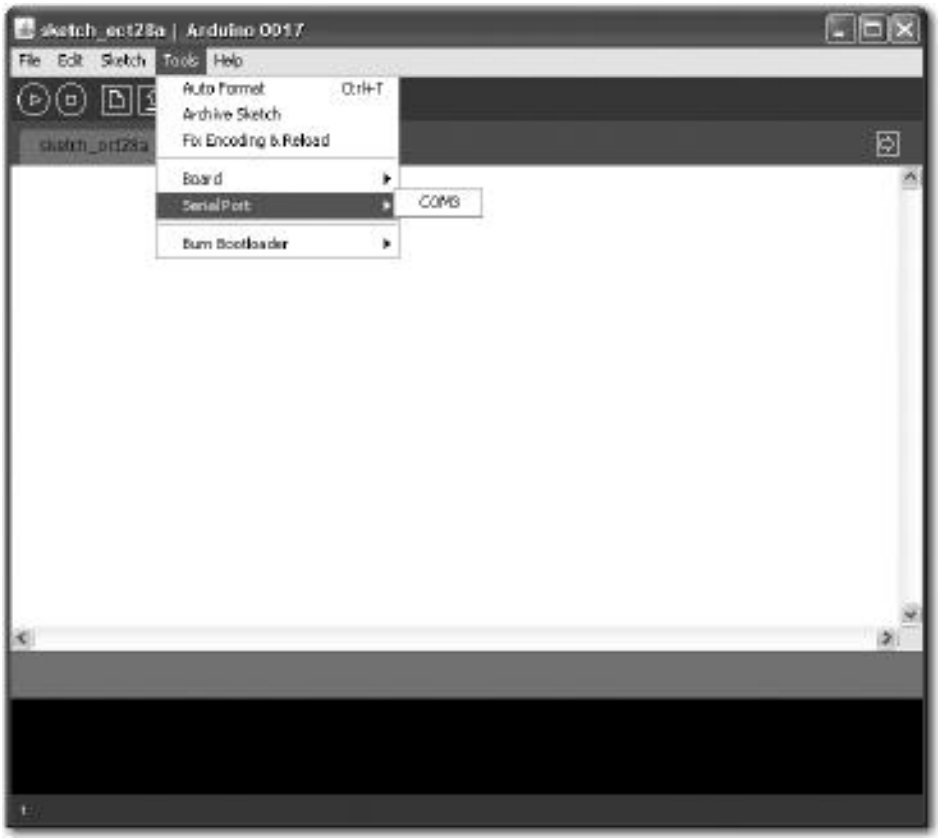


Figure 1-12 Setting the serial port on Windows.

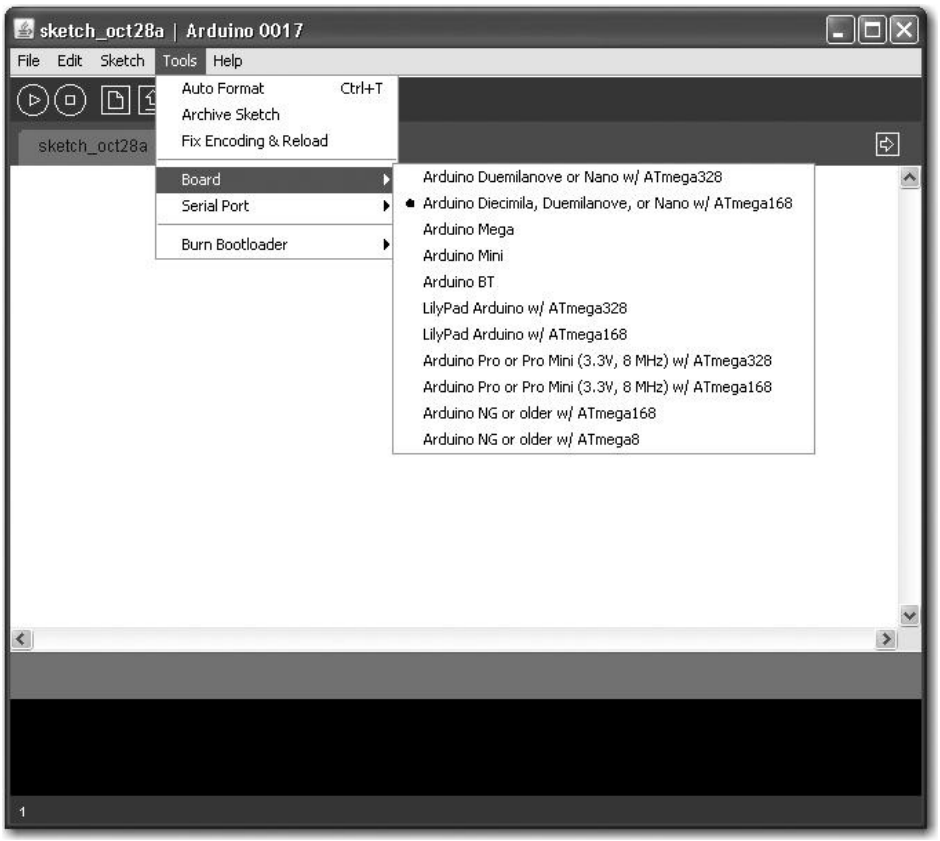


Figure 1-13 Setting the board.

Click the evil_genius.zip link to download a Zip file of all the projects. If you are using Windows, unzip the file to My Documents\Arduino. On a Mac and LINUX, you should unzip it to Documents/Arduino in your home directory.

Once the files are installed, you will be able to access them from the File | Sketchbook menu on the Arduino software.

Project 1
Flashing LED

Having assumed that we have successfully installed the software, we can now start on our first exciting project. Actually, it's not that exciting, but we need to start somewhere, and this will ensure that we have everything set up correctly to use our Arduino board.

We are going to modify the example Blink sketch that comes with Arduino. We will increase the frequency of the blinking and then install the modified sketch on our Arduino board. Rather than blink slowly, our board will flash its LED quickly. We will then take the project a stage further by using a bigger external LED and resistor rather than the tiny built-in LED.

COMPONENTS AND EQUIPMENT

	Description	Appendix
	Arduino Diecimila or Duemilanove board or clone	1
D1	5-mm red LED	23
R1	270 Ω 0.5W metal film resistor	6

- In actual fact, almost any commonly available LED and 270 Ω resistor will be fine.
- No tools other than a pair of pliers or wire cutters are required.
- The number in the Appendix column refers to the component listing in the appendix, which lists part numbers for various suppliers.

Software

First, we need to load the Blink sketch into the Arduino software. The Blink sketch is included as an example when you install the Arduino environment. So we can load it using the File menu, as shown in Figure 1-14.

The majority of the text in this sketch is in the form of comments. Comments are not actually part of the program but explain what is going on in the program to anyone reading the sketch.

Comments can be single-line comments that start after a // and continue to the end of the line, or they can be multiline comments that start with a /* and end some lines later with a */.

If all the comments in a sketch were to be removed, it would still work in exactly the same way, but we use comments because they are useful to anyone reading the sketch trying to work out what it does.

Before we start, a little word about vocabulary is required. The Arduino community uses the word "sketch" in place of "program," so from now on, I will refer to our Arduino programs as sketches. Occasionally I may refer to "code." Code is programmer speak for a section of a program or even as a generic term for what is written when creating a program. So, someone might say, "I wrote a program to do that," or they could say, "I wrote some code to do that."

To modify the rate at which the LED will blink, we need to change the value of the delay so that in the two places in the sketch where we have:

```
delay(1000);
```

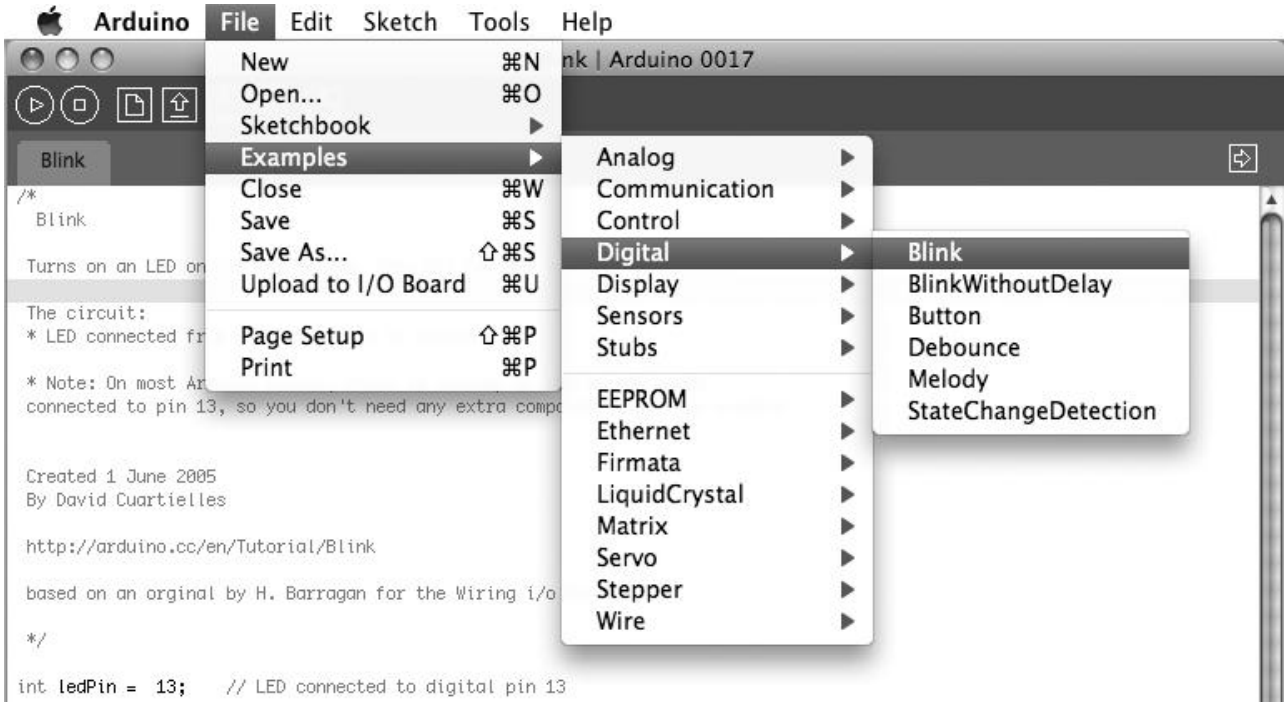


Figure 1-14 Loading the example Blink sketch.

change the value in the parentheses to 200 so that it appears as:

```
delay(200);
```

This is changing the delay between turning the LED on and off from 1000 milliseconds (1 second) to 200 milliseconds (1/5th of a second). In Chapter 3 we will explore this sketch further, but for now, we will just change the delay and download the sketch to the Arduino board.

With the board connected to your computer, click the Upload button on the Arduino. This is shown in Figure 1-15. If everything is okay, there will be a short pause and then the two red LEDs on the board will start flashing away furiously as the sketch is uploaded onto the board. This should take around 5 to 10 seconds.

If this does not happen, check the serial port and board type settings as described in the previous sections.

When the completed sketch has been installed, the board will automatically reset, and if everything has worked, you will see the LED for digital port 13 start to flash much more quickly than before.

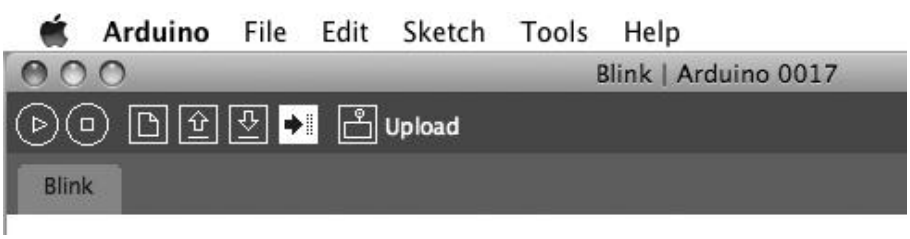


Figure 1-15 Uploading the sketch to the Arduino board.

Hardware

At the moment, this doesn't really seem like real electronics because the hardware is all contained on the Arduino board. In this section, we will add an external LED to the board.

LEDs cannot simply have voltage applied to them; they must have a current-limiting resistor attached. Both parts are readily available from any electronics suppliers. The component order codes for a number of suppliers are detailed in the appendix.

The Arduino board connectors are designed to attach "shield" plug-in boards. However, for experimentation purposes, they also allow wires or component leads to be inserted directly into the sockets.

Figure 1-16 shows the schematic diagram for attaching the external LED.

This kind of schematic diagram uses special symbols to represent the electronic components. The LED appears rather like an arrow, which indicates that light-emitting diodes, in common with all diodes, only allow the current to flow in one direction. The little arrows next to the LED symbol indicate that it emits light.

The resistor is just depicted as a rectangle. Resistors are also often shown as a zigzag line. The rest of the lines on the diagram represent electrical connections between the components. These connections may be lengths of wire or tracks on a circuit board. In this case, they will just be the wires of the components.

We can connect the components directly to the Arduino sockets between the digital pin 12 and the GND pin, but first we need to connect one lead of the LED to one lead of the resistor.

It does not matter which lead of the resistor is connected to the LED; however, the LED must be connected the correct way. The LED will have one lead slightly longer than the other, and it is the longer lead that must be connected to digital pin 12 and the shorter lead that should be connected to the resistor. LEDs and some other components have the convention of making the positive lead longer than the negative one.

To connect the resistor to the short lead of the LED, gently spread the leads apart and twist the short lead around one of the resistor leads, as shown in Figure 1-17.

Then push the LED's long lead into the digital pin 12 and the free lead of the resistor into one of

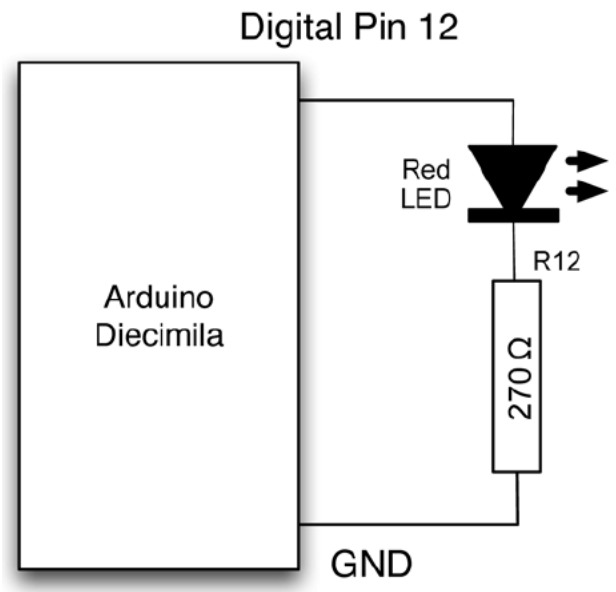


Figure 1-16 Schematic diagram for an LED connected to the Arduino board.

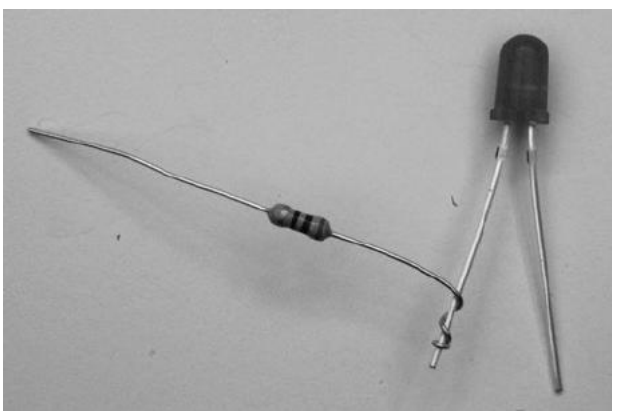


Figure 1-17 An LED connected to a serial resistor.

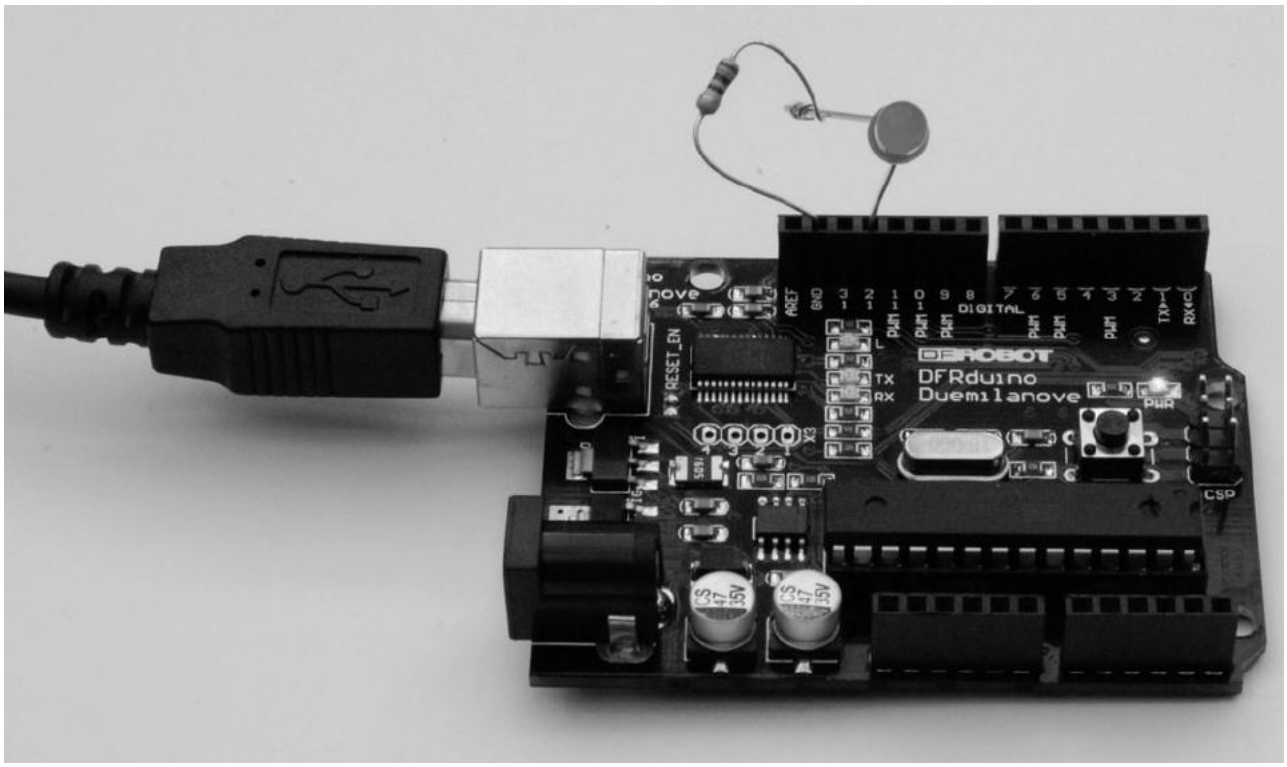


Figure 1-18 An LED connected to the Arduino board.

the two GND sockets. This is shown in Figure 1-18. Sometimes, it helps to bend a slight kink into the end of the lead so that it fits more tightly into the sockets.

We can now modify our sketch to use the external LED that we have just connected. All we need to do is change the sketch so that it uses digital pin 12 instead of 13 for the LED. To do this, we change the line:

```
int ledPin = 13;
  // LED connected to digital pin 13
```

to read:

```
int ledPin = 12;
  // LED connected to digital pin 12
```

Now upload the sketch by clicking the Upload To IO Board button in the same way as you did when modifying the flash rate.

Breadboard

Twisting together a few wires is not practical for anything much more than a single LED. A breadboard allows us to build complicated circuits without the need for soldering. In fact, it is a good idea to build all circuits on a breadboard first to get the design right and then commit the design to solder once everything is working.

A breadboard comprises a plastic block with holes in it, with sprung metal connections behind. Electronic components are pushed through the holes at the front.

Underneath the breadboard holes, there are strips of connectors, so each of the holes in a strip are connected together. The strips have a gap between them so that integrated circuits in dual-in-line packaging can be inserted without leads on the same row being shorted together.

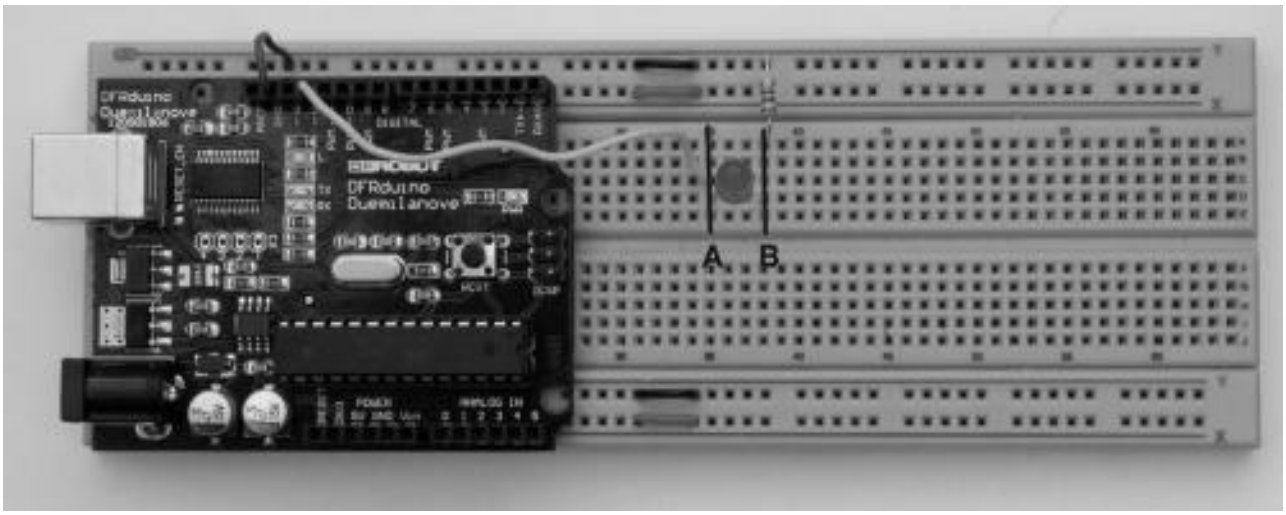


Figure 1-19 Project 1 on breadboard.

We can build this project on a breadboard rather than with twisted wires. Figure 1-19 shows a photograph of this. Figure 1-20 makes it a little easier to see how the components are positioned and connected together.

You will notice that at the edges of the breadboard (top and bottom), there are two long horizontal strips. The connections on the back of these long strips run at right angles to the normal strips of connections and are used to provide power to the components on the breadboard. Normally, there is one for ground (0V or GND) and one for the positive supply voltage (usually 5V). There are little linking wires between the left and right halves of the GND strip, as on this breadboard, as it does not go the whole width of the board.

In addition to a breadboard, you will need some solid-core wire and some wire strippers or pliers to cut and remove the insulation from the ends of the wire. It is a good idea to have at least three different colors: red for all wires connected to the positive side of the supply, black for negative, and some other color (orange or yellow) for other connections. This makes it much easier to understand the layout of the circuit. You can also buy prepared short lengths of solid-core wire in a variety of colors. Note that it is not advisable to use multicore wire, as it will tend to bunch up when you try to push it into the breadboard holes.

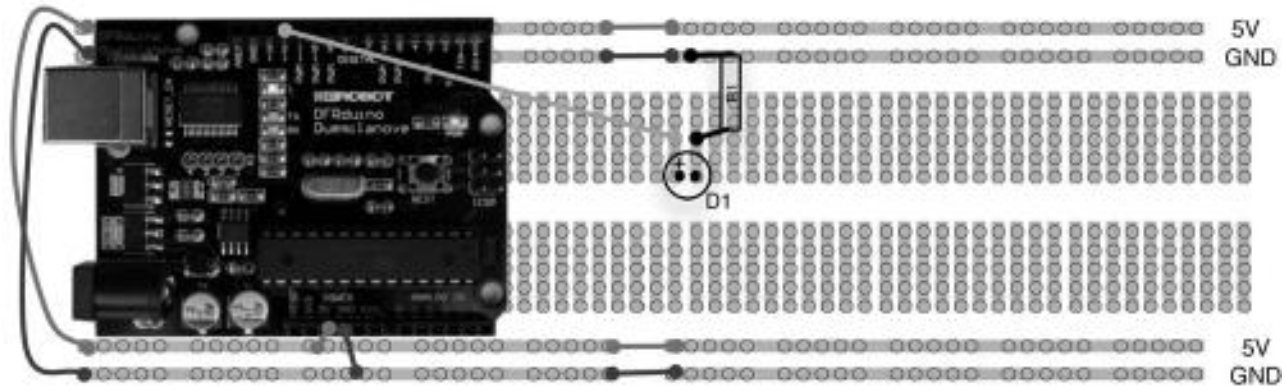


Figure 1-20 Project 1 breadboard layout.

Possible sources of these materials are included in the appendix.

We can straighten out the wires of our LED and resistor and plug them into a breadboard. It is best to use a reasonable-sized breadboard and attach the Arduino board to it. You probably do not want to attach the board permanently, so I use a small lump of adhesive putty. However, you may find it easier to dedicate one Arduino board to be your design board and leave it permanently attached to the breadboard.

Summary

We have created our first project, albeit a very simple one. In the next chapter we will get a bit more background on the Arduino before moving on to some more interesting projects.

This page intentionally left blank

CHAPTER 2

A Tour of Arduino

In this chapter, we look at the hardware of the Arduino board and also of the microcontroller at its heart. In fact, the board basically just provides support to the microcontroller, extending its pins to the connectors so that you can connect hardware to them and providing a USB link for downloading sketches, etc.

We also learn a few things about the C language used to program the Arduino, something we will build on in later chapters as we start on some practical project work.

Although this chapter gets quite theoretical at times, it will help you understand how your projects work. However, if you would prefer just to get on with your projects, you may wish to skim this chapter.

Microcontrollers

The heart of our Arduino is a microcontroller. Practically everything else on the board is concerned with providing the board with power and allowing it to communicate with your desktop computer.

So what exactly do we get when we buy one of these little computers to use in our projects?

The answer is that we really do get a little computer on a chip. It has everything and more than the first home computers had. It has a processor, a kilobyte of random access memory (RAM) for holding data, a few kilobytes of erasable programmable read-only memory (EPROM) or Flash memory for holding our programs, and it has input and output pins. These input/output pins are what link the microcontroller to the rest of our electronics.

Inputs can read both digital (is the switch on or off?) and analog (what is the voltage at a pin?). This enables us to connect many different types of sensors for light, temperature, sound, etc.

Outputs can also be analog or digital. So, you can set a pin to be on or off (0V or 5V) and this can turn LEDs on and off directly, or you can use the output to control higher-power devices such as motors. They can also provide an analog output voltage. That is, you can set the output of a pin to some particular voltage, allowing you to control the speed of a motor or the brightness of a light, for example, rather than simply turning it on or off.

What's on an Arduino Board?

Figure 2-1 shows our Arduino board—or in this case an Arduino clone. Let us have a quick tour of the various components on the board.

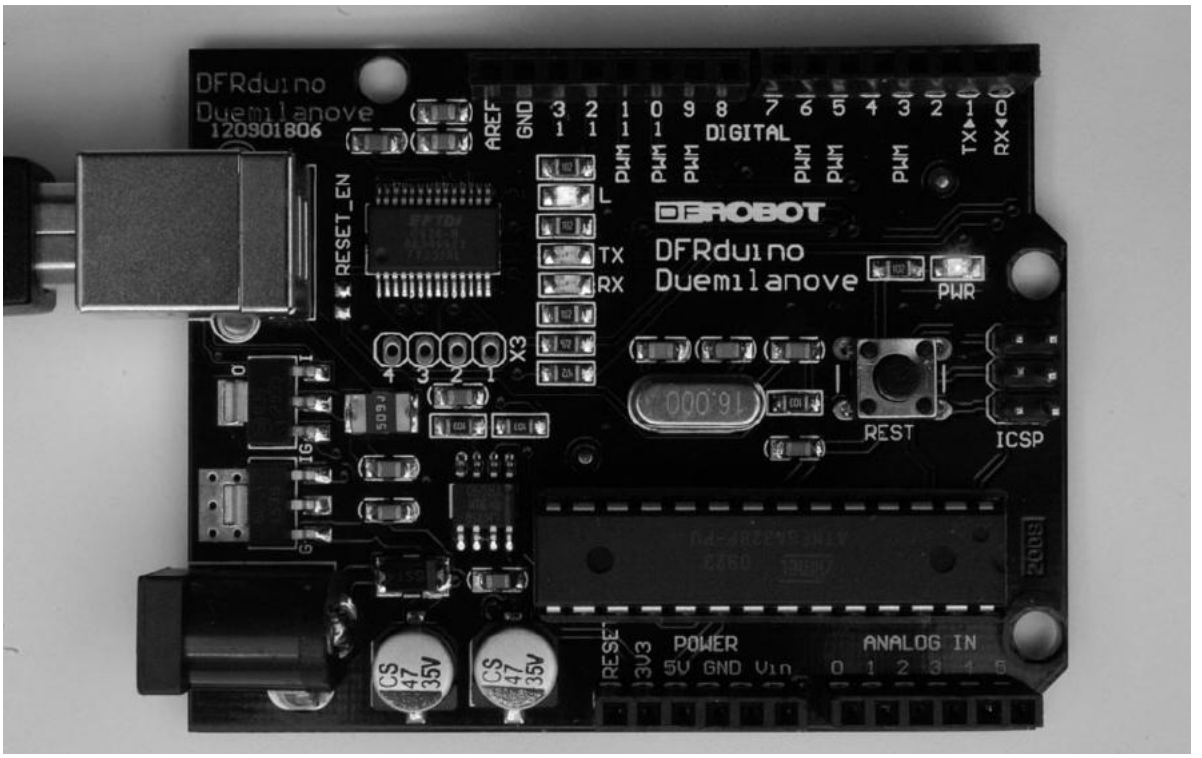


Figure 2-1 The components of an Arduino board.

Power Supply

Directly below the USB connector is the 5V voltage regulator. This regulates whatever voltage (between 7 and 12 volts) is supplied from the power socket into a constant 5V.

5V (along with 3V, 6V, 9V, and 12V) is a bit of a standard voltage in electronics. 3, 6, and 9V are standard because the voltage that you get from a single alkaline cell is 1.5V, and these are all convenient multiples of 1.5V, which is what you get when you make a "battery" of two, three, six, or eight cells.

So if that is the case, you might be wondering why 5V? You cannot make that using 1.5V cells. Well, the answer lies in the fact that in the early days of computing, a range of chips became available, each of which contained logic gates. These chips used something called TTL (Transistor-Transistor Logic), which was a bit fussy about its voltage requirements and required something between 4.5V and 5.5V. So 5V became the standard voltage for all digital electronics.

These days, the type of logic gates used in chips has changed and they are far more tolerant of different voltages.

The 5V voltage regulator chip is actually quite big for a surface-mount component. This is so that it can dissipate the heat required to regulate the voltage at a reasonably high current, which is useful when driving our external electronics.

Power Connections

Next, let us look at the connectors at the bottom of Figure 2-1. You can read the connection names next to the connectors.

The first is Reset. This does the same thing as pressing the Reset button on the Arduino. Rather like rebooting a PC, it resets the microcontroller, beginning its program from the start. The Reset connector allows you to reset the microcontroller

by momentarily setting this pin high (connecting it to +5V).

The rest of the pins in this section provide different voltages (3.3, 5, GND, and 9), as labeled. GND, or ground, just means zero volts. It is the reference voltage to which all other voltages on the board are relative.

At this point, it would be useful to remind the reader about the difference between voltage and current. There is no perfect analogy for the behavior of electrons in a wire, but the author finds an analogy with water in pipes to be helpful, particularly in dealing with voltage, current, and resistance. The relationship between these three things is called Ohm's Law.

Figure 2-2 summarizes the relationship between voltage, current, and resistance. The left side of the diagram shows a circuit of pipes, where the top of the diagram is higher up (in elevation) than the bottom of the diagram. So water will naturally flow from the top of the diagram to the bottom. Two factors determine how much water passes any point in the circuit in a given time (the current):

- The height of the water (or if you prefer, the pressure generated by the pump). This is like voltage in electronics.
- The resistance to flow offered by the constriction in the pipework

The more powerful the pump, the higher the water can be pumped and the greater the current that will flow through the system. On the other hand, the greater the resistance offered by the pipework, the lower the current.

In the right half of Figure 2-2, we can see the electronic equivalent of our pipework. In this case, current is actually a measure of how many electrons flow past a point per second. And yes, resistance is the resistance to the flow of electrons.

Instead of height or pressure, we have a concept of voltage. The bottom of the diagram is at 0V, or ground, and we have shown the top of the diagram as being at 5V. So the current that flows (I) will be the voltage difference (5) divided by the resistance R.

Ohm's Law is usually written as V = IR. Normally, we know what V is and are trying to

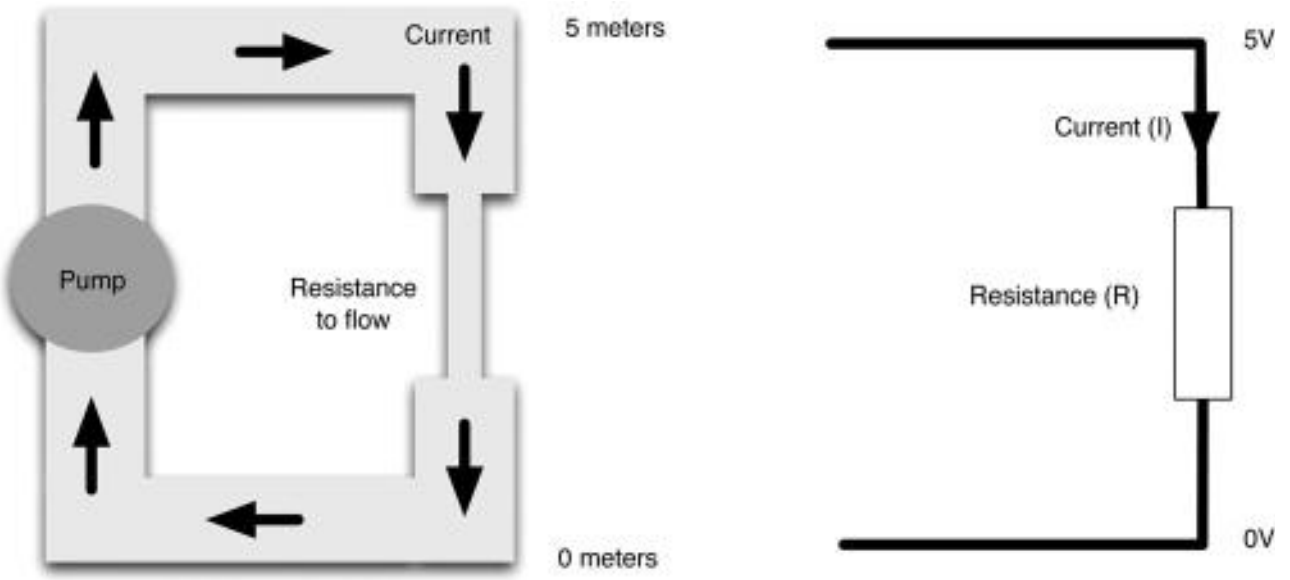


Figure 2-2 Ohm's Law.

calculate R or I, so we can do a bit of rearranging to have the more convenient I = V/R and R = V/I.

It is very important to do a few calculations using Ohm's Law when connecting things to your Arduino, or you may damage it if you ask it to supply too much current. Generally, though, the Arduino boards are remarkably tolerant of accidental abuse.

So, going back to our Arduino power pins, we can see that the Arduino board will supply us with useful voltages of 3.3V, 5V, and 9V. We can use any of those supplies to cause a current to flow, as long as we are careful not to make it a short circuit (no resistance to flow), which would cause a potentially large current to flow that could cause damage. In other words, we have to make sure that anything we connect to the supply has enough resistance to prevent too much current from flowing. As well as supplying a particular voltage, each of those supply connections will have a maximum current that can be allowed to flow. Those currents are 50 mA (thousandths of an amp) for the 3.3V supply, and although it is not stated in the Arduino specification, probably around 300 mA for the 5V.

Analog Inputs

The next section of connections is labeled Analog In 0 to 5. These six pins can be used to measure the voltage connected to them so that the value can be used in a sketch. Note that they measure a voltage and not a current. Only a tiny current will ever flow into them and down to ground because they have a very large internal resistance.

Although labeled as analog inputs, these connections can also be used as digital inputs or outputs, but by default, they are analog inputs.

Digital Connections

We now switch to the top connector and start on the right side (Figure 2-1). We have pins labeled Digital 0 to 13. These can be used as either inputs or outputs. When using them as outputs, they behave rather like the supply voltages we talked about earlier, except that these are all 5V and can be turned on or off from our sketch. So, if we turn them on from our sketch, they will be at 5V and if we turn them off, they will be at 0V. As with the supply connectors, we have to be careful not to exceed their maximum current capabilities.

These connections can supply 40 mA at 5V. That is more than enough to light a standard LED, but not enough to drive an electric motor directly.

As an example, let us look at how we would connect an LED to one of these digital connections. In fact, let's go back to Project 1 in Chapter 1.

As a reminder, Figure 2-3 shows the schematic diagram for driving the LED that we first used in the previous chapter. If we were to not use a resistor with our LED but simply connect the LED between pin 12 and GND, then when we turned digital output 12 on (5V), we might burn out the LED, destroying it.

This is because LEDs have a very low resistance and will cause a very high current to flow unless they are protected from themselves by using a resistor to limit the flow of current.

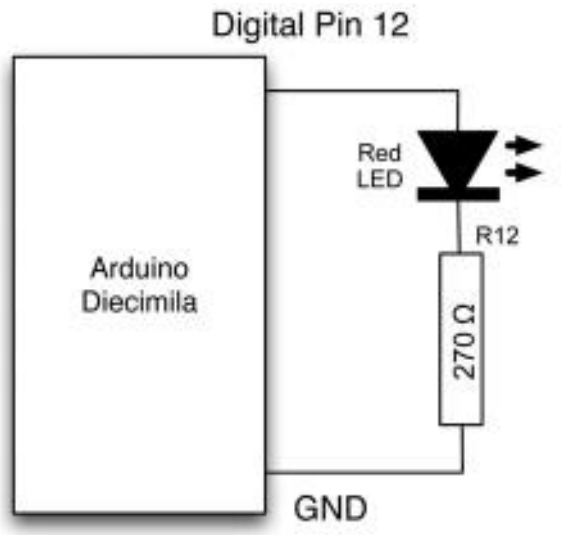


Figure 2-3 LED and series resistor.

An LED needs about 10 mA to shine reasonably brightly. The Arduino can supply 50 mA, so there is no problem there; we just need to choose a sensible value of resistor.

LEDs have the interesting property that no matter how much current flows through them, there will always be about 2V between their pins. We can use this fact and Ohm's Law to work out the right value of resistor to use.

We know that (at least when it's on) the output pin will be supplying 5V. Now, we have just said that 2V will be "dropped" by our LED, leaving 3V (5 – 2) across our current-limiting resistor. We want the current flowing around the circuit to be 10 mA, so we can see that the value for the resistor should be

$R = V/I$

$R = 3V/10\ mA$

$R = 3V/0.01\ A$

$R = 300\ \Omega$

Resistors come in standard values, and the closest value to 300 Ω is 270 Ω. This means that instead of 10 mA, the current will actually be

$I = V/R$

$I = 3/270$

$I = 11.111\ mA$

These things are not critical, and the LED would probably be equally happy with anything between 5 and 30 mA, so 270 Ω will work just fine.

We can also set one of these digital connections to be an input, in which case, it works rather like an analog input, except that it will just tell us if the voltage at a pin is above a certain threshold (roughly 2.5V) or not.

Some of the digital connections (3, 5, 6, 9, 10, and 11) have the letters PWM next to them. These can be used to provide a variable output voltage rather than a simple 5V or nothing.

On the left side of the top connector in Figure 2-1, there is another GND connection and a connection called AREF. AREF can be used to scale the readings for analog inputs. This is rarely used and can safely be ignored.

Microcontroller

Getting back to our tour of the Arduino board, the microcontroller chip itself is the black rectangular device with 28 pins. This is fitted into a DIL (dual in-line) socket so that it can be easily replaced. The 28-pin microcontroller chip used on Arduino Duemilanove is the ATmega328. Figure 2-4 is a block diagram showing the main features of this device.

The heart, or perhaps more appropriately the brain, of the device is the CPU (central processing unit). It controls everything that goes on within the device. It fetches program instructions stored in the Flash memory and executes them. This might involve fetching data from working memory (RAM), changing it, and then putting it back. Or, it may mean changing one of the digital outputs from 0 to 5 volts.

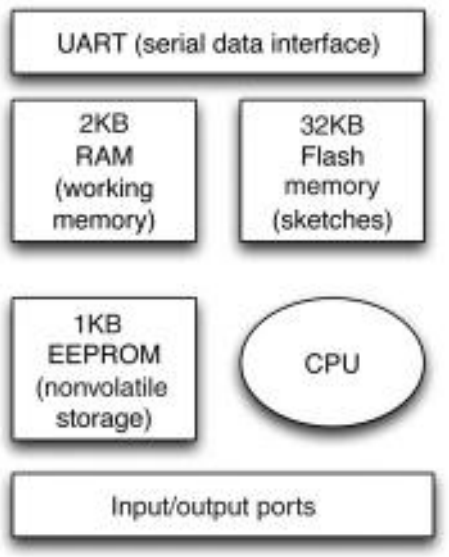


Figure 2-4 ATmega328 block diagram.

The electrically erasable programmable read-only memory (EEPROM) memory is a little like the Flash memory in that it is nonvolatile. That is, you can turn the device off and on and it will not have forgotten what is in the EEPROM. Whereas the Flash memory is intended for storing program instructions (from sketches), the EEPROM is used to store data that you do not want to lose in the event of a reset or power failure.

The older Diecimila uses the ATmega168, which functions in an identical way to the ATmega328 except that it has half the amount of every sort of memory. It has 16KB of Flash memory, 1KB of RAM, and 512 bytes of EEPROM.

Other Components

Above the microcontroller there is a small, silver, rectangular component. This is a quartz crystal oscillator. It "ticks" 16 million times a second, and on each of those ticks, the microcontroller can perform one operation—an addition, subtraction, etc.

To the right of the crystal, is the Reset switch. Clicking this sends a logic pulse to the Reset pin of the microcontroller, causing the microcontroller to start its program afresh and clear its memory. Note that any program stored on the device will be retained because this is kept in nonvolatile Flash memory—that is, memory that remembers even when the device is not powered.

To the right of the Reset button is the serial programming connector. It offers another means of programming the Arduino without using the USB port. Since we do have a USB connection and software that makes it convenient to use, we will not avail ourselves of this feature.

In the top left of the board next to the USB socket is the USB interface chip. This converts the signal levels used by the USB standard to levels that can be used directly by the Arduino board.

The Arduino Family

It's useful to have a little background on the Arduino boards. We will be using the Duemilanove for most of our projects; however, we will also dabble with the interesting Lilypad Arduino.

The Lilypad (Figure 2-5), is a tiny, thin Arduino board that can be stitched into clothing for applications that have become known as wearable computing. It does not have a USB connection, and you must use a separate adaptor to program it. This is an exceptionally beautiful design. Inspired by its clocklike appearance, we will use this in Project 29 (Unfathomable Binary Clock).

At the other end of the spectrum is the Arduino Mega. This board has a faster processor with more memory and a greater number of input/output pins.

Cleverly, the Arduino Mega can still use shields built for the smaller Arduino Diecimila and Duemilanove boards, which sit at the front of the board, allowing access to the double row of connectors for the Mega's additional connections at the rear. Only the most demanding of projects really need an Arduino Mega.

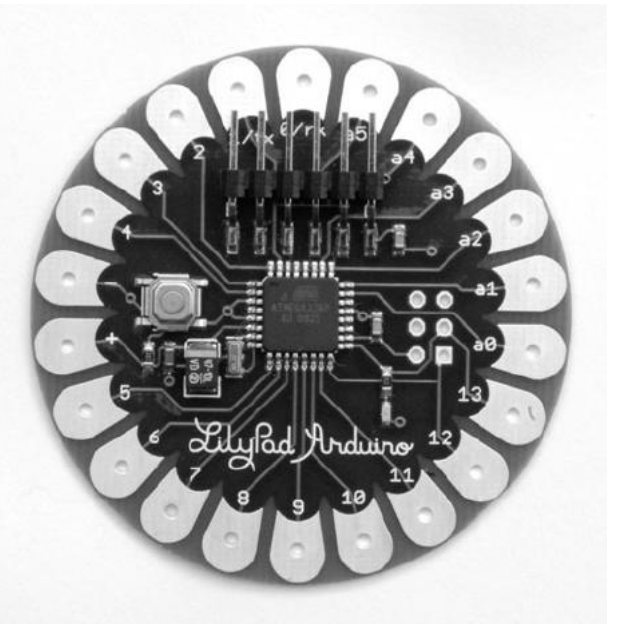


Figure 2-5 Arduino Lilypad.

The C Language

Many languages are used to program microcontrollers, from hard-core Assembly language to graphical programming languages like Flowcode. Arduino sits somewhere in between these two extremes and uses the C programming language. It does, however, wrap up the C language, hiding away some of the complexity. This makes it easy to get started.

The C language is, in computing terms, an old and venerable language. It is well suited to programming the microcontroller because it was invented at a time when compared to today's monsters, the typical computer was quite poorly endowed.

C is an easy language to learn, yet compiles into efficient machine code that only takes a small amount of room in our limited Arduino memory.

An Example

We are now going to examine the sketch for Project 1 in a bit more detail. The listing for this sketch to flash an LED on and off is shown here. We have ignored all the lines that begin with // or blocks of lines that start with /* and end with */ because these are comment lines that have no effect on the program and are just there for information.

```
int ledPin =  13;
   // LED connected to digital pin 13
void setup()
{
  pinMode(ledPin, OUTPUT);
}

void loop()
{
  digitalWrite(ledPin, HIGH);
   // set the LED on
  delay(1000);
   // wait for a second
  digitalWrite(ledPin, LOW);
   // set the LED off
  delay(1000);
   // wait for a second
}
```

It is standard practice to include such text at the top of any program file. You can also include comments that describe a tricky bit of code, or anything that requires some explanation.

The Arduino development environment uses something called a compiler that converts the script into the machine code that will run on the microcontroller.

So, moving onto the first real line of code, we have:

```
int ledPin = 13;
```

This line of code gives a name to the digital output pin that we are going to connect to the LED. If you look carefully at your Arduino board, you will see the connector for pin 13 between GND and pin 12 on the Arduino's top connector. The Arduino board has a small green LED already soldered onto the board and connected to pin 13. We are going to change the voltage of this pin to between 0V and 5V to make the LED flash.

We are going to use a name for the pin so that it's easy to change it and use a different one. You can see that we refer to "ledPin" later in the sketch. You may prefer to use pin 12 and the external LED that you used with your breadboard in Chapter 1. But for now, we will assume that you are using the built-in LED attached to pin 13.

You will notice that we did not just write:

```
led pin = 13
```

That is because compilers are kind of fussy and precise about how we write our programs. Any name we use in a program cannot use spaces, so it is a convention to use what is called "bumpy case."

So, we start each word (apart from the first) with an uppercase letter and remove the space; that gives us:

```
ledPin = 13
```

The word ledPin is what is termed a variable. When you want to use a variable for the first time in a sketch, you have to tell the compiler what type of variable it is. It may be an int, as is the case here, or a float, or a number of other types that we will describe later in this chapter.

An int is an integer—that is, a whole number—which is just what we need when referring to a particular pin on the Arduino. There is, after all, no pin 12.5, so it would not be appropriate to use a floating point number (float).

The syntax for a variable declaration is

```
type variableName = value;
```

So first we have the type (int), then a space, then a variable name in bumpy case (ledPin), then an equal sign, then a value, and finally a semicolon to indicate the end of the line:

```
int ledPin = 13;
```

As I mentioned, the compiler is fussy, so if you forget the semicolon, you will receive an error message when you compile the sketch. Try removing the semicolon and clicking the Play button. You should see a message like this:

```
error: expected unqualified-id before
   numeric constant
```

It's not exactly "you forgot a semicolon," and it is not uncommon for error messages to be similarly misleading.

The next lines of the sketch are

```
void setup()
   // run once, when the sketch starts
{
  pinMode(ledPin, OUTPUT);
   // sets the digital pin as output
}
```

This is what is called a function, and in this case, the function is called setup. Every sketch must contain a setup function, and the lines of code inside the function surrounded by curly brackets will be carried out in the order that they are written. In this case, that is just the line starting with pinMode.

A good starting point for any new project is to copy this example project and then alter it to your needs.

We will not worry too much about functions at this stage, other than to say that the setup function will be run every time the Arduino is reset, including when the power is first turned on. It will also be run every time a new sketch is downloaded.

In this case, the only line of code in setup is

```
pinMode(ledPin, OUTPUT);
   // sets the digital pin as output
```

The first thing to mention is that we have a different type of comment on the end of this line. That is, the single-line comment. This begins with a // and ends at the end of the line.

The line can be thought of as a command to the Arduino to use the ledPin as a digital output. If we had a switch connected to ledPin, we could set it as an input using:

```
pinMode(ledPin, INPUT);
```

However, we would call the variable something more appropriate, like switchPin.

The words INPUT and OUTPUT are what are called constants. They will actually be defined within C to be a number. INPUT may be defined as 0 and OUPUT as 1, but you never need to actually see what number is used, as you always refer to them as INPUT or OUTPUT. Later in this chapter, we will see two more constants, HIGH and LOW, that are used when setting the output of a digital pin to +5V or 0V, respectively.

The next section of code is another function that every Arduino sketch must have; it is called loop:

```
void loop()
{
  digitalWrite(ledPin, HIGH);
   // sets the LED on
  delay(1000);
   // waits for a second
  digitalWrite(ledPin, LOW);
   // sets the LED off
  delay(1000);
   // waits for a second
}
```

The function loop will be run continuously until the Arduino is powered down. That is, as soon as it finishes executing the commands it contains, it will begin again. Remember that an Arduino board is capable of running 16 million commands per second, so things inside the loop will happen frequently if you let them.

In this case, what we want the Arduino to keep doing continuously is to turn the LED on, wait a second, turn the LED off, and then wait another second. When it has finished doing this, it will begin again, turning the LED on. In this way it will go round the loop forever.

By now, the command syntax for digitalWrite and delay will be becoming more familiar. Although we can think of them as commands that are sent to the Arduino board, they are actually functions just like setup and loop, but in this case they have what are called parameters. In the case of digitalWrite, it is said to take two parameters: the Arduino pin to write to and the value to write.

In our example, we pass the parameters of ledPin and HIGH to turn the LED on and then ledPin and LOW to turn it off again.

Variables and Data Types

We have already met the variable ledPin and declared it to be of type int. Most of the variables that you use in your sketches are also likely to be ints. An int holds an integer number between –32,768 and +32,767. This uses just two bytes of data for each number stored from the 1024 available bytes of storage on an Arduino. If that range is not enough, you can use a long, which uses four bytes for each number and will give you a range of numbers from –2,147,483,648 to +2,147,483,647.

Most of the time, an int represents a good compromise between precision and use of memory.

If you are new to programming, I would use ints for almost everything and gradually expand your repertoire of data types as your experience grows.

Other data types available to you are summarized in Table 2-1.

One thing to consider is that if data types exceed their range, strange things happen. So if you have a byte variable with 255 in it and you add 1 to it, you get 0. More alarmingly, if you have an int variable with 32,767 and you add 1 to it, you will end up with –32,768.

Until you are completely happy with these different data types, I would recommend sticking to int, as it works for practically everything.

Arithmetic

It is fairly uncommon to need to do much in the way of arithmetic in a sketch. Occasionally, you will need to do a bit of scaling of, say, an analog

TABLE 2-1 Data Types in C

Type	Memory (bytes)	Range	Notes
boolean	1	true or false (0 or 1)	
char	1	–128 to +128	Used to represent an ASCII character code (e.g., A is represented as 65). Its negative numbers are not normally used.
byte	1	0 to 255	
int	2	–32,768 to +32,767	
unsigned int	2	0 to 65,536	Can be used for extra precision where negative numbers are not needed. Use with caution, as arithmetic with ints may cause unexpected results.
long	4	–2,147,483,648 to 2,147,483,647	Needed only for representing very large numbers.
unsigned long	4	0 to 4,294,967,295	See unsigned int.
float	4	–3.4028235E+38 to + 3.4028235E+38	
double	4	as float	Normally, this would be eight bytes and higher precision than float with a greater range. However, on Arduino, it is the same as float.

input to turn it into a temperature, or more typically, add 1 to a counter variable.

When you are performing some calculation, you need to be able to assign the result of the calculation to a variable.

The following lines of code contain two assignments. The first gives the variable y the value 50 and the second gives the variable x the value of y + 100.

```
y = 50;
x = y + 100;
```

Strings

When programmers talk of Strings, they are referring to a string of characters such as the much-used message "Hello World." In the world of Arduino, there are a couple of situations where you might want to use Strings: when writing messages to an LCD display or sending back serial text data over the USB connection.

Strings are created using the following syntax:

```
char* message = "Hello World";
```

The char* word indicates that the variable message is a pointer to a character. For now, we do not need to worry too much about how this works. We will meet this later in the book when we look at interfacing with textual LCD displays.

Conditional Statements

Conditional statements are a means of making decisions in a sketch. For instance, your sketch may turn the LED on if the value of a temperature variable falls below a certain threshold.

The code for this is shown here:

```
if (temperature < 15)
{
  digitalWrite(ledPort, HIGH);
}
```

The line or lines of code inside the curly braces will only be executed if the condition after the if keyword is true.

The condition has to be contained in parentheses, and is what programmers call a logical expression. A logical expression is like a mathematical sentence that must always return one of two possible values: true or false.

The following expression will return true if the value in the temperature variable is less than 15:

```
(temperature < 15)
```

As well as <, you have: >, <=, and >=.

To see if two numbers are equal, you can use == and to test if they are not equal, you can use !=.

So the following expression would return true if the temperature variable had a value that was anything except 15:

```
(temperature != 15)
```

You can also make complex conditions using what are called logical operators. The principal operators being && (and) and || (or).

So an example that turned the LED on if the temperature was less than 15 or greater than 20 might look like this:

```
if ((temperature < 15) || (temperature
   > 20))
{
  digitalWrite(ledPort, HIGH);
}
```

Often, when using an if statement, you want to do one thing if the condition is true and a different thing if it is false. You can do this by using the else keyword, as shown in the following example. Note the use of nested parentheses to make it clear what is being or'd with what.

```
if ((temperature < 15) || (temperature
   > 20))
{
  digitalWrite(ledPort, HIGH);
}
else
{
  digitalWrite(ledPort, LOW);
}
```

Summary

In this chapter, we have explored the hardware provided by the Arduino and refreshed our knowledge of a little elementary electronics.

We have also started our exploration of the C programming language. Don't worry if you found some of this hard to follow. There is a lot to take in if you are not familiar with electronics, and while the author's goal is to explain how everything works, you are completely at liberty to simply start on the projects first and come back to the theory when you are ready.

In the next chapter we will get to grips with programming our Arduino board and embark on some more serious projects.

This page intentionally left blank

CHAPTER 3

LED Projects

In this chapter, we are going to start building some LED-based projects. We will keep the hardware fairly simple so that we can concentrate on the programming of the Arduino.

Programming microcontrollers can be a tricky business requiring an intimate knowledge of the inner workings of the device: fuses, registers, etc. This is, in part, because modern microcontrollers are almost infinitely configurable. Arduino standardizes its hardware configuration, which, in return for a small loss of flexibility, makes the devices a great deal easier to program.

Project 2
Morse Code S.O.S. Flasher

Morse code used to be a vital method of communication in the 19th and 20th centuries. Its coding of letters as a series of long and short dots meant that it could be sent over telegraph wires, over a radio link, and using signaling lights. The letters S.O.S. (Save Our Souls) is still recognized as an international signal of distress.

In this project, we will make our LED flash the sequence S.O.S. over and over again.

For this project, you will need just the same components as for Project 1.

COMPONENTS AND EQUIPMENT

	Description	Appendix
	Arduino Diecimila or Duemilanove board or clone	1
D1	5-mm red LED	23
R1	270 Ω 0.5W metal film resistor	6

- Almost any commonly available LED and 270 Ω resistor will be fine.
- No tools other than a pair of pliers or wire cutters are required.

Hardware

The hardware is exactly the same as Project 1. So, you can either just plug the resistor and LED directly into the Arduino connectors or use a breadboard (see Chapter 1).

Software

Rather than start typing this project in from scratch, we will use Project 1 as a starting point. So if you have not already done so, please complete Project 1.

If you have not already done so, download the project code from www.arduinoevilgenius.com; then you can also just load the completed sketch for Project 1 from your Arduino Sketchbook and download it to the board (see Chapter 1). However,

it will help you understand Arduino better if you modify the sketch from Project 1 as suggested next.

Modify the loop function of Project 1 so that it now appears as shown here. Note that copy and paste is highly recommended in this kind of situation:

```
void loop()
{
  digitalWrite(ledPin, HIGH);
   // S (...) first dot
  delay(200);
  digitalWrite(ledPin, LOW);
  delay(200);
  digitalWrite(ledPin, HIGH);
   // second dot
  delay(200);
  digitalWrite(ledPin, LOW);
  delay(200);
  digitalWrite(ledPin, HIGH);
   // third dot
  delay(200);
  digitalWrite(ledPin, LOW);
  delay(500);
  digitalWrite(ledPin, HIGH);
   // O (—-) first dash
  delay(500);
  digitalWrite(ledPin, LOW);
  delay(500);
  digitalWrite(ledPin, HIGH);
   // second dash
  delay(500);
  digitalWrite(ledPin, LOW);
  delay(500);
  digitalWrite(ledPin, HIGH);
   // third dash
  delay(500);
  digitalWrite(ledPin, LOW);
  delay(500);
  digitalWrite(ledPin, HIGH);
   // S (...) first dot
  delay(200);
  digitalWrite(ledPin, LOW);
  delay(200);
  digitalWrite(ledPin, HIGH);
   // second dot
  delay(200);
  digitalWrite(ledPin, LOW);
  delay(200);
  digitalWrite(ledPin, HIGH);
   // third dot
  delay(200);
  digitalWrite(ledPin, LOW);
  delay(1000);
   // wait 1 second before we start
      again

}
```

This would all work, and feel free to try it; however, we are not going to leave it there. We are going to alter our sketch to improve it, and at the same time make it a lot shorter.

We can reduce the size of the sketch by creating our own function to replace the four lines of code involved in any flash with one line.

After the loop function's final curly brace, add the following code:

```
void flash(int duration)
{
  digitalWrite(ledPin, HIGH);
  delay(duration);
  digitalWrite(ledPin, LOW);
  delay(duration);
}
```

Now modify the loop function so that it looks like this:

```
void loop()
{
  flash(200); flash(200); flash(200);
   // S
  delay(300);
   // otherwise the flashes run
   together
  flash(500); flash(500); flash(500);
   // O
  flash(200); flash(200); flash(200);
   // S
  delay(1000);
   // wait 1 second before we start
   again
}
```

LISTING PROJECT 2

```
int ledPin = 13;

void setup()                          // run once, when the sketch starts
{
  pinMode(ledPin, OUTPUT);            // sets the digital pin as output
}

void loop()
{
  flash(200); flash(200); flash(200);  // S
  delay(300);                          // otherwise the flashes run together
  flash(500); flash(500); flash(500);  // O
  flash(200); flash(200); flash(200);  // S
  delay(1000);                         // wait 1 second before we start again
}

void flash(int duration)
{
  digitalWrite(ledPin, HIGH);
  delay(duration);
  digitalWrite(ledPin, LOW);
  delay(duration);
}
```

The whole final listing is shown in Listing Project 2.

This makes the sketch a lot smaller and a lot easier to read.

Putting It All Together

That concludes Project 2. We will now cover some more background on programming the Arduino before we go on to look at Project 3, where we will use our same hardware to write a Morse code translator, where we can type sentences on our computer and have them flashed as Morse code. In Project 4, we will improve the brightness of our flashing by replacing our red LED with a high-power Luxeon-type LED.

But first, we need a little more theory in order to understand Projects 3 and 4.

Loops

Loops allow us to repeat a group of commands a certain number of times or until some condition is met.

In Project 2, we only want to flash three dots for an S, so it is no great hardship to repeat the flash command three times. However, it would be far less convenient if we needed to flash the LED 100 or 1000 times. In that case we can use the for language command in C.

```
for (int i = 0; i < 100; i ++)
{
  flash(200);
}
```

The for loop is a bit like a function that takes three arguments, although here, those arguments are separated by semicolons rather than the usual

commas. This is just a quirk of the C language. The compiler will soon tell you when you get it wrong.

The first thing in the parentheses after "for" is a variable declaration. This specifies a variable to be used as a counter variable and gives it an initial value—in this case, 0.

The second part is a condition that must be true for us to stay in the loop. In this case, we will stay in the loop as long as "i" is less than 100, but as soon as "i" is 100 or more, we will stop doing the things inside the loop.

The final part is what to do every time you have done all the things in the loop. In this case, that is increment "i" by 1 so that it can, after 100 trips around the loop, cease to be less than 100 and cause the loop to exit.

Another way of looping in C is to use the while command. The same example shown previously could be accomplished using a while command, as shown here:

```
int i = 0;
while (i < 100)
{
  flash(200);
  i ++;
}
```

The expression in parentheses after while must be true to stay in the loop. When it is no longer true, the sketch will continue running the commands after the final curly brace.

The curly braces are used to bracket together a group of commands. In programming parlance, they are known as a block.

Arrays

Arrays are a way of containing a list of values. The variables we have met so far have only contained a single value, usually an int. By contrast, an array contains a list of values, and you can access any one of those values by its position in the list.

C, in common with the majority of programming languages, begins its index positions at 0 rather than 1. This means that the first element is actually element zero.

To illustrate the use of arrays, we could change our Morse code example to use an array of flash durations. We can then use a for loop to step through each of the items in the array.

First let's create an array of ints containing the durations:

```
int durations[] = {200, 200, 200, 500,
   500, 500, 200, 200, 200}
```

You indicate that a variable contains an array by placing [] after the variable name. If you are setting the contents of the array at the same time you are defining it, as in the previous example, you do not need to specify the size of the array. If you are not setting its initial contents, then you need to specify the size of the array inside the square brackets. For example:

```
int durations[10];
```

Now we can modify our loop method to use the array:

```
void loop()
   // run over and over again
{
  for (int i = 0; i < 9; i++)
  {
    flash(durations[i]);
  }
  delay(1000);
   // wait 1 second before we start
   // again
}
```

An obvious advantage of this approach is that it is easy to change the message by simply altering the durations array. In Project 3, we will take the use of arrays a stage further to make a more general-purpose Morse code flasher.

Project 3
Morse Code Translator

In this project, we are going to use the same hardware as for Projects 1 and 2, but we are going to write a new sketch that will let us type in a sentence on our computer and have our Arduino board convert that into the appropriate Morse code dots and dashes.

Figure 3-1 shows the Morse code translator in action. The contents of the message box are being flashed as dots and dashes on the LED.

To do this, we will make use of what we have learned about arrays and strings, and also learn something about sending messages from our computer to the Arduino board through the USB cable.

For this project, you will need just the same components as for Project 1 and 2. In fact, the hardware is exactly the same; we are just going to modify the sketch of Project 1.

COMPONENTS AND EQUIPMENT

	Description	Appendix A
	Arduino Diecimila or Duemilanove board or clone	1
D1	5-mm Red LED	23
R1	270 Ω 0.5W metal film resistor	6

Hardware

Please refer back to Project 1 for the hardware construction for this project.

You can either just plug the resistor and LED directly into the Arduino connectors, or use the

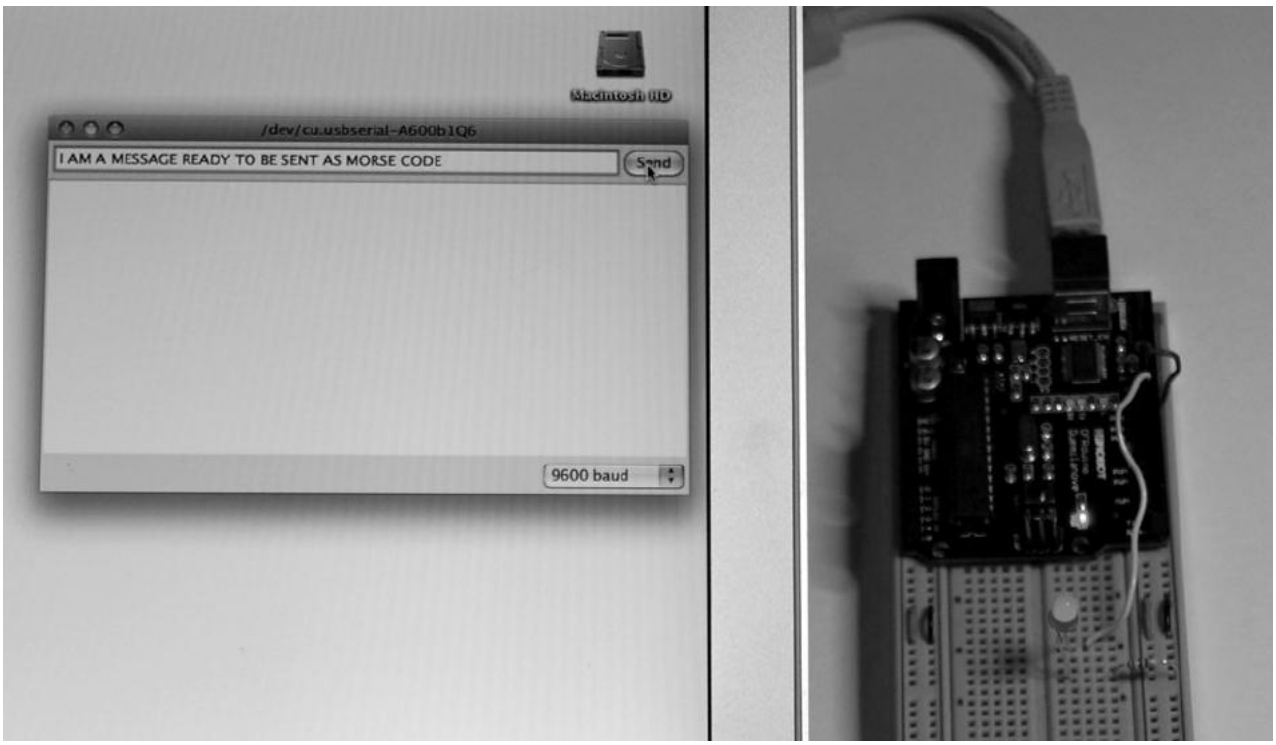


Figure 3-1 Morse code translator.

breadboard (see Chapter 1). You can even just change the ledPin variable in the sketch to be pin 13 so that you use the built-in LED and do not need any external components at all.

Software

The letters in Morse code are shown in Table 3-1.

Some of the rules of Morse code are that a dash is three times as long as a dot, the time between each dash or dot is equal to the duration of a dot, the space between two letters is the same length as a dash, and the space between two words is the same duration as seven dots.

For the sake of this project, we will not worry about punctuation, although it would be an interesting exercise for you to try adding this to the sketch. For a full list of all the Morse characters, see http://en.wikipedia.org/wiki/Morse_code.

TABLE 3-1 Morse Code Letters

A	.-	N	-.	0	——-
B	-...	O	—-	1	.——
C	-.-.	P	.--.	2	..---
D	-..	Q	--.-	3	...--
E	.	R	.-.	4	....-
F	..-.	S	...	5	.....
G	--.	T	-	6	-....
H	....	U	..-	7	--...
I	..	V	...-	8	---..
J	.---	W	.--	9	----.
K	-.-	X	-..-		
L	.-..	Y	-.--		
M	--	Z	--..		

The sketch for this is shown in Listing Project 3. An explanation of how it all works follows.

LISTING PROJECT 3

```
int ledPin = 12;

char* letters[] = {
  ".-", "-...", "-.-.", "-..", ".", "..-.", "--.", "....", "..",     // A-I
  ".---", "-.-", ".-..", "--", "-.", "---", ".--.", "--.-", ".-.",  // J-R
  "...", "-", "..-", "...-", ".--", "-..-", "-.--", "--.."          // S-Z
};

char* numbers[] = {"-----", ".----", "..---", "...--", "....-", ".....", "-....",
   "--...", "---..", "----."};

int dotDelay = 200;

void setup()
{
  pinMode(ledPin, OUTPUT);
  Serial.begin(9600);
}

void loop()
{
  char ch;
  if (Serial.available())      // is there anything to be read from USB?
```

LISTING PROJECT 3 (continued)

```
  {
    ch = Serial.read();               // read a single letter
    if (ch >= 'a' && ch <= 'z')
    {
      flashSequence(letters[ch - 'a']);
    }
    else if (ch >= 'A' && ch <= 'Z')
    {
      flashSequence(letters[ch - 'A']);
    }
    else if (ch >= '0' && ch <= '9')
    {
      flashSequence(numbers[ch - '0']);
    }
    else if (ch == ' ')
    {
     delay(dotDelay * 4);             // gap between words
    }
  }
}

void flashSequence(char* sequence)
{
   int i = 0;
   while (sequence[i] != NULL)
   {
       flashDotOrDash(sequence[i]);
       i++;
   }
   delay(dotDelay * 3);               // gap between letters
}

void flashDotOrDash(char dotOrDash)
{
  digitalWrite(ledPin, HIGH);
  if (dotOrDash == '.')
  {
    delay(dotDelay);
  }
  else // must be a -
  {
    delay(dotDelay * 3);
  }
  digitalWrite(ledPin, LOW);
  delay(dotDelay);                    // gap between flashes
}
```

We keep track of our dots and dashes using arrays of strings. We have two of these, one for letters and one for numerals. So to find out what we need to flash for the first letter of the alphabet (A), we will get the string letters[0]—remember, the first element of an array is element 0, not element 1.

The variable dotDelay is defined, so if we want to make our Morse code flash faster or slower, we can change this value, as all the durations are defined as multiples of the time for a dot.

The setup function is much the same as for our earlier projects; however, this time we are getting communications from the USB port, so we must add the command:

```
Serial.begin(9600);
```

This tells the Arduino board to set the communications speed through USB to be 9600 baud. This is not very fast, but fast enough for our Morse code messages. It is also a good speed to set it to because that is the default speed used by the Arduino software on your computer.

In the loop function, we are going to repeatedly see if we have been sent any letters over the USB connection and if we have to process the letter. The Arduino function Serial.available() will be true if there is a character to be turned into Morse code and the Serial.read() function will give us that character, which we assign to a variable called "ch" that we defined just inside the loop.

We then have a series of if statements that determine whether the character is an uppercase letter, a lowercase letter, or a space character separating two words. Looking at the first if statement, we are testing to see if the character's value is greater than or equal to "a" and less than or equal to "z." If that is the case, we can find the sequence of dashes and dots to flash using the letter array that we defined at the top of the sketch. We determine which sequence from the array to use by subtracting "a" from the character in ch. At first sight, it might look strange to be subtracting one letter from another, but it is perfectly acceptable to do this in C. So, for example, "a" - "a" is 0, whereas "d" - "a" will give us the answer 3. So, if the letter that we read from the USB connections was f, we will calculate "f" - "a," which gives us 5 as the position of the letters array. Looking up letters[5] will give us the string "..-.". We pass this string to a function called flashSequence.

The flashSequence function is going to loop over each of the parts of the sequence and flash it as either a dash or a dot. Strings in C all have a special code on the end of them that marks the end of the string, and this is called NULL. So, the first thing flashSequence does is to define a variable called "i." This is going to indicate the current position in the string of dots and dashes, starting at position 0. The while loop will keep going until we reach the NULL on the end of the string.

Inside the while loop, we first flash the current dot or dash using a function that we are going to discuss in a moment and then add 1 to "i" and go back round the loop flashing each dot or dash in turn until we reach the end of the string.

The final function that we have defined is flashDotOrDash'; this just turns the LED on and then uses an if statement to either delay for the duration of a single dot if the character is a dot, or for three times that duration if the character is a dash, before it turns the LED off again.

Putting It All Together

Load the completed sketch for Project 3 from your Arduino Sketchbook and download it onto your board (see Chapter 1).

To use the Morse code translator, we need to use a part of the Arduino software called the Serial Monitor. This window allows you to type messages that are sent to the Arduino board as well as see any messages that the Arduino board chooses to reply with.

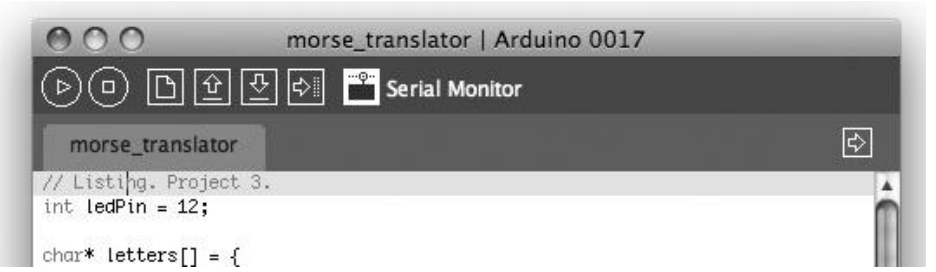


Figure 3-2 Launching the Serial Monitor.

The Serial Monitor is launched by clicking the rightmost icon shown highlighted in Figure 3-2.

The Serial Monitor (see Figure 3-3) has two parts. At the top, there is a field into which a line of text can be typed that will be sent to the board when you either click Send or press RETURN.

Below that is a larger area in which any messages coming from the Arduino board will be displayed. Right at the bottom of the window is a drop-down list where you can select the speed at which the data is sent. Whatever you select here must match the baud rate that you specify in your script's startup message. We use 9600, which is the default, so there is no need to change anything here.

So, all we need to do is launch the Serial Monitor and type some text into the Send field and press RETURN. We should then have our message flashed to us in Morse code.

Figure 3-3 The Serial Monitor window.

Project 4
High-Brightness Morse Code Translator

The little LED on Project 3 is unlikely to be visible from the ship on the horizon being lured by our bogus Evil Genius distress message. So in this project, we are going to up the power and use a 1W Luxeon LED. These LEDs are extremely bright and all the light comes from a tiny little area in the center, so to avoid any possibility of retina damage, do not stare directly into it.

We also look at how, with a bit of soldering, we can make this project into a shield that can be plugged into our Arduino board.

COMPONENTS AND EQUIPMENT

	Description	Appendix A
	Arduino Diecimila or Duemilanove board or clone	1
D1	Luxeon 1W LED	30
R1	270 Ω 0.5W metal film resistor	6
R2	4 Ω 1W resistor	16
T1	BD139 power transistor	41
	Protoshield kit (optional)	3

Hardware

The LED we used in Project 3 used about 10 mA at 2V. We can use this to calculate power using the formula:

$$P = I V$$

Power equals the voltage across something times the current flowing through it, and the unit of power is the watt. So that LED would be approximately 20 mW, or a fiftieth of the power of our 1W Luxeon LED. While an Arduino will cope just fine driving a 20 mW LED, it will not be able to directly drive the 1W LED.

This is a common problem in electronics, and can be summed up as getting a small current to control a bigger current, something that is known as amplification. The most commonly used electronic component for amplification is the transistor, so that is what we will use to switch our Luxeon LED on and off.

The basic operation of a transistor is shown in Figure 3-4. There are many different types of transistors, and probably the most common and the type that we are going to use is called an NPN bipolar transistor.

This transistor has three leads: the emitter, the collector, and the base. And the basic principle is that a small current flowing through the base will allow a much bigger current to flow between the collector and the emitter.

Just how much bigger the current is depends on the transistor, but it is typically a factor of 100. So a current of 10 mA flowing through the base could cause up to 1 A to flow through the collector. So, if we kept the 270 Ω resistor that we used to drive the LED at 10 mA, we could expect it to be more than enough to allow the transistor to switch the 350mA needed by the Luxeon LED.

The schematic diagram for our control circuit is shown in Figure 3-5.

The 270Ω resistor (R1) limits the current that flows through the base. We can calculate the current using the formula I = V/R. V will be 4.4V rather than 5V because transistors normally have a voltage of 0.6V between the base and emitter, and the highest voltage the Arduino can supply from an output pin is 5V. So, the current will be 4.4/270 = 16 mA.

R2 limits the current flowing through the LED to around 350 mA. We came up with the figure of 4 Ω by using the formula R = V/I. V will be

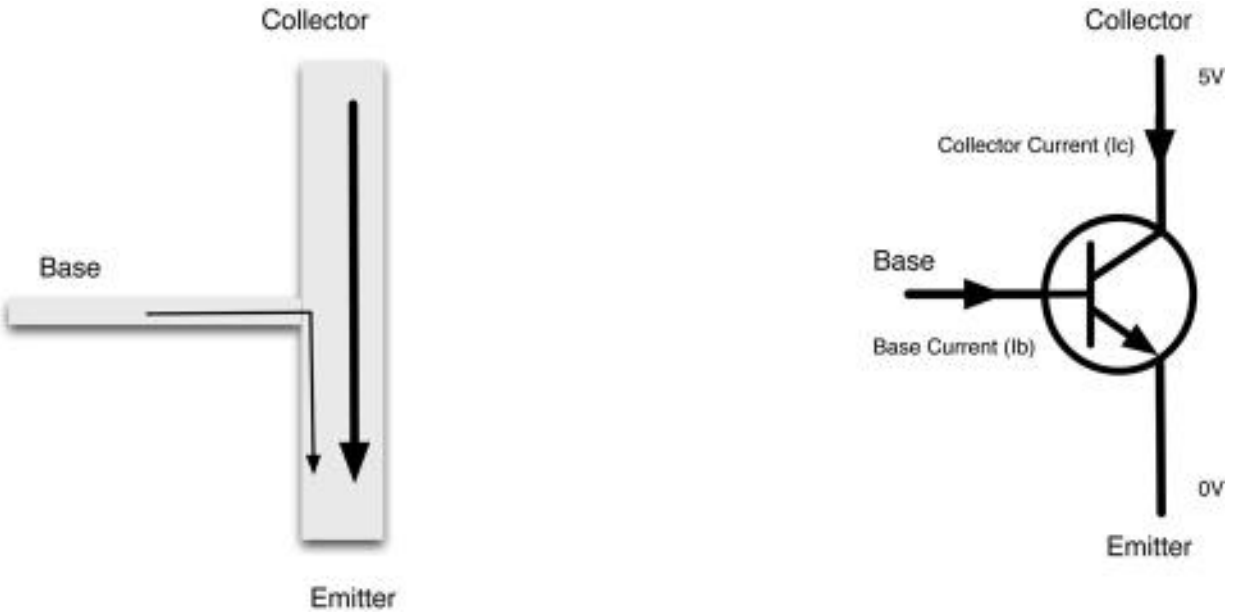


Figure 3-4 The operation of an NPN bipolar transistor.

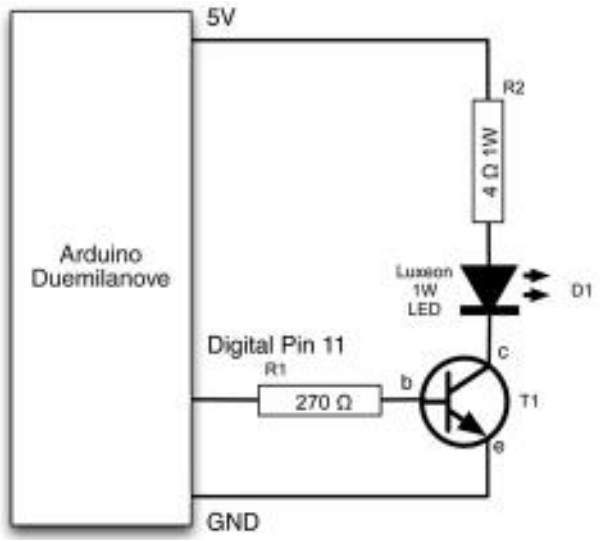


Figure 3-5 The schematic diagram for high-power LED driving.

roughly 5 – 3 – 0.6 = 1.4 V. 5V is the supply voltage, the LED drops roughly 3V and the transistor 0.6V, so the resistance should be 1.4V/350 mA = 4 Ω. We must also use a resistor that can cope with this relatively high current. The power that the resistor will burn off as heat is equal to the voltage across it multiplied by the current flowing through it. In this case, that is 350 mA * 1.4 V, which is 490 mW. To be on the safe side, we have selected a 1W resistor.

In the same way, when choosing a transistor, we need to make sure it can handle the power. When it is turned on, the transistor will consume power equal to current times voltage. In this case, the base current is small enough to ignore, so the power will just be 0.6V * 350 mA, or 210 mW. It is always a good idea to pick a transistor that can easily cope with the power. In this case, we are going to use a BD139 that has a power rating of over 12W. In Chapter 10, you can find a table of commonly used transistors.

Now we need to put out components into the breadboard according to the layout shown in Figure 3-6, with the corresponding photograph of Figure 3-8. It is crucial to correctly identify the leads of the transistor and the LED. The metallic side of the transistor should be facing the board. The LED will have a little + symbol next to the positive connection.

Later in this project we are going to show you how you can move the project from the breadboard to a more permanent design using the Arduino Protoshield. This requires some soldering, so if you think you might go on to make a shield and have the facilities to solder, I would solder some leads onto the Luxeon LED. Solder short lengths of solid-core wire to two of the six tags around the edge. They should be marked + and –. It is a good idea to color-code your leads with red for positive and blue or black for negative.

If you do not want to solder, that's fine; you just need to carefully twist the solid-core wire around the connectors as shown in Figure 3-7.

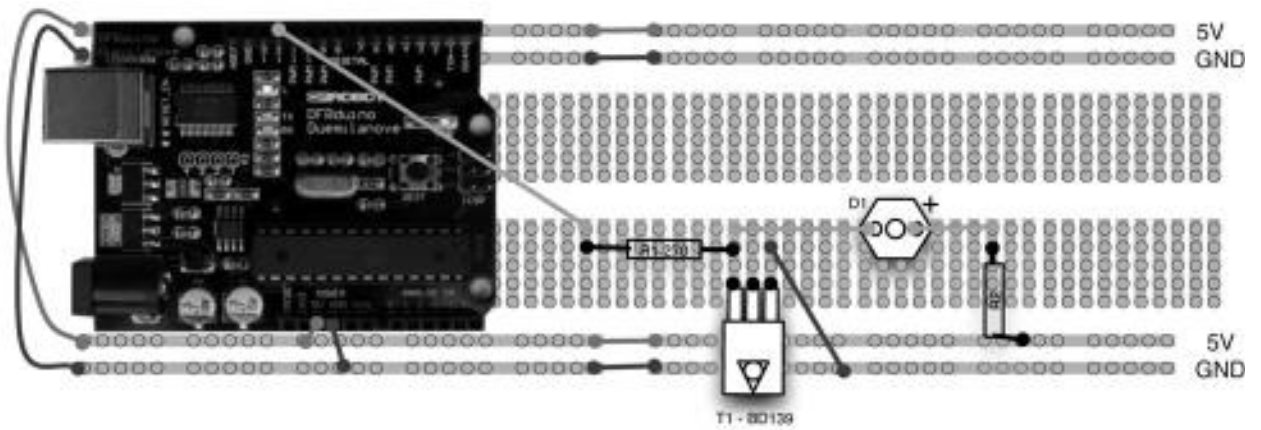


Figure 3-6 Project 4 breadboard layout.

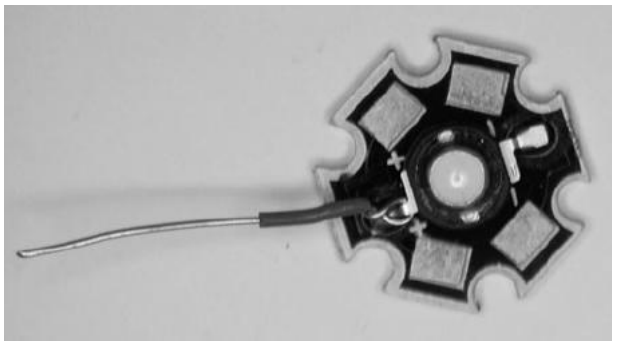


Figure 3-7 Attaching leads to the Luxeon LED without soldering.

Figure 3-8 shows the fully assembled breadboard.

Software

The only change in the software from Project 3 is that we are using digital output pin 11 rather than pin 12.

Putting It All Together

Load the completed sketch for Project 4 from your Arduino Sketchbook and download it onto your board (see Chapter 1).

Again, testing the project is the same as for Project 3. You will need to open the Serial Monitor window and just start typing.

The LED actually has a very wide angle of view, so one variation on this project would be to adapt an LED torch where the LED has a reflector to focus the beam.

Making a Shield

This is the first project that we have made that has enough components to justify making an Arduino Shield circuit board to sit on top of the Arduino board itself. We are also going to use this hardware with minor modifications in Project 6, so perhaps it is time to make ourselves a Luxeon LED Shield.

Making your own circuit boards at home is perfectly possible, but requires the use of noxious chemicals and a fair amount of equipment. But fortunately, there is another great piece of Arduino-related open-source hardware called the Arduino Protoshield. If you shop around, these can be obtained for $10 or less and will provide you with a kit of all you need to make a basic shield. That includes the board itself; the header connector pins that fit into the Arduino; and some LEDs, switches, and resistors. Please be aware that there are several variations of the Protoshield board, so you may have to adapt the following design if your board is slightly different.

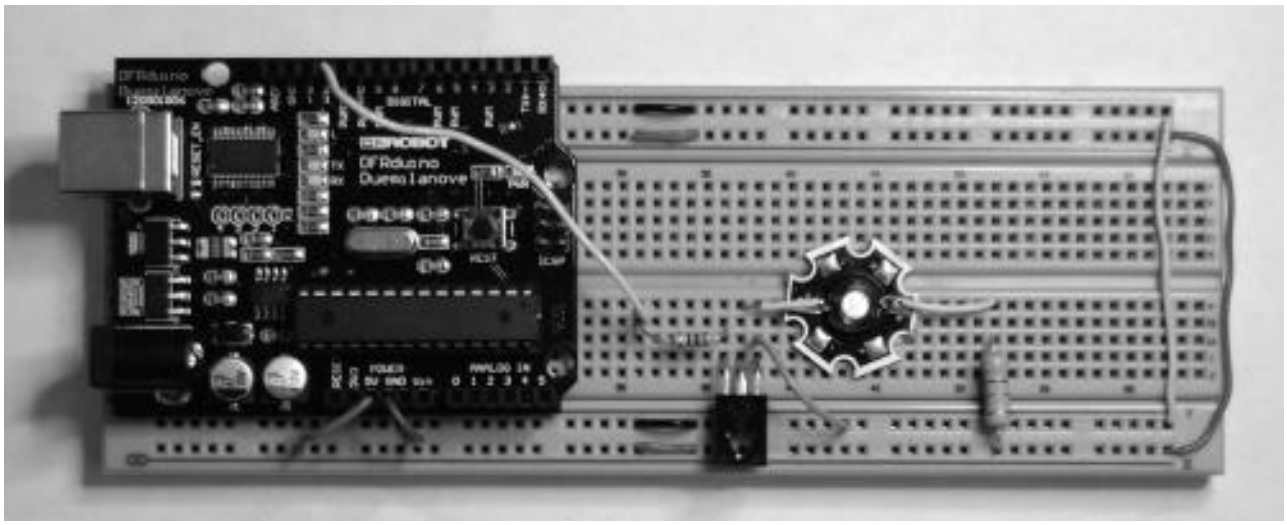


Figure 3-8 Photograph of complete breadboard for Project 4.

The components for a Protoshield are shown in Figure 3-9, the most important part being the Protoshield circuit board (PCB). It is possible to just buy the Protoshield circuit board on its own, which for many projects will be all you need.

We are not going to solder all the components that came with our kit onto the board. We are just going to add the power LED, its resistor, and just the bottom pins that connect to the Arduino board, as this is going to be a top shield and will not have any other shields on top of it.

A good guide for assembling circuit boards is to solder in place the lowest components first. So in this case we will solder the resistors, the LED, the reset switch, and then the bottom pin connectors.

The 1K resistor, LED, and switch are all pushed through from the top of the board and soldered underneath (Figure 3-10). The short part of the connector pins will be pushed up from underneath the board and soldered on top.

When soldering the connector pins, make sure they are lined up correctly, as there are two parallel rows for the connectors: one for the connection to the pins below and one for the sockets, which we are not using, that are intended to connect to further shields.

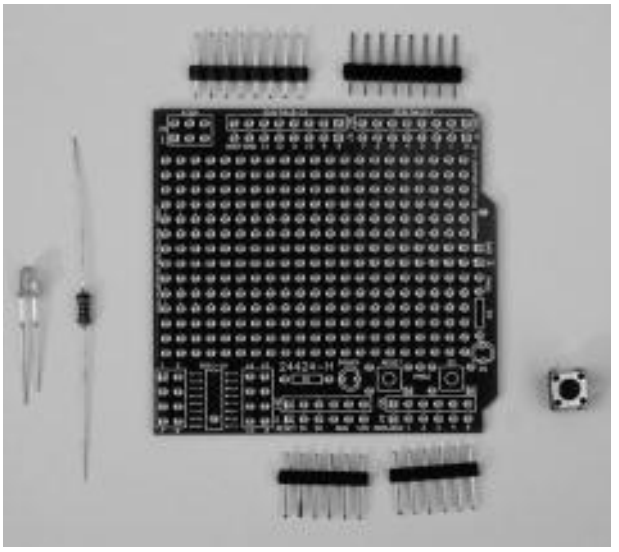


Figure 3-9 Protoshield in kit form.

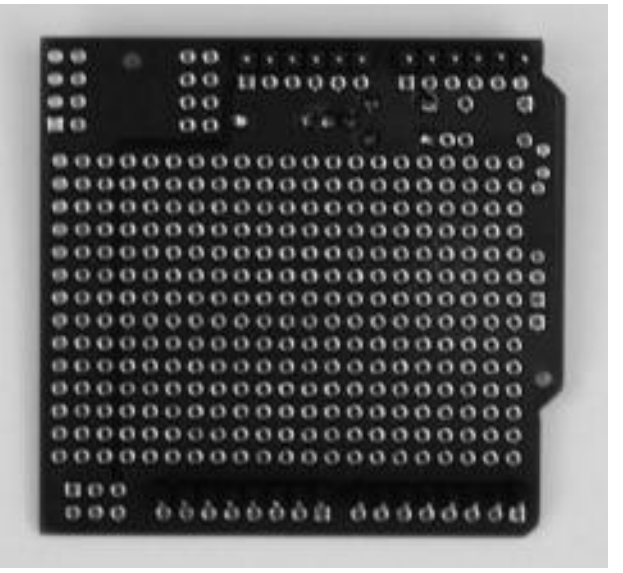


Figure 3-10 The underside of the Protoshield.

A good way to ensure that the headers are in the right place is to fit the sections of header into an Arduino board and then place the shield on top and solder the pins while it's still plugged into the Arduino board. This will also ensure that the pins are straight.

When all the components have been soldered in place, you should have a board that looks like Figure 3-11.

We can now add our components for this project, which we can take from the breadboard. First, line up all the components in their intended places according to the layout of Figure 3-12 to make sure everything fits in the available space.

This kind of board is double-sided—that is, you can solder to the top or bottom of the board. As you can see from the layout in Figure 3-12, some of the connections are in strips like a breadboard.

We are going to mount all the components on the top side, with the leads pushed through and soldered on the underside where they emerge from the board. The leads of the components underneath can then be connected up and excess leads snipped off. If necessary, lengths of solid-core wire can be used where the leads will not reach.

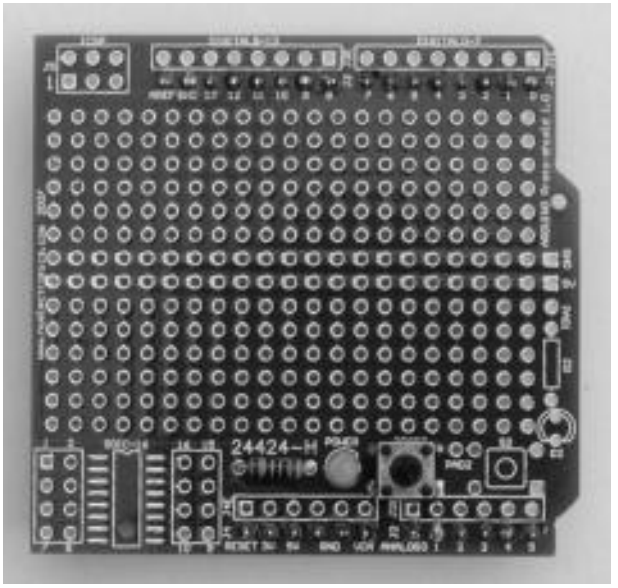


Figure 3-11 Assembled basic Protoshield.

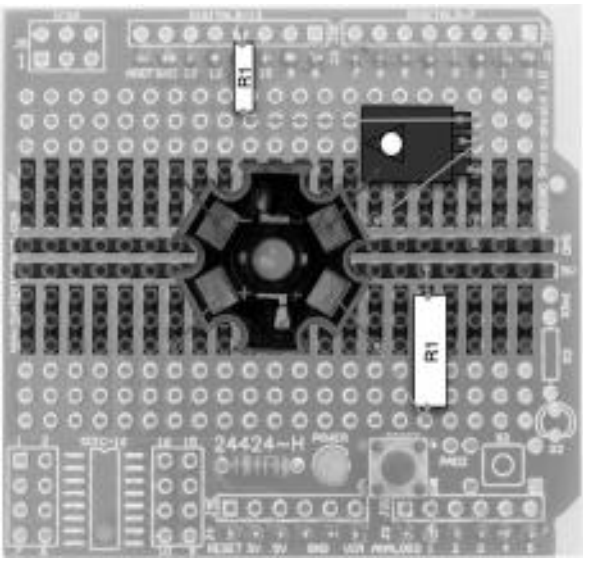


Figure 3-12 Project 4 Protoshield layout.

Figure 3-13 shows the completed shield. Power up your board and test it out. If it does not work as soon as you power it up, disconnect it from the power right away and carefully check the shield for any short circuits or broken connections using a multimeter.

Congratulations! You have created your first Arduino Shield, and it is one that we can reuse in later projects.

Summary

So, we have made a start on some simple LED Projects and discovered how to use high power Luxeon LEDs. We have also learnt a bit more about programming our Arduino board in C.

In the next chapter, we are going to extend this by looking at some more LED-based projects including a model traffic signal and a high power strobe light.

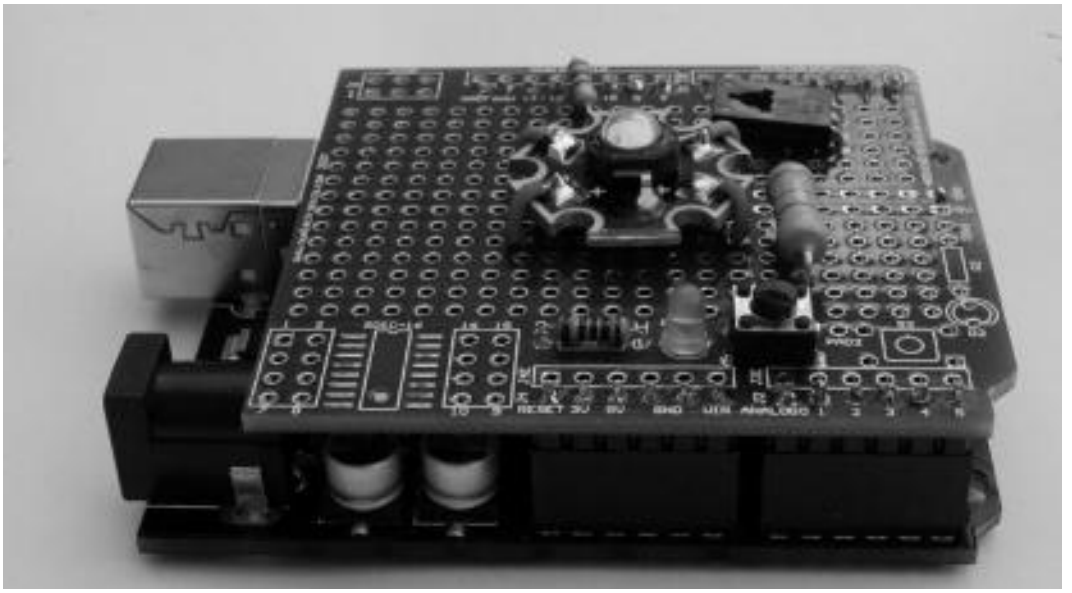


Figure 3-13 Complete Luxeon shield attached to an Arduino board.

CHAPTER 4

More LED Projects

In this chapter, we are going to build on those versatile little components, LEDs, and learn a bit more about digital inputs and outputs, including how to use push-button switches.

The projects that we are going to build in this chapter are a model traffic signal, two strobe light projects, and a bright light module using high-power Luxeon LEDs.

Digital Inputs and Outputs

The digital pins 0 to 12 can all be used as either an input or an output. This is set in your sketch. Since you are going to be connecting electronics to one of these pins, it is unlikely that you are going to want to change the mode of a pin. That is, once a pin is set to be an output, you are not going to change it to be an input midway through a sketch.

For this reason, it is a convention to set the direction of a digital pin in the setup function that must be defined in every sketch.

For example, the following code sets digital pin 10 to be an output and digital pin 11 to be an input. Note how we use a variable declaration in our sketch to make it easier to change the pin used for a particular purpose later on.

```
int ledPin = 10;
int switchPin = 11;

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

Project 5
Model Traffic Signal

So now we know how to set a digital pin to be an input, we can build a project for model traffic signals using red, yellow, and green LEDs. Every time we press the button, the traffic signal will go to the next step in the sequence. In the UK, the sequence of such traffic signals is red, red and amber together, green, amber, and then back to red.

As a bonus, if we hold the button down, the lights will change in sequence by themselves with a delay between each step.

The components for Project 5 are listed next. When using LEDs, for best effect, try and pick LEDs of similar brightness.

COMPONENTS AND EQUIPMENT		
	Description	**Appendix**
	Arduino Diecimila or Duemilanove board or clone	1
D1	5-mm red LED	23
D2	5-mm yellow LED	24
D3	5-mm green LED	25
R1-R3	270 Ω 0.5W metal film resistor	6
R4	100 KΩ 0.5W metal film resistor	13
S1	Miniature push to make switch	48

Hardware

The schematic diagram for the project is shown in Figure 4-1.

The LEDs are connected in the same way as our earlier project, each with a current-limiting resistor. The digital pin 5 is "pulled" down to GND by R4 until the switch is pressed, which will make it go to 5V.

A photograph of the project is shown in Figure 4-2 and the board layout in Figure 4-3.

Software

The sketch for Project 5 is shown in Listing Project 5.

The sketch is fairly self-explanatory. We only check to see if the switch is pressed once a second, so pressing the switch rapidly will not move the light sequence on. However, if we press and hold the switch, the lights will automatically sequence round.

We use a separate function setLights to set the state of each LED, reducing three lines of code to one.

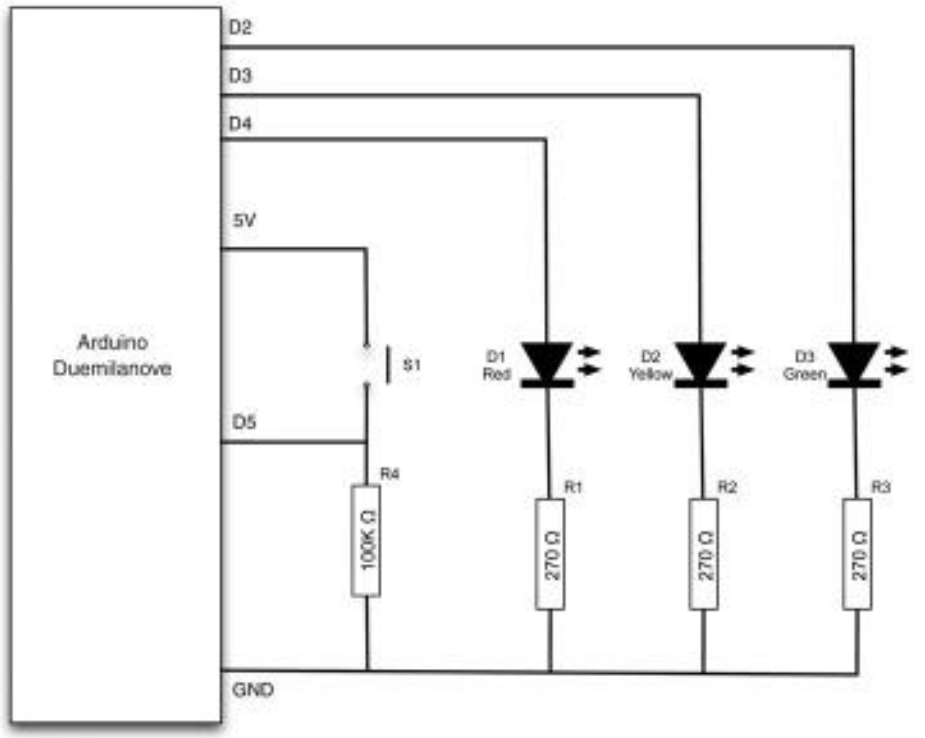


Figure 4-1 Schematic diagram for Project 5.

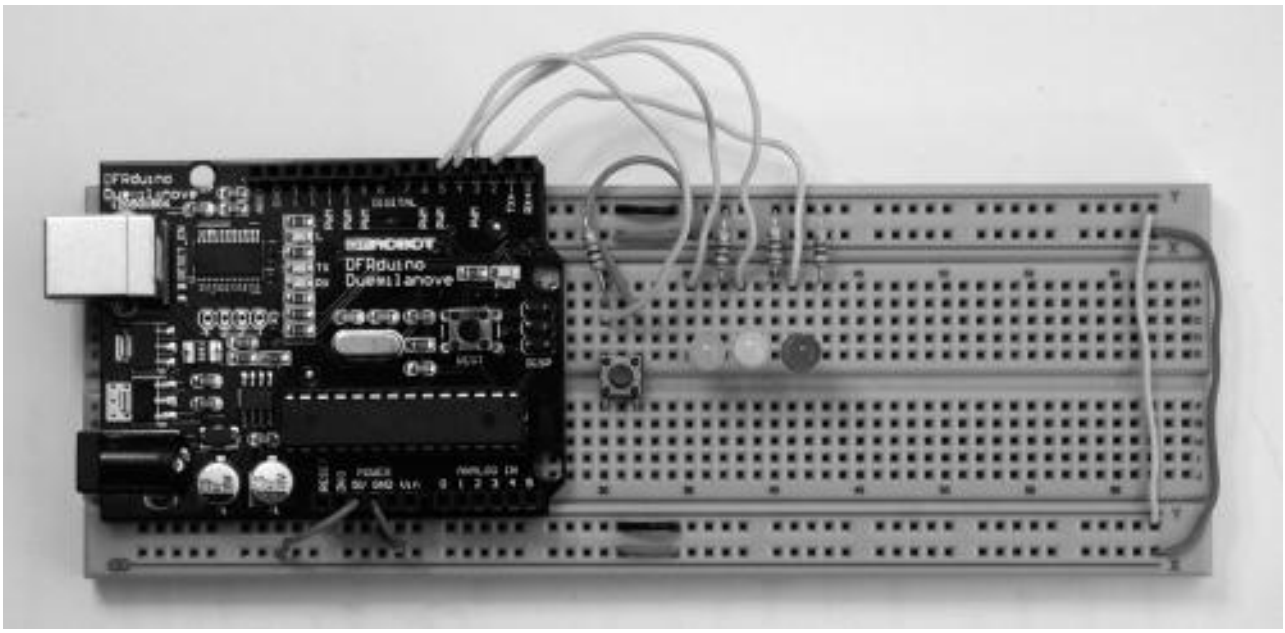


Figure 4-2 Project 5. A model traffic signal.

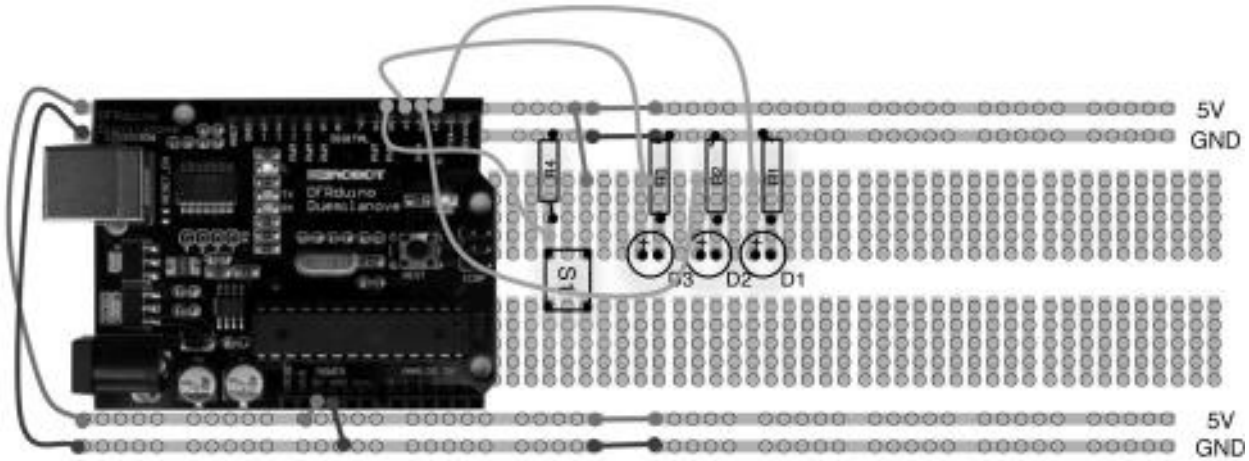


Figure 4-3 Breadboard layout for Project 5.

LISTING PROJECT 5

```
int redPin = 2;
int yellowPin = 3;
int greenPin = 4;
int buttonPin = 5;

int state = 0;

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

void loop()
{
  if (digitalRead(buttonPin))
  {
    if (state == 0)
    {
      setLights(HIGH, LOW, LOW);
      state = 1;
    }
    else if (state == 1)
    {
      setLights(HIGH, HIGH, LOW);
      state = 2;
    }
    else if (state == 2)
    {
      setLights(LOW, LOW, HIGH);
      state = 3;
    }
    else if (state == 3)
    {
      setLights(LOW, HIGH, LOW);
      state = 0;
    }
    delay(1000);
  }
}

void setLights(int red, int yellow,
   int green)
{
  digitalWrite(redPin, red);
  digitalWrite(yellowPin, yellow);
  digitalWrite(greenPin, green);
}
```

Putting It All Together

Load the completed sketch for Project 5 from your Arduino Sketchbook (see Chapter 1).

Test the project by holding down the button and make sure the LEDs all light in sequence.

Project 6
Strobe Light

This project uses the same high-brightness Luxeon LED as the Morse code translator. It adds to that a variable resistor, sometimes called a potentiometer. This provides us with a control that we can rotate to control the flashing rate of the strobe light.

CAUTION **This is a strobe light; it flashes brightly. If you have a health condition such as epilepsy, you may wish to skip this project.**

COMPONENTS AND EQUIPMENT

	Description	Appendix
	Arduino Diecimila or Duemilanove board or clone	1
D1	Luxeon 1W LED	30
R1	270 Ω 0.5W metal film resistor	6
R2	4 Ω 1W resistor	16
T1	BD139 power transistor	41
R3	100K linear potentiometer	17
	Protoshield kit (optional)	3
	2.1-mm power plug (optional)	49
	9V battery clip (optional)	50

Hardware

The hardware for this project is basically the same as for Project 4, but with the addition of a variable resistor (see Figure 4-4).

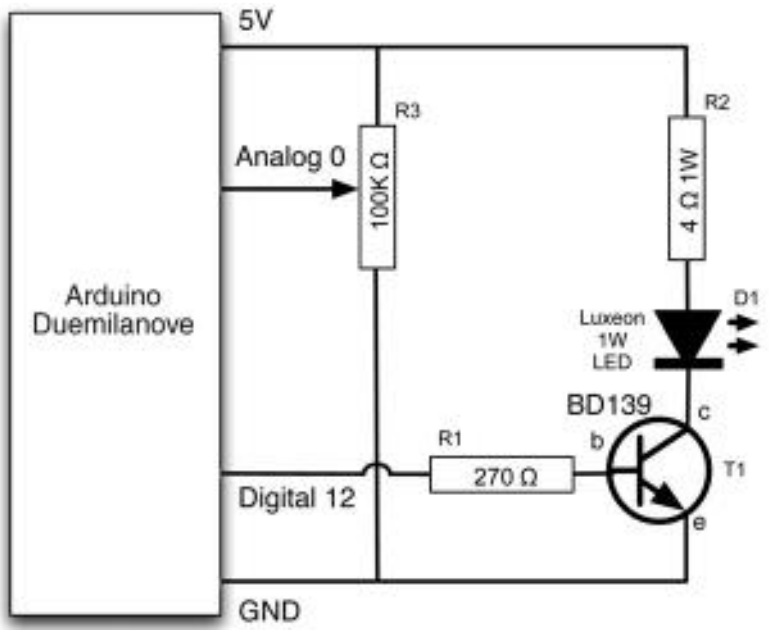


Figure 4-4 Schematic diagram for Project 6.

The Arduino is equipped with six analog input pins numbered Analog 0 to Analog 5. These measure the voltage at their input and give a number between 0 (0V) and 1023 (5V).

We can use this to detect the position of a control knob by connecting a variable resistor acting as a potential divider to our analog pin. Figure 4-5 shows the internal structure of a variable resistor.

A variable resistor is a component that is typically used for volume controls. It is constructed as a circular conductive track with a gap in it and connections at both ends. A slider provides a moveable third connection.

You can use a variable resistor to provide a variable voltage by connecting one end of the resistor to 0V and the other end to 5V, and then the voltage at the slider will vary between 0 and 5V as you turn the knob.

As you would expect, the breadboard layout (Figure 4-6) is similar to Project 4.

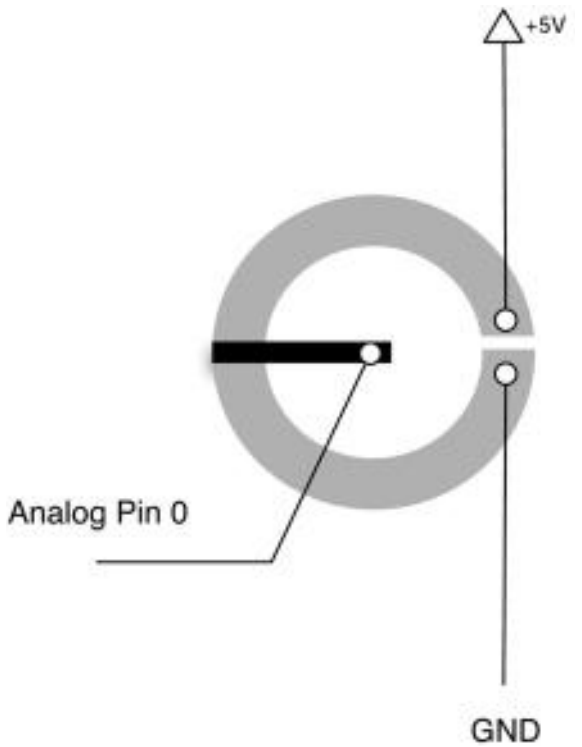


Figure 4-5 The internal workings of a variable resistor.

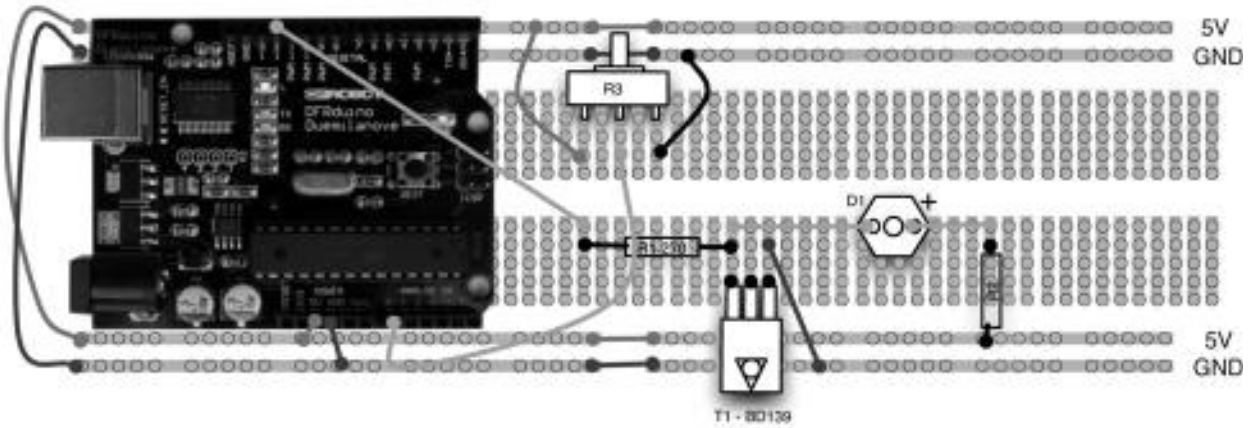


Figure 4-6 Breadboard layout for Project 6.

Software

The listing for this project is shown here. The interesting parts are concerned with reading the value from the analog input and controlling the rate of flashing.

For analog pins, it is not necessary to use the pinMode function, so we do not need to add anything into the setup function.

Let us say that we are going to vary the rate of flashing between once a second and 20 times a second; the delays between turning the LED on and off will be 500 milliseconds and 25 milliseconds, respectively.

So, if our analog input changes from 0 to 1023, the calculation that we need to determine the flash delay is roughly:

```
flash_delay = (1023 – analog_value) / 2
   + 25
```

So an analog_value of 0 would give a flash_delay of 561 and an analog_value of 1023 would give a delay of 25. We should actually be dividing by slightly more than 2, but it makes things easier if we keep everything as integers.

LISTING PROJECT 6

```
int ledPin = 11;
int analogPin = 0;

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

void loop()
{
  int period = (1023 -
   analogRead(analogPin)) / 2 + 25;
  digitalWrite(ledPin, HIGH);
  delay(period);
  digitalWrite(ledPin, LOW);
  delay(period);
}
```

Putting It All Together

Load the completed sketch for Project 6 from your Arduino Sketchbook and download it to the board (see Chapter 1).

You will find that turning the variable resistor control clockwise will increase the rate of flashing as the voltage at the analog input increases. Turning it counterclockwise will slow the rate of flashing.

Making a Shield

If you want to make a shield for this project, you can either adapt the shield for Project 4 or create a new shield from scratch.

The layout of components on the Protoshield is shown in Figure 4-7.

This is basically the same as for Project 4, except that we have added the variable resistor. The pins on a variable resistor are too thick to fit into the holes on the Protoshield, so you can either attach it using wires or, as we have done here, carefully solder the leads to the top surface where they touch the board. To provide some mechanical strength, the variable resistor can be glued in place first with a drop of Super Glue. The wiring for the variable resistor to 5V, GND, and Analog 0 can be made underneath the board out of sight.

Having made a shield, we can make the project independent of our computer by powering it from a 9V battery.

To power the project from a battery, we need to make ourselves a small lead that has a PP3 battery clip on one end and a 2.1-mm power plug on the other. Figure 4-8 shows the semi-assembled lead.

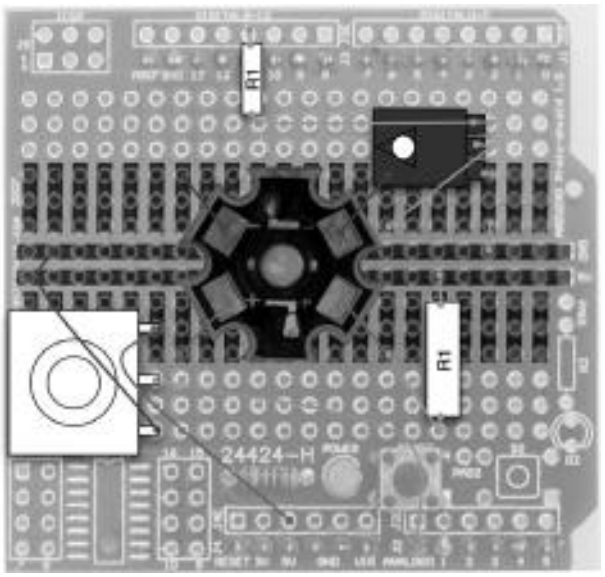


Figure 4-7 Protoshield layout for Project 6.

Figure 4-8 Creating a battery lead.

Project 7
S.A.D. Light

Seasonal affective disorder (S.A.D.) affects a great number of people, and research has shown that exposure to a bright white light that mimics daylight for 10 or 20 minutes a day has a beneficial effect. To use this project for such a purpose, I would suggest the use of some kind of diffuser such as frosted glass, as you should not stare directly at the point light sources of the LEDs.

This is another project based on the Luxeon high-brightness LEDs. We will use an analog input connected to a variable resistor to act as a timer control, turning the LED on for a given period set by the position of the variable resistor's slider. We will also use an analog output to slowly raise the brightness of the LEDs as they turn on and then slowly decrease it as they turn off. To make the light bright enough to be of use as a S.A.D. light, we are going to use not just one Luxeon LED but six.

At this point, the caring nature of this project may be causing the Evil Genius something of an identity crisis. But, fear not—in Project 8, we will turn this same hardware into a fearsome high-powered strobe light.

COMPONENTS AND EQUIPMENT

	Description	Appendix
	Arduino Diecimila or Duemilanove board or clone	1
D1-6	Luxeon 1W LED	30
R1-3	1 KΩ 0.5W metal film resistor	7
R4-5	4 Ω 2W resistor	16
R6	100K linear potentiometer	17
IC1-2	LM317 Voltage regulator	45
T1-2	2N7000 FET	42
	Regulated 15V 1A power supply	51
	Perf board	53
	Three-way screw terminal	52

- Please note this is one of the projects in this book that requires soldering.
- You are going to need six Luxeon LEDs for this project. If you want to save some money, look at online auctions, where ten of these should be available for $10 to $20.

Hardware

Some of the digital pins, namely digital pins 5, 6, 9, 10, and 11, can provide a variable output rather than just 5V or nothing. These are the pins with PWM next to them on the board. This is the reason that we have switched to using pin 11 for our output control.

PWM stands for Pulse Width Modulation, and refers to the means of controlling the amount of power at the output. It does so by rapidly turning the output on and off.

The pulses are always delivered at the same rate (roughly 500 per second), but the length of the pulses is varied. If the pulse is long, our LED will be on all the time. If, however, the pulses are short, the LED is only actually lit for a small portion of the time. This happens too fast for the observer to even tell that the LED is flickering, and it just appears that the LED is brighter or dimmer.

Readers may wish to refer to Wikipedia for a fuller description of PWM.

The value of the output can be set using the function analogWrite, which requires an output value between 0 and 255, where 0 will be off and 255 full power.

As you can see from the schematic diagram in Figure 4-9, the LEDs are arranged in two columns of three. The LEDs are also supplied from an external 15V supply rather than the 5V supply that we used before. Since each LED consumes about 300 mA, each column will draw about 300 mA and so the supply must be capable of supplying 0.6A (1 A to be on the safe side).

This is the most complex schematic so far in our projects. We are using two integrated-circuit variable voltage regulators to limit the current flowing to the LEDs. The output of the voltage regulators will normally be 1.25V above whatever the voltage is at the Ref pin of the chip. This means that if we drive our LEDs through a 4 Ω resistor, there will be a current of roughly $I = V/R$, or $1.25 / 4 = 312$ mA flowing through it (which is about right).

The FET (field effect transistor) is like our normal bipolar transistor, in that it can act as a switch, but it has a very high off resistance. So when it is not triggered by a voltage at its gate, it's as if it isn't there in the circuit. However, when it is turned on, it will pull down the voltage at the regulator's Ref pin to a low enough voltage to prevent any current flowing into the LEDs turning them off. Both of the FETs are controlled from the same digital pin 11.

The completed LED module is shown in Figure 4-10 and the perf board layout in Figure 4-11.

The module is built on perf (perforated) board. The perf board is just a board with holes in it. It has no connections at all. So it acts as a structure on which to fit your components, but you have to wire them up on the underside of the board, either by connecting their leads together or adding wires.

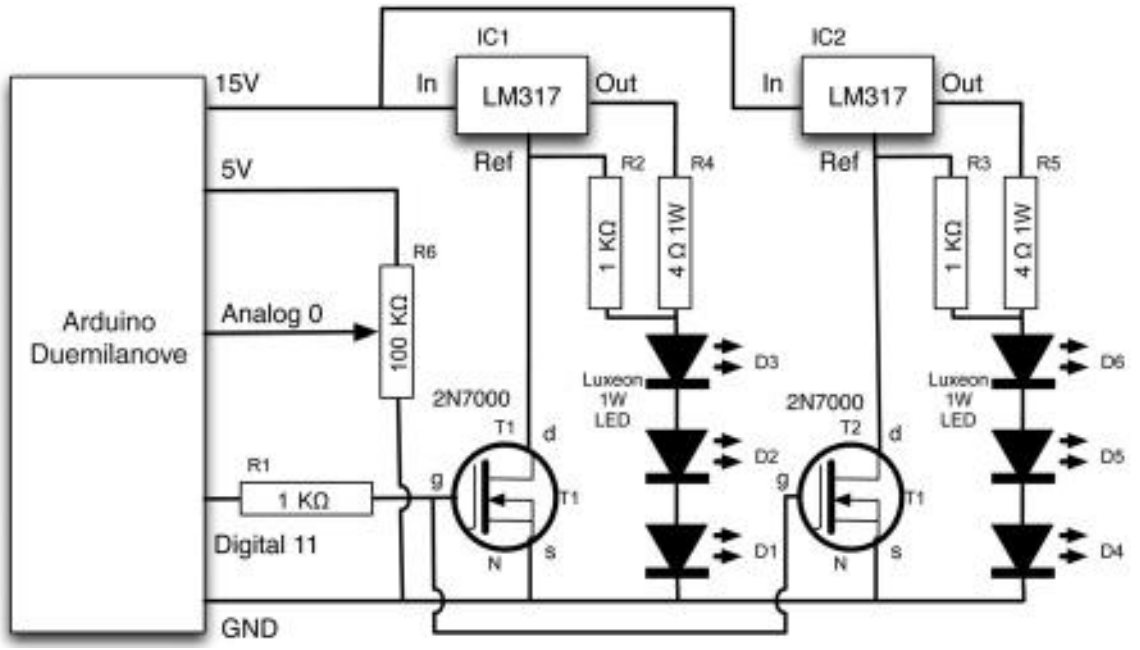


Figure 4-9 Schematic diagram for Project 7.

It is easier to solder two wires onto each LED before fitting them onto the board. It is a good idea to color-code those leads—red for positive and black or blue for negative—so that you get the LEDs in the correct way round.

The LEDs will get hot, so it is a good idea to leave a gap between them and the perf board using the insulation on the wire to act as a spacer. The voltage regulator will also get hot but should be okay without a heatsink. The voltage regulator

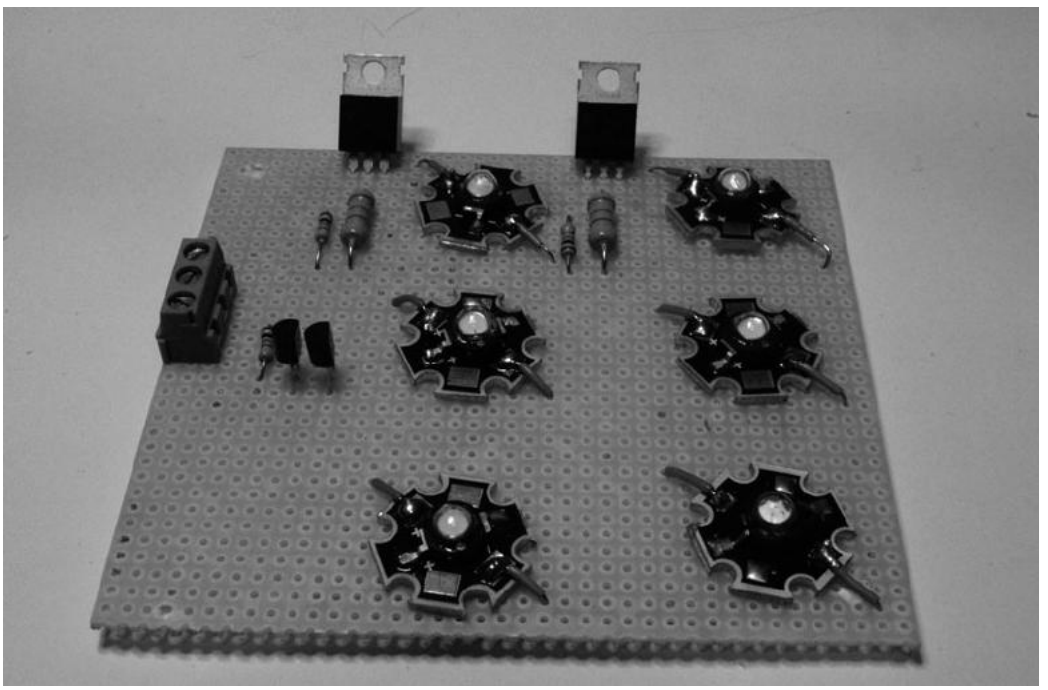


Figure 4-10 Project 7. High-power light module.

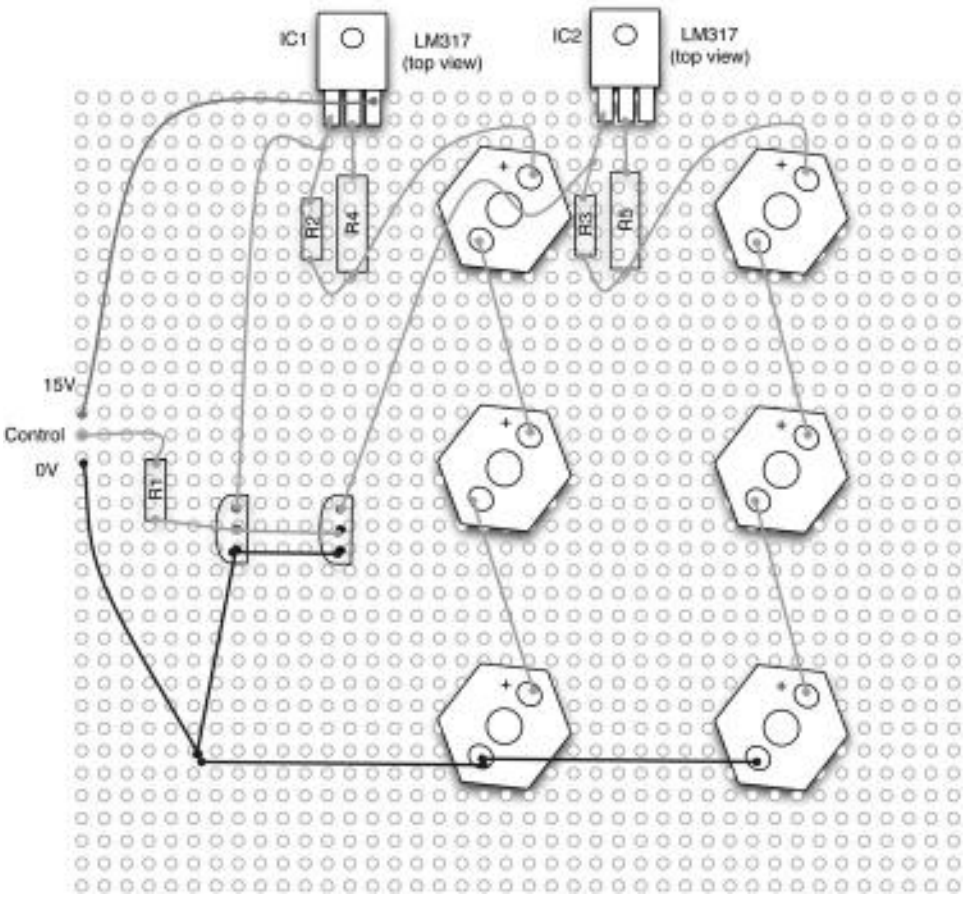


Figure 4-11 Perf board layout.

integrator circuits (ICs) actually have built-in thermal protection and will automatically reduce the current if they start to get too hot.

The screw terminals on the board are for the power supply GND and 15V and a control input. When we connect this to the Arduino board, the 15V will come from the Vin pin on the Arduino, which in turn is supplied from a 15V power supply.

Our high-power LED module will be of use in other projects, so we are going to plug the variable resistor directly into the Analog In strip of connectors on the Arduino board. The spacing of pins on the variable resistor is 1/5 of an inch, which means that if the middle slider pin is in the socket for Analog 2, the other two pins will be in the sockets for Analog 0 and 4. You can see this arrangement in Figure 4-12.

You may remember that analog inputs can also be used as digital outputs by adding 14 to their pin number. So in order to have 5V at one end of our variable resistor and 0V at the other, we are going to set the outputs of analog pins 0 and 4 (digital pins 14 and 18) to 0V and 5V, respectively.

Software

At the top of the sketch, after the variable used for pins, we have four other variables: startupSeconds, turnOffSeconds, minOnSeconds, and maxOnSeconds. This is common practice in programming. By putting these values that we might want to change into variables and making them visible at the top of the sketch, it makes it easier to change them.

LISTING PROJECT 7

```
int ledPin = 11;
int analogPin = 2;

int startupSeconds = 20;
int turnOffSeconds = 10;
int minOnSeconds = 300;
int maxOnSeconds = 1800;

int brightness = 0;

void setup()
{
  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, HIGH);
  pinMode(14, OUTPUT);                    // Use Analog pins 0 and 4 for
  pinMode(18, OUTPUT);                    // the variable resistor
  digitalWrite(18, HIGH);
  int analogIn = analogRead(analogPin);
  int onTime = map(analogIn, 0, 1023, minOnSeconds, maxOnSeconds);
  turnOn();
  delay(onTime * 1000);
  turnOff();
}

void turnOn()
{
  brightness = 0;
  int period = startupSeconds * 1000 / 256;
  while (brightness < 255)
  {
    analogWrite(ledPin, 255 - brightness);
    delay(period);
    brightness ++;
  }
}

void turnOff()
{
  int period = turnOffSeconds * 1000 / 256;
  while (brightness >= 0)
  {
    analogWrite(ledPin, 255 - brightness);
    delay(period);
    brightness —;
  }
}

void loop()
{}
```

The variable startupSeconds determines how long it will take for the brightness of the LEDs to be gradually raised until it reaches maximum brightness. Similarly, turnOffSeconds determines the time period for dimming the LEDs. The variables minOnSeconds and maxOnSeconds determine the range of times set by the variable resistor.

In this sketch, there is nothing in the loop function. Instead all the code is in setup. So, the light will automatically start its cycle when it is powered up. Once it has finished, it will stay turned off until the reset button is pressed.

The slow turn-on is accomplished by gradually increasing the value of the analog output by 1. This is carried out in a while loop, where the delay is set to 1/255 of the startup time so that after 255 steps maximum brightness has been achieved. Slow turn-off works in a similar manner.

The time period at full brightness is set by the analog input. Assuming that we want a range of times from 5 to 30 minutes, we need to convert the value of 0 to 1023 to a number of seconds between 300 and 1800. Fortunately, there is a handy Arduino function that we can use to do this. The function map takes five arguments: the value you want to convert, the minimum input value (0 in this case), the maximum input value (1023), the minimum output value (300), and the maximum output value (1800).

Putting It All Together

Load the completed sketch for Project 7 from your Arduino Sketchbook and download it to the board (see Chapter 1).

You now need to attach wires from the Vin, GND, and digital pin 11 of the Arduino board to the three screw terminals of the LED module (Figure 4-12). Plug a 15V power supply into the board's power socket and you are ready to try it.

To start the light sequence again, click the reset button.

Project 8
High-Powered Strobe Light

For this project, you can use the six Luxeon LED module of Project 7 or you can use the Luxeon shield that we created for Project 4. The software will be almost the same in both cases.

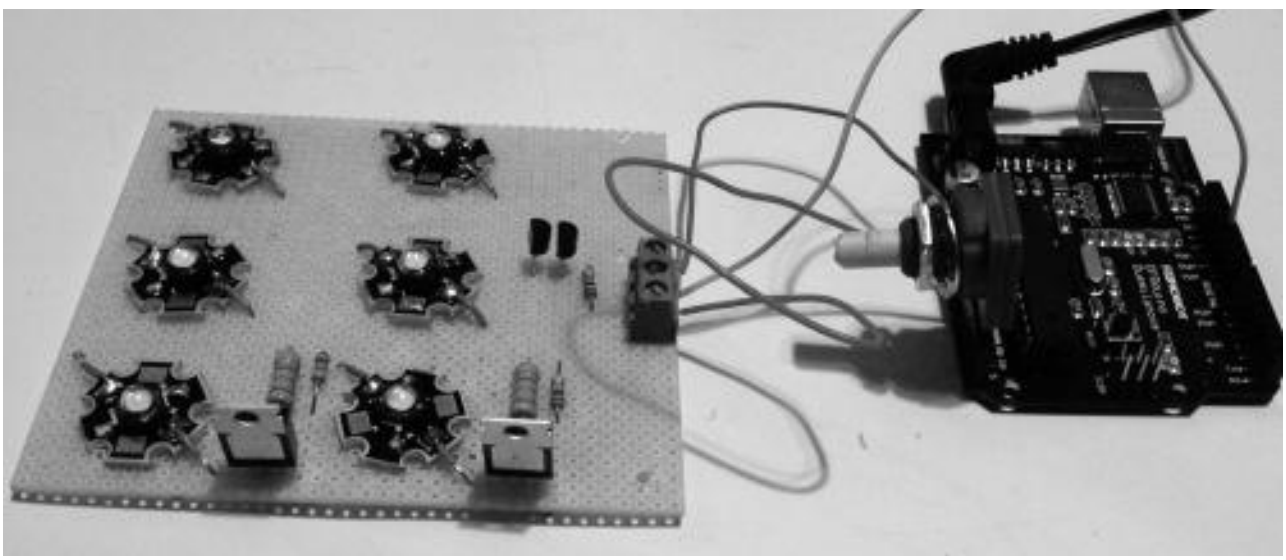


Figure 4-12 Project 7. S.A.D. light.

In this version of the strobe light, we are going to control the strobe light effect from the computer with commands. We will send the following commands over the USB connection using the Serial Monitor.

0–9	0–9 sets the speed of the following mode commands: 0 for off, 1 for slow, and 9 for fast
w	Wave effect gradually getting lighter then darker
s	Strobe effect

Hardware

See Project 4 (the Morse code translator using a single Luxeon LED shield) or Project 7 (array of six Luxeon LEDs) for components and construction details.

Software

This sketch uses the sin function to produce a nice, gently increasing brightness effect. Apart from that, the techniques we use in this sketch have mostly been used in earlier projects.

LISTING PROJECT 8

```
int ledPin = 11;

int period = 100;

char mode = 'o';  // o-off, s-strobe, w-wave

void setup()
{
  pinMode(ledPin, OUTPUT);
  analogWrite(ledPin, 255);
  Serial.begin(9600);
}

void loop()
{
  if (Serial.available())
  {
    char ch = Serial.read();
    if (ch == '0')
    {
      mode = 0;
      analogWrite(ledPin, 255);
    }
    else if (ch > '0' && ch <= '9')
    {
      setPeriod(ch);
    }
    else if (ch == 'w' || ch == 's')
```

(continued)

LISTING PROJECT 8 (continued)

```
    {
      mode = ch;
    }
  }
  if (mode == 'w')
  {
    waveLoop();
  }
  else if (mode == 's')
  {
    strobeLoop();
  }
}

void setPeriod(char ch)
{
 int period1to9 = 9 - (ch - '0');
 period = map(period1to9, 0, 9, 50, 500);
}

void waveLoop()
{
  static float angle = 0.0;
  angle = angle + 0.01;
  if (angle > 3.142)
  {
     angle = 0;
  }
  // analogWrite(ledPin, 255 - (int)255 * sin(angle));  // Breadboard
  analogWrite(ledPin, (int)255 * sin(angle));           // Shield
  delay(period / 100);
}

void strobeLoop()
{
 //analogWrite(ledPin, 0);        // breadboard
 analogWrite(ledPin, 255);        // shield
 delay(10);
 //analogWrite(ledPin, 255);      // breadboard
 analogWrite(ledPin, 0);          // shield
 delay(period);
}
```

Putting It All Together

Load the completed sketch for Project 8 from your Arduino Sketchbook and download it to the board (see Chapter 1).

When you have installed the sketch and fitted the Luxeon shield or connected the bright six-Luxeon panel, initially the lights will be off. Open the Serial Monitor window, type **s**, and press RETURN. This will start the light flashing. Try the speed commands 1 to 9. Then try typing the **w** command to switch to wave mode.

Random Number Generation

Computers are deterministic. If you ask them the same question twice, you should get the same answer. However, sometimes, you want chance to take a hand. This is obviously useful for games.

It is also useful in other circumstances—for example, a "random walk," where a robot makes a random turn, then moves forward a random distance or until it hits something, then reverses and turns again, is much better at ensuring the robot covers the whole area of a room than a more fixed algorithm that can result in the robot getting stuck in a pattern.

The Arduino library includes a function for creating random numbers.

There are two flavors of the function random. It can either take two arguments (minimum and maximum) or one argument (maximum), in which case the minimum is assumed to be 0.

Beware, though, because the maximum argument is misleading, as the highest number you can actually get back is the maximum minus one.

So, the following line will give x a value between 1 and 6:

```
int x = random(1, 7);
```

and the following line will give x a value between 0 and 9:

```
int x = random(10);
```

As we pointed out at the start of this section, computers are deterministic, and actually our random numbers are not random at all, but a long sequence of numbers with a random distribution. You will actually get the same sequence of numbers every time you run your script.

A second function (randomSeed) allows you to control this. The randomSeed function determines where in its sequence of pseudo-random numbers the random number generator starts.

A good trick is to use the value of a disconnected analog input, as this will float around at a different value and give at least 1000 different starting points for our random sequence. This wouldn't do for the lottery, but is acceptable for most applications. Truly random numbers are very hard to come by and involve special hardware.

Project 9
LED Dice

This project uses what we have just learned about random numbers to create electronic dice with six LEDs and a button. Every time you press the button, the LEDs "roll" for a while before settling on a value and then flashing it.

COMPONENTS AND EQUIPMENT

	Description	Appendix
	Arduino Diecimila or Duemilanove board or clone	1
D1-7	Standard red LEDs	23
R1-7	270 Ω 0.5W metal film resistor	6
S1	Miniature push-to-make switch	48
R8	100K Ω 0.5W metal film resistor	13

Hardware

The schematic diagram for Project 9 is shown in Figure 4-13. Each LED is driven by a separate digital output via a current-limiting resistor. The only other components are the switch and its associated pull-down resistor.

Even though a die can only have a maximum of six dots, we still need seven LEDs to have the normal arrangement of a dot in the middle for odd-numbered rolls.

Figure 4-14 shows the breadboard layout and Figure 4-15 the finished breadboard.

Software

This sketch is fairly straightforward; there are a few nice touches that make the dice behave in a similar way to real dice. For example, as the dice rolls, the number changes, but gradually slows. Also, the length of time that the dice rolls is also random.

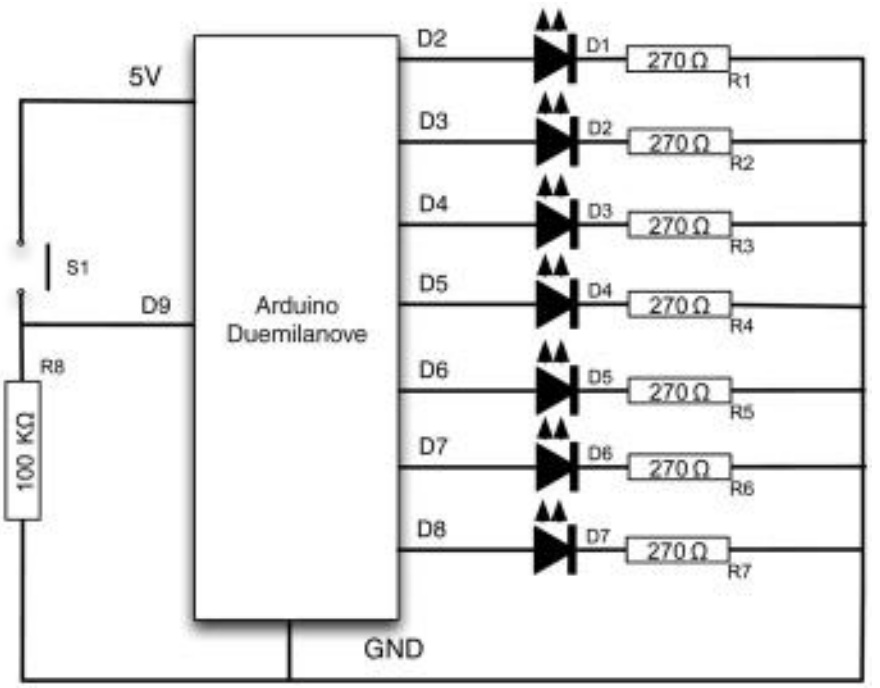


Figure 4-13 Schematic diagram for Project 9.

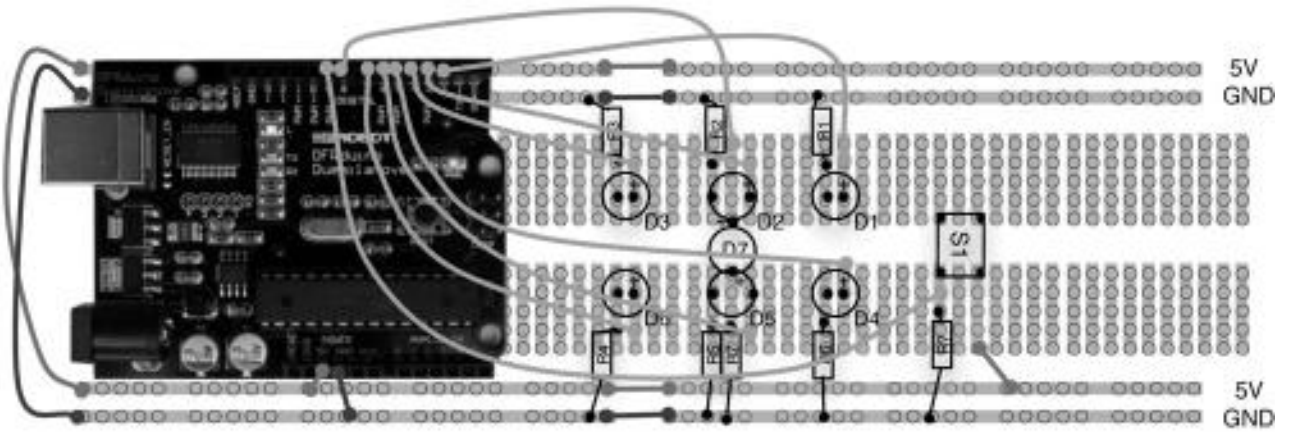


Figure 4-14 The breadboard layout for Project 9.

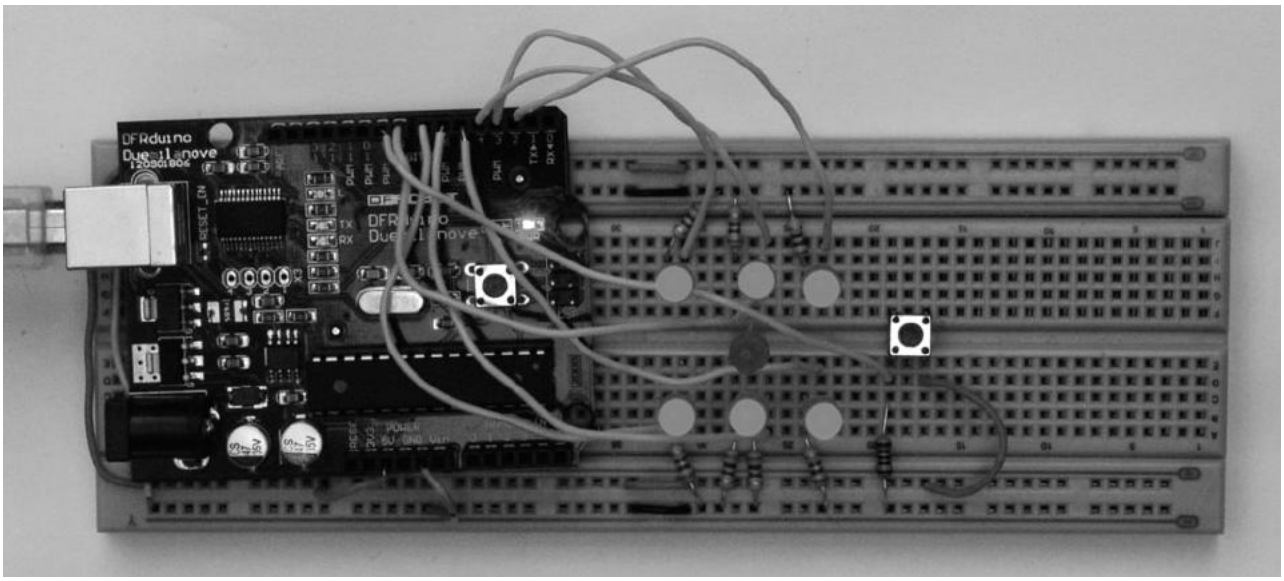


Figure 4-15 Project 9. LED dice.

LISTING PROJECT 9

```
int ledPins[7] = {2, 3, 4, 5, 6, 7, 8};
int dicePatterns[7][7] = {
  {0, 0, 0, 0, 0, 0, 1},                // 1
  {0, 0, 1, 1, 0, 0, 0},                // 2
  {0, 0, 1, 1, 0, 0, 1},                // 3
  {1, 0, 1, 1, 0, 1, 0},                // 4
  {1, 0, 1, 1, 0, 1, 1},                // 5
  {1, 1, 1, 1, 1, 1, 0},                // 6
  {0, 0, 0, 0, 0, 0, 0}                 // BLANK
};

int switchPin = 9;
int blank = 6;

void setup()
{
  for (int i = 0; i < 7; i++)
  {
    pinMode(ledPins[i], OUTPUT);
    digitalWrite(ledPins[i], LOW);
  }
  randomSeed(analogRead(0));
}


void loop()
{
```

(continued)

LISTING PROJECT 9 (continued)

```
  if (digitalRead(switchPin))
  {
    rollTheDice();
  }
  delay(100);
}

void rollTheDice()
{
  int result = 0;
  int lengthOfRoll = random(15, 25);
  for (int i = 0; i < lengthOfRoll; i++)
  {
    result = random(0, 6);            // result will be 0 to 5 not 1 to 6
    show(result);
    delay(50 + i * 10);
  }
  for (int j = 0; j < 3; j++)
  {
    show(blank);
    delay(500);
    show(result);
    delay(500);
  }
}

void show(int result)
{

 for (int i = 0; i < 7; i++)
 {
   digitalWrite(ledPins[i], dicePatterns[result][i]);
 }
}
```

We now have seven LEDs to initialize in the setup method, so it is worth putting them in an array and looping over the array to initialize each pin. We also have a call to randomSeed in the setup method. If this was not there, every time we reset the board, we would end up with the same sequence of dice throws. As an experiment, you may wish to try commenting out this line by placing a // in front of it and verifying this. In fact, as an Evil Genius, you may like to omit that line so that you can cheat at Snakes and Ladders!

The dicePatterns array determines which LEDs should be on or off for any particular throw. So each throw element of the array is actually itself an array of seven elements, each one being either HIGH or LOW (1 or 0). When we come to display a particular result of throwing the dice, we can just

loop over the array for the throw, setting each LED accordingly.

Putting It All Together

Load the completed sketch for Project 9 from your Arduino Sketchbook and download it to the board (see Chapter 1).

Summary

In this chapter we have used a variety of LEDs and software techniques for lighting them in interesting ways. In the next chapter we will investigate some different types of sensors and use them to provide inputs to our projects.

This page intentionally left blank

CHAPTER 5

Sensor Projects

Sensors turn real-world measurements into electronic signals that we can then use on our Arduino boards. The projects in this chapter are all about using light and temperature.

We also look at how to interface with keypads and rotary encoders.

Project 10
Keypad Security Code

This project would not be out of place in the lair of any Evil Genius worth their salt. A secret code must be entered on the keypad, and if it is correct, a green LED will light; otherwise, a red LED will stay lit. In Project 27, we will revisit this project and show how it cannot just show the appropriate light, but also control a door lock.

COMPONENTS AND EQUIPMENT

	Description	Appendix
	Arduino Diecimila or Duemilanove board or clone	1
D1	Red 5-mm LED	23
D2	Green 5-mm LED	25
R1-2	270 Ω 0.5W metal film resistor	6
K1	4 by 3 keypad	54
	0.1-inch header strip	55

Unfortunately, keypads do not usually have pins attached, so we will have to attach some, and the only way to do that is to solder them on. So this is another of our projects where you will have to do a little soldering.

Hardware

The schematic diagram for Project 10 is shown in Figure 5-1. By now, you will be used to LEDs; the new component is the keypad.

Keypads are normally arranged in a grid so that when one of the keys is pressed, it connects a row to a column. Figure 5-2 shows a typical arrangement for a 12-key keyboard with numbers from 0 to 9 and * and # keys.

The key switches are arranged at the intersection of row-and-column wires. When a key is pressed, it connects a particular row to a particular column.

By arranging the keys in a grid like this, it means that we only need to use 7 (4 rows + 3 columns) of our digital pins rather than 12 (one for each key).

However, it also means that we have to do a bit more work in the software to determine which keys are pressed. The basic approach we have to take is to connect each row to a digital output and each column to a digital input. We then put each output high in turn and see which inputs are high.

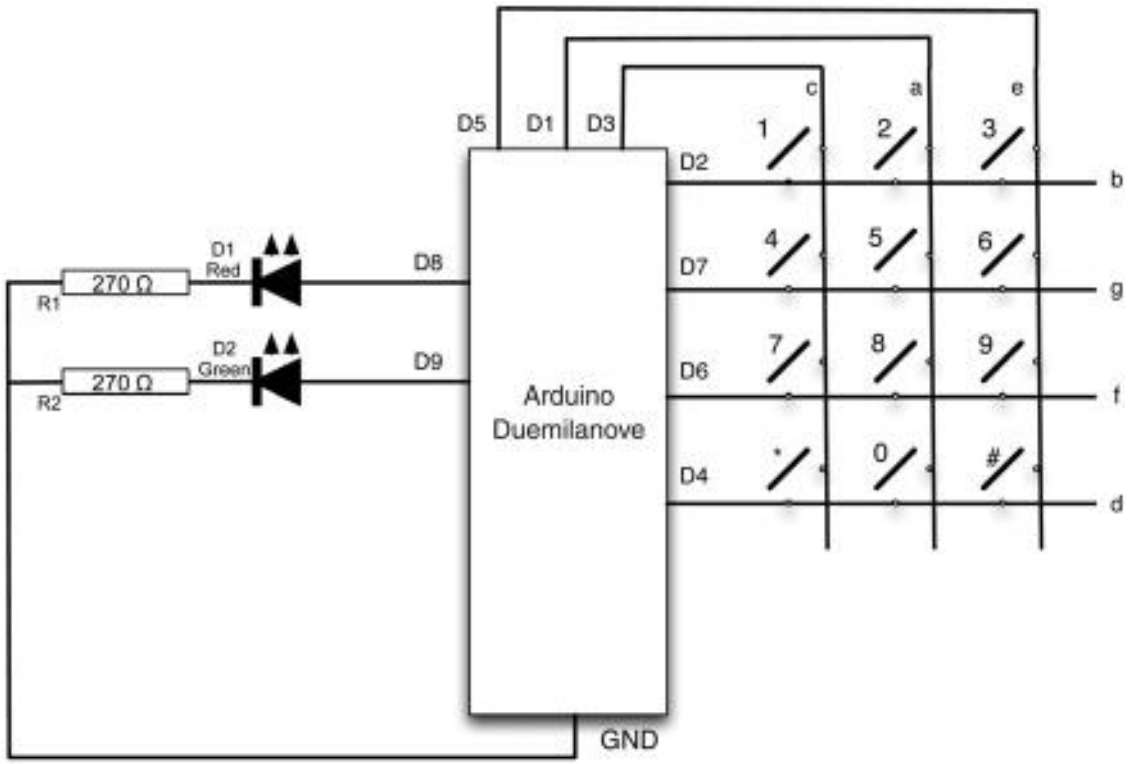


Figure 5-1 Schematic diagram for Project 10.

Figure 5-3 shows how you can solder seven pins from a pin header strip onto the keypad so that you can then connect it to the breadboard. Pin headers are bought in strips and can be easily snapped to provide the number of pins required.

Now, we just need to find out which pin on the keypad corresponds to which row or column. If we are lucky, the keypad will come with a datasheet that tells us this. If not, we will have to do some detective work with a multimeter. Set the multimeter to continuity so that it beeps when you connect the leads together. Then get some paper,

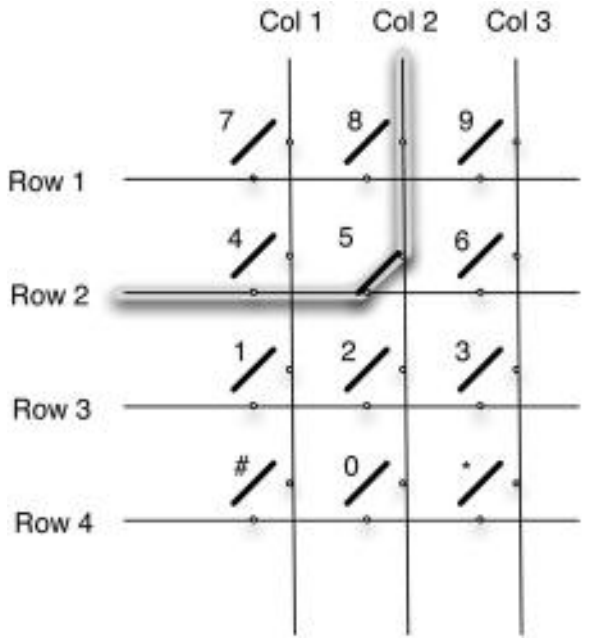


Figure 5-2 A 12-switch keypad.

Figure 5-3 Soldering pins to the keypad.

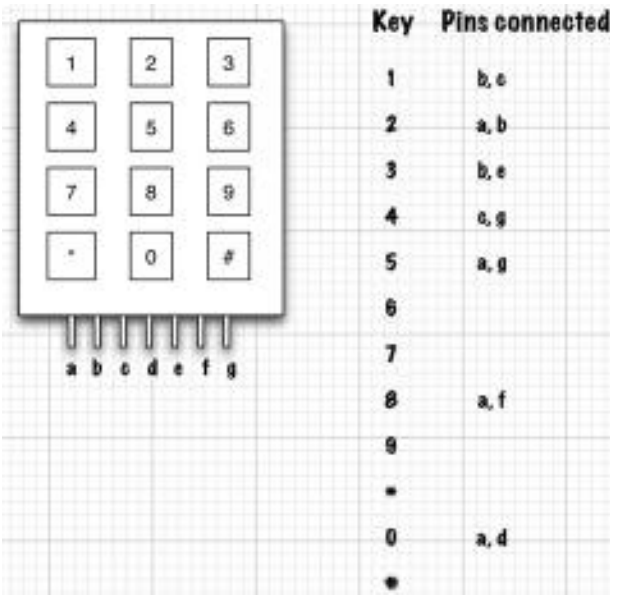


Figure 5-4 Working out the keypad connections.

draw a diagram of the keyboard connections, and label each pin with a letter from a to g. Then write a list of all the keys. Then, holding each key down in turn, find the pair of pins that make the multimeter beep indicating a connection (Figure 5-4). Release the key to check that you have indeed found the correct pair. After a while, a pattern will emerge and you will be able to see how the pins relate to rows and columns. Figure 5-4 shows the arrangement for the keypad used by the author.

The completed breadboard layout is shown in Figure 5-5. Note that the keypad conveniently has seven pins that will just fit directly into the Digital Pin 0 to 7 socket on the Arduino board (Figure 5-6), so we only need the breadboard for the two LEDs.

You may have noticed that digital pins 0 and 1 have TX and RX next to them. This is because they are also used by the Arduino board for serial communications, including the USB connection. In this case, we are not using digital pin 0, but we have connected digital pin 1 to the keyboard's middle column. We will still be able to program the board, but it does mean that we will not be able to communicate over the USB connection while the sketch is running. Since we do not want to do this anyway, this is not a problem.

Software

While we could just write a sketch that turns on the output for each row in turn and reads the inputs to get the coordinates of any key pressed, it is a bit more complex than that because switches do not always behave in a good way when you press them. Keypads and push switches are likely to bounce. That is, when you press them, they do not simply go from being opened to closed, but may open and close several times as part of pressing the button.

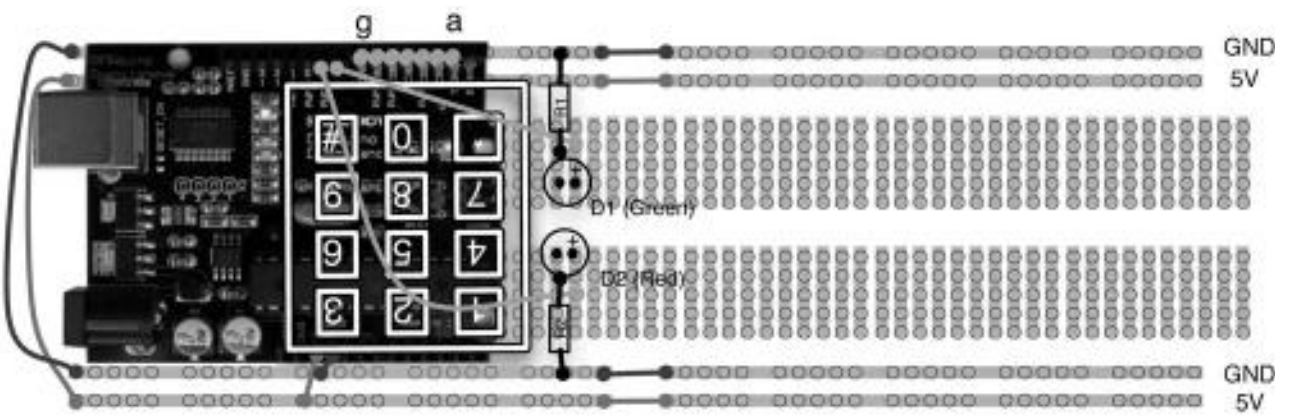


Figure 5-5 Project 10 breadboard layout.

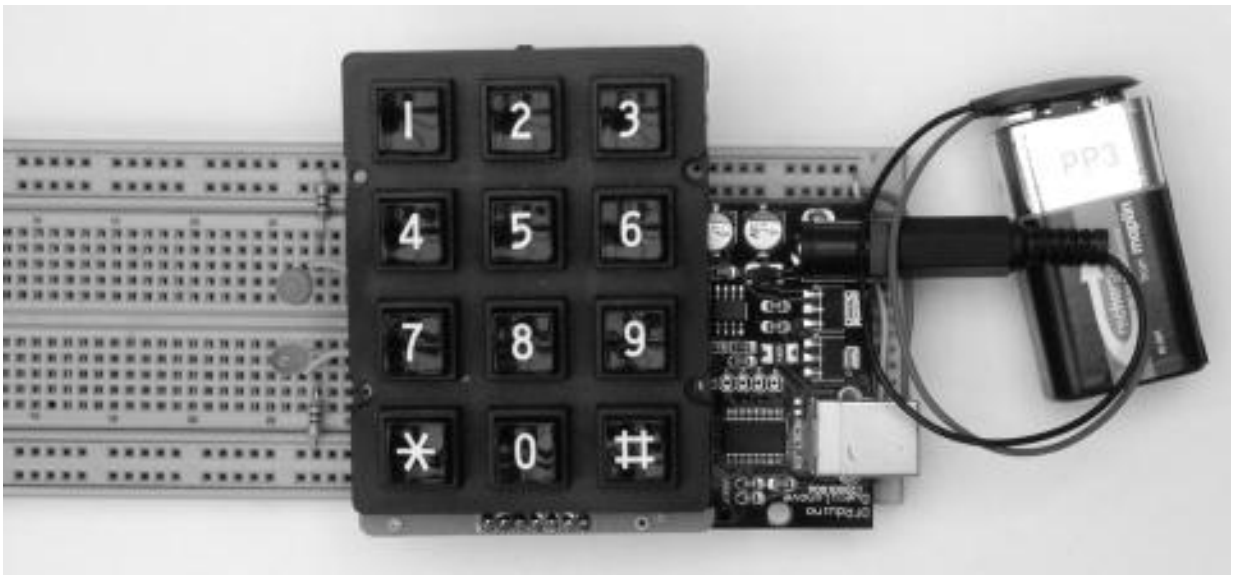


Figure 5-6 Project 10. Keypad security code.

Fortunately for us, Mark Stanley and Alexander Brevig have created a library that you can use to connect to keypads that handles such things for us. This is a good opportunity to demonstrate installing a library into the Arduino software.

In addition to the libraries that come with the Arduino, many people have developed their own libraries and published them for the benefit of the Arduino community. The Evil Genius is much amused by such altruism and sees it as a great weakness. However, the Evil Genius is not above using such libraries for their own devious ends.

To make use of this library, we must first download it from the Arduino website at this address: www.arduino.cc/playground/Code/Keypad.

Download the file Keypad.zip and unzip it. If you are using Windows, you right-click and choose Extract All and then save the whole folder into C:\Program Files\Arduino\Arduino-0017\hardware\libraries (Figure 5-7).

On LINUX, find the Arduino installation directory and copy the folder into hardware/libraries.

On a Mac, you do not put the new library into the Arduino installation. Instead, you create a folder called libraries in Documents/Arduino (Figure 5-8) and put the whole library folder in there. Incidentally, the Documents/Arduino directory is also the default location where your sketches are stored.

Once we have installed this library into our Arduino directory, we will be able to use it with any sketches that we write. But remember that on Windows and LINUX, if you upgrade to a newer version of the Arduino software, you will have to reinstall the libraries that you use.

You can check that the library is correctly installed by restarting the Arduino, starting a new sketch, and choosing the menu option Sketch | Import Library | Keypad. This will then insert the text "#include <Keypad.h>" into the top of the file.

The sketch for the application is shown in Listing Project 10. Note that you may well have to change your keys, rowPins, and colPins arrays so that they agree with the key layout of your keypad, as we discussed in the hardware section.

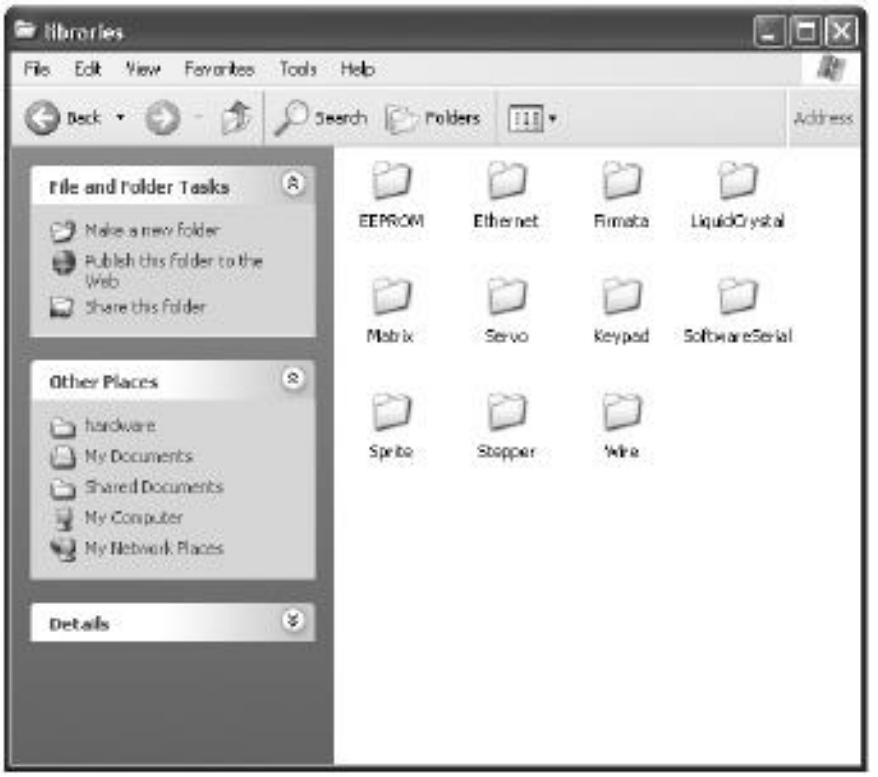


Figure 5-7 Installing the library for Windows.

Figure 5-8 Installing the library for Mac.

LISTING PROJECT 10

```
#include <Keypad.h>

char* secretCode = "1234";
int position = 0;

const byte rows = 4;
const byte cols = 3;
char keys[rows][cols] = {
  {'1','2','3'},
  {'4','5','6'},
  {'7','8','9'},
  {'*','0','#'}
};
byte rowPins[rows] = {2, 7, 6, 4};
byte colPins[cols] = {3, 1, 5};
Keypad keypad = Keypad(makeKeymap(keys), rowPins, colPins, rows, cols);

int redPin = 9;
int greenPin = 8;

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

void loop()
{
  char key = keypad.getKey();
  if (key == '*' || key == '#')
  {
    position = 0;
    setLocked(true);
  }
  if (key == secretCode[position])
  {
    position ++;
  }
  if (position == 4)
  {
    setLocked(false);
  }
  delay(100);
}

void setLocked(int locked)
```

LISTING PROJECT 10 (continued)

```
{
  if (locked)
  {
    digitalWrite(redPin, HIGH);
    digitalWrite(greenPin, LOW);
  }
  else
  {
    digitalWrite(redPin, LOW);
    digitalWrite(greenPin, HIGH);
  }
}
```

This sketch is quite straightforward. The loop function checks for a key press. If the key pressed is a # or a * it sets the position variable back to 0. If, on the other hand, the key pressed is one of the numerals, it checks to see if it is the next key expected (secretCode[position]) is the key just pressed, and if it is, it increments position by one. Finally, the loop checks to see if position is 4, and if it is, it sets the LEDs to their unlocked state.

Putting It All Together

Load the completed sketch for Project 10 from your Arduino Sketchbook and download it to the board (see Chapter 1).

If you have trouble getting this to work, it is most likely a problem with the pin layout on your keypad. So persevere with the multimeter to map out the pin connections.

Rotary Encoders

We have already met variable resistors: as you turn the knob, the resistance changes. These used to be behind most knobs that you could twiddle on electronic equipment. There is an alternative, the rotary encoder, and if you own some consumer electronics where you can turn the knob round and round indefinitely without meeting any kind of end stop, there is probably a rotary encoder behind the knob.

Some rotary encoders also incorporate a button so that you can turn the knob and then press. This is a particularly useful way of making a selection from a menu when used with a liquid crystal display (LCD) screen.

A rotary encoder is a digital device that has two outputs (A and B), and as you turn the knob, you get a change in the outputs that can tell you whether the knob has been turned clockwise or counterclockwise.

Figure 5-9 shows how the signals change on A and B when the encoder is turned. When rotating clockwise, the pulses will change, as they would moving left to right on the diagram; when moving counterclockwise, the pulses would be moving right to left on the diagram.

So if A is low and B is low, and then B becomes high (going from phase 1 to phase 2), that would indicate that we have turned the knob clockwise. A clockwise turn would also be indicated by A being low, B being high, and then A becoming high (going from phase 2 to phase 3), etc. However, if A was high and B was low and then B went high, we have moved from phase 4 to phase 3 and are, therefore, turning counterclockwise.

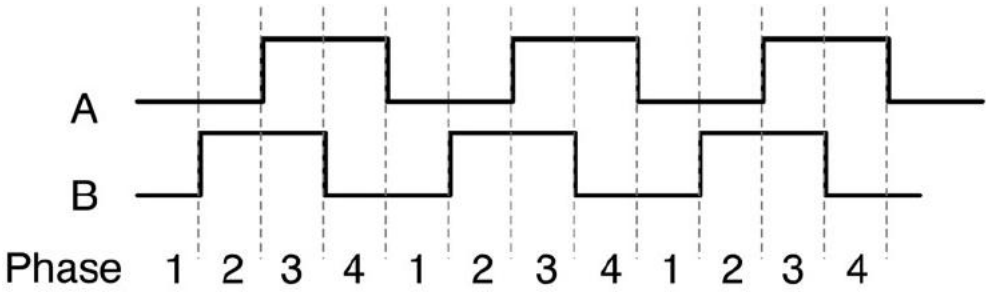


Figure 5-9 Pulses from a rotary encoder.

Project 11
Model Traffic Signal Using a Rotary Encoder

This project uses a rotary encoder with a built-in push switch to control the sequence of the traffic signals, and is based on Project 5. It is a much more realistic version of a traffic signal controller and is really not far off the logic that you would find in a real traffic signal controller.

Rotating the rotary encoder will change the frequency of the light sequencing. Pressing the button will test the lights, turning them all on at the same time, while the button is pressed.

The components are the same as for Project 5, with the addition of the rotary encoder and pull-up resistors in place of the original push switch.

COMPONENTS AND EQUIPMENT

	Description	Appendix
	Arduino Diecimila or Duemilanove board or clone	1
D1	5-mm Red LED	23
D2	5-mm Yellow LED	24
D3	5-mm Green LED	25
R1-R3	270 Ω 0.5W metal film resistor	6
R4-R6	100 KΩ 0.5W metal film resistor	13
S1	Rotary encoder with push switch	57

Hardware

The schematic diagram for Project 11 is shown in Figure 5-10. The majority of the circuitry is the same as for Project 5, except that now we have a rotary encoder.

The rotary encoder works just as if there were three switches: one each for A and B and one for the push switch. Each of these switches requires a pull-down resistor.

Since the schematic is much the same as for Project 5, it will not be much of a surprise to see that the breadboard layout (Figure 5-11) is similar to the one for that project.

Software

The starting point for the sketch is the sketch for Project 5. We have added code to read the encoder and to respond to the button press by turning all the LEDs on. We have also taken the opportunity to enhance the logic behind the lights to make them behave in a more realistic way, changing automatically. In Project 5, when you hold down the button, the lights change sequence roughly once per second. In a real traffic signal, the lights stay green and red a lot longer than they are yellow. So our sketch now has two periods: shortPeriod, which does not alter but is used when the lights are changing, and longPeriod, which determines how long they are illuminated for when green or red. This longPeriod is the period that is changed by turning the rotary encoder.

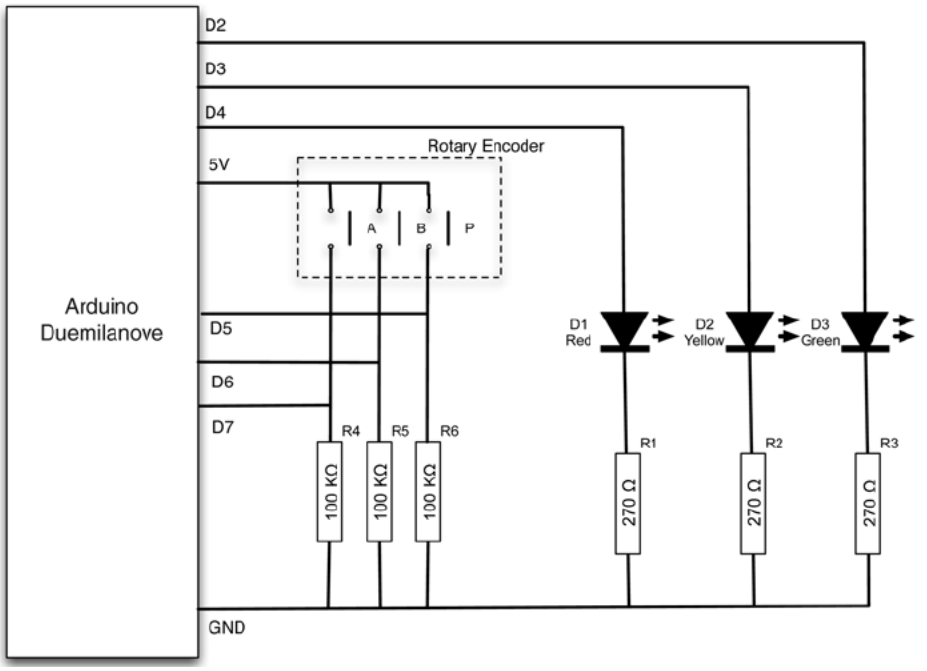


Figure 5-10 Schematic diagram for Project 11.

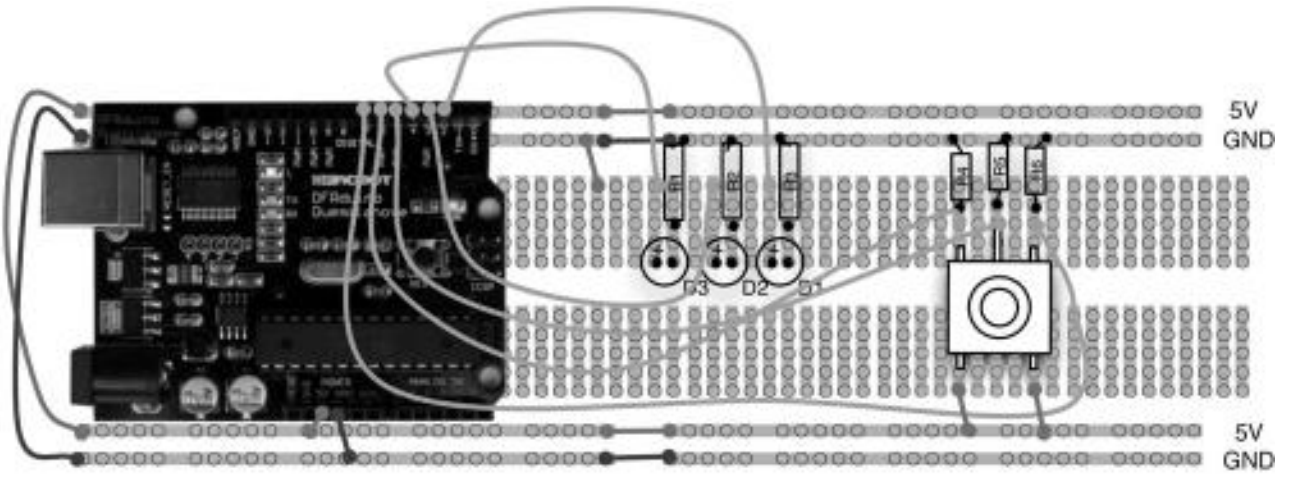


Figure 5-11 Breadboard layout for Project 11.

The key to handling the rotary encoder lies in the function getEncoderTurn. Every time this is called, it compares the previous state of A and B with their current state and if something has changed, works out if it was clockwise or counterclockwise and returns a –1 or 1, respectively. If there is no change (the knob has not been turned), it returns 0. This function must be called frequently or turning the rotary controller quickly will result in some changes not being recognized correctly.

If you want to use a rotary encoder for other projects, you can just copy this function. The function uses the static modifier for the oldA and oldB variables. This is a useful technique that allows the function to retain the values between one call of the function and the next, where normally it would reset the value of the variable every time the function is called.

LISTING PROJECT 11

```
int redPin = 2;
int yellowPin = 3;
int greenPin = 4;
int aPin = 6;
int bPin = 7;
int buttonPin = 5;

int state = 0;
int longPeriod = 5000;       // Time at green or red
int shortPeriod = 700;       // Time period when changing
int targetCount = shortPeriod;
int count = 0;

void setup()
{
  pinMode(aPin, INPUT);
  pinMode(bPin, INPUT);
  pinMode(buttonPin, INPUT);
  pinMode(redPin, OUTPUT);
  pinMode(yellowPin, OUTPUT);
  pinMode(greenPin, OUTPUT);
}

void loop()
{
  count++;
  if (digitalRead(buttonPin))
  {
    setLights(HIGH, HIGH, HIGH);
  }
  else
  {
    int change = getEncoderTurn();
    int newPeriod = longPeriod + (change * 1000);
    if (newPeriod >= 1000 && newPeriod <= 10000)
    {
      longPeriod = newPeriod;
    }
    if (count > targetCount)
    {
      setState();
      count = 0;
    }
  }
  delay(1);
}

int getEncoderTurn()
{
  // return -1, 0, or +1
  static int oldA = LOW;
```

LISTING PROJECT 11 (continued)

```
  static int oldB = LOW;
  int result = 0;
  int newA = digitalRead(aPin);
  int newB = digitalRead(bPin);
  if (newA != oldA || newB != oldB)
  {
    // something has changed
    if (oldA == LOW && newA == HIGH)
    {
      result = -(oldB * 2 - 1);
    }
  }
  oldA = newA;
  oldB = newB;
  return result;
}

int setState()
  {
    if (state == 0)
    {
      setLights(HIGH, LOW, LOW);
      targetCount = longPeriod;
      state = 1;
    }
    else if (state == 1)
    {
      setLights(HIGH, HIGH, LOW);
      targetCount = shortPeriod;
      state = 2;
    }
    else if (state == 2)
    {
      setLights(LOW, LOW, HIGH);
      targetCount = longPeriod;
      state = 3;
    }
    else if (state == 3)
    {
      setLights(LOW, HIGH, LOW);
      targetCount = shortPeriod;
      state = 0;
    }
  }

void setLights(int red, int yellow, int green)
{
  digitalWrite(redPin, red);
  digitalWrite(yellowPin, yellow);
  digitalWrite(greenPin, green);
}
```

This sketch illustrates a useful technique that lets you time events (turning an LED on for so many seconds) while at the same time checking the rotary encoder and button to see if they have been turned or pressed. If we just used the Arduino delay function with, say, 20,000, for 20 seconds, we would not be able to check the rotary encoder or switch in that period.

So what we do is use a very short delay (1 millisecond) but maintain a count that is incremented each time round the loop. Thus, if we want to delay for 20 seconds, we stop when the count has reached 20,000. This is less accurate than a single call to the delay function because the 1 millisecond is actually 1 millisecond plus the processing time for the other things that are done inside the loop.

Putting It All Together

Load the completed sketch for Project 11 from your Arduino Sketchbook and download it to the board (see Chapter 1).

You can press the rotary encoder button to test the LEDs and turn the rotary encoder to change how long the signal stays green and red.

Sensing Light

A common and easy-to-use device for measuring light intensity is the light-dependent resistor or LDR. They are also sometimes called photoresistors.

The brighter the light falling on the surface of the LDR, the lower the resistance. A typical LDR will have a dark resistance of up to 2 MΩ and a resistance when illuminated in bright daylight of perhaps 20 KΩ.

We can convert this change in resistance to a change in voltage by using the LDR, with a fixed resistor as a voltage divider, connected to one of our analog inputs. This schematic for this is shown in Figure 5-12.

With a fixed resistor of 100K, we can do some rough calculations about the voltage range to expect at the analog input.

In darkness, the LDR will have a resistance of 2 MΩ, so with a fixed resistor of 100K, there will be about a 20:1 ratio of voltage, with most of that voltage across the LDR, so that would mean about 4V across the LDR and 1V at the analog pin.

On the other hand, if the LDR is in bright light, its resistance might fall to 20 KΩ. The ratio of voltages would then be about 4:1 in favor of the fixed resistor, giving a voltage at the analog input of about 4V.

A more sensitive photo detector is the phototransistor. This functions like an ordinary transistor except there is not usually a base connection. Instead, the collector current is controlled by the amount of light falling on the phototransistor.

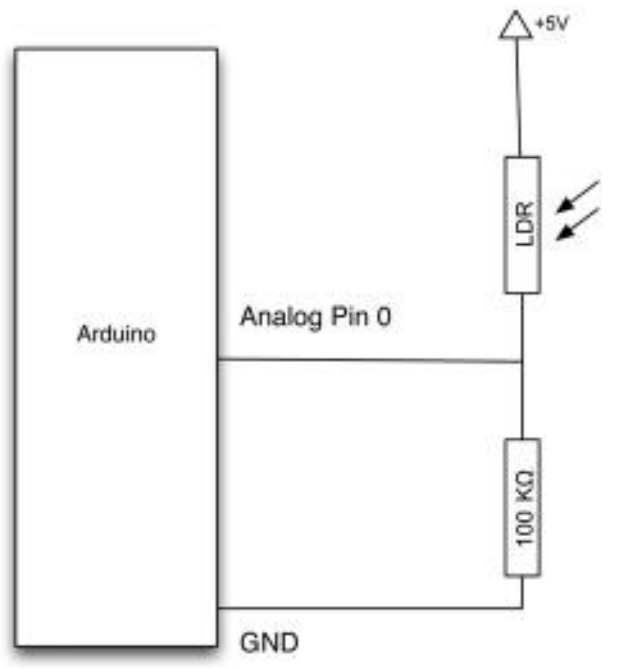


Figure 5-12 Using an LDR to measure light.

Project 12
Pulse Rate Monitor

This project uses an ultra-bright infrared (IR) LED and a phototransistor to detect the pulse in your finger. It then flashes a red LED in time with your pulse.

COMPONENTS AND EQUIPMENT

	Description	Appendix
	Arduino Diecimila or Duemilanove board or clone	1
D1	5-mm red LED	23
D2	5-mm IR LED sender 940 nm	26
R1	56 KΩ 0.5W metal film resistor	12
R2	270 Ω 0.5W metal film resistor	6
R4	39 Ω 0.5W metal film resistor	4
T1	IR phototransistor (same wavelength as D2)	36

Hardware

The pulse monitor works as follows: Shine the bright LED onto one side of your finger while the phototransistor on the other side of your finger picks up the amount of transmitted light. The resistance of the phototransistor will vary slightly as the blood pulses through your finger.

The schematic for this is shown in Figure 5-13 and the breadboard layout in Figure 5-15. We have chosen quite a high value of resistance for R1 because most of the light passing through the finger will be absorbed and we want the phototransistor to be quite sensitive. You may need to experiment with the value of the resistor to get the best results.

It is important to shield the phototransistor from as much stray light as possible. This is particularly important for domestic lights that actually fluctuate at 50Hz or 60Hz and will add a considerable amount of noise to our weak heart signal.

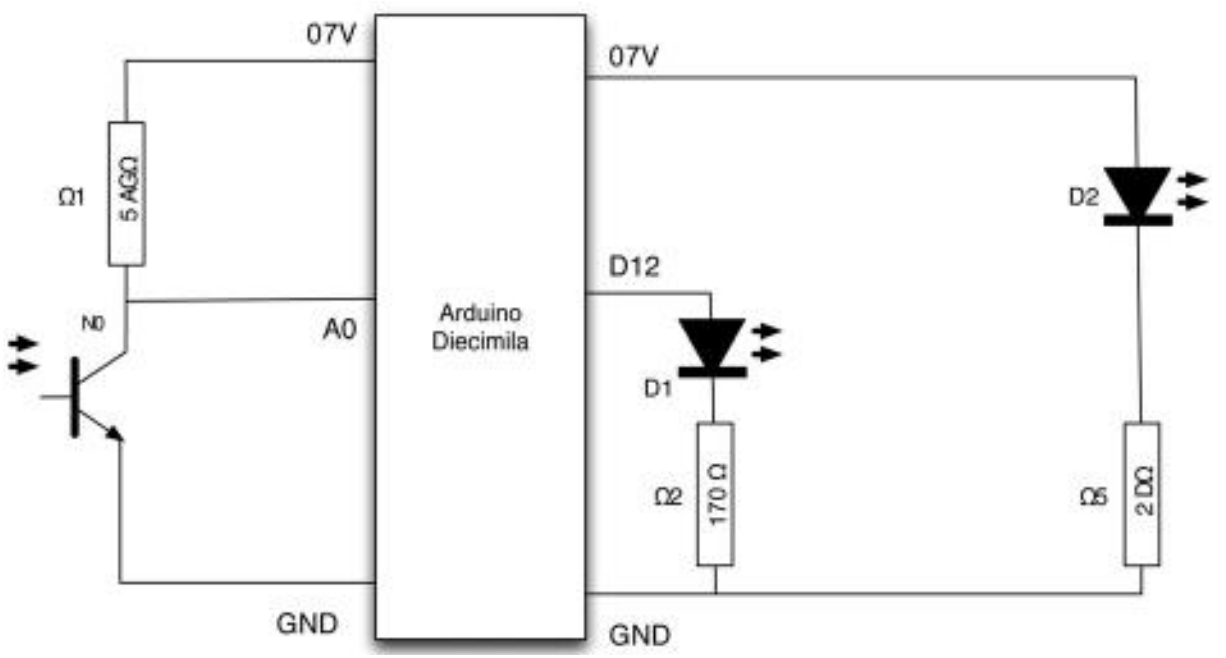


Figure 5-13 Schematic for Project 12.

Figure 5-14 Sensor tube for heart monitor.

For this reason, the phototransistor and LED are built into a tube or corrugated cardboard held together with duct tape. The construction of this is shown in Figure 5-14.

Two 5-mm holes are drilled opposite each other in the tube and the LED inserted into one side and the phototransistor into the other. Short leads are soldered to the LED and phototransistor, and then another layer of tape is wrapped over everything to hold it all in place. Be sure to check which colored wire is connected to which lead of the LED and phototransistor before you tape them up.

The breadboard layout for this project (Figure 5-15) is very straightforward.

The final "finger tube" can be seen in Figure 5-16.

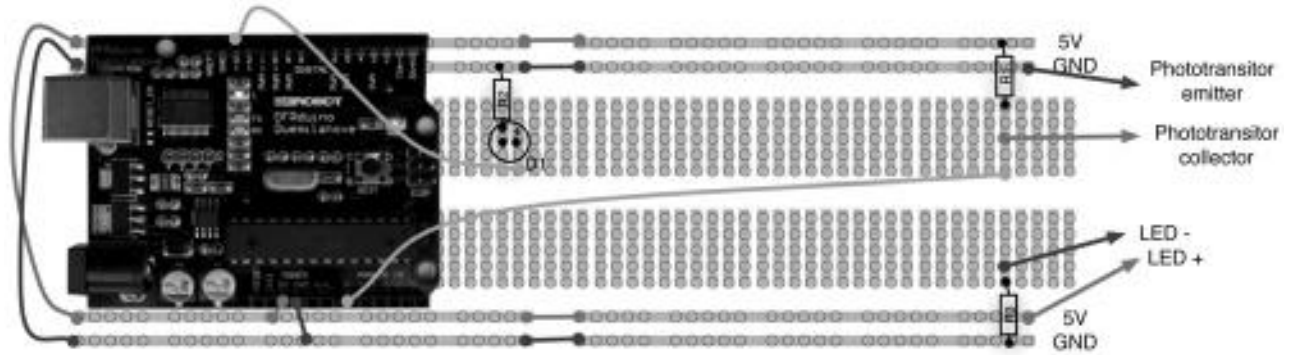


Figure 5-15 Breadboard layout for Project 12.

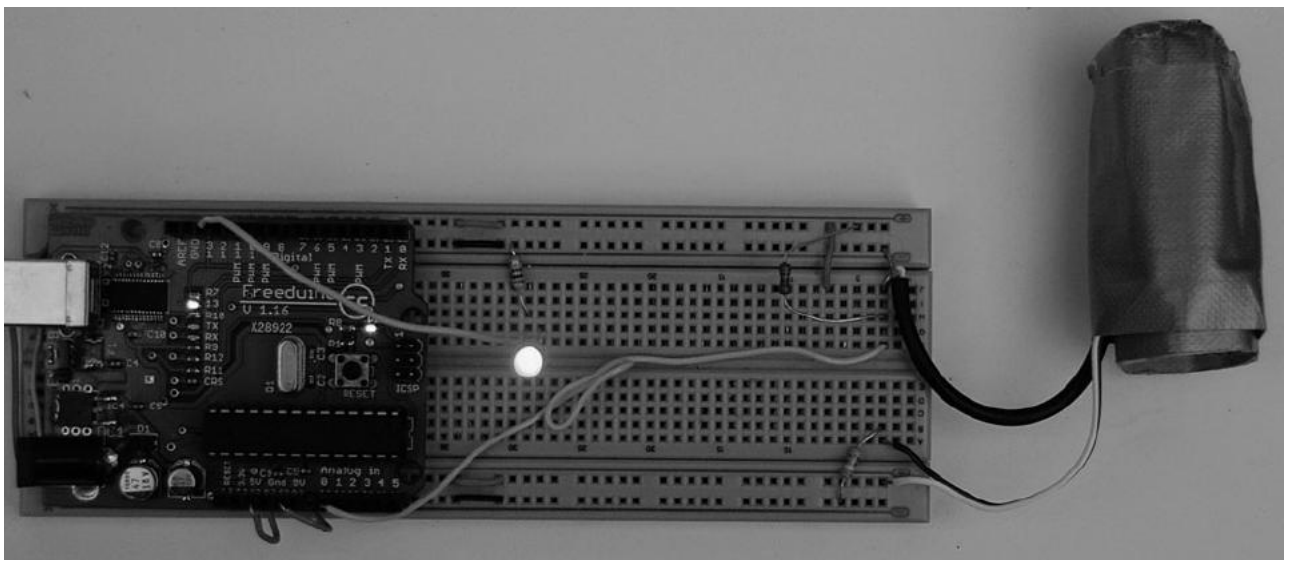


Figure 5-16 Project 12. Pulse rate monitor.

Software

The software for this project is quite tricky to get right. Indeed, the first step is not to run the entire final script, but rather a test script that will gather data that we can then paste into a spreadsheet and chart to test out the smoothing algorithm (more on this later).

The test script is provided in Listing Projet 12.

LISTING PROJECT 12—TEST SCRIPT

```
int ledPin = 13;
int sensorPin = 0;

double alpha = 0.75;
int period = 20;
double change = 0.0;

void setup()
{
  pinMode(ledPin, OUTPUT);
  Serial.begin(115200);
}

void loop()
{
    static double oldValue = 0;
    static double oldChange = 0;
    int rawValue =
   analogRead(sensorPin);
    double value = alpha * oldValue
       + (1 - alpha) * rawValue;

    Serial.print(rawValue);
    Serial.print(“,”);
    Serial.println(value);

    oldValue = value;
    delay(period);
}
```

This script reads the raw signal from the analog input and applies the smoothing function and then writes both values to the Serial Monitor, where we can capture them and paste them into a spreadsheet for analysis. Note that the Serial Monitor’s communications is set to its fastest rate to minimize the effects of the delays caused by sending the data. When you start the Serial Monitor, you will need to change the serial speed to 115200 baud.

The smoothing function uses a technique called “leaky integration,” and you can see in the code where we do this smoothing using the line:

```
double value = alpha * oldValue + (1 -
   alpha) * rawValue;
```

The variable alpha is a number greater than 0 but less than 1 and determines how much smoothing to do.

Put your finger into the sensor tube, start the Serial Monitor, and leave it running for three or four seconds to capture a few pulses.

Then, copy and paste the captured text into a spreadsheet. You will probably be asked for the column delimiter character, which is a comma. The resultant data and a line chart drawn from the two columns are shown in Figure 5-17.

The more jagged trace is from the raw data read from the analog port, and the smoother trace clearly has most of the noise removed. If the smoothed trace shows significant noise—in particular, any false peaks that will confuse the monitor—increase the level of smoothing by decreasing the value of alpha.

Once you have found the right value of alpha for your sensor arrangement, you can transfer this value into the real sketch and switch over to using the real sketch rather than the test sketch. The real sketch is provided in the following listing on the next page.

LISTING PROJECT 12

```
int ledPin = 13;
int sensorPin = 0;

double alpha = 0.75;
int period = 20;
double change = 0.0;

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

void loop()
{
     static double oldValue = 0;
     static double oldChange = 0;
     int rawValue =
    analogRead(sensorPin);
     double value = alpha * oldValue
        + (1 - alpha) * rawValue;
     change = value - oldValue;

     digitalWrite(ledPin, (change <
        0.0 && oldChange > 0.0));

     oldValue = value;
     oldChange = change;
     delay(period);
}
```

There now just remains the problem of detecting the peaks. Looking at Figure 5-17, we can see that if we keep track of the previous reading, we can see that the readings are gradually increasing until the change in reading flips over and becomes negative. So, if we lit the LED whenever the old change was positive but the new change was negative, we would get a brief pulse from the LED at the peak of each pulse.

Putting It All Together

Both the test and real sketch for Project 12 are in your Arduino Sketchbook. For instructions on downloading it to the board, see Chapter 1.

As mentioned, getting this project to work is a little tricky. You will probably find that you have to get your finger in just the right place to start getting a pulse. If you are having trouble, run the test script as described previously to check that your detector is getting a pulse and the smoothing factor alpha is low enough.

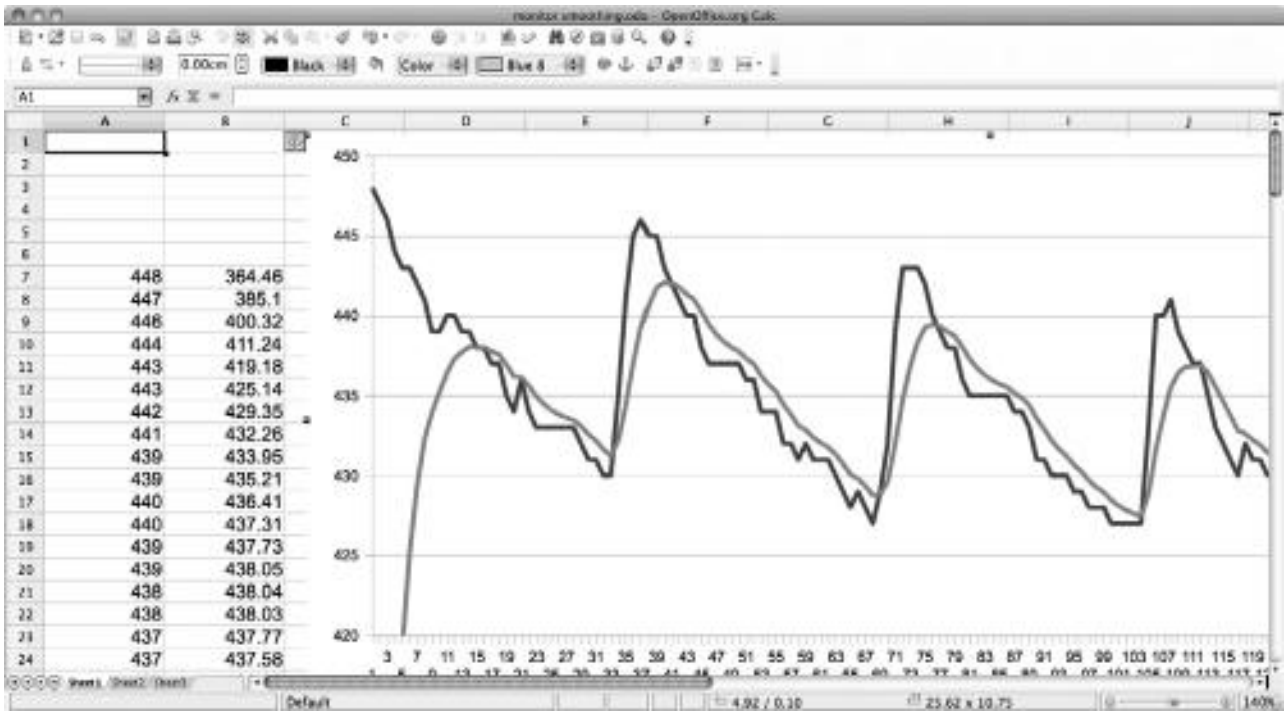


Figure 5-17 Heart monitor test data pasted into a spreadsheet.

The author would like to point out that this device should not be used for any kind of real medical application.

Measuring Temperature

Measuring temperature is a similar problem to measuring light intensity. Instead of an LDR, a device called a thermistor is used. As the temperature increases, so does the resistance of the thermistor.

When you buy a thermistor, it will have a stated resistance. In this case, the thermistor chosen is 33 KΩ. This will be the resistance of the device at 25°C.

The formula for calculating the resistance at a particular temperature is given by:

$$R = Ro \exp(-Beta/(T + 273) - Beta/(To + 273)$$

In this case, Ro is the resistance at 25ºC (33 KΩ) and beta is a constant value that you will find in the thermistor's datasheet. In this case, its value is 4090.

So,

$$R = Ro \exp(Beta/(T + 273) - Beta/298)$$

Rearranging this formula, we can get an expression for the temperature knowing the resistance.

$$R = Ro \exp(Beta/(T + 273) - Beta/298)$$

If we use a 33 KΩ fixed resistor, we can calculate the voltage at the analog input using the formula:

$$V = 5 * 33K/(R + 33K)$$

so

$$R = ((5 * 33K)/V) - 33K$$

Since the analog value "a" is given by:

$$a = V * (1023/5)$$

then

$$V = a/205$$

and

$$R = ((5 * 33K * 205)/a) - 33K$$

$$R = (1025 \times 33K/a) - 33K$$

We can rearrange our formula to give us a temperature from the analog input, reading "a" as:

$$T = Beta/(\ln(R/33) + (Beta/298)) - 273$$

and so finally we get:

$$T = Beta/(\ln(((1025 \times 33/a) - 33)/33) + (Beta/298)) - 273$$

Phew, that's a lot of math!

We will use this calculation in the next project to create a temperature logger.

Project 13
USB Temperature Logger

This project is controlled by your computer, but once given its logging instructions can be disconnected and run on batteries to do its logging. While logging, it stores its data, and then when the logger is reconnected it will transfer its data back over the USB connection, where it can be imported into a spreadsheet. By default, the logger will record 1 sample every five minutes, and can record up to 255 samples.

To instruct the temperature logger from your computer, we have to define some commands that can be issued from the computer. These are shown in Table 5-1.

TABLE 5-1 Temperature Logger Commands	
R	Read the data out of the logger as CSV text
X	Clear all data from the logger
C	Centigrade mode
F	Fahrenheit mode
1–9	Set the sample period in minutes from 1 to 9
G	Go! start logging temperatures
?	Reports the status of the device, number of samples taken, etc.

This project just requires a thermistor and resistor.

COMPONENTS AND EQUIPMENT

	Description	Appendix
	Arduino Diecimila or Duemilanove board or clone	1
R1	Thermistor, 33K at 25°C, beta 4090	18
R2	33 KΩ 0.5W metal film resistor	10

- If you cannot obtain a thermistor with the correct value of beta, or resistance, you can change these values in the sketch.

Hardware

The schematic diagram for Project 13 is shown in Figure 5-18.

This is so simple that we can simply fit the leads of the thermistor and resistor into the Arduino board, as shown in Figure 5-19.

Software

The software for this project is a little more complex than for some of our other projects (see Listing Project 13). All of the variables that we have used in our sketches so far are forgotten as soon as the Arduino board is reset or disconnected from the power. Sometimes we want to be able to store data persistently so that it is there next time we start up the board. This can be done by using the special type of memory on the Arduino called EEPROM, which stands for electrically erasable programmable read-only memory. The Arduino Duemilanove board has 1024 bytes of EEPROM.

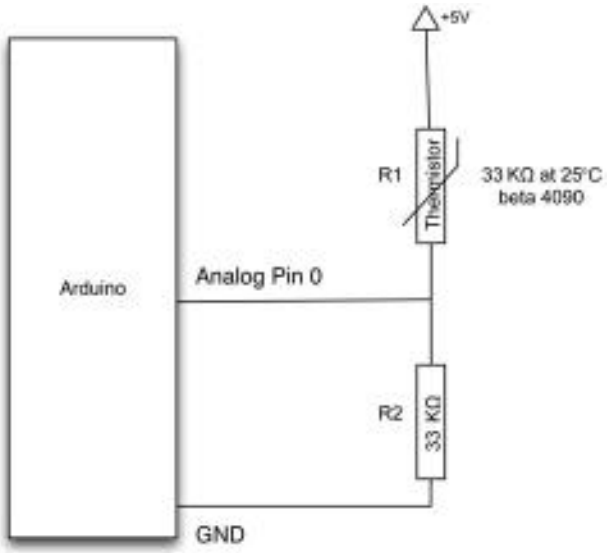


Figure 5-18 Schematic diagram for Project 13.

This is the first project where we have used the Arduino's EEPROM to store values so that they are not lost if the board is reset or disconnected from the power. This means that once we have set our data logging recording, we can disconnect it from the USB lead and leave it running on batteries. Even if the batteries go dead, our data will still be there the next time we connect it.

You will notice that at the top of this sketch we use the command #define for what in the past we would have used variables for. This is actually a more efficient way of defining constants—that is, values that will not change during the running of the sketch. So it is actually ideal for pin settings and constants like beta. The command #define is what is called a pre-processor directive, and what happens is that just before the sketch is compiled,

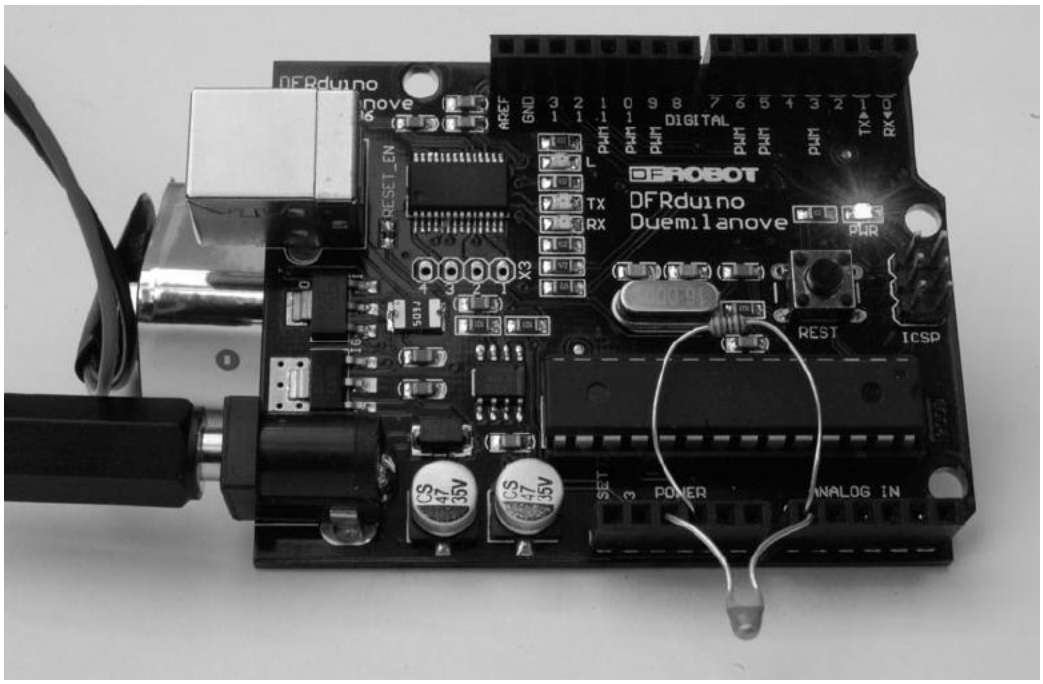


Figure 5-19 A powered-up Arduino board with LED lit.

LISTING PROJECT 13

```
#include <EEPROM.h>


#define ledPin 13
#define analogPin 0
#define maxReadings 255
#define beta 4090                    // from your thermistor's datasheet
#define resistance 33

float readings[maxReadings];
int lastReading = EEPROM.read(0);
boolean loggingOn = false;
long period = 300;
long count = 0;
char mode = 'C';

void setup()
{
   pinMode(ledPin, OUTPUT);
   Serial.begin(9600);
   Serial.println("Ready");
}

void loop()
{
```

(continued)

LISTING PROJECT 13 (continued)

```
  if (Serial.available())
  {
    char ch = Serial.read();
    if (ch == 'r' || ch == 'R')
    {
      sendBackdata();
    }
    else if (ch == 'x' || ch == 'X')
    {
      lastReading = 0;
      EEPROM.write(0, 0);
      Serial.println("Data cleared");
    }
    else if (ch == 'g' || ch == 'G')
    {
      loggingOn = true;
      Serial.println("Logging started");
    }
    else if (ch > '0' && ch <= '9')
    {
      setPeriod(ch);
    }
    else if (ch == 'c' or ch == 'C')
    {
      Serial.println("Mode set to deg C");
      mode = 'C';
    }
    else if (ch == 'f' or ch == 'F')
    {
      Serial.println("Mode set to deg F");
      mode = 'F';
    }
    else if (ch == '?')
    {
      reportStatus();
    }
  }
  if (loggingOn && count > period)
  {
    logReading();
    count = 0;
  }
  count++;
  delay(1000);
}

void sendBackdata()
{
   loggingOn = false;
   Serial.println("Logging stopped");
   Serial.println("------ cut here ---------");
```

LISTING PROJECT 13 (continued)

```
    Serial.print("Time (min)\tTemp (");
    Serial.print(mode);
    Serial.println(")");
    for (int i = 0; i < lastReading; i++)
    {
       Serial.print((period * i) / 60);
       Serial.print("\t");
       float temp = getReading(i);
       if (mode == 'F')
       {
         temp = (temp * 9) / 5 + 32;
       }
       Serial.println(temp);
    }
    Serial.println("------ cut here ---------");
}

void setPeriod(char ch)
{
    int periodMins = ch - '0';
    Serial.print("Sample period set to: ");
    Serial.print(periodMins);
    Serial.println(" mins");
    period = periodMins * 60;
}

void logReading()
{
  if (lastReading < maxReadings)
  {
    long a = analogRead(analogPin);
    float temp = beta / (log(((1025.0 * resistance / a) - 33.0) / 33.0) +
       (beta / 298.0)) - 273.0;
    storeReading(temp, lastReading);
    lastReading++;
  }
  else
  {
    Serial.println("Full! logging stopped");
    loggingOn = false;
  }
}

void storeReading(float reading, int index)
{
  EEPROM.write(0, (byte)index); // store the number of samples in byte 0
  byte compressedReading = (byte)((reading + 20.0) * 4);
  EEPROM.write(index + 1, compressedReading);
}
```

(continued)

LISTING PROJECT 13 (continued)

```
float getReading(int index)
{
  lastReading = EEPROM.read(0);
  byte compressedReading = EEPROM.read(index + 1);
  float uncompressesReading = (compressedReading / 4.0) - 20.0;
  return uncompressesReading;
}

void reportStatus()
{
 Serial.println("----------------");
 Serial.println("Status");
 Serial.print("Sample period\t");
 Serial.println(period / 60);
 Serial.print("Num readings\t");
 Serial.println(lastReading);
 Serial.print("Mode degrees\t");
 Serial.println(mode);
 Serial.println("----------------");
}
```

all occurrences of its name anywhere in the sketch are replaced by its value. It is very much a matter of personal taste whether you use #define or a variable.

Fortunately, reading and writing EEPROM happens just one byte at a time. So if we want to write a variable that is a byte or a char, we can just use the functions EEPROM.write and EEPROM.read, as shown in the example here:

```
char letterToWrite = 'A';
EEPROM.write(0, myLetter);

char letterToRead;
letterToRead = EEPROM.read(0);
```

The 0 in the parameters for read and write is the address in the EEPROM to use. This can be any number between 0 and 1023, with each address being a location where one byte is stored.

In this project we want to store both the position of the last reading taken (in the lastReading variable) and all the readings. So we will record lastReading in the first byte of EEPROM and then the actual reading data in the 256 bytes that follow.

Each temperature reading is kept in a float, and if you remember from Chapter 2, a float occupies 4 bytes of data. Here we had a choice: We could either store all 4 bytes or find a way to encode the temperature into a single byte. We decided to take the latter route, as it is easier to do.

The way we encode the temperature into a single byte is to make some assumptions about our temperatures. First, we assume that any temperature in Centigrade will be between –20 and +40. Anything higher or lower would likely damage our Arduino board anyway. Second, we assume that we only need to know the temperature to the nearest quarter of a degree.

With these two assumptions, we can take any temperature value we get from the analog input, add 20 to it, multiply it by 4, and still be sure that we always have a number between 0 and 240. Since a byte can hold a number between 0 and 255, that just fits nicely.

When we take our numbers out of EEPROM, we need to convert them back to a float, which we can do by reversing the process, dividing by 4 and then subtracting 20.

Both encoding and decoding the values are wrapped up in the functions storeReading and getReading. So, if we decided to take a different approach to storing the data, we would only have to change these two functions.

Putting It All Together

Load the completed sketch for Project 13 from your Arduino Sketchbook and download it to the board (see Chapter 1).

Now open the Serial Monitor (Figure 5-20), and for test purposes, we will set the temperature logger to log every minute by typing **1** in the Serial Monitor. The board should respond with the message "Sample period set to: 1 mins." If we wanted to, we could then change the mode to Fahrenheit by typing **F** into the Serial Monitor. Now we can check the status of the logger by typing **?**.

In order to unplug the USB cable, we need to have an alternative source of power, such as the battery lead we made back in Project 6. You need to have this plugged in and powered up at the same time as the USB connector is connected if you want the logger to keep logging after you disconnect the USB lead.

Finally, we can type the **G** command to start logging. We can then unplug the USB lead and leave our logger running on batteries. After waiting 10 or 15 minutes, we can plug it back in and see what data we have by opening the Serial Monitor and typing the **R** command, the results of which are shown in Figure 5-21. Select all the data, including the Time and Temp headings at the top.

Copy the text to the clipboard (press CTRL-C on Windows and LINUX, ALT-C on Macs), open a spreadsheet in a program such as Microsoft Excel, and paste it into a new spreadsheet (Figure 5-22).

Once in the spreadsheet, we can even draw a chart using our data.

Summary

We now know how to handle various types of sensors and input devices to go with our knowledge of LEDs. In the next section we will look at a number of projects that use light in various ways and get our hands on some more advanced display technologies, such as LCD text panels and seven-segment LEDs.

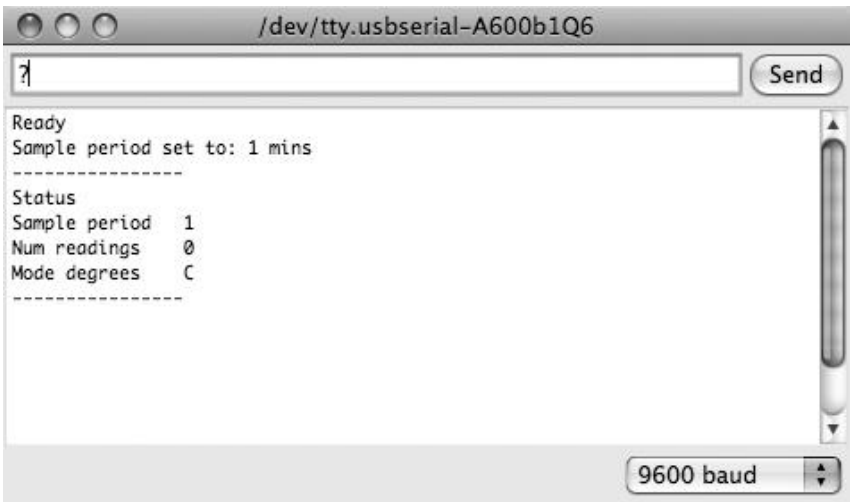


Figure 5-20 Issuing commands through the Serial Monitor.

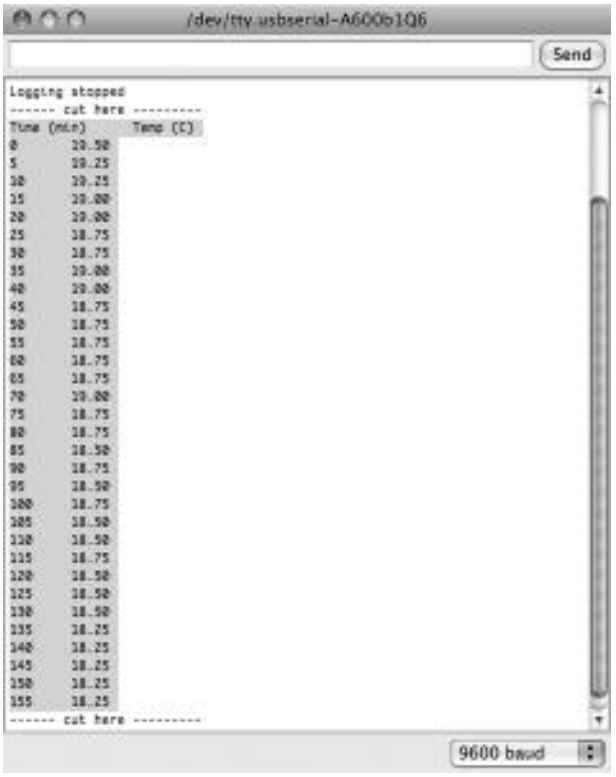


Figure 5-21 Data to copy and paste into a spreadsheet.

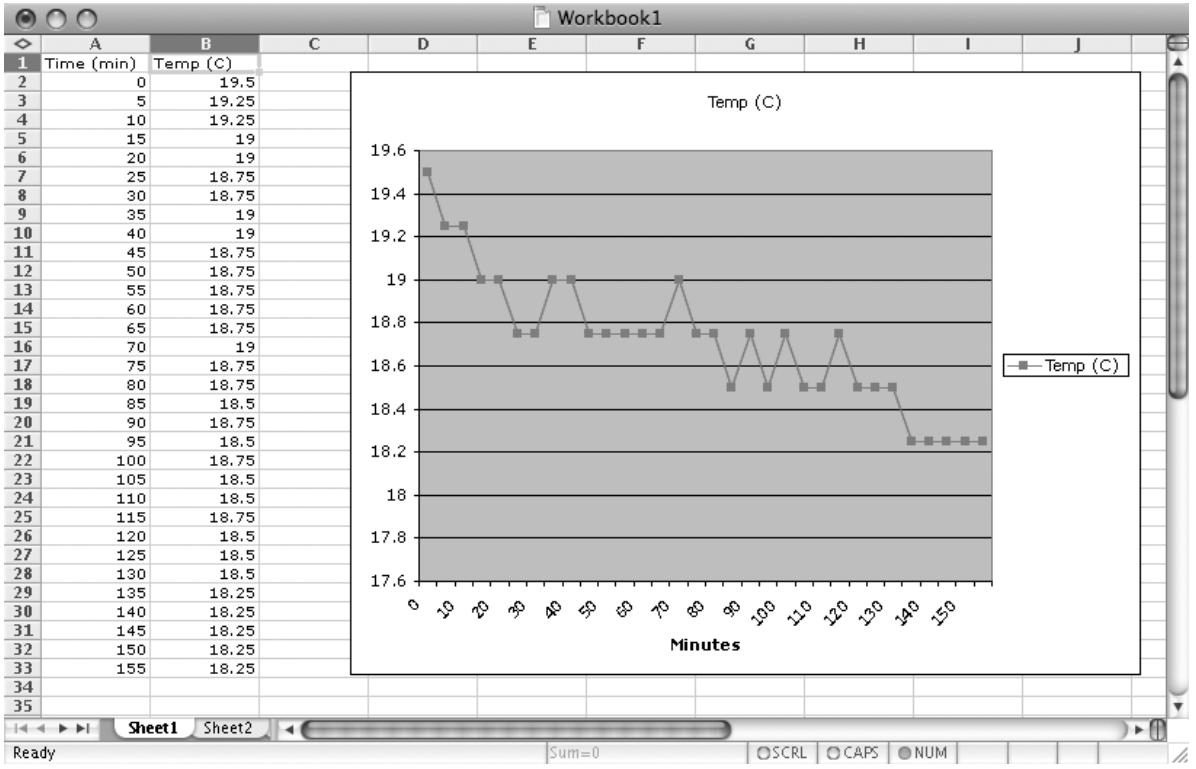


Figure 5-22 Temperature data imported into a spreadsheet.

CHAPTER 6

Light Projects

In this chapter, we look at some more projects based on lights and displays. In particular, we look at how to use multicolor LEDs, seven-segment LEDs, LED matrix displays, and LCD panels.

Project 14
Multicolor Light Display

This project uses a high-brightness, three-color LED in combination with a rotary encoder. Turning the rotary encoder changes the color displayed by the LED.

COMPONENTS AND EQUIPMENT

	Description	Appendix
	Arduino Diecimila or Duemilanove board or clone	1
D1	RGB LED (common anode)	31
R1-R3	100 Ω 0.5W metal film resistor	5
R4-6	100 KΩ 0.5W metal film resistor	13
S1	Rotary encoder with switch	57

The LED lamp is interesting because it has three LED lights in one four-pin package. The LED has a common anode arrangement, meaning that the positive connections of all three LEDs come out of one pin (pin 2).

If you cannot find a four-pin RGB (Red, Green, Blue) LED, you can use a six-pin device instead. Simply connect the separate anodes together, referring to the datasheet for the component.

Hardware

Figure 6-1 shows the schematic diagram for Project 14 and Figure 6-2 the breadboard layout.

It's a simple schematic diagram. The rotary encoder has pull-down resistors for the direction sensors and the push switch. For more detail on rotary encoders and how they work, see Chapter 5.

Each LED has its own series resistor to limit the current to about 30 mA per LED.

The LED package has a slight flatness to one side. Pin 1 is the pin closest to that edge. The other way to identify the pins is by length. Pin 2 is the common anode and is the longest pin.

The completed project is shown in Figure 6-3.

Each of the LEDs (red, green, and blue) is driven from a pulse width modulation (PWM) output of the Arduino board, so that by varying the output of each LED, we can produce a full spectrum of visible light colors.

The rotary encoder is connected in the same way as for Project 11: rotating it changes the color and pressing it will turn the LED on and off.

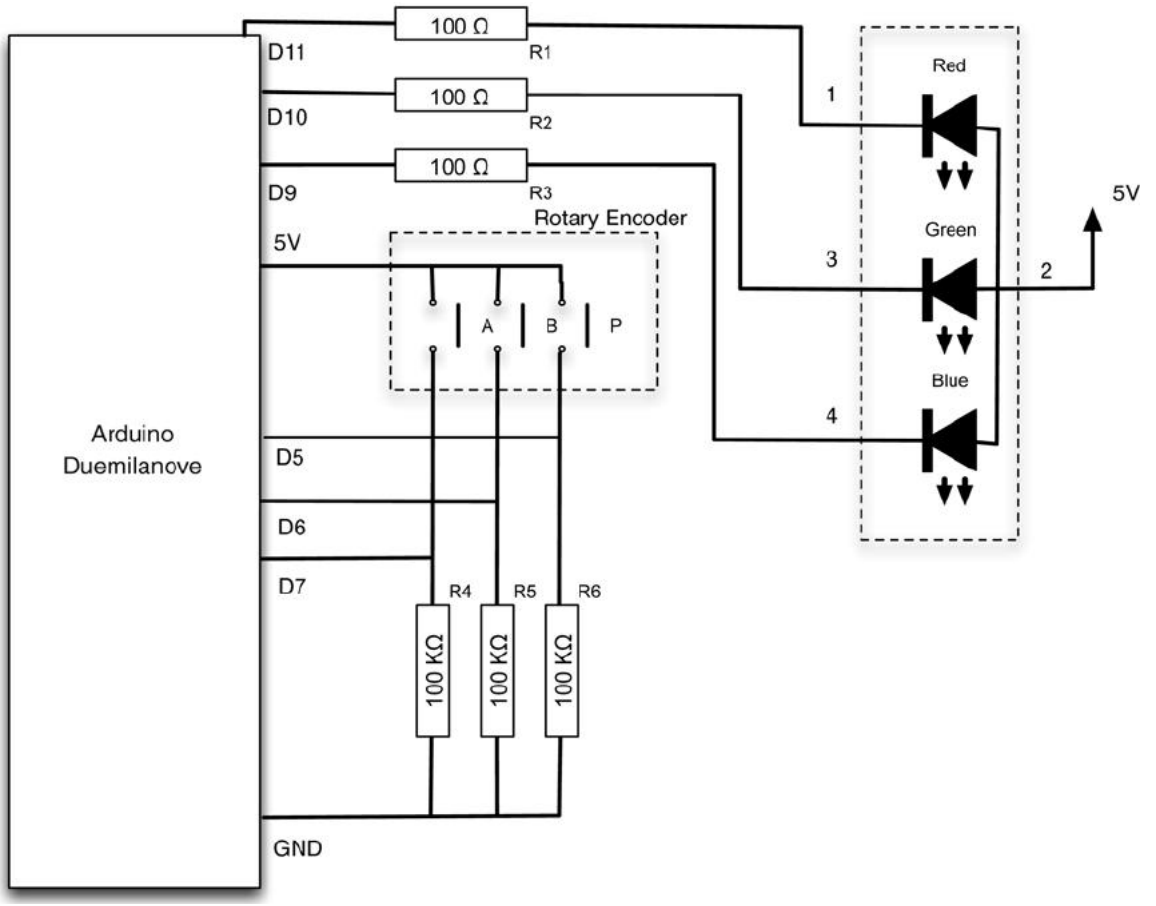


Figure 6-1 Schematic diagram for Project 14.

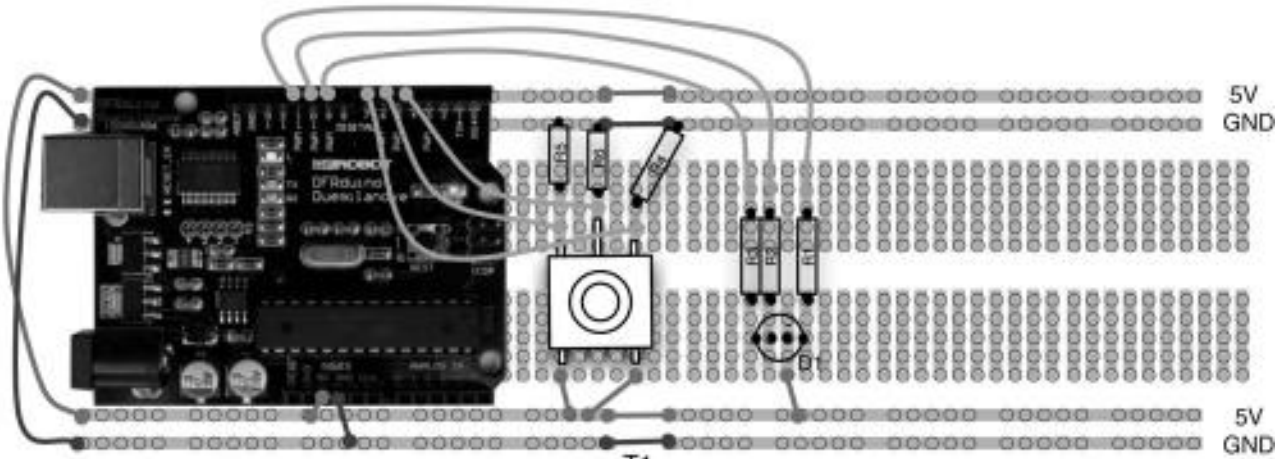


Figure 6-2 Breadboard layout for Project 14.

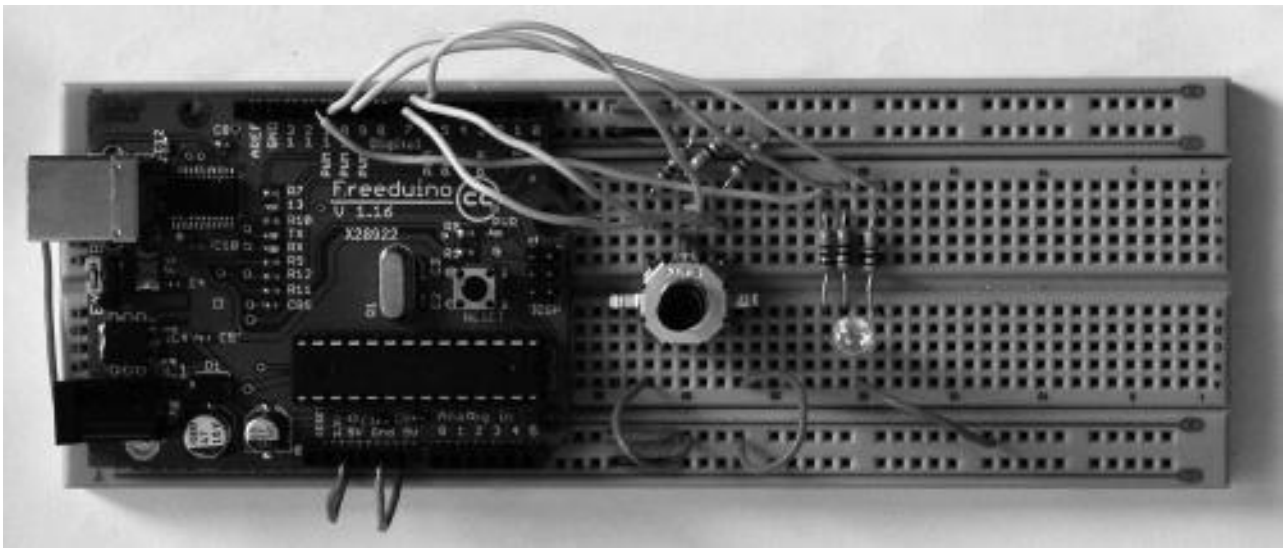


Figure 6-3 Project 14. Multicolor Light Display.

Software

This sketch (Listing Project 14) uses an array to represent the different colors that will be displayed by the LED. Each of the elements of the array is a long 32-bit number. Three of the bytes of the long number are used to represent the red, green, and blue components of the color, which correspond to how brightly each of the red, green, or blue LED elements should be lit. The numbers in the array are shown in hexadecimal and correspond to the hex number format used to represent 24-bit colors on webpages. If there is a particular color that you want to try and create, find yourself a web color chart by typing **web color chart** into your favorite search engine. You can then look up the hex value for the color that you want.

LISTING PROJECT 14

```
int redPin = 9;
int greenPin = 10;
int bluePin = 11;
int aPin = 6;
int bPin = 7;
int buttonPin = 5;

boolean isOn = true;
int color = 0;

long colors[48]= {
  0xFF2000, 0xFF4000, 0xFF6000, 0xFF8000, 0xFFA000, 0xFFC000, 0xFFE000, 0xFFFF00,
  0xE0FF00, 0xC0FF00, 0xA0FF00, 0x80FF00, 0x60FF00, 0x40FF00, 0x20FF00, 0x00FF00,
  0x00FF20, 0x00FF40, 0x00FF60, 0x00FF80, 0x00FFA0, 0x00FFC0, 0x00FFE0, 0x00FFFF,
  0x00E0FF, 0x00C0FF, 0x00A0FF, 0x0080FF, 0x0060FF, 0x0040FF, 0x0020FF, 0x0000FF,
  0x2000FF, 0x4000FF, 0x6000FF, 0x8000FF, 0xA000FF, 0xC000FF, 0xE000FF, 0xFF00FF,
```

(continued)

LISTING PROJECT 14 (continued)

```
  0xFF00E0, 0xFF00C0, 0xFF00A0, 0xFF0080, 0xFF0060, 0xFF0040, 0xFF0020, 0xFF0000
};

void setup()
{
  pinMode(aPin, INPUT);
  pinMode(bPin, INPUT);
  pinMode(buttonPin, INPUT);
  pinMode(redPin, OUTPUT);
  pinMode(greenPin, OUTPUT);
  pinMode(bluePin, OUTPUT);
}

void loop()
{
  if (digitalRead(buttonPin))
  {
    isOn = ! isOn;
    delay(200);    // de-bounce
  }
  if (isOn)
  {
    int change = getEncoderTurn();
    color = color + change;
    if (color < 0)
    {
      color = 47;
    }
    else if (color > 47)
    {
      color = 0;
    }
    setColor(colors[color]);
  }
  else
  {
   setColor(0);
  }
  delay(1);
}

int getEncoderTurn()
{
  // return -1, 0, or +1
  static int oldA = LOW;
  static int oldB = LOW;
  int result = 0;
  int newA = digitalRead(aPin);
  int newB = digitalRead(bPin);
  if (newA != oldA || newB != oldB)
  {
```

LISTING PROJECT 14 (continued)

```
      // something has changed
      if (oldA == LOW && newA == HIGH)
      {
        result = -(oldB * 2 - 1);
      }
    }
    oldA = newA;
    oldB = newB;
    return result;
}

void setColor(long rgb)
{
    int red = rgb >> 16;
    int green = (rgb >> 8) & 0xFF;
    int blue = rgb & 0xFF;
    analogWrite(redPin, 255 - red);
    analogWrite(greenPin, 255 - green);
    analogWrite(bluePin, 255 - blue);
}
```

The 48 colors in the array are chosen from just such a table, and are a range of colors more or less spanning the spectrum from red to violet.

Putting It All Together

Load the completed sketch for Project 14 from your Arduino Sketchbook and download it to the board (see Chapter 1).

Seven-Segment LEDs

There was a time when the height of fashion was an LED watch. This required the wearer to press a button on the watch for the time to magically appear as four bright red digits. After a while, the inconvenience of having to use both limbs to tell the time overcame the novelty of a digital watch, and the Evil Genius went out and bought an LCD watch instead. This could only be read in bright sunlight.

Seven-segment LEDs (see Figure 6-4) have largely been superseded by backlit LCD displays (see later in this chapter), but they do find uses from time to time. They also add that "Evil Genius" feel to a project.

Figure 6-5 shows the circuit for driving a single seven-segment display.

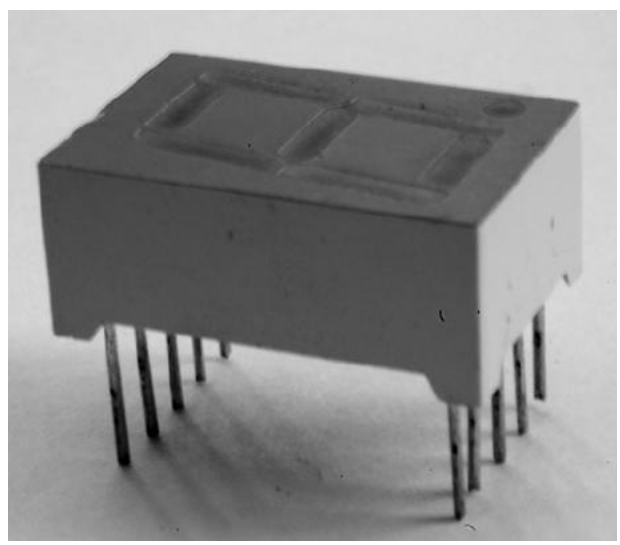


Figure 6-4 Seven-segment LED display.

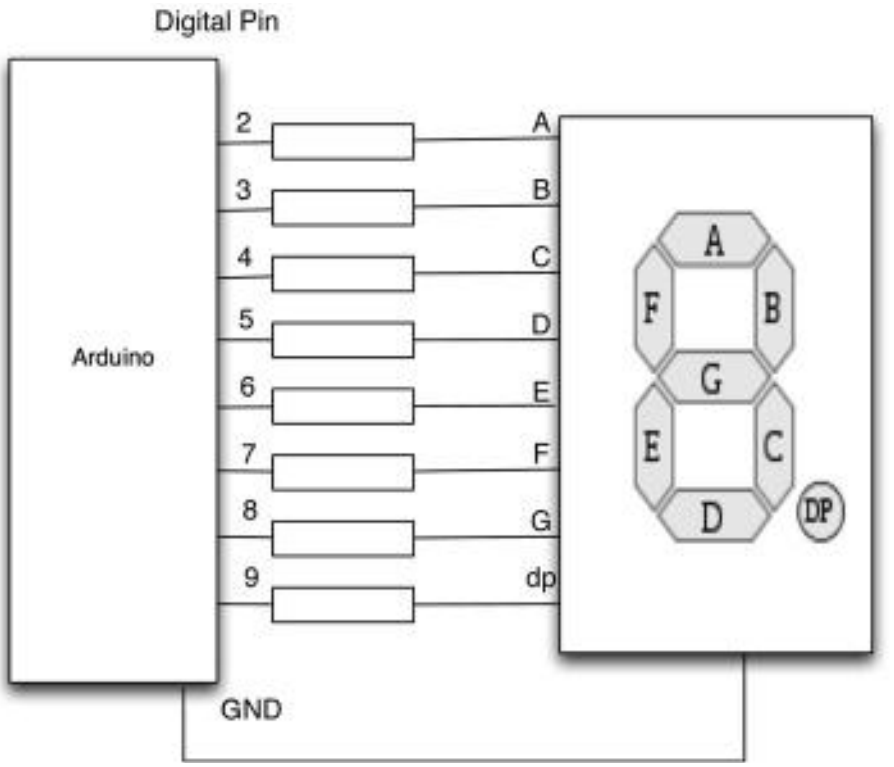


Figure 6-5 An Arduino board driving a seven-segment LED.

A single seven-segment LED is not usually a great deal of use. Most projects will want two or four digits. When this is the case, we will not have enough digital output pins to drive each display separately and so the arrangement of Figure 6-6 is used.

Rather like our keyboard scanning, we are going to activate each display in turn and set the segments for that before moving on to the next digit. We do this so fast that the illusion of all displays being lit is created.

Each display could potentially draw the current for eight LEDs at once, which could amount to 160 mA (at 20 mA per LED)—far more than we can take from a digital output pin. For this reason, we use a transistor that is switched by a digital output to enable each display in turn.

The type of transistor we are using is called a bipolar transistor. It has three connections: the emitter, base, and collector. When a current flows through the base of the transistor and out through the emitter, it allows a much greater current to flow through from the collector to the emitter. We have met this kind of transistor before in Project 4, where we used it to control the current to a high-power Luxeon LED.

We do not need to limit the current that flows through the collector to the emitter, as this is already limited by the series resistors for the LEDs. However, we do need to limit the current flowing into the base. Most transistors will multiply the current by a factor of 100 or more, so we only need to allow about 2 mA to flow through the base to fully turn on the transistor.

Transistors have the interesting property that under normal use, the voltage between base and emitter is a fairly constant 0.6V no matter how much current is flowing. So, if our Arduino pin supplies 5V, 0.6 of that will be across the

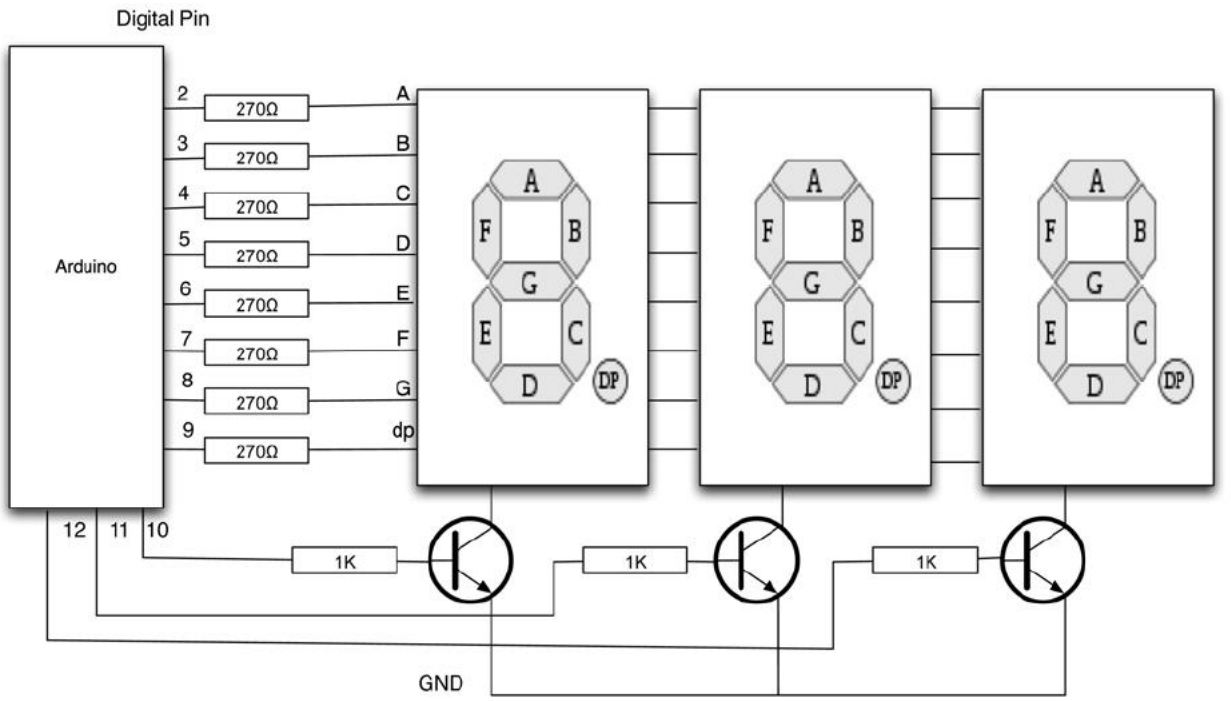


Figure 6-6 Driving more than one seven-segment LED from an Arduino board.

base/emitter of the transistor, meaning that our resistor should have a value of about:

$$R = V/I$$

$$R = 4.4/2mA = 2.2\ K\Omega$$

In actual fact it would be just fine if we let 4 mA flow because the digital output can cope with about 40 mA, so let's choose the nice standard resistor value of 1 KΩ, which will allow us to be sure that the transistor will act like a switch and always turn fully on or fully off.

Project 15
Seven-Segment LED Double Dice

In Project 9 we made a single dice using seven separate LEDs. In this project we will use two seven-segment LED displays to create a double dice.

COMPONENTS AND EQUIPMENT

	Description	Appendix
	Arduino Diecimila or Duemilanove board or clone	1
D1	Two-digit, seven-segment LED display (common anode)	33
R1	100 KΩ 0.5W metal film resistor	13
R4-13	270 Ω 0.5W metal film resistor	6
R2, R3	1 KΩ	7
T1, T2	BC307	39
S1	Push switch	48

Hardware

The schematic for this project is shown in Figure 6-7.

The seven-segment LED module that we are using is described as "common anode," which means that all the anodes (positive ends) of the segment LEDs are connected together. So to

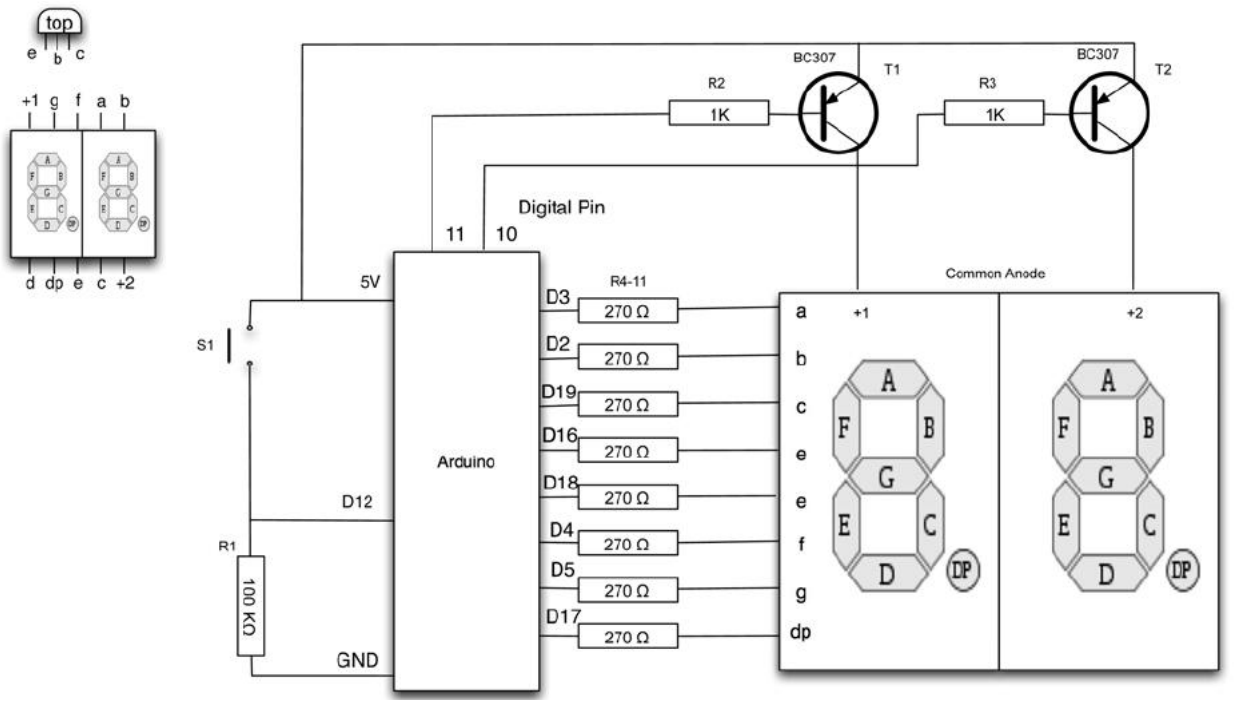


Figure 6-7 Schematic diagram for Project 15.

switch each display on in turn, we must control the positive supply to each of the two common anodes in turn. This is in contrast to how we controlled the power to the Luxeon LED in Project 4, where we controlled the power on the ground side of the circuit. This all means that we are going to use a different type of transistor. Instead of the NPN (negative-positive-negative) transistor we used before, we need to use a PNP (positive-negative-positive) transistor. You will notice the different position of the arrow in the circuit symbol for the transistor to indicate this.

If we were using a common cathode seven-segment display, then we would have an NPN transistor, but at the bottom of the circuit rather than at the top.

The breadboard layout and photograph of the project are shown in Figures 6-8 and 6-9.

To reduce the number of wires required, the seven-segment display is placed close to the Arduino board so that the resistors can directly connect between the Arduino board connectors and the breadboard. This does mean that there are relatively long and bare resistor leads, so take care to ensure that none of them are touching each other. This also accounts for the apparently random allocation of Arduino pins to segments on the LED display. They are arranged like that for ease of connection.

Software

We use an array to contain the pins that are connected to each of the segments “a” to “g” and the decimal point. We also use an array to determine which segments should be lit to display any particular digit. This is a two-dimensional array, where each row represents a separate digit (0 to 9) and each column a segment (see Listing Project 15).

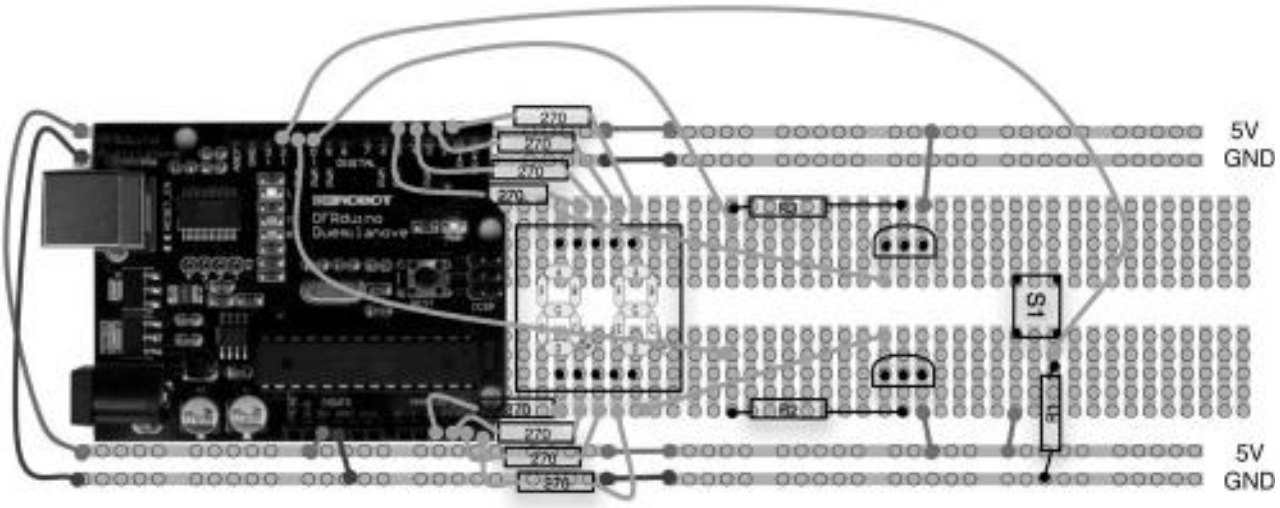


Figure 6-8 Breadboard layout for Project 15.

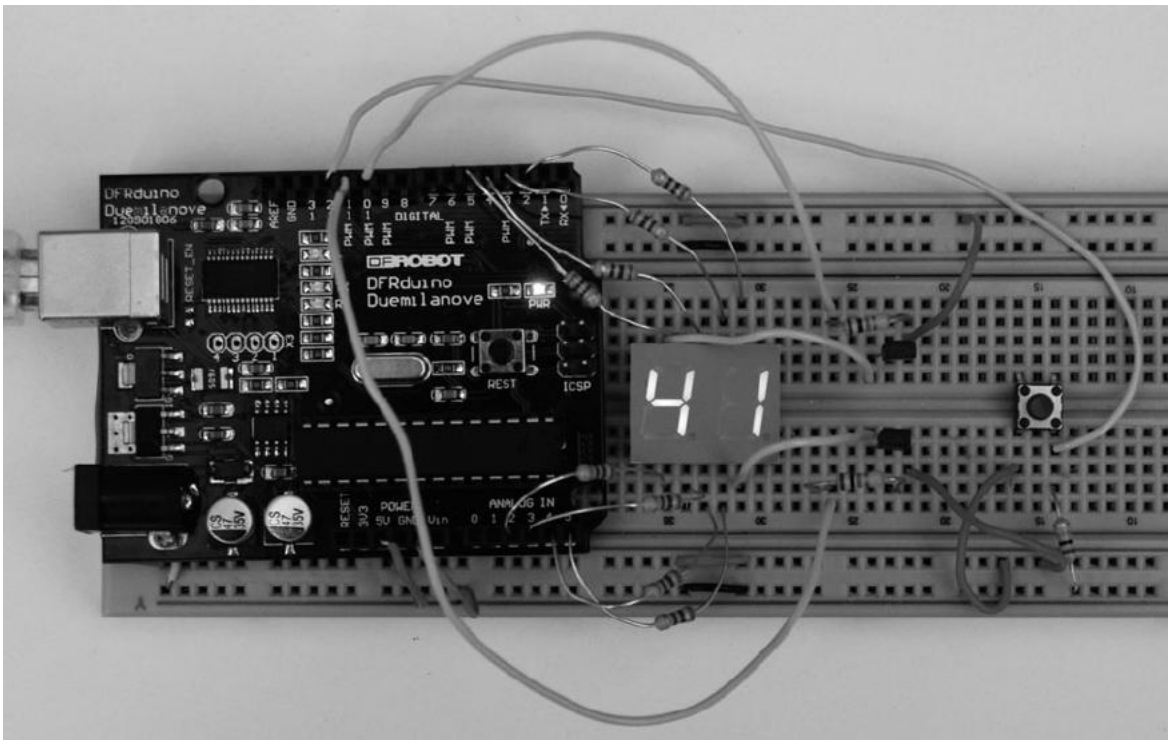


Figure 6-9 Project 15. Double seven-segment LED dice.

LISTING PROJECT 15

```
int segmentPins[] = {3, 2, 19, 16, 18, 4, 5, 17};
int displayPins[] = {10, 11};

int buttonPin = 12;

byte digits[10][8] = {
//  a  b  c  d  e  f  g  .
  { 1, 1, 1, 1, 1, 1, 0, 0},  // 0
  { 0, 1, 1, 0, 0, 0, 0, 0},  // 1
  { 1, 1, 0, 1, 1, 0, 1, 0},  // 2
  { 1, 1, 1, 1, 0, 0, 1, 0},  // 3
  { 0, 1, 1, 0, 0, 1, 1, 0},  // 4
  { 1, 0, 1, 1, 0, 1, 1, 0},  // 5
  { 1, 0, 1, 1, 1, 1, 1, 0},  // 6
  { 1, 1, 1, 0, 0, 0, 0, 0},  // 7
  { 1, 1, 1, 1, 1, 1, 1, 0},  // 8
  { 1, 1, 1, 1, 0, 1, 1, 0}  // 9
};

void setup()
{
  for (int i=0; i < 8; i++)
  {
    pinMode(segmentPins[i], OUTPUT);
  }
  pinMode(displayPins[0], OUTPUT);
  pinMode(displayPins[0], OUTPUT);
  pinMode(buttonPin, INPUT);
}

void loop()
{
  static int dice1;
  static int dice2;
  if (digitalRead(buttonPin))
  {
    dice1 = random(1,7);
    dice2 = random(1,7);
  }
  updateDisplay(dice1, dice2);
}

void updateDisplay(int value1, int value2)
{
  digitalWrite(displayPins[0], HIGH);
  digitalWrite(displayPins[1], LOW);
  setSegments(value1);
  delay(5);
  digitalWrite(displayPins[0], LOW);
```

LISTING PROJECT 15 (continued)

```
  digitalWrite(displayPins[1], HIGH);
  setSegments(value2);
  delay(5);
}

void setSegments(int n)
{
  for (int i=0; i < 8; i++)
  {
    digitalWrite(segmentPins[i], ! digits[n][i]);
  }
}
```

To drive both displays, we have to turn each display on in turn, setting its segments appropriately. So our loop function must keep the values that are displayed in each display in separate variables: dice1 and dice2.

To throw the dice, we use the random function, and whenever the button is pressed, a new value will be set for dice1 and dice2. This means that the throw will also depend on how long the button is pressed for, so we do not need to worry about seeding the random number generator.

Putting It All Together

Load the completed sketch for Project 15 from your Arduino Sketchbook and download it to the board (see Chapter 1).

Project 16
LED Array

LED arrays are one of those components that just look like they would be useful to the Evil Genius. They consist of an array of LEDs (in this case, 8 by 8). These devices can have just a single LED at each position; however, in the device that we are going to use, each of these LEDs is actually a pair of LEDs, one red and one green, positioned under a single lens so that they appear to be one dot. We can then light either one or both LEDs to make a red, green, or orange color.

The completed project is shown in Figure 6-10.

This project makes use of one of these devices and allows multicolor patterns to be displayed on it over the USB connection. As projects go, this one involves a lot of components and will use almost every connection pin of the Arduino.

COMPONENTS AND EQUIPMENT

	Description	Appendix
	Arduino Diecimila or Duemilanove board or clone	1
	8 by 8 LED array (two-color)	34
R1-16	100 Ω 0.5W metal film resistor	5
IC1	4017 decade counter	46
T1-8	2N7000	42
	Extra large breadboard	72 x 2

Hardware

Figure 6-11 shows the schematic diagram for the project. As you might expect, the LEDs are organized in rows and columns with all the negative leads for a particular column connected

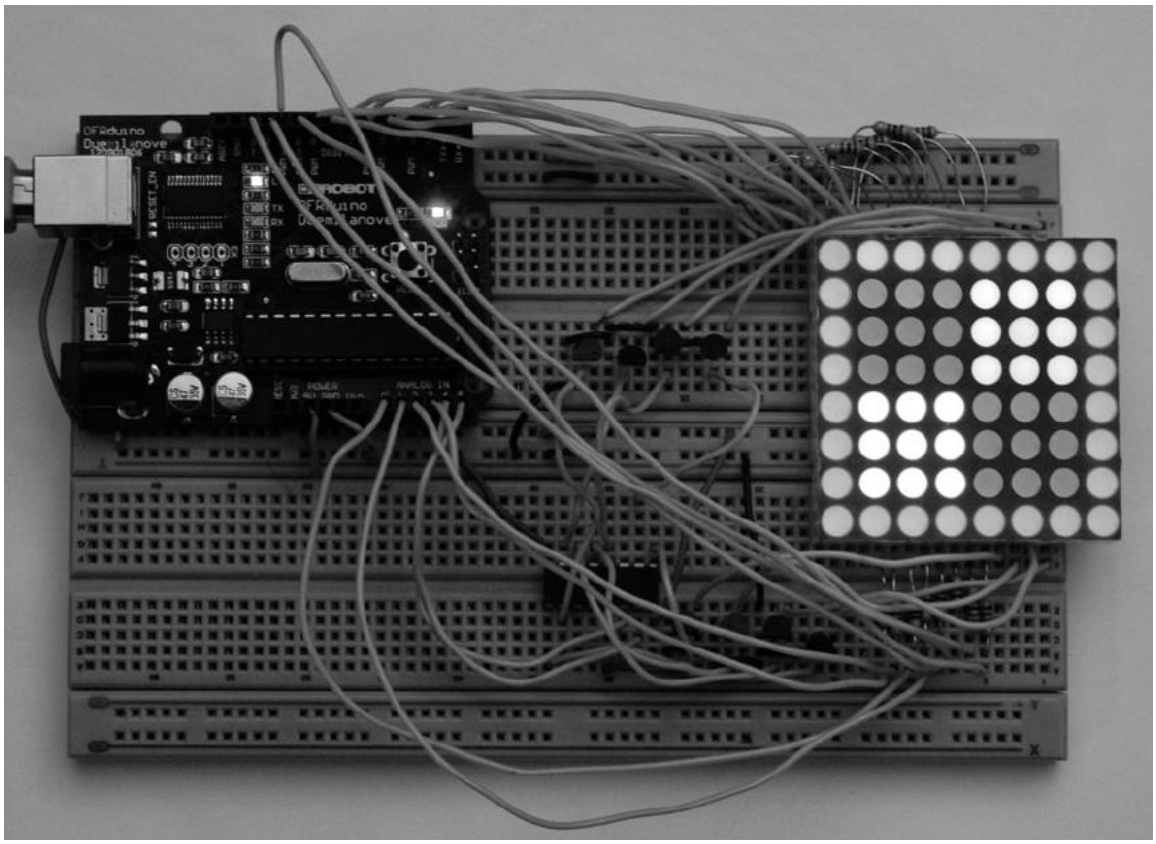


Figure 6-10 Project 16. LED Array.

together and a separate positive connection to each LED in the row.

To drive the matrix, we have to do the same kind of trick that we did with the two-digit, seven-segment display in Project 15 and switch between columns, each time setting the appropriate row of LEDs on and off to create the illusion that all the LEDs are lit at the same time. Actually, only a maximum of 16 (8 red + 8 green) are on at any instant.

There are 24 leads on the LED array, and only 17 pins on the Arduino that we can easily use (D2-13 and A0-5). So we are going to use an integrated circuit called a decade counter to control each of the columns in turn.

The 4017 decade counter has ten output pins, which take it in turn to go high whenever there is a pulse at the "clock" pin. It also has a "reset" pin that sets the count back to 0. So instead of needing an Arduino board output for each row, we just need two outputs: one for clock and one for reset.

Each of the outputs of the 4017 is connected to a field effect transistor (FET). The only reason that we have used an FET rather than a bipolar transistor is that we can connect the gate of the FET directly to an Arduino board output without having to use a current-limiting resistor.

Note that we do not use the first output of the 4017. This is because this pin is on as soon as the 4017 is reset and this would lead to that column being enabled for longer than it should be, making that column appear brighter than the others.

To build this project on the breadboard, we actually need a bigger breadboard than we have

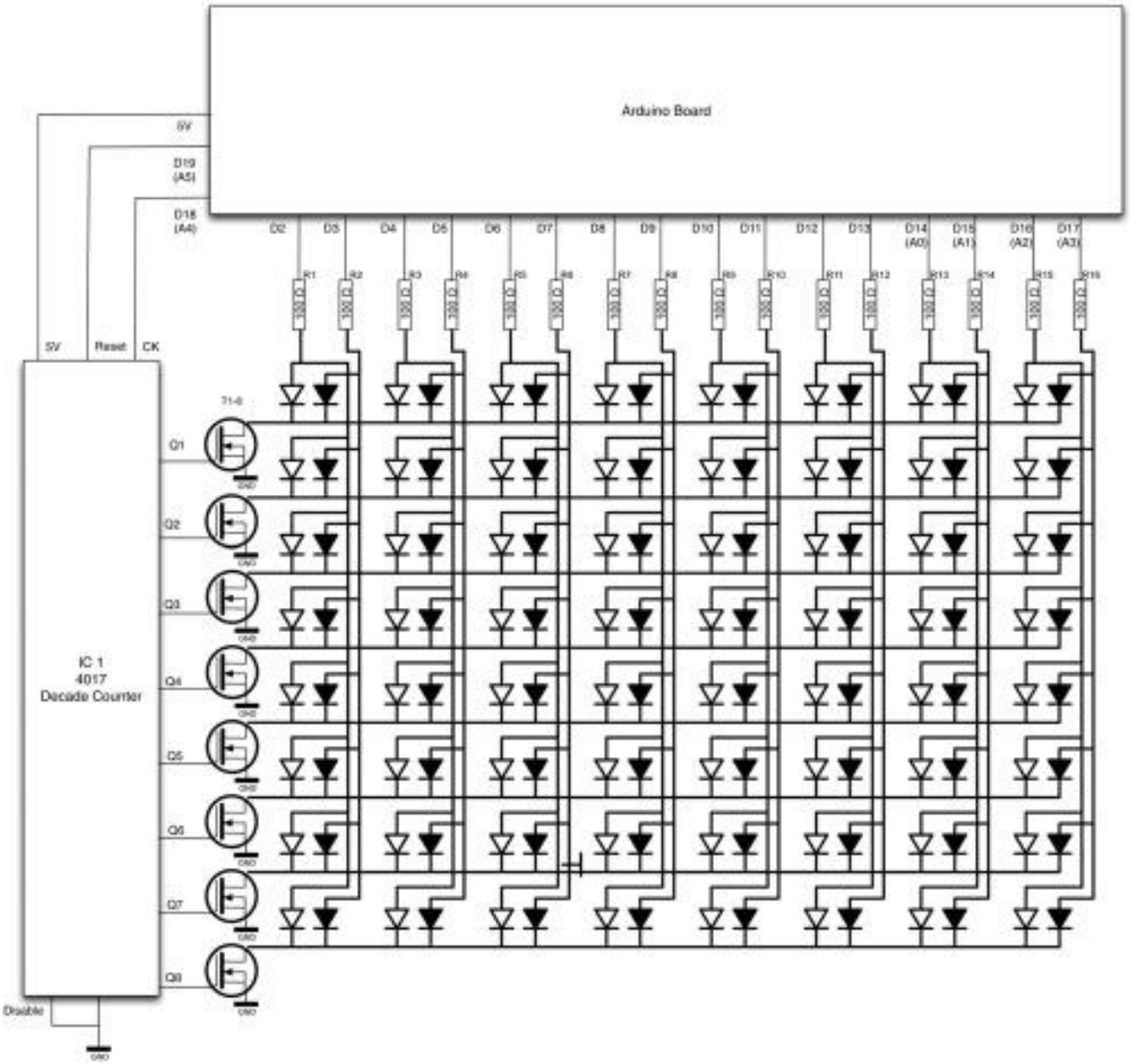


Figure 6-11 Schematic diagram for Project 16.

used so far. The layout for this is shown in Figure 6-12. Be careful to check every connection as you plug your wires in because connections accidentally swapped over produce very strange and hard-to-debug results.

Software

The software for this project is quite short (Listing Project 16), but the tricky bit is getting the timing right, because if you do things too quickly, the 4017 will not have moved properly onto the next row before the next column starts being set. This causes a blurring of colors. On the other hand, if you do things too slowly the display will flicker. This is the reason for the calls to delayMicroseconds. This

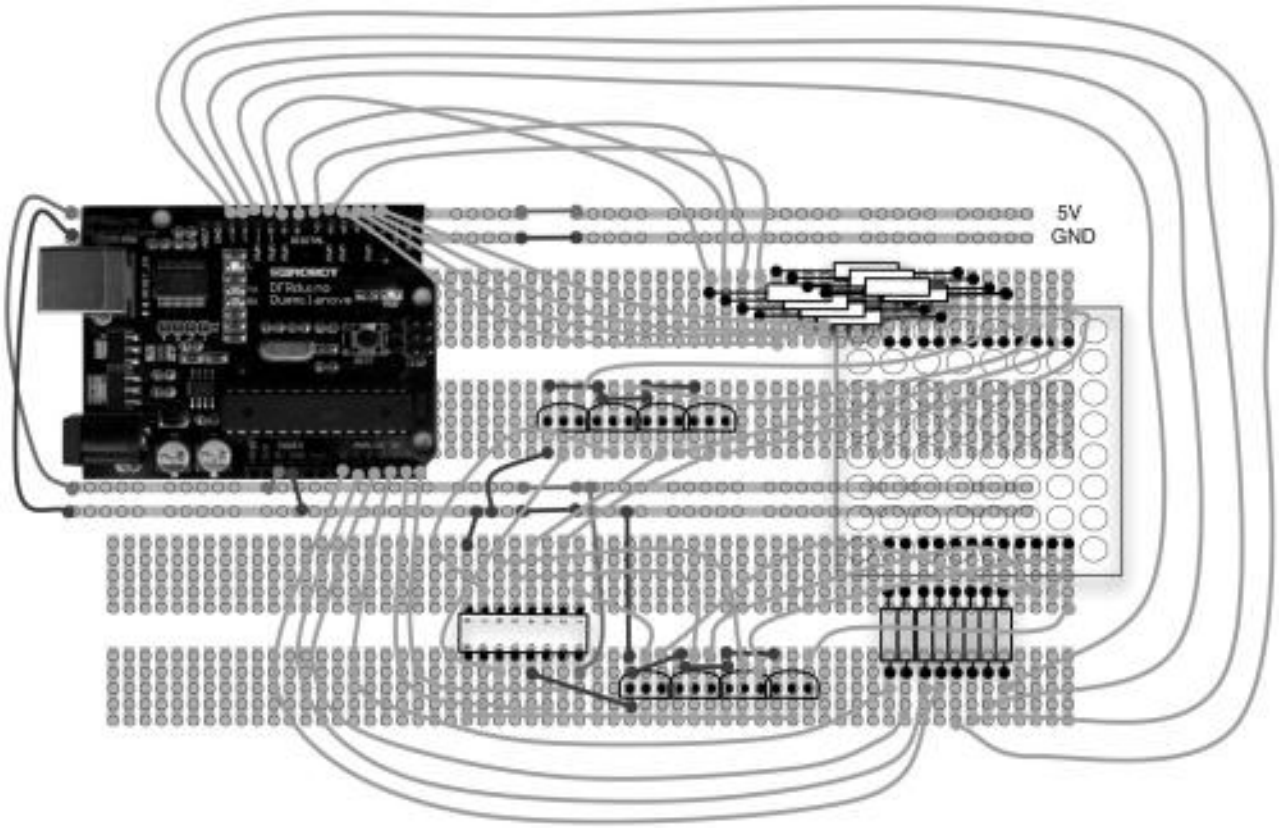


Figure 6-12 Breadboard layout for Project 16.

LISTING PROJECT 16

```
int clockPin = 18;
int resetPin = 19;

int greenPins[8] = {2, 3, 4, 5, 6, 7, 8, 9};
int redPins[8] = {10, 11, 12, 13, 14, 15, 16, 17};

int row = 0;
int col = 0;

// colors off = 0, green = 1, red = 2, orange = 3
byte pixels[8][8] = {
{1, 1, 1, 1, 1, 1, 1, 1},
{1, 2, 2, 2, 2, 2, 2, 1},
{1, 2, 3, 3, 3, 3, 2, 1},
{1, 2, 3, 3, 3, 3, 2, 1},
{1, 2, 3, 3, 3, 3, 2, 1},
{1, 2, 3, 3, 3, 3, 2, 1},
{1, 2, 2, 2, 2, 2, 2, 1},
{1, 1, 1, 1, 1, 1, 1, 1}
};
```

LISTING PROJECT 16 (continued)

```
void setup()
{
  pinMode(clockPin, OUTPUT);
  pinMode(resetPin, OUTPUT);
  for (int i = 0; i < 8; i++)
  {
    pinMode(greenPins[i], OUTPUT);
    pinMode(redPins[i], OUTPUT);
  }
  Serial.begin(9600);
}

void loop()
{
  if (Serial.available())
  {
    char ch = Serial.read();
    if (ch == 'x')
    {
      clear();
    }
    if (ch >= 'a' and ch <= 'g')
    {
      col = 0;
      row = ch - 'a';
    }
    else if (ch >= '0' and ch <= '3')
    {
      byte pixel = ch - '0';
      pixels[row][col] = pixel;
      col++;
    }
  }
  refresh();
}

void refresh()
{
  pulse(resetPin);
  delayMicroseconds(2000);
  for (int row = 0; row < 8; row++)
  {
    for (int col = 0; col < 8; col++)
    {
      int redPixel = pixels[col][row] & 2;
      int greenPixel = pixels[col][row] & 1;
      digitalWrite(greenPins[col], greenPixel);
      digitalWrite(redPins[col], redPixel);
    }
    pulse(clockPin);
```

(continued)

LISTING PROJECT 16 (continued)

```
      delayMicroseconds(1500);
    }
  }

  void clear()
  {
    for (int row = 0; row < 8; row++)
    {
      for (int col = 0; col < 8; col++)
      {
        pixels[row][col] = 0;
      }
    }
  }


  void pulse(int pin)
  {
    delayMicroseconds(20);
    digitalWrite(pin, HIGH);
    delayMicroseconds(50);
    digitalWrite(pin, LOW);
    delayMicroseconds(50);
  }
```

function is like the delay function, but allows shorter delays to be made.

Apart from that, the code is fairly straightforward.

Putting It All Together

Load the completed sketch for Project 16 from your Arduino Sketchbook and download it to the board (see Chapter 1).

You can now try out the project. As soon as it is connected to the USB port and has reset, you should see a test pattern of a green outer ring with a red ring inside that and then a block of orange in the center.

Open the Arduino software's Serial Monitor and type **x**. This should clear the display. You can now change each line of the display by entering a letter for the row (a–h) followed immediately by eight digits. Each digit will be 0 for off, 1 for green, 2 for red, and 3 for orange. So typing **a12121212** will set the top row to alternating red and green. When designing patterns to display, it is a good idea to write out the lines in a text editor or word processor and then paste the entire pattern into the Serial Monitor.

You might like to try entering the following:

```
x
a11222211
b11222211
c11222211
d11111111
e11111111
f11222211
g11222211
h11111111
```

or

```
x
a22222222
b12333321
c11211211
d11122111
e11233211
f12333321
g22222222
h11111111
```

or

```
x
a11111111
b22212221
c11212121
d22212121
e11212121
f22212221
g11111111
h11111111
```

This is a really good project for the Evil Genius to experiment with. You may like to try and produce an animation effect by changing the pixels array while in the loop.

LCD Displays

If our project needs to display more than a few numeric digits, we likely want to use an LCD display module. These have the advantage that they come with built-in driver electronics, so a lot of the work is already done for us and we do not have to poll round each digit, setting each segment.

There is also something of a standard for these devices, so there are lots of devices from different manufacturers that we can use in the same way. The devices to look for are the ones that use the HD44780 driver chip.

LCD panels can be quite expensive from retail electronic component suppliers, but if you look on the Internet, they can often be bought for a few dollars, particularly if you are willing to buy a few at a time.

Figure 6-13 shows a module that can display two rows of 16 characters. Each character is made up of an array of 7 by 5 segments. So it is just as well that we do not have to drive each segment separately.

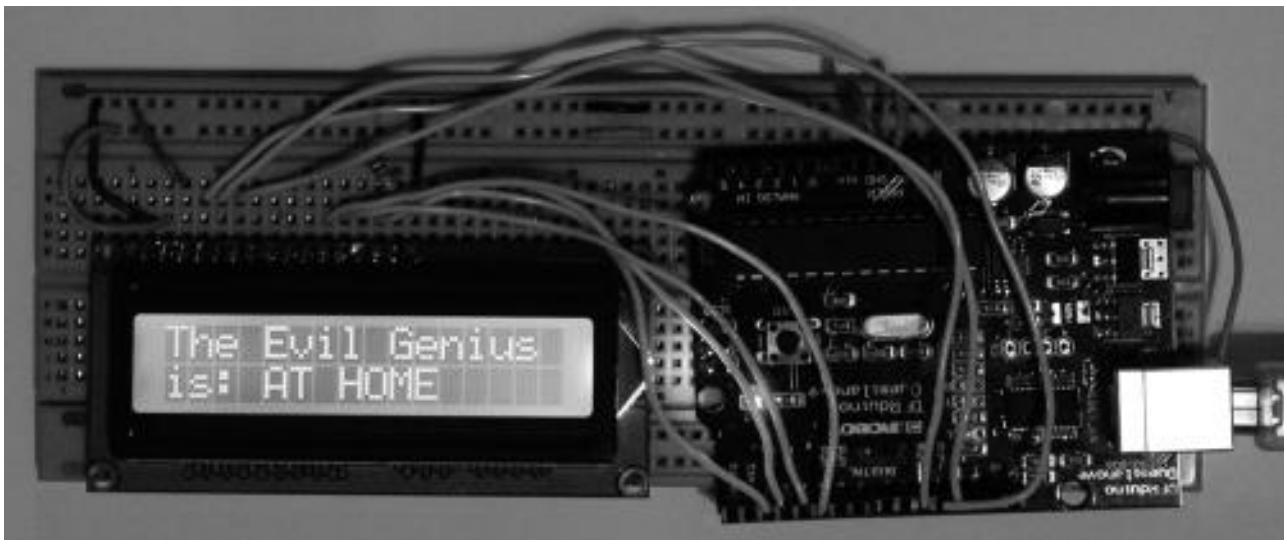


Figure 6-13 2 by 16 LCD module.

The display module includes a character set so that it knows which segments to turn on for any character. This means we just have to tell it which character to display where on the display.

We need just seven digital outputs to drive the display. Four of these are data connections and three control the flow of data. The actual details of what is sent to the LCD module can be ignored, as there is a standard library that we can use.

This is illustrated in the next project.

Project 17
USB Message Board

This project will allow us to display a message on an LCD module from our computer. There is no reason why the LCD module needs to be right next to the computer, so you could use it on the end of a long USB lead to display messages remotely—next to an intercom at the door to the Evil Genius's lair, for example.

COMPONENTS AND EQUIPMENT

	Description	Appendix
	Arduino Diecimila or Duemilanove board or clone	1
	LCD Module (HD44780 controller)	58
R1	100 Ω 0.5W metal film resistor	5
	Strip of 0.1-inch header pins (at least 16)	55

Hardware

The schematic diagram for the LCD display is shown in Figure 6-14 and the breadboard layout in Figure 6-15. As you can see, the only components required are the LCD module itself and a resistor to limit the current to the LED backlight.

The LCD module receives data four bits at a time through the connections D4-7. The LCD module also has connectors for D0-3, which are only used for transferring data eight bits at a time. To reduce the number of pins required, we do not use these.

The easiest way to attach the LCD module to the breadboard is to solder header pins into the connector strip, and then the module can be plugged directly into the breadboard.

Software

The software for this project is straightforward (Listing Project 17). All the work of communicating with the LCD module is taken care of by the LCD library. This library is included as part of the standard Arduino software installation, so we do not need to download or install anything special.

The loop reads any input and if it is a # character clears the display. If it is a / character, it moves to the second row; otherwise, it just displays the character that was sent.

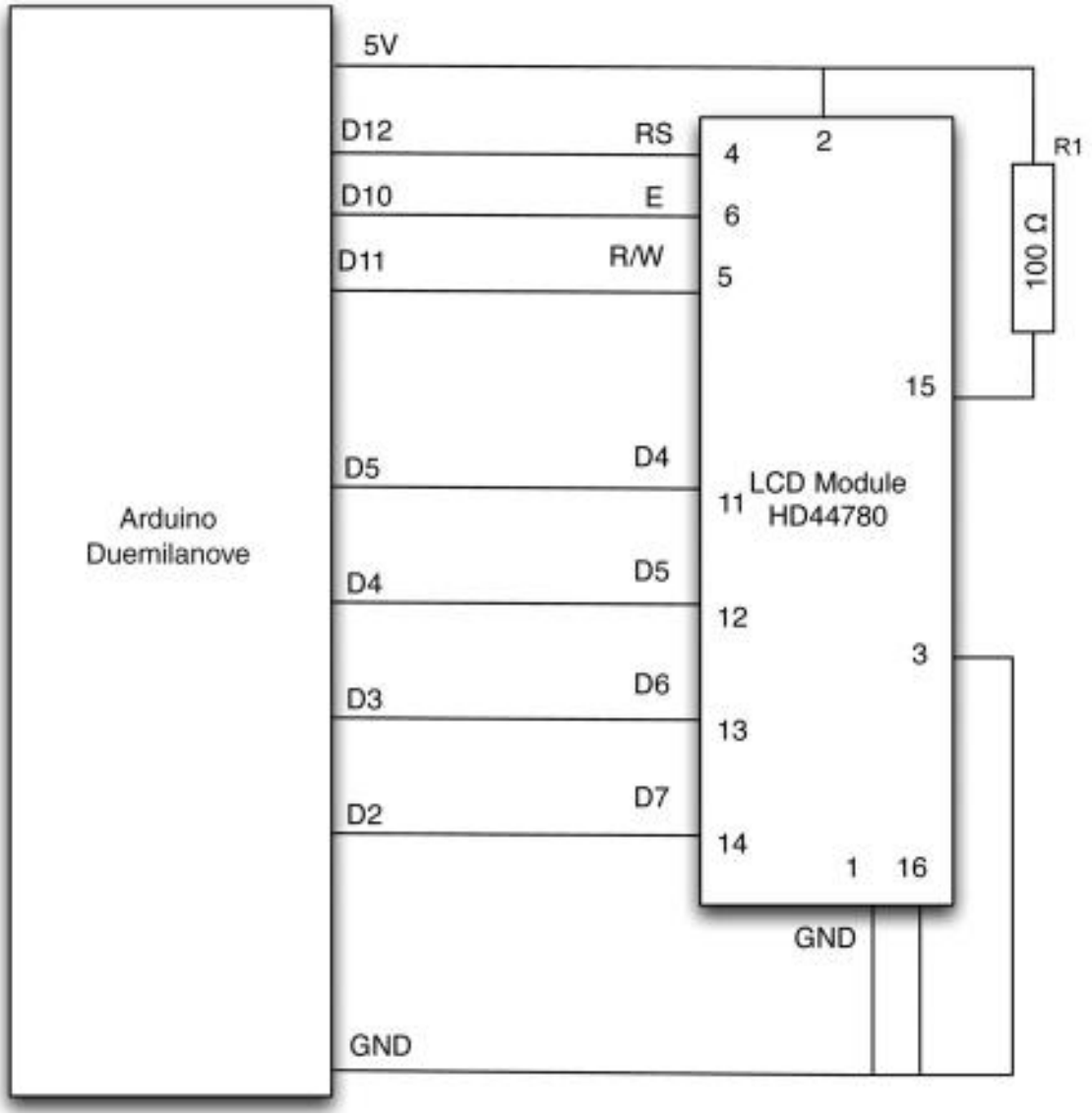


Figure 6-14 Schematic diagram for Project 17.

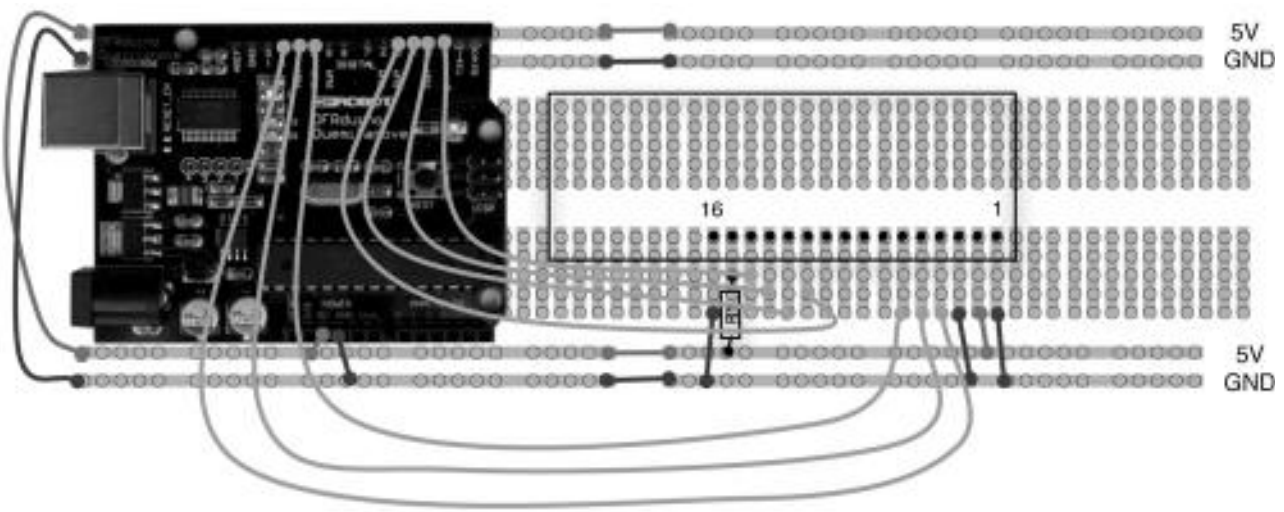


Figure 6-15 Breadboard layout for Project 17.

LISTING PROJECT 17

```
#include <LiquidCrystal.h>

// LiquidCrystal display with:
// rs on pin 12
// rw on pin 11
// enable on pin 10
// d4-7 on pins 5-2
LiquidCrystal lcd(12, 11, 10, 5, 4, 3, 2);

void setup()
{
  Serial.begin(9600);
  lcd.begin(2, 20);
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print("Evil Genius");
  lcd.setCursor(0,1);
  lcd.print("Rules");
}



void loop()
{
  if (Serial.available())
  {
    char ch = Serial.read();
    if (ch == '#')
    {
      lcd.clear();
    }
    else if (ch == '/')
    {
      lcd.setCursor(0,1);
    }
    else
    {
      lcd.write(ch);
    }
  }
}
```

Putting It All Together

Load the completed sketch for Project 17 from your Arduino Sketchbook and download it to the board (see Chapter 1).

We can now try out the project by opening the Serial Monitor and entering some text.

Later on in Project 22, we will be using the LCD panel again with a thermistor and rotary encoder to make a thermostat.

Summary

That's all for LED- and light-related projects. In the next chapter we will look at projects that use sound in one way or another.

This page intentionally left blank

CHAPTER 7

Sound Projects

An Arduino board can be used to both generate sounds as an output and receive sounds as an input using a microphone. In this chapter, we have various "musical instrument" type projects and also projects that process sound inputs.

Although not strictly a "sound" project, our first project is to create a simple oscilloscope so that we can view the waveform at an analog input.

Project 18
Oscilloscope

An oscilloscope is a device that allows you to see an electronic signal so that it appears as a waveform. A traditional oscilloscope works by amplifying a signal to control the position of a dot on the Y-axis (vertical axis) of a cathode ray tube while a timebase mechanism sweeps left to right on the X-axis and then flips back when it reaches the end. The result will look something like Figure 7-1.

These days, cathode ray tubes have largely been replaced by digital oscilloscopes that use LCD displays, but the principles remain the same.

This project reads values from the analog input and sends them over USB to your computer. Rather than be received by the Serial Monitor, they are received by a little program that displays them in an oscilloscope-like manner. As the signal changes, so does the shape of the waveform.

Note that as oscilloscopes go, this one is not going to win any prizes for accuracy or speed, but it is kind of fun.

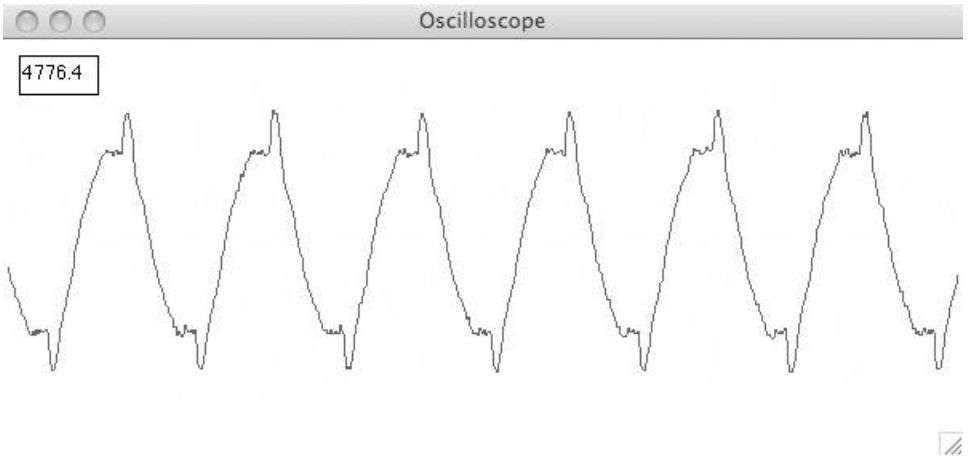


Figure 7-1 50-Hz noise on oscilloscope.

COMPONENTS AND EQUIPMENT

	Description	Appendix
	Arduino Diecimila or Duemilanove board or clone	1
C1	220nF nonpolarized	21
C2, C3	100 μF electrolytic	22
R1, R2	1 MΩ 0.5W metal film resistor	15
R3, R4	1 KΩ 0.5W metal film resistor	7

This is the first time we have used capacitors. C1 can be connected either way round; however, C2 and C3 are polarized and must be connected the correct way round, or they are likely to be damaged. As with LEDs, on polarized capacitors, the positive lead (marked as the white rectangle on the schematic symbol) is longer than the negative lead. The negative lead also often has a – (minus) or diamond shape next to the negative lead.

Hardware

Figure 7-2 shows the schematic diagram for Project 18 and Figure 7-3 the breadboard layout.

There are two parts to the circuit. R1 and R2 are high-value resistors that "bias" the signal going to the analog input to 2.5V. They are just like a voltage divider. The capacitor C1 allows the signal to pass without any direct current (DC) component to the signal (alternating current, or AC, mode in a traditional oscilloscope).

R3, R4, C2, and C3 just provide a stable reference voltage of 2.5V. The reason for this is so that our oscilloscope can display both positive and negative signals. So one terminal of our test lead is fixed at 2.5V; any signal on the other lead will be relative to that. A positive voltage will mean a value at the analog input of greater than 2.5V, and a negative value will mean a value at the analog input of less than 2.5V.

Figure 7-4 shows the completed oscilloscope.

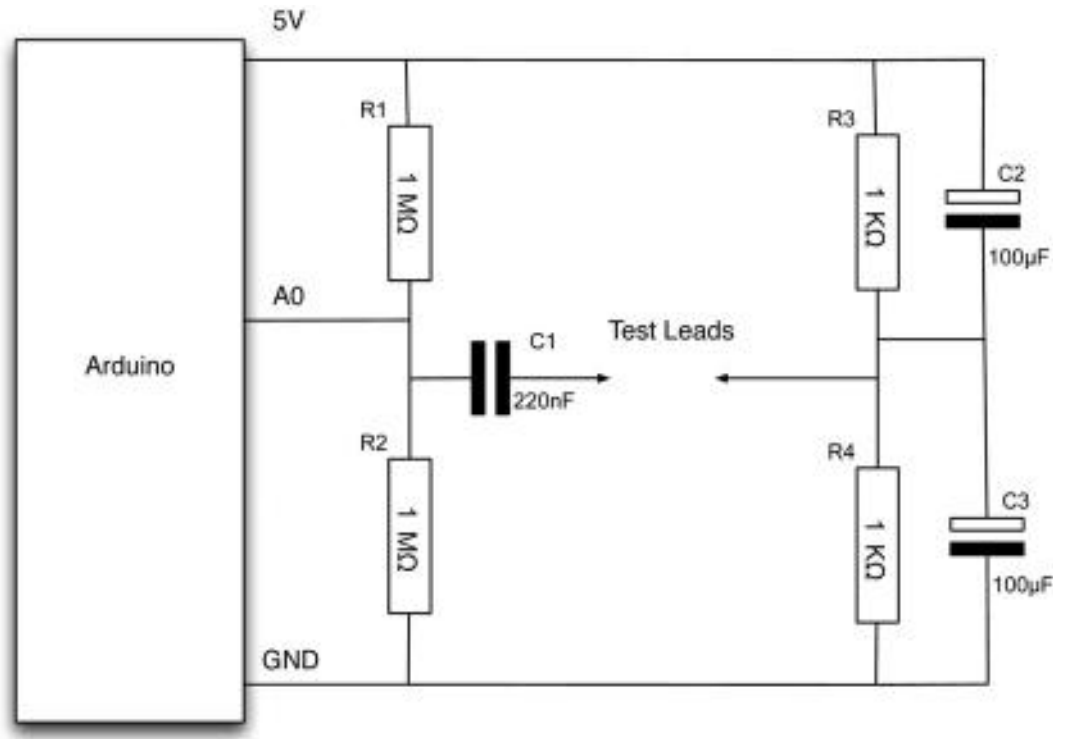


Figure 7-2 Schematic diagram for Project 18.

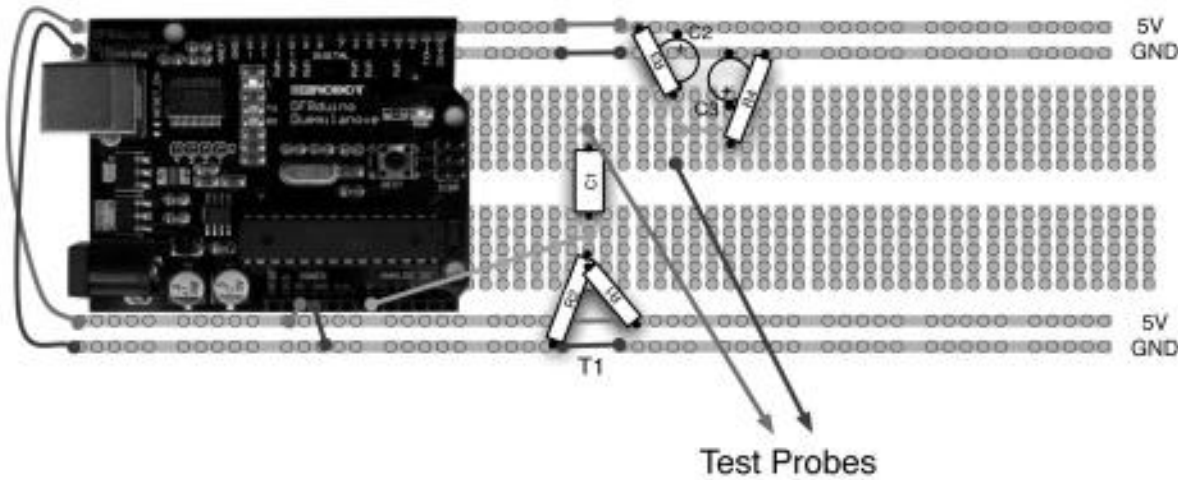


Figure 7-3 Breadboard layout for Project 18.

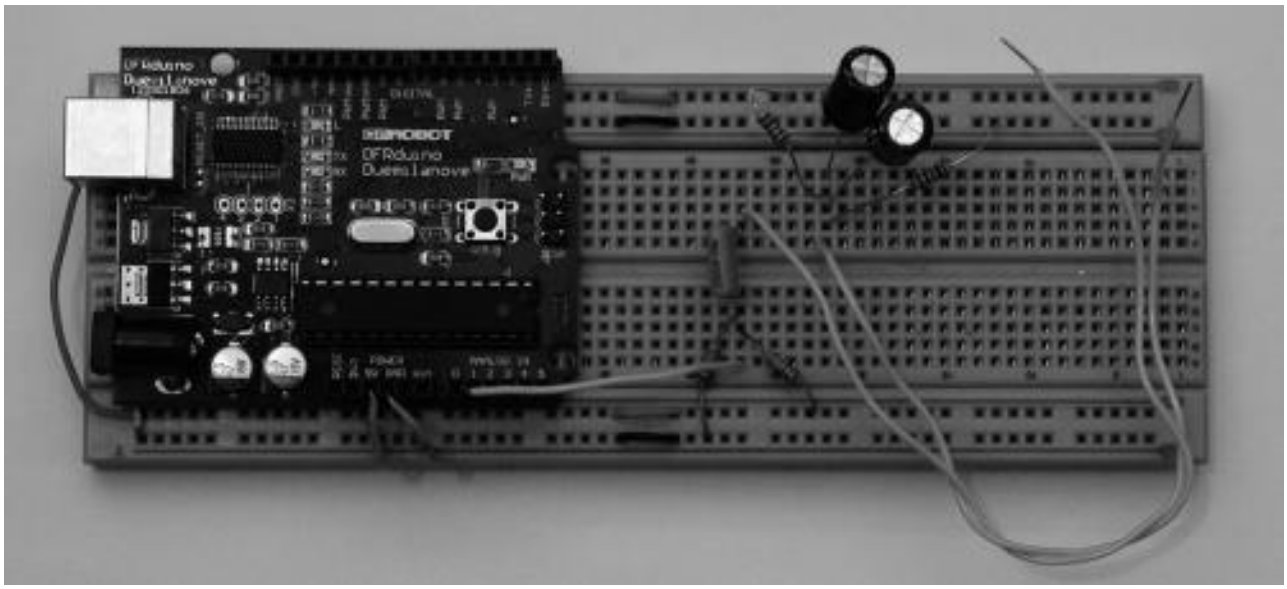


Figure 7-4 Project 18. Oscilloscope.

Software

The sketch is short and simple (Listing Project 18). Its only purpose is to read the analog input and blast it out to the USB port as fast as possible.

The first thing to note is that we have increased the baud rate to 115,200, the highest available. To get as much data through the connection as possible without resorting to complex compression techniques, we are going to shift our raw ten-bit value right two bits (>> 2); this has the effect of dividing it by four and making it fit into a single byte.

We obviously need some corresponding software to run on our computer so that we can see the data sent by the board (Figure 7-1). This can be downloaded from www.arduinoevilgenius.com.

To install the software, you first need to install the Ruby language on your computer. If you use a Mac, you are in luck, because they come with Ruby pre-installed. If you are using Windows or

LISTING PROJECT 18

```
#define CHANNEL_A_PIN 0

void setup()
{
 Serial.begin(115200);
}

void loop()
{
  int value =
      analogRead(CHANNEL_A_PIN);
  value = (value >> 2) & 0xFF;
  Serial.print(value, BYTE);
  delayMicroseconds(100);
}
```

LINUX, please follow the instructions at http://www.ruby-lang.org/en/downloads.

Once you have installed Ruby, the next step is to install an optional Ruby module to communicate with the USB port. To install this, open a command prompt in Windows or a terminal for Mac and Linux, and if using Windows type:

```
gem install ruby-serialport
```

If you are using Mac or Linux, enter:

```
sudo gem install ruby-serialport
```

If everything has worked okay, you should see a message like this:

```
Building native extensions.  This could
   take a while...
Successfully installed ruby-
   serialport-0.7.0
1 gem installed
Installing ri documentation for
   ruby-serialport-0.7.0...
Installing RDoc documentation for
   ruby-serialport-0.7.0...
```

To run the software, change directory in your terminal or command prompt to the directory where you downloaded scope.rb. Then just type:

```
ruby scope.rb
```

A window like Figure 7-1 should then appear.

Putting It All Together

Load the completed sketch for Project 18 from your Arduino Sketchbook and download it to the board (see Chapter 1). Install the software for your computer as described previously, and you are ready to go.

The easiest way to test the oscilloscope is to use the one readily available signal that permeates most of our lives and that is mains hum. Mains electricity oscillates at 50 or 60Hz (depending on where you live in the world), and every electrical appliance emits electromagnetic radiation at this frequency. To pick it up, all you have to do is touch the test lead connected to the analog input and you should see a signal similar to that of Figure 7-1. Try waving your arm around near any electrical equipment and see how this signal changes.

As well as showing the waveform, the window contains a small box with a number in it. This is the number of samples per second. Each sample represents one pixel across the window, and the window is 600 pixels wide. A sample rate of 4700 samples per second means that each sample has a duration of 1/4700 seconds and so the full width of 600 samples represents 600/4700 or 128 milliseconds. The wavelength in Figure 7-1 is about 1/6th of that, or 21 milliseconds, which equals a frequency of 1/0.021 or 47.6Hz. That's close enough to confirm that what we are seeing there is a mains frequency hum of 50Hz.

The amplitude of the signal, as displayed in Figure 7-1, has a resolution of one pixel per sample step, there being 256 steps. So if you connect the two test leads together, you should see a horizontal line halfway across the window. This corresponds to 0V and is 128 pixels from the top of the screen, as the window is 256 pixels high. This means that since the signal is about two-thirds of the window, the amplitude of the signal is about 3V peak to peak.

You will notice that the sample rate changes quite a lot, something that reinforces that this is the crudest of oscilloscopes and should not be relied on for anything critical.

To alter the timebase, change the value in the delayMicroseconds function.

Sound Generation

You can generate sounds from an Arduino board by just turning one of its pins on and off at the right frequency. If you do this, the sound produced is rough and grating. This is called a *square wave*. To produce a more pleasing tone, you need a signal that is more like a sine wave (see Figure 7-5).

Generating a sine wave requires a little bit of thought and effort. A first idea may be to use the analog output of one of the pins to write out the waveform. However, the problem is that the analog outputs from an Arduino are not true analog outputs but PWM outputs that turn on and off very rapidly. In fact, their switching frequency is at an audio frequency, so without a lot of care, our signal will sound as bad as a square wave.

A better way is to use a digital-to-analog converter, or DAC as they are known. A DAC has a number of digital inputs and produces an output voltage proportional to the digital input value. Fortunately, it is easy to make a simple DAC—all you need are resistors.

Figure 7-6 shows a DAC made from what is called an R-2R resistor ladder.

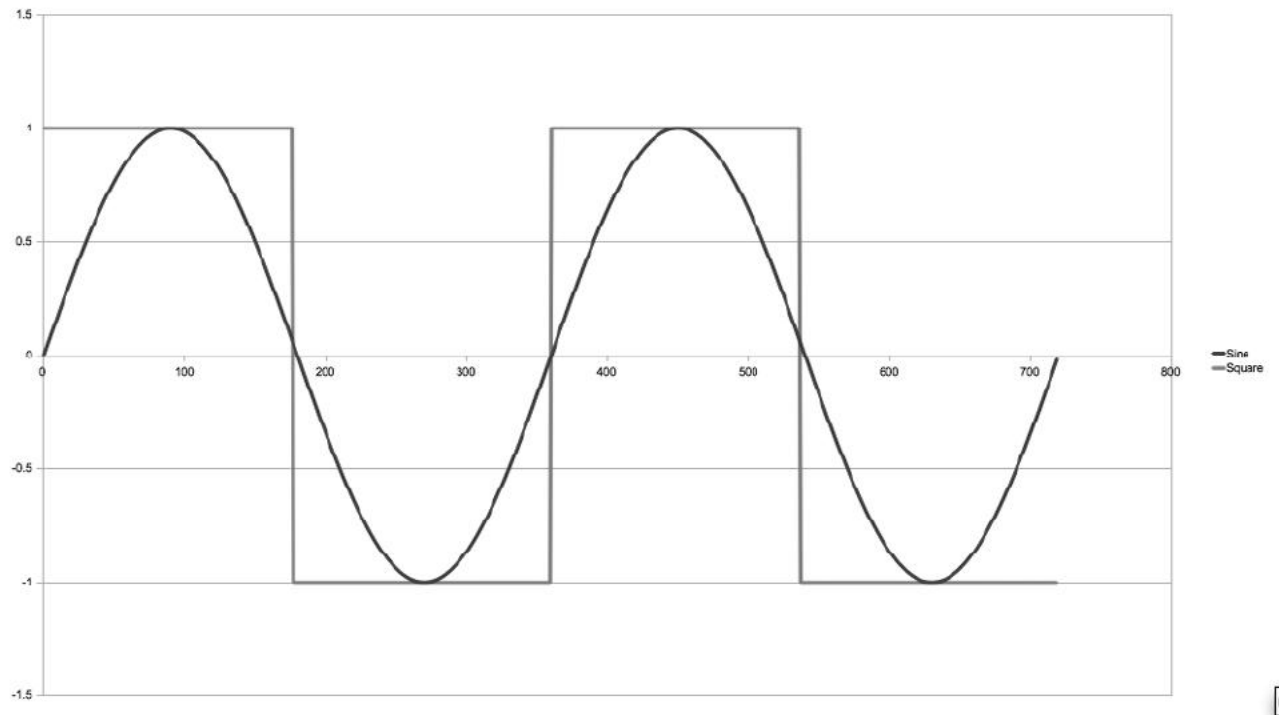


Figure 7-5 Square and sine waves.

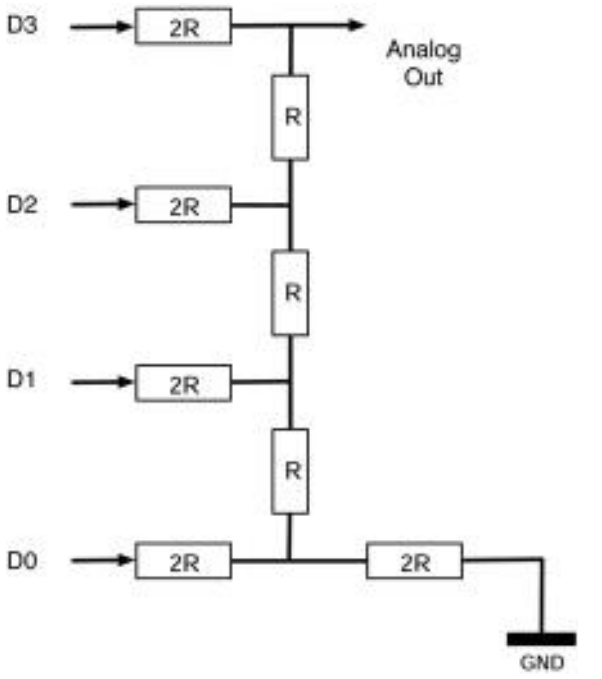


Figure 7-6 DAC using an R-2R ladder.

TABLE 7-1 Analog Output from Digital Inputs

D3	D2	D1	D0	Output
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	10
1	0	1	1	11
1	1	0	0	12
1	1	0	1	13
1	1	1	0	14
1	1	1	1	15

It uses resistors of a value R and twice R, so R might be 5 KΩ and 2R 10 KΩ. Each of the digital inputs will be connected to an Arduino digital output. The four digits represent the four bits of the digital number. So this gives us 16 different analog outputs, as shown in Table 7-1.

Project 19
Tune Player

This project will play a series of musical notes through a miniature loudspeaker using a DAC to approximate a sine wave.

COMPONENTS AND EQUIPMENT

	Description	Appendix
	Arduino Diecimila or Duemilanove board or clone	1
C1	100nF non-polarized	20
C2	100 μF, 16V electrolytic	22
R1-5	10 KΩ 0.5W metal film resistor	9
R6-8	4.7 KΩ 0.5W metal film resistor	8
R9	1 MΩ 0.5W metal film resistor	15
R10	100 KΩ linear potentiometer	13
IC1	TDA7052 1W audio amplifier	47
	Miniature 8 Ω loudspeaker	59

If you can get a miniature loudspeaker with leads for soldering to a PCB, then this can be plugged directly into the breadboard. If not, you will either have to solder short lengths of solid-core wire to the terminals or, if you do not have access to a soldering iron, carefully twist some wires round the terminals.

Hardware

To try and keep the number of components to a minimum, we have used an integrated circuit to amplify the signal and drive the loudspeaker. The TDA7052 IC provides 1W of power output in an easy-to-use little eight-pin chip.

Figure 7-7 shows the schematic diagram for Project 19 and the breadboard layout is shown in Figure 7-8.

C1 is used to link the output of the ADC to the input of the amplifier, and C2 is used as a decoupling capacitor that shunts any noise on the power lines to ground. This should be positioned as close as possible to IC1.

R9 and the variable resistor R10 form a potential divider to reduce the signal from the resistor ladder by at least a factor of 10, depending on the setting of the variable resistor.

Software

To generate a sine wave, the sketch steps through a series of values held in the sin16 array. Theses values are shown plotted on a chart in Figure 7-9. It is not the smoothest sine wave in the world, but is a definite improvement over a square wave (see Listing Project 19).

The playNote function is the key to generating the note. The pitch of the note generated is controlled by the delay after each step of the signal. The loop to write out the waveform is itself inside a loop that writes out a number of cycles sufficient to make each note about the same duration.

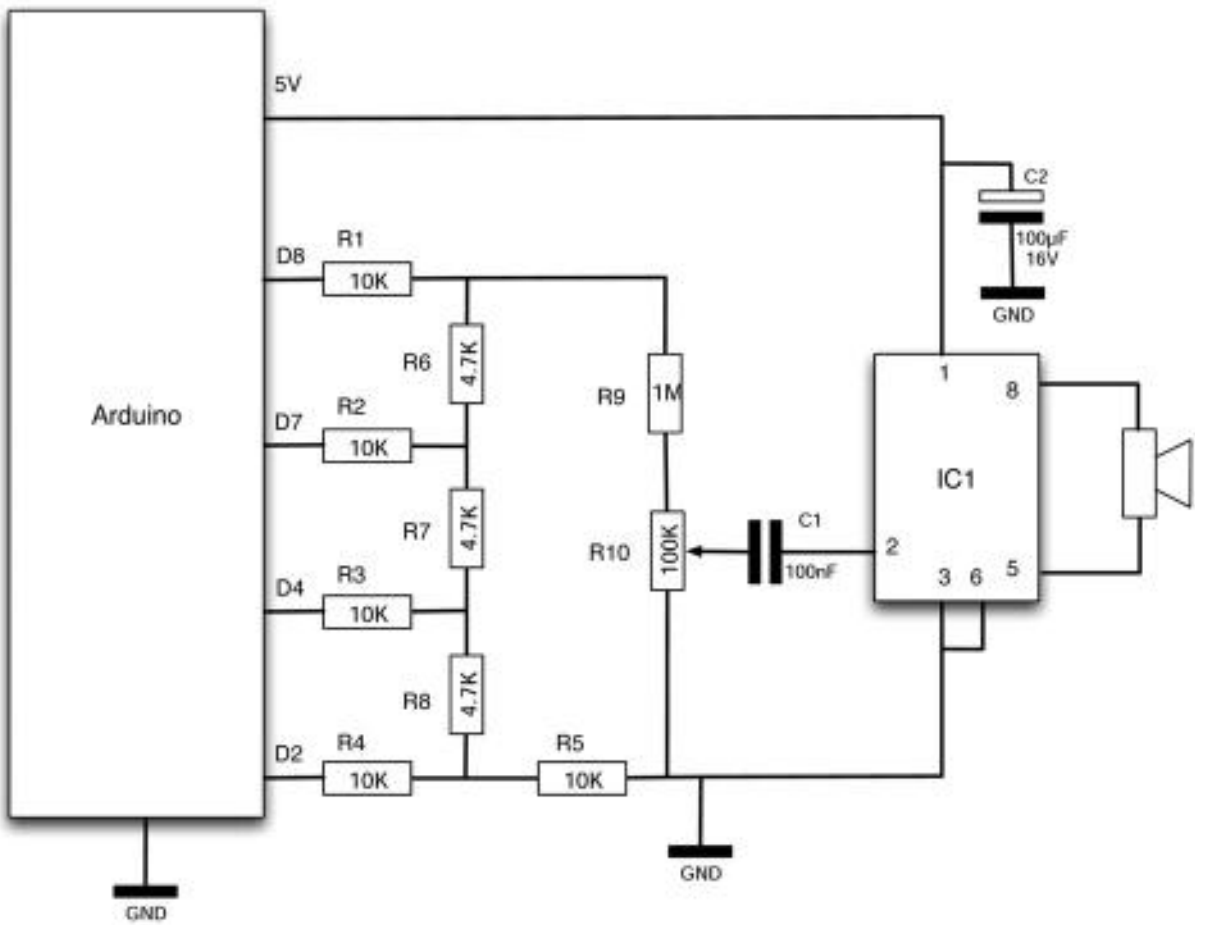


Figure 7-7 Schematic diagram for Project 19.

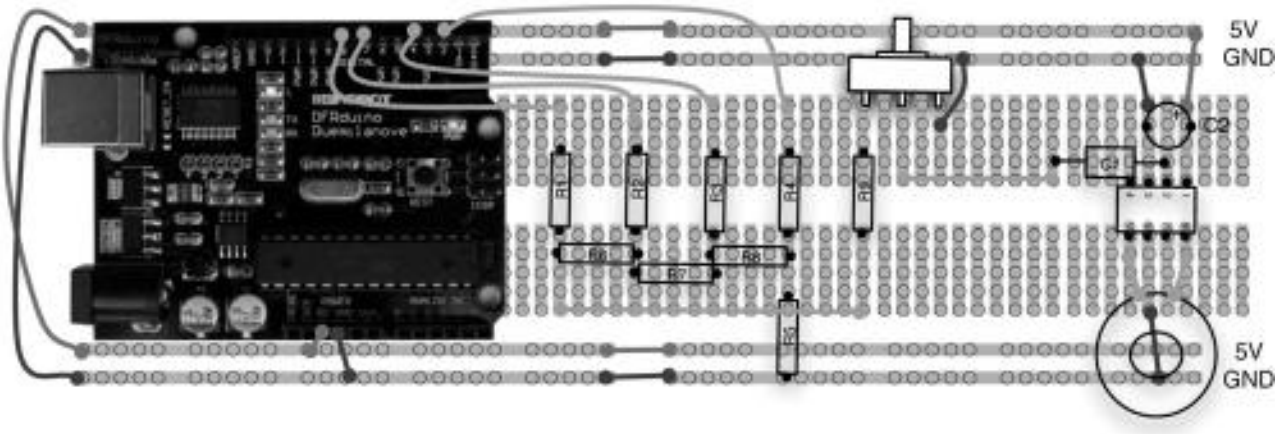


Figure 7-8 Breadboard layout for Project 19.

LISTING PROJECT 19

```
int dacPins[] = {2, 4, 7, 8};
int sin16[] = {7, 8, 10, 11, 12, 13, 14, 14, 15, 14, 14, 13, 12, 11,
               10, 8, 7, 6, 4, 3, 2, 1, 0, 0, 0, 0, 0, 1, 2, 3, 4, 6};

int lowToneDurations[] = {120, 105, 98, 89, 78, 74, 62};
//                          A    B   C   D    E  F   G
int highToneDurations[] = { 54, 45, 42, 36, 28, 26, 22 };
//                            a   b   c   d   e   f   g

// Scale
//char* song = "A B C D E F G a b c d e f g";

// Jingle Bells
//char* song = "E E EE E E EE E G C D EEEE F F F F F E E E E D D E DD GG E E EE
   E E EE E G C D EEEE F F F F F E E E G G F D CCCC";

// Jingle Bells - Higher
char* song = "e e ee e e ee e g c d eeee f f f f f e e e e d d e dd gg e e ee e
   e ee e g c d eeee f f f f f e e e g g f d cccc";


void setup()
{
  for (int i = 0; i < 4; i++)
  {
    pinMode(dacPins[i], OUTPUT);
  }
}

void loop()
```

LISTING PROJECT 19 (continued)

```
{
  int i = 0;
  char ch = song[0];
  while (ch != 0)
  {
    if (ch == ' ')
    {
      delay(75);
    }
    else if (ch >= 'A' and ch <= 'G')
    {
      playNote(lowToneDurations[ch - 'A']);
    }
    else if (ch >= 'a' and ch <= 'g')
    {
      playNote(highToneDurations[ch - 'a']);
    }
    i++;
    ch = song[i];
  }

  delay(5000);
}

void setOutput(byte value)
{
    digitalWrite(dacPins[3], ((value & 8) > 0));
    digitalWrite(dacPins[2], ((value & 4) > 0));
    digitalWrite(dacPins[1], ((value & 2) > 0));
    digitalWrite(dacPins[0], ((value & 1) > 0));
}

void playNote(int pitchDelay)
{
  long numCycles = 5000 / pitchDelay + (pitchDelay / 4);
  for (int c = 0; c < numCycles; c++)
  {
    for (int i = 0; i < 32; i++)
    {
      setOutput(sin16[i]);
      delayMicroseconds(pitchDelay);
    }
  }
}
```

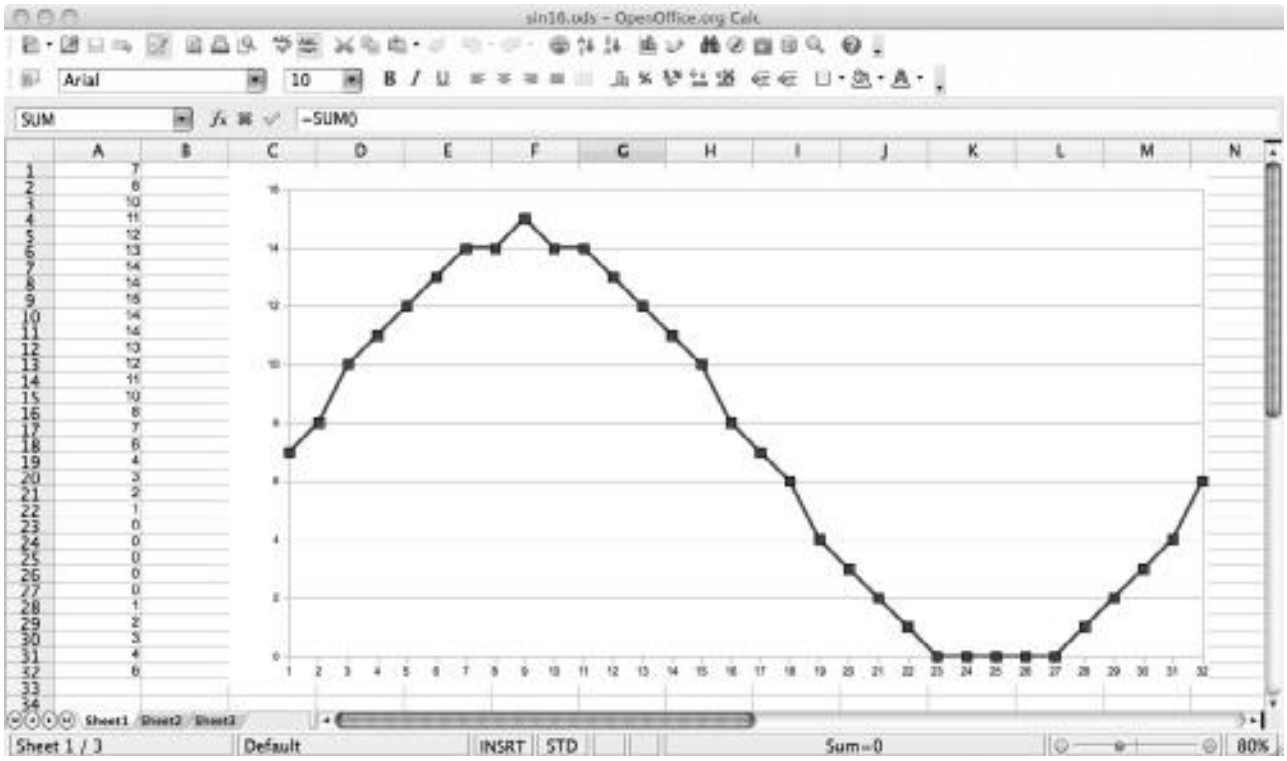


Figure 7-9 A plot of the sin16 array.

The playNote function calls setOutput to set the value of the four digital pins connected to the resistor ladder. The & (and) operator is used to mask the value so that we only see the value of the bit we are interested in.

Tunes are played from an array of characters, each character corresponding to a note, and a space corresponding to the silence between notes. The main loop looks at each letter in the song variable and plays it. When the whole song is played, there is a pause of five seconds and then the song begins again.

The Evil Genius will find this project useful for inflicting discomfort on his or her enemies.

Putting It All Together

Load the completed sketch for Project 19 from your Arduino Sketchbook and download it to the board (see Chapter 1).

You might like to change the tune played from "Jingle Bells." To do this, just comment out the line starting with "char* song =" by putting // in front of it and then define your own array.

The notation works as follows. There are two octaves, and high notes are lowercase "a" to "g" and low notes "A" to "G." For a longer duration note, just repeat the note letter without putting a space in between.

You will have noticed that the quality is not great. It is still a lot less nasty than using a square wave, but is a long way from the tunefulness of a real musical instrument, where each note has an "envelope" where the amplitude (volume) of the note varies with the note as it is played.

Project 20
Light Harp

This project is really an adaptation of Project 19 that uses two light sensors (LDRs): one that controls the pitch of the sound, the other to control the volume. This is inspired by the Theremin musical instrument that is played by mysteriously waving your hands about between two antennae. In actual fact, this project produces a sound more like a bagpipe than a harp, but it is quite fun.

COMPONENTS AND EQUIPMENT

	Description	Appendix
	Arduino Diecimila or Duemilanove board or clone	1
C1	100nF un-polarized	20
C2, C3	100 μF, 16V electrolytic	22
R1-5	10 KΩ 0.5W metal film resistor	9
R6-8	4.7 KΩ 0.5W metal film resistor	8
R9, R11	1 MΩ 0.5W metal film resistor	15
R10	100 KΩ 0.5W metal film resistor	13
R12, R13	47 KΩ 0.5W metal film resistor	11
R14, R15	LDR	19
IC1	TDA7052 1W audio amplifier	47
	Miniature 8 Ω loudspeaker	59

If you can get a miniature loudspeaker with leads for soldering to a PCB, then this can be plugged directly into the breadboard. If not, you will either have to solder short lengths of solid-core wire to the terminals or, if you do not have access to a soldering iron, carefully twist some solid-core wires round the terminals.

Hardware

The volume of the sound will be controlled using a PWM output (D6) connected to the volume control input of the TDA7052. We want to eliminate all traces of the PWM pulses so we can pass the output through a low-pass filter consisting of R11 and C3. This allows only slow changes in the signal to get past. One way to think of this is to pretend that the capacitor C3 is a bucket that is filled with charge from resistor R11. Rapid fluctuations in the signal will have little effect, as there is a smoothing (integration) of the signal.

Figures 7-10 and 7-11 show the schematic diagram and breadboard layout for the project and you can see the final project in Figure 7-12.

The LDRs, R14 and R15, are positioned at opposite ends of the breadboard to make it easier to play the instrument with two hands.

Software

The software for this project has a lot in common with Project 19 (see Listing Project 20).

The main differences are that the pitchDelay period is set by the value of the analog input 0. This is then scaled to the right range using the map function. Similarly, the volume voltage is set by reading the value of analog input 1, scaling it using map, and then writing it out to PWM output 6. It would be possible to just use the LDR R14 to directly control the output of the resistor ladder, but this way gives us more control over scaling and offsetting the output, and we wanted to illustrate smoothing a PWM signal to use it for generating a steady output.

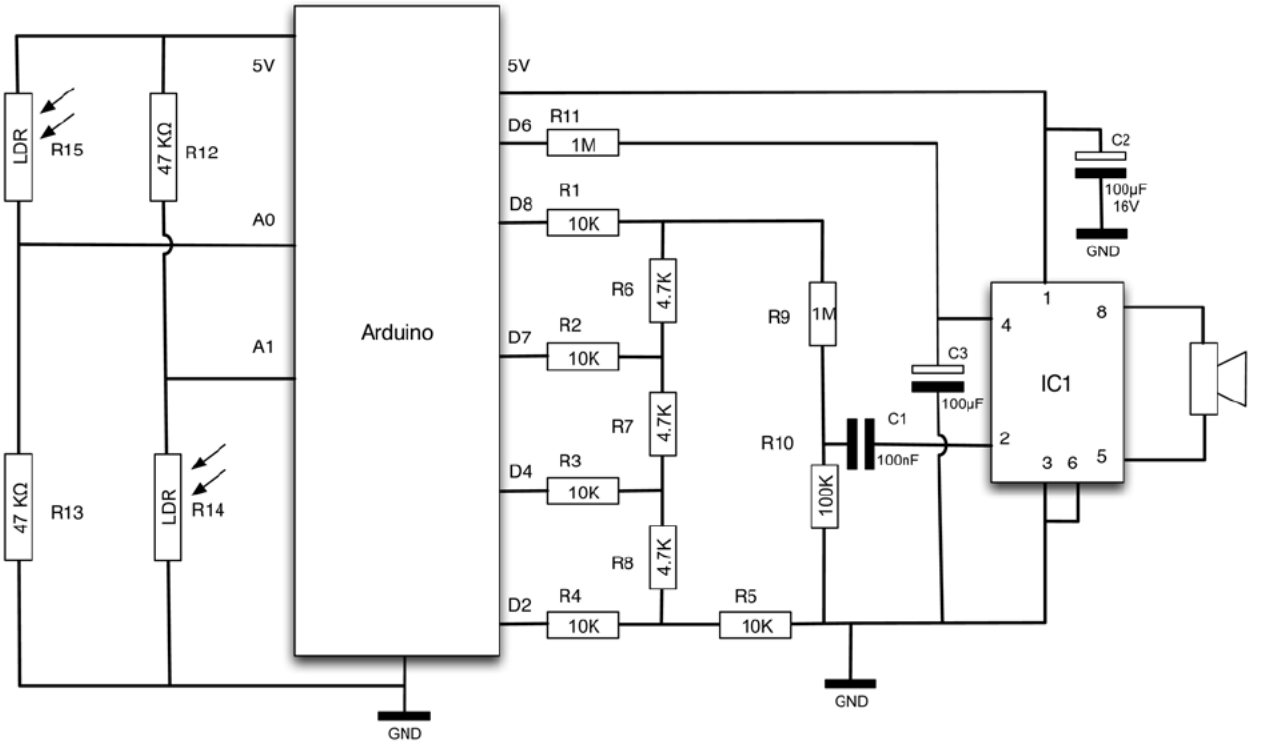


Figure 7-10 Schematic diagram for Project 20.

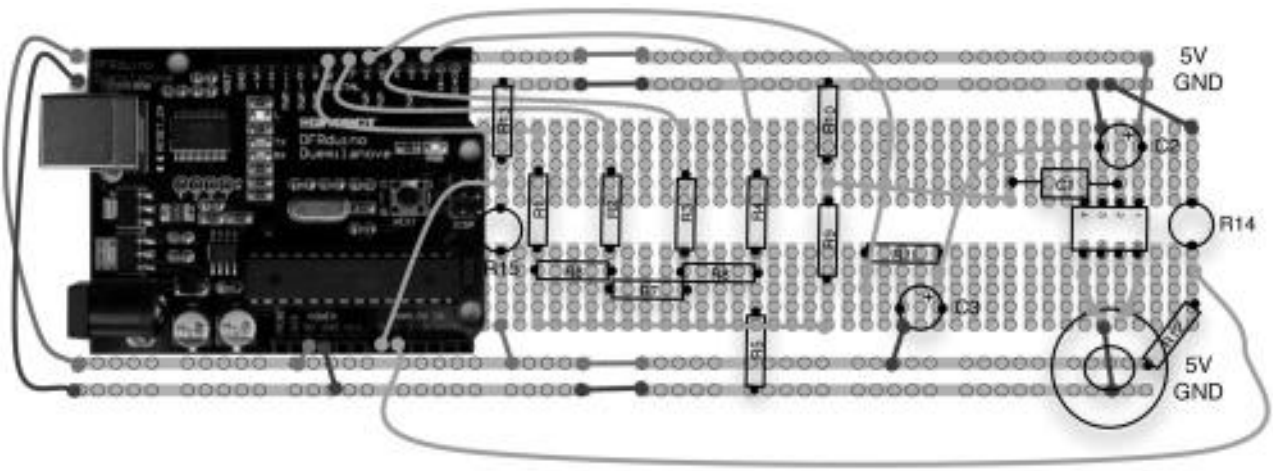


Figure 7-11 Breadboard layout for Project 20.

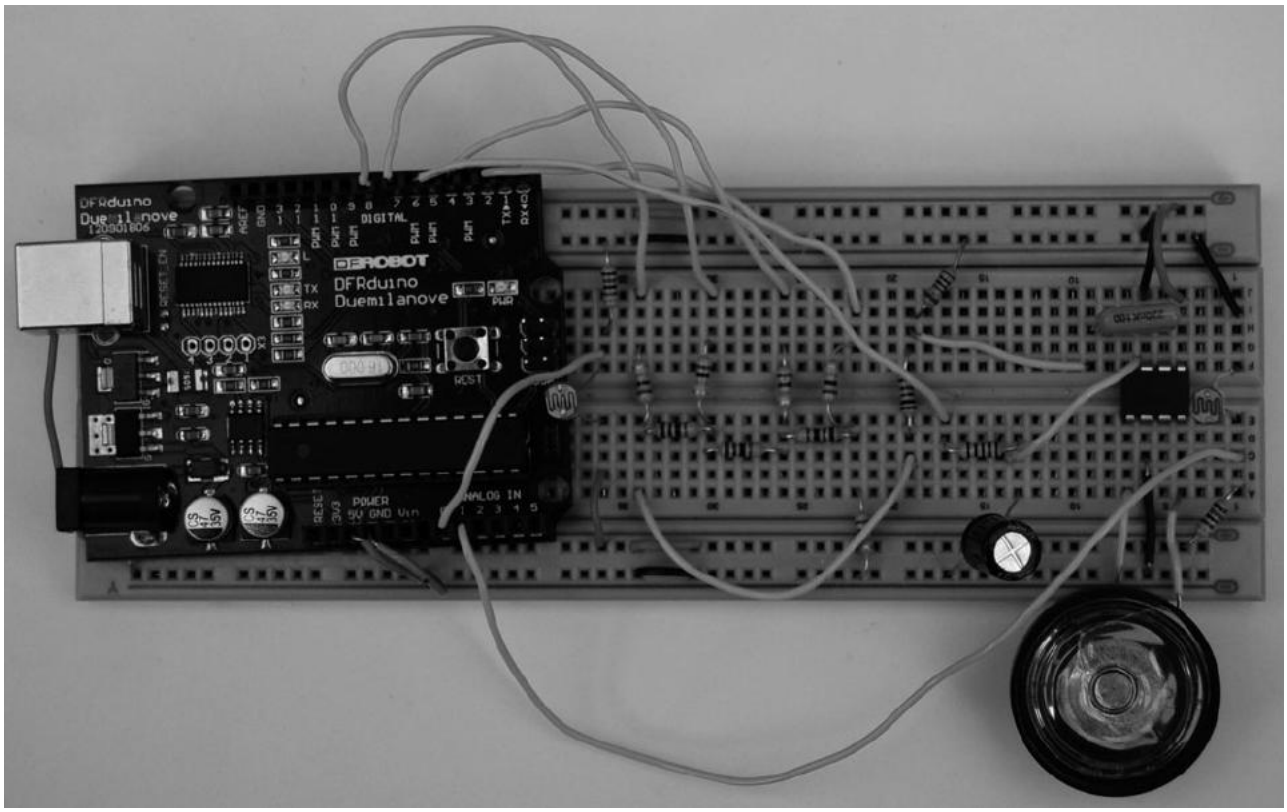


Figure 7-12 Project 20. Light harp.

LISTING PROJECT 20

```
int pitchInputPin = 0;
int volumeInputPin = 1;
int volumeOutputPin = 6;

int dacPins[] = {2, 4, 7, 8};
int sin16[] = {7, 8, 10, 11, 12, 13, 14, 14, 15, 14, 14, 13, 12, 11,
               10, 8, 7, 6, 4, 3, 2, 1, 0, 0, 0, 0, 0, 1, 2, 3, 4, 6};

int count = 0;

void setup()
{
  for (int i = 0; i < 4; i++)
  {
    pinMode(dacPins[i], OUTPUT);
  }
  pinMode(volumeOutputPin, OUTPUT);
}
```

(continued)

LISTING PROJECT 20 (continued)

```
void loop()
{
  int pitchDelay = map(analogRead(pitchInputPin), 0, 1023, 10, 60);
  int volume = map(analogRead(volumeInputPin), 0, 1023, 10, 70);
  for (int i = 0; i < 32; i++)
  {
    setOutput(sin16[i]);
    delayMicroseconds(pitchDelay);
  }
  if (count == 10)
  {
    analogWrite(volumeOutputPin, volume);
    count = 0;
  }
  count++;
}

void setOutput(byte value)
{
    digitalWrite(dacPins[3], ((value & 8) > 0));
    digitalWrite(dacPins[2], ((value & 4) > 0));
    digitalWrite(dacPins[1], ((value & 2) > 0));
    digitalWrite(dacPins[0], ((value & 1) > 0));
}
```

Putting It All Together

Load the completed sketch for Project 20 from your Arduino Sketchbook and download it to the board (see Chapter 1).

To play the "instrument," use the right hand over one LDR to control the volume of the sound and the left hand over the other LDR to control the pitch. Interesting effects can be achieved by waving your hands over the LDRs.

Note that you may need to tweak the values in the map functions in the sketch, depending on the ambient light.

Project 21
VU Meter

This project (shown in Figure 7-13) uses LEDs to display the volume of noise picked up by a microphone. It uses an array of LEDs built into a dual-in-line (DIL) package.

The push button toggles the mode of the VU meter. In normal mode, the bar graph just flickers up and down with the volume of sound. In maximum mode, the bar graph registers the maximum value and lights that LED, so the sound level gradually pushes it up.

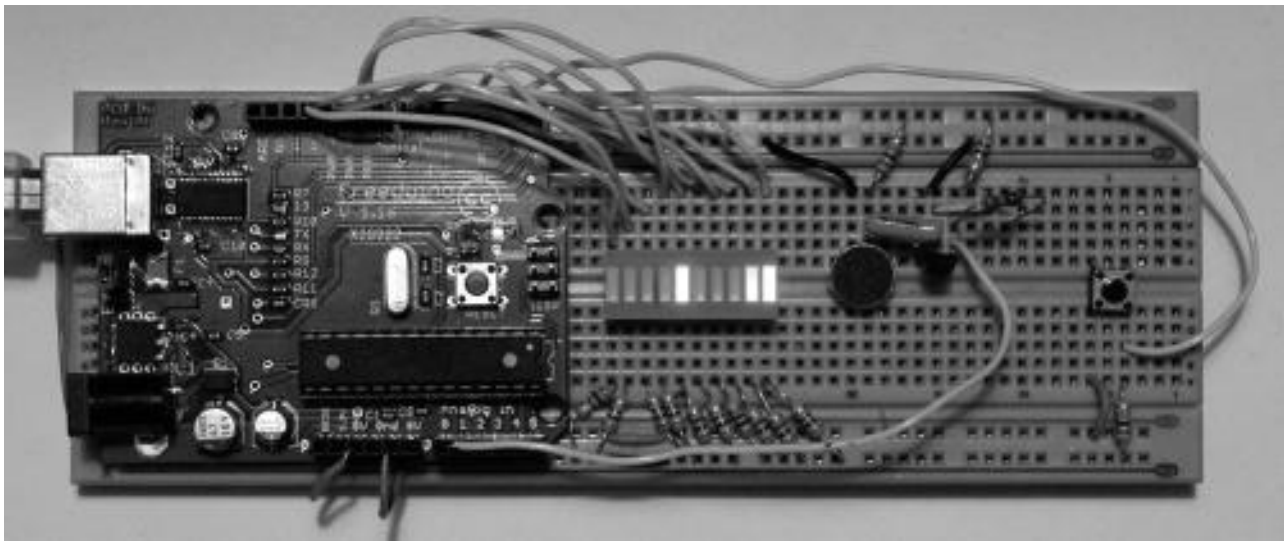


Figure 7-13 Project 21. VU meter.

COMPONENTS AND EQUIPMENT

	Description	Appendix
	Arduino Diecimila or Duemilanove board or clone	1
R1, R3, R4	10 KΩ 0.5W metal film resistor	9
R2	100 KΩ 0.5W metal film resistor	13
R5-14	270 Ω 0.5W metal film resistor	6
R10	10 KΩ 0.5W metal film resistor	9
C1	100 nF	20
	10 Segment bar graph display	35
S1	Push to make switch	48
	Electret microphone	60

Hardware

The schematic diagram for this project is shown in Figure 7-14. The bar graph LED package has separate connections for each LED. These are each driven through a current-limiting resistor.

The microphone will not produce a strong enough signal on its own to drive the analog input. So to boost the signal, we use a simple single-transistor amplifier. We use a standard arrangement called collector-feedback bias, where a proportion of the voltage at the collector is used to bias the transistor on so that it amplifies in a loosely linear way rather than just harshly switching on and off.

The breadboard layout is shown in Figure 7-15. With so many LEDs, a lot of wires are required.

Software

The sketch for this project (Listing Project 21) uses an array of LED pins to shorten the setup function. This is also used in the loop function, where we iterate over each LED deciding whether to turn it on or off.

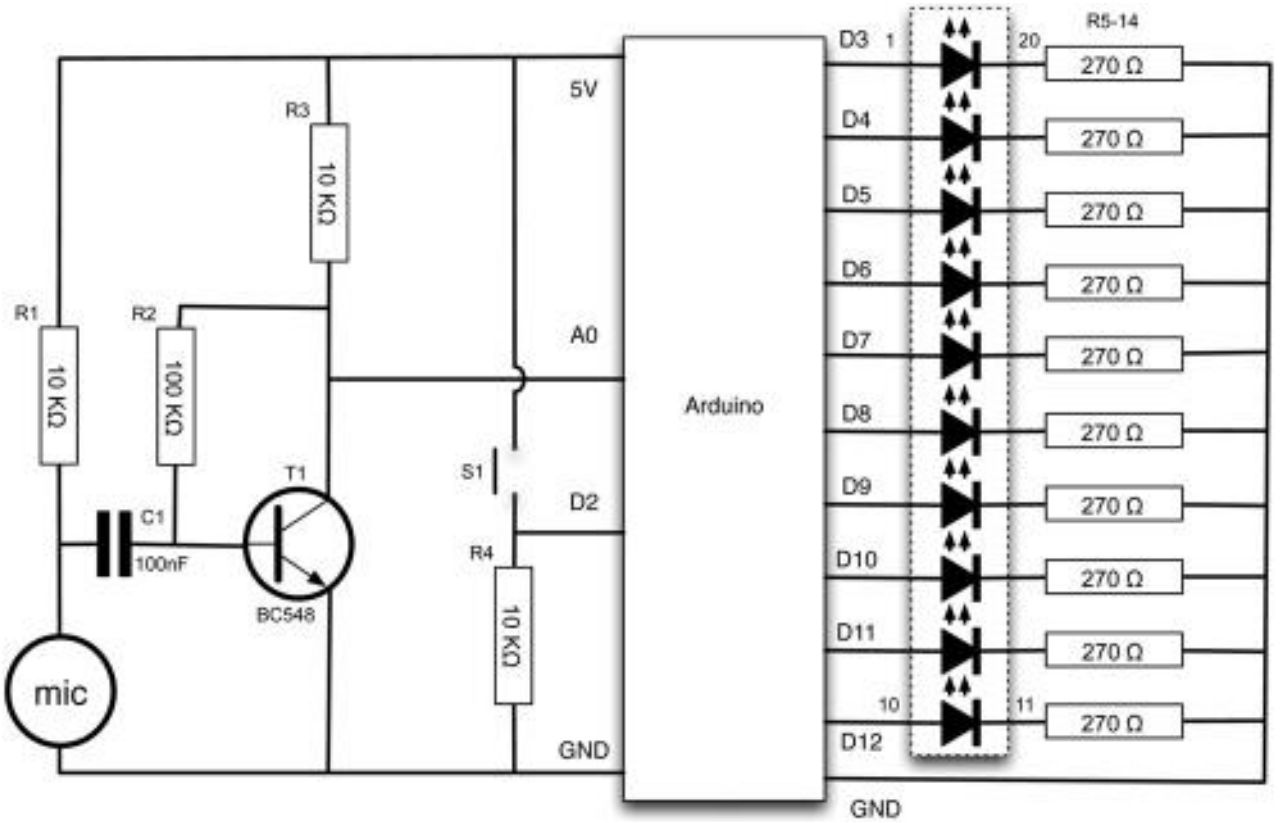


Figure 7-14 Schematic diagram for Project 21.

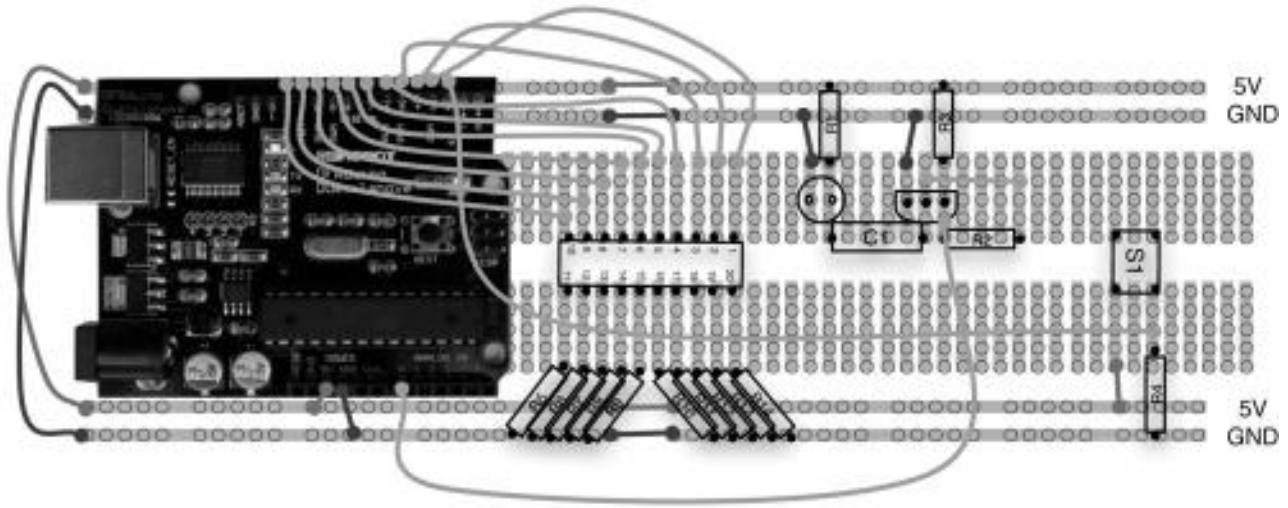


Figure 7-15 Breadboard layout for Project 21.

LISTING PROJECT 21

```
int ledPins[] = {3, 4, 5, 6, 7, 8, 9, 10, 11, 12};
int switchPin = 2;
int soundPin = 0;

boolean showPeak = false;
int peakValue = 0;

void setup()
{
  for (int i = 0; i < 10; i++)
  {
     pinMode(ledPins[i], OUTPUT);
  }
  pinMode(switchPin, INPUT);
}

void loop()
{
  if (digitalRead(switchPin))
  {
    showPeak = ! showPeak;
    peakValue = 0;
    delay(200); // debounce switch
  }
  int value = analogRead(soundPin);
  int topLED = map(value, 0, 1023, 0, 11) - 1;
  if (topLED > peakValue)
  {
    peakValue = topLED;
  }
  for (int i = 0; i < 10; i++)
  {
      digitalWrite(ledPins[i], (i <= topLED || (showPeak && i == peakValue)));
  }
}
```

At the top of the loop function, we check to see if the switch is depressed; if it is, we toggle the mode. The ! command inverts a value, so it will turn true into false and false into true. For this reason, it is sometimes referred to as the "marketing operator." After changing the mode, we reset the maximum value to 0 and then delay for 200 milliseconds to prevent keyboard bounce from changing the mode straight back again.

The level of sound is read from analog pin 0, and then we use the map function to convert from a range of 0 to 1023 down to a number between 0 and 9, which will be the top LED to be lit. This is adjusted slightly by extending the range up to 0 to 11 and then subtracting 1. This prevents the two bottom-most LEDs being permanently lit due to the transistor bias.

We then iterate over the numbers 0 to 9 and use a Boolean expression that returns true (and hence lights the LED) if "i" is less than or equal to the top LED. It is actually more complicated than that because we also should display that LED if we are in peak mode and that LED happens to be the peakValue.

Putting It All Together

Load the completed sketch for Project 21 from your Arduino Sketchbook and download it to the board (see Chapter 1).

Summary

That project concludes our sound-based projects. In the next chapter we go on to look at how we use an Arduino board to control power—a topic always close to the heart of the Evil Genius.

CHAPTER 8

Power Projects

Having looked at light and sound, the Evil Genius now turns their attention to controlling power. In essence, that means turning things on and off and controlling their speed. This mostly applies to motors and lasers and the long-awaited Servo-Controlled Laser project.

Project 22
LCD Thermostat

The temperature in the Evil Genius' lair must be regulated, as the Evil Genius is particularly susceptible to chills. This project uses an LCD screen and a thermistor temperature sensor to both display the current temperature and the set temperature. It uses a rotary encoder to allow the set temperature to be changed. The rotary encoder's button also acts as an override switch.

When the measured temperature is less than the set temperature, a relay is activated. Relays are old-fashioned electromagnetic components that activate a mechanical switch when a current flows through a coil of wire. They have a number of advantages. First, they can switch high currents and voltages, making them suitable for controlling mains equipment. They also electrically isolate the control side (the coil) from the switching side so that the high and low voltages never meet, which is definitely a good thing.

If the reader decides to use this project to switch mains electricity, they should only do so if they really know what they are doing and exercise extreme caution. Mains electricity is very dangerous and kills about 500 people a year in the United States alone. Many more suffer painful and damaging burns.

COMPONENTS AND EQUIPMENT

	Description	Appendix
	Arduino Diecimila or Duemilanove board or clone	1
R1	33 KΩ thermistor beta = 4090	18
R2	33 KΩ 0.5W metal film resistor	10
R3-5	100 KΩ 0.5W metal film resistor	13
R6	270 Ω 0.5W metal film resistor	6
R7	1 KΩ 0.5W metal film resistor	7
D1	5-mm red LED	23
D2	1N4004	38
T1	BC548	40
	5V relay	61
	LCD module HD44780	58
	Header pin strip	55

Hardware

The LCD module is connected up in exactly the same way as Project 17. The rotary encoder is also connected up in the same way as previous projects.

The relay will require about 70 mA, which is a bit too much for an Arduino output to handle unaided, so we use an NPN transistor to increase the current. You will also notice that a diode is connected in parallel with the relay coil. This is to prevent something called back EMF (electromotive force), which occurs when the relay is turned off. The sudden collapse of the magnetic field in the coil generates a voltage that can be high enough to damage the electronics if the diode is not there to effectively short it out if it occurs.

Figure 8-1 shows the schematic diagram for the project.

The breadboard layout for the project is quite cramped, as the LCD module uses a lot of the space.

Check your datasheet for the relay, as the connection pins can be quite counterintuitive and there are several pin layouts, and your layout may not be the same as the relay that the author used.

Figure 8-2 shows the breadboard layout for the project.

You can also use a multimeter to find the coil connections by putting it on resistance mode. They will be the only pair of pins with a resistance of 40 to 100 Ω.

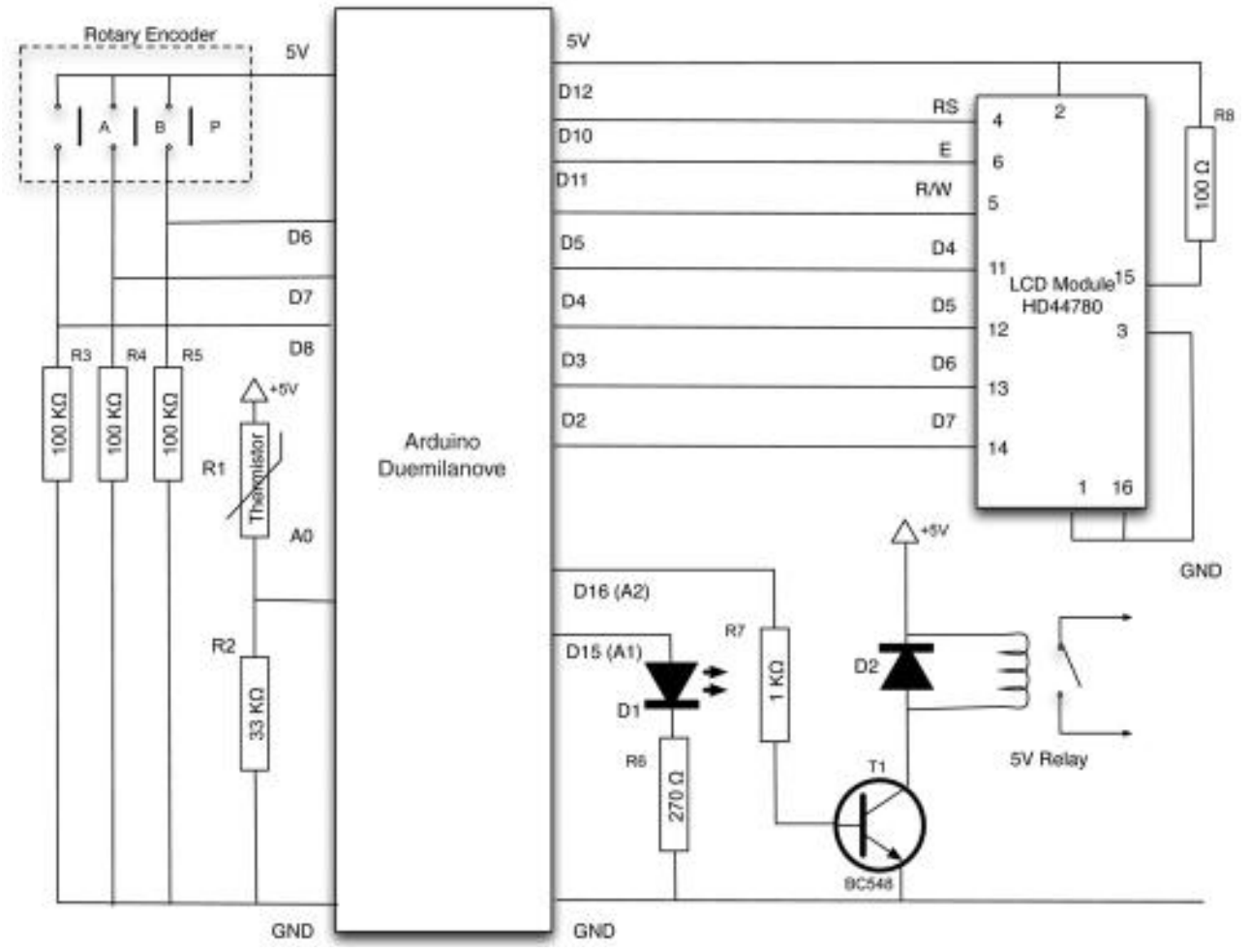


Figure 8-1 Schematic diagram for Project 22.

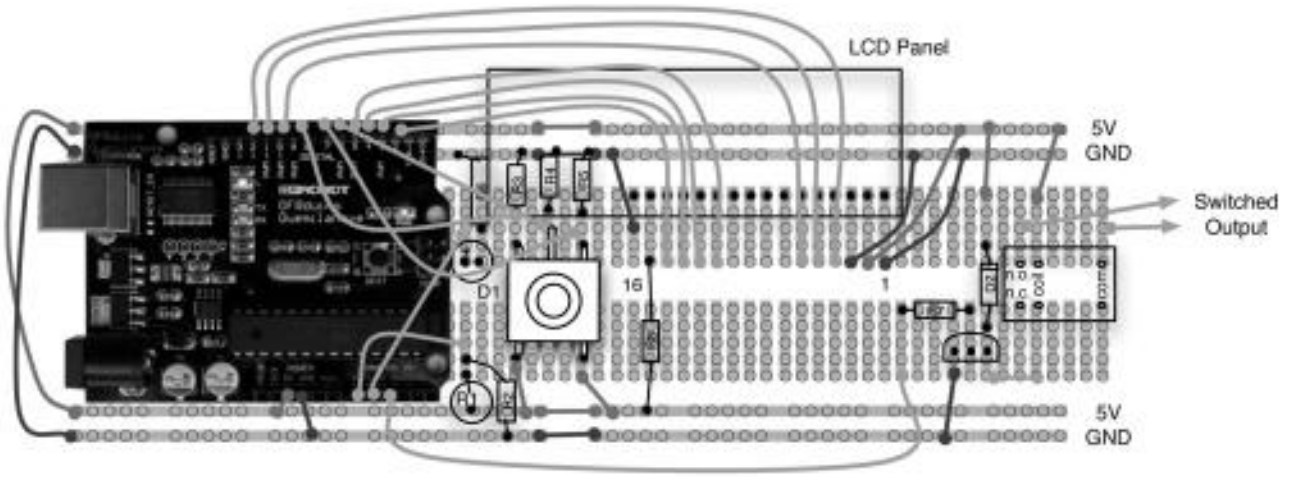


Figure 8-2 Breadboard layout for Project 22.

Software

The software for this project borrows heavily from several of our previous projects: the LCD display, the temperature data logger, and the traffic signal project for use of the rotary encoder (see Listing Project 22).

One thing that requires a bit of consideration when designing a thermostat like this is that you want to avoid what is called "hunting." Hunting

LISTING PROJECT 22

```
#include <LiquidCrystal.h>

#define beta 4090 // from your thermistor's datasheet
#define resistance 33

// LiquidCrystal display with:
// rs on pin 12
// rw on pin 11
// enable on pin 10
// d4-7 on pins 5-2
LiquidCrystal lcd(12, 11, 10, 5, 4, 3, 2);

int ledPin = 15;
int relayPin = 16;
int aPin = 8;
int bPin = 7;
int buttonPin = 6;
int analogPin = 0;

float setTemp = 20.0;
float measuredTemp;
char mode = 'C';             // can be changed to F
```

(continued)

LISTING PROJECT 22 (continued)

```
boolean override = false;
float hysteresis = 0.25;

void setup()
{
  lcd.begin(2, 20);
  pinMode(ledPin, OUTPUT);
  pinMode(relayPin, OUTPUT);
  pinMode(aPin, INPUT);
  pinMode(bPin, INPUT);
  pinMode(buttonPin, INPUT);
  lcd.clear();
}

void loop()
{
  static int count = 0;
  measuredTemp = readTemp();
  if (digitalRead(buttonPin))
  {
    override = ! override;
    updateDisplay();
    delay(500); // debounce
  }
  int change = getEncoderTurn();
  setTemp = setTemp + change * 0.1;
  if (count == 1000)
  {
    updateDisplay();
    updateOutputs();
    count = 0;
  }
  count ++;
}

int getEncoderTurn()
{
  // return -1, 0, or +1
  static int oldA = LOW;
  static int oldB = LOW;
  int result = 0;
  int newA = digitalRead(aPin);
  int newB = digitalRead(bPin);
  if (newA != oldA || newB != oldB)
  {
    // something has changed
    if (oldA == LOW && newA == HIGH)
```

LISTING PROJECT 22 (continued)

```
    {
      result = -(oldB * 2 - 1);
    }
  }
  oldA = newA;
  oldB = newB;
  return result;
}

float readTemp()
{
  long a = analogRead(analogPin);
  float temp = beta / (log(((1025.0 * resistance / a) - 33.0) / 33.0) +
       (beta / 298.0)) - 273.0;
  return temp;
}

void updateOutputs()
{
  if (override ||  measuredTemp < setTemp - hysteresis)
  {
    digitalWrite(ledPin, HIGH);
    digitalWrite(relayPin, HIGH);
  }
  else if (!override && measuredTemp > setTemp + hysteresis)
  {
    digitalWrite(ledPin, LOW);
    digitalWrite(relayPin, LOW);
  }
}

void updateDisplay()
{
  lcd.setCursor(0,0);
  lcd.print("Actual: ");
  lcd.print(adjustUnits(measuredTemp));
  lcd.print(" o");
  lcd.print(mode);
  lcd.print(" ");

  lcd.setCursor(0,1);
  if (override)
  {
    lcd.print("  OVERRIDE ON   ");
  }
  else
```

(continued)

LISTING PROJECT 22 (continued)

```
  {
    lcd.print("Set:    ");
    lcd.print(adjustUnits(setTemp));
    lcd.print(" o");
    lcd.print(mode);
    lcd.print(" ");
  }
}

float adjustUnits(float temp)
{
  if (mode == 'C')
  {
    return temp;
  }
  else
  {
    return (temp * 9) / 5 + 32;
  }
}
```

occurs when you have a simple on-off control system. When the temperature falls below the set point, the power is turned on and the room heats until it is above the set point, and then the room cools until the temperature is below the set point again, at which point the heat is turned on again, and so on. This may take a little time to happen, but when the temperature is just balanced at the switch-over temperature, this hunting can be frequent. High-frequency switching like this is undesirable because turning things on and off tends to wear them out. This is true of relays as well.

One way to minimize this effect is to introduce something called hysteresis, and you may have noticed a variable called hysteresis in the sketch that is set to a value of 0.25°C.

Figure 8-3 shows how we use a hysteresis value to prevent high-frequency hunting.

As the temperature rises with the power on, it approaches the set point. However, it does not turn off the power until it has exceeded the set point plus the hysteresis value. Similarly, as the temperature falls, the power is not reapplied the moment it falls below the set point, but only when it falls below the set point minus the hysteresis value.

We do not want to update the display continuously, as any tiny changes in the reading would result in the display flickering wildly. So

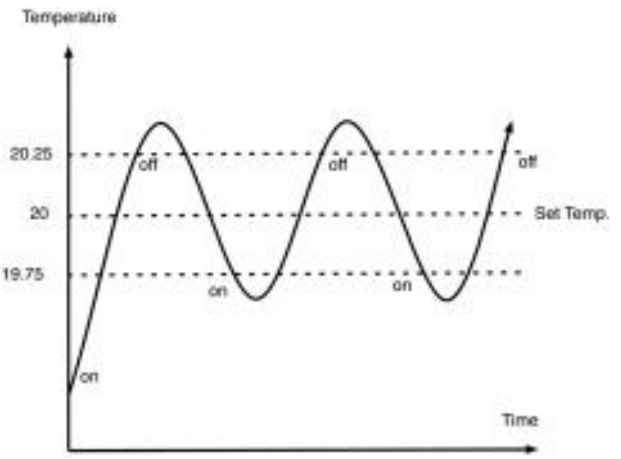


Figure 8-3 Hysteresis in control systems.

instead of updating the display every time round the main loop, we just do it one time in 1000. This still means it will update three or four times per second. To do this, we use the technique of having a counter variable that we increment each time round the loop. When it gets to 1000, we update the display and reset the counter to 0.

Using lcd.clear() each time we change the display would also cause it to flicker. So we simply write the new temperatures on top of the old temperatures. This is why we pad the "OVERRIDE ON" message with spaces so that any text that was previously displayed at the edges will be blanked out.

Putting It All Together

Load the completed sketch for Project 22 from your Arduino Sketchbook and download it to the board (see Chapter 1).

The completed project is shown in Figure 8-4. To test the project, turn the rotary encoder, setting the set temperature to slightly above the actual temperature. The LED should be on. Then put your finger onto the thermistor to warm it up. If all is well, then when the set temperature is exceeded, the LED should turn off and you will hear the relay click.

You can also test the operation of the relay by connecting a multimeter in continuity test (beep) mode to the switched output leads.

I cannot stress enough that if you intend to use your relay to switch mains electricity, first put this project onto a properly soldered Protoshield. Second, be very careful and check and double-check what you are doing. Mains electricity kills.

You *must* only test the relay with low voltage unless you are going to make a proper soldered project from this design.

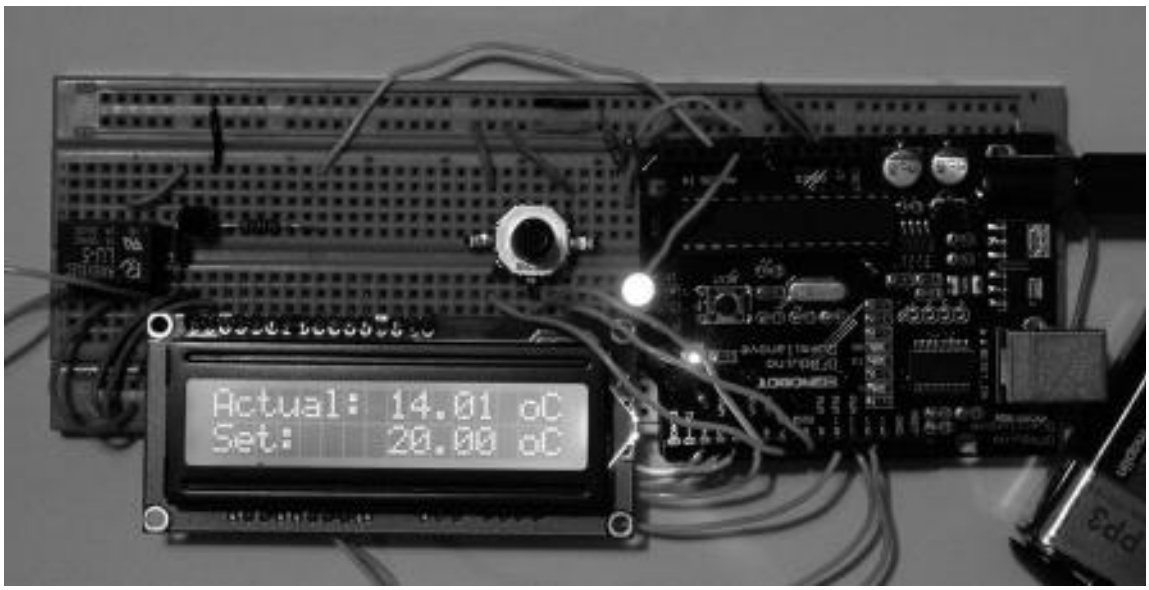


Figure 8-4 Project 22. LCD thermostat.

Project 23
Computer-Controlled Fan

One handy part to reclaim from a dead PC is the case fan (Figure 8-5). We are going to use one of these fans to keep ourselves cool in the summer. Obviously, a simple on/off switch would not be in keeping with the Evil Genius' way of doing things, so the speed of the fan will be controllable from our computer.

COMPONENTS AND EQUIPMENT

	Description	Appendix
	Arduino Diecimila or Duemilanove board or clone	1
R1	270 Ω 0.5W metal film resistor	6
T1	BD139 power transistor	41
M1	12V computer cooling fan	63
	12V 1 A power supply	62

If you do not happen to have a dead computer lying around, fear not, because you can buy new cooling fans quite cheaply.

Hardware

We can control the speed of the fan using the analog output (PWM) driving a power transistor to pulse the motor. Since these computer fans are usually 12V, we will use an external power supply to provide the drive power for the fan.

Figure 8-6 shows the schematic diagram for the project and Figure 8-7 the breadboard layout.

Software

This is a really simple sketch (Listing Project 23). Essentially, we just need to read a digit 0 to 9 from USB and do an analogWrite to the motorPin of that value, multiplied by 28, to scale it up to a number between 0 and 252.

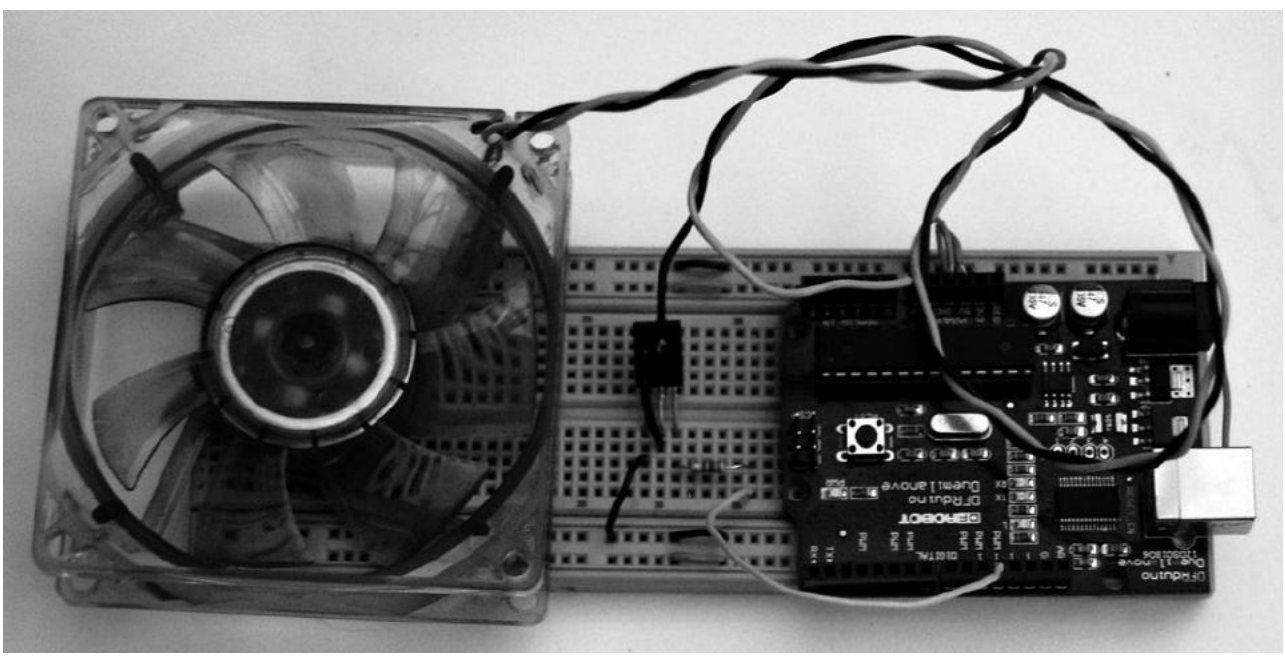


Figure 8-5 Project 23. Computer-controlled fan.

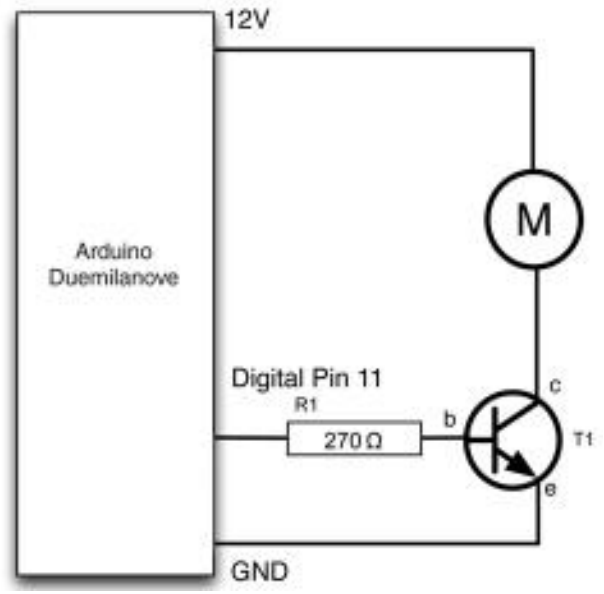


Figure 8-6 Schematic diagram for Project 23.

LISTING PROJECT 23

```
int motorPin = 11;

void setup()
{
  pinMode(motorPin, OUTPUT);
  analogWrite(motorPin, 0);
  Serial.begin(9600);
}

void loop()
{
  if (Serial.available())
  {
    char ch = Serial.read();
    if (ch >= '0' && ch <= '9')
    {
       int speed = ch - '0';
       analogWrite(motorPin, speed
            * 28);
    }
  }
}
```

Putting It All Together

Load the completed sketch for Project 23 from your Arduino Sketchbook and download it to the board (see Chapter 1).

There are so few components in this project that you could twist a few wires together and fit them directly into the Arduino board, thus doing away with the breadboard altogether.

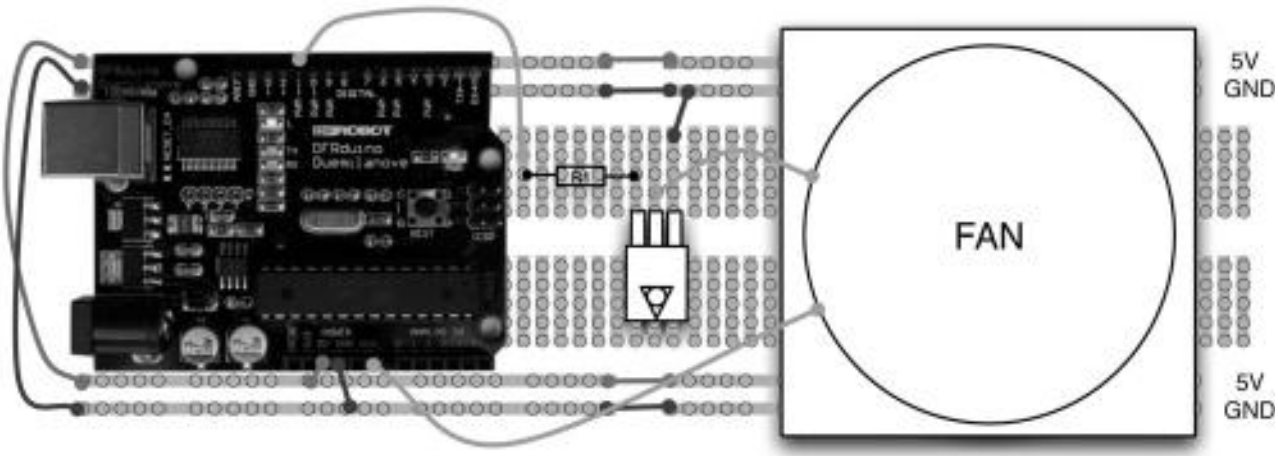


Figure 8-7 Breadboard layout for Project 23.

H-Bridge Controllers

To change the direction in which a motor turns, you have to reverse the direction in which the current flows. To do this requires four switches or transistors. Figure 8-8 shows how this works, using switches in an arrangement that is, for obvious reasons, called an H-bridge.

In Figure 8-8, S1 and S4 are closed and S2 and S3 are open. This allows current to flow through the motor with terminal A being positive and terminal B being negative. If we were to reverse the switches so that S2 and S3 are closed and S1 and S4 are open, then B will be positive and A will be negative and the motor will turn in the opposite direction.

However, you may have spotted a danger with this circuit. That is, if by some chance S1 and S2 are both closed, then the positive supply will be directly connected to the negative supply and we will have a short-circuit. The same is true if S3 and S4 are both closed at the same time.

In the following project, we will use transistors in place of the switches to control an electric motor.

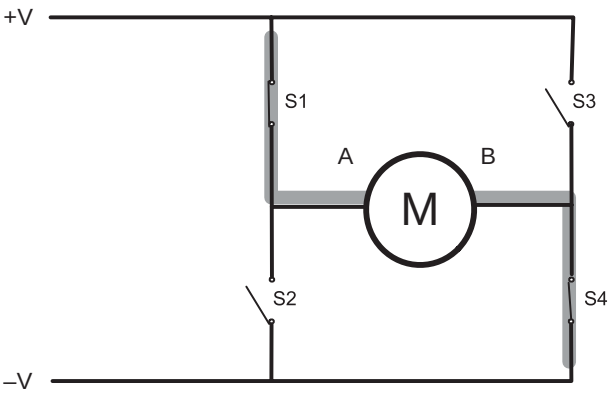


Figure 8-8 An H-bridge.

Project 24
Hypnotizer

Mind control is one of the Evil Genius' favorite things. This project (see Figure 8-9) takes complete control of a motor to not only control its speed, but also to make it turn clockwise and counterclockwise. Attached to the motor will be a swirling spiral disk intended to mesmerize the unfortunate victims.

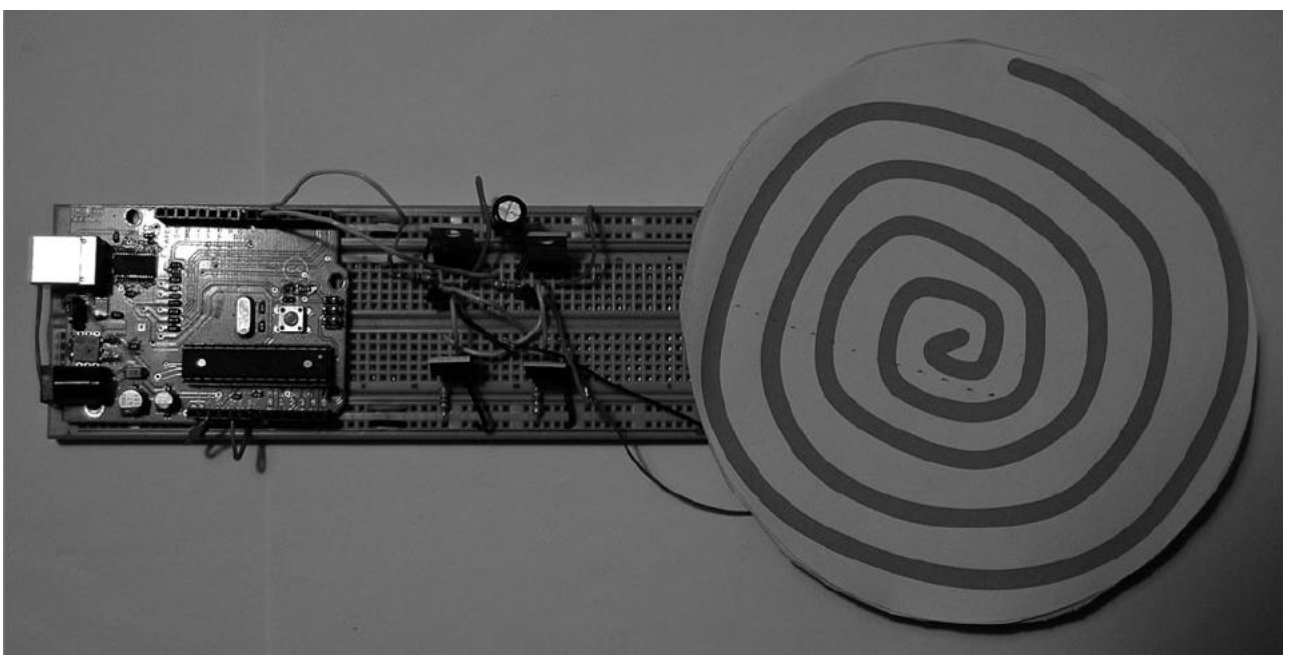


Figure 8-9 Project 24. The hypnotizer.

COMPONENTS AND EQUIPMENT

	Description	Appendix
	Arduino Diecimila or Duemilanove board or clone	1
T2, T4	N-channel power MOSFET. FQP33N10	43
T1, T3	P-channel power MOSFET. FQP27P06	44
T4, T5	BC548	40
R1-6	10 KΩ 0.5W metal film resistor	9
M1	6V motor	64

The motor that we used in this project was reclaimed from a broken computer CD drive. An alternative cheap source for a motor would be an old motorized child's toy. One with gears driving a wheel would lend itself particularly well to having the hypnotic disk attached.

Hardware

The schematic diagram for the hypnotizer is shown in Figure 8-10. It uses a standard H-bridge arrangement. Note that we are using metal oxide semiconductor field effect transistors (MOSFETs) rather than bipolar transistors for the main power control. In theory, this will allow us to control quite high-powered motors, but also has the advantage in that the MOSFETs will barely even get warm with our little motor and, therefore, we will not need heatsinks.

The gate connections of the bottom MOSFETs are cunningly connected to the outputs of their diagonally opposite transistors, so when T1 turns on, T4 will automatically turn on with it; when T3 turns on, T2 will turn on with it.

The resistors R1 to R4 ensure that the default state of T1 to T4 is off by pulling the gates of the P-channel MOSFETS high and the N-channel low.

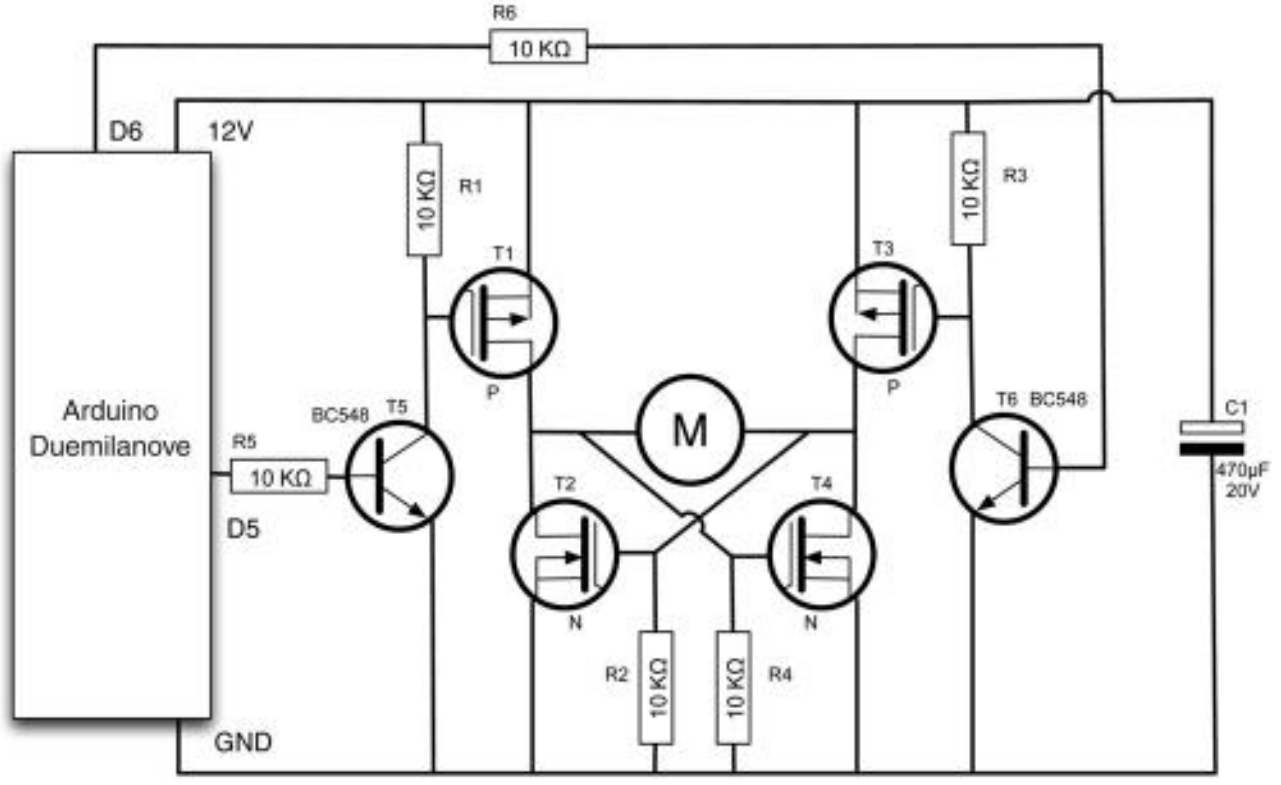


Figure 8-10 Schematic diagram for Project 24.

T5 and T6 are low-current bipolar transistors that are used to turn on T1 and T3, respectively. In this case, we could do without these transistors and drive the gates of T1 and T3 directly from the Arduino board. However, to do this, the logic would be inverted (high at the gate would turn the transistor off). We could cope with this in the software, but the other reason for using these two additional components is that with them, we could use this circuit to control higher voltage motors, simply by using a higher positive supply voltage. If we were driving the MOSFETS directly, then the positive output of the Arduino would have to go higher than 5V to turn off the MOSFET if the motor supply was 9V or more, something that is not possible.

This does make the circuit overengineered, but the Evil Genius may have big ambitions on the motor front!

Finally, C1 smoothes out some of the pulses of power use that you get when you drive a device like a motor.

Figure 8-11 shows the breadboard layout for the project.

Our hypnotizer needs a spiral pattern to work. You may decide to photocopy Figure 8-12, cut it out, and stick it to the fan. Alternatively, a more colorful version of the spiral is available to print out from www.arduinoevilgenius.com.

Figure 8-12 Spiral for the hypnotizer.

The spiral was cut out of paper and stuck onto cardboard that was then glued on to the little cog on the end of the motor.

Software

The key thing about this sketch (Listing Project 24) is to make it impossible for all the transistors to be on at the same time. If this happens, there will be a burning smell and something somewhere will fizzle and die.

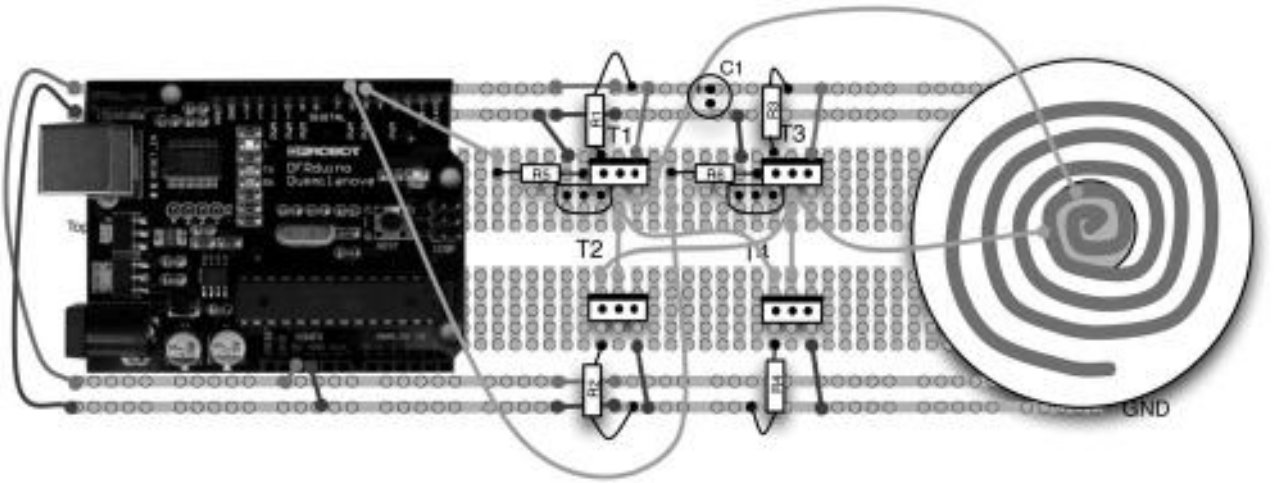


Figure 8-11 Breadboard layout for Project 24.

LISTING PROJECT 24

```
int t1Pin = 5;
int t3Pin = 6;

int speeds[] = {20, 40, 80, 120, 160, 180, 160, 120, 80, 40, 20,
                  -20, -40, -80, -120, -160, -180, -160, -120, -80, -40, -20};
int i = 0;

void setup()
{
  pinMode(t1Pin, OUTPUT);
  digitalWrite(t1Pin, LOW);
  pinMode(t3Pin, OUTPUT);
  digitalWrite(t3Pin, LOW);
}

void loop()
{
  int speed = speeds[i];
  i++;
  if (i == 22)
  {
    i = 0;
  }
  drive(speed);
  delay(1500);
}

void allOff()
{
  digitalWrite(t1Pin, LOW);
  digitalWrite(t3Pin, LOW);
  delay(1);
}

void drive(int speed)
{
  allOff();
  if (speed > 0)
  {
    analogWrite(t1Pin, speed);
  }
  else if (speed < 0)
  {
    analogWrite(t3Pin,  -speed);
  }
}
```

Before turning any transistor on, all the transistors are turned off using the allOff function. In addition, the allOff function includes a slight delay to ensure that the transistors are properly off before anything is turned on.

The sketch uses an array, speeds, to control the disk's progression in speed. This makes the disk spin faster and faster in one direction, and then slow until it eventually reverses direction and then starts getting faster and faster in that direction and so on. You may need to adjust this array for your particular motor. The speeds you will need to specify in the array will vary from motor to motor, so you will probably need to adjust these values.

Putting It All Together

Load the completed sketch for Project 24 from your Arduino Sketchbook and download it to the board (see Chapter 1).

Take care to check your wiring before applying power on this project. You can test each path through the H-bridge by connecting the control wires that go to digital pins 5 and 6 to ground. Then connect one of the leads to 5V and the motor should turn one way. Connect that lead back to ground and then connect the other lead to 5V and the motor should rotate the other way.

Servo Motors

Servo motors are great little components that are often used in radio-controlled cars to control steering and the control surfaces on model aircraft. They come in a variety of sizes for different types of applications, and their wide use in models makes them relatively inexpensive.

Unlike normal motors, they do not rotate continuously; rather, you set them to a particular angle using a PWM signal. They contain their own control electronics to do this, so all you have to provide them with is power (which, for many devices, can be 5V) and a control signal that we can generate from the Arduino board.

Over the years, the interface to servos has become standardized. The servo must receive a continuous stream of pulses at least every 20 milliseconds. The angle that the servo maintains is determined by the pulse width. A pulse width of 1.5 milliseconds will set the servo at its midpoint, or 90 degrees. A pulse of 1.75 milliseconds will normally swing it round to 180 degrees, and a shorter pulse of 1.25 milliseconds will set the angle to 0 degrees.

Project 25
Servo-Controlled Laser

This project (see Figure 8-13) uses two servo motors to aim a laser diode. It can move the laser quite quickly, so you can "write" on distant walls using it.

This is a real laser. It is not high-powered, only 3 mW, but nonetheless, do not shine the beam in your own or anybody else's eyes. To do so could cause retina damage.

COMPONENTS AND EQUIPMENT

	Description	Appendix
	Arduino Diecimila or Duemilanove board or clone	1
D1	3 mW red laser diode	32
M1, M2	9g servo motor	65
R1	100 Ω 0.5W metal film resistor	5
	Arduino Protoshield (optional)	3
	0.1-inch header strip (6 pins) (optional)	55
	0.1-inch socket strip (2 sockets) (optional)	56

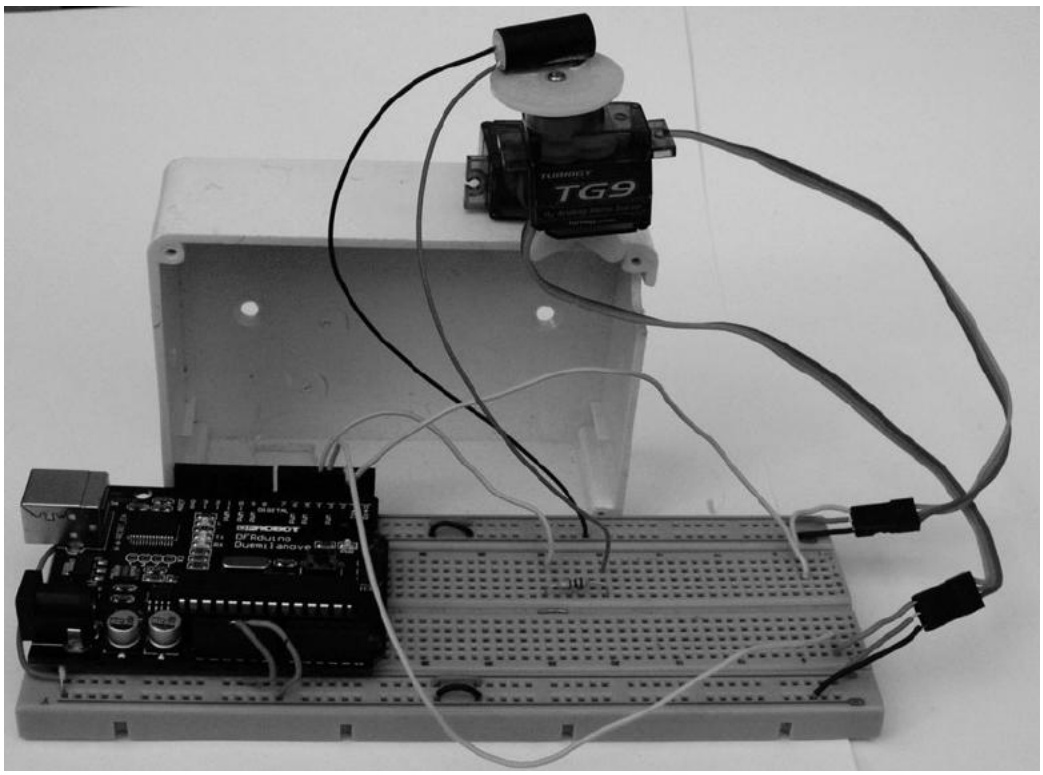


Figure 8-13 Project 25. Servo-controlled laser.

Hardware

The schematic diagram for the project is shown in Figure 8-14. It is all quite simple. The servos have just three leads. For each servo, the brown lead is connected to ground, the red lead to +5V, and the orange (control) lead to digital outputs 2 and 3. The servos are terminated in sockets designed to fit over a pin header. Solid-core wire can be used to connect these to the breadboard.

The laser diode is driven just like an ordinary LED from D4 via a current-limiting resistor.

The servos are usually supplied with a range of "arms" that push onto a cogged drive and are secured by a retaining screw. One of the servos is glued onto one of these arms (see Figure 8-15). Then the arm is attached to the servo. Do not fit the retaining screw yet, as you will need to adjust the angle. Glue the laser diode to a second arm and attach that to the servo. It is a good idea to fix some of the wire from the laser to the arm to prevent strain on the wire where it emerges from the laser. You can do this by putting a loop of solid-core wire through two holes in the server arm and twisting it round the lead. You can see this in Figure 8-17.

You now need to attach the bottom servo to a box or something that will provide support. In Figure 8-15, you can see how it is attached to an old project box. Make sure you understand how the servo will move before you glue the bottom servo to anything. If in doubt, wait until you have installed the software and try the project out just holding the bottom servo before you glue it in place. Once you are sure everything is in the right place, fit the retaining screws onto the servo arms.

You can see how the breadboard is used to anchor the various wires in Figure 8-13. There are no components except the resistor on the breadboard.

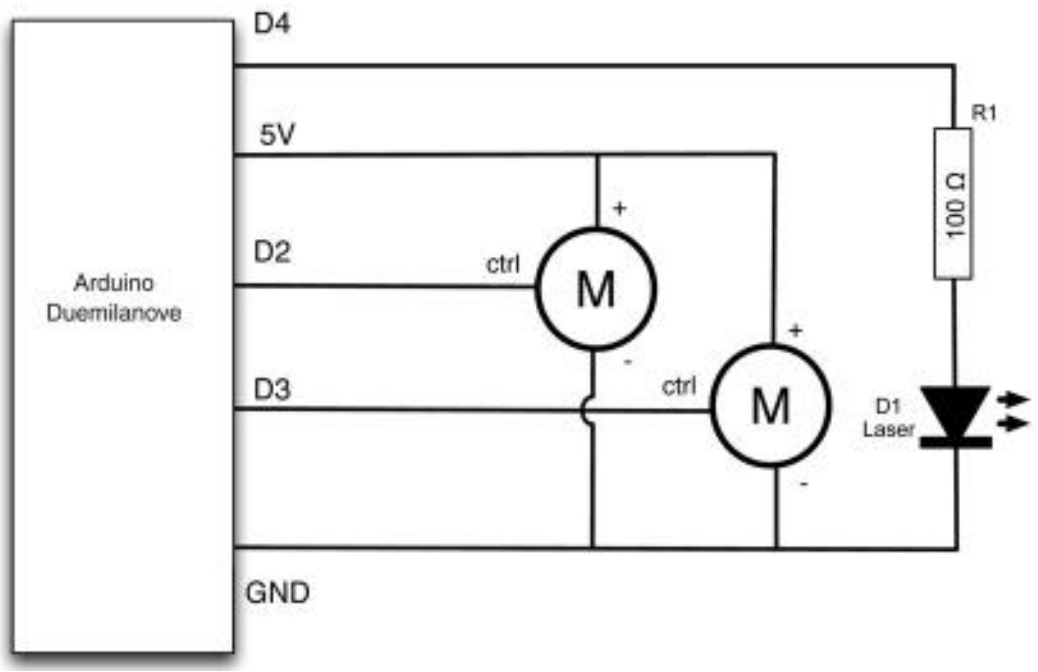


Figure 8-14 Schematic diagram for Project 25.

Software

Fortunately for us, a servo library comes with the Arduino library, so all we need to do is tell each servo what angle to set itself at. There is obviously more to it than that, as we want to have a means of issuing our evil project with coordinates at which to aim the laser.

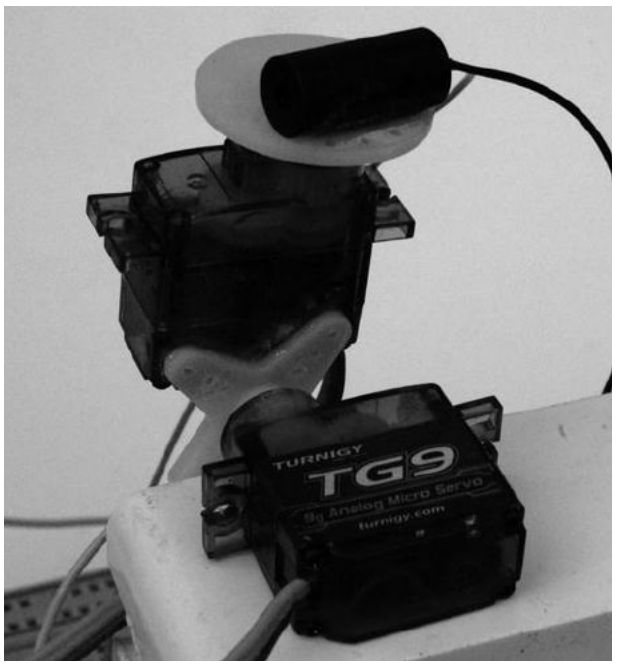


Figure 8-15 Servo and laser assembly.

To do this, we allow commands to be sent over USB. The commands are in the form of letters. R, L, U, and D direct the laser right, left, up, or down, respectively, by five degrees. For finer movements, r, l, u, and d move the laser by just one degree. To pause and allow the laser to finish moving, you can send the – (dash) character. (See Project Listing 25.)

There are three other commands. The letter c will center the laser back at its resting position, and the commands 1 and 0 turn the laser on and off, respectively.

Putting It All Together

Load the completed sketch for Project 25 from your Arduino Sketchbook and download it to the board (see Chapter 1).

LISTING PROJECT 25

```
#include <Servo.h>

int laserPin = 4;
Servo servoV;
Servo servoH;

int x = 90;
int y = 90;
int minX = 10;
int maxX = 170;
int minY = 50;
int maxY = 130;

void setup()
{
  servoH.attach(3);
  servoV.attach(2);
  pinMode(laserPin, OUTPUT);
  Serial.begin(9600);
}

void loop()
{
  char ch;
  if (Serial.available())
  {
    ch = Serial.read();
    if (ch == '0')
    {
      digitalWrite(laserPin, LOW);
    }
    else if (ch == '1')
    {
      digitalWrite(laserPin, HIGH);
    }
    else if (ch == '-')
    {
      delay(100);
    }
    else if (ch == 'c')
    {
      x = 90;
      y = 90;
    }
    else if (ch == 'l' || ch == 'r' || ch == 'u' || ch == 'd')
    {
```

(continued)

LISTING PROJECT 25 (continued)

```
        moveLaser(ch, 1);
      }
      else if (ch == 'L' || ch == 'R' || ch == 'U' || ch == 'D')
      {
        moveLaser(ch, 5);
      }
    }
    servoH.write(x);
    servoV.write(y);
}

void moveLaser(char dir, int amount)
{
    if ((dir == 'r' || dir == 'R') && x > minX)
    {
      x = x - amount;
    }
    else if ((dir == 'l' || dir == 'L') && x < maxX)
    {
      x = x + amount;
    }
    else if ((dir == 'u' || dir == 'U') && y < maxY)
    {
      y = y + amount;
    }
    else if ((dir == 'd' || dir == 'D') && x > minY)
    {
      y = y - amount;
    }
}
```

Open up the Serial Monitor and type the following sequence. You should see the laser trace the letter A, as shown in Figure 8-16:

```
1UUUUUU—RRRR—DDDDDD—0UUU—1LLLL—0DDD
```

Making a Shield

Creating a shield is no problem at all for this project. The bottom servo can be glued in place on one edge of the board. The pin headers are soldered in place near the 5V and GND lines that run down the center of the shield so that they can be connected easily to the positive and negative pins on the servo connectors.

The top and bottom sides of the shield are shown in Figures 8-17 and 8-18.

Summary

In the previous chapters we have built up our knowledge to understand how to use light, sound, and various sensors on the Arduino. We have also learned how to control the power to motors and to use relays. This covers nearly everything we are likely to want to do with our Arduino board, so in the next chapter, we can put all these things together to create some wider-ranging projects.

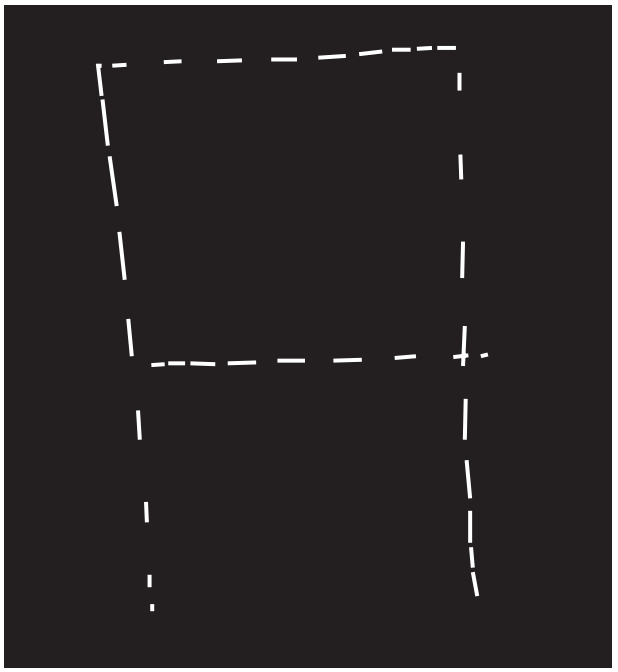


Figure 8-16 Writing the letter A with the laser.

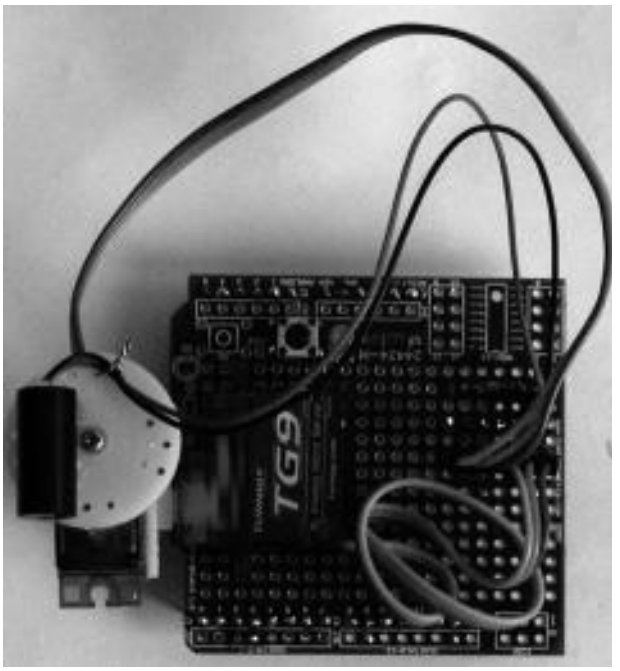


Figure 8-17 Servo laser shield.

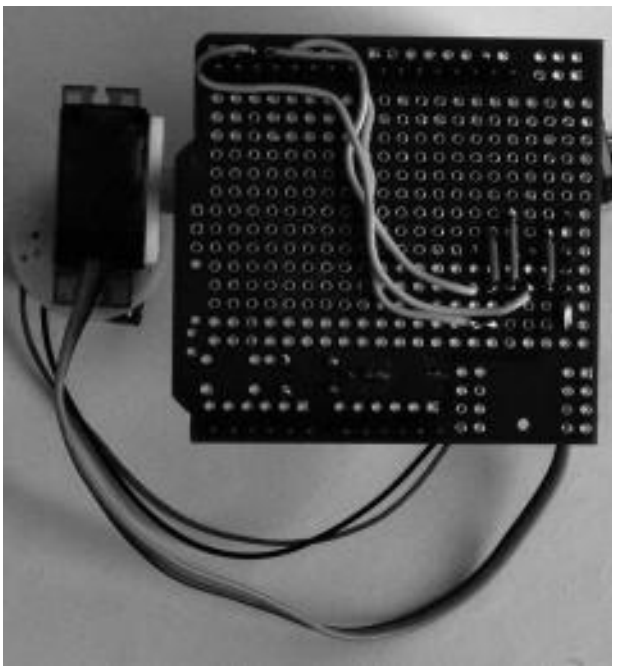


Figure 8-18 Bottom side of the servo laser shield.

This page intentionally left blank

CHAPTER 9

Miscellaneous Projects

This chapter is just a collection of projects that we can build. They do not illustrate any particular point except that Arduino projects are great fun to make.

Project 26
Lie Detector

How can an Evil Genius be sure that their prisoners are telling the truth? By using a lie detector, of course. This lie detector (see Figure 9-1) uses an effect known as galvanic skin response. As a person becomes nervous—for example, when telling a lie—their skin resistance decreases. We can measure this resistance using an analog input and use an LED and buzzer to indicate an untruth.

We use a multicolor LED that will display red to indicate a lie, green to indicate a truth, and blue to show that the lie detector should be adjusted by twiddling the variable resistor.

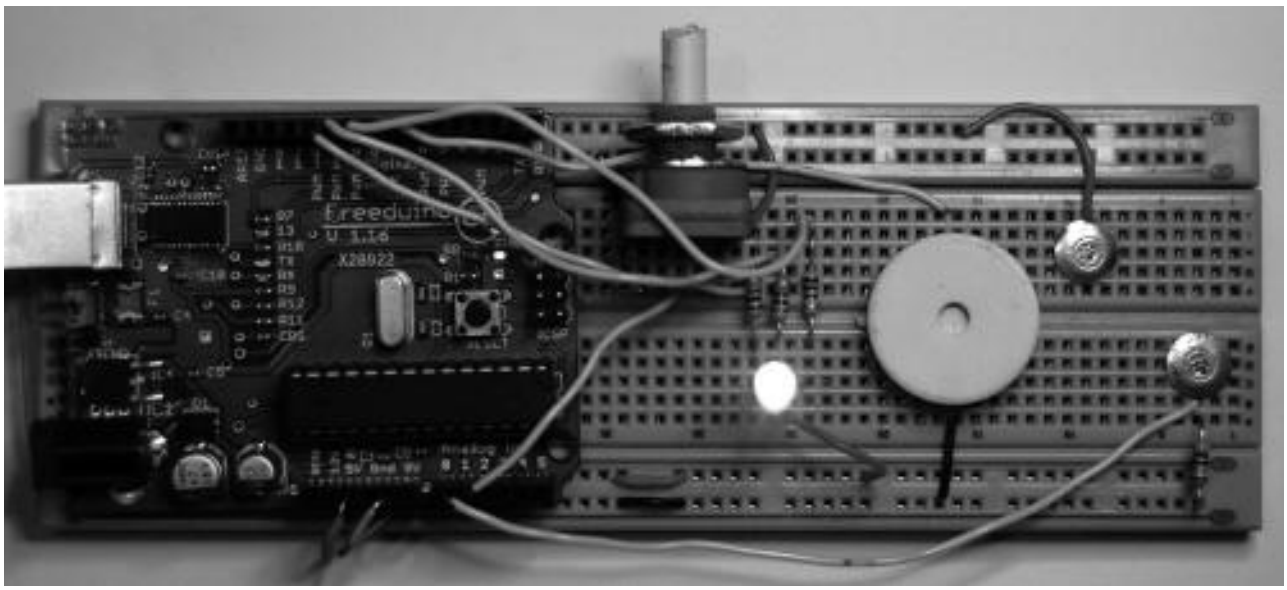


Figure 9-1 Project 26. Lie detector.

COMPONENTS AND EQUIPMENT		
	Description	**Appendix**
	Arduino Diecimila or Duemilanove board or clone	1
R1-3	100 Ω 0.5W metal film resistor	5
R4	470 KΩ 0.5W metal film resistor	14
R5	100 KΩ variable resistor	17
D1	RGB LED (common anode)	31
S1	Piezotransducer (without driver electronics)	67

There are two types of piezobuzzers. Some are just a piezoelectric transducer, while some also include the electronic oscillator to drive them. In this project we want the former type without the electronics, as we are going to generate the necessary frequency from the Arduino board itself.

Hardware

The subject's skin resistance is measured by using the subject as one resistor in a potential divider and a fixed resistor as the other. The lower their resistance, the more analog input 0 will be pulled towards 5V. The higher the resistance, the closer to GND it will become.

The piezobuzzer, despite the level of noise these things generate, is actually quite low in current consumption and can be driven directly from an Arduino digital pin.

This project uses the same multicolor LED as Project 14. In this case, however, we are not going to blend different colors but just turn one of the LEDs on at a time to display red, green, or blue.

Figure 9-2 shows the schematic diagram for the project and Figure 9-3 the breadboard layout.

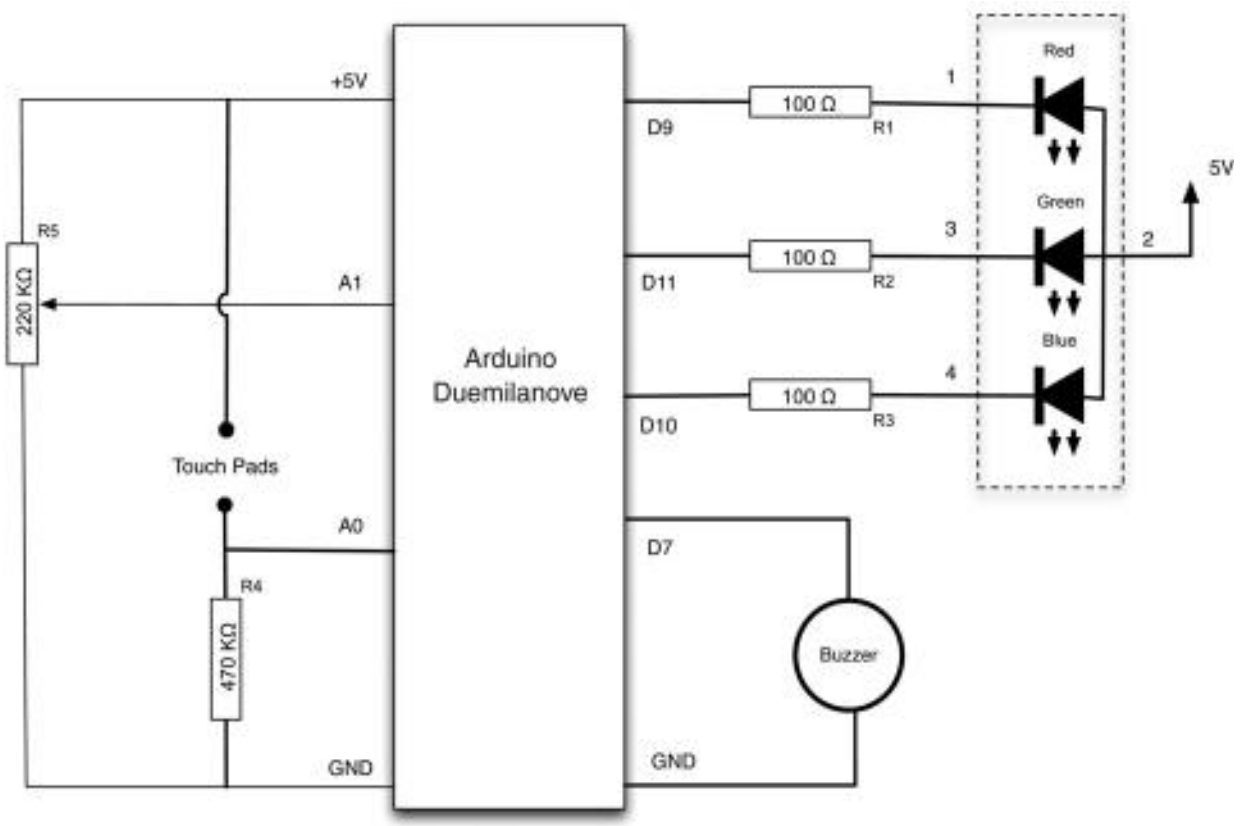


Figure 9-2 Schematic diagram for Project 26.

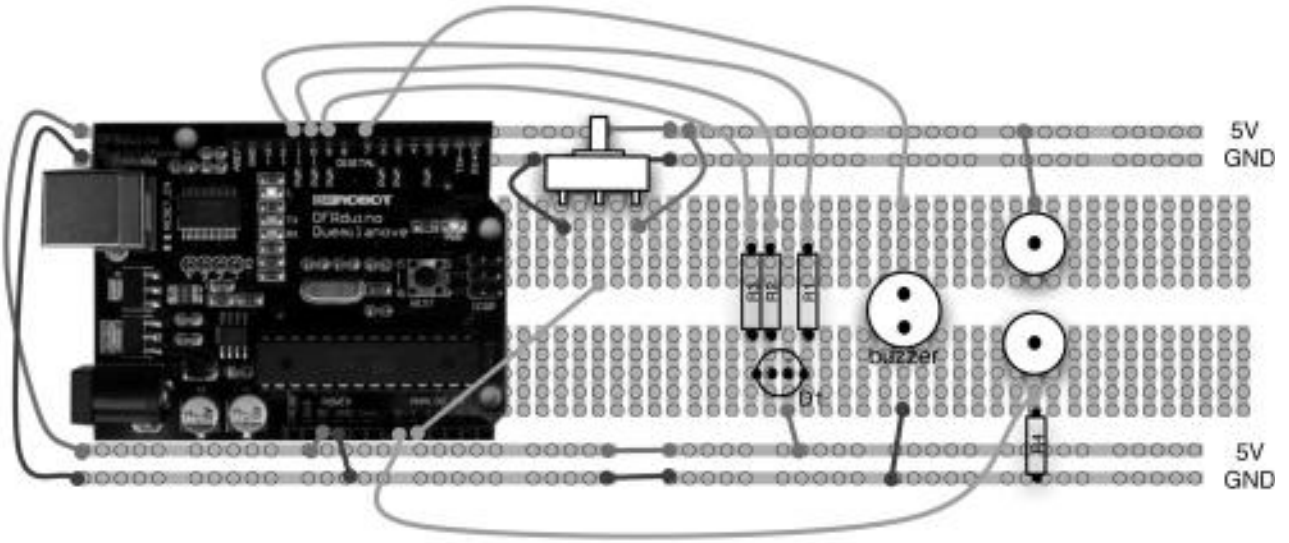


Figure 9-3 Breadboard layout for Project 26.

The variable resistor is used to adjust the set point of resistance, and the touch pads are just two metal thumbtacks pushed into the breadboard.

Software

The script for this project (Listing Project 26) just has to compare the voltage at A0 and A1. If they are about the same, the LED will be set to green. If the voltage from the finger sensor (A0) is significantly higher than A1, the variable resistor will indicate a fall in skin resistance, the LED will change to red, and the buzzer will sound. On the other hand, if A0 is significantly lower than A1, the LED will turn blue, indicating a rise in skin resistance.

The buzzer requires a frequency of about 5KHz or 5000 cycles per second to drive it. We accomplish this with a simple for loop with commands to turn the appropriate pin on and off with delays in between.

Putting It All Together

Load the completed sketch for Project 26 from your Arduino Sketchbook and download it to the board (see Chapter 1).

LISTING PROJECT 26

```
int redPin = 9;
int greenPin = 10;
int bluePin = 11;
int buzzerPin = 7;

int potPin = 1;
int sensorPin = 0;

long red = 0xFF0000;
long green = 0x00FF00;
long blue = 0x000080;

int band = 10;
      // adjust for sensitivity

void setup()
{
  pinMode(potPin, INPUT);
  pinMode(sensorPin, INPUT);
  pinMode(redPin, OUTPUT);
  pinMode(greenPin, OUTPUT);
  pinMode(bluePin, OUTPUT);
  pinMode(buzzerPin, OUTPUT);
}

void loop()
{
```

(continued)

LISTING PROJECT 26 (continued)

```
  int gsr = analogRead(sensorPin);
  int pot = analogRead(potPin);
  if (gsr > pot + band)
  {
    setColor(red);
    beep();
  }
  else if (gsr < pot - band)
  {
    setColor(blue);
  }
  else
  {
    setColor(green);
  }
}


void setColor(long rgb)
{
  int red = rgb >> 16;
  int green = (rgb >> 8) & 0xFF;
  int blue = rgb & 0xFF;
  analogWrite(redPin, 255 - red);
  analogWrite(greenPin, 255 - green);
  analogWrite(bluePin, 255 - blue);
}

void beep()
{
  // 5 Khz for 1/5th second
 for (int i = 0; i < 1000; i++)
 {
    digitalWrite(buzzerPin, HIGH);
    delayMicroseconds(100);
    digitalWrite(buzzerPin, LOW);
    delayMicroseconds(100);
 }
}
```

To test the lie detector, you really need a test subject, as you will need one hand free to adjust the knob.

First, get your subject to place two adjoining fingers on the two metal thumbtacks. Then turn the knob on the variable resistor until the LED turns green.

You may now interrogate your victim. If the LED changes to either red or blue, you should adjust the knob until it changes to green again and then continue the interrogation.

Project 27
Magnetic Door Lock

This project (Figure 9-4) is based on Project 10, but extends it so that when the correct code is entered, it lights a green LED in addition to operating a small solenoid. The sketch is also improved so that the secret code can be changed without having to modify and install a new script. The secret code is stored in EEPROM, so if the power is disconnected, the code will not be lost.

COMPONENTS AND EQUIPMENT

	Description	Appendix
	Arduino Diecimila or Duemilanove board or clone	1
D1	Red 5-mm LED	23
D2	Green 5-mm LED	25
R1-3	270 Ω 0.5W metal film resistor	6
K1	4 x 3 keypad	54
	0.1-inch header strip	55
T1	BC548	40
	5V solenoid (< 100 mA)	66
D3	1N4004	38

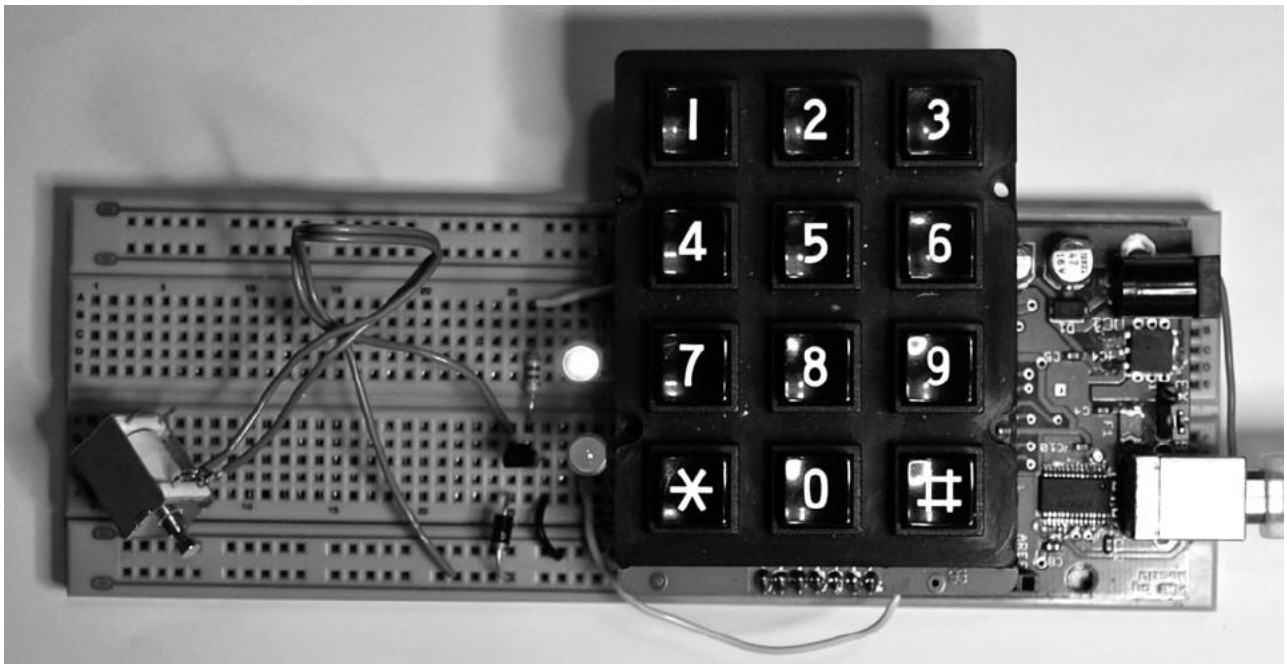


Figure 9-4 Project 27. Magnetic door lock.

When powered, the solenoid will strongly attract the metal slug in its center, pulling it into place. When the power is removed, it is free to move.

Hardware

The schematic diagram (see Figure 9-5) and breadboard layout (see Figure 9-6) are much the same as Project 10, but with additional

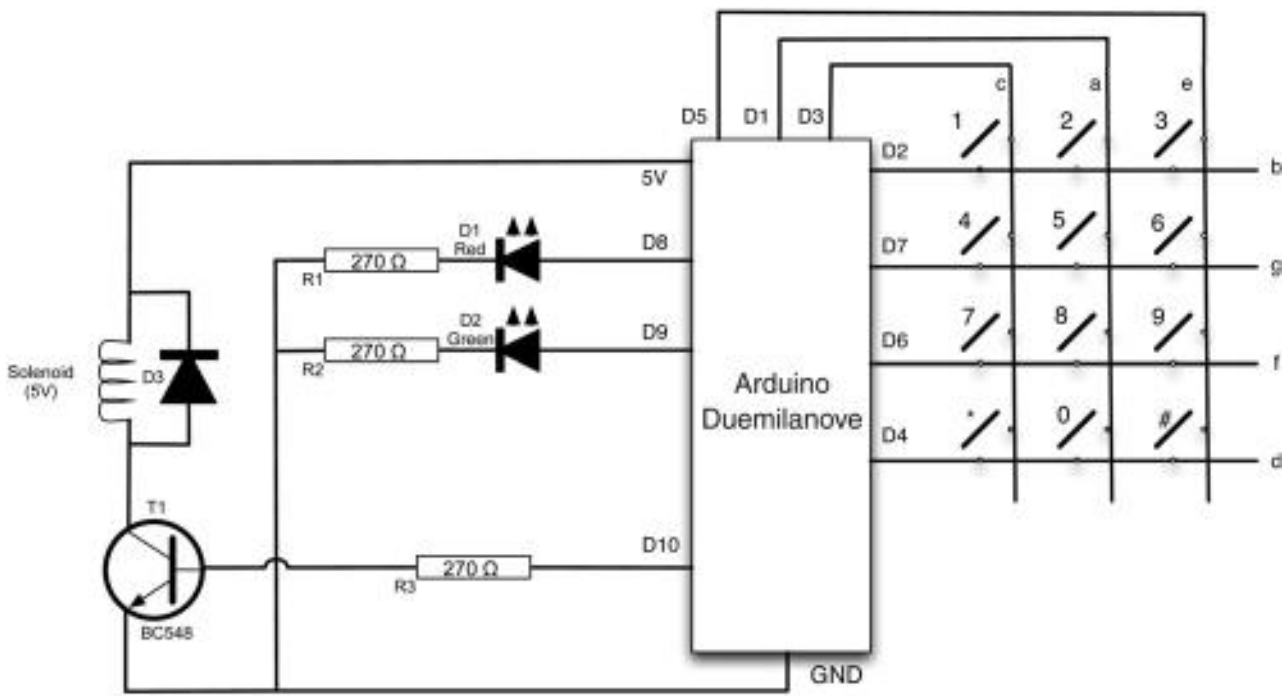


Figure 9-5 Schematic diagram for Project 27.

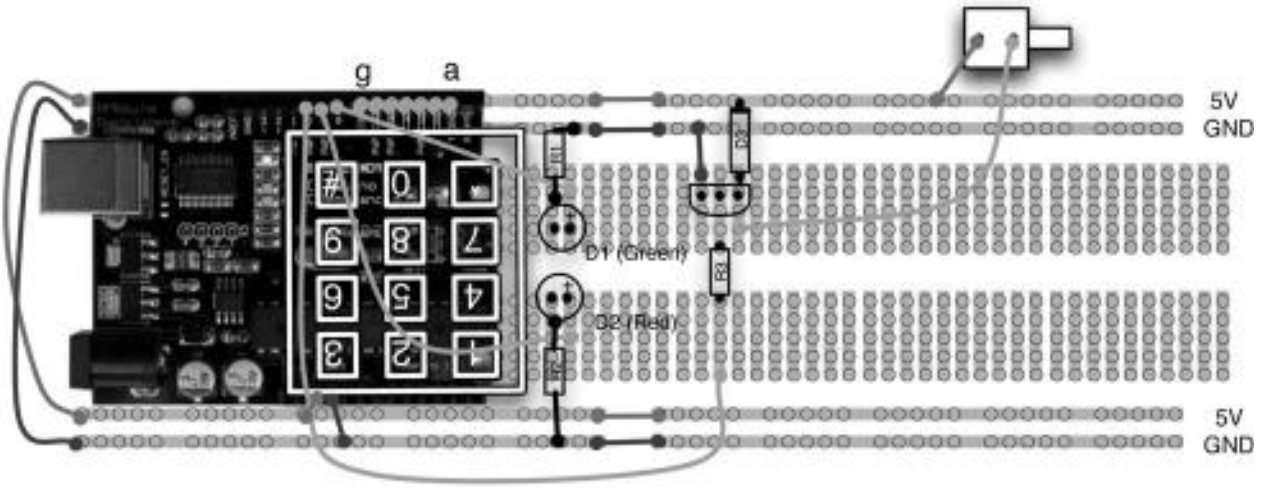


Figure 9-6 Breadboard layout for Project 27.

components. Like relays, the solenoid is an inductive load and therefore liable to generate a back EMF, which diode D3 protects against.

The solenoid is controlled by T1, so be careful to select a solenoid that will not draw more than 100 mA, which is the maximum collector current of the transistor.

We are using a very low power solenoid, and this would not keep intruders out of the Evil Genius' lair. If you are using a more substantial solenoid, a BD139 transistor would be better.

If the solenoid can be mounted on the breadboard, this is all well and good. If not, you will need to attach leads to it that connect it to the breadboard.

Software

The software for this project is, as you would expect, similar to that of Project 10 (see Project Listing 27).

LISTING PROJECT 27

```
#include <Keypad.h>
#include <EEPROM.h>

char* secretCode = "1234";
int position = 0;
boolean locked = true;

const byte rows = 4;
const byte cols = 3;
char keys[rows][cols] = {
  {'1','2','3'},
  {'4','5','6'},
  {'7','8','9'},
  {'*','0','#'}
};
byte rowPins[rows] = {2, 7, 6, 4};
byte colPins[cols] = {3, 1, 5};
```

LISTING PROJECT 27 (continued)

```
Keypad keypad = Keypad(makeKeymap(keys), rowPins, colPins, rows, cols);

int redPin = 9;
int greenPin = 8;
int solenoidPin = 10;

void setup()
{
  pinMode(redPin, OUTPUT);
  pinMode(greenPin, OUTPUT);
  loadCode();
  flash();
  updateOutputs();
}

void loop()
{
  char key = keypad.getKey();
  if (key == '*'  && ! locked)
  {
    // unlocked and * pressed so change code
    position = 0;
    getNewCode();
    updateOutputs();
  }
  if (key == '#')
  {
    locked = true;
    position = 0;
    updateOutputs();
  }
  if (key == secretCode[position])
  {
    position ++;
  }
  if (position == 4)
  {
    locked = false;
    updateOutputs();
  }
  delay(100);
}

void updateOutputs()
{
  if (locked)
  {
    digitalWrite(redPin, HIGH);
    digitalWrite(greenPin, LOW);
    digitalWrite(solenoidPin, HIGH);
  }
  else
```

(continued)

LISTING PROJECT 27 (continued)

```
  {
    digitalWrite(redPin, LOW);
    digitalWrite(greenPin, HIGH);
    digitalWrite(solenoidPin, LOW);
  }
}

void getNewCode()
{
  flash();
  for (int i = 0; i < 4; i++ )
  {
    char key;
    key = keypad.getKey();
    while (key == 0)
    {
      key = keypad.getKey();
    }
    flash();
    secretCode[i] = key;
  }
  saveCode();
  flash();flash();
}

void loadCode()
{
  if (EEPROM.read(0) == 1)
  {
    secretCode[0] = EEPROM.read(1);
    secretCode[1] = EEPROM.read(2);
    secretCode[2] = EEPROM.read(3);
    secretCode[3] = EEPROM.read(4);
  }
}

void saveCode()
{
  EEPROM.write(1, secretCode[0]);
  EEPROM.write(2, secretCode[1]);
  EEPROM.write(3, secretCode[2]);
  EEPROM.write(4, secretCode[3]);
  EEPROM.write(0, 1);
}

void flash()
{
    digitalWrite(redPin, HIGH);
    digitalWrite(greenPin, HIGH);
    delay(500);
    digitalWrite(redPin, LOW);
    digitalWrite(greenPin, LOW);
}
```

Since we can now change the secret code, we have changed the loop function so that if the * key is pressed while the lock is in its unlocked state, the next four keys pressed will be the new code.

Since each character is exactly one byte in length, the code can be stored directly in the EEPROM memory. We use the first byte of EEPROM to indicate if the code has been set. If it has not been set, the code will default to 1234. Once the code has been set, the first EEPROM byte will be given a value of 1.

Putting It All Together

Load the completed sketch for Project 27 from your Arduino Sketchbook and download it to the board (see Chapter 1).

We can make sure everything is working by powering up our project and entering the code 1234, at which point, the green LED should light and the solenoid release. We can then change the code to something a little less guessable by pressing the * key and then entering four digits for the new code. The lock will stay unlocked until we press the # key.

If you forget your secret code, unfortunately, turning the power to the project on and off will not reset it to 1234. Instead, you will have to comment out the line:

```
loadCode();
```

in the setup function, so that it appears as shown here:

```
// loadCode();
```

Now reinstall the sketch and the secret code will be back to 1234. Remember to change your sketch back after setting the code to something that you will remember.

Project 28
Infrared Remote

This project (see Figure 9-7) allows the Evil Genius to control any household devices with an infrared remote control directly from their computer. With it, the Evil Genius can record an infrared message from an existing remote control and then play it back from their computer.

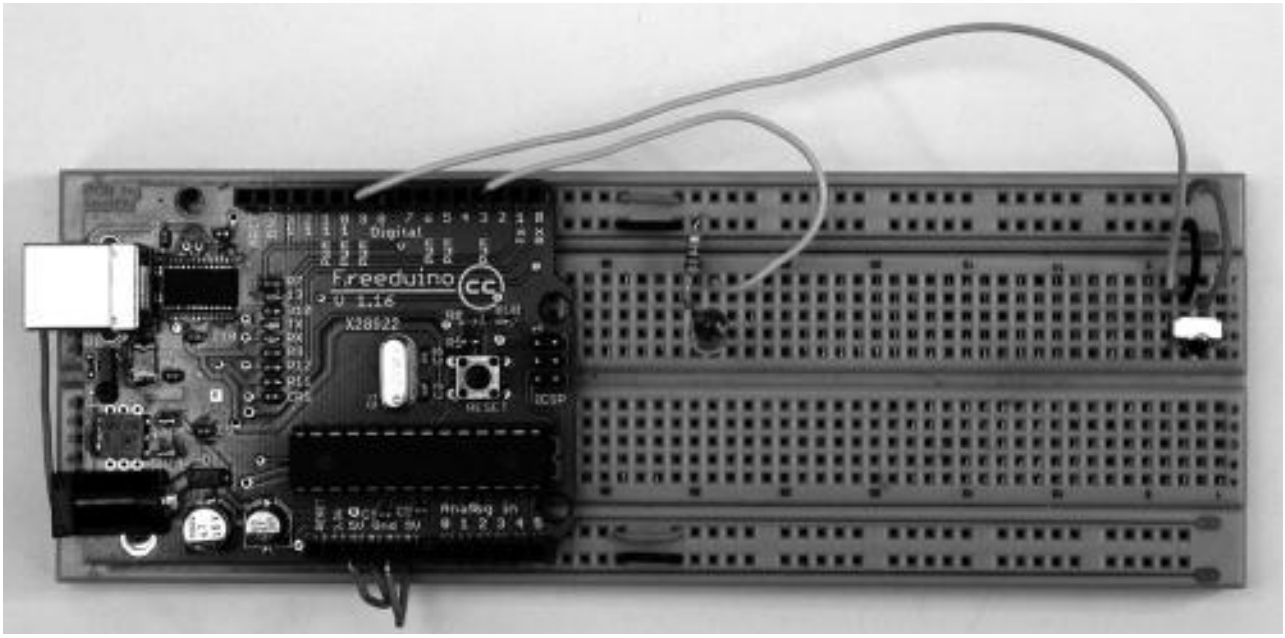


Figure 9-7 Project 28. Infrared remote.

We use the EEPROM memory to store the remote control codes so that they are not lost when the Arduino board is disconnected.

COMPONENTS AND EQUIPMENT

	Description	Appendix
	Arduino Diecimila or Duemilanove board or clone	1
R1	100 Ω 0.5W metal film resistor	5
D1	IR LED sender	26
IC1	IR remote control receiver	37

Hardware

The IR remote receiver is a great little module that combines an infrared photodiode, with all the amplification filtering and smoothing needed to produce a digital output from the IR message. This output is fed to digital pin 9. The schematic diagram (see Figure 9-8) shows how simple this package is to use, with just three pins, GND, +V, and the output signal.

The IR transmitter is an IR LED. These work just like a regular red LED, but in the invisible IR end of the spectrum. On some devices, you can see a slight red glow when they are on.

Figure 9-9 shows the breadboard layout for this project.

Software

Ken Shirriff has created a library that you can use to do just about anything you would want to with an IR remote. We are going to use this library rather than reinvent the wheel.

We first looked at installing a library in Chapter 5. To make use of this library, we must first download it from http://arcfn.com/2009/08/multi-protocol-infrared-remote-library.html.

Download the file IRRemote.zip and unzip it. If you are using Windows, you right-click and choose Extract All and then save the whole folder into C:\Program Files\Arduino\Arduino-0017\hardware\libraries.

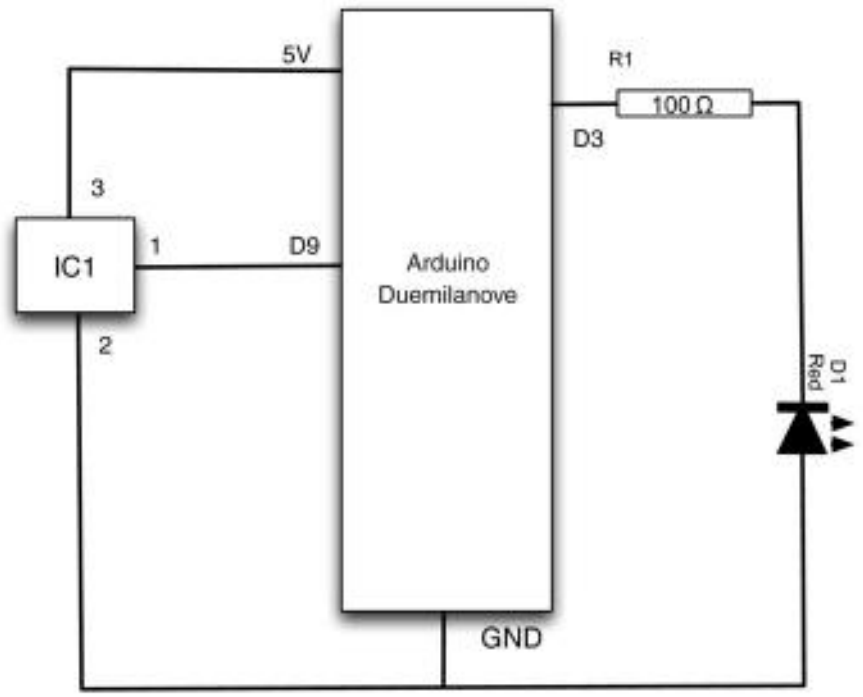


Figure 9-8 Schematic diagram for Project 28.

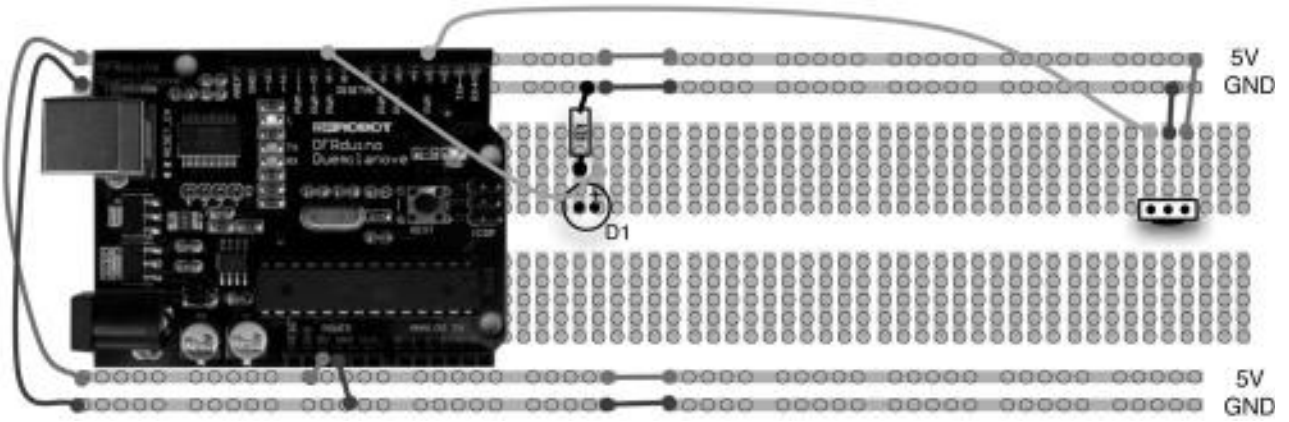


Figure 9-9 Breadboard layout for Project 28.

On LINUX, find the Arduino installation directory and copy the folder into hardware/libraries.

On a Mac, you do not put the new library into the Arduino installation. Instead, you create a folder called "libraries" in Documents/Arduino and put the whole library folder in there.

Once we have installed this library into our Arduino directory, we will be able to use it with any sketches that we write.

Ken's library is essentially the last word in decoding and sending IR commands. It will attempt to match the different protocol standards from different manufacturers. Our sketch actually only uses a small part of the library concerned with capturing and sending the raw pulses of data. (See Listing Project 28.)

Infrared remote controls send a series of pulses at a frequency of between 36 and 40kHz. Figure 9-10 shows the trace from an oscilloscope.

A bit value of 1 is represented by a pulse of square waves at 36 to 40kHz and a 0 by a pause in which no square waves are sent.

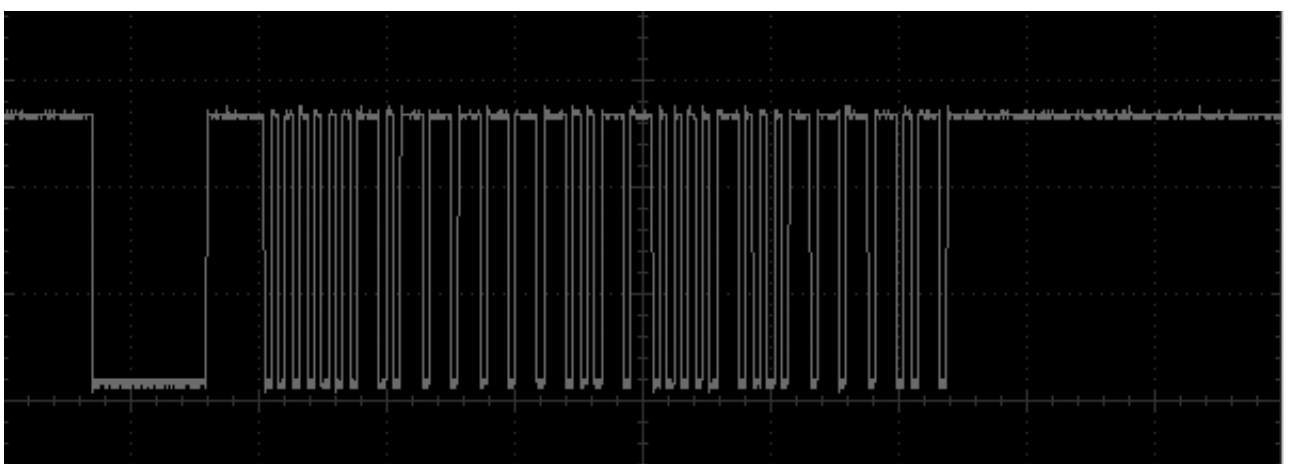


Figure 9-10 Infrared code from an oscilloscope.

LISTING PROJECT 28

```
#include <EEPROM.h>
#include <IRremote.h>

int irRxPin = 9;

int f = 38; // 40, 36, 38

IRrecv irrecv(irRxPin);
IRsend irsend;

decode_results results;
int codeLength = 0;
int currentCode = 0;

void setup()
{
  Serial.begin(9600);
  Serial.println("0-9 to set code memory, s - to send");
  irrecv.enableIRIn();
  setCodeMemory(0);
}

void loop()
{
  if (Serial.available())
  {
    char ch = Serial.read();
    if (ch >= '0' && ch <= '9')
    {
      setCodeMemory(ch - '0');
    }
    else if (ch == 's')
    {
      sendIR();
    }
  }
  if (irrecv.decode(&results))
  {
    storeCode();
    irrecv.resume();
  }
}

void setCodeMemory(int x)
{
  currentCode = x;
  Serial.print("Set current code memory to: ");
```

LISTING PROJECT 28 (continued)

```
  Serial.println(currentCode);
  irrecv.resume();
}

void storeCode()
{
   // write the code to EEPROM, first byte is length
   int startIndex = currentCode * (RAWBUF + 1);
   int len = results.rawlen - 1;
   EEPROM.write(startIndex, (unsigned byte)len);
   for (int i = 0; i < len; i++)
   {
     if (results.rawbuf[i] > 255)
     {
       EEPROM.write(startIndex + i + 1, 255);
     }
     else
     {
      EEPROM.write(startIndex + i + 1, results.rawbuf[i]);
     }
   }
   Serial.print("Saved code, length: ");
   Serial.println(len);
}

void sendIR()
{
  // construct a buffer from the saved data in EEPROM and send it
  int startIndex = currentCode * (RAWBUF + 1);
  int len = EEPROM.read(startIndex);
  unsigned int code[RAWBUF];
  for (int i = 0; i < len; i++)
  {
     int pulseLen = EEPROM.read(startIndex + i + 2);
     if (i % 2)
     {
        code[i] = pulseLen * USECPERTICK + MARK_EXCESS;
     }
     else
     {
        code[i] = pulseLen * USECPERTICK - MARK_EXCESS;
     }
  }
  irsend.sendRaw(code, len, f);
  Serial.print("Sent code length: ");
  Serial.println(len);
}
```

The IRRemote library requires the IR LED to be driven from digital pin 3. Hence, we only specify the receiving pin at the top of the sketch (irRxPin).

In the setup function, we start serial communications and write instructions for using the project back to the Serial Console. It is from the Serial Console that we are going to control the project. We also set the current code memory to memory 0.

The loop function follows the familiar pattern of checking for any input through the USB port. If it is a digit between 0 and 9 it makes the corresponding memory the current memory. If an "s" character is received from the Serial Monitor, it sends the message in the current message memory.

The function then checks to see if any IR signal has been received; if it has, the function writes it to EEPROM using the storeCode function. It stores the length of the code in the first byte and then the number of 50-microsecond ticks for each subsequent pulse in bytes that follow. RAWBUF is a constant defined in the library as the maximum message length.

Note that as part of the process of sending the code in the sendIR function an array of pulse timing integers is created from the stored bytes, the timings of the pulses are then in microseconds rather than ticks, and are adjusted by an offset MARK_EXCESS to compensate for the hardware that reads the IR signals. This tends to make marks slightly longer than they should be and spaces slightly shorter.

We also use an interesting technique in storeCode and sendIR when accessing the EEPROM that lets us use it rather like an array for the message memories. The start point for recording or reading the data from EEPROM is calculated by multiplying the currentCode by the length of each code (plus the byte that says how long it is).

Putting It All Together

Load the completed sketch for Project 28 from your Arduino Sketchbook and download it to the board (see Chapter 1).

To test the project, find yourself a remote and the bit of equipment that it controls. Then power up the project.

Open the Serial Monitor, and you should be greeted by the following message:

```
0-9 to set code memory, s - to send
Set current code memory to: 0
```

By default, any message we capture will be recorded into memory 0. So aim the remote at the sensor, press a button (turning power on or ejecting the tray on a DVD player are impressive actions). You should then see a message like:

```
Saved code, length: 67
```

Now point the IR LED at the appliance and type **s** into the Serial Monitor. You should receive a message like this:

```
Sent code length: 67
```

More importantly, the appliance should respond to the message from the Arduino board.

Note that the IR LED may not be very bright in its signal, so if it does not work, try moving it around and putting it closer to the appliance's IR sensor.

You can now try changing the memory slot by entering a different digit into the Serial Monitor and recording a variety of different IR commands. Note that there is no reason why they need to be for the same appliance.

Project 29
Lilypad Clock

The Arduino Lilypad works in much the same way as the Duemilanove board, but instead of a boring rectangular circuit board, the Lilypad is circular and designed to be stitched into clothing using conductive thread. Even an Evil Genius appreciates beauty when they see it. So this project is built into a photo frame to show off the natural beauty of the electronics (see Figure 9-11). A magnetic reed switch is used to adjust the time.

This is a project where you have to use a soldering iron.

COMPONENTS AND EQUIPMENT

	Description	Appendix
	Arduino Lilypad and USB programmer	2
R1-16	100 Ω 0.5W metal film resistor	5
D1-4	2-mm red LEDs	27
D5-10	2-mm blue LEDs	29
D11-16	2-mm green LEDs	28
R17	100 KΩ 0.5W metal film resistor	13
S1	Miniature reed switch	9
	7 × 5 inch picture frame	70
	5V power supply	71

Hardware

We have an LED and series resistor attached to almost every connection of the Lilypad in this project.

The reed switch is a useful little component that is just a pair of switch contacts in a sealed glass envelope. When a magnet comes near to the switch, the contacts are pulled together and the switch is closed.

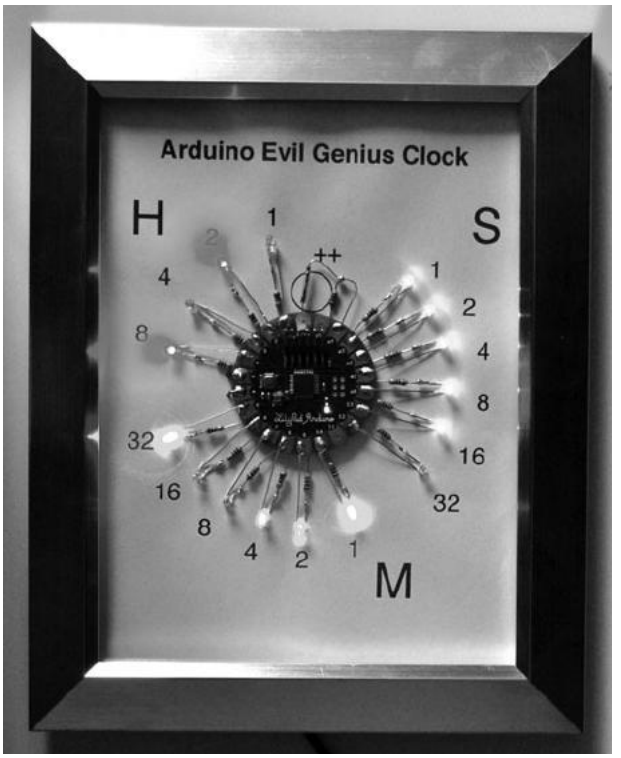


Figure 9-11 Project 29. Lilypad binary clock.

We use a reed switch rather than an ordinary switch so that the whole project can be mounted behind glass in a photo frame. We will be able to adjust the time by holding a magnet close to the switch.

Figure 9-12 shows the schematic diagram for the project.

Each LED has a resistor soldered to the shorter negative lead. The positive lead is then soldered to the Arduino Lilypad terminal and the lead from the resistor passes under the board, where it is connected to all the other resistor leads.

Figure 9-13 shows a close-up of the LED and resistor, and the wiring of the leads under the board is shown in Figure 9-14. Note the rough disc of paper protecting the back of the board from the soldered resistor leads.

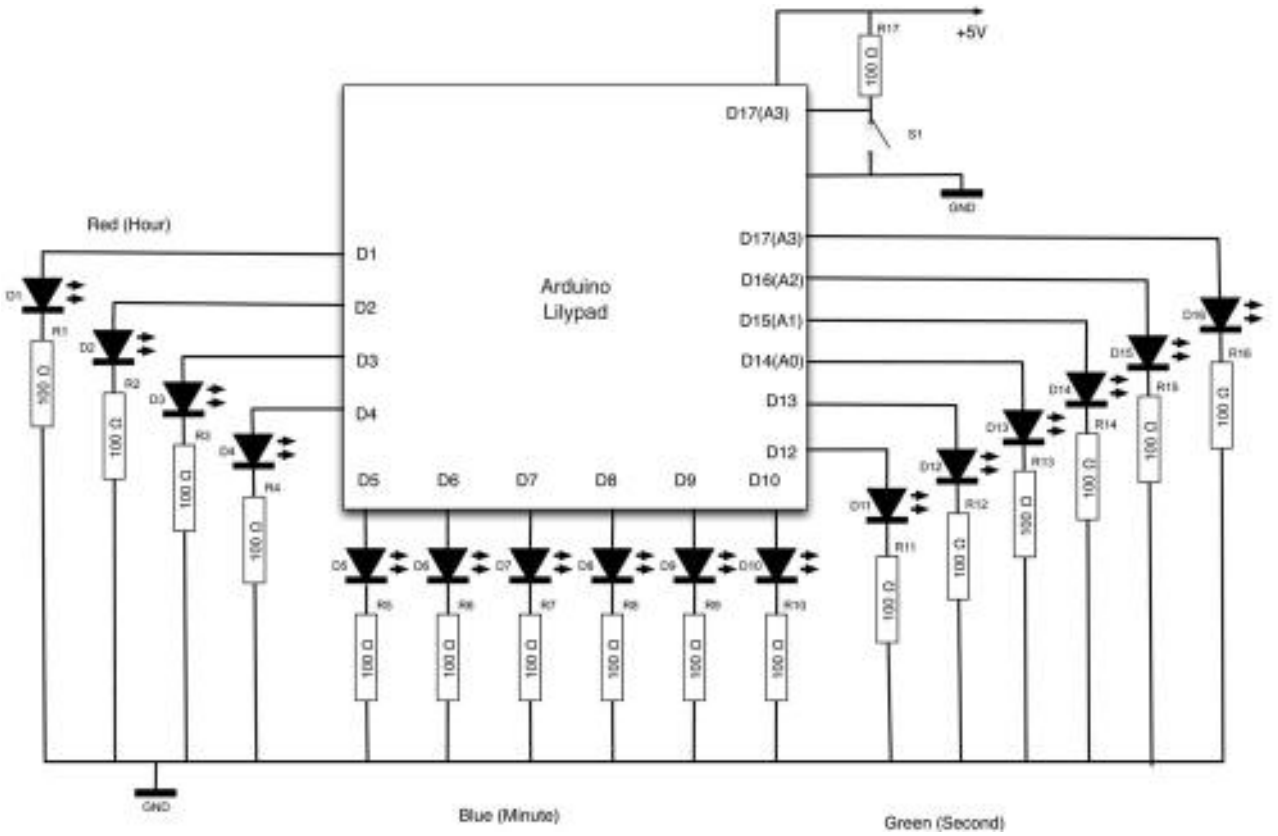


Figure 9-12 Schematic diagram for Project 29.

Figure 9-13 Close-up of LED attached to a resistor.

Figure 9-14 Bottom side of Lilypad board.

A 5V power supply is used, as a significant amount of power is used when all the LEDs are lit and so batteries would not last long. The power wires extend from the side of the picture frame, where they are soldered to a connector.

The author used a redundant cell phone power supply. Be sure to test that any supply you are going to use provides 5V at a current of at least 500 mA. You can test the polarity of the power supply using a multimeter.

Software

This is another project in which we make use of a library. This library makes dealing with time easy and can be downloaded from www.arduino.cc/playground/Code/Time.

Download the file Time.zip and unzip it. If you are using Windows, right-click and choose Extract All and then save the whole folder into C:\Program Files\Arduino\Arduino-0017\hardware\libraries.

On LINUX, find the Arduino installation directory and copy the folder into hardware/libraries.

On a Mac, you do not put the new library into the Arduino installation. Instead, you create a folder called "libraries" in Documents/Arduino and put the whole library folder in there.

Once we have installed this library into our Arduino directory, we will be able to use it with any sketches that we write. (See Listing Project 29.)

Arrays are used to refer to the different sets of LEDs. These are used to simplify installation and also in the setOutput function. This function sets the binary values of the array of LEDs that is to display a binary value. The function also receives arguments of the length of that array and the value to be written to it. This is used in the loop function to successively set the LEDs for hours, minutes, and seconds. When passing an array into a

LISTING PROJECT 29

```
#include <Time.h>

int hourLEDs[] = {1, 2, 3, 4};
 // least significant bit first
int minuteLEDs[] = {10, 9, 8, 7, 6,
 5};
int secondLEDs[] = {17, 16, 15, 14,
 13, 12};

int loopLEDs[] = {17, 16, 15, 14, 13,
 12, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1};

int switchPin = 18;

void setup()
{
  for (int i = 0; i < 4; i++)
  {
     pinMode(hourLEDs[i], OUTPUT);
  }
  for (int i = 0; i < 6; i++)
  {
     pinMode(minuteLEDs[i], OUTPUT);
  }
  for (int i = 0; i < 6; i++)
  {
     pinMode(secondLEDs[i], OUTPUT);
  }
  setTime(0);
}

void loop()
{
  if (digitalRead(switchPin))
  {
     adjustTime(1);
  }
  else if (minute() == 0 && second()
     == 0)
  {
    spin(hour());
  }
  updateDisplay();
  delay(1);
}
```

(continued)

LISTING PROJECT 29 (continued)

```
void updateDisplay()
{
  time_t t = now();
  setOutput(hourLEDs, 4,
     hourFormat12(t));
  setOutput(minuteLEDs, 6, minute(t));
  setOutput(secondLEDs, 6, second(t));
}


void setOutput(int *ledArray, int
 numLEDs, int value)
{
    for (int i = 0; i < numLEDs; i++)
    {
     digitalWrite(ledArray[i],
        bitRead(value, i));
    }
}


void spin(int count)
{
  for (int i = 0; i < count; i++)
  {
      for (int j = 0; j < 16; j++)
      {
         digitalWrite(loopLEDs[j],
            HIGH);
         delay(50);
         digitalWrite(loopLEDs[j],
            LOW);
      }
  }
}
```

function like this, you must prefix the argument in the function definition with a *.

An additional feature of the clock is that every hour, on the hour, it spins the LEDs, lighting each one in turn. So at 6 o'clock, for example, it will spin six times before resuming the normal pattern.

If the reed relay is activated, the adjustTime function is called with an argument of 1 second. Since this is in the loop function with a one-millisecond delay, the seconds are going to pass quickly.

Putting It All Together

Load the completed sketch for Project 29 from your Arduino Sketchbook and download it to the board. On a Lilypad, this is slightly different to what we are used to. You will have to select a different board type and serial port from the Arduino software before downloading.

Assemble the project, but test it connected to the USB programmer before you build it into the picture frame.

Try to choose a picture frame that has a thick card insert that will allow a sufficient gap into which the components can fit between the backing board and the glass.

You may wish to design a paper insert to provide labels for your LEDs to make it easier to tell the time. A suitable design can be found at www.arduinoevilgenius.com.

To read the time from the clock, you look at each section (Hours, Minutes, and Seconds) in turn and add the values next to the LEDs that are lit. So, if the hour LEDs next to 8 and 2 are lit, then the hour is 10. Then do the same for the minutes and seconds.

Project 30
Evil Genius Countdown Timer

No book on projects for an Evil Genius should be without the Bond-style countdown timer, complete with a rat's nest of colored wires (see Figure 9-15). This timer also doubles as an egg timer, because there is nothing that annoys the Evil Genius more than an overcooked soft-boiled egg!

COMPONENTS AND EQUIPMENT

	Description	Appendix
	Arduino Diecimila or Duemilanove board or clone	1
D1-4	2-digit, 7-segment LED display (common anode)	33
R1-3	100 KΩ 0.5W metal film resistor	13
R4-7	1 KΩ 0.5W metal film resistor	7
R8-15	100 Ω 0.5W metal film resistor	5
T1-4	BC307	39
	Rotary encoder	57
	Piezobuzzer (integrated electronics)	68

For optimal loudness, the piezobuzzer used is the kind that has integrated electronics so all that is necessary to make it sound is to provide it with 5V. Be careful to connect it in the correct way.

Hardware

The project is similar to Project 15, but with the extra seven-segment display and associated transistors. We also have a rotary encoder that we will use to set the time to count down from. We have used both components before; for more information on rotary encoders, see Chapter 6.

The schematic diagram for the project is shown in Figure 9-16 and the breadboard layout in Figure 9-17.

Software

The sketch for this project (Project Listing 30) is mostly concerned with keeping the display up-to-date and creating the illusion that all four displays are lit at the same time, when in fact, only one will ever be lit. The way this is accomplished is described in Project 15.

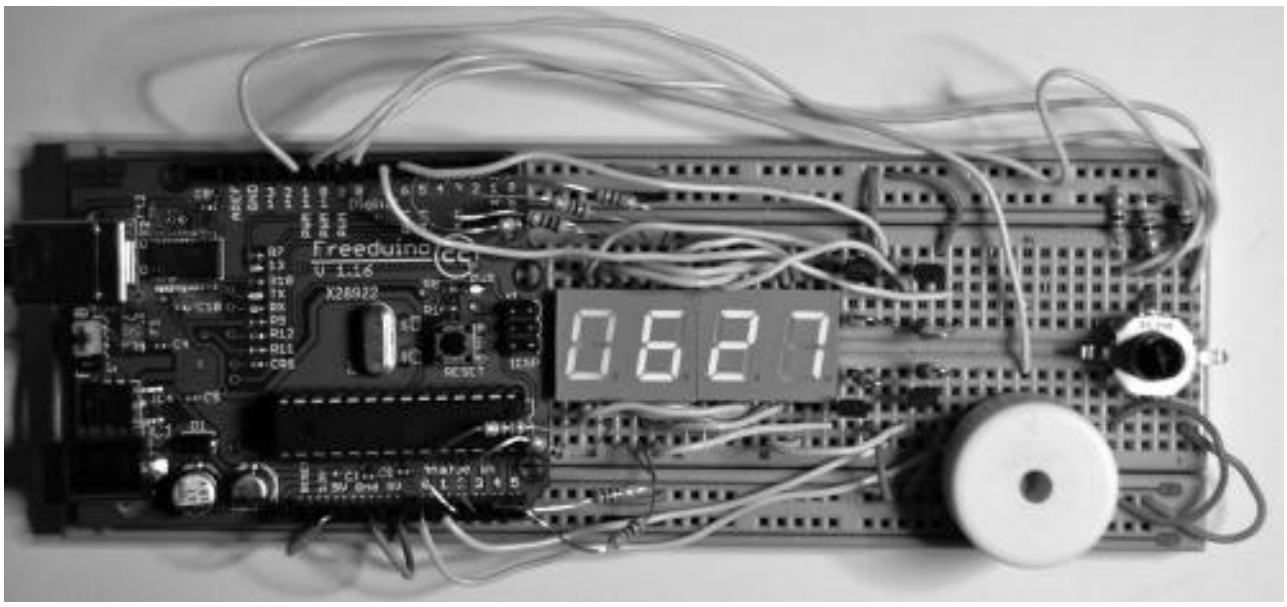


Figure 9-15 Project 30. Evil Genius countdown timer.

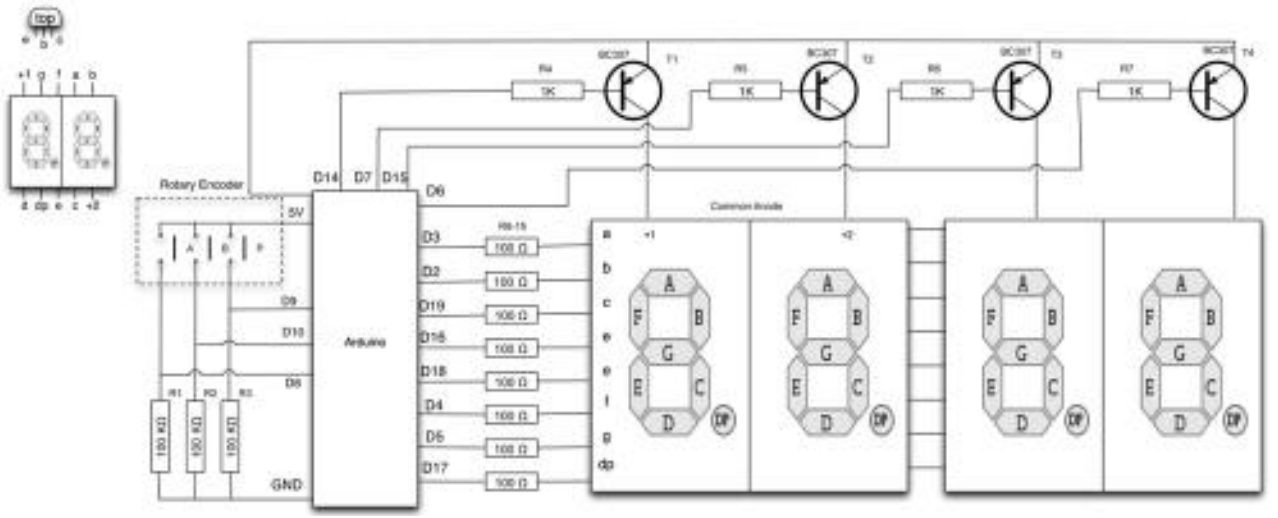


Figure 9-16 Schematic diagram for Project 30.

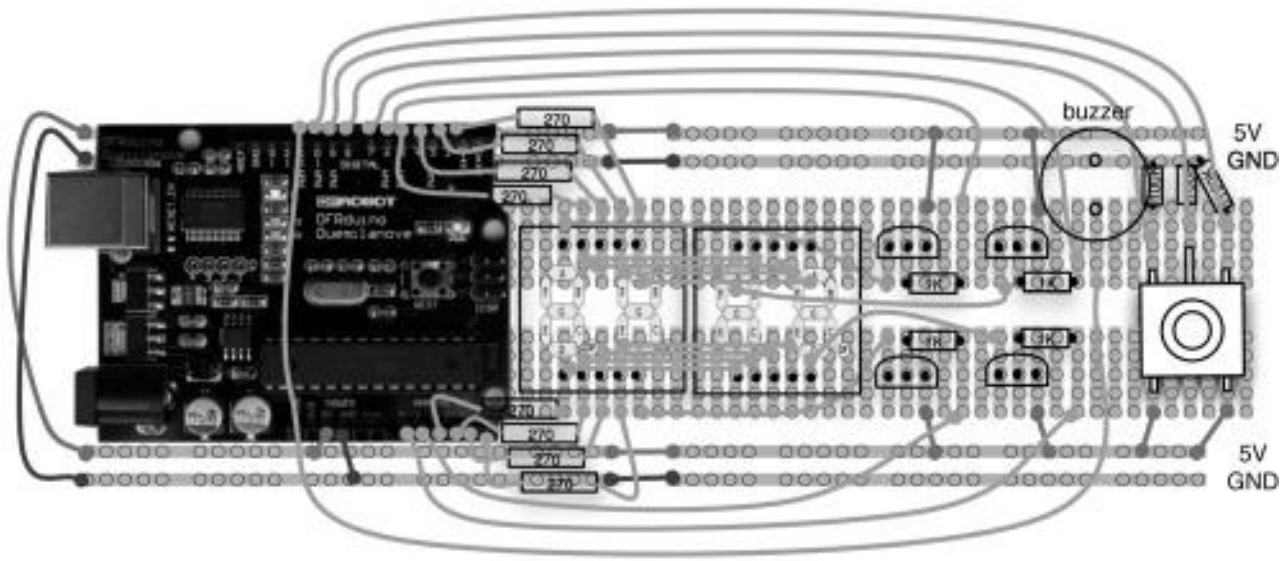


Figure 9-17 A powered-up Arduino board with LED lit.

LISTING PROJECT 30

```
#include <EEPROM.h>

int segmentPins[] = {3, 2, 19, 16, 18, 4, 5, 17};
int displayPins[] = {14, 7, 15, 6};
int times[] = {5, 10, 15, 20, 30, 45, 100, 130, 200, 230, 300, 400, 500, 600,
    700, 800, 900, 1000, 1500, 2000, 3000};
int numTimes = 19;
byte selectedTimeIndex;
int timerMinute;
int timerSecond;

int buzzerPin = 11;
int aPin = 8;
int bPin = 10;
int buttonPin = 9;

boolean stopped = true;


byte digits[10][8] = {
//  a  b  c  d  e  f  g  .
  { 1, 1, 1, 1, 1, 1, 0, 0},  // 0
  { 0, 1, 1, 0, 0, 0, 0, 0},  // 1
  { 1, 1, 0, 1, 1, 0, 1, 0},  // 2
  { 1, 1, 1, 1, 0, 0, 1, 0},  // 3
  { 0, 1, 1, 0, 0, 1, 1, 0},  // 4
  { 1, 0, 1, 1, 0, 1, 1, 0},  // 5
  { 1, 0, 1, 1, 1, 1, 1, 0},  // 6
  { 1, 1, 1, 0, 0, 0, 0, 0},  // 7
  { 1, 1, 1, 1, 1, 1, 1, 0},  // 8
  { 1, 1, 1, 1, 0, 1, 1, 0}  // 9
};

void setup()
{
  for (int i=0; i < 8; i++)
  {
    pinMode(segmentPins[i], OUTPUT);
  }
  for (int i=0; i < 4; i++)
  {
    pinMode(displayPins[i], OUTPUT);
  }
  pinMode(buzzerPin, OUTPUT);
  pinMode(buttonPin, INPUT);
  pinMode(aPin, INPUT);
  pinMode(bPin, INPUT);
  selectedTimeIndex = EEPROM.read(0);
  timerMinute = times[selectedTimeIndex] / 100;
  timerSecond = times[selectedTimeIndex] % 100;
```

(continued)

LISTING PROJECT 30 (continued)

```
}

void loop()
{
  if (digitalRead(buttonPin))
  {
    stopped = ! stopped;
    digitalWrite(buzzerPin, LOW);
    while (digitalRead(buttonPin)) {};
    EEPROM.write(0, selectedTimeIndex);
  }
  updateDisplay();
}

void updateDisplay() // mmss
{
  int minsecs = timerMinute * 100 + timerSecond;
  int v = minsecs;
  for (int i = 0; i < 4; i ++)
  {
      int digit = v % 10;
      setDigit(i);
      setSegments(digit);
      v = v / 10;
      process();
  }
  setDigit(5); // all digits off to prevent uneven illumination
}

void process()
{
  for (int i = 0; i < 100; i++) // tweak this number between flicker and blur
  {
    int change = getEncoderTurn();
    if (stopped)
    {
      changeSetTime(change);
    }
    else
    {
      updateCountingTime();
    }
  }
  if (timerMinute == 0 && timerSecond == 0)
  {
    digitalWrite(buzzerPin, HIGH);
  }

}

void changeSetTime(int change)
```

LISTING PROJECT 30 (continued)

```
{
   selectedTimeIndex += change;
   if (selectedTimeIndex < 0)
   {
     selectedTimeIndex = numTimes;
   }
   else if (selectedTimeIndex > numTimes)
   {
     selectedTimeIndex = 0;
   }
   timerMinute = times[selectedTimeIndex] / 100;
   timerSecond = times[selectedTimeIndex] % 100;
}

void updateCountingTime()
{
   static unsigned long lastMillis;
   unsigned long m = millis();
   if (m > (lastMillis + 1000) && (timerSecond > 0 || timerMinute > 0))
   {
    digitalWrite(buzzerPin, HIGH);
    delay(10);
    digitalWrite(buzzerPin, LOW);

    if (timerSecond == 0)
    {
       timerSecond = 59;
       timerMinute –;
    }
    else
    {
       timerSecond –;
    }
    lastMillis = m;
  }
}

void setDigit(int digit)
{
  for (int i = 0; i < 4; i++)
  {
    digitalWrite(displayPins[i], (digit != i));
  }
}

void setSegments(int n)
{
  for (int i = 0; i < 8; i++)
  {
    digitalWrite(segmentPins[i], ! digits[n][i]);
```

(continued)

LISTING PROJECT 30 (continued)

```
  }
}


int getEncoderTurn()
{
  // return -1, 0, or +1
  static int oldA = LOW;
  static int oldB = LOW;
  int result = 0;
  int newA = digitalRead(aPin);
  int newB = digitalRead(bPin);
  if (newA != oldA || newB != oldB)
  {
    // something has changed
    if (oldA == LOW && newA == HIGH)
    {
      result = -(oldB * 2 - 1);
    }
  }
  oldA = newA;
  oldB = newB;
  return result;
}
```

The timer will always be in one of two states. It will either be stopped, in which case, turning the rotary encoder will change the time, or it can be running, in which case it will be counting down. Pressing the button on the rotary encoder will toggle between the two states.

Rather than make the rotary encoder change the time one second per rotation step, we have an array of standard times that fit with the egg-cooking habits of the Evil Genius. This array can be edited and extended, but if you change its length, you must alter the numTimes variable accordingly.

The EEPROM library is used to store the last used time so that each time the project is powered up, it will remember the last time used.

The project makes a little chirp as each second ticks by. You may wish to disable this. You will find the relevant lines of code to comment out or delete this in the updateCountTime function.

Putting It All Together

Load the completed sketch for Project 30 from your Arduino Sketchbook and download it to the board (see Chapter 1).

Summary

This is the final chapter containing projects. The author hopes that in trying the projects in this book, the Evil Genius' appetite for experimentation and design has been stirred and they will have the urge to design some projects of their own.

The next chapter sets out to help you in the process of developing your own projects.

CHAPTER 10

Your Projects

So, you have tried your hand at some of the author's projects and hopefully learned something along the way. Now it's time to start developing your own projects using what you have learned. You will be able to borrow bits of design from the projects in this book, but to help you along, this chapter gets you started with some design and construction techniques.

Circuits

The author likes to start a project with a vague notion of what he wants to achieve and then start designing from the perspective of the electronics. The software usually comes afterwards.

The way to express an electronic circuit is to use a schematic diagram. The author has included schematic diagrams for all the projects in this book, so even if you are not very familiar with electronics, you should now have seen enough schematics to understand roughly how they relate to the breadboard layout diagrams also included.

Schematic Diagrams

In a schematic diagram, connections between components are shown as lines. These connections will use the connective strips beneath the surface of the breadboard and the wires connecting one breadboard strip to another. For the kinds of projects in this book, it does not normally matter how the connection is made. The arrangement of the actual wires does not matter as long as all the points are connected.

Schematic diagrams have a few conventions that are worth pointing out. For instance, it is common to place GND lines near the bottom of the diagram and higher voltages near the top of the diagram. This allows someone reading the schematic to visualize the flow of charge through the system from higher voltages to lower voltages.

Another convention in schematic diagrams is to use the little bar symbol to indicate a connection to GND where there is not enough room to draw all the connections.

Figure 10-1, originally from Project 5, shows three resistors, all with one lead connected to the GND connection of the Arduino board. In the corresponding breadboard layout (Figure 10-2), you can see that the connections to GND go through three wires and three strips of breadboard connector block.

There are many different tools for drawing schematic diagrams. Some of them are integrated-electronics CAD (computer-aided design) products that will go on to lay out the tracks on a printed circuit board for you. By and large, these create fairly ugly-looking diagrams, and the author prefers to use pencil and paper or general-purpose drawing software. All the diagrams for this book

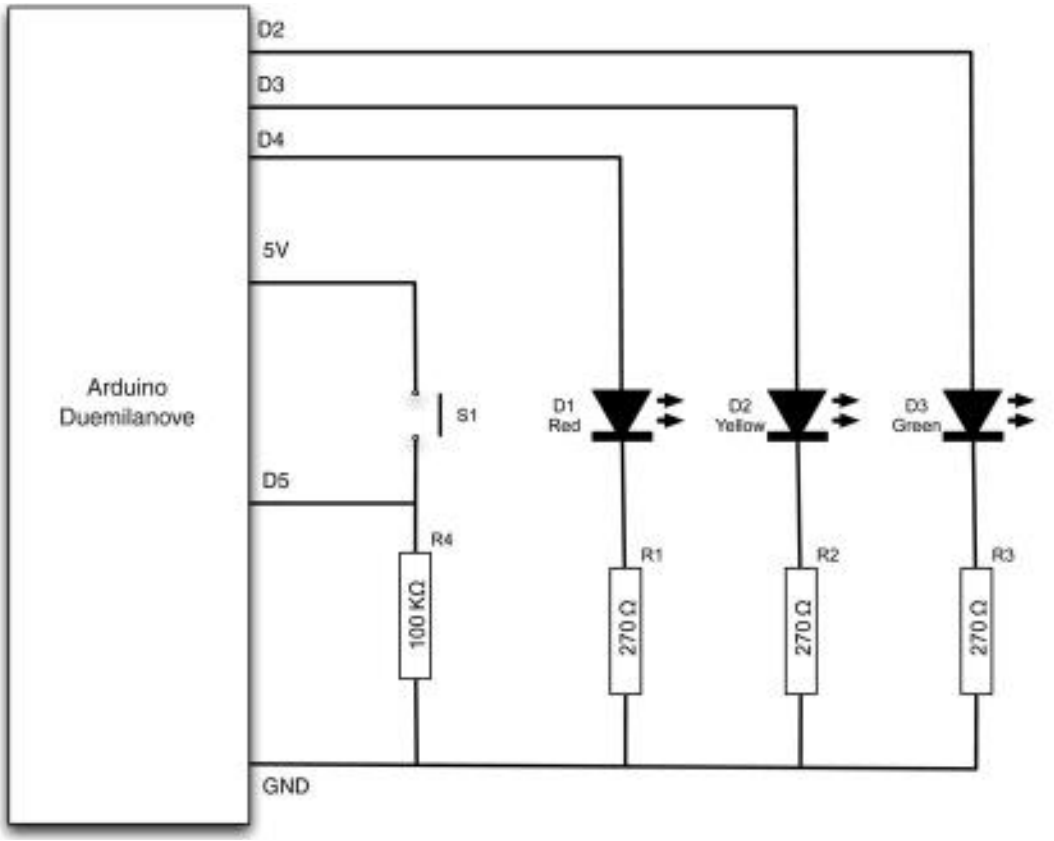


Figure 10-1 A schematic diagram example.

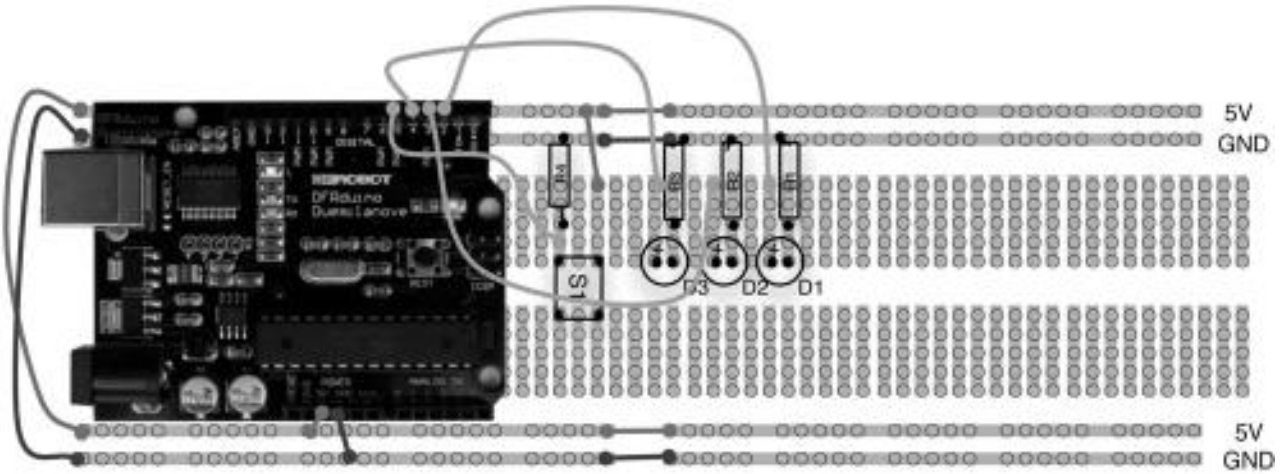


Figure 10-2 Example breadboard layout.

were created using Omni Group's excellent but strangely named OmniGraffle software, which is only available for Apple Macs. OmniGraffle templates for drawing breadboard layouts and schematic diagrams are available for download from www.arduinoevilgenius.com.

Component Symbols

Figure 10-3 shows the circuit symbols for the electronic components that we have used in this book.

There are various different standards for circuit diagrams, but the basic symbols are all recognizable between standards. The set used in this book does not closely follow any particular standard. I have just chosen what I consider to be the most easy-to-read approach to the diagrams.

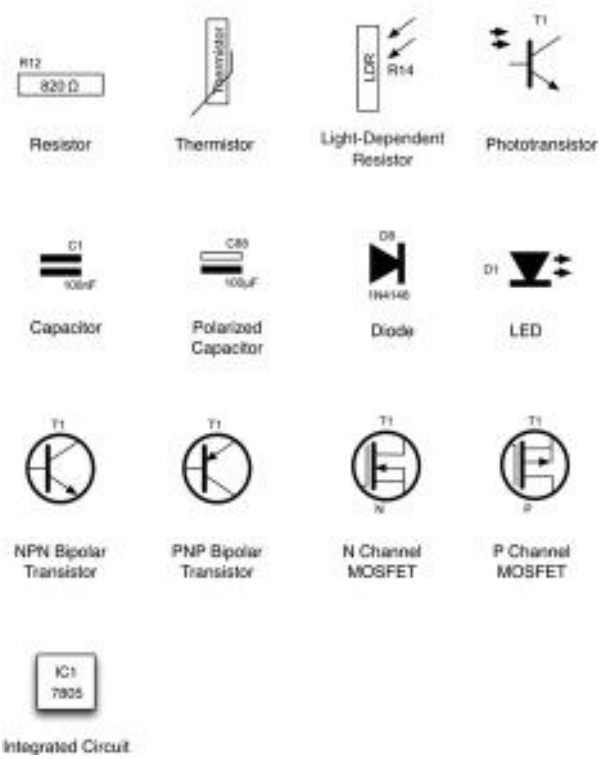


Figure 10-3 Circuit symbols.

Components

In this section we look at the practical aspects of components: what they do and how to identify, choose, and use them.

Datasheets

All component manufacturers produce datasheets for their products. These act as a specification for how the component will behave. They are not of much interest for resistors and capacitors, but are much more useful for semiconductors and transistors, but especially integrated circuits. They will often include application notes that include example schematics for using the components.

These are all available on the Internet. However, if you search for "BC158 datasheet" in your favorite search engine, you will find many of the top hits are for organizations cashing in on the fact that people search for datasheets a lot. These organizations surround the datasheets with pointless advertising and pretend that they add some value to looking up datasheets by subscribing to their service. These websites usually just lead to a frustration of clicking and should be ignored in favor of any manufacturer's websites. So scan through the search results until you see a URL like www.fairchild.com.

Alternatively, many of the component retail suppliers such as Farnell provide free-of-charge and nonsense-free datasheets for practically every component they sell, which is to be much applauded. It also means that you can compare prices and buy the components while you are finding out about them.

Resistors

Resistors are the most common and cheap electronic components around. Their most common uses are

- To prevent excessive current flowing (see any projects that use an LED)
- In a pair or as a variable resistor to divide a voltage

Chapter 2 explained Ohm's Law and used it to decide on a value of series resistor for an LED. Similarly, in Project 19, we reduced the signal from our resistor ladder using two resistors as a potential divider.

Resistors have colored bands around them to indicate their value. However, if you are unsure of a resistor, you can always find its resistance using a multimeter. Once you get the hang of it, it's easy to read the values using the colored bands.

Each band color has a value associated with it, as shown in Table 10-1.

TABLE 10-1 Resistor Color Codes

Black	0
Brown	1
Red	2
Orange	3
Yellow	4
Green	5
Blue	6
Violet	7
Gray	8
White	9

There will generally be three of these bands together starting at one end of the resistor, a gap, and then a single band at the other end of the resistor. The single band indicates the accuracy of the resistor value. Since none of the projects in this book require accurate resistors, there is no need to select your resistors on this basis.

Figure 10-4 shows the arrangement of the colored bands. The resistor value uses just the three bands. The first band is the first digit, the second the second digit, and the third "multiplier" band is how many zeros to put after the first two digits.

So a 270 Ω resistor will have first digit 2 (red), second digit 7 (violet), and a multiplier of 1 (brown). Similarly, a 10K Ω resistor will have bands of brown, black, and orange (1, 0, 000).

Most of our projects use resistors in a very low-power manner. A quick calculation can be used to work out the current flowing through the resistor, and multiplying it by the voltage across it will tell you the power used by the resistor. The resistor burns off this surplus power as heat, and so resistors will get warm if a significant amount of current flows through them.

You only need to worry about this for low-value resistors of less than 100 Ω or so, because higher values will have such a small current flowing through them.

As an example, a 100 Ω resistor connected directly between 5V and GND will have a current through it of I = V/R, or 5/100, or 0.05 Amps. The power it uses will be IV or $0.05 \times 5 = 0.25$W.

A standard power rating for resistors is 0.5W or 0.6W, and unless otherwise stated in projects, 0.5W metal film resistors will be fine.

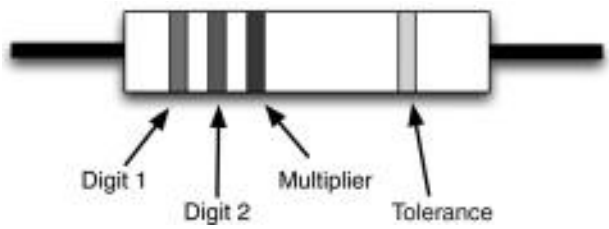


Figure 10-4 A color-coded resistor.

Transistors

Browse through any component catalog and you will find literally thousands of different transistor types. In this book, the list has been simplified to what's shown in Table 10-2.

The basic switch circuit for a transistor is shown in Figure 10-5.

The current flowing from base to emitter (b to e) controls the larger current flowing from the collector to emitter. If no current flows into the base, then no current will flow through the load. In most transistors, if the load had zero resistance, the current flowing into the collector would be 50 to 200 times the base current. However, we are going to be switching our transistor fully on or fully off, so the load resistance will always limit the collector current to the current required by the load. Too much base current will damage the transistor and also rather defeat the objective of controlling a bigger current with a smaller one, so the base will have a resistor connected to it.

When switching from an Arduino board, the maximum current of an output is 40 mA, so we could choose a resistor that allows about 30 mA to flow when the output pin is high at 5V. Using Ohm's Law:

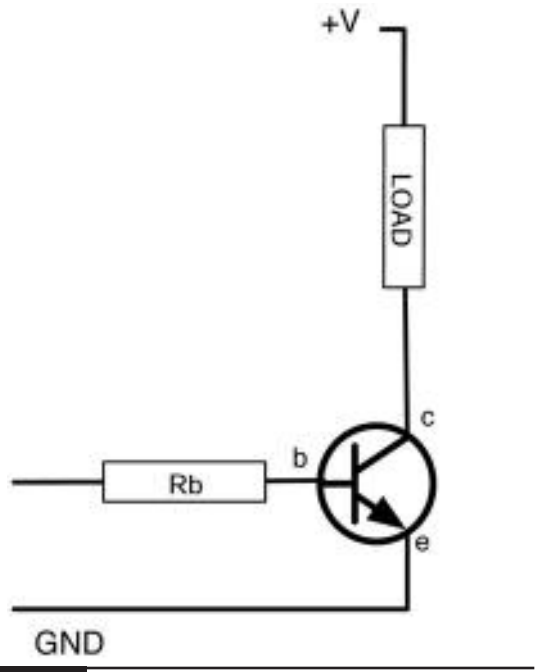


Figure 10-5 Basic transistor switch circuit.

$$R = V/I$$

$$R = (5 - 0.6)/30 = 147\ \Omega$$

The – 0.6 is because one characteristic of bipolar transistors is that there is always a voltage of about 0.6V between base and emitter when a transistor is turned on.

Therefore, using a 150 Ω base resistor, we could control a collector current of 40 to 200 times 30 mA, or 1.2 A to 6 A, which is more than

TABLE 10-2 Transistors Used in This Book

Transistor	Type	Purpose
BC548	Bipolar NPN	Switching small loads greater than 40 mA
BD139	Bipolar NPN power	Switching higher-load currents (e.g., Luxeon LED). See Project 6.
BC307	Bipolar PNP	Switching common anode LED displays where total current is too much for Arduino output (40 mA)
2N7000	N-channel FET	Low-power switching with very low 'on' resistance. See Project 7.
FQP33N10	N-channel power MOSFET	High-power switching.
FQP27P06	P-channel power MOSFET	High-power switching.

enough for most purposes. In practice, we would probably use a resistor of 1 KΩ or perhaps 270 Ω.

Transistors have a number of maximum parameter values that should not be exceeded or the transistor may be damaged. You can find these by looking at the datasheet for the transistor. For example, the datasheet for a BC548 will contain many values. The ones of most interest to us are summarized in Table 10-3.

TABLE 10-3 Transistor Datasheet

Property	Value	What It Means
I_c	100 mA	The maximum current that can flow through the collector without the transistor being damaged.
h_{FE}	110-800	DC current gain. This is the ratio of collector current to base current, and as you can see, could be anything between 110 and 800 for this transistor.

Other Semiconductors

The various projects have introduced a number of different types of components, from LEDs to temperature sensors, Table 10-4 provides some pointers into the various projects. If you want to develop your own project that senses temperature or whatever, first read about the projects developed by the author that use these components.

It may even be worth building the project and then modifying it to your own purposes.

Modules and Shields

It does not always make sense to make everything from scratch. That is, after all, why we buy an Arduino board rather than make our own. The same is true of some modules that we may want to use in our projects.

TABLE 10-4 The Use of Specialized Components in Pprojects

Component	Project
Single-color LEDs	Almost every project
Multicolor LEDs	14
LED matrix displays	16
7-segment LEDs	15, 30
Audio amplifier chip	19, 20
LDR (light sensor)	20
Thermistor (temperature sensor)	13
Variable voltage regulator	7

For instance, the LCD display module that we used in Projects 17 and 22 contains the driver chip needed to work the LCD itself, reducing both the amount of work we need to do in the sketch and the number of pins we need to use.

Other types of modules are available that you may wish to use in your projects. Suppliers such as Sparkfun are a great source of ideas and modules. A sample of the kinds of modules that you can get from such suppliers includes

- GPS
- Wi-Fi
- Bluetooth
- Zigbee wireless
- GPRS cellular modem

You will need to spend time reading through datasheets, planning, and experimenting, but that is what being an Evil Genius is all about.

Slightly less challenging than using a module from scratch is to buy an Arduino shield with the module already installed. This is a good idea when the components that you would like to use will not go on a breadboard (such as surface mount

devices). A ready-made shield can give you a real leg up with a project.

New shields become available all the time, but at the time of this writing, you can buy Arduino shields for:

- Ethernet (connect your Arduino to the Internet)
- XBee (a wireless data connection standard used in home automation, among other things)
- Motor driver
- GPS
- Joystick
- SD card interface
- Graphic LCD touch screen display
- Wi-Fi

Buying Components

Thirty years ago, the electronic enthusiast living in even a small town would be likely to have the choice of several radio/TV repair and spare stores where they could buy components and receive friendly advice. These days, there are a few retail outlets that still sell components, like RadioShack in the United States and Maplins in the UK, but the Internet has stepped in to fill the gap, and it is now easier and cheaper than ever to buy components.

With international component suppliers such as RS and Farnell you can fill a virtual shopping basket online and have the components arrive in a day or two. Shop around, because prices vary considerably between suppliers for the same components.

You will find eBay to be a great source of components. If you don't mind waiting a few weeks for your components to arrive, there are great bargains to be had from China. You often have to buy large quantities, but may find it cheaper to get 50 of a component from China than 5 locally. That way, you have some spares for your component box.

Tools

When making your own projects, there are a few tools that you will need at a bare minimum. If you do not intend to do any soldering, then you will need

- Solid-core wire in a few different colors, something around 0.6 mm (23 swg) wire diameter
- Pliers and wire snips, particularly for making jumper wires for the breadboard
- Breadboard
- Multimeter

If you intend to solder, then you will also need

- Soldering iron (duh)
- Lead-free alloy solder

Component Box

When you first start designing your own projects, it will take you some time to gradually build up your stock of components. Each time you are finished with a project, a few more components will find their way back to your stock.

It is useful to have a basic stock of components so that you do not have to keep ordering things when you just need a different value resistor. You will have noticed that most of the projects in this book tend to use values of resistor, like 100 Ω, 1 KΩ, 10 KΩ, etc. You actually don't need that many different components to cover most of the bases for a new project.

A good starting kit of components is listed in the appendix.

Boxes with compartments that can be labeled save a lot of time in selecting components, especially resistors that do not have their value written on them.

Snips and Pliers

Snips are for cutting, and pliers are for holding things still (often while you cut them).

Figure 10-6 shows how you strip the insulation off wire. Assuming you are right-handed, hold your pliers in your left hand and the snips in the right. Grip the wire with the pliers close to where you want to start stripping the wire from and then gently pinch round the wire with the snips and then pull sideways to pull the insulation away. Sometimes, you will pinch too hard and cut or weaken the wire, and other times you will not pinch hard enough and the insulation will remain intact. It's all just a matter of practice.

You can also buy an automatic wire stripper that grips and removes insulation in one action. In practice, these often only work well for one particular wire type and sometimes just plain don't work.

Soldering

You do not have to spend a lot of money to get a decent soldering iron. Temperature-controlled solder stations, such as the one shown in Figure 10-7, are better, but a fixed-temperature mains electricity iron is fine. Buy one with a fine tip, and make sure it is intended for electronics and not plumbing use.

Use narrow lead-free solder. Anyone can solder things together and make them work; however, some people just have a talent for neat soldering. Don't worry if your results do not look as neat as a robot-made printed circuit. They are never going to.

Soldering is one of those jobs that you really need three hands for: one hand to hold the soldering iron, one to hold the solder, and one to hold the thing you are soldering. Sometimes the thing you are soldering is big and heavy enough to stay put while you solder it; on other occasions, you will need to hold it down. Heavy pliers are good for this, as are mini vises and "helping hand" type holders that use little clips to grip things.

Figure 10-6 Snips and pliers.

Figure 10-7 Soldering iron and solder.

The basic steps for soldering are

1. Wet the sponge in the soldering iron stand.
2. Allow the iron to come up to temperature.
3. Tin the tip of the iron by pressing the solder against it until it melts and covers the tip.
4. Wipe the tip on the wet sponge—this produces a satisfying sizzling sound, but also cleans off the excess solder. You should now have a nice bright silver tip.
5. Touch the iron to the place where you are going to solder to heat it; then after a short pause (a second or two), touch the solder to the point where the tip of the iron meets the thing you are soldering. The solder should flow like a liquid, neatly making a joint.
6. Remove the solder and the soldering iron, putting the iron back in its stand, being very careful that nothing moves in the few seconds that the solder will take to solidify. If something does move, then touch the iron to it again to reflow the solder; otherwise, you can get a bad connection called a dry joint.

Above all, try not to heat sensitive (or expensive) components any longer than necessary, especially if they have short leads.

Practice soldering any old bits of wire together or wires to an old bit of circuitboard before working on the real thing.

Multimeter

A big problem with electrons is that you cannot see the little monkeys. A multimeter allows you to see what they are up to. It allows you to measure voltage, current, resistance, and often other features too like capacitance and frequency. A cheap $10 multimeter is perfectly adequate for almost any purpose. The professionals use much more solid and accurate meters, but that's not necessary for most purposes.

Multimeters, such as the one shown in Figure 10-8, can be either analog or digital. You can tell more from an analog meter than you can a digital, as you can see how fast a needle swings over and how it jitters, something that is not possible with a digital meter, where the numbers just change. However, for a steady voltage, it is much easier to read a digital meter, as an analog meter will have a number of scales, and you have to work out which scale you should be looking at before you take the reading.

You can also get autoranging meters, which, once you have selected whether you are measuring current or voltage, will automatically change ranges for you as the voltage or current increases. This is useful, but some would argue that thinking about the range of voltage before you measure it is actually a useful step.

To measure voltage using a multimeter:

1. Set the multimeter range to voltage (start at a range that you know will be higher than the voltage you are about to measure).
2. Connect the black lead to GND. A crocodile clip on the negative lead makes this easier.
3. Touch the red lead to the point whose voltage you want to measure. For instance, to see if an Arduino digital output is on or off, you can

Figure 10-8 A multimeter.

touch the red lead to the pin and read the voltage, which should be either 5V or 0V.

Measuring current is different from measuring voltage because you want to measure the current flowing through something and not the voltage at some point. So you put the multimeter in the path of the current that you are measuring. This means that when the multimeter is set to a current setting, there will be a very low resistance between the two leads, so be careful not to short anything out with the leads.

Figure 10-9 shows how you could measure the current flowing through an LED.

To measure current:

1. Set the multimeter range to a current range higher than the expected current. Note that multimeters often have a separate high-current connector for currents as high as 10 A.
2. Connect the positive lead of the meter to the more positive side from which the current will flow.
3. Connect the negative lead of the meter to the more negative side. Note that if you get this the wrong way round, a digital meter will just indicate a negative current; however, connecting an analog meter the wrong way round may damage it.
4. In the case of an LED, the LED should still light as brightly as before you put the meter into the circuit and you will be able to read the current consumption.

Another feature of a multimeter that is sometimes useful is the continuity test feature. This will usually beep when the two test leads are connected together. You can use this to test fuses,

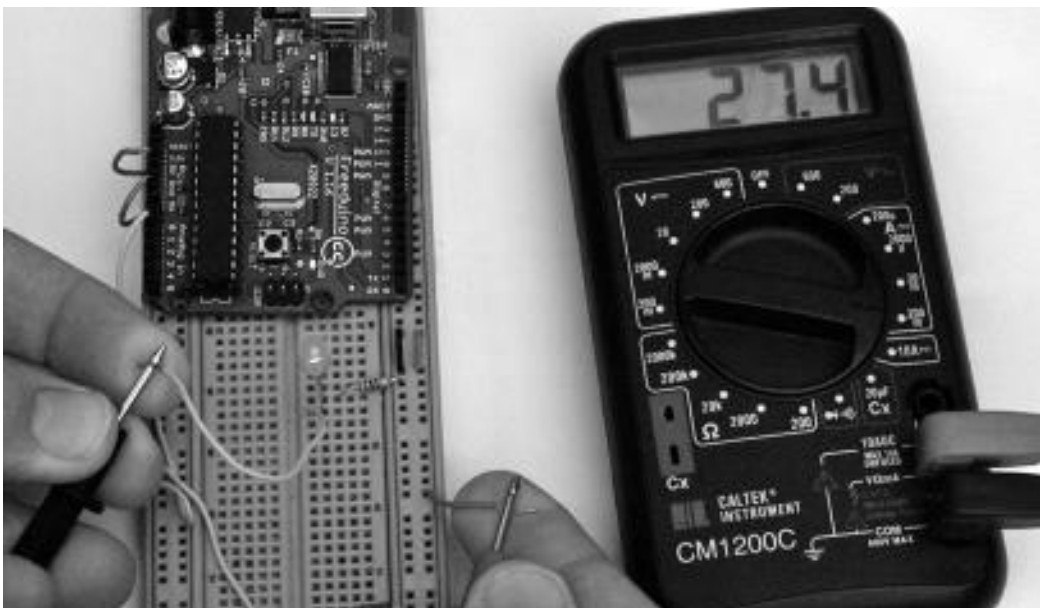


Figure 10-9 Measuring current.

etc., but also to test for accidental short circuits on a circuit board or broken connections in a wire.

Resistance measurement is occasionally useful, particularly if you want to determine the resistance of an unmarked resistor, for instance.

Some meters also have diode and transistor test connections, which can be useful to find and discard transistors that have burned out.

Oscilloscope

In Project 18, we built a simple oscilloscope. They are an indispensable tool for any kind of electronics design or test where you are looking at a signal that changes over time. They are a relatively expensive bit of equipment, and there are various types. One of the most cost-effective types is similar in concept to Project 19. That oscilloscope just sends its readings across to a computer that is responsible for displaying them.

Entire books have been written about using an oscilloscope effectively, and every oscilloscope is different, so we will just cover the basics here.

As you can see from Figure 10-10, the screen showing the waveform is displayed over the top of a grid. The vertical grid is in units of some fraction of volts, which on this screen is 2V per division. So the voltage of the square wave in total is 2.5 × 2, or 5V.

The horizontal axis is the time axis, and this is calibrated in seconds—in this case, 500 microseconds (mS) per division. So the length of one complete cycle of the wave is 1000 mS, that is, 1 millisecond, indicating a frequency of 1KHz.

Project Ideas

The Arduino Playground on the main Arduino website (www.arduino.cc) is a great source of ideas for projects. Indeed, it even has a section specifically for project ideas, divided into easy, medium, or difficult.

If you type “Arduino project” into your favorite search engine or YouTube, you will find no end of interesting projects that people have embarked on.

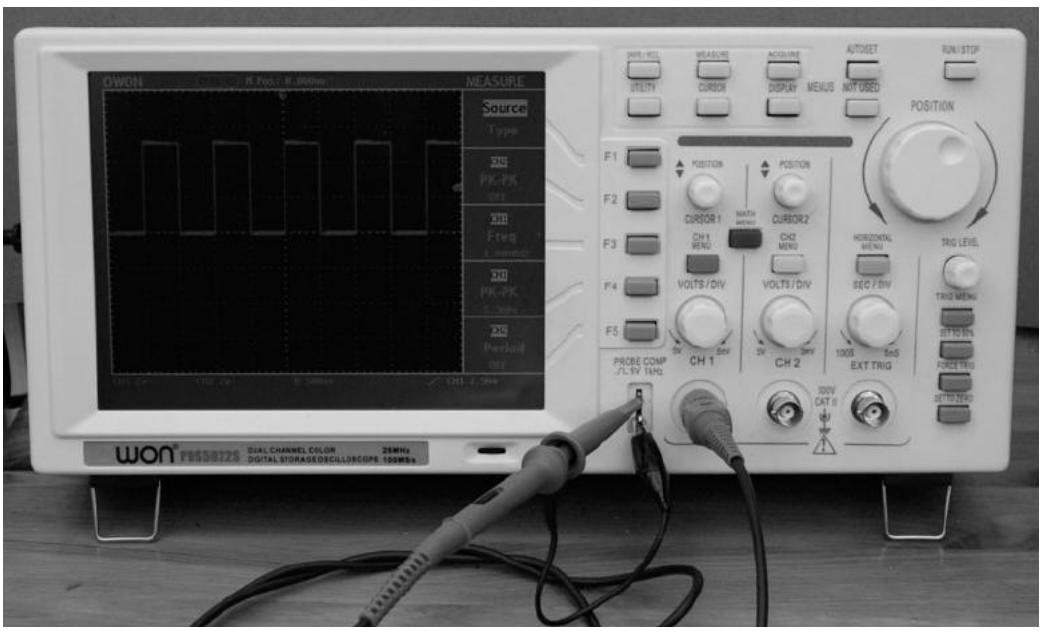


Figure 10-10 An oscilloscope.

Another source of inspiration is the component catalog, either online or on paper. Browsing through, you might come across an interesting component and wonder what you could do with it. Thinking up a project is something that should be allowed to gestate in the mind of the Evil Genius. After exploring all the options and mulling everything over, the Evil Genius' project will start to take shape!

APPENDIX

Components and Supplies

All of the parts used in this book are readily available through the Internet. However, sometimes it is a little difficult to track down exactly what you are looking for. For this reason, this appendix lists the components along with some order codes for various suppliers. This is information that will become inaccurate with time, but the big suppliers like Farnell and RS will usually list an item as "no longer stocked" and offer alternatives.

Suppliers

There are so many component suppliers out there that it feels a little unfair to list the few that the author knows. So have a look around on the Internet, as prices vary considerably between suppliers.

I have listed order codes for Farnell and RS because they are international, but also carry a fantastically broad range of stock. There is very little that you cannot buy from them. They can be surprisingly cheap for common components, like resistors and semiconductors, but for unusual components like laser diode modules, their prices can be ten times what you can find elsewhere on the Internet. Their main role is to supply to professionals.

Some smaller suppliers specialize in providing components for home constructors building microcontroller projects like ours. They do not have the range of components, but do often have more exotic and fun components at reasonable prices. A prime example of this is Sparkfun Electronics, but there are many others out there.

Sometimes, when you find you just need a couple of components, it's great to be able to go to a local store and pick them up. RadioShack in the United States and Maplins in the UK stock a range of components, and are great for this purpose.

The sections that follow list components by type, along with some possible sources and order codes where available.

Arduino and Clones		
Code	**Description**	**RS**
1	Arduino Duemilanove	696-1655
2	Arduino Lilypad	—
3	Arduino Shield Kit	696-1673

Other suppliers to check include eBay, Sparkfun, Robotshop.com, and Adafruit.

Resistors			
Code	**Description**	**Farnell**	**RS**
4	39 Ω 0.5W metal film	9338756	683-3601
5	100 Ω 0.5W metal film	9339760	683-3257
6	270 Ω 0.5W metal film	9340300	148-360A
7	1 KΩ 0.5W metal film	9339779	477-7928
8	4.7 KΩ 0.5W metal film	9340629	683-3799
9	10 KΩ 0.5W metal film	9339787	683-2939
10	33 KΩ 0.5W metal film	9340424	683-3544
11	47 KΩ 0.5W metal film	9340637	506-5434
12	56 KΩ 0.5W metal film	9340742	683-4206
13	100 KΩ 0.5W metal film	9339795	683-2923
14	470 KΩ 0.5W metal film	9340645	683-3730
15	1 MΩ 0.5W metal film	9339809	683-4159
16	4 Ω 1W	1155042	683-5477
17	100 KΩ linear potentiometer	1227589	249-9266
18	Thermistor, NTC, 33K at 25C, beta 4090	1672317RL (note, beta = 3950)	188-5278 (note R = 30K, beta = 4100)
19	LDR	7482280	596-141

Capacitors			
Code	**Description**	**Farnell**	**RS**
20	100nF nonpolarized	1200414	538-1203A
21	220nF nonpolarized	1216441	107-029
22	100 μF electrolytic	1136275	501-9100

Semiconductors				
Code	**Description**	**Farnell**	**RS**	**Other**
23	5-mm red LED	1712786	247-1151	Local store
24	5-mm yellow LED	1612434	229-2554	
25	5-mm green LED	1461633	229-2548	
26	5-mm IR LED sender 940 nm	1020634	455-7982	
27	3-mm red LED	7605481	654-2263	
28	3-mm green LED	1142523	619-2852	
29	3-mm blue LED	1612441	247-1561	
30	1W white Luxeon LED	1106587	467-7698	eBay
31	RGB LED (common anode)	1168585 (note: this has separate leads rather than common anode)	247-1505 (note: this has separate leads rather than common anode)	eBay
32	3mW red laser diode module	$$$	$$$	eBay or salvage from cheap laser pointer
33	2-digit, 7-segment LED display (common anode)	1003316	195-136	
34	8 x 8 LED array (2 color)	—	—	Sparkfun
35	10-segment bar graph display	1020492	—	
36	IR phototransistor 935 nm	1497882	195-530	
37	IR remote control receiver 940 nm	4142822	315-387	
38	1N4004 diode	9109595	628-9029	
39	BC307/ BC556 transistor	1611157	544-9309A	
40	BC548 transistor	1467872	625-4584	
41	DB139 transistor	1574350	—	
42	2N7000 FET	9845178	671-4733	
43	N-channel power MOSFET. FQP33N10	9845534	671-5095	
44	P-channel power MOSFET. FQP27P06	9846530	671-5064	
45	LM317 voltage regulator	1705990	686-9717	
46	4017 decade counter	1201278	519-0120	
47	TDA7052 1W audio amplifier	526198	658-485A	

Other suppliers to check, especially for LEDs, etc., include eBay, Sparkfun, Robotshop.com, and Adafruit.

Other				
Code	**Description**	**Farnell**	**RS**	**Other**
48	Miniature push switch	1448152	102-406	
49	2.1-mm Power plug	1200147	455-132	Local store
50	9V battery clip	1650667	489-021A	Local store
51	Regulated 15V 1.5A power supply	1354828	238-151	
52	3-way screw terminal	1641933	220-4276	
53	Perf board	1172145	206-8648	Local store
54	4 x 3 keypad	1182232	115-6031	
55	0.1-inch header strip	1097954	668-9551	
56	0.1-inch socket strip	1218869	277-9584	
57	Rotary encoder with push switch	1520815	—	
58	LCD module (HD44780 controller)	1137380	532-6408	eBay, Sparkfun
59	Miniature 8 Ω loudspeaker	1300022	628-4535	Local store
60	Electret microphone	1736563	—	
61	5V relay	9913734	499-6595	Local store
62	12V 1A power supply	1279478	234-238	Local store
63	12V computer cooling fan	1755867	668-8842	Local store
64	6V motor	599128	238-9721	Salvage
65	9g servo motor	—	—	eBay
66	5V solenoid (< 100 mA)	9687920	533-2217	
67	Piezotransducer (without driver electronics)	1675548	511-7670	Local store
68	Piezobuzzer (integrated electronics)	1192513	—	Local store
69	Miniature reed switch	1435590	289-7884	
70	7 x 5 inch picture frame			Supermarket
71	5V power supply	1297470	234-222	Salvaged phone charger
72	Breadboard	4692597	—	Local store

Local stores like RS and Maplins allow you to see components before you buy, and are good for components like power supplies and computer fans, which will usually be cheaper than the big professional suppliers.

Starter Kit of Components

It's good to have a bit of a stock of common components. The following list gives some components that you are likely to find yourself using over and over again.

- Resistors: 0.5W metal film, 100 Ω, 270 Ω, 1 KΩ, 10 KΩ, 100 KΩ
- 5-mm red LEDs
- Transistors: BC548, BD139

This page intentionally left blank

Index

References to figures are in italics.

R

S

T

U

V

W