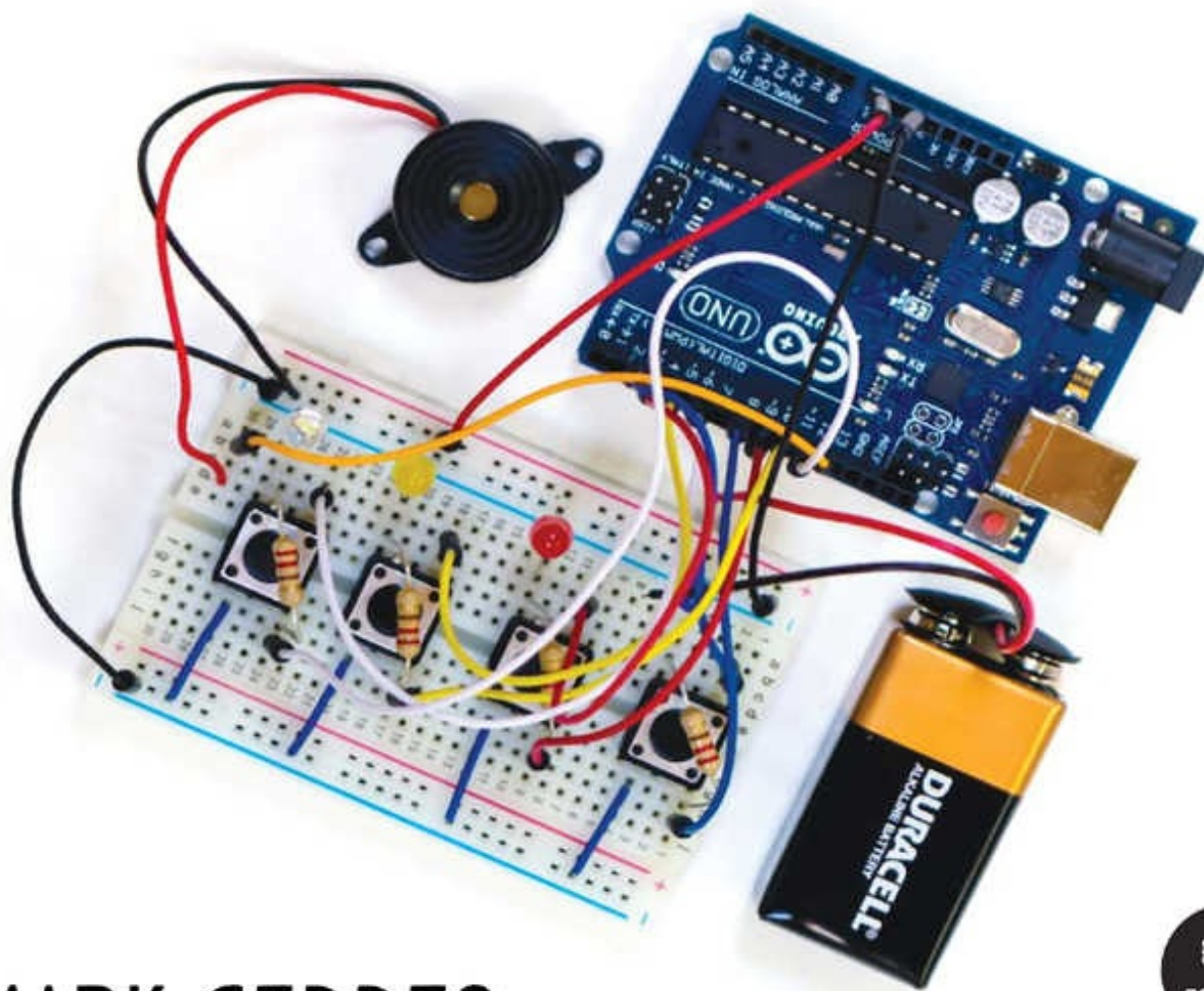


ARDUINO PROJECT HANDBOOK

25 PRACTICAL PROJECTS TO GET YOU STARTED



MARK GEDDES



ARDUINO PROJECT HANDBOOK

25 PRACTICAL PROJECTS TO GET YOU STARTED

MARK GEDDES



**no starch
press**

SAN FRANCISCO

ARDUINO PROJECT HANDBOOK. Copyright © 2016 by Mark Geddes.

All rights reserved. No part of this work may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage or retrieval system, without the prior written permission of the copyright owner and the publisher.

Printed in USA

First printing

20 19 18 17 16 1 2 3 4 5 6 7 8 9

ISBN-10: 1-59327-690-7

ISBN-13: 978-1-59327-690-4

Publisher: William Pollock

Production Editor: Serena Yang

Cover and Interior Design: Beth Middleworth

Developmental Editor: Liz Chadwick

Technical Reviewer: Christopher Stanton

Copyeditor: Rachel Monaghan

Compositor: Serena Yang

Proofreader: James Fraleigh

Circuit diagrams made using Fritzing (<http://fritzing.org/>).

For information on distribution, translations, or bulk sales, please contact No Starch Press, Inc. directly:

No Starch Press, Inc.

245 8th Street, San Francisco, CA 94103

phone: 415.863.9900; info@nostarch.com

www.nostarch.com

Library of Congress Cataloging-in-Publication Data:

Names: Geddes, Mark.

Title: Arduino project handbook : 25 practical projects to get you started /
by Mark Geddes.

Description: San Francisco : No Starch Press, [2016] | Includes index.

Identifiers: LCCN 2015033781 | ISBN 9781593276904 | ISBN 1593276907

Subjects: LCSH: Programmable controllers. | Microcontrollers--Programming. |
Science projects--Design and construction. | Arduino (Programmable
controller)

Classification: LCC TJ223.P76 G433 2016 | DDC 629.8/9551--dc23

LC record available at <http://lccn.loc.gov/2015033781>

No Starch Press and the No Starch Press logo are registered trademarks of No Starch Press, Inc. Other product and company names mentioned herein may be the trademarks of their respective owners. Rather than use a trademark symbol with every occurrence of a trademarked name, we are using the names only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

The information in this book is distributed on an “As Is” basis, without warranty. While every precaution has been taken in the preparation of this work, neither the author nor No Starch Press, Inc. shall have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the information contained in it.

CAMERON AND JEMMA, YOU ARE THE CREATORS AND MAKERS OF THE
FUTURE. THIS BOOK IS FOR YOU!

CONTENTS

ACKNOWLEDGMENTS

INTRODUCTION

PROJECT 0: GETTING STARTED

PART 1: LEDES

PROJECT 1: PUSHBUTTON-CONTROLLED LED

PROJECT 2: LIGHT DIMMER

PROJECT 3: BAR GRAPH

PROJECT 4: DISCO STROBE LIGHT

PROJECT 5: PLANT MONITOR

PROJECT 6: GHOST DETECTOR

PART 2: SOUND

PROJECT 7: ARDUINO MELODY

PROJECT 8: MEMORY GAME

PROJECT 9: SECRET KNOCK LOCK

PART 3: SERVOS

PROJECT 10: JOYSTICK-CONTROLLED LASER

PROJECT 11: REMOTE CONTROL SERVO

PART 4: LCDS

PROJECT 12: LCD SCREEN WRITER

PROJECT 13: WEATHER STATION

PROJECT 14: FORTUNE TELLER

PROJECT 15: REACTION TIMER GAME

PART 5: NUMERIC COUNTERS

PROJECT 16: ELECTRONIC DIE

PROJECT 17: ROCKET LAUNCHER

PART 6: SECURITY

PROJECT 18: INTRUDER SENSOR

PROJECT 19: LASER TRIP WIRE ALARM

PROJECT 20: SENTRY GUN

PROJECT 21: MOTION SENSOR ALARM

PROJECT 22: KEYPAD ENTRY SYSTEM

PROJECT 23: WIRELESS ID CARD ENTRY SYSTEM

PART 7: ADVANCED

PROJECT 24: RAINBOW LIGHT SHOW

PROJECT 25: BUILD YOUR OWN ARDUINO!

APPENDIX A: COMPONENTS

APPENDIX B: ARDUINO PIN REFERENCE

CONTENTS IN DETAIL

ACKNOWLEDGMENTS

INTRODUCTION

THE ARDUINO REVOLUTION

ABOUT THIS BOOK

ORGANIZATION OF THIS BOOK

PROJECT 0: GETTING STARTED

HARDWARE

THE ARDUINO UNO

POWER

BREADBOARDS

JUMPER WIRES

PROGRAMMING THE ARDUINO

THE IDE INTERFACE

ARDUINO SKETCHES

LIBRARIES

TESTING YOUR ARDUINO: BLINKING AN LED

UNDERSTANDING THE SKETCH

PROJECT COMPONENT LIST

SETTING UP YOUR WORKSPACE

EQUIPMENT AND TOOL GUIDE

QUICK SOLDERING GUIDE

SAFETY FIRST

PART 1: LEDES

PROJECT 1: PUSHBUTTON-CONTROLLED LED

HOW IT WORKS

THE BUILD

THE SKETCH

PROJECT 2: LIGHT DIMMER

HOW IT WORKS

THE BUILD

THE SKETCH

PROJECT 3: BAR GRAPH

HOW IT WORKS
THE BUILD
THE SKETCH

PROJECT 4: DISCO STROBE LIGHT

HOW IT WORKS
THE BUILD
THE SKETCH

PROJECT 5: PLANT MONITOR

HOW IT WORKS
THE BUILD
THE SKETCH

PROJECT 6: GHOST DETECTOR

HOW IT WORKS
THE BUILD
THE SKETCH

PART 2: SOUND

PROJECT 7: ARDUINO MELODY

HOW IT WORKS
THE BUILD
THE SKETCH

PROJECT 8: MEMORY GAME

HOW IT WORKS
THE BUILD
THE SKETCH

PROJECT 9: SECRET KNOCK LOCK

HOW IT WORKS
THE BUILD
THE SKETCH

PART 3: SERVOS

PROJECT 10: JOYSTICK-CONTROLLED LASER

HOW IT WORKS
THE BUILD
MOUNTING THE LASER

THE SKETCH

PROJECT 11: REMOTE CONTROL SERVO

HOW IT WORKS

THE SETUP

THE BUILD

THE SKETCH

PART 4: LCDS

PROJECT 12: LCD SCREEN WRITER

HOW IT WORKS

PREPARING THE LCD SCREEN

THE BUILD

THE SKETCH

PROJECT 13: WEATHER STATION

HOW IT WORKS

THE BUILD

THE SKETCH

PROJECT 14: FORTUNE TELLER

HOW IT WORKS

THE BUILD

THE SKETCH

PROJECT 15: REACTION TIMER GAME

HOW IT WORKS

THE BUILD

THE SKETCH

PART 5: NUMERIC COUNTERS

PROJECT 16: ELECTRONIC DIE

HOW IT WORKS

THE BUILD

THE SKETCH

PROJECT 17: ROCKET LAUNCHER

HOW IT WORKS

THE BUILD

CREATE A WORKING FUSE

THE SKETCH

PART 6: SECURITY

PROJECT 18: INTRUDER SENSOR

HOW IT WORKS

THE BUILD

THE SKETCH

PROJECT 19: LASER TRIP WIRE ALARM

HOW IT WORKS

THE BUILD

THE SKETCH

PROJECT 20: SENTRY GUN

HOW IT WORKS

THE BUILD

THE SKETCH

PROJECT 21: MOTION SENSOR ALARM

HOW IT WORKS

THE BUILD

THE SKETCH

PROJECT 22: KEYPAD ENTRY SYSTEM

HOW IT WORKS

TESTING THE KEYPAD

THE BUILD

THE SKETCH

PROJECT 23: WIRELESS ID CARD ENTRY SYSTEM

HOW IT WORKS

THE BUILD

THE SKETCH

PART 7: ADVANCED

PROJECT 24: RAINBOW LIGHT SHOW

HOW IT WORKS

THE BUILD

THE SKETCH

PROJECT 25: BUILD YOUR OWN ARDUINO!

HOW IT WORKS

PREPARING THE CHIP

BUILDING THE ARDUINO CIRCUIT

APPENDIX A: COMPONENTS

COMPONENTS GUIDE

ARDUINO UNO R3

9V BATTERY PACK

BREADBOARD

LED

RESISTOR

PUSHBUTTON

POTENTIOMETER

HL-69 SOIL SENSOR

PIEZO BUZZER

SERVOMOTOR

JOYSTICK

INFRARED LED RECEIVER

LCD SCREEN

DHT11 HUMIDITY SENSOR

TILT BALL SWITCH

RGB LED

SEVEN-SEGMENT LED DISPLAY

FOUR-DIGIT, SEVEN-SEGMENT SERIAL DISPLAY

ULTRASONIC SENSOR

PHOTORESISTOR

RC V959 MISSILE LAUNCHER

PIR SENSOR

KEYPAD

RFID READER

RGB MATRIX

SHIFT REGISTER

ATMEGA328P CHIP

16 MHZ CRYSTAL OSCILLATOR

5V REGULATOR

CAPACITOR

DISC CAPACITOR

BATTERY CLIP

RETAILER LIST

DECODING RESISTOR VALUES

APPENDIX B: ARDUINO PIN REFERENCE

ACKNOWLEDGMENTS

Many thanks to the fantastic team at No Starch Press, particularly Elizabeth Chadwick and Serena Yang, for their support and guidance in the creation of this book. Thanks to Christopher Stanton for his technical reviews and suggestions.

This book wouldn't exist if it wasn't for the inspirational Arduino founders: Massimo Banzi, David Cuartielles, Tom Igoe, Gianluca Martino, and David Mellis. Thank you for introducing me and the world to the wonder that is Arduino. Thanks also to Ken Shirriff, Warwick Smith, Steven de Lannoy, and Abdullah Alhazmy for kind permission to reproduce their projects.

I have to thank my wonderful wife, Emily, for being so supportive and patient over the last year and for freeing up a room in our house as a “man cave” so I could put together all these projects—and for resisting the temptation to tidy it on a daily basis!

Thank you to my parents, Frank and Lorna, for allowing me the freedom as a child to take things apart and for not complaining when I had wires everywhere. If not for their support, I wouldn't have the passion for electronics and gadgets that I still have today. Thanks also to David and Linda for their fantastic support, encouragement, and belief.

INTRODUCTION

The Arduino is a small, inexpensive computer that can be programmed to control endless creations limited only by your imagination. As you'll soon see, the Arduino can be used to make a whole host of projects, like a ghost detector, joystick-controlled laser, electronic die, laser trip wire alarm, motion sensor alarm, keypad entry system, and many others. All of these projects are easy to build and have one thing in common—they use the power of the Arduino.

In the early 1980s, I picked up a great Penguin paperback titled something like *Gadgets and Gizmos*, hidden away in a local bookstore. The projects were simple ones like making a working lighthouse using flashlight bulbs and building a revolving display table using an old clock. The ideas in that book sparked my imagination, and I've been creating ever since.

My curiosity led me to take apart various electrical items to experiment with and find out how they worked. I usually struggled to put them back together but amassed a good selection of components to tinker with. (This is a great way of gathering lots of parts, by the way.)

I remember wiring together a string of small flashlight bulbs to make floodlights for my Subbuteo table-top soccer game and creating a loudspeaker system to blast out music at the halftime break in a game. I even managed to extract some LEDs from a Star Wars toy, only to burn them out because I didn't understand what a resistor was at the time. I used small motors, buzzers, and solar cells to create burglar alarms and super whizzy cars, and I burned out a few motors too!

At roughly the same time (1983), Sinclair Research in the United Kingdom launched the ZX Spectrum 48k microcomputer, introducing home computing to the UK mass market. (The United States had its Commodore 64.) While intended as a serious computer, the ZX Spectrum inadvertently lent itself more to gaming due to its inclusion of the simple programming language BASIC. As a result, software houses sprouted in bedrooms across the country as people rushed to build games for the ZX.

This sparked my interest in programming, but at the time I couldn't combine my two passions. Physical computing, where software and hardware react to the physical world, was around in the '80s but confined to the realm of very high-end computing and robotics, way out of reach of most households. Now, some 30 years later, with the introduction of the Arduino, I find myself tinkering again with electronics, but this time I can use programming to bring the projects to life.

THE ARDUINO REVOLUTION

In simple terms, the Arduino is a small computer that can be programmed to connect and control various electronic parts. The Arduino has a number of pins that can be set as either *input*, which means they can receive data from items such as switches, buttons, and sensors, or *output*, which means they send data to control items such as motors, lights, and buzzers. This type of programmable development board is better known as a *microcontroller*.

The Arduino project began in Ivrea, Italy, in 2005 with the goal of creating a device to control

student-built interactive design projects that would be less expensive than other prototyping systems available at the time. Founders Massimo Banzi and David Cuartielles named the project after a local bar called Arduino (an Italian masculine first name meaning “strong friend”).

The Arduino board is composed of two main elements: the hardware, or microcontroller, which is the brain of the board, and the software that you’ll use to send your program to the brain. The software, called the *Arduino integrated development environment (IDE)*, is available free for download.

The IDE is a simple interface that runs on a computer running Windows, OS X, or Linux. You use the IDE to create a *sketch* (an Arduino program) that you then upload to the Arduino board using a PC and USB cable. The sketch tells the hardware what to do. I’ll go into both the hardware and software in more detail in the next couple of chapters.

The Arduino can be powered by batteries, USB, or an external power supply. Once the Arduino is programmed, it can be disconnected from your computer and will run independently using a power supply or batteries.

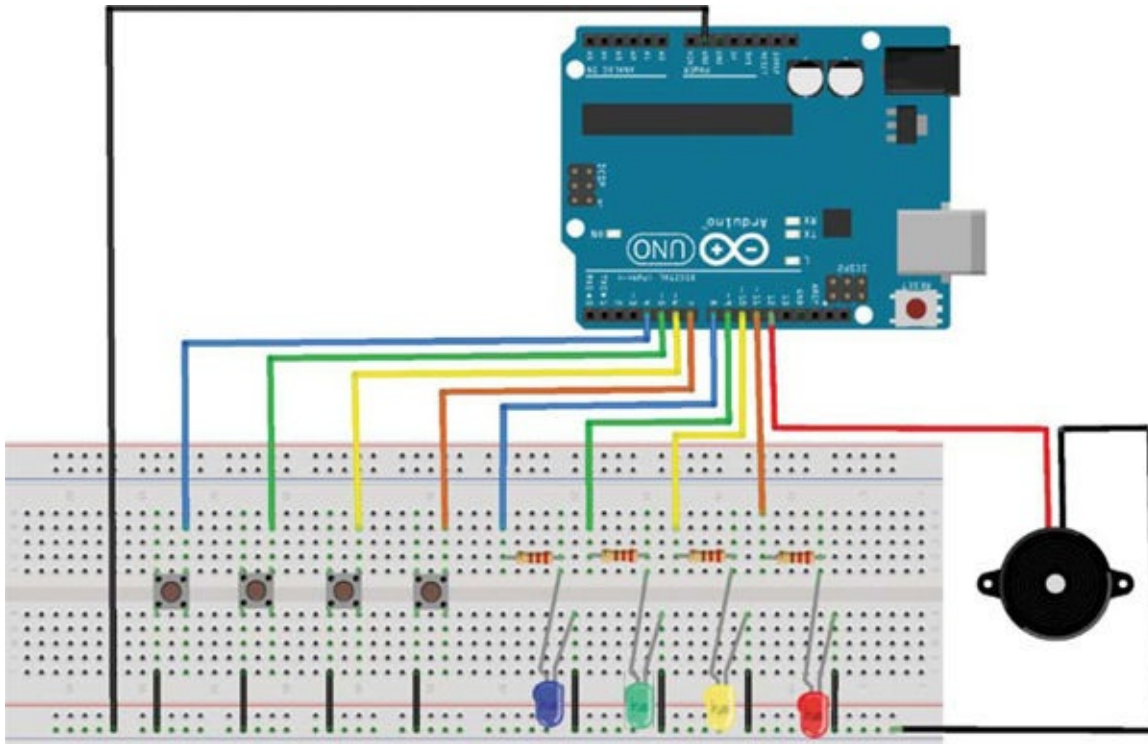
ABOUT THIS BOOK

What was it that encouraged me to write this book? The Internet is bursting with tutorials, videos, and articles covering the Arduino and potential projects, but many lack detailed visuals or the code required to build these projects. Like the *Gizmos and Gadgets* book that inspired me many years ago, this book is intended to help you build simple projects that will inspire you to create your own contraptions using the skills and techniques that you’ll learn.

In this book you’ll concentrate on creating your project on a breadboard. This is the best way to learn about how circuits work, because the connections are not permanent; if you make a mistake, you can just unplug the wire or component and try again. Each project has step-by-step instructions for connecting the main components, along with photographs to help you with layout. Tables are used in most projects for quick reference.

The projects will have a circuit diagram to show the connections clearly, like that in [Figure 1](#). These have been created with the Fritzing program (<http://www.fritzing.org/>), a free, open source program for creating visual schematics of your projects.

FIGURE 1:
Example of a Fritzing diagram



Each project also has the code required to program your Arduino, so you don't need to worry about learning to program before you begin. The early projects provide simple explanations of what's happening in the code, to help you understand the process of programming enough to make your own modifications if you want to. If you don't want to type all the code, you can download the programs from <http://www.nostarch.com/arduinohandbook/>.

The projects in this book begin with the basics and progress steadily to more complex designs. That said, this book won't go deeply into electronics theory or programming, but I will give you a good starting point. I've written this book to teach you how to create your own gadgets. By giving you the technical know-how, I allow you to focus on the creative design element. The idea is that learning the function of circuits can open up your imagination to ways of using those circuits practically.

This book gives practical information so you can, for example, reference the pin connections and replicate them when needed in a different project. You can also combine projects to make more complicated and interesting gadgets.

A lot of Arduino books focus on the programming element, and that's great for a certain kind of learning, but I think there is a place for plug-and-play electronics. By following the steps in the projects, you will learn as you go.

ORGANIZATION OF THIS BOOK

The book progresses from simple to more complex projects as follows to help you build your skills and learn about components:

Part I: LEDs You'll start by learning how to control simple LEDs with buttons and variable resistors, and then combine components to build disco strobe lights, plant monitors to tell you

when your plant needs watering, and even a ghost detector.

Part II: Sound In this part, you'll learn about the piezo buzzer, a very useful device that emits and can also detect sound. You'll make music with the Arduino Melody, create a simple and fun memory game, and set up a secret code lock system that detects the volume of a knock.

Part III: Servos These projects all use the servomotor, a small motor with an arm that can be used for a whole host of purposes. You'll build a joystick-controlled laser and decode a remote control so you can move your servo with buttons on the remote.

Part IV: LCDs The LCD screen is useful in lots of projects for displaying messages and results. In this part, you'll learn how to set up an LCD screen, build a weather station to report conditions, and set up two games: a fortune teller and a reaction timer game.

Part V: Numeric Counters You'll use LED number displays in this part to build an electronic die and a rocket launcher countdown system that sets off a fuse.

Part VI: Security These more complex projects will show you how to protect your space with trip wires and intruder trackers, motion sensors that trigger alarms or sentry missiles, and security systems that use keypads and card readers to keep unauthorized persons out.

Part VII: Advanced In this final part, you'll combine the Arduino with a matrix of lights to create the Rainbow Light Show. Then you'll round off your skills by building your own Arduino to use in future projects.

These projects don't have to be built in order, so if you see something you like and feel confident enough to take it on, you can skip to it. I recommend you try out some of the earlier projects first, though, as you'll find information there that's useful for more complicated builds.

I've written the book that I was looking for but couldn't find when I started out with the Arduino. I hope that you'll enjoy reading and working through this book as much as I've enjoyed writing it.


ARDUINO

BOARD MODEL
UNO R3
OPEN-SOURCE ELECTRONICS
PROTOTYPING PLATFORM
MADE IN ITALY
WWW.ARDUINO.CC



COMPLIANT FOOTPRINT
ROHS
ZERA
TTPATTO
CARBON
ZERO

PROJECT 0: GETTING STARTED

BEFORE YOU START BUILDING WITH THE ARDUINO, THERE ARE A FEW THINGS YOU NEED TO KNOW AND DO. LET'S TAKE A LOOK AT THE HARDWARE AND SOFTWARE YOU'LL NEED FOR THIS BOOK AND HOW TO SET UP A WORKSTATION. YOU'LL THEN TEST OUT THE ARDUINO WITH A SIMPLE LED PROJECT AND GET STARTED WITH A FEW TECHNIQUES THAT WILL COME IN HANDY, LIKE SOLDERING AND DOWNLOADING USEFUL CODE LIBRARIES.

HARDWARE

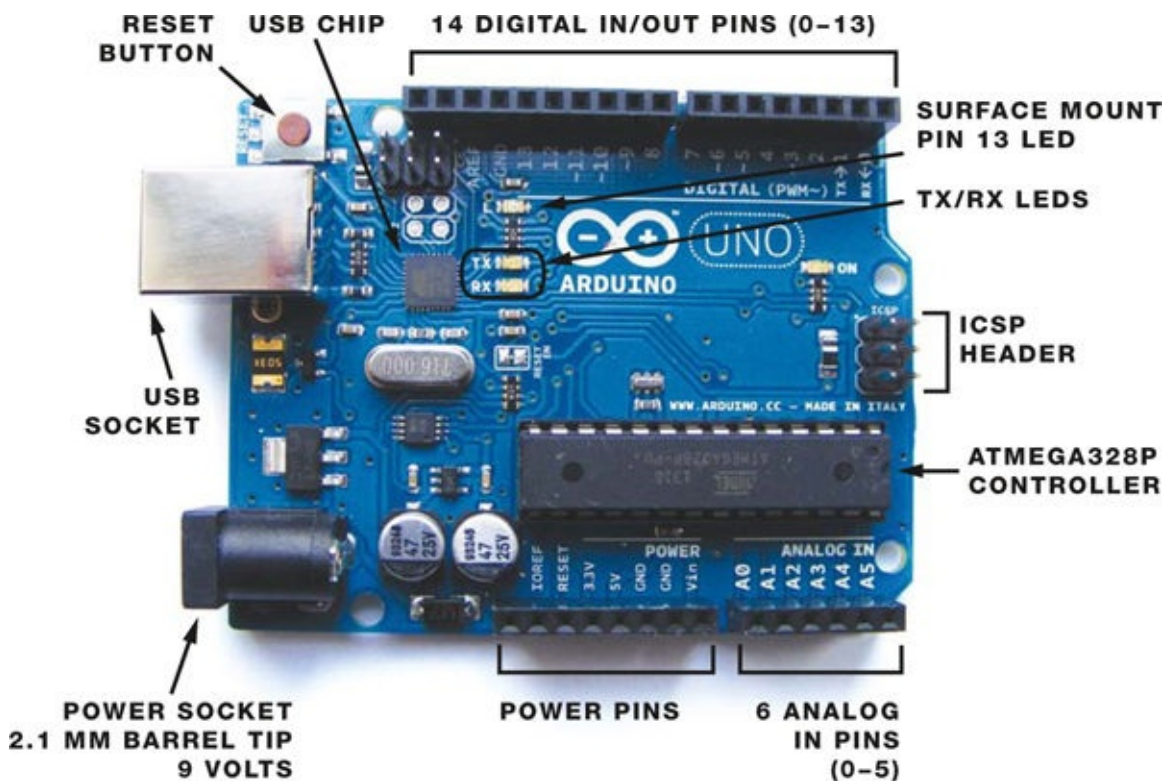
First let's look at the Arduino Uno board and a few pieces of hardware that you'll use in almost every project.

The Arduino Uno

There are numerous types of Arduino boards available, but this book will exclusively use the most popular one—the Arduino Uno, shown in [Figure 0-1](#). The Arduino Uno is open source (meaning its designs may be freely copied), so in addition to the official board, which costs about \$25, you'll find numerous compatible clone boards for around \$15.

Let's walk through the different elements of the Arduino Uno.

FIGURE 0-1:
The Arduino Uno board



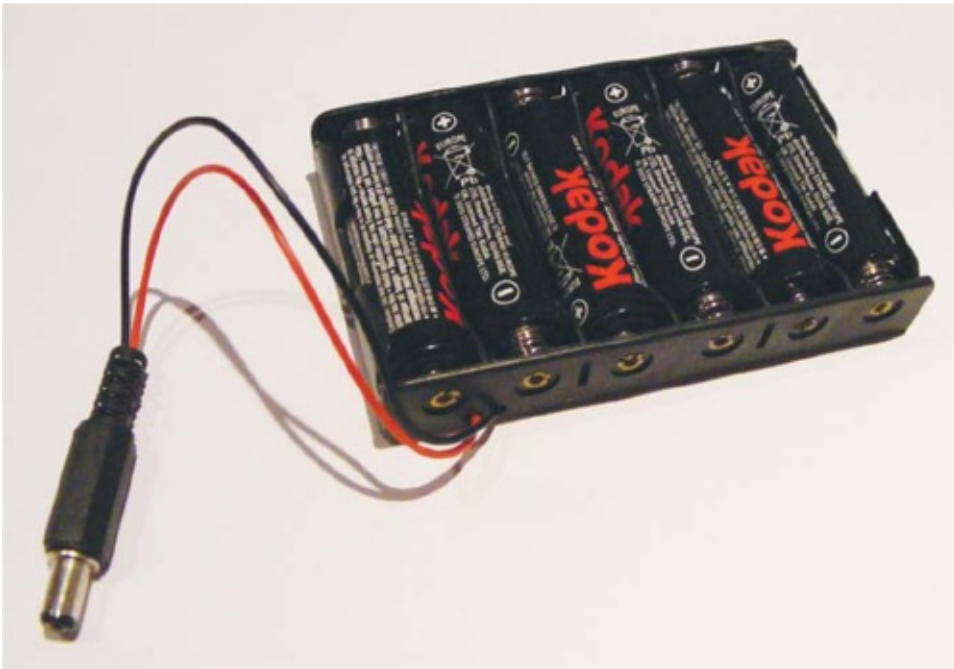
The Arduino controls components you attach to it, like motors or LEDs, by sending information to them as *output* (information sent *out* from the Arduino). Data that the Arduino reads from a sensor is *input* (information going *in* to the Arduino). There are 14 digital input/output pins (pins 0–13). Each can be set to either input or output, and [Appendix B](#) has a full pin reference table.

Power

The Arduino Uno board is powered from your computer’s USB port when you connect it to your PC to upload a program. When the Arduino is not linked to your PC, you can run it independently by connecting a 9-volt AC adapter or 9-volt battery pack with a 2.1 mm jack, with the center pin connected to the positive wire, shown in [Figure 0-2](#). Simply insert the jack into the power socket of the Arduino.

FIGURE 0-2:

A 9-volt battery pack, which you can plug into the Arduino to give it power

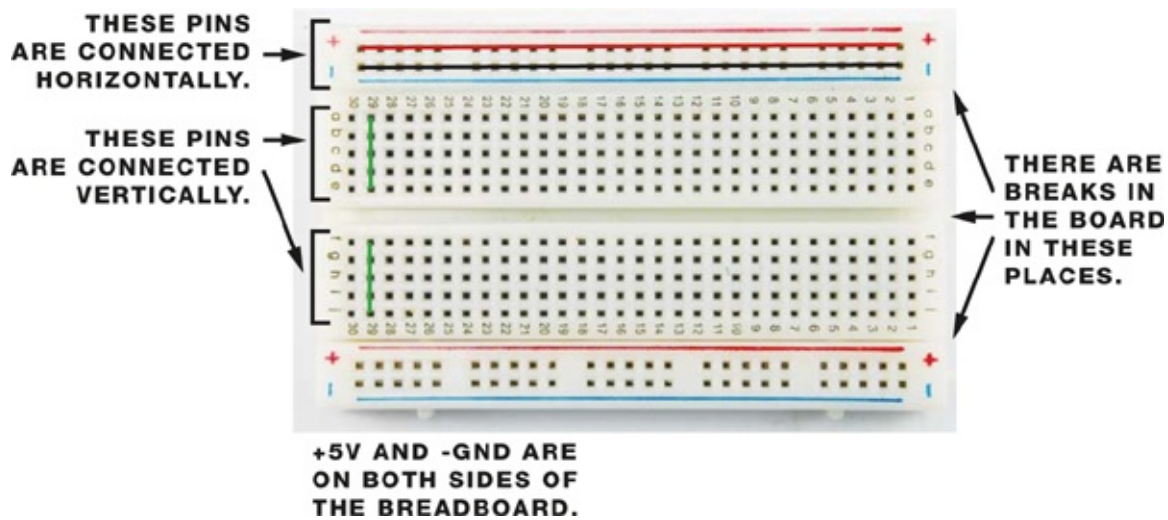


Breadboards

A *breadboard* acts as a construction base for electronics prototyping. All of the projects in this book use a breadboard instead of soldering.

The word *breadboard* dates back to when electronics projects were created on wooden boards. Nails were hammered into the wood and wires wrapped around them to connect components without the use of solder. Today's breadboards, such as the one shown in [Figure 0-3](#), are made of plastic with predrilled holes (called *tie points*) into which you insert components or wires that are held in place by clips. The holes are connected by strips of conductive material that run underneath the board.

FIGURE 0-3:
Breadboard connections



Breadboards come in various sizes. To build the projects in this book, you'll need four breadboards: two full-size, typically with 830 holes; one half-size with 420 holes; and one mini with 170 holes. The full-size breadboard is ideal for projects that use an LCD screen or a lot of components, and the half-size and mini boards are best for smaller projects. I recommend that for the projects in this book you buy breadboards that look like the one shown in [Figure 0-3](#), with red and blue lines and a center break between the holes.

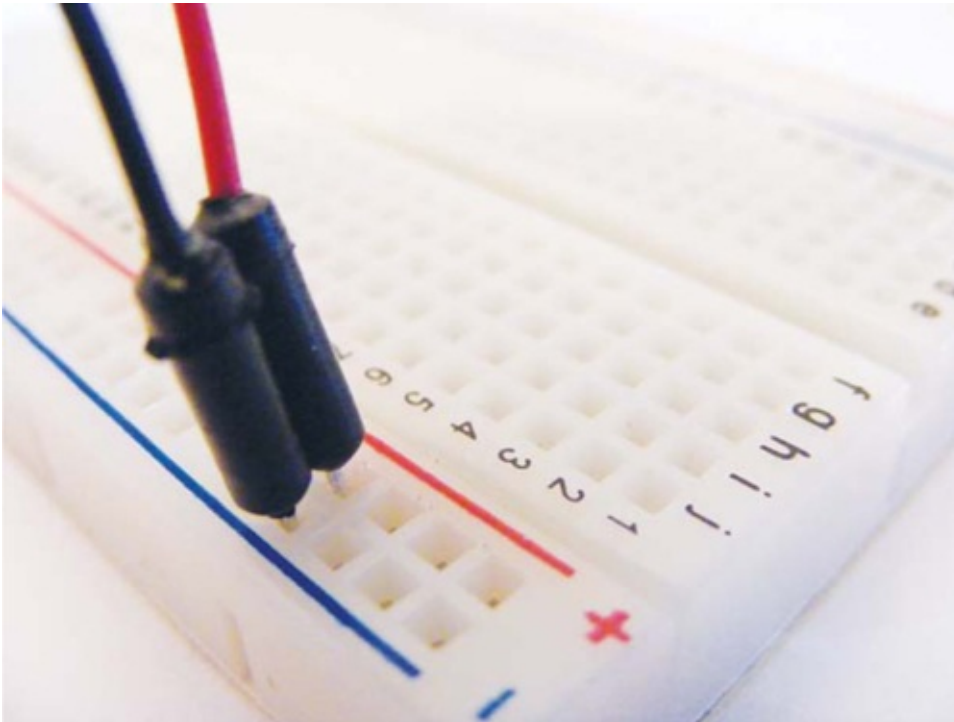
TIP

It's conventional to use red wires for connections to 5V and black wires for connections to ground (GND), so you can easily tell which is which. The rest of the wires can be your choice of color.

The main board area has 30 columns of tie points that are connected vertically, as shown in [Figure 0-3](#). There is a break in the center of the board, which you'll often have to straddle with components to make your circuit. This break helps to connect the pins individually so they are not shorted together unintentionally, which can doom your project and even damage your components.

The blue and red lines at the top and bottom are *power rails* that you use to power the components inserted in the main breadboard area (see [Figure 0-4](#)). The power rails connect all the holes in the rail horizontally; the red lines are for positive power and the blue lines for negative power (or *ground*, as you'll often see it referred to).

FIGURE 0-4:
Positive and negative breadboard rails

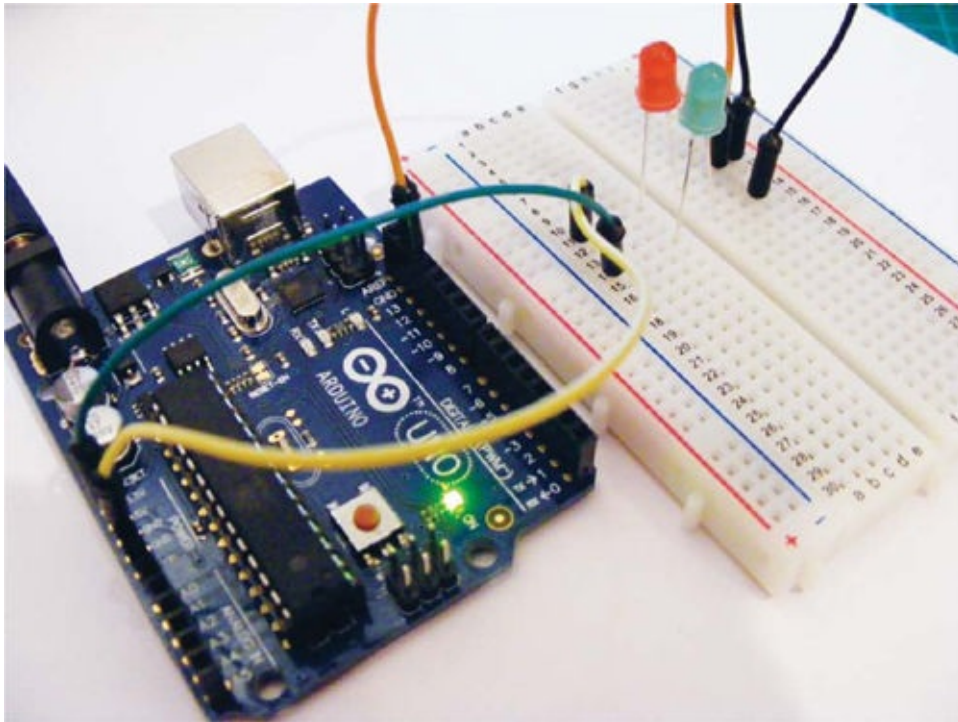


Jumper Wires

You'll use *jumper wires* to make connections on the breadboard. Jumper wires are solid-core wires with a molded plastic holder on each end that makes it easier to insert and remove the wires. (You could use your own wire if you have it, but make sure to use solid-core wire, as stranded wire is not strong enough to push into the hole clips.)

When you insert a jumper wire into a breadboard hole, it's held in place beneath the board by a small spring clip, making an electrical connection in that row that typically consists of five holes. You can then place a component in an adjoining hole to help create a circuit, as shown in [Figure 0-5](#).

FIGURE 0-5:
An example breadboard circuit



PROGRAMMING THE ARDUINO

To make our projects do what we want, we need to write programs that give the Arduino instructions. We do so using a tool called the Arduino *integrated development environment (IDE)*. The Arduino IDE is free to download from <http://www.arduino.cc/>, and will run on Windows, OS X, and Linux. It enables you to write computer programs (a set of step-by-step instructions, known as *sketches* in the Arduino world) that you then upload to the Arduino using a USB cable. Your Arduino will carry out the instructions based on its interaction with the outside world.

NOTE

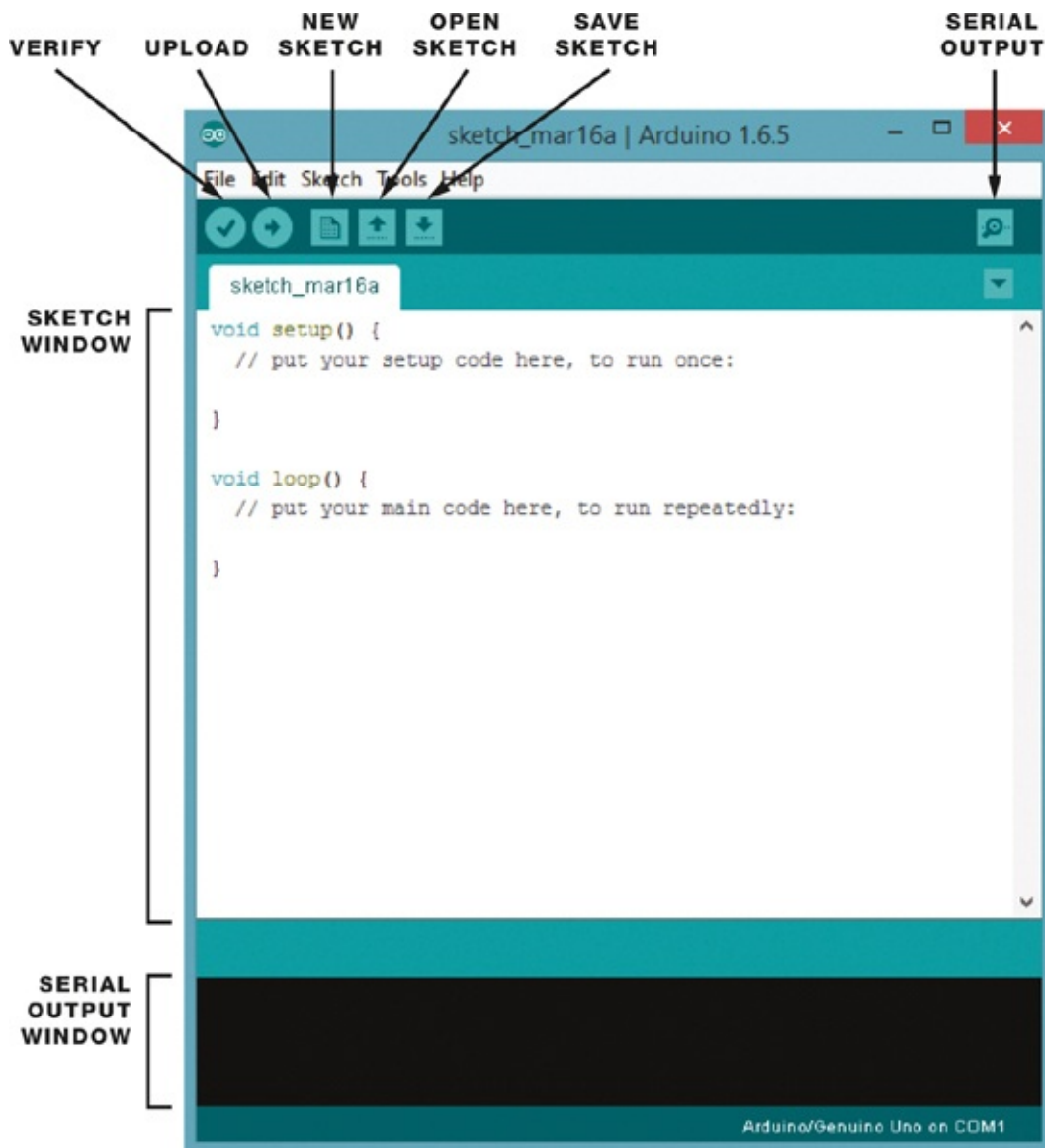
Because the IDE versions can change fairly quickly, I won't take you through installing them, but you should find installation straightforward. All versions of the IDE and full details of how to install for your operating system are available online at <http://www.arduino.cc/>.

The IDE Interface

When you open the Arduino IDE, it should look very similar to [Figure 0-6](#).

The IDE is divided into a toolbar at the top, with buttons for the most commonly used functions; the code or sketch window in the center, where you'll write or view your programs; and the Serial Output window at the bottom. The Serial Output window displays communication messages between your PC and the Arduino, and will also list any errors if your sketch doesn't compile properly.

FIGURE 0-6:
The Arduino IDE



Arduino Sketches

I'll give you the sketch for each project within the relevant project, and talk through it there. All of the sketches are available to download from <http://www.nostarch.com/arduinohandbook/>.

Like any program, sketches are a very strict set of instructions, and very sensitive to errors. To make sure you've copied the sketch correctly, press the green check mark at the top of the screen. This is the Verify button, and it checks for mistakes and tells you in the Serial Output window whether the sketch has compiled correctly. If you get stuck, you can always download the sketch and then copy and paste it into the IDE.

Libraries

In the Arduino world, a *library* is a small piece of code that carries out a specific function. Rather than enter this same code repeatedly in your sketches, you can add a command that borrows code from the library. This shortcut saves time and makes it easy for you to connect to items such as a sensor, display, or module.

The Arduino IDE includes a number of built-in libraries—such as the LiquidCrystal library, which makes it easy to talk to LCD displays—and there are many more available online. To create the projects in the book, you will need to import the following libraries: RFID, Tone, Pitches, Keypad, Password, Ultrasonic, NewPing, IRRemote, and DHT. You'll find all of the libraries you need at <http://www.nostarch.com/arduinohandbook/>.

Once you've downloaded the libraries, you'll need to install them. To install a library in Arduino version 1.0.6 and higher, follow these steps:

1. Choose **Sketch ▶ Include Library ▶ Add .ZIP Library**.
2. Browse to the ZIP file you downloaded and select it. For older versions of Arduino, you'll need to unzip the library file and then put the whole folder and its contents into the *sketchbook/libraries* folder on Linux, *My Documents\Arduino\Libraries* on Windows, or *Documents/Arduino/libraries* on OS X.

To install a library manually, go to the ZIP file containing the library and uncompress it. For example, if you were installing a library called *keypad* in a compressed file called *keypad.zip*, you would uncompress *keypad.zip*, which would expand into a folder called *keypad*, which in turn contains files like *keypad.cpp* and *keypad.h*. Once the ZIP file was expanded, you would drag the *keypad* folder into the *libraries* folder on your operating system: *sketchbook/libraries* in Linux, *My Documents\Arduino\Libraries* on Windows, and *Documents/Arduino/libraries* on OS X. Then restart the Arduino application.

Libraries are listed at the start of a sketch and are easily identified because they begin with the command `#include`. Libraries are surrounded by angle brackets, `<>`, and end with `.h`, as in the following call to the Servo library:

```
#include <Servo.h>
```

Go ahead and install the libraries you'll need for the projects now to save yourself a bit of time later.

TESTING YOUR ARDUINO: BLINKING AN LED

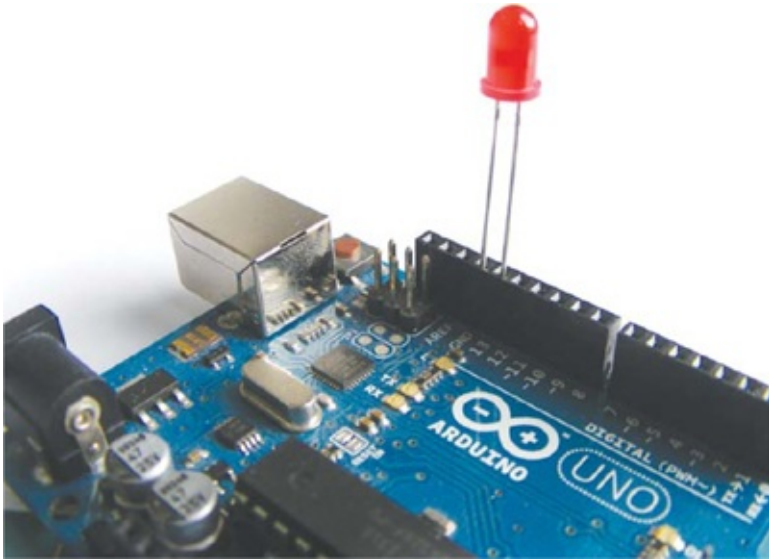
Now that you've seen the hardware and software, let's begin our tour with the classic first Arduino project: blinking a *light emitting diode (LED)*. Not only is this the simplest way to make sure that your Arduino is working correctly, but it will also introduce you to a simple sketch. As I mentioned earlier, a sketch is just a series of instructions that run on a computer. The Arduino can hold only one sketch at a time, so once you upload your sketch to your Arduino, that sketch will run every time the Arduino is switched on until you upload a new one.

For this project we'll use the *Blink* example sketch that comes with the Arduino IDE. This program turns on an LED for 1 second and then off for 1 second, repeatedly. An LED emits light when a small current is passed through it. The LED will work only with current flowing in one direction, so the longer wire must connect to a positive power connection. LEDs also require a current limiting resistor; otherwise, they may burn out. There is a built-in resistor inline with pin 13 of the Arduino.

Follow these steps to set up your test:

1. Insert the long positive leg (also known as +5V or *anode*) of the LED into pin 13 on the Arduino, as shown in [Figure 0-7](#). Connect the short negative leg (also known as *cathode*) to the GND pin next to pin 13.

1. **FIGURE 0-7:**
The *Blink* project setup



2. Connect the Arduino to your computer with the USB cable.
3. Enter the following sketch into the IDE.

```
❶ // Blinking LED Project
❷ int led = 13;
❸ void setup() {
❹   pinMode(led, OUTPUT);
   }
❺ void loop() {
❻   digitalWrite(led, HIGH);
❼   delay(1000);
❽   digitalWrite(led, LOW);
❾   delay(1000);
❿ }
```

4. Click the **Verify** button (which looks like a check mark) to confirm that the sketch is working correctly.
5. Now click the **Upload** button to send the sketch to your Arduino.

Understanding the Sketch

Here's what's happening on each line of the sketch:

- ❶ This is a comment. Any line in your program starting with `//` is meant to be read by the user only, and is ignored by the Arduino, so use this technique to enter notes and describe your code (called *commenting* your code). If a comment extends beyond one line, start the first line with `/*` and end the comment with `*/`. Everything in between will be ignored by the Arduino.
- ❷ This gives pin 13 the name `led`. Every mention of `led` in the sketch refers to pin 13.
- ❸ This means that the code between the curly brackets, `{}`, that follow this statement will run once when the program starts. The open curly bracket, `{`, begins the setup code.

- ④ This tells the Arduino that pin 13 is an output pin, indicating that we want to send power to the LED. The close curly bracket, `}`, ends the `setup` code.
- ⑤ This creates a loop. Everything between the curly brackets, `{}`, after the `loop()` statement will run once the Arduino is powered on and then repeat until it is powered off.
- ⑥ This tells the Arduino to set `led` (pin 13) to `HIGH`, which sends power to that pin. Think of it as switching the pin on. In this sketch, this turns on the LED.
- ⑦ This tells the Arduino to wait for 1 second. Time on the Arduino is measured in milliseconds, so 1 second = 1,000 milliseconds.
- ⑧ This tells the Arduino to set `led` (pin 13) to `LOW`, which removes power and switches off the pin. This turns off the LED.
- ⑨ Again the Arduino is told to wait for 1 second.
- ⑩ This closing curly bracket ends the loop. All code that comes after the initial `setup` must be enclosed within curly brackets. A common cause of errors in a sketch is missing open or close brackets, which will prevent your sketch from compiling correctly. After this curly bracket, the sketch goes back to the start of the loop at ⑤.

Running this code should make your LED flash on and off. Now that you've tested your Arduino and understand how a sketch works and how to upload it, we'll take a look next at the components you'll need to carry out all of the projects in this book. [Appendix A](#) has more details about each component, what it looks like, and what it does.

PROJECT COMPONENT LIST

This is a complete list of the items you'll need in order to complete the projects in this book. The most important part, of course, is the Arduino board itself—all projects use the Arduino Uno R3 version. As mentioned earlier, only the official boards are named Arduino, but clone boards compatible with the software can be bought from companies like SlicMicro, Sainsmart, and Adafruit and will be referred to as Uno R3 or Arduino Uno R3 compatible. (You'll find a list of official suppliers at <http://arduino.cc/en/Main/Buy/>.)

Each project will list the required items first, so if you want to complete only a few of the projects, you can flip to a project that appeals to you and obtain just those components. Although you can buy each item individually, I suggest buying an electronics hobby starter kit or Arduino kit. You'll find many of them online, and there is a list of suggested suppliers in [Appendix A](#). The components marked with an asterisk (*) can all be found in an Arduino Bare Bones Kit, which can save you a bit of time and money.

- 1 Arduino Uno R3 (or compatible alternative)
- 1 9V battery pack with 2.1 mm jack
- 2 full-size breadboards
- 1 half-size breadboard
- 1 mini breadboard
- 50 male-to-male jumper wires

10 female-to-male jumper wires
30 220-ohm resistors
10 330-ohm resistors
1 470-ohm resistor
1 10k-ohm resistor
1 1M-ohm resistor
40 5 mm LEDs: red, green, yellow, blue (10 of each color)
1 50k-ohm potentiometer
4 momentary tactile four-pin pushbuttons
1 HL-69 hygrometer soil moisture sensor
1 piezo buzzer
1 3.5 mm phone jack
2 Tower Pro SG90 9g servomotors
1 photoresistor (also known as a light resistor, or LDR)
1 analog five-pin, two-axis joystick module
1 pan-and-tilt housing module
1 four-pin HC-SR04 ultrasonic range sensor
1 4x4 membrane keypad
1 seven-segment LED display
1 four-digit, seven-segment serial display
1 DHT11 humidity sensor
1 16x2 LCD screen (Hitachi HD44780 compatible)
1 tilt ball switch
1 8x8 RGB LED matrix
1 38 kHz infrared (IR) sensor
1 HC SR501 PIR (passive infrared) sensor
1 Mifare RFID RC-522 reader, card, and fob
4 74HC595 shift registers
1 low-powered laser-pointer pen
1 WLToys RC V959 missile launcher
1 ATMEL ATmega328p chip*
1 16 MHz crystal oscillator (HC-495)*
1 L7805cv 5V regulator*
2 100 μ F electrolytic capacitors*
1 PP3 9V battery clip*
2 22 pF disc capacitors*
9V battery*

SETTING UP YOUR WORKSPACE

To get the most out of working with the Arduino, you should create a workspace that allows you to let your imagination loose but keeps you organized at the same time. If possible, it should also be a dedicated space, something like the one shown in [Figure 0-8](#); some projects can take a few hours to put together, so you may not have time to finish them all in one sitting, and there is nothing worse than having to stop and put everything away only to get it all out again next time.

FIGURE 0-8:
An example workspace



A workspace can be anywhere, but the main thing you will need is a table or flat surface big enough for your computer or laptop (so you can use the IDE and upload programs easily) and for you to actually do your building.

You may also want space to keep your components at hand as well as any tools you may need, such as a soldering iron, wire strippers, hobby knife, hobby drill, and so on. It may not be practical to have all of your tools and materials out all of the time, so it's a good idea to buy some hobby or craft cases to store your parts. I use one bin for equipment, like soldering irons or wire cutters, and smaller bins for components. Plastic boxes for fishing tackle or craft use are perfect for storing components (see [Figure 0-9](#)), and a cantilever toolbox is great to house your soldering iron and other small equipment ([Figure 0-10](#)). Small plastic boxes, usually designed to store jewelry or craft supplies, are also a good way to store very small components ([Figure 0-11](#)).

FIGURE O-9:
Tackle or craft boxes are handy for storing components.



FIGURE O-10:

A cantilever toolbox works well for storing a soldering iron and other small tools.



FIGURE O-11:

Plastic jewelry boxes are perfect for organizing very small items.



Consider buying a ledger-sized cutting mat to use as a defined and *nonconductive* workspace (one that doesn't pass electricity), so you won't run the risk of short-circuiting your sensitive electronics.

EQUIPMENT AND TOOL GUIDE

While they're not necessarily required for the projects in this book, here are some of the more useful pieces of equipment that you may consider buying when setting up a workspace.

- Helping hands—useful for holding items



- Ledger-sized, nonconductive cutting mat



- Needle-nose pliers



- Wire cutters



- 30-watt soldering iron and solder (see the [“Quick Soldering Guide”](#) on [page 18](#))
- Solder sucker to suck up solder!



- Wire stripper—especially useful for making jumper wires



- USB A-to-B cable for use with your Arduino



- Digital multimeter



- Screwdriver



- Rotary tool and attachments



- Glue gun



QUICK SOLDERING GUIDE

A few of the components you'll need may come without their header pins ([Figure 0-12](#)) attached for ease of transport, and you'll need to solder them in place. Header pins are rows of pins you attach to a component so you can make connections with jumper wires or insert into a breadboard. They come in strips that can be easily snapped to the size needed, and they are usually inserted into holes on the component designed for them.

FIGURE 0-12:
Header pins



The RFID module used in [Project 23](#), for example, doesn't come with the pins attached, so I'll demonstrate how to solder those in place now as a quick guide to soldering. If you want something more in-depth, there's a handy cartoon soldering guide at https://mightyohm.com/files/soldercomic/FullSolderComic_EN.pdf.

First you will need a soldering iron ([Figure 0-13](#)). A general-purpose, 30-watt soldering iron with a fine tip should meet your needs. It's worthwhile to buy a kit that includes a soldering iron, stand, and solder.

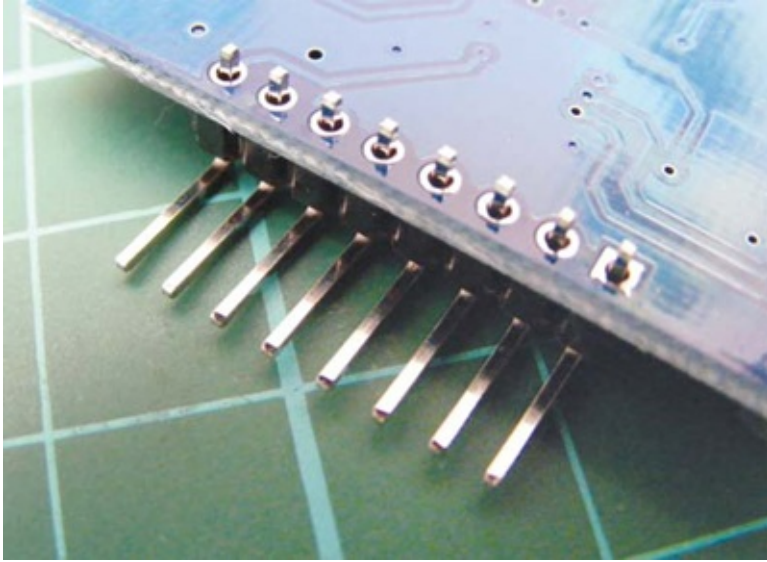
FIGURE 0-13:
Soldering iron and solder wire



To solder, you heat the area you want to solder with the soldering iron—for example, the place where the pin and the component meet—and then apply the soldering wire to the heated area; the wire quickly melts, and when it resets, it should create a clean connection between the two items you soldered. Here’s a demonstration.

1. Plug in your soldering iron and wait at least five minutes for it to reach operating temperature.
2. Break off the right number of header pins for your component. For the RFID module in [Project 23](#), we need a row of eight pins. Insert them into the module as shown in [Figure 0-14](#).

2. **FIGURE 0-14:**
Insert the header pins into the module.



NOTE

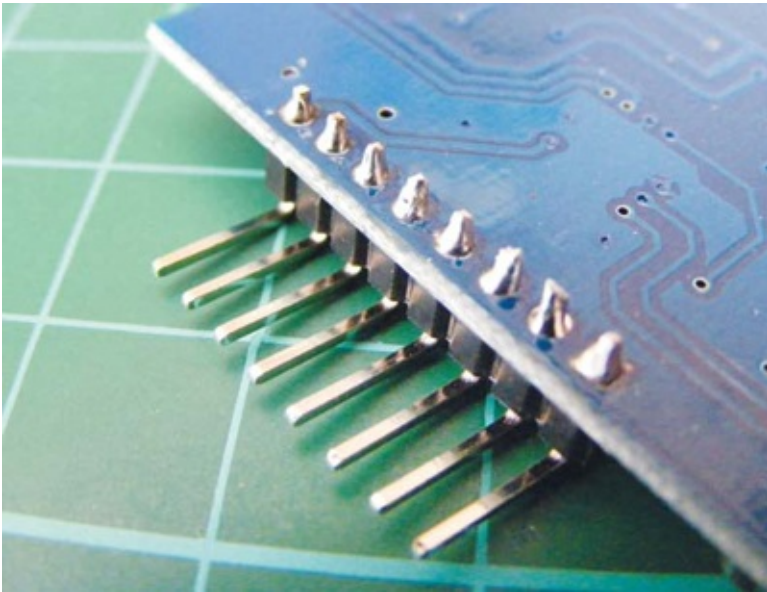
You do not apply solder directly to the iron, only to the joint you are soldering.

3. Now we will solder the pins in place. Start with the leftmost pin. Hold the heated tip of the soldering iron to the pin and module at the same time. You only need to hold it there for about two seconds. While holding the iron in place, add solder to the area; the solder should melt and create a joint.
4. Quickly remove both the iron and solder—more than a couple of seconds could damage your components. Wait for the joint to cool.

A good solder joint should be like a shiny cone ([Figure 0-15](#)). With a little bit of practice, you will be able to solder in no time at all.

FIGURE O-15:

Solder joints should look like this.



Safety First

Soldering irons get very, very hot and should be used with extreme care and not used by unsupervised children. Here are a few safety tips:

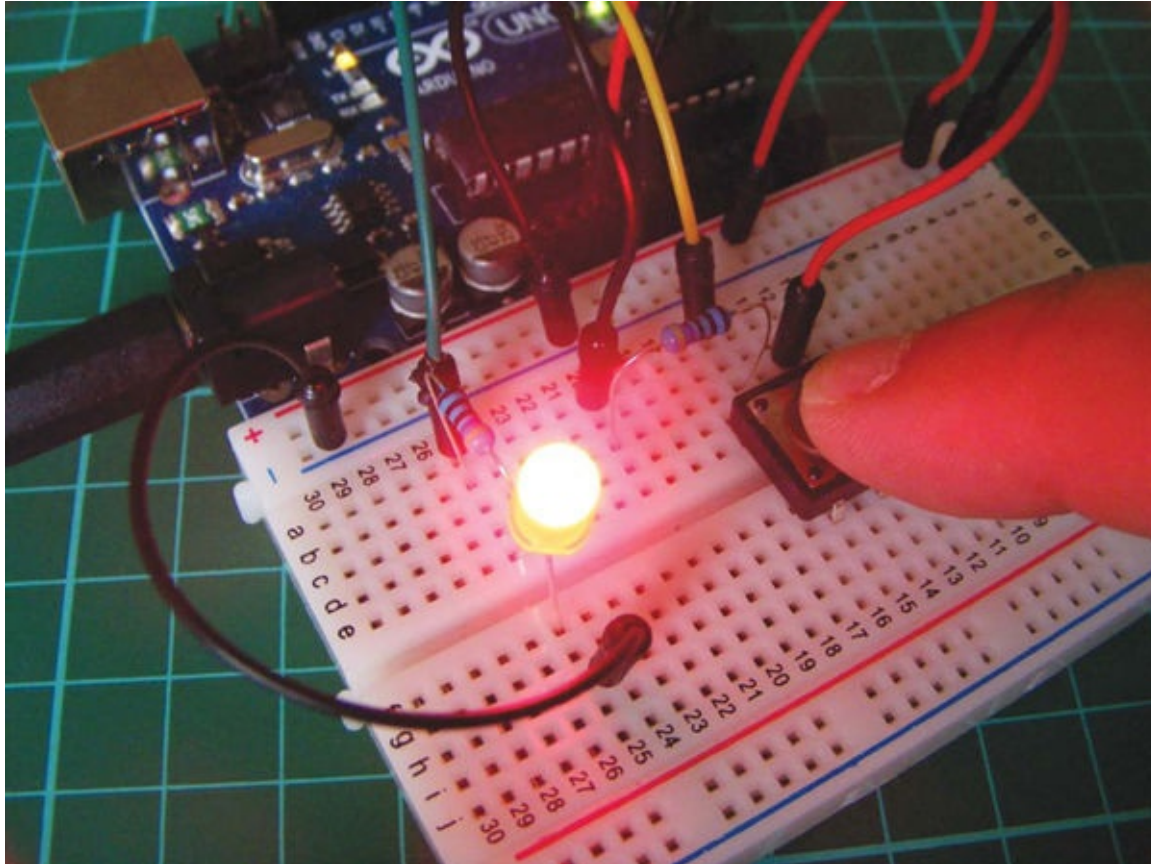
- Be sure to use a stand and never lay a hot soldering iron down on a table.
- Solder in a well-ventilated room. The fumes released from melting solder can be harmful.
- Keep flammable materials away from your work area.
- Keep equipment out of reach of children.
- Wear eye protection.
- Wait for a soldering iron to cool down completely before storing it.

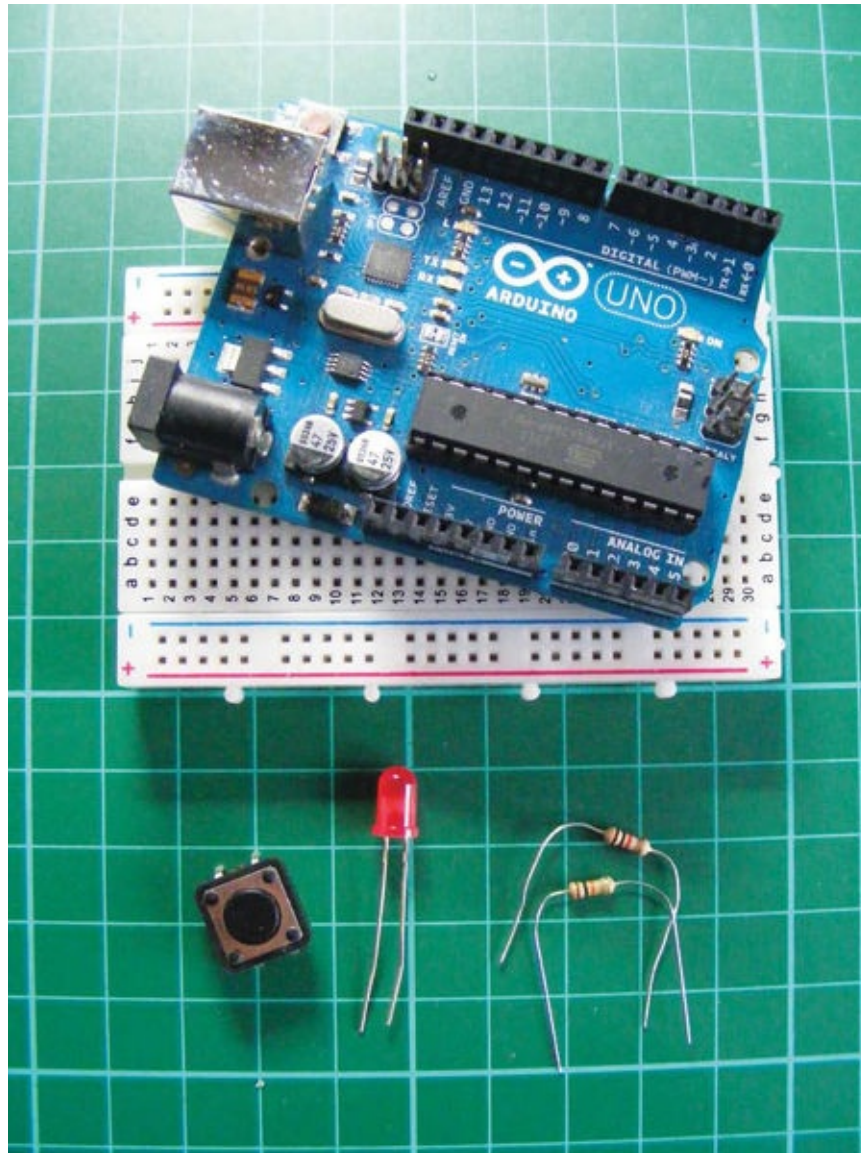
PART 1

LÉDS

PROJECT 1: PUSHBUTTON-CONTROLLED LED

IN THIS PROJECT, YOU'LL ADD A PUSHBUTTON SWITCH TO AN LED CIRCUIT TO CONTROL WHEN THE LED IS LIT.



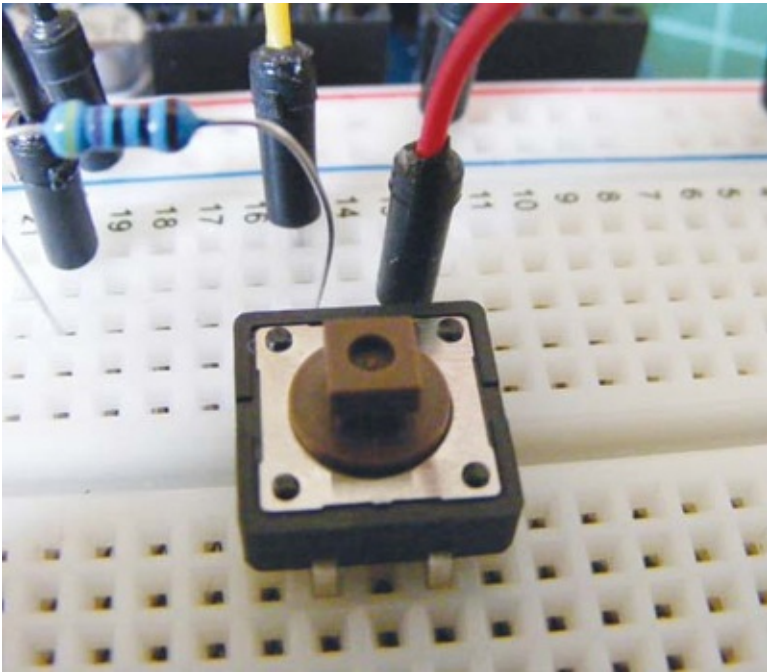


PARTS REQUIRED

- Arduino board
- Breadboard
- Jumper wires
- LED
- Momentary tactile four-pin pushbutton
- 10k-ohm resistor
- 220-ohm resistor

This project will take you through the basics of switches, which you'll be using a lot throughout this book. Almost all electrical items use switches to turn an element on or off. There are many types of switches, and the one you'll use now is a pushbutton ([Figure 1-1](#)).

FIGURE 1-1:
A pushbutton



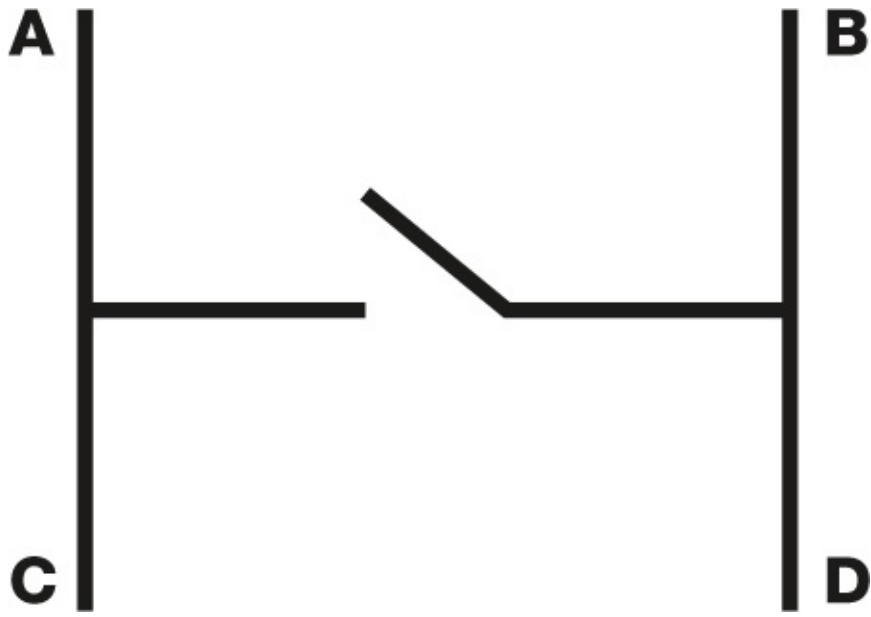
HOW IT WORKS

When pressed, a pushbutton completes a circuit, turning it on. As soon as the button is released, the connection will spring back and break that circuit, turning it off. The pushbutton switch is also known as a *momentary* or *normally open* switch, and is used in, for example, computer keyboards. This is in contrast to a *toggle switch*, which stays either on or off until you toggle it to the other position, like a light switch.

This type of pushbutton has four pins, but you generally use only two at a time for connection. You'll use the top connections in this project, although the two unused pins at the bottom would do the same job. As [Figure 1-2](#) shows, the pins work in a circuit. Pins A and C are always connected, as are pins B and D. When the button is pressed, the circuit is complete.

FIGURE 1-2:

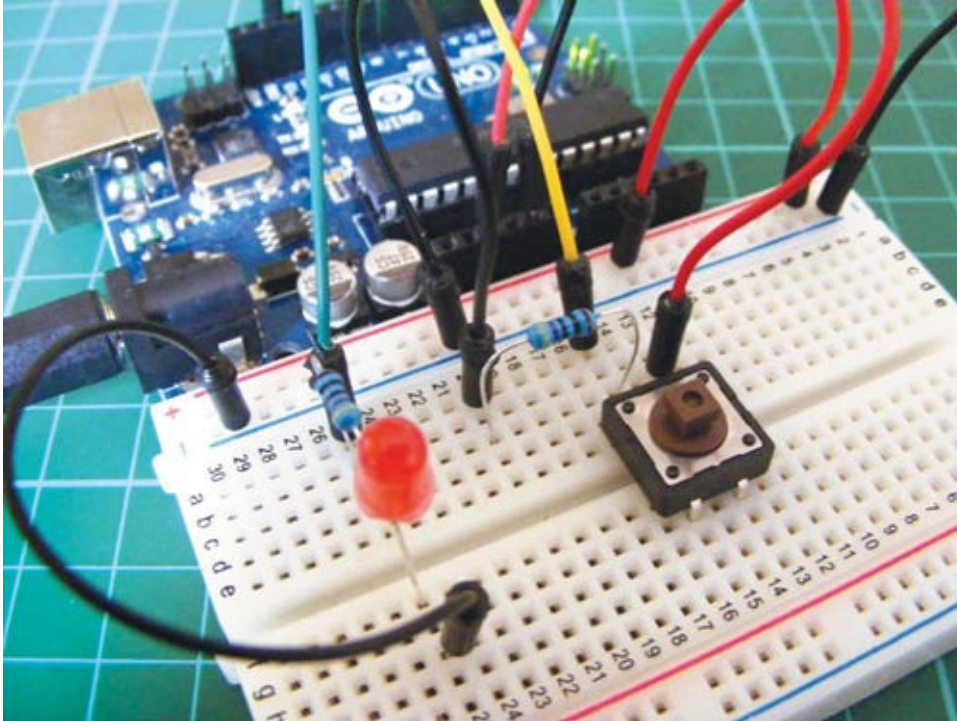
A pushbutton's incomplete circuit



THE BUILD

1. Place your pushbutton in a breadboard, as shown in [Figure 1-3](#).

- FIGURE 1-3:**
Placing your pushbutton



- Connect pin A to one leg of a 10k-ohm resistor, and connect that same resistor leg to Arduino pin 2. Connect the other resistor leg to the GND rail, and connect the GND rail to the Arduino's GND. Connect pin B on the switch to the +5V rail, and connect this rail to +5V on the Arduino.

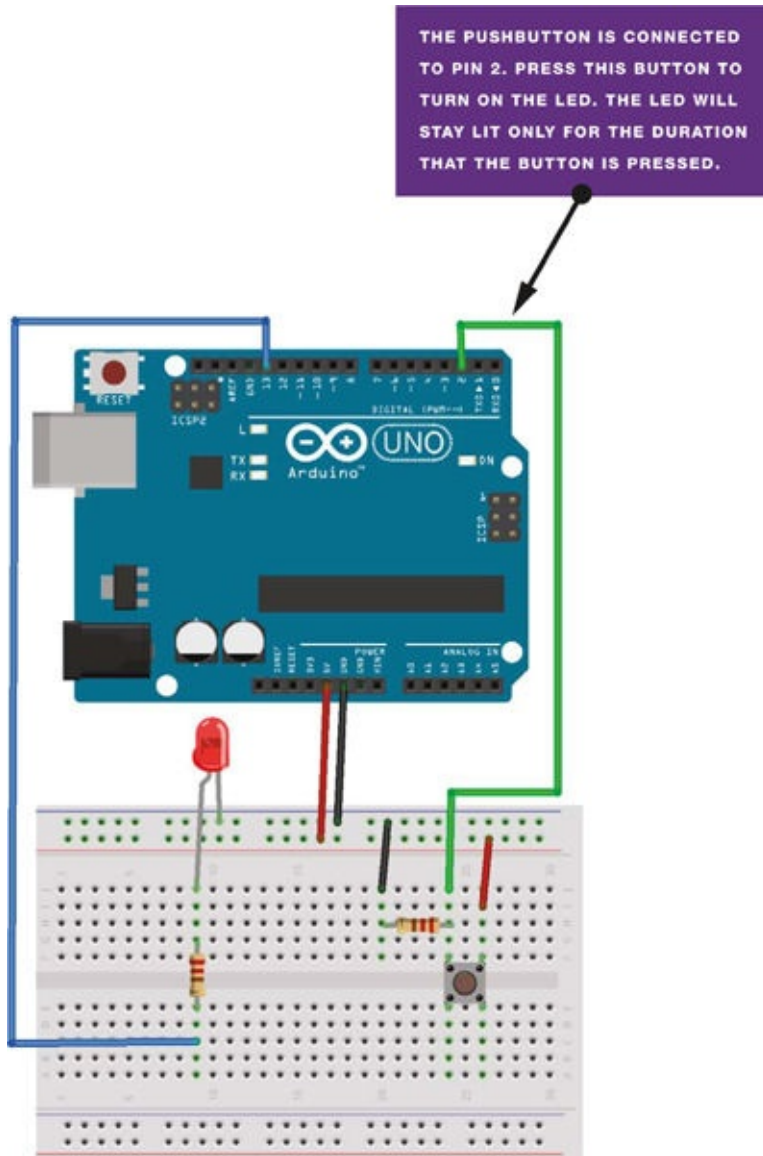
PUSHBUTTON	ARDUINO
Pin A	GND and pin 2 via 10k-ohm resistor
Pin B	+5V

- Add the LED to your breadboard, connecting the longer, positive leg to Arduino pin 13 via a 220-ohm resistor and the shorter leg to GND.

LED	ARDUINO
Positive leg	Pin 13 via 220-ohm resistor
Negative leg	GND

- Confirm that your setup matches the circuit diagram shown in [Figure 1-4](#), and then upload the code in “[The Sketch](#)” on [page 27](#).

4. **FIGURE 1-4:**
Circuit diagram for the pushbutton-controlled LED



THE SKETCH

In this sketch, you assign a pin for the pushbutton and set it as `INPUT`, and a pin for the LED and set it as `OUTPUT`. The code tells the Arduino to turn the LED on as long as the button is being pressed (completing the circuit), and to keep the LED off when the button is not being pressed. When the button is released, the circuit breaks and the LED will turn off again.

```
/* by DojoDave <http://www.0j0.org>
   modified 30 Aug 2011 by Tom Igoe
   This example code is in the public domain.
   http://www.arduino.cc/en/Tutorial/Button
*/

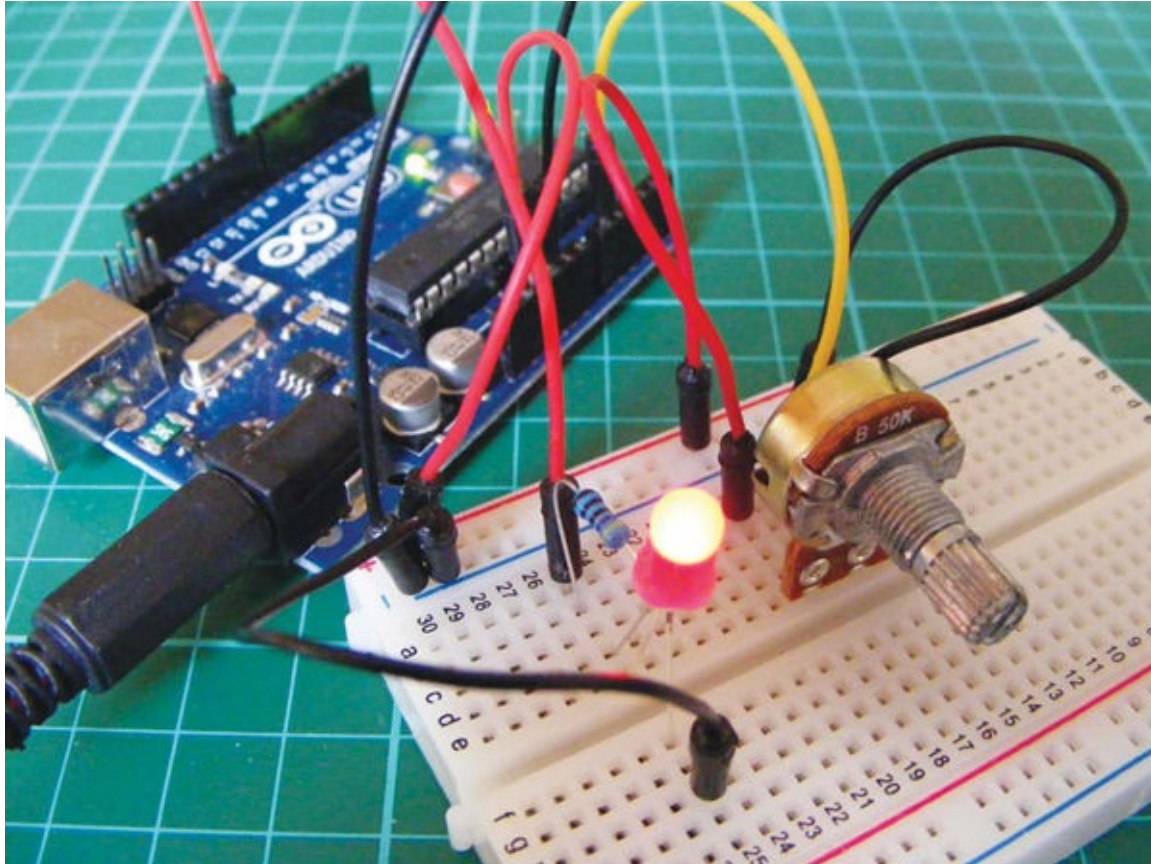
const int buttonPin = 2;    // Pin connected to pushbutton
const int ledPin = 13;     // Pin connected to LED
int buttonState = 0;       // Give pushbutton a value

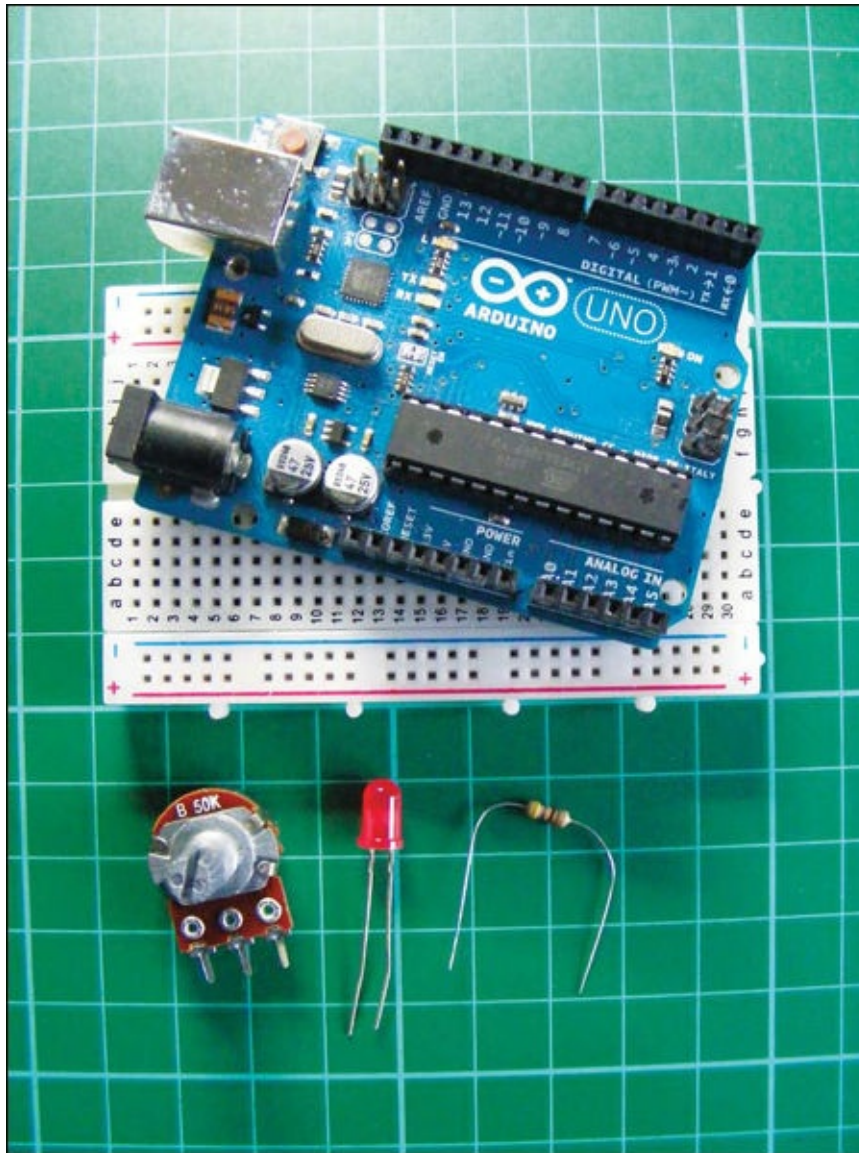
void setup() {
  pinMode(ledPin, OUTPUT); // Set LED pin as output
  pinMode(buttonPin, INPUT); // Set pushbutton pin as input
```

```
}  
  
void loop() {  
  buttonState = digitalRead(buttonPin); // Read input from pin 2  
  if (buttonState == HIGH) { // If pushbutton is pressed, set as HIGH  
    digitalWrite(ledPin, HIGH); // Turn on LED  
  }  
  else {  
    digitalWrite(ledPin, LOW); // Otherwise, turn off LED  
  }  
}
```

PROJECT 2: LIGHT DIMMER

IN THIS PROJECT, YOU'LL CREATE A DIMMER SWITCH BY ADDING A POTENTIOMETER TO CONTROL THE BRIGHTNESS OF AN LED.





PARTS REQUIRED

- Arduino board
- Breadboard
- Jumper wires
- LED
- 50k-ohm potentiometer
- 470-ohm resistor

A *potentiometer* is a variable resistor with a knob that allows you to alter the resistance of the potentiometer as you turn it. It is commonly used in electrical devices such as volume controls on audio equipment. This project uses a 50k-ohm potentiometer.

HOW IT WORKS

The potentiometer manipulates a continuous *analog* signal, which represents physical measurements. Humans perceive the world in analog; everything we see and hear is a continuous transmission of information to our senses. This continuous stream is what defines analog data. Digital information, on the other hand, estimates analog data using only numbers. To approximate the continuous analog data from the potentiometer, the Arduino must represent the signal as a series of discrete numbers—in this case, voltages. The center pin of the potentiometer sends the signal to an Arduino analog IN—any pin from A0 to A5—to read the value.

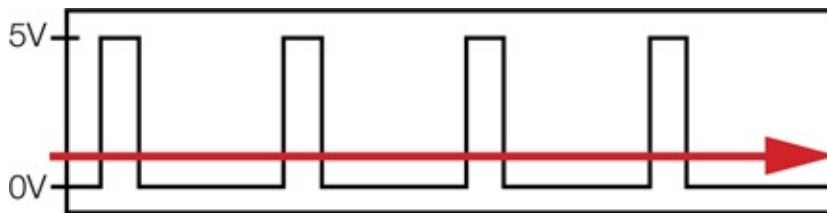
The LED is actually being switched on and off, but it happens so quickly that our eyes compensate and we see a continuously lit LED at varying light levels. This is known as *persistence of vision*.

To create persistence of vision, the Arduino uses a technique called *pulse width modulation (PWM)*. The Arduino creates a pulse by switching the power on and off very quickly. The duration that the power is on or off (known as the *pulse width*) in the cycle determines the average output, and by varying this pulse width the pattern can simulate voltages between full on (5 volts) and off (0 volts). If the signal from the Arduino is on for half the time and off for half, the average output will be 2.5 volts, halfway between 0 and 5. If the signal is on for 80 percent and off for 20 percent, then the average voltage is 4 volts, and so on. You can vary the signal, which in turn varies the pulse width, by turning the potentiometer left or right, increasing or decreasing the resistance.

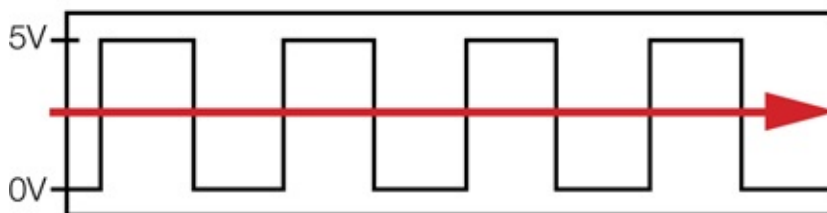
Using this technique, you can change the voltage sent to the LED and make it dimmer or brighter to match the analog signal from the potentiometer. Only pins 3, 5, 6, 9, 10, or 11 on the Arduino can use PWM. [Figure 2-1](#) gives examples of how PWM would look as a waveform.

FIGURE 2-1:

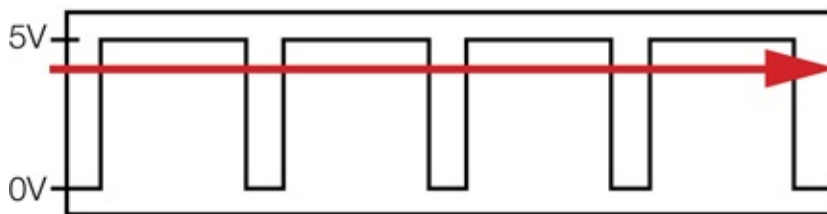
Pulse width modulation as a waveform



High signal (at 5 volts) for 20% of the time and low (at 0 volts) for 80% of the time, so output (red line) will be $0.2 \times 5V = 1V$.



High signal for 50% and low for 50%, so output will be $0.5 \times 5V = 2.5V$.

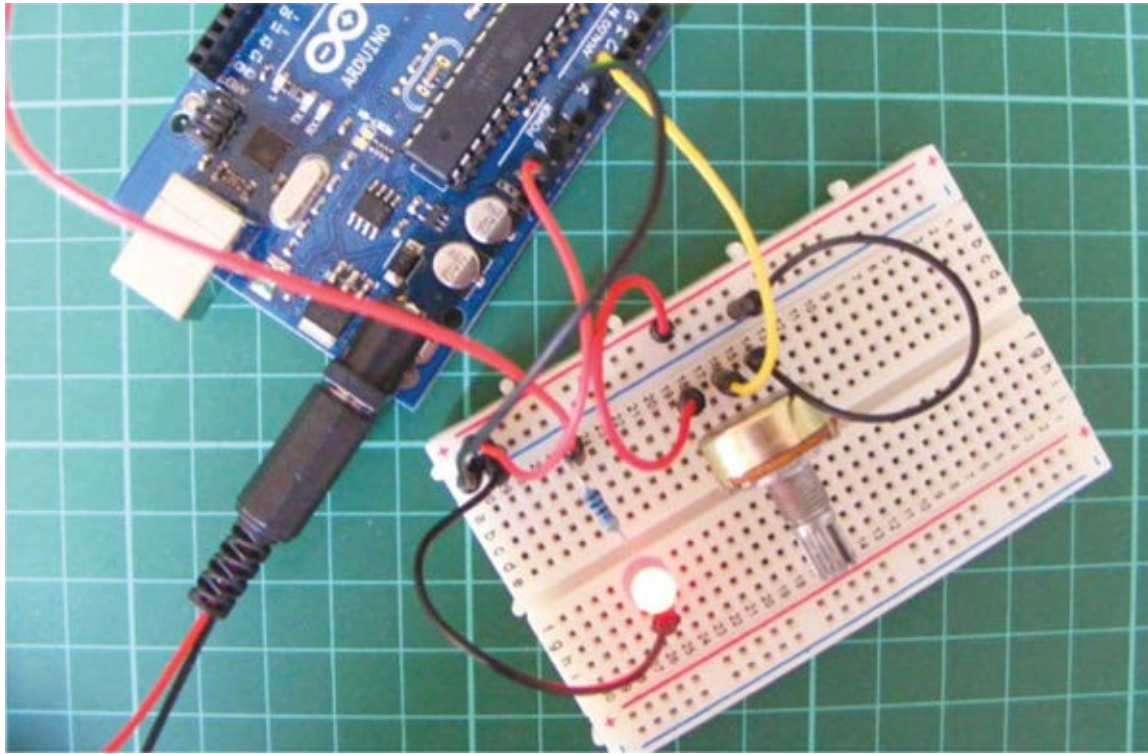


High signal for 80% and low for 20%, so output will be $0.8 \times 5V = 4V$.

THE BUILD

1. Insert the potentiometer into your breadboard and connect the center pin to the Arduino's A0 pin. Connect one of the outer pins to the +5V rail of the breadboard and the other outer pin to GND on the breadboard (it doesn't actually matter which way around the outer potentiometer pins are connected; these instructions just reflect the diagrams in this project), as shown in [Figure 2-2](#).

1. **FIGURE 2-2:**
Connecting the potentiometer to the Arduino

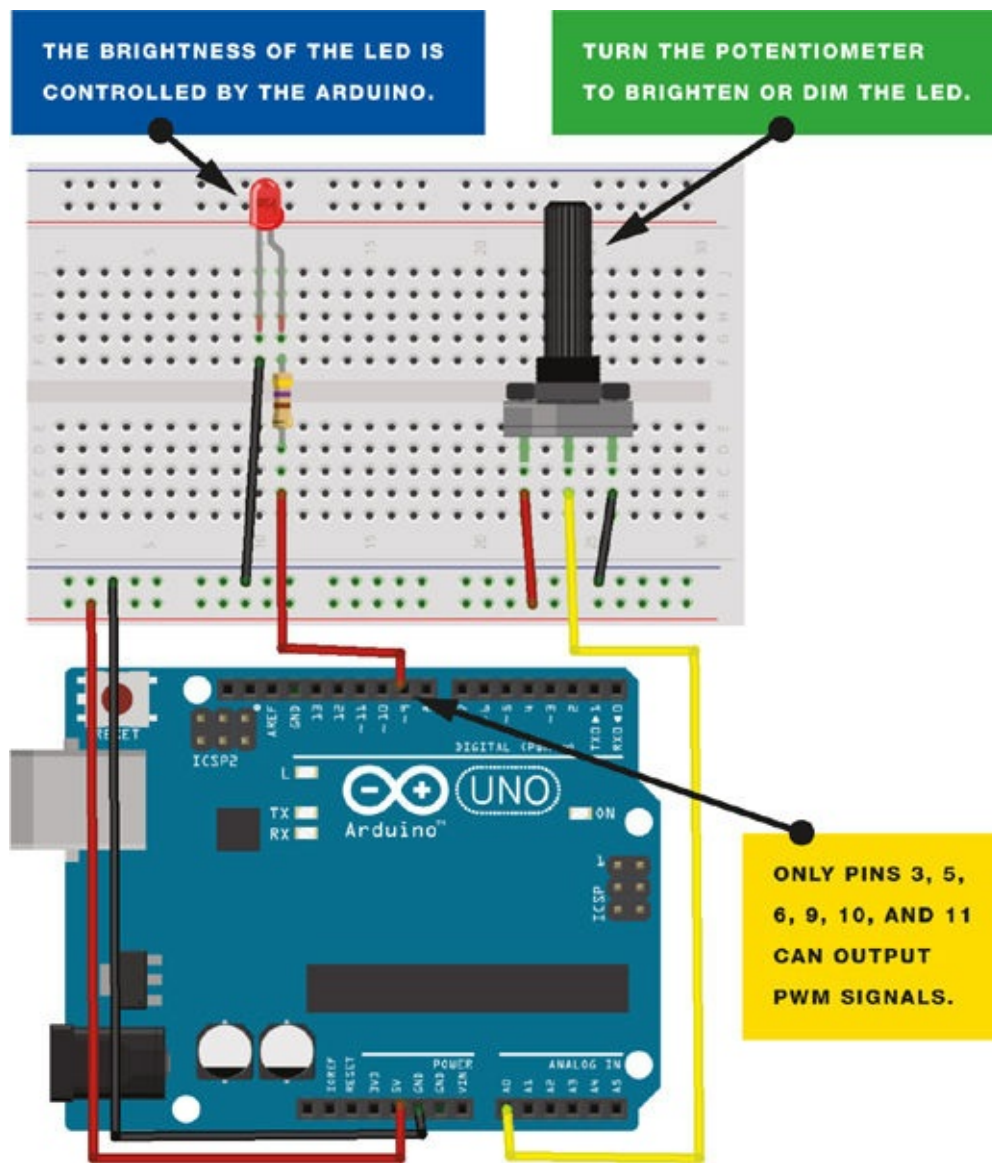


POTENTIOMETER	ARDUINO
Left pin	+5V
Center pin	A0
Right pin	GND

2. Insert the LED into the breadboard. Attach the positive leg (the longer leg) to pin 9 of the Arduino via the 470-ohm resistor, and the negative leg to GND, as shown in [Figure 2-3](#).

LED	ARDUINO
Positive leg	Pin 9
Negative leg	GND via 470-ohm resistor

2. **FIGURE 2-3:**
Circuit diagram for the light dimmer



3. Upload the code in “[The Sketch](#)” below.
4. Turn the potentiometer to control the brightness of the LED.

This project has many potential uses: you can cluster a number of LEDs together to create an adjustable flashlight, a night-light, a display case light, or anything else that uses dimming lights.

THE SKETCH

This sketch works by setting pin A0 as your potentiometer and pin 9 as an OUTPUT to power the LED. You then run a loop that continually reads the value from the potentiometer and sends that value as voltage to the LED. The voltage value is between 0–5 volts, and the brightness of the LED will vary accordingly.

```
/* http://arduino.cc/en/Reference/AnalogWrite by Tom Igoe
   from http://itp.nyu.edu/physcomp/Labs/AnalogIn */

int potPin = A0; // Analog input pin connected to the potentiometer
```



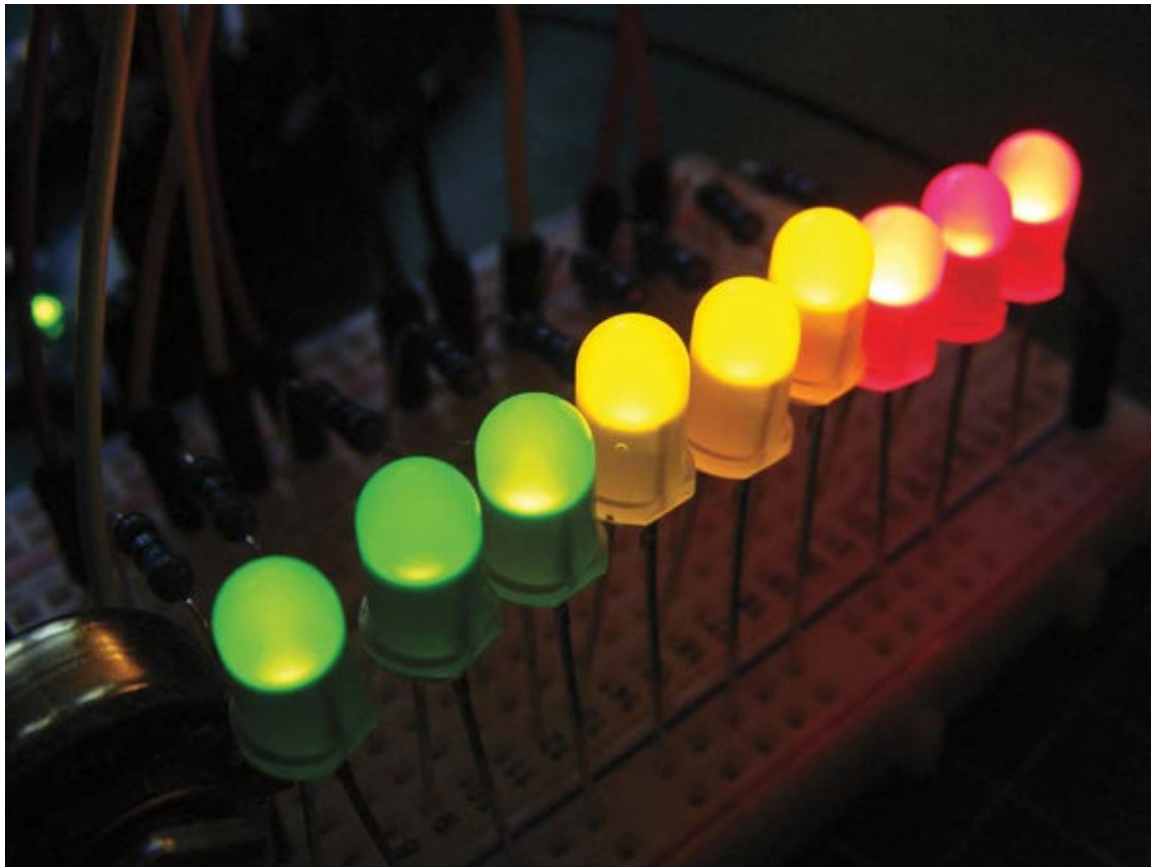
```
int potValue = 0; // Value that will be read from the potentiometer
int led = 9; // Pin 9 (connected to the LED) is capable of PWM

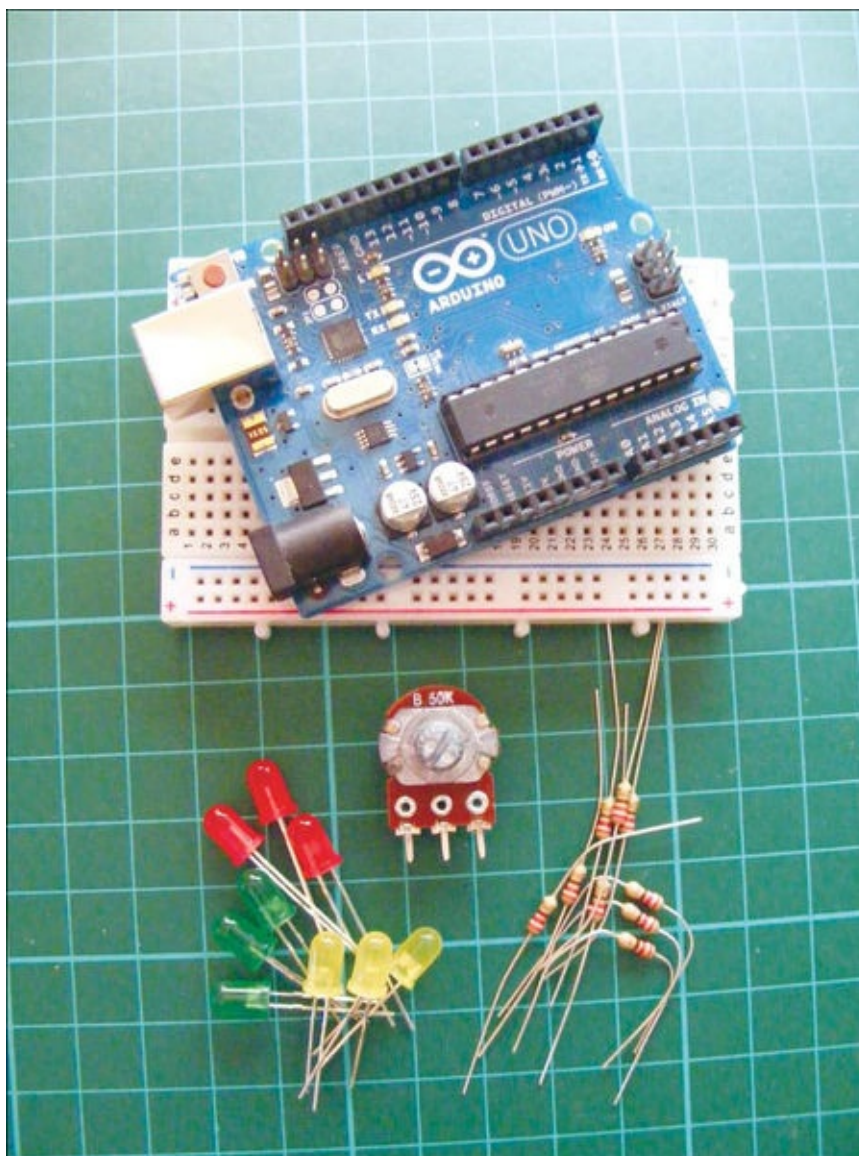
// Runs once at beginning of the program
void setup() {
  pinMode(led, OUTPUT); // Set pin 9 to output
}

// Loops continuously
void loop() {
  potValue = analogRead(potPin); // Read potentiometer value
                                  // from A0 pin
  analogWrite(led, potValue/4); // Send potentiometer value to LED
                                 // to control brightness with PWM
  delay(10); // Wait for 10 ms
}
```

PROJECT 3: BAR GRAPH

IN THIS PROJECT, YOU'LL COMBINE WHAT YOU'VE LEARNED IN THE PREVIOUS LED PROJECTS TO CREATE AN LED BAR GRAPH THAT YOU CAN CONTROL WITH A POTENTIOMETER.





PARTS REQUIRED

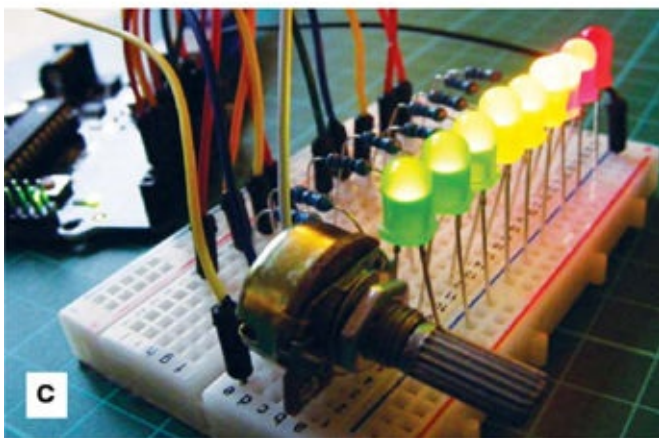
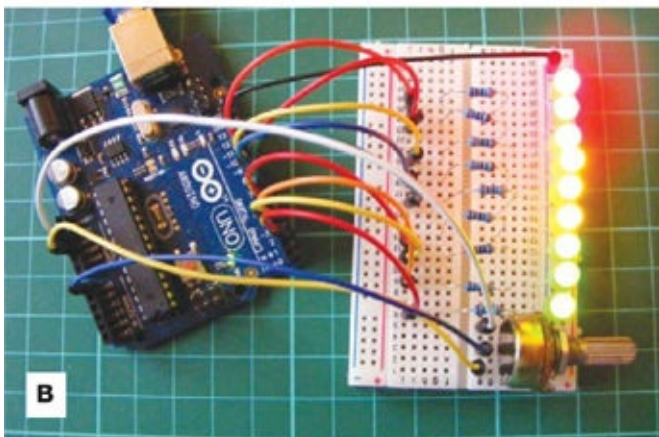
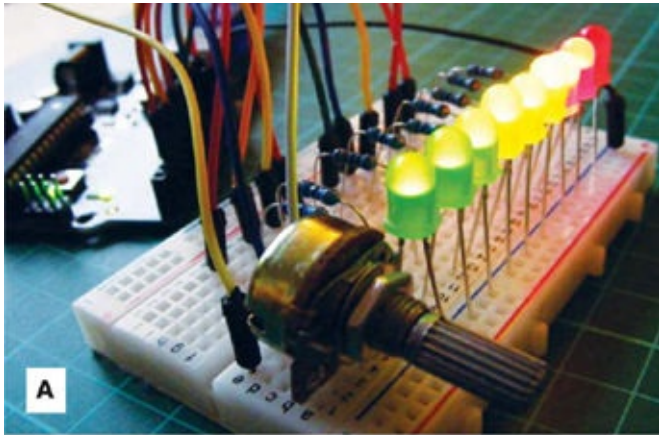
- Arduino board
- Breadboard
- Jumper wires
- 9 LEDs
- 50k-ohm potentiometer
- 9 220-ohm resistors

HOW IT WORKS

A bar graph is a series of LEDs in a line, similar to what you might see on an audio display. It's made up of a row of LEDs with an analog input, like a potentiometer or microphone. In this project, you use the analog signal from the potentiometer to control which LEDs are lit. When you turn the potentiometer one way, the LEDs light up one at a time in sequence, as shown in

Figure 3-1(a), until they are all on, shown in Figure 3-1(b). When you turn it the other way, they turn off in sequence, as shown in Figure 3-1(c).

FIGURE 3-1:
The LEDs light up and turn off in sequence as you turn the potentiometer.



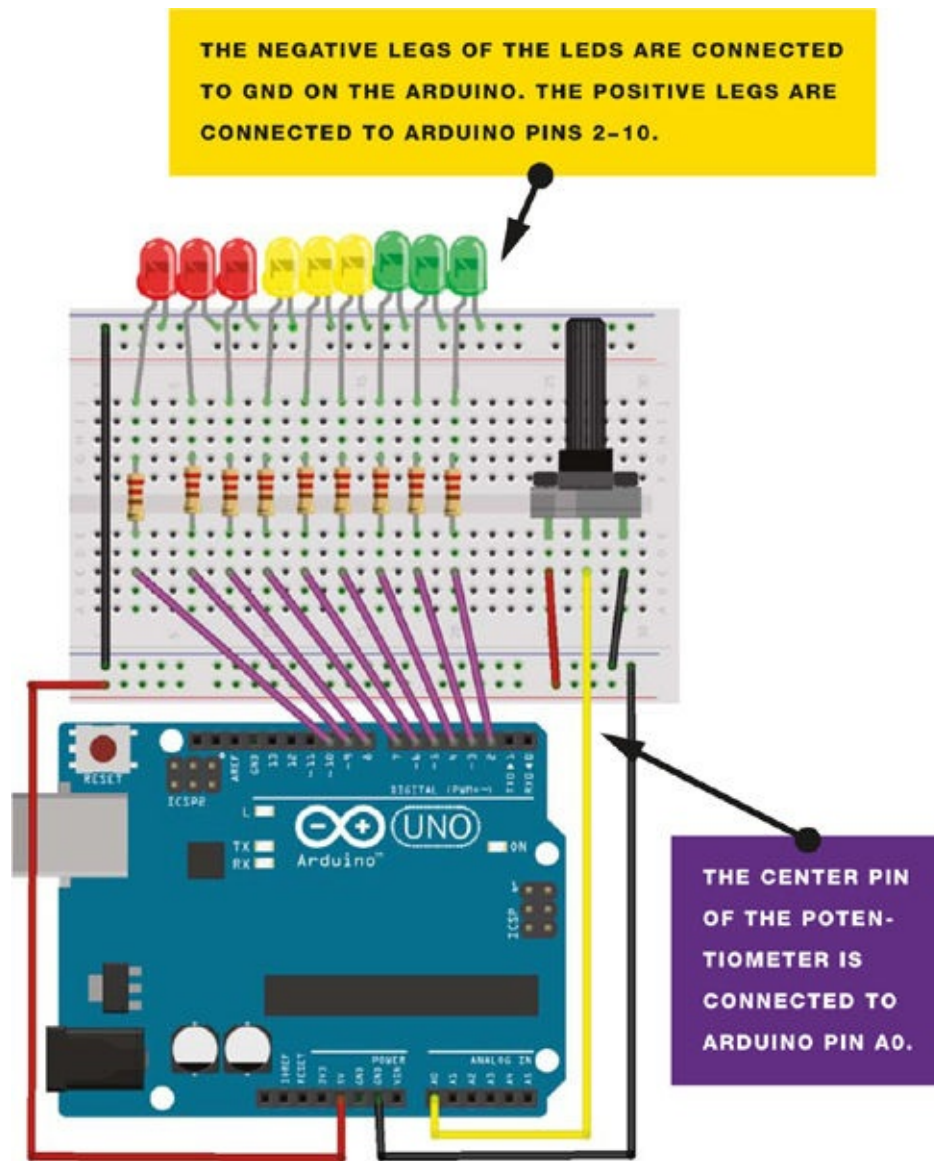
THE BUILD

1. Insert the LEDs into the breadboard with their shorter, negative legs in the GND rail. Connect this rail to Arduino GND using a jumper wire.
2. Insert a 220-ohm resistor for each LED into the breadboard, with one resistor leg connected to the positive LED leg. Connect the other legs of the resistors to digital pins 2–10 in sequence, as shown in [Figure 3-2](#). It's important that the resistors bridge the break in the breadboard as shown.

LEDS	ARDUINO

Positive legs	Pins 2–10 via 220-ohm resistor
Negative legs	GND

2. **FIGURE 3-2:**
Circuit diagram for the bar graph



NOTE

As mentioned in [Project 2](#), it doesn't actually matter which way the outer potentiometer pins are connected, but I've given instructions here to reflect the images.

3. Place the potentiometer in the breadboard and connect the center pin to Arduino A0. Connect the right outer pin to +5V and the left potentiometer pin to GND.

POTENTIOMETER	ARDUINO
Left pin	GND
Center pin	A0
Right pin	+5V

4. Upload the code in [“The Sketch”](#) below.

THE SKETCH

The sketch first reads the input from the potentiometer. It maps the input value to the output range, in this case nine LEDs. Then it sets up a `for` loop over the outputs. If the output number of the LED in the series is lower than the mapped input range, the LED turns on; if not, it turns off. See? Simple! If you turn the potentiometer to the right, the LEDs light up in sequence. Turn it to the left, and they turn off in sequence.

```
/* By Tom Igoe. This example code is in the public domain.
   http://www.arduino.cc/en/Tutorial/BarGraph */

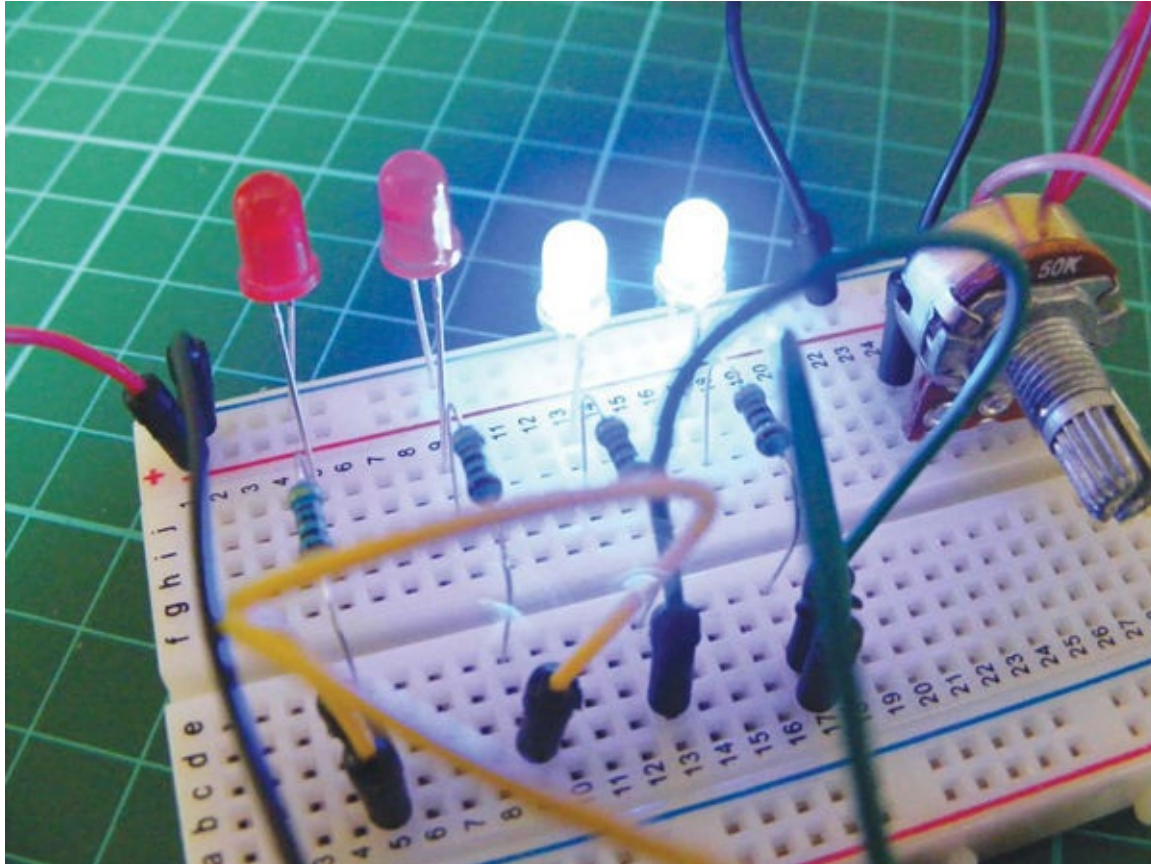
const int analogPin = A0; // Pin connected to the potentiometer
const int ledCount = 9;   // Number of LEDs
int ledPins[] = {2,3,4,5,6,7,8,9,10}; // Pins connected to the LEDs

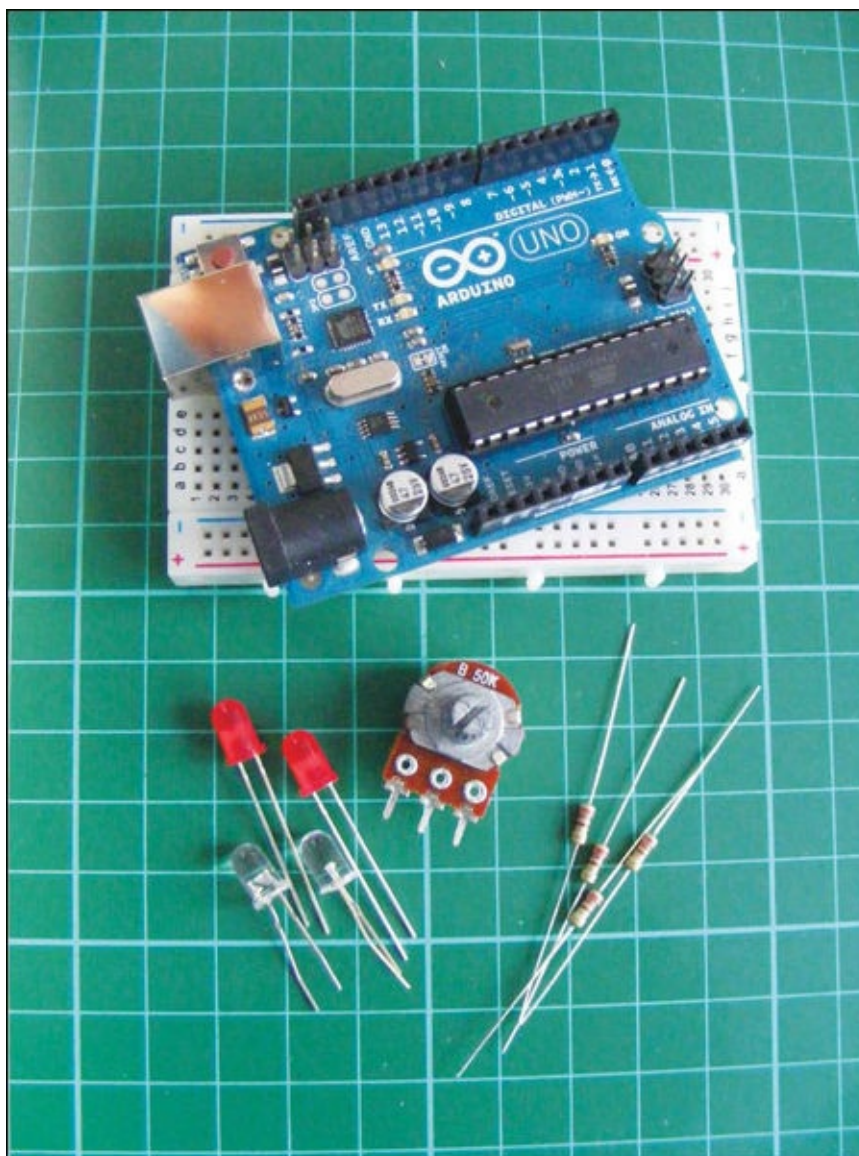
void setup() {
  for (int thisLed = 0; thisLed < ledCount; thisLed++) {
    pinMode(ledPins[thisLed], OUTPUT); // Set the LED pins as output
  }
}

// Start a loop
void loop() {
  int sensorReading = analogRead(analogPin); // Analog input
  int ledLevel = map(sensorReading, 0, 1023, 0, ledCount);
  for (int thisLed = 0; thisLed < ledCount; thisLed++) {
    if (thisLed < ledLevel) { // Turn on LEDs in sequence
      digitalWrite(ledPins[thisLed], HIGH);
    }
    else { // Turn off LEDs in sequence
      digitalWrite(ledPins[thisLed], LOW);
    }
  }
}
```

PROJECT 4: DISCO STROBE LIGHT

IN THIS PROJECT, YOU'LL APPLY THE SKILLS YOU LEARNED IN PROJECT 3 TO MAKE A STROBE LIGHT WITH ADJUSTABLE SPEED SETTINGS.





PARTS REQUIRED

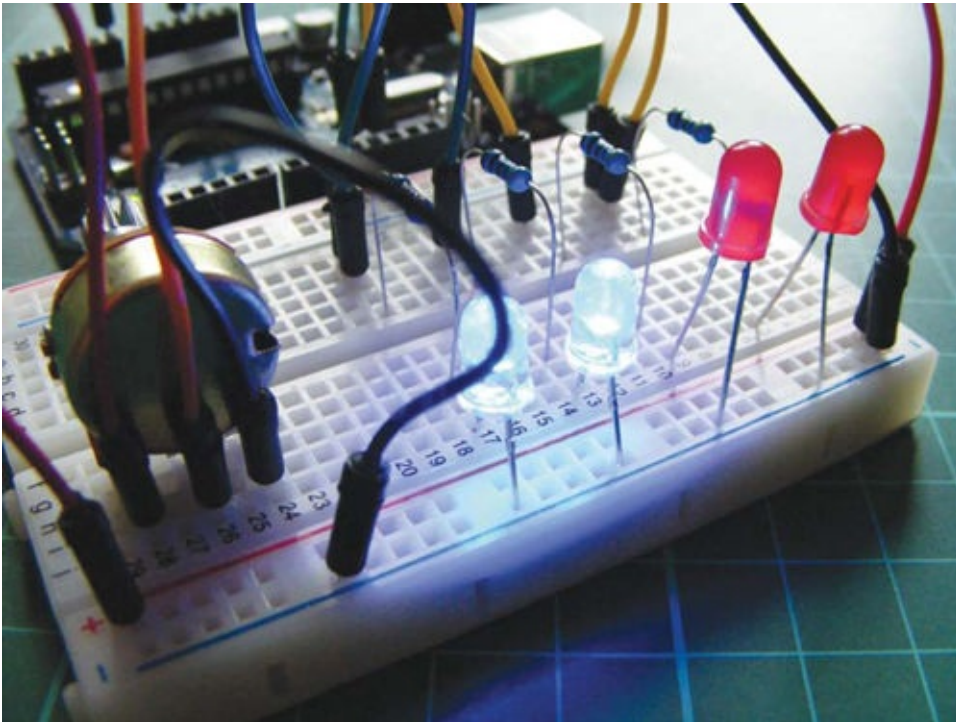
- Arduino board
- Breadboard
- Jumper wires
- 2 blue LEDs
- 2 red LEDs
- 50k-ohm potentiometer
- 4 220-ohm resistors

HOW IT WORKS

Turning the potentiometer up or down changes the speed of the flashing lights, creating a strobe effect. You can use red and blue LEDs for a flashing police light effect (see [Figure 4-1](#)). Connect the LEDs of the same color to the same Arduino pin so they'll always light together. If you build a

casing to house your LEDs, you'll have your own mobile strobe unit. You can add up to 10 LEDs; just update the sketch to include your output pins and the new number of LEDs.

FIGURE 4-1:
Red and blue LEDs mimic the lights of a police car.



THE BUILD

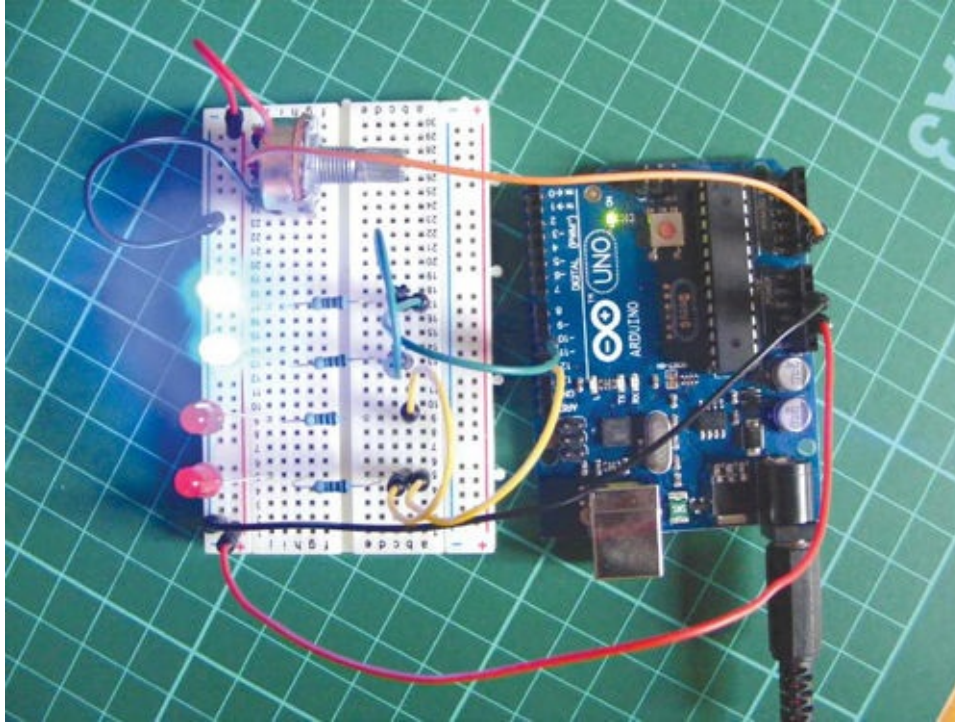
1. Place your LEDs into the breadboard with the short, negative legs in the GND rail, and then connect this rail to Arduino GND.

NOTE

Remember to add power to the breadboard.

2. Insert the resistors into the board, connecting them to the longer, positive legs of the LEDs. Use jumper wires to connect the two red LEDs together and the two blue LEDs together via the resistors, as shown in [Figure 4-2](#); this allows the LEDs of the same color to be controlled by a single pin.

2. **FIGURE 4-2:**
Connecting LEDs with jumper wires



3. Connect the red LEDs to Arduino pin 12 and the blue LEDs to Arduino pin 11.

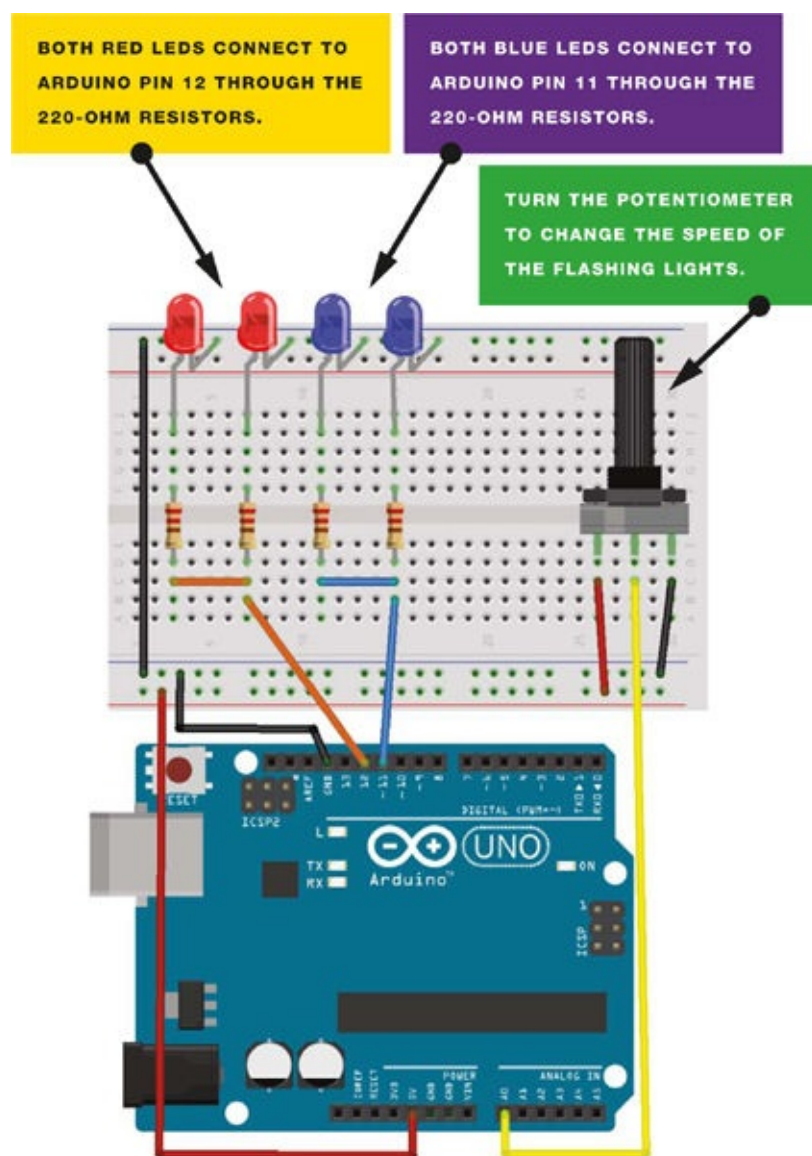
LEDS	ARDUINO
Negative legs	GND
Positive leg (red)	Pin 12
Positive leg (blue)	Pin 11

4. Place the potentiometer in the breadboard and connect the center pin to Arduino A0, the left pin to GND, and the right pin to +5V.

POTENTIOMETER	ARDUINO
Left pin	GND
Center pin	A0
Right pin	+5V

5. Confirm that your setup matches that of [Figure 4-3](#), and then upload the code in “[The Sketch](#)” on [page 43](#).

5. **FIGURE 4-3:**
Circuit diagram for the disco strobe light



THE SKETCH

The sketch works by setting the analog signal from the potentiometer to the Arduino as an input and the pins connected to the LEDs as outputs. The Arduino reads the analog input from the potentiometer and uses this value as the *delay value*—the amount of time that passes before the LEDs change state (either on or off). This means that the LEDs are on and off for the duration of the potentiometer value, so changing this value alters the speed of the flashing. The sketch cycles through the LEDs to produce a strobe effect.

```
const int analogInPin = A0; // Analog input pin connected to the
                             // potentiometer
int sensorValue = 0;        // Value read from the potentiometer
int timer = 0;              // Delay value

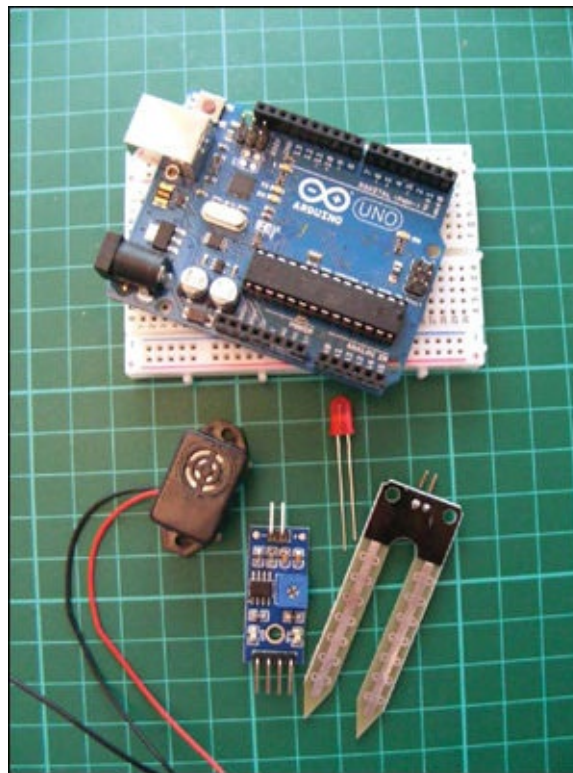
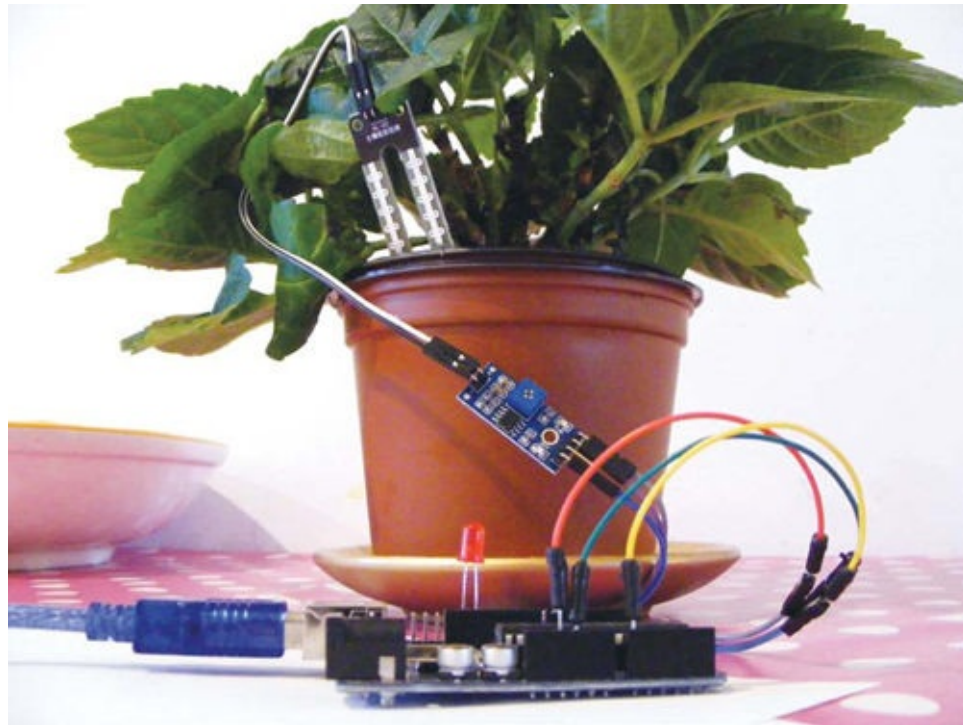
// Set digital pins 12 and 11 as outputs
void setup() {
  pinMode(12, OUTPUT);
  pinMode(11, OUTPUT);
}
```

```
// Start a loop to turn LEDs on and off with a delay in between
void loop() {
  sensorValue = analogRead(analogInPin); // Read value from the
                                          // potentiometer
  timer = map(sensorValue, 0, 1023, 10, 500); // Delay 10 to 500 ms
  digitalWrite(12, HIGH); // LED turns on
  delay(timer);           // Delay depending on potentiometer value
  digitalWrite(12, LOW);  // LED turns off
  delay(timer);
  digitalWrite(12, HIGH);
  delay(timer);
  digitalWrite(12, LOW);
  digitalWrite(11, HIGH);
  delay(timer);
  digitalWrite(11, LOW);
  delay(timer);
  digitalWrite(11, HIGH);
  delay(timer);
  digitalWrite(11, LOW);
}

```

PROJECT 5: PLANT MONITOR

IN THIS PROJECT I'LL INTRODUCE A NEW TYPE OF ANALOG SENSOR THAT DETECTS MOISTURE LEVELS. YOU'LL SET UP A LIGHT AND SOUND ALARM SYSTEM TO TELL YOU WHEN YOUR PLANT NEEDS WATERING.



PARTS REQUIRED

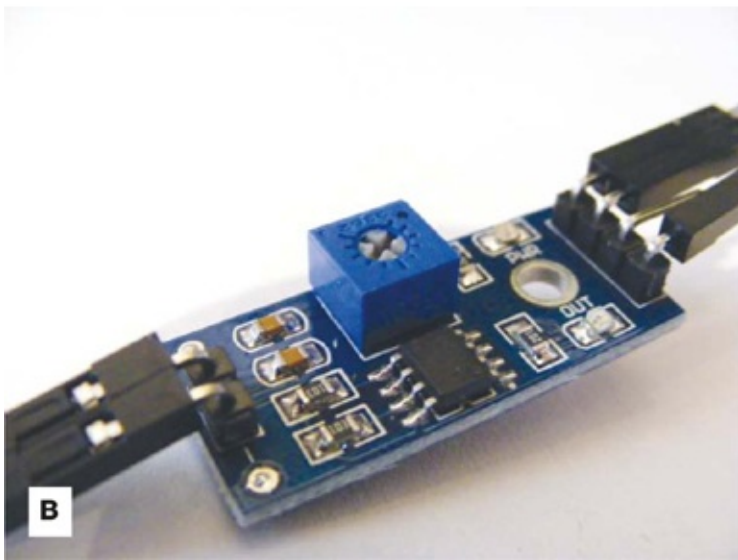
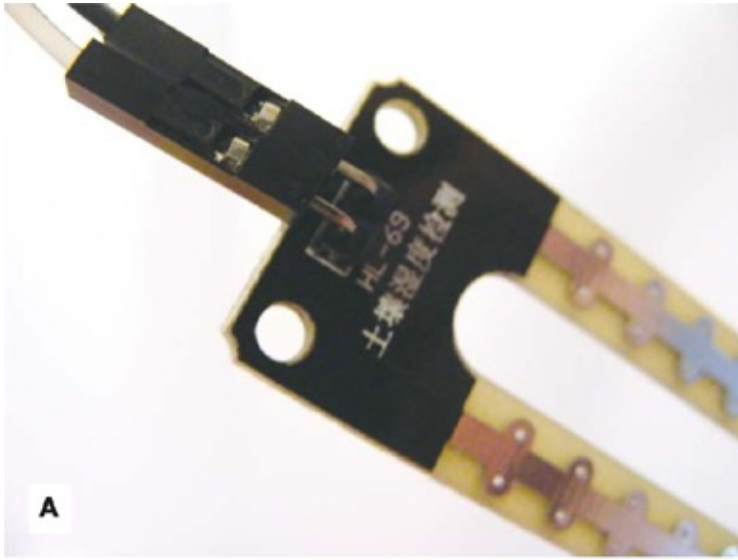
- Arduino board
- Jumper wires
- LED
- HL-69 hygrometer soil moisture sensor
- Piezo buzzer

HOW IT WORKS

You'll use an HL-69 moisture sensor, readily available online for a few dollars or from some of the retailers listed in [Appendix A](#). The prongs of the sensor detect the moisture level in the surrounding soil by passing current through the soil and measuring the resistance. Damp soil conducts electricity easily, so it provides lower resistance, while dry soil conducts poorly and has a higher resistance.

The sensor consists of two parts, as shown in [Figure 5-1](#): the actual prong sensor (a) and the controller (b). The two pins on the sensor need to connect to the two separate pins on the controller (connecting wires are usually supplied). The other side of the controller has four pins, three of which connect to the Arduino.

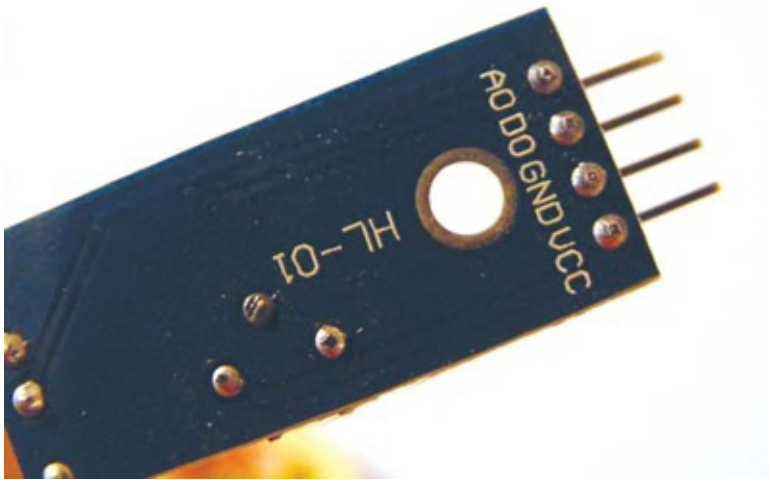
FIGURE 5-1:
The HL-69 moisture sensor prong (a) and controller (b)



The four pins are, from left to right, AO (analog out), DO (digital out), GND, and VCC (see [Figure 5-2](#)). You can read the values from the controller through the IDE when it's connected to your computer. This project doesn't use a breadboard, so the connections are all made directly to the Arduino.

FIGURE 5-2:

The pins are labeled on the underside of the module

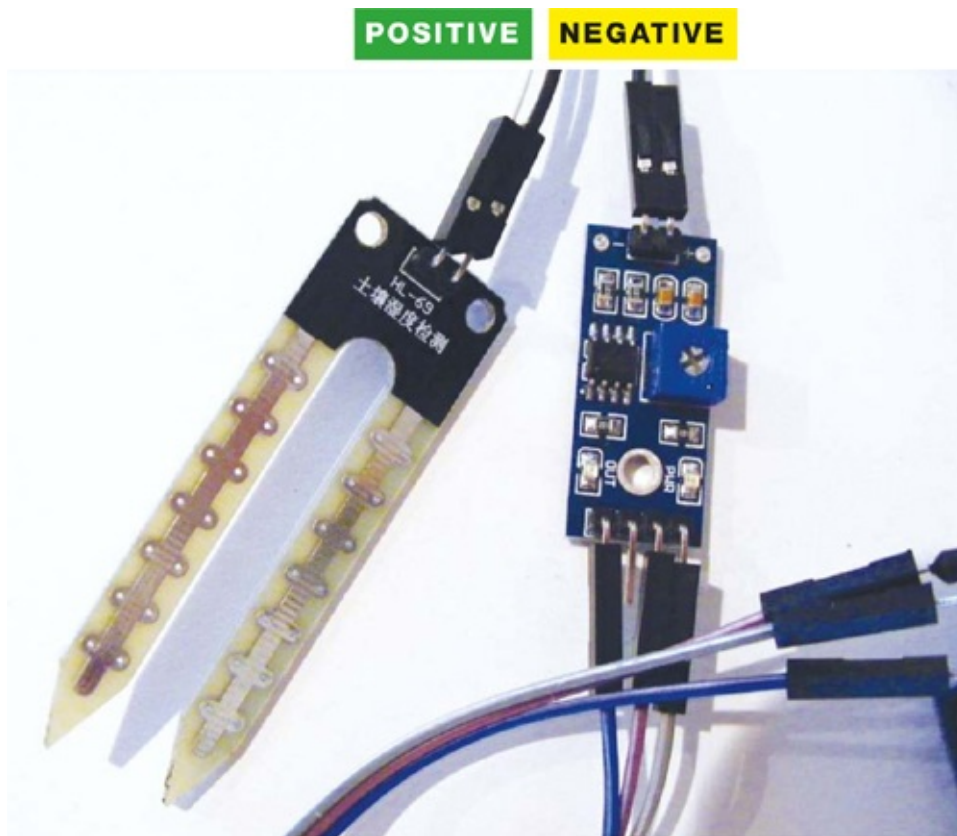


Lower readings indicate that more moisture is being detected, and higher readings indicate dryness. If your reading is above 900, your plant is seriously thirsty. If your plant gets too thirsty, the LED will light and the piezo buzzer will sound. *Piezos* are inexpensive buzzers and are explained more in [Project 7](#).

THE BUILD

1. Connect the sensor's two pins to the + and – pins on the controller using the provided connecting wires, as shown in [Figure 5-3](#).

1. **FIGURE 5-3:**
Connecting the sensor to the controller

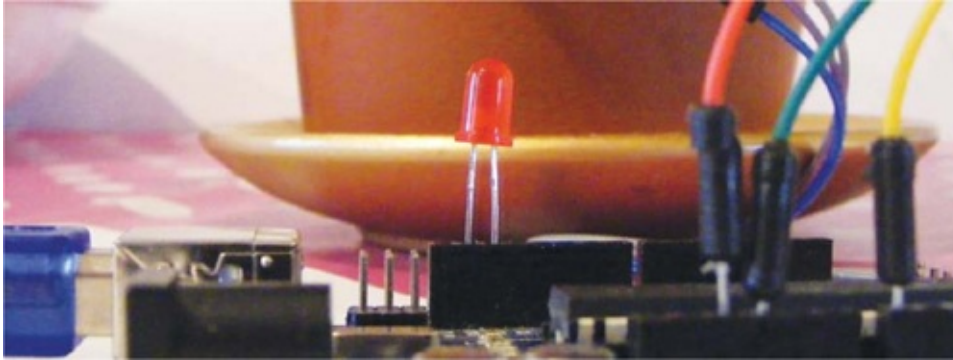


2. Connect the three prongs from the controller to +5V, GND, and Arduino A0 directly on the Arduino, as shown in the following table. The DO pin is not used.

SENSOR CONTROLLER	ARDUINO
VCC	+5V
GND	GND
A0	A0
DO	Not used

3. Connect an LED directly to the Arduino with the shorter, negative leg in GND and the longer, positive leg in Arduino pin 13, as shown in [Figure 5-4](#).

3. **FIGURE 5-4:**
Connecting the LED to the Arduino



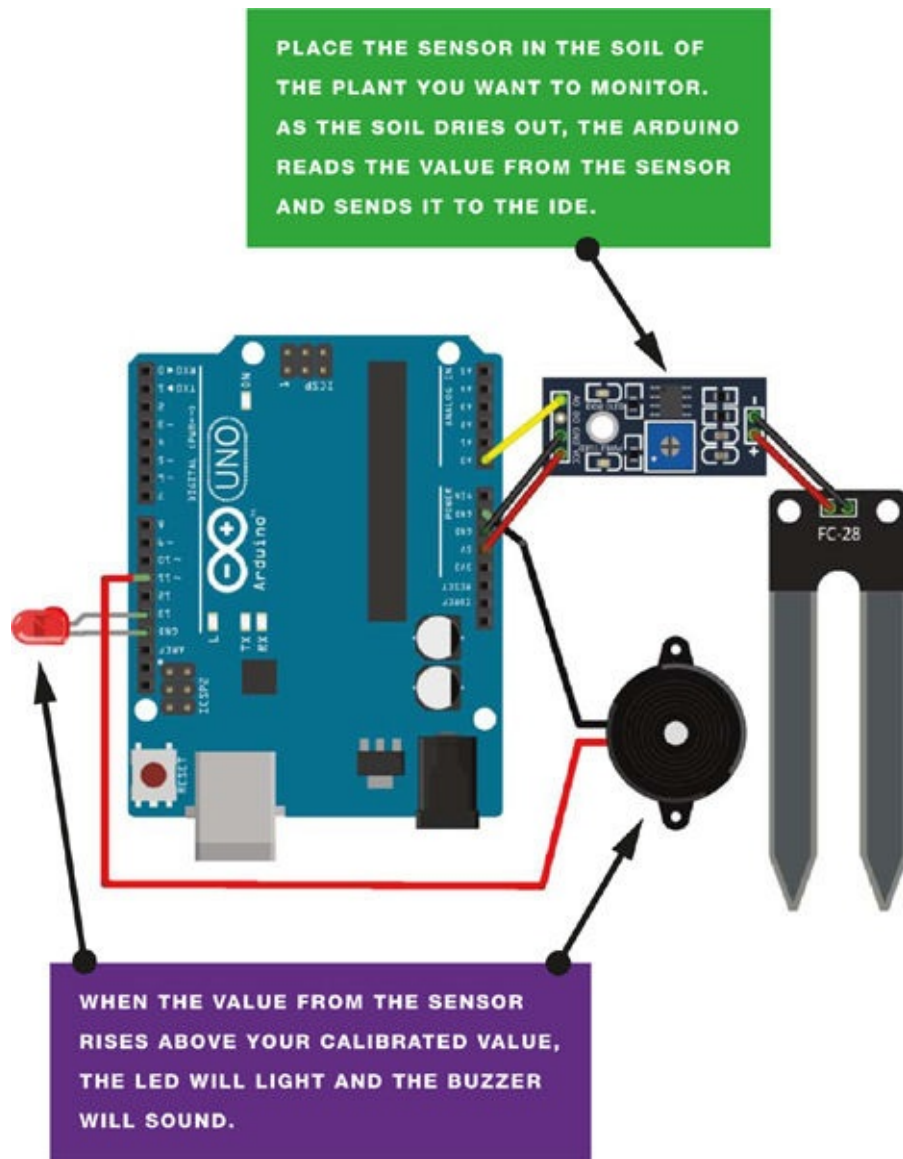
LED	ARDUINO
Positive leg	Pin 13
Negative leg	GND

4. Connect the piezo buzzer's black wire to GND and its red wire to Arduino pin 11.

PIEZO	ARDUINO
Red wire	Pin 11
Black wire	GND

5. Check that your setup matches that of [Figure 5-5](#), and then upload the code in “[The Sketch](#)” on [page 51](#).

5. **FIGURE 5-5:**
Circuit diagram for the plant monitor

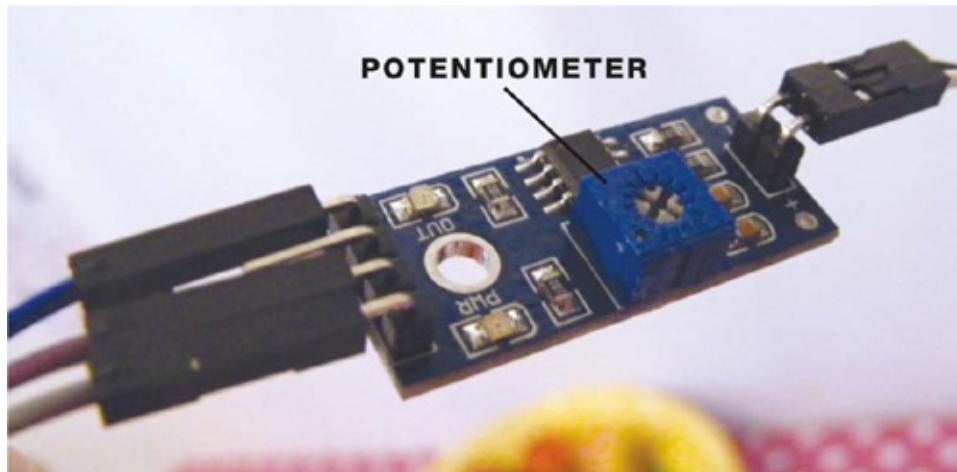


6. Connect the Arduino to your computer using the USB cable. Open the Serial Monitor in your IDE to see the values from the sensor—this will also help you to calibrate your plant monitor. The IDE will display the value of the sensor’s reading. My value was 1000 with the sensor dry and not inserted in the soil, so I know this is the highest, and driest, value. To calibrate this value, turn the potentiometer on the controller clockwise to increase the resistance and counterclockwise to decrease it (see [Figure 5-5](#)).

When the sensor is inserted into moist soil, the value will drop to about 400. As the soil dries out, the sensor value rises; when it reaches 900, the LED will light and the buzzer will sound.

6. FIGURE 5-6:

Turn the potentiometer to calibrate your plant monitor.



THE SKETCH

The sketch first defines Arduino pin A0 so that it reads the moisture sensor value. It then defines Arduino pin 11 as output for the buzzer, and pin 13 as output for the LED. Use the `Serial.println()` function to send the reading from the sensor to the IDE, in order to see the value on the screen.

Change the value in the line

```
if(analogRead(0) > 900){
```

depending on the reading from the sensor when it is dry (here it's 900). When the soil is moist, this value will be below 900, so the LED and buzzer will remain off. When the value rises above 900, it means the soil is drying out, and the buzzer and LED will alert you to water your plant.

```
const int moistureAO = 0;
int AO = 0;          // Pin connected to A0 on the controller
int tmp = 0;        // Value of the analog pin
int buzzPin = 11;   // Pin connected to the piezo buzzer
int LED = 13;       // Pin connected to the LED

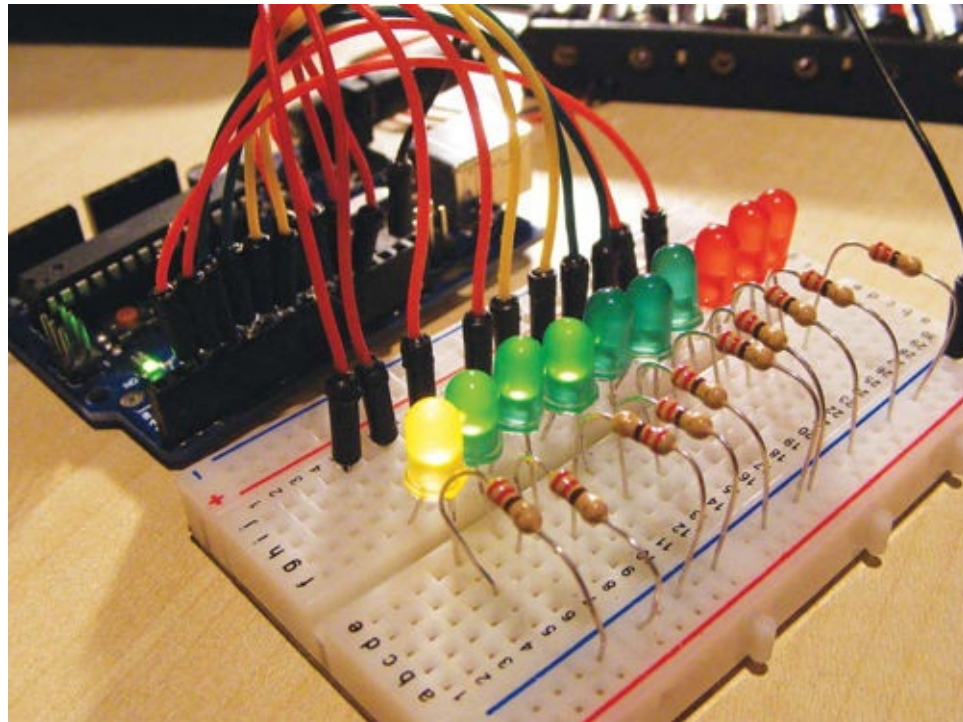
void setup () {
  Serial.begin(9600); // Send Arduino reading to IDE
  Serial.println("Soil moisture sensor");
  pinMode(moistureAO, INPUT);
  pinMode(buzzPin, OUTPUT); // Set pin as output
  pinMode(LED, OUTPUT);    // Set pin as output
}

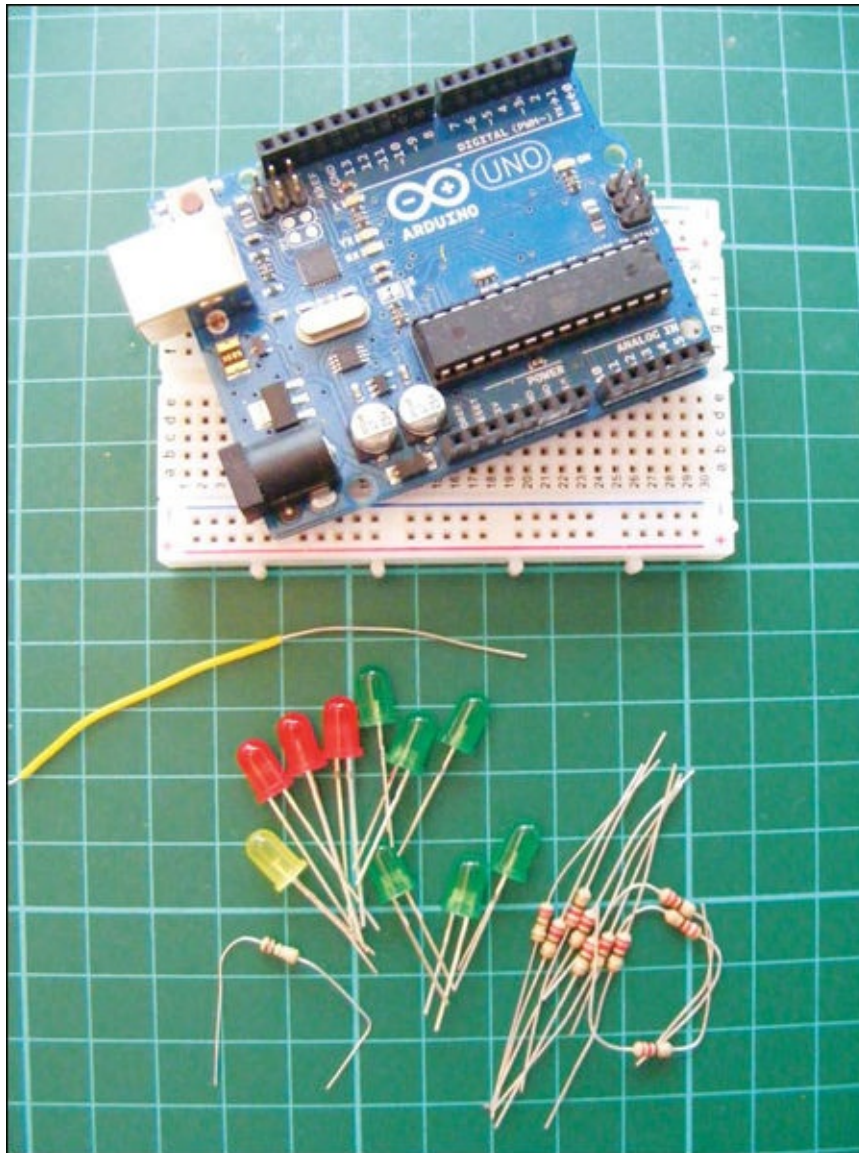
void loop () {
  tmp = analogRead( moistureAO );
  if ( tmp != AO ) {
    AO = tmp;
    Serial.print("A = "); // Show the resistance value of the sensor
                        // in the IDE
    Serial.println(AO);
  }
  delay (1000);
  if (analogRead(0) > 900) { // If the reading is higher than 900,
```

```
digitalWrite(buzzPin, HIGH); // the buzzer will sound
digitalWrite(LED, HIGH); // and the LED will light
delay(1000); // Wait for 1 second
digitalWrite(buzzPin, LOW);
digitalWrite(LED, HIGH);
}
else {
digitalWrite(buzzPin, LOW); // If the reading is below 900,
// the buzzer and LED stay off
digitalWrite(LED, LOW);
}
}
```

PROJECT 6: GHOST DETECTOR

WHO WOULDN'T WANT TO MAKE A GHOST DETECTOR? THIS IS A REALLY SIMPLE PROJECT THAT DOESN'T TAKE LONG TO PUT TOGETHER, SO YOU CAN START DETECTING GHOSTS RIGHT AWAY.





PARTS REQUIRED

- Arduino board
- Breadboard
- Jumper wires
- 3 red LEDs
- 1 yellow LED
- 6 green LEDs
- 10 220-ohm resistors
- 20-centimeter length of single-core wire
- 1M-ohm resistor

HOW IT WORKS

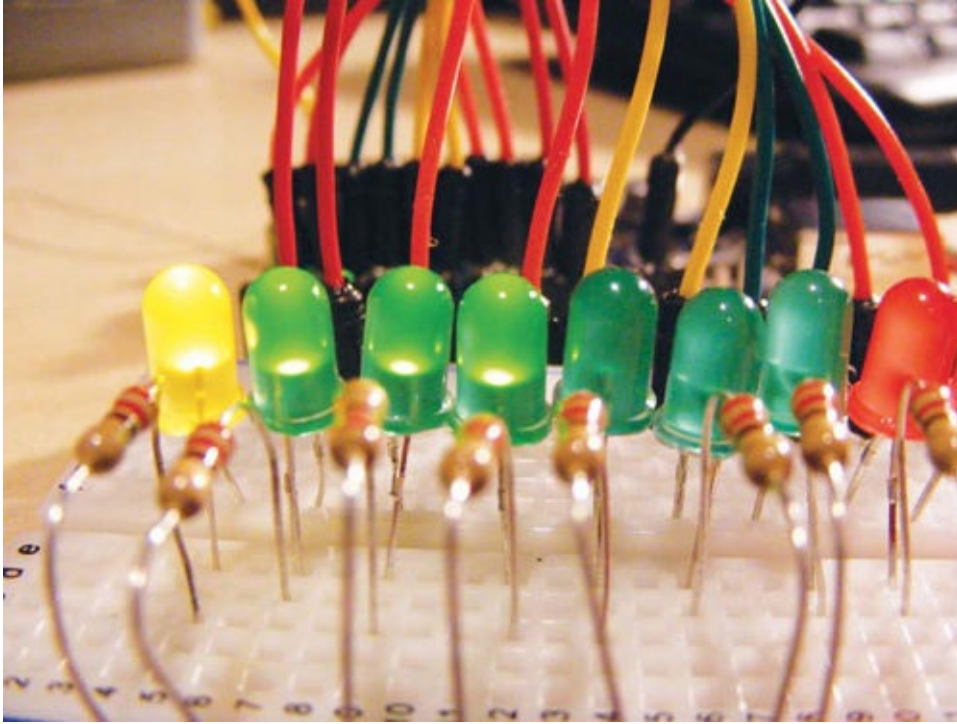
Okay, so I might be stretching things a bit by calling this project a ghost detector. This project actually detects *electromagnetic fields*, but many people believe this is how to tell if there are ghosts or spirits around.

In this project, you'll set up a ghost-detecting antenna and LED bar graph system to tell whether there is a high level of electromagnetic activity in the vicinity. A length of bare wire acts as an antenna to pick up an electromagnetic field within a radius of two meters. Depending on the strength of the signal, the LEDs will light in sequence: the stronger the signal, the more LEDs will light. Power up the Arduino, and point your detector into a room to pick up any unusual presences. Be aware that electrical appliances such as televisions will cause the detector to dance around because of the signal they emit.

THE BUILD

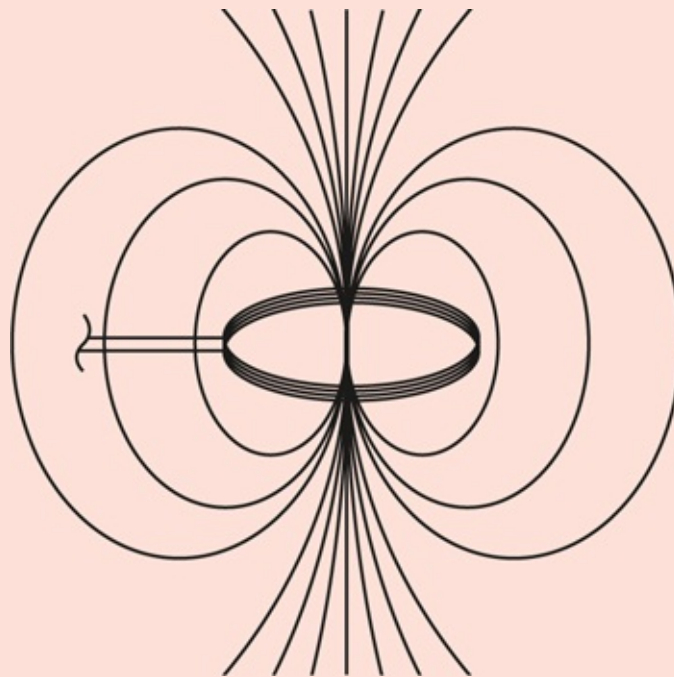
1. Place the LEDs into the breadboard with the legs on either side of the center divide (see “[Breadboards](#)” on [page 4](#) for more on the layout of the breadboard), as shown in [Figure 6-1](#). I started with a yellow LED, then used six green and three red LEDs to create a scale from left to right. You can use any color LEDs and position them in the sequence you prefer.

1. **FIGURE 6-1:**
Placing the LEDs



ELECTROMAGNETIC FIELDS

Electric fields are created by differences in voltage: the higher the voltage, the stronger the resultant field. *Magnetic fields* are created when electric current flows: the greater the current, the stronger the magnetic field. An *electromagnetic field (EMF)* can be thought of as a combination of the two.



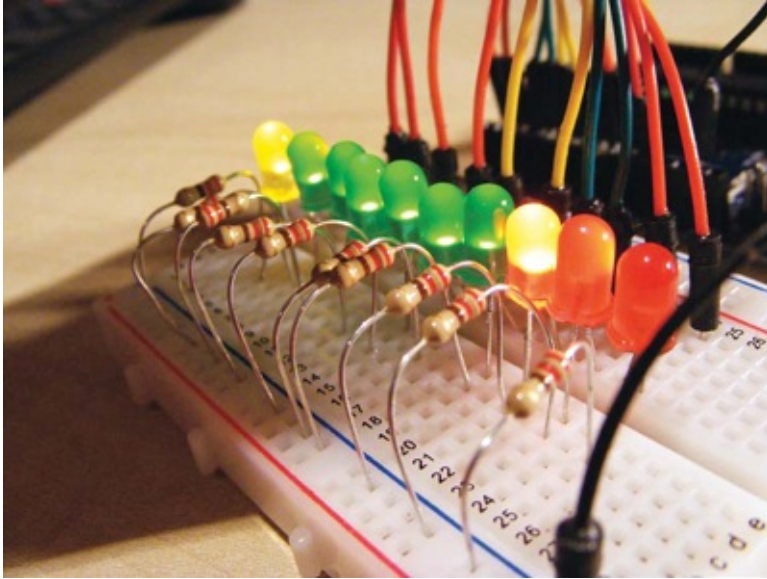
Electromagnetic fields are present everywhere in the environment but are invisible to the human eye. Electric fields are produced by the local buildup of electric charges in the

atmosphere and associated with thunderstorms. The earth constantly emits a magnetic field. It is used by birds and fish for navigation and causes a compass needle to orient to the north.

2. Connect one leg of a 220-ohm resistor to each negative LED leg, and insert the other resistor leg in the GND rail of the breadboard (see [Figure 6-2](#)). Connect each positive LED leg to digital pins 2 through 11 in turn.

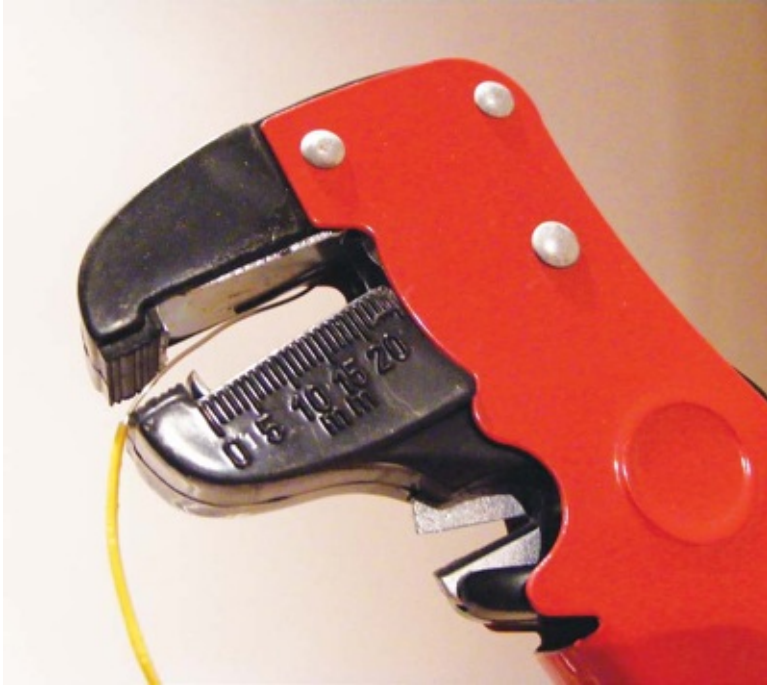
LEDS	ARDUINO
Positive legs	Pins 2–11
Negative legs	GND via 220-ohm resistors

2. **FIGURE 6-2:**
Connecting the LEDs to the breadboard



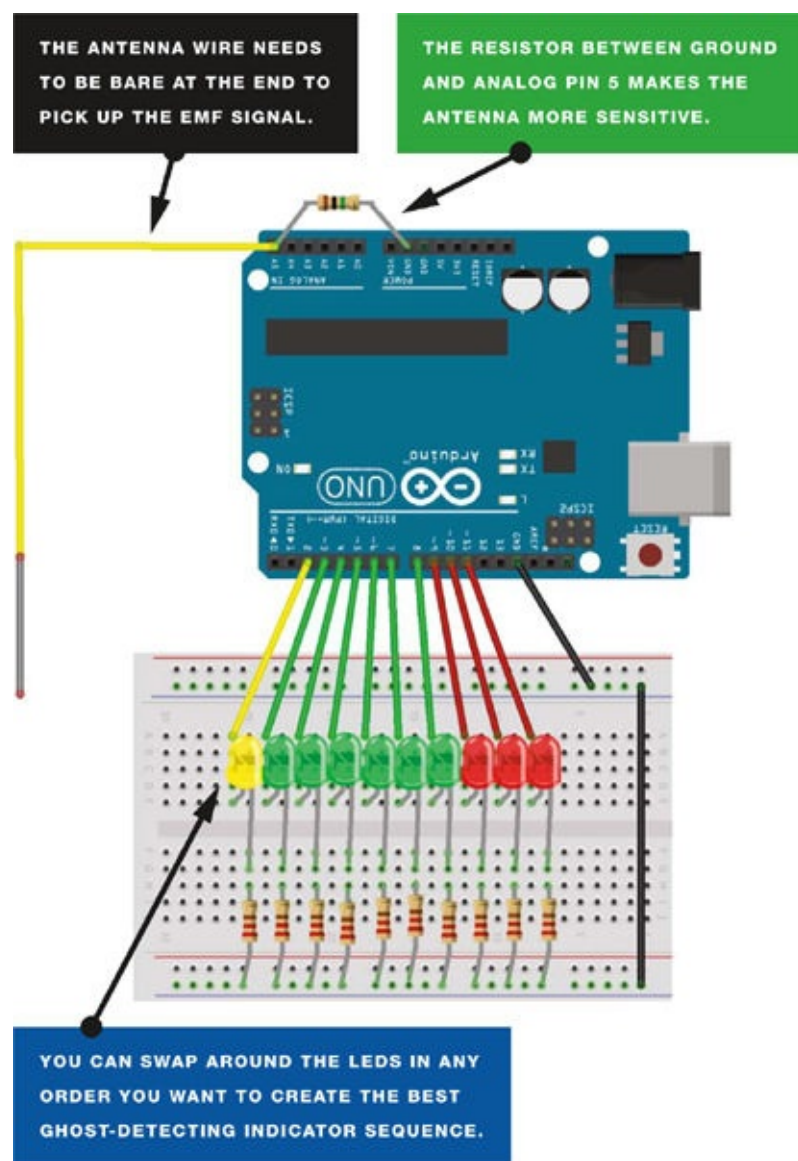
3. Take the 20-centimeter length of single-core wire and use a wire stripper to strip about 1 centimeter of the insulation from one end. Attach this end to Arduino pin A5. Strip about 7 centimeters from the other end—this open, bare wire end is your antenna and will pick up the electromagnetic signal (see [Figure 6-3](#)).

3. **FIGURE 6-3:**
Stripping wire to create an antenna



4. Connect one leg of the 1M-ohm resistor directly to GND on the Arduino and the other leg to Arduino pin A5; this will increase the sensitivity of your device.
5. Check that your setup matches that of [Figure 6-4](#), and then upload the code in “[The Sketch](#)” on [page 59](#).

5. **FIGURE 6-4:**
Circuit diagram for the ghost detector



THE SKETCH

The bare wire picks up the signal from electromagnetic fields in the atmosphere and sends a value between 0 and 1023 to the Arduino. The sketch evaluates the reading from the analog pin to determine how many LEDs are switched on or off in sequence to indicate the strength of the electromagnetic signal. For example, 1023 would be the highest value, so all LEDs would be lit; a reading of 550 would light five LEDs. The sketch loops to continuously read the analog input, and the LED lights constantly move to show the reading. If you find that the EMF readings set off your LED sequence to the maximum level every time, reduce the `senseLimit` value to compensate. The sketch takes an average of 25 number readings each time it loops through, and uses the average from those readings to mitigate big fluctuations that may cause the LEDs to light up too quickly.

NOTE

Once you've completed the ghost detector, try adding some sounds that beep at increasing speeds or volumes depending on the reading. Build a casing for the project to have your own handheld sensor to take on ghost-

bunting endeavors. You can also experiment by trying various types and thicknesses of wire, and by taking away the resistor for different levels of sensitivity.

```
// Code by James Newbould used with kind permission
#define NUMREADINGS 25 // Raise number to increase data smoothing
int senseLimit = 1023; // Raise number to decrease sensitivity of
                        // the antenna (up to 1023 max)
int probePin = 5; // Set analog pin 5 as the antenna pin
int val = 0;      // Reading from probePin

// Pin connections to LED bar graph with resistors in series
int LED1 = 11;
int LED2 = 10;
int LED3 = 9;
int LED4 = 8;
int LED5 = 7;
int LED6 = 6;
int LED7 = 5;
int LED8 = 4;
int LED9 = 3;
int LED10 = 2;
int readings[NUMREADINGS]; // Readings from the analog input
int index = 0;              // Index of the current reading
int total = 0;              // Running total
int average = 0;           // Final average of the probe reading

void setup() {
  pinMode(2, OUTPUT); // Set LED bar graph pins as outputs
  pinMode(3, OUTPUT);
  pinMode(4, OUTPUT);
  pinMode(5, OUTPUT);
  pinMode(6, OUTPUT);
  pinMode(7, OUTPUT);
  pinMode(8, OUTPUT);
  pinMode(9, OUTPUT);
  pinMode(10, OUTPUT);
  pinMode(11, OUTPUT);

  Serial.begin(9600); // Initiate serial connection with IDE for
                     // debugging and so on
  for (int i = 0; i < NUMREADINGS; i++)
    readings[i] = 0; // Initialize all readings to 0
}

void loop() {
  val = analogRead(probePin); // Take a reading from probe
  if (val >= 1) {              // If the reading isn't zero, proceed
    val = constrain(val, 1, senseLimit); // If the reading is
                                         // higher than the current
                                         // senseLimit value, update
                                         // senseLimit value with
                                         // higher reading
    val = map(val, 1, senseLimit, 1, 1023); // Remap the constrained
                                             // value within a 1 to
                                             // 1023 range

    total -= readings[index]; // Subtract the last reading
    readings[index] = val;    // Read from the sensor
    total += readings[index]; // Add the reading to the total
    index = (index + 1);     // Advance to the next index
    if (index >= NUMREADINGS) // If we're at the end of the array
      index = 0;              // loop around to the beginning
    average = total / NUMREADINGS; // Calculate the average reading
  }
}
```

```

if (average > 50) { // If the average reading is higher than 50
    digitalWrite(LED1, HIGH); // turn on the first LED
}
else {
    digitalWrite(LED1, LOW); // If it's not
    // turn off that LED
}
if (average > 150) { // And so on
    digitalWrite(LED2, HIGH);
}
else {
    digitalWrite(LED2, LOW);
}
if (average > 250) {
    digitalWrite(LED3, HIGH);
}
else {
    digitalWrite(LED3, LOW);
}
if (average > 350) {
    digitalWrite(LED4, HIGH);
}
else {
    digitalWrite(LED4, LOW);
}
if (average > 450) {
    digitalWrite(LED5, HIGH);
}
else {
    digitalWrite(LED5, LOW);
}
if (average > 550) {
    digitalWrite(LED6, HIGH);
}
else {
    digitalWrite(LED6, LOW);
}
if (average > 650) {
    digitalWrite(LED7, HIGH);
}
else {
    digitalWrite(LED7, LOW);
}
if (average > 750) {
    digitalWrite(LED8, HIGH);
}
else {
    digitalWrite(LED8, LOW);
}
if (average > 850) {
    digitalWrite(LED9, HIGH);
}
else {
    digitalWrite(LED9, LOW);
}
if (average > 950) {
    digitalWrite(LED10, HIGH);
}
else {
    digitalWrite(LED10, LOW);
}
Serial.println(val); // Use output to aid in calibrating
}
}

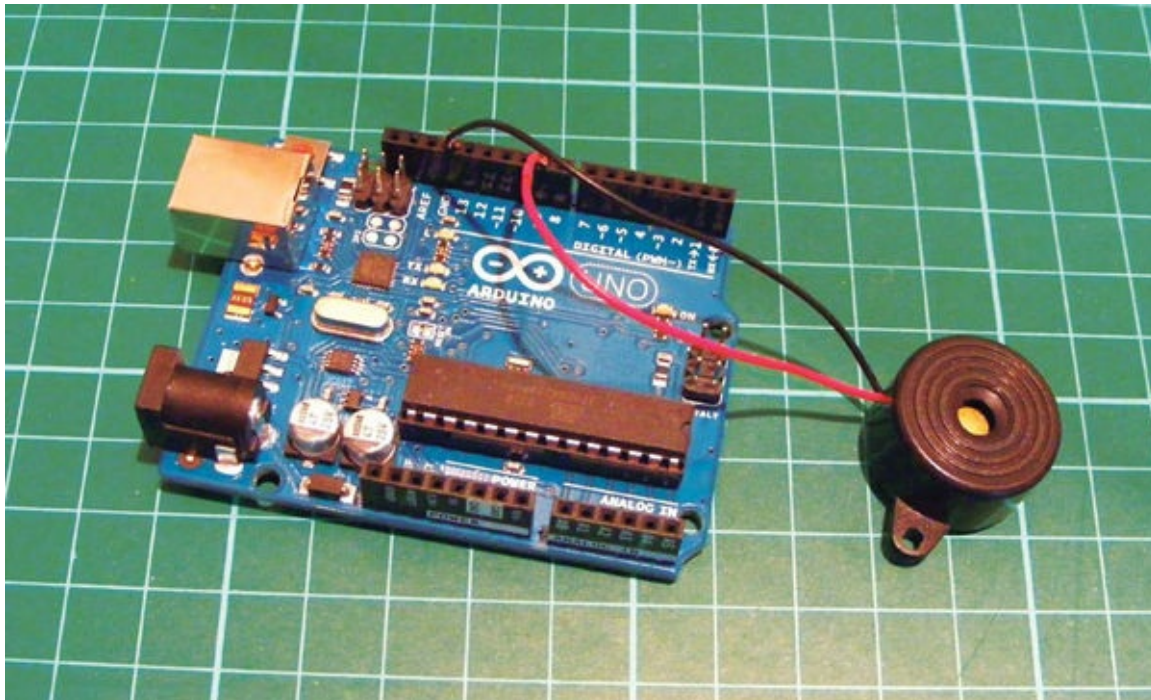
```

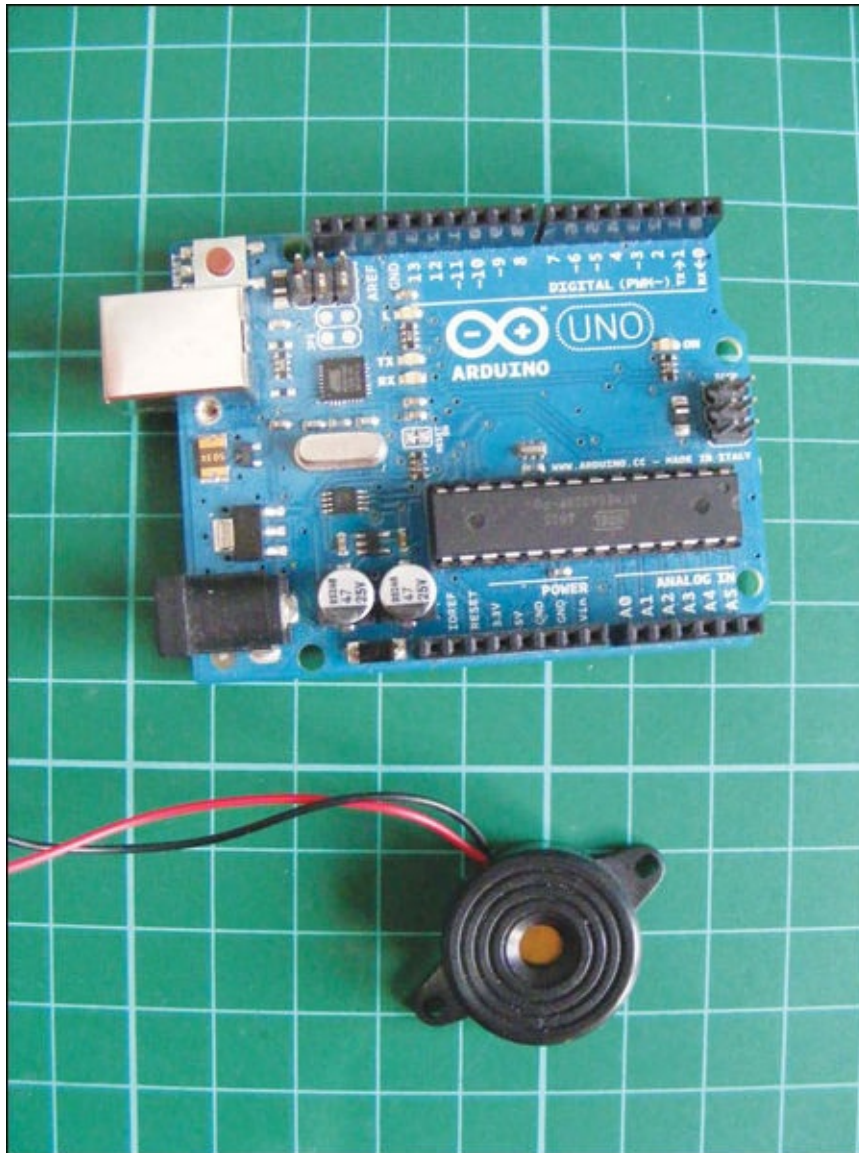
PART 2

SÖND

PROJECT 7: ARDUINO MELODY

SO FAR ALL THE PROJECTS HAVE BEEN VISUAL, SO NOW IT'S TIME TO MAKE SOME MUSIC. IN THIS PROJECT WE WILL BE USING A PIEZOELECTRIC BUZZER TO PLAY SOME MELODIES.





PARTS REQUIRED

- Arduino board
- Piezo buzzer

HOW IT WORKS

The Arduino melody uses a piezo buzzer to create frequencies that resemble recognizable notes. You use the Arduino IDE to give the order, rate, and duration of the notes to play a specific tune.

Piezos are inexpensive buzzers often used in small toys. A piezo element without its plastic housing looks like a gold metallic disc with connected positive (typically red) and negative (typically black) wires. A piezo is capable only of making a clicking sound, which we create by applying voltage. We can make recognizable notes by getting the piezo to click hundreds of times a second at a particular frequency, so first we need to know the frequency of the different tones we want. [Table 7-1](#) shows the notes and their corresponding frequencies. *Period* is the duration of time, in microseconds, at which the frequency is created. We halve this number to get the `timeHigh` value, which is used in the code to create the note.

TABLE 7-1:
Notes and their corresponding frequencies

NOTE	FREQUENCY	PERIOD	TIMEHIGH
C	261 Hz	3,830	1915
D	294 Hz	3,400	1700
E	329 Hz	3,038	1519
F	349 Hz	2,864	1432
G	392 Hz	2,550	1275
A	440 Hz	2,272	1136
B	493 Hz	2,028	1014
C	523 Hz	1,912	956

The code sends a square wave of the appropriate frequency to the piezo, generating the corresponding tone (see [Project 2](#) for more on waveform). The tones are calculated through the following equation:

$$\text{timeHigh} = \text{period} / 2 = 1 / (2 * \text{toneFrequency})$$

The setup of this project is really simple and uses only two wires connected to the Arduino.

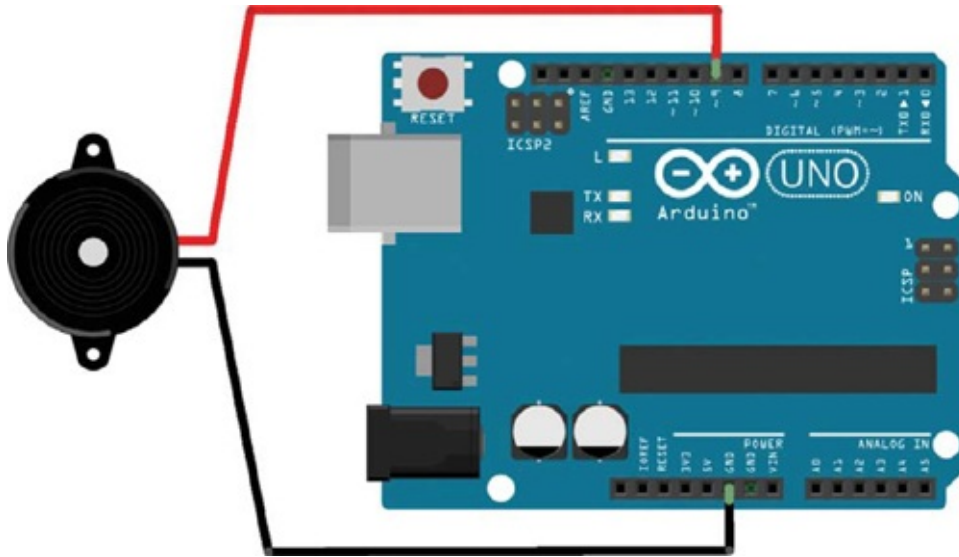
THE BUILD

1. Connect the piezo's black wire directly to GND on the Arduino, and the red wire to Arduino pin 9.

PIEZO	ARDUINO
Red wire	Pin 9
Black wire	GND

2. Check that your setup matches that of [Figure 7-1](#), and then upload the code shown next in "[The Sketch](#)".

2. **FIGURE 7-1**
Circuit diagram for the Arduino melody



THE SKETCH

We'll start off with a simple tune. At ❶, we tell the IDE that the tune is made up of 15 notes. Then we store the notes of the melody in a character array as a text string in the order in which they should be played, and the length for which each note will play is stored in another array as integers. If you want to change the tune, you can alter the notes in the array at ❷, and the number of beats for which each corresponding note plays at ❸. Finally at ❹ we set the tempo at which the tune will be played. Put it all together, and what does it play?

```
// Melody (cleft) 2005 D. Cuartielles for K3

int speakerPin = 9; // Pin connected to the piezo
❶ int length = 15; // Number of notes
❷ char notes[] = "ccggaagffeeddc "; // A space represents a rest
❸ int beats[] = { 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 2, 4 };
❹ int tempo = 300;

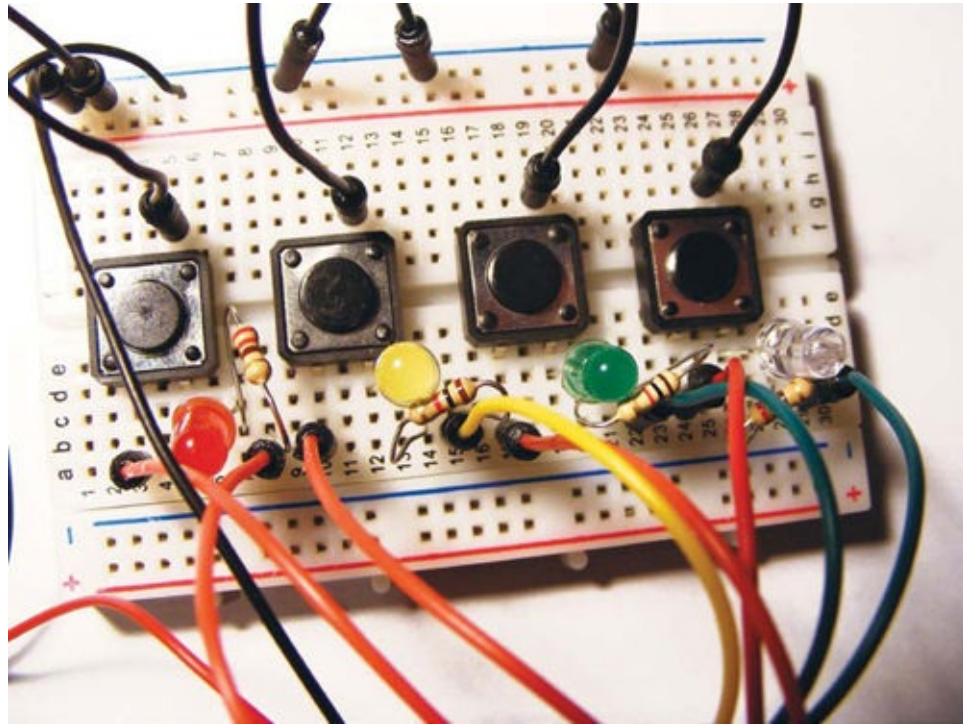
void playTone(int tone, int duration) {
  for (long i = 0; i < duration * 1000L; i += tone * 2) {
    digitalWrite(speakerPin, HIGH);
    delayMicroseconds(tone);
    digitalWrite(speakerPin, LOW);
    delayMicroseconds(tone);
  }
}

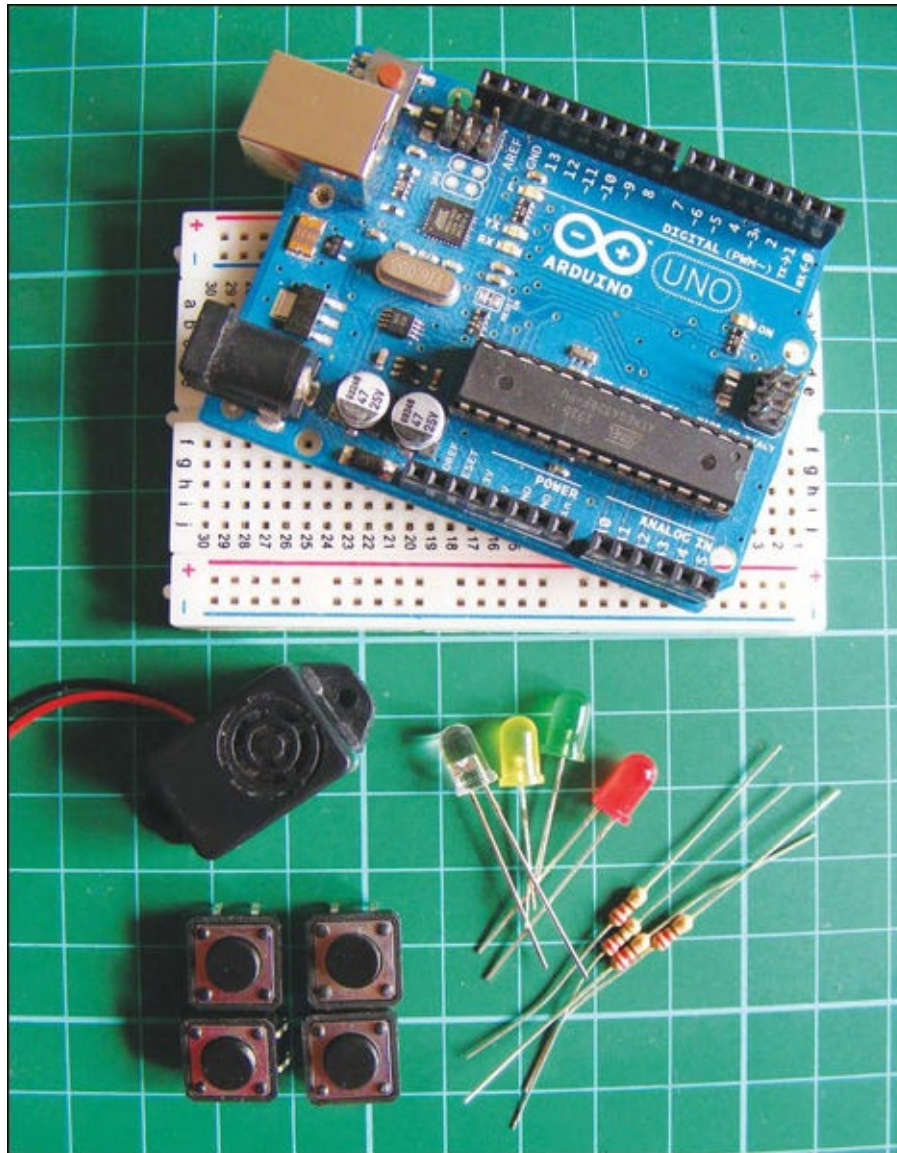
// Set timeHigh value to specific notes
void playNote(char note, int duration) {
  char names[] = { 'c', 'd', 'e', 'f', 'g', 'a', 'b', 'C' };
  int tones[] = { 1915, 1700, 1519, 1432, 1275, 1136, 1014, 956 };
  for (int i = 0; i < 8; i++) { // Play tone that corresponds
    // to note name
    if (names[i] == note) {
      playTone(tones[i], duration);
    }
  }
}
```

```
}  
  
void setup() {  
  pinMode(speakerPin, OUTPUT); // Set speakerPin as output  
}  
  
// Play the tune  
void loop() {  
  for (int i = 0; i < length; i++) {  
    if (notes[i] == ' ') {  
      delay(beats[i] * tempo); // Rest  
    }  
    else {  
      playNote(notes[i], beats[i] * tempo);  
    }  
    delay(tempo / 2); // Pause between notes  
  }  
}
```

PROJECT 8: MEMORY GAME

IN THIS PROJECT WE'LL CREATE OUR OWN VERSION OF AN ATARI ARCADE MEMORY GAME CALLED TOUCH ME, USING FOUR LEDS, FOUR PUSHBUTTON SWITCHES, A PIEZO BUZZER, AND SOME RESISTORS AND JUMPER WIRES.





PARTS REQUIRED

- Arduino board
- Breadboard
- Jumper wires
- Piezo buzzer
- 4 momentary tactile four-pin pushbuttons
- 4 LEDs
- 4 220-ohm resistors

LIBRARIES REQUIRED

- Tone

HOW IT WORKS

The original Atari game had four colored panels, each with an LED that lit up in a particular pattern that players had to repeat back (see [Figure 8-1](#)).

FIGURE 8-1:
The original *Touch Me* game

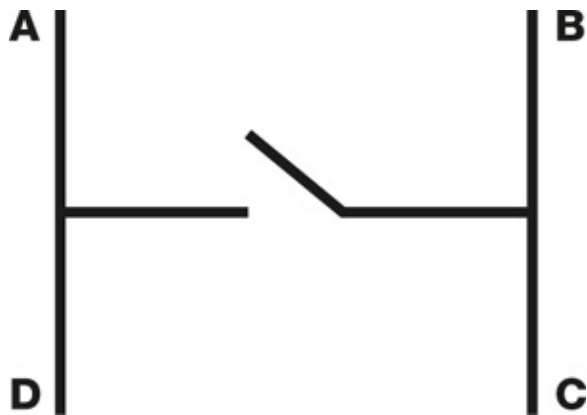


This memory game plays a short introductory tune and flashes an LED. When you press the correct corresponding button, the lights flash again in a longer sequence. Each time you repeat the sequence back correctly, the game adds an extra step to make the sequence more challenging for you. When you make an error, the game resets itself.

THE BUILD

1. Place the pushbuttons in the breadboard so they straddle the center break with pins A and B on one side of the break, and C and D on the other, as shown in [Figure 8-2](#). (See [Project 1](#) for more information on how the pushbutton works.)

- FIGURE 8-2:**
A pushbutton has four pins.



- Connect pin B of each pushbutton to the GND rail of your breadboard, and connect the rail to Arduino GND.
- Connect pin D of each pushbutton to Arduino’s digital pins 2 through 5 in order.
- Insert the LEDs into the breadboard with the shorter, negative legs connected to pin C of each pushbutton. Insert the positive leg into the hole on the right, as shown in the circuit diagram in [Figure 12-3](#).

PUSHBUTTON	ARDUINO/LED
Pin B	GND
Pin C	LED negative legs
Pin D	Arduino pins 2–5

- Place a 220-ohm resistor into the breadboard with one wire connected to the positive leg of each LED. Connect the other wire of the resistor to the Arduino as follows.

LEDS	ARDUINO/PUSHBUTTON
Positive legs	Arduino pins 8–11 via 220-ohm resistors
Negative legs	Pushbutton pin C

Make sure the red LED connected to pin 11 is paired with the pushbutton connected to pin 5, the yellow LED connected to pin 10 is paired with the pushbutton connected to pin 4, the green LED connected to pin 9 is paired with the pushbutton connected to pin 3, and the blue LED connected to pin 8 is paired with the pushbutton connected to pin 2.

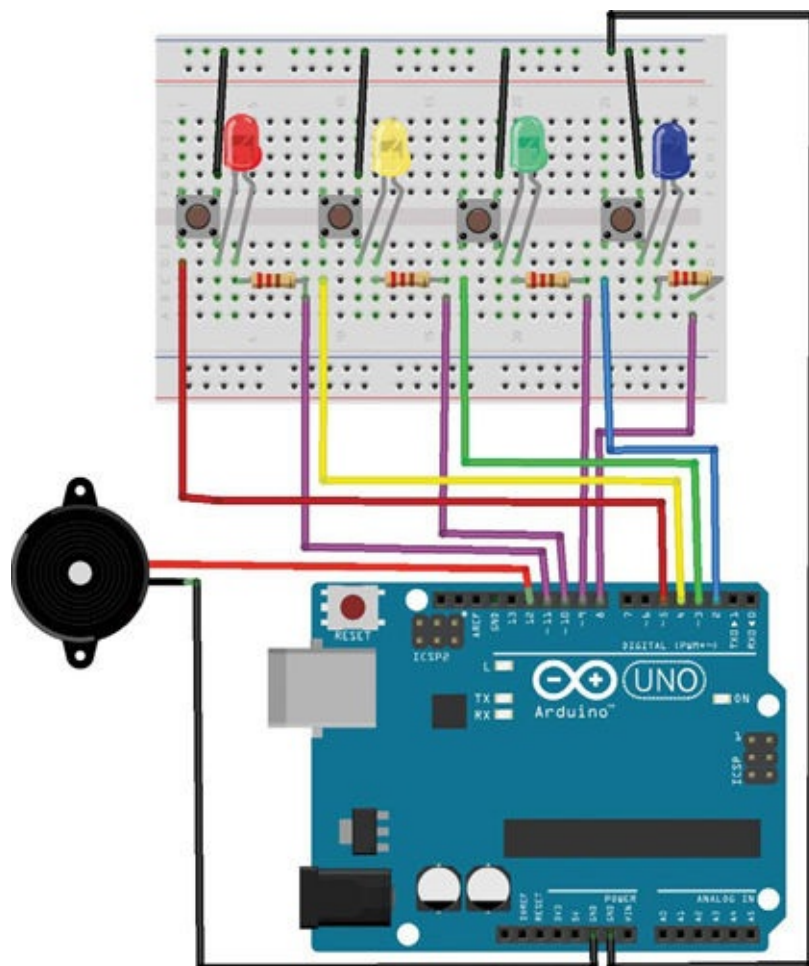
- Connect the black wire of the piezo directly to Arduino GND, and the red wire to Arduino pin 12.



PIEZO	ARDUINO
Red wire	Pin 12
Black wire	GND

7. Check your setup against [Figure 8-3](#), and then upload the code in “[The Sketch](#)” on [page 73](#).

7. **FIGURE 8-3:**
Circuit diagram for the memory game



THE SKETCH

The sketch generates a random sequence in which the LEDs will light; a random value generated for y in the pattern loop determines which LED is lit (e.g., if y is 2, the LED connected to pin 2 will light). You have to follow and repeat back the pattern to advance to the next level.

In each level, the previous lights are repeated and one more randomly generated light is added to the pattern. Each light is associated with a different tone from the piezo, so you get a different tune each time, too. When you get a sequence wrong, the sketch restarts with a different random sequence. For the sketch to compile correctly, you will need to install the Tone library (available from <http://nostarch.com.com/arduinohandbook/>). See “Libraries” on page 7 for details.

```
// Used with kind permission from Abdullah Alhazmy www.Alhazmy13.net
```

```
#include <Tone.h>
Tone speakerpin;
int starttune[] = {NOTE_C4, NOTE_F4, NOTE_C4, NOTE_F4, NOTE_C4,
                  NOTE_F4, NOTE_C4, NOTE_F4, NOTE_G4, NOTE_F4,
                  NOTE_E4, NOTE_F4, NOTE_G4};
int duration2[] = {100, 200, 100, 200, 100, 400, 100, 100, 100, 100,
                  200, 100, 500};
int note[] = {NOTE_C4, NOTE_C4, NOTE_G4, NOTE_C5, NOTE_G4, NOTE_C5};
int duration[] = {100, 100, 100, 300, 100, 300};
boolean button[] = {2, 3, 4, 5}; // Pins connected to
                                // pushbutton inputs
```

```

boolean ledpin[] = {8, 9, 10, 11}; // Pins connected to LEDs
int turn = 0; // Turn counter
int buttonstate = 0; // Check pushbutton state
int randomArray[100]; // Array that can store up to 100 inputs
int inputArray[100];

void setup() {
  Serial.begin(9600);
  speakerpin.begin(12); // Pin connected to piezo buzzer
  for (int x = 0; x < 4; x++) {
    pinMode(ledpin[x], OUTPUT); // Set LED pins as output
  }
  for (int x = 0; x < 4; x++) {
    pinMode(button[x], INPUT); // Set pushbutton pins as inputs
    digitalWrite(button[x], HIGH); // Enable internal pullup;
    // pushbuttons start in high
    // position; logic reversed
  }
  // Generate "more randomness" with randomArray for the output
  // function so pattern is different each time
  randomSeed(analogRead(0));
  for (int thisNote = 0; thisNote < 13; thisNote++) {
    speakerpin.play(starttune[thisNote]); // Play the next note
    if (thisNote == 0 || thisNote == 2 || thisNote == 4 ||
        thisNote == 6) { // Hold the note
      digitalWrite(ledpin[0], HIGH);
    }
    if (thisNote == 1 || thisNote == 3 || thisNote == 5 ||
        thisNote == 7 || thisNote == 9 || thisNote == 11) {
      digitalWrite(ledpin[1], HIGH);
    }
    if (thisNote == 8 || thisNote == 12) {
      digitalWrite(ledpin[2], HIGH);
    }
    if (thisNote == 10) {
      digitalWrite(ledpin[3], HIGH);
    }
    delay(duration2[thisNote]);
    speakerpin.stop(); // Stop for the next note
    digitalWrite(ledpin[0], LOW);
    digitalWrite(ledpin[1], LOW);
    digitalWrite(ledpin[2], LOW);
    digitalWrite(ledpin[3], LOW);
    delay(25);
  }
  delay(1000);
}

void loop() {
  // Generate the array to be matched by the player
  for (int y = 0; y <= 99; y++) {
    digitalWrite(ledpin[0], HIGH);
    digitalWrite(ledpin[1], HIGH);
    digitalWrite(ledpin[2], HIGH);
    digitalWrite(ledpin[3], HIGH);
    // Play the next note
    for (int thisNote = 0; thisNote < 6; thisNote++) {
      speakerpin.play(note[thisNote]); // Hold the note
      delay(duration[thisNote]); // Stop for the next note
      speakerpin.stop();
      delay(25);
    }
    digitalWrite(ledpin[0], LOW);
    digitalWrite(ledpin[1], LOW);
  }
}

```



```

digitalWrite(ledpin[2], LOW);
digitalWrite(ledpin[3], LOW);
delay(1000);
// Limited by the turn variable
for (int y = turn; y <= turn; y++) {
  Serial.println("");
  Serial.print("Turn: ");
  Serial.print(y);
  Serial.println("");
  randomArray[y] = random(1, 5); // Assign a random number (1-4)
  // Light LEDs in random order
  for (int x = 0; x <= turn; x++) {
    Serial.print(randomArray[x]);
    for (int y = 0; y < 4; y++) {
      if (randomArray[x] == 1 && ledpin[y] == 8) {
        digitalWrite(ledpin[y], HIGH);
        speakerpin.play(NOTE_G3, 100);
        delay(400);
        digitalWrite(ledpin[y], LOW);
        delay(100);
      }
      if (randomArray[x] == 2 && ledpin[y] == 9) {
        digitalWrite(ledpin[y], HIGH);
        speakerpin.play(NOTE_A3, 100);
        delay(400);
        digitalWrite(ledpin[y], LOW);
        delay(100);
      }
      if (randomArray[x] == 3 && ledpin[y] == 10) {
        digitalWrite(ledpin[y], HIGH);
        speakerpin.play(NOTE_B3, 100);
        delay(400);
        digitalWrite(ledpin[y], LOW);
        delay(100);
      }
      if (randomArray[x] == 4 && ledpin[y] == 11) {
        digitalWrite(ledpin[y], HIGH);
        speakerpin.play(NOTE_C4, 100);
        delay(400);
        digitalWrite(ledpin[y], LOW);
        delay(100);
      }
    }
  }
}
input();
}
}

// Check whether input matches the pattern
void input() {
  for (int x = 0; x <= turn;) {
    for (int y = 0; y < 4; y++) {
      buttonstate = digitalRead(button[y]); // Check for button push
      if (buttonstate == LOW && button[y] == 2) {
        digitalWrite(ledpin[0], HIGH);
        speakerpin.play(NOTE_G3, 100);
        delay(200);
        digitalWrite(ledpin[0], LOW);
        inputArray[x] = 1;
        delay(250);
        Serial.print(" ");
        Serial.print(1);
      }
    }
  }
}

```

```

    // Check if value of user input matches the generated array
    if (inputArray[x] != randomArray[x]) {
        fail(); // If not, fail function is called
    }
    x++;
}
if (buttonstate == LOW && button[y] == 3) {
    digitalWrite(ledpin[1], HIGH);
    speakerpin.play(NOTE_A3, 100);
    delay(200);
    digitalWrite(ledpin[1], LOW);
    inputArray[x] = 2;
    delay(250);
    Serial.print(" ");
    Serial.print(2);
    if (inputArray[x] != randomArray[x]) {
        fail();
    }
    x++;
}
if (buttonstate == LOW && button[y] == 4) {
    digitalWrite(ledpin[2], HIGH);
    speakerpin.play(NOTE_B3, 100);
    delay(200);
    digitalWrite(ledpin[2], LOW);
    inputArray[x] = 3;
    delay(250);
    Serial.print(" ");
    Serial.print(3);
    if (inputArray[x] != randomArray[x]) {
        fail();
    }
    x++;
}
if (buttonstate == LOW && button[y] == 5) {
    digitalWrite(ledpin[3], HIGH);
    speakerpin.play(NOTE_C4, 100);
    delay(200);
    digitalWrite(ledpin[3], LOW);
    inputArray[x] = 4;
    delay(250);
    Serial.print(" ");
    Serial.print(4);
    if (inputArray[x] != randomArray[x]) {
        fail();
    }
    x++;
}
}
}
delay(500);
turn++; // Increment turn count
}

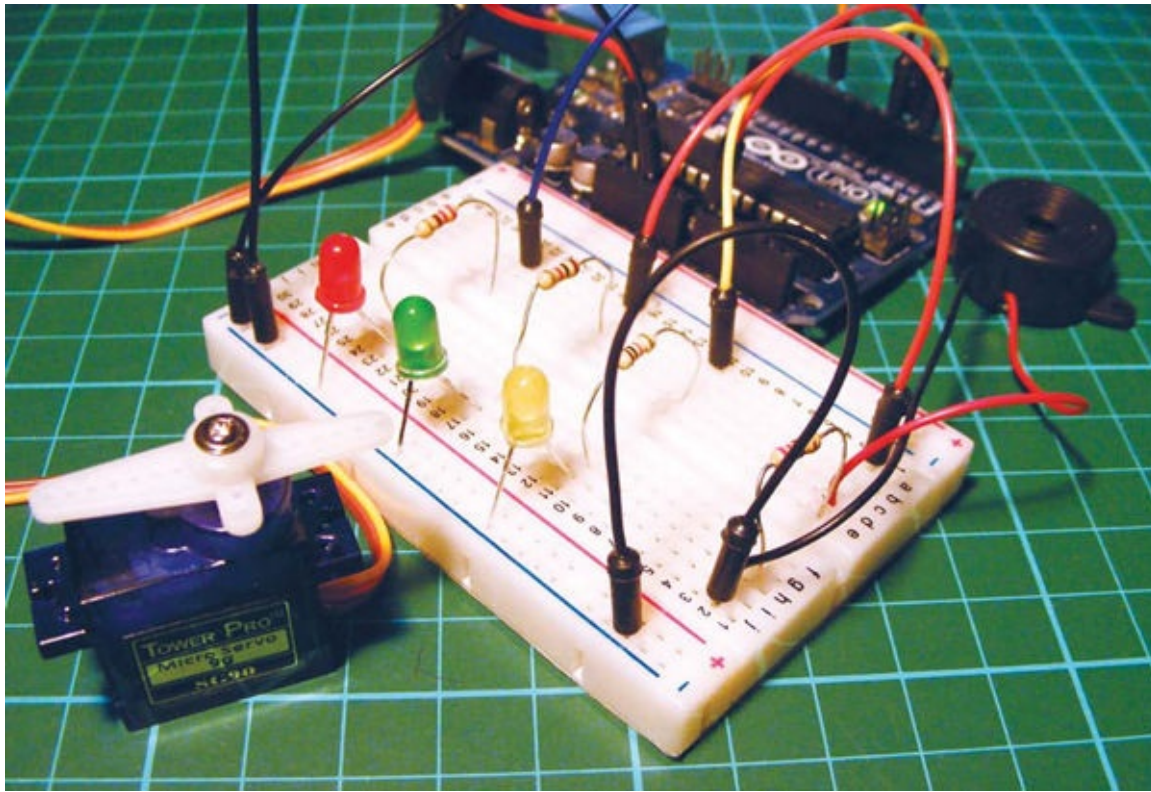
// Function used if player fails to match the sequence
void fail() {
    for (int y = 0; y <= 2; y++) { // Flash lights to indicate failure
        digitalWrite(ledpin[0], HIGH);
        digitalWrite(ledpin[1], HIGH);
        digitalWrite(ledpin[2], HIGH);
        digitalWrite(ledpin[3], HIGH);
        speakerpin.play(NOTE_G3, 300);
        delay(200);
        digitalWrite(ledpin[0], LOW);
    }
}

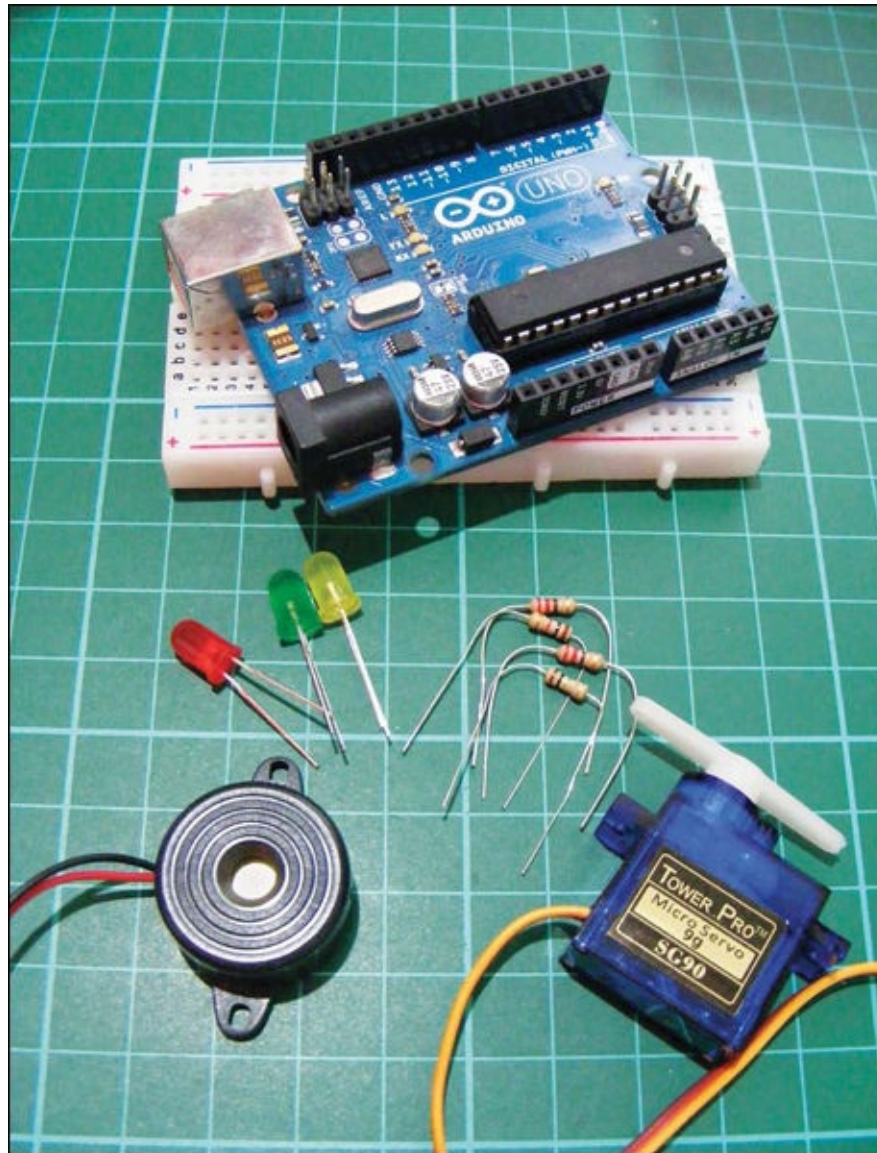
```

```
digitalWrite(ledpin[1], LOW);  
digitalWrite(ledpin[2], LOW);  
digitalWrite(ledpin[3], LOW);  
speakerpin.play(NOTE_C3, 300);  
delay(200);  
}  
delay(500);  
turn = -1; // Reset turn value to start the game again  
}
```

PROJECT 9: SECRET KNOCK LOCK

FOR CENTURIES CLANDESTINE GROUPS HAVE USED SECRET KNOCKS TO PREVENT UNAUTHORIZED ENTRY. LET'S BRING THIS SYSTEM INTO MODERN TIMES, BY CREATING OUR OWN ELECTRONIC GATEKEEPER.





PARTS REQUIRED

- Arduino board
- Breadboard
- Jumper wires
- Tower Pro SG90 9g servomotor
- Piezo buzzer
- 3 LEDs
- 1M-ohm resistor
- 3 220-ohm resistors

LIBRARIES REQUIRED

HOW IT WORKS

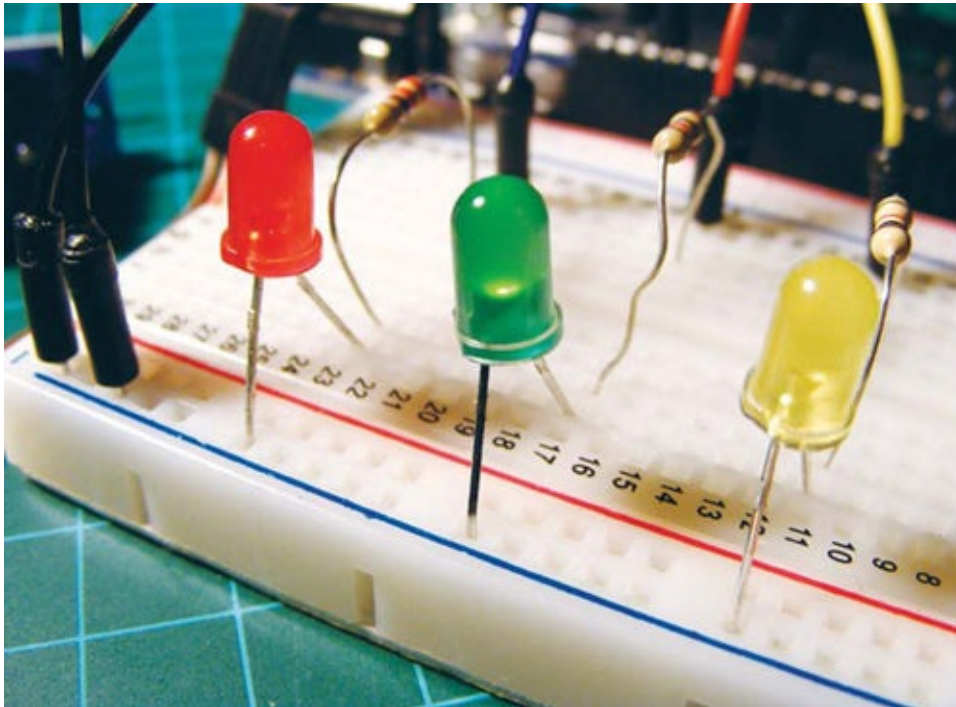
In this project, you'll make a circuit that moves a servo arm to unlock a box or door when you provide the correct secret knock. So far we've been using a piezo buzzer only to make noise, but we can also use it as a sensor to detect sounds—in this case, knocks. When a piezo is struck it rings like a bell, but instead of producing sound it outputs voltage, which generates a number depending on the force of the strike. We'll measure this voltage in numbers, and if the knocks fall within a certain range, the Arduino will register them as correct. If three knocks of the correct voltage are detected, you've cracked the code, and the servo arm moves to unlock the box or door.

Here are the two lines of code we'll use later in the sketch to set the range for the voltage; if the voltage is between 10 and 100, the knock will be registered.

```
const int quietKnock = 10;  
const int loudKnock = 100;
```

If you knock too softly or too hard, the knock won't register. You'll need to do three "correct" knocks to trigger the servo arm to move. When the correct sequence and strength of knock are registered, the servo arm swings 90 degrees to "unlock" whatever it is set up with. The LEDs, shown in [Figure 9-1](#), serve as indicators of your lock's status: the red LED lights when the knocks are incorrect and the servo arm has not moved (that is, the box or door is still locked); the yellow LED flashes when a knock is registered and a correct code is sensed; and the green LED lights and the servomotor moves after three correct knocks.

FIGURE 9-1:
The LED setup



For the best result, remove your piezo from its casing and attach it directly to the inside of a box or outside of a door so it is more sensitive to the vibration of the knock.

THE BUILD

1. Insert a 1M-ohm resistor into your breadboard and connect the piezo's red wire to one leg and its black wire to the other. Connect the black wire to the GND rail, and the red wire to Arduino pin A0.

PIEZO	ARDUINO
Red wire	A0 via 1M-ohm resistor
Black wire	GND via 1M-ohm resistor

2. Connect the servo's yellow signal wire directly to Arduino pin 9, its brown wire to GND, and its red wire to +5V.

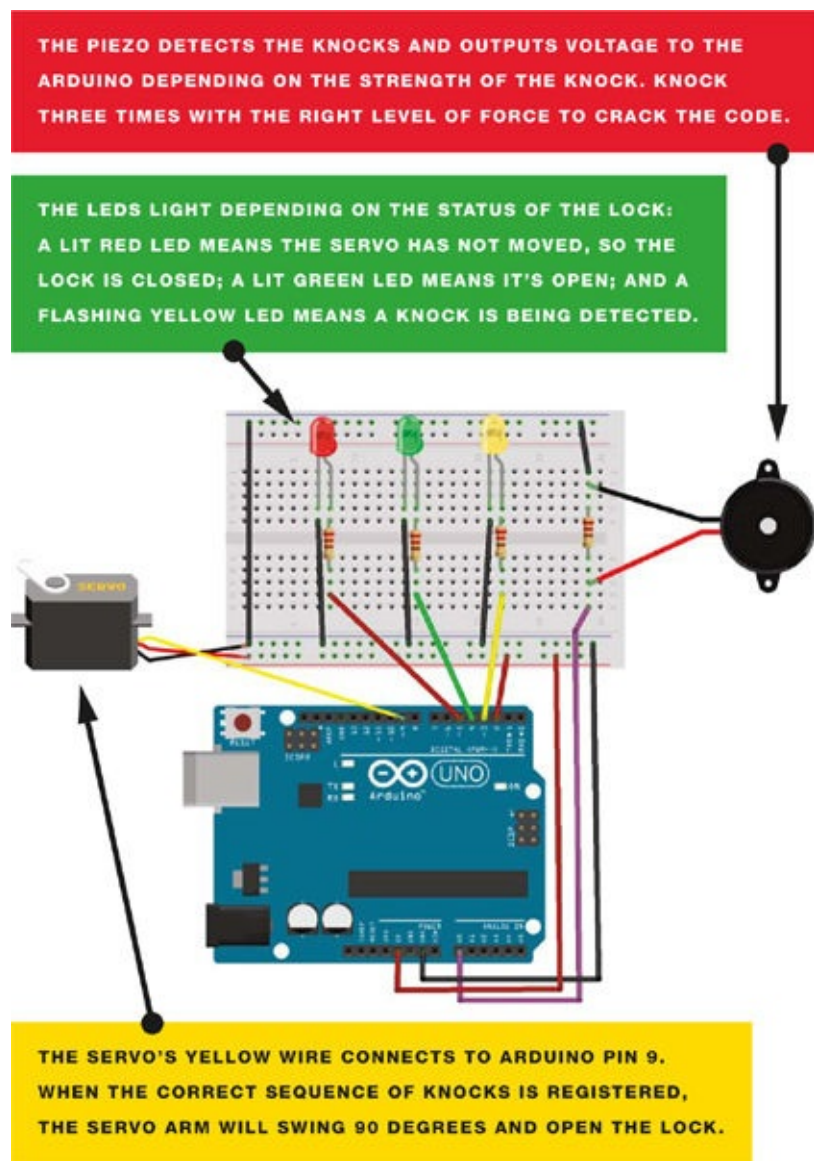
SERVO	ARDUINO
Yellow wire	Pin 9
Red wire	+5V
Brown wire	GND

3. Insert the LEDs into your breadboard with the short, negative legs connected to GND. The positive legs should connect to the pins via 220-ohm resistors as follows: yellow connects to Arduino pin 3, green to pin 4, and red to pin 5.

LEDS	ARDUINO
Positive legs	Pins 3–5 via 220-ohm resistors
Negative legs	GND

4. Connect Arduino pin 2 to the positive power rail. In our setup this is always on, but you could add a switch in the connection between Arduino pin 2 and the power rail to save power when the project is not in use.
5. Connect the breadboard rails to Arduino GND and +5V.
6. Make sure your setup matches the circuit diagram in [Figure 9-2](#), and then upload the code in “[The Sketch](#)” on [page 82](#).

6. **FIGURE 9-2:**
The circuit diagram for the secret knock lock



THE SKETCH

We first call on the Servo library and set Arduino pin 9 to control the servo. LEDs are attached to Arduino pins 3, 4, and 5, and these will light depending on the validity of a knock. The piezo acts as a sensor rather than a buzzer in this project and is attached to Arduino pin A0. When someone knocks, the knock is sensed by the piezo and a voltage value is sent to the A0 analog pin of the Arduino depending on the strength of the knock—the harder the knock, the higher the value. A knock with a value below 10 is considered too quiet, and one with a value above 100 too loud, so neither will be accepted as a valid knock. The red LED lights if the knock is not accepted, and the yellow LED lights if it is. Any knock value between 10 and 100 is accepted as a valid knock and counted, and if three valid knocks are received, the servomotor moves and the green LED lights.

As mentioned earlier, these are the two lines of code that set the parameters for measuring the voltage:

```
const int quietKnock = 10;  
const int loudKnock = 100;
```

If you were feeling particularly secretive, you could set this range even tighter to make the code harder to crack. Here's the sketch:

```
/* Created 18 September 2012 by Scott Fitzgerald
   Thanks to Federico Vanzati for improvements
   http://arduino.cc/starterKit
   This example code is part of the public domain.
*/

#include <Servo.h>
Servo servo9; // Pin connected to servo mpo

const int piezo = A0; // Pin connected to piezo
const int switchPin = 2; // Pin connected to servo
const int yellowLed = 3; // Pin connected to yellow LED
const int greenLed = 4; // Pin connected to green LED
const int redLed = 5; // Pin connected to red LED

int knockVal; // Value for the knock strength
int switchVal;

const int quietKnock = 10; // Set min value that will be accepted
const int loudKnock = 100; // Set max value that will be accepted
boolean locked = false; // A true or false variable
int numberOfKnocks = 0; // Value for number of knocks

void setup() {
  servo9.attach(9);
  pinMode(yellowLed, OUTPUT); // Set LED pins as outputs
  pinMode(greenLed, OUTPUT);
  pinMode(redLed, OUTPUT);
  pinMode(switchPin, INPUT); // Set servo pin as input
  Serial.begin(9600);
  digitalWrite(greenLed, HIGH); // Green LED is lit when the
  // sequence is correct

  servo9.write(0);
  Serial.println("The box is unlocked!");
}

void loop() {
  if (locked == false) {
    switchVal = digitalRead(switchPin);
    if (switchVal == HIGH) {
      locked = true;
      digitalWrite(greenLed, LOW);
      digitalWrite(redLed, HIGH);
      servo9.write(90);
      Serial.println("The box is locked!");
      delay(1000);
    }
  }
  if (locked == true) {
    knockVal = analogRead(piezo); // Knock value is read by analog pin
    if (numberOfKnocks < 3 && knockVal > 0) {
      if (checkForKnock(knockVal) == true) { // Check for correct
      // number of knocks
        numberOfKnocks++;
      }
      Serial.print(3 - numberOfKnocks);
      Serial.println(" more knocks to go");
    }
  }
  if (numberOfKnocks >= 3) { // If 3 valid knocks are detected,
```

```

// the servo moves
locked = false;
servo9.write(0);
delay(20);
digitalWrite(greenLed, HIGH);
digitalWrite(redLed, LOW);
Serial.println("The box is unlocked!");
}
}
}

boolean checkForKnock(int value) { // Checks knock value
  if (value > quietKnock && value < loudKnock) { // Value needs to be
                                                    // between these

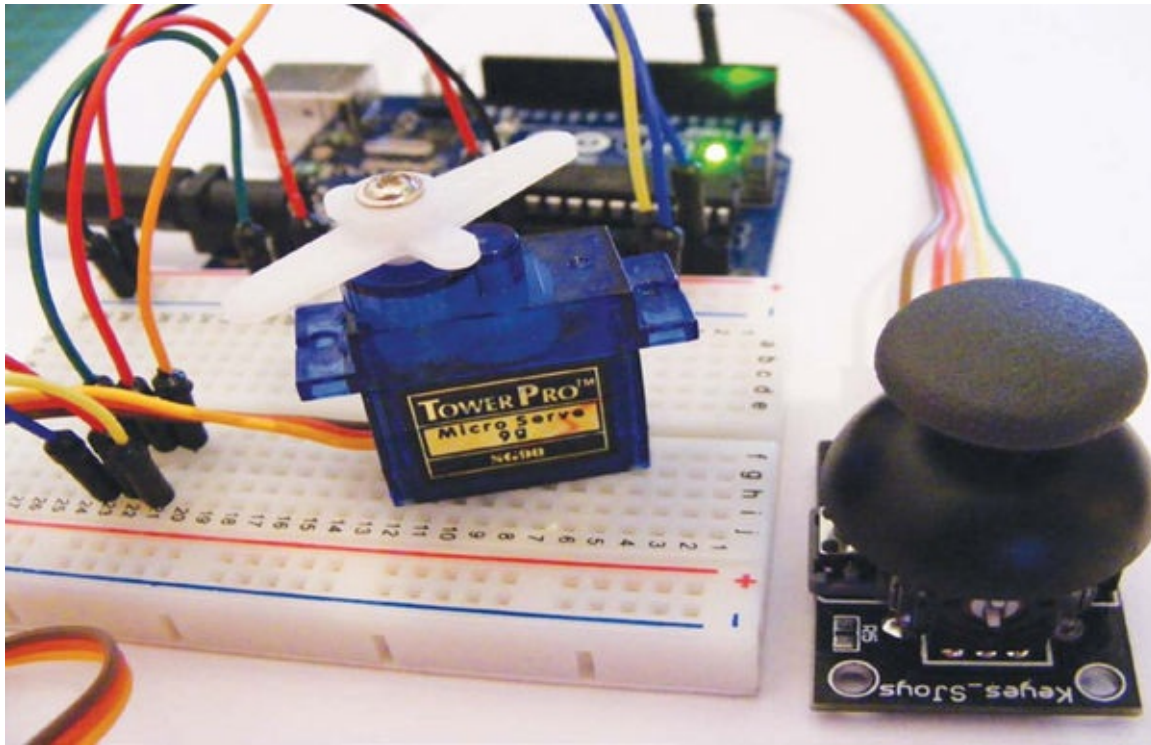
    digitalWrite(yellowLed, HIGH);
    delay(50);
    digitalWrite(yellowLed, LOW);
    Serial.print("Valid knock of value ");
    Serial.println(value);
    return true;
  }
  else { // If value is false then send this to the IDE serial
    Serial.print("Bad knock value ");
    Serial.println(value);
    return false;
  }
}
}
```

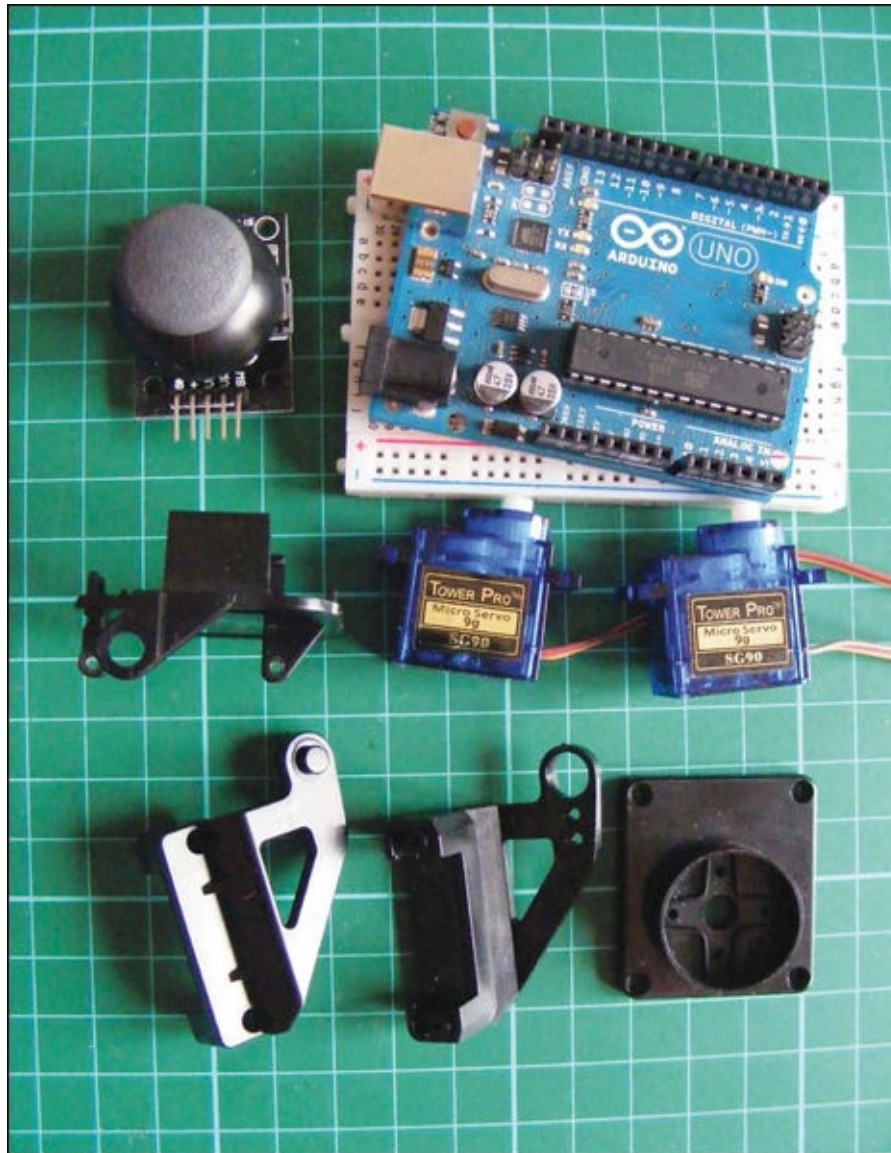
PART 3

SERVOS

PROJECT 10: JOYSTICK-CONTROLLED LASER

IN THIS PROJECT WE CREATE A JOYSTICK-CONTROLLED LASER BY CONNECTING TWO SERVOS TO A JOYSTICK AND USING THIS SETUP AS A PAN-AND-TILT CONTROLLER FOR A LASER POINTER.





PARTS REQUIRED

- Arduino
- Breadboard
- Jumper wires
- 2 Tower Pro SG90 9g servomotors
- Analog five-pin, two-axis joystick module
- Pan-and-tilt housing module

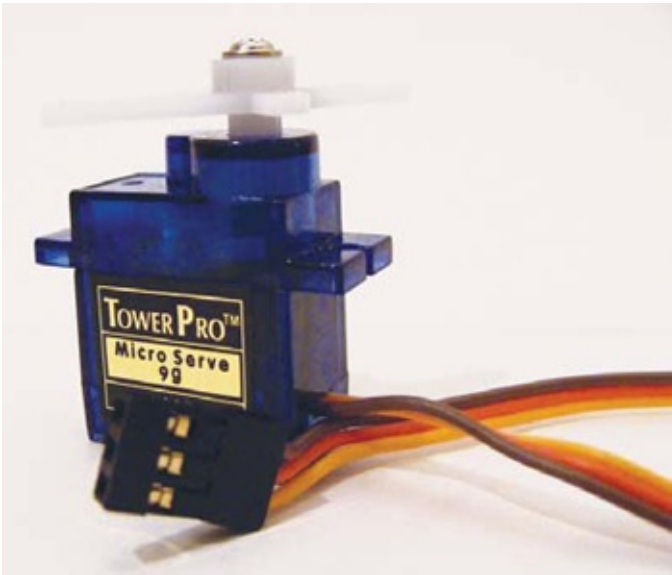
LIBRARIES REQUIRED

- Servo

HOW IT WORKS

Servos are small motors that can precisely angle their arms to positions between 0 and 180 degrees. In this project we'll place the servos into a tilt-and-pan mount. The tilt-and-pan mount is a worthy investment, as it makes it much easier to attach the laser to the servo. Here we're controlling a laser, but you could easily replace the laser with a webcam or another small device. We use two servos: one for left and right movement, and the other for up and down movement. As you might remember, servomotors have three wires, shown in [Figure 10-1](#): positive power (red), negative power or ground (black or brown), and signal (typically yellow, orange, or white).

FIGURE 10-1:
Servos have three wires.



Before we begin building, you need to know a little about how a joystick works. The joystick shown in [Figure 10-2](#) is basically two potentiometers and a button that allow us to measure the movement of the stick in two dimensions.

FIGURE 10-2:

This joystick has two potentiometers and a button for measuring movement.



Potentiometers are variable resistors and act as sensors that provide us with a voltage that varies depending on the rotation of the device around its shaft. So as you move the joystick around its center, its resistance—and therefore its output—varies. The outputs from the potentiometers are analog, so they can have a value only between 0 and 1,023 when read by the analog pin of the Arduino. This number sends a pulse to the Arduino, which in turn tells the servos how far to move. (See [Project 2](#) for more on potentiometers.)

A joystick typically has five pins: VRx (the x-axis signal), VRy (the y-axis signal), SW (a pushbutton we won't be using in this project), and GND and +5V for power.

When the x-axis of the joystick is moved to the left or right, the corresponding servo will move in that direction; when the y-axis of the joystick is moved up or down, the other servo will move up or down.

THE BUILD

1. Connect both servos' red wires to the + 5V rail, and their brown wires to GND on the breadboard.
2. Connect one of the servo's yellow signal wires directly to Arduino pin 9, and the other servo's signal wire directly to Arduino pin 10, as shown in the circuit diagram in [Figure 10-4](#).

SERVOS	ARDUINO
Red wires	+5V
Brown wires	GND
Yellow wire 1	Pin 9
Yellow wire 2	Pin 10

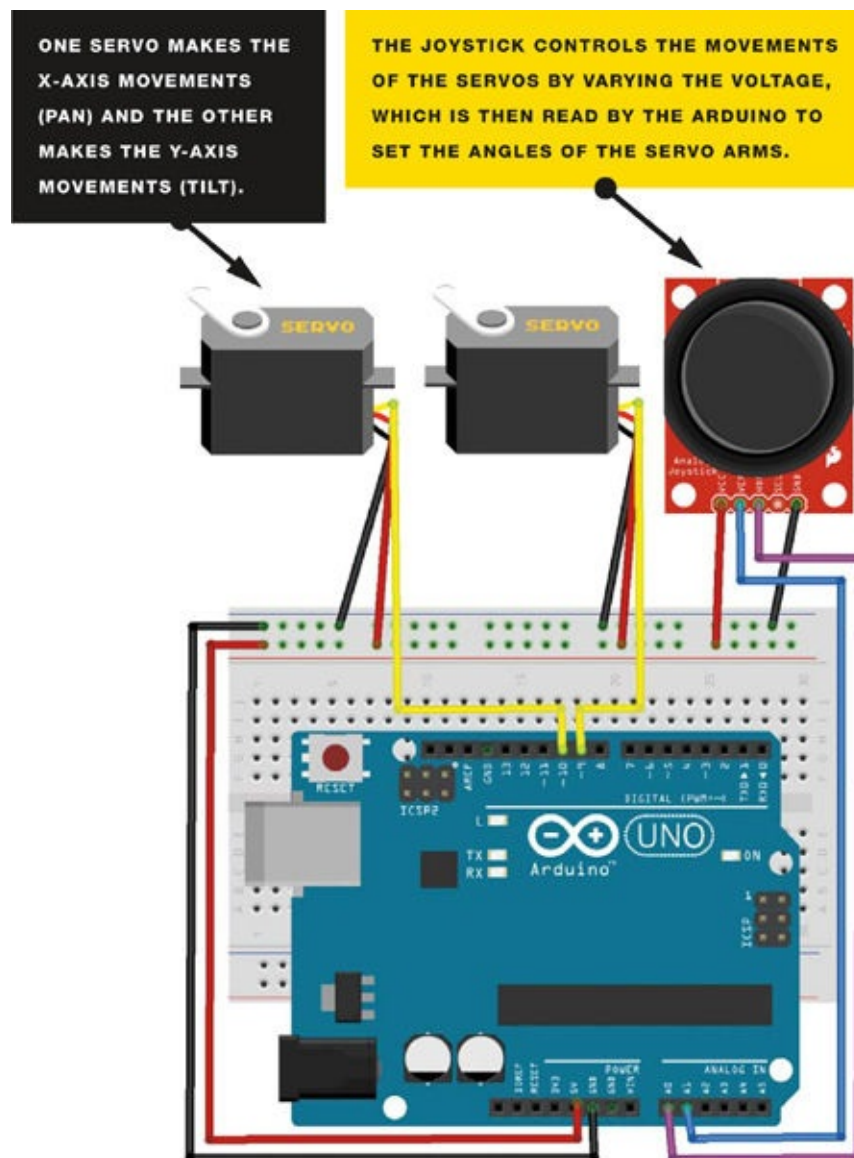
-
3. Connect the GND from the joystick module to the Arduino GND rail, and +5V to the Arduino +5V rail. Connect the VRx pin directly to Arduino A0, and the VRy pin directly to Arduino A1. Again, the SW switch connection is not used in this project.

JOYSTICK	ARDUINO
+5V	+5V
GND	GND
VRx	A0
VRy	A1
SW	Not used

4. Connect the breadboard rails to Arduino GND and +5V, and then check that your setup matches that of [Figure 10-3](#).

4. **FIGURE 10-3:**

The circuit diagram for the joystick-controlled laser. Note that the joystick in this diagram is a different brand than the one used in the project, but the connections are the same, so the instructions in the project will work fine.



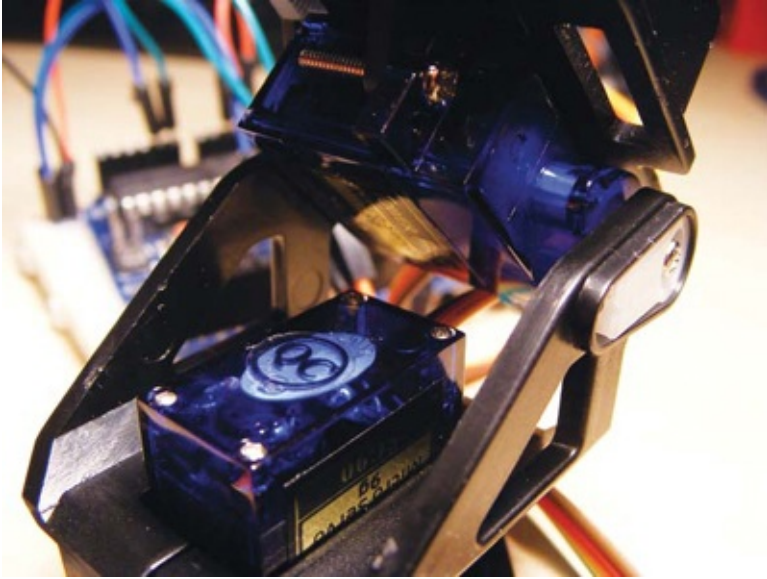
MOUNTING THE LASER

For this project, I've attached the servos to a pan-and-tilt housing module; you should be able to find this housing or a similar one for a relatively reasonable price on eBay by searching for "Arduino pan-and-tilt servo kit." You may have to assemble it yourself, but this is simple to do with the included instructions.

Attach a laser diode to the top of the module; I recommend using a glue gun for a permanent fixture, but you can use tape if you want something more temporary. Now you can control the laser using the joystick. The servos will clip into the tilt-and-pan module as shown in [Figure 10-5](#).

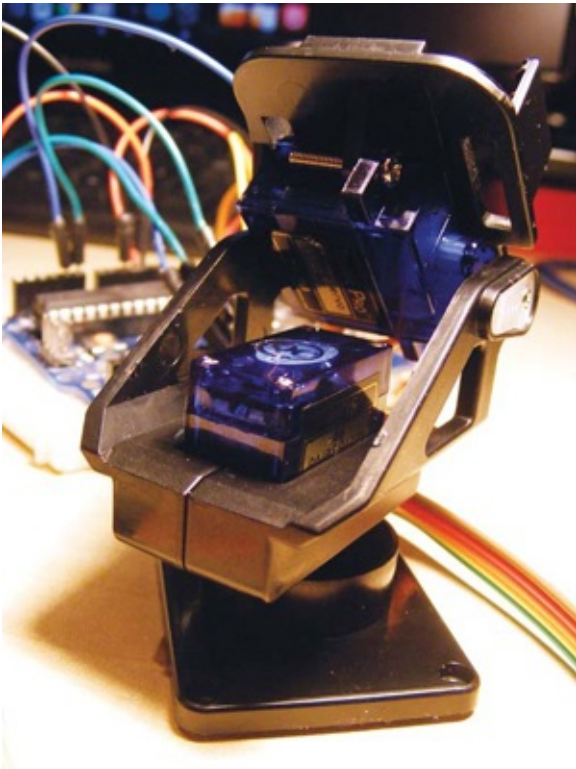
FIGURE 10-4:

Clipping the servos into the pan-and-tilt module



Moving the joystick left and right will move the x-axis servo, and moving the joystick up and down will move the y-axis servo. The complete assembly is shown in Figure 10-6.

FIGURE 10-5:
The complete assembly



THE SKETCH

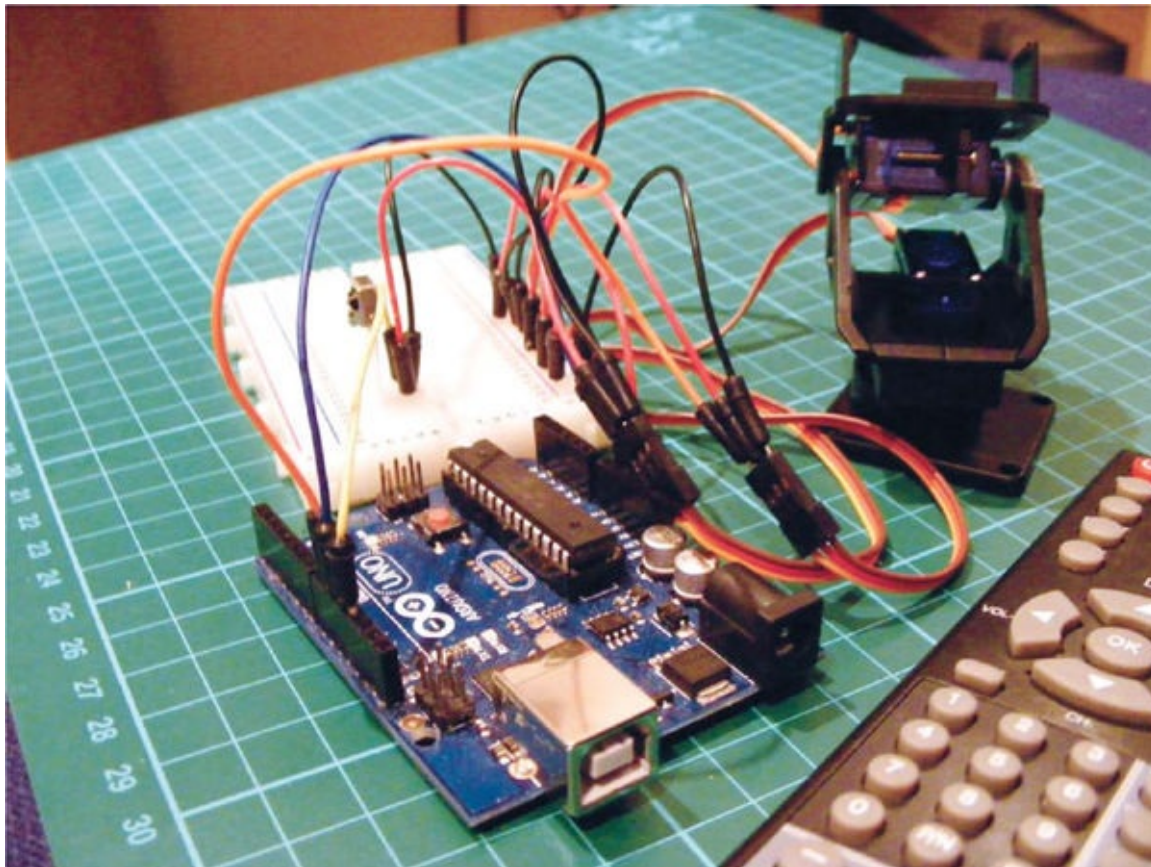
The sketch first calls on the Servo library and then defines the two servos as `tilt` and `pan`. The joystick x-axis is attached to Arduino pin A0 and the y-axis to Arduino A1, and these are our `INPUT`. The x- and y-axes are then set as variables for movement. The `tilt` servo is attached to Arduino pin 9 and `pan` is attached to Arduino pin 10, and these are our `OUTPUT`. The Arduino then reads the `INPUT` from the joystick and changes this voltage to `OUTPUT`, moving the servos according to which direction is chosen.

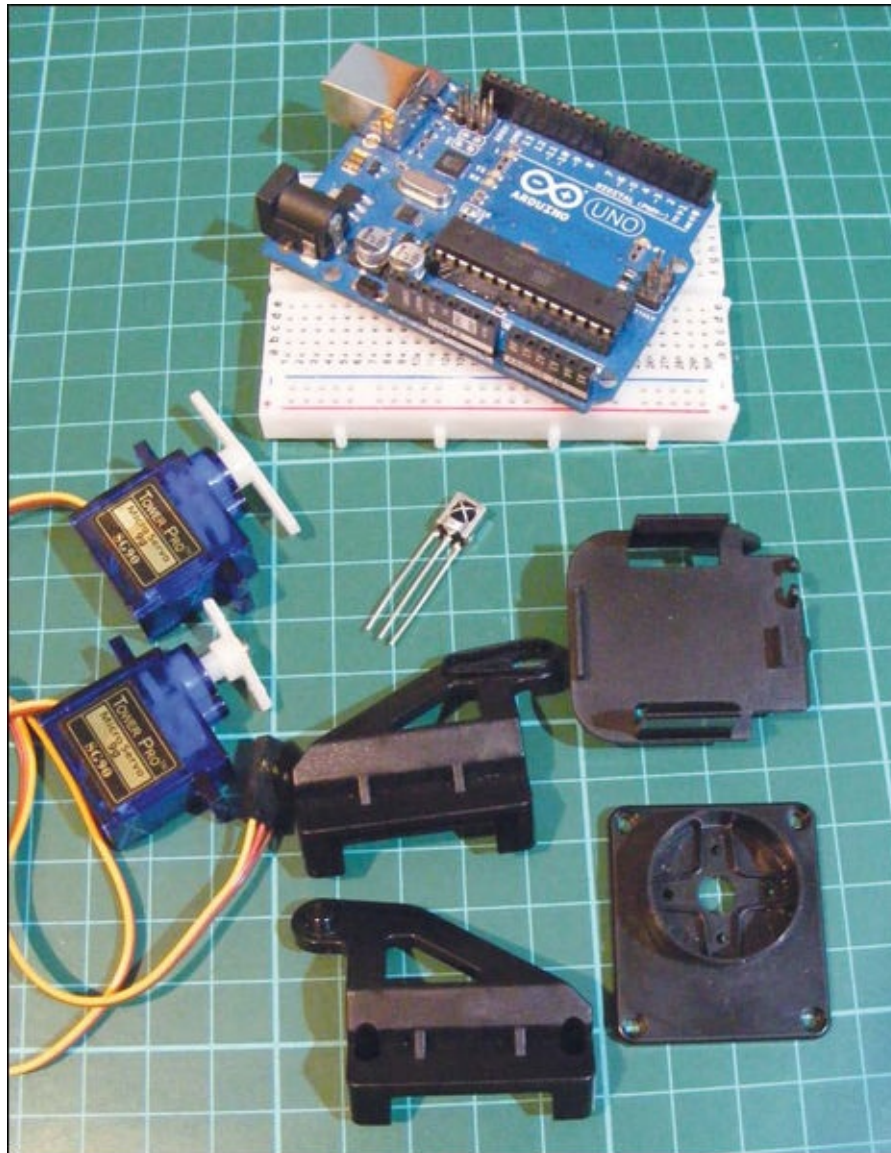
```
// Used with kind permission from http://learn.explorelabs.com/  
// Creative Commons 4.0 Share Alike (CC by SA 4.0) license  
  
#include <Servo.h>  
Servo tilt, pan; // Create servo object  
int joyX = A0; // Analog pin connected to x-axis servo  
int joyY = A1; // Analog pin connected to y-axis servo  
int x, y; // Variables to read values  
  
void setup() {  
  tilt.attach(9); // Attach tilt servo on pin 9 to the servo object  
  pan.attach(10); // Attach pan servo on pin 10 to the servo object  
}  
  
void loop() {  
  x = joyX; // Read value of x-axis (between 0 and 1023)  
  y = joyY; // Read value of y-axis (between 0 and 1023)  
  x = map(analogRead(joyX), 0, 1023, 900, 2100); // Scale it to use  
                                                    // with servo between  
                                                    // 900 to 2100  
                                                    // microseconds  
  y = map(analogRead(joyY), 0, 1023, 900, 2100);
```

```
tilt.write(x); // Set servo position according to scaled value
pan.write(y);
delay(15);     // Wait for servos to get to new position
}
```

PROJECT 11: REMOTE CONTROL SERVO

IN THIS PROJECT, WE'LL USE THE ARDUINO TO EXAMINE AND DECODE SIGNALS FROM A REMOTE CONTROL, AND THEN USE THESE CODES TO CONTROL A SERVO.





PARTS REQUIRED

- Arduino board
- Breadboard
- Jumper wires
- 38 kHz IR receiver
- Remote control
- 2 Tower Pro SG90 9g servomotors
- Pan-and-tilt housing module

LIBRARIES REQUIRED

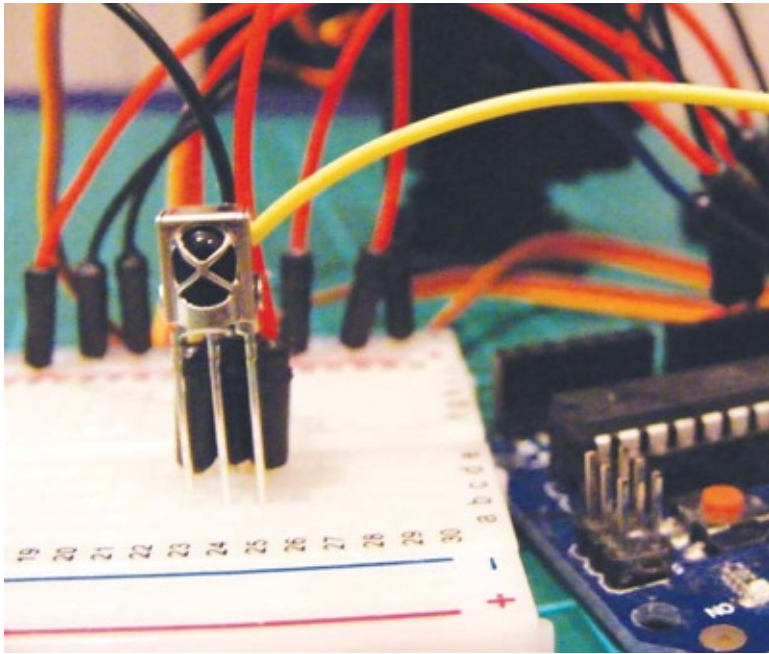
- Servo

HOW IT WORKS

First we'll decode the remote control using an IR receiver. An IR receiver has three pins: OUT, GND, and VCC (shown left to right in [Figure 11-1](#)). Check the data sheet for the receiver you bought to make sure it matches this pin layout. In rare cases you might find that your receiver's pin layout differs, but you should still be able to use the pinout to wire it up.

FIGURE 11-1:

IR receiver—from left to right, the pins are OUT, GND, and VCC



You will also need a remote control. You can use any kind of remote, including a TV remote, but it is best to use an old one that you no longer need. When you press a button on the remote, it sends out a digital value that is picked up by the receiver. This value is different for each button. We'll decode the values for each button with the Arduino and then assign them to Arduino pins in the sketch to control the output—in this case, a servo.

By personalizing the sketch with the values you decode, you can connect certain buttons to certain instructions and use your remote to control the servos. If you already built the pan-and-tilt housing model from [Project 10](#), you can reuse that here. Otherwise, flip to [Project 10](#) for instructions on setting it up.

We'll assign a button to the directional movement of the servos in the tilt-and-pan housing, so in total four buttons will control all movement: left and right for the x-axis servo, and up and down for the y-axis servo. Short button presses will move the servos in small increments, and extended presses will move the servo continuously until the maximum or minimum value is reached.

THE SETUP

1. Download the IRremote library from <http://www.nostarch.com/arduinohandbook/> and add it to your libraries folder, as shown in “[Libraries](#)” on [page 7](#).
2. Insert the IR receiver into a breadboard. Connect the OUT pin on the receiver to Arduino pin 11, GND to Arduino GND, and VCC to Arduino +5V. Again, with some versions of the 38 kHz receiver, the pin order may differ from what's shown here, so check the data sheet corresponding to your component.

IR RECEIVER	ARDUINO

OUT	Pin 11
GND	GND
VCC	+5V

3. Now upload and run the following code.

```
/* Copyright 2009 Ken Shirriff
   Used with kind permission
   http://arcfn.com
*/

#include <IRremote.h> // Use library
int receiver = 11;    // Pin connected to receiver

IRrecv irrecv(receiver);
decode_results results;
void setup() {
  Serial.begin(9600); // Show keypresses in IDE
  irrecv.enableIRIn(); // Start up receiver
}

void loop() {
  if (irrecv.decode(&results)) { // If there's an input, decode value
    Serial.println(results.value, HEX); // Display button value
                                         // on Serial Monitor in
                                         // hexadecimal format
    irrecv.resume(); // Receive next value
  }
}
```

The sketch first calls on the IRremote library, which reads from the IR receiver and sends the corresponding data to the Arduino. The IR receiver is assigned to pin 11 on the Arduino, and the sketch begins communicating with the Arduino IDE so that when a button is pressed the input is displayed in the Serial Monitor in real time. The sketch continues in a loop, looking for button presses, and shows the corresponding value to the IDE.

4. Open the Serial Monitor in your IDE.
5. Point your remote toward the receiver and try pressing different buttons. They will appear in the Serial Monitor decoded into letters and numbers in a format known as hexadecimal (HEX), as shown in [Figure 11-2](#). Try short, sharp presses to get the best results. If you press a button for too long, the Serial Monitor will show *F*s for as long as you hold the button.

5. **FIGURE 11-2:**

When a button on the remote is pressed, the HEX code for that button is displayed in the Arduino IDE Serial Monitor.



Write down the numbers that appear and the buttons they correspond to. You will need these numbers later.

Now that we've decoded the button signals from the remote control, we can use them to control two servos.

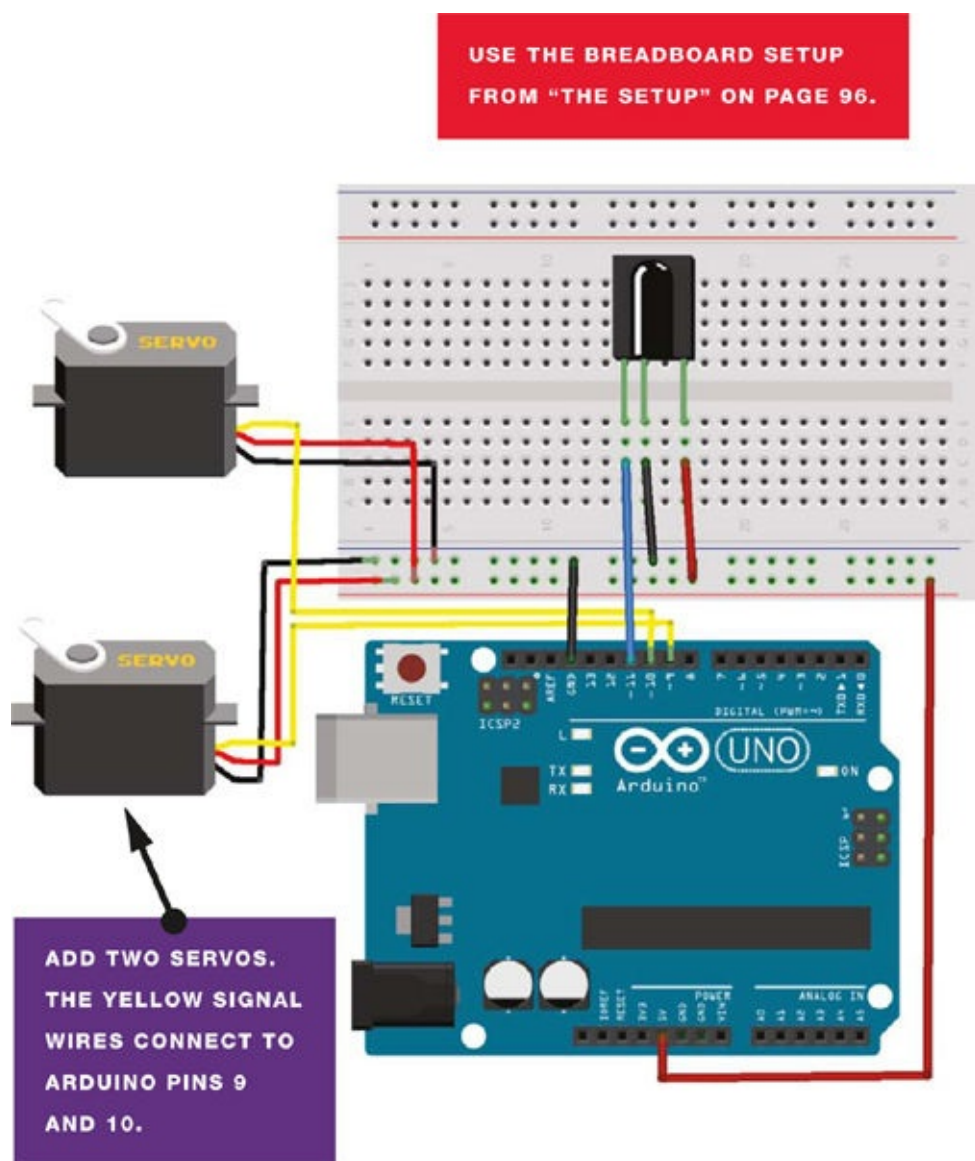
THE BUILD

1. Using your breadboard setup from step 2 on [page 96](#), with the receiver already connected, attach your servos to the Arduino by connecting the brown wire on each to GND, and the red wire to +5V. Then, connect the yellow control wire for the first servo to Arduino pin 10, and the yellow control wire for the second servo to Arduino pin 9.

SERVOS	ARDUINO
Red wires	+5V
Brown wires	GND
Yellow wire (servo 1)	Pin 10
Yellow wire (servo 2)	Pin 9

2. Remember to attach power to your breadboard.
3. Check that your setup matches the circuit diagram in [Figure 11-3](#), and then upload the code in “[The Sketch](#)” on [page 99](#).

3. **FIGURE 11-3:**
The circuit diagram for the remote control servo



THE SKETCH

Make sure you use the values that you decoded in step 3 of “[The Setup](#)” on [page 96](#) in place of the values included here when completing the sketch. When you’re changing the value in the sketch to match your own codes, keep the 0x and add your HEX code after it. For example, for the first button I decoded, the HEX code is FFA05F, which looks like this in the sketch:

```
unsigned long Value1 = 0xFFA05F;
```

In this project we’re controlling servos, but you could adapt the code slightly to remotely control anything that needs to be set to HIGH, such as an LED or piezo buzzer.

The sketch calls on the IRremote library to read from the receiver and the Servo library to move the motors. The first two buttons are assigned to the x-axis servo to move the angle to a maximum of 70 degrees for left pan or 160 degrees for right. The third and fourth buttons are assigned to the y-axis servo to control the up and down tilt movement.

If you want to adapt this to other output, change the code:

```
servo.write
```

to:

```
digitalWrite(pin, HIGH)
```

Enter the sketch as follows:

```
/* IR Library Copyright Ken Shirriff
   Used with kind permission
   http://arcfn.com
*/

#include <Servo.h>    // Include the Servo library
#include <IRremote.h> // Include the IRremote library

unsigned long Value1 = 0xFFA05F; // Change this to your value
unsigned long Value2 = 0xFF50AF; // Change this to your value
unsigned long Value3 = 0xFF807F; // Change this to your value
unsigned long Value4 = 0xFF609F; // Change this to your value

int RECV_PIN = 11;
IRrecv irrecv(RECV_PIN);
decode_results results;
Servo servo1;
Servo servo2;

void setup() {          // Set up routine
  Serial.begin(9600);
  irrecv.enableIRIn(); // Start the IR receiver
  servo1.attach(10);    // Pin connected to servo 1
  servo2.attach(9);     // Pin connected to servo 2
}

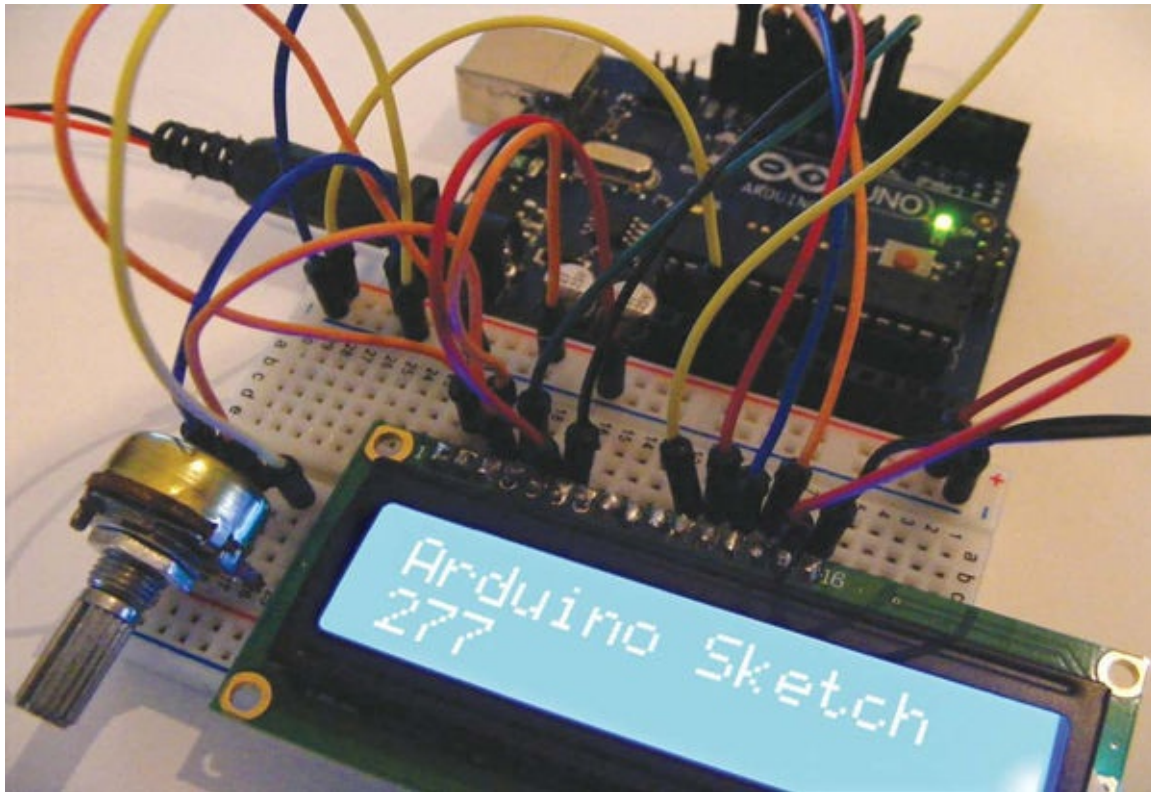
void loop() { // Loop routine runs forever
  if (irrecv.decode(&results)) {
    Serial.println(results.value, HEX);
    irrecv.resume(); // Receive the next value
  }
  if (results.value == Value1) { // If remote code matches value 1,
                                // then move the servo
    servo1.write(160);
  }
  else if (results.value == Value2) { // If remote code matches
                                     // value 2, then move the
                                     // servo, and so on
    servo1.write(70);
  }
  else if (results.value == Value3) {
    servo2.write(70);
  }
  else if (results.value == Value4) {
    servo2.write(160);
  }
}
```

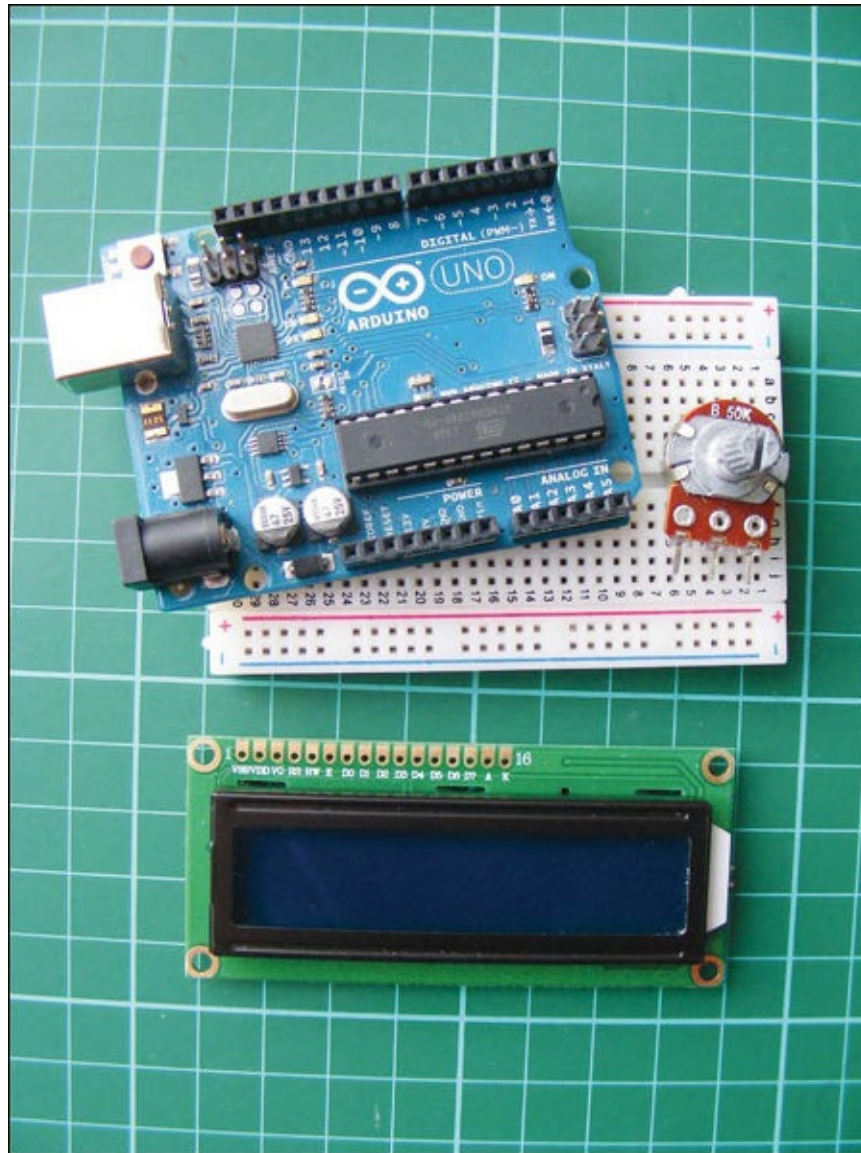
PART 4

LCDS

PROJECT 12: LCD SCREEN WRITER

NOT ONLY IS THERE SOMETHING VERY SATISFYING ABOUT HAVING AN LCD SCREEN DISPLAY YOUR OWN MESSAGES, BUT IT'S ALSO VERY USEFUL.





PARTS REQUIRED

- Arduino board
- Breadboard
- Jumper wires
- 16x2 LCD screen (Hitachi HD44780 compatible)
- 50k-ohm potentiometer

LIBRARIES REQUIRED

- LiquidCrystal

HOW IT WORKS

An LCD (liquid crystal display) screen is made of two sheets of polarizing material with a liquid crystal solution between them. Current passing through the solution creates an image or, in this case, characters. For this project, you'll need an LCD screen that's compatible with the Hitachi HD44780 driver for it to work with the Arduino—there are lots of them out there and you can usually identify them by their 16-pin interface.

We'll use the LiquidCrystal library to send characters to the LCD screen. The LiquidCrystal library maps the characters and uses the `print.lcd` commands to copy the message from the sketch to the screen.

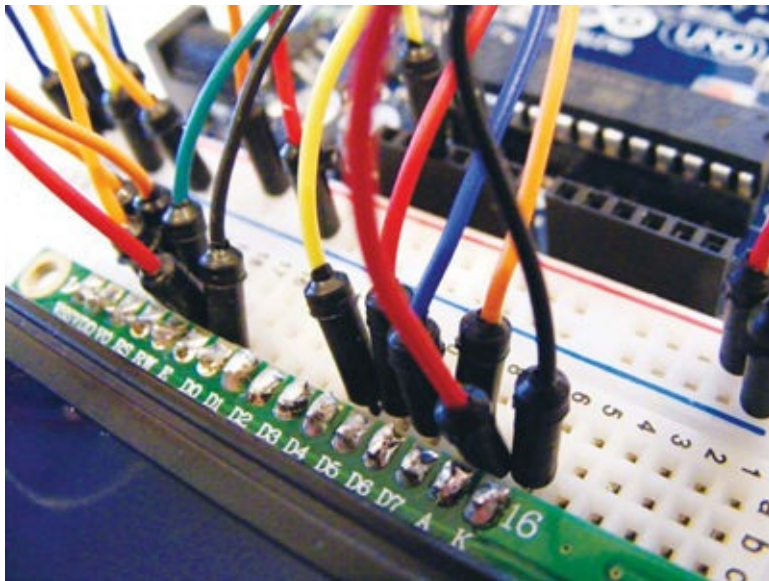
Before you start, you need to prepare your LCD screen.

PREPARING THE LCD SCREEN

The LCD screen will probably require a bit of assembly. Your screen should come with 16 holes (as shown in [Figure 12-1](#)) and a separate strip of header pins.

1. **FIGURE 12-2:**

Connections between the LCD screen and the Arduino. LCD screen pins 15 and 16 are the power and ground for the backlight of the screen.



LCD SCREEN	ARDUINO
1 VSS	GND
2 VDD	+5V
3 VO contrast	Potentiometer center pin
4 RS	Pin 7
5 R/W	GND
6 Enable	Pin 8
7 D0	Not used
8 D1	Not used
9 D2	Not used
10 D3	Not used
11 D4	Pin 9
12 D5	Pin 10
13 D6	Pin 11
14 D7	Pin 12

15 A BcL+	+5V
16 K BcL-	GND

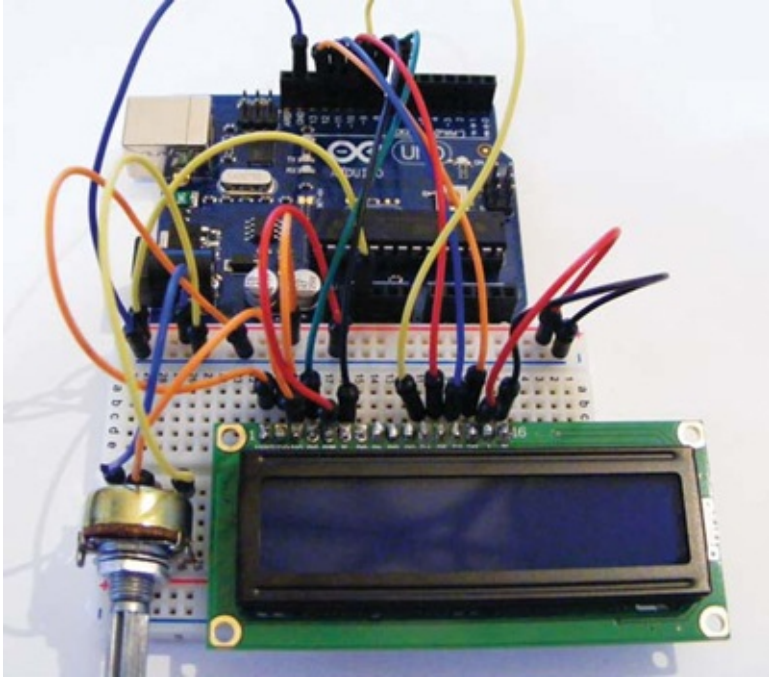
2. The center pin of the 50k-ohm potentiometer is connected to LCD pin 3 (VO). The potentiometer controls the screen contrast. Turn it until you can clearly see the characters on the screen. Now connect one of the outer pins to GND and the other to +5V.
3. Backlit LCD screens (see [Figure 12-3](#)) will have resistors built in, but if you have a non-backlit LCD screen, you should insert a 220-ohm resistor between LCD 15 and +5V. (The screen's packaging will say whether it is backlit or not.)

3. **FIGURE 12-3:**
A backlit LCD screen

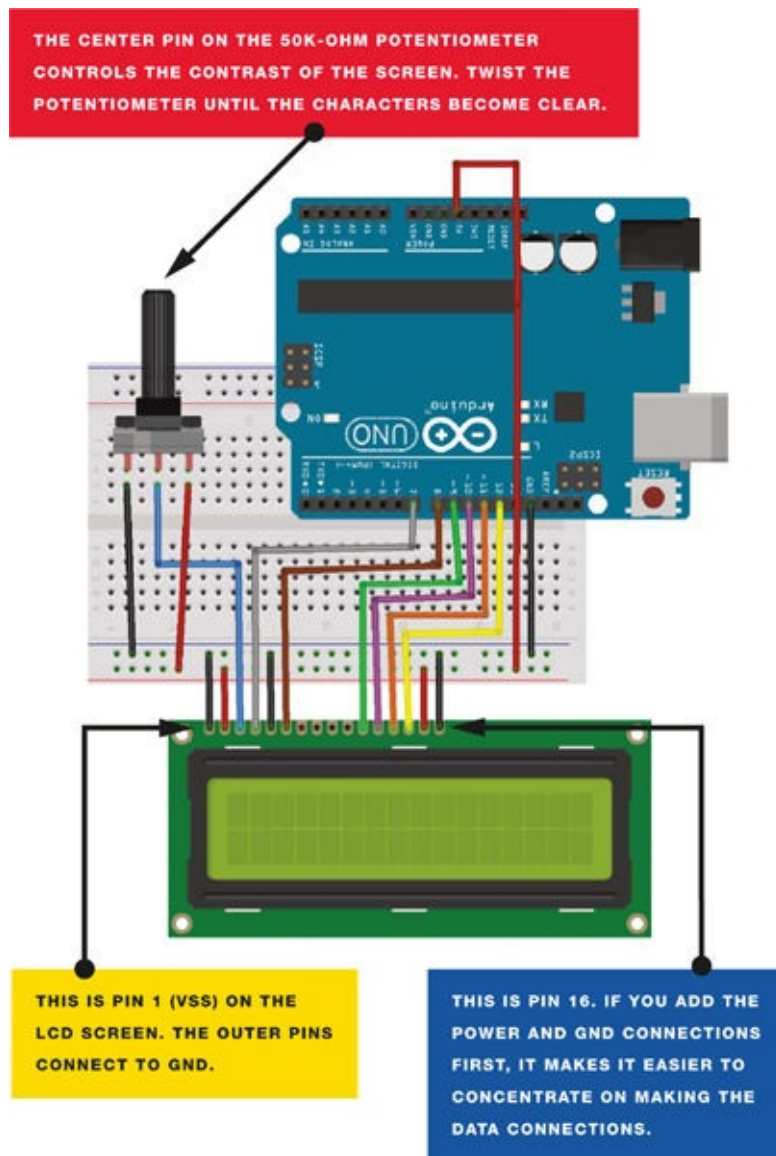


4. Your setup should look like [Figure 12-4](#). Check your work against the circuit diagram in [Figure 12-5](#), and then upload the code in “[The Sketch](#)” on [page 107](#).

4. **FIGURE 12-4:**
The complete setup



4. **FIGURE 12-5:**
The circuit diagram for the LCD screen writer



THE SKETCH

This sketch is included in your IDE examples. Load it from the IDE by going to **File** ▶ **Examples** ▶ **LiquidCrystal** and then clicking **Scroll**. The sketch uses the LiquidCrystal library that's built into the Arduino IDE to send messages from the Arduino to the LCD screen. You can change the message by replacing "Arduino Sketch" at ❷.

To use this circuit setup with the example sketches in the Arduino IDE, we also change the LCD pins in the sketch (12, 11, 5, 4, 3, 2) at ❶ to 7, 8, 9, 10, 11, 12, as these are the pins we've assigned. I've re-created the sketch here as you'll see it in the IDE, but with those changes made.

/*

```
Library originally added 18 Apr 2008 by David A. Mellis  
library modified 5 Jul 2009 by Limor Fried (http://www.ladyada.net)  
example added 9 Jul 2009 by Tom Igoe  
modified 22 Nov 2010 by Tom Igoe  
This example code is in the public domain.  
http://www.arduino.cc/en/Tutorial/LiquidCrystal
```


LiquidCrystal Library - scrollDisplayLeft() and scrollDisplayRight()

Demonstrates the use of a 16x2 LCD display. The LiquidCrystal library works with all LCD displays that are compatible with the Hitachi HD44780 driver. There are many of them out there, and you can usually tell them by the 16-pin interface.

This sketch prints "Arduino Sketch" to the LCD and uses the scrollDisplayLeft() and scrollDisplayRight() methods to scroll the text.

```
*/
```

```
// Include the library code  
#include <LiquidCrystal.h>
```

```
// Initialize the library with the numbers of the interface pins
```

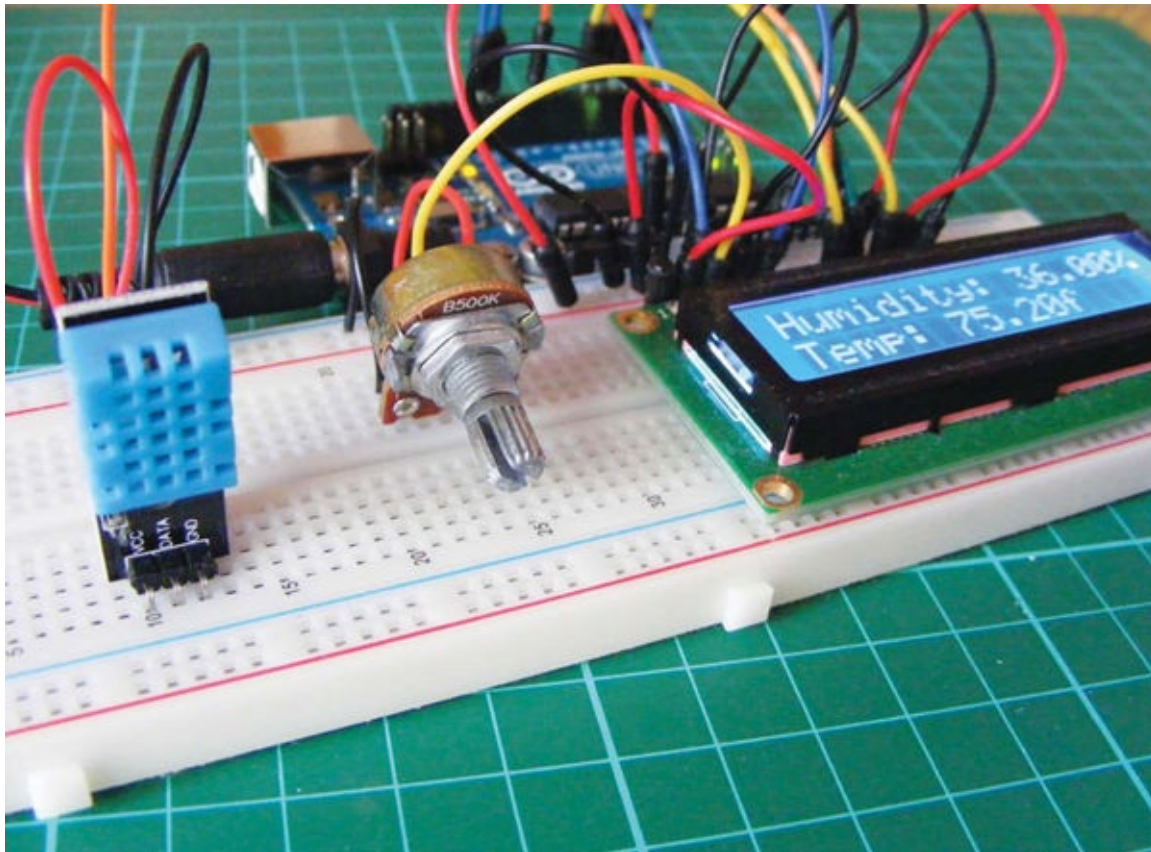
```
❶ LiquidCrystal lcd(7, 8, 9, 10, 11, 12);
```

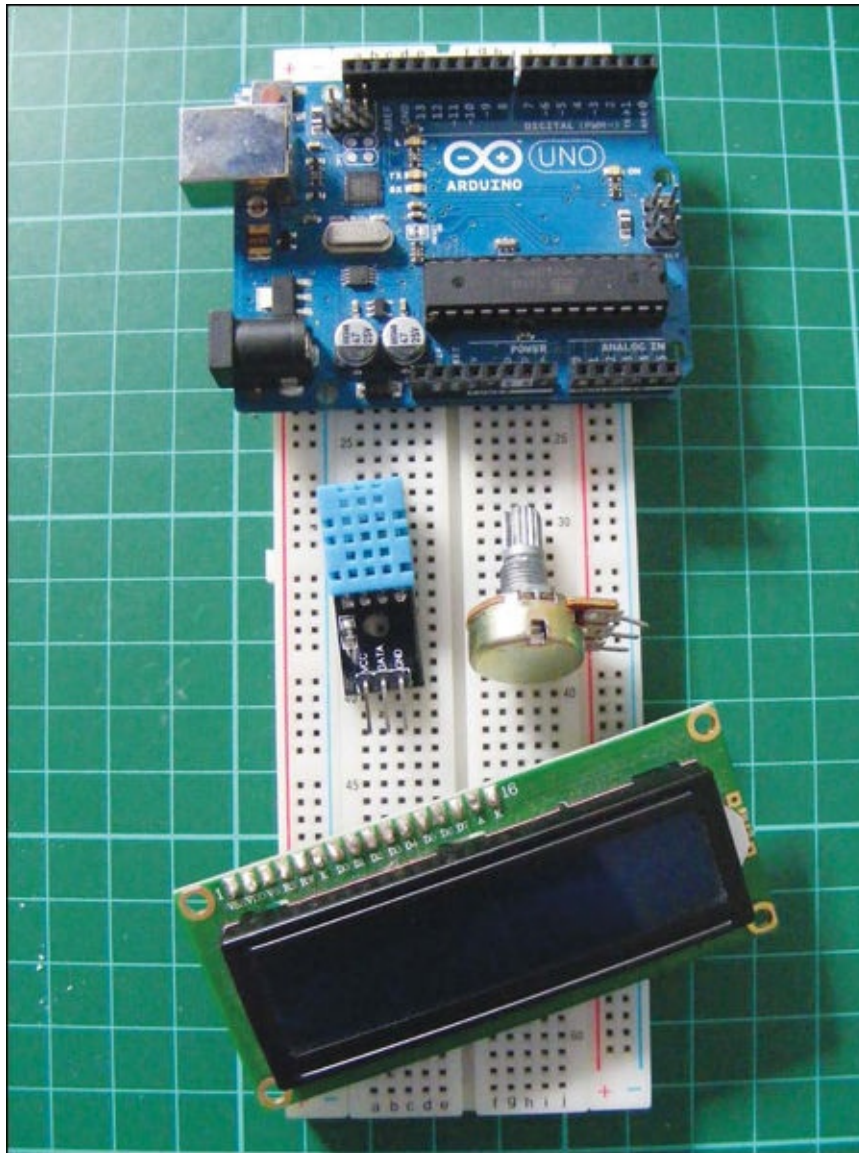
```
void setup() {  
  // Set up the LCD's number of columns and rows  
  lcd.begin(16, 2);  
  // Print a message to the LCD  
❷ lcd.print("Arduino Sketch");  
  delay(1000);  
}
```

```
void loop() {  
  // Scroll 13 positions (string length) to the left  
  // to move it offscreen left  
  for (int positionCounter = 0; positionCounter < 13;  
positionCounter++) {  
    // Scroll one position left  
    lcd.scrollDisplayLeft();  
    // Wait a bit  
    delay(150);  
  }  
  // Scroll 29 positions (string length + display length) to the right  
  // to move it offscreen right  
  for (int positionCounter = 0; positionCounter < 29;  
positionCounter++) {  
    // Scroll one position right  
    lcd.scrollDisplayRight();  
    // Wait a bit  
    delay(150);  
  }  
  // Scroll 16 positions (display length + string length) to the left  
  // to move it back to center  
  for (int positionCounter = 0; positionCounter < 16;  
positionCounter++) {  
    // Scroll one position left  
    lcd.scrollDisplayLeft();  
    // Wait a bit  
    delay(150);  
  }  
  // Delay at the end of the full loop  
  delay(1000);  
}
```

PROJECT 13: WEATHER STATION

IN THIS PROJECT YOU'LL SET UP A WEATHER STATION TO MEASURE TEMPERATURE AND HUMIDITY, AND DISPLAY THE VALUES ON AN LCD SCREEN.





PARTS REQUIRED

- Arduino board
- Breadboard
- Jumper wires
- 50k-ohm potentiometer
- 16x2 LCD screen (Hitachi HD44780 compatible)
- DHT11 humidity sensor

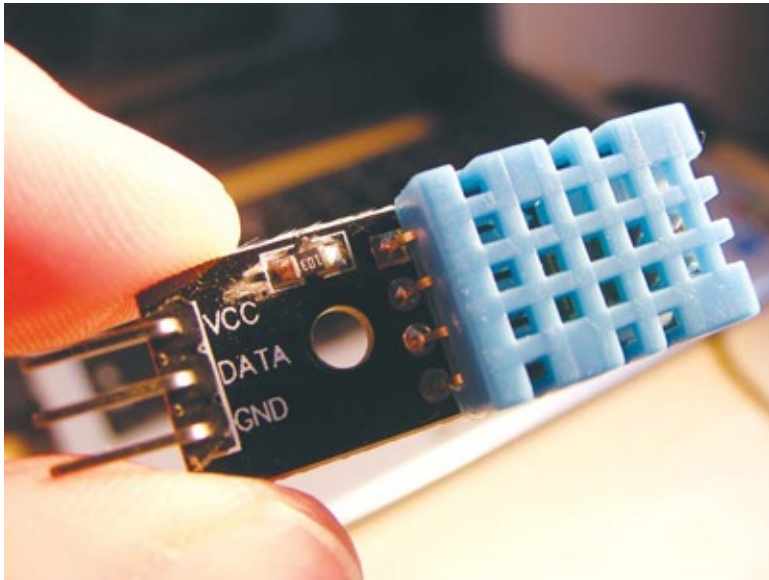
LIBRARIES REQUIRED

- LiquidCrystal
- DHT

HOW IT WORKS

The humidity sensor used in this project is the relatively cheap DHT11, shown in [Figure 13-1](#), which measures both humidity and temperature. It uses a capacitive humidity sensor and resistive-type temperature sensor to take a reading from its environment. It sends this reading to the Arduino as voltage, and the Arduino converts this to readable values displayed on the screen. For best results, you should mount your sensor on an outside wall with a decent amount of open space. You'll want to mount your LCD screen indoors or seal it carefully in a clear, waterproof bag or casing to keep it protected from the elements.

FIGURE 13-1:
The DHT11 measures both temperature and humidity.



The DHT11 comes with either four pins or three pins. The sensor shown in [Figure 13-1](#) has four pins, but you can use either version for this project, because you won't be using pin 3. Check the retailers at the beginning of the book for ideas on where to buy a DHT11.

THE BUILD

1. First, prepare the LCD screen as per the soldering instructions in [“Preparing the LCD Screen”](#) on [page 104](#). Insert the DHT11 sensor into your breadboard. The DHT11 pins are numbered 1 to 4 (or 3) from the left, when the front is facing you. Connect pin 1 to the +5V rail, connect pin 2 directly to Arduino pin 8, and connect pin 4 (or 3) to GND.

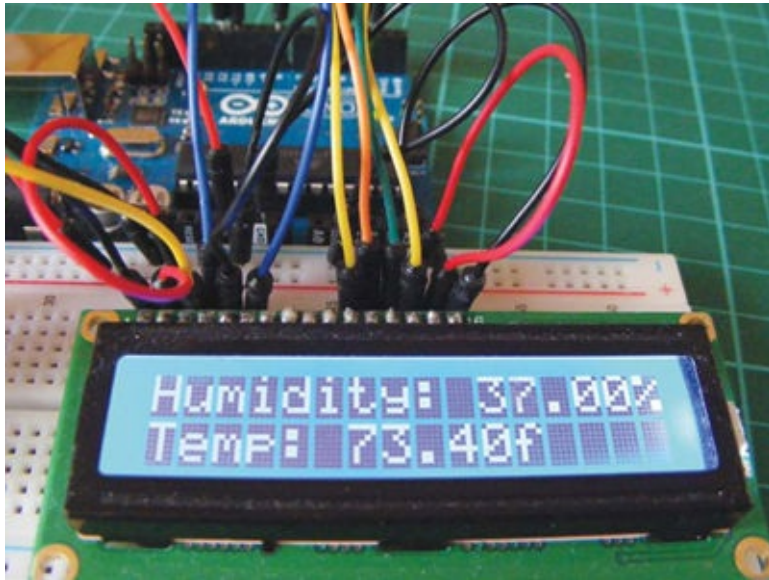
DHT11	ARDUINO
Pin 1	+5V
Pin 2	Pin 8
Pin 3	Not used
Pin 4	GND

2. Insert the LCD screen into the breadboard and connect the pins to the Arduino as shown in the following table and in [Figure 13-2](#). The GND and +5V rails will have multiple connections.

LCD SCREEN	ARDUINO
1 VSS	GND

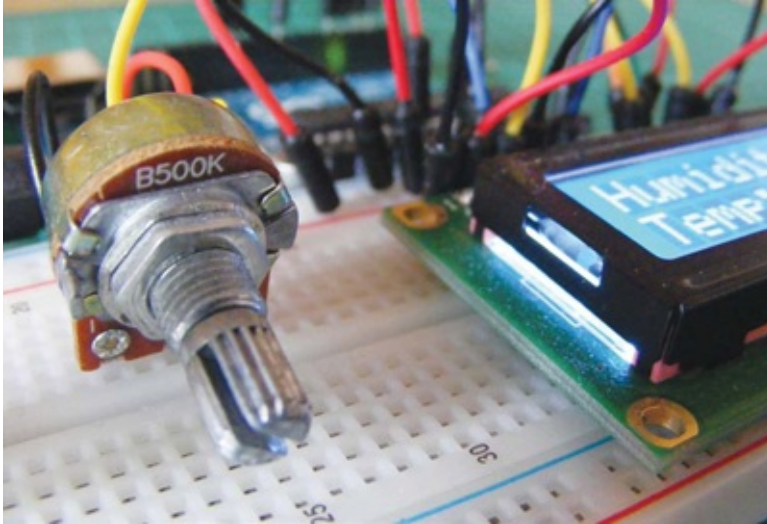
2 VDD	+5V
3 VO contrast	Potentiometer center pin
4 RS	Pin 12
5 R/W	GND
6 Enable	Pin 11
7 D0	Not used
8 D1	Not used
9 D2	Not used
10 D3	Not used
11 D4	Pin 5
12 D5	Pin 4
13 D6	Pin 3
14 D7	Pin 2
15 A BcL +	+5V
16 K BcL -	GND

2. **FIGURE 13-2:**
Inserting the LCD screen into the breadboard



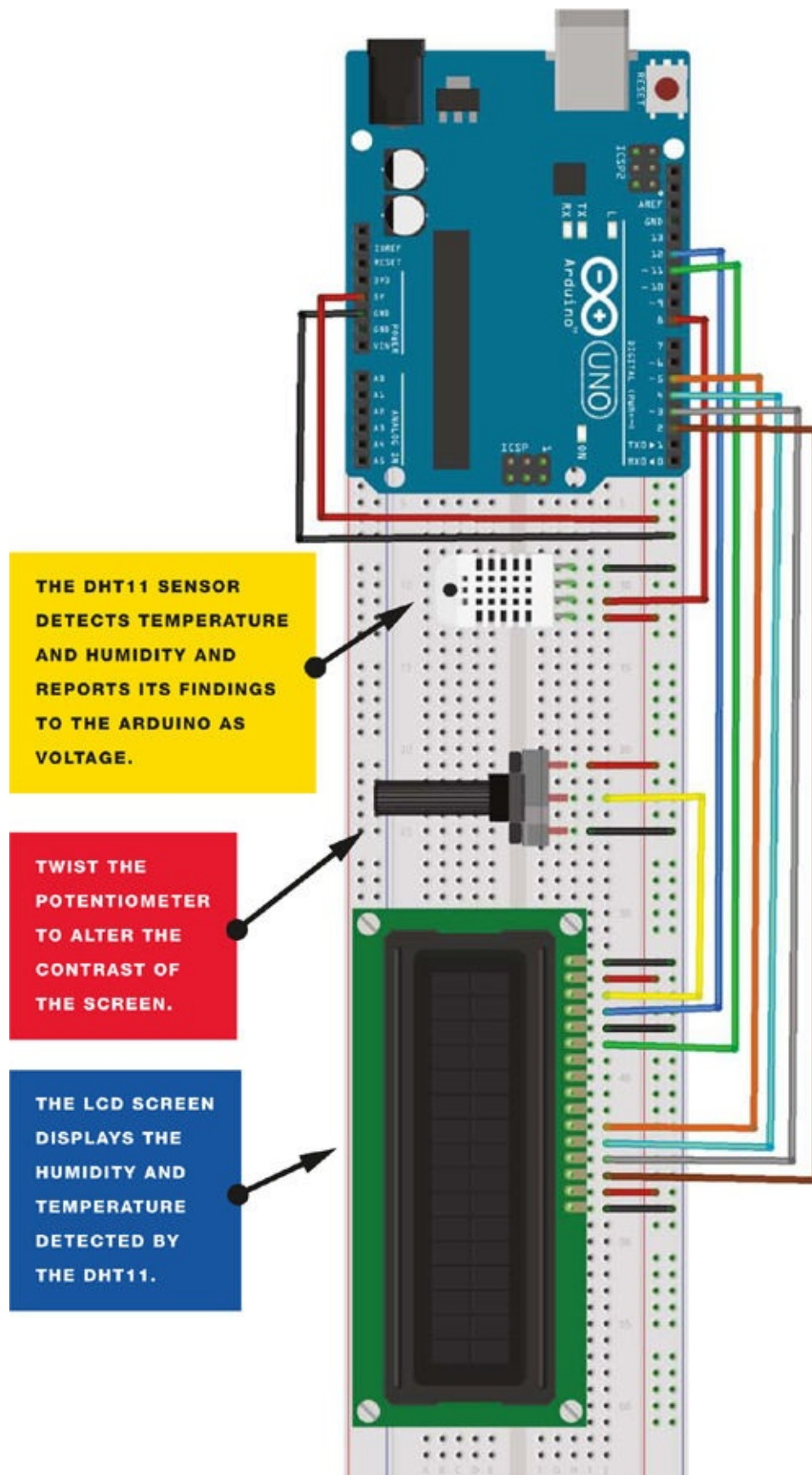
3. Insert a potentiometer into the breadboard as shown in [Figure 13-3](#) and connect the center pin to LCD pin 3. Connect one outer pin to the +5V rail and the other to the GND rail.

3. **FIGURE 13-3:**
Inserting the potentiometer into the breadboard



4. Remember to connect the power rails of the breadboard to Arduino GND and +5V. Confirm that your setup matches the circuit diagram in [Figure 13-4](#), and upload the code in “[The Sketch](#)” on [page 116](#).

4. **FIGURE 13-4:**
The circuit diagram for the weather station



THE SKETCH

This sketch uses the LiquidCrystal library, which comes with the Arduino IDE, and the DHT

library, which you will need to download and install from <http://nostarch.com/arduinohandbook/> (see “Libraries” on page 7). The DHT library controls the function of the sensor, and the LCD library displays the readings on the screen.

```
/* Example testing sketch for various DHT humidity/temperature
sensors. Written by ladyada, public domain. */

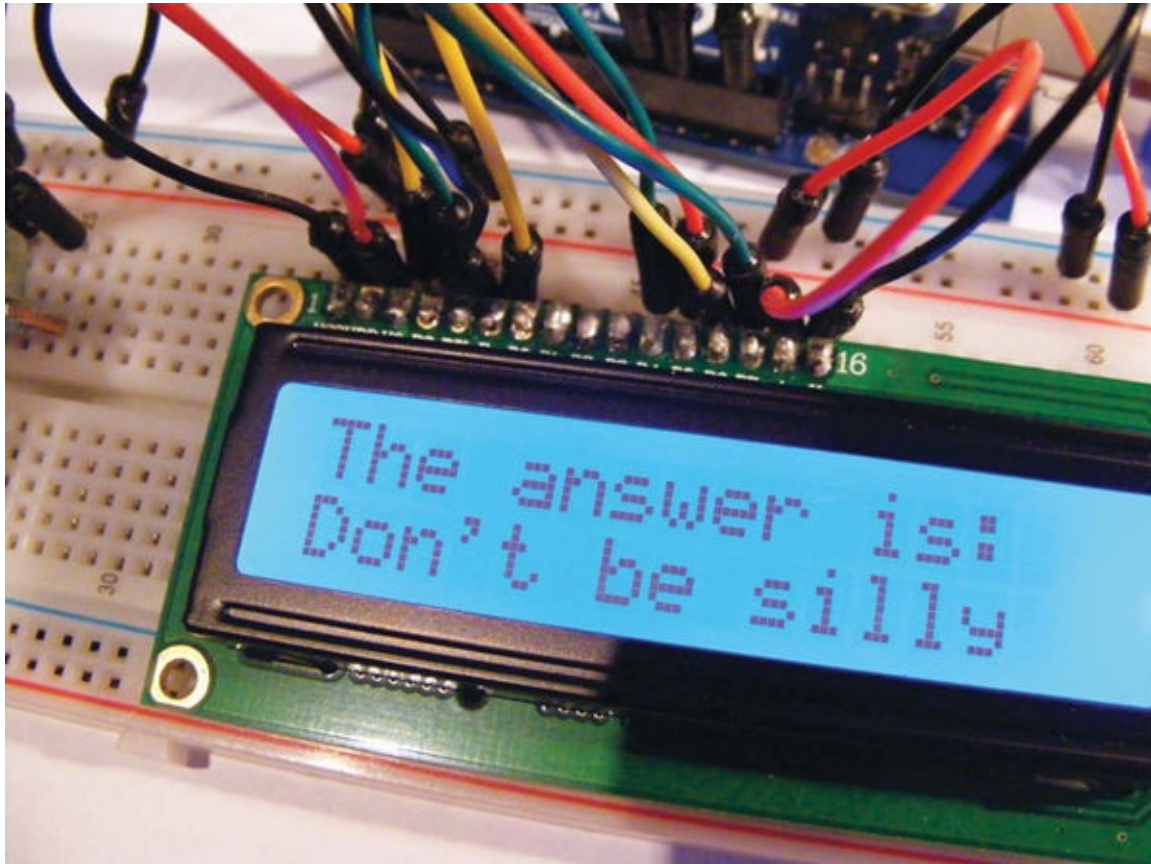
#include <LiquidCrystal.h>
#include "DHT.h" // Call the DHT library
#define DHTPIN 8 // Pin connected to DHT
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
#define DHTTYPE DHT11 // Define the type of DHT module
DHT dht(DHTPIN, DHTTYPE); // Command to the DHT.h library

void setup() {
  dht.begin(); // Start the sensor
  lcd.begin(16, 2); // LCD screen is 16 characters by 2 lines
}

void loop() {
  float h = dht.readHumidity(); // Value for humidity
  float t = dht.readTemperature(); // Value for temperature
  t = t * 9 / 5 + 32; // Change reading from Celsius to Fahrenheit
  if (isnan(t) || isnan(h)) { // Check that DHT sensor is working
    lcd.setCursor(0, 0);
    lcd.print("Failed to read from DHT"); // If DHT is not working,
                                          // display this
  } else { // Otherwise show the readings on the screen
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("Humidity: ");
    lcd.print(h);
    lcd.print("%");
    lcd.setCursor(0, 1);
    lcd.print("Temp: ");
    lcd.print(t);
    lcd.print("f");
  }
}
```

PROJECT 14: FORTUNE TELLER

IN THIS PROJECT, WE'LL CREATE AN ELECTRONIC VERSION OF A CLASSIC FORTUNE-TELLING DEVICE: THE MAGIC 8 BALL.





PARTS REQUIRED

- Arduino board
- Breadboard
- Jumper wires
- 16x2 LCD screen (Hitachi HD44780 compatible)
- Tilt ball switch
- 50k-ohm potentiometer
- 1k-ohm resistor

LIBRARIES REQUIRED

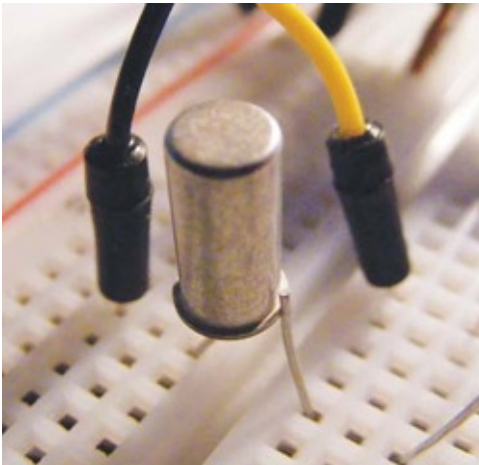
- LiquidCrystal

HOW IT WORKS

The Magic 8 Ball, a novelty toy created in the 1950s, is made of a hollow sphere in which a 20-sided die floated in alcohol. When you ask the ball a question and shake it, one side of the die floats up and displays your answer in the ball's window.

For this project, you'll use a tilt ball switch, shown in [Figure 14-1](#). The tilt ball switch is composed of a metal ball inside a metal casing that makes a connection when the switch is in an upright position. If you tilt the switch, the ball shifts and the connection is broken. There are lots of tilt switches available, and all do the same job. In this project, you'll ask a question and shake the switch. When the switch settles upright again, it connects to the Arduino, which then randomly selects a response from eight preset answers and displays it on the LCD screen.

FIGURE 14-1:
Tilt ball switch inserted in the breadboard



The potentiometer controls the contrast of the LCD screen.

THE BUILD

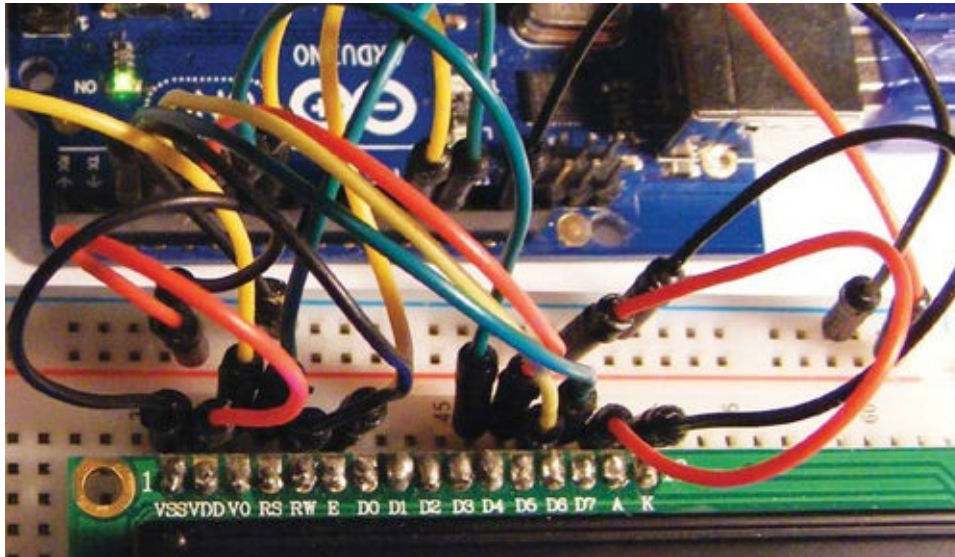
1. Prepare the LCD screen as per the soldering instructions in [“Preparing the LCD Screen”](#) on [page 104](#).
2. Place your LCD screen in the breadboard, inserting the header pins into the breadboard holes. Also place the potentiometer in the breadboard, and use the breadboard and jumper wires to connect your LCD screen, Arduino, and potentiometer.

LCD SCREEN	ARDUINO
1 VSS	GND
2 VDD	+5V
3 VO contrast	Potentiometer center pin
4 RS	Pin 12
5 R/W	GND
6 Enable	Pin 11
7 D0	Not used
8 D1	Not used
9 D2	Not used
10 D3	Not used

11 D4	Pin 5
12 D5	Pin 4
13 D6	Pin 3
14 D7	Pin 2
15 A BcL +	+5V
16 K BcL -	GND

- Remember to use a breadboard rail to make the multiple connections to the Arduino GND pin, as shown in [Figure 14-2](#).

3. **FIGURE 14-2:**
The LCD screen is connected to the Arduino.

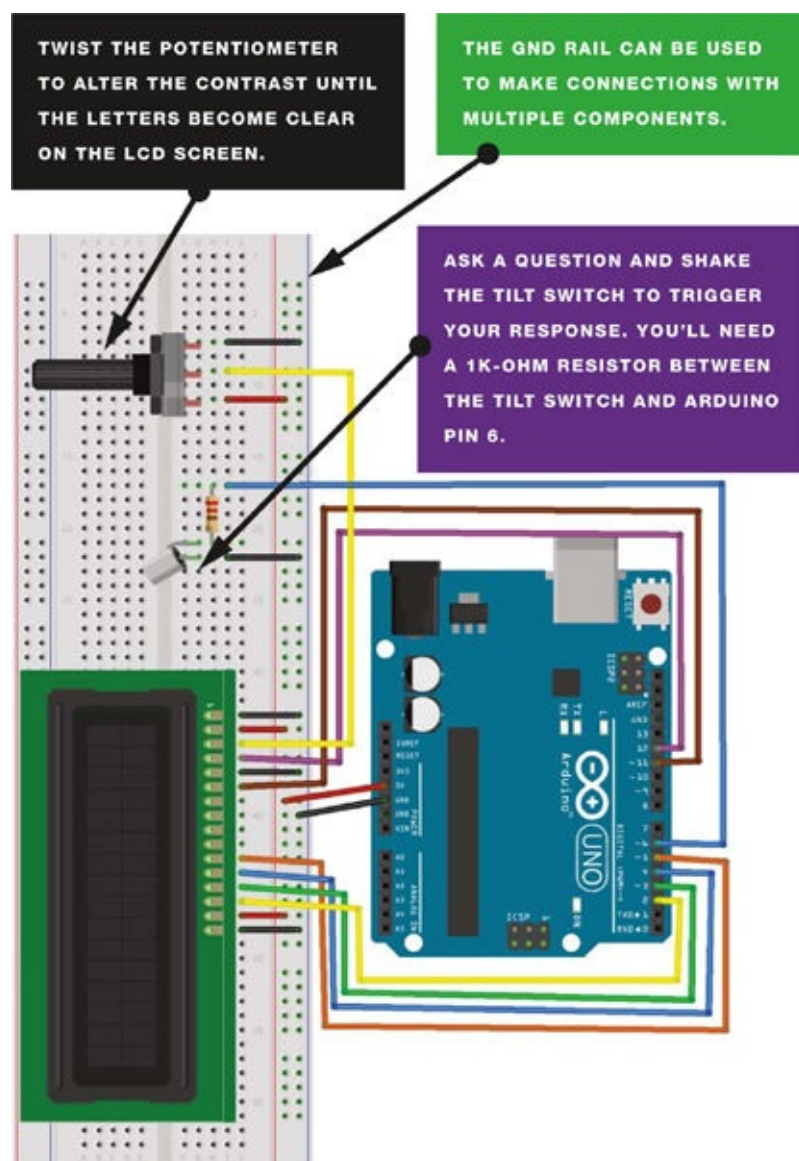


4. You should have already connected the center pin of the 10k-ohm potentiometer to LCD pin 3 (VO). Now connect one of the outer pins to GND and the other to +5V. This controls the contrast of your LCD screen.
5. Insert the tilt switch into your breadboard and attach one side to Arduino pin 6 via a 1k-ohm resistor and the other side to GND.

TILT BALL SWITCH	ARDUINO
Leg 1	Pin 6 via 1k-ohm resistor
Leg 2	GND

6. Connect your breadboard rails to the Arduino +5V and GND for power.
7. Confirm that your setup matches [Figure 14-3](#), and upload the code in “[The Sketch](#)” on [page 122](#).

7. **FIGURE 14-3:**
The circuit diagram for the fortune teller



THE SKETCH

The code for this project is fairly simple. When you switch on the Arduino, the LCD screen displays the message `Ask a Question`. Shaking the tilt switch activates the sketch, and the Arduino chooses a random answer from the eight available answers (cases 0–7).

Here's the line in the code that does this:

```
reply = random(8);
```

To add in your own responses, change the value 8 to the number of possible responses, and then add your responses (or cases) in the same style as the others:

```
case 8:  
  lcd.print("You betcha");  
  break;
```

Here's the full sketch:

```
/* Created 13 September 2012 by Scott Fitzgerald
   http://arduino.cc/starterKit
   This example code is part of the public domain
   */

#include <LiquidCrystal.h>

LiquidCrystal lcd(12, 11, 5, 4, 3, 2); // Pins attached to LCD screen

const int switchPin = 6; // Pin attached to tilt switch
int switchState = 0;
int prevSwitchState = 0;
int reply;

void setup() {
  lcd.begin(16, 2);
  pinMode(switchPin, INPUT); // Set tilt switch pin as an input
  lcd.print("FORTUNE TELLER"); // Print this on line 1
  lcd.setCursor(0, 1);
  lcd.print("Ask a Question"); // Print this on line 2
}

void loop() {
  switchState = digitalRead(switchPin); // Read tilt switch pin
  if (switchState != prevSwitchState) {
    if (switchState == LOW) { // If circuit is broken, give answer
      reply = random(8); // Reply is 1 of 8 random cases as below
      lcd.clear();
      lcd.setCursor(0, 0);
      lcd.print("The answer is: "); // Print this to the screen
      lcd.setCursor(0, 1);

      switch (reply) { // Reply will be one of the following cases
        case 0:
          lcd.print("Yes");
          break;

        case 1:
          lcd.print("Probably");
          break;

        case 2:
          lcd.print("Definitely");
          break;

        case 3:
          lcd.print("Don't be silly");
          break;

        case 4:
          lcd.print("Of course");
          break;

        case 5:
          lcd.print("Ask again");
          break;

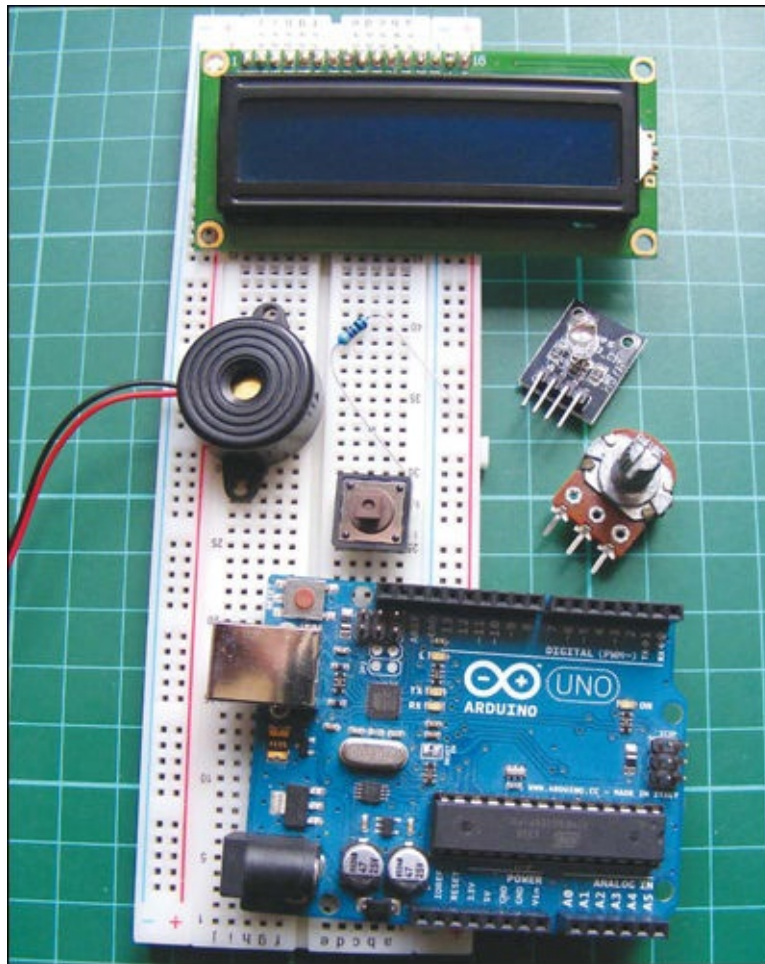
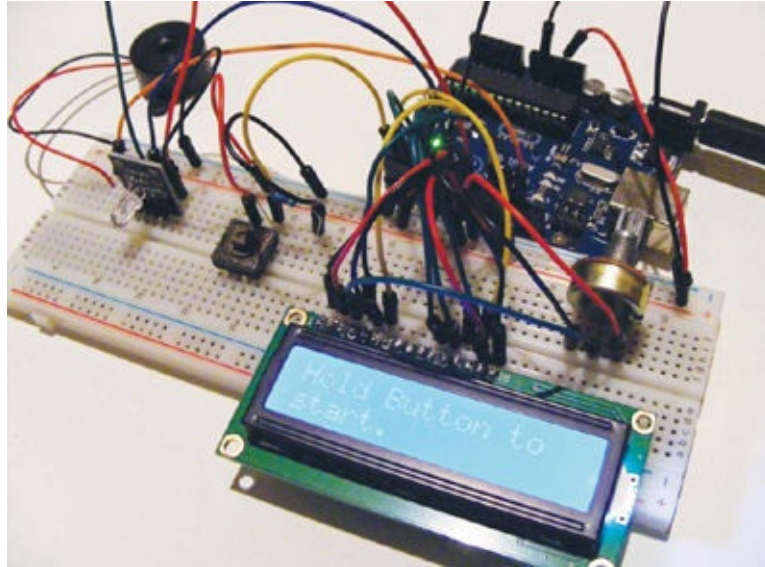
        case 6:
          lcd.print("Doubtful");
          break;

        case 7:
          lcd.print("No");
          break;
      }
    }
  }
}
```

```
        break;
    }
}
prevSwitchState = switchState; // Reset the switch
}
```

PROJECT 15: REACTION TIMER GAME

IN THIS PROJECT, LET'S COMBINE OUR LCD SCREEN WITH AN RGB LED AND A PIEZO BUZZER TO MAKE A REACTION TIMER GAME.



PARTS REQUIRED

- Arduino board
- Breadboard
- Jumper wires
- 16x2 LCD screen (Hitachi HD44780 compatible)
- RGB LED module
- Piezo buzzer
- Momentary tactile four-pin pushbutton
- 50k-ohm potentiometer
- 220-ohm resistor

LIBRARIES REQUIRED

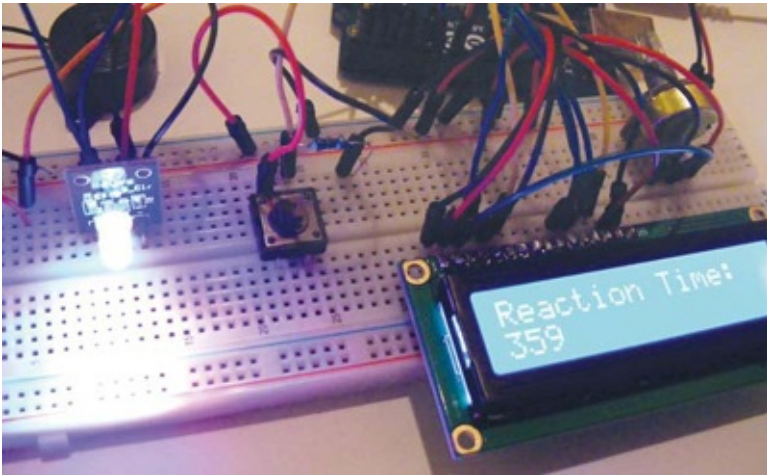
- LiquidCrystal

HOW IT WORKS

You start the game by holding down the pushbutton. The RGB LED lights up and fades through some random colors. Your aim is to react as quickly as possible when it turns red and release the pushbutton. The LCD screen shows your reaction time in milliseconds, from when the LED turned red to when you released the button (see [Figure 15-1](#)).

FIGURE 15-1:

After you release the pushbutton, your reaction time will be shown on the LED screen.



The piezo buzzer tries to distract you by making random sounds. If you release the button too soon, the LCD screen displays a message saying so, and you'll have to start over.

As its name implies, an RGB LED is actually three LEDs in one: red, green, and blue (see [Figure 15-2](#)).

FIGURE 15-2:

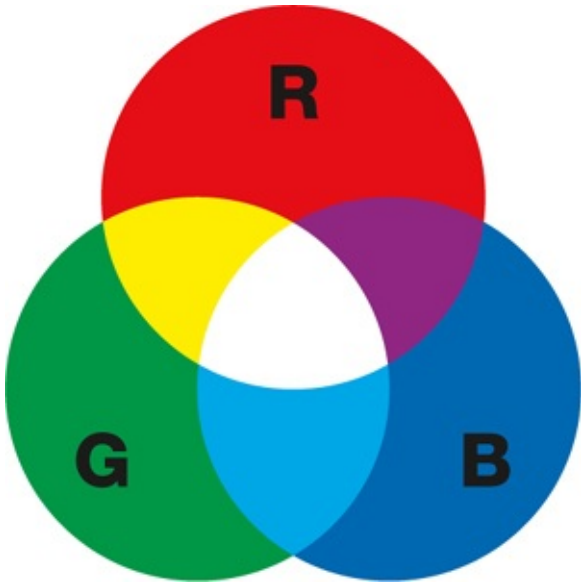
An RGB LED can be red, green, or blue.



RGB is an *additive* color model, which means that by combining the light of two or more colors we can create other colors. Red, green, and blue are the additive primary colors usually used as the base for other colors, as shown in [Figure 15-3](#).

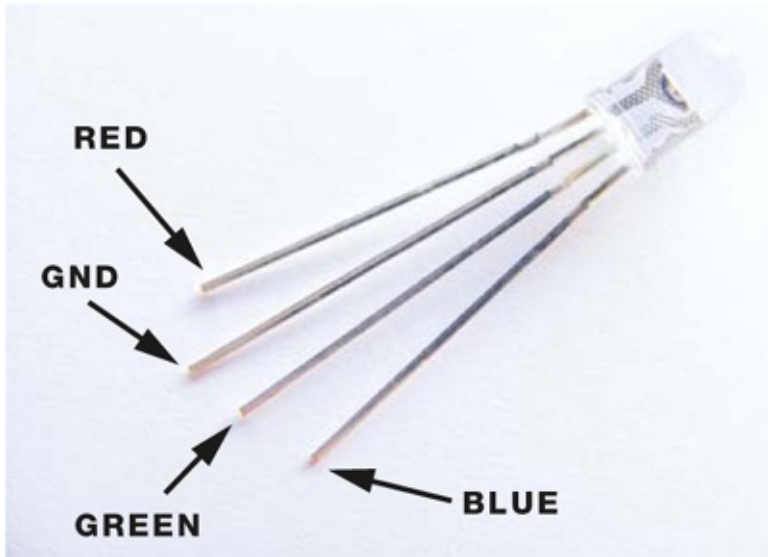
FIGURE 15-3:

The RGB color model is additive.



Let's take a look at an RGB LED in a bit more detail. [Figure 15-4](#) shows a clear common-cathode LED. Note that the LED has four legs instead of the usual two: one each for red, green, and blue, and the final one is either the cathode or anode. In this case the longest pin is the cathode, and it connects to ground (GND).

FIGURE 15-4:
An RGB LED has four legs instead of the usual two.



The RGB LED used in this project is on a module with built-in resistors, which allows us to save space on our breadboard.

THE BUILD

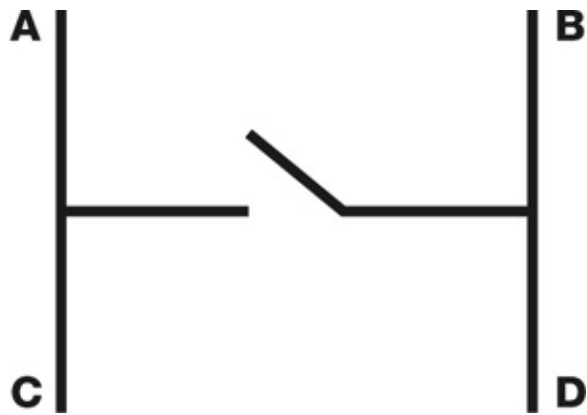
1. Prepare the LCD screen as per the soldering instructions in [“Preparing the LCD Screen”](#) on page 104.
2. Place your LCD screen in the breadboard, inserting the header pins into the breadboard holes. Also place the potentiometer in the breadboard, and use the breadboard and jumper wires to connect your LCD screen, Arduino, and potentiometer.

LCD SCREEN	ARDUINO
1 VSS	GND
2 VDD	+5V
3 VO contrast	Potentiometer center pin
4 RS	Pin 11
5 R/W	GND
6 Enable	Pin 12
7 D0	Not used
8 D1	Not used

9 D2	Not used
10 D3	Not used
11 D4	Pin 5
12 D5	Pin 4
13 D6	Pin 3
14 D7	Pin 2
15 A BcL +	+5V
16 K BcL -	GND

3. You should have already connected the center pin of the 50-kilohm potentiometer to LCD pin 3 (VO). Now connect one of the outer pins to GND and the other to +5V. This controls the contrast of your LCD screen.
4. Insert the pushbutton into the breadboard so that it straddles the break in the center. We'll label the pins as shown in [Figure 15-5](#).

4. **FIGURE 15-5:**
The pushbutton straddles the center break.



Connect pin A to ground via a 220-ohm resistor, pin C to Arduino pin 9, and pin D to +5V (see [Project 1](#) for more on how the pushbutton works).

PUSHBUTTON	ARDUINO
Pin A	GND via 220-ohm resistor
Pin C	Pin 9
Pin D	+5V

5. Insert the RGB module and connect the red pin to Arduino pin 8, green to pin 6, blue to pin 7, and + to +5V.

RGB LED	ARDUINO
Red	Pin 8
Green	Pin 6
Blue	Pin 7
+	+5V

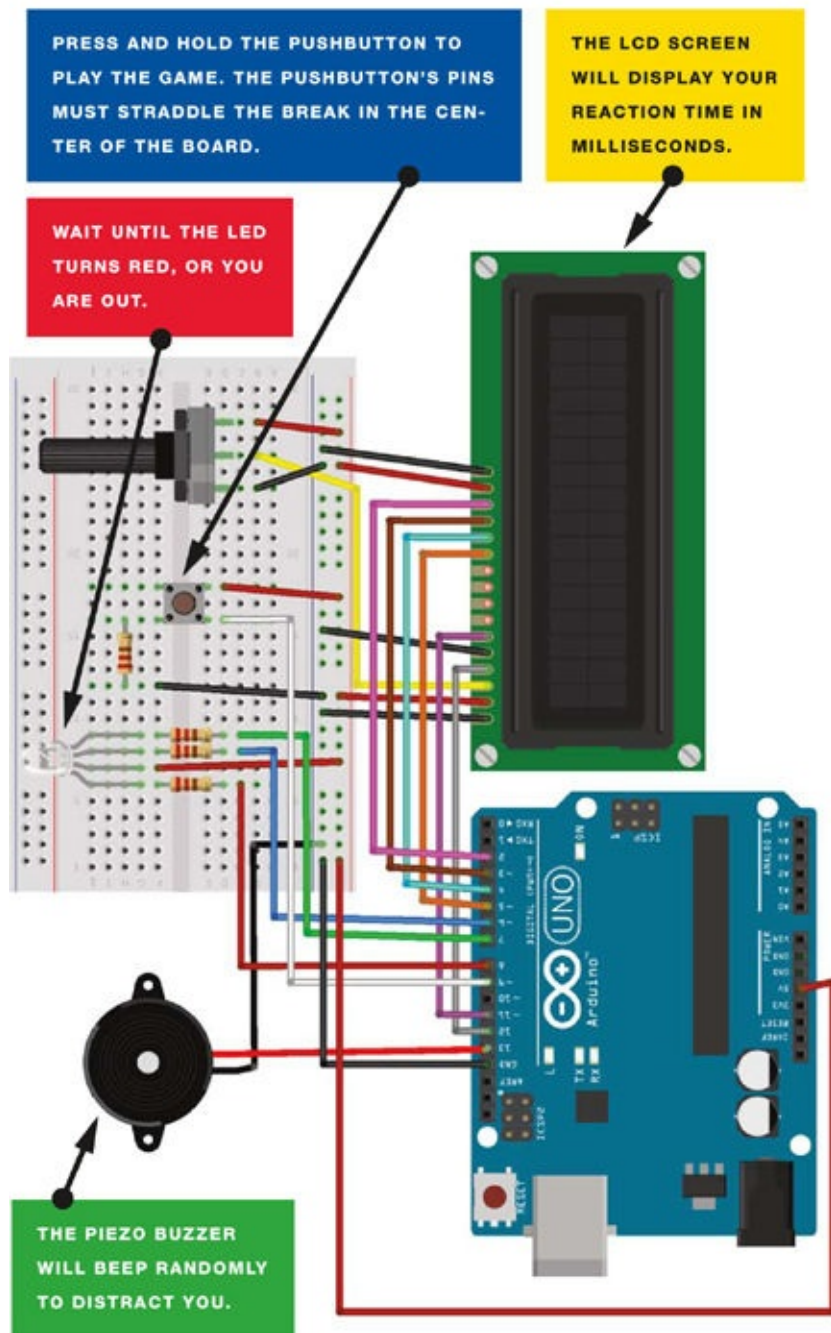
6. Connect the piezo buzzer's red wire directly to Arduino pin 13 and its black wire to GND.

PIEZO	ARDUINO
Red wire	Pin 13
Black wire	GND

7. Check your build against Figure 15-7, and then upload the code in [“The Sketch”](#) on [page 130](#)

to start playing!

7. **FIGURE 15-6:** Circuit diagram for the reaction timer game. You'll probably find that it's easier to add all the GND and +5V wires before the data wires.



THE SKETCH

When you press and hold the pushbutton, the LED flashes random colors and eventually turns red. The duration of time for which each color shows is set to random, as is the duration of the pauses between lights. This means you can't learn the sequence of the colors and predict when the LED might turn red.

You can make the game more difficult by increasing the duration of the intervals in the following line of the sketch:

```
PSE = random(500, 1200);
```

The full sketch is as follows:

```
// Created by Steven De Lannoy and reproduced with kind permission
// http://www.wingbike.nl
// Used an RGB LED with a common anode (3 cathodes: R, G, B)
#include <LiquidCrystal.h>
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
int LEDR = 8; // Pin connected to red LED
int LEDB = 7; // Pin connected to blue LED
int LEDGr = 6; // Pin connected to green LED
int Button = 9; // Pin connected to pushbutton
int COLOR; // Variable color
int Beep;
int PSE; // Variable pause
int TME; // Time
int RTME = 0; // Reaction time

void setup() {
  lcd.begin(16, 2);
  pinMode(LEDR, OUTPUT); // Set LED pins as output
  pinMode(LEDB, OUTPUT);
  pinMode(LEDGr, OUTPUT);
  pinMode(Button, INPUT); // Set pushbutton as input
  digitalWrite(LEDR, LOW); // Switch on all LED colors
  digitalWrite(LEDB, LOW);
  digitalWrite(LEDGr, LOW);
}

void loop() {
  lcd.clear(); // Clear screen
  lcd.print("Hold Button to"); // Display message on LCD screen
  lcd.setCursor(0, 1); // Move to second line
  lcd.print("start.");
  while (digitalRead(Button) == LOW) { // Test does not start until
                                        // button is pushed (and held)

    tone(13, 1200, 30);
    delay(1400);
    noTone(13);
  }

  lcd.clear();
  digitalWrite(LEDR, HIGH); // Switch off start light
  digitalWrite(LEDB, HIGH);
  digitalWrite(LEDGr, HIGH);
  randomSeed(analogRead(0)); // Random noise from pin 0
  COLOR = random(1, 4); // Generate random color
  PSE = random(500, 1200); // Set random pause duration between lights
  // Repeat this loop while color is green or blue AND pushbutton
  // is held
  while (COLOR != 1 && digitalRead(Button) == HIGH) {
    digitalWrite(LEDGr, HIGH);
    digitalWrite(LEDB, HIGH);
    delay(PSE);
    randomSeed(analogRead(0));
    Beep = random(1, 4); // Select random beep from buzzer
                        // (buzzer beeps 1 in 3 times)
    PSE = random(750, 1200); // Select random pause duration between
                            // lights (to increase surprise effect)

    if (Beep == 1) {
      tone(13, 1600, 350);
      delay(750);
      noTone(13);
    }
  }
}
```

```

if (COLOR == 2) {
    digitalWrite(LEDGr, LOW);
}
if (COLOR == 3) {
    digitalWrite(LEDB, LOW);
}
delay(PSE);
randomSeed(analogRead(0));
COLOR = random(1, 4); // Select random color
}
// Execute this loop if color is red
if (COLOR == 1 && digitalRead(Button) == HIGH) {
    digitalWrite(LEDGr, HIGH);
    digitalWrite(LEDB, HIGH);
    delay(PSE);
    TME = millis(); // Record time since program has started
    digitalWrite(LEDGr, LOW);
    while (digitalRead(Button) == HIGH) { // Runs until button is
        // released, recording the
        // reaction time

        delay(1);
    }
    lcd.display();
    RTME = millis() - TME; // Reaction time in ms
    lcd.print("Reaction Time:"); // Display on LCD screen
    lcd.setCursor(0, 1);
    lcd.print(RTME);
}

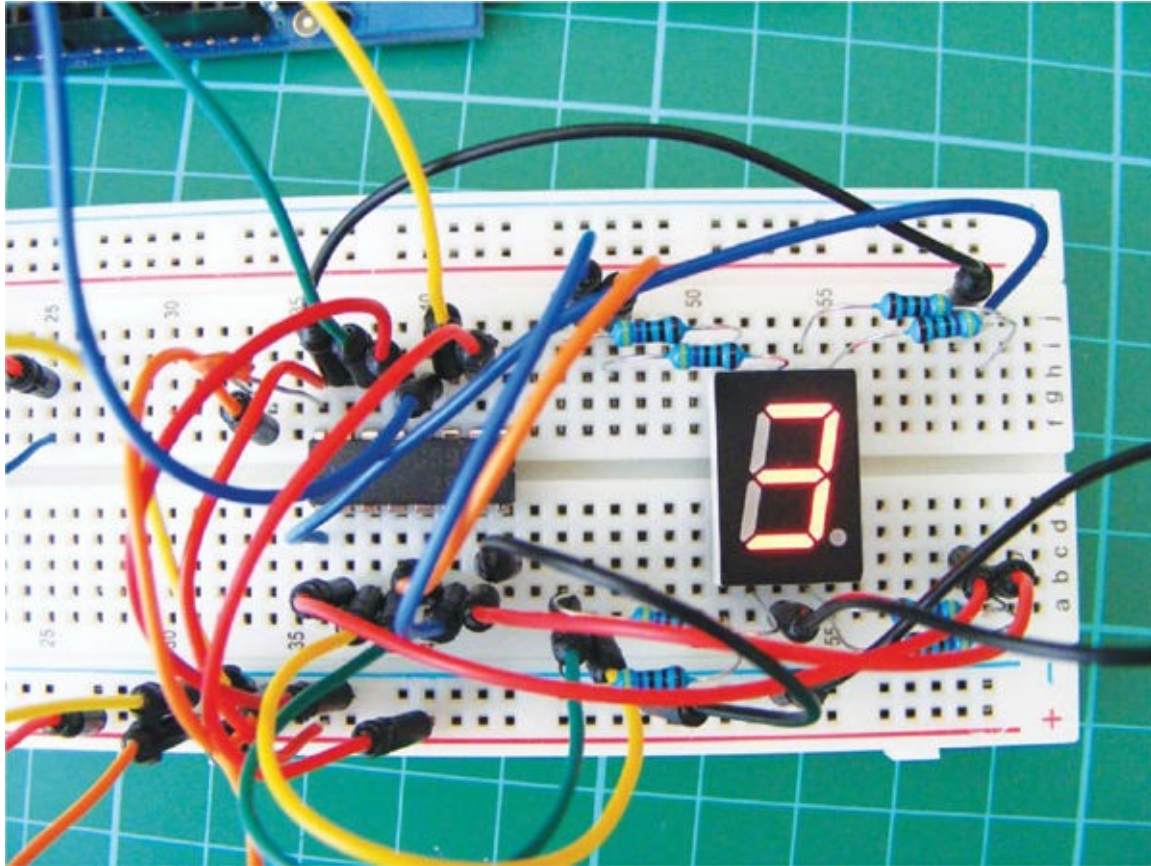
// Execute if color is NOT red but the pushbutton is released
if (COLOR != 1) {
    lcd.print("Released too");
    lcd.setCursor(0, 1); // Move to second line
    lcd.print("soon!!!");
    tone(13, 3000, 1500);
    delay(500);
    noTone(13);
}
// Test does not restart until the button is pushed once
while (digitalRead(Button) == LOW) {
    delay(10);
}
digitalWrite(LEDGr, LOW); // Reset all lights to begin again
digitalWrite(LEDB, LOW);
digitalWrite(LEDGr, LOW);
lcd.clear();
lcd.print("Hold Button to");
lcd.setCursor(0, 1);
lcd.print("start.");
int Time = 0;
delay(1000);
}

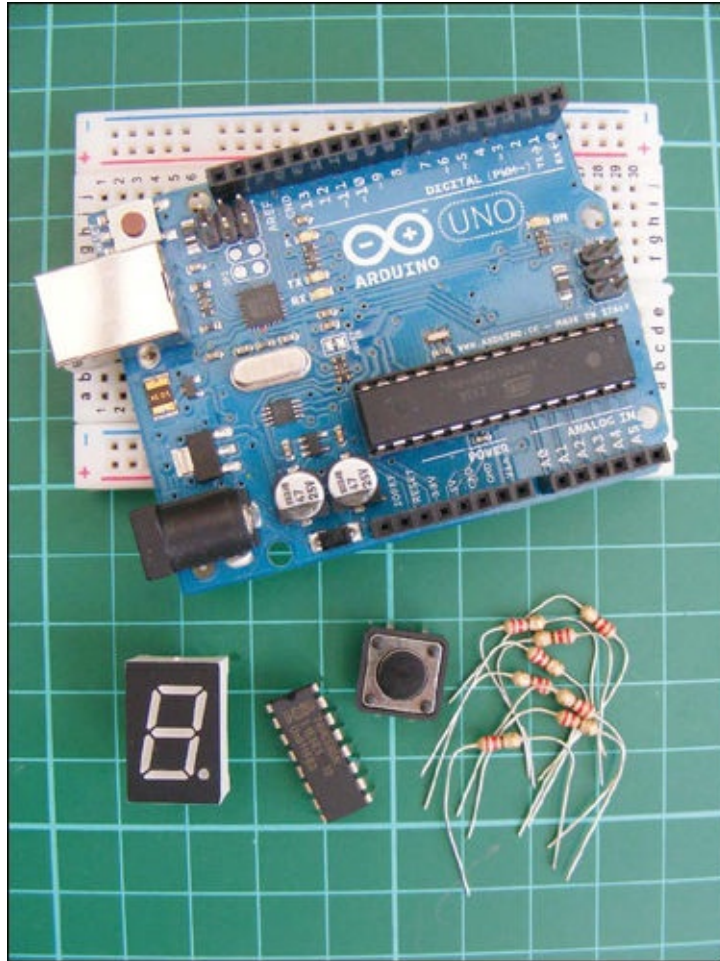
```

PART 5
NUMERIC COUNTERS

PROJECT 16: ELECTRONIC DIE

BOARD GAMES ARE PERILOUS ENOUGH WITHOUT ARGUMENTS OVER NUMBER READINGS FROM FALLEN OR LOST DICE. THE PERFECT SOLUTION: AN ELECTRONIC DIE.





PARTS REQUIRED

- Arduino board
- Breadboard
- Jumper wires
- 8 220-ohm resistors
- Seven-segment LED display
- 74HC595 shift register
- Momentary tactile four-pin pushbutton

HOW IT WORKS

In this project we'll create a die using a seven-segment LED display. When the pushbutton is pressed, a pulse is sent to the Arduino, and the LED “shakes” and displays a random digit between 1 and 6.

This project uses a 74HC595 *shift register*, a small integrated circuit (IC) and sequential logic counter that allows the Arduino to make more connections than it usually can with the pins it has, by “shifting” and storing data. The shift register has 16 pins; at one end you'll find a dot or semicircle, which marks pin 1 on the left. The pins are then numbered counterclockwise from

here. [Figure 16-1](#) shows the pinout, and [Table 16-1](#) describes the function of each pin.

FIGURE 16-1:
Pinout of the 74HC595 shift register

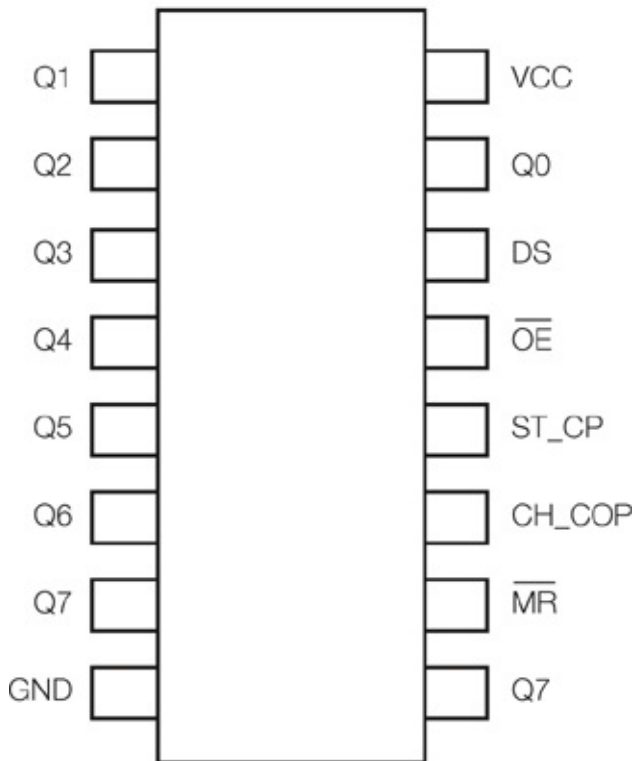


TABLE 16-1:
74HC595 shift register pins

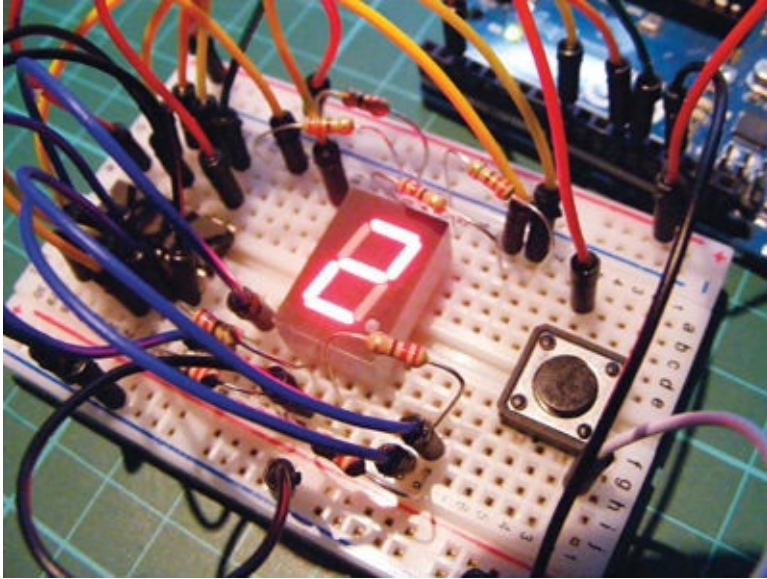
SHIFT REGISTER PINS	CONNECTIONS	PIN FUNCTION
Pins 1–7, 15	Q0–Q7	Output pins
Pin 8	GND	Ground, VSS
Pin 9	Q7	Serial out
Pin 10	MR	Master Reclear, active low
Pin 11	SH_CP	Shift register clock pin (CLOCK pin)
Pin 12	ST_CP	Storage register clock pin (LATCH pin)
Pin 13	OE	Output Enable, active low
Pin 14	DS	Serial data input (DATA pin)
Pin 16	VCC	Positive power

The wire attached to Arduino pin 2 is connected to our pushbutton and, when pressed, will create a pulse. To use the die, push the button to make the digit on the die shake and display a random digit.

THE BUILD

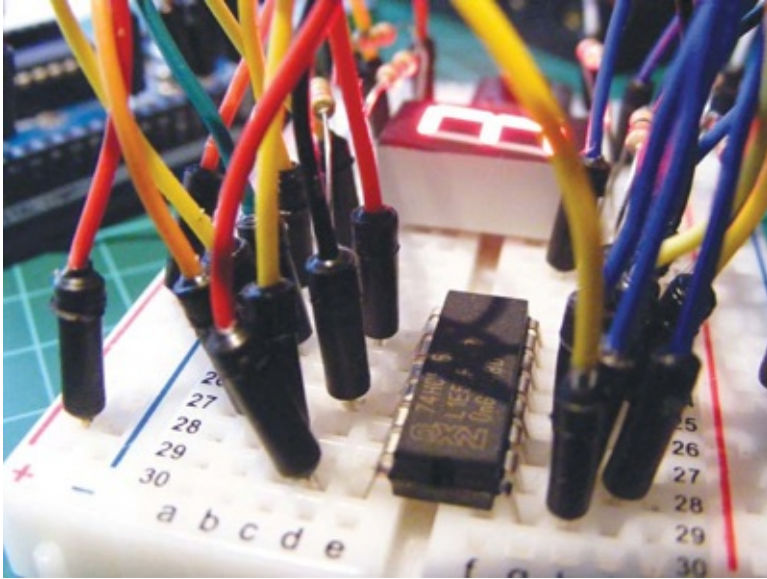
1. Insert the seven-segment LED into your breadboard, making sure it straddles the center break; otherwise, the pins opposite each other will connect and short-circuit. Connect pin 3 to the GND rail, and connect 220-ohm resistors to the remaining pins except pin 8, which is not used. The resistors are needed to prevent the segment LEDs from burning out. See [Figure 16-2](#) for this setup.

1. **FIGURE 16-2:**
Connecting the seven-segment LED



2. Insert the 74HC595 shift register into the breadboard with the semicircle marker of the IC on the left side. The bottom left-hand pin should be pin 1. Your IC needs to straddle the center break, as shown in [Figure 16-3](#).

2. **FIGURE 16-3:**
The 74HC595 shift register should straddle the breadboard center break.



3. Carefully make the connections shown in the following table between the seven-segment LED display and the 74HC595 shift register.

SEVEN-SEGMENT LED DISPLAY	SHIFT REGISTER	ARDUINO
Pin 1 (E)*	Pin 4	
Pin 2 (D)*	Pin 3	
Pin 3		GND
Pin 4 (C)*	Pin 2	
Pin 5 (DP)*	Pin 7	
Pin 6 (B)*	Pin 1	
Pin 7 (A)*	Pin 15	
Pin 8		Not used
Pin 9 (F)*	Pin 5	
Pin 10 (G)*	Pin 6	

* These pins require a 220-ohm resistor between the seven-segment LED display and the 74HC595 shift register.

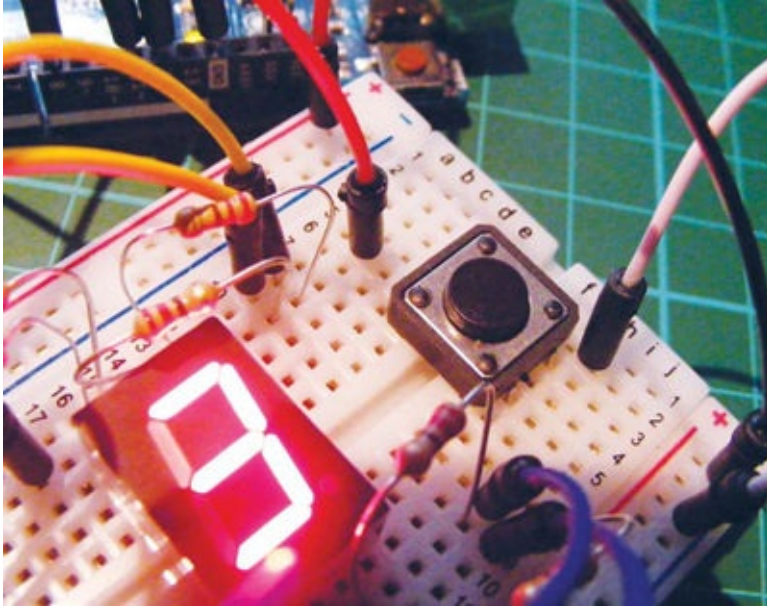
4. Now connect the remaining shift register pins to the Arduino as shown in the following table.

SHIFT REGISTER	ARDUINO

Pin 9	Not used
Pin 10	+5V
Pin 11	Pin 12
Pin 12	Pin 8
Pin 13	GND
Pin 14	Pin 11
Pin 16	+5V
Pulse	Pin 2

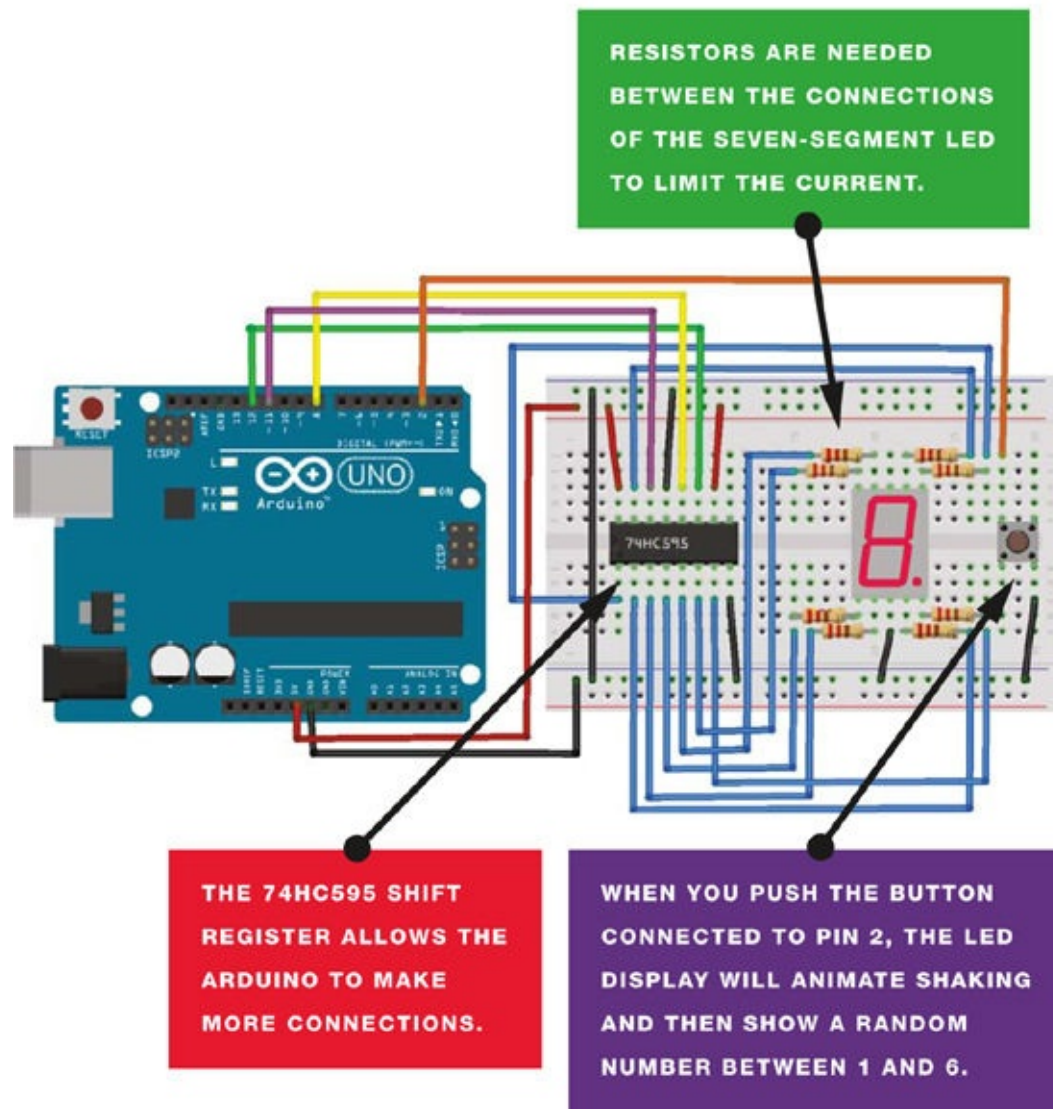
5. Insert the pushbutton into the breadboard with the pins straddling the center break, as shown in [Figure 16-4](#). Connect one side to pin 2 on the Arduino and the other side to GND.

5. **FIGURE 16-4:**
The pushbutton should also straddle the breadboard center break.



6. Confirm that your setup matches the circuit diagram in [Figure 16-5](#), and upload the code in “[The Sketch](#)” on [page 140](#).

6. **FIGURE 16-5:**
The circuit diagram for the electronic die



THE SKETCH

The sketch first sets the pins to control the 74HC595 chip that drives the seven-segment LED. When the seven-segment LED display is powered up, the dot is lit. When you press the pushbutton, the LEDs light in a short, rotating animation to signify that the die is shaking. After a moment a random number between 1 and 6 will be displayed. Press the button again to generate your next roll of the die.

```
// Code by Warrick A. Smith and reproduced with kind permission
// http://startingelectronics.com

const int latchPin = 8; // Pins connected to shift register
const int clockPin = 12;
const int dataPin = 11;
const int buttonPin = 2; // Pin connected to switch wire
// 1 to 6 and DP (decimal point) on 7-segment display
unsigned char lookup_7seg[] = {0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x80};
// Shaking die pattern on 7-segment display
unsigned char shake_dice[] = {0x63, 0x5C};
```

```

// Rolling die on 7-segment display
unsigned char roll_dice[] = {0x1C, 0x58, 0x54, 0x4C};
// Vary duration of time before die number is received
int rand_seed;
int rand_num = 0;           // Generate random number
unsigned char shake_toggle = 0; // For shaking dice animation
int index = 0;             // For rolling dice animation
int shake_speed;           // Accelerates dice shake speed

void setup() {
  pinMode(latchPin, OUTPUT); // Output pins for controlling the
                             // shift register

  pinMode(clockPin, OUTPUT);
  pinMode(dataPin, OUTPUT);
  pinMode(buttonPin, INPUT); // Read switch wire state
  digitalWrite(latchPin, LOW); // Display DP on 7-segment display
                             // at startup
  shiftOut(dataPin, clockPin, MSBFIRST, lookup_7seg[6]);
  digitalWrite(latchPin, HIGH);
  randomSeed(analogRead(0)); // Generate random seed
}

void loop() {
  if (digitalRead(buttonPin)) {
    shake_speed = 150; // Reset die shaking speed
    delay(30);
    // Generate number for random speed and show shaking animation
    while (digitalRead(buttonPin)) {
      rand_seed++; // Generate random number
      // Animate shaking die
      if (shake_toggle) {
        AnimateDice(0, shake_dice);
        shake_toggle = 0;
      }
      else {
        AnimateDice(1, shake_dice);
        shake_toggle = 1;
      }
      delay(80 + shake_speed); // Accelerate animation speed
      if (shake_speed > 0) {
        shake_speed -= 10;
      }
    }
    // Animate rolling die
    for (int rolls = 0; rolls < (rand_seed % 10 + 14); rolls++) {
      AnimateDice(index, roll_dice);
      delay((1 + rolls) * 20);
      index++;
      if (index > 3) {
        index = 0;
      }
    }
    rand_num = random(0, 6); // Generate number thrown on die
    DiceNumber(rand_num);
  }
}

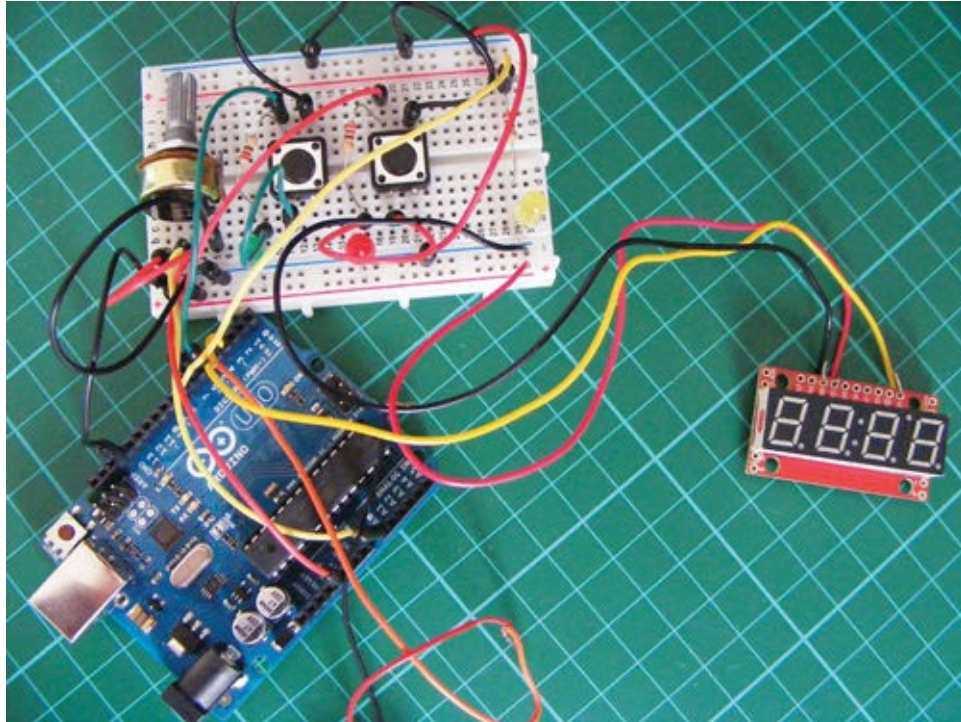
// Display the die number on 7-segment display
void DiceNumber(unsigned char num) {
  digitalWrite(latchPin, LOW);
  shiftOut(dataPin, clockPin, MSBFIRST, lookup_7seg[num]);
  digitalWrite(latchPin, HIGH);
}

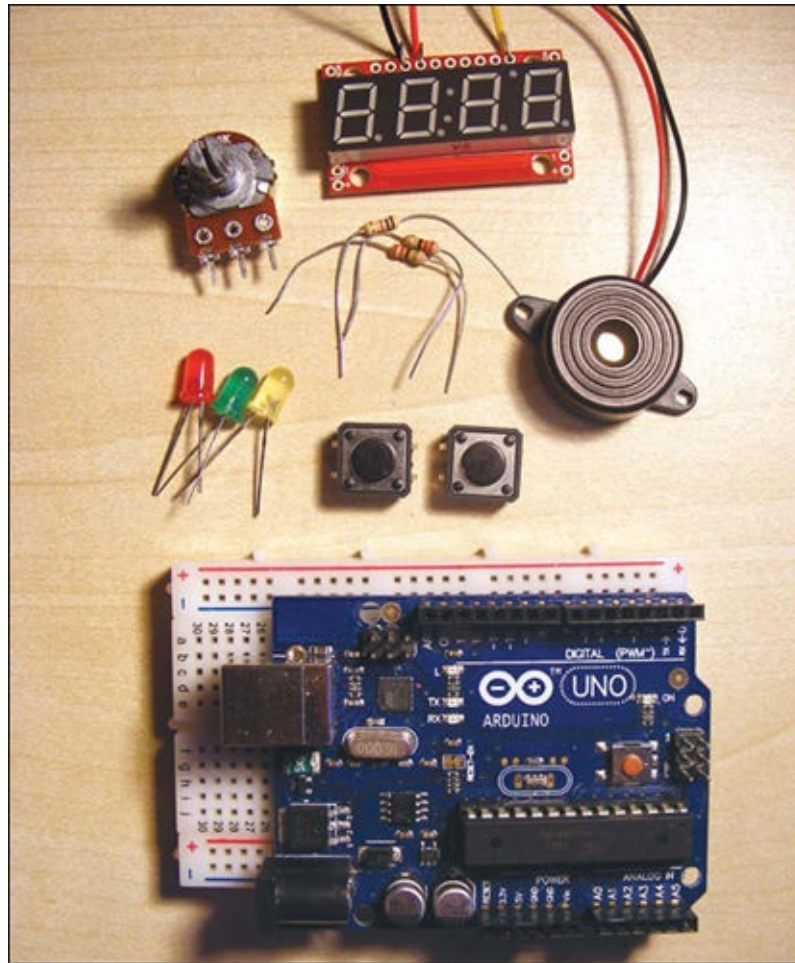
```

```
// Display one frame of the shaking or rolling dice
void AnimateDice(int seg, unsigned char *table) {
    digitalWrite(latchPin, LOW);
    shiftOut(dataPin, clockPin, MSBFIRST, table[seg]);
    digitalWrite(latchPin, HIGH);
}
```

PROJECT 17: ROCKET LAUNCHER

IN THIS PROJECT WE'LL CREATE A PROGRAMMABLE COUNTDOWN TIMER THAT WE'LL USE TO LAUNCH A ROCKET BY IGNITING A FUSE WHEN THE COUNTDOWN REACHES 0.





PARTS REQUIRED

- Arduino board
- Breadboard
- Jumper wires
- Four-digit, seven-segment serial display
- Piezo buzzer
- 2 momentary tactile four-pin pushbutton
- 50k-ohm potentiometer
- 3 LEDs (red, green, yellow)
- 3 220-ohm resistors

LIBRARIES REQUIRED

- SoftwareSerial

We'll use a four-digit, seven-segment serial display that has a built-in integrated circuit to control the LEDs and can be connected to the Arduino with only three wires. When choosing your display, make sure it has an RX input so you'll be able to control it with only one wire.

HOW IT WORKS

You could use a timer like this to set off anything that requires power, like a servomotor, LED, or alarm. You'll use a potentiometer to select the duration of your countdown (anywhere from 5 to 60 seconds). The LED screen will display the digits so you can see what you are setting the countdown to. We'll include two pushbuttons: an Arm button and a Launch button. Once you've chosen the duration of your countdown, press the Arm button to ready the timer. The red LED light shows that it's armed. (The Arm button is a safety feature to prevent you from accidentally setting off the launcher.) Once you've armed the rocket, press the Launch button to start the countdown. The green LED light signifies that it's ready, and the countdown begins.

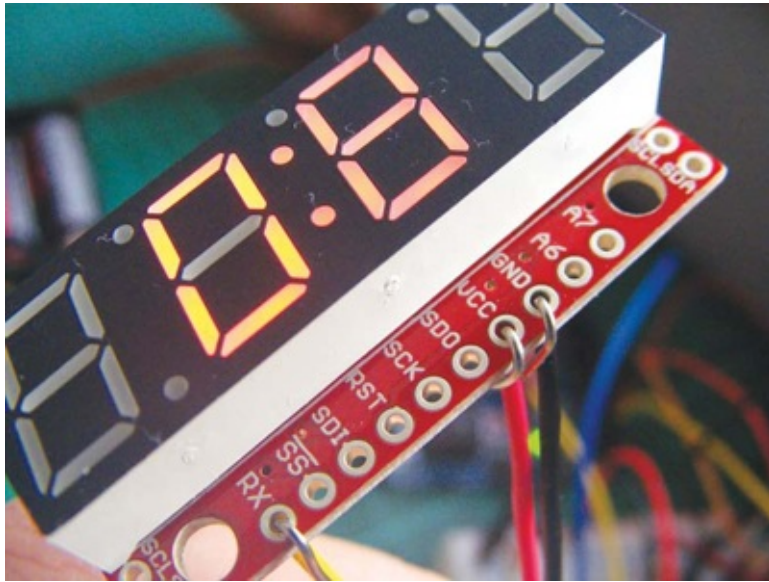
As the timer counts down, the piezo buzzer beeps every second. When the counter reaches five seconds, the timer beeps increasingly quickly until launch. When the timer reaches 0, power is sent through pin 7 to whatever output you have there—in this case, it lights the yellow LED. You could connect this timer to a buzzer, a servomotor to unlock a door, or even a fuse to ignite a rocket. I'll show you how to make your own simple ignition for a fuse later in this project.

THE BUILD

1. Connect the seven-segment serial display RX pin to Arduino pin 3, connect VCC to +5V, and connect GND to Arduino GND via the breadboard, as shown in [Figure 17-1](#). You might need to strip back some of the wire to make the connection.

SEVEN-SEGMENT SERIAL DISPLAY	ARDUINO
RX	Pin 3
VCC	+5V
GND	GND

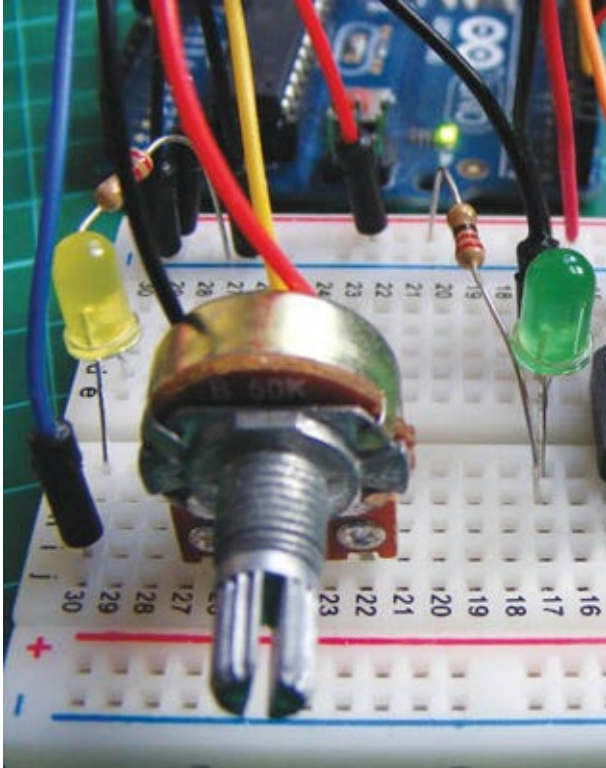
1. **FIGURE 17-1:**
Connecting the seven-segment display to the Arduino



2. Insert the potentiometer into the breadboard and connect the left pin to +5V, the center pin to Arduino pin A0, and the right pin to GND, as shown in [Figure 17-2](#).

POTENTIOMETER	ARDUINO
Left pin	+5V
Center pin	A0
Right pin	GND

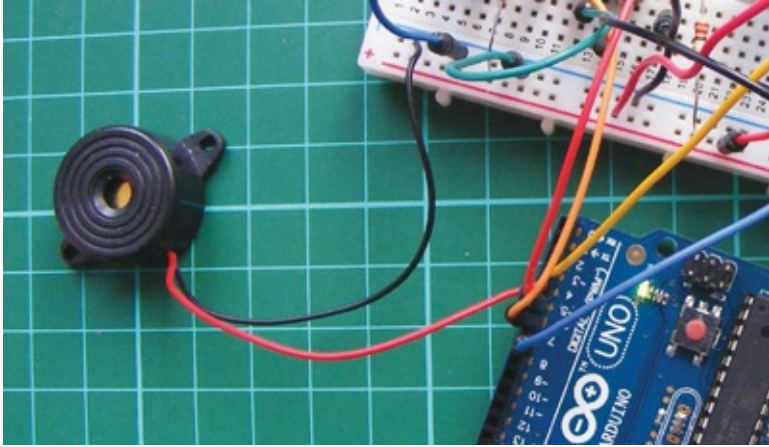
2. **FIGURE 17-2:**
Placing the potentiometer into the breadboard



3. Connect the red wire of the piezo buzzer to Arduino pin 4 and the black wire to GND, as shown in [Figure 17-3](#).

PIEZO	ARDUINO
Red wire	Pin 4
Black wire	GND

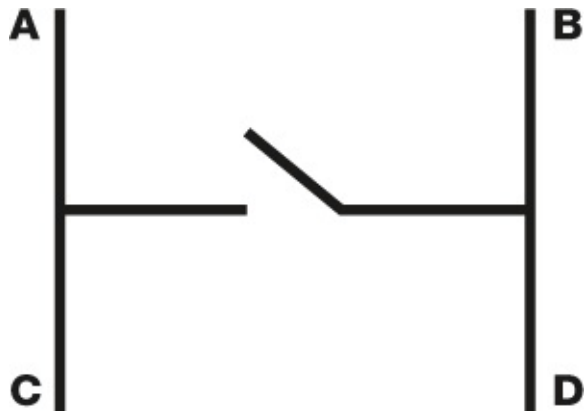
3. **FIGURE 17-3:**
Connecting the piezo buzzer



4. Insert the two pushbuttons into your breadboard, with pins A and B on one side of the center break and pins D and C on the other, following the configuration in [Figure 17-4](#).

4. **FIGURE 17-4:**

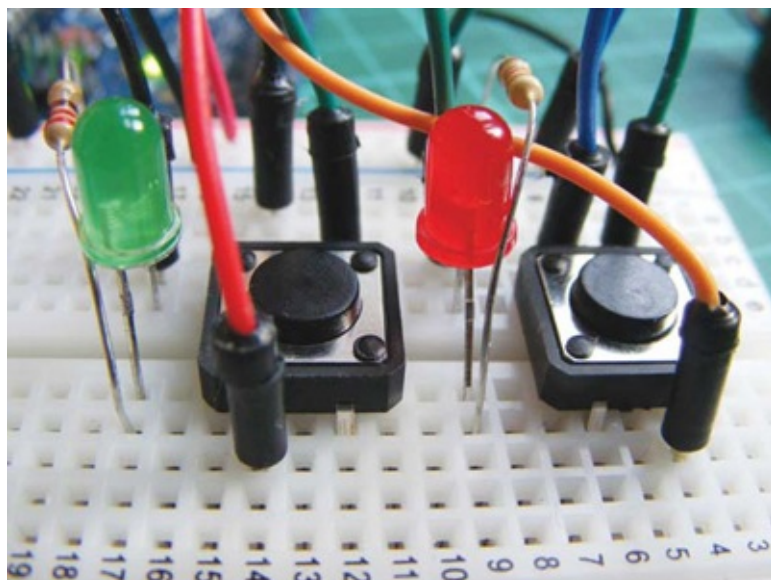
The pin connections of the pushbutton



5. Next, we'll connect the pushbuttons, as shown in [Figure 17-5](#). To create the Arm button, connect pin C of the first pushbutton to GND and pin D to Arduino pin 5. To create the Launch button, connect pin C of the other pushbutton to GND and pin D to Arduino pin 6.

PUSHBUTTONS	ARDUINO
Arm pin C	GND
Arm pin D	Pin 5
Launch pin C	GND
Launch pin D	Pin 6

5. **FIGURE 17-5:**
Connecting the pushbuttons and LEDs



6. Insert the red LED into the breadboard with the shorter, negative leg connected to pin B of the Arm button. Connect the other leg to a 220-ohm resistor, and connect the other side of the resistor to +5V. Then insert the green LED with the negative leg connected to pin B of the Launch button, and the positive leg connected to +5V via a 220-ohm resistor.

RESISTORS	ARDUINO
Negative legs	GND
Positive legs	+5V via 220-ohm resistor

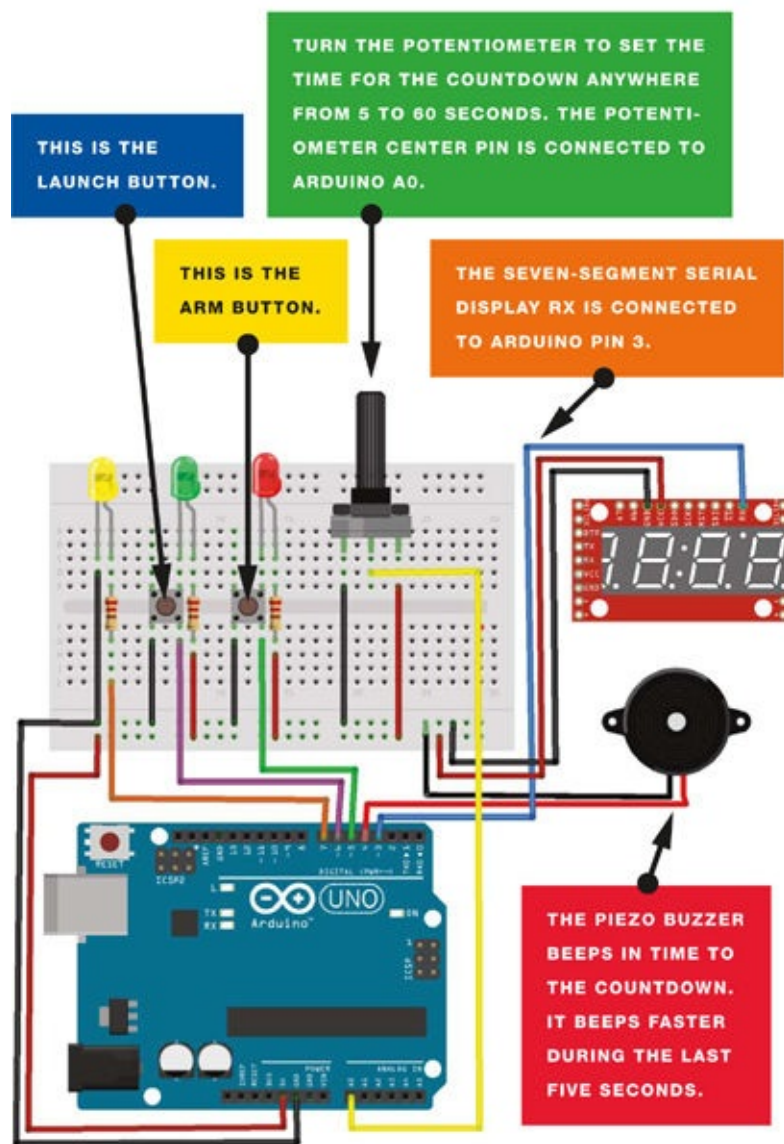
7. Connect the igniter. We’re using a yellow LED as our igniter indicator for now. Insert it into the breadboard with the negative leg connected to GND and the positive leg connected to Arduino pin 7 via a 220-ohm resistor. (See “[Create a Working Fuse](#)” on [page 149](#) to learn how to make your own fuse igniter.)

IGNITER	ARDUINO
Negative leg	GND
Positive leg	Pin 7 via 220-ohm resistor

When the countdown reaches 0, pin 7 is set to HIGH and triggers the igniter. Instead of actually igniting a fuse, we light the yellow LED to represent the ignition.

8. Confirm that your setup matches the circuit diagram in [Figure 17-6](#), and upload the code in “[The Sketch](#)” on [page 151](#).

8. **FIGURE 17-6:**
The circuit diagram for the rocket launcher



CREATE A WORKING FUSE

Instead of using an LED to indicate ignition, you can create a working fuse using a simple Christmas tree light. Be sure to wear eye protection when creating your fuse. These instructions are for entertainment purposes and should be carried out only by an adult.

WARNING

There may be restrictions to launching a hobby rocket or firework in your country or state, so please check beforehand. It is your responsibility to keep project use within the law.

1. Using a hobby drill, apply light pressure to the top of the glass casing on a Christmas light to cut it off (see [Figure 17-7](#)).

1. **FIGURE 17-7:**
Cutting the tip with a hobby drill



2. Cut near the tip of the glass casing and the top should pop off easily ([Figure 17-8](#)).

2. **FIGURE 17-8:**
Popping off the tip



3. Now cut off the head of a wooden match (make sure you don't ignite it!) and gently insert the match head into the open bulb, taking care not to damage the filament ([Figure 17-9](#)).

3. **FIGURE 17-9:**
Inserting a match head into the bottom half of the bulb



4. Finally, connect the bulb wires to your ignition wires. When power is sent to the bulb, the filament will heat up and ignite the match head ([Figure 17-10](#)), creating enough energy to ignite a fuse.

4. **FIGURE 17-10:**
After the fuse has been lit



THE SKETCH

The sketch first defines each component and its connection to the Arduino. The `SoftwareSerial` library controls the four-digit, seven-segment serial LED display, while the analog input from the potentiometer changes the time displayed from 5 to 60 seconds. When pressed, the Arm button acts as a digital switch and safety feature to allow the Launch button to be pressed. If the Arm button is pushed during countdown, the countdown aborts and the display resets.

The `tone` commands in the sketch pulse the piezo buzzer in time to the countdown to create a beep. When the countdown reaches 0, the igniter pin (in this case, connected to an LED) is set to `HIGH` and turns on the LED.

```
// Ardunaut Arduining.com, reproduced with kind permission

#define FuseTIME      1500    // Duration of fuse current in ms
#include <SoftwareSerial.h>    // Call the SoftwareSerial library

#define Fuse          7       // Pin connected to fuse (your LED or igniter)
#define GoButt       6       // Pin connected to Launch button
#define ArmButt      5       // Pin connected to Arm button
#define BuzzPin      4       // Pin connected to piezo buzzer
#define TXdata       3       // Pin connected to RX of display
#define RXdata       2       // Not used
#define SetPot       0       // Analog pin connected to potentiometer

SoftwareSerial mySerialPort(RXdata, TXdata);

void setup() {
  pinMode(TXdata, OUTPUT);
  pinMode(RXdata, INPUT);
  pinMode(Fuse, OUTPUT);
  pinMode(ArmButt, INPUT);    // Set Arm button pin to input
  pinMode(GoButt, INPUT);    // Set Launch button pin to input
}
```

```

digitalWrite(Fuse, LOW);           // Open igniter circuit
digitalWrite(ArmButt, HIGH);       // Turn on resistor
digitalWrite(GoButt, HIGH);       // Turn on resistor
mySerialPort.begin(9600);
delay(10);                          // Wait for serial display startup
mySerialPort.print("v");           // Reset the serial display
mySerialPort.print("z");           // Brightness
mySerialPort.write(0x40);          // 3/4 intensity
mySerialPort.print("w");           // Decimal point control
mySerialPort.write(0x10);          // Turn on colon in serial display
}

int DownCntr;    // Countdown (1/10 seconds)
int Go = 0;      // Stopped

void loop() {
  if (!digitalRead(GoButt) || !digitalRead(ArmButt)) {
    Go = 0;      // Abort the countdown
    tone(BuzzPin, 440, 1500);
    delay(1500);
  }

  if (Go == 0) {
    WaitARM();
    WaitGO();
  }
  ShowTimer();
  if (DownCntr > 50) {
    if (DownCntr % 10 == 0) tone(BuzzPin, 1000, 50); // One beep/sec
  }
  else if (DownCntr % 2 == 0) tone(BuzzPin, 1000, 50); // Beep faster

  if (DownCntr == 0) {
    tone(BuzzPin, 440, FuseTIME); // Launch tone
    digitalWrite(Fuse, HIGH);     // Close the fuse circuit
    delay(FuseTIME);
    digitalWrite(Fuse, LOW);      // Open the fuse circuit
    Go = 0;
  }
  while (millis() % 100);          // Wait 50 ms
  delay(50);
  DownCntr--;
}

void WaitGO() {
  ShowTimer();
  while (digitalRead(GoButt));
  Go = 1;
  delay(20);
  while (!digitalRead(GoButt)); // Debounce Launch button
}

void ReadTimer() {
  DownCntr = map(analogRead(SetPot), 0, 1023, 5, 60);
  DownCntr *= 10;
}

void ShowTimer() {
  String seconds = String(DownCntr, DEC);
  while (seconds.length() < 3) seconds = "0" + seconds; // Format to
                                                         // 3 numbers
  mySerialPort.print(seconds); // Write to serial display
  mySerialPort.print(" ");     // Last digit off
}

```

```
void WaitARM() {
  while (digitalRead(ArmButt) == 1) {
    ReadTimer();
    delay(50);
    ReadTimer();
    ShowTimer();
    delay(150);
  }

  Go = 0;
  ShowTimer();
  tone(BuzzPin, 2000, 150);
  delay(200);
  tone(BuzzPin, 2000, 150);
  delay(200);
  tone(BuzzPin, 2000, 150);

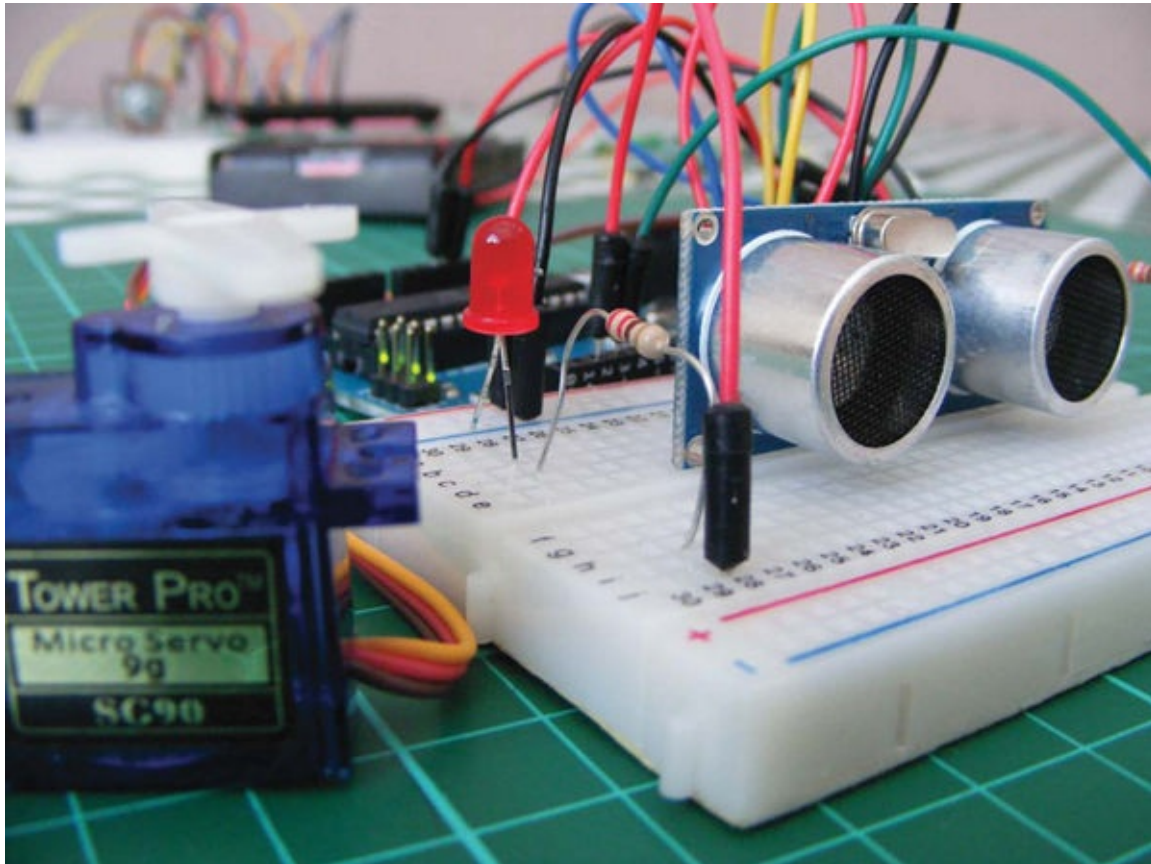
  delay(20);
  while (!digitalRead(ArmButt)); // Debounce Arm button
}
```

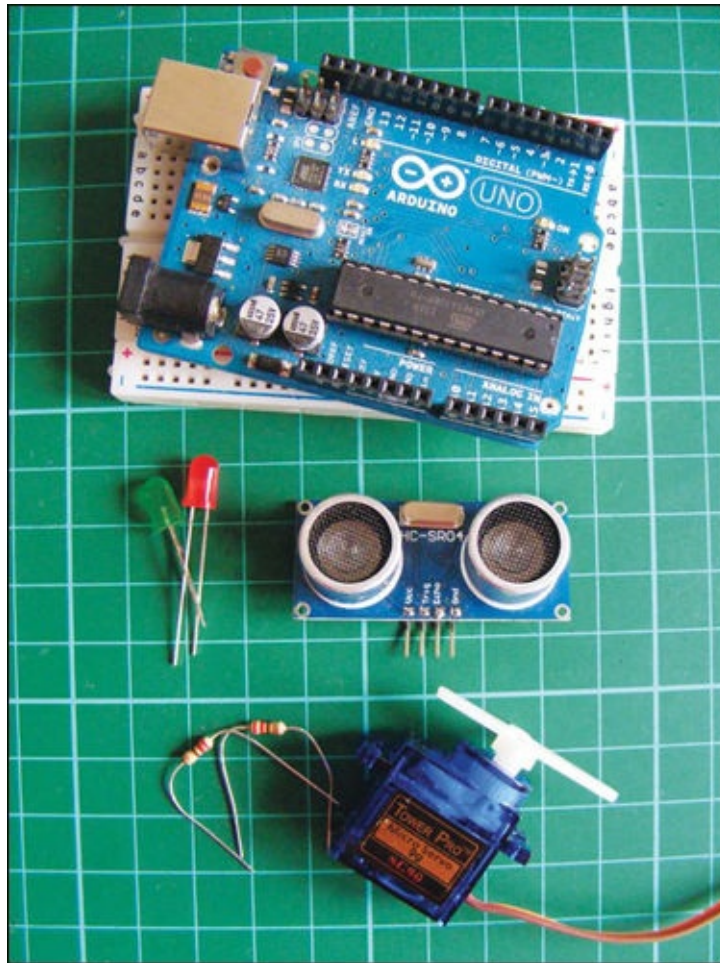
PART 6

SECURITY

PROJECT 18: INTRUDER SENSOR

IN THIS PROJECT, WE'LL USE AN ULTRASONIC SENSOR TO DETECT AN INTRUDER.



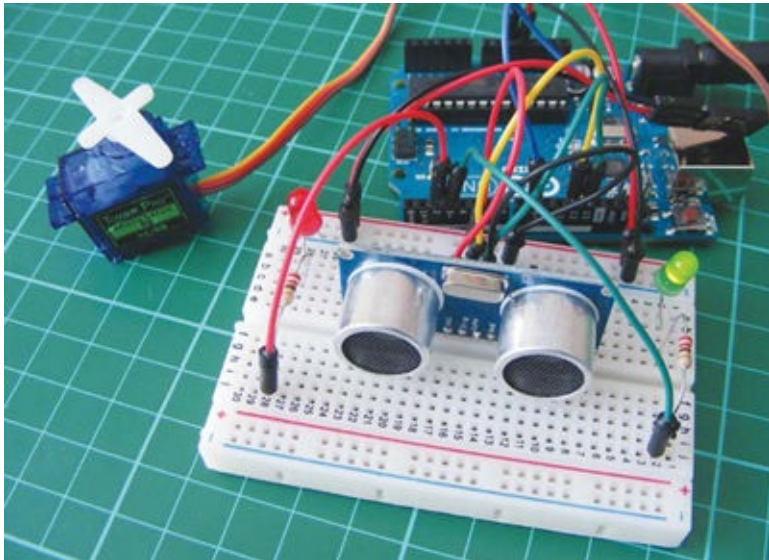


PARTS REQUIRED

- Arduino board
- Breadboard
- Jumper wires
- Four-pin HC-SR04 ultrasonic sensor
- Servomotor
- Red LED
- Green LED
- 2 220-ohm resistors

We'll connect the intruder sensor to a servo and some LEDs so that when someone comes within a certain distance, a green LED turns off, a red LED turns on, and the servomotor moves (see [Figure 18-1](#)).

FIGURE 18-1:
The LEDs alert you to an intruder.



HOW IT WORKS

This project is versatile and can be used and adapted in various ways. Because the ultrasonic sensor can define distance, you could, for example, use it to define an area and trigger an alarm when that perimeter is breached. The sensor works similarly to a radar: it sends out an ultrasonic signal, or *ping*. When this signal hits an object, it bounces back like an echo, and the time between the ping and the echo is used to calculate distance. The Arduino can use this calculation to trigger an event, depending on the value received.

In this project, when the sensor detects an intruder within a predefined vicinity, the red LED will light and the servo arm will move. You can adapt this project to trigger a different event when the intruder is detected, like pressing a security system button or locking a door. For a friendlier scenario, you could set the distance really close so that when you wave your hand in front of the sensor, the servo presses a button to release a treat, like candy.

NOTE

To use the same ultrasonic sensor shown in these figures, see “[Retailer List](#)” on [page 240](#) or search online for HC-SR04 ultrasonic module.

THE BUILD

1. Insert the ultrasonic sensor into the breadboard. The sensor we’re using in this project has four pins, as shown in [Figure 18-2](#). Connect the sensor’s GND to the Arduino GND rail, VCC to Arduino +5V, Trig to Arduino pin 12, and Echo to Arduino pin 13.

ULTRASONIC SENSOR	ARDUINO
GND	GND

VCC	+5V
Trig	Pin 12
Echo	Pin 13

1. **FIGURE 18-2:**
The HC-SR04 ultrasonic sensor



2. Connect the servo's brown (ground) wire to the Arduino GND rail, its red (power) wire to the Arduino +5V rail, and its yellow signal (control) wire to Arduino pin 9.

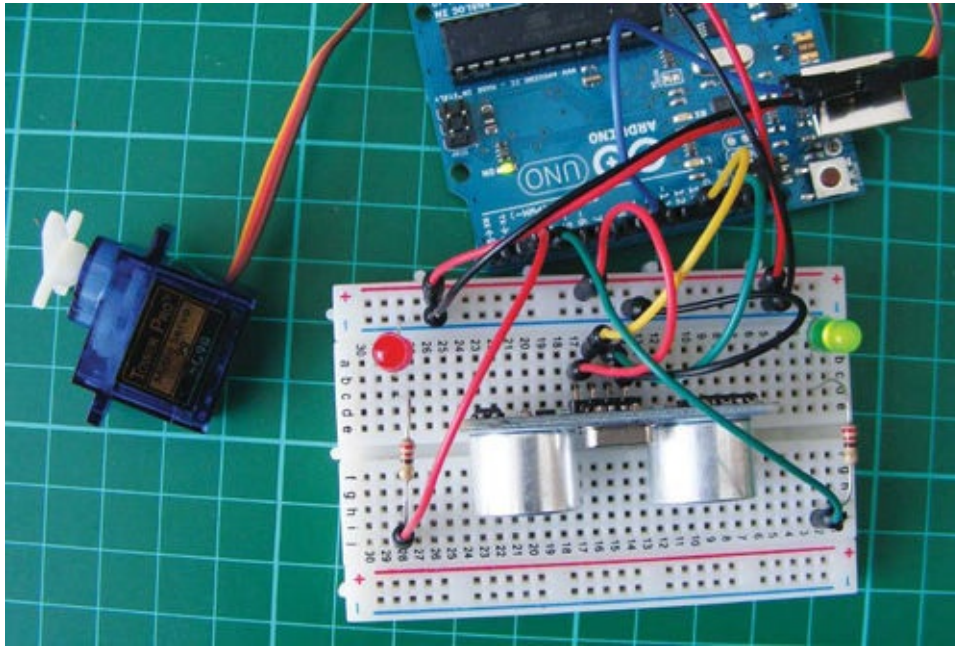
SERVO	ARDUINO
Red wire	+5V
Brown wire	GND
Yellow wire	Pin 9

3. Insert the red and green LEDs into the breadboard with the shorter, negative legs in the Arduino GND rail. Add a 220-ohm resistor to each of the positive legs, and connect the red LED to Arduino pin 2 and the green LED to pin 3 via the resistors.

LEDS	ARDUINO
Negative legs	GND
Positive leg (red)	Pin 2 via 220-ohm resistor
Positive leg (green)	Pin 3 via 220-ohm resistor

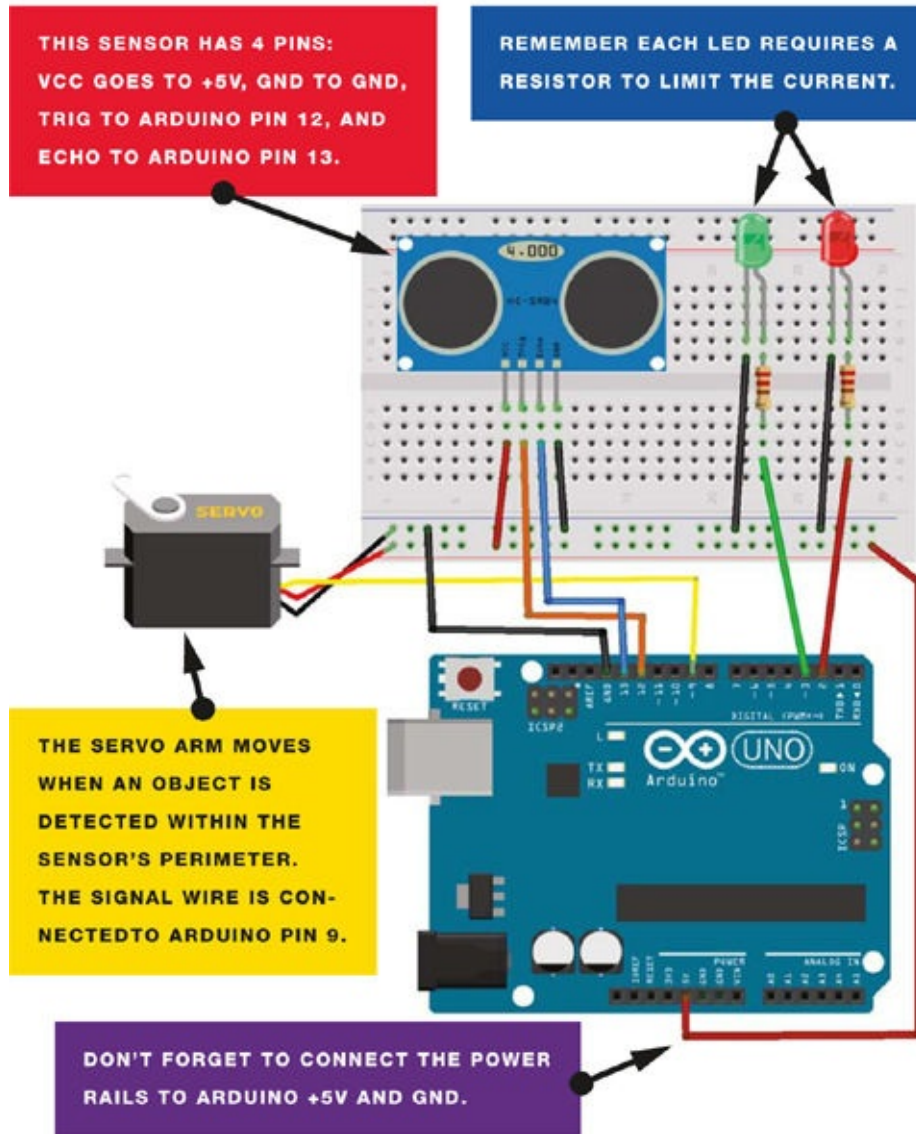
4. Connect the power rails on the breadboard to Arduino +5V and GND. The final configuration is shown in [Figure 18-3](#).

4. **FIGURE 18-3:**
The complete intruder sensor project



5. Check that your setup matches that of [Figure 18-4](#) and then upload the code in “[The Sketch](#)” on [page 161](#).

5. **FIGURE 18-4:**
The circuit diagram for the intruder sensor



THE SKETCH

When an object is within the trigger distance, the red LED will light and the servo will move 45 degrees. You can change the distance of the sensor field in the following line of the sketch:

```
if (distance <= 15)
```

In this example, if something is sensed within a distance of 15 centimeters, the next block of code will run.

The Trig pin on the sensor is connected to Arduino pin 12 and emits an ultrasonic signal or ping. When the signal reaches an object, it bounces back to the module, and this echo is sent to Arduino pin 13. The time difference between the two signals gives us our distance reading. If the distance is more than our set minimum, the green LED stays on; if not, the red LED lights and the servo moves.

```
/* NewPing Library created by Tim Eckel teckel@leethost.com.  
Copyright 2012 License: GNU GPL v3
```

```

    http://www.gnu.org/licenses/gpl-3.0.html
*/

#include <NewPing.h> // Call NewPing library
#include <Servo.h>    // Call Servo library
#define trigPin 12   // Trig pin connected to Arduino 12
#define echoPin 13   // Echo pin connected to Arduino 13
#define MAX_DISTANCE 500
NewPing sonar(trigPin, echoPin, MAX_DISTANCE); // Library setting
int greenLed = 3, redLed = 2; // Set green LED to pin 3, red to pin 2
int pos = 20;
Servo myservo;

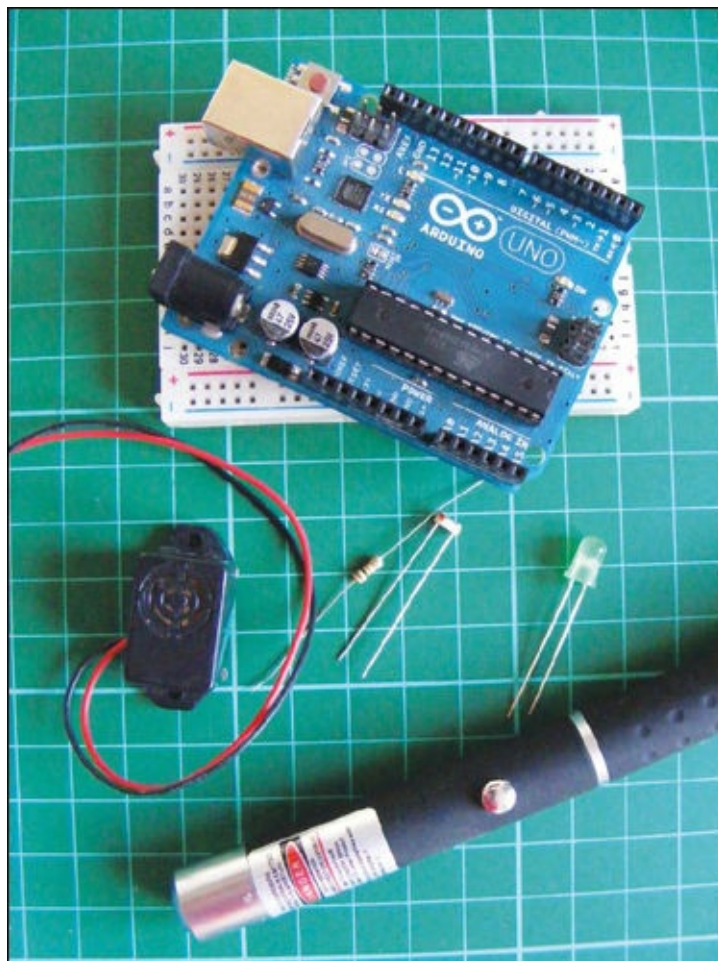
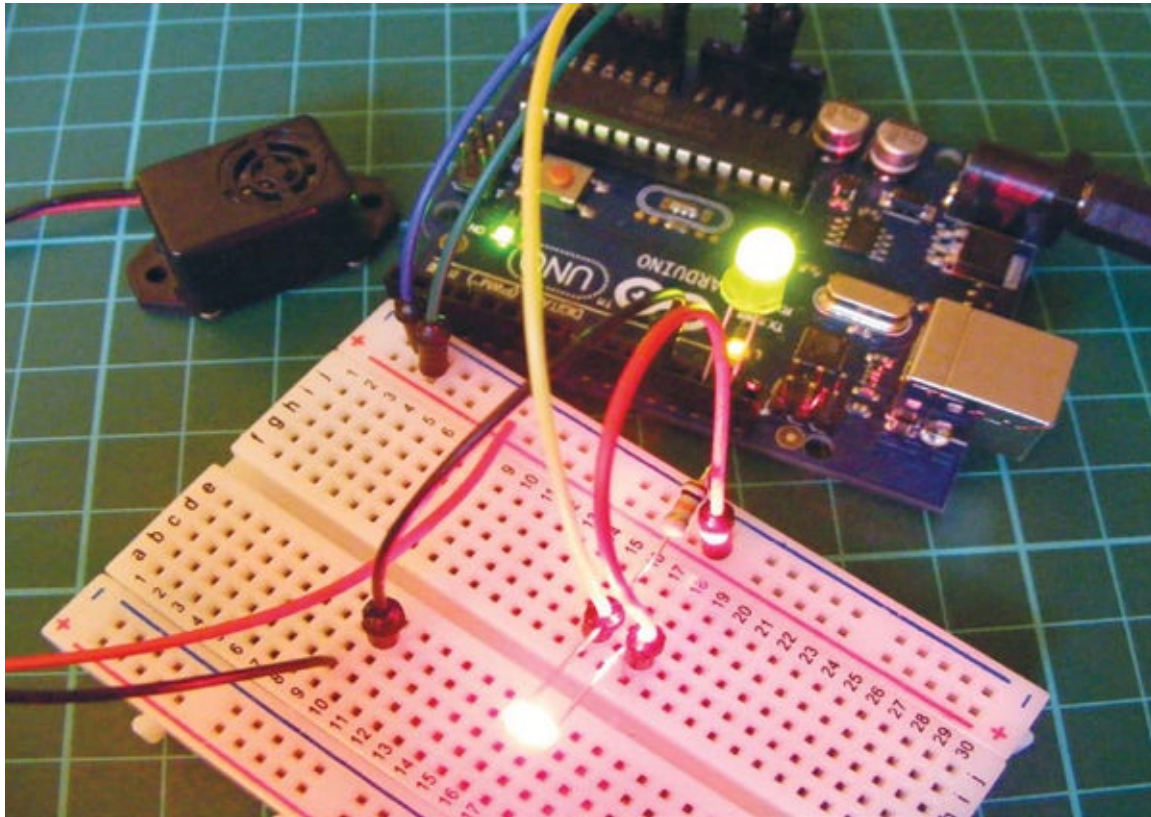
void setup() {
    Serial.begin (115200);
    pinMode(trigPin, OUTPUT);
    pinMode(echoPin, INPUT);
    pinMode(greenLed, OUTPUT);
    pinMode(redLed, OUTPUT);
    myservo.attach(9); // Servo attached to pin 9
}

void loop() {
    int duration, distance, pos = 0, i;
    digitalWrite(trigPin, LOW);
    delayMicroseconds(2);
    digitalWrite(trigPin, HIGH); // Trig pin sends a ping
    delayMicroseconds(10);
    digitalWrite(trigPin, LOW);
    duration = pulseIn(echoPin, HIGH); // Echo receives the ping
    distance = (duration / 2) / 29.1;
    Serial.print(distance);
    Serial.println(" cm");
    // If sensor detects object within 15 cm
    if (distance <= 15) {
        digitalWrite(greenLed, LOW); // Turn off green LED
        digitalWrite(redLed, HIGH);  // Turn on red LED
        myservo.write(180);          // Move servo arm 180 degrees
        delay(450);
        digitalWrite(redLed, LOW);   // Light the red LED
        myservo.write(90);
        delay(450);
        digitalWrite(redLed, HIGH);
        myservo.write(0);
        delay(450);
        digitalWrite(redLed, LOW);
        myservo.write(90);
    }
    // Otherwise
    else {
        digitalWrite(redLed, LOW);    // Turn off red LED
        digitalWrite(greenLed, HIGH); // Turn on green LED
        myservo.write(90);
    }
    delay(450);
}
}

```

PROJECT 19: LASER TRIP WIRE ALARM

IN THIS PROJECT, YOU'LL CREATE A SIMPLE LASER TRIP WIRE ALARM.



PARTS REQUIRED

- Arduino board
- Breadboard
- Jumper wires
- Photoresistor
- Piezo buzzer
- Green LED
- 10k-ohm resistor
- Laser pen

You've probably seen a movie where a valuable item is protected by a grid of laser beams. The beams look cool and seem pretty high-tech, but the principles behind them are actually very simple.

HOW IT WORKS

When the laser pen shines on the photoresistor, the green LED will light up to signify that the circuit is ready. When the laser beam is broken, the LED turns off and the buzzer sounds.

As we know from [Projects 13](#) and [18](#), photoresistors produce variable resistance depending on the amount of light falling on their sensor. When the photoresistor does not detect light from the laser, it will drop its resistance and trigger the Arduino to send voltage to the pin controlling the buzzer.

Laser beams that are visible in daylight or even in the dark are very powerful and can be extremely dangerous. In this project we'll use a low-powered laser pen instead (see [Figure 19-1](#)).

FIGURE 19-1:

Laser pens can still be dangerous and should never be directed toward anybody's eyes!



THE BUILD

1. Insert your photoresistor into the breadboard. Connect one leg to the +5V rail using a jumper wire. Connect a 10k-ohm resistor to the other leg, and connect the other side of this resistor to Arduino A0 and GND on the breadboard.

PHOTORESISTOR	ARDUINO
Leg 1	+5V
Leg 2	A0 via 10k-ohm resistor and GND

2. Connect the red (positive) wire of the piezo buzzer directly to Arduino pin 11 on the Arduino and the black (GND) wire to GND on the breadboard.

PIEZO	ARDUINO
Black wire	GND
Red wire	Pin 11

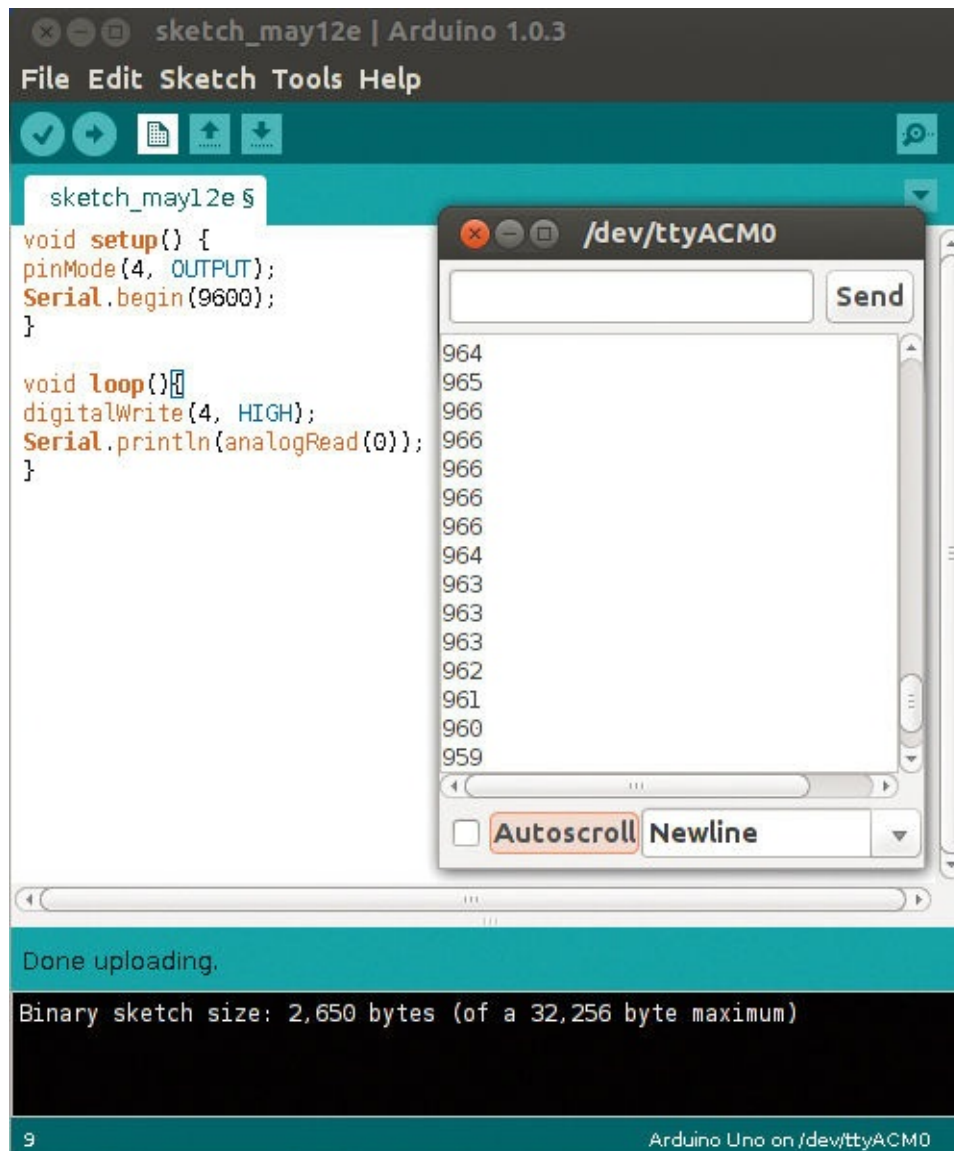
3. Insert the green LED's long leg into Arduino pin 13 and the short leg into GND.
4. Connect the power rails to the breadboard.
5. Before you upload the code, you need to check the photo-resistor's value in ambient light. Run the following small program with the photoresistor set up as instructed.

```
void setup() {  
  pinMode(4, OUTPUT);  
  Serial.begin(9600);  
}
```

```
}  
  
void loop() {  
  digitalWrite(4, HIGH);  
  Serial.println(analogRead(0));  
}
```

6. Open the Serial Monitor in the Arduino IDE. It will show the value being read from the light resistor—in [Figure 19-2](#), it's 964—in normal lighting conditions. Take note of your number, which will be different depending on your lighting conditions.

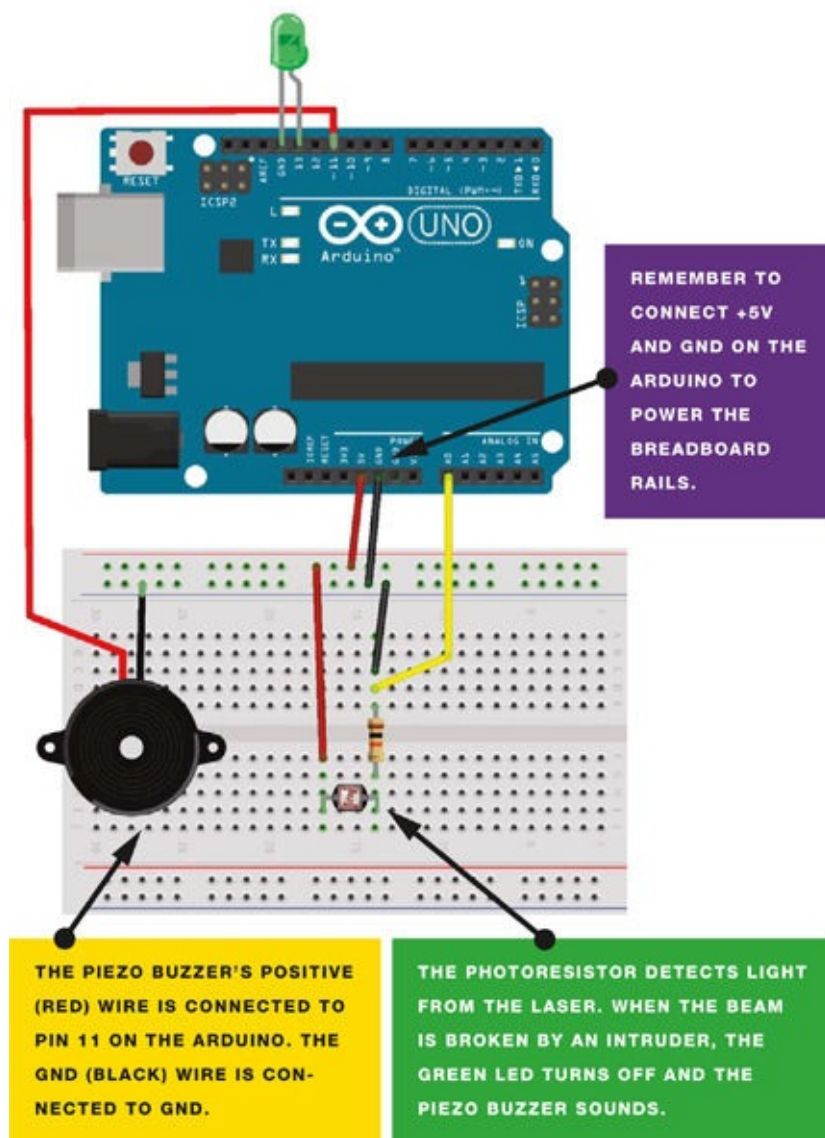
6. **FIGURE 19-2:**
Reading values from the photoresistor



Now shine the laser on the resistor's cell, and also note this number; my reading is 620. This might seem counterintuitive, as you would expect more light to provide a higher number, but the figure is actually translating the resistance—more light, less resistance. Your values will differ from those shown here, so make sure to record your two readings.

7. Check that your setup matches that of [Figure 19-3](#) and then upload the code in “[The Sketch](#)” on [page 168](#).

7. **FIGURE 19-3:**
The circuit diagram for the laser trip wire alarm



THE SKETCH

The sketch first sets Arduino pin 11 as an `OUTPUT` for the piezo buzzer and pin 13 as an `OUTPUT` for the LED. The photoresistor is connected to Arduino pin A0. If the analog reading from A0 is more than 850 (meaning that there is less light and the laser beam has been broken), the buzzer will be set to `HIGH` and turn on and the LED will turn off. Remember to change the resistance value depending on your calibration on this line:

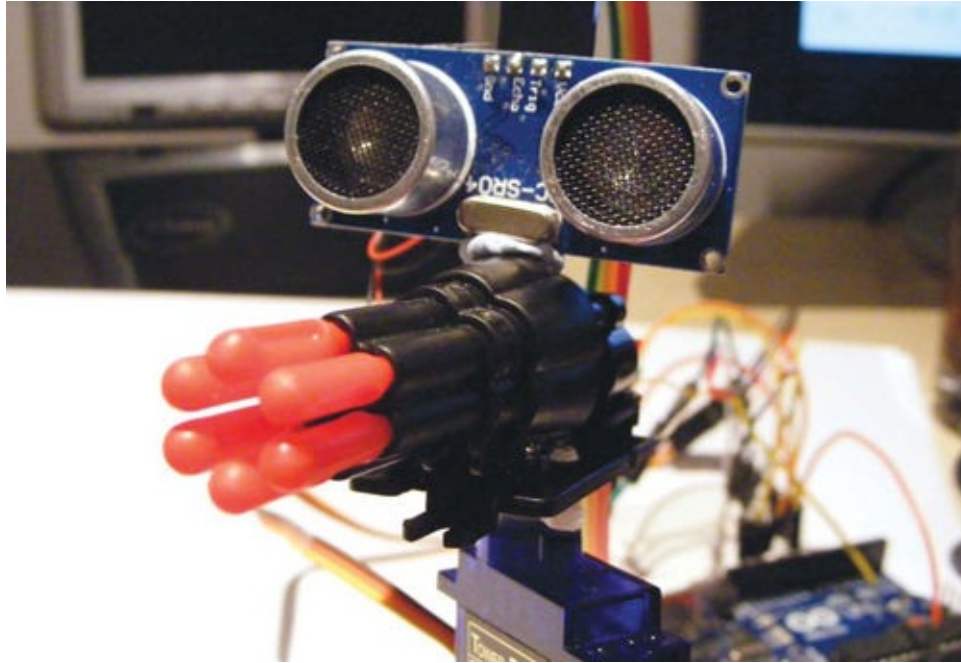
```
if (analogRead(0) > 850) {
```

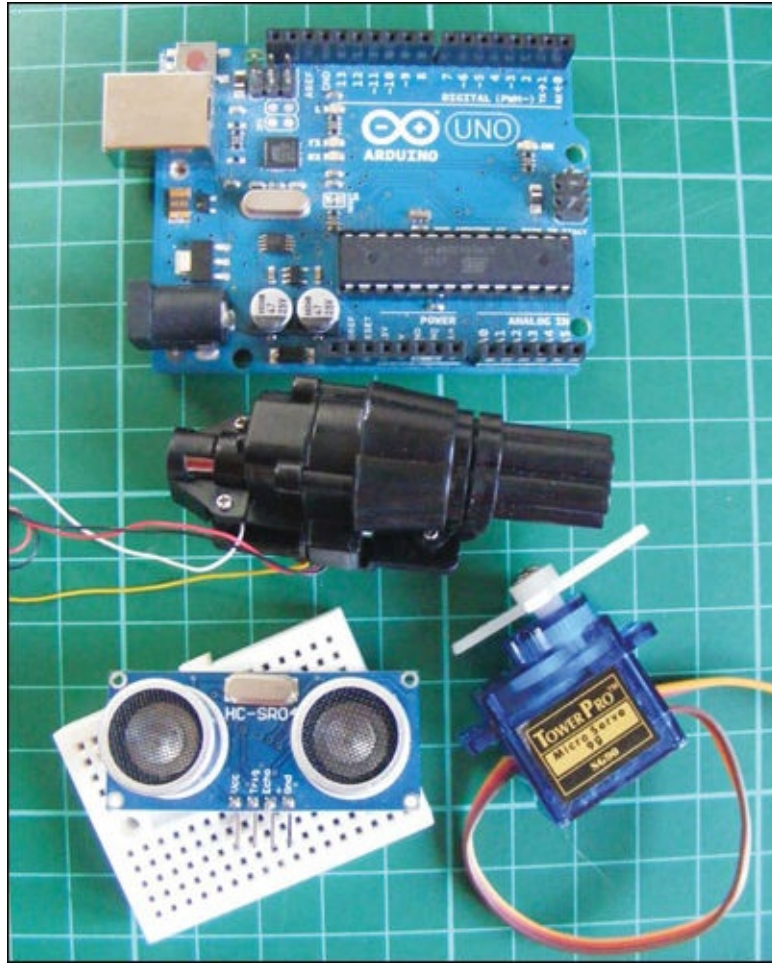
As noted earlier, when the laser is shining on the resistor it reads about 620, so in the sketch I've set the buzzer to sound only if the value is more than 850. This value is between our laser value and our nonlaser value, so we know the laser beam to the resistor has been broken if the value reaches 850.

```
int buzzPin = 11; // Pin connected to the piezo  
int LED = 13;    // Pin connected to the LED
```

PROJECT 20: SENTRY GUN

A SENTRY GUN IS AN UNMANNED WEAPON CAPABLE OF AUTONOMOUSLY SENSING AND FIRING UPON ENEMY TARGETS USING ULTRASONIC DETECTION. IN THIS PROJECT, WE'LL CREATE A MINIATURE VERSION OF THIS GUN.





PARTS REQUIRED

- Arduino board
- Mini breadboard
- Jumper wires
- Male-to-male jumper wires
- Four-pin HC-SR04 ultrasonic sensor
- WLToys RC V959 missile launcher
- Tower Pro SG90 9g servomotor

LIBRARIES REQUIRED

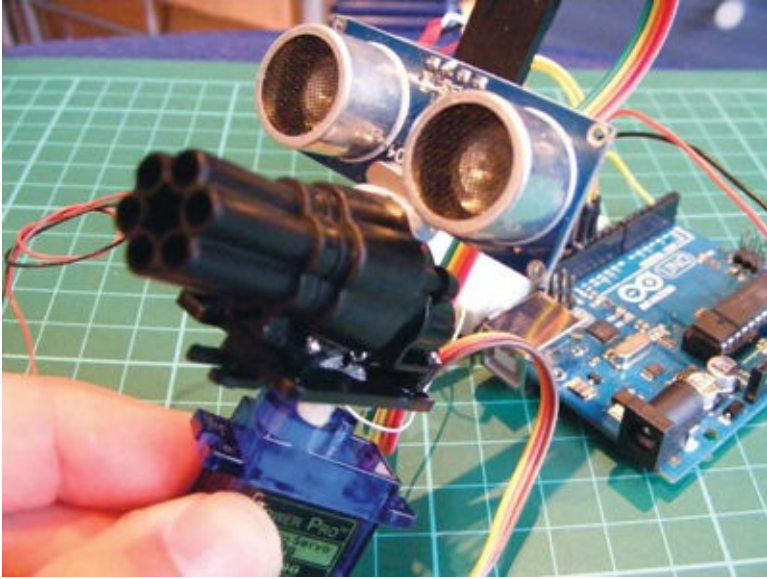
- Servo
- NewPing

HOW IT WORKS

We'll attach the toy missile launcher and the ultrasonic sensor to a servo arm (see [Figure 20-1](#)) so that the servo sweeps the gun and sensor back and forth across 180 degrees, giving the ultrasonic sensor a wide range of detection. When an enemy is detected, the Arduino triggers the sentry gun and discharges the missiles. For more on the ultrasonic sensor, see [Project 18](#).

FIGURE 20-1:

Attaching the toy gun and ultrasonic sensor to the servo arm gives them a wide range of detection and motion.



The key component for this project is the WLToys RC V959 missile launcher, also known as the Walkera Part RC V959-19 missile bullet launcher, intended for radio-controlled helicopters (Figure 20-2).

FIGURE 20-2:

The Walkera Part RC V959-19 missile bullet launcher

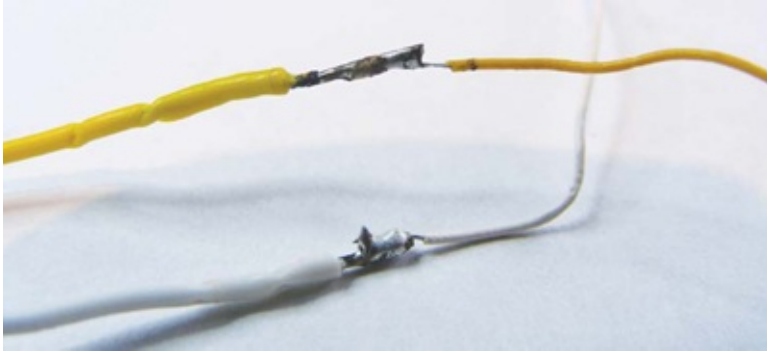


This cool part is really cheap (around \$6 –10) and is widely available online. Inside this launcher is a mini servo that revolves to set off the missiles. The wires that control this servo are white (GND) and yellow (+5V). You'll also find black and red wires, which are for a single shot, but we'll use only yellow and white for a continuous Gatling gun effect.

THE BUILD

1. First we'll prepare the toy missile launcher. Carefully remove the four wires from the small plastic socket; they should come out fairly easily. You can use a male-to-male jumper wire to push down on the plastic clip to help.
2. The core of the wire is stranded and quite flimsy, so strip the end of the yellow and white wires and solder them to separate solid-core wires that can be inserted into the Arduino, as shown in [Figure 20-3](#). Trim the black and red wires or tape them out of the way.

2. **FIGURE 20-3:**
Stripping and soldering the missile launcher wires



3. Glue the servo motor's arm to the base of the missile launcher, as shown in [Figure 20-4](#).

3. **FIGURE 20-4:**
Gluing the servo motor's arm



4. Attach the ultrasonic sensor to the top of the launcher, as shown in [Figure 20-5](#). You can use a hot-glue gun for a solid connection or just tape it for now if you might want to alter it later.

4. **FIGURE 20-5:**
Attaching the ultrasonic sensor



5. Use the jumper wires to connect the ultrasonic sensor to the Arduino: connect Trig directly to Arduino pin 13, and Echo directly to Arduino pin 12. We will use a mini breadboard to assist with multiple power connections to Arduino +5V and GND.

ULTRASONIC SENSOR	ARDUINO
VCC	+5V
Trig	Pin 13
Echo	Pin 12
GND	GND

6. Connect the servomotor's brown wire to Arduino GND and the red wire to +5V via the mini breadboard, and the yellow/white wire directly to Arduino pin 9.

SERVO	ARDUINO
Brown wire	GND
Red wire	+5V
Yellow wire	Pin 9

7. Connect the launcher's white wire to the GND rail of the mini breadboard, and the yellow wire directly to Arduino pin 3.

LAUNCHER	ARDUINO
White wire	GND
Yellow wire	Pin 3

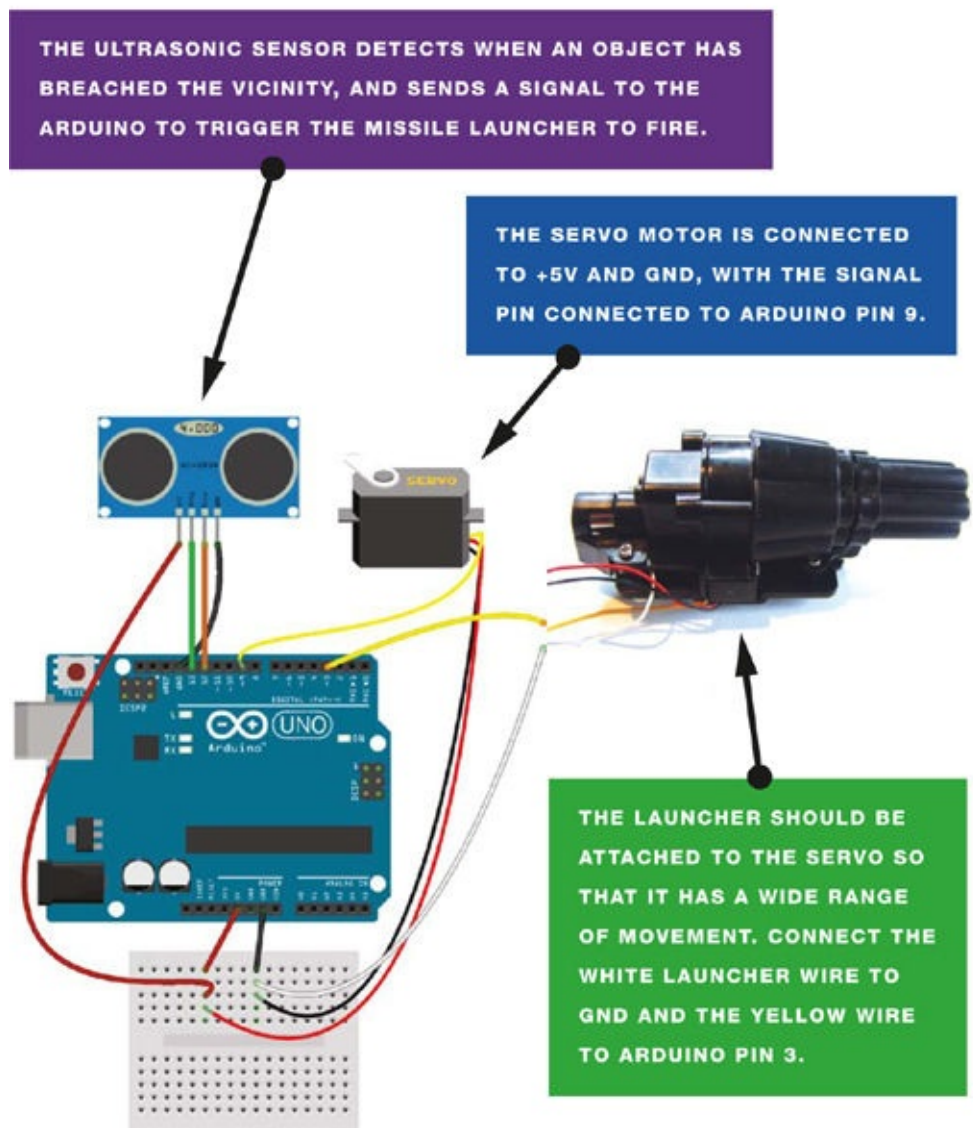
8. Your sentry gun should look like [Figure 20-6](#). Insert the missiles into the launcher.

8. **FIGURE 20-6:**
Your sentry gun is ready to fire!



9. Confirm that your completed setup matches that of [Figure 20-7](#). Upload the code in “[The Sketch](#)” on [page 176](#).

9. **FIGURE 20-7:**
The circuit diagram for the sentry gun



THE SKETCH

The sketch first calls the NewPing and Servo libraries to access the functions you'll need to control the servomotor and ultrasonic sensor, respectively. (Make sure the NewPing library is downloaded from <http://nostarch.com/arduinohandbook/> and saved in your Arduino folder.) The servomotor sweeps back one way and then forth the other, moving the ultrasonic sensor 180 degrees. The sensor sends out an ultrasonic signal, or *ping*, and when this ping reaches an object, it echoes back to give a time value. The Arduino converts this value into the distance between the sensor and the object. When the distance to the object is fewer than 15 centimeters, the servo stops and power is sent to the launcher to fire the bullets at the object. You can change this trigger distance (given in centimeters) at ❶.

```
#include <NewPing.h> // Call NewPing library
#include <Servo.h>    // Call Servo library
#define trigPin 12   // Pin connected to ultrasonic sensor Trig
#define echoPin 13  // Pin connected the ultrasonic sensor Echo
#define MAX_DISTANCE 500
```

```

NewPing sonar(trigPin, echoPin, MAX_DISTANCE);

int blaster = 3; // Pin connected to the blaster

int angle = 0; // Set servo position in degrees

Servo servo;

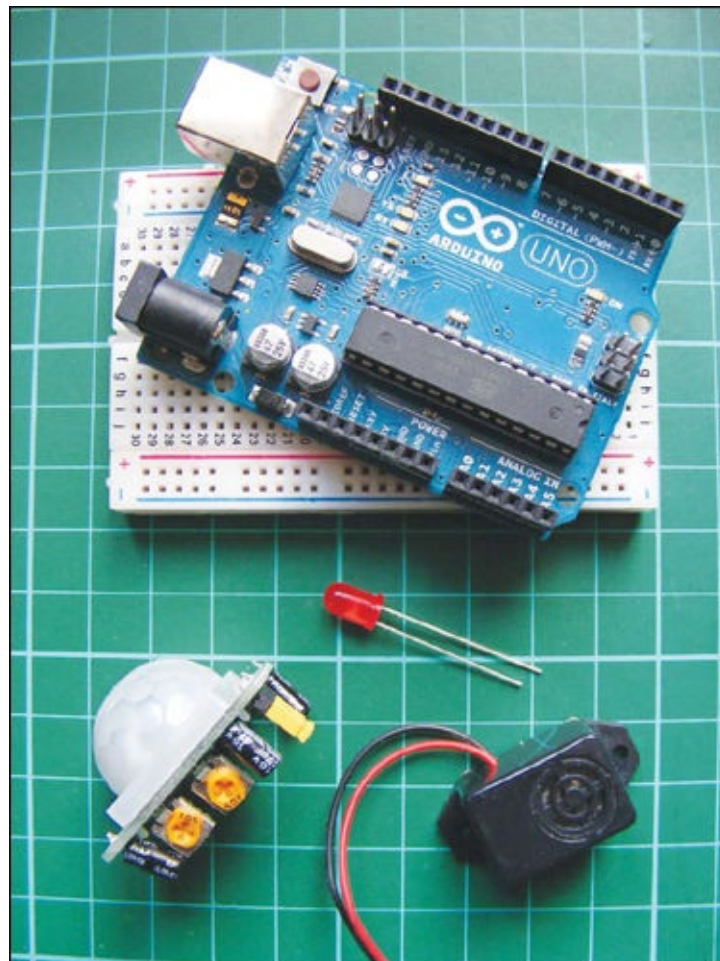
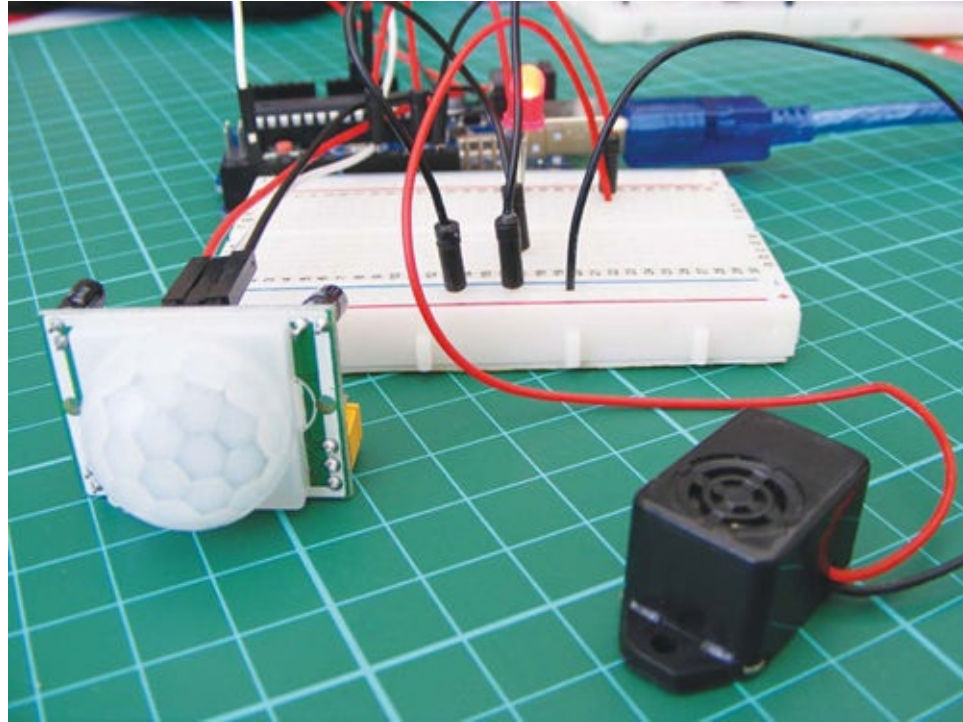
void setup() {
  Serial.begin (115200);
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  pinMode(blaster, OUTPUT);
  servo.attach(9); // Pin connected to servo
}

void loop() {
  int duration, distance, pos = 0, i;
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH); // trigPin sends a ping
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  duration = pulseIn(echoPin, HIGH); // echoPin receives the ping
  distance = (duration / 2) / 29.1;
  Serial.print(distance);
  Serial.println(" cm");
  ❶ if (distance <= 15) { // If distance is fewer than 15 cm
    digitalWrite(blaster, HIGH); // Blaster will fire
    servo.write(90);
  }
  else {
    digitalWrite(blaster, LOW); // Otherwise, blaster won't activate
    for (angle = 0; angle < 180; angle++) { // Sweep the servo
      servo.write(angle);
      delay(15);
    }
    for (angle = 180; angle > 0; angle--) {
      servo.write(angle);
    }
    delay(450);
  }
}

```

PROJECT 21: MOTION SENSOR ALARM

IN THIS PROJECT, WE'LL BUILD A MOTION-SENSING ALARM USING A PASSIVE INFRARED (PIR) SENSOR.



PARTS REQUIRED

- Arduino board
- Breadboard
- HC SR501 PIR sensor
- LED
- Piezo buzzer

You can use this alarm to trigger a variety of outputs, such as lights, motors, or even a “welcome home” message when you approach your front door.

HOW IT WORKS

This project is based on the HC SR501 PIR sensor, which is widely available online for a few dollars. We’re going to set it up so that when someone passes in front of the PIR sensor, the LED will light up and the piezo buzzer will sound (see [Figure 21-1](#)), but you can adapt it for various other output.

FIGURE 21-1:

Any piezo buzzer will work for this project, but remember that most have polarity, so the red wire must be connected to +5V and the black wire to GND.



Other similar PIR sensors will work with this code, but it's important to check the pin layout of your sensor on the data sheet, as this can vary. All sensors should have +5V, GND, and output pins. On this model, the pins are not clearly marked, but if you simply remove the outer lens (it's clipped in place and can be unclipped easily), you can identify the pins underneath, as shown in [Figure 21-2](#).

FIGURE 21-2:

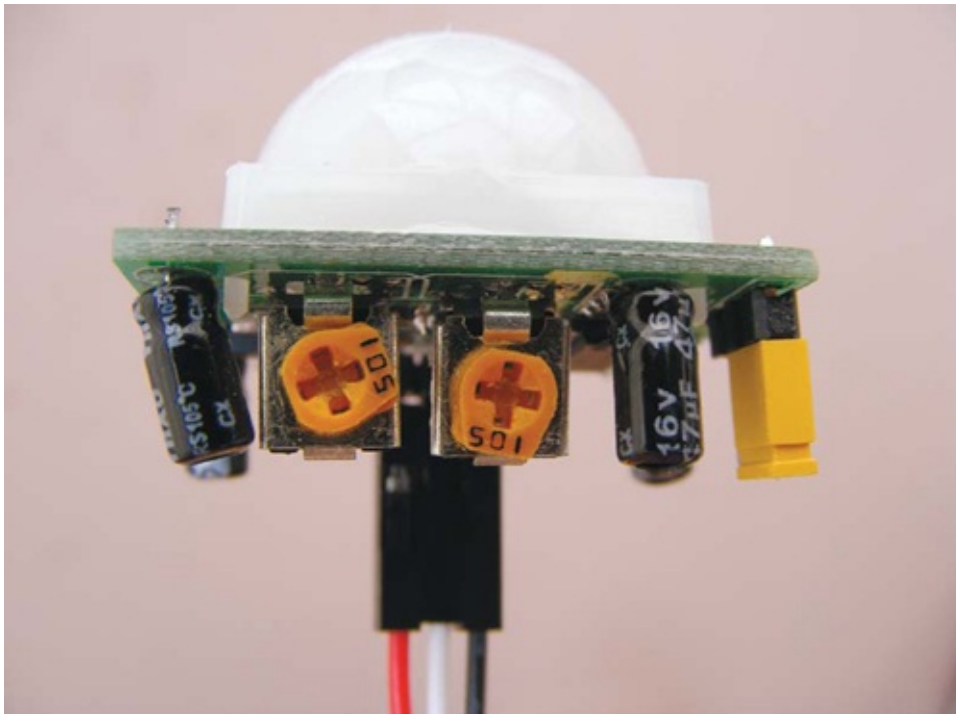
A PIR sensor with the lens removed



The two orange potentiometers on the sensor indicate that there are two adjustable settings. With the sensor upright, as shown in [Figure 21-3](#), the left potentiometer controls how long the output is set to `HIGH` when something is detected, and can be set between 5 and 200 seconds. When we attach an LED to the output, the LED will be lit for between 5 and 200 seconds depending on the setting. The right potentiometer adjusts the detection range from 0 to 7 meters.

FIGURE 21-3:

PIR sensor potentiometers. The left controls how long the output is set to HIGH (5–200 seconds), while the right controls the range (0–7 meters).



The sensor works by detecting infrared radiation, which is emitted from objects that generate heat. Crystalline material within the sensor detects the infrared radiation, and when it detects a set level, it triggers the output signal of the sensor. The Arduino reads this output as voltage, so we can use this as a simple switch to turn something on—in this instance, an LED.

We are setting up the sensor so that an alarm sounds when the sensor is triggered, but there are other ways that you can customize the project. For example, you could scare your friends by attaching a servo and setting it up to release a rubber band when they walk by.

THE BUILD

1. Connect the PIR sensor's +5V and GND wires to the +5V and GND rails on the breadboard, and connect these rails to the Arduino. Connect the PIR sensor's output wire to Arduino pin 2. (See [Figure 21-4](#).)

PIR SENSOR	ARDUINO
+5V	+5V
GND	GND
Output	Pin 2

1. **FIGURE 21-4:**
PIR sensor connected to wires



2. Insert an LED into the breadboard and connect the long, positive leg to Arduino pin 13, and the short, negative leg to GND. You don't need a resistor for the LED in this project.

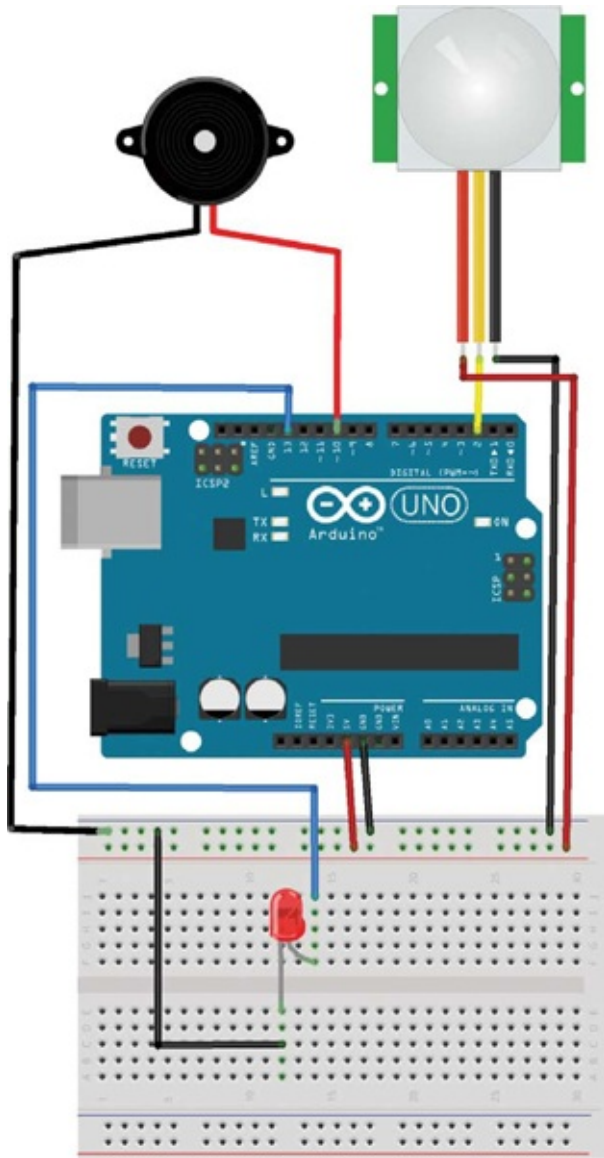
LED	ARDUINO
Positive leg	Pin 13
Negative leg	GND

3. Connect the piezo buzzer by attaching the red wire to Arduino pin 10 and the black wire to GND.

PIEZO	ARDUINO
Red wire	Pin 10
Black wire	GND

4. Confirm that your setup matches the circuit diagram in [Figure 21-5](#), and then upload the code in “[The Sketch](#)” on [page 183](#).

4. **FIGURE 21-5:**
The circuit diagram for the motion sensor alarm



THE SKETCH

The sketch works by setting Arduino pin 13 as output for the LED, pin 2 as input for the PIR sensor, and pin 10 as output for the piezo buzzer. When the PIR sensor is triggered, a HIGH signal is sent to the Arduino, which will in turn light the LED and play a tone on the piezo buzzer.

```
int ledPin = 13;           // Pin connected to LED
int inputPin = 2;         // Pin connected to PIR sensor
int pirState = LOW;      // Start PIR state LOW with no motion
int val = 0;             // Variable for reading the pin status
int pinSpeaker = 10;     // Pin connected to piezo

void setup() {
  pinMode(ledPin, OUTPUT); // Set LED as output
  pinMode(inputPin, INPUT); // Set sensor as input
  pinMode(pinSpeaker, OUTPUT);
  Serial.begin(9600);
}

void loop() {
```

```
val = digitalRead(inputPin); // Read PIR input value
if (val == HIGH) { // Check if input is HIGH
    digitalWrite(ledPin, HIGH); // If it is, turn ON LED
    playTone(300, 160);
    delay(150);
    if (pirState == LOW) {
        // Print to the Serial Monitor if motion detected
        Serial.println("Motion detected!");

        pirState = HIGH;
    }
} else {
    digitalWrite(ledPin, LOW); // If input is not HIGH,
                                // turn OFF LED

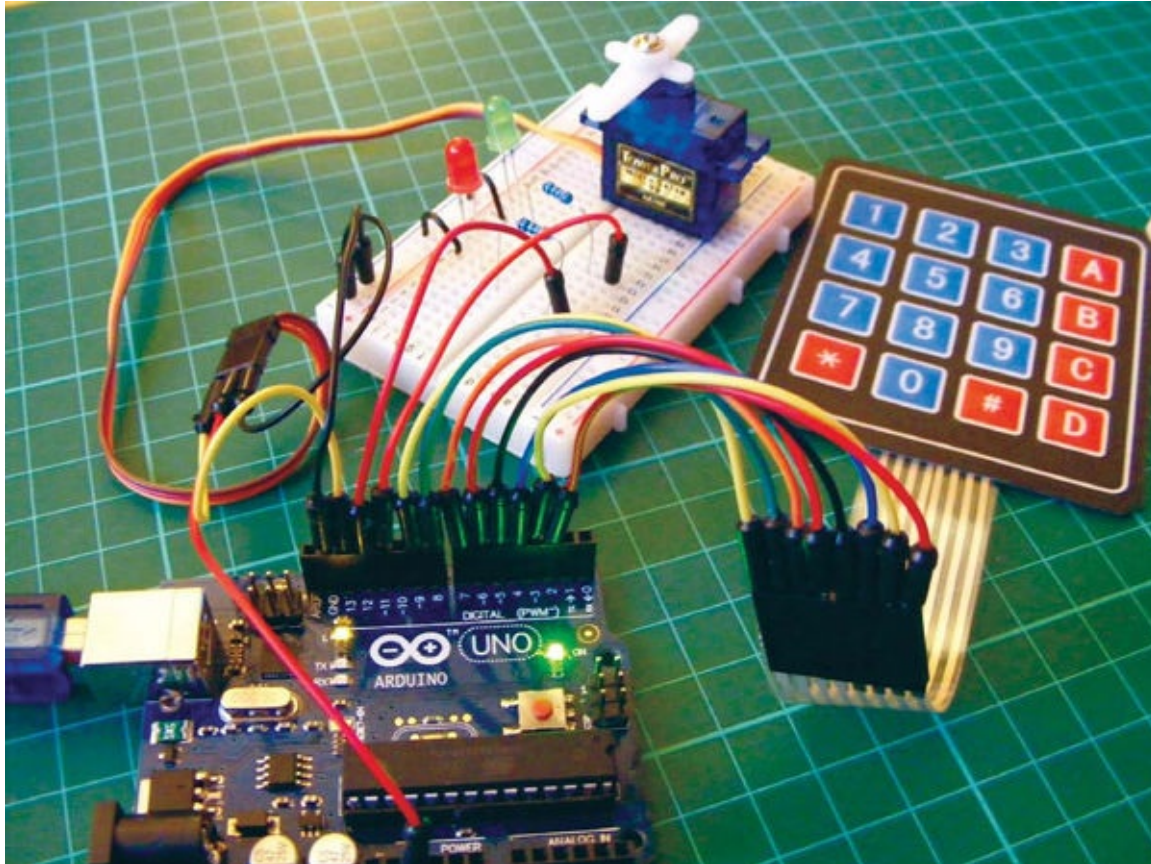
    playTone(0, 0);
    delay(300);
    if (pirState == HIGH) {
        Serial.println("Motion ended!");
        pirState = LOW;
    }
}
}

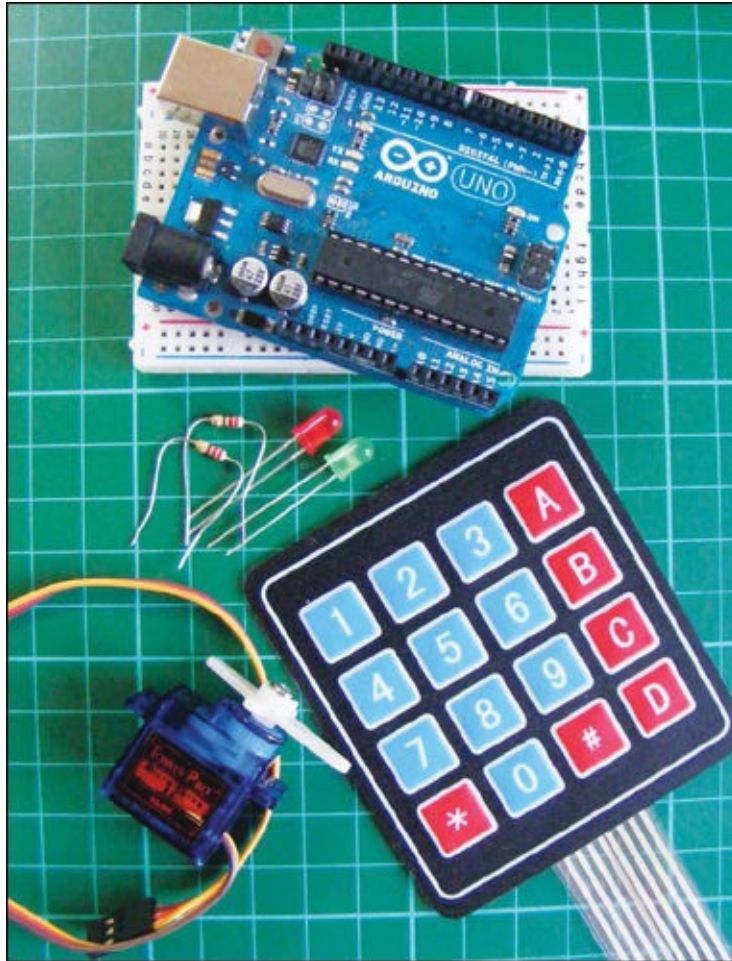
void playTone(long duration, int freq) { // Duration in ms,
                                        // frequency in Hz

    duration *= 1000;
    int period = (1.0 / freq) * 1000000;
    long elapsed_time = 0;
    while (elapsed_time < duration) {
        digitalWrite(pinSpeaker, HIGH);
        delayMicroseconds(period / 2);
        digitalWrite(pinSpeaker, LOW);
        delayMicroseconds(period / 2);
        elapsed_time += (period);
    }
}
```

PROJECT 22: KEYPAD ENTRY SYSTEM

IT'S TIME TO INTRODUCE A KEYPAD TO YOUR ARDUINO BY BUILDING A KEYPAD ENTRY SYSTEM.





PARTS REQUIRED

- Arduino board
- Breadboard
- Jumper wires
- Tower Pro SG90 9g servomotor
- Green LED
- Red LED
- 4x4 membrane keypad
- 2 220-ohm resistors

LIBRARIES REQUIRED

- Keypad
- Servo
- Password

This project uses a 4×4 membrane keypad with a ribbon of eight wires running from the bottom, connected to a servo that sweeps to open a lock.

HOW IT WORKS

A keypad is basically a series of buttons that output a number or character depending on which button is pressed. With the keypad face up, the wires are numbered 1–8 from left to right. The first four wires correspond to the rows, and the latter four to the columns.

You'll need to download the library for the keypad from the <http://nostarch.com/arduinohandbook/> and save it in your IDE's Arduino libraries folder.

We'll connect this keypad to a servo and some LEDs to create a lock system like the secret knock lock in [Project 9](#). To use the lock, enter your code and press the asterisk (*) to confirm. If the code matches the password defined in the sketch, the green LED will flash and the servo will move 90 degrees. If the code is incorrect, the red LED will flash. Use the hash key (#) to reset between code inputs. You could swap this servo for a more substantial one capable of unlocking a heavier deadbolt on a door, or locking and unlocking a box from the inside with the keypad and LEDs mounted externally.

TESTING THE KEYPAD

First we'll test the keypad with the following code:

```
#include <Keypad.h>

const byte ROWS = 4;
const byte COLS = 4;
char keys[ROWS][COLS] = {
  {'1', '2', '3', 'A'},
  {'4', '5', '6', 'B'},
  {'7', '8', '9', 'C'},
  {'*', '0', '#', 'D'}
};
byte rowPins[ROWS] = {2,3,4,5};
byte colPins[COLS] = {6,7,8,9};

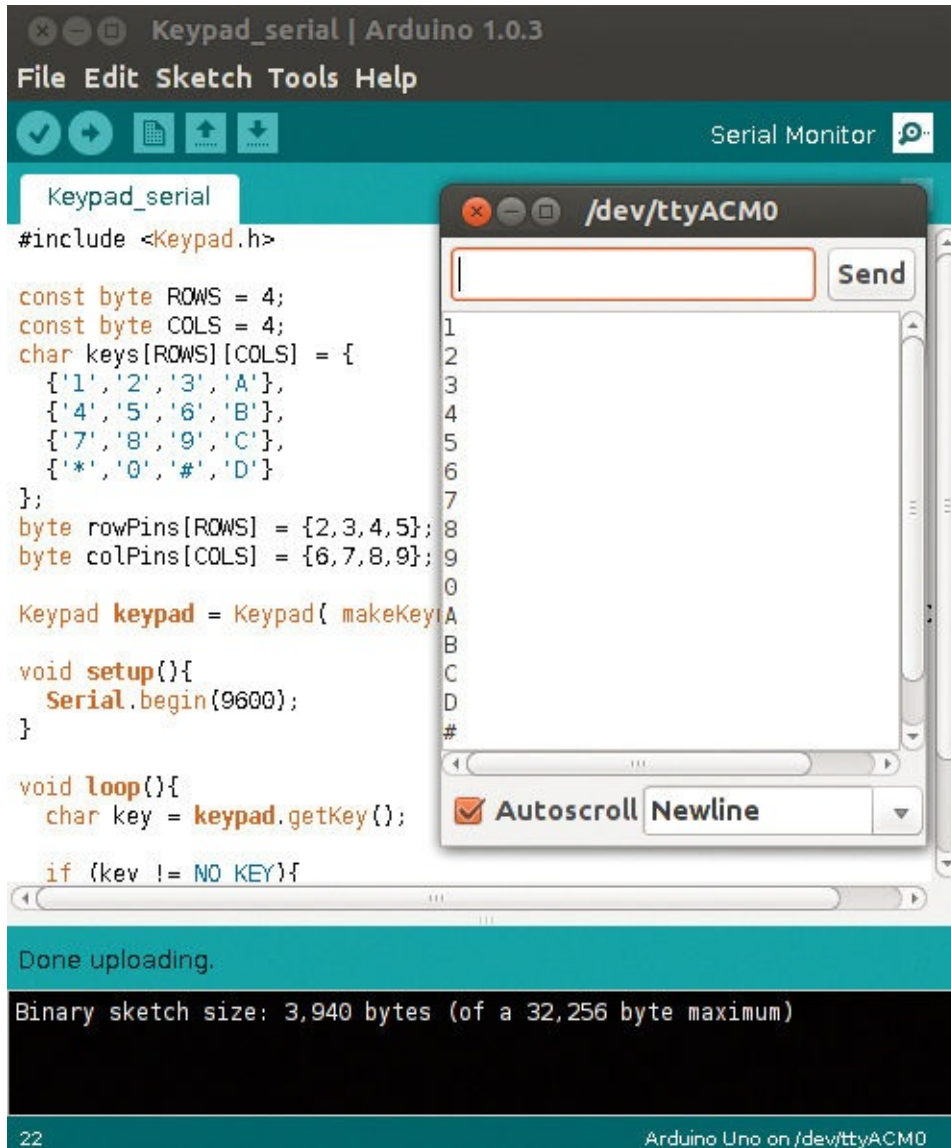
Keypad keypad = Keypad(makeKeymap(keys), rowPins, colPins,
                       ROWS, COLS);

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

void loop() {
  char key = keypad.getKey();
  if (key != NO_KEY){
    Serial.println(key);
  }
}
```

Upload this code and then open the Serial Monitor in your IDE ([Figure 22-1](#)).

FIGURE 22-1:
Testing the keypad



With the keypad face up, connect the wires in sequence from left to right to Arduino digital pins 9–2. Once you have uploaded the code, press a few keys. As each key is pressed, the corresponding character should appear on a separate line in the Arduino IDE’s serial console.

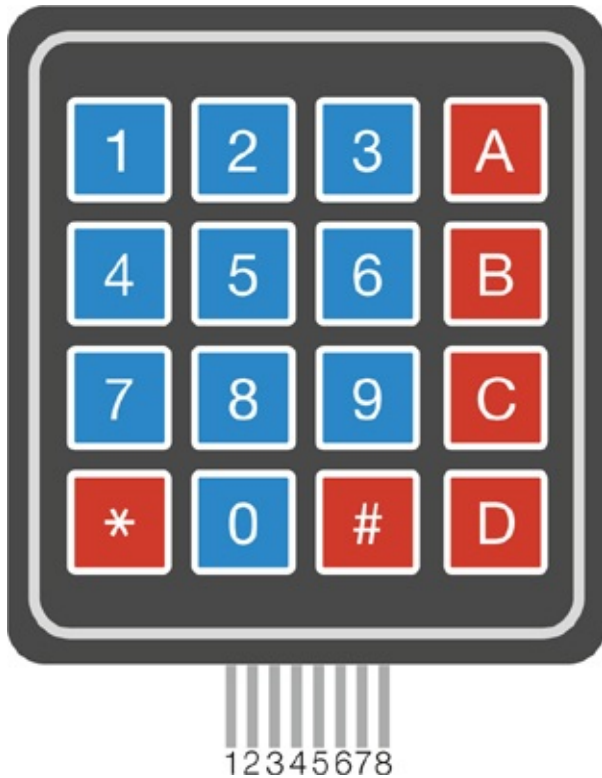
THE BUILD

1. Connect the pins of the keypad directly to the Arduino pins as follows. The keypad pins are numbered as shown in [Figure 22-2](#).

KEYPAD	ARDUINO
Pin 1	Pin 9
Pin 2	Pin 8
Pin 3	Pin 7

Pin 4	Pin 6
Pin 5	Pin 5
Pin 6	Pin 4
Pin 7	Pin 3
Pin 8	Pin 2

- FIGURE 22-2:**
Keypad pins 1–8



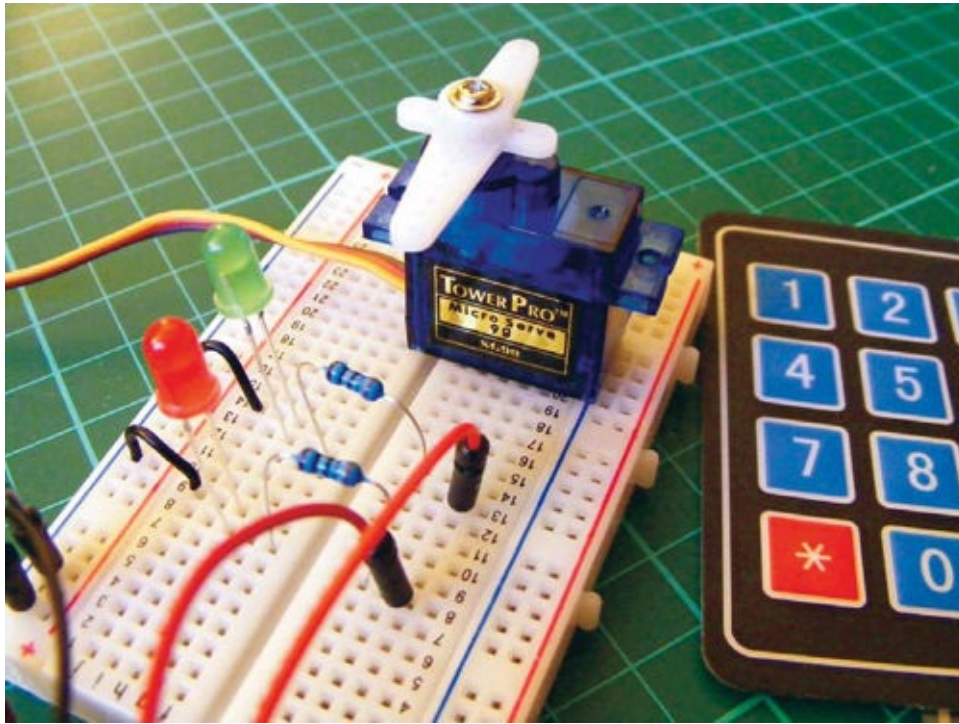
- Place a green LED and a red LED into the breadboard with the shorter, negative legs connected to the Arduino GND rail. Add a 220-ohm resistor to each longer, positive leg. Connect the resistor that's attached to the green LED to Arduino pin 11, and the resistor that's attached to the red LED to Arduino pin 12.

LEDS	ARDUINO
Positive legs	Pins 11 and 12 via 220-ohm resistors
Negative legs	GND

- Now attach the servo (see [Figure 22-3](#)). Connect the brown wire to the GND rail, the red wire to the +5V rail, and the yellow/white wire directly to pin 13 on the Arduino.

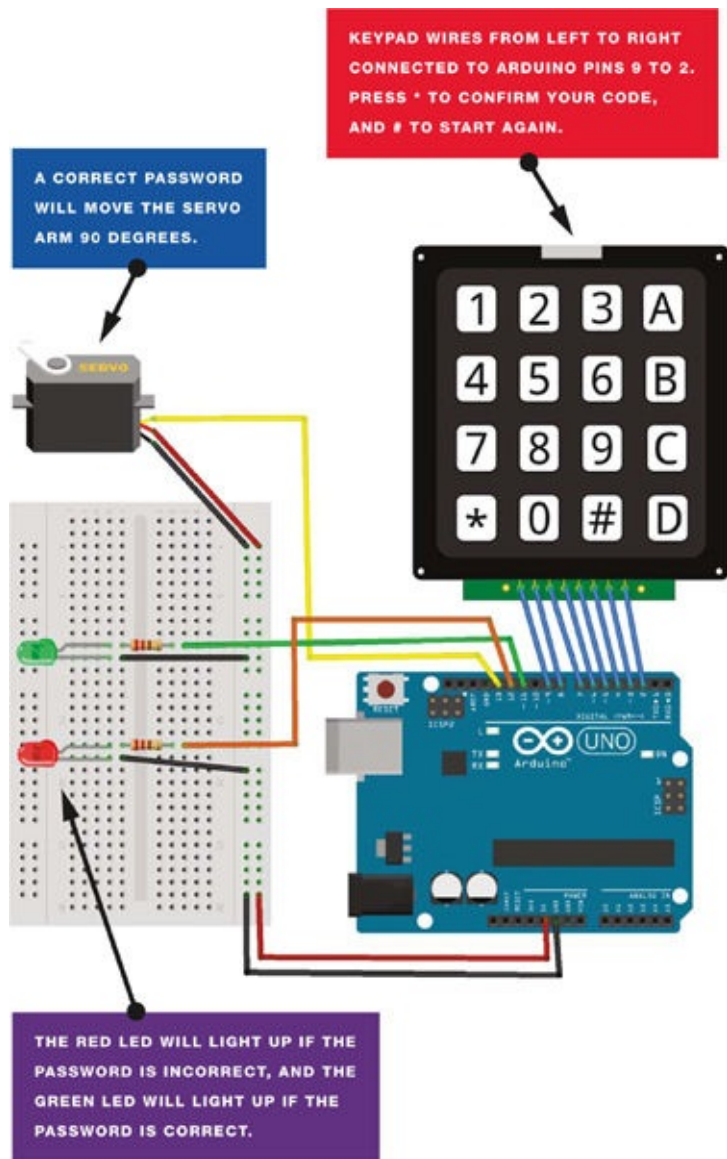
SERVO	ARDUINO
Brown wire	GND
Red wire	+5V
Yellow wire	Pin 13

3. **FIGURE 22-3:**
Attaching the servo



4. Make sure your setup matches that of [Figure 22-4](#), and upload the code in “[The Sketch](#)” on [page 192](#).

4. **FIGURE 22-4:**
Circuit diagram for the keypad entry system



THE SKETCH

First, the sketch calls on the Keypad, Servo, and Password libraries. The Servo library is included in the IDE, but you'll have to download the Keypad and Password libraries (<http://nostarch.com/arduinohandbook/>). We then set the eight pins that will determine the input from the keypad, and set Arduino pins 11 and 12 to control the LEDs and pin 13 to control the servomotor. The Arduino waits for your code input from the keypad and for you to confirm your input with *. Once you've pressed the asterisk key, the sketch will check the entry against the password in the code. If the entry doesn't match the password, the red LED will be set to HIGH and light; if the entry *does* match the password, the green LED will be set to HIGH and light, and the servomotor will turn. Pressing # will reset the sketch so it's ready for another entry.

To alter the password, change the number in quotation marks in the following line.

```
Password password = Password("2468");
```

The default password in the sketch is 2468.

```

/* Keypad Library for Arduino
   Authors: Mark Stanley, Alexander Brevig
   http://playground.arduino.cc/Main/KeypadTutorial
*/

#include <Password.h>
#include <Keypad.h>
#include <Servo.h>

Servo myservo;
Password password = Password("2468"); // Set password

const byte ROWS = 4; // Set four rows
const byte COLS = 4; // Set four columns

char keys[ROWS][COLS] = { // Define the keypad
  {'1','2','3','A'},
  {'4','5','6','B'},
  {'7','8','9','C'},
  {'*','0','#','D'}
};
byte rowPins[ROWS] = { 9,8,7,6 }; // Pins connected to keypad
// ROW0, ROW1, ROW2 and ROW3
byte colPins[COLS] = { 5,4,3,2, }; // Pins connected to keypad
// COL0, COL1 and COL2

// Create the keypad
Keypad keypad = Keypad(makeKeypad(keys), rowPins, colPins,
                       ROWS, COLS);

void setup() {
  Serial.begin(9600);
  delay(200);
  pinMode(11, OUTPUT); // Set green LED as output
  pinMode(12, OUTPUT); // Set red LED as output
  myservo.attach(13); // Pin connected to servo
  keypad.addEventListener(keypadEvent); // Add an event listener to
// detect keypresses
}

void loop() {
  keypad.getKey();
  myservo.write(0);
}

void keypadEvent(KeypadEvent eKey) {
  switch (keypad.getState()) {
    case PRESSED:
      Serial.print("Pressed: ");
      Serial.println(eKey);
      switch (eKey) {
        case '*': checkPassword(); break;
        case '#': password.reset(); break;
        default: password.append(eKey);
      }
    }
}

void checkPassword() {
  if (password.evaluate() ){
    Serial.println("Success"); // If the password is correct...
    myservo.write(90); // Move servo arm 90 degrees
    digitalWrite(11, HIGH); // Turn on green LED
    delay(500); // Wait 5 seconds
    digitalWrite(11, LOW); // Turn off green LED
  }
}

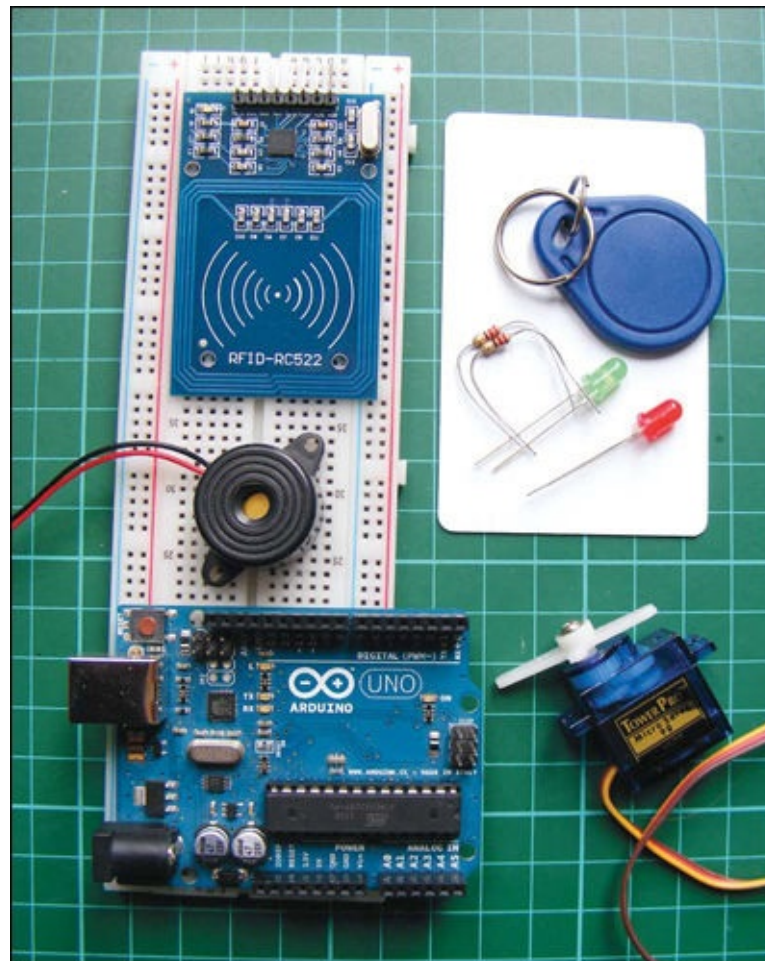
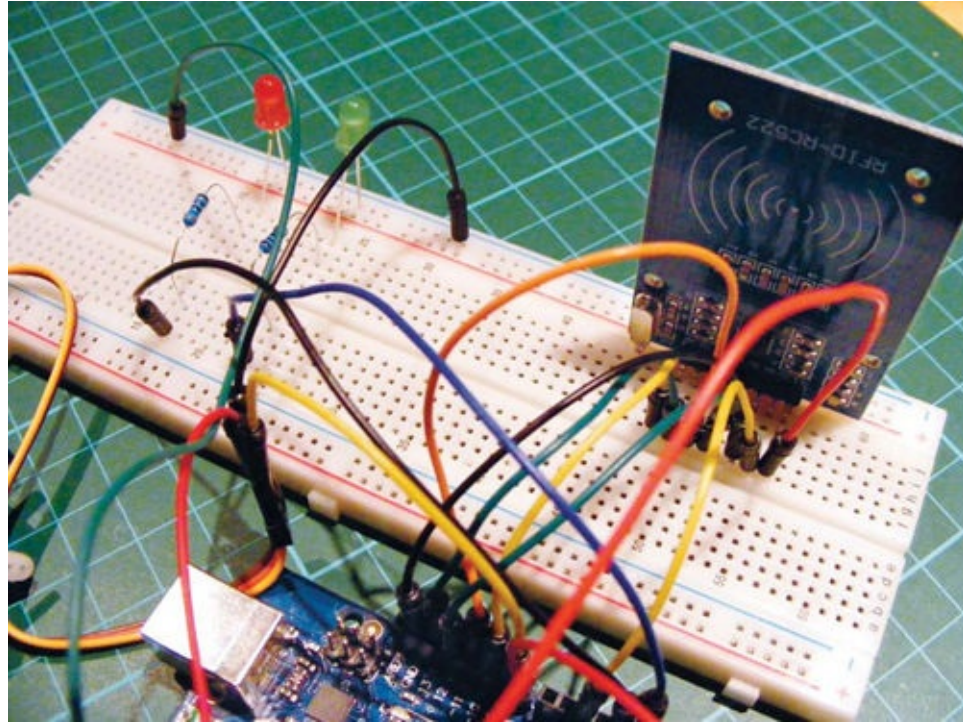
```



```
} else {  
  Serial.println("Wrong"); // If the password is incorrect...  
  myservo.write(0);  
  digitalWrite(12, HIGH); // Turn on red LED  
  delay(500); // Wait 5 seconds  
  digitalWrite(12, LOW); // Turn off red LED  
}  
}
```

PROJECT 23: WIRELESS ID CARD ENTRY SYSTEM

IN THIS PROJECT, WE'LL USE A RADIO FREQUENCY IDENTIFICATION (RFID) READER TO BUILD A WIRELESS ID CARD ENTRY SYSTEM.



PARTS REQUIRED

- Arduino board
- Breadboard
- Jumper wires
- Mifare RFID-RC522 module
- Tower Pro SG90 9g servomotor
- Piezo buzzer
- Red LED
- Green LED
- 2 220-ohm resistors

LIBRARIES REQUIRED

- RFID
- SPI
- Wire
- Servo
- Pitches

HOW IT WORKS

An RFID reader uses wireless technology to identify a card, tag, or key fob without contact. The reader will respond when the card is placed near it. First, we need the reader to read the unique number of our RFID card, and then we'll add a servo that will move depending on whether the RFID reader recognizes the card. We could use this ID system for something like a door or box lock, as with the secret knock code lock in [Project 9](#).

You may have seen a sticker like the one in [Figure 23-1](#) on an item you have purchased. These stickers use RFID to allow the store to track items for security purposes. If you pass through the RFID field at the exit without paying, the stickers will set off the alarm. RFID readers and cards are also often used as identification to allow access into restricted areas, like top-secret labs or gated communities.

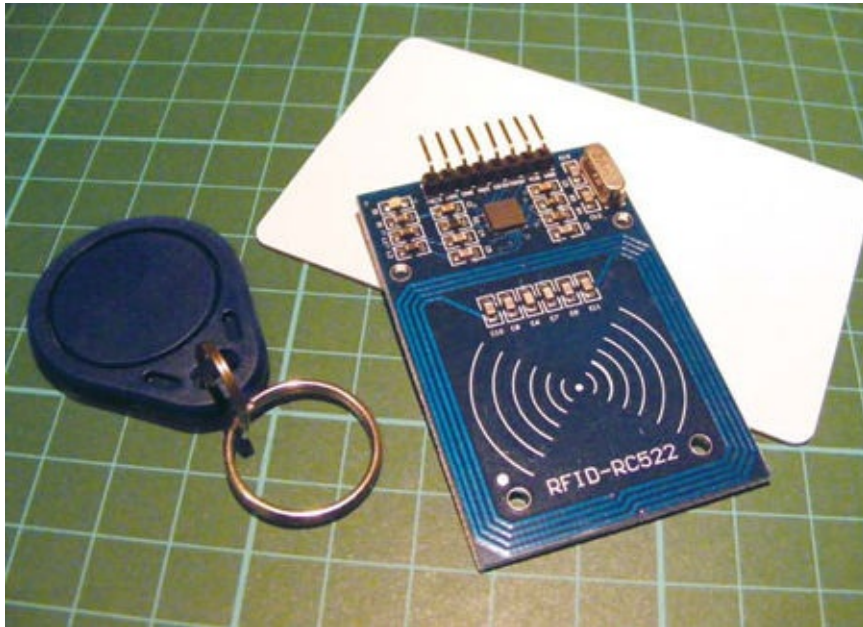
FIGURE 23-1:
An RFID sticker



There are two types of RFID: passive and active. Each RFID system uses a radio frequency to exchange a signal between the reader and the tag or card. This signal contains the tag or card's unique code, and if the RFID reader recognizes that code, it reacts appropriately—for example, by allowing the item to pass through the detectors in a store or by unlocking a door.

In a passive system, when the two are passed close to each other, the reader's radio signal powers the circuit in the tag or card just enough for them to exchange data. Active systems have a powered reader and a powered tag and can read tags accurately from much farther away. Active systems are very expensive and used for more sophisticated applications, so we'll be using a passive RFID system: the Mifare RFID-RC522 reader, which comes with a blank card and key fob, like those shown in [Figure 23-2](#). The reader operates at 13.56 MHz, which means it can identify the card or fob, each of which is powered by the reader, only if it is less than a few inches away. It's important to keep this in mind when positioning your reader.

FIGURE 23-2:
RFID reader with card and key fob



We'll create an RFID-controlled servo. When you pass your card in front of the RFID reader, it reads the card. If the module recognizes the card and the card has access rights, the green LED lights up, a tune plays, and the servomotor moves 180 degrees. If the module does not recognize the card, the red LED lights up, a different tune plays, and the servo does not move.

Table 23-1 describes the various functions of the RFID reader.

TABLE 23-1:
Functions of the RFID reader pins

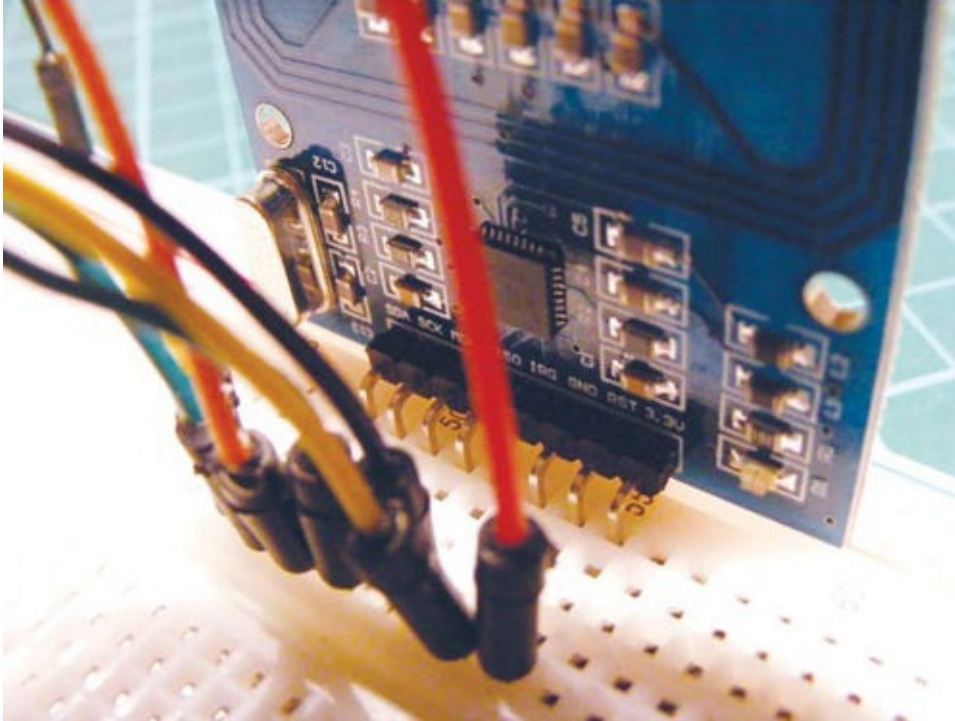
RFID	DETAIL	NOTES
3.3V	3.3 volts	The module can use only this amount of voltage.
RST	Reset	Will clear the module to initial state.
GND	Ground	Connects to the Arduino GND pin.
IRQ	Interrupt Request	Not used in this project.
MISO	Master In Slave Out	Sometimes referred to as “data in.”
MOSI	Master Out Slave In	Sometimes referred to as “data out.”
SCK	Serial Clock	Output from master. This creates a pulse that synchronizes data, usually set by the master.

SDA/SS	Serial Data/Slave Select	Modules will have either SDA or SS, although they are the same. This is how the Arduino and module share data and communicate.
Pin 16	VCC	Positive power.

THE BUILD

1. You may need to set up the module by soldering the header pins to it first. Snap off a strip of eight header pins. Solder one header pin to each point. Make sure to hold the solder iron in place for only a few seconds so you don't damage the circuits. See the "[Quick Soldering Guide](#)" on [page 18](#) for a primer on soldering.
2. Place your RFID module into a breadboard as shown in [Figure 23-3](#), and then connect the RFID pins to the Arduino pins as indicated in the following table. Remember to connect the RFID board to 3.3V power on the Arduino (not +5V), or you will damage the module.

2. **FIGURE 23-3:**
Placing the RFID module into the breadboard



RFID	ARDUINO
3.3V	3.3V
RST	Pin 5
GND	GND
IRQ	Not used
MISO	Pin 12
MOSI	Pin 11
SCK	Pin 13
SDA	Pin 10

3. Now we need to check that the RFID module is working. Download the RFID library from <http://www.nostarch.com/arduinohandbook/> and save it in your *libraries* directory (see “**Libraries**” on [page 7](#) for details on downloading libraries). Upload the following test sketch for the RFID reader. Keep the USB cable from your PC connected to the Arduino.

```
// RFID Library Created by Miguel Balboa (circuitito.com)
#include <SPI.h>
#include <RFID.h>
#define SS_PIN 10
```

```

#define RST_PIN 9
RFID rfid(SS_PIN, RST_PIN);

// Setup variables
int serNum0;
int serNum1;
int serNum2;
int serNum3;
int serNum4;

void setup() {
  Serial.begin(9600);
  SPI.begin();
  rfid.init();
}

void loop() { // This loop looks for a card(s) to read
  if (rfid.isCard()) {
    if (rfid.readCardSerial()) {
      if (rfid.serNum[0] != serNum0
          && rfid.serNum[1] != serNum1
          && rfid.serNum[2] != serNum2
          && rfid.serNum[3] != serNum3
          && rfid.serNum[4] != serNum4
      ) {
        // When a card is found, the following code will run
        Serial.println(" ");
        Serial.println("Card found");
        serNum0 = rfid.serNum[0];
        serNum1 = rfid.serNum[1];
        serNum2 = rfid.serNum[2];
        serNum3 = rfid.serNum[3];
        serNum4 = rfid.serNum[4];

        // Print the card ID to the Serial Monitor of the IDE
        Serial.println("Cardnumber:");
        Serial.print("Dec: ");
        Serial.print(rfid.serNum[0], DEC);
        Serial.print(", ");
        Serial.print(rfid.serNum[1], DEC);
        Serial.print(", ");
        Serial.print(rfid.serNum[2], DEC);
        Serial.print(", ");
        Serial.print(rfid.serNum[3], DEC);
        Serial.print(", ");
        Serial.print(rfid.serNum[4], DEC);
        Serial.println(" ");
        Serial.print("Hex: ");
        Serial.print(rfid.serNum[0], HEX);
        Serial.print(", ");
        Serial.print(rfid.serNum[1], HEX);
        Serial.print(", ");
        Serial.print(rfid.serNum[2], HEX);
        Serial.print(", ");
        Serial.print(rfid.serNum[3], HEX);
        Serial.print(", ");
        Serial.print(rfid.serNum[4], HEX);
        Serial.println(" ");
      } else {
        // If the ID matches, write a dot to the Serial Monitor
        Serial.print(".");
      }
    }
  }
}

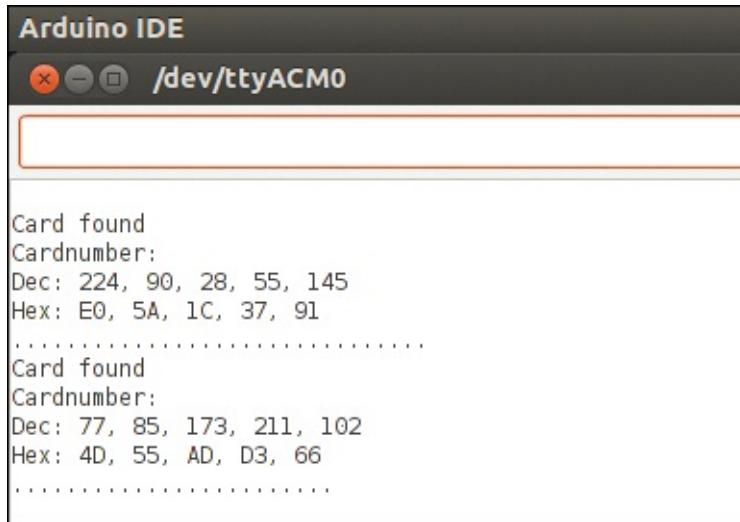
```



```
}  
  rfid.halt();  
}
```

4. Open the Arduino Serial Monitor in your IDE.
5. Pass either your card or key fob in front of the RFID module. The unique number should appear on the Serial Monitor, as shown in [Figure 23-4](#). Write down this number, because you'll need it later. In this case, my card number is 4D 55 AD D3 66.

5. **FIGURE 23-4:**
The RFID number represented in hexadecimal on the screen



6. Insert the two LEDs into the breadboard, with the shorter, negative wires connected to the GND rail. Connect the longer, positive wire on the red LED to Arduino pin 3 via a 220-ohm resistor. Connect the positive leg of the green LED to pin 2 via another 220-ohm resistor.

LEDS	ARDUINO
Negative legs	GND
Positive leg (red)	Pin 3 via 220-ohm resistor
Positive leg (green)	Pin 2 via 220-ohm resistor

7. Connect the servo to the Arduino by attaching the red wire to +5V, the brown (or black) wire to GND, and the yellow wire to Arduino pin 9.

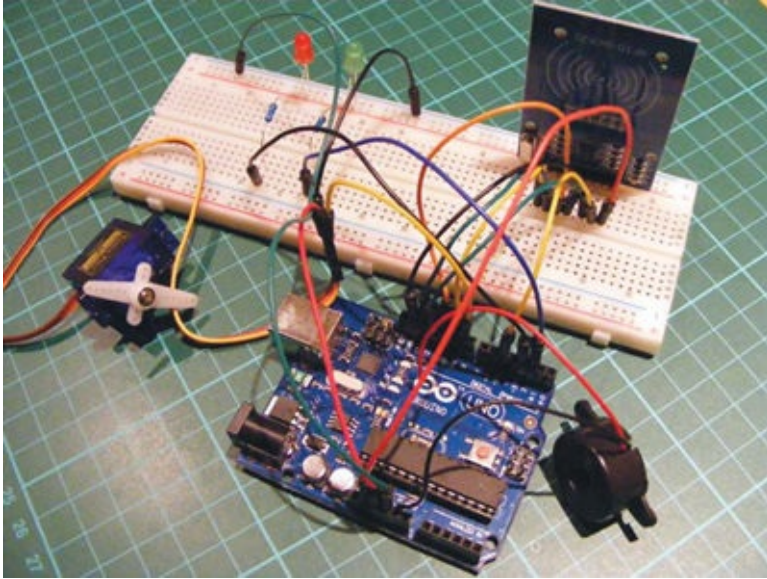
SERVO	ARDUINO
Red wire	+5V
Black wire	GND
Yellow wire	Pin 9

8. Connect the piezo buzzer to the Arduino by attaching the red wire to Arduino pin 8 and the black wire to GND. Your build should now look something like [Figure 23-5](#).

PIEZO	ARDUINO
Red wire	Pin 8

Black wire	GND
------------	-----

8. **FIGURE 23-5:**
Completed RFID project

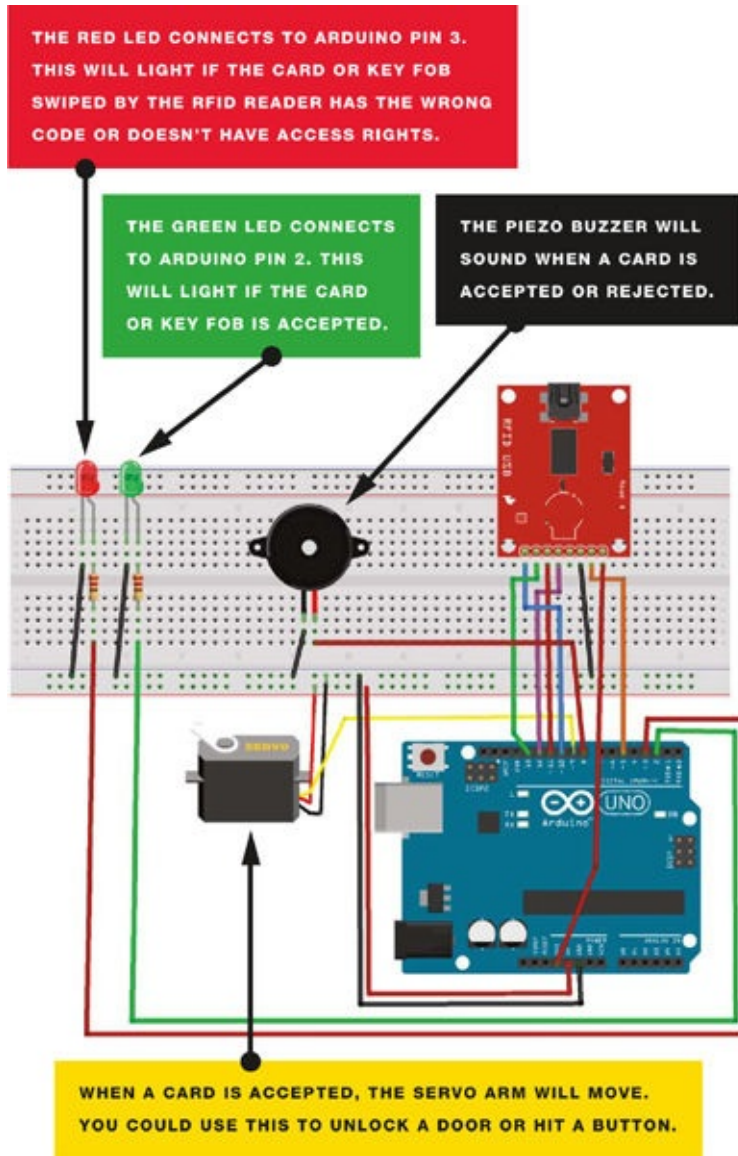


9. Open the project code in your Arduino IDE and change the following line to match the hex number you found for your card or key fob in step 5 using the RFID reader. Leave the `0x` as it appears, but fill in the rest with your number.

```
byte card[5] = {0x4D, 0x55, 0xAD, 0xD3, 0x66};
```

10. Confirm that your setup matches the circuit diagram in [Figure 23-6](#), and then upload the code from “[The Sketch](#)” on [page 203](#) to your Arduino.

10. **FIGURE 23-6:**
Circuit diagram for the wireless ID card entry system



THE SKETCH

The sketch begins by calling on the SPI, RFID, Servo, Pitches, and Wire libraries to control communication between the Arduino, RFID module, and servo. Two melodies are defined, one for a positive reading on your card and the other for a negative reading. The green LED is set to Arduino pin 2, the red LED to pin 3, the piezo buzzer to pin 8, and the servo to pin 9.

The following line is where you add your card's hex value:

```
byte card[5] = {0x4D, 0x55, 0xAD, 0xD3, 0x66};
```

Pass your card in front of the reader. If the hex code on the card matches that in your sketch, the green LED lights up, a tune plays, and the servo moves. The reader rejects all other cards unless you add their number to the code at ❶. If a card is rejected, the red LED lights up and a different tune plays, but the servo does not move.

```
#include <SPI.h>
```

```

#include <RFID.h>
#include <Servo.h>
#include "pitches.h"
#include <Wire.h>

RFID rfid(10, 5); // Define the RFID

// Replace this with the code from your card in hex form
❶ byte card[5] = {0x4D, 0x55, 0xAD, 0xD3, 0x66};
// List any other codes for cards with access here

byte serNum[5];
byte data[5];

// Define the melodies for successful access and denied access
int access_melody[] = {NOTE_G4, 0, NOTE_A4, 0, NOTE_B4, 0, NOTE_A4,
0, NOTE_B4, 0, NOTE_C5, 0};
int access_noteDurations[] = {8, 8, 8, 8, 8, 4, 8, 8, 8, 8, 8, 4};
int fail_melody[] = {NOTE_G2, 0, NOTE_F2, 0, NOTE_D2, 0};
int fail_noteDurations[] = {8, 8, 8, 8, 8, 4};

int LED_access = 2; // Pin connected to green LED
int LED_intruder = 3; // Pin connected to red LED
int speaker_pin = 8; // Pin connected to piezo buzzer
int servoPin = 9; // Pin connected to servo

Servo doorLock; // Define the servomotor

void setup() {
  doorLock.attach(servoPin); // Set servo as a pin
  Serial.begin(9600); // Start serial communication
  SPI.begin(); // Start serial communication between the RFID and PC
  rfid.init(); // Initialize the RFID
  Serial.println("Arduino card reader");
  delay(1000);
  pinMode(LED_access, OUTPUT);
  pinMode(LED_intruder, OUTPUT);
  pinMode(speaker_pin, OUTPUT);
  pinMode(servoPin, OUTPUT);
}

void loop() { // Create a variable for each user
  boolean card_card = true; // Define your card
  if (rfid.isCard()) {
    if (rfid.readCardSerial()) {
      delay(1000);
      data[0] = rfid.serNum[0];
      data[1] = rfid.serNum[1];
      data[2] = rfid.serNum[2];
      data[3] = rfid.serNum[3];
      data[4] = rfid.serNum[4];
    }
    Serial.print("Card found - code:");
    for (int i = 0; i < 5; i++) {
      // If it is not your card, the card is considered false
      if (data[i] != card[i]) card_card = false;
    }
    Serial.println();
    if (card_card) { // A card with access permission is found
      Serial.println("Hello!"); // Print to Serial Monitor
      for (int i = 0; i < 12; i++) { // Play welcome music
        int access_noteDuration = 1000 / access_noteDurations[i];
        tone(speaker_pin, access_melody[i], access_noteDuration);
      }
    }
  }
}

```

```
        int access_pauseBetweenNotes = access_noteDuration * 1.30;
        delay(access_pauseBetweenNotes);
        noTone(speaker_pin);
    }
}
else { // If the card is not recognized
    // Print message to Serial Monitor
    Serial.println("Card not recognized! Contact administrator!");
    digitalWrite(LED_intruder, HIGH); // Turn on red LED
    for (int i = 0; i < 6; i++) { // Play intruder melody
        int fail_noteDuration = 1000 / fail_noteDurations[i];
        tone(speaker_pin, fail_melody[i], fail_noteDuration);
        int fail_pauseBetweenNotes = fail_noteDuration * 1.30;
        delay(fail_pauseBetweenNotes);
        noTone(speaker_pin);
    }
    delay(1000);
    digitalWrite(LED_intruder, LOW); // Turn off red LED
}

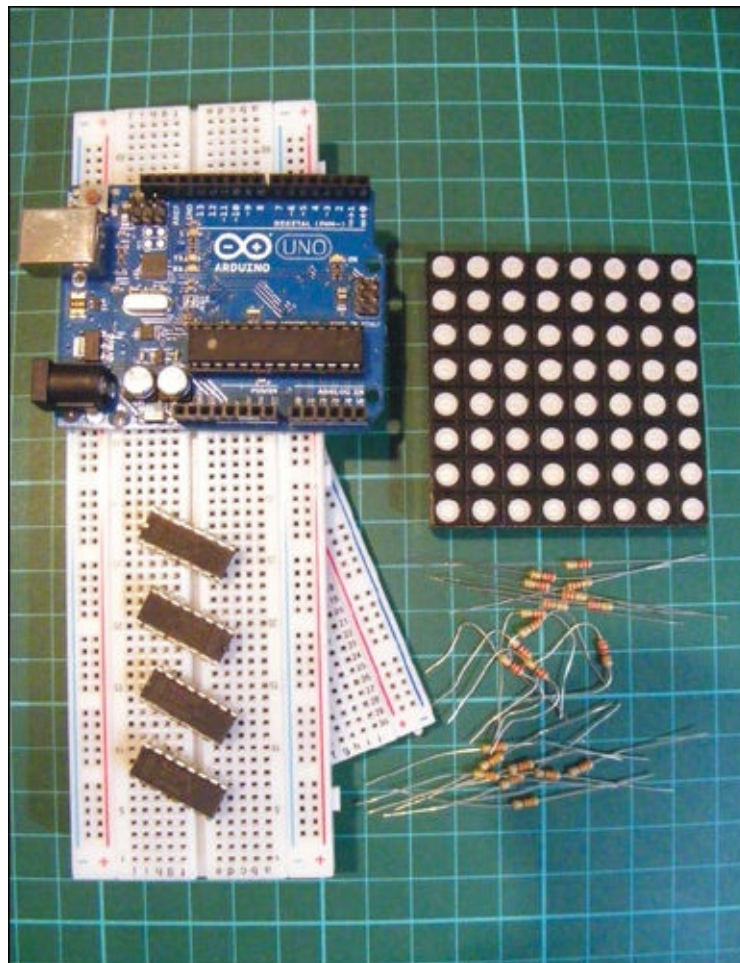
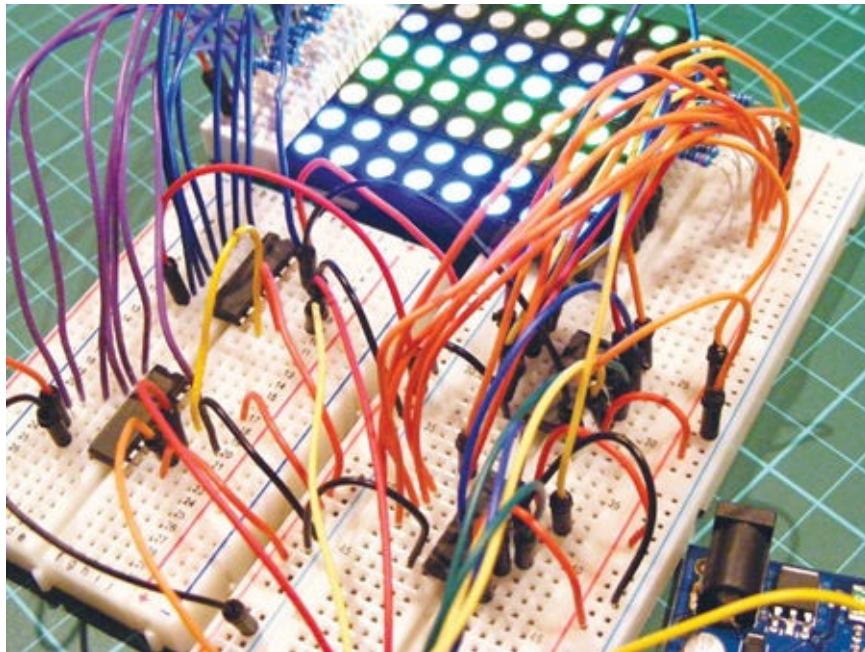
if (card_card) { // Add other users with access here
    Serial.println("Access granted.....Welcome!");
    digitalWrite(LED_access, HIGH); // Turn on green LED
    doorLock.write(180); // Turn servo 180 degrees
    delay(5000); // Wait for 5 seconds
    doorLock.write(0); // Turn servo back to 0 degrees
    digitalWrite(LED_access, LOW); // Turn off green LED
}
Serial.println();
delay(500);
rfid.halt();
}
}
```

PART 7

ADVANCED

PROJECT 24: RAINBOW LIGHT SHOW

IN THIS PROJECT, WE'LL CREATE A RAINBOW LIGHT SHOW USING AN 8×8 RGB LED MATRIX. WE'LL ALSO USE SHIFT REGISTERS TO EXTEND THE ARDUINO AND CONTROL THE MATRIX.



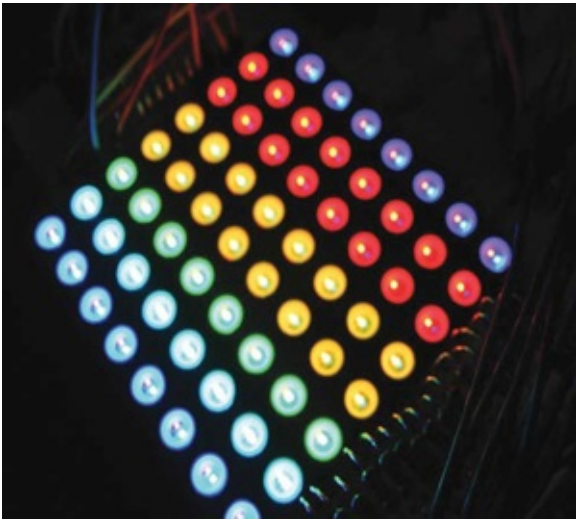
PARTS REQUIRED

- Arduino board
- 2 full-size breadboards
- Jumper wires
- 8×8 RGB LED matrix
- 4 74HC595 shift registers
- 16 220-ohm resistors
- 8 330-ohm resistors

HOW IT WORKS

An RGB LED matrix ([Figure 24-1](#)) is a grid of 64 red, green, and blue LEDs. You can create the colors of the rainbow by controlling each LED individually and by mixing colors together.

FIGURE 24-1:
An RGB LED matrix



The LED matrix has a total of 32 pins ([Figure 24-2](#)); 8 pins control the common-anode positive leg of each LED, and 8 pins apiece control the level of red, green, and blue. In the matrix we've used here, pins 17–20 and 29–32 are the anode pins, 9–16 are for red, 21–28 for green, and 1–8 for blue, but your matrix may have different connections. Pin number 1 will be identified as shown in the bottom-left corner of [Figure 24-2](#)—the pin numbers run clockwise in this image.

FIGURE 24-2:
The pins of an RGB LED matrix



Your matrix should have come with a data sheet that tells you which pins control the red, green, and blue LEDs. If the pin numbers on your data sheet are different from those listed in [Table 24-1](#), follow your data sheet to make the connections to the shift registers and the Arduino. Each color pin requires a resistor to prevent it from overloading and burning out, but the values are slightly different—use 220-ohm resistors for the blue and green, and 330-ohm resistors for the red.

TABLE 24-1:
Pin configuration for an RGB LED matrix

MATRIX PIN FUNCTION	MATRIX PIN NUMBER
Common anode (+)	17, 18, 19, 20, 29, 30, 31, 32
Red LEDs	9, 10, 11, 12, 13, 14, 15, 16
Green LEDs	21, 22, 23, 24, 25, 26, 27, 28
Blue LEDs	1, 2, 3, 4, 5, 6, 7, 8

The layout may look complicated, but that's simply because we're using so many different wires. Just remember to take the project one step at a time.

Because there are so many connections, we'll run out of pins on the Arduino board, so we'll extend the board using *shift registers*. A shift register is a digital memory circuit found in calculators, computers, and data-processing systems. This project uses the 74HC595 shift register to control eight outputs at a time, while taking up only three pins on your Arduino. We'll link multiple registers together to control more pins at once, using one for the common anode and one for each LED color.

The pin layout for the shift register is shown in [Figure 24-3](#), and the functions are described in [Table 24-2](#). When building the project, we'll refer to the pin number of the shift register and function to assist identification.

FIGURE 24-3:
Pin layout for the shift register

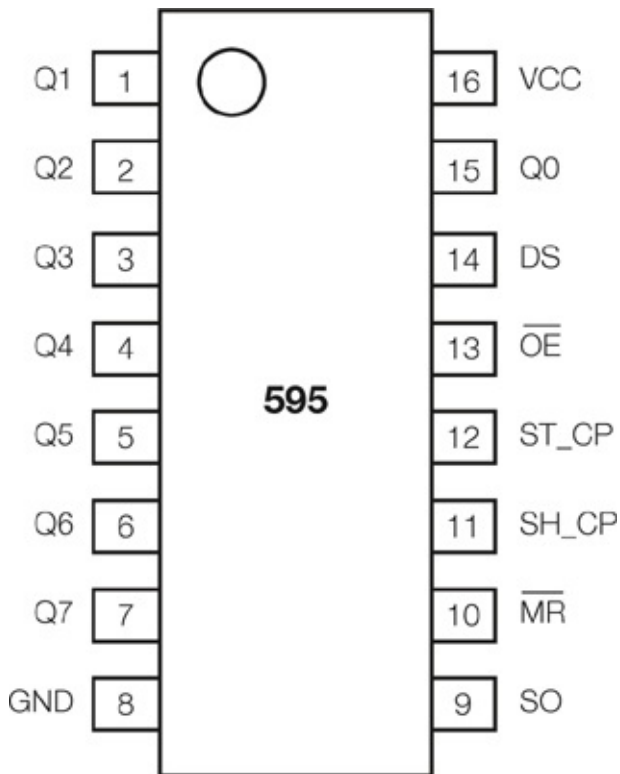


TABLE 24-2:
Shift register pins

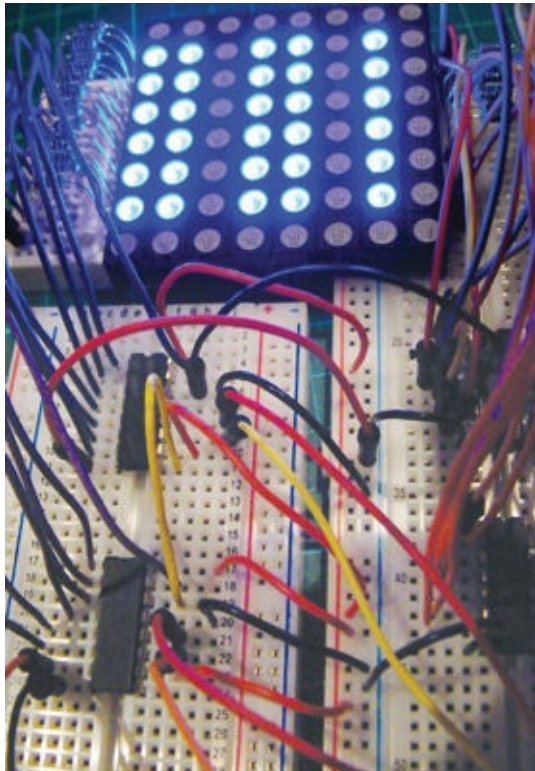
SHIFT REGISTER	CONNECTIONS	PIN FUNCTION
Pins 1–7, 15	Q0–Q7	Output pins
Pin 8	GND	Ground, VSS
Pin 9	SO	Serial out
Pin 10	MR	Master Reclear, active low
Pin 11	SH_CP	Shift register clock pin (CLOCK pin)
Pin 12	ST_CP	Storage register clock pin (LATCH pin)
Pin 13	OE	Output Enable, active low
Pin 14	DS	Serial data input (DATA pin)
Pin 16	VCC	Positive power

THE BUILD

1. Insert the 8x8 RGB LED matrix across two full-size breadboards.
2. Insert a 330-ohm resistor for each red LED pin and a 220-ohm resistor for each green or blue LED pin.
3. Insert the first shift register into one of the breadboards near the common-anode pins on the LED matrix. Place the register so that it straddles the center break, as shown in [Figure 24-4](#). Connect the common-anode pins of the LED matrix to shift register 1 as follows. These pins do not need resistors.

COMMON-ANODE PINS		SHIFT REGISTER 1 PINS	
LED MATRIX	SHIFT REGISTER	SHIFT REGISTER	ARDUINO
32	15: Q0	8: GND	GND
31	1: Q1	9: SO	Shift 3 DS
30	2: Q2	10: MR	+5V
29	3: Q3	11: SH-CP	13
20	4: Q4	12: ST-CP	10
19	5: Q5	13: OE	GND
18	6: Q6	14: DS	Shift 2 SO
17	7: Q7	16: VCC	+5V

3. **FIGURE 24-4:**
The shift registers should straddle the break of the breadboard.



4. Now insert the remaining three shift registers into the breadboard. Shift register 2 controls the green LEDs, shift register 3 controls the blue LEDs, and shift register 4 controls the red LEDs. Connect the wires for each shift register as shown in the following tables. All color LED pins will need resistors.

GREEN LED PINS		SHIFT REGISTER 2 PINS	
LED MATRIX	SHIFT REGISTER	SHIFT REGISTER	ARDUINO
28	15: Q0	8: GND	GND
27	1: Q1	9: SO	Shift 1 DS
26	2: Q2	10: MR	+5V
25	3: Q3	11: SH-CP	13
24	4: Q4	12: ST-CP	10
23	5: Q5	13: OE	GND
22	6: Q6	14: DS	11
21	7: Q7	16: VCC	+5V



BLUE LED PINS		SHIFT REGISTER 3 PINS	
LED MATRIX	SHIFT REGISTER	SHIFT REGISTER	ARDUINO
1	15: Q0	8: GND	GND
2	1: Q1	9: SO	Shift 4 DS
3	2: Q2	10: MR	+5V
4	3: Q3	11: SH-CP	13
5	4: Q4	12: ST-CP	10
6	5: Q5	13: OE	GND
7	6: Q6	14: DS	Shift 1 SO
8	7: Q7	16: VCC	+5V

RED LED PINS		SHIFT REGISTER 4 PINS	
LED MATRIX	SHIFT REGISTER	SHIFT REGISTER	ARDUINO
9	15: Q0	8: GND	GND
10	1: Q1	9: SO	Shift 3 DS
11	2: Q2	10: MR	+5V
12	3: Q3	11: SH-CP	13
13	4: Q4	12: ST-CP	10
14	5: Q5	13: OE	GND
15	6: Q6	14: DS	Shift 2 SO
16	7: Q7	16: VCC	+5V

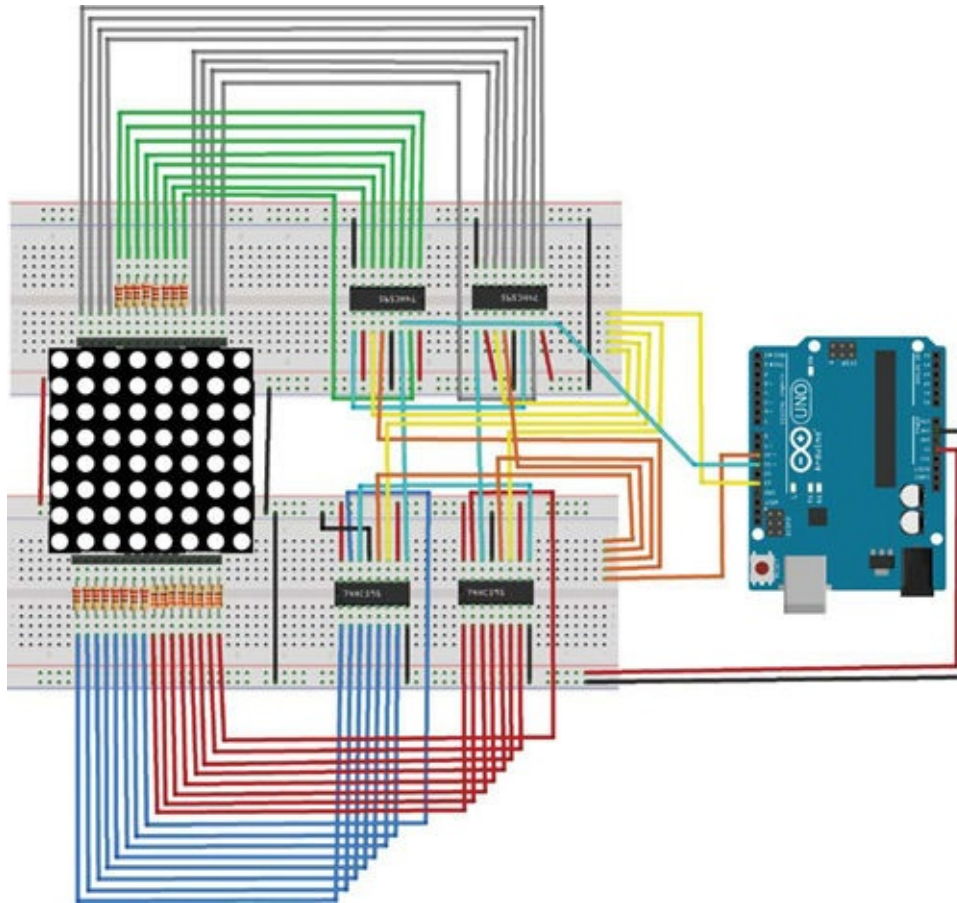
5. The Arduino controls the LEDs through three PWM pins, one each for clock, data, and latch. Each pin is connected to the Arduino as follows.

SHIFT REGISTER	ARDUINO	FUNCTION
Pin 9 (shift reg 2)	Pin 11	Data

Pin 12 (all shift reg)	Pin 10	Latch
Pin 11 (all shift reg)	Pin 13	Clock

6. Check that your setup matches the circuit diagram in [Figure 24-5](#), and then upload the code in “[The Sketch](#)” below.

6. **FIGURE 24-5:**
The circuit diagram for the rainbow maker



THE SKETCH

The sketch first defines the three Arduino pins that control the shift registers. The latch pin is defined as Arduino pin 10, the clock pin as 13, and the data pin as 11. We define a number of variables between 0 and 255 to control the brightness of the LED colors. The sketch then turns on each LED fully in turn and combines the three colors to create the colors of the rainbow. For instance, with green on, blue off, and red on, the color yellow is displayed. The sketch then finishes by cycling through random colors.

```
/* Example 18.1 - experimenting with RGB LED matrix
   CC by-sa 3.0
   http://tronixstuff.wordpress.com/tutorials
*/

int latchpin = 10; // Connect to pin 12 on all shift registers
int clockpin = 13; // Connect to pin 11 on all shift registers
int datapin = 11; // Connect to pin 14 on shift register 2
int zz = 500; // Delay variable
int va[] = {
  1, 2, 4, 8, 16, 32, 64, 128, 255
};
int va2[] = {
  1, 3, 7, 15, 31, 63, 127, 255
};

void setup() {
  pinMode(latchpin, OUTPUT);
```

```

pinMode(clockpin, OUTPUT);
pinMode(datapin, OUTPUT);
digitalWrite(latchpin, LOW);
shiftOut(datapin, clockpin, MSBFIRST, 0);
shiftOut(datapin, clockpin, MSBFIRST, 0);
shiftOut(datapin, clockpin, MSBFIRST, 0);
shiftOut(datapin, clockpin, MSBFIRST, 0);
digitalWrite(latchpin, HIGH);
randomSeed(analogRead(0));
}

void allRed() { // Turn on all red LEDs
digitalWrite(latchpin, LOW);
shiftOut(datapin, clockpin, MSBFIRST, 255); // Turn cathodes to full
shiftOut(datapin, clockpin, MSBFIRST, 0); // Turn green to 0
shiftOut(datapin, clockpin, MSBFIRST, 0); // Turn blue to 0
shiftOut(datapin, clockpin, MSBFIRST, 255); // Turn red to full
digitalWrite(latchpin, HIGH);
}

void allBlue() { // Turn on all blue LEDs
digitalWrite(latchpin, LOW);
shiftOut(datapin, clockpin, MSBFIRST, 255); // Turn cathodes to full
shiftOut(datapin, clockpin, MSBFIRST, 0); // Turn green to 0
shiftOut(datapin, clockpin, MSBFIRST, 255); // Turn blue to full
shiftOut(datapin, clockpin, MSBFIRST, 0); // Turn red to 0
digitalWrite(latchpin, HIGH);
}

void allGreen() { // Turn on all green LEDs
digitalWrite(latchpin, LOW);
shiftOut(datapin, clockpin, MSBFIRST, 255); // Cathodes
shiftOut(datapin, clockpin, MSBFIRST, 255); // Green
shiftOut(datapin, clockpin, MSBFIRST, 0); // Blue
shiftOut(datapin, clockpin, MSBFIRST, 0); // Red
digitalWrite(latchpin, HIGH);
}

void allOn() { // Turn on all LEDs
digitalWrite(latchpin, LOW);
shiftOut(datapin, clockpin, MSBFIRST, 255); // Cathodes
shiftOut(datapin, clockpin, MSBFIRST, 255); // Green
shiftOut(datapin, clockpin, MSBFIRST, 255); // Blue
shiftOut(datapin, clockpin, MSBFIRST, 255); // Red
digitalWrite(latchpin, HIGH);
}

void allYellow() { // Turn on green and red LEDs (yellow)
digitalWrite(latchpin, LOW);
shiftOut(datapin, clockpin, MSBFIRST, 255); // Cathodes
shiftOut(datapin, clockpin, MSBFIRST, 255); // Green
shiftOut(datapin, clockpin, MSBFIRST, 0); // Blue
shiftOut(datapin, clockpin, MSBFIRST, 255); // Red
digitalWrite(latchpin, HIGH);
}

void allAqua() { // Turn on green and blue LEDs (aqua)
digitalWrite(latchpin, LOW);
shiftOut(datapin, clockpin, MSBFIRST, 255); // Cathodes
shiftOut(datapin, clockpin, MSBFIRST, 255); // Green
shiftOut(datapin, clockpin, MSBFIRST, 255); // Blue
shiftOut(datapin, clockpin, MSBFIRST, 0); // Red
digitalWrite(latchpin, HIGH);
}

```

```

void allPurple() { // Turn on blue and red LEDs (purple)
    digitalWrite(latchpin, LOW);
    shiftOut(datapin, clockpin, MSBFIRST, 255); // Cathodes
    shiftOut(datapin, clockpin, MSBFIRST, 0); // Green
    shiftOut(datapin, clockpin, MSBFIRST, 255); // Blue
    shiftOut(datapin, clockpin, MSBFIRST, 255); // Red
    digitalWrite(latchpin, HIGH);
}

void clearMatrix() { // Turn off all LEDs
    digitalWrite(latchpin, LOW);
    shiftOut(datapin, clockpin, MSBFIRST, 0); // Cathodes
    shiftOut(datapin, clockpin, MSBFIRST, 0); // Green
    shiftOut(datapin, clockpin, MSBFIRST, 0); // Blue
    shiftOut(datapin, clockpin, MSBFIRST, 0); // Red
    digitalWrite(latchpin, HIGH);
}

void lostinspace() { // Random flashes of the LEDs
    for (int z = 0; z < 100; z++) {
        digitalWrite(latchpin, LOW);
        shiftOut(datapin, clockpin, MSBFIRST, va[random(8)]); // Cathodes
        shiftOut(datapin, clockpin, MSBFIRST, va[random(8)]); // Green
        shiftOut(datapin, clockpin, MSBFIRST, va[random(8)]); // Blue
        shiftOut(datapin, clockpin, MSBFIRST, va[random(8)]); // Red
        digitalWrite(latchpin, HIGH);
        delay(100);
    }
}

void displayLEDs(int rr, int gg, int bb, int cc, int dd) {
    // Insert the base-10 values into the shiftOut functions
    // and hold the display for dd milliseconds
    digitalWrite(latchpin, LOW);
    shiftOut(datapin, clockpin, MSBFIRST, cc); // Cathodes
    shiftOut(datapin, clockpin, MSBFIRST, gg); // Green
    shiftOut(datapin, clockpin, MSBFIRST, bb); // Blue
    shiftOut(datapin, clockpin, MSBFIRST, rr); // Red
    digitalWrite(latchpin, HIGH);
    delay(dd);
}

void loop() { // Light up the whole display in solid colors
    allOn();
    delay(zz);

    delay(zz);
    allRed();
    delay(zz);

    delay(zz);
    allGreen();
    delay(zz);

    delay(zz);
    allBlue();
    delay(zz);

    delay(zz);
    allPurple();
    delay(zz);
}

```

```
delay(zz);
allYellow();
delay(zz);

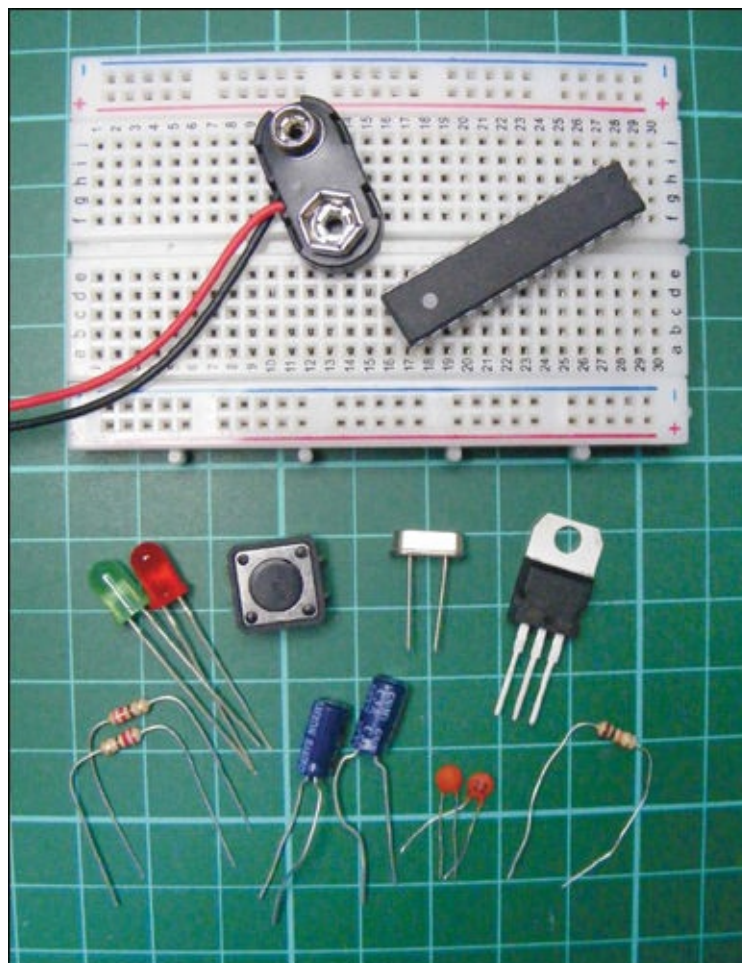
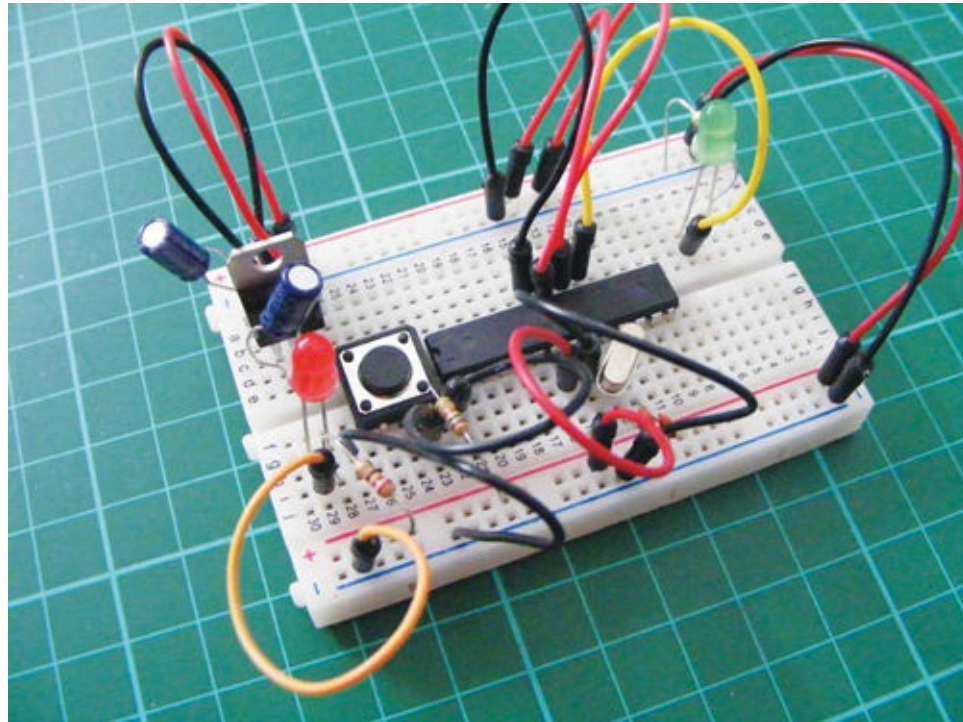
delay(zz);
allAqua();
delay(1000);
// Light some individual LEDs using random values
lostinspace(); // Scroll some horizontal and vertical lines
for (int z = 0; z < 5; z++) {
  for (int q = 1; q < 129; q *= 2) {
    displayLEDs(255, 0, 0, q, 200);
  }
}
clearMatrix();
delay(1000);

for (int z = 0; z < 5; z++) {
  for (int q = 1; q < 129; q *= 2) {
    displayLEDs(0, 255, 0, q, 200);
    displayLEDs(q, 0, 0, 255, 200);
  }
}
clearMatrix();
delay(1000);

for (int z = 0; z < 5; z++) {
  for (int q = 1; q < 9; q++) {
    displayLEDs(0, 0, 255, va2[q], 200);
  }
}
clearMatrix();
delay(1000);
}
```

PROJECT 25: BUILD YOUR OWN ARDUINO!

THIS PROJECT WILL TEACH YOU HOW TO BUILD YOUR OWN ARDUINO USING MINIMAL INDIVIDUAL COMPONENTS.



PARTS REQUIRED

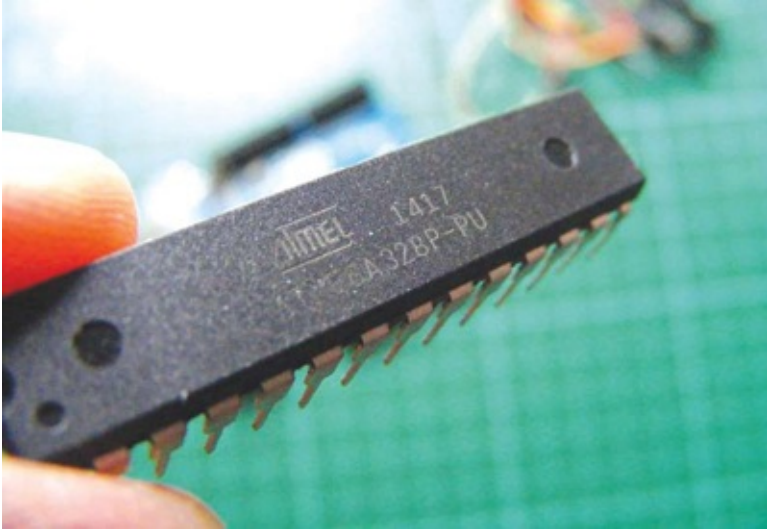
- ATMEL ATmega328p chip
- Breadboard
- Green LED
- Red LED
- 3 220-ohm resistors
- 16 MHz crystal oscillator (HC-495)
- L7805cv 5V regulator
- 2 100 μ F electrolytic capacitors
- PP3 9V battery clip
- Momentary tactile four-pin pushbutton
- 2 22 pF disc capacitors
- Jumper wires
- 9V battery

This is a fun and inexpensive little board with the same functionality as an Arduino, so it can be used as part of a permanent project in place of the pricier Arduino board.

HOW IT WORKS

Our project board works exactly the same as an Arduino board. At its heart is the ATMEL ATmega328p chip ([Figure 25-1](#)), to which we'll connect additional components. The ATmega chip is the brain of the Arduino and carries out the instructions from an uploaded sketch.

FIGURE 25-1:
The ATMEL ATmega328p chip



The L7805cv 5V regulator regulates the voltage and limits the current of the 9V battery to 5V, the level at which the ATmega chip operates, thereby protecting the chip and additional components. The 16 MHz crystal oscillator ([Figure 25-2](#)) allows the Arduino to calculate time, and the capacitors act as a filter to smooth voltage.

FIGURE 25-2:
The 16 MHz crystal oscillator



Table 25-1 details the pins of the ATmega328p chip and how they correspond to the Arduino pins. For example, pin 13 on the Arduino, which we used to test our Arduino in “[Testing Your Arduino: Blinking an LED](#)” on page 9, would be pin 19 on the actual chip. The top of the chip can be identified by the small semicircle indentation ([Figure 25-3](#)). Pin 1 is below this indentation, and the pins are numbered 1–28 counterclockwise from there.

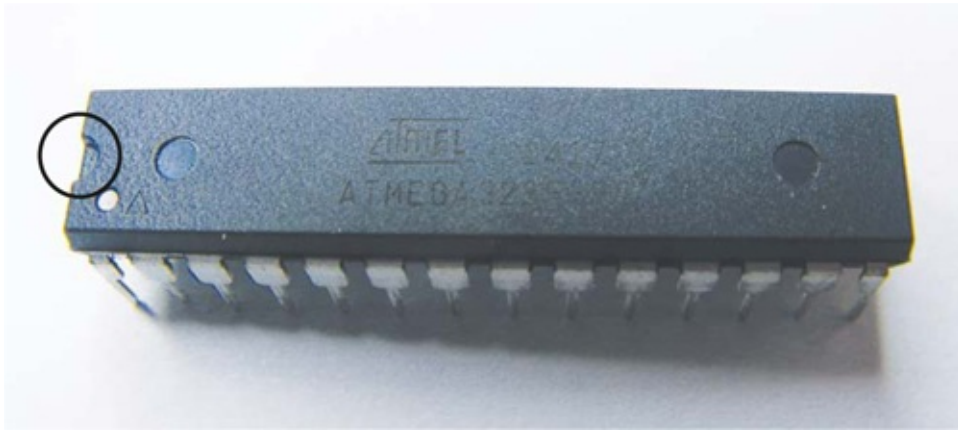
TABLE 25-1:
The ATmega chip’s pins and their corresponding Arduino pins

ATMEGA PIN	ARDUINO FUNCTION	ATMEGA PIN	ARDUINO FUNCTION
1	Reset	15	Pin 9
2	Pin 0	16	Pin 10
3	Pin 1	17	Pin 11
4	Pin 2	18	Pin 12
5	Pin 3	19	Pin 13
6	Pin 4	20	BCC
7	VCC	21	AREF
8	GND	22	GND
9	Crystal	23	A0
10	Crystal	24	A1
11	Pin 5	25	A2
12	Pin 6	26	A3

13	Pin 7	27	A4
14	Pin 8	28	A5

FIGURE 25-3:

The top of the chip is marked with a semicircle indentation.



PREPARING THE CHIP

Make sure to buy an ATmega chip with the Arduino bootloader installed, as it will also come preloaded with the blinking LED sketch, which you'll need for this project.

Our homemade Arduino does not have a USB connector for the chip to connect directly to your PC, so if you want to use this Arduino breadboard with a different sketch (or if your chip didn't come with the bootloader installed), you'll need to use an existing Arduino board as a host and upload the sketch to your ATmega chip as follows:

1. Carefully pry the Arduino ATmega chip from your existing Arduino board ([Figure 25-4](#)), and replace it with your ATmega chip.

1. **FIGURE 25-4:**
Removing the ATmega chip from the Arduino



2. Connect the Arduino to your PC using a USB cable.
3. Open the Arduino IDE on your PC.
4. Load the sketch onto the chip.
5. Once the sketch is uploaded, disconnect the Arduino from your PC, gently remove this chip from the board, and replace the original Arduino ATmega chip.

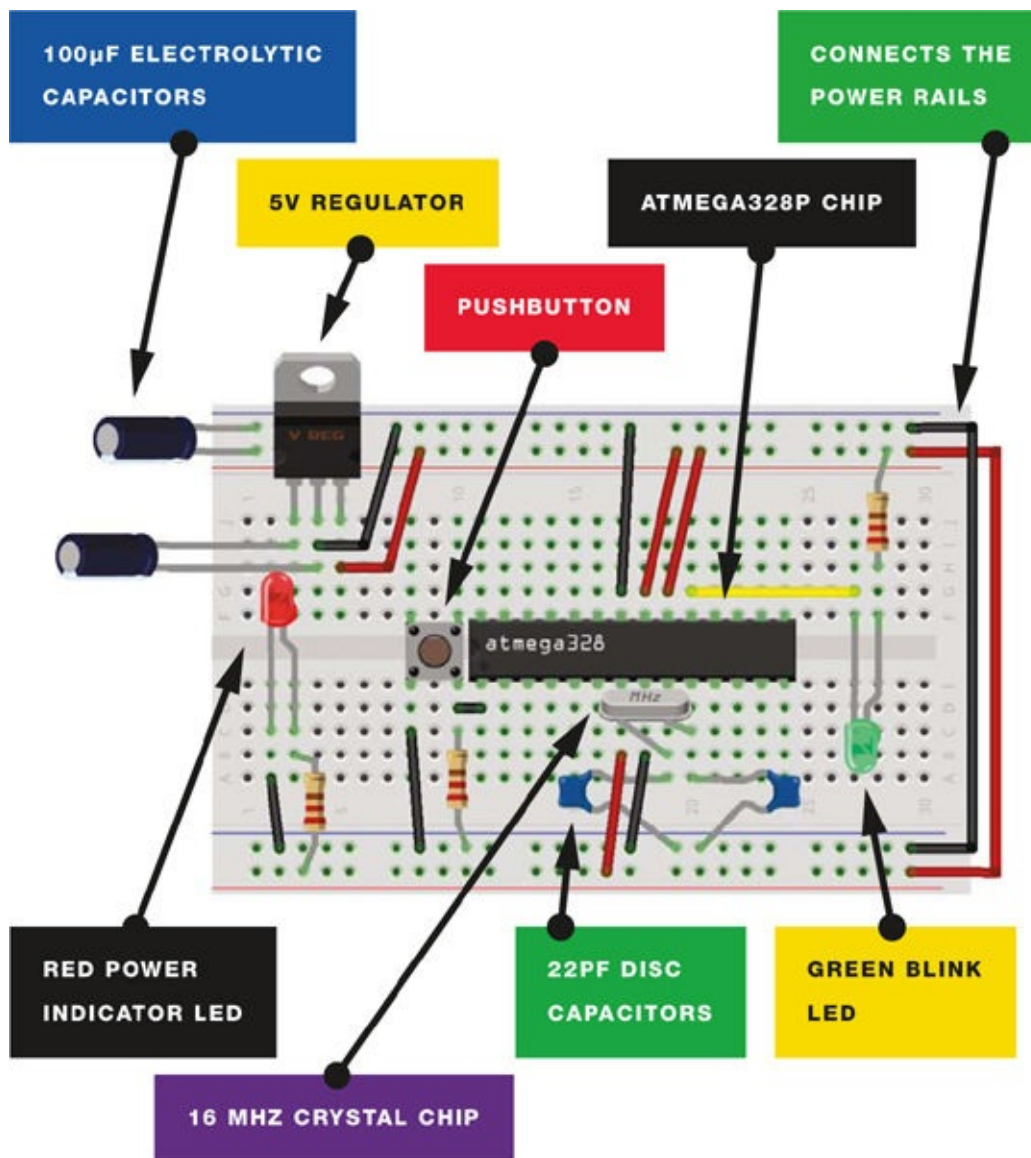
The new ATmega chip should be loaded with the desired sketch. Generally you'd want to build your own Arduino as part of a permanent project, so the ability to easily load new sketches is not usually required; you'd just load one sketch at the beginning of the project and use that sketch from then on.

You are now ready to prepare your own board.

BUILDING THE ARDUINO CIRCUIT

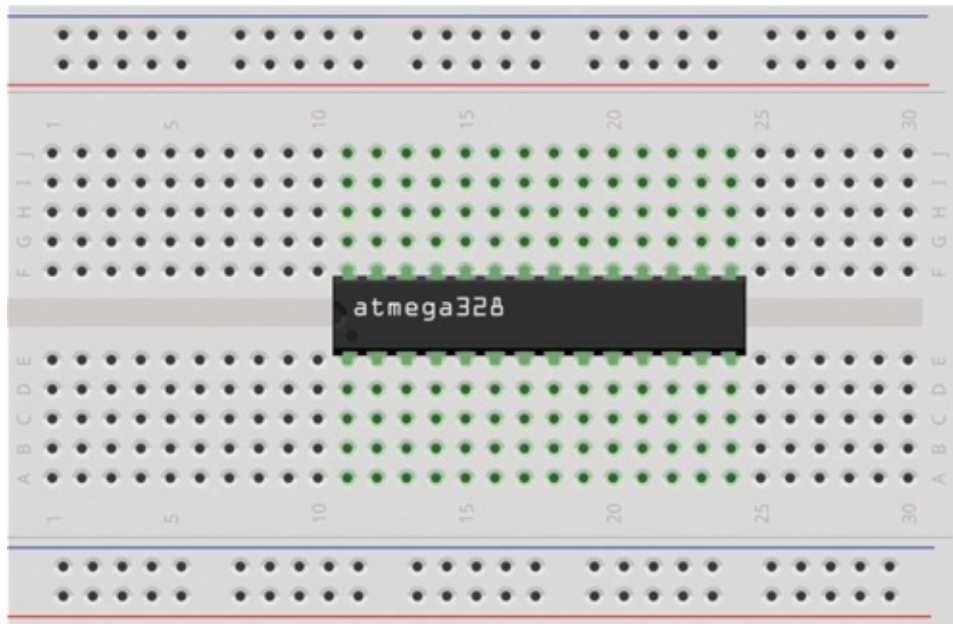
I normally show the circuit diagram at the end of the chapter, but in this instance it's helpful to look at it first to reference the layout and identify the components being used ([Figure 25-5](#)).

FIGURE 25-5:
The complete circuit diagram



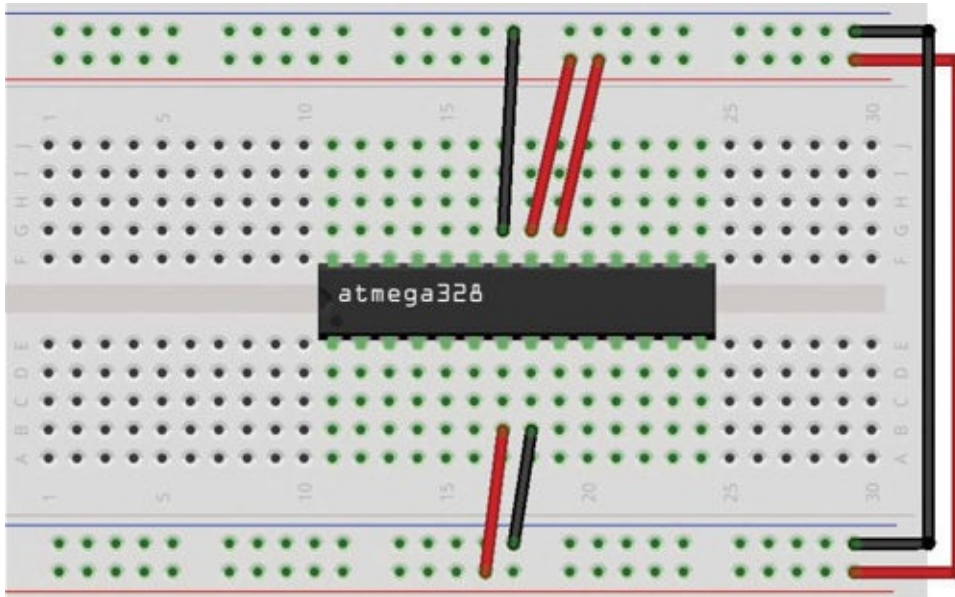
1. Insert the ATmega chip into the breadboard with its legs straddling either side of the center break. You need a little space at either end for components, so place it roughly as shown in [Figure 25-6](#). Remember, pin 1 of the ATmega328p is directly below the small semicircle indentation on the chip. From here, pins are numbered 1–28 counterclockwise. Use this to position your chip correctly. The semicircle should be on the left side of your circuit.

1. **FIGURE 25-6:**
Placing the ATmega chip so it straddles the center break



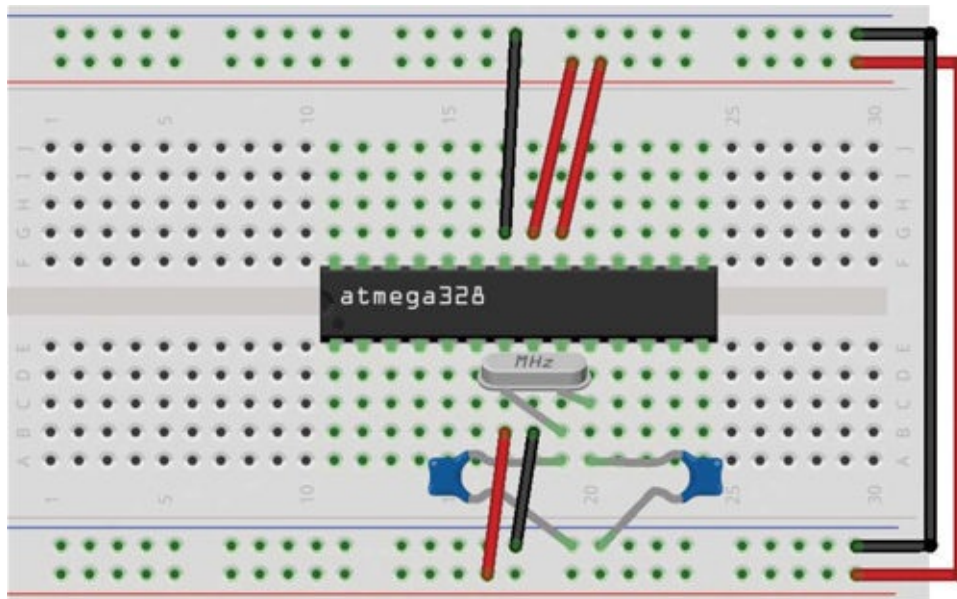
2. Connect pins 7, 20, and 21 of the ATmega to their closest positive power rail on the breadboard, and pins 8 and 23 to the negative power rails. Use jumper wires to connect the positive and GND power rails on either side of the board, as shown in [Figure 25-7](#).

2. **FIGURE 25-7:**
Connecting to the power rails



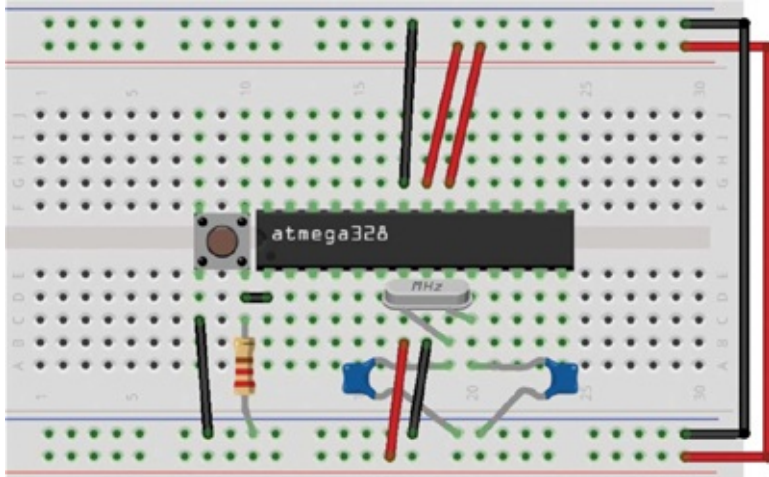
3. Connect one leg of the crystal oscillator to pin 9 on the ATmega chip, and connect the other leg to pin 10. Connect the legs of one of the 22 pF disc capacitors to pin 9 and GND, and the legs of the other disc capacitor to pin 10 and GND, as shown in [Figure 25-8](#).

3. **FIGURE 25-8:**
Inserting the crystal oscillator and 22pf disc capacitors



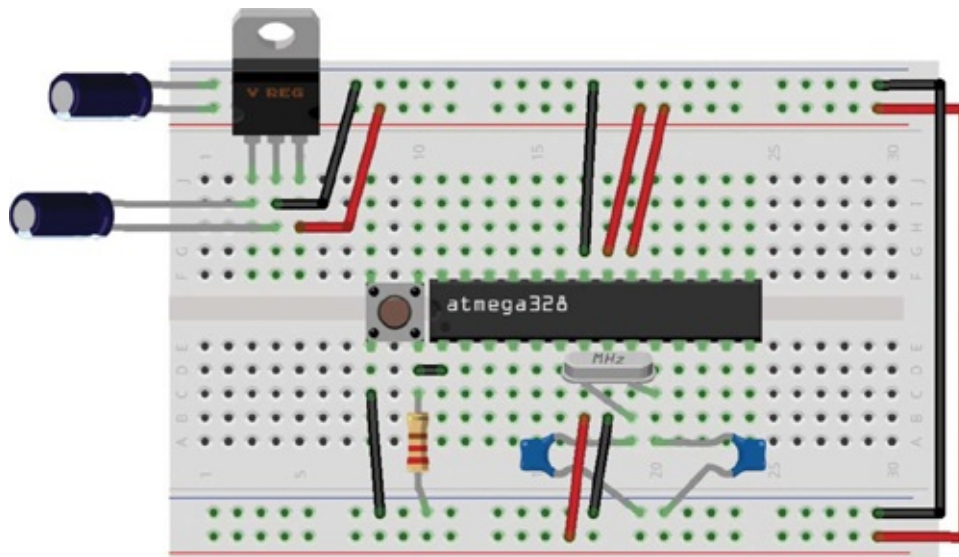
4. Insert the pushbutton into the breadboard to the left of the ATmega chip, with the legs straddling the center break in the breadboard. Using jumper wires, connect the lower-right pin of the pushbutton to pin 1 on the ATmega, and the lower-left pin to GND, as shown in [Figure 25-9](#). Connect a 220-ohm resistor to the lower-right pin, and connect the other side of this resistor to the GND rail. This pushbutton will act as our reset button.

4. **FIGURE 25-9:**
Inserting the reset button



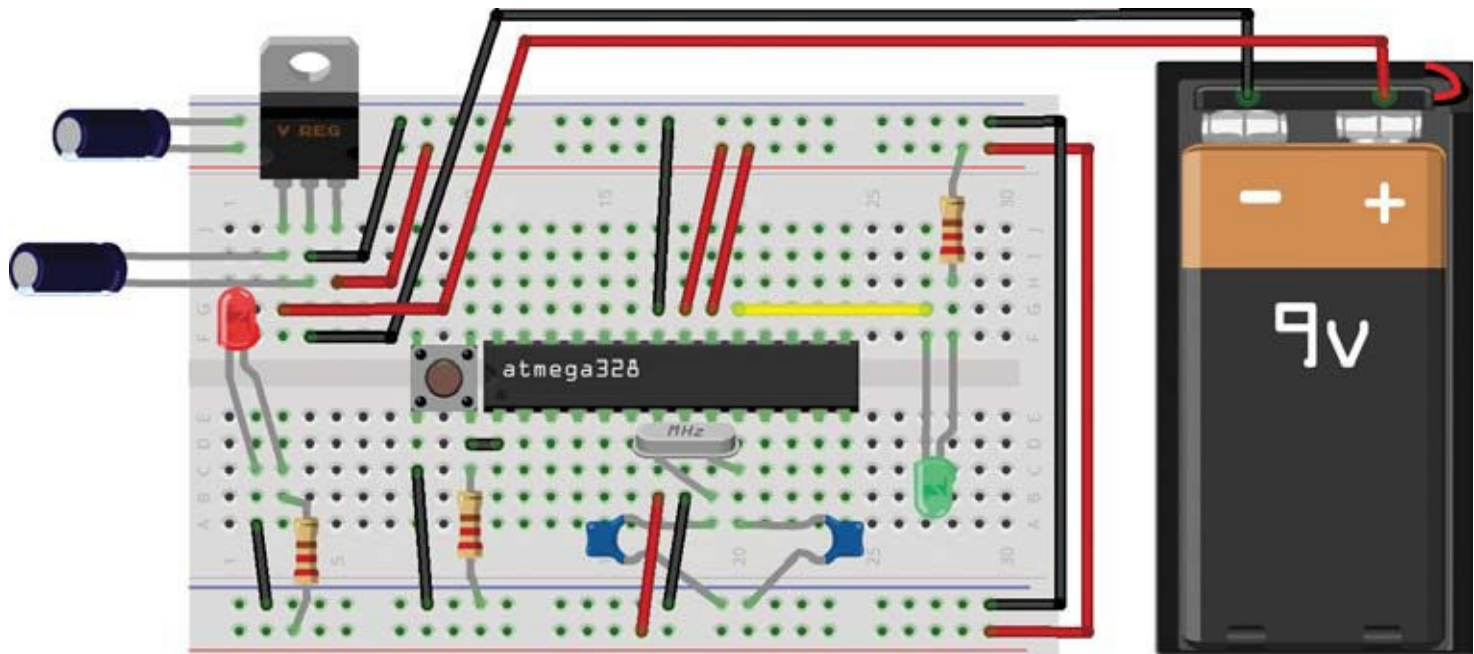
5. Insert the L7805cv 5V regulator into the top-left corner of the breadboard with the printed number of the component facing you, as shown in [Figure 25-10](#)—the pins are numbered 1–3 from left to right. Insert one 100 μF electrolytic capacitor into the top power rail of the breadboard, with one pin in the positive rail and the other pin in the negative rail. Connect the second 100 μF electrolytic capacitor to pins 1 and 2 of the 5V regulator. Then connect pin 2 of the regulator to the negative power rail and pin 3 to the positive power rail.

5. **FIGURE 25-10:**
Connecting the electrolytic capacitors and the L7805cv 5V regulator



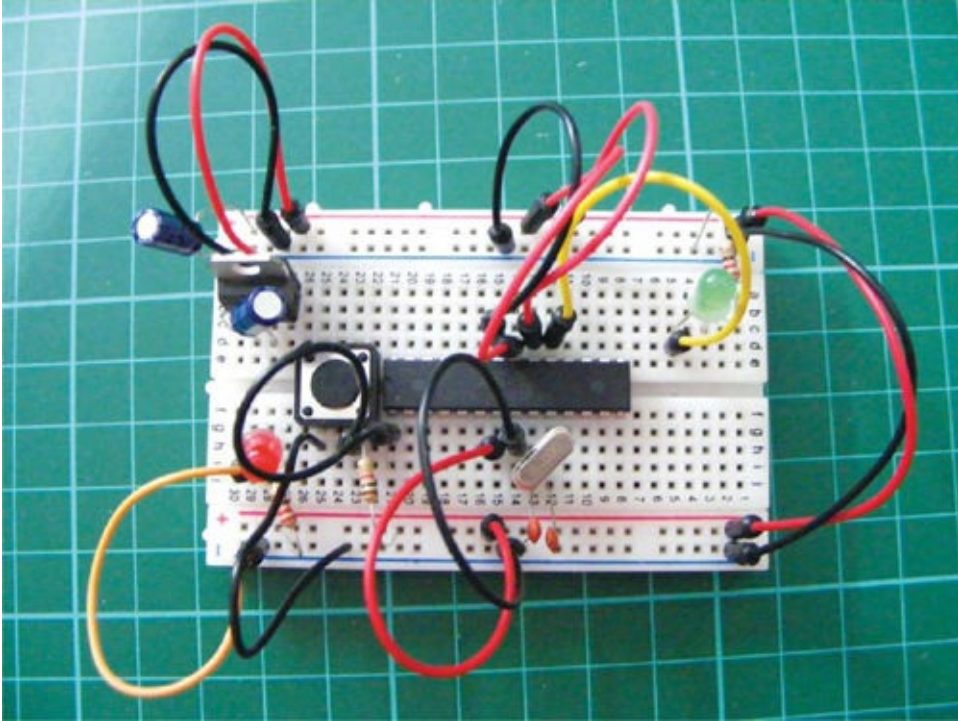
6. Insert the red LED into the breadboard, connecting the long, positive leg to the positive rail via a 220-ohm resistor, and the short, negative leg to GND. Then insert the green LED, connecting the short leg to pin 21 on the ATmega, and the long leg to the positive power rail via a 220-ohm resistor, as shown in [Figure 25-11](#). Add positive power from the battery to pin 1 on the 5V regulator and GND to pin 2 on the regulator.

6. **FIGURE 25-11:**
Inserting the LEDs and connecting the battery



Your board is now complete and should look like [Figure 25-12](#). The red LED lights when power is added to the breadboard rails to indicate that the Arduino is on and working, and the green LED lights in response to the “Blinking an LED” sketch loaded on the ATmega chip.

FIGURE 25-12:
The completed circuit



Using the reference in [Table 25-1](#), you can use this board just like an Arduino Uno by connecting components to the ATmega chip pins instead of the Arduino pins. If you want to make any of the projects from this book permanent, consider building your own Arduino to power it! Remember to load the sketch to the ATmega chip through the real Arduino board first.

APPENDIX A: COMPONENTS

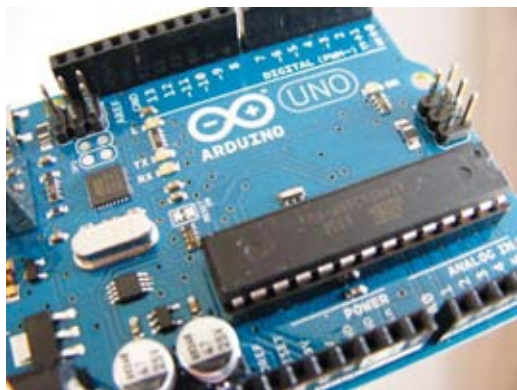
THIS APPENDIX GIVES YOU SOME MORE INFORMATION ON THE COMPONENTS USED IN THE PROJECTS IN THIS BOOK. EACH COMPONENT IS ACCOMPANIED BY A PHOTO AND A FEW DETAILS FOR QUICK REFERENCE AND IDENTIFICATION, AND I'VE INCLUDED A HANDY LIST OF RETAILERS TO BUY THE PARTS FROM. YOU'LL ALSO GET A QUICK LESSON ON READING RESISTOR VALUES.

COMPONENTS GUIDE

Here's a guide to the components you'll use, with a few details that you might find useful. The components are listed in the order in which they appear in the book. Many of the items can be found with a simple search on sites like eBay or Amazon, but a list of specialist retailers is also provided on [page 240](#).

Arduino Uno R3

The Arduino Uno R3 is the main component for the book and the brain for all our projects.

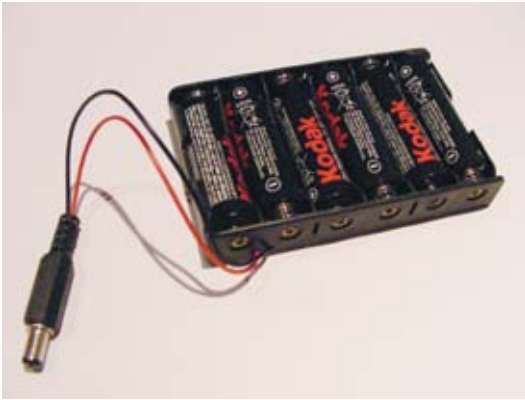


- Quantity: 1
- Connections: 14
- Projects: All except [Project 25](#)

9V Battery Pack

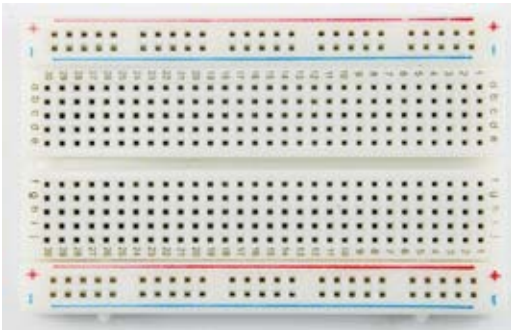
The 9V battery pack plugs into the Arduino to power your projects. You connect the batteries and plug the jack into the port in the Arduino, as discussed in “[Power](#)” on [page 3](#). Note that the Arduino can also be powered through the USB cable.

- Quantity: 1
- Connections: 1
- Projects: Optional for all



Breadboard

The breadboard is a prototyping board used to connect components and create your projects. See “[Breadboards](#)” on [page 4](#) for more information.



- Quantity: 2 full-size boards, 1 half-size board, 1 mini board
- Connections: 940 on a full board, 420 on a half board, 170 on a mini board
- Projects: All except [Project 7](#)

LED

An LED emits light when a small current is passed through it. It looks like a small light bulb with two legs. The longer leg is the positive connection. LEDs generally require a resistor or they may burn out. LEDs are polarized, meaning current flows only in one direction.



- Quantity: 40 (10 each of red, blue, yellow, green)
- Connections: 2
- Projects: 1–6, 8, 9, 17, 18, 19, 21, 22, 23, 25

Resistor

Resistors restrict the amount of current that can flow through a circuit to prevent components from overloading. They look like cylinders with colored bands and a wire from either end. The value is indicated by a color code—see “[Decoding Resistor Values](#)” on [page 241](#) for more details. Check this carefully, as it can be easy to choose the wrong value. Resistors come in two-, four-,

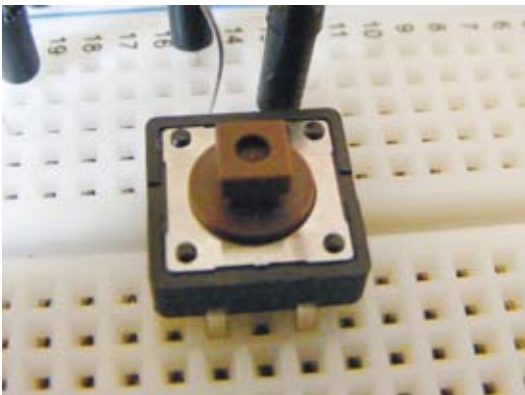
and five-band varieties, so be aware that, for example, a four-band 220-ohm resistor can look slightly different from a five-band resistor of the same value.



- Quantity: 30 220-ohm, 10 330-ohm, 1 10k-ohm, 1 1m-ohm resistors
- Connections: 2
- Projects: 1–4, 6, 8, 9, 16, 17, 18, 19, 22, 23, 24, 25

Pushbutton

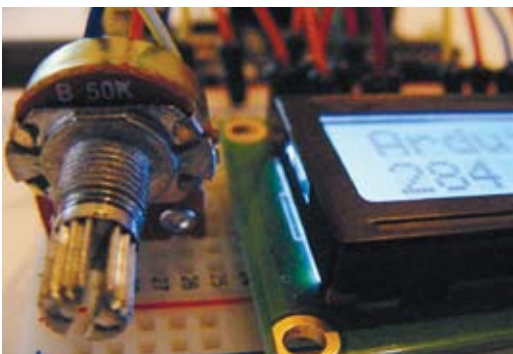
The pushbutton is a simple switch that makes a connection when pushed. This switch connects a circuit when pushed in, but will spring back when released and break the connection. It is also known as a momentary switch. Pushbuttons vary in size, but most will have four pins.



- Quantity: 4
- Connections: 4
- Projects: 2, 8, 15, 16, 17, 25

Potentiometer

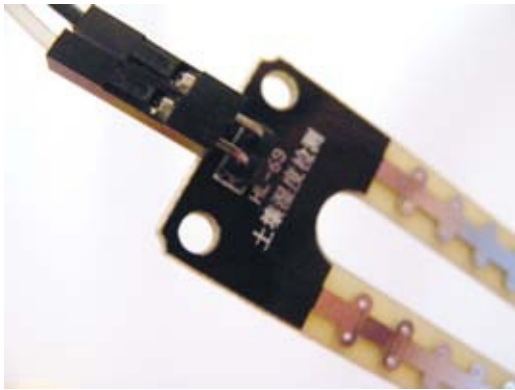
A potentiometer is a resistor whose value you can vary to manipulate the voltage flowing through it. It has a knob that you can turn and three pins at the bottom. The center pin is the control pin, with power to either side (it doesn't matter which way they are connected). It's commonly used to control an output such as the volume on a radio.



- Quantity: 1 50k-ohm potentiometer
- Connections: 3
- Projects: 2, 3, 4, 12, 13, 14, 15, 17

HL-69 Soil Sensor

A soil sensor measures the moisture content of soil. It has two prongs and two pins at the top. The sensor used in the book is the HL-69 soil hygrometer. It comes with a driver module that you connect to your Arduino, rather than connecting straight to the sensor.



- Quantity: 1
- Connections: 2
- Project: 5

Piezo Buzzer

The piezo buzzer is a very basic speaker. A pulse of current causes it to click extremely quickly, and a stream of pulses will emit a tone. It often looks like a small black box with two wires. Taken out of the case, it looks like a small gold disc. It's very cheap and used in inexpensive toys for noise generation (in sirens, for instance). It can also be used as a noise sensor, as shown in [Project 9](#).

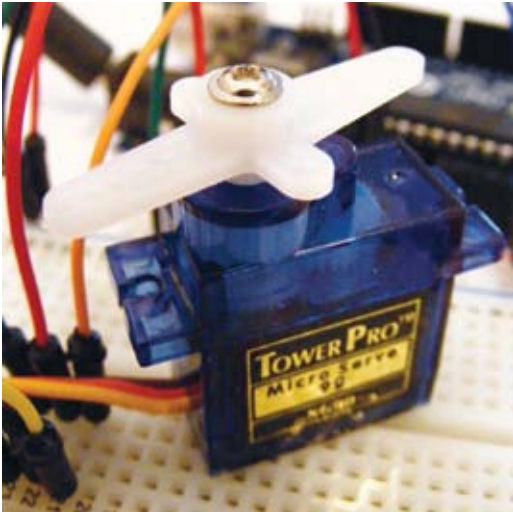


- Quantity: 1
- Connections: 2
- Projects: 5, 7, 8, 9, 15, 17, 18, 19, 21, 23

Servomotor

A servomotor is a motor with an arm that you can position to specific angles by sending the servo a coded signal. It is a small box with three wires and an output shaft, which can have an attachment (known as a horn). The red wire is POWER or +5V, the black/brown wire is GROUND or GND, and the orange/white wire is SIGNAL, which connects to your Arduino analog pin. The Tower Pro 9g servos used in this book will turn 180 degrees, but others are continuous and can turn the full 360 degrees.

- Quantity: 2
- Connections: 3
- Projects: 9, 10, 11, 18, 20, 22, 23



Joystick

A joystick records an analog input that can then be read to give a digital output. It's basically two potentiometers supplying a signal for two axes: left/right and up/down. It has lots of applications, like gaming or controlling a servomotor.

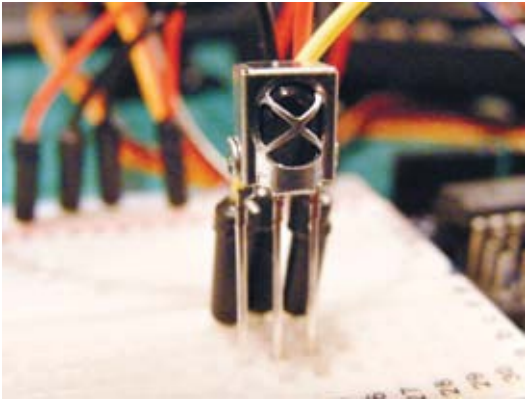


- Quantity: 1
- Connections: 5
- Project: 10

Infrared LED Receiver

An infrared (IR) LED receiver picks up infrared signals from, for example, a remote control. It is an LED in a small casing with three legs: OUT, GND, and +5V (positive power). It's polarized so it needs to be connected in the right way. Check the data sheet for your receiver, just in case the connections are different.

- Quantity: 1
- Connections: 3
- Project: 11



LCD Screen

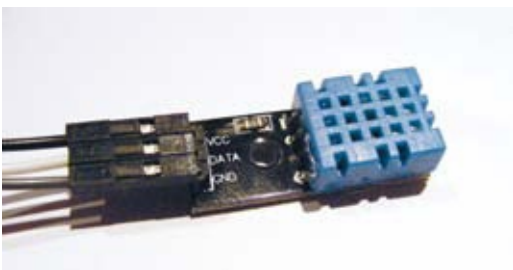
An LCD screen is a display screen for outputting characters. Screens come in various dimensions. The one shown here is an HD44780 (16 characters × 2 lines) and has 16 connections. An LCD screen consists of two sheets of polarizing material with a liquid crystal solution between them; current passing through the crystal creates an image.



- Quantity: 1
- Connections: 16
- Projects: 12, 13, 14, 15

DHT11 Humidity Sensor

The DHT11 sensor measures humidity and temperature. It is a small blue or white plastic box with four pins, though it's sometimes mounted on a module board that has only three pins. This book uses the DHT11 sensor, and we use only three of the pins: +5V, DATA, and GND.

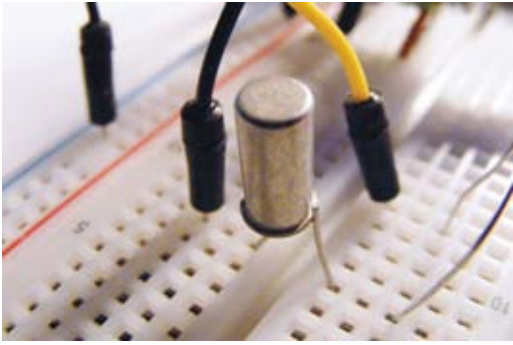


- Quantity: 1
- Connections: 4 (but we'll use only 3)
- Project: 13

Tilt Ball Switch

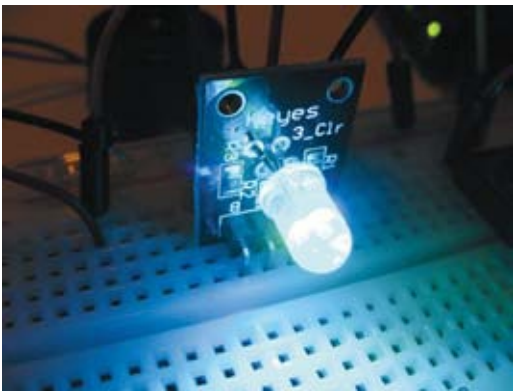
A tilt ball switch is a casing with a metal ball inside that makes a connection when in an upright position. Tilt the switch, and the connection is broken.

- Quantity: 1
- Connections: 2
- Project: 14



RGB LED

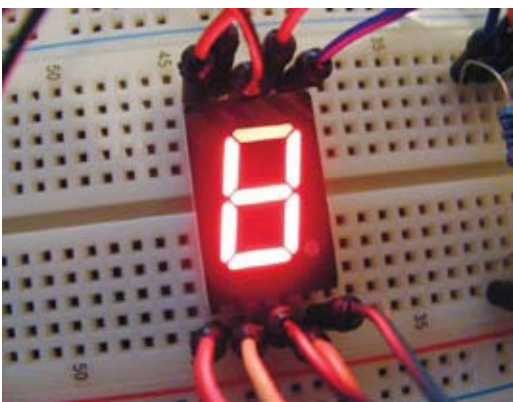
An RGB LED module is three colors in one—red, green, and blue. By combining the colors, you can make any color of the rainbow. It is a clear LED with four legs, sometimes mounted on a module with built-in resistors, as shown. You will need to use resistors to limit the current or the LED will burn out. The longest leg will be either the common cathode or anode.



- Quantity: 1
- Connections: 4
- Project: 15

Seven-Segment LED Display

A seven-segment LED display shows a digit or character using LED segments. They're often used to display numbers for counters, clocks, or timers. You can get single-digit to eight-digit displays, and four-digit displays are commonly used for digital clocks.

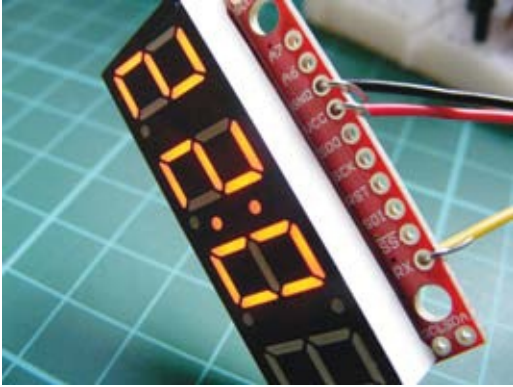


- Quantity: 1
- Connections: 10–12
- Projects: 16, 17

Four-Digit, Seven-Segment Serial Display

This is a four-digit version of the seven-segment LED, with an additional built-in circuit so it can be controlled with very few connections. This serial module is the SparkFun version and comes in

different colors. There are 10 connections, but it can be used with only 3 (VCC, GND, and RX) on the Arduino.



- Quantity: 1
- Connections: 10 (but we'll use only 3)
- Project: 17

Ultrasonic Sensor

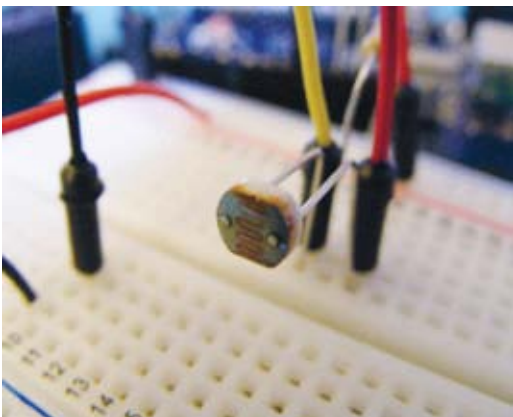
An ultrasonic sensor sends out a signal (often referred to as a *ping*), which bounces off an object and is returned to the sensor. Distance is calculated from the time the signal takes to return. The sensor used in this book is the HC-SR04 ultrasonic sensor. It is a module board with two round sensors and four pins.



- Quantity: 1
- Connections: 4
- Projects: 18, 20

Photoresistor

A photoresistor, also referred to as a light-dependent resistor or diode, produces a variable resistance depending on the amount of light falling on it and is used to detect light levels. There are different styles, but it's usually a small, clear oval with wavy lines and two legs. You will need to calibrate it to determine light levels before using it in a program.



- Quantity: 1
- Connections: 2
- Project: 19

RC V959 Missile Launcher

Produced for radio-controlled helicopters, the WLToys RC V959 missile launcher is a mini Gatling gun that can fire six plastic rockets in quick succession. It has four wires, but we use only the yellow and white for continuous firing.



- Quantity: 1
- Connections: 4 (but we'll use only 2)
- Project: 20

PIR Sensor

The PIR (passive infrared) sensor detects movement within its range. The book uses the HC SR501, the most commonly available PIR sensor. The module pictured has a golf ball-type lens on the front and three connections: +5V, OUTPUT, and GND. The orange cubes are potentiometers that change the distance range and output timing.



- Quantity: 1
- Connections: 3
- Project: 21

Keypad

A 4×4 keypad is basically a series of switches. The example shown here has 16 pushbuttons connected in series; a 12-button version is also available. Of the eight connections, four control the rows and four control the columns. The Arduino will replicate the number of the pressed button.

- Quantity: 1
- Connections: 8
- Project: 22



RFID Reader

An RFID (radio frequency identification) module reads RFID cards and key fobs to allow or deny actions depending on the access level of the card. It is a small board with eight pins and a built-in antenna. The module used in the book is the Mifare RFID-RC522 module, which usually comes with a card and fob.



- Quantity: 1
- Connections: 8
- Project: 23

RGB Matrix

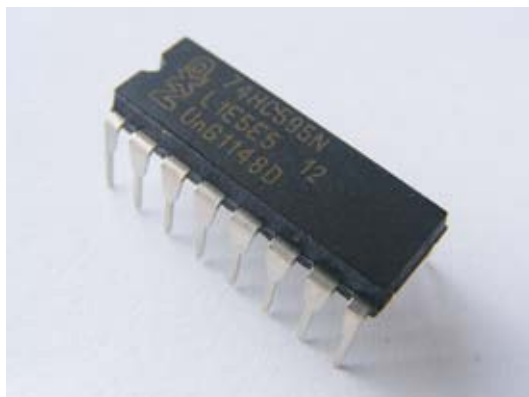
An 8x8 RGB matrix is a series of 64 LEDs that can change through red, green, and blue to create the colors of the rainbow. There are 32 pins on the matrix: 8 are for the common anode of each LED, 8 control the color red, 8 control green, and 8 control blue. Resistors are required for each pin controlling a color.



- Quantity: 1
- Connections: 32
- Project: 24

Shift Register

A shift register is a small integrated circuit and sequential logic counter that allows the Arduino to make more connections by “shifting” and storing data. It’s a small black chip with 16 legs. At one end, you’ll find a dot or semicircle—pin 1 is to the left of this marker. The electronic die in [Project 16](#) uses a 74HC595 shift register.



- Quantity: 1
- Connections: 16
- Projects: 16, 24

ATmega328p Chip

The ATMEL ATmega328p chip is the brain of the Arduino; it carries out the instructions from an uploaded sketch. It’s a small black chip with 32 legs. At one end you’ll find a dot or semicircle—pin 1 is to the left of this marker.



- Quantity: 1
- Connections: 32
- Project: 25

16 MHz Crystal Oscillator

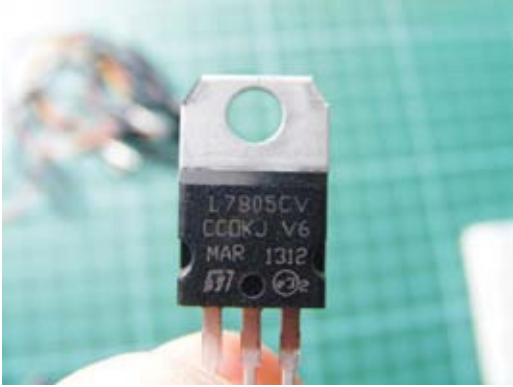
The 16 MHz crystal oscillator allows the Arduino to calculate time. It is a small metal casing with two legs and requires a capacitor on each leg to help smooth voltage to the crystal. The frequency of the crystal is printed on the front.



- Quantity: 1
- Connections: 2
- Project: 25

5V Regulator

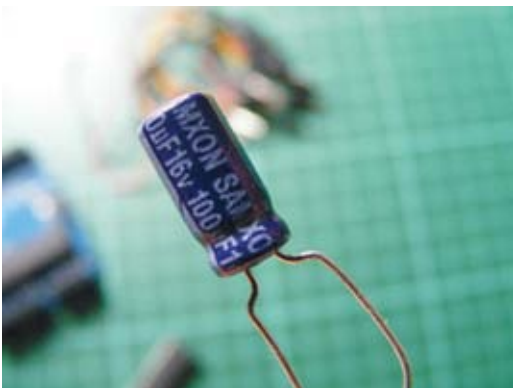
The L7805cv 5V regulator takes a voltage between 7 and 11 volts and steps it down to a constant 5 volts.



- Quantity: 1
- Connections: 3
- Project: 25

Capacitor

Capacitors can store a small amount of electricity for later use and can be used to smooth voltage output and flow. They look like small cylinders with two legs, and the value is usually printed on the side. Capacitors have polarity and need to be inserted correctly. The long leg is positive, and the short leg is negative; this is generally indicated on the cylinder. There are various types available; the one shown here is an aluminum 100 μ F electrolytic capacitor.



- Quantity: 2
- Connections: 2
- Project: 25

Disc Capacitor

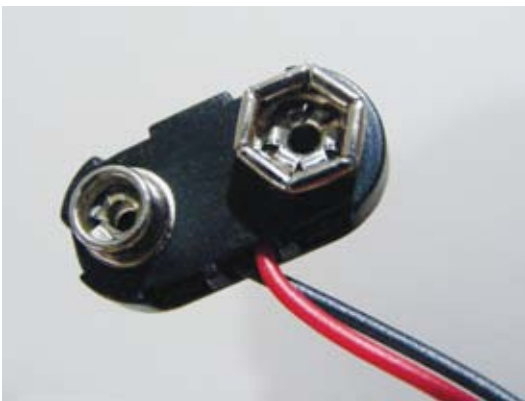
The 22pf disc capacitor is another type of capacitor that can store a small amount of electricity for later use. It looks like a small disc with two legs, and the value is usually printed on the front. There are various types available; the one shown here is a ceramic version.

- Quantity: 2
- Connections: 2
- Project: 25



Battery Clip

The PP3 9V battery clip is a simple connector for a 9V battery. It's a small black clip that has two wires: black for ground and red for positive.



- Quantity: 1
- Connections: 2
- Project: 25

RETAILER LIST

Most electronic components can be found on generic sites like eBay or Amazon, but if you have trouble finding anything, the retailers listed here can likely help you out.

US Retailers

Adafruit <https://www.adafruit.com/>

DigiKey <http://www.digikey.com/>

Jameco Electronics <http://www.jameco.com/>

Little Bird Electronics <http://www.littlebirdelectronics.com/>

MCM <http://www.mcmelectronics.com/>

Newark element14 <http://www.newark.com/>

RadioShack <http://www.radioshack.com/>

RS Components <http://www.rs-components.com/>

Seeed Studio <http://www.seeedstudio.com/depot/>

SparkFun <https://www.sparkfun.com/>

European Retailers

Electronic Sweet Pea's <http://www.sweetpeas.se/>

Element 14 <http://www.element14.com/>

Farnell <http://www.farnell.com/>

Jameco Electronics <http://www.jameco.com/>

UK Retailers

4tronix <http://www.4tronix.co.uk/store/>

Cool Components <http://www.coolcomponents.co.uk/>

CPC <http://cpc.farnell.com>

Hobby Components <https://www.hobbycomponents.com/>

Mallinson Electrical <http://www.mallinson-electrical.com/shop/>

Maplin <http://www.maplin.co.uk/>

Oomlout <http://oomlout.co.uk/>

The Pi Hut <http://thepihut.com/>

Proto-pic <http://proto-pic.co.uk/>

Rapid Electronics <http://www.rapidonline.com/>

RS <http://uk.rs-online.com/web/>

Spiratronics <http://spiratronics.com/>

DECODING RESISTOR VALUES

In most of projects in this book we've used *resistors*, electrical components that limit the amount of current allowed through a circuit (measured in ohms). They are used to protect components, like LEDs, from overloading and burning out. The value of a resistor is identified by colored bands on the body. Resistors can have four, five, or six colored bands.

It's important to be able to determine the value of a resistor so that you know you're using the correct one in your project. Let's try to determine the value of the four-band resistor shown in [Figure A-1](#).

FIGURE A-1:
A four-band resistor



Viewing the resistor with the silver or gold band on the right, note the order of the colors from left to right. If the resistor has no silver or gold band, make sure the side with the three colored bands is on the left.

Use [Table A-1](#) to determine the value of the resistor.

TABLE A-1:
Calculating resistor values

COLOR	FIRST BAND	SECOND BAND	THIRD BAND	MULTIPLIER	TOLERANCE
Black	0	0	0	1 Ω	
Brown	1	1	1	10 Ω	+/-1%
Red	2	2	2	100 Ω	+/-2%
Orange	3	3	3	1K Ω	
Yellow	4	4	4	10K Ω	
Green	5	5	5	100K Ω	+/-0.5%
Blue	6	6	6	1M Ω	+/-0.25%
Violet	7	7	7	10M Ω	+/-0.10%
Gray	8	8	8		+/-0.05%
White	9	9	9		
Gold				0.1 Ω	+/-5%

Silver				0.01Ω	+/-10%
--------	--	--	--	-------	--------

NOTE

The band that denotes the tolerance is most commonly silver or gold, though it can be any color that has a percentage listed in the Tolerance column. If you have a resistor with a tolerance band that isn't silver or gold, there should be a small gap between the value bands and the tolerance band so you can tell them apart.

The values that correspond to the first and second bands give you the numerical value, the third band tells you how many zeros to add to that number, and the fourth band tells you the *tolerance*—that is, how much the actual value can vary from the intended value. For the resistor in [Figure A-1](#):

- First band is brown (1) = 1.
- Second band is black (0) = 0.
- Third band is red (2) = 00 (2 is the number of zeros).
- Fourth band is gold, so the tolerance (accuracy) is +/- 5 percent.

So this resistor is 1,000 ohms or 1 kilohm, with a tolerance of 5 percent, meaning that the actual value can be up to 5 percent more or less than 1 kilohm. We can do the same calculation for a five- or six-band resistor.

If you're ever unsure of a resistor's value, a quick online search of the colored bands on the resistor's body will help; just make sure to list the colors in the correct order, reading them from left to right, with the tolerance band on the right.

APPENDIX B: ARDUINO PIN REFERENCE

WITHOUT GOING INTO TOO MUCH DETAIL, THIS APPENDIX GIVES YOU A REFERENCE TO THE PINS ON THE ARDUINO UNO, THEIR TECHNICAL NAMES, AND THEIR FUNCTIONS. THE PINS ARE EXPLAINED IN MORE DETAIL IN THE PROJECTS IN WHICH THEY'RE USED, SO THE INFORMATION HERE WILL PROBABLY MAKE MORE SENSE ONCE YOU'VE BUILT A FEW PROJECTS.

ARDUINO PIN	FUNCTION AND LABEL	ADDITIONAL FUNCTION
0	RX—Used to receive TTL serial data	
1	TX—Used to transmit TTL serial data	
2	External interrupt	
3	External interrupt	Pulse width modulation
4	XCK/TO—External Clock Input/Output (Timer/Counter 0)	
5	T1 (Timer/Counter 1)	Pulse width modulation
6	AIN0—Analog comparator positive input	Pulse width modulation
7	AIN1—Analog comparator negative input	
8	ICP1—Input capture	
9	OC1A—Timer register	Pulse width modulation
10	SS—Slave Select (serial data) used in SPI communication	Pulse width modulation
11	MOSI—Master Out Slave In (data in) used in SPI communication	Pulse width modulation
12	MISO—Master In Slave Out (data out) used in SPI communication	
13	SCK—Serial Clock (output from master) used in SPI communication	

AREF	Reference voltage for analog inputs	
A0	Analog input can give 1,024 different values.	
A1	Analog input can give 1,024 different values.	
A2	Analog input can give 1,024 different values.	
A3	Analog input can give 1,024 different values.	
A4	Analog input can give 1,024 different values.	SDA (serial data line) pin supports TWI (two-wire interface) using the Wire library for I2C components.
A5	Analog input can give 1,024 different values.	SCL (serial clock line) pin supports TWI using the Wire library for I2C components.
RESET	Can be used to reset the microcontroller	
3.3V	3.3 volt output used for low voltage components. This is the only 3.3V source. The digital and analog pins operate at 5V.	
5V	Standard +5V output	
GND	Ground/negative power	
V _{in}	9V power can be input here or accessed if using power jack.	

Serial: 0 (RX) and 1 (TX) These pins are used to receive (RX) and transmit (TX) transistor-transistor logic (TTL) serial data. We use the TX pin in the rocket launcher in [Project 17](#).

External interrupts: 2 and 3 These pins can be configured to trigger an interrupt on a low value, a *rising* or *falling edge* (a signal going from low to high or high to low, respectively), or a change in value. An *interrupt* is a signal that tells the Arduino to stop and carry out another function when the pins have detected an external event, such a pushbutton being pressed.

PWM: 3, 5, 6, 9, 10, and 11 These pins can be used with pulse width modulation through the `analogWrite()` function. There's more information on this in [Project 2](#).

SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK) These pins support SPI communication using the SPI library and are used a number of times in this book. We use SPI communication for the electronic die in [Project 16](#) so that the Arduino can send and receive data from the shift register

used to control the seven-segment LED.

LED: 13 There is a built-in LED connected to digital pin 13. When the pin is `HIGH`, the LED is on; when the pin is `LOW`, it's off. The built-in LED on pin 13 is used to show when the onboard ATmega328p bootloader is running, usually when the Arduino is starting up.

AREF This is the reference voltage for the analog inputs; it's used with `analogReference()`. We can input from 0 to 5V, so if your sensor requires a lower voltage than 5V, you can use this pin to increase the resolution for a more accurate reading.

Analog inputs: A0–A5 The Uno has six analog inputs, each of which provides 1,024 different values.

TWI: A4 and A5 These pins support *TWI (two-wire interface)* communication using the `Wire` library. This is used to control and communicate with an I2C device, such as a serial LCD screen, using only two wires.

RESET Set this to `LOW` to reset the microcontroller. This is typically used to add a reset button.

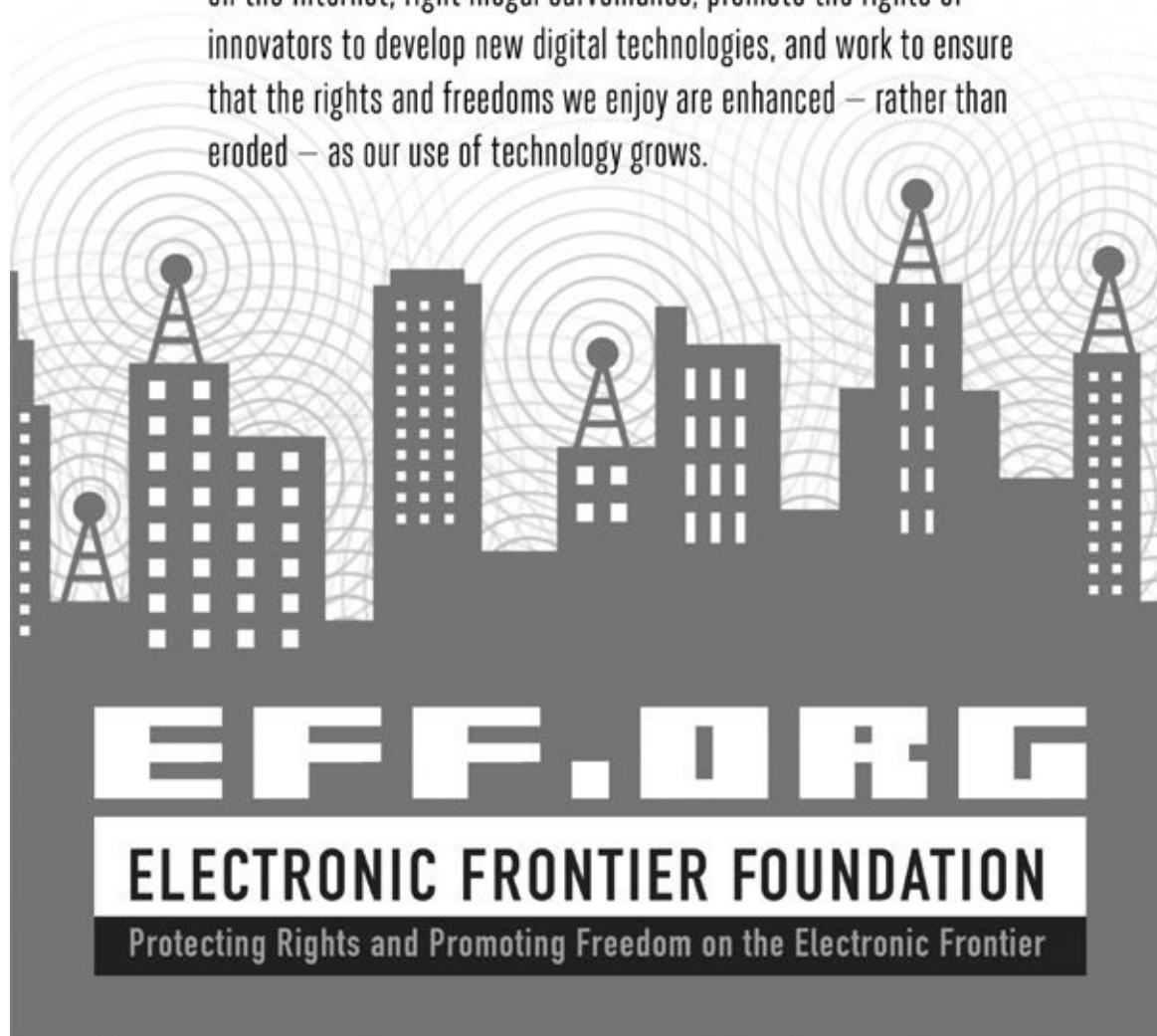
Don't worry if this information doesn't mean much to you right now. You might find it useful in your future Arduino endeavors, and you can reference it as you progress through the projects in the book.

Arduino Project Handbook is set in Helvetica Neue, Montserrat, True North, and TheSansMono Condensed. The book was printed and bound by Versa Printing in East Peoria, Illinois. The paper is 60# Evergreen Skyland.

The book uses a layflat binding, in which the pages are bound together with a cold-set, flexible glue and the first and last pages of the resulting book block are attached to the cover. The cover is not actually glued to the book's spine, and when open, the book lies flat and the spine doesn't crack.




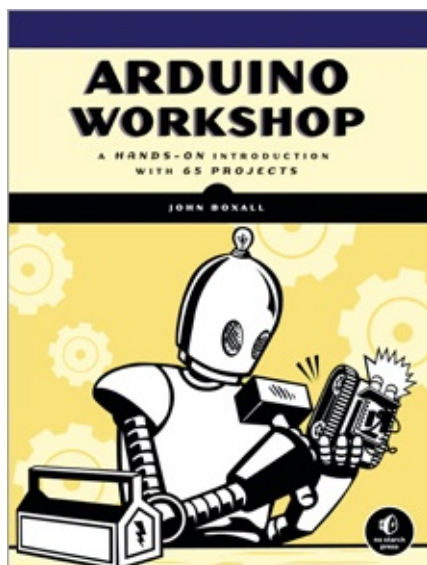
The Electronic Frontier Foundation (EFF) is the leading organization defending civil liberties in the digital world. We defend free speech on the Internet, fight illegal surveillance, promote the rights of innovators to develop new digital technologies, and work to ensure that the rights and freedoms we enjoy are enhanced – rather than eroded – as our use of technology grows.



UPDATES

Visit <http://www.nostarch.com/arduinohandbook/> for updates, errata, and other information.

More no-nonsense books from  **NO STARCH PRESS**



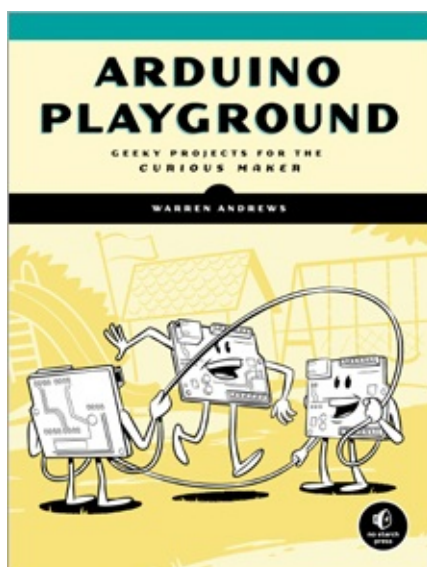
ARDUINO WORKSHOP

A Hands-On Introduction with 65 Projects

by JOHN BOXALL

MAY 2013, 392 PP., \$29.95

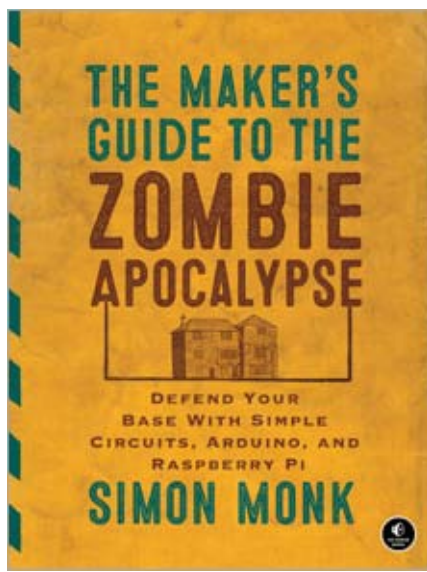
ISBN 978-1-59327-448-1



ARDUINO PLAYGROUND

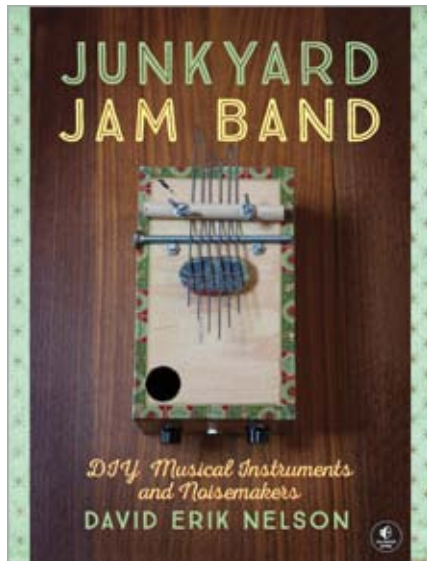
Geeky Projects for the Curious Maker

by WARREN ANDREWS
FALL 2016, 350 PP., \$29.95
ISBN 978-1-59327-744-4



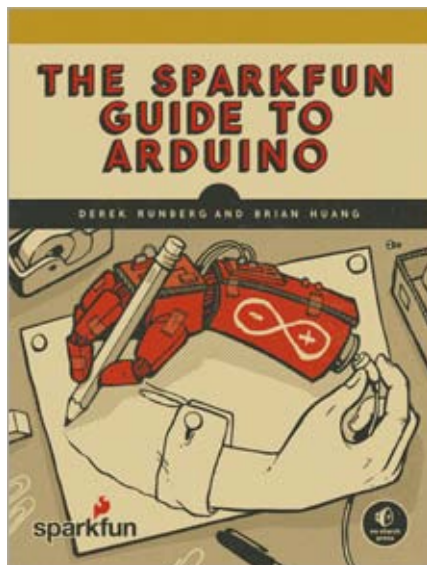
THE MAKER'S GUIDE TO THE ZOMBIE APOCALYPSE
Defend your Base with Simple Circuits, Arduino, and Raspberry Pi

by SIMON MONK
OCTOBER 2015, 296 PP., \$24.95
ISBN 978-1-59327-667-6



JUNKYARD JAM BAND
DIY Musical Instruments and Noisemakers

by DAVID ERIK NELSON
OCTOBER 2015, 408 PP., \$24.95
ISBN 978-1-59327-611-9



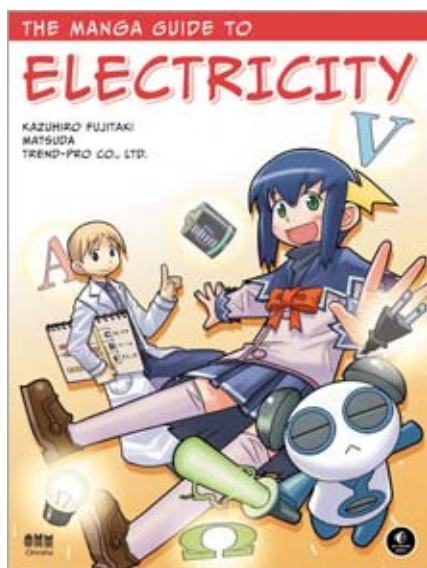
THE SPARKFUN GUIDE TO ARDUINO

by DEREK RUNBERG *and* BRIAN HUANG

WINTER 2017, 312 pp., \$29.95

ISBN 978-1-59327-652-2

full color



THE MANGA GUIDE TO ELECTRICITY

by KAZUHIRO FUJITAKI, MATSUDA, *and* TREND-PRO CO., LTD.

MARCH 2009, 224 pp., \$19.95

ISBN 978-1-59327-197-8

PHONE:

800.420.7240 OR
415.863.9900

EMAIL:

SALES@NOSTARCH.COM

WEB:

WWW.NOSTARCH.COM



REQUIRES: ARDUINO UNO

YOU GOT AN ARDUINO—NOW WHAT?

Arduino Project Handbook is a beginner-friendly collection of electronics projects using the low-cost Arduino board. With just a handful of components, an Arduino, and a computer, you'll learn to build and program everything from light shows to arcade games to an ultrasonic security system.

First you'll get set up with an introduction to the Arduino and valuable advice on tools and components. Then you can work through the book in order or just jump to projects that catch your eye. Each project includes simple instructions, colorful photos and circuit diagrams, and all necessary code.

Arduino Project Handbook is a fast and fun way to get started with microcontrollers that's perfect for beginners, hobbyists, parents, and educators.

25 STEP-BY-STEP PROJECTS

- Pushbutton-Controlled LED
- Light Dimmer
- Bar Graph
- Disco Strobe Light
- Plant Monitor
- Ghost Detector
- Arduino Melody
- Memory Game
- Secret Knock Lock
- Joystick-Controlled Laser
- Remote Control Servo
- LCD Screen Writer
- Weather Station
- Fortune Teller
- Reaction Timer Game
- Electronic Die
- Rocket Launcher
- Intruder Sensor

- Laser Trip Wire Alarm
- Sentry Gun
- Motion Sensor Alarm
- Keypad Entry System
- Wireless ID Card Entry System
- Rainbow Light Show
- Build Your Own Arduino



THE FINEST IN GEEK ENTERTAINMENT™

www.nostarch.com