Bonus Chapter

Hacking Other Hardware

In This Chapter

- Hacking hardware
- Taking over a remote control
- Controlling a camera
- Making cordless drills do your bidding
- Pretending to be a keyboard

You must be familiar with the idea that using the right tool for the right job makes all difference in how well that job goes. The world is full of products that are intended to do jobs well, and a great deal of time and money was spent designing and refining them. Unfortunately, not everything is made with Arduino in mind, so you can encounter quite a gap to bridge to make these devices into the "right tool" for whatever you have in mind.

Hacking hardware is all about repurposing devices and systems to do the jobs you want them to do. In this chapter, you learn about safely transforming these existing devices, such as remote controls, into something for useful for your own project, allow your Arduino to do more without risking its life or yours.

Disassembling Devices without Destroying Them

Before I start talking about gutting hardware (which I do in the next section with a PlayStation controller), you should review these important rules to make sure that this isn't the last time that the hardware in question ever works:

- **Check whethersomeone else has done this before.** The Internet houses a wealth of knowledge, so be sure to skim through it before embarking on your own into the unknown. Chances are, if you want to do something, someone else has tried it before (or at least tried something similar). Nothing is better than finding a website where all the guesswork

has been done for you. This saves you the trial and error of choosing which remote control or games controller to buy. If you're lucky, you may even find complete step-by-step instructions, photographs, and shopping lists for specific pieces of hardware.

- **Power down.** Make sure that hardware you plan to take apart isn't powered; that is, take out any batteries and unplug any leads. If you're poking a screwdriver around or pulling at wires, the last thing you want is a short circuit damaging the board (or you).
- **Take pictures beforehand.** Digital photos are free, and most of us have access to a digital camera of some sort. Use it. It's easy to forget where parts or connections were attached when they aren't any longer, and a simple photo at each stage of the disassembly can be a lifesaver for reassembly.
- **Know your joints.** Most plastic controllers and other hardware have tiny screw joints. You can remove these with a precision screwdriver and carefully take apart the controller. If all the screws are out and it still won't budge, make sure to check under the stickers. Screws may be hidden under them, often as a clever way for retailers or manufacturers to check whether the product has been tampered with.
- **Use a parts tray.** As you're removing parts, it's extremely easy for stray screws and components to go missing, so make sure that you have a parts tray. This could be a box or a box with dividers for extra organization, or even a magnetic screw tray, found in most good hardware stores. Or go really low tech and use a jar or even an envelope.
- **Don't force it.** And if you do, live with it. A lot of plastic pieces in hardware aren't made to be taken apart, and fittings that snap into place may take considerable effort to undo. Sometimes — with a great amount of patience — you can find these clips and unclip them with screwdrivers with just the right amount of force. But they may break, and that's just the way it is. Internal fixings or glue mean that brute force is the sometimes the only way to open hardware enclosures. Because you often want just the inner workings anyway, cracked and broken pieces aren't a problem, but be careful not to damage what's inside when you force something open.
- **Know your warranty.** Don't expect to be able to get your money back if you break a piece of hardware. These devices have been closed since they left the factory to ensure that standards are kept. What you're doing is major surgery on a device that's prepackaged for you, so it's very unlikely that a company refunds you after a failed "optimization."

Hacking a Remote Control

A remote control hack is an incredibly useful way of bypassing difficult or unknown hardware that you want to include in your project. By using a

component called an optocoupler, you can trigger buttons on a remote control in the same way that you can blink an LED with your Arduino. Understanding how to use this simple component allows you to do amazing things with your Arduino, quickly, easily, and safely.

Some hacking tricks have been around for years before Arduino, of course, particularly in the art world. If you've ever been to a fine art show, you have doubtless seen video installations with many monitors or projectors running off DVD players (or before DVDs, VCRs) that are all synchronized. This was in most cases done by eye, by pressing the Play button on the DVD players simultaneously. That's fine for looping videos, but what if you want to have a more accurate timer for the piece, or better yet, make the installation more responsive, playing only when someone enters the room? The Arduino, of course, can help.

One part of a DVD player *is* low voltage: the remote control. This is normally true of any remote control that's wireless and runs off AA or AAA batteries. This fact is extremely important because the remote control is made to control bigger devices without any electrical connection. By hacking the remote, you can directly control whatever it's talking to without needing to look "under the hood" of the potentially dangerous mains-powered device.

You commonly encounter two types of remote control that you commonly come across. The first is the infrared control, which is typically used for televisions and DVD players, for example. This type of remote control blinks an infrared LED that is invisible to the human eye but highly visible to the infrared receiver (similar to a light sensor) on the device. To prevent interference from other light and head sources, such as the sun or light bulbs, the device has a filter that allows only a very precise wavelength of infrared light to be received.

The other type of remote control is similar to that found in remote control cars or aircraft (that everyone enjoys for 10 minutes every Christmas before they break). Rather than light, these remote controls use extremely low-frequency electromagnetic waves that are invisible but can travel much further than infrared.

The type of communication that your remote uses determines its range, but the physical hardware inside is what's really important. If you disassemble a remote that you don't need or feel confident about put back together, you can find out how it works underneath. Most TV remote controls have rubbery, silicone buttons. Each rubber button usually has a small gray or black conductive pad on the underside. This pad presses down onto lots of copper strips that interlock with each other like fingers but aren't actually connected. When the conductive pad presses down on these strips, the two sides connect and activate the button. With some remote controls, such as for RC cars and aircraft, you often find that some pushbuttons are mounted in various ways and are triggered by control sticks. These are similar to those in your Arduino kit.

To control one of these controllers, you need to know about optocouplers, which I cover in the following section.

Making an optocoupler circuit

Because you are using an optocoupler to control a button, the first task is to find that button. Many different kinds of remote controls are available. You may choose a TV remote or a remote for a remote control car, but for this example, I use a wired remote control in the form of an old PlayStation controller.

These buttons all take different forms as well, so it's important to know what to look for. Essentially, each button is a broken connection that is connected by something conductive. On most TV remotes and game console controllers, the button is often be hidden under a layer of rubber or plastic. This layer of rubber or plastic usually has a conductive pad on its underside that connects two conductive surfaces to complete a circuit and trigger an action.

After you have located these two conductive pads, you need something that can replace the button and be controlled by your Arduino: For this you use an *optocoupler*. An optocoupler is a switch that is activated by light — take a look at one in Figure BC-1 — and goes by many other names, including opto-isolator, photo coupler, and optical isolator. The light is enclosed inside the black housing of this tiny, integrated circuit, so you never see it turn on and off, but the switch does.

An optocoupler has four legs, two for each side of the circuit. On one side is an LED and on the other is a phototransistor. The phototransistor is the same as the transistor used in Chapter 8, but instead of closing the circuit when a small voltage is applied, it closes when light is illuminated. This design allows the Arduino side of the circuit to remain electrically isolated from the remote control, meaning that it can integrate with the remote control circuit with very little disruption to the circuit. In this section's example, you learn how to attach an optocoupler to an existing pushbutton to turn it into an Arduino-controlled pushbutton. This is a simple task that requires very little soldering.

For this project, you need:

- Your chosen remote control
- A multimeter
- A soldering iron
- Two lengths of equipment wire
- An Arduino
- A breadboard
- An optocoupler
- Jump wires

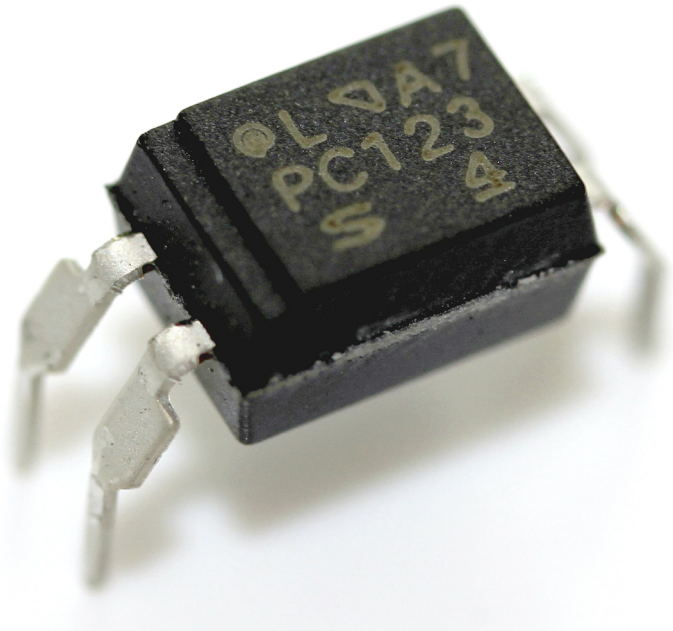

Figure BC-1: A typical optocoupler.

First you need to solder your two lengths of equipment wire onto either side of the existing switch. Doing so makes it easy to incorporate the switch into your breadboard. Make sure that you have enough wire for the circuit board to sit next to the Arduino and breadboard comfortably.

Bare the ends of the wire and twist them between your thumb and forefinger. When they are in a neat, uniform point, tin them with a small amount of solder, as in Figure BC-2. You do so to ensure that they don't unravel, and to make it easier to slot them into a breadboard and solder to other surfaces. When tinning, it is a good idea to slide the soldering iron down to the tip of the exposed wire to scrape away any excess solder. If a bead of solder forms at the tip, you can always cut it off to leave a neat tip.

If you're hacking a battery-powered remote control, remember to remove the batteries before you begin applying heat to live wires.

The first step is to find the button you want to control. For the PlayStation controller, either side of the button is a metal pad, and the part that connects them is attached to the plastic case. After you remove the case, you can easily find the metal pads and attach wires to them, as you can see in Figure BC-2.

The control may also be a pushbutton or switch, so you may need some ingenuity to connect wires to the button. Find the button you want to control.

First you need to remove the batteries from your remote. You need to find out which way it is connected, and the most reliable method is to use your multimeter to check the continuity. When it beeps, there is a connection.

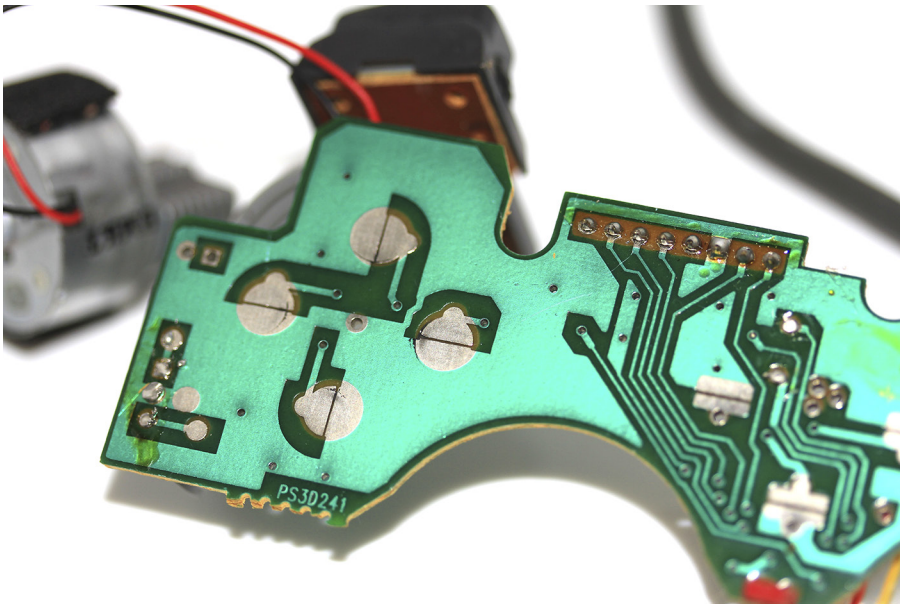

Figure BC-2: Inside a PlayStation controller, you can see the contacts for each button just waiting to be hacked!

Four-leg pushbuttons are often connected in pairs, so while using your multimeter to check the continuity (see Chapter 5), make sure to find a pair of legs that bleeps only when the button is pressed. Mark the legs or make a mental note.

You need to find a surface to solder onto. The PlayStation pads are flat and easy to apply solder to. For pushbuttons, you can either remove the button completely with a pair of clippers and solder directly onto the circuit board or solder a wire onto the leg itself. The leg itself is usually the surest option.

Tin the metal pad or the leg in the same way as you tinned the wire, leaving a small coating of solder.

You also need to find out which way the switch is connected — that is, which side is positive and which is negative. As with the Arduino Button sketch (described in Chapter 7), a button is often connected to ground when not in use, and a small voltage when pressed. To find ground, you can use the continuity checker on your multimeter with other parts that you know are ground, such as the ground of the battery. When it beeps, you have a connection, and by a process of elimination, you know that this side is ground and that the opposing side is supplying the voltage.

In this example, I solder the lengths of wire to the metal pads. Hold the tinned tip of the wire to the tinned pad of the controller and apply heat with your soldering iron. Doing so melts the solder on both halves and helps you get good contact. After the solder has cooled, you have two handy jump wires to connect your remote control to your breadboard.

You might consider three optional extras that add a bit of robustness to your wiring. First, you can use a couple of blobs of hot glue to hold the wires in place and prevent strain from breaking the solder joint. The glue should be on the plastic insulation of the wire, which is extremely strong when pulled on, and on the circuit board itself, which provides a larger surface area for the glue to fix to. Second, you can twist the two wires together to give the two wires a greater strength and make the connection look neater. Third, you can attach a terminal block (choc-block) to the end of the wires, thereby making it easy to connect and disconnect your controller as needed, as I do in Figure BC-3.

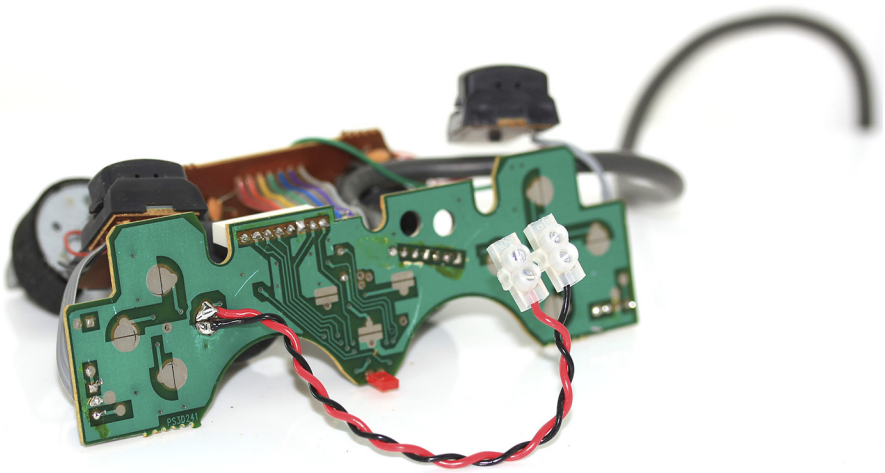

Figure BC-3: Two jump wires connected to a PlayStation button.

Now that you have all the components, you can assemble the breadboard and test the circuit. Complete the circuit as shown in Figures BC-4 and BC-5 to control your remote with your Arduino.

You can break the optocoupler into two halves. On one side is the input, which is essentially an LED light source, similar to the LED on pin 13 of your Arduino. This has a polarity, so the power must be applied to the anode (+), and the cathode (-) must be connected to ground for it to function. The other half is a light-operated transistor. The power must also flow through this half the correct way to avoid damaging the optocoupler. You may need to do a bit of detective work (as mentioned earlier in this section) using your multimeter to discover which side of the switch is supplying the power and which side is grounding it.

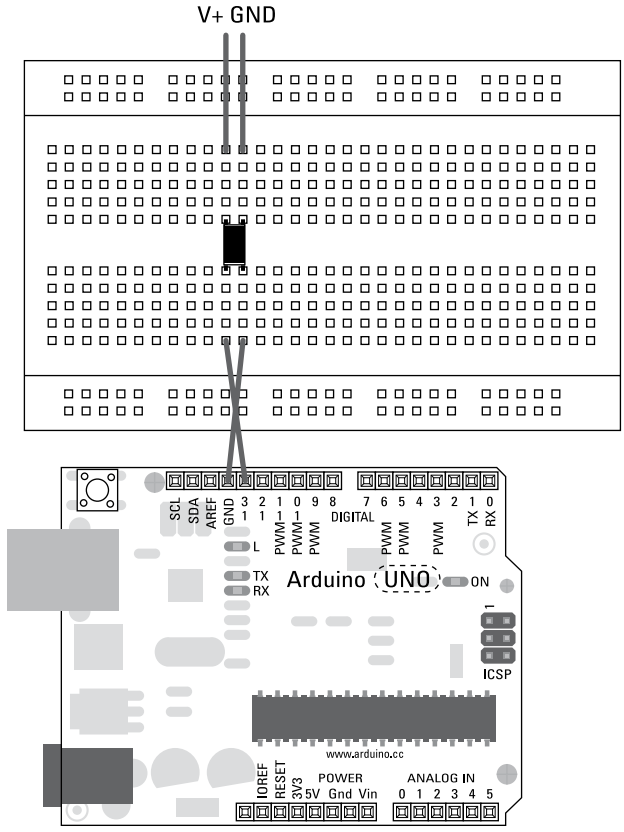

Figure BC-4: The circuit layout for the optocoupler circuit.

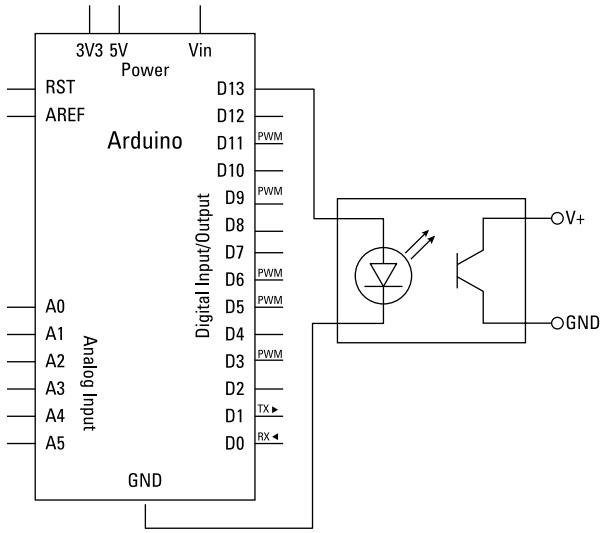

Figure BC-5: The schematic for the optocoupler circuit.

Connect your button to the optocoupler using the lengths of equipment wire and you're ready to assume control of it!

After you've assembled your circuit, you need the appropriate software to use it. From the Arduino menu, choose File➪Examples➪01.Basics➪Blink, and you should be presented with the Blink sketch that follows.

```
/*
  Blink
  Turns on an LED on for one second, then off for one second, repeatedly.

  This example code is in the public domain.
 */

// Pin 13 has an LED connected on most Arduino boards.
// give it a name:
int led = 13;

// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pin as an output.
  pinMode(led, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {
  digitalWrite(led, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);               // wait for a second
  digitalWrite(led, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);               // wait for a second
}
```

After you've uploaded the sketch, the Arduino triggers the optocoupler once every second (on for a second, off for a second). If you've followed the previous steps, nothing happens because the batteries to the remote have been removed. Gotcha!

Unplug your Arduino from the USB port, place the batteries back in the controller, and then plug your Arduino back in. Whatever your button should do should start happening: on for a second, off or a second. The remote for a car may cause it to accelerate in bursts; a remote for your TV may crank the volume up one step at a time.

If you don't see anything happening, double-check your wiring:

- Make sure that you're using the correct pin number.
- Check the connections on the breadboard. If the jump wires or components are not connected using the correct rows in the breadboard, they will not work.

- Check that your optocoupler is in the right position and the correct pins are being used. If in doubt, disconnect the equipment wire and test the legs of the optocoupler with the continuity tester on your multimeter to make sure that it is switching.

If you are unfamiliar with the Blink sketch, check it out in Chapter 4.

Doing more with the optocoupler circuit

The optocoupler circuit is a great way to control remote-controlled objects with a tiny integrated circuit. In this section's example, you simply use Blink to blink the circuit on and off, which is basic, and often all that you need for controlling bigger things. However, you can also use optocouplers to `analogWrite` to output circuits, generating the same output that fades an LED or controls the speed of a motor. Make sure to consider your output, however. If you're switching an office fan on and off, it takes a long time for the fan to get up to speed, so high-frequency pulses are wasted on that particular application.

Whatever you are attempting, always start with something simple like Blink and then adjust the timing after you're happy with the communication.

Making a Digital Single-Lens Reflex (SLR) Shutter Release

A wide variety of software (some of which is mentioned in Chapter 16 in the book) manipulates webcams to capture images, but (for the time being, at least) the resolution and quality of webcams is often quite low, resulting in grainy, unclear images. Webcams with high-quality lenses and good resolution are available but often prohibitively expensive, in a similar price range to most digital SLRs. Wouldn't it be great if you could take pictures remotely with a far superior digital SLR?

Digital SLRs are capable of taking extremely high-resolution images and storing them or passing them directly on to a computer. If you can control one remotely, you open huge possibilities for time-lapse photography, high dynamic range (HDR) photography, and high-speed photography. In this example you learn how to wire up the remote shutter and autofocus connection of your digital SLR to your Arduino.

Remote shutter releases already exist as a product, commonly used for tripod photography to prevent camera movements when pressing down on the shutter release button. Some are on cable and others are wireless, but both have a fairly hefty cost for what they are. They are available from most

retail stores or online photography shops, but for the best prices, look on sites like eBay. In this example, you learn about a basic wired connector.

Inside the plastic trigger case are a couple of switches. Normally there is a slide switch, which triggers the auto focus before the shutter release. On the camera, there is a socket, which varies with the make of camera. Many plugs and sockets are available for various camera brands, but unfortunately, I can't cover all of them in this book.

My camera is a Canon 60D, which has a three-pole, 2.5mm jack. This is a size down from a 3.5mm headphone jack but looks identical. The number of poles refers to the number of wires used. In this case, there are three wires: one for the autofocus, one for the shutter release, and one for the ground, which triggers either or both. These connectors are widely available, and you can solder them yourself if you fancy a challenge. Figure BC-6 is an illustration of the connector with the appropriate pins.

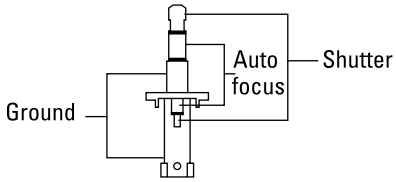

Figure BC-6: Canon 60D remote shutter release diagram.

Soldering a 2.5mm jack

If you are soldering a 2.5mm jack yourself, take great care when soldering the connectors.

Inside the 2.5mm jack, very little space exists for wires that are too long or blobs of solder that are too big. The plastic parts that sit in between the metal contacts are also extremely sensitive to heat, and if you apply the soldering iron for too long, they melt and deform.

First, tin the tip of each wire. A three-core 7/0.2 wire or thinner is recommended. You may find that the insulation on the wire shrinks back when heat is applied; make sure that it is tinned all the way to the end of the insulation and then trim the tinned wire back to 2-3mm. The contacts are very small, so you need only a tiny bit of wire exposed.

Next, tin each contact on the 2.5mm jack. Only a small amount of solder is needed, so make sure to remove any blobs with a solder sucker. Carefully solder the wire and the contacts together. The wires sit on top of each other, so make sure to start with the lowest (least accessible) first and work up. Also make sure that some insulation sits between the contacts. When you have a working socket and three bare wires, perform a quick test. Plug the shutter release into your camera and turn it on. Touch the shutter release wire and the ground wires together — the camera should take a picture. Try setting your camera to auto focus and touch the auto focus and ground wires together as well. Try both, and the camera should focus and then take a picture!

If you have an existing shutter release that you're not using, remove the trigger end by unscrewing and disassembling the switch housing or by simply cutting the cable and stripping it back to reveal the three wires. You can confirm the wires by using the continuity tester on your multimeter. Read the "Soldering a 2.5mm jack" sidebar in this chapter if you plan to solder the 2.5mm jack yourself.

Making a transistor circuit

Now that you have the bare wires, all you have left to do is to make your own circuit to switch them when you want to. You can do so with a simple transistor circuit.

For this project, you need:

- A digital SLR camera
- A remote shutter release cable (bought or homemade)
- A multimeter
- A soldering iron
- An Arduino Uno
- A breadboard
- A transistor
- Jump wires

If you measure the voltage between either wire and ground, it should measure 2.5V, so you can be sure that you're dealing with a low-voltage signal. Many transistors work at such low voltages. Two good examples are P2N2222AG, which is included in many kits, or the TIP120 (or 121/122). Other transistors, such as the IRF510/520 MOSFET, are overkill for such a small application because they are intended for high-powered applications, such as the drill hack in the next section.

The transistor acts the same way as in Chapter 8, opening and closing the gate between the power source(s) and ground to complete the circuit. You can choose to connect only one wire or connect them both to independent transistors. In this example, you connect them both to the transistor circuit. Doing so allows the camera to focus and then trigger every time while set to auto focus. When set to manual focus, the camera just takes pictures, which is considerably faster and better for action shots.

Using your soldering iron, tin the tip of your bare wires to prepare them for use in the breadboard. Assemble the circuit as shown in Figures BC-7 and BC-8 to control your shutter with your Arduino. As noted in Chapter 8, be sure to look up the details of your transistor before you begin. After you have studied the datasheet, connect the Base pin to pin 13 on your Arduino.

Because an onboard resistor is connected to pin 13, you do not need to add one, but if you use another pin, be sure to use an appropriate resistor using the figures from your transistors datasheet and the calculations in Chapter 6.

The Collector should be connected to the shutter pin of the 2.5mm jack and the Emitter should be connected to the ground of the jack.

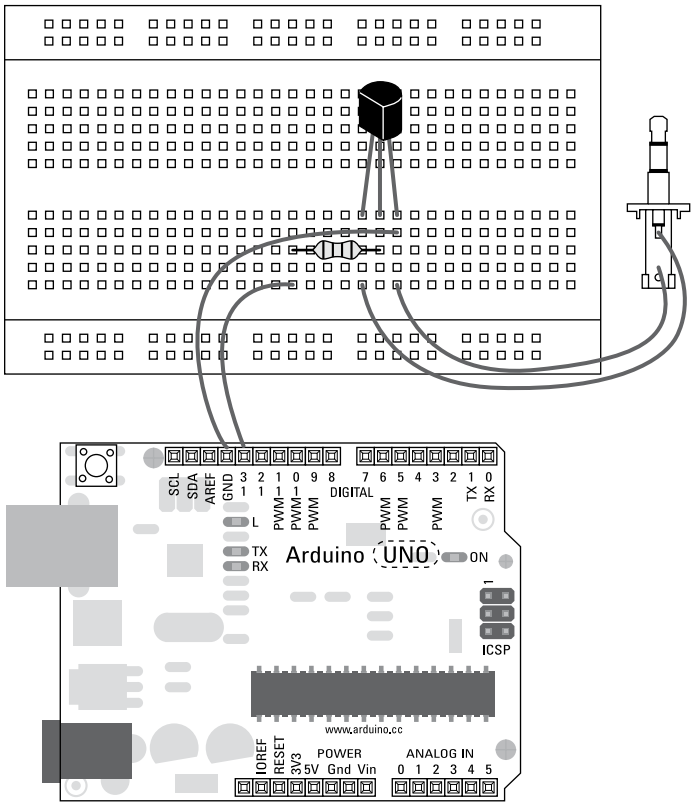

Figure BC-7: The circuit layout for the digital SLR transistor circuit.

After your circuit is assembled, you need the appropriate software to use it. From the Arduino menu, choose File➪Examples➪01.Basics➪Blink and you should be presented with the Blink sketch. To see this sketch, refer to Chapter 4.

Because you are interfacing with hardware, it's important that you know the settings for your camera. The Canon 60D has different settings for the camera trigger, single shot, multiple shots, high-speed multiple shots, timer, and remote control. If the camera is set to single shot, it takes one shot when the trigger is pressed and does not take another until the trigger is fully released and pressed again. With the Blink sketch, you are effectively holding the trigger for one second, releasing for one second, and then repeating. This setting

works fine for taking one picture every two seconds, but if you change the trigger mode to multiple shots or high-speed multiple shots, the camera will take as many pictures as it can while the trigger is pressed, stop for a second, and then take another burst of shots. This setting is great for fast-moving subjects but eats up your battery.

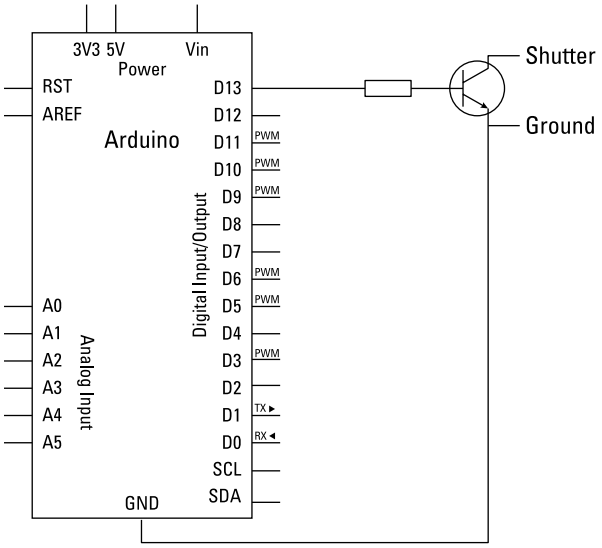

Figure BC-8: The schematic for the digital SLR transistor circuit.

It's best to set your camera to single shot and edit the code to create a shorter delay between the `HIGH` and `LOW` and a longer delay of a few seconds after. For example, the edited delay amounts in the following code are 100 and 3000:

```
digitalWrite(led, HIGH);   // turn the LED on (HIGH is the voltage level)
    delay(100);               // wait for a second
digitalWrite(led, LOW);    // turn the LED off by making the voltage LOW
    delay(3000);               // wait for a second
```

After you've uploaded the code, unplug your Arduino, connect the 2.5mm jack to your camera, and turn the camera on. Reconnect your Arduino, and the camera begins to take one picture every four seconds.

If you don't see anything happening, double-check your wiring:

- Make sure that you're using the correct pin number.
- Check the connections on the breadboard. If the jump wires or components are not connected using the correct rows in the breadboard, they will not work.

- Check that your transistor is the right way around and the correct pins are being used. If in doubt, check the voltage across the collector and emitter legs of the transistor with the voltage setting on your multimeter. Make sure a small voltage is being displayed on the screen of the multimeter each time the pin is triggered.

I cover the Blink sketch in Chapter 4, so if you're unsure of what's going on, check there.

Doing more with a transistor circuit

Now that you have full control of your camera's shutter, you want something to trigger it. An excellent combination is a laser trip wire, as described in Chapter 12. You could use it as a security device to catch people in the act committing a crime or even in a playful way to get high-quality candid shots of people at a party. To get this device out in the real world, you need to make it sturdy enough to survive. You can easily transfer this circuit to a stripboard or Proto Shield and house your circuit in a sturdy box using the tips in Chapter 13.

If you don't fancy the hardwired solution or can't find details for your connector, you have other ways to talk to cameras as well, such as through infrared remotes. Alternatively, hack an existing remote or find a library that can blink an LED at the right frequencies to imitate a remote. Many such solutions appear in the Arduino playground (`http://www.arduino.cc/playground/Main/InterfacingWithHardware#Camera`).

Replacing a Mouse and Keyboard

In Chapters 16 and 17 in the book, you learn about communicating with other software on a computer. You can do this in many ways, but there are also some handy tricks for doing it quickly and easily. One of these methods is to disguise your Arduino as another peripheral that your computer already recognizes, such as a keyboard or mouse. By using a recognized peripheral, you can talk to a computer with virtually no setup at all, just as you can plug in a mouse and instantly use it

There are ways to hack keyboards and mice that involve stripping back the peripherals the way you did in the rest of this chapter's examples. But wouldn't it be nice if Arduino could just pretend to be a mouse or keyboard? It is another USB device, after all. Well, until recently, this wasn't possible, but a recent addition to the Arduino line-up — the Leonardo — changed that situation. Check out the Leonardo in Figure BC-9.

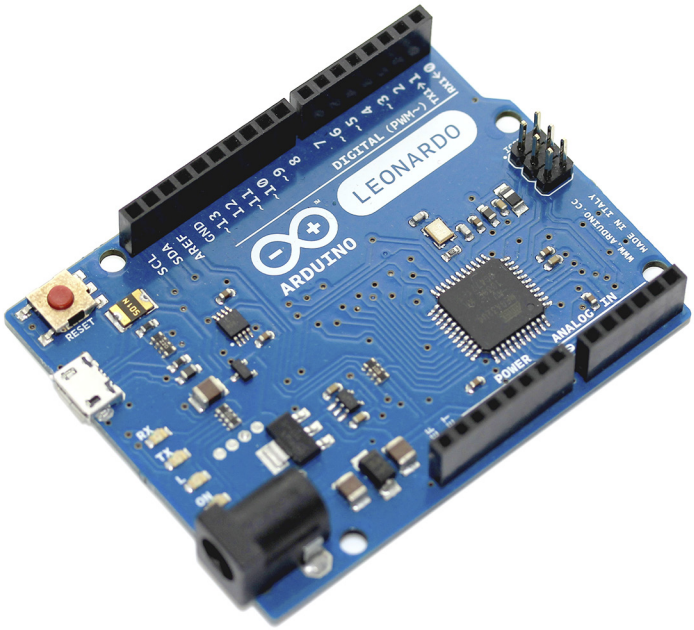

Figure BC-9: A shiny new Leonardo.

The Arduino Leonardo uses a different microcontroller from that of the Uno or Mega. It uses an ATmega32u4 (the "u" stands for USB). It has built-in USB communication, so rather than convert from USB to serial and then back from serial to USB, the Arduino talks directly to the USB port and is treated like another USB device.

The Leonardo has other benefits:

- It can act as a human interface device (HID). This means that the Leonardo can behave like a mouse or keyboard. This capability is excellent for easy interfacing with software and also for making your own unique computer peripherals.
- It can also communicate over serial just as the Uno does for traditional Arduino communication.
- It's cheaper. Because it has fewer components, the board is cheaper to make and, therefore, cheaper to buy. You can also buy the board without header sockets for more savings.
- It has one more PWM pin than an Arduino Uno. Not a huge benefit, but every little bit helps. Pin 13 is now capable of PWM, as well.

Bear in mind an important issue before getting started using the Leonardo. Because the Leonardo is acting as a keyboard or mouse, you can lose control of the inputs on your computer, making it very hard to reprogram. If you tell your Leonardo to type a poem and loop that command, that's exactly what will happen — forever. If that happens in your sketch window, you can imagine the resulting chaos. The best option is to have the keyboard and mouse outputs controlled by a physical switch or button connected to your Arduino, so that you can keep your board plugged in but stop it when you need to. Think of it as an emergency stop button in case things go wrong.

The Leonardo a few other differences from your regular Arduino Uno:

- It has no fixed serial port. Because the serial communication takes place via a virtual serial port, the Leonardo does not have a fixed id. This means that every time the board is reset, the serial port is given a fresh id (/dev/tty.usbmodemXXXXXX or COM X). This is a problem for programs looking for a fixed serial port, so an Uno may be preferable on these occasions.
- The serial port is not reset when opened. On an Uno, the sketch is reset whenever the serial port is opened, either for serial communication or the serial monitor. This is not the case with the Leonardo, which can mean that Serial.print, Serial.println, or Serial.write lines in the `setup()` aren't seen. One technique for avoiding missing these statements is to check for a serial connection:

```
// while the serial connection is not present, do nothing
while (!Serial);
```

- Serial communication is considerably faster. Because there is no serial to USB conversion, the Leonardo is capable of filling your computer's serial buffer a lot faster than previous boards. If you find your serial monitor slowing or lagging considerably, inserting a small delay of just 1 millisecond helps.

```
// wait for 1 millisecond
delay(1);
```

- A micro-USB socket replaces the conventional USB socket. Instead of the usual USB A to B lead, you need a USB A to micro B like those on most Kindle devices and Android phones and tablets.
- The indicator LEDs have been moved. Instead of the usual arrangement of LEDs on the Arduino Uno and previous boards, all the LEDs are found along the left-hand edge of the board between the micro USB socket and the power socket (see Figure BC-10).

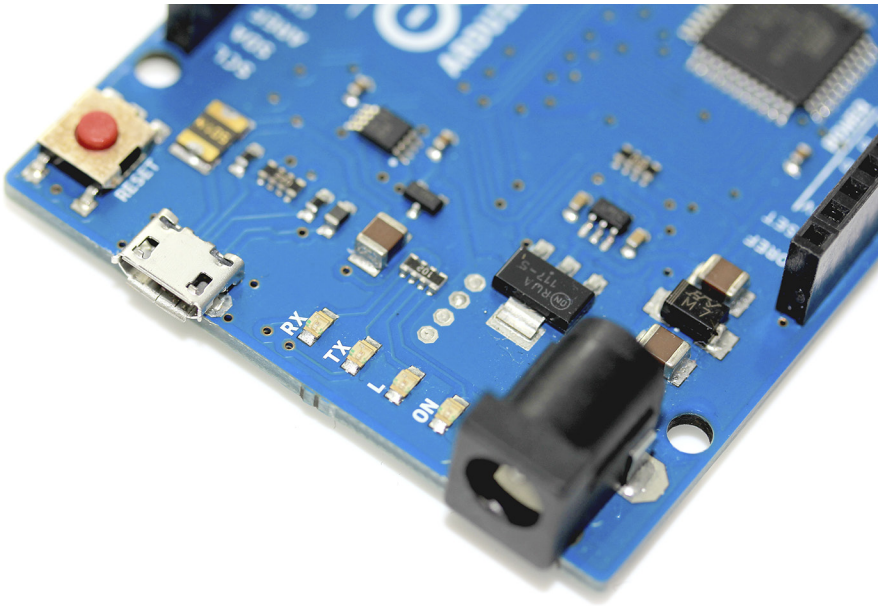

Figure BC-10: A new LED arrangement for the Leonardo.

Setting up the Leonardo

As with any new device on your computer, you need to ensure that that device is recognized for it to operate correctly.

On Mac OS, setting up the Leonardo is straightforward. When you first plug your Leonard in, the Keyboard Setup Assistant informs you that "Your keyboard cannot be identified." There is nothing else to configure for the Leonardo; simply click the red button to close the window, and you are good to go.

In Windows 7, you must find the correct drivers. When you first plug in your Leonardo, the Found New Hardware Wizard attempts to find the drivers. If it doesn't, follow these steps:

1. **Go to the Start menu and, in the Search Programs and Files box, enter** Device Manager**.**
2. **Select Device Manager from the list that appears.**

   You should see a list of every device connected to your computer.

3. **Find Arduino Leonardo and right-click it.**
4. **Choose Update Driver Software and then, in the window that appears, choose Browse My Computer for Driver Software.**
5. **Navigate to your Arduino application folder and the Drivers folder it contains, and click OK.**
6. **Continue through the menus to finish installation of the drivers and your Leonardo will be ready to go.**

If you're using Ubuntu 10.0.4 or later, you don't need to install any drivers.

Implementing the KeyboardMessage sketch

In this example, you learn how to imitate a keyboard using an Arduino Leonardo. In the KeyboardMessage sketch, your Leonardo types a message every time a physical pushbutton is pressed. The message is a mixture of predefined text and a variable counter that increases by one with every button press.

You need:

- An Arduino Leonardo
- A breadboard
- A pushbutton
- A 10k ohm resistor
- Jump wires

The circuit is the same as the basic Button sketch (covered in Chapter 7), but with the Leonardo, you have the additional option of a keyboard or mouse output. This feature can be great for communicating with other onscreen software or even logging data to a text document. Complete the circuit as shown in Figures BC-11 and BC-12.

On one side, the button is connected to the 5V supply on your Arduino. On the other side, the button is connected to Leonardo pin 2 and also to GND, via a 10k resistor.

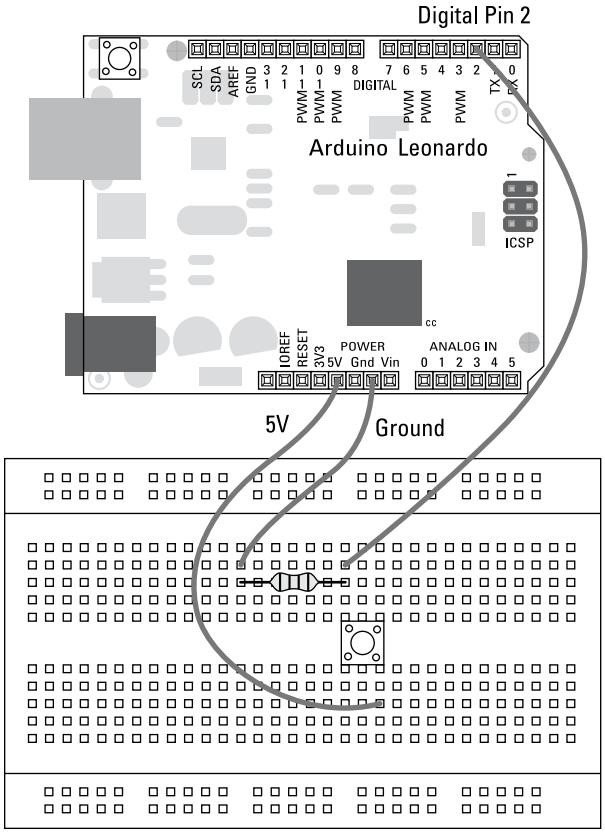

Figure BC-11: A circuit diagram of a pushbutton circuit talking to digital pin 2.

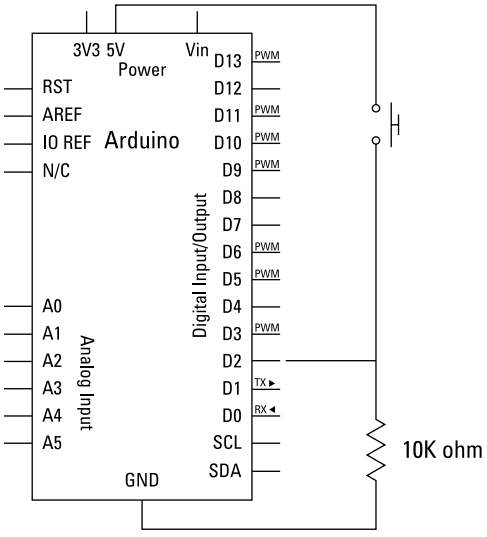

Figure BC-12: A schematic of a pushbutton circuit.

When your circuit is assembled, you need the appropriate software to use it. From the Arduino menu, choose File➪Examples➪09.USB➪Keyboard➪ KeyboardMessage.

```
/*
 Keyboard Button test

 Sends a text string when a button is pressed.

 The circuit:
 * pushbutton attached from pin 2 to +5V
 * 10-kilohm resistor attached from pin 4 to ground

 created 24 Oct 2011
 modified 27 Mar 2012
 by Tom Igoe

 This example code is in the public domain.

 http://www.arduino.cc/en/Tutorial/KeyboardButton
 */

const int buttonPin = 2;          // input pin for pushbutton
int previousButtonState = HIGH;   // for checking the state of a pushButton
int counter = 0;                  // button push counter

void setup() {
  // make the pushButton pin an input:
  pinMode(buttonPin, INPUT);
  // initialize control over the keyboard:
  Keyboard.begin();
}

void loop() {
  // read the pushbutton:
  int buttonState = digitalRead(buttonPin);
  // if the button state has changed,
  if ((buttonState != previousButtonState)
    // and it's currently pressed:
  && (buttonState == HIGH)) {
    // increment the button counter
    counter++;
    // type out a message
    Keyboard.print("You pressed the button ");
    Keyboard.print(counter);
    Keyboard.println(" times.");
  }
  // save the current button state for comparison next time:
  previousButtonState = buttonState;
}
```

After the sketch has successfully uploaded, open a text editor. Using your mouse, make sure that the text entry part of the application is selected and press the button. You should see a line of text stating, "You pressed the button 1 times." Not the best grammar in the world, but an excellent example of the keyboard emulation working. This message appears every time the button is pressed to keep you informed of the number of button presses.

Note that if you unplug your Leonardo and plug it in again, you see the Pin 13 LED marked "L" fade up and down while the board initializes. During this time, your Leonardo does not respond to inputs, and you may receive incomplete responses if someone interacts with the board. It's best to wait until the fading has stopped before using the board to avoid these false starts.

If you don't see a message, double-check your wiring:

- Make sure that you're using the correct pin number.
- Check the connections on the breadboard. If the jump wires or components are not connected using the correct rows in the breadboard, they will not work.
- Check that the upload was successful and that the correct board and serial port are selected.

Understanding the KeyboardMessage sketch

Some comments in the sketch have a couple of mistakes that can lead to confusion.

The line "* 10-kilohm resistor attached from pin 4 to ground" should refer to pin 2, not pin 4.

The constant button pin is declared as pin 2. Because this sketch is monitoring button presses, a variable is needed to store the previous button state. Because the first reading will be `LOW`, the previous value is set to `HIGH` to start with. The counter value is also declared and initialized with a zero value.

```
const int buttonPin = 2;          // input pin for pushbutton
int previousButtonState = HIGH;   // for checking the state of a pushButton
int counter = 0;                  // button push counter
```

In `setup`, the `buttonPin` is declared as an input.

```
void setup() {
  // make the pushButton pin an input:
  pinMode(buttonPin, INPUT);
```

The Leonardo-specific function `Keyboard.begin` is called to initialize communication as a keyboard. This is similar to the way `Serial.begin()` is called to start serial communication.

```
  // initialize control over the keyboard:
  Keyboard.begin();
}
```

The first task in the loop is to read the state of the `buttonPin`. This reading is then stored to the local (temporary) variable `buttonState`, which is cleared the next time the sketch loops.

```
void loop() {
  // read the pushbutton:
  int buttonState = digitalRead(buttonPin);
```

An `if` statement checks to see whether the current `buttonState` is not equal to (`!=`) the `previousButtonState` and (`&&`) if `buttonState` is equal to (`==`) a `HIGH` value.

```
  // if the button state has changed,
  if ((buttonState != previousButtonState)
    // and it's currently pressed:
  && (buttonState == HIGH)) {
```

If this statement is true, the button has just been pressed, so the counter is incremented and the keyboard message is displayed.

```
    // increment the button counter
    counter++;
```

The message is broken into three lines. A line of text is followed by the numerical value of the counter variable, and then a line of text completes the sentence and ends the line. `Keyboard.println()` indicates a carriage return starting a new line for the next message.

```
    // type out a message
    Keyboard.print("You pressed the button ");
    Keyboard.print(counter);
    Keyboard.println(" times.");
  }
```

The `if` statement ends and the current value becomes the previous value for the next loop.

```
  // save the current button state for comparison next time:
  previousButtonState = buttonState;
}
```

This simple sketch quickly and easily interacts with a software interface on a computer. It could be used for something as simple as hitting the spacebar to advance slides with a single button or to turn button presses into mouse movements, making your PC a game controller. In the next chapter, you build on this technique to communicate with other software using the serial port.

Remember, however, that a key press is preferable on many occasions — something to bear in mind when planning your project.