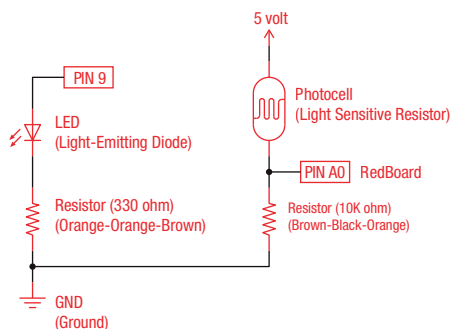


Photo Resistor

So you've already played with a potentiometer, which varies resistance based on the twisting of a knob. In this circuit, you'll be using a photo resistor, which changes resistance based on how much light the sensor receives. Since the RedBoard can't directly interpret resistance (rather, it reads voltage), we use a voltage divider to use our photo resistor. This voltage divider will output a high voltage when it is getting a lot of light and a low voltage when it is not.



PARTS:

Photo Resistor



x1

LED



x1

330Ω Resistor



x1

Wire

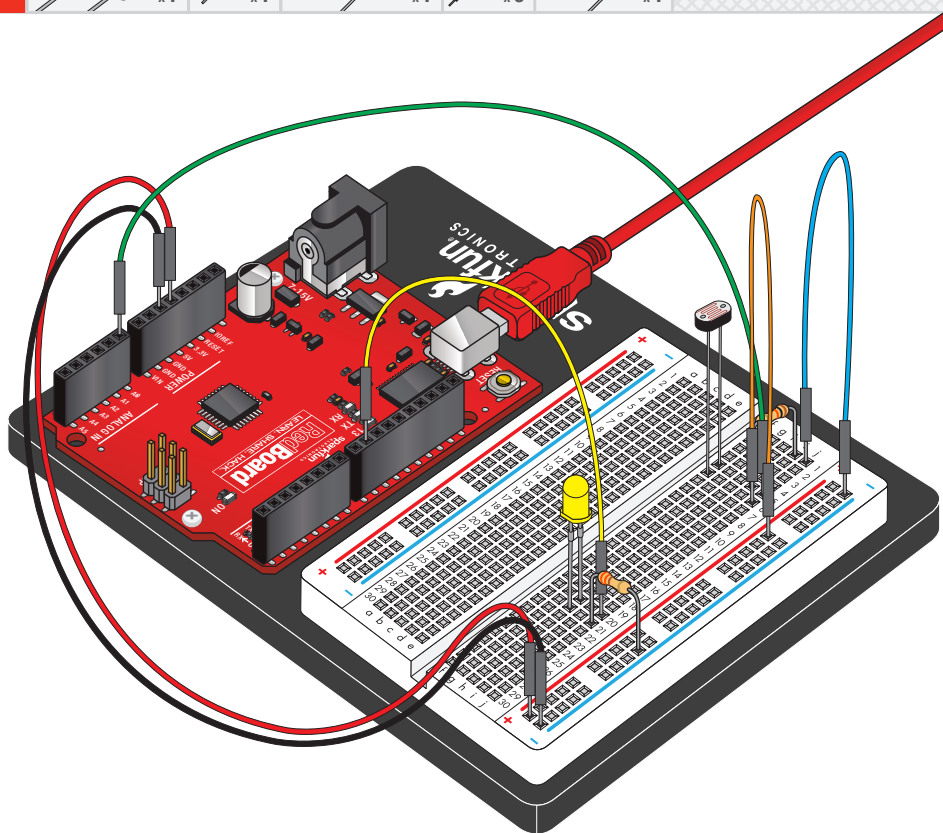


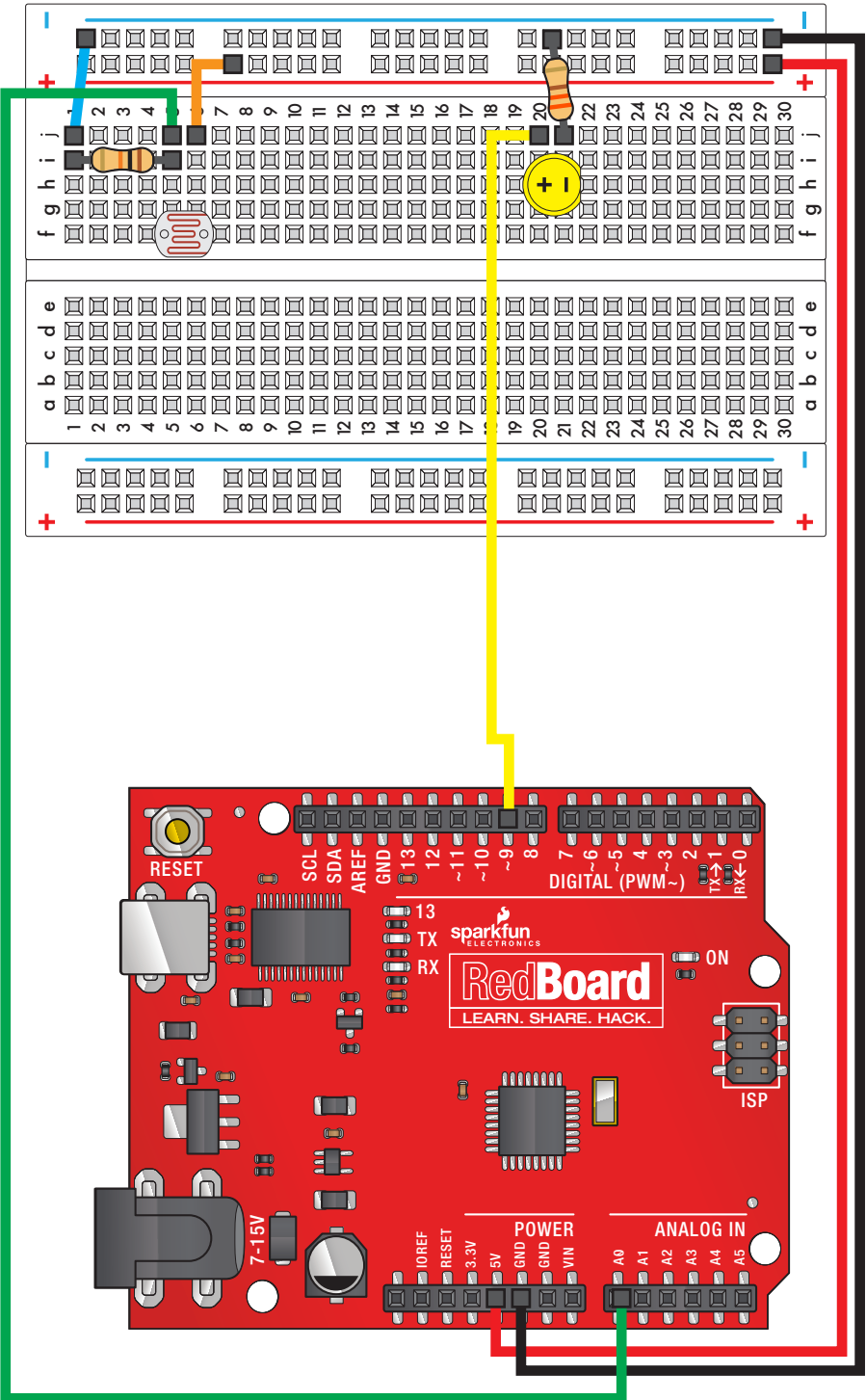
x6













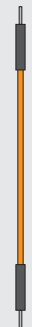
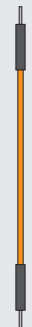


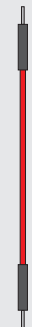
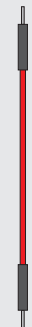


10KΩ Resistor



x1



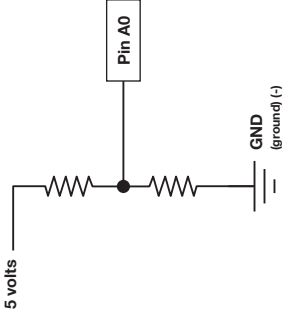


Component:	Image Reference:		
Photo Resistor			f5 - f6
LED (5mm)			h20-h21
330Ω Resistor (sensor)			j21 -
10KΩ Resistor			i1 - i5
Jumper Wire			j1 -
Jumper Wire			j5
Jumper Wire			j6 +
Jumper Wire			j20
Jumper Wire			5V
Jumper Wire			GND

Measuring resistive sensors:

Many of the sensors you'll use (potentiometers, photoresistors, etc.) are resistors in disguise. Their resistance changes in proportion to whatever they're sensing (light level, temperature, sound, etc.).

The RedBoard's analog input pins measure voltage, not resistance. But we can easily use resistive sensors with the RedBoard by including them as part of a "voltage divider".



A voltage divider consists of two resistors. The "top" resistor is the sensor you'll be using. The "bottom" one is a normal, fixed resistor. When you connect the top resistor to 5 volts, and the bottom resistor to ground. The voltage at the middle will be proportional to the bottom resistor relative to the total resistance (top resistor + bottom resistor). When one of the resistors changes (as it will when your sensor senses things), the output voltage will change as well!

Although the sensor's resistance will vary, the resistive sensors (flex sensor light sensor, softpot, and trimpot) in the SIK are around 10K ohms. We usually want the fixed resistor to be close to this value, so using a 10K resistor is a great choice for the fixed "bottom" resistor. Please note the fixed resistor isn't necessarily the bottom resistor. We do that with the photodiode only so that more light = more voltage, but it could be flipped and we'd get the opposite response.



Open Arduino IDE // File > Examples > SIK Guide > Circuit # 6

Code to Note:



```
lightLevel = map(lightLevel, 0, 1023, 0, 255);
```

Parameters

map(value, fromLow, fromHigh, toLow, toHigh)

value: the number to map

fromLow: the lower bound of the value's current range

fromHigh: the upper bound of the value's current range

toLow: the lower bound of the value's target range

toHigh: the upper bound of the value's target range

When we read an analog signal using `analogRead()`, it will be a number from 0 to 1023. But when we want to drive a PWM pin using `analogWrite()`, it wants a number from 0 to 255. We can "squeeze" the larger range into the smaller range using the `map()` function.

See <http://arduino.cc/en/reference/map> for more info.

```
lightLevel = constrain(lightLevel, 0, 255);
```

Parameters

constrain(x, a, b)

x: the number to constrain, all data types

a: the lower end of the range, all data types

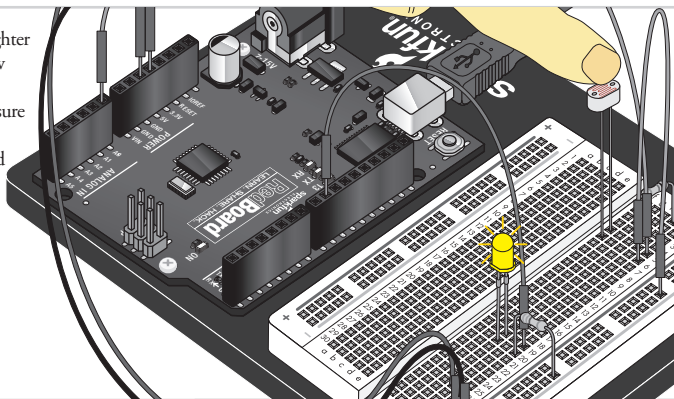
b: the upper end of the range, all data types

Because `map()` could still return numbers outside the "to" range, we'll also use a function called `constrain()` that will "clip" numbers into a range. If the number is outside the range, it will make it the largest or smallest number. If it is within the range, it will stay the same.

See <http://arduino.cc/en/reference/constrain> for more info.

What You Should See:

You should see the LED grow brighter or dimmer in accordance with how much light your photoresistor is reading. If it isn't working, make sure you have assembled the circuit correctly and verified and uploaded the code to your board or see the troubleshooting tips below.



Troubleshooting:

LED Remains Dark

This is a mistake we continue to make time and time again, if only they could make an LED that worked both ways. Pull it up and give it a twist.

It Isn't Responding to Changes in Light

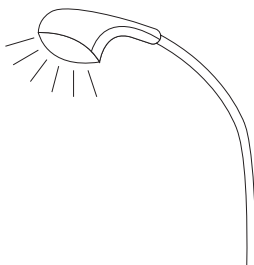
Given that the spacing of the wires on the photo-resistor is not standard, it is easy to misplace it. Double check it's in the right place.

Still Not Quite Working

You may be in a room which is either too bright or dark. Try turning the lights on or off to see if this helps. Or if you have a flashlight near by give that a try.

Real World Application:

A street lamp uses a light sensor to detect when to turn the lights on at night.



Temperature Sensor

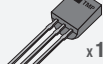
A temperature sensor is exactly what it sounds like – a sensor used to measure ambient temperature. This particular sensor has three pins – a positive, a ground, and a signal. This is a linear temperature sensor. A change in temperature of one degree centigrade is equal to a change of 10 millivolts at the sensor output. The TMP36 sensor has a nominal 750 mV at 25°C (about room temperature). In this circuit, you'll learn how to integrate the temperature sensor with your RedBoard, and use the Arduino IDE's serial monitor to display the temperature.



When you're building the circuit be careful not to mix up the transistor and the temperature sensor, they're almost identical. Look for "TMP" on the body of the temperature sensor.

PARTS:

Temp. Sensor

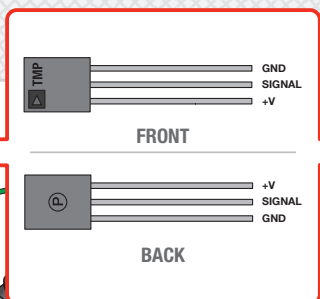


x 1

Wire

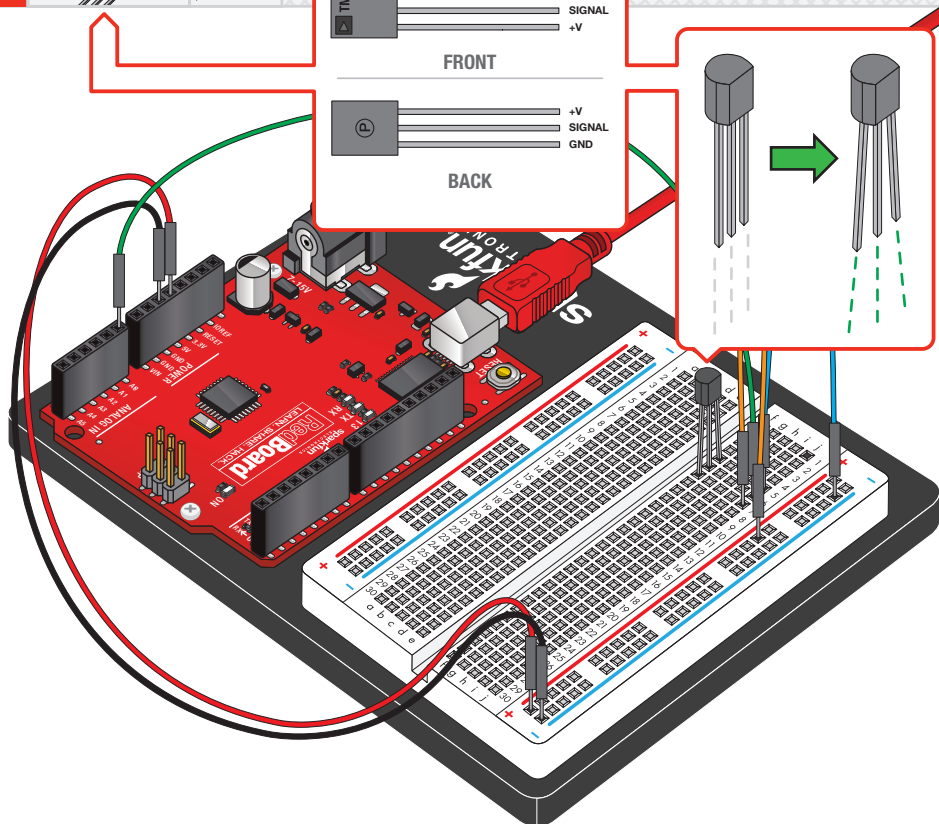


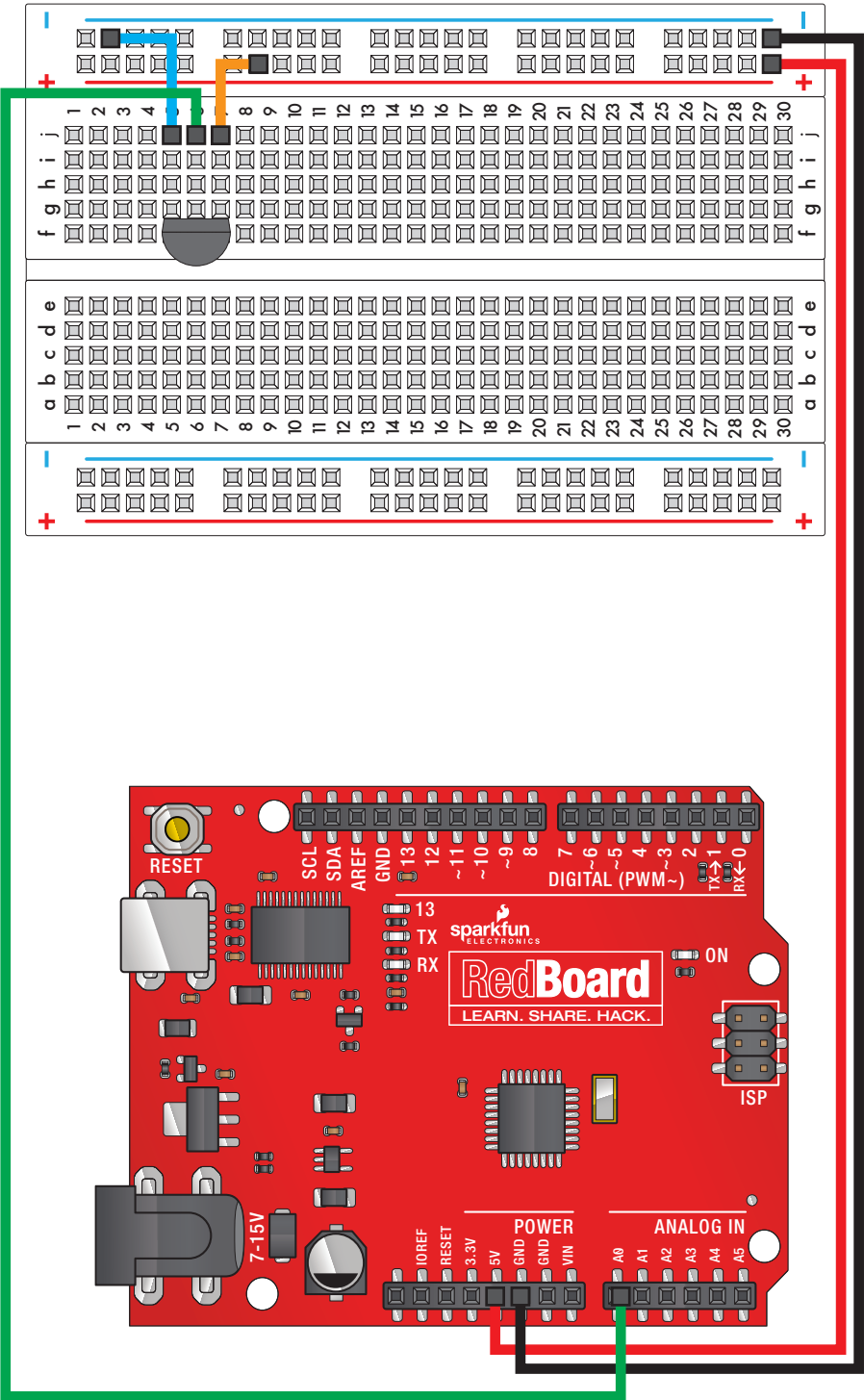
x 5










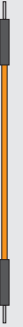










FRONT

BACK

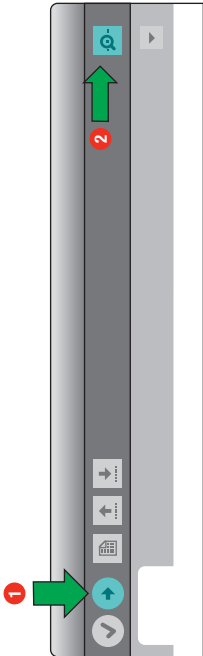

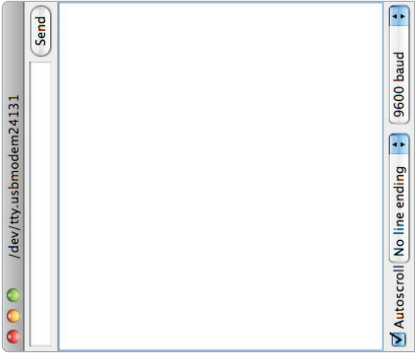




Component:	Image Reference:		
Temperature Sensor			
Jumper Wire			
Jumper Wire			
Jumper Wire			
Jumper Wire			
Jumper Wire			

Opening your serial monitor:

This circuit uses the Arduino IDE's **serial monitor**. To open this, first upload the program then click the button which looks like a magnifying glass in a square. In order for the serial monitor to operate correctly it must be set to the same baud rate (speed in bits per second) as the code you're running. This code runs at 9600 baud; if the baud rate setting is not 9600, change it to 9600.

Page 47



Open Arduino IDE // File > Examples > SIK Guide > Circuit # 7

Code to Note:



`Serial.begin(9600);`



Before using the serial monitor, you must call `Serial.begin()` to initialize it. 9600 is the "baud rate", or communications speed. When two devices are communicating with each other, both must be set to the same speed.

`Serial.print(degreesC);`



The `Serial.print()` command is very smart. It can print out almost anything you can throw at it, including variables of all types, quoted text (AKA "strings"), etc.

See <http://arduino.cc/en/serial/print> for more info.

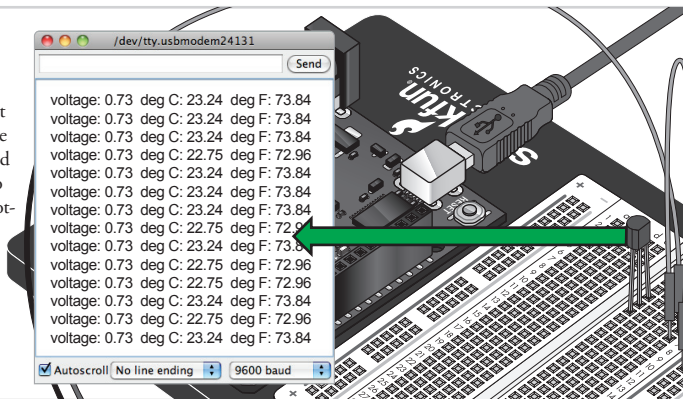
`Serial.println(degreesF);`



`Serial.print()` will print everything on the same line. `Serial.println()` will move to the next line. By using both of these commands together, you can create easy-to-read printouts of text and data.

What You Should See:

You should be able to read the temperature your temperature sensor is detecting on the serial monitor in the Arduino IDE. If it isn't working, make sure you have assembled the circuit correctly and verified and uploaded the code to your board or see the troubleshooting tips below.



Troubleshooting:

Nothing Seems to Happen

This program has no outward indication it is working. To see the results you must open the Arduino IDE's serial monitor (instructions on previous page).

Gibberish is Displayed

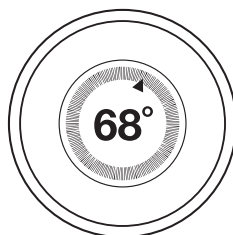
This happens because the serial monitor is receiving data at a different speed than expected. To fix this, click the pull-down box that reads "**** baud" and change it to "9600 baud".

Temperature Value is Unchanging

Try pinching the sensor with your fingers to heat it up or pressing a bag of ice against it to cool it down.

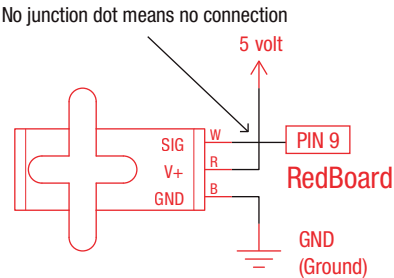
Real World Application:

Building climate control systems use a temperature sensor to monitor and maintain their settings.

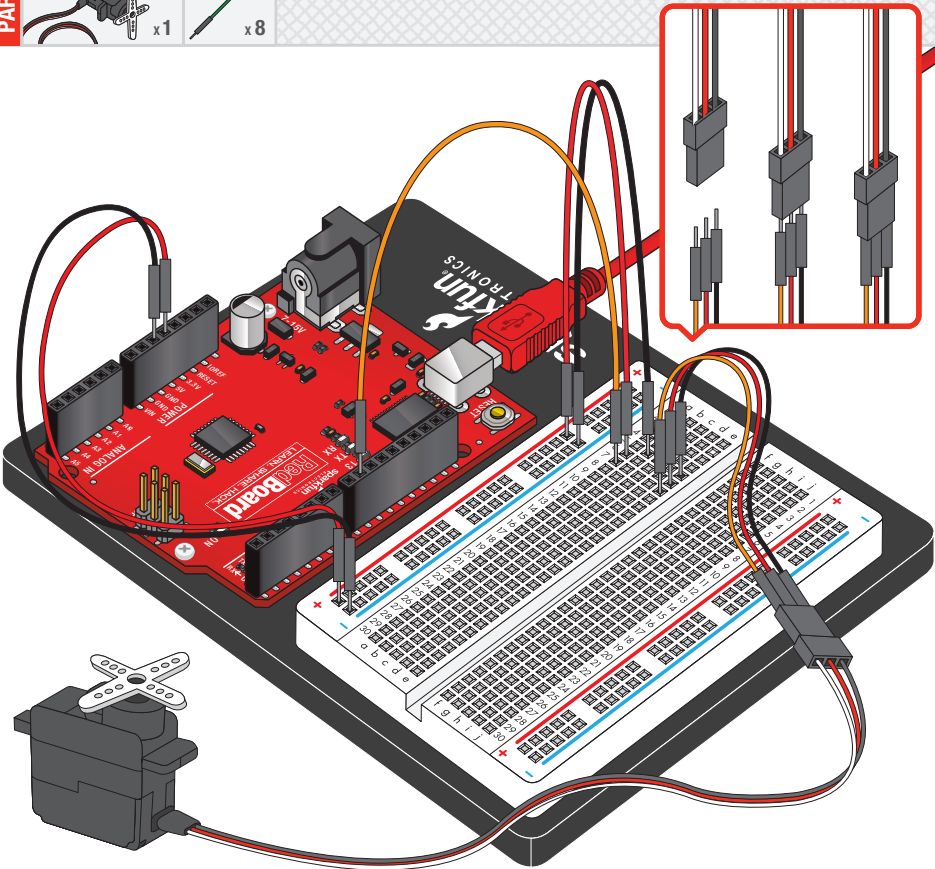


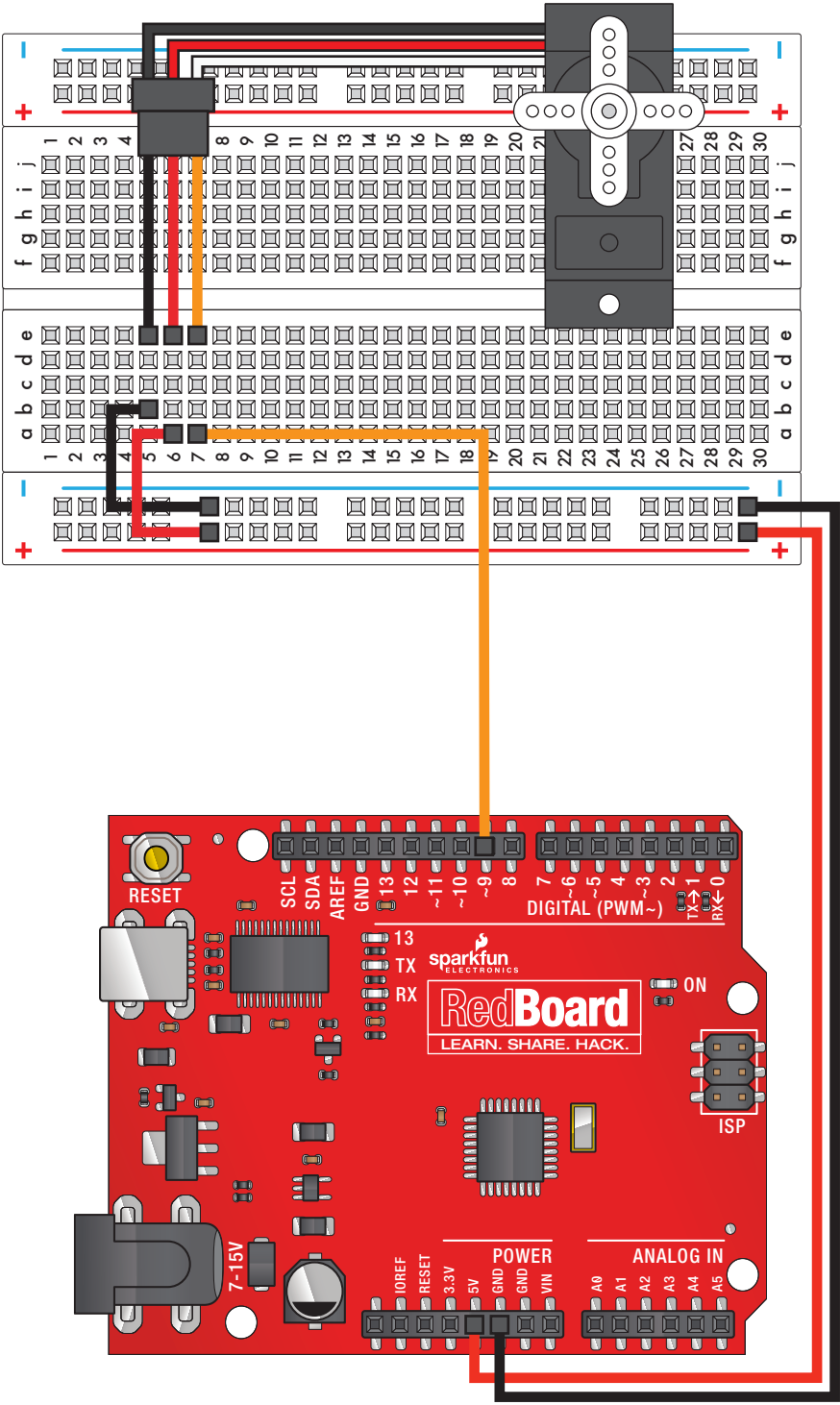
A Single Servo



























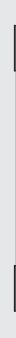


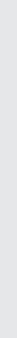


Servos are ideal for embedded electronics applications because they do one thing very well that motors cannot – they can move to a position accurately. By varying the pulse width of the output voltage to a servo, you can move a servo to a specific position. For example, a pulse of 1.5 milliseconds will move the servo 90 degrees. In this circuit, you'll learn how to use PWM (pulse width modulation) to control and rotate a servo.



- PARTS:**
- Servo x1
 - Wire x8





Component:	Image Reference:		
Servo			
Jumper Wire			
Jumper Wire			
Jumper Wire			
Jumper Wire			
Jumper Wire			
Jumper Wire			
Jumper Wire			
Jumper Wire			
Jumper Wire			

Expand your horizons using Libraries:

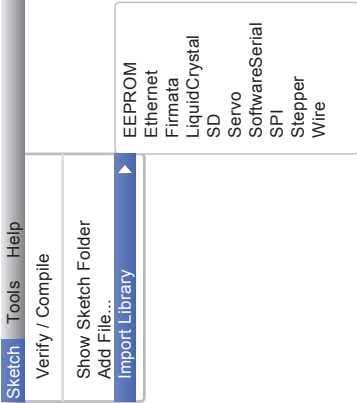
The Arduino development environment gives you a very useful set of built-in commands for doing basic input and output, making decisions using logic, solving math problems, etc. But the real power of Arduino is the huge community using it and their willingness to share their work.

Libraries are collections of new commands that have been packaged together to make it easy to include them in your sketches. Arduino comes with a handful of useful libraries, such as the servo library used in this example, that can be used to interface to more advanced devices (LCD displays, stepper motors, ethernet ports, etc.)

See <http://arduino.cc/en/reference/libraries> for a list of the standard libraries and information on using them.

But anyone can create a library, and if you want to use a new sensor or output device, chances are that someone out there has already written one that interfaces that device to the RedBoard. Many of SparkFun's products come with Arduino libraries, and you can find even more using Google and the Arduino Playground at <http://arduino.cc/playground/>. When YOU get the RedBoard working with a new device, consider making a library for it and sharing it with the world!

To use a library in a sketch, select it from **Sketch > Import Library**.



After importing the library into your code, you will have access to a number of pre-written commands and functions. More information on how to use the standard library functions can be accessed at: <http://arduino.cc/en/Reference/Libraries>.



Open Arduino IDE // File > Examples > SIK Guide > Circuit # 8

Code to Note:



#include <Servo.h>



#include is a special "preprocessor" command that inserts a library (or any other file) into your sketch. You can type this command yourself, or choose an installed library from the "sketch / import library" menu.

Servo servo1;



The servo library adds new commands that let you control a servo. To prepare the Arduino to control a servo, you must first create a Servo "object" for each servo (here we've named it "servo1"), and then "attach" it to a digital pin (here we're using pin 9).

servo1.attach(9);

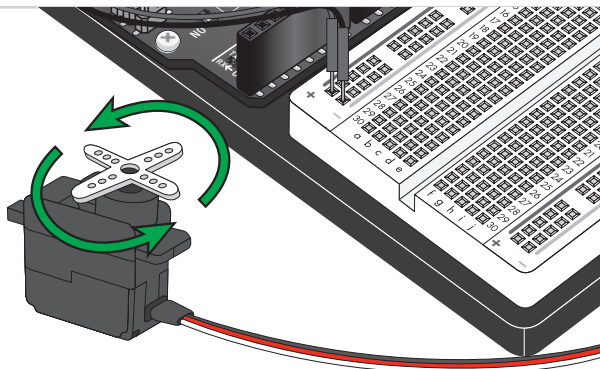
servo1.write(180);



The servos in this kit don't spin all the way around, but they can be commanded to move to a specific position. We use the servo library's write() command to move a servo to a specified number of degrees(0 to 180). Remember that the servo requires time to move, so give it a short delay() if necessary.

What You Should See:

You should see your servo motor move to various locations at several speeds. If the motor doesn't move, check your connections and make sure you have verified and uploaded the code, or see the troubleshooting tips below.



Troubleshooting:

Servo Not Twisting

Even with colored wires it is still shockingly easy to plug a servo in backward. This might be the case.

Still Not Working

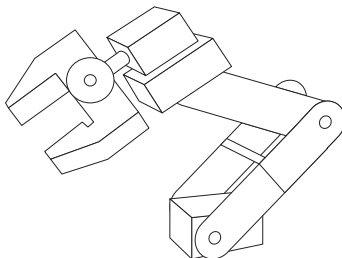
A mistake we made a time or two was simply forgetting to connect the power (red and brown wires) to +5 volts and ground.

Fits and Starts

If the servo begins moving then twitches, and there's a flashing light on your RedBoard, the power supply you are using is not quite up to the challenge. Using a wall adapter instead of USB should solve this problem.

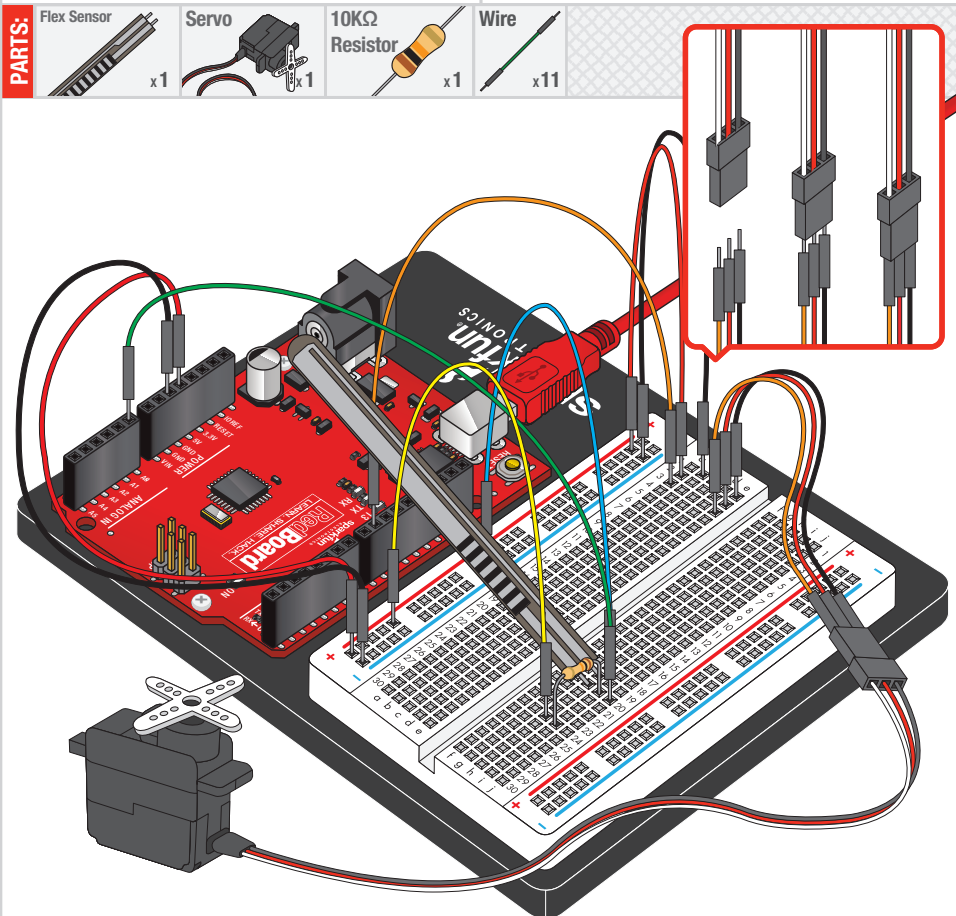
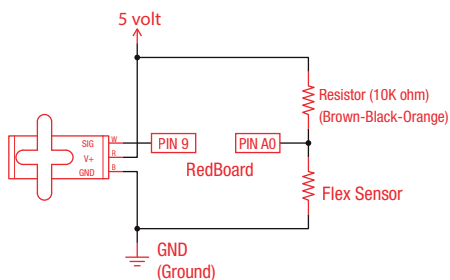
Real World Application:

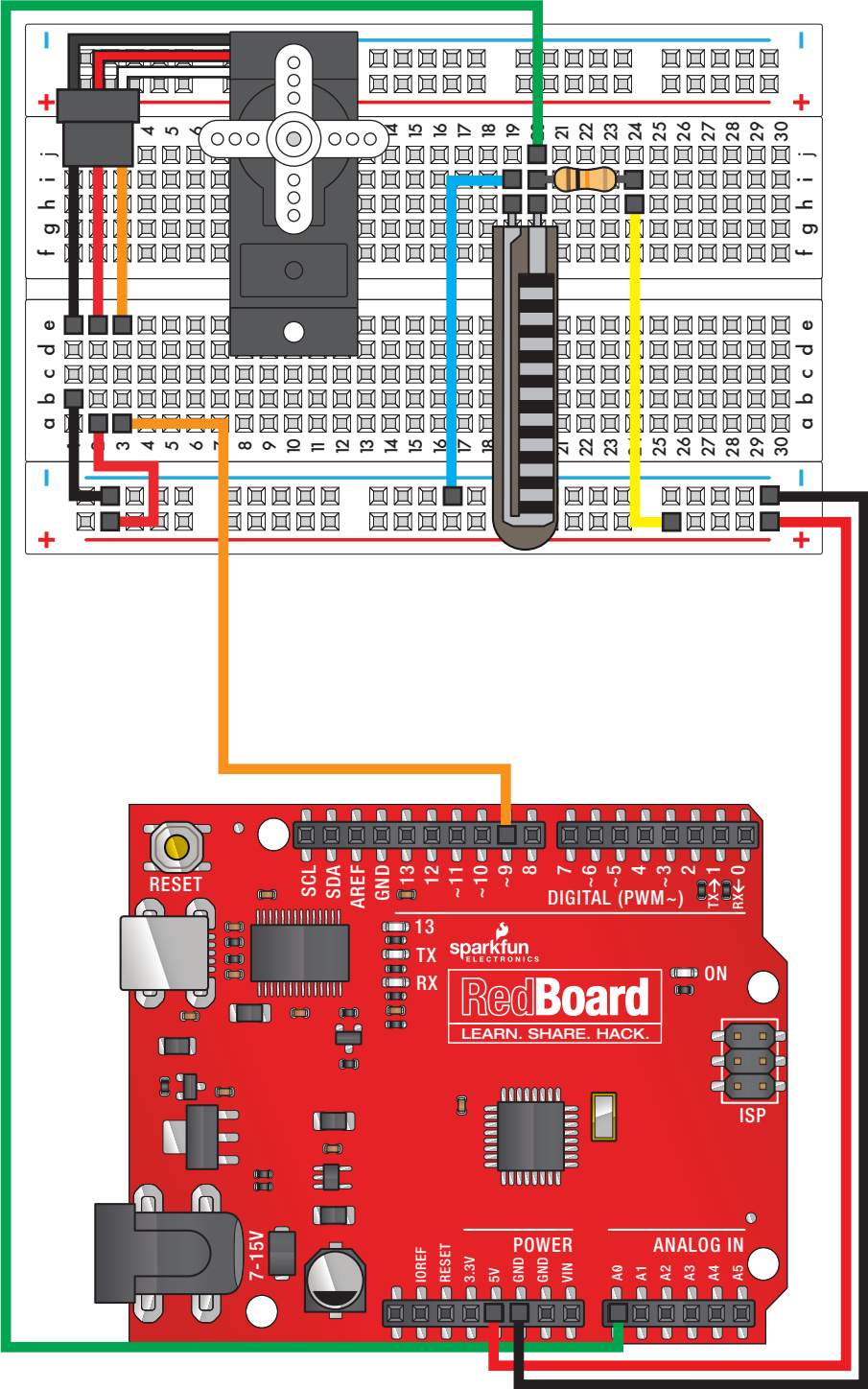
Robotic arms you might see in an assembly line or sci-fi movie probably have servos in them.






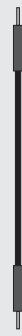













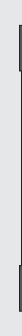


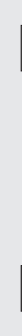


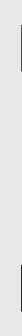


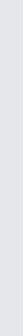


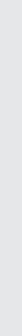


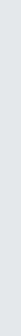


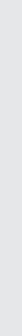


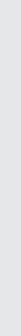




Flex Sensor

In this circuit, we will use a flex sensor to measure, well, flex! A flex sensor uses carbon on a strip of plastic to act like a variable resistor, but instead of changing the resistance by turning a knob, you change it by flexing (bending) the component. We use a "voltage divider" again to detect this change in resistance. The sensor bends in one direction and the more it bends, the higher the resistance gets; it has a range from about 10K ohm to 35K ohm. In this circuit we will use the amount of bend of the flex sensor to control the position of a servo.





Component:	Image Reference:		
Servo			
Jumper Wire			
Jumper Wire			
Jumper Wire			
Flex Sensor			
10KQ Resistor			
Jumper Wire			
Jumper Wire			
Jumper Wire			
Jumper Wire			
Jumper Wire			
Jumper Wire			
Jumper Wire			
Jumper Wire			

Debugging your sketches using the Serial Monitor:


It happens to everyone - you write a sketch which successfully compiles and uploads, but you can't figure out why it's not doing what you want it to. Larger computers have screens, keyboards, and mice that you can use to debug your code, but tiny computers like the RedBoard have no such things.

The key to visibility into a microcontroller is output. This can be almost anything, including LEDs and buzzers, but one of the most useful tools is the serial monitor. Using `Serial.print()` and `println()`, you can easily output human-readable text and data from the RedBoard to a window back on the host computer. This is great for your sketch's final output, but it's also incredibly useful for debugging.

```
for (x = 0; x < 8; x++)
{
  Serial.print(x);
}
```


Let's say you wanted a `for()` loop from 1 to 8, but your code just doesn't seem to be working right. Just add `Serial.begin(9600);` to your `setup()` function, and add a `Serial.print()` or `println()` to your loop:

You wanted 1 to 8, but the loop is actually giving you 0 to 7. Whoops! Now you just need to fix the loop.



```
for (x = 1; x < 9; x++)
{
  Serial.print(x);
}
```

And if you run the code again, you'll see the output you wanted:





Open Arduino IDE // File > Examples > SIK Guide > **Circuit # 9**

Code to Note:



```
servoposition = map(flexposition, 600, 900, 0, 180);  
map(value, fromLow, fromHigh, toLow, toHigh)
```



Because the flex sensor / resistor combination won't give us a full 0 to 5 volt range, we're using the map() function as a handy way to reduce that range. Here we've told it to only expect values from 600 to 900, rather than 0 to 1023.

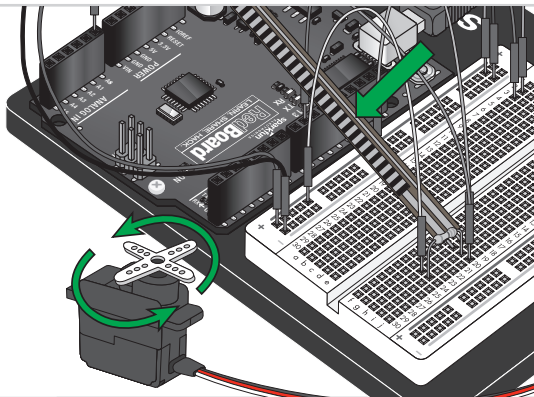
```
servoposition = constrain(servoposition, 0, 180);  
constrain(x, a, b)
```



Because map() could still return numbers outside the "to" range, we'll also use a function called constrain() that will "clip" numbers into a range. If the number is outside the range, it will make it the largest or smallest number. If it is within the range, it will stay the same.

What You Should See:

You should see the servo motor move in accordance with how much you are flexing the flex sensor. If it isn't working, make sure you have assembled the circuit correctly and verified and uploaded the code to your board or see the troubleshooting tips below.



Troubleshooting:

Servo Not Twisting

Even with colored wires it is still shockingly easy to plug a servo in backwards. This might be the case.

Servo Not Moving as Expected

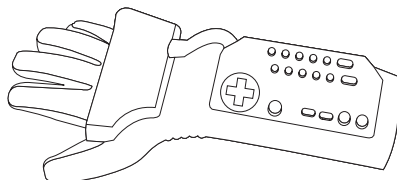
The sensor is only designed to work in one direction. Try flexing it the other way (where the striped side faces out on a convex curve).

Servo Doesn't Move very Far

You need to modify the range of values in the call to the map() function.

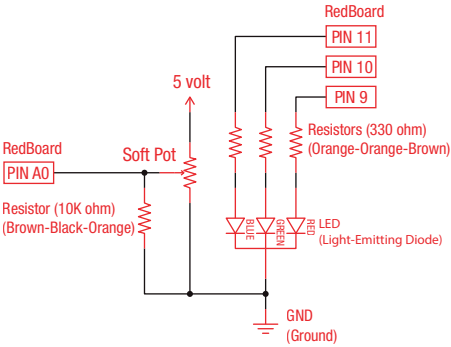
Real World Application:

Controller accessories for video game consoles like Nintendo's "Power Glove" use flex-sensing technology. It was the first video game controller attempting to mimic hand movement on a screen in real time.

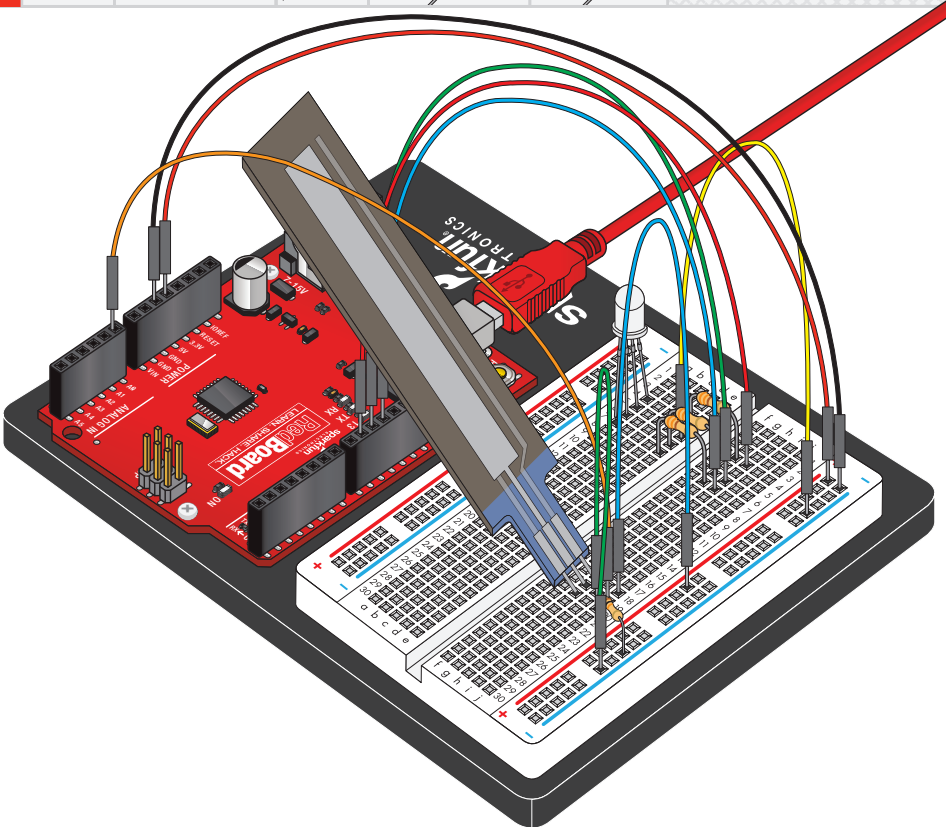
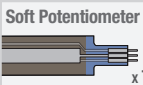


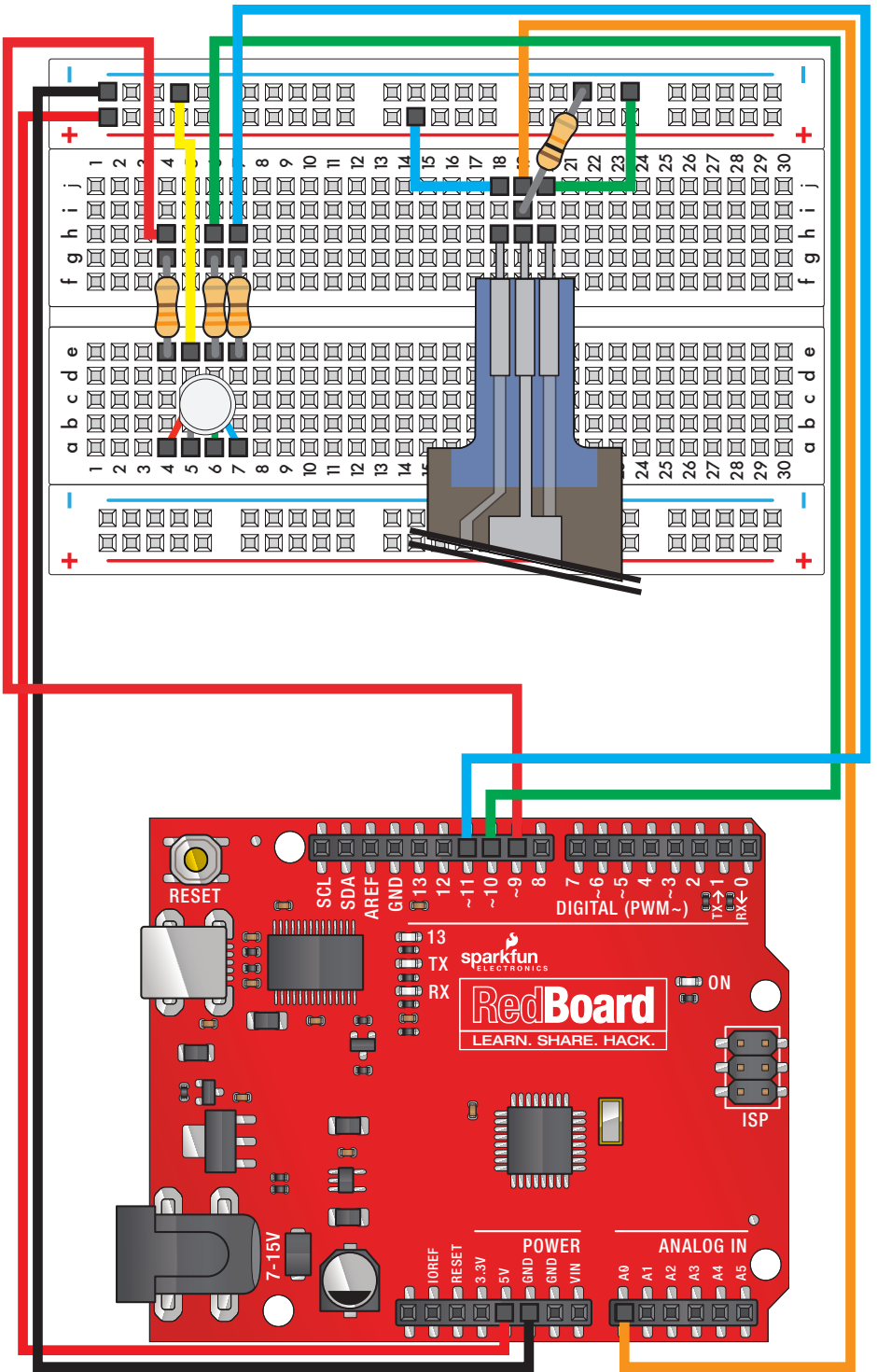
Soft Potentiometer

In this circuit, we are going to use yet another kind of variable resistor – this time, a soft potentiometer (or soft pot). This is a thin and flexible strip that can detect where pressure is being applied. By pressing down on various parts of the strip, you can vary the resistance from 100 to 10K ohms. You can use this ability to track movement on the soft pot, or simply as a button. In this circuit, we'll get the soft pot up and running to control an RGB LED.



PARTS:







Open Arduino IDE // File > Examples > SIK Guide > **Circuit # 10**

Code to Note:

```
redValue = constrain(map(RGBposition, 0, 341, 255, 0), 0, 255)
+ constrain(map(RGBposition, 682, 1023, 0, 255), 0, 255);
```

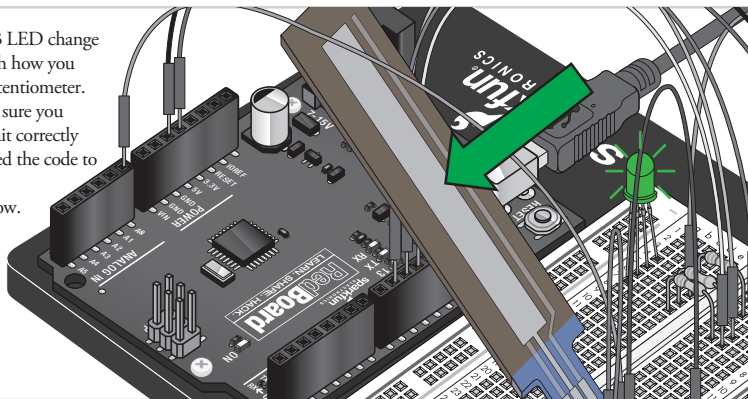
```
greenValue = constrain(map(RGBposition, 0, 341, 0, 255), 0, 255)
- constrain(map(RGBposition, 341, 682, 0, 255), 0, 255);
```

```
blueValue = constrain(map(RGBposition, 341, 682, 0, 255), 0, 255)
- constrain(map(RGBposition, 682, 1023, 0, 255), 0, 255);
```

➡ These big, scary functions take a single Value (RGBposition) and calculate the three RGB values necessary to create a rainbow of color. The functions create three "peaks" for the red, green, and blue values, which overlap to mix and create new colors. See the code for more information! Even if you're not 100% clear how it works, you can copy and paste this (or any) function into your own code and use it yourself. If you want to know more about creating your own functions - take a look at circuit #11.

What You Should See:

You should see the RGB LED change colors in accordance with how you interact with the soft potentiometer. If it isn't working, make sure you have assembled the circuit correctly and verified and uploaded the code to your board, or see the troubleshooting tips below.



Troubleshooting:

LED Remains Dark or Shows Incorrect Color

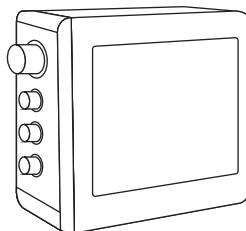
With the four pins of the LED so close together, it's sometimes easy to misplace one. Try double checking each pin is where it should be.

Bizarre Results

The most likely cause of this is if you're pressing the potentiometer in more than one position. This is normal and can actually be used to create some neat results.

Real World Application:

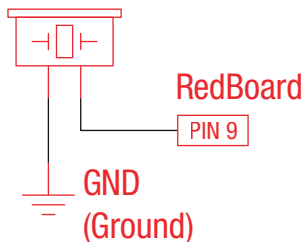
The knobs found on many objects, like a radio for instance, are using similar concepts to the one you just completed for this circuit.



Piezo Buzzer

In this circuit, we'll again bridge the gap between the digital world and the analog world. We'll be using a buzzer that makes a small "click" when you apply voltage to it (try it!). By itself that isn't terribly exciting, but if you turn the voltage on and off hundreds of times a second, the buzzer will produce a tone. And if you string a bunch of tones together, you've got music! This circuit and sketch will play a classic tune. We'll never let you down!

Piezo Buzzer

**PARTS:**

Piezo Buzzer

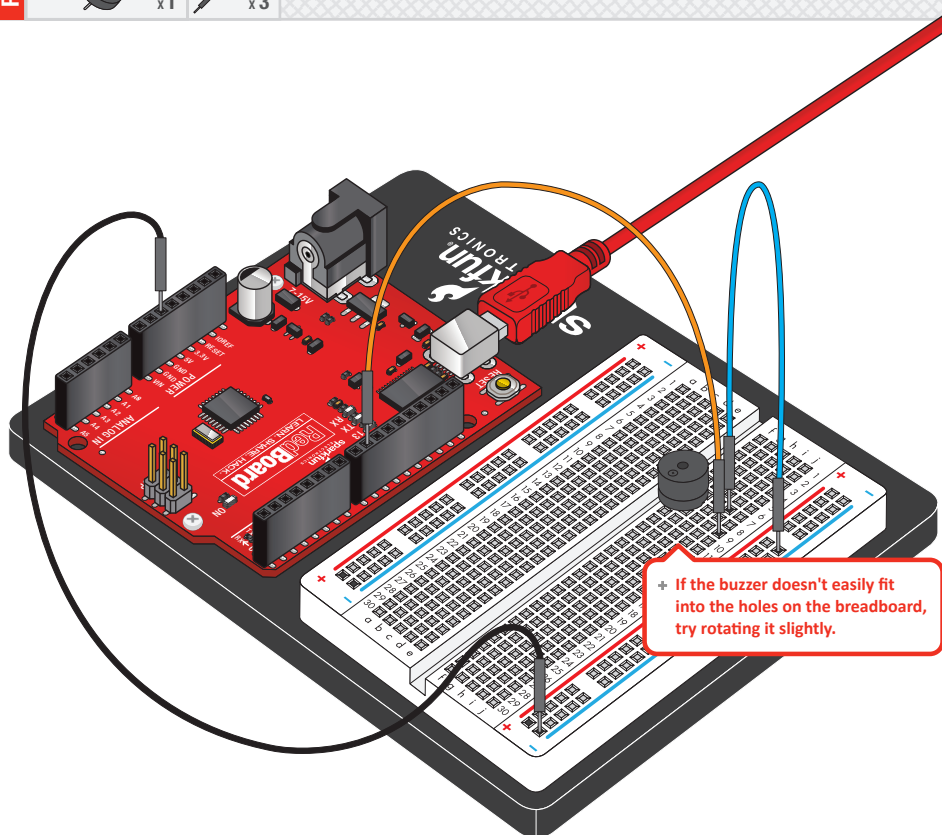


x1

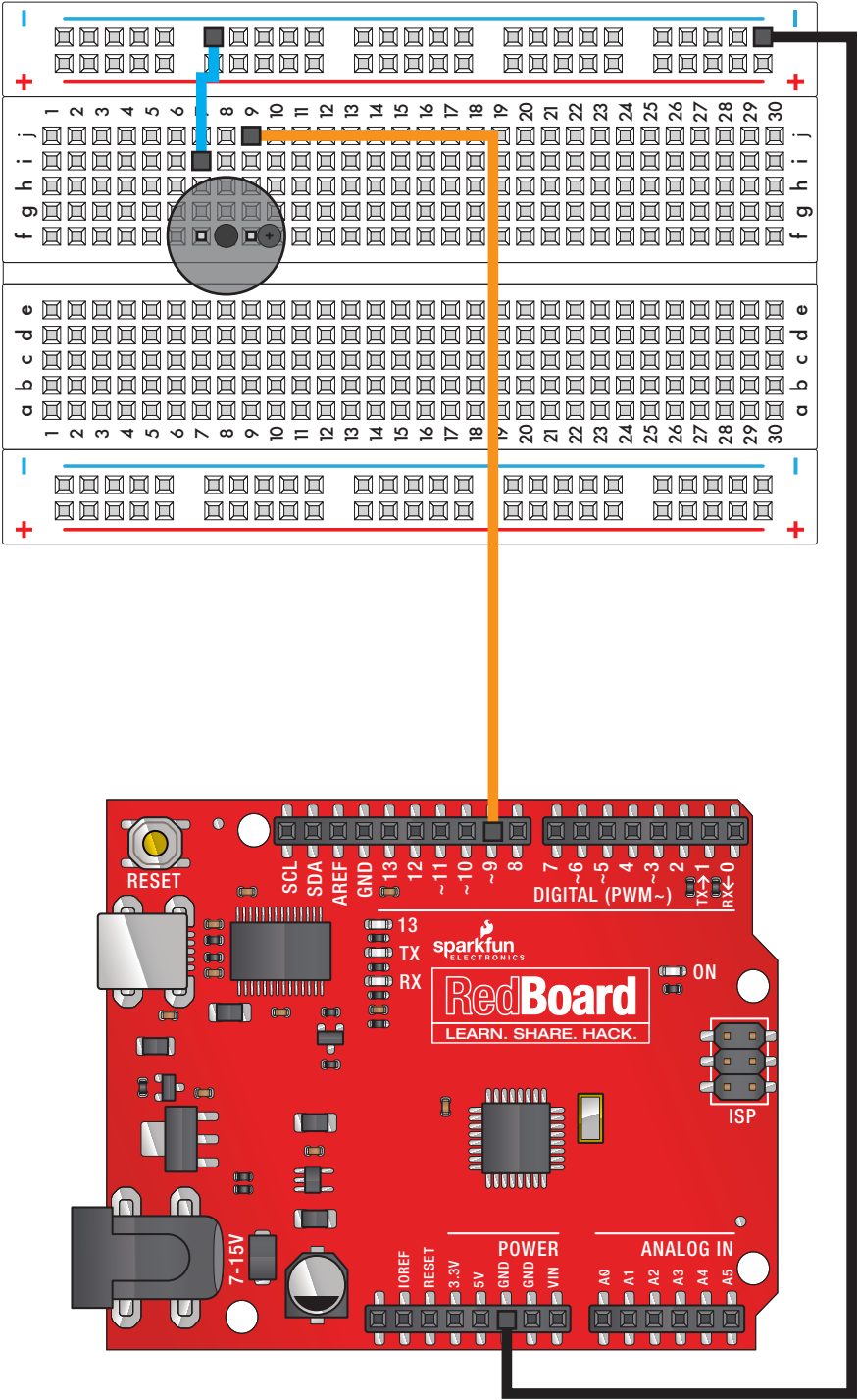
Wire






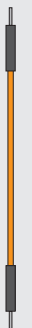


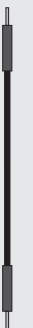




x3



+ If the buzzer doesn't easily fit into the holes on the breadboard, try rotating it slightly.



Component:	Image Reference:		
Piezo Buzzer			
Jumper Wire			
Jumper Wire			
Jumper Wire			

Creating your own functions:

Arduino contains a wealth of built-in functions that are useful for all kinds of things. (See <http://arduino.cc/en/reference> for a list). But you can also easily create your own functions. First, we need to declare a function. Here's a simple example named "add," which adds two numbers together and returns the result. Let's break it down.

```
int add(int parameter1, int parameter2)
{
    int x;
    x = parameter1 + parameter2;
    return(x);
}
```

Your functions can take in values ("parameters"), and return a value, as this one does. If you'll be passing parameters to your function, put them (and their types) in the parentheses after the function name. If your function is not using any parameters, just use an empty parenthesis () after the name.

If your function is returning a value from your function, put the type of the return value in front of the function name. Then in your function, when you're ready to return the value, put in a return(value) statement. If you won't be returning a value, put "void" in front of the function name (similar to the declaration for the setup() and loop() functions).

When you write your own functions, you make your code neater and easier to re-use. See <http://arduino.cc/en/reference/functiondeclaration> for more information about functions.



Open Arduino IDE // File > Examples > SIK Guide > **Circuit # 11**

Code to Note:



```
char notes[] = "cdfda ag cdfdg gf";  
char names[] = {'c','d','e','f','g','a','b','C'};
```



Up until now we've been working solely with numerical data, but the Arduino can also work with text. Characters (single, printable, letters, numbers and other symbols) have their own type, called "char". When you have an array of characters, it can be defined between double-quotes (also called a "string"), OR as a list of single-quoted characters.

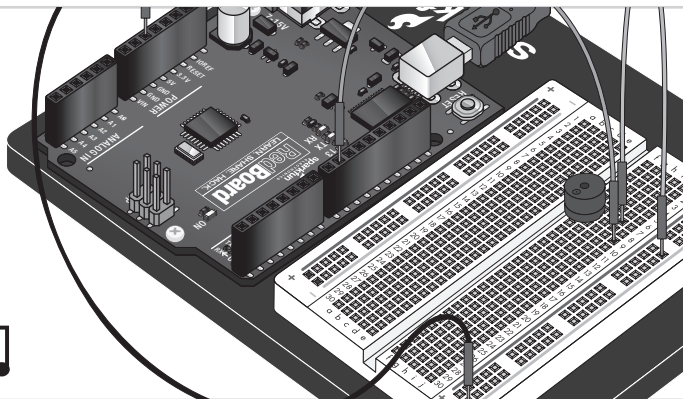
```
tone(pin, frequency, duration);
```



One of Arduino's many useful built-in commands is the `tone()` function. This function drives an output pin at a certain frequency, making it perfect for driving buzzers and speakers. If you give it a duration (in milliseconds), it will play the tone then stop. If you don't give it a duration, it will keep playing the tone forever (but you can stop it with another function, `noTone()`).

What You Should See:

You should see - well, nothing! But you should be able to hear a song. If it isn't working, make sure you have assembled the circuit correctly and verified and uploaded the code to your board or see the troubleshooting tips below.



Troubleshooting:

No Sound

Given the size and shape of the piezo buzzer it is easy to miss the right holes on the breadboard. Try double checking its placement.

Can't Think While the Melody is Playing

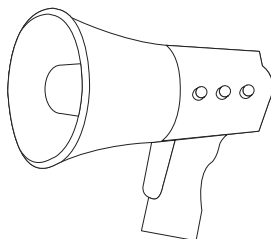
Just pull up the piezo buzzer whilst you think, upload your program then plug it back in.

Feeling Let Down and Deserted

The code is written so you can easily add your own songs.

Real World Application:

Many modern megaphones have settings that use a loud amplified buzzer. They are usually very loud and quite good at getting people's attention.

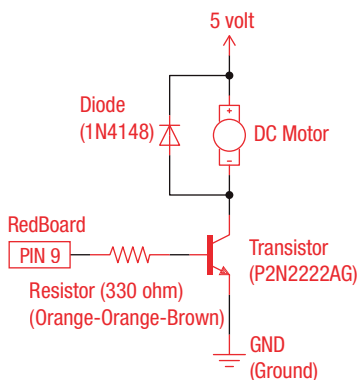


Spinning a Motor

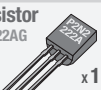
Remember before when you played around with a servo motor? Now we are going to tackle spinning a motor. This requires the use of a transistor, which can switch a larger amount of current than the RedBoard can. When using a transistor, you just need to make sure its maximum specs are high enough for your use. The transistor we are using for this circuit is rated at 40V max and 200 milliamps max – perfect for our toy motor! When the motor is spinning and suddenly turned off, the magnetic field inside it collapses, generating a voltage spike. This can damage the transistor. To prevent this, we use a "flyback diode", which diverts the voltage spike around the transistor.



When you're building the circuit be careful not to mix up the transistor and the temperature sensor, they're almost identical. Look for "P2N2222A" on the body of the transistor.



PARTS:

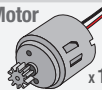
Transistor
P2N2222AG

x1

Diode
1N4148

x1

DC Motor



x1

Wire



x6

330Ω
Resistor

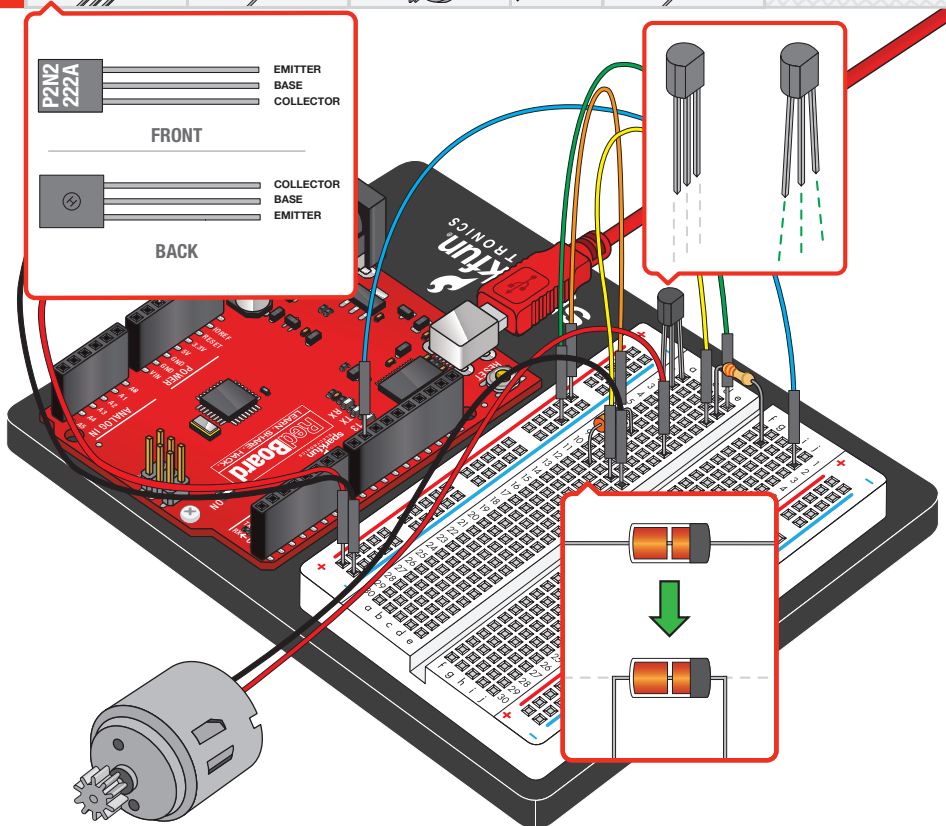
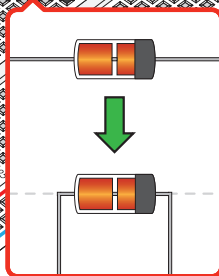
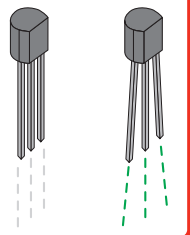
x1

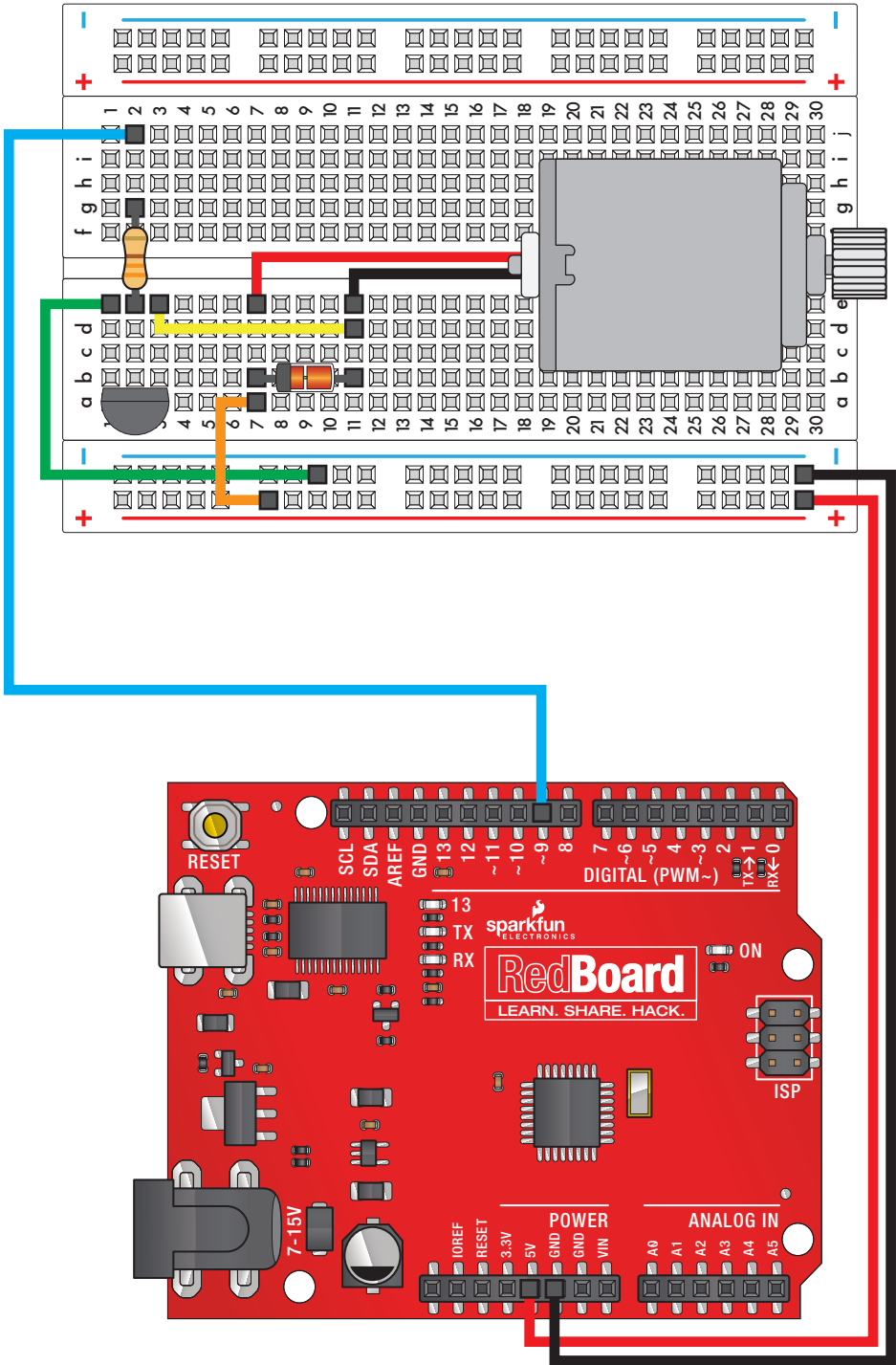
P2N2
222A








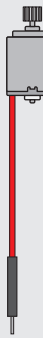








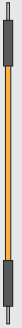








FRONT

EMITTER
BASE
COLLECTOR

BACK

COLLECTOR
BASE
EMITTER



Component:	Image Reference:		
Transistor P2N2222AG 			
Diode 1N4148 			
DC Motor			
330Ω Resistor			
Jumper Wire			
Jumper Wire			
Jumper Wire			
Jumper Wire			
Jumper Wire			
Jumper Wire			

Putting it all together:

At this point you're probably starting to get your own ideas for circuits that do fun things, or help solve a real problem. Excellent! Here are some tips on programming in general.

Most of the sketches you write will be a loop with some or all of these steps:

1. Perform some sort of input
2. Make some calculations or decisions
3. Perform some sort of output
4. Repeat! (Or not!)

We've already shown you how to use a bunch of different input sensors and output devices (and we still have a few more to go). Feel free to make use of the examples in your own sketches - this is the whole idea behind the "Open Source" movement.

It's usually pretty easy to pull pieces of different sketches together, just open them in two windows, and copy and paste between them. This is one of the reasons we've been promoting "good programming habits". Things like using constants for pin numbers, and breaking your sketch into functions, make it much easier to re-use your code in new sketches. For example, if you pull in two pieces of code that use the same pin, you can easily change one of the constants to a new pin. (Don't forget that not all of the pins support **analogWrite()**; the compatible pins are marked on your board.)

If you need help, there are internet forums where you can ask questions. Try Arduino's forum at arduino.cc/forum, and SparkFun's at forum.sparkfun.com. When you're ready to move to more advanced topics, take a look at Arduino's tutorials page at arduino.cc/en/tutorial. Many of SparkFun's more advanced products were programmed with Arduino. (allowing you to easily modify them), or have Arduino examples for them. See our product pages for info.

Finally, when you create something really cool, consider sharing it with the world so that others can learn from your genius. Be sure to let us know on https://www.sparkfun.com/project_calls so we can put it on our home page!



Open Arduino IDE // File > Examples > SIK Guide > **Circuit # 12**

Code to Note:



`while (Serial.available() > 0)`



The RedBoard's serial port can be used to receive as well as send data. Because data could arrive at any time, the RedBoard stores, or "buffers" data coming into the port until you're ready to use it. The `Serial.available()` command returns the number of characters that the port has received, but haven't been used by your sketch yet. Zero means no data has arrived.

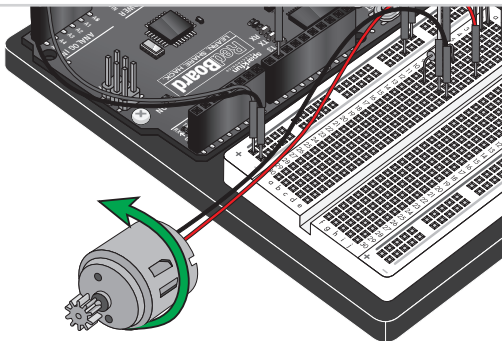
`speed = Serial.parseInt();`



If the port has data waiting for you, there are a number of ways for you to use it. Since we're typing numbers into the port, we can use the handy `Serial.parseInt()` command to extract, or "parse" integer numbers from the characters it's received. If you type "1" "0" "0" to the port, this function will return the number 100.

What You Should See:

The DC Motor should spin if you have assembled the circuit's components correctly, and also verified/uploaded the correct code. If your circuit is not working check the troubleshooting section below.



Troubleshooting:

Motor Not Spinning

If you sourced your own transistor, double check with the data sheet that the pinout is compatible with a P2N2222AG (many are reversed).

Still No Luck

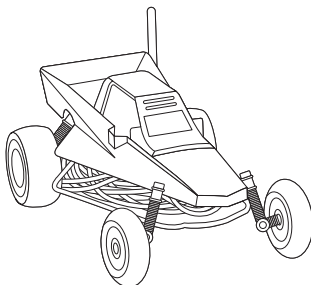
If you sourced your own motor, double check that it will work with 5 volts and that it does not draw too much power.

Still Not Working

Sometimes the RedBoard will disconnect from the computer. Try un-plugging and then re-plugging it into your USB port.

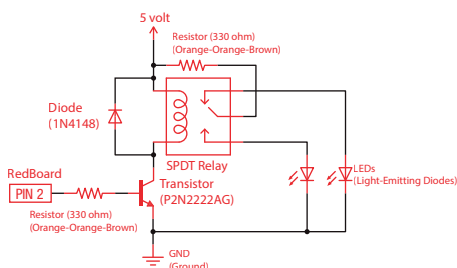
Real World Application:

Radio Controlled(RC) cars use Direct Current(DC) motors to turn the wheels for propulsion.



Relays

In this circuit, we are going to use some of the lessons we learned in circuit 12 to control a relay. A relay is basically an electrically controlled mechanical switch. Inside that harmless looking plastic box is an electromagnet that, when it gets a jolt of energy, causes a switch to trip. In this circuit, you'll learn how to control a relay like a pro – giving your RedBoard even more powerful abilities!



When the relay is off, the COM (common) pin will be connected to the NC (Normally Closed) pin. When the relay is on, the COM (common) pin will be connected to the NO (Normally Open) pin.

PARTS:

Relay



x1

Transistor
P2N2222AG

x1

Diode
1N4148

x1

330Ω
Resistor

x2

LED

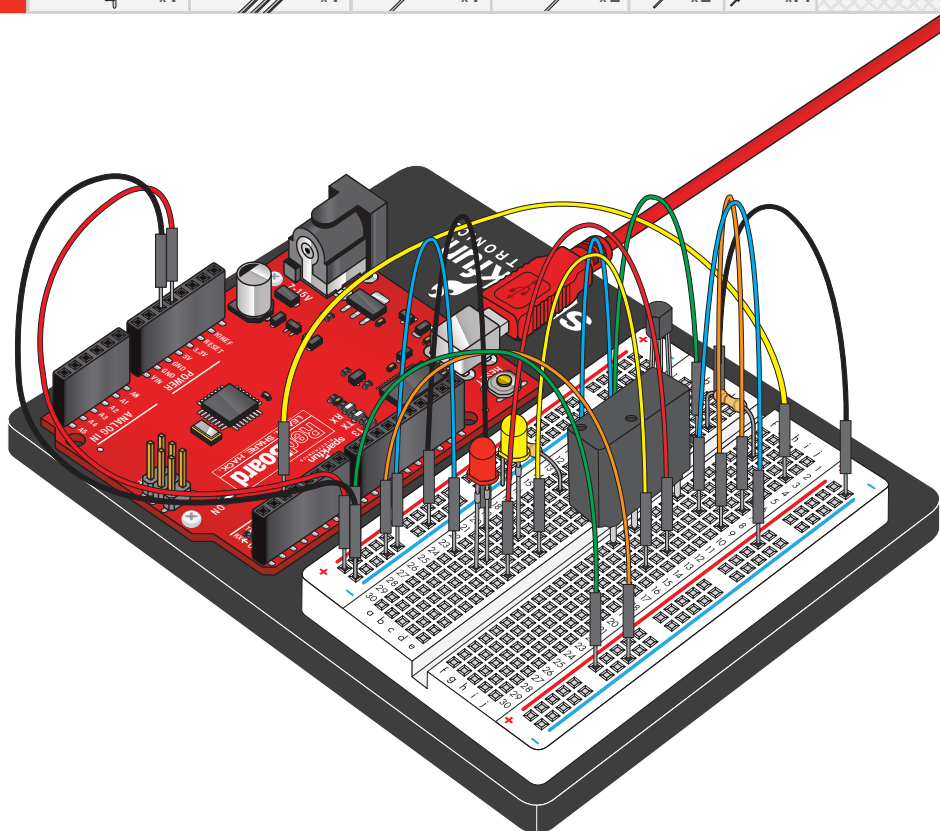


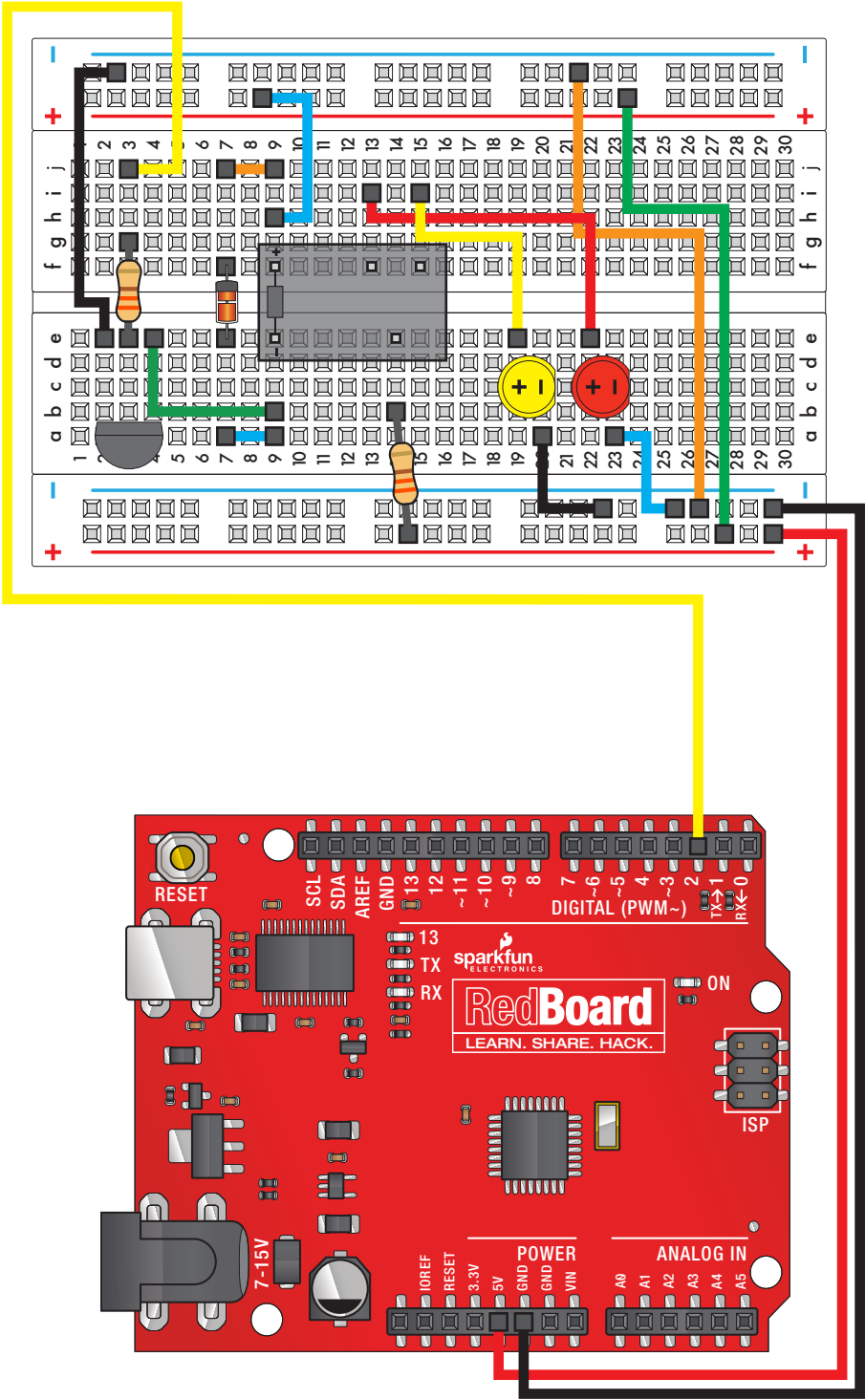
x2















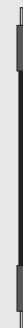






Wire



x14





Component:	Image Reference:			Image Reference:	Component:	
Relay					Jumper Wire	
Transistor P2N2222AG					Jumper Wire	
LED (5mm)					Jumper Wire	
LED (5mm)					Jumper Wire	
Diode 1N4148					Jumper Wire	
330Ω Resistor					Jumper Wire	
330Ω Resistor					Jumper Wire	
Jumper Wire					Jumper Wire	
Jumper Wire						
Jumper Wire						
Jumper Wire						
Jumper Wire						
Jumper Wire						



Open Arduino IDE // File > Examples > SIK Guide > **Circuit # 13**

Code to Note:



`digitalWrite(relayPin, HIGH);`



When we turn on the transistor, which in turn energizes the relay's coil, the relay's switch contacts are closed. This connects the relay's COM pin to the NO (Normally Open) pin. Whatever you've connected using these pins will turn on. (Here we're using LEDs, but this could be almost anything.)

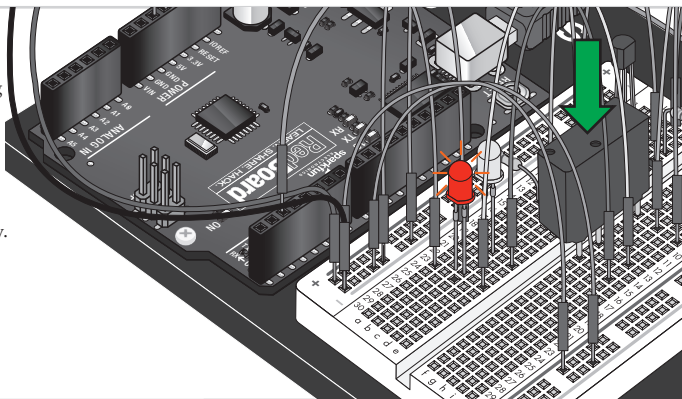
`digitalWrite(relayPin, LOW);`



The relay has an additional contact called NC (Normally Closed). The NC pin is connected to the COM pin when the relay is OFF. You can use either pin depending on whether something should be normally on or normally off. You can also use both pins to alternate power to two devices, much like railroad crossing warning lights.

What You Should See:

You should be able to hear the relay contacts click, and see the two LEDs alternate illuminating at 1-second intervals. If you don't, double-check that you have assembled the circuit correctly, and uploaded the correct sketch to the board. Also, see the troubleshooting tips below.



Troubleshooting:

LEDs Not Lighting

Double-check that you've plugged them in correctly. The longer lead (and non-flat edge of the plastic flange) is the positive lead.

No Clicking Sound

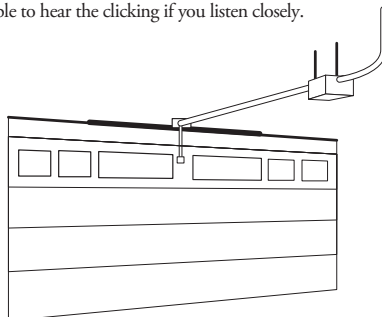
The transistor or coil portion of the circuit isn't quite working. Check the transistor is plugged in the right way.

Not Quite Working

The included relays are designed to be soldered rather than used in a breadboard. As such you may need to press it in to ensure it works (and it may pop out occasionally). When you're building the circuit be careful not to mix up the temperature sensor and the transistor, they're almost identical.

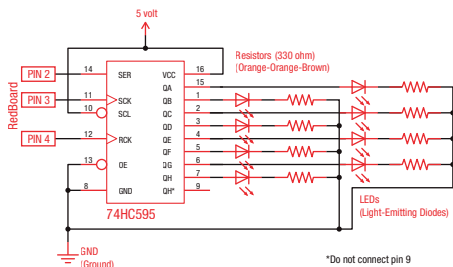
Real World Application:

Garage door openers use relays to operate. You might be able to hear the clicking if you listen closely.



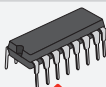
Shift Register

Now we are going to step into the world of ICs (integrated circuits). In this circuit, you'll learn all about using a shift register (also called a serial-to-parallel converter). The shift register will give your RedBoard an additional eight outputs, using only three pins on your board. For this circuit, you'll practice by using the shift register to control eight LEDs.



PARTS:

IC



x1

LED



x8

330 Ω
Resistor

x8

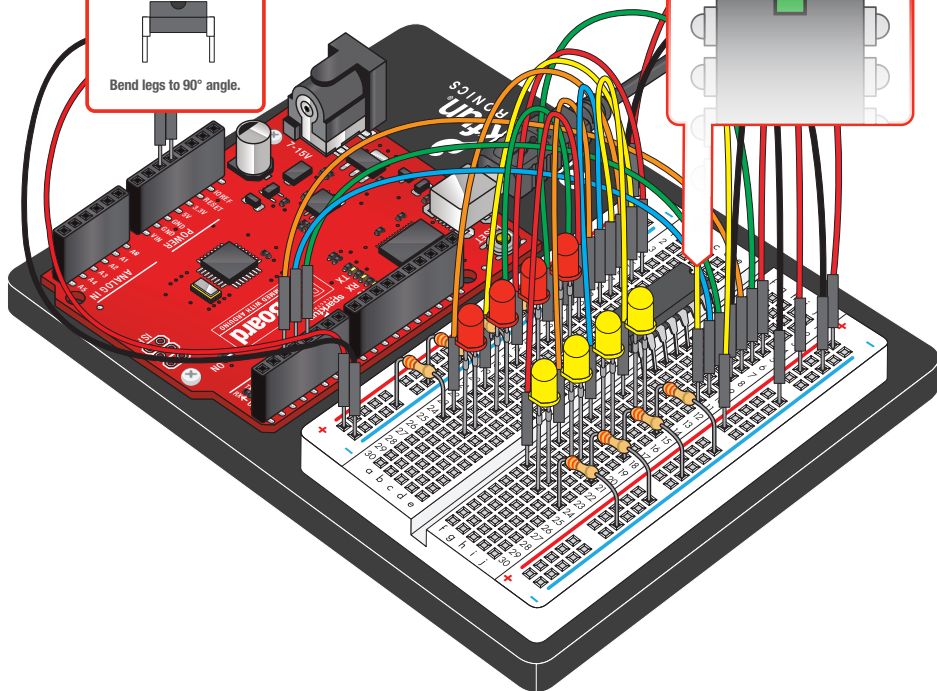
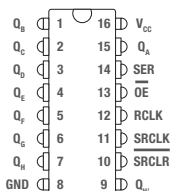
Wire

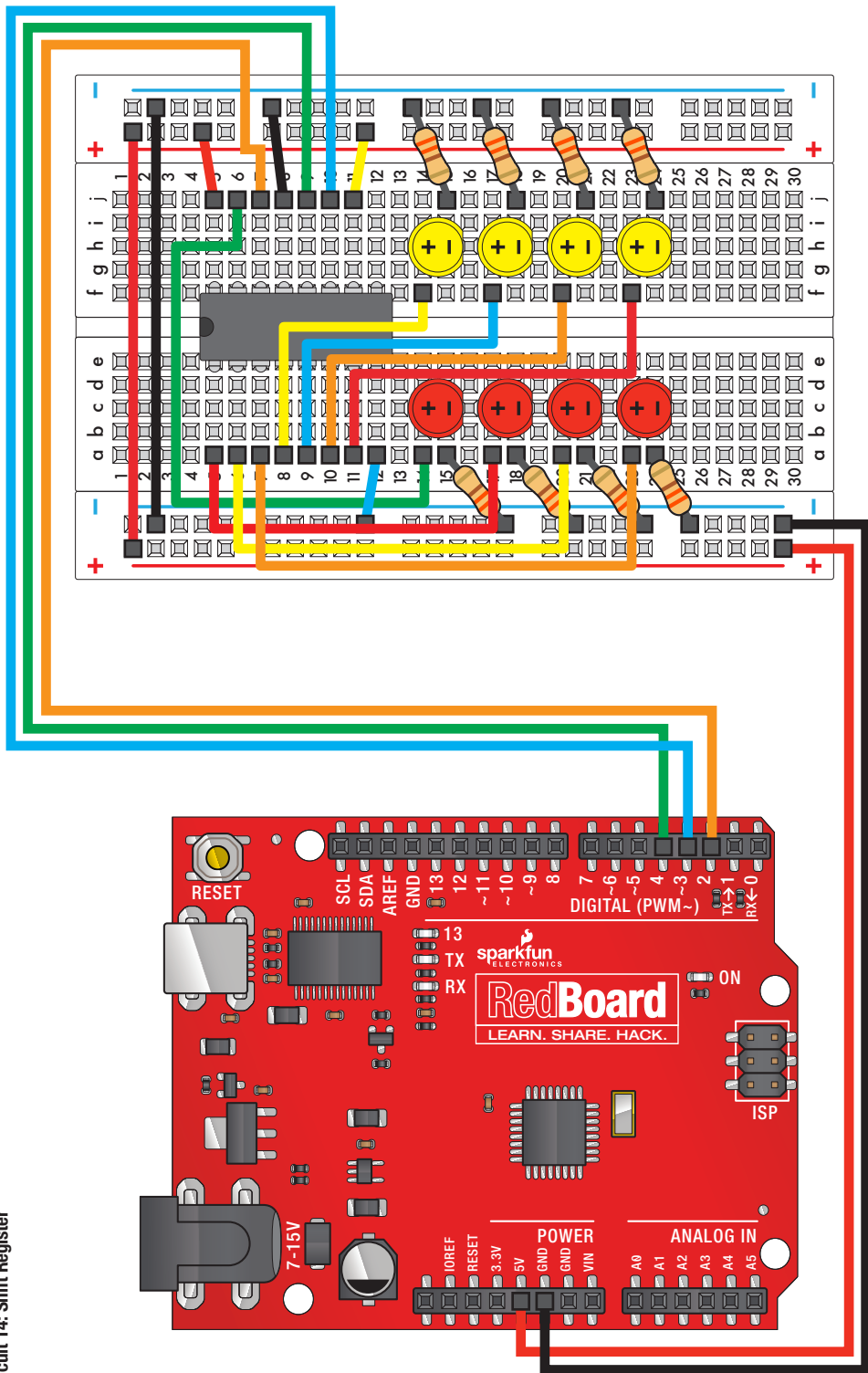













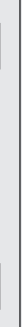

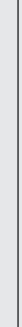



















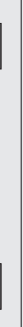

x19

Bend legs to 90° angle.

Align notch on top, inbetween "e5" and "f5" on the breadboard. The notch indicates where pin 1 is.





Component:	Image Reference:			Component:	Image Reference:	
IC		c5 e6 e7 e8 e9 a0 a1 a2 f5 f6 f7 f8 f9 f10 f11 f12				
LED (5mm)			c4 d5 +			j5 +
LED (5mm)			c7 d8 +			j6 a14
LED (5mm)			d0 d1 +			j7
LED (5mm)			d3 d4 +			j8 -
LED (5mm)			h4 h5 +			j9
LED (5mm)			h7 h8 +			j10
LED (5mm)			h0 h1 +			j11 +
LED (5mm)			h2 h3 h24 +			f14 a8
330Ω Resistor						f17 a9
330Ω Resistor						f20 a10
330Ω Resistor						f23 a11
330Ω Resistor						a23 a7
330Ω Resistor						a20 a6
330Ω Resistor						a17 a5
330Ω Resistor						5V +
330Ω Resistor						GND -
Jumper Wire						



Open Arduino IDE // File > Examples > SIK Guide > **Circuit # 14**

Code to Note:



```
shiftOut(datapin, clockpin, MSBFIRST, data);
```

You'll communicate with the shift register (and a lot of other parts) using an interface called SPI, or Serial Peripheral Interface. This interface uses a data line and a separate clock line that work together to move data in or out of the RedBoard at high speed. The MSBFIRST parameter specifies the order in which to send the individual bits, in this case we're sending the Most Significant Bit first.

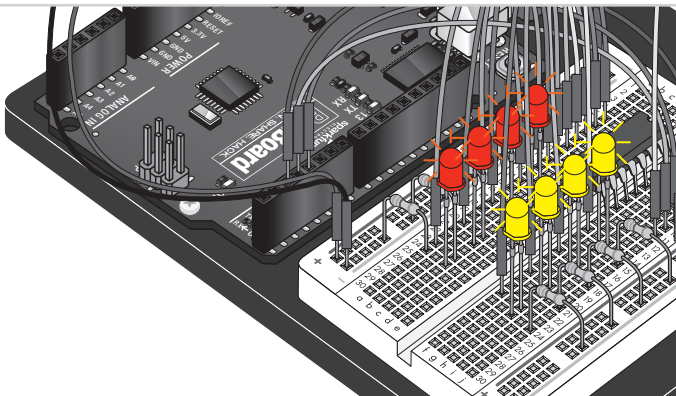


```
bitWrite(byteVar, desiredBit, desiredState);
```

Bits are the smallest possible piece of memory in a computer; each one can store either a "1" or a "0". Larger numbers are stored as arrays of bits. Sometimes we want to manipulate these bits directly, for example now when we're sending eight bits to the shift register and we want to make them 1 or 0 to turn the LEDs on or off. The RedBoard has several commands, such as bitWrite(), that make this easy to do.

What You Should See:

You should see the LEDs light up similarly to circuit 4 (but this time, you're using a shift register). If they aren't, make sure you have assembled the circuit correctly and verified and uploaded the code to your board. See the troubleshooting tips below.



Troubleshooting:

The RedBoard's power LED goes out

This happened to us a couple of times, it happens when the chip is inserted backward. If you fix it quickly nothing will break.

Not Quite Working

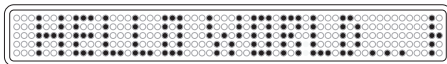
Sorry to sound like a broken record but it is probably something as simple as a crossed wire.

Frustration

Shoot us an e-mail, this circuit is both simple and complex at the same time. We want to hear about problems you have so we can address them in future editions: techsupport@sparkfun.com

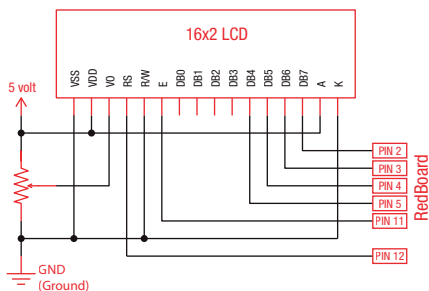
Real World Application:

Similar to circuit #4, a scrolling marquee display delivers a message with multiple LEDs. Essentially the same task the shift register achieves here in Circuit #14.



LCD

In this circuit, you'll learn about how to use an LCD. An LCD, or liquid crystal display, is a simple screen that can display commands, bits of information, or readings from your sensor - all depending on how you program your board. In this circuit, you'll learn the basics of incorporating an LCD into your project.



PARTS:

LCD



x1

Potentiometer

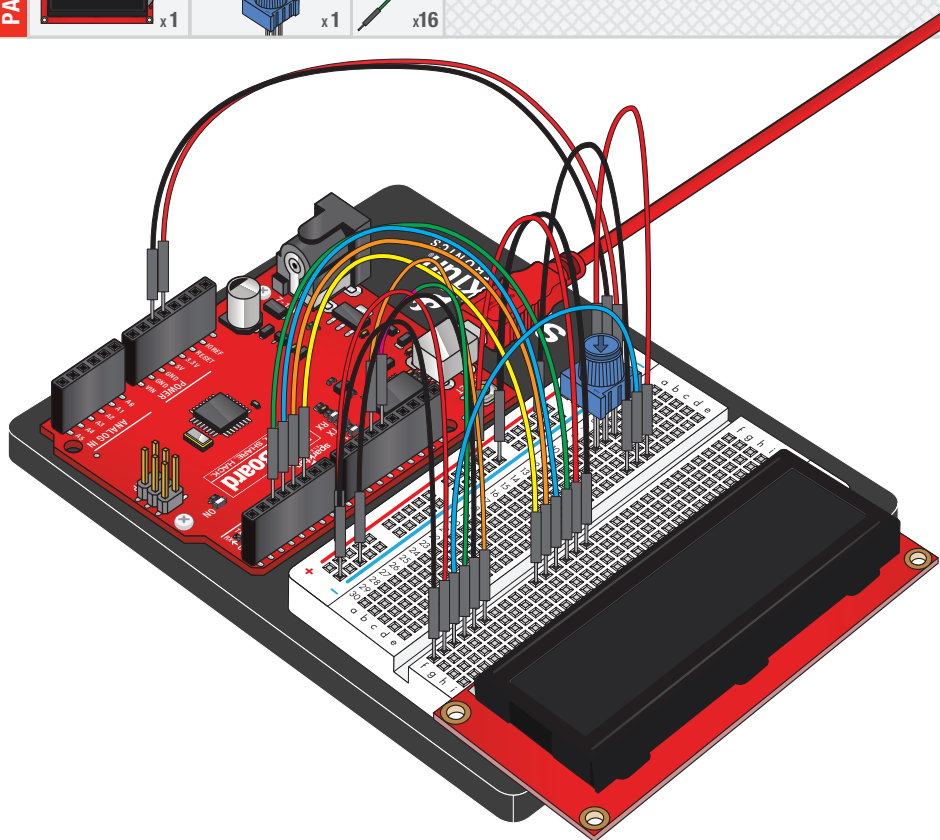


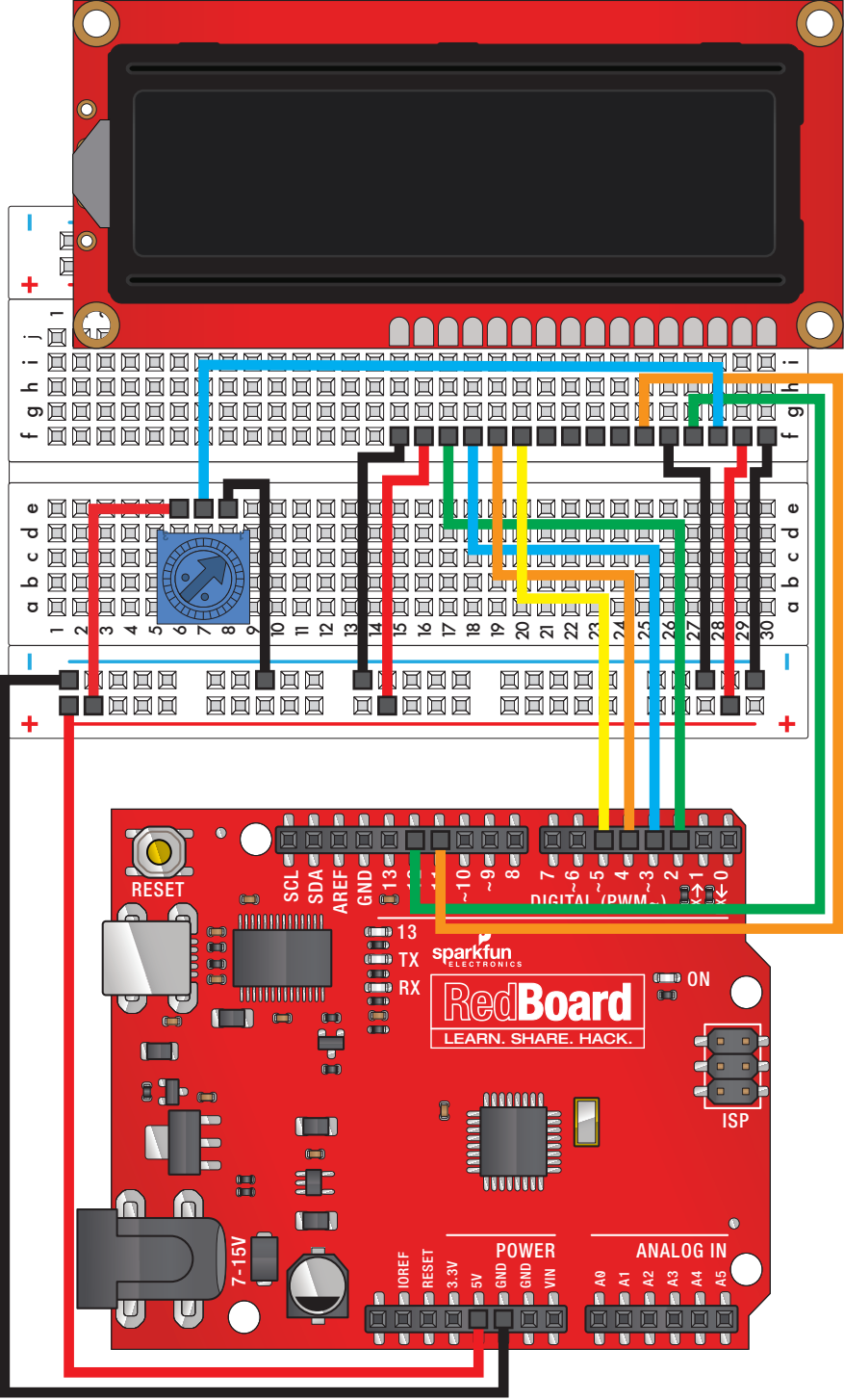
x1

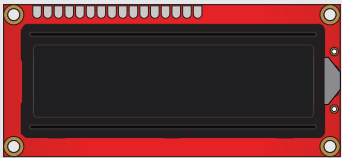












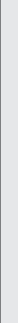



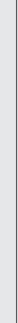






Wire



x16





Component:	Image Reference:			Component:	Image Reference:	
LCD			 <div> j30j29j28j27j26j25j24j23j22j21j20j19j18j17j16j15 </div>			
						<div>Pin 3</div> <div>Pin 4</div> <div>Pin 5</div> <div>Pin 11</div> <div>f26 -</div> <div>Pin 12</div> <div>f29 +</div> <div>f30 -</div>
Potentiometer			 <div> b8b7b6+ -e6e7f28e8f15-f16+f17 </div>			
Jumper Wire		<div>5V</div>	<div>+</div>	Jumper Wire		
Jumper Wire		<div>GND</div>	<div>-</div>	Jumper Wire		
Jumper Wire			<div>e6 +</div>	Jumper Wire		
Jumper Wire			<div>e7 f28</div>	Jumper Wire		
Jumper Wire			<div>e8 -</div>	Jumper Wire		
Jumper Wire			<div>f15 -</div>	Jumper Wire		
Jumper Wire			<div>f16 +</div>	Jumper Wire		
Jumper Wire		<div>Pin 2</div>	<div>f17</div>	Jumper Wire		



Open Arduino IDE // File > Examples > SIK Guide > Circuit # 15

Code to Note:



#include <LiquidCrystal.h>



This bit of code tells your Arduino IDE to include the library for a simple LCD display. Without it, none of the commands will work, so make sure you include it!

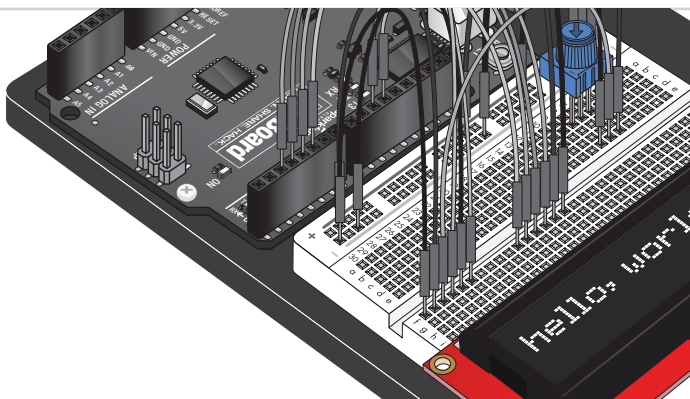
lcd.print("hello, world!");



This is the first time you'll fire something up on your screen. You may need to adjust the contrast to make it visible. Twist the potentiometer until you can clearly see the text!

What you Should See:

Initially, you should see the words "hello, world!" pop up on your LCD. Remember you can adjust the contrast using the potentiometer if you can't make out the words clearly. If you have any issues, make sure your code is correct and double-check your connections.



Troubleshooting:

The Screen is Blank or Completely Lit?

Fiddle with the contrast by twisting the potentiometer. If it's incorrectly adjusted, you won't be able to read the text.

Not Working At All?

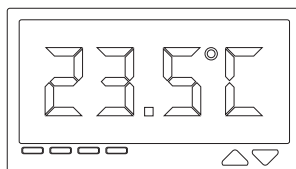
Double check the code, specifically that you include the LCD library.

Screen Is Flickering

Double check your connections to your breadboard and Arduino.

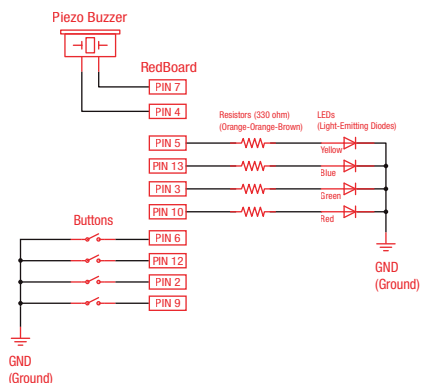
Real World Application:

LCDs are everywhere! From advanced LCDs like your television, to simple notification screens, this is a very common and useful display!



Simon Says

Now that we've learned all the basics behind the components in the SIK, let's put them together and have some fun. This circuit will show you how to create your own Simon Says game. Using some LEDs, some buttons, a buzzer and some resistors, you can create this and other exciting games with your SIK.



PARTS:

330Ω
Resistor

x 4

LED



x 4

Push Button



x 4

Piezo Element

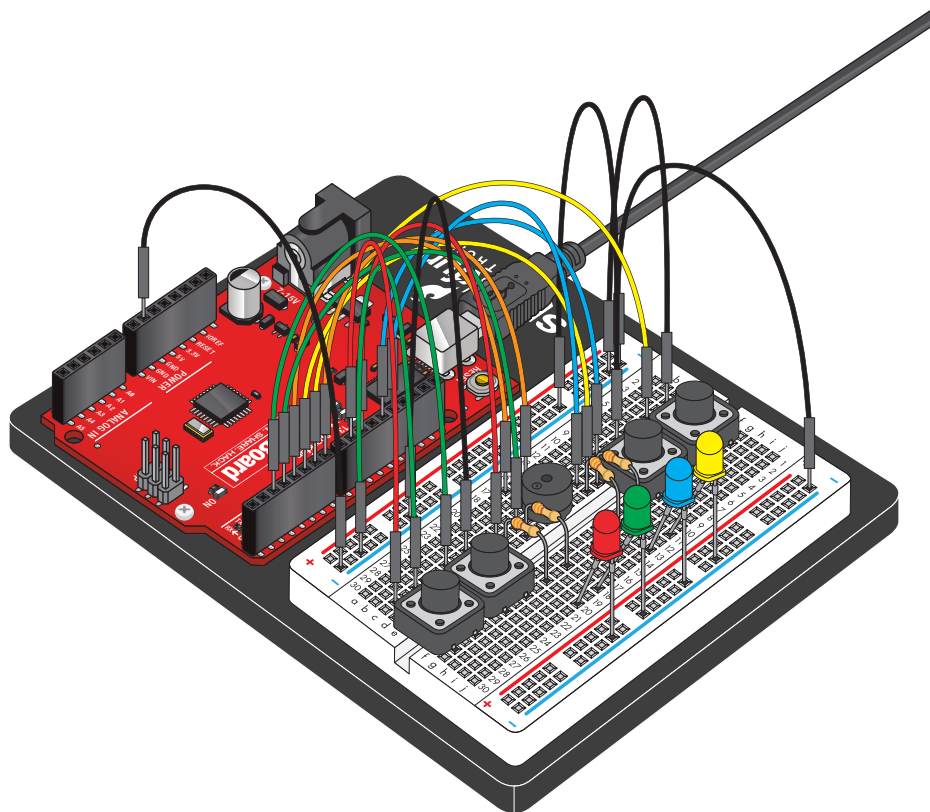


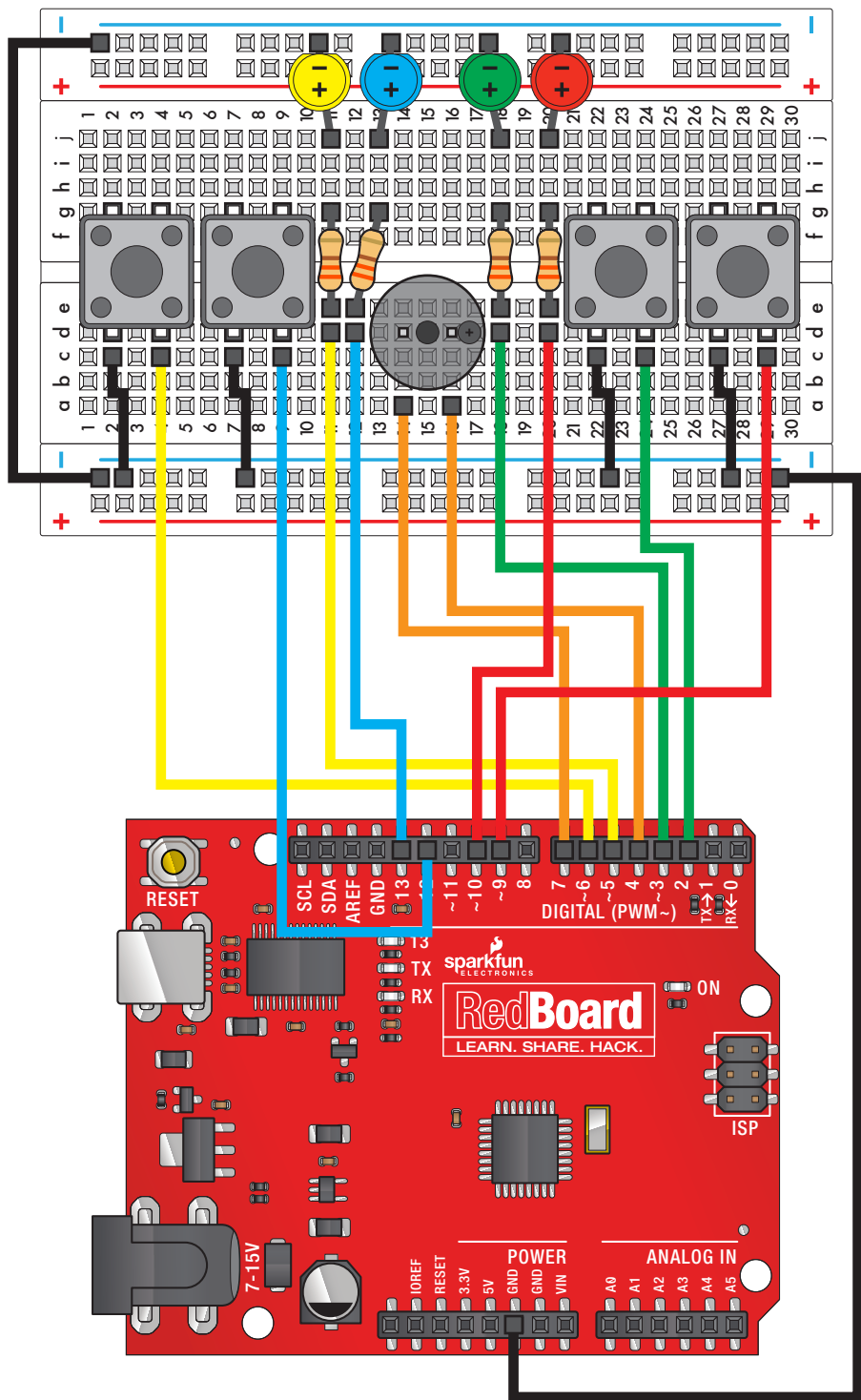
x 1

Wire



x 16





Component:	Image Reference:			Component:	Image Reference:	
330Ω Resistor				Jumper Wire		
330Ω Resistor				Jumper Wire		
330Ω Resistor				Jumper Wire		
330Ω Resistor				Jumper Wire		
LED (5mm)				Jumper Wire		
LED (5mm)				Jumper Wire		
LED (5mm)				Jumper Wire		
LED (5mm)				Jumper Wire		
Push Button				Jumper Wire		
Push Button				Jumper Wire		
Push Button				Jumper Wire		
Push Button				Jumper Wire		
Piezo Element				Jumper Wire		



Open Arduino IDE // File > Examples > SIK Guide > Circuit #16

Code to Note:



#define



The #define statement is used to create constants in your code. Constants are variables that will likely only have one value during the lifespan of your code. Thus, you can assign constants a value, and then use them throughout your code wherever you need them.

Then, if you need to change that value, you only have to change one line instead of going through all the code to find every instance of that variable.

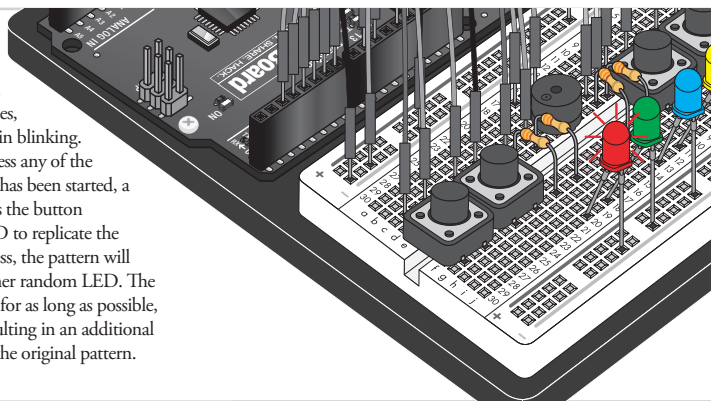
byte



Bytes are another variable type. In the world of computing, a byte is a chunk of space that contains 8 bits, and a bit is a single binary value. Binary is another way of counting and uses only 1's and 0's. So a byte can hold all 1's: 11111111, all 0's: 00000000, or a combination of the two: 10010110.

What You Should See:

With the circuit complete, plug the Arduino in to a power source. Once powered, the buzzer will beep a few times, and all four LEDs should begin blinking. The game begins once you press any of the four buttons. Once the game has been started, a random LED will blink. Press the button associated with that color LED to replicate the pattern. With a successful guess, the pattern will repeat, this time adding another random LED. The player is to follow the pattern for as long as possible, with each successful guess resulting in an additional layer of complexity added to the original pattern.



Troubleshooting:

Only half the circuit works

If only half of your circuit is working, make sure you added the additional wire from one ground rail to the other. Remember that breadboards have two power rails on each side and that these can be connected, or bussed, together to provide the power to both sides of the same circuit.

No sound

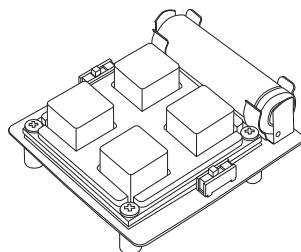
Once the buzzer is in the breadboard, it's hard to see the legs and which row they are connected to. If you aren't hearing any sound, make sure your wires are on the same row as the buzzer legs.

Game is not working

If everything starts up ok, but you're having trouble when it comes time to play the game, you may have a button or two misplaced. Pay close attention to which pin is connected to each button, as it matters which button is pressed when a particular color lights up.

Real World Application:

Toys and games, such as the original Simon from Milton Bradley, have relied on electronics to provide fun and entertainment to children across the world.





Visit us Online:

This is just the beginning of your exploration into embedded electronics and coding. Our website has a wealth of tutorials to whet your appetite for more knowledge. We also host a community of hackers, engineers, DIYers, etc. in our forums. So log on to our website for more information about Arduino, or to plan ahead for your next project!

www.sparkfun.com

NOTES:

Begin your Journey into Electronics

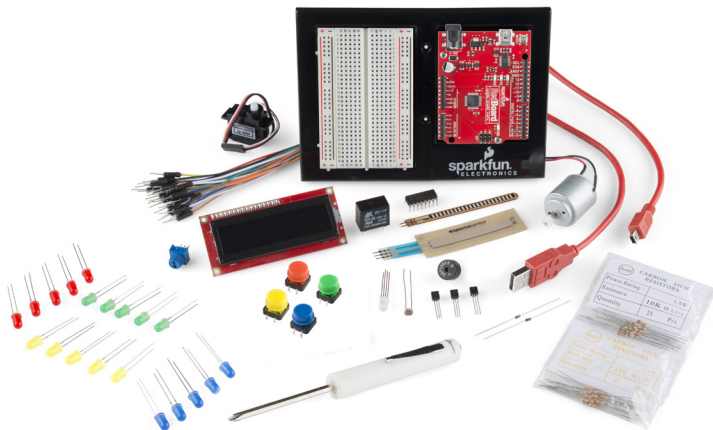
This kit will guide you through experiments of varying difficulty as you learn all about embedded systems, physical computing, programming and more! This kit is perfect for anyone who wants to explore the power of the RedBoard platform.



The SparkFun Inventor's Kit teaches basic programming, for which you will need both a computer and an internet connection.

You will also learn to assemble 16 basic physical electronic circuits, but **no soldering is required**. No previous experience is necessary!

KIT INCLUDES



SparkFun RedBoard
Breadboard
Instruction booklet
Sealed relay
Small servo
LEDs

RGB LED
Temperature sensor
DC motor
8-bit shift register
Push button switches
Potentiometer

Photo Resistor
Transistors
Jumper wires
USB cable
Signal diodes
10k ohm resistors

330 ohm resistors
Piezo buzzer
Flex sensor
Soft potentiometer
Baseplate
LCD

© SparkFun Electronics, Inc. All rights reserved. The SparkFun Inventor's Kit for the SparkFun RedBoard features, specifications, system requirements and availability are subject to change without notice. All other trademarks contained herein are the property of their respective owners.

The SIK Guide for the SparkFun Inventor's Kit for the SparkFun RedBoard is licensed under the Creative Commons Attribution Share-Alike 3.0 Unported License

To view a copy of this license visit: <http://creativecommons.org/by-sa/3.0/>

Or write: Creative Commons, 171 Second Street, Suite 300, San Francisco, CA 94105, USA.