SIK Experiment Guide for Arduino - V3.2

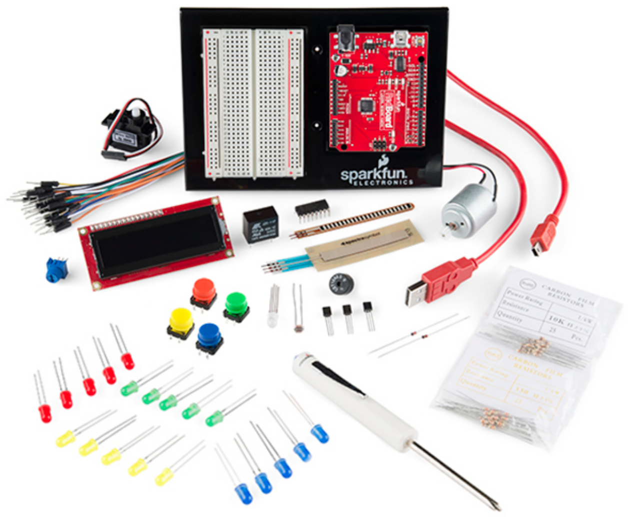
CONTRIBUTORS: *HELLO_TECHIE*

Introduction: SIK RedBoard & Sparkfun Mini Inventor's Kit

The SparkFun Inventor's Guide is your map for navigating the waters of beginning embedded electronics. This guide contains all the information you will need to explore the 16 circuits of the SparkFun Inventor's Kit for Arduino V3.2. At the center of this guide is one core philosophy - that anyone can (and should) play around with electronics. When you're done with this guide, you'll have the know-how to start creating your own projects and experiments. Now enough talking - let's get inventing!

For Starter Kit for RedBoard - Programmed with Arduino users: For those who have Starter Kit for RedBoard - Programmed with Arduino, you are able to follow through experiments 1, 2, 3, 6, 7, 9, 10, and 11.

SparkFun Inventor's Kit - V3.2



Here is all the parts in the SparkFun Inventor's Kit for Arduino:

- **SparkFun RedBoard - Programmed with Arduino** - The SparkFun RedBoard, fully assembled and tested
- **Arduino and Breadboard Holder** - A nice holder for your RedBoard and breadboard
- **SparkFun Inventor's Kit Guidebook** - A printed manual that you follow along through all the experiments
- **Breadboard** - Excellent for making circuits and connections off the Arduino.
- **Carrying Case** - Take your kit anywhere with ease
- **SparkFun Mini Screwdriver** - To help you screw on your RedBoard to the holder
- **16x2 White on Black LCD (with headers)** - This is a basic 16 character by 2 line display with a

snazzy black background with white characters.

- **74HC595 Shift Register**- Simple shift register IC. Clock in data and latch it to free up IO pins on your RedBoard.
- **2N2222 Transistors** - This little transistor can help in your project by being used to help drive large loads or amplifying or switching applications.
- **1N4148 Diodes** - This is a very common signal diode - 1N4148. Use this for signals up to 200mA of current.
- **DC Motor with Gear** - It works well for basic things like making a fan or spinning something pretty fast without much resistance.
- **Small Servo** - Here is a simple, low-cost, high quality servo for all your mechatronic needs.
- **SPDT 5V Relay** - This is a high quality Single Pole - Double Throw (SPDT) sealed relay. Use it to switch high voltage, and/or high current devices. This relay's coil is rated up to 12V, with a minimum switching voltage of 5V.
- **TMP36 Temp Sensor** - A sensor for detecting temperature changes.
- **Flex sensor** - As the sensor is flexed, the resistance across the sensor increases.
- **Softpot** - By pressing down on various parts of the strip, the resistance linearly changes from 100Ohms to 10,000Ohms allowing you to very accurately calculate the relative position on the strip.
- **SparkFun USB Mini-B Cable** - This 6' cable provides you with a USB-A connector at the host end and mini-B connector at the device end.
- **Male to Male jumper wires** - These are high quality wires that allow you to connect the female headers on the Arduino to the components and breadboard.
- **Photocell** - A sensor to detect ambient light. Perfect for detecting when a drawer is opened or when night-time approaches.
- **Tri-Color LED** - Because everyone loves a blinky.
- **Red, Blue, Yellow, and Green LEDs** - Light emitting diodes make great general indicators.
- **Red, Blue, Yellow, and Green Tactile Buttons** - Go crazy with different colored buttons
- **10K Trimpot** - Also known as a variable resistor, this is a device commonly used to control volume, contrast, and makes a great general user control input.
- **Piezo Buzzer** - Use this to make sounds and play songs
- **330 Ohm Resistors** - Great current limiting resistors for LEDs, and strong pull-up resistors.
- **10k Ohm Resistors** - These make excellent pull-ups, pull-downs, and current limiters.

Sparkfun Mini Inventor's Kit

If you're new to electronics and programming, the Sparkfun Mini Inventor's Kit is a great way for beginners to get their foot in the door. Sparkfun Mini Inventor's Kit can be taken straight out of the box to help you make a slew of basic circuits.

Here is all the parts in the Sparkfun Mini Inventor's Kit:

Please note: This product is in the works, however you are able to follow along if you already have the Starter Kit for RedBoard for a lot of the experiments.

- **SparkFun RedBoard** - The SparkFun RedBoard, fully assembled and tested.
- **SparkFun USB Mini-B Cable** - This 6' cable provides you with a USB-A connector at the host end and mini-B connector at the device end.
- **Breadboard** - Excellent for making circuits and connections off the Arduino.
- **Male to Male jumper wires** - These are high quality wires that allow you to connect the female

headers on the Arduino to the components and breadboard.

- **Photocell** - A sensor to detect ambient light. Perfect for detecting when a drawer is opened or when night-time approaches.
- **TMP36 Temp Sensor** - A sensor for detecting temperature changes.
- **Tri-Color LED** - Because everyone loves a blinky. Use this LED to PWM mix any color you need.
- **Basic LEDs** - Light emitting diodes make great general indicators.
- 10K Trimpot - Also known as a variable resistor, this is a device commonly used to control volume, contrast, and makes a great general user control input.
- **12mm button** - Because big buttons are easier to hit.
- **Small Servo** - Here is a simple, low-cost, high quality servo for all your mechatronic needs.
- **330 Ohm Resistors** - Great current limiting resistors for LEDs, and strong pull-up resistors.
- **10k Ohm Resistors** - These make excellent pull-ups, pull-downs, and current limiters.

Experiment List

Please note: Experiments marked with **double asterisks** are for both the Sparkfun Mini Inventor’s Kit **(which is a future release)** and the RedBoard SIK V3.2. Experiments without asterisks will only be available to those who purchased the RedBoard SIK V3.2.

In this SparkFun Inventor’s Kit for Arduino guide, you will learn the following:

- **Experiment 1: Blinking an LED**
- **Experiment 2: Reading a Potentiometer**
- **Experiment 3: Driving and RGB LED**
- **Experiment 4: Driving Multiple LEDs**
- **Experiment 5: Push Buttons**
- **Experiment 6: Reading a Photoresistor**
- **Experiment 7: Reading a Temperature Sensor**
- **Experiment 8: Driving a Servo Motor**
- Experiment 9: Using a Flex Sensor
- Experiment 10: Reading a Soft Potentiometer
- Experiment 11: Using a Piezo Buzzer
- Experiment 12: Driving a Motor
- Experiment 13: Using Relays
- Experiment 14: Using a Shift Register
- Experiment 15: Using an LCD
- Experiment 16: Simon Says

What is the RedBoard platform?

The DIY Revolution

We live in a unique time where we have access to resources that allow us to create our own solutions and inventions. The DIY revolution is composed of hobbyists, tinkerers and inventors who would rather craft their own projects than let someone do it for them.

A Computer for the Physical World

The RedBoard in your hand (or on your desk) is your development platform. At its roots, the RedBoard is essentially a small portable computer. It is capable of taking inputs (such as the push of a button or a reading from a light sensor) and interpreting that information to control various outputs (like a blinking LED light or an electric motor).

That’s where the term “physical computing” is born - this board is capable of taking the world of electronics and relating it to the physical world in a real and tangible way. Trust us - this will all make more sense soon.

The SparkFun RedBoard is one of a multitude of development boards based on the ATmega328. It has 14 digital input/output pins (6 of which can be PWM outputs), 6 analog inputs, a 16 MHz crystal oscillator, a USB connection, a power jack, an ISP header, and a reset button. Check out our RedBoard Hookup Guide, to get yourself familiar with the RedBoard.

Download the Arduino IDE

In order to get your RedBoard up and running, you’ll need to download the newest version of the Arduino software first from www.arduino.cc (it’s free and open source!). This software, known as the Arduino IDE, will allow you to program the board to do exactly what you want. It’s like a word processor for writing programs. With an internet-capable computer, open up your favorite browser and go to Arduino download page.

Check out our Installing Arduino IDE tutorial, to see in detail on how to install the Arduino IDE on your computer.

Connect your RedBoard to your Computer

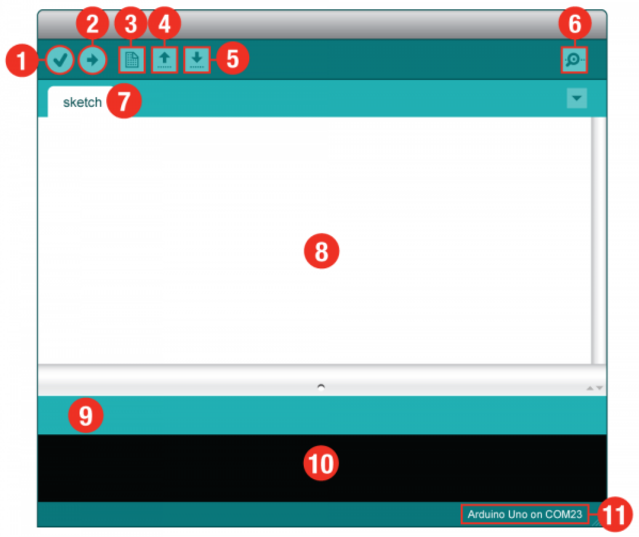
Use the USB cable provided in the SIK kit to connect the RedBoard to one of your computer’s USB inputs.

Install FTDI Drivers

Depending on your computer’s operating system, you will need to follow specific instructions. Please go to How to Install FTDI Drivers, for specific instructions on how to install the FTDI drivers onto your RedBoard.

Getting Started in the Arduino IDE

Now, it’s finally time to **open up the Arduino software**. You’ll be presented with a window that looks a little something like this:

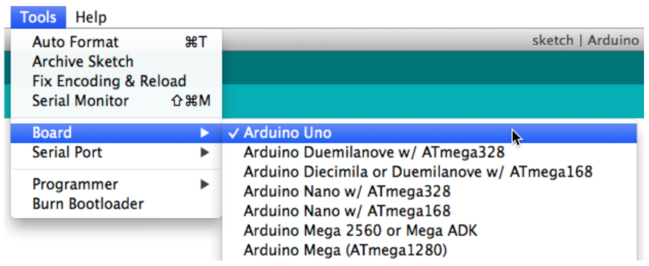


1. **Verify:** Compiles and approves your code. It will catch errors in syntax (like missing semi-colons or parenthesis).
2. **Upload:** Sends your code to the RedBoard. When you click it, you should see the lights on your board blink rapidly.
3. **New:** This buttons opens up a new code window tab.
4. **Open:** This button will let you open up an existing sketch.
5. **Save:** This saves the currently active sketch.
6. **Serial Monitor:** This will open a window that displays any serial information your RedBoard is transmitting. It is very useful for debugging.
7. **Sketch Name:** This shows the name of the sketch you are currently working on.
8. **Code Area:** This is the area where you compose the code for your sketch.
9. **Message Area:** This is where the IDE tells you if there were any errors in your code.
10. **Text Console:** The text console shows complete error messages. When debugging, the text console is very useful.
11. **Board and Serial Port:** Shows you what board and the serial port selections

Select your board: Arduino Uno

Before we can start jumping into the experiments, there are a couple adjustments we need to make. This step is required to tell the Arduino IDE *which* of the many Arduino boards we have. Go up to the **Tools** menu. Then hover over **Board** and make sure **Arduino Uno** is selected.

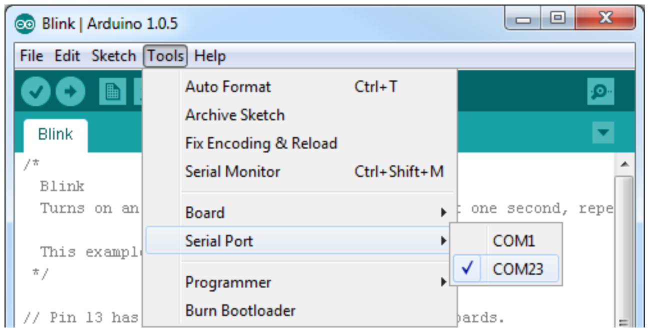
Please note: Your SparkFun RedBoard and the Arduino UNO are interchangeable but you won't find the RedBoard listed in the Arduino Software. Select "Arduino Uno" instead.


Tools
Help
sketch | Arduino
Auto Format ⌘T
Archive Sketch
Fix Encoding & Reload
Serial Monitor ⇧⌘M
Board
Serial Port
Programmer
Burn Bootloader
✓ Arduino Uno
Arduino Duemilanove w/ ATmega328
Arduino Diecimila or Duemilanove w/ ATmega168
Arduino Nano w/ ATmega328
Arduino Nano w/ ATmega168
Arduino Mega 2560 or Mega ADK
Arduino Mega (ATmega1280)

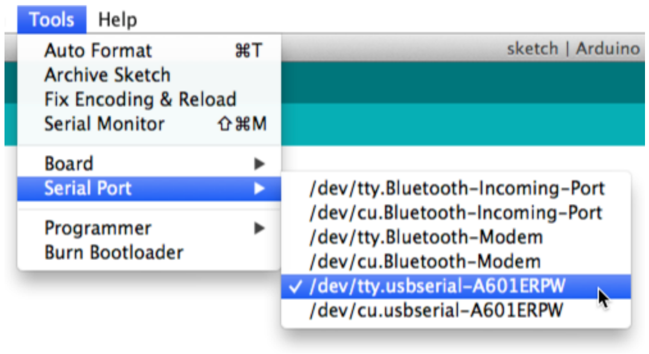
Select a Serial Port

Next up we need to tell the Arduino IDE which of our computer's serial ports the RedBoard is connected to. For this, again go up to **Tools**, then hover over **Serial Port** and select your RedBoard's serial port.

Window Users: This is likely to be com3 or higher (COM1 and COM2 are usually reserved for hardware serial ports). To find out, you can disconnect your RedBoard and re-open the menu; the entry that disappears should be the RedBoard. Reconnect the board and select that serial port.



Mac Users: Select the serial device of the RedBoard from the **Tools**, then hover over **Serial Port**. On the Mac, this should be something with /dev/tty.usbmodem or /dev/tty.usbserial in it.



Linux Users: Please visit the Arduino Learning Linux section, to learn more about Arduino on Linux.

Download Arduino Code

You are so close to to being done with setup! Download the SIK Guide Code. **Click** the following link to download the code:

SIK Guide Code

Copy "SIK Guide Code" into "examples" folder in the Arduino folder.

Window Users: Unzip the file "SIK Guide Code". It should be located in your browser's "Downloads" folder. Right click the zipped folder and choose "unzip". Copy the "SIK Guide Code" folder into Arduino's folder named "examples".

Mac Users: Unzip the file "SIK Guide Code". It should be located in your browser's "Downloads" folder. Right click the zipped folder and choose "unzip". Find "Arduino" in your applications folder. Right click (ctrl + click) on "Arduino". Select "Show Package Contents". Then, click through folders **Contents > Resources > Java > examples**. Copy the "SIK Guide Code" folder into Arduino's folder named "examples".

Suggested Reading

Before continuing on with this tutorial, we recommend you be somewhat familiar with the concepts in the following tutorial:

- How to Use a Breadboard - First time working with a breadboard? Please check out this tutorial! It will help you understand why the breadboard is a great for prototyping and how to use one.

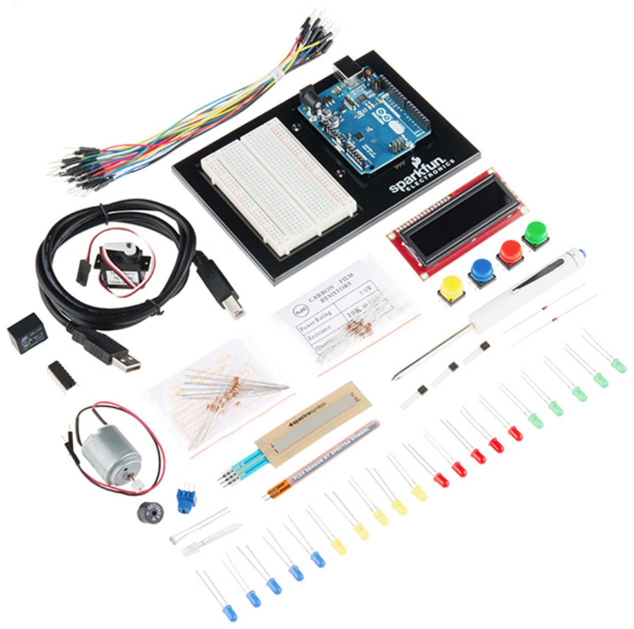
Time for the fun part!

Continue on to the first experiment and in no time you will be blinking an LED!

Introduction: SIK Arduino Uno

The SparkFun Inventor's Guide is your map for navigating the waters of beginning embedded electronics. This guide contains all the information you will need to explore the 16 circuits of the SparkFun Inventor's Kit for Arduino. At the center of this guide is one core philosophy - that anyone can (and should) play around with electronics. When you're done with this guide, you'll have the know-how to start creating your own projects and experiments. Now enough talking - let's get inventing!

SparkFun Inventor's Kit with Arduino Uno R3



Here is all the parts in the SparkFun Inventor's Kit for Arduino:

- **Arduino Uno R3** - The Arduino Uno is one of the more popular boards in the Arduino family and a great choice for beginners.
- **Arduino and Breadboard Holder** - A nice holder for your RedBoard and breadboard
- **Breadboard** - Excellent for making circuits and connections off the Arduino.
- **Carrying Case** - Take your kit anywhere with ease
- **SparkFun Mini Screwdriver** - To help you screw on your RedBoard to the holder
- **16x2 White on Black LCD (with headers)** - This is a basic 16 character by 2 line display with a snazzy black background with white characters.
- **74HC595 Shift Register**- Simple shift register IC. Clock in data and latch it to free up IO pins on your RedBoard.
- **2N2222 Transistors** - This little transistor can help in your project by being used to help drive large

loads or amplifying or switching applications.

- **1N4148 Diodes** - This is a very common signal diode - 1N4148. Use this for signals up to 200mA of current.
- **DC Motor with Gear** - It works well for basic things like making a fan or spinning something pretty fast without much resistance.
- **Small Servo** - Here is a simple, low-cost, high quality servo for all your mechatronic needs.
- **SPDT 5V Relay** - This is a high quality Single Pole - Double Throw (SPDT) sealed relay. Use it to switch high voltage, and/or high current devices. This relay's coil is rated up to 12V, with a minimum switching voltage of 5V.
- **TMP36 Temp Sensor** - A sensor for detecting temperature changes.
- **Flex sensor** - As the sensor is flexed, the resistance across the sensor increases
- **Softpot** - By pressing down on various parts of the strip, the resistance linearly changes from 100Ohms to 10,000Ohms allowing you to very accurately calculate the relative position on the strip.
- **SparkFun USB Mini-B Cable** - This 6' cable provides you with a USB-A connector at the host end and mini-B connector at the device end.
- **Male to Male jumper wires** - These are high quality wires that allow you to connect the female headers on the Arduino to the components and breadboard.
- **Photocell** - A sensor to detect ambient light. Perfect for detecting when a drawer is opened or when night-time approaches.
- **Tri-Color LED** - Because everyone loves a blinky.
- **Red, Blue, Yellow, and Green LEDs** - Light emitting diodes make great general indicators.
- **Red, Blue, Yellow, and Green Tactile Buttons** - Go crazy with different colored buttons
- **10K Trimpot** - Also known as a variable resistor, this is a device commonly used to control volume, contrast, and makes a great general user control input.
- **Piezo Buzzer** - Use this to make sounds and songs
- **330 Ohm Resistors** - Great current limiting resistors for LEDs, and strong pull-up resistors.
- **10k Ohm Resistors** - These make excellent pull-ups, pull-downs, and current limiters.

Experiment List

In this SparkFun Inventor's Kit for Arduino guide, you will learn the following:

Please note: Got the Sparkfun Mini Inventor's Kit? The experiments with the double asterisks you will want to follow along too.

- **Experiment 1: Blinking an LED**
- **Experiment 2: Reading a Potentiometer**
- **Experiment 3: Driving and RGB LED**
- **Experiment 4: Driving Multiple LEDs**
- **Experiment 5: Push Buttons**
- **Experiment 6: Reading a Photoresistor**
- **Experiment 7: Reading a Temperature Sensor**
- **Experiment 8: Driving a Servo Motor**
- Experiment 9: Using a Flex Sensor
- Experiment 10: Reading a Soft Potentiometer
- Experiment 11: Using a Piezo Buzzer
- Experiment 12: Driving a Motor

What is the Arduino platform?

The DIY Revolution

We live in a unique time where we have access to resources that allow us to create our own solutions and inventions. The DIY revolution is composed of hobbyists, tinkerers and inventors who would rather craft their own projects than let someone do it for them.

A Computer for the Physical World

The Arduino in your hand (or on your desk) is your development platform. It is capable of taking inputs (such as the push of a button or a reading from a light sensor) and interpreting that information to control various outputs (like a blinking LED light or an electric motor).

That's where the term "physical computing" is born - this board is capable of taking the world of electronics and relating it to the physical world in a real and tangible way. Trust us - this will all make more sense soon.

The Uno is a great choice for your first Arduino. It's got everything you need to get started, and nothing you don't. It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a USB connection, a power jack, a reset button and more. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started. Check out our What is an Arduino? tutorial, to learn more about Arduino.

Connect your Arduino Uno R3 to your Computer

Use the USB cable provided in the SIK kit to connect the Arduino Uno to one of your computer's USB inputs.

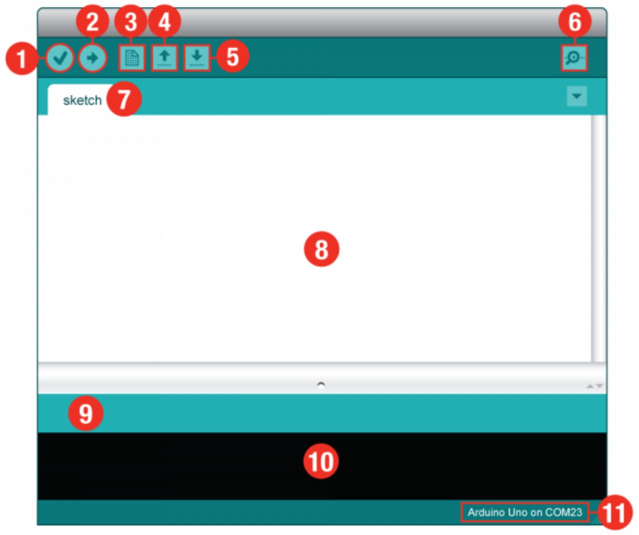
Download the Arduino IDE

In order to get your Arduino Uno R3 up and running, you'll need to download the newest version of the Arduino software first from www.arduino.cc (it's free and open source!). This software, known as the Arduino IDE, will allow you to program the board to do exactly what you want. It's like a word processor for writing programs. With an internet-capable computer, open up your favorite browser and go to Arduino download page.

Check out our Installing Arduino IDE tutorial, to see in detail on how to install the Arduino IDE on your computer.

Getting Started in the Arduino IDE

Now, it's finally time to **open up the Arduino software**. You'll be presented with a window that looks a little something like this:

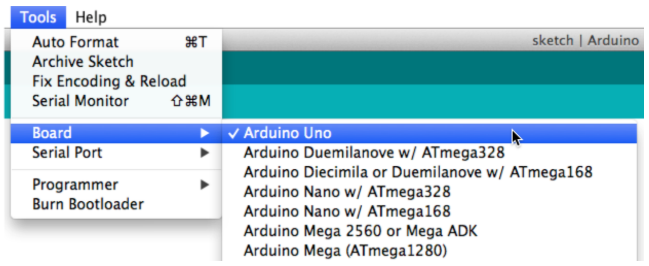


1. **Verify:** Compiles and approves your code. It will catch errors in syntax (like missing semi-colons or parenthesis).
2. **Upload:** Sends your code to the RedBoard. When you click it, you should see the lights on your board blink rapidly.
3. **New:** This buttons opens up a new code window tab.
4. **Open:** This button will let you open up an existing sketch.
5. **Save:** This saves the currently active sketch.
6. **Serial Monitor:** This will open a window that displays any serial information your RedBoard is transmitting. It is very useful for debugging.
7. **Sketch Name:** This shows the name of the sketch you are currently working on.
8. **Code Area:** This is the area where you compose the code for your sketch.
9. **Message Area:** This is where the IDE tells you if there were any errors in your code.
10. **Text Console:** The text console shows complete error messages. When debugging, the text console is very useful.
11. **Board and Serial Port:** Shows you what board and the serial port selections

Before we can start jumping into the experiments, there are a couple adjustments we need to make.

Select your board: Arduino Uno

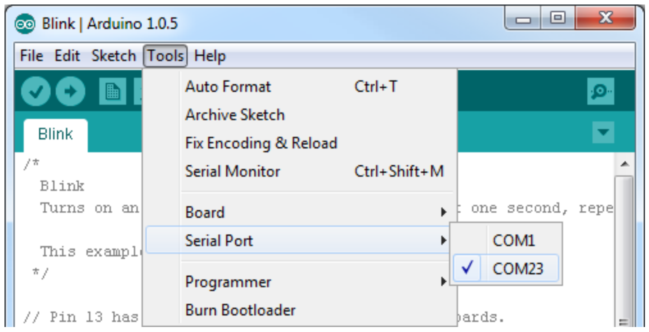
This step is required to tell the Arduino IDE *which* of the many Arduino boards we have. Go up to the **Tools** menu. Then hover over **Board** and make sure **Arduino Uno** is selected.


Tools
Help
Auto Format ⌘T
Archive Sketch
Fix Encoding & Reload
Serial Monitor ⇧⌘M
Board
Serial Port
Programmer
Burn Bootloader
sketch | Arduino
✓ Arduino Uno
Arduino Duemilanove w/ ATmega328
Arduino Diecimila or Duemilanove w/ ATmega168
Arduino Nano w/ ATmega328
Arduino Nano w/ ATmega168
Arduino Mega 2560 or Mega ADK
Arduino Mega (ATmega1280)

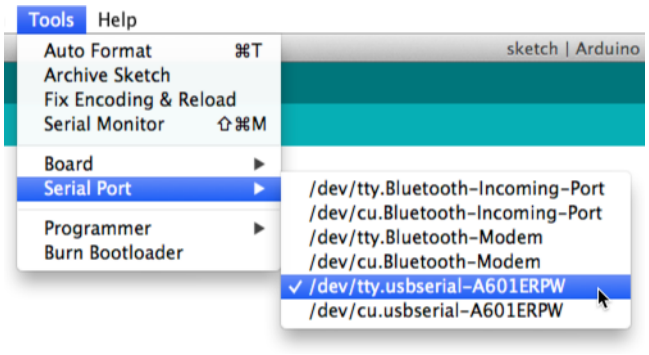
Select a Serial Port

Next up we need to tell the Arduino IDE which of our computer's serial ports the Arduino Uno R3 is connected to. For this, again go up to **Tools**, then hover over **Serial Port** and select your Arduino Uno R3's serial port.

Window Users: This is likely to be com3 or higher (COM1 and COM2 are usually reserved for hardware serial ports). To find out, you can disconnect your Arduino Uno R3 and re-open the menu; the entry that disappears should be the Arduino Uno R3. Reconnect the board and select that serial port.



Mac Users: Select the serial device of the Arduino Uno R3 from the **Tools**, then hover over **Serial Port**. On the Mac, this should be something with /dev/tty.usbmodem or /dev/tty.usbserial in it.



Linux Users: Please visit the Arduino Learning Linux section, to learn more about Arduino on Linux.

Download Arduino Code

You are so close to to being done with setup! Download the SIK Guide Code. **Click** the following link to download the code:

SIK Guide Code

Copy “SIK Guide Code” into “examples” folder in the Arduino folder.

Window Users: Unzip the file “SIK Guide Code”. It should be located in your browser’s “Downloads” folder. Right click the zipped folder and choose “unzip”. Copy the “SIK Guide Code” folder into Arduino’s folder named “examples”.

Mac Users: Unzip the file “SIK Guide Code”. It should be located in your browser’s “Downloads” folder. Right click the zipped folder and choose “unzip”. Find “Arduino” in your applications folder. Right click (ctrl + click) on “Arduino”. Select “Show Package Contents”. Then, click through folders **Contents > Resources > Java > examples**. Copy the “SIK Guide Code” folder into Arduino’s folder named “examples”.

Linux Users: Please visit the Arduino Learning Linux section, to learn more about Arduino on Linux.

Suggested Reading

Before continuing on with this tutorial, we recommend you be somewhat familiar with the concepts in the following tutorial:

- How to Use a Breadboard - First time working with a breadboard? Please check out this tutorial! It will help you understand why the breadboard is a great for prototyping and how to use one.

Time for the fun part!

Continue on to the first experiment and in no time you will be blinking an LED!

Experiment 1: Blinking an LED

Introduction

LEDs are small, powerful lights that are used in many different applications. To start off, we will work on blinking an LED, the Hello World of microcontrollers. That's right - it's as simple as turning a light on and off. It might not seem like much, but establishing this important baseline will give you a solid foundation as we work toward more complex experiments.

Parts Needed

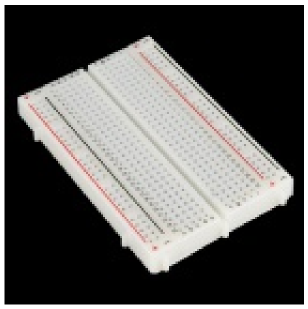
You will need the following parts:

- **1x** Breadboard
- **1x** RedBoard or Arduino Uno R3
- **1x** LED
- **1x** 330Ω Resistor
- **2x** Jumper Wires

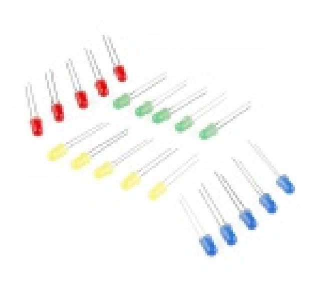
Didn't get the SIK?

If you are following through this experiment and didn't get the SIK, we suggest using these parts:

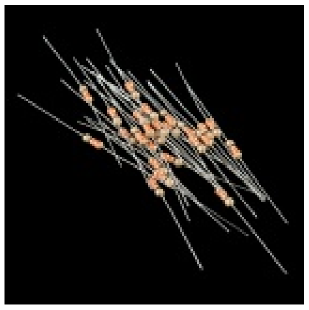
Jumper Wires Standard 7" M/M Pack of 30
◉ PRT-11026
$4.95



Breadboard - Self-Adhesive (White)
◉ PRT-12002
$4.95



LED - Assorted (20 pack)
◉ COM-12062
$2.95



Resistor 330 Ohm 1/6 Watt PTH - 20 pack
◉ COM-11507
$0.95

You will also need either a RedBoard or Arduino Uno R3.

SparkFun RedBoard - Programmed with Arduino
◉ DEV-12757
$19.95

Arduino Uno- R3 SMD
◉ DEV-11224
$29.95

Suggested Reading

Before continuing on with this experiment, we recommend you be familiar with the concepts in the following tutorial:

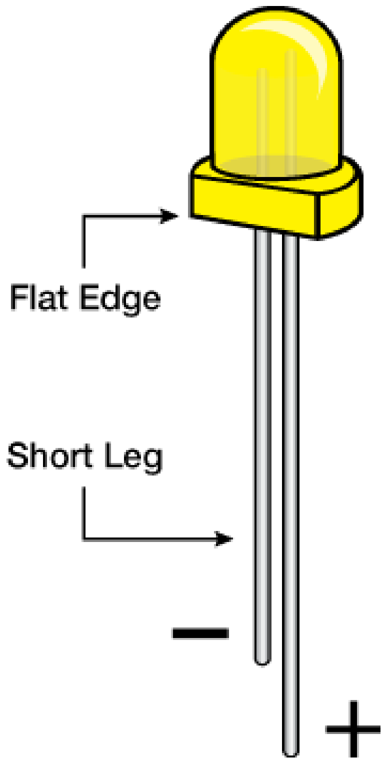
- Light-emitting Diodes - Learn more about LEDs!

Hardware Hookup

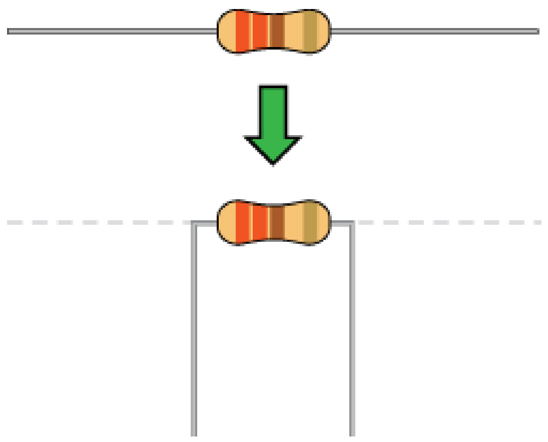
Ready to start hooking everything up? Check out the Fritzing diagram and hookup table below, to see how everything is connected.

Polarized Components ⚠	Pay special attention to the component's markings indicating how to place it on the breadboard. Polarized components can only be connected to a circuit in one direction. Polarized components are highlighted with a yellow warning triangle, in the table below.

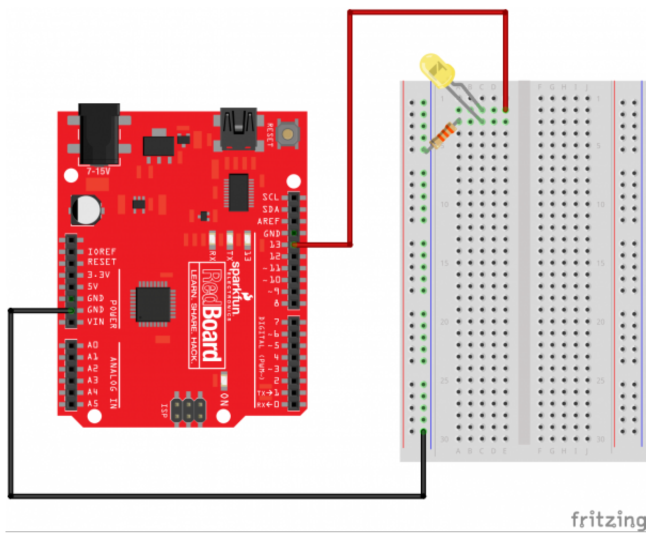
Please note: Pay close attention to the LED. The negative side of the LED is the short leg, marked with a flat edge.



Components like resistors need to have their legs bent into 90° angles in order to correctly fit the breadboard sockets. You can also cut the legs shorter to make them easier to work with on the breadboard.

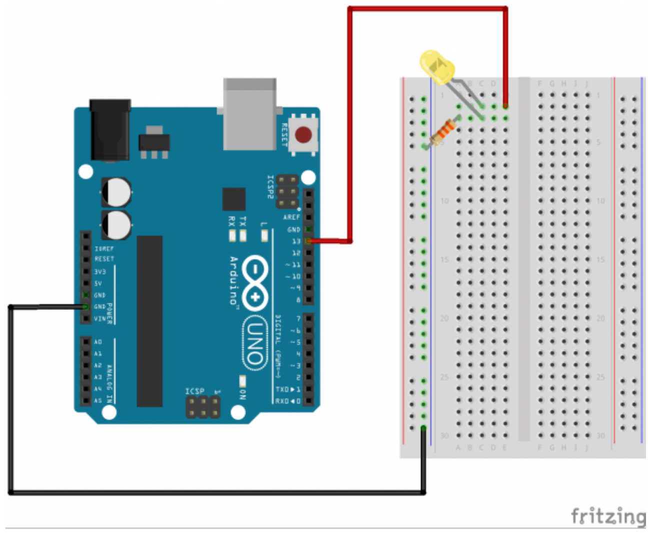


Fritzing Diagram for RedBoard



Having a hard time seeing the circuit? Click on the Fritzing diagram to see a bigger image.

Fritzing Diagram for Arduino



Having a hard time seeing the circuit? Click on the Fritzing diagram to see a bigger image.

Hookup Table

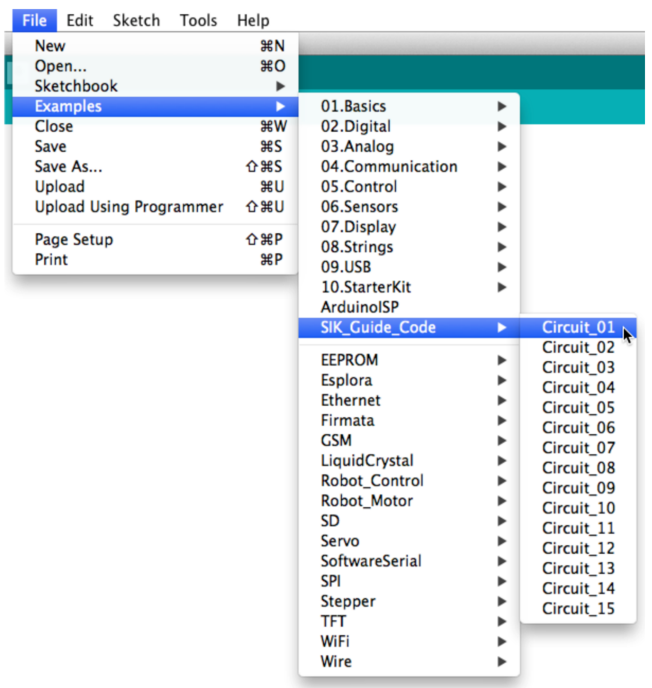
Component	RedBoard or Arduino Uno R3	Breadboard	Breadboard
LED ⚠		c2 LED (+)	c3 LED (-)
330 Resistor		a3	(-)
Jumper Wire	GND	(-)	
Jumper Wire	PIN 13	e2	

In the table, polarized components are highlighted in ***yellow*** *for the* ***whole row*** *and a warning triangle. Polarized components only be connected to a circuit in one direction.*

Open Your First Sketch

Open Up the Arduino IDE software on your computer. Coding in the Arduino language will control your circuit. Open the code for Circuit 1 by accessing the “SIK Guide Code” you downloaded and placed into your “examples” folder earlier.

To open the code go to: **File > Examples > SIK Guide Code > Circuit_01**



You can also copy and paste the following code into the Arduino IDE. Hit upload, and see what happens!

```
/*
SparkFun Inventor's Kit
Example sketch 01

BLINKING A LED

  Turn an LED on for one second, off for one second,
  and repeat forever.

Hardware connections:

  Most Arduinos already have an LED and resistor connected to
  pin 13, so you may not need any additional circuitry.

  But if you'd like to connect a second LED to pin 13, or use
  a different pin, follow these steps:

    Connect the positive side of your LED (longer leg) to Arduino
    digital pin 13 (or another digital pin, don't forget to change
    the code to match).

    Connect the negative side of your LED (shorter leg) to a
    330 Ohm resistor (orange-orange-brown). Connect the other side
    of the resistor to ground.

    pin 13 _____ + LED - _____ 330 Ohm _____ GND

    (We always use resistors between the Arduino and and LEDs
    to keep the LEDs from burning out due to too much current.)
```

```
This sketch was written by SparkFun Electronics,
with lots of help from the Arduino community.
This code is completely free for any use.
Visit http://learn.sparkfun.com/products/2 for SIK information.
Visit http://www.arduino.cc to learn about the Arduino.

Version 2.0 6/2012 MDG
*/


// Welcome to Arduino!

// If you're brand-new to this, there will be some new things to
// learn, but we'll jump right in and explain things as we go.

// The Arduino is a tiny computer that runs programs called
// "sketches". These are text files written using instructions
// the computer understances. You're reading a sketch right now.

// Sketches have computer code in them, but also (hopefully)
// "comments" that explain what the code does. Comments and code
// will have different colors in the editor so you can tell them
// apart.

// This is a comment - anything on a line after "//" is ignored
// by the computer.

/* This is also a comment - this one can be multi-line, but it
must start and end with these characters */

// A "function" is a named block of code, that performs a specific,
// well, function. Many useful functions are already built-in to
// the Arduino; others you'll name and write yourself for your
// own purposes.

// All Arduino sketches MUST have two specific functions, named
// "setup()" and "loop()". The Arduino runs these functions
// automatically when it starts up or if you press the reset
// button. You'll typically fill these function "shells" with your
// own code. Let's get started!


// The setup() function runs once when the sketch starts.
// You'll use it for things you need to do first, or only once:


void setup()
{
  // The Arduino has 13 digital input/output pins. These pins
  // can be configured as either inputs or outputs. We set this
  // up with a built-in function called pinMode().
  // The pinMode() function takes two values, which you type in
  // the parenthesis after the function name. The first value is
  // a pin number, the second value is the word INPUT or OUTPUT.

  // Here we'll set up pin 13 (the one connected to a LED) to be
  // an output. We're doing this because we need to send voltage
  // "out" of the Arduino to the LED.

  pinMode(13, OUTPUT);

  // By the way, the Arduino offers many useful built-in functions
  // like this one. You can find information on all of them at the
  // Arduino website: http://arduino.cc/en/Reference
}


// After setup() finishes, the loop() function runs over and over
// again, forever (or until you turn off or reset the Arduino).
// This is usually where the bulk of your program lives:


void loop()
{
  // The 13 digital pins on your Arduino are great at inputting
  // and outputting on/off, or "digital" signals. These signals
  // will always be either 5 Volts (which we call "HIGH"), or
  // 0 Volts (which we call "LOW").

  // Because we have an LED connected to pin 13, if we make that
  // output HIGH, the LED will get voltage and light up. If we make
  // that output LOW, the LED will have no voltage and turn off.

  // digitalWrite() is the built-in function we use to make an
  // output pin HIGH or LOW. It takes two values; a pin number,
  // followed by the word HIGH or LOW:

  digitalWrite(13, HIGH);   // Turn on the LED

  // delay() is a function that pauses for a given amount of time.
  // It takes one value, the amount of time to wait, measured in
  // milliseconds. There are 1000 milliseconds in a second, so if
  // you delay(1000), it will pause for exactly one second:

  delay(1000);              // Wait for one second

  digitalWrite(13, LOW);    // Turn off the LED

  delay(1000);              // Wait for one second

  // All together, the above code turns the LED on, waits one
  // second, turns it off, and waits another second.

  // When the computer gets to the end of the loop() function,
  // it starts loop() over again. So this program will continue
  // blinking the LED on and off!

  // Try changing the 1000 in the above delay() functions to
  // different numbers and see how it affects the timing. Smaller
  // values will make the loop run faster. (Why?)
}
```

Code to Note

`pinMode(13, OUTPUT);`

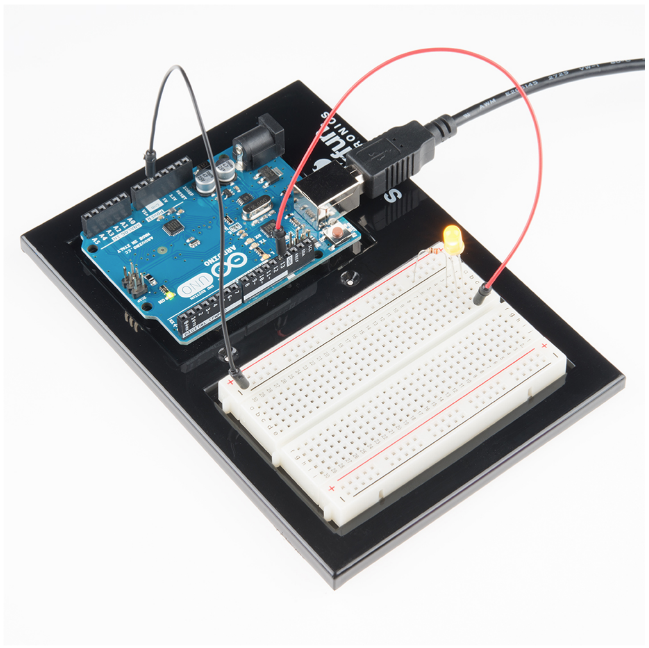
Before you can use one of the Arduino's pins, you need to tell the RedBoard or Arduino Uno R3 whether it is an INPUT or OUTPUT. We use a built-in "function" called `pinMode()` to do this.

`digitalWrite(13, HIGH);`

When you're using a pin as an OUTPUT, you can command it to be HIGH (output 5 volts), or LOW (output 0 volts).

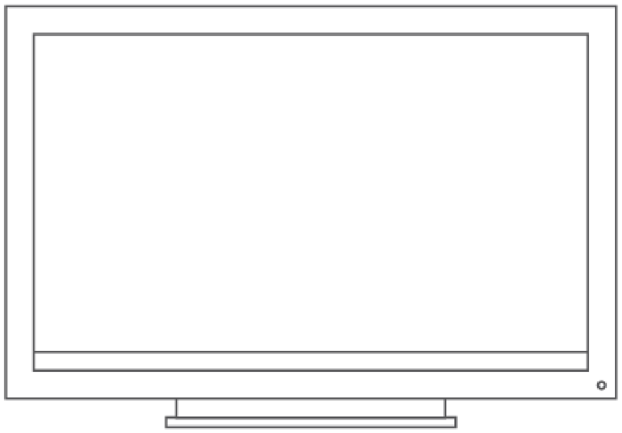
What You Should See

You should see your LED blink on and off. If it isn't, make sure you have assembled the circuit correctly and verified and uploaded the code to your board, or see the troubleshooting section.



Real World Application

Almost all modern flat screen televisions and monitors have LED indicator lights to show they are on or off.



Troubleshooting

Program Not Uploading

This happens sometimes, the most likely cause is a confused serial port, you can change this in **tools > serial port >**

Still No Success?

A broken circuit is no fun, send us an e-mail and we will get back to you as soon as we can: **techsupport@sparkfun.com**

Experiment 2: Reading a Potentiometer

Introduction

In this circuit you'll work with a potentiometer.

A potentiometer is also known as a variable resistor. When powered with 5V, the middle pin outputs a voltage between 0V and 5V, depending on the position of the knob on the potentiometer. A potentiometer is a perfect demonstration of a variable voltage divider circuit. The voltage is divided proportionate to the resistance between the middle pin and the ground pin. In this circuit, you'll learn how to use a potentiometer to control the brightness of an LED.

Parts Needed

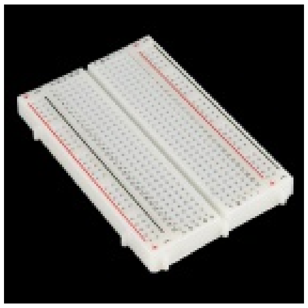
You will need the following parts:

- **1x** Breadboard
- **1x** RedBoard or Arduino Uno
- **1x** LED
- **1x** 330Ω Resistor
- **6x** Jumper Wires
- **1x** Potentiometer

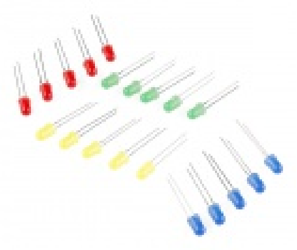
Didn't get the SIK?

If you are following through this experiment and didn't get the SIK, we suggest using these parts:

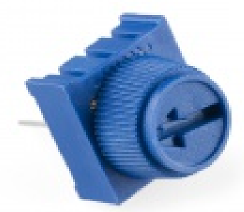
Jumper Wires Standard 7" M/M Pack of 30
◉ PRT-11026
$4.95



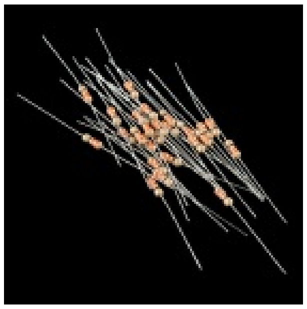
Breadboard - Self-Adhesive (White)
◉ PRT-12002
$4.95



LED - Assorted (20 pack)
◉ COM-12062
$2.95



Trimpot 10K with Knob
◉ COM-09806
$0.95



Resistor 330 Ohm 1/6 Watt PTH - 20 pack
◉ COM-11507
$0.95

You will also need either the RedBoard or Arduino Uno R3.

SparkFun RedBoard - Programmed with Arduino
◉ DEV-12757
$19.95

Arduino Uno- R3 SMD
◉ DEV-11224
$29.95

Suggested Reading

Before continuing on with this experiment, we recommend you be familiar with the concepts in the following tutorial:

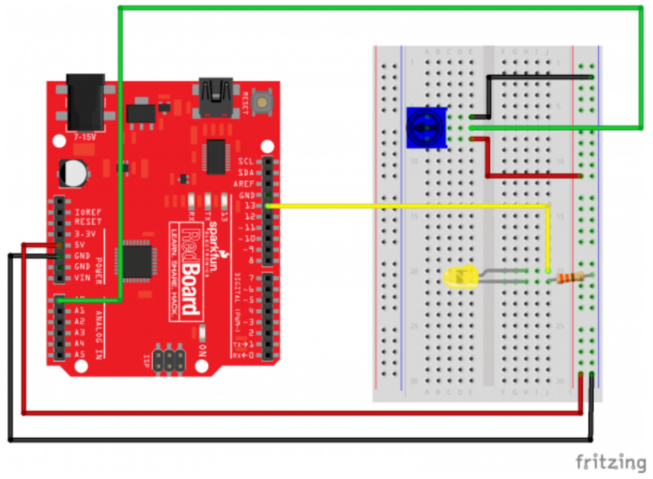
- Analog to Digital Conversion

Hardware Hookup

Ready to start hooking everything up? Check out the Fritzing diagram and hookup table below, to see how everything is connected.

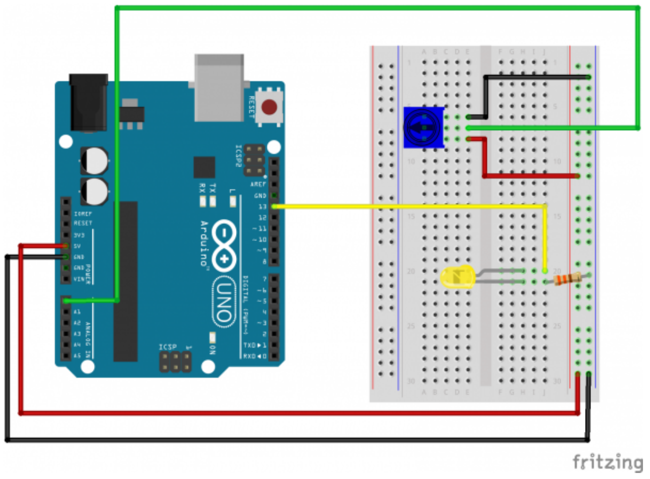
Polarized Components ⚠	Pay special attention to the component's markings indicating how to place it on the breadboard. Polarized components can only be connected to a circuit in one direction. Polarized components are highlighted with a yellow warning triangle, in the table.

Fritzing Diagram for RedBoard



Having a hard time seeing the circuit? Click on the Fritzing diagram to see a bigger image.

Fritzing Diagram for Arduino



Having a hard time seeing the circuit? Click on the Fritzing diagram to see a bigger image.

Hookup Table

Component	RedBoard or Arduino Uno R3	Breadboard	Breadboard	Breadboard
330 Resistor		j21	(-)	
LED ⚠		h20 | LED (+)	h21 | LED (-)	
Potentiometer		a6	a7	a8
Jumper Wire		e6	(-)	
Jumper Wire	A0	e7		
Jumper Wire		e8	(+)	
Jumper Wire	PIN 13	j20		
Jumper Wire	5V	(+)		
Jumper Wire	GND	(-)		

*In the table, polarized components are highlighted in **yellow** for the **whole row** and a warning triangle. Polarized components only be connected to a circuit in one direction.*

Open the Sketch

Open Up the Arduino IDE software on your computer. Coding in the Arduino language will control your circuit. Open the code for Circuit 2 by accessing the “SIK Guide Code” you downloaded and placed into your “Examples” folder earlier.

To open the code go to: **File > examples > SIK Guide Code > Circuit_02**

Copy and paste the following code into the Arduino IDE. Hit upload and see what happens!

```
/* SparkFun Inventor's Kit
Example sketch 02

POTENTIOMETER

  Measure the position of a potentiometer and use it to
  control the blink rate of an LED. Turn the knob to make
  it blink faster or slower!

What's a potentiometer?

  A potentiometer, or "pot" for short, is a control knob.
  It's the same type of control you'd use to change volume,
  dim a lamp, etc. A potentiometer changes resistance as it
  is turned. By using it as a "voltage divider", the Arduino
  can sense the position of the knob, and use that value to
  control whatever you wish (like the blink rate of an LED,
  as we're doing here).

Hardware connections:

  Potentiometer:

    Potentiometers have three pins. When we're using it as a
    voltage divider, we connect the outside pins to power and
    ground. The middle pin will be the signal (a voltage which
    varies from 0 Volts to 5 Volts depending on the position of
    the knob).

    Connect the middle pin to ANALOG IN pin 0 on the Galileo.
    Connect one of the outside pins to 5V.
    Connect the other outside pin to GND.

    (TIP: if once your program is running, the knob feels
    "backwards", you can swap the 5V and GND pins to reverse
    the direction.)

  LED:

    Most Arduinos already have an LED and resistor connected to
    pin 13, so you may not need any additional circuitry.

    But if you'd like to connect a second LED to pin 13, or use
    a different pin, follow these steps:

      Connect the positive side of your LED (longer leg) to
      Arduino digital pin 13 (or another digital pin, but don't
      forget to change the code to match).

      Connect the negative side of your LED (shorter leg) to a
      330 Ohm resistor (orange-orange-brown).
      Connect the other side of the resistor to ground.

This sketch was written by SparkFun Electronics,
with lots of help from the Arduino community.
This code is completely free for any use.
Visit http://learn.sparkfun.com/products/2 for SIK information.
Visit http://www.arduino.cc to learn about the Arduino.

Version 2.0 6/2012 MDG
*/



// Welcome back! In this sketch we'll start using "variables".

// A variable is a named number. We'll often use these to store
// numbers that change, such as measurements from the outside
// world, or to make a sketch easier to understand (sometimes a
// descriptive name makes more sense than looking at a number).

// Variables can be different "data types", which is the kind of
// number we're using (can it be negative? Have a decimal point?)
// We'll introduce more data types later, but for the moment we'll
// stick with good old "integers" (called "int" in your sketch).

// Integers are whole numbers (0, 3, 5643), can be negative, and
// for reasons we won't go into right now, can range from -32768
// to 32767. (Don't worry, if you need to work with larger numbers,
// there are other data types for that. See:
// http://arduino.cc/en/Reference/VariableDeclaration
// for a list of all the data types you can use).

// You must "declare" variables before you use them, so that the
// computer knows about them. Here we'll declare two integer
// variables, and at the same time, initialize them to specific
// values. We're doing this so that further down, we can refer to
// the pins by name rather than number.

// Note that variable names are case-sensitive! If you get an
// "(variable) was not declared in this scope" error, double-check
// that you typed the name correctly.

// Here we're creating a variable called "sensorPin" of type "int"
// and initializing it to have the value "0":

int sensorPin = 0;    // The potentiometer is connected to
                      // analog pin 0

int ledPin = 13;      // The LED is connected to digital pin 13

// One more thing. If you declare variables outside of a function,
// as we have here, they are called "global variables" and can be
// seen by all the functions. If you declare variables within a
// function, they can only be seen within that function. It's good
// practice to "limit the scope" of a variable whenever possible,
// but as we're getting started, global variables are just fine.


void setup() // this function runs once when the sketch starts up
{
  // We'll be using pin 13 to light a LED, so we must configure it
  // as an output.

  // Because we already created a variable called ledPin, and
  // set it equal to 13, we can use "ledPin" in place of "13".
  // This makes the sketch easier to follow.

  pinMode(ledPin, OUTPUT);

  // The above line is the same as "pinMode(13, OUTPUT);"

  // You might be wondering why we're not also configuring
  // sensorPin as an input. The reason is that this is an
  // "analog in" pin. These pins have the special ability to
  // read varying voltages from sensors like the potentiometer.
  // Since they're always used as inputs, there is no need to
  // specifically configure them.
}


void loop() // this function runs repeatedly after setup() finishes
{
  // First we'll declare another integer variable
  // to store the value of the potentiometer:

  int sensorValue;

  // The potentiometer is set up as a voltage divider, so that
  // when you turn it, the voltage on the center pin will vary
  // from 0V to 5V. We've connected the center pin on the
  // potentiometer to the Arduino's analog input 0.

  // The Arduino can read external voltages on the analog input
  // pins using a built-in function called analogRead(). This
  // function takes one input value, the analog pin we're using
  // (sensorPin, which we earlier set to 0). It returns an integer
  // number that ranges from 0 (0 Volts) to 1023 (5 Volts).
  // We're sticking this value into the sensorValue variable:

  sensorValue = analogRead(sensorPin);

  // Now we'll blink the LED like in the first example, but we'll
  // use the sensorValue variable to change the blink speed
  // (the smaller the number, the faster it will blink).

  // Note that we're using the ledPin variable here as well:

  digitalWrite(ledPin, HIGH);     // Turn the LED on

  delay(sensorValue);             // Pause for sensorValue
                                  // milliseconds

  digitalWrite(ledPin, LOW);      // Turn the LED off

  delay(sensorValue);             // Pause for sensorValue
                                  // milliseconds

  // Remember that loop() repeats forever, so we'll do all this
  // again and again.
}
```

Code To Note

`int sensorValue;`

A “variable” is a placeholder for values that may change in your code. You must introduce, or “declare” variables before you use them; here we’re declaring a variable called sensorValue, of type “int” (integer). Don’t forget that variable names are case-sensitive!

`sensorValue = analogRead(sensorPin);`

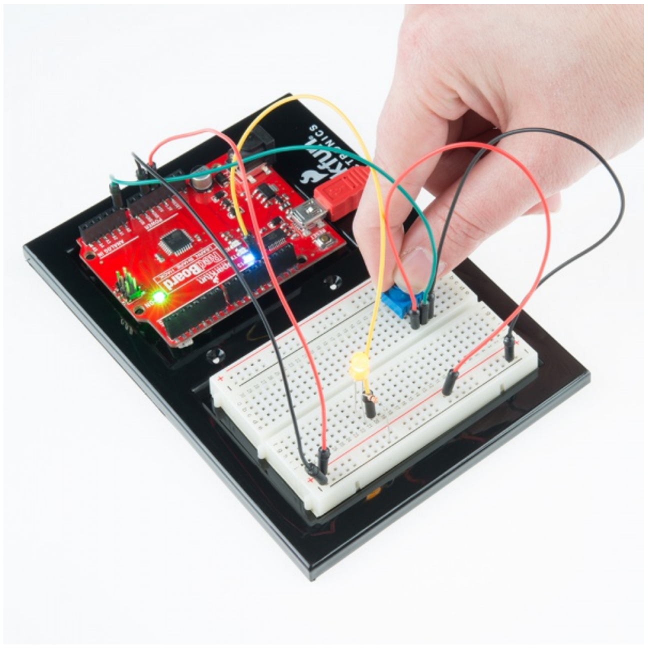
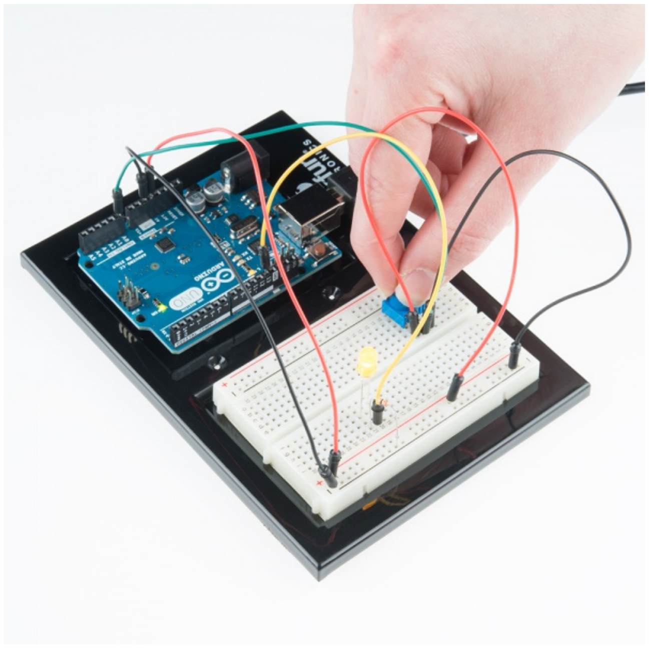
We use the `analogRead()` function to read the value on an analog pin. `analogRead()` takes one parameter, the analog pin you want to use (“sensorPin”), and returns a number (“sensorValue”) between 0 (0 volts) and 1023 (5 volts).

`delay(sensorValue);`

Microcontrollers are very fast, capable of running thousands of lines of code each second. To slow it down so that we can see what it’s doing, we’ll often insert delays into the code. `delay()` counts in milliseconds; there are 1000 ms in one second.

What You Should See

You should see the LED blink faster or slower in accordance with your 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 section.

Real World Application

Most traditional volume knobs employ a potentiometer.

Troubleshooting

Sporadically Working

This is most likely due to a slightly dodgy connection with the potentiometer's pins. This can usually be conquered by holding the potentiometer down.

Not Working

Make sure you haven't accidentally connected the wiper, the resistive element in the potentiometer, to digital pin 0 rather than analog pin 0. (the row of pins beneath the power pins).

LED Not Lighting Up?

LEDs will only work in one direction. Double check your connections.

Experiment 3: Driving and RGB LED

Introduction

You know what's even more fun than a blinking LED? Changing colors with one LED. RGB, or red-green-blue, LEDs have three different color-emitting diodes that can be combined to create all sorts of colors. In this circuit, you'll learn how to use an RGB LED to create unique color combinations. Depending on how bright each diode is, nearly any color is possible!

Parts Needed

You will need the following parts:

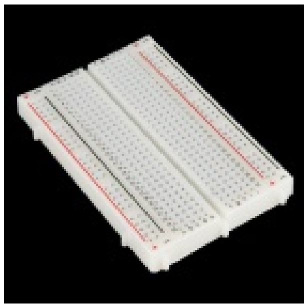
- **1x** Breadboard
- **1x** RedBoard or Arduino Uno
- **1x** LED - RGB Common Cathode

- **3x** 330Ω Resistors
- **5x** Jumper Wires

Didn't get the SIK?

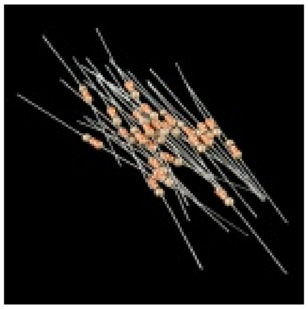
If you are following through this experiment and didn't get the SIK, we suggest using these parts:

Jumper Wires Standard 7" M/M Pack of 30
● PRT-11026
$4.95



Breadboard - Self-Adhesive (White)
● PRT-12002
$4.95

LED - RGB Clear Common Cathode
● COM-00105
$1.95



Resistor 330 Ohm 1/6 Watt PTH - 20 pack
● COM-11507
$0.95

You will also need either the RedBoard or Arduino Uno R3.

SparkFun RedBoard - Programmed with Arduino
● DEV-12757
$19.95

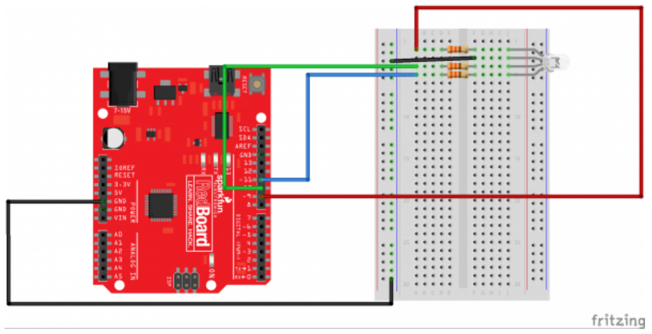
Arduino Uno- R3 SMD
● DEV-11224
$29.95

Hardware Hookup

Ready to start hooking everything up? Check out the Fritzing diagram and hookup table below, to see how everything is connected.

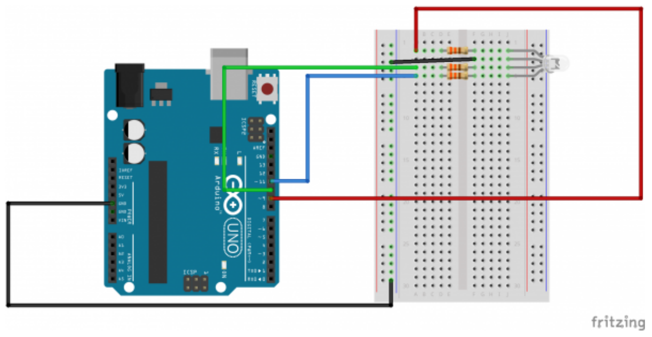
Polarized Components ⚠	Pay special attention to the component's markings indicating how to place it on the breadboard. Polarized components can only be connected to a circuit in one direction. Polarized components are highlighted with a yellow warning triangle, in the table below.

Fritzing Diagram for RedBoard



Having a hard time seeing the circuit? Click on the Fritzing diagram to see a bigger image.

Fritzing Diagram for Arduino



Having a hard time seeing the circuit? Click on the Fritzing diagram to see a bigger image.

Hookup Table

Component	**RedBoard or Arduino Uno R3**	**Breadboard**	**Breadboard**	**Breadboard**	**Breadboard**
RGB LED ⚠		j2 (RED)	j3 (GND)	j4 (GREEN)	j5 (BLUE)
330 Resistor		d2	f2		
330 Resistor		d4	f4		
330 Resistor		d5	f5		
Jumper Wire	GND	(-)			

Jumper Wire	PIN 9	a2			
Jumper Wire		f3	(-)		
Jumper Wire	PIN 10	a4			
Jumper Wire	PIN 11	a5			

Open the Sketch

Open Up the Arduino IDE software on your computer. Coding in the Arduino language will control your circuit. Open the code for Circuit 3 by accessing the “SIK Guide Code” you downloaded and placed into your “Examples” folder earlier.

To open the code go to: **File > examples > SIK Guide Code > Circuit_03**

You can also copy and paste the following code into the Arduino IDE. Hit upload, and see what happens!

```
/*
SparkFun Inventor's Kit
Example sketch 03

RGB LED

Make an RGB LED display a rainbow of colors!

Hardware connections:

An RGB LED is actually three LEDs (red, green, and blue) in
one package. When you run them at different brightnesses,
the red, green and blue mix to form new colors.

Starting at the flattened edge of the flange on the LED,
the pins are ordered RED, COMMON, GREEN, BLUE.

Connect RED to a 330 ohm resistor. Connect the other end
of the resistor to Arduino digital pin 9.

Connect COMMON pin to GND.

Connect GREEN to a 330 ohm resistor. Connect the other end
of the resistor to Arduino digital pin 10.

Connect BLUE to a 330 ohm resistor. Connect the other end
of the resistor to Arduino digital pin 11.
```

```
This sketch was written by SparkFun Electronics,
with lots of help from the Arduino community.
Visit http://learn.sparkfun.com/products/2 for SIK information.
Visit http://www.arduino.cc to learn about the Arduino.

Version 2.0 6/2012 MDG
*/


// First we'll define the pins by name to make the sketch
// easier to follow.

// Here's a new trick: putting the word "const" in front of a
// variable indicates that this is a "constant" value that will
// never change. (You don't have to do this, but if you do, the
// Arduino will give you a friendly warning if you accidentally
// try to change the value, so it's considered good form.)

const int RED_PIN = 9;
const int GREEN_PIN = 10;
const int BLUE_PIN = 11;

// This variable controls how fast we loop through the colors.
// (Try changing this to make the fading faster or slower.)

int DISPLAY_TIME = 100;  // In milliseconds


void setup()
{
// Here we'll configure the Arduino pins we're using to
// drive the LED to be outputs:

pinMode(RED_PIN, OUTPUT);
pinMode(GREEN_PIN, OUTPUT);
pinMode(BLUE_PIN, OUTPUT);
}


void loop()
{
// In this sketch, we'll start writing our own functions.
// This makes the sketch easier to follow by dividing up
// the sketch into sections, and not having everything in
// setup() or loop().

// We'll show you two ways to run the RGB LED.

// The first way is to turn the individual LEDs (red, blue,
// and green) on and off in various combinations. This gives you
// a total of eight colors (if you count "black" as a color).
// We've written a function called mainColors() that steps
// through all eight of these colors. We're only "calling" the
// function here (telling it to run). The actual function code
// is further down in the sketch.

mainColors();

// The above function turns the individual LEDs full-on and
// full-off. If you want to generate more than eight colors,
// you can do so by varying the brightness of the individual
// LEDs between full-on and full-off.

// The analogWrite() function lets us do this. This function
// lets you dim a LED from full-off to full-on over 255 steps.

// We've written a function called showSpectrum() that smoothly
// steps through all the colors. Again we're just calling it
// here; the actual code is further down in this sketch.

showSpectrum();
}


// Here's the mainColors() function we've written.

// This function displays the eight "main" colors that the RGB LED
// can produce. If you'd like to use one of these colors in your
// own sketch, you cancopy and paste that section into your code.

void mainColors()
{
// Off (all LEDs off):

digitalWrite(RED_PIN, LOW);
digitalWrite(GREEN_PIN, LOW);
digitalWrite(BLUE_PIN, LOW);

delay(1000);

// Red (turn just the red LED on):

digitalWrite(RED_PIN, HIGH);
digitalWrite(GREEN_PIN, LOW);
digitalWrite(BLUE_PIN, LOW);

delay(1000);

// Green (turn just the green LED on):

digitalWrite(RED_PIN, LOW);
digitalWrite(GREEN_PIN, HIGH);
digitalWrite(BLUE_PIN, LOW);

delay(1000);

// Blue (turn just the blue LED on):

digitalWrite(RED_PIN, LOW);
digitalWrite(GREEN_PIN, LOW);
digitalWrite(BLUE_PIN, HIGH);

delay(1000);

// Yellow (turn red and green on):

digitalWrite(RED_PIN, HIGH);
digitalWrite(GREEN_PIN, HIGH);
digitalWrite(BLUE_PIN, LOW);

delay(1000);

// Cyan (turn green and blue on):

digitalWrite(RED_PIN, LOW);
digitalWrite(GREEN_PIN, HIGH);
digitalWrite(BLUE_PIN, HIGH);

delay(1000);

// Purple (turn red and blue on):

digitalWrite(RED_PIN, HIGH);
digitalWrite(GREEN_PIN, LOW);
digitalWrite(BLUE_PIN, HIGH);

delay(1000);

// White (turn all the LEDs on):

digitalWrite(RED_PIN, HIGH);
digitalWrite(GREEN_PIN, HIGH);
digitalWrite(BLUE_PIN, HIGH);

delay(1000);
}


// Below are two more functions we've written,
// showSpectrum() and showRGB().

// showRGB() displays a single color on the RGB LED.
// You call showRGB() with the number of a color you want
// to display.

// showSpectrum() steps through all the colors of the RGB LED,
// displaying a rainbow. showSpectrum() actually calls showRGB()
// over and over to do this.

// We'll often break tasks down into individual functions like
// this, which makes your sketches easier to follow, and once
// you have a handy function, you can reuse it in your other
// programs.


// showSpectrum()

// This function steps through all the colors of the RGB LED.
// It does this by stepping a variable from 0 to 768 (the total
// number of colors), and repeatedly calling showRGB() to display
// the individual colors.

// In this function, we're using a "for() loop" to step a variable
// from one value to another, and perform a set of instructions
// for each step. For() loops are a very handy way to get numbers
// to count up or down.

// Every for() loop has three statements separated by semicolons:

//   1. Something to do before starting

//   2. A test to perform; as long as it's true,
//      it will keep looping

//   3. Something to do after each loop (usually
//      increase a variable)

// For the for() loop below, these are the three statements:

//   1. x = 0;     Before starting, make x = 0.

//   2. x < 768;   While x is less than 768, run the
//                 following code.

//   3. x++        Putting "++" after a variable means
//                 "add one to it". (You can also use "x = x + 1")

// Every time you go through the loop, the statements following
// the loop (those within the brackets) will run.

// And when the test in statement 2 is finally false, the sketch
// will continue.
```

```
void showSpectrum()
{
int x;  // define an integer variable called "x"

// Now we'll use a for() loop to make x count from 0 to 767
// (Note that there's no semicolon after this line!
// That's because the for() loop will repeat the next
// "statement", which in this case is everything within
// the following brackets {} )

for (x = 0; x < 768; x++)

// Each time we loop (with a new value of x), do the following:

{
showRGB(x);  // Call RGBspectrum() with our new x
delay(10);   // Delay for 10 ms (1/100th of a second)
}
}


// showRGB()

// This function translates a number between 0 and 767 into a
// specific color on the RGB LED. If you have this number count
// through the whole range (0 to 767), the LED will smoothly
// change color through the entire spectrum.

// The "base" numbers are:
// 0   = pure red
// 255 = pure green
// 511 = pure blue
// 767 = pure red (again)

// Numbers between the above colors will create blends. For
// example, 640 is midway between 512 (pure blue) and 767
// (pure red). It will give you a 50/50 mix of blue and red,
// resulting in purple.

// If you count up from 0 to 767 and pass that number to this
// function, the LED will smoothly fade between all the colors.
// (Because it starts and ends on pure red, you can start over
// at 0 without any break in the spectrum).


void showRGB(int color)
{
int redIntensity;
int greenIntensity;
int blueIntensity;
// Here we'll use an "if / else" statement to determine which
// of the three (R,G,B) zones x falls into. Each of these zones
// spans 255 because analogWrite() wants a number from 0 to 255.

// In each of these zones, we'll calculate the brightness
// for each of the red, green, and blue LEDs within the RGB LED.

if (color <= 255)          // zone 1
{
redIntensity = 255 - color;    // red goes from on to off
greenIntensity = color;        // green goes from off to on
blueIntensity = 0;             // blue is always off
}
else if (color <= 511)     // zone 2
{
redIntensity = 0;                     // red is always off
greenIntensity = 255 - (color - 256); // green on to off
blueIntensity = (color - 256);        // blue off to on
}
else // color >= 512       // zone 3
{
redIntensity = (color - 512);         // red off to on
greenIntensity = 0;                   // green is always off
blueIntensity = 255 - (color - 512);  // blue on to off
}

// Now that the brightness values have been set, command the LED
// to those values

analogWrite(RED_PIN, redIntensity);
analogWrite(BLUE_PIN, blueIntensity);
analogWrite(GREEN_PIN, greenIntensity);
}
```

Code To Note

```
for (x = 0; x < 768; x++)
{}
```

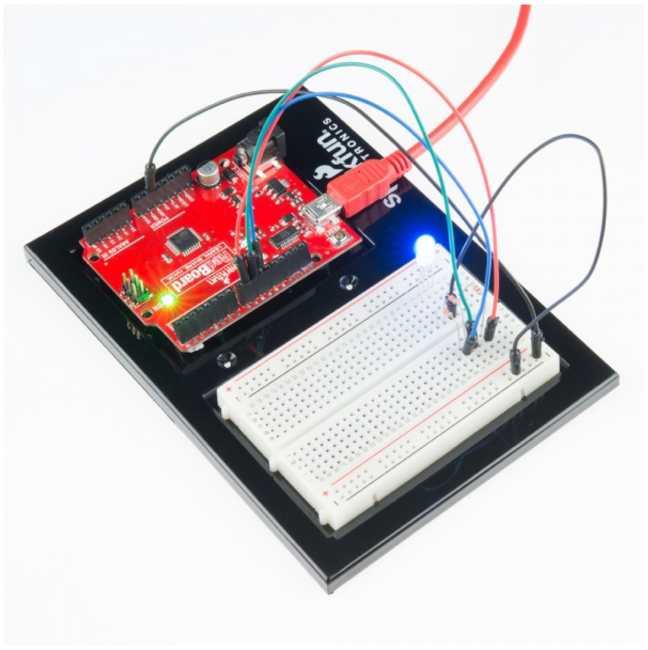
A `for()` loop is used to repeat an action a set number of times across a range, and repeatedly runs code within the brackets {}. Here the variable “x” starts a 0, ends at 767, and increases by one each time (“x++”).

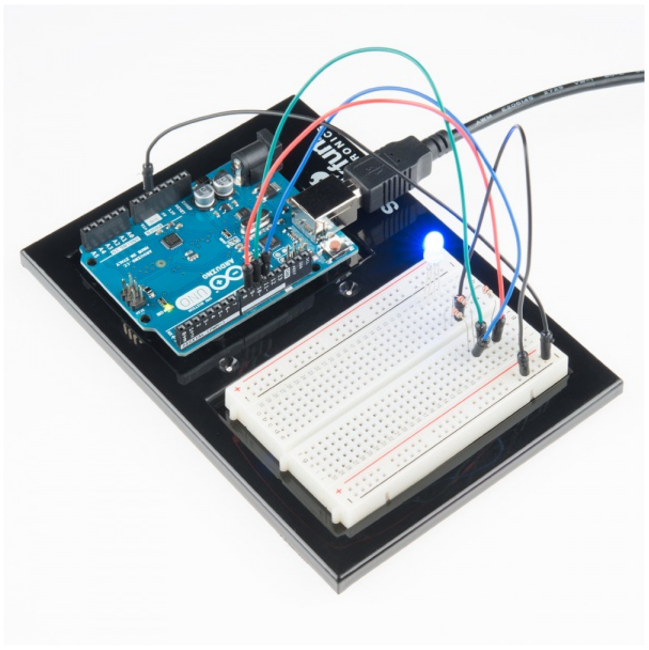
```
if (x <= 255)
{}
else
{}
```

“If / else” statements are used to make choices in your programs. The statement within the parenthesis () is evaluated; if it’s true, the code within the first brackets {} will run. If it’s not true, the code within the second brackets {} will run.

What You Should See

You should see your LED turn on, but this time in new, crazy colors! If it isn’t, make sure you have assembled the circuit correctly and verified and uploaded the code to your board or see the troubleshooting section.





Real World Application

Many electronics such as video game consoles use RGB LEDs to have the versatility to show different colors in the same area. Often times the different colors represent different states of working condition.

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. Double check each pin is where it should be.

Seeing Red

The red diode within the RGB LED may be a bit brighter than the other two. To make your colors more balanced, use a higher ohm resistor. Or adjust in code.

```
analogWrite(RED_PIN, redIntensity);
```

to

```
analogWrite(RED_PIN, redIntensity/3);
```

Experiment 4: Driving Multiple LEDs

Introduction

Now that you've gotten your LED to blink on and off, it's time to up the stakes a little bit – by connecting **eight LEDs at once**. We'll also give your RedBoard or Arduino R3 a little test by creating various lighting sequences. This circuit is a great setup to start practicing writing your own programs and getting a feel for the way Arduino works.

Along with controlling the LEDs, you'll learn about a couple programming tricks that keep your code neat and tidy:

`for() loops` - used when you want to run a piece of code several times

`arrays[ ]` - used to make managing variables easier by grouping them together

Parts Needed

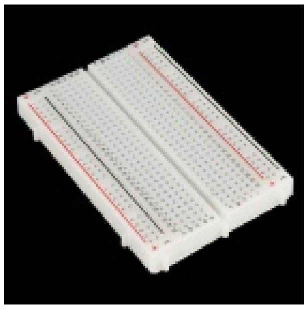
You will need the following parts:

- **1x** Breadboard
- **1x** RedBoard or Arduino Uno
- **8x** LEDs
- **8x** 330Ω Resistors
- **9x** Jumper Wires

Didn't get the SIK?

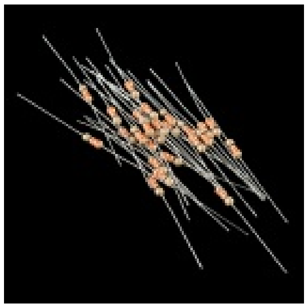
If you are following through this experiment and didn't get the SIK, we suggest using these parts:

Jumper Wires Standard 7" M/M Pack of 30
◉ PRT-11026
$4.95



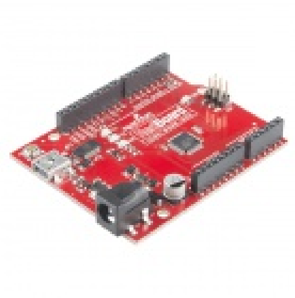
Breadboard - Self-Adhesive (White)
◉ PRT-12002
$4.95

LED - Assorted (20 pack)
◉ COM-12062
$2.95



Resistor 330 Ohm 1/6 Watt PTH - 20 pack
◉ COM-11507
$0.95

You will also need either a RedBoard or Arduino Uno R3.



SparkFun RedBoard - Programmed with Arduino

◉ DEV-12757

$19.95

Arduino Uno- R3 SMD

◉ DEV-11224

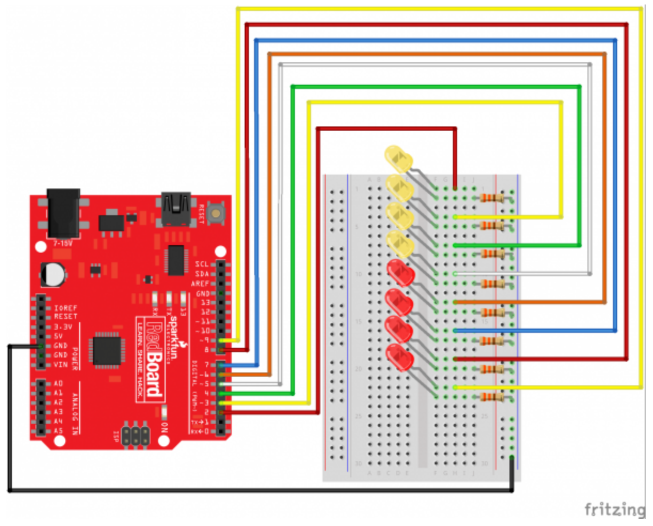
$29.95

Hardware Hookup

Ready to start hooking everything up? Check out the Fritzing diagram and hookup table below, to see how everything is connected.

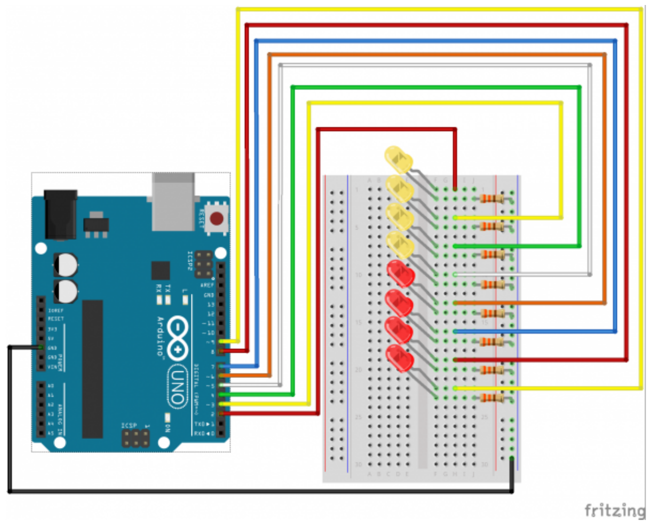
Polarized Components ⚠	Pay special attention to the component’s markings indicating how to place it on the breadboard. Polarized components can only be connected to a circuit in one direction.

Fritzing Diagram for RedBoard



Having a hard time seeing the circuit? Click on the Fritzing diagram to see a bigger image.

Fritzing Diagram for Arduino



Having a hard time seeing the circuit? Click on the Fritzing diagram to see a bigger image.

Open the Sketch

Open Up the Arduino IDE software on your computer. Coding in the Arduino language will control your circuit. Open the code for Circuit 4 by accessing the “SIK Guide Code” you downloaded and placed into your “Examples” folder earlier.

To open the code go to: **File > examples > SIK Guide Code > Circuit_04**

You can also copy and paste the following code into the Arduino IDE. Hit upload, and see what happens!

```
/*
SparkFun Inventor's Kit
Example sketch 04

MULTIPLE LEDs

  Make eight LEDs dance. Dance LEDs, dance!

Hardware connections:

  You'll need eight LEDs, and eight 330 Ohm resistors
  (orange-orange-brown).

    For each LED, connect the negative side (shorter leg)
    to a 330 Ohm resistor.

    Connect the other side of the resistors to GND.

    Connect the positive side (longer leg) of the LEDs
    to Arduino digital pins 2 through 9.

This sketch was written by SparkFun Electronics,
with lots of help from the Arduino community.
This code is completely free for any use.
Visit http://learn.sparkfun.com/products/2 for SIK information.
Visit http://www.arduino.cc to learn about the Arduino.

Version 2.0 6/2012 MDG
*/


// To keep track of all the LED pins, we'll use an "array".
// An array lets you store a group of variables, and refer to them
// by their position, or "index". Here we're creating an array of
// eight integers, and initializing them to a set of values:

int ledPins[] = {2,3,4,5,6,7,8,9};

// The first element of an array is index 0.
// We've put the value "2" in index 0, "3" in index 1, etc.
// The final index in the above array is 7, which contains
// the value "9".

// We're using the values in this array to specify the pin numbers
// that the eight LEDs are connected to. LED 0 is connected to
// pin 2, LED 1 is connected to pin 3, etc.


void setup()
{
  int index;

  // In this sketch, we'll use "for() loops" to step variables from
  // one value to another, and perform a set of instructions for
  // each step. For() loops are a very handy way to get numbers to
  // count up or down.

  // Every for() loop has three statements separated by
  // semicolons (;):

  //   1. Something to do before starting
  //   2. A test to perform; as long as it's true, keep looping
  //   3. Something to do after each loop (increase a variable)

  // For the for() loop below, these are the three statements:

  //   1. index = 0;    Before starting, make index = 0.
  //   2. index <= 7;   If index is less or equal to 7,
  //                    run the following code.
  //            (When index = 8, continue with the sketch.)
  //   3. index++   Putting "++" after a variable means
  //                    "add one to it".
  //            (You can also use "index = index + 1".)

  // Every time you go through the loop, the statements following
  // the for() (within the brackets) will run.
  // When the test in statement 2 is finally false, the sketch
  // will continue.


  // Here we'll use a for() loop to initialize all the LED pins
  // to outputs. This is much easier than writing eight separate
  // statements to do the same thing.

  // This for() loop will make index = 0, then run the pinMode()
  // statement within the brackets. It will then do the same thing
  // for index = 2, index = 3, etc. all the way to index = 7.

  for(index = 0; index <= 7; index++)
  {
    pinMode(ledPins[index],OUTPUT);
    // ledPins[index] is replaced by the value in the array.
    // For example, ledPins[0] is 2
  }
}


void loop()
{
  // This loop() calls functions that we've written further below.
  // We've disabled some of these by commenting them out (putting
  // "//" in front of them). To try different LED displays, remove
  // the "//" in front of the ones you'd like to run, and add "//"
  // in front of those you don't to comment out (and disable) those
  // lines.

  oneAfterAnotherNoLoop();  // Light up all the LEDs in turn

  //oneAfterAnotherLoop();  // Same as oneAfterAnotherNoLoop,
                            // but with much less typing

  //oneOnAtATime();         // Turn on one LED at a time,
                            // scrolling down the line

  //pingPong();             // Light the LEDs middle to the edges

  //marquee();              // Chase lights like you see on signs

  //randomLED();            // Blink LEDs randomly
}


/*
oneAfterAnotherNoLoop()

This function will light one LED, delay for delayTime, then light
```

```
the next LED, and repeat until all the LEDs are on. It will then
turn them off in the reverse order.

This function does NOT use a for() loop. We've done it the hard way
to show you how much easier life can be when you use for() loops.
Take a look at oneAfterAnotherLoop() further down, which does
exactly the same thing with much less typing.
*/

void oneAfterAnotherNoLoop()
{
  int delayTime = 100; // time (milliseconds) to pause between LEDs
                       // make this smaller for faster switching

  // turn all the LEDs on:

  digitalWrite(ledPins[0], HIGH);  //Turns on LED #0 (pin 2)
  delay(delayTime);                //wait delayTime milliseconds
  digitalWrite(ledPins[1], HIGH);  //Turns on LED #1 (pin 3)
  delay(delayTime);                //wait delayTime milliseconds
  digitalWrite(ledPins[2], HIGH);  //Turns on LED #2 (pin 4)
  delay(delayTime);                //wait delayTime milliseconds
  digitalWrite(ledPins[3], HIGH);  //Turns on LED #3 (pin 5)
  delay(delayTime);                //wait delayTime milliseconds
  digitalWrite(ledPins[4], HIGH);  //Turns on LED #4 (pin 6)
  delay(delayTime);                //wait delayTime milliseconds
  digitalWrite(ledPins[5], HIGH);  //Turns on LED #5 (pin 7)
  delay(delayTime);                //wait delayTime milliseconds
  digitalWrite(ledPins[6], HIGH);  //Turns on LED #6 (pin 8)
  delay(delayTime);                //wait delayTime milliseconds
  digitalWrite(ledPins[7], HIGH);  //Turns on LED #7 (pin 9)
  delay(delayTime);                //wait delayTime milliseconds

  // turn all the LEDs off:

  digitalWrite(ledPins[7], LOW);   //Turn off LED #7 (pin 9)
  delay(delayTime);                //wait delayTime milliseconds
  digitalWrite(ledPins[6], LOW);   //Turn off LED #6 (pin 8)
  delay(delayTime);                //wait delayTime milliseconds
  digitalWrite(ledPins[5], LOW);   //Turn off LED #5 (pin 7)
  delay(delayTime);                //wait delayTime milliseconds
  digitalWrite(ledPins[4], LOW);   //Turn off LED #4 (pin 6)
  delay(delayTime);                //wait delayTime milliseconds
  digitalWrite(ledPins[3], LOW);   //Turn off LED #3 (pin 5)
  delay(delayTime);                //wait delayTime milliseconds
  digitalWrite(ledPins[2], LOW);   //Turn off LED #2 (pin 4)
  delay(delayTime);                //wait delayTime milliseconds
  digitalWrite(ledPins[1], LOW);   //Turn off LED #1 (pin 3)
  delay(delayTime);                //wait delayTime milliseconds
  digitalWrite(ledPins[0], LOW);   //Turn off LED #0 (pin 2)
  delay(delayTime);                //wait delayTime milliseconds
}


/*
oneAfterAnotherLoop()

This function does exactly the same thing as oneAfterAnotherNoLoop(),
but it takes advantage of for() loops and the array to do it with
much less typing.
*/

void oneAfterAnotherLoop()
{
  int index;
  int delayTime = 100; // milliseconds to pause between LEDs
                       // make this smaller for faster switching

  // Turn all the LEDs on:

  // This for() loop will step index from 0 to 7
  // (putting "++" after a variable means add one to it)
  // and will then use digitalWrite() to turn that LED on.

  for(index = 0; index <= 7; index++)
  {
    digitalWrite(ledPins[index], HIGH);
    delay(delayTime);
  }

  // Turn all the LEDs off:

  // This for() loop will step index from 7 to 0
  // (putting "--" after a variable means subtract one from it)
  // and will then use digitalWrite() to turn that LED off.

  for(index = 7; index >= 0; index--)
  {
    digitalWrite(ledPins[index], LOW);
    delay(delayTime);
  }
}


/*
oneOnAtATime()

This function will step through the LEDs,
lighting only one at at time.
*/

void oneOnAtATime()
```

```
{
  int index;
  int delayTime = 100; // milliseconds to pause between LEDs
                       // make this smaller for faster switching

  // step through the LEDs, from 0 to 7

  for(index = 0; index <= 7; index++)
  {
    digitalWrite(ledPins[index], HIGH);  // turn LED on
    delay(delayTime);                    // pause to slow down
    digitalWrite(ledPins[index], LOW);   // turn LED off
  }
}


/*
pingPong()

This function will step through the LEDs,
lighting one at at time in both directions.
*/

void pingPong()
{
  int index;
  int delayTime = 100; // milliseconds to pause between LEDs
                       // make this smaller for faster switching

  // step through the LEDs, from 0 to 7

  for(index = 0; index <= 7; index++)
  {
    digitalWrite(ledPins[index], HIGH);  // turn LED on
    delay(delayTime);                    // pause to slow down
    digitalWrite(ledPins[index], LOW);   // turn LED off
  }

  // step through the LEDs, from 7 to 0

  for(index = 7; index >= 0; index--)
  {
    digitalWrite(ledPins[index], HIGH);  // turn LED on
    delay(delayTime);                    // pause to slow down
    digitalWrite(ledPins[index], LOW);   // turn LED off
  }
}


/*
marquee()
```

```
This function will mimic "chase lights" like those around signs.
*/

void marquee()
{
  int index;
  int delayTime = 200; // milliseconds to pause between LEDs
                       // Make this smaller for faster switching

  // Step through the first four LEDs
  // (We'll light up one in the lower 4 and one in the upper 4)

  for(index = 0; index <= 3; index++) // Step from 0 to 3
  {
    digitalWrite(ledPins[index], HIGH);    // Turn a LED on
    digitalWrite(ledPins[index+4], HIGH);  // Skip four, and turn that LED on
    delay(delayTime);                      // Pause to slow down the sequence
    digitalWrite(ledPins[index], LOW);     // Turn the LED off
    digitalWrite(ledPins[index+4], LOW);   // Skip four, and turn that LED off
  }
}


/*
randomLED()

This function will turn on random LEDs. Can you modify it so it
also lights them for random times?
*/

void randomLED()
{
  int index;
  int delayTime;

  // The random() function will return a semi-random number each
  // time it is called. See http://arduino.cc/en/Reference/Random
  // for tips on how to make random() even more random.

  index = random(8);    // pick a random number between 0 and 7
  delayTime = 100;

  digitalWrite(ledPins[index], HIGH);  // turn LED on
  delay(delayTime);                    // pause to slow down
  digitalWrite(ledPins[index], LOW);   // turn LED off
}
```

Code To Note

```
int ledPins[] = {2,3,4,5,6,7,8,9};
```

When you have to manage a lot of variables, an “array” is a handy way to group them together. Here we’re creating an array of integers, called ledPins, with eight elements.

```
digitalWrite(ledPins[0], HIGH);
```

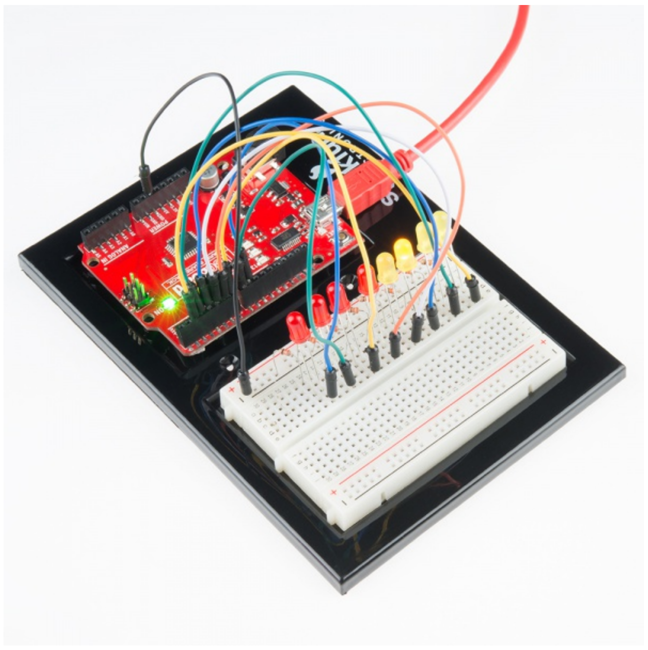
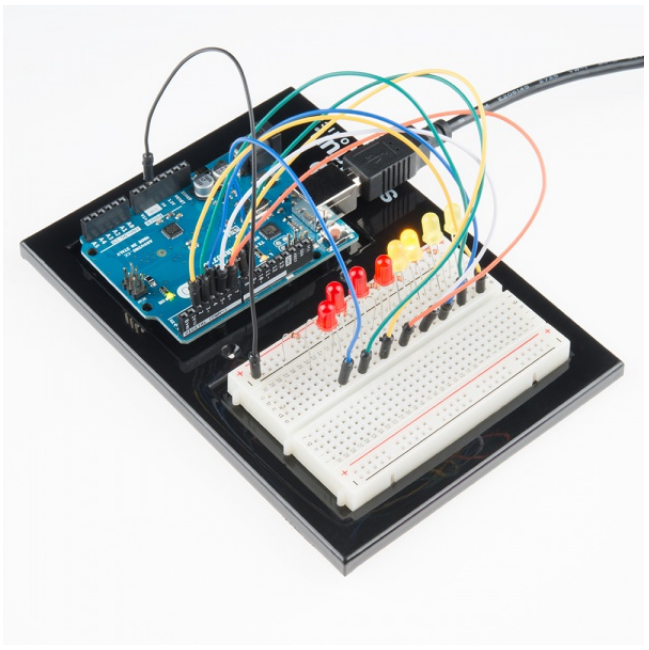
You refer to the elements in an array by their position. The first element is at position 0, the second is at position 1, etc. You refer to an element using “ledPins[x]” where x is the position. Here we’re making digital pin 2 HIGH, since the array element at position 0 is “2”.

```
index = random(8);
```

Computers like to do the same things each time they run. But sometimes you want to do things randomly, such as simulating the roll of a dice. The `random()` function is a great way to do this. See **http://arduino.cc/en/reference/random** for more information.

What You Should See

This is similar to circuit number one, but instead of one LED, you should see all the LEDs blink. If they aren’t, make sure you have assembled the circuit correctly and verified and uploaded the code to your board, or see the troubleshooting section.

Real World Application

Scrolling marquee displays are generally used to spread short segments of important information. They are built out of many LEDs.

Troubleshooting

Some LEDs Fail to Light

It is easy to insert an LED backwards. Check the LEDs that aren’t working and ensure they are in the correct orientation.

Operating out of sequence

With eight wires it’s easy to cross a couple. Double check that the first LED is plugged into pin 2 and each pin thereafter.

Starting Fresh

It’s easy to accidentally misplace a wire without noticing. Pulling everything out and starting with a fresh slate is often easier than trying to track down the problem.

Experiment 5: Push Buttons

Introduction

Up until now, we’ve focused mostly on outputs. Now we’re going to go to the other end of spectrum and play around with inputs. In experiment 2, we used an analog input to read the potentiometer. In this circuit, we’ll be reading in one of the most common and simple inputs – a push button – by using a digital input. The way a push button works with your RedBoard or Arduino Uno R3 is that when the button is pushed, the voltage goes LOW. You RedBoard or Arduino Uno R3 reads this and reacts accordingly.

In this circuit, you will also use a pull-up resistor, which keeps the voltage HIGH when you’re not pressing the button.

Parts Needed

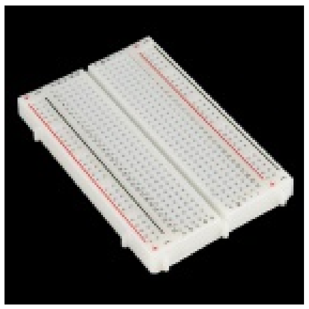
You will need the following parts:

- **1x** Breadboard
- **1x** RedBoard or Arduino Uno
- **1x** LED
- **1x** 330Ω Resistor
- **7x** Jumper Wires
- **2x** Push Buttons
- **2x** 10k Resistors

Didn’t get the SIK?

If you are following through this experiment and didn’t get the SIK, we suggest using these parts:

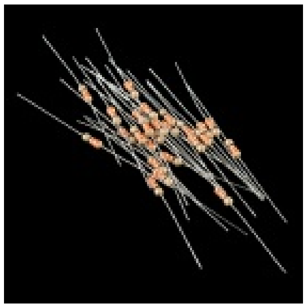
Jumper Wires Standard 7" M/M Pack of 30
◉ PRT-11026
$4.95



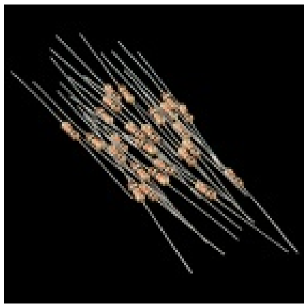
Breadboard - Self-Adhesive (White)
◉ PRT-12002
$4.95

LED - Assorted (20 pack)
◉ COM-12062
$2.95

Tactile Button Assortment
◉ COM-10302
$4.95



Resistor 330 Ohm 1/6 Watt PTH - 20 pack
◉ COM-11507
$0.95



Resistor 10K Ohm 1/6th Watt PTH - 20 pack
◉ COM-11508
$0.95

You will also need either a RedBoard or Arduino Uno R3.

SparkFun RedBoard - Programmed with Arduino
◉ DEV-12757
$19.95

Arduino Uno- R3 SMD
◉ DEV-11224
$29.95

Suggested Reading

Before continuing on with this tutorial, we recommend you be somewhat familiar with the concepts in these tutorials:

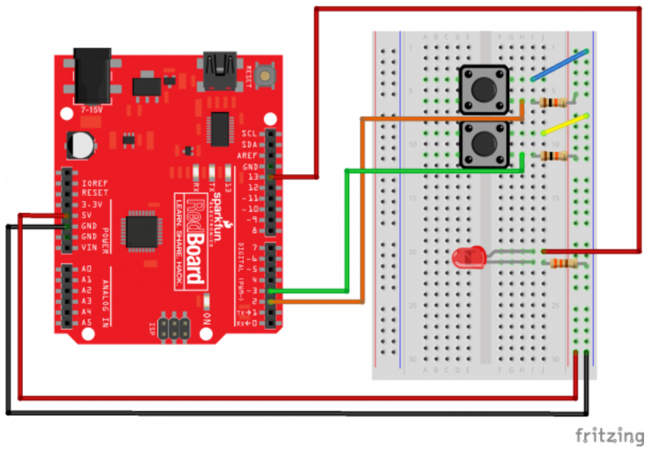
- Switch Basics
- Analog vs. Digital

Hardware Hookup

Ready to start hooking everything up? Check out the Fritzing diagram and hookup table below, to see how everything is connected.

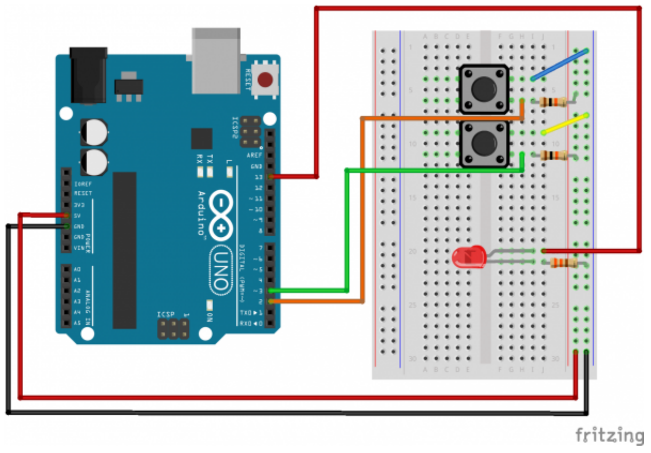
Polarized Components ⚠	Pay special attention to the component's markings indicating how to place it on the breadboard. Polarized components can only be connected to a circuit in one direction.

Fritzing Diagram for RedBoard



Having a hard time seeing the circuit? Click on the Fritzing diagram to see a bigger image.

Fritzing Diagram for Arduino



Having a hard time seeing the circuit? Click on the Fritzing diagram to see a bigger image.

Open the Sketch

Open Up the Arduino IDE software on your computer. Coding in the Arduino language will control your circuit. Open the code for Circuit 5 by accessing the "SIK Guide Code" you downloaded and placed into your "Examples" folder earlier.

To open the code go to: **File > examples > SIK Guide Code > Circuit_05**

You can also copy and paste the following code into the Arduino IDE. Hit upload, and see what happens!

```
/*
```

```
SparkFun Inventor's Kit
Example sketch 05

PUSH BUTTONS

  Use pushbuttons for digital input

  Previously we've used the analog pins for input, now we'll use
  the digital pins for input as well. Because digital pins only
  know about HIGH and LOW signals, they're perfect for interfacing
  to pushbuttons and switches that also only have "on" and "off"
  states.

  We'll connect one side of the pushbutton to GND, and the other
  side to a digital pin. When we press down on the pushbutton,
  the pin will be connected to GND, and therefore will be read
  as "LOW" by the Arduino.

  But wait - what happens when you're not pushing the button?
  In this state, the pin is disconnected from everything, which
  we call "floating". What will the pin read as then, HIGH or LOW?
  It's hard to say, because there's no solid connection to either
  5 Volts or GND. The pin could read as either one.

  To deal with this issue, we'll connect a small (10K, or 10,000 Ohm)
  resistance between the pin and 5 Volts. This "pullup" resistor
  will ensure that when you're NOT pushing the button, the pin will
  still have a weak connection to 5 Volts, and therefore read as
  HIGH.

  (Advanced: when you get used to pullup resistors and know when
  they're required, you can activate internal pullup resistors
  on the ATmega processor in the Arduino. See
  http://arduino.cc/en/Tutorial/DigitalPins for information.)

Hardware connections:

  Pushbuttons:

    Pushbuttons have two contacts that are connected if you're
    pushing the button, and disconnected if you're not.

    The pushbuttons we're using have four pins, but two pairs
    of these are connected together. The easiest way to hook up
    the pushbutton is to connect two wires to any opposite corners.

    Connect any pin on pushbutton 1 to ground (GND).
    Connect the opposite diagonal pin of the pushbutton to
    digital pin 2.

    Connect any pin on pushbutton 2 to ground (GND).
    Connect the opposite diagonal pin of the pushbutton to
    digital pin 3.

    Also connect 10K resistors (brown/black/red) between
    digital pins 2 and 3 and GND. These are called "pullup"
    resistors. They ensure that the input pin will be either
    5V (unpushed) or GND (pushed), and not somewhere in between.
    (Remember that unlike analog inputs, digital inputs are only
    HIGH or LOW.)

  LED:

    Most Arduinos, including the Uno, already have an LED
    and resistor connected to pin 13, so you don't need any
    additional circuitry.

    But if you'd like to connect a second LED to pin 13,

    Connect the positive side of your LED to Arduino digital pin 13
    Connect the negative side of your LED to a 330 Ohm resistor
    Connect the other side of the resistor to ground

This sketch was written by SparkFun Electronics,
with lots of help from the Arduino community.
This code is completely free for any use.
Visit http://learn.sparkfun.com/products/2 for SIK information.
Visit http://www.arduino.cc to learn about the Arduino.

Version 2.0 6/2012 MDG
*/



// First we'll set up constants for the pin numbers.
// This will make it easier to follow the code below.

const int button1Pin = 2;  // pushbutton 1 pin
const int button2Pin = 3;  // pushbutton 2 pin
const int ledPin =  13;    // LED pin


void setup()
{
  // Set up the pushbutton pins to be an input:
  pinMode(button1Pin, INPUT);
  pinMode(button2Pin, INPUT);

  // Set up the LED pin to be an output:
  pinMode(ledPin, OUTPUT);
}
```

```
void loop()
{
  int button1State, button2State;  // variables to hold the pushbutton states

  // Since a pushbutton has only two states (pushed or not pushed),
  // we've run them into digital inputs. To read an input, we'll
  // use the digitalRead() function. This function takes one
  // parameter, the pin number, and returns either HIGH (5V)
  // or LOW (GND).

  // Here we'll read the current pushbutton states into
  // two variables:

  button1State = digitalRead(button1Pin);
  button2State = digitalRead(button2Pin);

  // Remember that if the button is being pressed, it will be
  // connected to GND. If the button is not being pressed,
  // the pullup resistor will connect it to 5 Volts.

  // So the state will be LOW when it is being pressed,
  // and HIGH when it is not being pressed.

  // Now we'll use those states to control the LED.
  // Here's what we want to do:

  // "If either button is being pressed, light up the LED"
  // "But, if BOTH buttons are being pressed, DON'T light up the LED"

  // Let's translate that into computer code. The Arduino gives you
  // special logic functions to deal with true/false logic:

  // A == B means "EQUIVALENT". This is true if both sides are the same.
  // A && B means "AND". This is true if both sides are true.
  // A || B means "OR". This is true if either side is true.
  // !A means "NOT". This makes anything after it the opposite (true or false).

  // We can use these operators to translate the above sentences
  // into logic statements (Remember that LOW means the button is
  // being pressed)

  // "If either button is being pressed, light up the LED"
  // becomes:
  // if ((button1State == LOW) || (button2State == LOW)) // light the LED

  // "If BOTH buttons are being pressed, DON'T light up the LED"
  // becomes:
  // if ((button1State == LOW) && (button2State == LOW)) // don't light the LED

  // Now let's use the above functions to combine them into one statement:
```

```
  if (((button1State == LOW) || (button2State == LOW))  // if we're pushing button
 1 OR button 2
      && !                                               // AND we're NOT
      ((button1State == LOW) && (button2State == LOW))) // pushing button 1 AND bu
tton 2
                                                        // then...
  {
    digitalWrite(ledPin, HIGH);  // turn the LED on
  }
  else
  {
    digitalWrite(ledPin, LOW);  // turn the LED off
  }

  // As you can see, logic operators can be combined to make
  // complex decisions!

  // Don't forget that we use = when we're assigning a value,
  // and use == when we're testing a value for equivalence.
}
```

Code To Note

`pinMode(button2Pin, INPUT);`

The digital pins can be used as inputs as well as outputs. Before you do either, you need to tell the Galileo which direction you're going.

`button1State = digitalRead(button1Pin);`

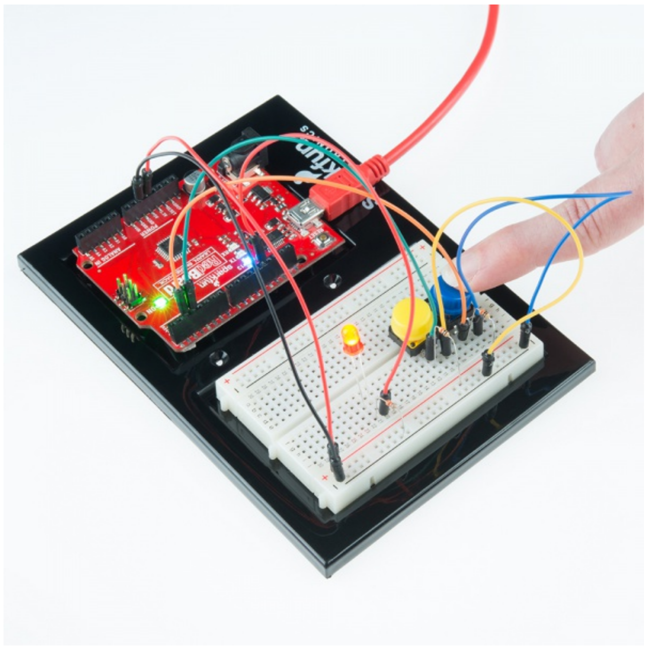
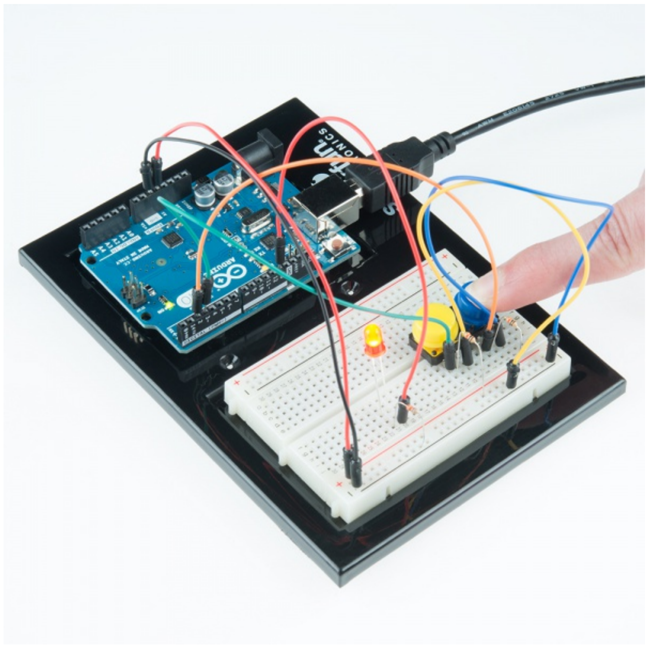
To read a digital input, you use the `digitalRead()` function. It will return HIGH if there's 5V present at the pin, or LOW if there's 0V present at the pin.

`if (button1State == LOW)`

Because we've connected the button to GND, it will read LOW when it's being pressed. Here we're using the "equivalence" operator ("==") to see if the button is being pressed.

What You Should See

You should see the LED turn on if you press either button, and off if you press both buttons. (See the code to find out why!) 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 section.

Real World Application

The buttons we used here are similar to the buttons in most video game controllers.

Troubleshooting

Light Not Turning On

The pushbutton is square, and because of this it is easy to put it in the wrong way. Give it a 90 degree twist and see if it starts working.

Underwhelmed

No worries, these circuits are all super stripped down to make playing with the components easy, but once you throw them together the sky is the limit.

Experiment 6: Reading a Photoresistor

Introduction

In experiment 2, you got to use a potentiometer, which varies resistance based on the twisting of a knob. In this circuit, you'll be using a photoresistor, which changes resistance based on how much light the sensor receives. Since the RedBoard and Arduino Uno R3 can't directly interpret resistance (rather, it reads voltage), we need to use a voltage divider to use our photoresistor. This voltage divider will output a high voltage when it is getting a lot of light and a low voltage when little or no light is present.

Parts Needed

You will need the following parts:

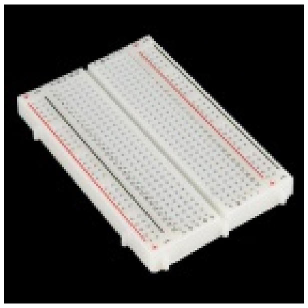
- **1x** Breadboard
- **1x** RedBoard or Arduino Uno
- **1x** LED
- **1x** 330Ω Resistor
- **6x** Jumper Wires

- **1x** Photoresistor
- **1x** 10k Resistor

Didn’t get the SIK?

If you are following through this experiment and didn’t get the SIK, we suggest using these parts:

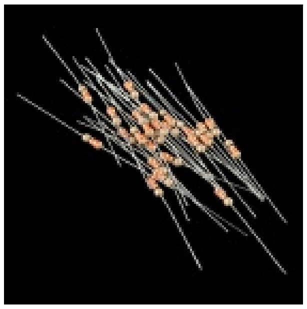
Jumper Wires Standard 7" M/M Pack of 30
◉ PRT-11026
$4.95



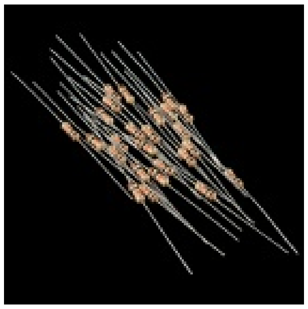
Breadboard - Self-Adhesive (White)
◉ PRT-12002
$4.95

Mini Photocell
◉ SEN-09088
$1.50

LED - Assorted (20 pack)
◉ COM-12062
$2.95



Resistor 330 Ohm 1/6 Watt PTH - 20 pack
◉ COM-11507
$0.95



Resistor 10K Ohm 1/6th Watt PTH - 20 pack
◉ COM-11508
$0.95

You will also need either a RedBoard or Arduino Uno R3.

SparkFun RedBoard - Programmed with Arduino
◉ DEV-12757
$19.95

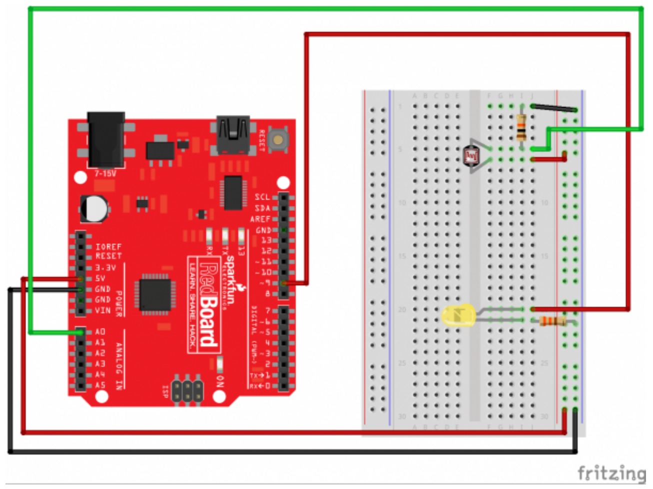
Arduino Uno- R3 SMD
◉ DEV-11224
$29.95

Hardware Hookup

Ready to start hooking everything up? Check out the Fritzing diagram below, to see how everything is connected.

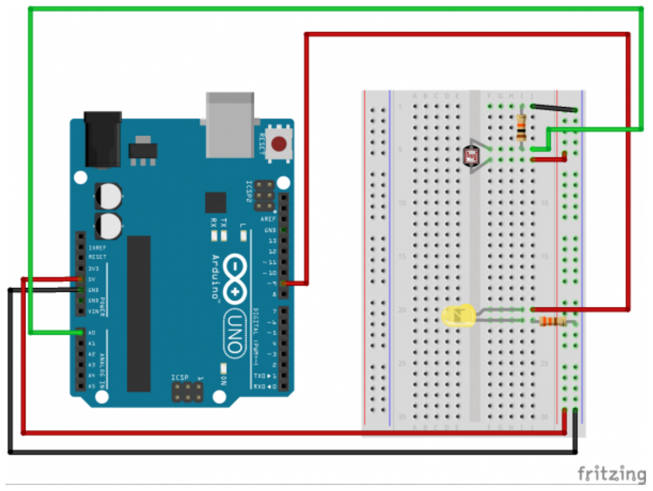
Polarized Components ⚠	Pay special attention to the component's markings indicating how to place it on the breadboard. Polarized components can only be connected to a circuit in one direction.

Fritzing Diagram for RedBoard



Having a hard time seeing the circuit? Click on the Fritzing diagram to see a bigger image.

Fritzing Diagram for Arduino



Having a hard time seeing the circuit? Click on the Fritzing diagram to see a bigger image.

Open the Sketch

Open Up the Arduino IDE software on your computer. Coding in the Arduino language will control your circuit. Open the code for Circuit 06 by accessing the “SIK Guide Code” you downloaded and placed into your “Examples” folder earlier.

To open the code go to: **File > examples > SIK Guide Code > Circuit_06**

You can also copy and paste the following code into the Arduino IDE. Hit upload, and see what happens!

```
/*
SparkFun Inventor's Kit
Example sketch 06

PHOTORESISTOR

  Use a photoresistor (light sensor) to control the brightness
  of a LED.

Hardware connections:

  Photo resistor:

    Connect one side of the photoresistor to 5 Volts (5V).
    Connect the other side of the photoresistor to ANALOG pin 0.
    Connect a 10K resistor between ANALOG pin 0 and GND.

    This creates a voltage divider, with the photoresistor one
    of the two resistors. The output of the voltage divider
    (connected to A0) will vary with the light level.

  LED:

    Connect the positive side (long leg) of the LED to
    digital pin 9. (To vary the brightness, this pin must
    support PWM, which is indicated by "~" or "PWM" on the
    Arduino itself.)

    Connect the negative side of the LED (short leg) to a
    330 Ohm resistor.

    Connect the other side of the resistor to GND.

This sketch was written by SparkFun Electronics,
with lots of help from the Arduino community.
This code is completely free for any use.
Visit http://learn.sparkfun.com/products/2 for SIK information.
Visit http://www.arduino.cc to learn about the Arduino.

Version 2.0 6/2012 MDG
*/


// As usual, we'll create constants to name the pins we're using.
// This will make it easier to follow the code below.

const int sensorPin = 0;
const int ledPin = 9;

// We'll also set up some global variables for the light level:

int lightLevel, high = 0, low = 1023;


void setup()
{
  // We'll set up the LED pin to be an output.
  // (We don't need to do anything special to use the analog input.)

  pinMode(ledPin, OUTPUT);
}


void loop()
{
  // Just as we've done in the past, we'll use the analogRead()
  // function to measure the voltage coming from the photoresistor
  // resistor pair. This number can range between 0 (0 Volts) and
  // 1023 (5 Volts), but this circuit will have a smaller range
  // between dark and light.

  lightLevel = analogRead(sensorPin);

  // We now want to use this number to control the brightness of
  // the LED. But we have a problem: the analogRead() function
  // returns values between 0 and 1023, and the analogWrite()
  // function wants values from 0 to 255.
  // We can solve this by using two handy functions called map()
  // and constrain(). Map will change one range of values into
  // another range. If we tell map() our "from" range is 0-1023,
  // and our "to" range is 0-255, map() will squeeze the larger
  // range into the smaller. (It can do this for any two ranges.)

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

  // Because map() could still return numbers outside the "to"
  // range, (if they're outside the "from" range), we'll also use
  // a function called constrain() that will "clip" numbers into
  // a given range. If the number is above the range, it will reset
  // it to be the highest number in the range. If the number is
  // below the range, it will reset it to the lowest number.
  // If the number is within the range, it will stay the same.

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

  // Here's one last thing to think about. The circuit we made
  // won't have a range all the way from 0 to 5 Volts. It will
  // be a smaller range, such as 300 (dark) to 800 (light).
  // If we just pass this number directly to map(), the LED will
  // change brightness, but it will never be completely off or
  // completely on.

  // You can fix this two ways, each of which we'll show
  // in the functions below. Uncomment ONE of them to
  // try it out:

  manualTune();  // manually change the range from light to dark

  //autoTune();  // have the Arduino do the work for us!

  // The above functions will alter lightLevel to be cover the
  // range from full-on to full-off. Now we can adjust the
  // brightness of the LED:

  analogWrite(ledPin, lightLevel);

  // The above statement will brighten the LED along with the
  // light level. To do the opposite, replace "lightLevel" in the
  // above analogWrite() statement with "255-lightLevel".
  // Now you've created a night-light!
}


void manualTune()
{
  // As we mentioned above, the light-sensing circuit we built
  // won't have a range all the way from 0 to 1023. It will likely
  // be more like 300 (dark) to 800 (light). If you run this sketch
  // as-is, the LED won't fully turn off, even in the dark.

  // You can accommodate the reduced range by manually
  // tweaking the "from" range numbers in the map() function.
  // Here we're using the full range of 0 to 1023.
  // Try manually changing this to a smaller range (300 to 800
  // is a good guess), and try it out again. If the LED doesn't
  // go completely out, make the low number larger. If the LED
  // is always too bright, make the high number smaller.

  // Remember you're JUST changing the 0, 1023 in the line below!

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

  // Now we'll return to the main loop(), and send lightLevel
  // to the LED.
}


void autoTune()
{
  // As we mentioned above, the light-sensing circuit we built
  // won't have a range all the way from 0 to 1023. It will likely
  // be more like 300 (dark) to 800 (light).

  // In the manualTune() function above, you need to repeatedly
  // change the values and try the program again until it works.
  // But why should you have to do that work? You've got a
  // computer in your hands that can figure things out for itself!

  // In this function, the Arduino will keep track of the highest
  // and lowest values that we're reading from analogRead().

  // If you look at the top of the sketch, you'll see that we've
  // initialized "low" to be 1023. We'll save anything we read
  // that's lower than that:

  if (lightLevel < low)
  {
    low = lightLevel;
  }

  // We also initialized "high" to be 0. We'll save anything
  // we read that's higher than that:

  if (lightLevel > high)
  {
    high = lightLevel;
  }
  // Once we have the highest and lowest values, we can stick them
  // directly into the map() function. No manual tweaking needed!

  // One trick we'll do is to add a small offset to low and high,
  // to ensure that the LED is fully-off and fully-on at the limits
  // (otherwise it might flicker a little bit).

  lightLevel = map(lightLevel, low+30, high-30, 0, 255);
  lightLevel = constrain(lightLevel, 0, 255);

  // Now we'll return to the main loop(), and send lightLevel
  // to the LED.
}
```

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 Arduino's map reference page for more info.

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

Parameters

constrain(x, a, b)

x: the number to constrain, all data types

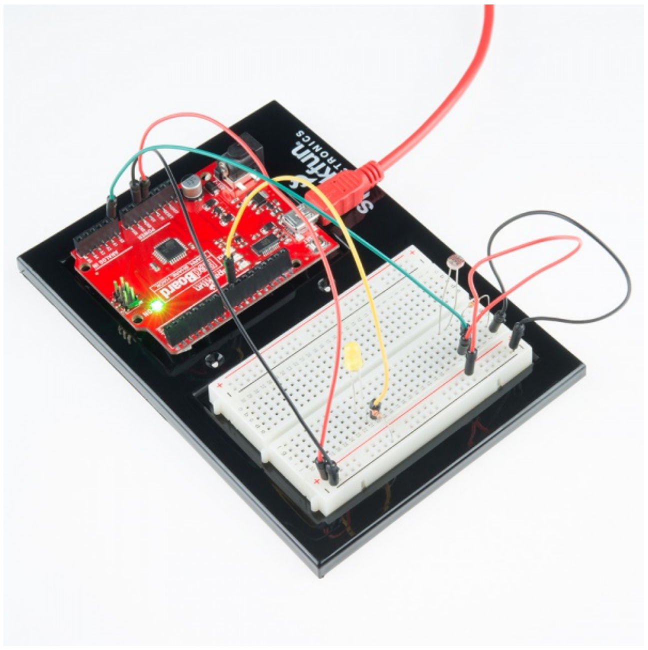
a: the lower end of the range, all data types

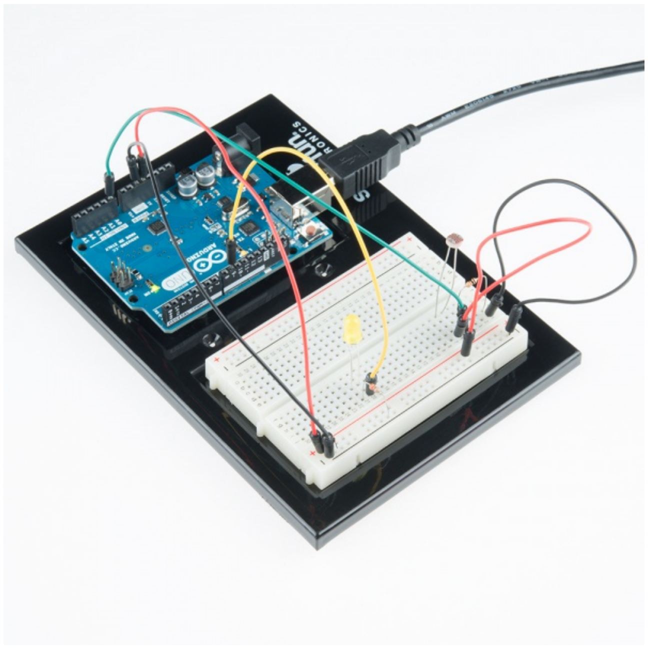
b: the upper end of the range, all data types

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 Arduino's constrain reference page 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 section.





Real World Application

Some street lamps as well as solar walkway lights use photoresistors to detect the absence of the sun and turn on the lights.

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 photoresistor 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.

Experiment 7: Reading a Temperature Sensor

Introduction

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 or Arduino Uno R3, and use the Arduino IDE's serial monitor to display the temperature.

Parts Needed

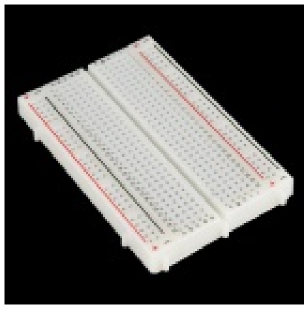
You will need the following parts:

- **1x** Breadboard
- **1x** RedBoard or Arduino Uno
- **5x** Jumper Wires
- **1x** Temperature Sensor

Didn't get the SIK?

If you are following through this experiment and didn't get the SIK, we suggest using these parts:

Jumper Wires Standard 7" M/M Pack of 30
◉ PRT-11026
$4.95



Breadboard - Self-Adhesive (White)
◉ PRT-12002
$4.95

Temperature Sensor - TMP36
○ SEN-10988
$1.50

You will also need either a RedBoard or Arduino Uno R3.

SparkFun RedBoard - Programmed with Arduino
◉ DEV-12757
$19.95

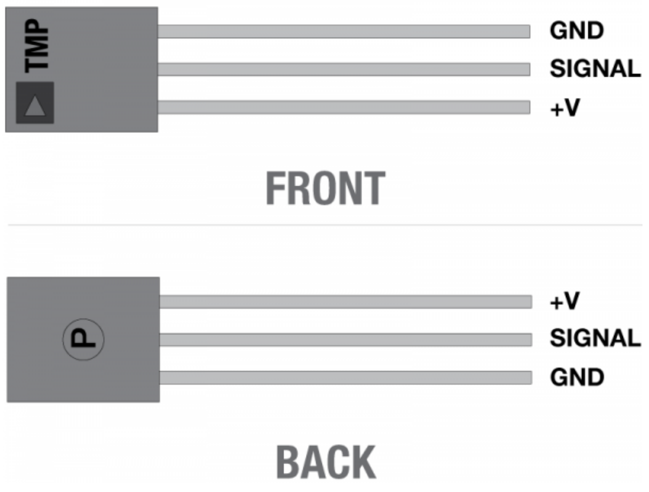
Arduino Uno- R3 SMD
◉ DEV-11224
$29.95

Hardware Hookup

Ready to start hooking everything up? Check out the Fritzing diagram below, to see how everything is connected.

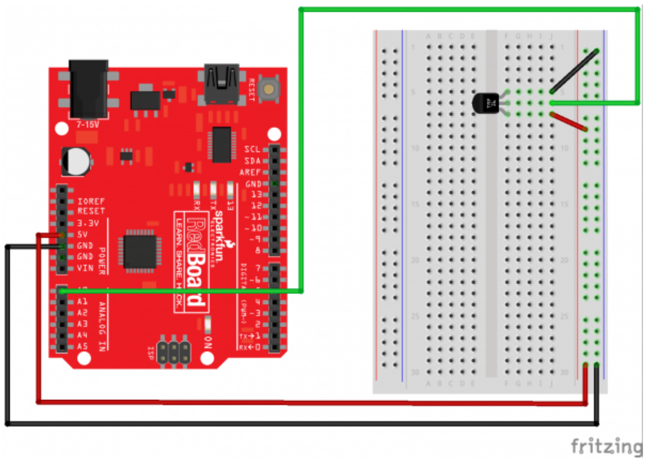
Polarized Components ⚠	Pay special attention to the component's markings indicating how to place it on the breadboard. Polarized components can only be connected to a circuit in one direction.

Please note: The temperature sensor can only be connected to a circuit in one direction. See below for the pin outs of the temperature sensor - TMP36



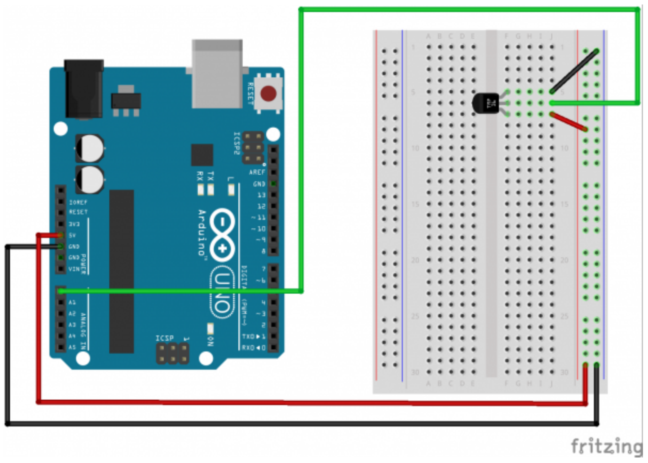
<-

Fritzing Diagram for RedBoard



Having a hard time seeing the circuit? Click on the Fritzing diagram to see a bigger image.

Fritzing Diagram for Arduino



Having a hard time seeing the circuit? Click on the Fritzing diagram to see a bigger image.

Open the Sketch

Open Up the Arduino IDE software on your computer. Coding in the Arduino language will control your circuit. Open the code for Circuit 7 by accessing the “SIK Guide Code” you downloaded and placed into your “Examples” folder earlier.

To open the code go to: **File > examples > SIK Guide Code > Circuit_07**

You can also copy and paste the following code into the Arduino IDE. Hit upload, and see what happens!

```
/*
SparkFun Inventor's Kit
Example sketch 07

TEMPERATURE SENSOR

  Use the "serial monitor" window to read a temperature sensor.

  The TMP36 is an easy-to-use temperature sensor that outputs
  a voltage that's proportional to the ambient temperature.
  You can use it for all kinds of automation tasks where you'd
  like to know or control the temperature of something.

  More information on the sensor is available in the datasheet:
  http://dlnmh9ip6v2uc.cloudfront.net/datasheets/Sensors/Temp/TMP35_36_37.pdf

  Even more exciting, we'll start using the Arduino's serial port
  to send data back to your main computer! Up until now, we've
  been limited to using simple LEDs for output. We'll see that
  the Arduino can also easily output all kinds of text and data.

Hardware connections:

  Be careful when installing the temperature sensor, as it is
  almost identical to the transistors! The one you want has
  a triangle logo and "TMP" in very tiny letters. The
  ones you DON'T want will have "222" on them.

  When looking at the flat side of the temperature sensor
  with the pins down, from left to right the pins are:
  5V, SIGNAL, and GND.

  Connect the 5V pin to 5 Volts (5V).
  Connect the SIGNAL pin to ANALOG pin 0.
  Connect the GND pin to ground (GND).

This sketch was written by SparkFun Electronics,
with lots of help from the Arduino community.
This code is completely free for any use.
Visit http://learn.sparkfun.com/products/2 for SIK information.
Visit http://www.arduino.cc to learn about the Arduino.

Version 2.0 6/2012 MDG
*/

// We'll use analog input 0 to measure the temperature sensor's
// signal pin.

const int temperaturePin = 0;


void setup()
{
  // In this sketch, we'll use the Arduino's serial port
  // to send text back to the main computer. For both sides to
  // communicate properly, they need to be set to the same speed.
  // We use the Serial.begin() function to initialize the port
  // and set the communications speed.

  // The speed is measured in bits per second, also known as
  // "baud rate". 9600 is a very commonly used baud rate,
  // and will transfer about 10 characters per second.
  Serial.begin(9600);
}


void loop()
{
  // Up to now we've only used integer ("int") values in our
  // sketches. Integers are always whole numbers (0, 1, 23, etc.).
  // In this sketch, we'll use floating-point values ("float").
  // Floats can be fractional numbers such as 1.42, 2523.43121, etc.

  // We'll declare three floating-point variables
  // (We can declare multiple variables of the same type on one line:)

  float voltage, degreesC, degreesF;

  // First we'll measure the voltage at the analog pin. Normally
  // we'd use analogRead(), which returns a number from 0 to 1023.
  // Here we've written a function (further down) called
  // getVoltage() that returns the true voltage (0 to 5 Volts)
  // present on an analog input pin.

  voltage = getVoltage(temperaturePin);

  // Now we'll convert the voltage to degrees Celsius.
  // This formula comes from the temperature sensor datasheet:

  degreesC = (voltage - 0.5) * 100.0;

  // While we're at it, let's convert degrees Celsius to Fahrenheit.
  // This is the classic C to F conversion formula:

  degreesF = degreesC * (9.0/5.0) + 32.0;

  // Now we'll use the serial port to print these values
  // to the serial monitor!

  // To open the serial monitor window, upload your code,
  // then click the "magnifying glass" button at the right edge
  // of the Arduino IDE toolbar. The serial monitor window
  // will open.

  // (NOTE: remember we said that the communication speed
  // must be the same on both sides. Ensure that the baud rate
  // control at the bottom of the window is set to 9600. If it
  // isn't, change it to 9600.)

  // Also note that every time you upload a new sketch to the
  // Arduino, the serial monitor window will close. It does this
  // because the serial port is also used to upload code!
  // When the upload is complete, you can re-open the serial
  // monitor window.

  // To send data from the Arduino to the serial monitor window,
  // we use the Serial.print() function. You can print variables
  // or text (within quotes).

  Serial.print("voltage: ");
  Serial.print(voltage);
  Serial.print("  deg C: ");
  Serial.print(degreesC);
  Serial.print("  deg F: ");
  Serial.println(degreesF);

  // These statements will print lines of data like this:
  // "voltage: 0.73 deg C: 22.75 deg F: 72.96"

  // Note that all of the above statements are "print", except
  // for the last one, which is "println". "Print" will output
  // text to the SAME LINE, similar to building a sentence
  // out of words. "Println" will insert a "carriage return"
  // character at the end of whatever it prints, moving down
  // to the NEXT line.

  delay(1000); // repeat once per second (change as you wish!)
}


float getVoltage(int pin)
{
  // This function has one input parameter, the analog pin number
  // to read. You might notice that this function does not have
  // "void" in front of it; this is because it returns a floating-
  // point value, which is the true voltage on that pin (0 to 5V).

  // You can write your own functions that take in parameters
  // and return values. Here's how:

    // To take in parameters, put their type and name in the
    // parenthesis after the function name (see above). You can
    // have multiple parameters, separated with commas.

    // To return a value, put the type BEFORE the function name
    // (see "float", above), and use a return() statement in your code
    // to actually return the value (see below).

    // If you don't need to get any parameters, you can just put
    // "()" after the function name.

    // If you don't need to return a value, just write "void" before
    // the function name.
  // Here's the return statement for this function. We're doing
  // all the math we need to do within this statement:

  return (analogRead(pin) * 0.004882814);

  // This equation converts the 0 to 1023 value that analogRead()
  // returns, into a 0.0 to 5.0 value that is the true voltage
  // being read at that pin.
}

// Other things to try with this code:

//   Turn on an LED if the temperature is above or below a value.

//   Read that threshold value from a potentiometer - now you've
//   created a thermostat!
```

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 section.

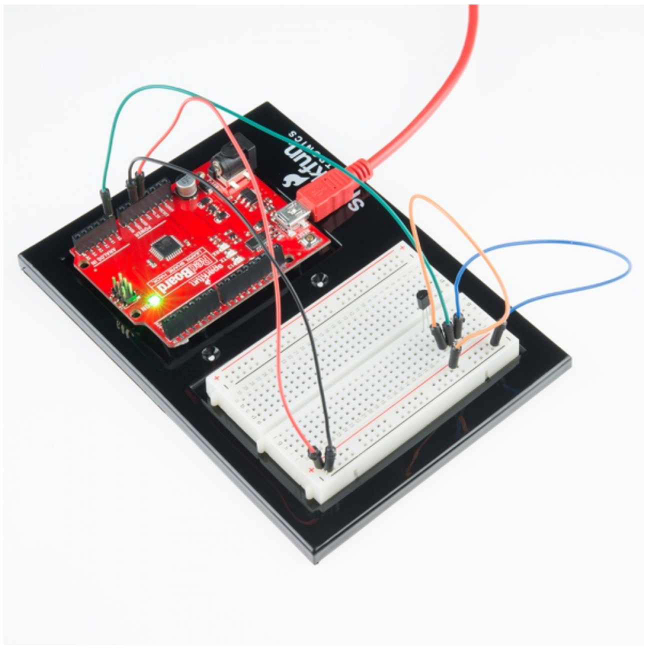
Example of what you should see in the Arduino IDE's serial monitor:

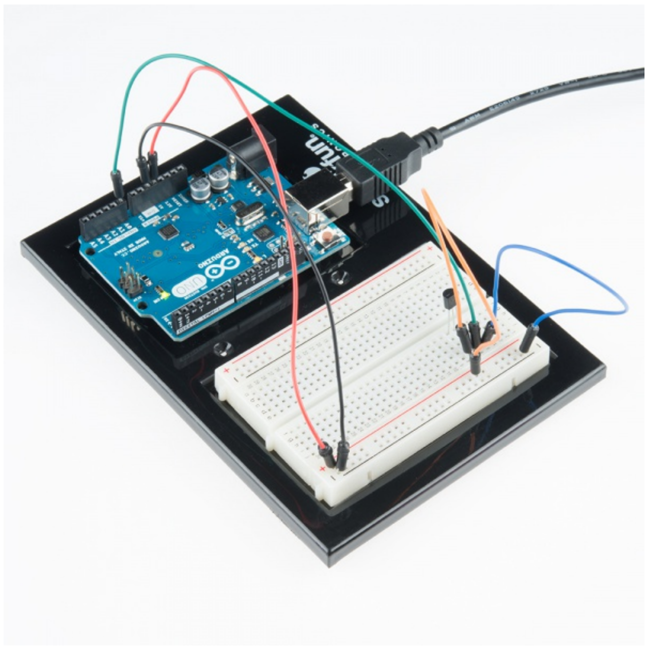
```
voltage: 0.73 deg C: 23.24 deg F: 73.84

voltage: 0.73 deg C: 23.24 deg F: 73.84

voltage: 0.73 deg C: 22.75 deg F: 72.96

voltage: 0.73 deg C: 23.24 deg F: 73.84
```

```
voltage: 0.73 deg C: 23.24 deg F: 73.84

voltage: 0.73 deg C: 23.24 deg F: 73.84

voltage: 0.73 deg C: 22.75 deg F: 72.96

voltage: 0.73 deg C: 23.24 deg F: 73.84

voltage: 0.73 deg C: 22.75 deg F: 72.96

voltage: 0.73 deg C: 22.75 deg F: 72.96

voltage: 0.73 deg C: 23.24 deg F: 73.84

voltage: 0.73 deg C: 22.75 deg F: 72.96

voltage: 0.73 deg C: 23.24 deg F: 73.84
```

Real World Application

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





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.

Experiment 8: Driving a Servo Motor

Introduction

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 Needed

You will need the following parts:

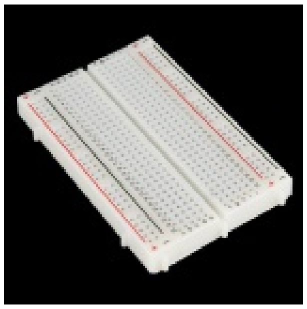
- **1x** Breadboard
- **1x** RedBoard or Arduino Uno
- **1x** Servo
- **8x** Jumper Wires

Didn't get the SIK?

If you are following through this experiment and didn't get the SIK, we suggest using these parts:

Jumper Wires Standard 7" M/M Pack of 30
◉ PRT-11026
$4.95

Servo - Generic (Sub-Micro Size)
○ ROB-09065
$8.96


Breadboard - Self-Adhesive (White)
◉ PRT-12002
$4.95

You will also need either a RedBoard or Arduino Uno R3.

SparkFun RedBoard - Programmed with Arduino
◉ DEV-12757
$19.95

Arduino Uno- R3 SMD
◉ DEV-11224
$29.95

Suggested Reading

Before continuing on with this experiment, we recommend you be familiar with the concepts in the following tutorial:

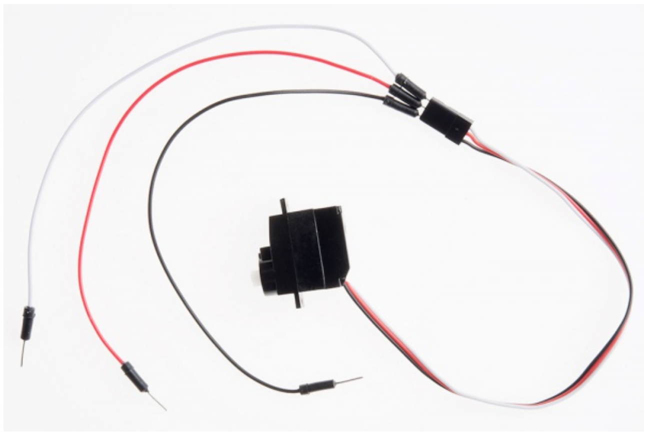
- Pulse-width Modulation

Hardware Hookup

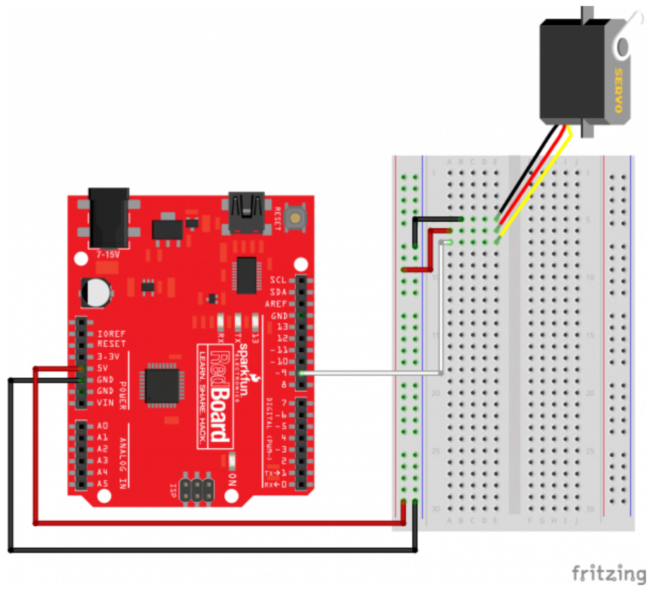
Ready to start hooking everything up? Check out the Fritzing diagram below, to see how everything is connected.

Polarized Components ⚠	Pay special attention to the component's markings indicating how to place it on the breadboard. Polarized components can only be connected to a circuit in one direction.

Connect 3x jumper wires to the female 3 pin header on the servo. This will make it easier to breadboard the servo.

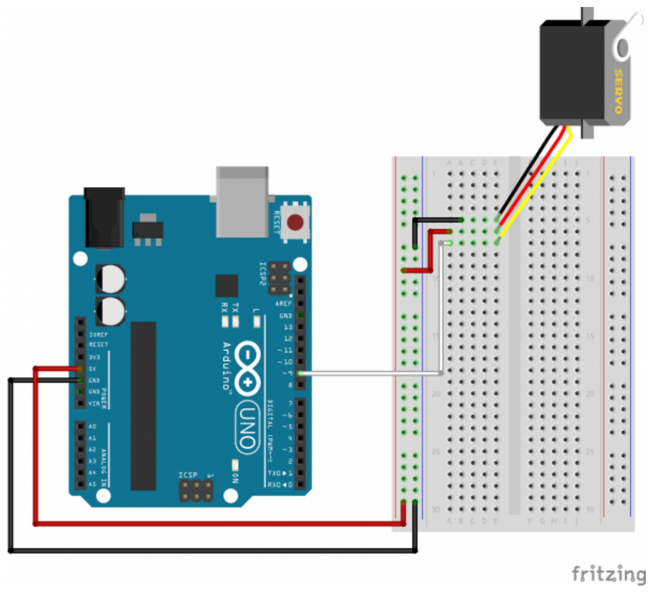


Fritzing Diagram for RedBoard



Having a hard time seeing the circuit? Click on the Fritzing diagram to see a bigger image.

Fritzing Diagram for Arduino



Having a hard time seeing the circuit? Click on the Fritzing diagram to see a bigger image.

Open the Sketch

Open Up the Arduino IDE software on your computer. Coding in the Arduino language will control your circuit. Open the code for Circuit 08 by accessing the “SIK Guide Code” you downloaded and placed into your “Examples” folder earlier.

To open the code go to: **File > examples > SIK Guide Code > Circuit_08**

You can also copy and paste the following code into the Arduino IDE. Hit upload, and see what happens!

```
/*
SparkFun Inventor's Kit
Example sketch 08

SINGLE SERVO
```

```
  Sweep a servo back and forth through its full range of motion.

  A "servo", short for servomotor, is a motor that includes
  feedback circuitry that allows it to be commanded to move to
  specific positions. This one is very small, but larger servos
  are used extensively in robotics to control mechanical arms,
  hands, etc. You could use it to make a (tiny) robot arm,
  aircraft control surface, or anywhere something needs to be
  moved to specific positions.

Hardware connections:

  The servo has a cable attached to it with three wires.
  Because the cable ends in a socket, you can use jumper wires
  to connect between the Arduino and the servo. Just plug the
  jumper wires directly into the socket.

  Connect the RED wire (power) to 5 Volts (5V)
  Connect the WHITE wire (signal) to digital pin 9
  Connect the BLACK wire (ground) to ground (GND)

  Note that servos can use a lot of power, which can cause your
  Arduino to reset or behave erratically. If you're using large
  servos or many of them, it's best to provide them with their
  own separate 5V supply. See this Arduino Forum thread for info:
  http://www.arduino.cc/cgi-bin/yabb2/YaBB.pl?num=1239464763

This sketch was written by SparkFun Electronics,
with lots of help from the Arduino community.
This code is completely free for any use.
Visit http://learn.sparkfun.com/products/2 for SIK information.
Visit http://www.arduino.cc to learn about the Arduino.

Version 2.0 6/2012 MDG
*/



// If we had to write a sketch to control a servo from scratch,
// it would be a lot of work. Fortunately, others have done the
// hard work for you. We're going to include a "library"
// that has the functions needed to drive servos.

// A library is an set of additional functions you can add to
// your sketch. Numerous libraries are available for many uses,
// see http://arduino.cc/en/Reference/Libraries for information
// on the standard libraries, and Google for others. When you're
// using a new part, chances are someone has written a library
// for it.

#include <Servo.h>  // servo library
```

```
// Once you "include" a library, you'll have access to those
// functions. You can find a list of the functions in the servo
// library at: http://arduino.cc/en/Reference/Servo
// Most libraries also have example sketches you can load from
// the "file/examples" menu.

// Now we'll create a servo "object", called myservo. You should
// create one of these for each servo you want to control.
// You can control a maximum of twelve servos on the Uno
// using this library. (Other servo libraries may let you
// control more). Note that this library disables PWM on
// pins 9 and 10!

Servo servo1;  // servo control object


void setup()
{
  // We'll now "attach" the servo1 object to digital pin 9.
  // If you want to control more than one servo, attach more
  // servo objects to the desired pins (must be digital).

  // Attach tells the Arduino to begin sending control signals
  // to the servo. Servos require a continuous stream of control
  // signals, even if you're not currently moving them.
  // While the servo is being controlled, it will hold its
  // current position with some force. If you ever want to
  // release the servo (allowing it to be turned by hand),
  // you can call servo1.detach().

  servo1.attach(9);
}


void loop()
{
  int position;

  // To control a servo, you give it the angle you'd like it
  // to turn to. Servos cannot turn a full 360 degrees, but you
  // can tell it to move anywhere between 0 and 180 degrees.

  // Change position at full speed:

  servo1.write(90);    // Tell servo to go to 90 degrees

  delay(1000);         // Pause to get it time to move

  servo1.write(180);   // Tell servo to go to 180 degrees
  delay(1000);          // Pause to get it time to move

  servo1.write(0);      // Tell servo to go to 0 degrees

  delay(1000);          // Pause to get it time to move

  // Change position at a slower speed:

  // To slow down the servo's motion, we'll use a for() loop
  // to give it a bunch of intermediate positions, with 20ms
  // delays between them. You can change the step size to make
  // the servo slow down or speed up. Note that the servo can't
  // move faster than its full speed, and you won't be able
  // to update it any faster than every 20ms.

  // Tell servo to go to 180 degrees, stepping by two degrees

  for(position = 0; position < 180; position += 2)
  {
    servo1.write(position);  // Move to next position
    delay(20);               // Short pause to allow it to move
  }

  // Tell servo to go to 0 degrees, stepping by one degree

  for(position = 180; position >= 0; position -= 1)
  {
    servo1.write(position);  // Move to next position
    delay(20);               // Short pause to allow it to move
  }
}
```

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;`

`servo1.attach(9);`

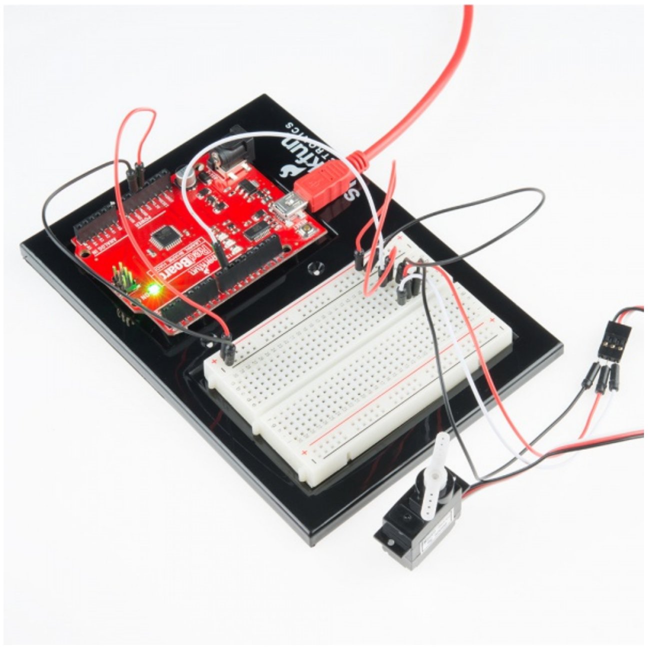
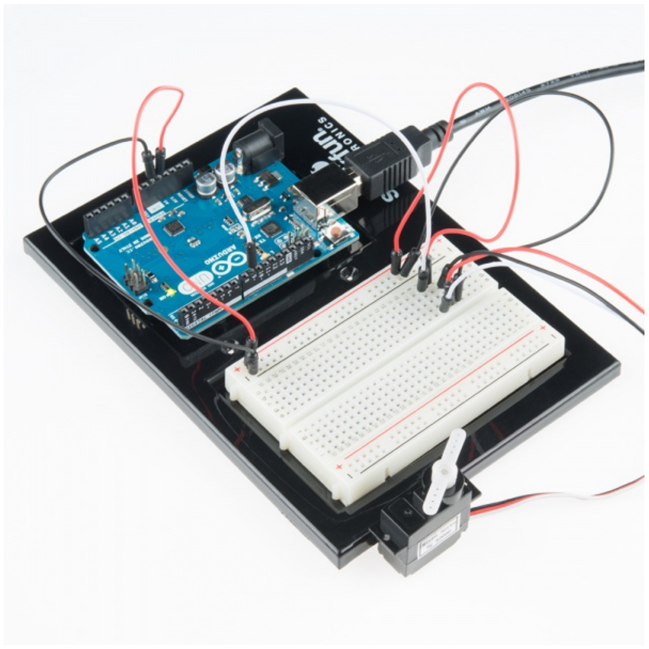
The servo library adds new commands that let you control a servo. To prepare the Galileo to control a servo, you must first create a Servo “object” for each servo (here we’ve named it “servo1”), and then “attach” it to a digital pin (here we’re using pin 9).

`servo1.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 section.

Real World Application

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

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 black 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 or Arduino Uno R3, the power supply you are using is not quite up to the challenge. Using a wall adapter instead of USB should solve this problem.

Experiment 9: Using a Flex Sensor

Introduction

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.

Parts Needed

You will need the following parts:

- **1x** Breadboard
- **1x** RedBoard or Arduino Uno
- **1x** Flex Sensor
- **1x** Servo
- **1x** 10k resistor
- **11x** Jumper Wires

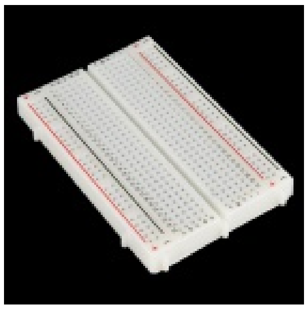
Didn’t get the SIK?

If you are following through this experiment and didn’t get the SIK, we suggest using these parts:

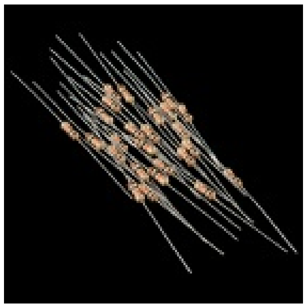
Flex Sensor 2.2"
◉ SEN-10264
$7.96

Jumper Wires Standard 7" M/M Pack of 30
◉ PRT-11026
$4.95

Servo - Generic (Sub-Micro Size)
○ ROB-09065
$8.96



Breadboard - Self-Adhesive (White)
◉ PRT-12002
$4.95



Resistor 10K Ohm 1/6th Watt PTH - 20 pack
◉ COM-11508
$0.95

You will also need either a RedBoard or Arduino Uno R3.

SparkFun RedBoard - Programmed with Arduino
◉ DEV-12757
$19.95

Arduino Uno- R3 SMD
◉ DEV-11224
$29.95

Suggested Reading

Before continuing on with this experiment, we recommend you be familiar with the concepts in the following tutorial:

- Voltage Dividers

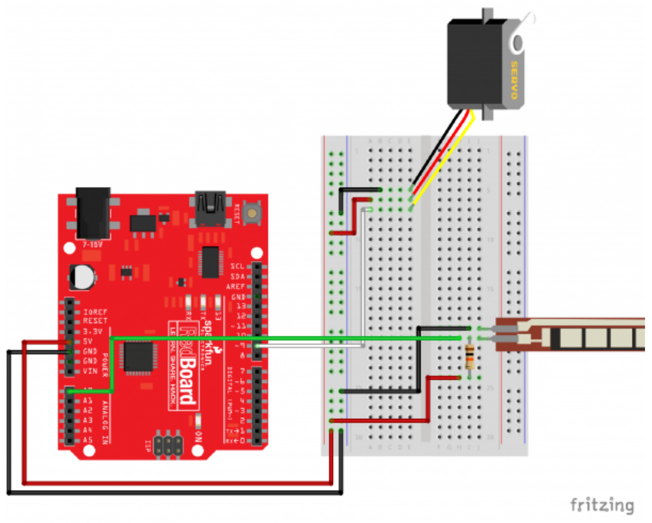
Hardware Hookup

Ready to start hooking everything up? Check out the Fritzing diagram below, to see how everything is connected.

Polarized Components ⚠	Pay special attention to the component's markings indicating how to place it on the breadboard. Polarized components can only be connected to a circuit in one direction.

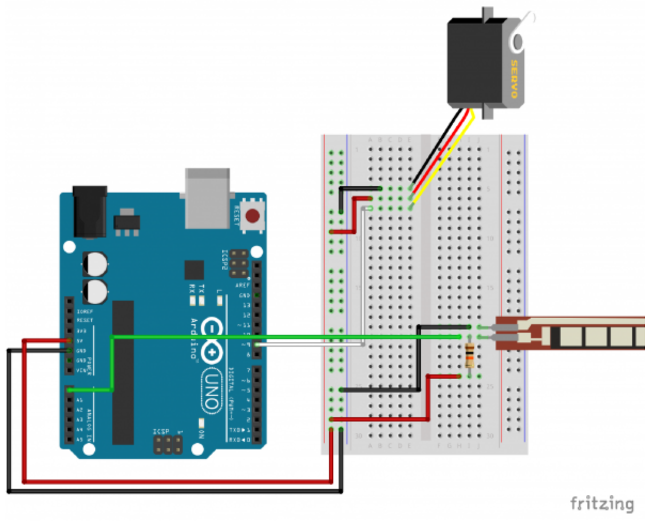
Connect 3x jumper wires to the female 3 pin header on the servo. This will make it easier to breadboard the servo.

Fritzing Diagram for RedBoard



Having a hard time seeing the circuit? Click on the Fritzing diagram to see a bigger image.

Fritzing Diagram for Arduino



Having a hard time seeing the circuit? Click on the Fritzing diagram to see a bigger image.

Open the Sketch

Open Up the Arduino IDE software on your computer. Coding in the Arduino language will control your circuit. Open the code for Circuit 9 by accessing the “SIK Guide Code” you downloaded and placed into your “Examples” folder earlier.

To open the code go to: **File > examples > SIK Guide Code > Circuit_09**

You can also copy and paste the following code into the Arduino IDE. Hit upload, and see what happens!

```
/*
SparkFun Inventor's Kit Galileo
Example sketch 09

FLEX SENSOR

  Use the "flex sensor" to change the position of a servo
  In the previous sketch, we learned how to command a servo to
  mode to different positions. In this sketch, we'll introduce
  a new sensor, and use it to control the servo.

  A flex sensor is a plastic strip with a conductive coating.
  When the strip is straight, the coating will be a certain
  resistance. When the strip is bent, the particles in the coating
  get further apart, increasing the resistance. You can use this
  sensor to sense finger movement in gloves, door hinges, stuffed
  animals, etc. See http://www.sparkfun.com/tutorials/270 for
  more information.

Hardware connections:

  Flex sensor:

    The flex sensor is the plastic strip with black stripes.
    It senses bending away from the striped side.

    The flex sensor has two pins, and since it's a resistor,
    the pins are interchangable.

    Connect one of the pins to ANALOG IN pin 0 on the Arduino.
    Connect the same pin, through a 10K Ohm resistor (brown
    black orange) to GND.
    Connect the other pin to 5V.

  Servo:

    The servo has a cable attached to it with three wires.
    Because the cable ends in a socket, you can use jumper wires
    to connect between the Arduino and the servo. Just plug the
    jumper wires directly into the socket.

    Connect the RED wire (power) to 5 Volts (5V)
    Connect the WHITE wire (signal) to digital pin 9
    Connect the BLACK wire (ground) to ground (GND)

    Note that servos can use a lot of power, which can cause your
    Arduino to reset or behave erratically. If you're using large
    servos or many of them, it's best to provide them with their
    own separate 5V supply. See this Arduino Forum thread for info:
    http://www.arduino.cc/cgi-bin/yabb2/YaBB.pl?num=1239464763

This sketch was written by SparkFun Electronics,
with lots of help from the Arduino community.
This code is completely free for any use.
Visit http://learn.sparkfun.com/products/2 for SIK information.
Visit http://www.arduino.cc to learn about the Arduino.

Version 2.0 6/2012 MDG
```

```
*/


// Include the servo library to add servo-control functions:

#include <Servo.h>

// Create a servo "object", called servo1. Each servo object
// controls one servo (you can have a maximum of 12):

Servo servo1;

// Define the analog input pin to measure flex sensor position:

const int flexpin = 0;


void setup()
{
  // Use the serial monitor window to help debug our sketch:

  Serial.begin(9600);

  // Enable control of a servo on pin 9:

  servo1.attach(9);
}


void loop()
{
  int flexposition;    // Input value from the analog pin.
  int servoposition;   // Output value to the servo.

  // Read the position of the flex sensor (0 to 1023):

  flexposition = analogRead(flexpin);

  // Because the voltage divider circuit only returns a portion
  // of the 0-1023 range of analogRead(), we'll map() that range
  // to the servo's range of 0 to 180 degrees. The flex sensors
  // we use are usually in the 600-900 range:

  servoposition = map(flexposition, 600, 900, 0, 180);
  servoposition = constrain(servoposition, 0, 180);

  // Now we'll command the servo to move to that position:

  servo1.write(servoposition);

  // Because every flex sensor has a slightly different resistance,
  // the 600-900 range may not exactly cover the flex sensor's
  // output. To help tune our program, we'll use the serial port to
  // print out our values to the serial monitor window:

  Serial.print("sensor: ");
  Serial.print(flexposition);
  Serial.print("  servo: ");
  Serial.println(servoposition);

  // Note that all of the above lines are "print" except for the
  // last line which is "println". This puts everything on the
  // same line, then sends a final carriage return to move to
  // the next line.

  // After you upload the sketch, turn on the serial monitor
  // (the magnifying-glass icon to the right of the icon bar).
  // You'll be able to see the sensor values. Bend the flex sensor
  // and note its minimum and maximum values. If you replace the
  // 600 and 900 in the map() function above, you'll exactly match
  // the flex sensor's range with the servo's range.

  delay(20);  // wait 20ms between servo updates
}
```

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.

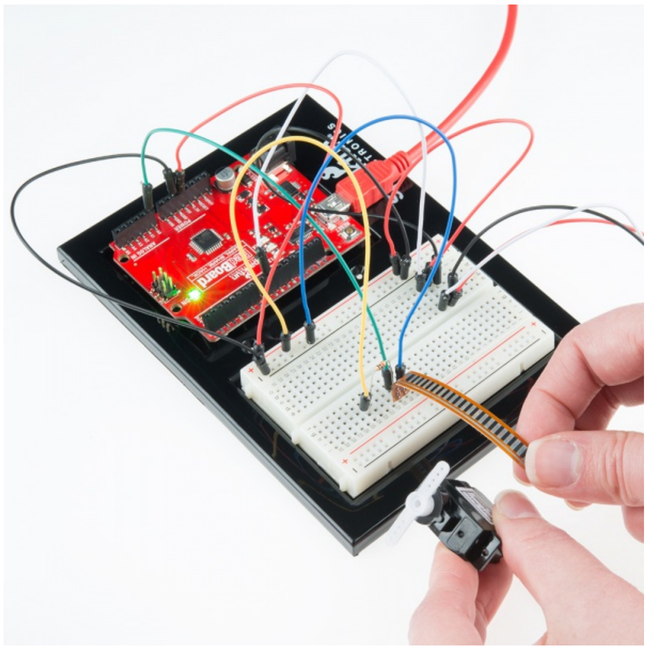
`servoposition = constrain(servoposition, 0, 180);`

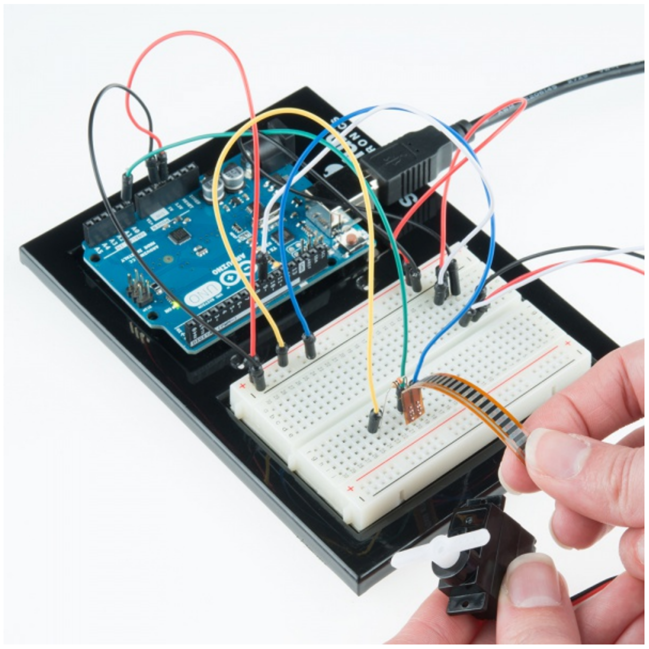
constrain(x, a, b)

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 section.





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.

Troublshooting

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.

Experiment 10: Reading a Soft Potentiometer

Introduction

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 Needed

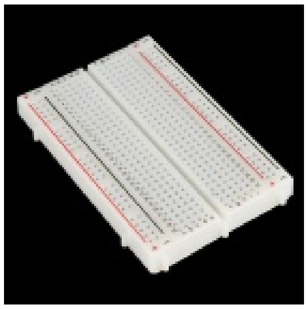
You will need the following parts:

- **1x** Breadboard
- **1x** RedBoard or Arduino Uno
- **1x** Soft Potentiometer
- **1x** 10k resistor
- **3x** 330Ω resistors
- **1x** RGB LED
- **9x** Jumper Wires

Didn’t get the SIK?

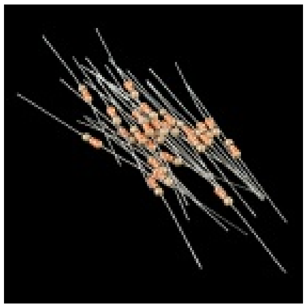
If you are following through this experiment and didn’t get the SIK, we suggest using these parts:

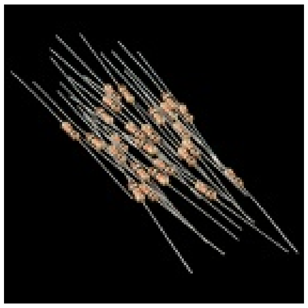
Jumper Wires Standard 7" M/M Pack of 30
◉ PRT-11026
$4.95


Breadboard - Self-Adhesive (White)
◉ PRT-12002
$4.95

LED - RGB Clear Common Cathode
◉ COM-00105
$1.95

SoftPot Membrane Potentiometer - 50mm
◉ SEN-08680
$4.95


Resistor 330 Ohm 1/6 Watt PTH - 20 pack
◉ COM-11507
$0.95


Resistor 10K Ohm 1/6th Watt PTH - 20 pack
◉ COM-11508
$0.95

You will also need either a RedBoard or Arduino Uno R3.

SparkFun RedBoard - Programmed with Arduino
◉ DEV-12757
$19.95

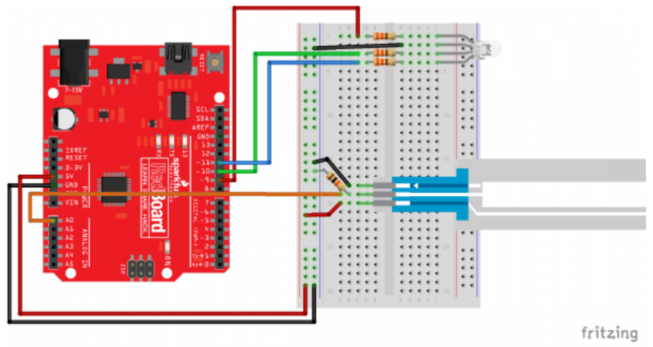
Arduino Uno- R3 SMD
◉ DEV-11224
$29.95

Hardware Hookup

Ready to start hooking everything up? Check out the Fritzing diagram below, to see how everything is connected.

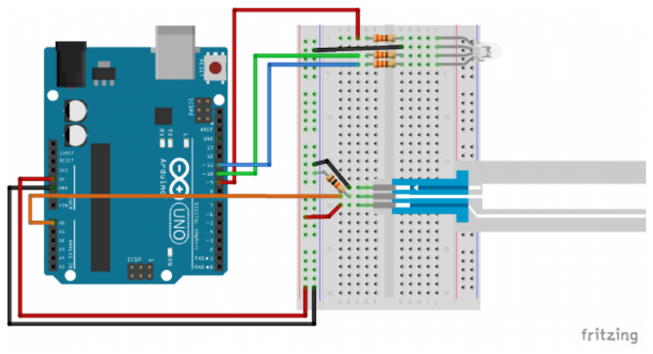
Polarized Components ⚠	Pay special attention to the component's markings indicating how to place it on the breadboard. Polarized components can only be connected to a circuit in one direction.

Fritzing Diagram for RedBoard



Having a hard time seeing the circuit? Click on the Fritzing diagram to see a bigger image.

Fritzing Diagram for Arduino



Having a hard time seeing the circuit? Click on the Fritzing diagram to see a bigger image.

Open the Sketch

Open Up the Arduino IDE software on your computer. Coding in the Arduino language will control your circuit. Open the code for Circuit 10 by accessing the “SIK Guide Code” you downloaded and placed into your “Examples” folder earlier.

To open the code go to: **File > examples > SIK Guide Code > Circuit_10**

You can also copy and paste the following code into the Arduino IDE. Hit upload, and see what happens!

```
/*
SparkFun Inventor's Kit
Example sketch 10

SOFT POTENTIOMETER

  Use the soft potentiometer to change the color
  of the RGB LED

  The soft potentiometer is a neat input device that detects
  pressure along its length. When you press it down with a finger
  (it works best on a flat surface), it will change resistance
  depending on where you're pressing it. You might use it to make
  a piano or light dimmer; here we're going to use it to control
  the color of an RGB LED.

Hardware connections:

  Soft potentiometer:

    The soft potentiometer is the large plastic strip with three
    pins. We'll be connecting it as a voltage divider, just like
    we did with the knob-type potentiometer back in circuit #2.

    Connect the middle pin to ANALOG IN pin 0 on the Arduino.
    Connect one side to 5V.
    Connect the other side to GND.
    Also connect a 10K resistor from the middle pin to GND.

    TIP: the soft pot will only work while you're actively
    pressing on it; at other times it will "float" to random
    values. To prevent this, we've added a 10K pull-down resistor
    to the middle pin (output voltage). This will keep the output
    at zero volts when the pot is not being pressed.

  RGB LED:

    An RGB LED is actually three LEDs (red, green, and blue)
    in one package. When we run them at different brightnesses,
    they mix to form new colors.

    Starting at the flattened edge of the flange on the LED,
    the pins are ordered RED, COMMON, GREEN, BLUE.

    Connect RED to a 330 Ohm resistor.
    Connect the other end of the resistor to Arduino digital pin 9.

    Connect COMMON to GND.

    Connect GREEN through a 330 Ohm resistor.
    Connect the other end of the resistor to Arduino digital pin 10.

    Connect BLUE through a 330 Ohm resistor.
    Connect the other end of the resistor to Arduino digital pin 11.

This sketch was written by SparkFun Electronics,
with lots of help from the Arduino community.
This code is completely free for any use.
Visit http://learn.sparkfun.com/products/2 for SIK information.
Visit http://www.arduino.cc to learn about the Arduino.

Version 2.0 6/2012 MDG
*/


// Constants for LED connections (note that these must be
// PWM pins, which are marked with "PWM" or have a "~" symbol
// next to them on the board).

const int RED_LED_PIN = 9;    // Red LED Pin
const int GREEN_LED_PIN = 10; // Green LED Pin
const int BLUE_LED_PIN = 11;  // Blue LED Pin

const int SENSOR_PIN = 0;      // Analog input pin

// Global PWM brightness values for the RGB LED.
// These are global so both loop() and setRGB() can see them.

int redValue, greenValue, blueValue;
```

```
void setup()
{
  // No need for any code here
  // analogWrite() sets up the pins as outputs
}


void loop()
{
  int sensorValue;

  // Read the voltage from the softpot (0-1023)

  sensorValue = analogRead(0);

  // We've written a new function called setRGB() (further down
  // in the sketch) that decodes sensorValue into a position
  // on the RGB "rainbow", and sets the RGB LED to that color.

  setRGB(sensorValue);
}


// setRGB()
// Set a RGB LED to a position on the "rainbow" of all colors.
// RGBposition should be in the range of 0 to 1023 (such as
// from an analog input).

void setRGB(int RGBposition)
{
  int mapRGB1, mapRGB2, constrained1, constrained2;

  // Here we take RGBposition and turn it into three RGB values.

  // The three values are computed so that the colors mix and
  // produce a rainbow of colors across the 0-1023 input range.

  // For each channel (red green blue), we're creating a "peak"
  // a third of the way along the 0-1023 range. By overlapping
  // these peaks with each other, the colors are mixed together.
  // This is most easily shown with a diagram:
  // http://sfecdn.s3.amazonaws.com/education/SIK/SchematicImages/Misc/RGB_functio
n.jpg

  // Create the red peak, which is centered at 0.
  // (Because it's centered at 0, half is after 0, and half
  // is before 1023):

  mapRGB1 = map(RGBposition, 0, 341, 255, 0);
  constrained1 = constrain(mapRGB1, 0, 255);
  mapRGB2 = map(RGBposition, 682, 1023, 0, 255);
  constrained2 = constrain(mapRGB2, 0, 255);

  redValue = constrained1 + constrained2;

  // Create the green peak, which is centered at 341
  // (one-third of the way to 1023):

  // Note that we've nested the functions by putting the map()
  // function inside the constrain() function. This can make your
  // code more compact, and requires fewer variabls:

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

  // Create the blue peak, which is centered at 682
  // (two-thirds of the way to 1023):

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

  // Now we have all three brightnesses,
  // we just need to display the computed color:

  analogWrite(RED_LED_PIN, redValue);
  analogWrite(GREEN_LED_PIN, greenValue);
  analogWrite(BLUE_LED_PIN, blueValue);

  // Feel free to use this function in your own code!
}
```

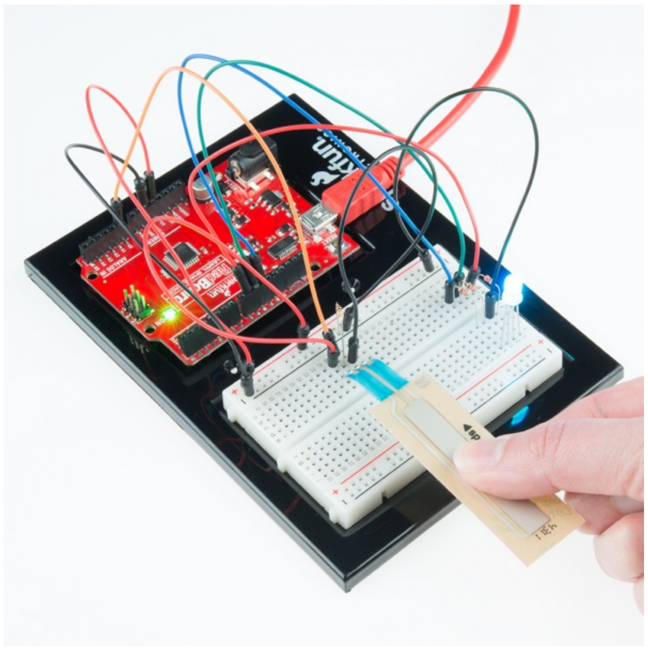
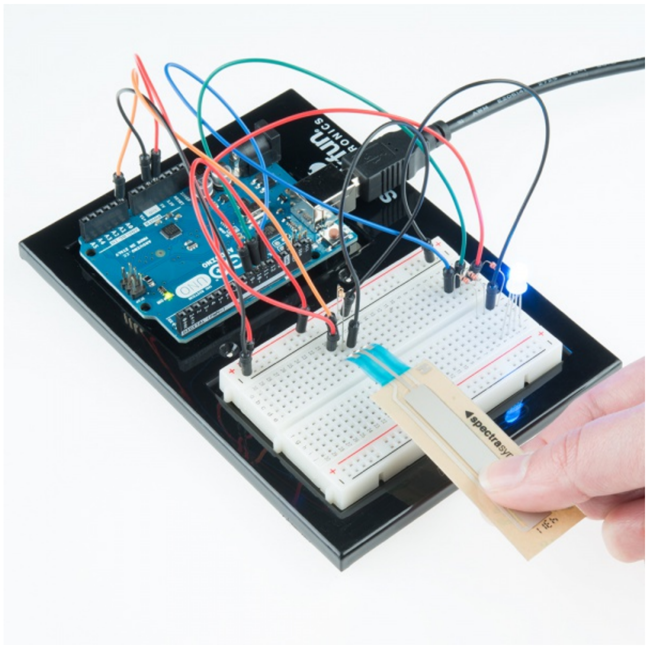
Code To Note

```
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.

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 section.

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.

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.

Experiment 11: Using a Piezo Buzzer

Introduction

In this circuit, we'll again bridge the gap between the digital world and the analog world. We'll be using a piezo 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 piezo 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!

Parts Needed

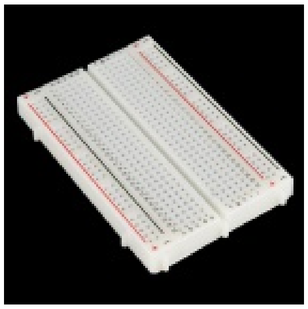
You will need the following parts:

- **1x** Breadboard
- **1x** RedBoard or Arduino Uno
- **1x** Piezo Buzzer
- **3x** Jumper Wires

Didn't get the SIK?

If you are following through this experiment and didn't get the SIK, we suggest using these parts:

Jumper Wires Standard 7" M/M Pack of 30
◉ PRT-11026
$4.95



Breadboard - Self-Adhesive (White)
◉ PRT-12002
$4.95

Piezo Speaker - PC Mount 12mm 2.048kHz
◉ COM-07950
$1.95

You will also need either a RedBoard or Arduino Uno R3.

SparkFun RedBoard - Programmed with Arduino
◉ DEV-12757
$19.95

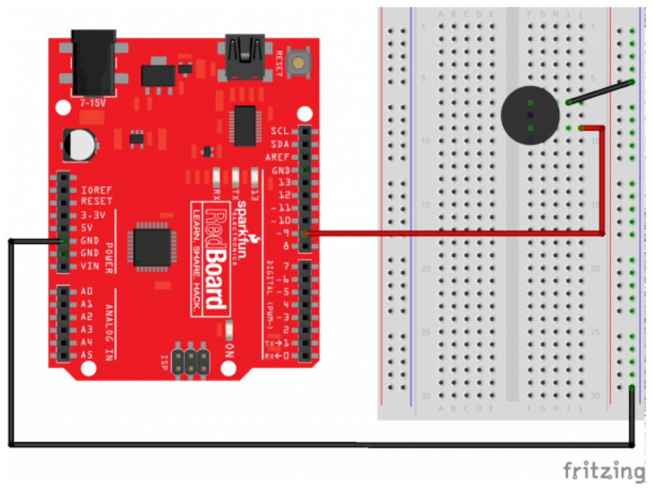
Arduino Uno- R3 SMD
◉ DEV-11224
$29.95

Hardware Hookup

Ready to start hooking everything up? Check out the Fritzing diagram below, to see how everything is connected.

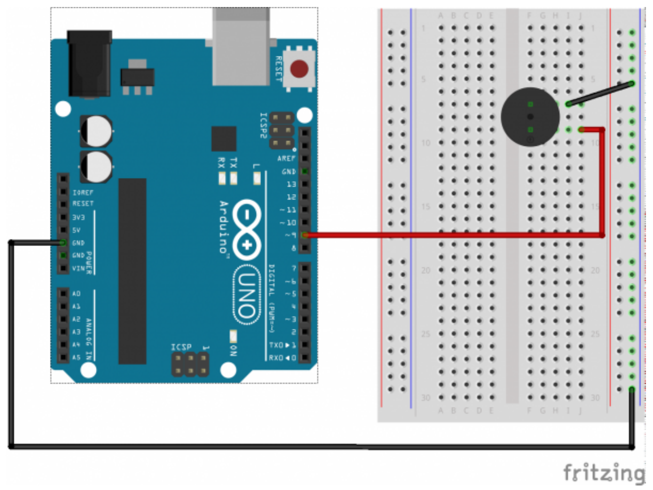
Polarized Components ⚠	Pay special attention to the component's markings indicating how to place it on the breadboard. Polarized components can only be connected to a circuit in one direction.

Fritzing Diagram for RedBoard



Having a hard time seeing the circuit? Click on the Fritzing diagram to see a bigger image.

Fritzing Diagram for Arduino



Having a hard time seeing the circuit? Click on the Fritzing diagram to see a bigger image.

Open the Sketch

Open Up the Arduino IDE software on your computer. Coding in the Arduino language will control your circuit. Open the code for Circuit 11 by accessing the "SIK Guide Code" you downloaded and placed into your "Examples" folder earlier.

To open the code go to: **File > examples > SIK Guide Code > Circuit_11**

You can also copy and paste the following code into the Arduino IDE. Hit upload, and see what happens!

```
/*
SparkFun Inventor's Kit
Example sketch 11

BUZZER

  Use the buzzer to play a song!

  The buzzer in your Inventor's Kit is an electromechanical
  component you can use to make noise. Inside the buzzer is a
  coil of wire and a small magnet. When current flows through
  the coil, it becomes magnetized and pulls towards the magnet,
  creating a tiny "click". When you do this thousands of times
  per second, you create tones.

  The Arduino has a built-in command called tone() which clicks
  the buzzer at a certain frequency. This sketch knows the
  frequencies of the common notes, allowing you to create songs.
  We're never going to let you down!

Hardware connections:

  The buzzer has two pins. One is positive and one is negative.
  The postitive pin is marked by a "+" symbol on both the top
  and bottom of the buzzer.

  Connect the positive pin to Arduino digital pin 9.
  (Note that this must be a PWM pin.)
  Connect the negative pin to GND.

  Tip: if the buzzer doesn't fit into the breadboard easily,
  try rotating it slightly to fit into diagonal holes.

This sketch was written by SparkFun Electronics,
with lots of help from the Arduino community.
(This sketch was originally developed by D. Cuartielles for K3)
This code is completely free for any use.
Visit http://learn.sparkfun.com/products/2 for SIK information.
Visit http://www.arduino.cc to learn about the Arduino.

Version 2.0 6/2012 MDG
*/

/*
This sketch uses the buzzer to play songs.
The Arduino's tone() command will play notes of a given frequency.
We'll provide a function that takes in note characters (a-g),
and returns the corresponding frequency from this table:

  note  frequency
  c     262 Hz
  d     294 Hz
  e     330 Hz
  f     349 Hz
  g     392 Hz
  a     440 Hz
  b     494 Hz
  C     523 Hz

For more information, see http://arduino.cc/en/Tutorial/Tone
*/

const int buzzerPin = 9;

// We'll set up an array with the notes we want to play
// change these values to make different songs!

// Length must equal the total number of notes and spaces

const int songLength = 18;

// Notes is an array of text characters corresponding to the notes
// in your song. A space represents a rest (no tone)

char notes[] = "cdfda ag cdfdg gf "; // a space represents a rest

// Beats is an array of values for each note and rest.
// A "1" represents a quarter-note, 2 a half-note, etc.
// Don't forget that the rests (spaces) need a length as well.

int beats[] = {1,1,1,1,1,1,4,4,2,1,1,1,1,1,1,4,4,2};

// The tempo is how fast to play the song.
// To make the song play faster, decrease this value.

int tempo = 150;


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


void loop()
{
  int i, duration;
  for (i = 0; i < songLength; i++) // step through the song arrays
  {
    duration = beats[i] * tempo;  // length of note/rest in ms

    if (notes[i] == ' ')          // is this a rest?
    {
      delay(duration);            // then pause for a moment
    }
    else                          // otherwise, play the note
    {
      tone(buzzerPin, frequency(notes[i]), duration);
      delay(duration);            // wait for tone to finish
    }
    delay(tempo/10);              // brief pause between notes
  }

  // We only want to play the song once, so we'll pause forever:
  while(true){}
  // If you'd like your song to play over and over,
  // remove the above statement
}


int frequency(char note)
{
  // This function takes a note character (a-g), and returns the
  // corresponding frequency in Hz for the tone() function.

  int i;
  const int numNotes = 8;  // number of notes we're storing

  // The following arrays hold the note characters and their
  // corresponding frequencies. The last "C" note is uppercase
  // to separate it from the first lowercase "c". If you want to
  // add more notes, you'll need to use unique characters.

  // For the "char" (character) type, we put single characters
  // in single quotes.

  char names[] = { 'c', 'd', 'e', 'f', 'g', 'a', 'b', 'C' };
  int frequencies[] = {262, 294, 330, 349, 392, 440, 494, 523};

  // Now we'll search through the letters in the array, and if
  // we find it, we'll return the frequency for that note.

  for (i = 0; i < numNotes; i++)  // Step through the notes
  {
    if (names[i] == note)         // Is this the one?
    {
      return(frequencies[i]);     // Yes! Return the frequency
    }
  }
  return(0);  // We looked through everything and didn't find it,
              // but we still need to return a value, so return 0.
}
```

Code To Note

`char notes[] = "cdfda ag cdfdg gf ";`

`char names[] = {'c','d','e','f','g','a','b','C'};`

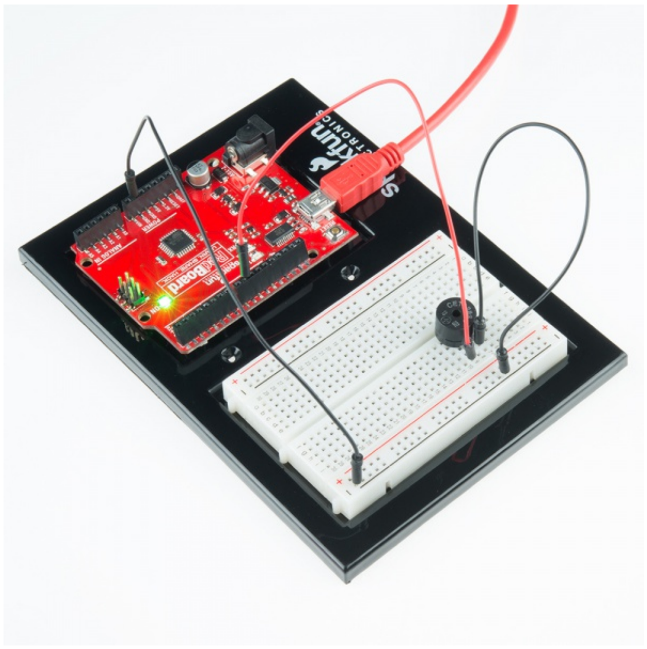
Up until now we've been working solely with numerical data, but the Arduino can also work with text. Characters (single, printable, letters, numbers and other symbols) have their own type, called "char". When you have an array of characters, it can be defined between double-quotes (also called a "string"),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

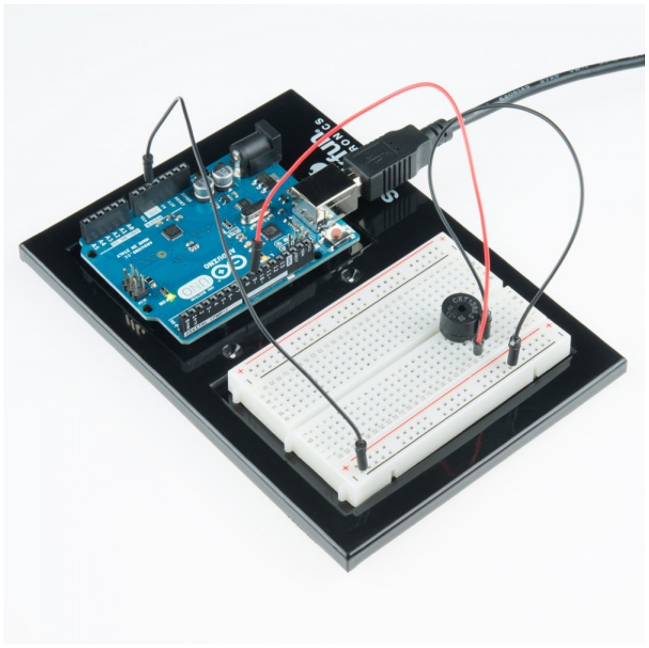
`tone(pin, frequency, duration);`

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 section.





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.

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.

Experiment 12: Driving a Motor

Introduction

Back in experiment 8, you got to work 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 or Arduino Uno R3 can.

When using a transistor, you just need to make sure its maximum specs are high enough for your use case. 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.

Parts Needed

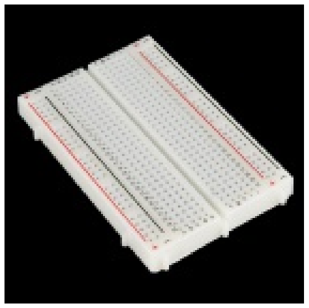
You will need the following parts:

- **1x** Breadboard
- **1x** RedBoard or Arduino Uno
- **1x** Motor
- **1x** 330Ω Resistor
- **1x** NPN transistor
- **1x** Diode 1N4148
- **6x** Jumper Wires

Didn’t get the SIK?

If you are following through this experiment and didn’t get the SIK, we suggest using these parts:

Jumper Wires Standard 7" M/M Pack of 30


Breadboard - Self-Adhesive (White)

◉ PRT-11026

$4.95

◉ PRT-12002

$4.95

Hobby Motor - Gear

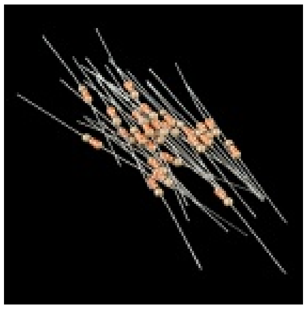
◉ ROB-11696

$1.95

Transistor - NPN (P2N2222A)

◉ COM-12852

$0.50



Resistor 330 Ohm 1/6 Watt PTH - 20 pack

◉ COM-11507

$0.95

Diode Small Signal - 1N4148

○ COM-08588

$0.15

You will also need either a RedBoard or Arduino Uno R3.

SparkFun RedBoard - Programmed with Arduino

◉ DEV-12757

$19.95

Arduino Uno- R3 SMD

◉ DEV-11224

$29.95

Suggested Reading

Before continuing on with this experiment, we recommend you be familiar with the concepts in the following tutorial:

- Motors and Selecting the Right One
- Diodes
- Transistors

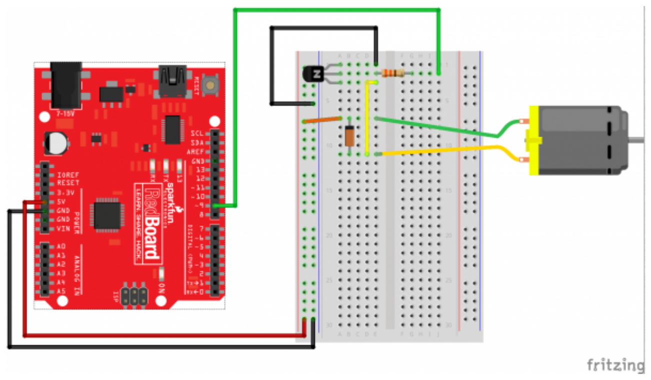
Hardware Hookup

Ready to start hooking everything up? Check out the Fritzing diagram below, to see how everything is connected.

Polarized Components ⚠	Pay special attention to the component's markings indicating how to place it on the breadboard. Polarized components can only be connected to a circuit in one direction.

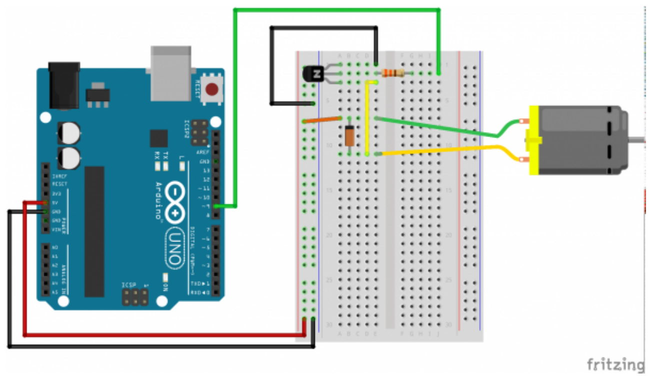
Please note: When you're building the circuit be careful not to mix up the transistor and the temperature sensor, they're almost identical. Look for "P2N2222A" on the body of the transistor.

Fritzing Diagram for RedBoard



Having a hard time seeing the circuit? Click on the Fritzing diagram to see a bigger image.

Fritzing Diagram for Arduino



Having a hard time seeing the circuit? Click on the Fritzing diagram to see a bigger image.

Open the Sketch

Open Up the Arduino IDE software on your computer. Coding in the Arduino language will control your circuit. Open the code for Circuit 12 by accessing the “SIK Guide Code” you downloaded and placed into your “Examples” folder earlier.

To open the code go to: **File > examples > SIK Guide Code > Circuit_12**

You can also copy and paste the following code into the Arduino IDE. Hit upload, and see what happens!

```
/*
SparkFun Inventor's Kit
Example sketch 12

SPINNING A MOTOR

  Use a transistor to spin a motor at different speeds.
  We'll also show you how to input data from the serial port
  (see the serialSpeed() function below).

  Motors are the basis for thousands of things in our daily lives,
  and the Arduino can control them. Here we'll use pulse-width
  modulation (PWM) to vary the speed of a motor.

  The Arduino pins are strong enough to light small LEDs (up to
  40 milliAmps), but they're not strong enough to run motors and
  other power-hungry parts. (This motor needs 50-100mA).
  Because the motor needs more current than an Arduino pin can
  provide, we'll use a transistor to do the heavy lifting.
  A transistor is a solid-state switch. When we give it a small
  amount of current, it can switch a much larger current.
  The transistors in your kit (2N2222) can switch up to 200mA.

  You can turn a transistor on and off using the digitalWrite()
  function, but you can also use the analogWrite() function to
  vary the speed of the motor. The analogWrite() function pulses
  a pin, varying the width of the pulse from 0% to 100%. We call
  this technique "PWM", for "Pulse-Width Modulation".

  One thing to keep in mind is that when you lower the speed of
  a motor using PWM, you're also reducing the torque (strength)
  of the motor. For PWM values below 50 or so, the motor won't have
  enough torque to start spinning. It will start spinning when you
  raise the speed a bit.

Hardware connections:

  Transistor:

    The transistor has three pins. Looking at the flat side with the
    pins down, the order is COLLECTOR, BASE, EMITTER.

    Connect the black wire on the motor to the
    COLLECTOR pin on the transistor.

    Connect the BASE pin through a 330 Ohm resistor to
    digital pin 9.

    Connect the EMITTER pin to GND.

  Motor:

    You've already connected the black wire on the motor to the
    COLLECTOR pin on the transistor.

    Connect the other (red) wire on the motor to 5V.

  Flyback diode:

    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.

    Connect the side of the diode with the band (cathode) to 5V
    Connect the other side of the diode (anode) to the black wire
    on the motor.

This sketch was written by SparkFun Electronics,
with lots of help from the Arduino community.
This code is completely free for any use.
Visit http://learn.sparkfun.com/products/2 for SIK information.
Visit http://www.arduino.cc to learn about the Arduino.

Version 2.0 6/2012 MDG
*/

// We'll be controlling the motor from pin 9.
// This must be one of the PWM-capable pins.

const int motorPin = 9;


void setup()
{
  // Set up the motor pin to be an output:

  pinMode(motorPin, OUTPUT);

  // Set up the serial port:

  Serial.begin(9600);
}
```

```
void loop()
{
  // Here we've used comments to disable some of the examples.
  // To try different things, uncomment one of the following lines
  // and comment the other ones. See the functions below to learn
  // what they do and how they work.

  // motorOnThenOff();
  // motorOnThenOffWithSpeed();
  // motorAcceleration();
     serialSpeed();
}


// This function turns the motor on and off like the blinking LED.
// Try different values to affect the timing.

void motorOnThenOff()
{
  int onTime = 3000;  // milliseconds to turn the motor on
  int offTime = 3000; // milliseconds to turn the motor off

  digitalWrite(motorPin, HIGH); // turn the motor on (full speed)
  delay(onTime);                // delay for onTime milliseconds
  digitalWrite(motorPin, LOW);  // turn the motor off
  delay(offTime);               // delay for offTime milliseconds
}


// This function alternates between two speeds.
// Try different values to affect the timing and speed.

void motorOnThenOffWithSpeed()
{
  int Speed1 = 200;  // between 0 (stopped) and 255 (full speed)
  int Time1 = 3000;  // milliseconds for speed 1

  int Speed2 = 50;   // between 0 (stopped) and 255 (full speed)
  int Time2 = 3000;  // milliseconds to turn the motor off

  analogWrite(motorPin, Speed1);  // turns the motor On
  delay(Time1);                   // delay for onTime milliseconds
  analogWrite(motorPin, Speed2);  // turns the motor Off
  delay(Time2);                   // delay for offTime milliseconds
}


// This function slowly accelerates the motor to full speed,
// then back down to zero.
```

```
void motorAcceleration()
{
  int speed;
  int delayTime = 20; // milliseconds between each speed step

  // accelerate the motor

  for(speed = 0; speed <= 255; speed++)
  {
    analogWrite(motorPin,speed);    // set the new speed
    delay(delayTime);               // delay between speed steps
  }

  // decelerate the motor

  for(speed = 255; speed >= 0; speed--)
  {
    analogWrite(motorPin,speed);    // set the new speed
    delay(delayTime);               // delay between speed steps
  }
}


// This function will let you type a speed into the serial
// monitor window. Open the serial monitor using the magnifying-
// glass icon at the top right of the Arduino window. Then
// type your desired speed into the small text entry bar at the
// top of the window and click "Send" or press return. The motor
// will then operate at that speed. The valid range is 0 to 255.

void serialSpeed()
{
  int speed;

  Serial.println("Type a speed (0-255) into the box above,");
  Serial.println("then click [send] or press [return]");
  Serial.println();  // Print a blank line

  // In order to type out the above message only once,
  // we'll run the rest of this function in an infinite loop:

  while(true)  // "true" is always true, so this will loop forever.
  {
    // First we check to see if incoming data is available:

    while (Serial.available() > 0)
    {
      // If it is, we'll use parseInt() to pull out any numbers:

      speed = Serial.parseInt();
      // Because analogWrite() only works with numbers from
      // 0 to 255, we'll be sure the input is in that range:

      speed = constrain(speed, 0, 255);

      // We'll print out a message to let you know that the
      // number was received:

      Serial.print("Setting speed to ");
      Serial.println(speed);

      // And finally, we'll set the speed of the motor!

      analogWrite(motorPin, speed);
    }
  }
}
```

Code To Note

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

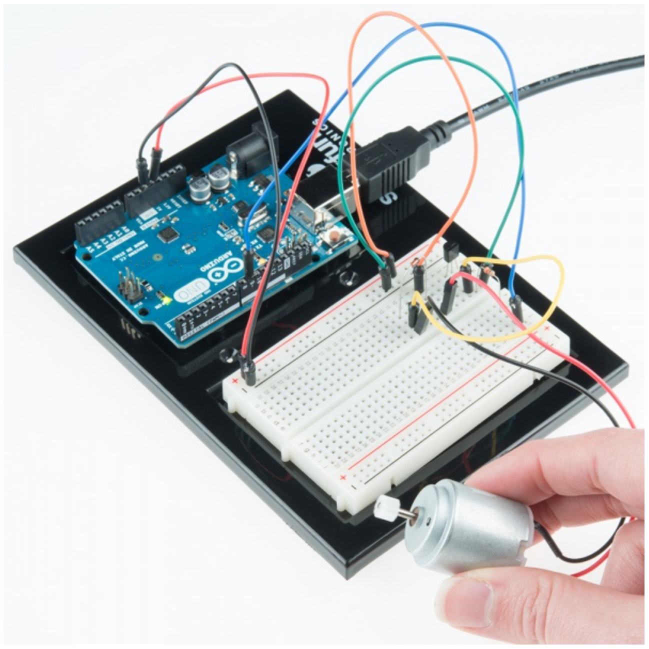
The Galileo's serial port can be used to receive as well as send data. Because data could arrive at any time, the Galileostores, or "buffers" data coming into the port until you're ready to use it. The `Serial.available()` command returns the number of characters that the port has received, but haven't 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.



Real World Application

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

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

Experiment 13: Using Relays

Introduction

In this circuit, we are going to use some of the lessons we learned in experiment 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 Galileo even more powerful abilities!

Parts Needed

You will need the following parts:

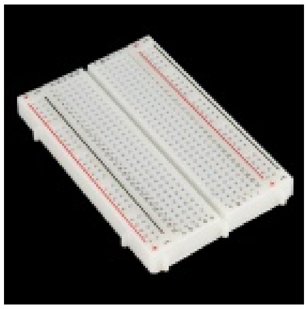
- **1x** Breadboard
- **1x** RedBoard or Arduino Uno

- **2x** LEDs
- **2x** 330Ω Resistors
- **1x** Relay (SPDT)
- **1x** Diode 1N4148
- **1x** NPN Transistor
- **14x** Jumper Wires

Didn’t get the SIK?

If you are following through this experiment and didn’t get the SIK, we suggest using these parts:

Jumper Wires Standard 7" M/M Pack of 30
◉ PRT-11026
$4.95

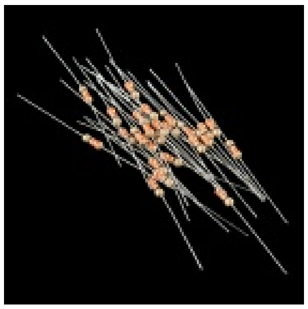


Breadboard - Self-Adhesive (White)
◉ PRT-12002
$4.95

LED - Assorted (20 pack)
◉ COM-12062
$2.95

Relay SPDT Sealed
○ COM-00100
$1.95

Transistor - NPN (P2N2222A)
◉ COM-12852
$0.50



Resistor 330 Ohm 1/6 Watt PTH - 20 pack
◉ COM-11507
$0.95

Diode Small Signal - 1N4148

○ COM-08588

$0.15

You will also need either a RedBoard or Arduino Uno R3.

SparkFun RedBoard - Programmed with Arduino

◉ DEV-12757

$19.95

Arduino Uno- R3 SMD

◉ DEV-11224

$29.95

Suggested Reading

Before continuing on with this experiment, we recommend you be familiar with the concepts in the following tutorial:

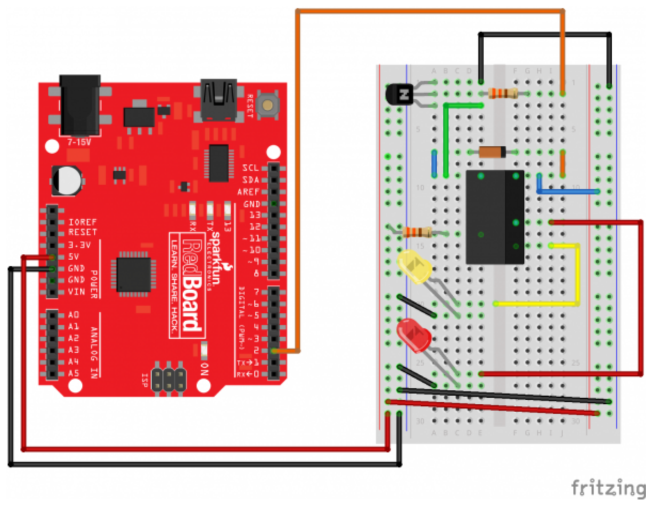
- Switch Basics

Hardware Hookup

Ready to start hooking everything up? Check out the Fritzing diagram below, to see how everything is connected.

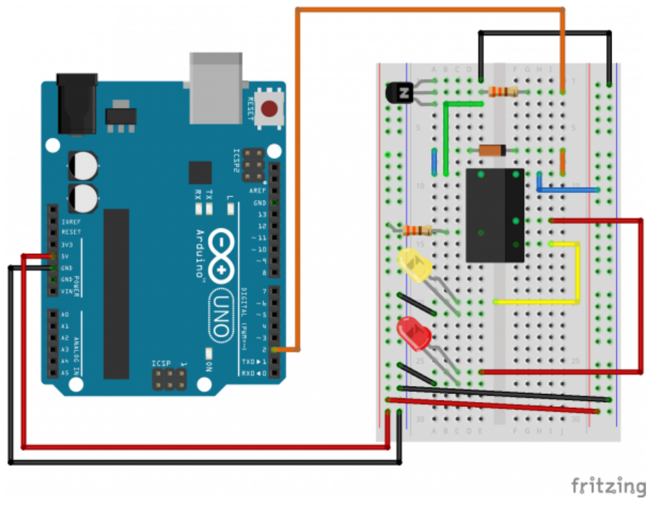
Polarized Components ⚠	Pay special attention to the component's markings indicating how to place it on the breadboard. Polarized components can only be connected to a circuit in one direction.

Fritzing Diagram for RedBoard



Having a hard time seeing the circuit? Click on the Fritzing diagram to see a bigger image.

Fritzing Diagram for Arduino



Having a hard time seeing the circuit? Click on the Fritzing diagram to see a bigger image.

Open the Sketch

Open Up the Arduino IDE software on your computer. Coding in the Arduino language will control your circuit. Open the code for Circuit 13 by accessing the “SIK Guide Code” you downloaded and placed into your “Examples” folder earlier.

To open the code go to: **File > examples > SIK Guide Code > Circuit_13**

You can also copy and paste the following code into the Arduino IDE. Hit upload, and see what happens!

```
/*
SparkFun Inventor's Kit
Example sketch 13

RELAYS

  Use a transistor to drive a relay

  A relay is a electrically-controlled mechanical switch.
  It can control much more voltage and current than an Arduino pin
  (or the transistor included in your kit) can. If you want to use
  the Arduino to control a 120V bulb, coffee maker, or other high-
  power device, a relay is an excellent way to do that. Because
  the relay needs more power to switch than an Arduino pin can
  provide, we'll use a transistor to drive the relay in exactly
  the same way we used a transistor to drive a motor in circuit 12.

  A relay consists of a coil of wire, and switch contacts. When
  you apply power to the coil, it becomes magnetized, and pulls
  the switch contacts closed. Since the switch contacts are
  completely isolated from the Arduino, you can safely use a
  relay to control normally dangerous voltages (but please only do
  this if you already know how to safely work with high voltage!).

  The relay has three contact pins, COM (common), NC (Normally
  Closed), and NO (Normally Open). When the relay is turned off,
  the COM pin is connected to the NC (Normally Closed) pin. When
  the relay is turned on, the COM pin is connected to the NO
  (Normally Open) pin.

  This code is very simple - it turns the relay on for one second,
  and off for one second, the same as the blink sketch!

Hardware connections:

  Transistor:

    The transistor has three pins. Looking at the flat side with
    the pins down, the order is COLLECTOR, BASE, EMITTER.

    Connect the BASE pin through a 1K resistor to digital pin 2.

    Connect the EMITTER pin to GND.

  Relay coil:

    The relay has pins for a coil (which you use to control the
    relay), and contacts (which you connect to the device you'd
    like to switch). The top or bottom of the relay should have
    a symbol indicating the coil pins.

    Connect one side of the coil to the COLLECTOR pin
    on the transistor.

    Connect other side of the coil to 5V.

  Diode:

    The relay has a coil that you energize to close the switch.
    When you disconnect power from a coil, the coil will generate
    a voltage spike that can damage the transistor. This diode
    protects the transistor from the voltage spike.

    Connect the side of the diode with the band (cathode) to 5V

    Connect the other side of the diode (anode) to the COLLECTOR
    pin of the transistor.

  Relay contacts and LEDs:

    We'll use the relay contacts to turn LEDs on and off, but you
    can use them to switch almost anything on and off.

    Connect the COMMON side of the switch to a 330 Ohm resistor.
    Connect the other side of the above resistor to 5V.

    Connect the NC (Normally Closed) side of the switch to the
    positive (longer) leg of LED 1.

    Connect the NO (Normally Open) side of the switch to the
    positive (longer) leg of LED 2.

    Connect the negative sides (shorter leg) of both LEDs to GND.

This sketch was written by SparkFun Electronics,
with lots of help from the Arduino community.
This code is completely free for any use.
Visit http://learn.sparkfun.com/products/2 for SIK information.
Visit http://www.arduino.cc to learn about the Arduino.

Version 2.0 6/2012 MDG
*/


const int relayPin = 2;     // use this pin to drive the transistor
const int timeDelay = 1000; // delay in ms for on and off phases

// You can make timeDelay shorter, but note that relays, being
// mechanical devices, will wear out quickly if you try to drive
// them too fast.


void setup()
{
  pinMode(relayPin, OUTPUT);  // set pin as an output
}


void loop()
{
  digitalWrite(relayPin, HIGH);  // turn the relay on
  delay(timeDelay);              // wait for one second

  digitalWrite(relayPin, LOW);   // turn the relay off

  delay(timeDelay);              // wait for one second
}
```

Code To Note

`digitalWrite(relayPin, HIGH);`

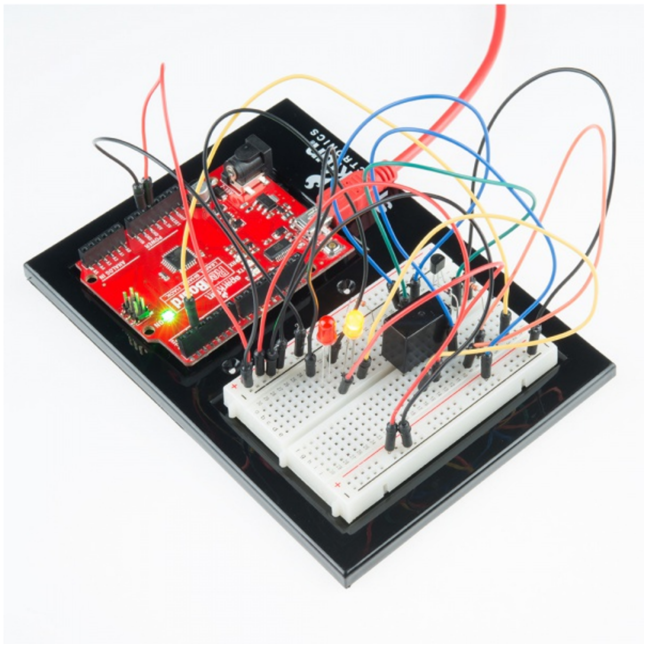
When we turn on the transistor, which in turn energizes the relay’s coil, the relay’s switch contacts are closed. This connects the relay’s COM pin to the NO (Normally Open) pin. Whatever you’ve connected using these pins will turn on. (Here we’re using LEDs, but this could be almost anything.)

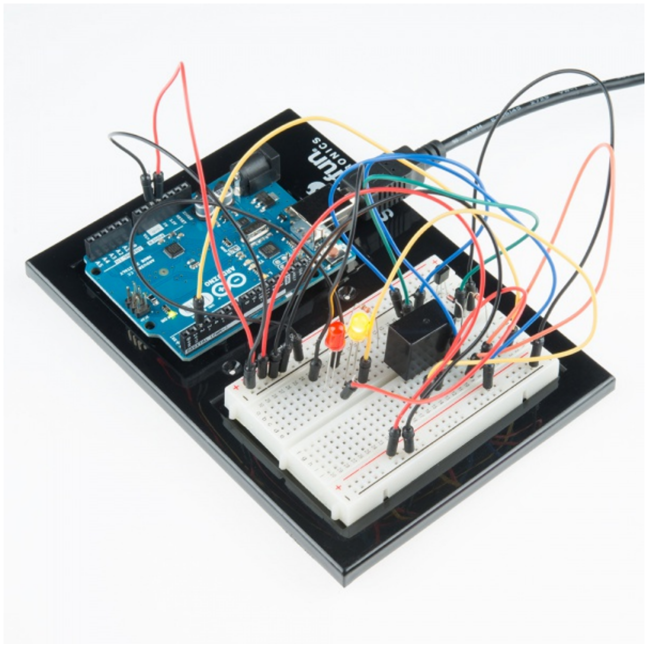
`digitalWrite(relayPin, LOW);`

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 section.





Real World Application

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

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.

Experiment 14: Using a Shift Register

Introduction

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 or Arduino Uno R3 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 Needed

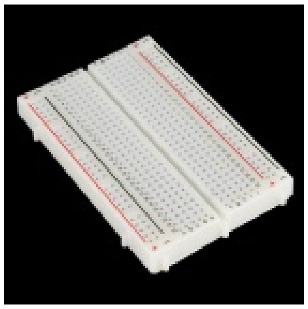
You will need the following parts:

- **1x** Breadboard
- **1x** RedBoard or Arduino Uno
- **8x** LEDs
- **8x** 330Ω Resistors
- **1x** Shift Register 8-Bit - 74HC595
- **19x** Jumper Wires

Didn't get the SIK?

If you are following through this experiment and didn't get the SIK, we suggest using these parts:

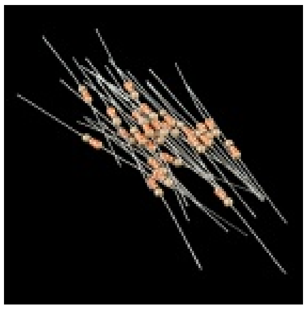
Jumper Wires Standard 7" M/M Pack of 30
● PRT-11026
$4.95



Breadboard - Self-Adhesive (White)
● PRT-12002
$4.95

LED - Assorted (20 pack)
◉ COM-12062
$2.95

Shift Register 8-Bit - 74HC595
◉ COM-00733
$1.50



Resistor 330 Ohm 1/6 Watt PTH - 20 pack
◉ COM-11507
$0.95

You will also need either a RedBoard or Arduino Uno R3.

SparkFun RedBoard - Programmed with Arduino
◉ DEV-12757
$19.95

Arduino Uno- R3 SMD
◉ DEV-11224
$29.95

Suggested Reading

Before continuing on with this experiment, we recommend you be familiar with the concepts in the following tutorial:

- Shift Registers

- Integrated Circuits

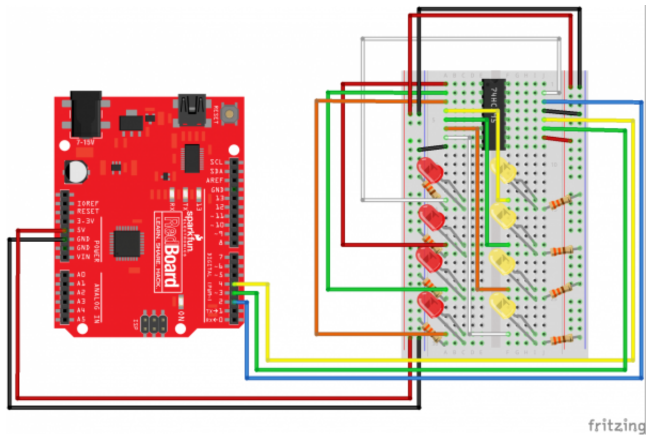
Hardware Hookup

Ready to start hooking everything up? Check out the Fritzing diagram below, to see how everything is connected.

Polarized Components ⚠	Pay special attention to the component's markings indicating how to place it on the breadboard. Polarized components can only be connected to a circuit in one direction.

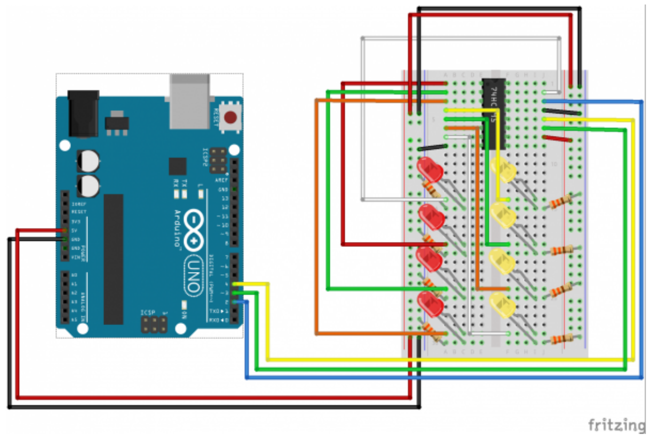
For the shift register, align notch on top, in-between "e1" and "f1" on the breadboard. The notch indicates where pin 1 is.

Fritzing Diagram for RedBoard



Having a hard time seeing the circuit? Click on the Fritzing diagram to see a bigger image.

Fritzing Diagram for Arduino



Having a hard time seeing the circuit? Click on the Fritzing diagram to see a bigger image.

Open the Sketch

Open Up the Arduino IDE software on your computer. Coding in the Arduino language will control your circuit. Open the code for Circuit 14 by accessing the "SIK Guide Code" you downloaded and placed into your "Examples" folder earlier.

To open the code go to: **File > examples > SIK Guide Code > Circuit_14**

You can also copy and paste the following code into the Arduino IDE. Hit upload, and see what happens!

```
/*
SparkFun Inventor's Kit
Example sketch 13

SHIFT REGISTER

  Use a shift register to turn three pins into eight (or more!)
  outputs

  An integrated circuit ("IC"), or "chip", is a self-contained
  circuit built into a small plastic package. (If you look closely
  at your Arduino board you'll see a number of ICs.) There are
  thousands of different types of ICs available that you can use
  to perform many useful functions.

  The 74HC595 shift register in your kit is an IC that has eight
  digital outputs. To use these outputs, we'll use a new interface
  called SPI (Serial Peripheral Interface). It's like the TX and
  RX you're used to, but has an additional "clock" line that
  controls the speed of the data transfer. Many parts use SPI
  for communications, so the Arduino offers simple commands called
  shiftIn() and shiftOut() to access these parts.

  This IC lets you use three digital pins on your Arduino to
  control eight digital outputs on the chip. And if you need
  even more outputs, you can daisy-chain multiple shift registers
  together, allowing an almost unlimited number of outputs from
  the same three Arduino pins! See the shift register datasheet
  for details:

  http://www.sparkfun.com/datasheets/IC/SN74HC595.pdf

Hardware connections:

  Shift register:

    Plug in the chip so it bridges the center "canyon"
    on the breadboard.

    The shift register has 16 pins. They are numbered
    counterclockwise starting at the pin 1 mark (notch
    in the end of the chip). See the datasheet above
    for a diagram.

    74HC595 pin     LED pin     Arduino pin

    1  (QB)     LED 2 +
    2  (QC)     LED 3 +
    3  (QD)     LED 4 +
    4  (QE)     LED 5 +
    5  (QF)     LED 6 +
    6  (QG)     LED 7 +
    7  (QH)     LED 8 +
    8  (GND)                GND

    9  (QH*)
    10 (SRCLR*)             5V
    11 (SRCLK)              Digital 3
    12 (RCLK)               Digital 4
    13 (OE*)                GND
    14 (SER)                Digital 2
    15 (QA)     LED 1 +
    16 (VCC)                5V

  LEDs:

    After making the above connections to the positive (longer)
    legs of the LEDs, connect the negative side (short lead) of
    each LED to a 330 Ohm resistor, and connect the other side
    of each resistor to GND.

This sketch was written by SparkFun Electronics,
with lots of help from the Arduino community.
This code is completely free for any use.
Visit http://learn.sparkfun.com/products/2 for SIK information.
Visit http://www.arduino.cc to learn about the Arduino.

Version 2.0 6/2012 MDG
*/


// Pin definitions:
// The 74HC595 uses a type of serial connection called SPI
// (Serial Peripheral Interface) that requires three pins:

int datapin = 2;
int clockpin = 3;
int latchpin = 4;

// We'll also declare a global variable for the data we're
// sending to the shift register:

byte data = 0;


void setup()
{
  // Set the three SPI pins to be outputs:
```

```
  pinMode(datapin, OUTPUT);
  pinMode(clockpin, OUTPUT);
  pinMode(latchpin, OUTPUT);
}


void loop()
{
  // We're going to use the same functions we played with back
  // in circuit 04, "Multiple LEDs", we've just replaced
  // digitalWrite() with a new function called shiftWrite()
  // (see below). We also have a new function that demonstrates
  // binary counting.

  // To try the different functions below, uncomment the one
  // you want to run, and comment out the remaining ones to
  // disable them from running.

  oneAfterAnother();      // All on, all off

  //oneOnAtATime();       // Scroll down the line

  //pingPong();           // Like above, but back and forth

  //randomLED();          // Blink random LEDs

  //marquee();

  //binaryCount();        // Bit patterns from 0 to 255
}


void shiftWrite(int desiredPin, boolean desiredState)

// This function lets you make the shift register outputs
// HIGH or LOW in exactly the same way that you use digitalWrite().

// Like digitalWrite(), this function takes two parameters:

//    "desiredPin" is the shift register output pin
//    you want to affect (0-7)

//    "desiredState" is whether you want that output
//    to be HIGH or LOW

// Inside the Arduino, numbers are stored as arrays of "bits",
// each of which is a single 1 or 0 value. Because a "byte" type
// is also eight bits, we'll use a byte (which we named "data"
// at the top of this sketch) to send data to the shift register.
// If a bit in the byte is "1", the output will be HIGH. If the bit
// is "0", the output will be LOW.

// To turn the individual bits in "data" on and off, we'll use
// a new Arduino commands called bitWrite(), which can make
// individual bits in a number 1 or 0.
{
  // First we'll alter the global variable "data", changing the
  // desired bit to 1 or 0:

  bitWrite(data,desiredPin,desiredState);

  // Now we'll actually send that data to the shift register.
  // The shiftOut() function does all the hard work of
  // manipulating the data and clock pins to move the data
  // into the shift register:

  shiftOut(datapin, clockpin, MSBFIRST, data);

  // Once the data is in the shift register, we still need to
  // make it appear at the outputs. We'll toggle the state of
  // the latchPin, which will signal the shift register to "latch"
  // the data to the outputs. (Latch activates on the high-to
  // -low transition).

  digitalWrite(latchpin, HIGH);
  digitalWrite(latchpin, LOW);
}


/*
oneAfterAnother()

This function will light one LED, delay for delayTime, then light
the next LED, and repeat until all the LEDs are on. It will then
turn them off in the reverse order.
*/

void oneAfterAnother()
{
  int index;
  int delayTime = 100; // Time (milliseconds) to pause between LEDs
                       // Make this smaller for faster switching

  // Turn all the LEDs on:

  // This for() loop will step index from 0 to 7
  // (putting "++" after a variable means add one to it)
  // and will then use digitalWrite() to turn that LED on.

  for(index = 0; index <= 7; index++)
  {
    shiftWrite(index, HIGH);
    delay(delayTime);
  }

  // Turn all the LEDs off:

  // This for() loop will step index from 7 to 0
  // (putting "--" after a variable means subtract one from it)
  // and will then use digitalWrite() to turn that LED off.

  for(index = 7; index >= 0; index--)
  {
    shiftWrite(index, LOW);
    delay(delayTime);
  }
}


/*
oneOnAtATime()

This function will step through the LEDs, lighting one at at time.
*/

void oneOnAtATime()
{
  int index;
  int delayTime = 100; // Time (milliseconds) to pause between LEDs
                       // Make this smaller for faster switching

  // step through the LEDs, from 0 to 7

  for(index = 0; index <= 7; index++)
  {
    shiftWrite(index, HIGH);    // turn LED on
    delay(delayTime);       // pause to slow down the sequence
    shiftWrite(index, LOW); // turn LED off
  }
}


/*
pingPong()

This function will step through the LEDs, lighting one at at time,
in both directions.
*/

void pingPong()
{
  int index;
  int delayTime = 100; // time (milliseconds) to pause between LEDs
                       // make this smaller for faster switching

  // step through the LEDs, from 0 to 7

  for(index = 0; index <= 7; index++)
  {
    shiftWrite(index, HIGH);    // turn LED on
    delay(delayTime);       // pause to slow down the sequence
    shiftWrite(index, LOW); // turn LED off
  }

  // step through the LEDs, from 7 to 0

  for(index = 7; index >= 0; index--)
  {
    shiftWrite(index, HIGH);    // turn LED on
    delay(delayTime);       // pause to slow down the sequence
    shiftWrite(index, LOW); // turn LED off
  }
}


/*
randomLED()

This function will turn on random LEDs. Can you modify it so it
also lights them for random times?
*/

void randomLED()
{
  int index;
  int delayTime = 100; // time (milliseconds) to pause between LEDs
                       // make this smaller for faster switching

  // The random() function will return a semi-random number each
  // time it is called. See http://arduino.cc/en/Reference/Random
  // for tips on how to make random() more random.

  index = random(8);    // pick a random number between 0 and 7

  shiftWrite(index, HIGH);  // turn LED on
  delay(delayTime);     // pause to slow down the sequence
  shiftWrite(index, LOW);   // turn LED off
}


/*
marquee()
```

```
This function will mimic "chase lights" like those around signs.
*/

void marquee()
{
  int index;
  int delayTime = 200; // Time (milliseconds) to pause between LEDs
                       // Make this smaller for faster switching

  // Step through the first four LEDs
  // (We'll light up one in the lower 4 and one in the upper 4)

  for(index = 0; index <= 3; index++)
  {
    shiftWrite(index, HIGH);    // Turn a LED on
    shiftWrite(index+4, HIGH);  // Skip four, and turn that LED on
    delay(delayTime);       // Pause to slow down the sequence
    shiftWrite(index, LOW); // Turn both LEDs off
    shiftWrite(index+4, LOW);
  }
}


/*
binaryCount()

Numbers are stored internally in the Arduino as arrays of "bits",
each of which is a 1 or 0. Just like the base-10 numbers we use
every day, The position of the bit affects the magnitude of its
contribution to the total number:

Bit position   Contribution
0              1
1              2
2              4
3              8
4              16
5              32
6              64
7              128

To build any number from 0 to 255 from the above 8 bits, just
select the contributions you need to make. The bits will then be
1 if you use that contribution, and 0 if you don't.

This function will increment the "data" variable from 0 to 255
and repeat. When we send this value to the shift register and LEDs,
you can see the on-off pattern of the eight bits that make up the
byte. See http://www.arduino.cc/playground/Code/BitMath for more
information on binary numbers.
*/
```

```
void binaryCount()
{
  int delayTime = 1000; // time (milliseconds) to pause between LEDs
                        // make this smaller for faster switching

  // Send the data byte to the shift register:

  shiftOut(datapin, clockpin, MSBFIRST, data);

  // Toggle the latch pin to make the data appear at the outputs:

  digitalWrite(latchpin, HIGH);
  digitalWrite(latchpin, LOW);

  // Add one to data, and repeat!
  // (Because a byte type can only store numbers from 0 to 255,
  // if we add more than that, it will "roll around" back to 0
  // and start over).

  data++;

  // Delay so you can see what's going on:

  delay(delayTime);
}
```

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 Galileo 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(data, desiredPin, desiredState);`

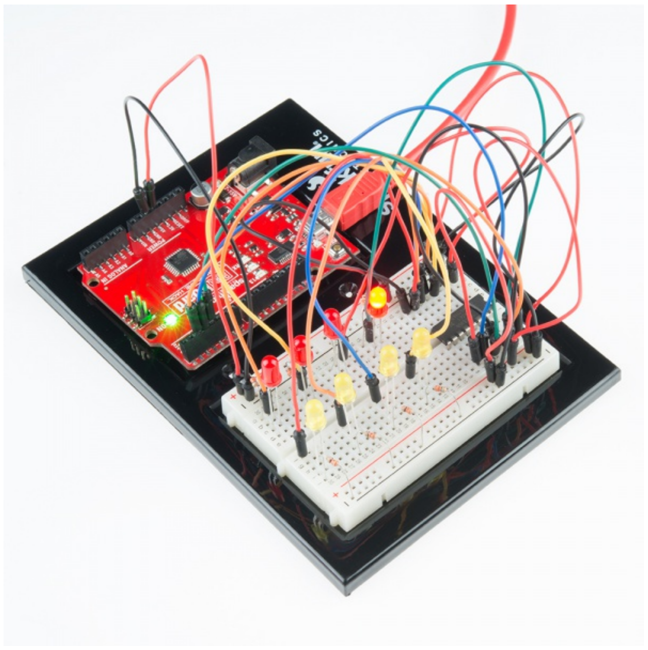
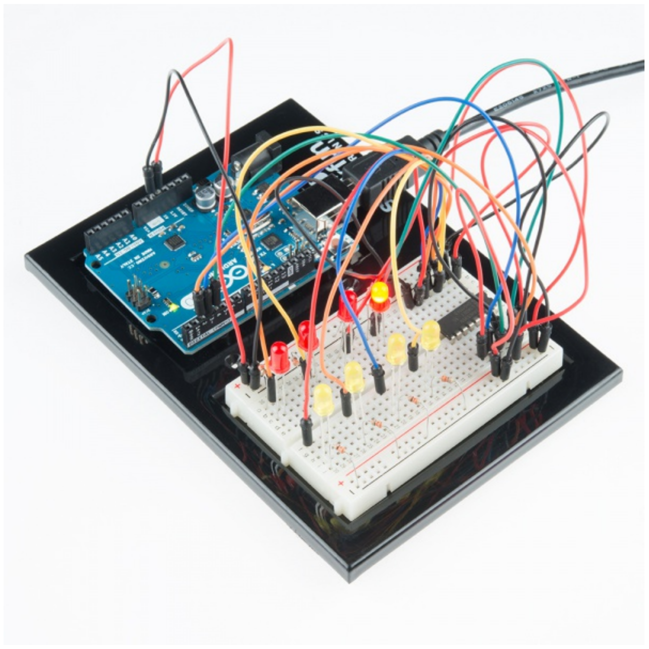
Bits are the smallest possible piece of memory in a computer; each one can store either a "1" or a "0". Larger numbers are stored as arrays of bits. Sometimes we want to manipulate these bits directly, for example now when we're sending eight bits to the shift register and we want to make them 1 or 0 to turn the LEDs on or off. The Galileo 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 experiment 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 section.

Real World Application

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

Troubleshooting

The Arduino'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**

Experiment 15: Using an LCD

Introduction

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 Needed

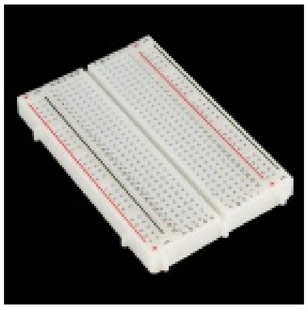
You will need the following parts:

- **1x** Breadboard
- **1x** RedBoard or Arduino Uno
- **1x** Potentiometer
- **1x** LCD
- **16x** Jumper Wires

Didn’t get the SIK?

If you are following through this experiment and didn’t get the SIK, we suggest using these parts:

Jumper Wires Standard 7" M/M Pack of 30
◉ PRT-11026
$4.95



Breadboard - Self-Adhesive (White)
◉ PRT-12002
$4.95

Basic 16x2 Character LCD - White on Black 5V
◉ LCD-00709
$15.95

Trimpot 10K with Knob
◉ COM-09806
$0.95

You will also need either a RedBoard or Arduino Uno R3.

SparkFun RedBoard - Programmed with Arduino
◉ DEV-12757
$19.95

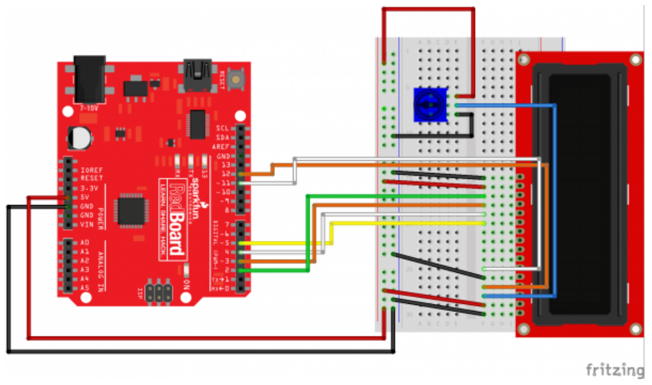
Arduino Uno- R3 SMD
◉ DEV-11224
$29.95

Hardware Hookup

Ready to start hooking everything up? Check out the Fritzing diagram below, to see how everything is connected.

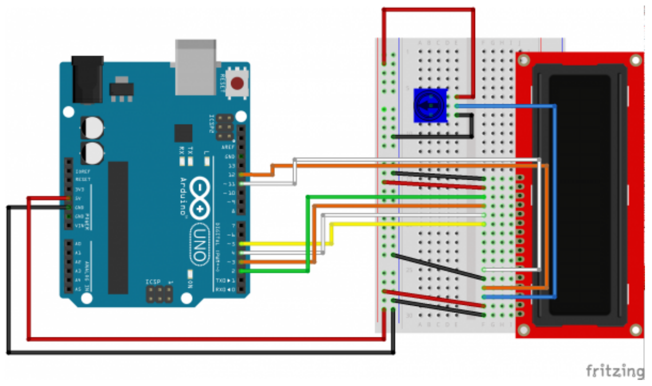
Polarized Components ⚠	Pay special attention to the component's markings indicating how to place it on the breadboard. Polarized components can only be connected to a circuit in one direction.

Fritzing Diagram for RedBoard



Having a hard time seeing the circuit? Click on the Fritzing diagram to see a bigger image.

Fritzing Diagram for Arduino



Open the Sketch

Open Up the Arduino IDE software on your computer. Coding in the Arduino language will control your circuit. Open the code for Circuit 15 by accessing the "SIK Guide Code" you downloaded and placed into your "Examples" folder earlier.

To open the code go to: **File > examples > SIK Guide Code > Circuit_15**

You can also copy and paste the following code into the Arduino IDE. Hit upload, and see what happens!

```
/*
SparkFun Inventor's Kit
Example sketch 15

LIQUID CRYSTAL DISPLAY (LCD)

  A Liquid Crystal Display (LCD) is a sophisticated module
  that can be used to display text or numeric data. The display
  included in your SIK features two lines of 16 characters, and
  a backlight so it can be used at night.

  If you've been using the Serial Monitor to output data,
  you'll find that a LCD provides many of the same benefits
  without needing to drag a large computer around.

  This sketch will show you how to connect an LCD to your Arduino
  and display any data you wish.

Hardware connections:

  The LCD has a 16-pin male header attached to it along the top
  edge. Pin 1 is the pin closest to the corner of the LCD.
  Pin 16 is the pin closest to the center of the LCD.

  Plug the LCD into your breadboard.

  As usual, you will want to connect the + and - power rails
  on the side of the breadboard to 5V and GND on your Arduino.

  Plug your 10K potentiometer into three unused rows on your
  breadboard. Connect one side of the potentiometer to 5V,
  and the other side to GND (it doesn't matter which). When you
  run this sketch, you'll use the potentiometer to adjust the
  contrast of the LCD so you can see the display.

  Now connect the LCD pins. Remember that pin 1 on the LCD
  is the one closest to the corner. Start there and work your
  way up.

  1 to GND
  2 to 5V
  3 to the center pin on the potentiometer
  4 to Galileo digital pin 12
  5 to GND
  6 to Galileo  digital pin 11
  7 (no connection)
  8 (no connection)
  9 (no connection)
  10 (no connection)
  11 to Galileo  digital pin 5
  12 to Galileo  digital pin 4
  13 to Galileo  digital pin 3
  14 to Galileo  digital pin 2
  15 to 5V
  16 to GND

  Once everything is connected, load this sketch into the
  Galileo, and adjust the potentiometer until the display
  is clear.

Library

  The LCD has a chip built into it that controls all the
  individual dots that make up the display, and obeys commands
  sent to it by the the Arduino. The chip knows the dot patterns
  that make up all the text characters, saving you a lot of work.

  To communicate with this chip, we'll use the LiquidCrystal
  library, which is one of the standard libraries that comes
  with the Arduino. This library does most of the hard work
  of interfacing to the LCD; all you need to pick a location
  on the display and send your data!

Tips

  The LCD comes with a protective film over the display that
  you can peel off (but be careful of the display surface as
  it scratches easily).

  The LCD has a backlight that will light up when you turn on
  your Arduino. If the backlight doesn't turn on, check your
  connections.

  As we said above, the potentiometer adjusts the contrast of
  the display. If you can't see anything when you run the sketch,
  turn the potentiometer's knob until the text is clear.

This sketch was written by SparkFun Electronics,
with lots of help from the Arduino community.
This code is completely free for any use.
Visit http://learn.sparkfun.com/products/2 for SIK information.
Visit http://www.arduino.cc to learn about the Arduino.

Version 1.0 2/2013 MDG
*/

// Load the LiquidCrystal library, which will give us
// commands to interface to the LCD:

#include <LiquidCrystal.h>

// Initialize the library with the pins we're using.
// (Note that you can use different pins if needed.)
// See http://arduino.cc/en/Reference/LiquidCrystal
// for more information:

LiquidCrystal lcd(12,11,5,4,3,2);

void setup()
```

```
{
  // The LiquidCrystal library can be used with many different
  // LCD sizes. We're using one that's 2 lines of 16 characters,
  // so we'll inform the library of that:

  lcd.begin(16, 2);

  // Data sent to the display will stay there until it's
  // overwritten or power is removed. This can be a problem
  // when you upload a new sketch to the Arduino but old data
  // remains on the display. Let's clear the LCD using the
  // clear() command from the LiquidCrystal library:

  lcd.clear();

  // Now we'll display a message on the LCD!

  // Just as with the Arduino IDE, there's a cursor that
  // determines where the data you type will appear. By default,
  // this cursor is invisible, though you can make it visible
  // with other library commands if you wish.

  // When the display powers up, the invisible cursor starts
  // on the top row and first column.

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

  // Adjusting the contrast (IMPORTANT!)

  // When you run the sketch for the first time, there's a
  // very good chance you won't see anything on the LCD display.
  // This is because the contrast likely won't be set correctly.
  // Don't worry, it's easy to set, and once you set it you won't
  // need to change it again.

  // Run the sketch, then turn the potentiometer until you can
  // clearly see the "hello, world!" text. If you still can't
  // see anything, check all of your connections, and ensure that
  // the sketch was successfully uploaded to the Arduino.
}

void loop()
{
  // You can move the invisible cursor to any location on the
  // LCD before sending data. Counting starts from 0, so the top
  // line is line 0 and the bottom line is line 1. Columns range
  // from 0 on the left side, to 15 on the right.

  // In additon to the "hello, world!" printed above, let's
  // display a running count of the seconds since the Arduino
  // was last reset. Note that the data you send to the display
  // will stay there unless you erase it by overwriting it or
  // sending a lcd.clear() command.

  // Here we'll set the invisible cursor to the first column
  // (column 0) of the second line (line 1):

  lcd.setCursor(0,1);

  // Now we'll print the number of seconds (millis() / 1000)
  // since the Arduino last reset:

  lcd.print(millis()/1000);

  // TIP: Since the numeric data we're sending is always growing
  // in length, new values will always overwrite the previous ones.
  // However, if you want to display varying or decreasing numbers
  // like a countdown, you'll find that the display will leave
  // "orphan" characters when the new value is shorter than the
  // old one.

  // To prevent this, you'll need to erase the old number before
  // writing the new one. You can do this by overwriting the
  // last number with spaces. If you erase the old number and
  // immediately write the new one, the momentary erase won't
  // be noticeable. Here's a typical sequence of code:

  // lcd.setCursor(0,1);   // Set the cursor to the position
  // lcd.print("       "); // Erase the largest possible number
  // lcd.setCursor(0,1);   // Reset the cursor to the original position
  // lcd.print(millis()/1000); // Print our value

  // NEXT STEPS:

  // Now you know the basics of hooking up an LCD to the Arduino,
  // and sending text and numeric data to the display!

  // The LCD library has many commands for turning the
  // cursor on and off, scrolling the screen, etc. See:
  // http://arduino.cc/en/Reference/LiquidCrystal
  // for more information.

  // Arduino also comes with a number of built-in examples
  // showing off the features of the LiquidCrystal library.
  // These are locted in the file/examples/LiquidCrystal menu.

  // Have fun, and let us know what you create!
  // Your friends at SparkFun.
}
```

Code To Note

```
#include <LiquidCrystal.h>
```

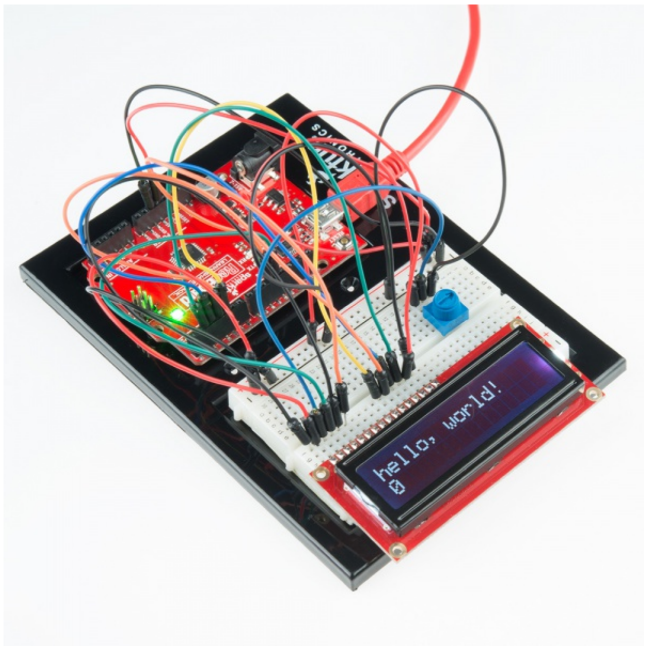
This bit of code tells your Arduino IDE to include the library for a simple LCD display. Without it, none of the commands will work, so make sure you include it!

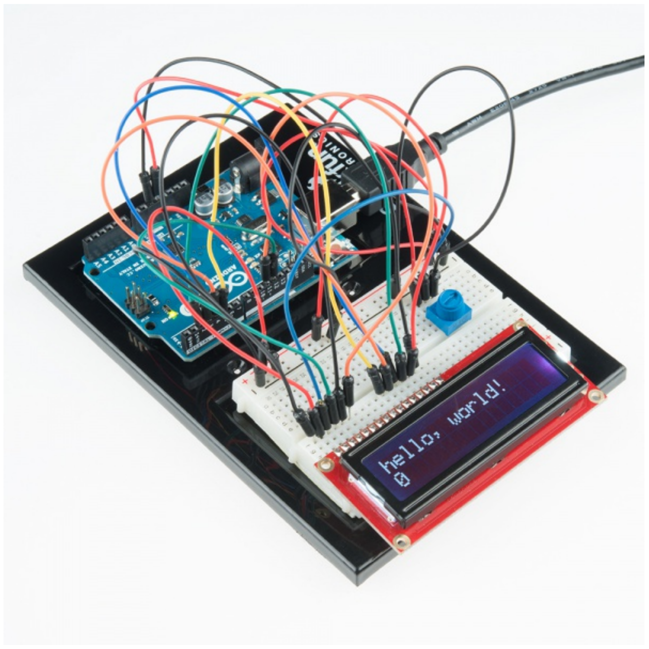
```
lcd.print("hello, world!");
```

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.





Real World Application

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

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.

Experiment 16: Simon Says

Introduction

Now that we've learned all the basics behind the components in the SIK experiments, let's put them all together to make 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 the RedBoard or Arduino Uno R3.

Parts Needed

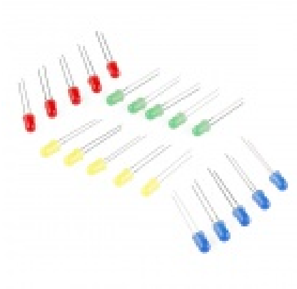
You will need the following parts:

- **1x** Breadboard
- **1x** RedBoard or Arduino Uno
- **4x** LEDs
- **1x** Piezo Buzzer
- **4x** 330Ω Resistors
- **4x** Push Buttons
- **17x** Jumper Wires

Didn't get the SIK?

If you are following through this experiment and didn't get the SIK, we suggest using these parts:

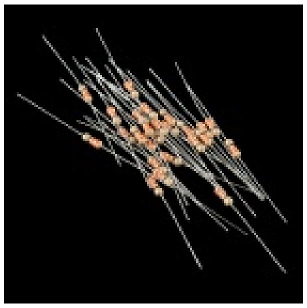
Jumper Wires Standard 7" M/M Pack of 30
◉ PRT-11026
$4.95



LED - Assorted (20 pack)
◉ COM-12062
$2.95

Momentary Pushbutton Switch - 12mm Square
◉ COM-09190
$0.50

Piezo Speaker - PC Mount 12mm 2.048kHz
◉ COM-07950
$1.95



Resistor 330 Ohm 1/6 Watt PTH - 20 pack

◉ COM-11507

$0.95

You will also need either a RedBoard or Arduino Uno R3.

SparkFun RedBoard - Programmed with Arduino

◉ DEV-12757

$19.95

Arduino Uno- R3 SMD

◉ DEV-11224

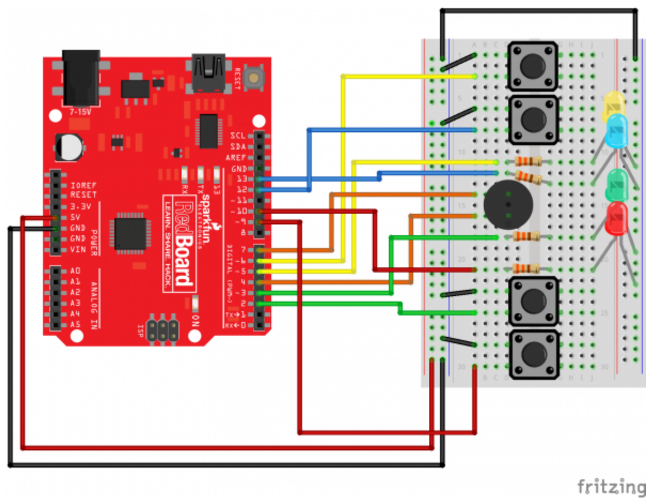
$29.95

Hardware Hookup

Ready to start hooking everything up? Check out the Fritzing diagram below, to see how everything is connected.

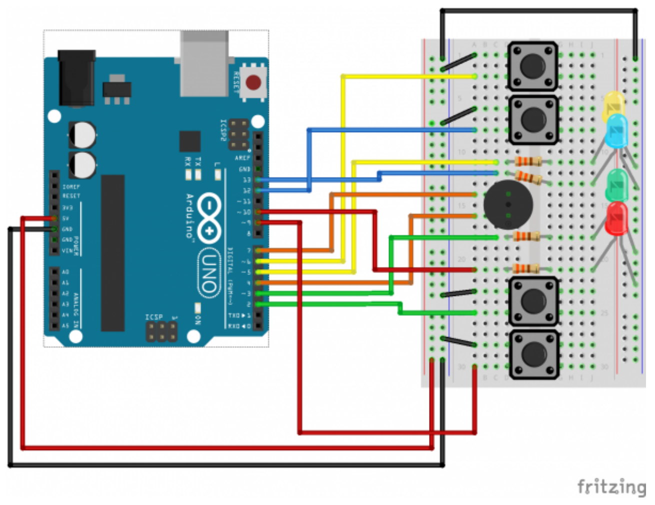
Polarized Components ⚠	Pay special attention to the component’s markings indicating how to place it on the breadboard. Polarized components can only be connected to a circuit in one direction.

Fritzing Diagram for RedBoard



Having a hard time seeing the circuit? Click on the Fritzing diagram to see a bigger image.

Fritzing Diagram for Arduino



Open the Sketch

Open Up the Arduino IDE software on your computer. Coding in the Arduino language will control your circuit. Open the code for Circuit 16 by accessing the “SIK Guide Code” you downloaded and placed into your “Examples” folder earlier.

To open the code go to: **File > examples > SIK Guide Code > Circuit_16**

You can also copy and paste the following code into the Arduino IDE. Hit upload, and see what happens!

```
/*
 SparkFun Inventor's Kit
 Example sketch 16
 Spark Fun Electronics
 Oct. 7, 2014

 Simon Says is a memory game. Start the game by pressing one of the four buttons.
When a button lights up,
 press the button, repeating the sequence. The sequence will get longer and longer
. The game is won after
 13 rounds.
 Generates random sequence, plays music, and displays button lights.

 Simon tones from Wikipedia
 - A (red, upper left) - 440Hz - 2.272ms - 1.136ms pulse
 - a (green, upper right, an octave higher than A) - 880Hz - 1.136ms,
 0.568ms pulse
 - D (blue, lower left, a perfect fourth higher than the upper left)
 587.33Hz - 1.702ms - 0.851ms pulse
 - G (yellow, lower right, a perfect fourth higher than the lower left) -
 784Hz - 1.276ms - 0.638ms pulse

 Simon Says game originally written in C for the PIC16F88.
 Ported for the ATmega168, then ATmega328, then Arduino 1.0.
 Fixes and cleanup by Joshua Neal <joshua[at]trochotron.com>

 This sketch was written by SparkFun Electronics,
 with lots of help from the Arduino community.
 This code is completely free for any use.
 Visit http://www.arduino.cc to learn about the Arduino.
 */

/*************************************************
* Public Constants
*************************************************/
#define NOTE_B0 31
#define NOTE_C1 33
#define NOTE_CS1 35
#define NOTE_D1 37
#define NOTE_DS1 39
#define NOTE_E1 41
#define NOTE_F1 44
#define NOTE_FS1 46
#define NOTE_G1 49
#define NOTE_GS1 52
#define NOTE_A1 55
#define NOTE_AS1 58
#define NOTE_B1 62
#define NOTE_C2 65
#define NOTE_CS2 69
#define NOTE_D2 73
#define NOTE_DS2 78
#define NOTE_E2 82
#define NOTE_F2 87
#define NOTE_FS2 93
#define NOTE_G2 98
#define NOTE_GS2 104
#define NOTE_A2 110
#define NOTE_AS2 117
#define NOTE_B2 123
#define NOTE_C3 131
```

```
#define NOTE_CS3 139
#define NOTE_D3 147
#define NOTE_DS3 156
#define NOTE_E3 165
#define NOTE_F3 175
#define NOTE_FS3 185
#define NOTE_G3 196
#define NOTE_GS3 208
#define NOTE_A3 220
#define NOTE_AS3 233
#define NOTE_B3 247
#define NOTE_C4 262
#define NOTE_CS4 277
#define NOTE_D4 294
#define NOTE_DS4 311
#define NOTE_E4 330
#define NOTE_F4 349
#define NOTE_FS4 370
#define NOTE_G4 392
#define NOTE_GS4 415
#define NOTE_A4 440
#define NOTE_AS4 466
#define NOTE_B4 494
#define NOTE_C5 523
#define NOTE_CS5 554
#define NOTE_D5 587
#define NOTE_DS5 622
#define NOTE_E5 659
#define NOTE_F5 698
#define NOTE_FS5 740
#define NOTE_G5 784
#define NOTE_GS5 831
#define NOTE_A5 880
#define NOTE_AS5 932
#define NOTE_B5 988
#define NOTE_C6 1047
#define NOTE_CS6 1109
#define NOTE_D6 1175
#define NOTE_DS6 1245
#define NOTE_E6 1319
#define NOTE_F6 1397
#define NOTE_FS6 1480
#define NOTE_G6 1568
#define NOTE_GS6 1661
#define NOTE_A6 1760
#define NOTE_AS6 1865
#define NOTE_B6 1976
#define NOTE_C7 2093
#define NOTE_CS7 2217
#define NOTE_D7 2349
#define NOTE_DS7 2489
```

```
#define NOTE_E7 2637
#define NOTE_F7 2794
#define NOTE_FS7 2960
#define NOTE_G7 3136
#define NOTE_GS7 3322
#define NOTE_A7 3520
#define NOTE_AS7 3729
#define NOTE_B7 3951
#define NOTE_C8 4186
#define NOTE_CS8 4435
#define NOTE_D8 4699
#define NOTE_DS8 4978

#define CHOICE_OFF      0 //Used to control LEDs
#define CHOICE_NONE     0 //Used to check buttons
#define CHOICE_RED  (1 << 0)
#define CHOICE_GREEN    (1 << 1)
#define CHOICE_BLUE (1 << 2)
#define CHOICE_YELLOW   (1 << 3)

#define LED_RED     10
#define LED_GREEN   3
#define LED_BLUE    13
#define LED_YELLOW  5

// Button pin definitions
#define BUTTON_RED    9
#define BUTTON_GREEN  2
#define BUTTON_BLUE   12
#define BUTTON_YELLOW 6

// Buzzer pin definitions
#define BUZZER1  4
#define BUZZER2  7

// Define game parameters
#define ROUNDS_TO_WIN      13 //Number of rounds to succesfully remember before yo
u win. 13 is do-able.
#define ENTRY_TIME_LIMIT   3000 //Amount of time to press a button before game tim
es out. 3000ms = 3 sec

#define MODE_MEMORY  0
#define MODE_BATTLE  1
#define MODE_BEEGEES 2

// Game state variables
byte gameMode = MODE_MEMORY; //By default, let's play the memory game
byte gameBoard[32]; //Contains the combination of buttons as we advance
byte gameRound = 0; //Counts the number of succesful rounds the player has made it
 through
```

```
void setup()
{
  //Setup hardware inputs/outputs. These pins are defined in the hardware_versions
 header file

  //Enable pull ups on inputs
  pinMode(BUTTON_RED, INPUT_PULLUP);
  pinMode(BUTTON_GREEN, INPUT_PULLUP);
  pinMode(BUTTON_BLUE, INPUT_PULLUP);
  pinMode(BUTTON_YELLOW, INPUT_PULLUP);

  pinMode(LED_RED, OUTPUT);
  pinMode(LED_GREEN, OUTPUT);
  pinMode(LED_BLUE, OUTPUT);
  pinMode(LED_YELLOW, OUTPUT);

  pinMode(BUZZER1, OUTPUT);
  pinMode(BUZZER2, OUTPUT);

  //Mode checking
  gameMode = MODE_MEMORY; // By default, we're going to play the memory game

  // Check to see if the lower right button is pressed
  if (checkButton() == CHOICE_YELLOW) play_beegees();

  // Check to see if upper right button is pressed
  if (checkButton() == CHOICE_GREEN)
  {
    gameMode = MODE_BATTLE; //Put game into battle mode

    //Turn on the upper right (green) LED
    setLEDs(CHOICE_GREEN);
    toner(CHOICE_GREEN, 150);

    setLEDs(CHOICE_RED | CHOICE_BLUE | CHOICE_YELLOW); // Turn on the other LEDs u
ntil you release button

    while(checkButton() != CHOICE_NONE) ; // Wait for user to stop pressing button

    //Now do nothing. Battle mode will be serviced in the main routine
  }

  play_winner(); // After setup is complete, say hello to the world
}

void loop()
{
  attractMode(); // Blink lights while waiting for user to press a button

  // Indicate the start of game play
  setLEDs(CHOICE_RED | CHOICE_GREEN | CHOICE_BLUE | CHOICE_YELLOW); // Turn all LE
```

```
Ds on
  delay(1000);
  setLEDs(CHOICE_OFF); // Turn off LEDs
  delay(250);

  if (gameMode == MODE_MEMORY)
  {
    // Play memory game and handle result
    if (play_memory() == true)
      play_winner(); // Player won, play winner tones
    else
      play_loser(); // Player lost, play loser tones
  }

  if (gameMode == MODE_BATTLE)
  {
    play_battle(); // Play game until someone loses

    play_loser(); // Player lost, play loser tones
  }
}

//-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
//The following functions are related to game play only

// Play the regular memory game
// Returns 0 if player loses, or 1 if player wins
boolean play_memory(void)
{
  randomSeed(millis()); // Seed the random generator with random amount of millis(
)

  gameRound = 0; // Reset the game to the beginning

  while (gameRound < ROUNDS_TO_WIN)
  {
    add_to_moves(); // Add a button to the current moves, then play them back

    playMoves(); // Play back the current game board

    // Then require the player to repeat the sequence.
    for (byte currentMove = 0 ; currentMove < gameRound ; currentMove++)
    {
      byte choice = wait_for_button(); // See what button the user presses

      if (choice == 0) return false; // If wait timed out, player loses

      if (choice != gameBoard[currentMove]) return false; // If the choice is inco
rect, player loses
    }
    delay(1000); // Player was correct, delay before playing moves
  }

  return true; // Player made it through all the rounds to win!
}

// Play the special 2 player battle mode
// A player begins by pressing a button then handing it to the other player
// That player repeats the button and adds one, then passes back.
// This function returns when someone loses
boolean play_battle(void)
{
  gameRound = 0; // Reset the game frame back to one frame

  while (1) // Loop until someone fails
  {
    byte newButton = wait_for_button(); // Wait for user to input next move
    gameBoard[gameRound++] = newButton; // Add this new button to the game array

    // Then require the player to repeat the sequence.
    for (byte currentMove = 0 ; currentMove < gameRound ; currentMove++)
    {
      byte choice = wait_for_button();

      if (choice == 0) return false; // If wait timed out, player loses.

      if (choice != gameBoard[currentMove]) return false; // If the choice is inco
rect, player loses.
    }

    delay(100); // Give the user an extra 100ms to hand the game to the other play
er
  }

  return true; // We should never get here
}

// Plays the current contents of the game moves
void playMoves(void)
{
  for (byte currentMove = 0 ; currentMove < gameRound ; currentMove++)
  {
    toner(gameBoard[currentMove], 150);

    // Wait some amount of time between button playback
    // Shorten this to make game harder
    delay(150); // 150 works well. 75 gets fast.
  }
}

// Adds a new random button to the game sequence, by sampling the timer
```

```
void add_to_moves(void)
{
  byte newButton = random(0, 4); //min (included), max (exluded)

  // We have to convert this number, 0 to 3, to CHOICEs
  if(newButton == 0) newButton = CHOICE_RED;
  else if(newButton == 1) newButton = CHOICE_GREEN;
  else if(newButton == 2) newButton = CHOICE_BLUE;
  else if(newButton == 3) newButton = CHOICE_YELLOW;

  gameBoard[gameRound++] = newButton; // Add this new button to the game array
}

//-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
//The following functions control the hardware

// Lights a given LEDs
// Pass in a byte that is made up from CHOICE_RED, CHOICE_YELLOW, etc
void setLEDs(byte leds)
{
  if ((leds & CHOICE_RED) != 0)
    digitalWrite(LED_RED, HIGH);
  else
    digitalWrite(LED_RED, LOW);

  if ((leds & CHOICE_GREEN) != 0)
    digitalWrite(LED_GREEN, HIGH);
  else
    digitalWrite(LED_GREEN, LOW);

  if ((leds & CHOICE_BLUE) != 0)
    digitalWrite(LED_BLUE, HIGH);
  else
    digitalWrite(LED_BLUE, LOW);

  if ((leds & CHOICE_YELLOW) != 0)
    digitalWrite(LED_YELLOW, HIGH);
  else
    digitalWrite(LED_YELLOW, LOW);
}

// Wait for a button to be pressed.
// Returns one of LED colors (LED_RED, etc.) if successful, 0 if timed out
byte wait_for_button(void)
{
  long startTime = millis(); // Remember the time we started the this loop

  while ( (millis() - startTime) < ENTRY_TIME_LIMIT) // Loop until too much time h
as passed
  {
    byte button = checkButton();
    if (button != CHOICE_NONE)
    {
      toner(button, 150); // Play the button the user just pressed

      while(checkButton() != CHOICE_NONE) ;  // Now let's wait for user to release
 button

      delay(10); // This helps with debouncing and accidental double taps

      return button;
    }

  }

  return CHOICE_NONE; // If we get here, we've timed out!
}

// Returns a '1' bit in the position corresponding to CHOICE_RED, CHOICE_GREEN, et
c.
byte checkButton(void)
{
  if (digitalRead(BUTTON_RED) == 0) return(CHOICE_RED);
  else if (digitalRead(BUTTON_GREEN) == 0) return(CHOICE_GREEN);
  else if (digitalRead(BUTTON_BLUE) == 0) return(CHOICE_BLUE);
  else if (digitalRead(BUTTON_YELLOW) == 0) return(CHOICE_YELLOW);

  return(CHOICE_NONE); // If no button is pressed, return none
}

// Light an LED and play tone
// Red, upper left:     440Hz - 2.272ms - 1.136ms pulse
// Green, upper right:  880Hz - 1.136ms - 0.568ms pulse
// Blue, lower left:    587.33Hz - 1.702ms - 0.851ms pulse
// Yellow, lower right: 784Hz - 1.276ms - 0.638ms pulse
void toner(byte which, int buzz_length_ms)
{
  setLEDs(which); //Turn on a given LED

  //Play the sound associated with the given LED
  switch(which)
  {
  case CHOICE_RED:
    buzz_sound(buzz_length_ms, 1136);
    break;
  case CHOICE_GREEN:
    buzz_sound(buzz_length_ms, 568);
    break;
  case CHOICE_BLUE:
    buzz_sound(buzz_length_ms, 851);
    break;
  case CHOICE_YELLOW:
    buzz_sound(buzz_length_ms, 638);
    break;
  }

  setLEDs(CHOICE_OFF); // Turn off all LEDs
}

// Toggle buzzer every buzz_delay_us, for a duration of buzz_length_ms.
void buzz_sound(int buzz_length_ms, int buzz_delay_us)
{
  // Convert total play time from milliseconds to microseconds
  long buzz_length_us = buzz_length_ms * (long)1000;

  // Loop until the remaining play time is less than a single buzz_delay_us
  while (buzz_length_us > (buzz_delay_us * 2))
  {
    buzz_length_us -= buzz_delay_us * 2; //Decrease the remaining play time

    // Toggle the buzzer at various speeds
    digitalWrite(BUZZER1, LOW);
    digitalWrite(BUZZER2, HIGH);
    delayMicroseconds(buzz_delay_us);

    digitalWrite(BUZZER1, HIGH);
    digitalWrite(BUZZER2, LOW);
    delayMicroseconds(buzz_delay_us);
  }
}

// Play the winner sound and lights
void play_winner(void)
{
  setLEDs(CHOICE_GREEN | CHOICE_BLUE);
  winner_sound();
  setLEDs(CHOICE_RED | CHOICE_YELLOW);
  winner_sound();
  setLEDs(CHOICE_GREEN | CHOICE_BLUE);
  winner_sound();
  setLEDs(CHOICE_RED | CHOICE_YELLOW);
  winner_sound();
}

// Play the winner sound
// This is just a unique (annoying) sound we came up with, there is no magic to it
void winner_sound(void)
{
  // Toggle the buzzer at various speeds
  for (byte x = 250 ; x > 70 ; x--)
  {
    for (byte y = 0 ; y < 3 ; y++)
    {
      digitalWrite(BUZZER2, HIGH);
      digitalWrite(BUZZER1, LOW);
      delayMicroseconds(x);

      digitalWrite(BUZZER2, LOW);
      digitalWrite(BUZZER1, HIGH);
      delayMicroseconds(x);
    }
  }
}

// Play the loser sound/lights
void play_loser(void)
{
  setLEDs(CHOICE_RED | CHOICE_GREEN);
  buzz_sound(255, 1500);

  setLEDs(CHOICE_BLUE | CHOICE_YELLOW);
  buzz_sound(255, 1500);

  setLEDs(CHOICE_RED | CHOICE_GREEN);
  buzz_sound(255, 1500);

  setLEDs(CHOICE_BLUE | CHOICE_YELLOW);
  buzz_sound(255, 1500);
}

// Show an "attract mode" display while waiting for user to press button.
void attractMode(void)
{
  while(1)
  {
    setLEDs(CHOICE_RED);
    delay(100);
    if (checkButton() != CHOICE_NONE) return;

    setLEDs(CHOICE_BLUE);
    delay(100);
    if (checkButton() != CHOICE_NONE) return;

    setLEDs(CHOICE_GREEN);
    delay(100);
    if (checkButton() != CHOICE_NONE) return;

    setLEDs(CHOICE_YELLOW);
    delay(100);
    if (checkButton() != CHOICE_NONE) return;
  }
}
```

```
//-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
// The following functions are related to Beegees Easter Egg only

// Notes in the melody. Each note is about an 1/8th note, "0"s are rests.
int melody[] = {
  NOTE_G4, NOTE_A4, 0, NOTE_C5, 0, 0, NOTE_G4, 0, 0, 0,
  NOTE_E4, 0, NOTE_D4, NOTE_E4, NOTE_G4, 0,
  NOTE_D4, NOTE_E4, 0, NOTE_G4, 0, 0,
  NOTE_D4, 0, NOTE_E4, 0, NOTE_G4, 0, NOTE_A4, 0, NOTE_C5, 0};

int noteDuration = 115; // This essentially sets the tempo, 115 is just about righ
t for a disco groove :)
int LEDnumber = 0; // Keeps track of which LED we are on during the beegees loop

// Do nothing but play bad beegees music
// This function is activated when user holds bottom right button during power up
void play_beegees()
{
  //Turn on the bottom right (yellow) LED
  setLEDs(CHOICE_YELLOW);
  toner(CHOICE_YELLOW, 150);

  setLEDs(CHOICE_RED | CHOICE_GREEN | CHOICE_BLUE); // Turn on the other LEDs unti
l you release button

  while(checkButton() != CHOICE_NONE) ; // Wait for user to stop pressing button

  setLEDs(CHOICE_NONE); // Turn off LEDs

  delay(1000); // Wait a second before playing song

  digitalWrite(BUZZER1, LOW); // setup the "BUZZER1" side of the buzzer to stay lo
w, while we play the tone on the other pin.

  while(checkButton() == CHOICE_NONE) //Play song until you press a button
  {
    // iterate over the notes of the melody:
    for (int thisNote = 0; thisNote < 32; thisNote++) {
      changeLED();
      tone(BUZZER2, melody[thisNote],noteDuration);
      // to distinguish the notes, set a minimum time between them.
      // the note's duration + 30% seems to work well:
      int pauseBetweenNotes = noteDuration * 1.30;
      delay(pauseBetweenNotes);
      // stop the tone playing:
      noTone(BUZZER2);
    }
  }
}

// Each time this function is called the board moves to the next LED
```

```
void changeLED(void)
{
  setLEDs(1 << LEDnumber); // Change the LED

  LEDnumber++; // Goto the next LED
  if(LEDnumber > 3) LEDnumber = 0; // Wrap the counter if needed
}
```

Code To Note

`#define`

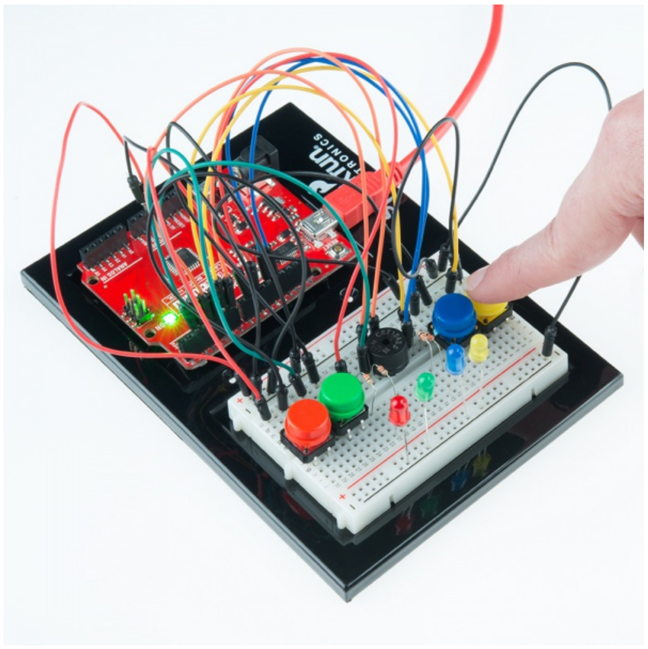
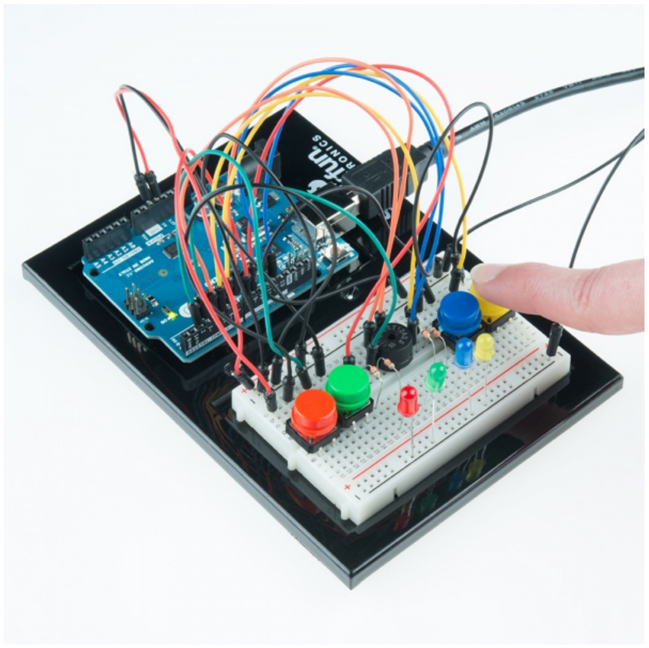
The `#define` statement is used to create constants in your code. Constants are variables that will likely only have one value during the lifespan of your code. Thus, you can assign constants a value, and then use them throughout your code wherever. Then, if you need to change that value, you can change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

Once the code is uploaded, 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.

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.

Troubleshooting

Only Half the Circuit Works

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

Resources and Going Further

There are tons of sensors and shields you can hookup to an Arduino that will help take your projects to the next level. Here’s some further reading that may help you along in learning more about the world of electronics. For more info on Arduino, check out these tutorials:

- Intro to Arduino Curriculum
- Arduino Comparison Guide
- Arduino Shields
- Installing Arduino
- Installing an Arduino Library
- Arduino Data Types

For more hardware related tutorials, give these a read:

- Breadboards
- Working with Wire
- Sewing with conductive thread
- How do I power my project?