SparkFun Inventor's Kit

VERSION 4.1

Your Guide to the SIK
for the SparkFun RedBoard

sparkfun
START SOMETHING

SparkFun Inventor's Kit, Version 4.1

WELCOME TO THE SPARKFUN INVENTOR'S KIT (SIK) GUIDE.

This is your map for navigating beginning embedded electronics. This booklet contains all the information you will need to build five projects encompassing the 16 circuits of the SIK for the SparkFun RedBoard. At the center of this manual is one core philosophy: that anyone can (and should) play around with electronics. When you're done with this guide, you will have built five great projects and acquired the know-how to create countless more. Now enough talk — let's start something!

For a digital version of this guide with more in-depth information for each circuit and links explaining relevant terms and concepts, visit:

sparkfun.com/SIKguide

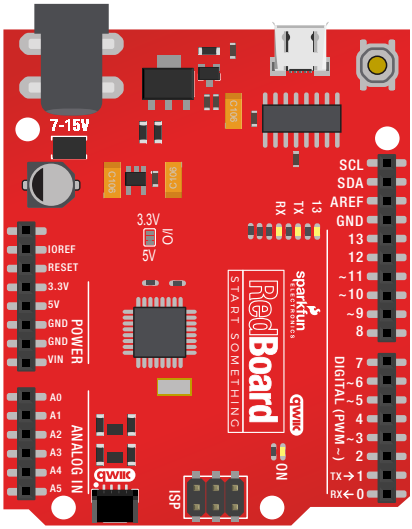
Contents

The RedBoard Platform

THE DIY REVOLUTION: At SparkFun we believe that an understanding of electronics is a core literacy that opens up a world of opportunities in the fields of robotics, Internet of Things (IoT), engineering, fashion, medical industries, environmental sciences, performing arts and more. This guide is designed to explore the connection between software and hardware, introducing Arduino code and SparkFun parts as they are used in the context of building engaging projects. The circuits in this guide progress in difficulty as new concepts and components are introduced. Completing each circuit means much more than just "experimenting"; you will walk away with a fun project you can use — and a sense of accomplishment that is just the beginning of your electronics journey. At the end of each circuit, you'll find coding challenges that extend your learning and fuel ongoing innovation.

A COMPUTER FOR THE PHYSICAL WORLD

The SparkFun RedBoard Qwiic is your development platform. At its roots, the RedBoard is essentially a small, portable computer, also known as a microcontroller. 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 blinking an LED light or spinning an electric motor). That's where the term "physical computing" comes in; this board is capable of taking the world of electronics and relating it to the physical world in a real and tangible way.



THE SPARKFUN REDBOARD QWIIC is one of a multitude of development boards based on the ATmega328 microprocessor. It has 14 digital input/output pins (six of which can be PWM outputs), six analog inputs, a 16MHz crystal oscillator, a USB connection, a power jack, and a reset button. You'll learn more about each of the RedBoard's features as you progress through this guide.

> **NOTE:** For the remainder of this guide, in the interest of brevity, we will refer to the RedBoard Qwiic simply as the "RedBoard."

Baseplate Assembly

Before you can build circuits, you'll want to first assemble the breadboard baseplate. This apparatus makes circuit building easier by keeping the RedBoard microcontroller and the breadboard connected without the worry of disconnecting or damaging your circuit.

TO BEGIN, collect your parts: the RedBoard, breadboard, included screwdriver, baseplate and two baseplate screws.

Your screwdriver has both Phillips and flatheads. If it is not already in position, pull the shaft out and switch to the Phillips head.

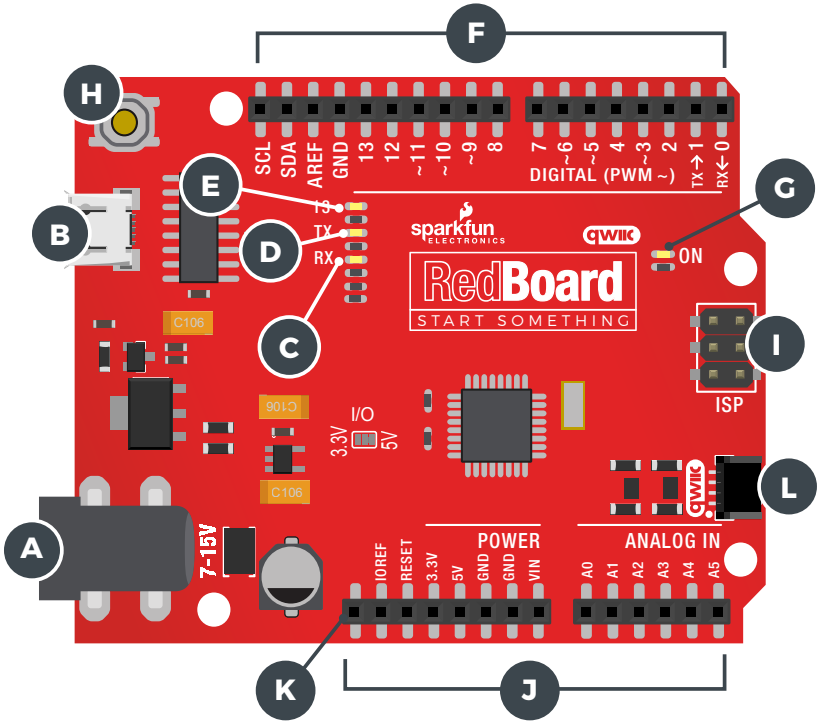
PEEL the adhesive backing off the breadboard.

CAREFULLY ALIGN the breadboard over its spot on the baseplate. The text on the breadboard should face the same direction as the text on the baseplate. Firmly press the breadboard to the baseplate to adhere it.

ALIGN THE REDBOARD with its spot on the baseplate. The text on it should face the same direction as the text on the breadboard and the baseplate. Using one of the two included screws, affix the RedBoard to one of the four stand-off holes found on the baseplate. The plastic holes are not threaded, so you will need to apply pressure as you twist the screwdriver.

Screw the second screw in the stand-off hole diagonally across from the first. With that, your baseplate is now assembled.

Anatomy of the SparkFun RedBoard



REDBOARD HARDWARE OVERVIEW

A	**POWER IN (BARREL JACK)**	Can be used with either a 9V or 12V “wall-wart” or a battery pack.
B	**POWER IN (USB PORT)**	Provides power and communicates with your board when plugged into your computer via USB.
C	**LED (RX: RECEIVING)**	Shows when the USB-to-serial chip is receiving data bits from the computer.
D	**LED (TX: TRANSMITTING)**	Shows when the USB-to-serial chip is transmitting data bits to the computer.
E	**ONBOARD LED PIN D13**	This LED, connected to digital pin 13, can be controlled in your program and is great for troubleshooting.
F	**PINS AREF, GROUND, DIGITAL, RX, TX, SDA, SCL**	These pins can be used for inputs, outputs, power and ground.
G	**POWER LED**	Illuminated when the board is connected to a power source.
H	**RESET BUTTON**	A manual reset switch that will restart the RedBoard and your code.
I	**ISP HEADER**	This is the In-System Programming header. It is used to program the ATMega328 directly. It will not be used in this guide.
J	**ANALOG IN, VOLTAGE IN, GROUND, 3.3 AND 5V OUT, RESET**	The power bus has pins to power your circuits with various voltages. The analog inputs allow you to read analog signals.
K	**RFU**	This stands for Reserved for Future Use.
L	**QWIIC® CONNECTOR**	SparkFun Qwiic® Cable Connector for I²C Devices. This connection will not be used in this guide.

Anatomy of the Breadboard

A breadboard is a circuit-building platform that allows you to connect multiple components without using a soldering iron.

POWER BUS

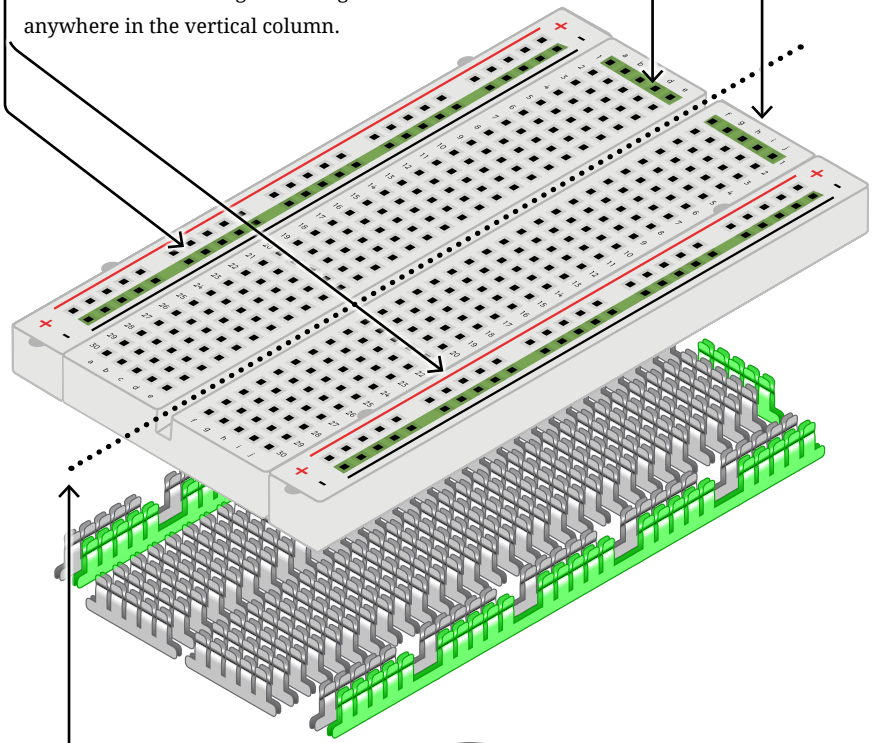
Each side of the breadboard has a pair of vertical connections marked – and +

+ POWER: Each + sign runs power anywhere in the vertical column.

– GROUND: Each – sign runs to ground anywhere in the vertical column.

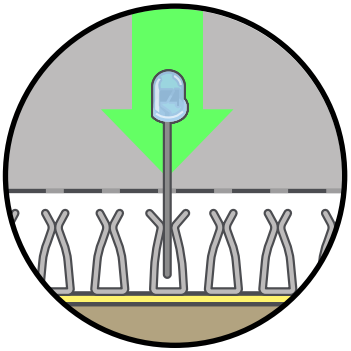
HORIZONTAL ROWS

Each series of 5 sockets marked a–e and f–j are connected. Components connected to a row will be connected to any other part inserted in the same row.



CENTERLINE

This line divides the breadboard in half, restricting electricity to one half or the other.



MAKING A CONNECTION

Most of the components in this kit are breadboard-friendly and can be easily installed and removed.

The Arduino IDE

IN ORDER TO GET YOUR REDBOARD UP AND RUNNING, you'll need to download the newest version of the Arduino software from **www.arduino.cc** (it's free!). This software, known as the Arduino IDE (Integrated Development Environment), will allow you to program the RedBoard to do exactly what you want. It's like a word processor for coding. With an internet-capable computer, open up your favorite browser and type the following URL into the address bar:

DOWNLOAD THE SOFTWARE HERE: arduino.cc/downloads

1. DOWNLOAD AND INSTALL ARDUINO IDE

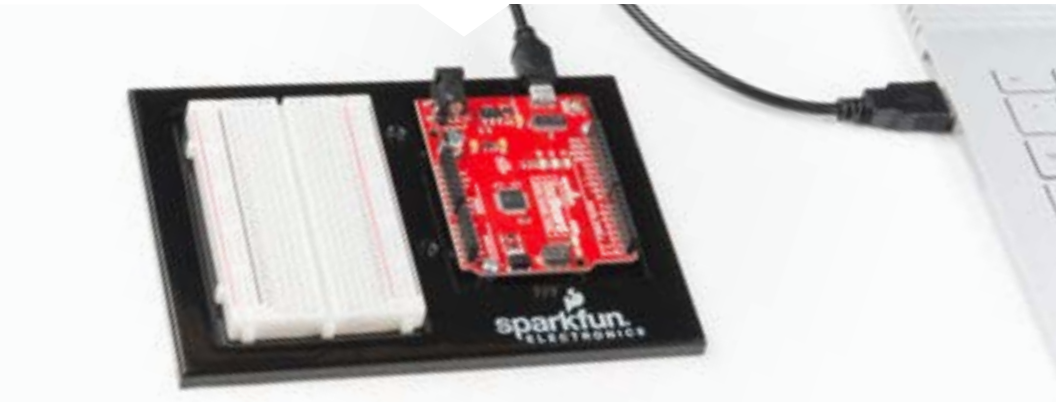
Select the installer option appropriate for the operating system you are using. Once finished downloading, open the file and follow the instructions to install.

2. INSTALL USB DRIVERS

In order for the RedBoard hardware to work with your computer's operating system, you will need to install a few drivers. Please go to **www.sparkfun.com/ch340** for specific instructions on how to install the USB drivers onto your computer.

3. CONNECT THE REDBOARD TO A COMPUTER

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



4. DOWNLOAD AND INSTALL THE SIK CODE

Each of the circuits you will build in the SparkFun Inventor's Kit has an Arduino code sketch already written for it. This guide will show you how to manipulate that code to control your hardware.

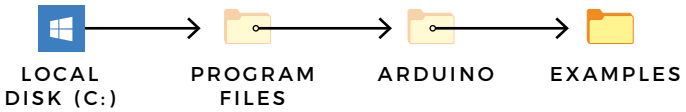
DOWNLOAD THE CODE HERE: sparkfun.com/SIKcode

COPY "SIK GUIDE CODE" INTO "EXAMPLES" LIBRARY IN ARDUINO FOLDER

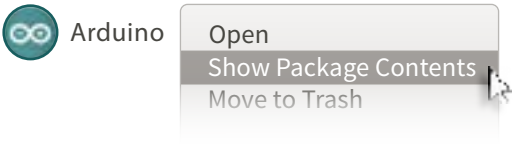
Your browser will download the code automatically or ask you if you would like to download the .zip file. Select "Save File." Locate the code (usually in your browser's "Downloads" folder). You'll need to relocate it to the "Examples" subfolder in your Arduino IDE installation in order for it to function properly.

Unzip the file **"SIK GUIDE CODE."** It should be located in your browser's "Downloads" folder. Right-click (or ctrl + click) the zipped folder and choose **"unzip."**

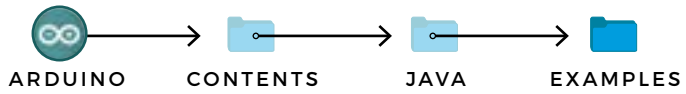
WINDOWS: Copy or move the unzipped "SIK Guide Code" files from "Downloads" to the Arduino application's "Examples" folder.



MAC OS: Find "Arduino" in your "Applications" folder in Finder. Right-click (ctrl + click) on "Arduino" and select "Show Package Contents."



Copy or move the unzipped "SIK Guide Code" folder from your "Downloads" folder into the Arduino application's folder named "Examples."

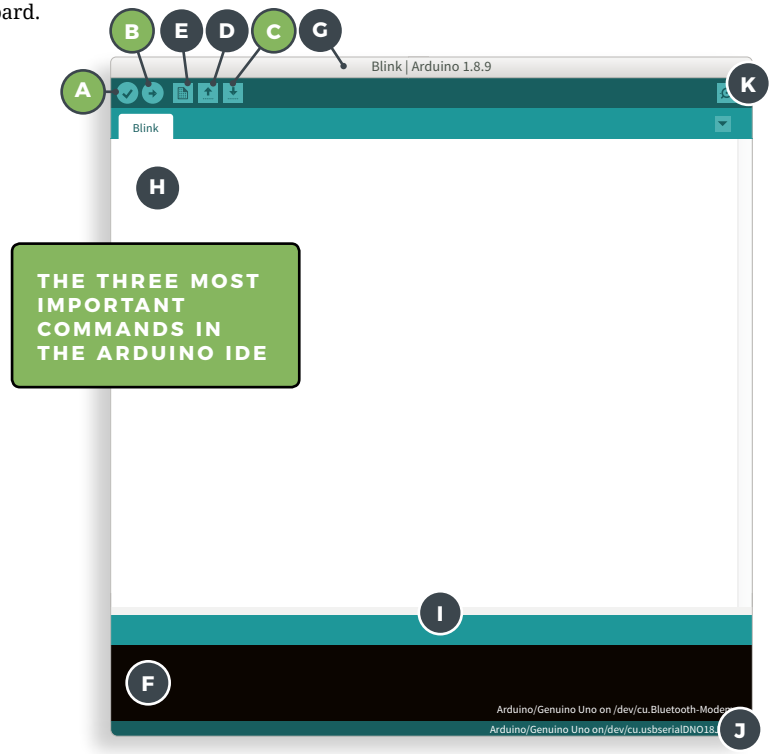


LINUX: Distribution-specific setup instructions for Linux can be found at:

www.sparkfun.com/ch340

5. OPEN THE ARDUINO IDE:

Open the Arduino IDE software on your computer. Poke around and get to know the interface. We aren't going to code right away; this step is to set your IDE to identify your RedBoard.

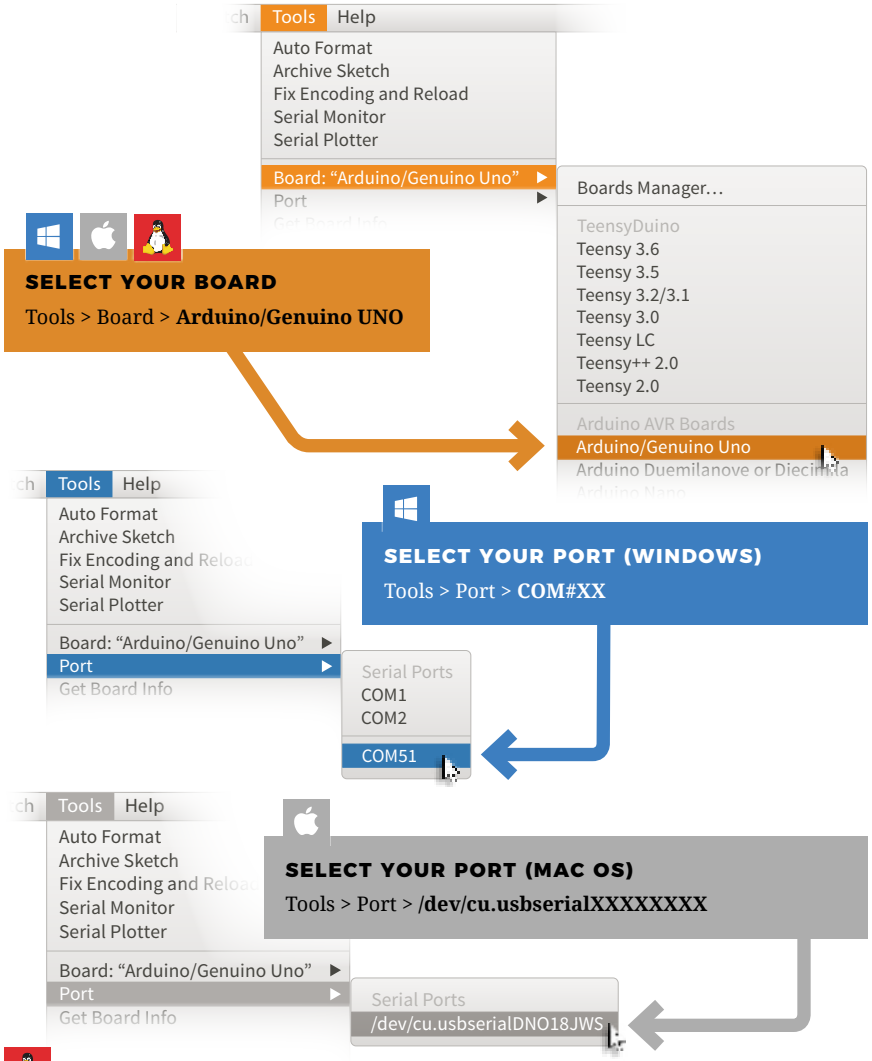


GRAPHICAL USER INTERFACE (GUI)

A	VERIFY	Compiles and approves your code. It will catch errors in syntax (like missing semicolons or parentheses).
B	UPLOAD	Sends your code to the RedBoard. When you click it, you should see the lights on your board blink rapidly.
C	SAVE	Saves the currently active sketch.
D	OPEN	Opens an existing sketch.
E	NEW	Opens up a new code window tab.
F	DEBUG WINDOW	Displays any errors generated by your sketch.
G	SKETCH NAME	Displays the name of the sketch you are currently working on.
H	CODE AREA	Where you compose or edit the code for your sketch.
I	MESSAGE AREA	Indicates if the code is compiling, uploading or has errors.
J	CONNECTION AREA	Displays the board and serial port currently selected.
K	SERIAL MONITOR	Opens a window that displays any serial information your RedBoard is transmitting (useful for debugging).

6. SELECT YOUR BOARD AND SERIAL DEVICE

NOTE: Your SparkFun RedBoard and the Arduino/Genuino UNO are interchangeable, but you won't find the RedBoard listed in the Arduino software. Select **"ARDUINO/GENUINO UNO"** instead.



SELECT YOUR PORT (LINUX)

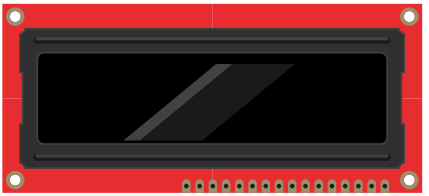
Distribution-specific serial device setup instructions can be found **HERE:**

http://arduino.cc/playground/Learning/Linux CASE SENSITIVE

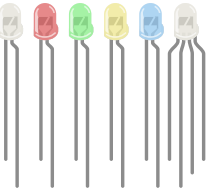
Inventory of Parts

The SparkFun Inventor's Kit contains an extensive array of electronic components. As you work your way through each circuit, you will learn to use new and more complicated parts to accomplish increasingly complex tasks.

LCD DISPLAY



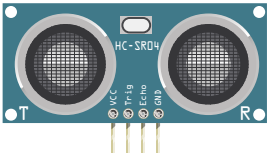
LEDS



POTENTIOMETER

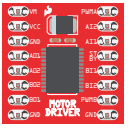
SWITCH

ULTRASONIC DISTANCE SENSOR

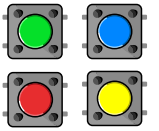


PIEZO BUZZER

MOTOR DRIVER



PUSH BUTTONS



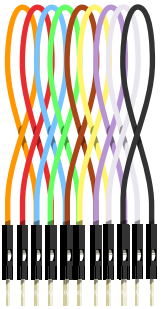
10KΩ RESISTORS

330Ω RESISTORS

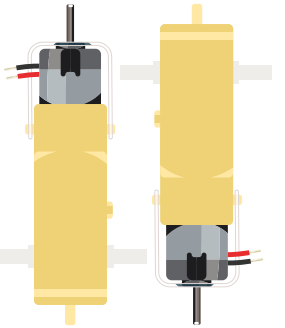
PHOTORESISTOR

TEMPERATURE SENSOR

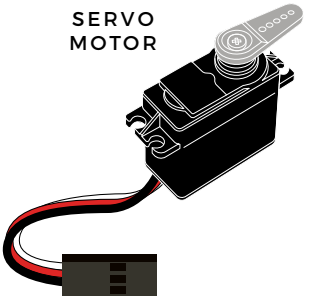
JUMPER WIRES



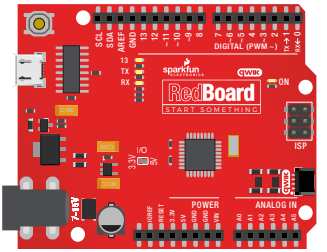
GEARMOTORS



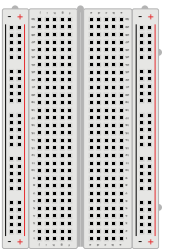
SERVO MOTOR

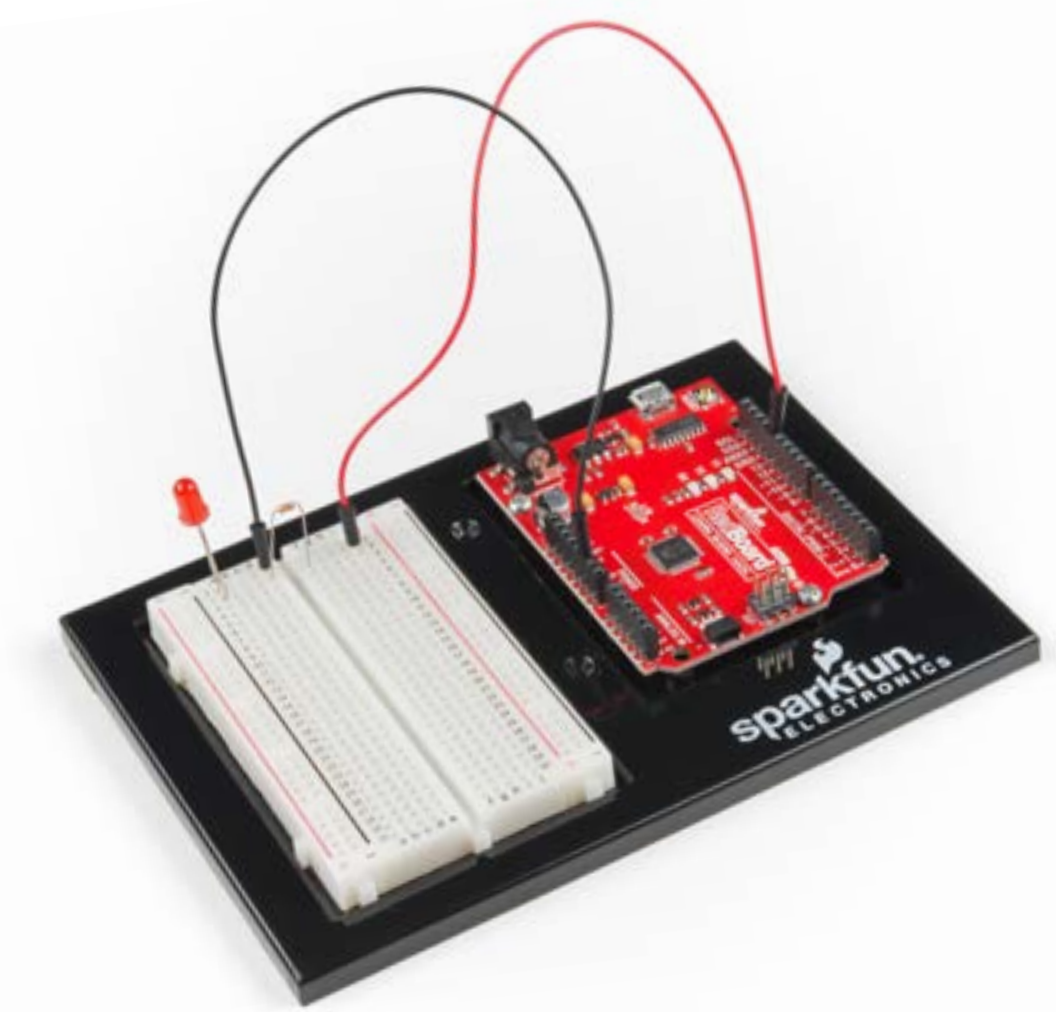


SPARKFUN REDBOARD

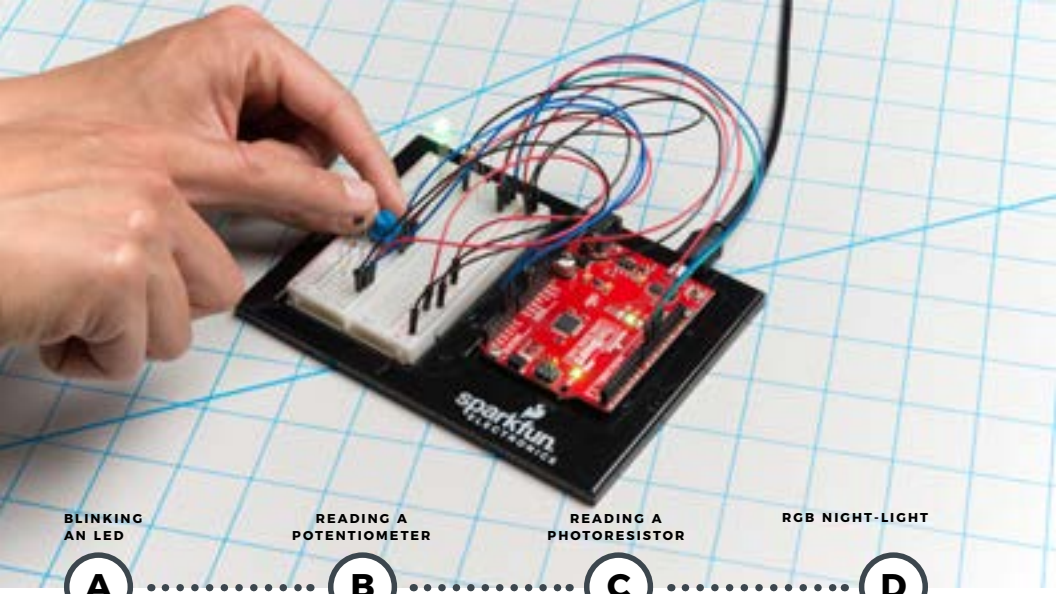


BREADBOARD





Let's Get Started With Your First Circuit!



BLINKING AN LED — A

READING A POTENTIOMETER — B

READING A PHOTORESISTOR — C

RGB NIGHT-LIGHT — D

PROJECT 1

Welcome to your first SparkFun Inventor's Kit project. Each **project** is broken up into several **circuits,** the last circuit being a culmination of the technologies that came before. There are five projects total, each designed to help you learn about new technologies and concepts. This first project will set the foundation for the rest and will aid in helping you understand the fundamentals of circuit building and electricity!

In Project 1, you will learn about **Light-Emitting Diodes (LEDs)**, **resistors**, **inputs**, **outputs** and **sensors**. The first project will be to build and program your own multicolored night-light! The night-light uses a sensor to turn on an RGB (Red, Green, Blue) LED when it gets dark, and you will be able to change the color using an input knob.

NEW IDEAS

Each project will introduce new concepts and components, which will be described in more detail as you progress through the circuits.

NEW COMPONENTS INTRODUCED IN THIS PROJECT

- LEDS
- RESISTORS
- POTENTIOMETERS
- PHOTORESISTORS

NEW CONCEPTS INTRODUCED IN THIS PROJECT

- POLARITY
- OHM'S LAW
- DIGITAL OUTPUT
- ANALOG VS. DIGITAL
- ANALOG INPUT
- ANALOG TO DIGITAL CONVERSION
- VOLTAGE DIVIDER
- PULSE-WIDTH MODULATION
- FUNCTIONS

YOU WILL LEARN

- HOW TO UPLOAD A PROGRAM TO YOUR REDBOARD
- CIRCUIT-BUILDING BASICS
- HOW TO CONTROL LEDS WITH DIGITAL OUTPUTS
- HOW TO READ SENSORS WITH ANALOG INPUTS

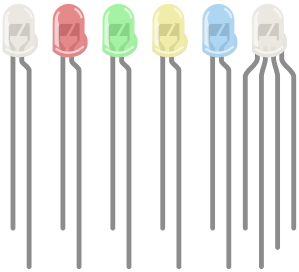
Circuit 1A: Blinking an LED

You can find LEDs in just about any source of light, from the bulbs lighting your home to the tiny status lights flashing on your home electronics. Blinking an LED is the classic starting point for learning how to program embedded electronics. It's the "Hello, World!" of microcontrollers. In this circuit, you'll write code that makes an LED blink on and off.

YOU NEED

NEW COMPONENTS

LIGHT-EMITTING DIODES (LEDS) are small lights made from a silicon diode. They come in different colors, brightnesses and sizes. LEDs (pronounced el-ee-dees) have a positive (+) leg and a negative (-) leg, and they will only let electricity flow through them in one direction. LEDs can also burn out if too much electricity flows through them, so you should always use a resistor to limit the current when you wire an LED into a circuit.



RESISTORS resist the flow of electricity. You can use them to protect sensitive components like LEDs. The strength of a resistor (measured in ohms) is marked on the body of the resistor using small colored bands. Each color stands for a number, which you can look up using a resistor chart. One can be found at the back of this book.

NEW CONCEPTS

POLARITY: Many electronics components have polarity, meaning electricity can (and should) flow through them in only one direction. Polarized components, like an LED, have a positive and a negative leg and only work when electricity flows through them in one direction. Some components, like resistors, do not have polarity; electricity can flow through them in either direction.

OHM'S LAW describes the relationship between the three fundamental elements of electricity: **voltage**, **resistance** and **current**. This relationship can be represented by this equation:

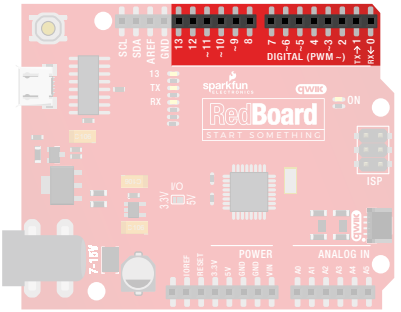
$$V=I \bullet R$$

V = Voltage in volts

I = Current in amps

R = Resistance in ohms (Ω)

This equation is used to calculate what resistor values are suitable to sufficiently limit the current flowing to the LED so that it does not get too hot and burn out.

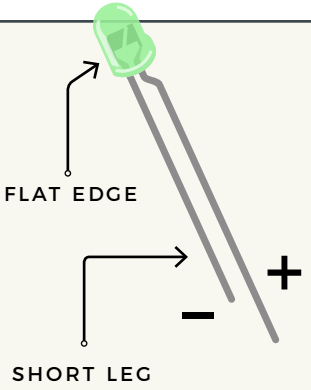


DIGITAL OUTPUT: When working with microcontrollers such as the RedBoard, there are a variety of pins to which you can connect electronic components. Knowing which pins perform which functions is important when building your circuit. In this circuit, we will be using what is known as a digital output. There are 14 of these pins found on the RedBoard. A digital output only has **two states: ON or OFF**. These two states can also be thought of as **HIGH or LOW, TRUE or FALSE**. When an LED is connected to one of these pins, the pin can only perform two jobs: turning on the LED and turning off the LED. We'll explore the other pins and their functions in later circuits.

NEW IDEAS

ELECTRICAL SAFETY: Never work on your circuits while the board is connected to a power source. The SparkFun RedBoard operates at 5 volts, which, while not enough to injure you, is enough to damage the components in your circuit.

COMPONENT ORIENTATION & POLARITY: Instructions on how to orient each of the new components will be given before each circuit diagram. Many components have polarity and have only one correct orientation, while others are nonpolarized.

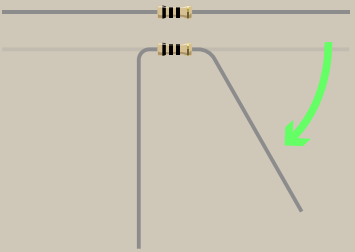


POLARIZED COMPONENTS

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

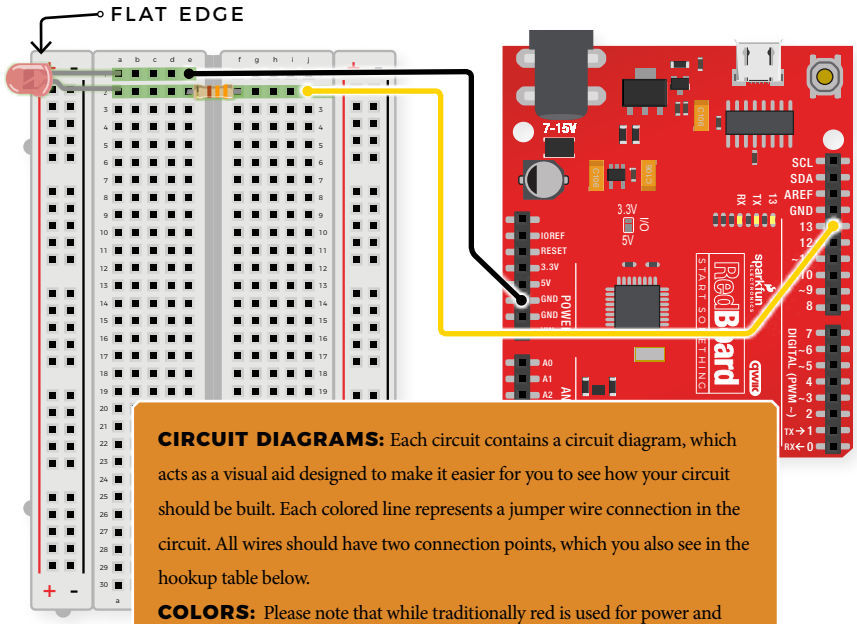
RESISTOR LEADS

Components like resistors need to have their legs bent into 90° angles in order to correctly fit in the breadboard sockets.



HOOKUP GUIDE

READY TO START HOOKING EVERYTHING UP? Check out the circuit diagram and hookup table below to see how everything is connected.



CIRCUIT DIAGRAMS: Each circuit contains a circuit diagram, which acts as a visual aid designed to make it easier for you to see how your circuit should be built. Each colored line represents a jumper wire connection in the circuit. All wires should have two connection points, which you also see in the hookup table below.

COLORS: Please note that while traditionally red is used for power and black is used for ground, all wires, no matter their color, function the same.

HOOKUP TABLES: Many electronics beginners find it helpful to have a coordinate system when building their circuits. For each circuit, you'll find a hookup table that lists the coordinates of each component or wire and where it connects to the RedBoard, the breadboard, or both. The breadboard has a letter/number coordinate system, just like the game Battleship.

◆ **D13** *to* ■ **J2**

...means one end of a component connects to digital pin 13 on your RedBoard and the other connects to J2 on the breadboard

CONNECTION TYPES	◆ REDBOARD CONNECTION ■ BREADBOARD CONNECTION
JUMPER WIRES	◆ **D13** *to* ■ **J2** ; ◆ **GND** *to* ■ **E1**
LED	■ **A1(-)** *to* ■ **A2(+)**
330Ω RESISTOR (ORANGE, ORANGE, BROWN)	■ **E2** *to* ■ **F2**

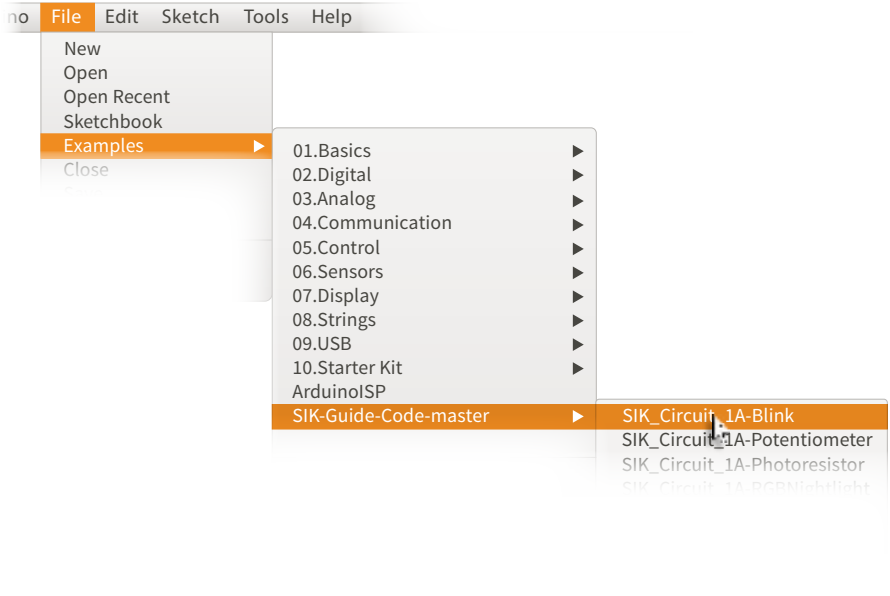
In this table, a yellow highlight indicates that a component has polarity and will only function if properly oriented.

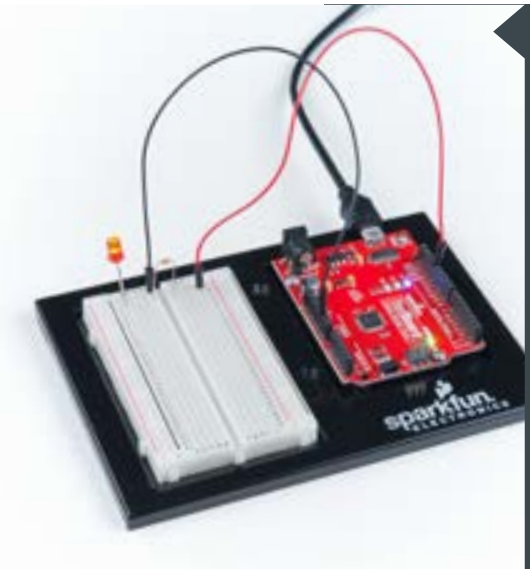
Open the Arduino IDE

Connect the RedBoard to a USB port on your computer.

Open the Sketch: File > Examples > SIK-Guide-Code-master > **CIRCUIT_1A-BLINK**

Select **UPLOAD** to program the sketch on the RedBoard.



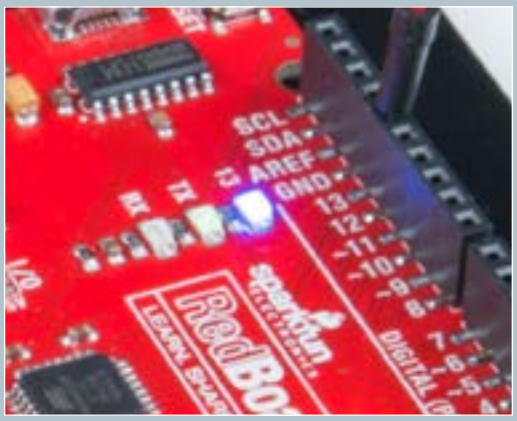


WHAT YOU SHOULD SEE

The LED will flash on for two seconds, then off for two seconds. If it doesn't, make sure you have assembled the circuit correctly and verified and uploaded the code to your board. See the Troubleshooting section at the end of this circuit if that doesn't work. One of the best ways to understand the code you uploaded is to change something and see how it affects the behavior of your circuit. What happens when you change the number in one or both of the `delay(2000);` lines of code (try 100 or 5000)?

PROGRAM OVERVIEW

1. Turn the LED on by sending power (5V) to digital pin 13.
2. Wait 2 seconds (2000 milliseconds).
3. Turn the LED off by cutting power (0V) to digital pin 13.
4. Wait 2 seconds (2000 milliseconds).
5. Repeat.



ONBOARD LED PIN 13: You may have noticed a second, smaller LED blinking in unison with the LED in your breadboard circuit. This is known as the onboard LED, and you can find one on almost any Arduino or Arduino-compatible board. In most cases, this LED is connected to **digital pin 13 (D13)**, the same pin used in this circuit.

NEW IDEAS

CODE TO NOTE: The sketches that accompany each circuit introduce new programming techniques and concepts as you progress through the guide. The Code to Note section highlights specific lines of code from the sketch and explains them in greater detail.

CODE TO NOTE

SETUP AND LOOP:

```
void setup(){} &
void loop(){}
```

Every Arduino program needs these two functions. Code that goes in between the curly brackets `{}` of `setup()` runs once. The code in between the `loop()` curly brackets `{}` runs over and over until the RedBoard is reset or powered off.

INPUT OR OUTPUT?:

```
pinMode(13, OUTPUT);
```

Before you can use one of the digital pins, you need to tell the RedBoard whether it is an `INPUT` or `OUTPUT`. We use a built-in "function" called `pinMode()` to make pin 13 a digital output. You'll learn more about digital inputs in Project 2.

CODE TO NOTE

DIGITAL OUTPUT:

```
digitalWrite(D13, 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).

DELAY:

```
delay(2000);
```

Causes the program to wait on this line of code for the amount of time in between the brackets, represented in milliseconds (2000ms = 2s). After the time has passed, the program will continue to the next line of code.

COMMENTS:

```
//This is a comment

/* So is this */
```

Comments are a great way to leave notes in your code explaining why you wrote it the way you did. Single line comments use two forward slashes `//`, while multi-line comments start with a `/*` and end with a `*/`.

NEW IDEAS

CODING CHALLENGES: The Coding Challenges section is where you will find suggestions for changes to the circuit or code that will make the circuit more challenging. If you feel underwhelmed by the tasks in each circuit, visit the Coding Challenges section to push yourself to the next level.

CODING CHALLENGES

PERSISTENCE OF VISION: Computer screens, movies and the lights in your house all flicker so quickly that they appear to be on all of the time but are actually blinking faster than the human eye can detect. See how much you can decrease the delay time in your program before the light appears to be on all the time but is still blinking.

MORSE CODE: Try adding and changing the `delay()` values and adding more `digitalWrite()` commands to make your program blink a message in Morse code.

TROUBLESHOOTING

I get an error when uploading my code

The most likely cause is that you have the wrong board selected in the Arduino IDE. Make sure you have selected **Tools > Board > Arduino/Genuino Uno.**

TROUBLESHOOTING

I still get an error when uploading my code	If you're sure you have the correct Board selected but you still can't upload, check that you have selected the correct serial port. You can change this in **Tools > Serial Port > your_serial_port**.
Which serial port is the right one?	Depending on how many devices you have plugged into your computer, you may have several active serial ports. Make sure you are selecting the correct one. A simple way to determine this is to look at your list of serial ports. Unplug your RedBoard from your computer. Look at the list again. Whichever serial port has disappeared from the list is the one you want to select once you plug your board back into your computer.
My code uploads, but my LED won't turn on	LEDs will only work in one direction. Try taking it out of your breadboard, turning it 180 degrees and reinserting it.
Still not working?	Jumper wires unfortunately can go "bad" from getting bent too much. The copper wire inside can break, leaving an open connection in your circuit. If you are certain that your circuit is wired correctly and that your code is error-free and uploaded, but you are still encountering issues, try replacing one or more of the jumper wires for the component that is not working.

You've completed Circuit 1A!

Continue to circuit 1B to learn about analog signals and potentiometers.

Circuit 1B: Potentiometer

Potentiometers (also known as “trimpots” or “knobs”) are one of the basic inputs for electronic devices. By tracking the position of the knob with your RedBoard, you can make volume controls, speed controls, angle sensors and a ton of other useful inputs for your projects. In this circuit, you’ll use a potentiometer as an input device to control the speed at which your LED blinks.

YOU NEED


LED


POTENTIOMETER


330Ω RESISTOR


7 JUMPER WIRES

NEW COMPONENTS

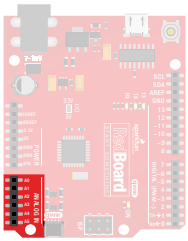
POTENTIOMETER: A potentiometer is a 3-pin variable resistor. When powered with 5V, the middle pin outputs a voltage between 0V and 5V, depending on the position of the knob on the potentiometer. Internal to the trimpot is a single resistor and a wiper, which cuts the resistor in two and moves to adjust the ratio between both halves.

NEW CONCEPTS

ANALOG VS. DIGITAL: We live in an analog world. There are an infinite number of colors to paint an object, an infinite number of tones we can hear, and an infinite number of smells we can smell. The common theme among these analog signals is their infinite possibilities.

Digital signals deal in the realm of the discrete or finite, meaning there is a limited set of values they can be. The LED from the previous circuit had only two states it could exist in, ON or OFF, when connected to a digital output.

ANALOG INPUTS: So far, we’ve only dealt with outputs. The RedBoard also has inputs. Both inputs and outputs can be analog or digital. Based on our previous definition of analog and digital, that means an analog input can sense a wide range of values versus a digital input, which can only sense two values, or states.



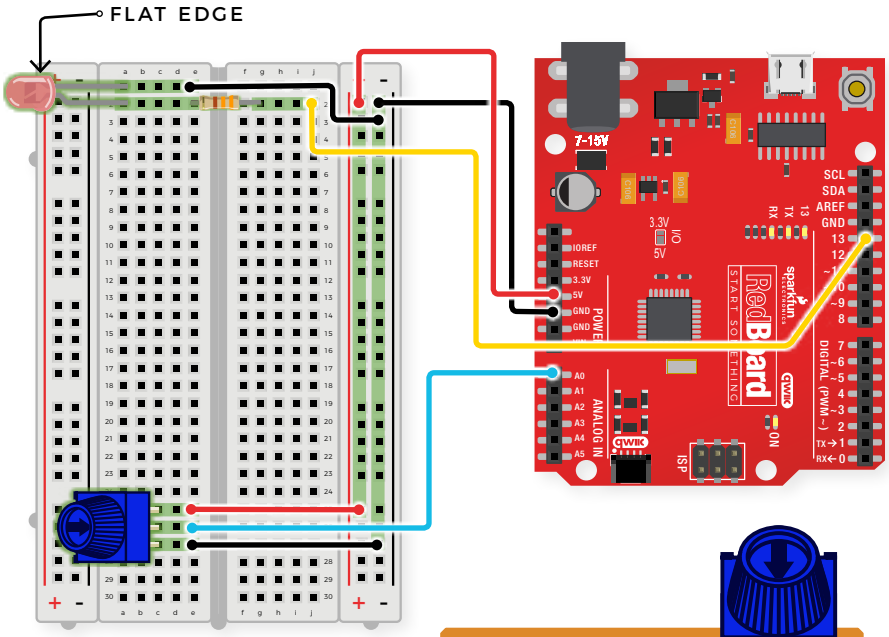
You may have noticed some pins labeled **Digital** and some labeled **Analog In** on your RedBoard. There are only six pins that function as analog inputs; they are labeled A0–A5.

VOLTAGE DIVIDER

VOLTAGE DIVIDERS are simple circuits that turn some voltage into a smaller voltage using two resistors. A potentiometer is a variable resistor that can be used to create an adjustable voltage divider. A wiper in the middle position means the output voltage will be half of the input. Voltage dividers will be covered in more detail in the next circuit.

HOOKUP GUIDE

READY TO START HOOKING EVERYTHING UP? Check out the circuit diagram and hookup table below to see how everything is connected.



NEW IDEAS

POTENTIOMETERS are not polarized and can be installed in either direction. Note that swapping the 5V and GND pins will reverse its behavior.

CONNECTION TYPES	◆ REDBOARD CONNECTION ■ BREADBOARD CONNECTION
JUMPER WIRES	◆ 5V to ■ 5V ◆ GND to ■ GND (-) ◆ A0 to ■ E26 ■ E25 to ■ 5V (+) ■ E27 to ■ GND (-) ■ E1 to ■ GND (-) ◆ D13 to ■ J2
LED	■ A1(-) to ■ A2(+)
330Ω RESISTOR (ORANGE, ORANGE, BROWN)	■ E2 to ■ G2
POTENTIOMETER	■ C25 + ■ C26 + ■ C27

Open the Arduino IDE

Connect the RedBoard to a USB port on your computer.

Open the Sketch:
File > Examples >SIK-Guide-Code-master > **CIRCUIT_1B-POTENTIOMETER**

Select **UPLOAD** to program the sketch on the RedBoard.

WHAT YOU SHOULD SEE

You should see the LED blink faster or slower in accordance with your potentiometer. The delay between each flash will change based on the position of the knob. If it isn't working, make sure you have assembled the circuit correctly and verified and uploaded the code to your board. If that doesn't work, see the Troubleshooting section.

PROGRAM OVERVIEW

1	Read the position of the potentiometer (from 0 to 1023) and store it in the variable **potPosition**.
2	Turn the LED on.
3	Wait from 0 to 1023 milliseconds, based on the position of the knob and the value of **potPosition**.
4	Turn the LED off.
5	Wait from 0 to 1023 milliseconds, based on the position of the knob and the value of **potPosition**.
6	Repeat.

ARDUINO PRO TIP

ARDUINO SERIAL MONITOR: The Serial Monitor is one of the Arduino IDE's many great included features. When working with embedded systems, it helps to see and understand the values that your program is trying to work with, and it can be a powerful debugging tool when you run into issues where your code is not behaving the way you expected it to. This circuit introduces you to the Serial Monitor by showing you how to print the values from your potentiometer to it. To see these values, click the Serial Monitor button, found in the upper-right corner of the IDE in most recent versions. You can also select **Tools** > **Serial Monitor** from the menu.

Serial Monitor button in the upper-right of the Arduino IDE.

Serial Monitor printout and baud-rate menu.

You should see numeric values print out in the monitor. Turning the potentiometer changes the value as well as the delay between each print.

If you are having trouble seeing the values, ensure that you have selected 9600 baud and have auto scroll checked.

CODE TO NOTE

INTEGER VARIABLES:

```
int potPosition;
```

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 **potPosition** of type **int** (integer). We will cover more types of variables in later circuits. Don't forget that variable names are case-sensitive!

CODE TO NOTE

SERIAL BEGIN:

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

Serial commands can be used to send and receive data from your computer. This line of code tells the RedBoard that we want to "begin" that communication with the computer, the same way we would say "Hi" to initiate a conversation. Notice that the baud rate, 9600, is the same as the one we selected in the monitor. This is the speed at which the two devices communicate, and it must match on both sides.

ANALOG INPUT:

```
potPosition =
analogRead(A0);
```

We use the `analogRead()` function to read the value on an analog pin. `analogRead()` takes one parameter, the analog pin you want to use, `A0` in this case, and returns a number between 0 (0 volts) and 1023 (5 volts), which is then assigned to the variable `potPosition`.

SERIAL PRINT:

```
Serial.
println(potPosition);
```

This is the line that actually prints the trimpot value to the monitor. It takes the variable `potPosition` and prints whatever value it equals at that moment in the `loop()`. The `ln` at the end of `println` tells the monitor to print a new line at the end of each value; otherwise the values would all run together on one line. Try removing the `ln` to see what happens.

CODING CHALLENGES

CHANGING THE RANGE: Try multiplying, dividing or adding to your sensor reading so that you can change the range of the delay in your code. For example, can you multiply the sensor reading so that the delay goes from 0–2046 instead of 0–1023?

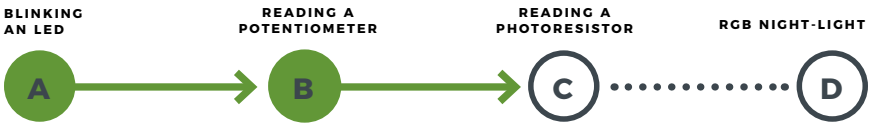
ADD MORE LEDS: Add more LEDs to your circuit. Don't forget the current-limiting resistors. You will need to declare the new pins in your code and set them all to `OUTPUT`. Try making individual LEDs blink at different rates by changing the range of each using multipilcation or division.

TROUBLESHOOTING

The potentiometer always reads as 0 or 1023	Make sure that your 5V, A0 and GND pins are properly connected to the three pins on your potentiometer. It is easy to misalign a wire with the actual pot pin.
No values or random characters in Serial Monitor	Make sure that you have selected the correct baud rate, **9600**. Also ensure that you are on the correct serial port. The same serial port you use when uploading code to your board is the same serial port you use to print values to the Serial Monitor.

You've completed Circuit 1B!

Continue to circuit 1C to learn about photoresistors and analog to digital conversion.



Circuit 1C: Photoresistor

In circuit 1B, 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. Using this sensor you can make a simple night-light that turns on when the room gets dark and turns off when it is bright.

YOU NEED

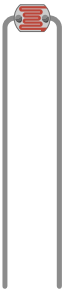

LED


PHOTORESISTOR


330Ω RESISTOR


10KΩ RESISTOR


7 JUMPER WIRES



NEW COMPONENTS

PHOTORESISTORS are light-sensitive, variable resistors. As more light shines on the sensor's head, the resistance between its two terminals decreases. They're an easy-to-use component in projects that require ambient-light sensing.

NEW CONCEPTS

ANALOG TO DIGITAL CONVERSION:
In order to have the RedBoard sense analog signals, we must first pass them through an Analog to Digital Converter (or ADC). The six analog inputs (A0–A5) covered in the last circuit all use an ADC. These pins sample the analog signal and create a digital signal for the microcontroller to interpret. The **resolution** of this signal is based on the resolution of the ADC. In the case of the RedBoard, that resolution is 10-bit. With a 10-bit ADC, we get 2 ^ 10 = 1024 possible values, which is why the analog signal can vary between 0 and 1023.

VOLTAGE DIVIDERS CONTINUED:
Since the RedBoard can't directly interpret resistance (rather, it reads voltage), we need to use a voltage divider to use our photoresistor, a part that doesn't output voltage. The resistance of the photoresistor changes as it gets darker or lighter. That changes or "divides" the voltage going through the divider circuit. That divided voltage is then read in on the analog to digital converter of the analog input.

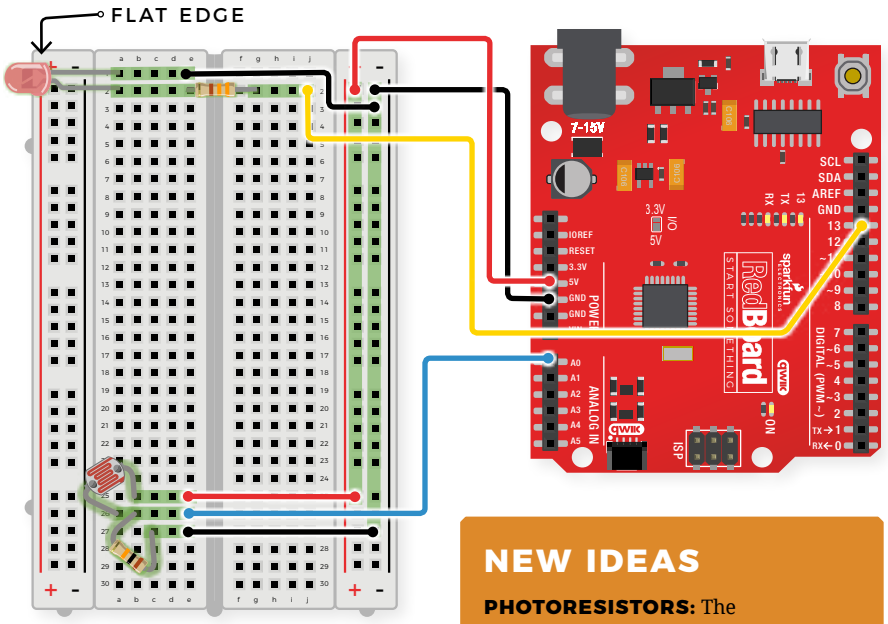
The voltage divider equation:

$$V_{out} = V_{in} \bullet \frac{R_2}{R_1 + R_2}$$

assumes that you know three values of the above circuit: the input voltage (V_{in}), and both resistor values (R_1 and R_2). If R1 is a constant value (the resistor) and R2 fluctuates (the photoresistor), the amount of voltage measured on the Vout pin will also fluctuate.

HOOKUP GUIDE

READY TO START HOOKING EVERYTHING UP? Check out the circuit diagram and hookup table below to see how everything is connected.



NEW IDEAS

PHOTORESISTORS: The photoresistors, like regular resistors, are not polarized and can be installed in either direction.

CONNECTION TYPES ◆ REDBOARD CONNECTION ■ BREADBOARD CONNECTION

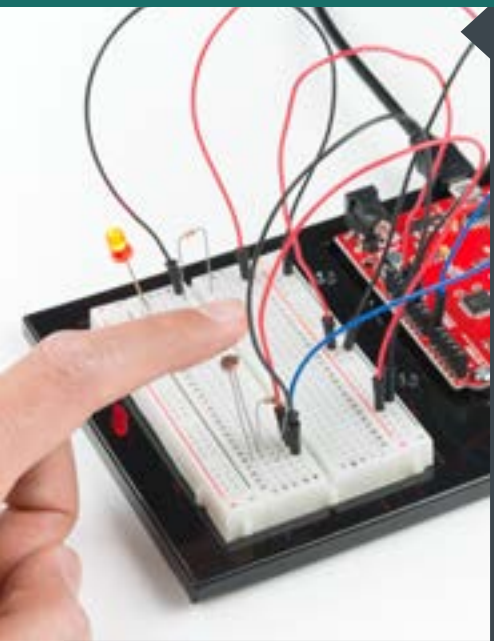
JUMPER WIRES	◆ 5V to ■ 5V(+)
	◆ GND to ■ GND (-)
	◆ D13 to ■ J2
	◆ A0 to ■ E26
	■ E1 to ■ GND(-)
	■ E25 to ■ 5V(+)
	■ E27 to ■ GND(-)
LED	■ A1(-) to ■ A2(+)
330Ω RESISTOR (ORANGE, ORANGE, BROWN)	■ E2 to ■ G2
10KΩ RESISTOR (BROWN, BLACK, ORANGE)	■ B26 to ■ C27
PHOTORESISTOR	■ A26 to ■ B25

Open the Arduino IDE

Connect the RedBoard to a USB port on your computer.

Open the Sketch:
File > Examples > SIK-Guide-Code-master > **CIRCUIT_1C-PHOTORESISTOR**

Select **UPLOAD** to program the sketch on the RedBoard.



WHAT YOU SHOULD SEE

The program stores the light level in a variable. Using an **if**/**else** statement, the variable value is compared to the threshold. If the variable is above the threshold (it's bright), turn the LED off. If the variable is below the threshold (it's dark), turn the LED on. Open the Serial Monitor in Arduino. The value of the photoresistor should be printed every so often. When the photoresistor value drops below the threshold, the LED should turn on (you can cover the photoresistor with your finger for testing).

NEW IDEAS

LIGHT LEVELS: If the room you are in is very bright or dark, you may have to change the value of the `threshold` variable in the code to make your night-light turn on and off. See the Troubleshooting section for instructions.

PROGRAM OVERVIEW

1	Store the light level in the variable **`photoresistor`**.
2	If the value of the **`photoresistor`** is above the **`threshold`** (it's bright), turn the LED off.
3	Otherwise, the value of the **`photoresistor`** is below the **`threshold`** (it's dark), turn the LED on.

CODE TO NOTE

IF ELSE STATEMENTS:

```
if(logic statement){
//run if true
}
else{
//run if false
}
```

The **if else** statement lets your code react to the world by running one set of code when the logic statement in the round brackets is true and another set of code when the logic statement is false. For example, this sketch uses an **if** statement to turn the LED on when it is dark, and **else** statement to turn the LED off when it is light.

LOGICAL OPERATORS:

```
(photoResistor <
threshold)
```

Programmers use logic statements to translate things that happen in the real world into code. Logic statements use logical operators like 'equal to' ==, 'greater than' > and 'less than' <, to make comparisons. When the comparison is true (e.g., 4 < 5), then the logic statement is true. When the comparison is false (e.g., 5 < 4) then the logic statement is false. This example is asking whether the variable `photoresistor` is less than the variable `threshold`.

CODING CHALLENGE

RESPONSE PATTERN: Right now your **if** statement turns the LED on when it gets dark, but you can also use the light sensor like a no-touch button. Try using `digitalWrite()` and `delay()` to make the LED blink a pattern when the light level drops, then calibrate the threshold variable in the code so that the blink pattern triggers when you wave your hand over the sensor.

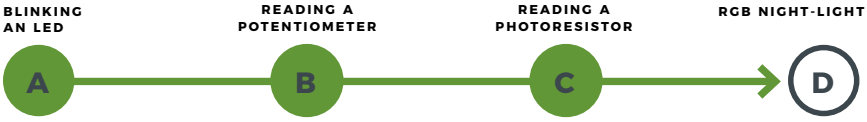
REPLACE 10KΩ RESISTOR WITH AN LED: Alter the circuit by replacing the 10KΩ resistor with an LED (the negative leg should connect to GND). Now what happens when you place your finger over the photoresistor? This is a great way to see Ohm's law in action by visualizing the effect of the change in resistance on the current flowing through the LED.

TROUBLESHOOTING

Nothing is printing in the Serial Monitor	Try unplugging your USB cable and plugging it back in. In the Arduino IDE, go to Tools > Port, and make sure that you select the right port.
The light never turns on or always stays on	Start the Serial Monitor in Arduino. Look at the value that the photoresistor is reading in a bright room (e.g., 915). Cover the photoresistor, or turn the lights off. Then look at the new value that the photoresistor is reading (e.g., 550). Set the threshold in between these two numbers (e.g., 700) so that the reading is above the threshold when the lights are on and below the threshold when the lights are off.

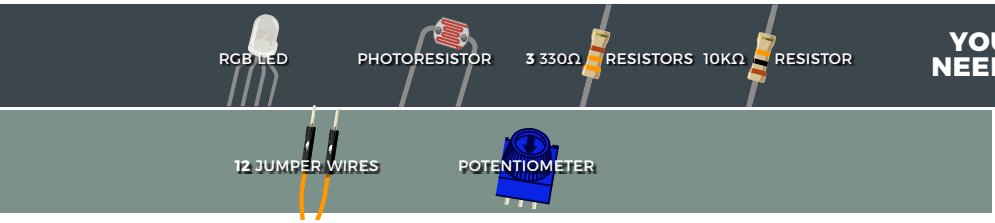
You've completed Circuit 1C!

Continue to circuit 1D to learn about RGB LEDs, functions and pulse-width modulation.



Circuit 1D: RGB Night-Light

In this circuit, you'll take the night-light concept to the next level by adding an RGB LED, which is three differently colored Light-Emitting Diodes (LEDs) built into one component. RGB stands for Red, Green and Blue, and these three colors can be combined to create any color of the rainbow!



NEW COMPONENTS

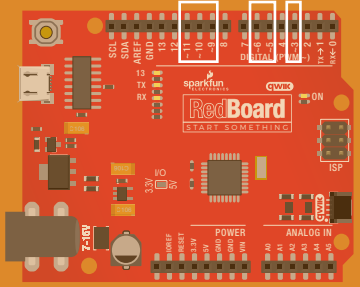
RGB LED: An RGB LED is actually three small LEDs — one red, one green and one blue — inside a normal LED housing. This RGB LED has all the internal LEDs share the same ground wire, so there are four legs in total. To turn on one color, ensure ground is connected, then power one of the legs just as you would a regular LED. Don't forget the current-limiting resistors. If you turn on more than one color at a time, you will see the colors start to blend together to form a new color.

NEW CONCEPTS

ANALOG OUTPUT (PULSE-WIDTH MODULATION): The `digitalWrite()` command can turn pins on (5V) or off (0V), but what if you want to output 2.5V? The `analogWrite()` command can output 2.5 volts by quickly switching a pin on and off so that it is only on 50 percent of the time (50% of 5V is 2.5V). By doing this, any voltage between 0 and 5V can be produced. This is what is known as Pulse-Width Modulation (PWM). It can create many different colors on the RGB LED.

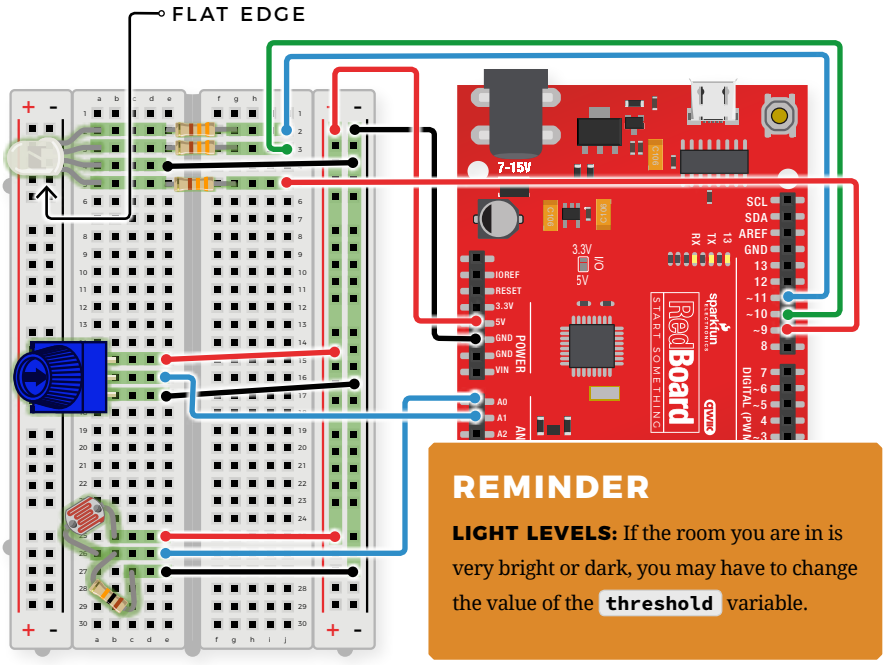
NEW IDEAS

PWM PINS: Only a few of the pins on the RedBoard have the circuitry needed to turn on and off fast enough for PWM. These are pins 3, 5, 6, 9, 10 and 11. Each PWM pin is marked with a ~ on the board. Remember, you can only use `analogWrite()` on these pins.



HOOKUP GUIDE

READY TO START HOOKING EVERYTHING UP? Check out the circuit diagram and hookup table below to see how everything is connected.



REMINDER

LIGHT LEVELS: If the room you are in is very bright or dark, you may have to change the value of the `threshold` variable.

CONNECTION TYPES ◆ REDBOARD CONNECTION ■ BREADBOARD CONNECTION

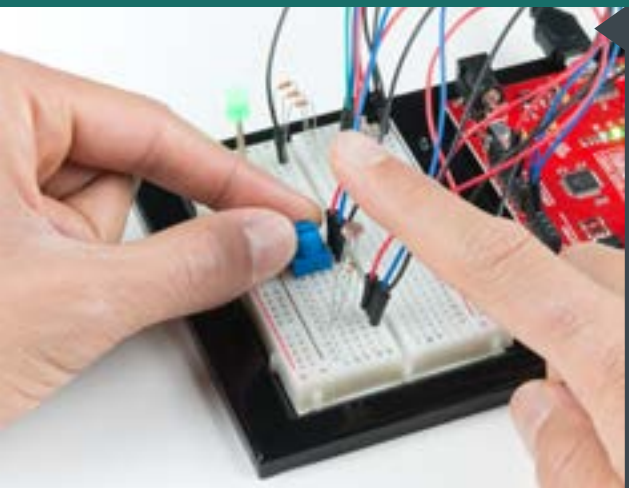
JUMPER WIRES	◆ 5V to ■ 5V(+) · ◆ GND to ■ GND (-) · ◆ D9 to ■ J5 · ◆ D10 to ■ J3 · ◆ D11 to ■ J2 · ◆ A0 to ■ E26 · ◆ A1 to ■ E16 · ■ E15 to ■ 5V(+) · ■ E17 to GND(-) · ■ E4 to ■ GND(-) · ■ E25 to ■ 5V(+) · ■ E27 to ■ GND (-)
RGB LED	■ A5(RED) + ■ A4(GND) + ■ A3(GREEN) + ■ A2(BLUE)
330Ω RESISTORS (ORANGE, ORANGE, BROWN)	■ E2 to ■ G2 · ■ E3 to ■ G3 · ■ E5 to ■ G5
10KΩ RESISTOR (BROWN, BLACK, ORANGE)	■ B26 to ■ C27
PHOTORESISTOR	■ A26 to ■ B25
POTENTIOMETER	■ B15 + ■ B16 + ■ B17

Open the Arduino IDE

Connect the RedBoard to a USB port on your computer.

Open the Sketch:
File > Examples > SIK-Guide-Code-master > **SIK_CIRCUIT_1D-RGB NIGHT LIGHT**

Select **UPLOAD** to program the sketch on the RedBoard.



WHAT YOU SHOULD SEE

This sketch is not dissimilar from the last. It reads the value from the photoresistor, compares it to a threshold value, and turns the RGB LED on or off accordingly. This time, however, we've added a potentiometer back into the circuit. When you twist the trimpot, you should see the color of the RGB LED change based on the trimpot's value.

PROGRAM OVERVIEW

1	Store the light level from pin A0 in the variable `photoresistor`.
2	Store the potentiometer value from pin A1 in the variable `potentiometer`.
3	If the light level variable is above the `threshold`, call the function that turns the RGB LED off.
4	If the light level variable is below the `threshold`, call one of the color functions to turn the RGB LED on.
5	If `potentiometer` is between 0 and 150, turn the RGB LED on red.
6	If `potentiometer` is between 151 and 300, turn the RGB LED on orange.
7	If `potentiometter` is between 301 and 450, turn the RGB LED on yellow.
8	If `potentiometer` is between 451 and 600, turn the RGB LED on green.
9	If `potentiometer` is between 601 and 750, turn the RGB LED on cyan.
10	If `potentiometer` is between 751 and 900, turn the RGB LED on blue.
11	If `potentiometer` is greater than 900, turn the RGB LED on magenta.

CODE TO NOTE

ANALOG OUTPUT (PWM):

```
analogWrite(RedPin, 100);
```

The `analogWrite()` function outputs a voltage between 0 and 5V on a pin. The function breaks the range between 0 and 5V into 255 little steps. Note that we are not turning the LED on to full brightness (255) in this code so that the night-light is not too bright. Feel free to change these values and see what happens.

NESTED IF STATEMENTS:

```
if(logic statement){
  if(logic statement){

  }
}
```

A nested **if** statement is one or more **if** statements "nested" inside of another **if** statement. If the parent **if** statement is true, then the code looks at each of the nested **if** statements and executes any that are true. If the parent **if** statement is false, then none of the nested statements will execute.

MORE LOGICAL OPERATORS:

```
(potentiometer > 0 &&

  potentiometer <= 150)
```

These **if** statements are checking for two conditions by using the AND **&&** operator. In this line, the **if** statement will only be true if the value of the variable `potentiometer` is greater than 0 AND if the value is less than or equal to 150. By using **&&**, the program allows the LED to have many color states.

DEFINING A FUNCTION:

```
void function_name(){
}
```

This is a definition of a simple function. When programmers want to use many lines of code over and over again, they write a function. The code inside the curly brackets "executes" whenever the function is "called" in the main program. Each of the colors for the RGB LED is defined in a function.

CALLING A FUNCTION:

```
function_name();
```

This line "calls" a function that you have created. In a later circuit, you will learn how to make more complicated functions that take data from the main program (these pieces of data are called **parameters**).

CODING CHALLENGES

ADD MORE COLORS: You can create many more colors with the RGB LED. Use the `analogWrite()` function to blend different values of red, green and blue together to make even more colors. You can divide the potentiometer value and make more nested **if** statements so that you can have more colors as you twist the knob.

MULTI-COLOR BLINK: Try using delays and multiple color functions to have your RGB LED change between multiple colors when it is dark.

CHANGE THE THRESHOLD: Try setting your threshold variable by reading the value of a potentiometer. By turning the potentiometer, you can then change the threshold level and adjust your night-light for different rooms.

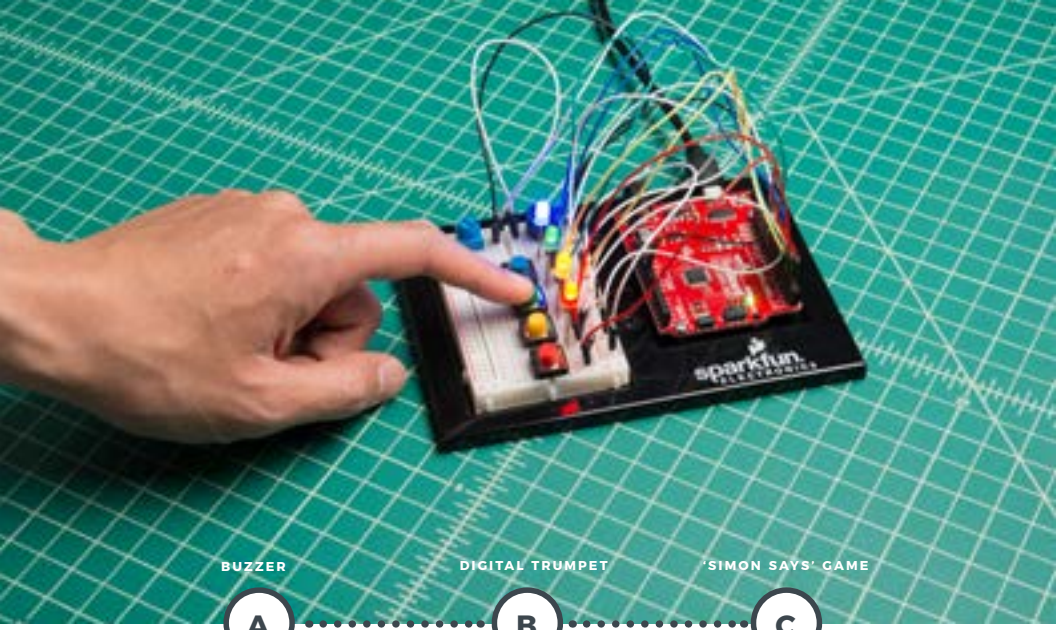
FADING THE LED: Use `analogWrite()` to get your LED to pulse gently or smoothly transition between colors.

TROUBLESHOOTING

The LED never turns on or off	Open the Serial Monitor and make sure that your photoresistor is returning values between 0 and 1023. Cover the photoresistor; the values should change. If they do not change, check your circuit. Make sure that your threshold variable sits in between the value that the photoresistor reads when it is bright and the value when it is dark (e.g., bright = 850, dark = 600, threshold = 700).
My LED doesn't show the colors that I expect	Make sure that all three of the pins driving your RGB LED are set to `OUTPUT`, using the `pinMode()` command in the setup section of the code. Then make sure that each leg of the LED is wired properly.
Nothing is printing in the Serial Monitor	Try unplugging your USB cable and plugging it back in. In the Arduino IDE, go to **Tools** > **Port**, and select the right port.

You've completed Circuit 1D!

Continue to Project 2 to explore using buzzers to make sound.



PROJECT 2

In Project 2, you will venture into the world of **buttons** and **buzzers** while building your own "Simon Says" game! "Simon Says" is a game in which the LEDs flash a pattern of red, green, yellow and blue blinks, and the user must recreate the pattern using color-coded buttons before the timer runs out.

NEW COMPONENTS INTRODUCED IN THIS PROJECT

- BUZZER
- BUTTONS

NEW CONCEPTS INTRODUCED IN THIS PROJECT

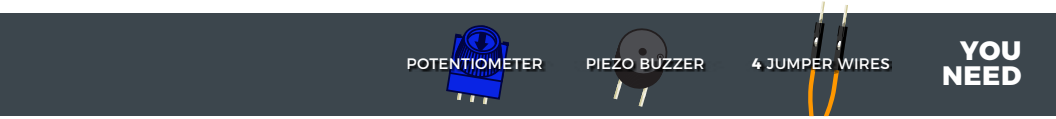
- ARRAYS
- BINARY
- DIGITAL INPUTS
- PULL-UP RESISTORS
- FOR LOOPS
- MEASURING ELAPSED TIME

YOU WILL LEARN

- HOW TO MAKE TONES WITH A BUZZER
- HOW TO READ A BUTTON USING DIGITAL INPUTS
- HOW TO PROGRAM A GAME

Circuit 2A: Buzzer

In this circuit, you'll use the RedBoard and a small buzzer to make music, and you'll learn how to program your own songs using arrays.

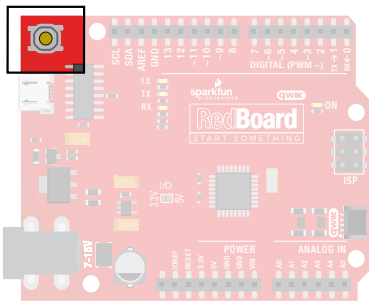


NEW COMPONENTS

BUZZER: The buzzer uses a small magnetic coil to vibrate a metal disc inside a plastic housing. By pulsing electricity through the coil at different rates, different frequencies (pitches) of sound can be produced. Attaching a potentiometer to the output allows you to limit the amount of current moving through the buzzer and lower its volume.

NEW CONCEPTS

RESET BUTTON: The RedBoard has a built-in reset button. This button will reset the board and start the code over from the beginning, running `setup()` then `loop()`.



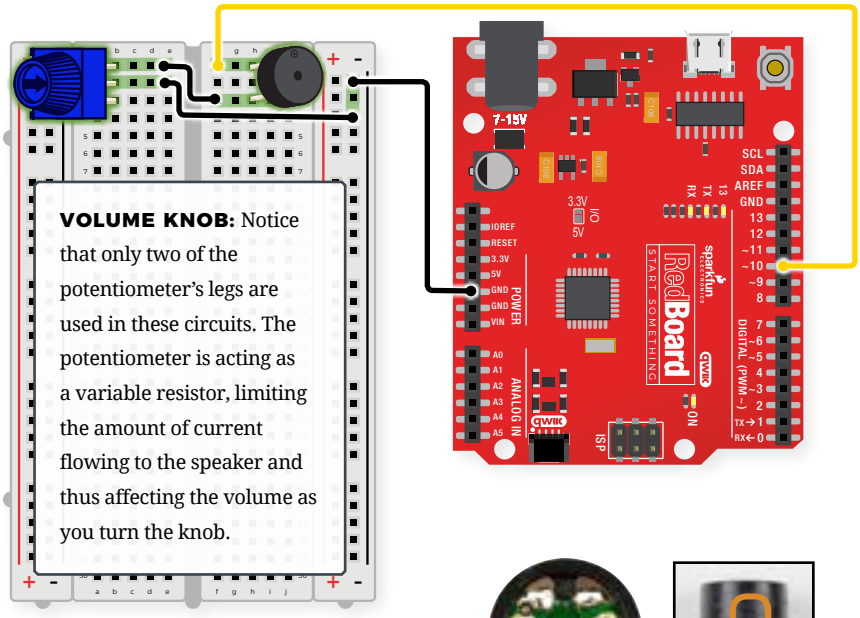
TONE FUNCTION: To control the buzzer, you will use the `tone()` function. This function is similar to PWM in that it generates a wave that is of a certain frequency on the specified pin. The frequency and duration can both be passed to the `tone()` function when calling it. To turn the tone off, you need to call `noTone()` or pass a duration of time for it to play and then stop. Unlike PWM, `tone()` can be used on any digital pin.

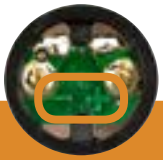
ARRAYS are used like variables, but they can store multiple values. The simplest array is just a list. Imagine that you want to store the frequency for each note of the C major scale. We could make seven variables and assign a frequency to each one, or we could use an array and store all seven in the same list. To refer to a specific value in the array, an index number is used. Arrays are indexed from 0. For example, to call the first element in the array, use `array_name[0];` to call the second element, use `array_name[1];` and so on.

MUSICAL NOTE	FREQUENCY (HZ)	USING VARIABLES	USING AN ARRAY
A	220	A_FREQUENCY	FREQUENCY[0]
B	247	B_FREQUENCY	FREQUENCY[1]
C	261	C_FREQUENCY	FREQUENCY[2]
D	294	D_FREQUENCY	FREQUENCY[3]
E	330	E_FREQUENCY	FREQUENCY[4]
F	349	F_FREQUENCY	FREQUENCY[5]
G	392	G_FREQUENCY	FREQUENCY[6]

HOOKUP GUIDE

READY TO START HOOKING EVERYTHING UP? Check out the circuit diagram and hookup table below to see how everything is connected.





REMEMBER!

POLARITY: The buzzer is polarized. To see which leg is positive and which is negative, flip the buzzer over and look at the markings underneath. Keep track of which pin is where, as they will be hard to see once inserted into the breadboard. There is also text on the positive side of the buzzer, along with a tiny (+) symbol.

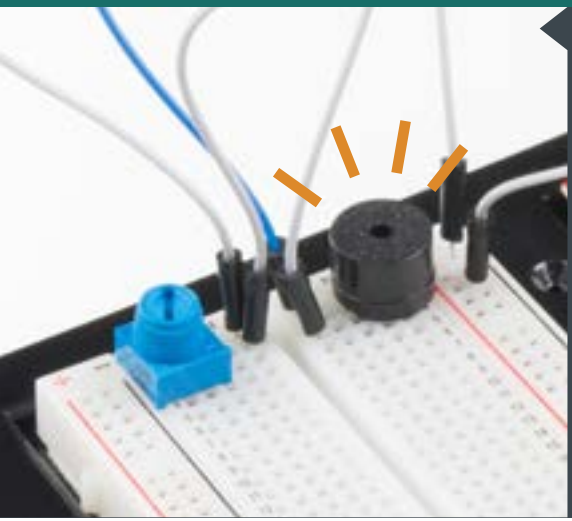
CONNECTION TYPES	◆ REDBOARD CONNECTION ■ BREADBOARD CONNECTION
JUMPER WIRES	◆ GND *to* ■ GND(-) ◆ D10 *to* ■ F1 ■ E2 *to* ■ GND (-) ■ E1 *to* ■ F3
BUZZER	■ H1(+) *to* ■ H3(-)
POTENTIOMETER	■ B1 + ■ B2 + ■ B3

Open the Arduino IDE

Connect the RedBoard to a USB port on your computer.

Open the Sketch:
File > Examples > SIK-Guide-Code-master > **SIK_CIRCUIT_2A-BUZZER**

Select **UPLOAD** to program the sketch on the RedBoard.



WHAT YOU SHOULD SEE

When the program begins, a song will play from the buzzer once. To replay the song, press the reset button on the RedBoard. Use the potentiometer to adjust the volume.

PROGRAM OVERVIEW

1

Play the first note for x number of beats using the **`play()`** function.

A: (Inside the **`play()`** function): Take the note passed to the play function and compare it to each letter in the notes array. When you find a note that matches, remember the index position of that note (e.g., sixth entry in the notes array).

B: Get a frequency from the frequency array that has the same index as the note that matched (e.g., the sixth frequency).

C: Play that frequency for the number of beats passed to the **`play()`** function.

2

Play the second note using the **`play()`** function
...and so on.

CODE TO NOTE

CHARACTER VARIABLES:

```
void play(char note, int beats)
```

The `char`, or **character**, variable stores character values. In this sketch, the **`play()`** function gets passed two variables: a character variable that represents the musical note we want to play and an integer variable that represents how long to play that note. A second array takes the character variable and associates a frequency value to it. This makes programming a song easier as you can just reference the character and not the exact frequency.

TONE FUNCTION:

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

The **`tone()`** function will pulse power to a pin at a specific frequency. The duration controls how long the sound will play. **`tone()`** can be used on any digital pin.

DECLARING AN ARRAY:

```
array_name[array_size];
```

To declare an **array**, you must give it a name, then either tell Arduino how many positions the array will have or assign a list of values to the array. An array must contain all the same type of variables and be declared as such.

CALLING AN ARRAY:

```
array_name[index_#];
```

To call one of the values in an array, simply type the name of the array and the index of the value. Don't forget the index starts at 0, not 1, so to call the first element, use **`array_name[0];`**.

CODING CHALLENGES

CHANGE THE TEMPO OF THE SONG: Experiment with the `beatLength;` variable to change the tempo of the song.

MAKE YOUR OWN SONG: Try changing the notes to make a different song. Spaces "▮" can be used for rests in the song.

TROUBLESHOOTING

The song is too quiet or too loud	Turn the potentiometer to adjust the volume.
No sound is playing	Try pressing the reset button on the RedBoard. If that doesn't work, check your wiring of the buzzer. It's easy to misalign a pin with a jumper wire or reverse the buzzer.

You've completed Circuit 2A!

Continue to circuit 2B to explore digital inputs and buttons.

Circuit 2B: Digital Trumpet

Learn about digital inputs and buttons as you build your own digital trumpet! Buttons are all around us, from the keys on your keyboard to the buttons on your remote control.

YOU NEED


POTENTIOMETER

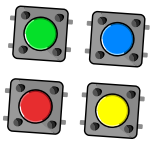

PIEZO BUZZER


10 JUMPER WIRES


3 PUSH BUTTONS

NEW COMPONENTS

BUTTONS: Also known as momentary switches, buttons only remain in their ON state as long as they're being actuated, or pressed. Most often momentary switches are best used for intermittent user-input cases: reset button and keypad buttons. These switches have a nice, tactile, "clicky" feedback when you press them.



Note that the different colors are just aesthetic. All of the buttons included behave the same, no matter their color.

NEW CONCEPTS

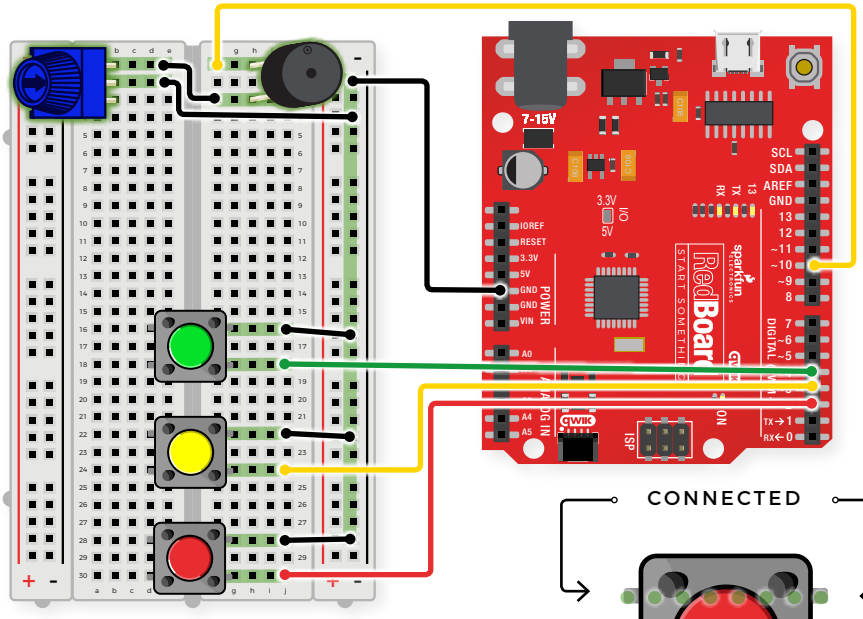
BINARY NUMBER SYSTEM: Number systems are the methods we use to represent numbers. We're most used to operating within the comfy confines of a base-10 number system, but there are many others. The base-2 system, otherwise known as binary, is common when dealing with computers and electronics. Computers, at their lowest level, really only have two ways to represent the state of anything: 1 or 0, which can also be thought of as ON or OFF, TRUE or FALSE, HIGH or LOW. Almost all electronics rely on a base-2 number system to store and manipulate numbers. The heavy reliance electronics places on binary numbers means it's important to know how the base-2 number system works.

DIGITAL INPUT: In circuit 1A, you worked with digital outputs. Each of the 14 digital pins can also be digital inputs. Digital inputs only care if something is in one of two states, 0 or 1. Digital inputs are great for determining if a button has been pressed or if a switch has been flipped.

PULL-UP RESISTORS: A pull-up resistor is a small circuit that holds the voltage HIGH (5V) on a pin until a button is pressed, pulling the voltage LOW (0V). The most common place you will see a pull-up resistor is when working with buttons. A pull-up resistor keeps the button in one state until it is pressed. The RedBoard has built-in pull-up resistors, but they can also be added to a circuit externally. This circuit uses the internal pull-up resistors, covered in more detail in the Code to Note section.

HOOKUP GUIDE

READY TO START HOOKING EVERYTHING UP? Check out the circuit diagram and hookup table below to see how everything is connected.



BUTTONS ARE NOT POLARIZED, but they do merit a closer look. Each row of legs is connected internally. When the button is pressed, one row connects to the other, connecting all four pins. If the button's legs don't line up with the rows on the breadboard, rotate it 90 degrees.

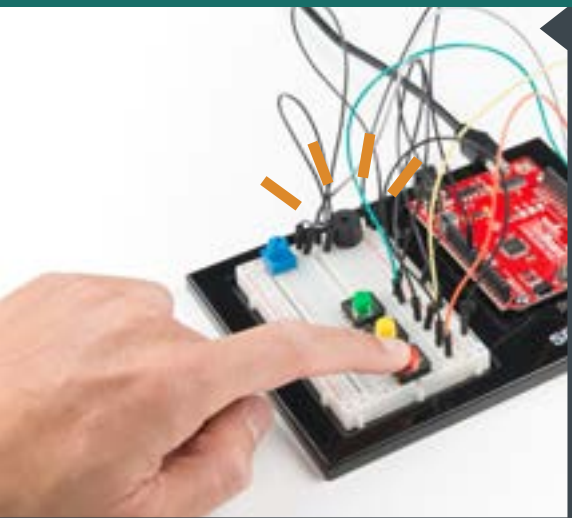
JUMPER WIRES	GND to GND(-) · D10 to F1 · D4 to J18 · D3 to J24 · D2 to J30 · E2 to GND (-) · J16 to GND (-) · J22 to GND (-) · J28 to GND (-) · E1 to F3
BUZZER	H1(+) to H3(-)
PUSH BUTTONS	D16/18 to G16/18 · D22/24 to G22/24 · D28/30 to G28/30
POTENTIOMETER	B1 + B2 + B3

Open the Arduino IDE

Connect the RedBoard to a USB port on your computer.

Open the Sketch:
File > Examples > SIK-Guide-Code-master > **SIK_CIRCUIT_2B-DIGITAL TRUMPET**

Select **UPLOAD** to program the sketch on the RedBoard.



WHAT YOU SHOULD SEE

Different tones will play when you press different keys. Turning the potentiometer will adjust the volume.

PROGRAM OVERVIEW

1	Check to see if the first button is pressed. **A:** If it is, play the frequency for c. **B:** If it isn't, skip to the next **else if** statement.
2	Check to see if the second button is pressed. **A:** If it is, play the frequency for e. **B:** If it isn't, skip to the next **else if** statement.
3	Check to see if the third button is pressed. **A:** If it is, play the frequency for g. **B:** If it isn't, skip to the **else** statement.
4	If none of the **if** statements are true, turn the buzzer off.

CODE TO NOTE

INTERNAL PULL-UP RESISTOR:

```
pinMode(pin, INPUT_PULLUP);
```

To declare a standard input, use the line `pinMode(pin, INPUT);`. If you would like to use one of the RedBoard's built-in pull-up 20kΩ resistors, it would look like this: `pinMode(pin, INPUT_PULLUP);`. The advantage of external pull-ups is being able to choose a more exact value for the resistor.

DIGITAL INPUT:

```
digitalRead(pin);
```

Check to see if an input pin is reading HIGH (5V) or LOW (0V). Returns TRUE (1) or FALSE (0) depending on the reading.

IS EQUAL TO:

```
if(digitalRead(pin) == LOW)
```

This is another logical operator. The "is equal to" symbol `==` can be confusing. Two equals signs are the same as asking, "Are these two values equal to one another?" Contrarily, one equals sign means assigning a particular value to a variable. Don't forget to add the second equals sign if you are comparing two values.

CODING CHALLENGES

CHANGE THE KEY OF EACH BUTTON: Use the frequency table in the comment section at the end of the code to change the notes that each button plays.

PLAY MORE THAN THREE NOTES WITH IF STATEMENTS: By using combinations of buttons, you can play up to seven notes of the scale. You can do this in a few ways. To get more practice with **if** statements, try adding seven **if** statements and using the Boolean AND `&&` operator to represent all of the combinations of keys.

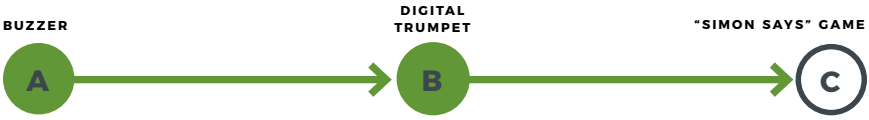
PLAY MORE THAN THREE NOTES WITH BINARY MATH: You can use a clever math equation to play more than three notes with your three keys. By multiplying each key by a different number, then adding up all of these numbers, you can make a math equation that produces a different number for each combination of keys.

TROUBLESHOOTING

The buzzer is too loud or too quiet	Turn the potentiometer to adjust the volume.
The RedBoard thinks one key is always pressed	Check your wiring. You may have GND and 5V backward if one or more buttons behave as though they're pressed all the time.
The buttons are not working	First, make sure that the wiring is correct. It is easy to misalign a wire with a button leg. Second, make sure that you have declared your buttons as inputs and have enabled the internal pull-up resistors with `INPUT_PULLUP`.

You've completed Circuit 2B!

Continue to circuit 2C and learn how to build your own game using buttons and LEDs.



Circuit 2C: "Simon Says" Game

The "Simon Says" game uses LEDs to flash a pattern, which the player must remember and repeat using four buttons. This simple electronic game has been a classic since the late 1970s. Now you can build your own!

NEW CONCEPTS

FOR LOOPS: A **for loop** repeats a section of code a set number of times. The loop works by using a counter (usually programmers use the letter "i" for this variable) that increases each loop until it reaches a stop value. Here's an example of a simple **for loop**:

```
for (int i = 0; i < 5; i++){
Serial.print(i);
}
```

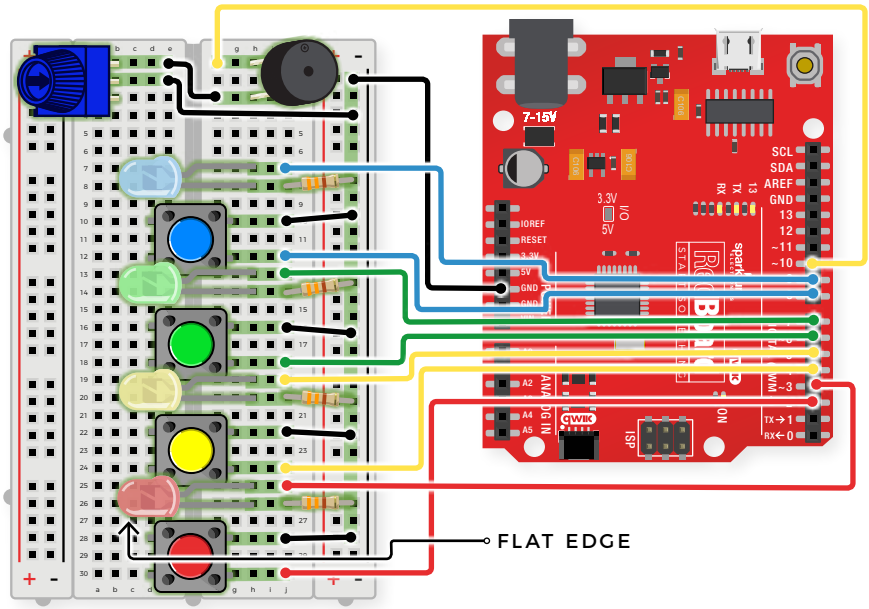
The **for loop** takes three parameters in the brackets, separated by semicolons. The first parameter is the start value. In this case, integer `i` starts at 0. The second value is the stop condition. In this case, we stop the loop when `i` is no longer less than 5 (i < 5 is no longer true). The final parameter is an increment value. `i++` is shorthand for increase i by 1 each time, but you could also increase i by different amounts. This loop would repeat five times. Each time it would run the code in between the brackets, which prints the value of `i` to the Serial Monitor.

MEASURING DURATIONS OF TIME WITH MILLIS(): The RedBoard has a built-in clock that keeps accurate time. You can use the `millis()` command to see how many milliseconds have passed since the RedBoard was last powered. By storing the time when an event happens and then subtracting the current time, you can measure the number of milliseconds (and thus seconds) that have passed. This sketch uses this function to set a time limit for repeating the pattern.

CUSTOM FUNCTIONS: This sketch uses several user-defined functions. These functions perform operations that are needed many times in the program (for example, reading which button is currently pressed or turning all of the LEDs off). Functions are essential to make more complex programs readable and compact.

HOOKUP GUIDE

READY TO START HOOKING EVERYTHING UP? Check out the circuit diagram and hookup table below to see how everything is connected.



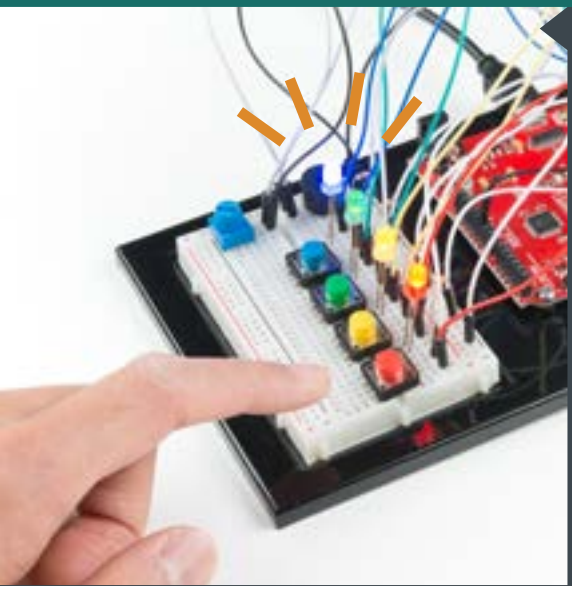
JUMPER WIRES	GND to GND(-) · D10 to F1 · D9 to J7 · D8 to J12 · D7 to J13 · D6 to J18 · D5 to J19 · D4 to J24 · D3 to J25 · D2 to J30 · E2 to GND(-) · E1 to F3 · J16 to GND(-) · J22 to GND(-) · J28 to GND(-) · J10 to GND(-)
BUZZER	H1(+) to H3(-)
LEDS	H7+ to H8– · H13+ to H14– · H19+ to H20– · H25+ to H26–
POTENTIOMETER	B1 + B2 + B3
PUSH BUTTONS	D10/12 to G10/12 · D16/18 to G16/18 · D22/24 to G22/24 · D28/30 to G28/30
330Ω RESISTOR	J8 to GND(-) · J14 to GND(-) · J20 to GND(-) · J26 to GND(-)

Open the Arduino IDE

Connect the RedBoard to a USB port on your computer.

Open the Sketch:
File > Examples > SIK-Guide-Code-master > **SIK_CIRCUIT_2C-SIMON SAYS**

Select **UPLOAD** to program the sketch on the RedBoard.



WHAT YOU SHOULD SEE

The circuit will flash all of the LEDs and play a melody. After a few seconds, it will flash the first light in the pattern. If you repeat the pattern correctly by pressing the corresponding colored button, then the game will move to the next round and add another color to the pattern sequence. If you make a mistake, the "Game Over" melody will play. If you get to round 10, the "You Win" melody will play. Press any button to start a new game.

PROGRAM OVERVIEW

1	Check if a new game is starting. If it is, play the start sequence. Reset the counter that keeps track of rounds, and randomly generate a sequence of numbers from 0 to 3 that controls which LEDs the user will have to remember.
2	The game works in rounds that progress from 0 to 10. Each round the game will flash LEDs in a pattern, and then the player has to recreate the pattern by pressing the button(s) that match the LED(s). In the first round, one LED will flash, and the player will have to press one button. In the eighth round, eight LEDs will flash, and the player will have to press eight buttons.
3	A loop is used to flash LEDs from the sequence until you have flashed the number of LEDs that matches the round number (1 for round 1, 2 for round 2, etc.).

4	Start a timer, and wait for the player to press a button. The player has 1.5 seconds to press the correct button. **A:** If the time limit runs out before a button is pressed, the player loses. **B:** If the player presses the wrong button, the player loses. **C:** If the player presses the right button, move on to the next number in the sequence. **D:** Repeat this process until the player has lost or correctly repeated the sequence for this round.
5	If the player repeats the entire sequence for that round, increase the round number by one (this will add one extra item to the end of the pattern). Then go back to step 3.
6	Keep incrementing the round until the player loses or finishes 10 rounds. If the player finishes 10 rounds, play the winning sequence.

CODE TO NOTE

ELAPSED TIME:

```
millis();
```

The `millis()` function returns the number of milliseconds that have passed since the RedBoard was last turned on.

BOOLEAN VARIABLES:

```
boolean variable_name;
```

The name for these variables comes from Boolean logic. The `boolean` variable type only has two values: 1 or 0 (also known as HIGH or LOW, ON or OFF, TRUE or FALSE). Using Boolean variables helps save memory on your microcontroller if you only need to know if something is true or false. Space in your microcontroller's memory is reserved when a variable is declared. How much memory is reserved depends on the type of variable.

STORING PIN NUMBERS IN ARRAYS:

```
int led[ ] = {3,5,7,9};
```

Sometimes you will want to cycle through all of the LEDs or buttons connected to a project. You can do this by storing a sequence of pin numbers in an array. The advantage of having pins in an array instead of a sequence of variables is that you can use a loop to easily cycle through each pin.

FUNCTIONS TO NOTE

```
flashLED(#LED to flash);
```

This turns one of the four LEDs on and plays the tone associated with it.
0 = Red, 1 = Yellow, 2 = Green, 3 = Blue.

FUNCTIONS TO NOTE

`allLEDoff();`	Turns all four LEDs off.
`buttonCheck();`	Uses `digitalRead()` to check which button is pressed. Returns 0, 1, 2 or 3 if one of the buttons is pressed. Returns 4 if no button is pressed.
`startSequence();`	Flashes the LEDs and plays tones in a sequence. Resets the round counter and generates a new random sequence for the user to remember.
`winSequence();`	Plays a sequence of tones, turns all of the LEDs on, and then waits for the player to press a button. If a button is pressed, restarts the game.
`loseSequence();`	Plays a sequence of tones, turns all of the LEDs on, and then waits for the player to press a button. If a button is pressed, restarts the game.

CODING CHALLENGES

CHANGE THE DIFFICULTY OF THE GAME: Change the difficulty of the game by changing how fast the player has to press each button or by increasing or decreasing the number of rounds needed to win. Note that if you increase the number of rounds to be larger than 16, you will need to change the size of the array (it is set at the top of the code in a line that looks like this: `int buttonSequence[16];`).

CHANGE THE SOUND EFFECTS: Try changing the sequence of notes that plays when you start, win or lose the game.

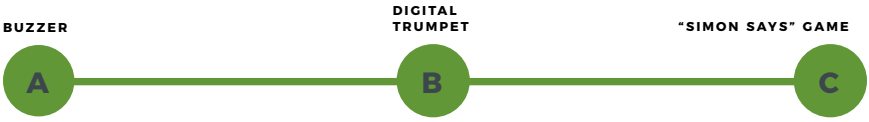
2-PLAYER MODE: Try changing the code so that two players can play head-to-head.

TROUBLESHOOTING

One of the LEDs isn't lighting up	Make sure your LED is oriented in the right direction. If the LED still doesn't work, try wiggling the resistor and the wires that connect to the LED.
The buzzer is too loud or too quiet	Turn the potentiometer to adjust the volume.
One of the buttons isn't working	Carefully check your wiring for each button. One leg of the button should connect to a pin on the RedBoard; the other leg should connect to the ground rail on the breadboard. Make sure they are declared correctly.
None of the buttons or LEDs are working	Make sure you don't have 5V and GND mixed up. Double check that you have a GND connection from the RedBoard to the GND rail on the breadboard.
Still not working?	Jumper wires unfortunately can go "bad" from getting bent too much. The copper wire inside can break, leaving an open connection in your circuit. If you are certain that your circuit is wired correctly and that your code is error-free and uploaded, but you are still encountering issues, try replacing one or more of the jumper wires for the component that is not working.

You've completed Project 2!

Continue to Project 3 to explore using servos and sensors.



PROJECT 3

Tired of your cat walking all over the kitchen counter? How about the dog getting into the garbage? Need a way to stop your younger sibling from sneaking into your bedroom? Learn how to protect against all of these annoyances as you build a multipurpose motion alarm. The alarm detects distance and motion using an **ultrasonic distance sensor**, and creates motion using a **servo motor**.

NEW COMPONENTS INTRODUCED IN THIS PROJECT

- SERVO MOTOR
- ULTRASONIC DISTANCE SENSOR

NEW CONCEPTS INTRODUCED IN THIS PROJECT

- PWM DUTY CYCLE
- ARDUINO LIBRARIES
- OBJECTS AND METHODS
- DIGITAL SENSORS
- DATASHEETS
- SERVO MECHANISMS

YOU WILL LEARN

- HOW TO CONTROL A SERVO MOTOR
- HOW TO USE AN ULTRASONIC DISTANCE SENSOR
- HOW TO MOVE OBJECTS USING SERVO MECHANISMS

Circuit 3A: Servo Motors

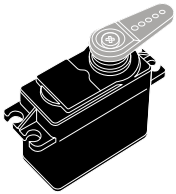
In this circuit, you will learn how to wire a servo and control it with code. Servo motors can be told to move to a specific position and stay there. Low-cost servo motors were originally used to steer RC airplanes and cars, but they have become popular for any project where precise movement is needed.

YOU NEED

SCISSORS
(NOT INCLUDED)

NEW COMPONENTS

SERVO MOTORS: Regular DC motors have two wires. When you hook the wires up to power, the motor spins around and around. Servo motors, on the other hand, have three wires: one for power, one for ground and one for signal. When you send the right signal through the signal wire, the servo will move to a specific angle and stay there. Common servos rotate over a range of about 0° to 180°. The signal that is sent is a **PWM signal**, the same used to control the RGB LED in Project 1.

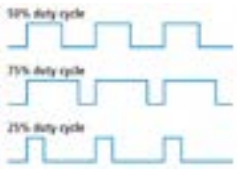


Included with your servo motor you will find a variety of motor mounts that connect to the shaft of your servo. You may choose to attach any mount you wish for this circuit. It will serve as a visual aid, making it easier to see the servo spin. The mounts will also be used at the end of this project.

ATTACHING YOUR SERVO: A strip of adhesive Dual Lock™ fastening tape is included in your kit. Cut two pieces of it to temporarily affix your servo to your baseplate.

NEW CONCEPTS

DUTY CYCLE: Pulse-Width Modulation (PWM) is a great way to generate servo control signals. The length of time a PWM signal is on is referred to as the duty cycle. Duty cycle is measured in percentage. Thus a duty cycle of 50 percent means the signal is on 50 percent of the time. The variation in the duty cycle is what tells the servo which position to go to in its rotation.



ARDUINO LIBRARIES: Writing code that sends precise PWM signals to the servo would be time consuming and would require a lot more knowledge about the servo. Luckily, the Arduino IDE has hundreds of built-in and user-submitted containers of code called libraries. One of the built-in libraries, the **Servo Library**, allows us to control a servo with just a few lines of code!

To use one of the built-in Arduino libraries, all you have to do is "include" a link to its header file. A header file is a smaller code file that contains definitions for all the functions used in that library. By adding a link to the header file in your code, you are enabling your code to use all of those library functions. To use the Servo Library, you would add the following line to the top of your sketch.

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

OBJECTS AND METHODS: To use the Servo Library, you will have to start by creating a servo object, like this:

```
Servo myServo;
```

Objects look a lot like variables, but they can do much more. Objects can store values, and they can have their own functions, which are called methods.

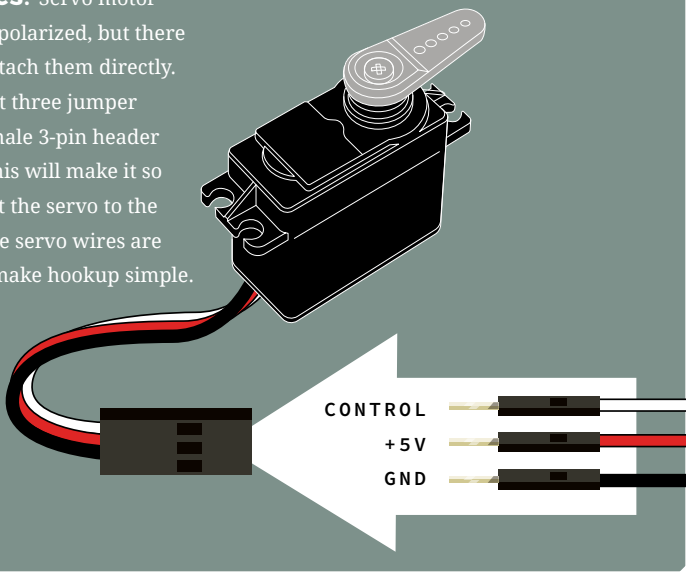
The most used method that a servo object has is `.write()`:

```
myServo.write(90);
```

The write method takes one parameter, a number from 0 to 180, and moves the servo arm to the specified position (in this case, degree 90).

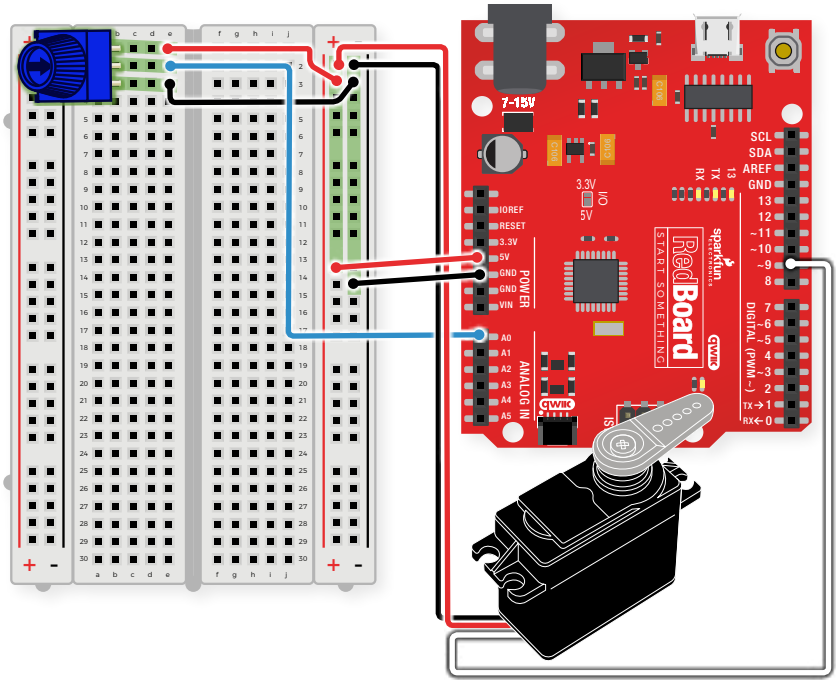
Why would we want to go to the trouble of making an object and a method instead of just sending a servo control signal directly over a pin? First, the servo object does the work of translating our desired position into a signal the servo can read. Second, using objects makes it easy for us to add and control more than one servo.

SERVO BASICS: Servo motor connectors are polarized, but there is no place to attach them directly. Instead, connect three jumper wires to the female 3-pin header on the servo. This will make it so you can connect the servo to the breadboard. The servo wires are color coded to make hookup simple.



HOOKUP GUIDE

READY TO START HOOKING EVERYTHING UP? Check out the circuit diagram and hookup table below to see how everything is connected.



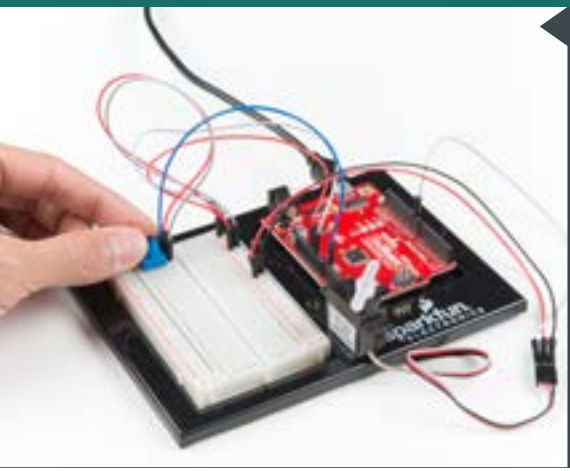
CONNECTION TYPES	◆ REDBOARD CONNECTION ■ BREADBOARD CONNECTION
JUMPER WIRES	◆ A0 *to* ■ E2 · ◆ 5V *to* ■ 5V(+) · ◆ GND *to* ■ GND(–) · ■ E1 *to* ■ 5V (+) · ■ E3 *to* ■ GND (-)
POTENTIOMETER	■ B1 + ■ B2 + ■ B3
SERVO LEADS	WHITE WIRE *to* ◆ D9 · RED WIRE *to* ■ 5V(+) · BLACK WIRE *to* ■ GND(–)

Open the Arduino IDE

Connect the RedBoard to a USB port on your computer.

Open the Sketch:
File > Examples > SIK-Guide-Code-master > **SIK_CIRCUIT_3A-SERVO**

Select **UPLOAD** to program the sketch on the RedBoard.



WHAT YOU SHOULD SEE

Turning the potentiometer will cause the servo arm to turn. The servo will mimic the movement of the potentiometer, twisting in the same clockwise or counter-clockwise direction. If you've attached a servo mount to the arm as shown, this movement will be easier to see.

PROGRAM OVERVIEW

1	Read the value of the potentiometer.
2	Convert the potentiometer value (0–1023) to an angle (20–160).
3	Tell the servo to go to this angle.

CODE TO NOTE

INCLUDING LIBRARIES:

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

The `#include` command adds a library to your Arduino program. After you include a library, you can use the commands in the library in your program. This line adds the built-in Servo Library.

CREATING SERVO OBJECTS:

```
Servo myServo;
```

The `Servo` command creates a new servo object and assigns a name to it, `myServo` in this case. If you make more than one servo object, you will need to give them different names.

CODE TO NOTE

SERVO ATTACH:

```
myServo.attach(9);
```

The `.attach();` method tells the servo object to which pin the signal wire is attached. It will send position signals to this pin. In this sketch, pin 9 is used. Remember to only use digital pins that are capable of PWM.

RANGE MAPPING:

```
map(potPosition,0,1023,20,160);
```

As shown in previous circuits, the analog pin values on your microcontroller vary from 0 to 1023. But what if we want those values to control a servo motor that only accepts a value from 0 to 180? The `map()` function takes a range of values and outputs a different range that can contain more or fewer values than the original. In this case, we are taking the range 0–1023 and mapping it to the range 20–160.

SERVO WRITE:

```
myServo.write(90);
```

The `.write();` method moves the servo to a specified angle. In this example, the servo is being told to go to angle 90.

CODING CHALLENGES

REVERSE THE SERVO DIRECTION: Try making the servo move in the opposite direction of the potentiometer.

CHANGE THE RANGE: Try altering the map function so that moving the potentiometer a lot only moves the servo a little or vice versa.

SWAP IN A DIFFERENT SENSOR: Try swapping a light sensor in for the potentiometer. Then you can make a dial that reads how much light is present!

TROUBLESHOOTING

The servo doesn't move	Check the wiring on your servo. Make sure the red wire on the servo cord is connected to 5V, the black wire is connected to GND and the white signal wire is connected to digital pin 9.
The servo is twitching	Although these servos are supposed to move from 0 to 180 degrees, sometimes sending them to the extremes of their range causes them to twitch (the servo is trying to move farther than it can). Make sure you aren't telling the servo to move outside of the 20–160 degree range.

You've completed Circuit 3A!

Continue to circuit 3B to learn about using distance sensors.

Circuit 3B: Distance Sensor

Distance sensors are amazing tools with all kinds of uses. They can sense the presence of an object, they can be used in experiments to calculate speed and acceleration, and they can be used in robotics to avoid obstacles. This circuit will walk you through the basics of using an ultrasonic distance sensor, which measures distance using sound waves!

YOU NEED


RGB LED

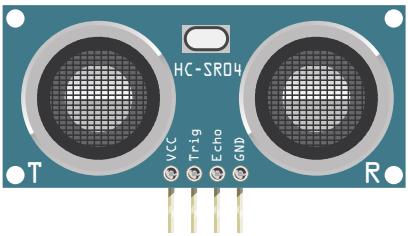

3 330Ω RESISTORS


DISTANCE SENSOR


10 JUMPER WIRES

NEW COMPONENTS

ULTRASONIC DISTANCE SENSOR: Distance sensors work by sending pulses of light or sound out from a transmitter, then timing how long it takes for the signals to bounce off an object and return to a receiver (just like sonar). Some sensors use infrared light, some use lasers, and some, like the HC-SR04 included in your kit, use ultrasonic sound (sound so high-pitched that you can't hear it).



NEW CONCEPTS

DATASHEETS: When working with electronics, datasheets are your best friend. Datasheets contain all the relevant information needed for a part. In this circuit, we are calculating distance based on the time it takes sound waves to be transmitted, bounce off an object and then be received. But, how can we tell distance from that information? The answer lies in the datasheet for the distance sensor. In it, you can find the equation the program needs to interpret the distance. View the datasheet at **http://sfe.io/HCSR04**.

ELSE IF STATEMENTS: In the night-light circuit, you used an **if/else** statement to run one set of code when a logic statement was true, and another when it was false. What if you wanted to have more than two options? **Else if** statements let you run as many logical tests as you want in one statement. For example, in the code for this circuit, there is an **if** statement that flows like this:

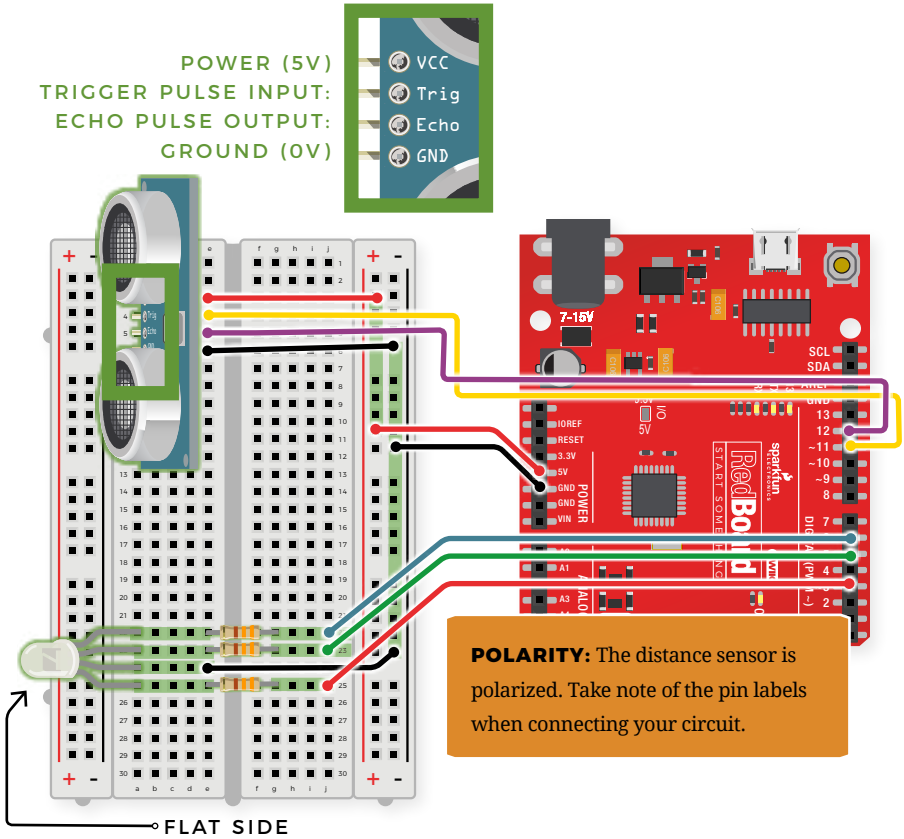
1. **If** the distance is less than 10, make the RGB LED red.
2. **Else if** the distance is more than 10 but less than 20, make the RGB LED yellow.
3. **Else** make the RGB LED green.

To have four or five colors for different distances, add more **else if** statements.

Else if statements are different from nested **if** statements in that only one of the statements above can be true, whereas multiple nested **if** statements could be true.

HOOKUP GUIDE

READY TO START HOOKING EVERYTHING UP? Check out the circuit diagram and hookup table below to see how everything is connected.



POLARITY: The distance sensor is polarized. Take note of the pin labels when connecting your circuit.

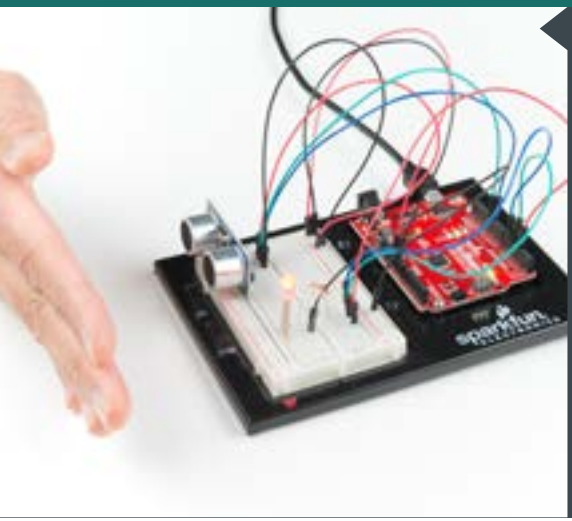
CONNECTION TYPES	◆ REDBOARD CONNECTION ■ BREADBOARD CONNECTION
JUMPER WIRES	◆ 5V to ■ 5V ◆ GND to ■ GND (-) ◆ D3 to ■ J25 ◆ D5 to ■ J23 ◆ D6 to ■ J22 ◆ D11 to ■ E4 ◆ D12 to ■ E5 ■ E3 to ■ 5V (+) ■ E6 to ■ GND(–) ■ E24 to ■ GND(–)
RGB LED	■ A25(RED) + ■ A24(GND) + ■ A23(GREEN) + ■ A22(BLUE)
330Ω RESISTORS (ORANGE, ORANGE, BROWN)	■ E22 to ■ G22 ■ E23 to ■ G23 ■ E25 to ■ G25
DISTANCE SENSOR	■ A3(VCC) + ■ A4(TRIG) + ■ A5(ECHO) + ■ A6(GND)

Open the Arduino IDE

Connect the RedBoard to a USB port on your computer.

Open the Sketch:
File > Examples > SIK-Guide-Code-master > **CIRCUIT_3B-DISTANCE SENSOR**

Select **UPLOAD** to program the sketch on the RedBoard.



WHAT YOU SHOULD SEE

Move your hand or a large, flat object closer and farther away from the distance sensor. As the object approaches, the light will change from green to yellow to red. Open the Arduino Serial Monitor to see the distance being read from the sensor.

PROGRAM OVERVIEW

Check what distance the sensor is reading.

1: If the distance is less than 10 inches, make the RGB LED red.
2: If the distance is between 10 and 20 inches, make the RGB LED yellow.
3: If the distance value is not equal to the fist two conditions, make the RGB LED green.

TROUBLESHOOTING WARNING: HVAC systems in offices and schools have been known to interfere with the performance of the ultrasonic distance sensor. If you are experiencing sporadic behavior from your circuit, check your surroundings. If there are numerous air ducts in the room you are using, try moving to a different room that does not have ducts. The airflow from these ducts can interfere with the waves sent from the sensor, creating noise and resulting in bad readings.

CODE TO NOTE

FLOAT VARIABLES:

```
float echoTime;
```

The `float` variable, short for **floating-point number**, is similar to an integer except it can represent numbers that contain a decimal point. Floats are good for representing values that need to be more precise than an integer. Floats allow us to measure precise distances such as 9.33 inches instead of just 9 inches.

ELSE IF STATEMENT:

```
if(logic statement){
//run if first is true
}

else if(second logic
statement){
//run if second is true
}

else{
//run if neither is true
}
```

Else if statements let you combine more than one logic statement. Arduino will test each logic statement in order; if one is true it will run the code in that section and then skip all of the other sections of code in the remaining statements.

USER-DEFINED FUNCTION:

```
getDistance();
```

This function tells the distance sensor to send out an ultrasonic wave form, measures the time it takes to bounce back to the sensor, and then calculates the distance based on the speed of sound. This calculation is based off information found in the distance sensor's datasheet.

CODING CHALLENGES

CHANGE THE LIMITS OF THE DISTANCE SENSOR: Try editing the values in the logic statements so that the RGB LED changes color at different distances.

CHANGE THE UNITS OF THE DISTANCE SENSOR: Try editing the code so that the distance sensor outputs a different unit of length, such as centimeters or feet.

ADD A FOURTH COLOR: Try adding another **else if** statement so that there are four different colors instead of three.

TROUBLESHOOTING

The RGB LED colors aren't working or a color is missing	Check the connection for the wire and resistor connected to each leg of the LED. Ensure the RGB LED is inserted in the correct orientation.
The distance sensor doesn't seem to work	Open up the Serial Monitor on your computer. You should see a stream of distances being printed in the monitor. If they are all reading 0 or jumping around, then check the wiring on your sensor.
The distance sensor still doesn't work	Ultrasonic noise pollution will interfere with your distance sensor readings. If you aim two distance sensors at each other, they will confuse each other. Some air-conditioning systems may also emit noises in the ultrasonic range. Try pointing your sensor away from the other distance sensors or changing to a different location.

You've completed Circuit 3B!

Continue to circuit 3C to explore building mechanisms that interact with your circuits.

Circuit 3C: Motion Alarm

Time to take your distance sensor project to the next level. Let's imagine you want to stop your cat from prowling around your countertop. This circuit uses light, sound and motion to scare away your cat when it is detected by the distance sensor. Using a servo motor, you can add a moving pop-up to animate your alarm.

NEW CONCEPTS

MECHANISMS: This circuit gets really fun when you start to use your servo to animate the world around you. To do this, you'll need to connect your servo to some physical mechanisms. Tape and hot glue are easy ways to connect things to your servo. You can also loop a paper clip through the small holes in the servo arm to serve as a linkage.

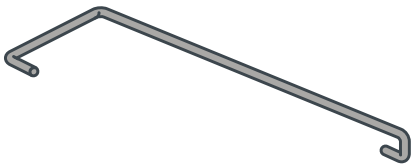
Linkage rods are found on many RC airplanes, which use servo motors to control the ailerons, elevators and rudder.

ASSEMBLY

If you have opted for the extra materials, use the following instructions to create the moving pop-up for your motion alarm.

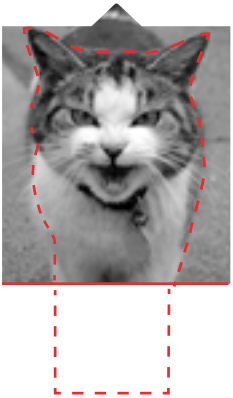
1. Attach the servo mount of your choice. The motor mounts also come with screws to secure the mount to the motor. Once you are finished with this circuit, you may choose to add a screw to make for a more robust mechanism. It is recommended you upload your code and test the mechanism before screwing it down.

2. Use needle-nose pliers to bend the paper clip straight. Bend about 1 inch of the paper clip 90 degrees. Then bend the other end so it's about 1/8 inch long. Repeat this bend once more, making a hook shape. You should now have a linkage rod that looks something like this:



3. Attach the hook end of the linkage rod to the end hole on your servo mount. The motor should be reattached to the baseplate with Dual Lock.

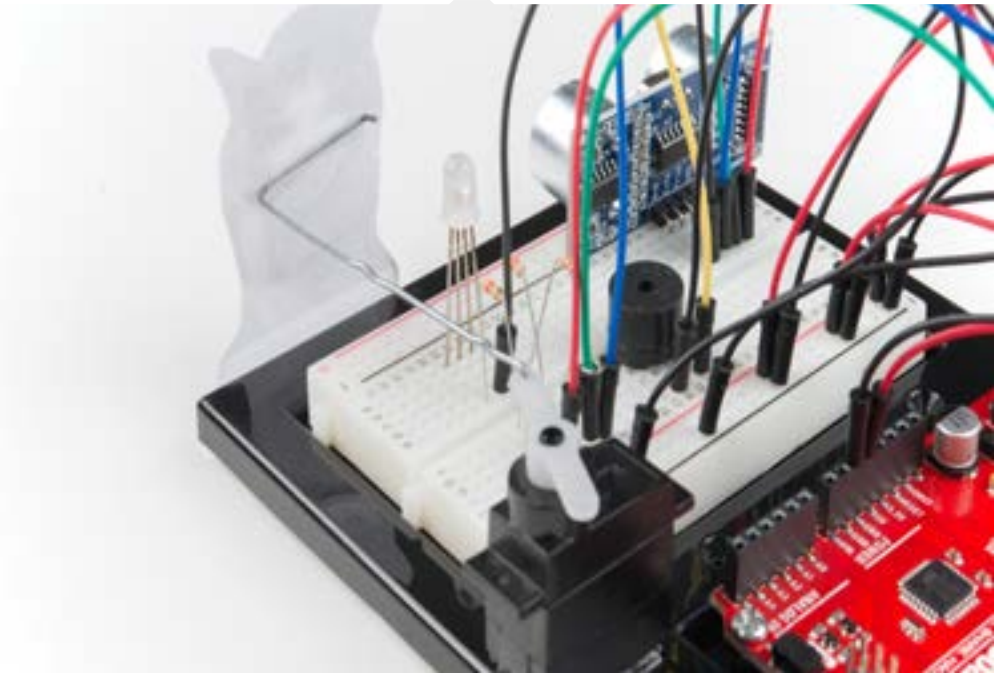
4. Cut out the pop-up image of your choice. We chose this public domain menacing cat image (**http://sfe.io/cat**). The image you choose should be about 2.5 inches x 2.5 inches and can be drawn or printed. Leave a rectangular strip of paper under the image that is about 2 inches long.



5. Fold along the bottom of the image. Tape the bottom of the pop-up to the underside of the breadboard baseplate on the same side to which the servo is connected.

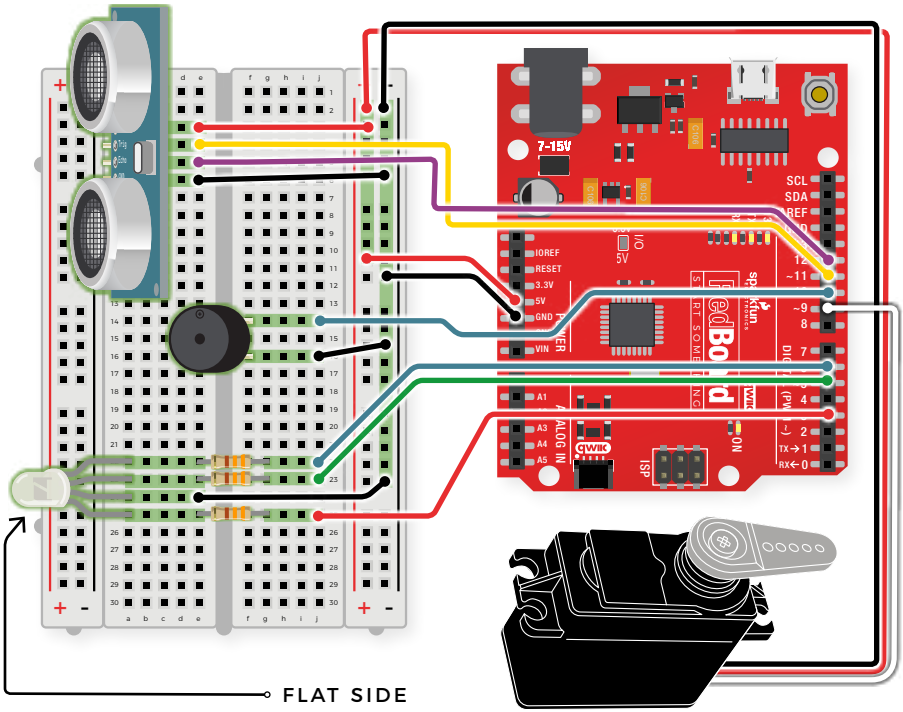
6. Tape the free end of the rod to the back of your pop-up image, near the center.

7. Once you have the rest of the circuit built and the code uploaded, you can fine-tune your moving pop-up and make any necessary adjustments. Remember to wait until these adjustments have been made before you screw the servo mount onto the motor.



HOOKUP GUIDE

READY TO START HOOKING EVERYTHING UP? Check out the circuit diagram and hookup table below to see how everything is connected.



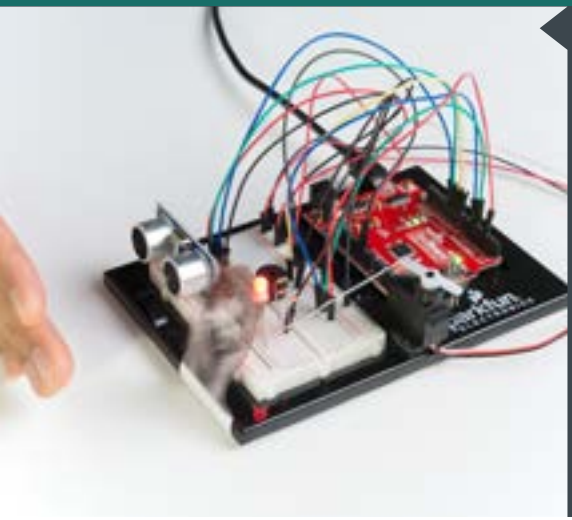
JUMPER WIRES	5V to 5V	GND to GND (-)	D3 to J25	D5 to J23
	D6 to J22	D11 to E4	D12 to E5	D10 to J14
	E3 to 5V (+)	E6 to GND(–)	E24 to GND(–)	J16 to GND(–)
RGB LED	A25(RED)	A24(GND)	A23(GREEN)	A22(BLUE)
330Ω RESISTORS (ORANGE, ORANGE, BROWN)	E22 to F22	E23 to F23	E25 to F25	
DISTANCE SENSOR	A3(VCC)	A4(TRIG)	A5(ECHO)	A6(GND)
PIEZO BUZZER	F14 (+)	F16(-)		
SERVO LEADS	WHITE WIRE to D9	RED WIRE to 5V(+)	BLACK WIRE to GND(–)	

Open the Arduino IDE

Connect the RedBoard to a USB port on your computer.

Open the Sketch:
File > Examples > SIK-Guide-Code-master > **SIK_CIRCUIT_3C-MOTION ALARM**

Select **Upload** to program the sketch on the RedBoard.



WHAT YOU SHOULD SEE

The RGB LED will behave as in your last circuit. It will be green when objects are far, yellow when they are midrange and red when they are close. When an object is close, the buzzer will also beep, and the servo will rotate back and forth. If you decided to attach a pop-up, it will move back and forth.

PROGRAM OVERVIEW

Check what distance the sensor is reading.

1: If the distance is less than 10 inches, make the RGB LED red. Then make the servo rotate back and forth and make the buzzer beep.
2: If the distance is between 10 and 20 inches, make the RGB LED yellow.
3: If the distance value is not equal to the fist two conditions, make the RGB LED green.

CODE TO NOTE

CONSTANTS:

```
const int trigPin
= 11;
```

Constants are variables that have been marked as “read-only” and cannot have their value changed as the program progresses. Constants are great for declaring pin number variables that will not change throughout the program.

CODE TO NOTE

NO TONE FUNCTION:

```
noTone(pin_number);
```

In circuit 2A, you made songs using a buzzer and the `tone()` function, but you gave the function three parameters: a pin number, a frequency and a duration. You can leave out the third parameter, and the tone will play until you change it or turn it off. `noTone()` turns off a pin that has been activated with the `tone()` command.

CODING CHALLENGES

CHANGE THE SERVO BEHAVIOR: Try changing the way your servo behaves when the distance sensor is triggered.

CHANGE THE ALARM SETTINGS: Try altering the code so the alarm goes off from much farther or closer distances.

ADD A SECOND MECHANISM: Time to use your imagination! Try your hand at making different objects move with your servo motor. Don't have a cat? Make an interactive pop-up story, room alarm, treat dispenser or automatic fish feeder.

TROUBLESHOOTING

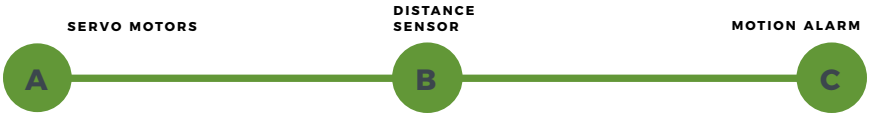
The RGB LED colors aren't working or a color is missing	Check the connection for the wire and resistor connected to each leg of the LED. Ensure the RGB LED is inserted in the correct orientation.
The distance sensor doesn't seem to work	Open up the Serial Monitor on your computer. You should see a stream of distances being printed in the monitor. If they are all reading 0 or jumping around, check the wiring on your sensor.
The distance sensor still doesn't work	Ultrasonic noise pollution will interfere with your distance sensor readings. If you aim two distance sensors at each other, they will confuse each other. Some air-conditioning systems may also emit noises in the ultrasonic range. Try pointing your sensor away from the other distance sensors or moving to a different location.

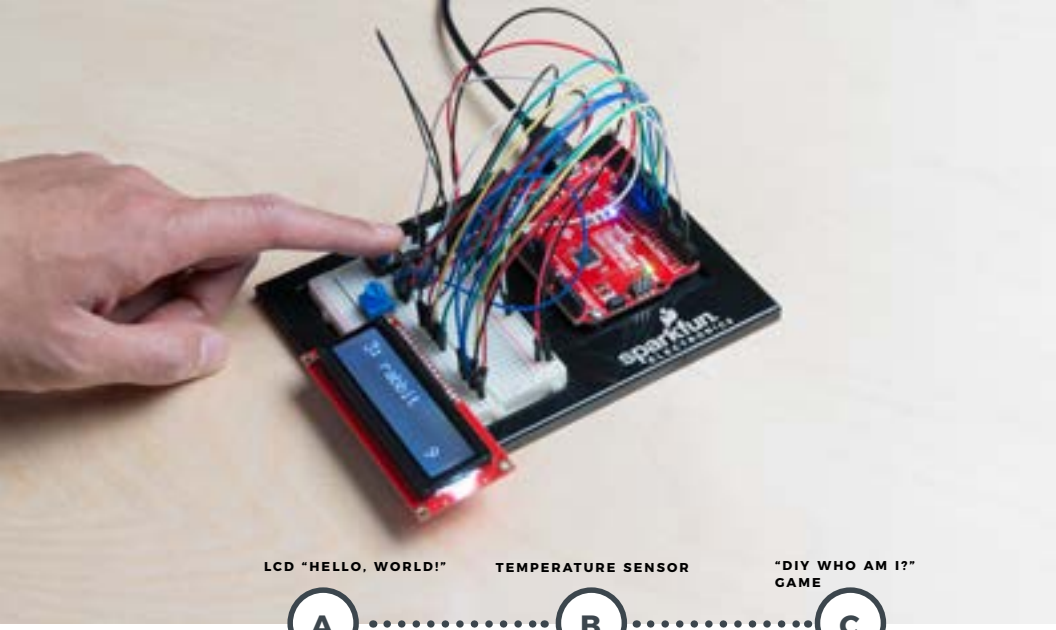
TROUBLESHOOTING

The servo doesn't work	Make sure all of your servo wires are connected. Be sure that the black wire is connected to the negative rail and the red wire is connected to the positive rail. Make sure you are using a digital pin that is capable of PWM.
The pop-up is moving too much or not enough	The two lines of code that pass angles to the servo motor are `myservo.write(45);` and `myservo.write(135);`. Try changing these angle values to fine-tune your mechanism.

You've completed Circuit 3C!

Continue to Project 4 to learn how to use an LCD in your circuits.





LCD "HELLO, WORLD!" A — TEMPERATURE SENSOR B — "DIY WHO AM I?" GAME C

PROJECT 4

Printing data to the Arduino Serial Monitor is a great way to see data from the RedBoard. But, what if you want to make your project mobile and see sensor values away from your computer? This project will show you how to do exactly that. You'll learn about **Liquid Crystal Displays (LCDs)** and how to print things like sensor data and strings of words to the display.

NEW COMPONENTS INTRODUCED IN THIS PROJECT

- LIQUID CRYSTAL DISPLAY (LCD)
- TMP36 DIGITAL TEMPERATURE SENSOR
- 4XAA BATTERY HOLDER

NEW CONCEPTS INTRODUCED IN THIS PROJECT

- CONTRAST
- PIXELS
- ALGORITHMS
- BUTTON DEBOUNCE
- STRINGS
- POINTERS

YOU WILL LEARN

- HOW TO PRINT SIMPLE MESSAGES TO AN LCD
- HOW TO USE A TEMPERATURE SENSOR
- HOW TO PRINT SENSOR DATA TO AN LCD
- HOW TO MAKE AN INTERACTIVE GAME THAT INCORPORATES THE LCD

Circuit 4A: LCD "Hello, World!"

Printing "Hello, world!" is usually the first thing that programming tutorials will have you do in a new language. This guide starts by blinking an LED, but now we're going to print out real text using a Liquid Crystal Display (LCD).

YOU NEED

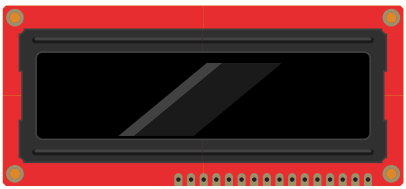

LCD DISPLAY


POTENTIOMETER


16 JUMPER WIRES

NEW COMPONENTS

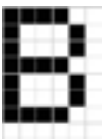
CHARACTER LIQUID CRYSTAL DISPLAY (LCD): Designed to show a grid of letters, numbers and other special characters, LCDs are great for printing data and showing values. When current is applied to this special kind of crystal, it turns opaque. This is used in a lot of calculators, watches and simple displays. Adding an LCD to your project will make it super portable and allow you to integrate up to 32 characters (16x2) of information.



NEW CONCEPTS

CONTRAST: Pin 3 on the LCD controls the contrast and brightness of the LCD. Using a simple voltage divider with a potentiometer, the contrast can be adjusted. As you rotate the knob on the potentiometer, you should notice that the screen will get brighter or darker and that the characters become more visible or less visible. The contrast of LCDs is highly dependent on factors such as temperature and the voltage used to power them. Thus, external contrast knobs are needed for displays that cannot automatically account for temperature and voltage changes.

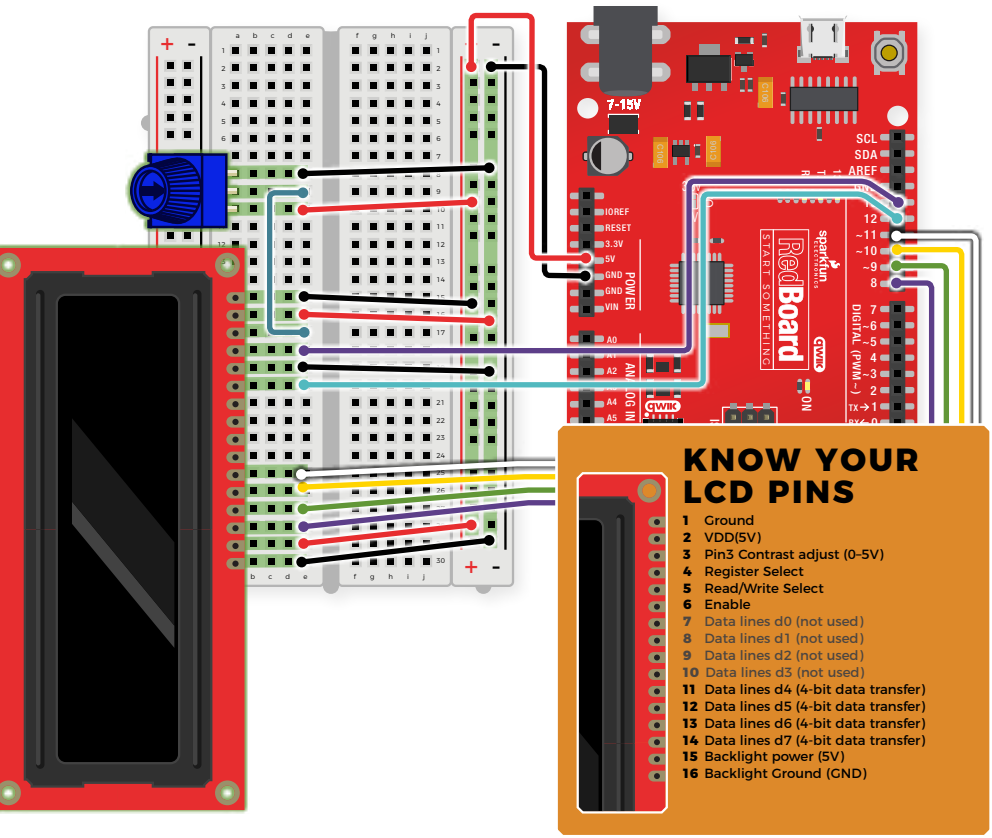
PIXELS: If you look closely at the characters on the LCD, you will notice that they are actually made up of lots of little squares. These little squares are called **pixels**. The size of displays is often represented in pixels. Pixels make up character space, which is the number of pixels in which a character can exist.



Here is a capital letter B as created in pixels. The character space in this example is 6 pixels x 8 pixels.

HOOKUP GUIDE

READY TO START HOOKING EVERYTHING UP? Check out the circuit diagram and hookup table below to see how everything is connected.



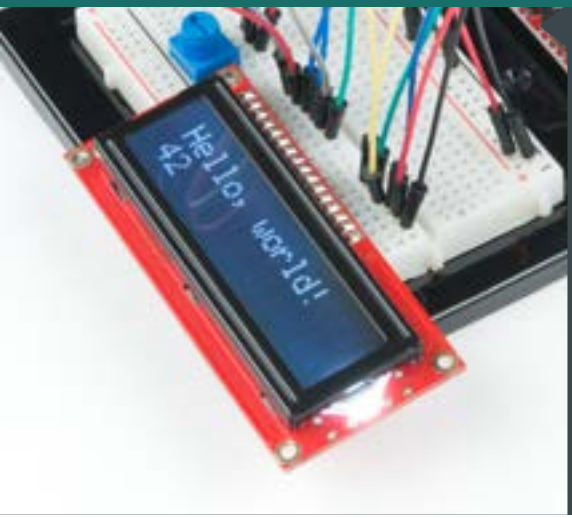
JUMPER WIRES	5V to 5V · GND to GND (-) · D8 to E28 · D9 to E27 · D10 to E26 · D11 to E25 · D12 to E20 · D13 to E18 · E30 to GND (-) · E29 to 5V(+) · E19 to GND (-) · E16 to 5V(+) · E15 to GND (-) · E9 to E17 · E8 to GND (-) · E10 to 5V(+)
LCD DISPLAY	A15-A30 *(pin 1 on A15)*
POTENTIOMETER	A8 + A9 + A10

Open the Arduino IDE

Connect the RedBoard to a USB port on your computer.

Open the Sketch:
File > Examples > SIK-Guide-Code-master > **CIRCUIT_4A-LCD HELLO WORLD**

Select **UPLOAD** to program the sketch on the RedBoard.



WHAT YOU SHOULD SEE

The LCD screen will show "Hello, world!" and on the row below a counter will count every second that passes.

Adjusting the potentiometer will change the contrast on the LCD screen.

PROGRAM OVERVIEW

1	Import the LCD library.
2	Make an LCD object called "lcd" that will be controlled using pins 8, 9, 10, 11, 12 and 13.
3	"Begin" the LCD. This sets the dimensions of the LCD that you are working with (16 x 2). It needs to be called before any other commands from the LCD library are used.
4	Clear the display.
5	Set the cursor to the top left corner `lcd.setCursor(0,0);` then print "Hello, world!"
6	Move the cursor to the first space of the lower line `lcd.setCursor(0,1);` then print the number of seconds that have passed since the RedBoard was last reset.

CODE TO NOTE

LCD LIBRARY:

```
#include <LiquidCrystal.h>
```

Includes the LiquidCrystal library in your program.

LCD LIBRARY INSTANCE:

```
LiquidCrystal LCD_name(RS_pin,
enable_pin, d4, d5, d6, d7);
```

As with servos, you need to create an LCD object and give it a name (you can make more than one). The numbers in the brackets are pins on the RedBoard that connect to specific pins on the LCD.

LCD BEGIN:

```
lcd.begin(16, 2);
```

This line initializes the LCD object and tells the program the LCD's dimensions. In this case it is 2 rows of 16 characters each.

LCD CLEAR:

```
lcd.clear();
```

This method clears all the pixels on the display.

LCD CURSOR:

```
lcd.setCursor(0,0);
```

Moves the cursor to a point on the 16x2 grid of characters. Text that you write to the LCD will start from the cursor. This line is starting back at position (0,0).

LCD PRINT:

```
lcd.print("Hello, world!");
```

Prints a string of characters to the LCD starting at the cursor position.

CODING CHALLENGES

CHANGE THE MESSAGE: Try changing the code to display another message.

SHOW HOURS, MINUTES AND SECONDS: Try adding some code so that the display shows the hours, minutes and seconds that have passed since the RedBoard was last reset.

COUNT BUTTON PRESSES: By adding a button to the circuit, you can count the number of times the button was pressed or have the button change what displays.

TROUBLESHOOTING

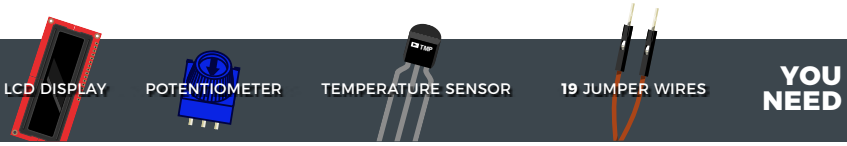
The screen is blank or flickering	Adjust the contrast by twisting the potentiometer. Try both directions until you see characters display. Do not twist the potentiometer past its stopping points. Also, check the potentiometer, and make sure it's wired correctly.
Not working at all	Double check the circuit's wiring. There are a lot of wires in this circuit, and it's easy to mix up one or two.
Rectangles in first row	If you see 16 rectangles (like "■") on the first row, it may be due to the jumper wires being loose on the breadboard. This is normal and can happen with other LCDs wired in parallel with a microcontroller. Make sure that the wires are fully inserted into the breadboard, then try pressing the reset button and adjusting the contrast using the potentiometer.
Still not working?	Jumper wires unfortunately can go "bad" from getting bent too much. The copper wire inside can break, leaving an open connection in your circuit. If you're certain that your circuit is wired correctly and that your code is error-free and uploaded but you are still encountering issues, try replacing one or more of the jumper wires for the component that is not working.

You've completed Circuit 4A!

Continue to circuit 4B to learn about using temperature sensors.

Circuit 4B: Temperature Sensor

Want to create a DIY environmental monitor or weather station? You can use a small, low-cost sensor like the TMP36 to make devices that track and respond to temperature. In this activity you will also use the LCD screen to display sensor readings, a common use for LCDs in electronics projects.



NEW COMPONENTS

TMP36 TEMPERATURE SENSOR: This temperature sensor has three legs. One connects to 5V, one to ground, and the voltage output from the third leg varies proportionally to changes in temperature. By doing some simple math with this voltage, we can measure temperature in degrees Celsius or Fahrenheit.

NEW CONCEPTS

ALGORITHMS: An algorithm is a process used in order to achieve a desired result. Often, the information needed to create an algorithm lives in the part's datasheet. This sketch uses a few formulas to turn a voltage value into a temperature value, making them all part of the larger temperature-retrieving algorithm. The first formula takes the voltage read on analog pin 0 and multiplies it to get a voltage value from 0V–5V:

```
voltage = analogRead(A0) * 0.004882813;
```

The number we are multiplying by comes from dividing 5V by the number of samples the analog pin can read (1024), so we get: 5 / 1024 = 0.004882813.

The second formula takes that 0–5V value and calculates degrees Celsius:

```
degreesC = (voltage - 0.5) * 100.0;
```

The reason 0.5V is subtracted from the calculated voltage is because there is a 0.5V offset, mentioned on page 8 of the TMP36 datasheet found here: **http://sfe.io/TMP36**. It's then multiplied by 100 to get a value that matches temperature.

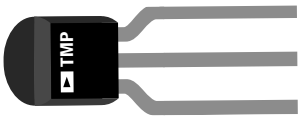
The last formula takes the Celsius temperature and converts it to a Fahrenheit temperature using the standard conversion formula:

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

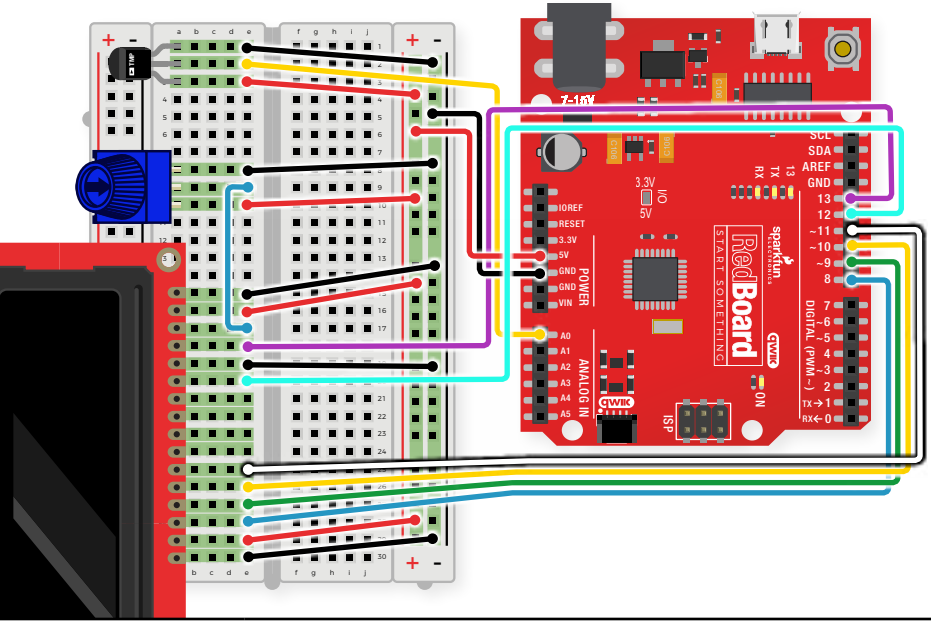
Together, these three formulas make up the algorithm that converts voltage to degrees Fahrenheit.

HOOKUP GUIDE

READY TO START HOOKING EVERYTHING UP? Check out the circuit diagram and hookup table below to see how everything is connected.



HEADS UP! Double check the polarity of the TMP36 temperature sensor before powering the RedBoard. It can become very hot if it is inserted backward!



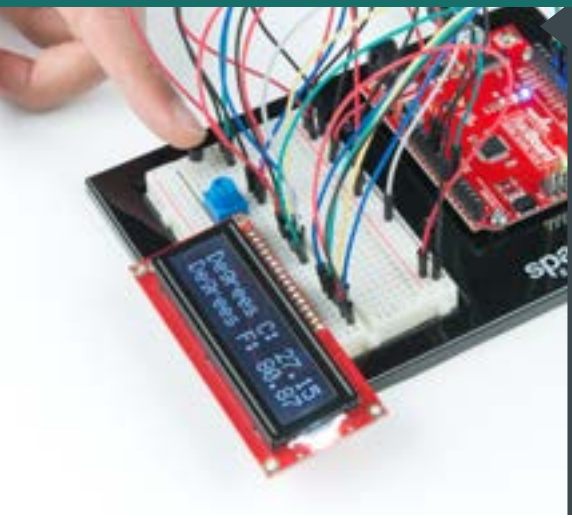
JUMPER WIRES	5V to 5V · GND to GND(-) · D8 to E28 · D9 to E27 · D10 to E26 · D11 to E25 · D12 to E20 · D13 to E18 · A0 to E2 · E30 to GND(-) · E29 to 5V(+) · E19 to GND(-) · E16 to 5V(+) · E15 to GND(-) · E9 to E17 · E8 to GND(-) · E10 to 5V(+) · E1 to GND(-) · E3 to 5V(+)
LCD SCREEN	A15-A30 *(pin 1 on A15)*
TMP36 SENSOR	A1 (GND) + A2 (SIG) + A3(5V)
POTENTIOMETER	A8 + A9 + A10

Open the Arduino IDE

Connect the RedBoard to a USB port on your computer.

Open the Sketch:
File > Examples > SIK-Guide-Code-master > **SIK_CIRCUIT_4B-TEMPERATURE SENSOR**

Select **UPLOAD** to program the sketch on the RedBoard.



WHAT YOU SHOULD SEE

The LCD will show the temperature in Celsius and Fahrenheit. The temperature readings will update every second. An easy way to see the temperature change is to press your finger to the sensor.

PROGRAM OVERVIEW

1	Get the analog value from the TMP36 and convert it back to a voltage between 0 and 5V.
2	Calculate the degrees Celsius from this voltage.
3	Calculate the degrees Fahrenheit from this voltage.
4	Clear the LCD.
5	Print the degrees C with a label on the first row.
6	Print the degrees F with a label on the second row.
7	Wait for a second before taking the next reading.

CODE TO NOTE

VOLTAGE CONVERSION ALGORITHMS

Many of the sensors that you will use with your microcontroller work by changing a voltage in some predictable way in response to a property of the world (like temperature, light or magnetic fields). Often, you will need to build an algorithm that converts these voltages to the desired value and units. The temperature sensor is a great example of this code. We use three equations to convert a voltage value into degrees in C and F.

```
voltage = analogRead(A0) * 0.004882813;
degreesC = (voltage - 0.5) * 100.0;
degreesF = degreesC * (9.0/5.0) + 32.0;
```

CODING CHALLENGES

DISPLAY THE TEMPERATURE IN DEGREES KELVIN: Try adding an equation so that the temperature is displayed in degrees Kelvin. (You will have to look up the formula for converting from degrees Celsius or Fahrenheit to Kelvin.)

DISPLAY A BAR GRAPH: By changing the code you can display the temperature as a bar graph instead of a number.

DISPLAY VALUES FROM ANOTHER SENSOR: You can swap out the TMP36 for a potentiometer, photoresistor or other sensor and display the new set of values.

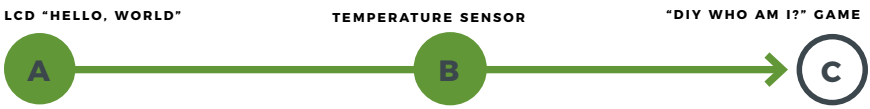
ADD AN RGB LED: Add an RGB LED that changes color based on the temperature.

TROUBLESHOOTING

Sensor is heating up	Make sure that you wired the temperature sensor correctly. The temperature sensor can get warm to the touch if it is wired incorrectly. Disconnect power from your microcontroller, rewire the circuit, and connect it back to your computer.
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. Also, make sure that the wires are connected properly to the temperature sensor.
Values not printing to screen	If you see text but no temperature values, there could be an error in your code. If you see no text at all, adjust the LCD contrast.

You've completed Circuit 4B!

Continue to circuit 4C to learn how to make a "DIY Who Am I?" game.



Circuit 4C: "DIY Who Am I?" Game

"DIY Who Am I?" is based on the popular Hedbanz game or HeadsUp! app. It's a fun party game in which a player holds an LCD screen to his/her forehead, and other players give hints to help the player with the LCD guess the word on the screen.

YOU NEED


LCD DISPLAY


POTENTIOMETER


PUSH BUTTON


PIEZO BUZZER


20 JUMPER WIRES

AA BATTERY HOLDER

DUAL LOCK TAPE

4 AA BATTERIES (NOT INCLUDED)

SCISSORS

NEW COMPONENTS

4XAA BATTERY HOLDER: Included in your kit is a 4-cell AA battery holder. The 5-inch cable is terminated with a standard barrel jack connector. The connector mates with the barrel jack on the RedBoard, allowing you to easily make your project battery powered.

NEW CONCEPTS

BUTTON DEBOUNCE: When working with momentary buttons, it is usually necessary to add button debouncing to your code. This is because the code that is meant to execute when the button is pressed may execute faster than you can press and release the button (microcontrollers are fast!). The simplest way to debounce a button is to add a small delay to the end of your code. This sketch adds a 500 millisecond delay at the end of `loop()` to account for this. This simple addition will prevent a word from getting skipped when you press the button for the game.

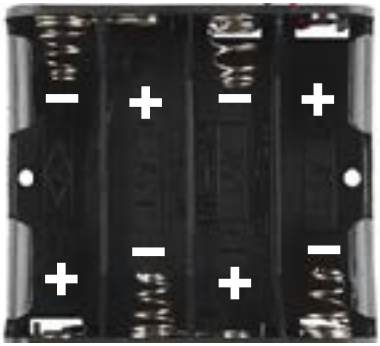
For a more complex example of button debouncing, in the Arduino IDE open **File > Examples > 02.Digital > Debounce.**

STRINGS: Strings are used to print words and even sentences to an LCD or the Serial Monitor. Strings are actually just an array of characters with a null character at the end to let the program know where the end of the string is.

ARRAY OF STRINGS: In circuit 2A you used an array of characters to represent musical notes. In this program, you'll want to make an array of strings. Strings use multiple characters to make words, so you'll need to use a little trick to put them in an array. The trick is to use a pointer. When you declare your array, you'll use an asterisk (*) after the `char` data type, as follows:

```
const char* arrayOfStrings =
{"Feynman" "Sagan", "Tyson",
"Nye"};
```

POINTERS: As an advanced programming topic, pointers can be difficult to understand at first. For now, think of pointers as a variable that "points" to the value contained in a certain address in memory. In this sketch, the `char*` variable points to `arrayOfStrings` address and returns the character values to create a list of strings.



BATTERY HOLDER ASSEMBLY

Batteries are polarized. They have a positive end and a negative end. The battery holder has images indicating which end goes in which orientation for each cell.

To attach the battery holder to the breadboard baseplate, first cut two strips of Dual Lock that are roughly 1 inch x 1 inch each, or 2.5cm x 2.5cm.

Remove the adhesive backing, and attach one piece to the back of the battery holder.

Adhere the second piece to the bottom of the breadboard baseplate (directly in the middle is recommended, as this will come into play in Project 5).

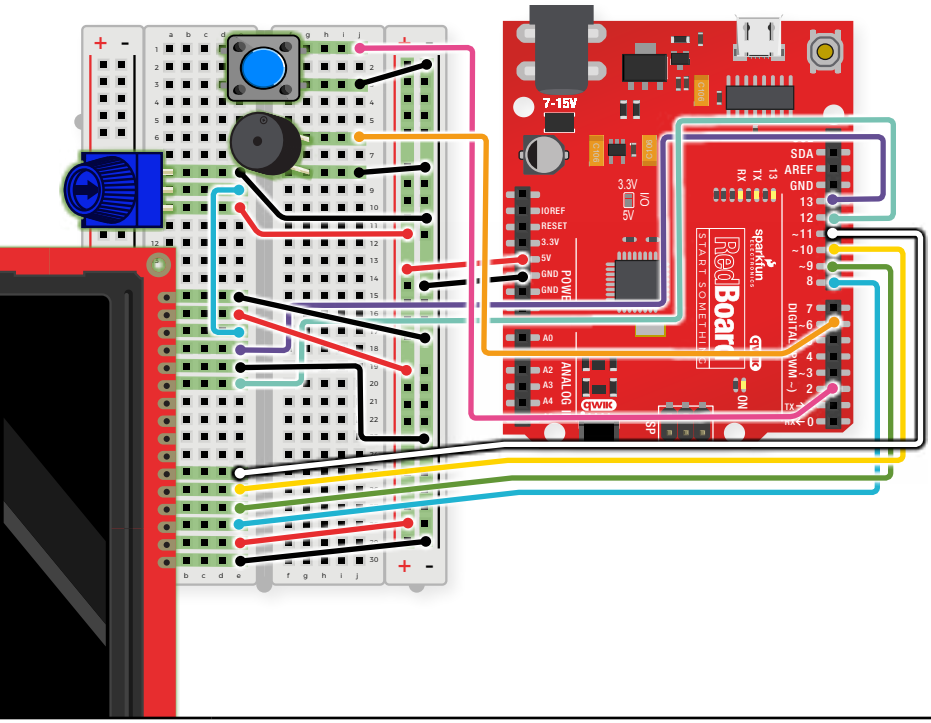
Last, press the battery holder to the baseplate so that the two pieces of Dual Lock snap together. Insert the batteries into the holder. Remember that batteries are polarized. Remove the pack before building the circuit, so it doesn't slide around.

STOP!

Disconnect the battery pack from power while building your circuit. Working on your circuit while connected to a power source risks damaging your components.

HOOKUP GUIDE

READY TO START HOOKING EVERYTHING UP? Check out the circuit diagram and hookup table below to see how everything is connected.



JUMPER WIRES	5V to 5V · GND to GND(-) · D8 to E28 · D9 to E27 · D10 to E26 · D11 to E25 · D12 to E20 · D13 to E18 · D6 to J6 · D2 to J1 · E30 to GND(-) · E29 to 5V(+) · E19 to GND(-) · E16 to 5V(+) · E15 to GND(-) · E9 to E17 · E8 to GND(-) · E10 to 5V(+) · J8 to GND(-) · J3 to GND(-)
LCD SCREEN	A15-A30 *(pin 1 on A15)*
PUSH BUTTON	D1/D3 to G1/G3
POTENTIOMETER	A8 to A9 to A10
BUZZER	G6(+) to G8(–)

Open the Arduino IDE

Connect the RedBoard to a USB port on your computer.

Open the Sketch:
File > Examples > SIK-Guide-Code-master > **SIK_CIRCUIT_4C-DIY WHO AM I**

Select **UPLOAD** to program the sketch on the RedBoard.

WHAT YOU SHOULD SEE

The game begins with the category of words, then runs through a short countdown. When the first round starts, the word to be guessed is displayed at top left, and a countdown starts in the bottom right. Each time the button is pressed (before the timer expires) a new word is displayed. If you win or lose, a short song will play.

PROGRAM OVERVIEW

1	Generate a random order for the words to be displayed.
2	Show the starting countdown on the LCD.
3	Start a loop that will run 25 times (there are 25 words total). For each round: **A:** Print the round number and the word to be guessed. **B:** Display a countdown timer in the lower right-hand corner of the screen that counts down the time limit for each round. **C:** If the time limit runs out, play the losing song, print "Game Over" and show the player's final score. **D:** If the player presses the button before the time limit is up, advance to the next word.
4	If the player gets through all 25 words, play the winning song and print "YOU WIN!"

CODE TO NOTE

ARRAY OF STRINGS:

```
const char* array_name [array_length] =
{“string1”, “string2”...};
```

Makes an array of strings. The strings are stored as constants, so they can’t be changed once the program starts.

ROUNDING FUNCTION:

```
round(value_to_round);
```

This math function rounds a number up or down to the nearest whole number.

RANDOM FUNCTION:

```
random(min, max);
```

This function takes a set of numbers and generates a pseudo-random number from that set.

BUTTON DEBOUNCE:

```
delay(500);
```

This 500 millisecond delay at the end of the loop adds button debounce so that erroneous button presses are not detected by the RedBoard.

FUNCTIONS TO NOTE

`generateRandomOrder();`	Makes an array that is a random ordering of the numbers from 1–25. This is used to display words for the game in a random order.
`showStartSequence();`	Shows the category of words on the LCD, then displays a countdown before the game starts.
`gameOver();`	Plays a sound and shows the text “Game Over” along with the player’s final score.
`winner();`	Shows the text “YOU WIN!” and plays a winning sound.

CODING CHALLENGES

CHANGE THE TIME LIMIT: Changing the time limit variable will change the difficulty of the game.

CHANGE THE WORDS IN THE WORD LIST: Try changing the categories and words. The number of words in your words array must match the value of the variable `arraySize`.

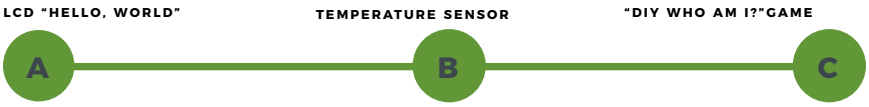
CHANGE THE WINNING AND LOSING SONGS: By changing the tones in the `winner()` and `gameover()` functions you can change which song plays at the end of the game.

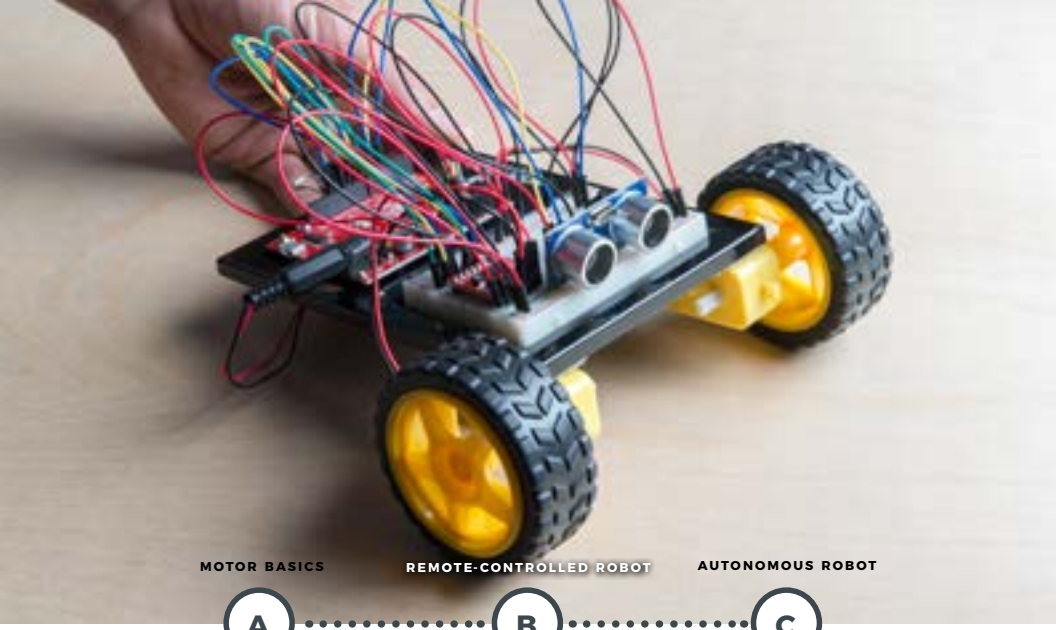
TROUBLESHOOTING

The screen is blank or flickering	Adjust the contrast by twisting the potentiometer. If it's incorrectly adjusted, you won't be able to read the text. Also, check the potentiometer to make sure it's connected correctly.
No sound is coming from the buzzer	Check the wiring to the buzzer and the polarity. Make sure you are using the correct pin as defined in your code. You may add a potentiometer volume knob if you desire.
The button doesn't work or words are getting skipped before they are guessed	If the button isn't working, check your wiring. If words are being skipped when the button is pressed, increase the debounce delay found at the end of the loop. It should be 500 milliseconds by default. Increasing this number by tiny increments will help with this problem.

You've completed Circuit 4C!

Continue to Project 5 to learn how to build your first robot!





MOTOR BASICS A — REMOTE-CONTROLLED ROBOT B — AUTONOMOUS ROBOT C

PROJECT 5

Ah, robots. One of the most iconic and exciting electronics applications. In this project, you will learn all about **DC motors** and **motor drivers** by building your own robot! You'll first learn motor control basics. Then you'll control a tethered robot by sending it commands over serial. Last, you will unleash your robot by removing the tether and making it autonomous! By adding a distance sensor, the robot can learn how to avoid obstacles.

NEW COMPONENTS INTRODUCED IN THIS PROJECT

- TB6612FNG MOTOR DRIVER
- SWITCH
- DC GEARMOTOR
- WHEEL

NEW CONCEPTS INTRODUCED IN THIS PROJECT

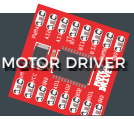
- INPUT VOLTAGE
- INTEGRATED CIRCUITS
- H-BRIDGE MOTOR DRIVER
- ASCII CHARACTERS
- CONVERTING STRINGS
- AUTONOMOUS VEHICLES

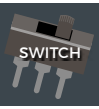
YOU WILL LEARN

- HOW TO CONTROL A MOTOR USING A MOTOR DRIVER
- HOW TO SEND SERIAL COMMANDS TO CREATE A REMOTE-CONTROLLED ROBOT
- HOW TO BUILD A ROBOT THAT USES SENSORS TO REACT TO ITS ENVIRONMENT

Circuit 5A: Motor Basics

In this circuit, you will learn the basic concepts behind motor control. Motors require a lot of current, so you can't drive them directly from a digital pin on the RedBoard. Instead, you'll use what is known as a motor controller or motor driver board to power and spin the motor accordingly.





YOU NEED

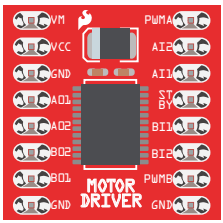
NEW COMPONENTS

SWITCHES are components that control the open-ness or closed-ness of an electric circuit. Just like the momentary buttons used in earlier circuits, this type of switch can only exist in one of two states: open or closed. However, a switch is different in that it will stay in the position it was last in until it is switched again.

THE MOTORS in your Inventor's Kit have two main parts: a small DC motor that spins quickly and a plastic gearbox that gears down the output from the hobby motor so that it is slower but stronger, allowing it to move your robot. The motors

have a clever design allowing you to attach things that you want to spin fast (like a small fan or flag) to the hobby motor, and things that you want to be strong (like a wheel) to the plastic axle sticking out the side of the motor. The included wheels just so happen to fit on the plastic axles.

TB6612FNG MOTOR DRIVER: If you switch the direction of current through a motor by swapping the positive and negative leads, the motor will spin in the opposite direction. Motor controllers

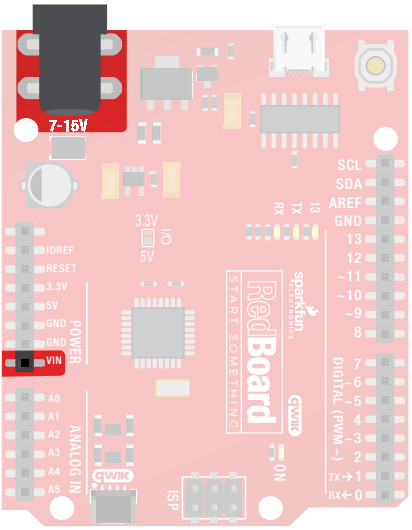


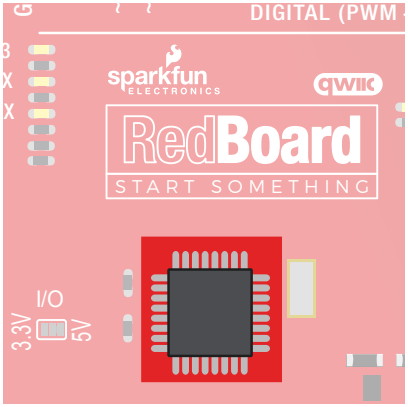
contain a set of switches (called an H-bridge) that lets you easily control the direction of one or more motors. The TB6612FNG Motor Driver takes commands for each motor over three wires (two wires control direction, and one controls speed), and then uses these signals to control the current through two wires attached to your motor.

NEW CONCEPTS

VOLTAGE IN (VIN): This circuit utilizes the VIN pin found with the other RedBoard power pins. The VIN pin outputs a voltage that varies based on whatever voltage the RedBoard is powered with. If the RedBoard is powered through the USB port, then the

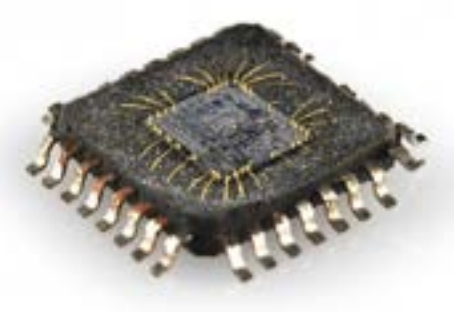
voltage on VIN will be about 4.6–5V. However, if you power the RedBoard through the barrel jack (highlighted in the picture), the VIN pin will reflect that voltage. For example, if you were to power the barrel jack with 9V, the voltage out on VIN would also be 9V. Notice that the voltage range listed on the RedBoard near the barrel jack is 7–15V. This means that the recommended input voltage should always be at or above 7V or should be at or below 15V. Never exceed this range.





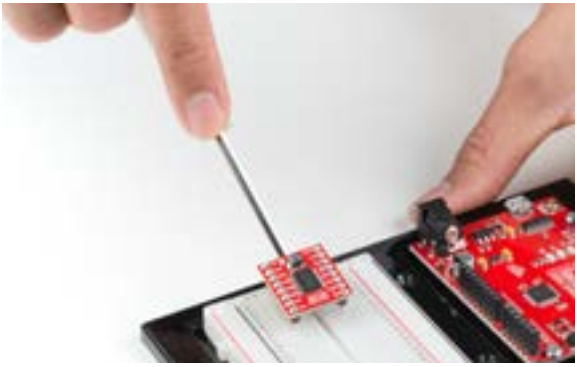
INTEGRATED CIRCUITS (ICS) AND BREAKOUT BOARDS: An Integrated Circuit (IC) is a collection of electronic components — resistors, transistors, capacitors, etc. — all stuffed into a tiny chip and connected together to achieve a common goal. They come in all sorts of flavors, shapes and sizes. The chip that powers the RedBoard, the ATmega328, is an IC. The chip on the motor driver, the TB6612FNG, is another IC.

Integrated circuits are often too small to work with by hand. To make working with ICs easier and to make them breadboard-compatible, they are often added to a breakout board, which is a printed circuit board that connects all the IC's tiny legs to larger ones that fit in a breadboard. The motor driver board in your kit is an example of a breakout board.



The guts of an integrated circuit, visible after removing the top.

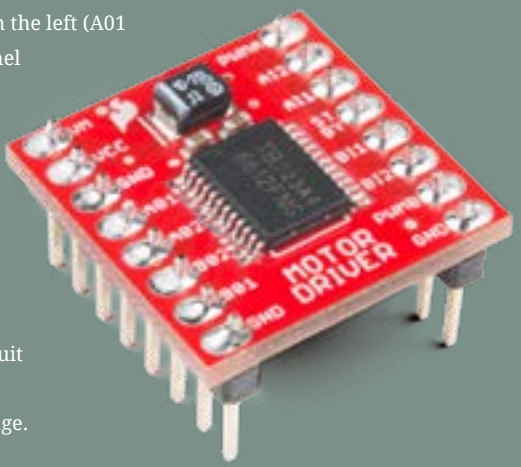
Once you're finished with this project, removing the motor driver from the breadboard can be difficult due to its numerous legs. To make this easier, use the included screwdriver as a lever to gently pry it out. Be careful not to bend the legs as you remove it.



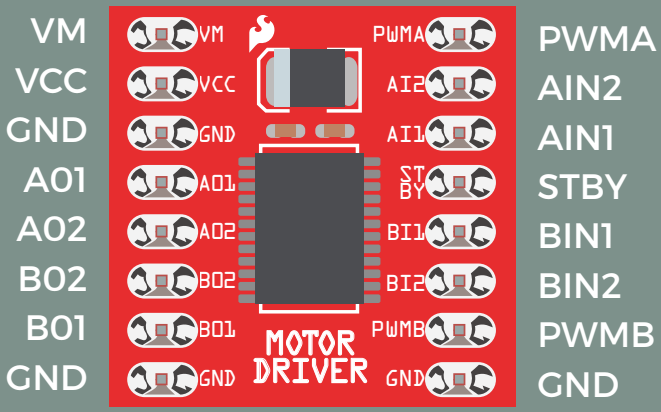
The motors are polarized. However, motors are unique in that they will still work when the two connections are reversed. They will just spin in the opposite direction when hooked up backward. To keep things simple, always think of the red wire as positive (+) and the black wire as negative (-).

MEET YOUR MOTOR CONTROLLER.

The TB6612FNG Motor Driver may look complicated, but it's easy to use. Three pins on the right (PWMA, A12 and A11) control the two pins on the left (A01 and A02). The same is true for channel B. Motors require more current, which is why the VIN voltage is needed.



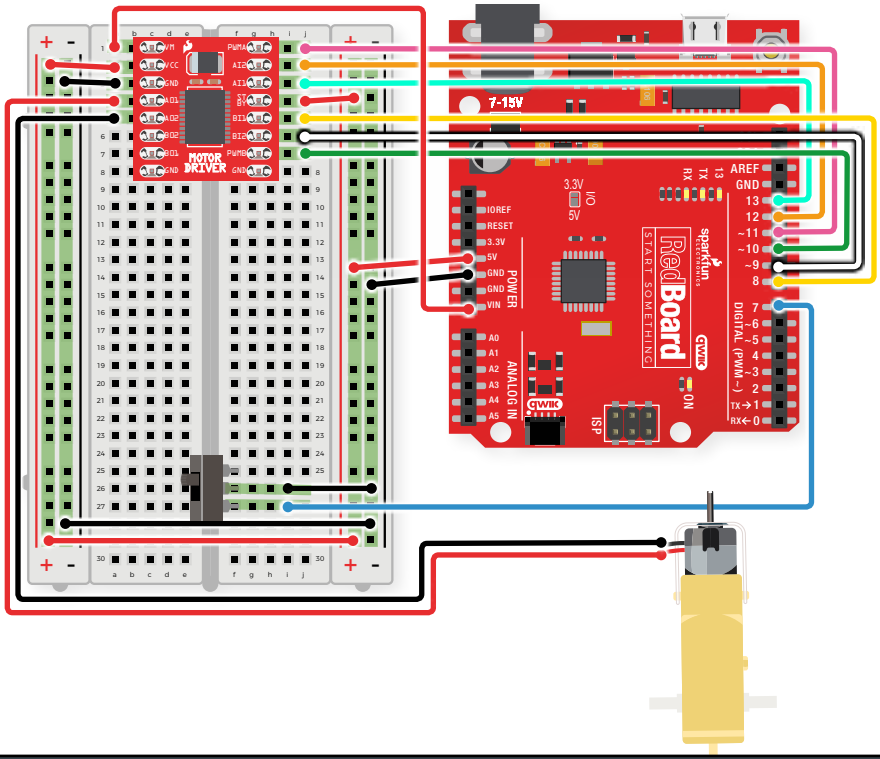
Most ICs have polarity and usually have a polarity marking in one of the corners. The motor driver is no exception. Be sure to insert the motor driver as indicated in the circuit diagrams. The motor driver pins are explained in the table on the next page.



PIN LABEL	FUNCTION	POWER/INPUT/ OUTPUT	NOTES
VM	Motor Voltage	Power	This is where you provide power for the motors (2.2V to 13.5V)
VCC	Logic Voltage	Power	This is the voltage to power the chip and talk to the microcontroller (2.7V to 5.5V)
GND	Ground	Power	Common Ground for both motor voltage and logic voltage (all GND pins are connected)
STBY	Standby	Input	Allows the H-bridges to work when high (has a pull-down resistor, so it must actively be pulled high)
AIN1/BIN1	Input 1 for channels A/B	Input	One of the two inputs that determine the direction
AIN2/BIN2	Input 2 for channels A/B	Input	One of the two inputs that determine the direction
PWMA/ PWMB	PWM input for channels A/B	Input	PWM input that controls the speed
A01/B01	Output 1 for channels A/B	Output	One of the two outputs to connect the motor
A02/B02	Output 2 for channels A/B	Output	One of the two outputs to connect the motor

HOOKUP GUIDE

READY TO START HOOKING EVERYTHING UP? Check out the circuit diagram and hookup table below to see how everything is connected.



CONNECTION TYPES ◆ REDBOARD CONNECTION ■ BREADBOARD CONNECTION

JUMPER WIRES

- ◆ 5V *to* ■ 5V
- ◆ GND *to* ■ GND (-)
- ◆ VIN *to* ■ A1
- ◆ D8 *to* ■ J5
- ◆ D9 *to* ■ J6
- ◆ D10 *to* ■ J7
- ◆ D11 *to* ■ J1
- ◆ D12 *to* ■ J2
- ◆ D13 *to* ■ J3
- ◆ D7 *to* ■ I27
- ■ 5V (+) *to* ■ 5V (+)
- ■ GND (-) *to* ■ GND (-)
- ■ A2 *to* ■ 5V (+)
- ■ A3 *to* ■ GND (-)
- ■ J4 *to* ■ 5V (+)
- ■ I26 *to* ■ GND (-)

MOTOR

- ■ A4(RED +)
- ■ A5(BLACK -)

MOTOR DRIVER

- ■ C1-C8 *to* ■ G1-G8 *(VM on C1, PWMA on G1)*

SWITCH

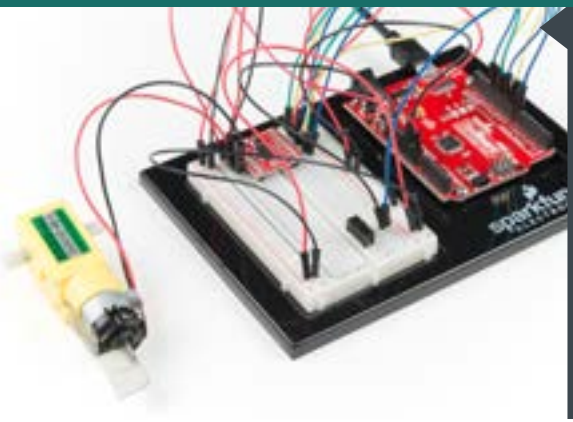
- ■ F25 + ■ F26 + ■ F27

Open the Arduino IDE

Connect the RedBoard to a USB port on your computer.

Open the Sketch:
File > Examples > SIK-Guide-Code-master > **CIRCUIT_5A-MOTOR BASICS**

Select **UPLOAD** to program the sketch on the RedBoard.



WHAT YOU SHOULD SEE

Flip the switch. The motor will spin at the speed set by the motor speed variable (default is 0). Open the Serial Monitor, type any number from 30 to 255 or -30 to -255 , and then press Enter. Changes in speed will be hard to notice. Send 0 to stop the motor.

PROGRAM OVERVIEW

1	Check to see if a command has been sent through the Serial Monitor. If a command has been sent, then set the motor speed to the number that was sent over the Serial Monitor.
2	Check to see if the switch is ON or OFF. **A:** If the switch is ON, drive the motor at the motor speed. **B:** If the switch is OFF, stop the motor.

CODE TO NOTE

PARSING INTEGERS:

`Serial.parseInt();`

`.parseInt()` receives integer numbers from the Serial Monitor. It returns the value of the number that it receives, so you can use it like a variable.

SERIAL AVAILABLE:

`Serial.available();`

This command checks how many bytes of data are being sent to the RedBoard. If it is greater than 0, then a message has been sent. It can be used in an **if** statement to run code only when a command has been received.

CODING CHALLENGES

MAKE THE SWITCH CHANGE DIRECTIONS: Change the code so that the position of the switch changes the direction of the motor instead of turning it on and off.

REPLACE THE SWITCH WITH A BUTTON: Try wiring a button into the circuit instead of the sliding switch. Now the motor only turns on when you push the button.

REPLACE THE SWITCH WITH A SENSOR: Try changing the code so that the motor is activated by another sensor, like the photoresistor.

TROUBLESHOOTING

Motor not spinning	Check the wiring to the motor driver. There are a lot of connections, and it's easy to mix one of them up with another. Double check the polarity of the motor driver. All the text should face the same direction as everything else.
Switch not working	Make sure that you are hooked up to the middle pin and one side pin on the switch, and not both side pins.
Still not working?	Jumper wires unfortunately can go "bad" from getting bent too much. The copper wire inside can break, leaving an open connection in your circuit. If you are certain that your circuit is wired correctly and that your code is error-free and uploaded but you are still encountering issues, try replacing one or more of the jumper wires for the component that is not working.

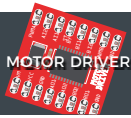
You've completed Circuit 5A!

Continue to circuit 5B to construct a remote-controlled robot.

Circuit 5B: Remote-Controlled Robot

In this circuit, you'll control two motors and build your own remote-controlled roving robot! You will also learn how to read information from a serial command so that you can use the Serial Monitor to tell the robot in what direction to move and how far to move.

YOU NEED


MOTOR DRIVER


2 GEAR MOTORS


SWITCH


16 JUMPER WIRES

2 WHEELS

DUAL LOCK TAPE

BINDER CLIP

SCISSORS (NOT INCLUDED)

NEW CONCEPTS

ASCII CHARACTERS: ASCII is a standard for character encoding, formalized in the 1960s, that assigns numbers to characters. When typing on a computer keyboard, each character you type has a number associated with it. This is what allows computers to know whether you are typing a lowercase "a," an uppercase "A" or a random character such as ampersand (&). In this experiment, you will be sending characters to the Serial Monitor to move your remote-controlled robot. When you send a character, the microcontroller is interpreting that as a specific number. ASCII tables available online (**http://sfe.io/ASCII**) make it easier to know which character is represented by which number.

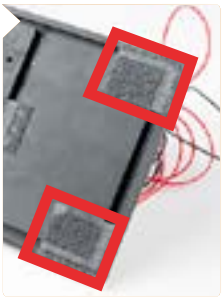
CONVERTING STRINGS TO INTEGERS: String variables hold words like "dog" or "Robert Smith" that are made up of multiple characters. Arduino has a set of special built-in methods for string variables that you can call by putting a period after the variable name, as follows:

```
string_variable_name.toInt();
```

The `.toInt()` method converts the string to a number, and there are a dozen other methods that can do things like tell you the length of a word or change all of the characters in a string to uppercase or lowercase.

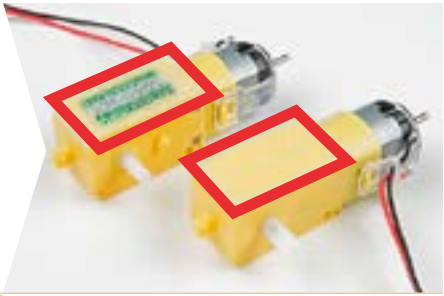
ASSEMBLY

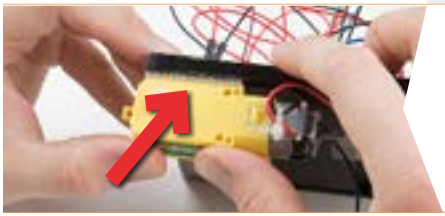
Before you build this circuit, you'll need to make a few modifications to the breadboard baseplate to make it more robot-like!



1. Cut and attach two short pieces of Dual Lock tape to the very corners of the baseplate on the side located under the breadboard.

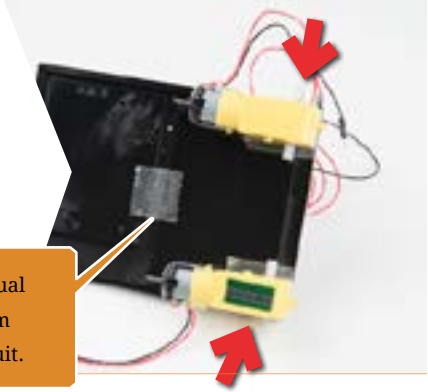
2. CUT TWO MORE STRIPS that are 1.25 inches (3.175cm) long and ¾ inch (1.9cm) wide. Remove the adhesive backing, and attach the strips to the two motors. Be sure that your motors are mirror images of each other when you attach the Dual Lock.





3. PRESS THE MOTORS TO THE BASEPLATE, connecting the two Dual Lock surfaces. Try to get the motors as straight as possible so your robot will drive straight.

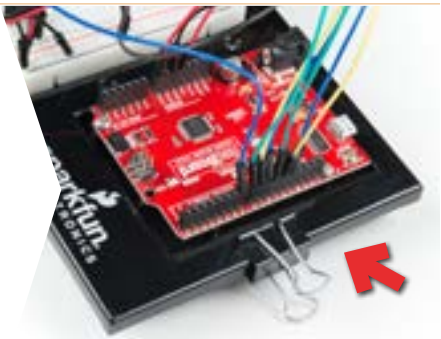
4. THE BOTTOM OF YOUR BASEPLATE should look like the image. Remember that the two motors should be mirror images of each other.



NOTE: You will likely have a piece of Dual Lock in the center of your baseplate from Project 4. It will be used in the next circuit.

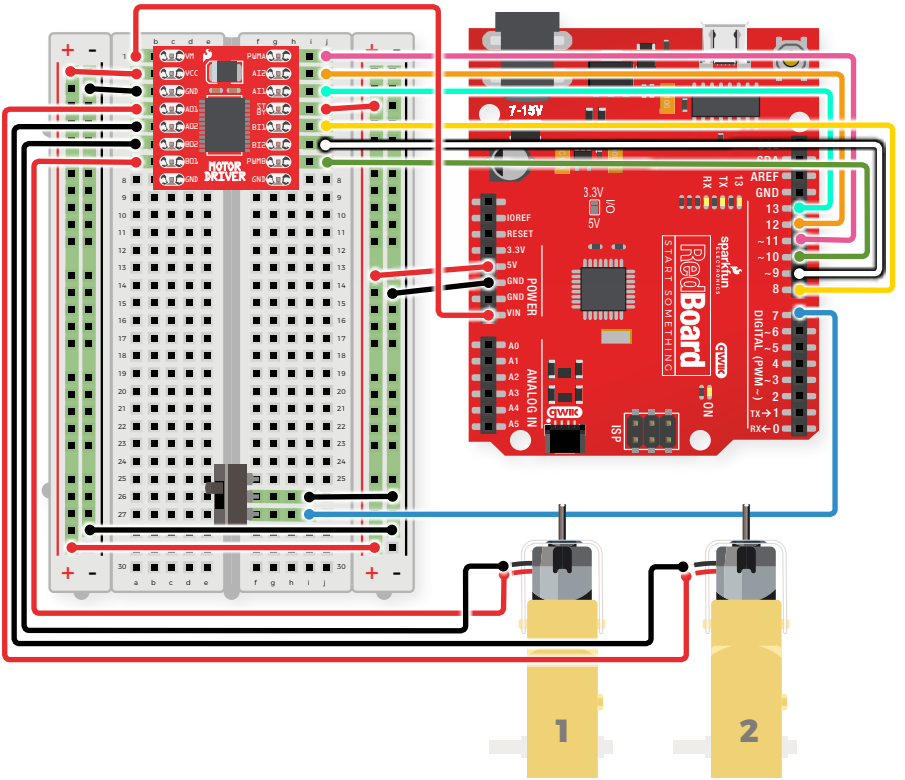
5. ATTACH THE WHEELS by sliding them onto the plastic shafts on the gearmotor. The shaft is flat on one side, as is the wheel coupler. Align the two, and then press to fit the wheel onto the shaft.

6. LAST, CLIP THE BINDER CLIP onto the back end of the robot. This will act as a caster as the robot drives around. Once you're finished, it's time to build the circuit.



HOOKUP GUIDE

READY TO START HOOKING EVERYTHING UP? Check out the circuit diagram and hookup table below to see how everything is connected.



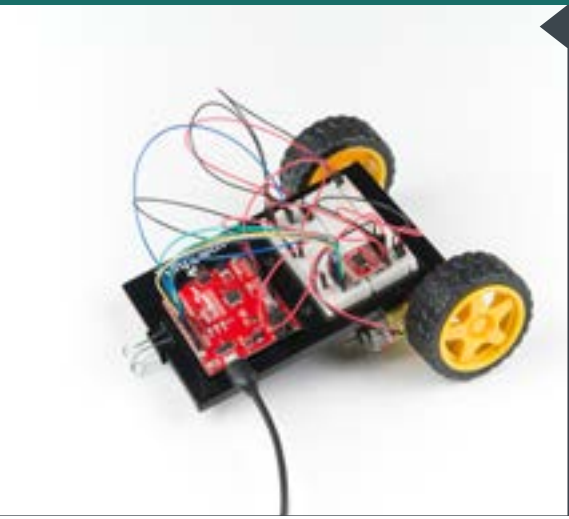
JUMPER WIRES	5V to 5V; GND to GND (-); VIN to A1; D8 to J5; D9 to J6; D10 to J7; D11 to J1; D12 to J2; D13 to J3; D7 to I27; 5V (+) to 5V (+); GND (-) to GND (-); A2 to 5V (+); A3 to GND (-); J4 to 5V (+); I26 to GND (-)
MOTOR 1 (RIGHT)	A4(RED +); A5(BLACK -)
MOTOR 2 (LEFT)	A7(RED +); A6(BLACK -)
MOTOR DRIVER	C1-C8 to G1-G8 (VM on C1, PWMA on G1)
SWITCH	F25 + F26 + F27

Open the Arduino IDE

Connect the RedBoard to a USB port on your computer.

Open the Sketch:
File > Examples > SIK-Guide-Code-master > **SIK_CIRCUIT_5B-REMOTE CONTROL ROBOT**

Select **UPLOAD** to program the sketch on the RedBoard.



WHAT YOU SHOULD SEE

Start by flipping the switch to the ON position. Open the Serial Monitor. It should prompt you to enter a command. When you type a direction into the Serial Monitor the robot will move or turn.

PROGRAM OVERVIEW

1. Prompt the user to enter a command and list the shortcuts for the directions.
2. Wait for a serial command.
3. Read the serial command and set that as the direction:
   - **A:** If the direction is “f”, drive both motors forward for the distance.
   - **B:** If the direction is “b”, drive both motors backward for the distance.
   - **C:** If the direction is “r”, drive the right motor backward and the left motor forward.
   - **D:** If the direction is “l”, drive the left motor backward and the right motor forward.

CODE TO NOTE

PARSING STRINGS: `Serial.readStringUntil(' ');`	Reads a serial message until the first space and saves it as a string.
STRING TO INT: `string_name.toInt();`	If a number is stored in a string variable, this will convert it to an integer, which can be used in math equations.

FUNCTIONS TO NOTE

`rightMotor(motor_distance);`	Drive the right motor long enough to travel the specified distance.
`leftMotor(motor_distance);`	Drive the left motor long enough to travel the specified distance.

CODING CHALLENGES

READ MORE COMMANDS: Add code to the sketch that takes not only direction but also distance. The two should be separated by a character the code can parse out and know where the next value begins.

ADD MORE COMMANDS: This sketch only uses four of the many ASCII characters. Use different keys to move the robot in different ways or have commands turn on LEDs.

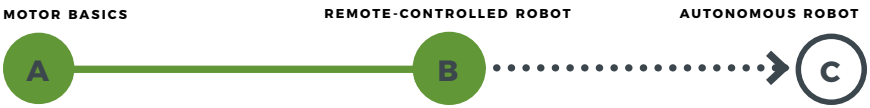
TROUBLESHOOTING

Motor not spinning	Check the wiring to the motor driver. There are a lot of connections, and it's easy to mix one of them up with another. If only one motor is working, check the wires coming from the nonworking motor. Make sure they have not come loose from the motor.
Switch not working	Make sure that you are hooked up to the middle pin and one side pin on the switch.

Still not working?	Jumper wires unfortunately can go “bad” from getting bent too much. The copper wire inside can break, leaving an open connection in your circuit. If you are certain that your circuit is wired correctly and that your code is error-free and uploaded but you are still encountering issues, try replacing one or more of the jumper wires for the component that is not working.

You’ve completed Circuit 5B!

Continue to circuit 5C to learn how to use sensors to program your robot to navigate on its own.

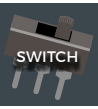


Circuit 5C: Autonomous Robot

Free the robots! In this circuit, you'll unplug your robot and program it to navigate the world on its own. When the robot senses an object using the distance sensor, it will back up and change course.

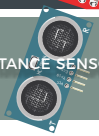
YOU NEED


MOTOR DRIVER


SWITCH


20 JUMPER WIRES


2 GEAR MOTORS


DISTANCE SENSOR

DUAL LOCK TAPE

BINDER CLIP

2 WHEELS

SCISSORS (NOT INCLUDED)

NEW CONCEPTS

AUTONOMOUS VEHICLES: The robot that you will build uses a simple sensor to avoid obstacles. This kind of system is used in Mars rovers, autonomous cars and the bots built for all kinds of robotics competitions. Understanding this example code will set you on the path to building bigger and better autonomous vehicles!

Keep in mind that the ultrasonic distance sensor needs a clear path to avoid unwanted interruptions in your robot's movements. Keep the distance sensor clear of any wires from your circuit.

ASSEMBLY

BATTERY HOLDER ATTACHMENT: If you did not attach the battery pack in Project 4, cut two pieces of Dual Lock, about 1 inch x 1 inch (2.5cm x 2.5cm) each. Remove the adhesive backing, and attach one piece to the back of the battery holder.

Adhere the second piece to the bottom of the baseplate, directly in the middle.

Press the battery holder to the baseplate so that the two pieces of Dual Lock snap together. Insert the batteries into the holder if you have not done so already. Remember that batteries are polarized and can only go in one way.

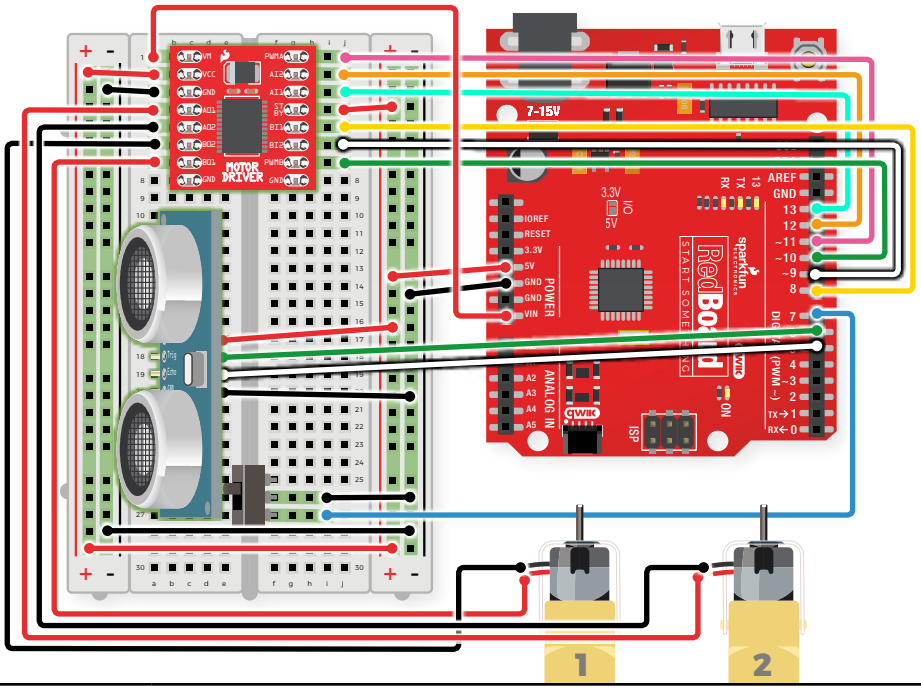
Clip the binder clip back on, and you are ready to roll!

HEADS UP!

Make sure your switch is in the **OFF** position. As soon as the code is finished uploading, your robot will begin driving. Make sure it cannot drive off a table or other high surface and injure itself.

HOOKUP GUIDE

READY TO START HOOKING EVERYTHING UP? Check out the circuit diagram and hookup table below to see how everything is connected.



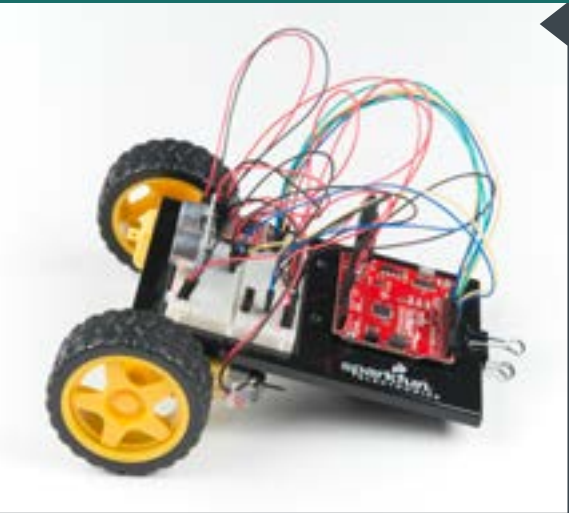
JUMPER WIRES	5V to 5V GND to GND (-) VIN to A1 D8 to J5 D9 to J6 D10 to J7 D11 to J1 D12 to J2 D6 to E18 D5 to E19 D13 to J3 D7 to I27 5V (+) to 5V (+) GND (-) to GND (-) A2 to 5V (+) A3 to GND (-) J4 to 5V (+) I26 to GND (-) E17 to 5V (+) E20 to GND (-)
MOTOR 1 (RIGHT)	A4(RED +) A5(BLACK -)
MOTOR 2 (LEFT)	A7(RED +) A6(BLACK -)
MOTOR DRIVER	C1-C8 to G1-G8 (VM on C1, PWMA on G1)
SWITCH	F25 + F26 + F27
DISTANCE SENSOR	A17(VCC) + A18(TRIG) + A19(ECHO) + A20(GND)

Open the Arduino IDE

Connect the RedBoard to a USB port on your computer.

Open the Sketch:
File > Examples > SIK-Guide-Code-master > **SIK_CIRCUIT_5C-AUTONOMOUS ROBOT**

Select **UPLOAD** to program the sketch on the RedBoard.



WHAT YOU SHOULD SEE

When the switch is turned off, the robot will sit still. When the switch is turned on, the robot will drive forward until it senses an object. When it senses an object in its path, it will reverse and then turn to avoid the obstacle.

PROGRAM OVERVIEW

1. **If** the switch is turned on,
2. **Then** start sensing the distance.
   - **A:** If no obstacle is detected, then drive forward.
   - **B:** If an obstacle is detected, stop, back up, and turn right.
   - **C:** If no obstacle is detected, start driving forward again.

TROUBLESHOOTING WARNING

HVAC systems in offices and schools have been known to interfere with the performance of the ultrasonic distance sensor. If you are experiencing sporadic behavior from your circuit, check your surroundings. If there are numerous air ducts in the room you are using, try moving to a different room that does not have ducts. The airflow from these ducts can interfere with the waves sent from the sensor, creating noise and resulting in bad readings.

CODING CHALLENGES

CHANGE THE DISTANCE AT WHICH YOUR ROBOT REACTS: Try changing the distance at which your robot stops and turns away from an obstacle.

CHANGE THE BEHAVIOR OF THE ROBOT WHEN IT SENSES AN OBSTACLE: Try changing the code so that your robot does something different when it senses an obstacle.

TROUBLESHOOTING

The robot drives backward and/or turns in the wrong direction	Check the wiring of your motors and the way that they are mounted to the baseplate. If one of your motors is flipped around, reposition it, or switch its black and red wires on the breadboard (this will reverse the direction that it turns).
The robot runs into obstacles	You can try gently bending the pins of the distance sensor so that it points straight ahead. The robot will get stuck if one wheel hits an object that it is driving past (the distance sensor won't see the obstacle unless it's in front of the robot).
The robot drives slow or not at all, though the RedBoard is powered	Try installing fresh batteries. These slow or sporadic behaviors are symptoms that your robot may be running out of power. Please note that the 4 AA batteries output about 6 or 7V, which is just below the recommended input voltage for the RedBoard. You can also use 9V batteries with a proper adapter, though their battery life won't last as long.
Still not working?	Jumper wires unfortunately can go "bad" from getting bent too much. The copper wire inside can break, leaving an open connection in your circuit. If you are certain that your circuit is wired correctly and that your code is error-free and uploaded but you are still encountering issues, try replacing one or more of the jumper wires for the component that is not working.

You've Completed All the Circuits in the SparkFun Inventor's Kit!

EXPLORE MORE WITH THE DIGITAL GUIDE

You can find a digital version of this guide online. In it are links that provide more in-depth explanations of components and concepts.

www.sparkfun.com/SIKguide

RESOURCES AND GOING FURTHER

The No. 1 question asked when one finishes the SIK is, "What's next?!" Now that you have a taste of what you can build, the sky's the limit! For additional circuits, projects and expansion ideas for your Inventor's Kit, please visit our SIK resource webpage.

www.sparkfun.com/SIKnext

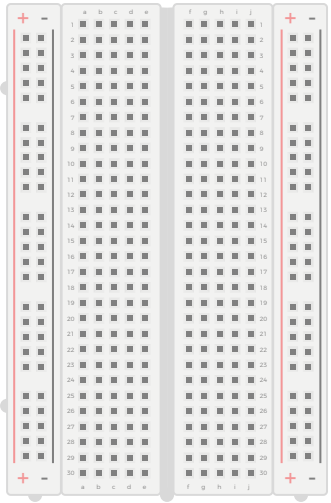
VISIT US ONLINE

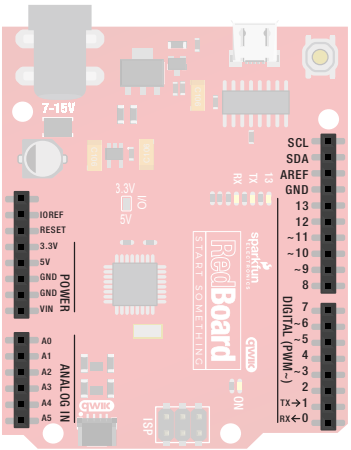
Our website has hundreds of tutorials to teach you more about embedded electronics and programming. Search for new projects to inspire your creativity, learn about new concepts and components, and discover the vast catalog of SparkFun product-specific guides to take your skills to the next level.

www.learn.sparkfun.com

NOTES

PROJECT: **DATE:**





Know Your Resistors

COMMON RESISTOR VALUES: Resistors are electronic components that have a specific, never-changing electrical resistance. The resistor's resistance limits the flow of electrons through a circuit. Included in your kit are both 330Ω resistors and 10kΩ resistors, two very common values that can be used in numerous circuits.

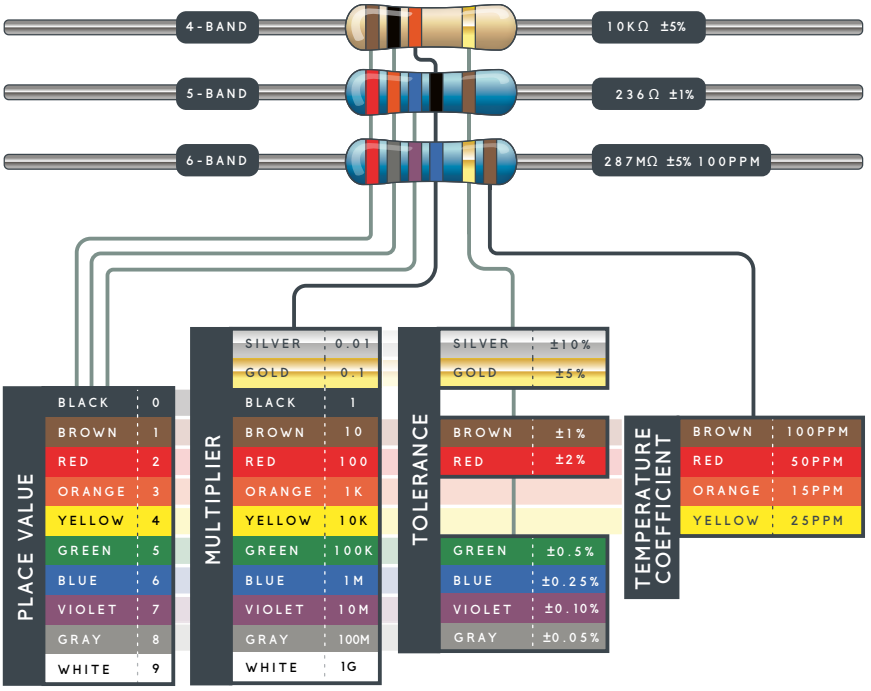
YOUR KIT INCLUDES:

10KΩ RESISTOR

330Ω RESISTOR

IDENTIFYING OTHER RESISTORS

MOST RESISTORS, not just the ones included with your SparkFun Inventor's Kit, use the same system of colored bands to identify how much resistance they can provide to a circuit. **FOR MOST BASIC RESISTORS, SIMPLY ADD TOGETHER VALUES OF THE FIRST THREE COLOR BANDS TO GET THE TOTAL RESISTANCE VALUE.**



Want to take your kit to the next level?

The SparkFun Qwiic Connect System uses cables to easily connect boards, sensors and much more to your Inventor's Kit – no soldering required! Check out some of our favorite Qwiic boards below, and learn how to easily expand your kit and build customized circuits at sparkfun.com/qwiic.

PROXIMITY SENSOR

ADD OBJECT DETECTION TO YOUR PROJECT

The SparkFun Proximity Sensor Breakout is a simple IR presence and ambient light sensor utilizing the VCNL4040.

TRIPLE AXIS ACCELEROMETER

DETECT MOTION AND ORIENTATION

This breakout board enables the tiny MMA8452Q accelerometer to communicate over I2C in your project.

GPS CHIP, SAM-M8Q

EQUIP YOUR PROJECT WITH GPS TECHNOLOGY

The SparkFun SAM-M8Q GPS Breakout is a high quality GPS board (antenna included), with equally impressive configuration options.

Even More Online

SIK RESOURCES: sparkfun.com/SIK
DIGITAL GUIDE: sparkfun.com/SIKguide
EDUCATION RESOURCES: sparkfun.com/SIKedu
TUTORIALS AND VIDEOS: learn.sparkfun.com
CUSTOMER SUPPORT: sparkfun.com/support
CALL US! 303-284-0979

sparkfun
START SOMETHING

CONTAINING MORE THAN A DOZEN COMPONENTS AND SENSORS, **THE SPARKFUN INVENTOR'S KIT** TEACHES YOU HOW TO ASSEMBLE AND USE FIVE INTERCONNECTED PROJECTS TO UNLEASH YOUR INNER INNOVATOR WITH ARDUINO! **NO PREVIOUS PROGRAMMING, SOLDERING OR ELECTRONICS EXPERIENCE IS NEEDED.**

THE SIK TEACHES BASIC PROGRAMMING, FOR WHICH YOU WILL NEED A COMPUTER WITH AN INTERNET CONNECTION.

EXPERIENCING A PROBLEM NOT COVERED BY THE TROUBLESHOOTING GUIDE?

We are constantly working to improve your SparkFun Inventor's Kit experience. Visit our SIK errata page at sparkfun.com/SIKerrata to find a solution.