sparkfun ELECTRONICS

Introduction to Arduino

// Basic Arduino reference sheet:

Installation:

- **Arduino:** http://www.arduino.cc/en/Guide/HomePage
- **Fritzing:** http://fritzing.org/download/

Support:

- **Arduino:** http://www.arduino.cc, http://www.freeduino.org, google.com
- **Fritzing:** http://www.fritzing.org/learning/

Forums:

- **Arduino:** http://forum.sparkfun.com/viewforum.php?f=32
- **Fritzing:** http://fritzing.org/forum/

Name:
Date:

// Basic Arduino code definitions:

- **setup( ):** A function present in every Arduino sketch. Run once before the loop( ) function. Often used to set pinmode to input or output. The setup( ) function looks like:

```
void setup( ){
        //code goes here
        }
```

- **loop( ):** A function present in every single Arduino sketch. This code happens over and over again. The loop( ) is where (almost) everything happens. The one exception to this is setup( ) and variable declaration. ModKit uses another type of loop called “forever( )” which executes over Serial. The loop( ) function looks like:

```
void loop( ) {
        //code goes here
        }
```

- **input:** A pin mode that intakes information.

- **output:** A pin mode that sends information.

- **HIGH:** Electrical signal present (5V for Uno). Also ON or True in boolean logic.

- **LOW:** No electrical signal present (0V). Also OFF or False in boolean logic.

- **digitalRead:** Get a HIGH or LOW reading from a pin already declared as an input.

- **digitalWrite:** Assign a HIGH or LOW value to a pin already declared as an output.

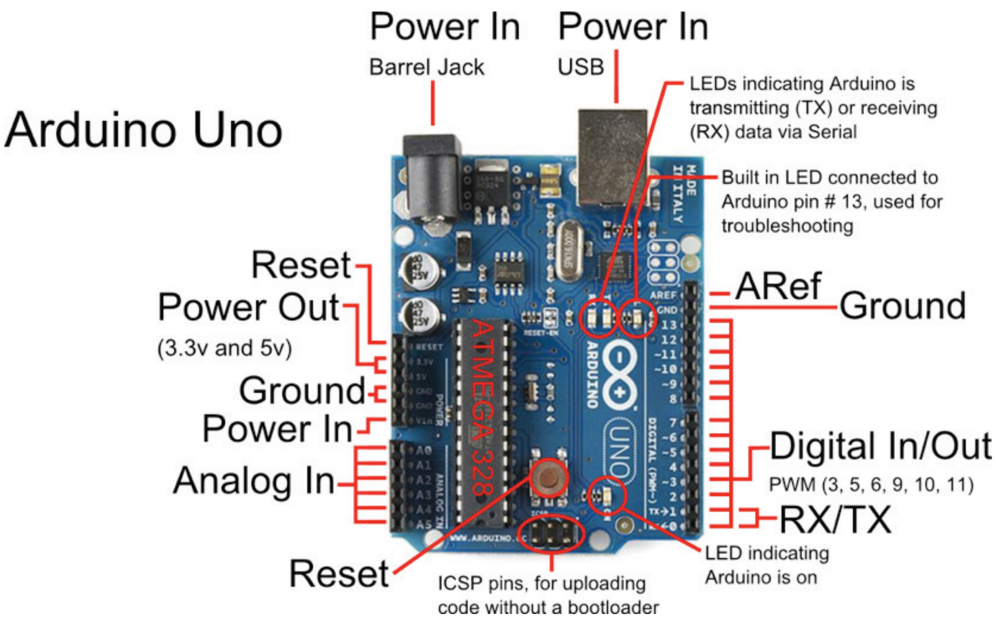
- **analogRead:** Get a value between or including 0 (LOW) and 1023 (HIGH). This allows you to get readings from analog sensors or interfaces that have more than two states.

- **analogWrite:** Assign a value between or including 0 (LOW) and 255 (HIGH). This allows you to set output to a PWM value instead of just HIGH or LOW.

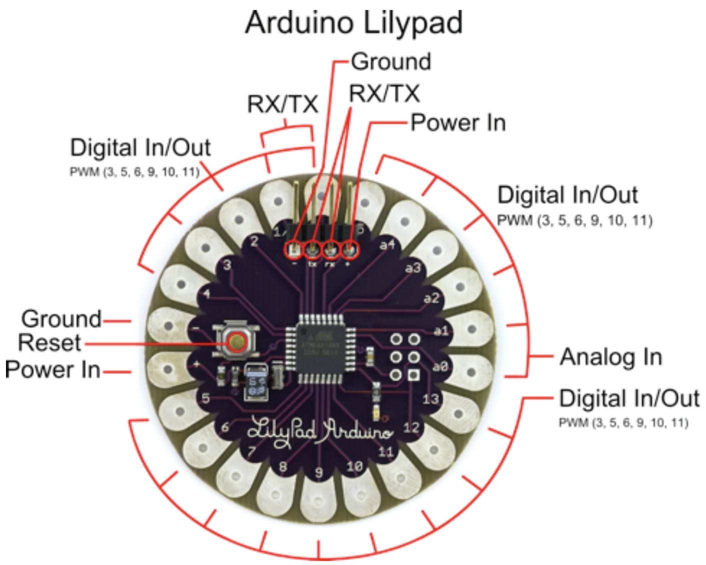
- **PWM:** Stands for Pulse-Width Modulation, a method of emulating an analog signal through a digital pin. A value between or including 0 and 255. Used with analogWrite.

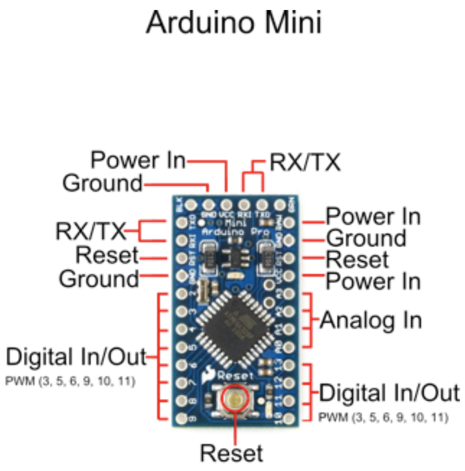
// Arduino Uno pin type definitions: (Take a look at your Arduino board)

Reset	3v3	5v	Gnd	Vin	Analog In	RX/TX	Digital	PWM(`~)	AREF
Resets Arduino sketch on board	3.3 volts in and out	5 volts in and out	Ground	Voltage in for sources over 7V (9V-12V)	Analog inputs, can also be used as Digital	Serial comm. Receive and Transmit	Input or output, HIGH or LOW	Digital pins with output option of PWM	External reference voltage used for analog


Arduino Uno
Power In
Barrel Jack
Power In
USB
LEDs indicating Arduino is transmitting (TX) or receiving (RX) data via Serial
Built in LED connected to Arduino pin # 13, used for troubleshooting
ARef
Ground
Reset
Power Out
(3.3v and 5v)
Ground
Power In
Analog In
Digital In/Out
PWM (3, 5, 6, 9, 10, 11)
RX/TX
Reset
ICSP pins, for uploading code without a bootloader
LED indicating Arduino is on

These boards below use the same microcontroller, just in a different package. The Lilypad is designed for use with conductive thread instead of wire and the Arduino Mini is simply a smaller package without the USB, Barrel Jack and Power Outs. Other boards in the Arduino family can be found at http://arduino.cc/en/Main/Hardware.





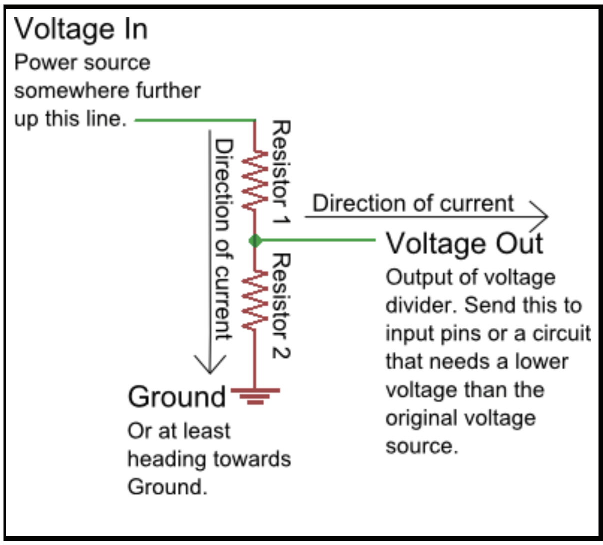
// Voltage Dividers:

What is a voltage divider?
Voltage dividers are a way to produce a voltage that is a fraction of the original voltage.

Why is a voltage divider useful?
One of the ways this is useful is when you want to take readings from a circuit that has a voltage beyond the limits of your input pins. By creating a voltage divider you can be sure that you are getting an accurate reading of a voltage from a circuit. Voltage dividers are also used to provide an analog Reference signal.

What is in a voltage divider?
A voltage divider has three parts; two resistors and a way to read voltage between the two resistors.

How do you put together a voltage divider?
It's really pretty easy. Here is a schematic and explanation detailing how:


Voltage In
Power source somewhere further up this line.
Resistor 1
Direction of current
Direction of current
Voltage Out
Output of voltage divider. Send this to input pins or a circuit that needs a lower voltage than the original voltage source.
Resistor 2
Ground
Or at least heading towards Ground.

Often resistor # 1 is a resistor with a value that changes, possibly a sensor or a potentiometer.
Resistor # 2 has whatever value is needed to create the ratio the user decides is acceptable for the voltage divider output.
The Voltage In and Ground portions are just there to establish which way the electrical current is heading, there can be any number of circuits before and after the voltage divider.
Here is the equation that represents how a voltage divider works:

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

If both resistors have the same value then Voltage Out is equal to ½ Voltage In.

Ok, how is this voltage divider information used?
It depends on what you want to do with it really. There are two different purposes outlined above for the voltage divider, we will go over both.

If you wish to use the voltage divider as a sensor reading device first you need to know the maximum voltage allowed by the analog inputs you are using to read the signal. On an Arduino this is 5V. So, already we know the maximum value we need for Vout. The Vin is simply the amount of voltage already present on the circuit before it reaches the first resistor. You should be able to find the maximum voltage your sensor outputs by looking on the Datasheet, this is the maximum amount of voltage your sensor will let through given the voltage in of your circuit. Now we have exactly one variable left, the value of the second resistor. Solve for R2 and you will have all the components of your voltage divider figured out! We solve for R1's highest value because a smaller resistor will simply give us a smaller signal which will be readable by our analog inputs.

Powering an analog Reference is exactly the same as reading a sensor except you have to calculate for the Voltage Out value you want to use as the analog Reference.

Given three of these values you can always solve for the missing value using a little algebra, making it pretty easy to put together your own voltage divider. The S.I.K. has many voltage dividers in the example circuits. These include: Circuits # 7, 8, 9, 13 and 14.

Name:
Date:

// Digital:

All of the electrical signals that the Arduino works with are either Analog or Digital. It is extremely important to understand the difference between these two types of signal and how to manipulate the information these signals represent.

Digital

An electronic signal transmitted as binary code that can be either the presence or absence of current, high and low voltages or short pulses at a particular frequency.

Humans perceive the world in analog, but robots, computers and circuits use Digital. A digital signal is a signal that has only two states. These states can vary depending on the signal, but simply defined the states are ON or OFF, never in between.

In the world of Arduino, Digital signals are used for everything with the exception of Analog Input. Depending on the voltage of the Arduino the ON or HIGH of the Digital signal will be equal to the system voltage, while the OFF or LOW signal will always equal 0V. This is a fancy way of saying that on a 5V Arduino the HIGH signals will be a little under 5V and on a 3.3V Arduino the HIGH signals will be a little under 3.3V.

To receive or send Digital signals the Arduino uses Digital pins # 0 - # 13. You may also setup your Analog In pins to act as Digital pins. To set up Analog In pins as Digital pins use the command: *pinMode(pinNumber, value);* where pinNumber is an Analog pin (A0 – A5) and value is either INPUT or OUTPUT. To setup Digital pins use the same command but reference a Digital pin for pinNumber instead of an Analog In pin. Digital pins default as input, so really you only need to set them to OUTPUT in pinMode. To read these pins use the command: *digitalRead(pinNumber);* where pinNumber is the Digital pin to which the Digital component is connected. The digitalRead command will return either a HIGH or a LOW signal. To send a Digital signal to a pin use the command: *digitalWrite(pinNumber, value);* where pinNumber is the number of the pin sending the signal and value is either HIGH or LOW.

The Arduino also has the capability to output a Digital signal that acts as an Analog signal, this signal is called Pulse Width Modulation (PWM). Digital Pins # 3, # 5, # 6, # 9, # 10 and #11 have PWM capabilities. To output a PWM signal use the command: *analogWrite(pinNumber, value);* where pinNumber is a Digital Pin with PWM capabilities and value is a number between 0 (0%) and 255 (100%). For more information on PWM see the PWM worksheets or S.I.K. circuit 12.

Examples of Digital:

Values: On/Off, Men’s room/Women’s room, pregnancy, consciousness, the list goes on....
Sensors/Interfaces: Buttons, Switches, Relays, CDs, etc....

Things to remember about Digital:

- Digital Input/Output uses the Digital pins, but Analog In pins can be used as Digital
- To receive a Digital signal use: *digitalRead(pinNumber);*
- To send a Digital signal use: *digitalWrite(pinNumber, value);*
- Digital Input and Output are always either HIGH or LOW

Name:
Date:

// Analog:

All of the electrical signals that the Arduino works with are either Analog or Digital. It is extremely important to understand the difference between these two types of signal and how to manipulate the information these signals represent.

Analog

A continuous stream of information with values between and including 0% and 100%.

Humans perceive the world in analog. Everything we see and hear is a continuous transmission of information to our senses. The temperatures we perceive are never 100% hot or 100% cold, they are constantly changing between our ranges of acceptable temperatures. This continuous stream is what defines analog data. Digital information, the complementary concept to Analog, estimates analog data using only ones and zeros.

In the world of Arduino an Analog signal is simply a signal that can be HIGH (on), LOW (off) or anything in between these two states. This means an Analog signal has a voltage value that can be anything between 0V and 5V (unless you mess with the Analog Reference pin). Analog allows you to send output or receive input about devices that run at percentages as well as on and off. The Arduino does this by sampling the voltage signal sent to these pins and comparing it to a voltage reference signal (5V). Depending on the voltage of the Analog signal when compared to the Analog Reference signal the Arduino then assigns a numerical value to the signal somewhere between 0 (0%) and 1023 (100%). The digital system of the Arduino can then use this number in calculations and sketches.

To receive Analog Input the Arduino uses Analog pins # 0 - # 5. These pins are designed for use with components that output Analog information and can be used for Analog Input. There is no setup necessary, and to read them use the command: *analogRead(pinNumber);* where pinNumber is the Analog In pin to which the the Analog component is connected. The analogRead command will return a number including or between 0 and 1023.

The Arduino also has the capability to output a digital signal that acts as an Analog signal, this signal is called Pulse Width Modulation (PWM). Digital Pins # 3, # 5, # 6, # 9, # 10 and #11 have PWM capabilities. To output a PWM signal use the command: *analogWrite(pinNumber, value);* where pinNumber is a Digital Pin with PWM capabilities and value is a number between 0 (0%) and 255 (100%). On the Arduino UNO PWM pins are signified by a ~ sign. For more information on PWM see the PWM worksheets or S.I.K. circuit 12.

Examples of Analog:

Values: Temperature, volume level, speed, time, light, tide level, spiciness, the list goes on....
Sensors: Temperature sensor, Photoresistor, Microphone, Turntable, Speedometer, etc....

Things to remember about Analog:

- Analog Input uses the Analog In pins, Analog Output uses the PWM pins
- To receive an Analog signal use: *analogRead(pinNumber);*
- To send a PWM signal use: *analogWrite(pinNumber, value);*
- Analog Input values range from 0 to 1023 (1024 values because it uses 10 bits, 210)
- PWM Output values range from 0 to 255 (256 values because it uses 8 bits, 28)

Name:
Date:

// Output:

All of the electrical signals that the Arduino works with are either input or output. It is extremely important to understand the difference between these two types of signal and how to manipulate the information these signals represent.

Output Signals

A signal exiting an electrical system, in this case a microcontroller.

Output to the Arduino pins is always Digital, however there are two different types of Digital Output; regular Digital Output and Pulse Width Modulation Output (PWM). Output is only possible with Digital pins # 0 - # 13. The Digital pins are preset as Output pins, so unless the pin was used as an Input in the same sketch, there is no reason to use the pinMode command to set the pin as an Output. Should a situation arise where it is necessary to reset a Digital pin to Output from Input use the command: *pinMode(pinNumber, OUTPUT);* where pinNumber is the Digital pin number set as Output. To send a Digital Output signal use the command: *digitalWrite(pinNumber, value);* where pinNumber is the Digital pin that is outputting the signal and value is the signal. When outputting a Digital signal value can be either HIGH (On) or LOW (Off).

Digital Pins # 3, # 5, # 6, # 9, # 10 and #11 have PWM capabilities. This means you can Output the Digital equivalent of an Analog signal using these pins. To Output a PWM signal use the command: *analogWrite(pinNumber, value);* where pinNumber is a Digital Pin with PWM capabilities and value is a number between 0 (0%) and 255 (100%). For more information on PWM see the PWM worksheets or S.I.K. circuit 12.

Output can be sent to many different devices, but it is up to the user to figure out which kind of Output signal is needed, hook up the hardware and then type the correct code to properly use these signals.

Things to remember about Output:

- Output is always Digital
- There are two kinds of Output: regular Digital or PWM (Pulse Width Modulation)
- To send an Output signal use *analogWrite(pinNumber, value);* (for analog) or *digitalWrite(pinNumber, value);* (for digital)
- Output pin mode is set using the pinMode command: *pinMode(pinNumber, OUTPUT);*
- Regular Digital Output is always either HIGH or LOW
- PWM Output varies from 0 to 255

Examples of Output:

Light Emitted Diodes (LED’s), Piezoelectric Speakers, Servo Motors

Name:
Date:

// Input:

All of the electrical signals that the Arduino works with are either input or output. It is extremely important to understand the difference between these two types of signal and how to manipulate the information these signals represent.

Input Signals

A signal entering an electrical system, in this case a microcontroller. Input to the Arduino pins can come in one of two forms; Analog Input or Digital Input.

Analog Input enters your Arduino through the Analog In pins # 0 - # 5. These signals originate from analog sensors and interface devices. These analog sensors and devices use voltage levels to communicate their information instead of a simple yes (HIGH) or no (LOW). For this reason you cannot use a digital pin as an input pin for these devices. Analog Input pins are used only for receiving Analog signals. It is only possible to read the Analog Input pins so there is no command necessary in the *setup( )* function to prepare these pins for input. To read the Analog Input pins use the command: *analogRead(pinNumber);* where pinNumber is the Analog Input pin number. This function will return an Analog Input reading between 0 and 1023. A reading of zero corresponds to 0 Volts and a reading of 1023 corresponds to 5 Volts. These voltage values are emitted by the analog sensors and interfaces. If you have an Analog Input that could exceed Vcc + .5V you may change the voltage that 1023 corresponds to by using the Aref pin. This pin sets the maximum voltage parameter your Analog Input pins can read. The Aref pin's preset value is 5V.

Digital Input can enter your Arduino through any of the Digital Pins # 0 - # 13. Digital Input signals are either HIGH (On, 5V) or LOW (Off, 0V). Because the Digital pins can be used either as input or output you will need to prepare the Arduino to use these pins as inputs in your *setup( )* function. To do this type the command: *pinMode(pinNumber, INPUT);* inside the curly brackets of the *setup( )* function where pinNumber is the Digital pin number you wish to declare as an input. You can change the pinMode in the *loop( )* function if you need to switch a pin back and forth between input and output, but it is usually set in the *setup( )* function and left untouched in the *loop( )* function. To read the Digital pins set as inputs use the command: *digitalRead(pinNumber);* where pinNumber is the Digital Input pin number.

Input can come from many different devices, but each device's signal will be either Analog or Digital, it is up to the user to figure out which kind of input is needed, hook up the hardware and then type the correct code to properly use these signals.

Things to remember about Input:

- Input is either Analog or Digital, make sure to use the correct pins depending on type.
- To take an Input reading use *analogRead(pinNumber);* (for analog)
- Or *digitalRead(pinNumber);* (for digital)
- Digital Input needs a pinMode command such as *pinMode(pinNumber, INPUT);*
- Analog Input varies from 0 to 1023
- Digital Input is always either HIGH or LOW

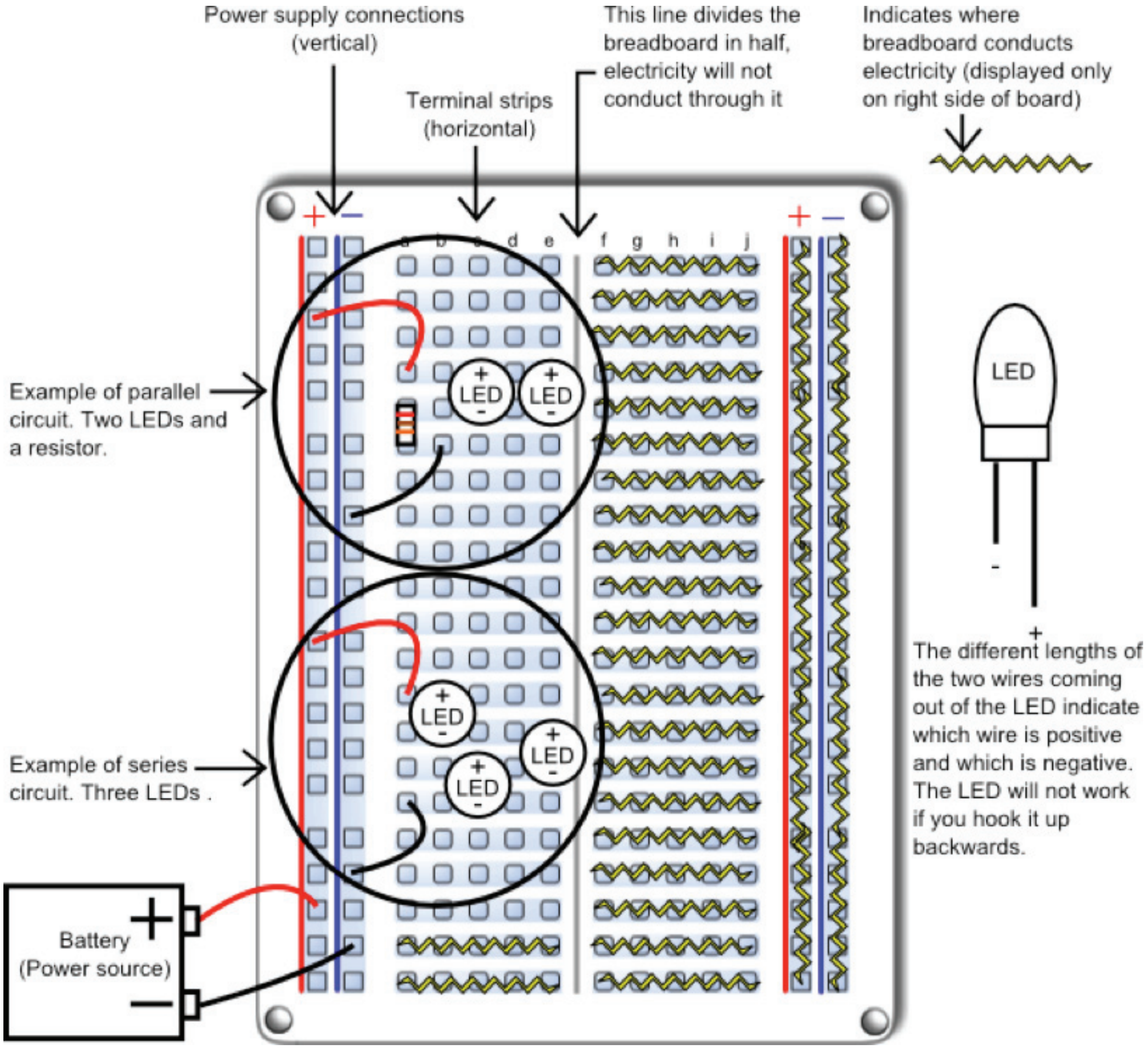
Examples of Input:

Push Buttons, Potentiometers, Photoresistors, Flex Sensors

Name:
Date:

// Breadboard:

One of the most important tools for electrical prototyping and invention is the breadboard. It's not a piece of bread that you stick electronics into, it's a piece of plastic with holes to place wires into and copper connecting the holes so electricity can get to all the pieces you are working with. But not all the holes are connected! Below is a diagram and explanation of how a breadboard works as well as examples of parallel and series circuits. Not sure what parallel and series circuits are? Don't worry! The important thing is learning how to use the breadboard so you can play around with some electronics.


Power supply connections (vertical)
Terminal strips (horizontal)
This line divides the breadboard in half, electricity will not conduct through it
Indicates where breadboard conducts electricity (displayed only on right side of board)
+ −
a b c d e
f g h i j
+ −
Example of parallel circuit. Two LEDs and a resistor.
LED
LED
Example of series circuit. Three LEDs .
LED
LED
LED
Battery (Power source)
+
−
LED
-
+
The different lengths of the two wires coming out of the LED indicate which wire is positive and which is negative. The LED will not work if you hook it up backwards.

The right side of this breadboard shows you which holes are connected and allow electricity to flow between them without anything else connecting them. This is made possible by strips of copper on the underside of the board. The power supply connections have a + and – indicating how to hook up your power source. The connections for the power supply run up and down. The terminal strips are labeled "a" through "j", these connections run across the board, but are broken down the middle. This cuts the connection across the entire terminal area in half, giving you two unconnected sections to work with.

Name:
Date:

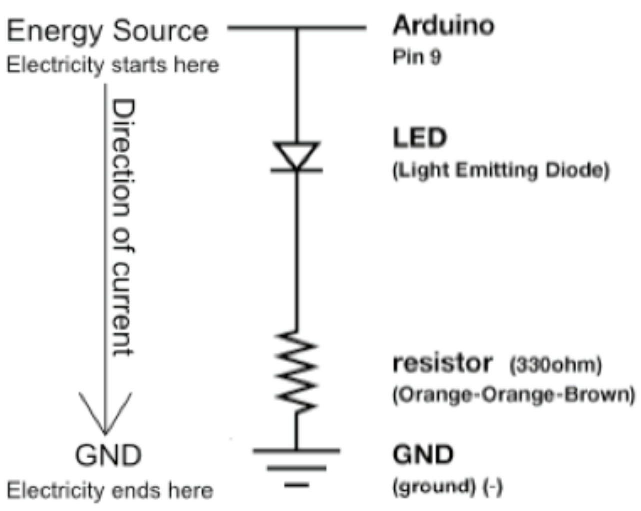
// Circuit 1, How the Circuits Work:

Circuit 1

Explanation:

This circuit takes electricity from digital Pin # 9 on the Arduino. Pin # 9 on the Arduino has Pulse Width Modulation capability allowing the user to change the brightness of the LED when using analogWrite. The LED is connected to the circuit so electricity enters through the anode (+, or longer wire) and exits through the cathode (-, or shorter wire). The resistor dissipates current so the LED does not draw current above the maximum rating and burn out. Finally the electricity reaches ground, closing the circuit and allowing electricity to flow from power source to ground.

Schematic:



Components:

Arduino Digital Pin # 9: Power source, PWM (if code uses analogWrite) or digital (if code uses digitalWrite) output from Arduino board.

LED: As in other diodes, current flows easily from the + side, or anode (longer wire), to the - side, or cathode (shorter wire), but not in the reverse direction. Also lights up!

330 Ohm Resistor: A resistor resists the current flowing through the circuit. In this circuit the resistor reduces the current so the LED does not burn out.

Gnd: Ground

Code:

```
int ledPin = 3;

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

void loop() {
   digitalWrite(ledPin, HIGH); //LED on
   delay(1000); // wait second
   digitalWrite(ledPin, LOW); //LED off
   delay(1000); // wait second
}
```

or for PWM output loop could read :
int ledPin = 3;

```
void setup() {
 pinMode(ledPin, OUTPUT);
}

 void loop() {
   analogWrite(ledPin, 255); // LED on
   delay(1000); // wait second
   analogWrite(ledPin, 0); // LED off
   delay(1000); // wait second
```

This first circuit is the simplest form of output in the kit. You can use the LED to teach both analog and digital output before moving on to more exciting outputs. There is an LED built into your Arduino board which corresponds to Digital Pin # 13.

Name:
Date:

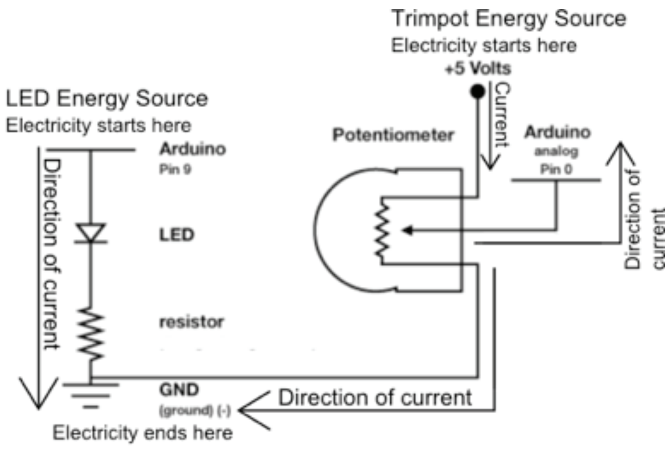
// Circuit 8, How the Circuits Work:

Circuit 8

Explanation:

This circuit is actually two different circuits. One circuit for the potentiometer and another for the LED. See How the Circuits Work, Circuit 1 for an explanation of the LED circuit. The potentiometer circuit gets electricity from the 5V on the Arduino. The electricity passes through the potentiometer and sends a signal to Analog Pin # 0 on the Arduino. The value of this signal changes depending on the setting of the dial on the potentiometer. This analog reading is then used in the code you load onto the Arduino and effects the power signal in the LED circuit. Finally the electricity reaches ground, closing the circuit and allowing electricity to flow from power source to ground.

Schematic:



Components:

Arduino Digital Pin # 13: Power source, PWM (if code uses analogWrite) or digital (if code uses digitalWrite) output from Arduino board.

Arduino Analog Pin # 0: Analog input to Arduino board.

330 Ohm Resistor: A resistor resists the current flowing through the circuit. In the LED circuit it reduces the current so the LED in the circuit does not burn out.

LED: As in other diodes, current flows easily from the + side, or anode (longer wire), to the - side, or cathode (shorter wire), but not in the reverse direction.

Potentiometer: A voltage divider which outputs an analog value.

+5V: Five Volt power source.

Gnd: Ground

Code:

```
int sensorPin = 0;
int ledPin = 13;
int sensorValue = 0;

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

void loop() {

//this line assigns whatever the //Analog Pin 0
reads to sensorValue

 sensorValue = analogRead(sensorPin);

 digitalWrite(ledPin, HIGH);
 delay(sensorValue);
 digitalWrite(ledPin, LOW);
 delay(sensorValue);
}
```

This is another example of input, only this time it is Analog. Circuits 7 and 8 in the S.I.K. introduces you to the two kinds of input your board can receive: Digital and Analog. Not sure what a voltage divider is? Check the Voltage Divider page towards the back of this section.

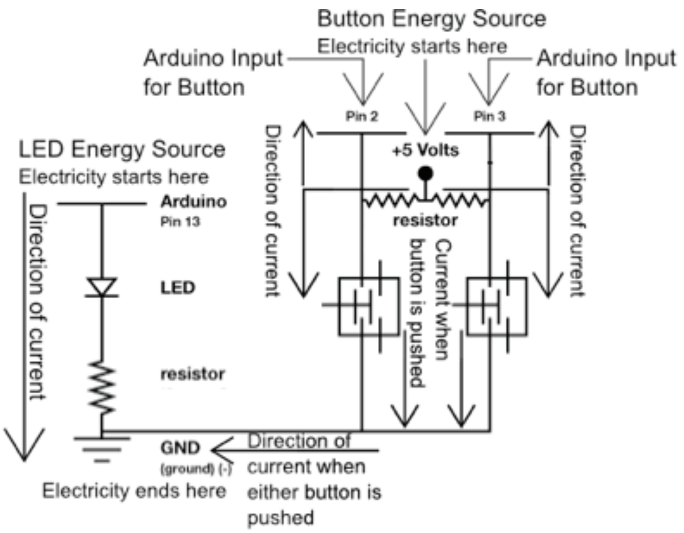
// Circuit 7, How the Circuits Work:

Circuit 7

Explanation:

This circuit is actually two different circuits. One circuit for the buttons and another for the LED. See 'How the Circuits Work', Circuit 1 for an explanation of the LED circuit. The button circuit gets electricity from the 5V on the Arduino. The electricity passes through a pull up resistor, causing the input on Arduino Pins # 2 and # 3 to read HIGH when the buttons are not being pushed. When a button is pushed it allows the current to flow to ground, causing a LOW reading on the input pin connected to it. This LOW reading is then used in the code you load onto the Arduino and effects the power signal in the LED circuit.

Schematic:



Components:

Arduino Digital Pin # 13: Power source, PWM (if code uses analogWrite) or digital (if code uses digitalWrite) output from Arduino board.

Arduino Digital Pin # 2 and # 3: Digital input to Arduino board.

330 & 10K Ohm Resistors: Resistors resist the current flowing through the circuit. In the LED circuit the 330 ohm resistor reduces the current so the LED in the circuit does not burn out. In the button circuits the 10K's ensure that the buttons will read HIGH when they are not pressed.

LED: As in other diodes, current flows easily from the + side, or anode (longer wire), to the - side, or cathode (shorter wire), but not in the reverse direction. Lights up!

Button: A press button which is open (or disconnected) when not in use and closed (or connected) when pressed. This allows you to complete a circuit when you press a button.

+5V: Five Volt power source.

Gnd: Ground

Code:

```
const int buttonPin = 2;
const int ledPin =  13;

int buttonState = 0;

void setup() {
  pinMode(ledPin, OUTPUT);
  //this line below declares the button pin as input
  pinMode(buttonPin, INPUT);
}

void loop(){
  //this line assigns whatever the Digital Pin 2 reads
to buttonState
  buttonState = digitalRead(buttonPin);

  if (buttonState == HIGH) {
    digitalWrite(ledPin, HIGH);
   }
  else {
    digitalWrite(ledPin, LOW);
   }
}
```

This circuit is the first to use the input capabilities of the Arduino. Notice the difference in *setup ( )*. You are using a Digital Pin but you are using it as input rather than output.

Name:
Date:

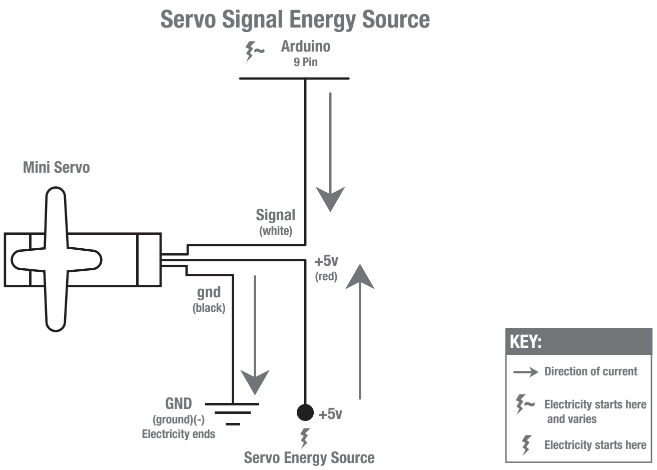
// Circuit 4, How the Circuits Work:

Circuit 4

Explanation:

The servo in this circuit takes electricity from 5V on the Arduino. Pin # 9 on the Arduino supplies a PWM signal which sets the position of the servo. Each voltage value has a distinct correlating position. Finally the electricity reaches ground, closing the circuit and allowing electricity to flow from power source to ground.

Schematic:



Components:

Arduino Digital Pin #9: Signal power source for servo.

Servo: Sets the position of the servo arm depending on the voltage of the signal received.

+5V: Five Volt power source.

Gnd: Ground

Code:

```
//include the servo library for use
#include <Servo.h>
Servo myservo;  //create servo object

int pos = 0;

void setup() {
  myservo.attach(9);
}
void loop() {
//moves servo from 0° to 180°
  for(pos = 0; pos < 180; pos += 1) {
    myservo.write(pos);
    delay(15);
    }
  // moves servo from 180° to 0°
  for(pos = 180; pos>=1; pos-=1)  {
    myservo.write(pos);
    delay(15);
    }
}
```

Name:
Date:

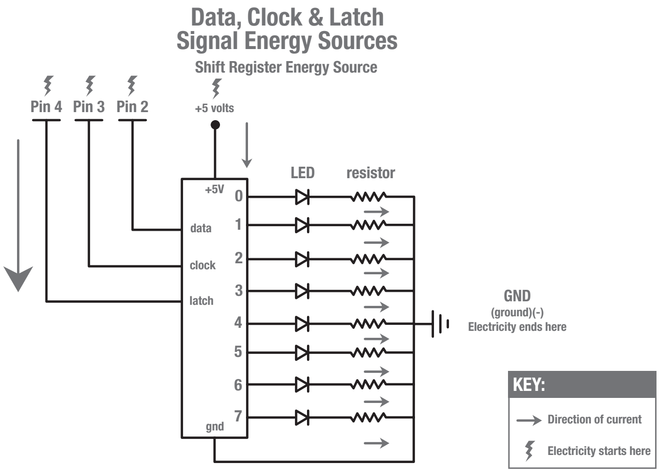
// Circuit 5, How the Circuits Work:

Circuit 5

Explanation:

The shift register in this circuit takes electricity from 5V on the Arduino. Pin # 2, # 3 and # 4 on the Arduino supply a digital value. The latch and clock pins are used to allow data into the shift register. The shift register sets the eight output pins to either HIGH or LOW depending on the values sent to it via the data pin. The LEDs are connected to the circuit so electricity enters through the anode (+, or longer wire) and exits through the cathode (-, or shorter wire) if the shift register pin is HIGH. The resistor dissipates current so the LEDs do not draw current above the maximum rating and burn out. Finally the electricity reaches ground, closing the circuit and allowing electricity to flow from power source to ground.

Schematic:



Components:

Arduino Digital Pin # 2, # 3 and # 4: Signal power source for data, clock and latch pins on shift register.

Shift register: Allows usage of eight output pins with three input pins, a power and a ground. Link to datasheet.

LED: As in other diodes, current flows easily from the + side, or anode (longer wire), to the - side, or cathode (shorter wire), but not in the reverse
direction. Lights up!

330 Ohm Resistor: A resistor resists the current flowing through the circuit. In this circuit the resistor reduces the current so the LED does not burn out.

+5V: Five Volt power source.

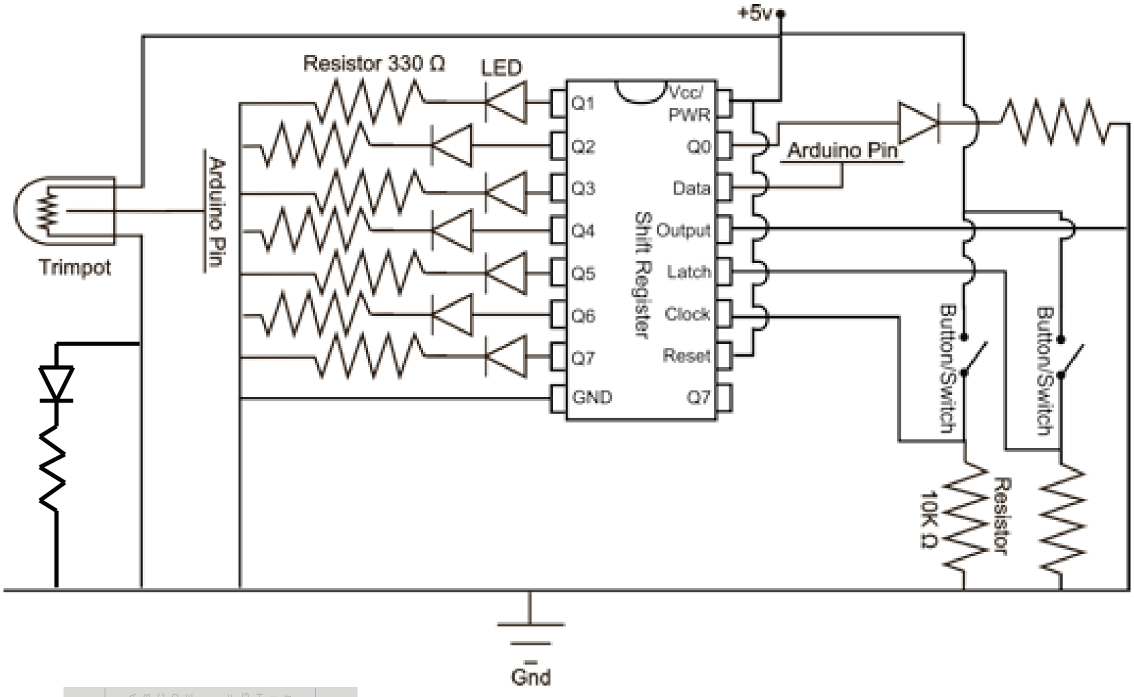
Gnd: Ground

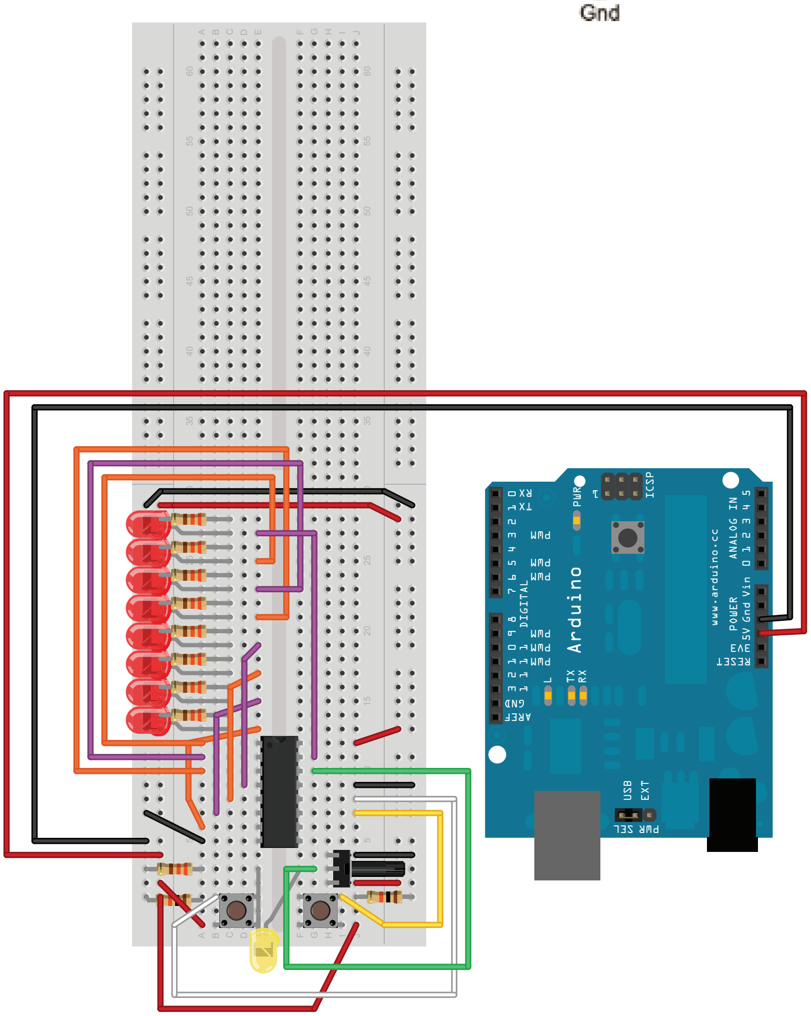
Code:

```
int data = 2;
int clock = 3;
int latch = 4;

int ledState = 0;
const int ON = HIGH;
const int OFF = LOW;

void setup() {
  pinMode(data, OUTPUT);
  pinMode(clock, OUTPUT);
  pinMode(latch, OUTPUT);
}

void loop(){
  for(int i = 0; i < 256; i++) {
    updateLEDs(i);
    delay(25);
    }
}

void updateLEDs(int value) {
  digitalWrite(latch, LOW);
  shiftOut(data, clock, MSBFIRST, value);
  digitalWrite(latch, HIGH);
}
```

Name:
Date:

// Circuit 5, How the Circuits Work:



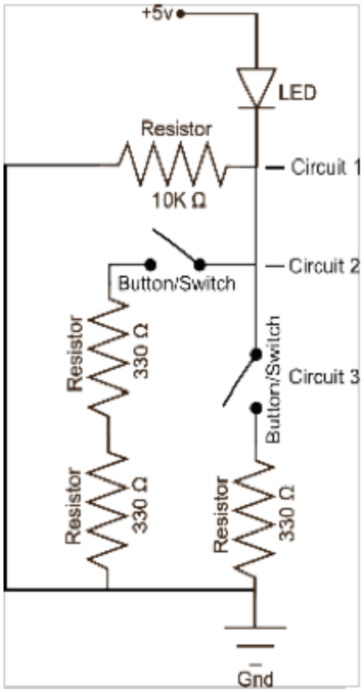


SIK Worksheets v. 1.0

Name:
Date:

// Resistance:

Resistance is an important concept when you are creating circuits. Resistance is the difficulty a current encounters when it passes through a component. Everything that electricity passes through provides some measure of resistance, wires, motors, sensors, even the human body! Measuring voltage, current and resistance are all done in different ways. To measure resistance you disconnect (turn off) your circuit and place both multimeter leads on either side of the portion of the circuit you wish to measure. For example: for measuring just a component you would place your leads on the power and ground leads of the component. To measure the resistance of multiple components you leave them connected and place the positive (red) multimeter lead closer to the disconnected power source and the negative (black) multimeter lead closer to the ground. Sometimes you will want to measure the resistance of input and output leads, but more often you will find yourself measuring resistance along the power to ground circuit. It is important to know how much resistance is present in components and circuits for many reasons. Too much resistance and the current will never travel through the whole circuit, too little and the current may fry some of your components! But most importantly you can use resistance to choose the path the current takes through your circuit.



Hook up the circuit to the right using red LEDs. (Don’t hook up the power yet.)

Measure the resistance of each of the possible paths the current can take from power (5v) to ground. There are three possible paths. You will have to measure each component separately and then add the resistance up for the total. You will can add the component’s resistance together because the components are in series, if they were parallel it would require more math. Record the total resistance for each circuit below. (Hint: you won’t be able to measure the LED)

Circuit 1: _______Ω Circuit 2: _________Ω Circuit 3: ___________Ω
Now connect the power and, one at a time, press the two buttons. Which circuit makes the LED the dimmest? Circuit # ______
If you press both buttons which path does the current take? Circuit # ______
If the voltage is staying at 5v in this circuit no matter which paths are closed, there is a way to calculate the current given the resistance. Write the name of the law and the equation that solves for resistance below. Label all variables.

____

Now measure the resistance of a potentiometer when it is dialed all the way up and down.
Record the highest and lowest values you get.

Highest:______________________ Ω

Lowest:_____________________ Ω

Redraw the schematic above (use the back of the worksheet if necessary) but use a potentiometer to control the LED brightness instead of the buttons and various resistors. Remember that you must have at least 330Ω of total resistance, otherwise you’ll burn out your LED!

Since a circuit or component does not need a current running through it in order to measure the resistance you can take your multimeter and measure the resistance of anything you can think of. Wander around and measure the resistance of various objects. Start with a penny. Record the most interesting things that have resistance and the value of their resistance below. List at least three.

____

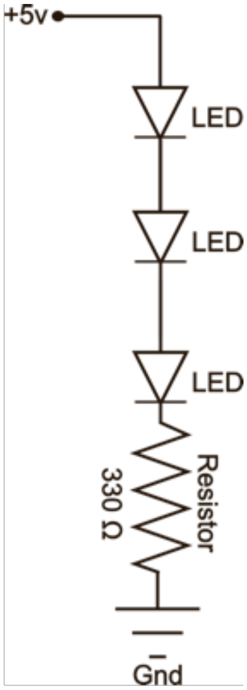
____

____

Name:
Date:

// Voltage Drop:

Voltage drop is an important concept when you are creating circuits. Voltage drop is the amount that the voltage drops when it passes through a component. The following exercises will show how to measure voltage drop in real life. This is essential when you are fixing your remote control car, electric guitar or even a cell phone. Measuring voltage, current and resistance are all done in different ways. To measure voltage you connect your positive (red) multimeter lead to the side of the circuit that closer to your power source and the negative (black) multimeter lead to the side of the circuit that is closer to the ground. It is important to know how much voltage is going through a circuit for many reasons. The most important reasons being that too much voltage can damage your components and too little voltage may not allow electricity to flow all the way through to ground.



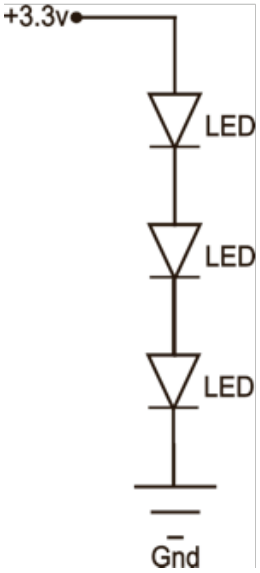
Hook up the circuit to the right using red LEDs.

Close the circuit so only one LED is grounded with the 300Ω resistor. Insert the end of the resistor not plugged into the ground into a hole on the same row as the first LED's negative lead. The other LEDs don't light up, why is this?

____

____

____



Measure the voltage drop across just the LED and record. _____________v
Measure the voltage drop across the LED and the resistor. _____________v
Close the circuit so two LEDs light up.
Voltage drop across one LED = _____v Voltage drop across two LEDs = _____v
Measure the voltage drop across the whole circuit and record. ___________v
Close the circuit so three LEDs light up.
Voltage drop across one LED = _____v Voltage drop across two LEDs = _____v
Voltage drop for three LEDs = _____v Voltage drop for whole circuit = _____v
What happened to the LEDs with the last question? ________________________
____

Now hook up the same circuit to the 3.3V power source without the resistor.
Why do you think you don't need the resistor?

____

____

Measure the voltage drop across all the LEDs and record. __________v

Close the circuit so only two LEDs light up.

Voltage drop across one LED = _____v Voltage drop across two LEDs = _____v
Hook up the circuit above to the 5V power source but use the 3.3v as ground.
Wait a second! You can't use a power source as a ground! Or can you?

What is the voltage available and how many LEDs can you light up with it?

Voltage available = ________v # of LEDs you can light up = __________
Many people think of Gnd as the ONLY place to connect a 'negative' pin, but all you need is a voltage drop from the beginning of a circuit to the end. This difference in voltage is what draws the current in the correct direction.

Name:
Date:

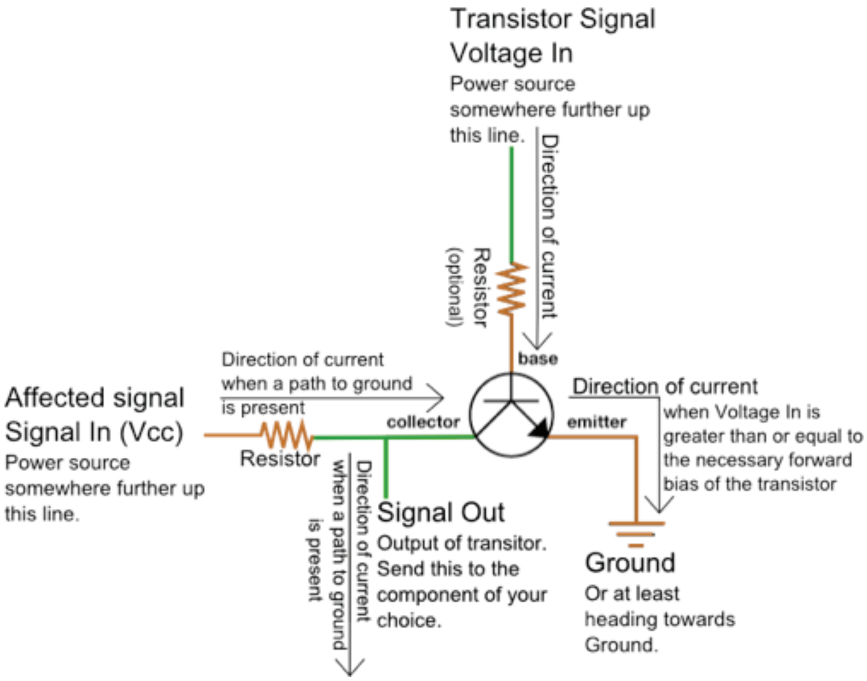
// Transistors:

What is a transistor?
Transistors are semiconductors used to amplify an electrical signal or switch an electrical signal on and off.

Why is a transistor useful?
Often you will need more power to run a component than your Arduino can provide. A transistor allows you to control the higher power signal by breaking or closing a circuit to ground. Combining this higher power allows you to amplify the electrical signal in your circuit.

What is in a transistor?
A transistor circuit has four parts; a signal power source (connects to transistor base), an affected power source (connects to transistor collector), voltage out (connects to transistor collector), and ground (connected to transistor emitter).

How do you put together a transistor?
It's really pretty easy. Here is a schematic and explanation detailing how:


Transistor Signal Voltage In
Power source somewhere further up this line.
Direction of current
Resistor (optional)
base
Direction of current when a path to ground is present
Affected signal Signal In (Vcc)
Power source somewhere further up this line.
Resistor
collector
emitter
Direction of current when Voltage In is greater than or equal to the necessary forward bias of the transistor
Direction of current when a path to ground is present
Signal Out
Output of transitor. Send this to the component of your choice.
Ground
Or at least heading towards Ground.

The transistor voltage in signal is the signal that is used to control the transistor's base.

Signal in is the power source for the signal out which is controlled by the transistor's action.

Signal out is the output of the signal originating from signal in, it is controlled by the collector.

The amount of electrical current allowed through the transistor and out of the emitter to ground is what closes the entire circuit, allowing electrical current to flow through signal out.

Ok, how is this transistor information used?
It depends on what you want to do with it really. There are two different purposes outlined above for the transistor, we will go over both.

If you wish to use the transistor as a switch the signal in and voltage in signal are connected to the same power source with a switch between them. When the switch is moved to the closed position an electrical signal is provided to the transistor base creating forward bias and allowing the electrical signal to travel from the signal in to the transistor's collector to the emitter and finally to ground. When the circuit is completed in this way the signal out is provided with an electrical current from signal in.

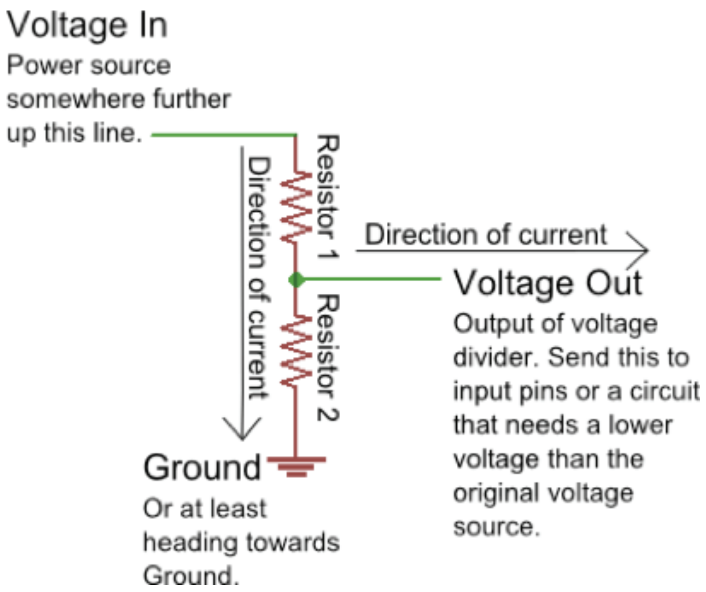
The signal amplifier use of the transistor works the same way only Signal In and Voltage In are not connected. This disconnection allows the user to send differing values to the base of the transistor. The closer the voltage in value is to the saturation voltage of the transistor the more electrical current that is allowed through the emitter to ground. By changing the amount of electrical current allowed through to ground you change the signal value of signal out. For examples of transistor uses see S.I.K. circuits # 3 and # 11.

Name:
Date:

// Voltage Dividers:

What is a voltage divider?
Voltage dividers are a way to produce a voltage that is a fraction of the original voltage.

Why is a voltage divider useful?
One of the ways a voltage divider is useful is when you want to take readings from a circuit that has a voltage beyond the limits of your input pins. By creating a voltage divider you can be sure that you are getting an accurate reading of a voltage from a circuit. Voltage dividers are also used to provide an analog reference signal.


Voltage In
Power source somewhere further up this line.
Resistor 1
Direction of current
Direction of current
Voltage Out
Output of voltage divider. Send this to input pins or a circuit that needs a lower voltage than the original voltage source.
Resistor 2
Ground
Or at least heading towards Ground.

What is in a voltage divider?
A voltage divider has three parts; 2 resistors and a way to read voltage between the 2 resistors.

How do you put together a voltage divider?
It’s really pretty easy. Here is a schematic and explanation detailing how:

Often resistor # 1 is a resistor with a value that changes, possibly a sensor or a potentiometer.

Resistor # 2 has whatever value is needed to create the ratio the user decides is acceptable for the voltage divider output.

The Voltage In and Ground portions are just there to establish which way the electrical current is heading, there can be any number of circuits before and after the voltage divider.

Here is the equation that represents how a voltage divider works:

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

If both resistors have the same value then Voltage Out is equal to ½ Voltage In.

Ok, how is this voltage divider information used?
It depends on what you want to do with it really. There are two different purposes.
If you wish to use the voltage divider as a sensor reading device first you need to know the maximum voltage allowed by the analog inputs you are using to read the signal. On an Arduino this is 5V. So, already we know the maximum value we need for Vout. The Vin is simply the amount of voltage already present on the circuit before it reaches the first resistor. You should be able to find the maximum voltage your sensor outputs by looking on the Datasheet, this is the maximum amount of voltage your sensor will let through given the voltage in of your circuit. Now we have exactly one variable left, the value of the second resistor. Solve for R2 and you will have all the components of your voltage divider figured out! We solve for R1’s highest value because a smaller resistor will simply give us a smaller signal which will be readable by our analog inputs. Powering an analog Reference is exactly the same as reading a sensor except you have to calculate for the Voltage Out value you want to use as the analog Reference. Given three of these values you can always solve for the missing value using a little algebra, making it pretty easy to put together your own voltage divider. The S.I.K. has many voltage dividers in the example circuits. These include: Circuits # 7, 8, 9, 13 and 14 to use the input capabilities of the Arduino. Notice the difference in *setup( )*. You are still using a Digital Pin but you are using it as input rather than output.

Name:

Date:

// Using a Multimeter:

Often you will have to use a multimeter for troubleshooting a circuit, testing components, materials or the occasional worksheet. This section will cover how to use a digital multimeter, specifically a SparkFun VC830L. We will discuss how to use this multimeter to measure voltage, current, resistance and continuity on the circuits in the S.I.K.

Parts of a multimeter

Display: Where values are displayed.

Knob/Setting: Used to select what is being measured and the upper limit of how much is being measured.

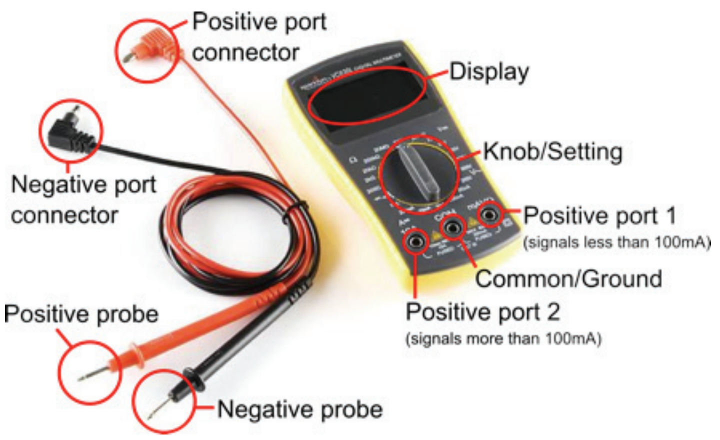
Positive port 1: Where the positive port connector is plugged in if you are measuring less than 100mA of current.

Common/Ground: Where the negative port connector is plugged in no matter what.

Positive port 2: Where the positive port connector is pluuged in if you are measuring more than 100mA of current.

Probes: The points of contact for measuring electrical signals. Place the positive probe closer to the energy source and the negative probe closer to ground.

Port Connectors: Plug them into multimeter.



Important: Sometimes the reading will not remain steady or will display a value that you believe is wrong. If this happens make sure your probes are making firm, constant contact with your circuit on a conductive material.

Settings

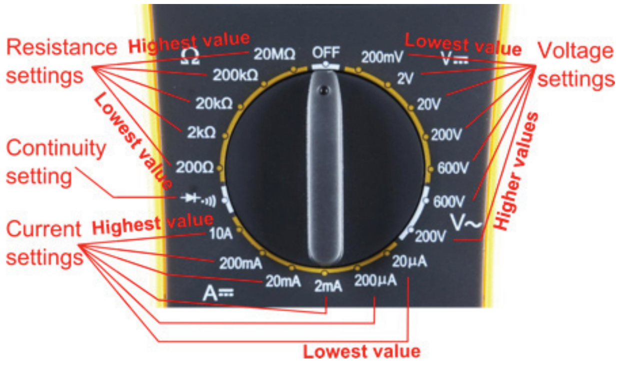
There are a bunch of different settings depending on how much of a signal the multimeter is being used to measure. This is a good opportunity to talk about unit conversion.

Voltage: The options for measuring voltage range from 200mV all the way up to 600 Volts.

Current: The options for measuring current range from 20µA all the way up to 10 Amps.

Resistance: The options for measuring resistance range from 200Ω to 20MΩ.

Continuity: This option is for testing to see if there is an electrical connection between two points.



Changing com ports:

Use the first positive com port if you are measuring a signal with less than 100mA of current. Switch to the second positive com port if you are using more. With the Arduino you will usually be using the first positive com port.

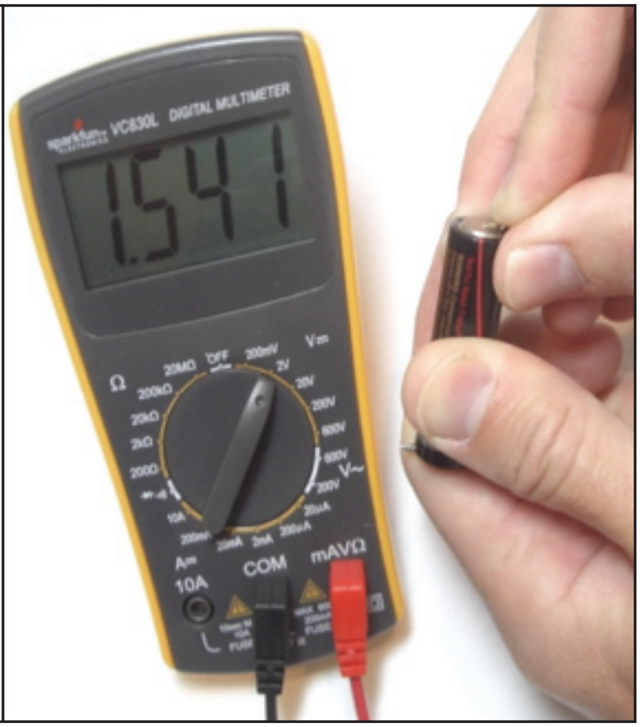
Replacing fuses:

If you try to measure more than 100mA of current through the first positive com port you will most likely blow the fuse in your multimeter. Don't worry, the multimeter isn't broken, it simply needs a new fuse. Replacing fuses is easy, this tutorial explains it: http://www.sparkfun.com/tutorials/202.

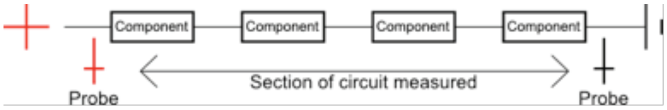
// Using a Multimeter, Voltage:

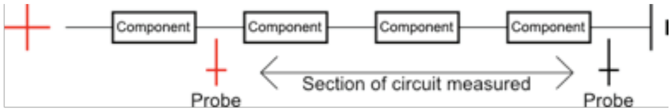
Measuring voltage

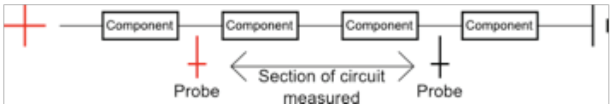
To start with something simple, let's measure voltage on a AA battery: Pull out your multimeter and plug the black probe into COM ('common') jack and the red probe into mAVΩ. Set the multimeter to "2V". Squeeze the probes with a little pressure against the positive and negative terminals of the AA battery. The black probe is customarily connected to ground or '-' and red goes to power or '+'. If you've got a fresh battery, you should see around 1.5V on the display! What happens if you switch the red and black probes? Nothing bad happens! The reading on the multimeter is simply negative - so don't worry too much about getting the red or black probe in the right place.



For most Arduino uses you will be measuring voltages that are 9V or less. Knowing this allows you to start your voltage measurement setting at 20V and work your way down. On a circuit use the multimeter to measure voltage from one point in the circuit to another point somewhere along the same circuit. The multimeter can be used to measure the voltage of the whole circuit (if it's going from 5V to GND this will usually read 4.8 to 5V) or just a portion. If you want to measure the voltage of just a portion of your circuit, you have to pay attention to where you place your probes. Find the portion of the circuit you want to measure, and place one probe on the edge of that portion nearest to the energy source. Place the other probe on the edge of that portion nearest to ground. Voila - you have found the voltage of just that section between your probes! Confused? See the schematic images below. Still confused?For more on this see voltage drop.





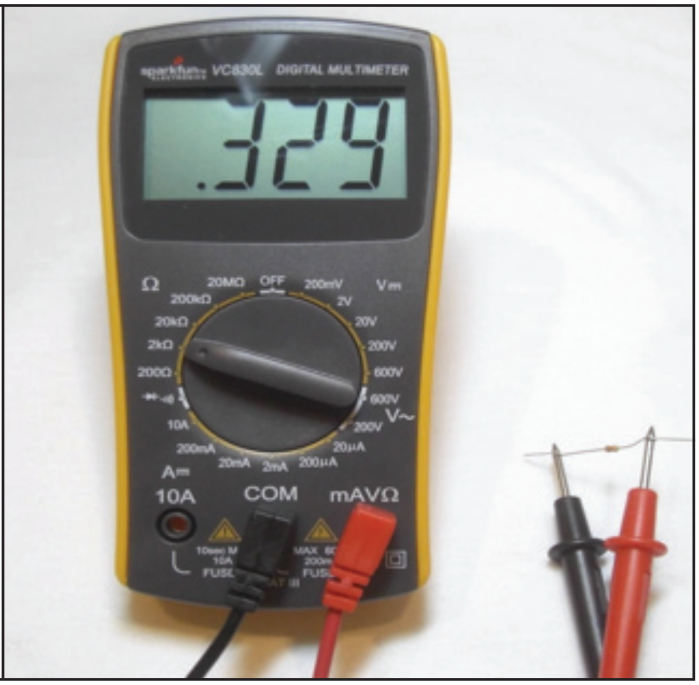


If your multimeter reads 1. the multimeter voltage setting you are using is too low. Try a larger voltage setting, if you still encounter the same problem try an even higher setting. If your multimeter reads 0 the multimeter voltage setting you are using is too high. Try a smaller voltage setting, if you still encounter the same problem try an even smaller setting.

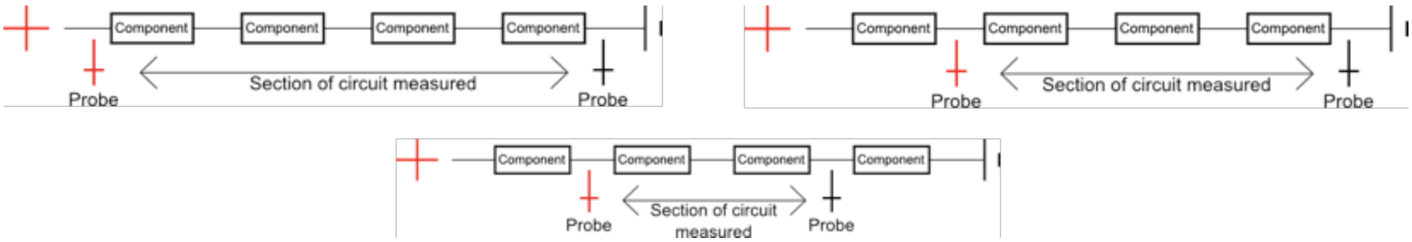
// Using a Multimeter, Resistance:

Measuring resistance

To start with something simple, let's measure the resistance of a resistor: Pull out your multimeter and plug the black probe into COM ('common') jack and the red probe into mAVΩ. Set the multimeter to "2kΩ". Squeeze the probes with a little pressure against the wires on either end of the resistor. The black probe is customarily connected to ground or '-' and red goes to power or '+'. The multimeter will measure the resistance of all the components between the two probes. It is important to remember to turn off the power of a circuit before measuring resistance. Measuring resistance is one of the few times you will use a multimeter on a circuit with no power. The example to the right is a 330Ω resistor. Notice the multimeter does not read exactly .330, often there is some margin of error.



When measuring resistance first make sure that the circuit or component(s) you are measuring do not have any electricity running through them. On a circuit use the multimeter to measure resistance from one point in the circuit to another point somewhere along the same circuit. The multimeter can be used to measure the resistance of the whole circuit or just a portion. If you want to measure the resistance of just a portion of your circuit, you have to pay attention to where you place your probes. Find the portion of the circuit you want to measure, and place one probe on the edge of that portion nearest to the energy source. Place the other probe on the edge of that portion nearest to ground. Voila - you have found the resistance of just that section between your probes! Confused? See the schematic images below. Still confused? For more on this see resistance.



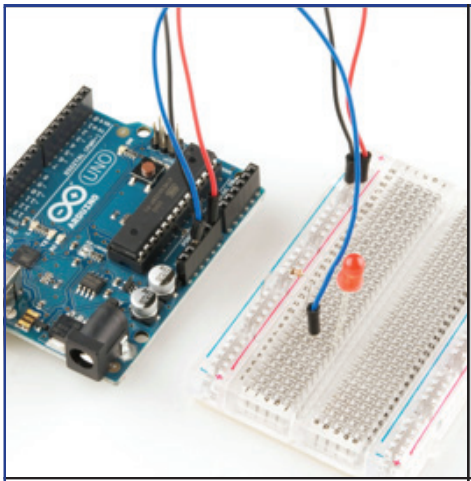
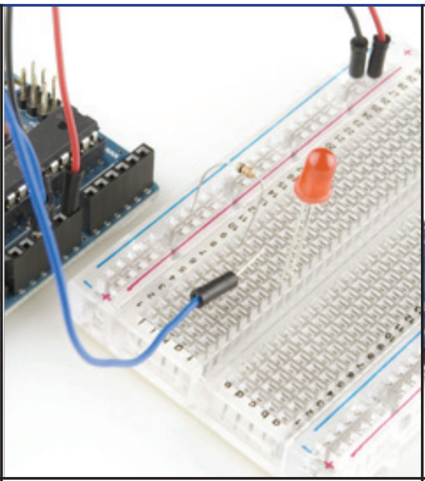
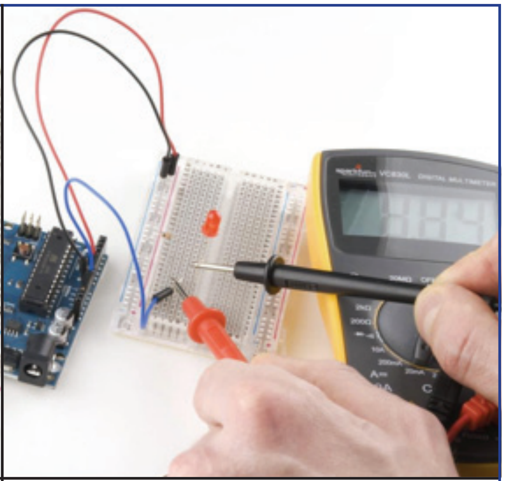
If your multimeter reads 1. the multimeter resistance setting you are using is too low. Try a larger resistance setting, if you still encounter the same problem try an even higher setting. If your multimeter reads 0 the multimeter resistance setting you are using is too high. Try a smaller resistance setting, if you still encounter the same problem try an even smaller setting. You can measure the resistance of any conductive material whether it is in a circuit or not. Depending on how conductive the material is you may need to change your resistance multimeter setting, or even use a multimeter with a larger range, but if the material is conductive you can measure the resistance of it. This is an easy way to get students to wander around getting comfortable with measuring resistance. Maybe start them off measuring the resistance of some of the S.I.K. circuits, then move to a penny and finally just set them loose to measure anything and everything.

Name:
Date:

// Using a Multimeter, Current:

Measuring current

Ok, we’re done with simple. Measuring current is a little more complicated than measuring voltage or resistance. In order to measure current you will need to “break” your circuit and insert the multimeter in series as if the multimeter and it’s two probes were a wire. The pictures below are an example of how to measure the current of the first S.I.K. circuit.

		
Unbroken circuit	Circuit broken by unplugging wire connected to power	Multimeter probes touching wire connected to power and positive lead of LED, putting multimeter in series

It doesn’t matter where in the circuit you insert your multimeter. The important thing is that the electricity has no choice but to travel through your multimeter in order to get through the rest of the circuit.

So, pull out your multimeter and plug the black probe into COM (‘common’) jack and the red probe into mAVΩ. Set the multimeter to “20mA”. Squeeze the probes with a little pressure against the two wires you used to “break” your circuit. The black probe is customarily connected closer to ground or ‘-’ and red goes closer to power or ‘+’. The multimeter will measure the total current running through the circuit.

When you are measuring current the multimeter measures the current that is present at that very instant. If your circuit or Arduino is changing the amount of current you will see that change happen instantly on your multimeter. In order to get a good reading make sure you keep the multimeter connected for at least a couple seconds. (You may also get two readings, a high and a low.)

If your multimeter reads 1. the multimeter resistance setting you are using is too low. Try a larger resistance setting, if you still encounter the same problem try an even higher setting.

If your multimeter reads 0 the multimeter resistance setting you are using is too high. Try a smaller resistance setting, if you still encounter the same problem try an even smaller setting.

// Using a Multimeter, Continuity:

Measuring continuity

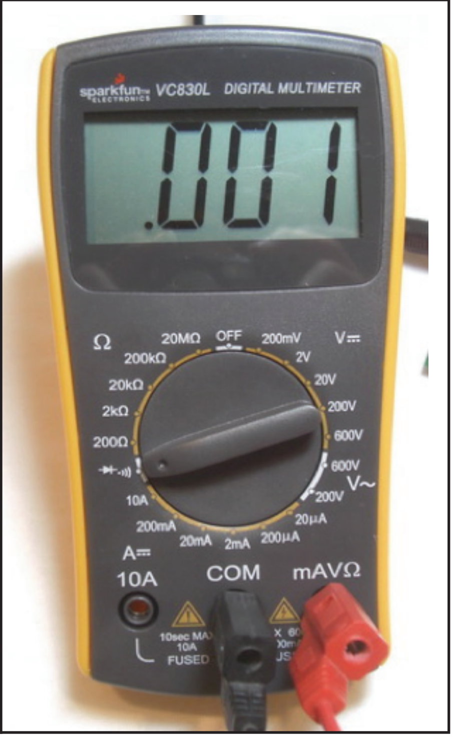
Ok, we’re done with simple. Measuring current is a little more complicated than measuring voltage or resistance. In order to measure current you will need to “break” your circuit and insert the multimeter in series as if the multimeter and it’s two probes were a wire. The pictures below are an example of how to measure the current of the first S.I.K. circuit.

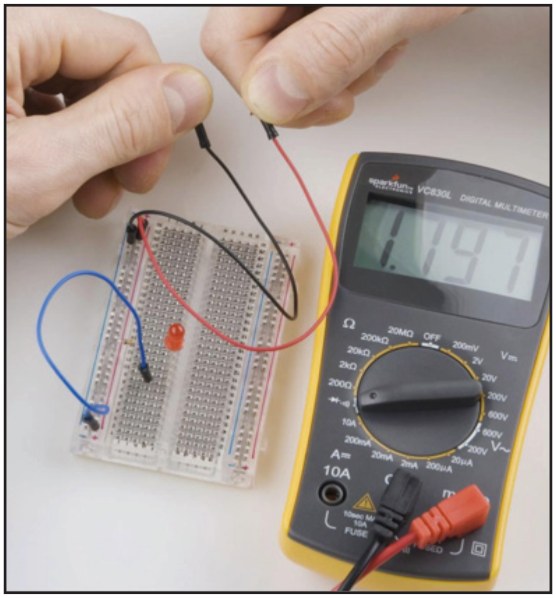
Continuity is how you check to see if two pieces of a circuit are actually connected. The multimeter does this by sending a very small current from the positive probe to the negative, when there is electricity present the multimeter beeps. This is useful when you have a circuit that you think should work but doesn’t. Make sure to turn power off when checking continuity.

Set the multimeter to the continuity setting as shown to the right. Touch your probes together and you should hear a beep. This means that electricity is free to travel between the two probes without too much resistance.

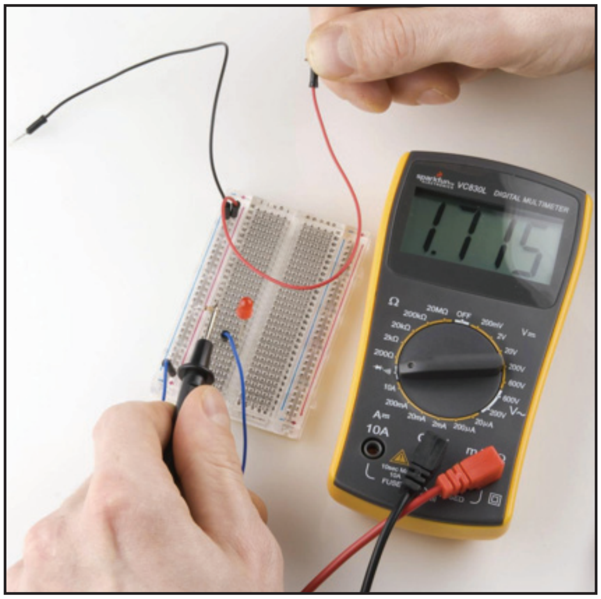
If your circuit is plugged in incorrectly, or if it is broken somewhere (maybe your breadboard or a wire is broken) when you touch the probes to the wire providing power and the wire connected to ground the multimeter will not beep. If it were hooked up correctly you would hear a beep and you wouldn’t be using the continuity setting!

In order to figure out where the circuit is broken move one of the probes along the circuit towards the other probe. Do this component by component. When the multimeter beeps you know that the probes now detect electricity passing between them so the break must be between where the probe you are moving is now, and where it was the last time the multimeter didn’t beep.





Measuring continuity of the whole circuit from power to ground. Probes are touching wires normally plugged into power and ground.



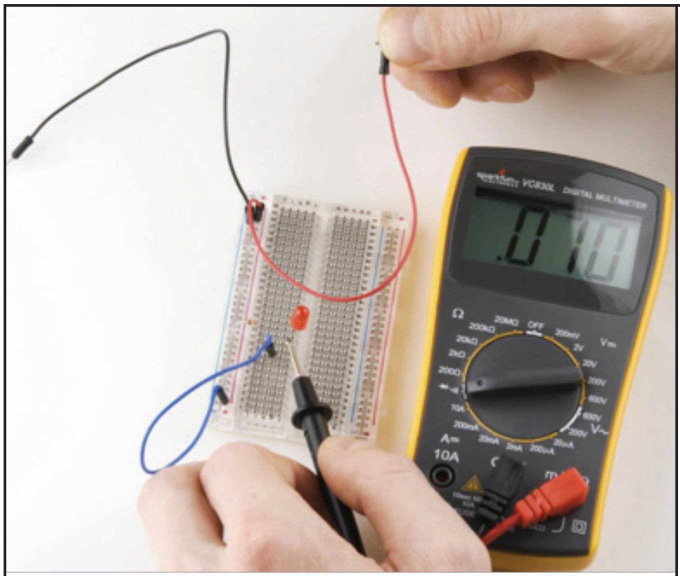
Measuring continuity of the circuit excluding wire connected to ground and resistor. Measured continuity includes LED and two wires connected to breadboard power rail and power. Probes are touching resistor wire and power.

Name:
Date:

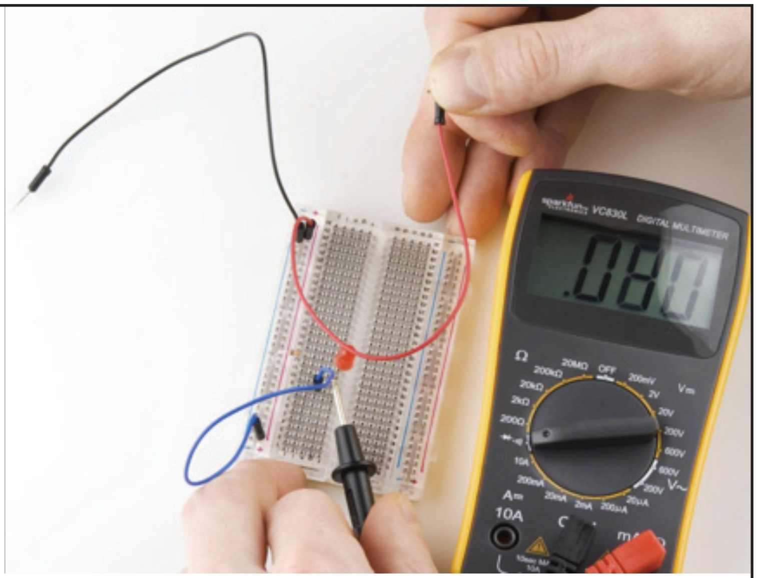
// Using a Multimeter, Continuity:

Measuring Continuity II

Ok, we're done with simple. Measuring current is a little more complicated than measuring voltage or resistance. In order to measure current you will need to "break" your circuit and insert the multimeter in series as if the multimeter and it's two probes were a wire. The pictures below are an example of how to measure the current of the first S.I.K. circuit.

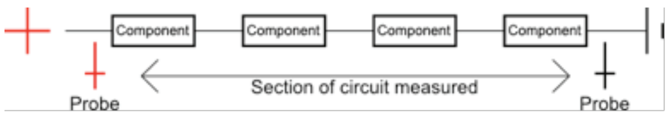


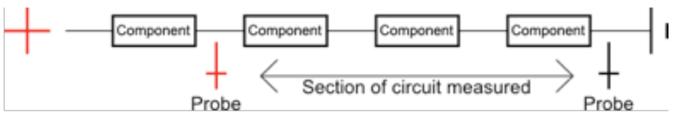
Measuring continuity of the circuit excluding wire connected to ground, resistor and LED. Measured continuity includes two wires connected to breadboard power rail and power. Probes are touching negative LED wire and power.

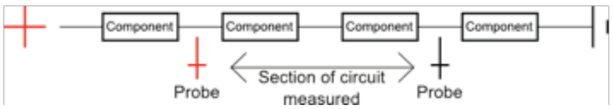


Measuring continuity of the circuit excluding wire connected to ground, resistor and LED. Measured continuity includes two wires connected to breadboard power rail and power. This image looks similar to the image on the right, but the black probe is touching the blue wire, not the negative LED wire. Probes are touching blue wire and power.

The multimeter can be used to measure the continuity of the whole circuit or just a portion. If you want to measure the continuity of just a portion of your circuit, you have to pay attention to where you place your probes. Find the portion of the circuit you want to measure, and place one probe on the edge of that portion nearest to the energy source. Place the other probe on the edge of that portion nearest to ground. Voila - you are testing the continuity of just that section between your probes! Confused? See the schematic images below.







Continuity is one of the most useful settings on a multimeter and you will most likely use it constantly simply to check for connections that aren't quite connected. Breadboards sometimes break so if your multimeter tells you there is no continuity but you know everything is plugged in correctly try switching breadboards.

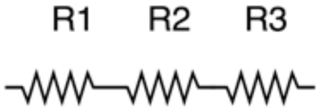
The beep of the multimeter only tells you that there is very little resistance between the two probes.
If there is a resistor in the circuit you will not hear a beep but the display will show a number indicating there is continuity between the two probes.

// Series and Parallel Circuits:

Series and Parallel

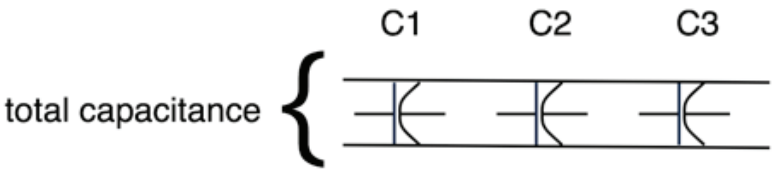
One of the most important concepts in circuit building is the difference between components in **series** and components in **parallel**. Basically you can think of components in series as being one after another, like in a chain, while parallel components as hooked up next to each other. It's important to know how certain components affect your circuit when hooked up in these two ways. There are a few things to remember, mostly that resistors and inductors work in the opposite way from capacitors:

Resistors and Inductors in **series** can simply be added together:



R1 + R2 + R3 = total resistance

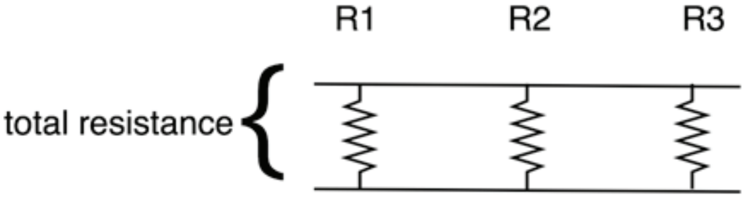
As can capacitors that are in **parallel**:



C1 + C2 + C3 = total capacitance

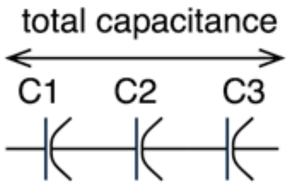
However, for resistors and inductors in parallel, as well as capacitors in series, the equation is a bit more complex. Basically the values between any two elements in these setups equal the product of the values divided by the sum of the values. For three elements or more, solve for two and repeat until done.
For example (the // indicates that the elements are in parallel):

Resistors:



R1 // R2 = (R1 * R2) / (R1 + R2)
Total Resistance = (R1// R2) * R3 / (R1//R2) + R3

Capacitors:



C1 // C2 = (C1 * C2) / (C1 + C2)
Total Capacitance = (C1// C2) * C3 / (C1//C2) + C3

It seems quite dry, but you never know when basic knowledge like this will come in handy. (Hint: read the next section on powering your projects).

Name:
Date:

// Powering Your Project and Battery Ratings:

Powering Your Projects

When dealing with electronics, it is always a good idea to know how much power you need and how you're going to get it. If you want your project to be portable, or run separately from a computer, you'll need an alternate power source. Plus, not all Arduino projects can be run off 5V from the USB port. Fortunately there are a lot of options, one or more of which should suit your purposes perfectly.

Understanding Battery Ratings

One popular way to get power to your project is through batteries. There are tons of different kinds of batteries (AA, AAA, C, D, Coin Cell, Lithium Polymer, etc.). In fact, there are too many to go over here, however, they all have a few things in common which can help you choose which ones to use. Each battery has a positive (+) and negative terminal (-) that you can think of as your power and ground. Batteries also have ratings in volts and milliamp hours (written mAh). Given this info along with how much current your circuit will draw, you can figure out how long a battery will last. For example, if I have a battery rated at 1.2v for 2500 mAh, and my circuit requires 100mA (milliamps) current, my battery will last around 17.5 hours. Wait, what? Why not 25 hours you say? Well, you shouldn't drain your battery completely, and other factors such as temperature and humidity can affect battery life, so typically the equation for determining battery life is:

(Capacity rating of battery (in mAh) ÷ Current Consumption of Circuit) x 0.7

Note that we could still use our 2500 mAh battery in a 500mA circuit, but then our battery life would only be 3.5 hours. Make sense? There's a lot to understand about powering circuits, so don't worry if it's not all clicking. Just take an educated guess, be safe, use your multimeter, and make adjustments.

It is also worth mentioning that batteries are not the only potential source of power for your project. If your project will be outside or near a window, consider using solar power. There's plenty of good documentation online, but basically, solar cells have the same kinds of voltage and current ratings that any power source might have; the only difference is that the percentage you get from your solar panel depends on how much sunlight it's getting. Check out http://www.solarbotics.com/ for some good products and documentation using solar power.

// Powering Your Project and Battery Ratings:

Powering Your Projects

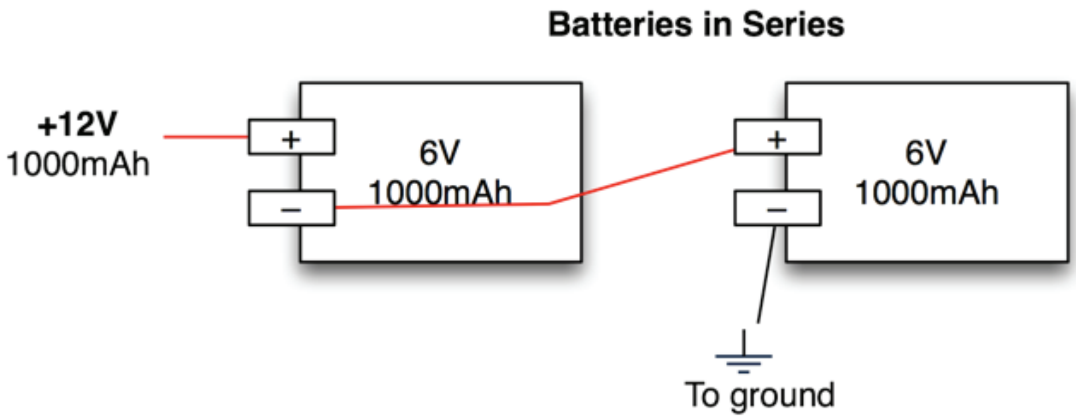
So, what if your circuit needs 12v, and all you have are a bunch of 1.5v batteries? Or what if you need your project to be powered for longer, but you don’t want to give it too much power? This is where your knowledge of series and parallel may actually come in handy.

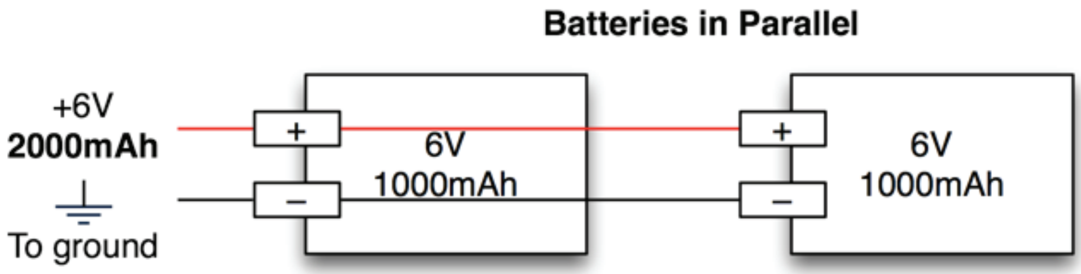
Here’s the rule:

Connecting batteries in **series** increases the voltage but maintains the capacity (mAh) - this what you want to do if you need more power.

Connecting batteries in **parallel** maintains the voltage but increases the capacity. This is what you want to do if you need your power supply to last longer.

Here’s how to hook them up:





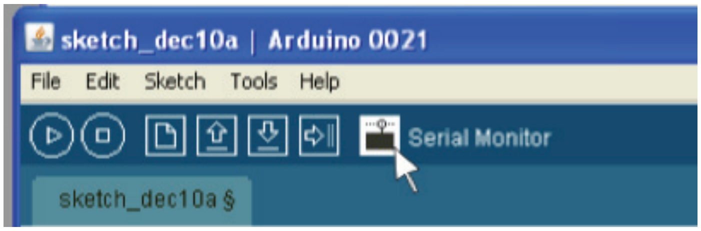
As always, use caution. Batteries of the same kind (same voltage and capacitance) work best in these kinds of situations. Using different kinds of batteries may also work but it is not recommended, as the results are not as predictable.

// Serial Basics:

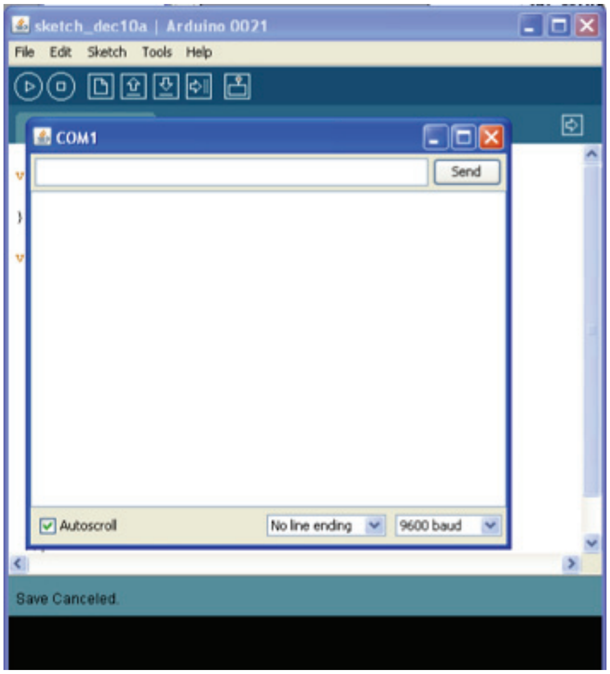
Serial is used to communicate between your computer and the Arduino as well as between Arduino boards and other devices. Serial uses a serial port (makes sense huh?) also known as UART, which stands for universal asynchronous receiver/transmitter to transmit and receive information. In this case the computer outputs Serial Communication via USB while the Arduino receives and transmits Serial using, you guessed it, the RX and TX pins. You use serial communication every time you upload code to your Arduino board. You will also use it to debug code and troubleshoot circuits. Basic serial communication is outlined in the following pages along with a simple activity to help you understand the concepts.

Serial Monitor: This is where you monitor your serial communication and set baud rate.

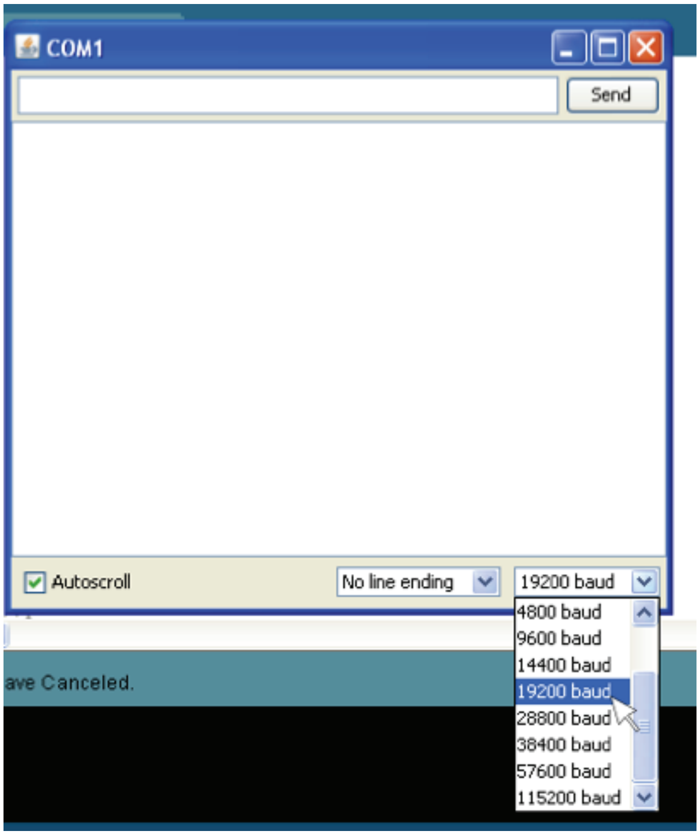
Activating the Serial Monitor:



When the activated Serial Monitor looks like:



Setting the Serial Monitor baud rate:



There are many different baud rates, (9600 is the standard for Arduino) the higher the baud rate the faster the machines are communicating.

In the examples above there is no Serial communication taking place yet. When you are running code that uses Serial any messages or information you tell Serial to display will show up in the window that opens when you activate the monitor.

Things to remember about Serial from this page:

1. Serial is used to communicate, debug and troubleshoot.
2. Serial baud rate is the rate at which the machines communicate.

Name:
Date:

// Serial Basics:

Serial setup:

The first thing you need to know to use Serial with your Arduino code is Serial setup. To setup Serial you simply type the following line inside your setup( ) function: *Serial.begin (9600);.*
This line establishes that you are using the Digital Pins # 0 and # 1 on the Arduino for Serial communication. This means that you will not be able to use these pins as Input or Output because you are dedicating them to Serial communication. The number 9600 is the baud rate, this is the rate at which the computer and the Arduino communicate. You can change the baud rate depending on your needs but you need to make sure that the baud rate in your Serial setup and the baud rate on your Serial Monitor are the same. If your baud rates do not match up the Serial Monitor will display what appears to be gibberish, but is actually the correct communication incorrectly translated.

Using Serial for code debugging and circuit troubleshooting:

Once Serial is configured using the basic communication for debugging and troubleshooting is pretty easy. Anywhere in your sketch you wish the Arduino board to send a message type the line *Serial.println("communication here");*. This command will print whatever you type inside the quotation marks to the Serial Monitor followed by a return so that the next communication will print to the next line. If you wish to print something without the return use *Serial.print("communication here");*. To display the value of a variable using println simple remove the quotation marks and type the variable name inside the parenthesis. For example, type *Serial.println(i);* to display the value of the variable named i. This is useful in many different ways, if, for example, you wish to print some text followed by a variable or you want to display multiple variables before starting a new line in the Serial Monitor.

These lines are useful if you are trying to figure out what exactly your Arduino code is doing. Place a *println* command anywhere in the code, if the text in the *println* command shows up in your Serial Monitor you will know exactly when the Arduino reached that portion of code, if the text does not show up in the Serial Monitor you know that portion of code never executed and you need to rewrite.

To use Serial to troubleshoot a circuit use the *println* command just after reading an input or changing an output. This way you can print the value of a pin signal. For example, type *Serial.print("Analog pin 0 reads:");* and *Serial.println(analogRead(A0));* to display the signal on Analog Input Pin # 0. Replace the second portion with *Serial.println(digitalRead(10));* to display the signal on Digital Pin # 10.

Things to remember about Serial from this page:

1. If Serial is displaying gibberish check the baud rates.
2. Use *Serial.print("communication here");* to display text.
3. Use *Serial.println("communication here");* to display text and start a new line.
4. Use *Serial.print(variableName);* to display the value stored in variableName.
5. Use *Serial.print(digitalRead(10));* to display the state of Digital Pin # 10.

// Serial Basics:

Using Serial for communication:

This is definitely beyond the scope of the S.I.K. but here are some basics for using Serial for device to device communication (other than your computer), not just debugging or troubleshooting. (The following paragraphs assume that you have Serial Communication hardware properly connected and powered on two different devices.)

First set up Serial as outlined on the previous page.

Use *Serial.println("Outgoing communication here");* to send information out on the transmit line.

When receiving communication the Serial commands get a little more complicated. First you need to tell the Arduino to listen for incoming communication. To do this you use the command *Serial.available();*, this command tells the computer how many bytes have been sent to the receive pin and are available for reading. The Serial receive buffer (computer speak for a temporary information storage space) can hold up to 128 bytes of information.

Once the Arduino knows that there is information available in the Serial receive buffer you can assign that information to a variable and then use the value of that variable to execute code. For example to assign the information in the Serial receive buffer to the variable *incomingByte* type the line; *incomingByte = Serial.read();* *Serial.read()* will only read the first available byte in the Serial receive buffer, so either use one byte communications or study up on parsing and string variable types. Below is an example of code that might be used to receive Serial communication at a baud rate of 9600.

```
//declare the variable incomingByte and assign it the value 0.
int incomingByte = 0;

void setup ( ) {
//establish serial communication at a baud rate of 9600
        Serial.begin(9600);
}
void loop ( ) {
//if there is information in the Serial receive buffer
        if (Serial.available() > 0){
//assign the first byte in buffer to incomingByte
                incomingByte = Serial.read();
        }
        if (incomingByte == 'A'){      //if incomingByte is A
                //execute code inside these brackets if incomingByte is A
        }
        if (incomingByte == 'B'){      //if incomingByte is B
                //execute code inside these brackets if incomingByte is B
        }
}
```

Additional things to note about Serial:

You cannot transmit and receive at the same time using Serial, you must do one or the other. You cannot hook more than two devices up to the same Serial line. In order to communicate between more than two devices you will need to use an Arduino library such as NewSoftSerial.

Things to remember about Serial from this page:

1. Serial communication requires knowing some code, but you can just look it up!
2. You cannot transmit and receive at the same time or hook up more than two devices.

// Fritzing Version 0.4.3, What is it?:

Prototype to Product

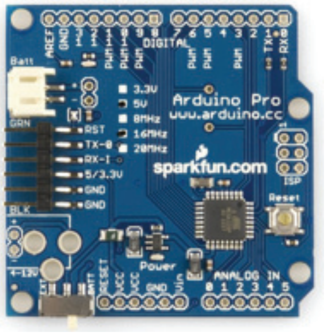
Ok, maybe you've created some sweet circuits using the S.I.K. and now you're wondering what is next. How do you turn your mass of components, wires and breadboard into something you can solder components onto and put into a tidy little package? Or in some cases a gigantic, take over the world, robot. If this describes you then you are looking to step up to the world of of virtual prototyping and printed circuit boards. Uh... what a second, what exactly does that last sentence mean?

Virtual Prototyping

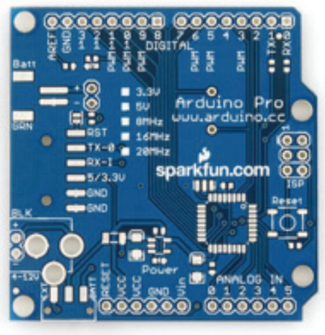
You know all those circuits you have been putting together using your breadboard? Virtual prototyping is the process of recreating those electrical circuits in a computer application so you can turn them into a finished product. There are many different applications you can use for virtual prototyping, they range from fairly simple to very complicated. All of these applications create things called "Gerber files" which are the plans, or layouts, that an inventor can send to a printed circuit board manufacturer so the manufacturer can create the actual printed circuit board. This section explains how to use an introductory application called Fritzing.

Printed Circuit Boards (PCBs)

Printed circuit boards are the boards inside of most electrical items. You have probably already seen them (they are everywhere). Printed circuit boards are also referred to as PCBs. PCBs are the second to last step of creating your invention. They are basically boards with electrical paths inside and on top for hooking up all the necessary components.



Populated Arduino Pro PCB (Parts on it)



Unpopulated Arduino Pro PCB (Parts not on it)

Fritzing

Fritzing is a free, open source application with an established online community. Fritzing can be used to create single sided PCB layouts which you can send to PCB manufacturers for mass production. This is a big step so it is important to double and triple-check everything about your design before, during and after this process. You don't want to spend money ordering a bunch of PCBs only to find out that you overlooked a detail and your final product won't work like your prototype.

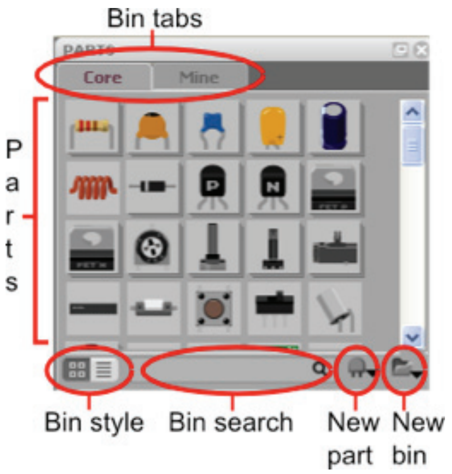
In order to properly use Fritzing you need to understand the three different views. The breadboard view is for recreating your own physical breadboard and components, the schematic view simplifies the connections and components for easier viewing and the PCB layout view allows you to place the actual leads and pins for connecting components.

In order to help you create PCBs Fritzing outputs Gerber files and an etchable .PDF file. There are eight different types of Gerber files but Fritzing creates six because it only uses one side of the PCB. Gerber files and exporting are addressed at the end of this section.

// Fritzing, Menus:

Fritzing Parts Menu

The Parts menu is where you will find all the components you will need to create your virtual prototype. The standards components, wires, PCB parts and even a ruler are already there, ready for use under the "Core" tab. The rest of the buttons in this menu are outlined below.



• **Bin tabs:** Bins are lists of parts that are available for use in Fritzing, for your convenience you can create your own bins to hold only the parts you need for a particular project. If you wish to create your own "bin" of parts there is already another tab labeled "Mine" which you can use. Simply click on the tab labeled "Mine" to see the parts in this bin, drag and drop from the core menu to add parts or use the new part sub menu. Right click on a part to activate the part editor window for modifying it for your particular purpose.

• **Parts:** This is where your parts are listed. The images give you an idea of what each part is. Place your cursor on a part to see the name and properties in the Inspector Menu. Right click on a part to edit, export or remove. The edit option will pop-up a part editor window. You can edit the image, connections and properties of a part in this window. For more on this see Fritzing Part Creation.

• **Bin style:** This button allows you to switch between two viewing styles. Try it out, see which you like.

• **Bin search:** Enter the part or information about a part to find it. For example: to find a resistor with a value of 220Ω you can enter either "resistor" or "220", results are displayed in the "search" bin tab.

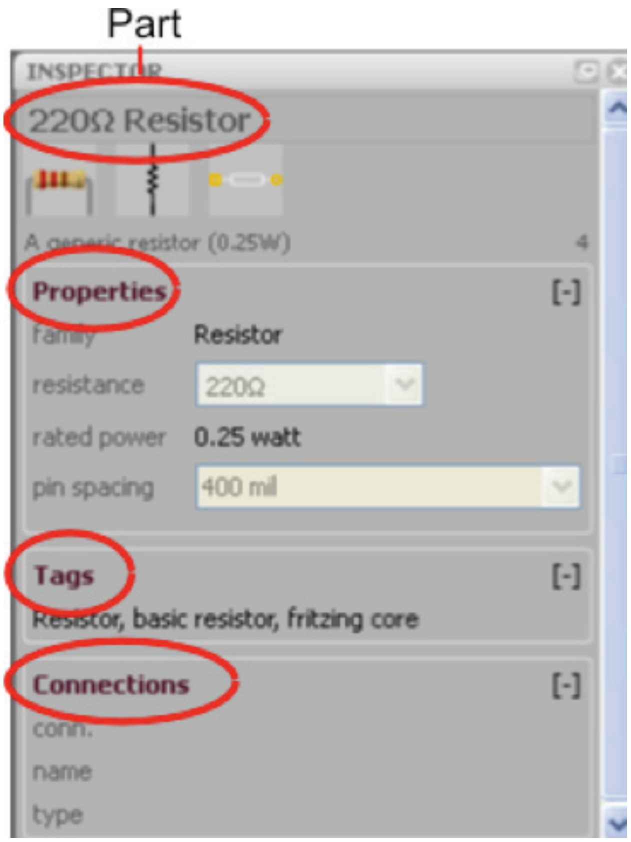
• **New part:** This sub menu allows you to create or import a new part, and edit, remove or export an existing part. The new and edit options will pop-up a part editor window. You can edit the image, connections and properties of a part in this window. For more on this see the Fritzing, New Part Creation section.

• **New bin:** This sub menu allows you to control the bin tabs. You can create new bins, as well as open, save, export, close and rename existing bins.

// Fritzing, Menus:

Fritzing Inspector Menu

The Inspector menu is where information about the currently selected part is displayed.



220Ω Resistor

A generic resistor (0.25W)

Properties

family Resistor
resistance 220Ω
rated power 0.25 watt
pin spacing 400 mil

Tags

Resistor, basic resistor, fritzing core

Connections

conn.
name
type

• **Part name and images:** Pretty self explanatory, this is where information and images of the part you have selected are displayed. The three different images from left to right are the breadboard, schematic and PCB silkscreen images.

• **Properties:** This sub menu displays properties particular to the part selected. The family section displays the part's component type. Below the family section various properties of the part are displayed. In this example resistance is an electrical value that you can change to fit your project's needs. Rated power is the amount of power this resistor can safely tolerate. Pin spacing indicates the amount of space between the two pins on this part, at this point in development pin spacing only effects the PCB footprint so you will not see any difference in the other views. Other examples of properties you may see in this sub menu include: Capacitance, rated Voltage, component type (crystal or ceramic for example), package type, doping (impurity type of transistor), maximum resistance and more depending on the part!

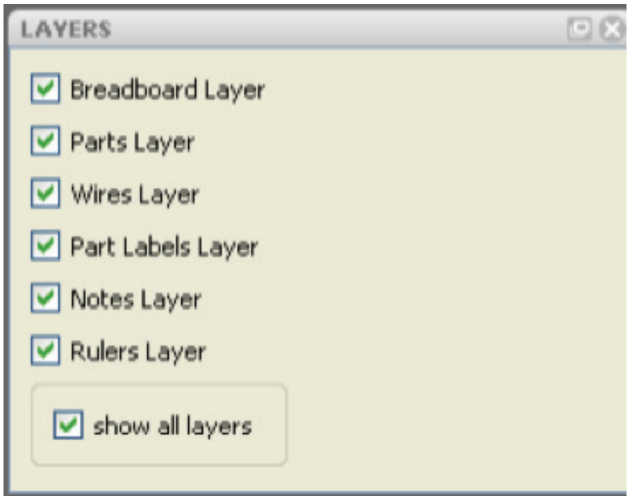
• **Tags:** This portion is in development, for now it includes information that designates where a part is located and text that can be used to search for the part.

• **Connections:** Once you have placed a part in your virtual prototype the connections made with this part will show up in this menu. Place your cursor over the wires (AKA leads) on the part in the main view to see the connection information. The Conn. field displays how many items are connected to to the lead. The name and type fields display the name of the connection (pin1, pin0, wire1, wire2, etc.) and the type (wire, male, female, etc.).

// Fritzing, Menus:

Fritzing Layer Menu

The Layers menu is where you can turn the various layers of the view on and off. Some of these layers are specific to the view, but others are available in all views.



LAYERS

☑ Breadboard Layer
☑ Parts Layer
☑ Wires Layer
☑ Part Labels Layer
☑ Notes Layer
☑ Rulers Layer

☑ show all layers

- **Breadboard Layer:** Shows the breadboard, available only in breadboard view.

- **Parts Layer:** Shows parts, available in all views.

- **Wires Layer:** Shows wires, available in all views.

- **Part Labels Layer:** In order to show the parts labels of a particular part right click on the part and select “show part label”, shows all parts labels, available in all views.

- **Nets Layer:** Shows connecting wires that are not yet routed, available only in schematic view.

- **Board Layer:** Shows the Printed Circuit Board, available only in PCB view.

- **Silkscreen Bottom Layer:** This layer is in development (N/A), available only in PCB view.

- **Silkscreen Bottom (Parts Label) Layer:** Same as Silkscreen Bottom Layer.

- **Copper Bottom Layer:** Shows the copper around the connections, available only in PCB view.

- **Copper Top Layer:** This layer is in development (N/A), available only in PCB view.

- **Silkscreen Top Layer:** Shows images (AKA silkscreen) on PCB which indicate where parts are placed, text and images, available only in PCB view.

- **Silkscreen Top (Parts Label) Layer:** In order to show the parts labels of a particular part right click on the part and select “show part label”, shows all images (AKA silkscreen) on PCB of parts labels, available only in PCB view.

- **Part Image Layer:** This layer is in development (N/A), available only in PCB view.

- **Rat’s Nest Layer:** Shows the most direct route between connected parts, available only in PCB view.

- **Notes Layer:** Shows notes, available in all views.

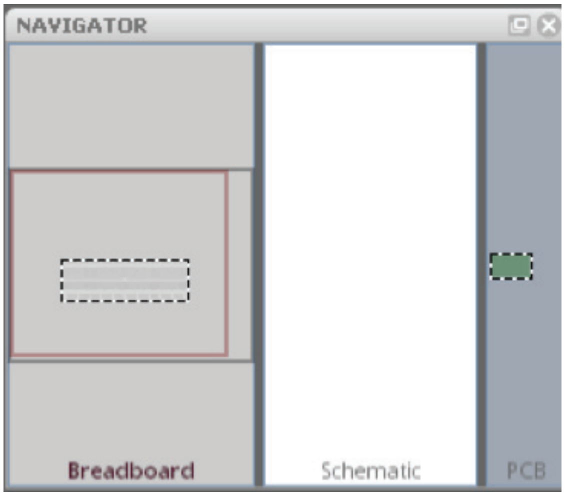
- **Rulers Layer:** Shows rulers, available in all views.

// Fritzing, Menus:

Fritzing Navigator Menu

The Navigator menu is where you can switch between the three different views; Breadboard, Schematic and PCB.

Another cool aspect of Fritzing is that when you add or rearrange wires (or traces) in one view, you can see the results in the navigator menu images of the other views. So if you are paying attention you can tell when you have altered your original circuit in a view other than the Breadboard view.



- **Breadboard view:** Click on this section to display the breadboard view.

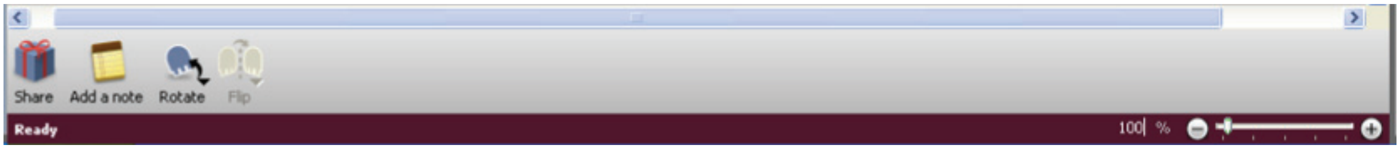
- **Schematic view:** Click on this section to display the schematic view.

- **PCB view:** Click on this section to display the PCB view.

Note: The Fritzing Parts, Inspector, Layer and Navigator windows are all re-sizable. It is also possible to undock these menu windows and place them where ever you like.

View Menu Options

The three different views (Breadboard, Schematic and PCB) have various menu options below them. These view menu options are explained below.



- **Share:** Upload your project to the Fritzing website for help, comments, or just to share how awesome your project is! Available in all views.

- **Add a note:** Add a note about your project. The note will only display in the view you add it to. Available in all views.

- **Rotate:** Rotate a part, available in all views.

- **Flip:** Flip a part, available in breadboard and schematic views. Not available for all parts.

- **Autoroute:** Place electrical traces automatically, the computer will only autoroute existing wire connections on your breadboard. Note that autoroute will leave connections unrouted and cross wires, so always double check this! You will need to route by hand. Not available in breadboard view.

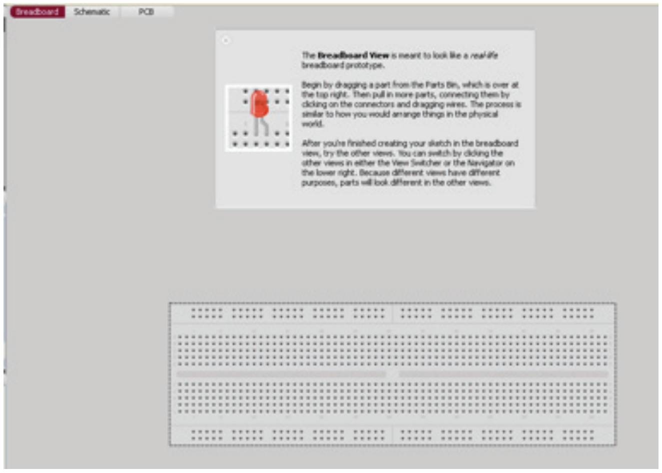
- **Export Etchable PDF:** Export a PDF with the traces and vias, for do it yourself PCB creation.

- **Percentage slider (set to 100% in the image above):** Zoom, available in all views.

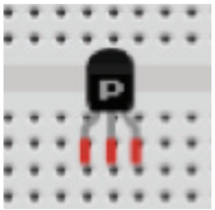
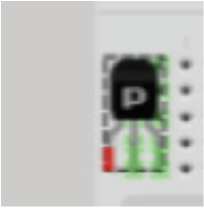
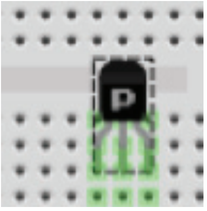
Name:
Date:

// Fritzing, Views:

Fritzing Breadboard View

The first of the three views in Fritzing is the Breadboard View. This is where you will recreate the physical prototype you created using your breadboard, Arduino, wires and components. In order to use Fritzing effectively it is important to actually create your physical prototype on a real live breadboard before using Fritzing. Believe me, it will save you a world of hassle in the long run.	 The **Breadboard View** is meant to look like a *real-life* breadboard prototype. Begin by dragging a part from the Parts Bin, which is over at the top right. Then pull in more parts, connecting them by clicking on the connectors and dragging wires. The process is similar to how you would arrange things in the physical world. After you're finished creating your sketch in the breadboard view, try the other views. You can switch by clicking the other views in either the View Switcher or the Navigator on the lower right. Because different views have different purposes, parts will look different in the other views.

- **Placing parts:** To place parts click and drag from your parts bin to the breadboard, when the part is placed correctly and the leads of the piece are inserted in holes the leads will turn green instead of red. Pay close attention to this step, it is possible to correctly place some leads but not all the leads. Once a part is placed correctly all the other breadboard holes that are connected to the part will be highlighted in green. These green holes are where you can connect wires to provide an electrical path for this lead. Right click on placed parts for additional options.

Incorrectly placed part	Sort of correctly placed part	Correctly placed part
		

- **Placing wires:** To place wires click on a breadboard hole and drag. To move the wire click and drag either end. To create a bend in the wire double-click or click and drag anywhere along the wire other than the two ends. Right click on the wire for additional options.

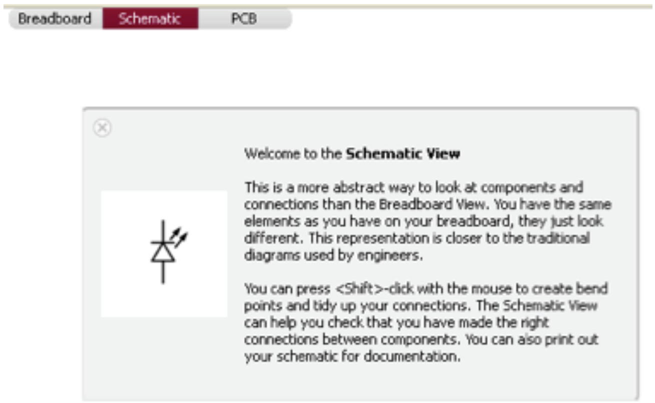
- **Connections:** To see all the wires and breadboard holes that are connected to any given point on the breadboard click and hold on that point. Any breadboard holes or wires that are connected will highlight in yellow.

Name:
Date:

// Fritzing, Views:

Fritzing Schematic View

The second of the three views in Fritzing is the Schematic View. This is where you can see the schematic version of the circuit you are creating in the breadboard view. This view is nice if you are familiar with schematics because everything is simplified and you can check your connections easily.



Welcome to the **Schematic View**

This is a more abstract way to look at components and connections than the Breadboard View. You have the same elements as you have on your breadboard, they just look different. This representation is closer to the traditional diagrams used by engineers.

You can press <Shift>-click with the mouse to create bend points and tidy up your connections. The Schematic View can help you check that you have made the right connections between components. You can also print out your schematic for documentation.

- **Nets layer:** This layer displays the connections you created using wires in the Breadboard view. These connections, known as nets, are unrouted, meaning they do not have a copper trace. The schematic view is where you will create the first set of routes. Nets are displayed as thin lines of different colors. When you first enter the schematic view after creating a circuit in the Breadboard view all connections are displayed as nets and need routing. To create a route, simply click and drag from the two ends of the components you want to connect. Once the connections have been routed the net will no longer be visible because a thicker wire line will overlay it.

- **Wires layer:** This displays the routes that have been created. Connected route ends are highlighted in green, unconnected route ends are highlighted in red.

- **Autoroute:** While Fritzing 0.4.3's autorouter is a step up from the original, you will need to double check the routes it creates as well as creating some on your own. To the right of the autoroute button there is a status field that tells you how many nets have been routed and how many are still unconnected.

- **Hand routing:** Let's face it, computers aren't all that great at creative solutions, you're going to have to create your own net routing time and time again. I promise it will get easier as you do it more often until it requires little to no thought. To make it easier for others to read your schematic make sure that the routes do not cross each other unintentionally.

- **Unconnected parts:** It is possible to place parts in the Breadboard view without connecting them. Unconnected parts will appear highlighted in red. To connect them in the Schematic view simply click and drag. This is not recommended as it will change your breadboard circuit without your knowledge. It is extremely confusing when you switch back to the Breadboard view and discover your circuit wires look nothing like the wires you originally placed.

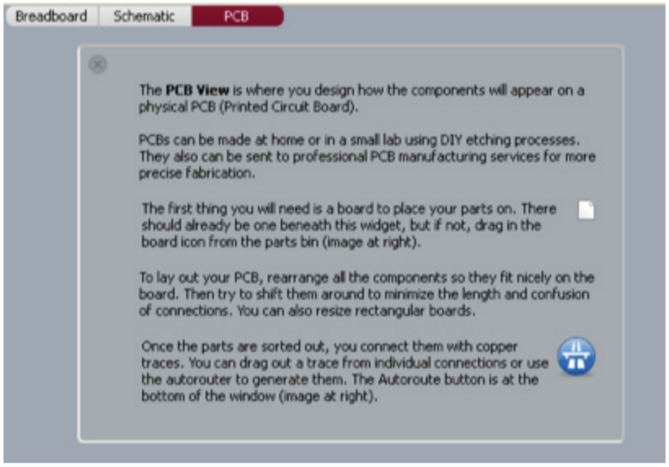
- **Additional options:** Right click on nets, wires and parts for more options including changing the circuit, autorouting a single net at a time and deleting wires and nets. Be careful with this option because it will actually delete wires and traces from the other views.

Name:
Date:

// Fritzing, Views:

Fritzing PCB View

The third of the three views in Fritzing is the PCB View. This is where you can see the Printed Circuit Board version of the circuit you are creating in the breadboard view. This is what the actual finished Printed Circuit Board will look like before you solder the components onto it.



The **PCB View** is where you design how the components will appear on a physical PCB (Printed Circuit Board).

PCBs can be made at home or in a small lab using DIY etching processes. They also can be sent to professional PCB manufacturing services for more precise fabrication.

The first thing you will need is a board to place your parts on. There should already be one beneath this widget, but if not, drag in the board icon from the parts bin (image at right).

To lay out your PCB, rearrange all the components so they fit nicely on the board. Then try to shift them around to minimize the length and confusion of connections. You can also resize rectangular boards.

Once the parts are sorted out, you connect them with copper traces. You can drag out a trace from individual connections or use the autorouter to generate them. The Autoroute button is at the bottom of the window (image at right).

- **The PCB:** The green rectangle that represents the actual board you are placing part “footprints” (the electrical contacts you will solder the parts to) on. The PCB is also a part you can find in the core parts bin, it looks like a green square.

- **Arranging parts:** PCB manufacturers usually charge a flat rate plus an amount that depends on the size of your PCB, so it’s a good idea to try and create as small a PCB as possible. Even if you are planning on etching your own board using Ferric Chloride (talk about a cool science project!) smaller traces will use less of your materials and time, as well as creating less frustration. Rearranging parts in the PCB view will not effect the parts in the other views.

- **Traces:** Traces are the copper paths that connect the various parts on your PCB. In the PCB view unconnected traces look like plain black lines running from part to part. Every single unconnected trace should be turned into a connected trace by clicking and dragging from one end of the unconnected trace (at the connector) to the other end of the trace (the other connector). Clicking once in the middle of the unconnected trace will also create a trace, but it will put a bend in the trace as well.

- **Connector:** Connectors are the circles in parts that connect the part to the board and traces. Connectors with any type of trace (connected or unconnected) are displayed as green dots with yellow outlines. Connectors with no traces are displayed as red dots with yellow outlines, pay special attention to these connectors because usually all connectors need traces. To see what connectors are directly connected to each other click and hold on a connector. All other attached connectors will highlight in yellow.

- **Silkscreen layer:** The silkscreen layer is where part outlines, text, and images are displayed. For example if you are creating a PCB to control an autonomous marshmallow catapult you might place the text “Catapult Brain v. 2.0” along with part outlines on your PCB so anyone putting together the board will know where to put the parts as well as what the board is used for.

Name:

Date:

// Fritzing, Views:

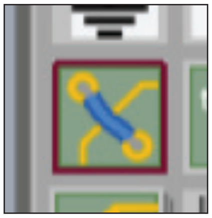
Fritzing PCB View II (Special Parts and Tools to Note)

- **Jumpers:** Sometimes there is no way to connect all the traces on a board without crossing traces over each other, which will short out the traces. To avoid this use a jumper. A jumper is two connectors that are left untraced so later you can attach a wire “jumping” from one connector over the trace to the connector on the other side. To place a jumper click and drag the jumper to where you wish to cross a trace, then click and drag from one connector on the jumper to the connector on the part. Do this with both sides of the jumper. If you have correctly placed the jumper it should look like a blue line crossing a trace. When you click on one side of the jumper all traced connectors will highlight in yellow, indicating you have correctly placed the jumper.

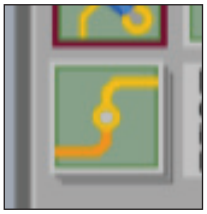
- **Vias:** Vias are holes with copper lining the inside. On two sided PCB designs vias are used in place of jumpers to switch a trace from one side of the board to the other. To place a via simply click and drag the part to where you want a via.

- **Silkscreen text:** Often a PCB designer will need to put some text on the PCB for one reason or another. The silkscreen text part is designed for that purpose. To place silkscreen text click and drag the Silkscreen text part onto your board. Use the Inspector menu to change the text displayed. This part can also be used to place images.

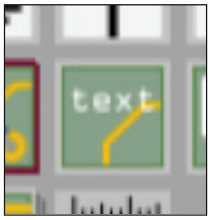
- **Silkscreen image:** If you’ve got a sweet logo you want to put on your PCB this is the part you will use to place it. To place a silkscreen image click and drag this part onto your board. Click on “Load Image” in the Inspector menu to load an image you have created for the board, then select the image from the drop down menu just above the “Load Image” button.



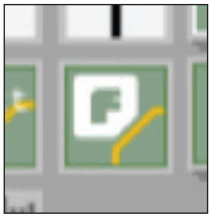
Jumper



Via



Text



Image

- **Design Rules Check:** To use the Design Rules Check click on the Trace menu and select Design Rules Check. The Design Rules Check checks your PCB design for overlapping parts and traces. The tool displays the number of overlapping parts (includes parts and wires) in text below the PCB View. This tool will not check for overlapping traces that have not yet been connected. Make sure to run this tool before you consider your design finished.

- **Programming window:** To open the Programming window click on the Window menu and select “open Programming window”. The Programming window is an experimental feature in development. It can be used for Picaxe and Arduino programming. This window plays the same role as the Arduino Environment, but is not intended to replace it.

- **Copper fill:** To create a copper fill click on the Trace menu and select Copper Fill. To remove a copper fill click on the Trace menu and select Remove Copper Fill. A copper fill is a layer of copper inside the PCB. You can treat this layer like a huge wire, in more advanced PCB layout software it is often used as a ground connection. It is possible to connect existing traces to the copper fill by moving the trace after creating a copper fill. Be careful with this option though because if you refill the copper it will insulate all wires from this layer.

// Fritzing, Views:

Fritzing PCB View III
Routing Tips and Tricks (From the Fritzing Website)

Place the parts with the most connections in the middle of the board.

Try to get short connections by moving and rotating parts.

Use the highlighting of equipotential connectors feature.

Add bend points for tidy routing and so that lines do not cross.

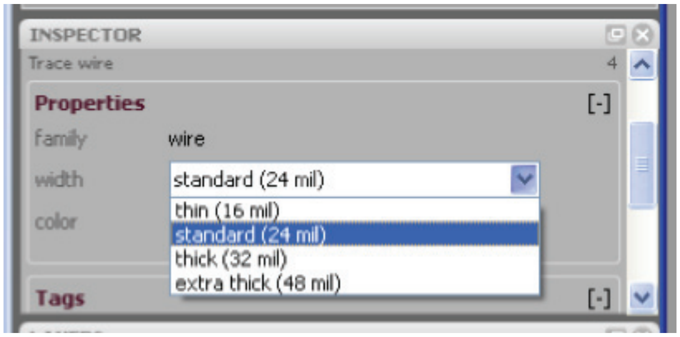
Don’t forget the traces can go under parts like resistors.

Use jumper wires instead of watching the auto-route go crazy.

To achieve a better and nicer design, you would need to edit traces by moving, adjusting width and adding bend points. Width adjustment can be done in the Inspector. Please note that thin traces might ruin a DIY PCB production (if the traces are too thin the electrical current will not be able to flow properly), so keeping traces in medium thickness is safer.

Adjusting Traces and Wire Width

To adjust the trace or wire width click on the trace and select Width in the Properties submenu of the part Inspector Menu.



family wire

width standard (24 mil)

Name:
Date:

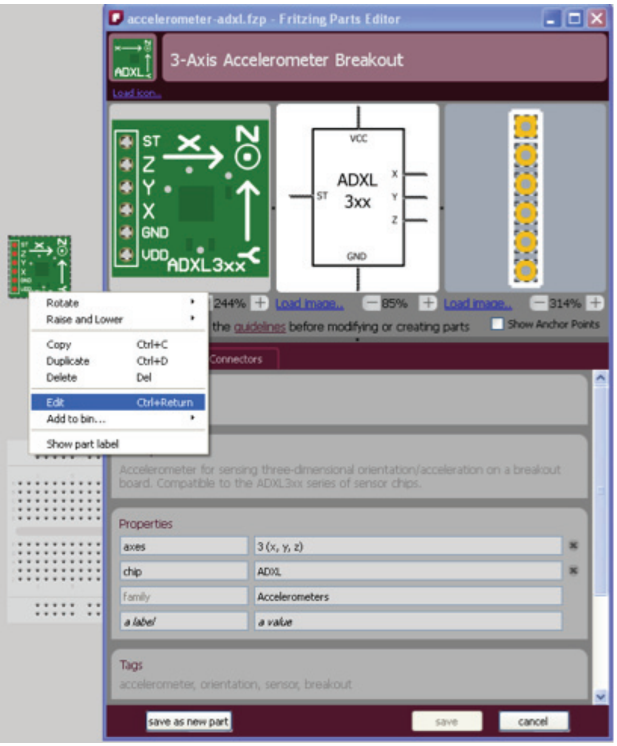
// Fritzing, Editing a Part:

Editing existing Parts in Fritzing

Once you have become familiar with Fritzing and virtual prototyping you may find that a part you are working with on your physical breadboard is not present in the Fritzing library. Your first move should be to see if you can find a part in the existing library that is similar to the part you are working with. If you can find a similar part, then all you need to do is edit the existing virtual part so it matches the one you are working with.

To do this drag and drop the similar part to your view window. Open the part editor by either right clicking on the part and selecting Edit or selecting Edit under the Part menu.

The pop-up Fritzing Parts Editor window will look like the example to the right. If you have an internet connection definitely check out the guidelines for editing and creating parts.



accelerometer-adxl.fzp - Fritzing Parts Editor

3-Axis Accelerometer Breakout

the guidelines before modifying or creating parts

Accelerometer for sensing three-dimensional orientation/acceleration on a breakout board. Compatible to the ADXL3xx series of sensor chips.

Properties

axes	3 (x, y, z)
chip	ADXL
family	Accelerometers
a label	*a value*

Tags
accelerometer, orientation, sensor, breakout

- **Images:** Load images you have created for each of the three views at the top of the Parts Editor. The image on the left should look like the physical part, the image in the center should be a schematic representation of the part (all leads must be shown) and the image on the right is simply an image of all the leads present on the part. The leads on these images are very important, without the proper connections your PCB will not work. You may also wish to include a silkscreen image of the part so you do not try to place the footprints too close together on the PCB. If the parts are too close together you will not be able to physically fit them next to each other on the PCB.

- **Specifications:** Various text fields that provide information about the part. These do not actually effect the virtual part, but provide information about the part for users.

- **Connectors:** This is the most important portion of editing a part. The next page is dedicated to the connectors menu.

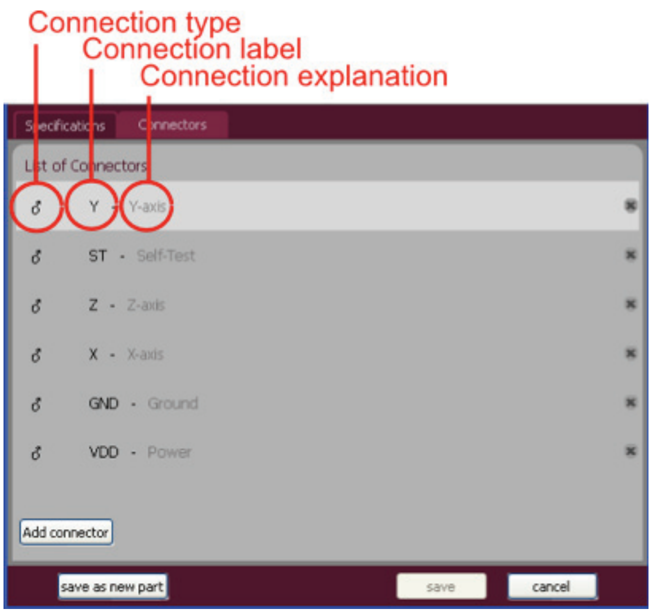
- **Saving:** Each sub-menu inside of the part editor will ask you to either save or cancel the changes you make. At the very bottom of the parts editor are the buttons labeled Save as New Part, Save and Cancel. When editing an existing part always Save as New Part in order to avoid permanently changing the part in case you will need it later in the prototyping process.

Name:
Date:

// Fritzing, Connectors:

Editing Connector Parts in Fritzing

Connectors are very important for PCB layout. The connectors are where the parts are actually connected to the PCB and traces. Make sense, huh? Because connectors are so important you need to make sure that the information about the connectors in the parts you edit or create are correct and in the proper places.



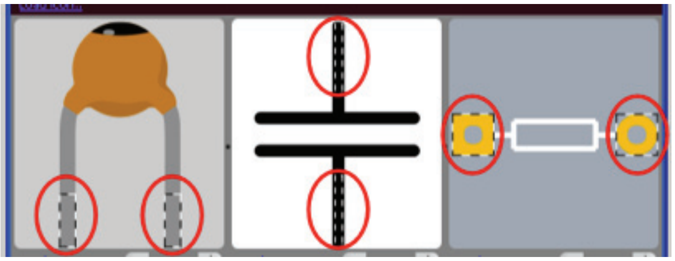
♂	Y	Y-axis
♂	ST	Self-Test
♂	Z	Z-axis
♂	X	X-axis
♂	GND	Ground
♂	VDD	Power

- **Connection type:** The type of connection present on the physical part. The three options are Male, Female and Pad. Pad is the connection used in surface mount soldering.

- **Connection label:** The label on the part label layer, often an abbreviation.

- **Connection explanation:** An explanation of the connection label.

The images in the Fritzing Parts Editor indicate the location of the connectors with bounding boxes, circled in red in the image to the right. You can move and resize these bounding boxes to reflect the physical connectors.



Creating New Parts in Fritzing

The process of creating new parts in Fritzing uses the exact same set of menus and options that editing an existing part does, only you will need to fill in every single field and compare every connector to the physical part because the New Part fields start completely blank.

For more help with editing and creating parts:

Online Fritzing Libraries: http://fritzing.org/parts/
Help with creating parts: http://fritzing.org/learning/tutorials/creating-custom-parts/

// From Fritzing to Physical PCB:

OK, you've checked and double checked your virtual prototype to make sure it matches the physical prototype on your breadboard. You've checked and double checked the PCB layout in Fritzing and you're ready to take the final step of virtual prototyping; exporting PCB layout files so you can create a physical PCB!

There are two ways you can export the necessary files to create your physical PCB. The standard option is to export Gerber files, zip them and send them to a PCB prototype facilitator such as BatchPCB. A facilitator allows you, the inventor (that's right you are now officially an inventor), to purchase your prototype boards one at a time instead of five or ten at a time, because, let's face it, there might be some errors.

Exporting files for a PCB prototype facilitator: Create a folder to hold the Gerber files and only the Gerber files. Go to the File drop down menu, click on Export and select To Gerber. This will pop-up a window asking you where you would like to save the files, find the folder you created, select it and click OK. If you have not already created a folder there is also a Create Folder option. This will create five Gerber files and a text file. Zip all these files and send them to your friendly PCB prototype facilitator. Sit tight for a couple weeks and when your PCB prototypes show up in the mail, if you have the parts on hand, you're ready to start soldering everything together!

The second option for PCB prototype creation is to export an etchable PDF file and create the PCB yourself. No matter how you attempt this second option it requires some fairly advanced technology.

Exporting files for etching your own PCB: This first step is pretty easy, just click on the Export Etchable PDF button in the PCB view and save wherever you like. Actually etching the PCB is a whole different topic, here's a link to get you started:

For help etching your own PCBs:
Fritzing tutorial: http://fritzing.org/learning/tutorials/pcb-production-tutorials/diy-pcb-etching/

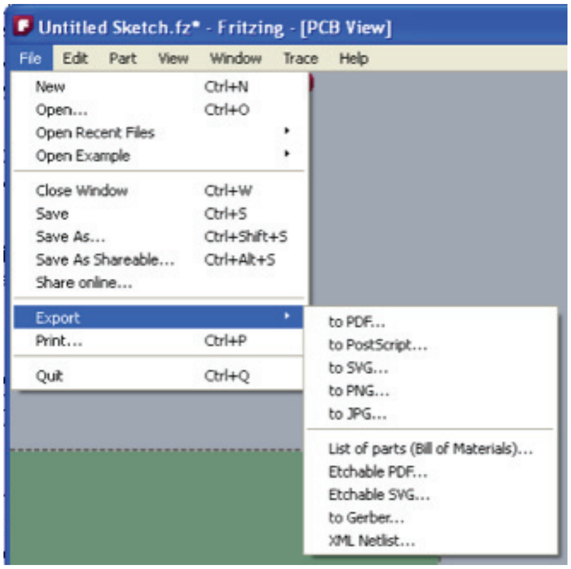
Other export file type options:

PDF, PostScript, SVG, PNG and JPG:
Image file formats.

List of parts:
Text file format listing the parts and components necessary to create your prototype.

Etchable SVG:
Similar to Etchable PDF, a different image file format.

XML Netlist:
Code file format.


Untitled Sketch.fz* - Fritzing - [PCB View]
File
Edit
Part
View
Window
Trace
Help
New Ctrl+N
Open... Ctrl+O
Open Recent Files
Open Example
Close Window Ctrl+W
Save Ctrl+S
Save As... Ctrl+Shift+S
Save As Shareable... Ctrl+Alt+S
Share online...
Export
Print... Ctrl+P
Quit Ctrl+Q
to PDF...
to PostScript...
to SVG...
to PNG...
to JPG...
List of parts (Bill of Materials)...
Etchable PDF...
Etchable SVG...
to Gerber...
XML Netlist...