http://www.cs.csub.edu/~is-green/

Phone:
(661) 654-6005

Fax:
(661) 654-6960

Email:
isgreen@cs.csubak.edu

California State University Bakersfield
Department of Computer & Electrical Engineering & Computer Science

Mail Stop: 63 SCI
9001 Stockdale Highway
Bakersfield, California 93311-1022
Office: Science III 339

Summer Outreach Programs 2015

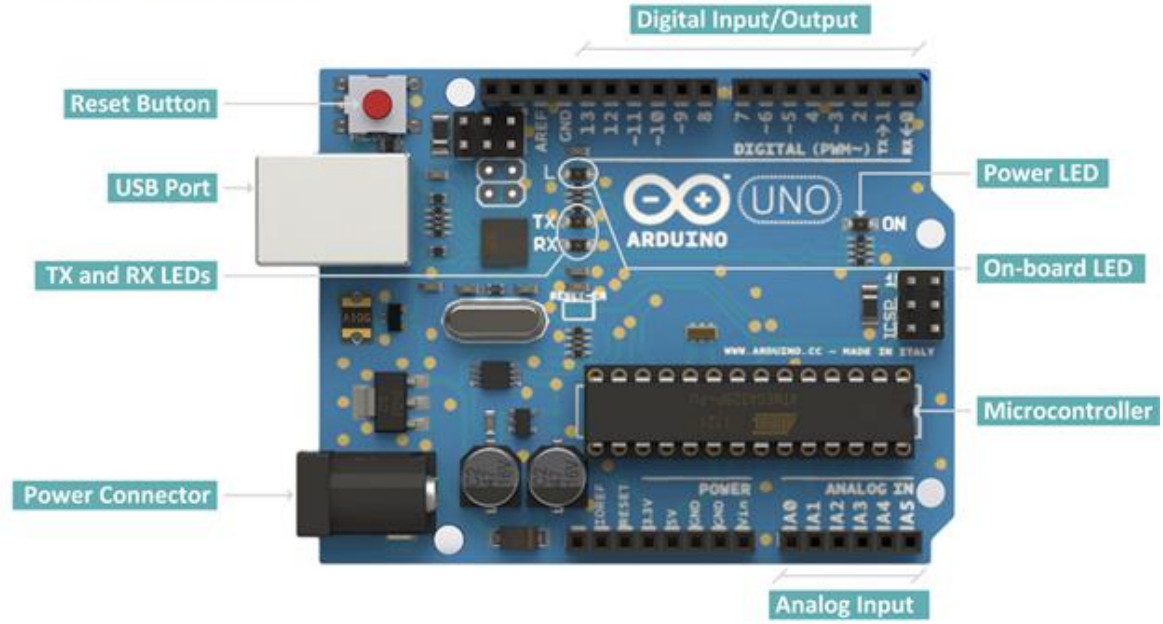
Packet prepared by:

Sheriff Sadiqbatcha Alex Ramirez Joshua Ward

Introduction to Arduino Microcontroller

General Overview

- What is an Embedded System?
  - Microcontroller
  - Programmable vs. Assignment Specific
- Arduino



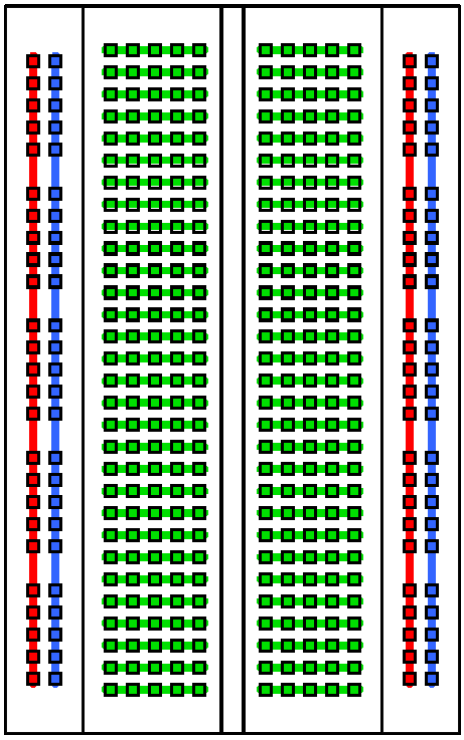
- Arduino IDE
  - Programming languages
  - System level C
  - Bare minimum code needed to start an Arduino sketch

```
// the setup routine runs once when you press reset:
void setup() {
}


// the loop routine runs over and over again forever:
void loop() {
}
```

Hardware Essentials

Breadboard



A breadboard is a construction base for prototyping of electronics. Originally it was literally a bread board, a polished piece of wood used for slicing bread. In the 1970s the solderless breadboard (AKA plug-board) became available and nowadays the term "breadboard" is commonly used to refer to these. "Breadboard" is also a synonym for "prototype".

Because the solderless breadboard does not require soldering, it is reusable. This makes it easy to use for creating temporary prototypes and experimenting with circuit design. For this reason, solderless breadboards are also extremely popular with students and in technological education.

USEFUL LINKS

Breadboard

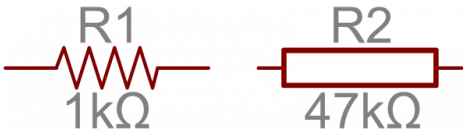
Resistors

Resistors are electronic components which have a specific, never-changing electrical resistance. The resistor's resistance limits the flow of electrons through a circuit.

They are passive components, meaning they only consume power (and can't generate it). Resistors are usually added to circuits where they complement active components like op-amps, microcontrollers, and other integrated circuits. Commonly resistors are used to limit current, divide voltages, and pull-up I/O lines.

The electrical resistance of a resistor is measured in ohms. The symbol for an ohm is the Greek capital-omega: Ω. The (somewhat roundabout) definition of 1Ω is the resistance between two points where 1 volt (1V) of applied potential energy will push 1 ampere (1A) of current.

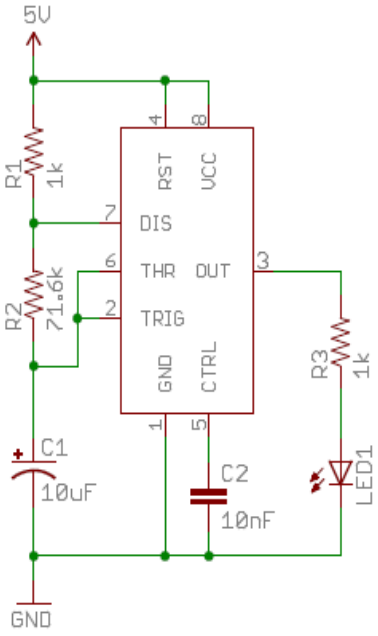
All resistors have two terminals, one connection on each end of the resistor. When modeled on a schematic, a resistor will show up as one of these two symbols:

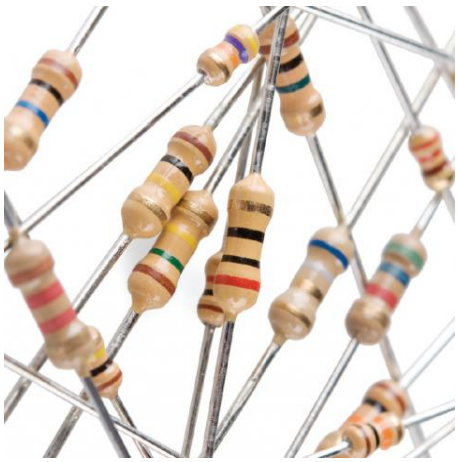


Two common resistor schematic symbols. R1 is an American-style 1kΩ resistor, and R2 is an international-style 47kΩ resistor.

The terminals of the resistor are each of the lines extending from the squiggle (or rectangle). Those are what connect to the rest of the circuit.

The resistor circuit symbols are usually enhanced with both a resistance value and a name. The value, displayed in ohms, is obviously critical for both evaluating and actually constructing the circuit. The name of the resistor is usually an *R* preceding a number. Each resistor in a circuit should have a unique name/number. For example, here's a few resistors in action on a 555 timer circuit:





Through-hole, axial resistors usually use the color-band system to display their value. Most of these resistors will have four bands of color circling the resistor.

The first two bands indicate the two most-significant digits of the resistor's value. The third band is a weight value, which multiplies the two significant digits by a power of ten.

The final band indicates the tolerance of the resistor. The tolerance explains how much more or less the *actual* resistance of the resistor can be compared to what its nominal value is. No resistor is made to perfection, and different manufacturing processes will result in better or worse tolerances. For example, a 1kΩ resistor with 5% tolerance could actually be anywhere between 0.95kΩ and 1.05kΩ.

How do you tell which band is first and last? The last, tolerance band is often clearly separated from the value bands, and usually it'll either be silver or gold.

Here's a table of each of the colors and which value, multiplier or tolerance they represent:

Color	Digit value	Multiplier	Multiplied Out	Tolerance
Black	0	10^0	1	
Brown	1	10^1	10	
Red	2	10^2	100	
Orange	3	10^3	1,000	
Yellow	4	10^4	10000	
Green	5	10^5	100,000	
Blue	6	10^6	1,000,000	
Violet	7	10^7	10,000,000	
Gray	8	10^8	100,000,000	
White	9	10^9	1,000,000,000	
Gold				±5%
Silver				±10%

Here's an example of a 4.7kΩ resistor with four color bands:

When decoding the resistor color bands, consult a resistor color code table like the one above. For the first two bands, find that color's corresponding digit value. The 4.7kΩ resistor has color bands of yellow and violet to begin - which have digit values of 4 and 7 (47). The third band of the 4.7kΩ is red, which indicates that the 47 should be multiplied by 10^2 (or 100). 47 times 100 is 4,700!

Digital Tutorials

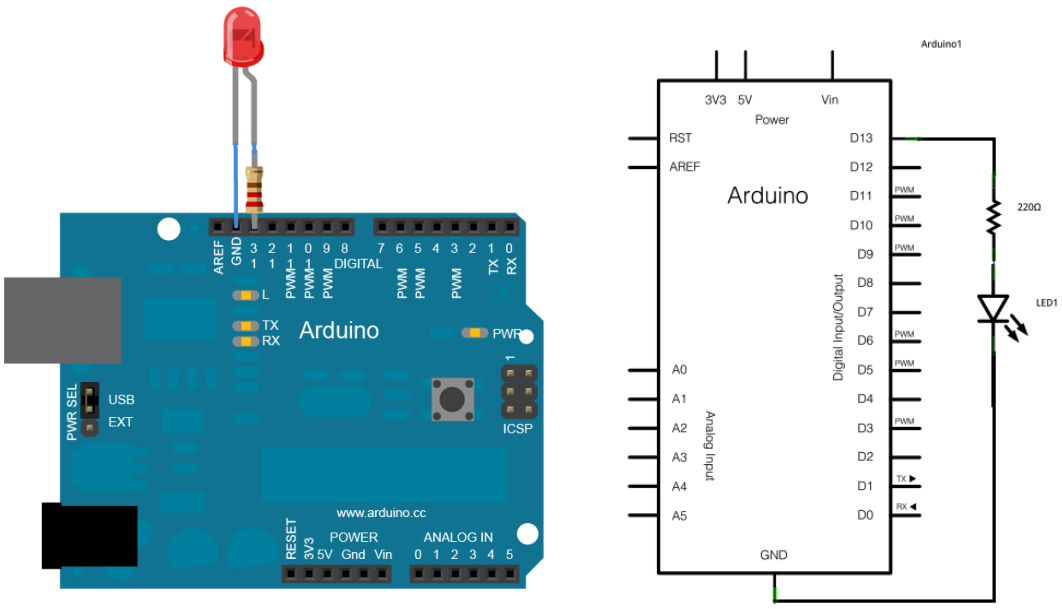
Turn on an LED

This example shows the simplest thing you can do with an Arduino to see a physical output.

Hardware Required

- Arduino Board
- LED
- Resistor, anything between 220 ohm to 1K ohm

To build the circuit, attach a 220-ohm resistor to pin 13. Then attach the long leg of an LED (the positive leg, called the anode) to the resistor. Attach the short leg (the negative leg, called the cathode) to ground. Then plug your Arduino board into your computer, start the Arduino program.



In the program, before the setup, the first thing you do is assign a name to pin 13

```
int ledPin = 13;
```

In the setup, initialize pin 13 as an output pin with the line

```
pinMode(ledPin, OUTPUT);
```

Then you turn the LED on with the line:

```
digitalWrite(ledPin, HIGH);
```

CODE

```
int ledPin = 13;                // LED connected to digital pin 13

void setup()
{
  pinMode(ledPin, OUTPUT);      // sets the digital pin as output
  digitalWrite(ledPin, HIGH);   // sets the LED on
}
```

```
void loop()
{
}
```

USEFUL LINKS

setup()
pinMode()
digitalWrite()

Blink LED

Turn the LED on, delay for one second, and then turn the LED off.
Delay: Pauses the program for the amount of time (in miliseconds) specified as parameter. (There are 1000 milliseconds in a second.)

CODE

```
int ledPin = 13;                // LED connected to digital pin 13

void setup()
{
  pinMode(ledPin, OUTPUT);      // sets the digital pin as output
  delay(1000);                  // waits for a second
  digitalWrite(ledPin, HIGH);   // sets the LED on
  delay(1000);                  // waits for a second
  digitalWrite(ledPin, LOW);    // sets the LED off
}

void loop()
{
}
```

NOTE: You may have to press the reset button on the Arduino board in order to notice the LED blink.

USEFUL LINKS

delay()

Blink continuously

Moving the lines of code used to turn the LED on and off over to the loop function will allow the sequence to execute indefinitely.
NOTE: Pin-mode always has to be declared in the setup since it only needs to be executed once to set the behavior of the pin as an input or an output.

CODE

```
int ledPin = 13;                // LED connected to digital pin 13

void setup()
{
  pinMode(ledPin, OUTPUT);      // sets the digital pin as output
}

void loop()
{
  digitalWrite(ledPin, HIGH);   // sets the LED on
  delay(1000);                  // waits for a second
  digitalWrite(ledPin, LOW);    // sets the LED off
  delay(1000);                  // waits for a second
}
```

USEFUL LINKS

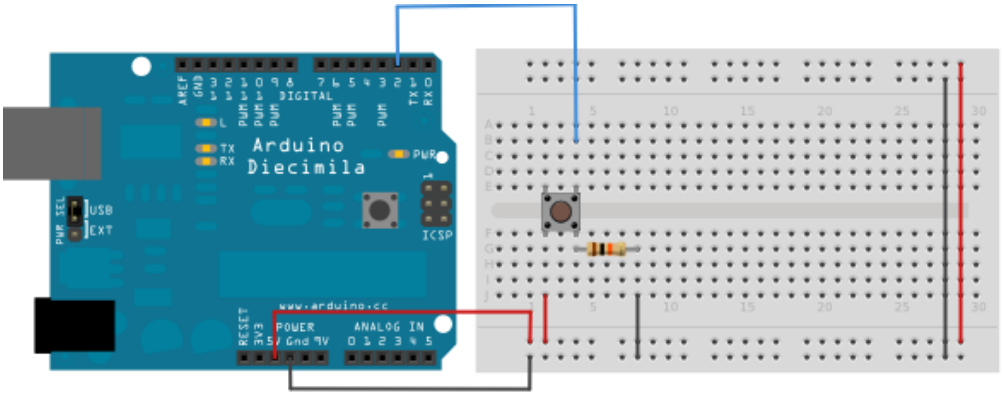
loop()

LED push button control

Pushbuttons or switches connect two points in a circuit when you press them. This example turns on the LED on pin 13 when you press the button.

Hardware Required

- Arduino Board
- LED
- Momentary button or switch
- 10K ohm resistor
- Breadboard
- Hook-up wire



Connect three wires to the Arduino board. The first two, red and black, connect to the two long vertical rows on the side of the breadboard to provide access to the 5 volt supply and ground. The third wire goes from digital pin 2 to one leg of the pushbutton. That same leg of the button connects through a pull-down resistor (here 10 KOhms) to ground. The other leg of the button connects to the 5 volt supply.

When the pushbutton is open (unpressed) there is no connection between the two legs of the pushbutton, so the pin is connected to ground (through the pull-down resistor) and we read a LOW. When the button is closed (pressed), it makes a connection between its two legs, connecting the pin to 5 volts, so that we read a HIGH.

You can also wire this circuit the opposite way, with a pullup resistor keeping the input HIGH, and going LOW when the button is pressed. If

so, the behavior of the sketch will be reversed, with the LED normally on and turning off when you press the button.

If you disconnect the digital i/o pin from everything, the LED may blink erratically. This is because the input is "floating" - that is, it will randomly return either HIGH or LOW. That's why you need a pull-up or pull-down resistor in the circuit.

NOTE: The LED is still connected to pin 13 from your previous exercise.

CODE

```
// constants won't change. They're used here to
// set pin numbers:
const int buttonPin = 2;     // the number of the pushbutton pin
const int ledPin =  13;      // the number of the LED pin

// variables will change:
int buttonState = 0;         // variable for reading the pushbutton status

void setup() {
  // initialize the LED pin as an output:
  pinMode(ledPin, OUTPUT);
  // initialize the pushbutton pin as an input:
  pinMode(buttonPin, INPUT);
}

void loop(){
  // read the state of the pushbutton value:
  buttonState = digitalRead(buttonPin);

  // check if the pushbutton is pressed.
  // if it is, the buttonState is HIGH:
  if (buttonState == HIGH) {
    // turn LED on:
    digitalWrite(ledPin, HIGH);
  }
```

```
  else {
    // turn LED off:
    digitalWrite(ledPin, LOW);
  }
}
```

USEFUL LINKS

digitalRead()
if()
else()

LED push button state change

Once you've got a pushbutton working, instead of having to hold down the button you often want to change your program to work on individual button presses. The sketch below continually reads the button's state, when a button press is detected the state of the LED is flipped.

NOTE: LOW, false, and 0 can be used interchangeably. HIGH, true, and 1 can also be used interchangeably.

CODE

```
// constants won't change. They're used here to
// set pin numbers:
const int buttonPin = 2;     // the number of the pushbutton pin
const int ledPin =  13;      // the number of the LED pin

boolean lastButton =  LOW;
boolean ledOn =  false;

void setup() {
  // initialize the LED pin as an output:
  pinMode(ledPin, OUTPUT);
  // initialize the pushbutton pin as an input:
  pinMode(buttonPin, INPUT);
}
```

```
void loop(){

   // if it is, the buttonState is HIGH:
  if (digitalWrite(buttonPin) == HIGH && lastButton == LOW) {
   // turn LED on:
    ledOn = !ledOn;
    lastButton = HIGH;
  }
  else {
   // lastButton = LOW;
    ledButton = digitalRead(buttonPin);
  }
  digitalWrite(ledPin, ledOn);
}
```

USEFUL LINKS

constants
boolean operators
Variable Declaration
Debounce - read a pushbutton filtering noise

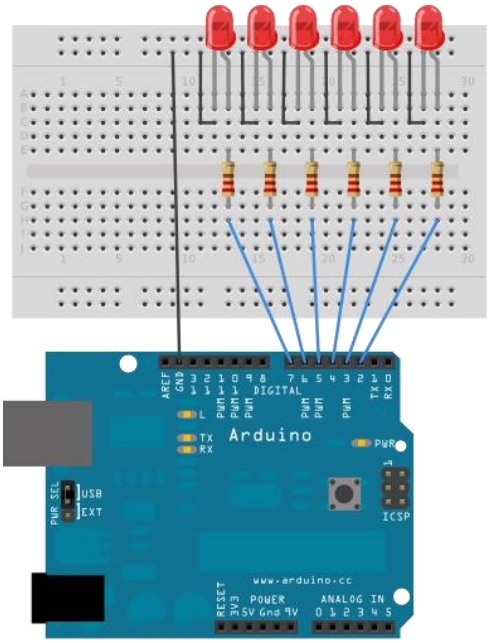
LED loop (A.k.a. Knight Rider)

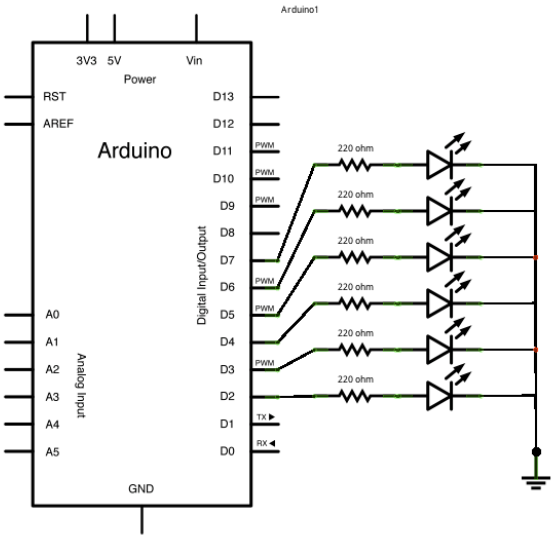
Often you want to iterate over a series of pins and do something to each one. For instance, this example blinks 6 LED's attached the Arduino by using a for() loop to cycle back and forth through digital pins 2-7. The LEDS are turned on and off, in sequence, by using both the digitalWrite() and delay() functions.

We also call this example "Knight Rider" in memory of a TV-series from the 80's where David Hasselhoff had an AI machine named KITT driving his Pontiac. The car had been augmented with plenty of LEDs in all possible sizes performing flashy effects. In particular, it had a display that scanned back and forth across a line, as shown in this exciting fight between KITT and KARR. This example duplicates the KITT display.

Hardware Required

- Arduino Board
- (6) 220 ohm resistors
- (6) LEDs
- hook-up wire
- breadboard





The code below begins by utilizing a for() loop to assign digital pins 2-7 as outputs for the 6 LEDs used.

In the main loop of the code, two for() loops are used to loop incrementally, stepping through the LEDs, one by one, from pin 2 to pin seven. Once pin 7 is lit, the process reverses, stepping back down through each LED.

CODE

```
int timer = 100;          // The higher the number, the slower the
                          // timing.

void setup() {
  // use a for loop to initialize each pin as an output:
  for (int thisPin = 2; thisPin < 8; thisPin++)  {
    pinMode(thisPin, OUTPUT);
  }
```

```
}

void loop() {
  // loop from the lowest pin to the highest:
  for (int thisPin = 2; thisPin < 8; thisPin++) {
    // turn the pin on:
    digitalWrite(thisPin, HIGH);
    delay(timer);
    // turn the pin off:
    digitalWrite(thisPin, LOW);
  }

  // loop from the highest pin to the lowest:
  for (int thisPin = 7; thisPin >= 2; thisPin--) {
    // turn the pin on:
    digitalWrite(thisPin, HIGH);
    delay(timer);
    // turn the pin off:
    digitalWrite(thisPin, LOW);
  }
}
```

USEFUL LINKS

for()

Traffic light with pedestrian crossing

In this exercise you will be left to design a system using the skills developed from the previous tutorials. You will have to create a traffic light system with a cross walk using red, green, yellow and white LEDs controlled by a button for crossing. The traffic light will begin with the green LED on, once the button is pressed the green LED will turn off. Once the green LED is off, the yellow LED will turn on for a short time period after which the red LED will turn on. Once the red LED is on, then the white LED will turn on for a certain time period signaling the pedestrians to cross. The system should then reset by turning the green LED on again.

Analog Tutorials

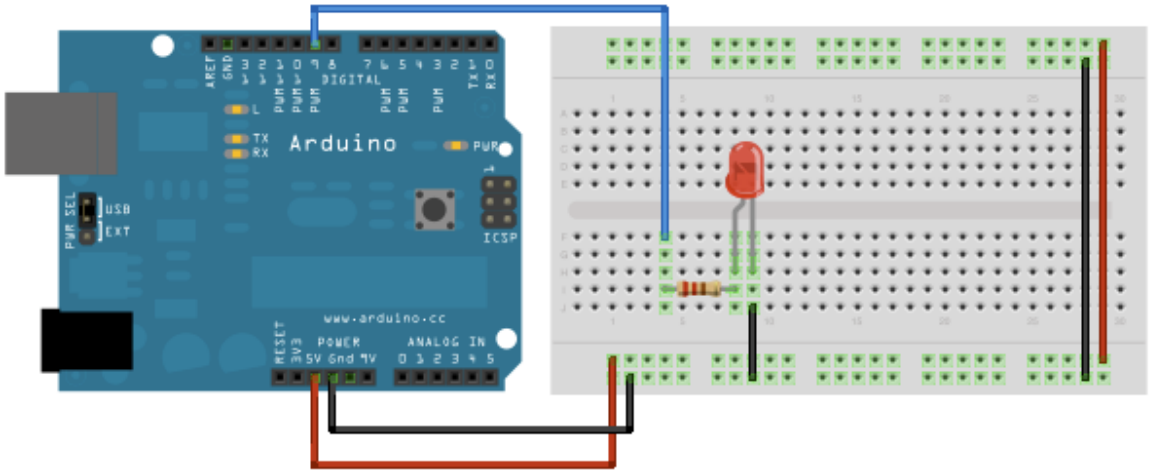
Fade LED

Demonstrates the use of the analogWrite() function in fading an LED off and on. AnalogWrite uses pulse width modulation (PWM), turning a digital pin on and off very quickly, to create a fading effect.

Hardware Required

- Arduino Board
- Breadboard
- An LED
- A 220 ohm resistor

Connect the anode (the longer, positive leg) of your LED to digital output pin 9 on your Arduino through a 220-ohm resistor. Connect the cathode (the shorter, negative leg) directly to ground.



After declaring pin 9 to be your ledPin, there is nothing to do in the setup() function of your code.

The analogWrite() function that you will be using in the main loop of your code requires two arguments: One telling the function which pin to write to, and one indicating what PWM value to write.

In order to fade your LED off and on, gradually increase your PWM value from 0 (all the way off) to 255 (all the way on), and then back to 0 once again to complete the cycle. In the sketch below, the PWM value is set using a variable called brightness. Each time through the loop, it increases by the value of the variable fade Amount. Once the maximum value of 255 is reached, it decreases by the same value until it reaches 0 and starts over again.

analogWrite() can change the PWM value very fast, so the delay at the end of the sketch controls the speed of the fade. Try changing the value of the delay and see how it changes the program.

CODE

```
int led = 9;           // the pin that the LED is attached to
int brightness = 0;    // how bright the LED is
int fadeAmount = 5;    // how many points to fade the LED by

// the setup routine runs once when you press reset:
void setup()  {
  // declare pin 9 to be an output:
  pinMode(led, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop()  {
  // set the brightness of pin 9:
  analogWrite(led, brightness);

  // change the brightness for next time through the loop:
  brightness = brightness + fadeAmount;

  // reverse the direction of the fading at the ends of the fade:
  if (brightness == 0 || brightness == 255) {
    fadeAmount = -fadeAmount ;
  }
  // wait for 30 milliseconds to see the dimming effect
```

```
  delay(30);
}
```

USEFUL LINKS

PWM

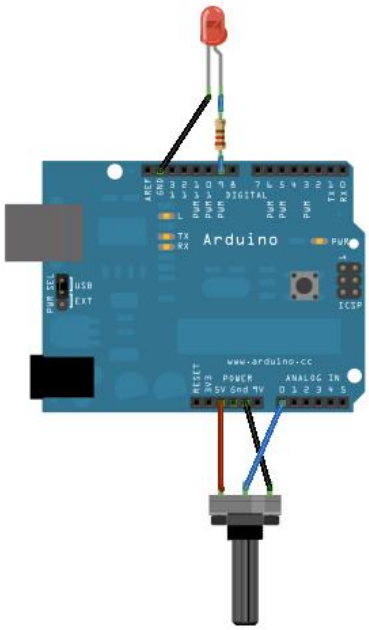
Fade LED with an analog input

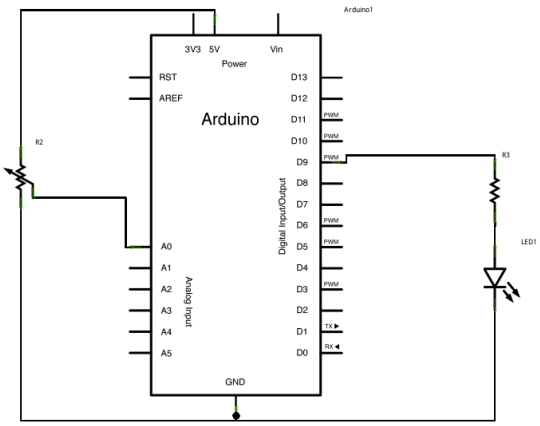
This example shows how to read an analog input pin, map the result to a range from 0 to 255, and then use that result to set the pulse width modulation (PWM) of an output pin to dim or brighten an LED.

Hardware Required

- Arduino Board
- Breadboard
- An LED
- A 220 ohm resistor
- Potentiometer (pot)

Connect one pin from your pot to 5V, the center pin to analog pin 0, and the remaining pin to ground. Next, connect a 220 ohm current limiting resistor to digital pin 9, with an LED in series. The long, positive leg (the anode) of the LED should be connected to the output from the resistor, with the shorter, negative leg (the cathode) connected to ground.





In the program below, two pin assignments (analog 0 for your potentiometer and digital 9 for your LED) and two variables, sensorValue and outputValue are first declared at the top. Then in setup the behavior of the input and output pins are set and serial communication is initiated.

Next, in the main loop of the code, sensorValue is assigned to store the raw analog value coming in from the potentiometer. Because the Arduino has an analogRead resolution of 0-1023, and an analogWrite resolution of only 0-255, this raw data from the potentiometer needs to be scaled before using it to dim the LED.

In order to scale this value, use a function called map()

outputValue = map(sensorValue, 0, 1023, 0, 255);

outputValue is assigned to equal the scaled value from the potentiometer. map() accepts five arguments: The value to be mapped, the low range and high range of the raw data, and the low and high values for that data to be scaled too. In this case, the sensor data is mapped down from its original range of 0 to 1023 to 0 to 255.

The newly mapped sensor data is then outputted to the analogOutPin dimming or brightening the LED as the potentiometer is turned. Finally, both the raw and scaled sensor values are sent to the Arduino serial window in a steady stream of data.

CODE

```
// These constants won't change.  They're used to give names
// to the pins used:
const int analogInPin = A0;  // Analog input pin that the potentiometer is attached to
const int analogOutPin = 9; // Analog output pin that the LED is attached to

int sensorValue = 0;        // value read from the pot
int outputValue = 0;        // value output to the PWM (analog out)

void setup() {
  pinMode(analogInPin, INPUT);
  pinMode(analogOutPin, OUTPUT);
  Serial.begin(9600);    // begin serial communication
}

void loop() {
  // read the analog in value:
  sensorValue = analogRead(analogInPin);
  // map it to the range of the analog out:
  outputValue = map(sensorValue, 0, 1023, 0, 255);
  // change the analog out value:
  analogWrite(analogOutPin, outputValue);

  // print the results to the serial monitor:
  Serial.print("sensor = " );
  Serial.print(sensorValue);
  Serial.print("\t output = ");
  Serial.println(outputValue);

  // wait 2 milliseconds before the next loop
```

```
  // for the analog-to-digital converter to settle
  // after the last reading:
  delay(2);
}
```

USEFUL LINKS

map()
analogRead()
analogWrite()
AnalogInput
serial()

DC Motor speed control

The same setup from the previous example can be used to control the speed of a DC motor. Just replace the LED with a small DC motor and you should be able to control the motor's speed with the potentiometer. Note: Please do not run the motor for an extended period of time, as direct connection to a motor can damage some components of the Arduino (the reason why this happens will be explained during the session).

Automatic night lamp

In this exercise you will be left to design a system using the skills developed from the previous tutorials. The goal of this assignment is to create an automatic night light that turns on when the room is dark. In the previous tutorial, you got to use a potentiometer, which varies resistance based on the twisting of a knob. In this circuit, you'll be using a photo-resistor, which changes resistance based on how much light the sensor receives. The photo-resistor will output a high voltage when it is getting a lot of light and a low voltage when little or no light is present. Replace the potentiometer with a photo-resistor and use it to control your LED. Your LED should stay off initially and should only turn on when the room is dark or when the photo-resistor is covered.