St. Francis Xavier Secondary School

Arduino
Lab Manual

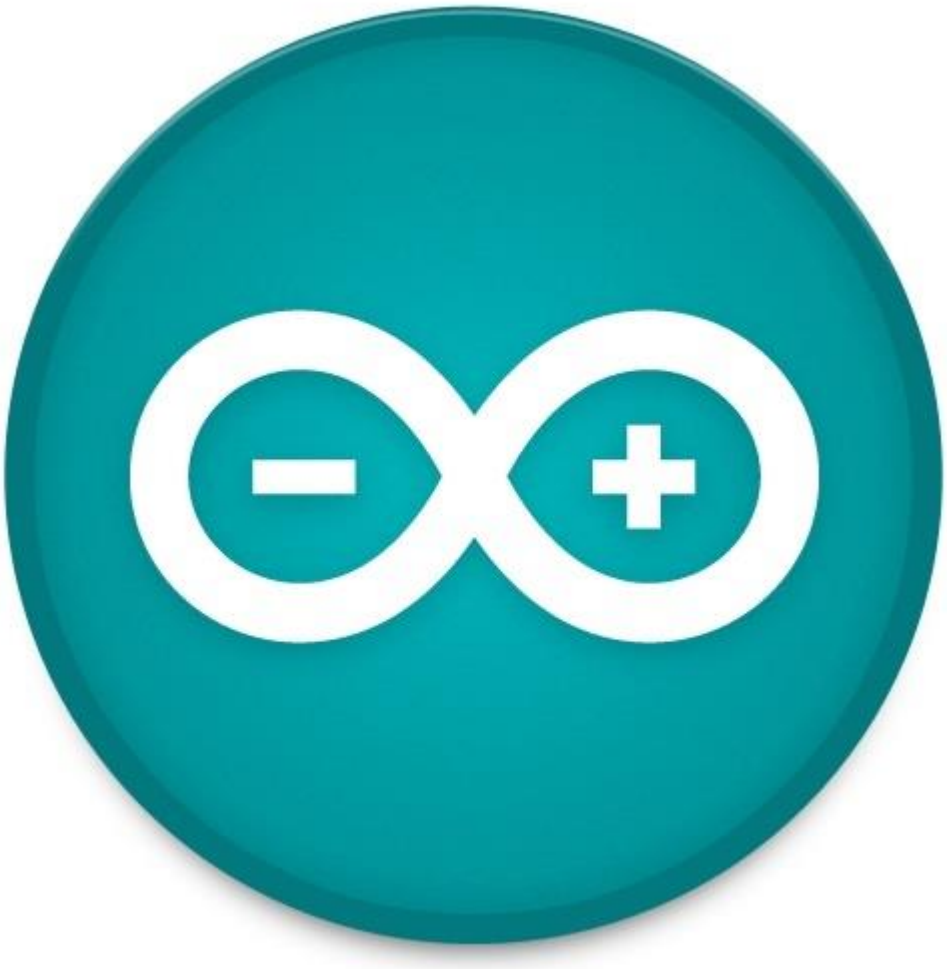
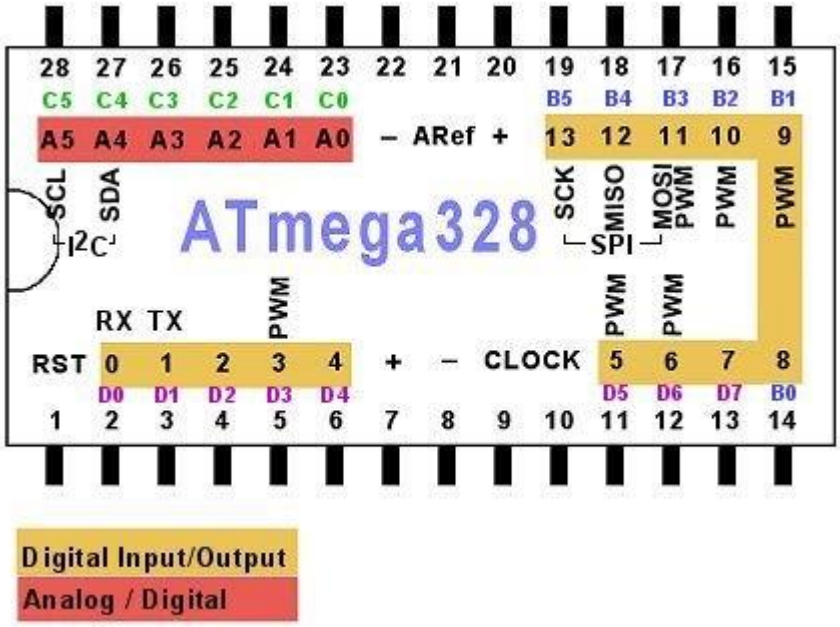


Table of Contents

PIN OUT DIAGRAM



Port B has pins **B0 to B5** Port C has pins **C0 to C5** Port D has Pins **D0 to D7**

Digital In/Out
Analog in
Buzzer/Piezo
LCD
LDR
Servo
Debug to Screen

INSTRUCTIONS

Most of these labs are taken from the Arduino website and the sketches are already included in the examples section of the Arduino's IDE.

Read the lab, load the sketch and then build the circuit. Once the program is running properly, the instructor will sign off on the lab sheet. You must then identify all the commands that were used and describe the what the circuit is doing. At the end of all the labs hand in the lab sheet to your instructor.

Make sure that the IDE is installed on the computer. Run the software. Hook up the board to the computer's serial port. Turn on the Boarduino. Using the menu at the top Load the first sketch and download it.

LAB 1: BLINK

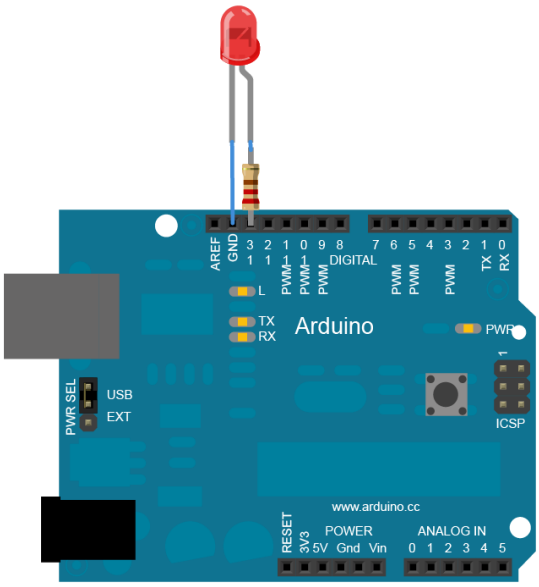
This example shows the simplest thing you can do with an Arduino to see physical output: it blinks an LED.

Hardware Required

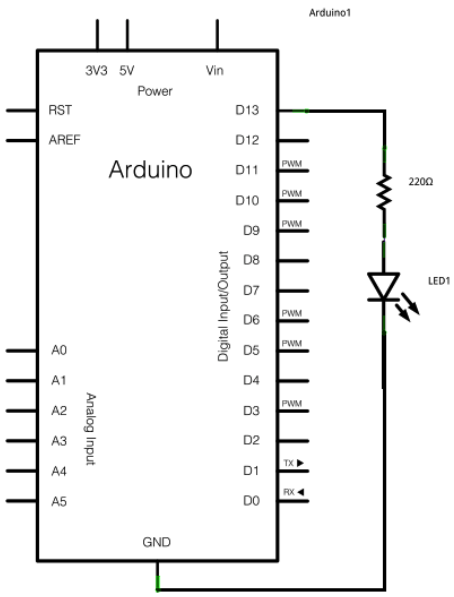
- Arduino Board
- LED

Circuit

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, and enter the code below. Most Arduino boards already have an LED attached to pin 13 on the board itself. If you run this example with no hardware attached, you should see that LED blink.



Schematic



Code

In the program below, the first thing you do is to initialize pin 13 as an output pin with the line
`pinMode(13, OUTPUT);`
In the main loop, you turn the LED on with the line:
`digitalWrite(13, HIGH);`
This supplies 5 volts to pin 13. That creates a voltage difference across the pins of the LED, and lights it up. Then you turn it off with the line:
`digitalWrite(13, LOW);`
That takes pin 13 back to 0 volts, and turns the LED off. In between the on and the off, you want enough time for a person to see the change, so the `delay()` commands tell the Arduino to do nothing for 1000 milliseconds, or one second. When you use the `delay()` command, nothing else happens for that amount of time. Once you've understood the basic examples, check out the BlinkWithoutDelay example to learn how to create a delay while doing other things.
Once you've understood this example, check out the DigitalReadSerial example to learn how read a switch connected to the Arduino.

```
/*
  Blink
  Turns on an LED on for one second, then off for one second,
repeatedly.

  This example code is in the public domain.
 */

// Pin 13 has an LED connected on most Arduino boards.
// give it a name:
int led = 13;

// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pin as an output.
  pinMode(led, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {
  digitalWrite(led, HIGH);   // turn the LED on (HIGH is the
voltage level)
  delay(1000);               // wait for a second
  digitalWrite(led, LOW);    // turn the LED off by making the
voltage LOW
  delay(1000);               // wait for a second
}
```

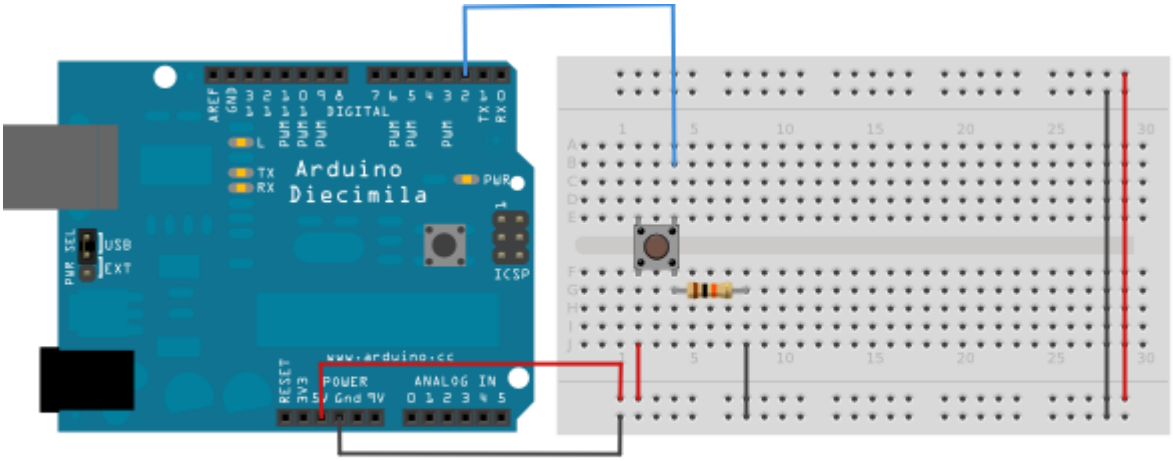
LAB 2: BUTTON

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

Hardware Required

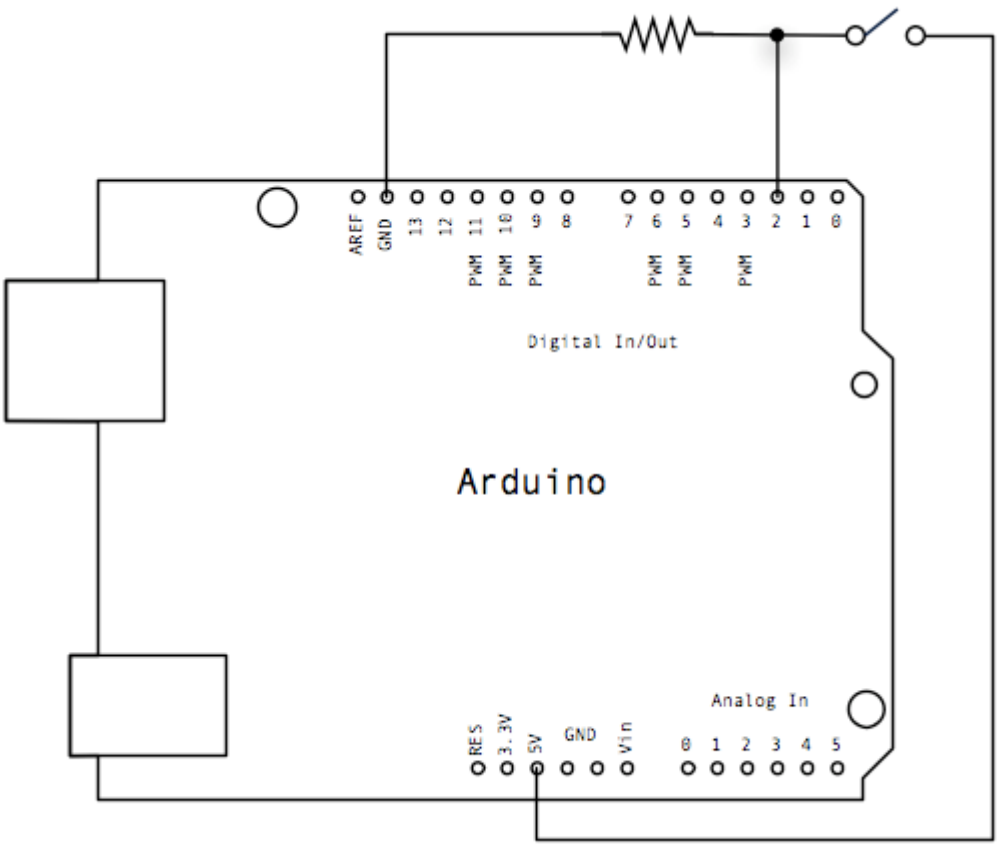
- Arduino Board
- momentary button or switch
- 10K ohm resistor
- breadboard
- hook-up wire

Circuit



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.

Schematic:



Code

```
/*
  Button

 Turns on and off a light emitting diode(LED) connected to
digital
 pin 13, when pressing a pushbutton attached to pin 2.


 The circuit:
 * LED attached from pin 13 to ground
 * pushbutton attached to pin 2 from +5V
 * 10K resistor attached to pin 2 from ground

 * Note: on most Arduinos there is already an LED on the board
 attached to pin 13.


 created 2005
 by DojoDave <http://www.0j0.org>
 modified 30 Aug 2011
 by Tom Igoe

 This example code is in the public domain.
```

```
 http://www.arduino.cc/en/Tutorial/Button
 */

// 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);
  }
}
```

LAB 2: TONE

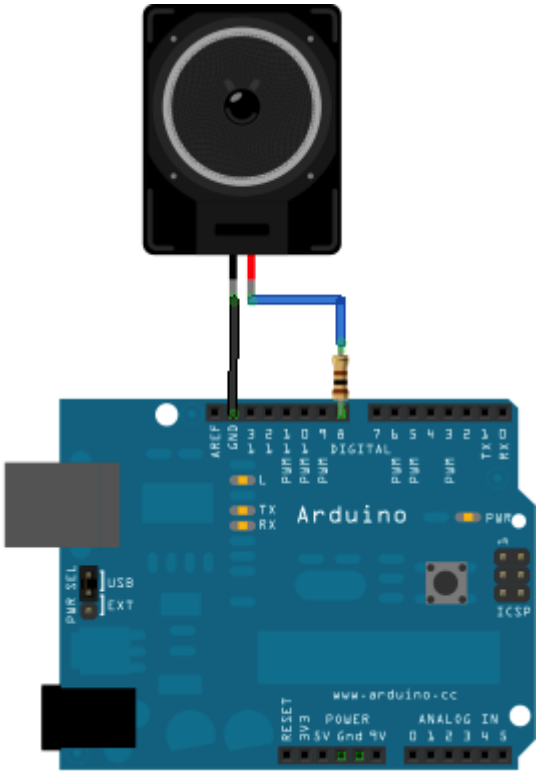
Play a Melody using the tone() function

This example shows how to use the tone() command to generate notes. It plays a little melody you may have heard before.

Hardware Required

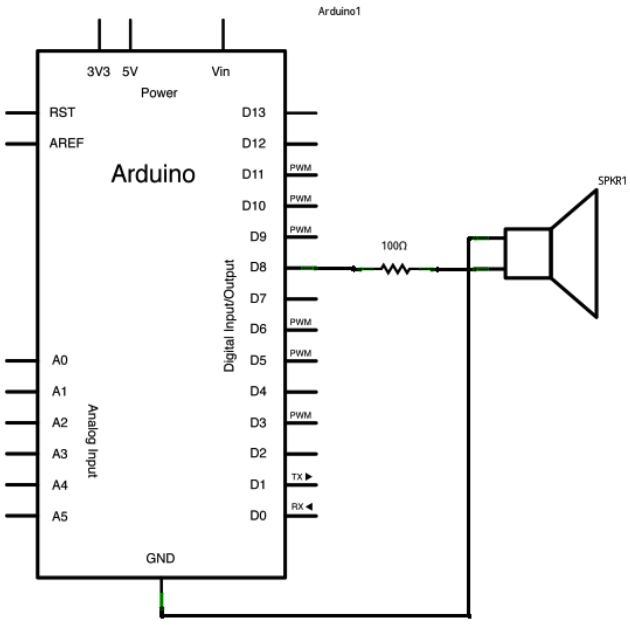
- Arduino board
- 8 ohm small speaker
- 100 ohm resistor
- hook-up wire

Circuit



Connect one terminal of your speaker to digital pin 8 through a 100 ohm resistor. Connect the other terminal to ground.

Schematic



Code

The code below uses an extra file, pitches.h. This file contains all the pitch values for typical notes. For example, NOTE_C4 is middle C. NOTE_FS4 is F sharp, and so forth. This note table was originally written by Brett Hagman, on whose work the tone() command was based. You may find it useful for whenever you want to make musical notes.
The main sketch is as follows:

```
/*
 Melody

 Plays a melody

 circuit:
 * 8-ohm speaker on digital pin 8

 created 21 Jan 2010
 modified 30 Aug 2011
 by Tom Igoe

This example code is in the public domain.

 http://arduino.cc/en/Tutorial/Tone

 */
 #include "pitches.h"

// notes in the melody:
int melody[] = {
  NOTE_C4, NOTE_G3,NOTE_G3, NOTE_A3, NOTE_G3,0, NOTE_B3, NOTE_
C4};

// note durations: 4 = quarter note, 8 = eighth note, etc.:
int noteDurations[] = {
  4, 8, 8, 4,4,4,4,4 };

void setup() {
  // iterate over the notes of the melody:
  for (int thisNote = 0; thisNote < 8; thisNote++) {

    // to calculate the note duration, take one second
    // divided by the note type.
    //e.g. quarter note = 1000 / 4, eighth note = 1000/8, etc.
    int noteDuration = 1000/noteDurations[thisNote];
    tone(8, melody[thisNote],noteDuration);

    // to distinguish the notes, set a minimum time between
them.
    // the note's duration + 30% seems to work well:
    int pauseBetweenNotes = noteDuration * 1.30;
    delay(pauseBetweenNotes);
    // stop the tone playing:
    noTone(8);
```

```
  }
}

void loop() {
  // no need to repeat the melody.
}
```

[Get Code]

To make the pitches.h file, click on the "new Tab" button in the upper right hand corner of the window. It looks like this:

The paste in the following code:

```
/*************************************************
 * Public Constants
 *************************************************/

#define NOTE_B0  31
#define NOTE_C1  33
#define NOTE_CS1 35
#define NOTE_D1  37
#define NOTE_DS1 39
#define NOTE_E1  41
#define NOTE_F1  44
#define NOTE_FS1 46
#define NOTE_G1  49
#define NOTE_GS1 52
#define NOTE_A1  55
#define NOTE_AS1 58
#define NOTE_B1  62
#define NOTE_C2  65
#define NOTE_CS2 69
#define NOTE_D2  73
#define NOTE_DS2 78
#define NOTE_E2  82
#define NOTE_F2  87
#define NOTE_FS2 93
#define NOTE_G2  98
#define NOTE_GS2 104
#define NOTE_A2  110
#define NOTE_AS2 117
#define NOTE_B2  123
#define NOTE_C3  131
#define NOTE_CS3 139
#define NOTE_D3  147
#define NOTE_DS3 156
#define NOTE_E3  165
#define NOTE_F3  175
#define NOTE_FS3 185
#define NOTE_G3  196
#define NOTE_GS3 208
#define NOTE_A3  220
#define NOTE_AS3 233
#define NOTE_B3  247
#define NOTE_C4  262
#define NOTE_CS4 277
#define NOTE_D4  294
#define NOTE_DS4 311
#define NOTE_E4  330
#define NOTE_F4  349
#define NOTE_FS4 370
#define NOTE_G4  392
#define NOTE_GS4 415
#define NOTE_A4  440
#define NOTE_AS4 466
#define NOTE_B4  494
#define NOTE_C5  523
#define NOTE_CS5 554
#define NOTE_D5  587
#define NOTE_DS5 622
#define NOTE_E5  659
#define NOTE_F5  698
#define NOTE_FS5 740
#define NOTE_G5  784
#define NOTE_GS5 831
#define NOTE_A5  880
#define NOTE_AS5 932
#define NOTE_B5  988
#define NOTE_C6  1047
#define NOTE_CS6 1109
#define NOTE_D6  1175
#define NOTE_DS6 1245
#define NOTE_E6  1319
#define NOTE_F6  1397
#define NOTE_FS6 1480
#define NOTE_G6  1568
#define NOTE_GS6 1661
```

```
#define NOTE_A6  1760
#define NOTE_AS6 1865
#define NOTE_B6  1976
#define NOTE_C7  2093
#define NOTE_CS7 2217
#define NOTE_D7  2349
#define NOTE_DS7 2489
#define NOTE_E7  2637
#define NOTE_F7  2794
#define NOTE_FS7 2960
#define NOTE_G7  3136
#define NOTE_GS7 3322
#define NOTE_A7  3520
#define NOTE_AS7 3729
#define NOTE_B7  3951
#define NOTE_C8  4186
#define NOTE_CS8 4435
#define NOTE_D8  4699
#define NOTE_DS8 4978
```

LAB 3: PITCH FOLLOWER

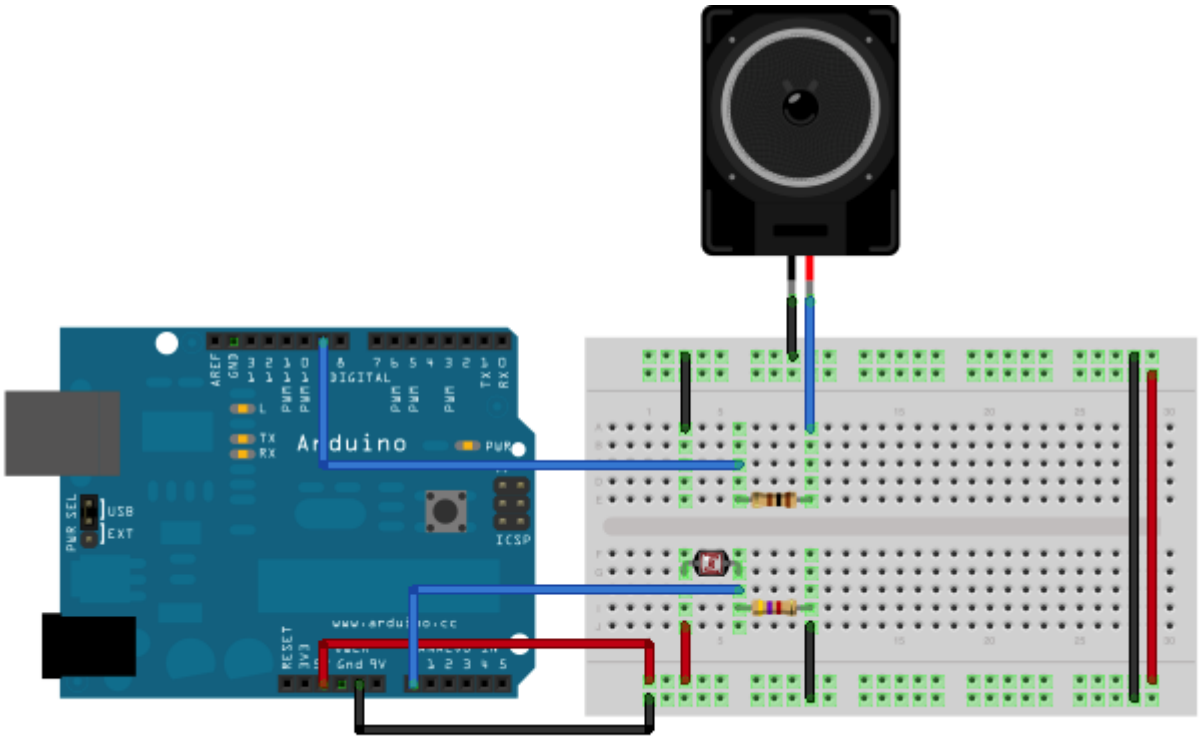
Pitch follower using the tone() function

This example shows how to use the tone() command to generate a pitch that follows the values of an analog input

Hardware Required

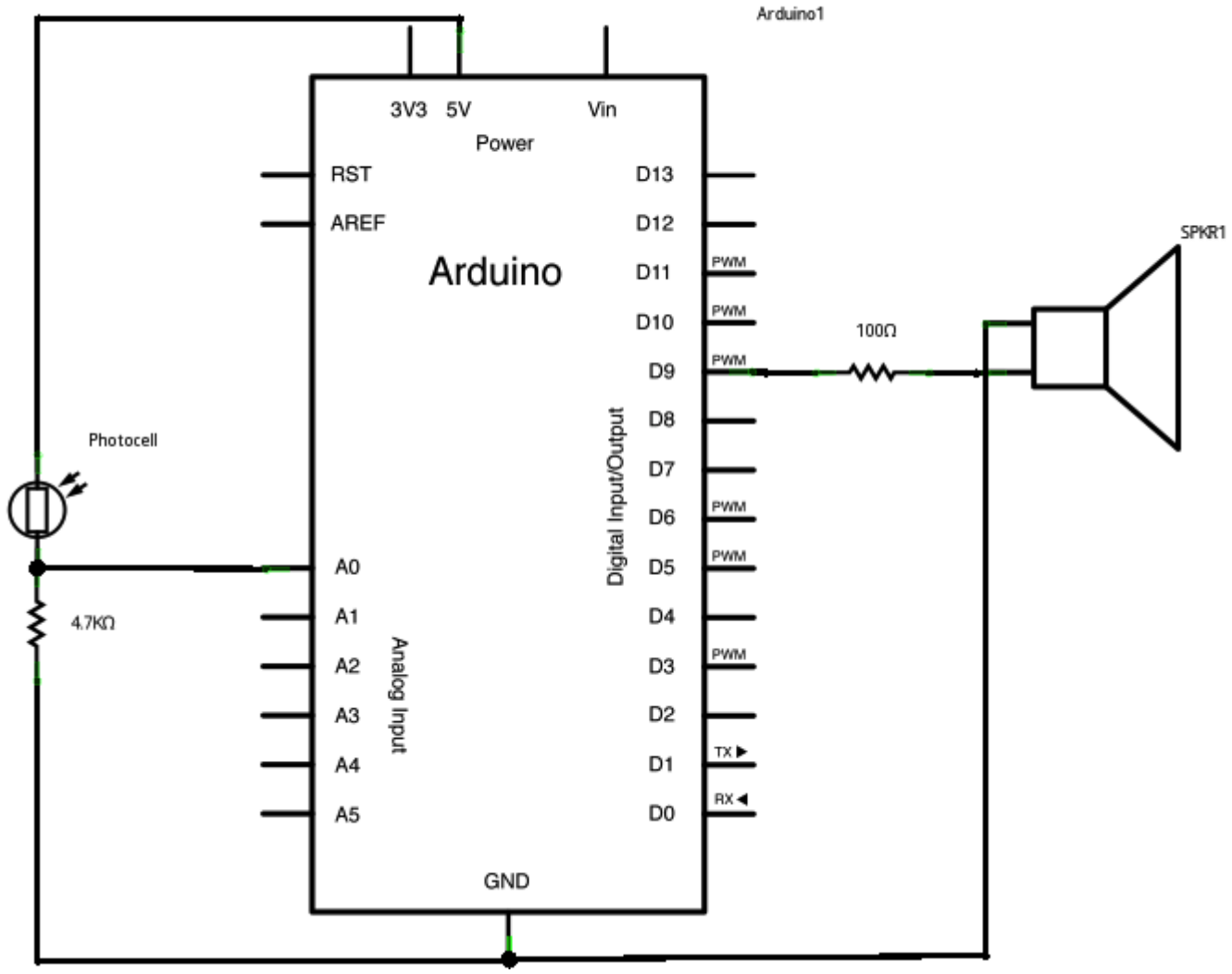
- 8-ohm speaker
- 1 photocell
- 4.7K ohm resistor
- 100 ohm resistor
- breadboard
- hook up wire

Circuit



Connect one terminal of your speaker to digital pin 9 through a 100 ohm resistor, and its other terminal to ground. Power your photoresistor with 5V, and connect it to analog 0 with the addition of a 4.7K resistor to ground.

Schematic



Code

The code for this example is very simple. Just take an analog input and map its values to a range of audible pitches. Humans can hear from 20 - 20,000Hz, but 120 - 1500 usually works pretty well for this sketch.
You'll need to get the actual range of your analog input for the mapping. In the circuit shown, the analog input value ranged from about 400 to about 1000. Change the values in the map() command to match the range for your sensor.
The sketch is as follows:

```
/*
  Pitch follower

  Plays a pitch that changes based on a changing analog input

  circuit:
```

```
 * 8-ohm speaker on digital pin 8
 * photoresistor on analog 0 to 5V
 * 4.7K resistor on analog 0 to ground

 created 21 Jan 2010
 modified 31 May 2012
 by Tom Igoe, with suggestion from Michael Flynn

This example code is in the public domain.

 http://arduino.cc/en/Tutorial/Tone2

 */


void setup() {
  // initialize serial communications (for debugging only):
  Serial.begin(9600);
}

void loop() {
  // read the sensor:
  int sensorReading = analogRead(A0);
  // print the sensor reading so you know its range
  Serial.println(sensorReading);
  // map the analog input range (in this case, 400 - 1000 from
the photoresistor)
  // to the output pitch range (120 - 1500Hz)
  // change the minimum and maximum input numbers below
  // depending on the range your sensor's giving:
  int thisPitch = map(sensorReading, 400, 1000, 120, 1500);

  // play the pitch:
  tone(9, thisPitch, 10);
  delay(1);        // delay in between reads for stability
}
```

LAB 5: FOR LOOP

Often you want to iterate over a series of pins and do something to each one. For instance, this example blinks 6 LEDs 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

Circuit

Connect six LEDS, with 220 ohm resistors in series, to digital pins 2-7 on your Arduino.
click the image to enlarge

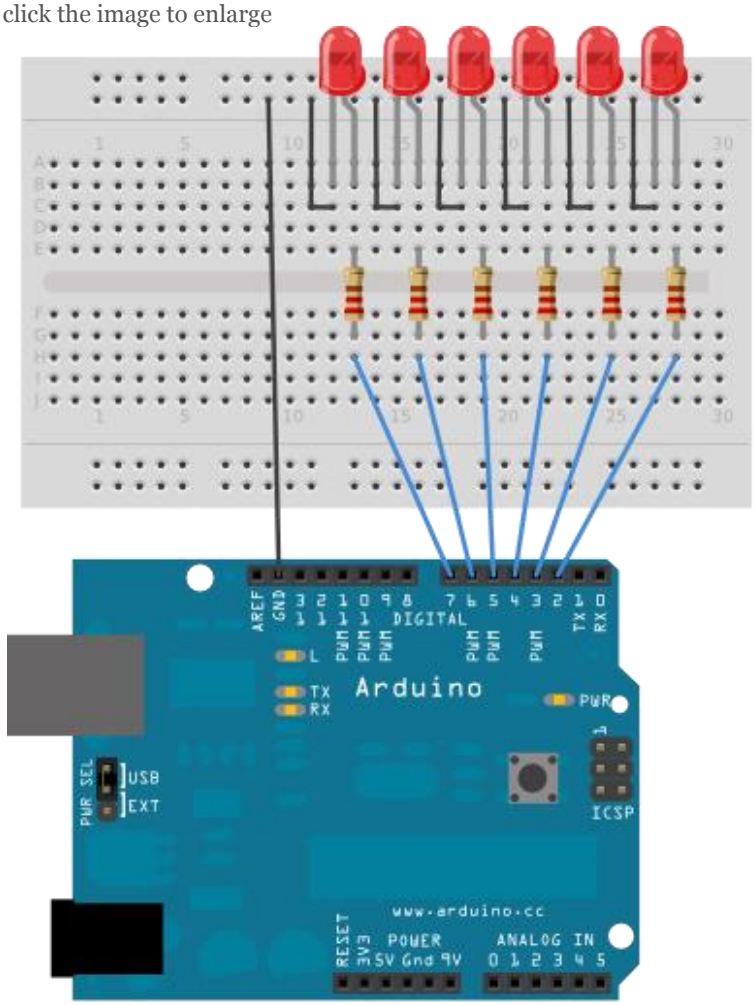
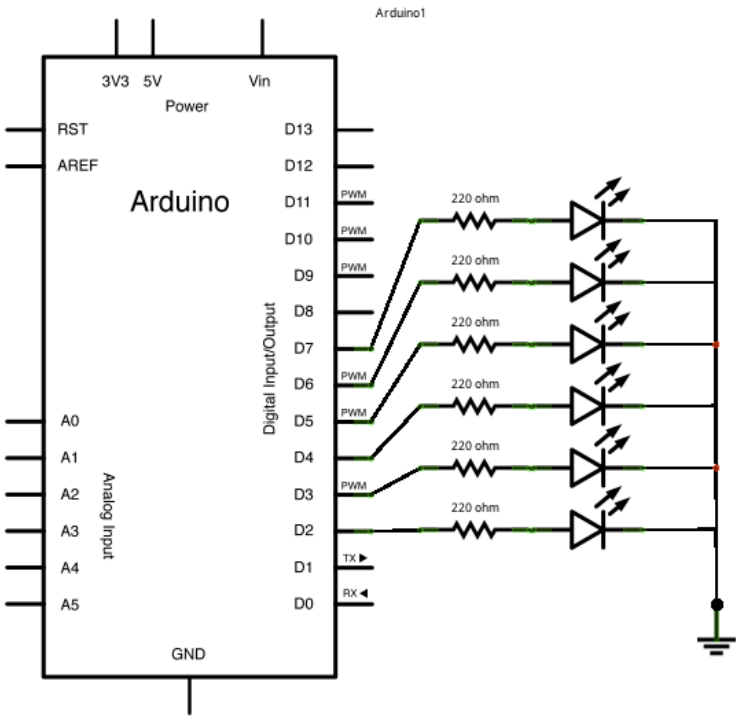


image developed using Fritzing. For more circuit examples, see the Fritzing project page

Schematic:

click the image to enlarge



Code

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.

```
/*
  For Loop Iteration

 Demonstrates the use of a for() loop.
 Lights multiple LEDs in sequence, then in reverse.

 The circuit:
 * LEDs from pins 2 through 7 to ground

 created 2006
 by David A. Mellis
 modified 30 Aug 2011
 by Tom Igoe

This example code is in the public domain.

 http://www.arduino.cc/en/Tutorial/ForLoop
 */

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);
  }
}
```

[Get Code]

See Also:

for()
digitalWrite()
delay()
WhileLoop - Use a While Loop to calibrate a sensor while a button is being pressed.
SwitchCase - Choose between a number of discrete values in a manner that is the equivalent of using multiples If statements. This example shows how to divide a sensor's range into a set of four bands and to take four different actions depending on which band the result is in.
Array: a variation on the For Loop example that demonstrates how to use an array.

LAB 6: FADING

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.

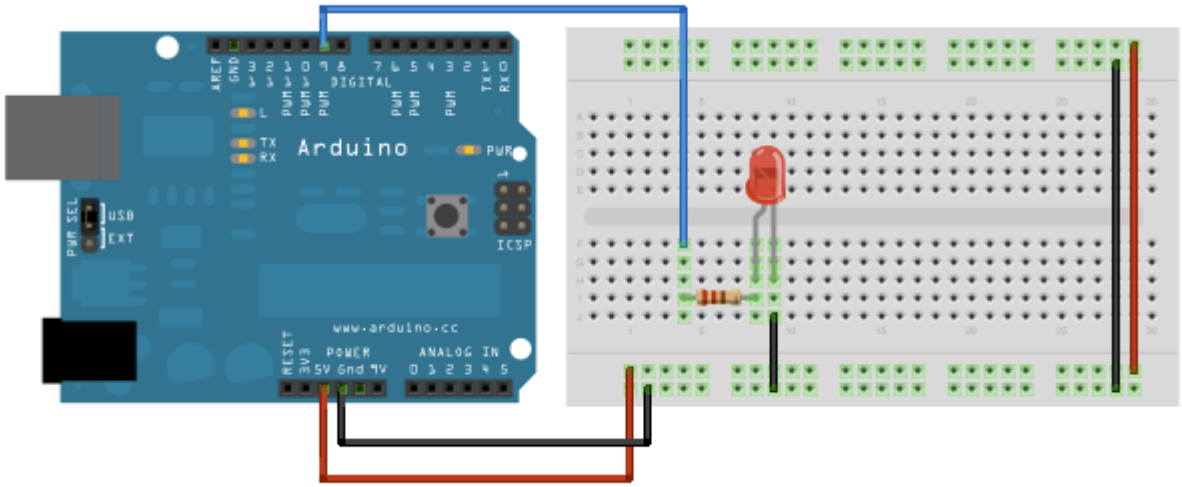
Hardware Required

- Arduino board
- Breadboard
- a LED
- a 220 ohm resistor

Circuit

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.

click the image to enlarge



Schematic

click the image to enlarge

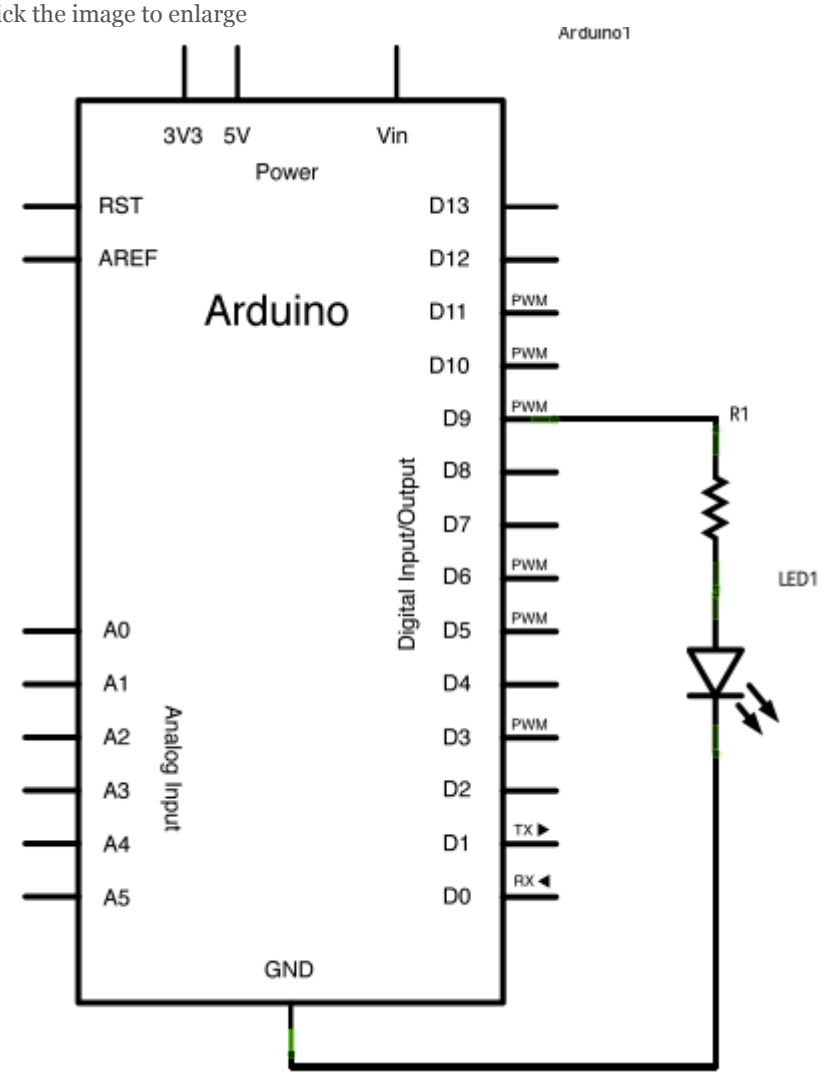


image developed using Fritzing. For more circuit examples, see the Fritzing project page

Code

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 `fadeAmount`.

If `brightness` is at either extreme of its value (either 0 or 255), then `fadeAmount` is changed to its negative. In other words, if `fadeAmount` is 5, then it is set to -5. If it's 55, then it's set to 5. The next time through the loop, this change causes`brightness` to change direction as well.

`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.

```
/*
 Fade

 This example shows how to fade an LED on pin 9
 using the analogWrite() function.

 This example code is in the public domain.
 */

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);
}
```

LAB 7: SERIAL 4 LINE LCD TUTORIAL

In this lab you will connect a 20x4 LCD (Liquid Chrystal Display) to an Arduino. This LCD has 20 characters per line and has 4 lines. The LCD display has a serial connection which make it very easy to connect to an Arduino.

Purpose:

To learn how to connect a serial LCD display to an Arduino and to program the Arduino to send text to the LCD. The final product is for the LCD to display jokes about "Engineering".

Instruction:

1. Read the Spark Fun Data Sheet on the LCD and understand how the LCD works and what commands are used to send data and control the LCD.

2. Connect the LCD to the Arduino with the red wire going to 5V(MAX!), black wire to ground and the yellow wire going to digital pin 2 (pin 4 on the Arduino).

3. Copy and paste the Sample LCD Code (see below) and upload onto the Arduino.

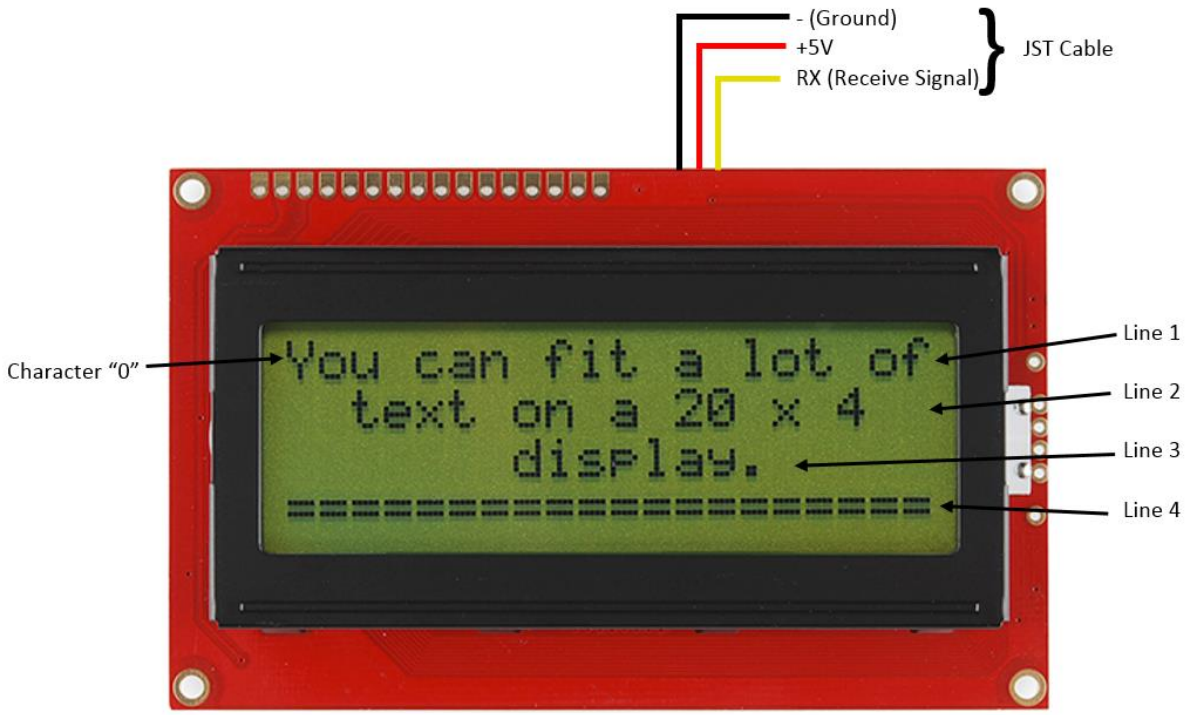
   What is observed? (Be detailed.)

4. Modify the program so that it tells 4 engineering type jokes. Make sure to use all four lines and include some of the other function available to the LCD display (eg. Brightness, blinking cursor, set cursor position, etc.).

5. Copy code onto a USB and give to your instructor.

Hardware:

Signal Name	Signal Specs.	Wire Colour (JST cable)
RX (receive)	Serial receive (input to the display). 5V TTL level, 9600 baud (default rate, can be changed), 8 bits, 1 stop, no parity.	Yellow
GND (ground)	Ground for the power supply.	Black
VDD (power)	Power supply, this should be +5V at up to 60mA if the backlight is fully on.	Red



Line	Position Number for Code	Character Number
Line 1	128 to 147*	0 to 19*
Line 2	192 to 211	64 to 83
Line 3	148 to 167	20 to 38
Line 4	212 to 231	84 tO 103
* 128 is added to the character number to get position number.		

Sample LCD Code:

```
// SparkFun Serial LCD example 2
// Format and display fake RPM and temperature data
// This sketch is for Arduino versions 1.0 and later
// If you're using an Arduino version older than 1.0, use // the other
example code available on the tutorial page.
// Use the softwareserial library to create a new "soft" serial port
// for the display. This prevents display corruption when uploading
code. #include <SoftwareSerial.h>
// Attach the serial display's RX line to digital pin 2 SoftwareSerial
mySerial(3,2); // pin 2 = TX, pin 3 = RX (unused)
void setup()
{
mySerial.begin(9600); // set up serial port for 9600 baud delay(500);
// wait for display to boot up
mySerial.write(254); // attention command mySerial.write(1); // clear
the screen command
mySerial.write(254); // attention command mySerial.write(128); // set
cursor to beginning of first line

mySerial.write("RPM:
mySerial.write(254);
```

```
mySerial.write(192);
mySerial.write("TEMP:
mySerial.write(254);
mySerial.write(148);
mySerial.write("COMPASS:
mySerial.write(254);
mySerial.write(212);
mySerial.write("POWER:
}

"); // writing RPM: and spaces at the cursor location
// attention command
// set cursor to beginning of second line
"); // write TEMP: and spaces at the cursor location
// attention command
// set cursor to beginning of third line
"); // wrie COMPASS: and spaces at the cursor location
// attention command
// set cursor to beginning of third line
"); // wrie COMPASS: and spaces at the cursor location

int temp, rpm, compass, power;
char tempstring[10], rpmstring[10], compassstring[10], powerstring[10];
// create string arrays
void loop()
{
temp = random(1000); // make some fake data rpm = random(10000);
compass = random(360);
power = random(500);
sprintf(tempstring,"%4d",rpm); // create strings from the numbers
sprintf(rpmstring,"%4d",temp); // right-justify to 4 spaces
sprintf(compassstring,"%4d",compass); //
sprintf(powerstring, "%4d", power); //
mySerial.write(254); // cursor to 9th position on first line
mySerial.write(137);
mySerial.write(rpmstring); // write out the RPM value
mySerial.write(254); // cursor to 9th position on second line
mySerial.write(201);
mySerial.write(tempstring); // write out the TEMP value
mySerial.write(254); // cursor to 9th position on second line
mySerial.write(157);
mySerial.write(compassstring); // write out the COMPASS value
mySerial.write(254); // cursor to 9th position on second line
mySerial.write(221);
mySerial.write(powerstring); // write out the POWER value
delay(1000); // short delay
}
```

LAB 8: ULTRASONIC TUTORIAL

Purpose:

To create a ultrasonic circuit that will measure distances while on top of a rotating servo motor.

Instructions:

1. Watch the two YouTube videos to learn how to work with ultrasonic sensors.

   https://www.youtube.com/watch?v=ZejQOX69K5M

   https://www.youtube.com/watch?v=PG2VhpkPqoA

2. Create a 4 wire 60mm grey cable to connect the ultrasonic sensor to your Arduino breadboard. The breadboard end of the cable will have male right angle header pins soldered to the end of the wire. Shrink wrap tubing will cover solder connection. Use a red and black permanent ink pen to mark the Vcc and ground wires. Then use masking tape to label the four wires. The ultrasonic end will have a 4 pin female header. Once soldered bend the pins 90☐ and hot glue the end around the wires and header pins.

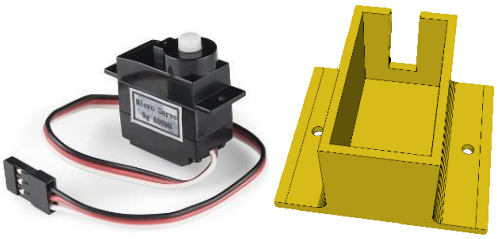
3. Connect the cable to the ultrasonic sensor and to the Arduino.

4. Load the following program onto the Arduino.

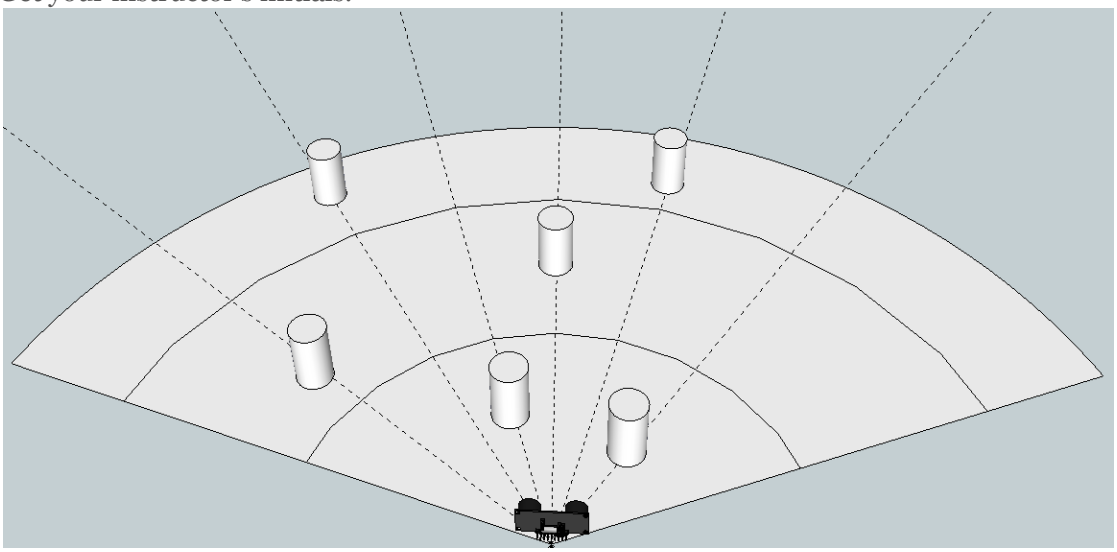
```
#define trigPin 4 // (Trigger connects to pin 4.)
#define echoPin 3 // (Echo connects to pin 3.)
void setup(){
Serial.begin (9600);
pinMode(trigPin, OUTPUT);
pinMode(echoPin, INPUT);
}
void loop(){
int duration, distance;
digitalWrite(trigPin, HIGH);
delayMicroseconds(1000);
digitalWrite(trigPin, LOW);
duration=pulseIn(echoPin, HIGH);
distance=(duration/2) / 29.2;
Serial.print(duration);
Serial.print(" ");
Serial.print(distance);
Serial.println(" cm");
delay (200);
}
```

5. Run the program and see if the program/sensor is working properly. Get the instructor's initials.

6. Using SketchUp design a box that would hold a servo upright (shaft at the top). Keep in mind that there is a lead at the bottom of the servo that must have access to. The inside dimensions should be 21mm x 41mm. The walls should be 2mm thick. There should also be a way to secure the box to a base (holes for screws). Give your design to your teacher on a flash drive.



7. Now that you have your ultrasonic sensor attached to your servo, modify your program so that the sensor will scan across 90° and read back what it scans. The sensor will be placed on a board where it will need to detect obstacles paced at different distances on the board. Get your instructor's initials.



LAB 9: SERVO - KNOB

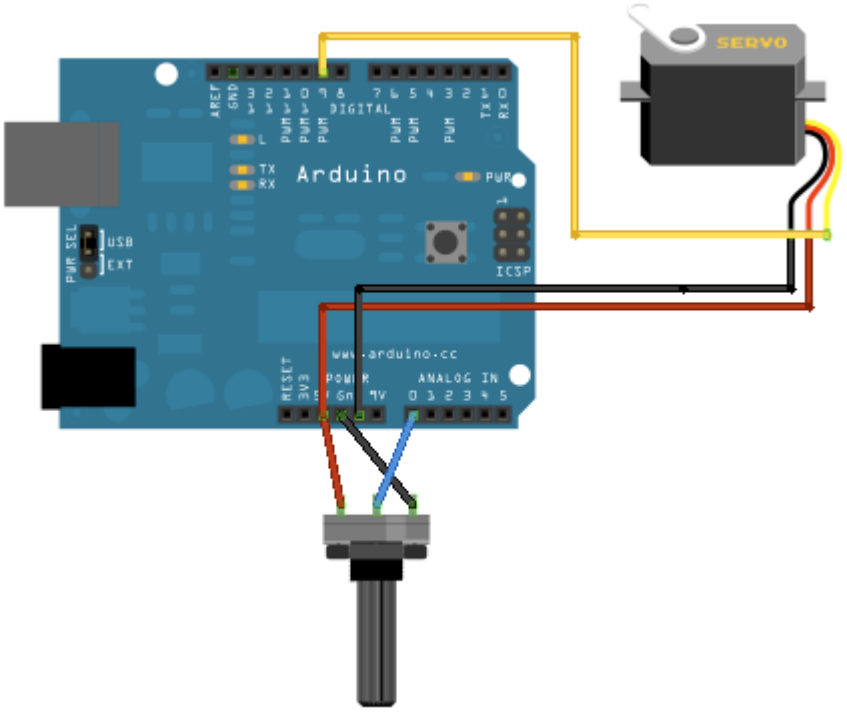
Control the position of a RC (hobby) servo motor with your Arduino and a potentiometer. This example makes use of the Arduino **servo library**.

Hardware Required

- Arduino Board
- (1) Servo Motor
- (1) Potentiometer
- hook-up wire

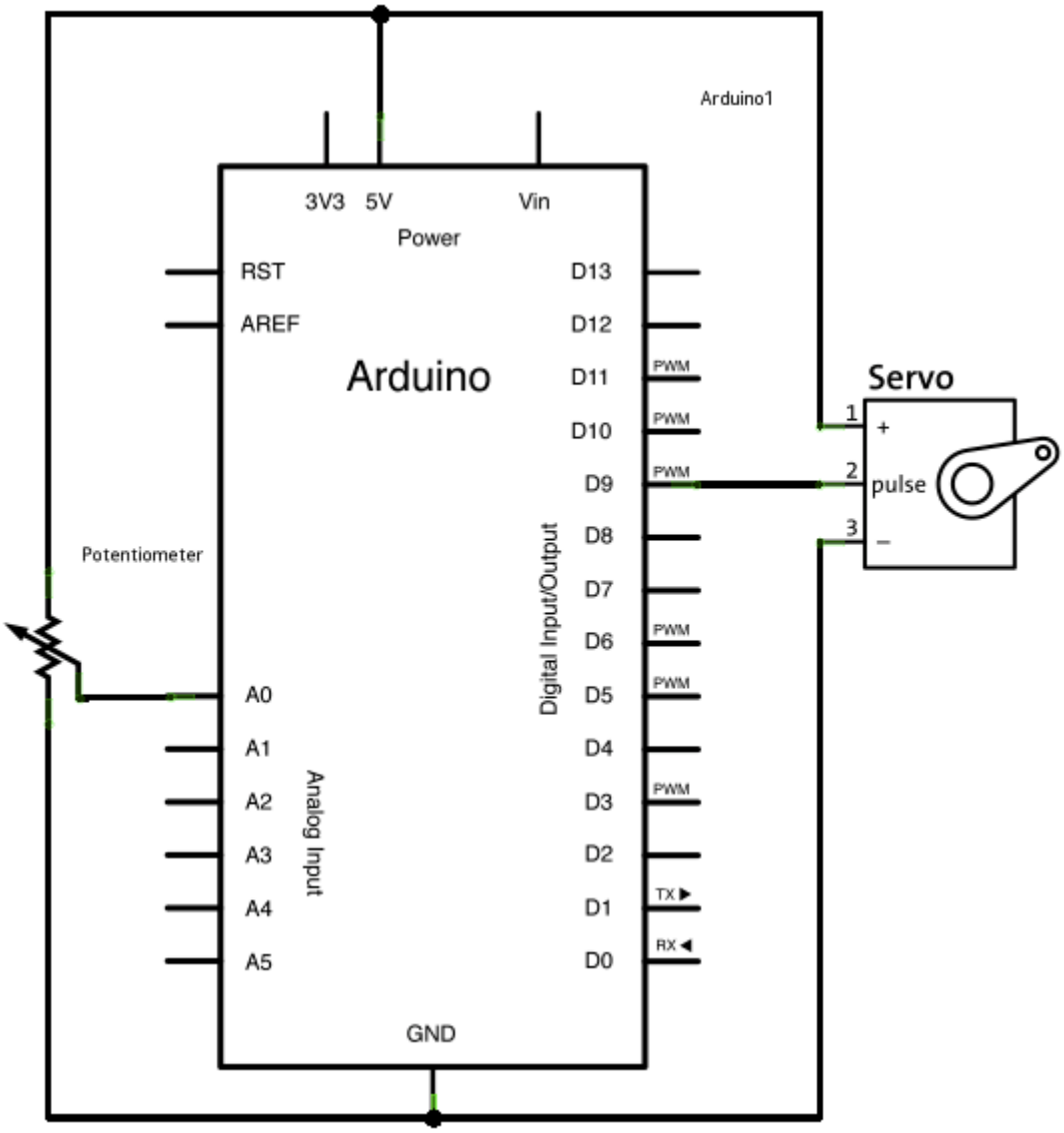
Circuit

Servo motors have three wires: power, ground, and signal. The power wire is typically red, and should be connected to the 5V pin on the Arduino board. The ground wire is typically black or brown and should be connected to a ground pin on the Arduino board. The signal pin is typically yellow or orange and should be connected to pin 9 on the Arduino board.
The potentiometer should be wired so that its two outer pins are connected to power (+5V) and ground, and its middle pin is connected to analog input 0 on the Arduino.
click the images to enlarge



images developed using Fritzing. For more circuit examples, see the Fritzing project page

Schematic



Code

```
// Controlling a servo position using a potentiometer
(variable resistor)
// by Michal Rinott <http://people.interaction-
ivrea.it/m.rinott>

#include <Servo.h>

Servo myservo;  // create servo object to control a servo

int potpin = 0;  // analog pin used to connect the
potentiometer
int val;    // variable to read the value from the analog pin

void setup()
```

```
{
  myservo.attach(9);  // attaches the servo on pin 9 to the
servo object
}

void loop()
{
  val = analogRead(potpin);            // reads the value of
the potentiometer (value between 0 and 1023)
  val = map(val, 0, 1023, 0, 179);     // scale it to use it
with the servo (value between 0 and 180)
  myservo.write(val);                  // sets the servo
position according to the scaled value
  delay(15);                           // waits for the servo
to get there
}
```