

# ITP Physical Computing

## LAB: TONE OUTPUT USING AN ARDUINO

Last edited 31 Aug 2014 by Tom Igoe

### Introduction

This lab is an introduction to generating simple tones on an Arduino. In order to make the most of this lab, you should understand the basics of [how to program digital input and output on an Arduino](#), and how to read a [simple circuit diagram](#).

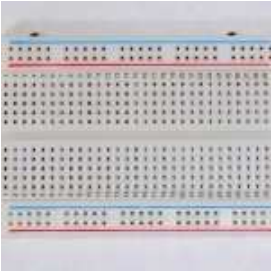



### Video

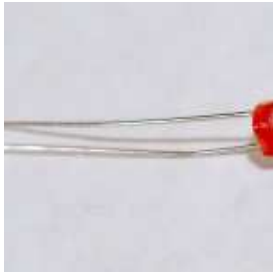
#### Contents [\[hide\]](#)

- 1 Introduction
- 2 What You'll Need
- 3 Why not use AnalogOut()?
- 4 Prepare the breadboard
- 5 Connect the sensors and the speaker
- 6 Check the sensor input range
- 7 Play Tones
- 8 A more complex example
- 9 A Musical Instrument
- 10 Program it
- 11 Get Creative

### What You'll Need

For this lab you'll need:

			
Solderless breadboard	22-AWG hookup wire	Arduino Microcontroller	100 Ohm resistors



*photocell (or a  
different form of  
variable resistor)*

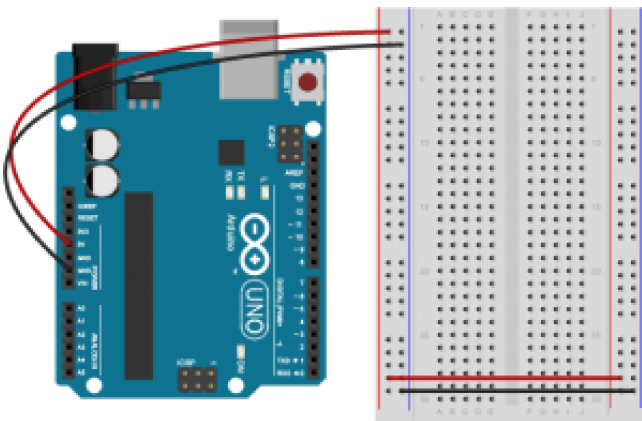
*8-ohm speaker*

## Why not use AnalogOut()?

When you use `analogOut()` to create pulsewidth modulation (PWM) on an output pin, you can change the on-off ratio of the output (also known as the *duty cycle*) but not the frequency. If you have a speaker connected to an output pin running `analogOut()`, you'll get a changing loudness, but a constant tone. To change the tone, you need to change the frequency. The `tone()` command does this for you.

## Prepare the breadboard

Connect power and ground on the breadboard to power and ground from the microcontroller. On the Arduino module, use the 5V and any of the ground connections:

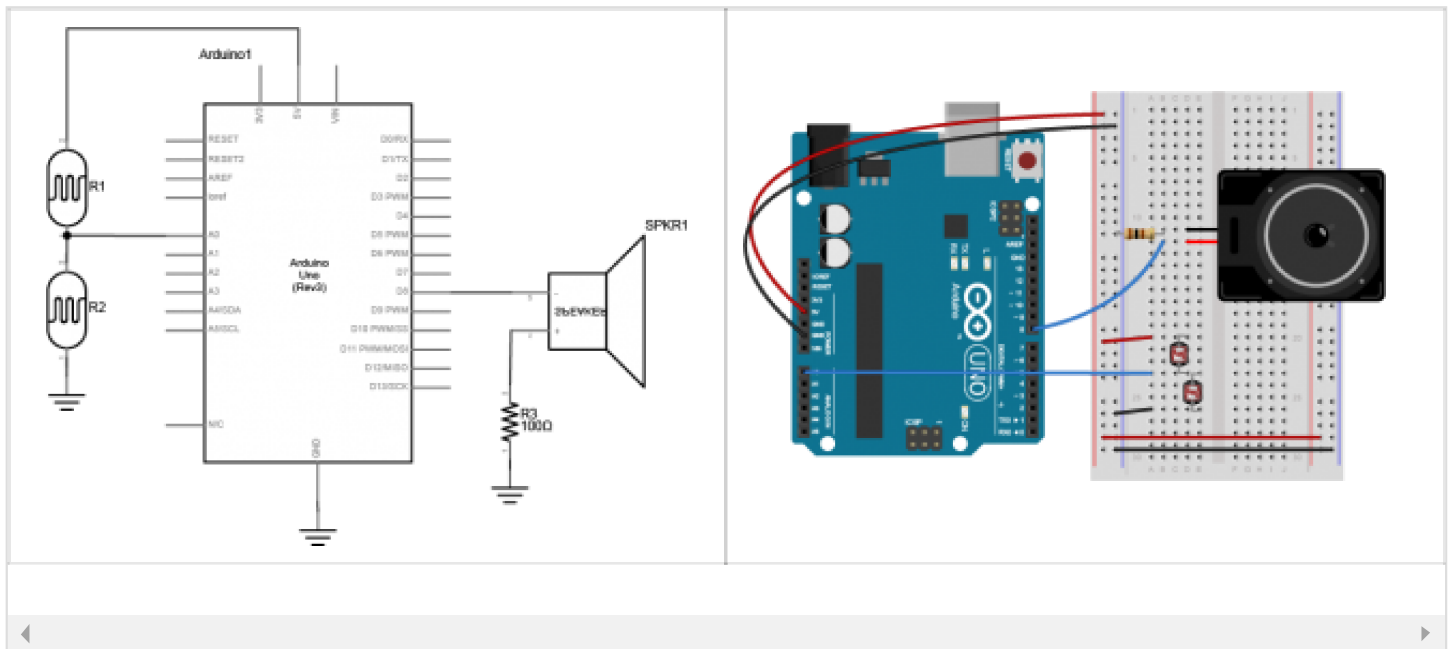


Made with *Fritzing*

## Connect the sensors and the speaker

Connect two photoresistors to analog pin 0 in a voltage divider circuit as shown below. The 8-

ohm speaker connects to pin 8 of the Arduino. You can use any digital I/O pin if you don't like 8. The other end of the speaker connects to ground.



**NOTE:** this sensor circuit is not the normal way of connecting an **analog input**. There is no fixed resistor. The two photocells act as a voltage divider together, so you can change the value of the analog in by covering either one. if you are using variable resistors that can both go to 0 ohms, you should connect a fixed resistor in series from the junction of the two resistors to the input, to avoid a short.

## Check the sensor input range

Once you've got the circuit connected, check the range of the analog input and note the highest and lowest values you can reach.

```
void setup() {
  Serial.begin(9600);           // initialize serial communications
}

void loop()
{
  int analogValue = analogRead(A0); // read the analog input
  Serial.println(analogValue);      // print it
}
```

## Play Tones

Write a sketch to read the analog input and map the result to a range from 100 to 1000. Store the

result in a local variable called frequency. This will be the frequency you play on the speaker. Then use the `tone()` command to set the frequency of the speaker on pin 8.

```
void setup() {  
  // nothing to do here  
}  
  
void loop() {  
  // get a sensor reading:  
  int sensorReading = analogRead(A0);  
  // map the results from the sensor reading's range  
  // to the desired pitch range:  
  float frequency = map(sensorReading, 200, 900, 100, 1000);  
  // change the pitch, play for 10 ms:  
  tone(8, frequency, 10);  
}
```

Once you've uploaded this, move your hands over the photocells, and listen to the frequency change. It will change from 100 Hz to 1000 HZ, because that's what you set in the `map()` command. If you want to change the frequency range, change those two numbers. See if you can get it to play a little tune.

## A more complex example

The `pitches.h` file includes constants that give you the pitches for a standard western scale. To include them in your sketch, click the New Tab button on the toolbar and create a new tab called `pitches.h`. Then copy the file below and paste it in the new tab in your sketch.



[Click here to see pitches.h](#)

For this sketch, you'll play a simple melody. A melody consists of notes played in a sequence, and rests between the note. Each note and rest has its own particular duration. You're going to play a seven note sequence, as follows (in C Major):



image from seventhstring.com

or:

C4, G3, G3, G#3, G3, rest, B3, C4

the durations are:

quarter note, eighth note, eighth note, quarter note, quarter note, quarter rest, quarter note, quarter note.

Start your sketch with two global variables. Using `pitches.h`, make an array variable holding those notes. Make a second array to hold the note durations, marking quarter notes as 4 and eighth notes as 8.

```
// notes in the melody:
int melody[] = {
  NOTE_C4, NOTE_G3, NOTE_G3, NOTE_G#3, NOTE_G3, NOTE_B3, NOTE_C4};

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

How can you use the durations to play notes? Well, imagine that a quarter note is one quarter of a second, and eighth note is one eighth of a second, and so forth. In that case, the actual duration for each note is 1000 milliseconds divided by the value for it in the durations array. For example, the first note is  $1000/4$ , the second is  $1000/8$ , and so forth.

Now, you only want to play the song once, so everything will happen in the `setup()`. Make a for loop in the setup to iterate over the seven notes. For each time through the loop, use `tone()` to play the next note in the array. Use the formula in the last paragraph to determine how long each note should play for. After you start the note, delay for as long as the note plays, plus 30

milliseconds or so, to separate each note.

```
#include "pitches.h"

// notes in the melody:
int melody[] = {
  NOTE_C4, NOTE_G3, NOTE_G3, NOTE_G3, 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);

    //pause for the note's duration plus 30 ms:
    delay(noteDuration +30);
  }
}

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

That's the whole tune!

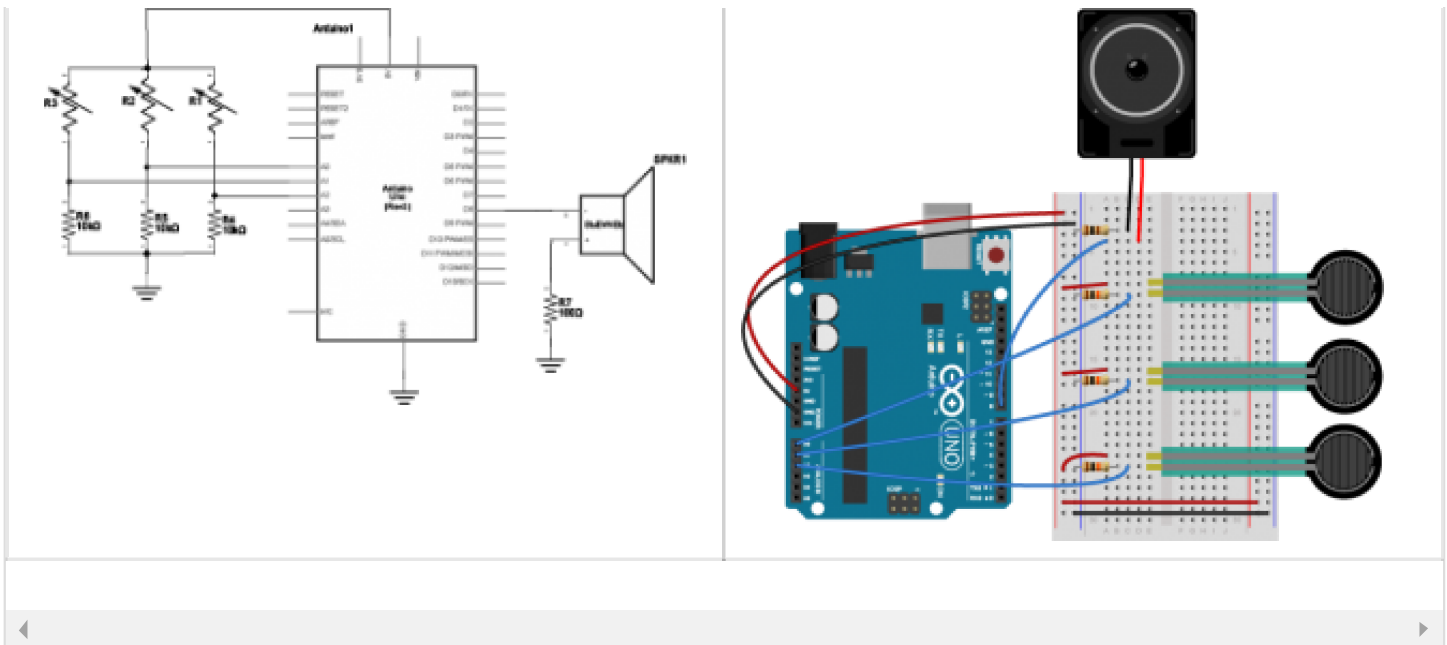
## A Musical Instrument

Playing a tune like you just doesn't allow for much user interaction, so you might want to build more of a musical instrument.

Here's an example of how to use the note constants to make a simple keyboard:

**The circuit:**





## Program it

Make a sketch that plays a note on each sensor when the sensor is above a given threshold.

Start with a few constants: one for the threshold, one for the speaker pin number, and one for the duration of each tone. To make this instrument work, you need to know what note corresponds to each sensor. Include `pitches.h` as you did above, then make an array that contains three notes, A4, B4, and C3 as well. Set all of this up at the beginning of your sketch, before `setup()`.

```
#include "pitches.h"

const int threshold = 10;      // minimum reading of the sensors that general
const int speakerPin = 8;      // pin number for the speaker
const int noteDuration = 20;   // play notes for 20 ms

// notes to play, corresponding to the 3 sensors:
int notes[] = {
  NOTE_A4, NOTE_B4, NOTE_C3 };

```

You don't need anything in your `setup()`, but in your loop, you need a for loop that iterates over the first three analog inputs, 0, 1, and 2. For each one, read the sensor, and if it's above a threshold, play the corresponding note in the notes array for the note duration.

```
#include "pitches.h"

const int threshold = 10;      // minimum reading of the sensors that general
const int speakerPin = 8;      // pin number for the speaker
const int noteDuration = 20;   // play notes for 20 ms

```

```
// notes to play, corresponding to the 3 sensors:
int notes[] = {
  NOTE_A4, NOTE_B4, NOTE_C3 };

void setup() {

}

void loop() {
  for (int thisSensor = 0; thisSensor < 3; thisSensor++) {

    // get a sensor reading:
    int sensorReading = analogRead(thisSensor);

    // if the sensor is pressed hard enough:
    if (sensorReading > threshold) {

      // play the note corresponding to this sensor:
      tone(speakerPin, notes[thisSensor], 20);
    }
  }
}
```

When you upload this sketch, you should have a three-key keyboard.

## Get Creative

*This is just a suggestion for a short project. It's not a requirement for the class homework.*

Now that you've got the basics, make a musical instrument. Consider a few things in designing your instrument:

- Do you want to play discrete notes (like a piano), or sliding pitches (like a theremin)? How do you program to achieve these effects?
- Do you want to control the tempo and duration of a note?
- Do you want the same physical action to set both the pitch and the velocity (volume) of a note?
- Do you want to be able to play more than one note at a time (e.g. chords)?

All of these questions, and many more, will affect what sensors you use, how you read them, and how you design both the physical interface and the software.