# ECE 1001/1002 Introduction to Robotics
# Lab #3: Musical System

## Objectives

Introduction multiple inputs and sound output

## Requirements and Signoffs

Lab #2 introduced inputs which the Arduino used to make an action. This lab expands on that with switches for input and sound for output.

### Sound Output Circuit

The kit includes a simple speaker (a buzzer, actually, but it can produce acceptable quality sounds) which looks like this:


Buzzer (speaker)
sparkfun

Buzzer (bottom view)
sparkfun

If you turn the speaker over, you will find one pin is marked with a "+" and one pin is marked with a "-" The speaker is polarized, and it won't work if you connect to it incorrectly! Not only that, but it's hard to see where the pins are as you plug it in. Be careful when wiring it that you attach wires to the correct pins, and that you know which rows in the breadboard you insert the speaker in.

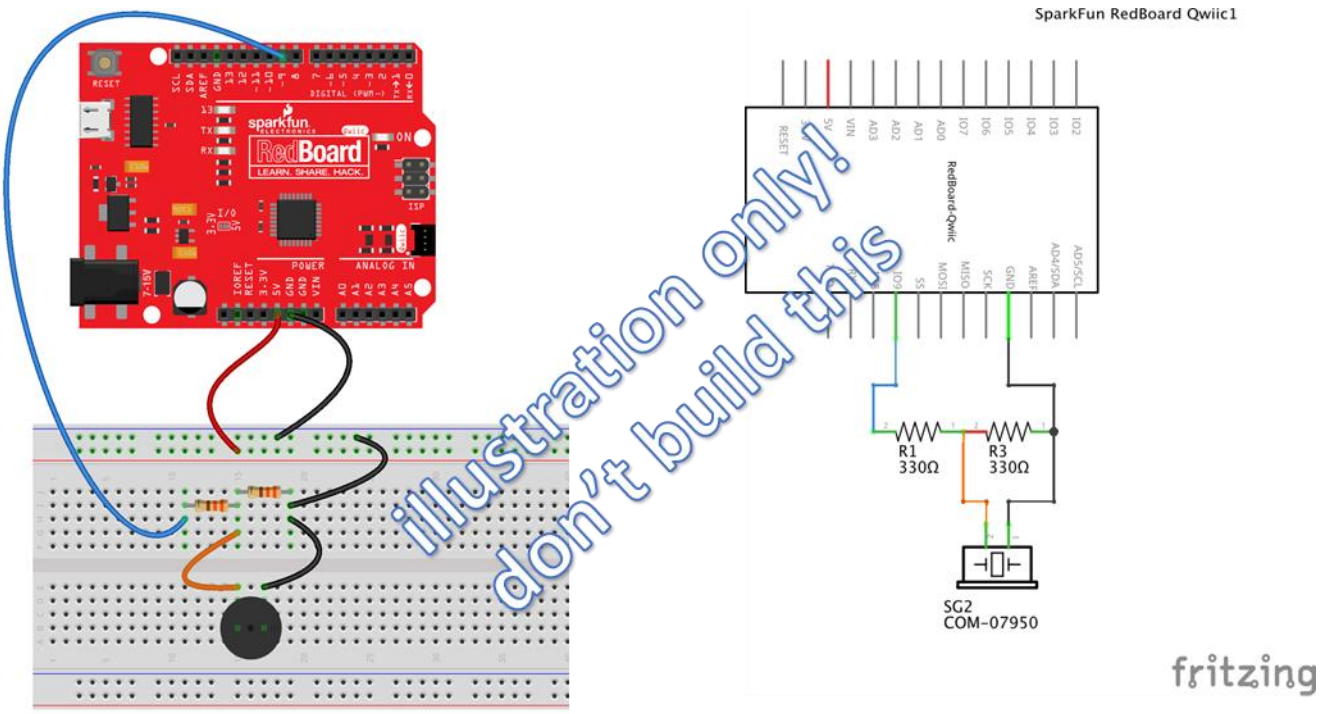
The speaker can play sound by using a special Arduino command made for this purpose: tone(); more on that later in this lab. The speaker must be connected to certain Arduino pins, they are marked with a ~ character: only pins 3, 5, 6, 9, 10 or 11 can output tones. These pins have the ability to make "pulse width modulation" (PWM) which is a simple way for digital circuits like the Arduino to make smooth continuous outputs. More on that in later labs too.

The speaker will play at full volume, but there is a way to reduce the volume using resistors. The volume on the speaker is related to the voltage it receives. If we build a voltage divider similar to the photoresistor circuit in lab 2[1], the voltage on the speaker can be reduced, making the sound less annoying to those near you.

That could look like the circuit below (but don't build this! We can do better) where we have pin 9 connected to a 330Ω resistor, which then connects to another 330Ω resistor. The speaker would receive half the voltage from pin 9, that would make it less loud (but not necessarily half as loud). The problem with this circuit is it's hard to adjust the volume. We could change the size of these resistors, making the two different values to get a different volume level. But it's a pain to find the right size resistor and change it. And if you

[1] You *could* use the photoresistor-divider to control the speaker volume, but it might be frustrating to actually control, as you now have a non-linear thing (how your ear senses sound volume) controlled by a very non-linear resistor divider. Feel free to try this outside of lab-time! Note, it may work better (but not well) if the photoresistor is on the ground side of the divider.

neighbor says it's still too loud, you'll have to find a different resistor. Besides that, your kit only has two sizes of resistors!



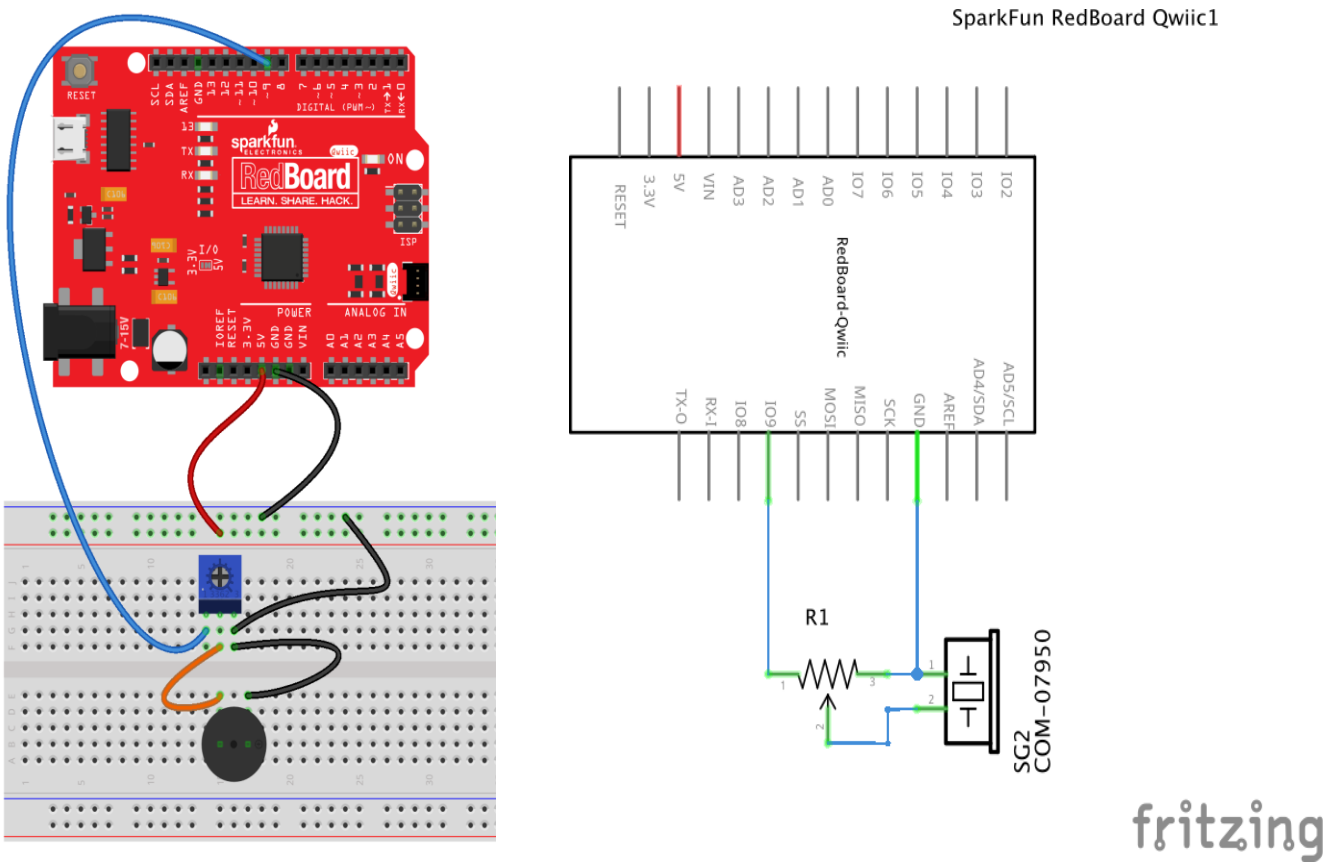

Adjusting the volume of a speaker is a common thing to do, of course. Sometimes volume is still controlled with a "volume knob." One easy (and cheap) way to make a volume knob is with a pair of variable resistors in a device called a "potentiometer" or usually just called a "pot." Your kit has one, it is the little blue knob with pins:

sparkfun

Inside this pot are two resistors, which change in value by a of ratio knob position, in a way that the total resistance between the two outer pins doesn't change: as the resistance between pins 1 and 2 goes up, the resistance between pins 2 and 3 goes down; the total resistance between pins 1 and 3 is constant. We can replace the two resistors in the circuit above with a single pot, and have a variable volume knob to adjust speaker loudness.

The pot is not polarized, but there are three pins, and the connections do matter. You must connect one of the outer pins (1 or 3) to the higher or + voltage, and the other pin (3 or 1) to ground. The middle pin, pin 2 should be connected to the + pin on the speaker, and the – pin on the speaker should be connected to the ground connection. The way the breadboard image is shown below uses more wires than might be necessary, just for clarity. If you can connect the speaker and make it work with fewer wires, that is good.

Build the speaker circuit as shown below.




## Fixed Music Program

The circuit above just needs a program to play music now. As mentioned above, the Arduino IDE has a built in "function" (a function is something we'll get to in a few labs) for making sounds. Like light, sounds are a common and easy way for computers to communicate with people. The Arduino hardware can't make pure sine waves, but it can make square waves which sound not all that bad. The command is:

*tone(pin, frequency, duration)*

"pin" is the pin number the speaker is attached to, "frequency" is the frequency of the tone desired, and "duration" is the amount of time to play the tone, in milliseconds. Once a tone is started, the Arduino can do something else while the tone finishes playing. That can be useful, unless you try to play a different tone before the previous note is finished. If you try to play two notes at the same time, awful sounds will result. One safe way to play a tone is to include a delay for the same length of time as the duration of the tone after the tone starts. This is shown in the lab program.

What frequency to play? The basic notes correspond to these frequencies:

| Musical Note | Frequency (Hz) |
|---|---|
| A | 220 |
| B | 247 |
| C | 261 |
| D | 294 |
| E | 330 |
| F | 349 |
| G | 392 |

You can find other note's frequencies with some searching. To play a C for 250 ms (¼ second) the command is: tone(9, 261, 250) for example.

Open the program Lab3_note_run.ino and look at it. It's just a series of tone commands to play notes in order. Each tone is followed by a delay(); to allow the tone to reach the duration time. Compile and upload it to your Arduino. Adjust the volume, and be nice to your neighbors—unplug the Arduino as soon as you understand the volume control and what the hardware/software is doing. Feel free to change the notes, add notes, and change the duration. Keep the volume low!!

You can understand how you can synthesize music with a computer. The Arduino is limited in musical abilities, however. Not only does it substitute square waves for sine waves, but it has only one "voice" which means it can play only one note at a time[2].

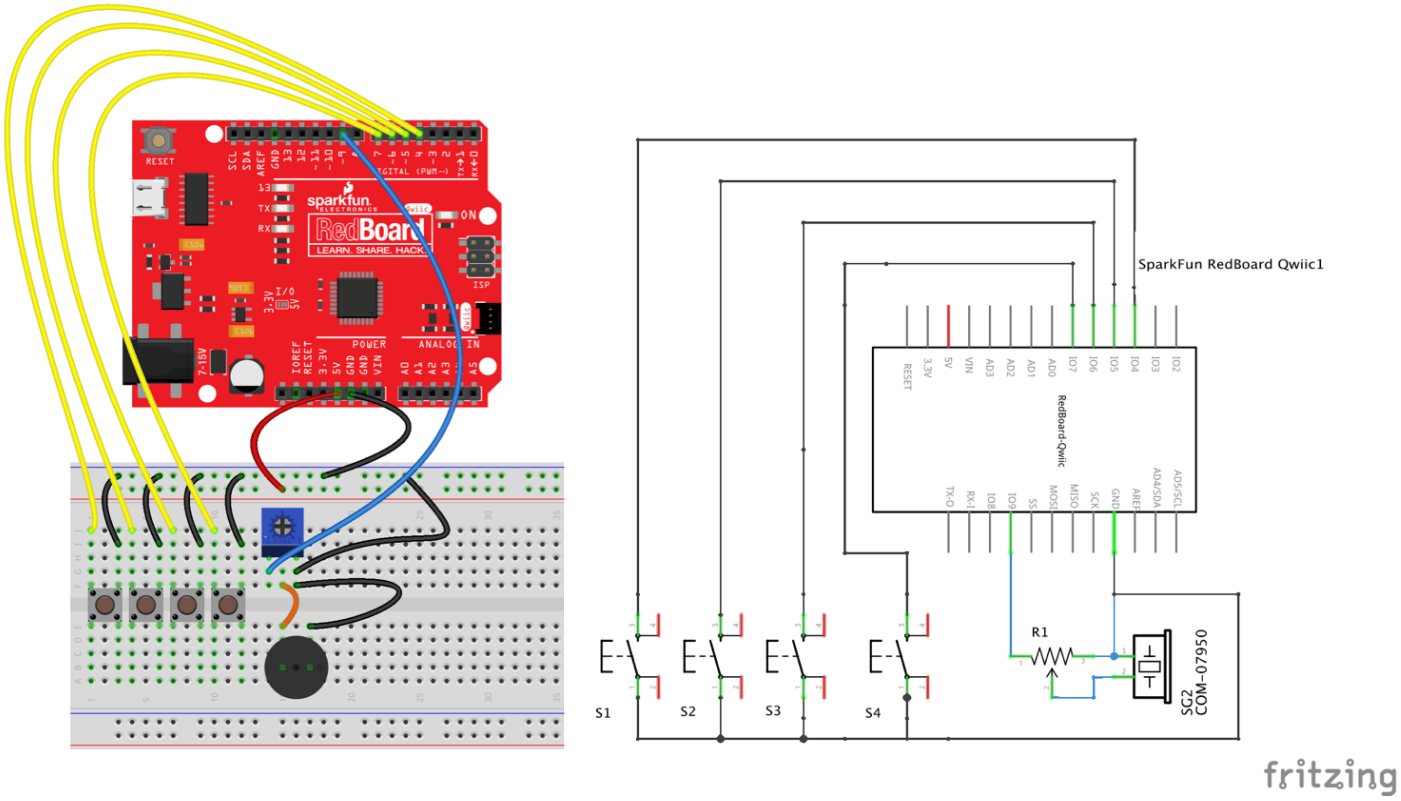
## Musical Flexibility: Four-Key Instrument

With buttons, the Arduino can become a 4-note instrument you can play. Your kit comes with four buttons: red, yellow, green and blue. They look like this:


The two pins opposite each other are always connected. The pins on the same side are connected while the button is pushed down. Never put the switch in the breadboard so that the two pins on the same side are in the same row; everything is always connected that way. Instead, orient the switch so that the pins on the same side are in different breadboard rows. See the picture below, and ask for help if needed.

[2] This same limitation existed on the Apple II computer and IBM PC, both with considerably slower processors than the Arduino, by the way. Although you can make up for the lack of multiple notes at once (voices) in software, as the Apple II did, but modern hardware provides much greater musical capability.


If each of the four buttons is connected to an input, the Arduino can monitor the four inputs and play a different note while one of the buttons is pushed. Add the buttons to your breadboard, re-arrange as you please and wire up as below:




## Four-Key Instrument Software

Open the lab3 four-key instrument program Lab3_Four_Key_Instrument.ino, and look at it.

The buttons are connected to I/O pins 4, 5, 6, 7. These will be inputs. The other end of the buttons are all attached to ground. When a button is pushed, ground is connected to the I/O pin, and the Arduino will read a Low signal. When the button is released, it needs to present a High signal to the Arduino (without making a direct short to ground). This is such a common situation that the Arduino can be configured internally to present the High signal when a button is released.

In the setup area, the pins need to be configured to have an "internal pullup resistor." For example, to configure pin 4 as this kind of input, use this in the setup:

*pinMode(4, INPUT_PULLUP);*

This setup must be repeated for each of the button input pins.

The loop () now has a series of: if ()…else if ()_…else statements. The Arduino will check the buttons in order and will execute the first one it comes to which is true. If none of the conditions are true (which happens when no button is pushed) the Arduino plays noTone() that is, it is silent.

Upload the program, Lab3_Four_Key_Instrument.ino, and test it.

Question 1) Does it work as you expect?
Question 2) What happens if you push two or more keys at once? Is it consistent? (that is, push the pin 4 button before the pin 5 button and vice-versa)
Question 4) Can you explain the reason some buttons make notes, while others don't depending on what button is already pressed?
Question 3) What happens if you press keys one at a time but quickly in succession?

I call these kinds of tests "trying to break it." That is, try to see what happens when something the software wasn't designed to do happens. How does the system respond? Is it acceptable? Demonstrate this circuit for signoff and turn in your question answers.

You must print off the sign-off sheet, write your answers to get signoff. To receive credit for this lab, you *must* get the sign-off before the due date (see Canvas), during class time or with a TA during their hours. The instructor will not signoff outside of class hours.