Lab 14
Electronic Sensors and the Arduino Microcontroller

Objectives – in this lab you will

- Understand how microcontrollers use electronics to interact with the outside world
- Learn the basics of writing programs for a microcontroller
- Write programs that measure and respond to input from electronic sensors
- Read a potentiometer, control LEDs, and produce audio output

Key Prerequisites

- Labs 1-13

Required Resources

- Arduino UNO, PC with internet access, Lab Kits, DMM and Oscilloscope

References

- Arduino on Wikipedia
- Arduino Programming Language Reference
- Intro to Arduino from SparkFun
- Arduino on Mac OS-X
- What's a Microcontroller? (Good sensor reference for Parallax Microcontroller)

This lab introduces a ubiquitous element of modern electronics systems: the microcontroller. Up to now, our circuit labs have demonstrated various physical properties of electronics. In this lab we will explore how circuit designers use the principals we've been learning to interface computing systems (microcontrollers) with the outside world, allowing for a vast array of consumer electronics and industrial applications.

Microcontrollers occupy a key (though hidden) place in the electronic devices that we use every day. They are literally everywhere, providing the intelligence to make products behave properly.

Figure 1
Every-Day Examples of Devices that Contain Microcontrollers

Today's hobby microcontroller market allows non-engineers access to these powerful devices for independent experimentation and invention. The market is growing rapidly and now includes such celebrity products as Arduino, PicAxe, and Raspberry Pi, and classics such as the Lego Mindstorms RCX/NXT Brick and one of the earliest—and still popular—Parallax Basic Stamp 2.

In this lab we will introduce one of the more recent arrivals, the Arduino UNO Rev 3**,** one in a broad line of product from the Arduino open-source hardware and software company and its devoted user community. From Wikipedia:

> The first Arduino was introduced in 2005. The project leaders sought to provide an inexpensive and easy way for hobbyists, students, and professionals to create devices that interact with their environment using sensors and actuators. Common examples for beginner hobbyists include simple robots, thermostats and motion detectors [1].

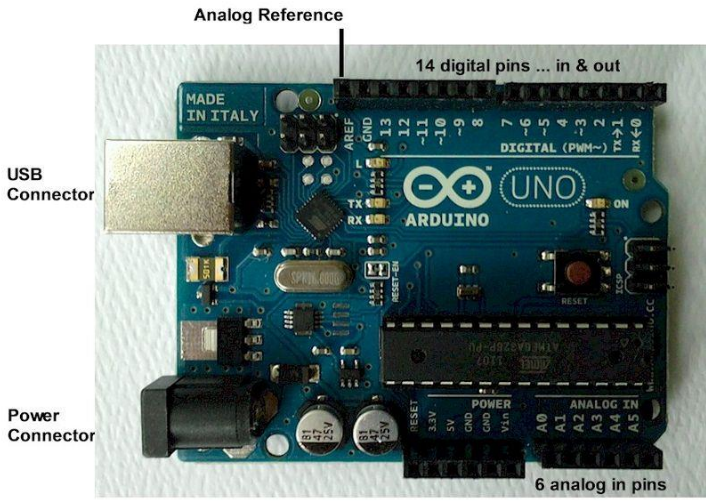


Figure 2: The Arduino UNO R3 is a complete microcontroller system for about $25 [2].

Vocabulary

All key vocabulary used in this lab are listed below, with closely related words listed together:

- Integrated Development Environment
- Sketch
- variable
- function

Discussion and Procedure

Part 1. Setup and Testing

Installing Arduino Software (for online students)

To begin, if you are working remotely on this lab, you'll need to download the Arduino software Integrated Development Environment (IDE), which is the application where we write and execute programs on our Arduino microcontroller. Simply visit the Getting Started with Arduino page and follow the link for your operating system (Windows, MAC OS-X or Linux). Continue following the setup instructions all the way through running the "Blink" example program. Verify that the yellow LED near pin 13 blinks 1 second on, 1 second off after the program has been uploaded. If you run into problems, visit the "troubleshooting suggestions" link from the "Getting Started" guide and/or email the instructor for suggestions.

Connecting for the First Time (for classroom students)

If you are working in the classroom, your laptop should already have the Arduino software installed. Here are the steps to bringing up the Arduino software for the first time:

1. Before running the Arduino IDE, first plug the USB cable into your Arduino, and the other end into your laptop. Windows will load the correct USB drivers and display a message in the lower right hand corner "Arduino UNO on COM Port X" where X is a number. Remember this number, since it can be useful later.
2. Open the Arduino program
3. Select Tools > Board and choose Arduino UNO
4. Select Tools > Serial Port and if you see multiple ports, choose COM Port X, where X was the number shown in step 1. This step was able to resolve connection problems that some people ran into in the lab.
5. Open the blink program (in Arduino-speak, a program is a "sketch"). Choose File > Examples > 01.Basics > Blink

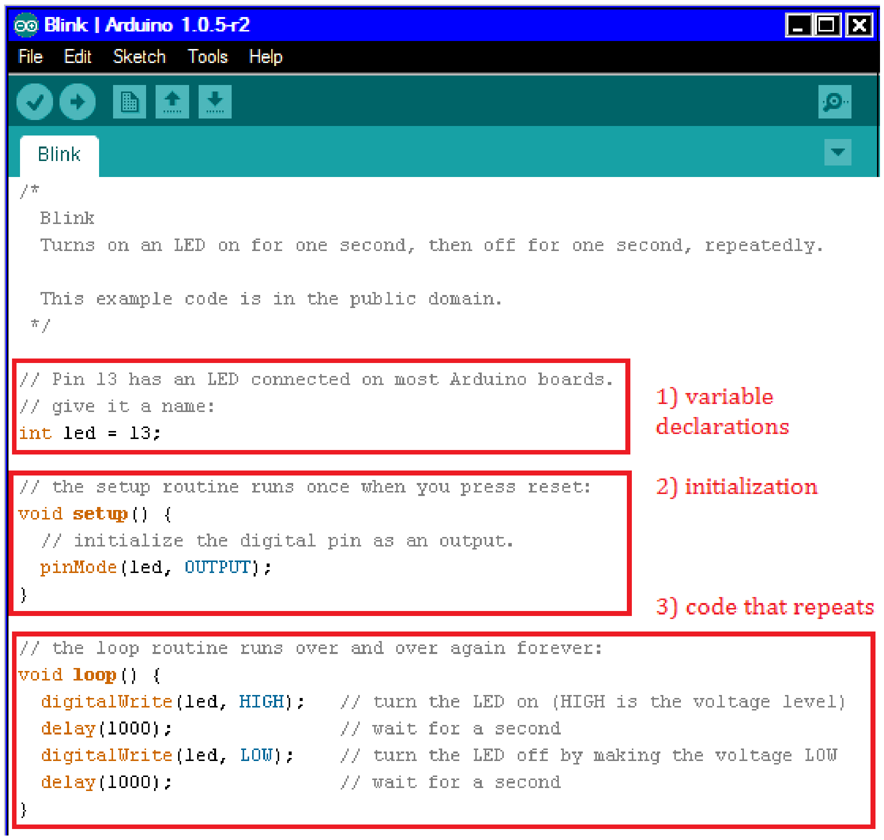
You should get a program that looks something like the code in Figure 3 below.

6. Before running the blink program, make a note of how fast the yellow LED next to Pin 13 on the Arduino is blinking. It's probably going on for ½ second, off for ½ second, or about 1 second for a complete cycle.
7. Now upload the Blink program from your editor into the Arduino. Click the right-arrow button beneath the word "Edit". There will be some messages shown on the bottom of the screen as the upload takes place. If everything works ok, you should notice the yellow LED on the Arduino slow down to 1 second on, 1 second off, or about 2 seconds for a complete cycle.

All Students

The programming language used by the Arduino is a variant of C, a powerful programming language used in a lot of microcontrollers. If you look closely at the program in Figure 3, you'll notice it has some things that your program doesn't. For instance the line `int led = 13;` This line creates a variable – just like in MATLAB. This variable holds the number 13. It is also restricted to only hold values of type "integer" or whole numbers, hence the word `int` that comes at the beginning. This variable comes in handy farther down the program when the `pinMode` and `digitalWrite` commands are used. We use the variable `led` instead of `13` because that provides a little extra clarity on what our program is doing. It also makes it easier to change the pin we are using later to something else, such as 7 or 9, without editing all those statements as well.

8. Modify your program so it looks exactly like the program in Figure 3 below, by declaring the led variable and using it in the other commands shown. Rerun your program and make sure it still works.



```
/*
  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
}
```

Figure 3. Blink, a typical Arduino Sketch

Figure 3 also has a number of red boxes to show you the structure of an Arduino program. In general, it's best to create variables in the area marked by the first red box. Variables created here can be referred to anywhere else in your program.

The second red box shows the `setup` function. A function is a block of code, surrounded by { and }, with a name at the top (in this case, `void setup()`). The setup function is where we perform any initialization steps we need to do. In this case, we have to indicate that Pin 13 will be used for output. This allows an extra amount of current to come out of the pin when it is set to high, so that we can illuminate an LED with it.

The third red box shows the loop function. This function starts running after the setup function, and it keeps running over and over again until we turn off the power to the Arduino, or upload a new program. This is where all of the action happens! In this case we are turning on the LED, waiting 1 second, turning off the LED, waiting another second, and repeating this over and over again.

This general form is common to all Arduino programs. In summary, there are three sections:

1) a section for creating variables that will be used in the program
2) a section for initializing any pins or other variables we will be using
3) a section that repeats continuously until the power is turned off or a new sketch uploaded

We will be exploring this program structure further in the following example applications.

Part 2. Flashing LEDs

In this section, we will continue with the flashing LED program by connecting two more LEDs to the Arduino and programming different blinking patterns. Note that the Arduino can run on power provided to it through the USB cable, so you do not need to connect power to the Arduino board.

9. To begin, disconnect the USB cable from the Arduino and wire two LEDs and two 220 Ω resistors to your breadboard with jumper wires connecting to the Arduino as shown in Figure 4 below right. One LED is connected to Pin 13 of the Arduino, while the second LED is connected to Pin 12. Recall that the 220 Ω resistor is required to keep the current from overdriving the LED and burning it out, and that the **<u>long wire</u>** of the LED goes on the higher voltage side.

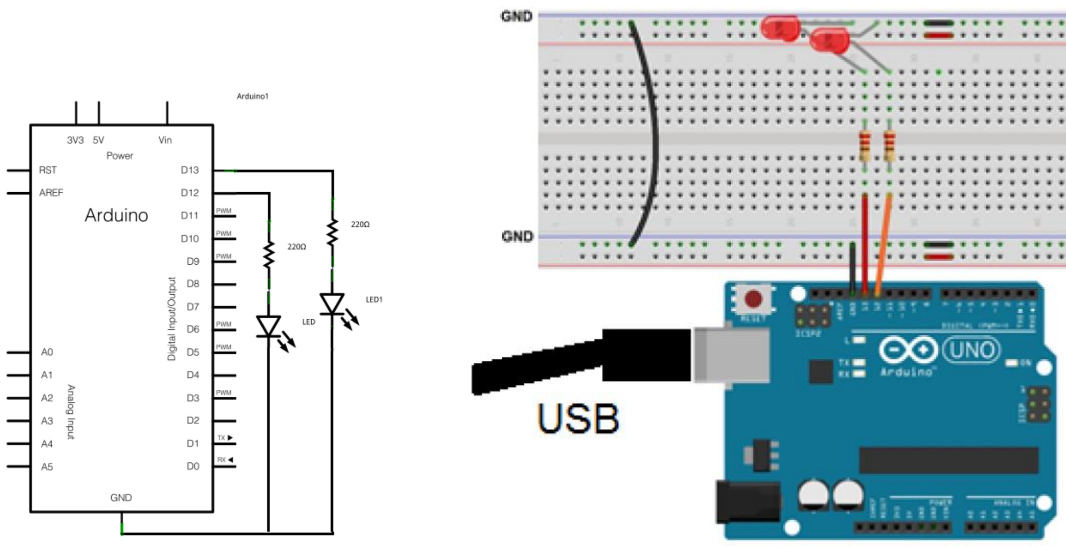


Figure 4. Schematic and Breadboard view of Arduino connected to two LEDs.

10. When you reconnect the USB to your Arduino, you should notice the left LED on your breadboard blinking at the same rate as the yellow LED on the Arduino. You can learn a little

more about how the program works by reading the following tutorial link: http://www.arduino.cc/en/Tutorial/Blink

11. By following the example of the code you are already running, add statements to flash pin 12 the same way that Pin 13 is flashing. You can create another variable if you like in the initialization section, such as led2 (declare `int led2 = 12;` right below `int led = 13;` -- don't forget the semicolon, which acts like a period in C), and set it equal to 12. All else should be pretty easy to modify and run, just make sure you don't mess up the blocks created by the curly braces { and }!

12. You might have come up with a `loop` function that does something like this:

```
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
   digitalWrite(led2, HIGH);   // turn the LED2 on
   delay(1000);               // wait for a second
   digitalWrite(led2, LOW);    // turn the LED2 off
   delay(1000);               // wait for a second
}
```

However, this doesn't really work the way you might think. Notice the pattern of the lights. The first LED turns on and off completely, then the second LED turns on and off completely.

How can you make it so the two LEDs come on at the same time and turn off at the same time? If you think about it, you only need two delay statements. Statements that happen one after the other without a delay between them will occur practically simultaneously (the Aruino can process thousands of statements in one second). So if you remove the first and the third delay statement, then change the led variables around a little, you'll get something like this:

```
void loop() {
   digitalWrite(led, HIGH);   // turn the LED on
   digitalWrite(led2, HIGH);  // turn the LED2 on  (at the same time as LED)
   delay(1000);               // wait for a second
   digitalWrite(led, LOW);    // turn the LED off
   digitalWrite(led2, LOW);   // turn the LED2 off (at the same time as LED)
   delay(1000);               // wait for a second
}
```

If you run your program now, you should see the two LEDs are now flashing in sync.

13. Modify the delay so the LEDs blink at twice the rate.

14. Now modify so the LEDs flash in opposition (Pin 12 On and Pin 13 off, followed by Pin 12 Off and Pin 13 on, in succession.

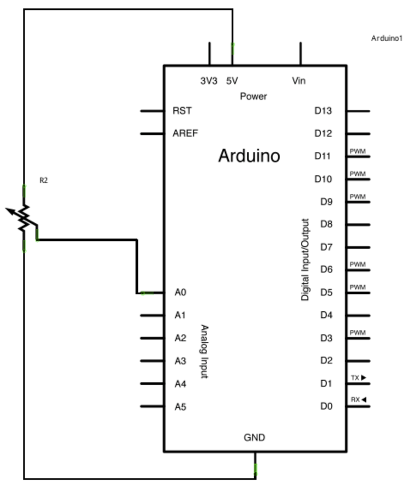
15. Save your program as Blink2. Then paste a copy of your program into the datasheet.

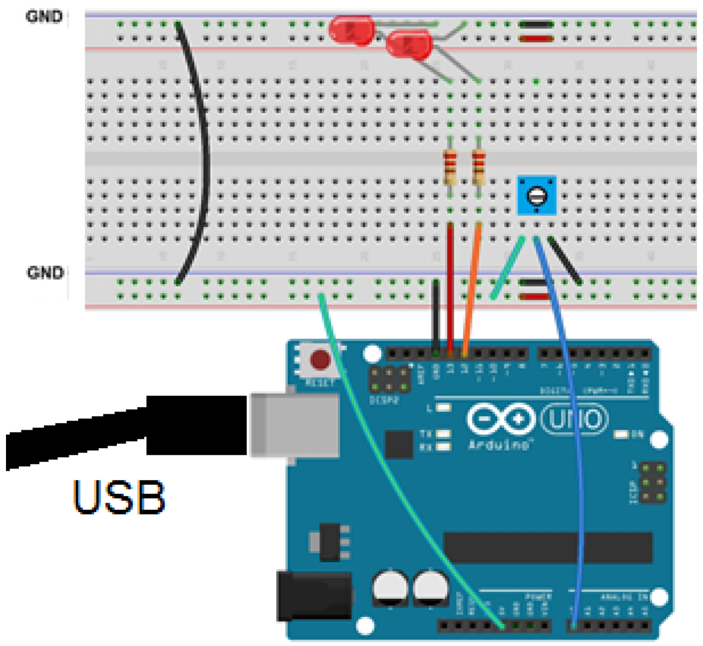
Part 3. Measuring the Position of a Potentiometer

The Arduino has a set of pins that measure analog input voltages ranging from 0 to 5 V, and converting them to digital numbers ranging from 0 to 1023 using a build in Analog-to-Digital Converter (ADC). We can use these pins to read the position of a potentiometer. Recall that a potentiometer is a variable resistor. The 5K potentiometer on the breadboard shown below is the blue cube-like component labelled R502 in your lab parts kit.

If we connect one end of the "pot" to 0 V, the other end to 5 V, and the center pin to pin A0 of the Arduino, the voltage appearing at pin A0 will reflect the rotational position of the potentiometer shaft according to the properties of a voltage divider. By reading this voltage with the Arduino, we can infer the setting of the potentiometer and modify the program to respond to changes in its position. More about this step can be found at http://www.arduino.cc/en/Tutorial/AnalogReadSerial

16. Disconnect the USB cable and add the additional wiring to incorporate the potentiometer in the system. Then plug the USB cable back in.





17. Open the demo program by selecting File>Examples>01.Basics>AnalogReadSerial.
18. Run the program by clicking the upload button.
19. Open a Serial Monitor window by clicking the magnifying glass icon, at top right
20. You should see a steady stream of numbers ranging from 0-1023, correlating to the position of the pot. As you turn your potentiometer, these numbers will respond almost instantly.

21. Now for the challenge: modify your Blink2 program to use the value read by the analogRead command as the delay for the flashing of your LEDs in the Blink2 program. When you get this working, the LED flashing will speed up or slow down as you rotate the potentiometer shaft.

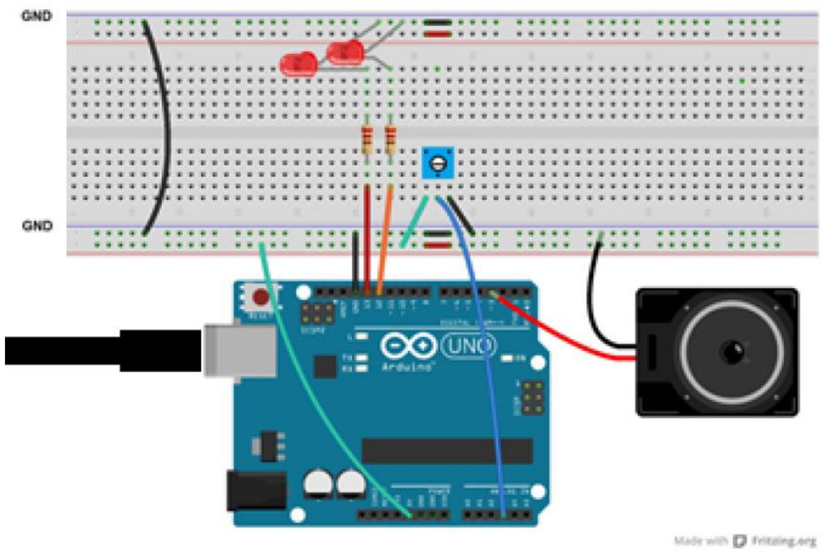
    Here are some hints to get you started:

    a. copy the "`Serial.begin(9600);`" command from the AnalogReadSerial program into the setup function – inside the block of statements that begin with { and end with } – for the Blink2 program.
    b. copy the "`int sensorValue = analogRead(A0);`" statement from the AnalogReadSerial program into the loop function – inside the block of statements that begin with { and end with } – for the Blink2 program. Put this statement at the top of the block, right after the line "`void loop() {`"
    c. copy the `delay(1);` statement into the bottom of the loop( ) block in Blink2
    d. Last of all, change the number you are using in the delay commands for your LED blinking to `sensorValue`.
22. Save your program as Blink3 and run it again. You should observe the LED blink rate change as you rotate the pot.
23. When it works, copy your code for Blink3 into the datasheet.

Part 4. Sending Audio to a Speaker

In this experiment, we will connect one wire of our speaker to digital pin 3 of the Arduino, and the other to ground, and send a tone to the speaker based on the position of the potentiometer. When we rotate the potentiometer, we will hear the pitch of the tone changing frequency. We will also observe the waveform produced by the Arduino using our Analog Discovery Oscilloscope.

24. To begin, wire up the speaker from your kit as shown below, red to digital pin3, black to ground.



25. Open the demo program by selecting File>Examples>02.Digital>tonePitchFollower.

26. At the bottom of the `loop()` function, below where it says "play the pitch" locate the number 9 in `tone(9, thisPitch, 10);` and change it from 9 to 3 to reflect that we are using pin 3 instead of 9 for our hookup.

27. Run the program and listen to the output change pitch as you vary the potentiometer.

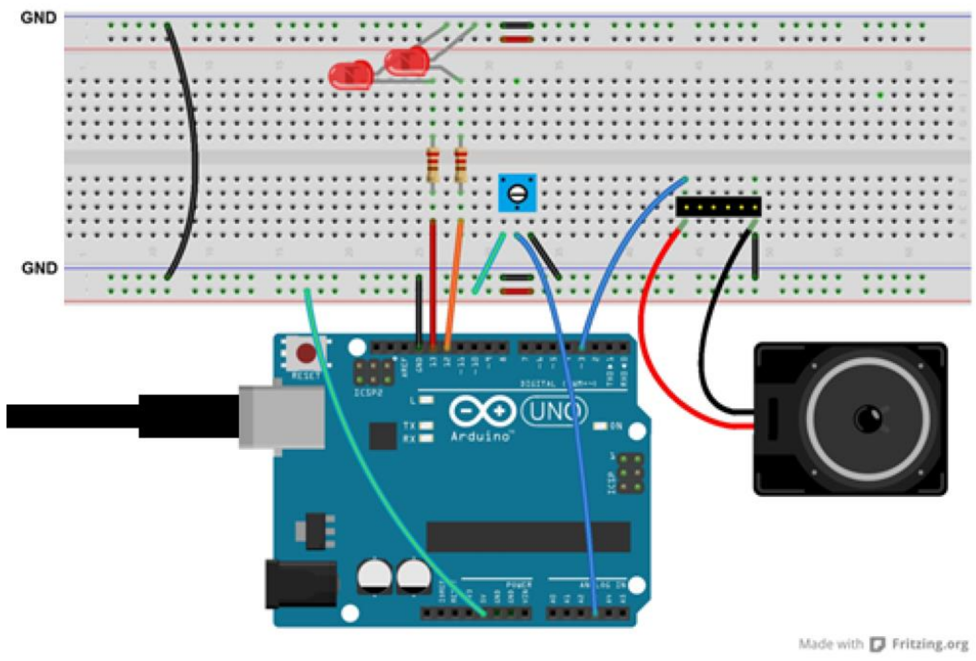
This program reads the potentiometer as before, but uses the `map` command to change the value read by the analogRead command to a value in the range of frequencies we wish to hear. You can change the numbers in this statement to reflect the actual range of input values from your potentiometer, and the desired range of frequencies you wish to hear. It sends out a tone equal to the value in `thisPitch` for 10 ms before reading the next setting on the potentiometer.

You may wish to explore the frequency limit of the speaker. Realizing the speaker is like a coil (inductor) you may find there is a preferred frequency where the speaker sounds the loudest. This would be when the impedance of the speaker matches the output resistance of the digital pin driving the speaker.

Finally, we are going to observe the waveform produced by the Arduino while the sound is being generated.

28. You will need to rewire the speaker connection slightly as shown below in order to connect the Analog Disovery.

29. Connect the Analog Discovery orange wire to the header pin where the red speaker wire comes in, and the orange-white wire to the pin where the ground comes in, as shown.



30. Start up the WaveForms program and select the Oscilloscope. While the Arduino is playing the tone, click RUN and then AutoSet. You should see a square wave on the display, but you may need to change the time base to resolve it properly. Uncheck the “C2” box on the right to hide the blue trace from channel 2, since that is just showing noise.

31. Add measurements (click the yellow triangle icon) to show the frequency of the tone.

32. Zoom in or out in time if necessary to capture a good waveform that clearly shows the square wave you are hearing.

33. Use the snipping tool to capture your view of this waveform and the measurement window showing the frequency and paste it into your datasheet.

34. Comment on the appearance of the waveform and how it relates to the sound you are hearing.

This has been just a brief introduction to the Arduino, but you should feel that you now have a foundation that you can build on with additional experiments.

When you are finished, add an estimate of the time required to complete the lab to your datasheet and upload the datasheet to the server.