Summer 2019

Arduino Lab Assignments for Principles of Physics II

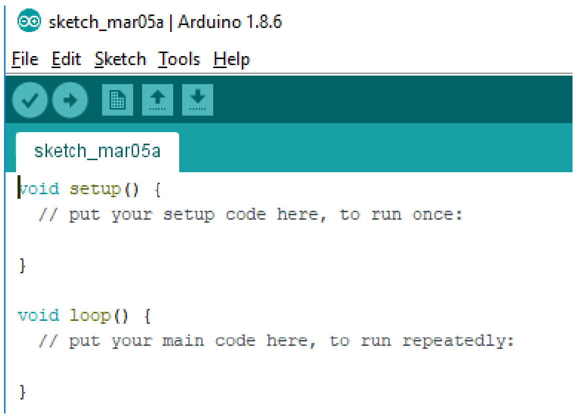
William Baird

Jeffery Secrest

Arduino Part I

The Arduino is an extremely popular microcontroller/programming language system. Because these devices are under $30 (in some cases, under $15) and the programming environment is free, they have been used for just about every kind of project you can imagine. Today, we are going to start exploring them so that we'll be able to use them in an actual lab later. The first step is to plug the board into a free USB slot in your computer. From your desktop, start the Arduino program by clicking the icon:

When the program starts up, it should pop up a window that looks something like this:



```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

If the title of the window at the top left isn't "sketch_" and today's date, you can click on File →New and then you should get a window like the one above. Notice it is very simple. The first thing in the file is a line that says `void setup() {`. Every Arduino program starts with this. Notice that there is an opening curly brace at the end of this line, and then a few lines down there is a closing curly brace by itself. As the name suggests, the stuff inside these curly braces is "setup" material, or things that only need to be done once rather than over and over again. For example, when we get to the point that we're sending data from the Arduino to the PC, we will have to tell the PC what data rate to expect. We only have to do that once, so it goes inside "setup".

All that is there now is the line that starts "`// put your setup`…". Anything after two consecutive slashes is completely ignored by the program; it is there only for humans to use. The point of this is to allow you to comment your code so that other people can understand what you're trying to do. This is one of the most neglected but most important parts of programming; people tend to be careless and think "I don't need to comment this – no one will ever see it but me, and I obviously know what I mean here". You would be (unpleasantly) surprised to find out how quickly you will forget what you were trying to do if you have to go back and alter/maintain this code. There is a balance to this – if you have a line that says "`x=x+1`", you don't need a comment that says "`//add one to x`". With a name like x, though, you might want to put something like "`//x = number of users`" to remind you of what you are counting.

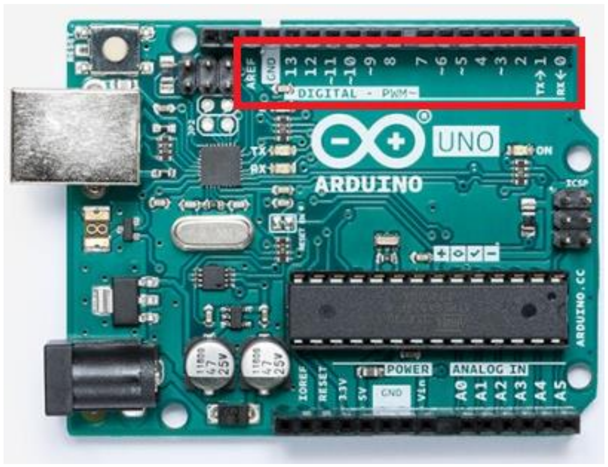
After the closing curly brace from the setup function, you'll see `void loop() {`. As the comment below it indicates, this is where you put any statements that you want the Arduino to execute over and over again. After it does everything you've put between *these* curly braces, it will go back to the top of the `loop` and start all over again. Note that it will **not** go all the way back and execute the code in the setup loop again.

The advantage of the popularity of the Arduino is that there are so many pre-existing programs out there on the internet you could conceivably use it for years and never write your own program from start to finish, but rather just modify other people's work to fit your needs. If you look under the "File" menu, you will see "Examples" and then under that, many more choices. Pick "01.Basics" and then "Blink". This is about the simplest program that actually does anything.

The top of this file is about a paragraph of comments explaining it. Notice that another way to indicate comments is to start a block with "/*" and then end it with "*/". Everything in between, over however many lines, will be ignored. The comments explain that the purpose of this program (known as a "sketch" in Arduino world) is to blink the on-board LED. This setup function has only one command:

```
pinMode(LED_BUILTIN, OUTPUT);
```

The command pinMode is used to specify what you want a particular pin (called a pin though we generally interface with it through a socket instead) to do - gather data (input) or send data (output). Ordinarily, we would put a number in as the first argument, where the numbers are stenciled beside the relevant sockets on the board as shown below:



(https://store.arduino.cc/usa/arduino-uno-rev3)

The reason that isn't done here is that these samples are written for all Arduinos, and which pin is connected to the built-in LED changes from board to board. Once you tell the program which Arduino you have, it will know what pin `LED_BUILTIN` really refers to in your case. If we want this to turn on and off, that's an `OUTPUT` (and it's really all we can do for the particular pin that has an LED attached to it).

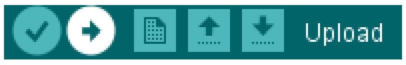
Next in the program listing, the main loop handles actually blinking the LED.

```
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);                       // wait for a second
  digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);                       // wait for a second
}
```

There's not much left to explain here – we change the value of a pin (as long as it's already been set to `OUTPUT`) using the `digitalWrite` command. The program again makes this usable for all Arduinos by not referring to the pin with a number but with the keyword `LED_BUILTIN`. Our two output choices are `HIGH` and `LOW`, referred to in other electronic situations as ON and OFF, 1 and 0, or TRUE and FALSE.

Finally, the statement `delay(1000);` tells the Arduino to wait 1000 milliseconds before doing anything else. Notice that all of these statements have semicolons after them. Practically every statement will require that, except for some things like "`If`" statements that require curly braces instead. A very valuable resource when learning to program is a dictionary of what statements are available and how to use them. You can find the Arduino's at https://www.arduino.cc/reference/en/ .

Now you need to send this to the Arduino board. First, go to "Tools" from the top menu. Near the bottom, it should tell you which Arduino board you have ("Arduino/Genuino Uno" for what we use) and the "Port" it is using. This is generally a number less than 10 or so, and there's no way to guess what it will be on a particular computer, so just look and see where it landed on your machine. These two things are necessary to program your Arduino and communicate with it. If those are set, you can look for the "Upload" button (right arrow) and press it:



This will transfer the "Blink" program to the Arduino, and after a brief wait (under a minute in almost all cases) the board's LED should start blinking. Now that you've done that, imagine you want the LED to be on for 0.12 seconds and then off for 0.55 seconds. Write the four lines of code below that will make that happen:

____

____

____

____

Now put this into your program, upload it, and see if it works as expected.

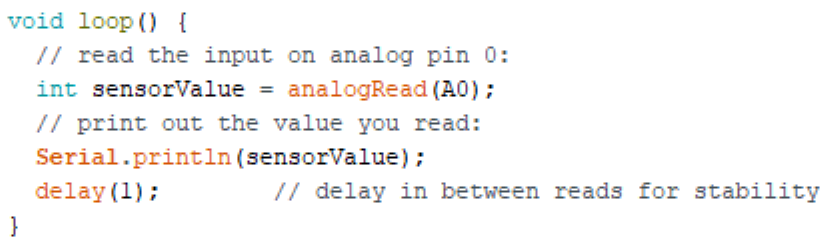
One of the most important things the Arduino can do is to measure a voltage and then tell us what it got. It does this using the idea of Analog to Digital Conversion (ADC). The details of how this happens are not important for this class, but the big idea is that we get a digital representation of an analog (i.e., real-world) voltage. Like all voltmeters, the Arduino has a range within which it can measure voltages. For our board, it's the range from 0 V (ground) to 5 V. Don't ever put a negative voltage or a voltage more than 5 V above ground into one of the Arduino pins. It could damage the Arduino or possibly even the PC connected to it.

The ADC inside our boards is known as a 10-bit ADC. This means it will measure a voltage and rank its strength on a scale from 0 to $2^{10} – 1$, or 0 to 1023. Connecting the ADC pin to ground should make it read 0, and connecting it to 5 V should make it read 1023. All other voltages are scaled proportionally. If we have 1024 possible voltages equally spaced between 0 and 5 V, that tells us our resolution here is 5 V/1024 or about 5 mV. Not bad. Other Arduino boards have 12-bit ADCs that are restricted to the range from 0 to 3.3 V. What would their resolution be?

____________Volts.

To get the Arduino to read an analog voltage and send it to us, there is already an example sketch in the environment. Again, go to File → Examples → 01. Basics → AnalogReadSerial. Again, it's a very short sketch. One difference between this and "Blink" is that we don't have to set up the pin for input or output, because this requires us to use the Analog pins instead of the digital ones. In the photo from before, notice that there are six pins on the bottom right under "Analog In" labeled A0 through A5. These are always inputs.

We do have to setup the Arduino to talk to the PC. The command to do this is simple: `Serial.begin(9600);` where the 9600 refers to how many bits per second the Arduino will send the PC and vice versa. The main loop isn't much worse:



```
void loop() {
  // read the input on analog pin 0:
  int sensorValue = analogRead(A0);
  // print out the value you read:
  Serial.println(sensorValue);
  delay(1);        // delay in between reads for stability
}
```

The `int` tells the Arduino that we want to create a variable called sensorValue (you can change the name of this to anything you want that isn't already a keyword like pinMode, etc.) and that it only needs to reserve two bytes of space (16 bits) to hold it. We don't really need 16 bits for a 10-bit ADC, but it's the smallest size that's not too small to hold it. The part that tells the Arduino to read the voltage is `analogRead(A0)`. This tells us we will be looking at the voltage on the pin labeled A0.

That information is sent to the computer by the line `Serial.println(sensorValue);` . Serial means send it to the PC and we could either use print or println. The "ln" at the end of print means to print whatever and then start a new line. If we just used Serial.print(whatever), we'd get one long string of numbers with no breaks that would be very hard to read.

Finally, we wait 1 ms between reads to that we don't send too much data at once. Put one end of a wire in the A0 socket and let the other one wave free. Hit the upload button. After it's finished, we want to look at the data we're getting. At the top right of your window, click on the . This opens a serial monitor. Notice that at the bottom right of this window, it has a "baud" setting. Baud means "bits per second", and the Arduino is sending at 9600. If your window is set for anything else, you will get garbage. You're free to try it – it won't cause permanent damage. When you're done, switch it back to 9600 and then put the free end of the wire into one of the sockets labeled "GND". What numbers are you seeing in your monitor?

Now put the free end in the 5 V socket. What do you see now?

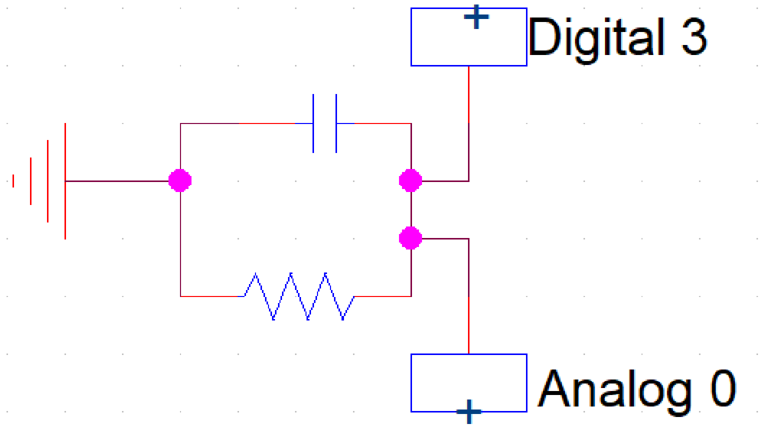
Now put the free end in the 3.3 V socket. What do you see now?

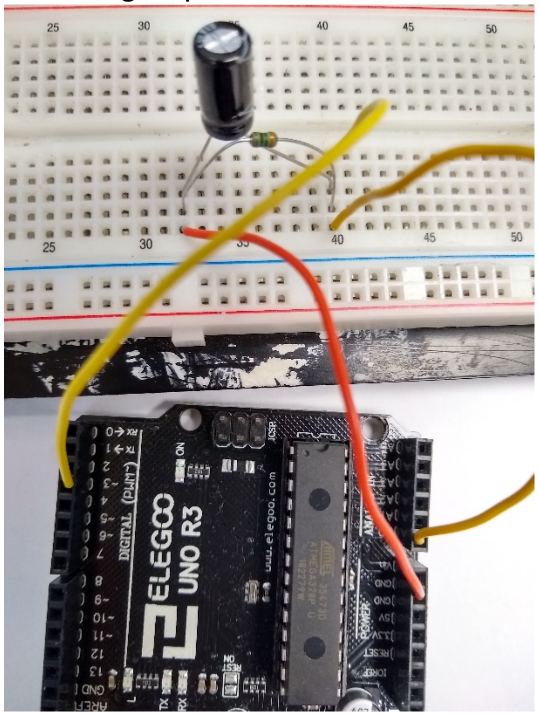
Finally, go to Tools → Serial Plotter and see what that does.

Arduino Part II

Now that you have used the Arduino at least once before, we're going to use it today to actually do a lab that we've already done with the Analog Discovery black box. We will measure the time it takes a capacitor to discharge through a resistor, but this time we want to completely automate it so that you don't have to move wires back and forth. We'll use most of what you did before to do this. The capacitor's voltage will be read by the `analogRead` statement from before and we'll charge the capacitor using one of the digital output pins.

First, we can set up the physical circuit. We need to connect a resistor in parallel with the capacitor. At one end of the pair, we connect a wire to one of the "GND" sockets on the Arduino. At the other end of the RC combination, we connect a wire to Analog 0 and a wire to Digital pin 3 as shown below.





Now that you have the physical setup in place, you can start programming. To get started, you can go to File →Examples→Basics→ Bare Minimum. That will give you the `setup` and `loop` structures every Arduino program has. Once this opens, save it somewhere under a different name. At the very beginning, even before the `void setup()` line, we need to declare a couple of variables (in other words, tell the Arduino to save some space for two variables we want to use). This is all you need:

```
int sensorValue = 0;
int tot=0;

void setup() {
```

The "int" just tells the Arduino how much space to save. It means an integer between -32768 and +32767.

In the `setup()` loop, all we need to do is prepare the serial output. We can communicate at a wide variety of speeds, but we want it to be quick. Try this:

```
Serial.begin(115200);
```

Now you have to set up the rest of the program, which is inside the `void loop()` function. Remember that to read the voltage from the pin labeled ANALOGIN A0, you just need the `analogRead` function. It will look like

```
  sensorValue = analogRead(A0);
```

Here's one of the only parts that will be different from last time. Our plan with this circuit is to constantly measure the voltage, and then when it drops to a small value, we recharge the capacitor. Remember that `analogRead` doesn't provide us with a "real" voltage but rather with a number from 0 to 1023 where 0 = 0 V and 1023 = 5 V. In this case, we might pick 50 as a small value of voltage since it would correspond to about 0.25 V. The control structure we want is an `if` statement, and it looks like this:

```
If (condition) {
Do stuff
}
```

Our condition might be "sensorValue<50" in this case. If this happens, we want to turn on pin 3 as an output, make it high (= 5 volts), wait a short amount of time, and then turn it back off. It might seem that all we need here is a digitalWrite to turn pin 3 HIGH, then something like delay(20) to wait 20 ms, and then we digitalWrite pin 3 low. The only problem here is that we want pin 3 completely out of the circuit after 20 ms. If it is still connected and set to low, it will just drain the capacitor almost instantly.

The fix is to turn it into an input instead of just making it low. We're not really going to read it, but making it an input pin makes it high impedance and therefore effectively removes it from the circuit. Since we make it an input as we leave the "if" loop, we will have to make it an output at the beginning of that same loop. So, we need `pinMode(3,OUTPUT)`, `digitalWrite(3,HIGH)`, delay for 20 ms, and then (we can skip the "LOW") `pinMode(3,INPUT)`.

Once you're done with the If loop, that's most of the work. We could just output sensorValue over and over again, but it turns out that the Arduino is not the best voltmeter in the world. To fix some of its noise, you should take many measurements and average them. Of course, since the real voltage value should be dropping all the time, we don't want to take too much time gathering data or we'll be averaging things that really shouldn't be the same. We'll go for 10 measurements. This can be done with a "For" loop:

```
for (int i=0; i<=9; i++){
  sensorValue = analogRead(A0);
  tot=tot+sensorValue;
}
```

The first line starts the loop and gives it the conditions: start at zero, go as long as i is less than or equal to 9, and i++ means add one to i every time through the loop. When you leave the loop, “tot” will contain the sum of 10 readings. We could divide it by 10, but we don’t really have to since we aren’t measuring in volts anyway. **(Don’t forget that at some point, you’ll want to reset tot to zero; you decide where that should be in your program).**

The last thing we need to do is output tot and the time. The Arduino has an easy-to-use time counter function called `micros()`. It gives the elapsed time since the program started in microseconds. The only catch now is that we also have to have a way to gather all of this data. We could **watch** it go by in the Arduino’s serial monitor, but that won’t give us data that Excel can use and fit to an exponential.

The data collection program can be downloaded at my web site https://sites.google.com/georgiasouthern.edu/wbaird/home if you click in the upper right corner and go to the “Labs” page. At the bottom, you should see “Serial Data Collection”. Click on it and download it to your computer. Ignore whatever warnings you may get about how horribly dangerous this is.

When you start it, you’ll see a simple interface. This program is designed to collect up to 9 pieces of information from the Arduino, but we’re only sending two. It should default to 115200 baud, so that will match your Arduino. The port number may not match, so change it to match whatever the Arduino program said to use. Use the “Set Path” button to pick a location for the saved file, and if you want to change the filename, you can do that, too.

When you hit “Start Collection”, it will begin gathering data. However, you aren’t sending anything yet. Go back to the Arduino program so we can finish that part. The Data Collection program expects you to start with a < and then give it 9 numbers separated by commas, and end with a >.

Since we only have two data points at a time, the other 7 will be garbage and we can just fix them as shown below:

```
Serial.print("<");
Serial.print(micros());
Serial.print(",");
Serial.print(tot);
Serial.println(",3,4,5,6,7,8,9>");
```

Notice the last statement is `Serial.println` instead of `Serial.print` so that we get an end of line character. This will give us a file that Excel understands.

You should now be ready to collect some data. Once your program is uploaded to the Arduino, you can start the Data collector. Watch the activity in the Data2 box. You should see the numbers go from about 10000 (10 measurements of about 1023 or so) down to around 500 (10 measurements of 50 or so), and then restart at 10000. We only need one full cycle of this, but you can collect more if you want.

When you're done, hit "End Collection" and then use Excel to read in the file. You can of course delete the columns of just 3, 4, 5, etc. You may also need to delete a few mangled rows of garbage in the beginning.

Column B will actually be clock time as recorded by the computer. If you click on that column, right click, and pick "Format Cells" you can pick Time→13:30:55 to get it to show you time in seconds. Skip the first few hundred rows, pick a time, and see how many data points were taken in that one second. Record that below.

____________ rows per second

Go through and delete rows from the top until you reach your first value of 10000 or more. Now move down until the first time (after 10000) you get to about 500. This could be several thousand rows, depending on R and C. Delete everything below that. Now you should have one single discharge. Before we plot it, we need to convert time to seconds and Voltage to volts. Insert a column between C and D. We want to start time at zero and we'll have to divide by 1000000 to go from microseconds to seconds. The top cell in this new column D should be "Voltage" and then in D2, you can write

=(C2-C2)/1000000

and then fill that in all the way down. In the next empty column (should be F), write in a formula that will convert your numbers to voltages. You figured out this conversion in the last Arduino lab. Remember that you also have to divide by 10 now since you took 10 voltage measurements at once.

Finally, graph columns D and F and have Excel fit an exponential trendline to your data. Be sure to include both the R^2 value of the line and the equation of fit.

Using the marked values on your resistor and capacitor, what do you expect for RC?

RC expected ______________ s.

What did you get from your Excel fit?

RC measured ______________ s.

Turn in this sheet and email me both the spreadsheet and the Arduino program for your group.