**Welcome to 10C Labs!**

A note from your TA:

Due to the COVID circumstances, this year, we are going to do things differently. In the past, we had access to sophisticated and expensive instruments like function generators and Oscilloscopes (~ 1000$) in the ECE-10 series lab. It’s not viable for us to purchase and send these instruments to each individual. So we had to adapt. This year, we will use an Arduino Uno kit (~50 $) to do our experiments. Hopefully, you have received your Arduino kits by now. Arduino is a popular microcontroller board that will enable you to do a lot of DIY (Do It Yourself) projects, including 10C labs. However, a 20-fold reduction in our lab instruments cost also comes with obvious measurement limitations. But this is not necessarily a bad thing. The definition of being an engineer is the ability to adapt. You don’t always get the equipment you need, and you need to find smarter ways to do what you’re supposed to do.

My hope is during the following labs; you learn two things. 1- Why the experiments and observations you are making are important. 2- What are the critical design considerations behind each lab.

Okay, let’s get started.

- Kamyar Parto

**Lab-1: Introduction to Arduino**

**(or how to create the cheapest oscilloscopes and function generators on earth!)**

**Lab Objective:** Introduction to Arduino. Using Arduino as an oscilloscope to read transient voltage signals. Learn how to create square wave signals (with varied frequency) and simultaneously read them with the oscilloscope.

**Step-0: Installation:**

Arduino is an open-source electronics platform. An Arduino board consists of a microcontroller, integrated circuits (IC), and input/outputs pins. The board can be programmed to process inputs and can output desired signals.

Let’s start with the “Elegoo The Super Starter Kit for Uno” documentation [here](https://www.elegoo.com/tutorial/Elegoo%20The%20Super%20Starter%20Kit%20for%20UNO%20%20V2.0.2020.8.27.zip) to install and familiarize ourselves with Arduino’s syntax. Unzip and go to folder /English/Part 1 Preparation. **Follow the installation instructions and initial lessons (lesson 1.1-2.1) in the folder** (Do lesson 1.3 if you have windows, and lesson 1.4 if you have mac. You can skip the add libraries section in 1.5). There are a lot of resources online that can also help you debug any initial problems you might face. Check out [this](https://www.youtube.com/watch?v=MIZQTvGcp3g) video and other resources on YouTube. If problems insist, I’ll be happy to help you.

**Step-1: Make the LED blink**

Let’s integrate the code from lesson 1.5 with the circuit from 2.1. Tweak your Arduino code from lesson 1.5, to switch an external LED (set it up on breadboard) on and off with a period of 2 seconds. (Hint: instead of using 5V supply, try to use a digital output, for instance, digital pin 5, as a power source that goes high and low with periodic of 2s).

**Step-2: Introduction to Arduino’s oscilloscope**

**From this point, we diverge from the documentation and will prepare Arduino for analog experiments.**

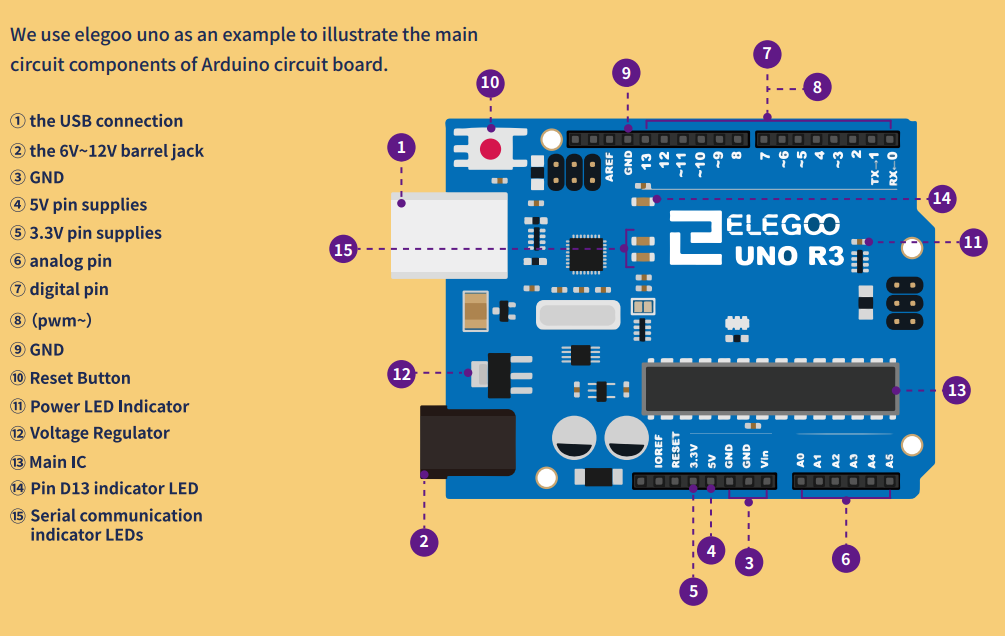


Figure.1. Schematics of Arduino Uno.

Arduino UNO has 13 Digital pins on top (+ ground), 6 analog-IN pins and several power supply pins on the bottom. Both the digital and analog pins **can only** have either a High (5 volts) or a low (0 volts) values and cannot be set to any intermediate voltages (Both have digital output). However, the analog pins (A0-A6) are equipped with an Analog input, meaning they can detect/read any voltages between 0 and 5 volt (hence the term Analog-In).

That is why we can use Analog pins as a replacement for oscilloscope. This year, in ECE-10 series courses, we will be using the [following code](https://create.arduino.cc/projecthub/Meeker6751/arduino-oscilloscope-6-channel-674166?ref=tag&ref_id=oscilloscope&offset=1) to equip Arduino’s Analog input pins with the basic functions of an oscilloscope. What this code does is it keeps sampling the voltage on A0 and plotting it across time. Remember that Arduino cannot run two or more programs at the same time. Therefore, to use the oscilloscope while running other functions (like generating a signal), you need to integrate your code with the oscilloscope code. **You essentially have to treat this code as more of a black box**. To make things easier, I have made a template sketch (download from gauchospace) that clearly shows what parts of the code you can disregard and where to write your original code, so it does not interfere with the oscilloscope.

Remember to write your variable definitions, setup code, and loop code in the designated areas in the template file. The only important settings you need to change on the oscilloscope is on the very top of the template in “Oscilloscope Configuration.” I briefly discuss them here.

- **Beams:** This changes the number of channels (signals) that are being read. For instance, if N=2, you can also simultaneously read the voltage on both A0 and A1. This can work up to a 6 channel oscilloscope.

* **SamplingInterval:** probably the most crucial setting here. This determines the time interval between different sampling events on the A0 pin voltage. If you want to probe high-speed signals, you need to choose the lowest possible time.
* **DisplayMode:** Continuous sampling vs. Triggered. Use Trigger mode if you have an oscillating signal that you want to trigger. You might find it easier also to keep the setting on continuous and snap the output whenever possible using windows + shift + S.

My recommendation is to keep the sampling time on a minimum and use continuous-wave setting unless you are told to do otherwise.

Using a 50$ Arduino scope comes with its own limitations. Here is what you need to know:

1. Arduino’s sampling rate is limited to 0.1 milliseconds. This puts a higher bound on your maximum observable frequency (~1 khz). You will not be able to notice any signals on time scales less than ~100us.

2. A0-A5 pins can only detect voltages between 5 and 0 volt (no negative read-out)! This also means we have to carefully design circuits that function between 0-5 volts to fall within the detectable voltage range.

All of the labs are designed so that you will be able to work in the detection range of Arduino (1hz to 1Khz and 0-5V). This adds some engineering complexity to the labs. However, on the bright side, you will learn a lot of EE tricks on how to deal with these practical situations.

**Step-3: Testing out the Arduino’s scope**

Copy the Arduino’s template code to the terminal and upload it to the Arduino’s board. For beams=1, your A0 pin will act as the oscilloscope input (For beam=2, A0 and A1). From Tools, open the serial plotter. Set the baud rate to 115200 (it should always match the baud rate declared in Serial.begin(baud rate) code in your sketch).

Connect your A0 pin to the 5V supply (See Figure.1). Observe the scope. Now connect A0 to the 3.3V supply pin You should see a flat line set at 5V and 3.3V, respectively, in the serial plotter. Congratulations, we made an oscilloscope.

**[Side Note]:** The time-scale of your x-axis in the serial plotter depends on the value of your sample interval, but usually for the suggested minimum sampling interval, your x-axis has a unit of ~1 millisecond. For larger values of your sample interval (~100), your x-axis timescale will change (~10ms). You can always confirm your timescale by generating a 1Hz signal (1s period) and probing it with your scope.

**Step-4: Creating a function signal and observing it - ACT I**

**(or how to fail and learn from your mistakes!)**

As you remember from 10A and 10B, the primary ingredient of analog electronic labs are oscilloscopes and function generators. These two, most of the time, work together. For instance, the function generator supplies sinusoidal waves, square waves, and pulses to our circuits, and at the same time, we probe our circuits response with an oscilloscope. We already created the scope in the last step. Now we focus on integrating the scope with a function generator to complete our setup.

In Step-1, you also wrote a very simple code to create a square wave with a period of 2s. Let’s try to integrate the probe and the function generator together now. Integrate your code from Step-1 with the scope’s template. Upload the code to Arduino. Confirm that your LED is lighting up with correct timing. Now open the serial plotter and use A0 pin to scope the signal on your digital output pin (the pin you are using to supply the resistor and the LED. What are you seeing? Why do you think the scope is not functioning correctly?

**Step-5: Creating a function signal and observing it - ACT II**

As you probably figured it out by now. The delay function in the code for the generation of the square wave, also delays the probe of the A0 by 2000ms.Therefore, you will not be able to see the correct output on the serial plotter. This is a major limitation; it means we should not use a delay code in the loop function if we want to probe using the oscilloscope at the same time.

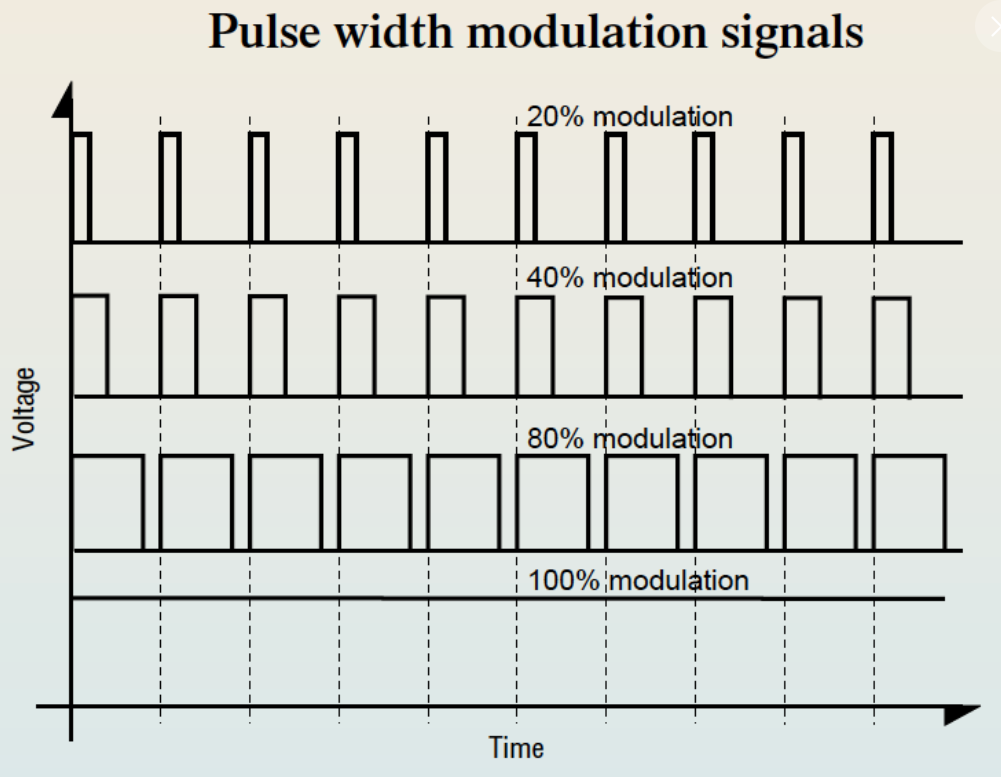


Figure.2. Pulse width modulation.

But some good news, Arduino Digital pins are equipped with Pulse Wave Modulation ([PWM](https://www.arduino.cc/en/tutorial/PWM)). PWM is a method for generating an output signal with analog behavior using digital sources. This is usually achieved by modulating the duty cycle (the period where the signal is high) of square waves (See Figure.2). For instance, a 5v square wave, with a duty cycle of 50%, acts as a fixed 2.5 voltage source in delivering power (See lesson 2.2 for more details).

Since Arduino is already equipped with PWM, this means digital pins can be activated to produce a square wave with different duty cycles. However, the frequency of this wave is usually set at 490 Hz by default, and you can only change the duty cycle of the square wave using digitalwrite() function (skim through lesson 2.2). To circumvent this, we will download PWM library (for more information, look [here](https://circuitdigest.com/microcontroller-projects/arduino-waveform-generator)). This library accesses the Arduino’s internal timing registers and allows us to change the frequency of the PWM waves too! This enables Arduino to act as a “good” square wave function generator without interfering with the scope’s functionality.

To install the PWM library, download “PWM.zip” from Gauchospace or from [here](https://storage.googleapis.com/google-code-archive-downloads/v2/code.google.com/arduino-pwm-frequency-library/Arduino%20PWM%20Frequency%20Library%20v_05.zip) and unzip the file. Copy folder "PWM" and paste it to "C:\Program Files (x86)\Arduino\libraries" (for Windows) or "Documents/Arduino/libraries" (for Mac).

The example below shows a basic sketch for generating square waves with a frequency of 1Hz on digital pin 9 using this library. At the very top of your sketch, the PWM library must be included. In the setup, *InitTimersSafe()* starts allowing the modification to internal Arduino’s timers. To create the signal, in the void loop, use *pwmWriteHR(pin\_number, duty cycle \*2\*32768)* to first define the duty cycle of the PWM signal on the output pin and then use *SetPinFrequencySafe(pin\_number, frequency)* to set the square wave’s frequency.

Integrate the following code with your oscilloscope template.

#include <PWM.h> //PWM librarey for controlling freq. of PWM signal

void setup() {

// put your setup code here, to run once:

InitTimersSafe();

}

void loop() {

pwmWriteHR(9, 32768); //Duty cycle 50% = 32768

SetPinFrequencySafe(9, 1);

Use pin.9 to supply your previous LED and resistor setup. Upload your sketch. Probe the output of the PIN-9 using the oscilloscope. Confirm your last problem in Step.4 is now resolved, and you can observe a square wave. Change the frequency from the pwmWriteHr code from 1 Hz, to 10 Hz, and 100 Hz. Is your LED blinking faster?

**Lab Submission: Record a ~1-minute video of your last step when modulating the LED blinking rate using the code below. Make sure to show me your blinking setup, your code, and your oscilloscope output.**

Congrats, guys, now we have all the ingredients we need to start our lab!