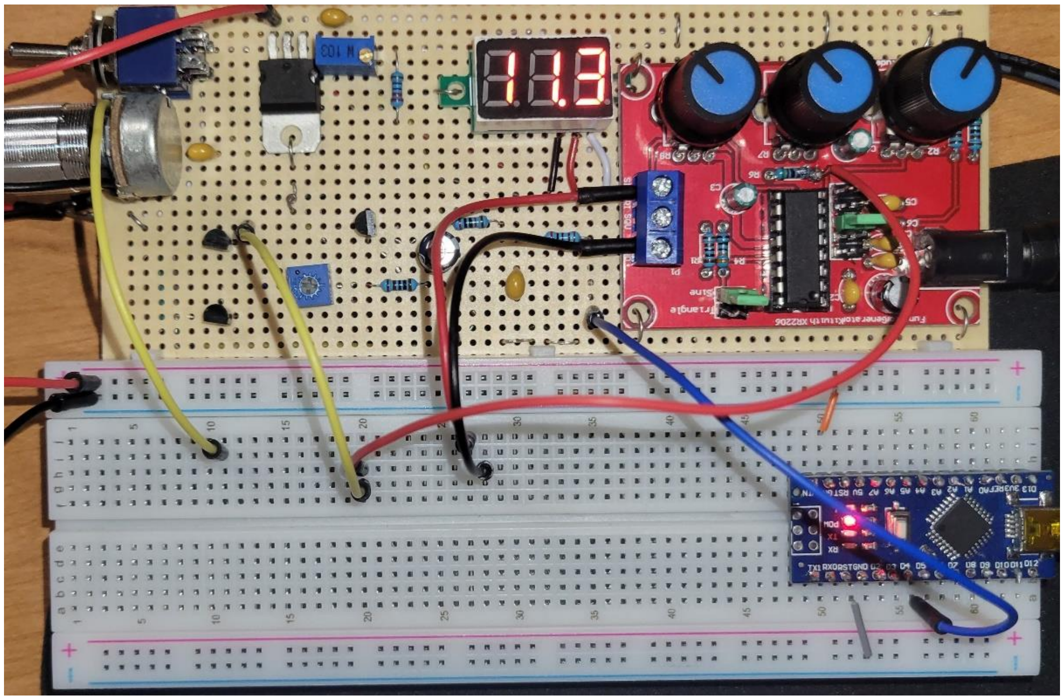
California State University, Channel Islands (CSUCI)
Department of Physics

Final Portfolio



Semester: Fall 2022

Student Name: Travis Anger
Student ID: 003170238

Table of Contents

LED Night light……………………………………………………3

DC Power Supply…………………………………………………...11

Transistor HFE………………………………………………………16

Arduino Controlled DC Power Supply………………………………27

Function Generator…………………………………………………..35

Appendix……………………………………………………………50

Objective:

Through using an average 12-volt, 1 amp wall wart, create a LED night light that can be turned brighter or dimmer through the use of a potentiometer. Be sure to find the V Thevenin, R Thevenin, Forward voltage and the cost per month in Kilowatt Hours.

Method:

Finding the I-V curve and Vf and I:

The LED I-V curve was found using a solderless breadboard creating a circuit that resembles Figure 1. The black portion of the voltmeter was set in between the 1k resistor and the LED light shown with the dashed lines in the figure. The red portion of the voltmeter was moved between the LED and the resistor as shown and the voltages were measured five times to create a curve in MATLAB

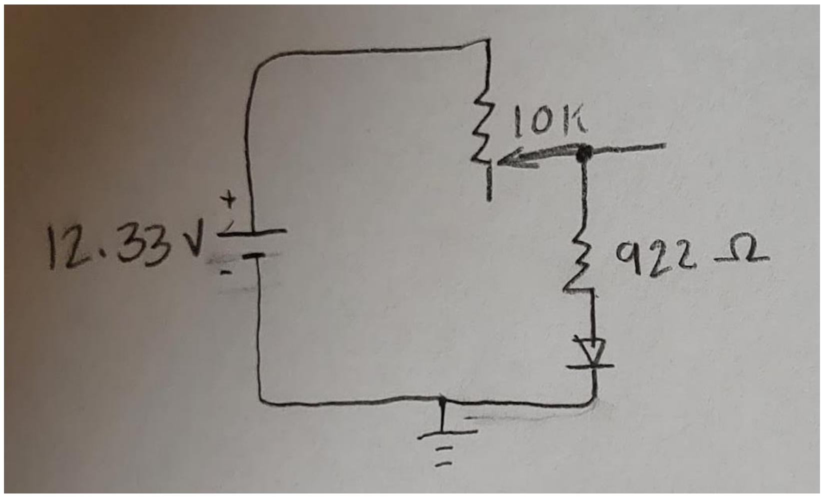
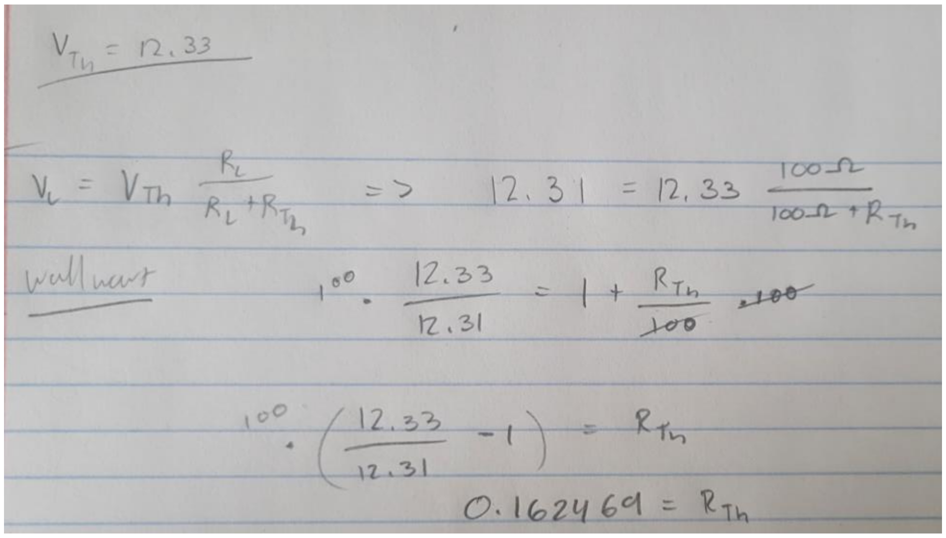


Figure 1 Schematic Diagram

Finding the wall warts Characteristics:

In order to find V thevenin I Measured the open circuit's voltage of the wall wart by simply applying the ground "COM" portion of the multimeter to the ground wire and the red V**Ω** port to the positive end.

In order to find R thevenin I used the voltage divider equation $V_L = V_{th}\ (\frac{R_L}{R_L+R_{th}})$ which is shown below in Figure 2



$V_{Th} = 12.33$

$V_L = V_{Th} \frac{R_L}{R_L + R_{Th}} \Rightarrow 12.31 = 12.33 \frac{100\Omega}{100\Omega + R_{Th}}$

wall wart

$100 \cdot \frac{12.33}{12.31} = 1 + \frac{R_{Th}}{\cancel{100}} \cancel{\cdot 100}$

$100 \cdot \left(\frac{12.33}{12.31} - 1\right) = R_{Th}$

$0.162469 = R_{Th}$

Figure 2 Finding Rth

My Rth in this case will be 0.16

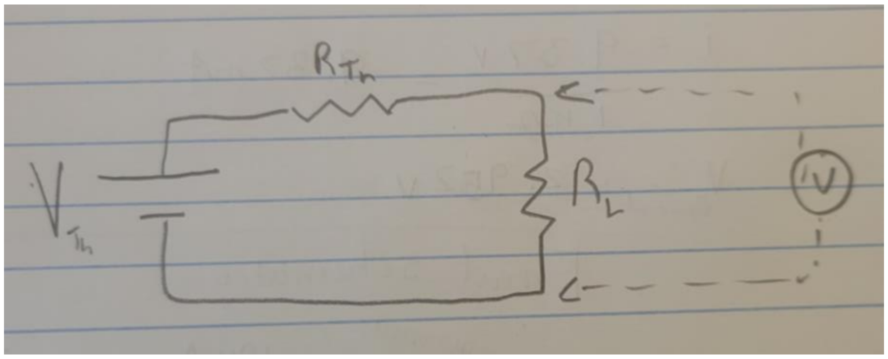


Figure 3 Measuring with Voltmeter

Designing the circuit:

Since I am creating a circuit with one LED, there must be a circuit created that meets the LED's forward voltage of 2 volts while operating at a current of 10 mA. So through the use of Ohm's Law I am able to calculate the exact type or resistor I need for my circuit.

In order to get a better reading to create the I-V curve properly, a potentiometer will need to be added to the circuit in series in order to gather more i and Vf data.

My circuit has a singular Red LED attached to one resistor (RL) and a 10k potentiometer. In order to find the proper resistance of resistor RL

Construction Techniques:

In order to create said circuit I need a wire cutter, wire stripper and a soldering iron with solder. First, I cut off the male DC connector from the wires that it is attached to save for a later lab. I did give the male connector 8 inches of wire so there is some room to work with in the later function generator lab. Next, I cut the positive wire shorter than the grounded wire to allow for each wire to be easily recognizable without having to look for the long-dashed lines on the ground portion. I then strip each wire and twist each respective inner copper wire together. After that I get two wire jumpers (red and black for differentiation) and solder them to their proper wires from the wall wart, red(W1) for positive and black(W2) for ground as seen in Figure 3.

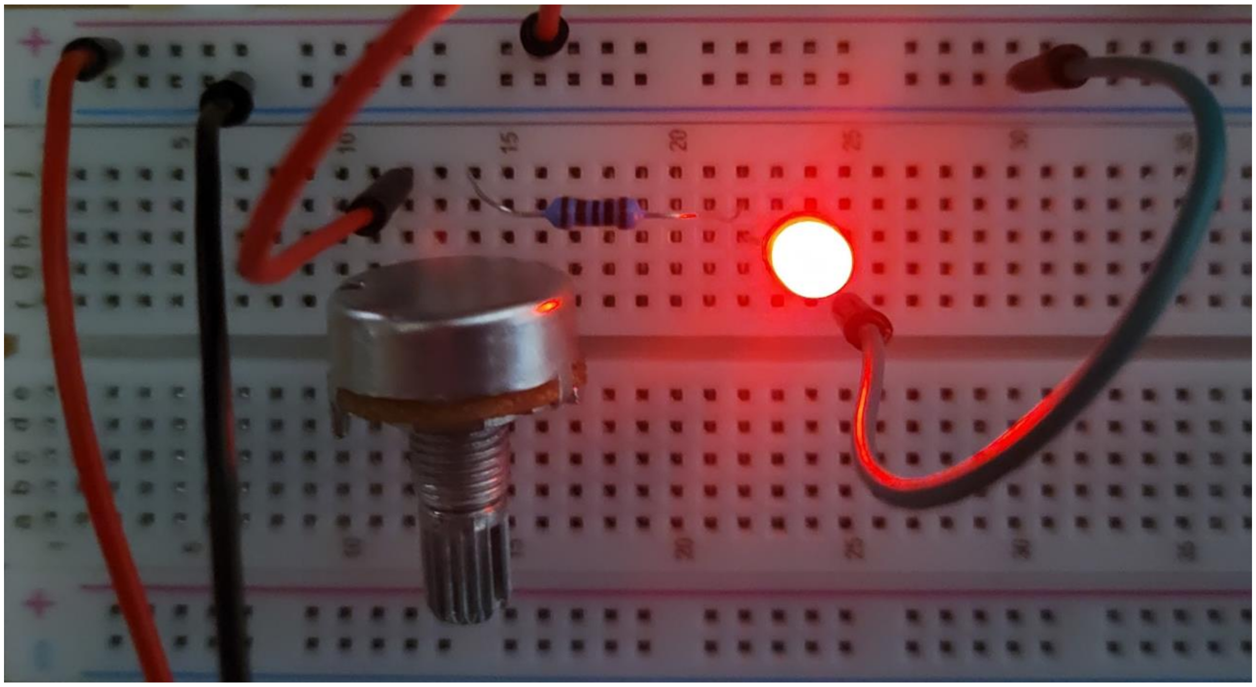


Figure 4 Prototype on solderless breadboard

Figure 4 is a simple prototype before I solder the desired resistors to the wall warts wires.

Results:

Finding the I-V Curve

In order to create an I-V curve of the LED, I need to record the volts going across just the LED and then the volts that go across both the 1k resistor and the LED.

Vled	**Vf**
0	**0**
1.796	**2.755**
1.816	**3.308**
1.839	**4.221**
1.898	**8.37**
1.940	**12.30**

The above data was the recorded values from the LED and the values of Vf are from the volts going across the resistor and LED. all of the values were put into the MATLAB code below in order to plot the I-V characteristic curve for the Red LED

```
%% measuring a diode I-V curve
Vd = [0  1.796 1.816 1.839 1.898 1.940 ]; % volts across red led
Vrd = [0 2.755 3.308 4.221 8.37 12.30 ];% volts across both R and led
Vr = Vrd - Vd;
R = 922; % ohms
i = Vr/R * 1000; % mA
plot(Vd,i, 'o-')
xlabel('LED forward voltage / V');
ylabel('LED current/mA');
grid
title('Red LED')
```

The code, from MATLAB, takes in the values from Vrd and Vd and subtracts them in order to get Vr. The reason that Vr is needed is because if we divide that by the resistance of the resistor and multiply by 1000, we get mA, which is exactly how the I-V curve is graphed in the vertical direction. The plot function is then called which plots the Volts across the LED with respect to the mA across the LED with ‘o’ dots on the graph to signify where the data points are on the graph for a visual aid.

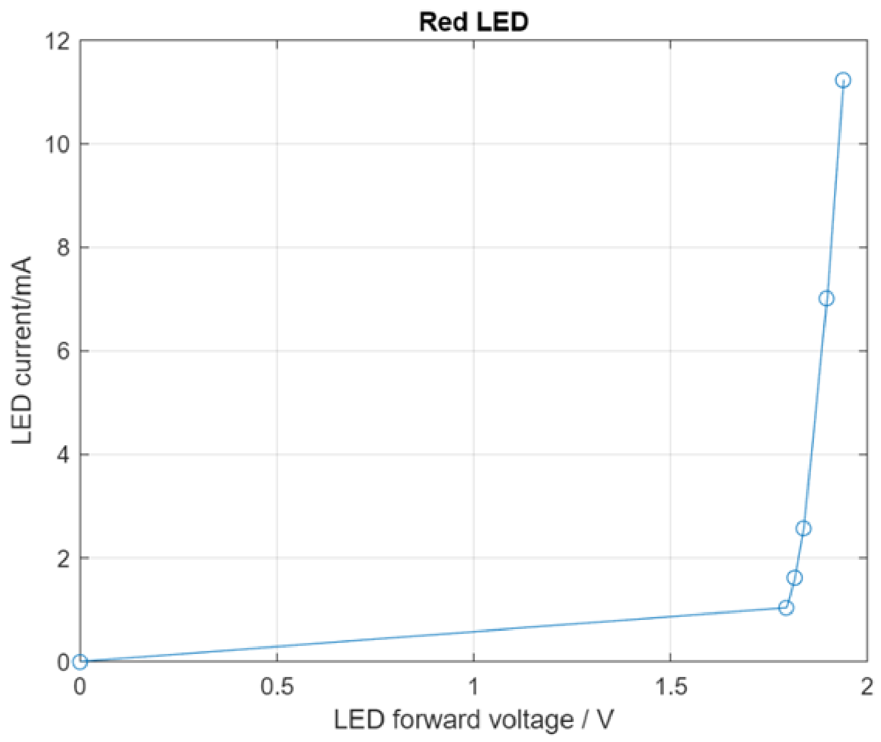
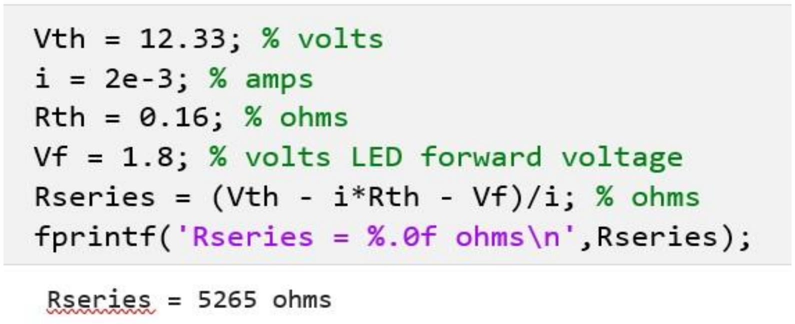


Figure 5 LED Characteristic Curve

Prototyping:

For my desired light brightness, I decided to go around 1.8V and 2 mA since that is on my I-V curve plot.



```
Vth = 12.33; % volts
i = 2e-3; % amps
Rth = 0.16; % ohms
Vf = 1.8; % volts LED forward voltage
Rseries = (Vth - i*Rth - Vf)/i; % ohms
fprintf('Rseries = %.0f ohms\n',Rseries);
```

```
Rseries = 5265 ohms
```

In order to validate my design, I must find the mA needed and the power for the resistors. Given that my Rseries is 5265 ohms I have to use around 5k ohms in series with each other when I solder them to the wall wart's wires.

```
Rseries = 0.922*10/14.7*1e3; % ohms, 10k and 1k in parallel
Rseries = 5.265e3; % measured
Vrseries = 10.22; % volts measured
Vled = 1.682; % volts measured
i = Vrseries / Rseries * 1000 % mA
```

```
i = 1.9411
```

```
% power calculations
Presistors = Vrseries^2 ./ [4.7e3 10e3] * 1e3 % mW
```

```
Presistors = 1×2
   22.2231   10.4448
```

```
Pled = Vled * i % mW
```

```
Pled = 3.2650
```

All of these components are well under the dangerous levels of 250mW so there is no need to be afraid of this LED or resistor to burn out.

Now if there wa such a short in my final schematic R series would have the full force of the voltage and would dissipate:

```
PshortedLED = Vth^2 / Rseries * 1000 % mW
```

```
PshortedLED = 28.8754
```

Final schematic and execution:

Shown below is the final design of the LED night light with a 10k and 1k resistor. I had decided on those two since the light from the LED was less intrusive and bright than the rest when testing it with the potentiometer on the solderless breadboard. For there to be no shorting out, I applied some blue shrink wrap around the resistors and soldered copper wire. After soldering I used the soldering iron’s heat to shrink the shrink wrap around the resistors and wires.

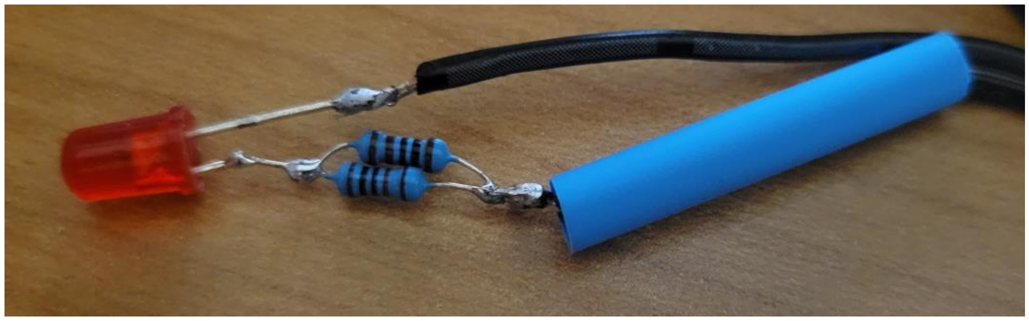
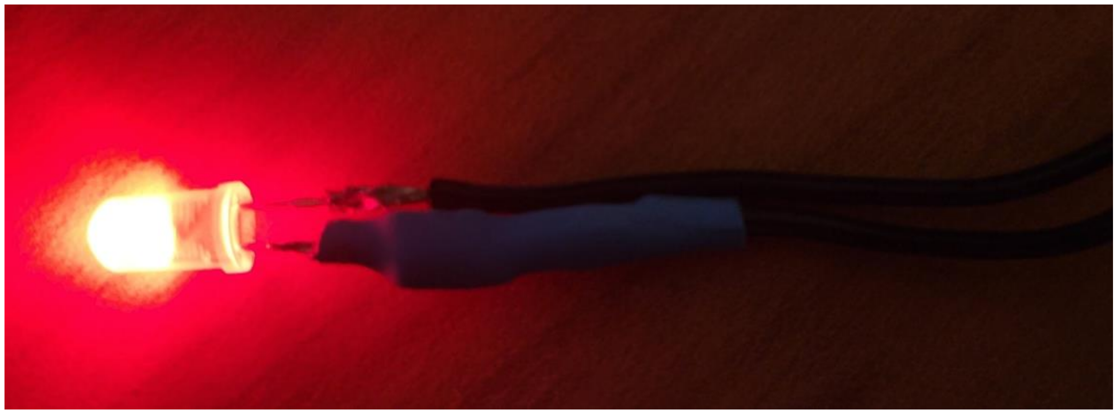



Figure 6 Before and after shrink wrapping

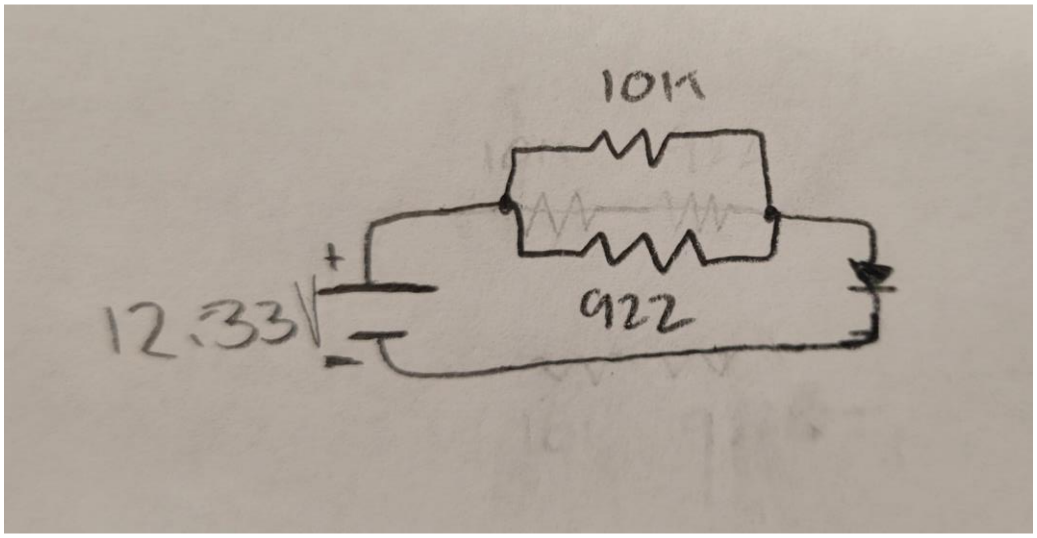


Figure 7 Final schematic

- The final schematic is shown as well to further clarify. The reason R2 says 922 in particular is because I had measured that value with the multimeter when I was doing the calculations.

Discussion:

Everything in the experiment worked to its desired and expected outcome. The solder had some trouble properly adhering to the copper wire and the resistor legs which added some unnecessary time to create the circuit but that was easily fixed within minutes.

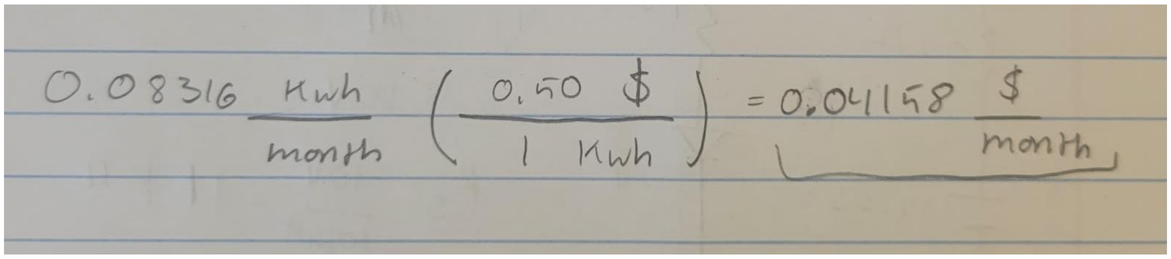
Theory and measurements both seemed to agree on everything, everything was measured properly, and the calculations all came out correct according to the given $V_{th}$and $R_{th}$from the wall wart.

Power consumption of this LED, if measured in months would be estimated as follows:

The total power created in this circuit can be defined by watts law as 12.33V * 9.37mA = 115.5mW.

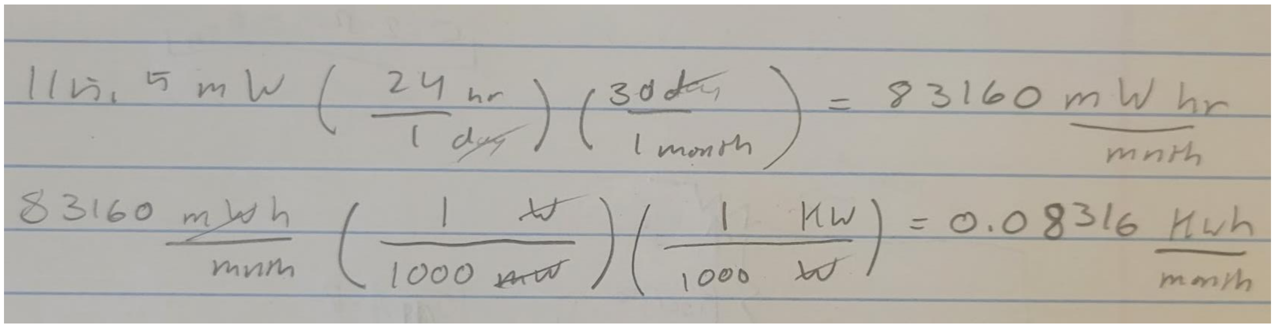
Now transferring this into Kilowatt Hours can be defined below:

To find a price for this we would have to find the average price for kilowatt hours which would be around $0.50 per:



$$0.08316 \frac{\text{Kwh}}{\text{month}} \left(\frac{0.50\ \$}{1\ \text{Kwh}} \right) = \underbrace{0.04158 \frac{\$}{\text{month}}}$$

There is a fine possibility of there being a way to reduce the cost, possibly though using a less taxing LED light or potentially using a weaker wall wart making the KWH less and making it cheaper to maintain.



$$115.5\ \text{mW} \left(\frac{24\ \text{hr}}{1\ \cancel{\text{day}}} \right) \left(\frac{30\ \cancel{\text{day}}}{1\ \text{month}} \right) = 83160 \frac{\text{mW hr}}{\text{mnth}}$$

$$83160 \frac{\text{m}\cancel{\text{W}}\text{h}}{\text{mnth}} \left(\frac{1\ \cancel{\text{W}}}{1000\ \cancel{\text{mW}}} \right) \left(\frac{1\ \text{KW}}{1000\ \cancel{\text{W}}} \right) = 0.08316 \frac{\text{Kwh}}{\text{month}}$$

Objective:

Using a pcb board, a switch, a 10k pot, a 3-digit display, and an LM317, create an adjustable DC power supply that displays the voltage on the 3-digit display

Method:

Design

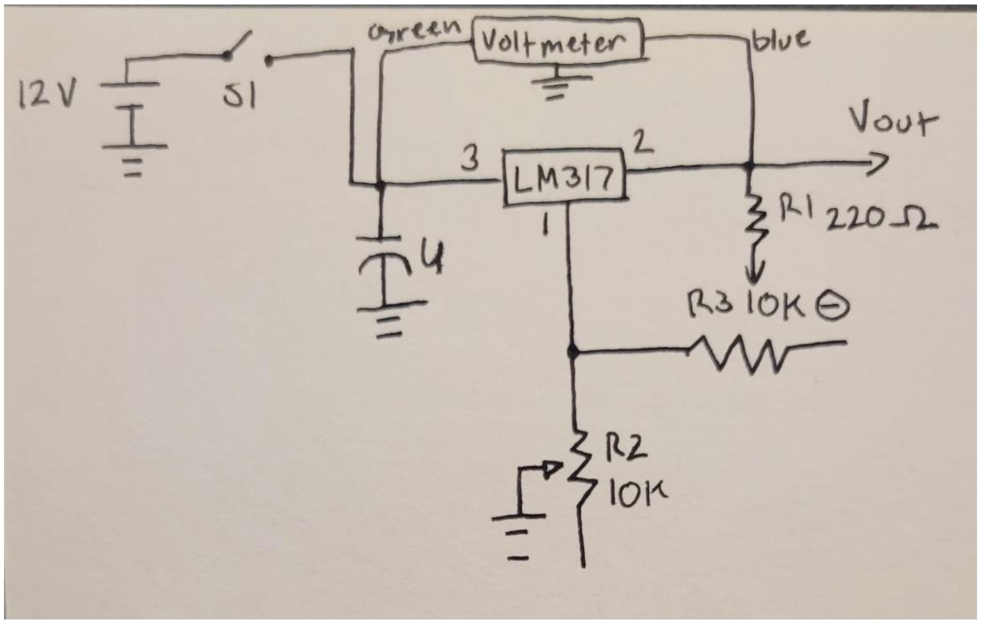


Figure 1 Schematic

Connecting the DC power supply to the pcb bard takes a lot of intricate work. Initially I connected the wall wart cables to the switch as seen in the Figure 1 schematic. A 4.7 microfarad capacitor was added to the positive and ground power rails as seen in the Figure 2 prototype. The LM317 has multiple connections so starting with pin 1, the 10k potentiometer is connected via its first pin while being connected to ground on its second pin, as shown in Figure 1. A 10k trim pot is then added and connected to the potentiometer via the trim pots second (middle) pin, this is all connected to LM317 pin 1.

Pin 2 has a 220-ohm resistor connected to pin 1 of the trim pot, while there is a cable jumper connecting to the voltmeters Voltage/Hz reader.

Pin 3 has the 4.7 microfarad capacitor, with the switch connected as stated prior, connected via a green wire as shown in figure 2. There is a green wire that is connected on the power rail that attaches to the COM portion of the voltmeter

Testing

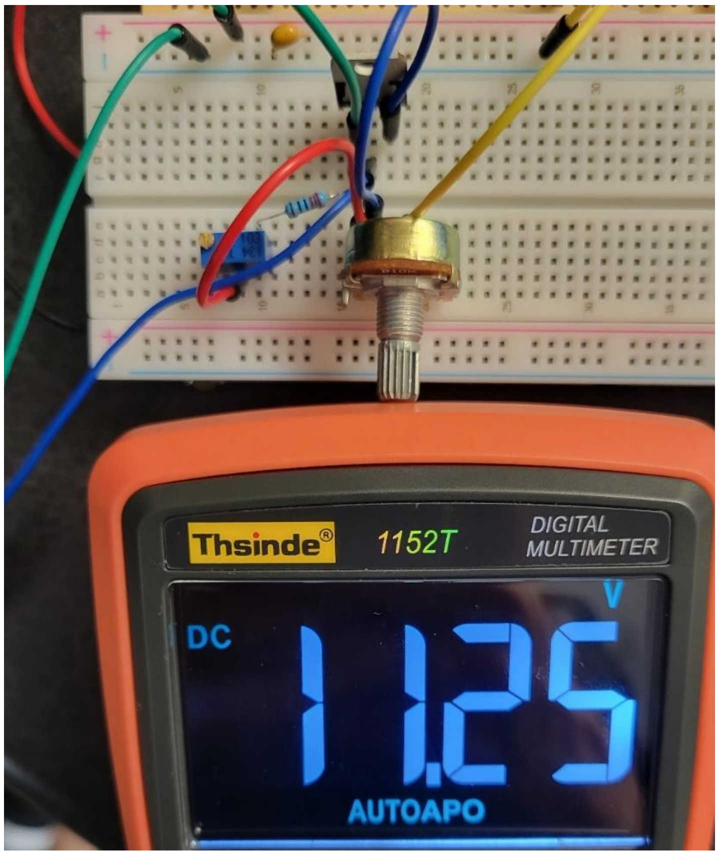


Figure 2 Breadboard Prototype

As shown in Figure 2, the prototype was properly executed on the breadboard from the schematic and gave out a maximum voltage of 11.25V when I turned the 10k potentiometer to its maximum setting.

Results

Implementation

Taking all that has been done on the breadboard, it is now time to transition all of that over onto the pcb permanently

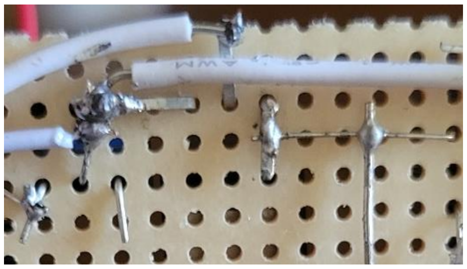


Figure 3 PCB underside

Figure 3 shows a neatly soldered implementation of the prototype as shown in Figure3.

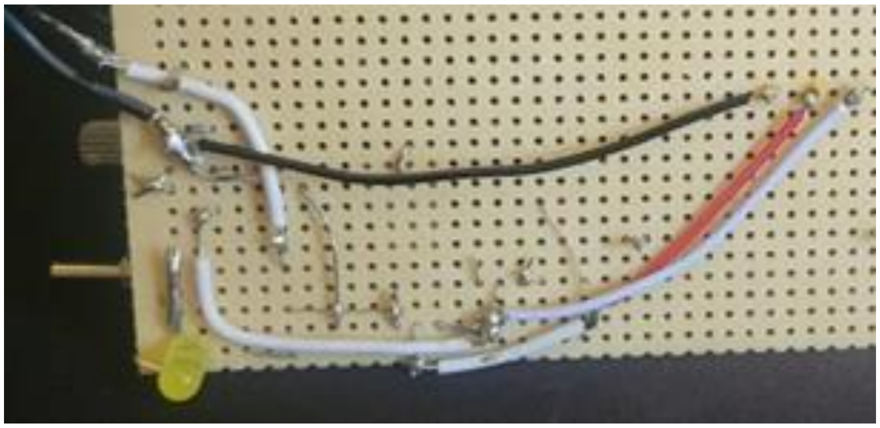


Figure 4 3-digit display connected

Figure 4 has the 3-digit display soldered to the pins on the LM317 where the voltmeter was once before as shown from the schematic diagram.

Minimum and Maximum

Since the 3-digit display cannot go past 3 digits I set up a voltmeter to measure how much voltage is actually being put through the circuit

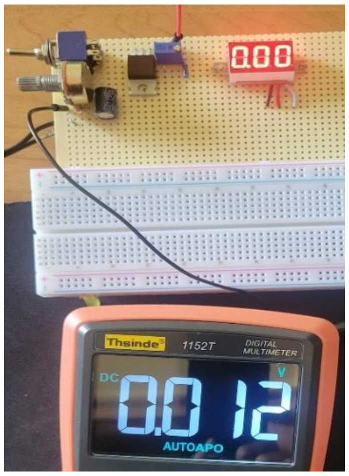
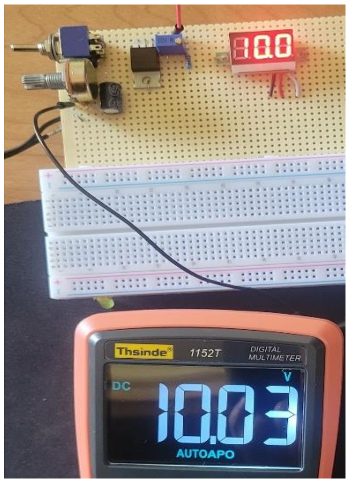


Figure 5 Minimum Voltage set to 0 (Shown left)

Figure 8 Maximum Voltage set to 10 (Shown right)6



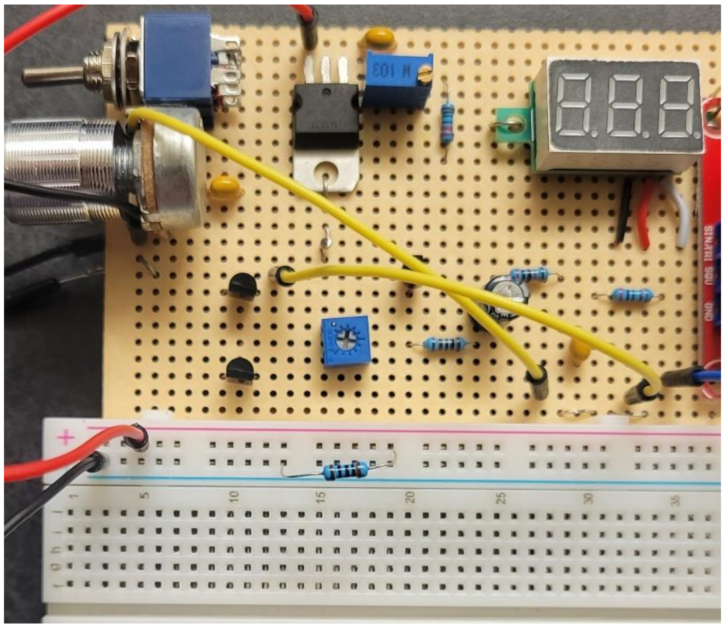
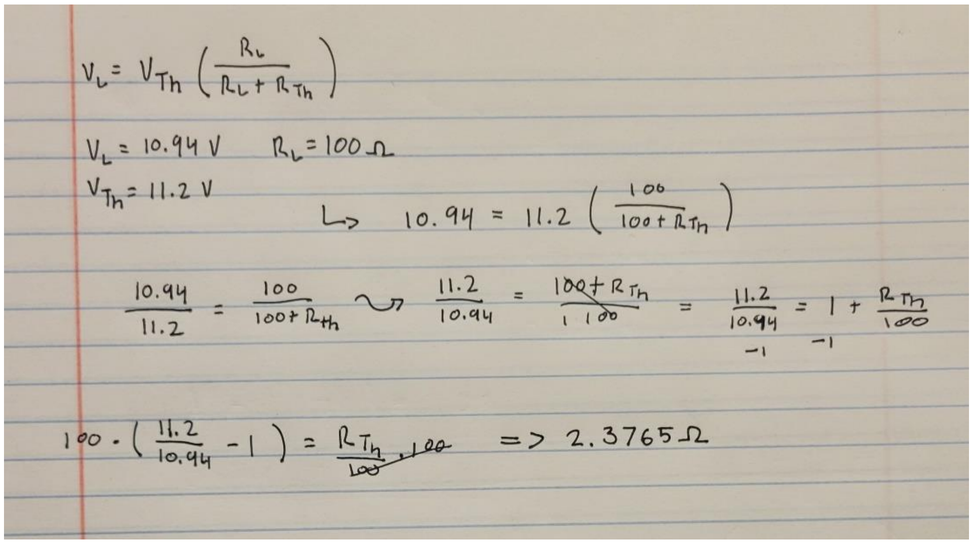


Figure 7 100-ohm resistor attached to circuit

Shown in figure 7, the 100-ohm resistor applied on the breadboard however, it is turned off because the resistor would have gotten too hot and could have burned out. The recorded voltage drop on the voltmeter was 10.94V from the maximum 11.2V that I have normally when the circuit is turned on. This gives the overall power supplies Thevenin resistance value of 2.3765-ohms as shown in Figure 8.



$$V_L = V_{Th}\left(\frac{R_L}{R_L+R_{Th}}\right)$$

$V_L = 10.94\ V$ $\quad R_L = 100\,\Omega$

$V_{Th} = 11.2\ V$

$$\hookrightarrow\ 10.94 = 11.2\left(\frac{100}{100+R_{Th}}\right)$$

$$\frac{10.94}{11.2} = \frac{100}{100+R_{th}} \rightsquigarrow \frac{11.2}{10.94} = \frac{\cancel{100}+R_{Th}}{1\ \cancel{100}} = \underset{-1}{\frac{11.2}{10.94}} = \underset{-1}{1} + \frac{R_{Th}}{100}$$

$$100\cdot\left(\frac{11.2}{10.94}-1\right) = \frac{R_{Th}}{\cancel{100}}\cdot\cancel{100} \Rightarrow 2.3765\,\Omega$$

Figure 8 Calculated Rth

Discussion:

Throughout the building of the variable DC power supply the soldering and the passing of wires through the holes on the PCB board proved to be the most difficult. There was one small copper wire from the 3-digit display that bent up instead of through the hole on the board making it to be quite annoying and shorting the display however, that was easily averted once I had twisted and soldered them. To my surprise the permanent board worked on the first try and nothing shorted out. Thankfully I had done the prototype so there was no need to trouble shoot. For a message to my future self, I would tell him to be very diligent on the twisting of the copper wire when stripping any insulated wire. Do not have random wires crossing without insulation and make sure that everything is soldered if it is stacked on top of other wires, it may look soldered and in place but it could very well not be connected and that is why it is not working.

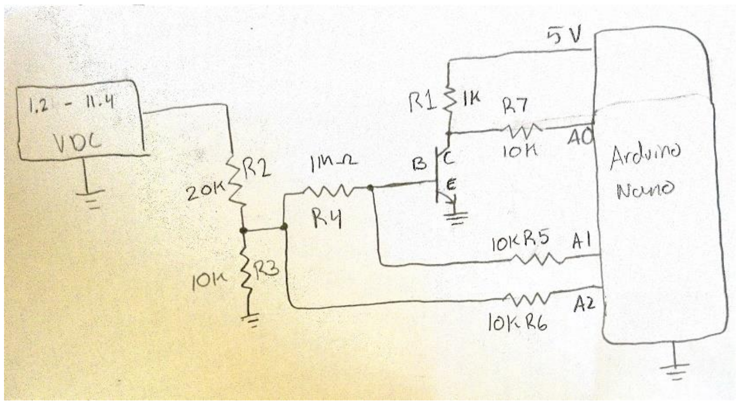
Transistor HFE

Objective

Measure the hfe of an NPN transistor.

Methods

Hardware Interface



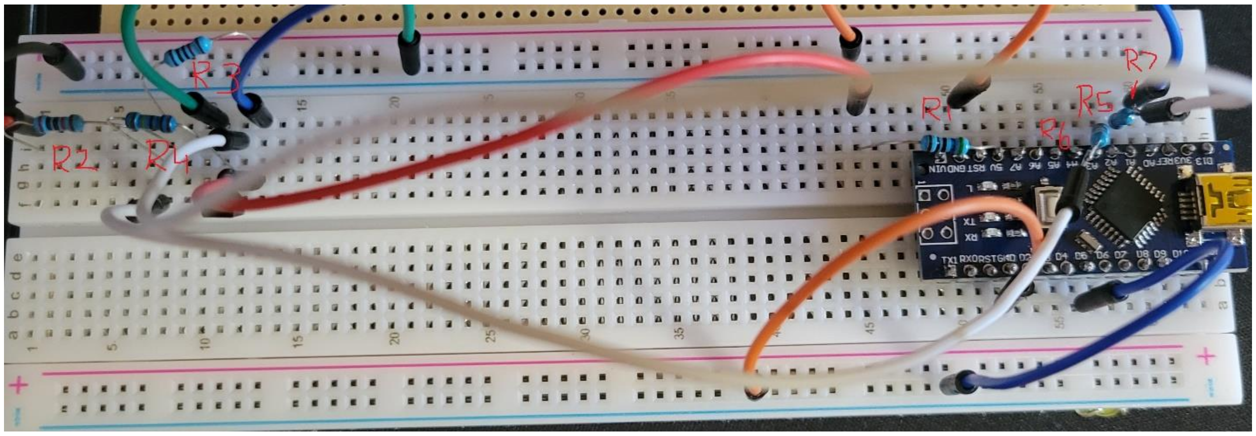
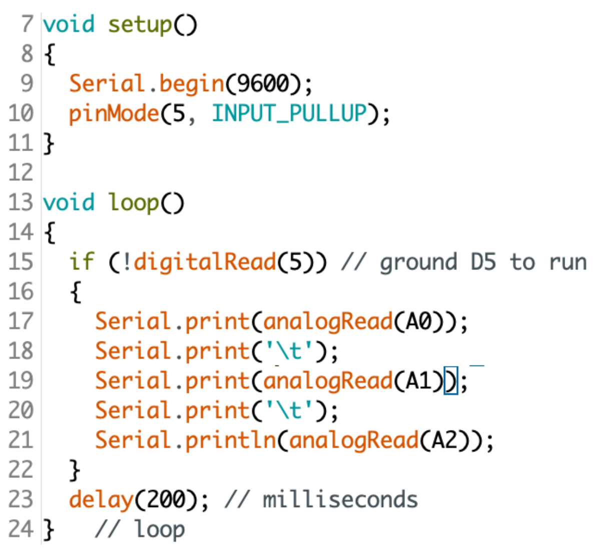


Figure 1. Schematic and breadboard of the automated Hfe circuit.

The transistor for this experiment was an S8050.

Software Interface

Same Arduino code as prior transistor experiments however I added another 'Serial.print function to allow for another pin receptor to be read in the Arduino pop out window. Pin D5 is grounded to toggle data logging. When grounded, data was printed to the Serial Monitor at roughly 5 times per second.



```
void setup()
{
  Serial.begin(9600);
  pinMode(5, INPUT_PULLUP);
}

void loop()
{
  if (!digitalRead(5)) // ground D5 to run
  {
    Serial.print(analogRead(A0));
    Serial.print('\t');
    Serial.print(analogRead(A1));
    Serial.print('\t');
    Serial.println(analogRead(A2));
  }
  delay(200); // milliseconds
}   // loop
```

Data Interface

Data from the Serial Monitor was copied/pasted below into variable `data`, and converted to units of volts. Resistances (measured with the DVM ohmmeter) converted these voltages to currents as shown below in the MATLAB code interspersed with the Results.

Results

```
vref = 4.59; % volts
Rb = 1e6; % R4, ohms
Rc = 998; % R1, ohms
data = [... A0, A1, A2
1023 118   89
1023 118   91
1021 119   113
1015 121   158
1006 126   206
996  125   248
990  127   284
976  128   335
966  128   395
956  129   455
943  130   500
935  130   543
931  131   586
911  131   642
899  132   702
890  132   761
875  133   807
882  133   807
879  133   811
878  132   811
879  133   810
876  133   812


] * vref / 1023; % now in volts
plot(data);
xlabel('sample #'); ylabel('Volts');
legend('A0','A1','A2');
title('quick look at "raw" data')
```

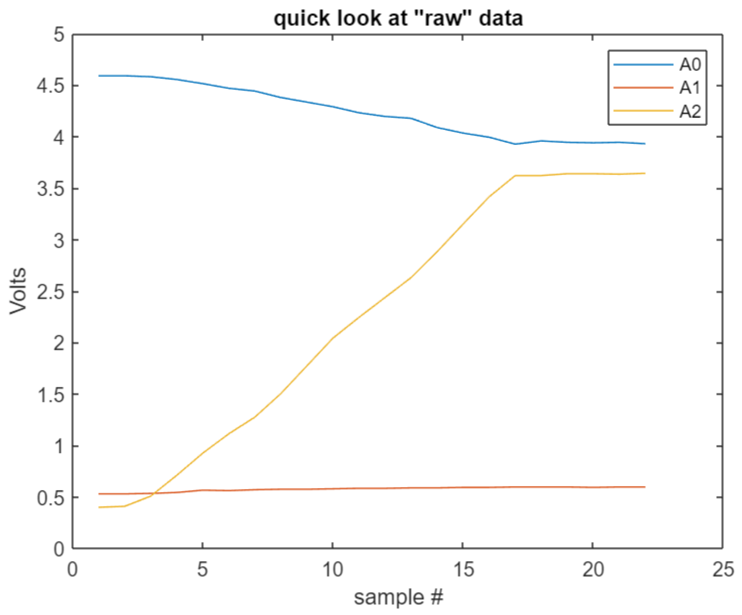


Figure 2. First look at raw data from the Arduino.

Vbe = A1 hardly moves since it acts like a forward biased diode in conducting mode. The base current must be changing, but the base voltage (orange) hardly does, because the base-emitter diode has a nearly vertical i-v curve. For A2, it increases which means that the base current increases as well, so the collector current should, in theory, increase which causes more V = icRc voltage drop across Rc which then causes Vce to drop:

KCL says Vref = Vce + icRc

So if the 2nd term increases, the first one must decrease proportionally, and the blue curve above does that.

Now the characteristics of the transistor (the key voltages and currents) must be measured and I do so by:

```
vce = data(:,1); % collector-emitter voltage
vbe = data(:,2); % base-emitter voltage
```

```
vin = data(:,3);
ib = (vin-vbe)/Rb * 1e6; % base current in uA
ic = (vref-vce)/Rc * 1000; % collector current in mA
plot(ib, ic,'d-')
grid; xlabel('base current/\muA');
ylabel('collector current/mA')
```

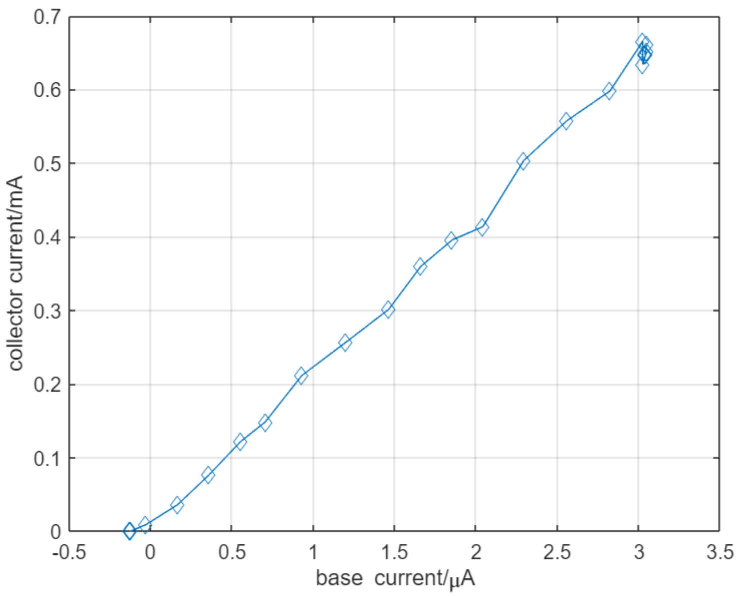


Figure 3. Relation between base and collector currents (with Rc = 1k) is very linear above the lowest currents (bottom left).

Seemingly from the data I have recorded I can get a slope from it. I will exclude the lowest points because of repetition and possibly ruining the linear slope and points where ib > .1uA. I am also plotting mA/uA, so I need to multiply the slope by 1000:

```
p = polyfit(ib(ib>.1), ic(ib>.1), 1); % help polyfit for more info
fprintf('beta = %.1f\n', p(1)*1e3)
```

```
beta = 214.1
```

To explore higher collector currents I must use smaller resistor, I chose one around a 560 ohm:

```
ib1k = ib; ic1k = ic; beta(1) = p(1)*1e3;
Rc = 559.8; % ohms
data = [...
1023 121   89
1023 115   96
1016 121   168
1013 123   219
1008 124   254
1004 126   294
997  127   326
994  127   360
991  128   389
989  128   424
982  129   461
981  130   490
977  129   517
973  130   552
969  131   585
964  131   629
958  131   673
953  132   718
946  132   763
943  133   805
939  133   807
943  133   809
940  133   809
941  132   810

] * vref / 1023; % now in volts
vce = data(:,1); % collector-emitter voltage
vbe = data(:,2); % base-emitter voltage
vin = data(:,3);
ib = (vin-vbe)/Rb * 1e6; % base current in uA
ic = (vref-vce)/Rc * 1000; % collector current in mA
plot(ib, ic,'d-')
grid; xlabel('base current/\muA');
ylabel('collector current/mA')
```

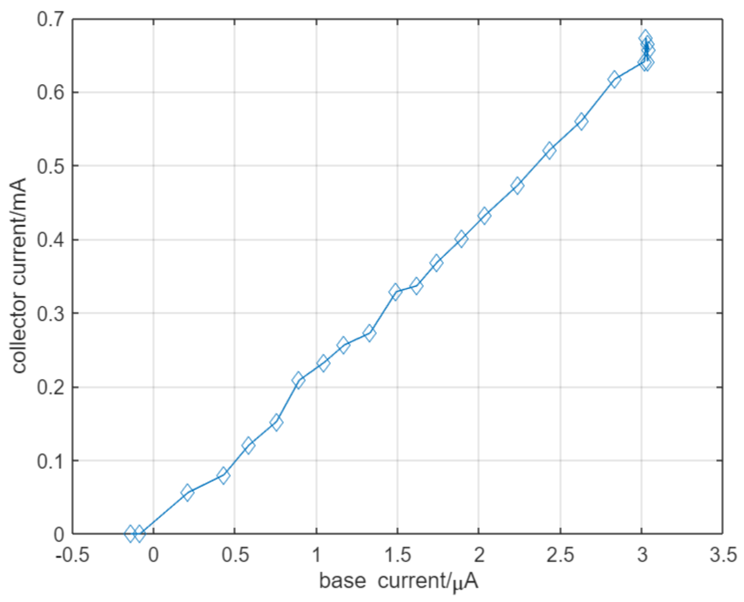


Figure 4. Switching to $Rc = 681\Omega$ appears to change nothing.

the beta is:

```
p = polyfit(ib(ib>.1), ic(ib>.1), 1); % help polyfit for more info
fprintf('beta = %.1f\n', p(1)*1e3)
```

```
beta = 216.3
```

There seems to be a very consistent theme happening along these curves and plots. Which makes me believe that the collector resistor doesn't affect this curve. The slope is a *characteristic of the transistor*. For posterity I will change out the resistor into a 10kohm resistor:

```
ib681 = ib; ic681 = ic; beta(2) = p(1)*1e3; % save the former results
Rc = 9.93e3; % collector resistor
data = [...
1023 128   89
1022 130   93
1009 129   114
948  129   150
886  129   184
829  129   211
785  130   234
732  130   258
688  129   282
635  129   308
587  129   334
548  129   354
503  129   377
461  129   397
410  129   418
377  129   436
343  130   455
291  129   477
228  129   510
175  129   535
126  130   558
80   130   580
48   130   597
33   130   616
29   130   633
27   130   660
26   130   680
24   130   705
22   130   730
22   130   758
21   130   779
20   130   799
20   130   810


] * vref/1023; % volts
vce = data(:,1); % collector-emitter voltage
vbe = data(:,2); % base-emitter voltage
vin = data(:,3);
ib = (vin-vbe)/Rb * 1e6; % base current in uA
ic = (vref-vce)/Rc * 1000; % collector current in mA
plot(ib, ic,'d-')
grid; xlabel('base current/\muA');
ylabel('collector current/mA')
```

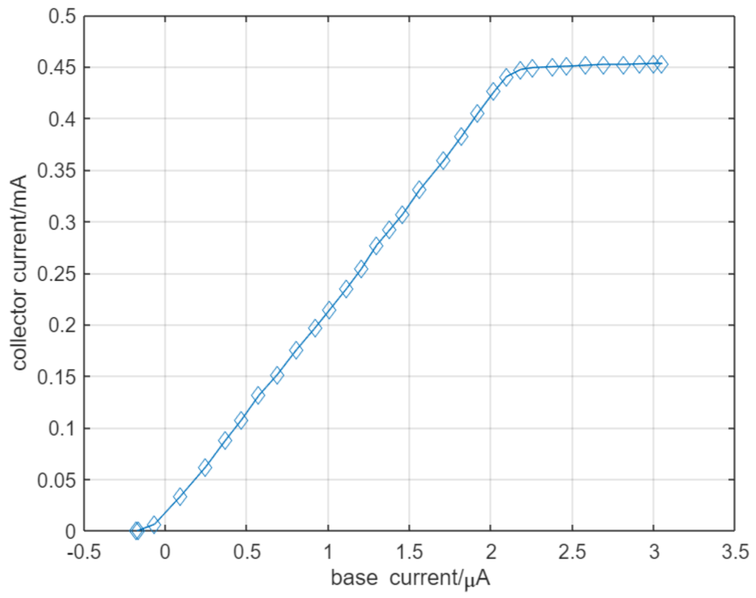


Figure 5. Collector vs. base currents with $Rc = 10k\Omega$, which causes the BJT to saturate around $i_C \;\; .45mA$.

I superimposed these 3 plots to have a better understanding of what happens:

```
ib10k = ib; ic10k = ic; % for consistency
plot(ib681,ic681, ib1k,ic1k, ib10k, ic10k);
grid;
xlabel('base current/\muA');
ylabel('collector current/mA');
legend('681\Omega','1k','10k','location','best');
title('varying R_C');
```

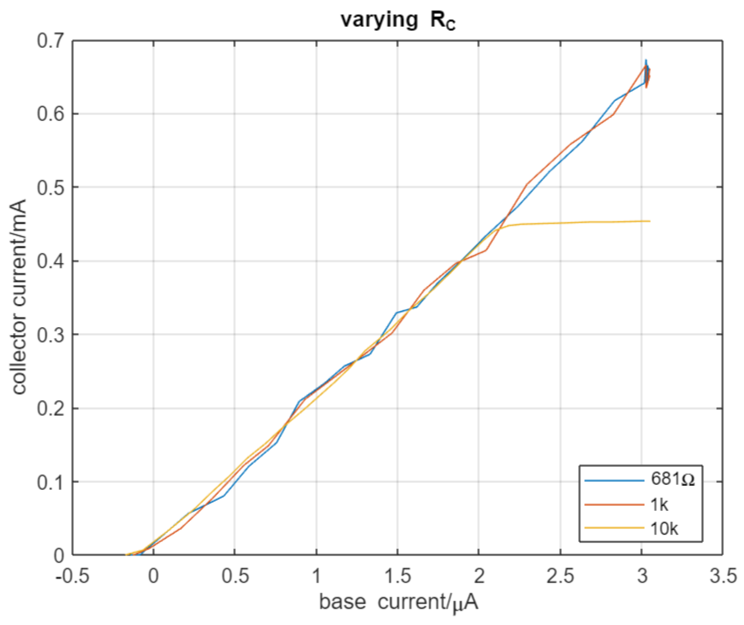


Figure 6. Superposition of Figs. 3-5, all values of Rc show similar slope before saturation.

```
indx = ic > .1 & ic < .8*max(ic);
p = polyfit(ib(indx), ic(indx), 1);
beta(3) = p(1)*1e3;
fprintf('beta = %.1f\n', beta(3))
```

```
beta = 202.6
```

The plateau value was expected from saturation, when the collector-emitter resistance approaches a minimum near zero. Hence ic ~ Vcc / Rc =

```
vref / Rc * 1000 % mA
```

```
ans = 0.4622
```

compared to the graph:

```
max(ic)
```

```
ans = 0.4532
```

Discussion

These curves are ic vs ib curves since transistors have 3 terminals, and hence 2 circuits (with a common emitter). BJTs are current-controlled-current devices and at the bottom left of Fig 6, the transistor is in **cut-off**, with both currents ~0. Then the BJT enters the **active region** characterized by $i_C = \beta i_B$, the linear, constant slope = β, until the BJT becomes **saturated**, as happened with Rc = 10k at ic ~ .45mA.

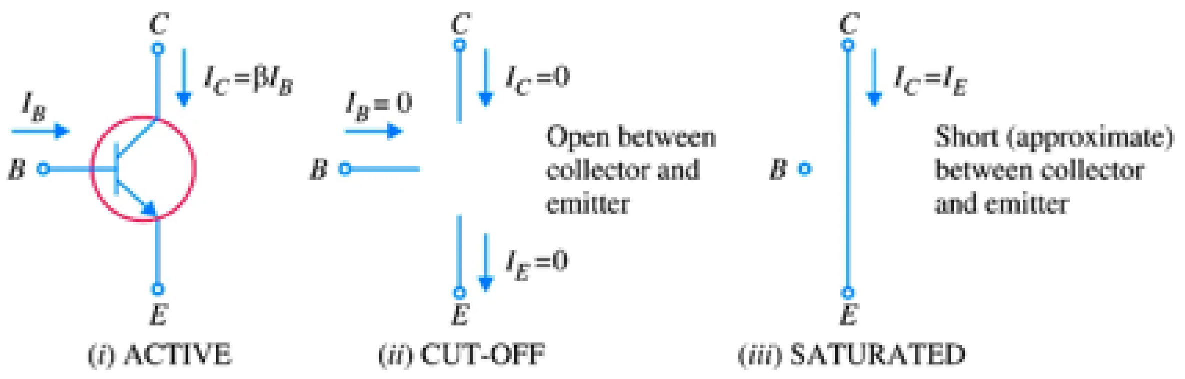


Figure 7. Three states of a BJT.

The value of β is a characteristic of the transistor, and relatively independent of the circuit. Many factors affect β, including geometry, doping and structural composition, and temperature. Unlike resistors, who's values are specified within a narrow range, the range of β is often extremely wide. Good designs of transistors work as long as β > minimum value, and incorporate feedback or other mechanisms so the circuit's critical behaviors are unaffected by β's particular value. Below shows what my β value was:

```
fprintf('beta = %.0f±%.0f\n',mean(beta), std(beta))
```

```
beta = 211±7
```

The Digital Abstraction

Digital circuits use the transistor's cut-off and saturated states to represent the abstract Boolean states of true and false, 1 and 0, or visa versa. But to transition from either Boolean state to the other, every transistor has to pass through it's active region. So all digital circuits are in fact analog circuits! this begs the quesion:

What makes digital circuits "digital", i.e., having only 2 discrete states (representing 0 and 1)?

Using a transistor in cut-off and saturation to represent the Boolean states 0 and 1 (or 1 and 0), in order to transition from cut-off to saturation or visa versa, the transistor has to go through the active region, where Ic takes on continuous, analog values. In other words, a transistor can't go from 0 to 1 without momentarily passing through 0.5 (and 0.6, 0.7321, etc. -- except those "analog values" don't exist in a digital world ... So if the voltages and currents in digital circuits are continuous and analog, what is it that makes "digital" circuits digital?

Answer:

This question is very thought provoking since I have never considered that idea of the values having to go through the 0.5 and 0.78432 values in the digital domain. Initially I had thought that it was almost like instantaneous on or off like a switch. But i have never considered this idea since I have only been recently getting into electronics.

Answer: A clock!!! All computers, microcontrollers, and digital systems have a clock (or multiple ones). The job of the clock is tell when it is safe to query the state of the transistors, after they've had time to cross the active region and settle into cut-off or saturation states. So digital circuits are analog *when nobody is looking*, and only when the clock "ticks" are the states queried for their (ideally settled, Boolean) values. Digital computers are queried in discrete (digital) time, compared to analog computers that not only have continuous variables, but their states evolve in continuous (vs. discrete) time. Is that an answer you expected? Is it right or wrong? Why? ...

My answer:

With the form of a clock, i suppose that makes sense and is somewhat close to what I had said. a clock can pertain to something that goes on and off in a quick succession as if it was going in a circle. I would have to do some more research and experimentation but so far this sounds correct to me.

Arduino-Controlled DC Power Supply

Objective:

Create an Arduino controlled DC power Supply.

Methods:

Hardware interface:

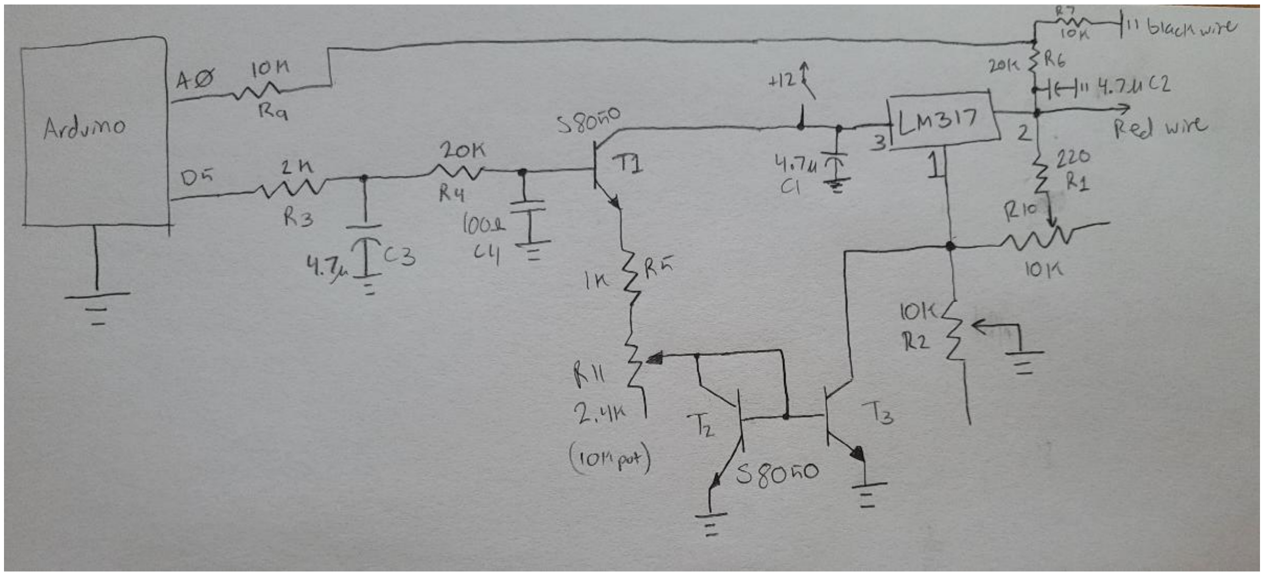


Figure 1 Schematic Diagram

The pwm filter is created in order to record volts and pwm at the same time and input it into the Arduino in order to be recorded in MATLAB. It records the voltage coming out of the power supply and goes through a few transistors, capacitors and resistors that then get inputted into D5 on the Arduino.

The emitter follower is a small transistor (S8050) that collects the voltage coming from the + side and then redirects it through the emitter into the small trim pot that we have adjusted via a small screwdriver. The way T1(the emitter follower) is changed is from that screwdriver onto the pot. It controls the experimental/A0 slope that is displayed on the graphs.

Software interface:

See Appendix [1]

Data interface:

MATLAB talks to Arduino through a file called 'oscope1.m' [1] which operates through a usb hub, in my case it is 'COM3'. The Arduino has the code above (Figure 3) uploaded into it and the oscope1.m adjusts the DC supply voltage automatically while MATLAB reads and records the voltage data in a matrix in order to be grasped at the end.

Calibration:

The pwm was measured by an intake of a0 on the nano, the Arduino spoke to MATLAB via its uploaded code all the while MATLAB was recording this into an array/matrix being able to be graphed. When it takes in the A0 voltage changes it can record the oscillation being controlled electronically from the Arduino. when graphed the poly fit function is able to graph the other mathematical estimations in order to get closer to the experimental data that is required.

Testing:

Collecting data is much harder given the fact that there is no discernable avenue to gather all of the voltage changes and paste them into the document, however that is the reason MATLAB and the Arduino talk to each other. It gets rid of the middleman being paper or the copy and paste function and allows for more streamlined automation of the lab. The error analysis given with the v command is able to be operated around with some changing of stiff values in the code below.

Results:

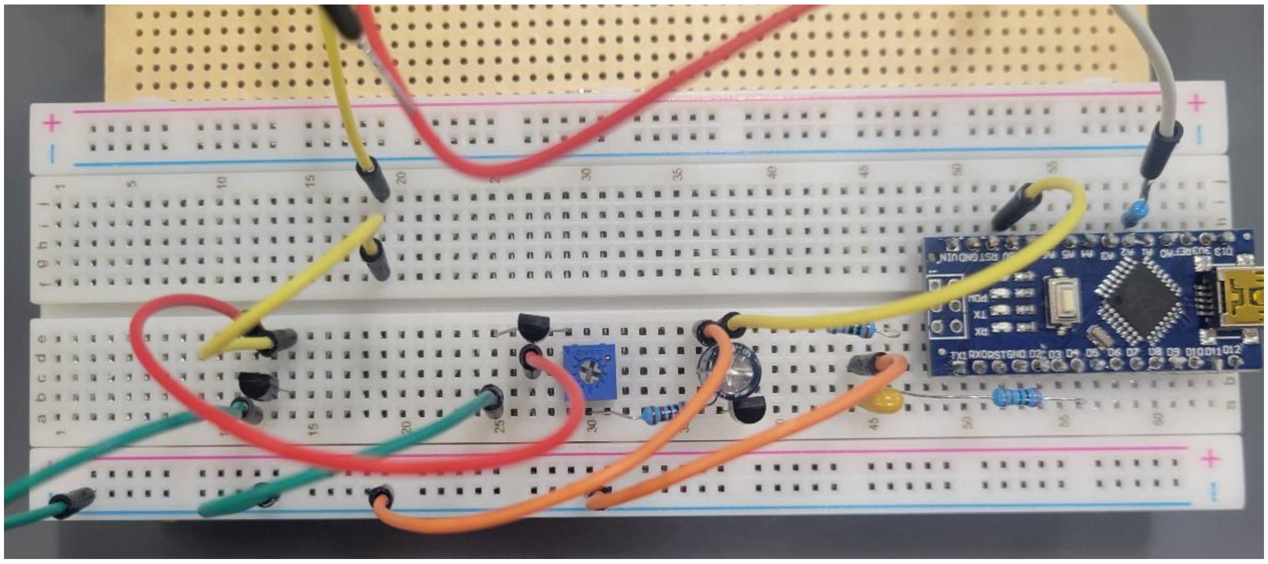
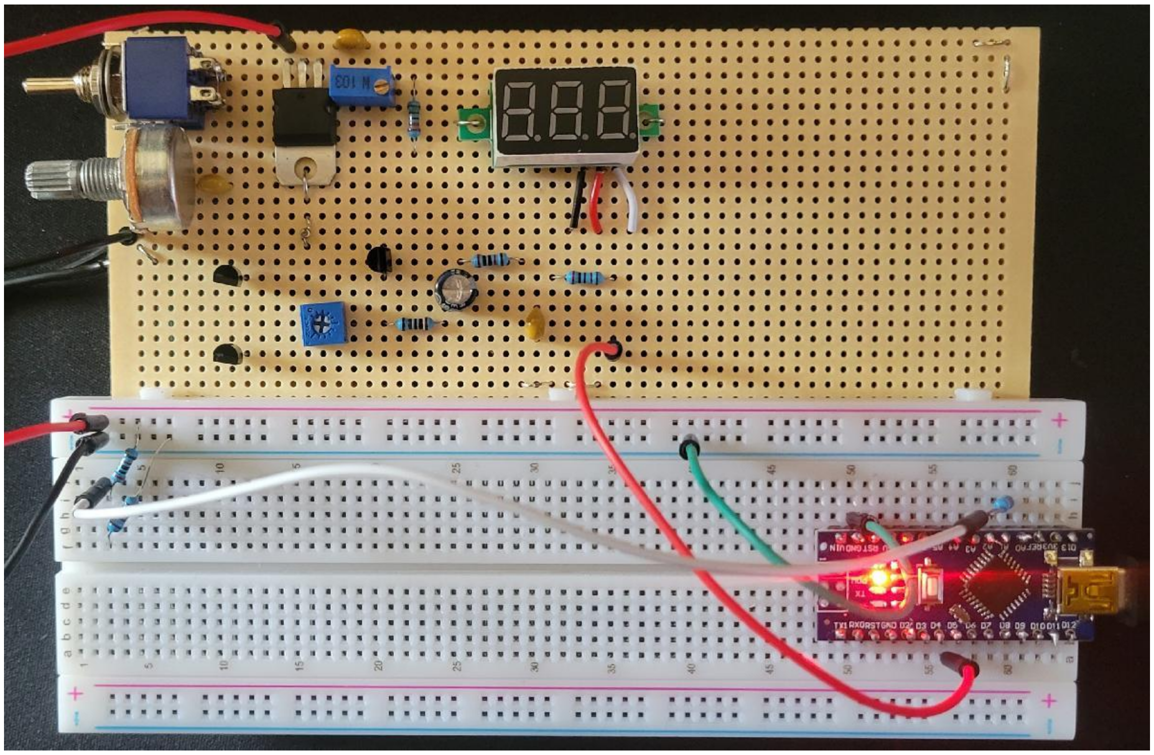


Figure 2 Prototype on solderless breadboard



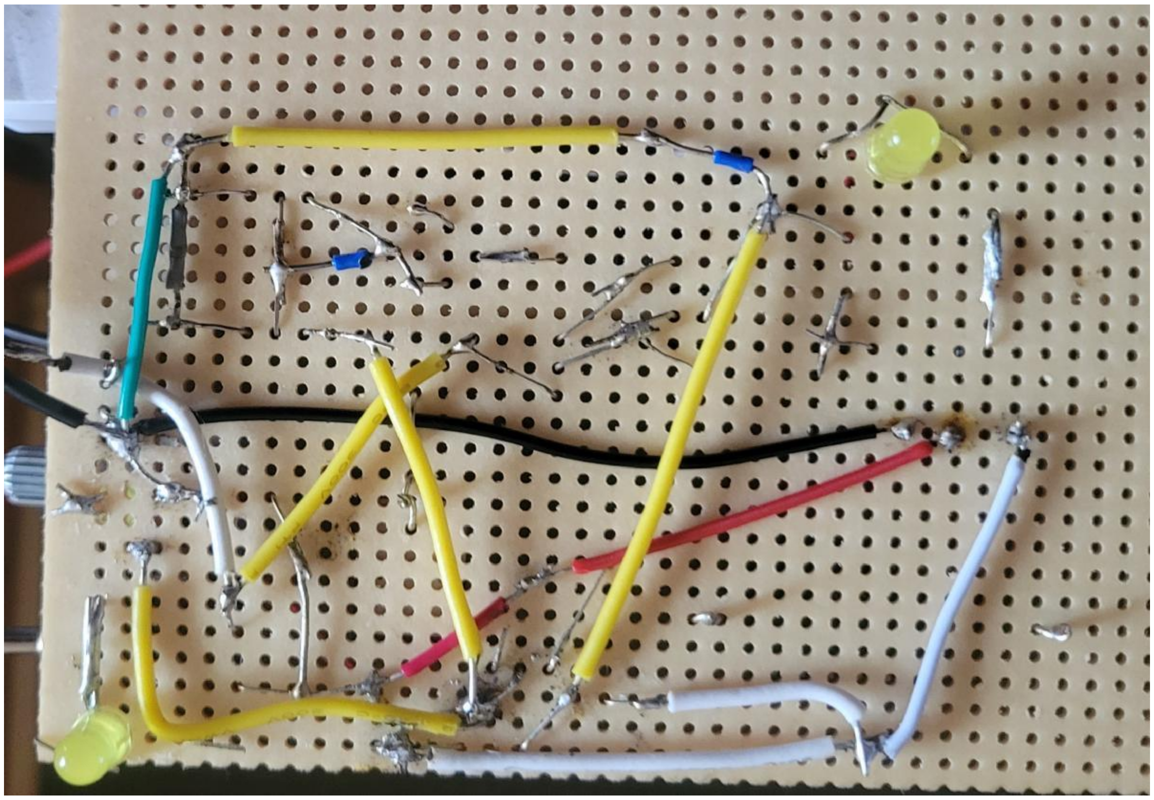


Figure 5 Topside and underside of the prototype put onto the PCB

Calibration Data:

```
vref = 4.62; % volts on vref pin
R7 = 9.92e3;
R6 = 20.08e3;
pwm = (50:5:255)'; % 42 different values
v317 = zeros(size(pwm));
oscope1; %initialize ard
writeline(ard,'p50');
pause(10);
tic;
for i=1:length(pwm)
    writeline(ard,['p ' num2str(pwm(i))]);
    pause(1); % extra time for the voltage to stabilize
    runOnce = true;
    oscope1; % --> new data = [A0 A1]
    v317(i) = vref * mean(data(:,1)) / 1023 * (R6+R7)/R7;
end
```

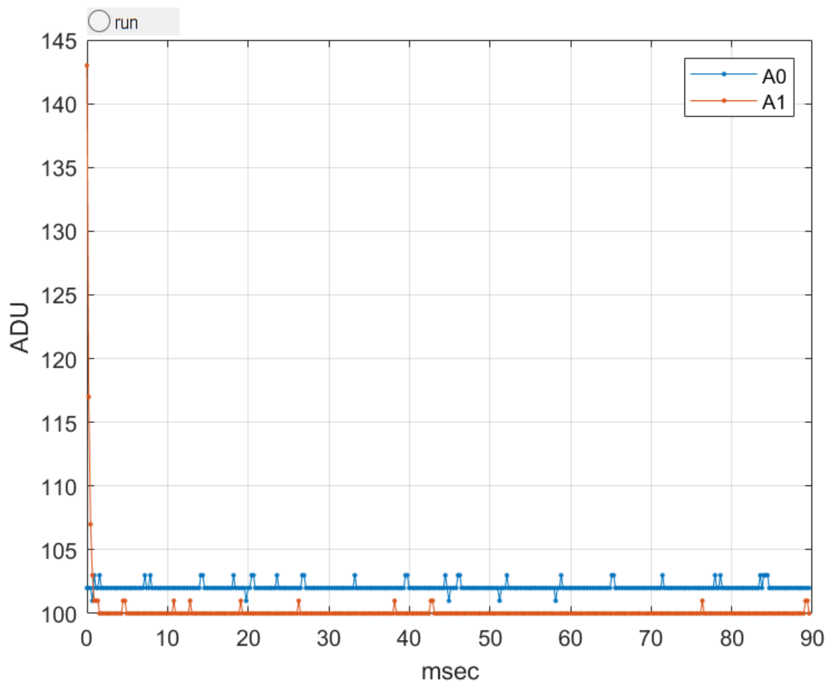


Figure 6 Calibration Data

```
toc
```

```
Elapsed time is 54.294351 seconds.
```

```
figure; plot(pwm, v317, '.-'); xlabel('pwm value'); ylabel('DCVout'); grid;
```

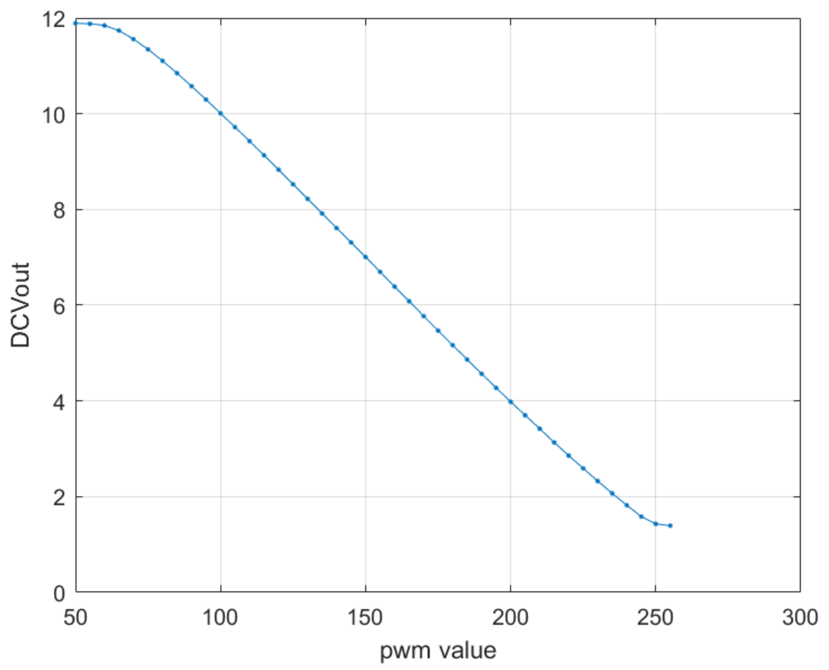


Figure 7 PWM plot

The PWM output on D5 drives a 2-stage low-pass filter (R3+C3 and R4+C4). The first stage's impedance set by R3 won't load D5 (5V/1k=5mA), and the larger 2nd stage's impedance, R4 won't load the first stage.

The emitter follower (T1) lowers R4's impedance by 1/hfe, driving current through R5 and R11 from Arduino's 5V supply, proportional to $(V_{T_1 b} - V_{T_1 be} - V_{T_2 b.})/(R_5 + R_{11}) \approx (V_{PWM} - 1.2) mA$.

```
%% model 1
R1 = 1300; % ohms measured between LM317 pins 1 and 2 (with power off)
R2 = 10.80e3; % max ohms measured between ground and *unconnected* terminal
R5 = 3.5e3  ; % R5+R11, calling it R5
T1baseVoltage = pwm * vref / 255; % the PWM filter's gain = 1 at DC
T1emitterVoltage = T1baseVoltage -0.6; % Vbe in saturation
T2collectorVoltage = 0.6;
iMirror = (T1emitterVoltage-T2collectorVoltage) / R5;
% if T1 isn't conducting then iMirror = 0:
iMirror(T1baseVoltage < 0.6 + T2collectorVoltage) = 0;
iR1 = 1.25/ R1; % A, from LM317 datasheet. 50uA from iADJ
iR2 = iR1 - iMirror ; % KCL says iR1 = iR2 + iMirror
VR2 = iR2 * R2;
DCVout = VR2 + 1.25;
```

```
% the output is always less than input voltage - dropout voltage
DCVout(DCVout < 1.25) = 1.25; % Vmin
plot(pwm, [v317 DCVout],'.-');
xlabel('PWM value'); ylabel('DCVout');
grid; axis tight;
legend('experiment','theory')
```

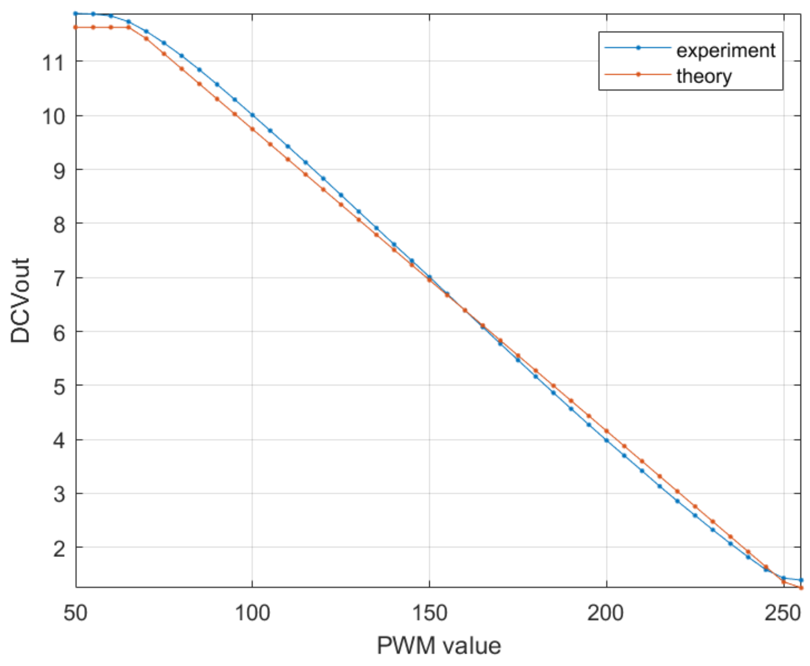


Figure 8 PWM experimental vs Theoretical 1

```
T1baseVoltage = pwm * AREF / 255  * (R5*hfe)/(R5*hfe+22e3);
```

```
%% model 2, with hfe
hfe = 240; % estimated
T1baseVoltage = pwm * vref / 255  * (R5*hfe)/(R5*hfe+22e3); % see below
T1emitterVoltage = T1baseVoltage -.6; % Vbe in saturation
iMirror = (T1emitterVoltage-T2collectorVoltage) / R5; % A
% if T1 isn't conducting then iMirror = 0:
iMirror(T1baseVoltage < 0.6 + T2collectorVoltage) = 0;
iR1 = 1.25 / R1;
iR2 = iR1 - iMirror; % KCL says iR1 = iR2 + iMirror
VR2 = iR2 * R2;
DCVout2 = VR2 + 1.25;

DCVout2(DCVout2 < 1.25) = 1.25; % Vmin
plot(pwm, [v317 DCVout DCVout2],'.-');
xlabel('PWM value'); ylabel('DCVout');
 grid; axis tight;
```

```
legend('experiment','theory1','theory2')
```

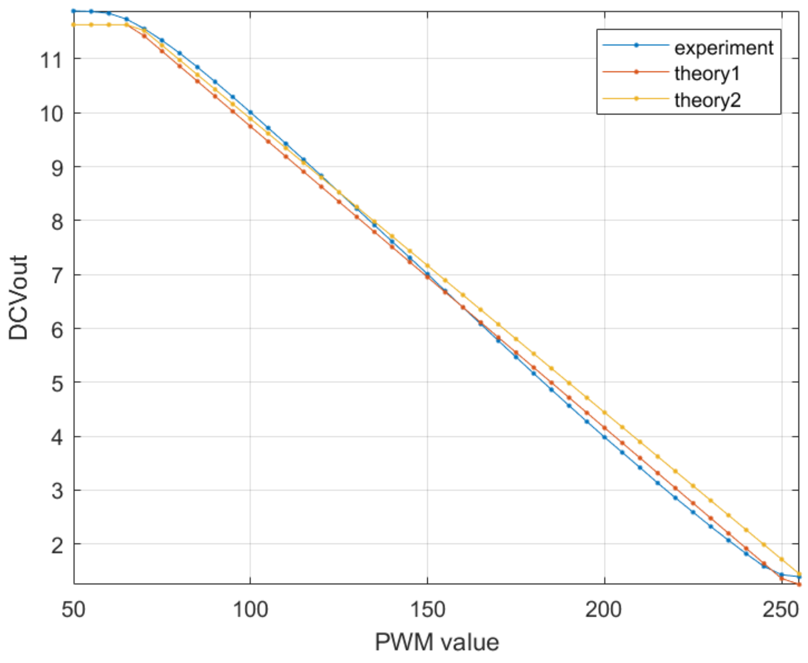


Figure 9 PWM vs theoretical 2

There needs to be one more change to this so now Vout = 1.25*(1+R1/R2)+iAdj*R2, where iAdj ~ 25-50 microamps:

```
%% model 3, with IAdj added
hfe = 240; % estimated
iAdj = 20e-6; % amps estimated
T1baseVoltage = pwm * vref / 255   * (R5*hfe)/(R5*hfe+22.2e3); % see below
T1emitterVoltage = T1baseVoltage -.6; % Vbe in saturation
iMirror = (T1emitterVoltage-T2collectorVoltage) / R5; % A
% if T1 isn't conducting then iMirror = 0:
iMirror(T1baseVoltage < 0.6 + T2collectorVoltage) = 0;
iR1 = 1.25 / R1;
iR2 = iR1 + iAdj - iMirror; % KCL
VR2 = iR2 * R2;
DCVout3 = VR2 + 1.25;

DCVout3(DCVout3 < 1.25) = 1.25; % Vmin
plot(pwm, [v317 DCVout DCVout2 DCVout3],'.-');
xlabel('PWM value'); ylabel('DCVout');
grid; axis tight;
legend('experiment','theory1','theory2','theory3')
```

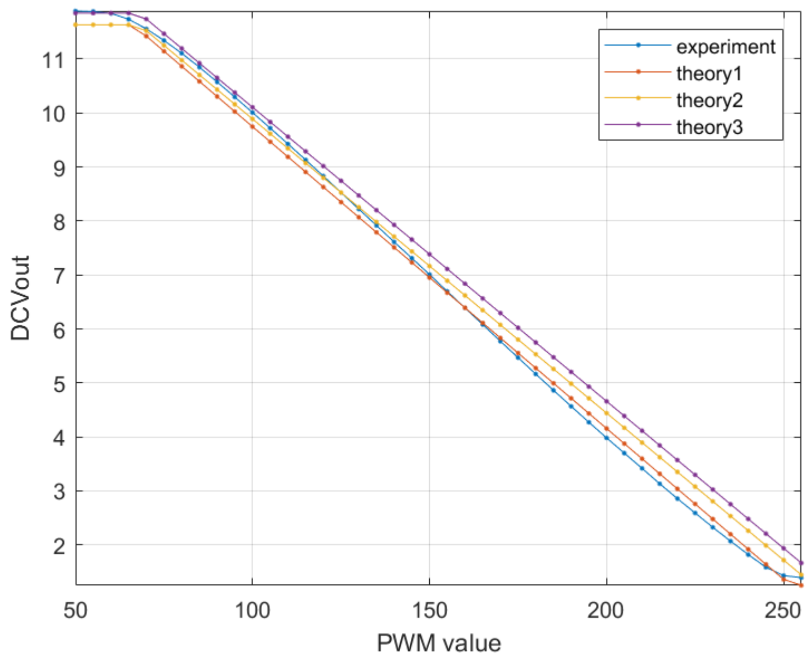


Figure 10 PWM vs Theoretical 3

Discussion

There appears to be something that is making the experimental values initially recorded to droop in the middle of the graph once it begins the recorded decent. At this time there is no defined culprit of that case.

This lab mainly showed me how difficult initially how automation can become but how helpful it can be in the long run. Recording 42 values of oscillating pwm values would be a pain if it was to be recorded on paper or copy and pasted. Having the Arduino talk to MATLAB in order to plot everything is exponentially more helpful.in addition to

Issues that were had was that initially mt t2 and t3 were connected incorrectly. I had them backwards so both of my collectors were connected to ground but luckily enough I had not turned on the power so I was able to re-orient them properly so nothing burned out or shorted. Implementing everything into the PCB was time consuming but creating the prototype was a good idea to allow me to realize the actual circuit that I needed to create

Function Generator

Objective:

Construct a function generator (sine/square wave voltage source) using the XR2206 IC kit given at the start of the semester.

Method:

The created function generator is able to produce 0-12V output, so I created a small voltage divider in the breadboard (Fig 1: column 3). From the small voltage divider, I used two jumper cables (white horizontal jumper cables) with 10k resistors soldered to the end to connect to A0 and A1 of the Arduino just incase there is an influx of power which could damage it. I also unplugged the LM317 red cable and stuck it on the end of the board in column 1 row A so there is no accidental touching which could lead to a short in the circuit or possible sparks.

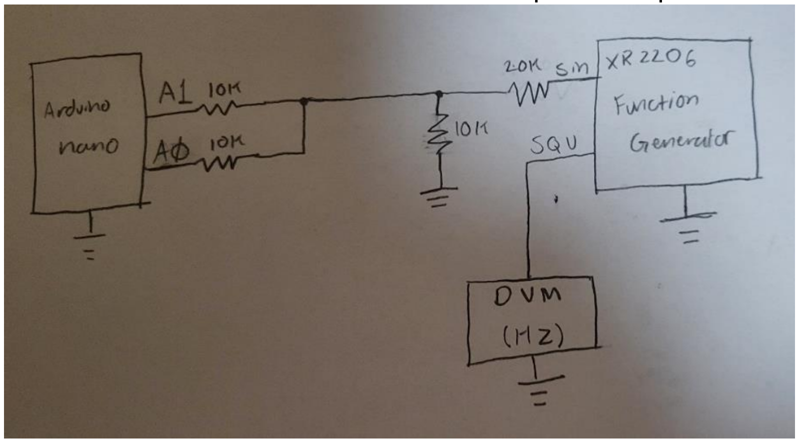


Figure 1 Schematic diagram

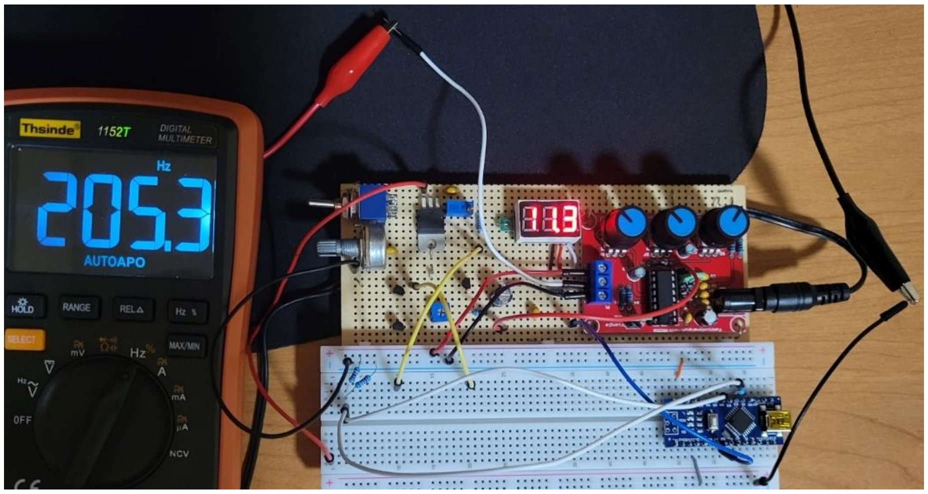


Figure 2 Function Generator on PCB

Figure 1. Black and red clip leads are attached the DVM to ground and Square (SQU) output from the function generator (white wire to red clip lead in the center). The sine wave's red wire, also seen at center) goes into the voltage divider that is connected to A0 and A1. Both of which have a 10k resistor (horizontal white wires on solderless breadboard). Now the reason for having both A0 and A1 measured is to calibrate a 2-channel oscilloscope.

Arduino software

The data we are recording is stored in the Arduino as binary digits. This code takes in the changing Voltages created by the function generator and tells pins A0 and A1 to read and save each input separately for 200 milliseconds. The stored data is binary data and will be read by MATLAB to be plotted for a visual aid see appendix [1] for code.

This is saved as *oscope1.ino [1]* and uploaded to the Arduino.

Connecting to the Arduino

This is *Oscope1,* see appendix [2], the first program that allowed MATLAB to connect and talk to the Arduino, and this is used to display the oscillation data that is stored on the Arduino.

```
initializing arduino ...
```

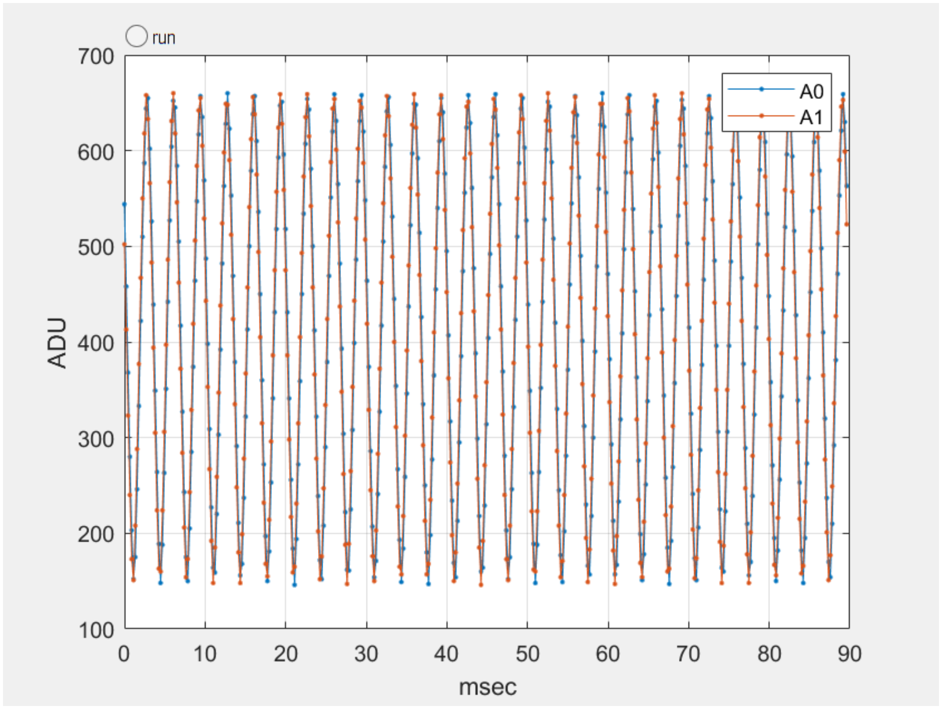


Figure 3 Oscope 1 plot

This is the basis that was worked with for the ending product of recording all of the necessary data, this was amended and updated as I progressed. The 'runBtn' if statement declares where the run button will appear on the figure window and gives a signal to MATLAB to start recording the input of both A0 and A1 to plot. The second if block tells MATLAB to look in the serial port list, and to grab the Arduinos (ard) data. All of the necessary data storage is started in the next while statement, which is to say that as long as the runBtn is pressed, record the data for both A0 and A1 with respect to time 't' in microseconds. That is the duty of the first 'try' function but with the second one is where it plots the data under the two-color codded lines with dots. After the final 'catch' the code labels the necessary axis respectively.

Triggering

Oscilloscopes have something implemented in them called "trigger" circuits so repetitive waveforms are repeated at the same phase or starting slope. The code below has taken all of that into account to allow for a 'trimmed' to start with the initial condition all of the time, and it also trims the period number as well. This is to make the most streamlined data collection and plotting:

```
%{
function [d, numperiods] = trim(data, thresh, npds, refchan)
% function [d, numperiods] = trim(data, thresh, npds, refchan);
% trim data to npds starting from the positive going thresh
% crossing (default: 0) on data(:,refchan).
% default refchan = data(:,end);
% npds < 1 --> return as many whole ones as possible (default)

% 10apr19 BR

if nargin < 4, refchan = length(data(1,:)); end
if nargin < 3, npds = -1; end
if nargin < 2, thresh = 0; end
a = data(:,refchan) > thresh; % boolean
b = [a(2:end); a(1)]; % shifted
p0x = find(b-a == 1); % positive zero crossing indices
p0x(end) = []; % the last one is length(data), not a p0x.

if npds < 1
    d = data(p0x(1):p0x(end)-1,:);
else
    d = data(p0x(end-npds):p0x(end)-1,:); % take the last ones ... why?
end
if nargout == 2
    numperiods = length(p0x)-1;
end
%}
```

This was saved in my MATLAB directory as 'trim.m' and is to be called in my new amended Oscope1 file which will be aptly renamed 'Oscope2

```
rawdata = reshape(bin(1:800),2,400)' - 1;
```

```
[data, npds] = trim(rawdata, 450); % adjust 2nd arg = ADU with steepest slope
t = linspace(0,dt/1000,400)';
t = t(1:length(data));
```

In addition to the above code, I have to amend the code to properly run so I have to rename the variable ‘runBtn.Callback’ as oscope2 instead of oscope1.

```
    runBtn.Callback = 'oscope2';
```

Nothing was touched on the Function generator so the data plots should be very comparable to each other

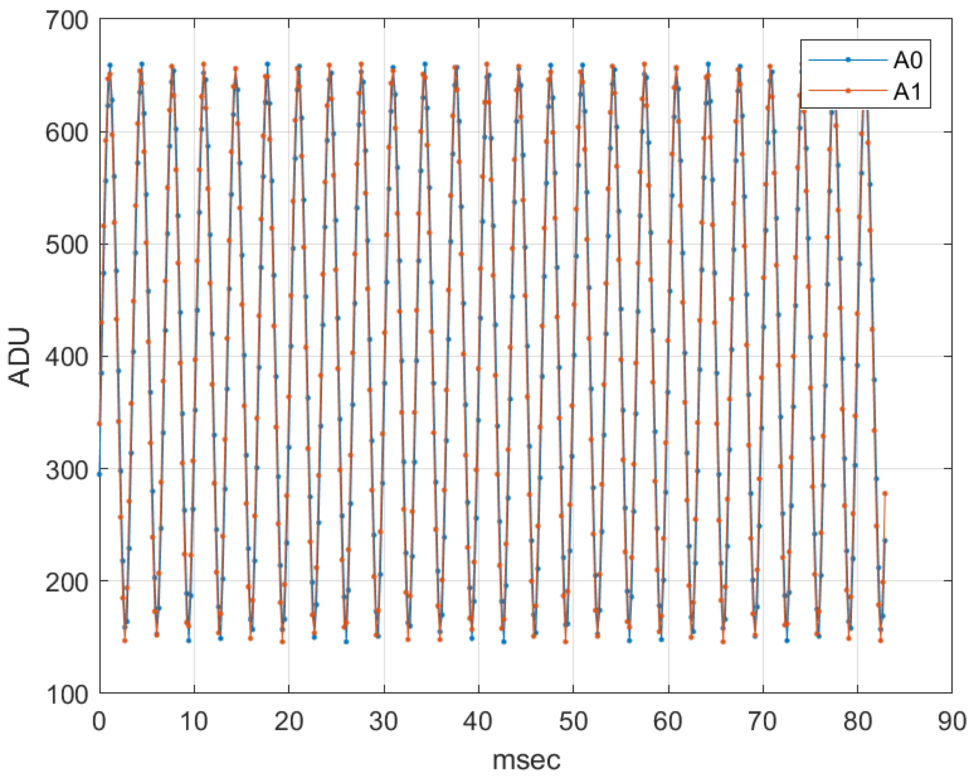


Figure 4 Oscope2 with trim.m

Notice how the data is less rambunctious and a tad more spaced out when compared to Figure 3. Thanks to the trimming of the period we get to see a much better plot of the new recorded data..

Measuring Frequency

Question: The square wave output of the function generator has an amplitude ~12V. A) Draw a schematic showing how could you would connect it to the Arduino. B) Which inputs would you have to connect it to in order to use interrupts?

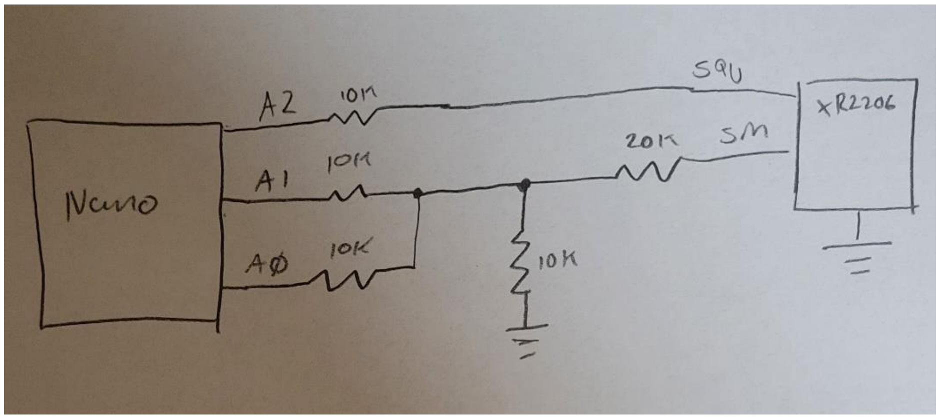


Figure 5 Schematic of Possible SQU recording

As shown from my schematic diagram I would simply attach the SQU to A2 on the Arduino. There have been previous labs where that has been the case. Some minor amendments to the oscope code could lead to some very interesting results.

```
sr = length(rawdata)/dt*1e6; % sample rate (#/sec)
freq = sr/(length(data)/npds); % Hz, 1/sec
sprintf('%.1fHz',freq)
```

```
ans = '301.6Hz'
```

The reason why the sample rate is given by that specific formula 'sr' is because it is taking the length of each position (datapoint) and then dividing it by the difference in time which is then divided by the length of the amended data to then be put into a frequency which is now displayed as 301.6 Hz. The conversion factors although confusing match up exactly how they are recorded which is extremely helpful for this process. When I connect my DVM to the SQU output of the function generator the output I get is vastly similar to the calculated frequency. The DVM read 300.9 Hz so I would have to say that I trust the DVM more than the Arduino because the DVM records more decimal places than the Arduino is allowed to calculate. When you give the DVM time to calibrate and level off it gives a very accurate reading rather than the simple calculation. Given that the calculated Hz is only 0.7 Hz off there is obviously room for debate, but the DVM is designed for this specific purpose, so it has to be more accurate?

Measuring Amplitude in the Time Domain

The code below is to show the amplitude of the wave forms given from the data, assuming that they are noise free:

```
%{
```

```
function y = mrms(x)
% returns the rms mean with the dc component removed of a
% vector or each col of a matrix.  Also see rms().

y = sqrt(mean(x .* x) - mean(x) .^ 2);
%}

vdvm = 3.031; % vac on dvm
vref = 5.57; % vdc on dvm
vard = mrms(data) * vref / 1023 * 3; % multiply by voltage divider
(20k+10k)/10k
[vard vdvm]
```

```
ans = 1×3

    2.7642    2.7640    3.0310
```

```
err = (vard - vdvm) / vdvm
```

```
err = 1×2

   -0.0880   -0.0881
```

Since I didn't have this code in my set path for MATLAB I created a new m file labeled MRMS so when MRMS is called in Oscope3 it will read this function and output the proper data The 8% difference between the Arduino and the DVM is less than expected which is good given that the mrms sums up the phase information and doesn't give a proper job at getting rid of the excess noise.

Measuring Amplitude and Phase with the Fourier Transform

A Fourier transform decompresses a given function into frequency components and with the code below this helps create complex amplitudes and phases versus the frequency that we use always starting at 0 Hz:

```
fd = fft(data); % data is trimmed to npds
ampl = abs(fd(npds+1,:)) / length(data) * 2
```

```
ampl = 1×2

  238.7913  238.7716
```

```
err = (ampl - mrms(data)*sqrt(2))./ampl
```

```
err = 1×2

   -0.0022   -0.0022
```

The FFT amplitudes are .2% lower but this leads to the possibility that they are more accurate because of the noise at all other frequencies which contribute to the ‘mrms’ values.

With the given below the phases were given in radians so there must be a conversion to degrees, so the code below properly does that for us without having to do any on paper math

```
theta = rad2deg(angle(fd(npds+1,:))), diff(theta)
```

```
theta = 1×2

 -118.5979 -106.4177
ans = 12.1802
```

The phase of each waveform is somewhat arbitrary -- related to the threshold we picked and the amplitude, but the difference in phase between the channels is what will have significance when we're not measuring the same waveform. Here we see a significant phase difference between A0 and A1 (which are the identical signal).

The phase of each recorded waveform is partially related to the threshold that was set when we turned on the function generator, but the main difference is between the channels that we're recording. The period over which the frequencies are recorded is drastically more important because it dictates the amplitude of V1 and V2. Any other specific degree would allow for a completely different plot and completely different data set.

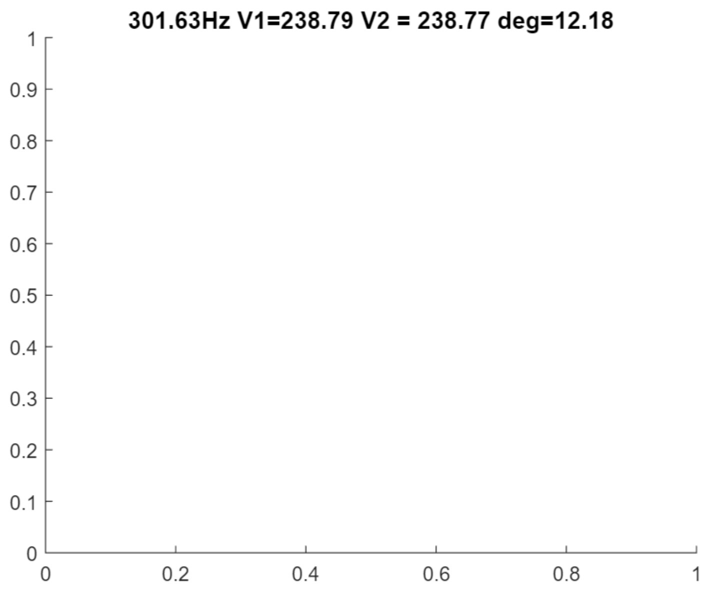
The code below is simply putting a title onto the graph going from the frequency (in Hz), the amplitude of V1 and V2, and the specific degree that was used for plotting the data:

```
sprintf('%.2fHz V1=%.2f V2 = %.2f deg=%.2f',freq , ampl, diff(theta))
```

```
ans = '301.63Hz V1=238.79 V2 = 238.77 deg=12.18'
```

Adding all this to the title requires shrinking the font:

```
plotHandle(3) = title(sprintf('%.2fHz V1=%.2f V2 = %.2f deg=%.2f', ...
             freq, ampl, diff(theta)),'FontSize',12);
```



This helps out immensely for data acquisition. If this code were to be called multiple different times with No title it would be near to impossible to figure out which type of data set we're looking at but given that with the Hertz being displayed as well as the degrees it allows for easy recognition of the data plot so there's less confusion.

Once again I put all these changes into ‘Oscope2’, and aptly renamed it to ‘Oscope3’ with the caveat of renaming the ‘runBtn’ to oscope3 so there is no error when reading the program.

De-skewing the plots

Given that the first measures A0 then A1 and then back to A0 This repeats until failure or until the specific parameters met in the code so this wouldn't make A1 a little skewed in the Arduino code when it is plotted via matlab. Now it's not entirely and I'll be all however depending on the frequency it could pose some very major difficulties so the code below would allow for that to not be as big of a problem:

```
rms([data data(:,2)-data(:,1)])
```

```
ans = 1×3

  443.0510  443.0539   36.3605
```

There are very minor differences in the given answers from A0 and A1, but I will still do an error analysis just for posterity

```
err = [ans(1)-ans(2) ans(3)] /ans(1)
```

```
err = 1×2

   -0.0000    0.0821
```

Less than 0.0001% difference even though it's not showing it in the calculated answer between rms amplitudes of A0 and A1, but the difference in the waveforms is a minor 8.2% of the signal, now thankfully it is not as big as it could be but this still shows that there is a significant difference in both waveforms.

To do this I will need to record every A0 on an odd time and every A1 on an even time so that when they are plotted together they seem to almost perfectly match as one sine wave rather than two sine waves placed on top of each other:

```
tt = (1:numel(data))';
deskewedA1 = spline(tt(2:2:end),data(:,2),tt(1:2:end));
plot(tt(1:2:end), data, 'o-', tt(1:2:end),deskewedA1, '.')
legend('A0','A1','deskewedA1')
```

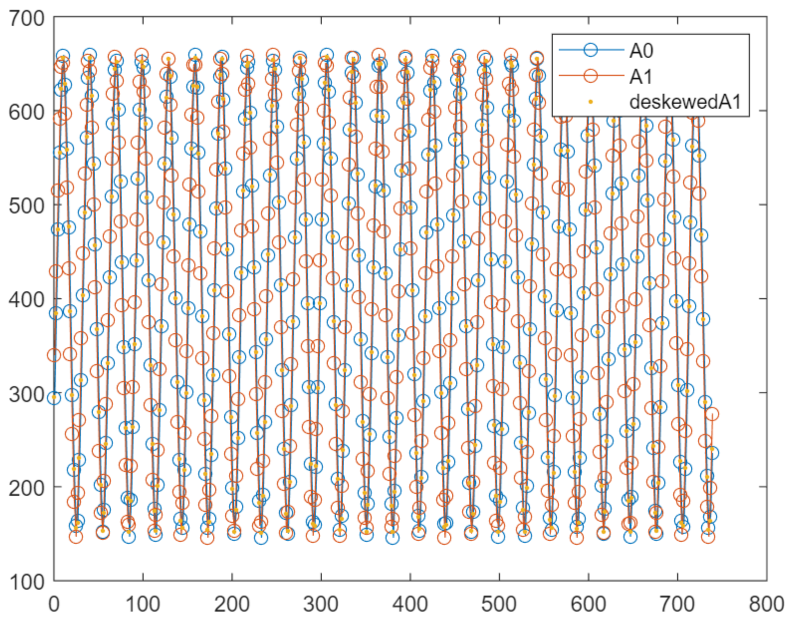


Figure 6 De-skewing A1 (orange) to match with A0 (blue)

Now the plots are almost perfectly on top of each other even though we had barely an 8% skew with A1. granted seeing all of the circles on the plot make it much harder to actually focus on everything but The only way to make it more widespread is if I lowered the frequency which will be done later.

```
data(:,2) = deskewedA1;
rms([data data(:,2)-data(:,1)])
```

```
ans = 1×3

   443.0510   443.0261     1.3791
```

```
err = [ans(1)-ans(2) ans(3)] /ans(1)
```

```
err = 1×2
```

```
    0.0001    0.0031
```

once again after the de-skewing I propose to do the error analysis once more and it gives out a 0.3% which means that it's almost perfectly on top of each other with gnome deviances it could be possible to get it down to zero however I will not go any further.

This de-skewing code will be added to the final m file 'Oscope3' to give the most accurate data plots that can possibly be had at this given time.

Results

Oscope3 is the final oscilloscope .m file that is required to give all of the data we need to graph everything. See appendix [3]

As one can see evidently see, everything is included from oscope 2's trim function and obviously oscope 1. They're given plot at the end of this will be the most accurate plot of the data set recorded:

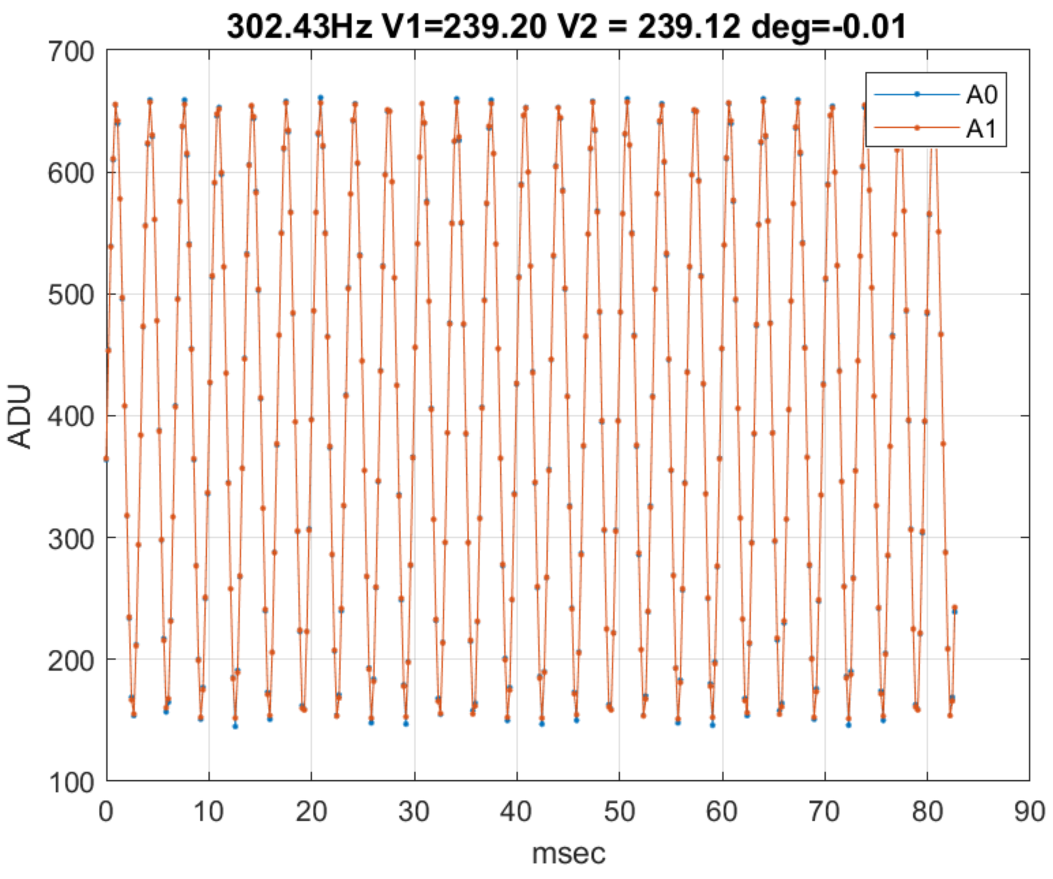


Figure 7 Oscope 3

Now it is almost impossible to see a zero however if you look at the peaks and valleys of the waveforms you can see tiny blue dots. This means that the graphs, when laid on top of each other, are almost exact with the almost the exact same recorded data points as well.

Turning the frequency and amplitude pots results in obvious expected changes in the data and values in the displayed title. Now, for posterity, I will look at the most extremes of the frequencies and waveforms. See that my recordings capped out at around 650 HZ I will see how much of a difference I can create when turning all of the potentiometer knobs.

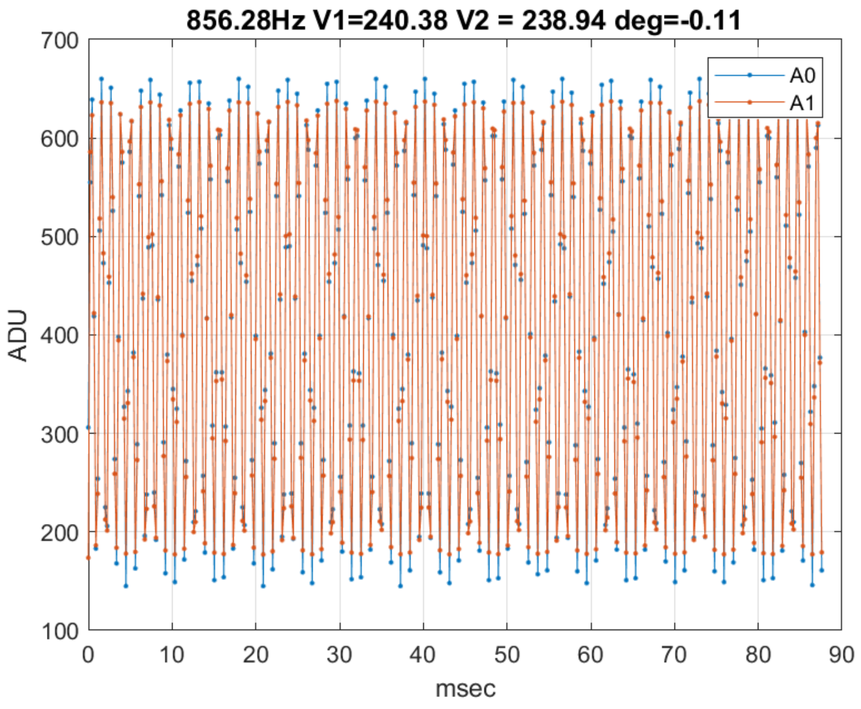


Figure 8 Maximum Hz

with this high end there's a distinct difference between A0 and A1, a zero seems to creep into both lower and higher frequency ranges which allows for minor deviation between the plotted dots. A1 seemingly has stayed in the same area that it has been before.

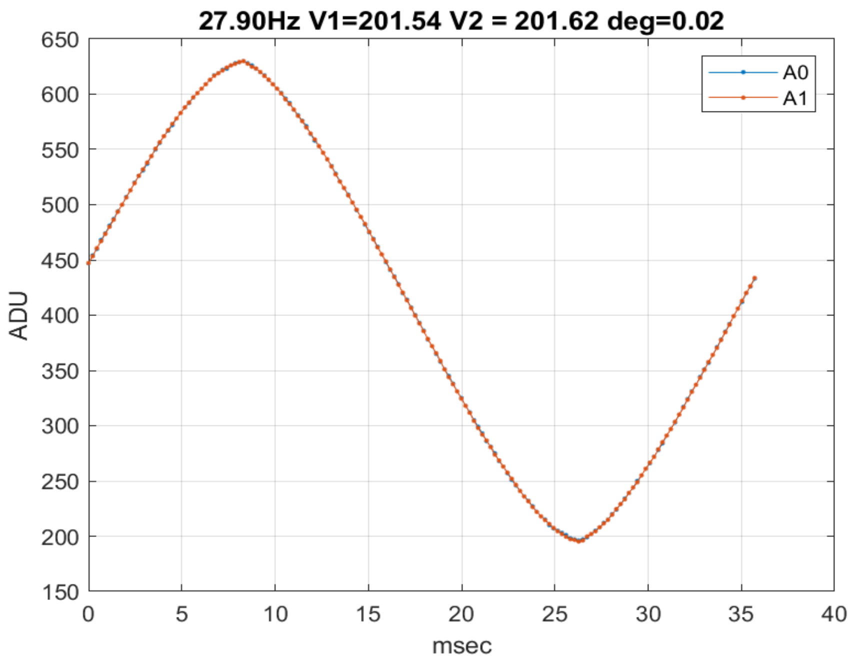


Figure 9 Minimum Hz

Now with the wind it is very hard to see the difference between a zero and a one given that they are almost perfectly set on top of each other. When looking at the peaks and valleys specifically you can see minor differences of coloration between blue and orange but overall, the graph seemingly matches up perfectly.

Now when I was trying to go as low as possible, I could seemingly not get past 27 Hz, sometimes my plot even disappeared and went into single digit ADU. So I went as low as I possibly could without that happening to me, As seen below, the lowest I could go before the plot broke was 27.72 Hz:

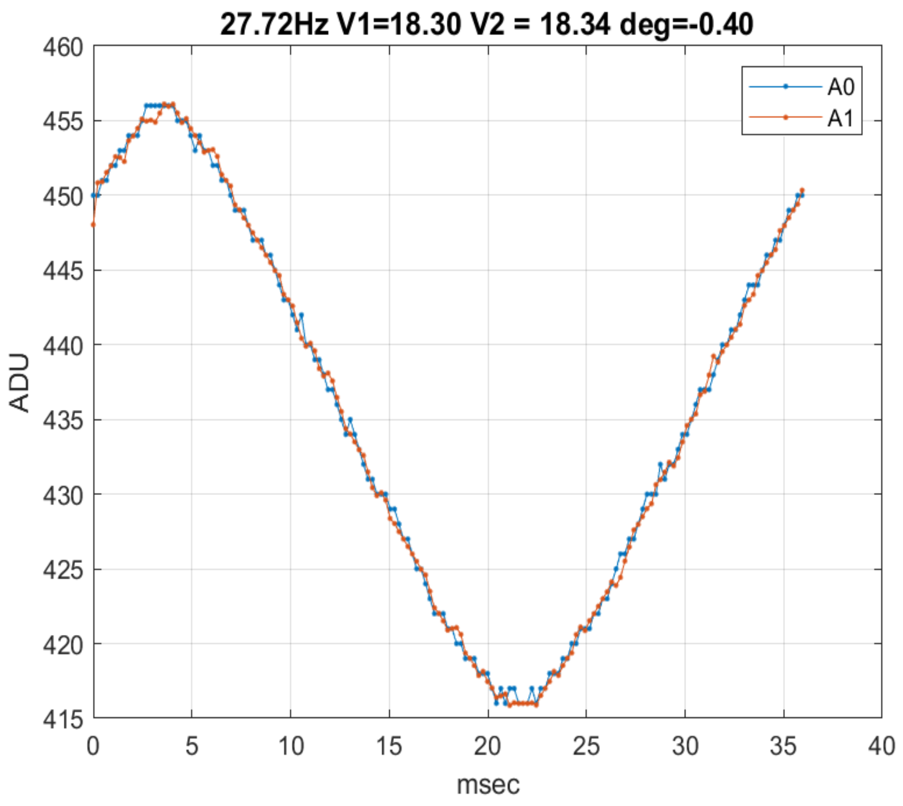


Figure 10 Very minimum Hz

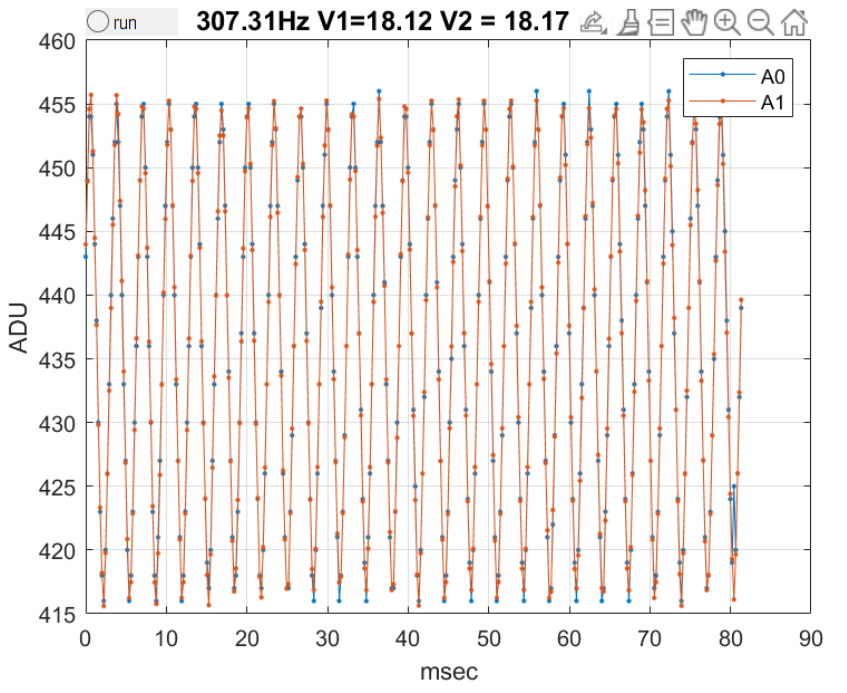
Now this plot in particular is very interesting because A0 and A1 are very sporadic and not as streamlined as they were before. As seen in Figure 9 there looks to be a smooth curve for each oscillation but with this plot it seems to be that it is not all smooth and perfect as thought before. For instance, with a zero there are some slight jumps from decreasing values as seen roughly around 441 ADU and 10.1 msec there's actually an increase of the plotted data instead of a decrease. For A1 there are actually some points on the plot where it begins to have the same data repeated so there are some flat areas of the curve.

Discussion

Throughout this lab, the range of frequencies going from 27 to 800 Hz and the amplitudes of each channel matched up almost perfectly with the meter I had connected.

The main goal of plotting voltage versus time from Arduino and MATLAB into the two-channel oscilloscope worked as intended. Now obviously our oscilloscope only works in the thousands per second but there are definitely other oscilloscopes that work in the millions per second, but this is still much more efficient than anyone human or group of people could do given the almost instantaneous amount of time it takes for it to be plotted and graphed and to be properly updated in real time. obviously with the code provided there can be improvements but this still is infinitesimally better for automated data acquisition and recording. The fact that there was almost 0% error on the mrms calculations shows that this is a much better way to acquire data. If it were to be recorded by a person there would definitely be confusions and mix ups in data that was plotted leading to much more drastic differences in error percentage.

```
tic;
for k=1:100
    runOnce = true;
    oscope3;
    freqs(k) = sr/(length(data)/npds);
    ampls(k,:) = rms(data);
end
```



```
toc
```

```
Elapsed time is 22.301831 seconds.
```

```
fprintf('CV of A0=%.4f, A1=%.4f frequency = %.4f\n',  ...
    std(ampls) ./ mean(ampls), std(freqs)/mean(freqs))
```

```
CV of A0=0.0001, A1=0.0001 frequency = 0.0119
```

```
fprintf('interchannel difference = %g\n',diff(mean(ampls))/mean(mean(ampls)))
```

```
  interchannel difference = -6.23955e-06
```

Over 22 seconds the amplitude stability is better than 0.01 percent (1 parts per hundred thousand), that is starkly perfect for what has happened thus far.

There were some very notable lessons learned in this lab and they go as follows:

1. The transformation of binary data being collected on an Arduino converted into MATLAB data to be plotted on a graph.

2. Being able to mess around with the function generator and seeing different distortions being shown on the MATLAB graphs in real time to see how turning each knob affected the amplitude and period of the plot.
3. Having to create two new .m files(trim and mrms) in my MATLAB directory allowed me to become invested more in how MATLAB works and even seeing the directory work its way through everything really allowed more insight into how everything talks to each other.
4. Seeing the major difference in this skewing that did happen before and after trim and even looking at the high and low ends of the frequency plots gave more insight into how a function generator can work and the differences in every frequency and how meticulous everything can be.

Conclusions

Some difficulties I had with this lab were the plots not actually showing up when I clicked the run button in MATLAB. Another problem that arose was that I had to clear my workspace multiple times in order to get the oscope programs to properly work. For example, I kept forgetting to >> clear runBtn on multiple occasions and it just led for data overlap so I had to restart multiple times when I forgot to do so. Another difficulty I had was attaching the function generator to the PCB board, this is more of a personal non data related difficulty given that my breadboard is double the size of others, so my PCB had less real estate to be used. This required me to restructure my PCB and my previous solder work on other circuits on the PCB however, this did allow me to become more streamlined in my setup for everything so even though this was a setback it was actually a step forward in a different, much better direction. Some people shooting advice that I would give to my future self for starters is to definitely clear 'runBtn' as many times as possible after data has been implemented into MATLAB. the reason why the amplitude is so high is because the amplitude potentiometer Is hardwired in reverse thanks to the function generator creators so I would have to spin it the opposite direction to make the amplitude smaller.

Appendix

[1] Arduino oscope.ino code

```
%{
/*
   b -- measure a buffer of A0 and A1 appended with read time in microseconds,
return in binary
   p <pwmValue> write pwmValue to D5
  16oct22 BR
  31oct22 BR: adding 1 to all data values, so we can use a 0 as terminator.
Otherwise,
  data that equals default terminator (LF=10) stops the read.
*/
void measureWaveformsBinary(void);

const int analogOutPin = 5;
const int settleTime = 200; // msec
#define NUMSAMP 400
int data[NUMSAMP + 2][2], i;
unsigned long t0, t1;

void setup()
{
  Serial.begin(115200);
  pinMode(analogOutPin, OUTPUT);
}   // setup

void loop()
{
  if (Serial.available())
  {
    switch (Serial.read())
    {
      case 'p':   // set pwm value
        {
          int pwmVal = Serial.parseInt();
          analogWrite(analogOutPin, constrain(pwmVal, 0, 255));
          delay(settleTime);
          break;
        }
      case 'b':   // readAndWriteWaveformsBinary
        {
          measureWaveformsBinary();
          break;
        }
    }   // switch
  }
  delay(1);
} // loop

void measureWaveformsBinary()
{
  t0 = micros();    // time it to compute sample rate
  for (i = 0; i < NUMSAMP; i++)
  {
    data[i][0] = analogRead(A0)+1;
    data[i][1] = analogRead(A1)+1;
  }
  t1 = micros();
  data[i][0] = t1 - t0;  // put dt at end of data
  data[i][1] = (t1 - t0) >> 16; // most significant 2 bytes
  data[++i][0] = 0; // terminator
  data[i][1] = 0;   // terminator
  Serial.write((uint8_t*)data, (NUMSAMP + 2) * 2 * sizeof(int));
}   // measureWaveformsBinary
%}
```

[2] Oscope1.m

```
%% script oscope1.m -- first version of an Arduino oscilloscope display
% 16oct22 BR, Arduino running oscope1, command set = b, p, s
if ~exist('runBtn','var') % add the button once
    runBtn = uicontrol('style','radiobutton','string','run','units', ...
        'normalized','position',[.13 .93 .1 .04]);
    runBtn.Callback = 'oscope1';
end
if ~exist('ard','var') % initalize arduino
    disp('initializing arduino ...')
    ports = serialportlist;
    ard = serialport(ports{end},115200);
    ard.Timeout = 2;
    configureTerminator(ard, 0);
    clear ports;
    pause(2); % time to boot
end
if ~exist('runOnce','var'), runOnce = true; end

while runBtn.Value || runOnce
    writeline(ard,'b');
    try
        bin = read(ard,804,'int16');
        dt = bitshift(bin(802),16)+bin(801); % microseconds
        data = reshape(bin(1:800),2,400)' - 1;
        t = linspace(0,dt/1000,400)'; % calibrate the time axis
    catch
        flush(ard);
        break;
    end
    try   % change data inside the plot is faster
        set(plotHandle(1),'XData',t,'YData',data(:,1));
        set(plotHandle(2),'XData',t,'YData',data(:,2));
    catch
        plotHandle = plot(t, data, '.-');
        xlabel('msec'); ylabel('ADU')
        legend('A0','A1'); grid on;
    end
    drawnow;
    runOnce = false;
end
```

[3] Oscope2.m

```
%% oscope3.m -- added title with freq, ampls, phase diff, and spline deskew
%% oscope2.m -- added trim for triggering
%% script oscope1.m -- first version of an Arduino oscilloscope display
% 16oct22 BR, Arduino running oscope1, command set = b, p
if ~exist('runBtn','var') || ~isvalid(runBtn)  % add the button once
    runBtn = uicontrol('style','radiobutton','string','run','units', ...
        'normalized','position',[.13 .93 .1 .04]);
    runBtn.Callback = 'oscope3';
end
```

```
if ~exist('ard','var') % initalize arduino
    disp('initializing arduino ...')
    ports = serialportlist;
    ard = serialport(ports{end},115200);
    ard.Timeout = 2;
    configureTerminator(ard, 0);
    clear ports;
    pause(2); % time to boot
end
if ~exist('runOnce','var'), runOnce = true; end
```

```
 while runBtn.Value || runOnce
     writeline(ard,'b');
     try
         bin = read(ard,804,'int16');
         dt = bitshift(bin(802),16)+bin(801); % microseconds
         rawdata = reshape(bin(1:800),2,400)' - 1;
         [data, npds] = trim(rawdata, 450); % adjust 2nd arg = ADU with steepest
slope
         t = linspace(0,dt/1000,400)';
         t = t(1:length(data));
         sr = length(rawdata)/dt*1e6; % sample rate (#/sec)
         freq = sr/(length(data)/npds); % Hz, 1/sec
         tt = (1:numel(data))'; % deskew
         data(:,2) = spline(tt(2:2:end),data(:,2),tt(1:2:end));
         fd = fft(data); % data is trimmed to npds
         ampl = abs(fd(npds+1,:)) / length(data) * 2;
         theta = rad2deg(angle(fd(npds+1,:)));
     catch
         flush(ard);
         break;
     end
     try   % change data inside the plot is faster
         set(plotHandle(1),'XData',t,'YData',data(:,1));
         set(plotHandle(2),'XData',t,'YData',data(:,2));
         plotHandle(3) = title(sprintf('%.2fHz V1=%.2f V2 = %.2f deg=%.2f', ...
             freq, ampl, diff(theta)),'FontSize',12);
     catch
         plotHandle = plot(t, data, '.-');
         xlabel('msec'); ylabel('ADU')
         legend('A0','A1'); grid on;
         plotHandle(3) = title(sprintf('%.2fHz V1=%.2f V2 = %.2f deg=%.2f', ...
             freq, ampl, diff(theta)),'FontSize',12);
     end

     drawnow;
     runOnce = false;
 end
```