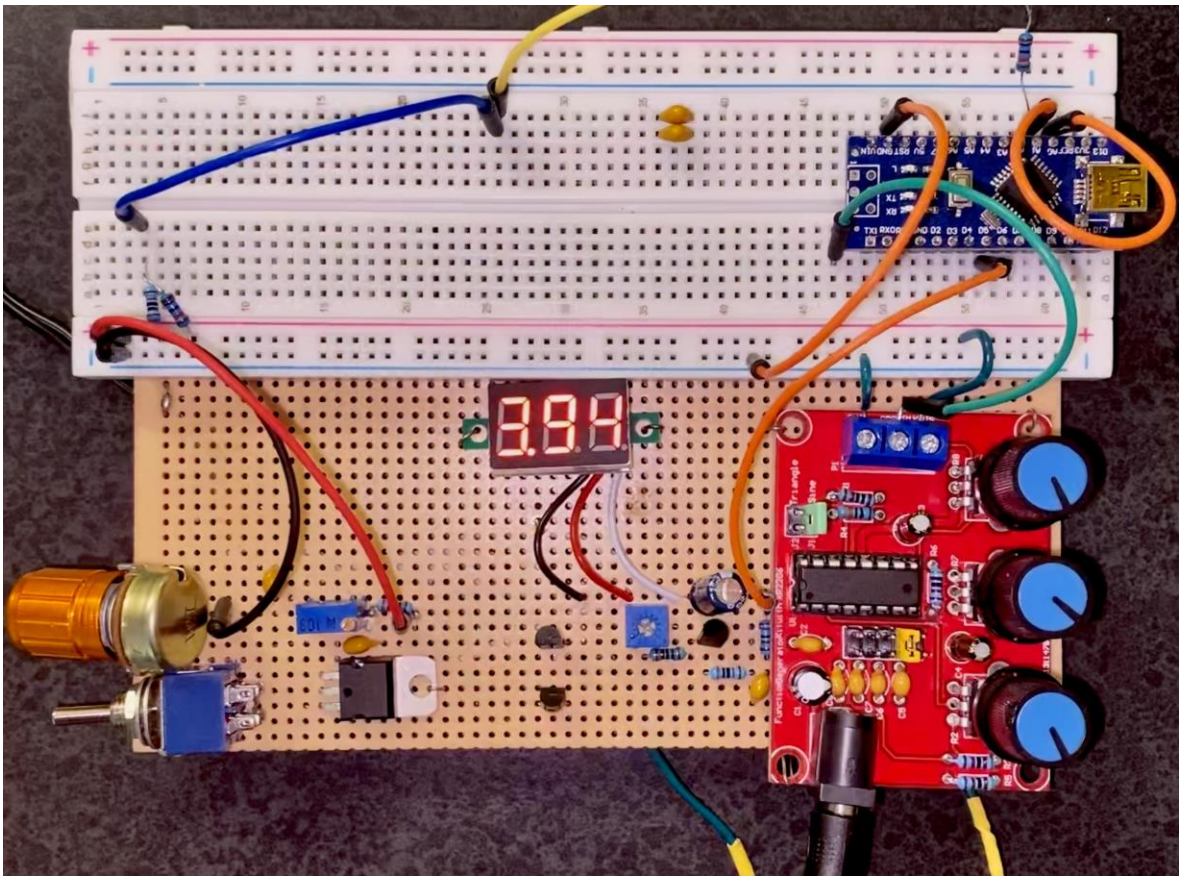
California State University, Channel Islands (CSUCI)
Department of Physics

EMEC/PHYSE 310 Electronics
Final Portfolio



Semester: Fall 2022

Student Name: Evan Frackelton
Student ID: 003173202

Table of Contents

LED Night Light

1. **Objective**
   Making an LED nightlight with an AC power adapter or Wall Wart.

2. **Methods**

   Schematic

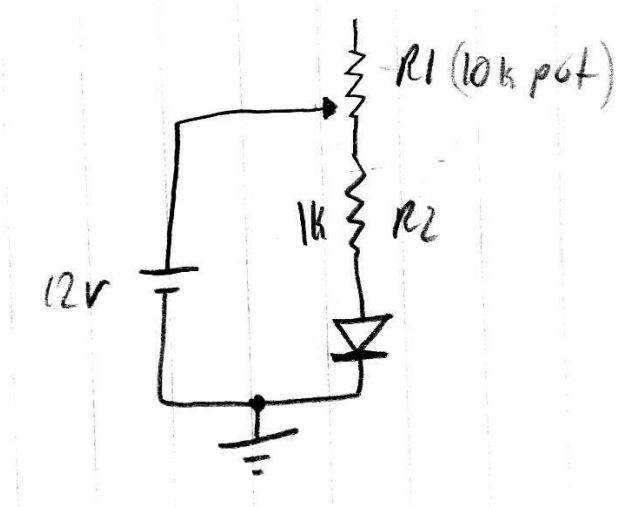


Figure 1: Schematic of circuit

The 12V wall wart positive connects to the first pin on the 10k pot (R1). The middle pin of the pot goes to a 1kΩ resistor (R2) which is soldered and connected in series with an LED. LEDs are diodes so on the schematic it is labeled as one. The LED's cathode is connected to ground.

LED IV curve and characteristics

By turning the 10k pot, the resistance increases and therefore drops the voltage and current. Using a multimeter, the voltage and current can be found at each interval. The higher the current, the brighter the LED. If the current drops more the LED will become dimmer. The *Vf*, or forward voltage, is the voltage going across the LED in relation to its source voltage and resistor. To calculate *I*, we use Kirchhoff's Law.

$$V = IR$$

```
% led i-v curve
% 31 aug 21 BR

i = [12.74 9.86 7 4.86 2.65 1.53 1.2]'; % mA
vdiode = [1.98 1.96 1.94 1.91 1.89 1.88]'; % volts
plot(vdiode, i,'o-');
xlabel('Voltage/volts');
ylabel('Current/mA');
title('Green LED');
grid;
```

Figure 2: MATLAB code to calculate IV curve

A MATLAB file is used to calculate the IV curve of our LED. Input the current values into variable 'i' and the voltage valued into variable 'vdiode'. It will plot the voltages on the x-axis and currents on the y-axis.

Wall Wart Characteristics

V Thevenin of the wall wart is gotten by measuring the load of the wall wart. Next, we need the R Thevenin. We can do so by plugging the wall wart directly into a 100Ω resistor. The resistor will get hot, so holding the resistor with pliers will help. Check the multimeter before and after putting the resistor in.

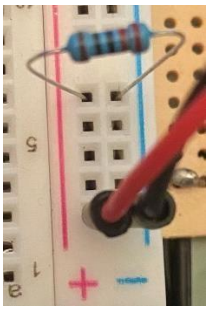


Figure 3: Resistor plugged into wall wart

We use a formula to find R Thevenin.

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

Circuit design and construction

To design the circuit to have the desired i we connect the wall wart to the power rails on the breadboard by stripping wall warts positive and negative and using jumper cables to connect them to breadboard wires.

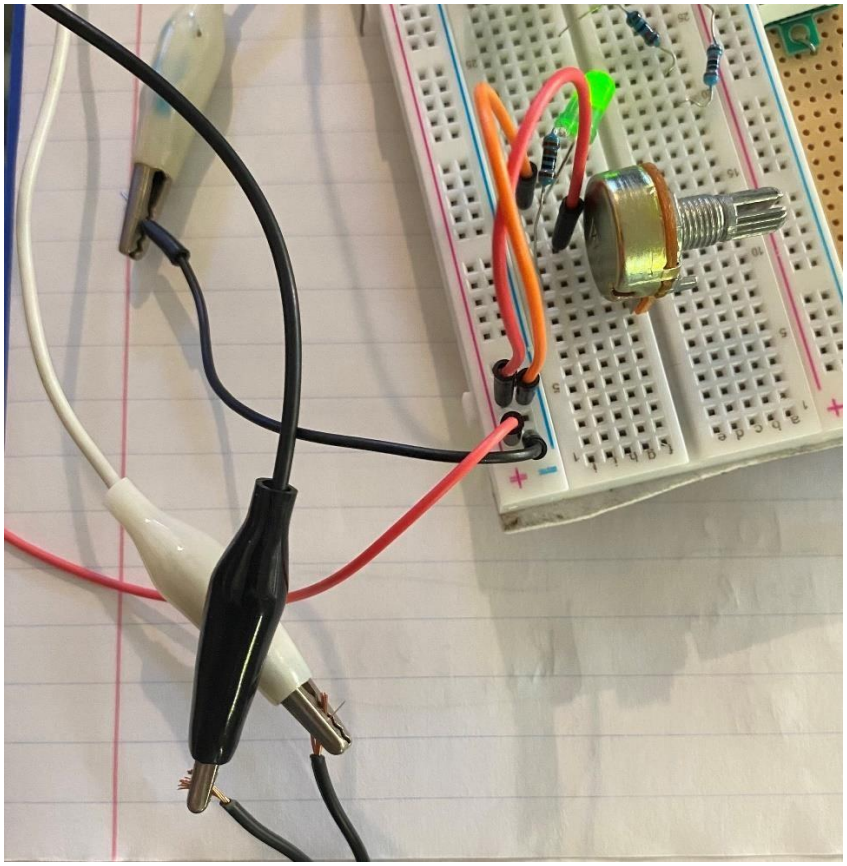


Figure 4: Wall Wart connected to breadboard wires

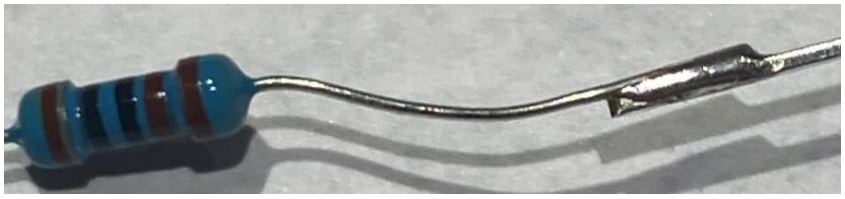


Figure 5: Resistor soldered to LED

The breadboard can easily hold the resistor and LED for soldering.

3. Results

LED IV curves and *Vf*

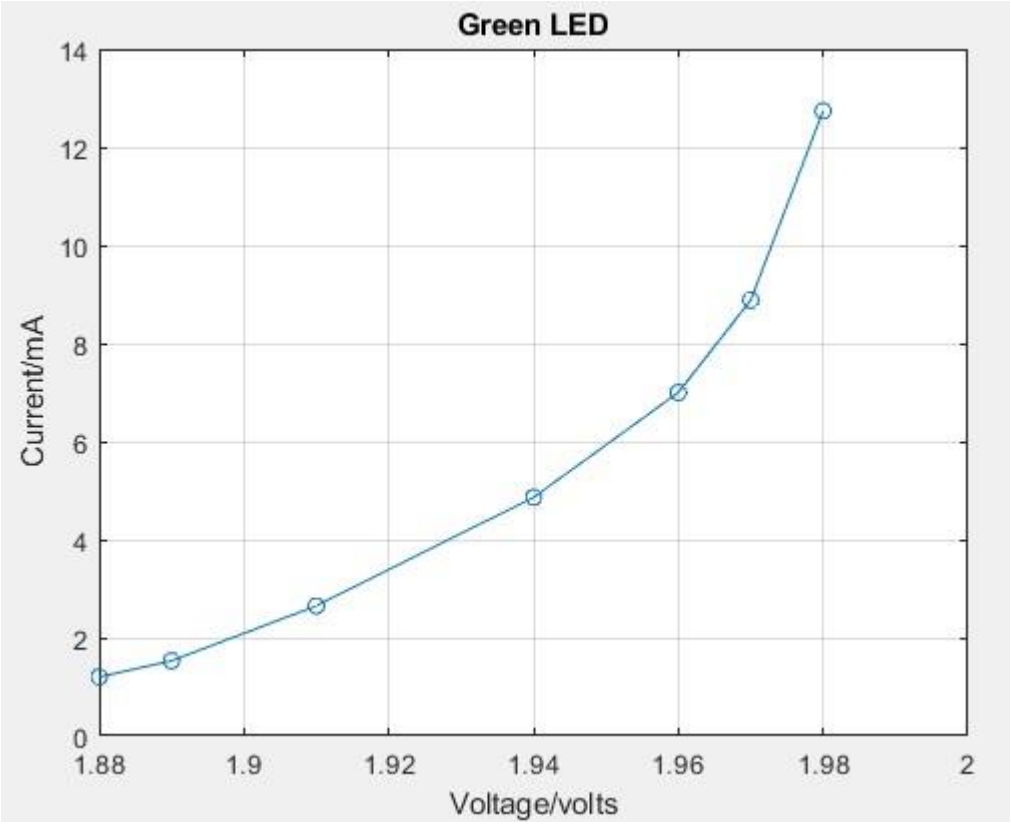


Figure 5: Green LED/diode IV curve

As said in methods, the LED's IV curve is gotten with a MATLAB file (diodeiv.m). The graph shows voltage versus the current, which is the definition of an IV curve.

Using the multimeter, the forward voltage is measured as 1.96V.
(*Vf* = 1.96V)

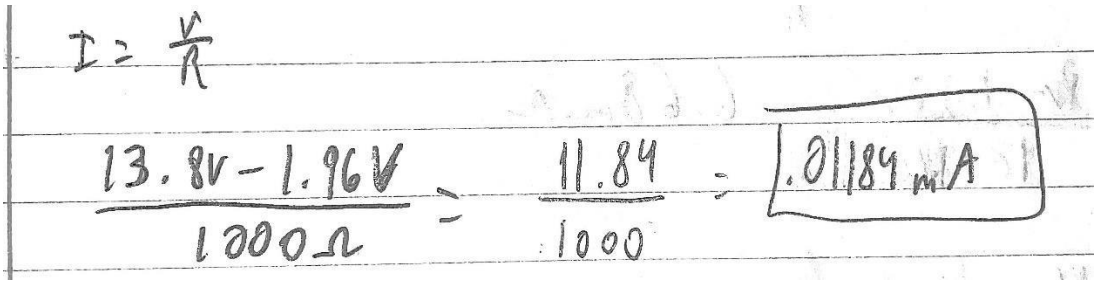


Figure 6: Calculation of current for green LED

Figure 6 shows that the current of the LED with respect to the forward voltage is .01184mA.

Wall Wart Thevenin

After using the method described in "Wall Wart Characteristics" the voltage dropped from 13.8V to 12.75V. V Thevenin is 13.8V. V_L = 12.75 which is used in the formula to calculate R Thevenin shown in Figure 7 below.

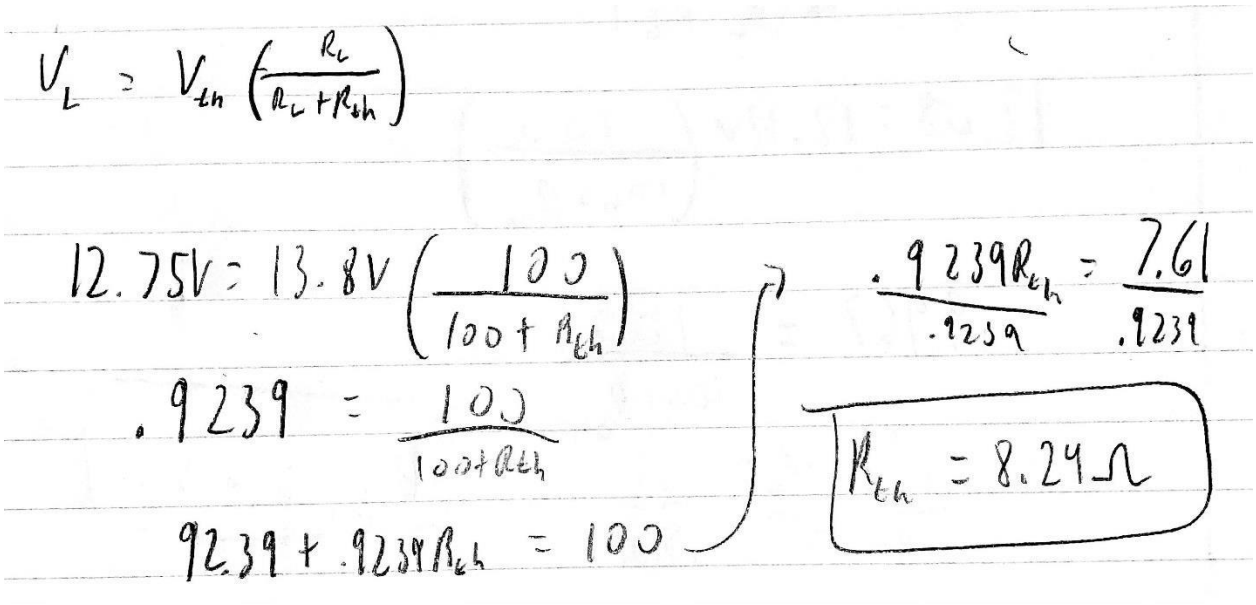


Figure 7: Calculation for wall wart R Thevenin

R Thevenin = 8.24Ω

Schematic

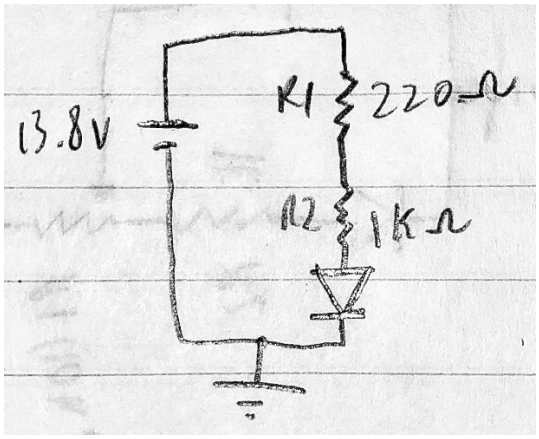


Figure 8: LED nightlight final schematic

A 13.8V power supply goes into a 220Ω resistor in series with a 1kΩ resistor. That is then connected in series to an LED. The cathode of the LED is connected to ground.

Final Circuit

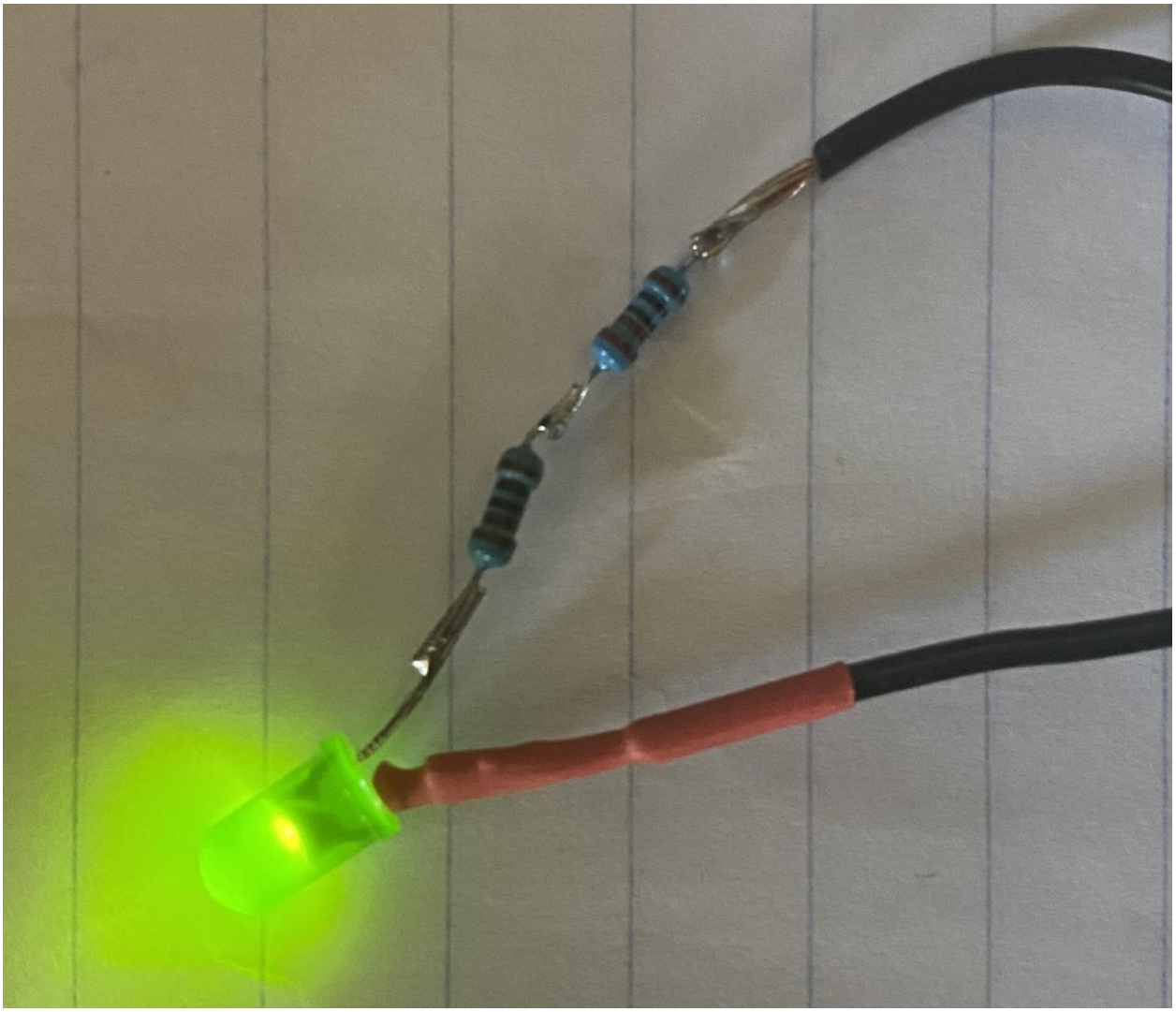


Figure 9: Final circuit of LED nightlight

A 220Ω resistor replaced the 10k pot because the brightness of the LED was close to the best at that resistance. When the pot was fully turned with no resistance, the 1k resistor (R2) got a little hot, but at 220Ω everything was a normal temperature.

4. Conclusions and Discussions

The circuit functioned properly with no issues. The wall wart that I used was a 12V, 1A wall adapter but its output voltage when measured was 13.8V. All data worked around the 13.8V power supply. According to IV curve theory, the LED's current should exponentially increase while the voltage remains close to the same. In the experiment, the curve accurately represents the exponential current increase, and the voltage had a difference of 0.1V.

Power usage

To calculate the amount of power the nightlight will use, we can use the power formula

$$P = VI$$

13.8 x 1.62mA = 22.36mW

We will assume that the month is 30 days long on average. To get power usage per month, 22.36mW is multiplied by the number of seconds in 30 days, which is 2592000. That comes out to 57957.12 mWpm (mWpm per month).

In California, the current electricity cost is 19.90 cents per kWh. Convert that to cents per mWh to get 1.99×10^5. Finally do cents per mWh times mWpm.

$(1.99 \times 10^5 \text{mWh}) * (57957.12\text{mWpm}) = 1.15$ cents

It costs 1.15 cents to run the LED night light each month. That cost can be marginally reduced by using different LEDs that use less voltage and current for similar brightness.

Variable DC Power Supply

1. **Objective**

   Build a DC Variable Power Supply

2. **Methods**

   Schematic

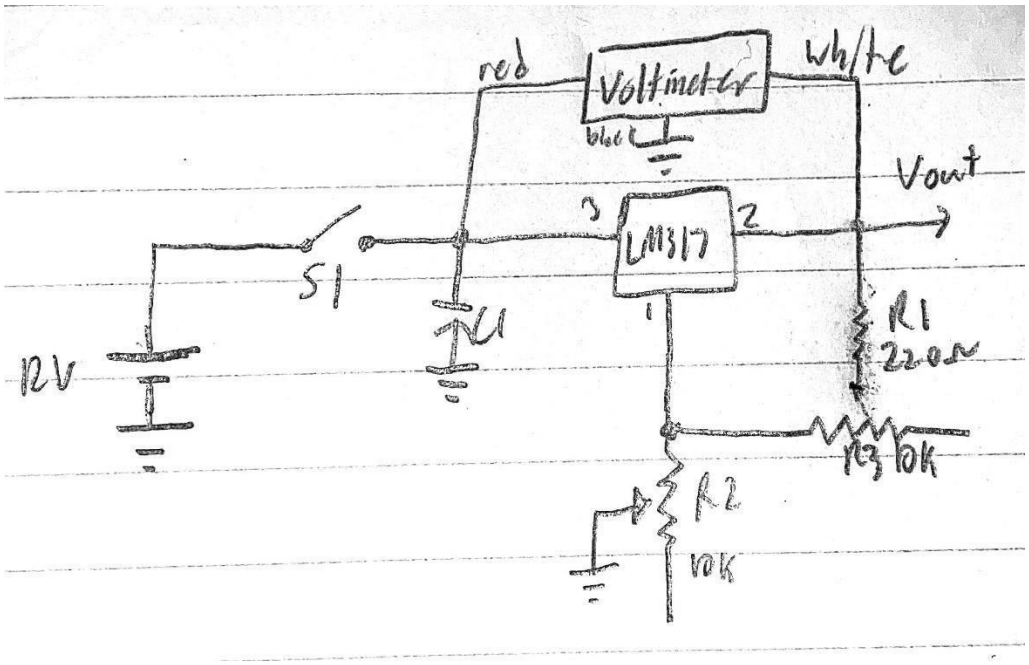


Figure 1: DC power supply schematic

The 12V wall adapter positive is connected to the to a switch to turn the power supply on and off. The output of the switch goes to the third pin on the LM317 and the red wire on the 3-digit voltmeter. Also, a 4.7µF capacitor is connected to ground on the same wire to lessen the noise transmitted which could affect the performance of the rest of the circuit. The 10k potentiometer is connected to the first pin on the LM317 which will be used to tune the voltage. The second pin of the LM317 is connected to the white wire of the voltmeter and a 220Ω resistor wired in series with another 10k potentiometer. However, the pot labeled R1 is a trim pot or a variable resistor rather than a normal potentiometer. The output of that variable resistor connects to the input of the potentiometer. Vout is connected to pin two and is the final positive output of the DC power supply.

Construction

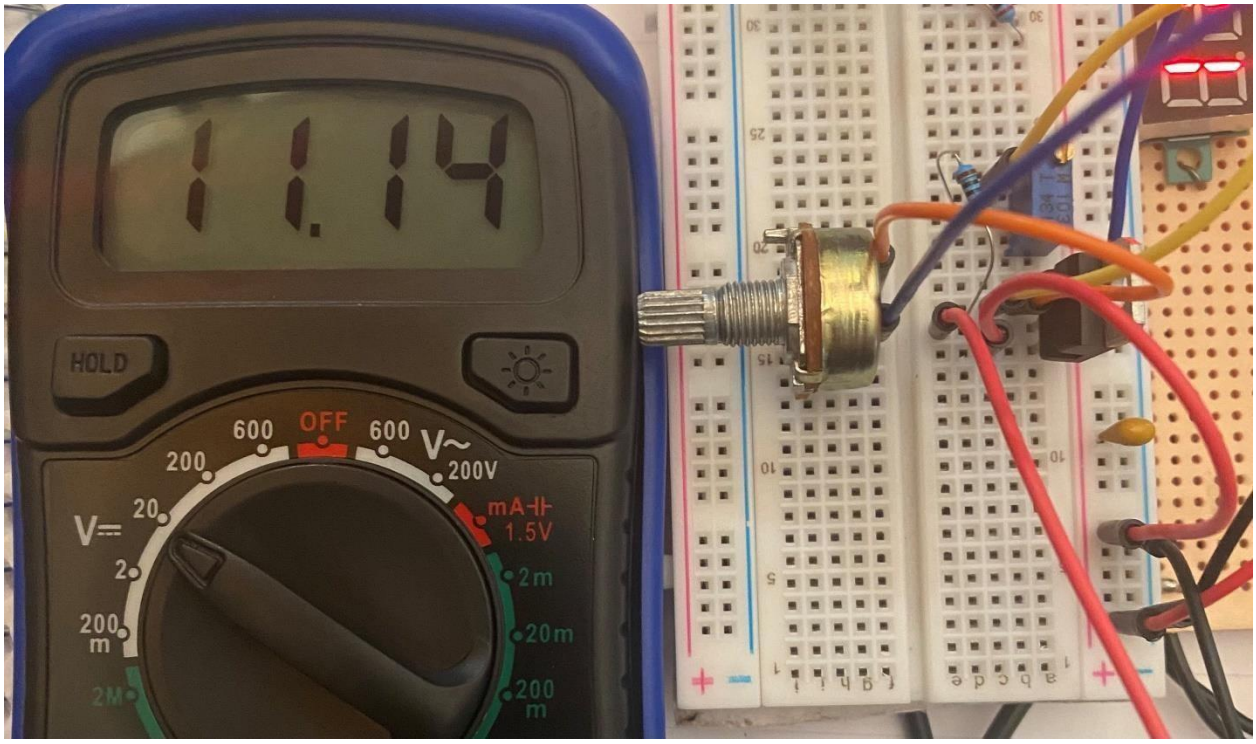


Figure 2: DC power supply prototype on breadboard

Following the schematic described in Figure 1, a functioning DC power supply is built. The switch and voltmeter are left out because it isn't necessary to test if the circuit it functional. Instead of the voltmeter we can connect a multimeter in the same way to see the voltage of our supply.

Testing

As the 10k potentiometer is turned, the voltage changes with it. This means that the LM317 is correctly regulating the voltage of our input. Turning the pot down will make our voltage lower and vice versa.

3. Results

Final Power Supply

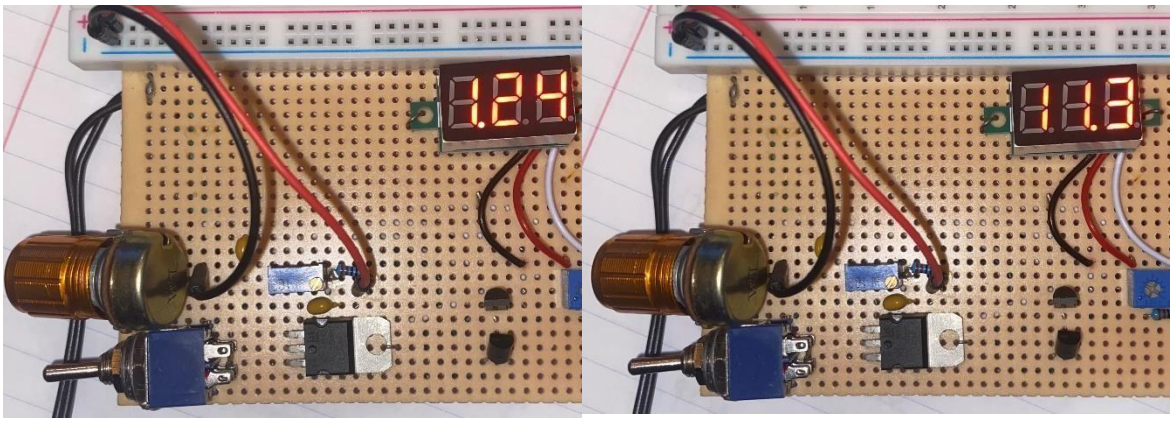


Figure 3: Functioning soldered DC power supply

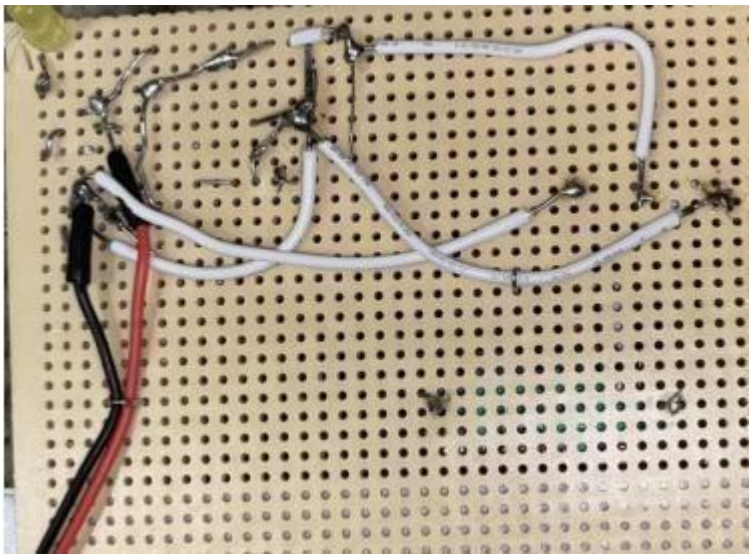


Figure 4: Bottom side of DC power supply

As shown in Figure 3, the power supply's minimum voltage is 1.24V and its maximum is 11.3V.

Power Supply Functionality

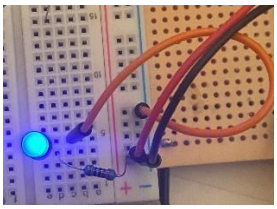


Figure 5: LED powered with power supply

The power supply can be used with the breadboard to do additional testing with circuit. In Figure 5 a 1k resistor in series with an LED is being powered and changes brightness when turning up the voltage on the power supply.

Power Supply Thevenin

To get the Thevenin resistance of the DC power supply a 100Ω resistor is connected directly to the positive and ground. A multimeter is used to read the voltage drop after the power supply is turned on. The resistor will become very hot and can be severely damaged so the power supply should be turned on for a few seconds to read the multimeter. V_L is equal to the voltage with the 100Ω resistor. We use a formula to then find Thevenin resistance.

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

$$10.84\,V = 11.3\,V\left(\frac{100}{100 + R_{th}}\right)$$

$$.959 = \frac{100}{100 + R_{th}}$$

$$95.9 + .959R_{th} = 100$$

$$R_{th} = 4.28\,\Omega$$

Figure 6: Calculation for R Thevenin

R Thevenin = 4.28Ω

4. Conclusions and Discussions

This lab was helpful in improving my soldering, wiring, and general electronic abilities. Understanding what each component of a circuit is for makes it much easier to build. There were a lot of tight spaces for soldering but the circuit functions properly which proves that tight, neat soldering is clean and efficient. I was surprised about using a 4.7μF capacitor connected directly to 12V. My knowledge of electronics when starting this class was minimal and because of that I thought the capacitor would not be able to withstand the voltage and current of the supply. For future troubleshooting, it is a good idea to use a multimeter to check circuit continuity. This is especially important if something is accidentally grounded. The wires are very close together so something shorting isn't out of the ordinary, especially when my soldering abilities are intermediate. In figure 3, the DC power supply includes other components from future labs which are unrelated but do not affect data recorded in this lab.

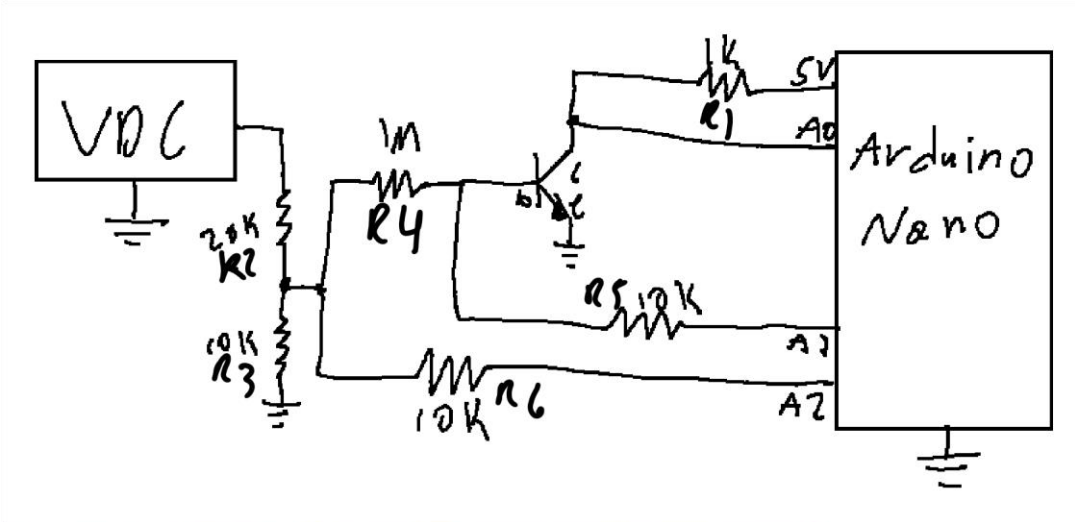
Hfe measurement

Objective

Measuring the HFE of an NPN transistor.

Methods

Hardware Interface



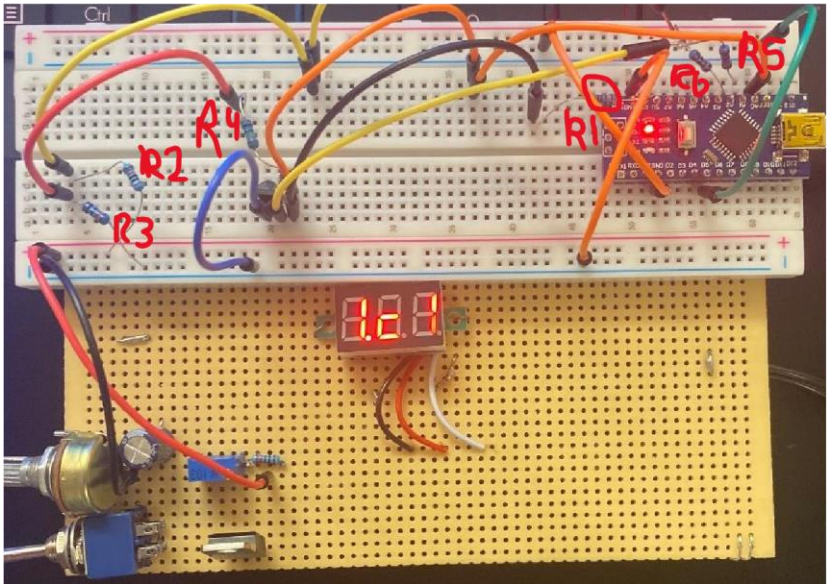


Figure 1. Schematic and breadboard of the automated Hfe circuit.

For this experiment we used an S8050 transistor. Note the value of $R4 = 1M\Omega$ here, vs. for a switch. Larger R4 allows exploring the transistor's behavior before it saturates at

$R4 = 10k\Omega$

higher i_BReducing R1 < ~500$^{\Omega}$ can severally damage the Arduino beyond repair as it provides the current through R1 of ~5V/R1 when the transistor is saturated.

Software Interface

The same code in the previous lab is used with an added line for A2. Pin D5 is grounded to toggle data logging. When grounded, data was printed to the Serial Monitor at ~5 times per second.

```
7  void setup()
8  {
9    Serial.begin(9600);
10   pinMode(5, INPUT_PULLUP);
11 }
12
13 void loop()
14 {
15   if (!digitalRead(5)) // ground D5 to run
16   {
17     Serial.print(analogRead(A0));
18     Serial.print('\t');
19     Serial.print(analogRead(A1));
20     Serial.print('\t');
21     Serial.println(analogRead(A2));
22   }
23   delay(200); // milliseconds
24 }   // loop
```

Data Interface

Data collected from the Arduinos code was copied/pasted below into variable data and converted into 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; % base resistor, R4, ohms Rc = 1011; % collector resistor,
R1, ohms data = [... A0, A1, A2
1023 119 89
1023 119 90
1023 119 90
1023 119 92
1023 119 92
1023 120 94
1023 120 98
1023 126 98
1023 127 101
1019 122 131
1014 124 161
1008 125 185
1004 126 203
```

```
998 128 228
993 129 250
987 129 272
982 130 293
974 130 326
966 130 358
957 131 393
949 131 425
941 132 458
929 132 502
919 133 540
906 133 591
893 133 642
882 134 682
868 134 739
854 134 791
853 135 792
853 135 794


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

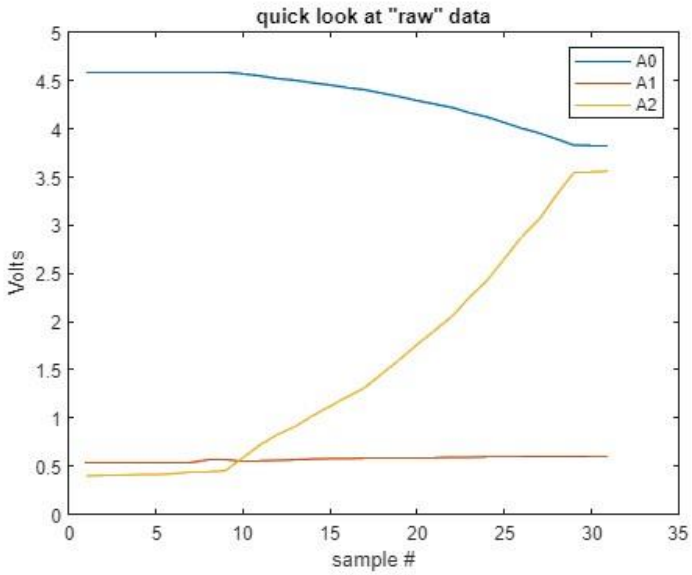


Figure 2. Raw data received from the Arduino.

The sample numbers we receive from the Arduino is completely arbitrary data. A1 is equal to Vbe which barely moves on the graph because it acts like a forward biased diode in conducting mode. The base current must be changing, but the base voltage hardly does, because the base-emitter diode has a nearly vertical IV curve. As A2 increases that means means the base current is

increasing, so the collector current should increase which causes more V = icRc voltage drop across Rc, causing Vce to DROP: KCL says Vref = Vce + icRc. So if the 2nd term increases, the first one must proportionally decrease, and the red curve above does that. To measure characteristics of the transistor, we need to compute the key voltages and currents using the code below.

```
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')
```

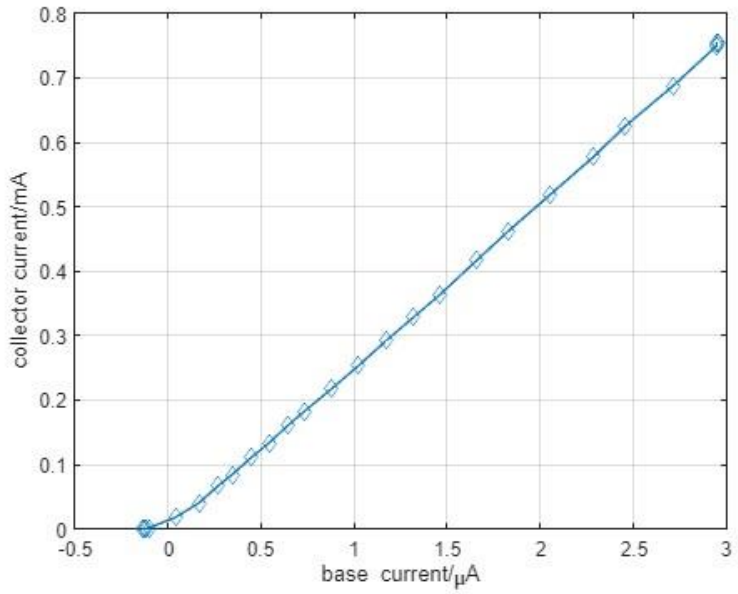


Figure 3. The relation between base and collector currents (with Rc = 1k) is very linear above the lowest currents (bottom left).

This graph allows us to get to a slope. We do a visual estimate to have an expectation for a more accurate least squares fit, or polyfit as described in MATLAB: $\beta = ic/ib$ ~ .8mA/3uA ~ 267. For polyfit, we should exclude the lowest few points, which might non-linear because the transistor hasn't quite turned on yet, Vbe is below the knee. For example we use points where ib > .1uA. We also are plotting mA/uA, so 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 =231.2

We repeat with a different collector resistor. To explore higher collector currents, smaller resistor. If more than 10mA is put into the Arduino for to long, then it may damage it so this is only done for a short period of time.

```
ib1k = ib; ic1k = ic; beta(1) =
p(1)*1e3;  Rc = 681; % ohms data = [...
1023 117 88
1023 123 90
1023 123 90
1023 117 100
1019 121 148
1015 123 186
1011 125 217
1007 126 246
1003 127 276 999
128 303
994 129 338
990 129 371
985 130 404
980 131 440
974 131 477
967 132 533
958 133 588
950 133 641
947 133 664
942 134 699
938 133 727
933 134 759
929 134 794
] * 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')
```

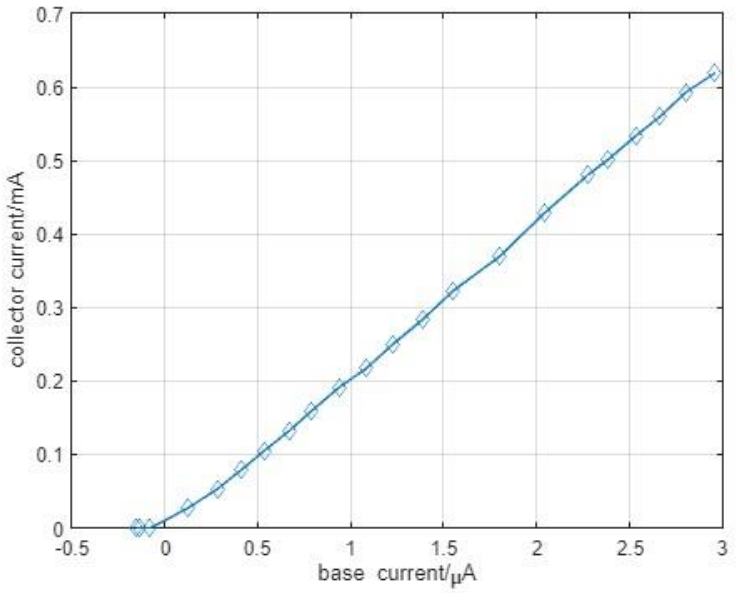


Figure 4. Switching to $Rc = 681\Omega$ *appears to change nothing.*

The data points have moved around, but the min and max appear the same, 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 = 213.1
```

This suggests that the collector resistor doesn't affect this curve. The slope is indeed a *characteristic of the transistor*. We then increase Rc such that say at ic ~0.5mA, ic*Rc ~ vref --> Rc = Vref / 0.5mA ~ 10kohm.

```
ib681 = ib; ic681 = ic; beta(2) = p(1)*1e3; % save the former results
Rc = 10.03e3; % collector resistor data = [... 1023 130 89
1023 130 91
1023 130 90
1022 129 93
1021 129 97
1002 130 118
962 130 142
892 130 175
835 130 201
787 131 221
742 131 240
705 131 255
657 130 276
```

```
623 130 290
594 131 303
555 131 319
512 131 337
453 130 361
369 130 397
261 130 441
194 131 470
126 131 498
75 131 518
38 131 546
31 131 567
27 131 604
26 131 625
25 131 655
24 131 679
22 131 712
21 132 754
20 131 794

] * 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')
```

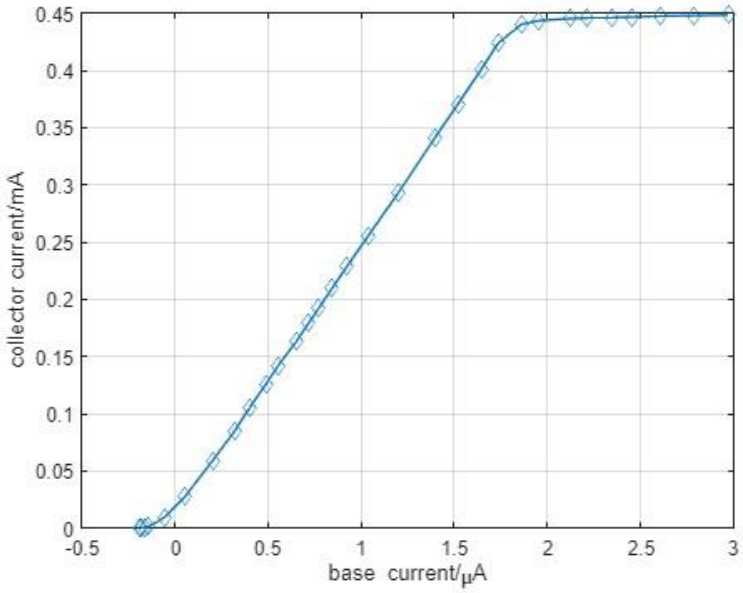


Figure 5. Collector vs. base currents with $Rc = 10k\Omega$*, which causes the BJT to saturate around* i_C $.45mA$.

We superimpose these 3 plots:

```
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')
```

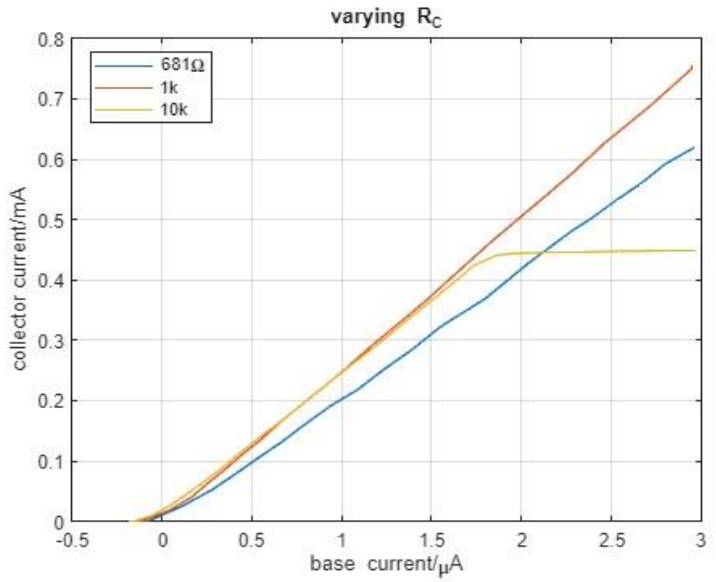


Figure 6. Superposition of Figs. 3-5, all values of Rc show similar slope before saturation.

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

```
beta = 236.8
```

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 = 0.4576
```

compared to the graph:

```
max(ic)
```

```
ans = 0.4487
```

Discussion

This lab was interesting in a lot of different factors. Learning more functions of the Arduino and the power that it has is amazing. MATLAB is equally as interesting. Making a function that can interpret and graph the raw data spewing from the Arduino is impressive. Both of the software have amazing potential for more functionality it other labs. Learning more about how transistors function and what an HFE it was helpful in my general understanding of electronics. Personally, I think transistors are confusing and complex but improving my understanding of HFE's vastly improved my understanding of a transistor.

These curves aren't i-v curves, but ic vs ib curves. This makes sense since transistors have 3 terminals, and hence 2 circuits with a common emitter, and BJTs are current-controlled-current devices. At the bottom left of Fig 6, the transistor is in **cut-off**, with both currents ~0. Next 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.

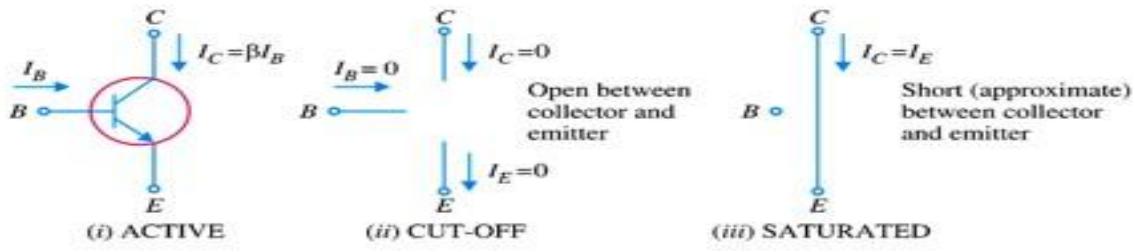


Figure 7. Three states of a BJT.

The value β of is a characteristic of the transistor, and relatively independent of the circuit. Many β factors affect , including geometry, doping and structural composition, and temperature. Unlike resistors, who's values are specified within a narrow range, for example, 1% or 5%, the range of β is often extremely wide, for example, 100-400. When using

transistors one generally doesn't design for a particular value. Good designs 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. We might say that my transistor today had:

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

```
beta = 235±22
```

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.

Question: 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), to transition from cut-off to saturation or vice 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.

A clock is what makes them digital. All computers, microcontrollers, and digital systems have a clock 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.

This is not the answer that I expected because while I had the reference for digital and analog differences, making the connection that a clock can make something digital does not come to mind. However, now that I have heard it, it makes sense.

Arduino Controlled DC Power Supply

Objective

Make, document, and describe an accurate model of the Arduino controlled voltage regulator circuit we have built.

Methods

-Hardware Interface

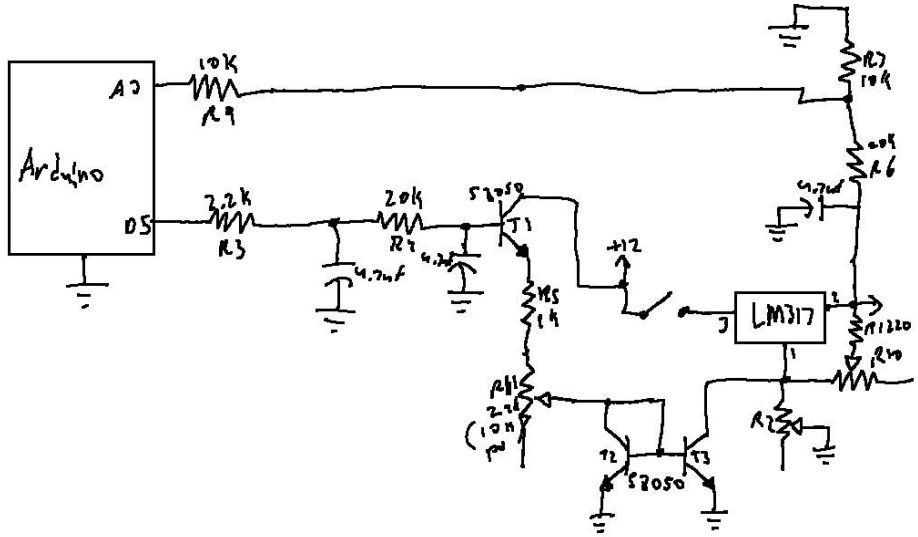


Figure 1: Schematic of Arduino controlled LM317 circuit

The point that is connected to D5 on the Arduino is what controls the LM317. That segment of circuit is composed of a PWM filter, emitter follower, and current mirror. The current mirror is the actual segment that controls the LM317. The PWM, or Pulse-width Modulation, is a technique used to control analog circuits with digital outputs. We use it to make a communication between the Arduino and the first pin on the LM317. It works by generating low frequency output signals from high frequency pulses. The low frequency output is thought of as an average voltage over a switching period. The emitter follower is the first transistor, labeled as T1 in Figure 1, which is the negative current feedback circuit. It provides high input impedance and low output impedance, making its purpose impedance matching. A current mirror is a circuit that copies a current through one active device by controlling the current of another device which keeps the output constant. The two transistors labeled T2 and T3 is the current mirror. This is how the current mirror controls the output voltage of the LM317. It controls the current of the LM317 and copies it to control its output voltage.

Software Interface

We use code provided by Professor Rasnow to receive data from our circuit using the Arduino and using the Arduino to control our circuit.

```
void measureWaveformsBinary(void);

const int analogOutPin = 5;
int settleTime = 200; // msec
#define NUMSAMP 400
#define MAX_SETTLING_TIME 5000
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;
        }
      case 's': // settling time (in msec)
        {
          int t = Serial.parseInt();
          settleTime = constrain(t, 0, MAX_SETTLING_TIME);
          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
```

Figure 3: Arduino code to control LM317

Data Interface

Our MATLAB code and Arduino code communicate by taking data output from the Arduino and applying those numbers to variables in the MATLAB code. MATLAB interprets the data and displays a live updated graph that acts like an oscilloscope. MATLAB displays the oscilloscope using the file oscope1.m which makes the figure window and a run button for us to start the code.

Calibration

```
const int CommSpeed = 19200; // baudrate
const int VoutPin = A0;
const int analogOutPin = 5;
const float VREF = 4.59; /* volts on Vref pin */
volatile int Vout;
int pwmValue, i, j;
void setup()
{
// initialize serial communications at 9600 bps:
  Serial.begin(CommSpeed);
  pinMode(analogOutPin, OUTPUT);
} // setup
void loop()
{
  Serial.println("Vout = [...");
  for (i = 0; i <= 255; i++)
{
Serial.print(i); Serial.print("\t");
analogWrite(analogOutPin, i);
for (j = 0; j < 5; j ++)
{
delay(5);
Vout = analogRead(VoutPin);
Serial.print(Vout);
Serial.print((j < 4) ? "\t" : "\n");
}
}
 Serial.println("];");
analogWrite(analogOutPin,0);
delay(1000);
} // loop
```

Figure 4: Arduino code for calibrating the LM317 automation

With this calibration code, the voltage of the DC power supply oscillates according to the potentiometer. Its voltage is displayed on the voltmeter and shows it oscillating in a sine wave manner close to its maximum and minimum values, or around 10.5V and 3.51V. The PWM value correlates to the voltage through MATLAB and the PWM data is measured and modeled on MATLAB graphs with polyfit.

Testing

The Arduino collects data and interprets it into MATLAB. We can now use MATLAB after calibration and using the code from Figure 3 to use the command line to control the voltage of our DC power supply.

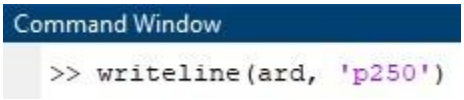


Figure 5: Command to change voltage of DC power supply

By changing the second variable in the command we can choose the voltage we want. For example, when we enter ‘p250’, we get the lowest voltage of our supply, 1.25V. However, if we change it to ‘p50’ we get the highest voltage, 11.3V. This connects the PVM data from MATLAB and converts it into data that can be interpreted by the Arduino, which then uses that to change the voltage on the PCB’s PWM filter.

Results

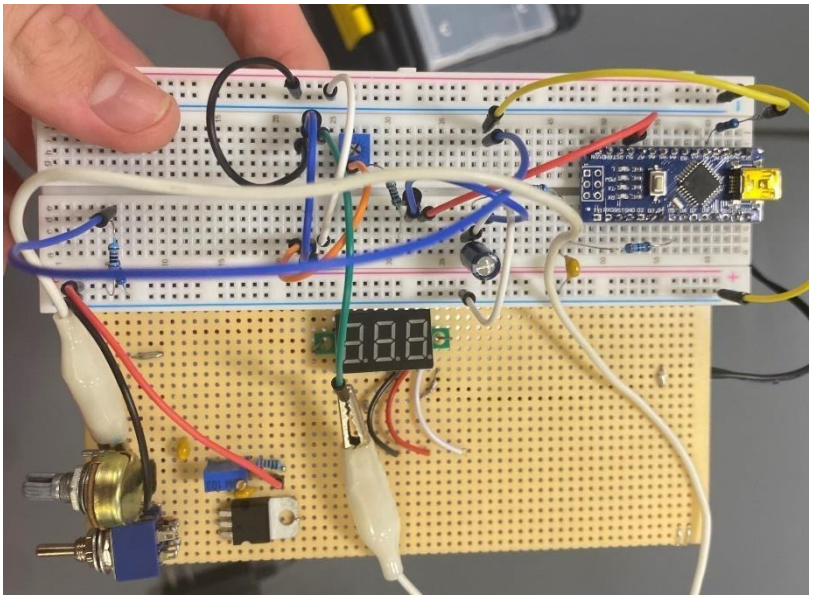


Figure 5: Prototype of full Arduino Controlled LM317 circuit

After doing a smoke test with the circuit on the solderless breadboard and making sure that correct data is getting sent to through the Arduino, we then put that same circuit on to our PCB. We solder and wire it to our DC variable power supply and connect the needed wires to the Arduino onto the breadboard.

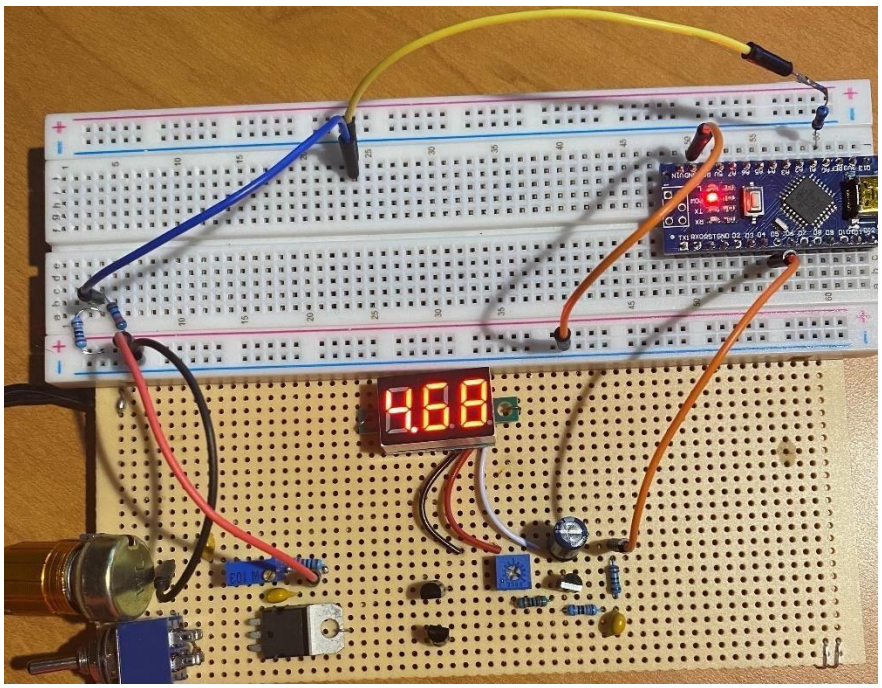
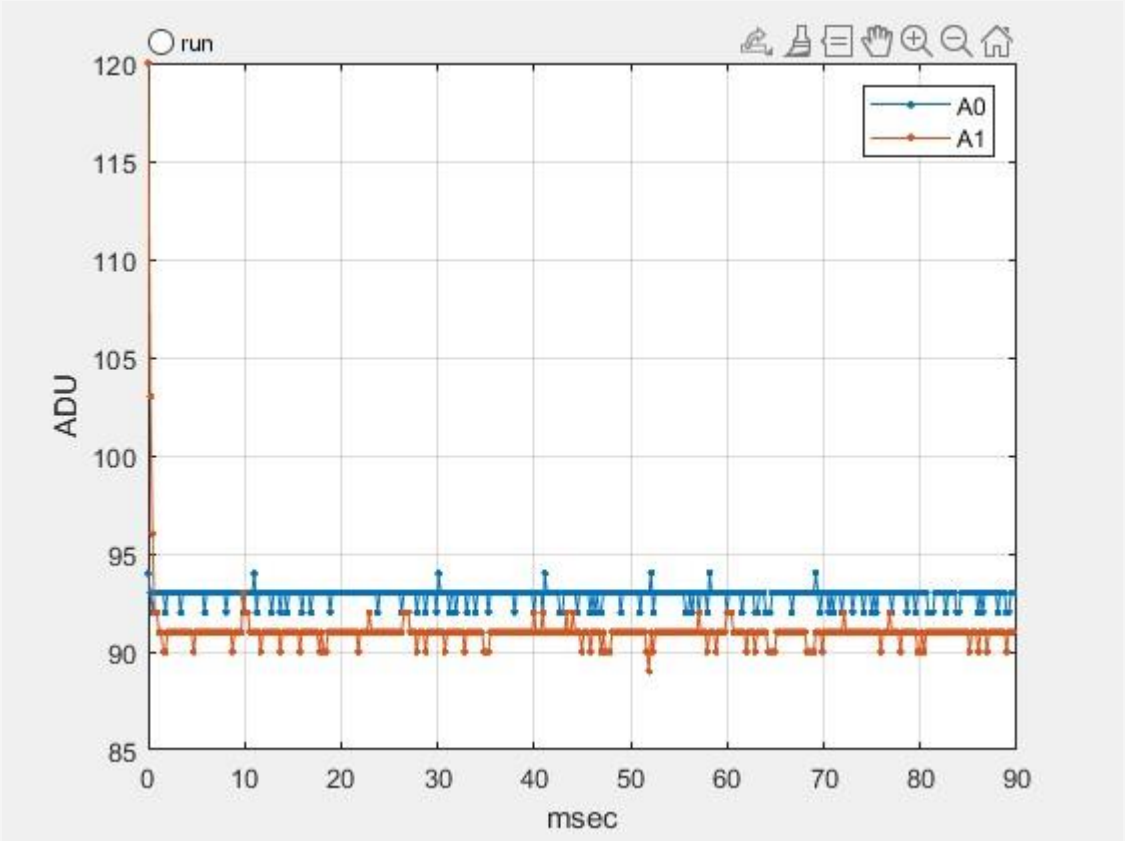
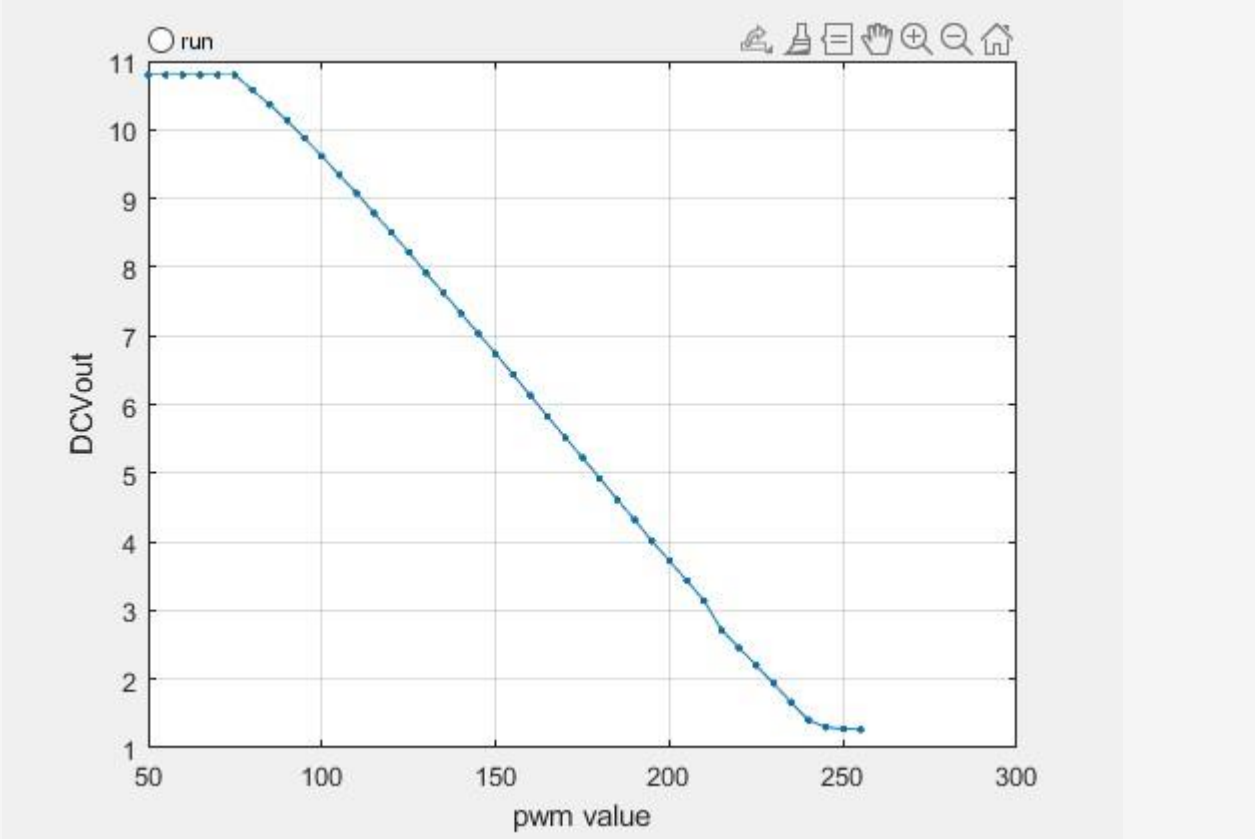


Figure 6: Final product of Arduino controlled LM317 circuit soldered to PCB.

```
vref = 4.62; % volts on vref pin
R7 = 9.92e3;
R6 = 20.08e3;
%ard = serialport("COM3",R6);
%flush(ard);
pwm = (50:5:255)'; % 42 different values
v317 = zeros(size(pwm));
oscope1;
initializing arduino ...
tic;
for i=1:length(pwm)
    writeline(ard,['p' num2str(pwm(i))]);
    pause(.2); % 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
```




```
toc
Elapsed time is 24.318305 seconds.
plot(pwm, v317, '.-'); xlabel('pwm value'); ylabel('DCVout'); grid;
```

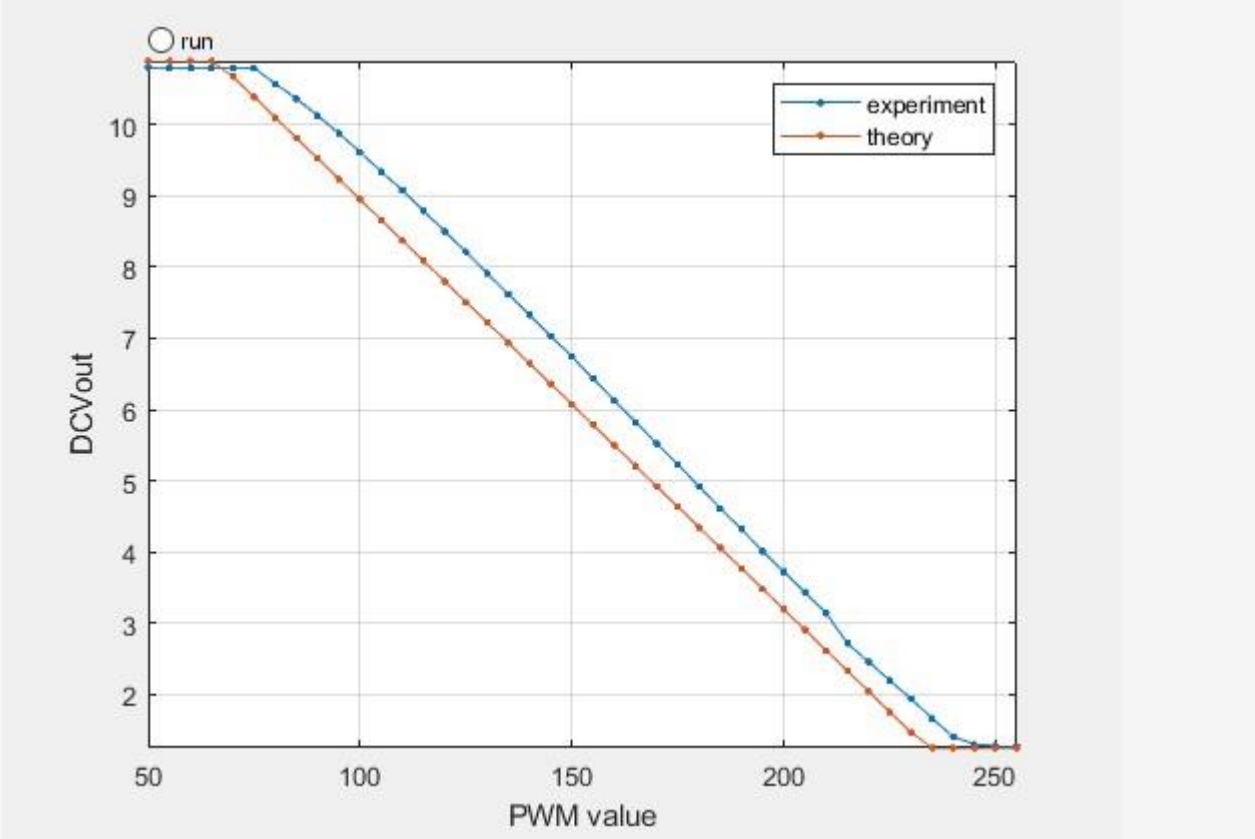


The graph above shows the Vout of the DC power supply vs the PWM value. We can trim the emitter follower pot to make the slope flatter but the difference it would make would be minimal. The flat part below 50 is because the PWM voltage is too low to turn on T1.

Modeling the circuit in Matlab

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 unduely load D5 (5V/1k=5mA), and the larger 2nd stage's impedance, R4 won't onduly load the first stage. An emitter follower (T1) lowers R4's impedance by 1/hfe, driving current through R5 and R11 from Arduino's 5V supply, proportional to $(V_{T_1b} - V_{T_1be} - V_{T_2be})/(R_5 + R_{11}) \approx (V_{PWM} - 1.2)mA$.

```
%% model 1
R1 = 1400; % ohms measured between LM317 pins 1 and 2 (with power off)
R2 = 10.8e3; % max ohms measured between ground and *unconnected* terminal
R5 = 3.4e3  ; % R5+R11, calling it R5
T1baseVoltage = pwm * vref / 255; % the PWM filter's gain = 1 at DC
T1emitterVoltage = T1baseVoltage -.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')
```

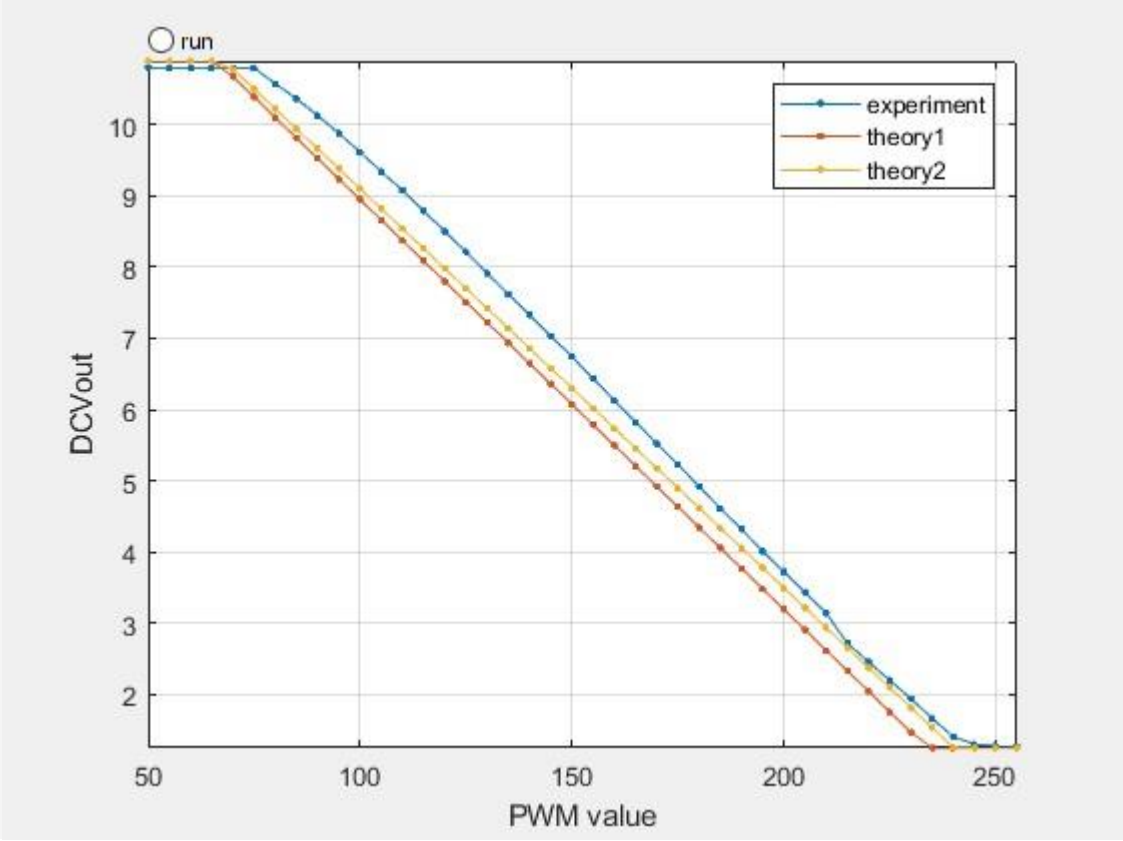


The experiment appears to agree with the theory somewhat. We were able to model thresholds where the circuit behavior changes (e.g., saturation) using inline logic, i.e., DCVout(DCVout > Vmax) = Vmax; and iMirror(T1baseVoltage < 0.6 + T2collectorVoltage) = 0; instead of more verbose if/else constructs show the power of Matlab's syntax. Notice how this is different than how Eagle would simulate the circuit. We didn't model every component, e.g., the PWM filter doesn't appear, because we're only modeling DC behavior and at DC the capacitors are infinite impedances (|1/j2*pi*infinity*C|). What about T1? It's emitter current < vref/R5 = 1.4mA, divided by hfe~240 = 7uA base current through R3 and R4. That should cause a voltage drop at Vb by 7uA*(R3+R4) = 0.14V, a small effect, but we can model it that as a voltage divider with "R1" = R3+R4 = 22k, and "R2" = R5*hfe, so:

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

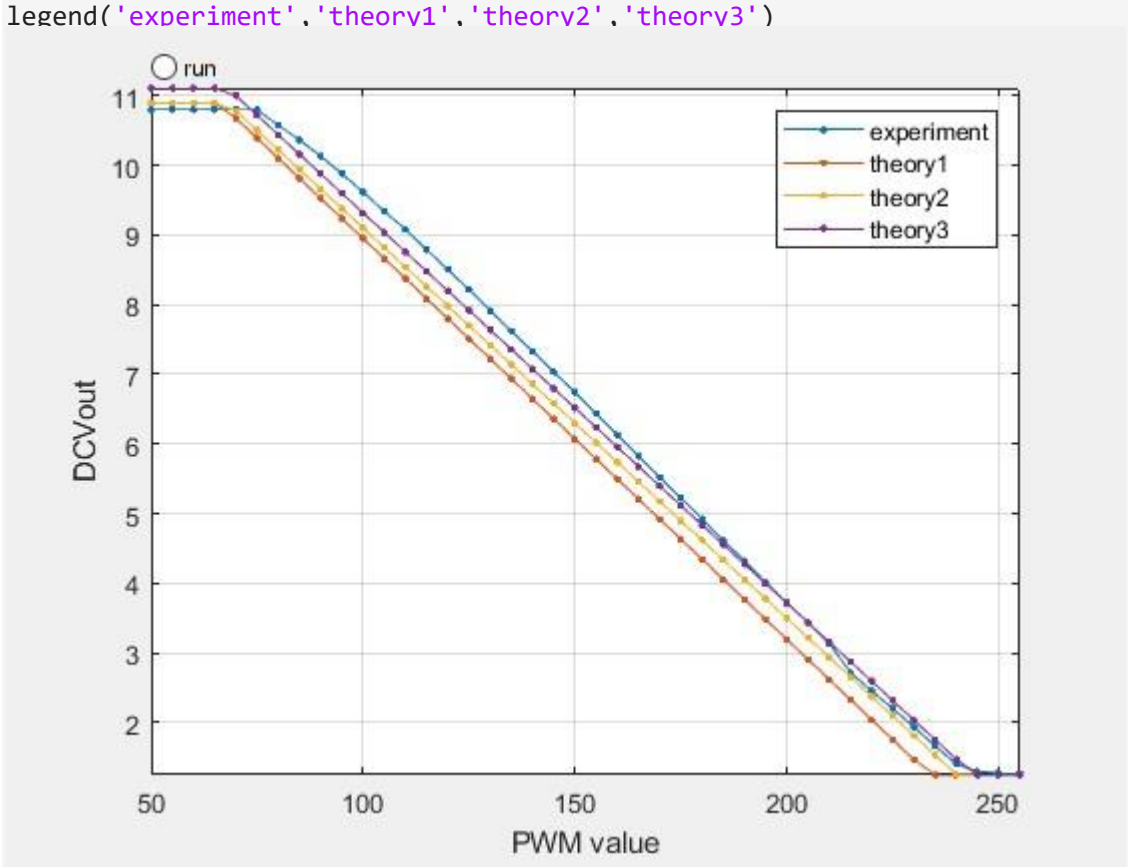
```
%% model 2, with hfe
hfe = 240; % estimated
T1baseVoltage = pwm * vref / 255  * (R5*hfe)/(R5*hfe+22e3); % see below
T1emitterVoltage = T1baseVoltage -.6; % Vbe in saturation
iMirror = (T1emitterVoltage-T2collectorVoltage) / R5; % A
% if T1 isn't conducting then iMirror = 0:
iMirror(T1baseVoltage < 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');
dots; grid; axis tight;
legend('experiment','theory1','theory2')
```



We will add one more tweak. 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');
dots; grid; axis tight;
legend('experiment','theory1','theory2','theory3')
```



Testing

Using the v-command, “writeline(ard,’p50’) as we did in Methods with testing our calibrated circuit we can see if data is being received and sent. Our figure in MATLAB when the code runs, starts the DC power supply at max voltage and drops it to minimum to read all the data from it. If the display is showing that occurring as well as the experiment like matching the theory, then these tests are accurate.

Conclusions and Discussions

This lab was intensively long and much more complicated compared to the other labs that we have done. Adding a whole other section to our PCB was something that we haven’t done since the first lab. Also learning some new techniques used in electronics and how they work is an important part of this class. Along with that, we are combining the usage of Arduino and MATLAB code, which I believe to be the most interesting and complex part of this lab. Doing the hardware of an electronic circuit comes with a lot of troubleshooting which is tedious and annoying, but with a schematic, it’s relatively simple to understand the hardware of a circuit. However, seeing how Arduino and MATLAB code can communicate with each other is the most useful and meaningful purpose of this lab. It introduces the idea of porting information across programs and applying them into our circuit to send and receive data. By controlling our DC power supply with our Arduino and interpreting the data we receive from it with MATLAB, we are shown how to utilize all the information that we have learned in past labs and combining and connecting our work together.

Function Generator

1. Objective

Construct a function generator using an XR2206 IC kit to display sine and square wave voltages.

2. Methods

Hardware Interface

Construction

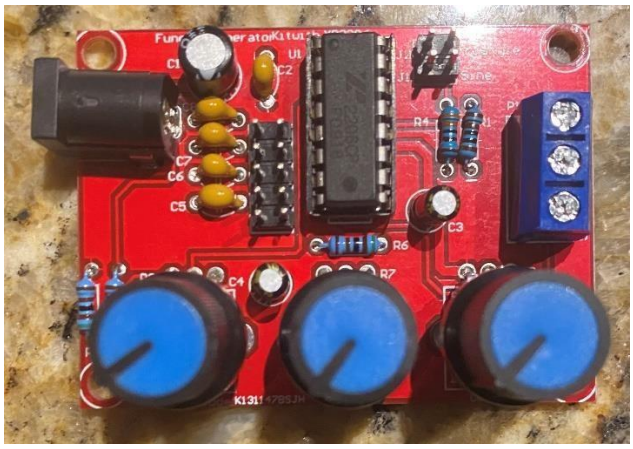


Figure 1: Completed XR2206 IC function generator

In our electronics kit we get a blank XR2206 IC kit. Along with that is a little bag full of all the parts that we need to solder to the board. The IO pins on the board in Figure 1 have no connections but by using the green pin connectors given in our kit, the frequencies in Hz of the generator can be exponentially increased based on how far up you place the connector. The other IO pins in the top right of Figure 1 designate whether the generator will display the wave as a sine or triangle form.

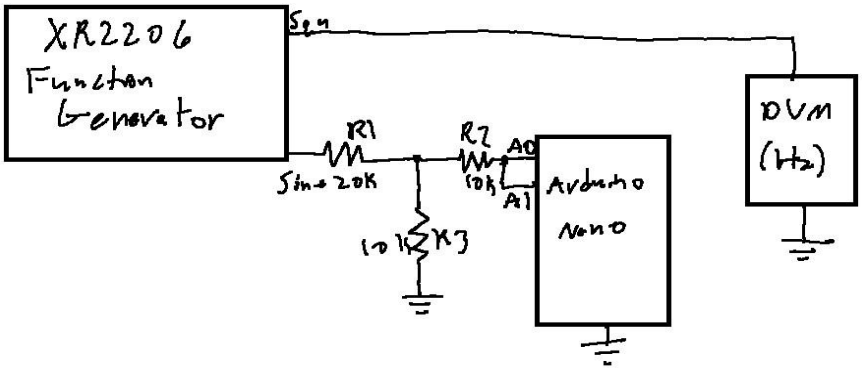


Figure 2: Schematic of XR2206 connected to Arduino and DVM

We connect the Square wave pin to the multimeter (DVM) to measure the frequency and the sine wave to the arduino. A voltage divider is used to drop the voltage going into the arduino to not damage it.

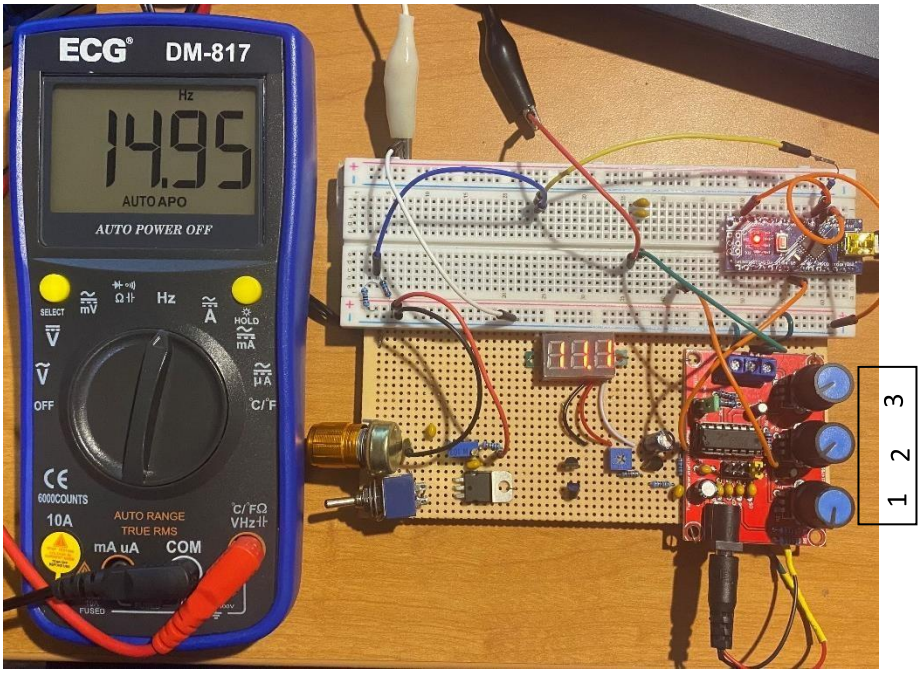


Figure 3: XR2206 IC on DC power supply PCB and wired to Arduino on the solderless breadboard

The DC power supply is connected to the XR2206 IC by the end of our 12V wall wart plug. The positive end is soldered to the output of the switch on the power supply, and the negative is soldered to ground. On the XR2206, a blue pin connector with three connections: sine, square, and ground. The sine connection is connected to the positive on the breadboard, and ground connected to ground. The square wave connection can be used to see what our frequency is by connecting that to the positive on our multimeter, and negative to ground. Now that the frequency is being displayed it is deducible that the potentiometers on the XR2206 change the frequency of the function. In Figure 3, pot 2 is described as fine and pot 3 is course when changing the frequency. Pot 1 changes the amplitude which will be shown in results.

Software Interface

Arduino Software

```
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
```

Figure 4: Arduino Code for Function Generator

MATLAB software

```
% oscope3 -- adding freq, amplitude, and phase to the display
% oscope2 -- adding trim for triggered display
%% 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 = '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, 410); % 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
    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).String = sprintf('%.1fHz',freq);
    catch
        plotHandle = plot(t, data, '.-');
        xlabel('msec'); ylabel('ADU')
        legend('A0','A1'); grid on;
        plotHandle(3) = title(sprintf('%.1fHz',freq));
    end
    drawnow;
    runOnce = false;
end
```

Figure 5: MATLAB oscilloscope software (oscope3.m)

The Arduino code in Figure 4 is for taking the information given by the XR2206 and DC power supply and makes data from the PWM values given to the Arduino. Pin D5 on the Arduino is connected to the oscilloscope circuit we made in the Arduino controlled LM317 lab. Pins A0 and A1 are connected to the XR2206 to give data of the wave being produced by it. MATLAB then takes the data from the Arduino and draws a live graph which displays the wave that the XR2206 is generating.

Testing and Validation

```
%% 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
```

```
initializing arduino ...
Warning: The specified amount of data was not returned within the Timeout period for
'serialport' unable to read any data. For more information on possible reasons, see
```

```
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
```

Figure 6: Oscope1.m code for testing

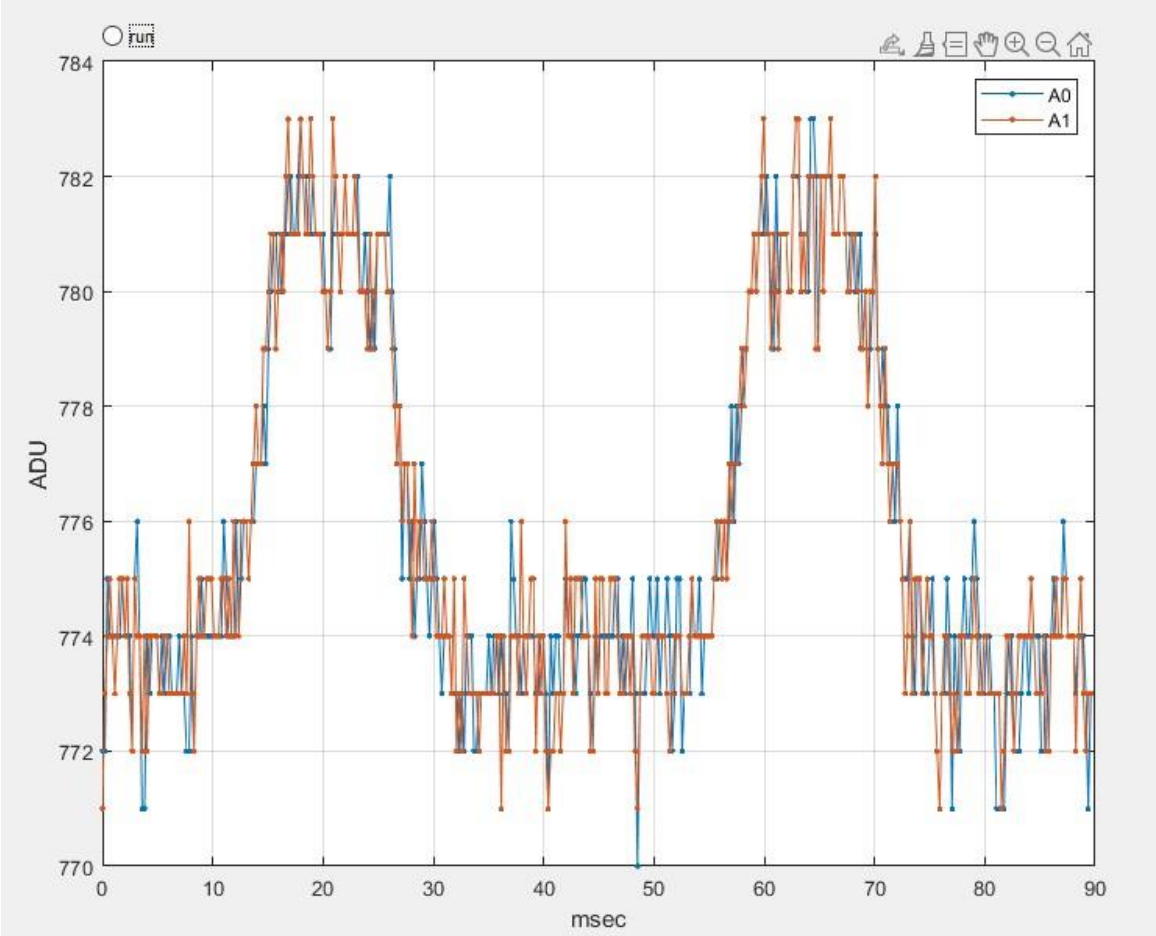


Figure 7: Oscope1.m wave

Using old code from the previous lab, we can test if the Arduino is sending data and MATLAB is interpreting it properly. Also, this test can check if out XR2206 is correctly connected to our entire circuit. As you can see in Figure 7, the wave looks very jagged. By using the code in the mlx file provided by Professor Rasnow the wave can be trimmed to give a much smoother looking wave. That smoother wave is gotten by using the code form oscope3.m in Figure 5.

3. Result

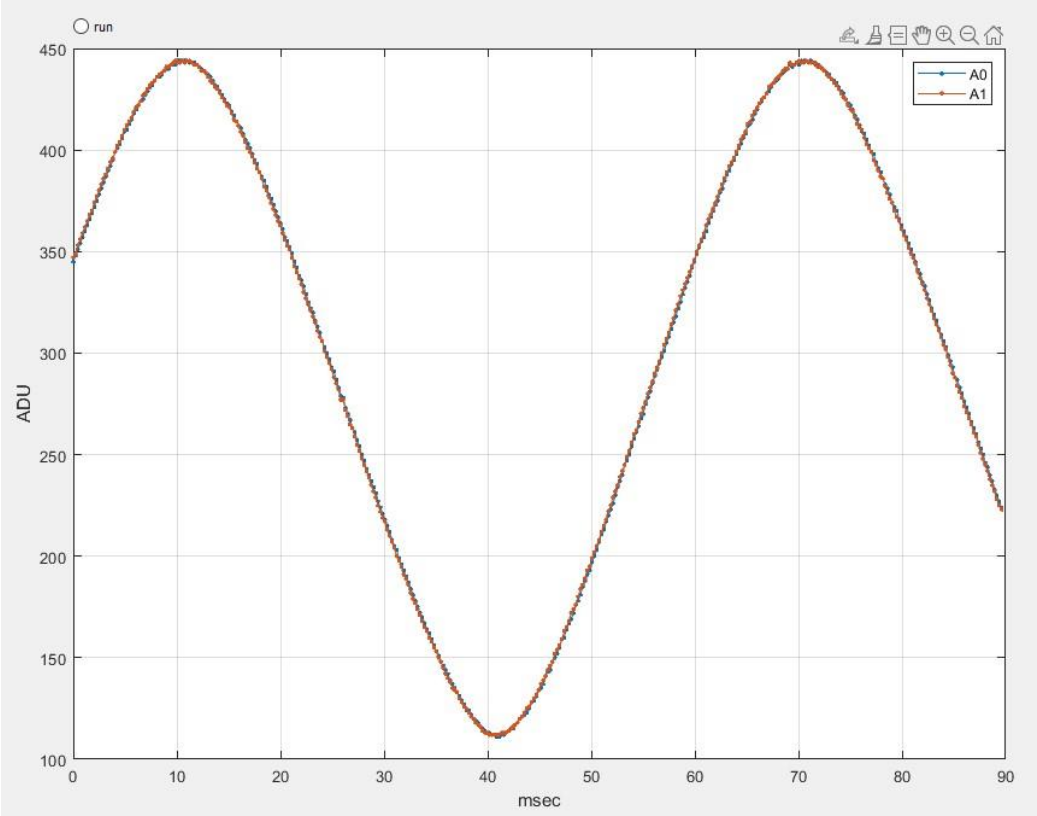


Figure 8: Oscope3.m wave

Oscilloscope file 3 displays a much smoother looking sine wave. To calculate the frequency of the wave we can see that the first peak to the second is at 10ms and 70 ms. That means the wave has a period of 60ms. Frequency is equal to 1/T, where T is the period. That means the frequency is 16.66Hz which is concurrent with the multimeter. Turning the potentiometers that changes the frequency up causes the waves to become much thinner, meaning the frequency is increasing.

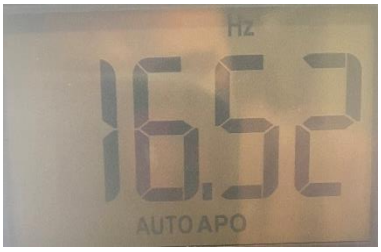


Figure 9: Multimeter Frequency reading

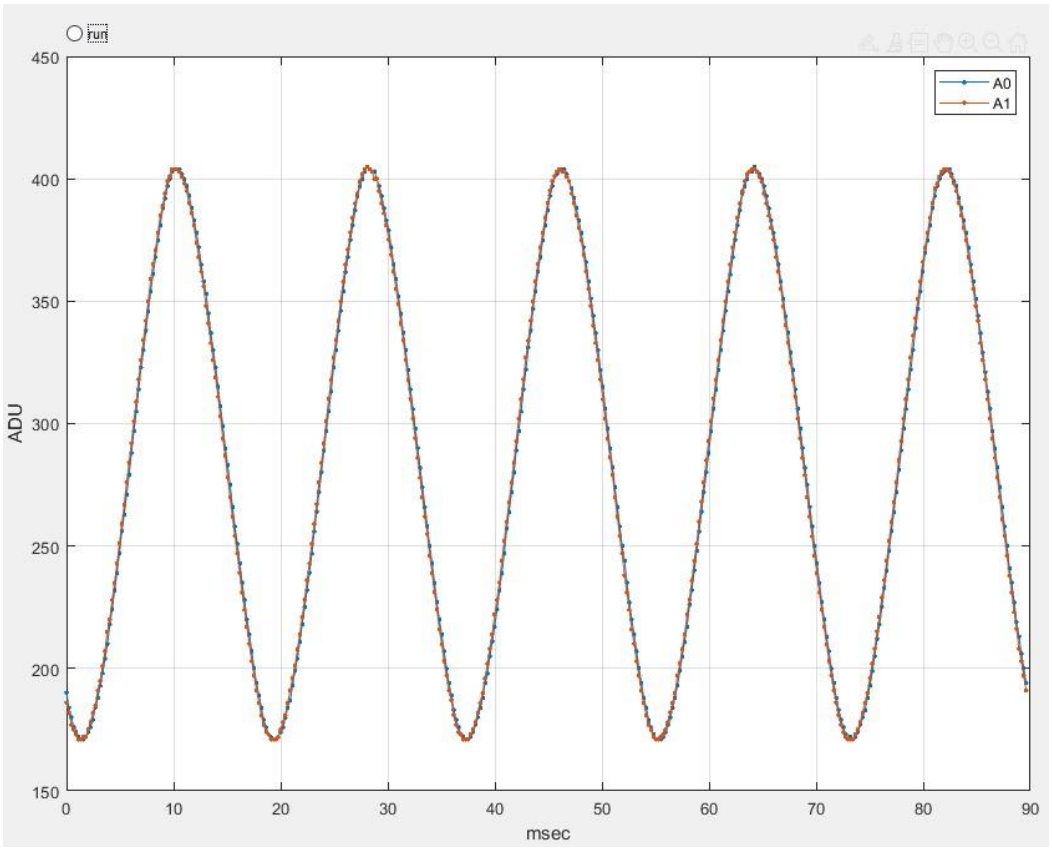


Figure 10: Higher frequency oscope3.m

The period in Figure 10 is about 20ms so the frequency is 50Hz which is also about what the multimeter shows give or take. Pot 1 on the XR2206 changes the amplitude and turning it changes the ADU value on the graph higher or lower respectively.

The multimeter and MATLAB data show that the XR2206 is successfully generating sine and square wave functions.

Figure 11: Multimeter Reading

The frequency range is found by turning all the frequency changing pots all the way down and then all the way up and recording the numbers. Using this method, the frequency range is 10.4Hz to 204Hz. However, by moving the IO pin connector up on the XR2206, we can increase the frequency range. Moving it to the middle pin gives us a range of 225Hz to 4.4kHz. A much larger range compared to the pins we used to collect our data.

4. Conclusion/Discussion

The biggest component of this lab was the XR2206 IC kit. Building the function generator gave me a lot more experience with soldering. The small components being put on the board and soldering in-between tiny spaces was a challenge and took some precision that I had to acquire while doing it. Understanding how all the components worked and connected was incredibly interesting. Another large component of this lab that was a struggle but very informative was the Arduino and MATLAB code. We used an oscilloscope in the last lab using MATLAB to interpret Arduino data and this time was very similar. However, we used a much more sophisticated MATLAB file that trimmed out the jagged data and allowed the Arduino data to be interpreted as a sine wave in MATLAB. Looking through how the code works, especially in MATLAB, really shows how powerful the MATLAB software is. Coding something like an oscilloscope in C++ would be wildly more difficult. MATLAB is amazing for electronics.

Years after this class and I remember that I have my PCB that I made, I would need to know some information about troubleshooting, especially if it is broken. The first advice I would give myself is use your multimeter. Check if there is continuity in my XR2206 and my entire DC power supply so there is no danger of ruining my Arduino. If I don't have the MATLAB files or Arduino code, then I would need to recode it myself, so becoming more proficient at Arduino and MATLAB programming would be very helpful as well. Having the files saved somewhere would be helpful too. Also make sure that the Arduino is connected to the correct port and its sending and receiving data from it, or else the Arduino code won't work, or MATLAB can't read it. Commands in MATLAB are good to remember too. Make sure that the functions have correct values or values at all. Bod rate problems and values not existing are problems I have had to fix many times.