ECEN 3730-
PCB Design,
Connor Sorrell

PCB Design Board 4 Report: Instrument Droid

1. Introduction

This board served as a final test for almost everything we learned throughout the course. On this board, the goal was to extend my Golden Arduino, a fully functional Atmega328P-based microcontroller board, with a standalone measurement system to characterize the Thevenin resistance of any voltage source. Board 4 brought together a 4 layer board stackup, real-time data acquisition capabilities, custom firmware, all attached to a custom built Arduino Uno.

Additionally, this project combined everything from MOSFET current sourcing to DAC driven pulsing, as well as ADC based voltage sensing, I2C communication, and more. Beyond simple hardware & PCB engineering, this board tested debugging methods, smart design decisions, integration of firmware, and physical assembly on a scale I have not attempted yet.

After countless hours of debugging, the resulting board was a fully operational instrument. It acts as an Arduino-but with better performance, signal integrity, and noise mitigation. It also produces clean, accurate thevenin resistance measurements of many different voltage sources.

2. Design Goals

- Build a 4-layer board with strong design choices in mind
- Measurement system to calculate Thevenin resistance
- Integrate an ATmega328P microcontroller to mimic an Arduino Uno
- Include a DAC (MCP4725) and ADC (ADS1115) to handle analog output and measurement
- Use a MOSFET + op-amp constant current source controlled by DAC output
- Add smart indicators: RGB LED + buzzer to represent feedback/real-time measurement results
- Measure a variety of power sources and export data for analysis in Excel
- All power control & data acquisition handled by the onboard microcontroller

3. System Architecture

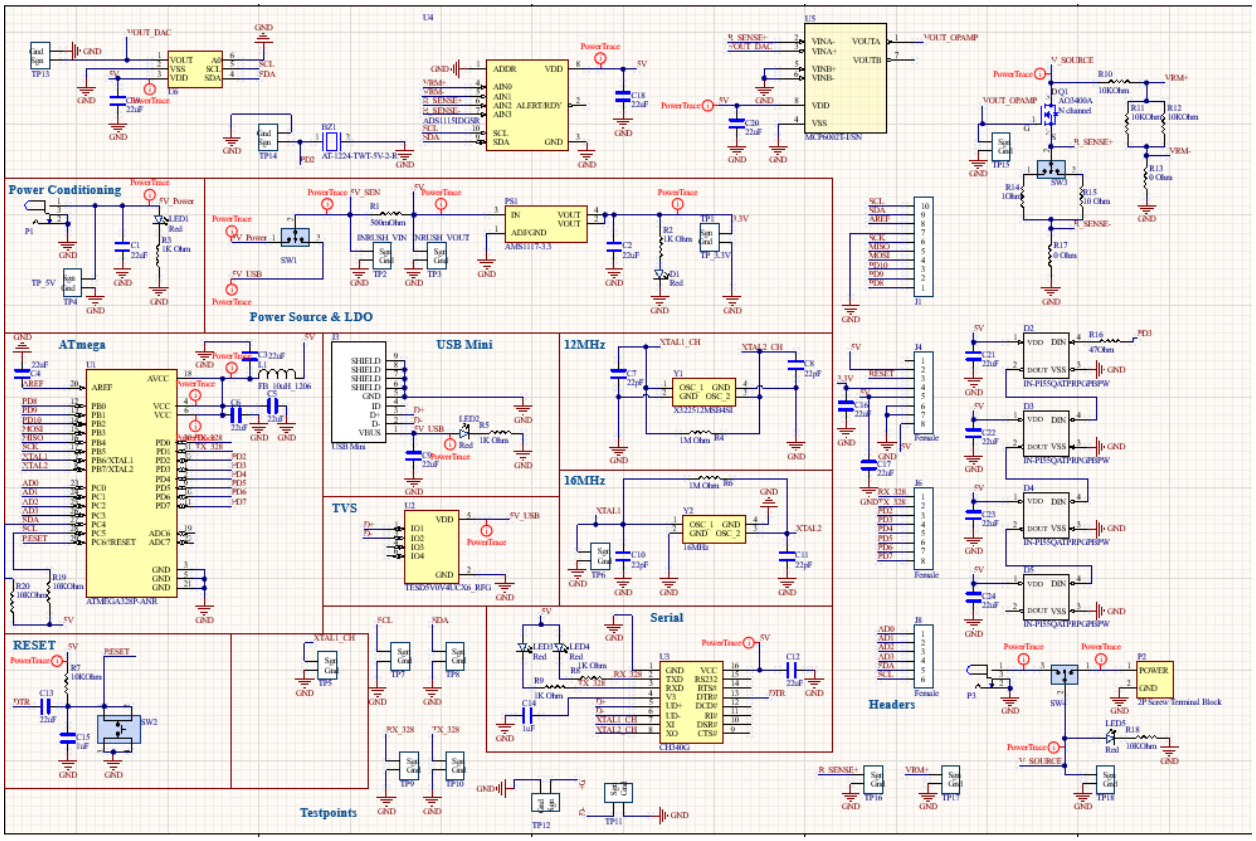


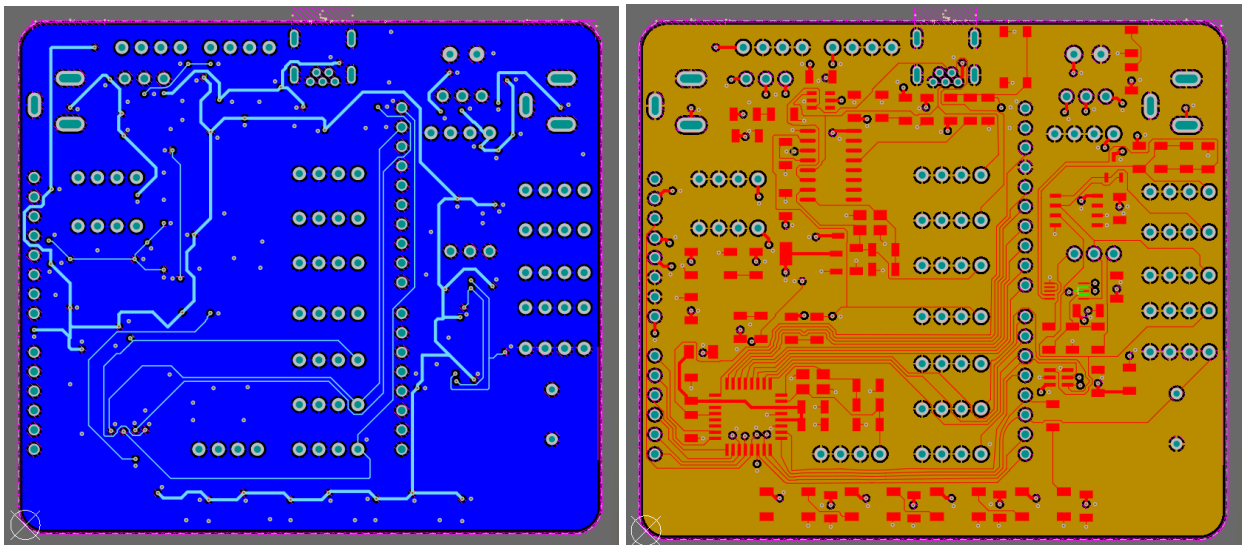
Figure 1: Annotated schematic in Altium

The system can be broken down into:

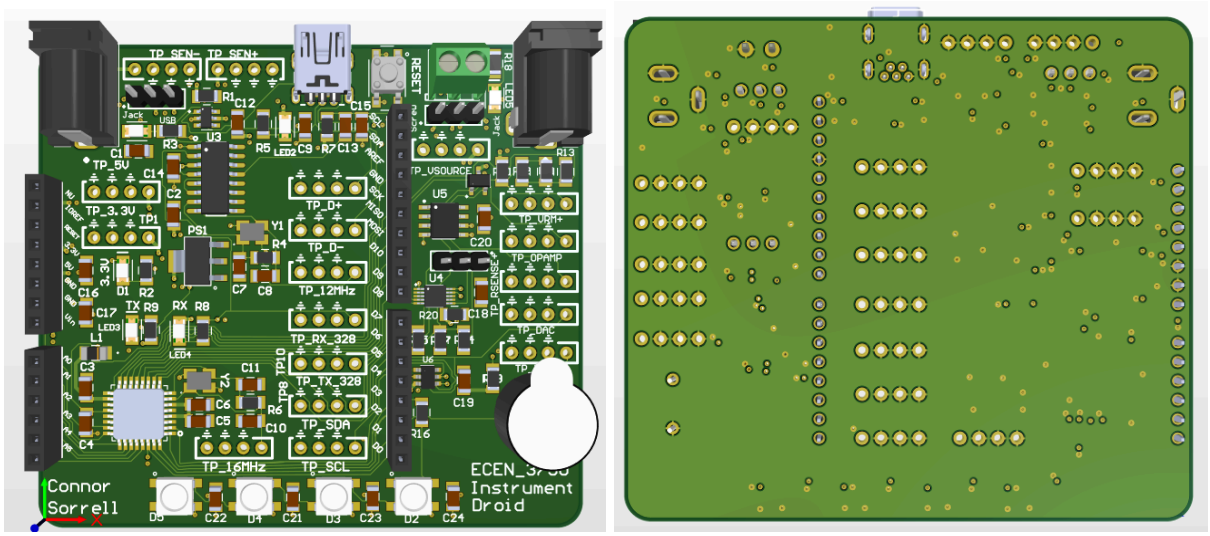
- DAC Section: MCP4725 generates the voltage setpoint that controls current draw
- Op-Amp + MOSFET: Amplifies DAC signal to control AO3400A gate
- Slammer Circuit: draws current through the load while measuring the voltage drop.
- ADC Section: ADS1115 reads the load voltage and sense resistor voltage
- AtMega328P: controls everything via I2C, generates pulses, firmware for math calculations
- Indicators: RGB LED and buzzer provide user feedback during operation

The overarching goal of the board is to safely stress test power sources and characterize them under varying loads, while also providing LED / Buzzer feedback to show the results of the tests in real-time.

4. 4-Layer Stackup and PCB Layout



Figures 2 and 3: Show the 1st and 3rd layers (routing layers)



Figures 4 and 5: Show the final PCB Layout, both front and back

Layer Assignment:

- Top Layer – Signal (components + high speed critical routing)
- Layer 2 – Solid Ground Plane
- Layer 3 – Additional Routing (mostly 5V, 3.3V and Vin signals)
- Bottom Layer – Solid Ground plane

Layout decisions were driven by signal integrity and testability:

- Crystal and CH340G were traced for minimal loop area and trace symmetry (as close together as possible)
- Decoupling capacitors placed as close as possible to each power pin
- Power rails (VCC, AVCC, USB 5V) routed thick (20mm)
- Ferrite bead isolates AVCC from digital switching noise
- Continuous return plane present
- Test points labeled and accessible

Ground and signal vias were strategically placed to ensure all high-speed signals had close adjacent return paths. Decoupling caps were placed within mm of their respective IC power pins. More than just connecting components, I focused on managing power delivery in a smart and efficient way to optimize board performance and keep measurements stable and accurate.

5. Assembly and Fabrication

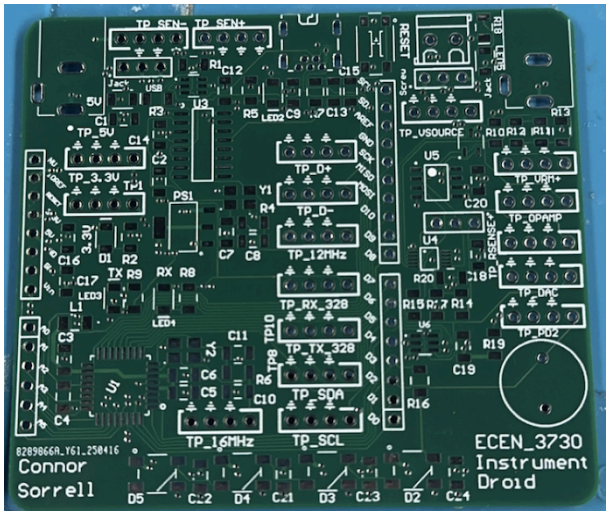


Figure 6: Shows unassembled, bare bone board

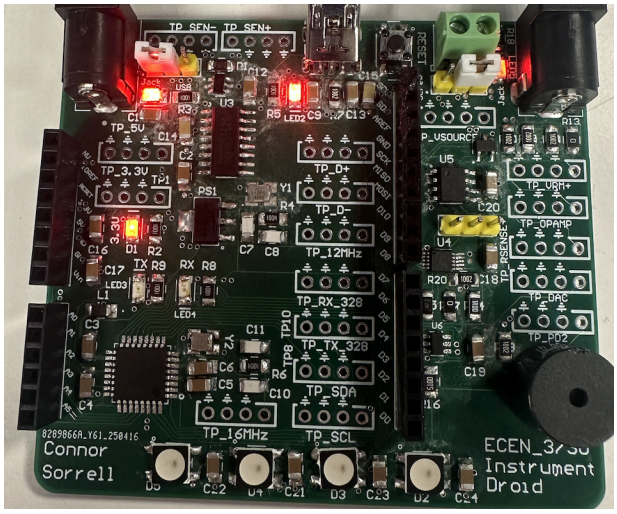


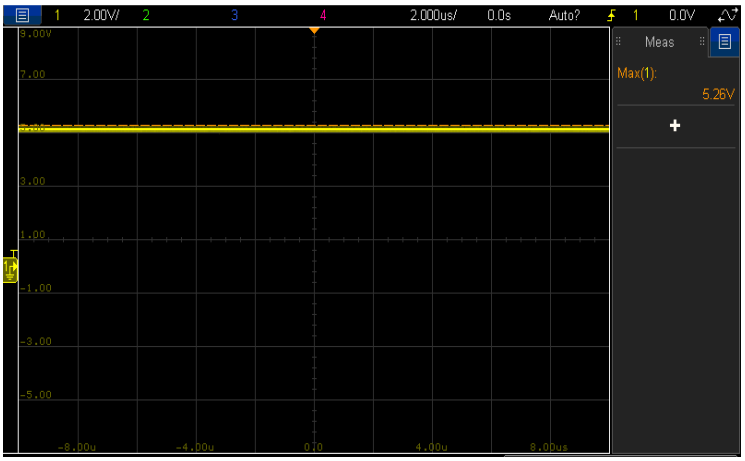
Figure 7: Shows board fully assembled

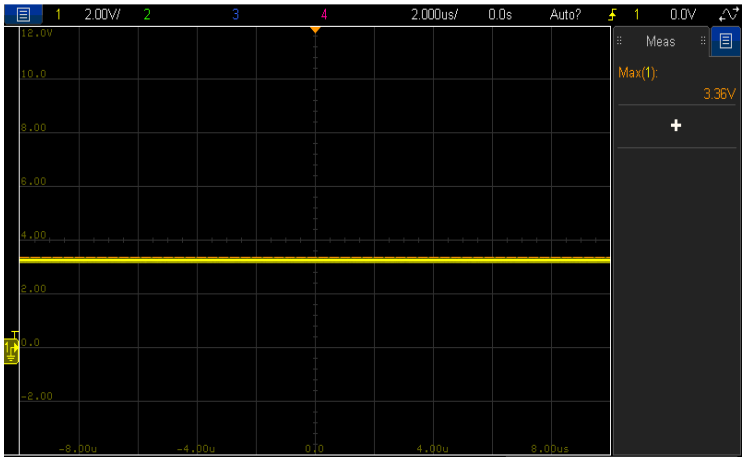
Assembly was done by hand with a combination of hot air reflow and fine-tip soldering. Particular care was taken with:

- The Op Amp, which was by far the smallest SMD part I've soldered. It required me to be extremely precise and use a sharp iron tip
- AtMega, CH340, other IC's with a lot of pins. For these, I used the hot air gun
- Debugging the CH340 later taught me just how fragile these connections can be.

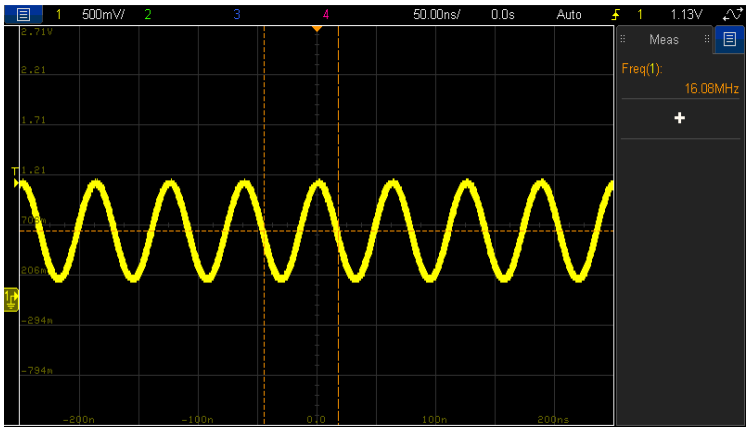
Functional Verification

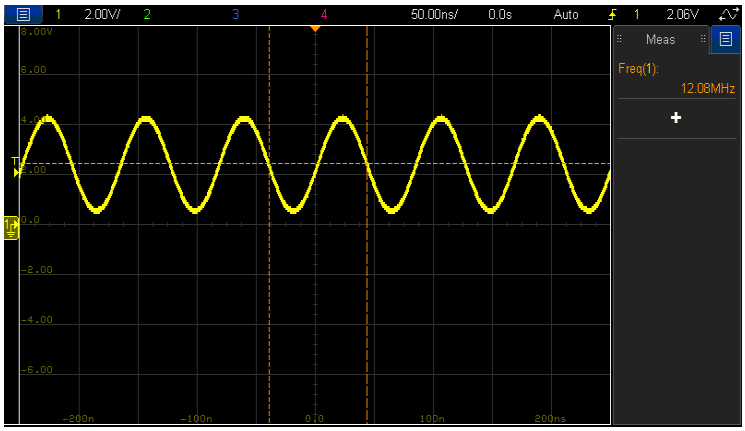
6.1 Baseline sanity checks for Board power, USB communication, and AtMega





- *Figures 8 and 9: Confirm proper power delivery on the board and LDO is working as expected to convert 5V to 3.3V*





- *Figures 10 and 11: Confirm both the 12MHz and 16MHz crystals are working properly*

- The 5V and 3.3V working as intended mean that the PDN, for the most part, works as intended. The 16MHz crystal confirms a successful bootload, and a working 12MHz crystal is a strong hint towards successful USB communication.

6.2 USB mini check

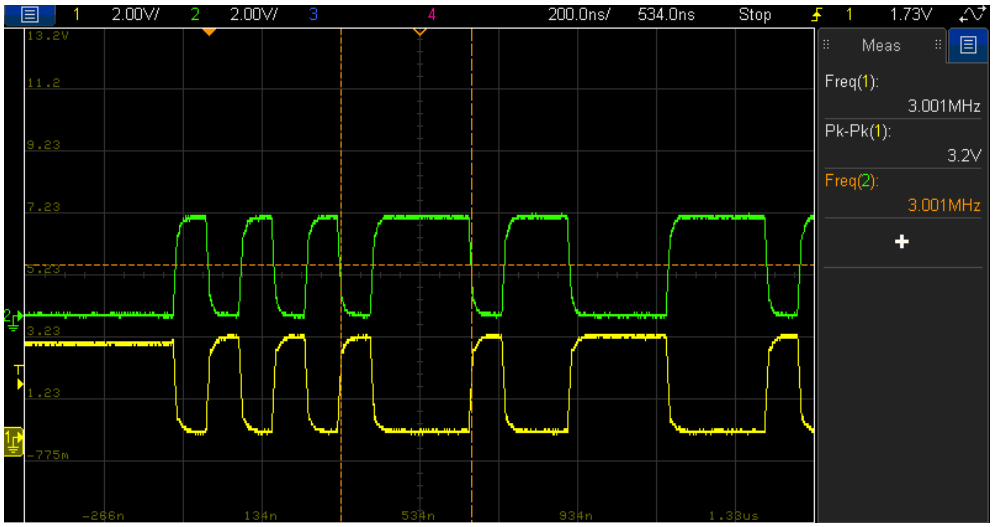
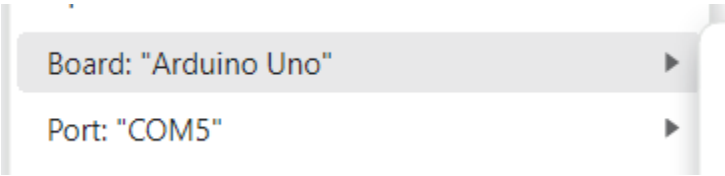


Figure 12: Shows the USB D+ and D- lines during communication. The signals toggle at 3MHz as expected and showcase a 3.3v peak to peak swing as expected.

- This verifies the USB section of the board is working properly and that the USB interface is alive and communicating

6.3 Bootloader Flashing & Uploading Blink Sketch



- After burning the arduino and connecting the USB, I can detect the arduino on my computer, proving full Arduino compatibility. This is the defining test of the board's functionality.
- I then uploaded a simple sketch on Arduino which is designed to blink pin 13 at 5 Hz.

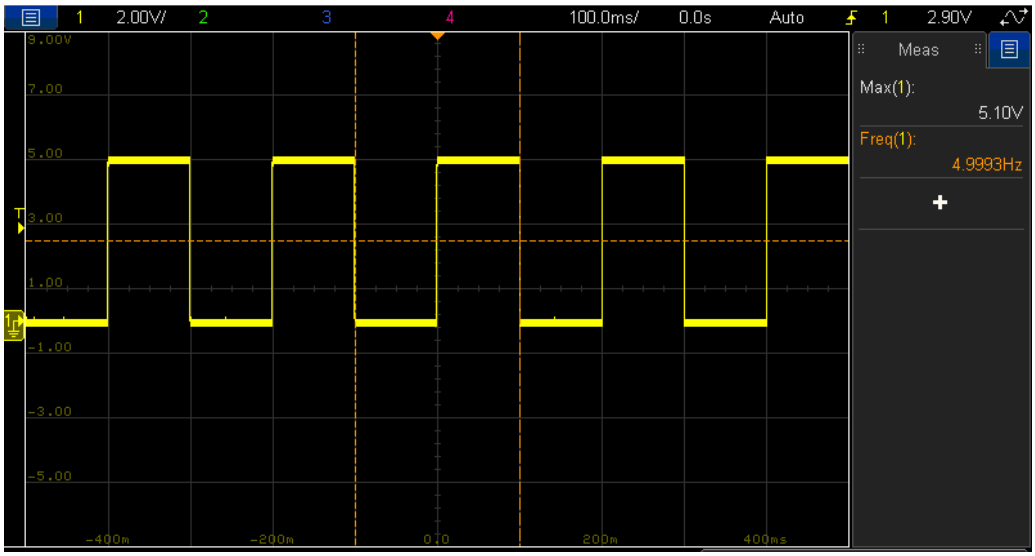


Figure 13: Shows the output of pin 13 after uploading the blink sketch.

- This shows that the ATmega328P is not only bootloaded but also fully programmable. It closes the loop between schematic, layout, assembly, USB communication, and software upload.

6.4 DAC Output

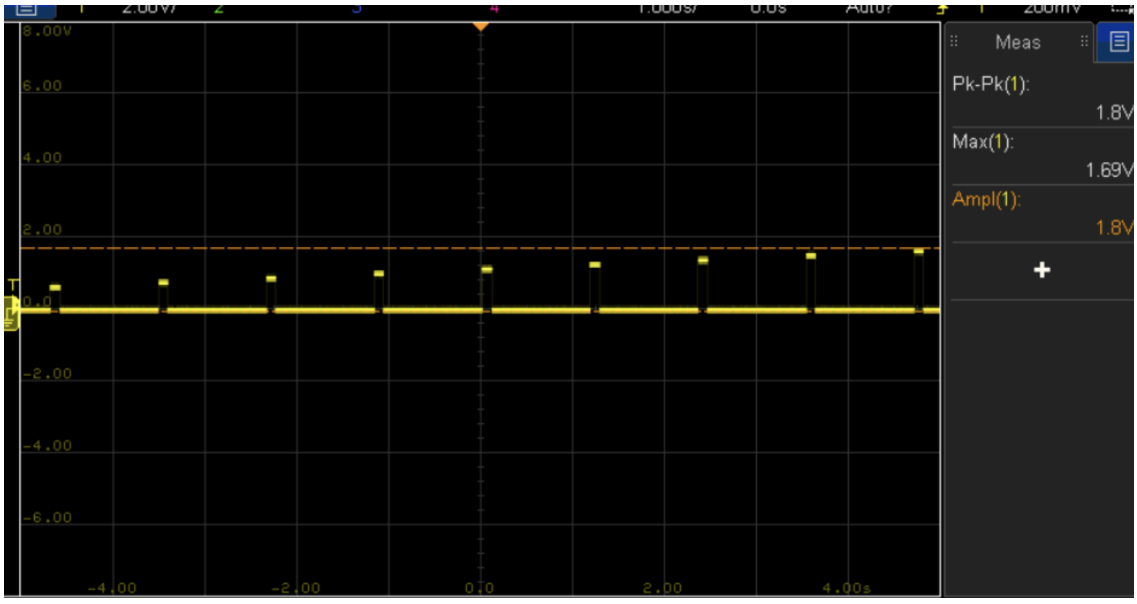


Figure 14: Shows DAC output

- Confirms the DAC is producing a series of short voltage pulses, with each pulse corresponding to a commanded step increase in current through the MOSFET and sense resistor

6.5 Smart Indicators

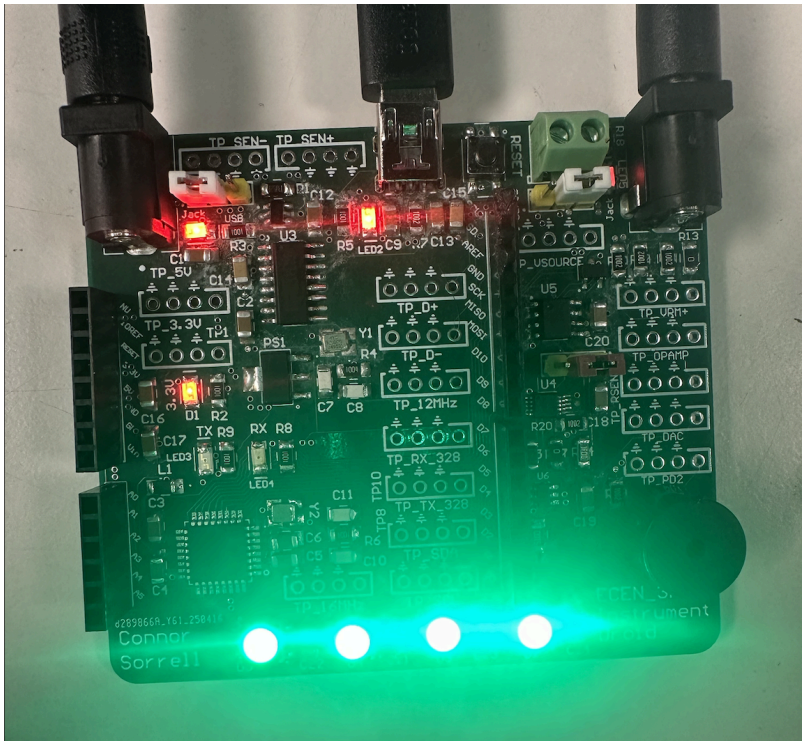


Figure 15: Shows daisy-chained LED's illuminating green to represent a flat & constant R_thevenin

- Additionally, the buzzer on the board plays a "Dun dun dun" sound when the resistance is dropping, and once the resistance is flat and leveled off, it plays a mario tune. (Shown to TA)

- LEDs light up when current is flowing

- Buzzer gives audible pulse at each DAC activation

- The Thevenin characterizer and Indicator subsystems worked together flawlessly. The droid board manages the current ramps, measures voltage drop, computes resistance, and then alerts the user via LED & Buzzer

7. Thevenin Characterization Results

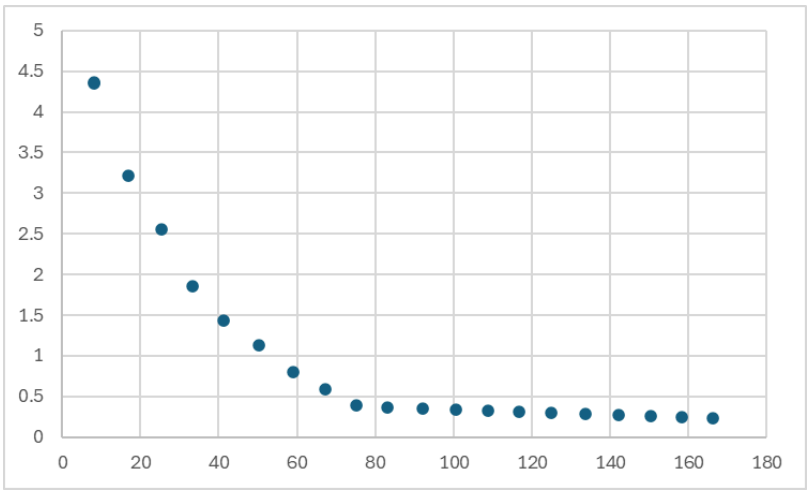
Some examples of voltage sources tested:

- 5V Wall Adapter
- 9v Wall Adapter
- 12V Wall Adapter

Data was logged using Arduino Serial Plotter and exported to Excel.

5V Adapter:

```
1, 8.208, 5.2475, 5.2117, 4.3573
1, 8.209, 5.2478, 5.2121, 4.3512
2, 16.951, 5.2483, 5.1937, 3.2214
3, 25.419, 5.2486, 5.1835, 2.5597
4, 33.276, 5.2493, 5.1877, 1.8522
5, 41.327, 5.2497, 5.1901, 1.4407
6, 50.146, 5.2501, 5.1931, 1.1366
7, 58.930, 5.2503, 5.2030, 0.8025
8, 67.289, 5.2510, 5.2112, 0.5914
9, 75.125, 5.2513, 5.2216, 0.3963
10, 83.190, 5.2523, 5.2215, 0.3702
11, 92.017, 5.2524, 5.2204, 0.3480
12, 100.670, 5.2527, 5.2190, 0.3348
13, 108.883, 5.2531, 5.2179, 0.3226
14, 116.816, 5.2536, 5.2178, 0.3065
15, 124.934, 5.2540, 5.2168, 0.2973
16, 133.704, 5.2534, 5.2149, 0.2876
17, 142.278, 5.2535, 5.2145, 0.2746
18, 150.501, 5.2541, 5.2147, 0.2618
19, 158.383, 5.2539, 5.2152, 0.2445
20, 166.334, 5.2541, 5.2148, 0.2366
done
```

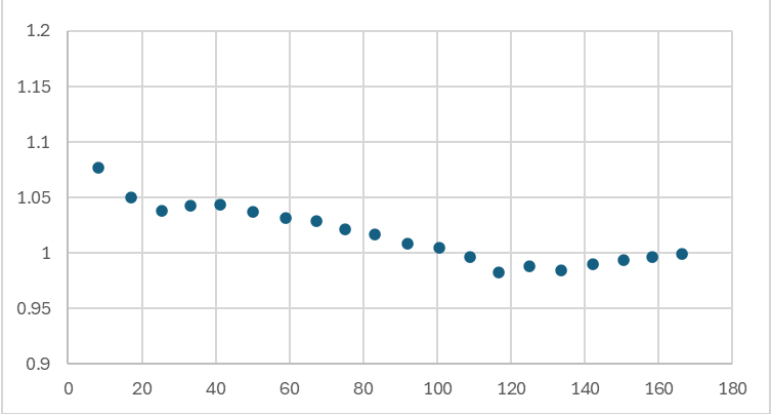


Thevenin Resistance: 0.2366 Ω

- From the data, we can make some observations. At low currents, (<20mA), the thevenin resistance is ~4 Ohms, and as current increases, the thevenin resistance sharply drops and flattens out around ~80mA. This behavior shows that the adapter has a relatively low output impedance once it is sourcing steady current, but is less effective at maintaining steady voltage under tiny loads
- In the context of the smart instrument droid, the LED's display a flashing red during the non-linear behavior of the thevenin resistance. Once the resistance value flattens out, the LED's display a constant green light.

9V Adapter:

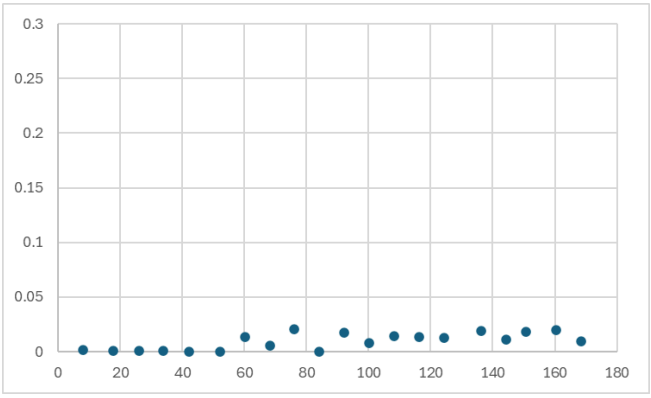
```
1, 8.212, 9.3353, 9.3269, 1.0280
2, 16.946, 9.3355, 9.3179, 1.0343
3, 25.419, 9.3354, 9.3094, 1.0248
4, 33.281, 9.3355, 9.3012, 1.0317
5, 41.329, 9.3355, 9.2930, 1.0298
6, 50.144, 9.3357, 9.2845, 1.0206
7, 58.927, 9.3357, 9.2756, 1.0195
8, 67.291, 9.3358, 9.2672, 1.0188
9, 75.136, 9.3358, 9.2597, 1.0130
10, 83.193, 9.3359, 9.2520, 1.0084
11, 92.013, 9.3360, 9.2433, 1.0080
12, 100.673, 9.3360, 9.2351, 1.0025
13, 108.899, 9.3361, 9.2273, 0.9991
14, 116.834, 9.3361, 9.2219, 0.9775
15, 124.939, 9.3363, 9.2139, 0.9796
16, 133.713, 9.3363, 9.2047, 0.9844
17, 142.283, 9.3364, 9.1961, 0.9861
18, 150.521, 9.3365, 9.1872, 0.9922
19, 158.391, 9.3366, 9.1793, 0.9929
20, 166.334, 9.3364, 9.1710, 0.9941
done
```



Thevenin Resistance: 0.9941 Ω

12V Adapter:

```
1, 8.013, 11.7866, 11.7866, 0.0017
2, 17.675, 11.7866, 11.7866, 0.0008
3, 25.962, 11.7866, 11.7866, 0.0005
4, 33.941, 11.7866, 11.7866, 0.0004
5, 42.254, 11.7866, 11.7866, 0.0003
6, 52.113, 11.7866, 11.7866, 0.0003
7, 60.138, 11.7866, 11.7858, 0.0132
8, 68.150, 11.7866, 11.7862, 0.0059
9, 76.175, 11.7866, 11.7850, 0.0207
10, 84.188, 11.7866, 11.7866, 0.0002
11, 92.213, 11.7866, 11.7850, 0.0171
12, 100.225, 11.7866, 11.7858, 0.0079
13, 108.250, 11.7866, 11.7850, 0.0146
14, 116.263, 11.7866, 11.7850, 0.0136
15, 124.288, 11.7866, 11.7850, 0.0127
16, 136.180, 11.7866, 11.7840, 0.0188
17, 144.188, 11.7866, 11.7850, 0.0109
18, 150.637, 11.7866, 11.7838, 0.0183
19, 160.363, 11.7866, 11.7835, 0.0196
20, 168.387, 11.7866, 11.7850, 0.0094
done
```



Thevenin Resistance: 0.0094 Ω

The onboard measurement instrument system not only worked, but also was effective in delivering useful insight such as the internal thevenin resistance of the power source, all by just letting the droid sweep current and collect data. It is useful to know the thevenin equivalent and sourcing characteristics of different sources in order to predict how something will act under a load. Knowing a power supply's limits, stability and performance under stress is extremely important in building an efficient & reliable system.

8. Challenges & Debugging

This board was less forgiving than earlier projects due to the 4-layer stack, which meant that not everything would be debuggable. Because of this, I added an abundance of test points which drastically improved and quickened my troubleshooting. Because of the fine pitch parts and how small I designed my board to be, I had several issues.

- A GND via ended up unintentionally solder-bridging to a nearby signal pin. This created a short on my board that took hours of probing and DMM use to find & fix
- Moving too quickly, I soldered the screw terminal block backwards.
- Initially, the board wouldn't show up in Arduino IDE. After finding a poor solder joint on the CH340 USB chip, I fixed it allowing for USB detection
- Initial soldering of Op Amp was incorrect which caused the MOSFET to fully turn on and dump too much current into 1 ohm sense resistor, burning it
- Because I minimized the board size, many components were crammed edge to edge. Any attempt to fix or replace one part meant desoldering multiple surrounding parts
- I forgot to label both the current range switch (1 Ω vs 10 Ω) and the input source selector (adapter vs terminal)

9. Conclusion

This was the most complex board I've designed, as well as the most rewarding. It brought together:

- Multilayer layout design
- Functionality of entire Arduino Uno microcontroller
- Analog and digital integration
- Firmware
- Real time testing
- Data capture and analysis

Takeaways:

- 4 layer board successfully built (signal/gnd/signal/gnd) provided good signal integrity, clean return paths, and relatively easy routing
- Principles of return vias; by placing a GND return via right next to each signal via, I ensured that return currents could follow their natural low impedance path
- Now understand the Thevenin source characterizer and how to pulse current through a load, measure voltage drop, and compute resistance in real time
- Firmware coordinated DAC stepping, ADC sampling, and resistance calculation, turning the board into its own instrument for collecting data
- Gained more practical experience searching for useful information within datasheets
- Designing, coding, and debugging this system deepened my understanding of how hardware, firmware, etc. all combine into one

Appendix: Arduino Code (LED_BuzzerPCB.ino)

```
// vrm characterizer board
#include <Wire.h>
#include <Adafruit_MCP4725.h>
#include <Adafruit_ADS1X15.h>
#include <Adafruit_NeoPixel.h>

#define LED_PIN 3
#define NUMPIXELS 5
#define BUZZER_PIN 2

Adafruit_NeoPixel pixels(NUMPIXELS, LED_PIN,  NEO_GRB + NEO_KHZ800);

Adafruit_ADS1115 ads;
Adafruit_MCP4725 dac;

float R_sense = 10.0; //current sensor
long itime_on_msec = 100; //on time for taking measurements
long itime_off_msec = itime_on_msec * 10; // time to cool off
int iCounter_off = 0; // counter for number of samples off
int iCounter_on = 0; // counter for number of samples on
float v_divider = 4700.0 / 14700.0;  //voltage divider on the VRM
float DAC_ADU_per_v = 4095.0 / 5.0; //conversion from volts to ADU
int V_DAC_ADU; // the value in ADU to output on the DAC
int I_DAC_ADU; // the current we want to output
float I_A = 0.0; //the current we want to output, in amps
long itime_stop_usec; // this is the stop time for each loop
float ADC_V_per_ADU = 0.125 * 1e-3; // the voltage of one bit on the gain of 1 scale
float V_VRM_on_v; // the value of the VRM voltage
float V_VRM_off_v; // the value of the VRM voltage
float I_sense_on_A; // the current through the sense resistor
float I_sense_off_A; // the current through the sense resistor
float I_max_A = 0.25; // max current to set for

int npts = 20; //number of points to measure
float I_step_A = I_max_A / npts; //step current change
float I_load_A; // the measured current load
float V_VRM_thevenin_v;
float V_VRM_loaded_v;
float R_thevenin;
int i;

void func_meas_off()
{
  dac.setVoltage(0, false); //sets the output current
  iCounter_off = 0; //starting the current counter
  V_VRM_off_v = 0.0; //initialize the VRM voltage averager
  I_sense_off_A = 0.0; // initialize the current averager
  itime_stop_usec = micros() + itime_off_msec * 1000; // stop time
  while (micros() <= itime_stop_usec)
  {
    V_VRM_off_v = ads.readADC_Differential_0_1() * ADC_V_per_ADU / v_divider + V_VRM_off_v;
    I_sense_off_A = ads.readADC_Differential_2_3() * ADC_V_per_ADU / R_sense + I_sense_off_A; iCounter_off++
  }
  V_VRM_off_v = V_VRM_off_v / iCounter_off;
  I_sense_off_A = I_sense_off_A / iCounter_off;
  // Serial.print(iCounter_off);Serial.print(", ");
  // Serial.print(I_sense_off_A * 1e3, 4); Serial.print(", ");
  // Serial.println(V_VRM_off_v, 4);
}

void func_meas_on()
{
//now turn on the current
  I_DAC_ADU = I_A * R_sense * DAC_ADU_per_v;
```

```
  I_DAC_ADU = I_A * R_sense * DAC_ADU_per_v;
  dac.setVoltage(I_DAC_ADU, false); //sets the output current
  iCounter_on = 0;
  V_VRM_on_v = 0.0; //initialize the VRM voltage averager
  I_sense_on_A = 0.00; // initialize the current averager
  itime_stop_usec = micros() + itime_on_msec * 1000; // stop time
  while (micros() <= itime_stop_usec)
  {
    V_VRM_on_v = ads.readADC_Differential_0_1() * ADC_V_per_ADU / v_divider + V_VRM_on_v;
    I_sense_on_A = ads.readADC_Differential_2_3() * ADC_V_per_ADU / R_sense + I_sense_on_A; iCounter_on++;
  }
  dac.setVoltage(0, false); //sets the output current to zero
  V_VRM_on_v = V_VRM_on_v / iCounter_on;
  I_sense_on_A = I_sense_on_A / iCounter_on;
  // Serial.print(iCounter_on);Serial.print(", ");
  // Serial.print(I_sense_on_A * 1e3, 4);Serial.print(", ");
  // Serial.println(V_VRM_on_v, 4);
}

int melody[] = {262, 196, 196, 220, 196, 0, 247, 262};

int noteDurations[] = {200, 200, 200, 200, 200, 200, 200, 200};

void playMario() {
  for (int thisNote = 0; thisNote < 8; thisNote++) {
    int noteDuration = noteDurations[thisNote];
    tone(BUZZER_PIN, melody[thisNote], noteDuration);
    delay(noteDuration * 1.3); // wait for note to finish + gap
    noTone(BUZZER_PIN);
  }
}

void playDunDunDun() {
  int dunNotes[] = {110, 110, 80}; // Lower for drama
  for (int i = 0; i < 3; i++) {
    tone(BUZZER_PIN, dunNotes[i], 300);
    for (int j = 0; j < NUMPIXELS; j++) pixels.setPixelColor(j, pixels.Color(255, 0, 0)); // RED
    pixels.show();
    delay(300);
    noTone(BUZZER_PIN);
    for (int j = 0; j < NUMPIXELS; j++) pixels.setPixelColor(j, pixels.Color(0, 0, 0)); // OFF
    pixels.show();
    delay(100);
  }
}


void setup()
{
  Serial.begin(2000000); dac.begin(0x60); // address is either 0x60, 0x61, 0x62,0x63, 0x64 or 0x65
  dac.setVoltage(0, false); //sets the output current to 0 initially
  // ads.setGain(GAIN_TWOTHIRDS); // 2/3x gain +/- 6.144V 1 bit = 3mV 0.1875mV (default)
  ads.setGain(GAIN_ONE); // 1x gain +/- 4.096V 1 bit = 2mV 0.125mV
  // ads.setGain(GAIN_TWO); // 2x gain +/- 2.048V 1 bit = 1mV 0.0625mV
  // ads.setGain(GAIN_FOUR); // 4x gain +/- 1.024V 1 bit = 0.5mV 0.03125mV
  // ads.setGain(GAIN_EIGHT); // 8x gain +/- 0.512V 1 bit = 0.25mV 0.015625mV
  // ads.setGain(GAIN_SIXTEEN); // 16x gain +/- 0.256V 1 bit = 0.125mV 0.0078125mV
  ads.begin(); // note- you can put the address of the ADS111 here if needed
  ads.setDataRate(RATE_ADS1115_860SPS);// sets the ADS1115 for higher speed

  pinMode(BUZZER_PIN, OUTPUT);
  //pixels.begin();
  for (int j = 0; j < NUMPIXELS; j++) pixels.setPixelColor(j, pixels.Color(0, 255, 0)); // GREEN
  //pixels.show();
}
```

```
128
129 void loop()
130 {
131   float R_prev = -1;
132   bool songPlayed = false;
133   bool dunPlayed = false;
134
135   for (i = 1; i <= npts; i++)
136   {
137     I_A = i * I_step_A;
138     dac.setVoltage(0, false); // disable DAC
139     func_meas_off();
140     func_meas_on();
141     dac.setVoltage(0, false);
142
143     I_load_A = I_sense_on_A - I_sense_off_A;
144     V_VRM_thevenin_v = V_VRM_off_v;
145     V_VRM_loaded_v = V_VRM_on_v;
146     R_thevenin = (V_VRM_thevenin_v - V_VRM_loaded_v) / I_load_A;
147
148     // --- ALERT 1: Thevenin < 1 or flat ---
149     if ((R_thevenin < 1.0 || abs(R_thevenin - R_prev) < 0.2) && !songPlayed) {
150       for (int j = 0; j < NUMPIXELS; j++) pixels.setPixelColor(j, pixels.Color(0, 255, 0)); // GREEN
151       pixels.show();
152       playMario();
153       songPlayed = true;
154     }
155
156     // --- ALERT 2: Sharp drop in R_thevenin ---
157     if (R_prev > 0 && (R_thevenin - R_prev) < -10.0 && !dunPlayed) {
158       playDunDunDun();
156     // --- ALERT 2: Sharp drop in R_thevenin ---
157     if (R_prev > 0 && (R_thevenin - R_prev) < -10.0 && !dunPlayed) {
158       playDunDunDun();
159       dunPlayed = true;
160     }
161
162     R_prev = R_thevenin;
163
164     // --- Output serial for debugging ---
165     Serial.print(i);
166     Serial.print(", ");
167     Serial.print(I_load_A * 1e3, 3);
168     Serial.print(", ");
169     Serial.print(V_VRM_thevenin_v, 4);
170     Serial.print(", ");
171     Serial.print(V_VRM_loaded_v, 4);
172     Serial.print(", ");
173     Serial.println(R_thevenin, 4);
174
175     if (V_VRM_loaded_v < 0.75 * V_VRM_thevenin_v) break;
176   }
177
178   // Clear LEDs after scan
179   for (int j = 0; j < NUMPIXELS; j++) pixels.setPixelColor(j, pixels.Color(0, 0, 0));
180   pixels.show();
181   noTone(BUZZER_PIN);
182   Serial.println("done");
183   delay(30000);
184 }
```