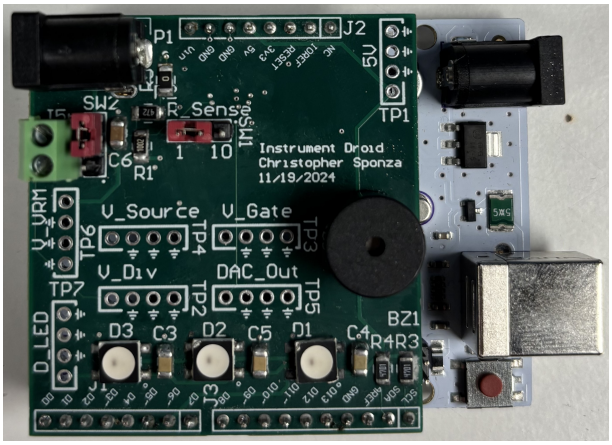
ECEN 3730 Board 4: Instrument Droid Shield

Christopher Sponza

December 4, 2024



1 Introduction

The purpose of the board was to create an Arduino shield that could take inputs from various voltage sources and characterize their Thevenin resistances and voltages at various currents.

Understanding the Thevenin equivalent circuit is a fundamental aspect of circuit design and analysis. By determining V_{th} and R_{th} engineers can gain better insight as to the response of a power source to varying loads, allowing for more accurate assessment of power stability and efficiency.

Although it was not necessary, this board was designed with 4-layers to serve as practice for working with multiple signal planes as well as mounting of parts on both sides of a PCB.

2 Plan of Record

2.1 "Back of Napkin" Sketch

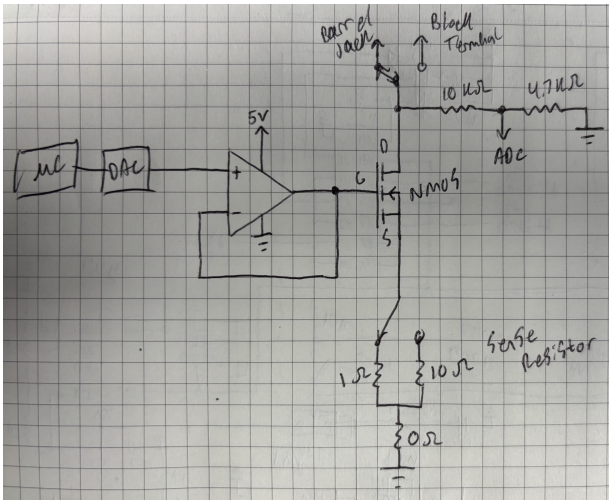


Figure 1: "Back of Napkin" Sketch

2.2 Description of the Circuit

Power is supplied via a barrel jack or block terminal to the drain of an NMOS. The voltage is divided with a $\frac{1}{3}$ divider using $10k\Omega$ and $5k\Omega$ resistors to meet the voltage requirements of the ADC. Using a digital signal from an Arduino the DAC sets the voltage at the gate of the NMOS to ensure a known current

through the sense resistor. The resolution of the measurement can be changed by selecting either the 1Ω or 10Ω sense resistor.

2.3 Engineering Specifications

The following were the specifications that were required to be met in order for the board to "work":

- Have the option to have power supplied by a barrel jack or stripped leads at a block terminal with a 3-header selection switch.
- Mount directly on an Arduino Uno via header pins.
- Be able to characterize voltage sources both with a 10Ω and 1Ω sense resistor.
- Provide test points for:
  - The voltage of the source at the barrel jack or block terminal
  - V_{out} of the DAC
  - Voltage of the voltage divider
  - Source and Gate voltage of the MOSFET
  - 5V supply to the DAC, ADC, and Op-Amp
- Provide indication of measurement status with smart LEDs and a piezo speaker

2.4 Bill of Materials

Part	Value	Quantity
ADS1115 (ADS)	-	1
MCP6002T (Op-Amp)	-	1
U_MCP4725 (DAC)	-	1
AO3400A (NMOS)	-	1
Resistor	0Ω	2
-	1Ω	1
-	10Ω	1
-	$4.7K\Omega$	3
-	$10K\Omega$	1
Capacitor	$1\mu F$	3
-	$22\mu F$	3
Smart LED	-	3
Piezo Speaker	-	1
Position Header Connector	6 Pin	1
-	8 Pin	1
-	10 Pin	1
10x Probe TP	-	7
Power Jack (Barrel)	-	1
Power Jack (Block Terminal)	-	1
3-Pin Header (Switch)	-	2

2.5 Datasheets

- **MCP6002T**: Op-Amp Datasheet
- **ADS1115**: ADS Datasheet
- **U_MCP4725**: DAC Datasheet
- **AO3400A**: NMOS Datasheet

2.6 Milestones

The following are the main design milestones to reach the final product in chronological order:

- Define the engineering specifications
- Complete schematic and PCB design in Altium
- Critical Design Review (CDR)
- Board Assembly and Bring-Up
- Testing and accumulation of results

2.7 Testing Plan

- Apply 9V to the board via the barrel jack and verify expected voltage at the drain of the MOSFET and voltage divider. Do the same for the block terminal.
- Mount the board on a commercial Arduino and verify the smart LEDs and Piezo speaker can be properly controlled.
- Upload code to control the voltage output of the DAC and perform verification measurement at the DAC output and the source of the MOSFET to ensure it is turning on properly.
- Perform Thevenin measurements as follows:
  - Using a max current of 250mA, increment the voltage output of the DAC over 20 divisions at a 10% duty cycle to prevent over-heating of the MOSFET.
  - Measure the voltage at the divider as fast as possible with the ADS and average the result with the DAC both on and off.
  - At the end of each power cycle of the MOSFET the voltage measured while the DAC is off will be V_{th}. This value will be used with the DAC on voltage to calculate the R_{th}.
- Repeat the measurement process for a 5V power supply.

2.8 Final Product

The following schematic and PCB were designed in Altium:

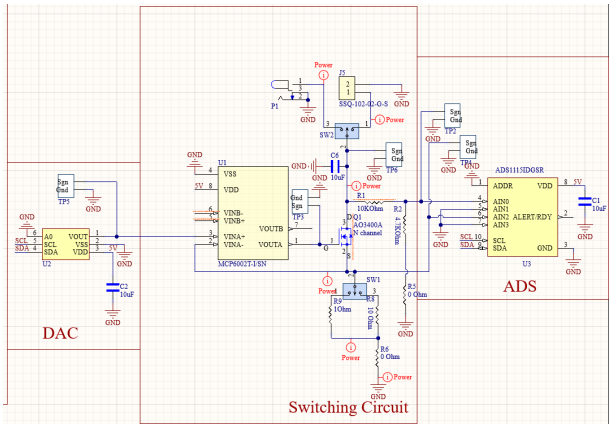


Figure 2: Data Collection Section of Schematic

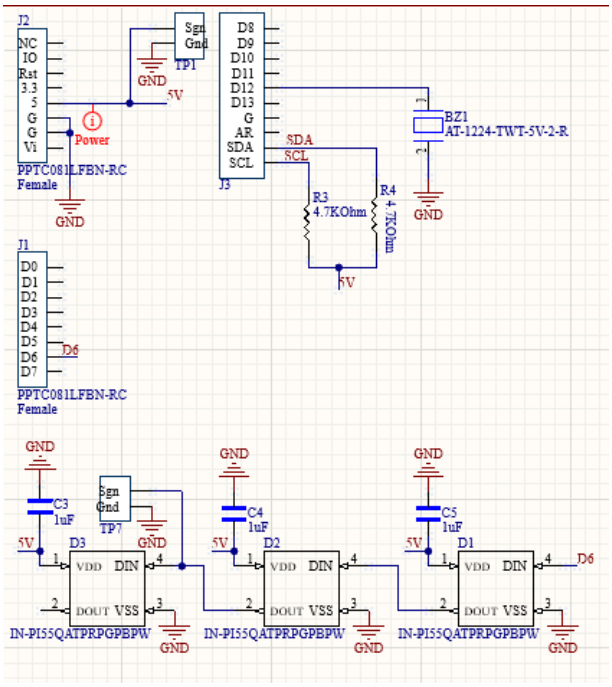


Figure 3: Smart LED and Header Pin Section of Schematic

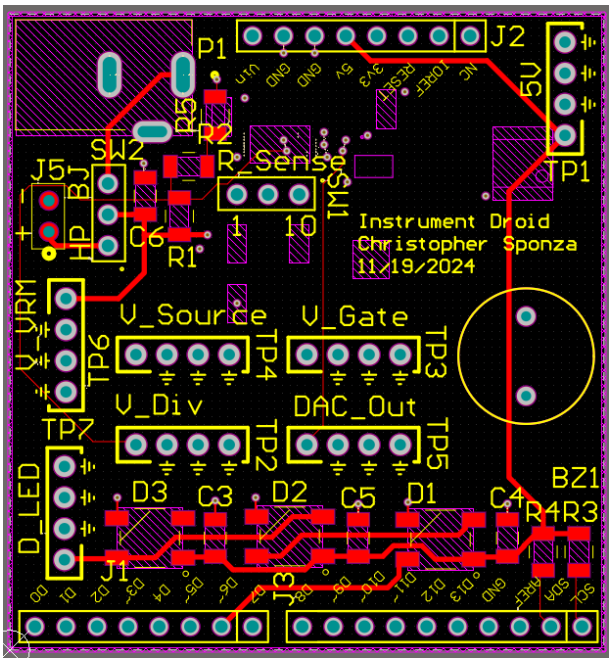


Figure 4: Board Designed in Altium (Top Signal Layer)

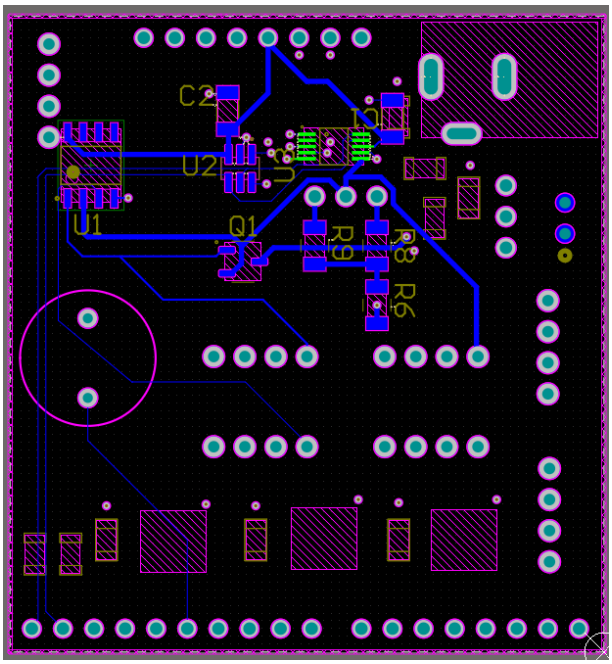


Figure 5: Board Designed in Altium (Bottom Signal Layer)

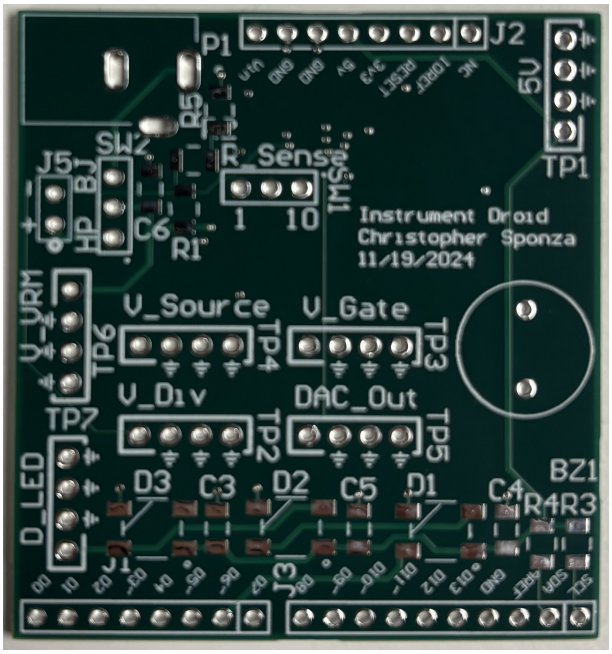


Figure 6: Board Received From the Manufacturer

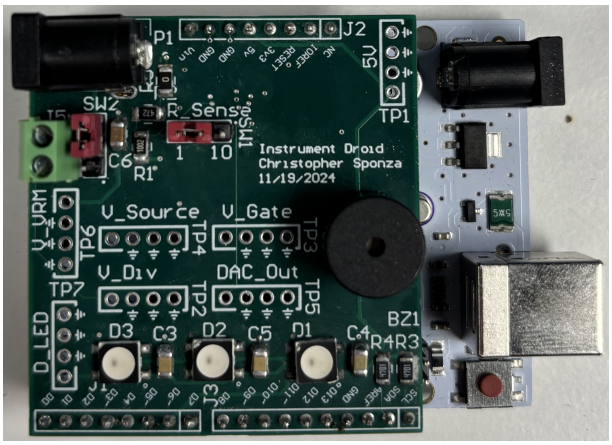


Figure 7: Constructed Board Mounted on Arduino Uno (Top)

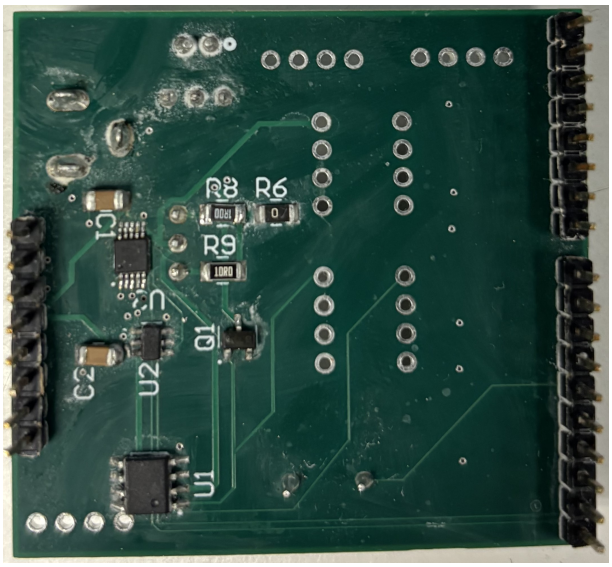


Figure 8: Constructed Board (Bottom)

3 Project Analysis and Measurement

3.1 Testing

3.1.1 Supply Voltage and Voltage Divider

I first supplied 9V to both the barrel jack and block terminal and verified expected voltages at the MOSFET drain and voltage divider.

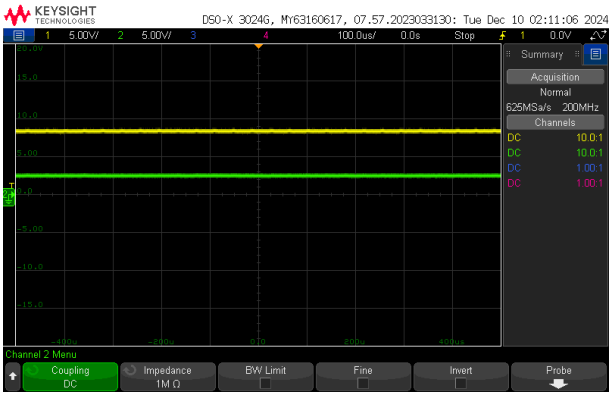


Figure 9: Oscilloscope Measurement at the drain of the MOSFET (Yellow) and Voltage Divider (Green) with 9V Supply at the Barrel Jack

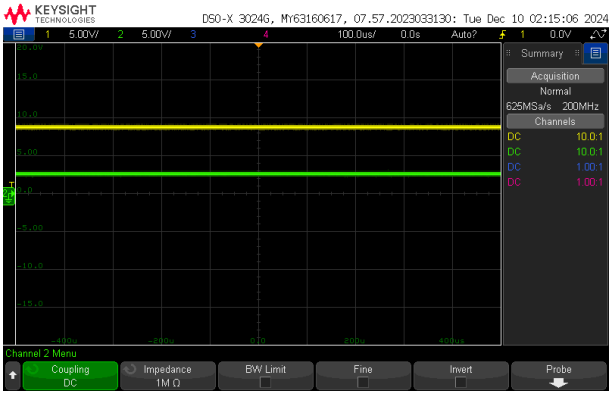


Figure 10: Oscilloscope Measurement at the drain of the MOSFET (Yellow) and Voltage Divider (Green) with 9V Supply at the Block Terminal

As can be seen in Figures 9 and 10 9V is properly received in both cases and there is roughly 3V at the voltage divider verifying the expected $\frac{1}{3}$ voltage division.

3.1.2 Smart LED Functionality

I next uploaded the code from Appendix A to perform the remainder of the testing including the control of the buzzer and smart LEDs, DAC output and ADC measurements, and Thevenin measurements.

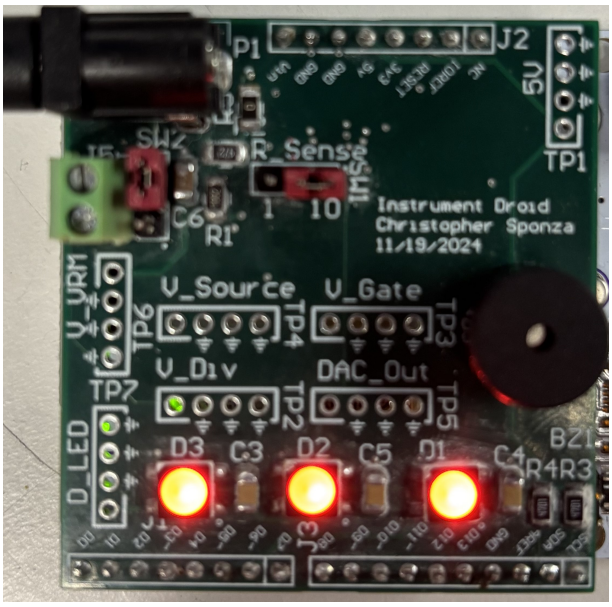


Figure 11: Smart LEDs at Beginning of Measurement Cycle

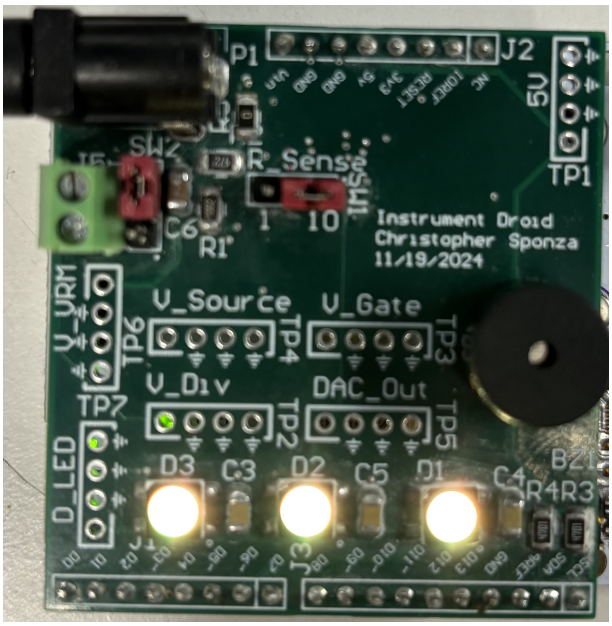


Figure 12: Smart LEDs in the Middle of Measurement Cycle

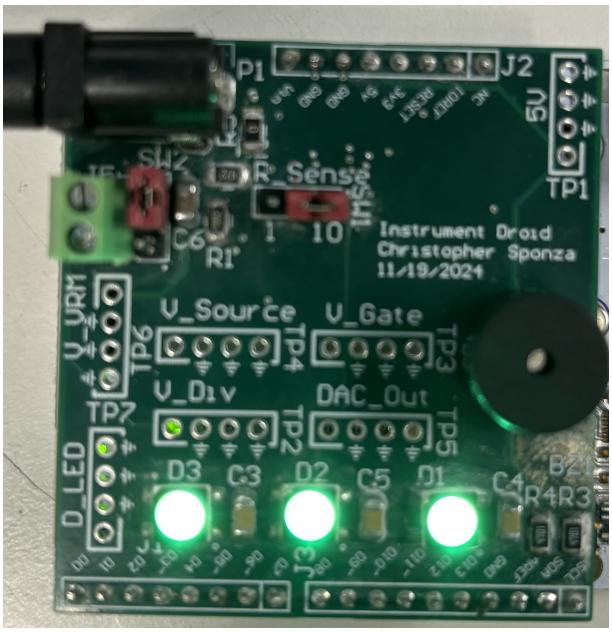


Figure 13: Smart LEDs at the end of the Measurement Cycle

Figures 11, 12, and 13 display the smart LED functionality through the beginning middle and end of the measurement cycle. Each time a measurement is taken the LEDs make an incremental change from red to green. At the completion of the measurement cycle the buzzer sounds as an indication.

3.1.3 DAC and ADC Verification

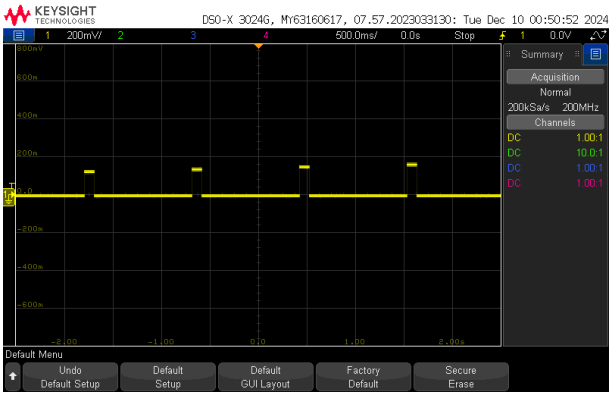


Figure 14: Output of the DAC Midway Through Measurement Cycle with 12.5mV Increments

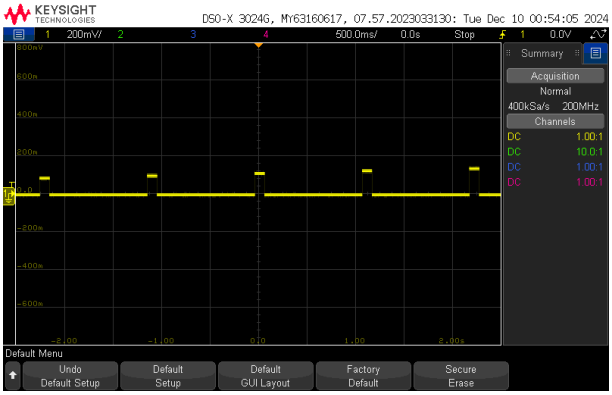


Figure 15: Voltage Measured at the Source of the NMOS

Figure 14 shows the output as read at the output of the DAC during the measurement cycle as verification of proper voltage control by the microcontroller. Figure 15 shows the voltage at the source of the NMOS transistor during the measurement cycle which verifies the proper switching of the MOSFET.

3.1.4 Current Measurement

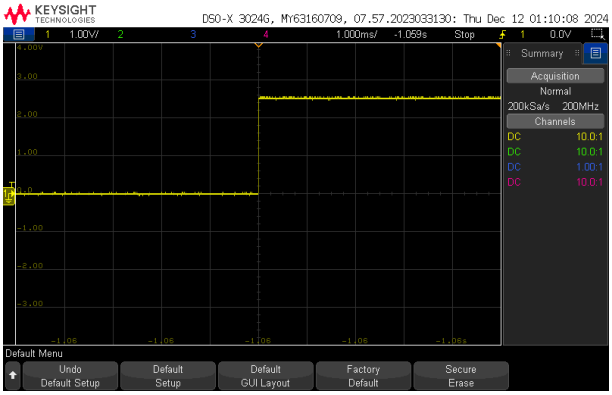


Figure 16: Voltage Measurement at the end of the Measurement Cycle Using a 9V Power Supply and 10Ω Sense Resistor

By measuring the voltage across the sense resistor during measurement, proper maximum current through the circuit can be verified. The voltage measurement from Figure 16 is the voltage during the final increment of the measurement step with a 9V power supply and a 10Ω sense resistor. The 2.5V measured together with the sense resistor corresponds to an expected 250mA.

3.1.5 Final Results

The following are the results on a scatter plot from measurements taken using the shield with an Arduino and the code from Appendix A:

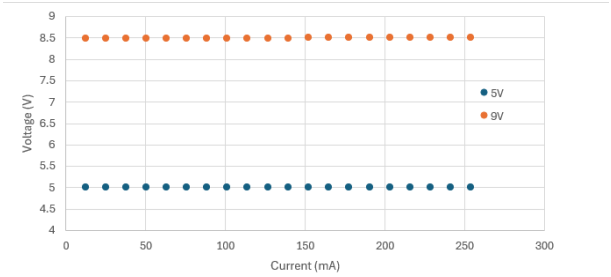


Figure 17: Thevenin Voltages from 5V (Blue) and 9V (Orange) Wall Wart Power Supplies and a 10Ω Sense Resistor

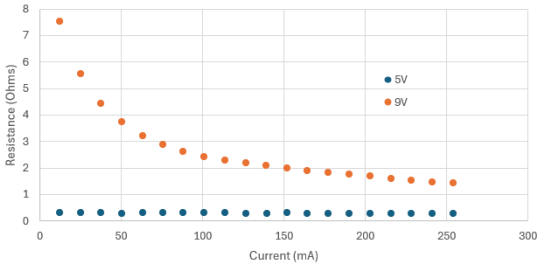


Figure 18: Thevenin Resistances from 5V (Blue) and 9V (Orange) Wall Wart Power Supplies and a 10Ω Sense Resistor

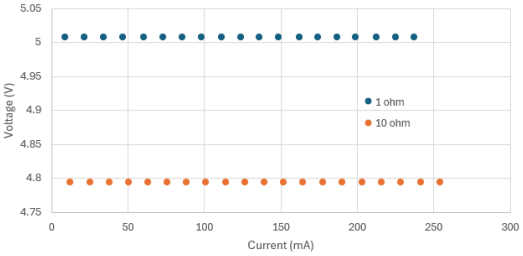


Figure 19: Thevenin Voltages from a Variable Power Supply at 5V Using 1Ω (Blue) and 10Ω (Orange) Sense Resistor

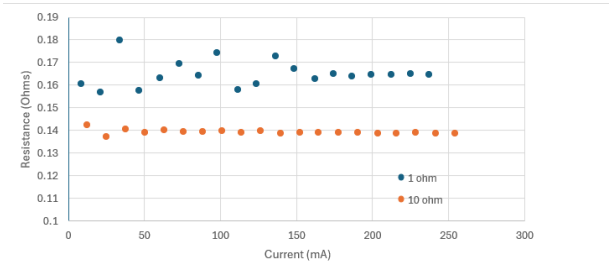


Figure 20: Thevenin Resistances from a Variable Power Supply at 5V Using 1Ω (Blue) and 10Ω (Orange) Sense Resistor

I first took measurements using the barrel jack using 5V and 9V wall warts. The Thevenin voltages and resistances output by the code are complied on the scatter plots from Figures 17 and 18 respectively. These measurements were used to verify proper characterization of power supplies at different voltages using the barrel jack and the 10Ω resistor. I next verified the operation of the block terminal and 1Ω sense resistor using a variable DC power supply. Figure 19 shows the Thevenin Voltage results from the block terminal using both sense resistors and Figure 20 shows the Thevenin resistance results. These measurements are also used to verify proper operation of the ADC.

3.2 What Worked

By the metrics provided in the engineering specifications my board "worked". Both the barrel jack and block terminal were able to supply power for measurements, the header pins were in the correct positions, and the DAC, ADC, and MOSFET were all operational. Mounting some of the parts on the bottom side of the board allowed for the size of the board to easily be constrained to the area of the Arduino header pins.

3.3 What Didn't Work

I blew out the 1Ω resistor during tested when I forgot to switch the shorting flag I could implement some indication of which sense resistor is in operation using an additional smart LED and digital pin. I also mistakenly used a 2-pin header footprint in place of a block terminal. The block terminal was still able to be added with some bending of pins but the 1-pin header for the power supply switch ended up very close to the terminal making it difficult to operate.

4 Final Remarks

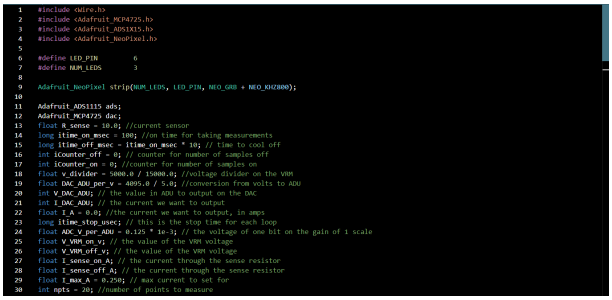
4.1 Things I Would Do Differently

- Place the switch for power supply selection further from the block terminal to make it easier to operate.
- Add some indication of the expected sense resistor.

4.2 Lessons Learned

- Pay close attention to the footprints used during design and their physical size in comparison to nearby parts.
- Adding additional indicators can help prevent the damage of power sensitive parts

Appendix A Code

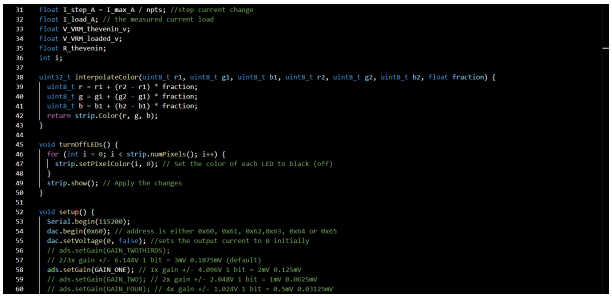


```
#include <Wire.h>
#include <Adafruit_MCP4725.h>
#include <Adafruit_ADS1X15.h>
#include <Adafruit_NeoPixel.h>

#define LED_PIN        6
#define NUM_LEDS       3

Adafruit_NeoPixel strip(NUM_LEDS, LED_PIN, NEO_GRB + NEO_KHZ800);

Adafruit_ADS1115 ads;
Adafruit_MCP4725 dac;
float R_sense = 10.0; //current sensor
long ltime_on_msec = 100; //on time for taking measurements
long ltime_off_msec = ltime_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 = 5000.0 / 15000.0; //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 ltime_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 = 0.250; // 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;

uint32_t interpolateColor(uint8_t r1, uint8_t g1, uint8_t b1, uint8_t r2, uint8_t g2, uint8_t b2, float fraction) {
  uint8_t r = r1 + (r2 - r1) * fraction;
  uint8_t g = g1 + (g2 - g1) * fraction;
  uint8_t b = b1 + (b2 - b1) * fraction;
  return strip.Color(r, g, b);
}

void turnOffLEDs() {
  for (int i = 0; i < strip.numPixels(); i++) {
    strip.setPixelColor(i, 0); // Set the color of each LED to black (off)
  }
  strip.show(); // Apply the changes
}

void setup() {
  Serial.begin(115200);
  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 0.1875mW (default)
  ads.setGain(GAIN_ONE); // 1x gain +/- 4.096V 1 bit = 2mV 0.125mW
  // ads.setGain(GAIN_TWO); // 2x gain +/- 2.048V 1 bit = 1mV 0.0625mW
  // ads.setGain(GAIN_FOUR); // 4x gain +/- 1.024V 1 bit = 0.5mV 0.03125mW
```

```
  // ads.setGain(GAIN_EIGHT); // 8x gain +/- 0.512V 1 bit = 0.25mV 0.0156250mV
  // ads.setGain(GAIN_SIXTEEN); // 16x gain +/- 0.256V 1 bit = 0.125mV 0.0078125mV
  ads.begin(); // note- you can put the address of the ADS1111 here if needed
  ads.setDataRate(RATE_ADS1115_860SPS); // sets the ADS1115 for higher speed

  //LED code
  strip.begin();           // Initialize the NeoPixel strip
  strip.setBrightness(50);
  strip.show();            // Ensure all LEDs are off initially
}

void loop() {
  uint8_t r1, g1, b1, r2, g2, b2;
  float fraction;
  for (i = 1; i <= npts; i++) {
    I_A = i * I_step_A;
    dac.setVoltage(0, false); //sets the output current
    func_meas_off();
    func_meas_on();
    dac.setVoltage(0, false); //sets the output current
    I_load_A = I_sense_on_A - I_sense_off_A; //load current
    V_VRM_thevenin_v = V_VRM_off_v;
    V_VRM_loaded_v = V_VRM_on_v;
    R_thevenin = (V_VRM_thevenin_v - V_VRM_loaded_v) / I_load_A;
    if (V_VRM_loaded_v < 0.75 * V_VRM_thevenin_v) i = npts; //stops the ramping
    Serial.print(i);
    Serial.print(", ");
    Serial.print(I_load_A * 1e3, 3);
    Serial.print(", ");
    Serial.print(V_VRM_thevenin_v, 4);
```

```
    Serial.print(", ");
    Serial.print(V_VRM_loaded_v, 4);
    Serial.print(", ");
    Serial.println(R_thevenin, 4);
    //LED Color Change code
    if (i < 10) {
      // Red to Yellow transition
      r1 = 255; g1 = 0; b1 = 0;  // Red
      r2 = 255; g2 = 255; b2 = 0; // Yellow
      fraction = i / 10.0;  // Fraction for this step
    }
    else {
      // Yellow to Green transition
      r1 = 255; g1 = 255; b1 = 0; // Yellow
      r2 = 0; g2 = 255; b2 = 0;   // Green
      fraction = (i - 10) / 10.0; // Fraction for this step
    }
    uint32_t color = interpolateColor(r1, g1, b1, r2, g2, b2, fraction);
    // Set all LEDs to this color
    for (int j = 0; j < NUM_LEDS; j++) {
      strip.setPixelColor(j, color);
    }
    strip.show();
  }
  Serial.println("done");
  tone(12, 500);
  turnOffLEDs();
  delay(600);
  noTone(12);
  delay(30000);
```

```
}
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;
}
void func_meas_on() {
  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;
}
```