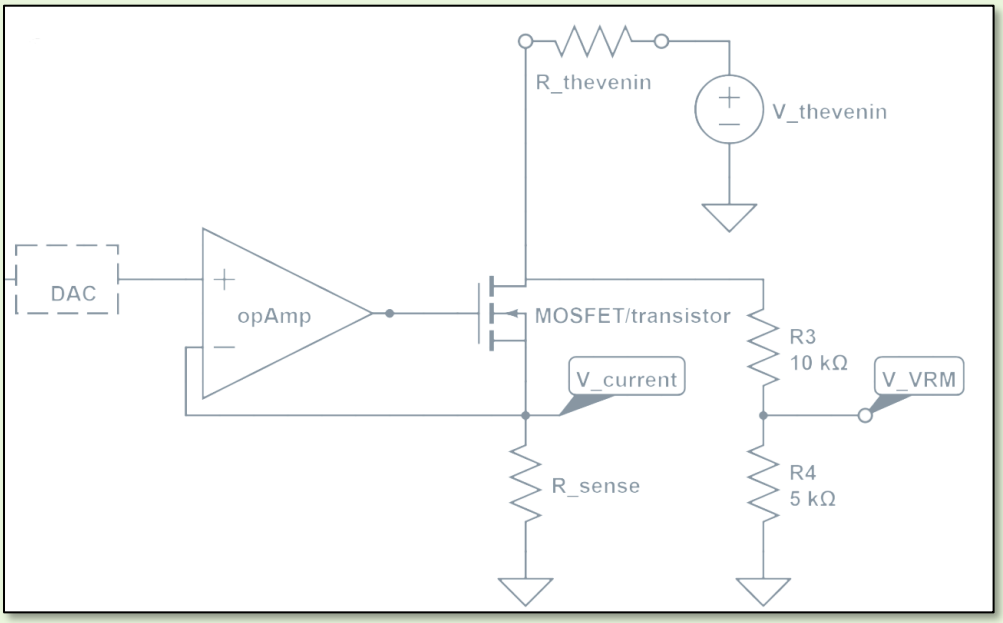
Practical PCB Design and Manufacture

Instrument Droid

BOARD 4
PCB Design



Practical PCB Design and Manufacture

Table of Contents

Practical PCB Design and Manufacture

☞ *INTRODUCTION*

In this project, I designed and constructed an instrument droid: a specialized, intelligent data acquisition system focused on a unique application, termed the "killer app." My primary goal was to automatically measure the Thevenin output resistance of a power source across various current loads. This task expanded on techniques previously applied in an earlier lab where we measured the Thevenin resistance of a function generator. This time, I assessed the output resistance over a broader range of current loads to evaluate its linearity or non-linearity and investigated the presence of any specific current clamps affecting the output. The project was split into two main components: the Arduino microcontroller (uC) section and the circuitry for Voltage Regulator Module (VRM) characterization. As a graduate student, I integrated these components into a single board. The device utilized principles from earlier experiments but was reconfigured to be controlled by a microcontroller, enhancing precision and usability.

The objective of this board is to construct an electronic load that draws current from a voltage source and measures the resultant voltage drop. This type of circuit is often referred to as a slammer circuit, which we constructed in the PDN lab. By analysing the known current load and the voltage drop across the power source, we can determine the Thevenin resistance, or the output resistance, of the source.

During the experiments with the function generator, I measured its Thevenin output resistance and found it to be 50 ohms. Typically, signal sources exhibit an output resistance ranging from 10 to 100 ohms. In contrast, power sources such as batteries or regulators usually have an output resistance between 0.1 ohms and 10 ohms.

☞ *PLAN OF RECORD*

The following was the plan of record for constructing my Board 4:

For the board features,

1. Power Input Selection: Flip the switch to select power only from the USB or the power jack.
2. Voltage regulation: A Low Dropout (LDO) regulator converts 5 volts to 3.3 volts.
3. Clock Configuration: The clock configuration includes two clock frequencies: 12 MHz for the CH340G and 16 MHz for the ATmega328.
4. Initial Programming: The ATmega328P is boot loaded using the ICSP pins.
5. Visual Indicators: The power status is shown by an LED next to the power input.
6. Improvements to Connectivity: Extra ground pins and decoupling/bypass capacitors at each integrated circuit are added to the standard Arduino headers.
7. Communication Support: Through the CH340G, enables USB UART communication.
8. Design and Safety Standards: A copper ground plane strengthens circuit stability, and test points and isolation switches increase usefulness and safety.
9. A four-layer board with ground layers in the middle and signal layers at the top and bottom layers.
10. The ground return plane is filled with copper.
11. Designated test points for the following signals:
    - VRM+ and VRM- for the Voltage Regulator Module (VRM)
    - Output of the operational amplifier (Op-amp)
    - Output of the Digital-to-Analog Converter (DAC)

Practical PCB Design and Manufacture

I have carried out the following implementations for appropriate testing and what it means to work:

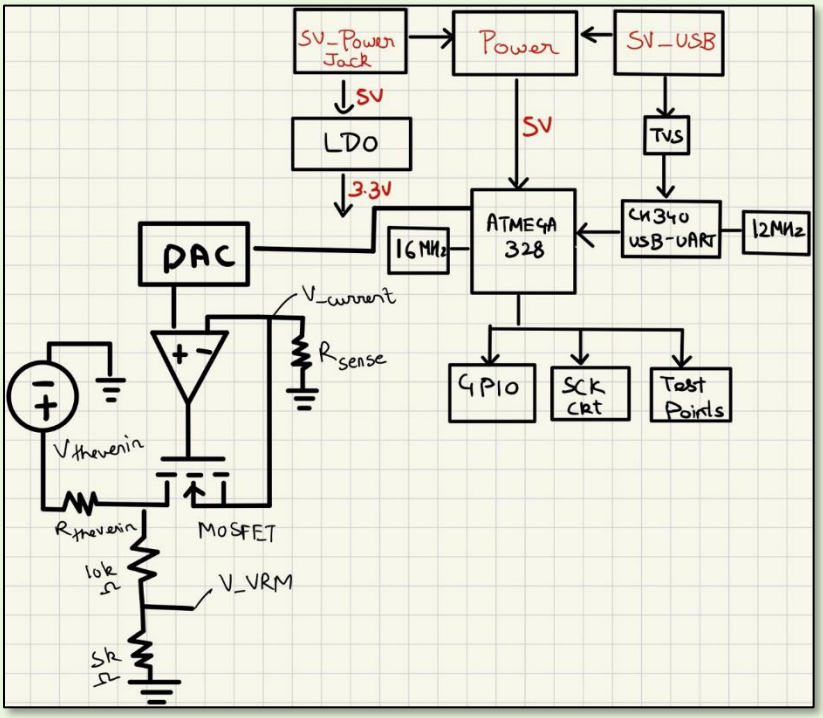
1. The board may now be powered and programmed via USB Mini support along with a screw terminal, power jack and another USB Mini for the instrument droid part.
2. To accept an external 5V AC to DC charger, a power plug was added.
3. To enable choosing between an external 5V power supply or a USB power source, a power selector switch was included.
4. To effectively convert the 5V input to 3.3V, a Low-Dropout Regulator (LDO) was employed.
5. To separate power to the CH340 chip and the microcontroller, a power isolator switch was incorporated.
6. Indicator LEDs were fitted for both 3.3V and 5V to give users a visual indication of the power condition.
7. To make it simple to reset the microcontroller in the event of any unpleasant behavior, a reset switch was thoughtfully positioned.
8. To boot load the microcontroller and provide an alternate programming technique, ICSP compatibility was included.
9. To guard against any harm from static electricity, a TVS (Transient electricity Suppressor) chip was included to the board.
10. Test points for important signals, such as USB connections, 3.3V input, and 5V input, were built.
11. Following the upload of the code, the Smart LEDs powered on as intended.
12. The buzzer plays various tones and functions properly.
13. The screw terminal is properly connected and there is a switch to change between the power jack / screw terminal.

For the risk management,

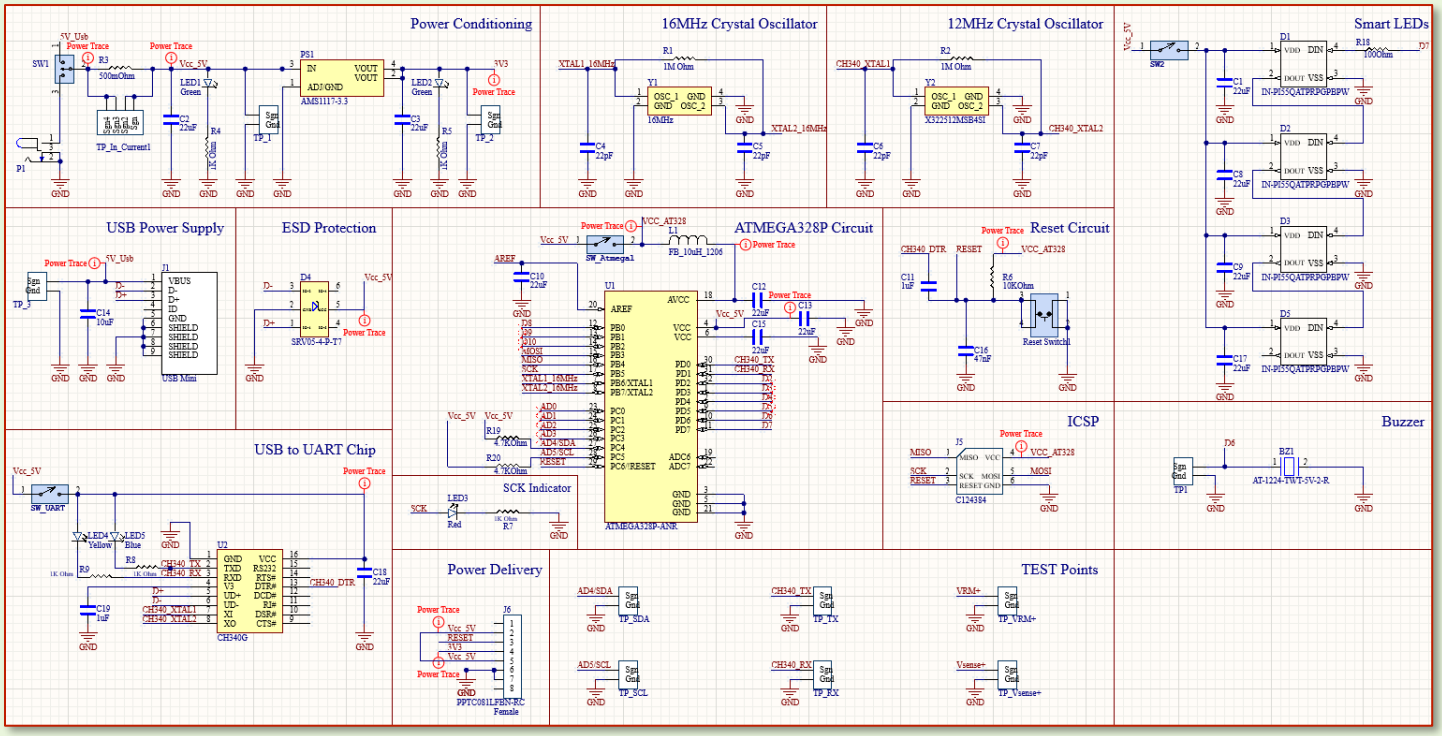
1. All the components on the board had names according to a naming system, and the board itself had labels.
2. Test points were added for validation and debugging purposes, as well as for the measurement of inrush current TX, RX, Vsense+, and VRM+.
3. To prevent voltage oscillations, a filter capacitor was used at the LDO's output. Additionally, all power lines were made 20 mills wide, and all signal lines were made 6 mills wide, as per the fabrication vendor's free offer.
4. At every step, switches are used for debugging and isolation.
5. Ground Vias are placed near every other via which connects to the 4th layer.
6. The board's power was managed by switches, and to provide precise timing, a 16 MHz clock was positioned next to the ATmega328P and a 12 MHz clock near the CH340G.
7. To minimize delays, the RX and TX line routing was planned to be around the same length. Additionally, LEDs were attached to the power stage and RX-TX output to monitor transmission activities, and indicators were employed for easy debugging. In addition, there was a QR code on the board that directed viewers to the portfolio website. The board's dimensions were restricted to 3.5 by 3.5 inches.

Practical PCB Design and Manufacture

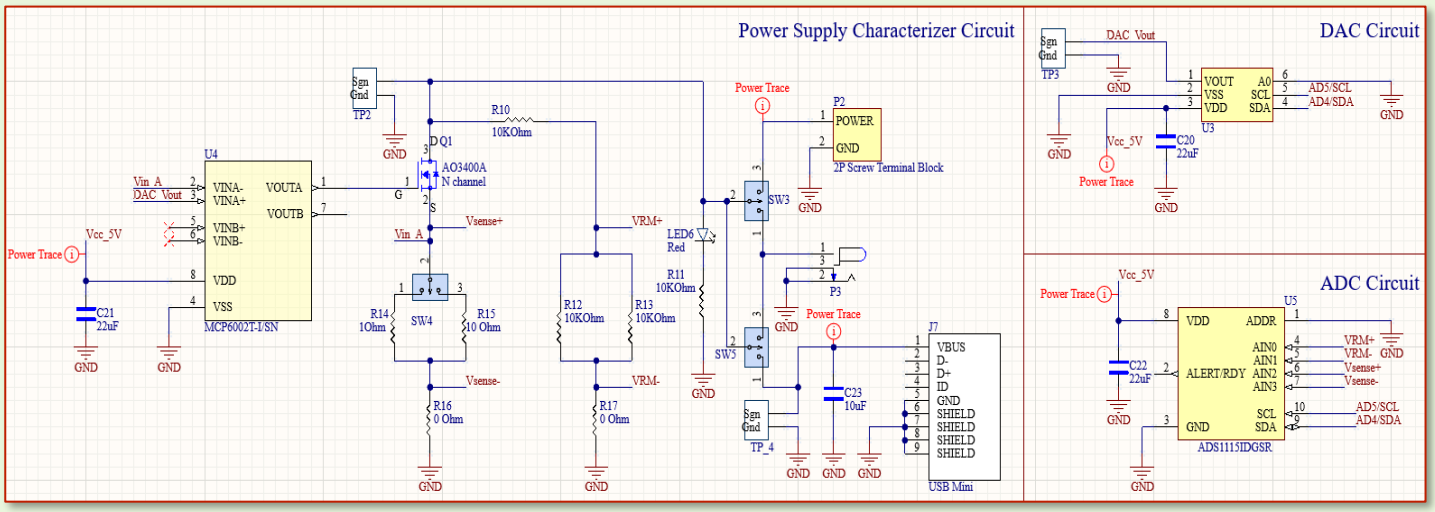
ROUGH SKETCH OF SKEMATIC



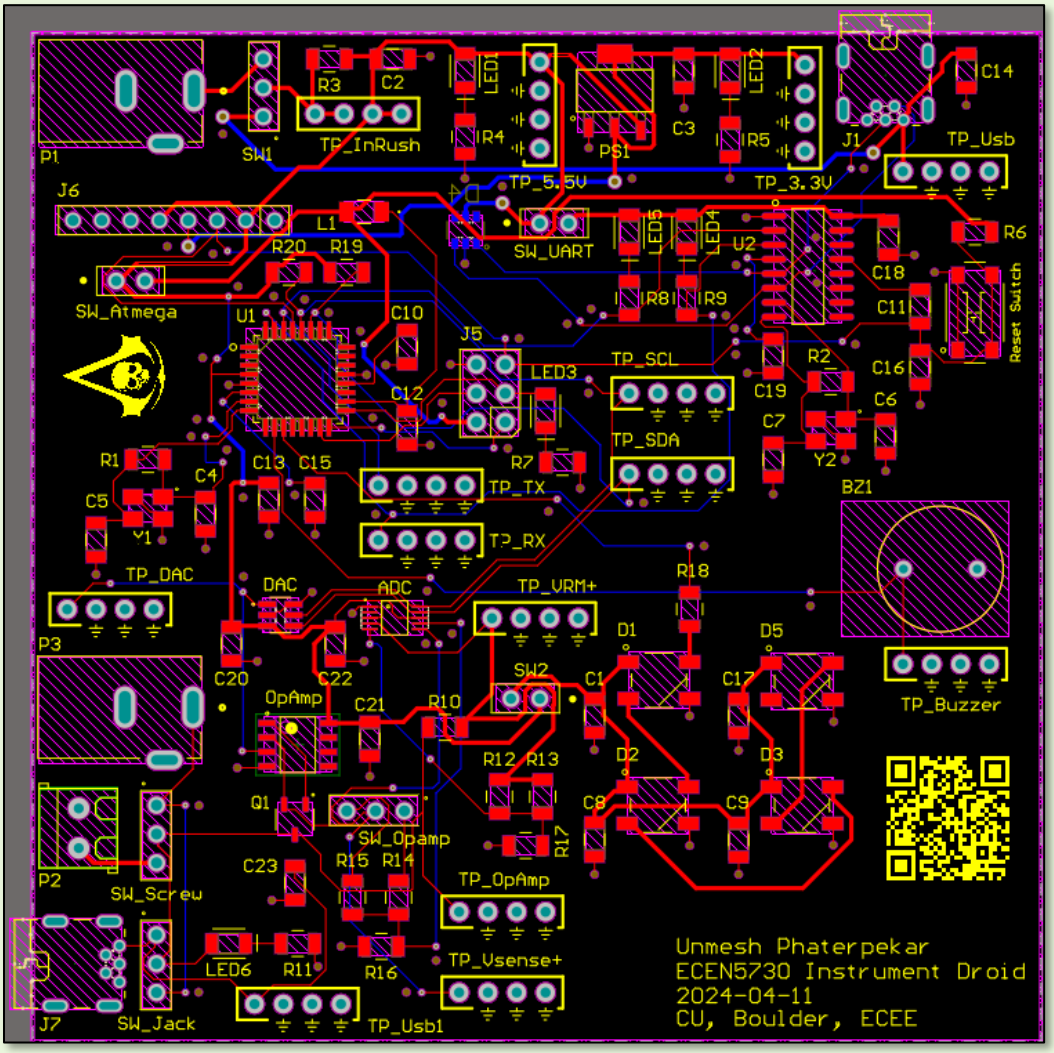
SKEMATIC CAPTURE ON ALTIUM



Practical PCB Design and Manufacture

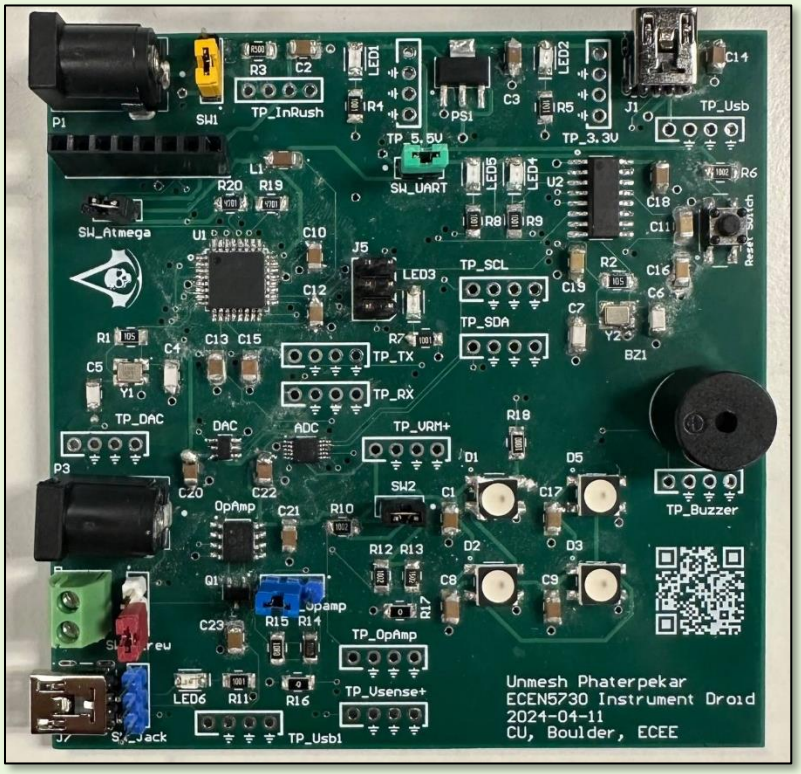


BOARD LAYOUT ON ALTIUM

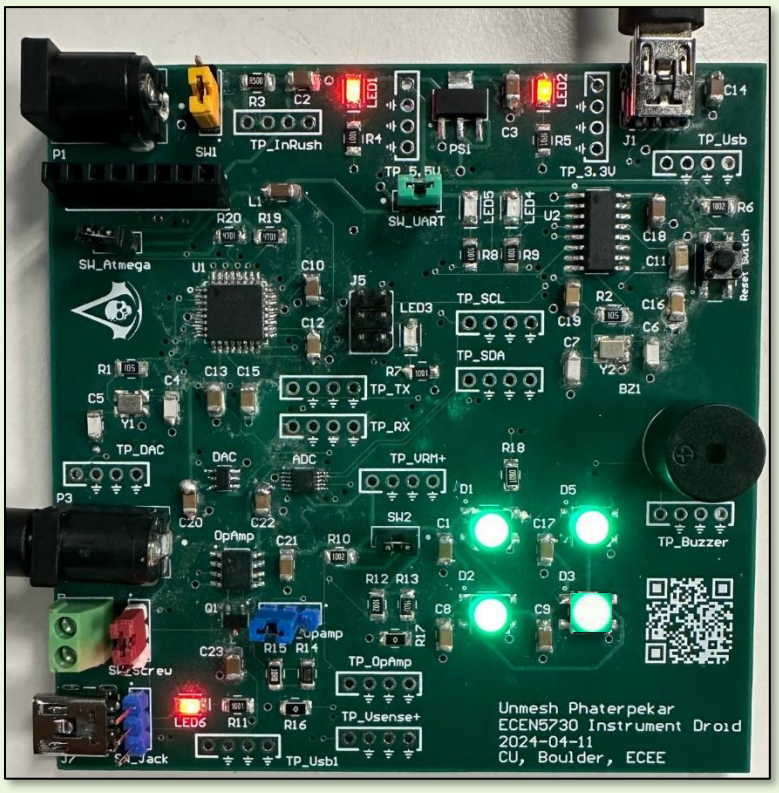


Practical PCB Design and Manufacture

☞ ASSEMBLED BOARD



☞ ASSEMBLED WORKING BOARD



Practical PCB Design and Manufacture

☞ *WHAT WORKED*

1. LED1 was illuminated, and a voltage of 5V was confirmed at the 5V rail test point, indicating that the board was operational.
2. LED2 lit up at 3.3V, verifying that the LDO successfully converted 5V to 3.3V.
3. Each I/O pin functioned as expected, capable of both lighting an external LED and reading a button's state that resets the circuit, thus confirming microcontroller communication.
4. The UART was validated through successful data transmission between the Arduino and a computer, ensuring data integrity.
5. A new program was successfully uploaded and executed on the board, confirming the correct installation and functionality of the bootloader.
6. The Smart LEDs were working as required.
7. The output from the DAC was accurately displayed on the oscilloscope.
8. The Op-Amp output was a bit noisy, but it was getting updated.
9. The buzzer functioned correctly and operated in conjunction with the Smart LEDs.
10. Seamless switching was achieved between the screw terminal, the power jack, and the USB mini supply for the Instrument Droid component.
11. The I2C connections were tested, and data transfer was verified using the I2C Scanner Code.

The following code was run:

```
 // --------------------------------------
// i2c_scanner
//
//
// This sketch tests the standard 7-bit addresses
// Devices with higher bit address might not be seen properly.
//

#include <Wire.h>


void setup()
{
  Wire.begin();

  Serial.begin(9600);
  while (!Serial);             // Leonardo: wait for serial monitor
  Serial.println("\nI2C Scanner");
}


void loop()
{
  byte error, address;
  int nDevices;
```

Practical PCB Design and Manufacture

```
  Serial.println("Scanning...");

  nDevices = 0;
  for(address = 1; address < 127; address++ )
  {
    // The i2c_scanner uses the return value of
    // the Write.endTransmisstion to see if
    // a device did acknowledge to the address.
    Wire.beginTransmission(address);
    error = Wire.endTransmission();

    if (error == 0)
    {
      Serial.print("I2C device found at address 0x");
      if (address<16)
        Serial.print("0");
      Serial.print(address,HEX);
      Serial.println("  !");

      nDevices++;
    }
    else if (error==4)
    {
      Serial.print("Unknown error at address 0x");
      if (address<16)
        Serial.print("0");
      Serial.println(address,HEX);
    }
  }
  if (nDevices == 0)
    Serial.println("No I2C devices found\n");
  else
    Serial.println("done\n");

  delay(5000);           // wait 5 seconds for next scan
}
```

Practical PCB Design and Manufacture

The output I got on the oscilloscope is as follows:

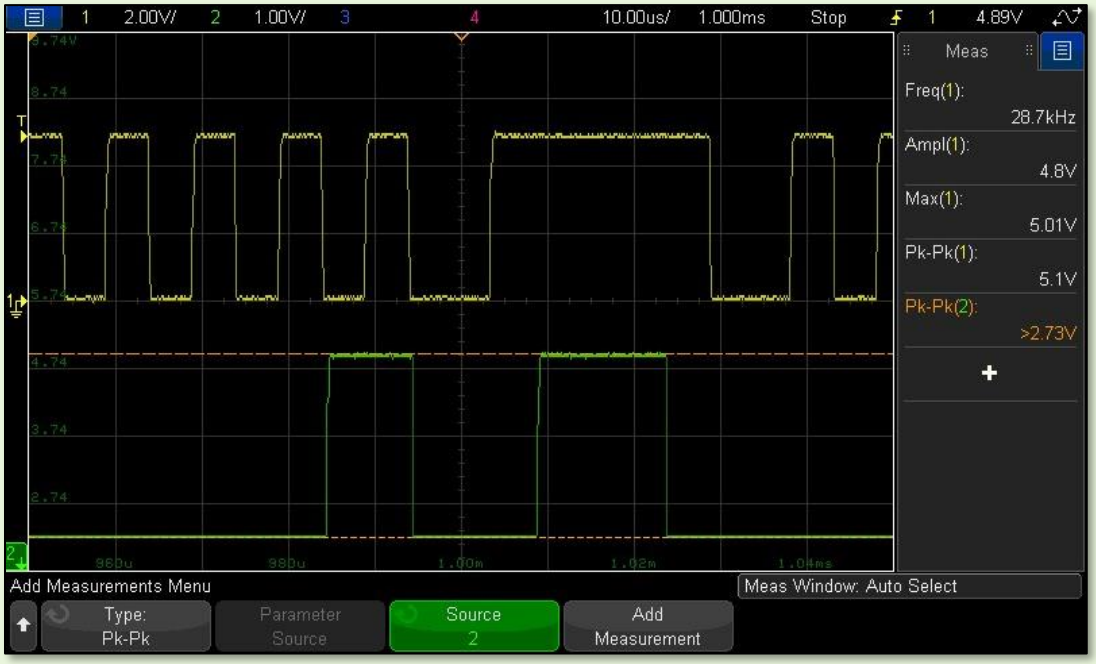


Fig 1: I2C SCL and SDA lines

☞ *BUZZER AND SMART LED INTEGRATION*

I integrated the Smart LEDs with the buzzer using the code given below: This code controls an LED strip and a buzzer using an Arduino-compatible board. It initializes the NeoPixel LED strip and sets its brightness. Special provisions are included for the Adafruit Trinket. In the loop function, it runs different light animations—color wipes, theater chases, and rainbows—on the LEDs, and the buzzer beeps at certain intervals to create an interactive light and sound display. Oscilloscope output:

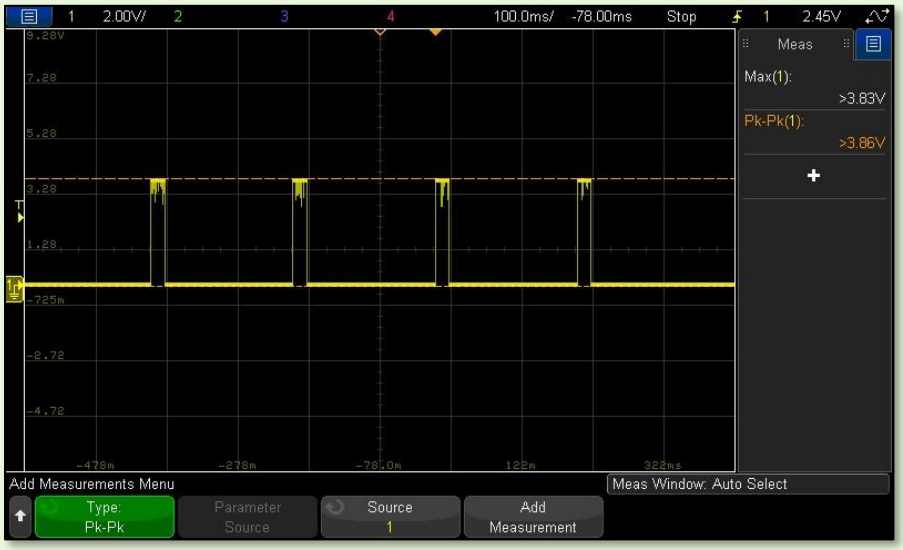


Fig 2: Buzzer with Smart LEDs

Practical PCB Design and Manufacture

```
#include <Adafruit_NeoPixel.h>

#ifdef __AVR__
 #include <avr/power.h> // Required for 16 MHz Adafruit Trinket
#endif

#define LED_PIN    7
#define LED_COUNT 60
#define BUZZER_PIN 6

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

void setup()
{
  // These lines are specifically to support the Adafruit Trinket 5V 16 MHz.
  // Any other board, you can remove this part (but no harm leaving it):
#if defined(__AVR_ATtiny85__) && (F_CPU == 16000000)
  clock_prescale_set(clock_div_1);
#endif
  // END of Trinket-specific code.

  strip.begin();           // INITIALIZE NeoPixel strip object (REQUIRED)
  strip.show();            // Turn OFF all pixels ASAP
  strip.setBrightness(50); // Set BRIGHTNESS to about 1/5 (max = 255)
  pinMode(BUZZER_PIN, OUTPUT); // Set the buzzer pin as output
}

void loop()
{
  // Color wipe animations
  colorWipe(strip.Color(255,   0,   0), 50); // Red
  colorWipe(strip.Color(  0, 255,   0), 50); // Green
  colorWipe(strip.Color(  0,   0, 255), 50); // Blue

  // Theater chase animations
  theaterChase(strip.Color(127, 127, 127), 50); // White, half brightness
  theaterChase(strip.Color(127,   0,   0), 50); // Red, half brightness
  theaterChase(strip.Color(  0,   0, 127), 50); // Blue, half brightness

  // Rainbow animations
  rainbow(10);             // Flowing rainbow cycle along the whole strip
  theaterChaseRainbow(50); // Rainbow-enhanced theaterChase variant
}

// Fill strip pixels one after another with a color. Strip is NOT cleared
```

Practical PCB Design and Manufacture

```
// first; anything there will be covered pixel by pixel. Pass in color
// (as a single 'packed' 32-bit value, which you can get by calling
// strip.Color(red, green, blue) as shown in the loop() function above),
// and a delay time (in milliseconds) between pixels.
void colorWipe(uint32_t color, int wait)
{
  for(int i=0; i<strip.numPixels(); i++)
  { // For each pixel in strip...
    strip.setPixelColor(i, color);         //  Set pixel's color (in RAM)
    strip.show();                          //  Update strip to match
    delay(wait);                           //  Pause for a moment
    if (i % 10 == 0)
    {
      beep(10, 250); // Beep for every 10th pixel
    }
  }
}

void theaterChase(uint32_t color, int wait)
{
  for(int a=0; a<10; a++)
  {  // Repeat 10 times...
    for(int b=0; b<3; b++)
    { //  'b' counts from 0 to 2...
      strip.clear();         //   Set all pixels in RAM to 0 (off)
      // 'c' counts up from 'b' to end of strip in steps of 3...
      for(int c=b; c<strip.numPixels(); c += 3)
      {
        strip.setPixelColor(c, color); // Set pixel 'c' to value 'color'
        if (c % 15 == 0)
        {
          beep(20, 180); // Beep for every 15th pixel set
        }
      }
      strip.show(); // Update strip with new contents
      delay(wait);  // Pause for a moment
    }
  }
}

void rainbow(int wait)
{
  for(long firstPixelHue = 0; firstPixelHue < 5*65536; firstPixelHue += 256)
  {
    strip.rainbow(firstPixelHue);
```

Practical PCB Design and Manufacture

```
    strip.show(); // Update strip with new contents
    delay(wait);  // Pause for a moment
  }
}

void theaterChaseRainbow(int wait)
{
  int firstPixelHue = 0;
  for(int a=0; a<30; a++)
  {
    for(int b=0; b<3; b++)
    {
      strip.clear();
      for(int c=b; c<strip.numPixels(); c += 3)
      {
        int      hue   = firstPixelHue + c * 65536L / strip.numPixels();
        uint32_t color = strip.gamma32(strip.ColorHSV(hue));
        strip.setPixelColor(c, color);
        if (c % 20 == 0) {
          beep(30, 150); // Beep for every 20th pixel in rainbow chase
        }
      }
      strip.show();
      delay(wait);
      firstPixelHue += 65536 / 90;
    }
  }
}

void beep(int duration, int pause)
{
    digitalWrite(BUZZER_PIN, HIGH);
    delay(duration);
    digitalWrite(BUZZER_PIN, LOW);
    delay(pause);
}
```

Practical PCB Design and Manufacture

☞ *SCOPE ANALYSIS*

The VRM Instrument Droid Code which is run on Board 4 is given below:

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

Adafruit_ADS1115 ads;
Adafruit_MCP4725 dac;

float R_sense = 10; //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 = 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 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
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 setup()
{
  Serial.begin(115200);
  dac.begin(0x62); // address is either 0x60, 0x61, 0x62,0x63, 0x64 or 0x65
  dac.setVoltage(0, false); //sets the output current to 0 initially
```

Practical PCB Design and Manufacture

```
  // 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(0x48); // note- you can put the address of the ADS111 here if needed
  ads.setDataRate(RATE_ADS1115_860SPS);// sets the ADS1115 for higher speed
}

void loop()
{
  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 < 0.25 * 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);
  }
  Serial.println("done");
  // 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
```

Practical PCB Design and Manufacture

```
  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;
  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);
}
```

Practical PCB Design and Manufacture

The operational amplifier (op-amp) is a versatile electronic component used in circuits for amplifying voltage signals. It boosts the difference in voltage between its two inputs and outputs the amplified signal. Op-amps are fundamental in applications like signal conditioning, filtering, and in building mathematical circuits that perform operations such as addition, integration, and differentiation. So, here we are using the op-amp in our circuit for boosting the voltage and following is the scope outcome of the op-amp.

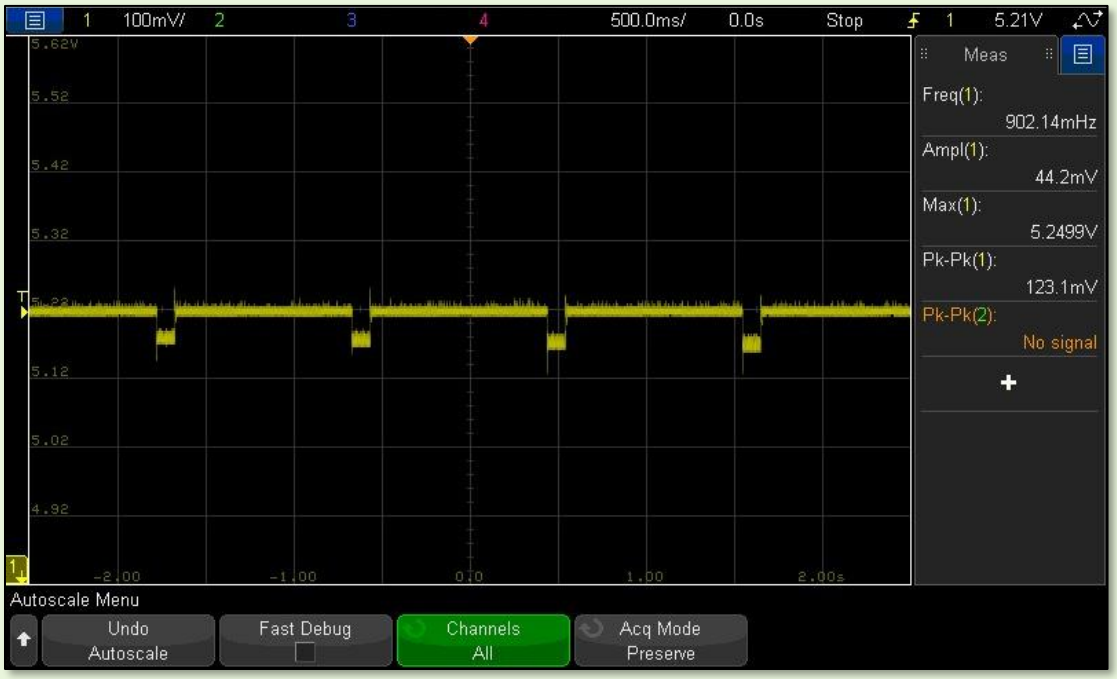


Fig 3: OP-AMP Output

A Digital-to-Analog Converter (DAC) is an electronic device that converts digital data (typically binary) into an analog signal. The output can be a continuous voltage or current signal that corresponds to the digital value. DACs are essential in applications where digital signals are needed to interact with real-world phenomena, such as in audio amplifiers, video encoders, and control systems in various devices.

We are using the DAC here to convert the analog values coming when the function generator is connected to the screw terminal.

We can see that the frequency of the DAC waveform is around 900mHz and the Peak – to – Peak value is around 1.5V.

Practical PCB Design and Manufacture

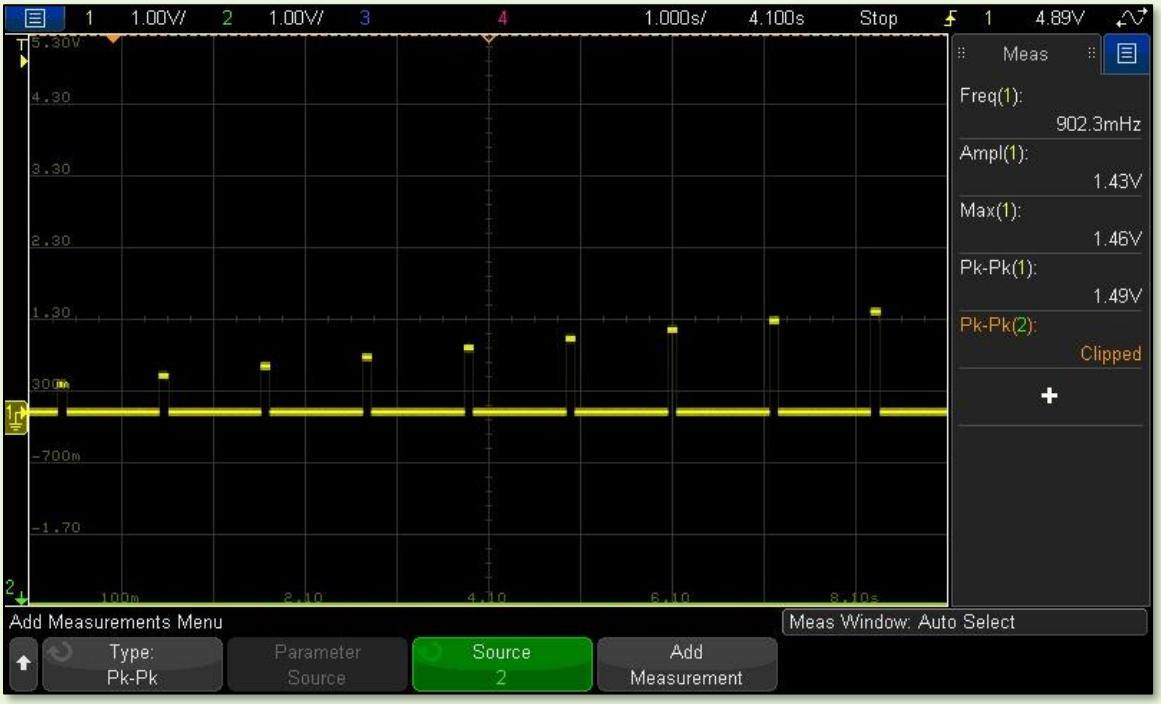


Fig 4: DAC Output

The Vsense+ we get on the oscilloscope is like the DAC as shown in Fig 5 below:

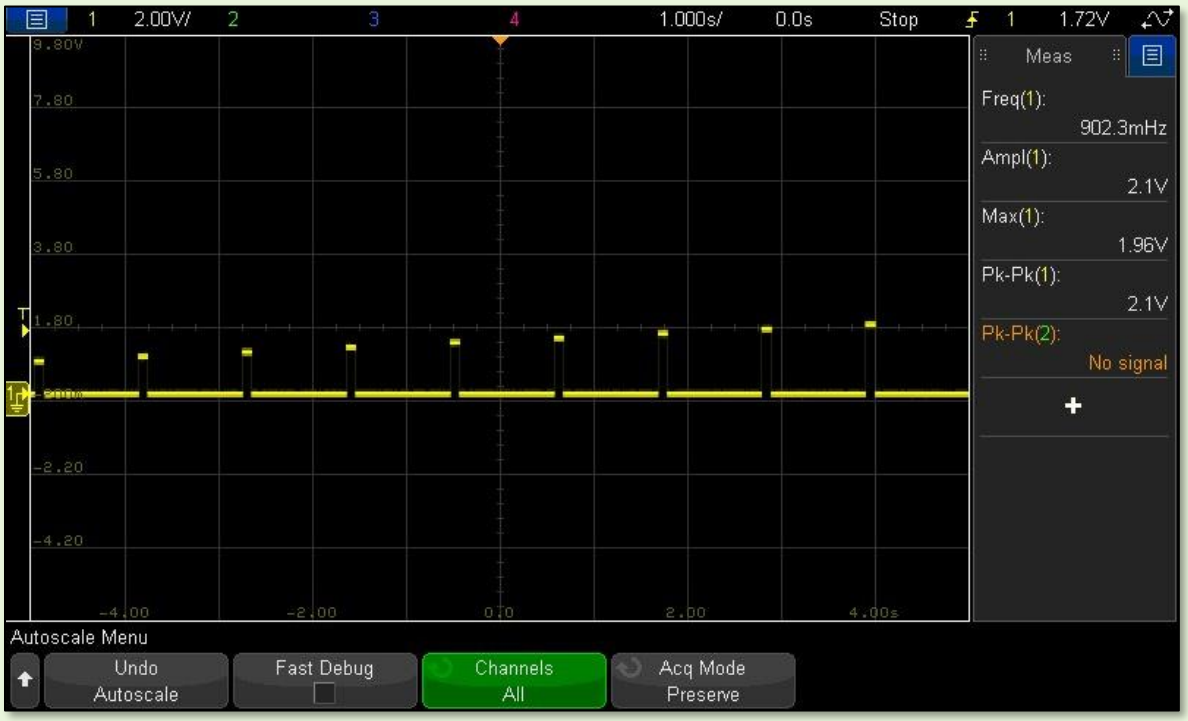


Fig 5: Vsense+ Output

Practical PCB Design and Manufacture

☞ *SBB VRM CIRCUIT*

The following VRM circuit reference was taken from the lab manual:

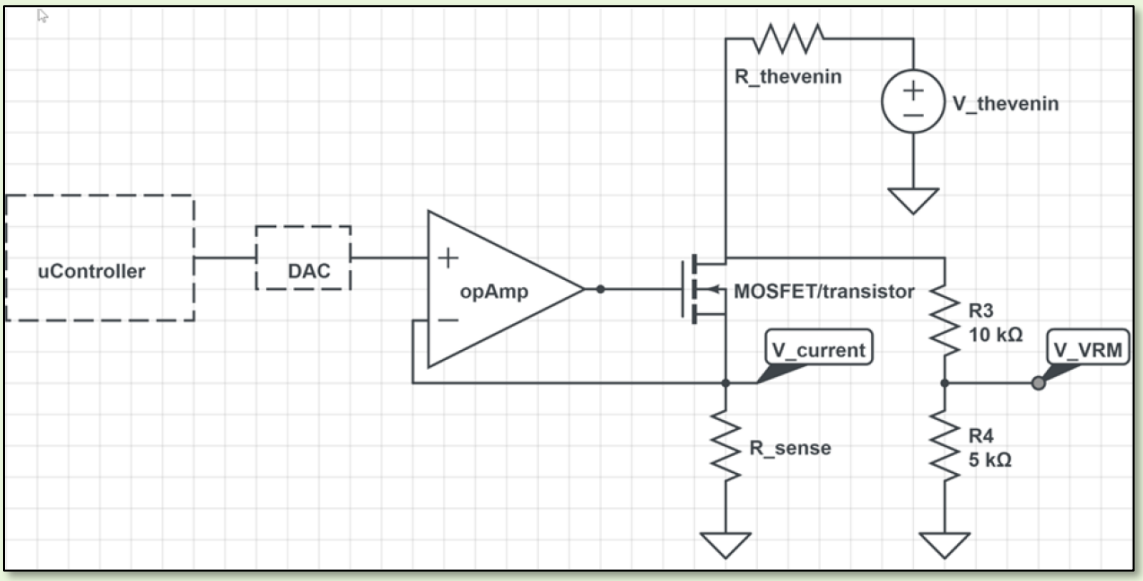


Fig 6: Instrument Droid Sketch

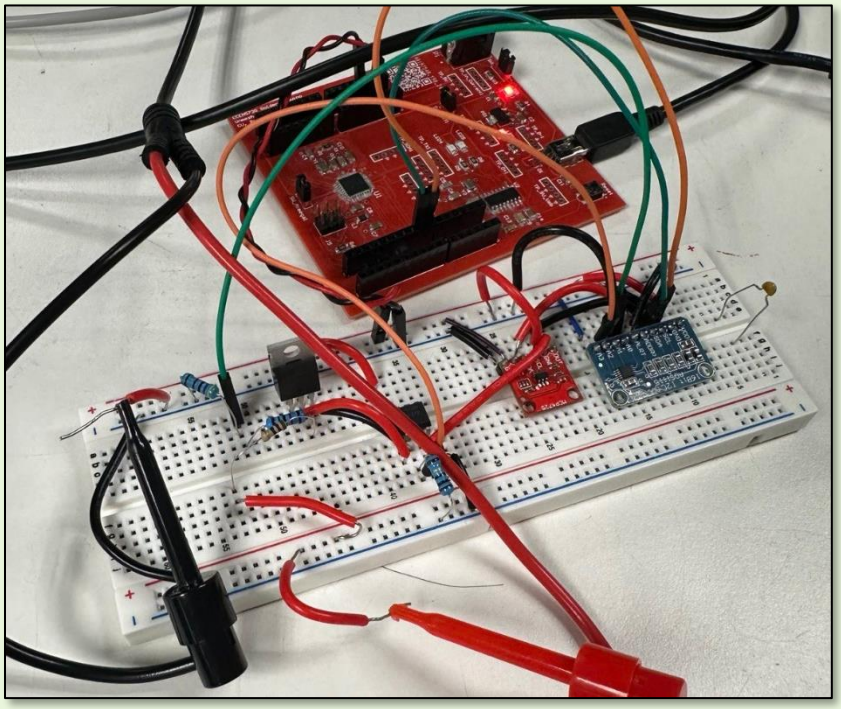


Fig 7: VRM SBB Circuit

Practical PCB Design and Manufacture

☞ *GRAPHS PLOTTED*

When connected to a 5V power supply, the following values of the current in mA and resistance in ohms were reflected as shown in Fig 8:

Fig 8: I vs R for 5V supply

Now when connected to a function generator with a 5V Dc settings, the following values of the current in mA and resistance in ohms were reflected as shown in Fig 9:

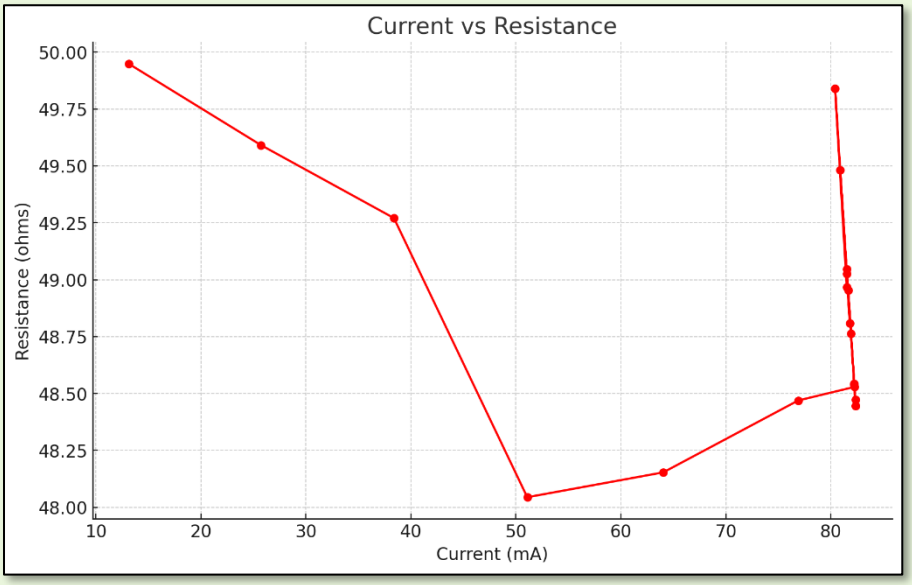


Fig 9: I vs R for Function generator

Practical PCB Design and Manufacture

☞ *BEST DESIGN PRACTICES*

I've added a lot of cutting-edge methods to my board 3, which compares successful and ineffective ways, to improve its dependability and performance:

1. **Regulated Power Supply:** Careful regulation of the power supply eliminates voltage instability, providing consistent current to the board.
2. **Ferrite Beads Integration:** Adding ferrite beads helps filter out high-frequency noise in the power lines, improving the board's electromagnetic compatibility.
3. **Short and Shielded Traces:** Keeping the trace paths short between vital components on the PCB lowers the risk of electromagnetic interference and signal loss.
4. **Comprehensive Ground Plane:** A well-integrated ground plane minimizes ground loop issues and ensures a stable reference voltage throughout the board.
5. **Strategic Test Points Placement:** Strategically positioned test points for essential signals and power outputs simplify the diagnostics and troubleshooting processes.
6. **Careful Component Layout:** Proper spacing of components prevents electrical and physical interference, minimizes short-circuit risks, and aids in heat management.
7. **Decoupling Capacitors Placement:** Placing decoupling capacitors near microcontrollers and other ICs provides stable voltage levels and dampens voltage fluctuations.
8. **EMI Shielding Measures:** Shielding around sensitive, high-frequency parts prevents electromagnetic interference, preserving the board's functionality.
9. **Thermal Management Solutions:** Features like thermal vias or heatsinks near heat-generating components help maintain safe operational temperatures.
10. **High-Quality Connectors:** Choosing robust connectors designed for frequent use ensures durable connections and contributes to the board's overall durability.
11. **Layer Stack-Up:** Configure with top and bottom signal layers, sandwiching inner grounds for improved EMI shielding and signal integrity.
12. **Impedance Control:** Ensure consistent impedance across signal traces to prevent signal degradation, especially important for high-speed data transmission.
13. **Decoupling and Power Integrity:** Place decoupling capacitors near IC power pins and design a continuous power plane to stabilize voltage levels and reduce noise.

Practical PCB Design and Manufacture

☞ *MISTAKES MADE*

After making errors in the previous boards, I was lucky enough to not make any hard errors in this board 4. The best design practices helped me to successfully run the board as expected. **Also, no soft errors were encountered.**

Although one mistake from my side was that I had connected the switch to the I ohm resistor and when I connected an external supply from the power jack, the resistor got burnt.

☞ *ANALYSIS AND EFFECTIVE LEARNINGS*

1. This design project made it simpler to see the importance of a well-designed layout and the reasons large-scale commercial designs shouldn't be viewed as perfect.
2. It's crucial to check the design before submitting.
3. The instrument droid uses the load current it receives from the VRM to determine the source's Thevenin resistance.
4. When designing the PCB, keep in mind that the Near Field Emission effect can generate Far Field Emission, which can interfere with EMC regulations.
5. It is best to use 2% duty for MOSFETs instead of large duty ones, as they can cause harm to the circuit and detect resistance and ultimately produce more current.
6. When soldering important components like resonators, exercise extra caution and pay attention to their footprints.
7. Never share a single trace or wire since doing so increases ground bounce on the return path.

☞ *CONCLUSION*

4-layer PCB boards offer improved signal integrity and reduced noise, making them ideal for high-speed and high-density applications. The circuit designed correctly measures Thevenin's resistance and voltage of any power supply. The measured resistance is close to the actual value, and the differential measurement method reduces noise. It's better to use a MOSFET than a transistor to avoid additional base current. The ground plane should be continuous, and individual traces or wires should not be used as a return path to prevent ground bounce. Decoupling capacitors and ferrite filters should be used judiciously. Signal paths should be spaced out to reduce crosstalk. MOSFETs are preferred over BJTs to avoid additional current. Ground vias should be placed beneath the IC surface and return vias should be placed adjacent to signal vias to decrease impedance and noise.