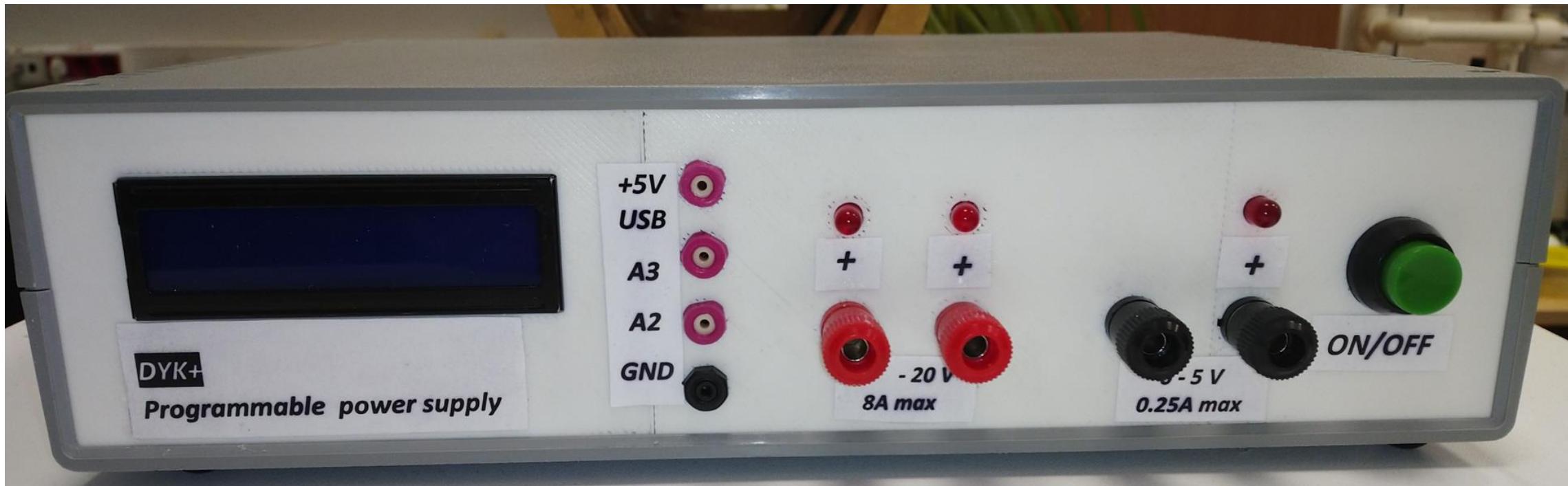


Programmable DC power supply based on Arduino UNO

Dmitriy Makhnovskiy, [DYK](#)

Nikolay Udanov and Egor Shevchenko, [MISIS](#)



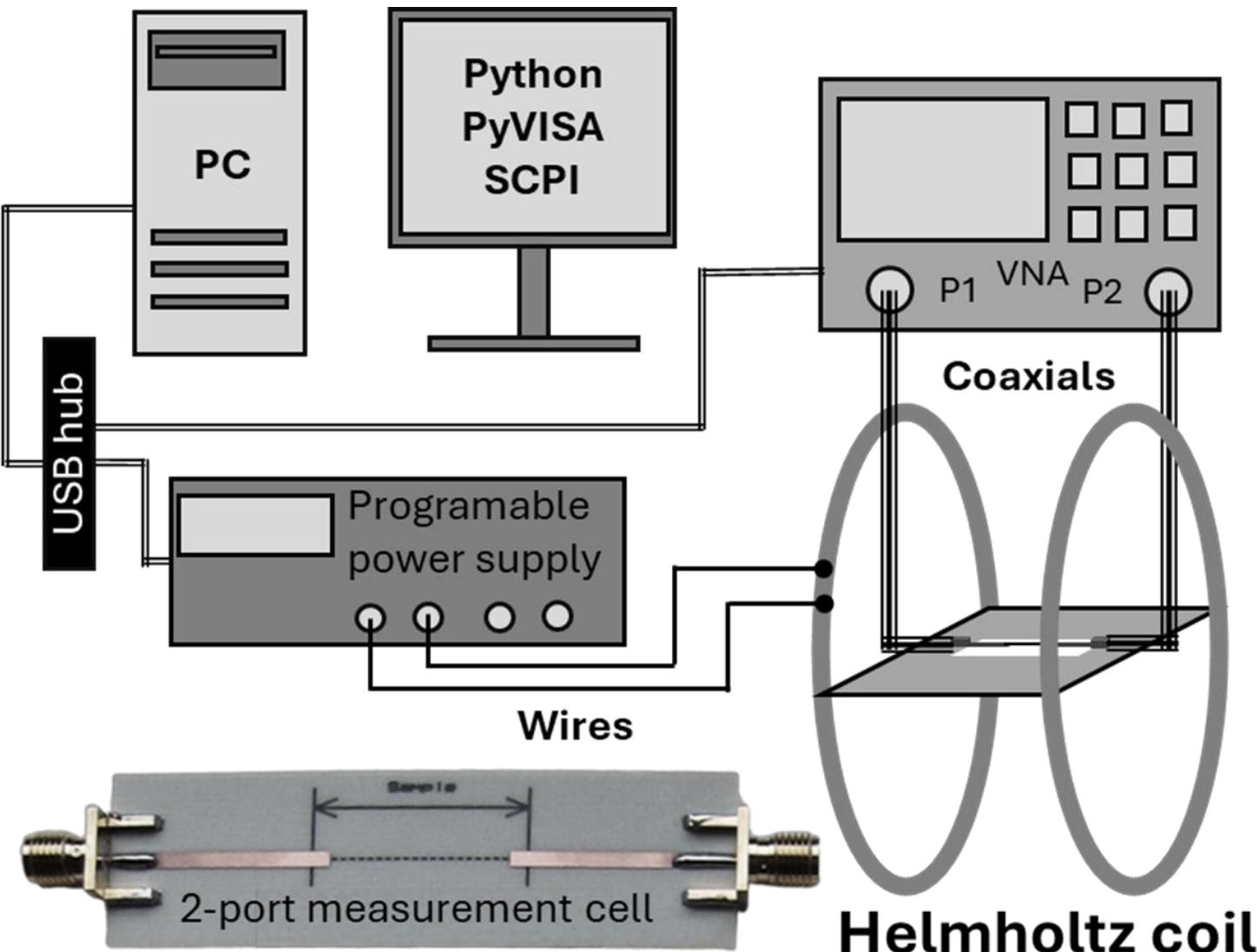
GitHub: <https://github.com/DYK-Team/Programmable-DC-power-supply>

In the realm of test and measurement, the concept of [software-defined instrumentation](#) is revolutionizing how we design and implement laboratory equipment. [Our programmable DC power supply](#), built on an Arduino UNO and a DAC, exemplifies this paradigm shift. By leveraging software-defined instrumentation principles, we deliver a versatile, cost-effective solution that integrates multiple functions into a single reconfigurable device. By simply uploading different Arduino's sketches, users can easily reconfigure the device for various tasks, including integrating sensors. This approach enhances flexibility, simplifies maintenance, and ensures adaptability to evolving technological demands.

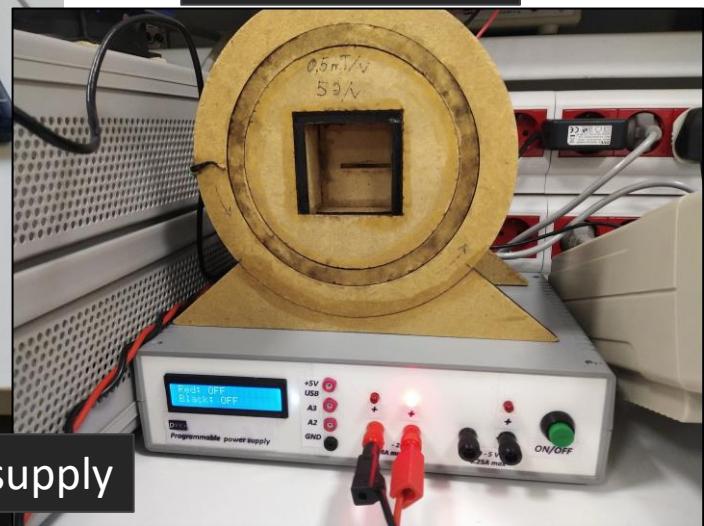
Overview of applications: MI measurements and motor control

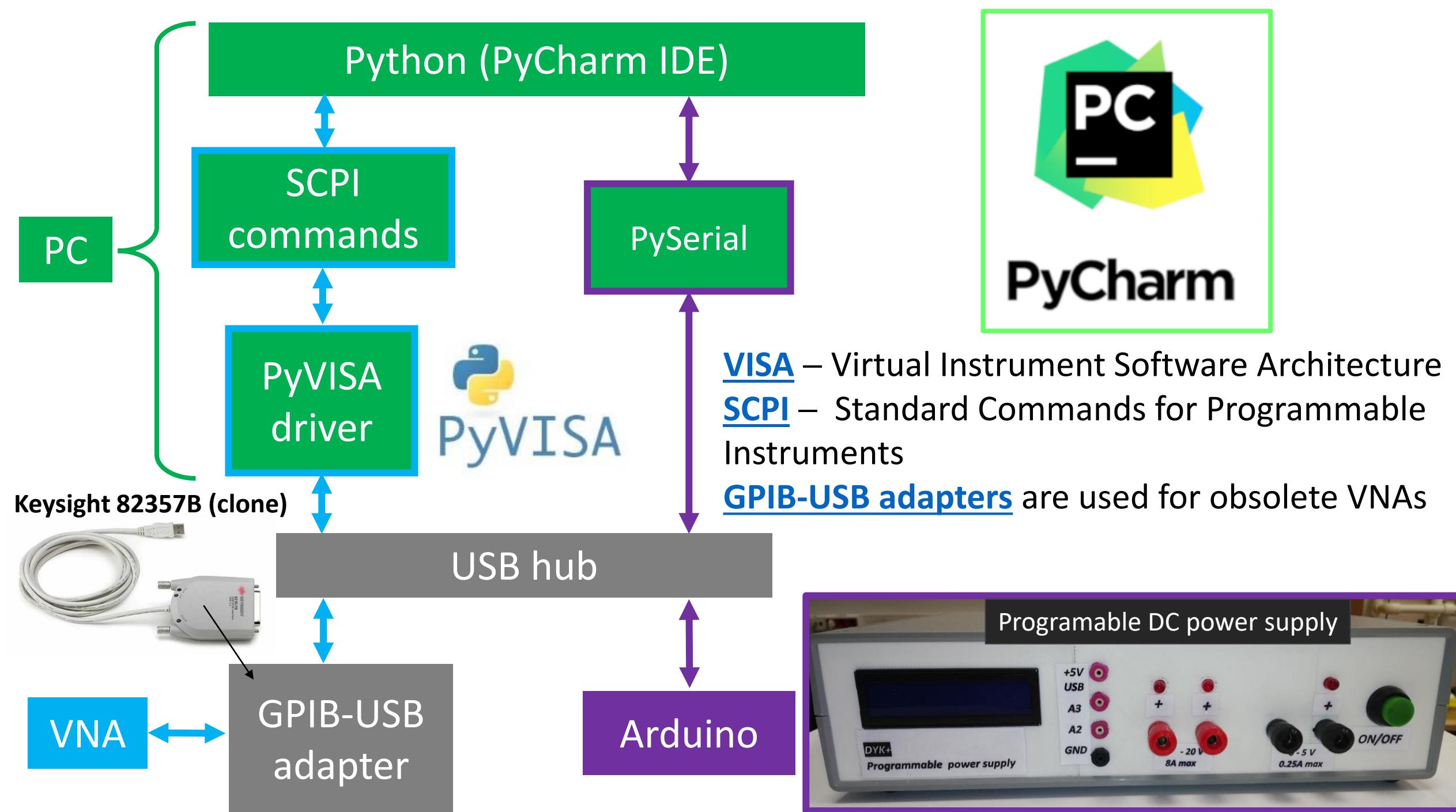
The programmable DC power supply in this project was developed for integration into an automated system designed to measure [magneto-impedance](#) (MI) in ferromagnetic samples, such as wires or thin films.

The setup in its minimal configuration comprises a Vector Network Analyzer ([VNA](#)), the programmable DC source, a [Helmholtz magnetic coil](#), and specialized measurement [PCB cells](#) for connecting samples to microstrips. The synchronous operation of two USB-connected devices is managed by a Python program. Commands for VNA are transmitted in [SCPI](#) script format using the [PyVISA](#) library, while commands for [Arduino](#) inside of the programmable source are sent in script format via the [PySerial](#) library. Python has already become a popular programming language for [test automation](#).



Arrangement of
the laboratory
for magneto-
impedance
measurements
at [MISIS](#)



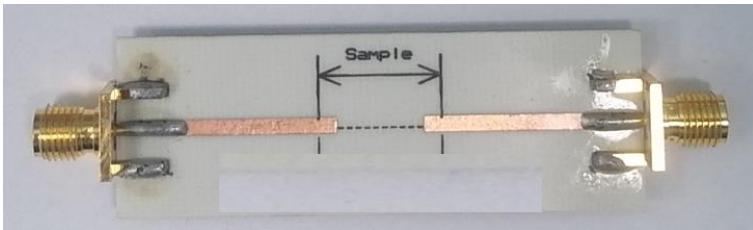


MI measurements: impedance field dependence

Loop back

$$V_c = V_c - 2 \times (i - 1) \times V_c / (N - 1)$$

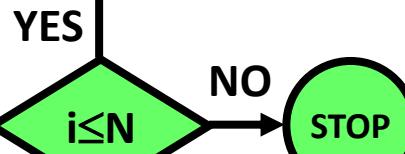
$$V_c, N \\ i = 1$$



Loop forward

$$V_c = -V_c + 2 \times (i - 1) \times V_c / (N - 1)$$

PySerial



Coil's LR parameters

Pause VNA's sweep
for $\tau = 20 \times L/R$

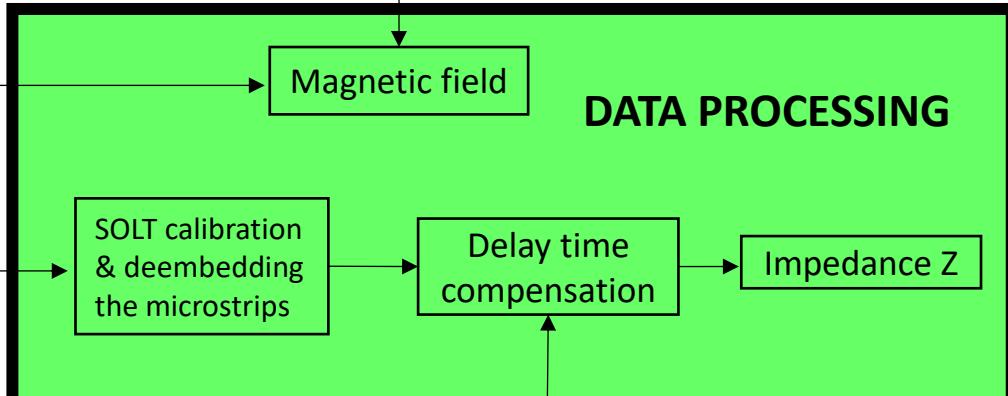
Coil

V_c

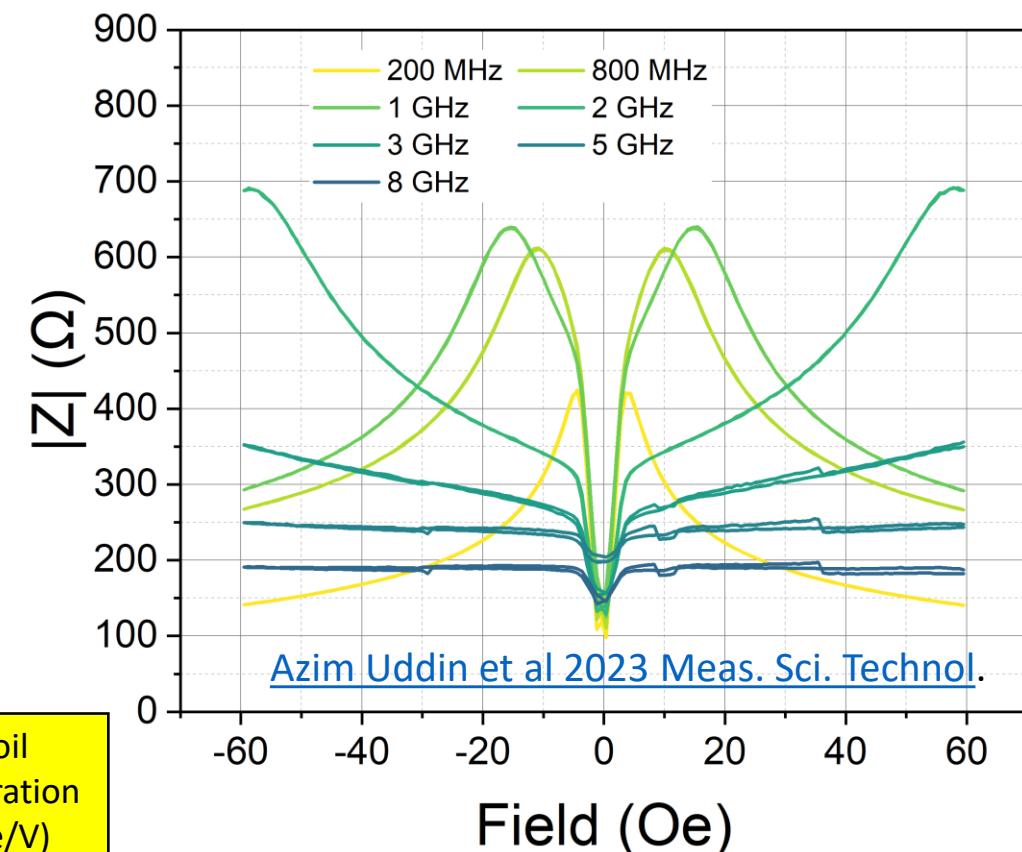
VNA measures S21/11 over N_f
frequency points within $[f_{start}, f_{stop}]$

PyVISA

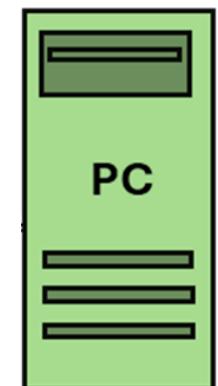
$i = i + 1$

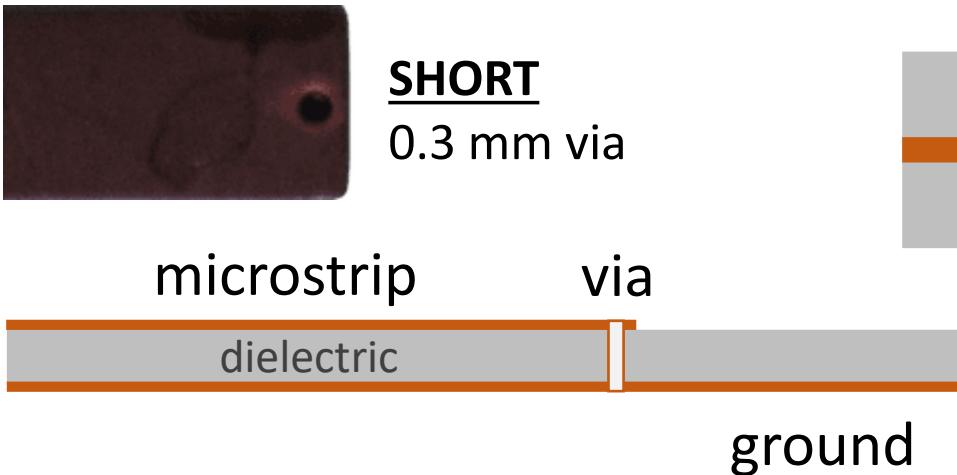
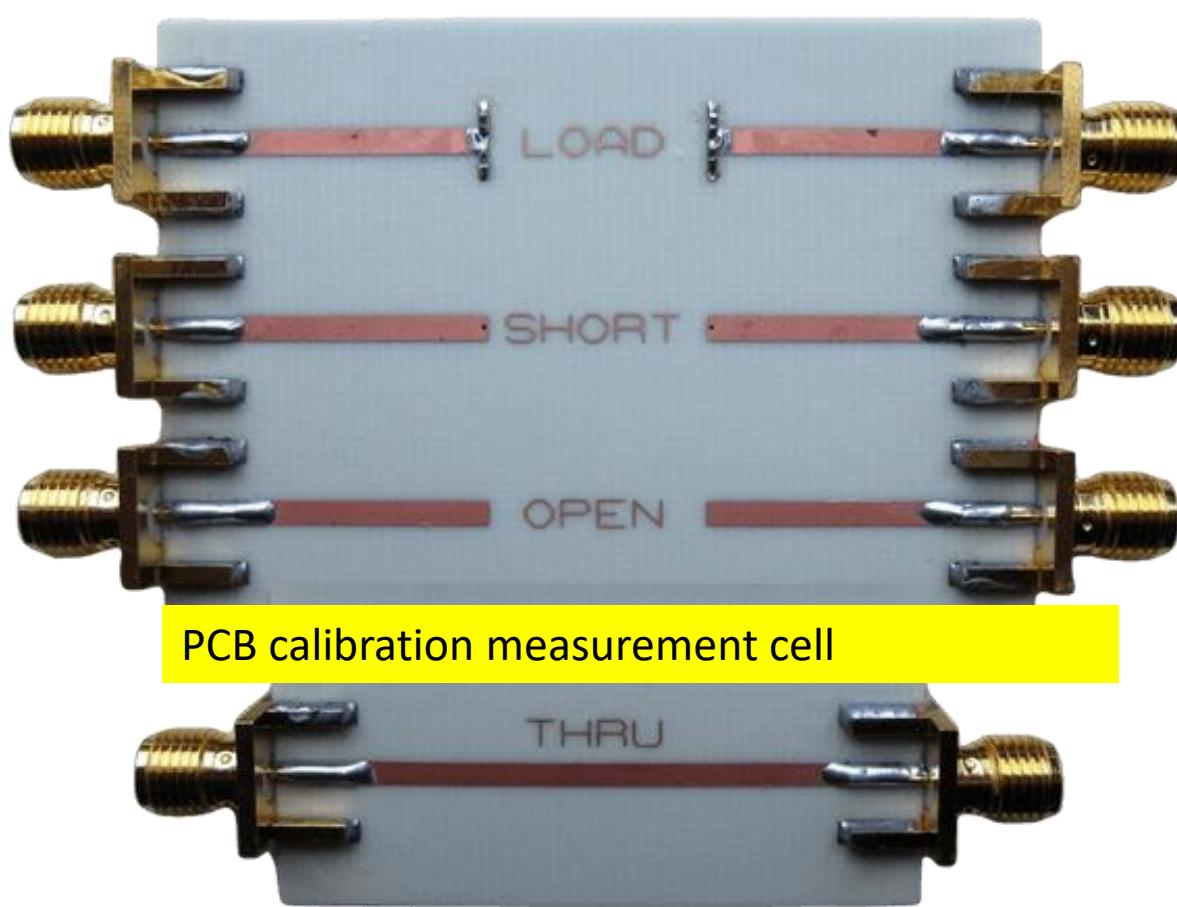


Sample's delay time

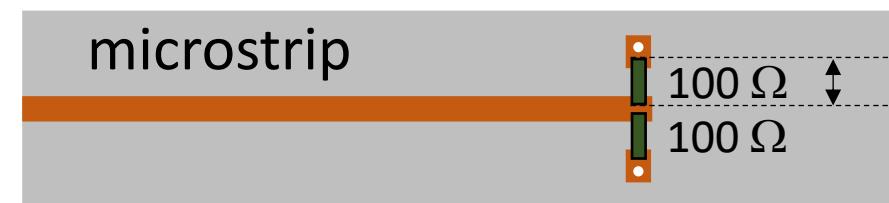
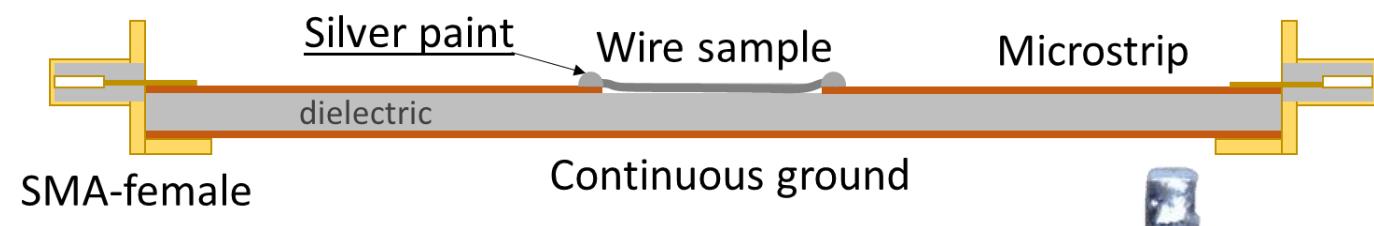
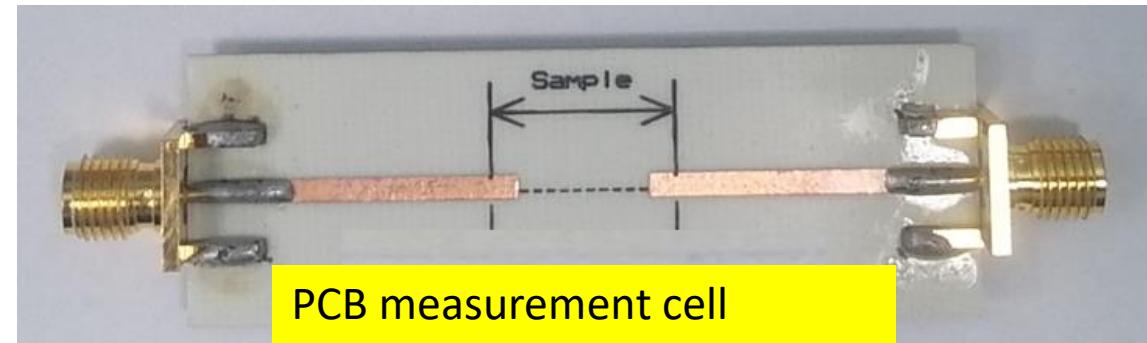


Azim Uddin et al 2023 Meas. Sci. Technol.





PCB microstrip cells used for the calibration and MI measurements



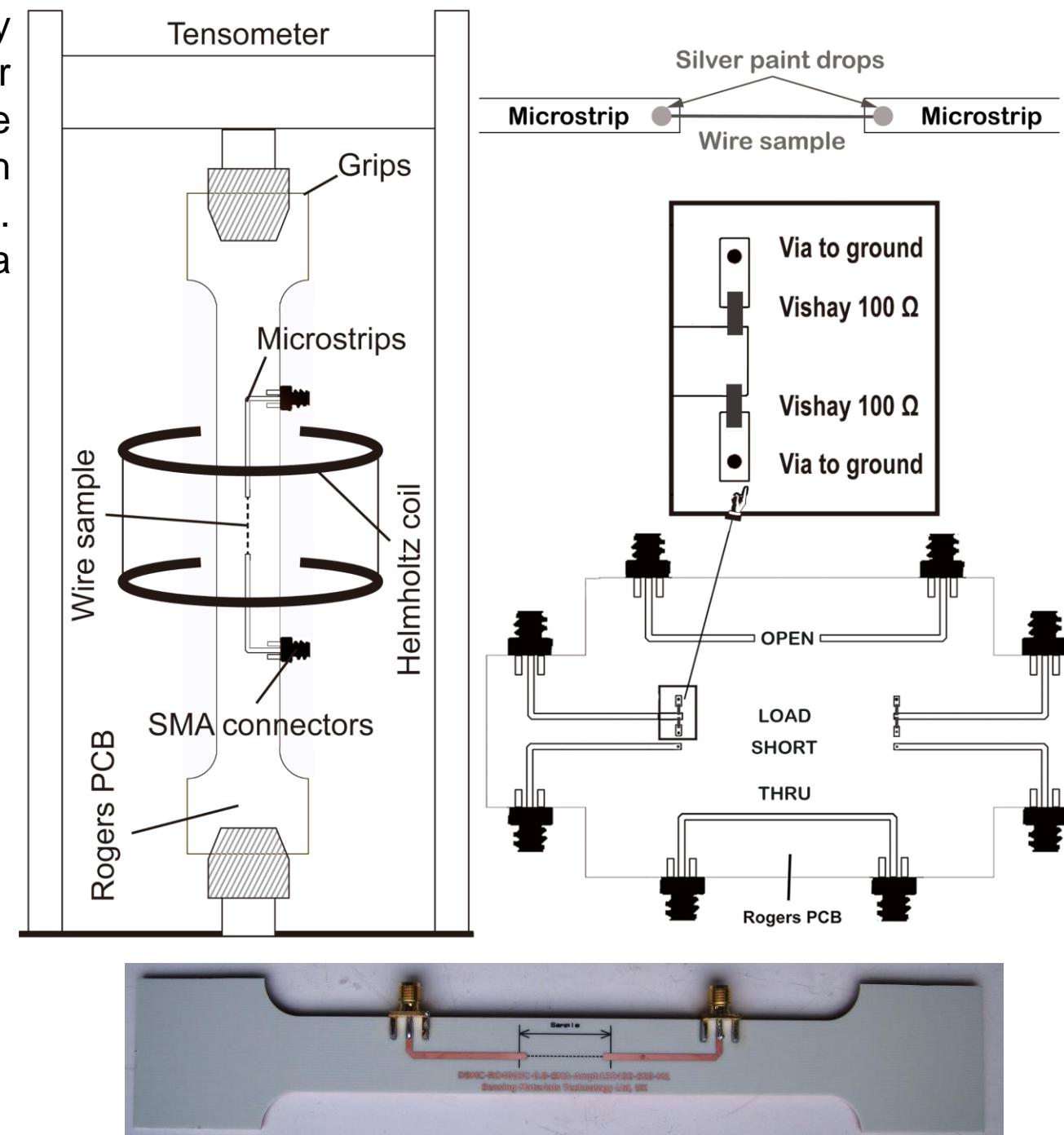
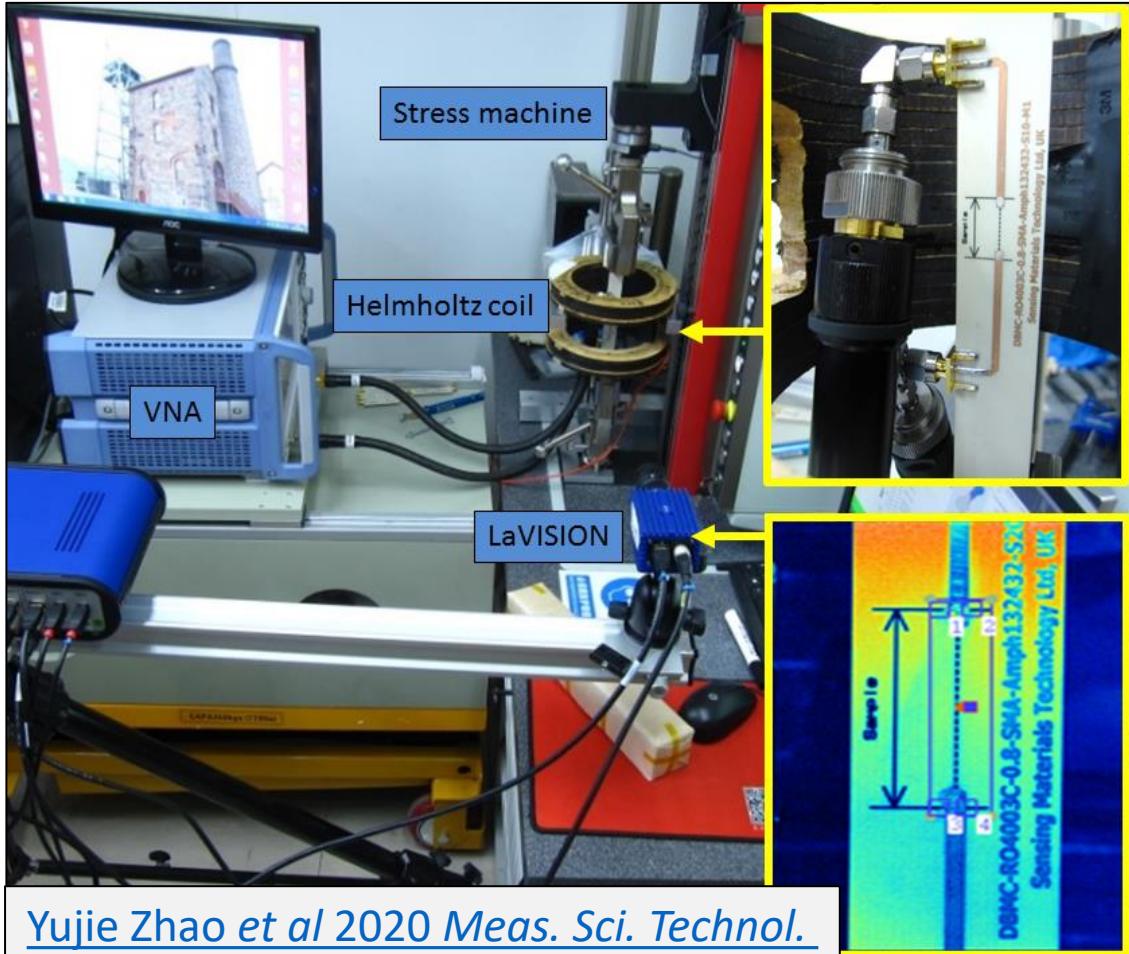
1.626 mm
(size 0603)

Pad with via

LOAD
Two parallel
100 Ω Vishay resistors



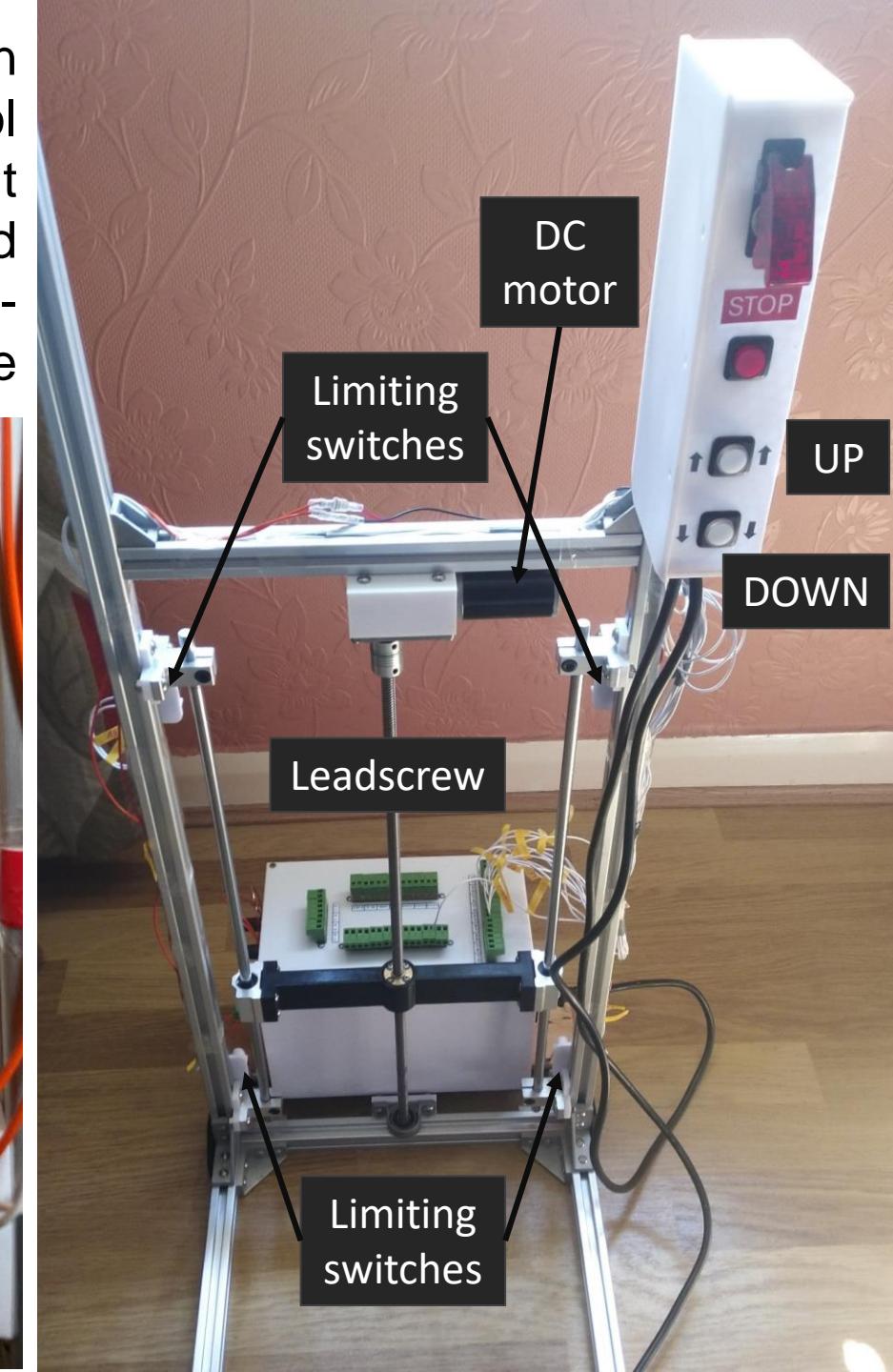
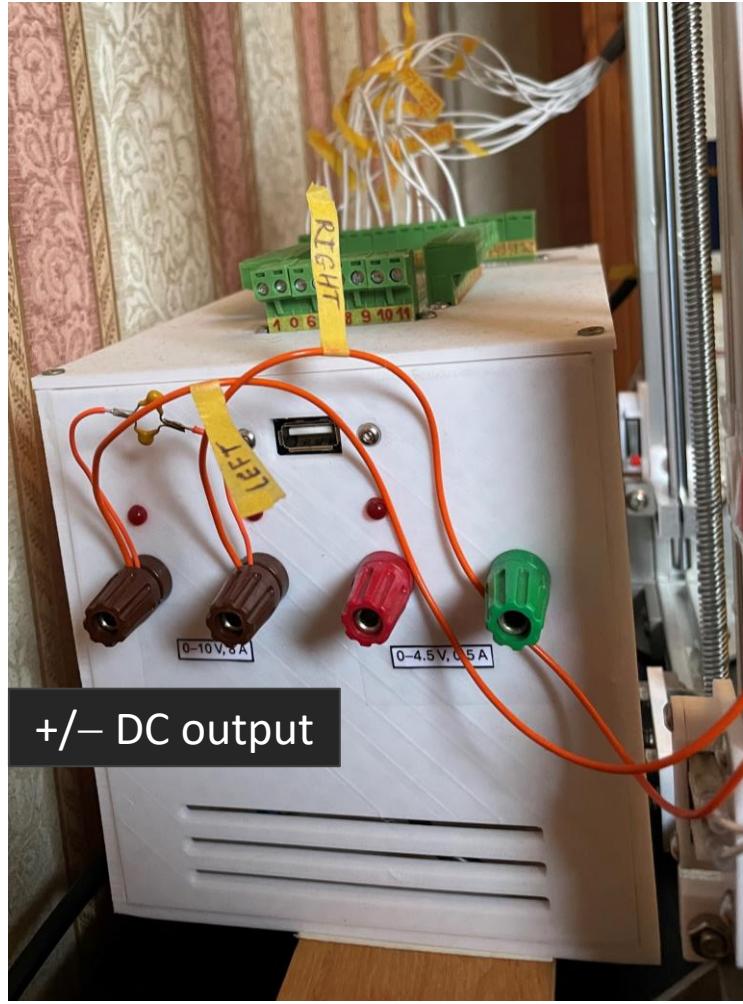
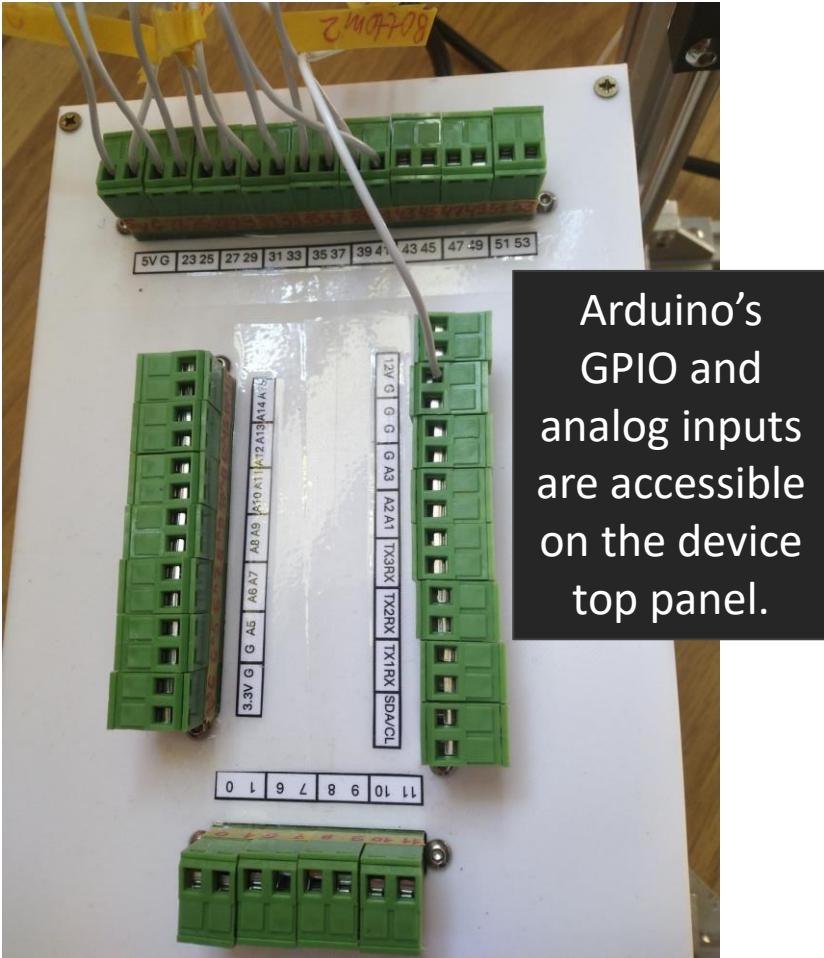
An advanced setup may incorporate devices to apply additional stimuli to the sample, such as heating or tension. Tension can be transferred to the sample through a specially designed "dogbone" PCB cell, which is secured between the grippers of a stress machine. This configuration has already been implemented for a microwave laboratory at Zhejiang University, China.



Other implementations of the MI measurement setups



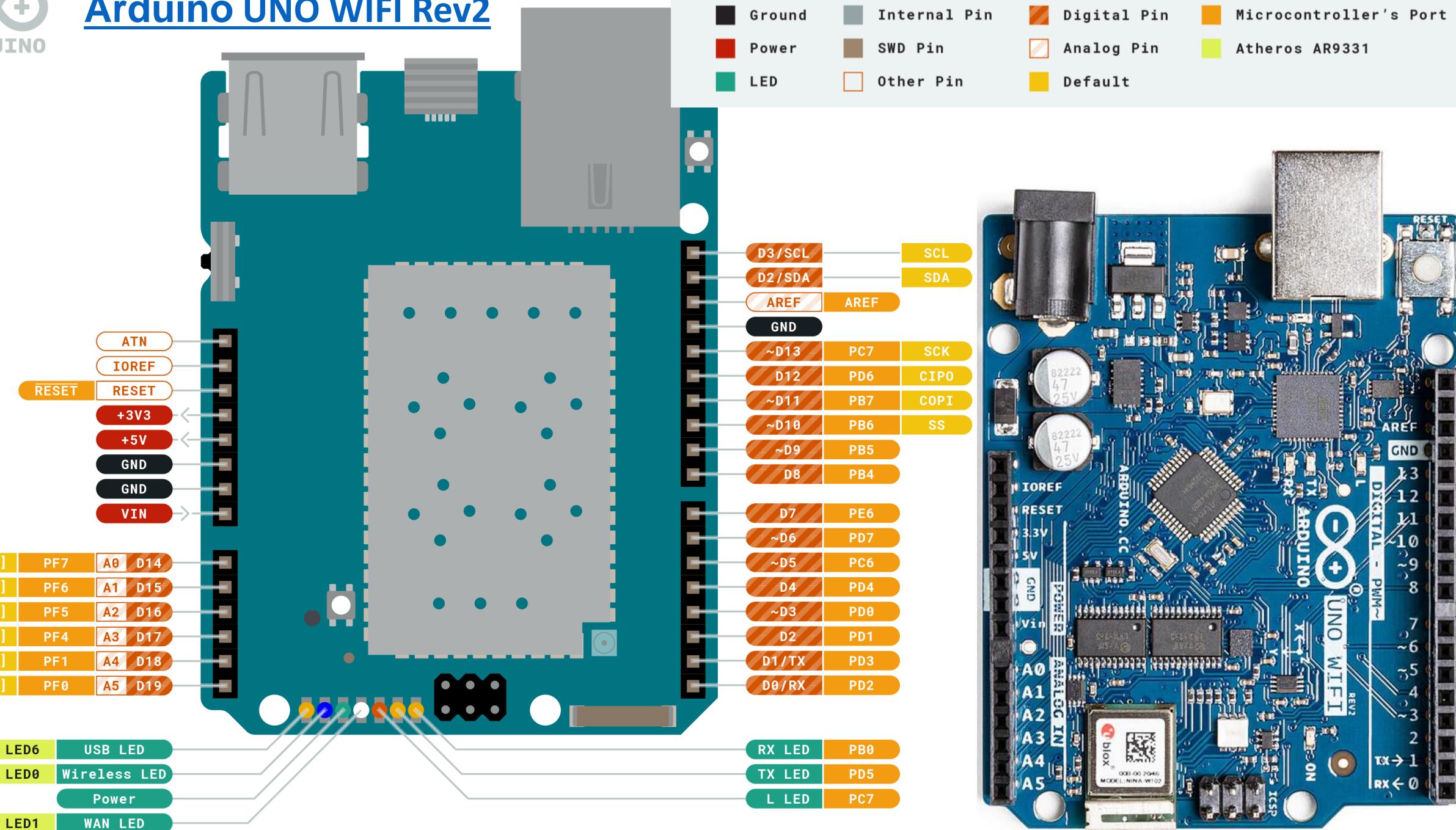
A similar DC power source with additional [GPIO](#) from Arduino can be easily reprogrammed to operate a lifting platform with control buttons (stop/up/down) and mechanical limit switches that prevent movement beyond the vertical guides. Both the control buttons and limit switches are "soft", meaning they are not connected to high-current circuits. Instead, they handle high/low potentials on the GPIO.



Design of the programmable power supply



Arduino UNO WIFI Rev2



ARDUINO with a shield

DAC
MCP4725

GRD

+5 V

A0

A1

⋮

D8

D9

D10

D11

D12

D13

0/1

PWM

0/1

2.2k

100 μ F

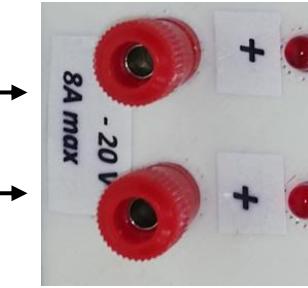
+24 V
0–5 V

OPA
549

D12 D13 +5 V

Relay
H-bridge

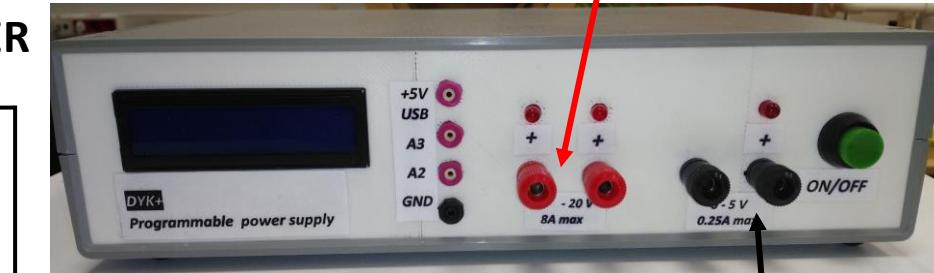
Architecture of the power supply



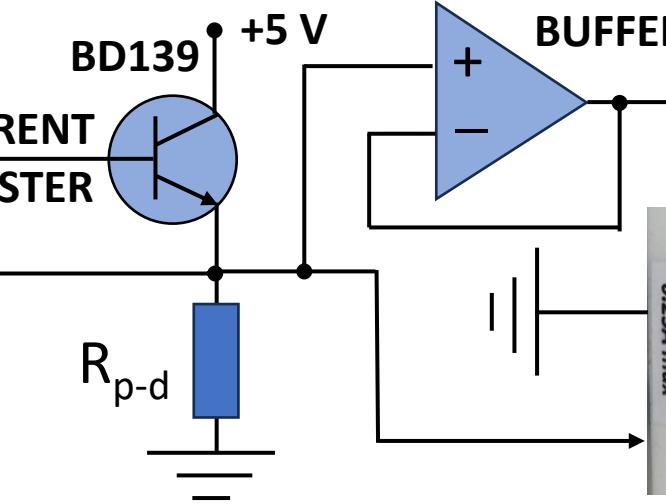
D8 (LED)

D10 (LED)

Dual polarity
“Red Output”,
max 20 V, 8 A



Single polarity
“Black Output”,
max 5 V, 250 mA



+5 V

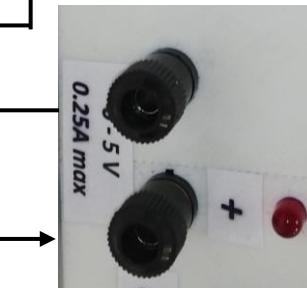
BD139

CURRENT
BOOSTER

R_{p-d}

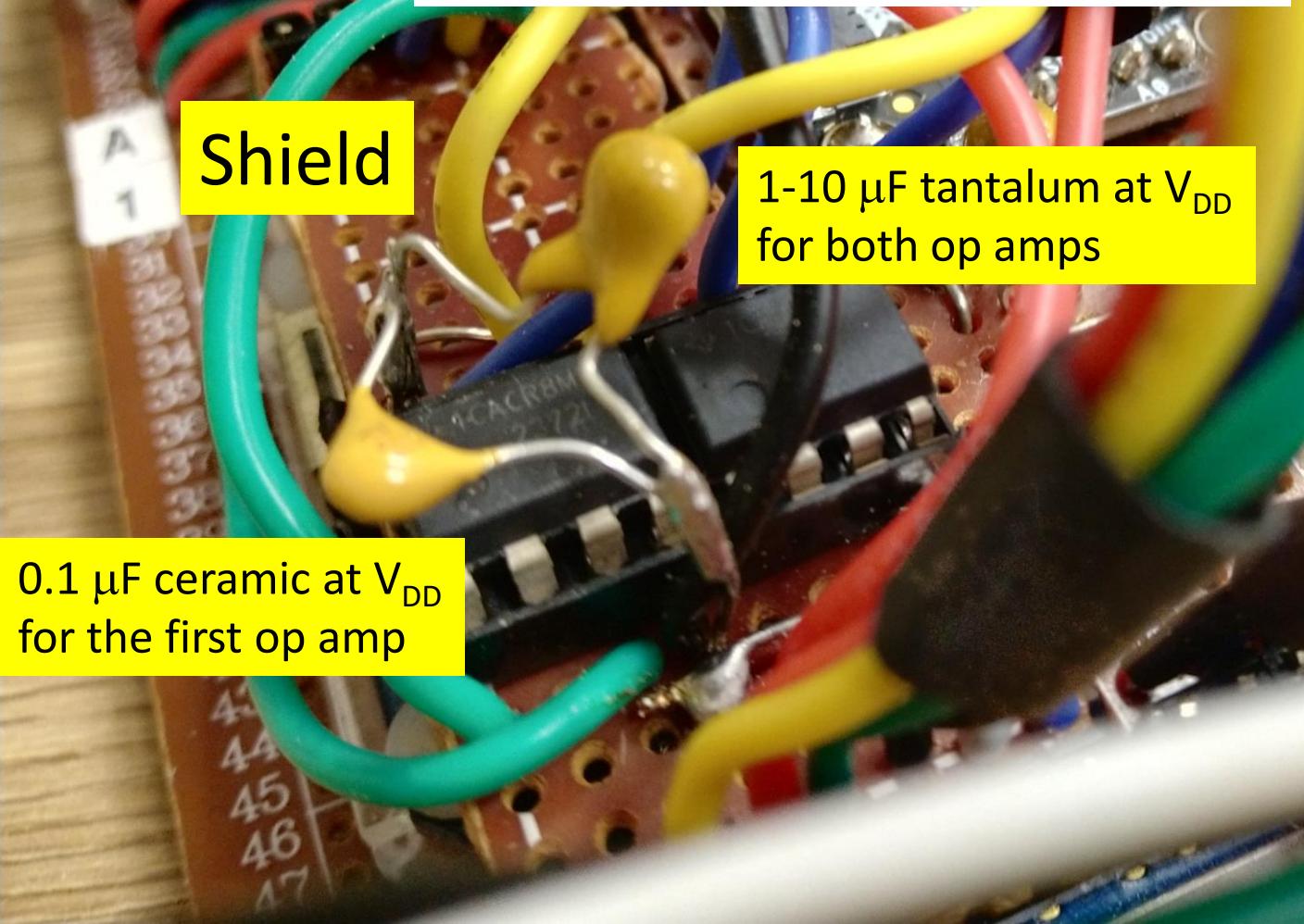
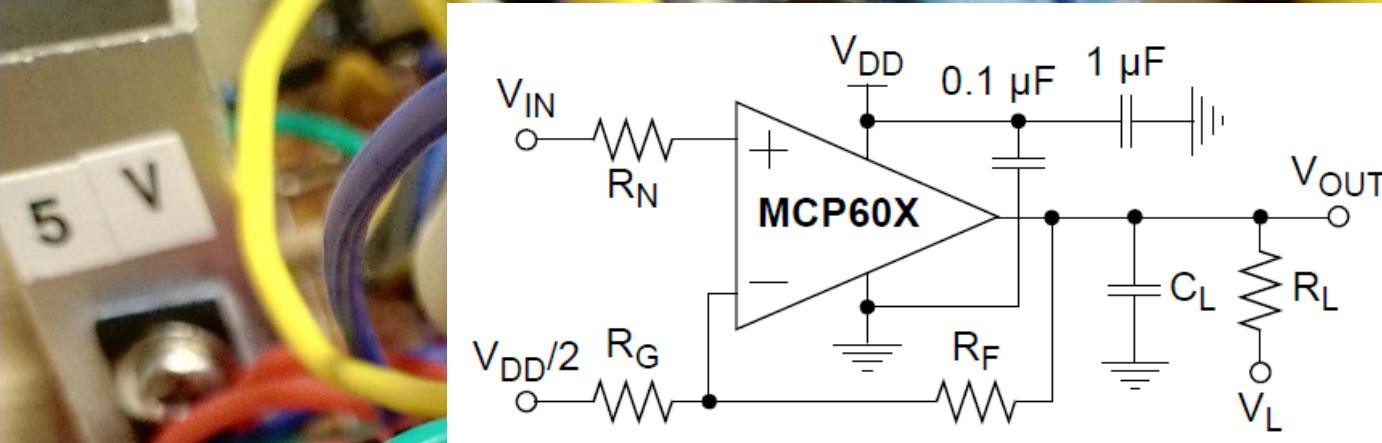
+5 V

BUFFER



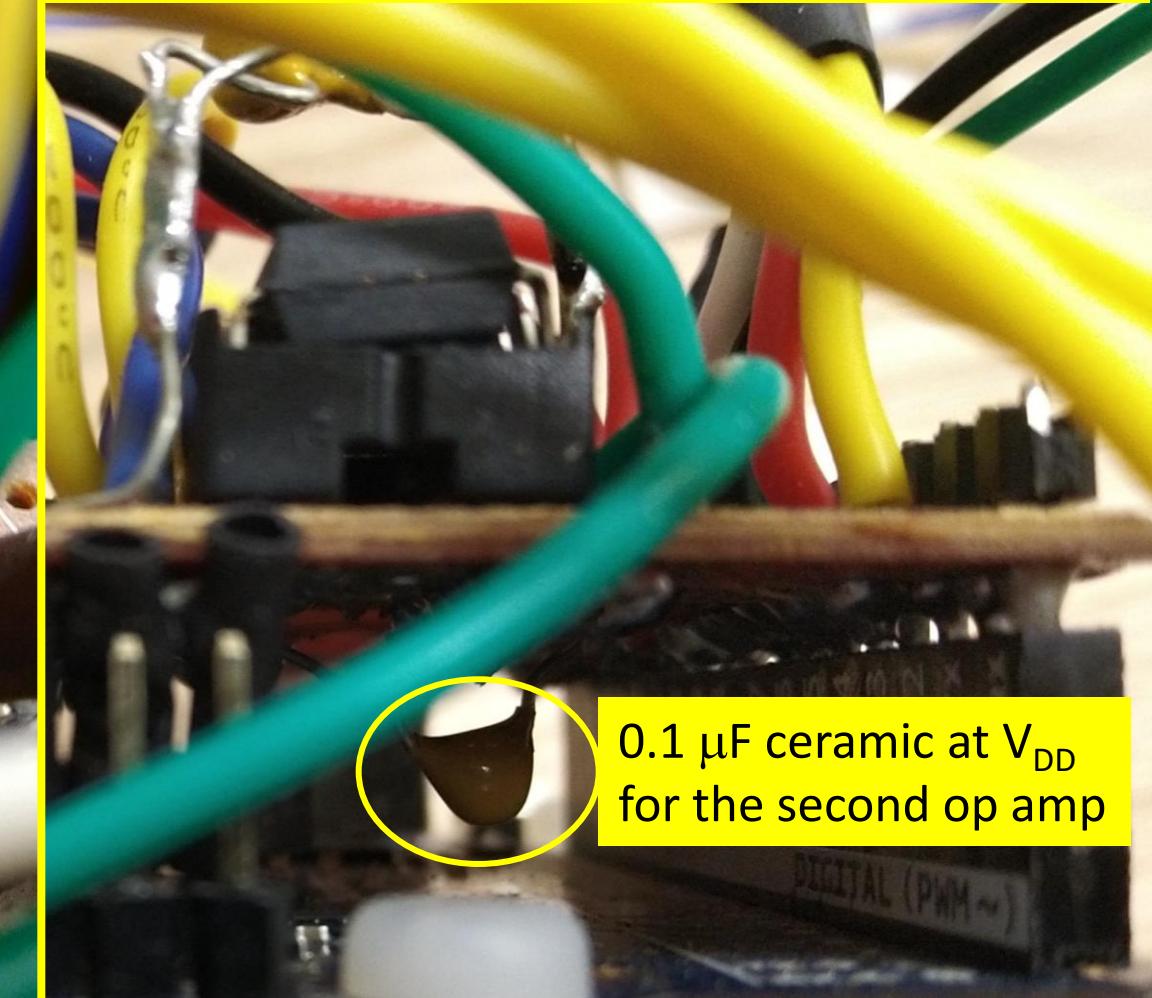
D11
(LED)

R_{p-d} is a [pull-down resistor](#) 10–100 k Ω to prevent a floating analog input on ADC

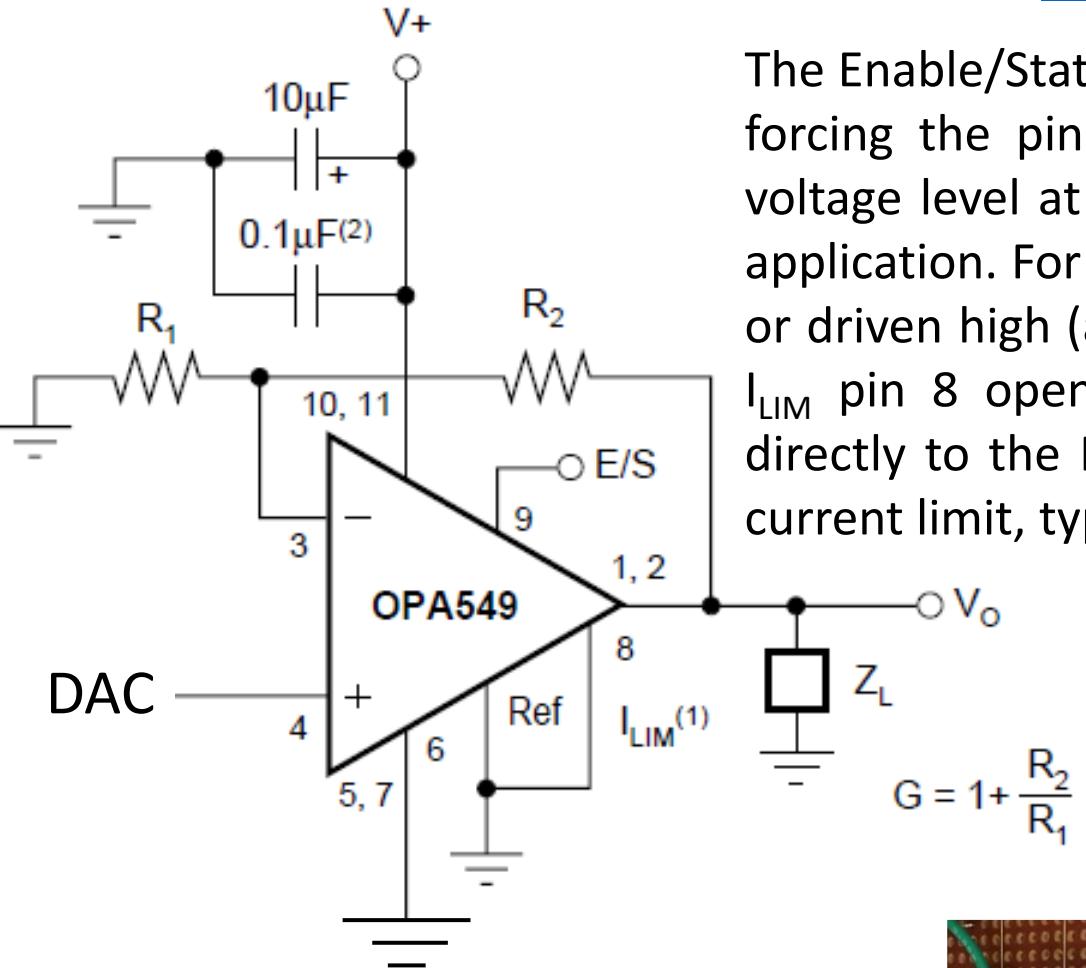


5 V DC supplied for low-power op amps from Arduino's 5 V reference

Recommended op amps: MCP606 ($\times 2$) or MCP607 ($\times 1$)



OPA549 pins



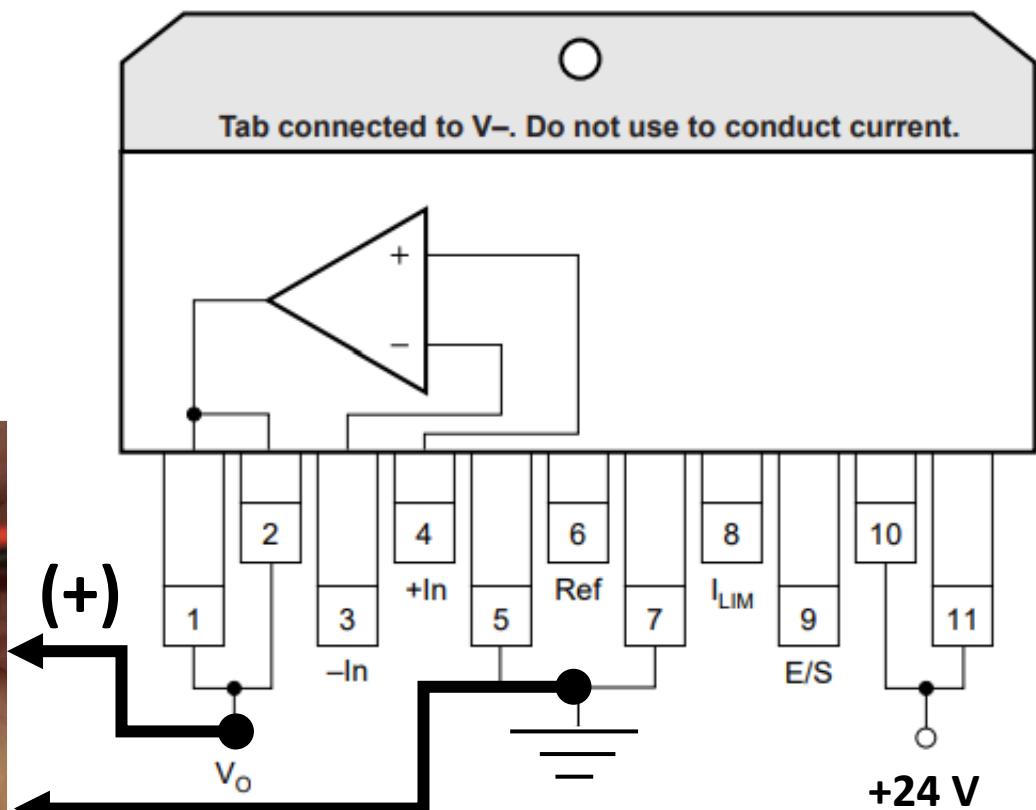
Connect both pins 1 and 2 to output.

Connect both pins 5 and 7 to GND

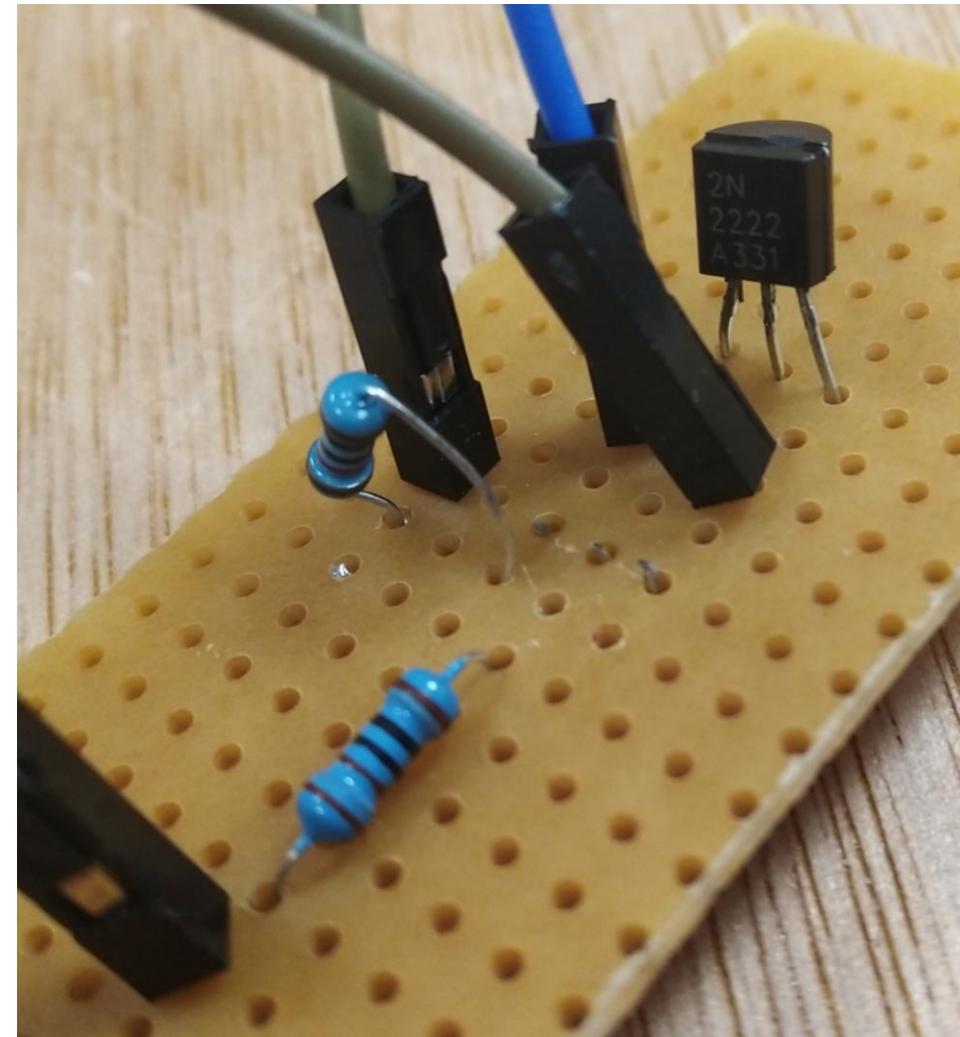
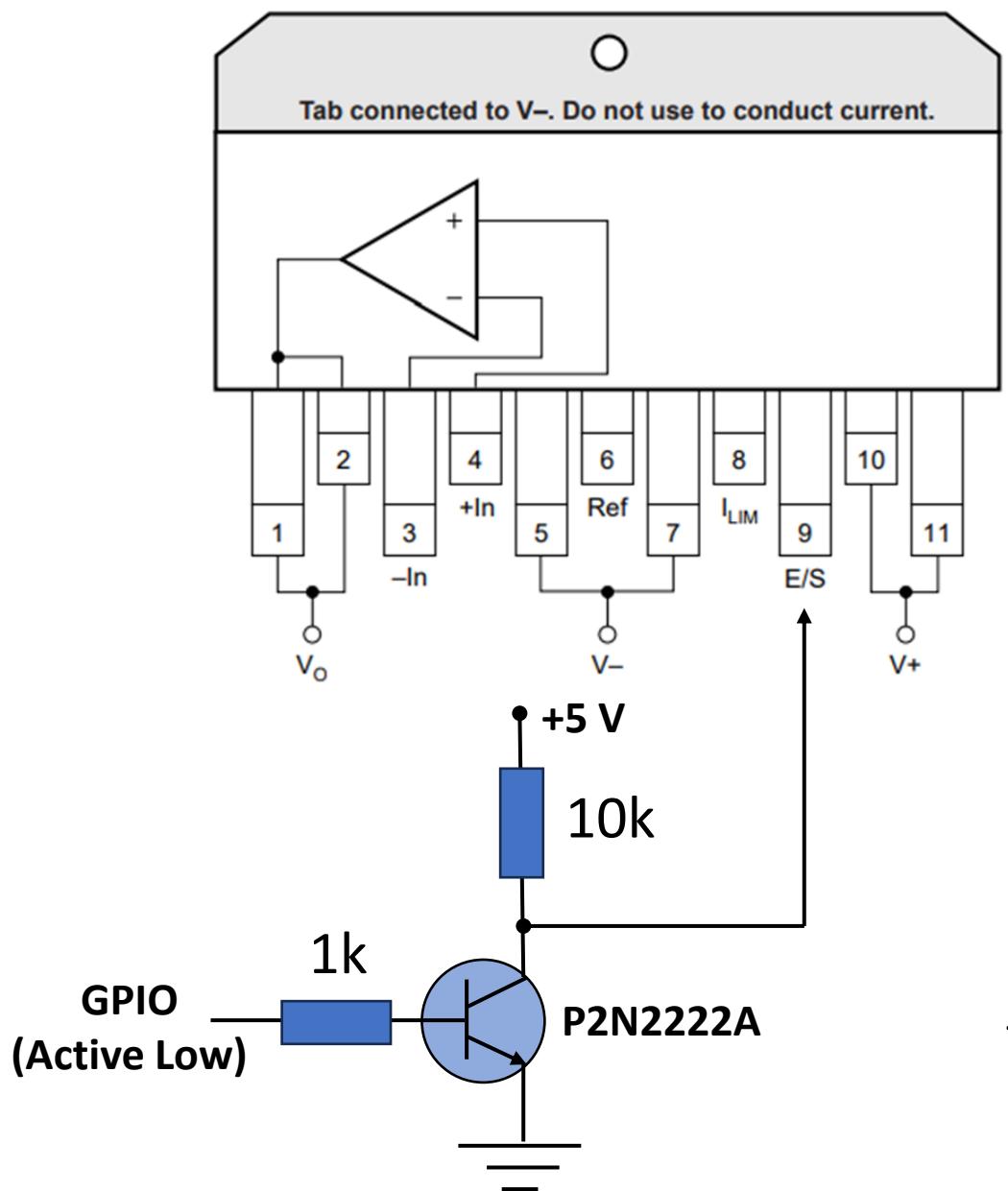
Connect both pins 10 and 11 to V+.

The Enable/Status (ES) pin 9 provides two unique functions: 1) output disable by forcing the pin low, and 2) thermal shutdown indication by monitoring the voltage level at the pin. Either or both of these functions can be utilized in an application. For normal operation (output enabled), the E/S pin can be left open or driven high (at least 2.4 V above Ref). We connected it to + 5 V. Leaving the I_{LIM} pin 8 open programs the output current to zero, while connecting I_{LIM} directly to the Ref pin 6 (and then to ground) programs the maximum output current limit, typically 10 A.

H-bridge

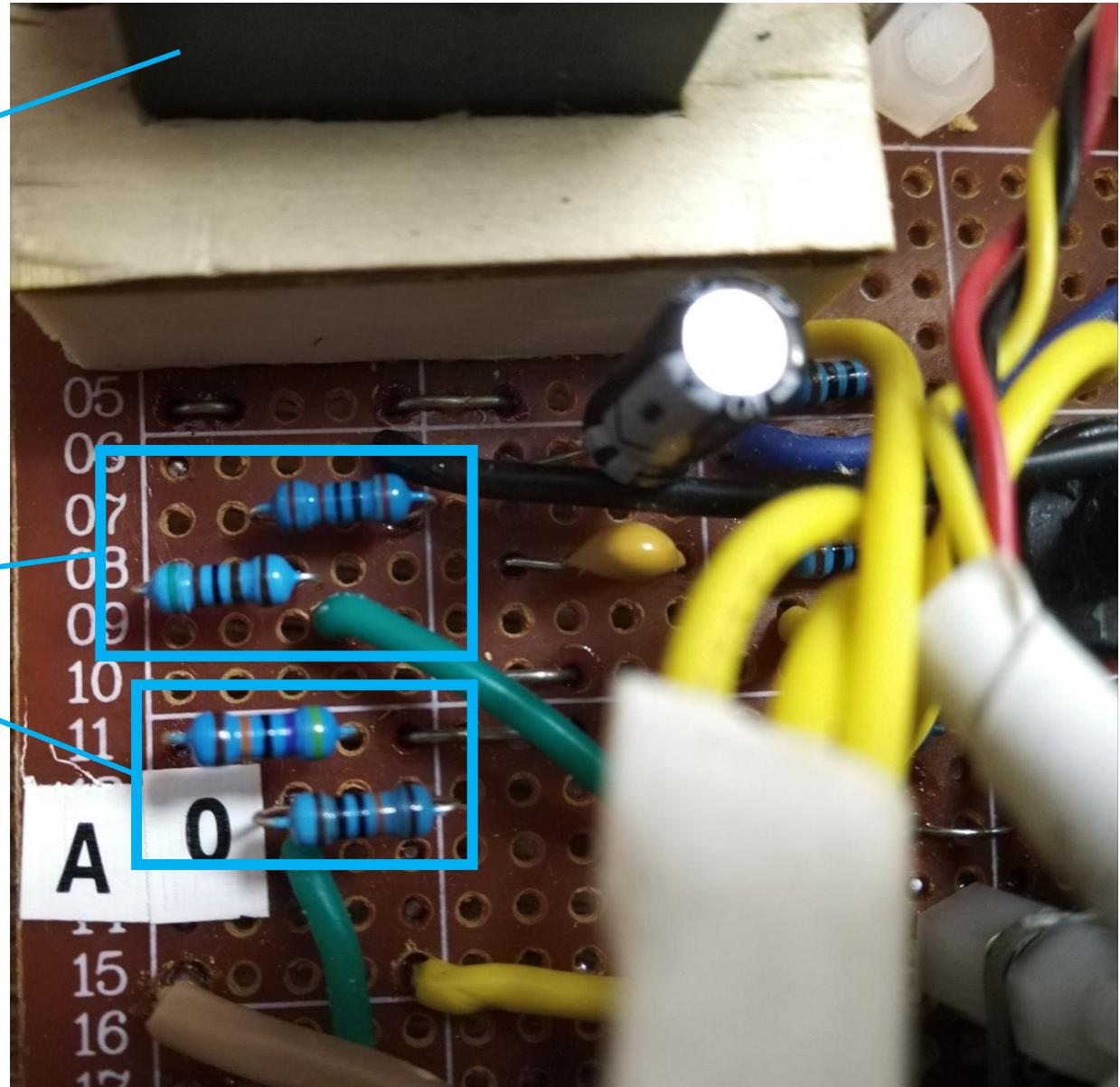
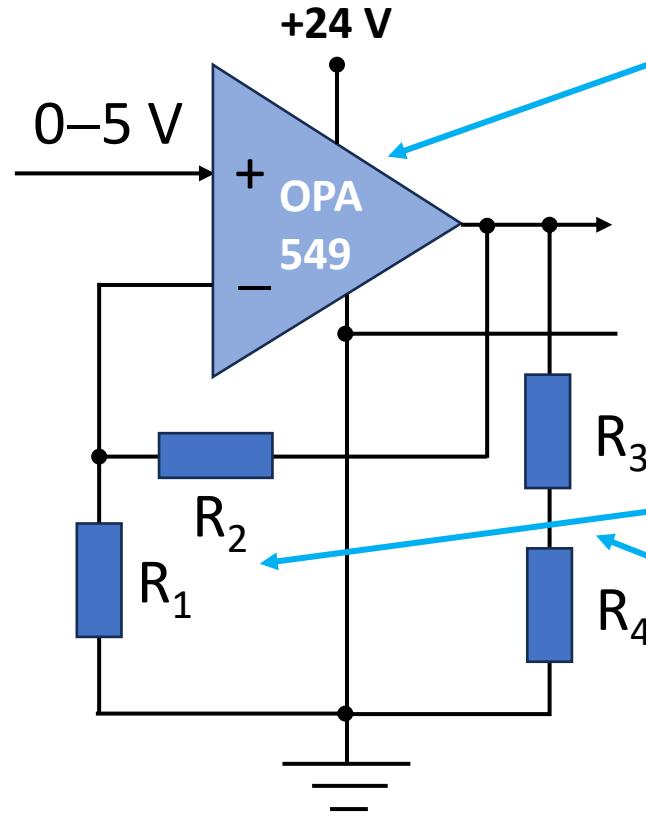


Digital control of the ES pin 9

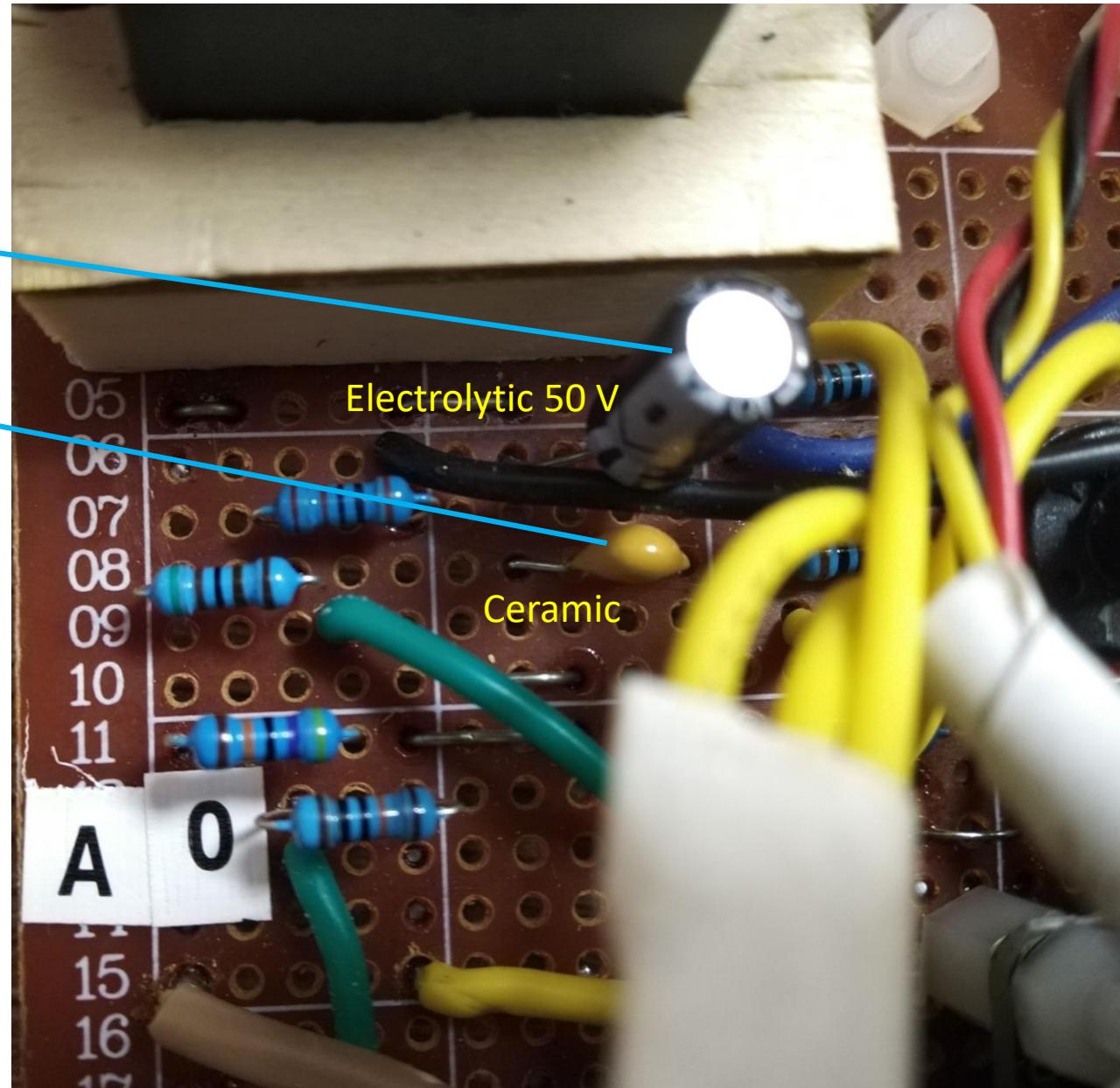
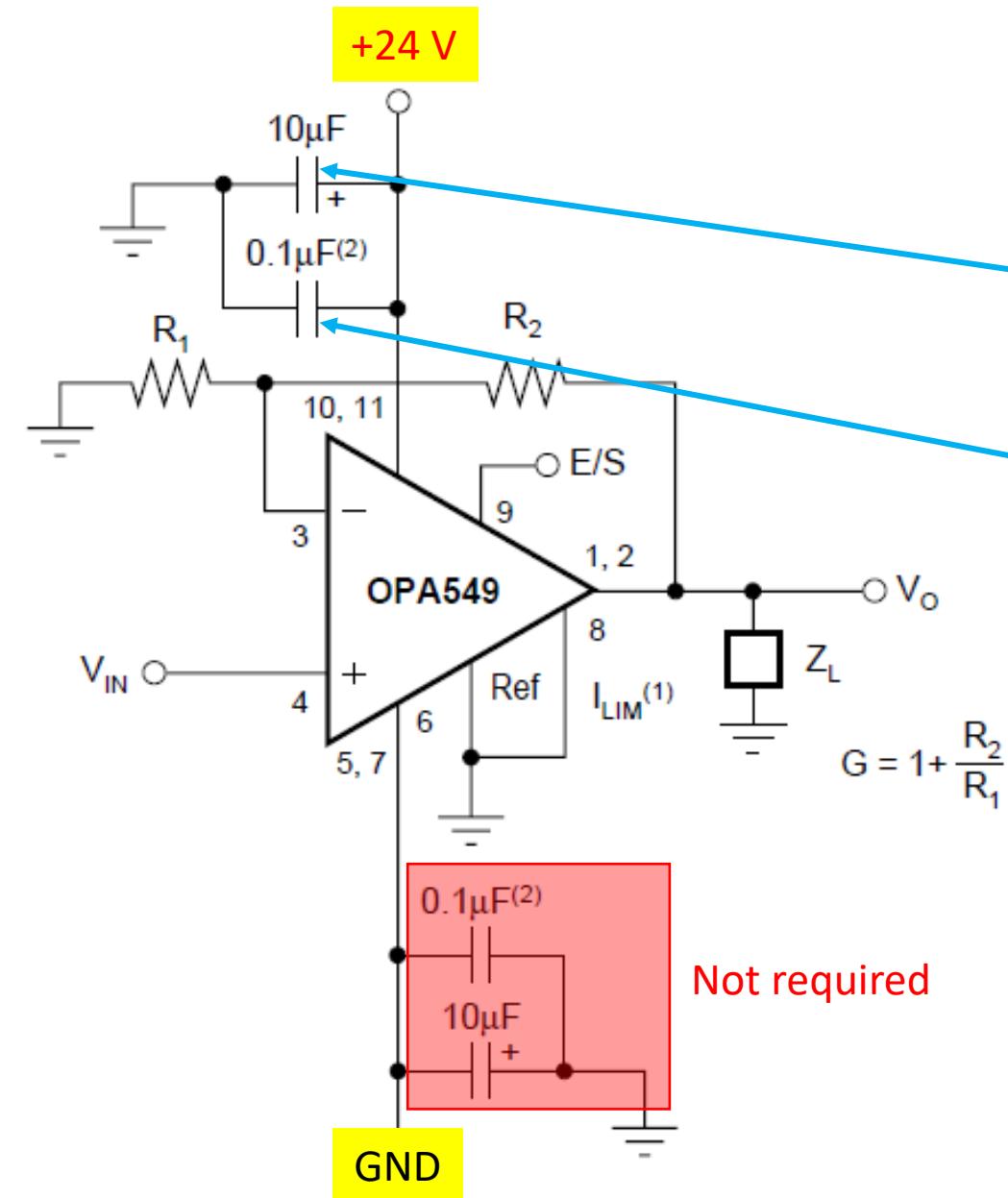


The ES pin 9 can be controlled from a GPIO, but this requires a simple switch using a npn transistor, for example [P2N2222A](#), the schematic of which we provide.

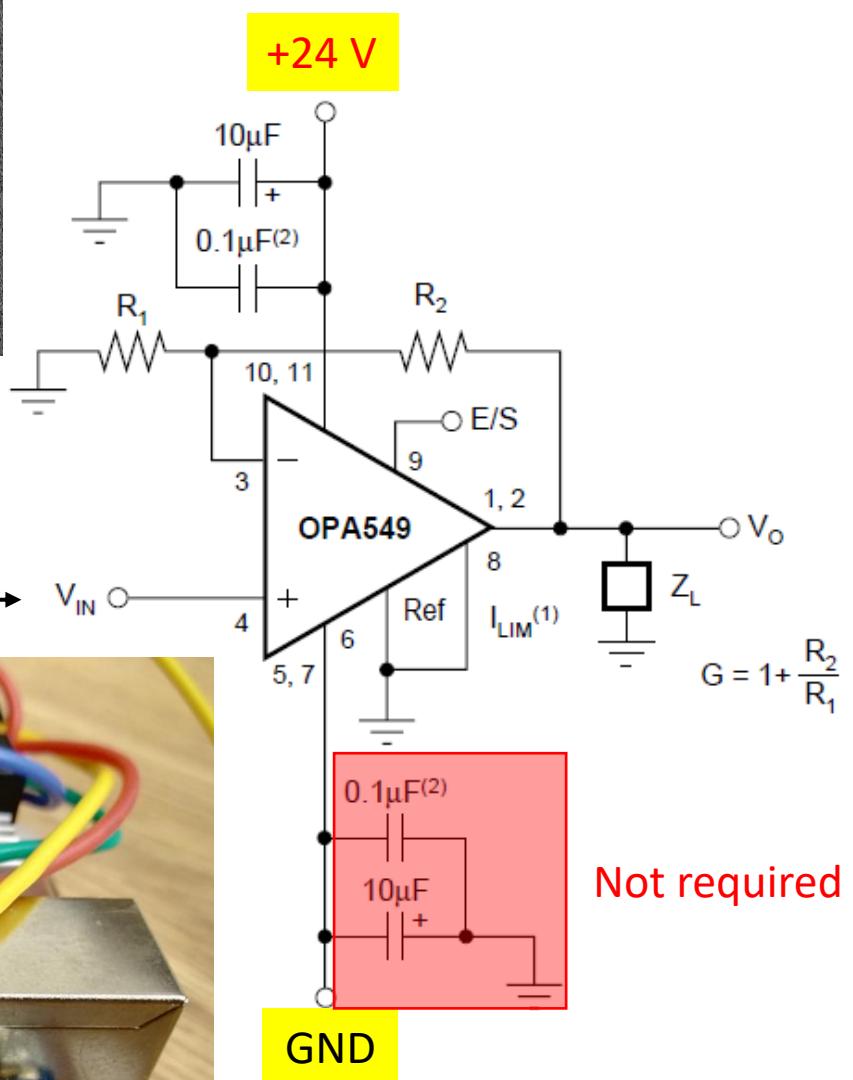
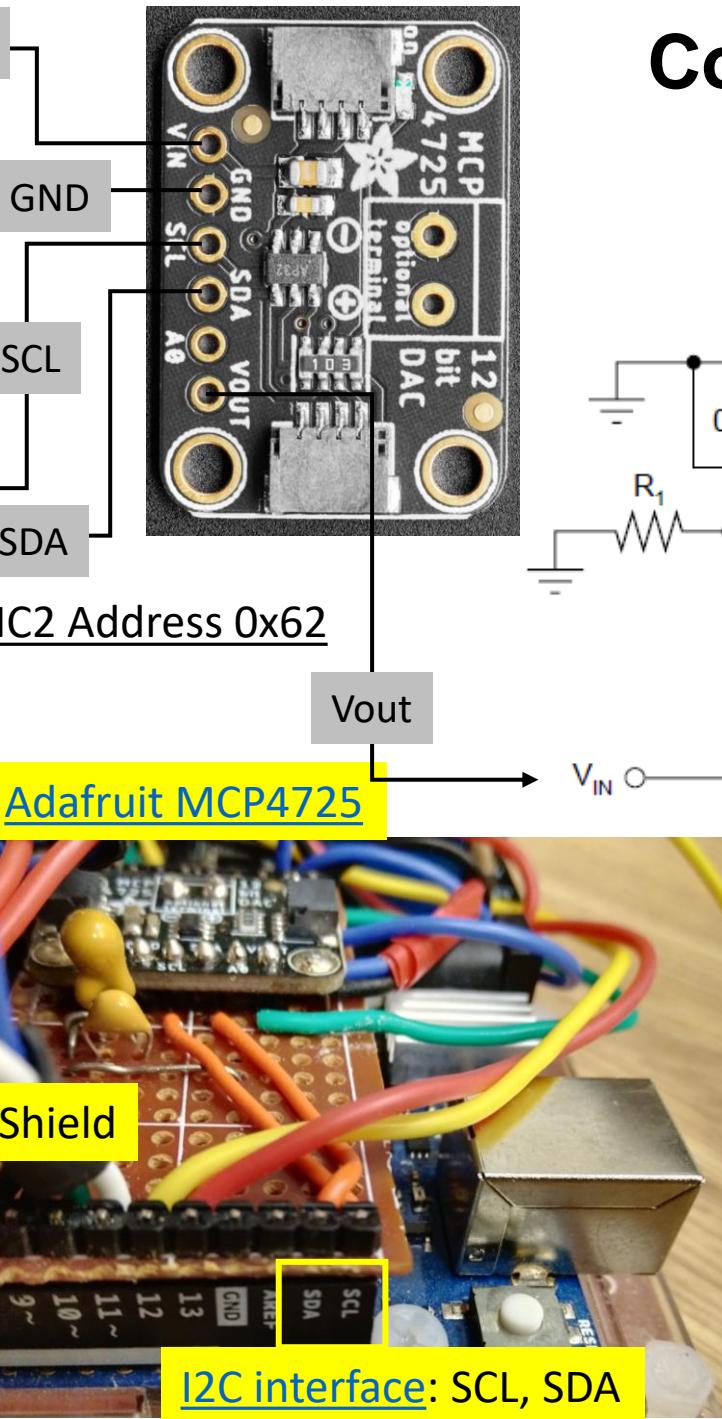
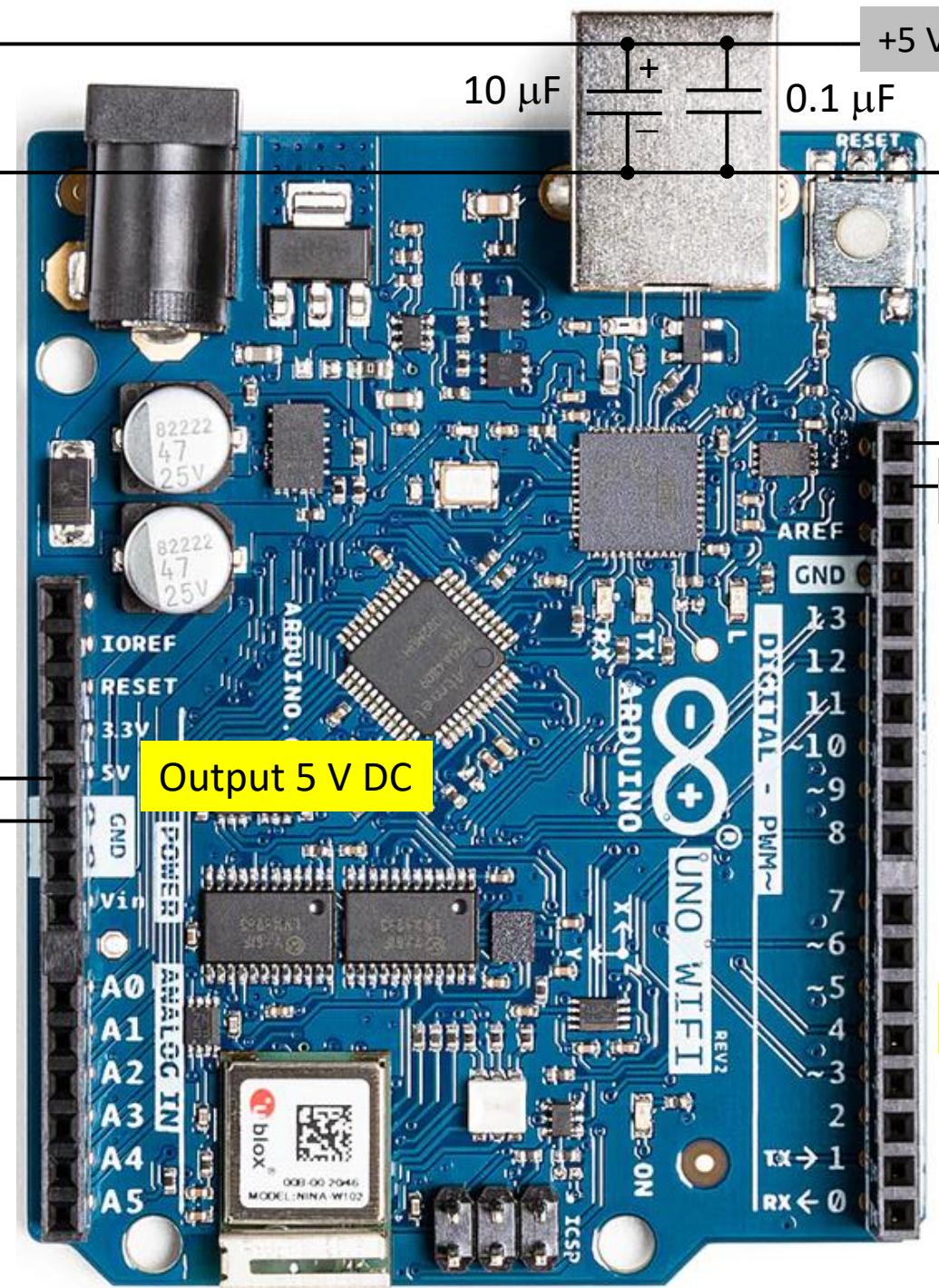
Resistor networks for the amplifier gain and the voltage divider



Source conditioning capacitors



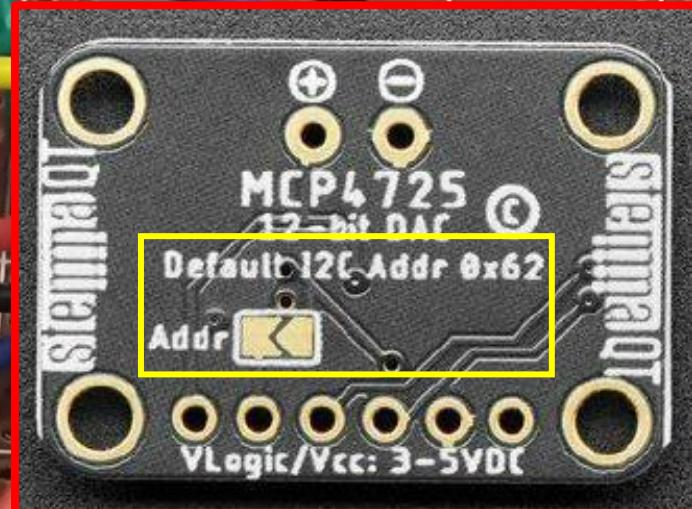
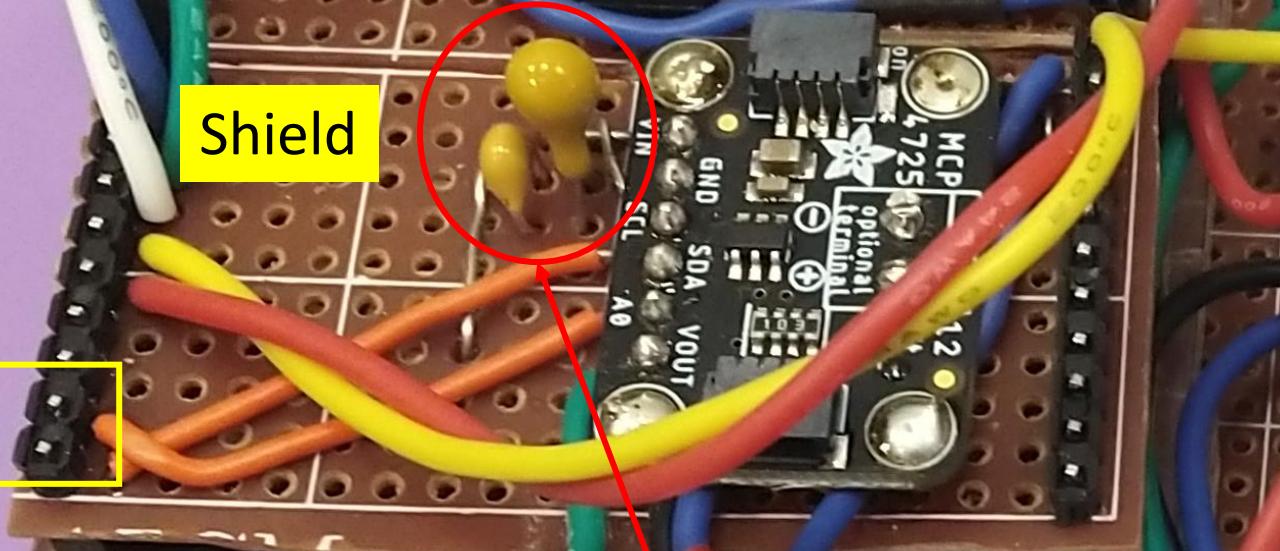
Connecting Adafruit MCP4725 DAC



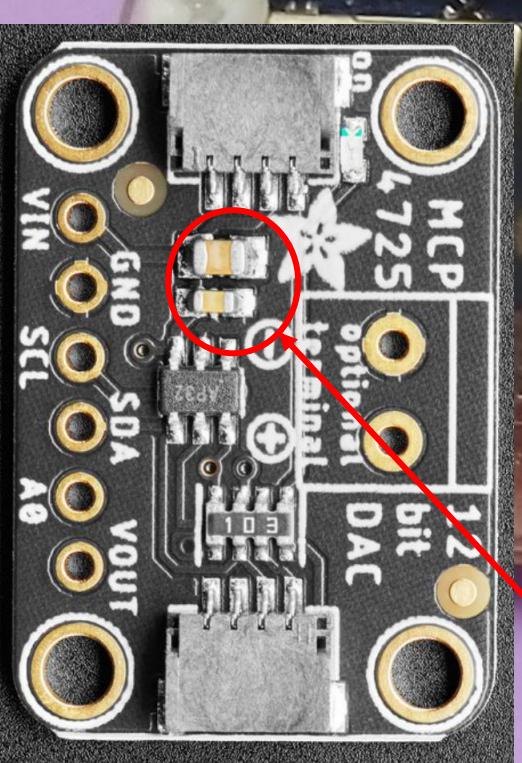
NOTES: (1) I_{LIM} connected to Ref gives the maximum current limit, 10A (peak). (2) Connect capacitors directly to package power-supply pins.

I2C interface

Shield

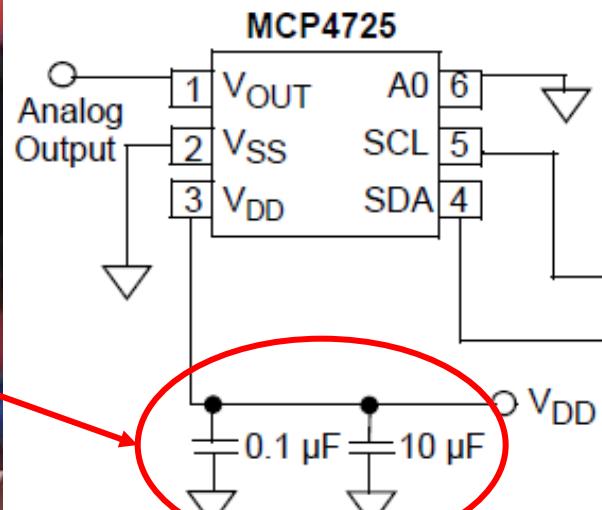


Using I2C with an Arduino Interface



The board already contains some capacitors

External ceramic and tantalum capacitors for V_{DD}



Note 1: R is the pull-up resistor. Typically $1 \sim 10 \text{ k}\Omega$

2: A0 can be tied to V_{SS} , V_{DD} or driven by MCU

Configuration of OPA549 and Arduino for the RED output

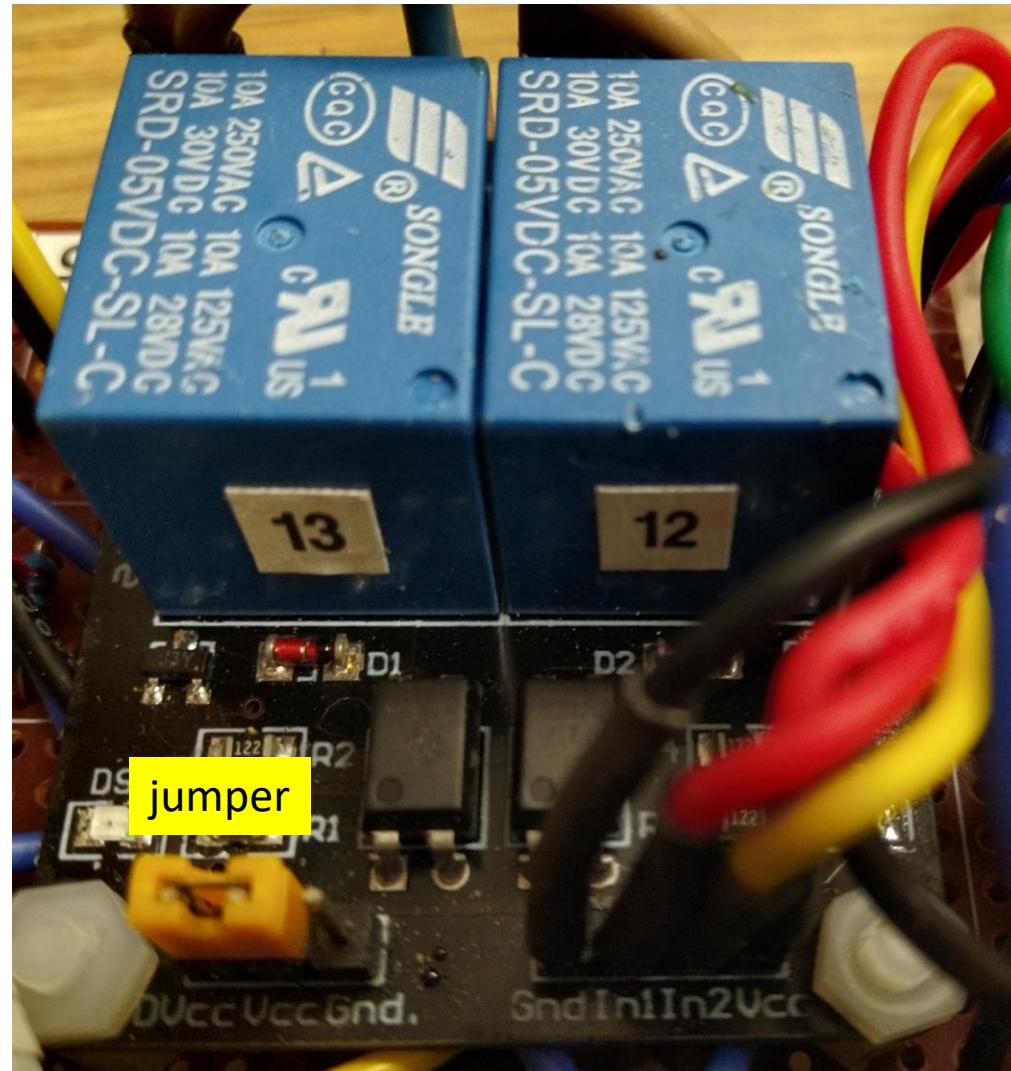
```
// Global constants
const int analogA0 = A0; // Analog pin A0 used for reading the output voltage from the DAC driver
const float max_v1 = 20.0; // Maximum voltage for channel 1
const float min_v1 = -20.0; // Minimum voltage for channel 1
const float op_amp_gain = 5.9567; // Gain of OPA549 for DAC output [0-5] V
const float A0_coeff = 5.4833; // A0 reading to the voltage output coefficient
const float bs_DAC = 1.5 * (5.0 / 1023.0) * A0_coeff; // bit sensitivity for DAC output
```

The [OPA549](#) operational amplifier ([op amp](#)) amp is configured as a non-inverting amplifier with a positive gain $G = 1 + R_2/R_1$. Both resistors $R_{1,2}$ are chosen within the **kilo-ohm** range. A Digital-to-Analog Converter ([DAC](#)), [Adafruit MCP4725](#), provides an output voltage ranging from 0 to 5 V with a 12-bit resolution. While the OPA549 is powered by a 24 V source, the actual output voltage will be slightly less than this value. We set the output voltage from the “RED” output in the range of $[-20,20]$ V, with polarity controlled by a relay-based [H-bridge](#). Consequently, the gain G must exceed 4. Using kilo-ohm resistor values from a standard resistor kit, we selected $G = 5.9567$. This precise value was determined through the calibration measurements shown in the next slide. To monitor the output voltage from the OPA549, [a voltage divider](#) was implemented using resistors $R_{3,4}$. To avoid introducing distortions in the circuit’s operation, these resistor values were chosen in the **hundreds of kilo-ohms** range. The ratio $R_4/(R_3 + R_4)$ must be close to $1/G$. The selected resistors yielded $R_4/(R_3 + R_4) = 1/5.4833$, with this value determined by the calibration measurements. The output of the voltage divider is fed into [a buffer](#) (voltage follower) to ensure electrical isolation of the analog input A0 on the Arduino from the rest of the circuit. The buffer is powered by the Arduino’s 5 V supply. Although the buffer’s gain is unity, it cannot output a full 5 V even for [a rail-to-rail performance](#). This limitation must be taken into account when choosing $R_{3,4}$.

Configuration of OPA549 and Arduino for the RED output

GPIO pins D8 through D13 are configured as digital outputs. D8 and D10 are used for LED indication of the positive polarity of the “RED” output, which is switched by two mechanical relays ([H-bridge](#)) controlled by D12 and D13. To limit the current through the LEDs, 220-ohm series resistors must be included in the circuits for D8 and D10. GPIO pins D9 and D11, which are not discussed here, are reserved for PWM control of the “BLACK” output and will be covered later.

```
void setup() {  
    // Initialize the pins as outputs  
    pinMode(8, OUTPUT); // LED "+" for the Red out channel  
    pinMode(10, OUTPUT); // LED "+" for the Red out channel  
    pinMode(12, OUTPUT); // Relay 2; H-bridge  
    pinMode(13, OUTPUT); // Relay 1; H-bridge  
  
    // Initialize DAC and the Red output channel  
    dac.begin(0x62); // I2C address (0x62) of DAC  
    dac.setVoltage(0, false); // DAC is OFF  
    digitalWrite(8, HIGH); // LED ON  
    digitalWrite(10, LOW); // LED OFF  
  
    // Relays controlling the polarity on the red output channel  
    digitalWrite(12, HIGH); // Relay 2 is OFF; activated by LOW  
    digitalWrite(13, HIGH); // Relay 1 is OFF; activated by LOW  
}
```



GND D13 D12 5 V

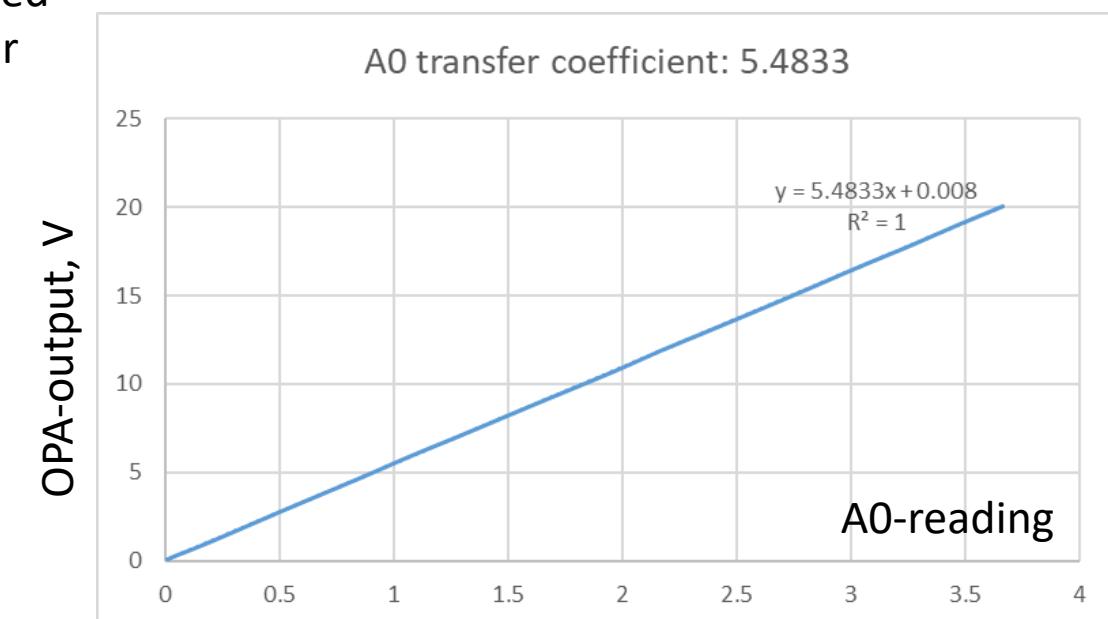
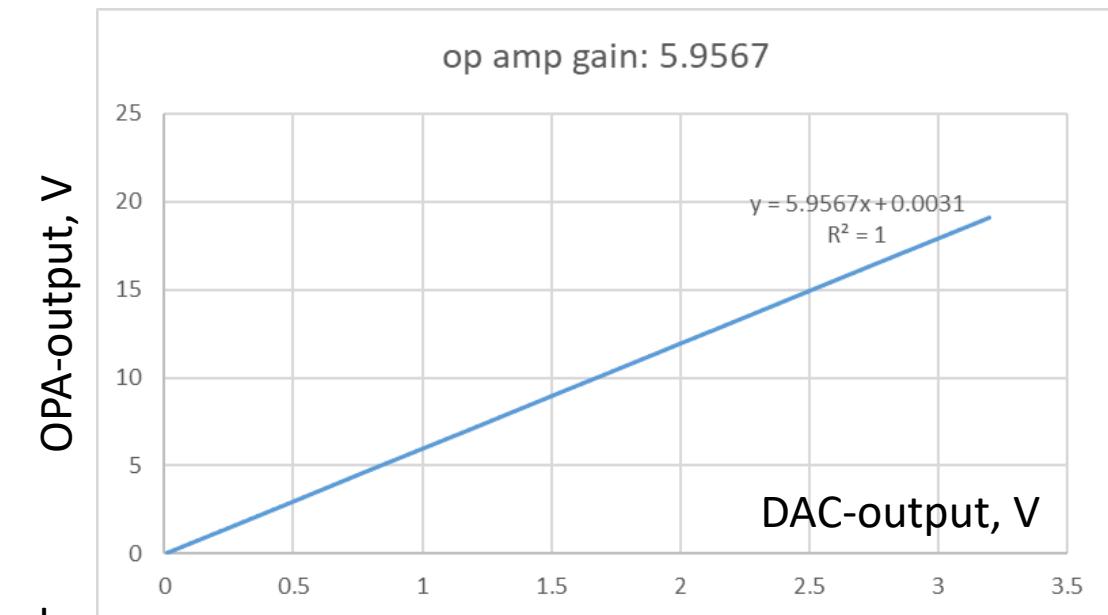
```

1. #include <Wire.h>
2. #include <Adafruit_MCP4725.h>
3.
4. Adafruit_MCP4725 dac;
5.
6. const int analogPin = A0;
7.
8. void setup() {
9.   Wire.begin();
10.  Serial.begin(9600);
11.
12.  dac.begin(0x62); // Replace with your actual I2C address (0x62)
13. }
14.
15. void loop() {
16.   // Manually set the desired voltage between 0 and 5 volts for the DAC
17.   float dacVoltage = 0.01;
18.
19.   // Convert DAC voltage to DAC value (0-4095)
20.   uint16_t dacValue = (dacVoltage / 5.0) * 4095;
21.
22.   // Set the DAC output
23.   dac.setVoltage(dacValue, false);
24.   delay(1000);
25.
26.   // Read the analog voltage from pin A0
27.   int analogValue = analogRead(analogPin);
28.   float analogVoltage = (analogValue * 5.0) / 1023.0;
29.   delay(1000);
30.
31.   Serial.print("Setting voltage on DAC to: ");
32.   Serial.print(dacVoltage, 2);
33.   Serial.println(" V");
34.
35.   Serial.print("Analog voltage: ");
36.   Serial.print(analogVoltage, 2);
37.   Serial.println(" V");
38.
39.   delay(1000); // Wait for 10 seconds before updating again
40. }
```

Arduino sketch

OPA-output was measured using a precise voltmeter

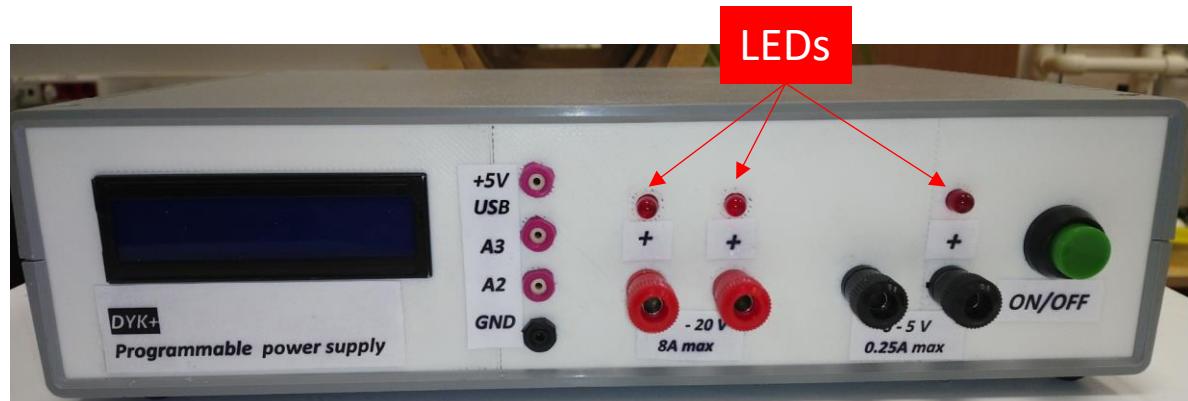
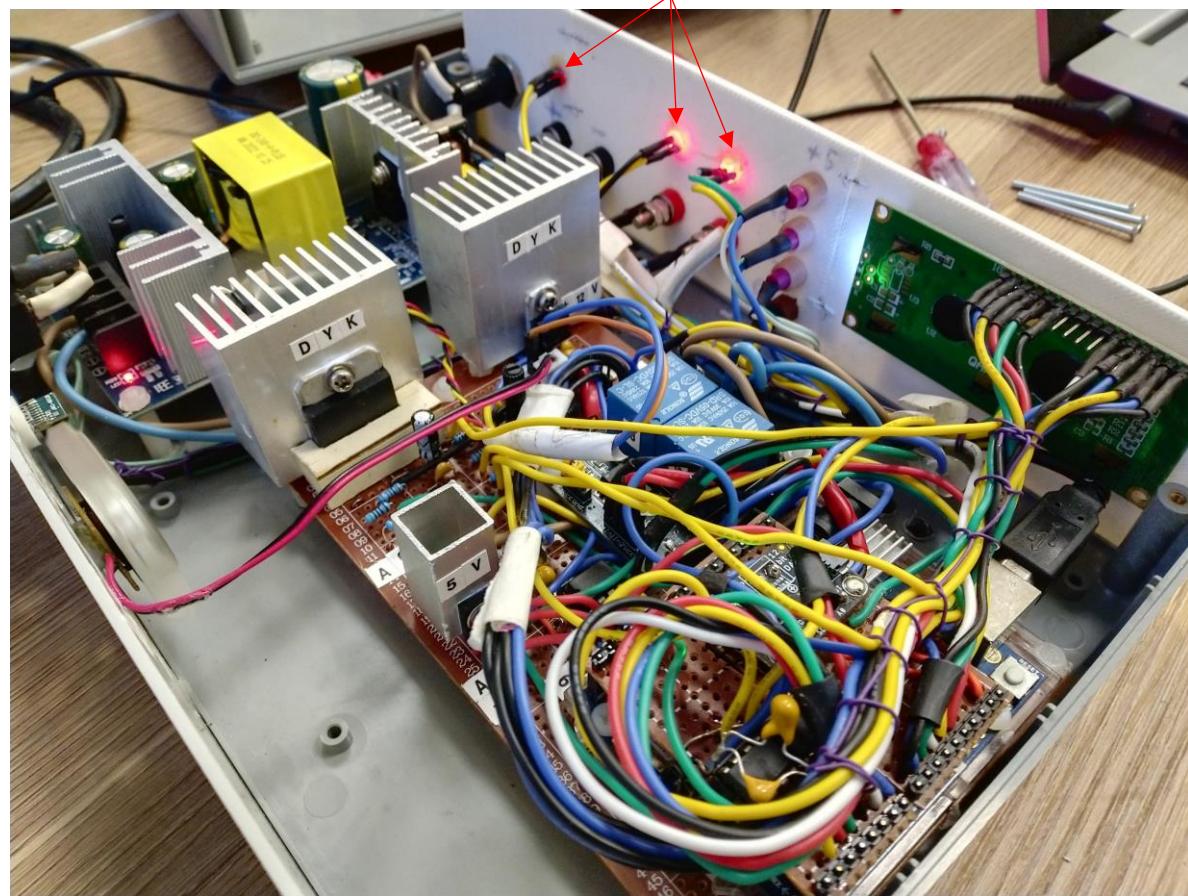
Testing and calibrating DAC-OPA



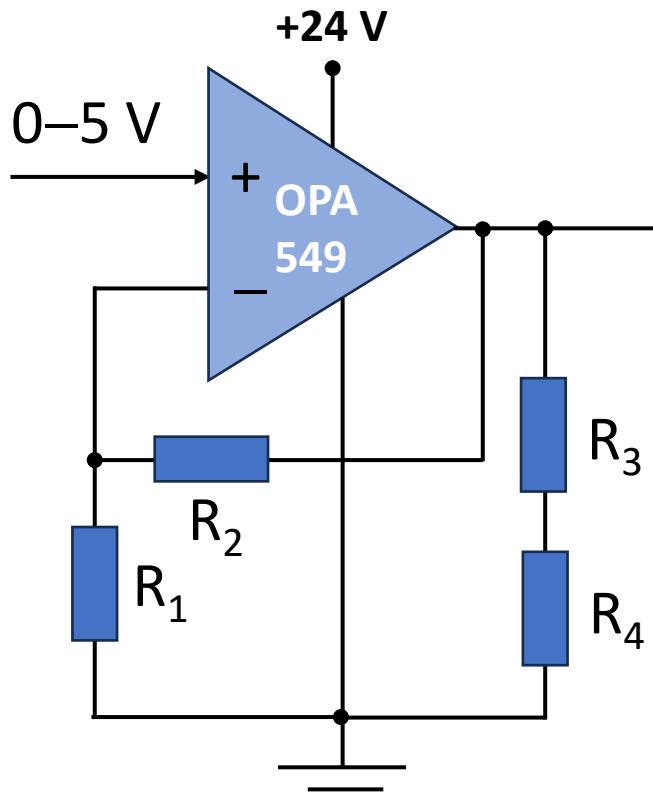
Testing LEDs

```
1. // Define the LED pins
2. const int ledPin1 = 8;
3. const int ledPin2 = 10;
4. const int ledPin3 = 11;
5.
6. void setup() {
7.   // Initialize the LED pins as OUTPUT
8.   pinMode(ledPin1, OUTPUT);
9.   pinMode(ledPin2, OUTPUT);
10.  pinMode(ledPin3, OUTPUT);
11. }
12.
13. void loop() {
14.   // Turn on the LEDs
15.   digitalWrite(ledPin1, HIGH);
16.   digitalWrite(ledPin2, HIGH);
17.   digitalWrite(ledPin3, HIGH);
18.
19.   // Wait for a period
20.   delay(1000); // Adjust the delay time (in milliseconds) as needed
21.
22.   // Turn off the LEDs
23.   digitalWrite(ledPin1, LOW);
24.   digitalWrite(ledPin2, LOW);
25.   digitalWrite(ledPin3, LOW);
26.
27.   // Wait for another period
28.   delay(1000); // Adjust the delay time (in milliseconds) as needed
29. }
```

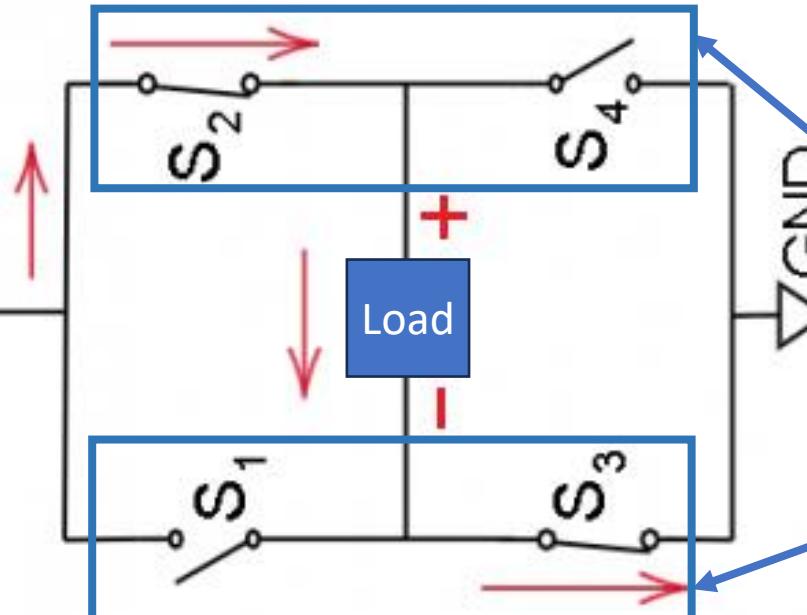
Arduino sketch



Relay H-bridge: operation principle

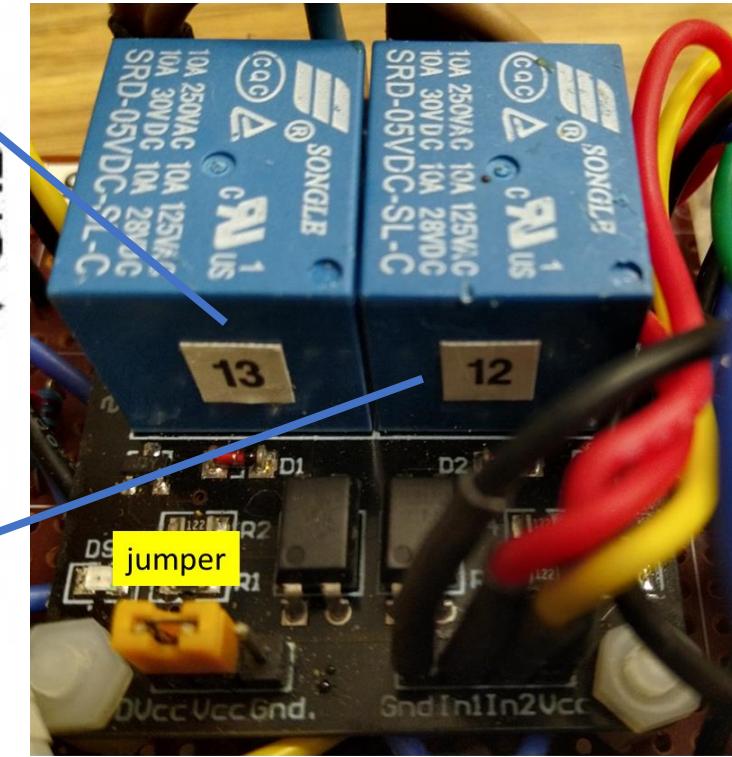
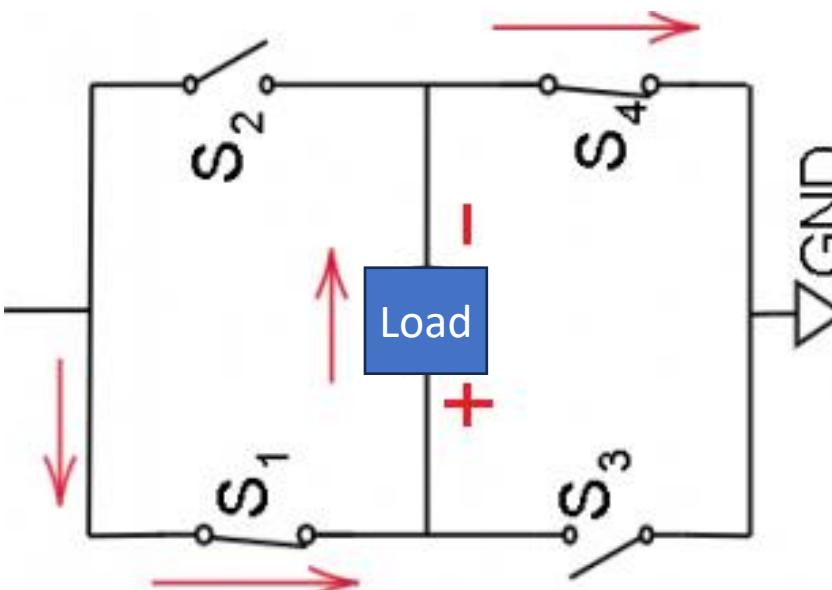


Normally closed Normally opened



Normally opened

Normally closed

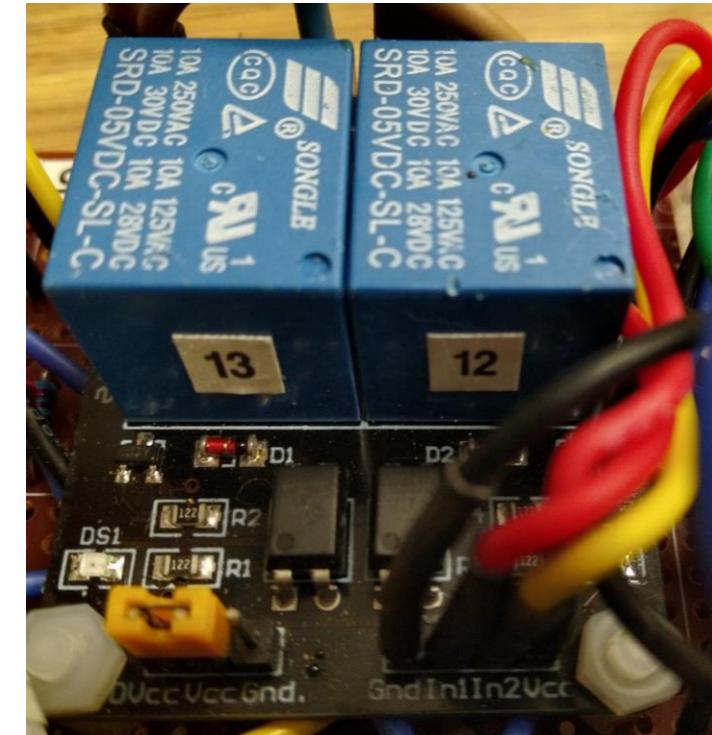


GND D13 D12 5 V

The uploaded Arduino sketch always monitors the voltage value before switching polarity. If the previous value was non-zero, the output will be set to zero first before applying the new polarity.

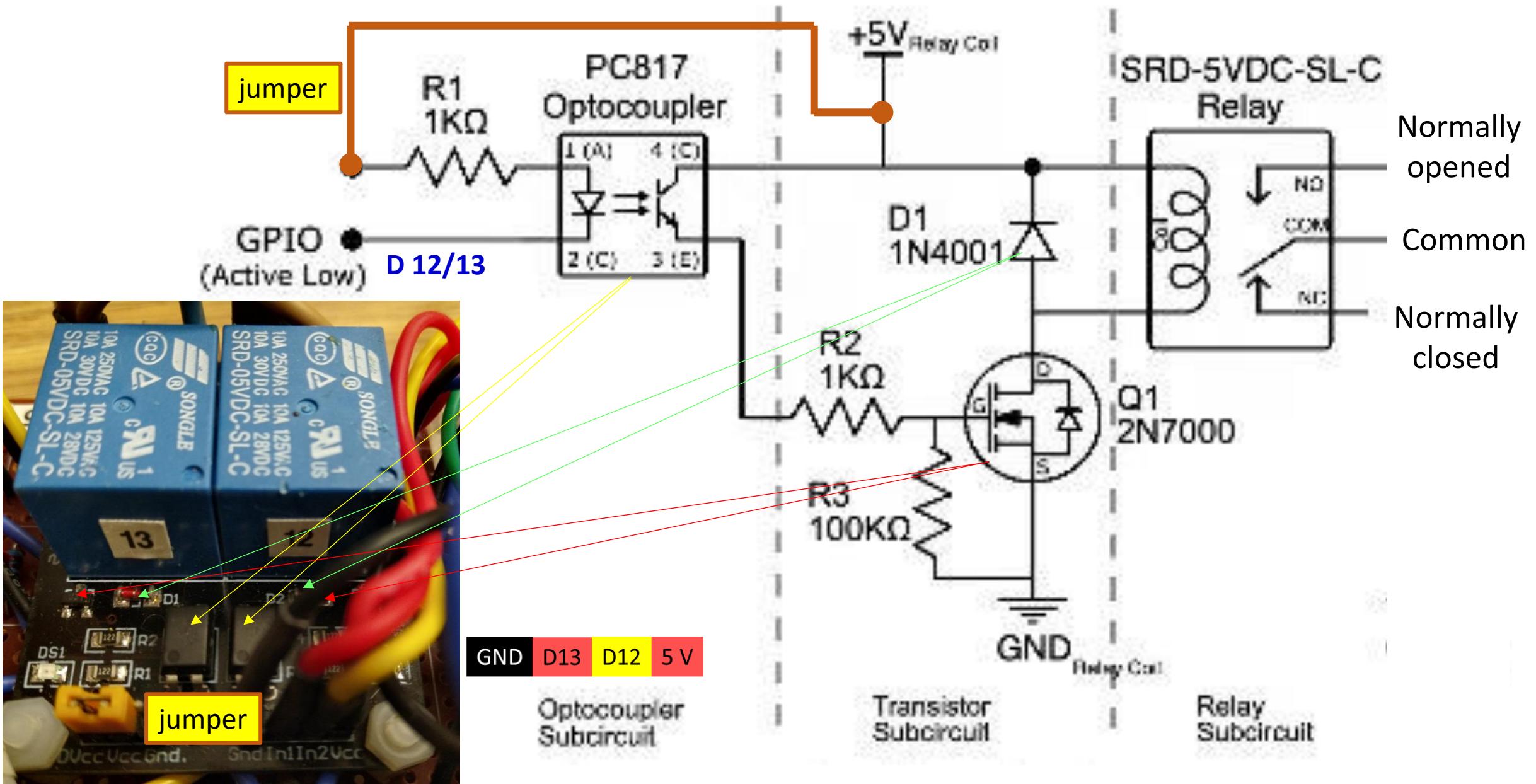
Control of the voltage polarity in the Arduino sketch for switching the H-bridge

```
if (chan1 == "on") {  
    sign = signbit(v1); //1 if '-' and 0 if '+'  
    if (sign != previous_sign) {  
        dac.setVoltage(0, false);  
        previous_sign = sign;  
        if (previous_pin12_13 == "HIGH") {  
            digitalWrite(8, LOW);  
            digitalWrite(10, HIGH);  
            digitalWrite(12, LOW);  
            digitalWrite(13, LOW);  
            previous_pin12_13 = "LOW";  
        } else {  
            digitalWrite(8, HIGH);  
            digitalWrite(10, LOW);  
            digitalWrite(12, HIGH);  
            digitalWrite(13, HIGH);  
            previous_pin12_13 = "HIGH";  
        }  
    }  
}
```

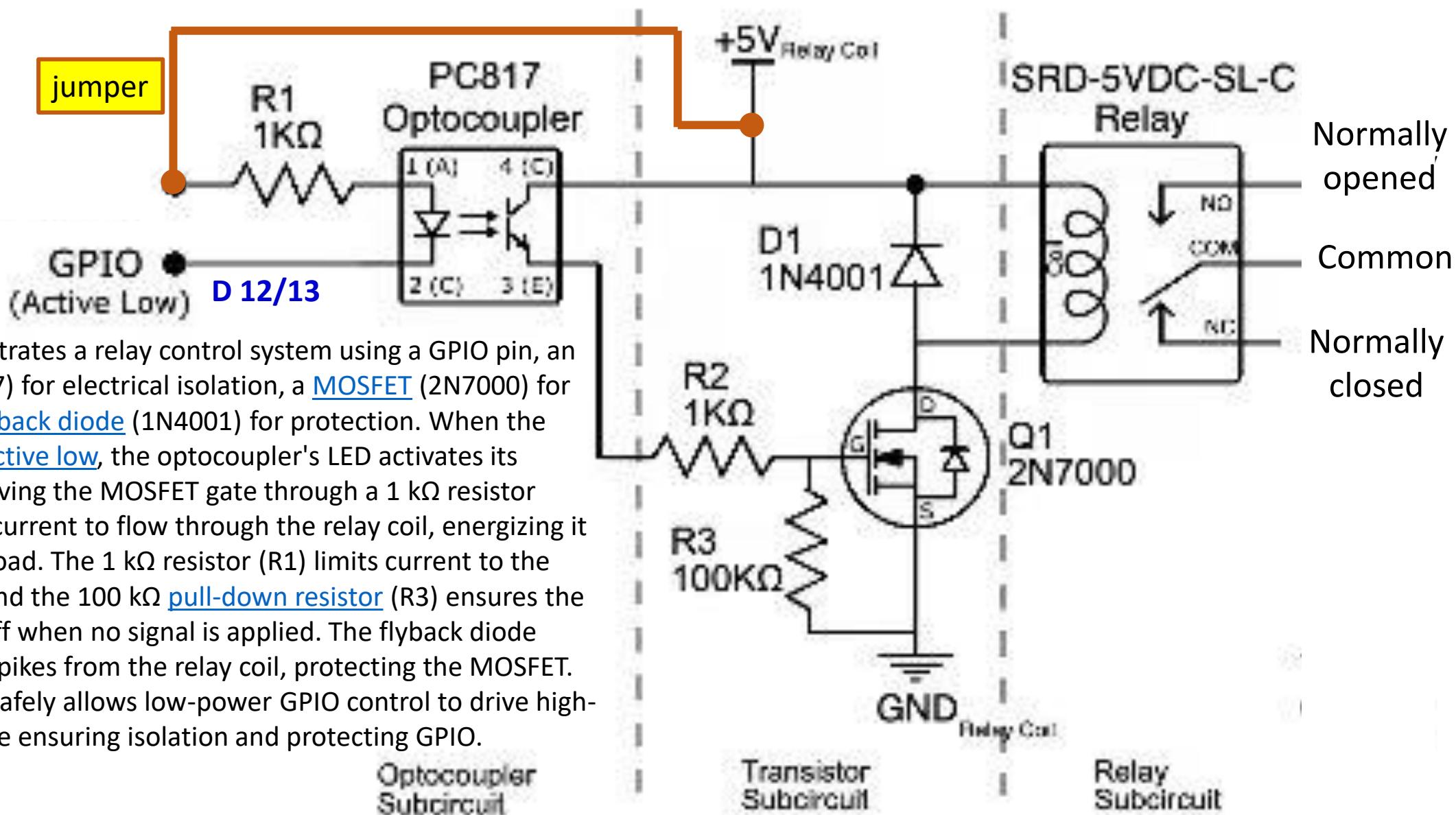


When setting a new voltage value on the Red output, its polarity is checked relative to the previous sign (+/-). If the polarities differ and the new voltage value is non-zero, the output is first reset to zero before switching polarity and applying the new value. This precaution is taken to prevent sparking on the relay and to protect the OPA output.

Relay H-bridge: electronic circuit for each relay



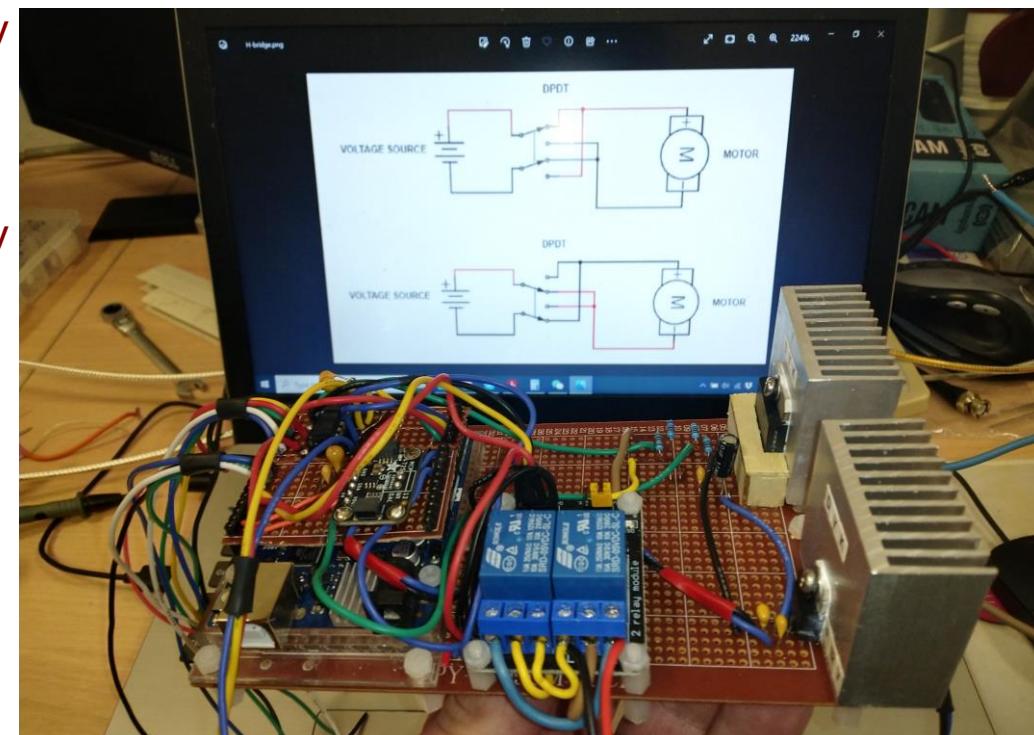
Relay H-bridge: electronic circuit for each relay



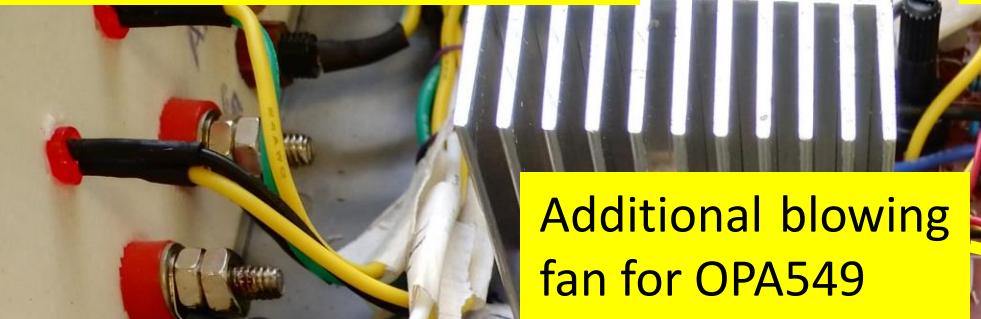
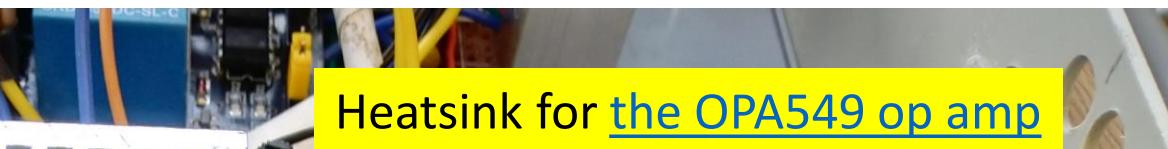
Testing the relays

```
1. const int relayPin1 = 12; // Pin connected to the first relay
2. const int relayPin2 = 13; // Pin connected to the second relay
3.
4. void setup() {
5.   pinMode(relayPin1, OUTPUT); // Set relayPin1 as an output
6.   pinMode(relayPin2, OUTPUT); // Set relayPin2 as an output
7. }
8.
9. void loop() {
10.   digitalWrite(relayPin1, HIGH); // Turn on the first relay
11.   digitalWrite(relayPin2, HIGH); // Turn on the second relay
12.   delay(3000); // Wait for 2 seconds
13.
14.   digitalWrite(relayPin1, LOW); // Turn off the first relay
15.   digitalWrite(relayPin2, LOW); // Turn off the second relay
16.   delay(3000); // Wait for 2 seconds
17. }
```

Arduino sketch



Heatsink for L7812 12 V DC
voltage regulator



Power management

+24 V for OPA549

+12 V for Arduino

+5 V for:

- low-power op amps
- DAC
- relays (H-bridge)

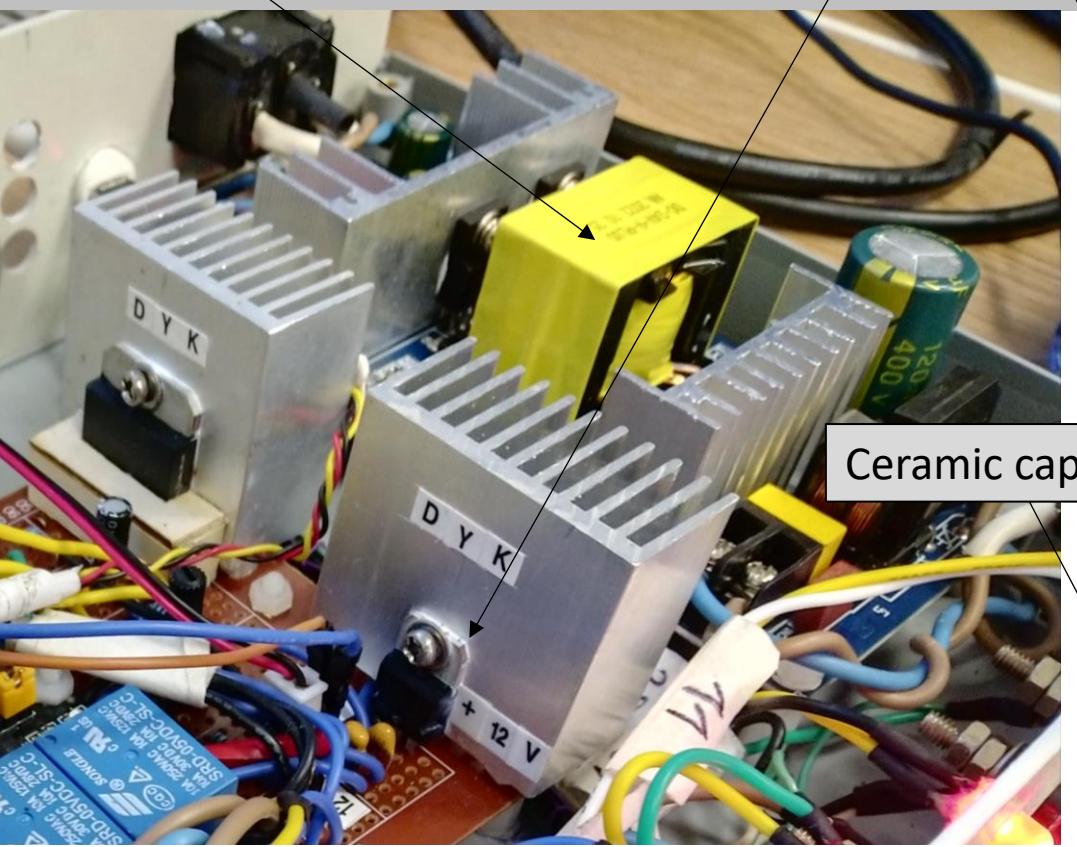
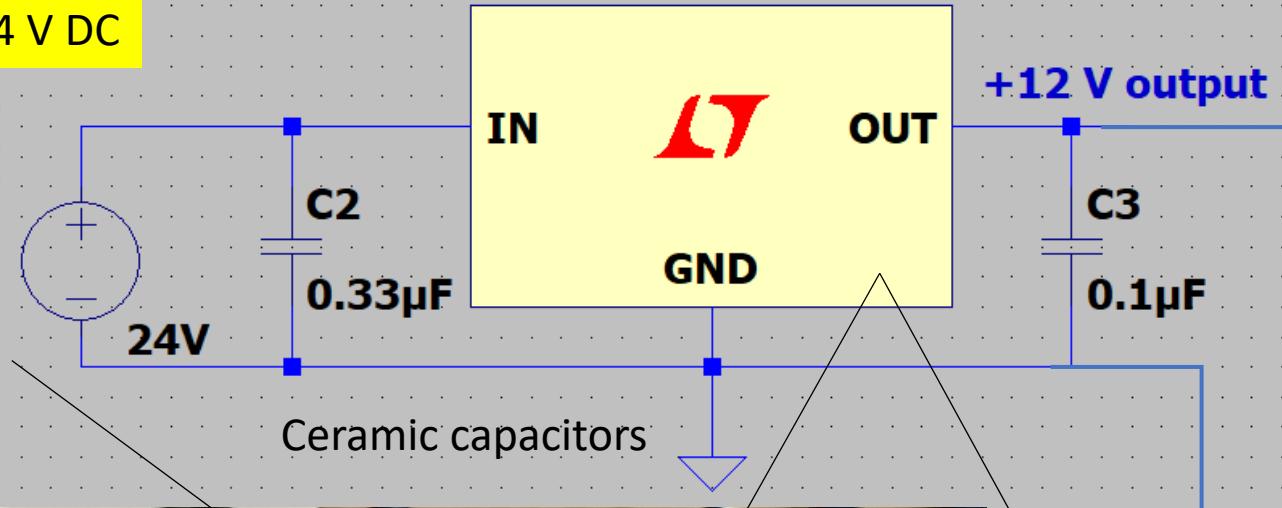


24 V DC, 10 A, 240 W power supply

Power_supply

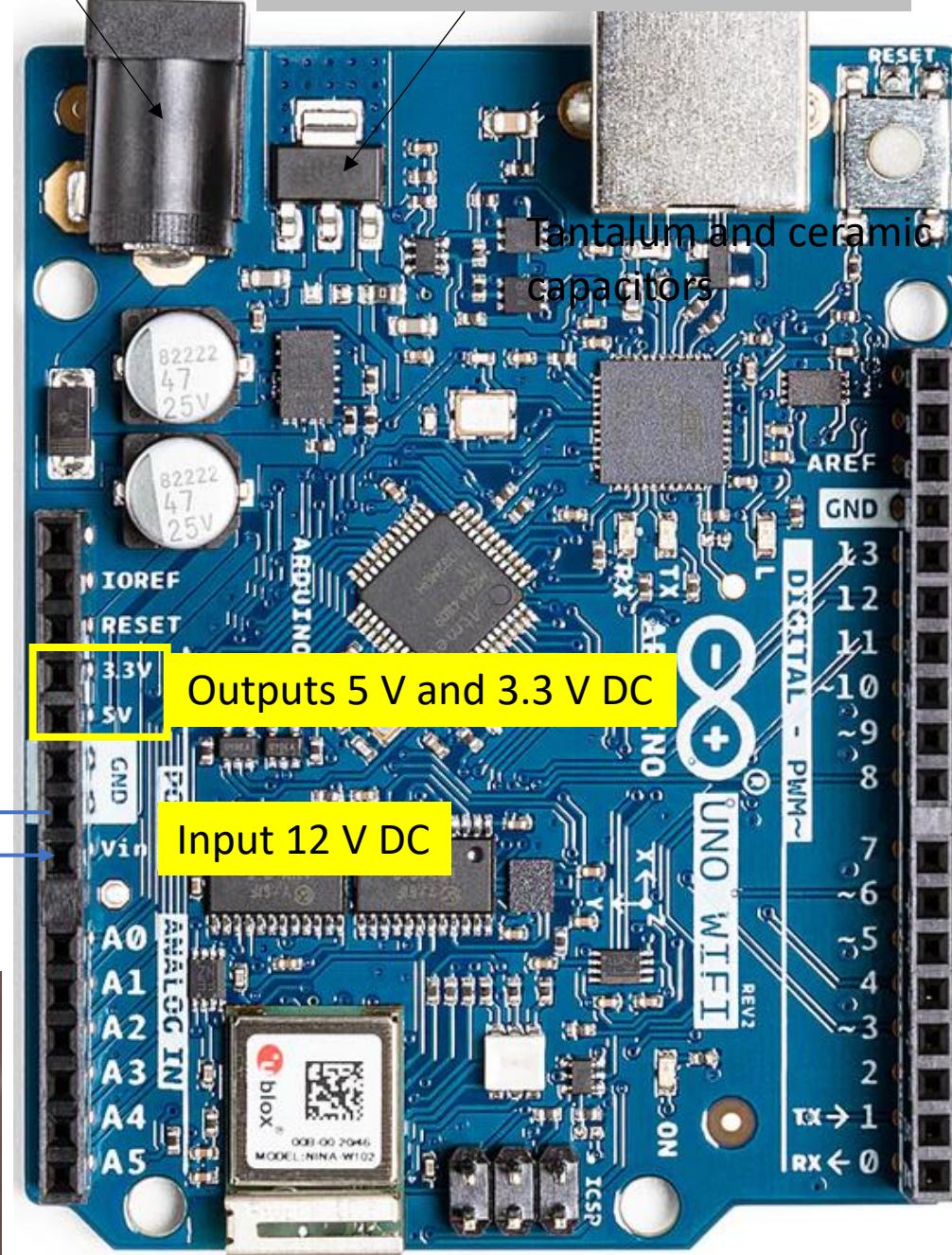
24 V DC

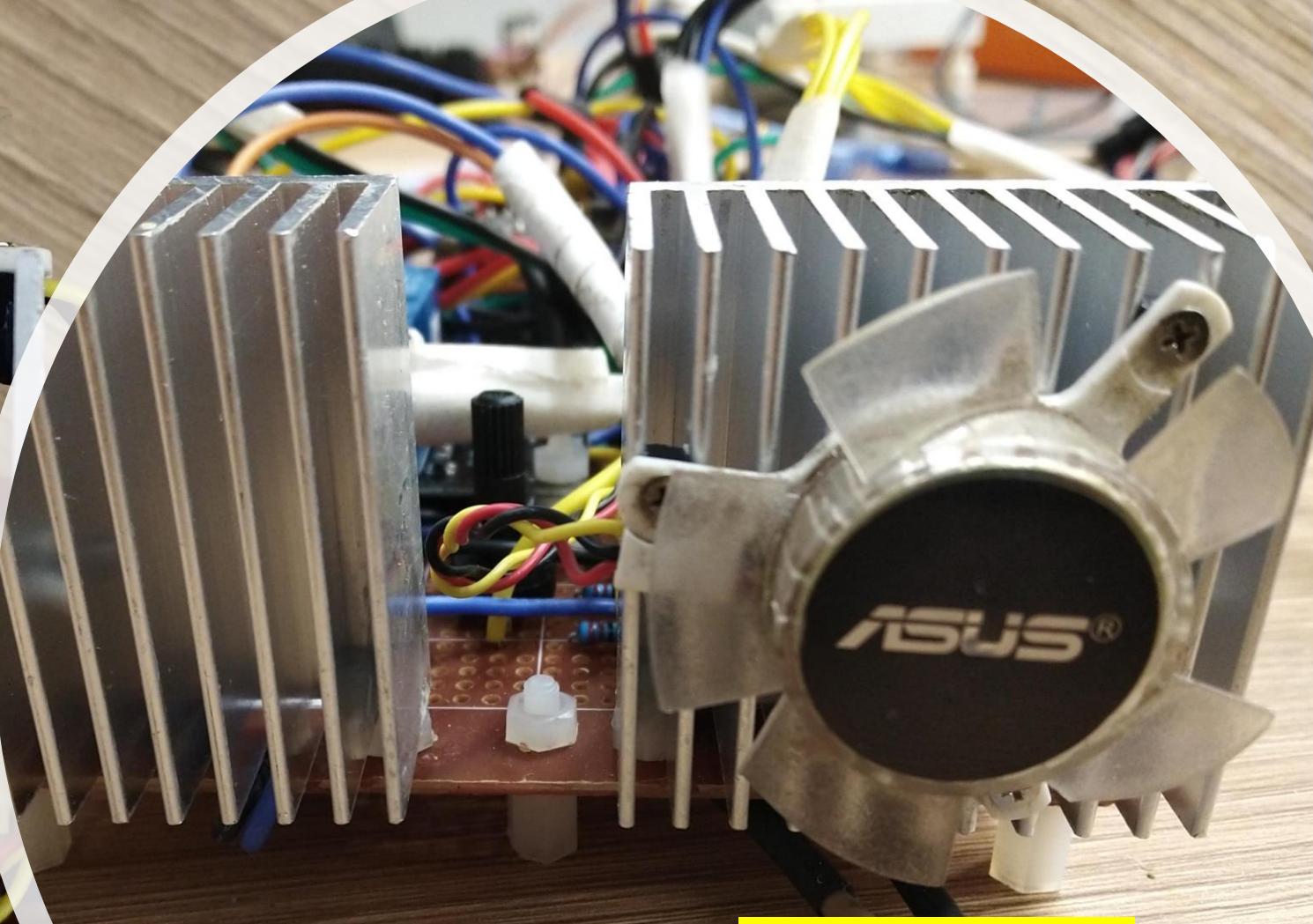
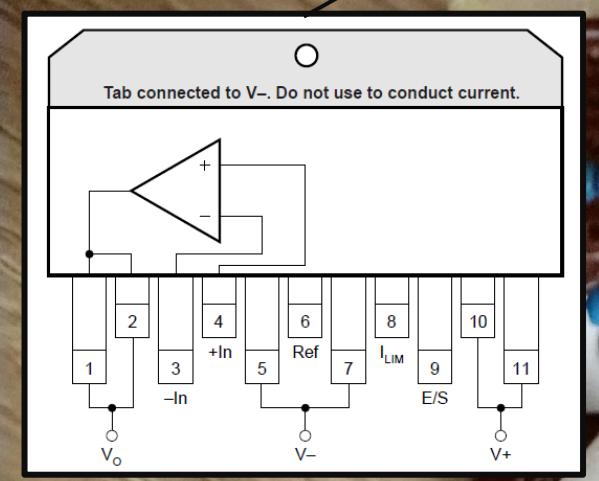
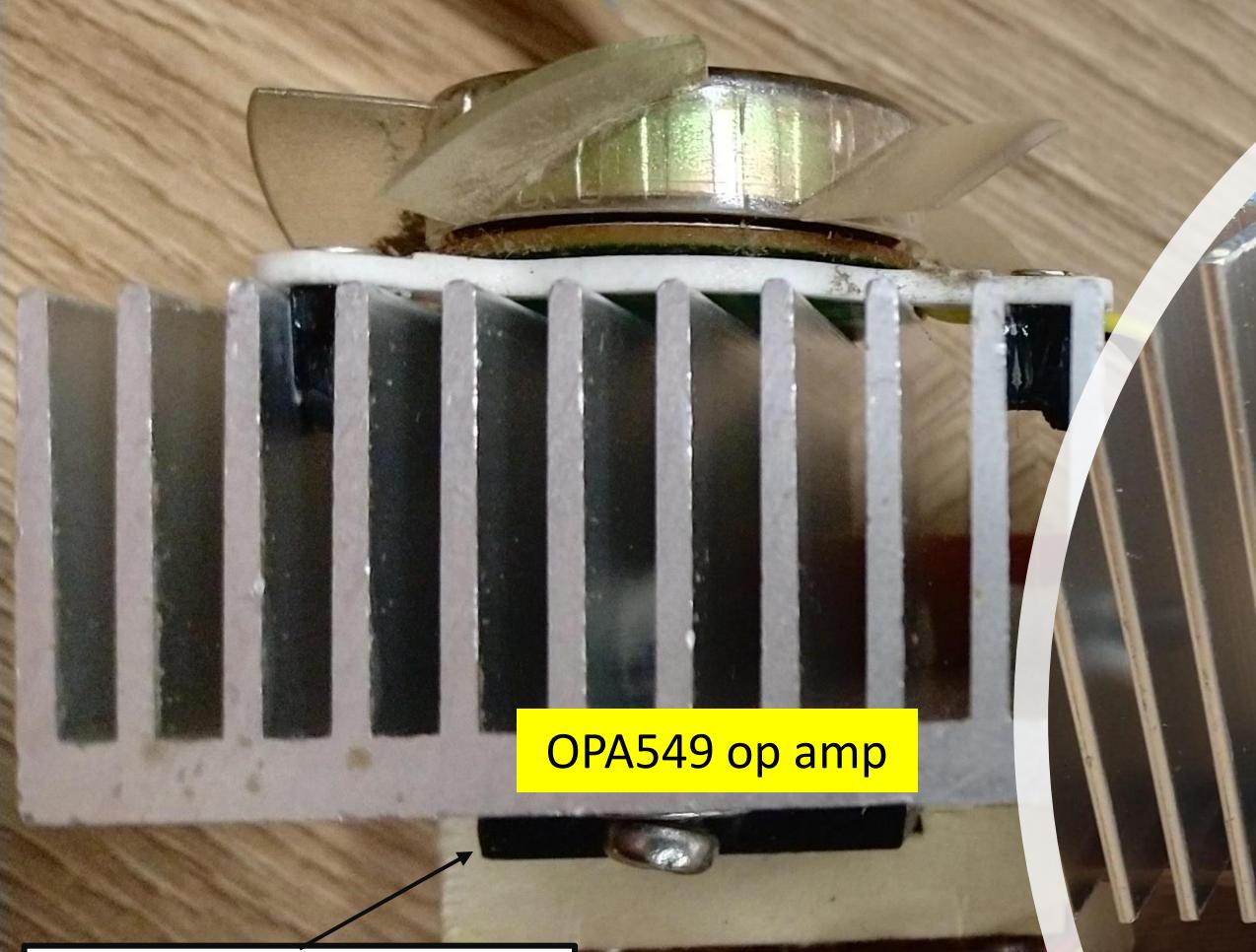
L7812_Voltage_Regulator

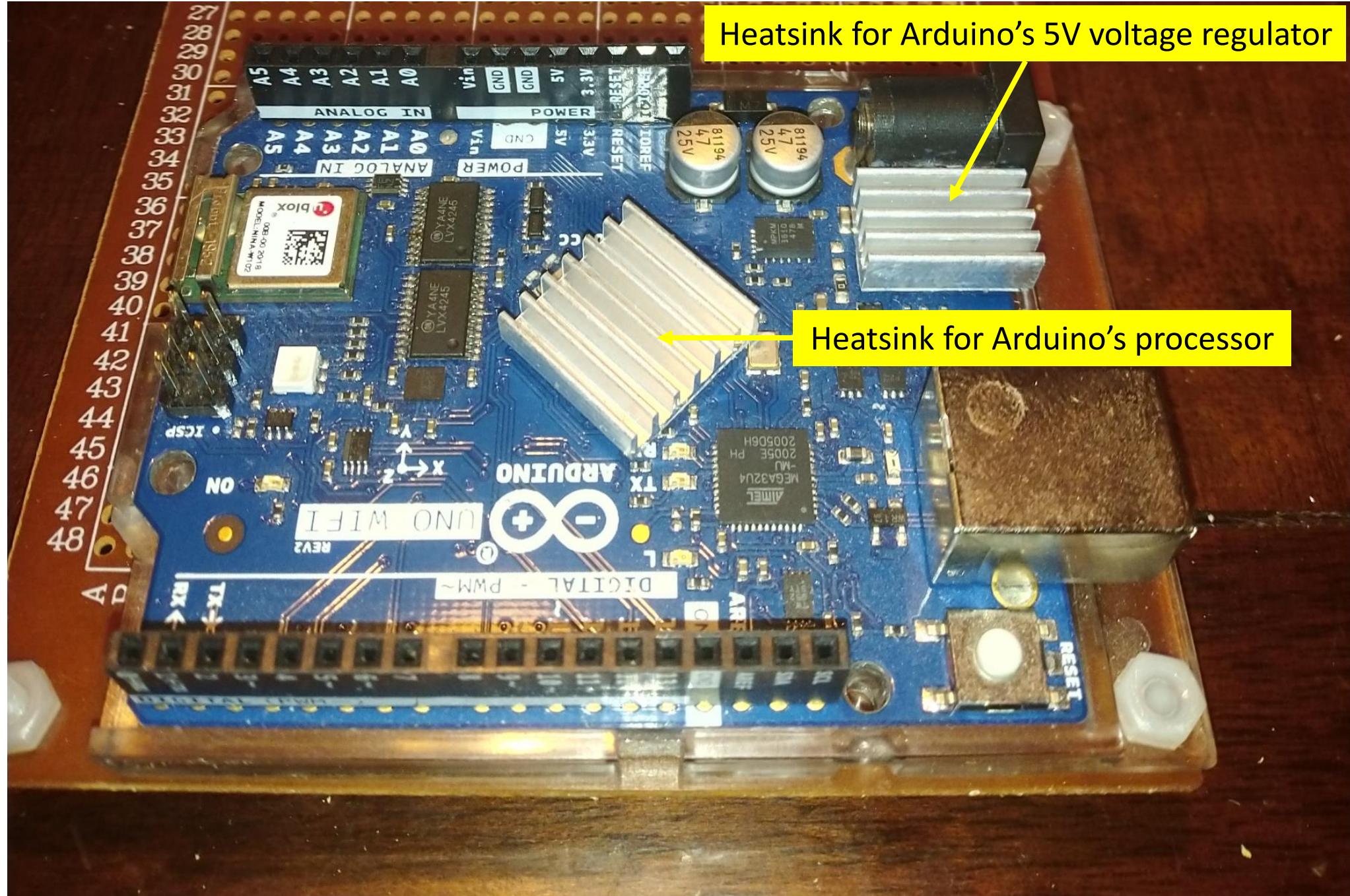


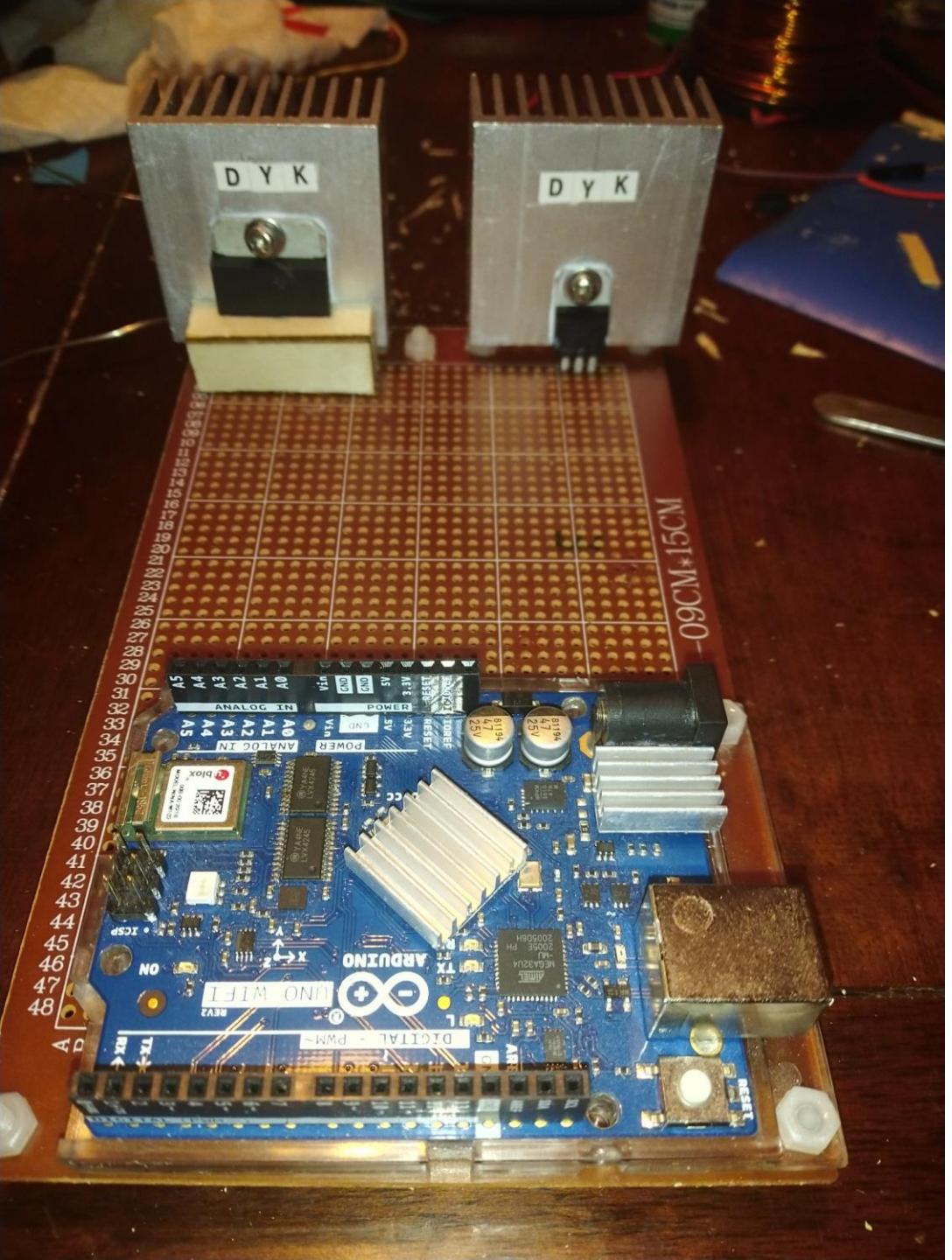
$9 \text{ V} < \text{Vin} < 12 \text{ V}$

Voltage Regulator 5 V (reference)

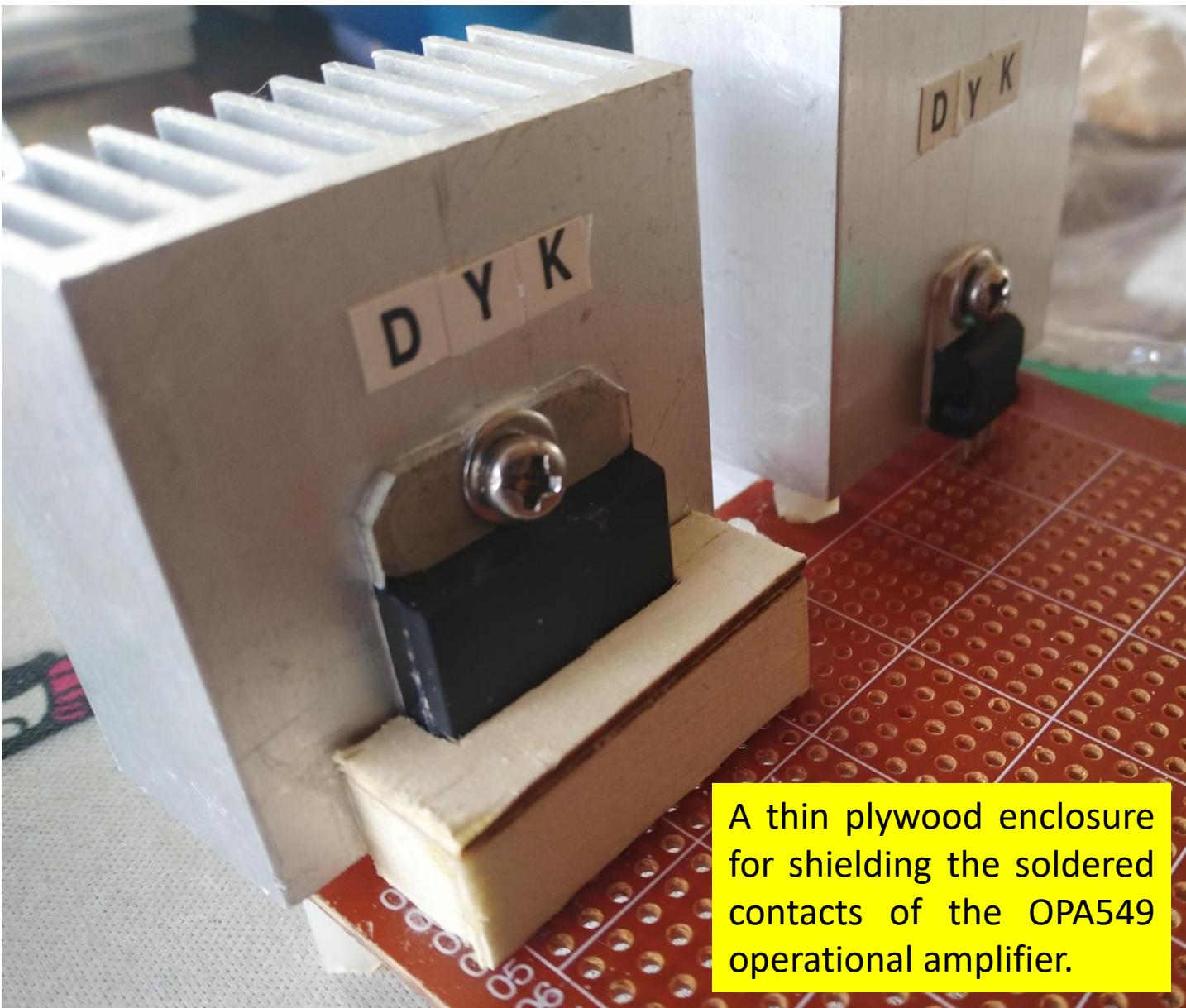








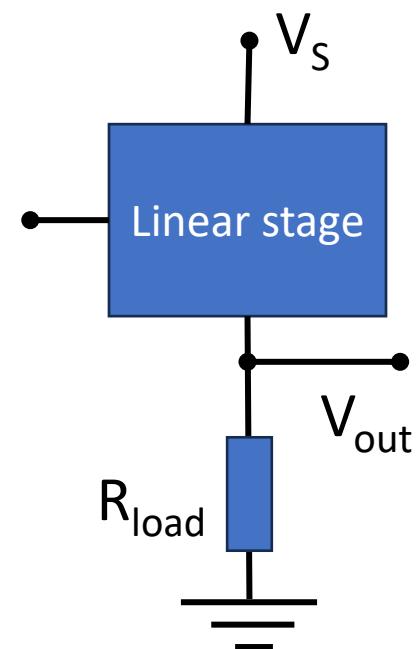
Heatsinks on OPA, 12 V voltage regulator, and Arduino



A thin plywood enclosure for shielding the soldered contacts of the OPA549 operational amplifier.

Heat management: maximum power dissipation

The power P dissipated in a linear stage is determined by the product of the voltage drop ($V_S - V_{out}$) across the component (transistor, voltage regulator, or op amp) and the current $I_{load} = V_{out}/R_{load}$ passing through it: $P = (V_S - V_{out}) \times I_{load} = (V_S - V_{out}) \times V_{out}/R_{load}$. This is a quadratic function by V_{out} that reaches its maximum at $V_{out} = V_S/2$: $P_{max} = V_S I_{load}/2 = V_S^2/(4R_{load})$. If the voltage drop is substantial, the maximum permissible power dissipation may be exceeded, even within the allowable current limits. For our applications, a current of 3 A through the Helmholtz coil would be sufficient to fully saturate MI samples (ferromagnetic wires or thin films) when scanning across the field points. Since the source voltage of the OPA549 is $V_S = 24$ V, the maximum power dissipated by it would be 36 W. At a maximum current of 8 A, the dissipated power would reach 96 W. This clearly shows that linear stages are quite inefficient in terms of energy consumption. However, their advantage lies in the rapid adjustment of voltage levels and low ripple noise, which are essential for our applications.



Heat management: thermal equation and heat transfer

The junction temperature T_J ($^{\circ}\text{C}$) in a semiconductor device can be determined according to the thermal equation: $T_J = T_A + P_D \times \theta_{JA}$, where $\theta_{JA} = \theta_{JC} + \theta_{CH} + \theta_{HA}$ is the junction-to-air thermal resistance ($^{\circ}\text{C}/\text{W}$), T_A is the ambient (room) temperature ($^{\circ}\text{C}$), P_D is the dissipated power (W), θ_{JC} is the junction-to-case thermal resistance ($^{\circ}\text{C}/\text{W}$), θ_{CH} is the case-to-heatsink thermal resistance ($^{\circ}\text{C}/\text{W}$), and θ_{HA} is the heatsink-to-air thermal resistance ($^{\circ}\text{C}/\text{W}$). In the absence of a heatsink, we would have: $\theta_{JA} = \theta_{JC} + \theta_{CA}$, where θ_{CA} is the case-to-air thermal resistance ($^{\circ}\text{C}/\text{W}$). For example, for a [P2N2222A](#) transistor in a plastic case, the thermal resistance θ_{JA} is $200\ ^{\circ}\text{C}/\text{W}$. Such a high value of θ_{JA} does not allow the transistor to dissipate more than $0.5\ \text{W}$ of power for $T_J = 125\ ^{\circ}\text{C}$ and $T_A = 25\ ^{\circ}\text{C}$. The solution lies in using components that allow for the attachment of a heatsink resulting in a much smaller $\theta_{CH} + \theta_{HA}$. The coefficient θ_{CH} can usually be neglected if a special [thermal grease](#) is applied between the casing and the heatsink. The coefficient θ_{HA} decreases when increasing the volume and surface area of the heatsink, optimizing its geometry to enhance natural air convection, as well as by adding airflow from an auxiliary fan, as explained on the next slide.

Understanding fan characteristics: CFM (cubic feet per minute) and LFM (linear feet per minute)

FEATURES

- dual ball bearing system
- 120 x 120 mm frame
- multiple speed options
- tachometer signal available

Online calculator

Air Flow: 169.5 CFM

Rectangle Duct H: _____ W: _____ in Circular Duct R: 60 mm

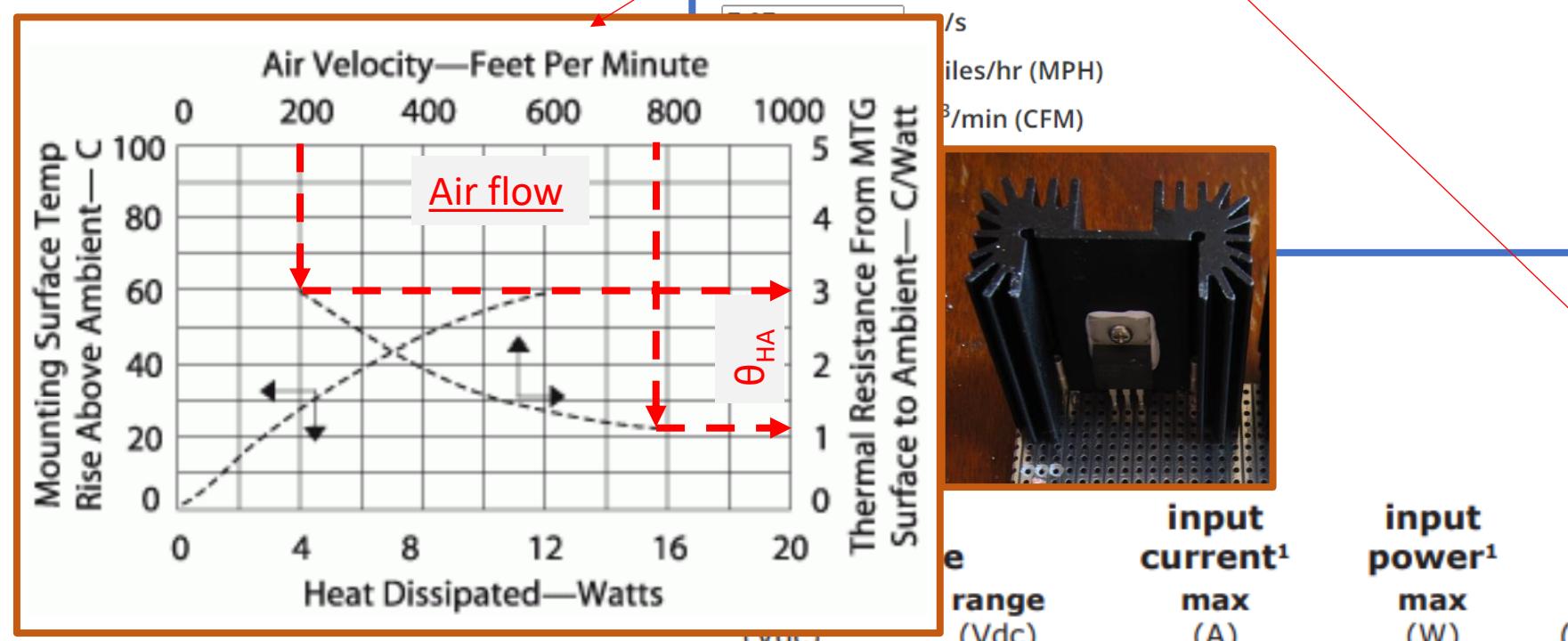
Calculate Clear

Results: 1392.32 ft/min (LFM)

$$LFM = \frac{CFM}{\text{Cross-Sectional Area}}$$

$$\text{Cross-Sectional Area} = \pi R^2 \text{ [sq. ft]}$$

Transfer R to feet!



range (Vdc)	input current ¹ max (A)	input power ¹ max (W)	rated speed ¹ typ (RPM±10%)	airflow ² (CFM)	static pressure ³ (inch H ₂ O)
12	10.8~13.2	1.87	22.44	5,300	169.5

Heat management

Most applications require a heat sink to assure that the maximum operating junction temperature (125°C) is not exceeded. In addition, the junction temperature should be kept as low as possible for increased reliability. Junction temperature can be determined according to the Equations:

$$T_J = T_A + P_D \theta_{JA}$$

where $\theta_{JA} = \theta_{JC} + \theta_{CH} + \theta_{HA}$

T_J = Junction Temperature ($^{\circ}\text{C}$)

T_A = Ambient Temperature ($^{\circ}\text{C}$)

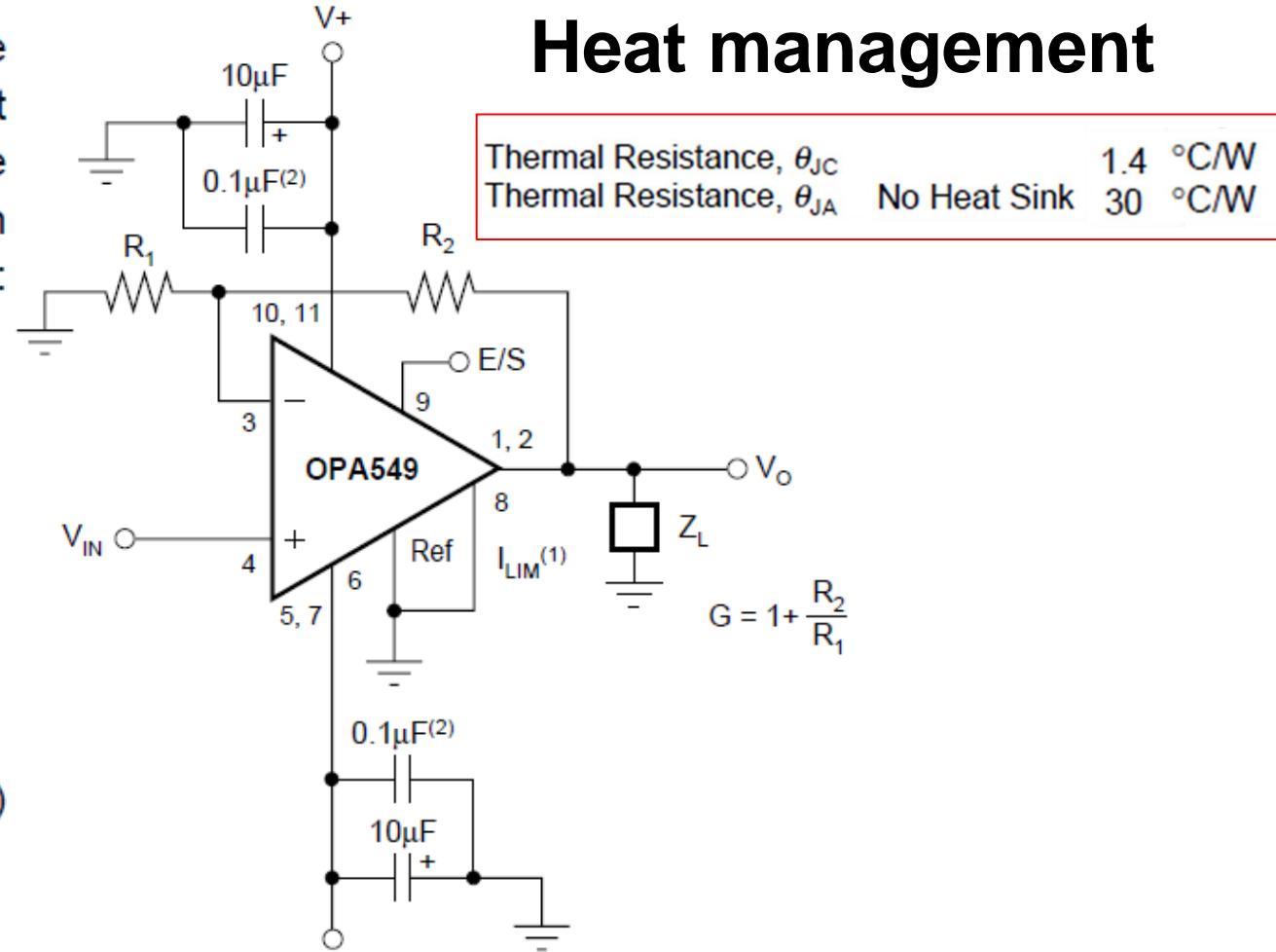
P_D = Power Dissipated (W)

θ_{JC} = Junction-to-Case Thermal Resistance ($^{\circ}\text{C/W}$)

θ_{CH} = Case-to-Heat Sink Thermal Resistance ($^{\circ}\text{C/W}$)

θ_{HA} = Heat Sink-to-Ambient Thermal Resistance ($^{\circ}\text{C/W}$)

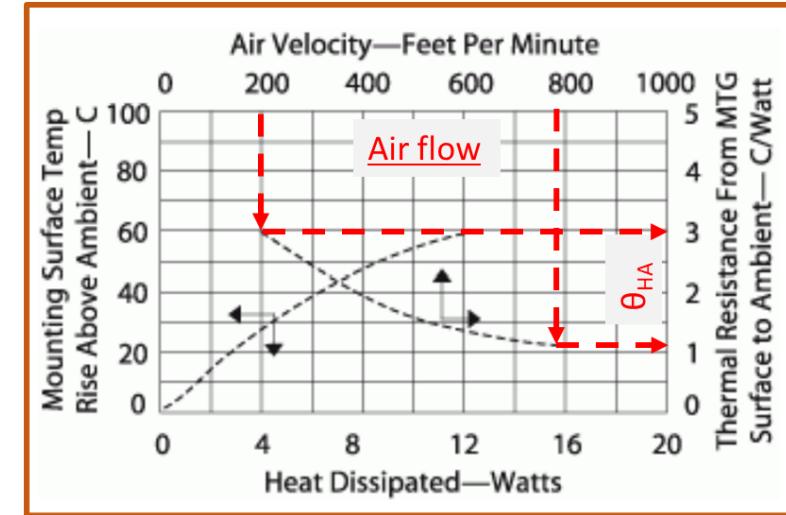
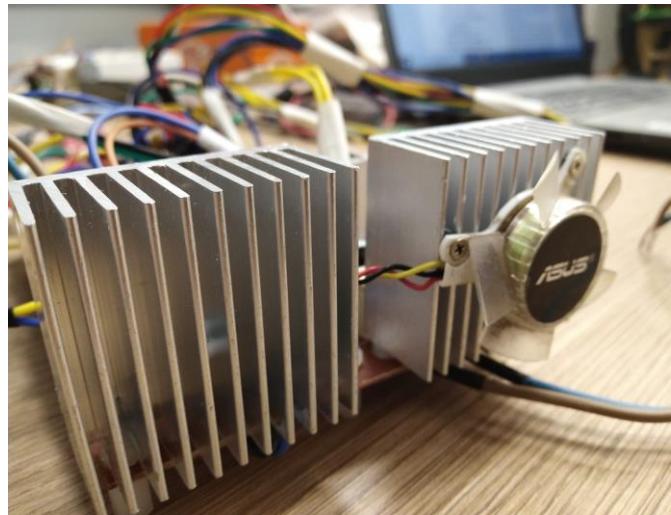
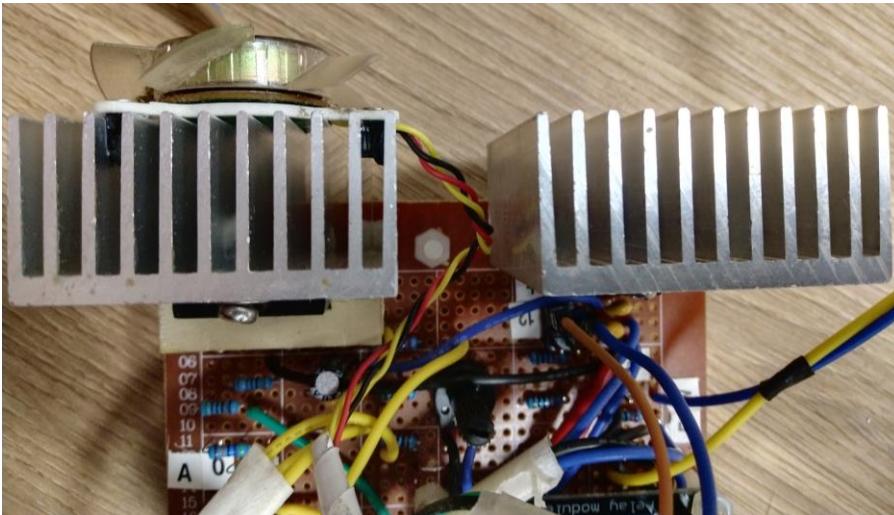
θ_{JA} = Junction-to-Air Thermal Resistance ($^{\circ}\text{C/W}$)



Since the theoretically minimum value of $\theta_{JA} = \theta_{JC} = 1.4 \text{ } ^{\circ}\text{C/W}$ (ideal heat transfer from the case to air, $\theta_{CH} + \theta_{HA} = 0$), we can conclude that the maximum dissipated power cannot exceed $P_D = (T_J - T_A)/\theta_{JA} \sim 70 \text{ W}$, if $T_A = 25 \text{ } ^{\circ}\text{C}$. Therefore, assuming a maximum operating current of 8 A, the supply voltage should not exceed $V_S = 2P_D/I_{load} \sim 17.5 \text{ V}$ (max + 60 V or $\pm 30 \text{ V}$ for OPA549).

Heat management

Our well-blown heatsink for OPA549 is likely to add $\theta_{HA} \sim 1 \text{ }^{\circ}\text{C/W}$ to the overall $\theta_{JA} \approx \theta_{JC} + \theta_{HA}$, meaning the maximum dissipated power cannot exceed 42 W. Previously, we estimated that at a supply voltage of 24 V and a maximum current of 3 A, which is necessary for our applications, the dissipated power would be 36 W. Thus, we are close to the maximum allowable values. The choice of supply voltage should be based on the typical load resistance (in our case, the Helmholtz coil) and the required current (to achieve magnetic saturation during scanning). Achieving the maximum current and voltage simultaneously for a particular component is practically impossible due to power dissipation limitations, although these peak values can be reached in certain short-term operating modes.



Principle of a PWM DC voltage source

The [PWM](#) signal is a periodic extension of the following step function defined over the interval T , which serves as the basis for the periodic extension period:

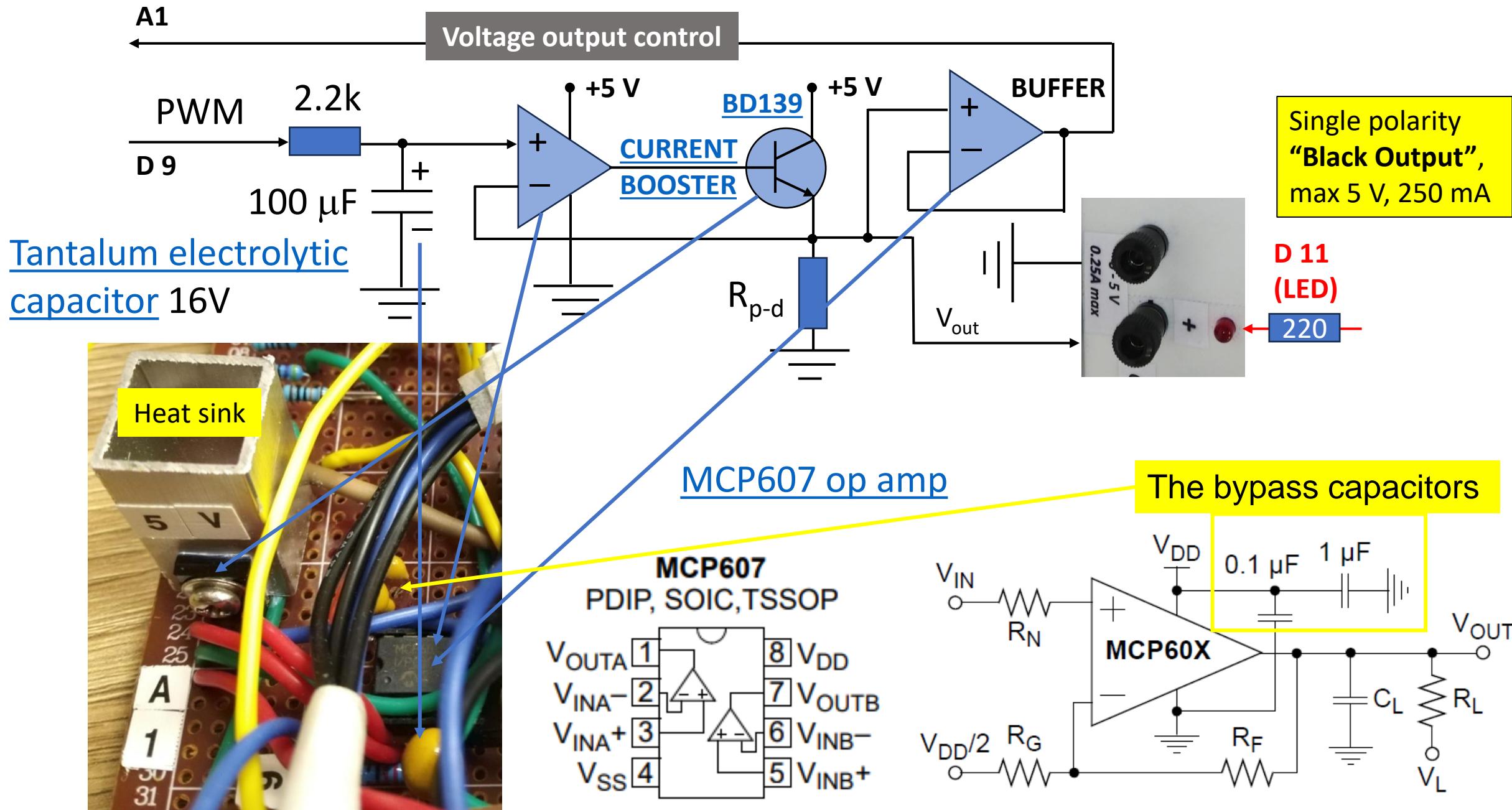
$$V(t) = \begin{cases} A, & 0 \leq t \leq \tau \\ 0, & \tau < t < T \end{cases}$$

where A is a positive constant (5V for Arduino or 3.3V for Raspberry Pi). The ratio τ/T (%) is called [the duty cycle](#). The Fourier spectrum of this function is represented by the following series:

$$V(t) = \frac{A\tau}{T} + \sum_{k=1}^{\infty} \left[\frac{A}{k\pi} \sin\left(\frac{2\pi k\tau}{T}\right) \cos\left(\frac{2\pi k}{T}t\right) + \frac{A}{k\pi} \left(1 - \cos\left(\frac{2\pi k\tau}{T}\right)\right) \sin\left(\frac{2\pi k}{T}t\right) \right]$$

By employing a RC [low-pass filter](#) with sufficiently high capacitance and resistance values, the elimination of all harmonics except the zeroth $A\tau/T$ is achieved, leading to the desired DC voltage level. This level is determined by the duty cycle. The load resistance must not be small, as this would hasten the discharge of the capacitor, leading to substantial fluctuations (ripple) in the output voltage $V_{out}(t)$.

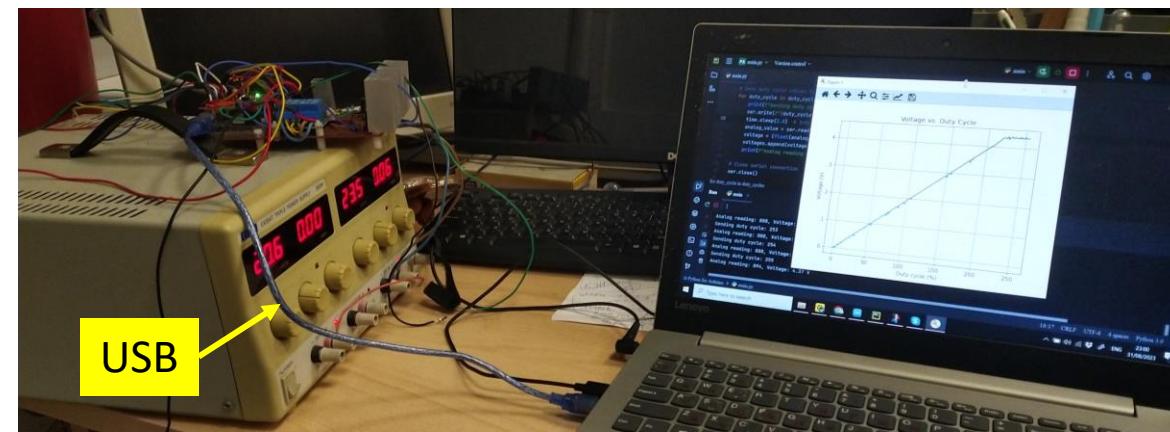
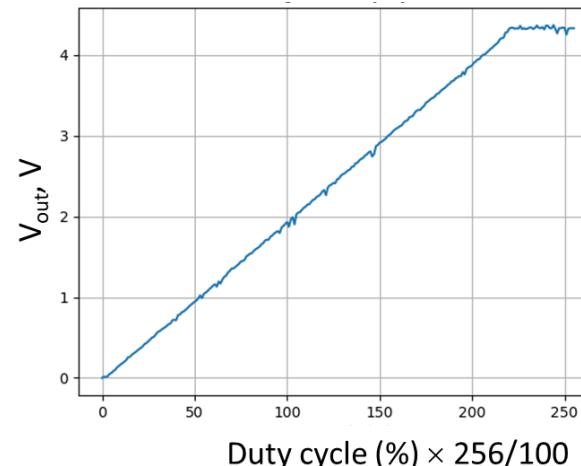
Implementation of the PWM DC voltage source



Testing the PWM driver from Python

```
1. import serial
2. import time
3. import numpy as np
4. import matplotlib.pyplot as plt
5.
6. # Open serial connection to Arduino
7. ser = serial.Serial('COM7', 9600)
8. time.sleep(1) # Wait for Arduino to reset
9.
10. # Sweep duty cycles from 0 to 255
11. duty_cycles = np.arange(0, 256)
12. voltages = []
13.
14. # Send duty cycle values to Arduino, collect analog readings, and convert
15. for duty_cycle in duty_cycles:
16.     print(f"Sending duty cycle: {duty_cycle}")
17.     ser.write(f"{duty_cycle}\n".encode()) # Send duty cycle to Arduino
18.     time.sleep(1.5) # Introduce a delay time (s)
19.     analog_value = ser.readline().decode().strip() # Read analog value from Arduino
20.     voltage = (float(analog_value) / 1023.0) * 5.0 # Convert to voltage
21.     voltages.append(voltage)
22.     print(f"Analog reading: {analog_value}, Voltage: {voltage:.2f} V")
23.
24. # Close serial connection
25. ser.close()
26.
27. # Plotting the graph
28. plt.plot(duty_cycles, voltages)
29. plt.xlabel("Duty cycle (0-255)")
30. plt.ylabel("Voltage (V)")
31. plt.title("Voltage vs. Duty Cycle")
32. plt.grid(True)
33. plt.show()
```

Python test program



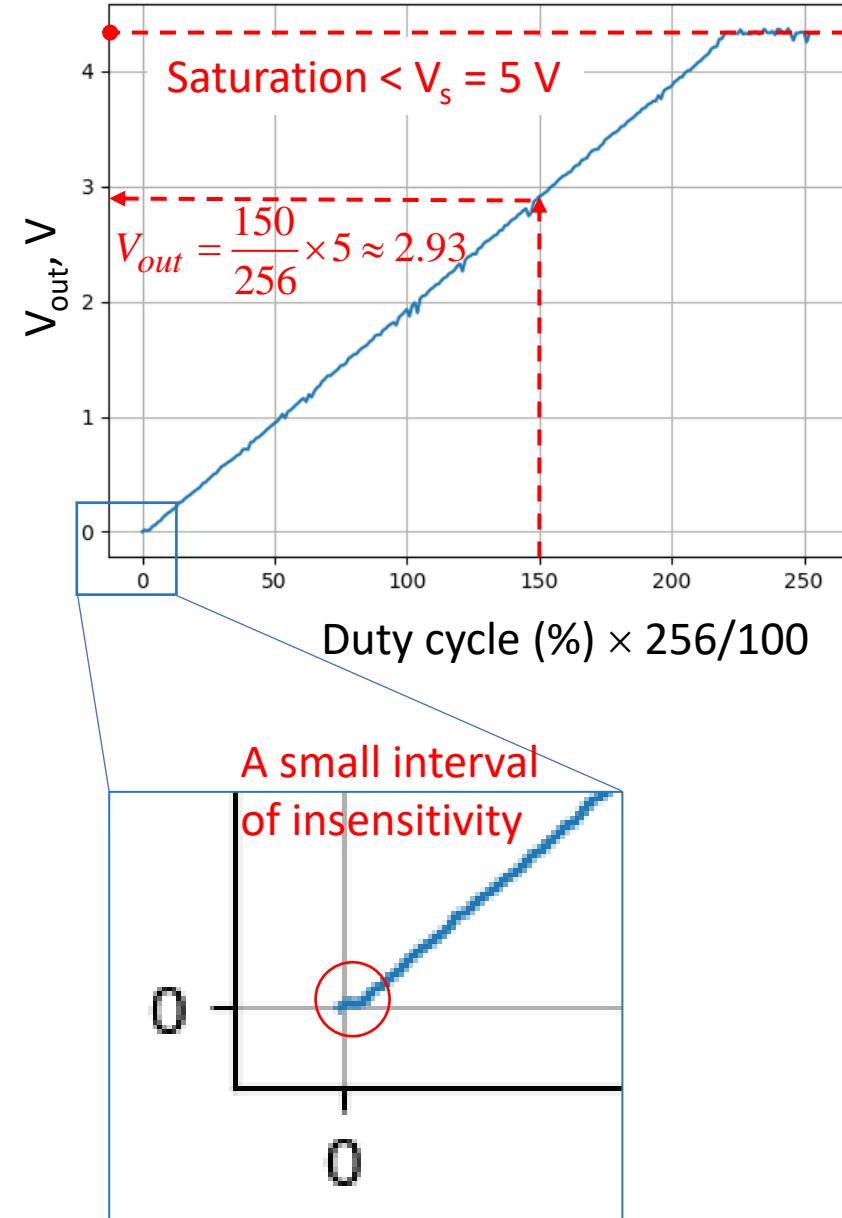
USB

```
1. #include <Arduino.h>
2.
3. const int pwmPin1 = 9; // PWM pin 1
4. int dutyCycle1 = 0; // Duty cycle for PWM pin 1
5.
6. void setup() {
7.     pinMode(pwmPin1, OUTPUT);
8.     analogWrite(pwmPin1, dutyCycle1);
9.
10.    Serial.begin(9600);
11. }
12.
13. void loop() {
14.     if (Serial.available() > 0) {
15.         dutyCycle1 = Serial.parseInt(); // Read duty cycle value
16.         analogWrite(pwmPin1, dutyCycle1);
17.     }
18.     delay(100);
19.
20.     // Read analog values from analog pin
21.     int analogValue1 = analogRead(A1);
22.     delay(1500);
23.
24.     // Send analog value to Python
25.     Serial.println(analogValue1);
26. }
```

Arduino sketch

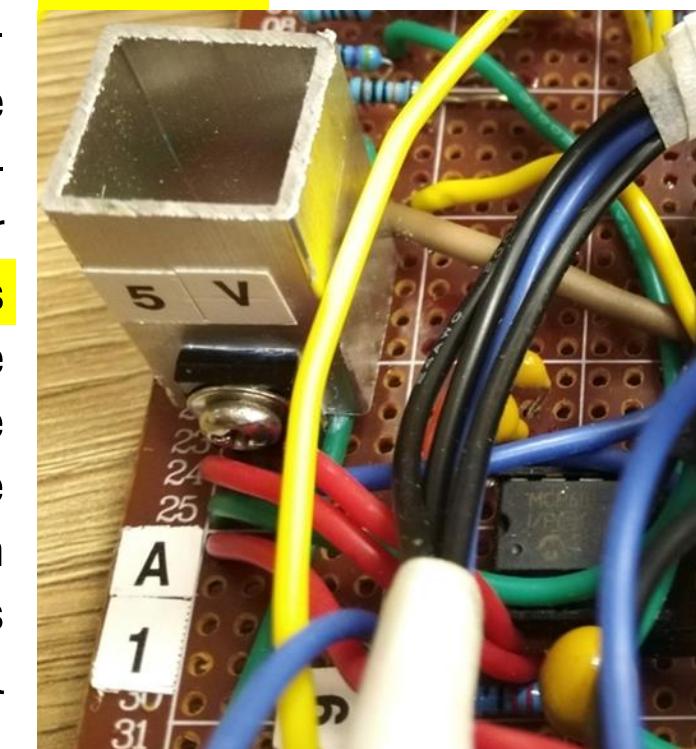
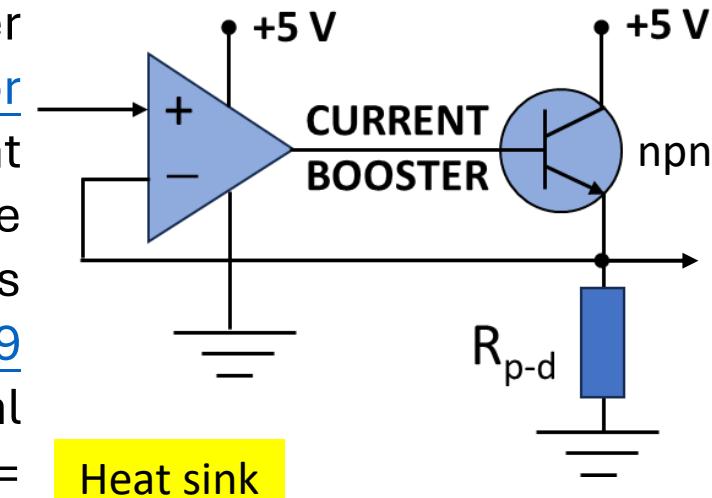
Operation of the PWM DC voltage source

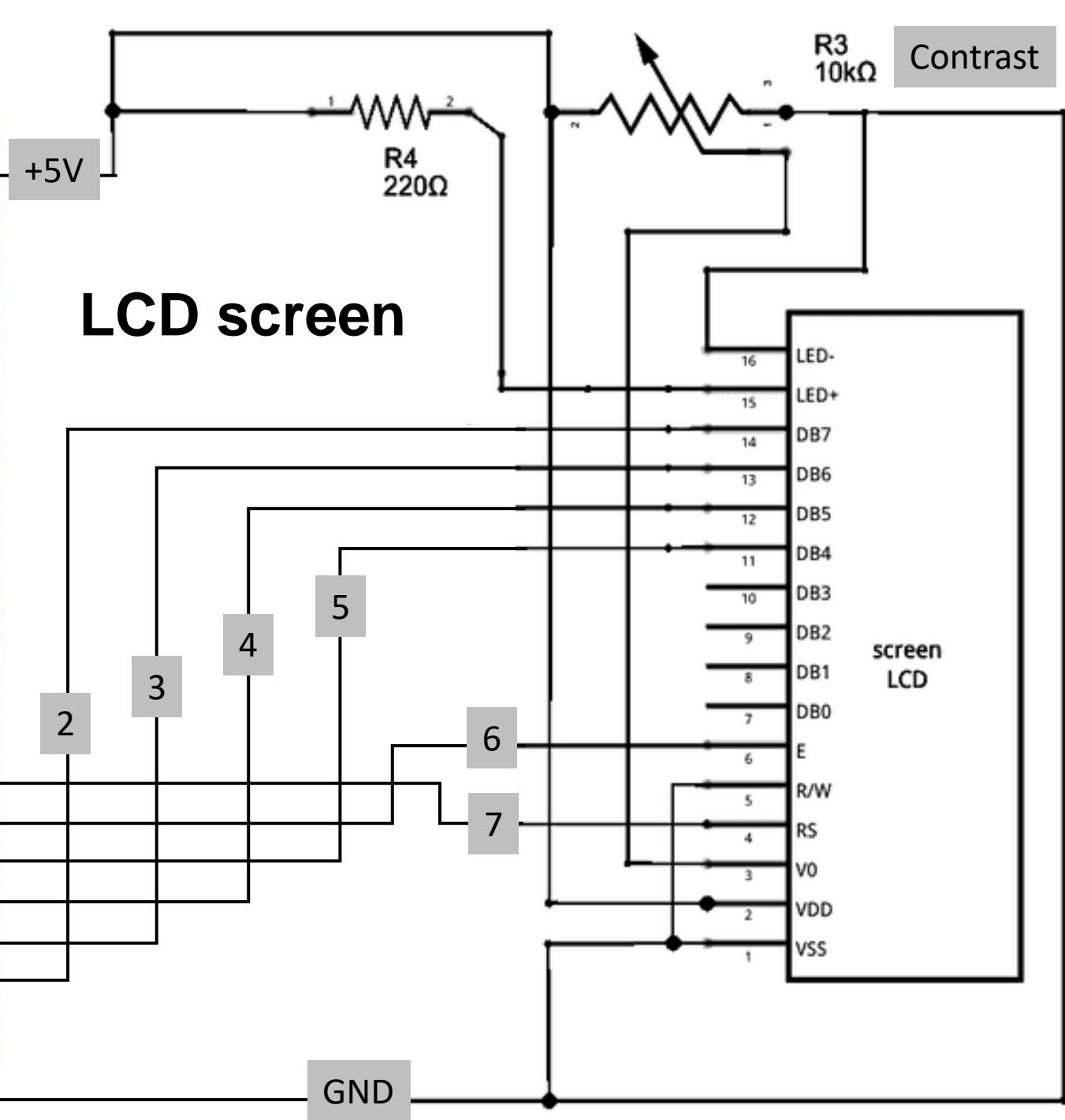
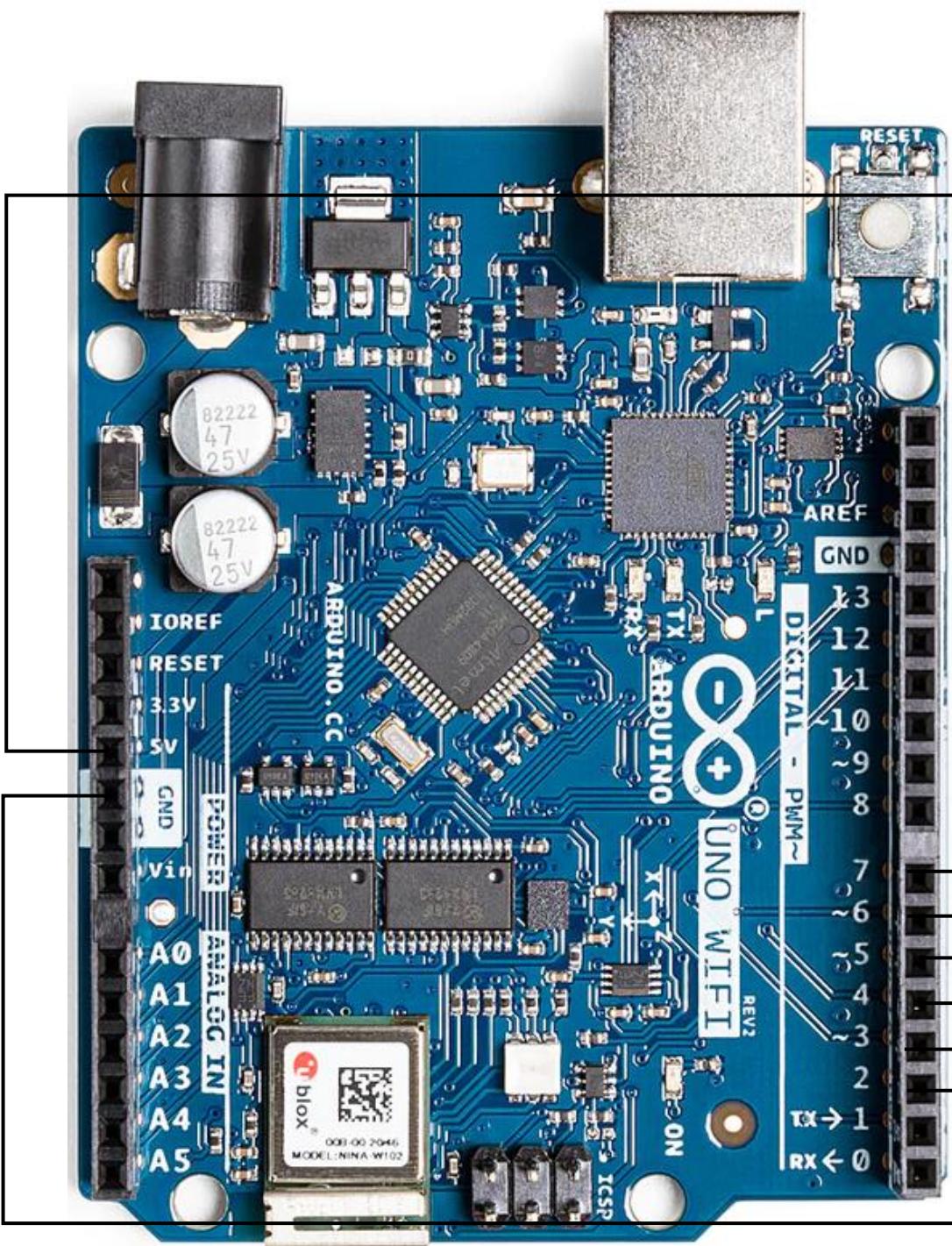
The high input impedance of a non-inverting operational amplifier ensures the proper operation of the RC low-pass filter with the characteristic time constant of $\tau = RC = 220$ ms (2.2 k Ω and 100 μF). When the duty cycle changes, it is necessary to wait at least $5\tau = 1.1\text{s}$ before the new (+) output voltage level stabilizes. This is a limitation of PWM sources – they are quite slow. It should also be noted that the maximum output voltage of the operational amplifier will be somewhat lower than the supply voltage, reaching a certain saturation level. Therefore, if we aim to achieve an output voltage of exactly 5 V, the supply voltage should be chosen at least 6 V if using a rail-to-rail op amp. The output characteristic is perfectly linear; however, at the beginning of the curve, i.e., at low duty cycle values, a small region of insensitivity may be observed.



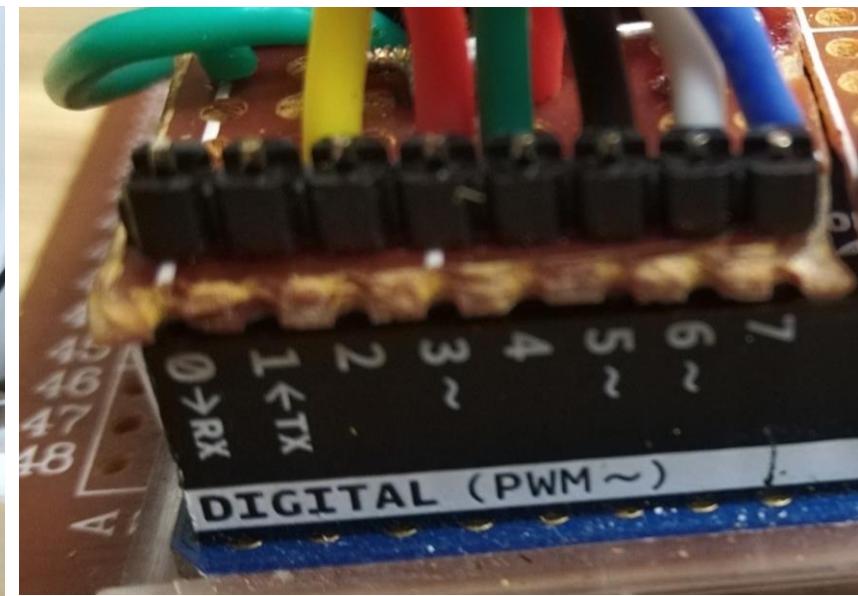
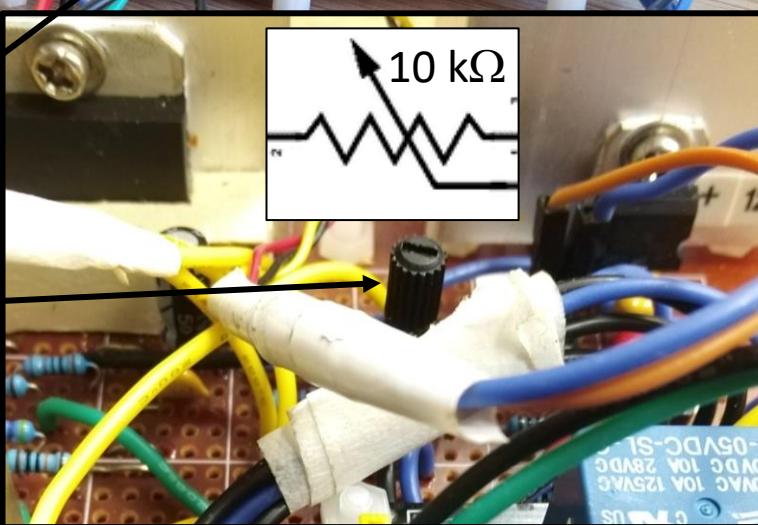
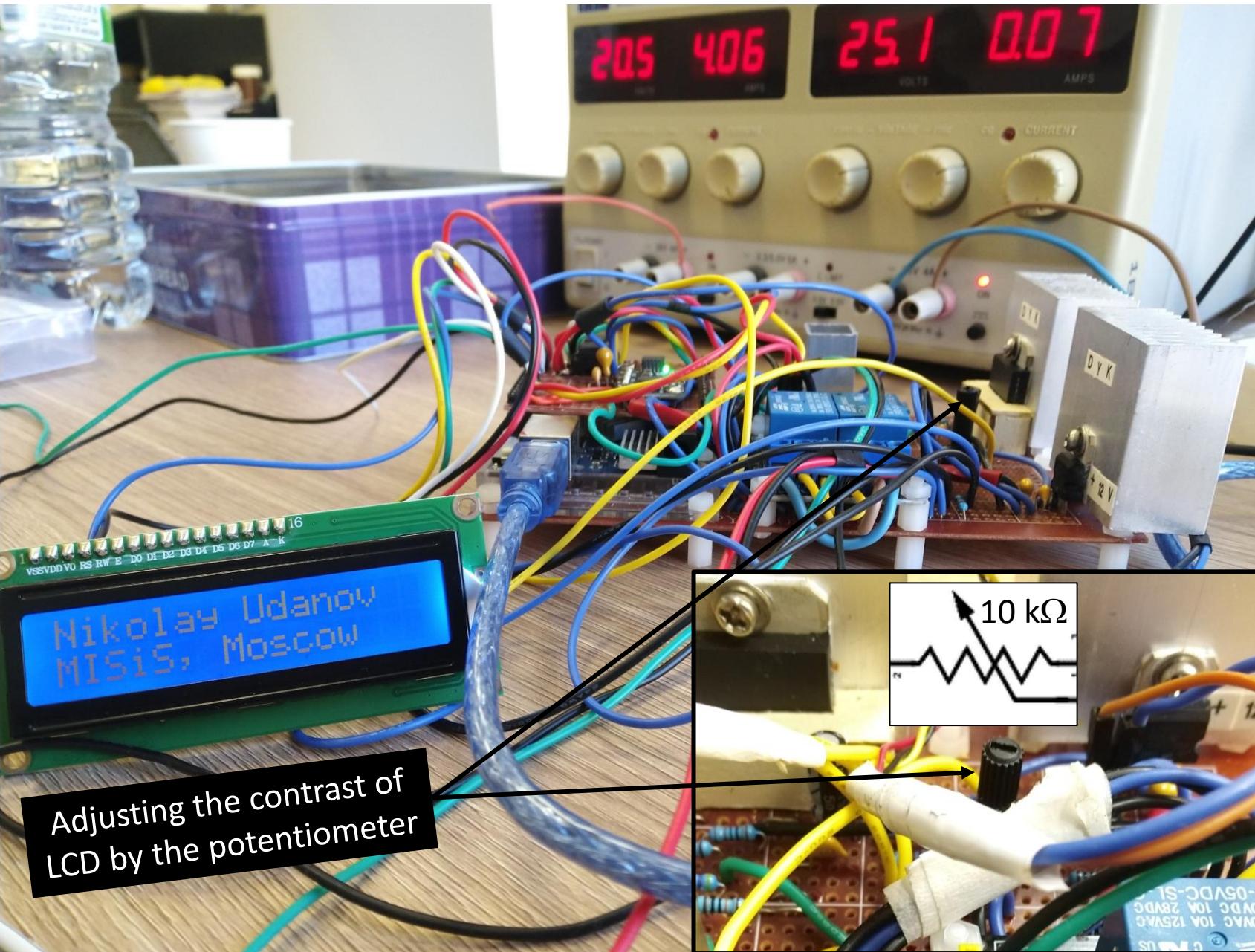
Current booster and heat management for the PWM DC voltage source

Since a standard operational amplifier cannot provide an output current greater than 20 – 25 mA, we used [a current booster circuit](#) based on a [npn transistor](#) incorporated into the feedback loop of the operational amplifier. The heat management of this transistor stage is handled using the same approach we explained above. With a 5 V supply voltage, maximum power dissipation occurs at an output voltage of 2.5 V. At the maximum current of 1.5 A for the [BD139](#) transistor, this results in 3.75 W. Without an external heatsink, the thermal resistance junction-to-air $\theta_{JA} = 100 \text{ }^{\circ}\text{C/W}$. Therefore, with $T_A = 25 \text{ }^{\circ}\text{C}$ and $T_J = 150 \text{ }^{\circ}\text{C}$, the maximum power dissipation is calculated as $P_D = (T_J - T_A) / \theta_{JA} = 1.25 \text{ W}$ (no heatsink). The thermal resistance junction-to-case $\theta_{JC} = 10 \text{ }^{\circ}\text{C/W}$. The use of a heatsink with fan cooling can achieve θ_{HA} (heatsink-to-air) close to 1 resulting in $\theta_{JA} \approx 11 \text{ }^{\circ}\text{C/W}$. In this case, the dissipated power can be increased up to 11.4 W. Thus, the operating power range for this transistor is 1.25 W (no heatsink) to 11.4 W (heatsink and fan cooling). The maximum voltage for the BD139 is 80 V. At a maximum current of 1.5 A, the maximum possible dissipated power is 60 W, which significantly exceeds the allowable limit of 11.4 W, even with highly efficient fan cooling. We used a small heatsink (no fan cooling) made from a piece of aluminium profile. This means that by selecting the maximum output current of 250 – 300 mA, we will stay well below 1.25 W. The pull-down resistance R_{p-d} can be chosen as 100 k Ω .

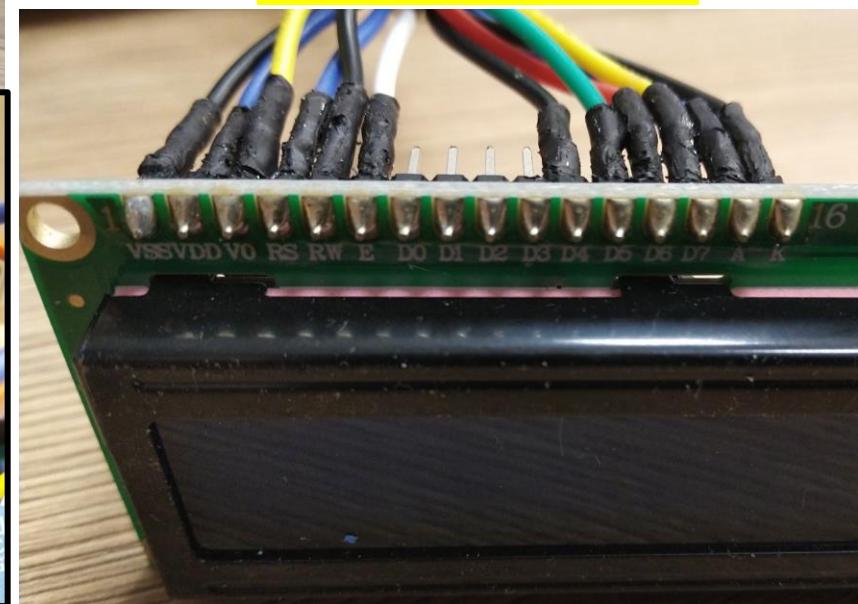




Wiring the LCD screen for displaying voltages on the Red and Black outputs.

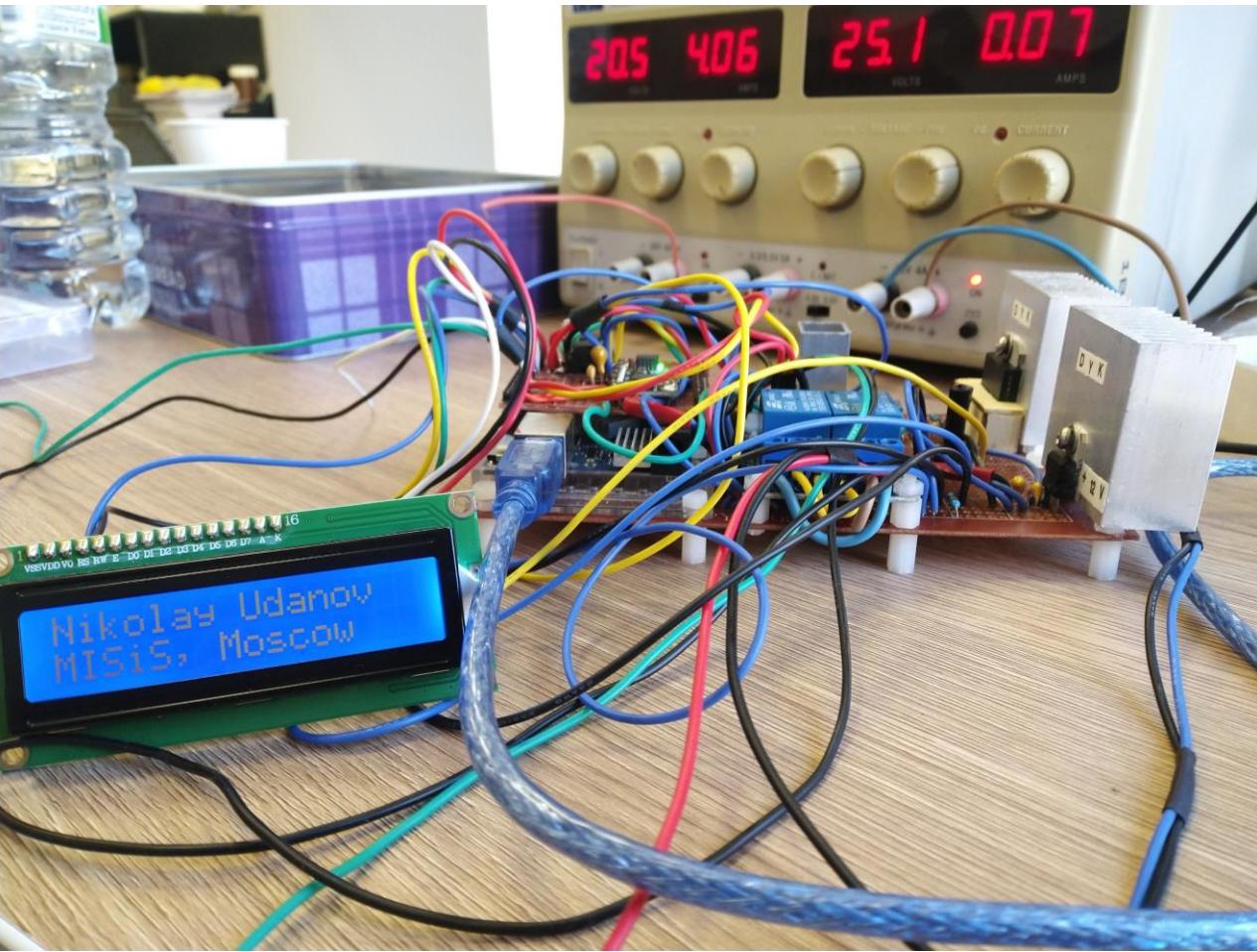


wire colour code



Testing the LCD screen

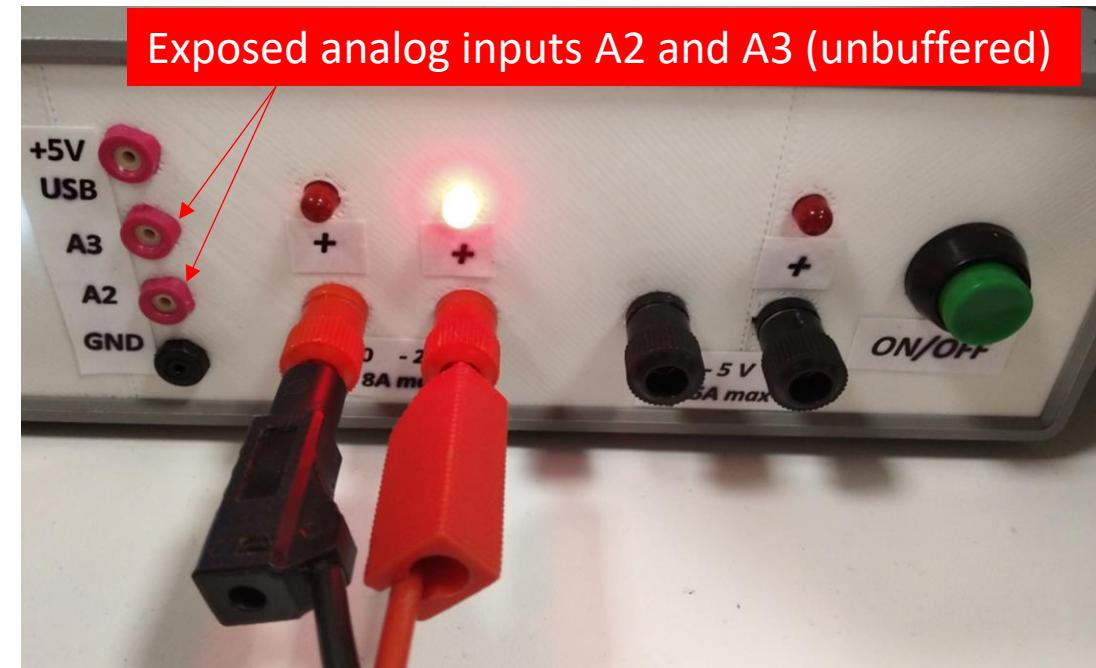
```
1. #include <LiquidCrystal.h>
2.
3. LiquidCrystal lcd(7, 6, 5, 4, 3, 2);
4.
5. void setup() {
6.   lcd.begin(16, 2);          Arduino sketch
7.   lcd.clear();
8.   lcd.setCursor(0, 0);
9.   lcd.print("Nikolay Udanov");
10.  lcd.setCursor(0, 1);
11.  lcd.print("MISiS, Moscow"); // Print the second line
12.
13. }
14.
15. void loop() {
16.   // Write-only operations here, you don't need to read from the LCD.
17. }
```



Outputs and inputs on the front and rear panel

We recommend exposing the unused analog inputs and GPIO pins of the Arduino on the front panel to allow the device to be reconfigured for other tasks, including the integration of additional sensors. The analog inputs can either be buffered and equipped with pull-down resistors within the power supply unit or left unconnected for external configuration. In the current project, this aspect has not been given sufficient attention. However, in another similar [project](#), this option was fully implemented. It is also possible to use specialized connectors on the rear panel, which provide integrated I/O functionality.

Pin	Circuit	Description
1	NC	Not connected
2	5V USB	5V from USB port
3	A3	ADC input A3
4	A2	ADC input A2
5	GND	Common ground
6	LRed	Red output, left socket
7	RRed	Red output, right socket
8	LBlac (GND)	Black output, Ground
9	RBlac	Black output, positive (+)



Instructions for Using the Power Supply Control Program

1. Upload the sketch [**Voltage_Red_Black_Outputs.ino**](#) to the Arduino.
2. Run the [**main.py**](#) program, which activates the Red or Black outputs and controls the voltage on them.
3. The **main.py** program utilizes the library function [**Volt_Red_Black_Out.py**](#), included in the project folder. This library handles communication with the Arduino via the serial port.
4. For projects involving measurement automation using Python, only the **main.py** script needs to be modified to adapt the functionality.

Voltage feedback control in the Arduino sketch

The program manages the output voltages, adjusting them within an accuracy of +/-1.5 bits (this parameter is configurable). The voltage value per bit will vary when controlling the Red and Black outputs. When reading the output from the OPA op amp, the DAC provides 12-bit precision, but the ADC is limited to 10-bit precision. For PWM outputs, the precision is determined by the resolution of the corresponding GPIO, which is only 8 bits. In both cases, the reference voltage is 5 V.

```
const float bs_DAC = 1.5 * (5.0 / 1023.0) * A0_coeff; // bit sensitivity for DAC output
const float bs_PWM = 1.5 * (5.0 / 255.0); // bit sensitivity for PWM output
```

```
DAC_value = round(((fabs(v1) / op_amp_gain) / 5.0) * 4095.0); // Convert the DAC voltage to
the DAC value (0-4095)
v1_out = DAC_Vout(DAC_value); // Output voltage how it is seen by the A0 analog input
while ((fabs(v1_out - fabs(v1)) > bs_DAC) ) {
    if (v1_out > fabs(v1) and DAC_value > 0)
        DAC_value--;
    else if (v1_out < fabs(v1) and DAC_value < 4095)
        DAC_value++;
    v1_out = DAC_Vout(DAC_value);
}
```

(Similar for the PWM output)