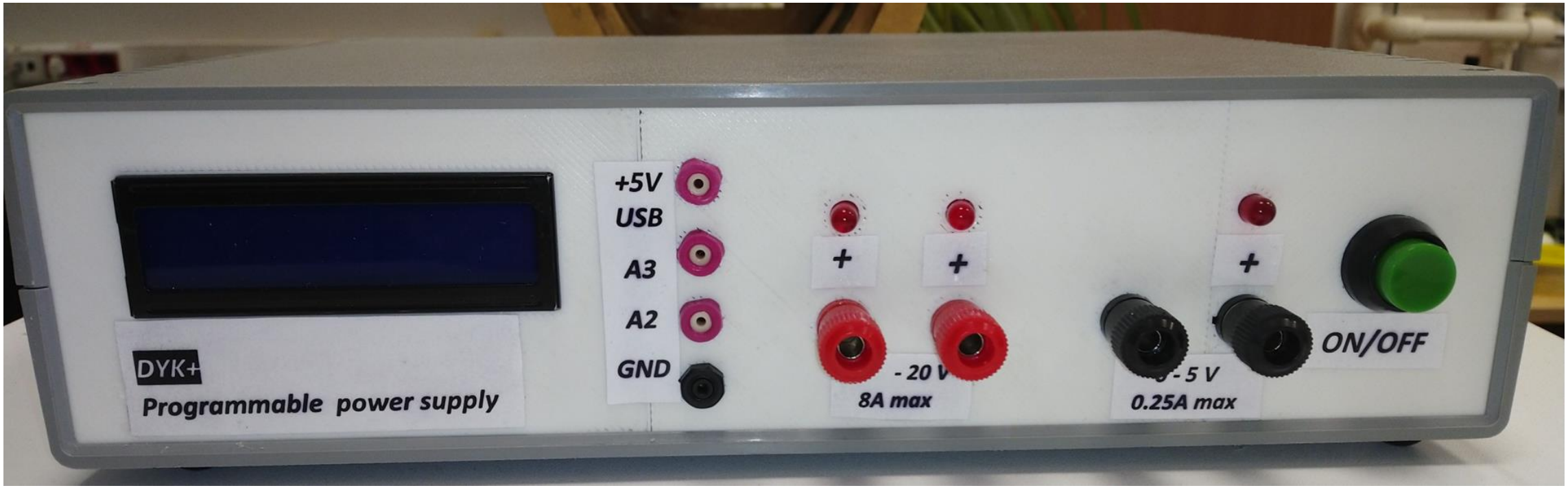
# Programmable DC power supply based on Arduino UNO

Dmitriy Makhnovskiy, DYK

Nikolay Udanov and Egor Shevchenko, MISIS



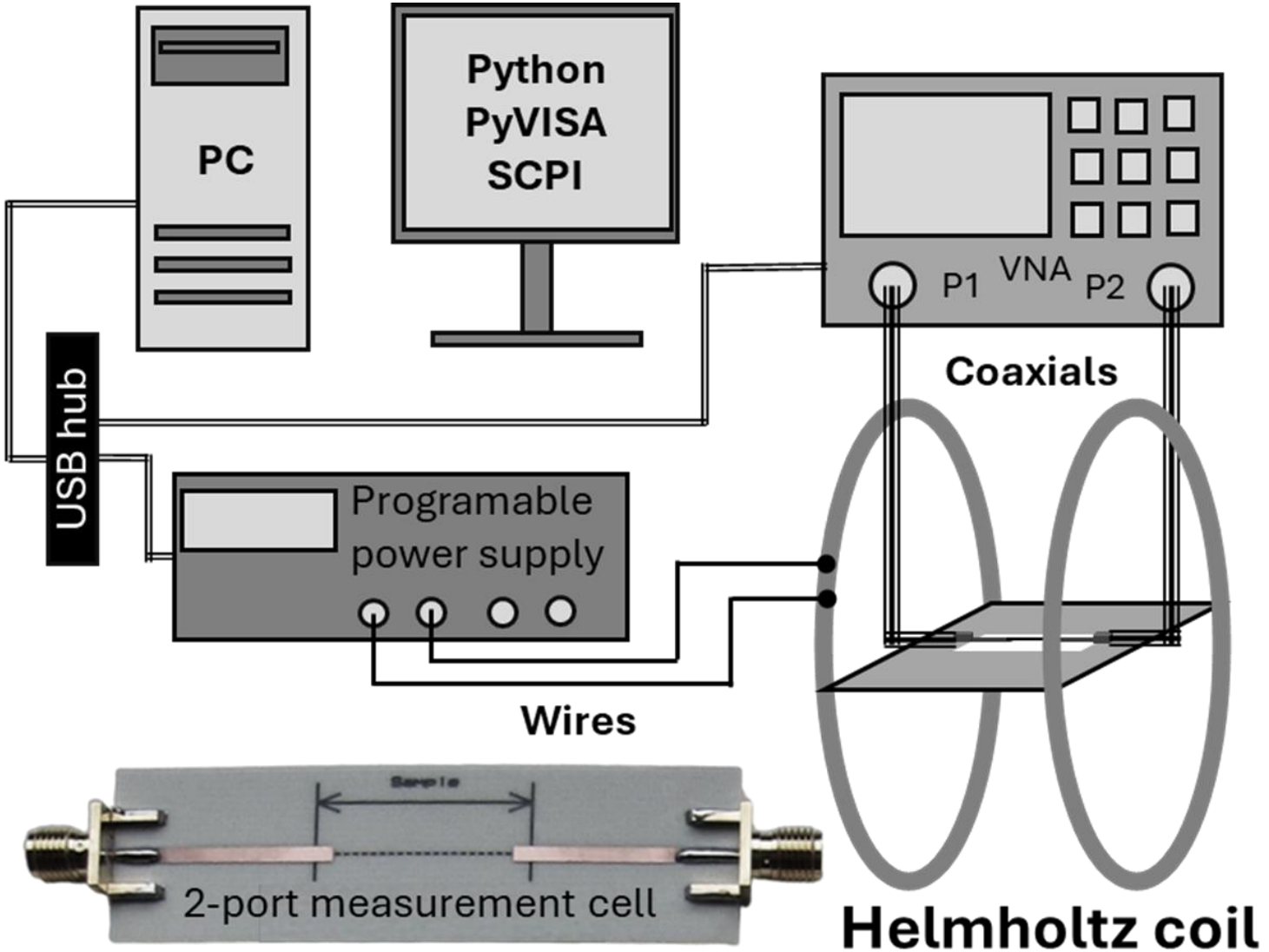

**GitHub:** https://github.com/DYK-Team/Programmable-DC-power-supply

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.

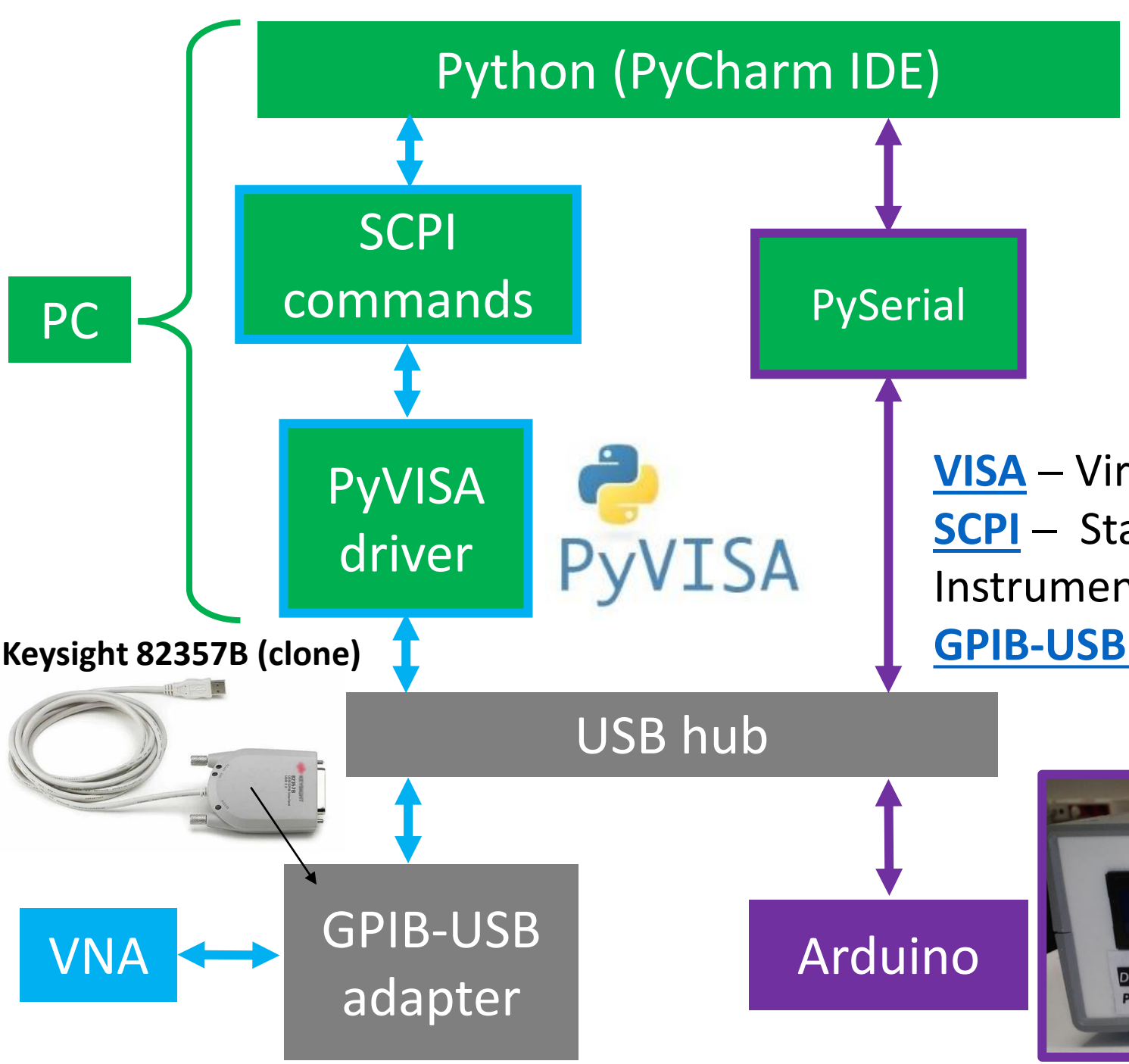
# Overview of applications: MI measurements and motor control

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

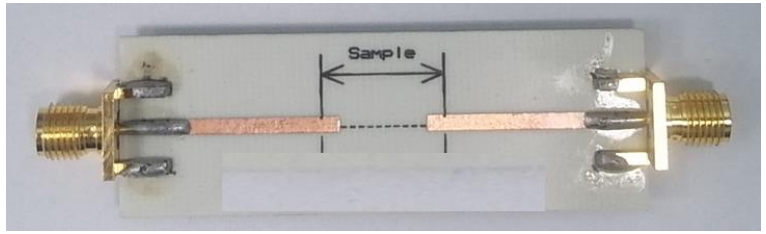

PyCharm

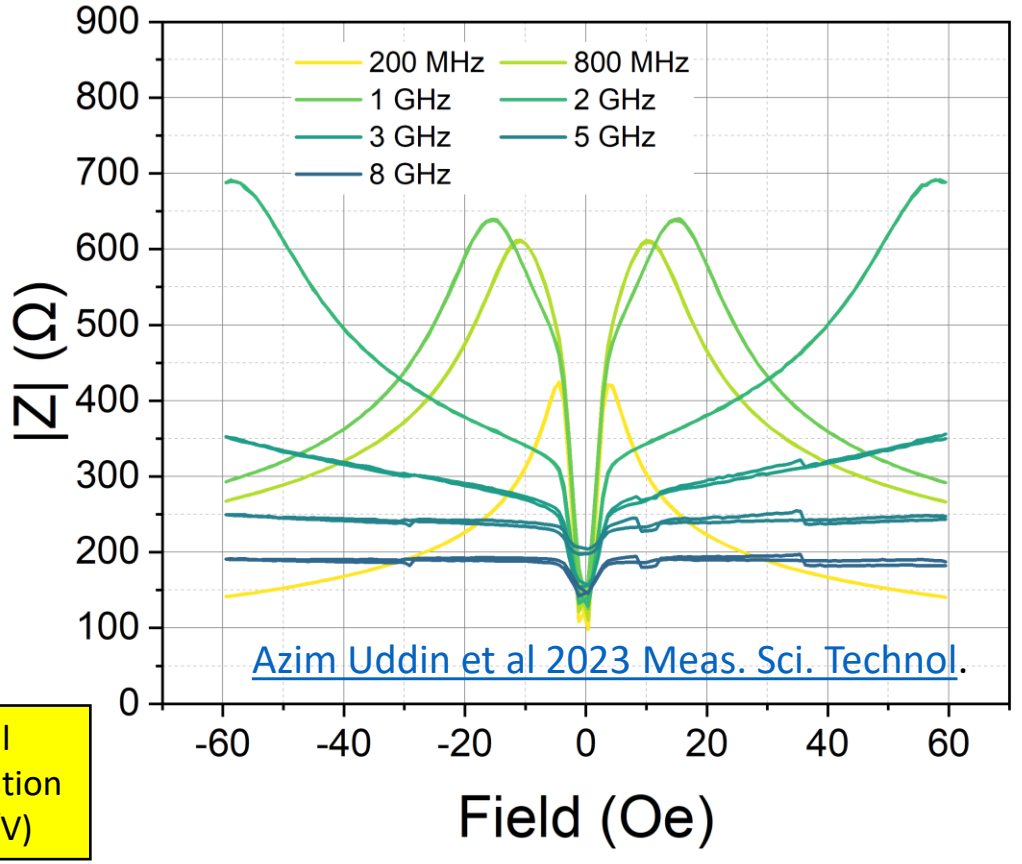
[VISA](#) – Virtual Instrument Software Architecture

[SCPI](#) – Standard Commands for Programmable Instruments

[GPIB-USB adapters](#) are used for obsolete VNAs

# 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

YES

i≤N

NO

STOP

DC power supply $V_C$

Coil

Coil's LR parameters

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

PyVISA

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

i = i+1

$V_C$

S

Coil calibration (Oe/V)

**DATA PROCESSING**

Magnetic field

SOLT calibration & deembedding the microstrips

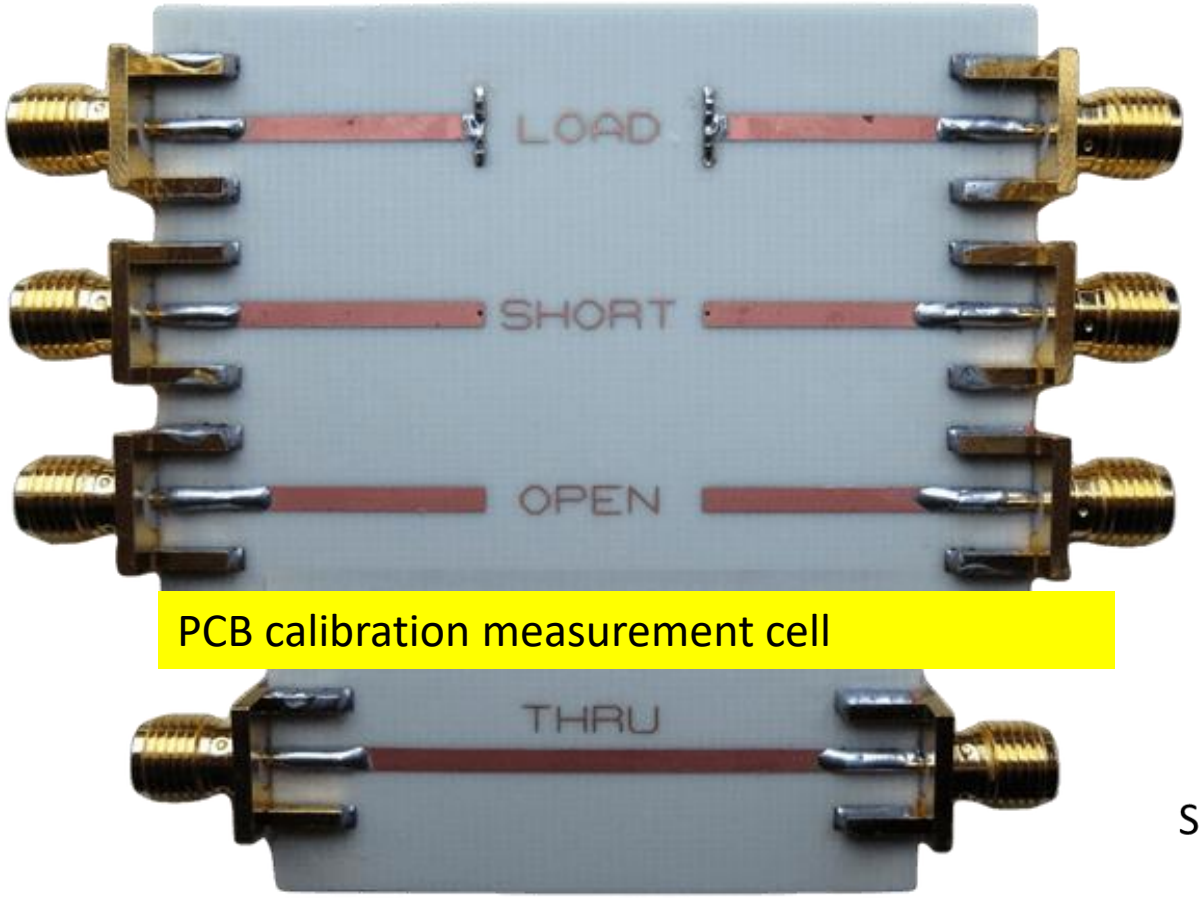
Delay time compensation

Impedance Z

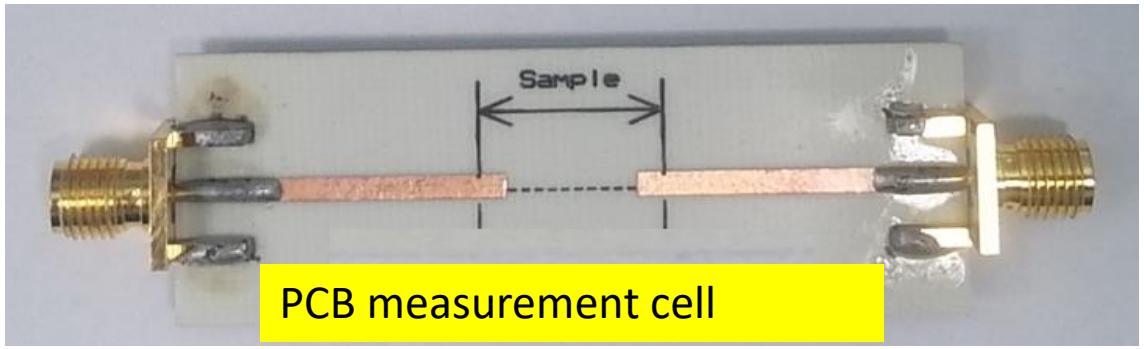
Sample's delay time

PC

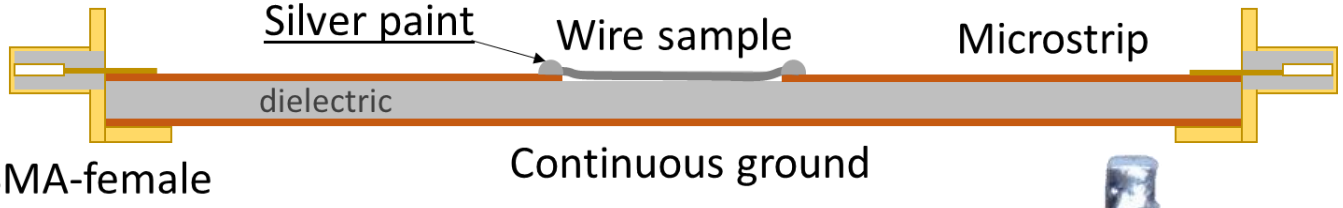
# PCB microstrip cells used for the calibration and MI measurements

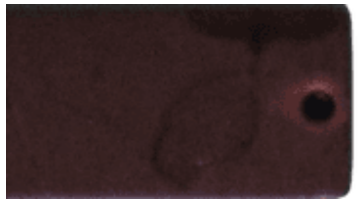



PCB calibration measurement cell

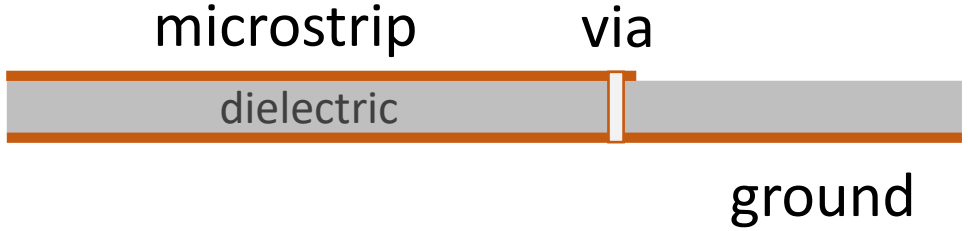



PCB measurement cell



**SHORT**
0.3 mm via



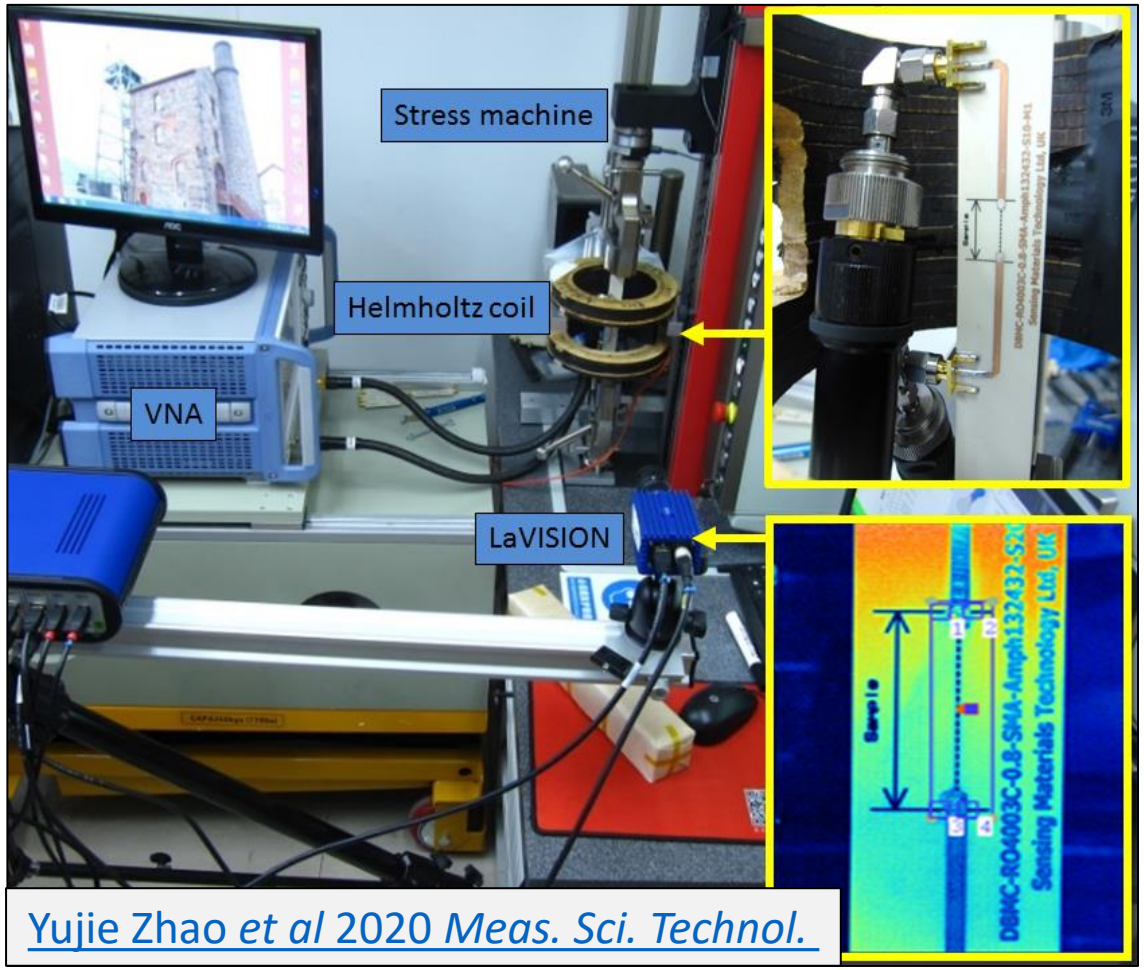

Pad with via


1.626 mm
(size 0603)

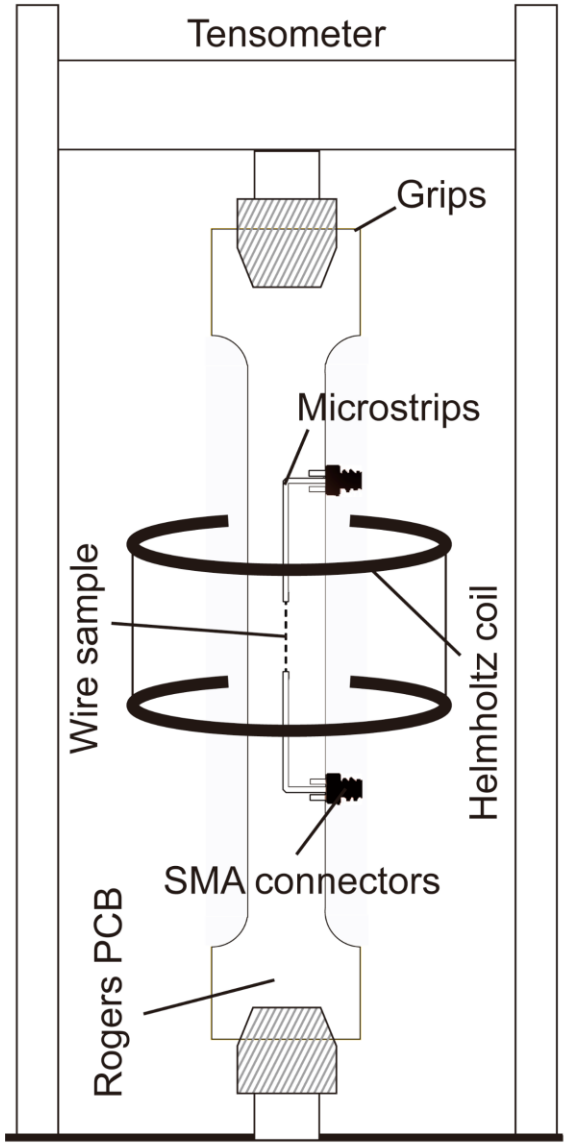
Pad with via

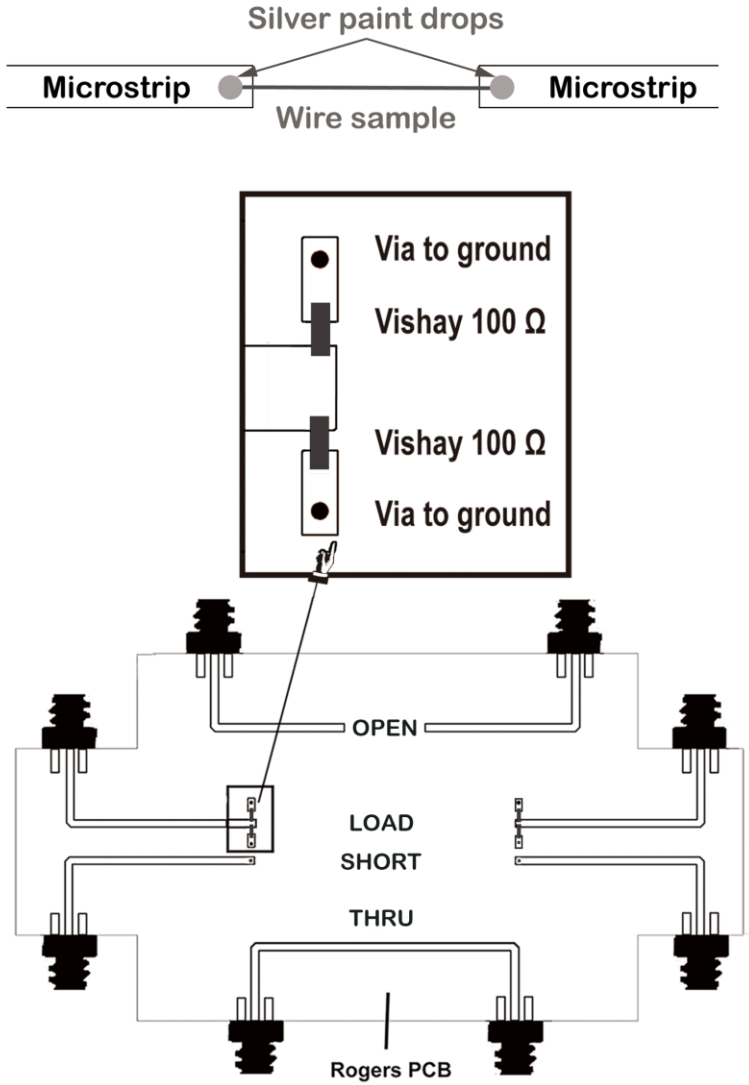
**LOAD**
Two parallel
100 Ω Vishay resistors

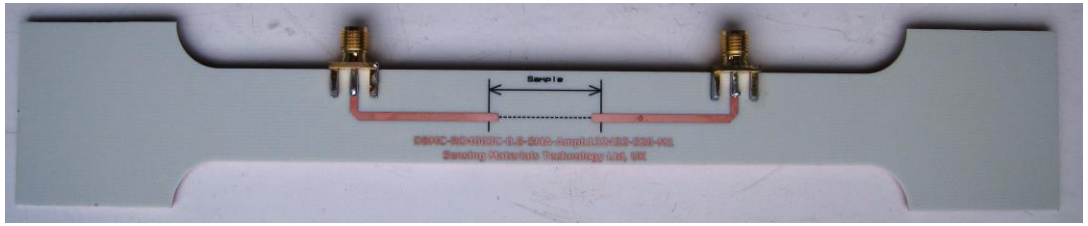
An advanced setup may incorporate devices to apply additional stimuli to the sample, such as heating or tension. Tension can be transferred to the sample through a specially designed "dogbone" PCB cell, which is secured between the grippers of a stress machine. This configuration has already been implemented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




Yujie Zhao *et al* 2020 *Meas. Sci. Technol.*



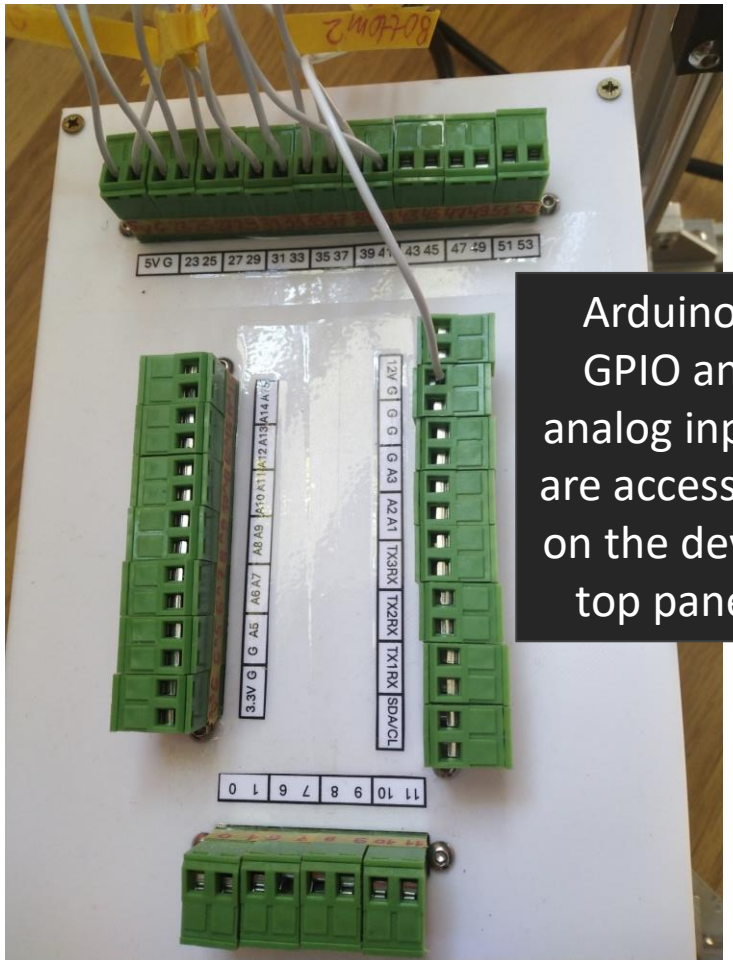



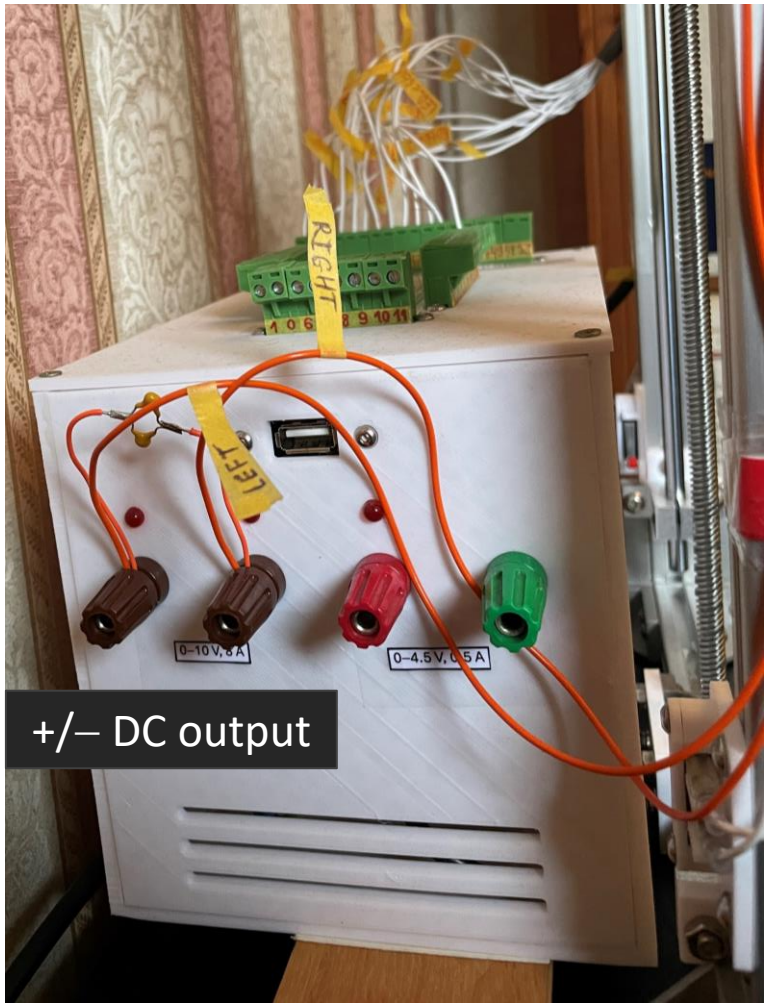
# Other implementations of the MI measurement setups

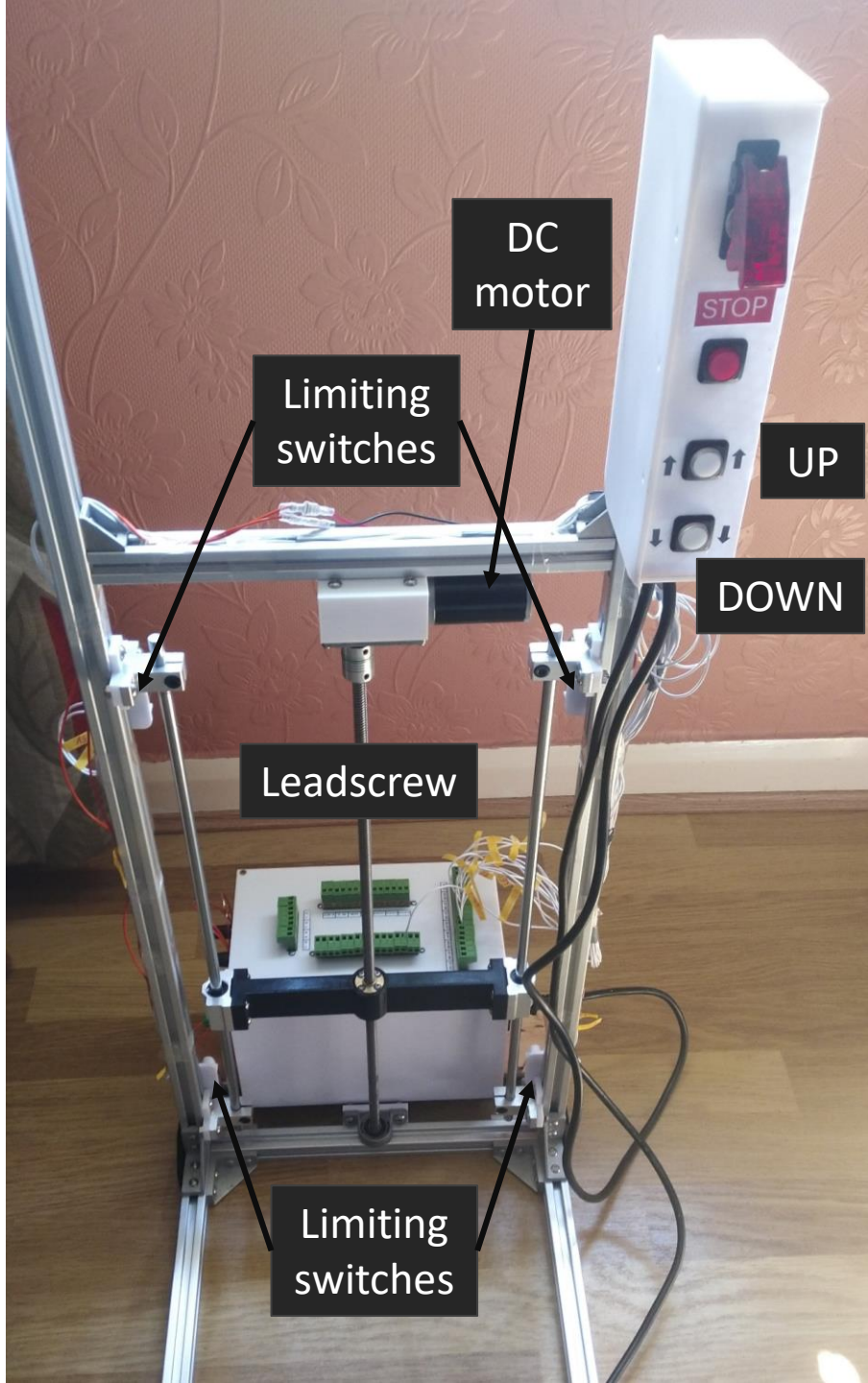

Zhejiang University, China


Plymouth University, UK

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.

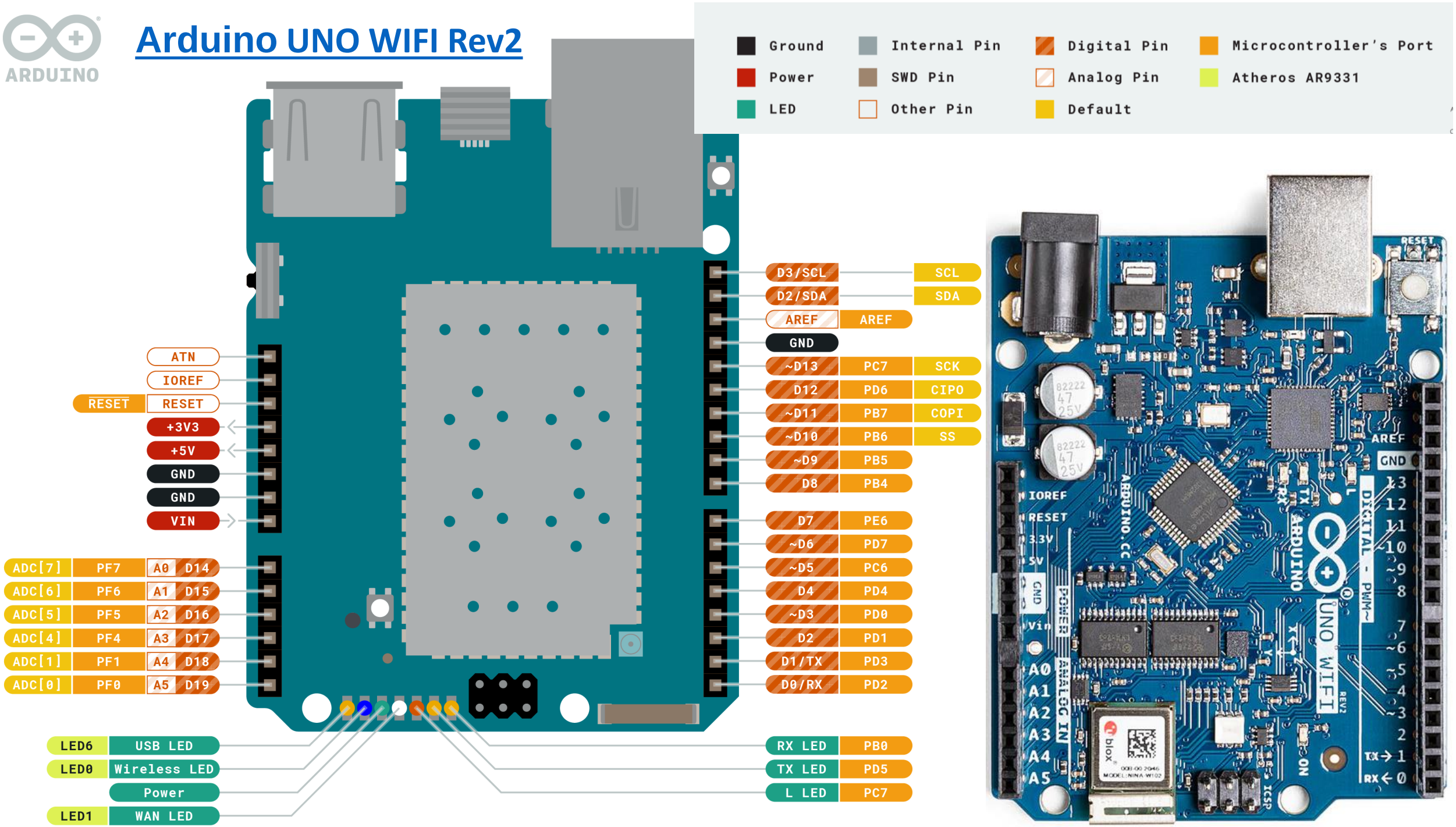


# Design of the programable power supply

# Arduino UNO WIFI Rev2



# Architecture of the power supply

DAC
MCP4725

ARDUINO with a shield

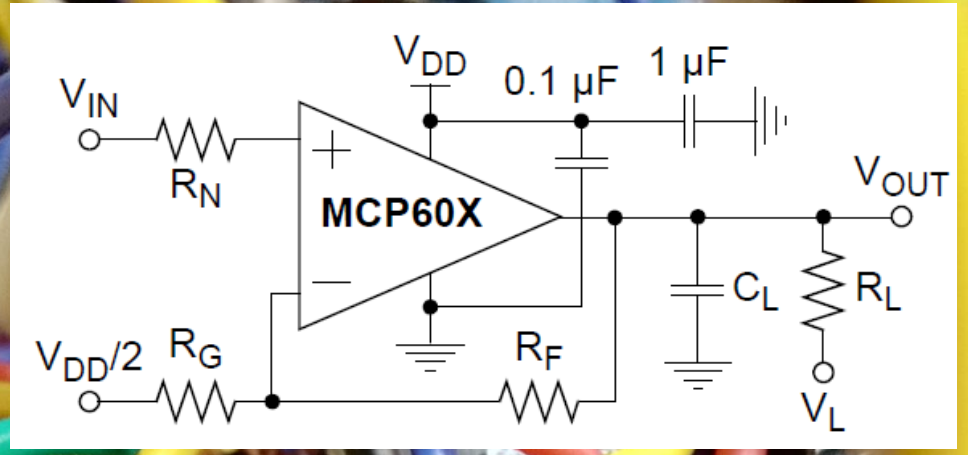
GRD
+5 V
A0
A1
D8
D9
D10
D11
D12
D13

0–5 V
+24 V
OPA 549
D12 D13 +5 V
Relay H-bridge
D8 (LED)
220
D10 (LED)
220
$R_1$
$R_2$
$R_3$
$R_4$
BUFFER
0/1
Voltage output control
PWM
2.2k
100 µF
0/1
+5 V
BD139
CURRENT BOOSTER
+5 V
BUFFER
$R_{p-d}$
D11 (LED)
220

Dual polarity
**"Red Output"**,
max 20 V, 8 A

Single polarity
**"Black Output"**,
max 5 V, 250 mA

$R_{p-d}$ is a pull-down resistor 10–100 kΩ 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 (×2) or MCP607 (×1)

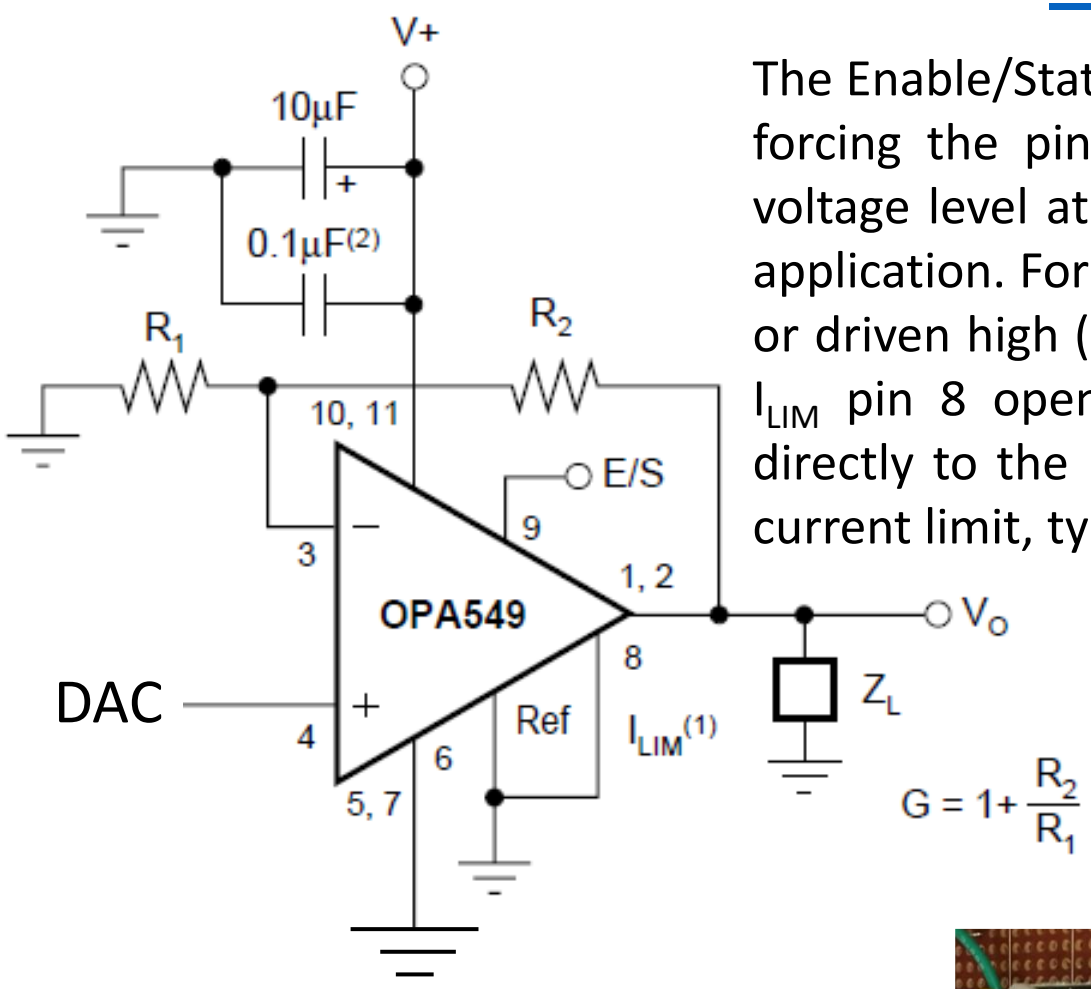
Shield

1-10 µF tantalum at $V_{DD}$ for both op amps

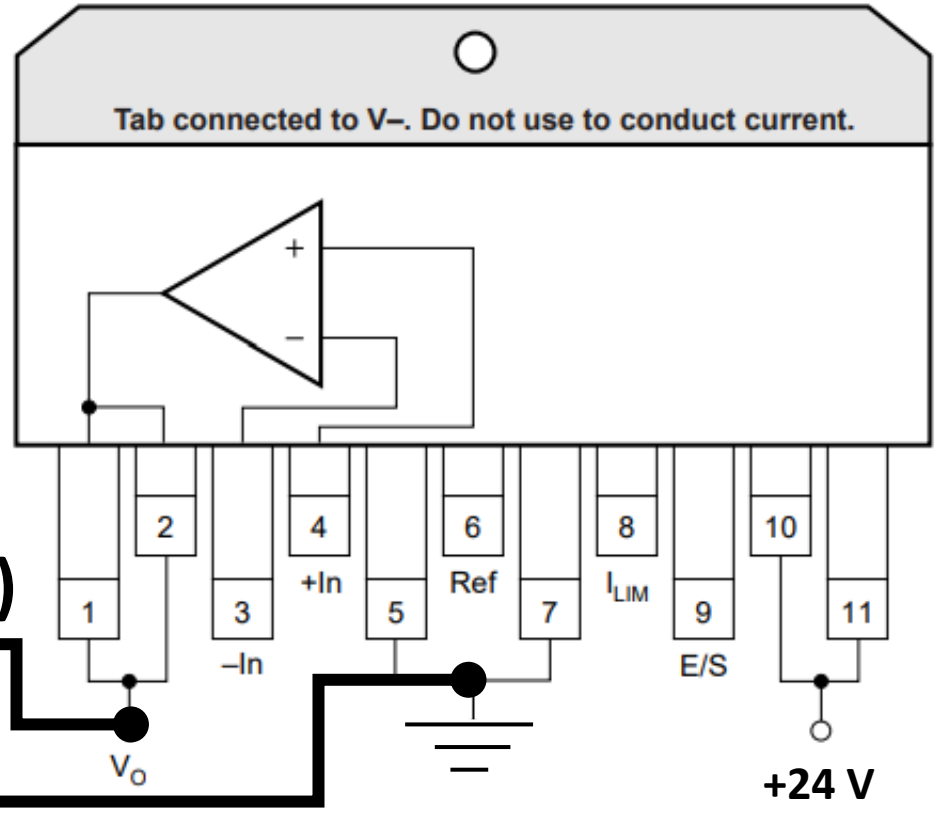
0.1 µF ceramic at $V_{DD}$ for the first op amp

0.1 µF ceramic at $V_{DD}$ for the second op amp

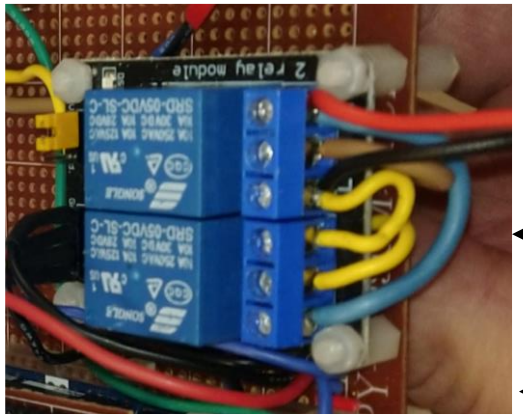
# OPA549 pins




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.

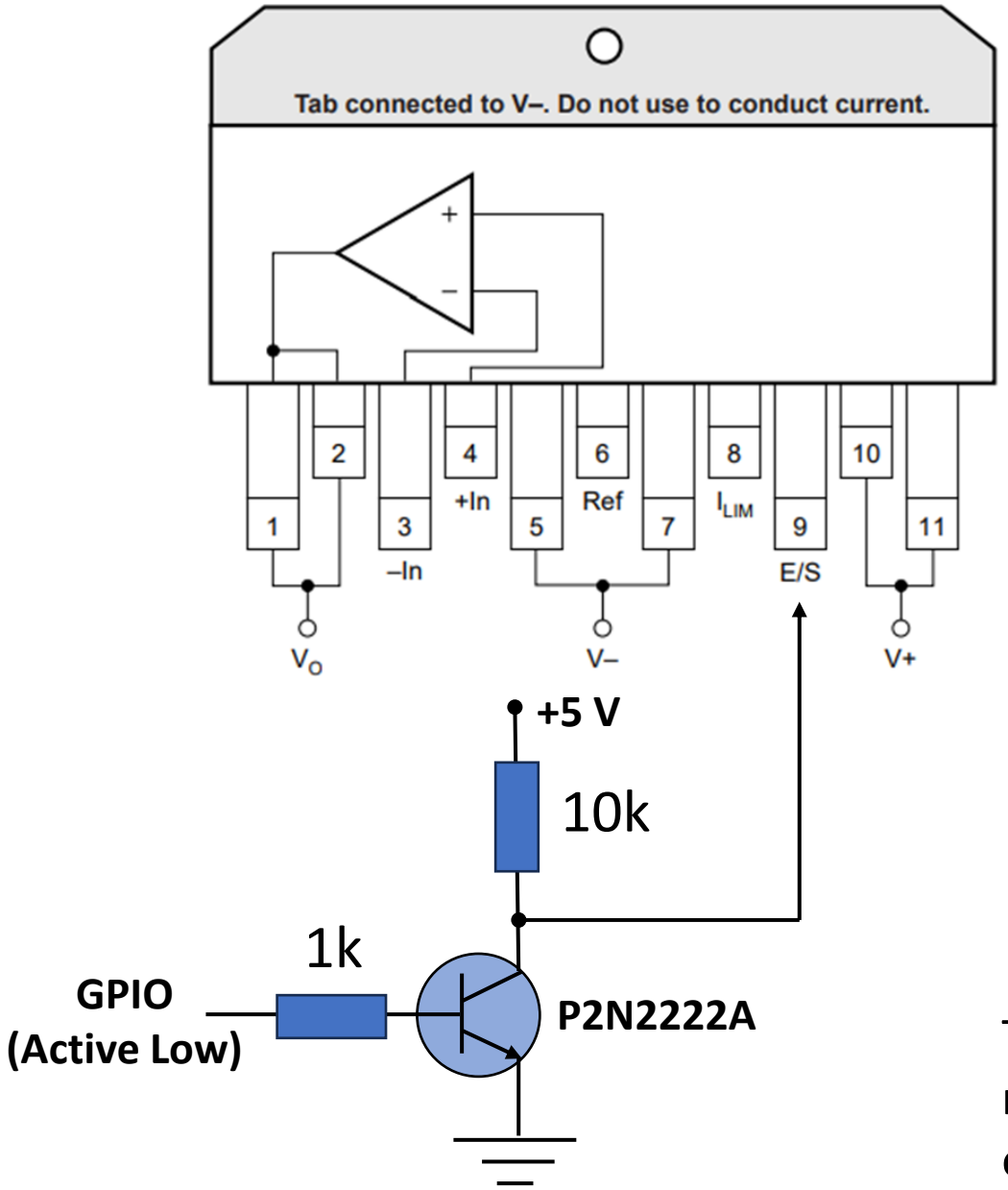


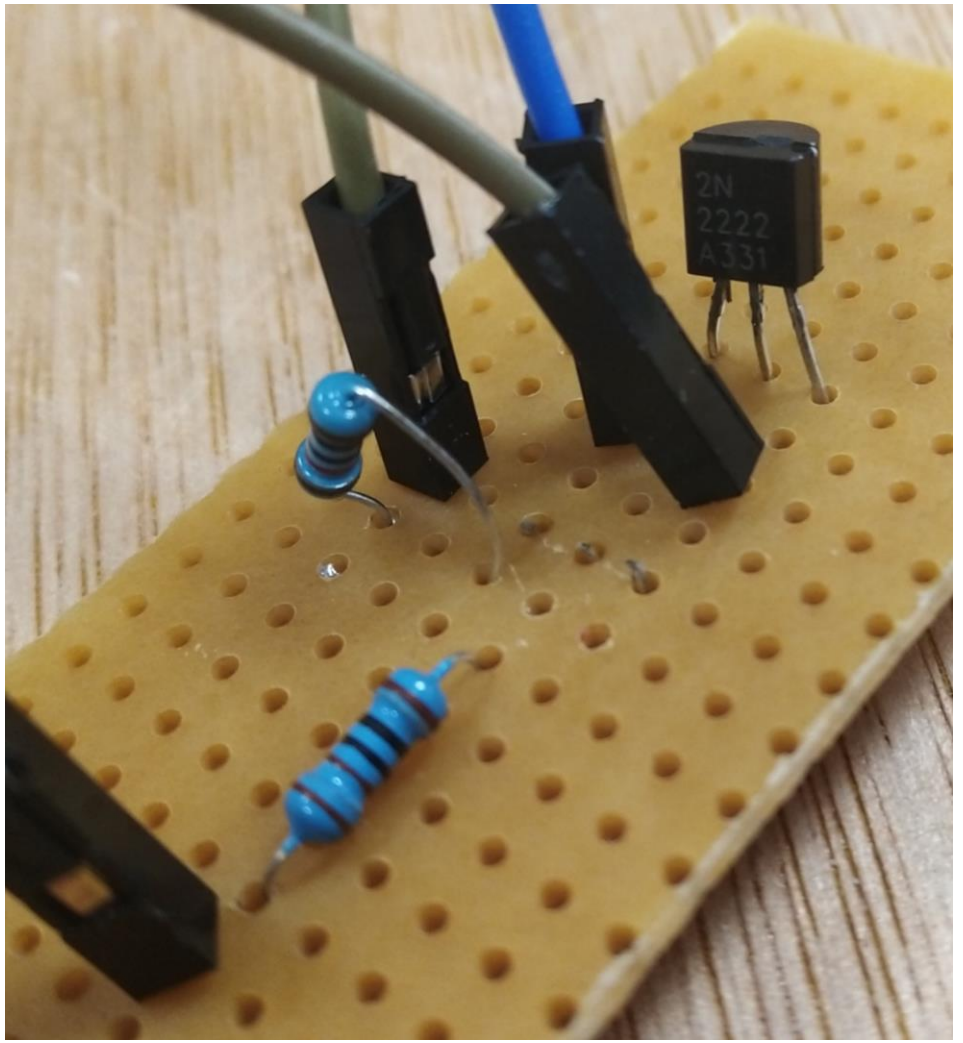

H-bridge



Connect both pins 1 and 2 to output.
Connect both pins 5 and 7 to GND
Connect both pins 10 and 11 to V+.

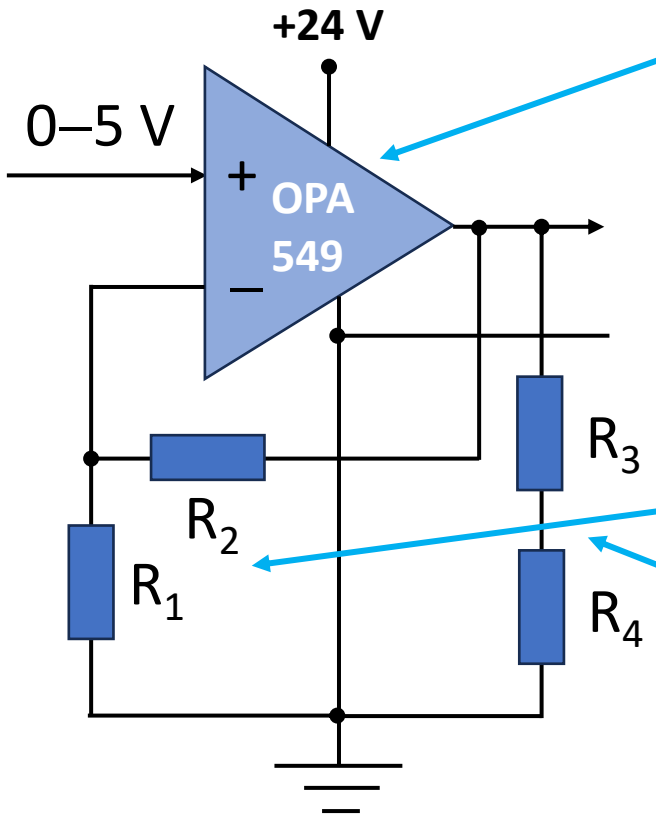
# Digital control of the ES pin 9

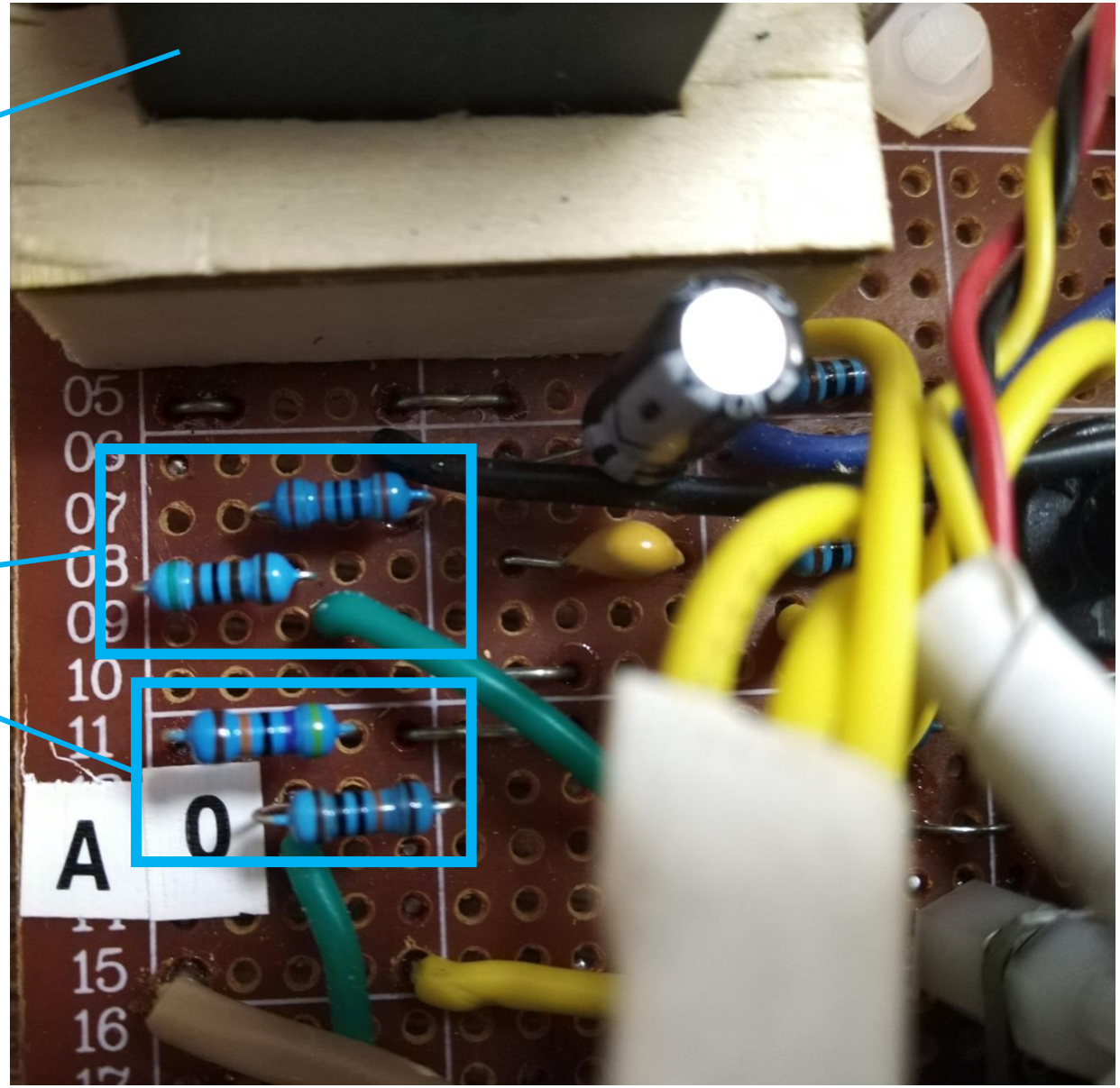


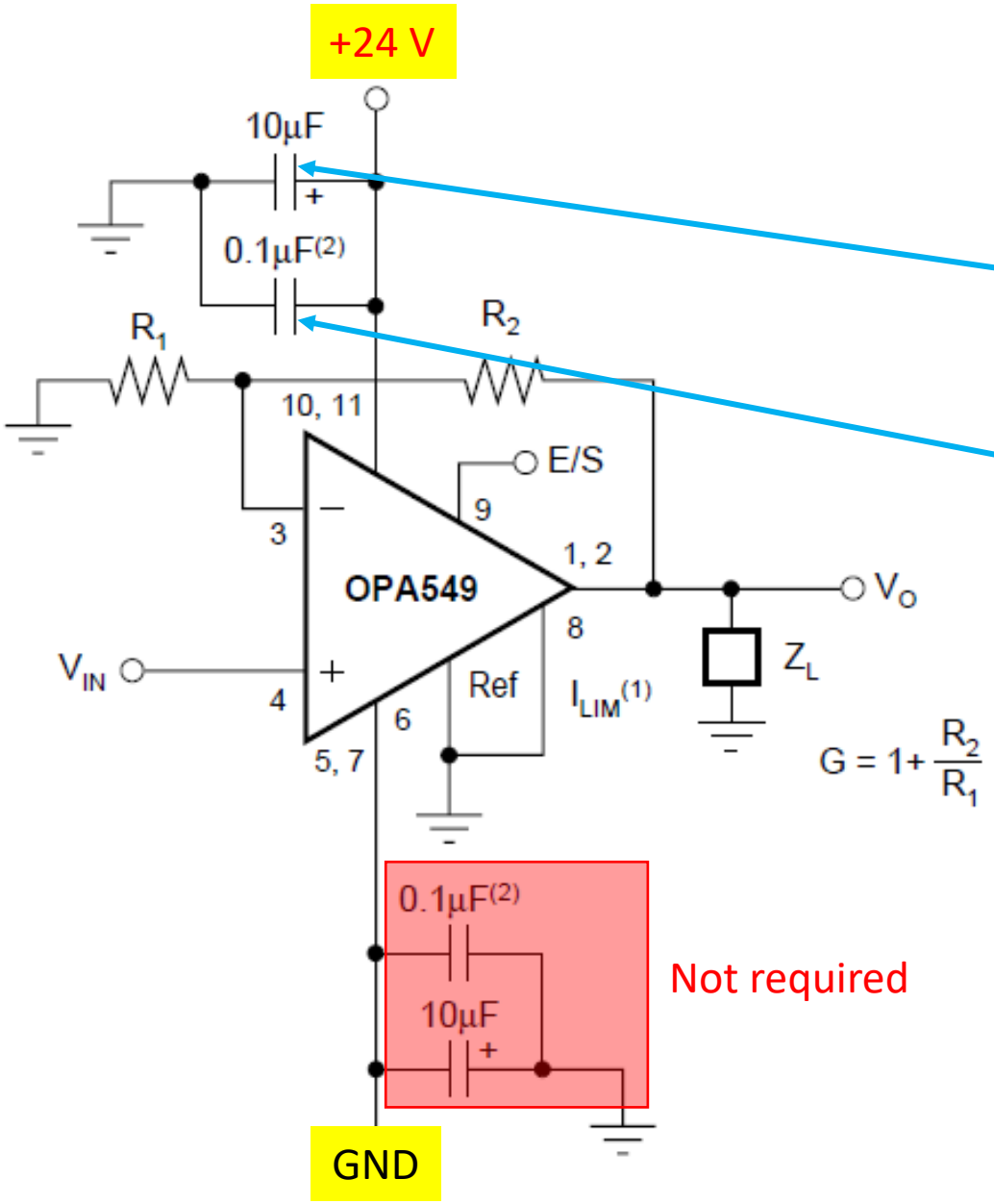

The ES pin 9 can be controlled from a GPIO, but this requires a simple switch using a npn transistor, for example P2N2222A, the schematic of which we provide.

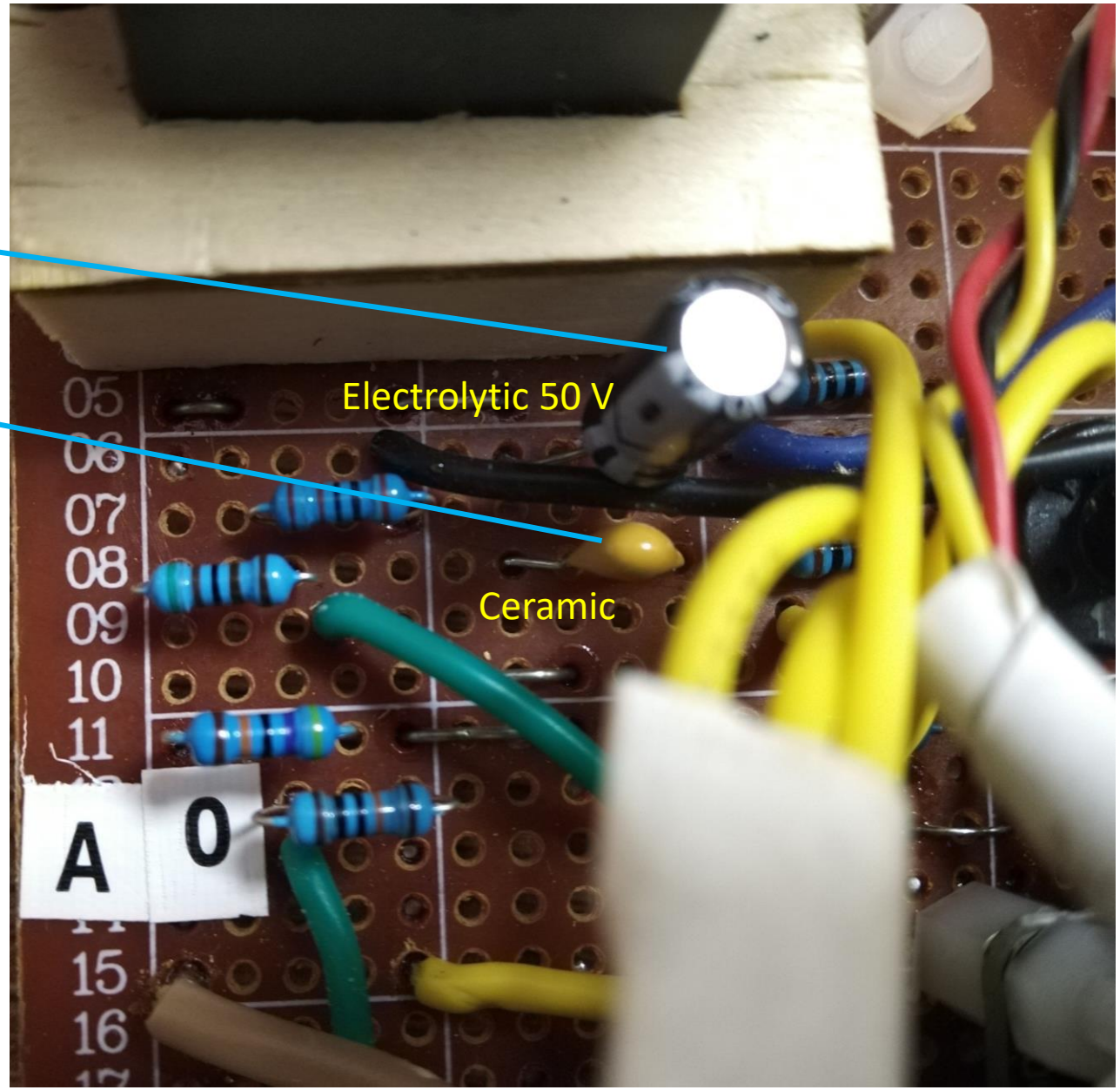
# Resistor networks for the amplifier gain and the voltage divider



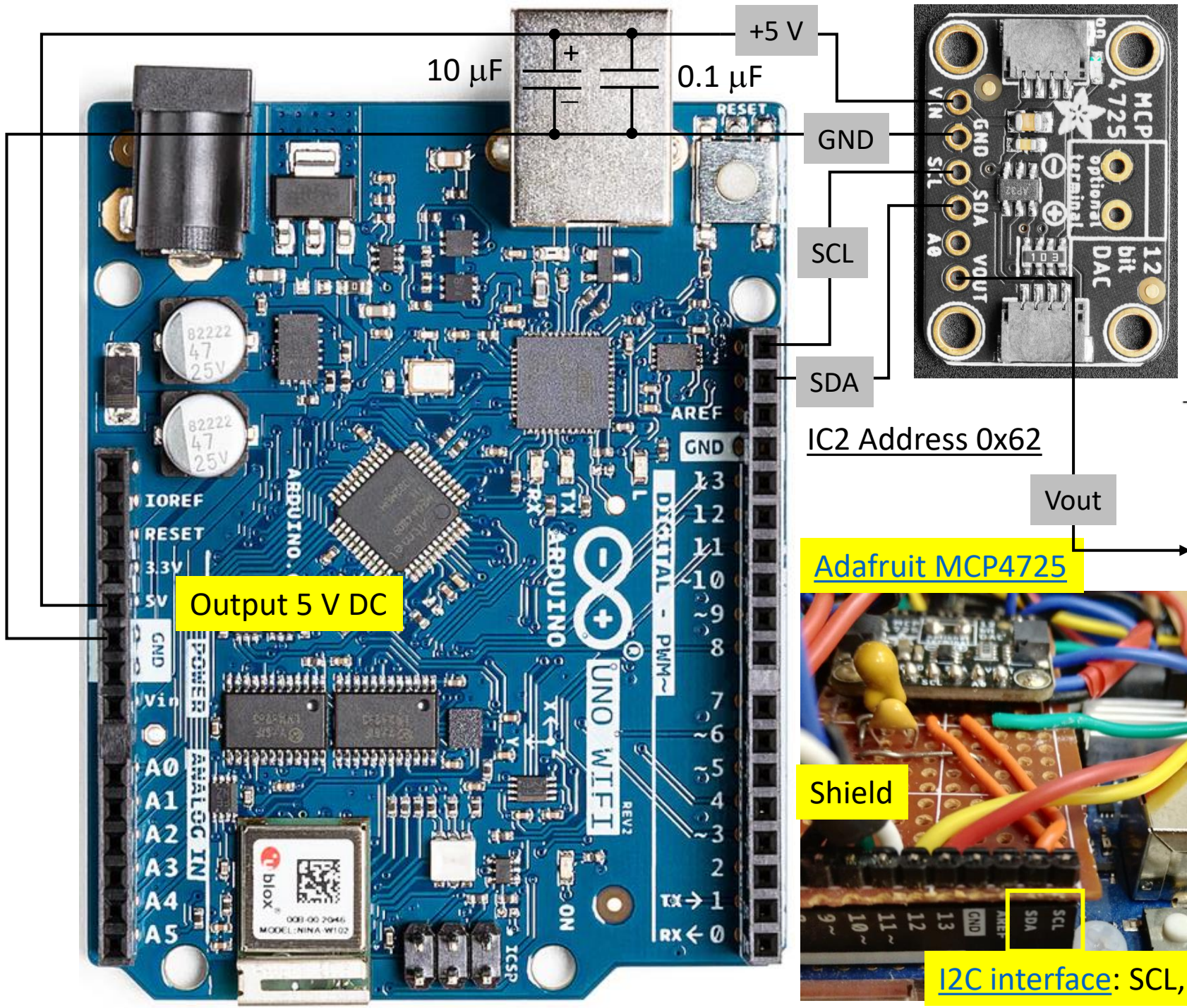



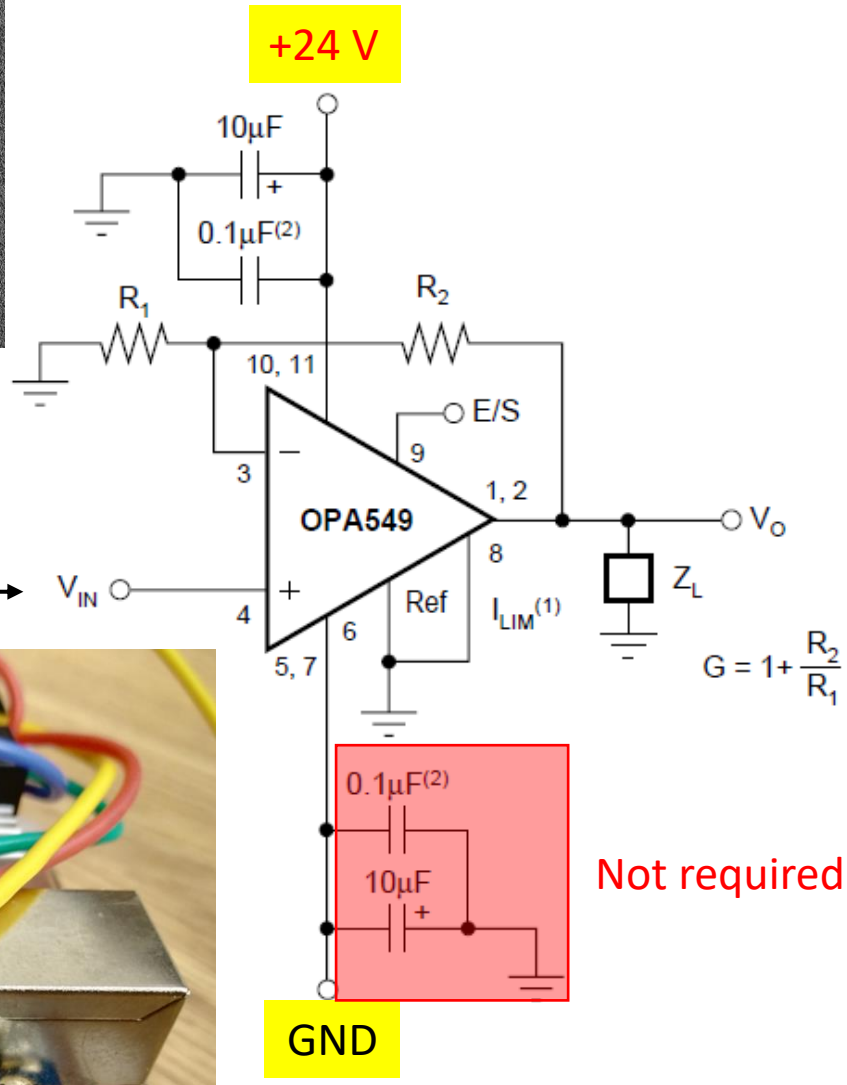
# Source conditioning capacitors



# Connecting Adafruit MCP4725 DAC

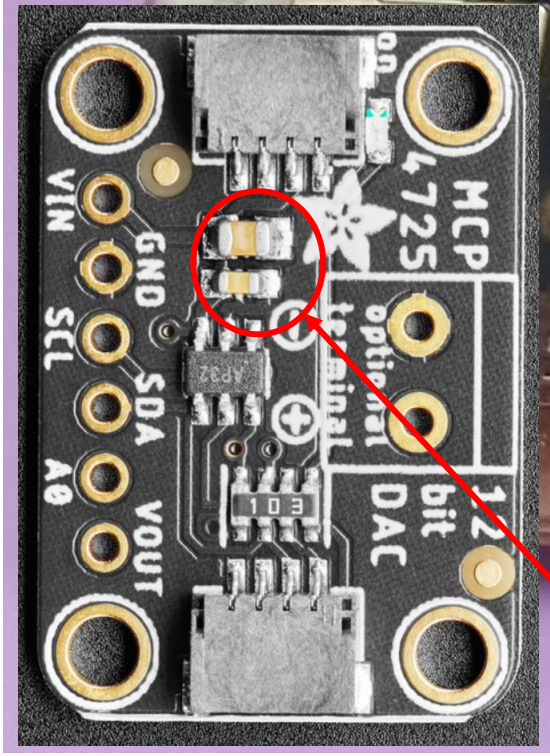


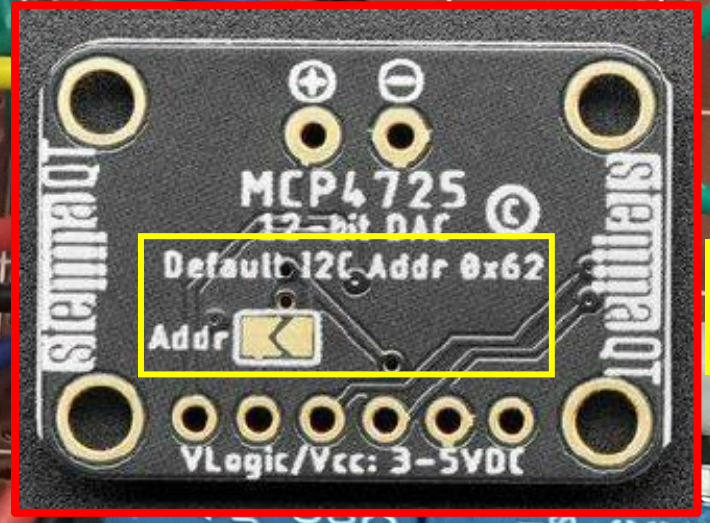




NOTES: (1) $I_{LIM}$ connected to Ref gives the maximum current limit, 10A (peak). (2) Connect capacitors directly to package power-supply pins.

Shield

I2C interface

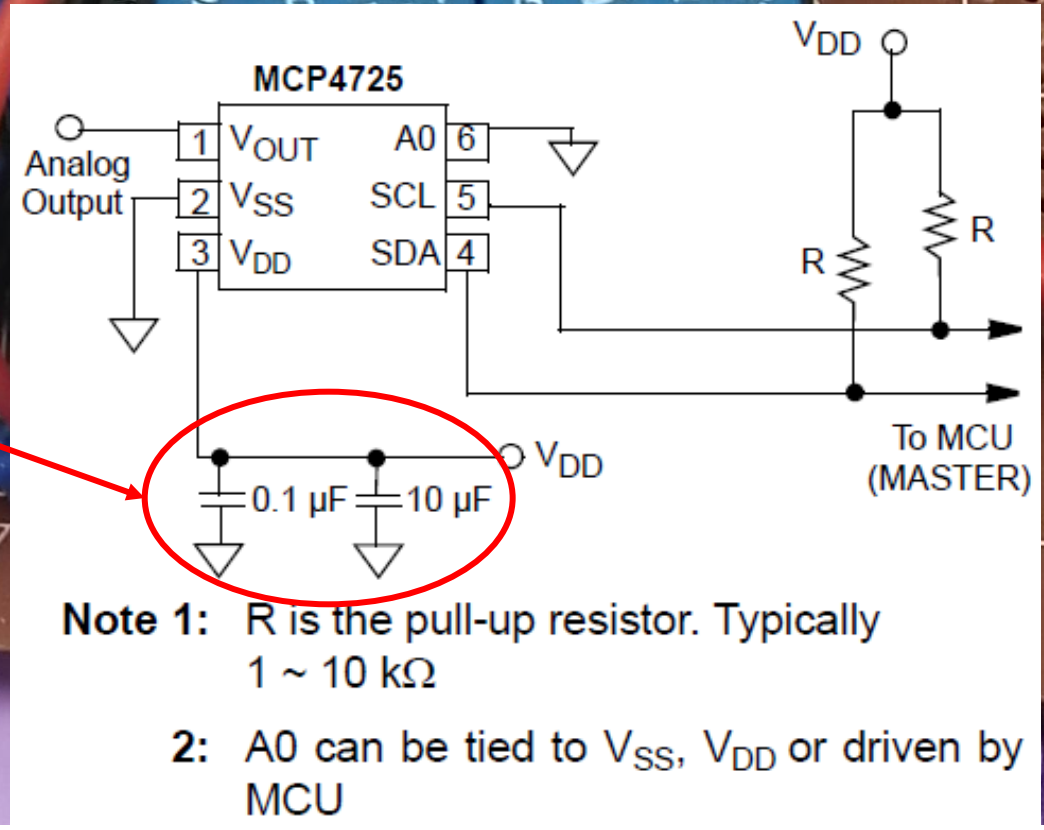



[Using I2C with an Arduino Interface]

External ceramic and tantalum capacitors for $V_{DD}$

The board already contains some capacitors




**Note 1:** R is the pull-up resistor. Typically 1 ~ 10 kΩ

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

# Configuration of OPA549 and Arduino for the RED output

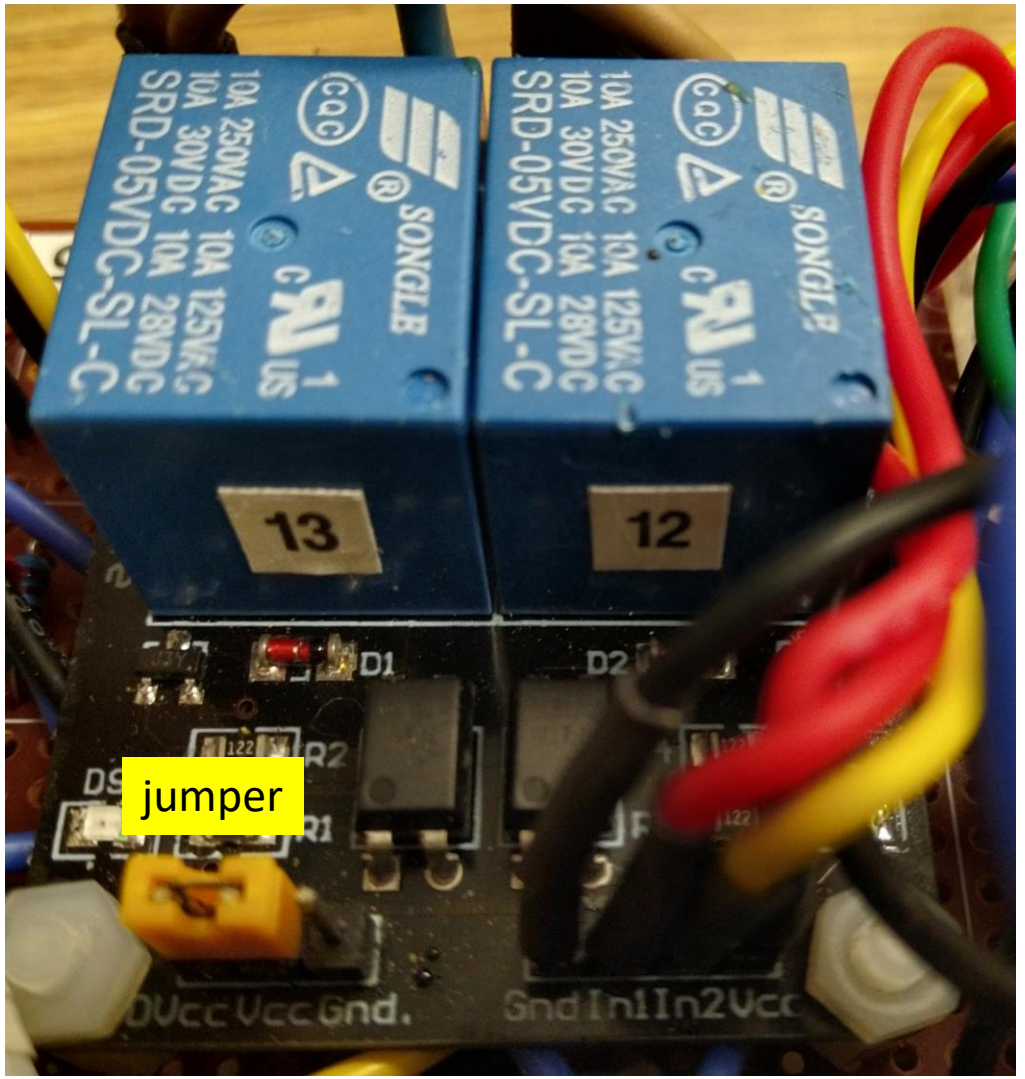
```
// Global constants
const int analogA0 = A0; // Analog pin A0 used for reading the output voltage from the DAC driver
const float max_v1 = 20.0;  // Maximum voltage for channel 1
const float min_v1 = -20.0; // Minimum voltage for channel 1
const float op_amp_gain = 5.9567;  // Gain of OPA549 for 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 $\mathrm{A0}$ 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
}
```

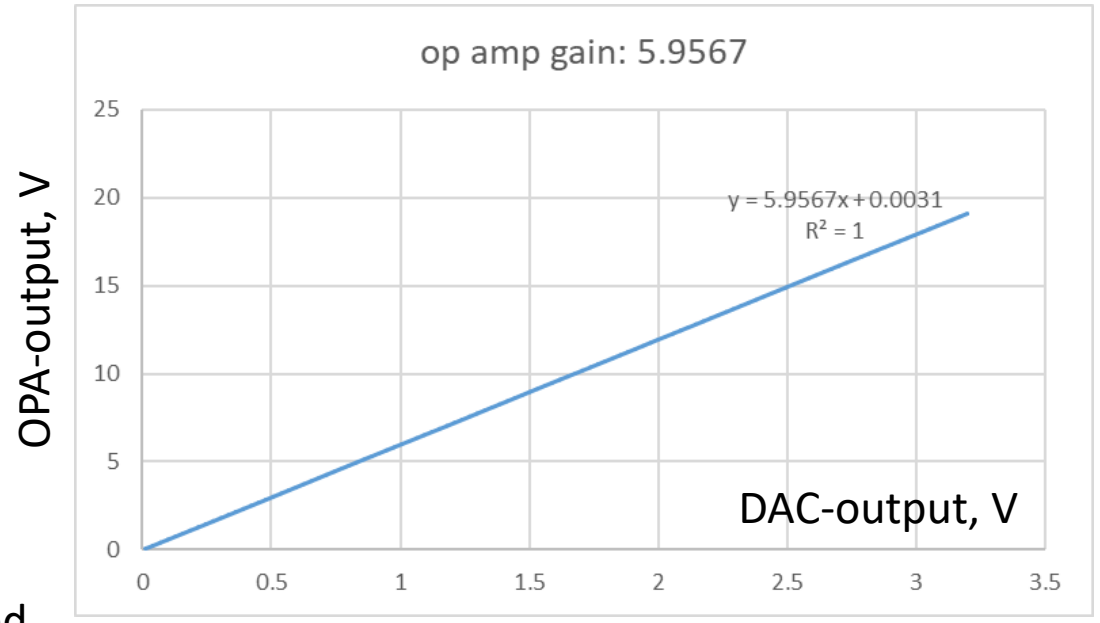


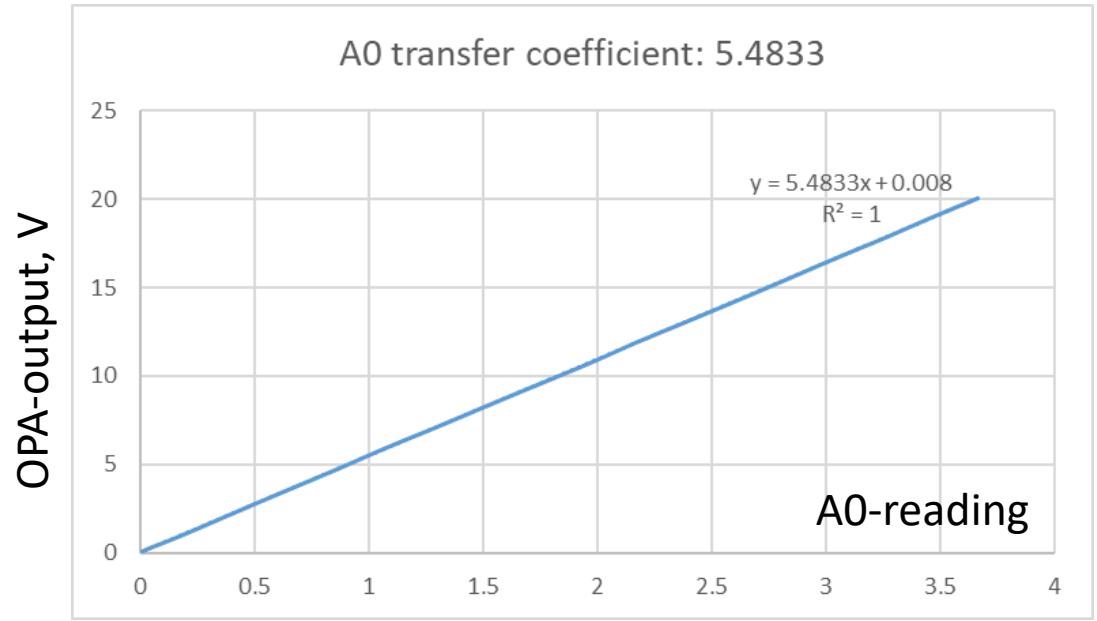
# Testing and calibrating DAC-OPA

Arduino sketch

```ino
#include <Wire.h>
#include <Adafruit_MCP4725.h>

Adafruit_MCP4725 dac;

const int analogPin = A0;

void setup() {
  Wire.begin();
  Serial.begin(9600);

  dac.begin(0x62); // Replace with your actual I2C address (0x62)
}

void loop() {
  // Manually set the desired voltage between 0 and 5 volts for the DAC
  float dacVoltage = 0.01;

  // Convert DAC voltage to DAC value (0-4095)
  uint16_t dacValue = (dacVoltage / 5.0) * 4095;

  // Set the DAC output
  dac.setVoltage(dacValue, false);
  delay(1000);

  // Read the analog voltage from pin A0
  int analogValue = analogRead(analogPin);
  float analogVoltage = (analogValue * 5.0) / 1023.0;
   delay(1000);

  Serial.print("Setting voltage on DAC to: ");
  Serial.print(dacVoltage, 2);
  Serial.println(" V");

  Serial.print("Analog voltage: ");
  Serial.print(analogVoltage, 2);
  Serial.println(" V");

  delay(1000); // Wait for 10 seconds before updating again
}
```

OPA-output was measured using a precise voltmeter

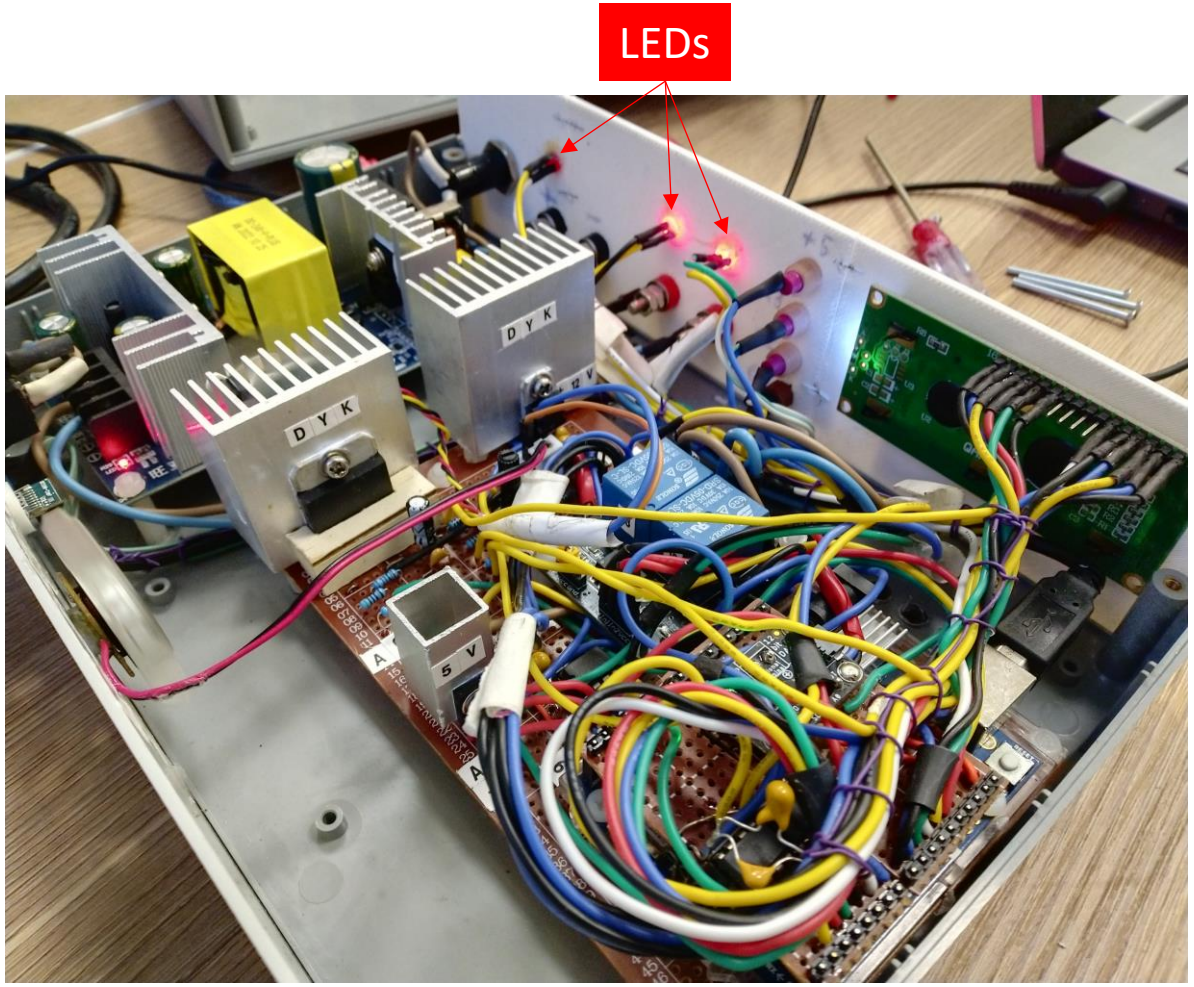


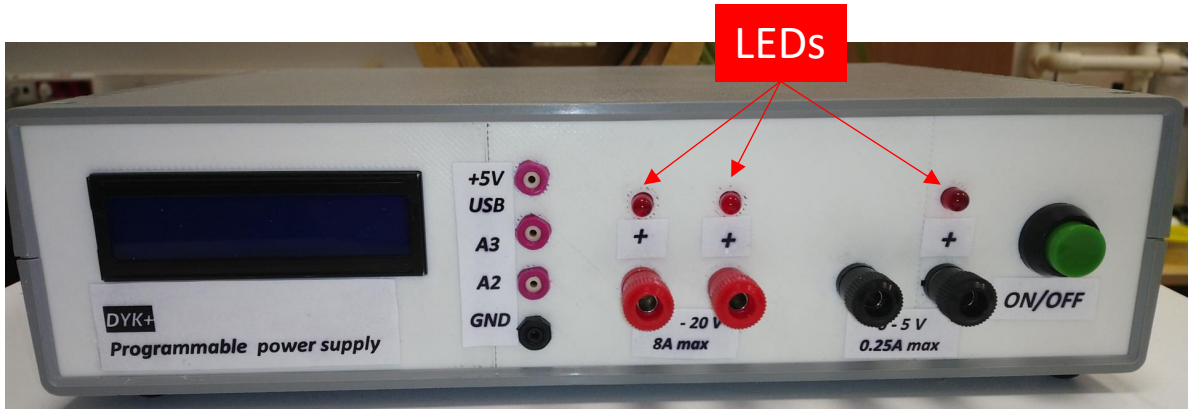



# Testing LEDs

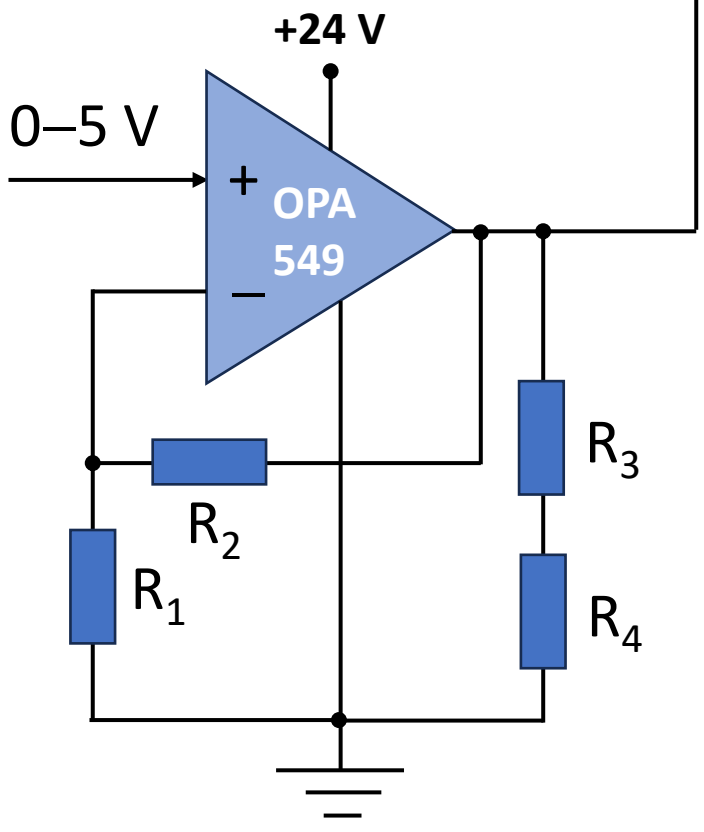
Arduino sketch

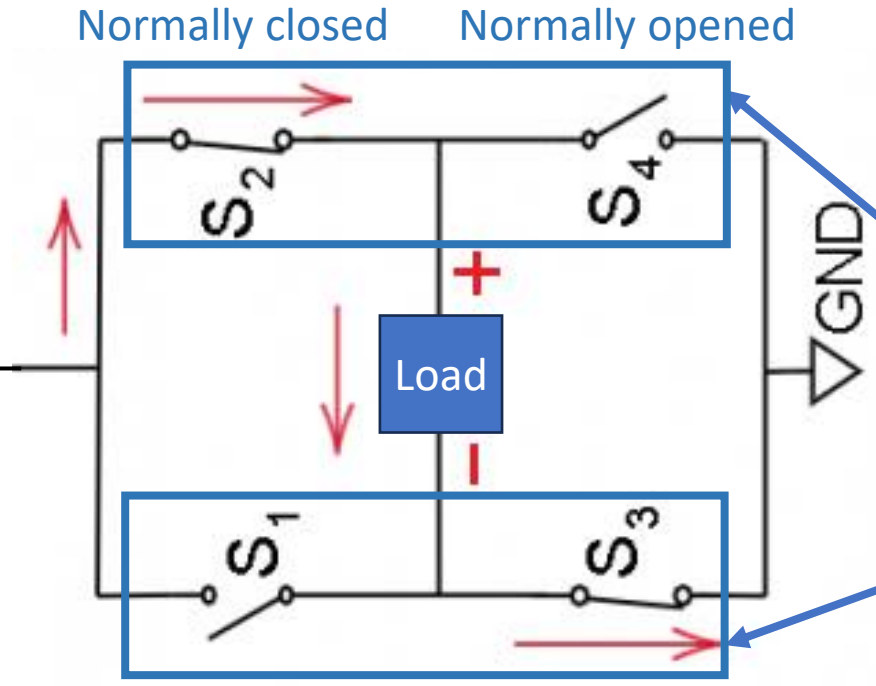
```
// Define the LED pins
const int ledPin1 = 8;
const int ledPin2 = 10;
const int ledPin3 = 11;

void setup() {
  // Initialize the LED pins as OUTPUT
  pinMode(ledPin1, OUTPUT);
  pinMode(ledPin2, OUTPUT);
  pinMode(ledPin3, OUTPUT);
}

void loop() {
  // Turn on the LEDs
  digitalWrite(ledPin1, HIGH);
  digitalWrite(ledPin2, HIGH);
  digitalWrite(ledPin3, HIGH);

  // Wait for a period
  delay(1000); // Adjust the delay time (in milliseconds) as needed

  // Turn off the LEDs
  digitalWrite(ledPin1, LOW);
  digitalWrite(ledPin2, LOW);
  digitalWrite(ledPin3, LOW);

  // Wait for another period
  delay(1000); // Adjust the delay time (in milliseconds) as needed
}
```

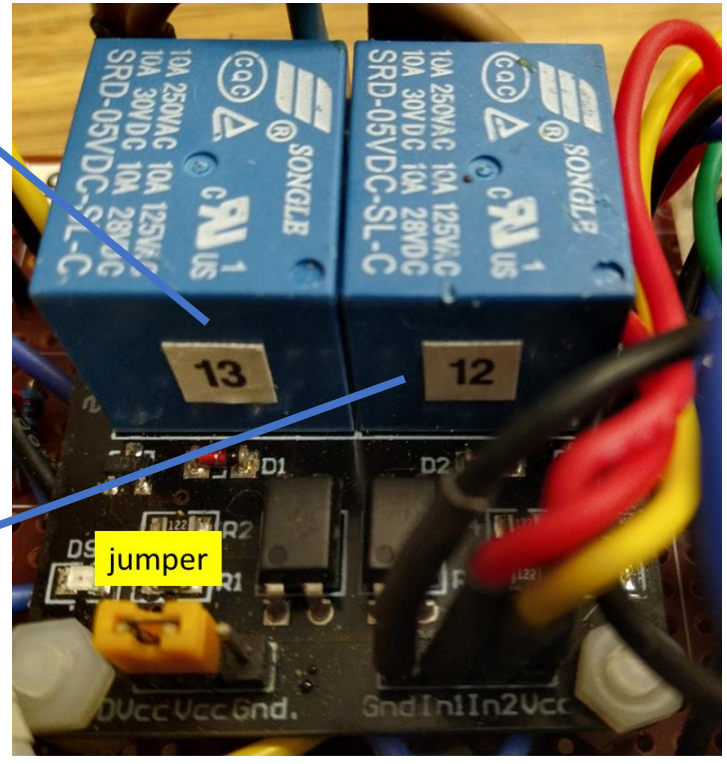


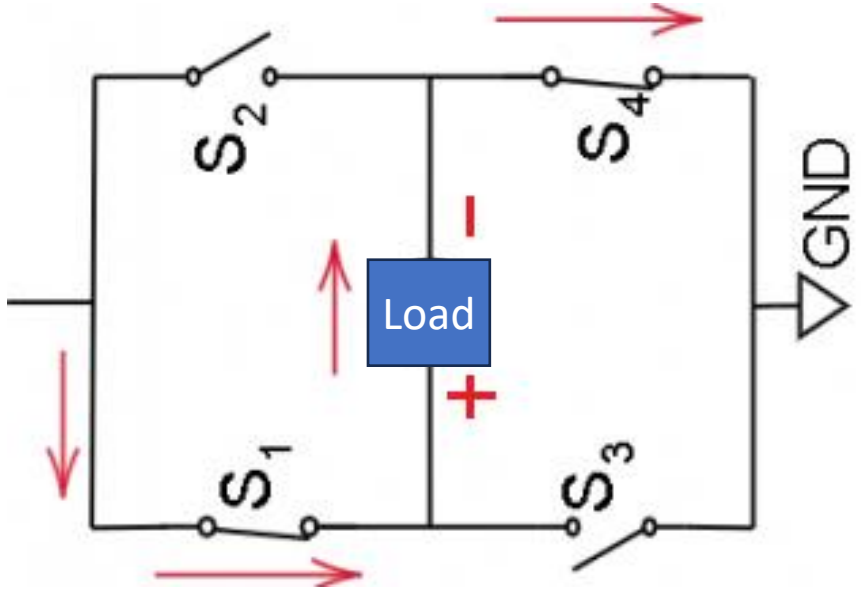
# Relay H-bridge: operation principle



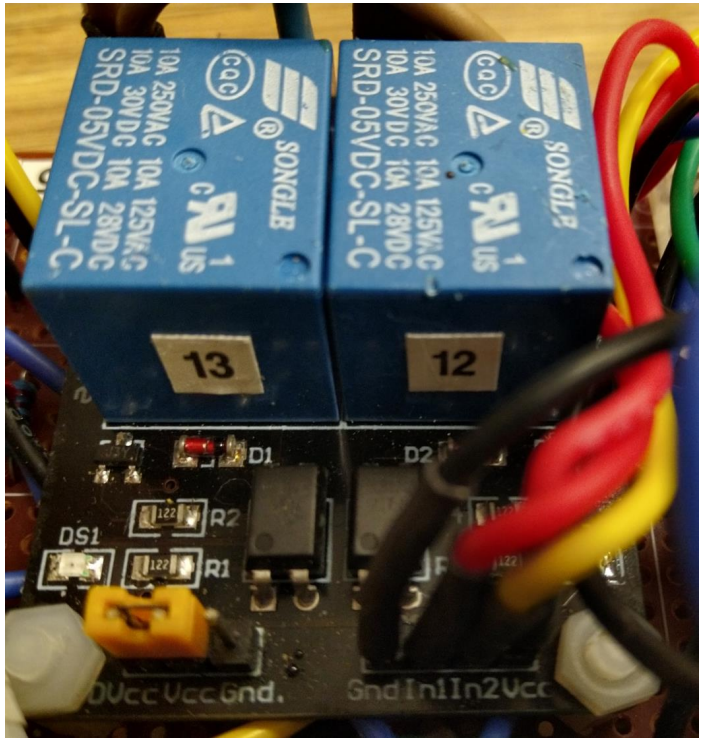

GND | D13 | D12 | 5 V




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.

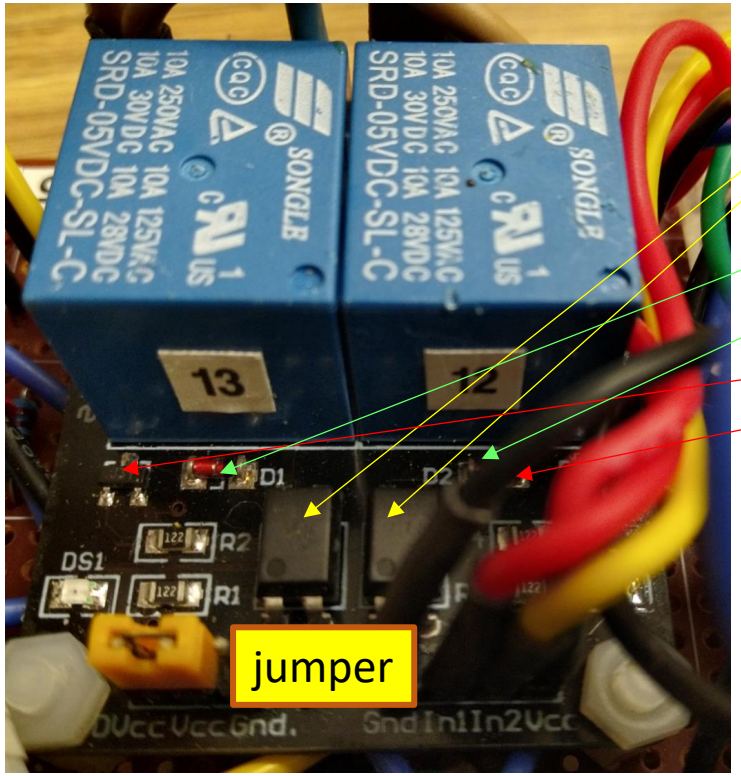
# Control of the voltage polarity in the Arduino sketch for switching the H-bridge

```
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";
    }
  }
```

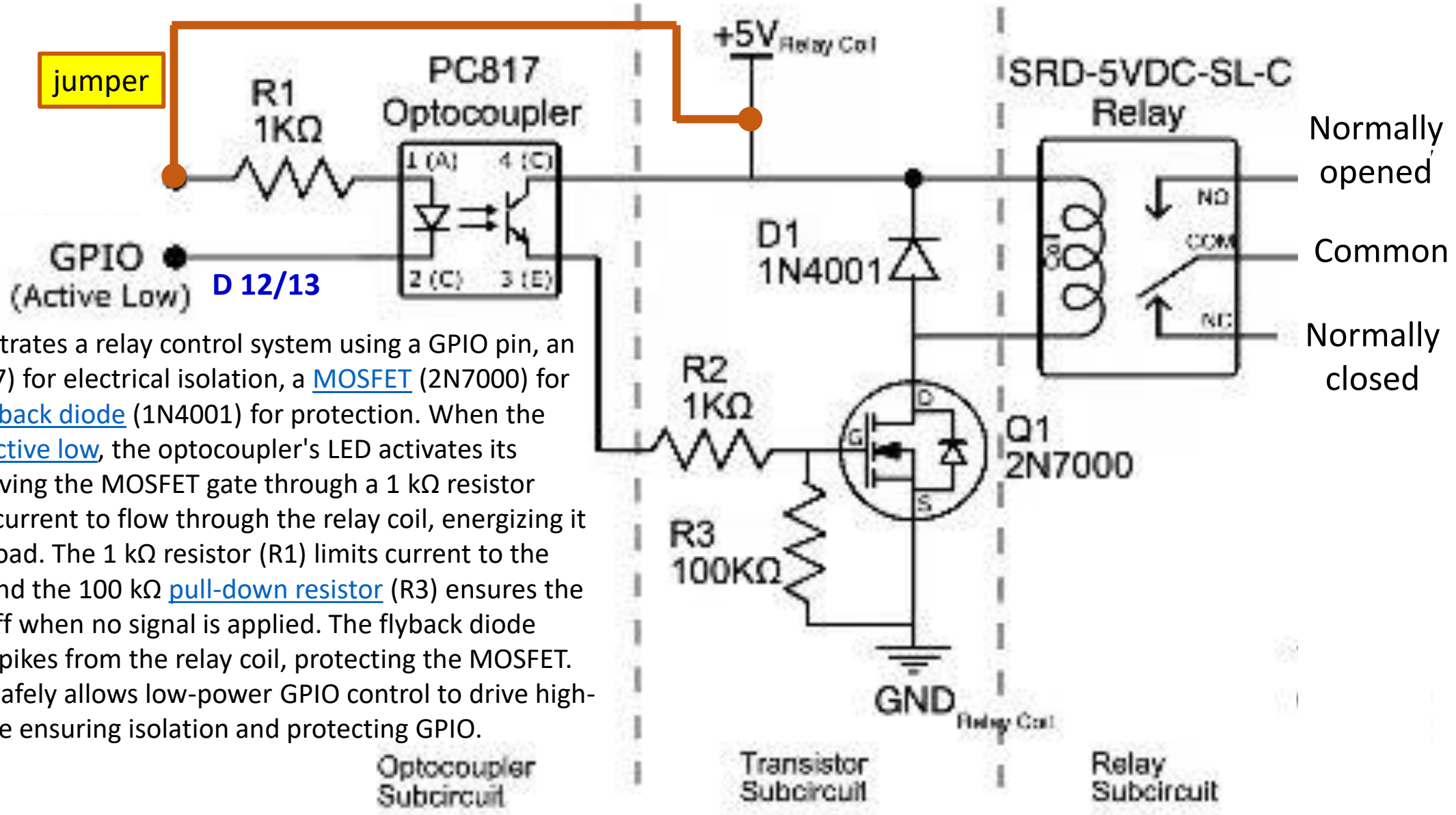



When setting a new voltage value on the Red output, its polarity is checked relative to the previous sign (+/-). If the polarities differ and the new voltage value is non-zero, the output is first reset to zero before switching polarity and applying the new value. This precaution is taken to prevent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

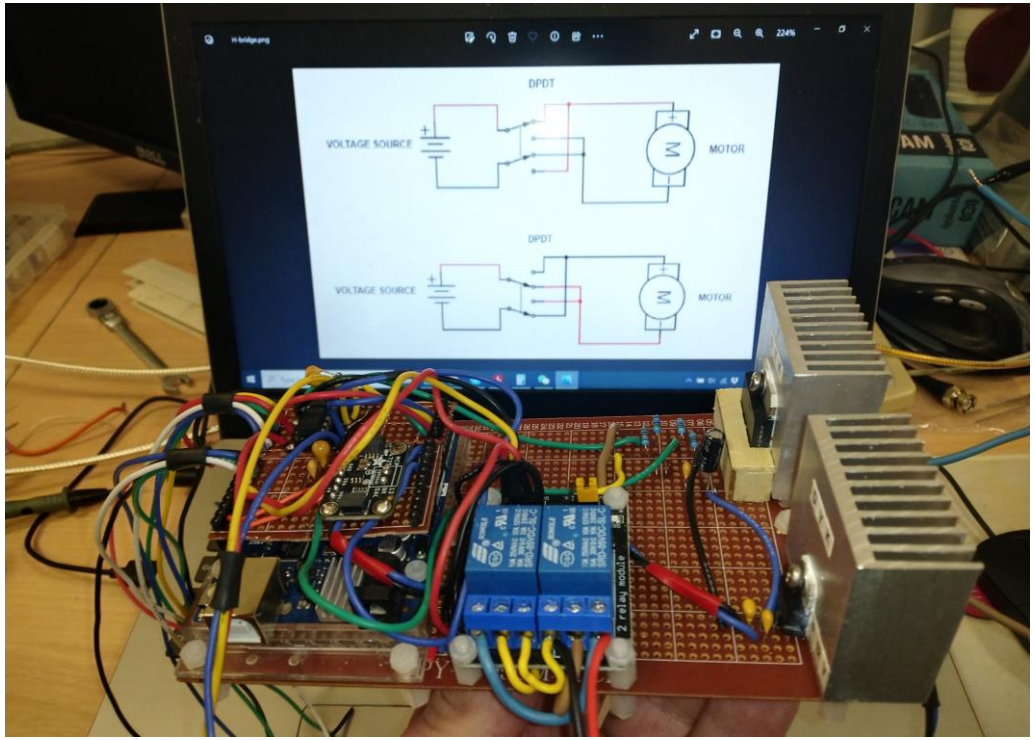


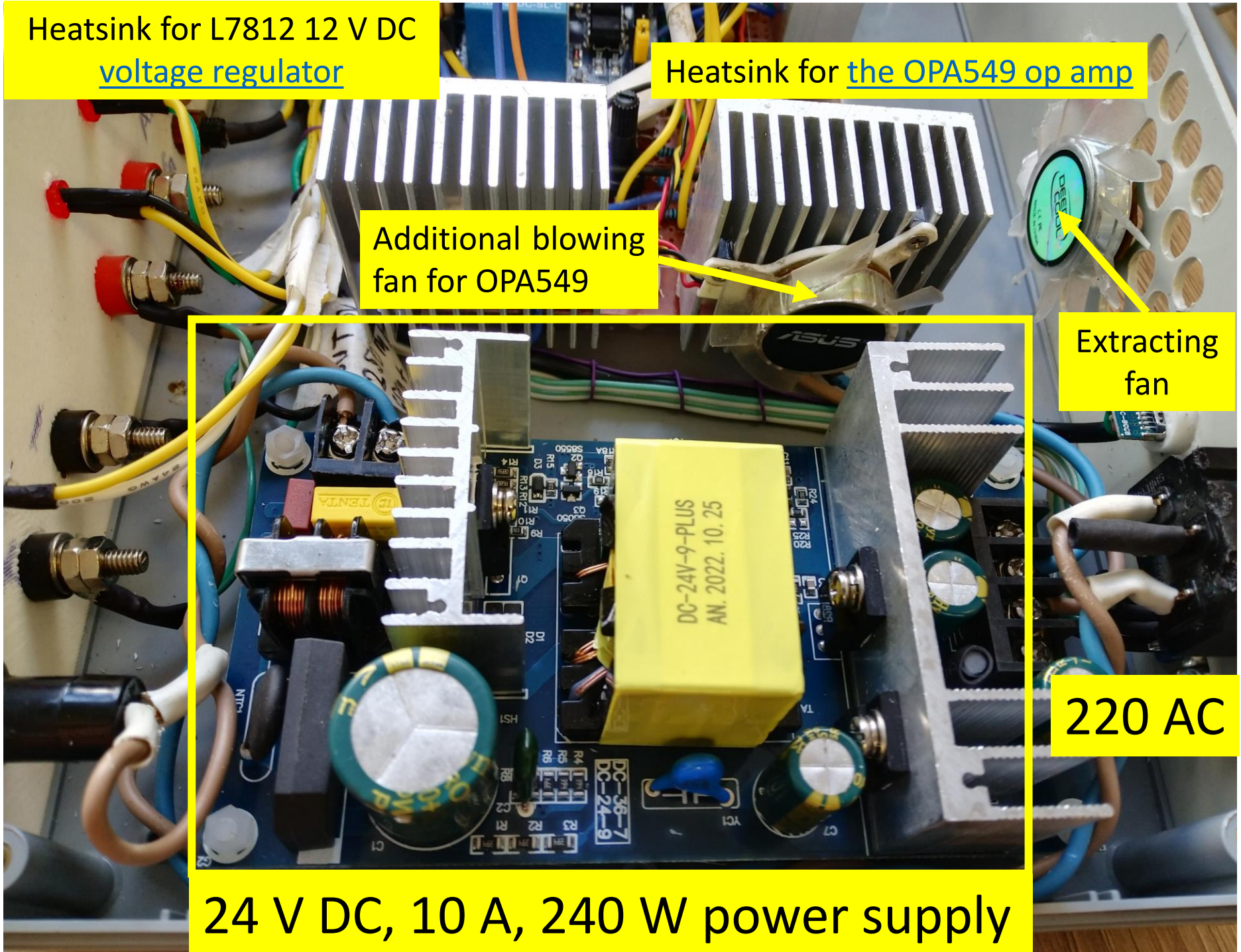

This circuit demonstrates a relay control system using a GPIO pin, an optocoupler (PC817) for electrical isolation, a MOSFET (2N7000) for switching, and a flyback diode (1N4001) for protection. When the GPIO pin is set to active low, the optocoupler's LED activates its phototransistor, driving the MOSFET gate through a 1 kΩ resistor (R2), which allows current to flow through the relay coil, energizing it and switching the load. The 1 kΩ resistor (R1) limits current to the optocoupler LED, and the 100 kΩ pull-down resistor (R3) ensures the MOSFET remains off when no signal is applied. The flyback diode dissipates voltage spikes from the relay coil, protecting the MOSFET. This configuration safely allows low-power GPIO control to drive high-power devices while ensuring isolation and protecting GPIO.

# 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





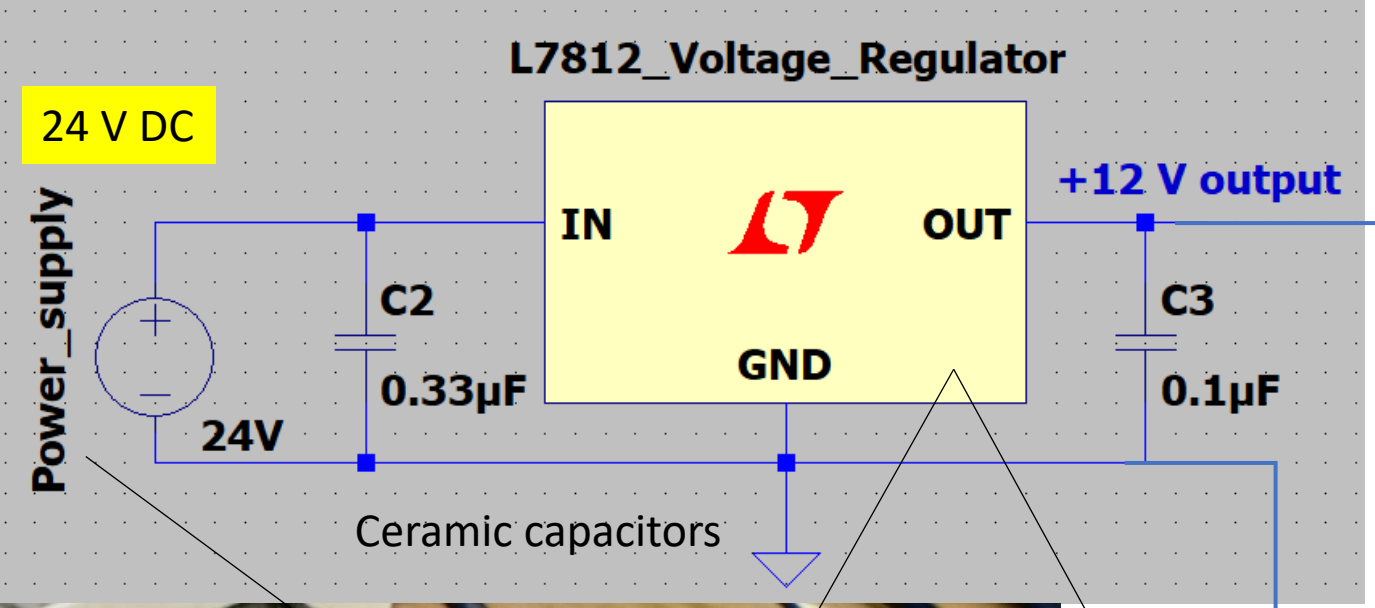

# Power management

**+24 V for OPA549**

**+12 V for Arduino**

**+5 V for:**

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




9 V < Vin < 12 V

Voltage Regulator 5 V (reference)

Tantalum and ceramic capacitors

Vin

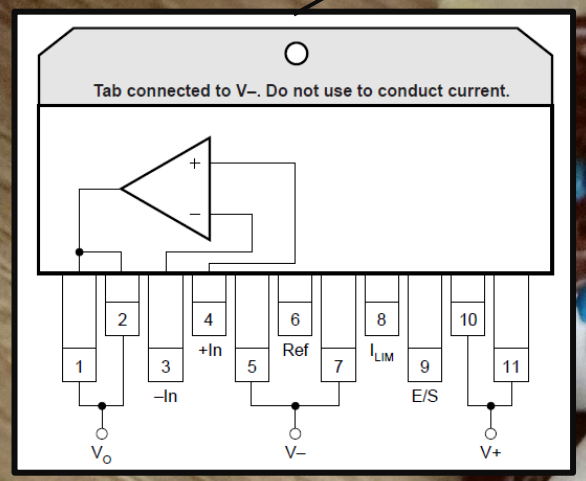
GND

Outputs 5 V and 3.3 V DC

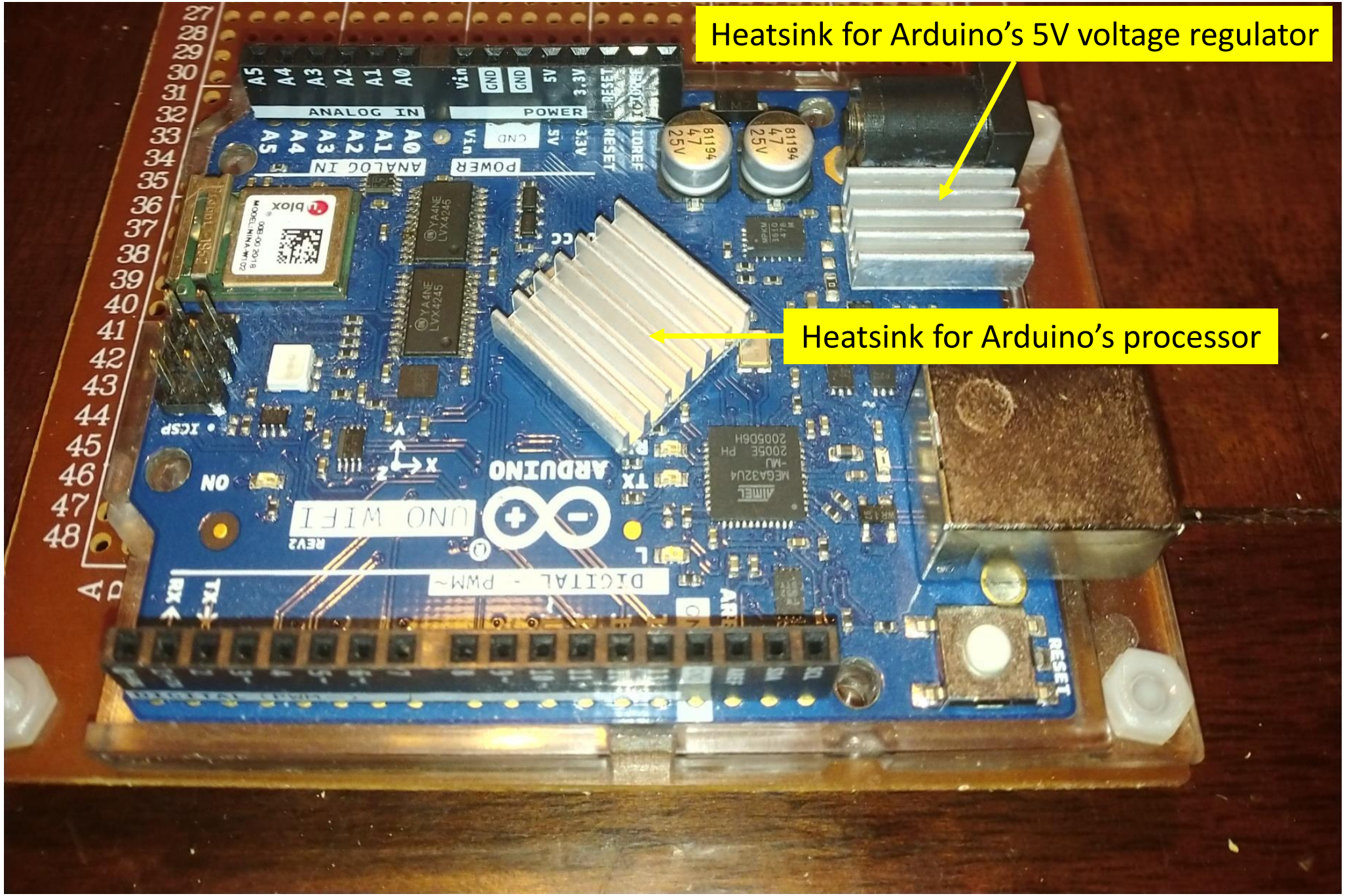
Input 12 V DC


Ceramic capacitors

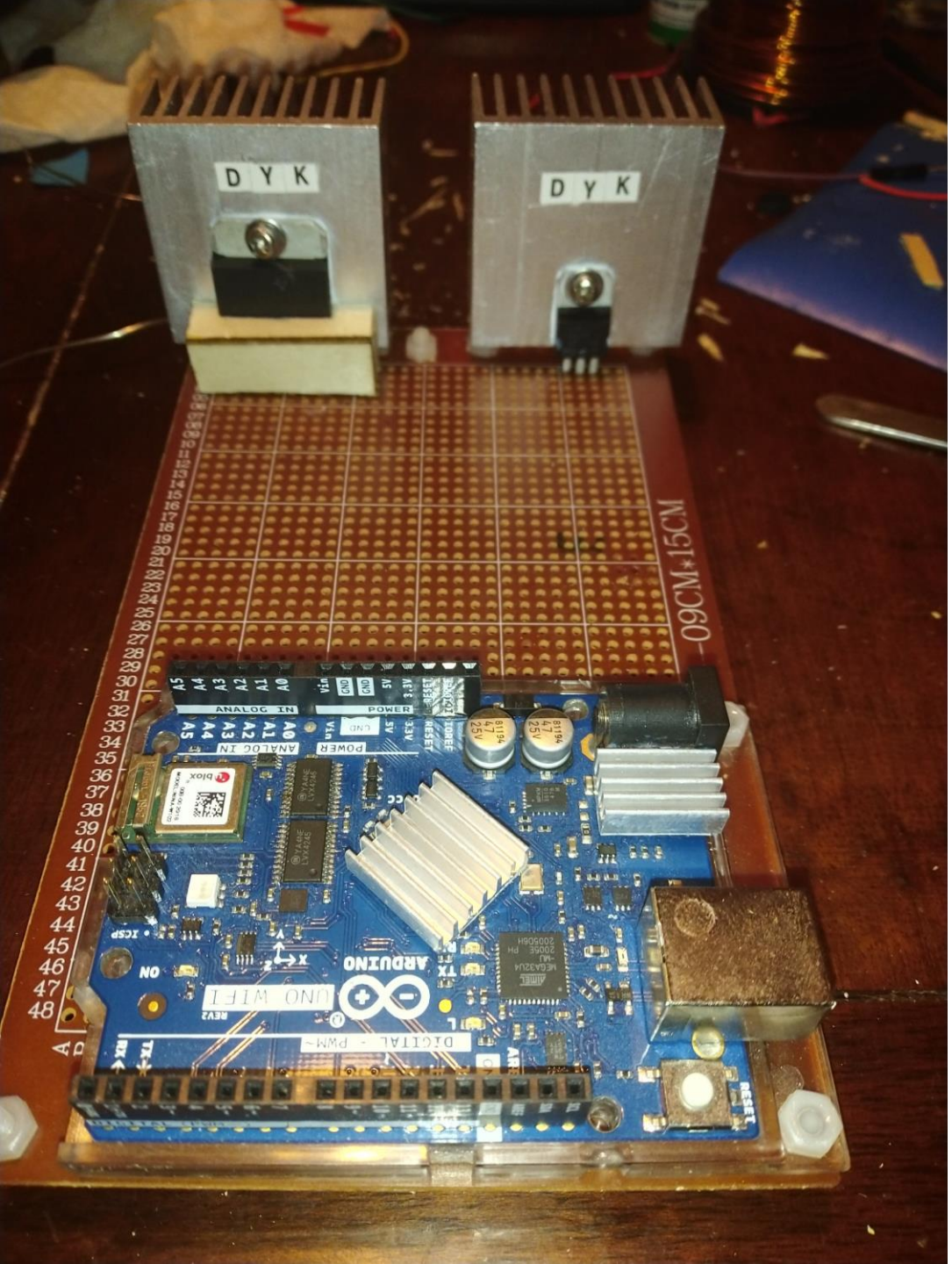
OPA549 op amp

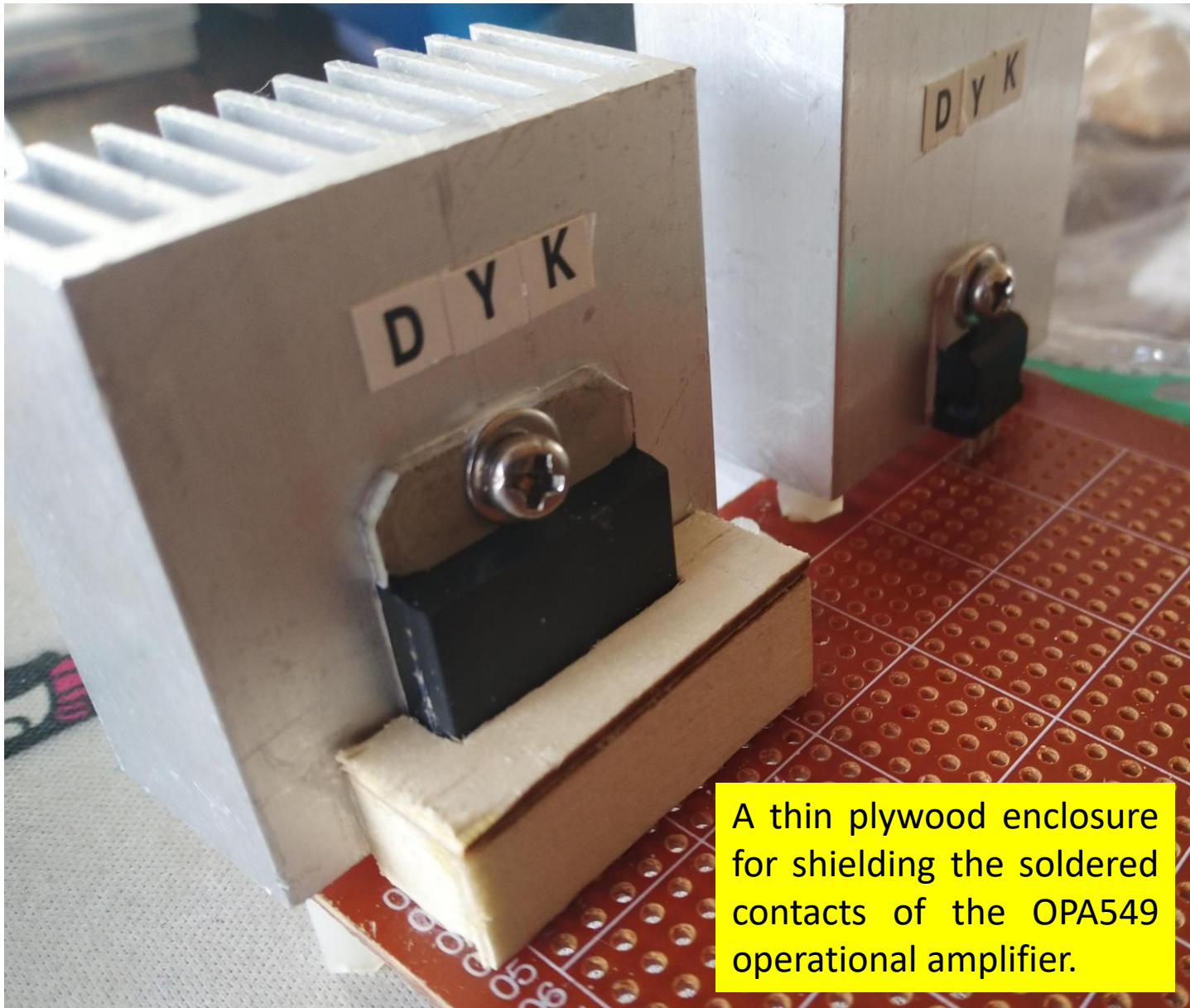



Additional blowing fan for OPA549


Heatsink for Arduino’s 5V voltage regulator
Heatsink for Arduino’s processor

# Heatsinks on OPA, 12 V voltage regulator, and Arduino

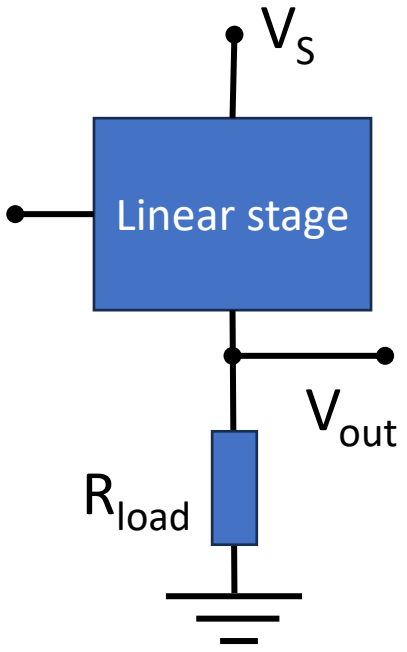



A thin plywood enclosure for shielding the soldered contacts of the OPA549 operational amplifier.

# 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 $T_J$ (°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 (°C/W), $T_A$ is the ambient (room) temperature (°C), $P_D$ is the dissipated power (W), $\theta_{JC}$ is the junction-to-case thermal resistance (°C/W), $\theta_{CH}$ is the case-to-heatsink thermal resistance (°C/W), and $\theta_{HA}$ is the heatsink-to-air thermal resistance (°C/W). In the absence of a heatsink, we would have: $\theta_{JA} = \theta_{JC} + \theta_{CA}$, where $\theta_{CA}$ is the case-to-air thermal resistance (°C/W). For example, for a P2N2222A transistor in a plastic case, the thermal resistance $\theta_{JA}$ is 200 °C/W. Such a high value of $\theta_{JA}$ does not allow the transistor to dissipate more than 0.5 W of power for $T_J = 125$ °C and $T_A = 25$ °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 $\theta_{CH}$ can usually be neglected if a special thermal grease is applied between the casing and the heatsink. The coefficient $\theta_{HA}$ 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

Air Flow: 169.5 CFM

Online calculator

Rectangle Duct

H: W: in

Circular Duct

R: 60 mm

Calculate Clear

Results:

1392.32 ft/min (LFM)

/s

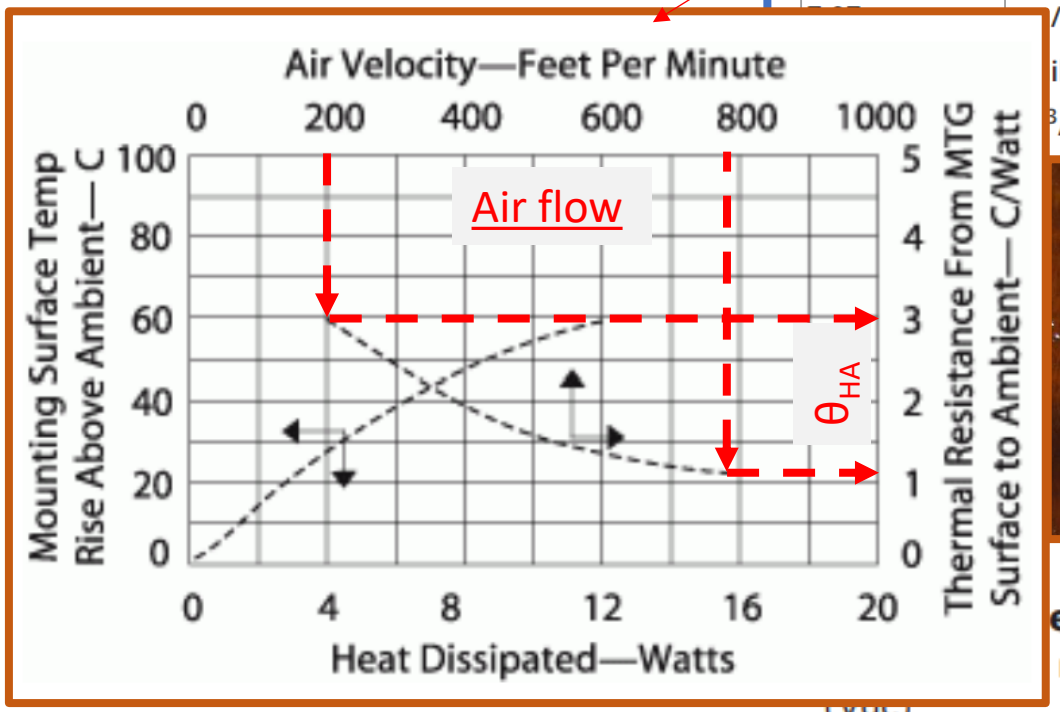
iles/hr (MPH)

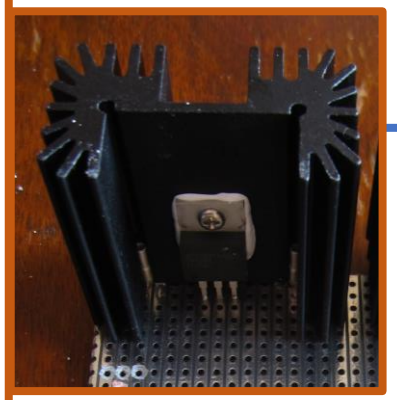
/min (CFM)

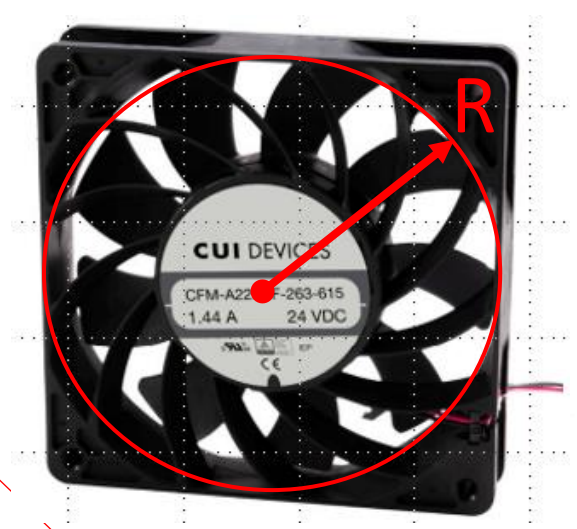
$$\text{LFM} = \frac{\text{CFM}}{\text{Cross–Sectional Area}}$$

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

Transfer R to feet!






| | | e range (Vdc) | input current¹ max (A) | input power¹ max (W) | rated speed¹ typ (RPM±10%) | airflow² (CFM) | static pressure³ (inch $H_2O$) | r |
|---|---|---|---|---|---|---|---|---|
| CFM-A225BF-153-577 | 12 | 10.8~13.2 | 1.87 | 22.44 | 5,300 | 169.5 | 0.82 | |

# Heat management

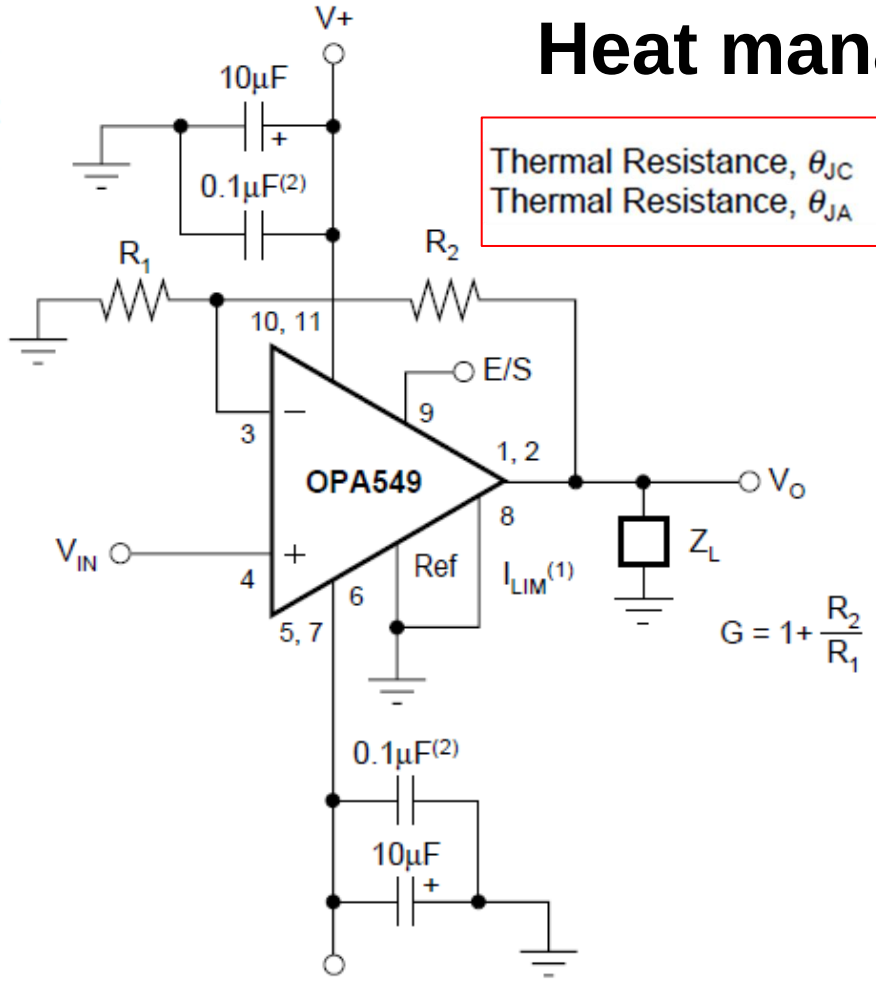
Most applications require a heat sink to assure that the maximum operating junction temperature (125°C) is not exceeded. In addition, the junction temperature should be kept as low as possible for increased reliability. Junction temperature can be determined according to the Equations:

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

where

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

- $T_J$ = Junction Temperature (°C)
- $T_A$ = Ambient Temperature (°C)
- $P_D$ = Power Dissipated (W)
- $\theta_{JC}$ = Junction-to-Case Thermal Resistance (°C/W)
- $\theta_{CH}$ = Case-to-Heat Sink Thermal Resistance (°C/W)
- $\theta_{HA}$ = Heat Sink-to-Ambient Thermal Resistance (°C/W)
- $\theta_{JA}$ = Junction-to-Air Thermal Resistance (°C/W)

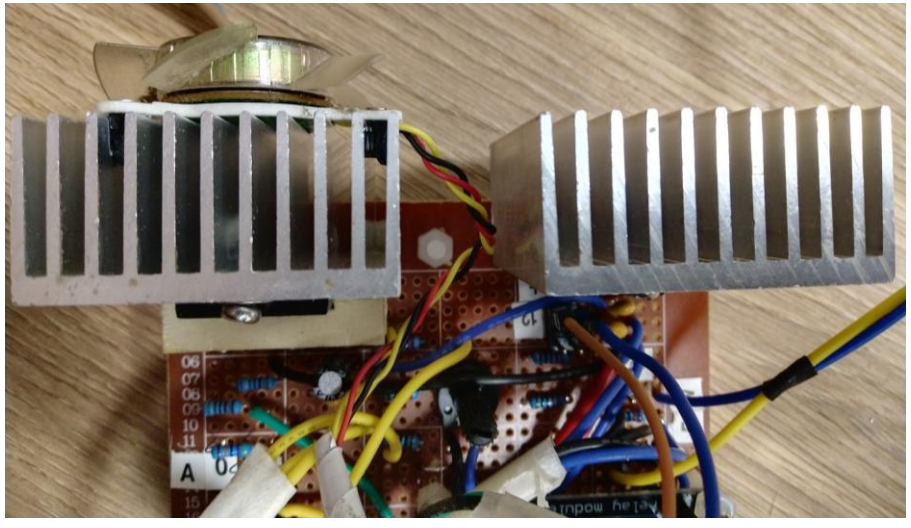


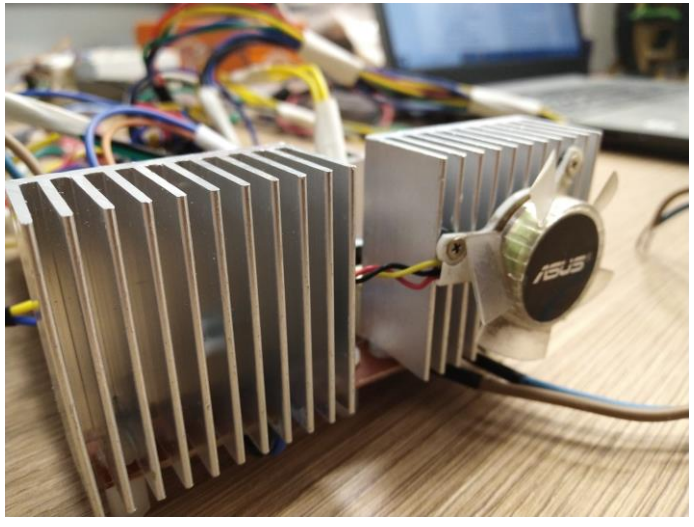

| | | | |
|---|---|---|---|
| Thermal Resistance, $\theta_{JC}$ | | 1.4 | °C/W |
| Thermal Resistance, $\theta_{JA}$ | No Heat Sink | 30 | °C/W |

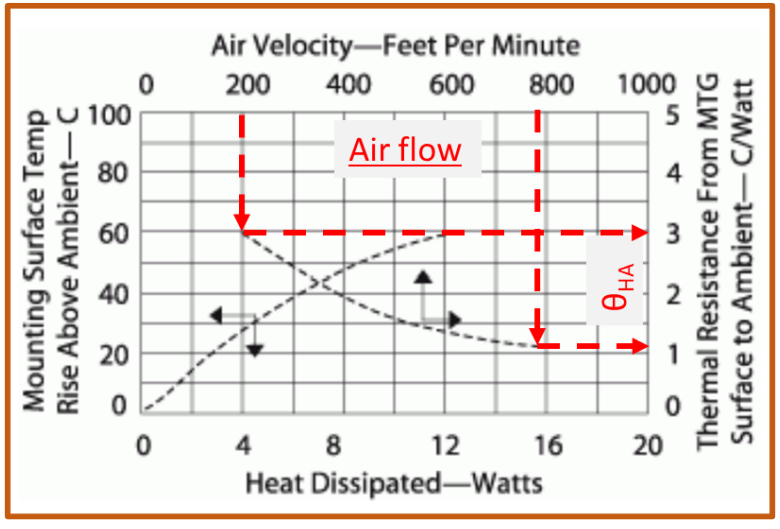
Since the theoretically minimum value of $\theta_{JA} = \theta_{JC} = 1.4$ ℃/W (ideal heat transfer from the case to air, $\theta_{CH} + \theta_{HA} = 0$), we can conclude that the maximum dissipated power cannot exceed $P_D = (T_J - TA)/\theta_{JA} \sim 70$ W, if $T_A = 25$ ℃. Therefore, assuming a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

# Heat management

Our well-blown heatsink for OPA549 is likely to add $\theta_{HA}$~1 ℃/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:

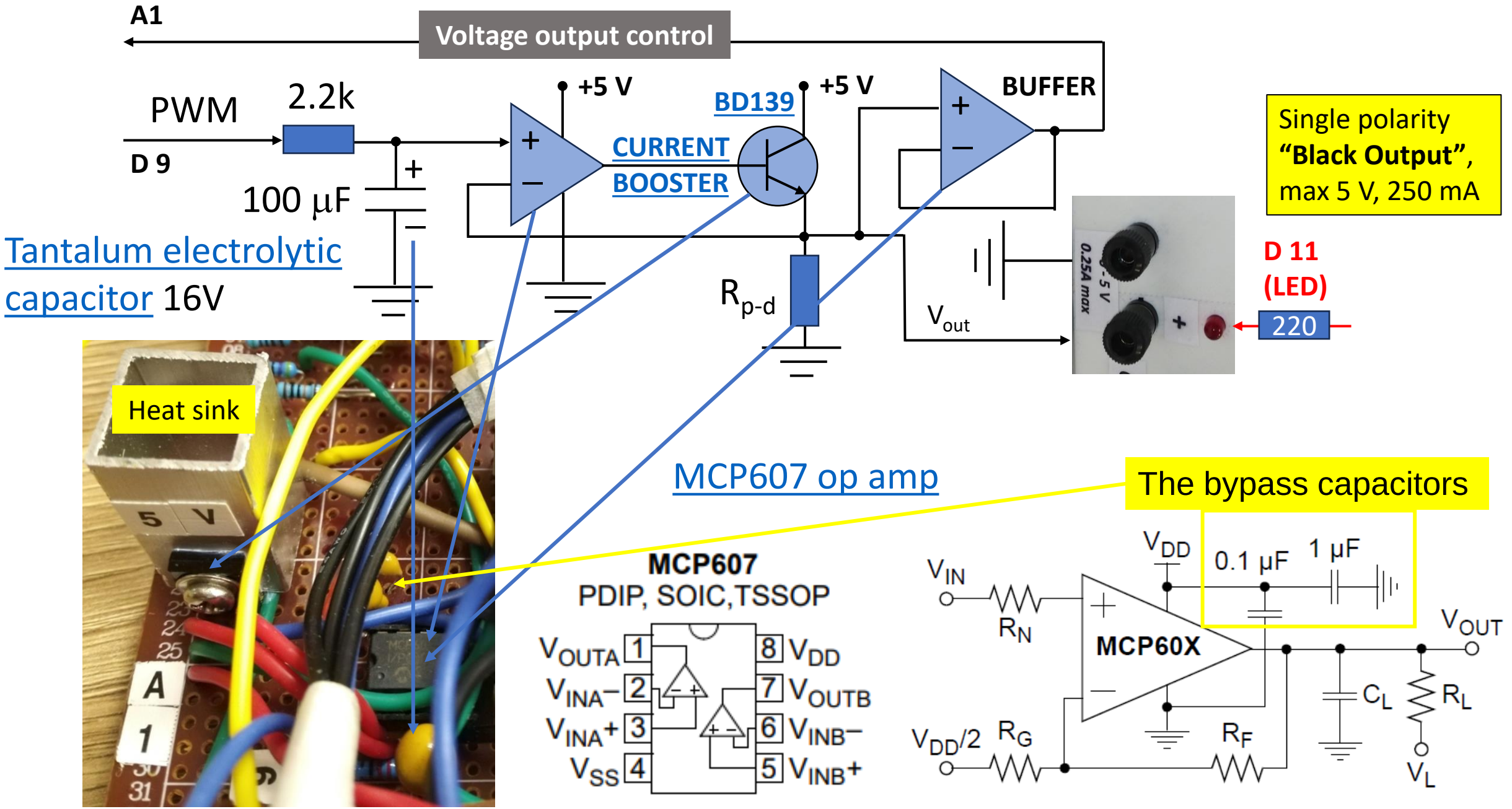
$$V(t) = \begin{cases} A, 0 \le t \le \tau \\ 0, \tau < t < T \end{cases}$$

where $A$ is a positive constant (5V for Arduino or 3.3V for Raspberry Pi). The ratio $\tau/T$ (%) 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 $V_{out}(t)$.

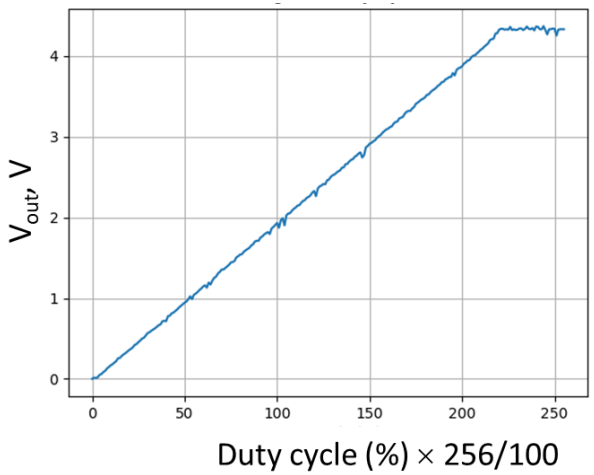
# Implementation of the PWM DC voltage source

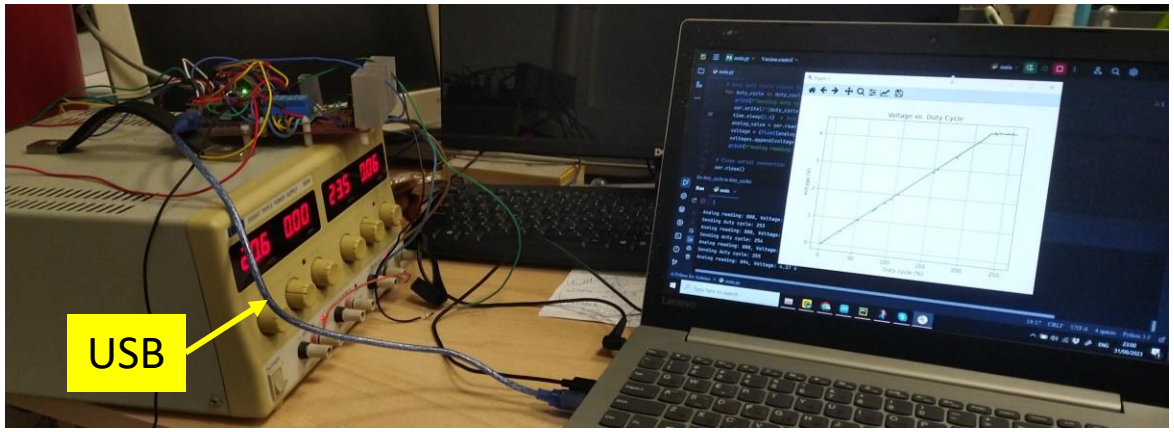


# Testing the PWM driver from Python

Python test program

```python
import serial
import time
import numpy as np
import matplotlib.pyplot as plt

# Open serial connection to Arduino
ser = serial.Serial('COM7', 9600)
time.sleep(1)  # Wait for Arduino to reset

# Sweep duty cycles from 0 to 255
duty_cycles = np.arange(0, 256)
voltages = []

# Send duty cycle values to Arduino, collect analog readings, and convert
for duty_cycle in duty_cycles:
    print(f"Sending duty cycle: {duty_cycle}")
    ser.write(f"{duty_cycle}\n".encode())  # Send duty cycle to Arduino
    time.sleep(1.5)  # Introduce a delay time (s)
    analog_value = ser.readline().decode().strip()  # Read analog value from Arduino
    voltage = (float(analog_value) / 1023.0) * 5.0  # Convert to voltage
    voltages.append(voltage)
    print(f"Analog reading: {analog_value}, Voltage: {voltage:.2f} V")

# Close serial connection
ser.close()

# Plotting the graph
plt.plot(duty_cycles, voltages)
plt.xlabel("Duty cycle (0-255)")
plt.ylabel("Voltage (V)")
plt.title("Voltage vs. Duty Cycle")
plt.grid(True)
plt.show()
```



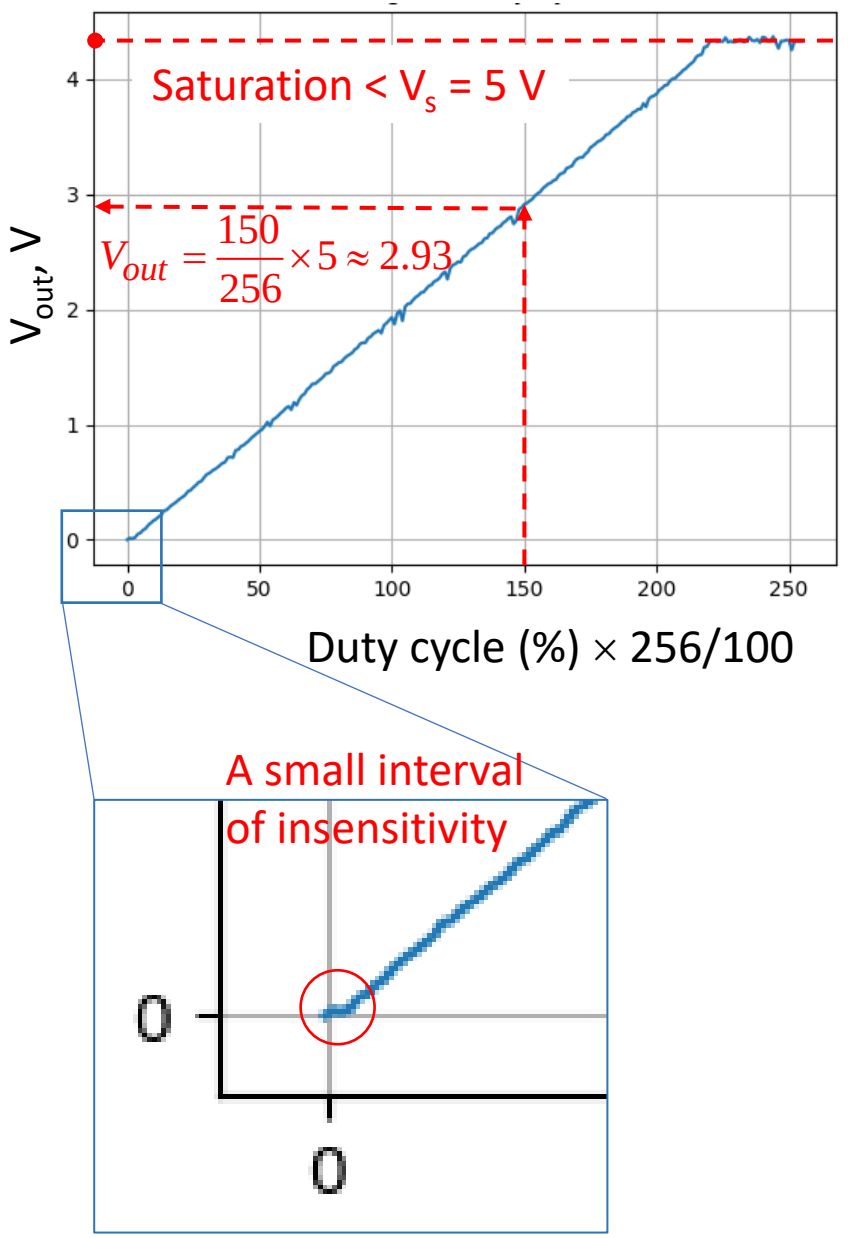




Arduino sketch

```arduino
#include <Arduino.h>

const int pwmPin1 = 9;  // PWM pin 1
int dutyCycle1 = 0;     // Duty cycle for PWM pin 1

void setup() {
  pinMode(pwmPin1, OUTPUT);
  analogWrite(pwmPin1, dutyCycle1);

  Serial.begin(9600);
}

void loop() {
  if (Serial.available() > 0) {
    dutyCycle1 = Serial.parseInt(); // Read duty cycle value from Python
    analogWrite(pwmPin1, dutyCycle1);
  }
  delay(100);

  // Read analog values from analog pin
  int analogValue1 = analogRead(A1);
  delay(1500);

  // Send analog value to Python
  Serial.println(analogValue1);
}
```

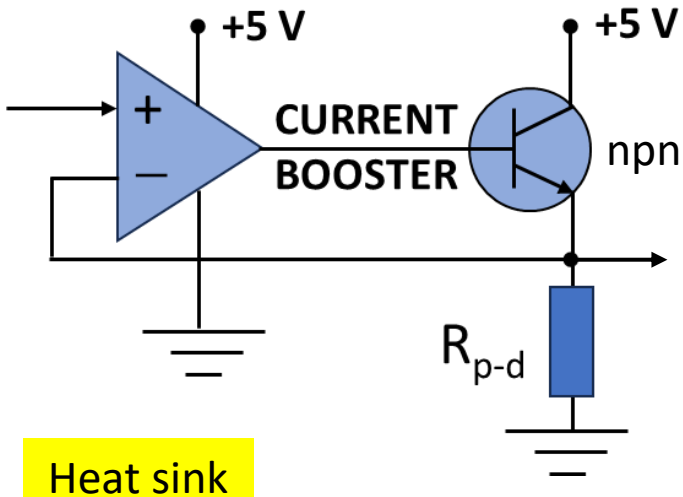
# Operation of the PWM DC voltage source

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 kΩ and 100 μF). When the duty cycle changes, it is necessary to wait at least $5\tau = 1.1$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.

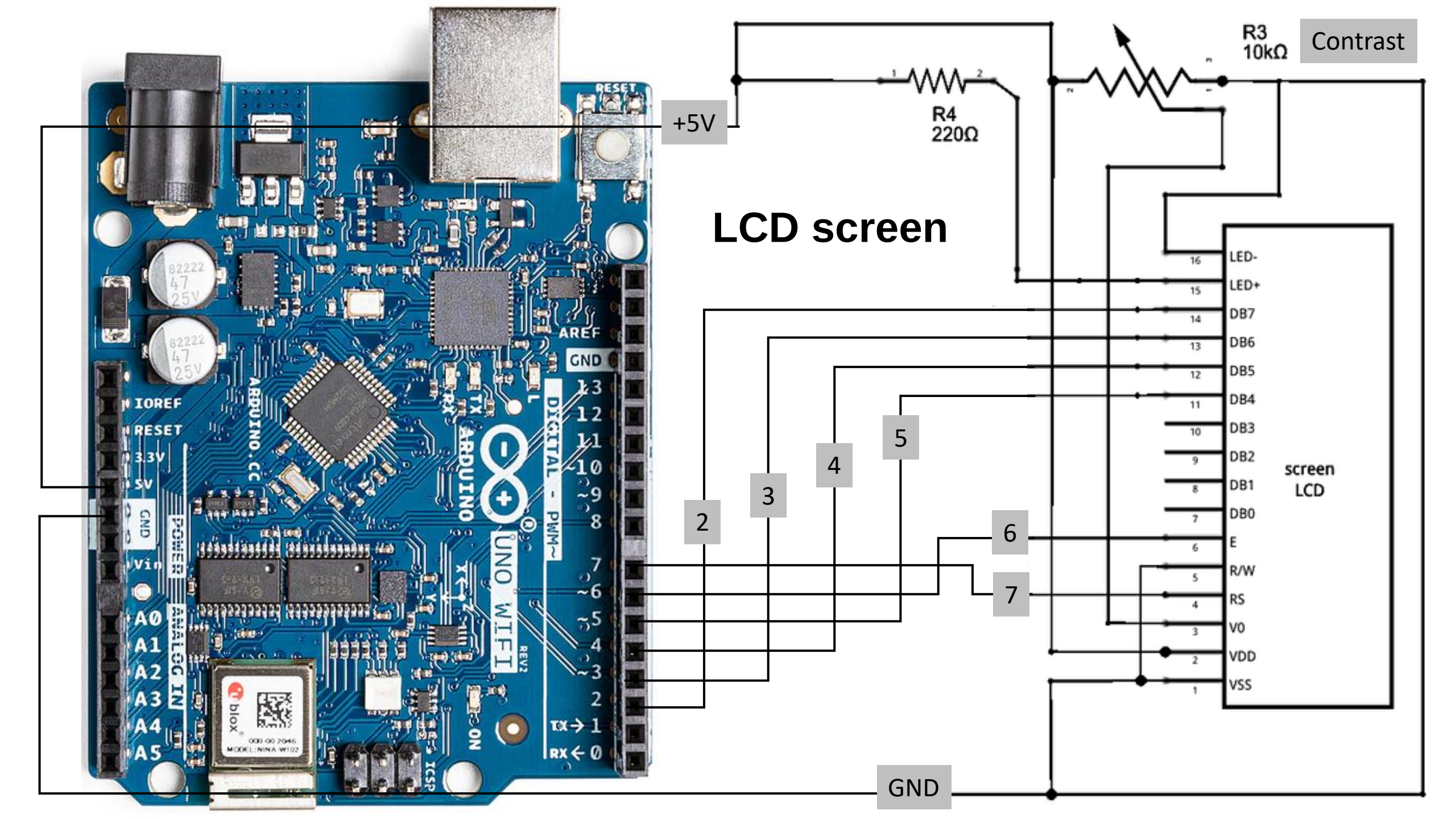


# Current booster and heat management for the PWM DC voltage source

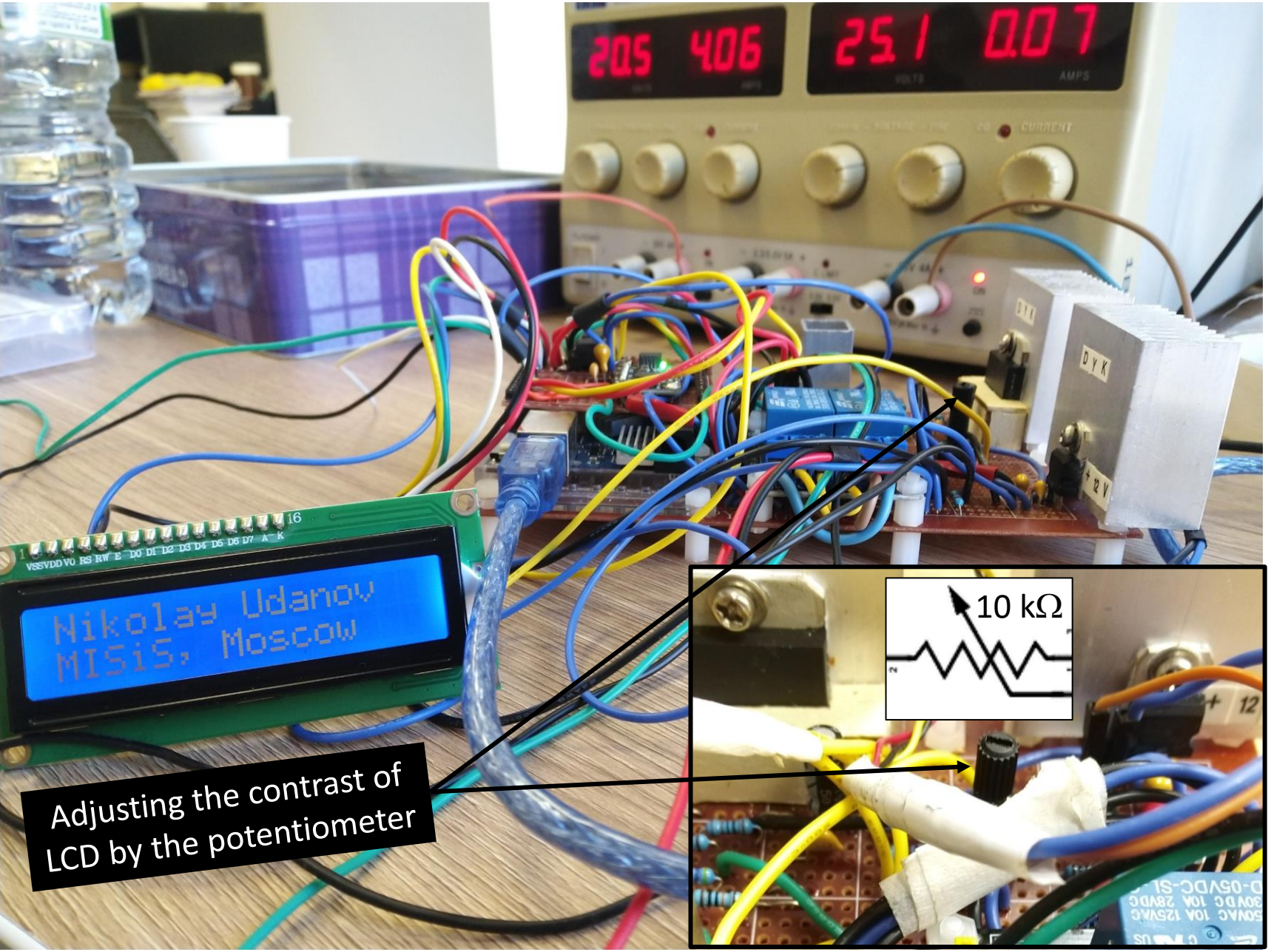
Since a standard operational amplifier cannot provide an output current greater than $20 - 25$ mA, we used a current booster circuit based on a npn transistor incorporated into the feedback loop of the operational amplifier. The heat management of this transistor stage is handled using the same approach 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 $\theta_{JA} = 100$ °C/W. Therefore, with $T_A = 25$ °C and $T_J = 150$ °C, the maximum power dissipation is calculated as $P_D = (T_J - T_A) / \theta_{JA} = 1.25$ W (no heatsink). The thermal resistance junction-to-case $\theta_{JC} = 10$ °C/W. The use of a heatsink with fan cooling can achieve $\theta_{HA}$ (heatsink-to-air) close to 1 resulting in $\theta_{JA} \approx 11$ °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 $R_{p\text{-}d}$ can be chosen as $100$ kΩ.




Heat sink

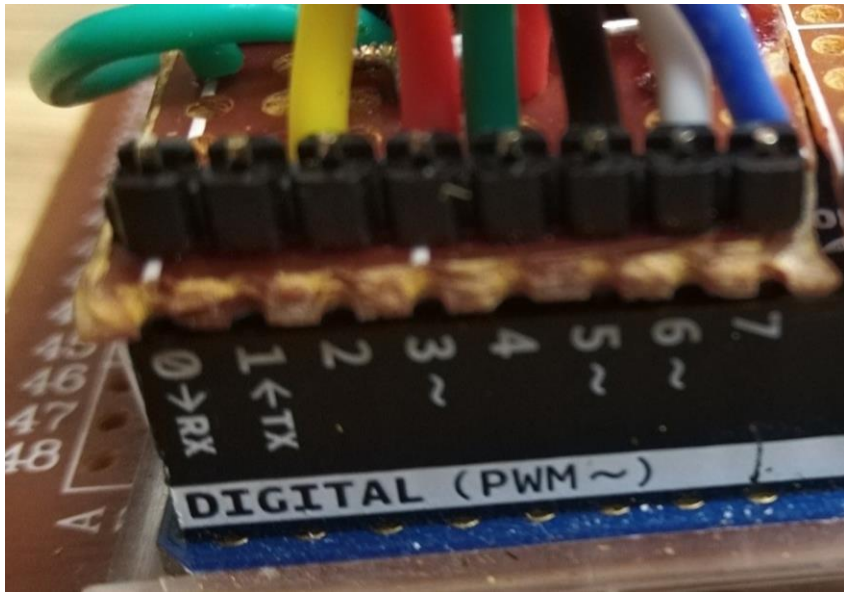

+5V
R4
220Ω
R3
10kΩ
Contrast
LCD screen
2
3
4
5
6
7
GND
screen
LCD
LED-
LED+
DB7
DB6
DB5
DB4
DB3
DB2
DB1
DB0
E
R/W
RS
V0
VDD
VSS
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
IOREF
RESET
3.3V
5V
GND
Vin
A0
A1
A2
A3
A4
A5
ANALOG IN
POWER
DIGITAL - PWM~
AREF
GND
ARDUINO.CC
ARDUINO
UNO WIFI
REV2
ON
ICSP
RESET

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

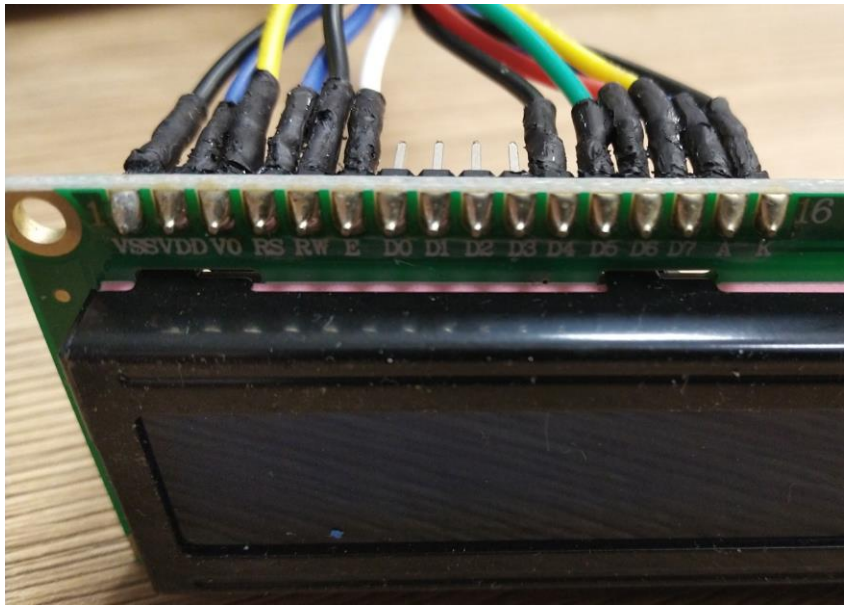








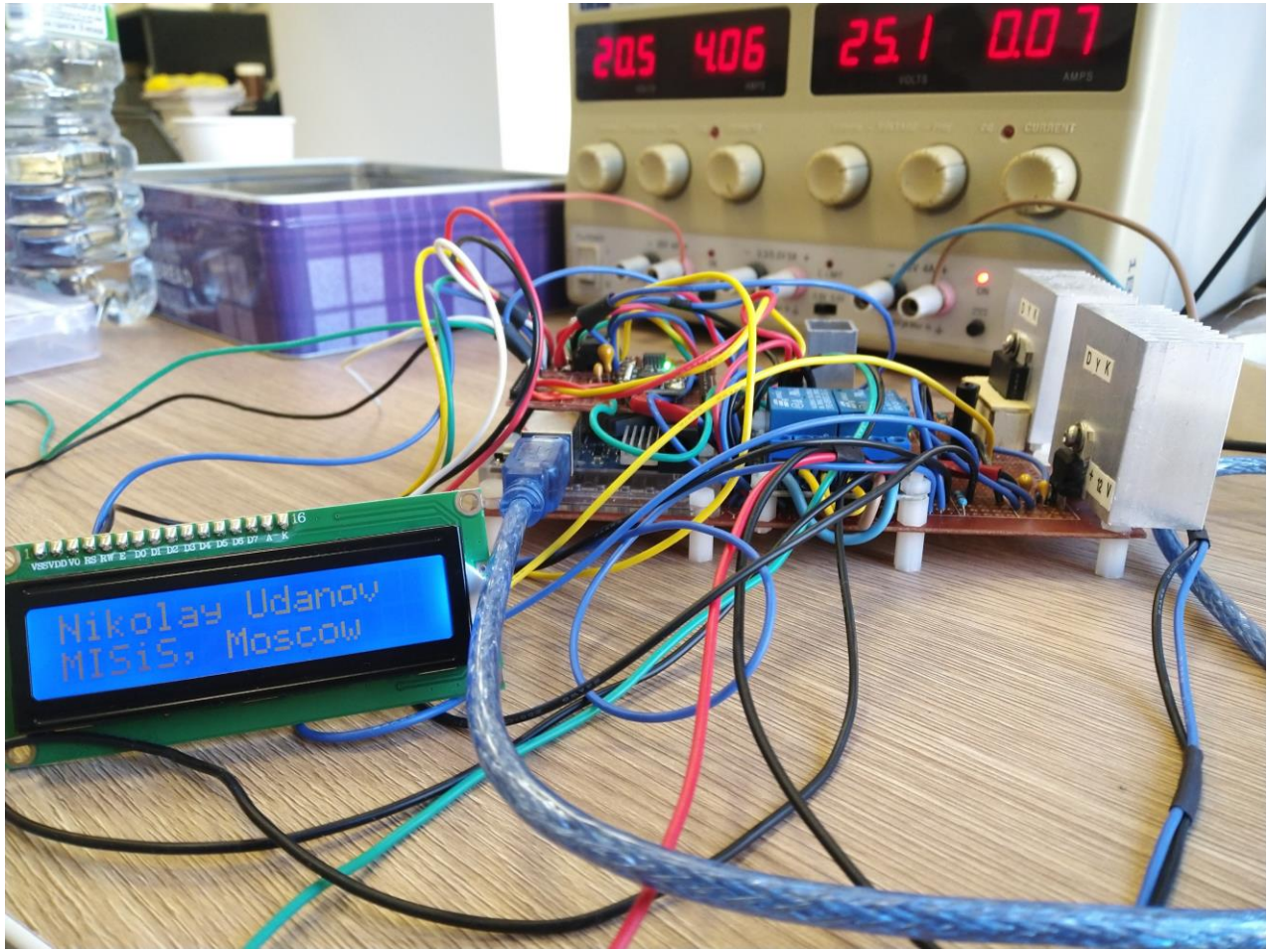

**wire colour code**



# Testing the LCD screen

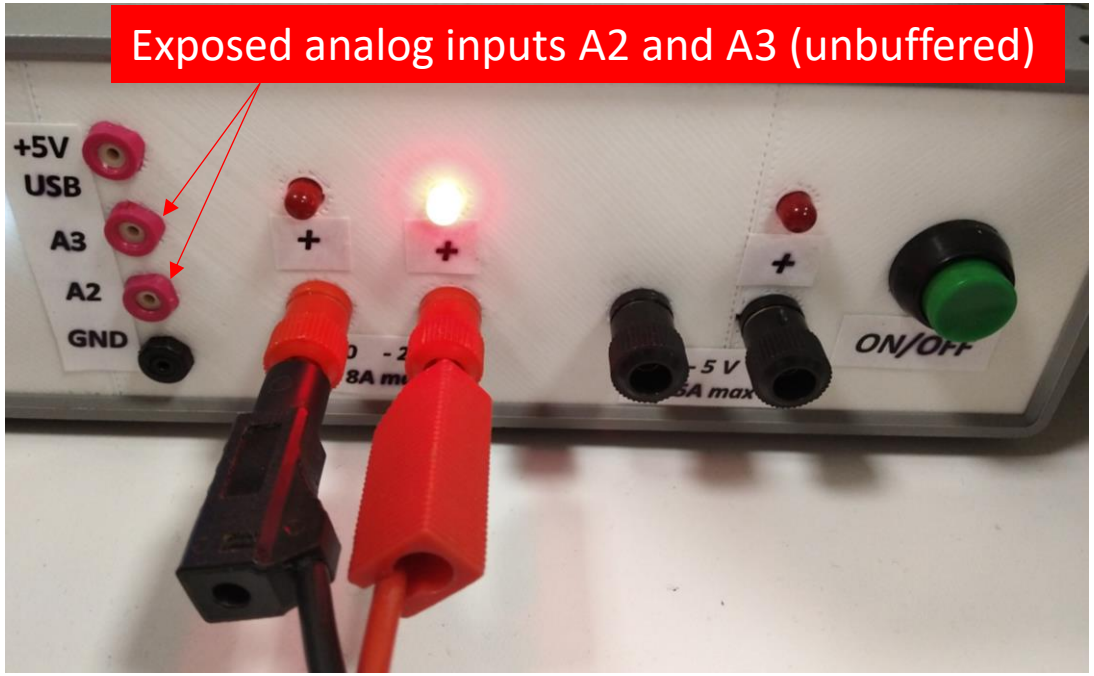
```
1. #include <LiquidCrystal.h>
2.
3. LiquidCrystal lcd(7, 6, 5, 4, 3, 2);
4.
5. void setup() {
6.   lcd.begin(16, 2);
7.   lcd.clear();
8.   lcd.setCursor(0, 0);
9.   lcd.print("Nikolay Udanov");
10.  lcd.setCursor(0, 1);
11.  lcd.print("MISiS, Moscow"); // Print the second line
12.
13. }
14.
15.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

Arduino sketch



# 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