Introduction

When you have to debug electronic systems or find failure on an electronic board or electrical system, you frequently need to locate short circuits.

Common multimeters are generally limited down to 1 ohm in their measurement and this is not sufficient for efficient characterization and localization of a short circuit.

Below the 1 ohm domain, a milliohmmemeter is the required instrument but Industrial miiliohmmeters are expensive so I decided to build my own device.

General design

I had viewed a good publication for a DIY milliohmmeter from the "Electro-bidouilleur" YouTube channel. The Elektor magazine sells also a add-on device for multimeters.

However this projects suffers several drawbacks:

- need a separate multimeter for the measurement
- provides no protection against excessive voltage on the device under test
- do not use 4 wires measurement for better accuracy

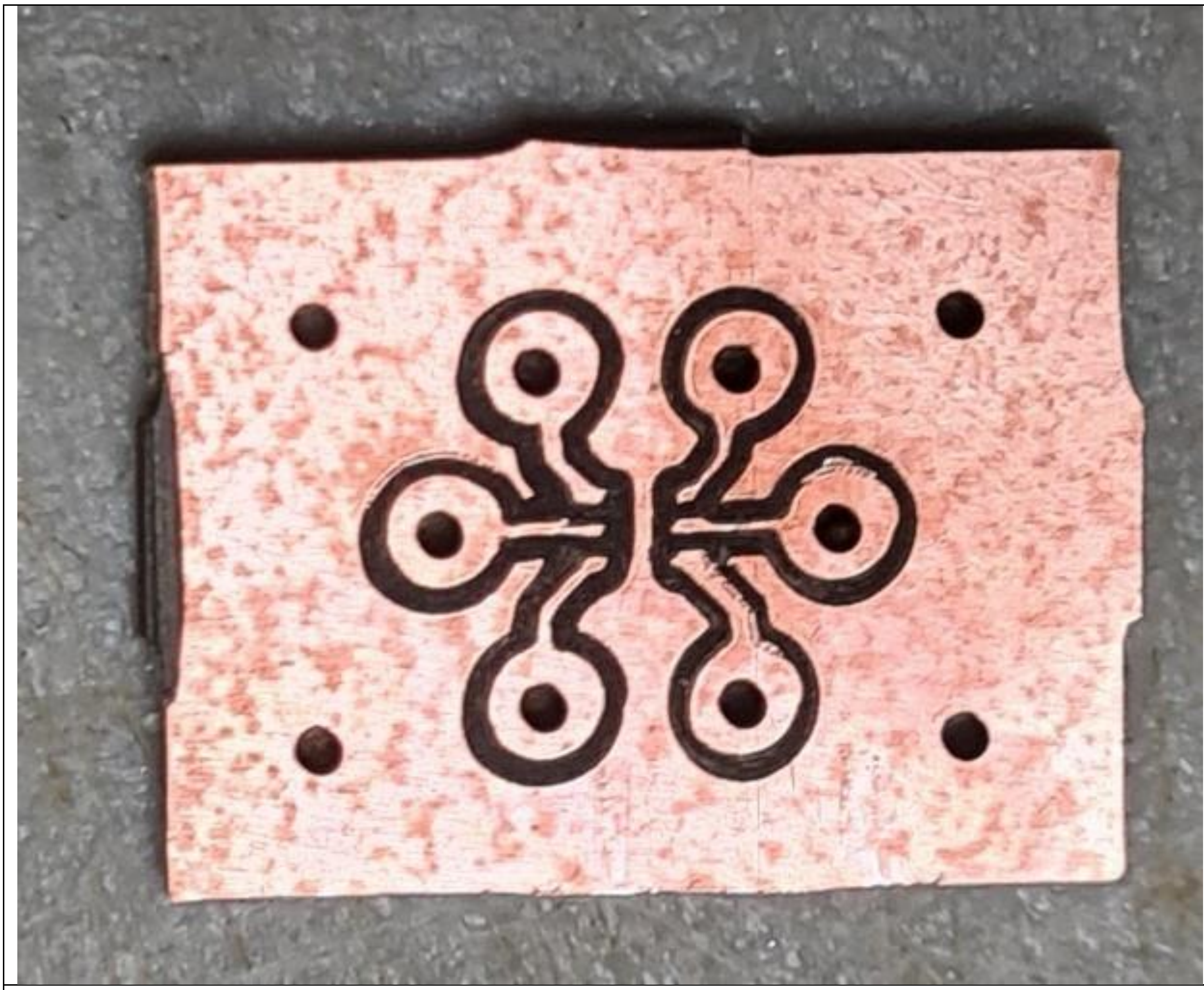
The main characteristics of my design are:

- use of a INA199 from Texas Instrument for the measurement
  - this is a Voltage-Output, Current-Shunt Monitor which need no additional component
  - provides very low offset voltage and very low drift
  - high accuracy gain
- current source of 10.6 mA build with a ½ TLC272 which is a precision dual operational amplifier
- voltage limiter with a BC548 transistor
- use of and Arduino Uno board for:
  - output voltage measurement,
  - resistance computation from voltage measurement
  - initial zero compensation
  - full range calibration
- use of a 2 lines x 16 characters LCD display shield for user interface
- use of a a 1200 mAh LiPo battery combined with a step up converter/ battery charger for portable operation
- use of a "Kelvin" 4 wires cable for device connection

Design details

One of the difficulties is that the INA199 is available only in a tiny SMD SC70 package and as I wanted to test the design with a breadboard, I had to build a SC70 adaptor to connect the INA199 to my breadboard.

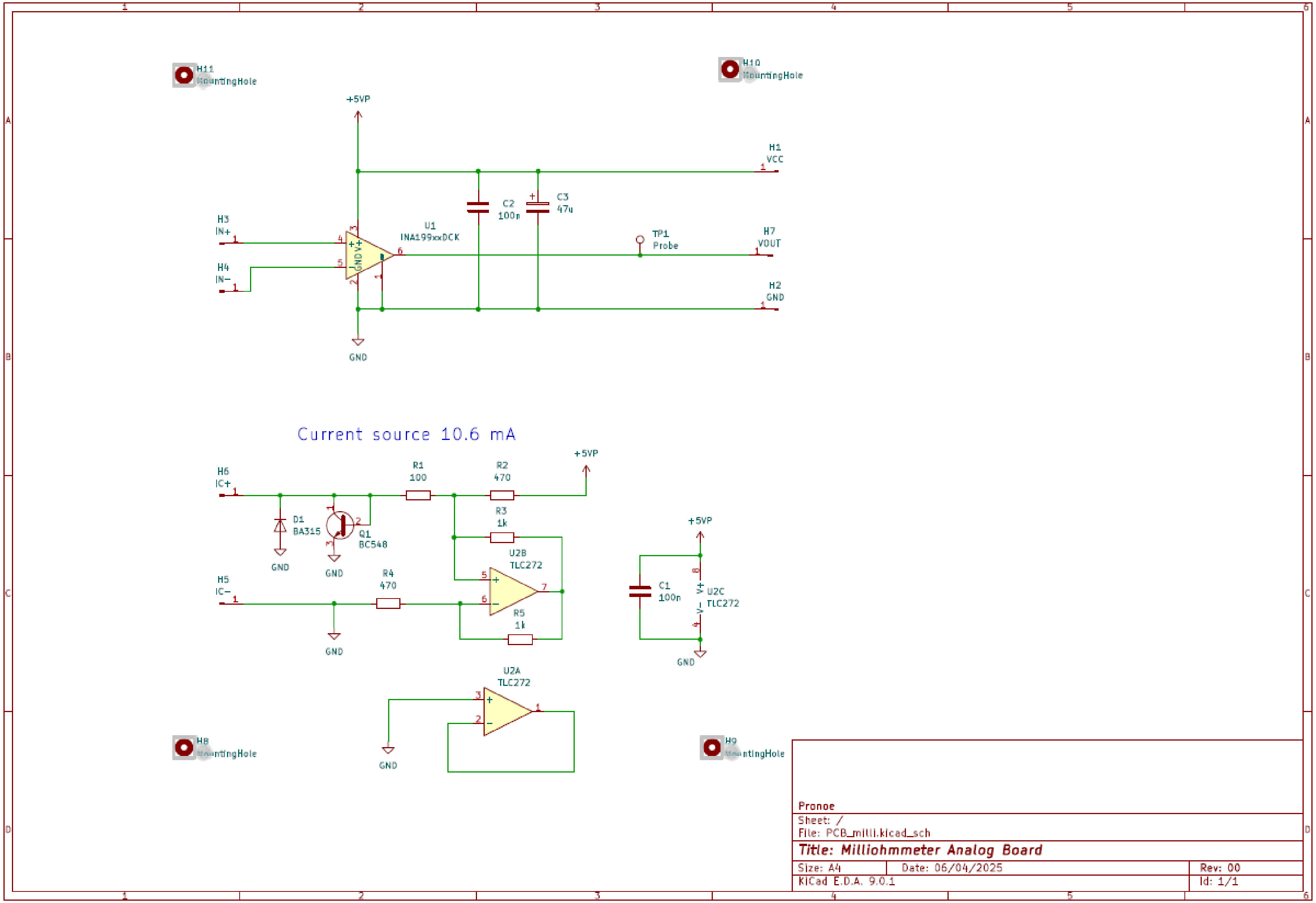
I have made this adaptor by milling a PCB with a CNC 3018. Meanwhile I was also testing the ability to implement a SC70 adaptor with such a limited machine as the CNC 3018.



SC70 package adaptor for breadboard testing

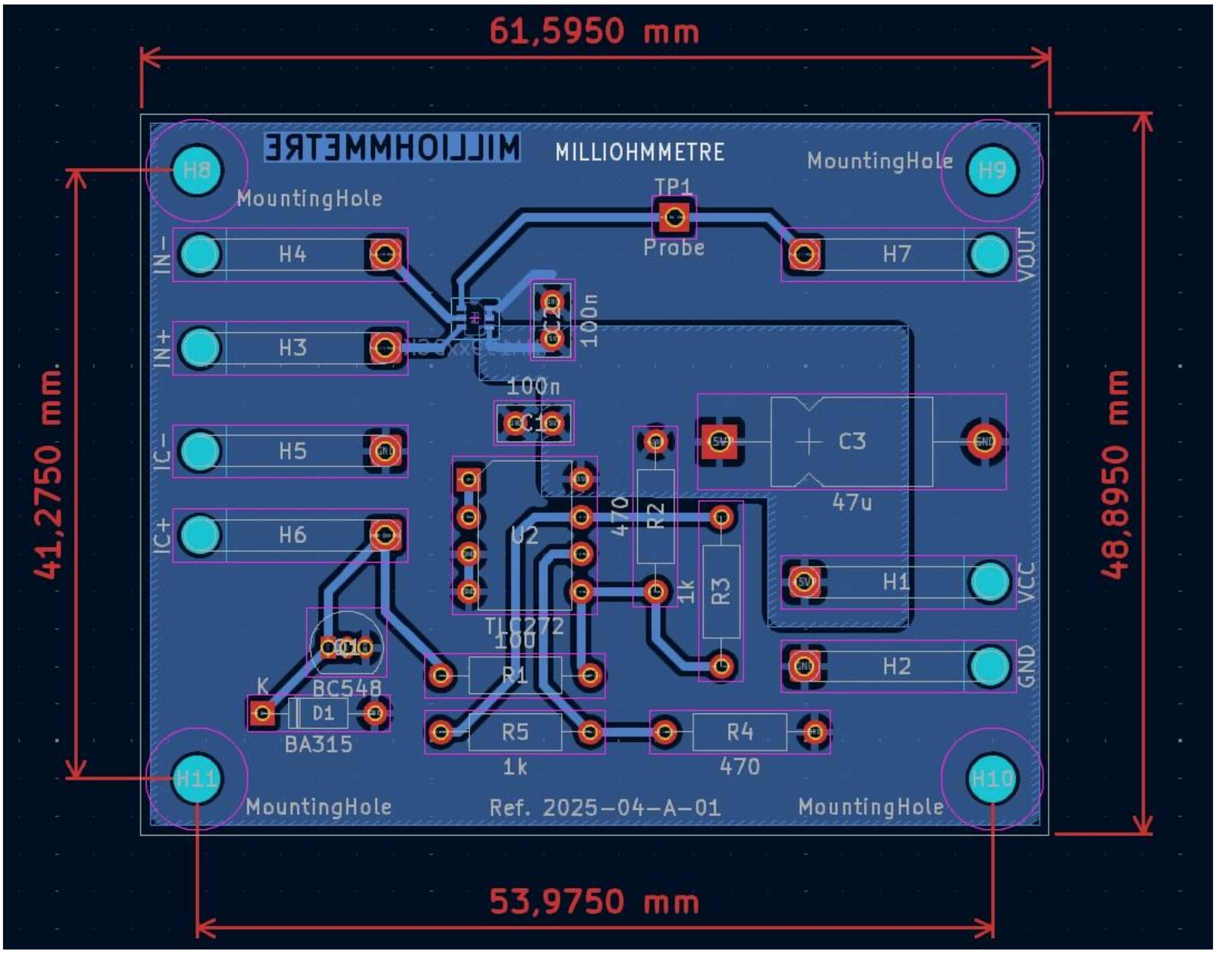
This was a success and my breadboard confirmed the good behavior of the design.

This is the final schematic that I have implemented.

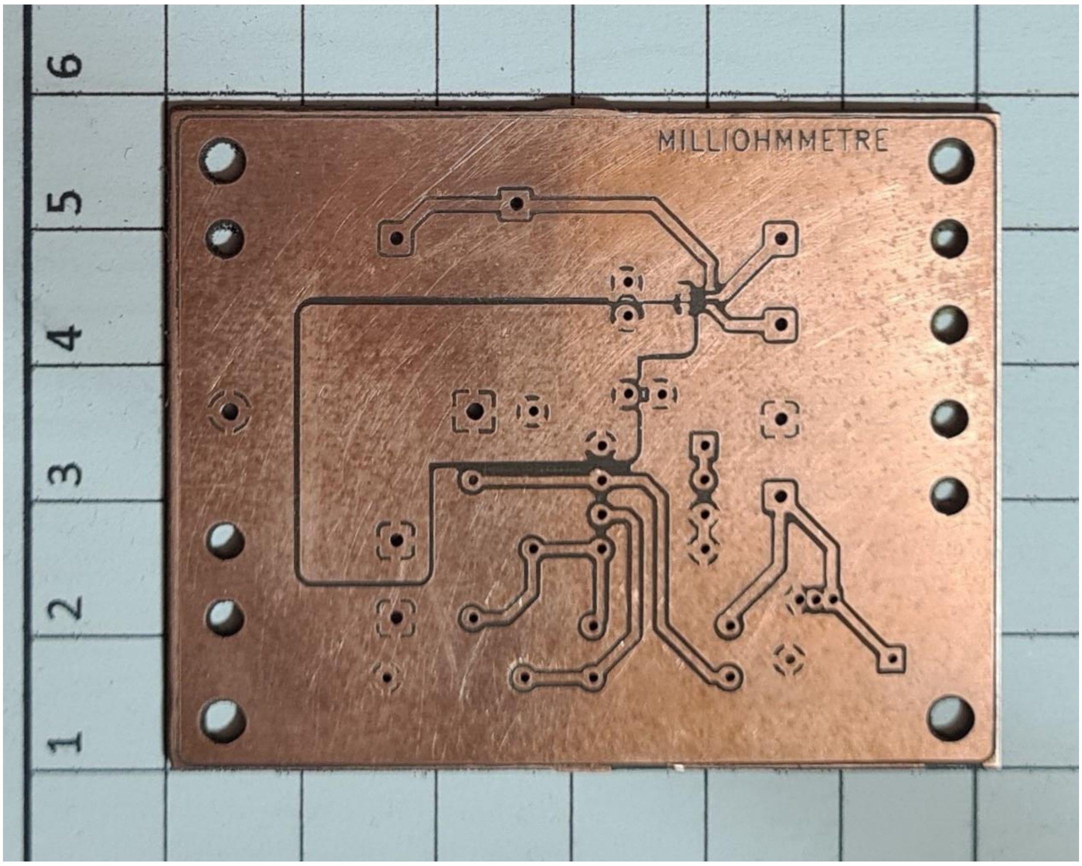


I have designed a single side PCB for this analog circuit, with Kicad and manufactured it by milling with my CNC 3018 router.

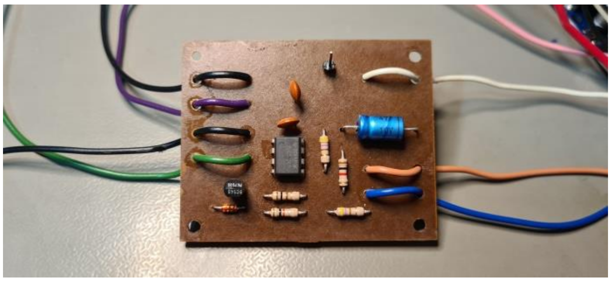
Kicad view

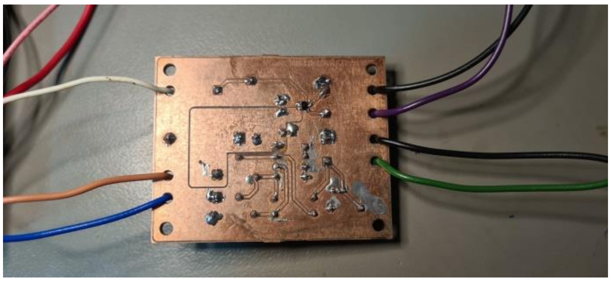


Result after milling and cleaning



Finished circuit

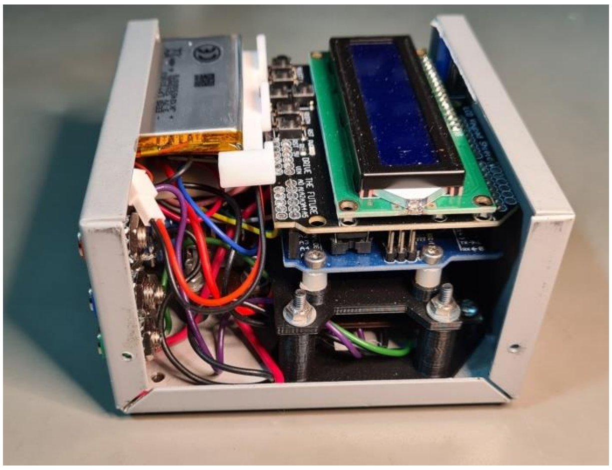


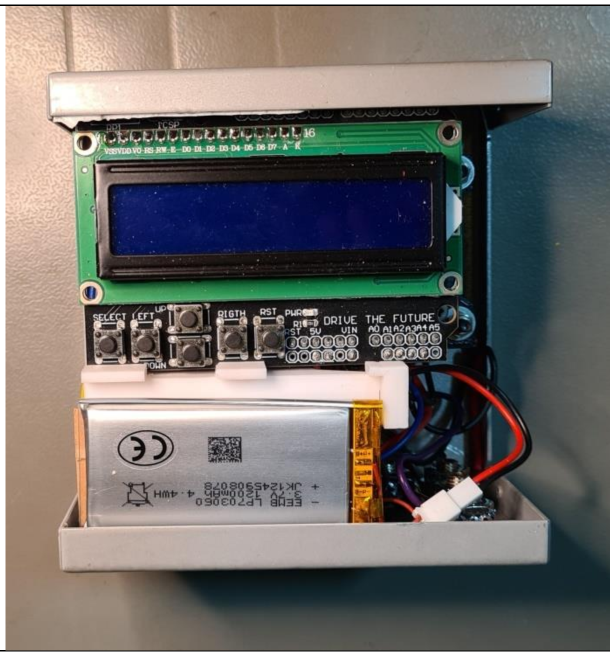


A small metallic box with 4 switches was reused from a previous project to integrate al the sub systems:

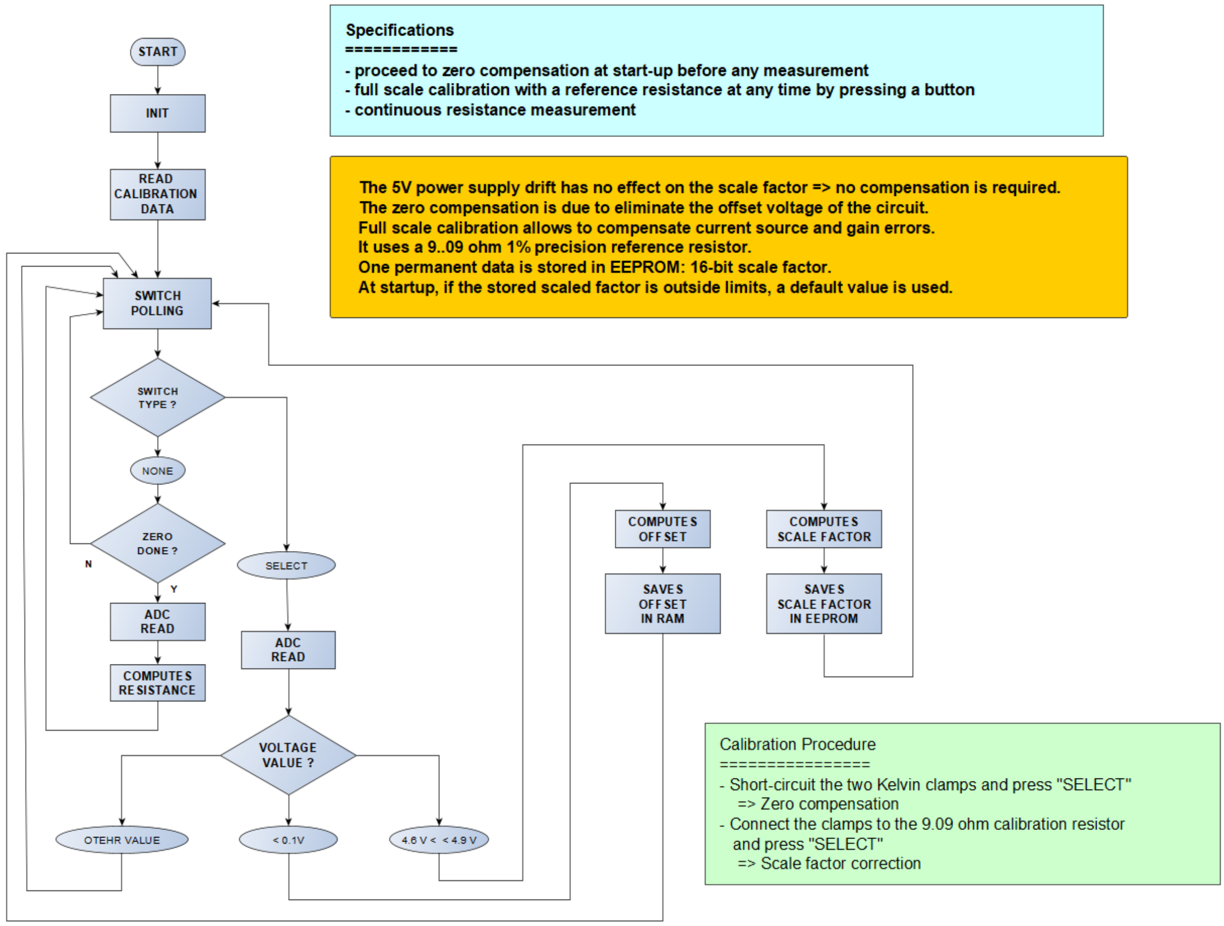
- analog board
- Arduino Uno board
- LCD shield
- Switches interface board
- LiPo battery
- Step-up CV and charger module
- USB-C female plug for the battery charging

Dedicated stands made by 3D printing in PLA have been used for the boards and the battery.





Arduino software



Equation Definition

Rmes = (Vmes - V0) x SF

With:

- Rmes = resistance measured value in mohm
- SF = Scale Factor in mohm/V
- Default SF = 1 / (50 x 10.55 e-3) = 1896 mohm/volt
- Vmes = ADC_read /1024 x Vcc

Vcc = 5.00 V (the exact value is irrelevant; it is compensated by SF)

Therefore, Rmes = (ADC_read - ADC_0) x SF1

SF1 = SF / 1024 x 5, nominal value 9.258 mohm / bit = 9258 µohm / bit

In practice, SF1 is stored in EEPROM in a 16 bits word (unsigned INT type).

Then, ADC_read x SF1 is calculated as a long INT and divided by 1000.

<u>Zero Compensation</u>

- Upon "SELECT" detection, Rmes is measured, and if Rmes < 100 mohm ((equivalent to ADC threshold of 11), this validates the zero compensation mode
- Then the current value of ADC_read is saved in ADC_0 variable
- While the initial zero compensation has not been done, a warning message reminds the user that zeroing must be performed

Scale factor calibration

- upon "SELECT" detection, Rmes is measured and if 8800 < Rmes < 9400 mohm (equivalent to ADC thresholds of 950 and 1015), this validates the calibration mode
- in this case, we compute the new value of SF1 = 9090000 / (ADC_read - ADC_0) and save it in EEPROM

Parameters

- EEPROM address for SF1 storage: #0
- Measurement frequency: 1 s (might be changed in future versions)
- Input switch polling rate: 100 ms (but the SELECT button can only be used once every second)
- At initialization:
  - ADC_0 set to -1 at init, which determines that zero is not set
  - SF1 read from EEPROM. If SF1 is outside the range [9700, 8800], SF1 is set to the default value of 9258

Resistance computation

- Calculation of SF1 and Rmes as 32-bit integers (+/- 2.1 109) and conversion to 16-bit integers after rounding to the nearest possible value.

Latest improvements

- Measurement rate selectable from 0.1 s and 1s
- Measurement filter selectable (IIR 1st order filter with coefficient 0.1)
- Implementation of 2 consecutive measurements and use of the 2nd only to optimize ADC precision

Future improvements

- Automatic standby to reduce battery drain
- Calculation and display of statistics for the last 100 measurements: min, max, average
- Use of the ATmega controller's "noise canceler" mode to reduce noise induced errors

Arduino code

```
//========================================================================/
// Software for milliohmmeter board with Arduino UNO                      /
// the board uses an INA199 IC for voltage amplification with low offset /
// and a constant current source of 10.55 mA powered by Arduino Vcc (5V) /
// The Arduino UNO board is fitted with a LCD 2x16 characters LCD dsplay /
// Software requirements:                                                 /
// - continuous resistance measurement and result display at 1s rate     /
// - zero compensation when "SELECT" key depressed                        /
// - scale factor calibration when "SELECT" key depressed and reference  /
//   value of 9.09 ohm +/- 1% is connected to the input                  /
// - calibration data saved in EEPROM                            /
//========================================================================/
// Author: Patrick SOUTY (Pronoe)
// Date of creation: 02/04/2025
// under GNU General Public License v3.0
// Versions:
// 1.0 - 02/04/2025 - initial version
// 2.0 - 08/05/2025 - use of external buttons following hardware integration
// 2.1 - 14/05/2025 - cleaning of code before publication and minor corrections
// 3.0 - 22/05/2025 - IIR filter mode added + double ADC measurement + measurement rate
selection
```

```
//=====================================================================================
========/

String SW_name = "Milliohm";
String SW_version = "V3.0";
#include <LiquidCrystal.h>  // using LiquidCrystal library
#include <EEPROM.h>         // using EEPROM library

// define EEPROM addresses used for SF1 and ADC_0 storage
#define SF1_ADD 0
#define ADC_0_ADD 4

#define SF1_default 9258  // default scale factor value in µohm/bit
// define analog input for V(R) measurement
#define Vin A1            // use A1 for INA199 output voltage measurement
#define polling_rate 200  // polling rate of buttons and switches in ms
// define variables used for resistance measurement
int meas_rate = 1000;                                 // delay between measurements
in ms
int ADC_read;                                         // used for ADC measurement
storage
int SF1;                                              // scale factor in µohm/bit
int ADC_0;                                            // ADC value for zero ohm
input (i.e. offset value)
int Rref = 9090;                                      // calibration resistance in
milliohm
int Rmeas;                                            // measured resistance value
in mohm
long Rmeas_long;                                      // for 32 bits computation of
Rmeas
long Rmeas_filtered;                                  // filtered value of Rmeas on
32 bits
bool filter_on;                                       // filter mode ON or OFF
int ADC_min = int((long(Rref) * 965) / long(SF1_default));   // ADC lowest value for scale
factor calibration (nominal value -3.5%)
int ADC_max = int((long(Rref) * 1035) / long(SF1_default));  // ADC highest value for
scale factor calibration (nominal value +3.5%)
int ADC_0_max = 11;                                   // ADC highest value for zero
compensation
int coeff_filt = 100;                                 // actual value of order 1
IIR filter multiplied by 1000 (i.e. 0.1 for the actual value)
int i;                                                // index for loops
unsigned long time;                                   // for time management of adc
sampling
unsigned long polling_time;                           // for time management of
buttns and switch polling
bool keyProcessed;                                    // used to avoid multiple
processing of a single key pressed
bool test = false;
int ADC_test = 2;  // used for test
String MyString;
String Blank = "                ";  // used to clear a full LCD line
String current_rate = "slow/";
```

```
String current_filter = "filter off";
// select the pins used on the LCD panel
LiquidCrystal lcd(8, 9, 4, 5, 6, 7);
// define some values used by the panel and buttons
#define pushButton1 A2   // gray wire - note: A2, A3 and A4 used as digital inputs
#define pushButton2 A3   // yellow wire
#define switchButton A4  // green wire
int calSelect = 0;
int validateSelect = 0;  // push button state - reserved for future use
int modeState = 0;       // stat of the mode button - reserved for future use

// reads HW push buttons and switch subroutine
void read_HW_buttons() {
  calSelect = !digitalRead(pushButton1);
  validateSelect = !digitalRead(pushButton2);
  modeState = !digitalRead(switchButton);
  if (test) {
    Serial.print("buttons state : ");
    Serial.print(calSelect);
    Serial.print(" ");
    Serial.print(validateSelect);
    Serial.print(" ");
    Serial.println(modeState);
  }
}

void setup() {
  // read calibration values in EEPROM
  Serial.begin(115200);
  ADC_0 = -1;                // negative value states that zero compensation has not been
done
  EEPROM.get(SF1_ADD, SF1);  // read SF1 value in EEPROM
  if ((SF1 > 9700) || (SF1 < 8800)) {
    SF1 = SF1_default;
  }
  // input buttons configuration
  pinMode(pushButton1, INPUT_PULLUP);
  pinMode(pushButton2, INPUT_PULLUP);
  pinMode(switchButton, INPUT_PULLUP);
  // LCD display init
  lcd.begin(16, 2);  // start the LCD library
  lcd.display();
  lcd.setCursor(0, 0);
  MyString = SW_name + "   " + SW_version;
  Serial.println(MyString);
  Serial.println("Program started ...");
  lcd.print(MyString);  // print SW information
  lcd.setCursor(0, 1);
  lcd.print("set zero & "
            "SELECT"
            "");

  time = millis() + meas_rate;             // save next time to make measurement
```

```
  polling_time = millis() + polling_rate;  // save next time to poll buttons
  keyProcessed = false;
  filter_on = false;
}

void loop() {
  if (millis() >= time) {
    // it's time to carry on measurement
    // ADC measurement
    if (test) {
      ADC_read = ADC_test;
    } else {
      ADC_read = analogRead(Vin);  // samples input voltage - dummy read
      ADC_read = analogRead(Vin);  // samples input voltage - second read with stabilized
input
    }
    if (test) {
      Serial.print("SF1: ");
      Serial.println(SF1);
    }
    time = time + meas_rate;  // sets next sample time
    if (ADC_0 > -1) {
      // zero compensation has been done, then resistance value may be computed
      if (ADC_read == 1023) {
        // overflow (open circuit or input resistance > 9.4 ohm)
        lcd.setCursor(0, 1);
        lcd.print("Overflow        ");
      } else {
        Rmeas_long = long(SF1) * long(ADC_read - ADC_0);  // computes with 32 bits in µohm
        Rmeas_filtered = (coeff_filt * (Rmeas_long - Rmeas_filtered) / 1000L +
Rmeas_filtered);
        if (filter_on) {
          Rmeas = int((Rmeas_filtered + 500L) / 1000L);  // round to the nearest value in
mohm
        } else {
          Rmeas = int((Rmeas_long + 500L) / 1000L);  // round to the nearest value in mohm
        }
        // display value on second line
        lcd.setCursor(0, 1);
        MyString = "R " + String(Rmeas) + " mohm       ";
        lcd.print(MyString);  // dispaly measured value
      }
    }
    keyProcessed = false;  // enable key processing
  }                        // end of measurement block
  // buttons management
  if (!keyProcessed && (millis() >= polling_time)) {
    // lcd_key = read_LCD_buttons();  // read the buttons
    polling_time = millis() + polling_rate;  // sets next polling time
    read_HW_buttons();
    if (calSelect)  // calibration button depressed
    {
      if (ADC_read < ADC_0_max) {
```

```
        // zero compensation validated
        if (test) {
          Serial.print("ADC_read: ");
          Serial.println(ADC_read);
        }
        ADC_0 = ADC_read;  // save offset value
        lcd.setCursor(0, 1);
        lcd.print(Blank);  // remove warning message
        if (test) { ADC_test = 960; }
      } else if ((ADC_read > ADC_min) && (ADC_read < ADC_max) && (ADC_0 > -1)) {
        // scale factor calibration validated
        SF1 = int((long(Rref) * 10000L / long(ADC_read - ADC_0) + 5L) / 10L);  // computes
with 32 bits and round to the nearest value
        lcd.clear();
        MyString = "Cal " + String(SF1);
        lcd.print(MyString);       // display scale factor on 1st line
        EEPROM.put(SF1_ADD, SF1);  // save SF1 value in EEPROM
      } else {
        // SELECT button depressed with wrong input conditions
        // blink the display for warning
        for (i = 0; i <= 5; i++) {
          lcd.noDisplay();
          delay(300);
          lcd.display();
          delay(300);
        }
      }
      keyProcessed = true;  // prevents further processing of the key until next sample
time
    }
    // management of mode state switch
    if (modeState) {
      if (!filter_on) {
        // a transition from OFF to ON of the state switch has been detected
        filter_on = true;              // activate filtering
        Rmeas_filtered = Rmeas_long;   // initialize filtered value with last unfiltered
value upon activation of filtered mode
        current_filter = "filter on ";
      };
    } else {
      filter_on = false;  // inhibits filter
      current_filter = "filter off";
    }
    if (validateSelect) {
      // manage measurement rate
      if (meas_rate == 1000) {
        meas_rate = 100;  // switch to 100 ms measurement rate
        current_rate = "fast/";
      } else {
        meas_rate = 1000;  // switch to 1 s measurement rate
        current_rate = "slow/";
      }
      Serial.println(meas_rate);
```

```
    }
    // display current operation mode
    if (ADC_0 > -1) {
      MyString = current_rate + current_filter;
      lcd.setCursor(0, 0);
      lcd.print(MyString);
    }
    keyProcessed = true;  // prevents further processing of the key until next polling
time
  }                       // end of !keyProcessed block
}
```

Measurement characteristics and test results

Voltage measurement resolution by ADC	4.88 mV
Measuring current	10.6 mA nominal
Resistance measurement resolution - without filter - with filter	 9,3 mohm 1 mohm
Measurement accuracy	≈ 2% after zero and scale factor calibration
Measurement range	9,5 ohm max
Voltage protection for the DUT	≈ 700 mV
Battery autonomy	≈ 13 h

Measurement of a 0.26 ohm precision shunt:

- without filter : commutes between 256 mohm and 263 mohm value
- with filter : 260 mohm +/- 1mohm

Zero ohm compensation

	
Measurement with filter activated	**Measurement with no filter**

Conclusion

This simple and low cost milliohmmeter provides very good performances and efficiency in a small and autonomous device.