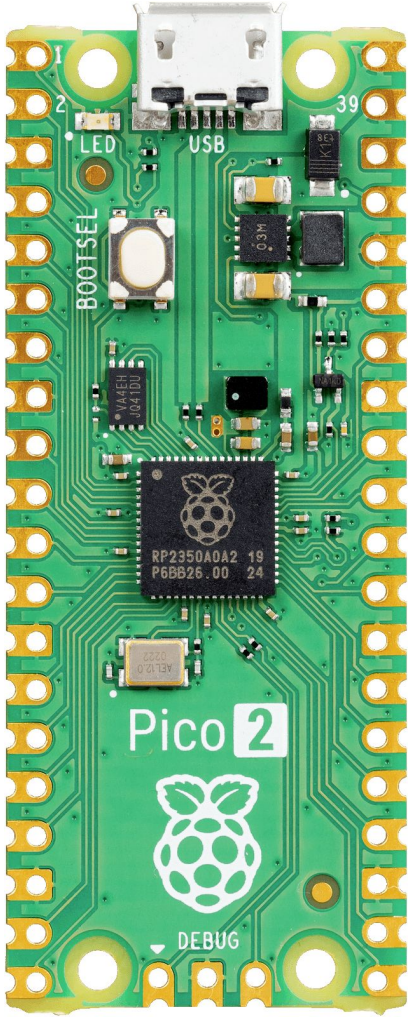
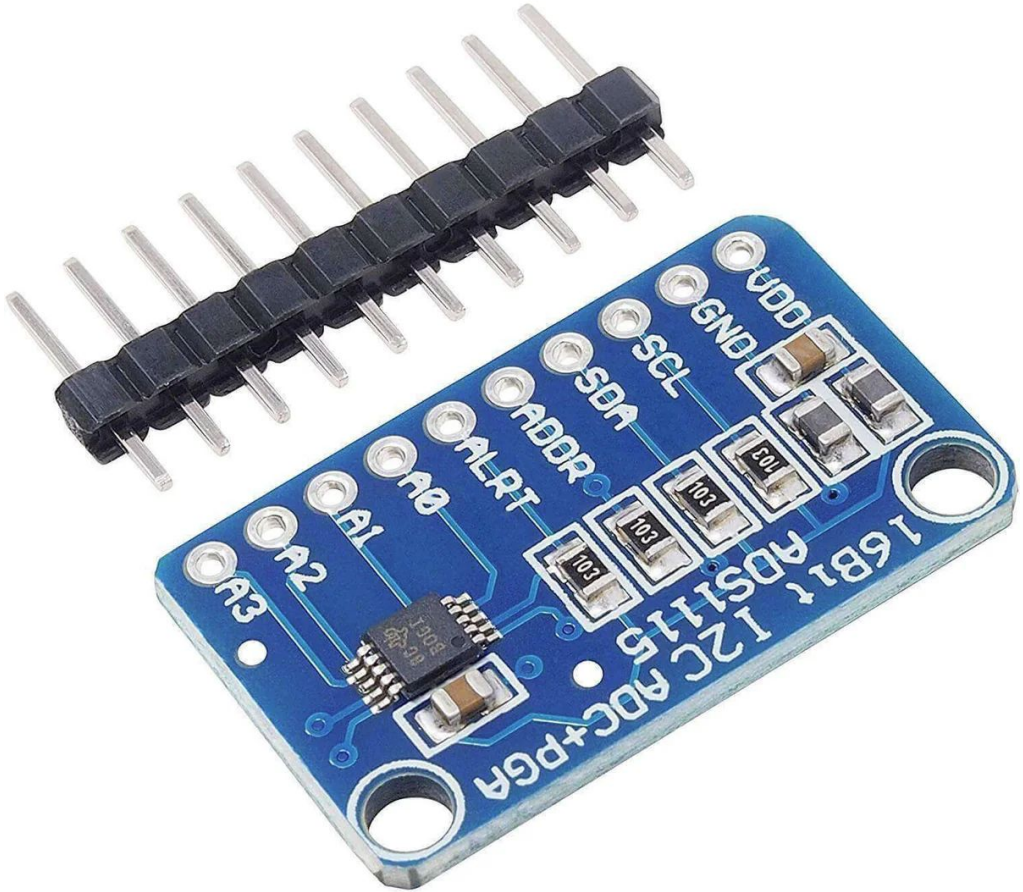


Raspberry Pi Pico 2

Raspberry Pi Pico 2 is a low-cost, high-performance microcontroller board with flexible digital interfaces. Key features include:

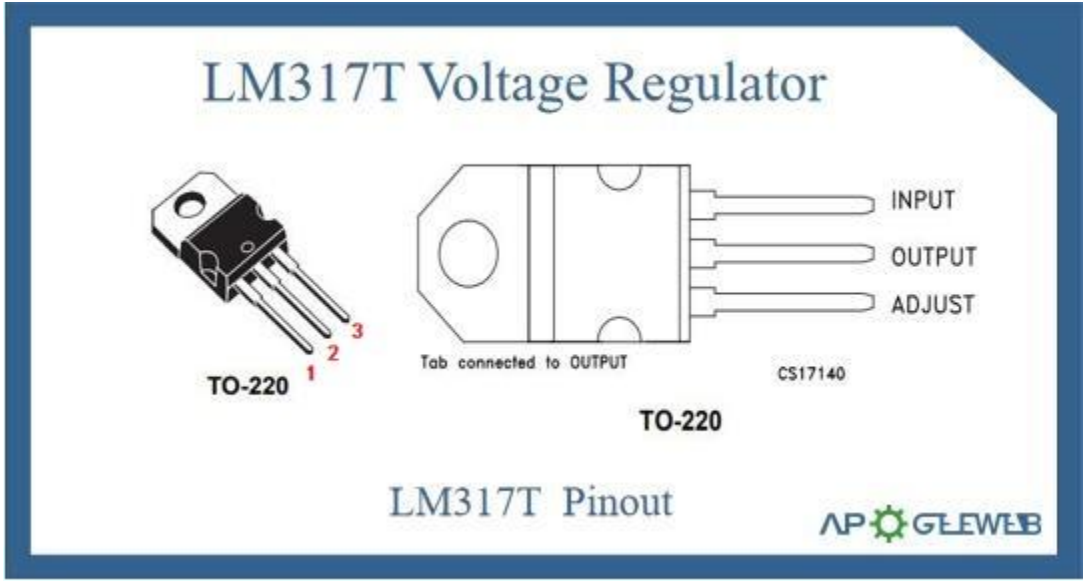
- RP2350 microcontroller chip designed by Raspberry Pi in the United Kingdom
- Dual Cortex-M33 or Hazard3 processors at up to 150MHz
- 520 KB of SRAM, and 4MB of on-board flash memory
- USB 1.1 with device and host support
- Low-power sleep and dormant modes
- Drag-and-drop programming using mass storage over USB
- 26× multi-function GPIO pins including 3 that can be used for ADC
- 2× SPI, 2× I2C, 2× UART, 3× 12-bit 500 ksps Analogue to Digital Converter (ADC), 24× controllable PWM channels
- 2× Timer with 4 alarms, 1× AON Timer
- Temperature sensor



ADS1115 I2C 16-Bit ADC 4 channel Module

This module is a 4-channel 16-bit ADC converter based on ADS1115 chip. You rarely would want a resolution higher than 16-bit, making this module an all-around perfect choice for an external ADC.

This is ADS1115 16-Bit ADC – 4 Channel with Programmable Gain Amplifier. For microcontrollers without an analog-to-digital converter or when you want a higher-precision ADC, the ADS1115 provides 16-bit precision at 860 samples/second over I2C. The chip can be configured as 4 single-end input channels or two differential channels. As a nice bonus, it even includes a programmable gain amplifier, up to x16, to help boost up smaller single/differential signals to the full range. We like this ADC because it can run from 2V to 5V power/logic, can measure a large range of signals and it’s super easy to use. It is a great general purpose 16-bit converter.



8.3.3 Precision Current-Limiter Circuit

This application limits the output current to I_{LIMIT} in Figure 8-8.

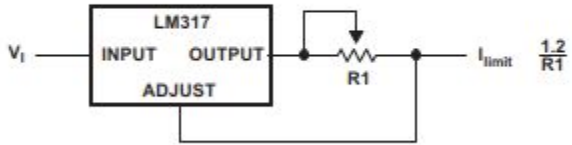
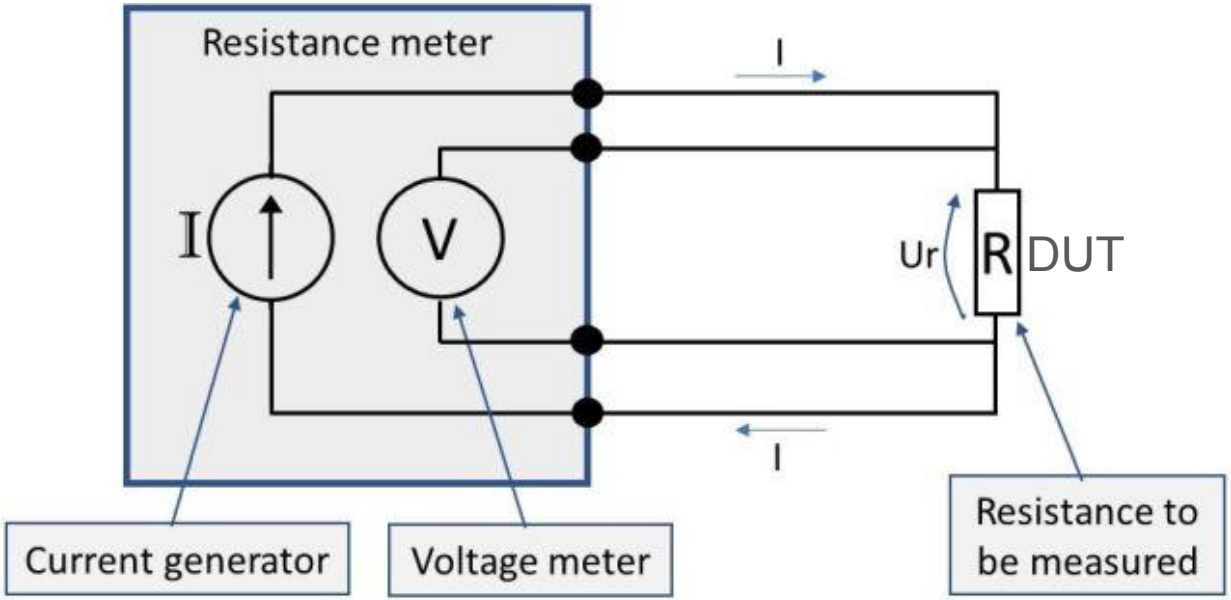


Figure 8-8. Precision Current-Limiter Circuit

Four-wire Kelvin Testing



The principal advantage of 4-wire measurement is that it eliminates any effect of fixture resistance (the lead wires) to obtain a precise resistance value of the DUT. Because 4-wire measurements typically employ test currents well above those needed for two-wire testing, a secondary advantage comes through the use of a high-current stress test for wiring by driving a current of 1 A or more through each conductor, and the ability to set a dwell time from 100 ms to many minutes — observing a slowly-increasing resistance during a long dwell period resulting from thermal heating may reveal problems not detected with a shorter measurement interval.

Αυτός ο κώδικας είναι ένα **“Milliohm Meter”** με Raspberry Pi Pico, ADS1115 ADC, 4-wire Kelvin μετρήσεις, και μια OLED οθόνη sh1106.

📌 Εισαγωγές βιβλιοθηκών

python Copy code

```python
from machine import Pin, I2C
import time
from ads1x15 import ADS1115
import sh1106
```

- `machine.Pin`, `machine.I2C`: drivers του MicroPython για GPIO και I2C.
- `time`: για καθυστερήσεις (ms/s).
- `ads1x15`: βιβλιοθήκη για τον ADC **ADS1115**, που έχει 16-bit ανάλυση.
- `sh1106`: βιβλιοθήκη για τον OLED controller **SH1106**.

📌 Median function

python

Copy code

```python
try:
    from statistics import median
except:
    def median(seq):
        s = sorted(seq)
        n = len(s)
        return s[n//2] if n % 2 else 0.5*(s[n//2-1] + s[n//2])
```

- Χρησιμοποιείται το `median` (διάμεσος) για να απορρίπτει spikes/θόρυβο από τις μετρήσεις.
- Αν δεν υπάρχει η βιβλιοθήκη `statistics` στο MicroPython, φτιάχνουμε μια δική μας έκδοση.

📌 Κλάση MilliohmMeterPro

Αυτή είναι η "καρδιά" του οργάνου.

python Copy code

```python
class MilliohmMeterPro:
    def __init__(self, i2c, address=0x48, gain=5, current=0.100):
        self.adc = ADS1115(i2c, address=address, gain=gain)
        self.current = current
        self.v_offset = 0.0
        self.scale = 1.0
```

- `self.adc` : Αντικείμενο ADS1115 (16-bit ADC).
- `current` : το ρεύμα που δίνουμε στο DUT (Device Under Test). Εδώ 100 mA.
- `v_offset` : η μέση τιμή offset, για να μη μετράμε θόρυβο όταν είναι βραχυκυκλωμένο.
- `scale` : συντελεστής κλίμακας, χρησιμοποιείται [illegible] calibration.

🔷 Μέθοδος `_continuous_samples`

python

Copy code

```
def _continuous_samples(self, n=32, rate=0, ch1=0, ch2=1):
    self.adc.conversion_start(rate=rate, channel1=ch1, channel2=ch2)
    samples = []
    for _ in range(n):
        raw = self.adc.alert_read()
        v   = self.adc.raw_to_v(raw)
        samples.append(v)
        time.sleep_ms(8)
    return samples
```

- Βάζει το ADS1115 να μετράει συνεχώς τη διαφορά μεταξύ δύο καναλιών (Kelvin sense).
- Συλλέγει `n` δείγματα, τα μετατρέπει σε Volt, και τα επιστρέφει.

🔷 Zero (Offset)

python

Copy code

```python
def zero(self, n=64, rate=0, ch1=0, ch2=1):
    vs = self._continuous_samples(n=n, rate=rate, ch1=ch1, ch2=ch2)
    self.v_offset = median(vs)
    return self.v_offset
```

- Ο χρήστης βραχυκυκλώνει τις Kelvin ακίδες.
- Μετράμε την τάση και αποθηκεύουμε τη διάμεσο ως `v_offset`.
- Αυτό διορθώνει offset λόγω θορύβου/σφάλματος του ADC.

🔷 Μέτρηση

python

Copy code

```python
def measure_once(self, n_groups=5, per_group=16, rate=0, ch1=0, ch2=1):
    group_means = []
    for _ in range(n_groups):
        vs = self._continuous_samples(n=per_group, rate=rate, ch1=ch1, ch2=ch2)
        vs_corr = [v - self.v_offset for v in vs]
        group_means.append(sum(vs_corr)/len(vs_corr))
    v_med = median(group_means)
    R = (v_med / self.current) * self.scale if self.current > 0 else None
    return R, v_med
```

- Παίρνουμε πολλές ομάδες μετρήσεων.
- Για κάθε ομάδα, διορθώνουμε με το `v_offset` και υπολογίζουμε μέσο όρο.
- Από τις μέσες τιμές ομάδων, παίρνουμε τη **διάμεσο** (πιο ανθεκτική σε θόρυβο).
- Υπολογίζουμε την αντίσταση με τον νόμο του Ohm:

$$R = \frac{V}{I} \cdot scale$$

🔷 Calibration

python

Copy code

```python
def calibrate_with_reference(self, R_ref_ohm, n_groups=6, per_group=20, rate=0, ch1=0, ch2=1):
    R_meas, _ = self.measure_once(n_groups=n_groups, per_group=per_group, rate=rate, ch1=ch1, ch2=
    if (R_meas is not None) and (R_meas > 0):
        self.scale = R_ref_ohm / R_meas
    return self.scale
```

- Ο χρήστης συνδέει μια γνωστή αντίσταση (π.χ. 0.100 Ω).
- Μετράμε και υπολογίζουμε τον συντελεστή διόρθωσης `scale`.
- Έτσι βελτιώνεται η ακρίβεια.

📌 Κύριο πρόγραμμα

python

Copy code

```
i2c = I2C(1, scl=Pin(15), sda=Pin(14), freq=400000)
meter = MilliohmMeterPro(i2c, current=0.100, gain=5)
oled = sh1106.SH1106_I2C(128, 64, i2c, addr=0x3C)
```

- Φτιάχνουμε I2C bus.
- Δημιουργούμε το όργανο `meter`.
- Φτιάχνουμε την OLED οθόνη.

Zero Offset

python

Copy code

```
print("=== Zero-offset ===")
print("Βραχυκύκλωσε τα Kelvin sense leads...")
time.sleep(5)
offset = meter.zero(n=80, rate=0)
print("Offset =", offset*1e6, "μV")
```

- Οδηγία στον χρήστη να βραχυκυκλώσει τις ακίδες.
- Υπολογίζει και τυπώνει το offset (σε μV).

Loop μετρήσεων

python

Copy code

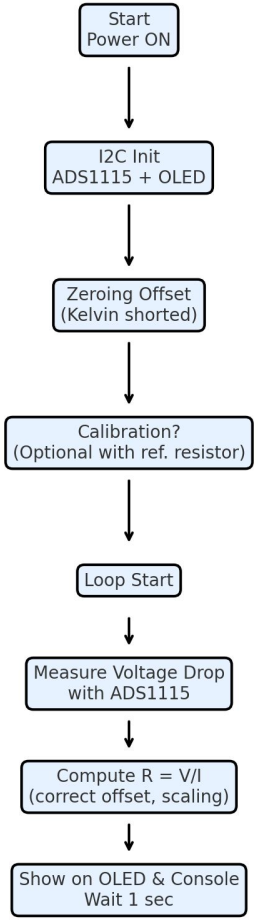
```python
while True:
    R, V = meter.measure_once(n_groups=7, per_group=24, rate=0)
    text_console = "U = {:.6f} V , R = {:.3f} mΩ".format(V, (R or 0)*1000)
    print(text_console)

    oled.fill(0)
    oled.text("Milliohm Meter", 0, 0)
    oled.text("U: {:.6f} V".format(V), 0, 20)
    oled.text("R: {:.3f} mOhm".format((R or 0)*1000), 0, 40)
    oled.show()
    time.sleep(1)
```

- Κάνει συνεχείς μετρήσεις.
- Υπολογίζει την τάση και την αντίσταση σε mΩ.
- Εμφανίζει τιμές σε **serial console** και στην **OLED**.

📌 Τι πετυχαίνει το πρόγραμμα

- Έχει **4-wire Kelvin μέτρηση** για ακρίβεια.
- Υποστηρίζει **μηδενισμό offset**.
- Υποστηρίζει **calibration με γνωστή αντίσταση**.
- Δείχνει τα αποτελέσματα σε **OLED** και **console**.

Flowchart Milliohm Meter



Επεξηγηση στο **διάγραμμα ροής** του προγράμματος:

- Ξεκινάει με το **Power ON**.
- Αρχικοποιεί **I2C, ADS1115, OLED**.
- Κάνει **Zero Offset** με βραχυκυκλωμένα Kelvin leads.
- Προαιρετικά κάνει **Calibration** με γνωστή αντίσταση.
- Μπαίνει σε **Loop** όπου:
    1. Μετράει τάση με το ADS1115.
    2. Υπολογίζει την αντίσταση R=V/IR = V/IR=V/I.
    3. Εμφανίζει στην **OLED + Console**.
    4. Περιμένει 1 δευτερόλεπτο και επαναλαμβάνει.