How to monitor 12V battery charge with a Raspberry Pi Pico

raspberrypi
pico
diy
30 minutes
Jan 05 2024

Last year I built a 12V powerbank to power my astrophotography rig and to light my shed. It has been working great so far, in fact I liked it so much I built a second one. While I've been very happy with my DIY powerbanks in general, they lack one functionality, and that is remote monitoring of the battery charge level. It would be awesome if I could check the battery voltage remotely, while the rig is working out there in the cold night. This need inspired me to look for battery monitoring solutions, but from the ones I found, none were interesting, and after all, it's much more fun to build something by yourself, rather than take something off the shelf.

I have a Raspberry Pi Pico W, which I already used for other projects, such as making a weather station and I decided I want to do something based on it. When browsing the internet, I came across this forum thread showing how to measure voltage with the Pico.

Theoretical background

The Raspberry Pi Pico has 3 ADC (Analogue to Digital Converter) inputs. Analogue in this case meaning it can measure voltage as a continuous spectrum, in contrast to the digital pins, which only work in binary, 0 and 1, voltage and no voltage. In this project I am using one of those inputs to measure the voltage of the battery. Now, the problem is that, it can only accept voltages up to 3.3V. If I wanted to measure, let's say, the voltage of a 1.2V AA battery that would be fine, but I want to monitor a 12V battery, which voltage can range from 13.5V at full charge, to 11V at depletion. That's why we need to use a tool from the basics of electronics: a voltage divider.

A voltage divider is a circuit which consists of two resistors connected in series. The voltage measured between the point between the resistors (Vout) and the negative battery terminal will be lower than the voltage of the battery itself (Vin). The voltage drop is constant and based on the values of the resistors, so by measuring the lowered voltage and knowing the conversion rate, we can calculate the voltage of the battery. Looking at the diagram should give you a better understanding of the whole idea.

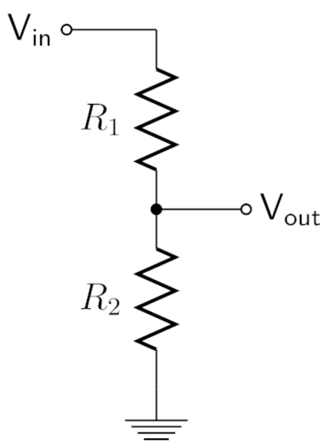


Image taken from Wikipedia commons.wikimedia.org/w/index.php?curid=36347358

Hardware

To build this battery voltage measuring device, we will need:

- A microcomputer or a microcontroller with GPIOs, with at least one analogue GPIO. I am using a Raspberry Pi Pico, because this is what I already have around.
- a prototype board
- two resistors
- cables
- the usual tools: a soldering iron, cable crimpers and strippers
- optionally: a buck converter to power the Pico from the 12V battery, but that won't be needed if you want to power the Pico from an external power source. I bought a variable step-down converter based on a LM2596 chip.

Choosing the resistors

We will need the voltage divider to reduce the incoming battery voltage (13.5V - 11V) to a value acceptable by the Pi Pico, which is below 3.3V.

Below is the formula to calculate the voltage drop. There is also a handy online calculator available.

$$V_{out} = V_{in} * \frac{R2}{R1 + R2}$$

After doing some trial and error calculations I went with a 75 kOhm and a 20 kOhm resistor. This way even at the maximum cell voltage of ~13.5V, the voltage coming to the Pico will be around 3V. The resistors are in the kiloohm range to reduce the amount of current flowing through the Pi.

Connecting stuff

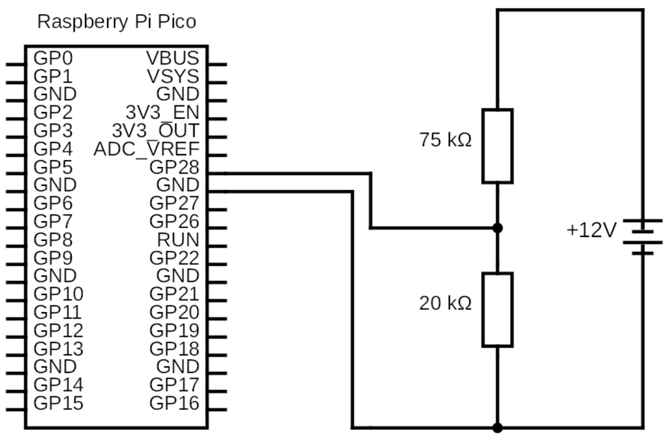
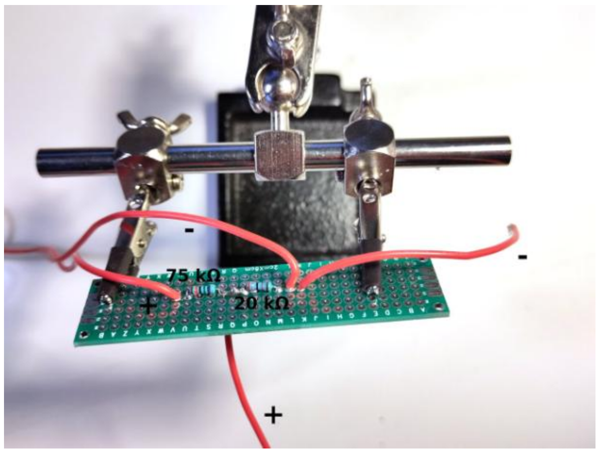


Diagram made with Circuit-Diagram

The circuit is very simple. The branch starts with the positive connector of the battery, goes through both resistors and returns to the negative connector. From a point between the resistors, another branch connects to one of the analogue pins on the Pi. I went with Pin 28, as it’s the nearest one to the AGND (Analogue Ground Pin). A final branch goes from the AGND to the negative connector on the battery. And we’re done.

I used a prototype board to connect all of the components. First I went with soldering the resistors to the prototype board. Remember or mark which resistor is the higher value one and which is the lower value. I left one hole of space between the resistors for a cable going to the Pico.

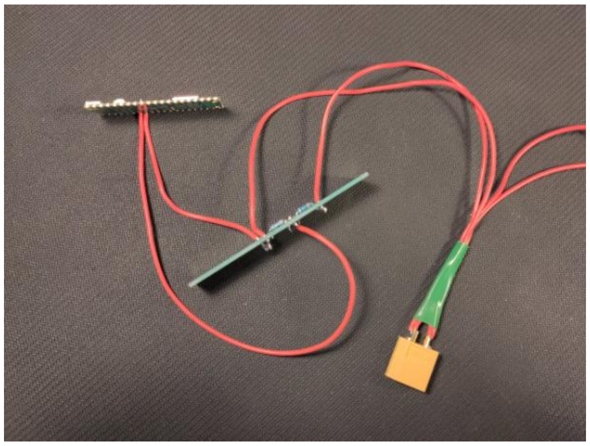
The next step is to solder the cables going from the battery to the resistors. If you are going to use a plug to connect the device to a battery as I am, it’s a good time to solder it now. Pay attention to the polarity of the plug, the positive cable needs to be connected to the resistor with the higher value.



Now solder to the resistors the two cables going from the board to the Pico. Don’t solder cables to the Pico itself yet! Once you have the resistors and the cables soldered, this is a good point to test that the voltage divider actually works. Connect the two battery cables to a battery and with a multimeter, measure the voltage between the the cables that will be connected to the Pico. If the battery is close to full charge, the measured voltage should be around 3V. If that is the case, cool, we’re home. If not, something went very wrong with the circuit, check all the connections again.

Now connect the board-to-Pico cables to the analogue input, and the analogue ground of the Pi.

The soldering part is done. Take a double look at your work and make sure everything is connected properly. With the Pico UNPLUGGED from the computer, connect the battery cables to the battery socket. Wait a few minutes. Nothing should get hot, nothing should emit smoke. If everything looks fine, it’s time to connect the Pico the a computer and start working on the software side.



Software

I am assuming you know how to access the Pi Pico in a code editor and you are using MicroPython. I am using Thonny to write and run code on the Pi.

Reading the voltage

Let’s start with something very simple, just measuring the voltage and printing it out.

```
from machine import Pin, ADC
from time import sleep
analogue_input = ADC(28)

VOLTAGE_DROP_FACTOR = 4.572

while True:
    sensor_value = analogue_input.read_u16()
    voltage = sensor_value * (3.3 / 65535) * VOLTAGE_DROP_FACTOR
    print(voltage)
    sleep(30)
```

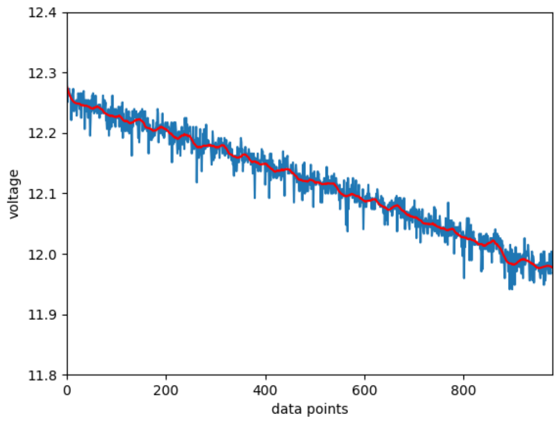
Most of the code should be obvious. Wer are importing the needed modules, measuring the voltage in a loop, and printing the measured value. But let’s talk about this line.

```
voltage = sensor_value * (3.3 / 65535) * VOLTAGE_DROP_FACTOR
```

This is how the voltage on the Pico should be measured. The actual value returned by measuring the analogue input is a number between 0 and 65535, with the max value meaning 3.3V and 0 meaning 0V. By multiplying that value by 3.3 we are getting the voltage coming through the analogue pin. However, we are not exactly interested in the voltage at the Pi pin, we want to know the voltage of the battery. We are using a voltage divider that drops the battery voltage, right? Therefore, we need to multiply the Pi voltage by the voltage divider factor. Here I went with the simplest approach. I measured at the same time the voltage reported by the pi, and, using a multimeter, the battery voltage and calculated the difference factor. For me it is 4.572, but as every resistor differs a tiny bit, it will be different for everyone else. You will need to measure and calculate your own drop factor.

The part below is totally optional, feel free to skip it.

When testing if the whole device actually works, I let it run for a few hours and saved the measured values to a file, and plotted it in a Jupyter Notebook using matplotlib.



Here is the code used to generate the plot:

`voltages` is a file with each value on its own line. The raw values from the file are plotted in blue, and as you can see the data is noisy, so I used a smoothing filter for the red line. I found how to do it here on stackoverflow

```
import matplotlib.pyplot as plt
import numpy as np
from math import factorial
from scipy.signal import savgol_filter

with open("voltages") as f:
    lines = f.readlines()
lines = [float(line.strip()) for line in lines]
lines = np.array(lines)

x = np.array([x for x in range(len(lines))])

smooth_lines = savgol_filter(lines, 51, 3)
plt.plot(x, lines)
plt.plot(x, smooth_lines,color='red')
plt.ylabel('voltage')
plt.xlabel('data points')
plt.axis([0, len(lines), 11.8, 12.4])
plt.savefig('voltage.png')
plt.show()
```

Adding internet connectivity

Once we are able to read the voltage, it would be good to send it somewhere, wouldn't it? And for that I am using the power of the Internet.

First we need to connect the Pi to a WiFi network. Here's the code to do it. Save it on the Pico in a file called `boot.py`, and this way the script will be run every time the Pico is powered on. This is exactly the same code I used in the Pico Weather Station post. I believe the code is self-explanatory, but is something is unclear, refer to that previous post, I explain it there in detail.

```
import time
import network
import machine

led = machine.Pin("LED", machine.Pin.OUT)

ssid = "NETWORK-NAME"
password = "PASSWORD"

wlan = network.WLAN(network.STA_IF)
wlan.active(True)
wlan.connect(ssid, password)

max_wait = 10
while max_wait > 0:
    led.on()
    time.sleep(0.1)
    led.off()
    if wlan.isconnected():
        break
    max_wait -= 1
    time.sleep(1)

if not wlan.isconnected():
    for _ in range(5):
        led.on()
        time.sleep(0.1)
        led.off()
        time.sleep(0.1)
        led.on()
        time.sleep(0.1)
        led.off()
        time.sleep(0.7)
    time.sleep(1)
    machine.reset()
else:
    status = wlan.ifconfig()
    led.on()
    time.sleep(3)
    led.off()
```

Pico web server

Next we need to make the voltage information available. The simpler way is to make a web server out of the Pico. I took the code below from a guide on the Raspberry Pi webpage and modified it a bit to accommodate to my needs.

In short, the Pi will be listening for any incoming connections, and once a client talks to the Pi, it will measure the voltage and return it in a JSON format. If you prefer HTML rather than JSON, check the guide linked above for an example.

One issue with using the Pi as a webserver is that you will need to know it's IP address to connect to it. To find the IP address you can use `nmap` under Linux, or an iOS/Android wifi scanner application.

Save the file below as `main.py`. `boot` and `main` are special names for files on the Pico. Every time the Pico is powered on, first the `boot` file is run, and then the `main` one. This happens even when the Pico is not connected to a computer.

```
import json
import network
import socket
import time

VOLTAGE_DROP_FACTOR = 4.572

analogue_input = machine.ADC(28)

addr = socket.getaddrinfo('0.0.0.0', 80)[0][-1]

s = socket.socket()
s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
s.bind(addr)
s.listen(1)

while True:
    try:
        cl, addr = s.accept()

        sensor_value = analogue_input.read_u16()
        voltage = sensor_value * (3.3 / 65535) * VOLTAGE_DROP_FACTOR

        response = {
            "voltage": voltage,
        }

        cl.send('HTTP/1.0 200 OK\r\nContent-type: application/json\r\n\r\n')
        cl.send(json.dumps(response))
        cl.close()

    except OSError as e:
        cl.close()
```

Sending data with MQTT

And here is the MQTT version. I prefer using MQTT over HTTP as I can pipe MQTT messages into my InfluxDB/Grafana monitoring pipeline and put the data into a nice graph.

To use MQTT you will need to install the `umqtt` library on the Pico, Thonny support installing additional packages using the “Tools -> Manage packages” menu.

What is also the upside of using MQTT is that we do not need to know the IP of the Pico, here the Pi needs to know the IP address of the MQTT broker, which is usually static or just more stable than the IP address of the Pico.

The `main.py` file for using MQTT:

```
import json
import time
import machine
from umqtt.simple import MQTTClient

led = machine.Pin("LED", machine.Pin.OUT)
analogue_input = machine.ADC(28)

VOLTAGE_DROP_FACTOR = 4.572

while True:
    sensor_value = analogue_input.read_u16()
    voltage = sensor_value * (3.3 / 65535) * VOLTAGE_DROP_FACTOR

    payload = {
        "voltage": voltage,
    }

    qt = MQTTClient("umqtt_client", "<MQTT BROKER IP>", keepalive=3600)
    qt.connect()
    qt.publish(b"battery", json.dumps(payload))
    qt.disconnect()
    led.on()
    time.sleep(0.5)
    led.off()
    time.sleep(30)
```

The code is very similar to the one shown in the Pico Weather Station post.

The main difference here is that with MQTT we are pushing the data in set intervals to an MQTT broker, a server that handles the incoming messages and allow other clients, such as InfluxDB, to read them and use them to for example create a graph. `battery` is the name of the topic, in MQTT topics are a way to separate different types of messages from each other. This part of the code sets up an MQTT client instance, connects and sends the payload.

```
qt = MQTTClient("umqtt_client", "<MQTT BROKER IP>", keepalive=3600)
qt.connect()
qt.publish(b"battery", json.dumps(payload))
qt.disconnect()
```

This is not the place to dive into MQTT specifics, if you want to start using this protocol, start with checking out Mosquitto, a popular MQTT broker.

Base done

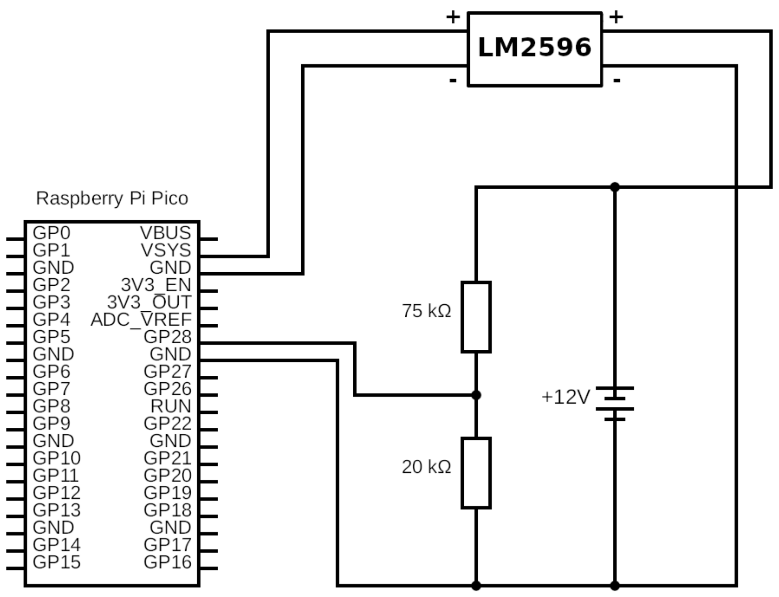
The base is now done, we have a working battery monitoring device that can send the battery status over the network. When connected to a battery, and powered via USB from a powerbank or a wall charger, it will connect to the internet and provide information on the voltage level of a 12V battery.

Now let’s tackle the issue that is is being powered from a power source that is external to the 12V battery. This can be seen as either a feature or a flaw of the design. A feature because this way it won't drain the main battery, and will run even if the main battery is temporarily turned off. A flaw because it adds a second battery that needs to be charged, and together with that additional cabling and complication in general.

Changing the power source

In this section let's talk about modifying the device so that it does not require an additional power source to run.

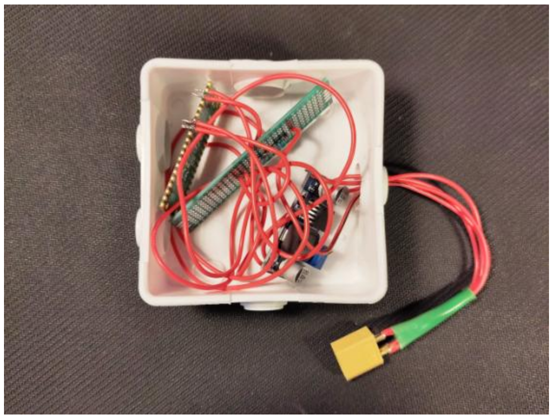
The Pi Pico can be powered from a 5V power source, either via the USB port or the VSYS and GND pins. In my previous project I went with powering it with the pins, and this is also what I am doing here.



I ordered a buck converter that can take a higher input voltage and output a stable 5V. The on I bought is a variable output converter based on the LM2596 chip. Before using with the Pi, it needs to be configured. There is a tiny potentiometer on the board, and by turning it you can set the output voltage. I connected the buck converter to a 12V battery and checked the output voltage with a multimeter. I turned the potentiometer until the output was exactly 5V. Now the thing that was left, was to connect the converter output to the VSYS and GND pins on the Pi. As always, follow the polarity.

And done, now there is only one plug coming out of the box. When the device is plugged into a 12V power source, it will automatically power up and start transmitting battery levels.

The finished project



Final words

I still need to tweak some things, when I finished the soldering I realized the cables are a bit short. I also need to weather-proof the box. I’m also thinking of adding a thermometer to monitor the battery temperature, but that is something for another project and another blog post.

This has been the longest and most complex blog post so far. I hope you enjoyed it and will build your own device! As always, I would love to hear feedback from you. If you have thoughts on what I wrote, please let me know either via email, or by talking to me on Mastodon. Links are in the footer. Thank you so much for reading!

And finally, you can help with funding my future projects by supporting me on these crowdfunding sites: