

Python & MicroPython: Programmieren lernen für Einsteiger

Erfassung analoger Werte

Teil 5

Einer der wesentlichen Nachteile des Raspberry Pi gegenüber vielen Mikrocontrollern wie dem Arduino oder dem ESP32 ist, dass der Pi nicht über analoge Eingänge verfügt. Die reale Welt ist aber analog. Bei Sonnenaufgängen wird es allmählich hell. Lufttemperaturen ändern sich stetig und nicht sprunghaft. Um also Umweltparameter exakt erfassen zu können, ist das Messen analoger Werte erforderlich. Auch bei dieser Aufgabe zeigt Python seine universelle Funktionalität. So ist es mit speziellen Verfahren möglich, ein Python-Programm zu erstellen, das analoge Werte erfassen kann, ohne dass dazu zusätzliche Hardware erforderlich wäre. Will man allerdings Messwerte mit hoher Präzision erfassen, kommt man um einen Analog-Digital-Konverter nicht herum. Der MCP3002 leistet hier gute Dienste.



In diesem Beitrag werden wir die Erfassung analoger Messwerte ausführlich in den folgenden Themen behandeln:

- Erfassung von Spannungspegeln über Zeitmessung
- Einsatz eines Analog-Digital-Konverters zur präzisen Erfassung analoger Messwerte
- grafische Darstellung von Spannungspegeln in der Thony Shell
- Aufbau und Test eines computergesteuerten Digitalvoltmeters mit Python

Damit werden dann auch Anweisungen und Befehle klar, die in früheren Beispielen bereits erforderlich waren, aber noch nicht im Detail diskutiert wurden.

Für Minimalisten: Spannungspegel über Zeitmessung erfassen

Mithilfe der digitalen Port-Pins lassen sich bereits mit minimalem zusätzlichem Hardware-Aufwand einfache Messaufgaben erledigen. Im Folgenden wird ein Python-Programm vorgestellt, das eine einfache Temperaturmessung ausführt, ohne dass dafür ein spezieller Analog-Digital-Konverter (ADC) erforderlich wäre.

Dazu kommt ein temperaturabhängiger Widerstand zum Einsatz. Dieser sogenannte NTC 10k hat bei 25 °C einen Widerstandswert von 10 kΩ. Bei höheren Temperaturen nimmt dieser Widerstand ab, bei geringeren zu.

Für das Laden bzw. Entladen eines Kondensators über diesen Widerstand werden also je nach Umgebungstemperatur unterschiedliche Zeiten benötigt. Diese Zeiten kann der Raspberry Pi problemlos und exakt messen. Ein passender Hardware-Aufbau ist in [Bild 1](#) dargestellt. Als Bauelemente können entweder Einzelkomponenten (wie in [Bild 1](#)) oder Module aus den PAD-Sets (s. Kasten Material) verwendet werden ([Bild 2](#)).

WICHTIG: Damit der Raspberry keinen Schaden nimmt, sollte der Kondensator immer entladen werden, bevor er in die Schaltung eingesetzt wird. Das kann erreicht werden, indem man die beiden Anschlüsse des Elkos kurzzeitig miteinander verbindet.

Im zugehörigen Python-Programm wird ein Pin als Output deklariert und auf HIGH, d. h. auf 3,3 V geschaltet. Dann wird der Kondensator geladen. Sobald eine bestimmte Ladespannung erreicht wird, erkennt ein als Eingang definierter Pin ebenfalls einen HIGH-Pegel. Die Zeit bis zum Erreichen dieses Pegels wird gemessen und dient als Maß für die Temperatur.

Über zwei Parameter – `cal` und `offset` – wird aus der gemessenen Zeit eine Temperatur errechnet. Falls die angezeigte Temperatur nicht mit der wirklichen Umgebungstemperatur übereinstimmt, kann man die beiden Parameter entsprechend anpassen.

Der NTC-Thermistor und der Kondensator bilden ein RC-Glied (Widerstand-Kondensator-Netzwerk). Die Entladezeit des Kondensators hängt vom Wider-

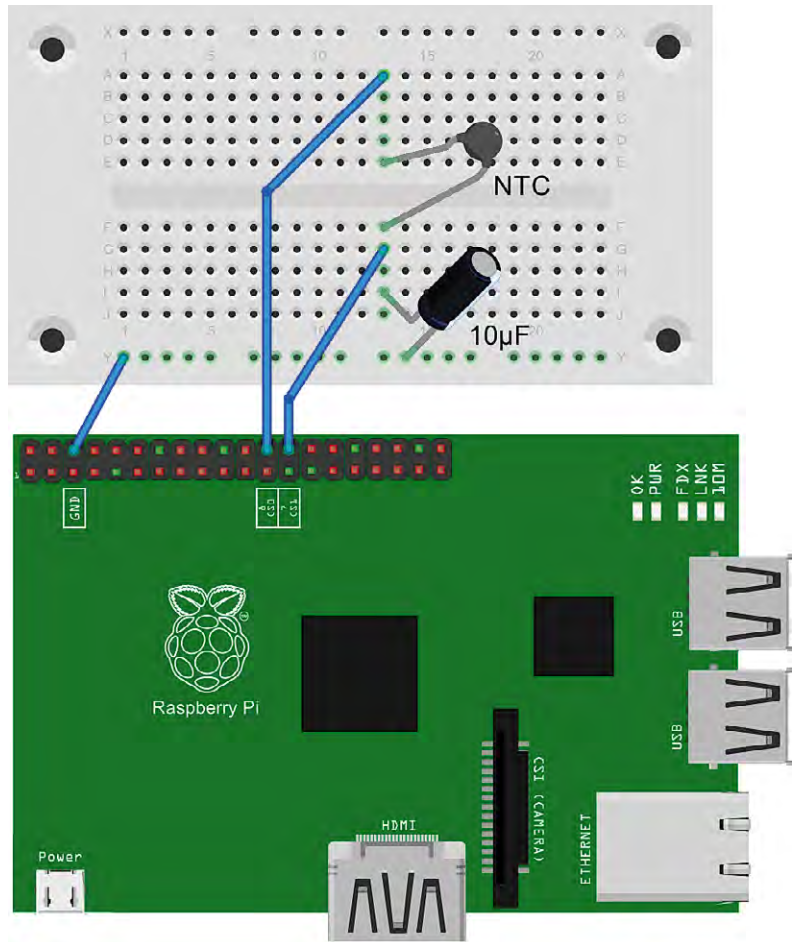


Bild 1: Temperaturmessung ohne ADC



Bild 2: Aufbau Bild 1 mit PAD-Set

stand des NTC-Thermistors ab, der wiederum temperaturabhängig ist. Durch Messen der Entladezeit kann auf die Temperatur geschlossen werden, da der Widerstand des NTC-Thermistors mit der Temperatur variiert.

Der Kalibrierungsfaktor und der Offset müssen experimentell bestimmt werden, um genaue Temperaturwerte zu erhalten (s. u.). Die vorgegebenen Werte sollten aber einen brauchbaren Startwert liefern.

Dieses Programm (NTC_RC.py) nutzt also eine Zeitmessung, um die analoge Spannung an einem Kondensator zu überwachen und daraus eine Temperatur abzuleiten.

```
# Pin Config: activePort-NTC 10k-readoutPort-10uF-GND

import RPi.GPIO as GPIO
import time

GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)

activePort = 8
readoutPort = 7
cal = 1000.0
offset = 100.0

GPIO.setup(activePort, GPIO.OUT)
GPIO.setup(readoutPort, GPIO.IN)

while(True):
    GPIO.output(activePort, 1)
    time.sleep(1)
    GPIO.output(activePort, 0)
    tStart = time.time()
    while GPIO.input(readoutPort) > 0:
        pass
    tStop = time.time()
    T = int (offset - ((tStop-tStart)*cal))
    print(T)
    time.sleep(1)
```

Im Programm wird zunächst die Pin-Konfiguration festgelegt:

activePort (Pin 8):

Dieser Pin wird verwendet, um den Kondensator aufzuladen.

readoutPort (Pin 7):

Dieser Pin misst den Spannungspegel am Kondensator

Die Variablen **cal** und **offset** dienen als Kalibrierfaktoren:

cal sorgt für die Umrechnung der gemessenen Zeit in eine Temperatur, **offset** dient zur Anpassung des Temperaturstartpunkts. Im Setup

werden zunächst GPIO-Warnungen unterdrückt. Dann wird der GPIO-Modus auf BCM (Broadcom Channels) gesetzt. Im Pin-Setup wird über `GPIO.setup(activePort, GPIO.OUT)` der „activePort“ als Ausgang festgelegt. Über `GPIO.setup(readoutPort, GPIO.IN)` wird der `readoutPort` als Eingang definiert.

In der Hauptschleife wird der Kondensator aufgeladen, indem `activePort` auf HIGH gesetzt wird. Mit `time.sleep(1)` wird eine Sekunde gewartet, damit der Kondensator sicher vollständig aufgeladen wird.

Dann wird der `activePort` auf LOW geschaltet, um das Entladen des Kondensators zu starten. Die Anweisung `tStart = time.time()` speichert den aktuellen Zeitpunkt als Startzeit. Die Anweisung

```
while GPIO.input(readoutPort) > 0
```

wartet, bis der Spannungspegel am `readoutPort` auf LOW fällt. Dies ist der Fall, wenn der Kondensator weitgehend entladen ist. Die Variable

```
tStop = time.time()
```

speichert diesen Zeitpunkt als Stoppzeit.

Dann wird über

```
T = int(offset - ((tStop - tStart) * cal))
```

die Temperatur basierend auf der Zeit, die der Kondensator zum Entladen benötigt hat, berechnet und über `print(T)` ausgegeben. Mit `time.sleep(1)` wird eine Sekunde bis zur nächsten Messung gewartet.

Bild 3 zeigt den Signalverlauf an den beiden Pins.

Mit den voreingestellten Parametern wird zunächst ein Wert ausgegeben, der meist relativ stark von der tatsächlichen Umgebungstemperatur abweicht.

Man sollte zunächst den Offset-Parameter verändern, um die aktuelle Temperatur (z. B. von einem Vergleichsthermometer) einzustellen. Dann wird der Sensor erwärmt, z. B. durch vorsichtiges Anblasen mit einem Fön. Ist die Temperatur am Vergleichsthermometer um ca. 10 °C angestiegen, sollte auch die Temperaturanzeige des NTC-Thermometers

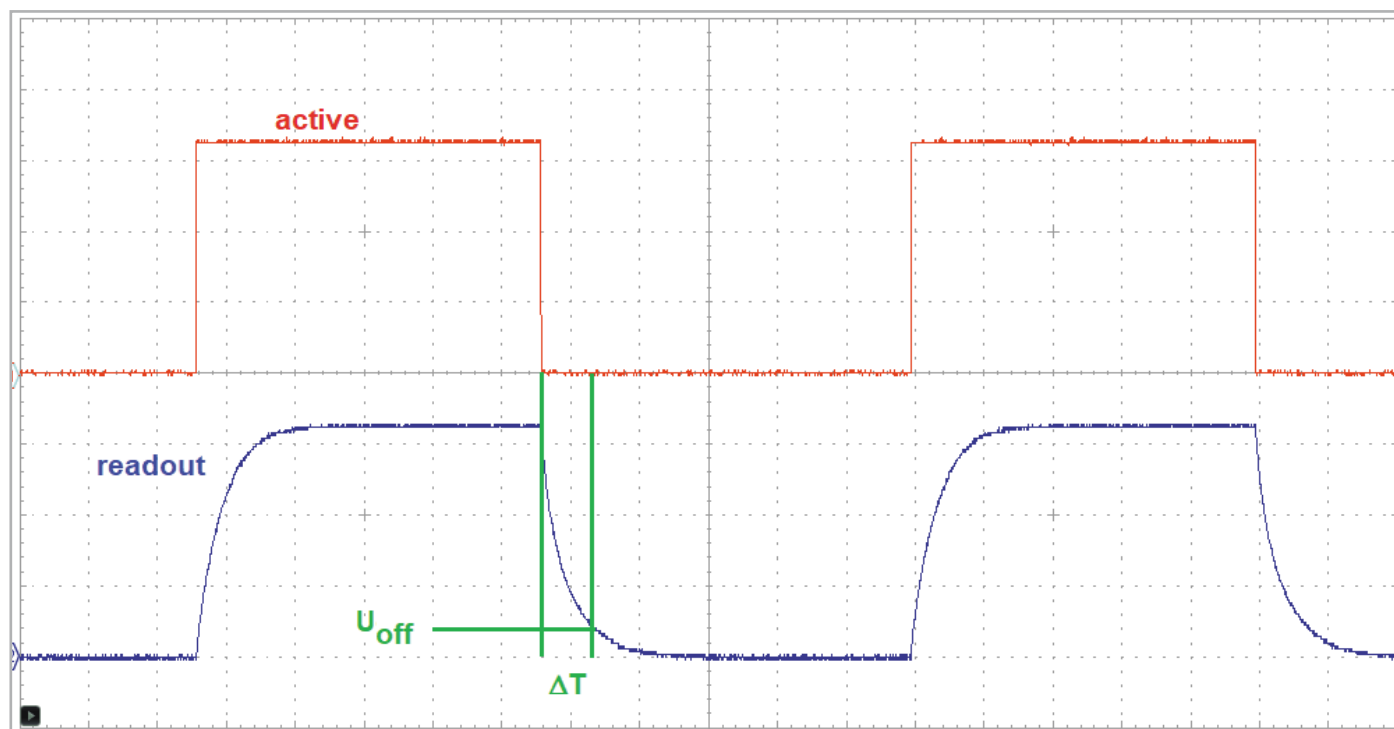


Bild 3: Signalverlauf an den Ports

höher liegen. Vorhandene Unterschiede kann man nun mit dem `cal`-Parameter korrigieren. Gegebenenfalls muss nun auch der Offset nochmals nachgestellt werden. Dieses Verfahren wird als iterative Kalibrierung bezeichnet.

Mathematisch versierte Anwender können `cal` und `offset` auch berechnen. Hierfür werden zwei feste Temperaturniveaus benötigt, z. B. Eiswasser (0 °C) und kochendes Wasser (100 °C). Dann werden die Anzeigewerte des NTC-Thermometers bei beiden Werten gemessen. Nun kann man zwei Gleichungen mit zwei Unbekannten aufstellen, deren Lösung die Werte `cal` und `offset` liefern.

Präzise Erfassung analoger Messwerte

Wie in den letzten Abschnitten gezeigt wurde, kann man zwar mit einigen Tricks auch mit GPIO-Ports direkt analoge Werte erfassen. Allerdings ist das meist nur eine Notlösung. Für präzise Messungen sind andere Verfahren erforderlich.

Will man analoge Spannungswerte zuverlässig und präzise messen, ist der Einsatz eines Analog-Digital-Converters (ADC) das Mittel der Wahl. ADCs werden über spezielle Bus-Systeme mit einem Prozessor oder Controller verbunden. Auch beim Raspberry Pi verfügen einige der GPIO-Pins über eine alternative Funktionalität. Diese ermöglicht es, z. B. über einen I2C-Bus oder über SPI mit externen Komponenten zu kommunizieren.

Viele ADCs werden über einen SPI-Bus gesteuert. Deshalb soll diese Variante im folgenden Abschnitt näher betrachtet werden.

Der ADC am SPI-Bus

Die Abkürzung SPI steht für Serial Peripheral Interface. Damit wird ein ursprünglich von der Firma Motorola entwickeltes Bus-System bezeichnet, das es erlaubt, Daten zwischen integrierten Schaltkreisen

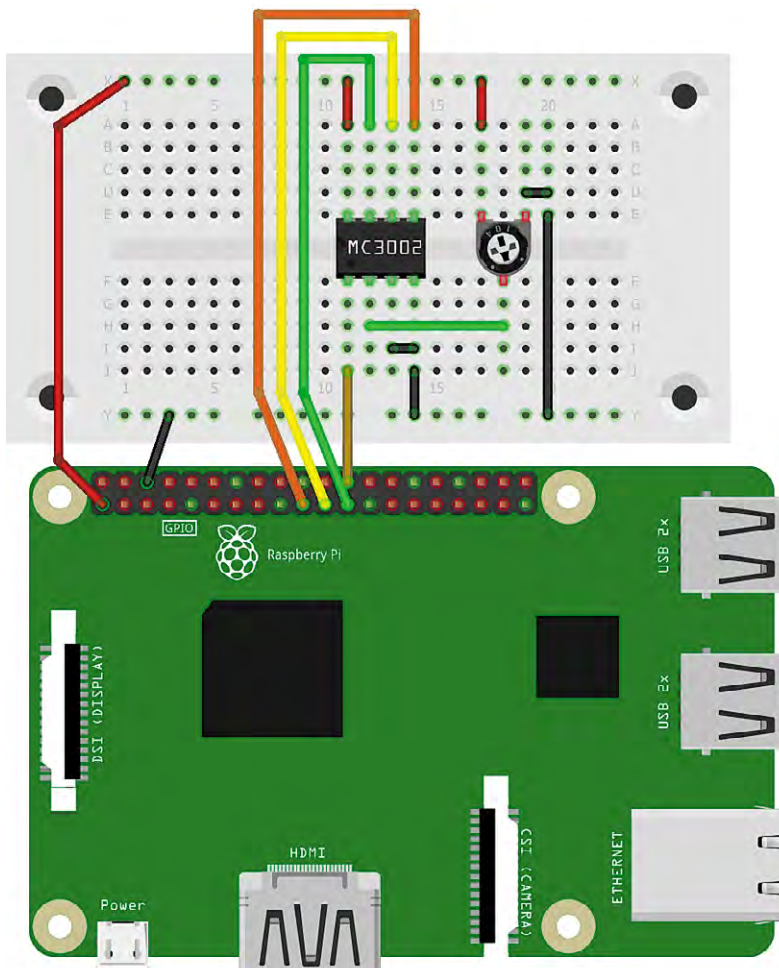


Bild 5: Schaltbild zum MCP3002 am Raspberry Pi

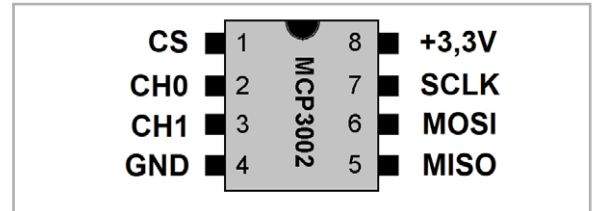


Bild 4: Der AD-Wandler MCP3002 verfügt über ein SPI-Interface.

auszutauschen. Ein Entwicklungsziel des Bus-Systems war, mit möglichst wenigen Leitungen auszukommen. Beim SPI-Bus werden die einzelnen Bausteine über ein Master-Slave-Prinzip miteinander verbunden.

Der Bus verfügt über vier Leitungen, die vom Master zum Slave oder auch zu mehreren Slaves führen. Der bekannte und preisgünstige Analog-Digital-Konverter DC MCP3002 wird über die SPI-Schnittstelle mit dem Raspberry Pi verbunden. Dieser integrierte Baustein verfügt über zwei unabhängig voneinander arbeitende Eingänge, CH0 und CH1. Die Auflösung beträgt zehn Bit. Die gesamte Funktionalität des Bausteins wird über die vier Leitungen des SPI-Busses gesteuert.

Für die Kommunikation auf dem SPI-Bus müssen Daten in beide Richtungen fließen. Also sowohl vom Master zum Slave als auch umgekehrt. Dieser Datenaustausch erfolgt über zwei getrennte Leitungen:

- MOSI (Master-Out-Slave-In)
- MISO (Master-In-Slave-Out)

Für jede Datenrichtung wird jeweils eine Leitung verwendet. Zusätzlich wird noch eine SCLK-Leitung (Serial-Clock) verwendet, die den Bustakt liefert.

Die Auswahl der am Bus angeschlossenen Komponenten (Slaves) erfolgt über die CS-Leitung (Chip-Select).

Bild 4 zeigt einen AD-Wandler-Baustein mit SPI-Interface. Der IC verfügt über die folgenden Pins:

MC 3002	Funktion	Verbinden mit RasPi Pin
Pin 1	Chip-Select, LOW-Aktiv	GPIO 10 - CE0
Pin 2	CH0: Analog-Kanal 0	-
Pin 3	CH1: Analog-Kanal 1	-
Pin 4	Vss (GND)	GND
Pin 5	MOSI (Din)	GPIO 12 - MOSI
Pin 6	MISO (Dout)	GPIO 13 - MISO
Pin 7	CLK (Serial Clock)	GPIO 14 - SCLK
Pin 8	VDD/VREF Spannungsversorgung 3,3 V	3V3

Da der MCP3002 in einem 8-poligen DIL-Gehäuse untergebracht ist, kann er leicht in ein Breadboard eingebaut werden (s. Bild 5).

Der ADC-Wandler MCP3002

Aufgrund seines kompakten Gehäuses und des SPI-Interfaces kann der MC3002 sehr einfach mit dem Raspberry Pi verbunden werden. Bild 5 und Bild 6 zeigen das Schaltbild und den Aufbau dazu.

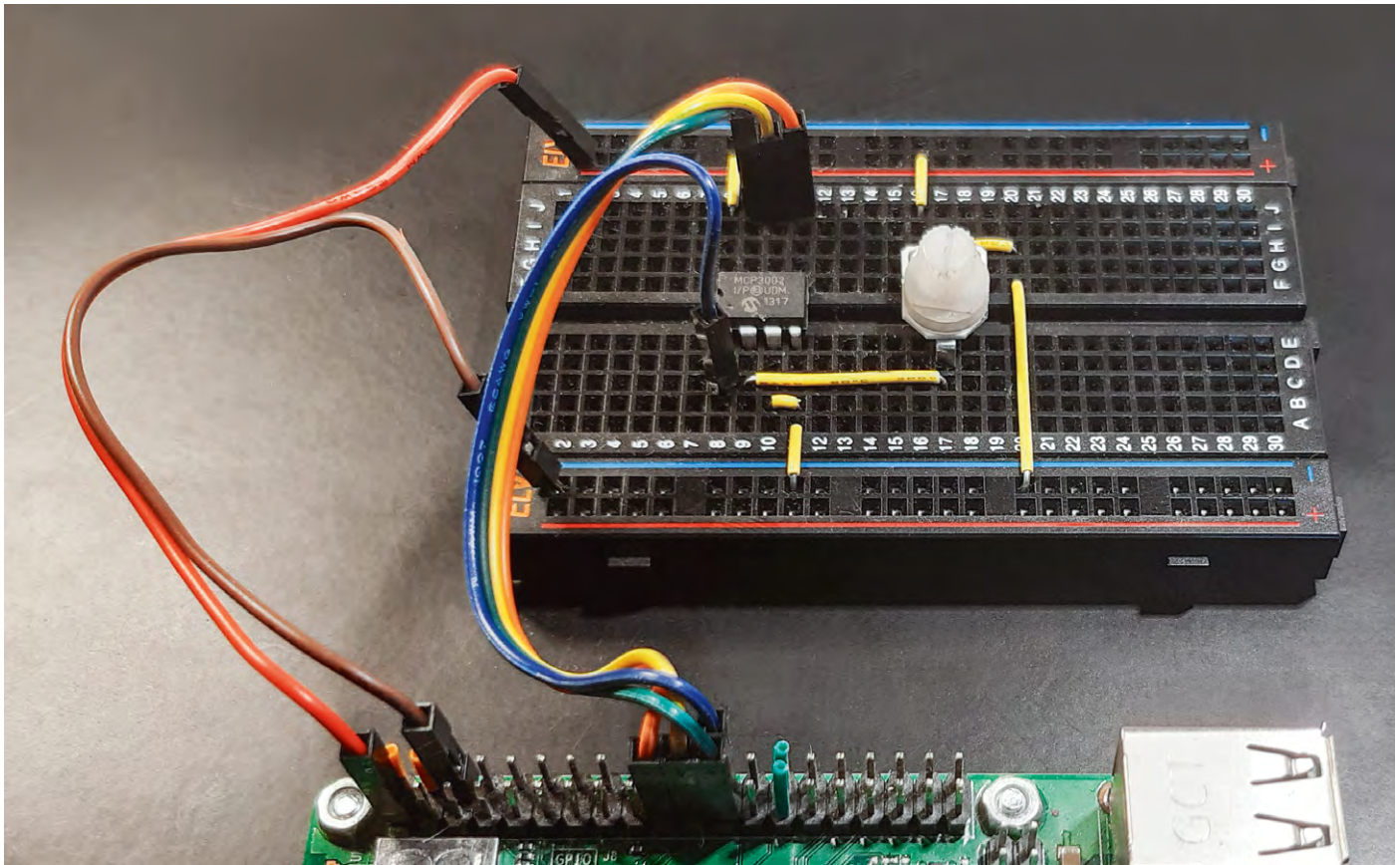


Bild 6: Aufbau zur Erfassung von Analogwerten mit dem MCP3002

Die Kommunikation mit dem Chip findet über die beiden Leitungen MISO (Dout) und MSO (Din) statt. Die analogen Eingänge liegen auf Pin 2 und Pin 3 und führen die Bezeichnung CH0 und CH1.

Der Pin VDD/VREF wird mit 3,3V versorgt, sodass Eingangsspannungen von 0V bis 3,3V direkt gemessen werden können. Für höhere Spannungspegel sind geeignete Spannungsteiler vorzuschalten (s. u.).

Die Auflösung des MCP3002 beträgt 10 Bit. Damit können

$$2^{10} = 1024$$

verschiedene Spannungswerte erfasst werden. Der Wertebereich der Digitalausgabe beträgt also

$$0 \dots 2^{10} - 1 = 0 \dots 1023$$

Bei einer Versorgungs- bzw. Referenzspannung von 3,3V sind somit prinzipiell Spannungsänderungen von 3,22 mV messbar.

Inbetriebnahme des MCP3002

Um den MCP3002 in Betrieb nehmen zu können, muss zuerst ein SPI-Treiber importiert werden, damit die Kommunikation über den SPI-Bus ermöglicht wird.

Für die Installation von spidev (SPI devices) sind die folgenden Schritte im Terminal erforderlich:

```
sudo apt-get install python-setuptools
sudo apt-get install python-dev
sudo apt-get install git git clone git://github.com/doceme/py-spidev
cd py-spidev/
sudo python setup.py install
```

Danach muss zudem die config.txt Datei aktualisiert werden:

```
sudo nano /boot/config.txt
```

Am Ende der Datei ist die Zeile

```
dtparam=spi=on
```

einzuführen.

Damit ist der Raspberry Pi bereit, über den SPI-Bus zu kommunizieren. Das Python-Programm zur Ansteuerung des MCP3002 sieht wie folgt aus (MCP3002_ADC.py):

```
import time, spidev

spi = spidev.SpiDev(0, 0) # device 0, channel 0
spi.max_speed_hz = 1200000

def read_adc(adc_ch, vref = 3.3):
    msg = 0b11
    reply = spi.xfer2([(msg << 1) + adc_ch << 5, 0b0000000])
    adc = 0
    for n in reply:
        adc = (adc << 8) + n
    adc = adc >> 1
    voltage = (vref * adc) / 1024
    return voltage

print("MCP3002")
print("=====\n")

while True:
    print("Ch 0:", round(read_adc(0), 2), "V \t Ch 1:",
          round(read_adc(1), 2), "V")
    time.sleep(0.2)
```

Das Python-Programm liest die analogen Spannungswerte beider Kanäle des MCP3002 über die SPI-Schnittstelle aus und zeigt diese auf der Konsole an. Dabei wird die Python-Bibliothek spidev verwendet, um mit dem ADC über SPI zu kommunizieren. Die notwendigen Bibliotheken werden in der ersten Programmzeile importiert:

`time` für die Zeitsteuerung
und

`spidev` für die SPI-Kommunikation.

Danach wird die SPI-Schnittstelle konfiguriert. Mit

```
spi = spidev.SpiDev(0, 0)
```

wird ein Default-Wert für den SPI-Kanal (0) auf dem ersten Baustein (0) ausgewählt und eine neue SPI-Verbindung mit dem SPI-Bus „0“ erstellt. Die maximale Übertragungsgeschwindigkeit für die SPI-Kommunikation wird über

```
spi.max_speed_hz = 1200000
```

auf 1,2 MHz festgelegt.

Die Funktion

```
read_adc(adc_ch, vref=3.3)
```

liest den ADC aus und gibt die gemessenen Spannungswerte zurück.

```
adc_ch:
```

Der Kanal des ADC, von dem gelesen werden soll (0 oder 1).

```
vref:
```

Die Referenzspannung des ADC, standardmäßig auf 3,3 V gesetzt.

Der ADC wird über SPI mit einem 2-Bit-Nachrichtentyp (Command) 0b11 konfiguriert, gefolgt vom gewünschten Kanal (adc_ch).

Innerhalb der Funktion werden die ersten vier Bits der SPI-Nachricht „msg“ konfiguriert:

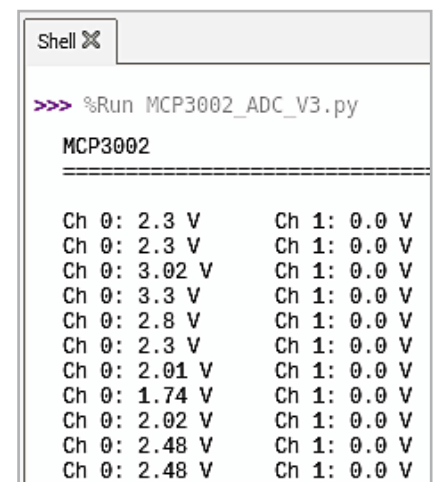
- Das erste Bit (Start) ist immer HIGH, um den Start der Nachricht anzugeben.
- Das zweite Bit (SGL/DIFF) wird ebenfalls auf 1 gesetzt, um den Einzelmodus (Single Mode) auszuwählen. Das bedeutet, dass jeweils nur ein einzelner Kanal gemessen wird.
- Das dritte Bit (ODD/SIGN) wird verwendet, um den Kanal auszuwählen, der gelesen werden soll. Es kann entweder 0 oder 1 sein, je nachdem welcher Kanal ausgewählt werden soll.

- Das vierte Bit (MSFB) wird auf 0 gesetzt, um anzugeben, dass die Übertragung mit dem Least Significant Bit (LSB) zuerst erfolgt, was auch als Little-Endian bezeichnet wird. Diese Reihenfolge wird für die meisten Anwendungen verwendet. Die gelesenen Daten werden in einem 10-Bit-Wert von 0 bis 1023 empfangen, und die Funktion konvertiert diesen Wert in einen Spannungswert mithilfe der angegebenen Referenzspannung (vref).

In der Hauptschleife werden die Spannungswerte auf zwei Dezimalstellen gerundet und zusammen mit den Kanalbezeichnungen „Ch 0:“ und „Ch 1:“ ausgegeben.

Die Anweisung `time.sleep(0.2)` sorgt für eine Verzögerung von 0,2 Sekunden zwischen den Messungen. Das Programm gibt damit alle 0,2 Sekunden die aktuellen Spannungen von Kanal 0 und Kanal 1 des ADC aus. Nach dem Start des Programms können die Werte in der Konsole ausgegeben werden (Bild 7).

Ch 0 zeigt die aktuellen Werte der Potentiometer-Spannungen. Ch 1 zeigt 0 Volt an, da dieser Kanal zunächst auf GND (0 V) gelegt wurde (s. Bild 5). Bei praktischen Anwendungen können natürlich beide Kanäle simultan verwendet werden.



```
Shell X
>>> %Run MCP3002_ADC_V3.py
MCP3002
-----
Ch 0: 2.3 V      Ch 1: 0.0 V
Ch 0: 2.3 V      Ch 1: 0.0 V
Ch 0: 3.02 V     Ch 1: 0.0 V
Ch 0: 3.3 V      Ch 1: 0.0 V
Ch 0: 2.8 V      Ch 1: 0.0 V
Ch 0: 2.3 V      Ch 1: 0.0 V
Ch 0: 2.01 V     Ch 1: 0.0 V
Ch 0: 1.74 V     Ch 1: 0.0 V
Ch 0: 2.02 V     Ch 1: 0.0 V
Ch 0: 2.48 V     Ch 1: 0.0 V
Ch 0: 2.48 V     Ch 1: 0.0 V
```

Bild 7: Ausgabe der ADC-Werte in der Konsole

Raspberry Pi als Transientenrekorder: Grafische Datenausgabe

Die Daten des ADCs können auch im Plotter der Thonny IDE ausgegeben werden. Dazu muss man die Ausgabezeile

```
print("Ch 0:", round(read_adc(0), 2), "V Ch 1:", round(read_adc(1), 2), "V")
```

auf rein numerische Werte umstellen:

```
print(round(read_adc(0), 2), round(read_adc(1)))
```

Dann hat man bereits ein grafisches Datenerfassungssystem vor sich, das in vielen Anwendungsfällen nutzbringend eingesetzt werden kann. Bild 8 zeigt die grafische Ausgabe der beiden Kanäle. Bild 9 zeigt die gleichen Signale auf einem digitalen Oszilloskop.

Man erkennt, dass die Werte des Transientenrekorders gut mit den Ausgabedaten des digitalen Speicheroszilloskops (DSO) übereinstimmen. Lediglich die Flanken sind in der Thonny-Grafik nicht ganz so steil, da hier nur mit fünf Werten pro Sekunde abgetastet wird.

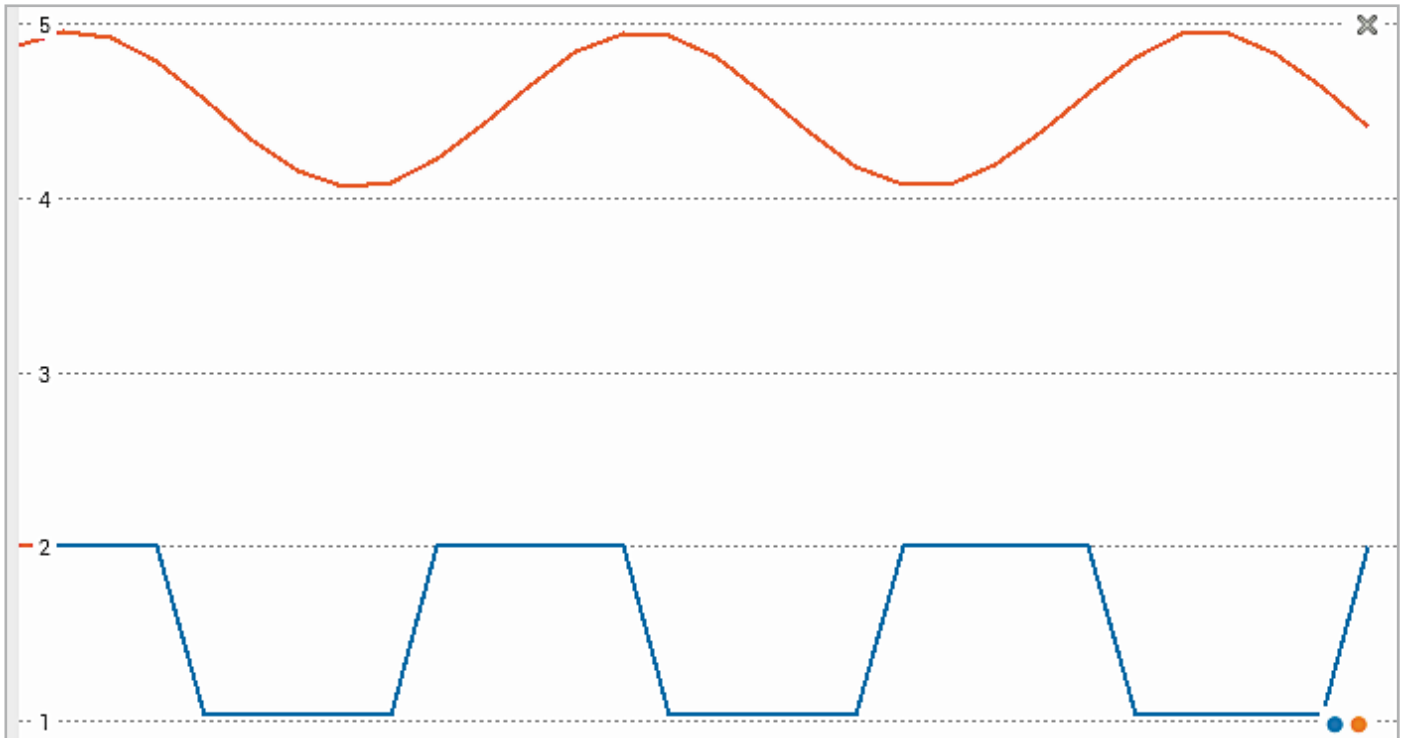


Bild 8: Grafische Ausgabe der beiden ADC-Kanäle im Thonny Plotter

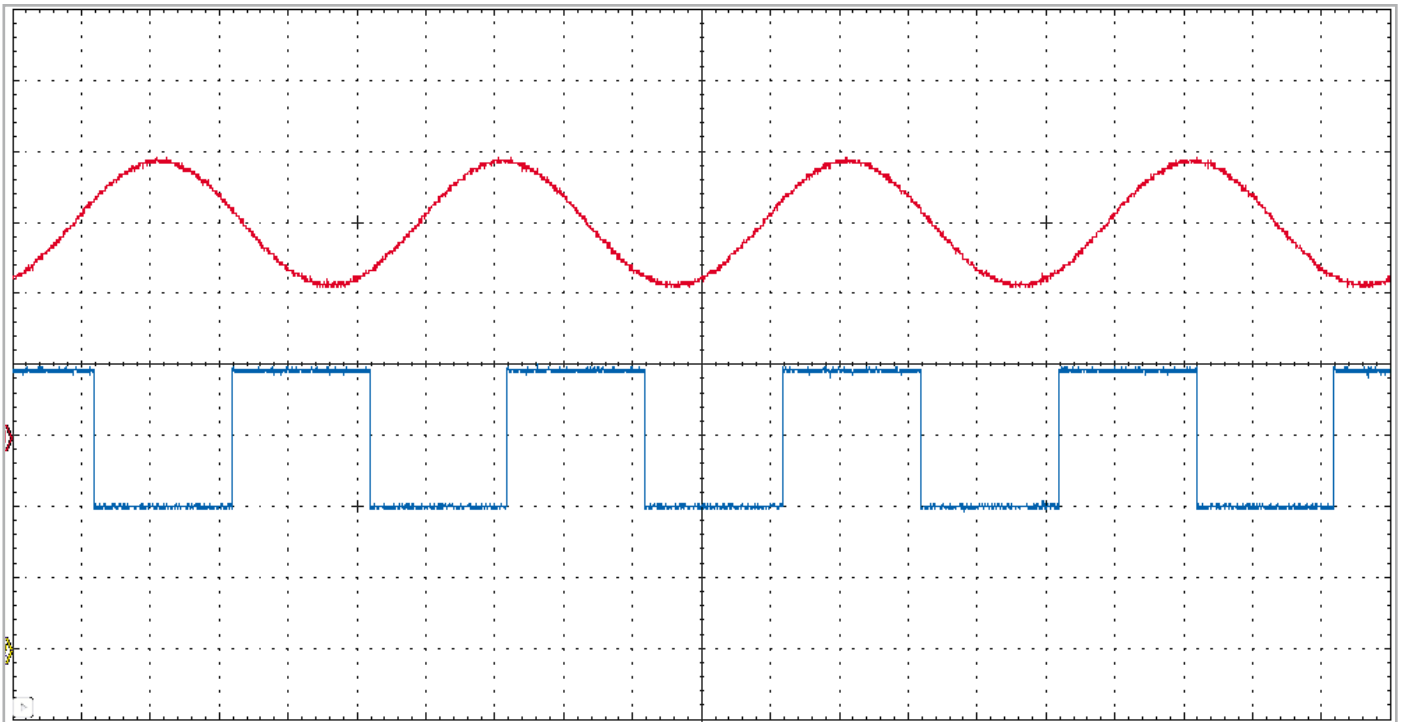


Bild 9: Die Werte aus Bild 8 auf einem Digitaloszilloskop

Darf in keinem Labor fehlen: Präzises Computer-Digitalvoltmeter mit Python-Steuerung

Mit dem MCP3002 ist es nun möglich, Spannungen sehr präzise zu messen. Allerdings darf die Eingangsspannung an den Analogpins CH0 und CH1 keinesfalls die Versorgungsspannung des A/D-Wandlers von 3,3 V überschreiten. Möchte man größere Spannungen messen, muss man einen Spannungsteiler vorschalten. Verwendet man einen Teiler aus einem 100-kOhm- und einem 10-kOhm-Widerstand, so erhält man eine Spannungsreduktion um den Faktor

$$F = 10 \text{ kOhm} / (100 \text{ kOhm} + 10 \text{ kOhm}) = 1/11$$

Die Referenzspannung von 3,3 V liefert als Eingangspegel den Wert von $2^{10} - 1 = 1023$

Als Kalibrationskonstante erhält man so

$$\text{cal} = 11 \times 3,3 / (2^{10} - 1) = 11 \times 3,3 / 1023 = 0,03548\dots$$

Mit diesen Konstanten kann der ADC-Wert im Python-Programm direkt in eine Spannung umgerechnet werden. Die Schaltung dazu ist in [Bild 10](#) zu sehen.

Damit ist es nun möglich, externe Spannungen mit bis zu 35 V zu messen. Vorsichtshalber sollte man aber einen gewissen Sicherheitsabstand zur zulässigen Maximalspannung einhalten. Bis 30 Volt kann man aber mit dem Computer-Voltmeter problemlos und sehr genau messen.

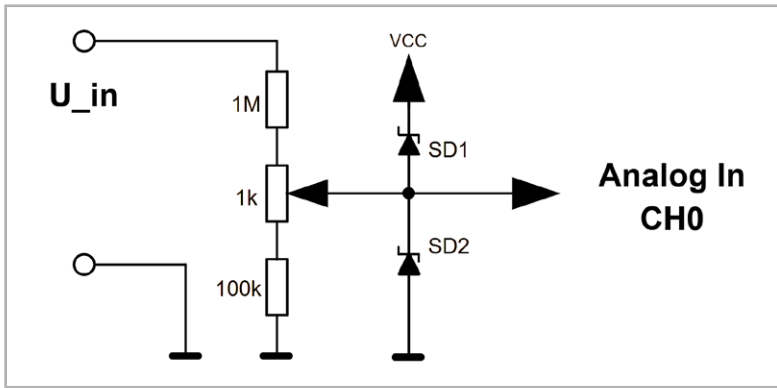


Bild 10: Computer-Voltmeter bis max. 30 V

Die zusätzlichen Schottky-Dioden dienen dazu, den A/D-Wandler vor Verpolung bzw. Überlastung zu schützen. Wird eine Spannung falsch herum an den Messeingang angelegt, schließt die Diode SD2 den Eingang des MCP3002 nahezu kurz. Es können dann maximal ca. $-0,3\text{ V}$ am Eingang des ICs anliegen, sodass dieses keinen Schaden nehmen kann. Die Diode SD2 schützt auf ähnliche Weise vor zu hohen Spannungen ($> V_{cc} + 0,3\text{ V}$). Nun muss nur noch das Programm entsprechend angepasst werden:

```
# MCP3002_voltmeter.py

from Tkinter import *
import spidev
import time

# SPI setup
spi_max_speed = 1000000    # 1 MHz for SPI
CE = 0                    # select SPI device

spi = spidev.SpiDev()
spi.open(0,CE)

root=Tk()
root.font=('Arial', 30, 'normal')
root.title("Voltmeter")
root.geometry('280x60') #window size

cal = 11*3.3/1023
offset = 0

def read_mcp3002(lb):
    cmd = 0b01100000 # CH0
    # cmd = 0b01110000 # CH1
    spi_data = spi.xfer2([cmd,0]) # start sp communication
    adc_data = ((spi_data[0] & 3) << 8) + spi_data[1]
    Voltage = 0.01*int((100*adc_data-offset)*cal)
    lb.config(text= str(Voltage) + " V")
    # time.sleep(1)
    lb.after(100, read_mcp3002, lb)
    # return adc_data

lb = Label(root)
lb.config(text="", font = root.font)
lb.pack()

read_mcp3002(lb)
root.mainloop()
```

Hardware- und Softwarekalibrierung

Nach der Inbetriebnahme des Voltmeters sollte am Bildschirm ein bestimmter Spannungswert angezeigt werden. Durch Drehen am Potentiometer lässt sich der Wert in gewissen Grenzen verändern. In einem ersten Schritt wird das Potentiometer so eingestellt, dass die aktuelle Spannung korrekt angezeigt wird. Dieser Abgleich wird als Hardware-Kalibration bezeichnet.

Nun kann man das Voltmeter bei anderen Spannungen testen. Falls es zu Messabweichungen kommt, kann man die Parameter `cal` und `offset` so verändern, dass alle Messwerte mit nur noch minimalen Fehlern angezeigt werden. Dieses Vorgehen ist auch als iterative Methode bekannt.

Durch die Anpassung der beiden Parameter `cal` und `offset` wird eine sogenannte Softwarekalibration durchgeführt. Wer mathematisch interessiert ist, kann die Parameter wieder aus verschiedenen Messpunkten mittels spezieller Verfahren wie etwa der gaußschen Fehlerquadratmethode berechnen. Alternativ kann man auch eine sogenannte Ausgleichsgerade durch die Messpunkte legen und dann die beiden Parameter ablesen.

Fazit und Ausblick

In diesem Beitrag wurde die Erfassung analoger Messwerte mit Python ausführlich dargestellt. Neben einer einfachen Ladezeit-Messung kam dabei auch ein SPI-gesteuerter ADC zum Einsatz. Die Werte konnten in Thonny sowohl als numerische Tabelle oder auch als einfache Zeit/Messwert-Grafiken ausgegeben werden.

Diese einfache Form der grafischen Ausgabe ist in vielen Fällen ausreichend. Allerdings ist Python auch in der Lage, komplexere Grafiken darzustellen. Damit wird es möglich, z. B. Messwerte in den verschiedensten Varianten aufzutragen. Neben einfachen x/y-Grafiken können dann auch Balken- oder Streudiagramme erstellt werden.

Auch komplexe Signalanalysemethoden sind in Python vergleichsweise leicht programmierbar. Die umfangreiche und weit verbreitete Matplotlib-Bibliothek stellt dazu eine Vielzahl von Methoden zur Verfügung. Sie soll daher im nächsten Artikel detailliert betrachtet werden. **ELV**

Material

z. B. Raspberry Pi 4 Model B,	
8 GB RAM	Artikel-Nr. 250567
Breadboard	Artikel-Nr. 251467
Kondensator und NTC sind im	
Set PAD-PRO-EXSB enthalten	Artikel-Nr. 158980
A/D-Konverter, z. B. MCP3002	

Zum Download-Paket