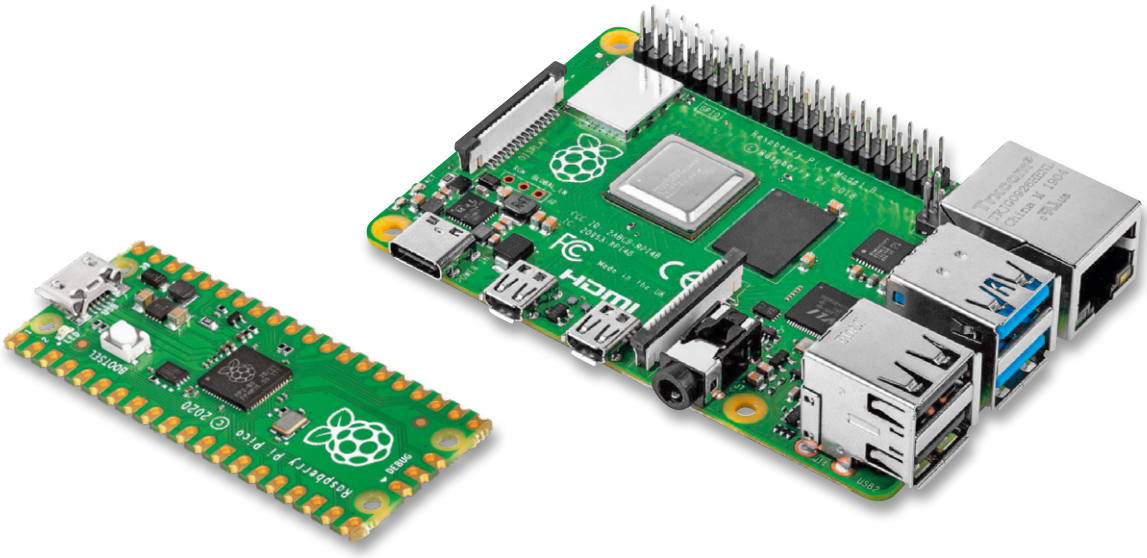
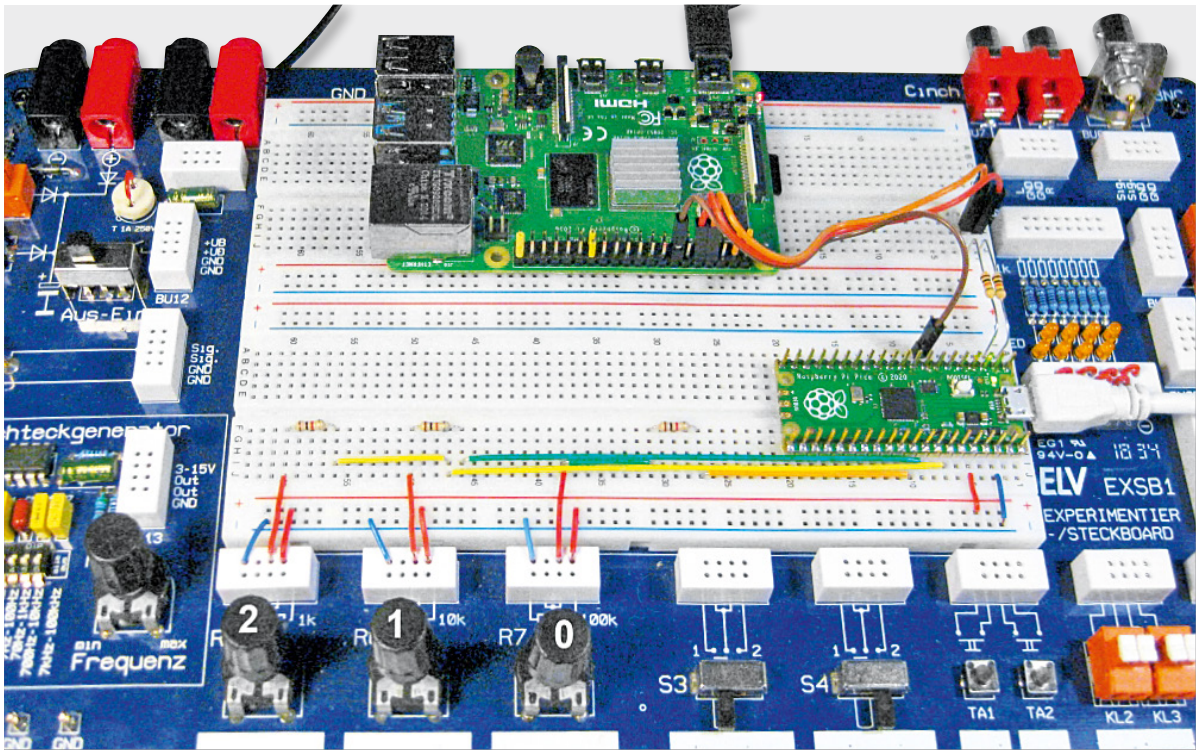
# Raspberry Pi Pico

## Perfektes Duo mit dem Raspberry Pi

Teil 2

**Der Raspberry Pi dominiert seit vielen Jahren die Maker-Szene. Insbesondere die frei verfügbaren I/O-Pins des Mini-Rechners machten ihn zu einem der beliebtesten Boards aller Zeiten. Allerdings weist der Raspberry Pi auch zwei wesentliche Nachteile auf. Zum einen sind die Pins direkt, d. h. ohne Treiber oder Schutzschaltung, mit den frei liegenden Kontakten verbunden. Dadurch kann der Controller und damit der gesamte Raspberry Pi schnell zerstört werden. Bereits eine kurzzeitige fehlerhafte Kontaktierung mit 5 Volt kann das Ende des Controllerboards bedeuten.**





## Ideales Duo

Zum anderen verfügt der klassische Raspberry Pi über keinerlei Analogeingänge. Eine direkte Messung analoger Werte ist somit nicht möglich. Es können weder Photodioden noch NTCs, Hallsensoren oder ähnliche Bauteile unmittelbar ausgelesen werden.

Mit dem Raspberry Pi Pico [1] kann dieses Problem elegant gelöst werden. Er kann als sogenanntes Frontend problemlos vielfältige Messaufgaben übernehmen. Zudem ist der Pico deutlich preisgünstiger als etwa ein Raspberry Pi 4 mit 8 Gigabyte RAM. Falls also eine Fehlbeschaltung zur Zerstörung des Pico führt, ist dies vor allem für nicht professionelle Anwender wesentlich leichter zu verkraften.

Der klassische Raspberry Pi und der Pico werden so zum idealen Duo, mit dem auch anspruchsvolle Messaufgaben gelöst werden können.

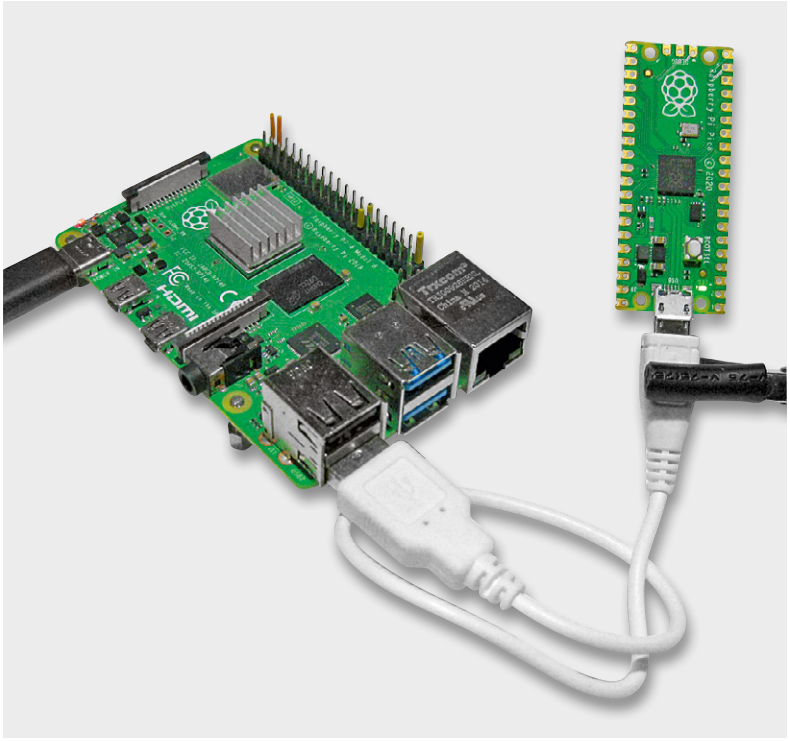


*Bild 1: Verbindung von Pico und Raspberry Pi via USB-Kabel*

## Raspberry Pi classic programmiert Pico

Bevor der Pico zur Messwerterfassung eingesetzt werden kann, muss er natürlich wie immer entsprechend programmiert werden. Üblicherweise wird für diese Aufgabe ein PC oder Laptop unter Windows eingesetzt. Allerdings kann der klassische Raspberry Pi die Aufgabe des Host-Rechners genauso gut übernehmen.

Die Entwicklungsumgebung Thonny [2] ist auf jedem neueren Raspberry Pi OS bereits vorinstalliert. Eine spezielle Installation wie unter Windows kann man sich daher in den meisten Fällen sparen. Hardwareseitig muss der Pico lediglich mit einem Micro-USB-Kabel an den Raspberry Pi angeschlossen werden (Bild 1).

Auf dem Raspberry Pi kann dann Thonny gestartet werden. Über

Run → Select Interpreter

wird nun „MicroPython (Raspberry Pi Pico)“ ausgewählt (Bild 2).

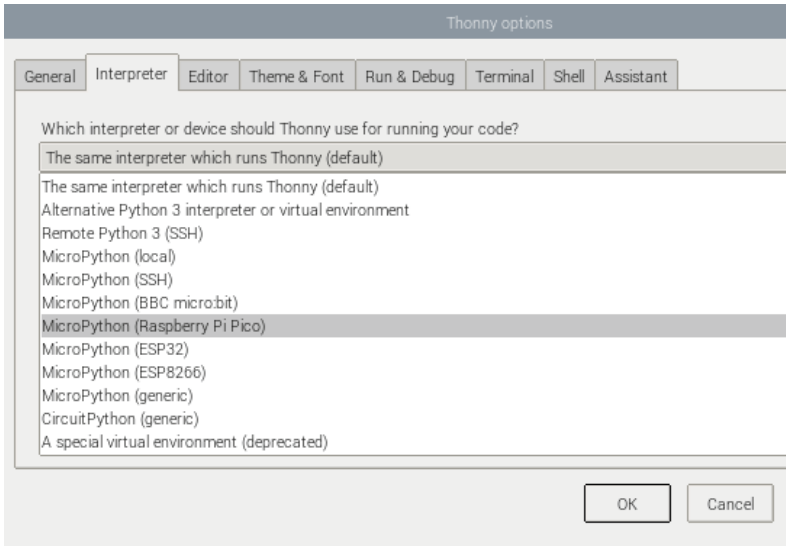



*Bild 2: Auswahl des Pico-Interpreters*

Der angeschlossene Pico kann nun über

/dev/ttyACM0

angesprochen werden (Bild 3).

Um die Verbindung zu testen, kann nun ein einfaches Python-Programm (z. B. LED_toggler.py – alle folgenden Programme sind unter [3] erhältlich) auf den Pico geschrieben werden.

Sobald das Programm auf den Pico übertragen wurde, blinkt die grüne On-Board-LED in schneller Folge. Damit ist das „perfekte Duo“ einsatzbereit.

Falls das MicroPython-System noch nicht auf dem Pico installiert ist, kann die Installation über „Install or update firmware“ (s. unten rechts in Bild 3) nachgeholt werden. Weitere Details dazu finden sich im ersten Beitrag zu dieser Serie [4].

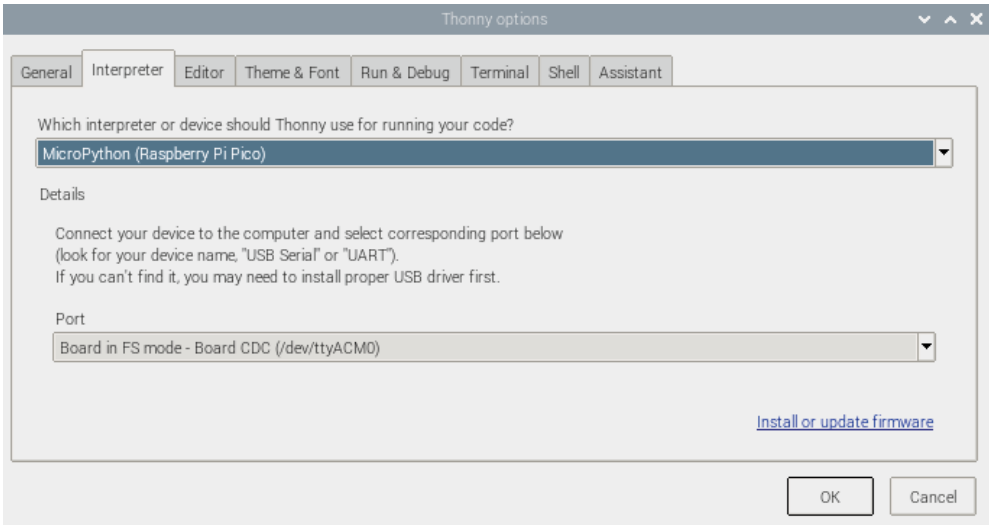



*Bild 3: Der Pico ist über /dev/ttyACM0 ansprechbar.*

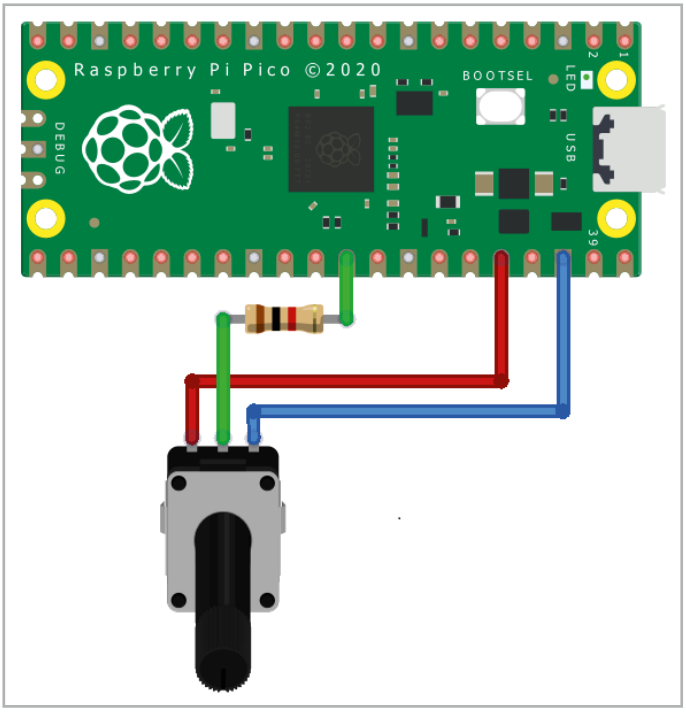



Bild 4: Potentiometer am Pico

## Python-Voltmeter

Der Pico kann nun als Frontend für den klassischen Raspberry Pi eingesetzt werden. In einer ersten Anwendung wird dazu ein Programm auf den Pico geladen, das die Spannung an einem ADC-Eingang erfasst und via USB zum Raspberry Pi überträgt (Voltmeter_Pico.py [3]):

```
from machine import ADC, Pin
import time

conversion_factor=3.3/(1<<16) #ADC scaled to 16 bit
adc=ADC(Pin(26))

while True:
    print(adc.read_u16()*conversion_factor)
    time.sleep(1)
```

Um eine variable Spannung am Analogeingang des Pico zur Verfügung zu stellen, kann man beispielsweise ein Potenziometer anschließen (Bild 4). Hierfür kann auch das Experimentierboard EXSB1 [5] eingesetzt werden. Bild 5 zeigt einen entsprechenden Aufbauvorschlag.

Sobald das Programm „Voltmeter_Pico.py" gestartet wird, erscheinen die Spannungswerte in der Thonny-Ausgabe. Wird zudem der Plotter aktiviert, steht nun bereits ein sehr brauchbares Messwerterfassungssystem mit grafischer Ausgabe zur Verfügung (Bild 6).

## Flexible Messwerterfassung

Die Erfassung und Darstellung von Messwerten über die Thonny-Shell ist durchaus bereits für viele Messaufgaben geeignet. Allerdings ist die einfache Ausgabe von Zahlenreihen oder auch eine simple y-t-Darstellung für viele Anwendungen zu unflexibel. Mithilfe von Python kann man die Messwertdarstellung wesentlich ansprechender gestalten. Dazu muss man lediglich Thonny wieder auf den lokalen Interpreter zurücksetzen (Bild 2: „The same interpreter which runs Thonny (default)"). Dann kann ein entsprechendes Messprogramm gestartet werden (Voltmeter_RasPi.py [3]):

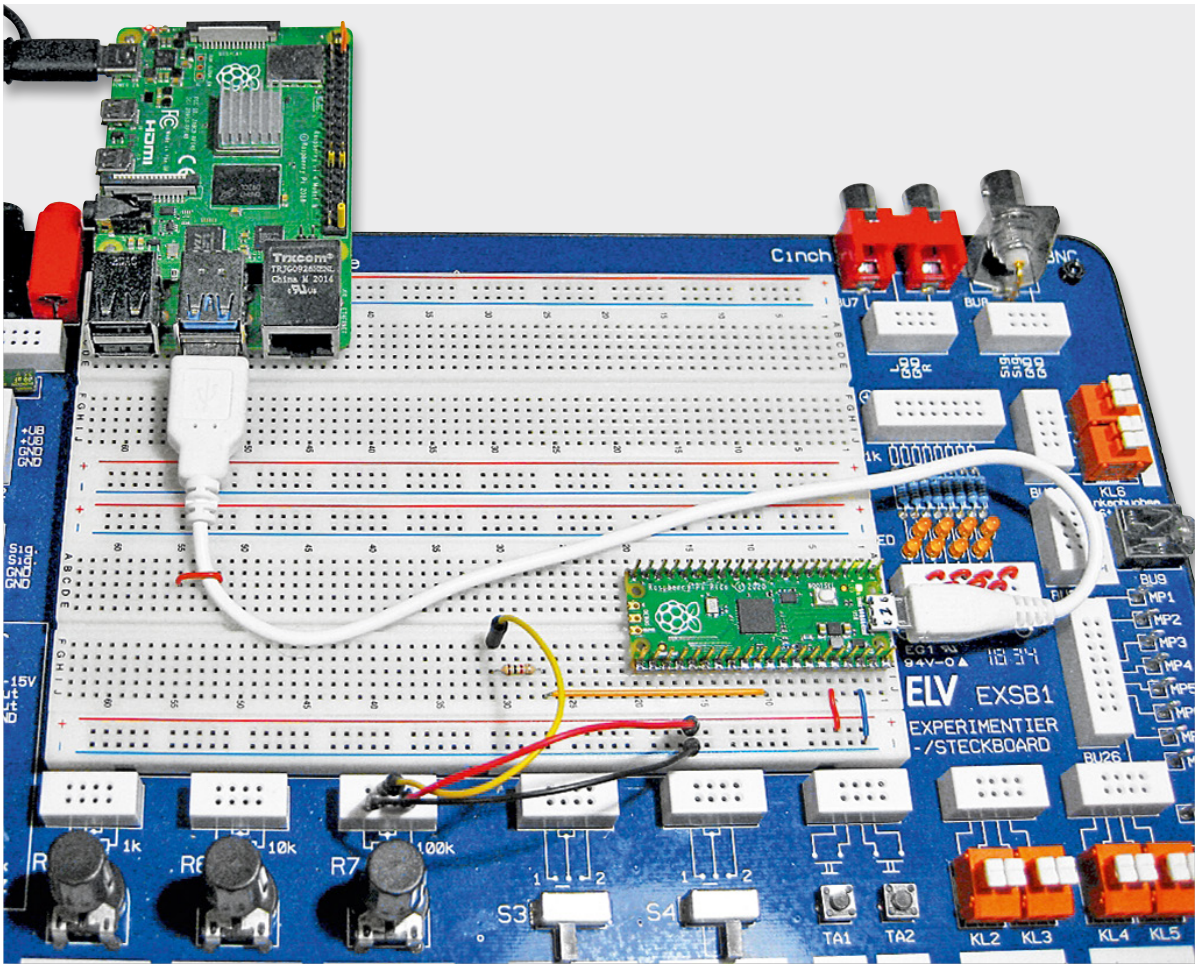



Bild 5: Spannungsmessung mit dem EXSB1

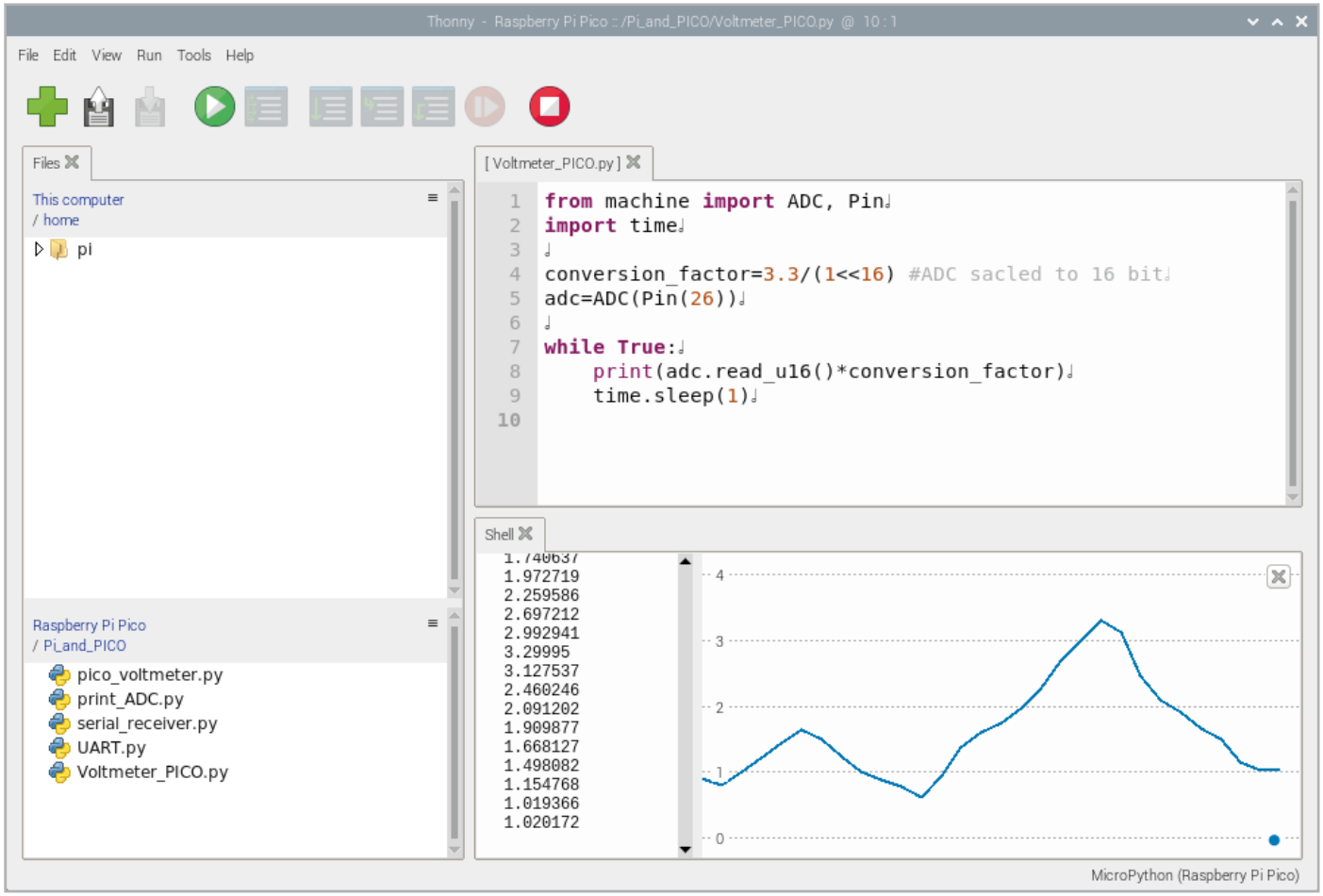



*Bild 6: Messwerterfassung mit dem „perfekten Duo"*

```
from tkinter import *
import serial, time

root=Tk()
root.font=('Arial', 30, 'normal')
root.title("Voltmeter")
root.geometry('250x60+100+100') #window size & position

ser=serial.Serial('/dev/ttyACM0',baudrate=115200,parity=serial.PARITY_NONE,stopbits=serial.STOPBITS_ONE)

def getValue(lb):
    received_data = ser.read()              #read serial port
    time.sleep(0.03)
    data_left=ser.inWaiting()               #check for remaining byte
    received_data+=ser.read(data_left)
 strVal=str("%.2f"%(float(received_data[0:4])))
 print(strVal)
 lb.config(text="U = "+strVal+" V")
 time.sleep(.1)
 lb.after(100, getValue , lb)

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

getValue(lb)
root.mainloop()
```

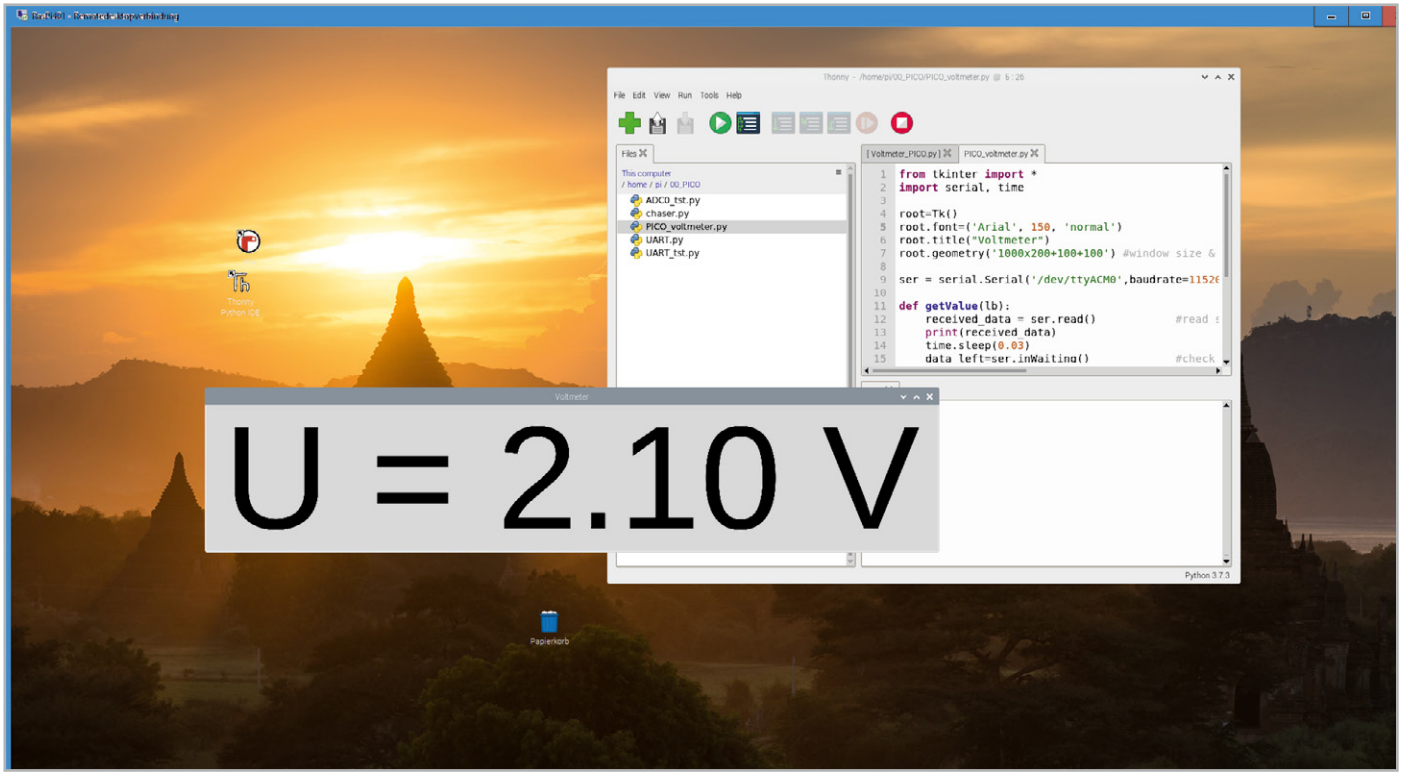



*Bild 7: Großflächige Datenausgabe von ADC-Werten*

Nun erscheinen die aktuellen Spannungswerte in einem eigenen, frei programmierbaren Fenster (Bild 7). Größe, Form und farbliche Gestaltung sind jetzt frei wählbar. So können etwa großflächige Anzeigen, die auch aus größerer Entfernung noch gut ablesbar sind, erstellt werden.

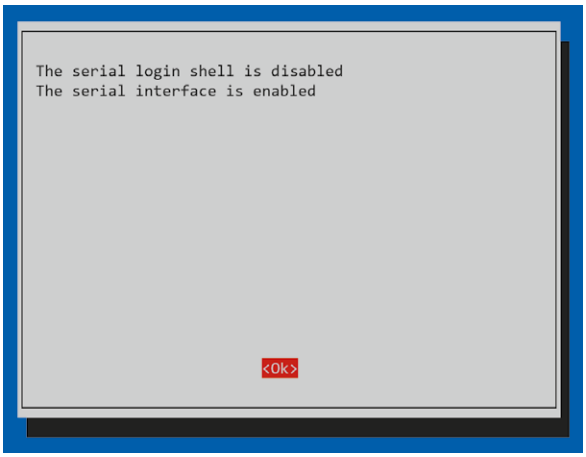



*Bild 8: Freigabe der seriellen Schnittstelle auf dem Raspberry Pi*

## Serielle Verbindung zwischen Raspberry Pi und Pico

Bislang wurden lediglich Messdaten vom Pico zum Raspberry Pi übertragen. Diese Erfassung von analogen Messwerten wäre aber auch z. B. mit einem einfachen externen A/D-Wandler-Baustein möglich gewesen. Als vollwertiges Mikrocontrollersystem kann der Pico jedoch noch wesentlich mehr leisten.

Über die serielle Schnittstelle ist eine vollständige Steuerung des Controllers möglich. Dabei fließen die Daten nicht nur vom Pico zum Raspberry Pi, sondern auch in umgekehrter Richtung. Bisher wurden die Daten immer über die USB-interne serielle Schnittstelle übertragen. Dies hat verschiedene Nachteile:

- Es ist immer eine galvanische Verbindung zwischen Pico und Raspberry Pi erforderlich
- Die Stromversorgung des Pico erfolgt über den Raspberry Pi
- Da sowohl der Raspberry Pi als auch der Pico über (mindestens) eine weitere separate RS232-Schnittstelle verfügen, können die oben genannten Nachteile vermieden werden.

  Für die Verbindung der beiden Boards sind dann lediglich drei einfache Leitungen erforderlich:
- Ground (GND)
- TxD (Raspberry Pi) → RxD (Pico)
- RxD (Raspberry Pi) → TxD (Pico)


*Bild 9: Die seriellen Schnittstellen des Raspberry Pi*

Der Pico kann dann unabhängig, z. B. mit einer USB-Powerbank, betrieben werden. Auch eine vollständige galvanische Trennung kann jetzt etwa mit zwei Optokopplern sehr leicht durchgeführt werden.

Für den Betrieb der seriellen Schnittstelle auf dem Raspberry Pi müssen zwei Voraussetzungen erfüllt sein. Zum einen muss die Schnittstelle explizit freigegeben werden. Dies erfolgt über

```
sudo raspi-config
```

in den Interface-Optionen und die Serial-Port-Einstellung (Bild 8). Hier wird das serielle Interface freigeschaltet (Enable). Die serielle Login-Shell sollte „disabled" sein, damit keine unerwünschten Bytes auflaufen. Mit dieser Einstellung steht die serielle Schnittstelle des Raspberry Pi als „ttyS0" für eigene Anwendungen zur Verfügung. Die verfügbaren Schnittstellen können über den Befehl

```
dmesg | grep tty
```

im Terminal (Konsole) abgefragt werden (Bild 9).

Zum anderen muss der Anwender über Superuser-Rechte verfügen, da ansonsten der direkte Zugriff auf die Hardware gesperrt ist. Im Bedarfsfall kann ein Superuser über „sudo su" und die Vergabe eine Superuser-Passworts erstellt werden. Hardwareseitig steht die Schnittstelle an den Pins 08 (TxD) und 10 (RxD) am Raspberry Pi und an Pin 1 (TxD) und Pin 2 (RxD) am Pico zur Verfügung. Die Verbindung der beiden Boards muss damit so aussehen (Bild 10):

- Raspberry Pi Tx (Pin 8) → Pico Rx (Pin 2)
- Raspberry Pi Rx (Pin 10) → Pico Tx (Pin 1)
- Raspberry Pi GND (Pin 14) → Pico GND (Pin 8)

Die 1-KΩ-Widerstände dienen zur Strombegrenzung, um die Komponenten bei einer fehlerhaften Beschaltung zu schützen. Natürlich muss der Pico nun mit einer eigenen Spannungsquelle über den USB-Anschluss versorgt werden. Vor der Verbindung der beiden Ground-Pins sollte man auch immer prüfen, ob diese tatsächlich auf gleichem Potential liegen. Wenn beide Boards aus unterschiedlichen Netzquellen gespeist werden, könnten ansonsten unerwünschte Ausgleichsströme fließen.

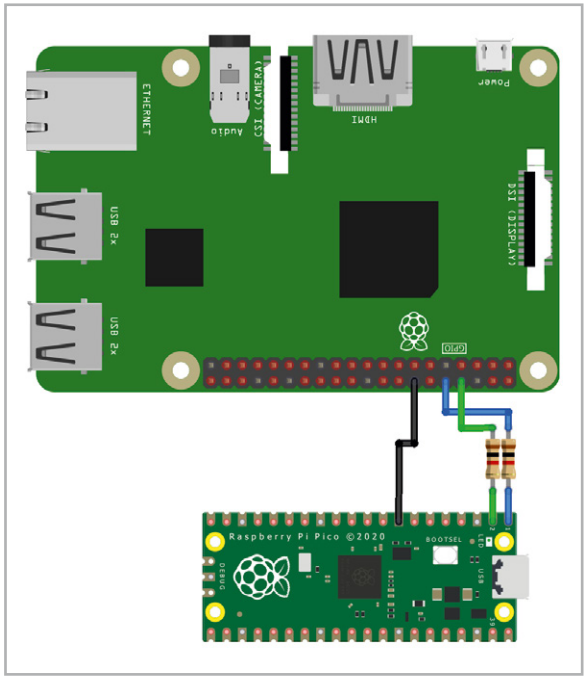



*Bild 10: Serielle Verbindung zwischen Pico und Pi*

## Programmieren mit mu

Prinzipiell könnten die Steuerprogramme für den Raspberry Pi auch wieder mit der Thonny IDE erstellt und ausgeführt werden. Für interaktive Steueraufgaben bietet die mu IDE allerdings einige Vorteile. Mu ist standardmäßig bereits im Raspberry Pi OS enthalten. Bei Bedarf kann mu jedoch auch über

```
sudo apt-get install mu-editor
```

nachinstalliert werden. Nach dem Start des Editors mu sollte direkt der Pygame-Zero-Modus über

```
Modus -> Pygame Zero
```

aktiviert werden (Bild 11).

Dass dieser Modus speziell für die Spieleentwicklung gedacht ist, erkennt man daran, dass der Programmstart jetzt nicht mehr mit „Ausführen", sondern mit „Spielen" bezeichnet ist. Dennoch eignet sich dieser interaktive Modus auch ganz hervorragend zum Erstellen allgemeiner Benutzeroberflächen, auch wenn diese keinen Bezug zu einem Computerspiel haben.

## Pygame Zero und interaktive Steuerung

Pygame Zero stellt die bekannte Pygame-Library von Python in interaktiver Form zur Verfügung. Damit wird es besonders leicht, interaktive Grafiken zu programmieren. Im Folgenden soll diese Möglichkeit genutzt werden, um ein Multichannel-Messsystem für die Kombination aus Raspberry Pi und Pico zu erstellen. Für Testzwecke kann dafür an jeden ADC-Eingang des Pico ein eigenes Potentiometer (analog zu Bild 4) angeschlossen werden. Das Titelbild des Beitrags zeigt einen entsprechenden Aufbau auf dem EXSB1-Experimentierboard.

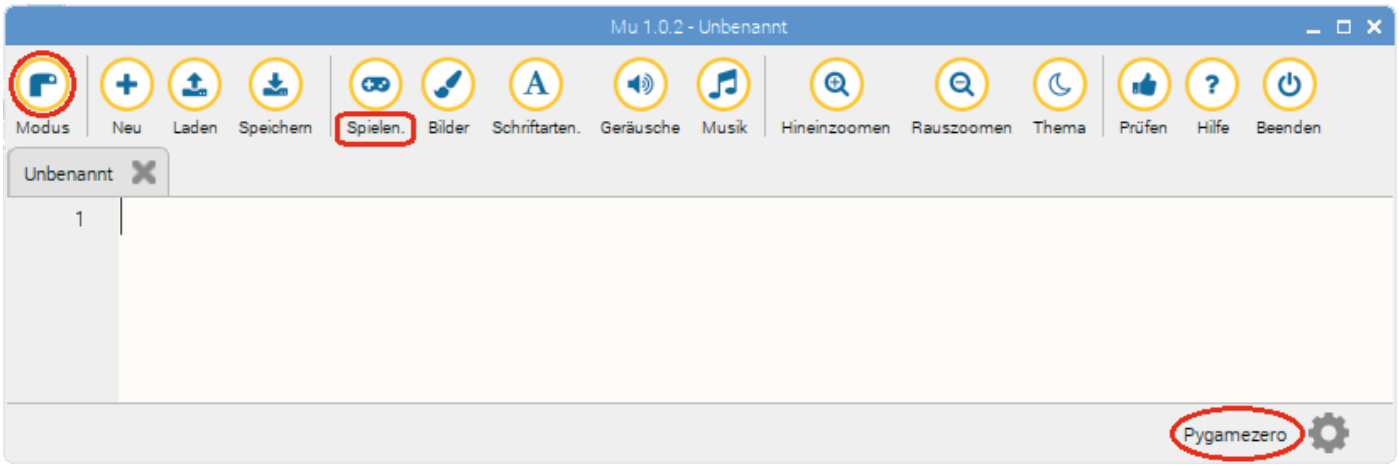



*Bild 11: Mu im Pygame-Zero-Modus*

Das zugehörige Python-Programm für den Raspberry Pi (Voltmeter_PyGame_RasPi_1V2.py im Ordner „Multi-channel_System“ [3]) sieht so aus:

```
import serial, time
from math import floor

WIDTH=600
HEIGHT=420
GREY=(195,195,195)

ser = serial.Serial('/dev/ttyS0', baudrate = 115200, timeout=5)

pico_img = Actor('pico_adcs')
pico_img.topleft = 10,10

pin_img_disabled = "pin-disabled"
pin_img_enabled = "pin-enabled"

pin_imgs = [
    Actor(pin_img_enabled, (300, 210)),
    Actor(pin_img_disabled, (300, 190)),
    Actor(pin_img_disabled, (300, 155)),
    ]

selected_pin = 0
voltage = 0


def draw ():
    screen.clear()
    screen.fill((GREY))
    pico_img.draw()
    for this_pin in pin_imgs :
        this_pin.draw()
    screen.draw.text("RasPi/Pico", centery=20, centerx=400, fontsize=50, color="#000080")
    screen.draw.text("Multi-channel", centery=60, centerx=400, fontsize=50, color="#000080")
    screen.draw.text("System", centery=100, centerx=400, fontsize=60, color="#000080")
    screen.draw.text("ADC"+str(selected_pin)+":", topleft=(300,250), fontsize=50, color="#000080")
    screen.draw.text(str(voltage), topleft=(300,300), fontsize=100, color="#000080")
    screen.draw.text("V", topleft=(450,300), fontsize=100, color="#000080")


def update ():
    global voltage
    ser.write((selected_pin+48).to_bytes(1, byteorder='little'))
    input=ser.read_until()
    sub_input=input.decode("utf-8").split()
    if (str(sub_input[1])==str(selected_pin)):
        voltage=round(float(sub_input[3]),2)
    print (str(sub_input))

def on_mouse_down(pos, button):
    global selected_pin
    if button == mouse.LEFT:
        for pin_id in range (0,len(pin_imgs)) :
            if pin_imgs[pin_id].collidepoint(pos):
                pin_imgs[selected_pin].image = pin_img_disabled
                selected_pin = pin_id
                pin_imgs[selected_pin].image = pin_img_enabled
```

Das Programm besteht im Wesentlichen aus den drei Routinen:

- def draw():
- def update():
- def on_mouse_down(pos, button):

In „draw“ wird das aktive Fenster aufgebaut. Es besteht aus einem Bild des Pico, einer Auswahl für den aktiven Kanal, dem Bildtitel und der Darstellung der aktuellen Messwerte.

Die Auswahl des aktuellen Kanals erfolgt über drei Pfeile, die auf den jeweiligen ADC-Eingang des Controllers deuten. Der jeweilige Kanal kann einfach durch Anklicken des gewünschten Pfeils aktiviert werden. Der aktive Pfeil erscheint dabei grün, die inaktiven Pfeile sind dagegen ausgegraut.

Durch diese sehr intuitive Darstellung wird die Steuerung des Pico zum Kinderspiel für den Anwender (Bild 12).

Die Routine „update" überträgt die Kanalnummer an den Pico und liest den aktuell gemessenen Wert zurück. Sie steuert also die Kommunikation zwischen Raspberry Pi und Pico.

In „on_mouse_down" wird schließlich dafür gesorgt, dass der aktive Kanal über die farbig markierten Pfeile per Mausklick ausgewählt werden kann.

Das Programm für den Pico (Multichannel_Pico.py [3]) liefert die Messdaten:

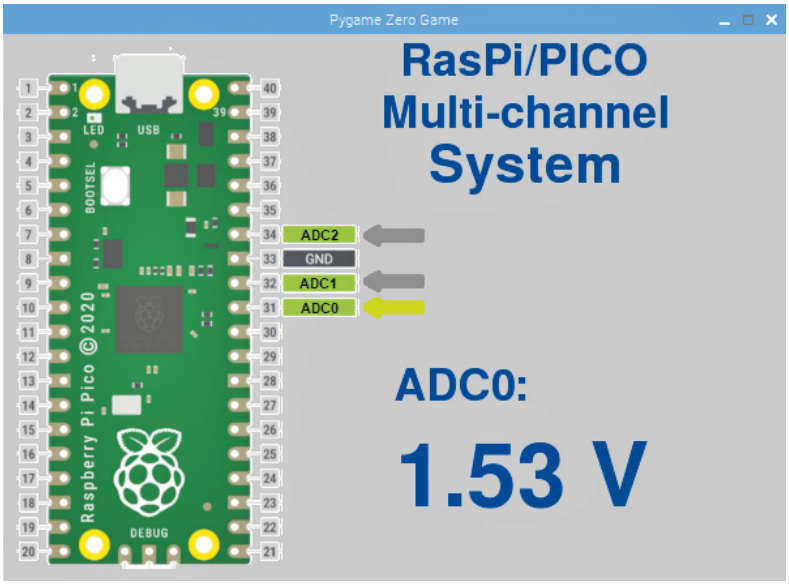



*Bild 12: Interaktive Oberfläche des Multikanal-Messsystems*

```
from machine import UART, ADC
from time import sleep

uart = UART(0,115200, timeout=5)

adc_pin = 0
adc_value = 0
adc = machine.ADC(adc_pin)

conversion_factor=3.3/(2<<15);

while True:
    ch=str(uart.read(1))
    ch=ch[2]
    try:
        ch=int(ch)
    except:
        ch=0
    adc=machine.ADC(ch)
    adc_value=adc.read_u16()
    print ("Pin {} value {}".format(ch, adc_value * conversion_factor))
    uart.write ("Pin {} value {}\n".format(ch, adc_value * conversion_factor))
    sleep(1)
```

Nach dem Importieren der erforderlichen Bibliotheken wird die erste serielle Schnittstelle (#0) des Pico mit einer Baudrate von 115200 und einem Time-out von 5 s gestartet:

```
uart = UART(0,115200, timeout=5)
```

Danach werden die Default-Werte für die Messwerte und die Kanalnummer festgelegt. Der Konversionsfaktor normiert wieder die Werte auf die Betriebsspannung des Pico (3,3 V) und die Auflösung des ADCs von 16 Bit. In der Hauptschleife wird die Kanalnummer aus der seriellen Schnittstelle gelesen. Wenn diese gültig ist, wird der entsprechende ADC-Kanal aktiviert. Ansonsten bleibt der Default-Channel aktiv. Schließlich wird der Wert über die Schnittstelle zum Raspberry Pi übertragen. Für Kontrollzwecke wird der Datenstrom auch noch an die Konsole ausgegeben. Falls dies nicht erwünscht ist, kann der entsprechende Print-Befehl gelöscht werden.

Für die Inbetriebnahme des Systems muss zunächst das Python-Programm auf dem Pico gestartet werden. Dieses beginnt unverzüglich damit, die Daten des voreingestellten Kanals 0 an die serielle Schnittstelle zu senden.

Dann kann das Programm in der mu-Umgebung über „Spielen" aktiviert werden. Die Messwerte erscheinen nun im grafischen Fenster des Systems (s. Bild 12). Durch Anklicken der Pfeile wird der aktive Kanal gewechselt. Der Vorgang kann auch in der Thonny-Konsole kontrolliert werden. Dort erscheinen die Kanäle als Pin 0 bis Pin 2 (Bild 13). Damit steht ein bidirektionales und interaktives System zur Verfügung, das in vielfältiger Weise auf die unterschiedlichsten Anwendungen adaptiert werden kann.

## Ausblick

In diesem Beitrag wurde gezeigt, wie man den klassischen Raspberry Pi und den Pico auf verschiedene Weisen miteinander verbinden kann. Neben der einfachen USB-Verbindung wurde auch eine direkte serielle Kommunikation vorgestellt.

Die beiden Boards ergänzen sich dabei in idealer Weise. Der Raspberry Pi verfügt über die notwendige Rechenleistung, um auch komplexe grafische Oberflächen flüssig darstellen zu können. Der Pico liefert mit seinen integrierten, hochwertigen

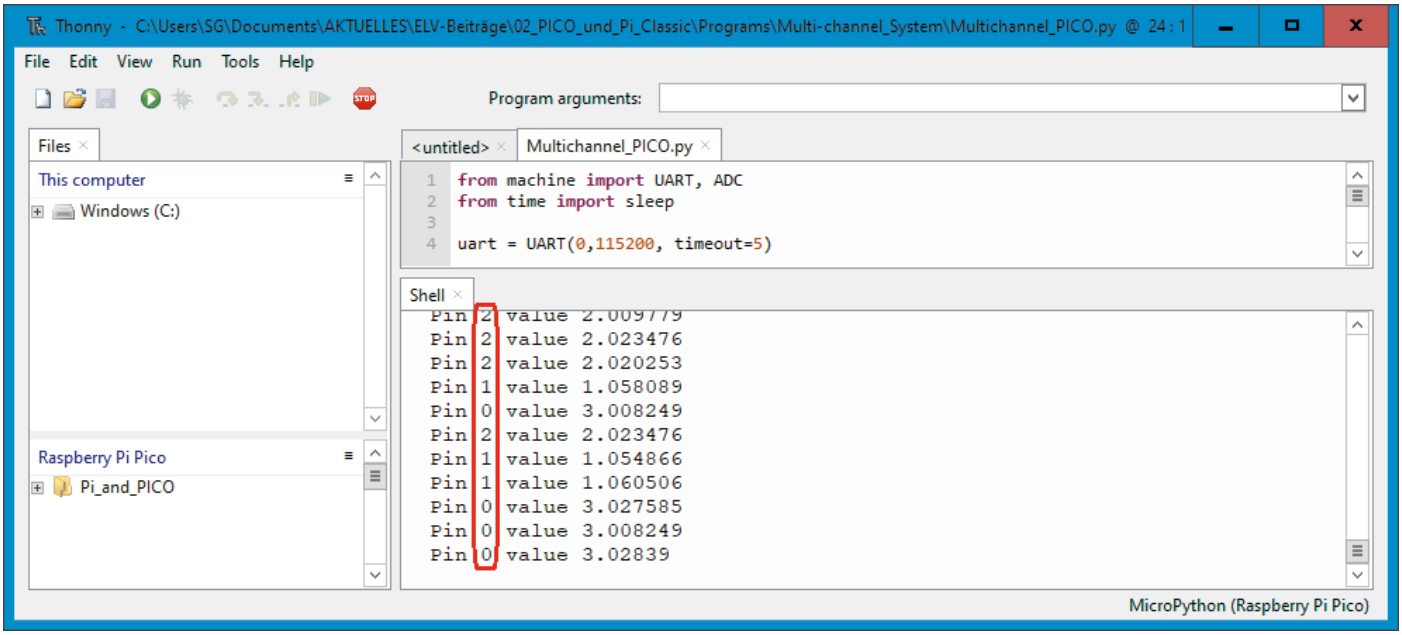



*Bild 13: Anzeige von Messwert und Kanal (Pin) in der Konsole*

Analog-Digitalwandlern präzise Messwerte. Damit lassen sich einfach und sehr kostengünstig interaktive Messsysteme aufbauen, die vielfältige Anwendungen im Elektroniklabor oder in der Hausautomatisierung abdecken können.

Im nächsten Beitrag werden die Anwendung der State-Machines und der programmierbaren I/O-Pins (PIOs) im Vordergrund stehen. Diese erlauben die Implementierung extrem schneller Schnittstellen. Sogar Signale mit Frequenzen von über 50 MHz lassen sich präzise erzeugen. Damit rücken hochinteressante Anwendungen wie etwa die kontinuierliche und flackerfreie Ansteuerung von Neopixeln oder aber der Aufbau eines HF-Signalgenerators in den Bereich des Machbaren. ELV

## i Weitere Infos

[1] Raspberry Pi Pico: Artikel-Nr. 251905
[2] Thonny IDE: https://thonny.org/
[3] Downloads zum Beitrag: Artikel-Nr. 253159
[4] Fachbeitrag Pico, Teil 1: Artikel-Nr. 253018
[5] ELV Bausatz Experimentier-/Steckboard EXSB1 inkl. Gehäuse: Artikel-Nr. 153753

Alle Links finden Sie auch online unter: de.elv.com/elvjournal-links