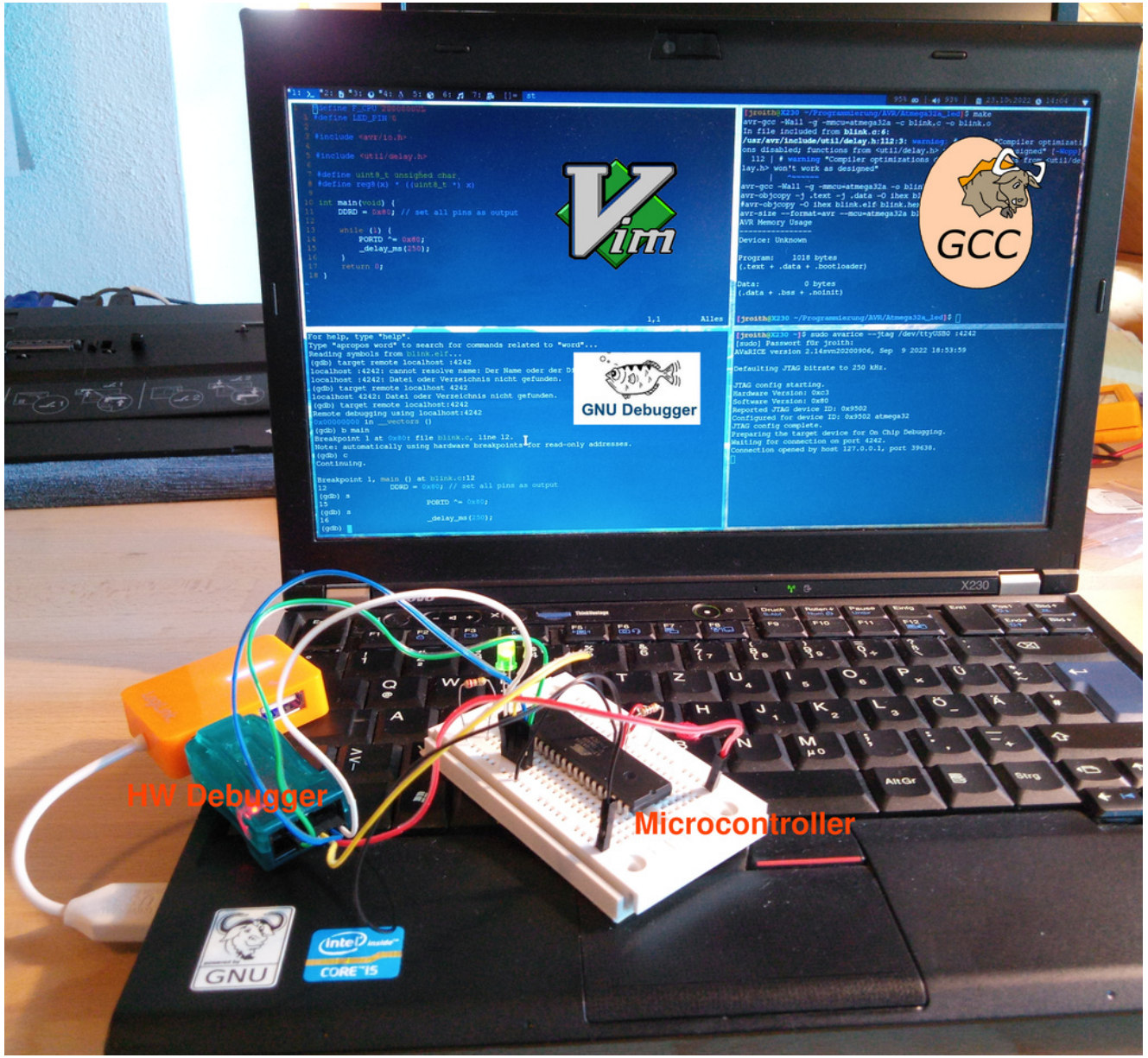
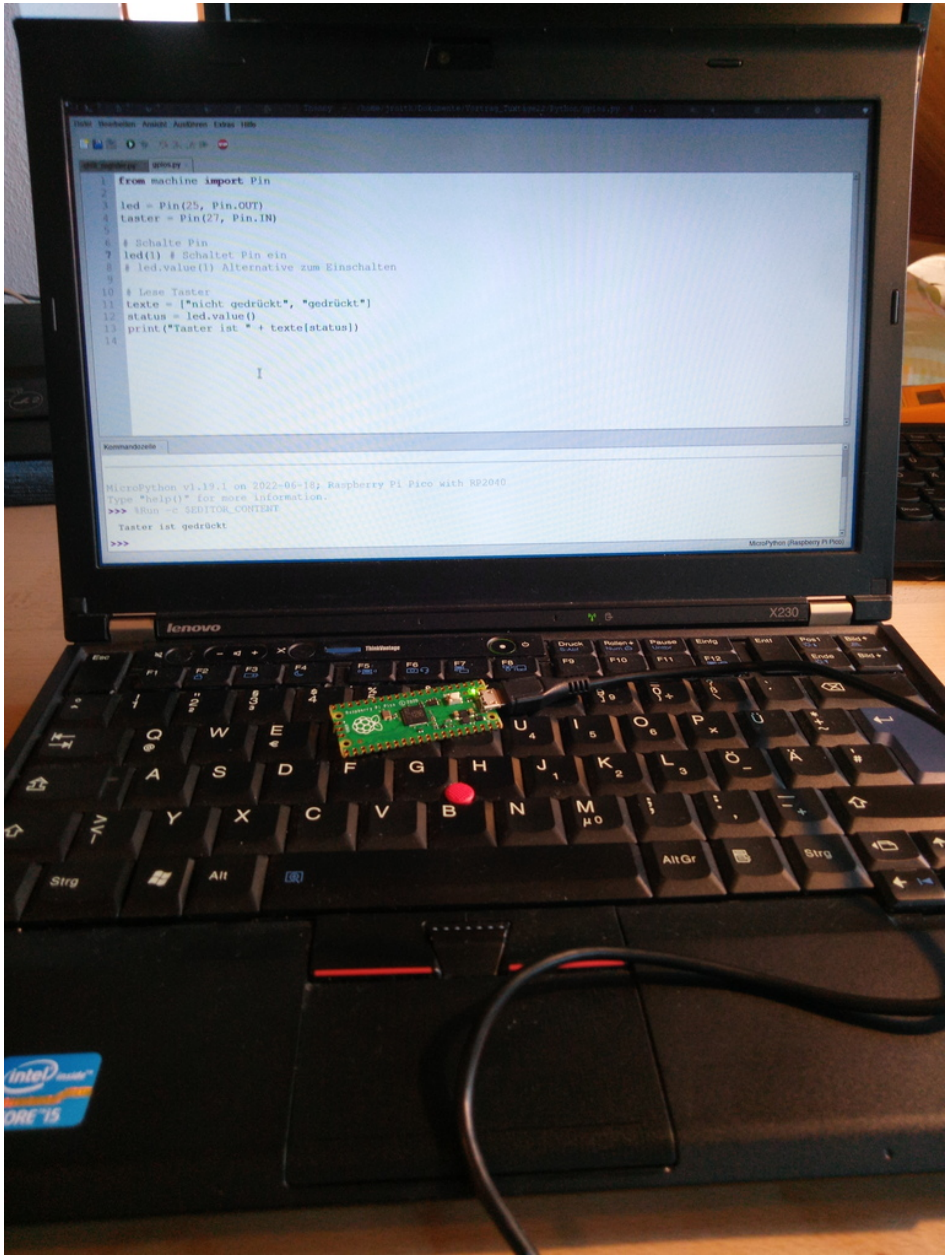

Vim
GCC
GNU Debugger
HW Debugger
Microcontroller
GNU
CORE i5


from machine import Pin
led = Pin(25, Pin.OUT)
taster = Pin(27, Pin.IN)
# Schalte Pin
led(1) # Schaltet Pin ein
# led.value(1) Alternative zum Einschalten
# Lese Taster
texte = ["nicht gedrückt", "gedrückt"]
status = led.value()
print("Taster ist " + texte[status])
MicroPython v1.19.1 on 2022-06-18; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>> $Run -c $EDITOR_CONTENT
Taster ist gedrückt
>>>
lenovo
X230

Micropython

Microcontroller mit Python in Thonny auf Linux programmieren

Johannes Roith

13.11.2022

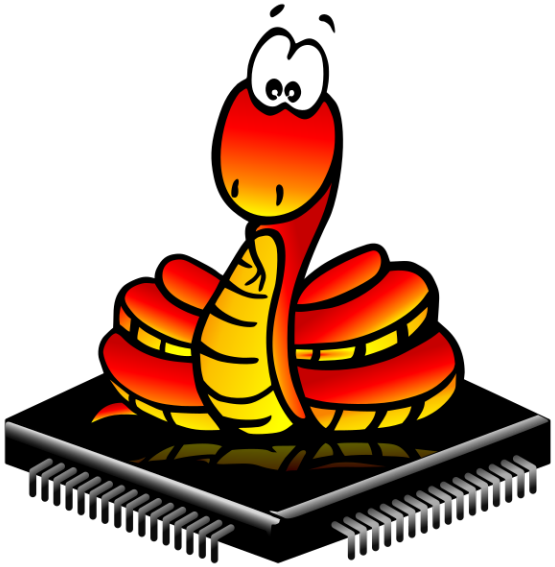
Über mich

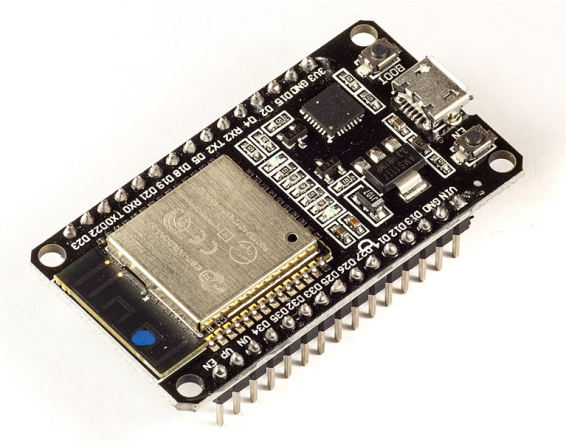
- Hardwarenaher Softwareentwickler
- GNU/Linux Nutzer seit 10 Jahren
- Youtube Kanal: Johannes 4GNU_Linux
- Webseite & Kontakt: `www.gnu-linux.rocks`

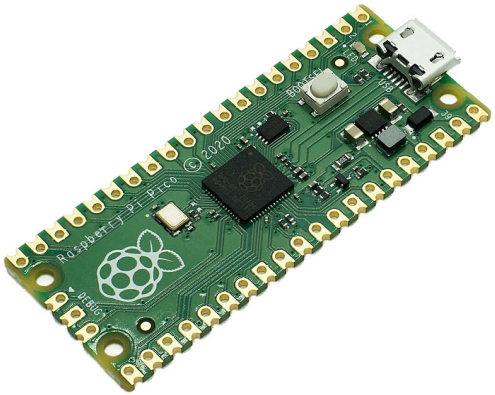
Über den Foliensatz

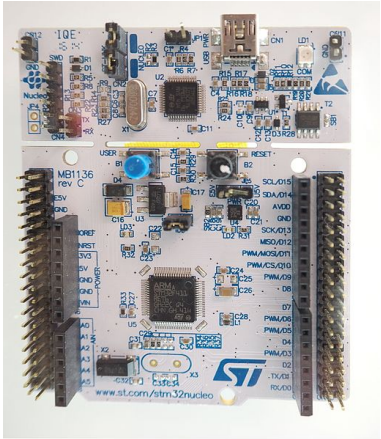
- Folien lizenziert unter
  - Weitergabe und Remix erlaubt
  - Namensnennung notwendig
  - Remix muss unter selben Lizenz weitergegeben werden
- Folien zum Nachschlagen gedacht

Micropython und unterstützte Boards



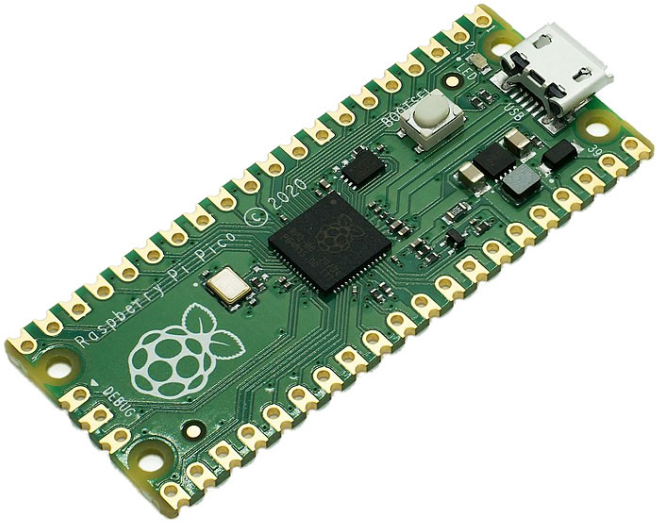


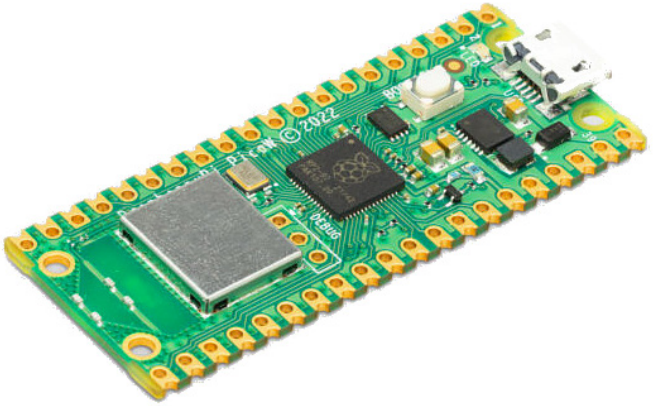




Mikropython Logo MIT Lizenz
ESP32 Board Lizenz: ⓢ
RPI Pico Board von Laserlicht Lizenz: ⓒⓘⓢ
Nucleo Board Lizenz: ⓢ

Der Raspberry Pi Pico





RPI Pico Board von Laserlicht Lizenz: 🅭🅯🅮
RPI Pico W Board von SparkFun Electronics Lizenz: 🅭🅯

Setup Micropython

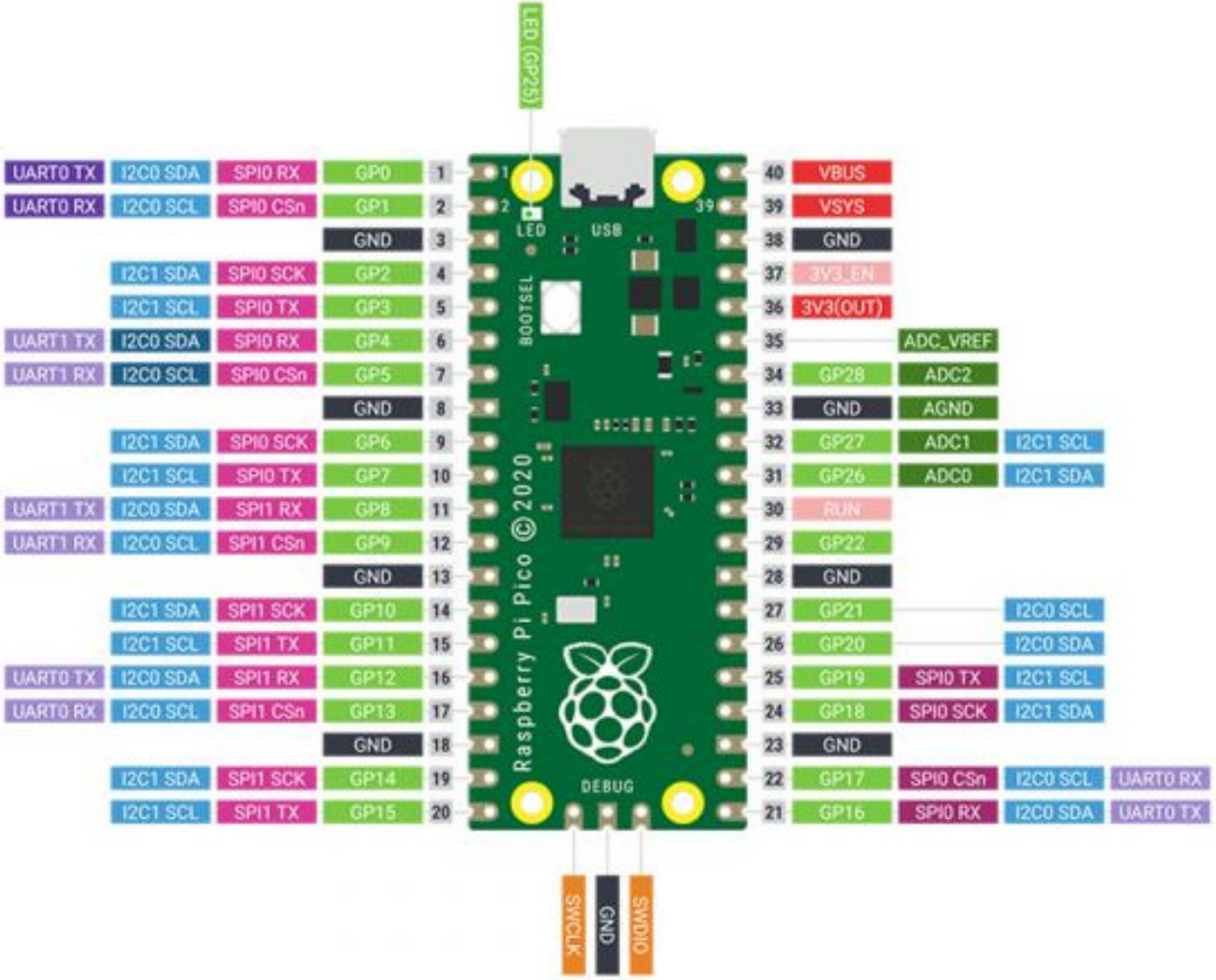
Raspberry Pi Pcio

- UF2 Datei für RPi Pico hier herunterladen
- oder UF2 Datei für RPi Pico W hier herunterladen
- Taster auf Raspberry Pi Pico gedrückt halten und an PC anschließen
- Raspberry Pi Pico wird als Massenspeicher erkannt und vom OS eingebunden
- UF2 Datei auf Raspberry Pi Pico kopieren

Andere Boards

Micropython Binaries und Anleitung zur Installation sind auf der offiziellen Micropython Webseite zu finden.

Die Stiftleiste



Pinout RPi Pico von Yoyojacky Lizenz:

Thonny IDE

Thonny - /home/jroith/Dokumente/Vortrag_Tuxtage22/shift_register.py @ 20 : 16

Datei Bearbeiten Ansicht Ausführen Extras Hilfe

shift_register.py

```python
from machine import SPI, Pin
from time import sleep

spi = SPI(2, baudrate=10000, polarity=0, phase=0, bits=8, firstbit=0, sck=Pin(18), mosi=Pin(23), miso=Pin(19))
cs = Pin(10, Pin.OUT)
cs(1)

def data_in_out(val):
    out_buf = bytearray(1)
    in_buf = bytearray(1)
    cs(0)
    sleep(0.005)
    cs(1)
    sleep(0.005)
    spi.write_readinto(out_buf, in_buf)
    return in_buf[0]

for i in range(16):
    print(data_in_out(i))
    sleep(0.25)
```

Kommandozeile

```
1
>>> data_in_out(0xf)
1
>>> data_in_out(0xf)
5
>>>
```

MicroPython (ESP32)

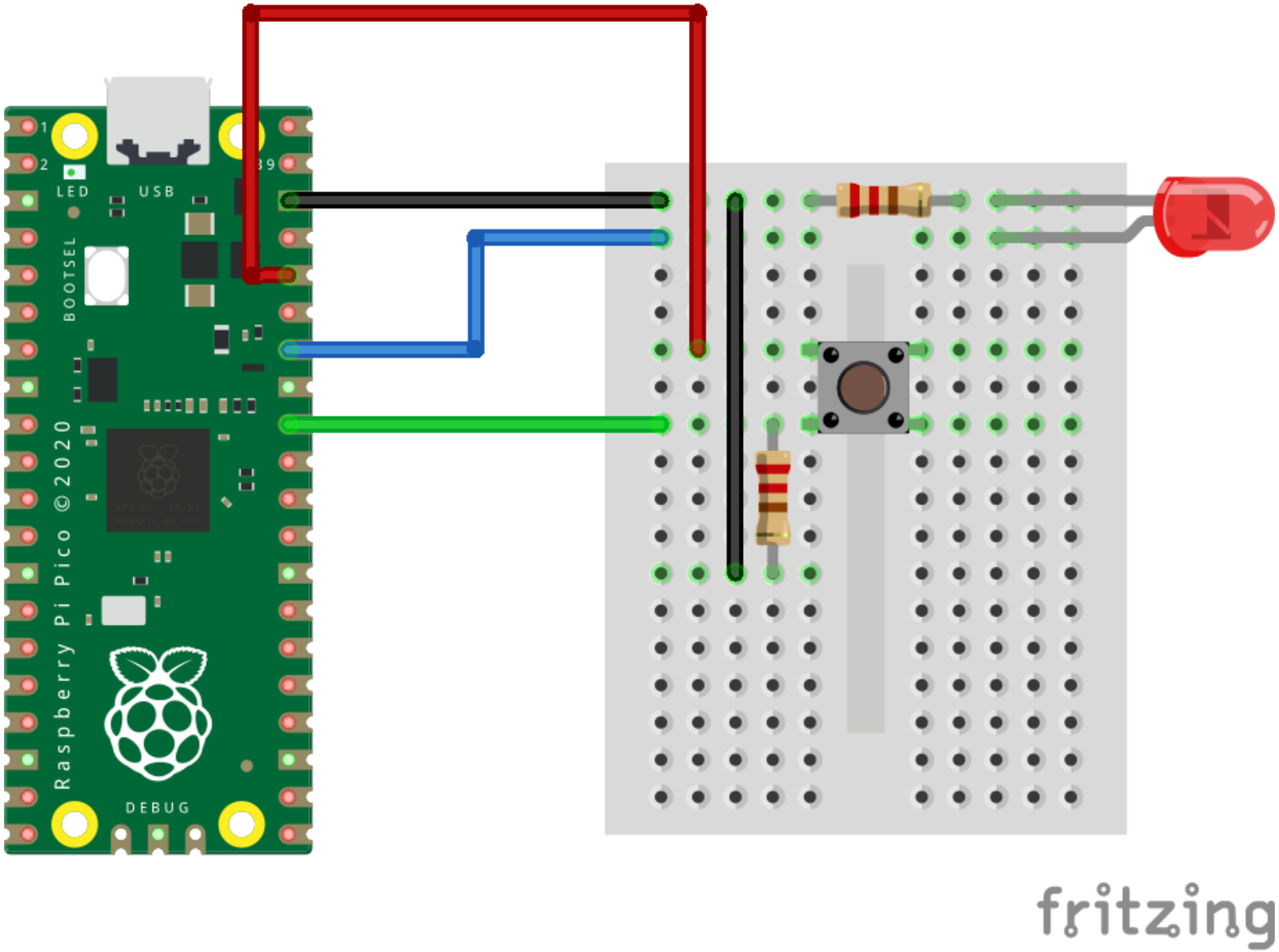
Das schauen wir uns an

1. General Purpose Input/Output Pins
2. Analog Digital Converter
3. I2C Bus
4. SPI Bus
5. Serielle Schnittstelle
6. PWM
7. Online mit Micropython
8. Warum genau Micropython?

General Purpose Input/Output Pins

- Digitale Ein- und Ausgänge
- Spannungspegel beim Raspberry Pi Pico 3,3V
- Achtung: Maximaler Strom begrenzt

GPIO Schaltung



Ansteuerung mit Micropython

```
from machine import Pin

led = Pin(28, Pin.OUT)
taster = Pin(27, Pin.IN)

# Schalte Pin
led(1) # Schaltet Pin ein
# led.value(1) Alternative zum Einschalten

# Lese Taster
texte = ["nicht gedrueckt", "gedrueckt"]
status = taster.value()
print("Taster ist " + texte[status])
```

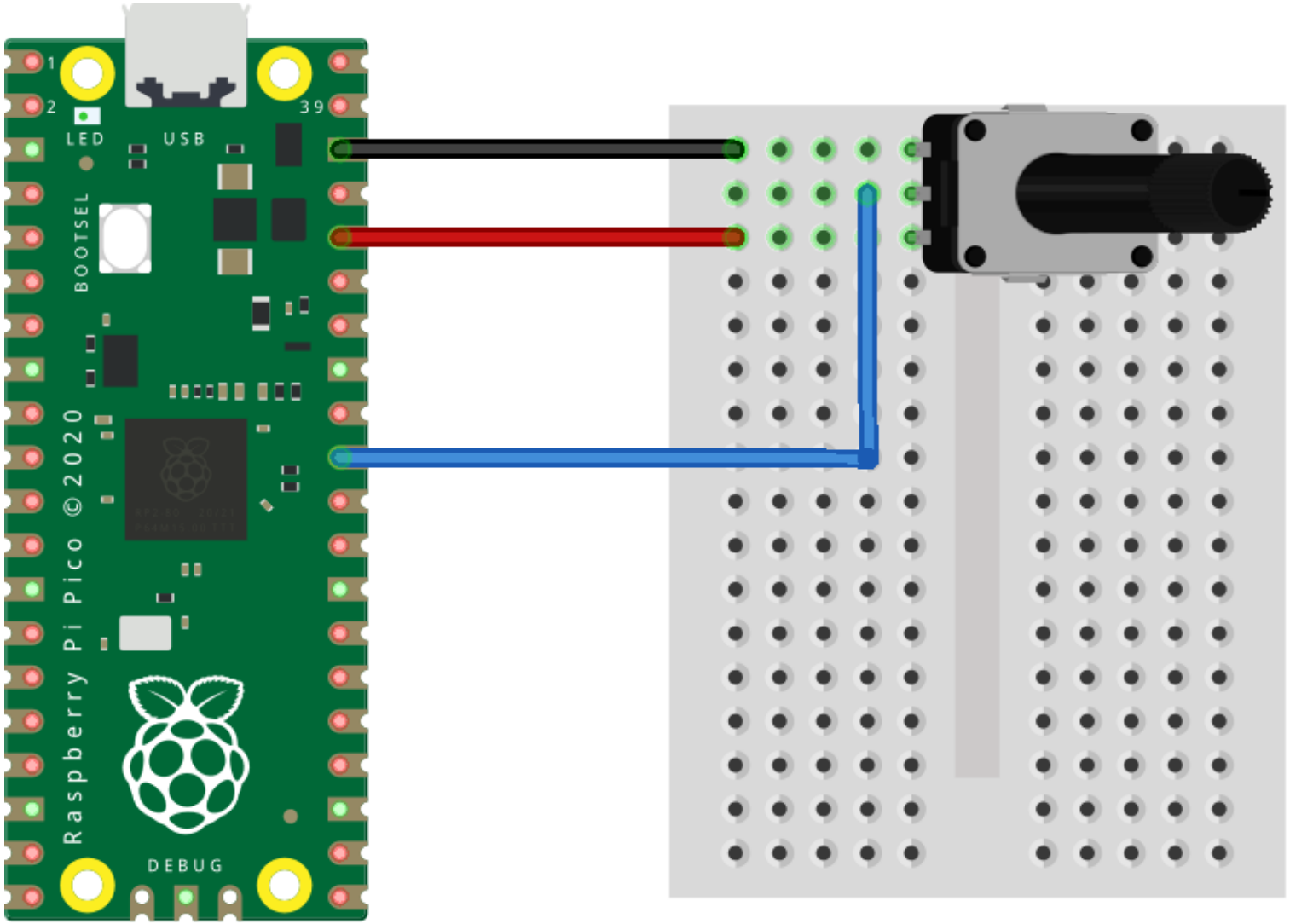
Weitere nützliche Bauteile

- Optokoppler, Relais
- Displays (z.B. HD44780, Segmentanzeigen)
- Ultraschall-Distanzsensor LINKER SEN-US01
- Buzzer
- Taster, DIP-Schalter
- ...

Analog Digital Converter

- Eingänge von analogen Signalen (0 - 3.3V)
- Analogwert wird in eine digitale Zahl gewandelt
- Auflösung abhängig von Hardware (8 - 16 Bit)

Analog Digital Converter Schaltung



Ansteuerung mit Micropython

```
from machine import Pin, ADC

adc = ADC(26)

# Lese ADC
adc.read()
```

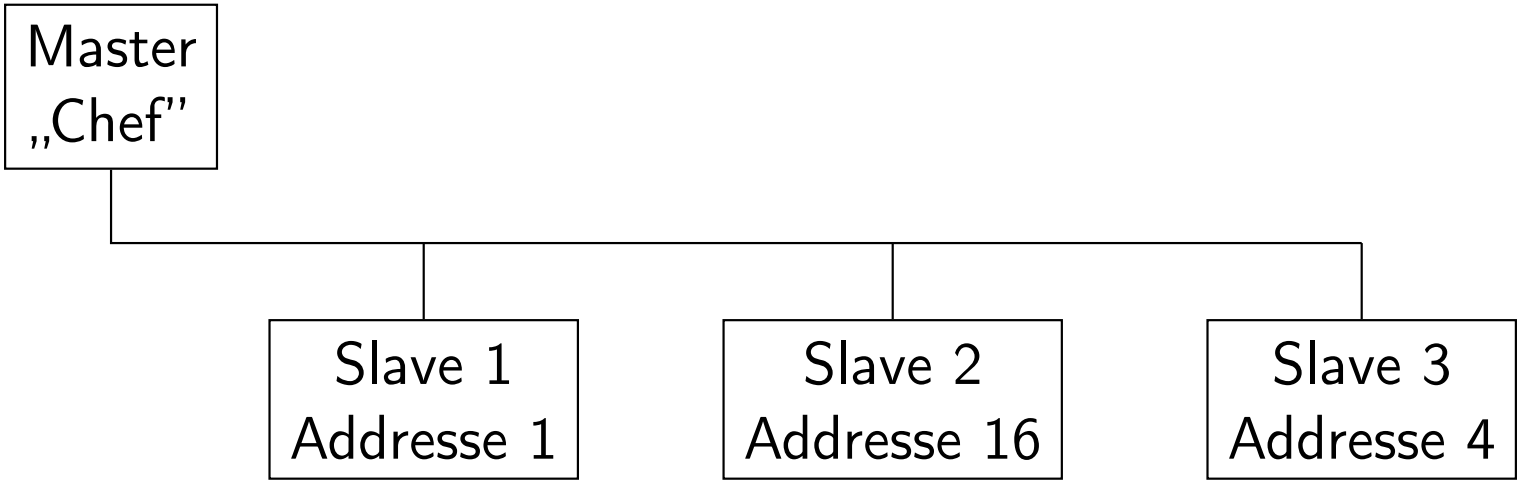
Weitere nützliche Bauteile

- Sensoren mit analogen Ausgang (Feuchtigkeitssensoren, Temperatursensoren)
- Temperaturabhängige Widerstände
- ...

I2C Bus

- Einfacher Zweidrahtbus
- Datenleitung: *SDA*
- Taktleitung: *SCK*
- Frequenzen: 100kbit/s, 400kbit/s, 1Mbit/s
- Pull-Up Widerstand bei Leitungen notwendig

I2C - Was ist ein Bus?



I2C - Komplexere Geräte

Interne Addresse	Register
0x0	Temperatur
0x1	Konfigurationsregister
0x2	Hysterese Einstellungen
0x3	Übertemperatur Einstellung

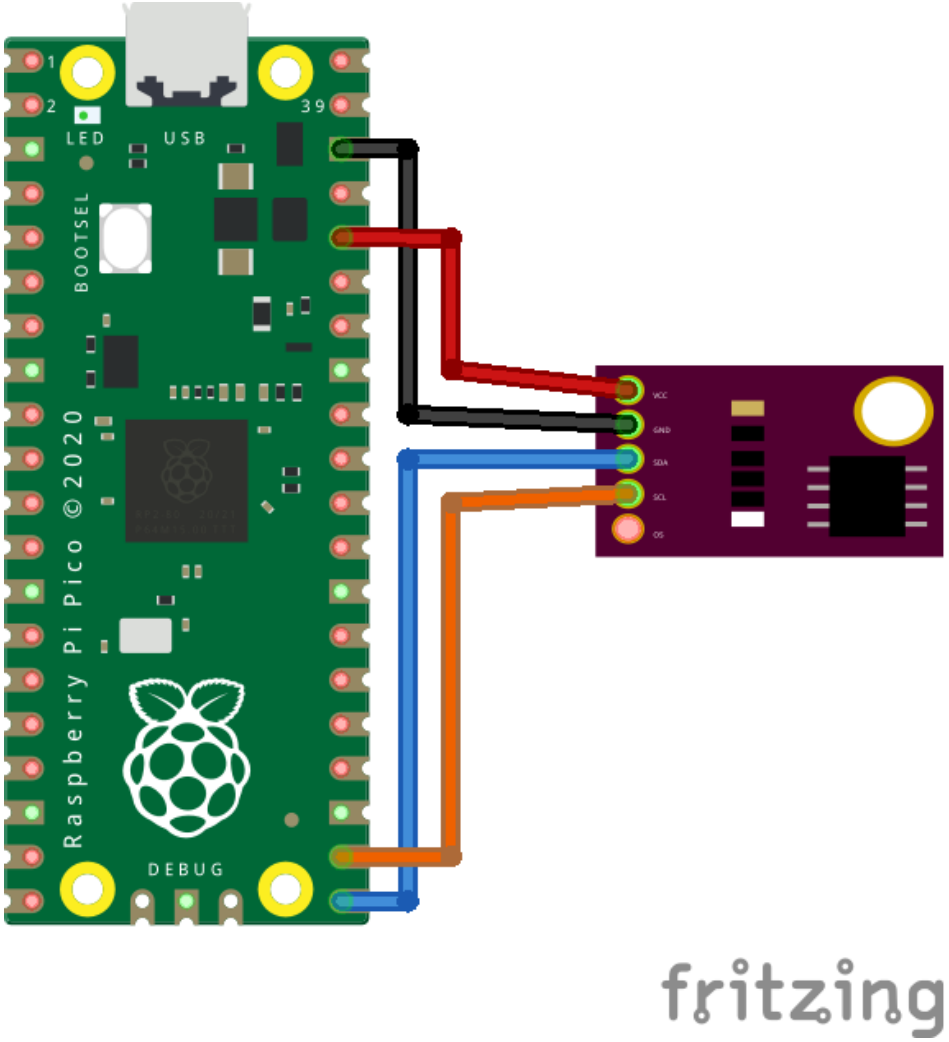
I2C - Komplexere Geräte

Interne Addresse	Register
0x0	Temperatur
0x1	Konfigurationsregister
0x2	Hysterese Einstellungen
0x3	Übertemperatur Einstellung

- Erster Zugriff: Master schreibt gewünschte Adresse
- Zweiter Zugriff: Master liest von Slave, Slave liefert Wert entsprechend der gesetzten Adresse

I2C Bus Schaltung

mit LM75 Temperatur Sensor



Ansteuerung mit Micropython

```
from machine import I2C, Pin

lm78_addr = 0x48
temp_reg = 0x0
ot_reg = 0x3
buf = bytearray(2) # Datenspeicher

i2c = I2C(0, sda=Pin(0), scl=Pin(1), freq=100000)
# Lese temperatur
i2c.readfrom_mem_into(lm78_addr, temp_reg, buf)
# Berechne Temperatur
temp = ((buf[0] << 3) | (buf[1] >> 5)) * 0.125
# Setze Limit auf 30 Grad
buf[0] = ((240 << 5) & 0xFF00) >> 8
buf[1] = ((240 << 5) & 0xFF)
i2c.writeto_mem(lm78_addr, ot_reg, buf)
```

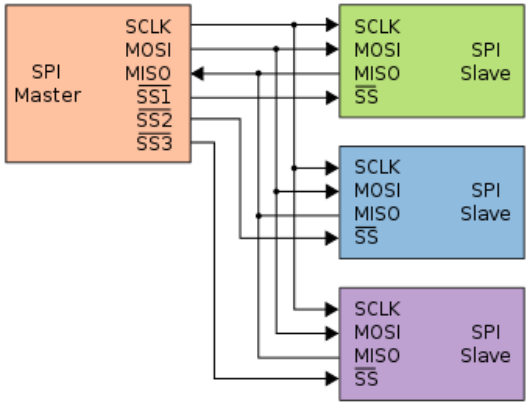
Weitere nützliche Bauteile

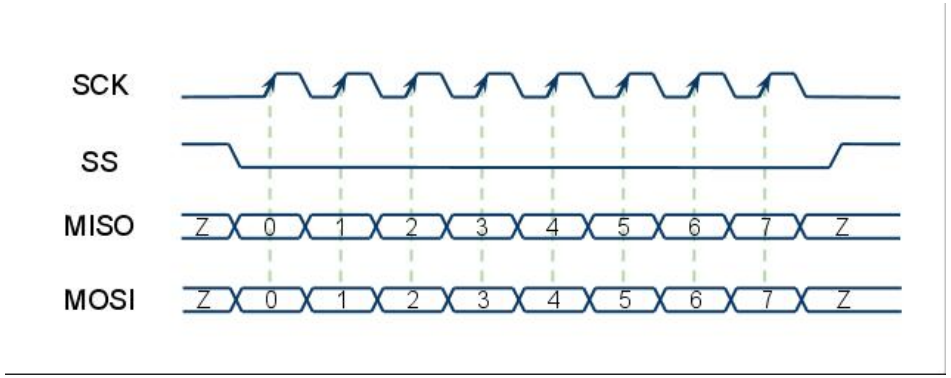
- ADCs / DACs
- BMP280 Temperatur Sensor
- LCD Controller
- GPIO Expander
- Sensoren (Beschleunigung, Luftqualität, ...)
- ...

SPI Bus

- Bidirectionaler Bus
- Master/Slave Prinzip
- *Chip Select (CS):* Auswahl des Slaves
- *Master Output Slave Input (MOSI):* Datenleitung vom Master zum Slave
- *Master Input Slave Output (MISO):* Datenleitung vom Slave zum Master
- *Serial Clock (SCK):* Taktleitung

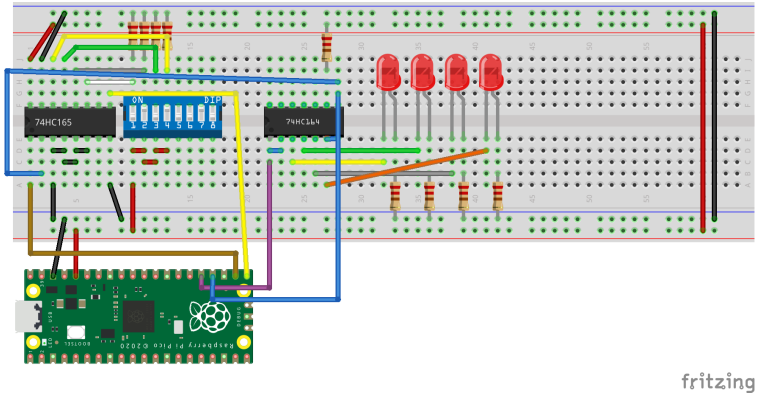
SPI Bus





SPI Bus Schaltung

mit 2 Schieberegister



Ansteuerung über Python

mit zwei Schieberegister (74HC165 und 74HC164)

```
from machine import SPI, Pin

spi = SPI(2, baudrate=20000000, polarity=0, phase=0, bit
cs = Pin(17, Pin.OUT)
cs(1)

out_buf = bytearray(1)
in_buf = bytearray(1)
out_buf[0] = 0x5a
# Taster einlesen
cs(0)
sleep(0.005)
cs(1)
sleep(0.005)
# SPI Transfer
spi.write_readinto(out_buf, in_buf)
print("Taster Status: " + bin(in_buf[0]))
```

Weitere nützliche Bauteile

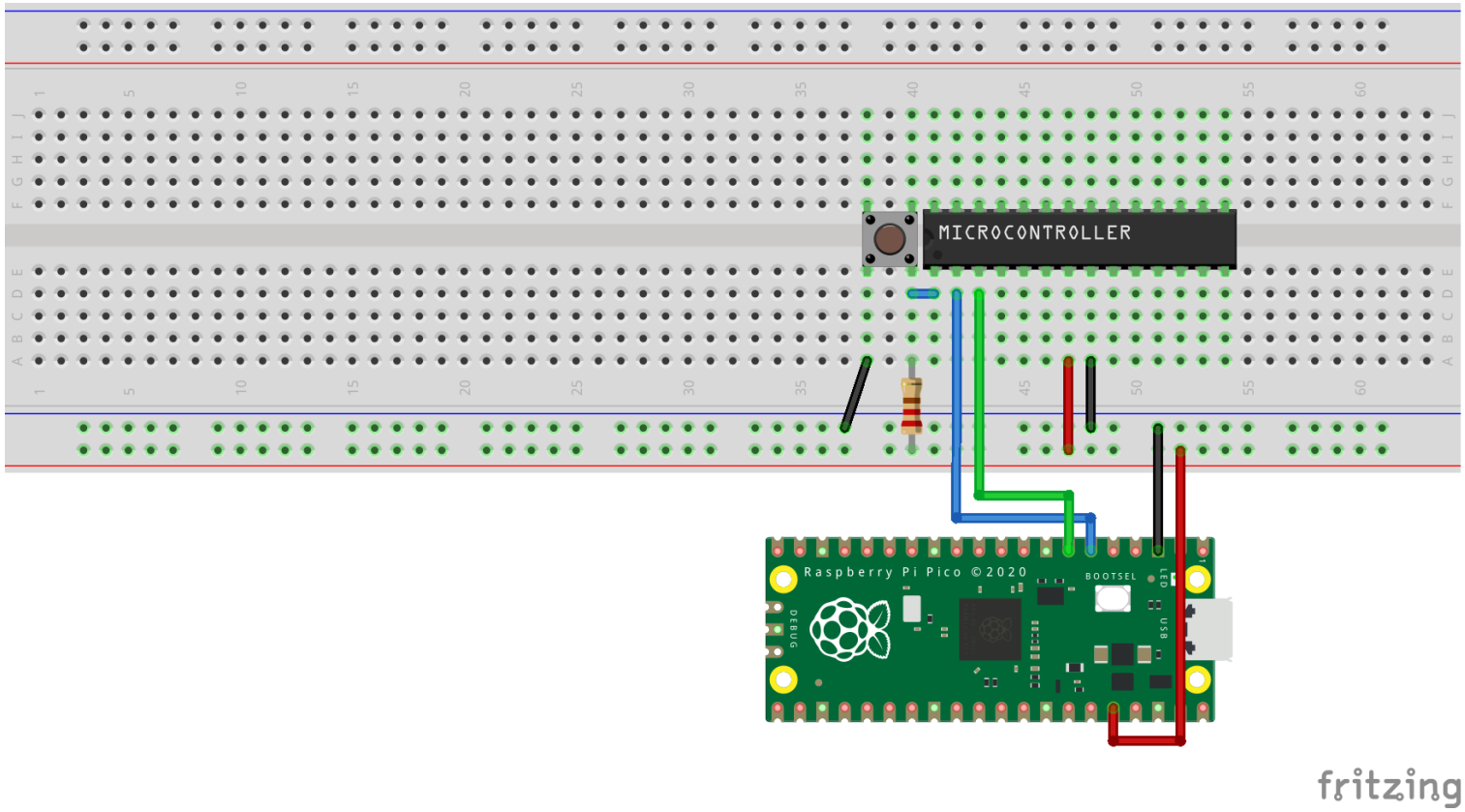
- Anschluss von SD-Karten (Tutorial hier verlinkt)
- SPI Flashes
- RFID Chip Reader
- Displays (Nokia 5510, . . .)
- ADCs und DACs
- Ethernet Controller (ENC28J60, Wiznet W5500)
- GPIO Expander
- . . .

Serielle Schnittstelle

- auch *Universal Asynchronous Receiver Transmitter (UART)* genannt
- Bidirektionale Byteweise Übertragung von Daten (meistens ASCII Characters (Buchstaben))
- Datenleitung zum Senden: *Transmit (TX)*
- Datenleitung zum Empfangen: *Receive (RX)*
- Leitungen müssen kreuzweise mit Gegenstelle verbunden werden
- Datenübertragung bis 3Mbit/s

Schaltung mit serieller Schnittstelle

Kommunikation zwischen Atmega uC & Raspberry Pi



Ansteuerung über Micropyython

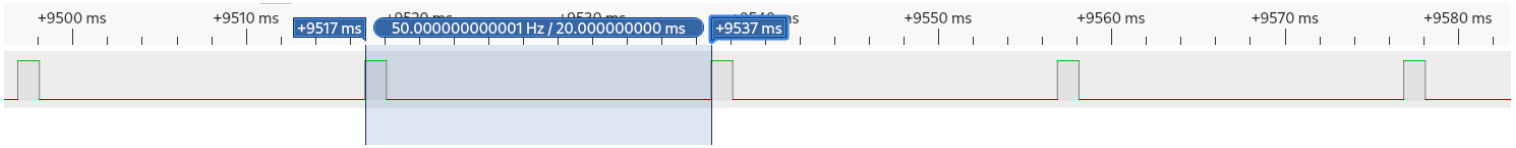
```
from machine import UART, Pin

uart = UART(1, 9600, tx=Pin(4), rx=Pin(5))

# Lese 10 Einkommende Zeichen
uart.read(10)

# Schreibe Zeichen
uart.write('Hello')
```

Weitere nützliche Bauteile

- Pegelwandler MAX232N für RS232
- USB-UART Converter: MCP2200, CH340G, FT232RQ
- ...

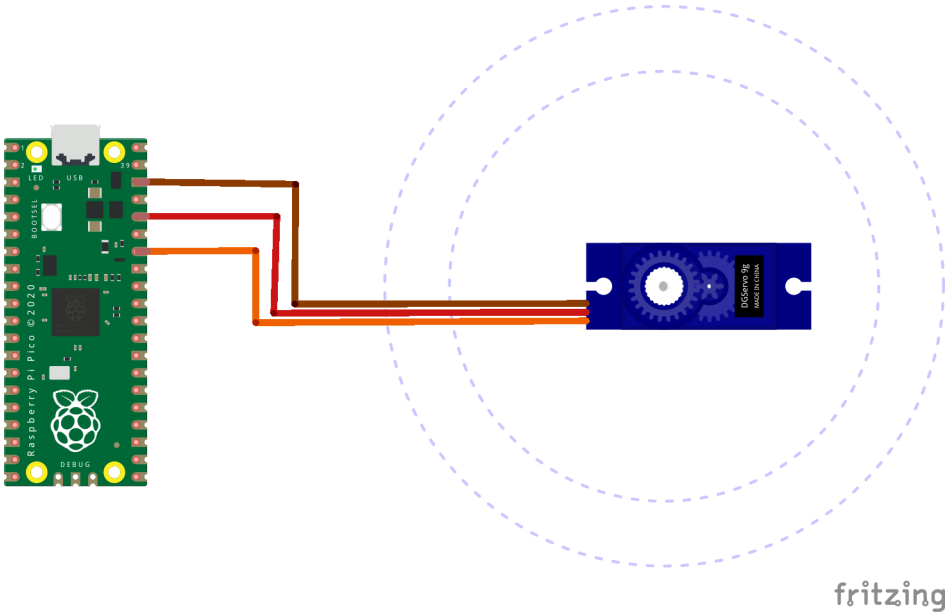
PWM

- Ausgabe einer analogen Spannung über Pulseweitenmodulation
- Z.B. Zum Ansteuern von kleinen Servo-Motoren
- Unterscheidung von Hardware und Software PWM



Schaltung mit PWM

Ansteuerung eines kleinen Servo-Motors



Ansteuerung über Python

am Beispiel eines kleinen Servo-Motors

```
from machine import Pin, PWM

motor = PWM(Pin(28))

# Setze Duty Cycle auf 50%
PWM.duty_u16(int(0.5 * 65535))
```

Weitere nützliche Bauteile

- Micro RC Mini Servo SG90 9G
- Transistoren
- RC-Glieder (Spule und Widerstand), um analogen Sinus zu modellieren

Online mit Micropython

- Nur bei manchen Board unterstützt (RPi Pico W, ESP32, ESP8266)
- Realisierung über Socket Server/Clients
- Tutorial zur Nutzung eines Webservers ist hier verlinkt

Warum genau Micropython?

mit C, C++ oder Rust geht das doch auch

Vorteile gegenüber C, C++, Rust

- einfacheres Setup (kein Compiler, Software Development Kit, ... notwendig)
- interactve Python Shell über UART → kein Compiling notwendig
- relativ einheitliche Schnittstellenansteuerung

Nachteile gegenüber C, C++, Rust

- C, C++, Rust sind oft schneller als Micropython
- kleinere Binaries → mehr Platz im Flash

Warum genau Micropython?

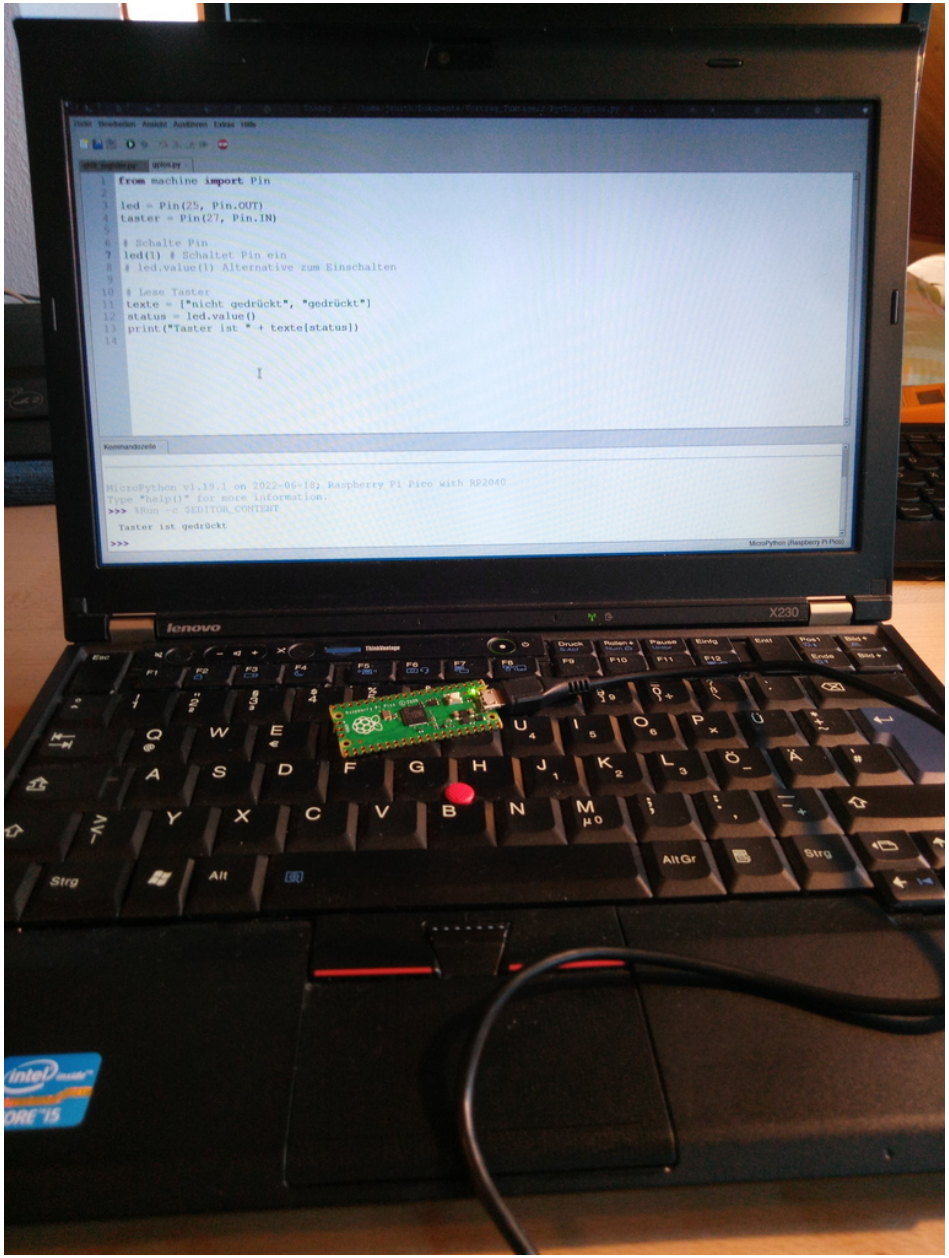
mit Raspberry Pi geht das doch auch

Vorteile gegenüber Raspberry Pi

- Optimiert auf Hardware Zugriff
- schnellere Boot-Zeit
- kein OS Overhead

Nachteile gegenüber Raspberry Pi

- weniger Rechenleistung und Speicher
- schwierigeres Nachinstallieren von Modulen
- weniger fertige Module vorhanden als bei Python am RPi
- kein Betriebssystem


from machine import Pin
led = Pin(25, Pin.OUT)
taster = Pin(27, Pin.IN)
# Schalte Pin
led(1) # Schaltet Pin ein
# led.value(1) Alternative zum Einschalten
# Lese Taster
texte = ["nicht gedrückt", "gedrückt"]
status = led.value()
print("Taster ist " + texte[status])
MicroPython v1.19.1 on 2022-06-18; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>> $Run -c $EDITOR_CONTENT
Taster ist gedrückt
>>>
lenovo
X230

Quellen

- Micropython Webseite `https://micropython.org/`
- Raspberry Pi Pcio Webseite `https://www.raspberrypi.com/documentation/microcontrollers/micropython.html`
- Mcauser LM75 Example `https://github.com/mcauser/micropython-lm75a`
- Datenblätter von 74HC164, 74HC165, LM75