



MicroPython Demoboard

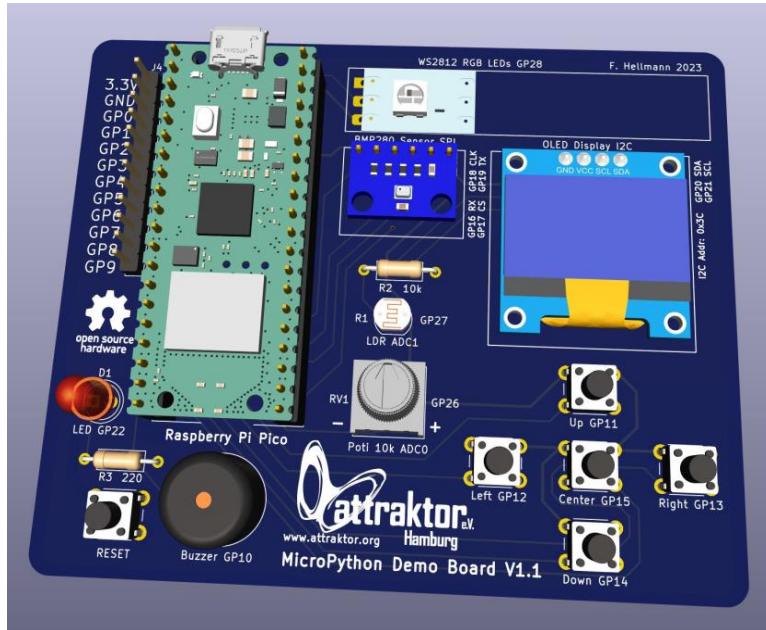
Übersicht über das MicroPython Demoboard mit Programmierbeispielen

OSHW unter MIT Lizenz

Frank Hellmann - 2024

Vorstellung des Attraktor MicroPython Demoboards

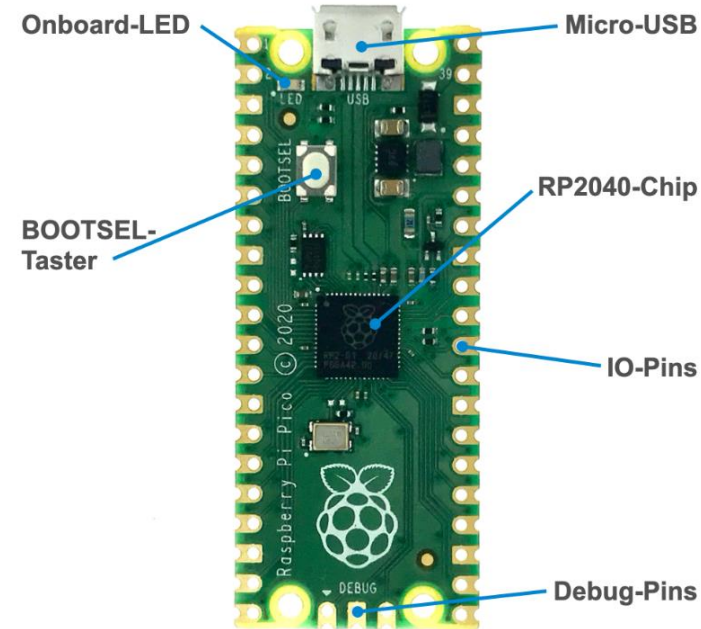
Das MicroPython Demoboard soll den einfachen Einstieg in die hardwarenahe Programmierung mit MicroPython erlauben und eine gute Basis für verschiedene Sensoren, Ein- und Ausgabe-Geräte über verschiedenste Bus-Systeme zur Veranschaulichung bieten. Keine Breadboards und Kabel sind nötig.



Hardware des MicroPython Demoboard:

- Pi Pico W Micro-Controller mit 2MB Flash, 264KB RAM, 28 Ein- und Ausgabe-Pins, Wifi
- OLED Display: 128x64, blau, SSD1306
- WS2812 LED Streifen mit 5 RGB LEDs
- 1 blaue LED
- Temperatur/Luftfeuchte Sensor: BMP280
- Helligkeitssensor: LDR
- Potentiometer
- 5 Taster
- 1 Reset Taster
- 10 freie I/O Pins bis max. 3.3V
- Frei verfügbare Schaltpläne und Fertigungsdateien.

Raspberry Pi Pico: Was ist das?

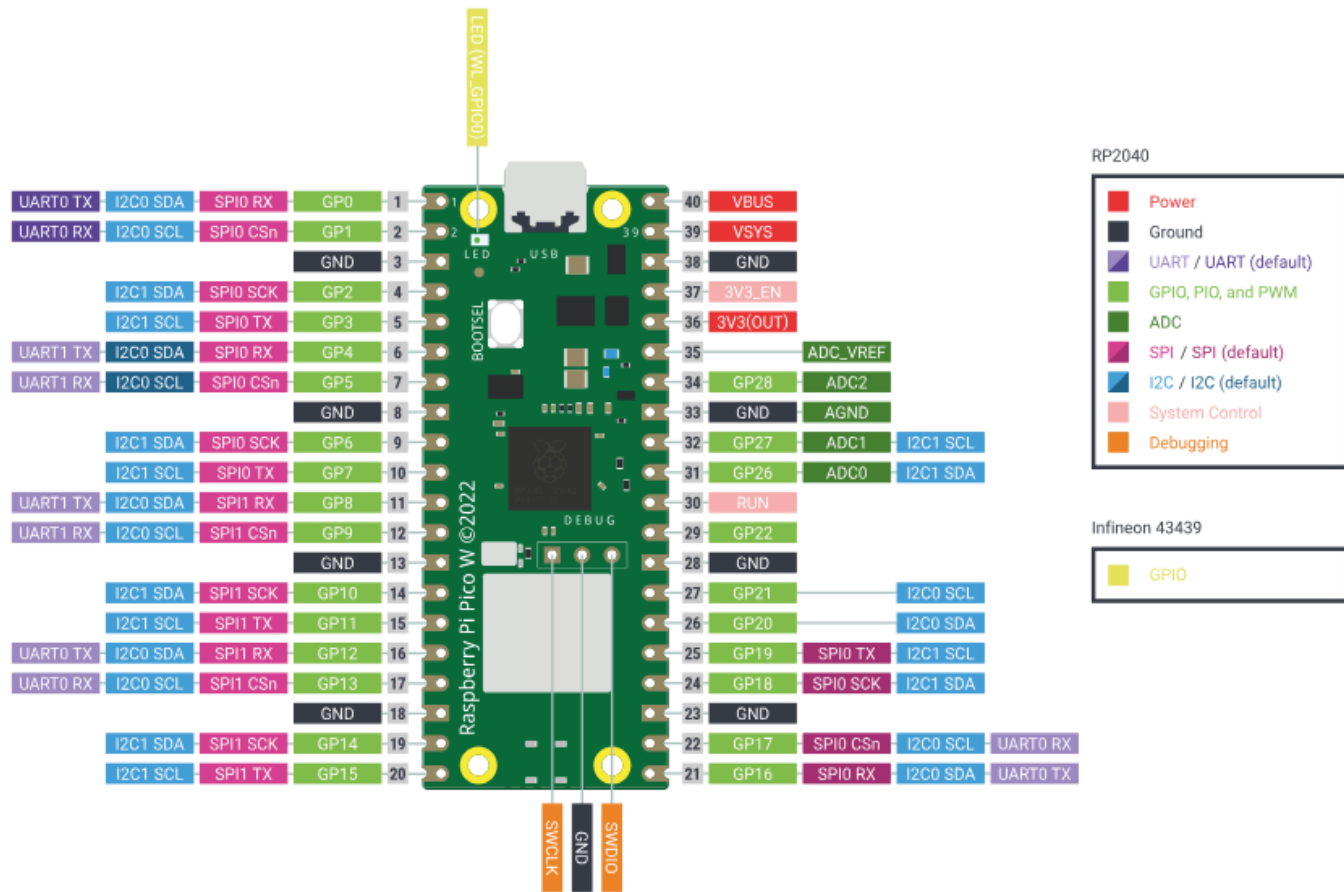


Ein Raspberry Pi Pico ist ein Mikrocontroller-Board von der Raspberry Pi Foundation. Oft wird er liebevoll einfach nur „Pico“ genannt. Er hat einige nützliche Funktionen für Steuerungen, wie man es von anderen Mikrocontrollern kennt. Er lässt sich mit MicroPython, C/C++ und Thonny oder Visual Studio Code programmieren. Die Platine lässt sich über eine Micro-USB-Buchse mit Strom versorgen und programmieren.

Der Mikrocontroller ist ein RP2040. Zum Speichern des Programmcodes ist ein 2 MByte großer Flash-Speicher und 264KB RAM vorhanden. Eine Besonderheit, im Vergleich zu anderen Mikrocontrollern, sind die Programmable IOs (PIO), die unabhängig programmierbar sind.

Ein Mikrocontroller, wie der Pico, ist dafür gedacht, einzelne Aufgaben zu übernehmen. Typischerweise aus dem Bereich der Steuerung, Regelung und Automation. Dafür verfügt er über zahlreiche analoge und digitale Eingänge und Ausgänge mit unterschiedlichen Funktionen.

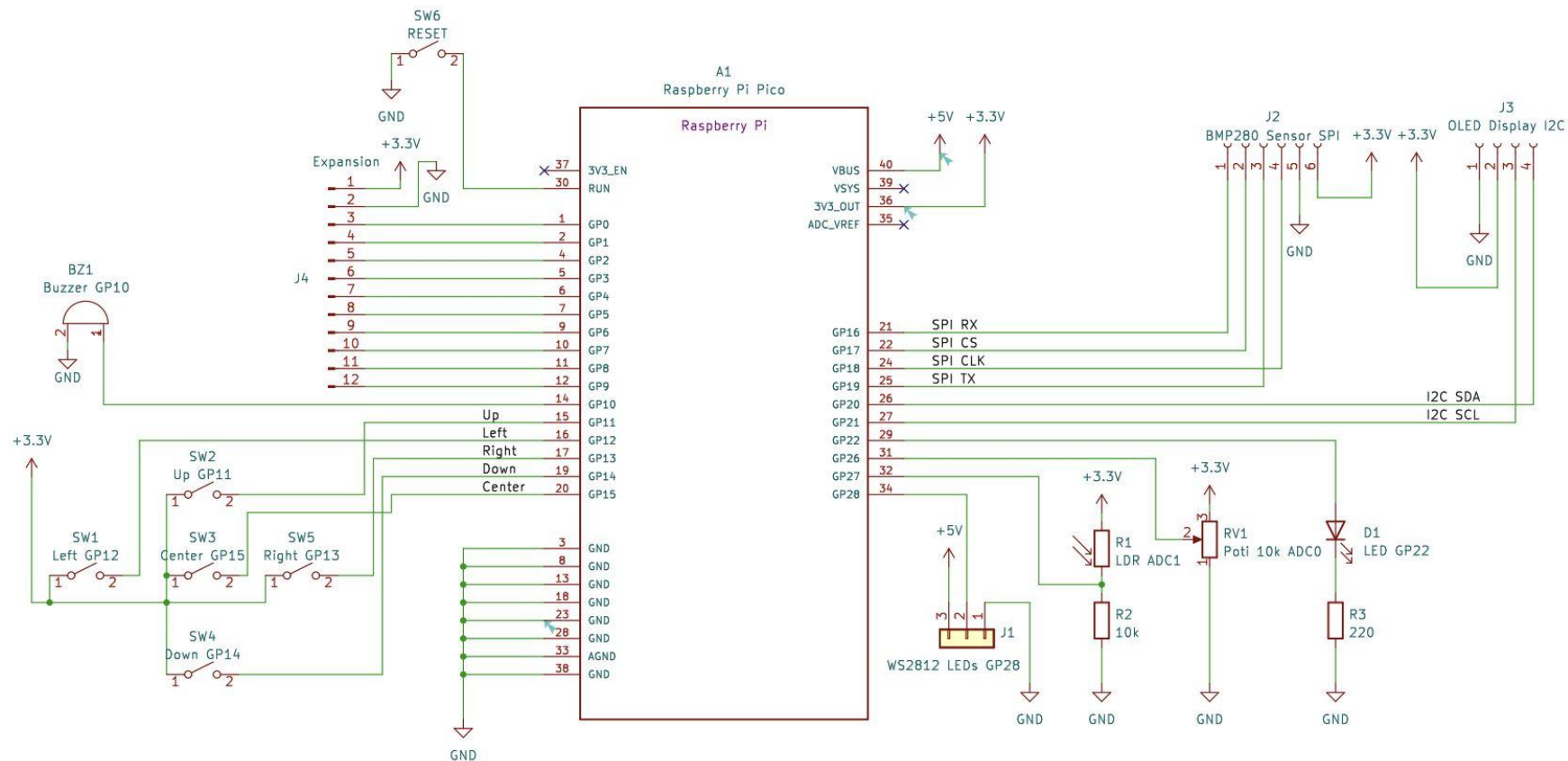
Raspberry Pi Pico W mit Wifi: Pinout



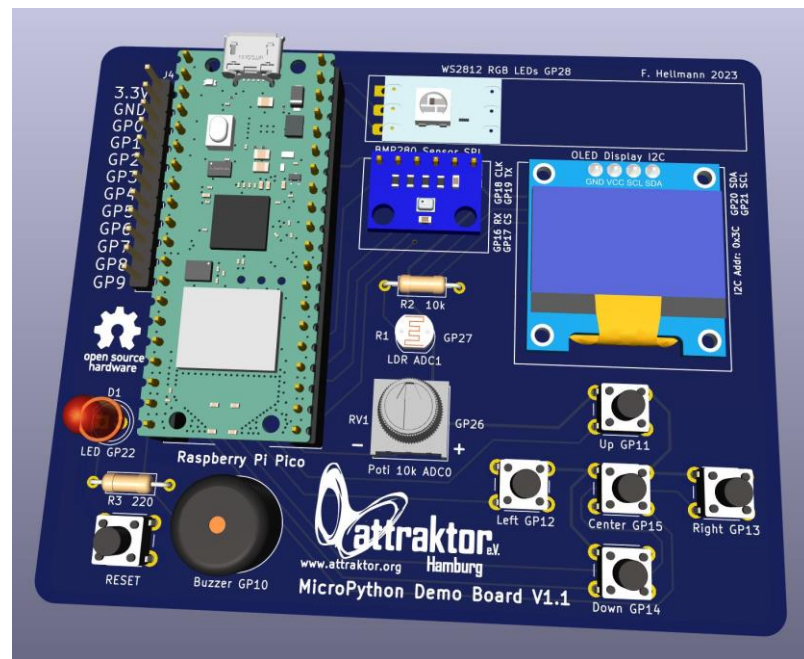
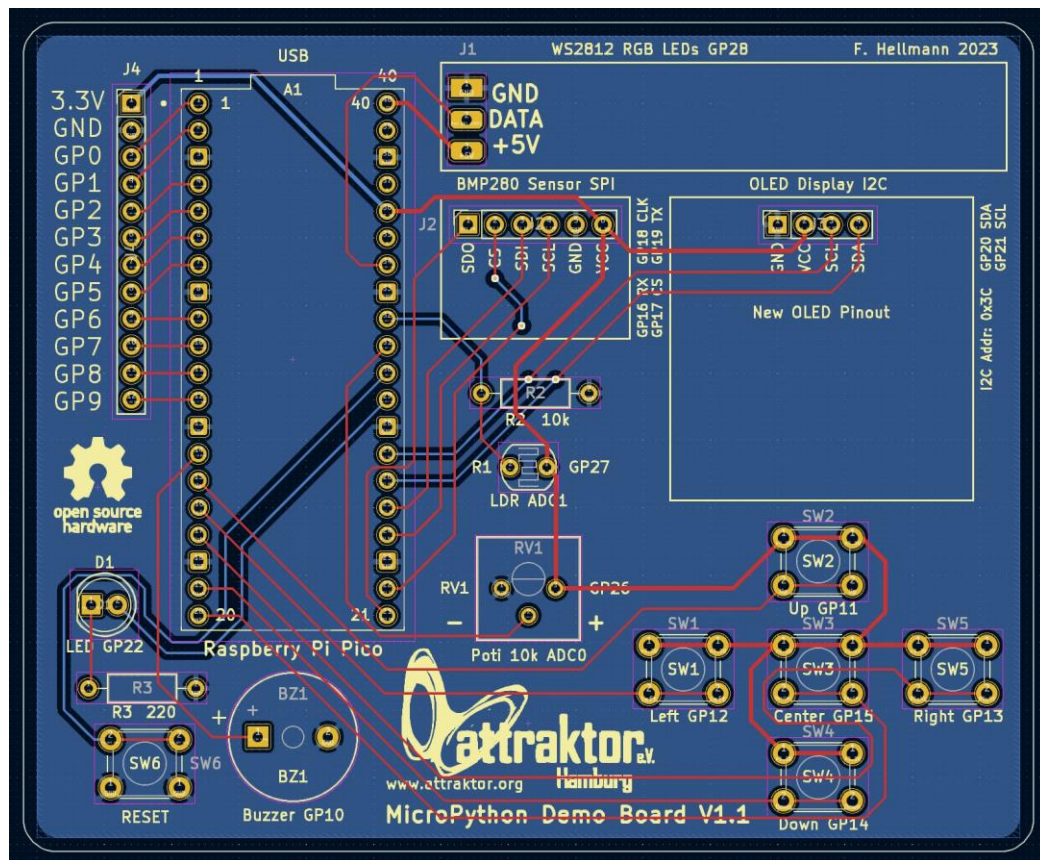
MicroPython Demoboard

- Der Schaltplan
- Die Platine
- Bestücken und Verlöten der Bauteile bei einem Kit
- Aufspielen der Firmware

Der MicroPython Demoboard Schaltplan



Die MicroPython Demoboard Platine



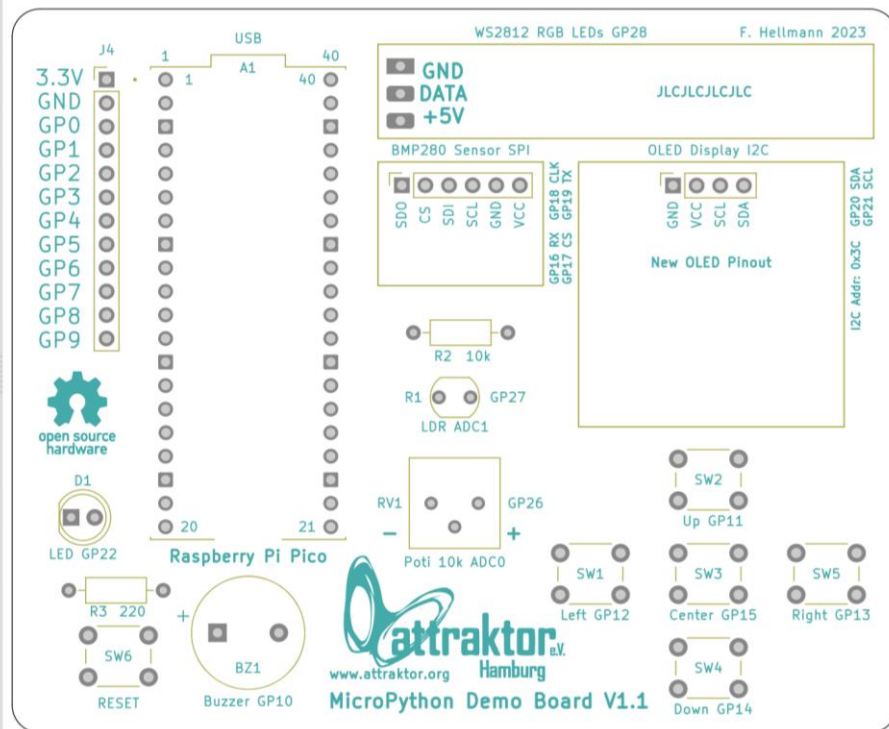
https://github.com/sandman72/Micropython_Demoboard

Der MicroPython Demoboard BOM - Bauteileliste

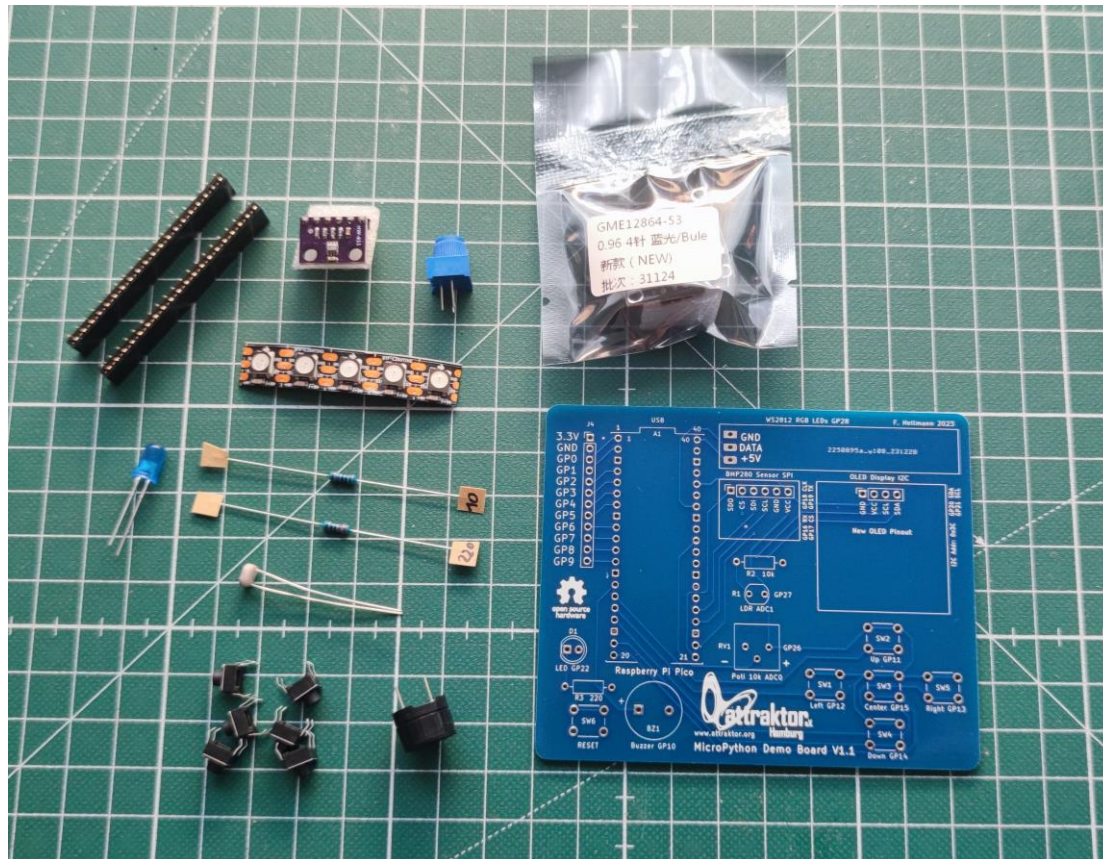
	Sour ced	Plac ed	References	Value	Footprint	Quantity
1	<input type="checkbox"/>	<input type="checkbox"/>	R1	LDR ADC1	R_LDR_5.1x4.3mm_P3.4mm_Ve rtical	1
2	<input type="checkbox"/>	<input type="checkbox"/>	R2	10k	R_Axial_DIN0207_L6.3mm_D2 .5mm_P10.16mm_Horizontal	1
3	<input type="checkbox"/>	<input type="checkbox"/>	R3	220	R_Axial_DIN0207_L6.3mm_D2 .5mm_P10.16mm_Horizontal	1
4	<input type="checkbox"/>	<input type="checkbox"/>	D1	LED GP22	LED_D5.0mm	1
5	<input type="checkbox"/>	<input type="checkbox"/>	SW1	Left GP12	SW_PUSH_6mm_H7.3mm	1
6	<input type="checkbox"/>	<input type="checkbox"/>	SW2	Up GP11	SW_PUSH_6mm_H7.3mm	1
7	<input type="checkbox"/>	<input type="checkbox"/>	SW3	Center GP15	SW_PUSH_6mm_H7.3mm	1
8	<input type="checkbox"/>	<input type="checkbox"/>	SW4	Down GP14	SW_PUSH_6mm_H7.3mm	1
9	<input type="checkbox"/>	<input type="checkbox"/>	SW5	Right GP13	SW_PUSH_6mm_H7.3mm	1
10	<input type="checkbox"/>	<input type="checkbox"/>	SW6	RESET	SW_PUSH_6mm_H7.3mm	1
11	<input type="checkbox"/>	<input type="checkbox"/>	A1	Raspberry Pi Pico	Raspberry_Pi_Pico_WH Dual 20pin Sockets	1
12	<input type="checkbox"/>	<input type="checkbox"/>	BZ1	Buzzer GP10	Buzzer_12x9.5RM7.6	1
13	<input type="checkbox"/>	<input type="checkbox"/>	RV1	Poti 10k ADC0	Potentiometer_Bourns_3386 P_Vertical	1
14	<input type="checkbox"/>	<input type="checkbox"/>	J1	WS2812 RGB LEDs GP28	LED Strip 100Leds/m 5pcs	1
15	<input type="checkbox"/>	<input type="checkbox"/>	J2	BMP280 Sensor SPI	BMP280 I2C/SPI 6pin Module	1
16	<input type="checkbox"/>	<input type="checkbox"/>	J3	OLED Display I2C	OLED 1306 0,96 4pin Module	1
17	<input type="checkbox"/>	<input type="checkbox"/>	J4	Erweiterung	PinHeader_1x12_P2.54mm_Ve rtical	1

Die Bauteilliste gibt es auch interaktiv unter:

https://github.com/sandman72/Micropython_Demoboard



Das MicroPython Demoboard löten

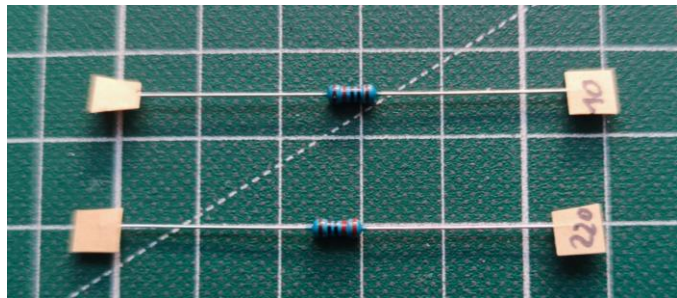


Das Demoboard und die Bauteile, die wir löten werden.

Das MicroPython Demoboard löten



R1



R2

R3

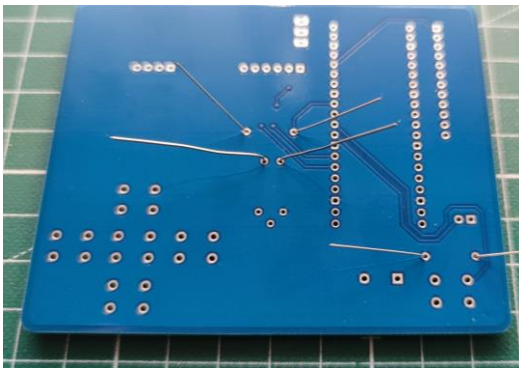
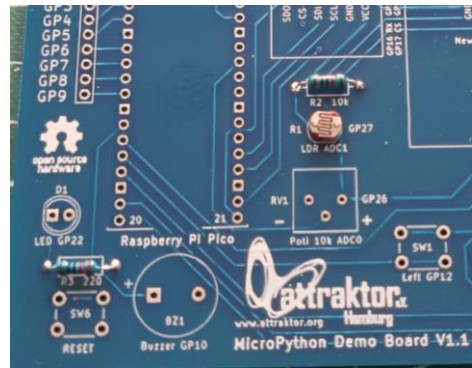
Als erstes verlöten wir die Widerstände R1, R2 und R3

R1 ist der helligkeitsabhängige Widerstand (LDR)

R2 ist der 10K Ohm Widerstand (10 Kilo Ohm)
(Farbcode: braun - schwarz - schwarz - rot)

R3 ist der 220 Ohm Widerstand
(Farbcode:: rot - rot - schwarz - schwarz)

Dazu biegen wir die Beinchen von R2 und R3 um, stecken die Widerstände in die passenden Löcher und biegen die Beinchen auf der Rückseite leicht um.



4 Ringe

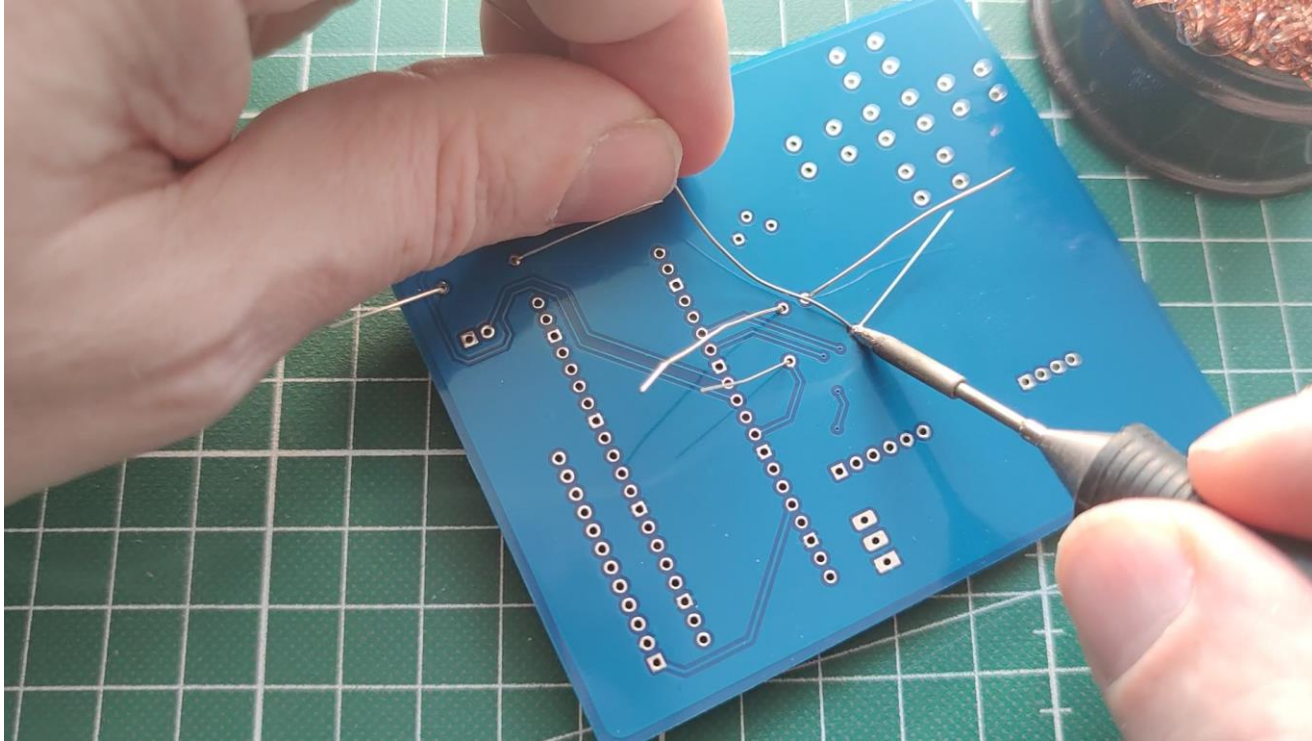
1.000 Ω
 $\pm 5 \%$

Farbe	1. Ring	2. Ring	3. Ring	Multiplikator	Toleranz
Schwarz	0	0	0	$\times 1 \Omega$	
Braun	1	1	1	$\times 10 \Omega$	$\pm 1 \%$
Rot	2	2	2	$\times 100 \Omega$	$\pm 2 \%$
Orange	3	3	3	$\times 1.000 \Omega$ (1 k Ω)	
Gelb	4	4	4	$\times 10.000 \Omega$ (10 k Ω)	$\pm 0,5 \%$
Grün	5	5	5	$\times 100.000 \Omega$ (100 k Ω)	$\pm 0,25 \%$
Blau	6	6	6	$\times 1.000.000 \Omega$ (1 M Ω)	$\pm 0,1 \%$
Lila	7	7	7	$\times 10.000.000 \Omega$ (10 M Ω)	
Grau	8	8	8		$\pm 0,05 \%$
Weiß	9	9	9		
Gold				$\times 0,1 \Omega$	$\pm 5 \%$
Silber				$\times 0,01 \Omega$	$\pm 10 \%$

5 Ringe

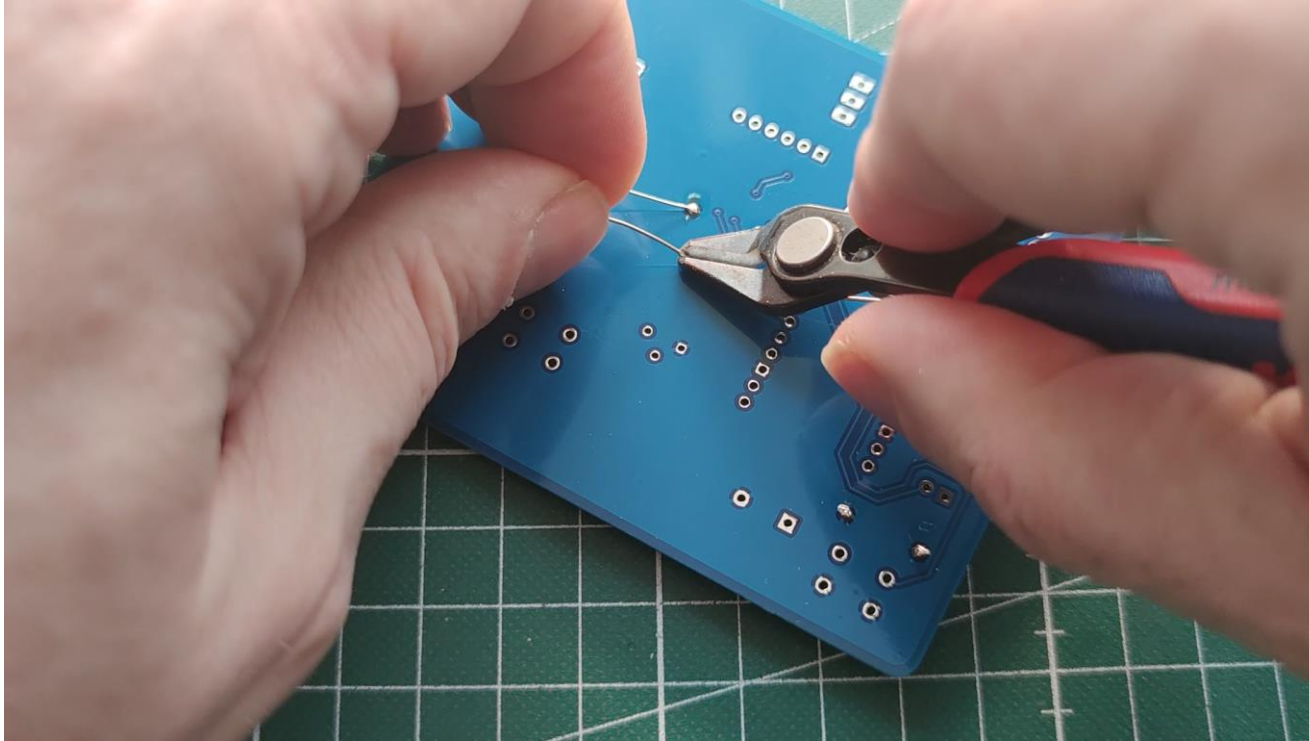
2.700 Ω
 $\pm 1 \%$

Das MicroPython Demoboard lötén



Dann verlöten wir die Widerstände.

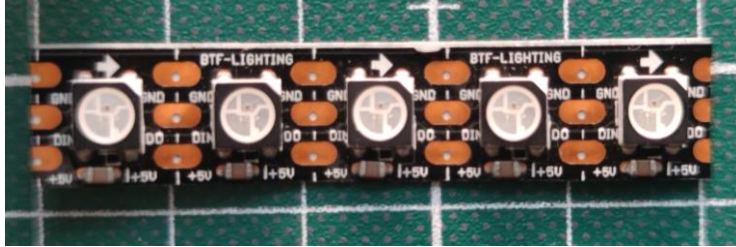
Das MicroPython Demoboard lötén



Und schneiden die Beinchen ab.

Dabei nur das Beinchen und nicht die Lötstelle mit abschneiden.

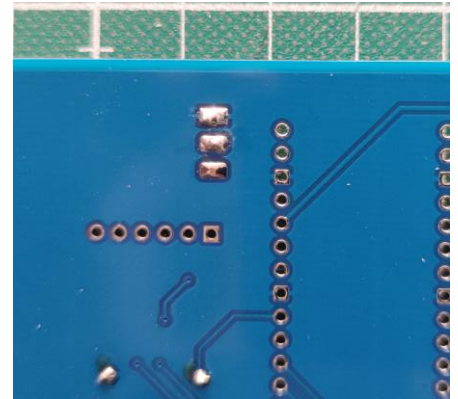
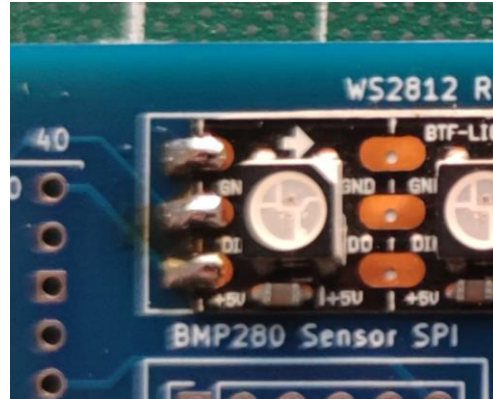
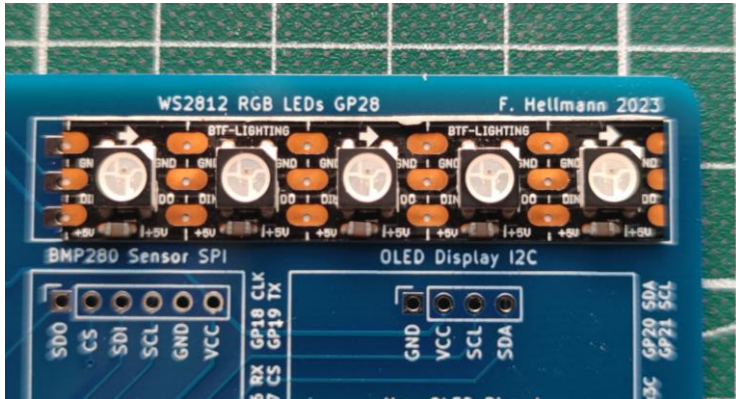
Das MicroPython Demoboard löten



Als nächstes verlöten wir den WS2812 LED Streifen

Hier ist wichtig, dass der **Pfeil nach rechts** zeigt bzw. Pin **DO rechts** ist!

Man zieht den Schutzfilm auf der Rückseite der LEDs ab und klebt den LED Streifen passend auf die Platine, dass die Lötäugen schön übereinander liegen. Dann verlötet man die drei Pads und guckt sich danach auch die Rückseite an, dass die Lötflächen auch gut aussehen.

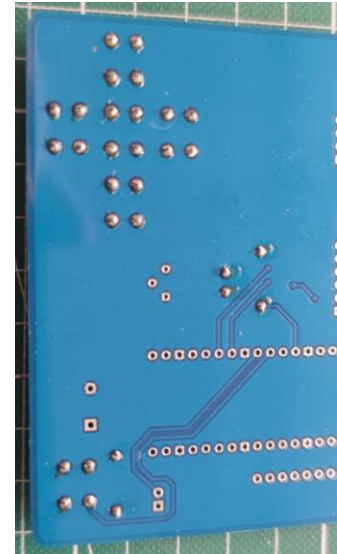
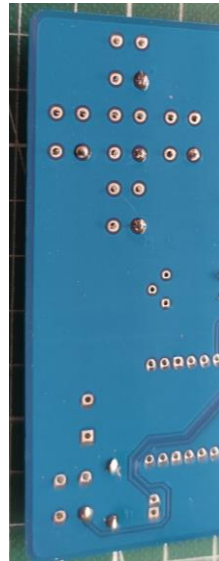


Das MicroPython Demoboard lötén

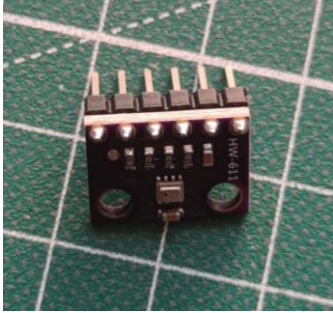


Dann kommen die 6 Taster

Einmal gucken, dass die Beinchen sinnvoll ausgerichtet sind und dann werden die Taster in die Platine gesteckt. Dann jeweils ein Beinchen anlöten und kontrollieren, dass die Taster schön auf der Platine aufliegen. Wenn alles passt, dann die restlichen Beinchen verlöten.



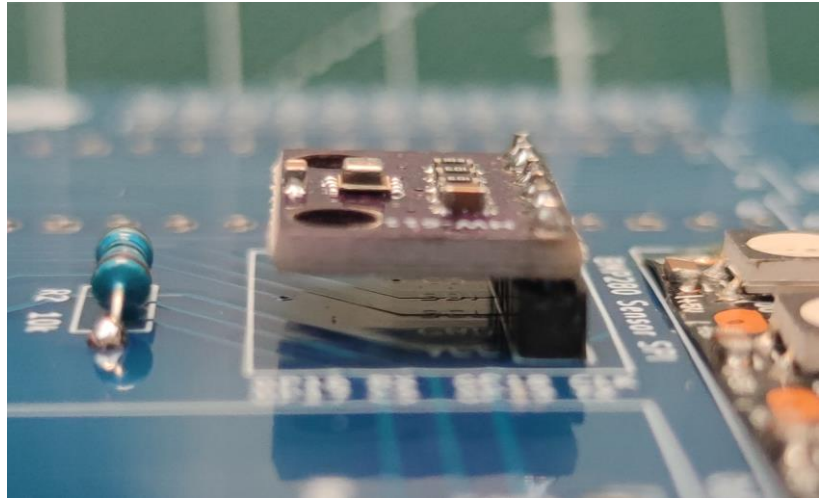
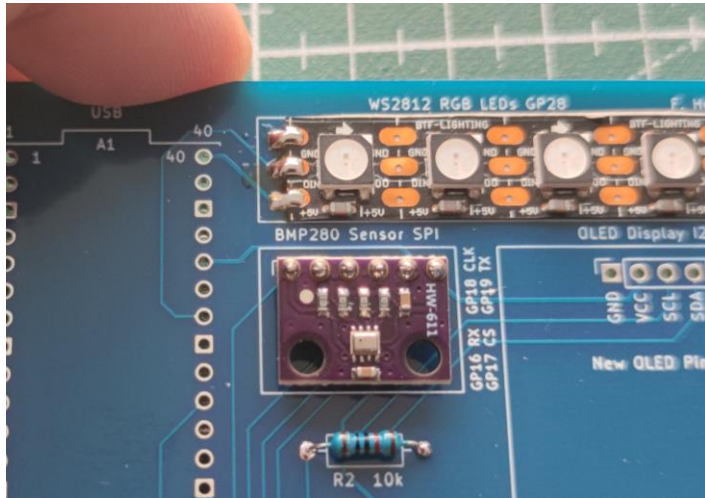
Das MicroPython Demoboard bestücken



Nun folgt der Klima Sensor BMP280

Dieser wird auch passend in die Platine gesteckt und an einem Pin verlötet. Dann die Ausrichtung kontrollieren und wenn alles passt die restlichen Pins anlöten.

Danach werden die überstehenden Pins abgeknipst. Vorsicht! Die sind etwas dicker und können beim Abschneiden pieken.



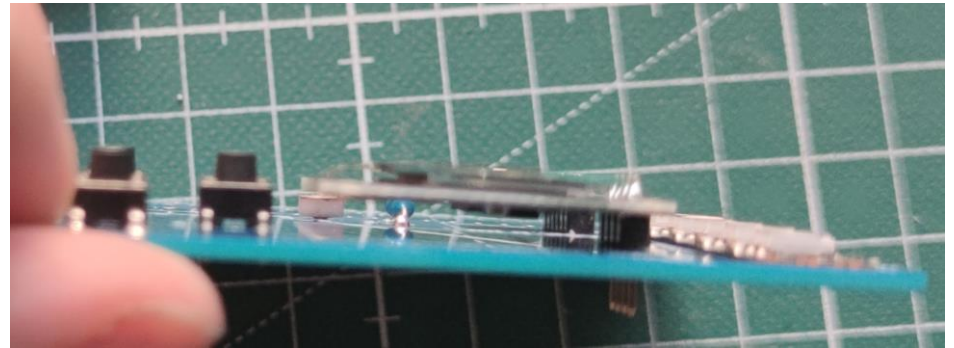
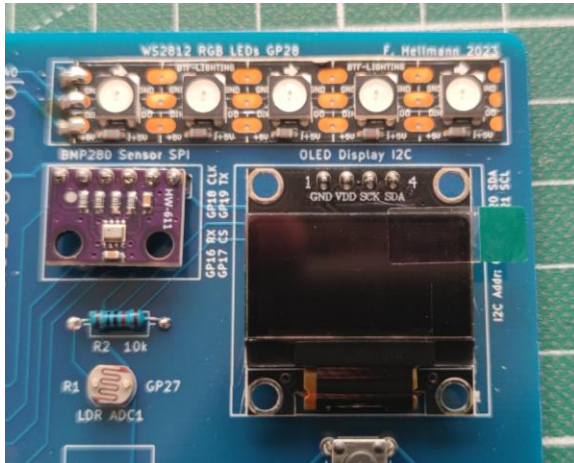
Das MicroPython Demoboard bestücken



Jetzt folgt das OLED Display

Dieses wird auch passend in die Platine gesteckt und an einem Pin verlötet. Dann die Ausrichtung kontrollieren und wenn alles passt die restlichen Pins anlöten.

Danach werden die überstehenden Pins abgeknipst. Vorsicht! Die sind etwas dicker und können beim Abschneiden pieken.



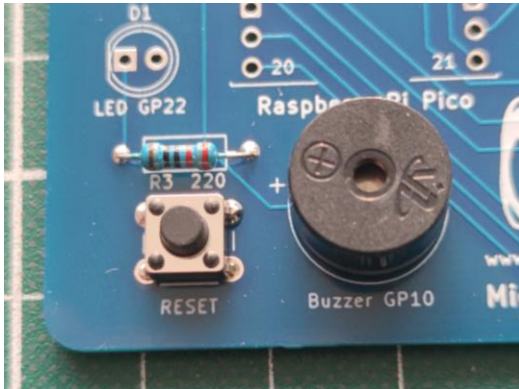
Das MicroPython Demoboard bestücken



Nun bestücken wir den Summer

Der Summer hat eine Plus + Markierung, die wichtig ist. Diese passend in die Platine stecken und an einem Pin verlötet. Dann die Ausrichtung kontrollieren und wenn alles passt den anderen Pin anlöten.

Danach werden die überstehenden Pins abgeknipst. Vorsicht! Die sind etwas dicker und können beim Abschneiden pieken.



Das MicroPython Demoboard bestücken



Anschließend bestücken wir die Leuchtdiode LED

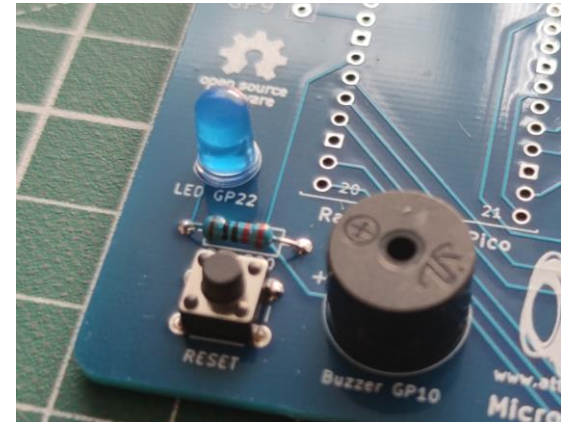
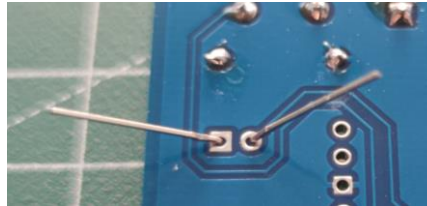
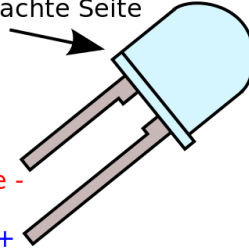
Die Leuchtdiode hat eine Einbaurichtung, die wichtig ist. Die flache Seite mit dem **kurzen Beinchen** ist die sogenannte **Kathode** - und muss in das **quadratische Loch** gesteckt werden. Die Anode + ins Loch daneben.

Dann die beiden Beinchen leicht umbiegen, festlöten und abknipsen.

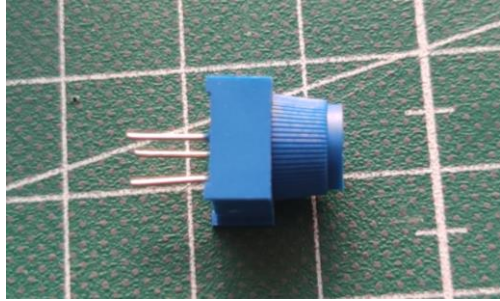
Abgeflachte Seite

Kathode -

Anode +



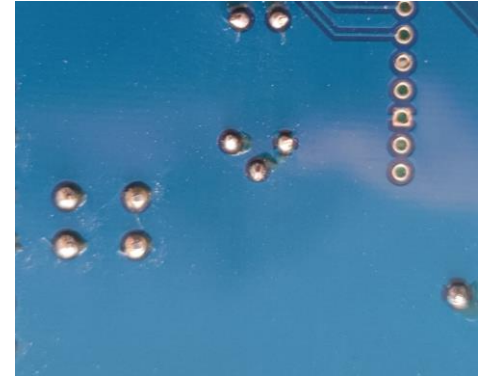
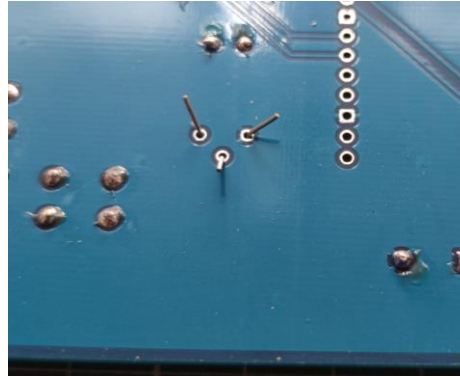
Das MicroPython Demoboard bestücken



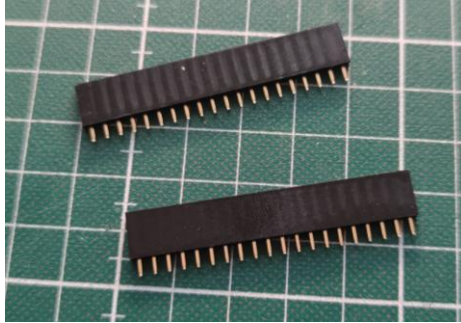
Jetzt ist das Potentiometer dran

Das Potentiometer ist ein einstellbarer Widerstand. Das Poti passend in die Platine stecken, die Beinchen leicht verbiegen und wenn alles passt die Pins anlöten.

Das Potentiometer stellt eine Spannung ein, die der Pi Pico an einem analogen Eingang auslesen kann. $0V = 0$ $1.8V = 512$ $3.3V = 1023$



Das MicroPython Demoboard bestücken

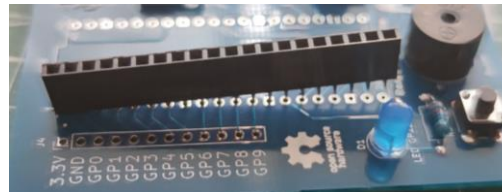
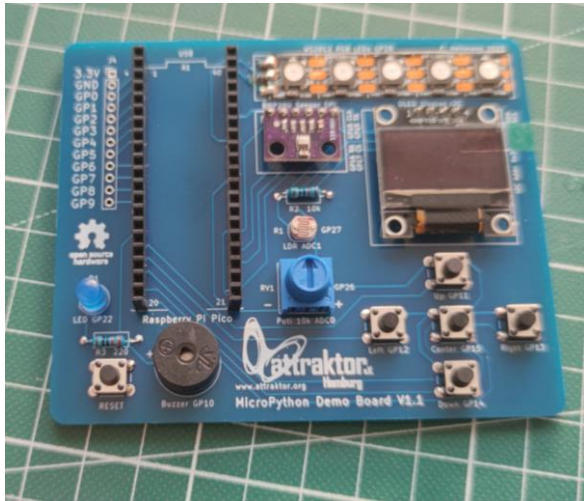


Als letztes kommen die beiden Sockelleisten dran

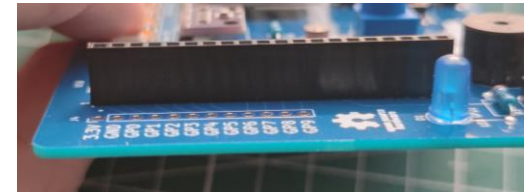
Achtung! Hier ist die Ausrichtung, **gerade** und auf der Platine **auflegend**, sehr wichtig! Sonst passt der Pi Pico am Ende nicht in den Sockel. Zur besseren Ausrichtung kann man auch die Leisten auf den Pi Pico stecken und dann die Sockelleisten auf der Platine verlöten.

Erst wird eine Sockelleiste passend in die Platine gesteckt und an einem Pin am Anfang verlötet. Dann die Ausrichtung kontrollieren und wenn alles passt den Pin am Ende anlöten und nochmal kontrollieren. Dann mit der zweiten Sockelleiste ebenso verfahren.

Wenn alles passt dann die restliche Pins verlöten.

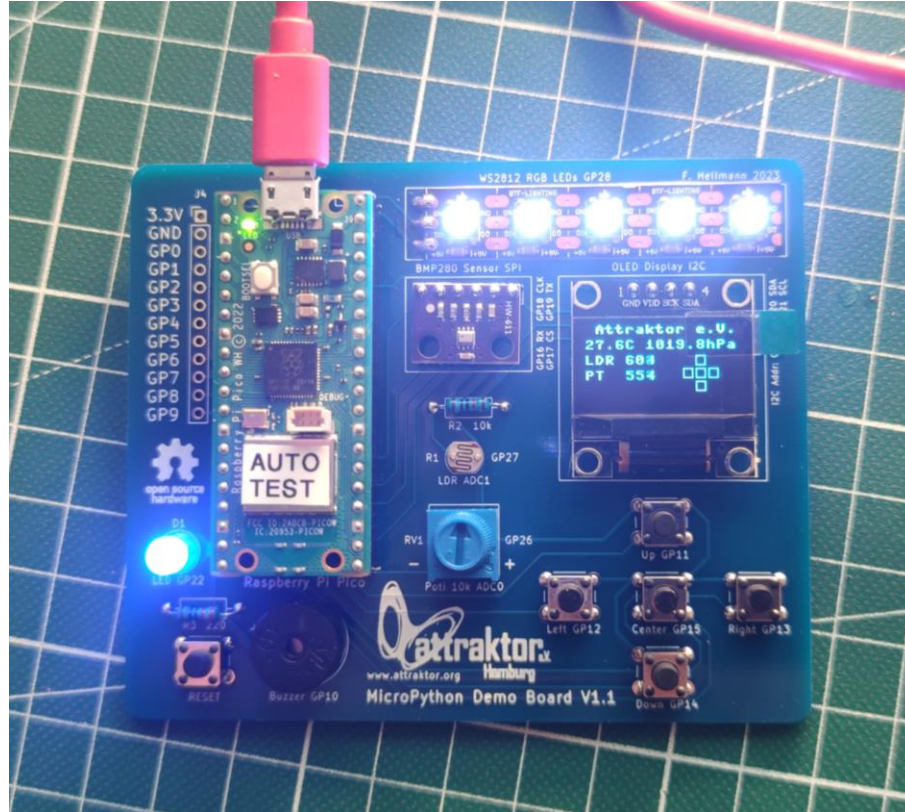


So bitte nicht!



So bitte!

Das MicroPython Demoboard bestücken



Fertig! Platine checken und dann testen!

Einmal nochmal alle Lötstellen kontrollieren. Gucken, ob noch kleine Lötspatzer an der Platine hängen. Schauen, ob einem alles gefällt, es keine kalten oder kurz geschlossenen Lötstellen gibt.

(Wenn der Pi Pico nicht mitgeliefert wurde, muss noch eine Firmware aufgespielt werden. Siehe nächste Seite: Firmware aufspielen)

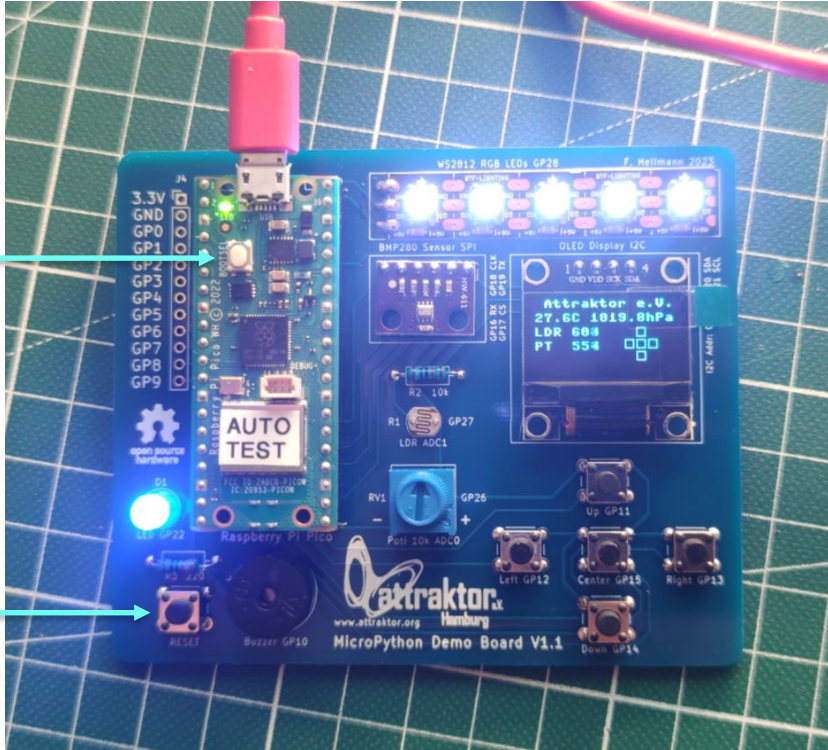
Dann kann der Pi Pico eingesteckt und mit dem USB Kabel mit Strom versorgt werden. Jetzt sollte das Demo-Menü gestartet werden. Unter dem Punkt Board Test kann nochmal man die Funktionalität der Bauteile checken.

Im besten Falle ist das Demoboard dann:

Fertig!

Die Demoboard Firmware aufspielen

1. Bootsel drücken
und gedrückt
halten



2. Reset kurz
drücken und
loslassen

Installation der Firmware:

Wenn der Pi Pico noch keine passende Firmware für das MicroPython Demoboard hat, oder man die originale Firmware neu aufspielen möchte, geht das so:

Den eingesteckten Pi Pico per USB-Kabel mit seinem Computer verbinden und dann die kleine weiße Bootsel-Taste drücken und gleichzeitig kurz einmal die Reset-Taste. Jetzt sollte auf dem Computer ein Laufwerk erscheinen (RPI-RP2) und man kann die Firmware Datei einfach per Drag&Drop draufziehen. Damit wird sie dann fest gespeichert.

Die Firmwares gibt es hier:

https://github.com/sandman72/Micropython_Demoboard

MicroPython Einführung

- Was ist MicroPython?
- Erste Versuche mit Thonny

Was ist MicroPython?

MicroPython ist eine einfachere Version der Programmiersprache Python, die speziell für Mikrocontroller entwickelt wurde. Es deckt die wichtigsten Funktionalitäten von Python ab, ist aber für den kleineren Speicher von Mikrocontrollern ausgelegt. Mit MicroPython wird auch die Hardware des Pi Picos komplett abstrahiert und kann mit einem einfachen `import machine` geladen werden.

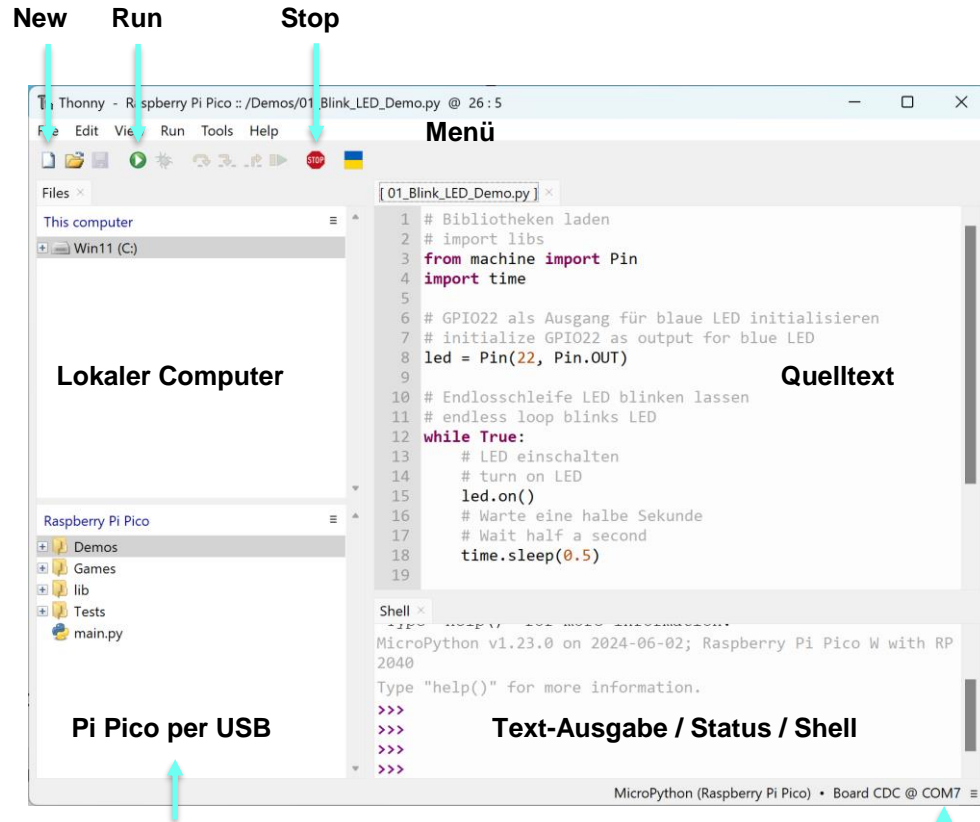
Auffälligster Unterschied zu anderen Programmiersprachen ist das Einrücken von Gruppierungen mit Tabs, anstelle von Klammern oder ähnlichem. Das sehen wir in den Beispielen.

Standardmäßig ist der Raspberry Pi Pico für die Programmierung mit C und C++ eingerichtet. Wenn der Pico mit MicroPython programmiert werden soll, dann muss MicroPython zuerst auf dem Board installiert werden. Der Vorgang ist einmalig und besteht darin, eine Firmware Datei aus dem Internet herunterzuladen und auf den Pico zu kopieren. Dies ist auf der Seite: *Firmware aufspielen* beschrieben.

Aber auch Editoren wie Thonny können dies, allerdings sind dort nicht alle benötigten Bibliotheken für dieses MicroPython Demoboard mit dabei.

Weitere Infos gibt es in der offiziellen Dokumentation:
<https://docs.micropython.org/en/latest/rp2/quickref.html>

Erste Versuche mit Thonny



Tipp: Unter Menü **View** den Punkt **Files** anschalten

USB Port auswählen

Die **Thonny** Python IDE verfügt über alle erforderlichen Funktionen und Bedienelemente, um den Pi Pico einfach mit MicroPython programmieren zu können.

Aufteilung der Entwicklungsumgebung

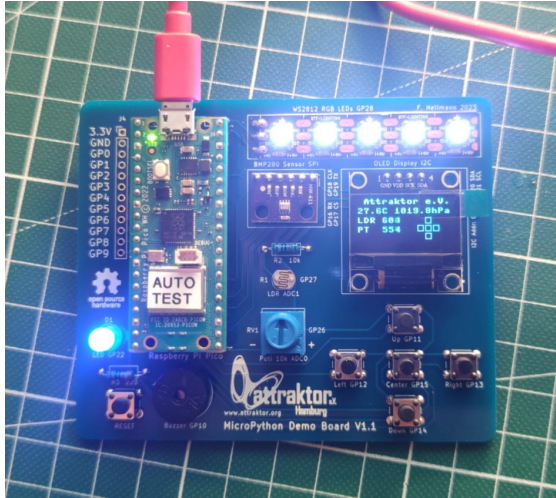
1. Menü-Leiste zum Bedienen und Steuern
2. Eingabefeld für den Quelltext
3. Ausgabefeld für die Textausgabe und Status-Infos
4. Auswahl des Mikrocontrollers und Python Version

Wichtige Funktionen der Menü-Leiste

- Neue Datei erstellen (New)
- Datei öffnen (Load)
- Datei speichern (Save)
- Programm ausführen (Run)
- Programm stoppen (Stop)

Thonny gibt es hier: <https://thonny.org/>

Erste Versuche mit Thonny



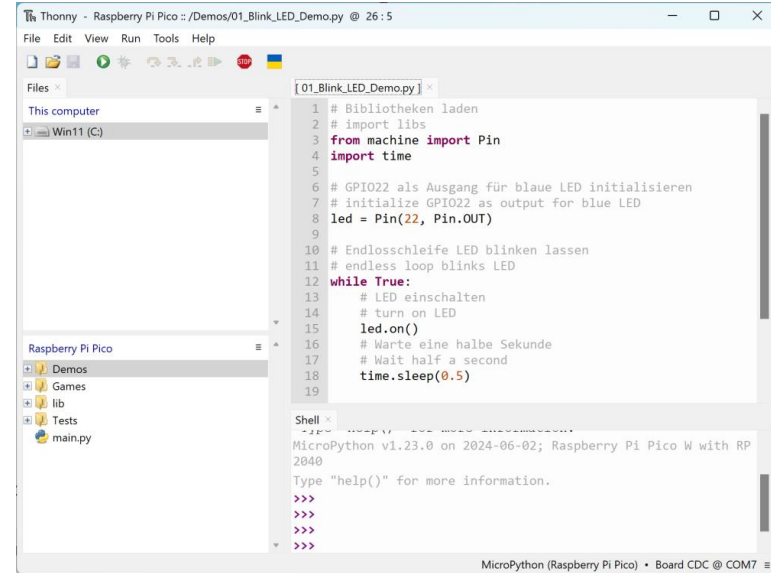
per USB anschließen

from machine import Pin
import time



led = Pin(22, Pin.OUT)

while True:
 led.on()
 time.sleep(0.5)
 led.off()
 time.sleep(0.5)

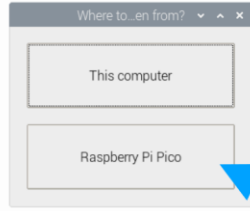
Quelltext eingeben



mit Thonny das Programm starten

1. Pi Pico mit dem USB Kabel am Computer anschließen und in Thonny unten rechts gucken, dass er erkannt wurde.
2. Den  Stop Button in Thonny drücken, damit der Pi Pico ansprechbar wird.
3. Dann den Quelltext eingeben.
4. Den  Start Button in Thonny drücken, damit das Programm gestartet wird.
5. Im besten Falle, läuft das Programm. Sonst im Status Bereich gucken, wo es hakt.

Erste Versuche mit Thonny




Programmcode mit der Dateierdung „.py“ auf dem Pico speichern.

Wenn man das Programm auf dem Pi Pico dauerhaft speichern möchte, dann kann man das mit **Save** und man wählt den Raspberry Pi Pico als Ziel aus. Am besten gibt man dem Programm einen sinnvollen Namen und die Endung **.py** damit der Pi Pico erkennt, dass das ein MicroPython Programm ist.

Es gibt bestimmte Dateien die man nur ändern sollte, wenn man weiß was man tut: main.py und die mitgelieferten Bibliotheken im lib Verzeichnis gehören dazu.

Wenn ein Programm nicht mehr läuft, leuchten z.B. die Leuchtdioden immer noch, die beim letzten Programmschritt geleuchtet haben. Es bleiben also „Zustände“ im Speicher erhalten. Wenn man dann ein anderes Programm startet, dann leuchten manche LEDs einfach weiter, ob wohl man den Stop-Button oder die Reset-Taste gedrückt hat.


Es empfiehlt sich, den Raspberry Pi Pico bei einem neuen Programm vorher einmal auszustecken und neu einzustecken. Dabei werden alle Rückstände des alten Programms aus dem Speicher gelöscht.

Hinweis: Nach dem erneuten Einstecken muss die Verbindung zum Raspberry Pi Pico durch Klicken auf den  Stop-Button zurückgesetzt werden.

Erste Versuche mit Thonny

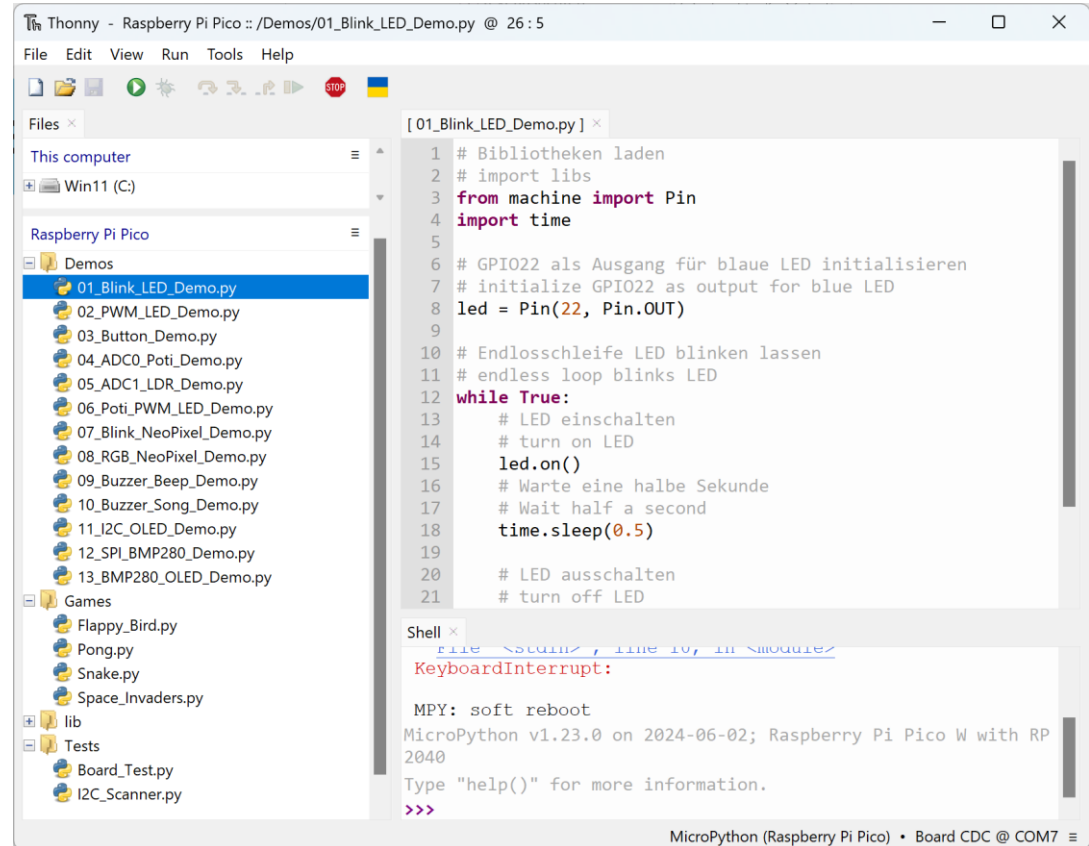
Mitgelieferte Beispiele

Damit man nicht alle Beispiele abtippen muss, gibt es auf dem Pi Pico einen **Demos** Ordner, der die folgenden Programm-Beispiele enthält.

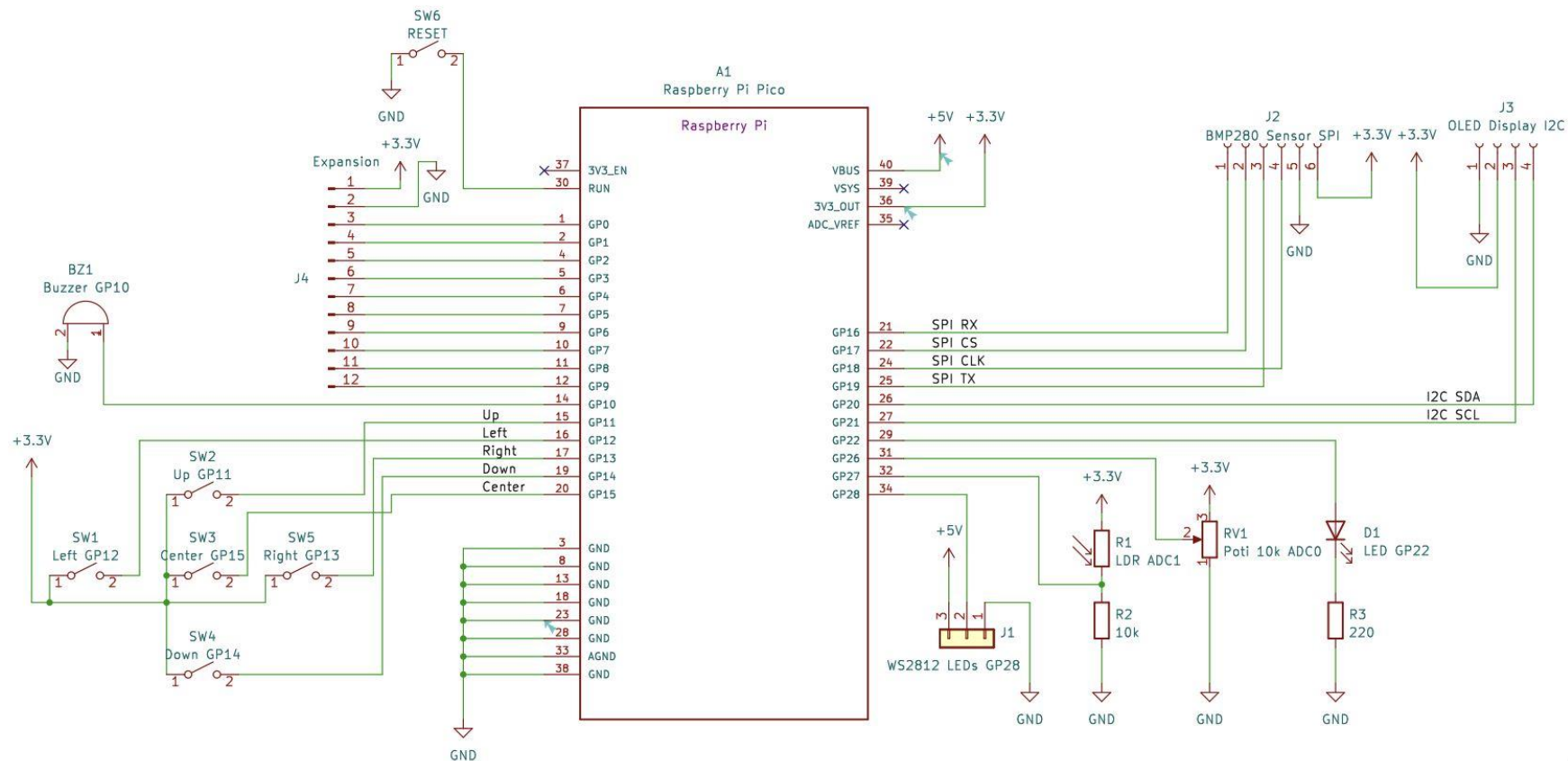
Dort fangen wir mit dem ersten Beispiel einfach mal an und laden es mit einem Doppelklick in den Editor. Wenn wir jetzt den  Start-Button drücken, sollte die blaue LED auf unserem Board im Halbsekundentakt aufleuchten.

Wenn wir nun den  Stop-Button drücken, bleibt der letzte Zustand der LED erhalten und das Programm ist gestoppt.

Tipp: Wenn links keine Dateien eingeblendet werden, unter Menü **View** den Punkt **Files** anschalten



Der MicroPython Demoboard Schaltplan



Beispiel 1 – LED blinken lassen

```
# Bibliotheken laden
from machine import Pin
import time

# Blaue LED and GPIO22 als Ausgang initialisieren
led = Pin(22, Pin.OUT)

# Endlosschleife LED blinken lassen
while True:
    # LED einschalten
    led.on()
    # Warte eine halbe Sekunde
    time.sleep(0.5)

    # LED ausschalten
    led.off()
    # Warte eine halbe Sekunde
    time.sleep(0.5)
```

In diesem Beispiel lassen wir eine LED blinken.

Auf der Platine ist die LED mit GP22 markiert und auf dem Schaltplan sehen wir rechts unten, dass dort der GPIO Pin 22 mit der LED verbunden ist. GPIO steht für General Purpose I/O, also allgemeiner Ein- und Ausgabe-Pin. Das heißt für uns, das wir mit dem Pin 22 die LED steuern können.

Als Erstes werden die Bibliotheken geladen, die MicroPython ermöglichen, mit der Hardware zu sprechen und auch zeitliche Verzögerungen zu erlauben.

Dann initialisieren wir den GPIO Pin 22 mit Pin.OUT als Ausgang.

In der *while* Schleife wird dann die LED eingeschaltet, eine halbe Sekunde gewartet, die LED wieder ausgeschaltet und wieder eine halbe Sekunde gewartet.

Die *while* Schleife wiederholt die eingerückten Befehle endlos, da sie mit der Bedingung *True*, also Wahr, nie abbricht.

Beispiel 2 – LED mit PWM ansteuern

```
# Bibliotheken laden
from machine import Pin, PWM
from time import sleep_ms

# Blaue LED an GPIO22 mit PWM initialisieren
pwm = PWM(Pin(22))

# Frequenz (Hz) einstellen
pwm.freq(4000)

# Endlosschleife LED Helligkeit hoch/runterfahren
while True:
    # Helligkeit hoch fahren
    for duty_cycle in range(0, 65536, 129):
        pwm.duty_u16(duty_cycle)
        sleep_ms(3)

    # Helligkeit runter fahren
    for duty_cycle in range(65536, 0, -129):
        pwm.duty_u16(duty_cycle)
        sleep_ms(3)
```

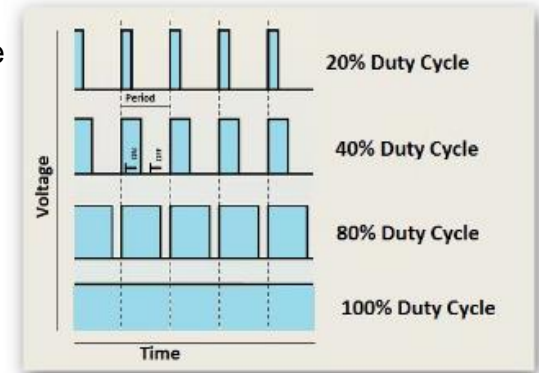
In diesem Beispiel lassen wir eine LED langsam auf- und abblenden.

Wir haben im ersten Beispiel gesehen, dass wir die LED mit Pin 22 steuern können. Hier tun wir dies aber nicht nur mit einem einfachen An/Aus, sondern tun dies mit Puls-Weiten-Modulation. Dabei hilft uns der Pi Pico mit seinem Hardware-PWM die Helligkeit einer LED zu steuern. Das heißt für uns, dass wir dem Pi Pico nur noch sagen müssen, wie Hell wir die LED haben möchten.

Dies können wir mit der Funktion `pwm.duty_u16` der wir einen Wert zwischen 0 und 65535 übergeben.

0 = 0% , 32768 = 50%, 65535 = 100%

Die Frequenz legt fest, wie oft pro Sekunde, dies passieren soll. Zu niedrige Frequenz würde flackern.



Beispiel 3 – Taster abfragen

```
# Bibliotheken laden
from machine import Pin
import time

# GPIO Pins als Eingänge mit Pull_Down initialisieren
button_up    = Pin(11, Pin.IN, Pin.PULL_DOWN)
button_left  = Pin(12, Pin.IN, Pin.PULL_DOWN)
button_right = Pin(13, Pin.IN, Pin.PULL_DOWN)
button_down  = Pin(14, Pin.IN, Pin.PULL_DOWN)
button_center = Pin(15, Pin.IN, Pin.PULL_DOWN)

# Endlosschleife mit Abfrage der Buttons
while True:
    if button_up.value() == 1:
        print("Up Button is Pressed")
    if button_left.value() == 1:
        print("Left Button is Pressed")
    if button_right.value() == 1:
        print("Right Button is Pressed")
    if button_down.value() == 1:
        print("Down Button is Pressed")
    if button_center.value() == 1:
        print("Center Button is Pressed")
    time.sleep(0.1)
```

In diesem Beispiel benutzen wir die Taster zur Eingabe.

Auf der Platine sind die Taster mit GP11 bis GP 15 markiert und auf dem Schaltplan sehen wir links unten, wie diese mit dem Pi Pico verbunden sind. Wenn ein Taster gedrückt wird, wird der entsprechende Eingang des Pi Pico mit 3.3V verbunden.

Als Erstes werden wieder die Bibliotheken geladen.

Dann initialisieren wir die Taster alle mit **Pin.IN** als Eingang und mit der Option **Pin.PULL_DOWN**. Dieser Pull Down zieht die Eingänge ganz leicht auf Masse, so lange kein Taster gedrückt wird. Das garantiert uns, dass wir nur zwei zulässige Zustände haben, entweder 3.3V = Taster gedrückt oder 0V = nix los.

In der *while* Endlosschleife werden dann die Taster mit der Funktion `value()` abgefragt und wenn einer gedrückt, also 1 ist, wird uns dies im Status Fenster mitgeteilt. Die `Print` Funktion erlaubt uns die Kommunikation vom Pi Pico zur Außenwelt, in diesem Fall zu unserem Thonny Status-Bereich.

Beispiel 4 – Wert des Potentiometers anzeigen

```
# Bibliotheken laden
import machine
import utime

# Analog/Digital Wandler ADC0 GPIO 26 initialisieren
# an ADC 0 hängt das einstellbare Potentiometer RV1
potentiometer = machine.ADC(26)

# Endlosschleife Poti auslesen und Wert anzeigen
while True:
    print("Poti: " + str(potentiometer.read_u16()))
    utime.sleep(1)
```

In diesem Beispiel nutzen wir einen analogen Eingang.

Normalerweise haben Computer Probleme mit analogen Werten und bevorzugen digitale Nullen und Einsen. Um aus unserer analogen Welt in die digitale zu kommen, gibt es sogenannte ADC – Analog/Digital Converter. Davon ist bei unserem Demoboard der ADC0 mit einem Drehregler, auch Potentiometer genannt, verbunden. Durch diesen einstellbaren Spannungsteiler können wir Spannungen von 0V bis 3.3V einstellen und uns diese als digitale Werte anzeigen lassen.

Als Erstes werden wieder die Bibliotheken geladen.

Dann initialisieren den Eingang GPIO 26 als ADC0 und lassen uns in der *while* Endlosschleife die abgefragten `potentiometer.read_u16()` Werte im Status-Bereich in Thonny anzeigen. Wenn wir nun das blau Potentiometer drehen, dann ändern sich die Werte einmal pro Sekunde.

In der Print Funktion setzen wir hier einen Text/String aus verschiedenen Werten zusammen. Daher müssen wir den Zahlenwert erst mit der `str()` Funktion umwandeln.

Beispiel 6 – LED Helligkeit mit Potentiometer einstellen

```
# Bibliotheken laden
from machine import Pin, PWM
from time import sleep
```

```
# Blaue LED an GPIO22 mit PWM initialisieren
pwm = PWM(Pin(22))
```

```
# Frequenz (Hz) einstellen
pwm.freq(4000)
```

```
# Analog/Digital Wandler ADC0 GPIO 26 initialisieren
# an ADC 0 hängt das einstellbare Potentiometer RV1
potentiometer = machine.ADC(26)
```

```
# Endlosschleife ADC0 auslesen und damit LED PWM Helligkeit
einstellen
while True:
    pwm.duty_u16(potentiometer.read_u16())
    sleep(0.005)
```

In diesem Beispiel fügen wir die gelernten Dinge einmal zusammen und regeln mit dem Potentiometer RV1 die Helligkeit der blauen LED.

In den ganzen Beispielen vorher gab es die verschiedensten Arten, den Pi Pico warten zu lassen. Hier mit *sleep* wieder eine neue.

Beispiel 7 & 8 – LED Streifen ansprechen

```
# Bibliotheken laden
from machine import Pin
from neopixel import NeoPixel
from time import sleep_ms

# NeoPixel/WS2812 LED Streifen mit GPIO 28 initialisieren
neopin = Pin(28, Pin.OUT)
# Anzahl der LEDs
leds = 5
# Helligkeit: 0 bis 255
brightness = 30
# Geschwindigkeit (Millisekunden)
speed = 50
# Initialisierung WS2812/NeoPixel
pixels = NeoPixel(neopin, leds)

# Endlosschleife LEDs abwechselnd einschalten
while True:
    for i in range (leds):
        # Nächste LED einschalten (R,G,B)
        pixels[i] = (brightness, brightness, brightness)
        pixels.write()
        # kurz warten
        sleep_ms(speed)
    # LED wieder zurücksetzen
    pixels[i] = (0, 0, 0)
```

In diesem Beispiel steuern wir den LED Streifen an.

Auf dem Demoboard ist ein intelligenter LED Streifen verbaut, den wir mit einer speziellen NeoPixel Bibliothek ansprechen können. Dieser LED Streifen hängt an Ausgang GPIO 28 welchen wir dann an die NeoPixel Bibliothek weitergeben, damit diese den Pin für uns steuert. Damit müssen wir uns um die Details der NeoPixel nicht kümmern und können einfach nur Werte für die rot, grün und blau Farbanteile übergeben.

Wir laden als erstes die Bibliotheken und legen den neopin GPIO 28 als Ausgang fest. Dann geben wir die Anzahl der LEDs an, wie hell und schnell unser blinken laufen soll und erzeugen uns ein NeoPixel Objekt, dass die Informationen zu unseren Pixeln enthält., z.B. wird der allererste Pixel mit `pixel[0] = (rot, grün, blau)` angesteuert. Python zählt also ab Null hoch.

In der Endlosschleife blinken wir dann jeweils einen Pixel kurz auf und dann den nächsten im nächsten Durchlauf der for Schleife.

Beispiel 8 erzeugt dann einen Regenbogen.

Beispiel 9 & 10 – Töne erzeugen

```
# Bibliotheken laden
from machine import Pin, PWM
from utime import sleep

# Buzzer an GPIO 10 mit PWM initialisieren
buzzer = PWM(Pin(10))

# Frequenz (Hz) einstellen
frequency = 1000
buzzer.freq(frequency)

# Einschalten für 0.3 Sekunden
buzzer.duty_u16(1000)
sleep(0.3)

# Ausschalten
buzzer.duty_u16(0)
```

In diesem Beispiel lassen wir den Buzzer tönen

Hier schauen wir uns nochmal den PWM Erzeuger an und nutzen diesen um Töne zu erzeugen. Hier nutzen wir die PWM Frequenz um die Höhe des Tons zu bestimmen und nutzen den Duty-Cycle nur zum Ein- und Ausschalten.

Den Duty-Cycle könnte man z. B. zum Einstellen der Lautstärke nutzen.

Beispiel 10 spielt uns dann ein kleines Liedchen.

Beispiel 11 – OLED Display „Hello World“

```
# Bibliotheken laden
from machine import Pin, I2C
from ssd1306 import SSD1306_I2C

# I2C Bus initialisieren
# GPIO 20 - SDA (Data)
# GPIO 21 - SDC (Clock)
i2c = I2C(0, sda=Pin(20), scl=Pin(21))

# OLED Display ist an diesen Bus angeschlossen
# Auflösung 128x64 Pixel
oled = SSD1306_I2C(128, 64, i2c)

# OLED Display löschen und Text anzeigen
oled.fill(0)
oled.rect(0,0,128,64,1)
oled.text("Attraktor e.V.", 10, 16)
oled.text("Hello World!", 16, 40)
oled.show()
```

In diesem Beispiel nutzen wir das OLED Display

Das OLED Display kann uns Texte direkt auf unserem Board anzeigen, ob ein Menü, Messwerte oder einfach Status Texte. Dafür ist dieses Display über einen sogenannten I2C Bus mit dem Pi Pico verbunden. Darüber werden die anzuzeigenden Daten seriell übertragen, was mit je einer Daten- und einer Takt-Leitung erfolgt. Das Display ist mit den Pins GPIO 20 (SDA Data) und GPIO 21 (SDC Clock) verbunden.

Die Bibliothek SSD1306_I2C erlaubt uns mit dem Display auf einfache Art und Weise zu kommunizieren und stellt uns die gängigsten Funktionen direkt zur Verfügung. Die Funktion *oled.fill(0)* löscht z.B. das Display und *oled.text("",x,y)* schreibt einen Text an eine bestimmte Position. Wichtig ist, dass man zum Anzeigen der vorangegangenen Befehlen am Ende *oled.show()* aufrufen muss.

Die ganzen vorhandenen Funktionen kann man hier sehen:
<https://docs.micropython.org/en/latest/esp8266/tutorial/ssd1306.html>

Beispiel 12 – Temperatur- und Luftdruck-Sensor BMP280

```
# Bibliotheken laden
# import libs
from machine import Pin, SPI
from utime import sleep
from bmp280_spi import BMP280SPI

# BMP280 Sensor an SPI Bus initialisieren
# GPIO 16 - Pico RX <- BMP TX
# GPIO 17 - Chip Select
# GPIO 18 - Clock
# GPIO 19 - Pico TX -> BMP RX
spi = SPI(0, sck=Pin(18), mosi=Pin(19), miso=Pin(16), polarity=1, phase=1)
spi_cs = Pin(17, Pin.OUT, value=1)
bmp280_spi = BMP280SPI(spi, spi_cs)

# Die BMP280 Chip ID Auslesen. Sollte 0x58 sein
# read BMP280 Chip ID. Should be 0x58
print(bmp280_spi.chip_id)

# Sensor auslesen und die gemessenen Werte anzeigen
# read sensor and print out measurements
while True:
    readout = bmp280_spi.measurements
    print(f"Temperature: {readout['t']} °C, pressure: {readout['p']} hPa.")
    sleep(1)
```

In diesem Beispiel nutzen wir den BMP280 Sensor

Der BMP280 Sensor kann uns Temperatur- und Luftdruck-Messwerte liefern und wird über einen etwas anderen Bus angesteuert. Der hier verwendete SPI Bus ist auch ein serieller Bus, wie I2C, der aber zusätzliche Leitungen hat und schneller getaktet werden kann. Für die Kommunikation nutzen wir hier die BMP280SPI Bibliothek, die es uns einfacher macht, die Messwerte auch gleich in menschenlesbare Formate zu bringen.

Erst laden wir wieder die Bibliotheken ein, initialisieren die GPIO Pins 16 bis 19 und erzeugen uns damit ein bmp280_spi Objekt, welches die ganze Funktionalität enthält. Wir lesen erst einmal die Chip_id aus und lesen dann im Sekundentakt die Messwerte ein und geben diese im Status-Bereich in Thonny aus.

Die Print Ausgabe ist hier etwas komplexer gestaltet, zeigt aber die mächtigen Textformatierungsmöglichkeiten von MicroPython.

Beispiel 13 – Raumklima auf OLED Display anzeigen

```
# Bibliotheken laden
from machine import Pin, I2C, SPI
from utime import sleep
from ssd1306 import SSD1306_I2C
from bmp280_spi import BMP280SPI
```

```
## OLED initialisieren
i2c = I2C(0, sda=Pin(20), scl=Pin(21))
oled = SSD1306_I2C(128, 64, i2c)
oled.fill(0)
oled.show()
```

```
# SPI BPM280 initialisieren
spi = SPI(0, sck=Pin(18), mosi=Pin(19), miso=Pin(16), polarity=1, phase=1)
spi_cs = Pin(17, Pin.OUT, value=1)
bmp280_spi = BMP280SPI(spi, spi_cs)
```

```
# Test Loop for Keys and Temp
while True:
    oled.fill(0)
    oled.text("Temp & Luftdruck", 0, 0)
```

```
    readout = bmp280_spi.measurements
    oled.text(f"Tc: {readout['t']:.2f}C", 2,16)
    oled.text(f"Tf: {(readout['t']*1.8+32):.2f}F", 2,26)
    oled.text(f"Tk: {(readout['t']+273.15):.2f}K", 2,36)
    oled.text(f"Ld: {readout['p']:.2f}hPa", 2,46)
```

```
    oled.show()
    sleep(0.25)
```

Hier das Grande Finale in dem wir den BMP280 Sensor nutzen und auf dem OLED Display ausgeben. Damit sind wir alle Funktionen des Demoboards einmal durchgegangen und eigenen Experimenten sollte jetzt nichts mehr im Wege stehen. Wie wärs z.B. mit einem Temperatur-Alarm oder -Ampel?