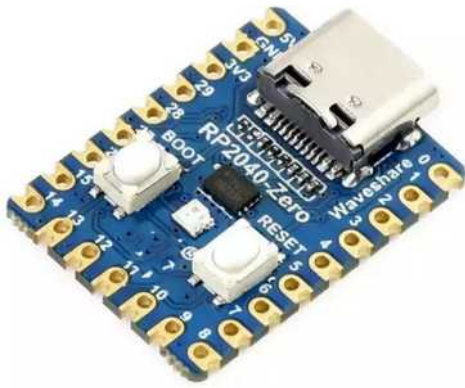


Anwendungen in Micropython



Titel

Anwendungen in Micropython

Kurze Einführung in Micropython.

Hier wird als Entwicklungsoberfläche 'Thonny' verwendet und als Prototyp_Board ein „Raspberry Pi Pico Zeo“.

Es werden Beispiele gezeigt wie Led-Ansteuerung, Taster-Abfrage, AD-Wandler(Poti), Tonerzeugung, CW- Bake.

Kapitel-Übersicht

1

Vorstellung einiger Boards

- ESP8266 , ESP32
- RaspberryPico RP2040

2

Python bzw. Micropython

- Ranking Programmiersprachen
- Vorführung: Python am PC

3

Entwicklungsoberfläche

- Installation von Thonny
- Einstellungen

4

Praktische Beispiele

Zielobjekt: CW-Bake ,CW-Keyer

- Led / Blink- Led
- Autostart
- Taster + Led
- Poti am AD-Wandler
- Tonerzeugung / Taste + Ton

5

Verweise

Übersicht

- Vorstellung einiger Boards
- Warum Python bzw. Micropython?
- Vorstellung der Entwicklungsoberfläche
- Praktische Beispiele mit Vorführungen;
dabei lernen wir einige Python Befehle
- Zum Schluss ein paar Verweise

Kapitel 1

1

Vorstellung einiger Boards

- ESP8266 , ESP32
- RaspberryPico RP2040

2

Python bzw. Micropython

- Ranking Programmiersprachen
- Vorführung: Python am PC

3

Entwicklungsoberfläche

- Installation von Thonny
- Einstellungen

4

Praktische Beispiele

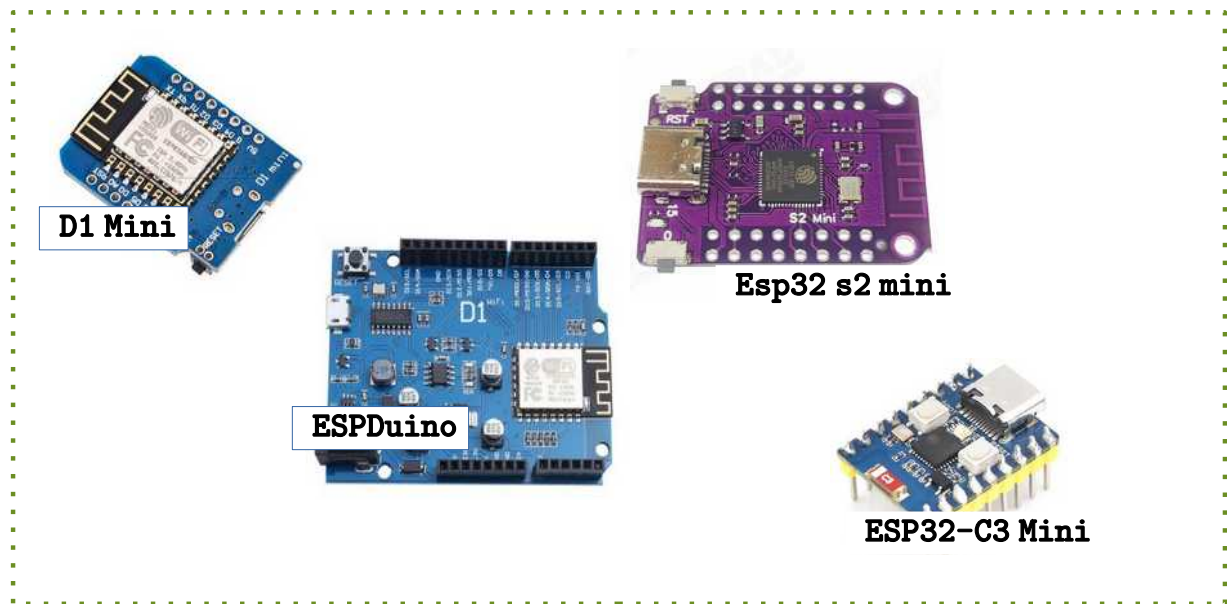
Zielobjekt: CW-Bake ,CW-Keyer

- Led / Blink- Led
- Autostart
- Taster + Led
- Poti am AD-Wandler
- Tonerzeugung / Taste + Ton

5

Verweise

ESPxx - Boards



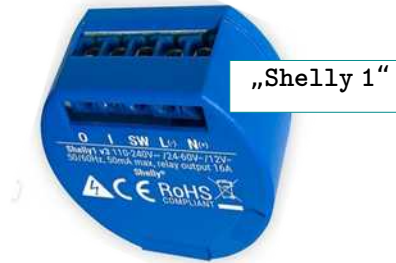
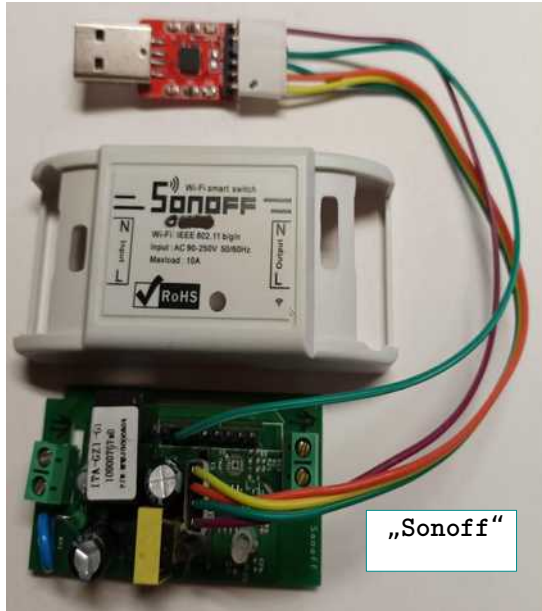
Das Bild zeigt einige Boards, die einen ESP8266- oder ES32-Prozessor verbaut haben.

Die Module werden meist unter dem Stichwort ‚Arduino‘ angeboten .

Das Board ESPDUINO ist in der Form dem Original Arduino nachempfunden, hat jedoch einen ESP8266 Prozessor (3.3V , weniger Ports als das Original) .

Für die vorgestellten Versuche kann auch eines dieser ESP- Boards verwendet werden.

Anwendungsbeispiel ESP8266



Die ESP8266 Boards werden hier genannt, da sie in Millionen Anwendungen im Beriech IoT („Internet Of Things“) verwendet werden.

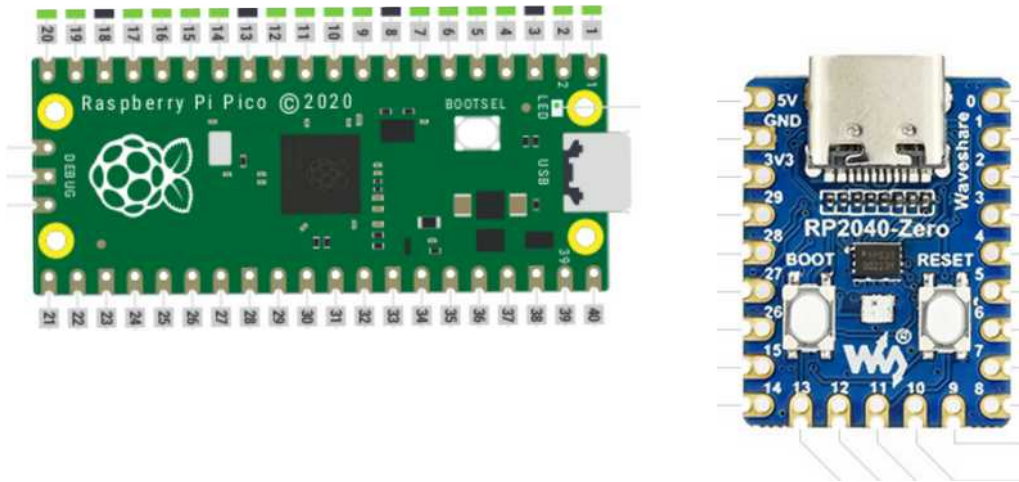
Das Bild zeigt zwei WIFI-Funkschalter.

Diese Funkschalter verwenden auch einen ESP8266-Prozessor, der sich zum Programmieren – sprich „Flashen“ – eigener Anwendungen anbietet, da die serielle Schnittstelle über einen USB-Serial-Wandler erreichbar ist.

Vorsicht 230 Volt !

Raspberry Pi Pico Boards

Ref.: [6]



Für meine Experimente habe ich mich für ein modernes RaspberryPi Pico Board („RP2040-Zero“) entschieden.

Kapitel 2

1

Vorstellung einiger Boards

- ESP8266 , ESP32
- RaspberryPico RP2040

2

Python bzw. Micropython

- Ranking Programmiersprachen
- Vorführung: Python am PC

3

Entwicklungsoberfläche

- Installation von Thonny
- Einstellungen

4

Praktische Beispiele

Zielobjekt: CW-Bake ,CW-Keyer

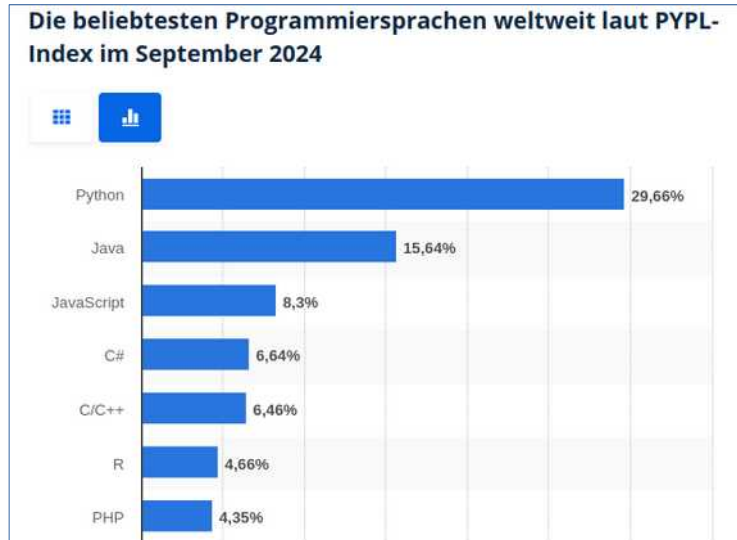
- Led / Blink- Led
- Autostart
- Taster + Led
- Poti am AD-Wandler
- Tonerzeugung / Taste + Ton

5

Verweise

Ranking der Programmiersprachen

<https://de.statista.com/statistik/daten/...>



Schaut man sich den Beliebtheitsgrad der aktuellen Programmiersprachen an , so rangiert Python an erster Stelle.

Python ist so etwas wie das Basic des 21. Jahrhunderts

Die Verbreitung von Single-Board-Mikrocontrollern wie z.B. Raspberry Pi , ESPxxxx-Module, BBC-Microbit und Calliope hat sicher auch zu dem Erfolg von Python beigetragen.

Das Ranking gilt nur für relativ leistungsfähige Prozessoren (32bit, ARM etc.)

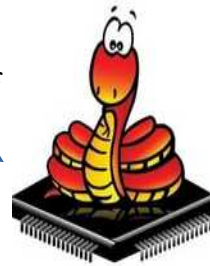
Für 8-Bit-Mikroprozessoren ist nach wie vor C++ die dominierende Sprache (z.B. bei dem Arduino ATmega328).

Micropython = Python für Mikroprozessor

Aus



wird für den Mikroprozessor



„Micropython“.

Stand Sept 2024:
Python 3.12.5
Micropython v1.23

Python ist mit Basic der 80er Jahre vergleichbar, jedoch findet man bei Python Elemente der Objektorientierten Programmierung und Module für Netzwerk und Internet.

Python gibt es praktisch für jedes Betriebssystem (Win, Linux, MacOS) [1].

Auf den meisten 32-Bit-Mikroprozessoren lässt sich ein Python – Interpreter implementieren; der Name „Python“ wird dann zu „Micropython“ [2].

Python stammt aus den 80er Jahren. Heute (Sept. 2024) sind wir bei Version 3.12.

Micropython hat sich seit 2023 sehr schnell entwickelt. Im Sept. 2024 sind wir bei der Micropython Version v1.23.

Als Prozessoren werden z.B. STM32, TI CC3200, SAMD21, ESP8266, ESP32, RP2040 unterstützt.

Python am PC in der Kommandozeile

Ref.: [1]

Start Python über Kommandozeile:

```
dk2jk@linux:~$ python3.12
Python 3.12.5+ (main, Aug  9 2024, 08:50:51) [GCC 9.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import time
>>> time.asctime()
'Thu Sep 19 11:05:34 2024'
>>> time.asctime().split()
['Thu', 'Sep', '19', '11:05:41', '2024']
>>> time.asctime().split()[3]
'11:05:48'
>>> █
```

Python läuft auf nahezu jedem Betriebssystem .

Bei Linux ist Python schon installiert, es ist sozusagen die Haussprache.

Nach der Installation [1], erfolgt der erste Test über die Kommandozeile.

Auf der Konsole wird die Python-Version angezeigt.

Hinter dem Prompt >>> kann nun ein Python-Befehl eingegeben werden.

Die Programmierung erfolgt interaktiv. Eventuelle Fehlermeldungen werden sofort ausgegeben.

Beispiele:

```
>>> import time      # das Modul time wird importiert
>>> time.asctime()   # diese Funktion liefert die Zeit im
ASCII-Format
>>> time.time()      # gibt die Sekunden seit 1970 an
                     'UNIX-Time'
```

Im Bild sind weitere Funktionen dargestellt.

Python am PC als Script

Datei : mytime.py

Das Gleiche als ‚Script‘ im Texteditor:

```
mytime.py
1 import time          # modul time einbinden
2 t=time.asctime()     # zeit lesbar holen 'Thu Sep 19 10:55:14 2024'
3 t1=t.split()         # string teilen ['Thu', 'Sep', '19', '10:55:24', '2024']
4 hms= t1[3]           # das 3. element der liste '10:55:39'
5 text= f'Es ist jetzt {hms} Uhr'
6 print (text)         # es ist jetzt Thu Sep 19 10:53:23 2024 Uhr
7
```

Start des Python-Scripts über Kommandozeile:

```
dk2jk@linux:~/Schreibtisch/swt2024$ python3 mytime.py
Es ist jetzt 11:55:41 Uhr
```

Zusammenhängende Kommandos wurden in der Datei
,my_time.py‘ als Text (Script) gespeichert.

Aufruf über Kommandozeile :
> **python3 my_time.py**

Man kann das Script auch im Python-Terminal starten:

```
dk2jk@linux:~/python3
...
>>> import mytime ( ohne .py )
Es ist jetzt 17:46:21 Uhr
>>>
```

Diese Vorgehensweise ist natürlich spartanisch und
unpraktisch.

Da hilft uns eine Entwicklungsoberfläche (IDE).

Hier wird die Entwicklungsoberfläche ‚Thonny‘ gewählt.

Alternative IDEs sind: ‚Vscode‘, ‚idle‘, ‚Geany‘, Pycharm
u. andere.

Kapitel 3

1

Vorstellung einiger Boards

- ESP8266 , ESP32
- RaspberryPico RP2040

2

Python bzw. Micropython

- Ranking Programmiersprachen
- Vorführung: Python am PC

3

Entwicklungsoberfläche

- Installation von Thonny
- Einstellungen

4

Praktische Beispiele

Zielobjekt: CW-Bake ,CW-Keyer

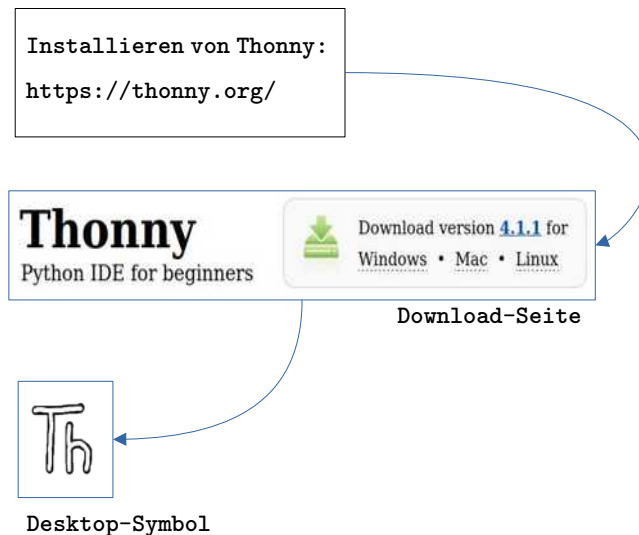
- Led / Blink- Led
- Autostart
- Taster + Led
- Poti am AD-Wandler
- Tonerzeugung / Taste + Ton

5

Verweise

IDE ,Thonny' installieren

Ref.: [3]



Das Entwicklungswerkzeug „Thonny“ ist von der angegebenen WEB-Adresse herunterladbar (Ref.: [3]).
<https://thonny.org/>

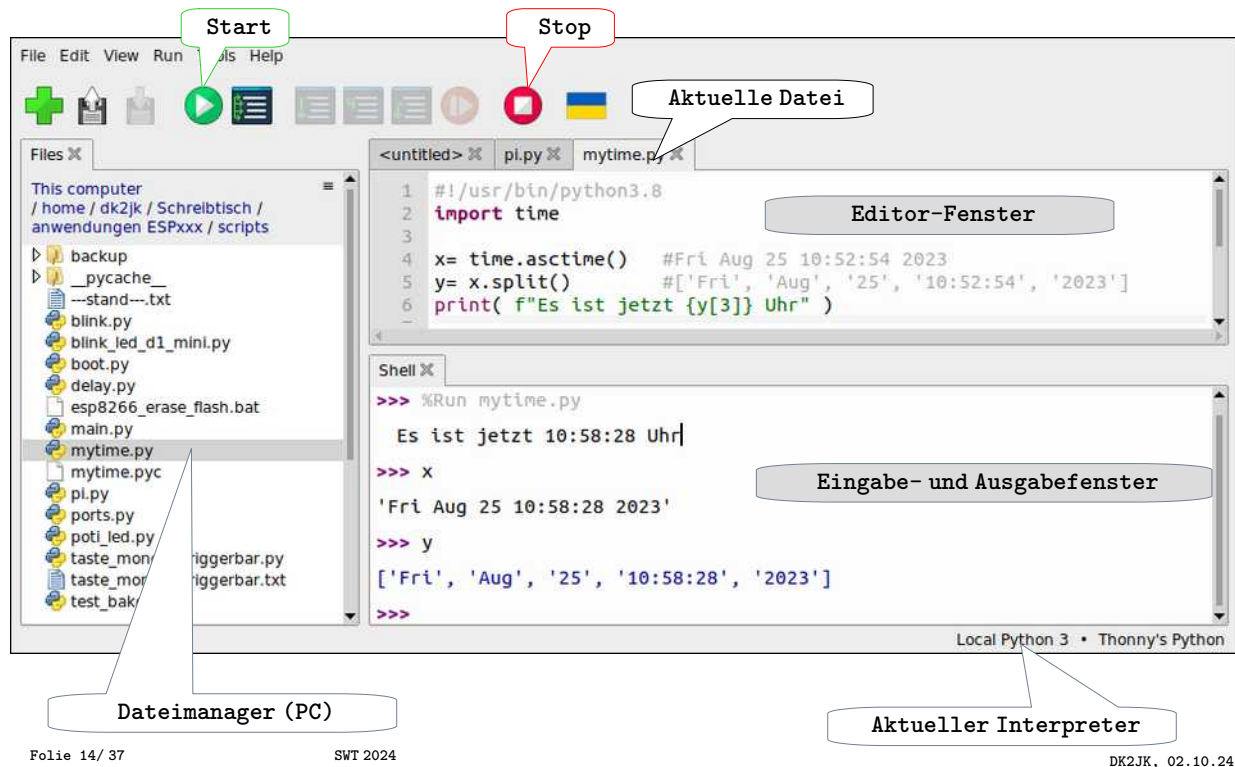
Für Windows gibt es eine Installations-Datei z.B.

thonny-4.1.2.exe (21 MB)

Unter Linux installiert man über das Paketmanagementsystem 'apt' mit dem Kommando

sudo apt install thonny

Entwicklungsumgebung ,Thonny' Ansicht



Hier nochmal die einzelnen Programmschritte:

Zeile 2 : Modul `time` wird importiert, damit stehen `time`-Funktionen zur Verfügung

Zeile 4 : Die `time`-Funktion `time.asctime()` wird ausgeführt; `x` ist ein String z.B. 'Tue Sep 24 12:02:50 2024'

Zeile 5 : Der String `x` wird zerlegt ('gesplittet'), wobei das Leerzeichen standardmäßig als Trennzeichen dient
→ ['Tue', 'Sep', '24', '12:02:50', '2024']

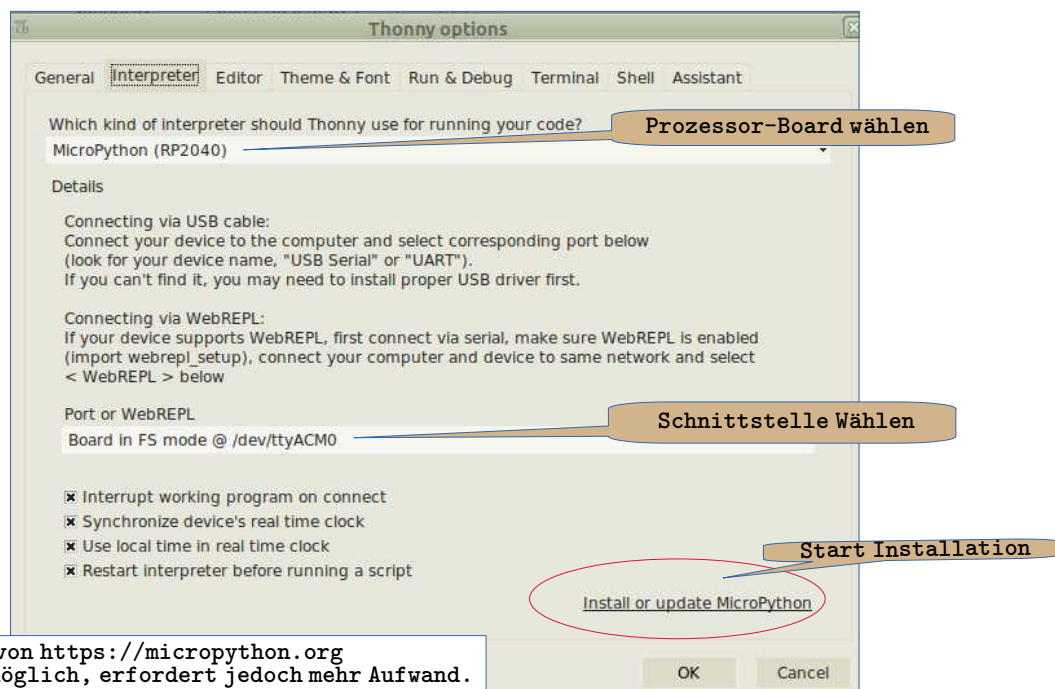
Zeile 6 : Ausgabe; `f""` bedeutet Format-String, in `{ }` steht die auszugebende Variable `y[3]` ist das 3. Element der Liste `y` -> '12:02:50'

Ausgabefenster:

%Run mytime ausgelöst über Button 'Run'

Die Variablen `x` und `y` stehen nach Programmende noch zu Verfügung und können interaktiv abgerufen werden; dazu muss das Programm von selbst enden oder mit `CNTRL C` abgebrochen werden.

Flash Laden mit Thonny



Das bisherige Beispiel wurde auf dem PC ausgeführt.

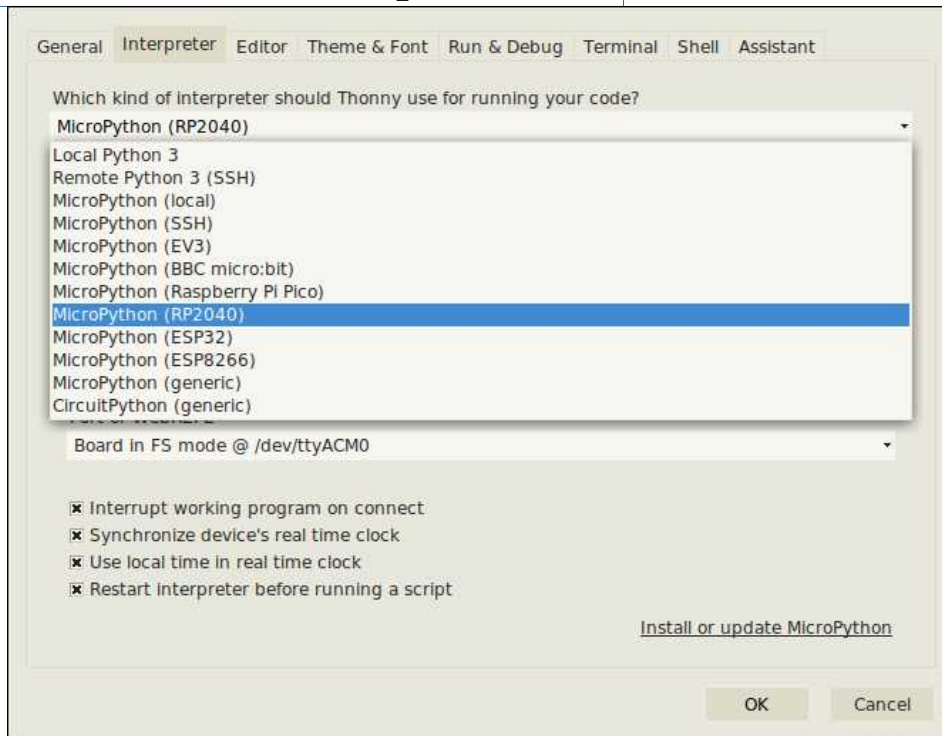
Wenden wir uns jetzt dem Mikrocontroller zu.

Bei einem neuen Board muss zunächst der Micropython Interpreter über den gewählten USB-Port geladen werden.

Es gibt da verschiedene Vorgehensweisen:

- Der Raspberry Pi Pico wird beim Einschalten – bei gleichzeitigen Drücken des ‚BOOTSEL‘ -Knopfes – als USB-Speicher erkannt , auf den die von [2] heruntergeladene Micropython – Binärdatei kopiert wird . Das Board ist nun bereit.
- Bei anderen Prozessoren(z.B. ESP32) wählt man z.B. in Thonny unter Tools – Options – Interpreter ‚Micropython <ProzessorTyp>‘. Die Installation wird interaktiv durchgeführt nach Anwahl des Punktes ‚install firmware‘ .

Starten des Interpreters



Das Board wird mit der USB- Schnittstelle verbunden. Der PC erkennt eine serielle Schnittstelle mit 115200 Baud.

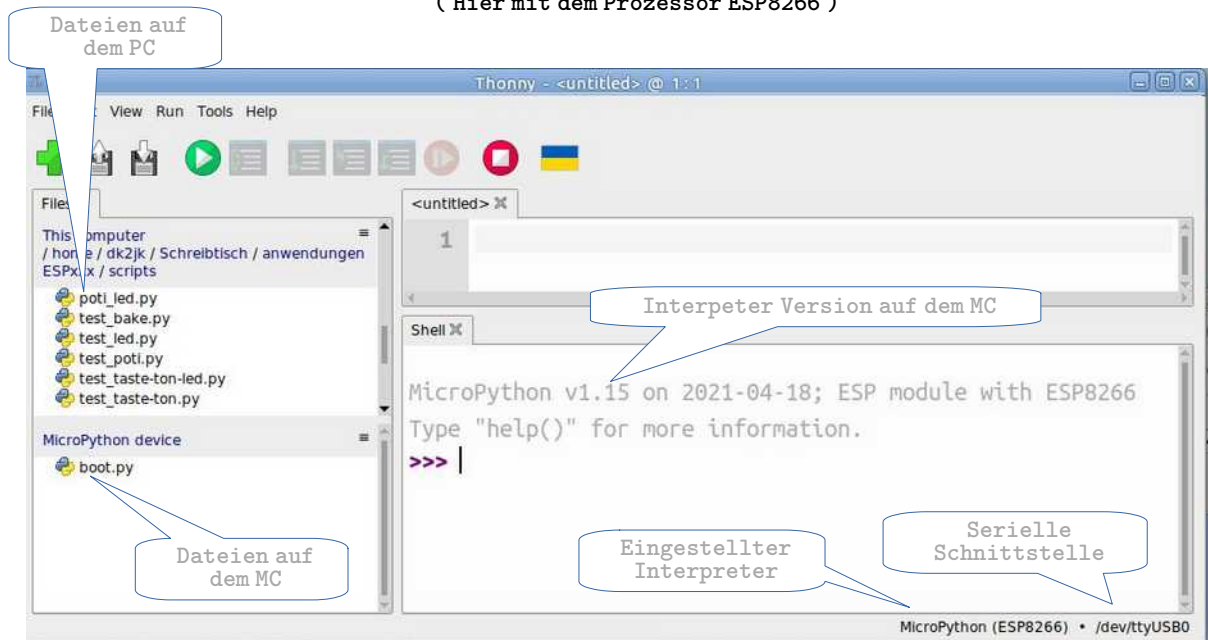
Dann wählen wir den Interpreter ‚Micropython RP2040‘ und die angebotene USB-Schnittstelle aus .

Die Schnittstellen heißen
bei Windows „COMx“
und
bei Linux „/dev/tty/USBx“.

Fertig mit ‘ok‘.

‚Micropython‘ , Erster Start

(Hier mit dem Prozessor ESP8266)



Jetzt erscheinen in der Thonny Oberfläche auf der linken Seite zwei Fensterim Dateimanager: ‚This Computer‘ und ‚MicropythonDevice‘.

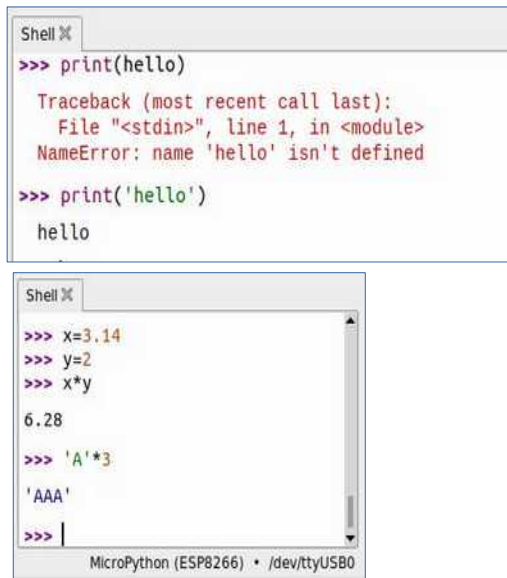
Ja, der Prozessor hat sein eigenes Dateisystem. Auf einem neuen Board ist eine Datei ‚boot.py‘ vorhanden, die direkt nach dem Reset ausgeführt wird.

Damit die Dateien angezeigt werden, muss View → Files gewählt sein.

Rechts oben ist das Editorfenster, hier noch mit der Bezeichnung ‚untitled‘. Durch Anklicken einer Datei im Dateimanager wird eine Datei im Editorfenster geöffnet.

In der Shell zeigt sich beim Start eine Systemmeldung mit der Softwareversion. Dieses Fenster ist die Kommandozeile ; hier werden auch Programmausgaben angezeigt (print-Befehle, Fehlermeldungen, interaktives Terminal).

Vorführung: Micropython interaktiv'

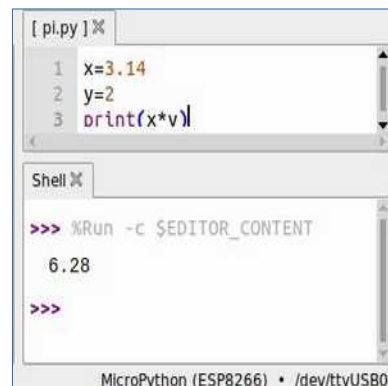


```
Shell X
>>> print(hello)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'hello' isn't defined

>>> print('hello')
hello

Shell X
>>> x=3.14
>>> y=2
>>> x*y
6.28
>>> 'A'*3
'AAA'
>>>
MicroPython (ESP8266) • /dev/ttyUSB0
```

Das erste Script



```
[ pi.py ] X
1 x=3.14
2 y=2
3 print(x*y)

Shell X
>>> %Run -c $EDITOR_CONTENT
6.28
>>>
MicroPython (ESP8266) • /dev/ttyUSB0
```

Hier ein paar Versuche auf der Kommandozeile in der Shell. Bei falscher Syntax kommt eine Fehlermeldung.

Im zweiten Bild definieren wir Variable x und y; diese werden multipliziert. Das Ergebnis erscheint hier sofort.

Die Anweisung 'A'*3 bedeutet 3* String 'A' = 'AAA'

Wir können auch eine Datei erstellen - im Beispiel 'pi.py'.

Die Datei kann direkt ausgeführt werden über 'Run' = Grüner Knopf. Beim Schließen der Datei wird nach dem Ziel gefragt; entweder unter 'This computer' oder 'Micropython Device'.

Print -Ausgabe erfolgt im unteren Fenster.

Kapitel 4

1

Vorstellung einiger Boards

- ESP8266 , ESP32
- RaspberryPico RP2040

2

Python bzw. Micropython

- Ranking Programmiersprachen
- Vorführung: Python am PC

3

Entwicklungsoberfläche

- Installation von Thonny
- Einstellungen

4

Praktische Beispiele

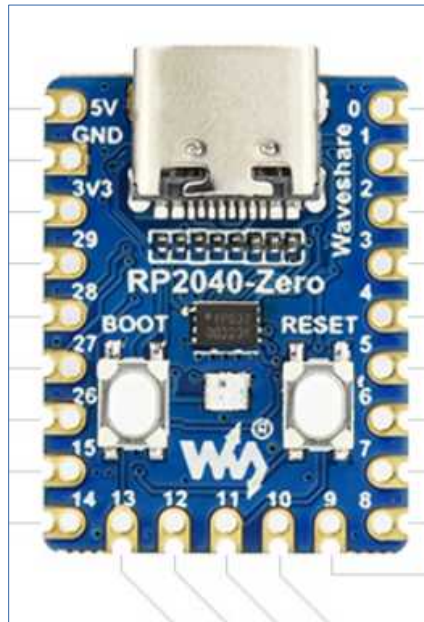
Zielobjekt: CW-Bake ,CW-Keyer

- Led / Blink- Led
- Autostart
- Taster + Led
- Poti am AD-Wandler
- Tonerzeugung / Taste + Ton

5

Verweise

Das hier verwendete Board: RP2040-Zero



Im Bild das Board „RP2040-Zero“.

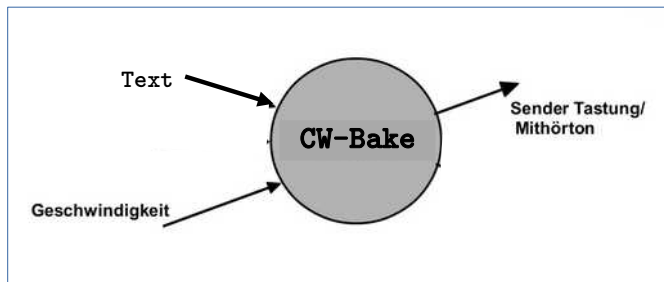
Die meisten Pins (GPIO xx) sind frei verfügbar.

Alle Pins vertragen nur 3.3V als Eingangsspannung.

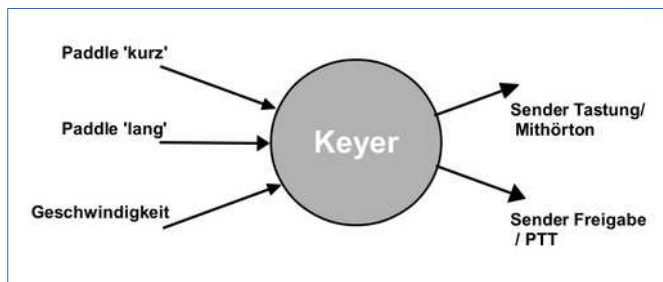
Der ADC hat Vollaussteuerung bei 3.3V

Die 5V vom USB liegen an Vin an; alternativ kann die Versorgung an diesem Pin von extern mit 5V erfolgen, wenn keine USB-5V anliegen.

Anwendungsbeispiele : CW-Bake und CW-Keyer



CW-Bake
...Sendet vorgegebenen Text in CW



CW-Keyer
...Macht aus der Kombination der Paddle Eingänge Cw-zeichen

Hier zwei Entwürfe: eine CW-Bake und ein CW-Keyer.

Die Grafik zeigt nur die notwendigen Ein-und Ausgänge und das eigentliche Programm als ‚Black box‘.

Zunächst beschäftigen wir uns nur mit den Ein- und Ausgängen.

Benötigte Funktionen für ,keyer' bzw ,bake'

- Poti zum Einstellen der Morsegeschwindigkeit → wpm ; $tdit = 1200 / wpm$
- Für Keyer: Paddle-Eingänge ,kurz' und ,lang'
- Für Bake: Text-Vorgabe , fest programmiert
- Für Bake: HF-Ausgang
- Schaltausgang, der den Sender tastet ,key'
- Option: Ausgang zum Freigeben des Tx / Sperren des Rx ,ptt'
- Option: Mithörton
- Zeitgeber für die Länge der Morse-Elemente (intern)

Die benötigten Funktionen werden wir uns im Einzelnen in kleinen Beispielen ansehen.

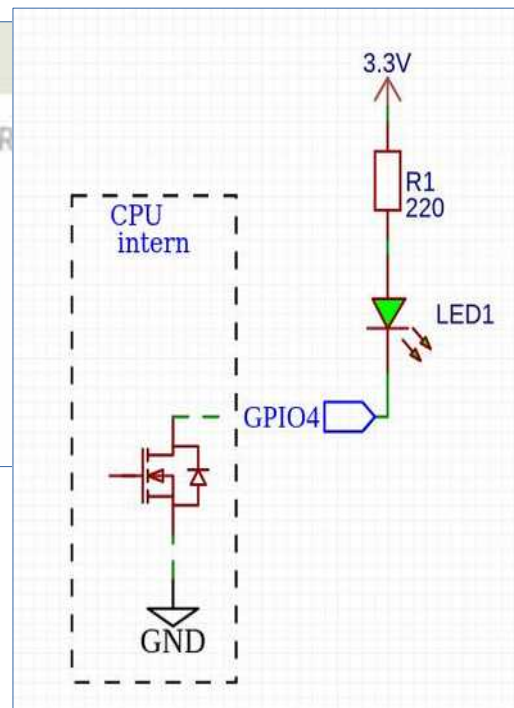
Schalten eines Ausgangs; Beispiel LED

Shell

```
MicroPython v1.20.0 on 2023-04-26; R
Type "help()" for more information.
>>> from machine import Pin
>>> led = Pin( 7 ,Pin.OUT,value=1)
>>> led.value(0)
>>> led.value()
0
```

An : led.value(0)

Aus: led.value(1)



Die LED wird über einen 220 Ohm Reihenwiderstand an 3.3V angeschlossen.

Zunächst arbeiten wir in der Shell, erkennbar an den 3 Pfeilen ,>>>', also interaktiv.

Schritt 1: Aus dem Modul ,machine' wird die Klasse ,Pin' importiert.

Schritt 2: Die LED mit dem Namen ,led' wird definiert(im Jargon heisst das ,instanziert').

led = Pin (7, Pin.OUT, value =1)

,led' ist ein beliebiger Name , hier eine Instanz der Klasse ,Pin'.

,7' ist die Pin-Nr.

,Pin.OUT' bedeutet : Pin ist ein Ausgang .

,value =1' setzt den Ausgang sofort auf HIGH, so dass die Led aus ist (Low Aktiv)

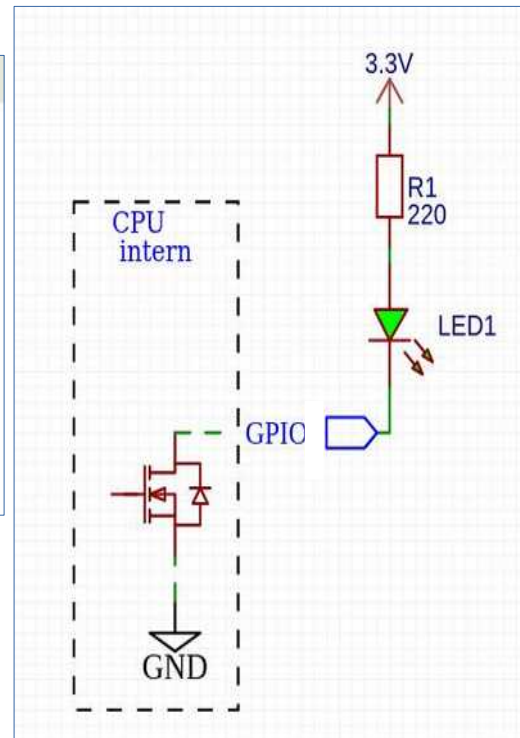
Jetzt können wir damit arbeiten:

led.value(0) ---> an

led.value(1) ---> aus

Script Beispiel :Blink-Led

```
1 from time import sleep
2 from machine import Pin
3 LED_BLAU_PIN = const(7)
4 led = Pin(LED_BLAU_PIN, Pin.OUT, value=1)
5
6 def blink():
7     while True:
8         x= not led()
9         led(x)
10        sleep(.33)
11
12 print("blink Led")
13 blink()
```



Mit dem Schlüsselwort ‚def‘ + Name beginnt man eine Funktion; die Funktion könnte Übergabeparameter haben; hier gibt es keine, also leere Klammern ‚()‘.

Die def-Zeile endet mit ‚:‘ und anschliessend erfolgt eine Einrückung um 4 Leerzeichen. Python hat keine geschweiften Klammern als Blockabgrenzung wie in ‚C‘. Der Block ist zu Ende, wenn die Einrückung wieder aufgehoben wird.

Hinter dem Doppelpunkt beginnt der eingerückte While-Block.

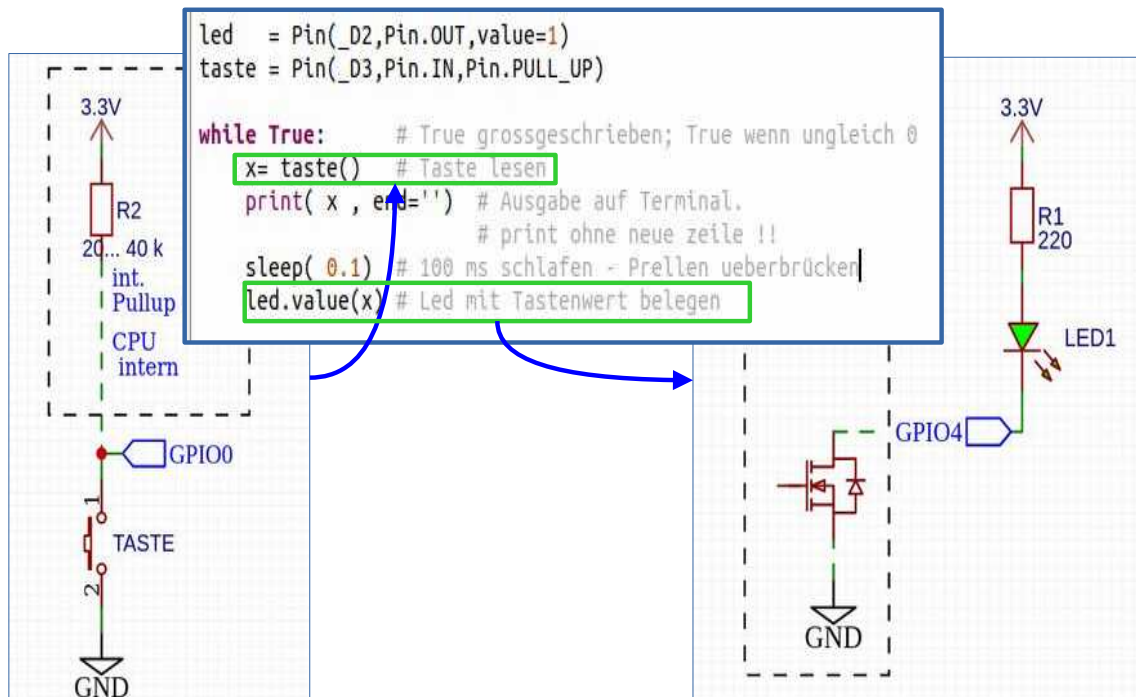
Der Zustand der Led wird mit ‚led()‘ gelesen, ohne Wert in der Klammer. Daraus wird 0 oder 1 .

Die Blinkzeit wird mit sleep(.33) eingestellt; die Led ist also 1/3 sec an und 1/3 sec aus.

sleep(Sekunden) nimmt Sekunden als Float-Wert als Parameter.

Abfragen eines Eingangs: Taster + Led

Script:
test_taste_led.py



Folie 25/ 37

SWT 2024

DK2JK, 02.10.24

Die Taste wird ähnlich definiert wie die LED:

```
taste = Pin(_D3, Pin.IN, Pin.PULL_UP)
```

Nur jetzt ist der Pin ein Eingang (Parameter 'Pin.IN') und eine Besonderheit: man kann mit dem Parameter 'Pin.PULL_UP' den Eingang mit einem Pullup-Widerstand belegen, so dass man extern keinen Widerstand benötigt.

_D2 und _D3 sind Konstanten mit der Pin Nummer, die vorher definiert wurden.

Die Taste ist LOW-aktiv, d.h. LOW bedeutet 'Taste gedrückt'

Im Bild wird das Programm mit Kommentar dargestellt.

Das Programm mag zunächst unsinnig erscheinen, da man ja die LED direkt an den Taster anschliessen könnte. In der Praxis wird vom Taster eine Funktion ausgelöst; die LED kann dann z.B. blinken, wenn ein Fehler auftrat. Also: alle Peripherie gehört an den Controller, damit man volle Kontrolle hat (wie der Name schon sagt).

Autostart am Beispiel ,Blink- Programm'

'boot.py'|
'main.py'

2 besondere Programme:

,boot.py' startet beim
Einschalten automatisch.

,main.py' wird nach
,boot.py' gestartet

In ,main.py' kann meine
Anwendung laufen oder ein
anderes Modul importiert
werden

```
[ boot.py ] *X
# boot.py
import gc
gc.collect()
```

```
[ test_led.py ] X
from machine import Pin
from time import sleep

_D2= const(4)
led = Pin(_D2,Pin.OUT,value=1)

def blink():
    while True:
        led( not led() )
        sleep(.5)
```

```
[ main.py ] X
from test_led import blink
blink()
```

Ausführen!

Autostart

Nachdem wir etwas fertig programmiert haben, soll das Board ja mein Programm sofort nach dem Einschalten starten.

Der Prozessor führt nach dem Reset ,boot.py' aus; da stehen Dinge drin , die nur einmal beim Start ausgeführt werden sollen.

Danach sucht der Prozessor nach dem Programm ,main.py'

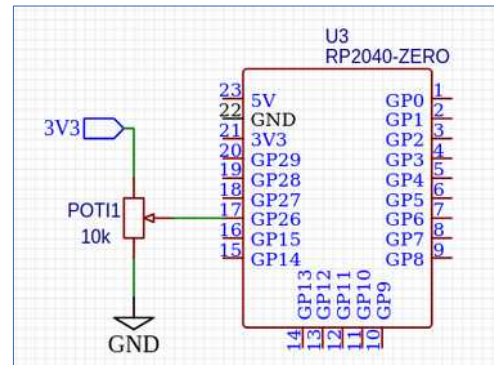
Um das eigene Programm automatisch zu starten, importiert man in 'main.py' das eigene Modul.

Hier im Beispiel wird das Blink-Programm ,test_led.py' gestartet.

Poti am AD-Wandler

Script:
test_poti.py

```
<untitled> *  
1 from machine import Pin,ADC  
2  
3 adc = ADC(Pin(26)) #rp2040  
4  
5 import time  
6 while True:  
7     x= adc.read_u16()  
8     print(x)  
9     time.sleep(3)  
  
Shell  
MPY: soft reboot  
80  
2256  
9442  
15619  
22149  
65535  
65535
```



Die Aufgabe besteht darin, die Stellung des Potentiometers lesen, um daraus bei dem einem CW-Keyer die Geschwindigkeit einzustellen.

Das Poti als Spannungsteiler liefert 0 ... 3,3Volt.

Der ADC übersetzt 0 bis 3.3V in den Zahlenwert 0 bis 65535 (Datentyp ‚unsigned Integer‘ = Ganze positive Zahl).

Anmerkung: bei ESP8266 Eingang: 0..1 Volt / digital: 0 bis 1023 (10 bit)

Zeile 3: Der ADC soll ‚adc‘ heißen.

Wir lesen den ADC-Wert mit `adc.read_u16()` im Sekundentakt aus und drehen derweil am Poti.

Die Ausgabe erscheint in der Shell.

Das ist eine Ausgabe über die serielle Schnittstelle des USB-Serial-Wandlers.

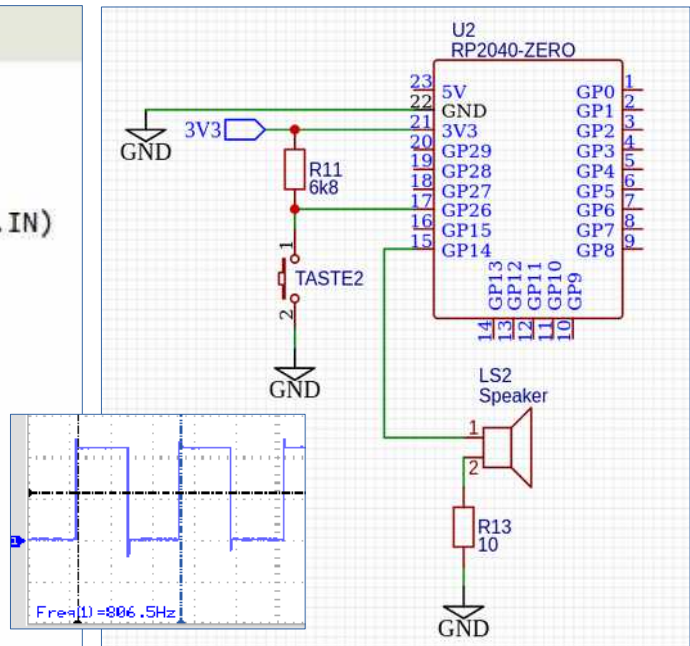
Generell gilt : Micropython lässt sich über COMx bedienen (oder bei Linux /dev/tty/USBx) mit 115200 Baud.

Tonerzeugung

Script:
taste_ton.py

[taste_ton.py] *

```
1 #aus unserem baukasten:
2 from ports_rp2040 import *
3 from machine import Pin,PWM
4
5 tunetaste = Pin(TUNE_PIN,Pin.IN)
6
7 # mithoerton
8 ton = PWM ( Pin(TON_PIN))
9 ton.freq(600)
10 ton.duty_u16(0)
11
12 while True:
13     if tunetaste()==0:
14         ton.duty_u16(32767)
15     else:
16         ton.duty_u16(0)
17     #sleep(.01)
```



Einen Ton benötigen wir z.B. als Quittungston oder als Mithörton von Morsezeichen.

Die Tonerzeugung erfolgt über Pulsweitenmodulation (PWM). Der Ausgang ist ein Rechtecksignal.

ton = PWM (Pin(GPIONR))

PWM wurde aus dem Modul `,from machine import PWM'` importiert.

Als zusätzliche Eigenschaften `freq'` und `,duty_u16'` angegeben.

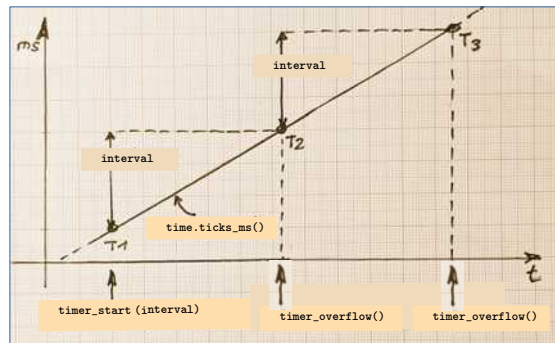
`,duty_u16'` bedeutet Dutycycle und gibt die Einschaltdauer in der PWM- Periode an (0 = 0% also Null Volt, 65535 = 100 % also 3.3 Volt).

Mit `,ton.duty_u16(0)'` wird der Ton wieder ausgeschaltet.

Anmerkung:

Das PWM-Modul hat bei anderen Prozessoren evtl. andere Funktionsaufrufe.

Zeitgeber für die Länge der Morse-Elemente



`time.ticks_ms() = Systemtimer`

```
[ timer_demo.py ] *
1  import time
2
3  interval=0
4  tn=0
5
6  def timer_start(_interval):
7      global tn,interval
8      interval=_interval
9      tn= time.ticks_ms()
10
11 def timer_overflow():
12     global tn
13     if time.ticks_ms() > tn:
14         tn=tn+interval
15         return True
16     else:
17         return False
```

Für die Elemente der Telegraphiezeichen (Kurz, Lang, Pause) wird ein Zeitgeber verwendet, der von einem Systemtimer abgeleitet wird.

Es werden Intervalle von ca. 30... 500 ms benötigt.

Der Zeitgeber wird gestartet, indem ein Schwellwert berechnet wird aus Jetzt-Zeit plus Interval.

Das Ereignis „Zeit-Ist-Abgelaufen“ tritt auf, wenn der Systemtimerwert größer als der vorausberechnete Schwellwert ist.

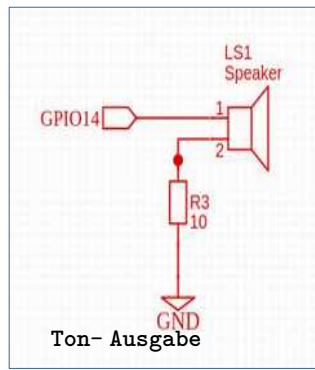
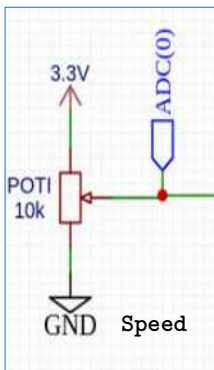
Der Prozessor ist in der Wartezeit nicht blockiert und kann derweil andere Dinge erledigen.

Realisierung: CW- Bake

Aufgabe: Text als Morsezeichen im Lautsprecher ausgeben.

Wir haben :

- Ein Poti zum Einstellen der Morsegeschwindigkeit (ADC)
- Einen Tongenerator (PWM)
- Einen Lautsprecher-Ausgang



Wir benötigen noch:

- eine Liste der Morsezeichen in der Form:
Tabelle = {
 '!': '-.-.-',
 '1': '.----', etc.
- Eine Programm-Schleife, die den Text Buchstabe für Buchstabe nacheinander liest
- Einen Übersetzer „Buchstabe in Morsecode“:
Aus ‚a‘ wird „.-“
- Einen Übersetzer der den Punkt-Strich-Morsecode in Töne wandelt:
Aus „.-“ wird hörbar dit dah

Wir haben :

- Ein Poti zum Einstellen der Morsegeschwindigkeit (ADC)
- Einen Tongenerator (PWM)
- Einen Ausgang.

Wir benötigen noch:

- Eine Schleife, die die Buchstaben des Baken-Textes nacheinander liest.
- Eine Liste der Morsezeichen mit der Zuordnung Buchstabe zu Morsecode:
Lookup = {
 '!': '-.-.-',
 '1': '.----',
- Einen Übersetzer „Buchstabe in Morsecode“
aus ‚a‘ wird „.-“
- Einen Übersetzer von Morsecode in Töne
Aus „.-“ wird hörbar dit dah

Programm Schnipsel: CW- Bake

Script:
cw_bake.py

```
'test' --> liste ['-','.', '...', '-']
```

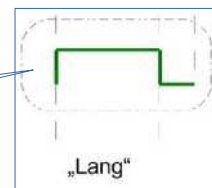
```
if element=='.':
```

```
    ton_an()  
    warte_millisekunden(dit_time_ms)  
    ton_aus()  
    warte_millisekunden(dit_time_ms)
```

```
if element=='-':
```

```
    ton_an()  
    warte_millisekunden(dit_time_ms*3)  
    ton_aus()  
    warte_millisekunden(dit_time_ms)
```

```
warte_millisekunden(dit_time_ms*2)
```



Buchstaben
Abstand

Ein praktisches Beispiel.

Input ist ein Text: z.B.: 'Test'

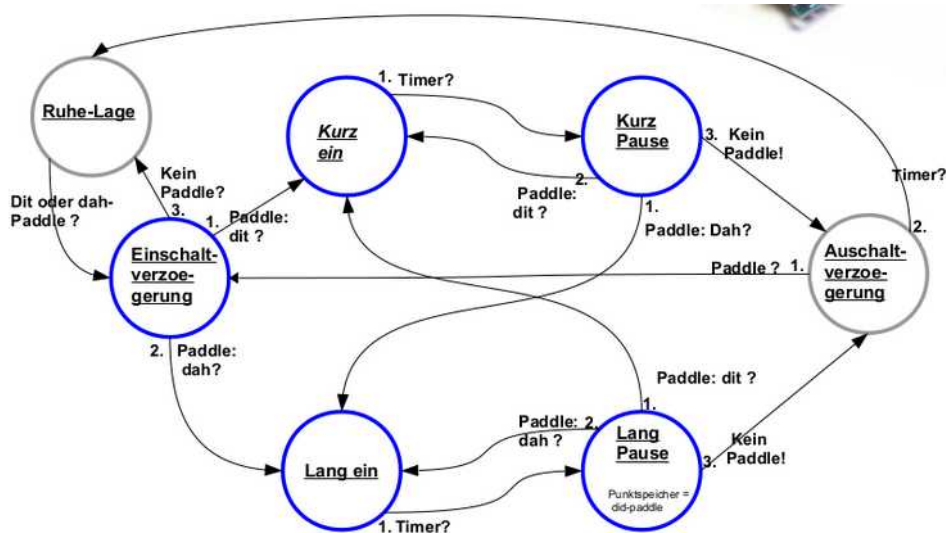
Folgende Funktionen werden verwendet:

- ton_an() ,
- ton_aus() ,
- warte_millisekunden(x) ,

Im Lautsprecher soll dieser Text als Morsezeichen ausgegeben werden.

Datei : cw_bake.py

Keyer Zustandsdiagramm (,state chart')



Hier ist das Programm ,keyer.py' als Zustandsdiagramm dargestellt.

Warum Zustandsdiagramm ?

Die Steuerung ist immer nur in einem einzigen Zustand (Kreis). Sie verharrt dort, bis ein Ereignis eintritt (,event'). Je nach Ereignis wird dem Ereignis entsprechend eine Aktion (Pfeil) ausgeführt und ein neuer Zustand eingeschaltet.

Ereignisse sind z.B. Betätigung einer Taste oder Ablauf eines Zeitgebers.

Da in jedem Zustand nur auf ein passendes Ereignis gewartet wird, sind die Prozessordurchlaufzeiten minimal, so dass der Prozessor noch andere Aufgaben (fast) gleichzeitig erledigen kann (z.B.: Frequenzeinstellung und Display)

Keyer Prototyp

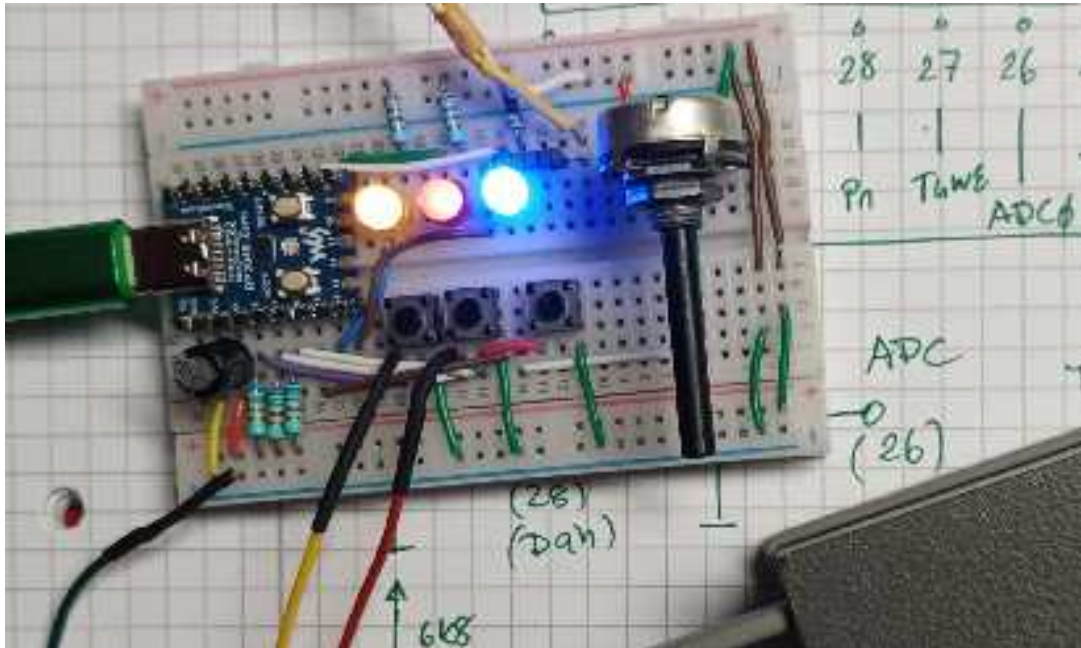
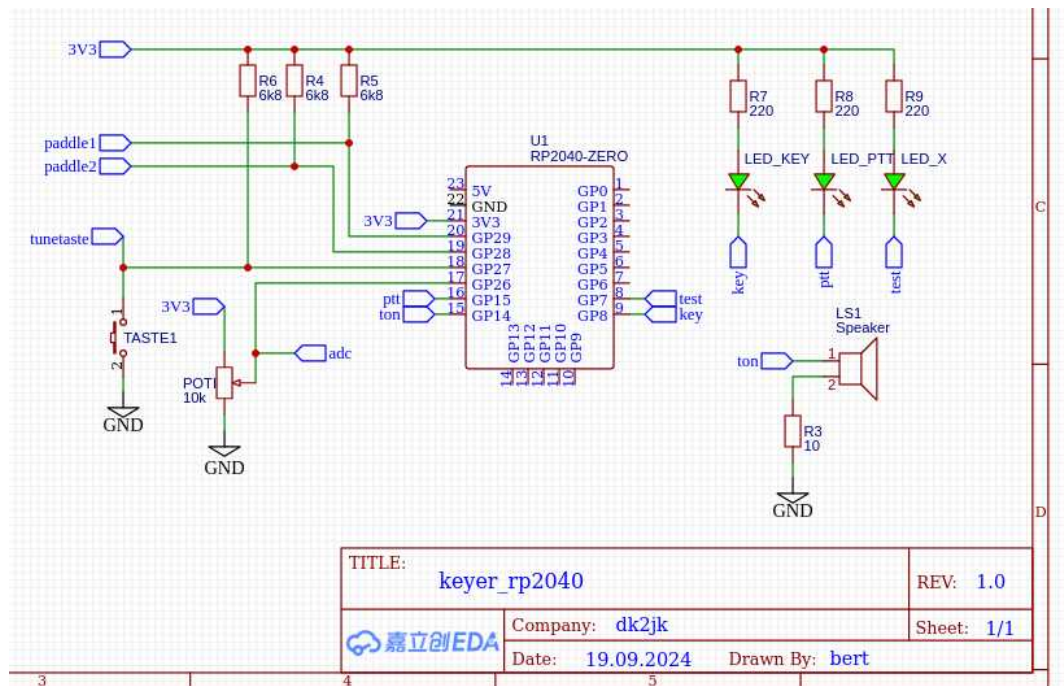


Bild: Keyer Prototyp

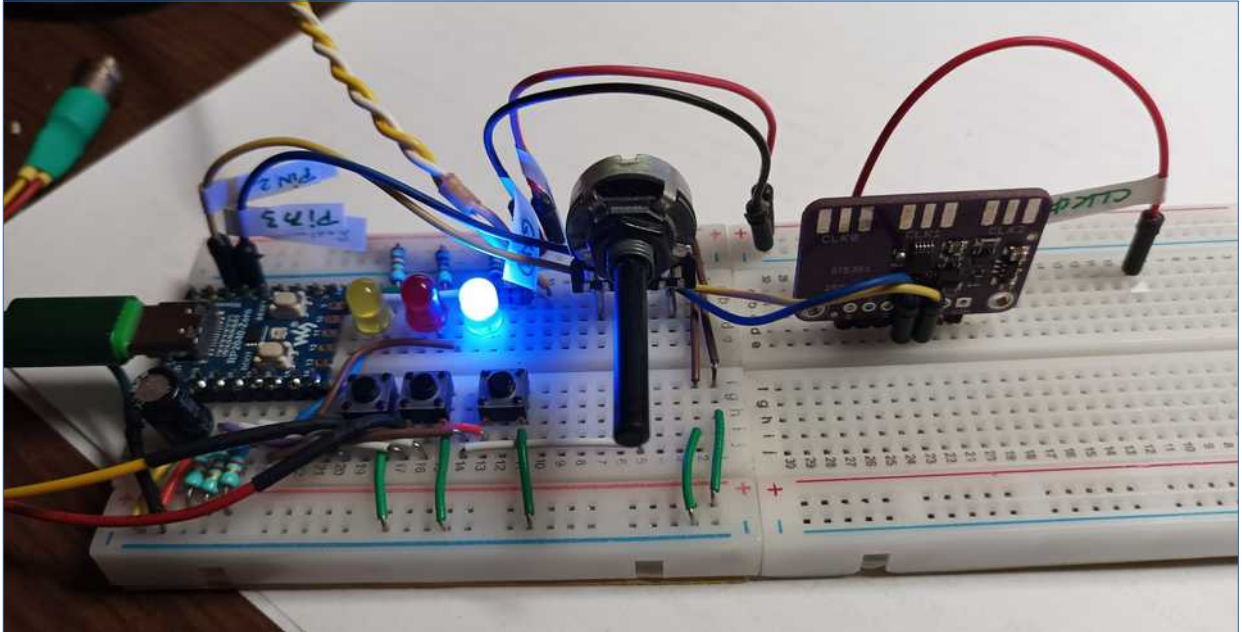
Links der Controller RP2040. Die LEDs stellen die Ausgänge dar. Die Buttons sind ,paddle links', ,paddle rechts' und ,tune'. Mit dem Poti wird die Geschwindigkeit eingestellt. Der gelbe und weiße Draht gehen zum Lautsprecher.

Keyer Schaltbild



Im Schaltbild sieht man, wie wenig Bauelemente notwendig sind. Links sind die Eingänge, rechts die Ausgänge.

Bake mit Si5351, Prototyp



Hier ein Prototyp der Bake mit HF-Ausgang. Die Bake wurde um ein si5351- Modul erweitert; dazu sind nur 4 zusätzliche Leitungen nötig (I2c-Bus sda,scl und 3.3v und GND)

, , ,

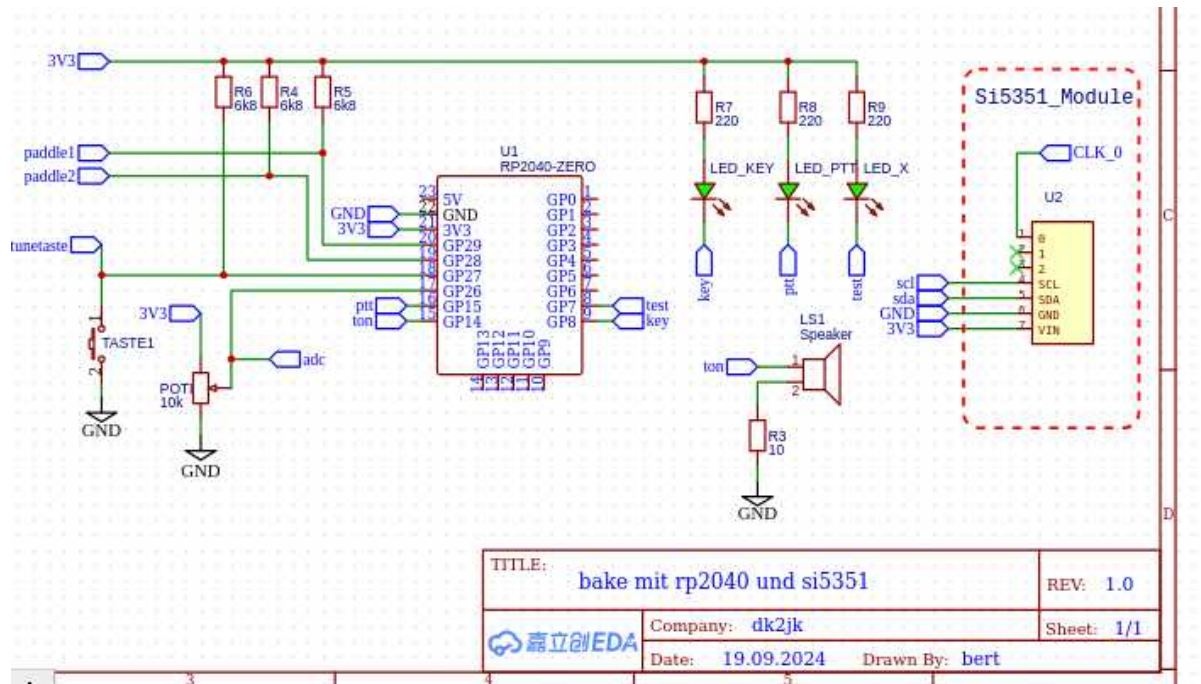
Einbindung des dig. Oszillators si5351
das Modul 'i2c.si5351_jk.py' stellt nur die Funktion
'frequenz(f, clockNr=0)'
zur Verfügung.

Die I2C Routinen sind im Modul 'versteckt'.

Test:

```
>>> import i2c.si5351_jk as si
>>> si.frequenz(7029500)
>>> si.frequenz(7029600)
>>> si.frequenz(0)
, , ,
```

Bake mit Si5351, Schaltbild



Auch hier das Schaltbild.

5

Verweise

- [1] Python für Windows / Linux:
<https://www.python.org/downloads/>
- [2] Micropython für verschiedene CPUs:
<https://micropython.org/>
- [3] Entwicklungsumgebung für Python ,Thonny' :
<https://thonny.org/>
- [4] „Das umfassende Handbuch“ :
<https://openbook.rheinwerk-verlag.de/python/>
- [5] Quick reference for the ESP8266 (englisch)
<https://docs.micropython.org/en/latest/esp8266/quickref.html>
- [6] Quick reference for RP2040 (englisch)
<https://docs.micropython.org/en/latest/rp2/quickref.html>
- [7] Dieses Projekt
https://github.com/dk2jk/anwendungen_micropython



Dieses Projekt ist als Dokument mit zugehörigen Python-Scripts verfügbar:

https://github.com/dk2jk/anwendungen_micropython