Python steuert Telefonvermittlung für Wählscheibentelefone

Photo: Hans Gelke

Analoge Telefonvermittlung mit Raspberry Pi und Python

- Kostengünstige Vermittlungsanlage für elektromechanische Telefone
- Geringer Stromverbrauch im Leerlauf
- Grundplatine unterstützt acht Telefone
- Erweiterung bis zu 64 Teilnehmer
- Töne werden in Software generiert
- Wählscheibe/DTMF wird in Software dekodiert
- Anrufe ins Telefonnetz mit Bluetooth und Smartphone
- Imitieren von Internationalen Signalen
- email: hans@gelke.ch
- GIT: https://github.com/hansgelke/retro_CircuitPython

Motivation

3…

Ziele

- Kostengünstige Vermittlungsanlage für elektromechanische Telefone
- Geringster Stromverbrauch im Leerlauf
- Verbindung von mindestens acht Telefonen
- Erweiterung um jeweils acht Telefone bis zu 64 Teilnehmer
- In jeder Ausbaustufe können immer bis zu acht Teilnehmer gleichzeitig miteinander telefonieren
- Töne werden in Software generiert
- Wählscheibe/DTMF wird in Software dekodiert
- Anrufe ins Telefonnetz mit Bluetooth und Smartphone
- Imitieren von Internationalen Signalen

Inhalt

1. Grundprinzipien der analogen Telefonie
2. Hardware Aufbau
3. Finden des passenden Microcontrollers und Software
4. Software Implementierung: Digitale Ein-Ausgangspins
5. Erzeugung der Weckerspannung mit Pulse Width Modulator
6. Digitale Audiogenerierung
7. Steuerung der Telephone
8. Vervielfältigen der Funktionen mit Objektorientierung
9. Gleichzeitige Überwachung aller Telephone mit Multitasking
10. DTMF Dekodierung in Software
11. Anrufen ins Netzt via Smartphone
12. Zusammenfassung
13. Eure Fragen
14. Appendix

1 Grundprinzipien der analogen Telefonie

Philipp Reis erstes "Telephon"

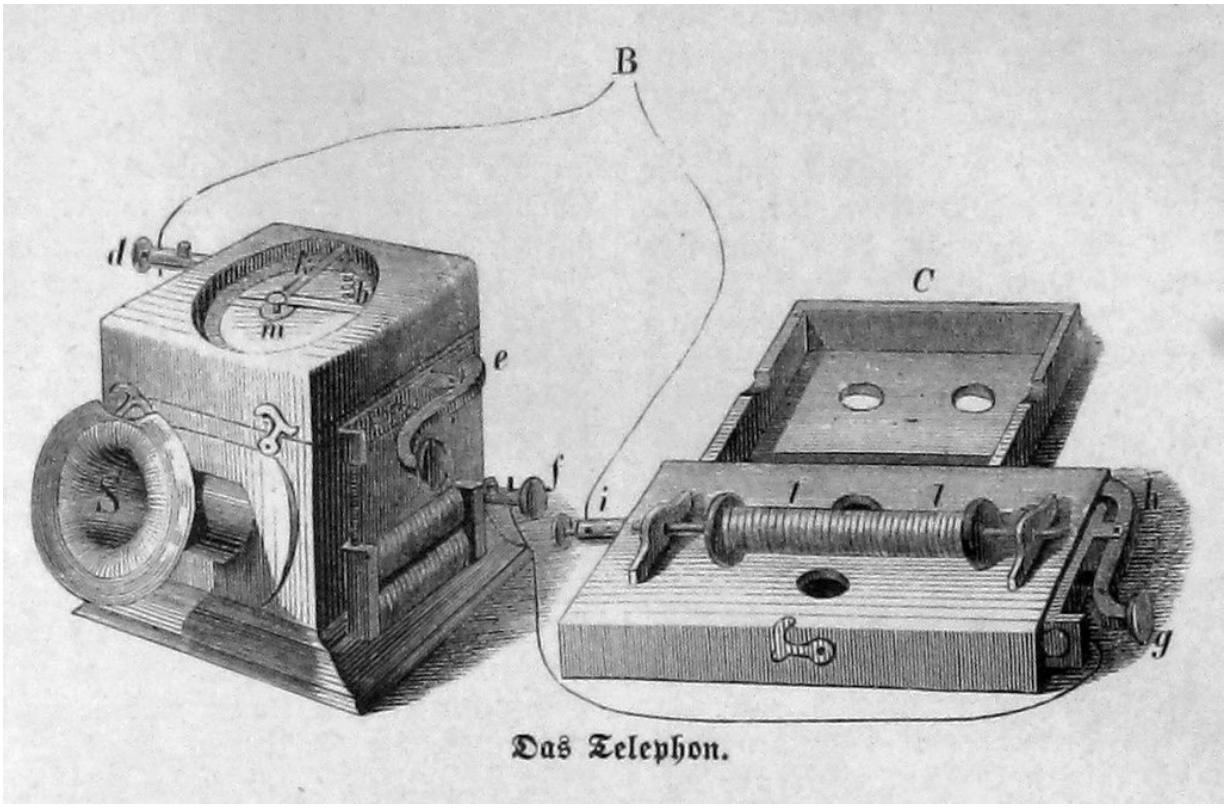


Image: Ernst Keil, from the Book : Die Gartenlaube (1863) Page 809

Philipp Reis erstes "Telephon"

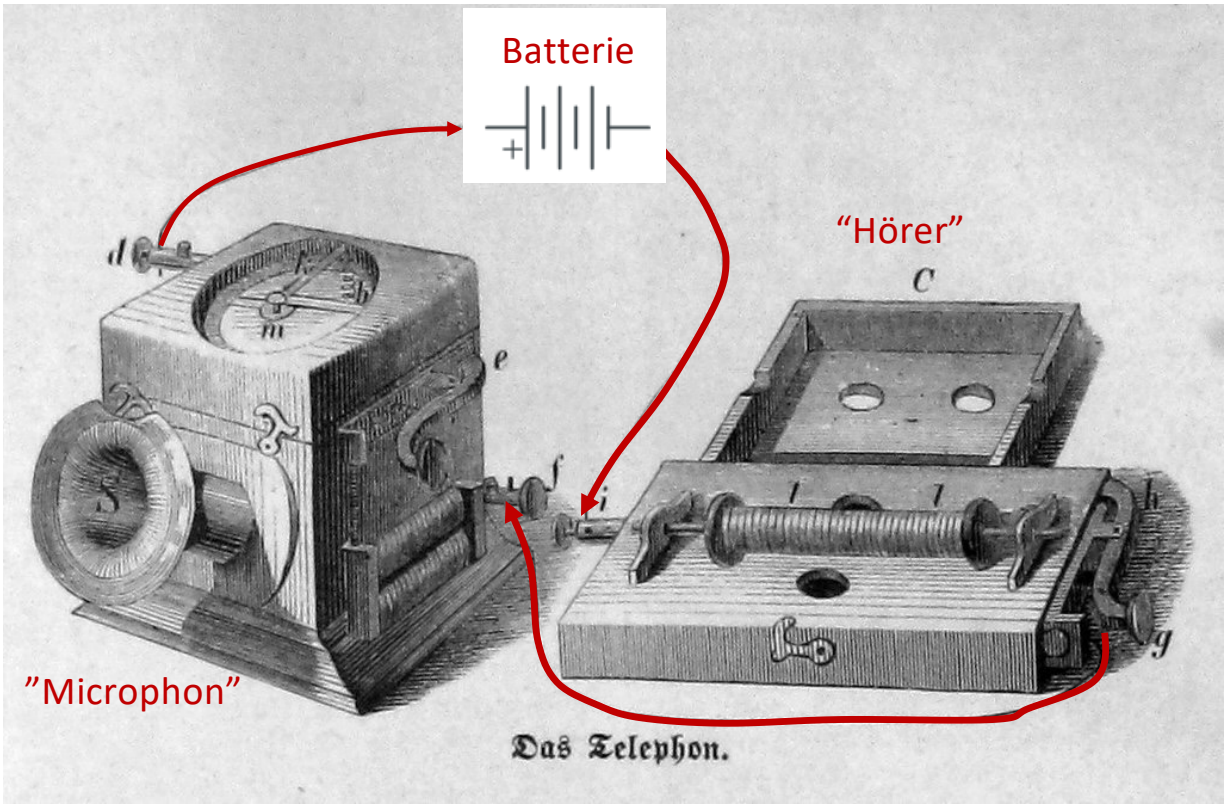
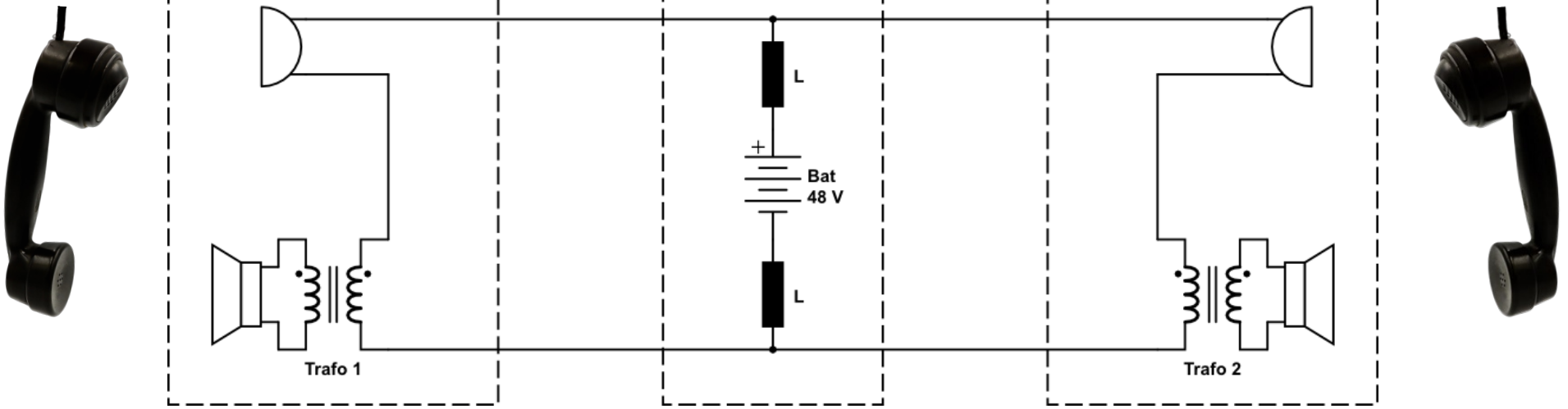


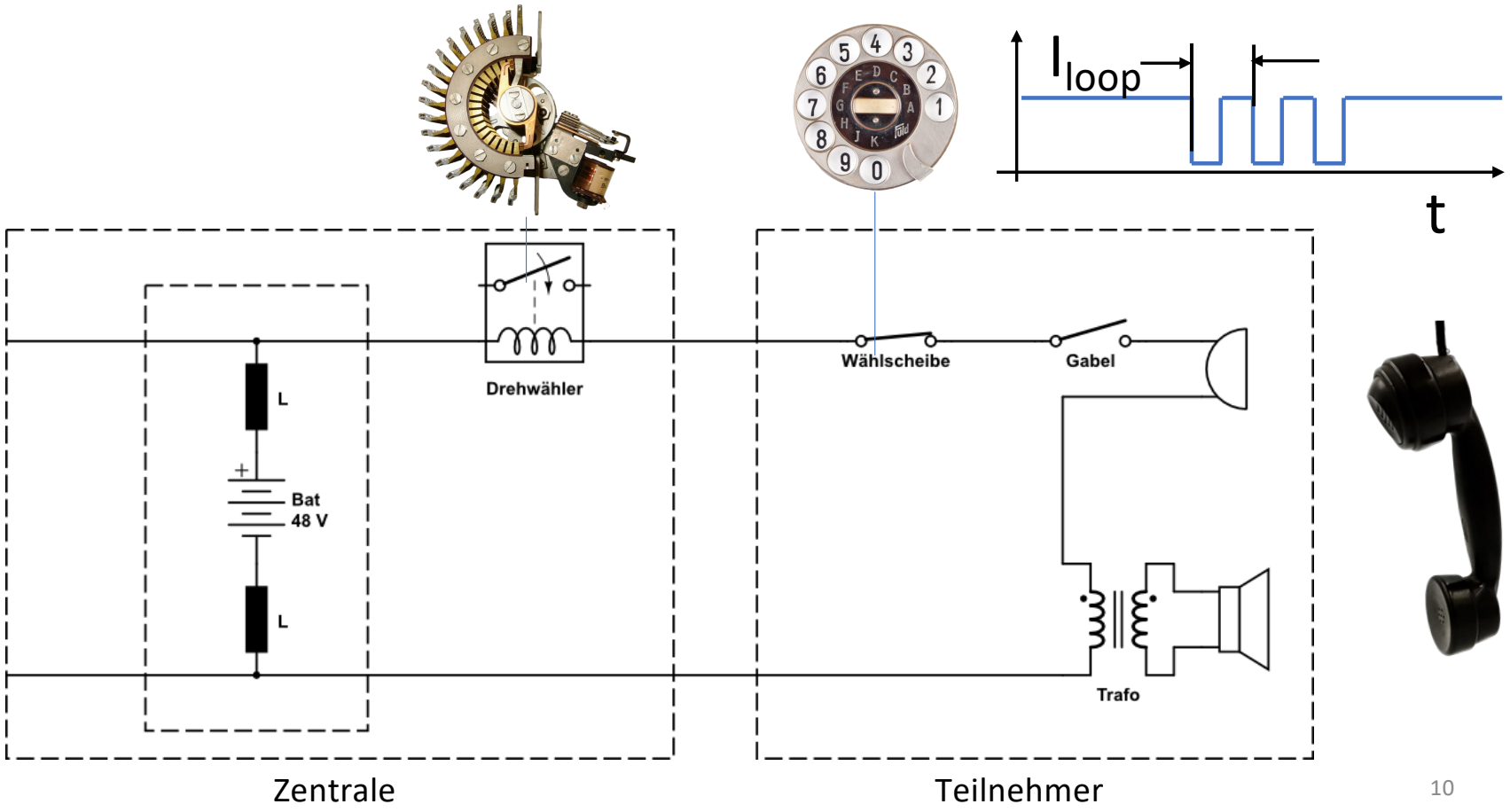
Image: Ernst Keil, from the Book : Die Gartenlaube (1863) Page 809

Sprechstellen mit Zentralbatterie

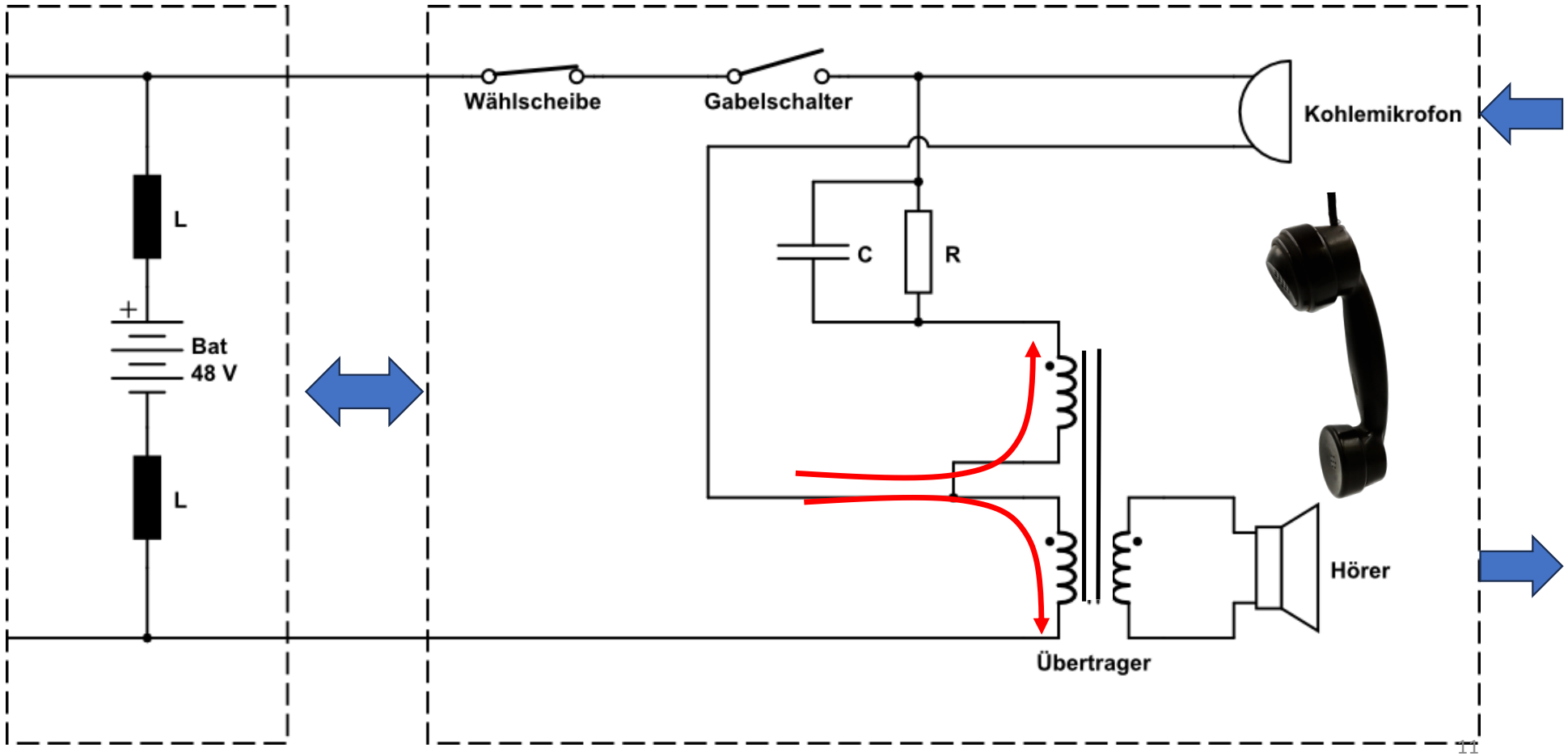


Sprechstelle 1 Zentrale Sprechstelle 2

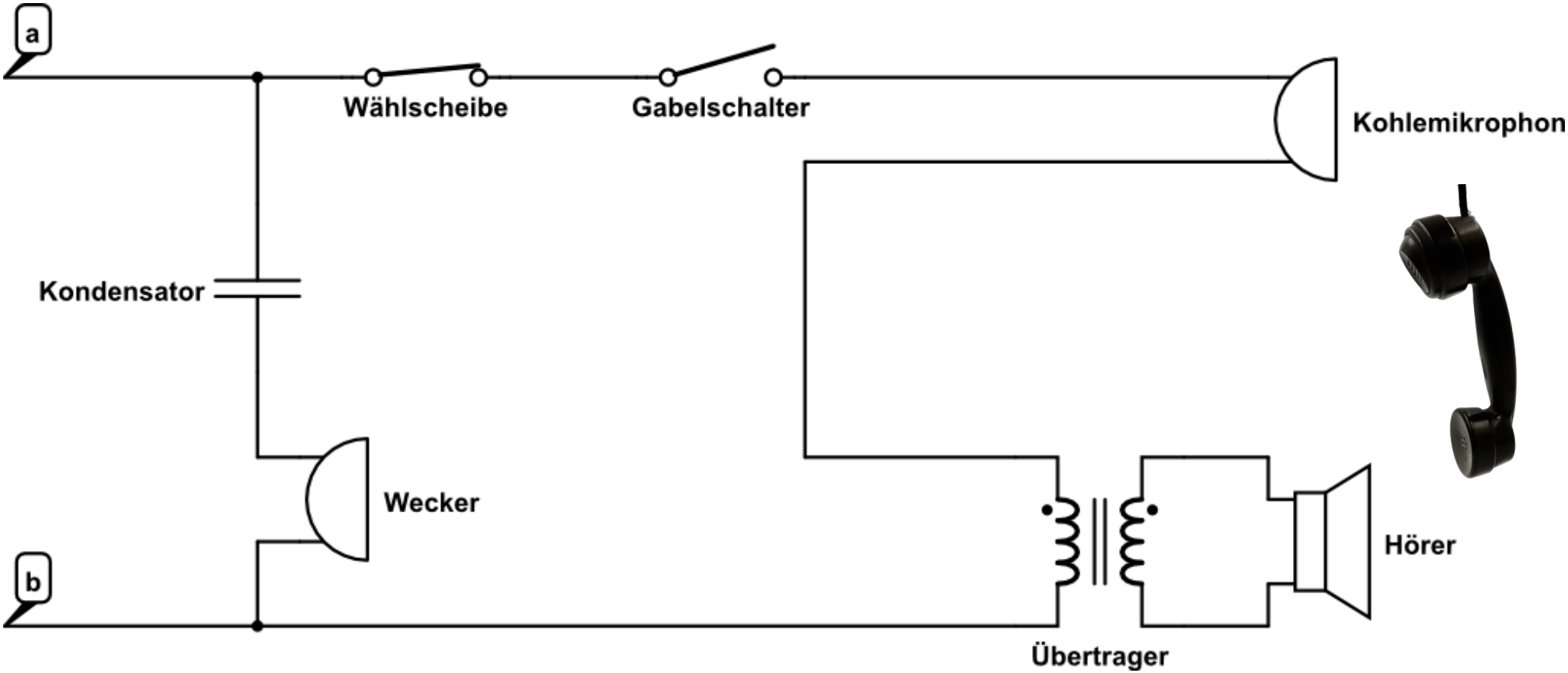
Erfindung des Strowger Drehwählers



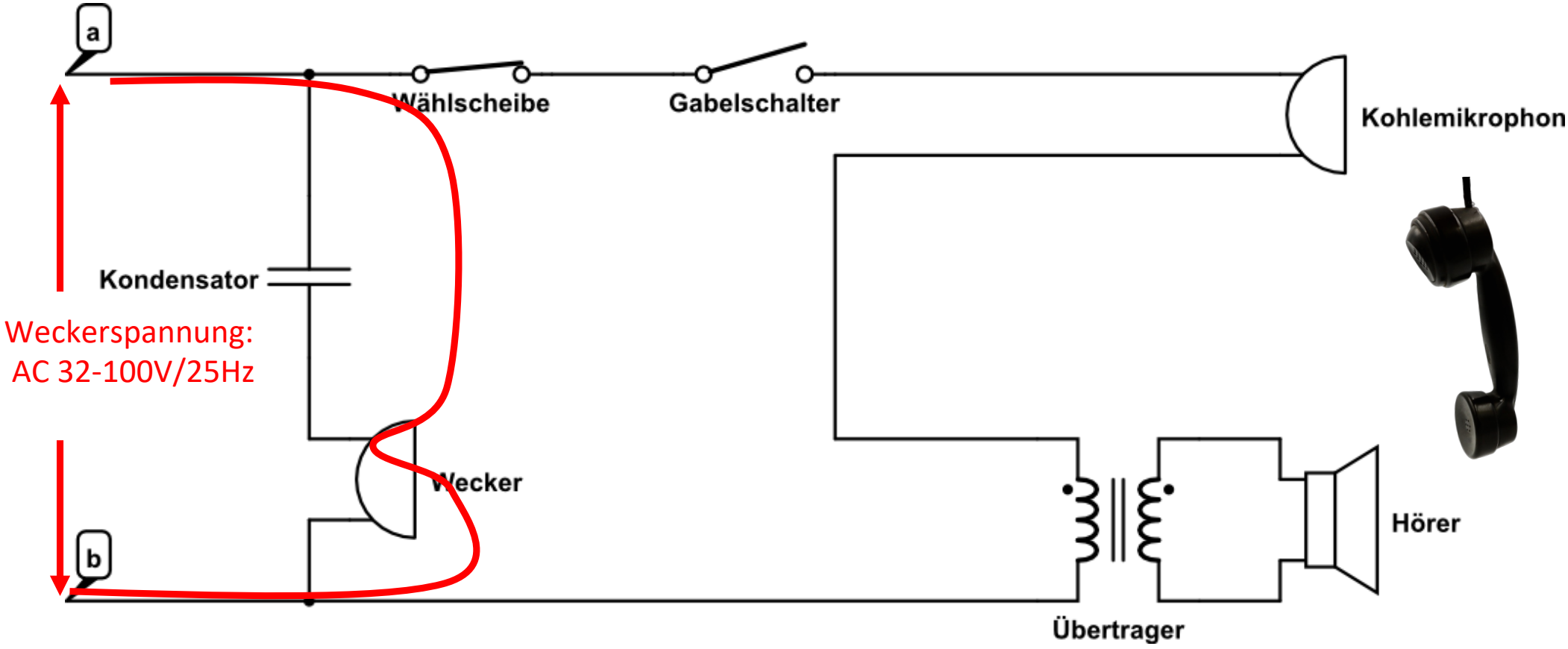
Rückhördämpfung



Bei Anruf

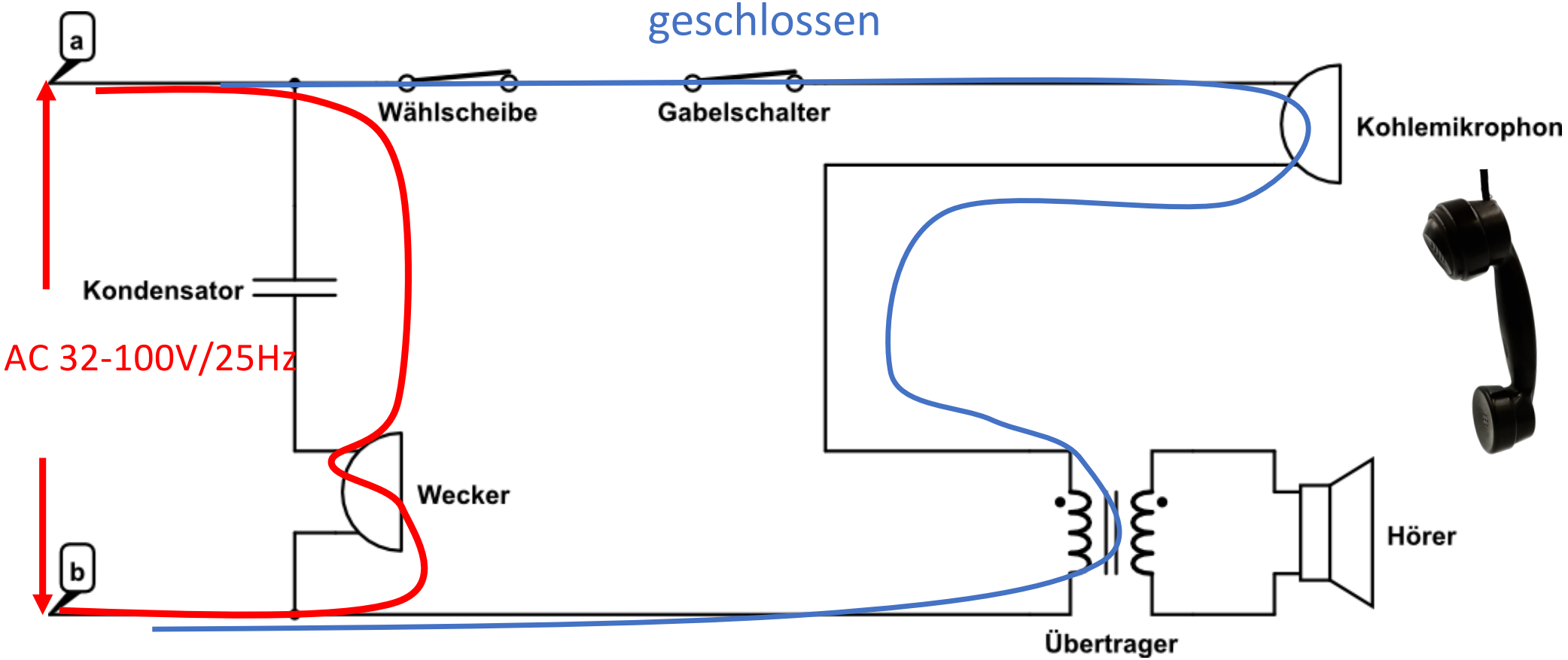

a
Wählscheibe
Gabelschalter
Kohlemikrophon
Kondensator
Wecker
b
Hörer
Übertrager

Bei Anruf


a
Wählscheibe
Gabelschalter
Kohlemikrophon
Kondensator
Weckerspannung:
AC 32-100V/25Hz
Wecker
b
Hörer
Übertrager

Ring Trip

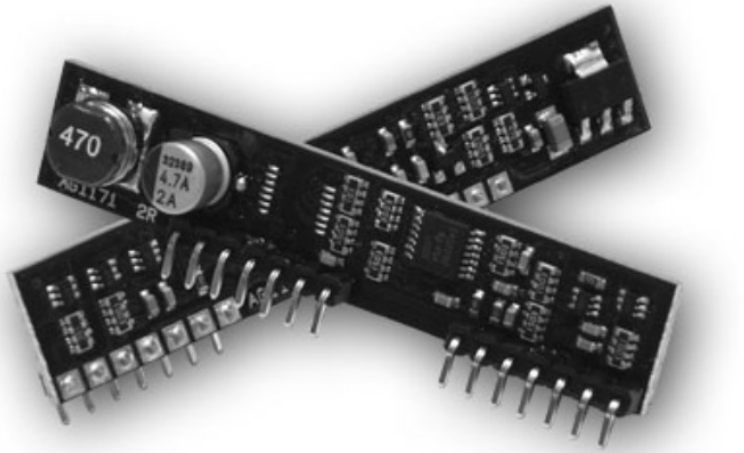
Gleichstromschlaufe
geschlossen



2 Hardware Aufbau

Das Gegenstück in der Zentrale

SLIC: Subscriber Line Interface Circuit



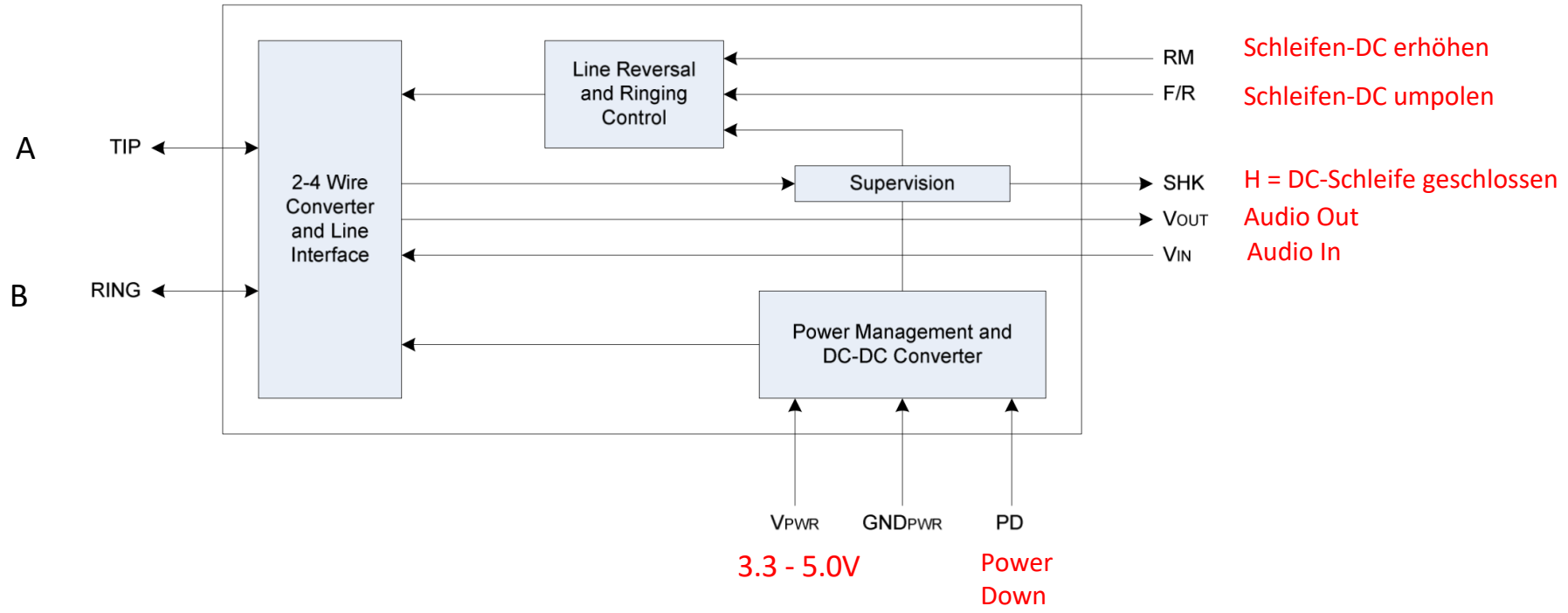
- Erzeugt Batteriespannung: 48V
- Konstanter Schleifenstrom: 30 mA
- Schleifenstrom Detektor mit Entpreller
- Weckerspannung: 65 Vrms
- Schleifenstromerkennung beim Wecken
- Stromsparmodus

Quelle: Silvertel Datenblatt

AG1171 von Silvertel
Beschaffungskosten: CHF 8.80

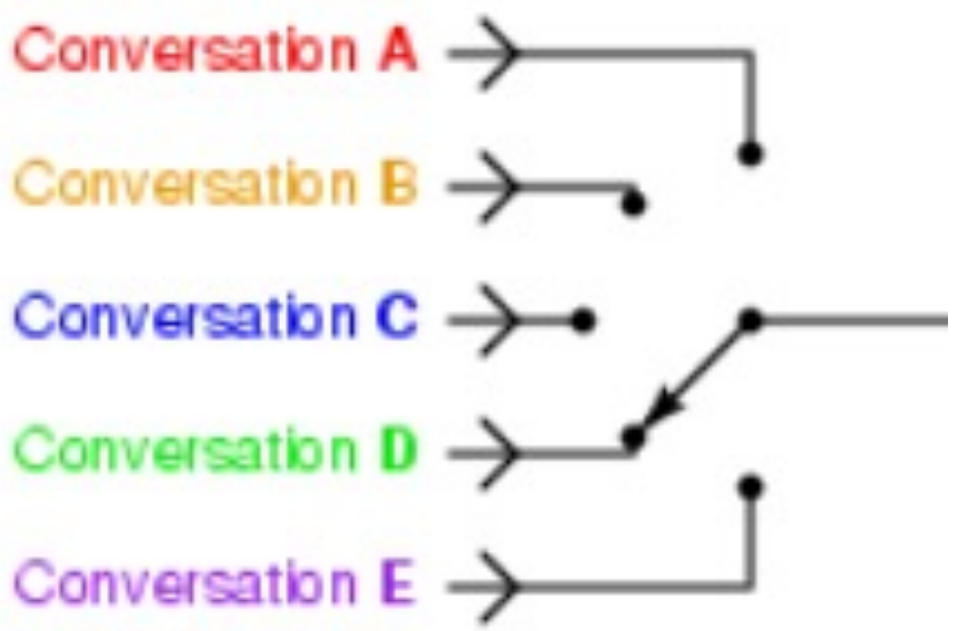
https://silvertel.com/images/datasheets/Ag1171-datasheet-Low-cost-ringing-SLIC-with-single-supply.pdf

SLIC Block Diagramm



Quelle: Silvertel Datenblatt

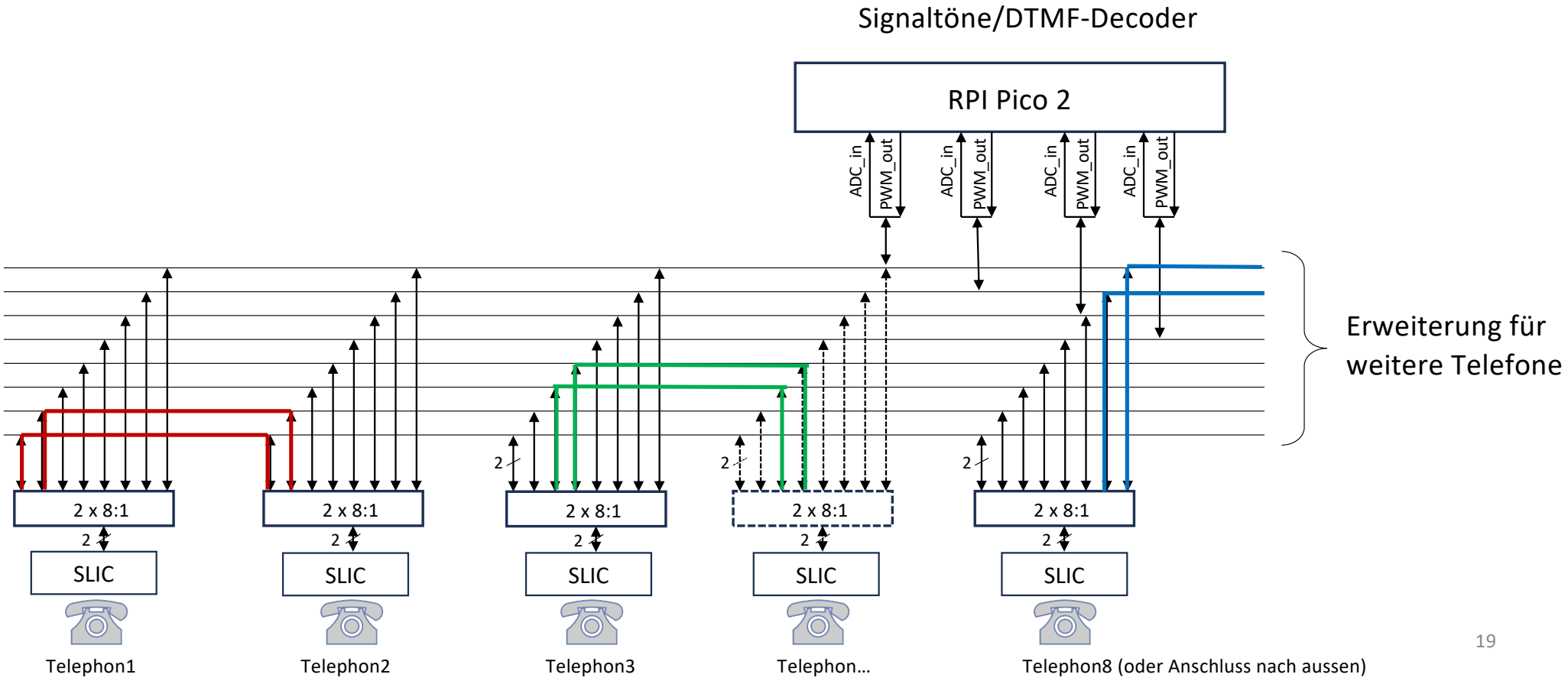
Verbindungsmatrix mit Multiplexer



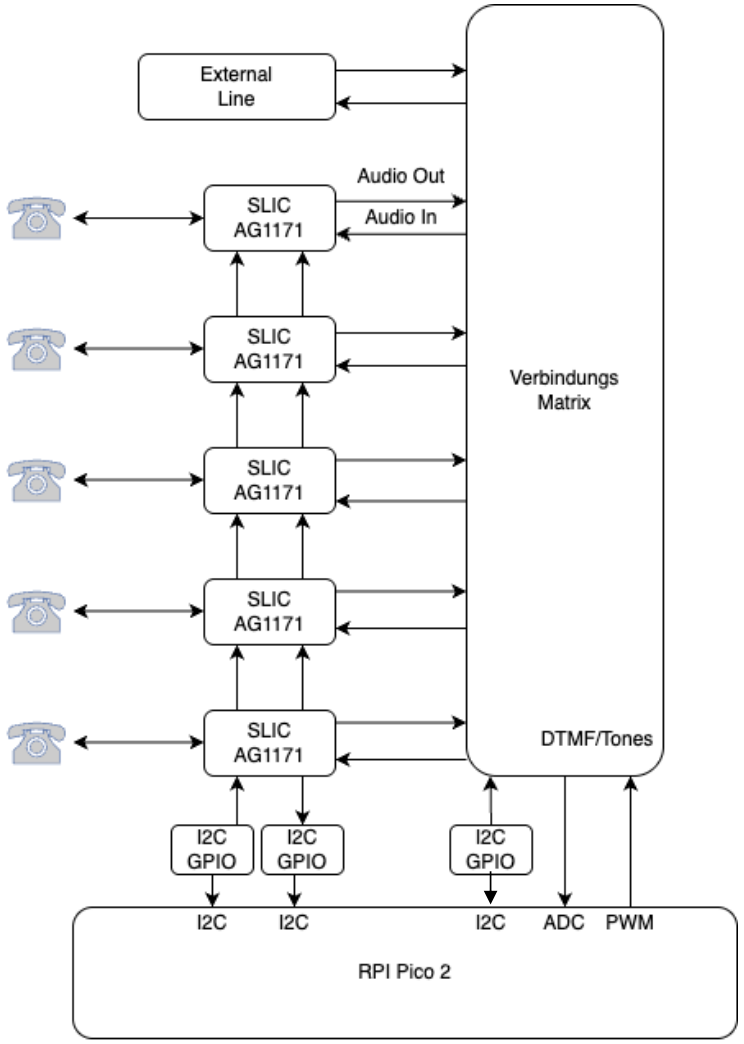
5:1 Multiplexer

Darstellung: Wikipedia Multiplexer

Verbindungsmatrix



Hardware Aufbau



3

Finden des passenden Microcontrollers und Software:

- Welcher Prozessor?
- Welche Programmiersprache?
- Welches Betriebssystem?

Anforderungen an Hardware

- Geringster Stromverbrauch im Leerlauf
- Prozessor muss Audio ausgeben können (kein externer Codec notwendig)
- Mindestens PWM für Signaltöne etc.
- ADC-Wandler für Audioverarbeitung (DTMF-Decoding)
- I2C/SPIO Busse für Pin-Erweiterung
- Wenige und günstige Bauteile (Prozessor < CHF 9)

Anforderung an die Software

- Einfach zum Programmieren
- Freie Entwicklungswerkzeuge
- Ein Prozessor steuert alle Telefone gleichzeitig (Multitaskingfähig)
- Libraries für Audio, I2C, PWM

Welches Software Entwicklungssystem

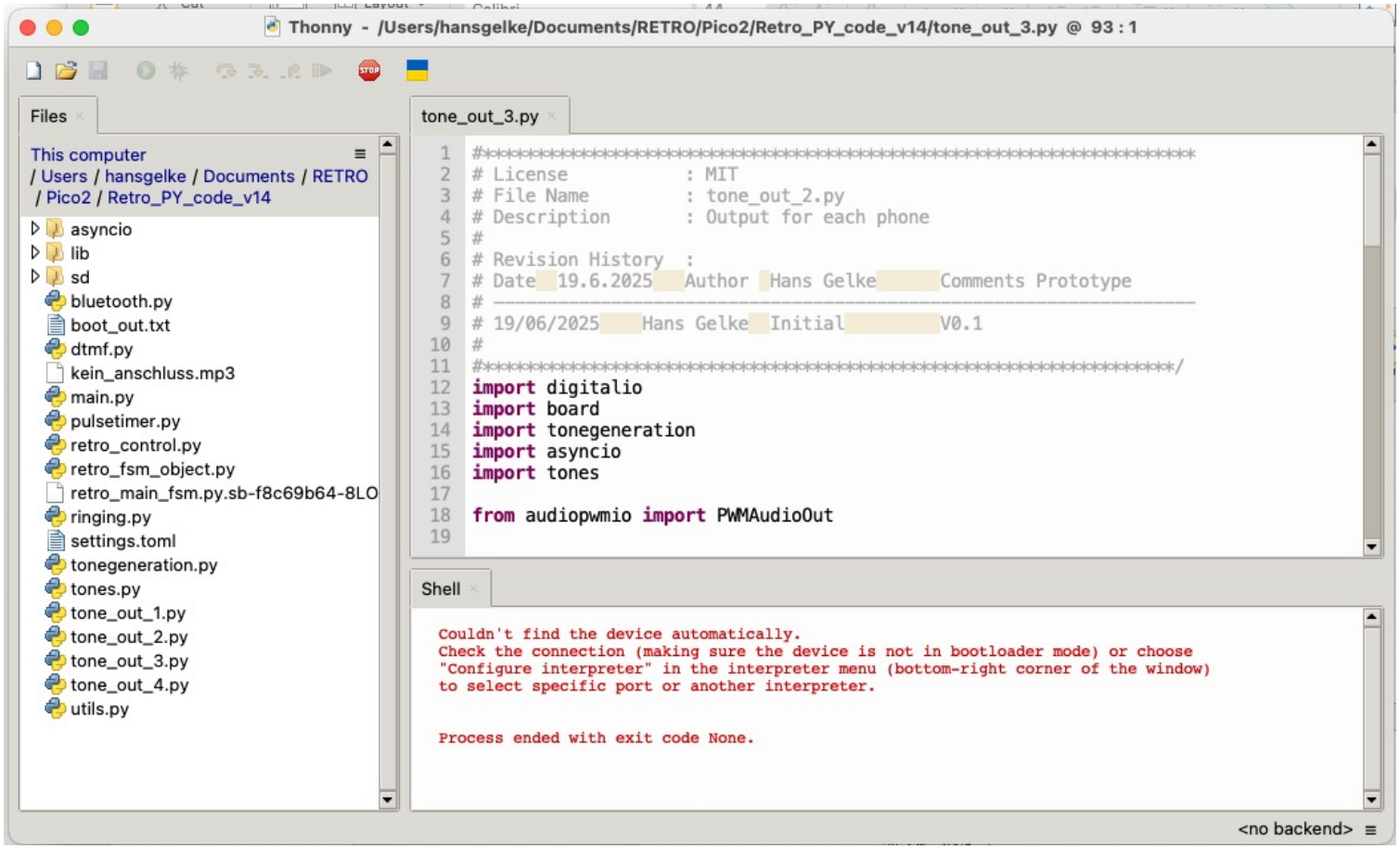
	Kosten	Prozessor	RTOS/Libraries	Programm-entwicklung	Debug
Yocto Linux/C	gratis	Application Prozessor	gut	schwierig	gut
Raspian/Python	gratis	Application Prozessor	gut	einfach	gut
RTOS/C Keil/IAR	hoch	Micro-controller	gut	mittel	gut
Zephir RTOS/C	gratis	Micro-controller	gut	mittel	gut
MicroPython CircuitPython	gratis	Micro-controller	gut	einfach	schlecht

21 u. 22…

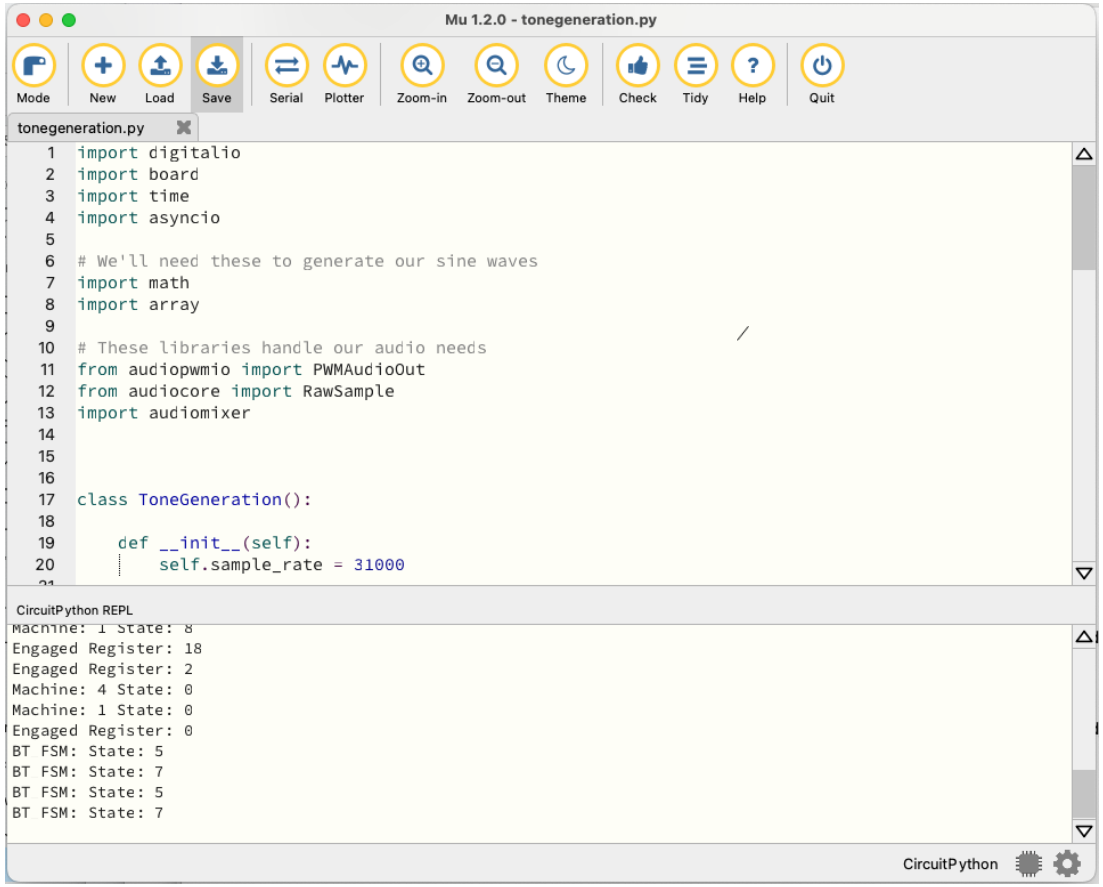
Circuitpython

- Python für Microcontroller
- Anfängerfreundlich
- Viele Libraries (Audio, Multitasking, GPIO, DSP, Objekt Orientierung)
- Gute Boardunterstützung
- Gute Dokumentation:
  `https://docs.circuitpython.org/en/latest/shared-bindings/audiopwmio/index.html`
- Viele Programmierbeispiele
- Community-Unterstützung

Thonny Editor

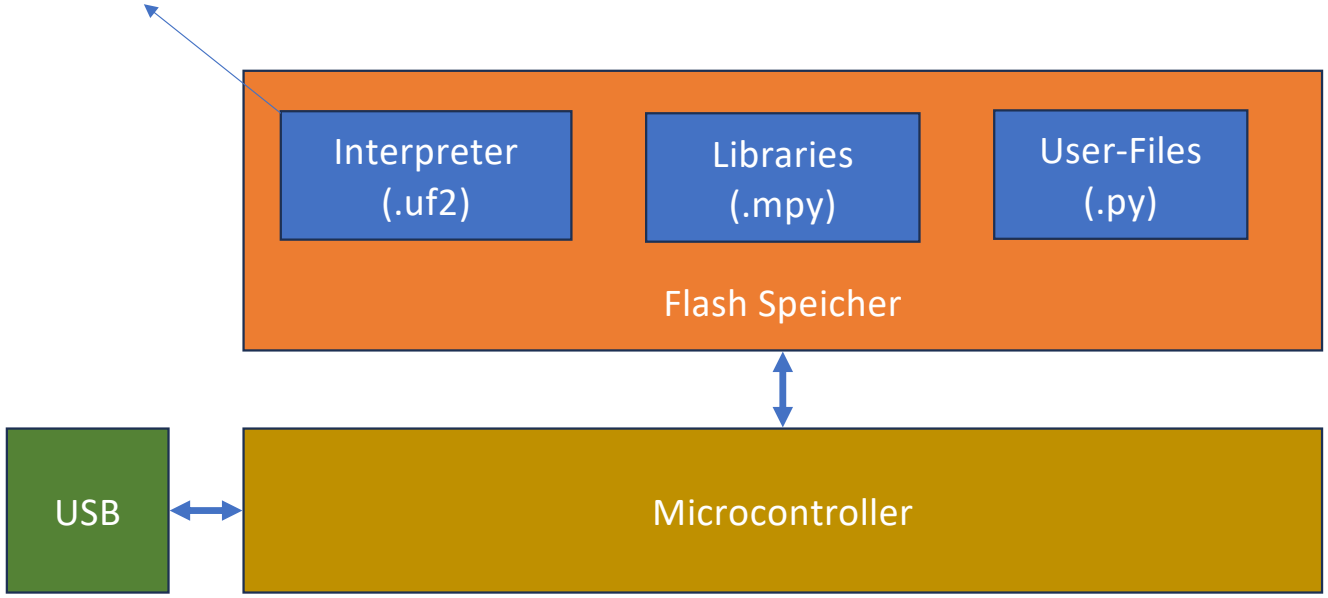


Mu Editor

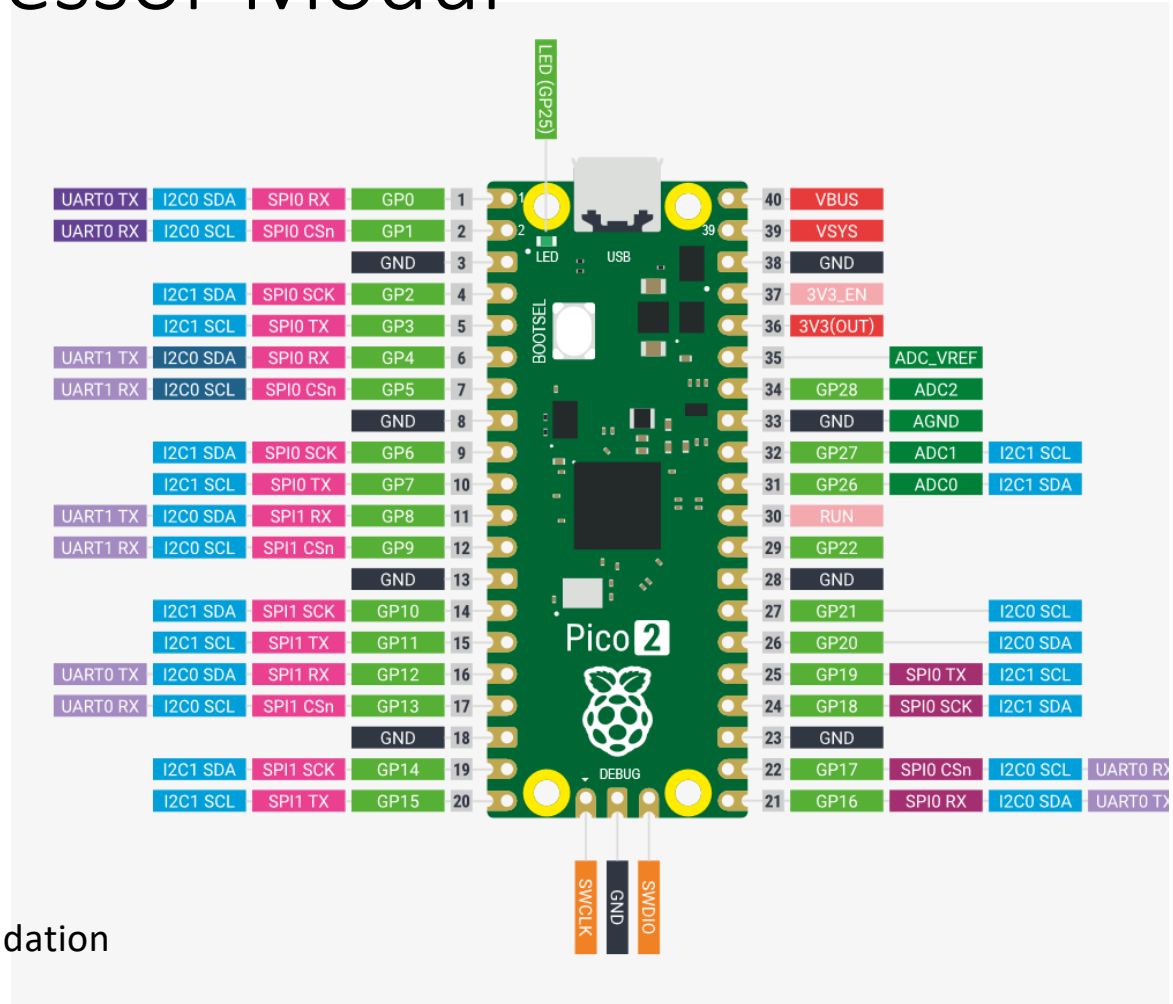

Mu 1.2.0 - tonegeneration.py
Mode
New
Load
Save
Serial
Plotter
Zoom-in
Zoom-out
Theme
Check
Tidy
Help
Quit
tonegeneration.py
1 import digitalio
2 import board
3 import time
4 import asyncio
5
6 # We'll need these to generate our sine waves
7 import math
8 import array
9
10 # These libraries handle our audio needs
11 from audiopwmio import PWMAudioOut
12 from audiocore import RawSample
13 import audiomixer
14
15
16
17 class ToneGeneration():
18
19 def __init__(self):
20 self.sample_rate = 31000
CircuitPython REPL
Engaged Register: 18
Engaged Register: 2
Machine: 4 State: 0
Machine: 1 State: 0
Engaged Register: 0
BT FSM: State: 5
BT FSM: State: 7
BT FSM: State: 5
BT FSM: State: 7
CircuitPython

Interpretor und User Scripts im Flash

https://circuitpython.org/downloads



Pico 2 Prozessor Modul



Bildquelle: Raspberry PI Foundation

Processor on Pico2 Module

- RP2350 Microcontroller
- 2 Arm Cortex M33 @ 150 MHz
- 2 RISC V
- DSP instructions, Floating Point
- 520 kByte SRAM
- 4 MB (RPI Foundation) on-Board-QSPI-Flash (16 MB, Adafruit Pimoroni Pico 2 plus)
- 1 USB 1.1
- 2 UART, 2 SPI, 2 I2C
- 24 Pulse With Modulators
- 1 A/D Wandler: 12-bit 500 ksps SAR (successive approximation) ADC mit 12 Kanälen
- Beschaffungskosten Pico 2 (4MB): CHF 7.90
- Beschaffungskosten Pimoroni (16MB): CHF 13.90

Bildquelle: Raspberry PI Foundation

RP2350 is Processor on Pico 2

- 2 Arm Cortex M33 @ 150 MHz
- 2 RISC V
- DSP instructions, Floating Point
- 520 kByte SRAM
- 1 USB 1.1
- 2 UART, 2 SPI, 2 I2C
- 24 Pulse With Modulators
- A/D Wandler: 12-bit 500 ksps SAR (successive approximation) ADC
- $0.80 (Verbaut auf Pico 2 Modul)

Software Implementierung:

4 Digitale GPIO

Digital IO – Ansteuern einer LED

```
import board
import digitalio

led_1 = digitalio.DigitalInOut(board.GP3)
led_1.direction = digitalio.Direction.OUTPUT
led_1.value = False
…
led_1.value = True
```

Descriptor

Library

Funktion

Pin Beschreibung,
wo LED angeschlossen

Digital IO – Auslesen eines Schalters

```
import board
import digitalio
```

Pin Beschreibung,
wo Schalter angeschlossen

Pin Beschreibung,
wo LED angeschlossen

Descriptor Library Funktion

```
sw_1 = digitalio.DigitalInOut(board.GP4)
sw_1.direction = digitalio.Direction.INPUT
<True/False> = sw_1.value
```

Kontaktentprellung

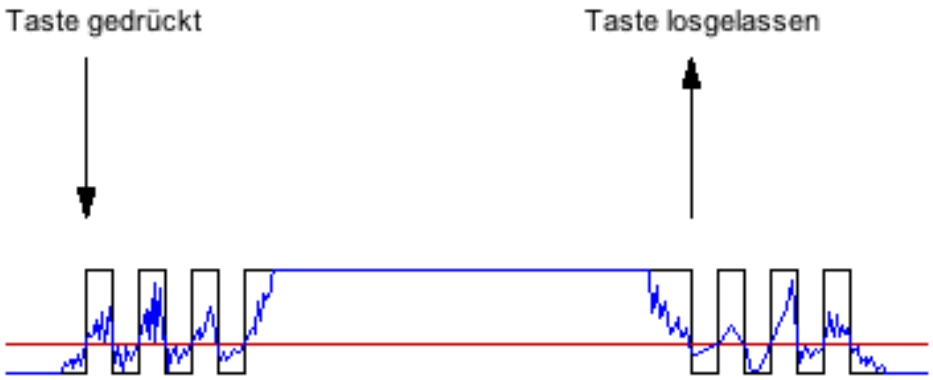


Abbildung: Microcontroller.net

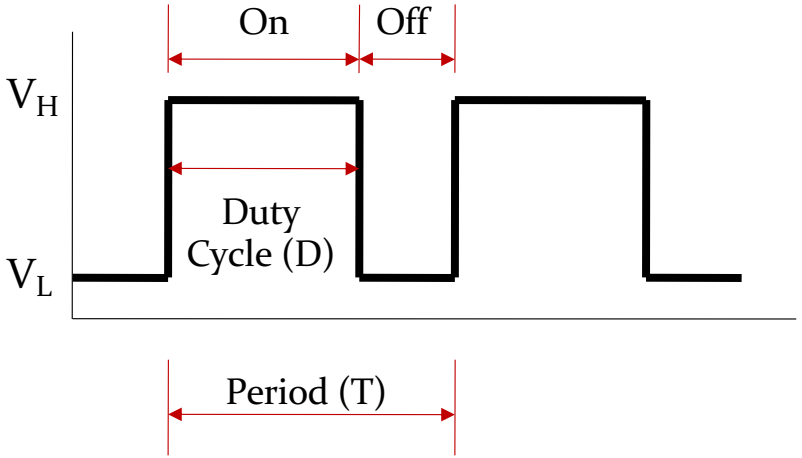
```
from adafruit_debouncer import Debouncer

switch_debounced = Debouncer(DigitalInOut(board.GP5))
```

Software Implementierung:

5 Erzeugung der Weckerspannung mit Pulse Width Modulator

Pulse-Width-Modulation (PWM)

- PWM Signale sind digitale Signale mit einer definierten Frequenz und Pulsbreite



Zur Vereinfachung: V_L = 0V

Anwendung:

- Dimming LED mit Variable on / off
- Digital/Analog-Converter (DAC)
- Duty Cycle wird in % angegeben

Darstellung: ZHAW

Digital IO – PWM

```
import board
import pwmio

pwm1 = pwmio.PWMOut(board.GP18,frequency=20,duty_cycle=65535)

pwm1.duty_cycle = 32767
```

GPIO Pin der PWM Ausgabe

Default Duty Cycle

(100%)

(50%)

Software Implementierung:

6 Digital Audiogenerierung


Dial_US

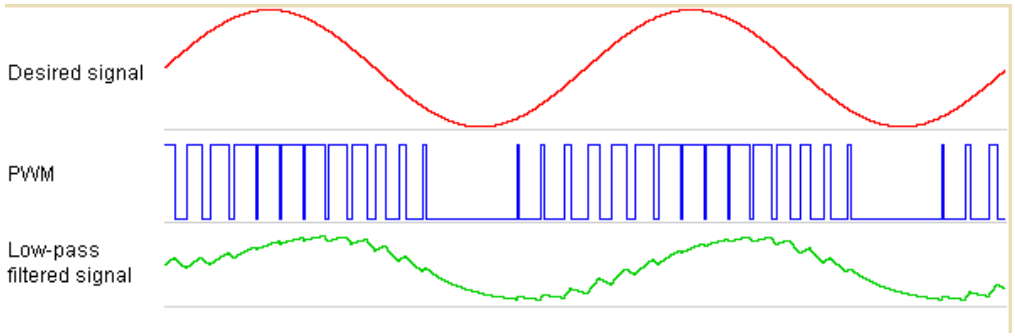

Dial_UK


Ring_UK_US

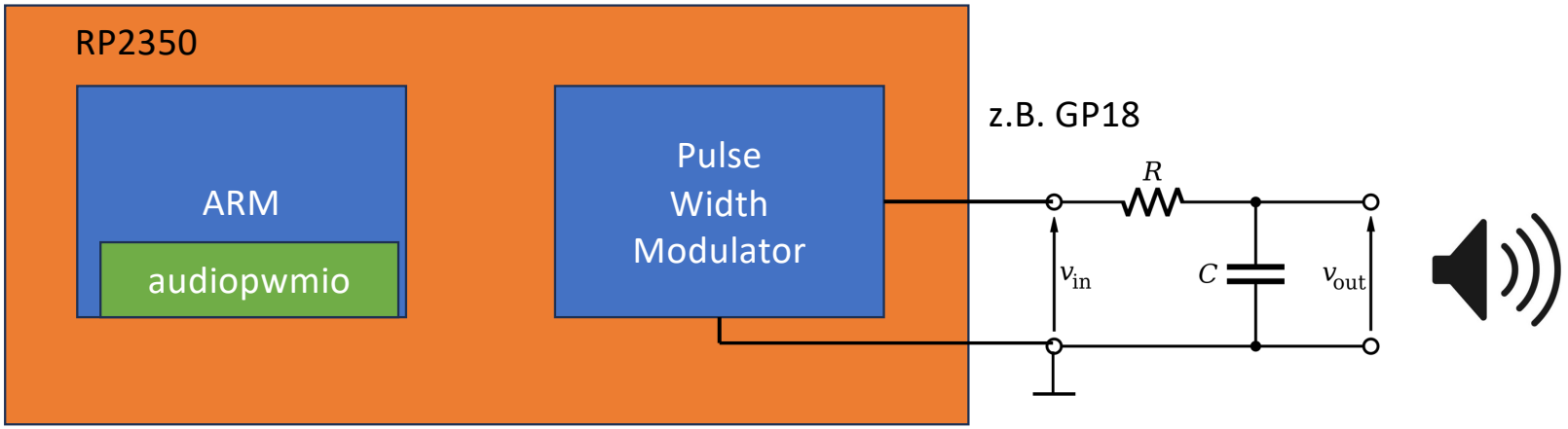

Kein Anschluss


DTMF

Ausgabe von PWM Audio

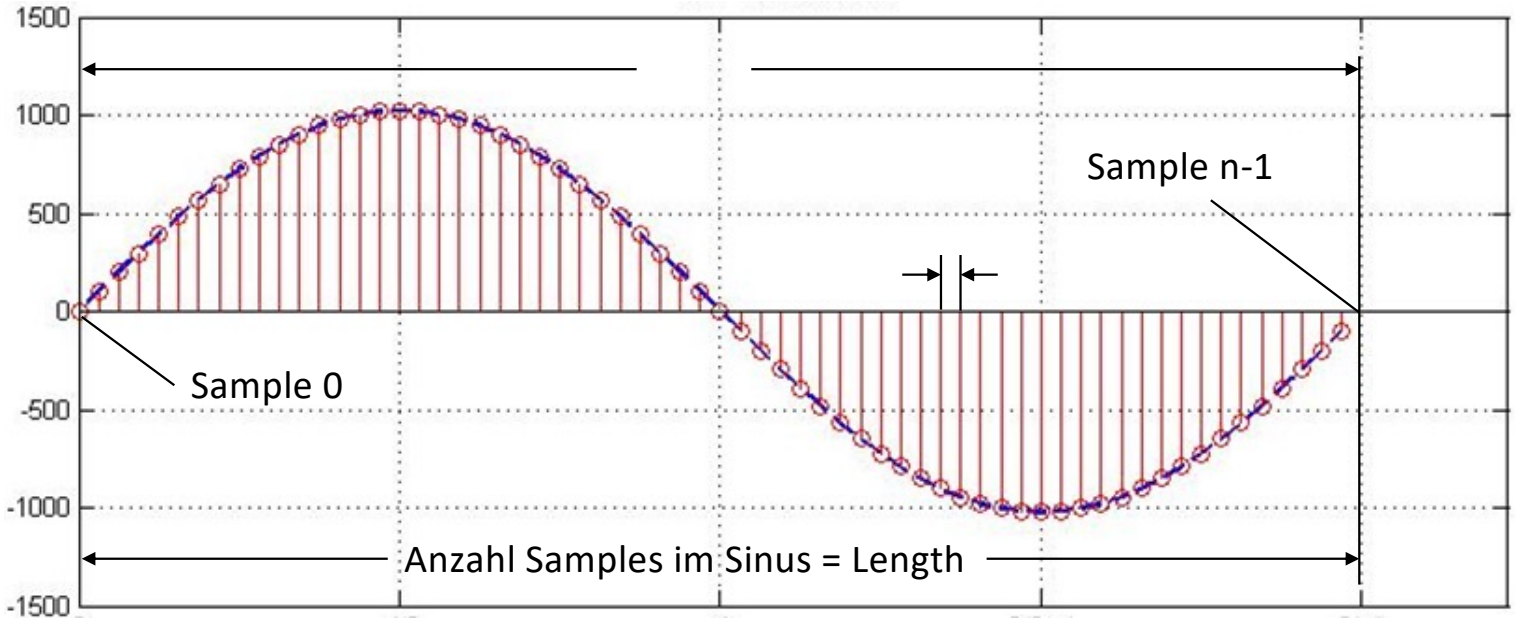


Durch dynamische Variierung des
Duty-Cyles lässt sich ein Sinus erzeugen



Bildquelle: eeguide.com and Wikipedia User: inductiveload

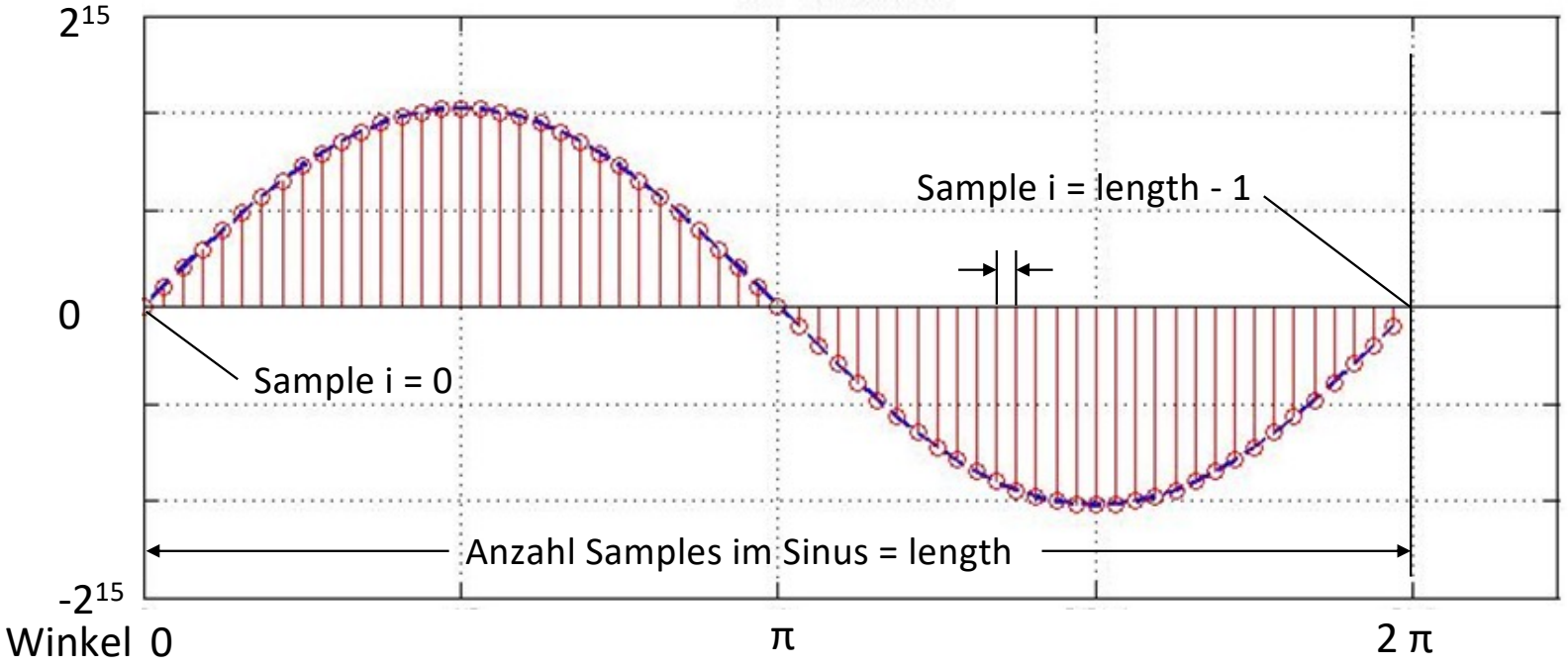
Frequenz eine synthetisierten Sinus



f_s= Audio Sample Rate
Length = Anzahl Samples im Sinus
f_{sig} = Frequenz des Signales

$$f_{sig} = \frac{fs}{Length}$$

Frequenz eine synthetisierten Sinus




```
for i in range(length):
    sine_wave[i] = int(math.sin(math.pi * 2 * i / length) * (2 ** 15) + 2 ** 15)
```

Normalisierte Amplitude (-1 .. +1)

32768

- 32768 … +32 768

Da “DA Wandler” keine negative Zahl kennt 0..65535

Sinus Generation

```
frequency = 440
sample_rate = 8000
length = sample_rate // frequency
sine_array = array.array("H", [0] * length)
for i in range(length):
    sine_array[i] = int(math.sin(math.pi * 2 * i / length)) * (2 ** 15)) + 2 ** 15)
audio_out = audiopwmio.PWMAudioOut(board.GP18)
sine_wave = audiocore.RawSample(sine_array, sample_rate = sample_rate)
audio_out.play(sine_wave, loop=True)
await asyncio.sleep(5)
audio_out.stop()
```

16-bit

Descriptor ID

Verwendeter GPIO als Audio Ausgang

Start Play Loop

Array Übergabe

Stop Play Loop

https://docs.circuitpython.org/en/latest/shared-bindings/audiopwmio/index.html

Audio Dateien Abspielen

```
import audiomp3
import audioio
from audiopwmio import PWMAudioOut
from audiocore import RawSample

audio_out = PWMAudioOut(board.GP26)

mp3 = audiomp3.MP3Decoder("melody.mp3")
if not audio_out.playing:
        audio_out.play(mp3)
        await asyncio.sleep(5)
        audio_out.stop()
```

MP3 Decoder

Datei die abgespielt werden soll

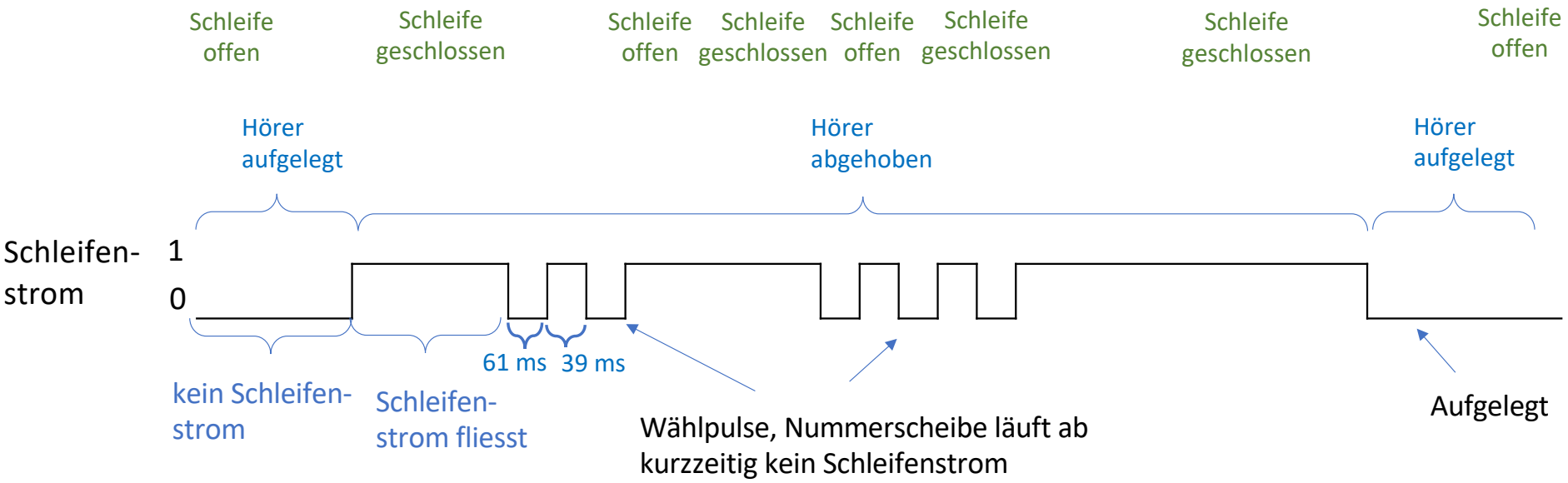
Die Pipeline wird in «Play» Zustand gebracht

https://docs.circuitpython.org/en/latest/shared-bindings/audiomp3/

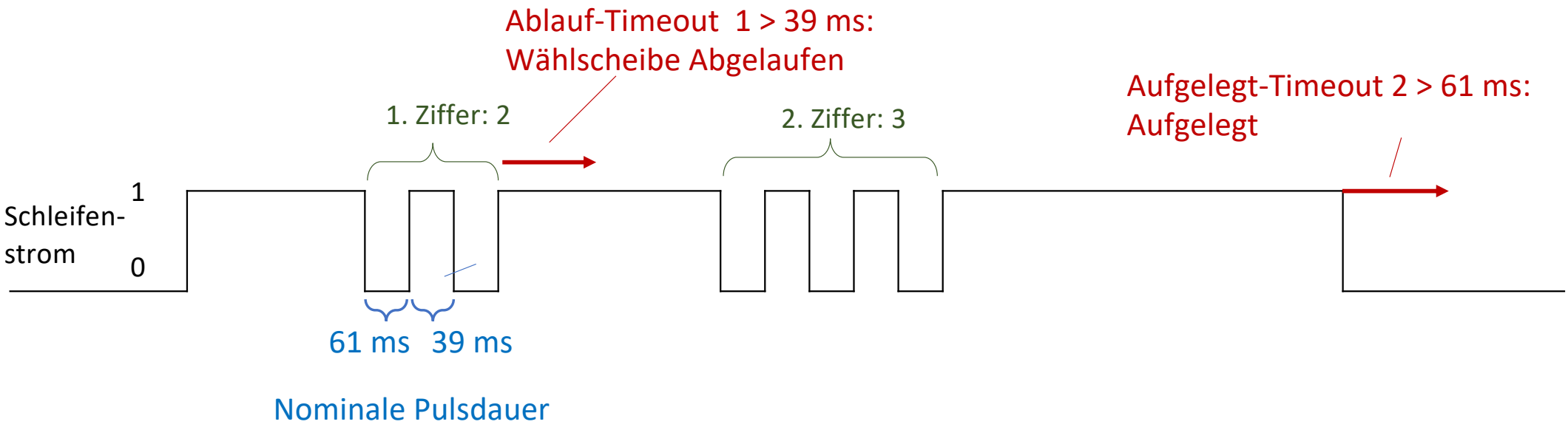
Software Implementierung:

7 Steuerung der Telephone

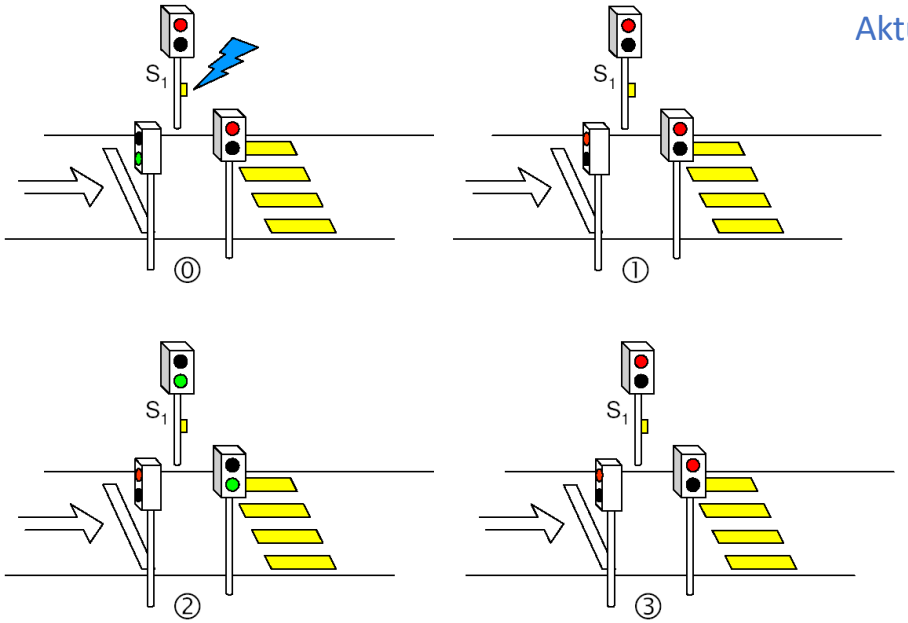
Beispiel: Wählen von zwei Ziffern



Beispiel: Wählen von zwei Ziffern



Was ist eine Zustandsmaschine oder FSM



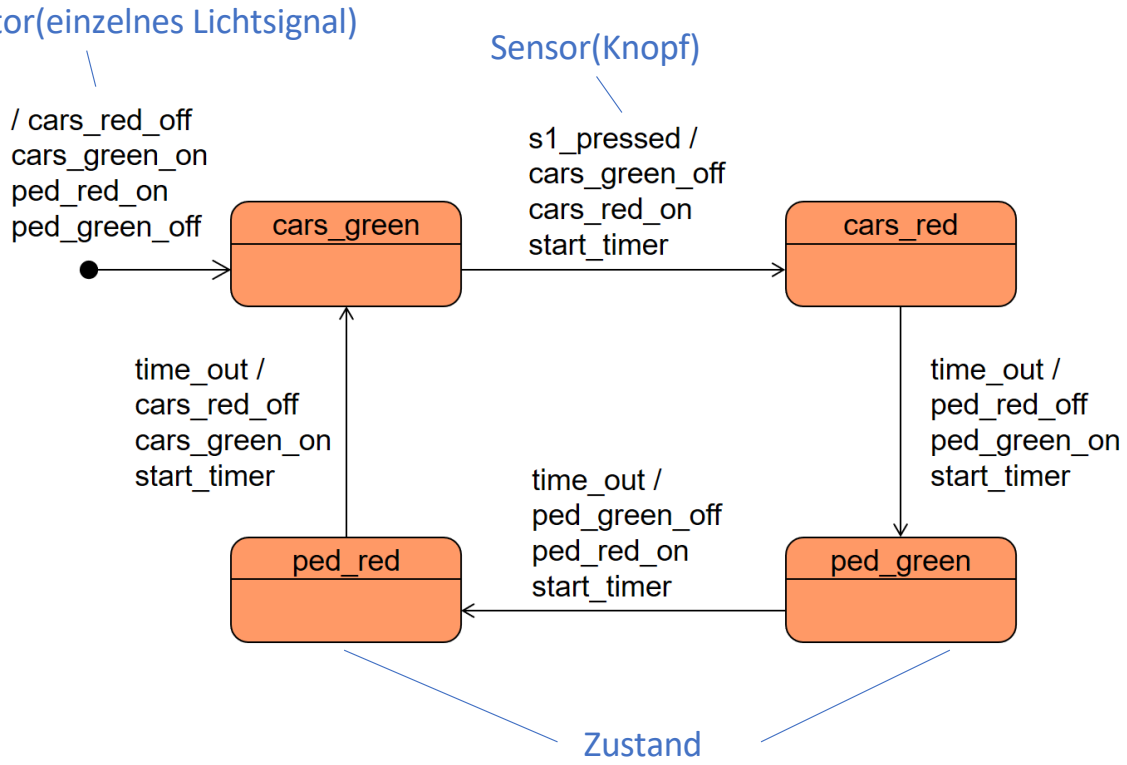
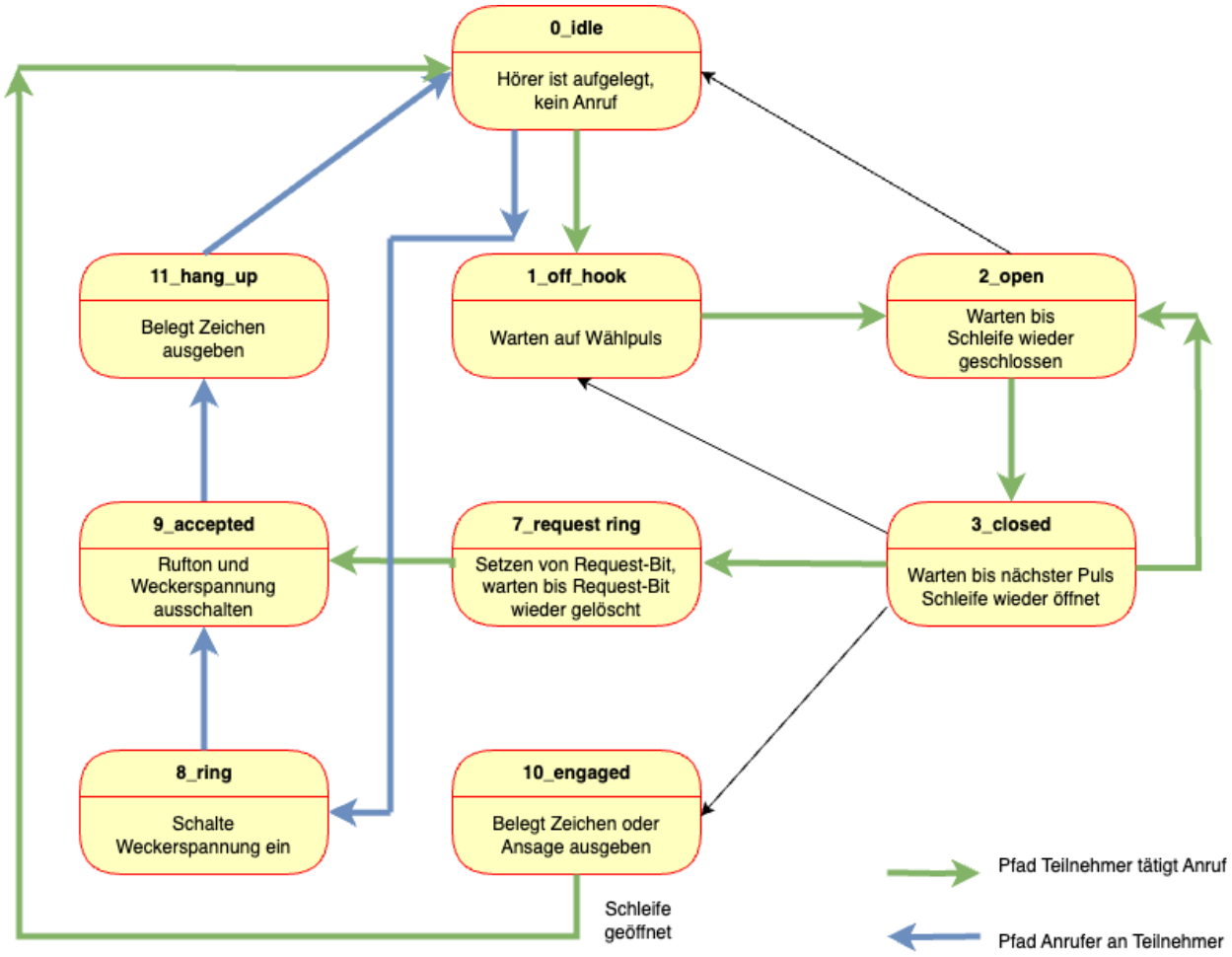
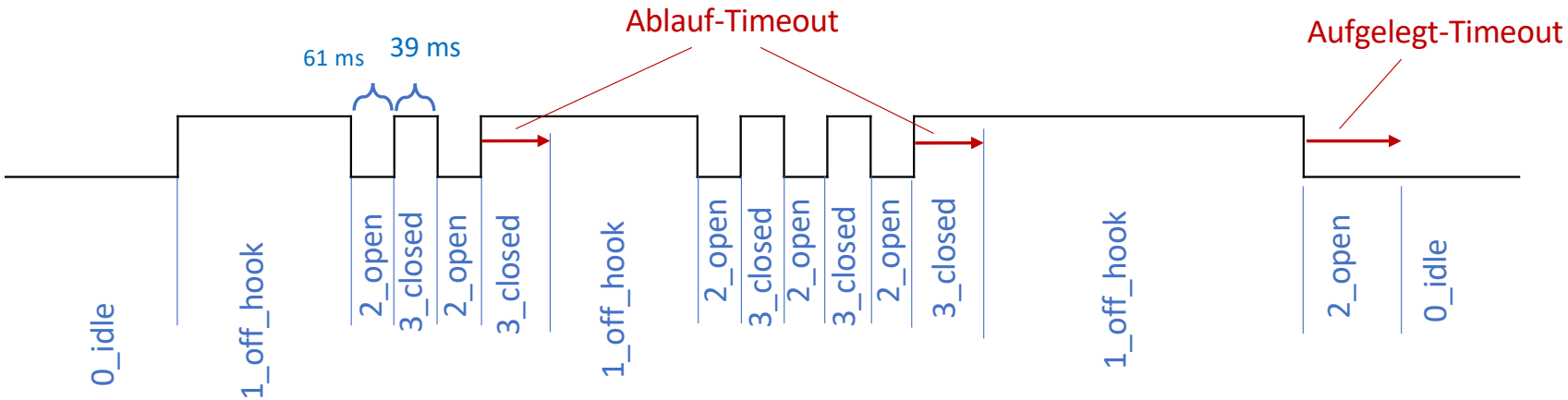


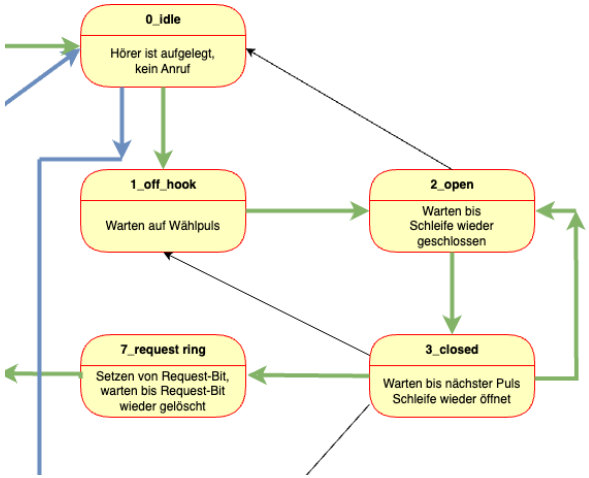
Image: ZHAW Computer Engineering

Eine FSM für jedes Telephon



FSM Zustände beim Wählen von zwei Ziffern





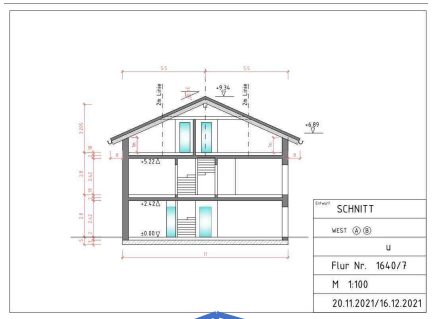
Software Implementierung:

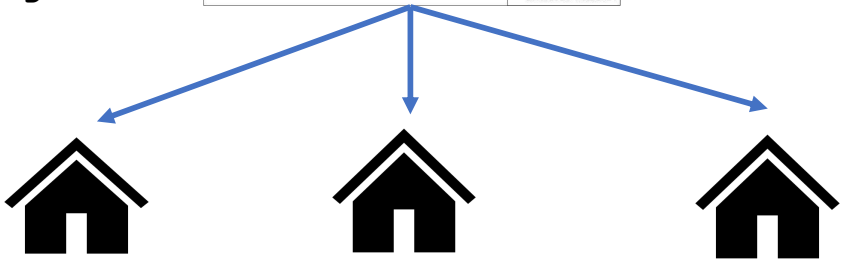
8 Vervielfältigung der FSM mit Objektorientierung

Object Orientation in Python

```
class EFH():
        def __init__(self):
                self.anstrich = weiss
                self.voltaik = false
                self.heizung = gas

        def ausbau(self,anstrich,voltaik,heizung):
                self.anstrich == rot:
                        bestellung = rote_farbe
                …
```

```
class EFH():
```





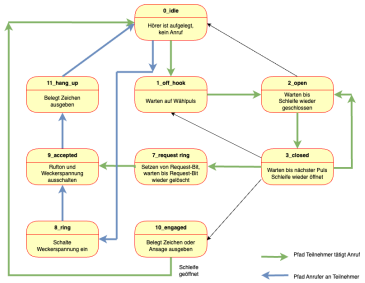
```
haus_1 = EFH()   haus_2 = EFH()   haus_3 = EFH()
```

```
haus_1.ausbau(blau,true,wp)
haus_2.ausbau(grün,false,wp)
haus_3.ausbau(rot,true,oel)
```

Main FSM als Class Beschrieben

```
class MainFSM():
        def __init__(self):
                self.machine = 0
                self.pulse_count = 0
```

```
class MainFSM():
```



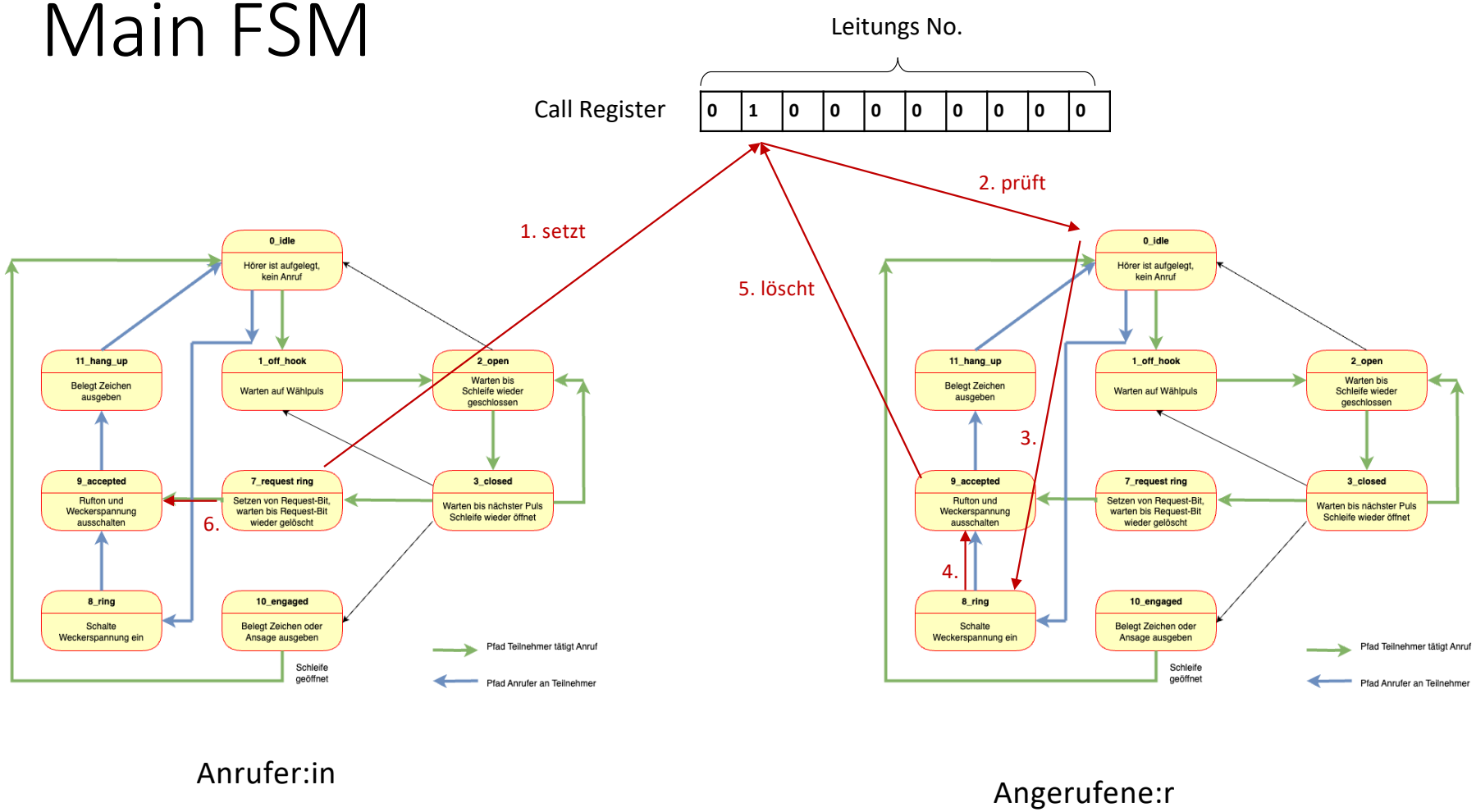
```
tel_1 = MainFSM()
```

```
tel_2 = MainFSM()
```

```
tel_3 = MainFSM()
```

```
tel_4 = MainFSM()
```

Main FSM



Anrufer:in

Angerufene:r

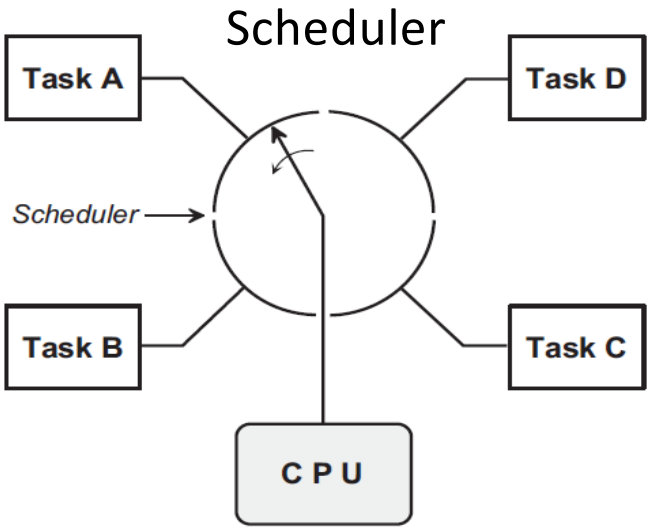
Software Implementierung:

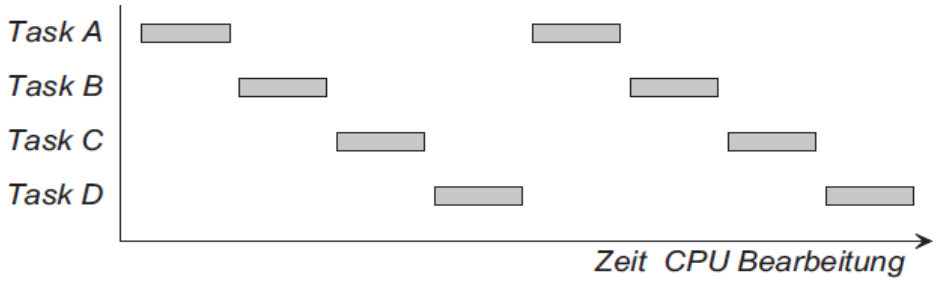
9 Gleichzeitige Überwachung aller Telephone mit Multitasking

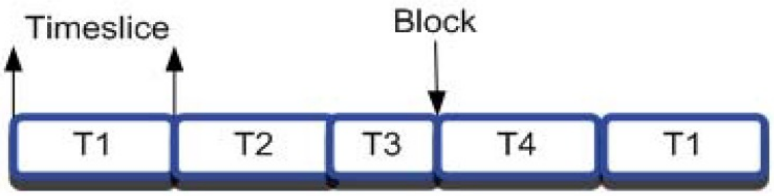
Multitasking

- Statusbeobachtung und Steuerung aller Telefone
- Gleichzeitige Erzeugung von verschiedenen Tönen
- DTMF Dekodierung
- Erzeugung Weckerwechselspannung (25Hz)
- Erzeugung der Timerperioden

Multitasking







Cooperative Multitasking

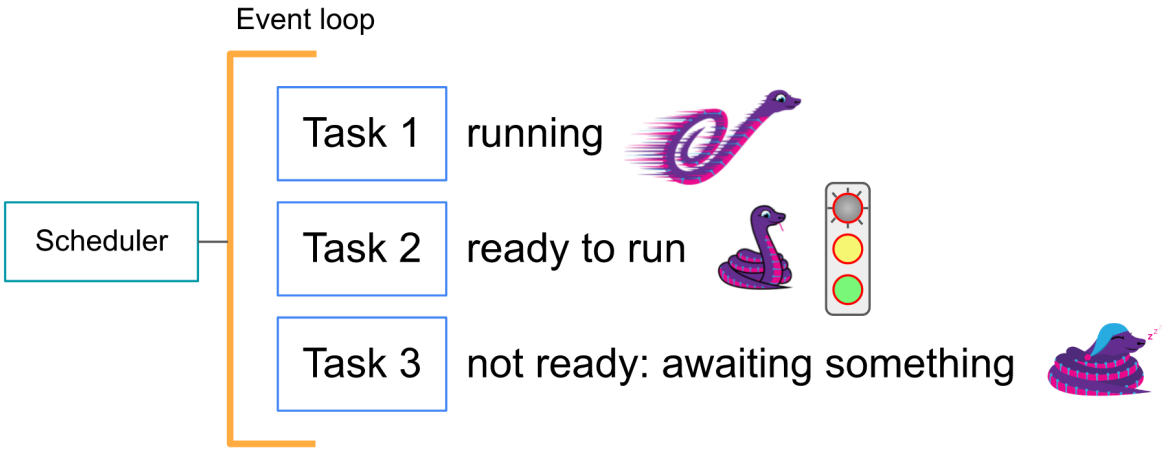
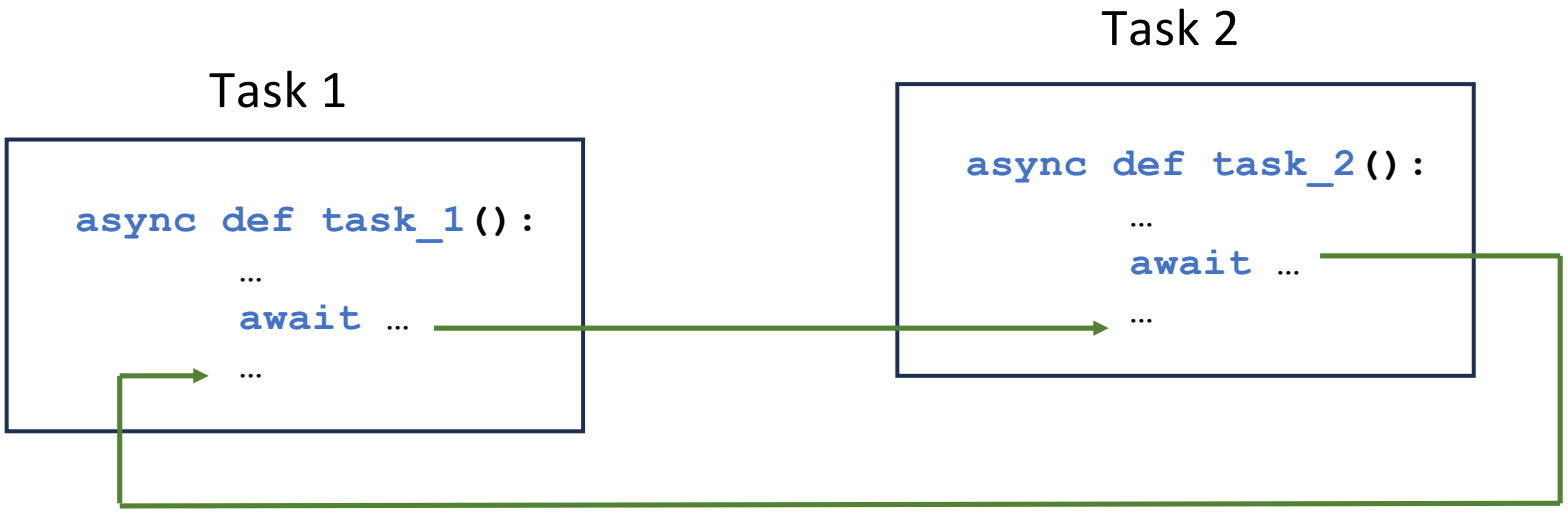


Abbildung: Dan Halbert, Adafruit

- Tasks laufen abwechselnd
- Zu einer bestimmten Zeit läuft jeweils nur ein Task
- Wenn ein Task die Kontrolle abgibt, kann ein anderer Task weiterlaufen
- Ein Scheduler steuert die aktivierung der Tasks und läuft in einer while Schlaufe

https://learn.adafruit.com/cooperative-multitasking-in-circuitpython-with-asyncio/overview

Python Cooperative Multitasking



Beispiele für die Aufgabe von Tasks:

```
await asyncio.sleep(interval)

await switch_event.wait(timeout)
```

Python Cooperative Multitasking

Multitasking Library

```
import asyncio
import board
import digitalio
```

Beschreibung der blink Funktion als Task

```
async def blink(pin, interval, count):
    with digitalio.DigitalInOut(pin) as led:
        led.switch_to_output(value=False)
        for _ in range(count):
            led.value = True
            await asyncio.sleep(interval)
            led.value = False
            await asyncio.sleep(interval)
```

Kennzeichnet Task Multitasking

Task wird für eine bestimmte Zeit blockiert

Beschreibung der Main Funktion als Task

Main Task

```
async def main():
    led1_task = asyncio.create_task(blink(board.led1, 0.25, 10))
    led2_task = asyncio.create_task(blink(board.led2, 0.1, 20))

    await asyncio.gather(led1_task, led2_task)

asyncio.run(main())
```

Task ID

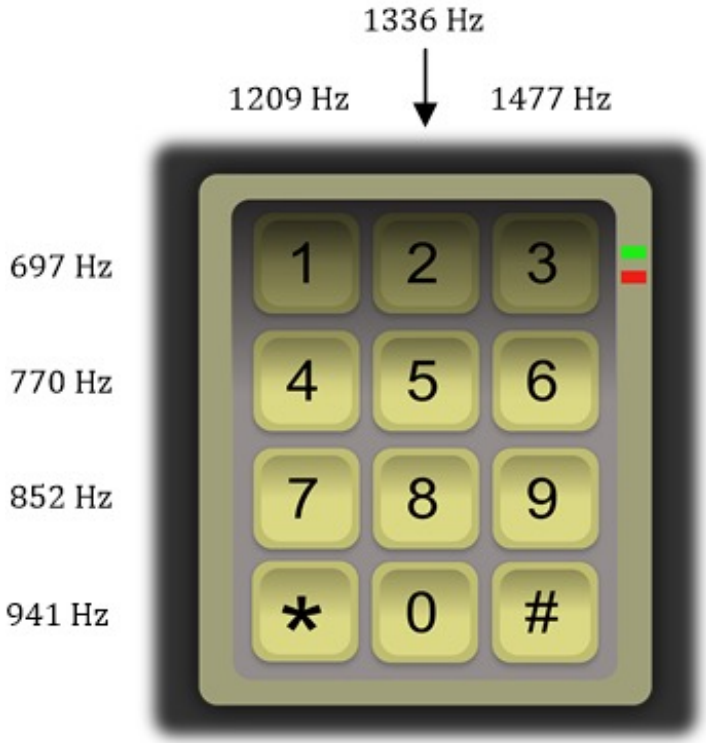
Veranlasst, dass Tasks gleichzeitig laufen

https://learn.adafruit.com/cooperative-multitasking-in-circuitpython-with-asyncio/concurrent-tasks

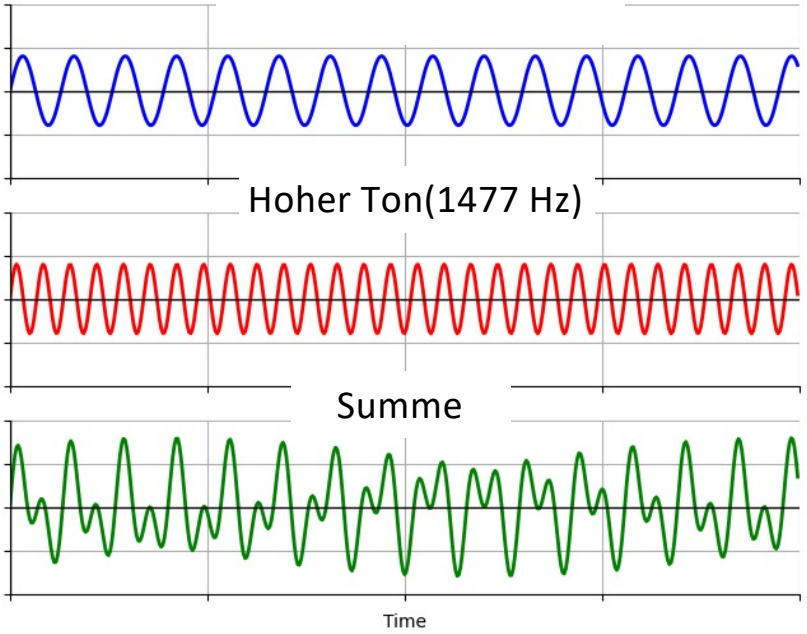
Software Implementierung:

10 DTMF Dekodierung mit Software

DTMF Prinzip



DTMF Signal für Taste “6”



https://wirelesspi.com/goertzel-algorithm-evaluating-dft-without-dft/

Der Goerzel Algorythmus

$$\frac{Y(z)}{X(z)} = \frac{1 - e^{-j2\pi\frac{k_0}{N}}z^{-1}}{\left(1 - e^{+j2\pi\frac{k_0}{N}}z^{-1}\right)\left(1 - e^{-j2\pi\frac{k_0}{N}}z^{-1}\right)}$$

Let us create an intermediate signal v_n with a z-Transform $V(z)$. Then, we have

$$\frac{Y(z)}{V(z)} \cdot \frac{V(z)}{X(z)} = \left(1 - e^{-j2\pi\frac{k_0}{N}}z^{-1}\right) \cdot \frac{1}{\left(1 - e^{+j2\pi\frac{k_0}{N}}z^{-1}\right)\left(1 - e^{-j2\pi\frac{k_0}{N}}z^{-1}\right)} \quad (1)$$

Consequently, the recursion y_n is generated from v_n through the inverse z-Transform of the first term above.

$$\frac{Y(z)}{V(z)} = \left(1 - e^{-j2\pi\frac{k_0}{N}}z^{-1}\right)$$

In time domain, this yields the Finite Impulse Response (FIR) filter term.

$$y_n = v_n - e^{-j2\pi\frac{k_0}{N}} \cdot v_{n-1} \quad (2)$$

Here, we produced y_n from the intermediate signal v_n. On the other hand, the signal v_n is generated from the input $x[n]$ through the second term in Eq (1) above.

$$\frac{V(z)}{X(z)} = \frac{1}{\left(1 - e^{+j2\pi\frac{k_0}{N}}z^{-1}\right)\left(1 - e^{-j2\pi\frac{k_0}{N}}z^{-1}\right)} = \frac{1}{1 - 2\cos\left(2\pi\frac{k_0}{N}\right)z^{-1} + z^{-2}}$$

where we have used the Euler's identity $2\cos\theta = e^{+j\theta} + e^{-j\theta}$ above. Next, collect $V(z)$ and $X(z)$ terms on each side.

$$V(z)\left[1 - 2\cos\left(2\pi\frac{k_0}{N}\right)z^{-1} + z^{-2}\right] = X(z)$$

In time domain, this produces the Infinite Impulse Response (IIR) filter term as

$$v_n = 2\cos\left(2\pi\frac{k_0}{N}\right)v_{n-1} - v_{n-2} + x[n] \quad (3)$$

The FIR Eq (2) and IIR Eq (3) can now be combined for the complete algorithm.

$$v_n = 2\cos\left(2\pi\frac{k_0}{N}\right)v_{n-1} - v_{n-2} + x[n]$$
$$y_n = v_n - e^{-j2\pi\frac{k_0}{N}} \cdot v_{n-1}$$

```
N = len(x)
mags = [0]*7
tones = np.concatenate((rowtones, coltones))
print(tones)

for fcount in range(7):
  k0 = N*tones[fcount]/fs # check the article on discrete frequency for this step

  omega_I = np.cos(2*np.pi*k0/N)
  omega_Q = np.sin(2*np.pi*k0/N)
  v1 = 0
  v2 = 0
  for n in range(N):
    v  = 2*omega_I*v1 - v2 + x[n] # see the IIR Eq (3)
    v2 = v1
    v1 = v

# Now value of v is in v1 and that of v1 is in v2
  y_I = v1 - omega_I*v2 # see the FIR Eq (2)
  y_Q = omega_Q*v2

  mags[fcount] = y_I**2 + y_Q**2 # figure below plotted after square root
```

Quelle: https://wirelesspi.com/goertzel-algorithm-evaluating-dft-without-dft/

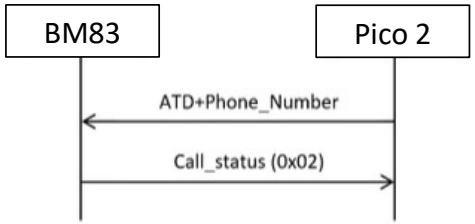
https://hackaday.io/project/190181-the-phone-friend/log/217664-reach-out-and-touch-someone

11

Anrufen ins Netz via Smartphone

Anrufen ins Netz

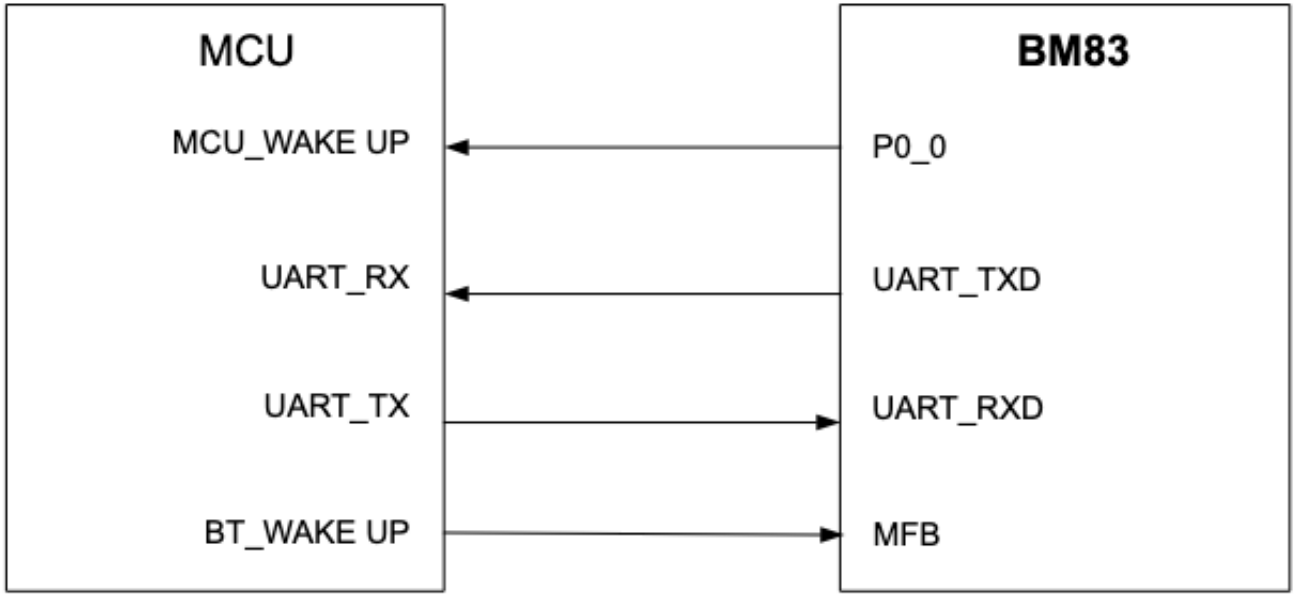
Microchip BM83



UART Pico 2 Protokoll

https://ww1.microchip.com/downloads/aemDocuments/documents/OTH/ProductDocuments/UserGuides/BM83_Host_MCU_Firmware_Development_Guide_DS50002896A.pdf

Bluetooth Modul UART Schnittstelle



Zusammenfassung

12

- Prototyp bis jetzt nur als Drahtaufbau
- Programmieren mit CircuitPython und Multitasking erweist sich als sehr zuverlässig
- Finden von Fehlern mit Print Statements ist akzeptabel
- Wer höhere Ansprüche an DTMF decodierung braucht verwendet CMX865A
- Python Prototype Code: https://github.com/hansgelke/retro_CircuitPython

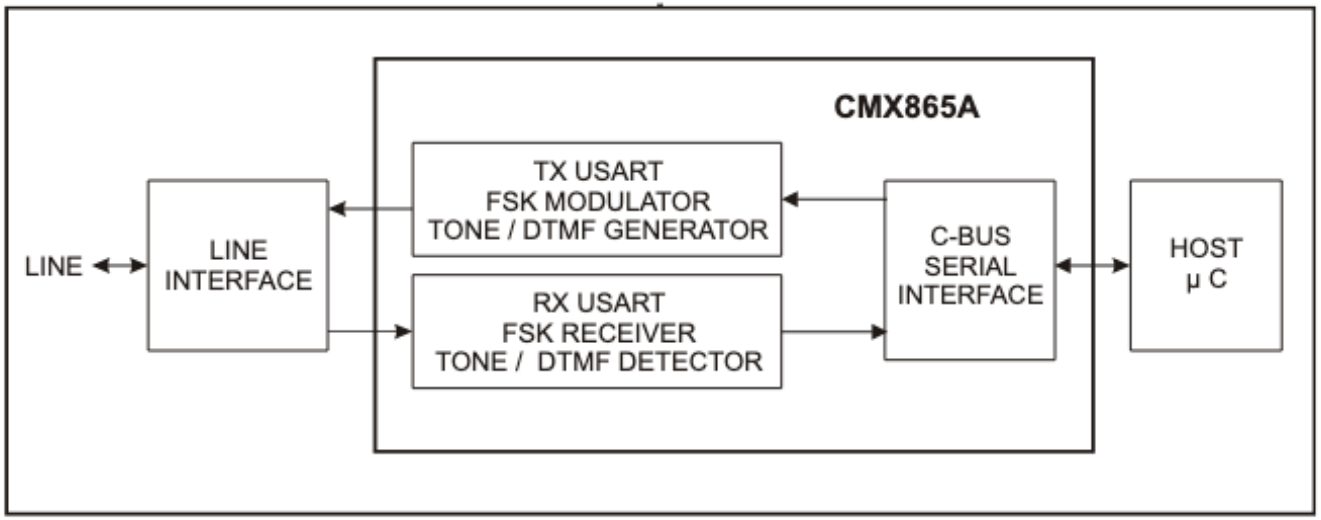
Fragen

13 ?

Python Prototype Code:
https://github.com/hansgelke/retro_CircuitPython

14 Appendix

Cooler Chip CMX865



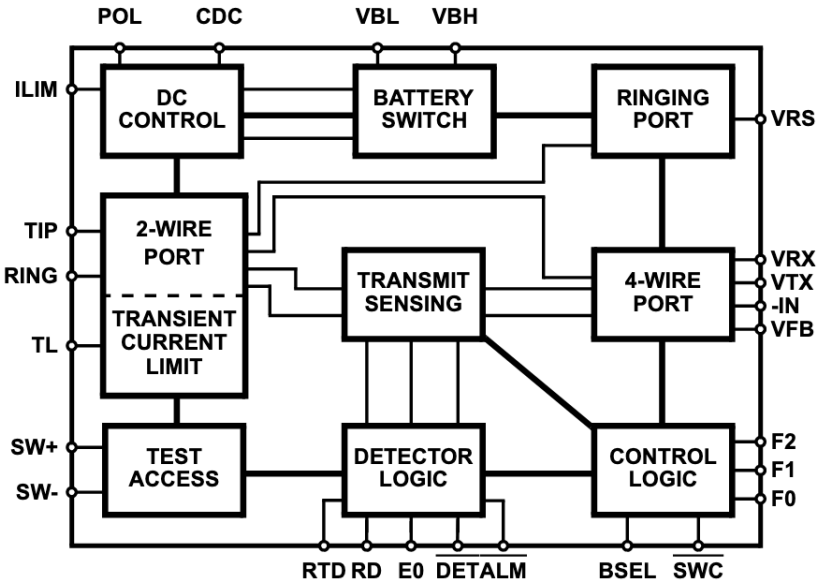
SPI Interface

- DTMF Decoder
- DTMF Generator
- Tone and Signalgenerator

CHF 3.75

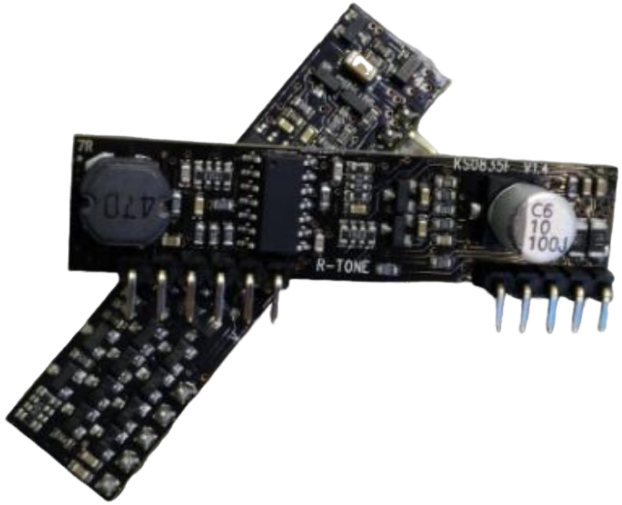
SLIC Alternativen

Renesas HC5518



- Unterstützt 100Vrms Weckerspannung
- Bessere Echo Cancelation
- Benötigt externe Batteriespannungen VBL und VBH

R-TONE KS0835F



- Unterstützt 150Vrms Weckerspannung
- Pin Kompatibel zu AG1171

Mischen von Sinus für DTMF

```
import board
import audioio
import audiocore
import audiomixer
import digitalio

audio_out = audiopwmio.PWMAudioOut(board.GP18)

sine_wave_1 = audiocore.RawSample(sine_array_1, sample_rate = sample_rate)
sine_wave_2 = audiocore.RawSample(sine_array_2, sample_rate = sample_rate)


mixer = audiomixer.Mixer(voice_count=2, sample_rate=sample_rate,
              channel_count=1, bits_per_sample=16, samples_signed=True)

audio_out.play(mixer)
mixer.voice[0].play(sine_wave_1 , loop=True)
while mixer.playing:
       mixer.voice[1].play(sine_wave_2 , loop=True)
       await asyncio.sleep(5)
```

https://docs.circuitpython.org/en/latest/shared-bindings/audiomixer/

Synchronisieren von Prozessen mit Event 1

```
import board
import digitalio
import asyncio

# Set up the LED
led = digitalio.DigitalInOut(board.LED)
led.direction = digitalio.Direction.OUTPUT

# Set up the button
button = digitalio.DigitalInOut(board.D2)
button.direction = digitalio.Direction.INPUT
button.pull = digitalio.Pull.UP

# Create an Event
button_event = asyncio.Event()

async def button_monitor():
    """Monitors the button and sets the event on press."""
    while True:
        if not button.value:  # Button is pressed (assuming active-low)
            print("Button pressed!")
            button_event.set()
        await asyncio.sleep(0.05)
```

Synchronisierung von Prozessen mit Event 2

```
async def led_control():
    """Waits for the event and toggles the LED."""
    while True:
        print("Waiting for button event...")
        await button_event.wait()
        led.value = not led.value
        print(f"LED is now {'on' if led.value else 'off'}")
        button_event.clear()
        await asyncio.sleep(0.3)  # debounce delay

async def main():
    await asyncio.gather(
        button_monitor(),
        led_control()
    )

asyncio.run(main())
```

Internationale Signalisierung

	Dial Tone/Hz	Engaged/Hz	Ringing/Hz	Unobtainable/Hz
Germany	425	425	425	950-1400-1800
England	350+450	400	400+450	400
US	350+440	440	440+480	Message

Legacy Projekt mit selbst enworfener SLIC