Wetterstation ‚Schülerlabor'
Softwarebeschreibung
10.01.2022 H.Schulte

Module:

```
.
├── batterie.py
├── BME280_1.py
├── boot.py
├── client.py
├── i2c_scan.py **)
├── main.py
├── reset.py
├── send_thingspeak2.py
├── test.py     **)
└── wdt.py
```

(Die mit **) gekennzeichneten Skripts sind nicht unbedingt erforderlich).
Skript-Fragmente sind mit

```
#Code
```

gekennzeichnet.

Modul boot.py

```
# boot.py
#...
print('\nboot: ' , f"'{reset.typ()}' reset")
print( 'rtc-memory:',machine.RTC().memory() )
```

‚boot' Meldet den Resettyp und den Inhalt vom RTC-Memory. Es sind folgende Resettypen möglich:

```
0: "poweron"
1: "watchdog"
4: "software"
5: "deepsleep“
6: "hardware"
x: "unbekannt"
```

RTC-Memory ist ein Speicher , der über Reset hinweg erhalten bleibt, jedoch bei Spannungsausfall gelöscht ist.

Beipiel für eine Boot-Meldung:

```
boot:  'poweron' reset
rtc-memory: b''    ( b'' heisst leer)
```

Modul main.py

```
#  main.py
print('...main')
import client
```

Nach boot.py geht es automatisch weiter bei ‚main.py'. Hier wird sofort weitergeleitet an ‚client.py'. Durch ‚#' bei *‚#import client' kann zum Testen der Autostart verhindert werden.*

Modul reset.py

```
    def gettyp():
        x=machine.reset_cause()
        y='unbekannter'
        if x==6:
            y="hardware"
        elif x==5:
            y="deepsleep"
        elif x==0:
            y="poweron"
        elif x==4:
            y="software"
        elif x==1:
            y= "watchdog"
        return y
```

Je nach Art des Neustarts wird der Resettyp als Text bestimmt.

Modul batterie.py

```
def volt():
        y=5.0*machine.ADC(0).read() /944

        return round (y,2) # 2 stellen hinter'm komma reichen
```

Der Ausdruck ‚machine.ADC(0).read()‘ liefert bei 1 Volt 1023 am Prozessoreingang AN0.
Der Messwert wird durch einen Spannungsteiler 470K / 100k geteilt.

U_adc= U_mess*100k/(100k+470k)= U_mess * 0,1754

Dreisatz liefert: 1 Volt => 1023
5 Volt * 0,1754 => x

x = (5 * 0,1754 / 1) * 1024 = 898
Der ADC sollte 898 leifern,
bei 5 Volt hat der ADC jedoch 944 gemessen (5 % Toleranz ist hier normal).

Modul client.py

Hier werden zur Übersicht nur die Funktionsaufrufe dargestellt, also ohne die Definionen der Funktionen.

```
schlaf= Schlafen(minuten=x)
```

Schlafzeit wird eingestellt (x = 0 heisst; kein Schlaf)

```
reset=Reset()
```

Resetereignis wird interpretiert

```
wdt= mywdt()
wdt.start()
```

Watchdog wird gestartet, das folgende Programm muss nach 10 Sekunden fertig sein oder es musss wdt.feed() aufgerufen werden; falls nicht so erfolgt über den watchdog Timer ein Reset.
Siehe ‚Modul wdt.py'

```
start_bme()
```

I2C Bus und BME-280 Modul starten; Module haben die I2C.Bus Adr. 0x76 oder 0x77. Die Adr. lässt sich mit ‚i2c_scan.py' prüfen.

```
for i in range(10): # 10sec, um kb interrupt zu ermoeglichen
    blink()
```

Eine kleine Pause mit blinkender LED; an dieser Stelle lässt sich das Programm durch Control-C unterbrechen.

```
WiFi_SSID, WiFi_PW= "ssid","password"
```

Wlan.Einstellungen

```
sta = network.WLAN(network.STA_IF)
```

Die WLAN Station wird eröffnet

```
if sta.active() and sta.isconnected():   # schon verbunden ?
```

Test, ob schon eine Verbindung besteht. ESP8266 seichert die letzten Einstellungen im nichtflüchtigen Speicher.
Wenn keine Verbindung besteht, neue Verbindung ausbauen

```
sta.active(True)
sta.connect(WiFi_SSID, WiFi_PW)
```

Warten, bis Verbindung besteht:

```
while not sta.isconnected ():
…
```

Messwerte vom BME280 Modul und Batteriespannung lesen:

```
t,h,d,v=messwerte()
```

An Thingspeak senden:

```
send(t,h,d,v)
```

siehe Modul ‚send_thingspeak2.py'

fertig; deepsleep starten:

```
schlaf.jetzt()     #ende
```

Modul BME280_1.py

```
i2c = I2C(scl=Pin(5), sda=Pin(4), freq=100000)
i2c_adr= i2c.scan()[0]
bme = BME280(address=i2c_adr, i2c=i2c)
```

Der Sensor BME280 wird über I2c-Bus-Leitungen ‚scl' und ‚sda' angesteuert. Da der Sensor der einzige Teilnehmer am Bus ist, kann als Adresse die eine Adresse genommen werden , die der i2c.scan findet.
Die Instanz von der Instanz ‚bme' wird benutzt:

```
t = bme.temperature
h = bme.humidity
d = bme.pressure
```

Modul send_thingspeak2.py

Neben einigen Überwachungen (z.B. kein Wlan, Falsche IP-Adresse etc.) besteht die Hauptaufgabe dieses Moduls, eine Verbindung zum ‚Thingspeak' Server herzustellen und die Messwerte zu senden. Dies geschieht mit:

```
urequests.get(url)
```

url="https://api.thingspeak.com/update?
api_key=OFSMOZN1COHG6PAV&
field1=20.1&field2=43.1&field3=999.0&field4=4.39" (eine Zeile)

‚url' ist der Internet Aufruf von Thingspeak, wobei unsere Daten darin eingebettet sind (api_key =… , fieldx=…)

Modul wdt.py „Watchdog"

Dieses Modul ist dafür da, unbestimmte Prozesszustände abzufangen („der Prozessor hängt"). Im wesentlichen besteht der Watchdog aus 2 Funktionen:

```
wdt.start():
    ''' hardware timer wird gestartet, watchdog ist jetzt scharf
'''
```

Nachdem ‚start' aufgerufen wurde, hat der Prozessor 10 Sekunden Zeit, etwas zu tun; falls es länger dauert, wird über einen eingebauten Timer ein Reset ausgelöst.

```
wdt.feed():
```

Falls der Prozessor etwas länger dauerndes zu tun hat, muss er periodisch ‚feed' aufrufen, dann hat er wieder 10 Sekunden Zeit bis zum automatischen Reset.