Diesen Beitrag gibt es auch als:
PDF in deutsch
This episode is also available as:
PDF in English

Heute geht es um das Senden von Textnachrichten über das Mobilfunknetz via SMS. ESP32 und SIM808 bilden eine Art Server oder Relaisstation, an den zur Steuerung Textnachrichten gesendet werden können, der aber auch in der Lage ist, selbst Nachrichten, zeit- oder eventgesteuert abzusetzen. Das Handy oder der PC, der Raspi… ist dann Schalt- und Empfangszentrale. Und weil die Strecke zwischen SIM808 und Handy keine Rolle spielt, gibt es auch kein Entfernungslimit, ausreichende Netzabdeckung vorausgesetzt. Das LCD-Keypad wird eigentlich überflüssig, weil eine direkte Steuerung der entfernten ESP32-Einheit sowieso nicht in Frage kommt. Dennoch leistet ein Display gute Dienste beim Start der Relaiseinheit und für weitere Statusmeldungen beim Debuggen. Interessanter als ein LCD ist da vielleicht ein kleines OLED-Display, rein zu Wartungszwecken, aber auch das kann über SMS laufen. Damit herzlich willkommen zum dritten Teil von

GPS und GSM mit MicroPython auf dem ESP32

Hardwarezuwachs – die SIM-Karte

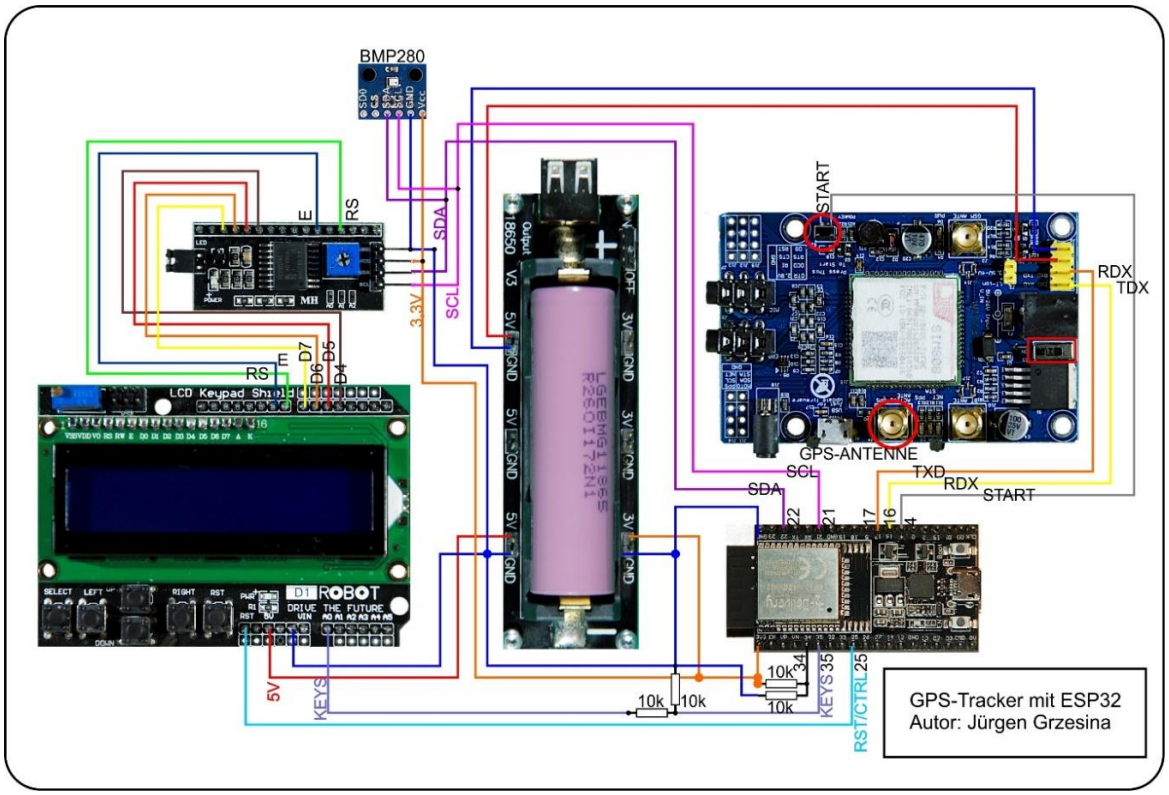
An Hardware kommt im Vergleich zu Tei2 wenig dazu. Wie? Sie haben die Teile 1 und 2 nicht gelesen und sind neu hier? Na gut, überredet, in der folgenden Liste finden Sie alle Teile für das aktuelle Projekt. Fast alles davon wurde bereits in Teil 1 und Teil2 eingesetzt und dort natürlich auch ausführlich beschrieben. Diese Bauteile verwenden wir selbstverständlich wieder. Neu hinzugekommen ist eine SIM-Karte, denn ohne diese werden Sie keine SMS-Nachricht versenden oder empfangen

können. Natürlich kann es sein, dass nicht alle Sensoren und/oder Aktoren wegen der Kabellänge direkt an der Relaisstation angeschlossen werden können. Dann wäre es doch nicht schlecht, wenn es dafür eine Funkstrecke gäbe. Deshalb zeige ich Ihnen in der nächsten Folge, wie man so etwas ganz unkompliziert mit dem UDP-Protokoll über WLAN-Verbindungen umsetzen kann. Für diesen Zweck werde ich als Beispiel einen ESP8266 mit einem LDR-Widerstand als Funksensor verwenden. Andere Sensoren wie DS18B20 (Temperatur), DHT22 AM2302(Feuchte und Temperatur), GY-302 BH1750 (Licht Sensor), etc. können dank der diversen MicroPython-Module, die es dafür gibt, ebenso gut hergenommen werden. Doch jetzt erst einmal zur GSM-Verbindung, wir wollen mit unserem ESP32 ein bisschen simsen.

Hier die Liste der Zutaten, das Rezept fürs Menü kommt später.

1	ESP32 Dev Kit C V4 unverlötet oder ähnlich
1	LCD1602 Display Keypad Shield HD44780 1602 Modul mit 2x16 Zeichen
1	SIM 808 GPRS/GSM Shield mit GPS Antenne für Arduino
1	Battery Expansion Shield 18650 V3 inkl. USB Kabel
1	Li-Akku Typ 18650
1	I2C IIC Adapter serielle Schnittstelle für LCD Display 1602 und 2004
4	Widerstand 10kΩ
1	GY-BMP280 Barometrischer Sensor für Luftdruckmessung
1	SIM-Karte (beliebiger Anbieter)

Die Schaltung für das Projekt wird erst einmal 1:1 von Teil 2 übernommen. Später entscheiden Sie selbst, welche Teile Sie weglassen, ersetzen oder neu hinzunehmen. Die programmtechnischen Möglichkeiten für die Umsetzung finden Sie in diesem Beitrag.



Ein besser lesbares Exemplar der Darstellung in DIN A4 bekommen Sie mit dem Download der PDF-Datei .

Die Software

Verwendete Software:
Fürs Flashen und die Programmierung des ESP:
Thonny oder
µPyCraft

Verwendete Firmware:
MicropythonFirmware

MicroPython-Module und Programme
GPS-Modul für SIM808 und GPS6MV2(U-Blocks)
LCD-Standard-Modul
HD44780U-I2C-Erweiterung zum LCD-Modul
Keypad-Modul
Button Modul
BMP208-Modul
i2cbus-Modul für standardisierten Zugriff auf den Bus
Das Hauptprogramm relais.py
testkeypad.py zum Testen der Tastendecodierung des LCD-Keypads

Tricks und Infos zu MicroPython

In diesem Projekt wird die Interpretersprache MicroPython benutzt. Der Hauptunterschied zur Arduino-IDE ist, dass Sie die MicroPython-Firmware auf den ESP32 flashen müssen, bevor der Controller MicroPython-Anweisungen versteht. Sie können dazu Thonny, µPyCraft oder esptool.py benutzen. Für Thonny habe ich den Vorgang im ersten Teil des Blogs zu diesem Thema beschrieben.

Nachdem die Firmware geflasht ist, können Sie sich zwanglos mit Ihrem Controller im Zwiegespräch unterhalten, einzelne Befehle testen und sofort die Antwort sehen, ohne vorher ein ganzes Programm compilieren und übertragen zu müssen. Bei der Entwicklung der Software für diesen Blog habe ich davon wieder reichlich Gebrauch gemacht. Das Spektrum reicht von einfachen Tests der Syntax bis zum Ausprobieren und Verfeinern von Funktionen und ganzen Programmteilen. Zu diesem Zweck werde ich, wie schon bei den vorangegangenen Folgen, kleine Testprogramme erstellen. Sie bilden eine Art Macro, weil sie wiederkehrende Befehle zusammenfassen.

Gestartet werden solche Programme aus dem aktuellen Editorfenster in der Thonny-IDE über die Taste F5, das geht schneller als der Mausklick auf den Startbutton oder über das Menü **Run**. Die Installation von Thonny habe ich auch im ersten Teil genau beschrieben.

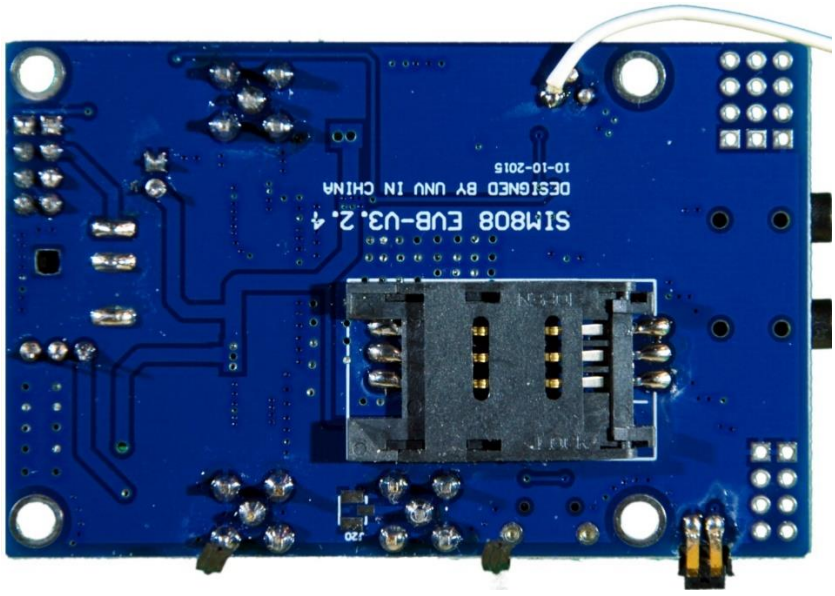
Die Klassen GPS, SIM808 und GSM

Gut, das Wichtigste für eine GPS-Anwendung ist: Wie spreche ich die GPS-Dienste des SIM808 an? Ach ja, richtig – es soll in diesem Beitrag ja nicht (allein) um GPS sondern vorrangig um GSM gehen. Die Frage muss also wohl anders lauten. Vielleicht: **Wie kann man mit dem SIM808 und dem ESP32 simsen**? Genau das wollen wir uns jetzt anschauen. Die AT-Befehle erlauben uns eine einfache Handhabung der vielfältigen Eigenschaften des SIM808-Moduls.Einen sehr kleinen Teil habe ich benutzt, um daraus die GSM-Klasse für mein Projekt zu basteln. Zusammen mit der Klasse GPS für die Ortung und der Klasse SIM808 für die Hardwareansteuerung finden Sie GSM im Modul **gps**.py friedlich vereint.

Für neu dazugestoßene Leser, so nehmen Sie das SIM808-Modul in Betrieb. Sie müssen den kleinen Schiebeschalter gleich neben der Rohrbuchse für die Spannungsversorgung, 5V bis 12V, in Richtung SIM808-Chip schieben. Eine rote LED leuchtet neben der GSM-Antennenbuchse auf. Das finden Sie sicher ganz leicht, denn alle Lötpunkte und Schnittstellen sind auf dem Board gut dokumentiert.

Ein Stück weiter links von der GSM-Antennenbuchse befindet sich die Starttaste. Sie können zur Orientierung auch die obige Schaltskizze zu Hilfe nehmen. Drücken Sie die Starttaste ca. 1 Sekunde lang, dann leuchten zwischen den anderen beiden Antennenbuchsen zwei weitere LEDs auf, die rechte davon blinkt im Sekundenrhythmus. An die linke Schraubbuchse sollte bereits die aktive GPS-Antenne angeschlossen sein. Diese legen Sie am besten in die Nähe eines Fensters.

Damit Sie jetzt nicht jedes Mal das Gehäuse ihres GPS-Empfängers öffnen müssen, um das SIM808 zu starten, empfehle ich Ihnen, es mir gleich zu tun und ein Kabel an den heißen Anschluss des Starttasters zu löten. Von oben betrachtet ist es der rechte, wenn die Rohrbuchse ebenfalls nach rechts zeigt. Sie können nun das SIM808 starten, indem Sie einen GPIO-Pin des ESP32 als Ausgang definieren und für eine Sekunde von High nach Low und zurück auf High schalten. Ich habe dafür den Pin 4 vorgesehen.



Beim Aufruf des Konstruktors für das GPS-Objekt wird die Nummer des Pins zusammen mit dem Displayobjekt als Parameter übergeben. Bevor Sie die nachfolgenden Kommandos an den ESP32 schicken, laden Sie bitte die eingangs verlinkten Module in den Flash-Speicher des Controllers hoch. Die Befehle werden über die Kommandozeile im Terminalbereich eingegeben.

```
>>> from gps import GPS,SIM808,GSM
>>> from lcd import LCD
>>> from machine import ADC, Pin, I2C
>>> from keypad import KEYPAD
>>> i2c=I2C(-1,Pin(21),Pin(22))
>>> d=LCD(i2c,0x27,cols=16,lines=2)
>>> g=GSM(4,d)
>>> k=KEYPAD(35)
```

Wird kein Displayobjekt (d) übergeben, gibt es natürlich auch keine Ausgabe auf LCD oder OLED. Es erfolgt aber keine Fehlermeldung, die Tastensteuerung arbeitet normal. Bei fast allen wichtigen Ergebnissen erfolgt eine Ausgabe im Terminalfenster.

Die Klasse GPS erledigt die Hauptarbeit. Der Konstruktor erwartet, wie erwähnt, ein Displayobjekt, das im aufrufenden Programm definiert oder bereits bekannt sein muss. Es wird ein serieller Kanal zum SIM808 auf 9600 Baud, 8,0,1 geöffnet, dann werden die Instanzvariablen für die Aufnahme der GPS-Daten eingerichtet.

Im Überblick hier die wichtigsten Methoden der GPS-Klasse.

Die Methode **waitForLine**() tut, was ihr Name sagt, sie wartet auf einen NMEA-Satz vom SIM808. Als Parameter wird der Typ des NMEA-Satzes angegeben, der erwartet wird. Ist der Satz vollständig und fehlerfrei, wird er an das aufrufende Programm zurückgegeben. Es können in der gegenwärtigen Ausbaustufe des Programms $GPRMC- und $GPGGA-Sätze empfangen werden. Sie enthalten alle relevanten Daten wie Gültigkeit, Datum, Zeit, geographische Breite (Latitude, vom Äquator aus bis zu den Polen in Grad) und Länge (Longitude vom Null-Meridian aus +/- 180°) sowie Höhe über NN in Metern. Analog zu dem bestehenden Code können leicht weitere Datensätze vom SIM808 aufgenommen und decodiert werden.

Die Methode **decodeLine**() nimmt den empfangenen Satz und versucht ihn zu parsen. Diese Methode enthält eine lokale Funktion, die nach Vorgabe des Attributs **Mode** die Winkelangaben in die Formate **Grad Minuten Sekunden und Bruchteile**, **Grad und Bruchteile** oder **Grad Minuten und Bruchteile** umwandelt.

Die Methode **printData**() gibt einen Datensatz im Terminalfenster aus. **showData**() liefert das Ergebnis an das Display. Weil mit dem LCD-Keypad nur ein zweizeiliges Display verwendet wird, muss die Anzeige in mehrere Abschnitte aufgeteilt werden. Die Tasten des Keypads übernehmen die Steuerung.

Weil die UART0-Schnittstelle für REPL reserviert ist, muss eine zweite Schnittstelle für die Kommunikation mit dem SIM808 vorhanden sein. Der ESP32 stellt eine solche als UART2 bereit. Die Anschlüsse für RXD (Empfang) und TXD (Sendung) können sogar frei gewählt werden. Für einen Vollduplexbetrieb (senden und

empfangen gleichzeitig) müssen die Anschlüsse RXD und TXD vom ESP32 zum SIM808 gekreuzt werden. Sie können das am Schaltplan nachvollziehen. Die Defaultwerte am ESP32 sind RXD=16 und TXD=17. Die Organisation des Anschlusses übernimmt die Klasse gps.GPS.

Das beginnt mit dem Einschalten des SIM808. Wenn Sie meiner Empfehlung gefolgt sind und ein Kabel an den Einschalttaster gelötet haben, können Sie das SIM808 jetzt mit folgendem Befehl einschalten, vorausgesetzt, dass dieses Kabel am Pin 4 des ESP32 liegt.

```
>>> g.SIMOn()
```

Befehle an das SIM808 werden im AT-Format übermittelt. Es gibt eine riesige Auswahl von Befehlen, die in einer PDF-Datei nachgelesen werden können. Aber keine Sorge, für unser Projekt reichen wenige Befehle. Zwei davon sind in den Methoden **init808**() und **deinit808**() zusammengefasst, ein paar weitere werden im GSM-Kapitel vorgestellt.

```
    def init808(self):
        self.u.write("AT+CGNSPWR=1\r\n")
        self.u.write("AT+CGNSTST=1\r\n")

    def deinit808(self):
        self.u.write("AT+CGNSPWR=0\r\n")
        self.u.write("AT+CGNSTST=0\r\n")
```

AT+CGNSPWR=1 aktiviert das GPS-Modul und AT+CGNSTST=1 aktiviert die Übertragung der NMEA-Sätze zum ESP32 über die serielle Schnittstelle UART2. Der Controller empfängt die Informationen des SIM808 und stellt sie in der oben beschriebenen Weise via Terminal und LCD bereit.

Das Modul **gps**.py enthält neben der Hardwaresteuerung des SIM808 auch noch die nötigen Befehle für das kleinere GPS-System GPS6MV2 mit dem Chip Neo 6M von UBLOX. Die Steuerung dieses Moduls erfolgt nicht über AT-Befehle, sondern über eine eigene Syntax. Die Klassen SIM808 und GPS6MV2 sind nicht gegen einander austauschbar, weil sie über unterschiedliche APIs verfügen.

Zum genaueren Studium des gps-Moduls folgt nun das Listing. Die drei enthaltenen Klassen GPS, SIM808(GPS) und GSM(SIM808) bauen durch Vererbung einen einheitlichen Namensraum auf. Daher sind alle Methoden in einem Objekt der Klasse GSM verfügbar. Wird GSM nicht gebraucht (kein SMS-Transfer), kann man auch über die Klasse SIM808 einsteigen, wie es im Teil 2 gemacht wurde.

Die Klasse SIM808 kümmert sich um die Hardwaresteuerung und um den Datentransfer zum ESP32. Die Klasse GPS enthält Methoden zur Decodierung der NMEA-Sätze vom SIM808, zu deren Darstellung auf dem Display und zur Kursberechnung. Die GSM-Klasse schließlich, stellt die Methoden für den SMS-Transfer und die Verwaltung von Nachrichten zur Verfügung. Diese Nachrichten müssen sich nicht allein auf GPS-Daten beziehen. Vielmehr habe ich die Klasse gps.GSM neutral gehalten, sodass sie auch in anderen Projekten einsetzbar ist. Auch im Programm relais.py finden Sie nicht nur einen GPS-Ansatz, sondern auch

die Umsetzung einer BMP280-Abfrage via GSM als Beispiel für die Einbindung weiterer Sensoren. Die dafür erforderlichen Klassen BMP280 und I2CBus wurden bereits in Teil2 vorgestellt.

```
"""
File: gps.py
Author: J. Grzesina
Rev. 1.0: AVR-Assembler
Rev. 2.0: Adaption auf Micropython
-----------------
Die enthaltenen Klassen sprechen einen ESP32 als Controller
an.
Dieses Modul beherbergt die Klassen GPS, GPS6MV2 und SIM808
GPS stellt Methoden zur Decodierung und Verarbeitung der NMEA-
Saetze
$GPGAA und $GPRMC bereit, welche die wesentlichen Infos zur
Position, Hoehe und Zeit einer Position liefern. Sie werden
dann
angezeigt, wenn die Datensaetze als "gueltig" gemeldet werden.
Eine Skalierung auf weitere NMEA-Sätze ist jederzeit möglich.
GPS6MV2 und SIM808 beziehen sich auf die entsprechende
Hardware.
"""
from machine import UART,I2C,Pin
import sys
from time import sleep, time, ticks_ms
from math import *

# ********************** Beginn GSM
****************************PS
class GPS:
    #
    gDeg=const(0)
    gFdeg=const(1)
    gMin=const(1)
    gFmin=const(2)
    gSec=const(2)
    gFsec=const(3)
    gHemi=const(4)

    #DEFAULT_TIMEOUT=500
    #CHAR_TIMEOUT=200

    def __init__(self,disp=None,key=None):  # display mit
OLED-API
        self.u=UART(2,9600)
        # u=UART(2,9600,tx=19,rx=18)  # mit alternativen Pins
        self.display=disp
        self.key=key
        self.timecorr=2
        self.Latitude=""
        self.Longitude=""
```

```
        self.Time=""
        self.Date=""
        self.Height=""
        self.Valid=""
        self.Mode="DMF" # default
        self.AngleModes=["DDF","DMS","DMF"]
        self.displayModes=["time","height","pos"]
        self.DMode="pos"
        # DDF = Degrees + DegreeFractions
        # DMS = Degrees + Minutes + Seconds + Fractions
        # DMF = Degrees + Minutes + MinuteFraktions
        self.DDLat=49.28868056  # aktuelle Position
        self.DDLon=11.47506105
        self.DDLatOld=49.3223 # vorige Position
        self.DDLonOld=11.5000
        self.zielPtr=0
        self.course=0
        self.distance=0
        print("GPS initialized,
Position:{},{}".format(self.DDLat,self.DDLon))

    def decodeLine(self,zeile):
        latitude=["","","","","N"]
        longitude=["","","","","E"]
        angleDecimal=0
        def formatAngle(angle):  # Eingabe ist Deg:Min:Fmin
            nonlocal angleDecimal
            minute=int(angle[1])     # min als int
            minFrac=float("0."+angle[2]) # minfrac als float

angleDecimal=int(angle[0])+(float(minute)+minFrac)/60
            if self.Mode == "DMS":
                seconds=minFrac*60
                secInt=int(seconds)
                secFrac=str(seconds - secInt)

a=str(int(angle[0]))+"*"+angle[1]+"'"+str(secInt)+secFrac[1:6]
+'"'+angle[4]
            elif self.Mode == "DDF":
                minutes=minute+minFrac
                degFrac=str(minutes/60)
                a=str(int(angle[0]))+degFrac[1:]+"* "+angle[4]
            else:

a=str(int(angle[0]))+"*"+angle[1]+"."+angle[2]+"' "+angle[4]
            return a

        # GPGGA-Fields
        nmea=[0]*16
        name=const(0)
        time=const(1)
        lati=const(2)
```

```
        hemi=const(3)
        long=const(4)
        part=const(5)
        qual=const(6)
        sats=const(7)
        hdop=const(8)
        alti=const(9)
        auni=const(10)
        geos=const(11)
        geou=const(12)
        aged=const(13)
        trash=const(14)
        nmea=zeile.split(",")
        lineStart=nmea[0]
        if lineStart == "$GPGGA":

self.Time=str((int(nmea[time][:2])+self.timecorr)%24)+":"+nmea
[time][2:4]+":"+nmea[time][4:6]
            latitude[gDeg]=nmea[lati][:2]
            latitude[gMin]=nmea[lati][2:4]
            latitude[gFmin]=nmea[lati][5:]
            latitude[gHemi]=nmea[hemi]
            longitude[gDeg]=nmea[long][:3]
            longitude[gMin]=nmea[long][3:5]
            longitude[gFmin]=nmea[long][6:]
            longitude[gHemi]=nmea[part]
            self.Height,despose=nmea[alti].split(".")
            self.Latitude=formatAngle(latitude)  # mode =
Zielmodus Winkelangabe
            self.DDLat=angleDecimal
            self.Longitude=formatAngle(longitude)
            self.DDLon=angleDecimal
        if lineStart == "$GPRMC":
            date=nmea[9]
            self.Date=date[:2]+"."+date[2:4]+"."+date[4:]
            try:
                self.Valid=nmea[2]
            except:
                self.Valid="V"

    def waitForLine(self,title,delay=2000):
        line=""
        c=""
        d=delay
        if delay < 1000: d=1000
        start = ticks_ms()
        end=start+d
        current=start
        while current <= end:
            #print(end-current)
            if self.u.any():
                c=self.u.read(1)
```

```
                if ord(c) <=126:
                    c=c.decode()
                    if c == "\n":
                        test=line[0:6]
                        if test==title:
                            #print(line)
                            return line
                        else:
                            line=""
                    else:
                        if c != "\r":
                            line +=c
            current = ticks_ms()
            sleep(0.05)
        return ""

    def showData(self):
        if self.display:
            if self.DMode=="time":

self.display.writeAt("Date:{}".format(self.Date),0,0)

self.display.writeAt("Time:{}".format(self.Time),0,1)
            if self.DMode=="height":
                self.display.writeAt("Height: {}m
".format(self.Height),0,0)

self.display.writeAt("Time:{}".format(self.Time),0,1)
            if self.DMode=="pos":
                self.display.writeAt(self.Latitude+" "*(16-
len(self.Latitude)),0,0)
                self.display.writeAt(self.Longitude+" "*(16-
len(self.Longitude)),0,1)

    def printData(self):
        print(self.Date,self.Time,sep="_")
        print("LAT",self.Latitude)
        print("LON",self.Longitude)
        print("ALT",self.Height)

    def showError(self,msg):
            if self.display:
                self.display.clearAll()
                self.display.writeAt(msg,0,0)
            print(msg)

    def storePosition(self):  # aktuelle Position als DD.dddd
merken
        lat=str(self.DDLat)+","
        lon=str(self.DDLon)+"\n"
        try:
            D=open("stored.pos","wt")
```

```
            D.write(lat)
            D.write(lon)
            D.close()
            if self.display:
                self.display.clearAll()
                self.display.writeAt("Pos. stored",0,0)
                sleep(3)
                self.display.clearAll()
        except (OSError) as e:
            enumber=e.args[0]
            if enumber==2:
                print("Not stored")
                if self.display:
                    self.display.clearAll()
                    self.display.writeAt("act. Position",0,0)
                    self.display.writeAt("not stored",0,0)
                    sleep(3)
                    self.display.clearAll()

    def chooseDestination(self, wait=3):
        if not self.display: return None
        self.display.clearAll()
        self.display.writeAt("ENTER=RST-Button",0,0)
        n="positions.pos"
        try:
            D=open(n,"rt")
            ziel=D.readlines()
            D.close()
            i = 0
            while 1:
                lat,lon=(ziel[i].rstrip("\r\n")).split(",")
                self.display.clearAll()
                self.display.writeAt("{}.
{}".format(i,lat),0,0)
                self.display.writeAt("   {}".format(lon),0,1)
                sleep(wait)
                if self.key.value()==0: break
                i+=1
                if i>=len(ziel): i=0
            self.zielPtr=i
            self.display.clearAll()
            self.display.writeAt("picked: {}".format(i),0,0)
            sleep(wait)
            self.display.clearAll()
            lat,lon=ziel[i].split(",")
            lon=lon.strip("\r\n")
            print("{}. Lat,Lon: {}, {}".format(i,lat,lon))
            lat=float(lat)
            lon=float(lon)
            return (lat,lon)
        except (OSError) as e:
            enumber=e.args[0]
```

```
            if enumber==2: print("File not found")
            self.display.clearAll()
            self.display.writeAt("There is no",0,0)
            self.display.writeAt("Positionfile",0,1)
            sleep(3)
            self.display.clearAll()
            return (0,0)

    def calcNewCourse(self,delay=3): # von letzter Position
bis hier
        lat,lon=self.chooseDestination(delay)
        if lat==0 and lon==0: return
        dy=(lat-self.DDLat)*60*1852
        dx=(lon-
self.DDLon)*60*1852*cos(radians(self.DDLatOld))
        print("Start {},{}".format(self.DDLat,self.DDLon),"
Ziel {},{}".format(lat,lon))
        #print("Ziel-Start",self.DDLon-self.DDLonOld,
self.DDLat-self.DDLatOld)
        return self.calcCourse(dx,dy)

    def calcLastCourse(self): # von letzter Position bis hier
        try:
            D=open("stored.pos","rt")
            lat,lon=(D.readline()).split(",")
            D.close()
            self.DDLatOld=float(lat)
            self.DDLonOld=float(lon)
        except:
            self.DDLatOld=49.3223
            self.DDLonOld=11.50
        dy=(self.DDLat-self.DDLatOld)*60*1852
        dx=(self.DDLon-
self.DDLonOld)*60*1852*cos(radians(self.DDLatOld))
        print("Start
{},{}".format(self.DDLonOld,self.DDLatOld),"  Ziel
{},{}".format(self.DDLon,self.DDLat))
        #print("Ziel-Start",self.DDLon-self.DDLonOld,
self.DDLat-self.DDLatOld)
        return self.calcCourse(dx,dy)

    def calcCourse(self,dx,dy): # von letzter Position bis
hier
        course=0
        distance=0
        #print(dx,dy,degrees(atan2(dy,dx)))
        if abs(dx) < 0.0002:
            if dy > 0:
                course=0
                #print("Trace: 1")
            if dy < 0:
                course=180
```

```
                #print("Trace: 2")
            if abs(dy) < 0.0002:
                course=None
                #print("Trace: 3")
        else:  # dx >= 0.0002
            if abs(dy) < 0.0002:
                if dx > 0:
                    course=90
                    #print("Trace: 4")
                if dx < 0:
                    course=270
                    #print("Trace: 5")
            else:  ## dy > 0.0002
                course=90-degrees(atan2(dy,dx))
                #print("Trace: 6")
                if course > 360:
                    course -= 360
                    #print("Trace: 7")
                if course < 0:
                    course += 360
                    print("Trace: 8")
        self.course=int(course)
        self.distance=int(sqrt(dx*dx+dy*dy))
        print("Distance: {}, Course:
{}".format(self.distance,self.course))
        return (self.distance,self.course)

# ********************** Ende GPS
*****************************

# ********************* Beginn SIM808
**************************
class SIM808(GPS):
    DEFAULT_TIMEOUT=const(500)
    CHAR_TIMEOUT=const(100)
    CMD=const(1)
    DATA=const(0)

    def __init__(self,switch=4,disp=None,key=None):
        self.switch=Pin(switch,Pin.OUT)
        self.switch.on()
        super().__init__(disp,key)
        self.display=disp
        self.key=key
        print("SIM808 initialized")

    def simOn(self):
        self.switch.off()
        sleep(1)
        self.switch.on()
        sleep(3)
```

```
    def simOff(self):
        self.switch.off()
        sleep(3)
        self.switch.on()
        sleep(3)

    def simStartPhone():
        pass

    def simGPSInit(self):
        self.u.write("AT+CGNSPWR=1\r\n")
        self.u.write("AT+CGNSTST=1\r\n")

    def simGPSDeinit(self):
        self.u.write("AT+CGNSPWR=0\r\n")
        self.u.write("AT+CGNSTST=0\r\n")

    def simStopGPSTransmitting(self):
        self.u.write("AT+CGNSTST=0\r\n")

    def simStartGPSTransmitting(self):
        self.u.write("AT+CGNSTST=1\r\n")

    def simFlushUART(self):
        while self.u.any():
            self.u.read()

    # Wartet auf Zeichen an UART -> 0: keine Zeichen bis
Timeout
    def simWaitForData(self,delay=CHAR_TIMEOUT):
        noOfBytes=0
        start=ticks_ms()
        end=start+delay
        current=start
        while current <= end:
            sleep(0.1)
            noOfBytes=self.u.any()
            if noOfBytes>0:
                break
        return noOfBytes

    def
simReadBuffer(self,cnt,tout=DEFAULT_TIMEOUT,ctout=CHAR_TIMEOUT
):
        i=0
        strbuffer=""
        start=ticks_ms()
        prevchar=0
        while 1:
            while self.u.any():
                c=self.u.read(1)
                c=chr(ord(c))
```

```
                prevchar=ticks_ms()
                strbuffer+=c
                i+=1
                if i>=cnt: break
            if i>= cnt:break
            if ticks_ms()-start > tout: break
            if ticks_ms()-prevchar > ctout: break
        return (i,strbuffer) # gelesene Zeichen

    def simSendByte(self,data):
        return self.u.write(data.to_bytes(1,"little"))

    def simSendChar(self,data):
        return self.u.write(data)

    def simSendCommand(self,cmd):
        self.u.write(cmd)

    def simSendCommandCRLF(self,cmd):
        self.u.write(cmd+"\r\n")

    def simSendAT(self):
        return self.simSendCmdChecked("AT","OK",CMD)

    def simSendEndMark(self):
        self.simSendChar(chr(26))

    def
simWaitForResponse(self,resp,typ=DATA,tout=DEFAULT_TIMEOUT,cto
ut=CHAR_TIMEOUT):
        l=len(resp)
        s=0
        self.simWaitForData(300)
        start=ticks_ms()
        prevchar=0
        while 1:
            if self.u.any():
                c=self.u.read(1)
                if ord(c) < 126:
                    c=c.decode()
                    prevchar=ticks_ms()
                    s=(s+1 if c==resp[s] else 0)
                if s == l: break
            if ticks_ms()-start > tout: return False
            if ticks_ms()-prevchar > ctout: return False
        if type==CMD:
            self.simFlushUART()
        return True

    def
simSendCmdChecked(self,cmd,response,typ,tout=DEFAULT_TIMEOUT,c
tout=CHAR_TIMEOUT):
```

```
        self.simSendCommand(cmd)
        return
self.simWaitForResponse(response,typ,tout,ctout)

# ********************** Ende  SIM808
*************************

# ********************** Beginn GSM
*************************

class GSM(SIM808):

    def __init__(self, switch=4, disp=None, key=None):
        super().__init__(switch,disp,key)
        self.gsmInit()
        print("GSM module initialized")

    def gsmInit(self):
        if not self.simSendCmdChecked("AT","OK\r\n",CMD):
            return False
        if not
self.simSendCmdChecked("AT+CFUN=1","OK\r\n",CMD):
            return False
        if not self.gsmCheckSimStatus():
            return False
        return True

    def gsmIsPowerUp(self):
        return self.simSendAT()

    def gsmPowerOn(self):
        self.switch.off()
        sleep(3)
        self.switch.on()
        sleep(3)

    def gsmPowerReset(self):
        self.switch.off()
        sleep(3)
        self.switch.on()
        sleep(3)
        self.switch.off()
        sleep(1)
        self.switch.on()
        sleep(3)

    def gsmCheckSimStatus(self):
        n=0
        a=""
        self.simFlushUART()
        while n < 3:
            self.simSendCommand("AT+CPIN?\r\n")
```

```
            a=self.simReadBuffer(32)
            if "+CPIN: READY" in a[1]:
                break
            n+=1
            sleep(0.3)
        if n == 3:
            return False
        return True


    def gsmSendSMS(self,phoneNbr,mesg):
        if not
self.simSendCmdChecked("AT+CMGF=1\r\n","OK\r\n",CMD):
            print("SMS-Mode not selcted")
            return False
        sleep(0.5)
        self.simFlushUART()
        if not
self.simSendCmdChecked('AT+CMGS="'+phoneNbr+'"\r\n',">",CMD):
            print("Phonenumber Problem")
            return False
        sleep(1)
        self.simSendCommand(mesg)
        sleep(0.5)
        self.simSendEndMark()
        sleep(1)
        return self.simWaitForResponse(mesg,CMD)
        #return self.simReadBuffer(50)

    def gsmAreThereSMS(self,stat):
        buf=""
        # SMS-Modus aktivieren
        self.simSendCmdChecked("AT+CMGF=1\r\n","OK\r\n",CMD)
        sleep(1)
        self.simFlushUART()
        # ungelesene SMS listen ohne Statusänderung ",1"
        #print("SMS-Status",stat)
        self.simSendCommand('AT+CMGL="{}",1\r\n'.format(stat))
        sleep(2)
        # OK findet sich in den ersten 30 Zeichen nur, wenn
        # keine ungelesene SMS vorliegt
        a=self.simReadBuffer(30)[1]
        #print(30,a)
        if "OK" in a:
            sleep(0.1)
            return 0
        else:
            # restliche Zeichen im UART-Buffer entsorgen
            self.simFlushUART()
            # erneuter Aufruf zum Einlesen

self.simSendCommand('AT+CMGL="{}",1\r\n'.format(stat))
```

```
            sleep(2)
            a=self.simReadBuffer(48)[1]
            #print(48,a)
            # suche nach der Position von "+CMGL:"
            p=a.find("+CMGL:")
            if p != -1:
                pkomma=a.find(",",p)
                #print("gefunden",a[p+6:pkomma])
                return int(a[p+6:pkomma])
            else:
                #print("CMGL not found", a)
                return None
        #print("Kein 'OK'")
        return None

    def gsmReadAll(self,stat,cnt=500):
        # SMS-Modus aktivieren
        self.simSendCmdChecked("AT+CMGF=1\r\n","OK\r\n",CMD)
        sleep(1)
        self.simFlushUART()
        # SMS listen ohne Statusänderung ",1"
        self.simSendCommand('AT+CMGL="{}",1\r\n'.format(stat))
        sleep(2)
        a=self.simReadBuffer(cnt)
        #print("BUFFER:\n",a[1],"\n")
        return a

    def gsmFindSMS(self,stat="ALL",cnt=150):
        Liste=[]
        i=0
        p0=0
        again=-1
        self.simFlushUART()
        m,a=self.gsmReadAll(stat,cnt)
        while again == -1:
            again=a.find("OK")  # wenn OK gefunden, letzter
Durchgang
            #print("again",again) # only for debugging
            #print("@@@@",a,"@@@@") # only for debugging
            while 1:
                merker=p0
                #print("merker: ",merker) # only for debugging
                p0=a.find("+CMGL:",p0)
                #print("While1_p0",p0)
                if p0 == -1:
                    p0=merker
                    #print("@p0_break",p0) # only for
debugging
                    break
                p1=a.find(",",p0)
                #print("p0 und p1: ",p0,p1) # only for
debugging
```

```
                if p1 == -1:
                    p0=merker
                    #print("@p1_break",p0) # only for
debugging
                    break
                n=int(a[p0+6:p1])
                #print("Liste: ",Liste,"neu: ",n) # only for
debugging
                Liste.append(n)
                p0=p1
            m,x=self.simReadBuffer(cnt)
            #print(p0,": Buffer: \n",a[p0:],"*****",x,"<<<<<")
# only for debugging
            a=a[p0:]+x
            p0=0
            if m == 0:
                break
        return Liste

    def gsmReadSMS(self,index,mode=0): # mode=1: Status nicht
veraendern
        # SMS-Modus einschalten
        self.simSendCmdChecked("AT+CMGF=1\r\n","OK\r\n",CMD)
        sleep(1)
        self.simFlushUART()
        try:
            if mode==1:

self.simSendCommand("AT+CMGR={},1\r\n".format(index))
            elif mode==0:

self.simSendCommand("AT+CMGR={}\r\n".format(index))
            sleep(1)
            a=self.simReadBuffer(250) # Antwort=(Anzahl,
Zeichen)
            #print(a[1]) # only for debugging
            if a[0] != 0:
                a=a[1].split(',',4+mode)
                #print(a)  # only for debugging
                p0=a[0+mode].find('"')
                p1=a[0+mode].find('"',p0+1)
                Status=a[0+mode][p0+1:p1]
                Phone=a[1+mode].strip('"')
                Date=a[3+mode].lstrip('"')
                Time=a[4+mode].split('"')[0]

Message=a[4+mode].split('"')[1].strip("\r\n").rstrip("\r\nOK")
                return (Status,Phone,Date,Time,Message)
            else:
                return None
        except (IndexError,TypeError) as e:
            print("Index out if range",e.args[0])
```

```
            return None

    def gsmShowAndDelete(self,stat,disp=None,delete=None):
        SMSlist=self.gsmFindSMS(stat, cnt=100)
        while 1:
            if len(SMSlist)==0:
                print("nothing to do")
                break
            for i in SMSlist:
                response=self.gsmReadSMS(i,mode=1)
                print(i,response[0],"\n",response[4])
                if disp:
                    disp.clearAll()
                    disp.writeAt("{}.
{}".format(i,response[0]),0,0)
                    disp.writeAt(response[4],0,1)
                    sleep(2)
                if delete or
not(response[1]=='+49xxxxxxxxxx'): #evtl. weitere PhoneNbr +
stati
                    if self.gsmDeleteSMS(i):
                        print("!!! deleted",response[1])
            first=SMSlist[0]
            SMSlist=self.gsmFindSMS(stat,cnt=100)
            if len(SMSlist)==0 or SMSlist[0] == first: break
        if disp: disp.clearAll()

    def gsmDeleteSMS(self,index):
        print("SMS[{}] deleted".format(index))
        return
self.simSendCmdChecked("AT+CMGD={},0\r\n".format(index),"OK\r\
n",CMD)

# ********************** Ende GSM
*****************************

# ******************** Beginn GPS6MV2
*************************
class GPS6MV2(GPS):
    # Befehlscodes setzen
    GPGLLcmd=const(0x01)
    GPGSVcmd=const(0x03)
    GPGSAcmd=const(0x02)
    GPVTGcmd=const(0x05)
    GPRMCcmd=const(0x04)

    def __init__(self,delay=1,disp=None,key=None):
        super().__init__(disp,key)
        self.display=disp
        self.delay=delay  # GPS sendet im delay Sekunden
Abstand
        period=delay*1000
```

```
SetPeriod=bytearray([0x06,0x08,0x06,0x00,period&0xFF,(period>>
8)&0xFF,0x01,0x00,0x01,0x00])
        self.sendCommand(SetPeriod)
        self.sendScanOff(bytes([GPGLLcmd]))
        self.sendScanOff(bytes([GPGSVcmd]))
        self.sendScanOff(bytes([GPGSAcmd]))
        self.sendScanOff(bytes([GPVTGcmd]))
        self.sendScanOn(bytes([GPRMCcmd]))
        print("GPS6MV2 initialized")

    def sendCommand(self,comnd):  # comnd ist ein bytearray
        self.u.write(b'\xB5\x62')
        a=0; b=0
        for i in range(len(comnd)):
            c=comnd[i]
            a+=c  # Fletcher Algorithmus
            b+=a
            self.u.write(bytes([c]))
        self.u.write(bytes([a&0xff]))
        self.u.write(bytes([b&0xff]))

    def sendScanOff(self,item):  # item ist ein bytes-objekt
        shutoff=b'\x06\x01\x03\x00\xF0'+item+b'\x00'
        self.sendCommand(shutoff)

    def sendScanOn(self,item):
        turnon=b'\x06\x01\x03\x00\xF0'+item+b'\x01'
        self.sendCommand(turnon)
```

GSM – oder Simsen für den ESP32

Zu den AT-Befehlen für das Einschalten der GPS-Einheit und das Öffnen der UART-Verbindung auf dem SIM808 gesellen sich für den SMS-Betrieb einige weitere, auf die ich kurz eingehen werde. Grundsätzlich sind AT-Befehle mit einem \r\n (Wagenrücklauf und Zeilenvorschub) abzuschließen, damit sie vom SIM808 als solche erkannt werden. Das SIM808 sendet Antworten oder Ergebnisse zurück. Außer bei Nutzdaten, wie Nachrichtentexten, ist es hier nur wichtig, ob die Antwort der erwarteten entspricht. Es gibt daher eine gsm-Methode, die genau diese Überprüfung durchführt. Das Klassenattribut CMD (Wert = 1) kennzeichnet den AT-Befehl als Kommando. Für Kommandos gilt, dass nach dem Einlesen der Antwort vom SIM808 der Rest des UART-Buffers automatisch geleert wird.

simSendCmdChecked("AT+CMGF=1\r\n","OK\r\n",CMD)
AT+CMGF=1: schaltet auf Textbetrieb um, und ermöglich so den SMS-Betrieb

Der Status von SMS-Nachrichten erlaubt deren Auswahl zum Beispiel beim Listbefehl. **stat** enthält demnach einen der folgenden Strings: "ALL", "REC UNREAD" oder "REC_READ". Dieser Wert wird durch die Formatanweisung in den Befehl eingebaut.

```
simSendCommand('AT+CMGL="{}",1\r\n'.format(stat))
```
Die '**1**' sorgt dafür, dass sich beim Auflisten der Status nicht ändert.

Beim Versenden von SMS-Nachrichten muss dem Befehl als erstes die Nummer des Anschlusses mitgeteilt werden. Das SIM808 antwortet darauf mit einem ">".

```
simSendCmdChecked('AT+CMGS="'+phoneNbr+'"\r\n',">",CMD)
```

Wurde ">" erkannt, kann die Nachricht gesendet werden.

```
simSendCommand(mesg)
```

Das Ende der Nachricht wird der Gegenstation durch Senden eines Textendezeichens (chr(26)) bekannt gegeben.

Um eine SMS-Nachricht aus dem Speicher des SIM808 zu lesen, muss deren Index bekannt sein. Nach Absetzen des Lesebefehls kann der UART-Puffer des SIM808 ausgelesen werden. Er enthält den Status, Datum, Uhrzeit und den Text der Nachricht.

```
"AT+CMGR={}\r\n".format(index))
```

Nachrichten können durch Angabe des Index auch gelöscht werden.

simSendCmdChecked("AT+CMGD={},0\r\n".format(index),"OK\r\n",CMD)

Erst nachdem eine SIM-Karte erkannt wurde, ist der SMS-Betrieb möglich. Der Befehl

```
simSendCommand("AT+CPIN?\r\n")
```

überprüft das. Die SIM-Karte darf dabei nicht durch eine PIN gesichert sein.

Ein Teil der GSM-Methoden dient der internen Verarbeitung und Steuerung. Das Hauptprogramm relais.py zeigt die Anwendung der Methoden der Klasse.

Das Anwendungsprogramm **relais**.py ist im Vergleich zur reinen GPS-Anwendung aus Teil2 ein wenig umfangreicher geworden. Das liegt an den verschiedenen Beispielen für die Abfrage von Sensoren und dem SMS-Nachrichtenverkehr. Das Programm hat schon einiges zu bieten. Im Einzelnen demonstriert es:

- zeitgesteuerte SMS (alle x Stunden, Minuten…)
- eventgesteuerte SMS (Temperatur außerhalb eines Bereichs)
- SMS on demand (Antwort auf eine SMS)
- GPS-Tracker (Entfernung überschritten oder Wegpunktübermittlung)
- Messwert übermitteln

```
# File: relais.py
# put contents into boot.py after successful testing
# Purpose: Booting SMS-Relais Station
# Author:  J. Grzesina
# Rev.:1.0 - 2021-04-29
#********************** Beginn Bootsequenz
************************
# Dieser Teil geht 1:1 an boot.py fuer autonomen Start
#************************ Importgeschaeft
*************************
# Hier werden grundlegende Importe erledigt
import os,sys       # System- und Dateianweisungen

import esp          # nervige Systemmeldungen aus
esp.osdebug(None)

import gc           # Platz fuer Variablen schaffen
gc.collect()
#
from machine import ADC, Pin, I2C
from button import BUTTONS, BUTTON32
from time import sleep,time,sleep_ms, ticks_ms
from gps import GPS,SIM808, GSM
from lcd import LCD
from keypad import KEYPAD
from bmp280 import BMP280

#****************** Variablen/Objekte deklarieren
******************
# ------------------------------------------------------------
----
# ****************** create essential objects
*********************
# ------------------------------------------------------------
----
rstNbr=25
rst=BUTTON32(rstNbr,True,"RST")
ctrl=Pin(rstNbr,Pin.IN,Pin.PULL_UP)
t=BUTTONS() # Methoden für Buttons bereitstellen

i2c=I2C(-1,Pin(21),Pin(22))

k=KEYPAD(35)

d=LCD(i2c,0x27,cols=16,lines=2)
d.printAt("SIM808-GSM",0,0)
d.printAt("RELAY booting",0,1)

sleep(1)

g=GSM(switch=4,disp=d,key=ctrl)
print("************************")
```

```
g.simGPSInit()
g.simOn()
g.mode="DDF"

b=BMP280(i2c)
#sleep(10)

timeFlag    =0b00000001 # Intervallsteuerung
distanceFlag=0b00000010 # Streckenalarm
tempFlag    =0b00000100 # Tempraturalarm
orderFlag   =0b00001000 # SMS Anforderung per SMS
Trigger=orderFlag # Hier die Flags der Dienste eintragen

distLimit=100 # Entfernungsrichtwert in Metern
tempMax=25 # *C
tempMin=20 # *C

heartbeat=0 # Actionblinker

timeBase=3600*1 # Zeitintervall ins Sekunden
distanceBase=3600*1#3600*2 # Sekunden
tempBase=20#3600*2 # Sekunden
timeEnde=time()+5
distanceEnde=time()+5
tempEnde=time()+5

# Die Dictionarystruktur (dict) erlaubt spaeter die
Klartextausgabe
# des Verbindungsstatus anstelle der Zahlencodes
#********************Funktionen deklarieren
**********************

def timeJob():
    t=str(b.calcTemperature())+"*C"
    p=str(b.calcPressureNN())+"hPa"
    print("sende Wetterdaten")
    g.simFlushUART()

g.gsmSendSMS("+49xxxxxxxxxx","Wetterwerte:\n{},\n{}".format(t
,p))
    return  True

def tempJob(temp):
    t=str(temp)+"*C"
    return g.gsmSendSMS("+49xxxxxxxxxx
","WARNUNG!!:\n{},\nTemperatur nicht im Limit!!".format(t))

def positionJob(dist):
    g.gsmSendSMS("+49xxxxxxxxxx ","Entfernung: {},\n
Positionswerte\n{},\n{}".format(dist,g.Latitude,g.Longitude))
    print("Position gemeldet")
```

```
def orderJob(Nachrichten):
    # decode message
    if "wetter" in Nachrichten.lower():
        print("Wetterdaten angefordert")
        return timeJob()
    # do jobs
    # send results
    # return True # if no errors occured

def getDistance():
    rmc=g.waitForLine("$GPRMC",delay=1000)
    if rmc:
        try:
            g.decodeLine(rmc)
            if g.Valid == "A":
                try:
                    gga=g.waitForLine("$GPGGA",delay=2)
                    g.decodeLine(gga)
                    g.printData()
                    if d: g.showData()
                    entfernung=g.calcLastCourse()[0]
                    print("Distanz: {}".format(entfernung))
                    return entfernung
                except:
                    g.showError("Invalid GGA-set!")
        except:
            g.showError("Invalid RMC-set!")
# ********************   Funktionen Ende
***********************

print("Check SMS")
g.gsmShowAndDelete("REC_READ",d,delete=True)
g.gsmShowAndDelete("ALL",d,delete=False)

# ***********************   Main Loop
*************************
while 1:
    getDistance()
    if d:
        heartbeat+=1
        if heartbeat % 2:
            d.writeAt("X ",14,0)
        else:
            d.writeAt(" "+g.Valid,14,0)

    if (Trigger & timeFlag) == timeFlag and time() >=
timeEnde:
        timeJob()
        timeEnde=time()+timeBase

    if (Trigger & distanceFlag) == distanceFlag and time() >=
distanceEnde:
```

```
        g.calcLastCourse()
        w=g.distance
        print(w)
        if w > distLimit:
            positionJob(w)
            ## naechsten Befehl aus kommentieren für
Kreisfläche
            g.storePosition()
        distanceEnde = time()+ distanceBase

    if (Trigger & orderFlag) == orderFlag:
        L=g.gsmFindSMS("REC UNREAD")
        if L:
            Nachricht=g.gsmReadSMS(L[0],mode=1)[4]
            print("neue SMS gefunden", L)
            orderJob(Nachricht)
            print("Job erledigt, mache jezt Kaffepause")
            sleep(5)
            g.gsmDeleteSMS(L[0])
            sleep(2)

    if (Trigger & tempFlag)==tempFlag and time() >= tempEnde:
        t=b.calcTemperature()
        #print("Temperatur: {} ## {}".format(t,tempEnde))
        if t < tempMin or t > tempMax:
            tempJob(t)
        tempEnde=time()+tempBase
        #print(tempEnde)

    # Job-Schleifenende
    if ctrl.value() == 0: break
    sleep(0.1)
```

Hinweis:
Damit Sie auch Nachrichten auf Ihr Handy bekommen, setzen Sie bitte an anstatt der +49xxxxxxxxxx Ihre eigene Handynummer ein.

Für die einzelnen Jobs gibt es jeweils ein Steuerflag. damit wird die Funktion scharf geschaltet. Alle Funktionen bis auf SMS on demand haben neben dem Steuerflag noch eine zusätzliche Zeitsperre. Diese bewirkt, dass zum Beispiel nach dem Überschreiten der Temperatur nicht pausenlos SMS versandt werden, bis der Wert wieder im Rahmen ist. Die Zeitsperre wird in Sekunden angegeben, kann aber durch entsprechende Faktoren fast beliebig gedehnt werden.

Im Eventmodus wird in festen Zeitabschnitten ein Sensor abgefragt. Nur wenn der Sensorwert nicht den Vorgaben entspricht, wird eine Nachricht versandt.

Eine Art Eventmodus ist auch die SMS-Nachricht über GPS-Positionen. Möglich sind natürlich viel mehr als die gezeigten zwei Optionen. Mit Aktivierung des

distanceFlags ist der Wegpunkt-Modus voreingestellt. Wenn Sie das Speichern der letzten Position auskommentieren, reagiert das System auf jede Entfernung über die vorgegebene Strecke hinaus, in festen zeitlichen Abständen, wie oben beschrieben.

Der rein zeitgesteuerte Modus verschickt unabhängig von einem Sensorwert das Ergebnis einer Messung. Hier wird per Voreinstellung auf die Messergebnisse des BMP280 zugegriffen, stellvertretend für beliebige weitere Sensoren.

Bei der SMS on demand können Codeworte an den ESP32 via SMS gesendet werden. Ungelesene SMS-Leichen werden bei jedem Neustart gelöscht. Dann informiert das System über noch im Speicher befindliche andere Nachrichten.

in der Jobschleife wird auf eingetroffene Nachrichten geprüft. Die Jobfunktion muss den Inhalt decodieren und gegebenenfalls entsprechende Aktionen einleiten oder Messungen durchführen. Denkbar sind auch Aktionen, die direkt auf das SIM808 wirken und zum Beispiel alle SMS löschen. In jedem Fall automatisch gelöscht wird auch die Mail, welche die Aktion ausgelöst hat. Diese Betriebsart ist voreingestellt.

Weitere Nachrichten warten im Eingangspuffer des SIM808 und werden der Reihe nach abgearbeitet. Sehr wichtig sind die Wartezeiten (fett). werden die weggelassen oder zu kurz angesetzt, dann gehen SMS im Dauerlauf raus, weil der nächste Schleifendurchlauf die Nachricht erneut auffindet und bearbeitet.

```
if (Trigger & orderFlag) == orderFlag:
    L=g.gsmFindSMS("REC UNREAD")
    if L:
        Nachricht=g.gsmReadSMS(L[0],mode=1)[4]
        print("neue SMS gefunden", L)
        orderJob(Nachricht)
        print("Job erledigt, mache jetzt Kaffepause")
        sleep(5)
        g.gsmDeleteSMS(L[0])
        sleep(2)
```

Sie sehen, die Einsatzmöglichkeiten sind sehr vielfältig. Dazu kommt, dass niemand außer Ihnen diese Steuerung unberechtigt benutzen kann, es sei denn, Sie hängen die Telefonnummer Ihrer SIM-Karte ans Schwarze Brett, zusammen mit all Ihren Steuercodes und dem Handy. Denn selbst Zugriffe von anderen Telefonnummern außer Ihrer eigenen sind im Programm geblockt.

So, und damit das alles autonom mit dem Einschalten des ESP32 startet, müssen Sie das Programm einfach in die Datei boot.py kopieren und diese zum ESP32 hochladen, Neustart, das war's. Sollte nachträglich etwas schief gehen und der ESP32 von Thonny oder einem anderen Terminalprogramm aus nicht mehr ansprechbar sein, dann ziehen Sie einfach die Notbremse, die das Programm abbricht. Es ist die RST-Taste am LCD-Keypad, die Sie so lange drücken, bis im Terminal der REPL-Prompt >>> erscheint. Jetzt können Sie Änderungen durchführen.

Ich wünsche viel Vergnügen beim Simsen mit Ihrem ESP32. In der nächsten Folge erlauben wir unserem Controller, zusammen mit dem SIM808, fremdzugehen. Ein oder auch mehrere Funkmodule, die mit ESP8266 oder ESP32 bestückt sein können, werden den Aktionsradius des Controllers erweitern. Das Übertragungsprotokoll wird UDP sein, das reicht für unsere Zwecke locker aus und ist viel schneller und simpler als TCP. Bleiben Sie dran!

Viel Spaß bei der Umsetzung des Projekts!

Weitere Downloadlinks:
PDF in deutsch
PDF in english