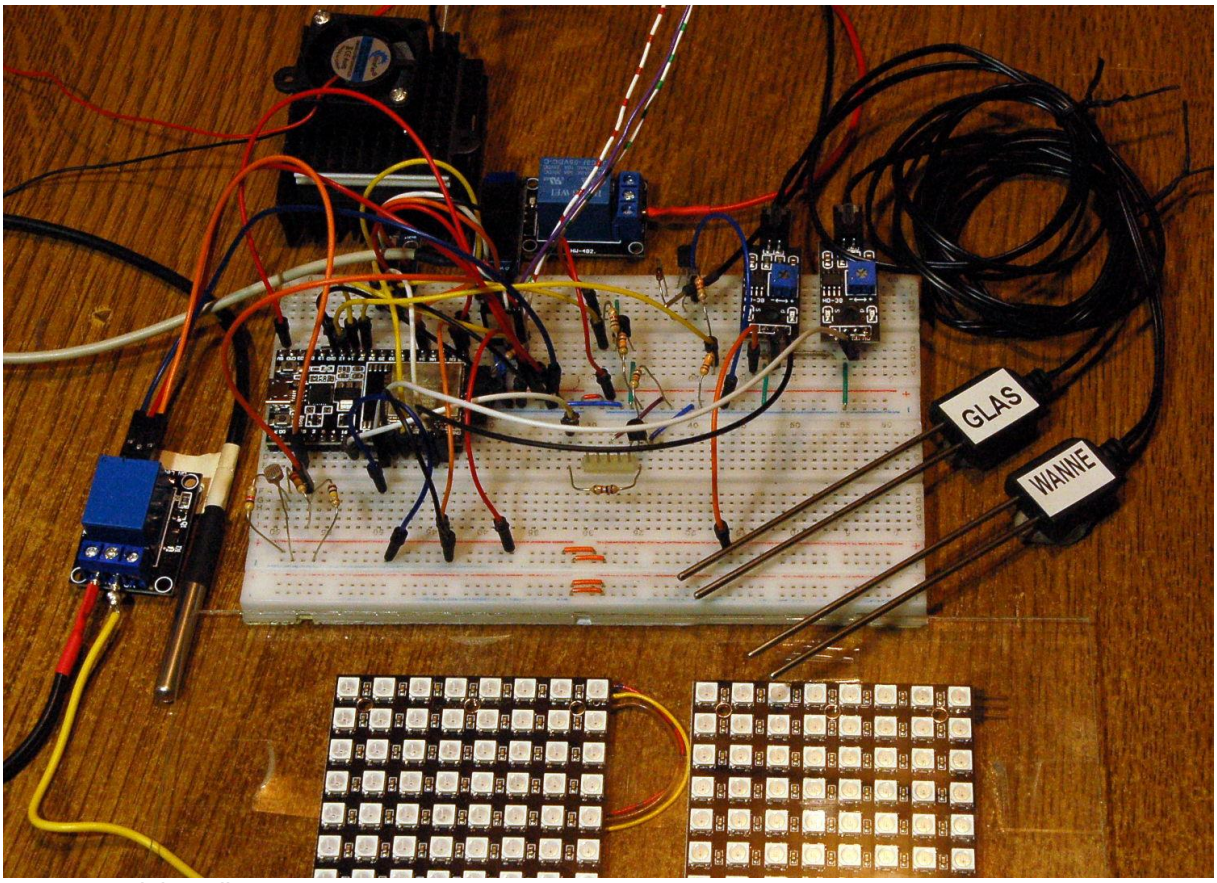

Gesamte Elektronik

Diese Folge ist auch als PDF-Dokument erhältlich.

Zum fleißigen Wachstum brauchen Pflanzen neben Licht und Wasser auch eine bestimmte Temperatur. Gerade in der Zeit, in der man gerne aus Samen Pflänzchen großziehen möchte, kann es aber auf der Fensterbank noch recht kühl werden, vor allem nachts. Im Moment sind wegen der Gas- und Öl-Knappheit Wärmepumpen sehr gefragt. Die Geräte arbeiten in der Regel mit einem Kompressor. Ich habe mich für eine Wärmepumpe ohne bewegliche Teile entschieden. Mit Peltierelementen kann man kühlen und wenn man die Stromrichtung umkehrt, auch heizen. Der positive Aspekt dabei ist, dass man mehr Wärmeenergie in die Saatbox bekommt, als man elektrische Energie dafür einsetzen muss. Allerdings muss man die heiße Seite permanent überwachen, damit zum einen der Kunststoff der Box nicht schmilzt und vor allem die maximale Betriebstemperatur des Peltierelements nicht überschritten wird. Die Schmelz- oder Erweichungstemperatur liegt deutlich niedriger und ist daher für uns somit entscheidend. Kommen Sie mit, auf eine Entdeckungstour durch eine neue Folge von

MicroPython auf dem ESP32 und ESP8266

heute

Teil 4 – Ein Peltier-Element gegen kalte Tage

Etwas Theorie zum Aufwärmen

Ein thermoelektrischer Wandler besteht aus zwei verschiedenen Metallen, deren Enden verschweißt oder verlötet sind. Einer der Leiter wird aufgetrennt. Je nach der weiteren Beschaltung sind zwei Betriebsmodi möglich.

Der thermoelektrische oder **Seebeck**-Effekt besteht darin, dass an den Kontaktstellen zweier unterschiedlicher Metalle eine Wanderung von Elektronen stattfindet. Wenn man die Kontaktstellen erhitzt oder abkühlt, führt das wegen der unterschiedlichen Geschwindigkeit der Elektronen an den Kontaktstellen zu einer elektrischen Spannung an der Stelle, wo einer der beiden Leiter aufgetrennt ist. Weil das Element selbst als Spannungsquelle wirkt, ist die technische Stromrichtung innerhalb des Elements vom Minus- zum Pluspol, denn durch das Voltmeter fließt der Strom von plus nach minus.

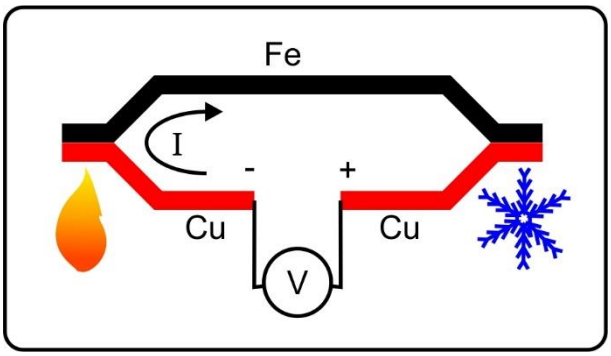

Abbildung 1: Seebeck-Effekt

Vertauscht man Ursache und Wirkung, dann ergibt sich daraus der **Peltier**-Effekt. Das heißt, dass ein Stromfluss durch das Element eine Temperaturdifferenz an den Kontaktstellen zur Folge hat. Bei gleicher Stromrichtung kühlt diejenige Kontaktstelle ab, die beim Seebeck-Effekt erhitzt werden müsste.

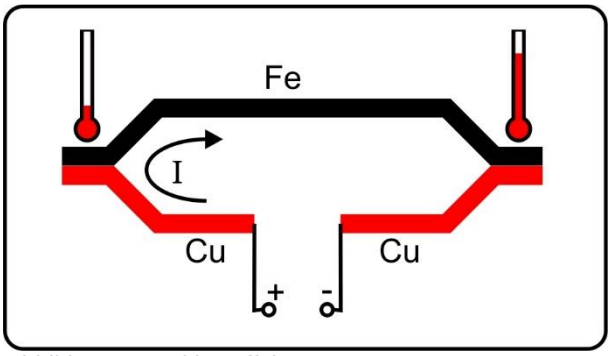

Abbildung 2: Peltier-Effekt

Das funktioniert im Prinzip auch mit Eisen und Kupfer. Jedoch wird der Peltier-Effekt, ähnlich wie beim Seebeck-Effekt, erheblich gesteigert, wenn man anstelle der Metalle Halbleitermaterial verwendet. Bei unseren Peltierelementen ist das positiv und negativ dotiertes Wismut-Tellurid. 127 kleine Quaderchen aus diesem Material sind in Serie geschaltet und zwischen zwei Keramikplatten aneinandergereiht. Die PN-Übergänge liegen damit parallel an der einen Platte (oben), die NP-Kontakte an

der anderen. Dadurch addieren sich die Auswirkungen des Peltier-Effekts an den Platten, die eine Seite kühlt ab, während die andere erhitzt wird.

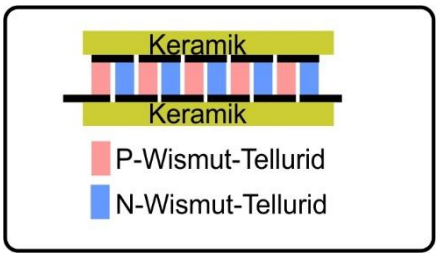

Abbildung 3: Peltier-Element_schematisch

So sieht das hier verwendete Peltierelement aus. Es hat die Maße 40mm x 40 mm x 3,6mm.

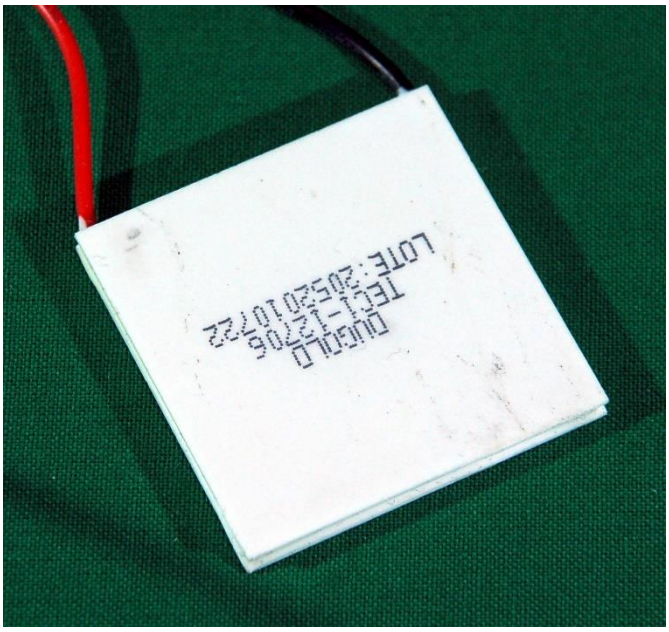

Abbildung 4: Peltierelement oder thermoelektrischer Wandler

Hardware

Als Controller habe ich einen ESP32 gewählt, weil der mit genügend frei wählbaren GPIO-Anschlüssen aufwarten kann, davon brauchen wir beim Vollausbau 10 Stück aufwärts.

Die ESP32-Modelle in der Teileliste sind alle brauchbar. Lediglich beim ESP32-Lolin-Board muss für den I2C-Anschluss SDA statt GPIO21 der GPIO25-Pin genommen werden.

1	ESP32 Dev Kit C unverlötet oder ESP32 Dev Kit C V4 unverlötet oder ESP32 NodeMCU Module WLAN WiFi Development Board mit CP2102 oder NodeMCU-ESP-32S-Kit oder ESP32 Lolin LOLIN32 WiFi Bluetooth Dev Kit
1	0,91 Zoll OLED I2C Display 128 x 32 Pixel
1	AHT10 Feuchte und Temperatursensor oder AHT10 Feuchte und Temperatursensor
1	TEC1-12706 Thermoelektischer Wandler - Peltierelement
2	Widerstand 10k
1	Widerstand 1k
1	NPN-Transistor BC548 o. ä.
1	1M Kabel DS18B20 digitaler Edelstahl Temperatursensor
1	DS18B20 digitaler Temperatursensor TO92
1	1-Relais 5V KY-019 Modul High-Level-Trigger
1	MB-102 Breadboard Steckbrett mit 830 Kontakten
diverse	Jumper Wire Kabel 3 x 40 STK. je 20 cm M2M/ F2M / F2F evtl. auch 65Stk. Jumper Wire Kabel Steckbrücken für Breadboard
1	Netzteil 5V / 3A
optional	Logic Analyzer

Die Software

Fürs Flashen und die Programmierung des ESP32:

Thonny oder
µPyCraft

Verwendete Firmware für den ESP32:

v1.19.1 (2022-06-18) .bin

Die MicroPython-Programme zum Projekt:

ssd1306.py Hardwaretreiber für das OLED-Display
oled.py API für das OLED-Display
temperatur.py Das Programm zur Lichtsteuerung
aht10.py API für den Klimasensor AHT10

MicroPython - Sprache - Module und Programme

Zur Installation von Thonny finden Sie hier eine ausführliche Anleitung (english version). Darin gibt es auch eine Beschreibung, wie die Micropython-Firmware (Stand 18.06.2022) auf den ESP-Chip gebrannt wird.

MicroPython ist eine Interpretersprache. Der Hauptunterschied zur Arduino-IDE, wo Sie stets und ausschließlich ganze Programme flashen, ist der, dass Sie die MicroPython-Firmware nur einmal zu Beginn auf den ESP32 flashen müssen, damit der Controller MicroPython-Anweisungen versteht. Sie können dazu Thonny, µPyCraft oder esptool.py benutzen. Für Thonny habe ich den Vorgang hier beschrieben.

Sobald 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 kompilieren und übertragen zu müssen. Genau das stört mich nämlich an der Arduino-IDE. Man spart einfach enorm Zeit, wenn man einfache Tests der Syntax und der Hardware bis hin zum Ausprobieren und Verfeinern von Funktionen und ganzen Programmteilen über die Kommandozeile vorab prüfen kann, bevor man ein Programm daraus strickt. Zu diesem Zweck erstelle ich auch gerne immer wieder kleine Testprogramme. Als eine Art Makro fassen sie wiederkehrende Befehle zusammen. Aus solchen Programmfragmenten entwickeln sich dann mitunter ganze Anwendungen.

Autostart

Soll das Programm autonom mit dem Einschalten des Controllers starten, kopieren Sie den Programmtext in eine neu angelegte Blankodatei. Speichern Sie diese Datei unter boot.py im Workspace ab und laden Sie sie zum ESP-Chip hoch. Beim nächsten Reset oder Einschalten startet das Programm automatisch.

Programme testen

Manuell werden Programme aus dem aktuellen Editorfenster in der Thonny-IDE über die Taste F5 gestartet. Das geht schneller als der Mausklick auf den Startbutton, oder über das Menü **Run**. Lediglich die im Programm verwendeten Module müssen sich im Flash des ESP32 befinden.

Zwischendurch doch mal wieder Arduino-IDE?

Sollten Sie den Controller später wieder zusammen mit der Arduino-IDE verwenden wollen, flashen Sie das Programm einfach in gewohnter Weise. Allerdings hat der ESP32/ESP8266 dann vergessen, dass er jemals MicroPython gesprochen hat. Umgekehrt kann jeder Espressif-Chip, der ein kompiliertes Programm aus der Arduino-IDE oder die AT-Firmware oder LUA oder … enthält, problemlos mit der MicroPython-Firmware versehen werden. Der Vorgang ist immer so, wie hier beschrieben.

Die Schaltung

Die gesamte Schaltung für das Projekt im Überblick zeigt Abbildung 5. Teile aus den vorangegangenen Blogfolgen Es werde Licht und Wasser marsch sind ausgegraut. Das AHT10-Modul aus der ersten Folge AHT10 wird hier wieder verwendet.

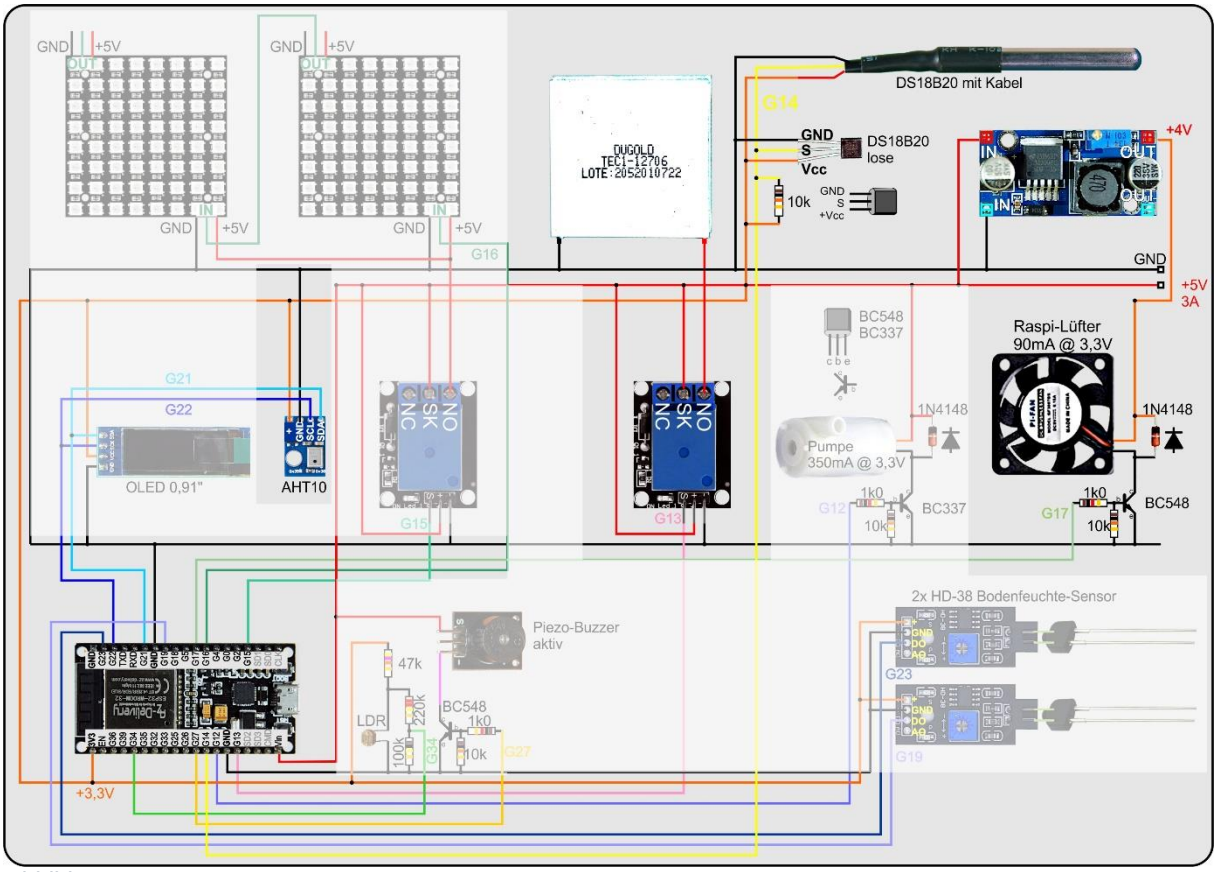

Abbildung 5: Temperatursteuerung

Den Abschnitt rechts oben kommt habe ich hier noch einmal herausvergrößert, damit das Kleingedruckte besser lesbar ist.

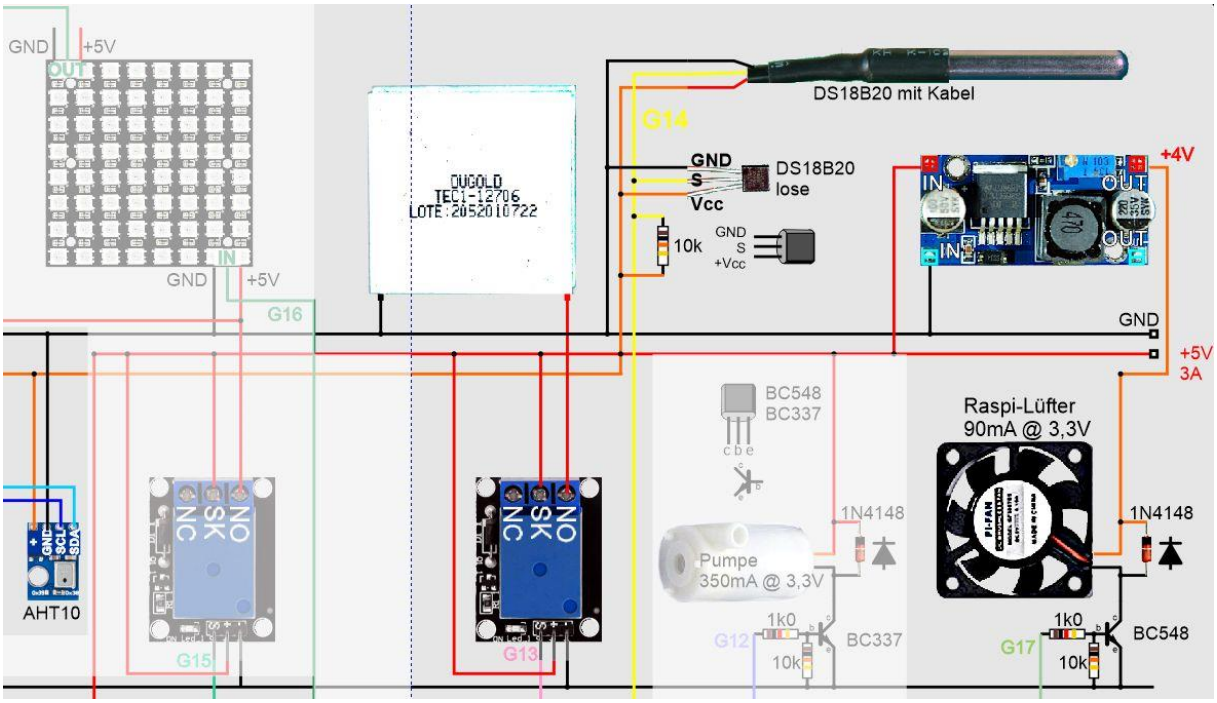

Abbildung 6: Temperatur-Einheit

Die Baugruppe rund um die Temperaturmessung setzt sich aus dem Peltierelement, einem Relais, einem Ventilator, zwei Kühlkörpern, einem Buck-Konverter, zwei DS18B20-Temepratur-Sensoren und dem AHT10-Modul aus dem ersten Beitrag zur Staffel Saatbox zusammen.

Der eine DS18B20 ist mit einem Kabelstück von einem Meter verbunden und kann, falls man das möchte, als Erdthermometer in einem der Pflanztöpfchen dienen oder auch als Temperatursensor in der Wasserwanne, denn er ist in einem Edelstahlrohr wasserdicht verpackt.

Der andere, lose Sensor wird an drei Kabelstücke gelötet, mit Schrumpfschlauchabschnitten gesichert und zwischen die Kühlrippen des Kühlkörpers auf der Warmseite des Peltierelements geklemmt. Er muss zuverlässig dem ESP32 helfen, die Temperatur auf ca. 50°C - 60°C zu begrenzen, damit der Kunststoff der Box nicht wegschmilzt.

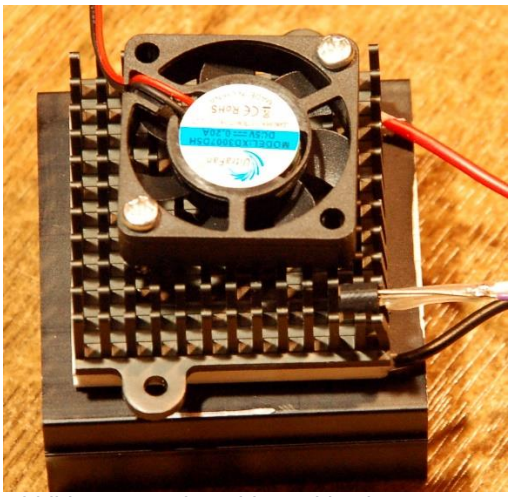

Abbildung 7: Heiz-Kühl-Kombination

Für die Verteilung der Warmluft dient der Raspi-Lüfter, der über einen Transistor ein- und ausgeschaltet wird. Der 10kΩ-Widerstand hält die Basis des Transistors sicher auf GND-Potenzial, solange der Steuer-Pin GPIO17 des ESP32 beim Booten noch als Eingang geschaltet ist. Sobald GPIO17 als Ausgang durch schreiben einer 1 auf 3,3V geht, schaltet der Transistor durch, eine programmierte 0 lässt GPIO17 auf 0V fallen und der BC548 sperrt. Für Lüfter mit höherer Stromaufnahme (> 100mA) muss der BC548 durch einen Typ mit höherem Kollektor-Dauerstrom ersetzt werden. Etwa ein BC337 (bis 800mA) oder ein BC517 (bis 1,0A) wären dann geeignet.

Für die Montage des Peltierelements wird eine Aussparung von 40mm x 40mm in die Wand der Box geschnitten oder gefräst. Nach dem Einsetzen des Elements werden von innen und außen die Kühlkörper befestigt. Achten Sie bitte darauf, dass sie guten flächigen Kontakt zum Peltierelement haben.

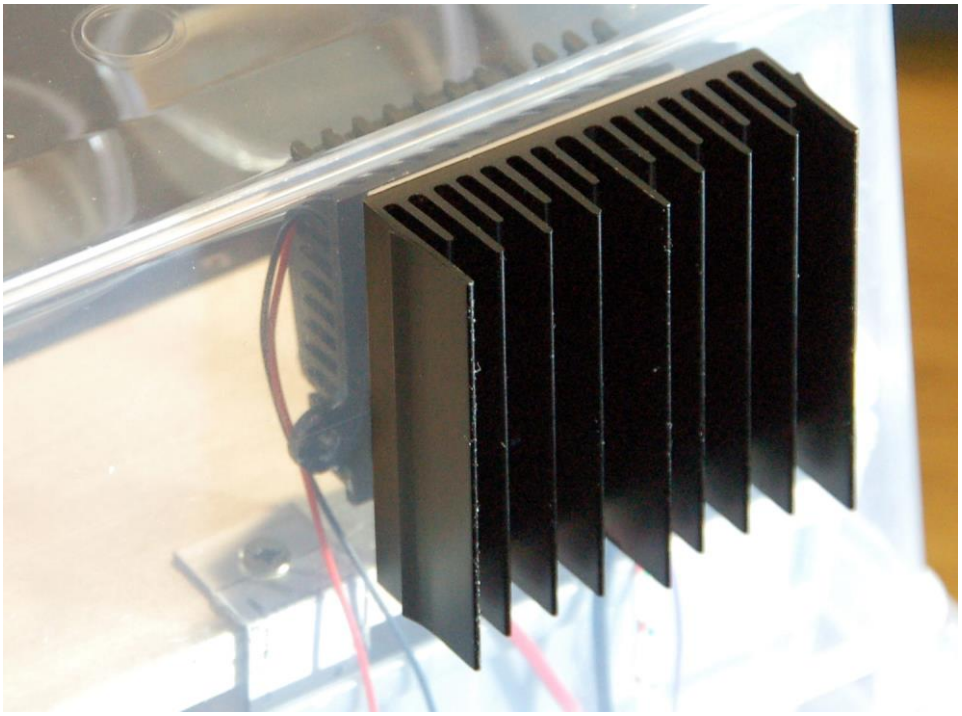

Abbildung 8: Heiz-Kühl-Kombi in der Saatboxabdeckung

Der Lüfter wird recht laut, wenn er mit 5V betrieben wird. Deshalb habe ich einen Schaltregler eingebaut, der es erlaubt, Geräusch und Umwälzwirkung der Luft zu optimieren. Mit zu viel Windgeschwindigkeit würden die zarten Pflänzchen schließlich auch im Bogen wachsen.

Das Peltierelement muss so eingebaut werden, dass die bedruckte Seite nach außen weist, denn die wird kalt, wenn das Element so wie in der Schaltskizze angeschlossen wird und wir wollen ja die Warmseite innen haben. Das Element zieht bei 5V ca. 1,5A. Das ist für einen kleinen Schalttransistor ein bisschen viel. Deshalb habe ich mich für den Einsatz eines Relais entschieden, wie bei den Beleuchtungs-LEDs. Möchte man mehr Heizleistung, dann ist es besser, einen MOSFET-N-Kanal Transistor zusammen mit einer Transistorvorstufe als Treiber einzusetzen und das Peltierelement an 12VDC zu legen. Dann lässt sich die Leistung des Elements mit einem PWM-Ausgang des ESP32 über das Tastverhältnis sogar in der Leistung steuern. Das Relais kann dann auch wegfallen. Bei 12V zieht das Peltierelement dann aber ca. 3,5A Spitzenstrom.

Unser Peltierelement kann zwei Dinge, die eine einfache Heizspirale nicht schafft. Ich habe oben schon erwähnt, dass auf der Warmseite des Elements mehr Wärmeenergie herauskommt, als wir über die elektrische Leistung für die Umsetzung in Wärme am ohmschen Widerstand hineinstecken müssen. Die überzähligen Joule kommen aus der Umgebung. Der Luft um unsere Box herum wird an der Kaltseite von selbst Wärmeenergie entzogen. Warme Umgebungsluft gibt von selbst Wärme an den kälteren Kühlkörper ab. Das Element pumpt diese aufgrund des Peltier-Effekts dann zur Warmseite. Andererseits können wir durch das Umpolen des Stromanschlusses am Element dafür sorgen, dass die Innenseite kalt wird. Damit könnten wir die Box an heißen Tagen kühlen. Dann muss aber an der Außenseite auch ein Lüfter angebracht werden, damit diese jetzt nicht überhitzt.

Das Programm

Für das OLED-Display und den AHT10 holen wir die Klassen **SoftI2C** und **Pin** ins Boot. Die Funktionen **sleep** und **sleep_ms** ermöglichen die Programmierung von Wartezeiten. Die Klasse **OLED** ist die API für das Display und **sys** liefert die Funktion **exit**. Mit einem Objekt der Klasse **OneWire** werden wir eine DS18X20-Instanz erzeugen. Das Thema Temperatur schließt auch die Erfassung der Lufttemperatur mit ein, denn damit steuern wir ja den Einsatz der Heizung. Also brauchen wir beim Importgeschäft auch die Klasse **AHT10**.

```
# temperatur.py
#
from machine import SoftI2C, Pin
from time import sleep, sleep_ms
from oled import OLED
import sys
from onewire import OneWire
from ds18x20 import DS18X20
from aht10 import AHT10
```

Wenn die Variable **debug** auf **True** gesetzt wird, bekommen wir durch die Funktion **klima**() Informationen über den aktuellen Zustand und die Aktionen in der Box während der Entwicklung. Im Produktionsbetrieb ist in der Regel kein PC angeschlossen, dann setzen wir die Variable auf **False**.

```
# ************** Objekte declarieren ********************
#
# debug=True
debug=False
```

Das I2C-Objekt **i2c** übergeben wir dem Konstruktor des Display-Objekts **d**, zusammen mit der Display-Höhe in Pixel. Auch der Konstruktor der Klasse AHT10 braucht das I2C-Objekt.

```
i2c=SoftI2C(scl=Pin(22),sda=Pin(21))
d=OLED(i2c,heightw=32)
aht=AHT10(i2c)
```

Zur Ansteuerung des Relais für das Peltierelement erzeugen wir das Pin-Objekt **heater** an GPIO13. GPIO17 steuert den Transistor an, der den Lüfter schaltet. **taste** ist der digitale Eingang GPIO0, an dem die Flash-Taste des ESP32 liegt.

```
heater=Pin(13,Pin.OUT, value=0)
vent=Pin(17,Pin.OUT, value=0)
taste=Pin(0,Pin.IN,Pin.PULL_UP)
```

Die beiden DS18B20 liegen am OneWire-Bus, den wir am Pin 18 instanziieren und bei der Erzeugung des DS18X20-Objekts brauchen.

```
ds18=14
dsPin=Pin(ds18)
ds = DS18X20(OneWire(dsPin))
```

Um die ROM-Codes eindeutig zuordnen zu können, wird jetzt zunächst nur einer der beiden DS18B20 angeschlossen, zum Beispiel der **Sensor am Kühlkörper**. Dann starte ich das bisher eingegebene Programm mit **F5** und gebe danach im Terminalbreich die Anweisung **ds.scan**() ein. Ich erhalte als Antwort den ROM-Code dieses Chips als bytearray als einziges Element der Liste.

OLED API OK
Size:128x32
AHT10 bereit
>>> ds.scan()
[bytearray(b'(\xffd\x0ei\x0f\x01>)]

Nun schließe ich den zweiten DS18B20 parallel zum ersten an und gebe den Befehl noch einmal ein.

>>> ds.scan()
[bytearray(b'(\xffd\x0ei\x0f\x01>'), bytearray(b'(\x12\xd3\xe9\x1f\x13\x01\x13')]

ds.scan() wird die Sensoren stets in derselben Reihenfolge finden. Die Liste, welche die Methode **scan**() zurückgibt, entpacke ich folglich in die beiden Variablen **heatsink** und **earthtemp**. Stünde in der Liste das zweite Element an erster Stelle, müsste ich die Reihenfolge der Variablen einfach vertauschen.

```
heatsink,earthtemp = ds.scan()
print("Chip-ROM-Codes:",earthtemp,heatsink)
```

Es folgen die Deklarationen einiger Funktionen.

```
def heizung(val=None):
    assert val in (0,1,None)
    if val is not None:
        heater.value(val)
    else:
        return state[heater.value()]
```

Der optionale Parameter der Funktion **heizung**() hat den Defaultwert **None**. Rufe ich **heizung**() ohne Argument auf, also mit leeren Klammern, dann liefert der Aufruf den gegenwärtigen Stand des GPIO-Ausgangs zurück. Andernfalls wird der Ausgang auf den übergebenen Wert gesetzt. Mit **assert** überprüfe ich, ob der Inhalt von **val** einem erlaubten Wert entspricht. Ist das nicht der Fall, wird ein **AssertationError** geworfen, den das aufrufende Programm abfangen sollte, damit durch den Fehler kein Abbruch erfolgt.

Die Funktion **luefter**() arbeitet nach demselben Schema.

```
def luefter(val=None):
    assert val in (0,1,None)
    if val is not None:
        vent.value(val)
    else:
        return state[vent.value()]
```

Die Funktion **temperatur**() nimmt als Argument einen der ROM-Codes in **heatsink** oder **earthtemp**. Zunächst wird mit **convert**() eine Messung angestoßen. 750 Millisekunden später steht das Ergebnis in beiden Bausteinen bereit. Zum Auslesen durch die Methode **read_temp**() muss der entsprechende Chip adressiert werden. Das geschieht durch die Übergabe des ROM-Codes in **c**.

```
def temperatur(c):
    ds.convert_temp()
    sleep_ms(750)
    return ds.read_temp(c)
```

Die Funktion klima() ist ziemlich komplex, sie muss ja auch alle möglichen Fälle abdecken. Jedenfalls holen wir am Start schon einmal die Messwerte ein. Die Temperaturen geben wir im Display aus und, falls debug den Wert True hat, die Lufttemperatur auch im Terminal. Dort erhalten wir durch weitere derartige if-Konstrukte ein Protokoll der Arbeitsweise.

```
def klima():
    tempKK=temperatur(heatsink)
    tempSubstrat=temperatur(earthtemp)
    aht.triggerDevice()
    temp=aht.getTemp()
    feuchte=aht.getHum()
    d.writeAt("TEMP_AIR: {0:>4.1f}*C".format(temp),0,0)
    d.writeAt("TEMP_KK: {0:>4.1f}*C".format(tempKK),0,1)
    d.writeAt("TEMP_SUB: {0:>4.1f} %".\
              format(tempSubstrat),0,2)
    if debug: print("Luft-Temperatur: {:0.2f}".format(temp))
```

Für den weiteren Verlauf sagt das Flussdiagramm der Funktion **klima**() mehr als Worte.

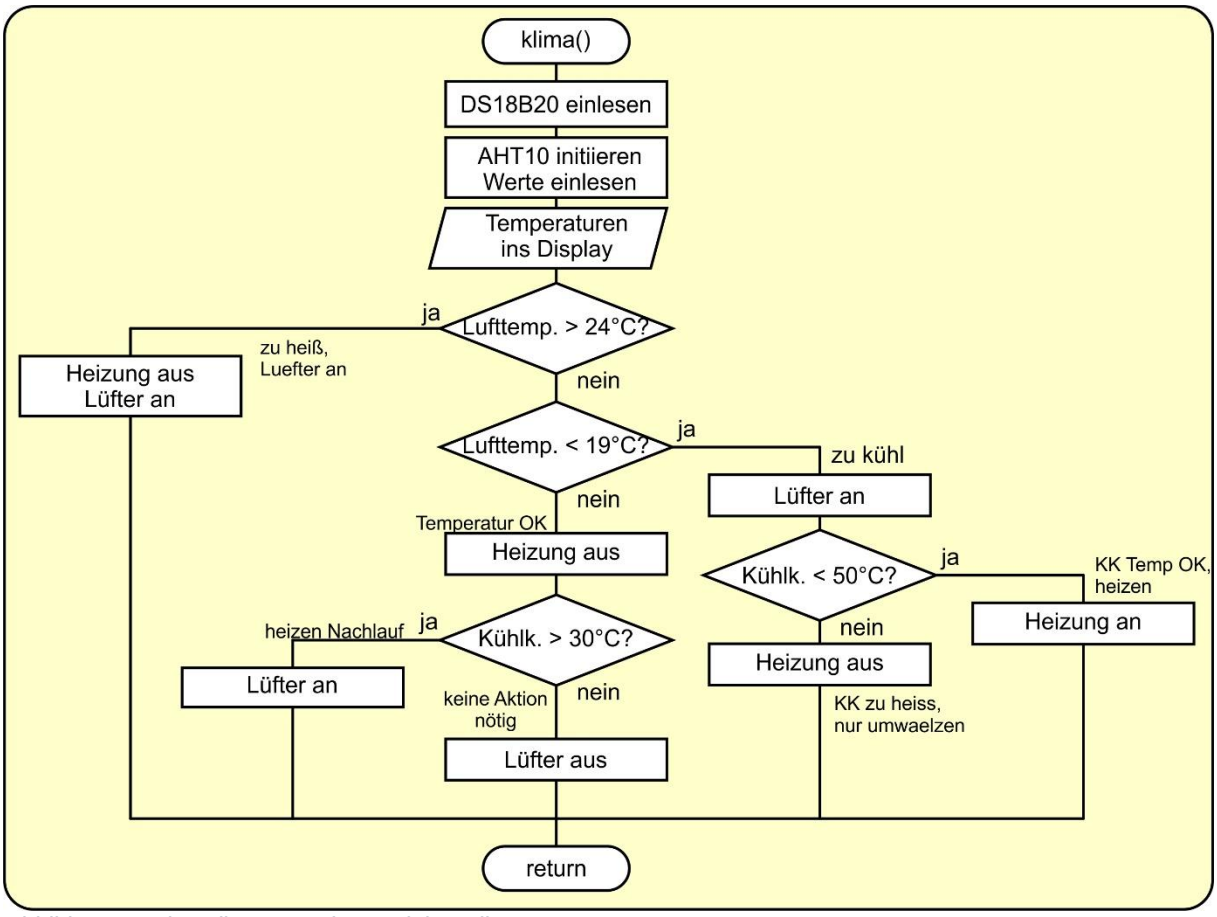

Abbildung 9: Flussdiagramm der Funktion Klima

```
    if temp >= 24:
        if debug: print("zu heiß, Luefter an")
        luefter(1)
        heizung(0)
    elif temp < 19: #
        if debug: print("zu kühl")
        if debug: print("KK-Temp:",tempKK)
        luefter(1)
        if tempKK <= 26:
            if debug: print("KK Temp OK, heizen")
            heizung(1)
        else:
            if debug: print("KK zu heiss, nur umwaelzen")
            heizung(0)
    else:
        if debug: print("Temperatur OK")
        heizung(0)
        if tempKK >= 25:
            if debug: print("heizen Nachlauf")
            luefter(1)
        else:
            if debug: print("No Action needed")
            luefter(0)
```

Die Hauptschleife ist wie üblich sehr übersichtlich. Ich rufe nur die Funktion **klima**() auf und frage die Flash-Taste ab. Ist sie gedrückt, dann erscheint im Display die Meldung **PROGRAM CANCELED**, Heizung und Lüfter werden ausgeschaltet und die entsprechenden Meldungen im Display dargestellt.

Mit diesem Beitrag haben wir nun alle Baugruppen für das Saatbox-Projekt beieinander. In der nächsten Folge stellen wir die Teile zu einem Programm zusammen. Weil Sie, wie ich auch, sicher keine Lust haben, stets um das Display herumzuschwirren, um auf dem Laufenden zu sein, habe ich mir eine Android-App gebastelt, die mir den Status der Box anzeigt. Auf Wunsch, könnte man sogar eine Fernsteuerung der Aktoren mit integrieren, zum Beispiel Heizung an / aus.

Viel Vergnügen beim Basteln und Programmieren.

Bis dann