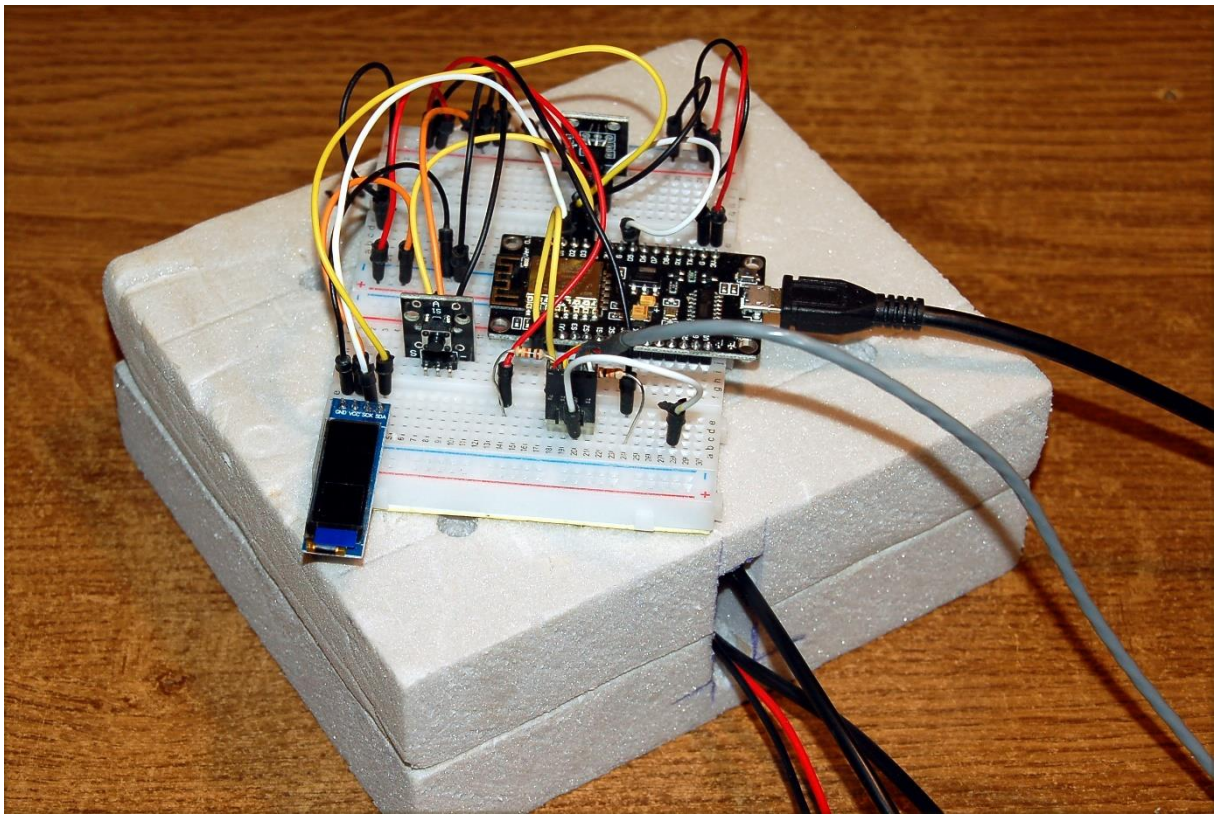


Abbildung 1: Wattmeter_Aufbau

Heimlich und unerkannt schleichen sie ins Zimmer. Sie können willkommen sein, aber auch höchst überflüssig erscheinen. Im Sommer halten wir sie lieber draußen, im Winter sind wir froh, ein gewisses Quantum davon im Zimmer zu haben. Die Rede ist von Calorien, oder mit der aktuellen physikalischen Sprechweise ausgedrückt, von Joule (sprich: "dschuul"). Herzlich willkommen im Club der Energiedetektive. Heute sind wir

Den Calorien auf der Spur

Ein Joule (= 4,16 Cal) ist die Einheit der Energie und kann als abgeleitete Größe, je nach Anwendungsgebiet, durch andere Einheiten ausgedrückt werden, elektrisch 1J = 1VAs, mechanisch 1J = 1Nm. Wir werden heute eine einfache Schaltung herstellen, mit der wir die thermische Energie auf ihrem Weg von hier nach dort beobachten und messen können. Denn die Gesamtenergie in einem Raum oder einem Gefäß kann man schlecht bestimmen, weil zu viele Parameter eine Rolle spielen. Aber Energie die unterwegs ist, durch Strahlung oder thermischen Kontakt, können wir dagegen recht gut erfassen. Damit Energie für uns messbar wird, brauchen wir ein Gerät, das wir in den Weg, den die Energie nimmt, einschleusen können. Zum besseren Verständnis kann man einen Blick auf die Elektrizität werfen und einen Vergleich anstellen.

Elektrische Ladungen (Elektronen), die sich durch einen Leiter bewegen, bezeichnen wir als elektrischen Strom. Sammeln sie sich auf einer Oberfläche, sprechen wir von deren Aufladung oder kurz Ladung Q. Die gesamte Ladung Q, die auf einem Körper sitzt, kann man nicht direkt messen, aber man kann die Stromstärke I und die Zeit t

messen, während die Ladungen durch einen Leiter auf die Oberfläche wandern oder sie verlassen. Ladungsströme messen wir mit einem Amperemeter, das wir direkt in den Stromkreis, den Weg, den die Ladungen nehmen, einfügen, es erfasst die Ladungsmenge Q, die pro Zeiteinheit den Leiterquerschnitt A an der Messstelle passiert.

$$I = \frac{\text{Ladungsmenge Q}}{\text{Zeit t}}$$

Abbildung 2: Definition Stromstärke

$$[\,I\,] = \frac{1\ C}{1\ s} = 1\ A$$

Abbildung 3: Einheit Stromstärke

Ein Gerät, um Energieströme zu messen, werden wir heute bauen, es erfasst Energiemengen in Joule, die pro Zeiteinheit den Querschnitt A der Messstelle passiert. Daraus ergibt sich eine neue Größe. Denn allgemein ist der Quotient aus Energie oder Arbeit durch Zeit als Leistung P definiert.

$$P = \frac{\text{Wärmemenge } W_{th}}{\text{Zeit t}}$$

Abbildung 4: Definition Wärmestrom - Wärmeleistung

$$[\,P\,] = \frac{1\ J}{1\ s} = 1\ \text{Watt} = 1W$$

Abbildung 5: Einheit Wärmestrom – Wärmeleistung

Und so, wie ein Amperemeter in einen Stromkreis eingebaut wird, werden wir unser thermisches Watt-Meter in Energieströme einschleusen. Wenn wir dann noch die Zeit messen, können wir berechnen, wie viele Joule insgesamt von einem Körper abgegeben oder aufgenommen wurden. Ein anderes Analogon wäre der Swimmingpool, die Füllmenge an Wasser, die Zuleitung und die Wasseruhr.

Was wir für das Wattmeter brauchen

Die Hardware

1*	0,91 Zoll OLED I2C Display 128 x 32 Pixel
1*	NodeMCU Lua Amica Modul V2 ESP8266 ESP-12F
1	TEC1-12706 Thermoelektischer Wandler Das 5-er-Bundel ist sehr günstig in Hinblick auf den Folgebeitrag
1	KY-004 Taster Modul Sensor Taste 3er Bundle ist günstiger als 2 Einzeltaster
1	Mini Breadboard 400 Pin mit 4 Stromschienen für Jumper Kabel 5er-Bundel ist günstiger als Einzelstück oder 3er-Bundle
1	DS18B20 digitaler Temperatursensor TO92-55°C - +125°C oder besser gleich mit für den Folgebeitrag bestellen 1M Kabel DS18B20 digitaler Edelstahl Temperatursensor Temperaturfühler, wasserdicht
3	Widerstand 10kOhm
div.	Jumperkabel
1	USB-A-Stecker auf Micro-Stecker
Reste	von Stiftreihen für DS18B20 und Peltier-Element
1	Netzteil mit 5..15V / >=3A zum Beispiel den DC-DC-Wandler aus der vorigen Blogfolge

Neben der Hardware wird etwas Software gebraucht. Zum einen die MicroPython-Firmware für den ESP8266 ganz allgemein und natürlich ganz speziell die Programme zum Eichen des Controllers als Wattmeter und dann zum Messen von Wärmeströmen.

Unser Gerät wird in der Lage sein, Wärmeströme mit einer Auflösung von ca. 25mW zu erfassen. Das wäre durch den Einsatz eines Vorverstärkers oder eines ESP32 auch noch zu verbessern.

Die Software

Verwendete Software:

Fürs Flashen und die Programmierung des ESP32:
Thonny oder
µPyCraft
micropython-font-to-py

Verwendete Firmware:

MicropythonFirmware
Bitte eine Stable-Version aussuchen

MicroPython-Programme

Module:
button.py zur Bedienung von Tasten
charset.py die Steuerklasse für größere Zeichensätze
geometer_24.py stellt den Zeichensatz geometer.ttf in Punkt24 auf dem OLED dar
oled.py, die Klasse OLED enthält Methoden zur bequemen Ansteuerung von OLED-Displays
ssd1306.py ist der Low-Level Treiber für das verwendete Display
Anwendungen:
eichen.py zur Ermittlung der Betriebsdaten des Peltierelements
wattmeter.py Messprogramm zur Erfassung von Wärmeströmen
font2py.rar Paket zur Erzeugung von Pixelzeichensätzen aus TTF-Fonts

MicroPython - Sprache - Module und Programme

Zur Installation von Thonny finden Sie hier eine ausführliche Anleitung. Darin gibt es auch eine Beschreibung wie die MicropythonFirmware 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, bevor 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 compilieren 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 Macro 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 ESP8266 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 compiliertes Programm aus der Arduino-IDE oder die AT-Firmware oder LUA oder … enthält, problemlos mit der MicroPython-Firmware versehen werden. Der Vorgang ist immer wie hier beschrieben.

Aufbau der Schaltung und der Peripherie

Wir hatten weiter oben davon gesprochen, dass man unser thermisches Wattmeter mit einem Amperemeter aus dem elektrischen Sektor vergleichen kann. Vergleichbar ist auch direkt die Arbeitsweise. Die ohmsche Widerstandsformel besagt, dass durch einen Widerstand ein elektrischer Strom fließt, wenn man eine Spannung anlegt. Ein digitales Strommessgerät bedient sich der Umkehrung dieses Satzes. An einem Widerstand fällt eine Spannung ab, wenn er von einem Strom durchflossen wird.

$$U = I \bullet R$$

Abbildung 6: Spannung am Widerstand

Diese Spannung ist direktproportional zur Stromstärke. Der Widerstand im Stromkreis ist somit der Stromstärkesensor, der die Stromstärke mittelbar über die Spannung messbar macht. Das Messgerät hat die Aufgabe, die Spannung am Sensor durch Kenntnis des Widerstandswerts in einem Stromstärkewert umzurechnen und als solchen anzuzeigen.

Unser Wärmestrommesser arbeitet genauso, nur mit einem anderen Effekt, einem anderen Sensor. Wir nutzen den Seebeck-Effekt eines thermoelektrischen Wandlers, auch bekannt unter dem Namen Peltierelement. Diese Bauteile werden primär dazu benutzt, um über den Peltier-Effekt elektrische Energie in einen Wärmestrom umzuwandeln. Das nutzen wir im dritten Beitrag für den Getränkekühler aus. Dieser Peltier-Effekt lässt sich aber auch umkehren, heraus kommt der Seebeck-Effekt.

Leitet man einen Wärmestrom durch ein Peltierelement, dann entsteht an dessen elektrischen Anschlüssen eine Spannung, die zur Temperaturdifferenz an den Flächen und damit auch zum Wärmestrom direkt proportional ist. Diesen Seebeck-Effekt nutzen wir für unseren Wärmestrommesser. Die physikalische Einheit eines Wärmestroms ergibt sich aus Wärmearbeit / Zeit als Wärmeleistung. Es gilt:

$$(I) \quad P_{th} = k \bullet U_{th}$$

Abbildung 7: Thermospannung

Die Idee dazu lieferte ein Beitrag von Werner B. Schneider und H. Dittmann, erschienen in Band 4, Wege in der Physikdidaktik, Erlangen 1998. Dort wird auch beschrieben, wie der Eichvorgang, ablaufen kann, denn als Erstes müssen wir das k aus der Gleichung (I) bestimmen, damit danach der ESP8266 die Spannung in einen Wärmestrom umrechnen und anzeigen kann.

Aber dazu brauchen wir erst einmal eine Schaltung und ein paar weitere Infos zu deren Umgebung. Das Schaltschema können Sie als PDF-Datei in DIN A4 herunterladen.

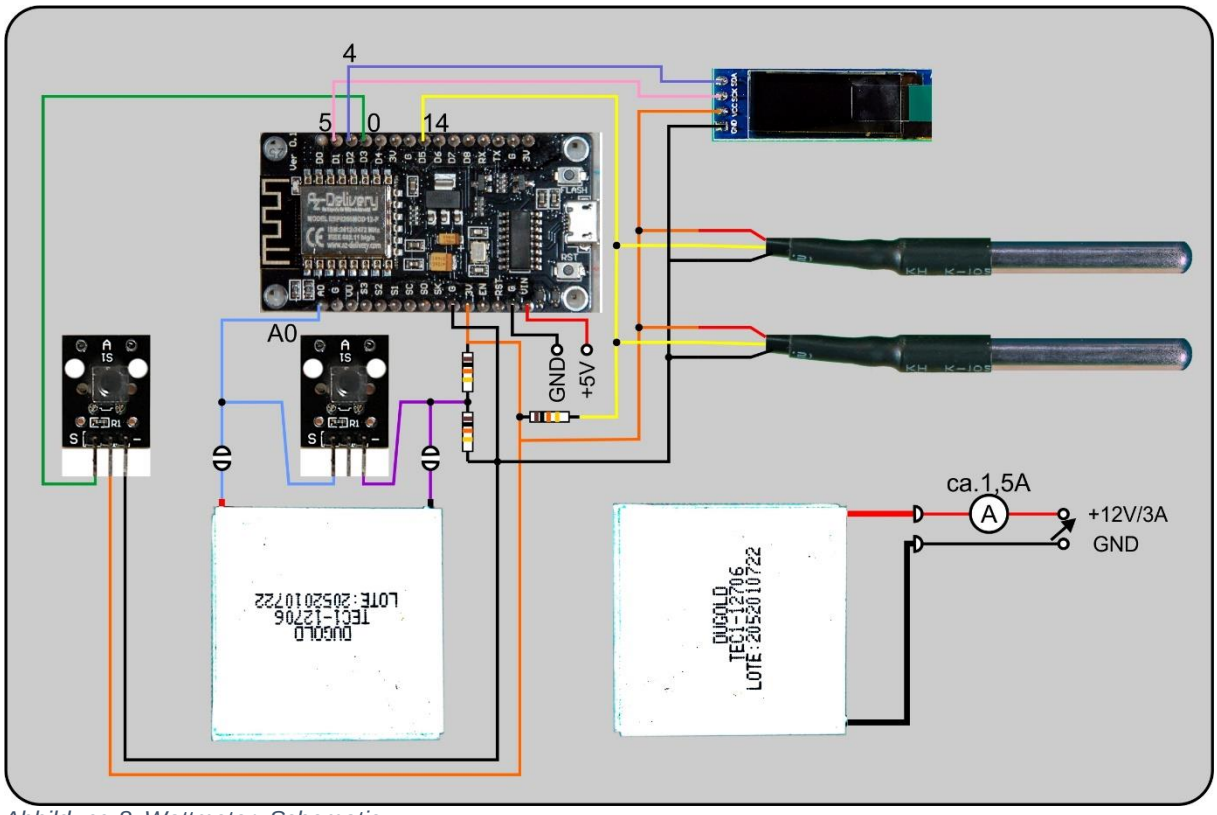


Abbildung 8: Wattmeter_Schematic

Die Wärmeleitungsgleichung, die hinter der Seebeck-Formel steht, verwendet primär anstelle der Thermospannung den Temperaturunterschied zwischen den beiden Seiten des Peltierelements.

$$(II)\quad Q = \lambda \bullet \frac{A}{l} \bullet t \bullet \Delta T$$

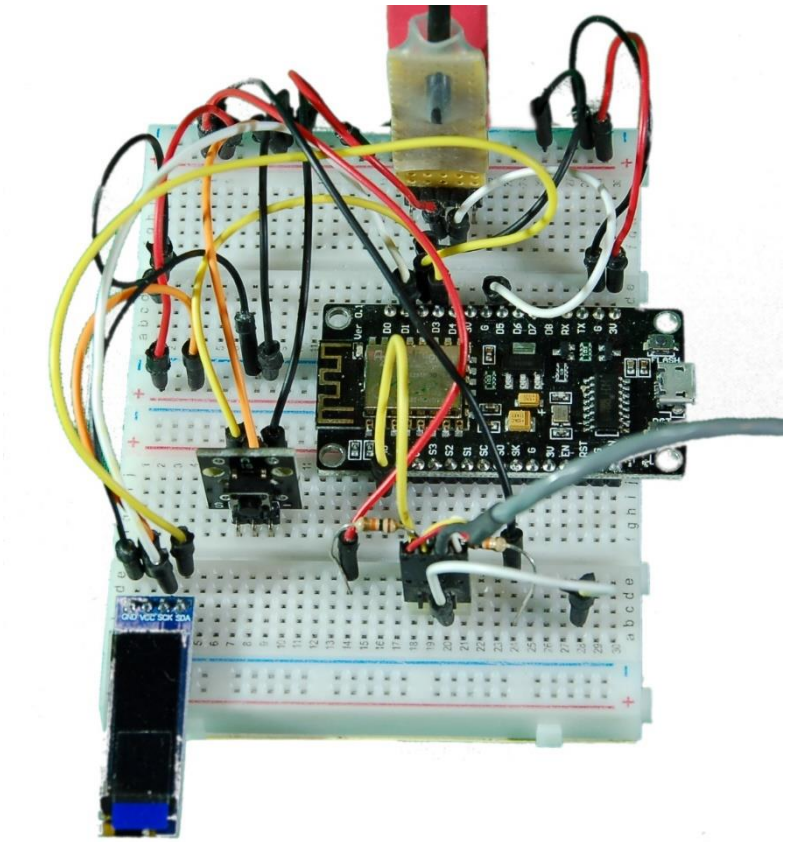
Abbildung 9: Wärmeleitungsgleichung

Die Thermospannung selbst ist auch direkt proportional zum Temperaturunterschied.

$$(III)\quad U_{th} = \beta \bullet \Delta T$$

Abbildung 10:Thermospannung und Temperaturdifferenz

β und λ sind Materialkonstanten. Mit der Fläche A und der Dicke l des Peltierelements sowie durch Umstellen und Zusammenfassen der Konstanten ergibt sich schließlich die Gleichung (I).



In unserem Aufbau messen wir die Temperaturen zweier Alu-Quader, sie stellen unsere Wärmereservoirs dar, die wir zu Beginn der Calibrierung auf Temperaturen etwas über und unter der Raumtemperatur bringen. Das verhindert zu großen Wärmeaustausch mit der Umgebung, was zu Messfehlern führen würde. Darüber hinaus packen wir die beiden Körper, zwischen denen das Peltierelement zu liegen kommt, in Schaumstoffpolster ein. Die Dübel sichern den Deckel und damit auch den oberen Quader sowie das Thermoelement gegen Verrutschen.

Abbildung 11: Wattmeter-Aufbau_calibrieren

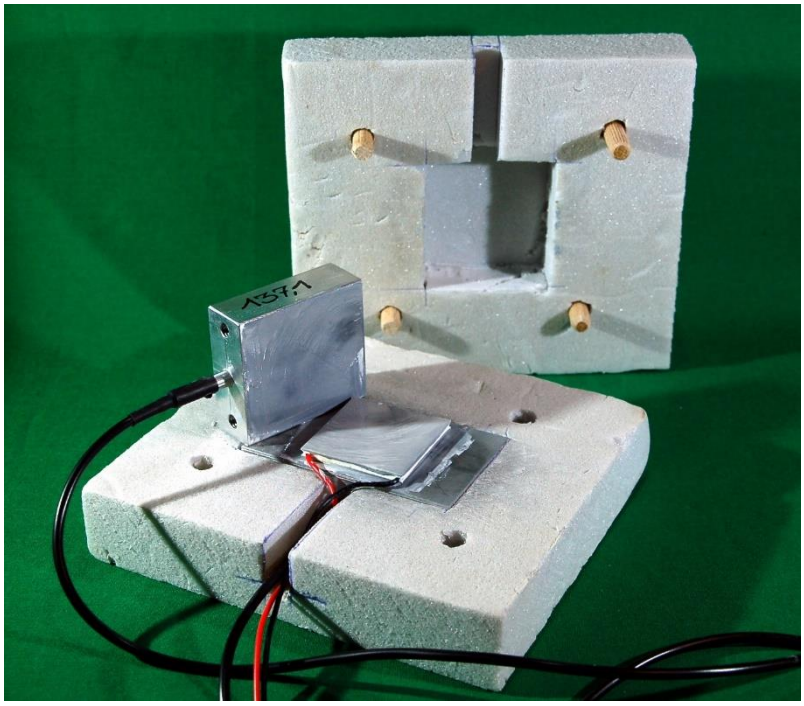


Abbildung 12: Peltierelement mit DS18B20-Fühlern

Das Isoliermaterial ist aus Hartschaum. Die Vertiefungen wurden mit einer Proxxon und einem Zylinderfräser auf dem Bohrständer hergestellt und die Kanten mit einem Cutter nachbearbeitet. Die Aluquader wurden mit einem 6mm-Präzisionsbohrer so tief gebohrt, dass die Kuppe des Zylinders mit dem DS18B20 in der Mitte zu liegen kommt. Etwas Silikon-Wärmeleitpaste sorgt dann für guten thermischen Kontakt zwischen Sensor, Aluquader und Peltierelement. Die Masse m_{hot} (=137,1g, oberer Quader) und m_{cold} (=172,8g, unterer Quader im Schaumstoffbett) sowie der c-Wert von Aluminium c_{Alu} = 0,896 J/(g K) müssen bekannt sein. Ihre Massewerte werden sich sicher von meinen unterscheiden. Meine Quader sind 20mm dick. Sie müssen das Element gut überdecken.

Die Eichung des Thermoelements als Wattmetersensor

Lassen Sie uns nun eine Messreihe starten, deren Ergebnisse uns den begehrten Proportionalitätsfaktor k aus der Formel (I) und den Seebeckkoeffizienten α liefern. Wir verwenden dazu das Programm eichen.py. Alle oben genannten Module müssen sich im Flash des ESP8266 befinden.

Im Vorfeld bestimmen wir die Zuordnung der DS18B20. Die Sensoren werden vom Programm automatisch erkannt. Wir starten ohne Calibrierung durch und prüfen durch Berühren eines Sensors mit der Hand, ob der linke Wert in der Liste im Terminal steigt. Dieser Sensor kommt in den oberen Quader, der andere in den unteren. Das erleichtert später die Zuordnung. Der obere Quader gilt als der wärmere bei der späteren Messung.

```
# thermometer.py
# Author: J. Grzesina
# Rev: 1.0
# Stand: 2021-06-22
# ********************************************************
from machine import Pin, I2C, ADC, Timer
from time import sleep, ticks_ms,sleep_ms
from onewire import OneWire
from ds18x20 import DS18X20
from oled import OLED
from button import BUTTON8266, BUTTONS

# Pintranslator fuer ESP8266-Boards
# LUA-Pins     D0 D1 D2 D3 D4 D5 D6 D7 D8
# ESP8266 Pins 16  5  4  0  2 14 12 13 15
#                 SC SD

adc=ADC(0)

SCL=Pin(5)
SDA=Pin(4)
i2c=I2C(-1,SCL,SDA)
d=OLED(i2c,128,32)
d.clearAll()

ds_pin = Pin(14)     # D5@esp8266
ds = DS18X20(OneWire(ds_pin))
chips = ds.scan() # Liste von bytearrays
numberOfChips = len(chips)
print('Found DS devices: ')
for chip in chips:
    print(chip)

t0,t1=0,0
def los_messen(tim):
    global timerFlag
    timerFlag=True
u=0
```

```
u0=0
U=""
def getTh(n=5):
    s=0
    for i in range(n):
        s+=adc.read()
    s=s//n
    u=(int(s/1023*3.0*10000))/10  # Spannung in mV
    return u

taste = BUTTON8266(0,invert=True) # D3@esp8266
k=BUTTONS()

d.clearAll()
d.writeAt("THERMOSENSOREN",0,0,False)
d.writeAt("KALIBRIEREN",0,1,False)
d.writeAt(">>> Taste",0,2)

act=k.waitForTouch(taste,3)
if act:
    sleep(1)
    k.waitForTouch(taste,6)
    d.clearAll()
    d.writeAt("Sensor A + B in",0,0,False)
    d.writeAt("Wasser tauchen",0,1,False)
    d.writeAt(">>> Taste",0,2)
    sleep(1)
    k.waitForTouch(taste,6)
    d.clearAll()
    d.writeAt("Temperaturen",0,0,False)
    d.writeAt("konstant?",0,1,False)
    d.writeAt(">>> Taste",0,2)
    sleep(1)
    while taste.tpin.value()==1:
        ds.convert_temp()
        u0=getTh(20)
        sleep(1)
        t0 = (int((ds.read_temp(chips[0]))*100))/100
        t1 = (int((ds.read_temp(chips[1]))*100))/100
        print(t0,t1)
        d.clearAll()
        d.writeAt("A {}".format(t0),0,0,False)
        d.writeAt("B {}".format(t1),0,1,False)
        d.writeAt("U {}".format(u0),0,2)
        sleep(2)
    dt=t0-t1
    d.clearAll()
    d.writeAt("Calibrat. done",0,0,False)
    d.writeAt("Release key!".format(dt),0,1,False)
    d.writeAt("U0={} dT={}".format(u0,dt),0,2)
    sleep(3)
    f=open("calibration.txt","w")
```

```
    f.write(str(dt)+"\n")
    f.write(str(u0)+"\n")
    f.close()
    f=None
else:
    f=open("calibration.txt","r")
    print("Position",f.tell())
    dt=float(f.readline())
    u0=float(f.readline())
    f.close()
    f=None

print("dt = {}".format(dt))
print("u0 = {}".format(u0))

sleep(1)
T=Timer(0)
timerFlag= True
intervall=10000
T.init(mode=Timer.PERIODIC,period=intervall, \
       callback=los_messen)
firstRun=True
start=ticks_ms()
while taste.tpin.value() == 1:
    if timerFlag==True:
        timerFlag=False
        ds.convert_temp()
        zeit=ticks_ms()-start
        s=getTh(20)
        u=s-u0  # Spannung in mV
        uString=(str(u)).replace(".",",")
        sleep_ms(750)
        t0 = (int((ds.read_temp(chips[0]))*100))/100
        t1 = dt+(int((ds.read_temp(chips[1]))*100))/100
        dT=t0-t1
        f="{:.2f}"
        dT = f.format(dT)
        t0 = f.format(t0)
        t1 = f.format(t1)
        dTString = dT.replace(".",",")
        t0String = t0.replace(".",",")
        t1String = t1.replace(".",",")
        f="{:>8};{:>6};{:>6};{:>6};{:>6}"

print(f.format(str(zeit),t0String,t1String,dTString,uString))
        d.clearAll()
        d.writeAt("A {}".format(t0),0,0,False)
        d.writeAt("B {}".format(t1),0,1,False)
        d.writeAt("U {}".format(u),0,2)
T.deinit()
d.clearAll()
d.writeAt("PROG DONE",0,0)
```

Das Programm enthält keine exotischen Code-Teile. Die Zeitsteuerung erfolgt über einen Timerinterrupt, dessen Interrupt-Service-Routine (ISR) ein Flag setzt. Dieses Flag sagt dem Hauptprogramm, dass eine Messung durchzuführen ist. Der IRQ läuft im Continuous Mode, bis das Programm mit der Taste an GPIO0 (Abbruchtaste) abgebrochen wird.

Beim Start sollten die beiden Aluquader gleiche Temperatur haben. Damit sowohl positive wie negative Spannungen am Thermoelement gemessen werden können, ist der negative Anschluss nicht an GND sondern an die Mitte des Spannungsteilers aus zwei 10kΩ-Widerständen gelegt. Die Mittenspannung muss bestimmt werden, damit der ESP8266 sie später bei den Messungen vom Messwert abziehen kann. Uns interessiert ja nur die Spannung am Peltierelement.

Dennoch ergeben sich, trotz ausreichenden Wartens, durch die beiden DS18B20-Sensoren leicht unterschiedliche Temperaturwerte und durch Rauschen auf der Leitung auch Schwankungen der ADC-Werte. Um diesen Effekt zu mildern tastet der ADC den Eingang 20-mal ab und berechnet daraus einen Mittelwert. Das passiert in der Funktion **getTh**().

Vor den Messungen kann deshalb eine Calibrierung durchgeführt werden, über die das Display informiert. Die Calibrierdaten werden in ein File geschrieben, das bei weiteren Messvorgängen eingelesen wird. Zur Calibrierung muss das Thermoelement nicht angeschlossen sein. Aber während des Vorgangs muss die Taste, die parallel dazu liegt gedrückt werden, oder wir schließen die Strecke mit einem Jumperkabel kurz. Wenn sich die Werte im Display nicht mehr wesentlich verändern, drücken wir die Taste an GPIO0.

Was geschieht nun weiter?

Das Thermoelement muss jetzt in jedem Fall vom ESP8266 abgetrennt sein, andernfalls wird der Controller sterben! Die Beschriftung des Peltierelements sollte nach unten zeigen. Dann stellen wir am Netzteil einen Wert zwischen 5V und 6V ein und schalten die Stromzufuhr zum Wandler ab. Wir legen den Pluspol des Wandlers an den roten Anschluss des Thermoelements, den Minuspol an den schwarzen.

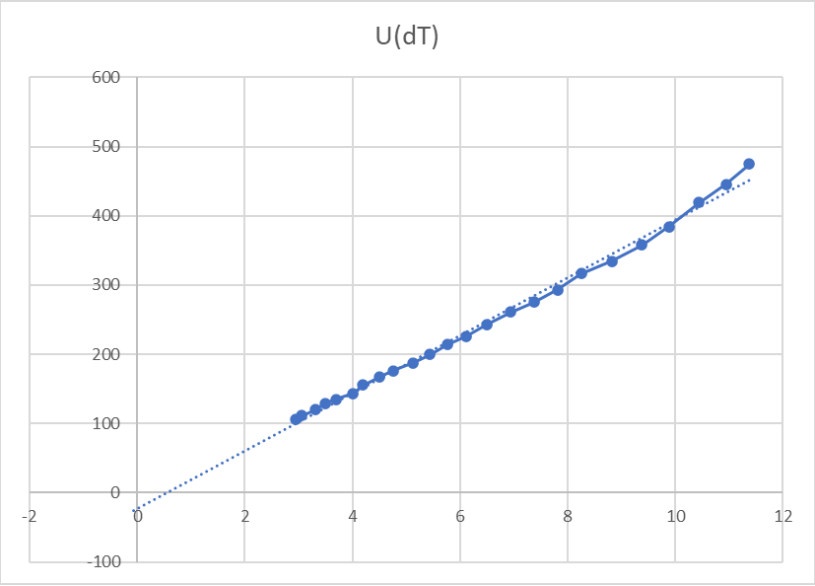
Das Programm eichen.py wird gestartet. Wir warten, bis im Display und im Terminal die Temperaturen der Alublöcke und die Spannung angezeigt werden. Der Temperaturwert des unteren Blocks wird notiert.

Jetzt wird die Stromzufuhr zum Wandler eingeschaltet. Sind die Temperaturen um gut 5 Grad Celsius gestiegen und gefallen, schalten wir die Stromzufuhr aus und trennen die Verbindung zwischen Wandler und Thermoelement. Wir stoppen das Programm mit der Abbruchtaste und starten es neu. Das Thermoelement wird mit dem ESP8266 verbunden. Im 10-Sekunden-Abstand wird Zeile um Zeile ausgegeben bis die Temperatur des kalten Würfels gerade wieder die Anfangstemperatur erreicht hat. Die Messung wird gestoppt.

Hier ist meine Messwerttabelle, die um die Spalte deltaT erweitert wurde, welche die Differenz zwischen warmem und kaltem Quader angibt. Die farbig markierten Werte verwenden wir zur Berechnung.

Zeit in	oben	unten		
Millisekunden	Thot	Tcold	deltaT	U
130003	29,43	18,06	11,37	475,1
140003	29,25	18,31	10,94	445,7
150003	28,93	18,49	10,44	419,3
160003	28,62	18,74	9,88	384,1
170003	28,31	18,93	9,38	357,8
180003	28	19,18	8,82	334,3
190003	27,68	19,43	8,25	316,7
200003	27,43	19,62	7,81	293,2
210003	27,18	19,81	7,37	275,6
220003	26,93	19,99	6,94	261
230003	26,68	20,18	6,5	243,4
240003	26,43	20,31	6,12	225,8
250003	26,25	20,49	5,76	214,1
260003	26,06	20,62	5,44	199,4
270003	25,87	20,74	5,13	187,7
280003	25,68	20,93	4,75	175,9
290003	25,56	21,06	4,5	167,1
300003	25,37	21,18	4,19	155,4
310003	25,25	21,24	4,01	143,7
320003	25,06	21,37	3,69	134,9
330003	24,93	21,43	3,5	129
340003	24,87	21,56	3,31	120,2
350003	24,68	21,62	3,06	111,4
360003	24,62	21,68	2,94	105,6

Die Grafik zeigt einen linearen Zusammenhang auf, der eigentlich rückläufig zu betrachten ist.



Jetzt muss gerechnet werden.

Das Wärmefassungsvermögen (aka Wärmekapazität) der Quader
$C_{hot} = c_{Alu} \cdot m_{hot} = 0{,}896\ J/(gK) \cdot 137{,}1g = 122{,}84\ J/K$
$C_{cold} = c_{Alu} \cdot m_{cold} = 0{,}896\ J/(gK) \cdot 172{,}8g = 154{,}83\ J/K$

Hier sind die in der Grafik markierten Zeilen zusammengefasst.

		oben	unten	
	t in s	T(hot) in °C	T(cold) in °C	U in mV
	130003	29,43	18,06	475,1
	360003	24,62	21,68	105,6
Differenz	230	4,81	3,62	369,5

Die mittlere Thermospannung
$U_{mid} = (475{,}1 + 105{,}6)/2\ mV = 290{,}35\ mV = 0{,}290\ V$

Die vom oberen Quader abgegebene innere Energie
$W_{th\_ab} = C_{hot} \cdot (29{,}43 - 24{,}62)\ K = 122{,}84\ J/K \cdot 4{,}81\ K = 590{,}87\ J$

Die Wärmeleistung des Quaders
$P_{th} = W_{th\_ab} / \Delta t = 590{,}87\ J / 230s = 2{,}57W$

und schließlich unser k-Wert
$k = P_{th} / U_{mid} = 2{,}57\ W / 0{,}290\ V = 8{,}8\ W/V$

und weiter

Der Mittelwert der Temperaturdifferenzen deltaT ist

$\Delta T_{mid} = (11{,}37+2{,}94)/2\ K = 7{,}16\ K$

$\beta = \Delta U / \Delta T_{mid} = 369{,}5mV / 7{,}16K = 51{,}64\ mV/K$

Bei 127 Elementen hat man 254 Kontaktstellen.

$\alpha = 51{,}64\ mV / 254 = 203\mu V/K$

Das liegt innerhalb der Herstellerangaben im Datenblatt. Perfekt!

Für uns am wichtigsten ist aber der k-Wert, denn damit können wir unseren Aufbau zum Wattmeter umfunktionieren. Pro Volt Thermospannung werden je Sekunde 8,8 J von der warmen zur kalten Seite transportiert. Unser ESP8266 kann 3,0V / 1024 LSB = 2,93 mV/LSB auflösen. Das entspricht 8,800 W/V • 0,00293V/LSB = 25 mW/LSB. Das heißt, wir messen in 25mW-Stufen. Wie an anderer Stelle angemerkt, lässt sich die Auflösung steigern, wenn man zusätzlich einen Operationsvorverstärker und/oder statt des ESP8266 einen ESP32 verwendet.

Das Wattmeter im Einsatz

Zur Realisierung des thermischen Joule- und Wattmeters ist jetzt nur noch ein kleiner Schritt. Das eben getestete Thermoelement wird herausgenommen und gereinigt. Die Siliconpaste muss gründlich entfernt werden, damit sich das Element in einen Plexiglashalter wie im folgenden Foto einkleben lässt. Bei den Messvorgängen darf keine Berührung mit den Fingern erfolgen, weil sonst unkontrollierte Wärmeströme fließen würden, die das Ergebnis verfälschen.

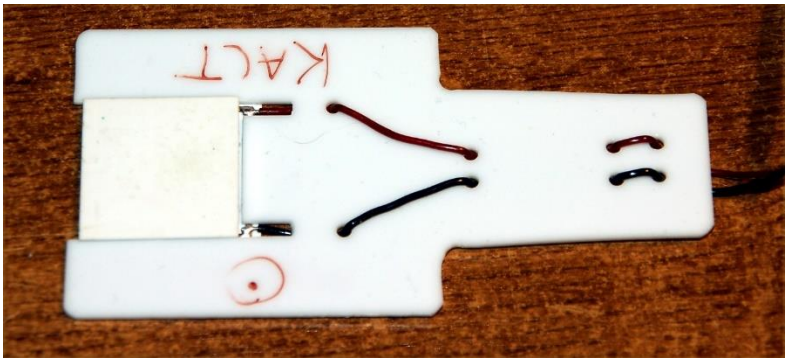


Abbildung 13: Wattmeter - Thermoelement als Fühler

Hält man eine Seite gegen eine glatte Fläche, dann verrät uns das nächste Programm, wie viele Joule pro Sekunde von der wärmeren Seite zur kälteren fließen. Die kleinere Zeile darunter gibt Auskunft über die Umgebungstemperatur. Über die Sensorfläche von 4 x 4 = 16 cm² kann man dann hochrechnen auf die ganze getestete Fläche, zum Beispiel Fenster, Wände, Backofen, Haustüren … Der Test an Fenstern mit und ohne Rollladen zeigt gravierende Unterschiede des Energieflusses im Sommer wie im Winter.

Wird der einseitig geschwärzte Sensor mit Wärmeleitkleber gegen einen großen Kühlkörper geklebt, den man in einen Styroporklotz einbaut und mit einem Kollimatorrohr versieht, dann kann damit auch sehr empfindlich Wärmestrahlung gemessen werden.

Abbildung 14: Strahlungssensor

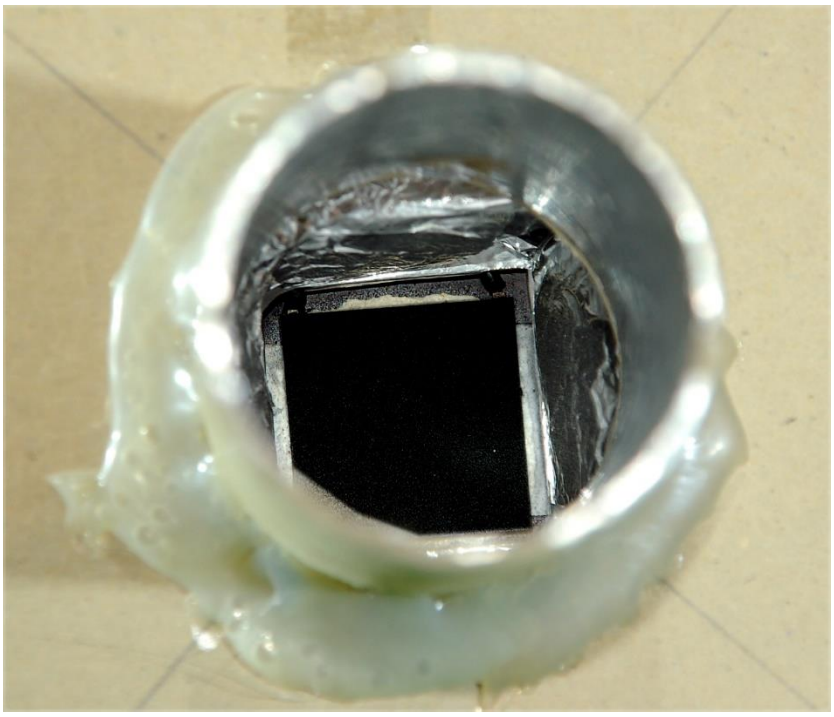


Abbildung 15: Strahlungssensor_Detail

Jetzt aber zum Programm. Für die Watt- und Jouleanzeige verwenden wir einen großen Zeichensatz mit 24 Punkt, in der Art wie er in der letzten Folge erstellt wurde. Wenn Sie selbst einen erstellen möchten, können Sie das Paket font2py.rar verwenden. Die Funktionen für die Steuerung habe ich dieses Mal allerdings in die Klasse CharSet im Modul charset.py gepackt. Der Zeichensatz ist geometer_24.py.

Ein Timerinterrupt sorgt alle 2 Sekunden für ein Messwertupdate. Innerhalb dieser Zeit wechseln sich jeweils die Watt- und die Jouleanzeige ab. In einer normalen Textzeile darunter kommt die Anzeige der Umgebungstemperatur, die durch einen (nackten) DS18B20 abgetastet wird. Die Schaltung ist also, bis auf das Fehlen des einen DS18B20, identisch mit der Schaltung zum Eichen des Peltierelements. Wie dort wird im Programm wattmeter.py eine Calibrierung angeboten, für die der Sensor am besten einige Zeit in ruhender Luft aufgestellt oder aufgehängt wird, ohne Kontakt der Sensorflächen mit anderen Gegenständen.

```
# wattmeter.py
# Author: J. Grzesina
# Rev: 1.0
# Stand: 2021-06-22
# ******************************************************
from machine import Pin, I2C, ADC, Timer
from time import sleep, ticks_ms,sleep_ms
from onewire import OneWire
from ds18x20 import DS18X20
from oled import OLED
from button import BUTTON8266, BUTTONS
import geometer_24 as zs
from charset import CharSet
import os,sys
```

```
# Pintranslator fuer ESP8266-Boards
# LUA-Pins     D0 D1 D2 D3 D4 D5 D6 D7 D8
# ESP8266 Pins 16  5  4  0  2 14 12 13 15
#                 SC SD

adc=ADC(0)

SCL=Pin(5)
SDA=Pin(4)
i2c=I2C(-1,SCL,SDA)
d=OLED(i2c,128,32)
d.clearAll()
d.setYoffset(4)

ds_pin = Pin(14)     # D5@esp8266
ds = DS18X20(OneWire(ds_pin))
chips = ds.scan() # Liste von bytearrays
numberOfChips = len(chips)
print('Found DS devices: ')
for chip in chips:
    print(chip)

t0,t1=0,0
def los_messen(tim):
    global timerFlag
    timerFlag=True

u=0
u0=0
U=""
def getUth(n=5):
    s=0
    for i in range(n):
        s+=adc.read()
    s=s//n
    u=(int(s/1023*3.0*10000))/10  # Spannung in mV
    return u

def testCalibration():
    calibrated=("ucal.txt" in os.listdir())
    print (calibrated)
    d.clearAll()
    d.writeAt("THERMOSPANNUNG",0,0,False)
    d.writeAt("KALIBRIEREN",0,1,False)
    d.writeAt(">>> KEYS A + B",0,2)
    act=k.waitForTouch(taste,3)
    print(act)
    if calibrated and act==None:
        f=open("ucal.txt","r")
        u0=float(f.readline())
        t0=0
```

```
        f.close()
        f=None
        d.clearAll()
        d.writeAt("CALIBR. READ",0,0,False)
        d.writeAt("U0={} T={}".format(u0,t0),0,2)
        sleep(3)
    elif act==1:
        u0,t0=calibrateUth()
    else:
        u0=1543
        t0=9999
        d.clearAll()
        d.writeAt("CALIBR. SET",0,0,False)
        d.writeAt("U0={} T={}".format(u0,t0),0,2)
        sleep(3)
    return (u0,t0)


def calibrateUth():
    d.clearAll()
    d.writeAt("PRESS KEY B",0,0,False)
    d.writeAt("WAIT FOR",0,1,False)
    d.writeAt("RELEASE MSG.",0,2)
    sleep(3)
    d.clearAll()
    d.writeAt("CALIBRATING",0,0)
    ds.convert_temp()
    u0=getUth(20)
    sleep(3)
    t0 = (int((ds.read_temp(chips[0]))*100))/100
    print(t0,u0)
    d.clearAll()
    d.writeAt("CALIBR. DONE",0,0,False)
    d.writeAt("RELEASE KEYS!",0,1)
    sleep(3)
    d.clearAll()
    d.writeAt("U0={} mV".format(u0),0,0,False)
    d.writeAt("T={} *C".format(t0),0,2)
    sleep(3)
    f=open("ucal.txt","w")
    f.write(str(u0)+"\n")
    f.close()
    f=None
    return (u0,t0)

taste = BUTTON8266(0,invert=True) # D3@esp8266
k=BUTTONS()

cs=CharSet(zs, d)

d.clearAll()
d.writeAt("THERMISCHES",0,0,False)
```

```
d.writeAt("WATT-METER",0,1,False)
d.writeAt("JOULE-METER",0,2)
sleep(3)

U0,T=testCalibration()
print("T = {}°C".format(T))
print("U0 = {}V".format(U0))

sleep(2)
T=Timer(0)
timerFlag= True
intervall=2000
T.init(mode=Timer.PERIODIC,period=intervall, \
       callback=los_messen)
energy=0
while taste.tpin.value() == 1:
    if timerFlag==True:
        timerFlag=False
        ds.convert_temp()
        s=getUth(20)
        u=s-U0  # Spannung in mV
        power=int((u*8.8/1000+0.05)*100)/100
        powerString=("{:.2f}".format(power)).replace(".",",")
        energy=energy+int((power*intervall/1000+0.05)*\
                          100)/100
        energyString=("{:.2f}".format\
                        (energy)).replace(".",",")
        sleep_ms(1000)
        t0 = (int(((ds.read_temp(chips[0]))+0.05)*10))/10
        f="{:.1f}"
        t0 = f.format(t0)
        t0String = t0.replace(".",",")
        d.clearAll(False)
        cs.putValue(powerString,"W",0,0,False)
        d.writeAt(" T= {} *C".format(t0String),0,2)
        sleep(1)
        d.clearAll(False)
        cs.putValue(energyString,"J",0,0,False)
        d.writeAt(" T= {} *C".format(t0String),0,2)
T.deinit()
d.clearAll()
d.writeAt("PROG DONE",0,0)
```

Mit Hilfe dieses Messgeräts sind wir gut gerüstet, wenn wir in der nächsten Folge unsere Kältemaschine mit den thermoelektrischen Wandlern bauen. Bei genügendem Abtransport der Energie auf der Warmseite der Maschine, kann man (theoretisch) laut Datenblatt eine Temperaturabsenkung auf der Kaltseite um ca. 60K erreichen. Wie es damit wirklich bestellt ist, werden wir dann testen. So long, viel Vergnügen beim Basteln und Messen.