

Stepupwandler für solares Batterieladen

Von jean-claude.feltes@education.lu

1. Zweck

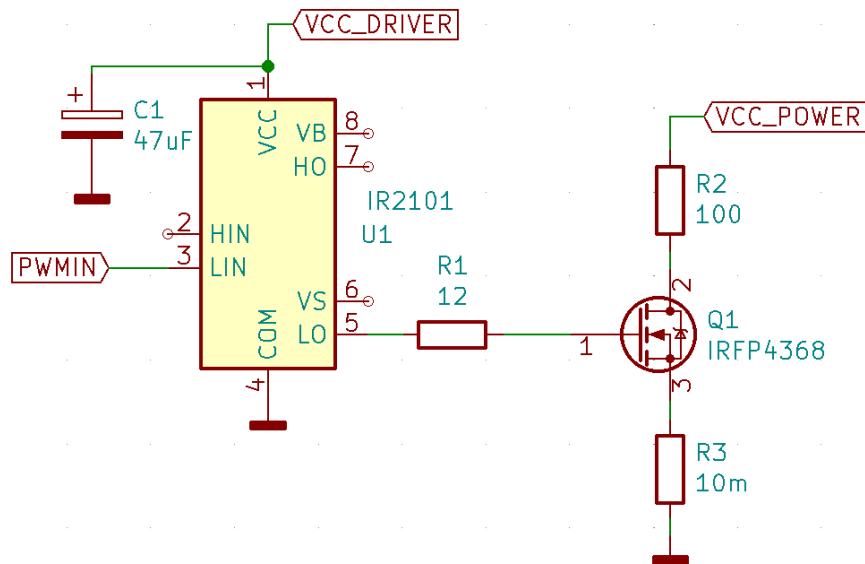
Der Wandler soll zwischen Solarpanel und eine 48V-LiIon-Batterie geschaltet werden, so dass diese mit Solarenergie geladen wird. Wegen der potentiellen Beschattung sind die Solarpanel nur zu jeweils zweit in Reihe geschaltet, diese speisen dann einen Wandler, oder wenn es sinnvoller erscheint, wird für jede Gruppe von 2 Paneln ein eigener Wandler benutzt.

Dieses Projekt ist experimentell, sein Ziel ist ein dreifaches: am Schluss soll eine brauchbare Schaltung herauskommen, daneben vieles über Leistungselektronik gelernt und dabei noch Elektronikschröpfen recycelt werden.

2. MOSFET + Driver Test

Für einen guten Wirkungsgrad ist es wichtig, Schalttransistoren mit niedrigem R_{DS} zu verwenden. Außerdem ist ein Gatedriver nötig um die erforderlichen Umladeströme für die Gatekapazität zu liefern. Als PWM-Signalquelle diente ein eigens für diesen Zweck gebauter Generator:

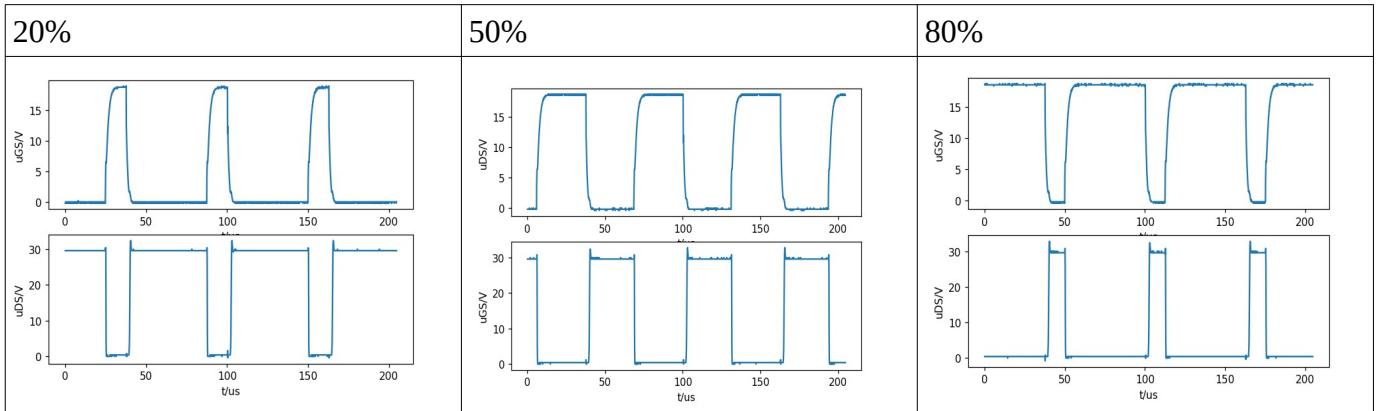
<https://github.com/jean-claudeF/PWM-generator>



VCCPOWER = 30V

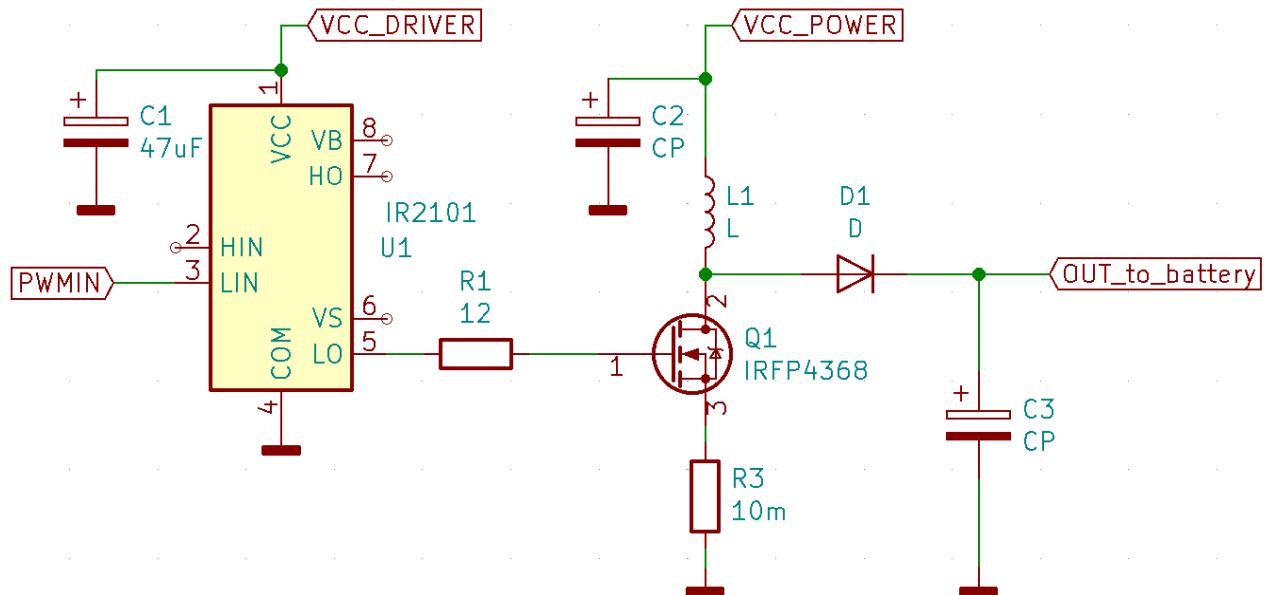
VCC_DRIVER = 18V (max. 20V, min. 10V)

$f = 16\text{kHz}$



Die Signale am Gate sehen schlechter aus als erwartet, der MOSFET scheint aber (laut u_{DS}) gut zu schalten.

3. Test Stepup converter



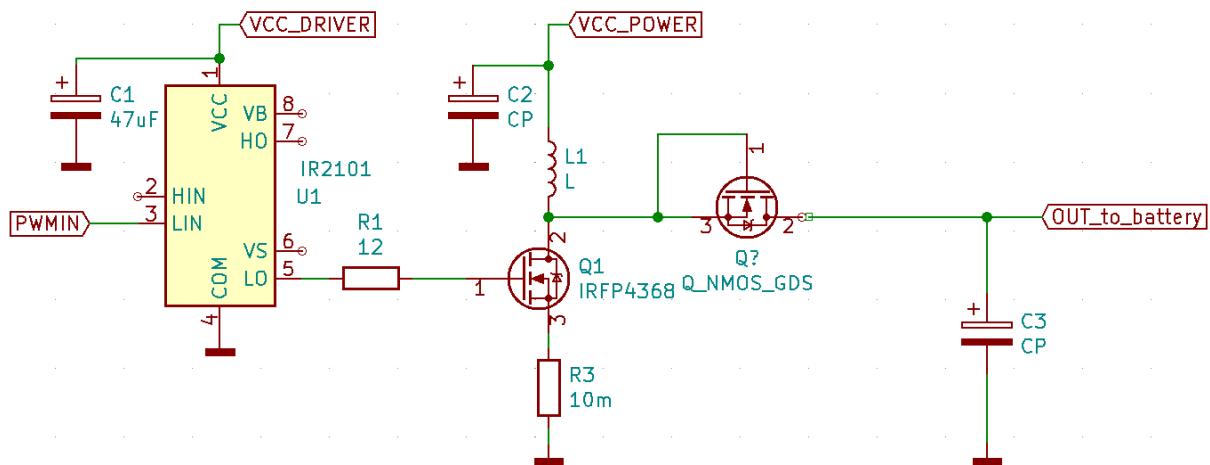
$$L = 300\mu\text{H}$$

$$C_2 = 330\mu\text{F}$$

$$C_3 = 470\mu\text{F}$$

D = STTH3010PI (1000V / 30A, ultra fast) ist eine schlechte Wahl. Sie wird heiss da sie eine hohe Durchlass-Spannung hat.

4. Test Stepup mit MOSFET als Diode an Lastwiderstand



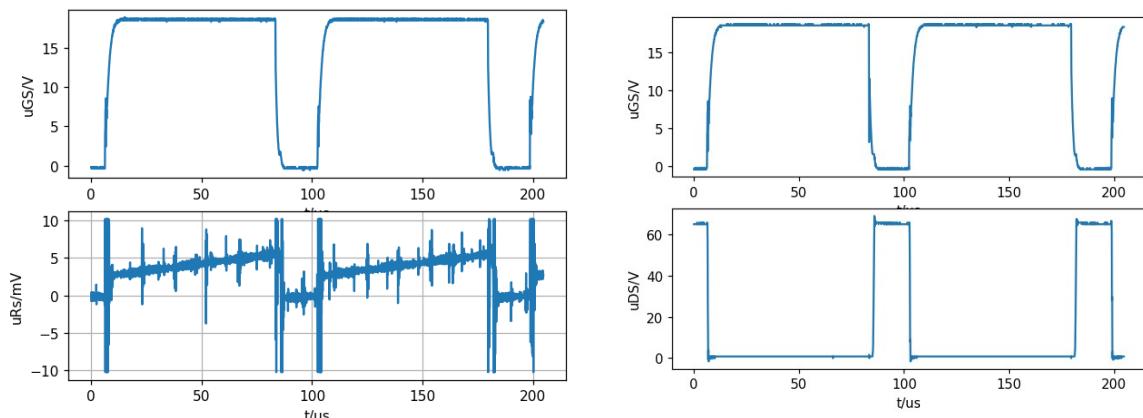
Beispielmessung:

VCC_Power = 12V

f = 10.4kHz, PWM = 80%

ICC = 4.25A, PCC = 51.3W

UOUT = 64.75V an RL = 100 Ohm.

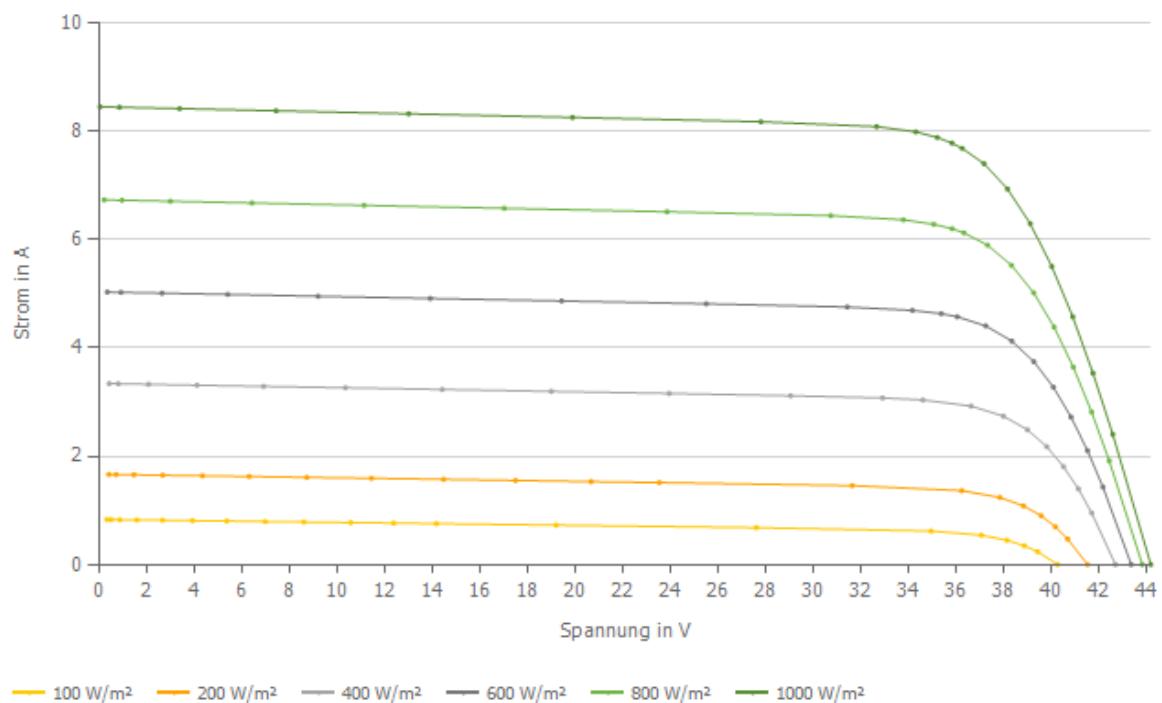


Die beiden Transistoren bleiben ziemlich kühl, die Spule wird leicht warm.

Eingang HIN des Drivers sollte auf Masse gelegt werden!

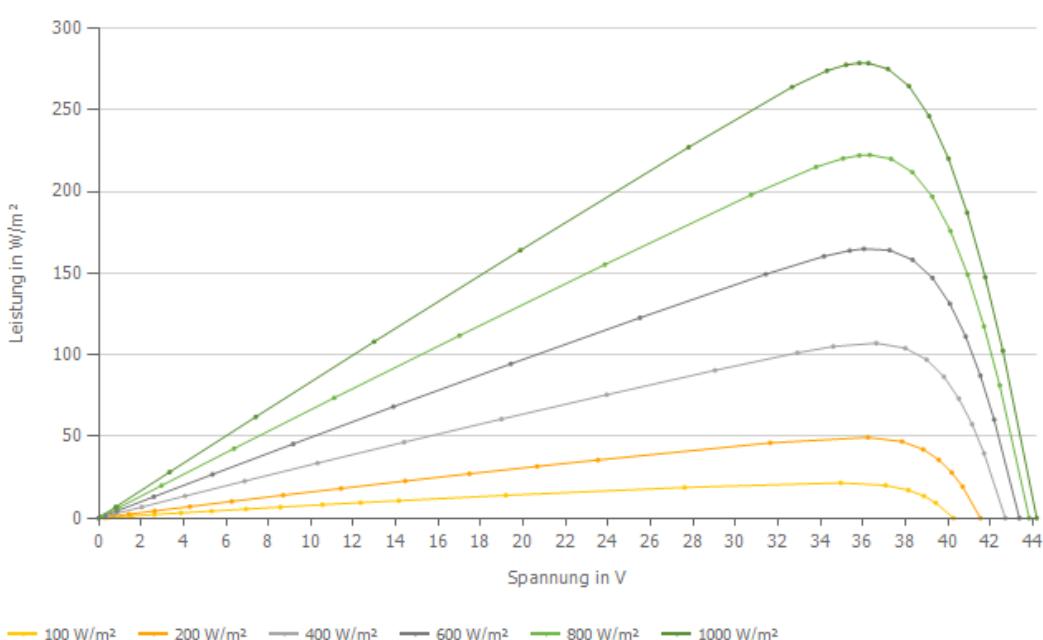
5. Kennlinien von Solarmodulen

Beispiel:



<https://help.valentin-software.com/pvsol/de/berechnungsgrundlagen/pv-module/arbeitspunkte-und-kennlinien/>

Maximum Power Point:



Für alle Einstrahlungsfälle liegt der MPP bei hoher Spannung 36-37V, nicht weit vom Leerlauffall entfernt. Dies bedeutet dass der PWM-Wert nur wenig variiert werden muss.

Daten unserer Panel:

solartronics 130W
Maximum power voltage = 18.13V
Ishortcircuit = 7.63A
Leerlaufspannung U0 = 22.25V

Davon 2 in Reihe geschaltet.

6. Test als Batterieladmodul

Als Entkopplung von der Batterie wird zur Sicherheit eine Diode P600B verwendet.

(Diese wird später weggelassen, da Q2 schon diese Aufgabe übernimmt)

Kleiner Praxistest.

Die Spannung VCC_POWER kommt von 2 in Reihe geschalteten 13W Solarmodulen, der PWM-Wert wird manuell nachgeregelt.

Der PWM-Wert für den MPP liegt um die 13-50%.

Batterie-Ladestrom ca. 0.5 – 2A bei bewölktem Himmel.

Solarspannung ca. 37 – 38V im MPP.

Bei etwas mehr Sonne: MPP bei PWM von 25%

7A, 36V, → 252W (Solarpanel)

4.9A, 44.9V → 220W (Batterie)

→ Wirkungsgrad 87%

7. Erfassung der Sonneneinstrahlung

Ohne diese gibt es keine verlässlichen Vergleiche. Es wird eine kleine Solarzelle von einem Lichtverschmutzungs-LED-Stab (Solar-LED für die Gartenbeleuchtung) benutzt (ca. 25cm²).

Erster Versuch: Messung des Kurzschluss-Stroms mit mA-Meter. Funktioniert, aber ungenau wegen recht hohem Innenwiderstand des Messgeräts.

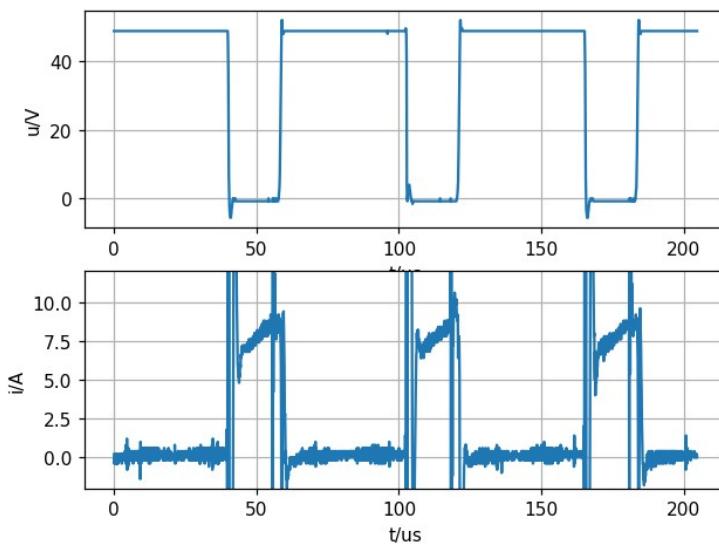
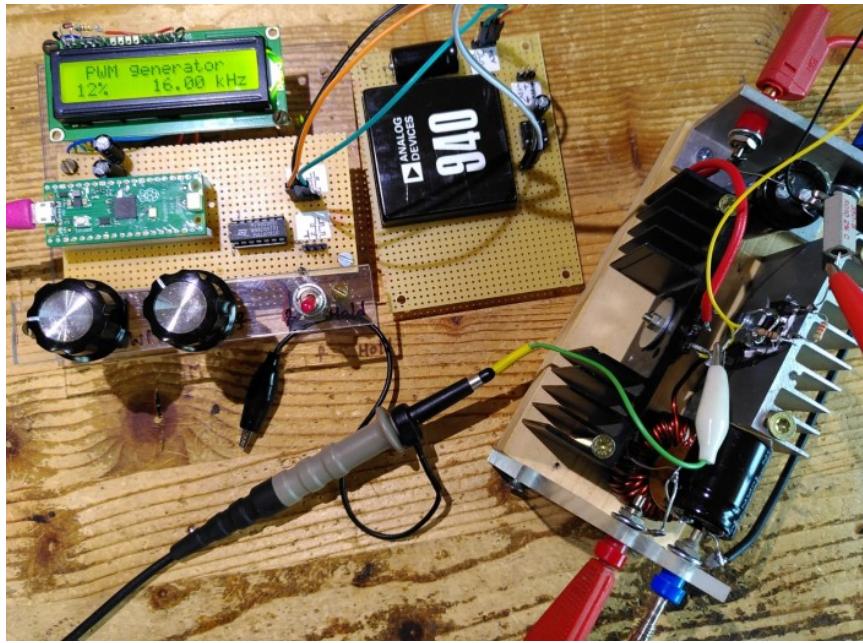
Zweiter Versuch: Nutzung eines vorhandenen I-U-Konverters mit OPV und Messung mit V-Meter. Scheint recht gut zu funktionieren.

8. Erzeugung der Gatedriver-Betriebsspannung aus den +5V

16.4.2023

Hier wurde ein fertiges (altes) Modul von Analog Devices benutzt : der 940 macht aus 5V eine +15V (VCC_DRIVER) und eine -15V Betriebsspannung (hier nicht benutzt).

Dieses Modul wurde zusammen mit einem Pufferelko und dem Gatedriver IR2101 auf einer separaten Platine untergebracht.



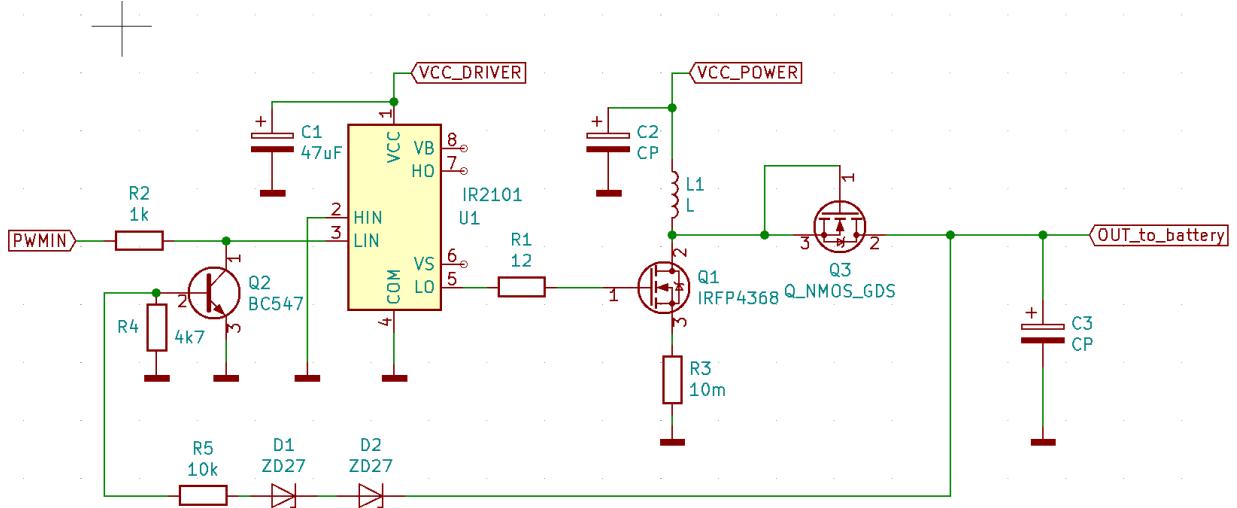
Oben: uDS, unten iDS

9. Problem ohne Belastung am Ausgang

Wenn der Akku keinen Strom aufnimmt (z.B. Abschaltung da voll), wird der Wandler sehr heiss und die Ausgangsspannung steigt stark an.

→ in diesem Fall muss abgeschaltet werden!

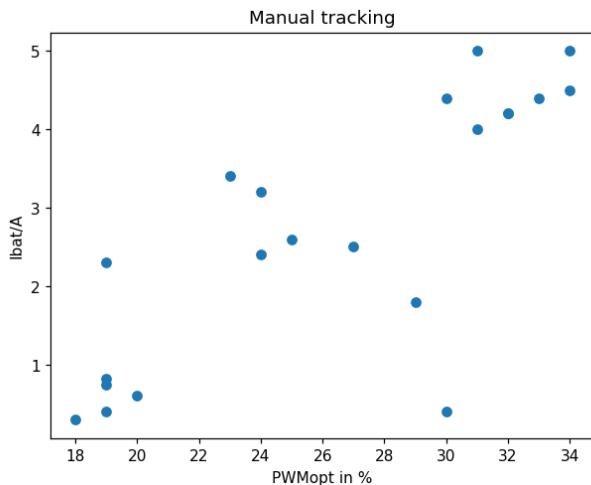
Diese Lösung scheint zu funktionieren:



Im Leerlauf steigt die Spannung am Ausgang auf über 54V, die Z-Dioden werden leitend, Q2 ebenfalls so dass das PWM-Signal kurzgeschlossen wird und der Wandler inaktiv.

Diese Lösung ist bei geringer Sonneneinstrahlung getestet worden. Der Härtetest bei voller Einstrahlung steht noch aus.

10. Manuelles Tracking



Das Diagramm zeigt den Zusammenhang zwischen manuell eingestelltem optimalen PWM-Wert und Batteriestrom (Maximaler Strom entspricht in etwa auch maximaler Leistung da die Batteriespannung relativ konstant ist).

Wie erwartet muss der Tastgrad bei höheren Strömen (durch stärkere Sonneneinstrahlung) für den MPP erhöht werden. Lediglich 2 Punkte entsprechen dem nicht. Messfehler?

Listing:

```
import matplotlib.pyplot as plt

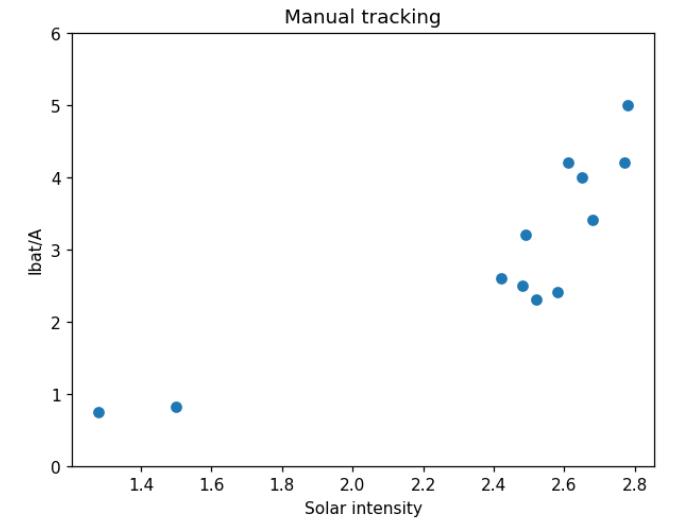
x = [23, 32, 27, 25, 24, 31, 19, 19, 19, 24, 34, 29, 18, 30, 19, 20, 33, 30, 31, 32, 34]
y = [3.4, 4.2, 2.5, 2.6, 3.2, 4, 2.3, 0.75, 0.82, 2.4, 4.5, 1.8, 0.3, 0.4, 0.4, 0.6, 4.4, 4.4, 5, 4.2, 5]
```

```

plt.title("Manual tracking")
plt.plot(x, y, 'o')
plt.xlabel("PWMopt in %")
plt.ylabel("Ibat/A")
plt.show()

```

Im nächsten Diagramm ist die Abhängigkeit des MPP-Batteriestroms von der Sonneneinstrahlung dargestellt:



11. Erweiterung der Messtechnik

19.4.2023

Für weitere Experimente mit manuellem MPP-Tracking wäre es wünschenswert folgendes zu erfassen (und die Werte gleich in eine Datei schreiben zu lassen, serielle Übertragung):

- Strom Batterie + Panel
- Spannung Batterie + Panel (neue I2C-Wandler!)
- Sonneneinstrahlung relativ
- Eingestellter PWM-Wert für MPP
- daraus berechnet Pzu, Pab, Wirkungsgrad

Die Steuerung soll weitgehend über den PC erfolgen, PWM-Wert manuell Pot oder Schieber am PC, später automatisch.

18.5.2023

Für die Erfassung von Spannung, Strom und Leistung wurde ein Breakoutboard mit **INA226** ausprobiert. Einen funktionierenden Micropython-Driver gab es, allerdings mit einem Bug den ich nicht sofort bemerkte. Das Datenblatt mit seinem Kapitel über Programmierung war eher hinderlich als nützlich, die Vorgehensweise erscheint mir auch heute noch extrem unpädagogisch und unlogisch. Nachdem ich dieses

Kapitel ignorierte und mich auf das Blockschaltbild fokussierte, gelang die Programmierung des Treibers problemlos. Als Grundgerüst wurde der Treiber von Chris Becker genommen.

Hier ist das Ergebnis:

<https://github.com/jean-claudeF/INA226>

Die bequeme Variante nutzt nur das Bus voltage register und das Shunt voltage register, und sollte eigentlich auch noch genauere Ergebnisse liefern:

```
import ina226_jcf as ina226
from machine import Pin, I2C
import time

i2c = I2C(0,scl=Pin(9), sda=Pin(8), freq=100000)
ina = ina226.INA226(i2c, 0x40, Rs = 0.002, voltfactor = 2 )

i=0
while True:

    v, i, p = ina.get_VIP()
    V = '%2.3f' % v
    I = '%2.3f' % i
    P = '%2.3f' % p
    print(i, '\t', V, '\t', I, '\t', P)
```

12. Intermezzo: Spulen für einen zweiten Leistungsteil

Es wurden aus verschiedenen, meist PC-Netzteilen Spulen ausgeschlachtet, neu gewickelt und vermessen. Dabei leistete das Spulen-Messgerät wertvolle Hilfe:

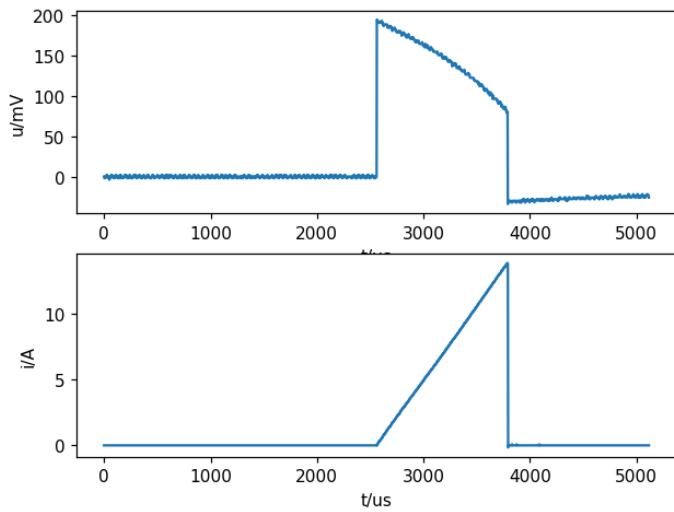
http://staff.ltam.lu/feljc/electronics/messtechnik/messung_ferritspulen.pdf

Der Impulsgenerator wurde noch leicht umgebaut, um einen grösseren Impulsdauer-Bereich zu ermöglichen.



Nicht alle Spulenkerne erfüllten die Erwartungen. Einige hatten auch bei bescheidenen Windungszahlen recht niedrige Sättigungsströme.

850uH-Spule (unten links im Bild)

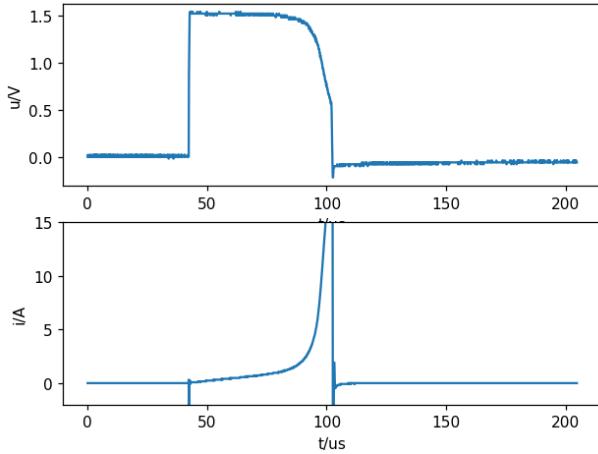


Unten: Strom, oben: induzierte Spannung in 1 Windung

Die zusätzliche Spannungsmessung in 1 Windung wurde vorgenommen, um eine zusätzliche Information über die Sättigung zu bekommen. Das Problem ist nämlich, dass sich eine durch den Widerstand nach unten gekrümmte Kurve und eine durch die Sättigung nach oben gekrümmte Kurve teilweise kompensieren können, so dass die Kurve linear aussieht, obwohl die Spule schon in der Sättigung ist. (Allerdings müsste sich die exponentielle Widerstandskrümmung schon zu Beginn der Kurve zeigen, während die Sättigungskrümmung erst bei hohen Strömen auftritt.)

Im Idealfall sollte die induzierte Spannung impulsförmig sein, im Sättigungsfall müsste sie stark abnehmen.

Ungeeignete 360uH Spule, $I_s = 1A$ (nicht im Bild):



Ab dem Sättigungsstrom bricht die induzierte Spannung stark ein.

13. Intermezzo: Schatten-Versuch

Wie wirkt sich eine partielle Abschattung der Panel aus?

Um dies zu untersuchen habe ich folgendes Experiment gemacht:

1. Zunächst wurde bei voller Sonneneinstrahlung der MPP manuell eingestellt, als Ergebnis floss ein Batterie-Ladestrom von ca. 4A bei einem PWM-Wert von 33%.
2. Dann wurde etwa 1/3 der Fläche eines der beiden in Reihe geschalteten Solarpanel abgedeckt (Also etwa 1/6 der Gesamtfläche). Das Ergebnis war dramatisch. Der Strom ging bei unverändertem PWM-Wert praktisch auf null zurück.
3. Nun wurde der MPP wieder manuell nachgestellt, mit dem Ergebnis dass sich der Strom wieder auf 1.8A bringen liess, nun bei einem PWM von 70%.
Das MPP Tracking lohnt sich also gerade bei Abschattung eines Teils.
4. Beim gleichen PWM-Wert wurde die Abdeckung entfernt. Das Ergebnis war schlechter als zu Beginn, der Strom betrug nur 2A. Mit manuellem Tracking auf einen PWM-Wert von 33% wie zu Beginn ergab sich wieder der Strom von 4A.

Fazit:

Ein MPP-Tracking kann erhebliche Leistungsgewinne bringen, vor allem wenn ein Teil der Zellen im Schatten liegt.

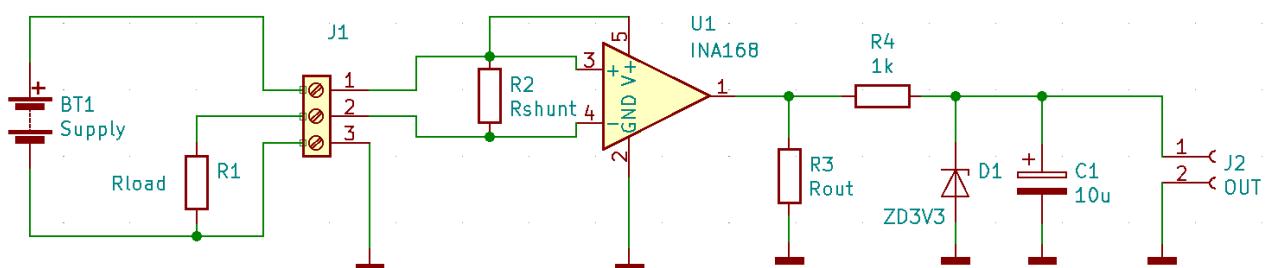
14. "Schönere" Strom-Messung

1.6.2023

Der Wunsch nach einer High Side Messung (Warum? Wegen der gemeinsamen Masse für alle Baugruppen) erinnerte mich an Versuche mit dem INA168. Dieser kann an bis zu 60V betrieben werden, er liefert ein analoges Ausgangssignal:

http://staff.ltam.lu/feljc/electronics/messtechnik/INA168_HighSide_Stromsensor.pdf

Erster Test:



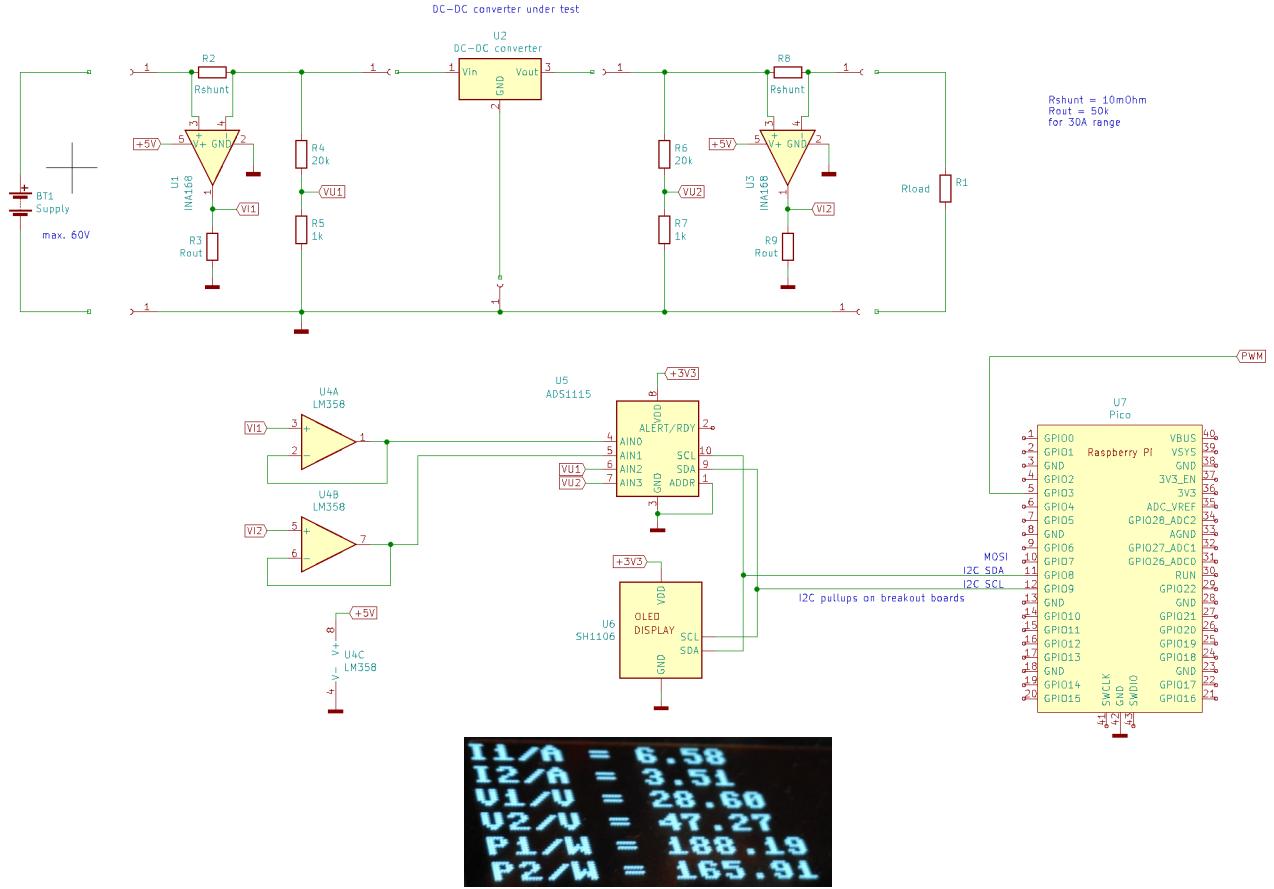
$$R_{shunt} = 10m\Omega$$

$$R_{out} = 50k\Omega$$

Da das Ausgangssignal auf eine ADS1115 – ADC geführt werden soll, wird es sicherheitshalber von D1 begrenzt (Es können schon mal Störimpulse mit zerstörerischer Spannung am Ausgang erscheinen.) Außerdem sorgen R4 und C1 für eine Tiefpass-Filterung.

15. Ein Leistungs-Vierpol-Messgerät

Für genauere Untersuchungen des Stepupwandlers (oder eines anderen Leistungs-Vierpols) wäre es wünschenswert, alle Spannungen, Ströme und Leistungen zu messen. Das geht mit dieser Schaltung:

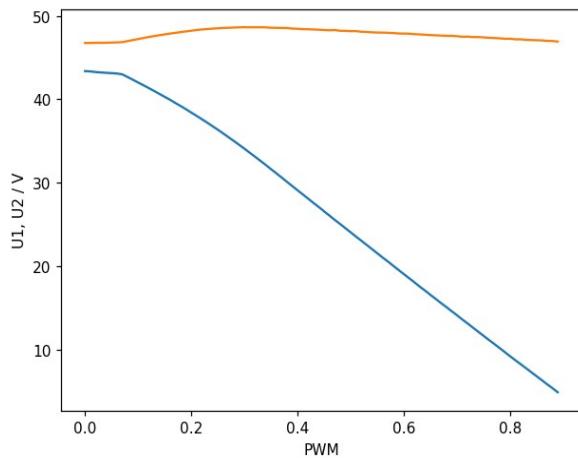
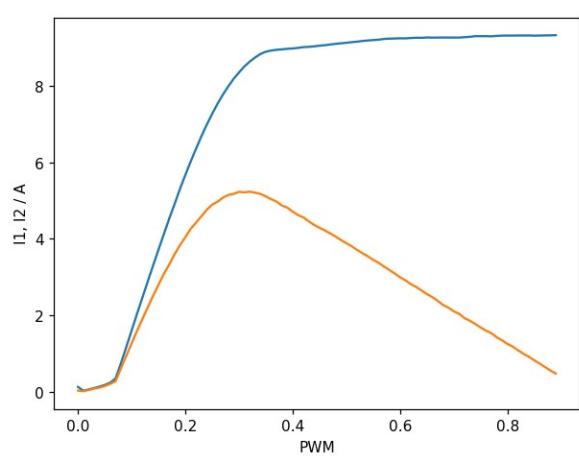
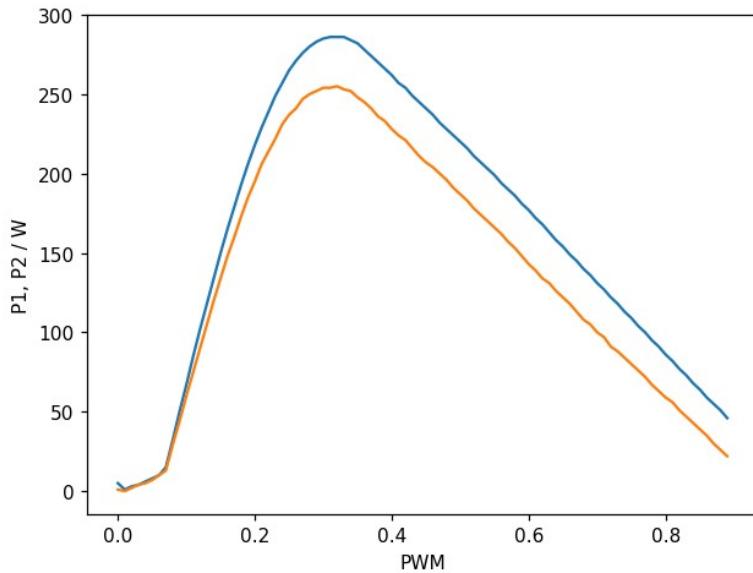


<https://github.com/jean-claudeF/MeasurePowerQuadrupole>

17.7.2023

16. MPP-Kurven

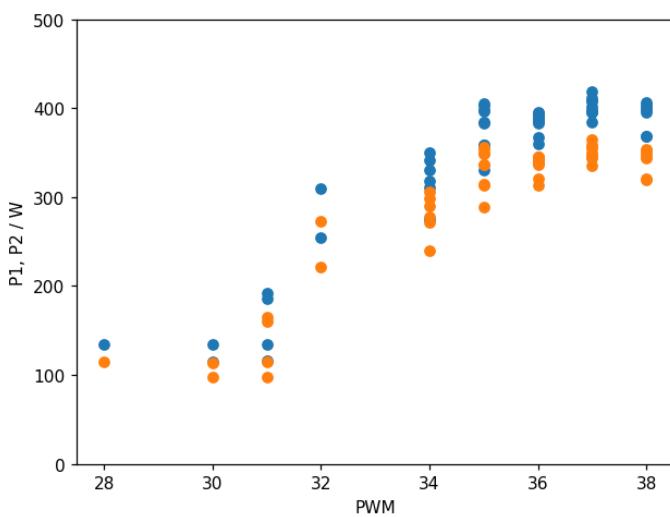
Die folgenden Kurven wurden mit dem INA168-Stromsensor aufgenommen. Dabei wurde eine Parallelschaltung von 2 x 2 in Reihe geschalteten Modulen verwendet.



1 = Eingangsgrößen

2 = Ausgangsgrößen des Stepup – Wandlers.

Wie ist der Zusammenhang zwischen optimalem PWM-Wert und Leistung?



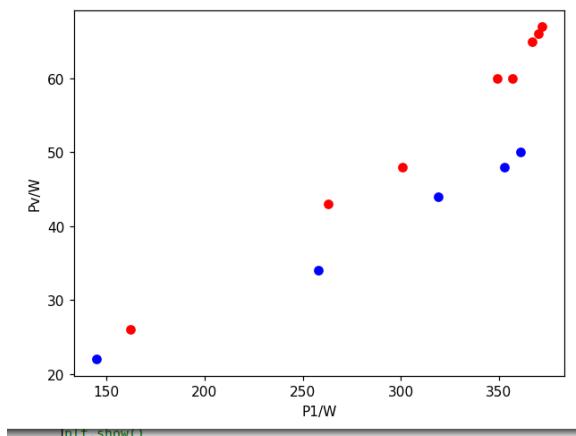
17. Spulenvergleich

Die MPP-Kurven wurden für 2 verschiedene Spulen aufgenommen:

Spule 1: Ringkern 300uH (blau)

Spule 2: Rechteckkern (trafoartig) 700uH, 89 Windungen (rot)

Aus den aufgenommenen Werten wurde $P_{vmax} = f(P_{1max})$ ermittelt:



Spule 2 schneidet schlechter ab.

18. MPP Tracking mit dem Leistungs-Vierpol-Messgerät

Für eine einfachere Software-Realisierung wurde das Gerät in einem Objekt abgebildet.

https://github.com/jean-claudeF/MeasurePowerQuadrupole/blob/main/Micropython/mpptrack_01.py

Das Hauptprogramm ist damit recht einfach:

```
# Define object with or without connected OLED:  
m4p = Measure4poleadc, pwmgen, oled = oled)  
#m4p = Measure4poleadc, pwmgen, oled = None)  
  
m4p.set_calibration(k0, k1, k2, k3, offset0, offset1)  
m4p.set_pwm(0.3)  
  
# Track MPP, set PWM accordingly in regular intervals  
# Display values  
i = 0  
while True:  
    if i % 10 == 0:  
        if oled:  
            oled.print("MPP tracking")  
        m4p.mpp_track()  
  
        ##i1, i2, v1, v2, p1, p2, eta = m4p.measure()  
        m4p.measure()  
        m4p.print_values()  
        m4p.print_oled()  
        i += 1  
        time.sleep(1)
```

Zunächst wird ein Objekt Measure4pole definiert. Bei der Definition sind adc, pwmgen und oled zu übergeben (die natürlich vorher definiert werden müssen. Dieser Teil des Programms ist hier nicht dargestellt, zu finden aber hier: <https://github.com/jean-claudeF/MeasurePowerQuadrupole>).

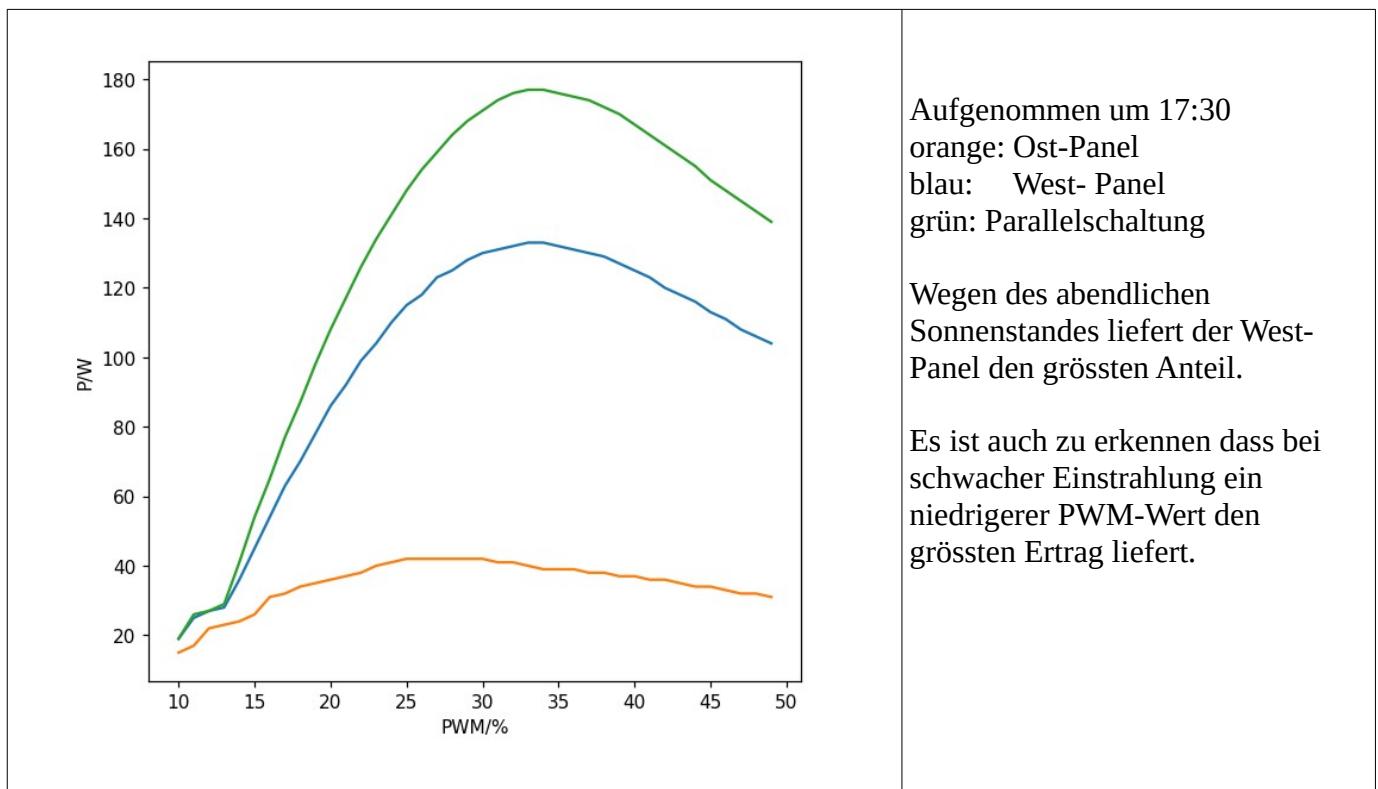
Es ist auch oled = None möglich, wenn kein OLED benutzt wird.

Dann werden die Kalibrierfaktoren gesetzt.

Im Beispiel erfolgt alle 10s ein Tracking, automatisch wird der beste PWM-Wert eingestellt, danach werden die Werte jede Sekunde einmal abgefragt und angezeigt.

Mit diesem Gerät können schnell Kurven bei verschiedener Sonneneinstrahlung, oder bei einer Zusammenschaltung von mehreren Panels aufgenommen werden.

Beispiel:

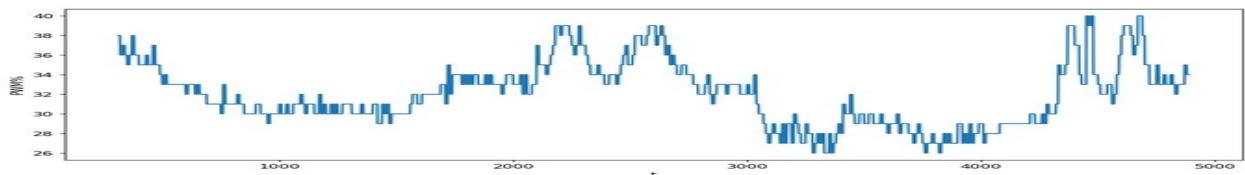


19. Messungen

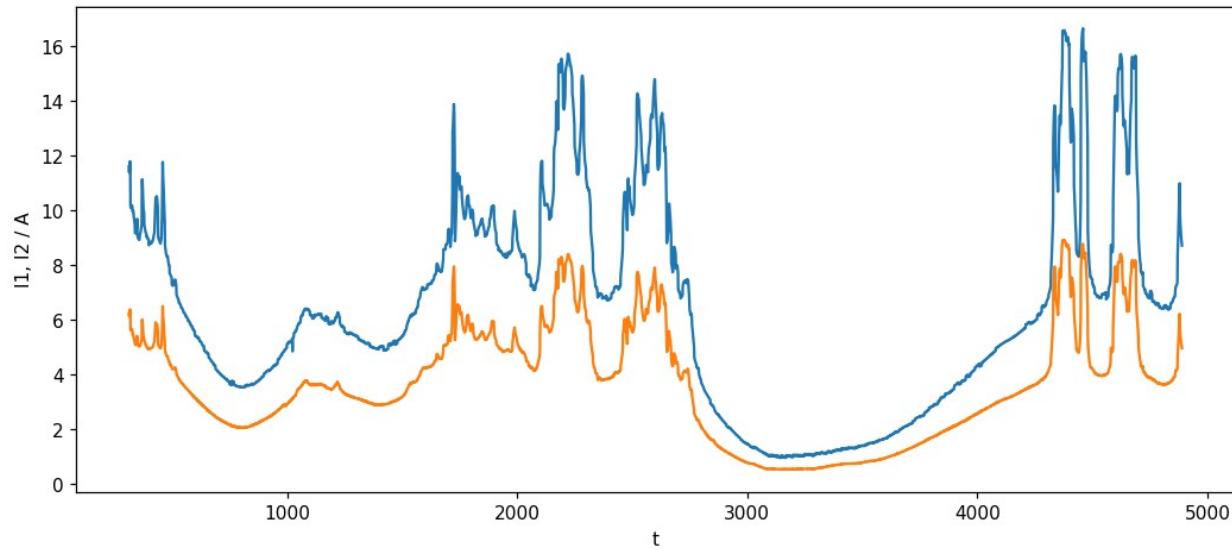
Die folgenden Messungen wurden am 4.8.2023 gegen Mittag gemacht, einem überwiegend bewölkten Tag, mit einer 3x2 – Gruppe von Panels die unterschiedlich ausgerichtet waren (Nordost, Südwest, Südost, jeweils 2 Panel in Reihe, diese Gruppen dann über eine Diodenentkopplung parallel) .

1. welche PWM-Werte treten auf?

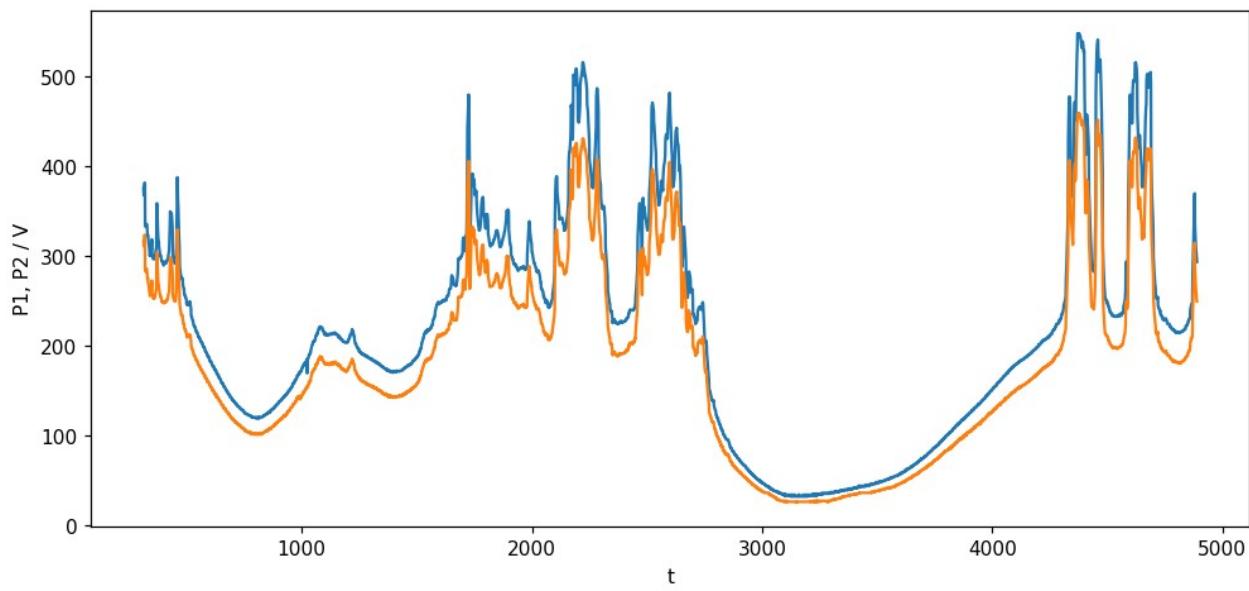
Alle Werte liegen zwischen 26 und 40%



2. In welchem Bereich liegen die Ströme? I1 bis ca. 16A, I2 bis ca. 8A

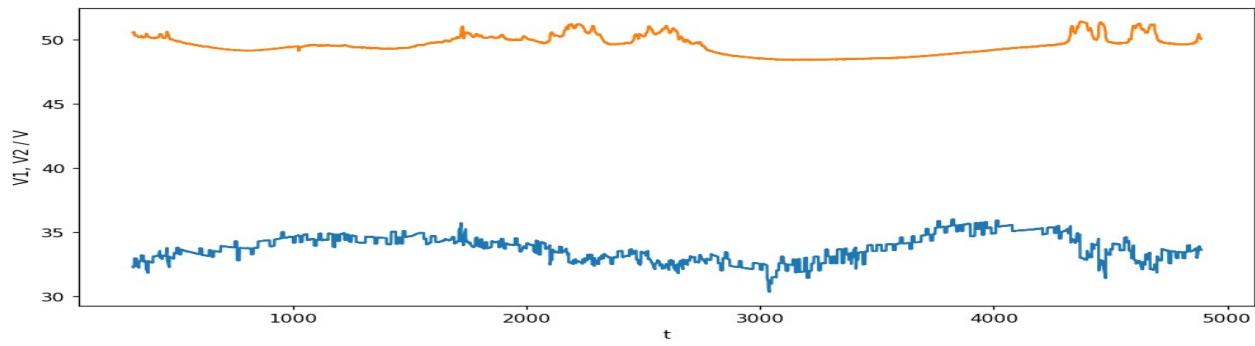


3. Welche Leistungen treten auf? P1max = ca. 570W, P2max = ca. 450W

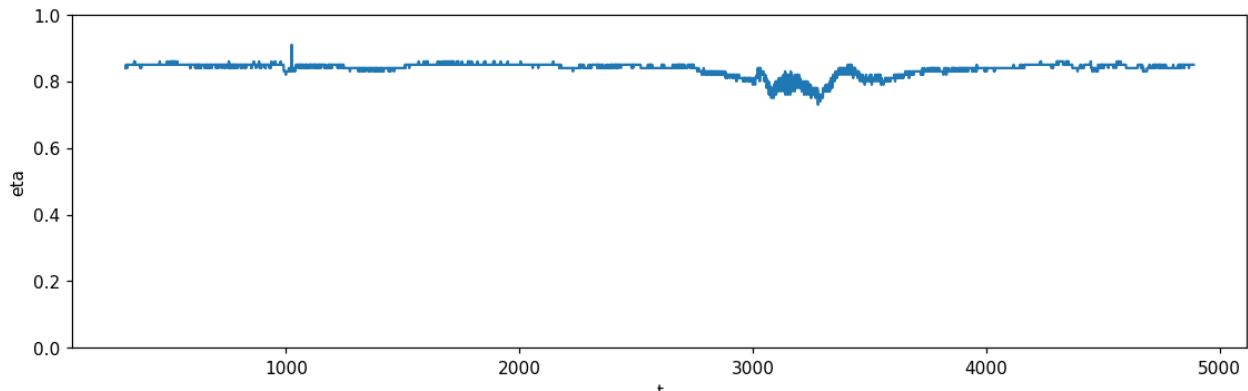


4. In welchem Bereich lag die Panel-Spannung? 32-37V für 2 Panel in Reihe.

Die Batteriespannung lag bei 48-49V, im Diagramm liegt die Kurve höher wegen des Spannungsabfalls am eigentlich zu dünnen Kabel. (Orange = Ausgangsspannung des Wandlers, es folgen einige Meter Kabel bis zur Batterie).



5. Was ist mit dem Wirkungsgrad des Wandlers?



Dieser liegt die meiste Zeit um 0.85.

Erstaunlicherweise liegt das Minimum nicht bei der höchsten Leistung, sondern gerade bei der kleinsten. Dies ist wohl darauf zurückzuführen dass dort die Genauigkeit der Leistungsmessung am kleinsten ist. Der Wirkungsgrad scheint also relativ unabhängig von der Leistung zu sein.

Sicher wäre es interessant, später einen Synchronwandler mit besseren Eigenschaften zu entwickeln. Aber erst soll diese Schaltung richtig fertig werden.

Für die Messungen wurde eine Spule 1140-221K-RC von BOURNS verwendet.

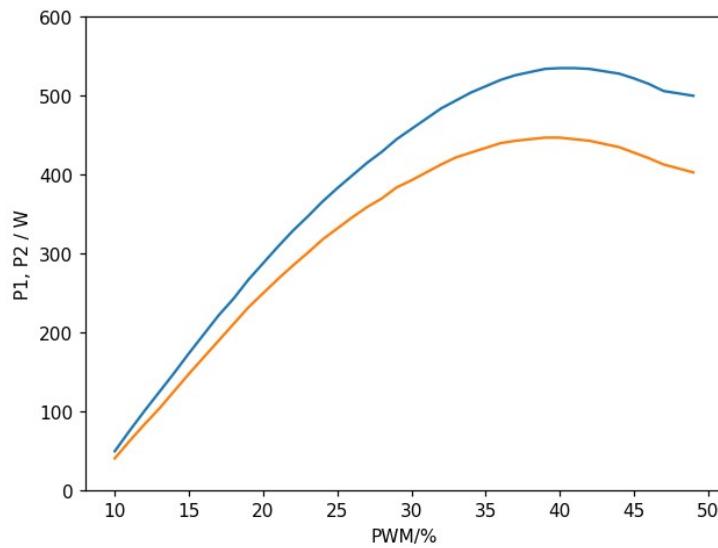
Dies da die Recycling-Spulen bei Leistungen bis ca. 200W gut funktionieren, darüber aber recht heiss werden

Da die MOSFETs und die Spule zeitweise ziemlich heiss wurden, wurde ein 12V Ventilator zur Kühlung benutzt, dies funktionierte recht gut. Eine Temperaturüberwachung sollte den Ventilator schalten.

Immer noch scheint die Spule das heisste und damit verlustreichste Teil zu sein.

Angestrebgt ist eine Leistung des Wandlers von 500-800W (für 6 Panel), dafür müsste eine "dickere" Spule verwendet werden.

Interessant ist noch eine Tracking-Kurve bei guter Sonneneinstrahlung:



20. Entwurf einer praktischen Schaltung

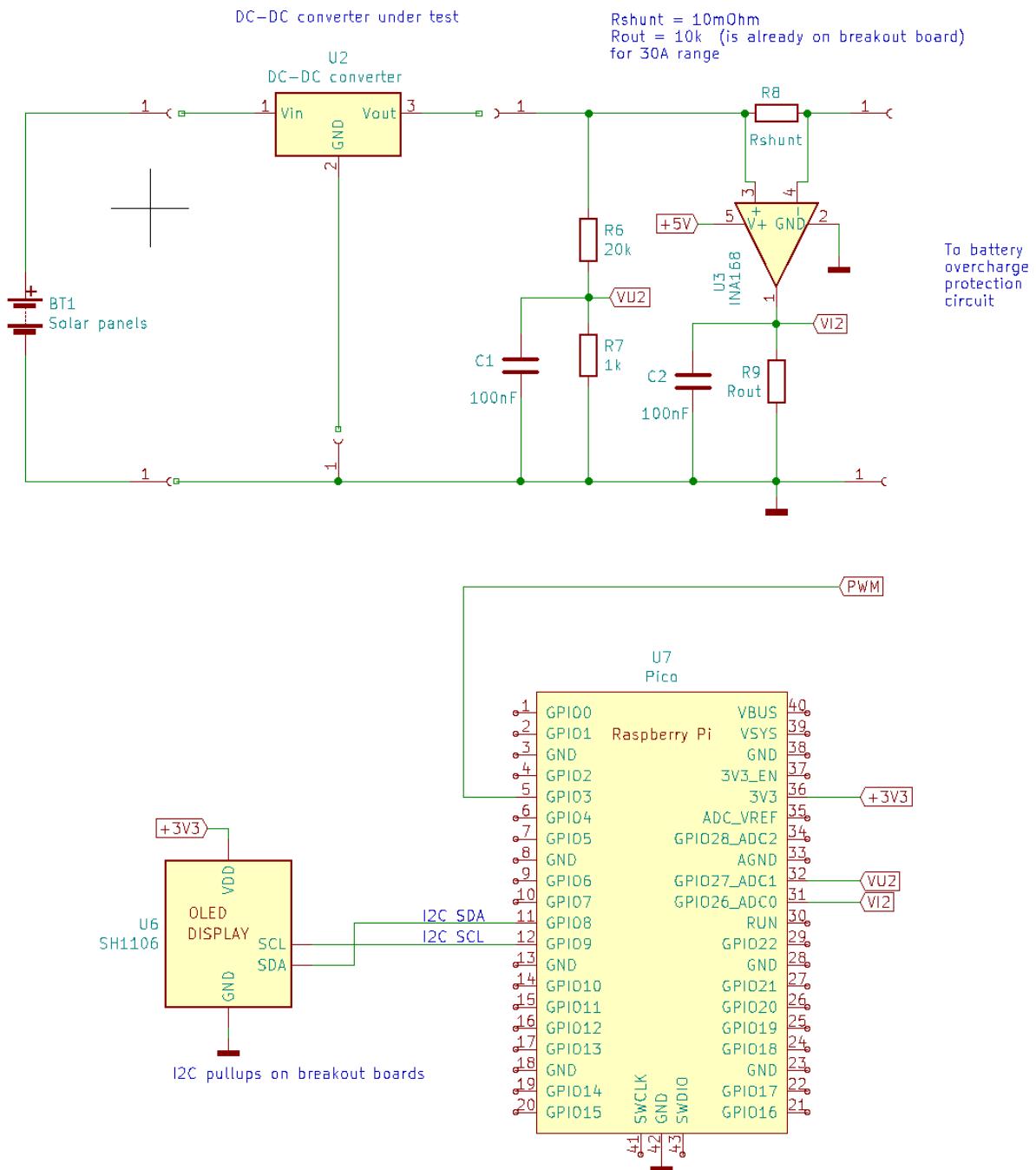
Die Messungen mit dem Measure4pole – Gerät lieferten wertvolle Erkenntnisse.

Für das praktische MPPT ist es aber nicht nötig alle Ein – und Ausgangsgrößen des Wandlers zu erfassen, es reicht, dies auf einer Seite zu tun (wahrscheinlich am besten am Ausgang, denn dies ist ja die Leistung die in die Batterie fliesst). Damit wird der Aufwand etwas kleiner.

Die Schaltung soll:

- U2, I2, P2 messen
- P2 periodisch tracken und den PWM-Wert einstellen so dass P2 maximal wird
- Die Temperatur überwachen und einen Ventilator zuschalten
- Die Leistung reduzieren wenn ein Grenzwert von P2 oder der Temperatur erreicht wird
- Alle Werte anzeigen (OLED) und über eine serielle Schnittstelle (separat, zusätzlich zu USB) ausgeben.

Hardware:



Zunächst wurde die Mess-Schaltung etwas abgespeckt um nur die Ausgangsspannung und den Ausgangsstrom (zur Batterie hin) zu messen. Da für das MPPT die Genauigkeit nicht unbedingt so gross sein muss, wurden die internen AD-Wandler des Pico benutzt.

Ein INA169 – Breakoutboard enthielt einen 100mΩ – Shunt, dieser wurde getauscht gegen einen höher belastbaren 10mΩ – Widerstand (bzw. dieser wurde parallel geschaltet, eine softwaremässige Korrektur des kleinen Fehlers ist ja leicht möglich). Der 10kΩ – Ausgangswiderstand wurde nelassen, damit ergibt sich ein günstiger Messbereich von 30A.

Auf einen Impedanzwandler wurde in dieser Version verzichtet, da Rout niedriger ist.

Dagegen wurde ein Filterkondensator eingebaut. Möglicherweise ist dies sogar günstiger als ein Impedanzwandler, bei dem es immer eine Offsetspannung gibt.

Achtung: der INA169 hat eine andere Steilheit $S = 1\text{mA/V}$ als der INA168 der in der vorigen Schaltung benutzt wurde.

Für $I = 1\text{A}$ ergibt sich mit den angegebenen Werten $U_s = R_s \cdot I = 10\text{m}\Omega \cdot 1\text{A} = 10\text{mV}$

Dies führt zu einer Spannung an R_{OUT} von $U_s \cdot S \cdot R_{OUT} = 10\text{mV} \cdot 1\text{mA/V} \cdot 10\text{k}\Omega = 100\text{mV}$

Da der Eingangsbereich des AD-Wandlers bis 3V reicht, könnten so 0...30A erfasst werden.

Ein praktischer Test zeigte eine Auflösung von ca. 10-20mA.

Die Filter-Grenzfrequenz ist $f_g = \frac{1}{2\pi R_{OUT} C_2} = 159\text{Hz}$.

Pico Software:

Die Messung von Spannung, Strom und Leistung wurde in einer Klasse `Measure_VIP` gekapselt, die leicht anzuwenden ist, im Prinzip mit

```
#meas = Measure_VIPadc0, adc1) # without oled
meas = Measure_VIPadc0, adc1, oled) # with oled

while True:
    v, i, p = meas.get_VIP()
    meas.print_VIP()
    meas.print_oled()
    time.sleep(0.5)
```

Diese wurde um eine Klasse MPPT erweitert, die von der `Measure_VIP` Klasse erbt.

Hiermit kann das Hauptprogramm übersichtlich strukturiert werden:

```
...
mppt = MPPTadc0, adc1, pwmgen, oled)
mppt.nbmean = 10
mppt.pwm_min = pwm_min
...
mppt.set_pwm(0)

i = 0
maxpower = 0 # max power over whole operation
-----
def main_loop():
    global i, maxpower
    while True:
        # MPP track every track_time:
        if (i % track_time) == 0:
            if pwm_manual.value():
                mppt.mpp_track()

        V, I, P = mppt.get_VIP()

        # Remember max power over whole operation
        if P > maxpower:
            maxpower = P
```

```

# print and display values:
print_my_values(i, V, I, P, mppt.pwmval)
mppt.print_oled(additional = '\tPmax = %2.0fW' % maxpower )

# PWM manual (potentiometer) or (normally) automatic
# if manual, must react faster
if pwm_manual.value() == 0:
    # PWM pot
    vpwm = adc2.read_u16() / 65535
    pwmgen.set_pwm(vpwm)

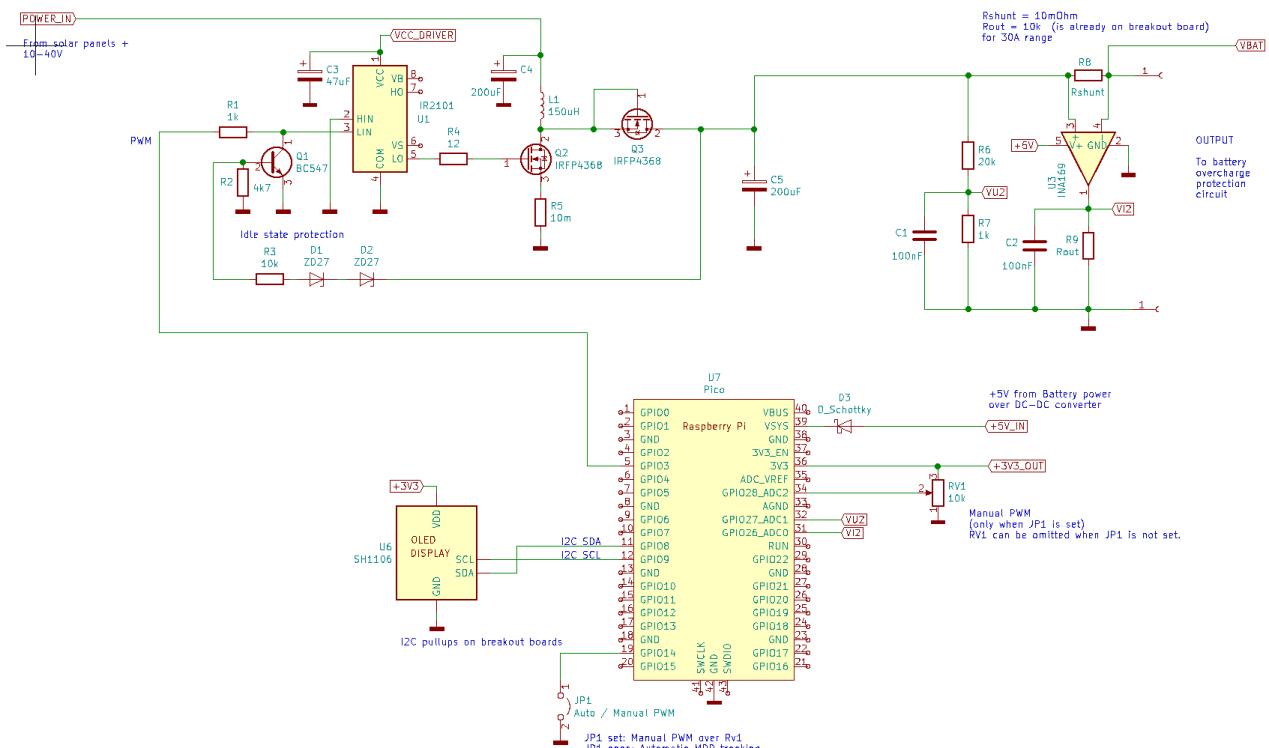
    time.sleep(0.05)
else:
    time.sleep(1)

i+= 1

main_loop()

```

21. Schaltung 20.September 2023

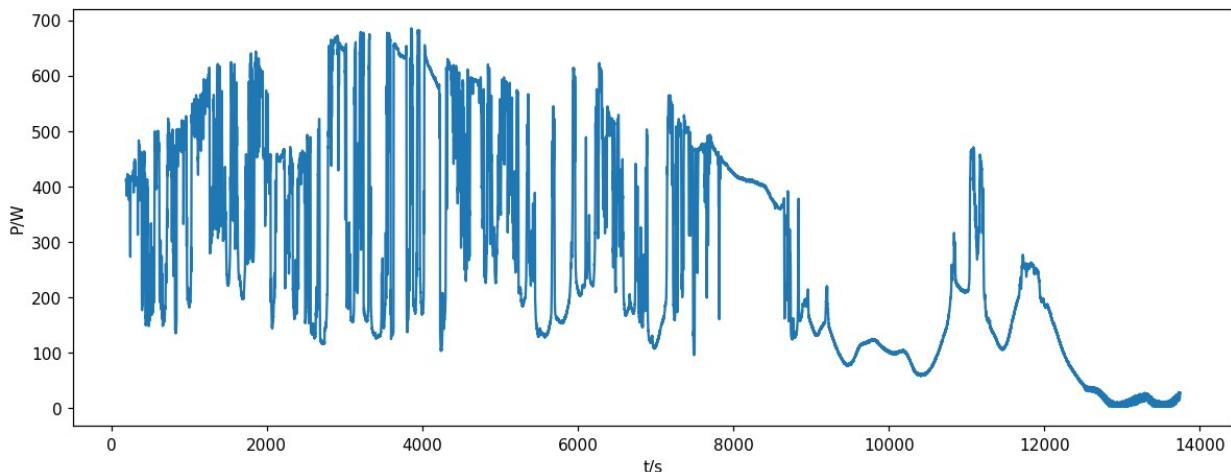


- Links: POWER_IN von 3-4 Gruppen von je 2 in Reihe geschalteten Paneln
- Rechts: Ausgang zur Batterie, über die analoge Schutzschaltung gegen Überladen
- Zu Testzwecken kann der PWM-Wert manuell eingestellt werden über RV1 (JP1 gesetzt)
Bei offenem JP1 erfolgt automatisches MPPT.
- Nicht eingezeichnet:
Die Stromversorgung des Pico (+5V_IN) erfolgt über ein kleines Stepdown-Netzteil aus der Batteriespannung.

- Die Stromversorgung des Gatedrivers (VCC_DRIVER) erfolgt nach wie vor über ein kleines Stepup-Netzteil aus der 5V-Versorgung.

22. Eine praktische Leistungsmessung

Eine Messung mit 4 Gruppen von je 2 Panelen zeigte, dass die Schaltung Leistungen bis über 600W abgeben kann.



Das Wetter war sehr wechselhaft, Bewölkung und Sonnenschein wechselten sich ab.

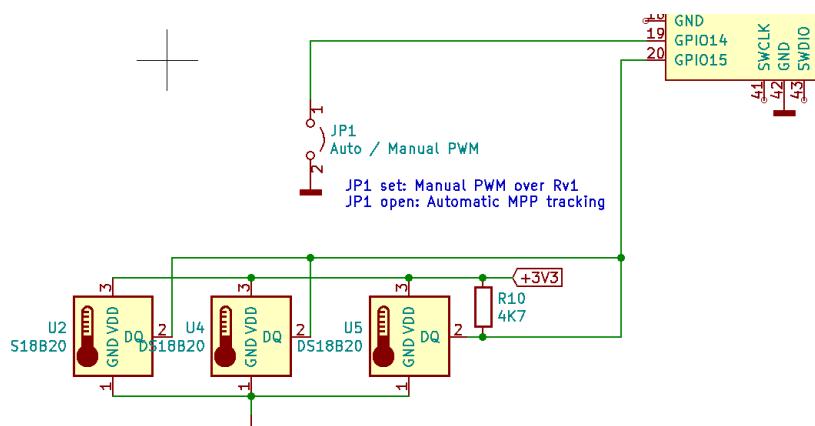
Für die Spule wurde hier eine Parallelschaltung von zwei "geretteten" 330uH – Ringkernspulen benutzt (Foto siehe Kapitel 12, links oben)

23. Temperaturüberwachung

Bei Leistungen über 300W erwärmen sich einige Bauteile ziemlich stark.

Um besser abschätzen zu können ob der Betrieb noch sicher ist, und gegebenenfalls einen Lüfter einzuschalten, wurde die Schaltung mit 3 DS18B20- Sensoren erweitert:

- am MOSFET
- am als Diode geschalteten MOSFET
- an einer der parallel geschalteten Spulen



Zum Auslesen wurde eine Klasse geschaffen:

```
class TemperatureSensors():
    def __init__(self, ds_pin):
        self.ds_sensor = ds18x20.DS18X20(onewire.OneWire(ds_pin))
        self.addresses = self.ds_sensor.scan()

    def convert(self):
        if len(self.addresses):
            self.ds_sensor.convert_temp()

    def get(self):
        self.temps = []
        for a in self.addresses:
            t = self.ds_sensor.read_temp(a)
            self.temps.append(t)
        return self.temps

    def print_temps(self, separator):
        for temp in self.temps:
            print(temp, end = separator)
        print()

    def checktemp(self, alarmtemp):
        alarms = []
        globalalarm = False
        temps = self.get()

        for t in temps:

            if t > alarmtemp:
                alarms.append(1)
                globalalarm = True
            else:
                alarms.append(0)
        return globalalarm, alarms
```

Die Temperaturmessung lässt sich so einfach in die Haptschleife integrieren:

```
ds_pin = Pin(15)
...
sensors = TemperatureSensors(ds_pin)
print(sensors.addresses)
def main_loop():
    global i, maxpower
    while True:

        sensors.convert()
        stemp = str(sensors.get())
        #sensors.print_temps('\t\t')

        # MPP track every track_time:
        if ( i % track_time) == 0:
            if pwm_manual.value():
                mppt.mpp_track()

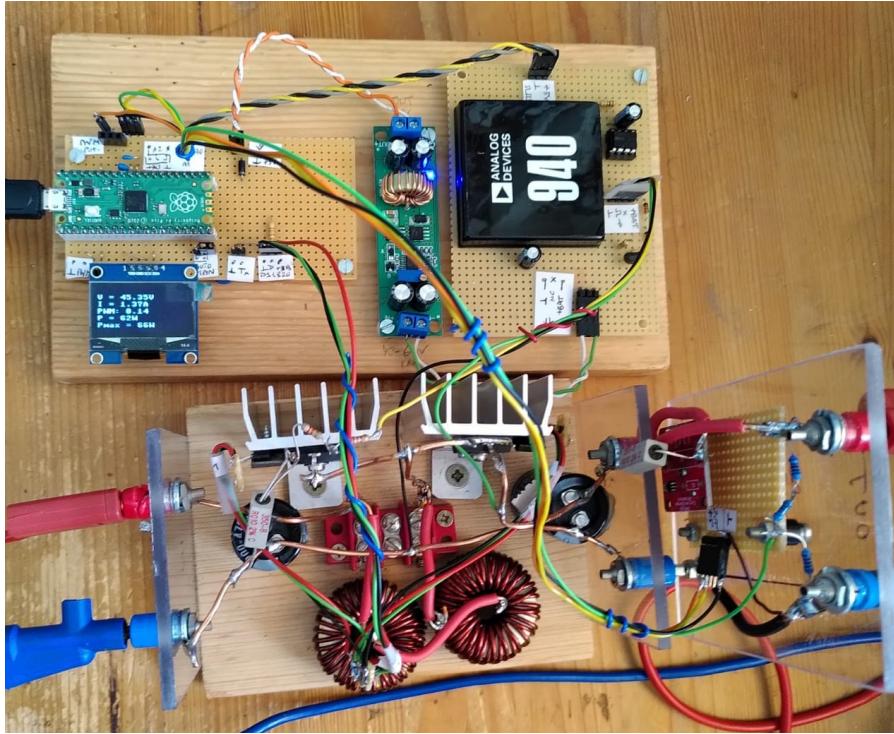
        V, I, P = mppt.get_VIP()

        # Remember max power over whole operation
        if P > maxpower:
            maxpower = P

        ...

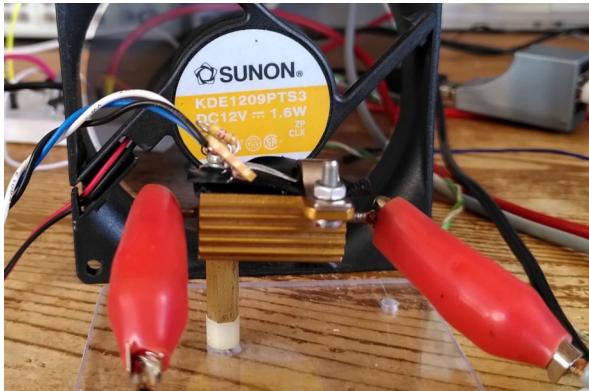
        i+= 1

main_loop()
```

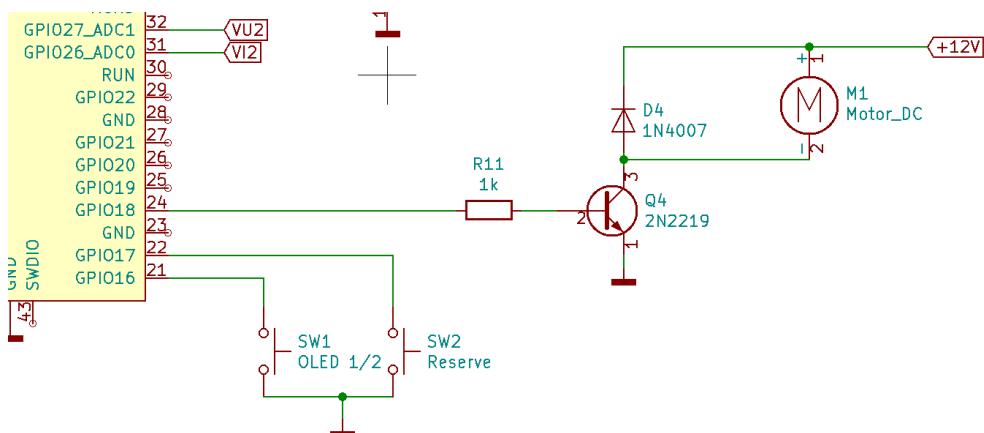


24. Temperaturregelung, Test

Zum Test wurde eine kleine Schaltung improvisiert:

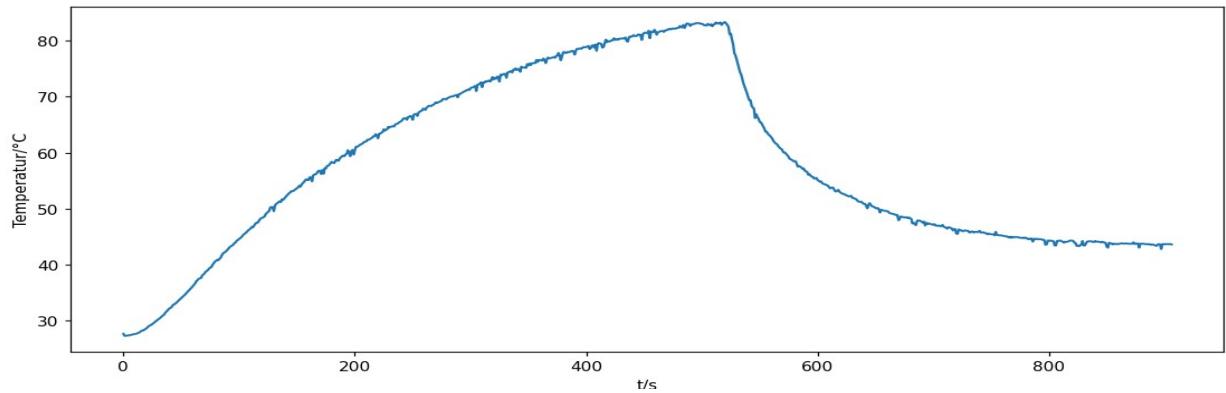


Auf einem Hochlastwiderstand ist der Sensor befestigt. Im Hintergrund der Ventilator, der durch einen Transistor geschaltet wird:

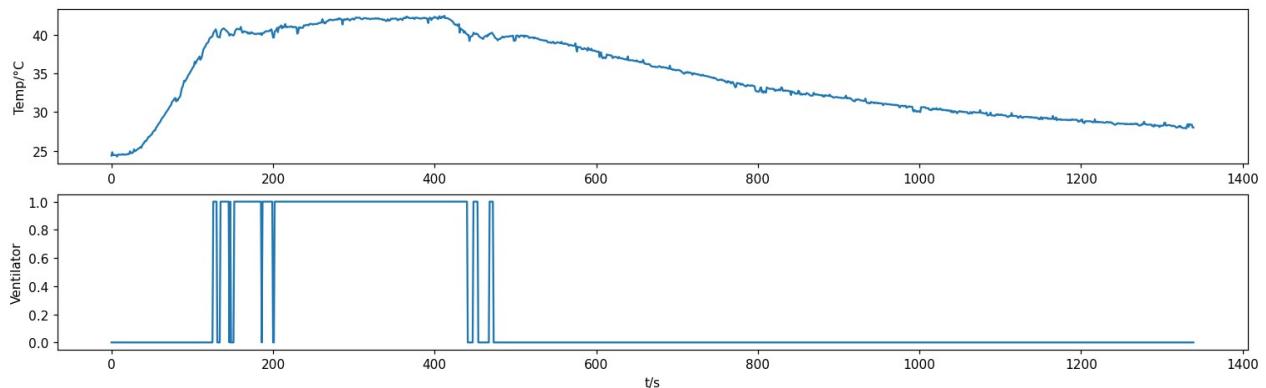


Ein ehemaliger Lehrer mit Hauptfach Regelungstechnik kann natürlich nicht widerstehen, zunächst einmal die Sprungantwort der Strecke aufzunehmen.

Links ohne, rechts (nach ca. 500s) mit Ventilator, aufgenommen bei einer Leistung von 5.5W für den Widerstand:



Mit Regelung sieht es dann so aus:



Hier wird der Ventilator bei einer Temperatur von 40°C eingeschaltet (ohne Hysterese)

```
trigger_temp = 40
ventilator = Pin(18, Pin.OUT)
sensors = TemperatureSensors(ds_pin)
print(sensors.addresses)

i = 0
while True:
    sensors.convert()
    time.sleep(0.75)
    temps = sensors.get()
    ga, a = sensors.checktemp(trigger_temp)
    if ga:
        ventilator.on()
    else:
        ventilator.off()

    for temp in temps:
        print(i, '\t', temp, end = '\t')
    print(ga)

    i += 1
```

25. Vorläufiges Ergebnis

11.12.2023

Software solar_track14.py

Die Schaltung bewährt sich mit drei Temperatursensoren, je einer für MOSFET, Diode und Spule(n). Wird eines dieser Bauteile zu heiss, so wird der Lüfter eingeschaltet.

In dieser Konfiguration überstand der Leistungsteil Ausgangsleistungen von bis zu ca. 750W ohne Probleme. Leider ist gerade (im Dezember) eine schlechte Zeit, um zu testen bis zu welcher Leistung die Schaltung durchhält. Es ist einfach nicht genug Sonnenschein da.

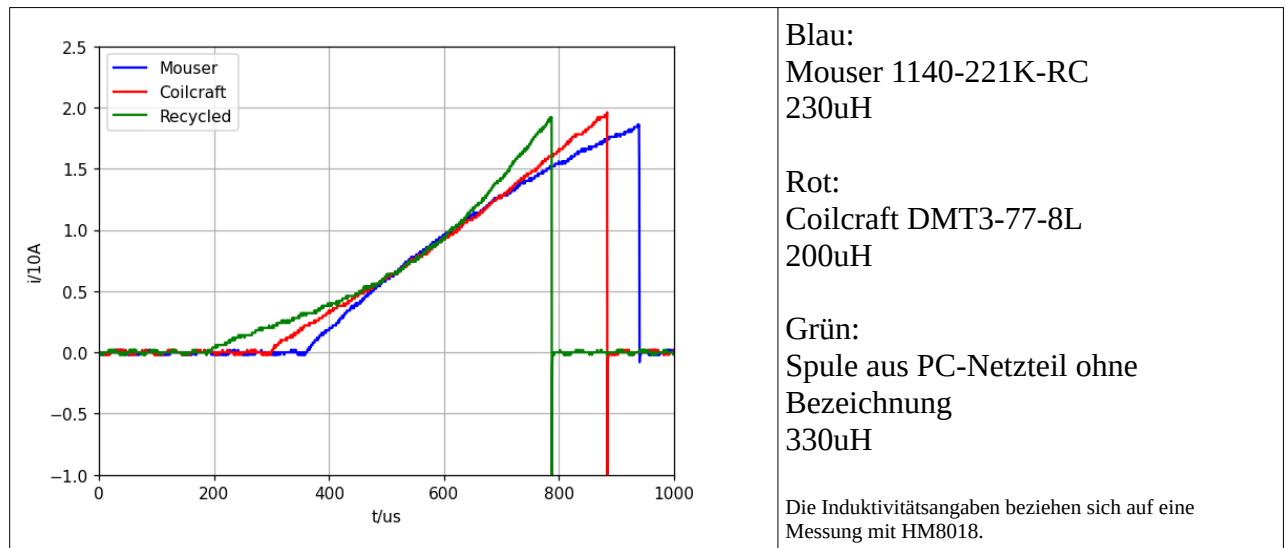
Um die aktuellen und die maximalen Temperaturen anzuzeigen wurden 2 Taster vorgesehen. Das Display wird einfach entsprechend umgeschaltet.

26. Intermezzo: Spulen, noch einmal

Der Wandler läuft momentan ganz gut mit 2 recycelten 330uH-Spulen aus PC-Netzteilen, welche parallel geschaltet sind.

Da ich über ein Sample-Angebot von Coilcraft gestolpert bin, erschien es mir aber sinnvoll, einige davon zu bestellen, denn es ist schwierig einige gleichartige Spulen zu bekommen, wenn man nur recycelt. Dies wäre aber notwendig oder zumindest wünschenswert bei mehrmaligem Aufbau des Wandlers.

Hier das Ergebnis einer Vergleichsmessung:

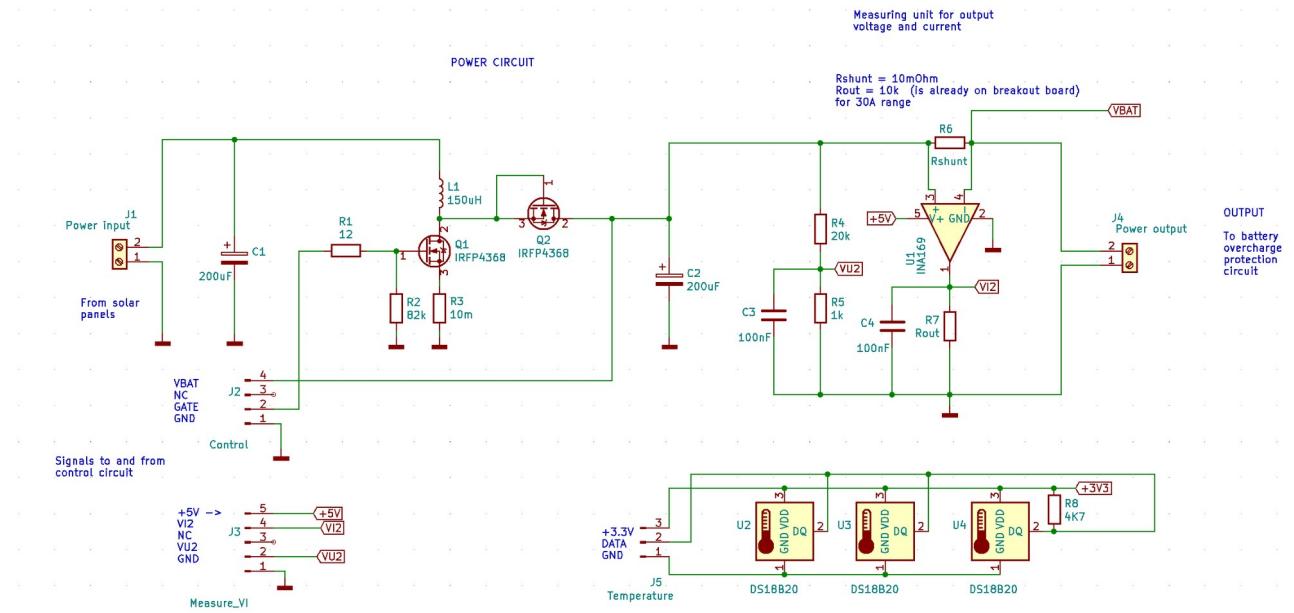


Man sieht,

- dass die grüne Kurve nach oben gekrümmmt ist, die Spule also schon bei ca. 5A amfängt in die Sättigung zu gehen.
- dass die blaue Kurve eine leicht exponentielle Krümmung hat, hier sind also schon Kupfer- oder Kernverluste im Spiel
- dass die rote Kurve bis ca. 20A recht linear verläuft, also entweder noch keine starke Sättigung auftritt (laut Datenblatt aber nur noch 100uH bei 10A) oder die Krümmung teilweise durch die Verluste komensiert wird.

27. "Definitiver" Leistungsteil

17.12.2023



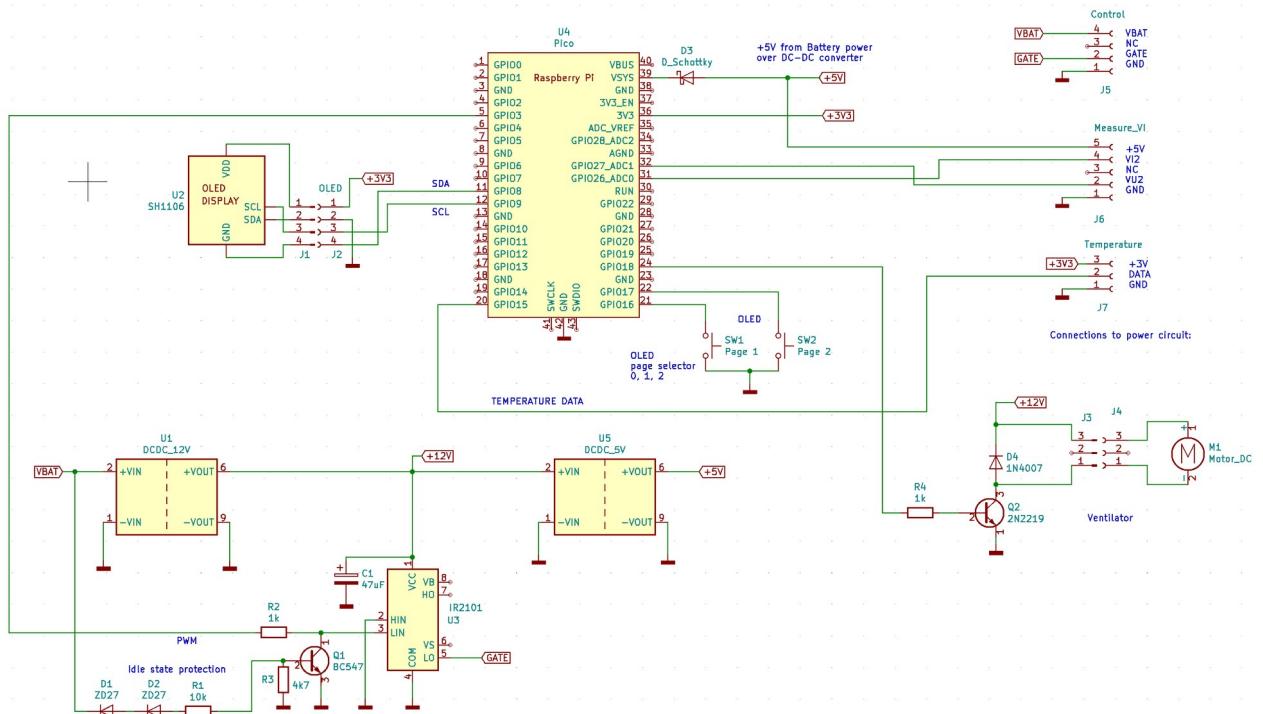
Das Leistungsmodul wurde aufgebaut mit 4 Coilcraft – Spulen DMT3-77-8L

200uH die ich als Samples erhielt, in Gruppenschaltung 2x seriell parallel geschaltet.

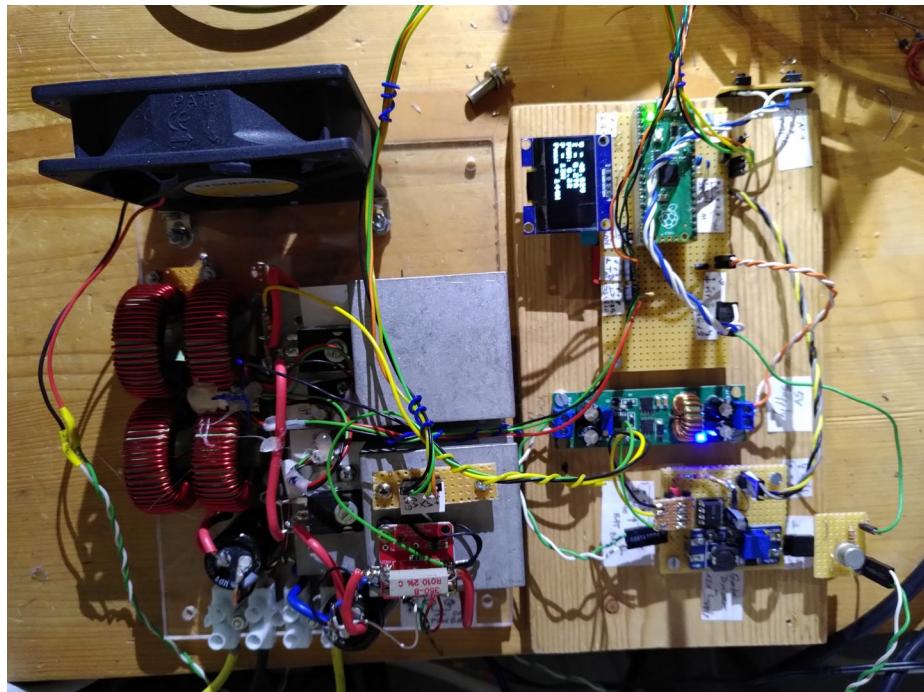
Damit hoffe ich eine Leistung von ca. 1kW zu erreichen.

28. "Definitiver" Steuerteil

20.12.2023



Achtung: die Schaltung auf dem Foto hat eine andre Stromversorgung (VBAT → 5V → 12V)



29. Remote-Zugriff auf die Software

Bald (hoffentlich !) wird diese Schaltung ihren Dienst ausserhalb des Hauses verrichten.

Wie nehme ich dann Änderungen in der Software vor? Das wird unbequem, mit dem Laptop anstatt vom Basteltisch aus. Ein PicoW würde den Zugriff über WiFi ermöglichen, aber das ist zu unzuverlässig und eigentlich möchte ich wieder alle Datentransporte in Kabel zurück verbannen.

Der Pico hat eine verfügbare serielle Schnittstelle. Die wäre geeignet. Es gibt nämlich (theoretisch) die Möglichkeit, die Terminalausgabe dorthin zu duplizieren. Das klappt auch ganz gut, wie ein Beispielprogramm zeigt. Einzig die Programmunterbrechung und das Reset klappen zumindest bei meiner Version von Micropython nicht.

Bis jetzt fallen mir 2 Möglichkeiten ein:

- Laut Forum gibt es im Quellcode die Möglichkeit mit dem Parameter DUPTERM (?) das Terminal zu duplizieren
- Ich muss in einer Schleife auf einkommende Zeichen warten und bei Bedarf die richtige Aktion machen.

Die zweite Möglichkeit scheint mir einfacher und sicherer, sie funktioniert auch wie ein Beispielprogramm zeigt:

```
import os
import time

from machine import UART, reset
uart = machine.UART(0, baudrate=115200)
os.dupterm(uart)

i=0
while 1:

    if uart.any():
        inp = uart.read()

        if b'\x04' in inp:
            reset()
        if b'\x03' in inp:
            break          # stop running program

    print(i)
    i+=1
    time.sleep(1)
```

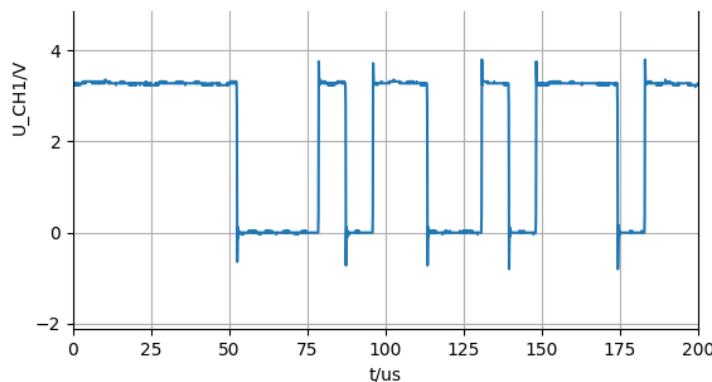
Das Programm zählt hoch, das Ergebnis ist über USB serial und UART verfügbar. Es kann mit <Ctrl-C> unterbrochen werden, und ein Reset mit <Ctrl-D> funktioniert auch.

30. Signalqualität

Zunächst hatte ich gedacht mit kleiner Baudrate zu arbeiten, da die Daten ja nicht schnell übertragen werden müssen. Das Programm PicoConnect wurde dafür ein wenig erweitert, doch es wird kompliziert und unübersichtlicher wenn die Baudrate für die klassische Verbindung `/dev/ttyACM0` und eine zusätzliche `/dev/ttyUSB0` unterschiedlich sein sollen.

Der Hinweis von weigu.lu, er hätte über Netzwerkkabel mit 115200 Baud schon grosse Distanzen überbrückt, ermutigte mich zu einem Experiment. Das Oszillogramm zeigt das Signal nach einem Hin- und Rückweg (keine Riesendistanz, insgesamt vielleicht 20m, aber mehr braucht es hier nicht).

Es sieht ganz ordentlich aus, und im Terminal zeigen sich keine Fehler.



31. Software für den Pico

Die Software wurde ein wenig aufgeräumt und modularisiert.

Der geforderte Remote-Zugriff erforderte eine recht neue Micropython-Version, ich habe diese extra aus dem Sourcecode kompiliert um in den “Genuss” des dupterm – Befehls zu kommen.

Momentan benutze ich die Version 1.23.

Hauptprogramm:

```
import onewire, ds18x20
from machine import Pin, ADC, I2C, UART, reset, soft_reset
import sys
import time
from OLEO_03 import OLED
from measure_vip import Measure_VIP      # Measure V2, I2, P2 with INA169
from pwmc import PWMc                   # PWM generator
import os
import blink
from temperature_sensors import TemperatureSensors
from MPPT_01 import *
#-----
# Parameters loaded from config.py, or default values if no config.py
try:
    from config import *
except:
    # load default parameters:
    use_oled = True
    track_time = 10                      # track every xx seconds
    dt = 1                                # display every dt

    pwm_min = 0.1
    pwm_max = 0.6
    pwm_step = 0.01
    powerlimit = 900           # Don't exceed this power, exit track loop when exceeded
    PWMfreq = 16E3

    vent_on_temp = 45      # switch on ventilator at this temperature
```

```

baud = 115200           # for UART0

# -----
# Hardware parameters:
# Temperature sensors
ds_pin = Pin(15)

# Buttons
btn1 = Pin(16, Pin.IN, Pin.PULL_UP)      # oled switch pages
btn2 = Pin(17, Pin.IN, Pin.PULL_UP)

led = Pin(25, Pin.OUT)

# ADC
a0 = Pin(26, Pin.IN)          # this is needed to turn input to high impedance
a1 = Pin(27, Pin.IN)          # this is needed to turn input to high impedance
a2 = Pin(28, Pin.IN)
adc0 = ADC(0)                 # Pin 31          Current I2
adc1 = ADC(1)                 # Pin 32          Voltage V2
adc2 = ADC(2)                 # Pin 34          Manual PWM pot
i2c_channel = 0
sclpin = 9
sdapin = 8
i2c = I2C(i2c_channel, scl=Pin(sclpin), sda=Pin(sdapin))

# PWM
PWMPin = 3
pwmgen = PWMC(PWMPin, freq= PWMfreq)

# Jumper auto PWM / manual PWM used now to activate remote REPL
### pwm_manual = Pin(14, Pin.IN, Pin.PULL_UP)

# OLED
if use_oled:
    try:
        oled = OLED(128, 64, i2c, rotate = 180)
        oled.clear()
    except:
        print("# NO OLED ?")
        oled = None
else:
    oled = None

# ventilator
vent_pin = 18
ventilator = Pin(vent_pin, Pin.OUT)

# serial data over Tx on Pin1:
uart0 = UART(0, baudrate=baud, tx=Pin(0), rx=Pin(1), bits=8, parity=None, stop=1)
uart0.write("# SOLAR TRACK\n")

# With Micropython 1.23: Communication over serial
os.dupterm(uart0, 0)

# -----
# Define class instances for temperature sensors + MPPT

# temperature sensors:
sensors = TemperatureSensors(ds_pin)
print("# ", sensors.nbsensors, " temperature sensors:")
print("# ", sensors.addresses)

# voltage + current sensors:
mppt = MPPT(adc0, adc1, pwmgen, oled, powerlimit = powerlimit)
mppt.nbmmean = 10
mppt.pwm_min = pwm_min
mppt.pwm_max = pwm_max
mppt.pwm_step = pwm_step

mppt.set_pwm(0)

## Use a dedicated print function, though MeasureVIP has one
def print_my_values( additional = ""):
    if i % 10 == 0:
        print("# i", "V2/V", "I2/A", "P2/W", "PWM", "E/Wh", "TMOS", "TD", "TCOIL", "TMmax", "TDmax", "TCmax",
              "P2max/W", "Limited", sep = '\t')
        #print(i, '\t', "%2.2f" % V, '\t', "%2.2f" % I, '\t', "%2.2f" % P, " ## %3.1f " % maxpower, " %1.2f" % mppt.pwmval,
        additional, "\t", sensors.get_maxtemps_as_string())

        print(i, "%2.2f" % V, "%2.2f" % I, "%3.0f" % P, " %1.3f" % mppt.pwmval, " %i" % EnergyWh, stemperatures +
              sensors.get_maxtemps_as_string() + " %i " % maxpower, mppt.powerlimit_reached, sep = '\t')

## Use dedicated oled function though MPPT has one

```

```

def print_my_oled(mppt, page = 0, additional = ""):
    # Display values if oled is defined
    if mppt.oled:
        if page == 0:
            V, I, P = mppt.values
            mppt.oled.clear()

            s = '\tV = %2.2fV' % V
            s += '\tI = %2.2fA' % I
            s += '\tPWM: %1.2f' % mppt.pwmval
            s += '\tP = %2.0fW' % P
            s += '\t' + additional
            mppt.oled.print_s(s)
        if page == 1:
            stemp = sensors.get_as_string()
            stemp = "Temperatures\tFET DIODE COIL\t" + stemp
            mppt.oled.print_s(stemp)
        if page == 2:
            stemp = sensors.get_maxtemps_as_string()
            stemp = "Max. Temps\tFET DIODE COIL\t" + stemp
            mppt.oled.print_s(stemp)
#-----



def check_buttons():

    if btn1.value():
        page = 0
    else:
        page = 1
    if btn2.value() == 0:
        page = 2

    return page


def check_temperature():
    # get temperatures and switch ventilator
    global stemperatures
    sensors.convert()
    stemperatures = sensors.get_as_string()
    ga, a = sensors.checktemp(vent_on_temp)
    if ga:
        ventilator.on()
    else:
        ventilator.off()


def check_uart():
    # STOP or RESET via RxD0:
    if uart0.any():
        inp = uart0.read()
        # stop running program or reset:
        if b'\x04' in inp:
            uart0.write("## RESET\n")
            time.sleep(0.1)
            soft_reset()
        if b'\x03' in inp:
            uart0.write("## STOP PROGRAM\n")
            #break
            sys.exit()
#-----



i = 0
maxpower = 0           # max power over whole operation
interval = 0
time_before = time.time()
EnergyWs = 0
page = 0
#-----
time1 = time.time()

def main_loop():
    global i, interval, time1, page
    global V, I, P, mppt, EnergyWs, EnergyWh, maxpower, stemperatures

    while True:

        interval = time.time() - time1
        if interval >= dt:
            blink.blink(1, dt = 0.01)

            check_temperature()

            # MPP track every track_time:
            if ( i % track_time) == 0:
                mppt.mpp_track()

```

```

# Get voltage, current, power, energy
V, I, P = mppt.get_VIP()
EnergyWs += P * interval
EnergyWh = EnergyWs / 3600

# Remember max power over whole operation
if P > maxpower:
    maxpower = P

# track again if power exceeds powerlimit -> limit power:
if P > powerlimit:
    mppt.mpp_track()

# Print + display values
print_my_values( )
### print_Tx0_table(i, V, I, P, mppt.pwmval, additional = str(EnergyWh) + '\t' + stemperatures)
print_my_oled(mppt, page = page, additional = 'Pmax = %2.0fW' % maxpower)

# Nb of measurement + time
i+= 1
time1= time.time()

page = check_buttons()
check_uart()

main_loop()

```

Modul MPPT_01:

```

from machine import Pin, ADC, I2C, UART, reset
import time
from OLED_03 import OLED
from measure_vip import Measure_VIP          # Measure V2, I2, P2 with INA169
from pwmc import PWMc                         # PWM generator

class MPPT(Measure_VIP):
    def __init__(self, adc0, adc1, pwmgen, oled = None, powerlimit = 500):
        Measure_VIP.__init__(self, adc0, adc1, oled)      # inherits everythin from Measure_VIP

        # PWM generator:
        self.pwmgen = pwmgen
        self.pwmval = 0
        self.pwm_min = 0.1
        self.pwm_max = 0.5
        self.pwm_step = 0.01
        self.powerlimit = powerlimit
        self.powerlimit_reached = False

        ''' Inherited from Measure_VIP:
        self.i
        self.v
        self.p
        self.values = v, i, p
        '''

    def set_pwm(self, pwmval):
        self.pwmgen.set_pwm(pwmval)
        self.pwmval = pwmval

    def mpp_track(self, printflag = True):
        # Tracks maximum power point in PWM range between self.pwm_min and self.pwm_max using self.pwm_step
        # returns p2max, p1max, pwmopt, i1max, i2max, pwmopt
        # printflag decides if values are printed
        ## Note: Only in this function pwm values are int in % because range is used

        # range accepts only int, so convert to %
        pwm_percent_min = int(self.pwm_min * 100)
        pwm_percent_max = int(self.pwm_max * 100)
        pwm_percent_step = int(self.pwm_step * 100)
        pwm_percent_range = range(pwm_percent_min, pwm_percent_max , pwm_percent_step)

        if printflag:
            print( '# PWM track', '\t', "PWM%", '\t', "V2/V", '\t', "I2/A", '\t', "P2/W", '\t', "P2max")

        power_max = 0
        pwmopt_percent = 0
        vmax = 0
        imax = 0
        self.powerlimit_reached = False

        for pw in pwm_percent_range:

            self.set_pwm(pw/100)
            time.sleep(0.01)                      # allow some time for results to be stable
            v, i, power = self.get_VIP()

```

```

if power > power_max:
    vmax = v
    imax = i
    power_max = power
    pwmopt_percent = pw

if power >= self.powerlimit:
    self.powerlimit_reached = True
    break

if printflag:
    print( '## PWM track', '\t', pw, '\t', '%2.2f' % v, '\t', '%2.2f' % i, '\t',
          '%3.0f' % power, '\t', '%3.0f' % power_max)

self.pwmval = pwmopt_percent/100
self.set_pwm(self.pwmval)
time.sleep(0.01)

if printflag:
    print("## PWM = %2.0i" % pwmopt_percent, "%")
    print("## P = %2.0f" % power_max, "W")
    print("## Power limit reached:", self.powerlimit_reached)
    print()

return power_max, pwmopt_percent, vmax, imax

## overwrite print_oled function from Measure_VIP
## to add print PWM value
def print_oled(self, additional = ""):
    # Display values if oled is defined
    if self.oled:
        V, I, P = self.values
        self.oled.clear()

        s = '\tV = %2.2fV' % V
        s += '\tI = %2.2fA' % I
        s += '\tPWM: %1.2f' % self.pwmval
        s += '\tP = %2.0fW' % P
        s += '\t' + additional
        self.oled.print_s(s)

```

Modul measure_vip:

```

from machine import Pin, ADC, I2C
import time
from OLED_03 import OLED
from pwmc import PWMc

class Measure_VIP():
    '''Read V, I from adc0, adc1 and calculate P
    values can also be printed and displayed on OLED
    adc0 -> I
    adc1 -> V'''

    def __init__(self, adc0, adc1, oled = None):
        self.oled = oled
        self.adc0 = adc0
        self.adc1 = adc1
        self.offset0 = -0.006
        self.k0 = 0.98522
        self.offset1 = -0.007
        self.k1 = 0.98791
        self.ifactor = 10.2866779
        self.vfactor = 21
        self.nbmean = 3

    def readADC0(self, nb):
        v = 0
        for i in range(0,nb):
            v += self.adc0.read_u16()
        v = v/nb * 3.3 / (65535)
        v += self.offset0
        v *= self.k0
        return v

    def readADC1(self, nb):
        v = 0
        for i in range(0,nb):
            v += self.adc1.read_u16()
        v = v/nb * 3.3 / (65535)
        v += self.offset1
        v *= self.k1
        return v

    def get_I(self):
        v = self.readADC0(self.nbmean)

```

```

        i = v * self.ifactor
        return i

    def get_V(self):
        v = self.readADC1(self.nbmean)
        v = v * self.vfactor
        return v

    def get_VIP(self):
        i = self.get_I()
        v = self.get_V()
        p = v * i
        self.i = i
        self.v = v
        self.p = p
        self.values = v, i, p
        return v, i, p

    def print_VIP(self):
        print("%2.3f" % self.v, '\t', "%2.3f" % self.i, '\t', "%3.1f" % self.p)

    def print_oled(self):
        # Display values if oled is defined
        if self.oled:
            v, i, p = self.values
            self.oled.clear()

            s = '\tV = %2.2f' % v
            s += '\tI = %2.2fA' % i
            s += '\tP = %2.0fW' % p

            self.oled.print_s(s)
#-----

def measure_loop():

    # ADC
    a0 = Pin(26, Pin.IN)                                # this is needed to turn input to high impedance
    a1 = Pin(27, Pin.IN)                                # this is needed to turn input to high impedance
    adc0 = ADC(0)                                      # Pin 31
    adc1 = ADC(1)                                      # Pin 32
    i2c_channel = 0
    sclpin = 9
    sdapin = 8
    i2c = I2C(i2c_channel, scl=Pin(sclpin), sda=Pin(sdapin))
    oled = OLED(128, 64, i2c, rotate = 180)
    oled.clear()

    #meas = Measure_VIP(adc0, adc1)                      # without oled
    meas = Measure_VIP(adc0, adc1, oled)                 # with oled
    meas.nbmean = 10

    while True:
        v, i, p = meas.get_VIP()
        meas.print_VIP()
        meas.print_oled()
        time.sleep(1)

if __name__ == "__main__":
    measure_loop()

```

Modul pwmc:

```

from machine import PWM, Pin

class PWMC:
    def __init__(self, pin, freq = 5000):
        # set PWM pin + frequency
        self.pwm = PWM(Pin(pin))
        self.pwm.freq(int(freq))
        self.pwm.duty_u16(0)

    def set_pwm(self, value):
        # set PWM value 0.0 to 1.0, returns integer value 0...65535 corresponding to PWM
        self.value = value                      # remember last value
        pwmval = int(65535 * value)
        if pwmval > 65535: pwmval = 65535
        if pwmval < 0: pwmval = 0
        self.pwm.duty_u16(pwmval)
        return pwmval

    def set_freq(self, freq):
        # set new PWM frequency without changing the PWM value, returns frequency
        self.pwm.freq(int(freq))
        self.set_pwm(self.value)
        return self.pwm.freq()

```

```

def stop(self):
    # deinit PWM
    self.set_pwm(0)
    self.pwm.deinit()

if __name__ == '__main__':
    print("Start PWM on GPIO 3, 200kHz, 60%")
    pw = PWMc(3, freq= 200E3)
    pw.set_pwm(0.6)

    import time
    time.sleep(2)

    print("Setting frequency to 1MHz")
    f = pw.set_freq(1000E3)
    print(f, "Hz")

    time.sleep(2)

    print("Setting PWM to 20%")
    d = pw.set_pwm(0.2)
    print(d ,"/ 65535")

    time.sleep(2)

    print("PWM stopped")
    pw.stop()

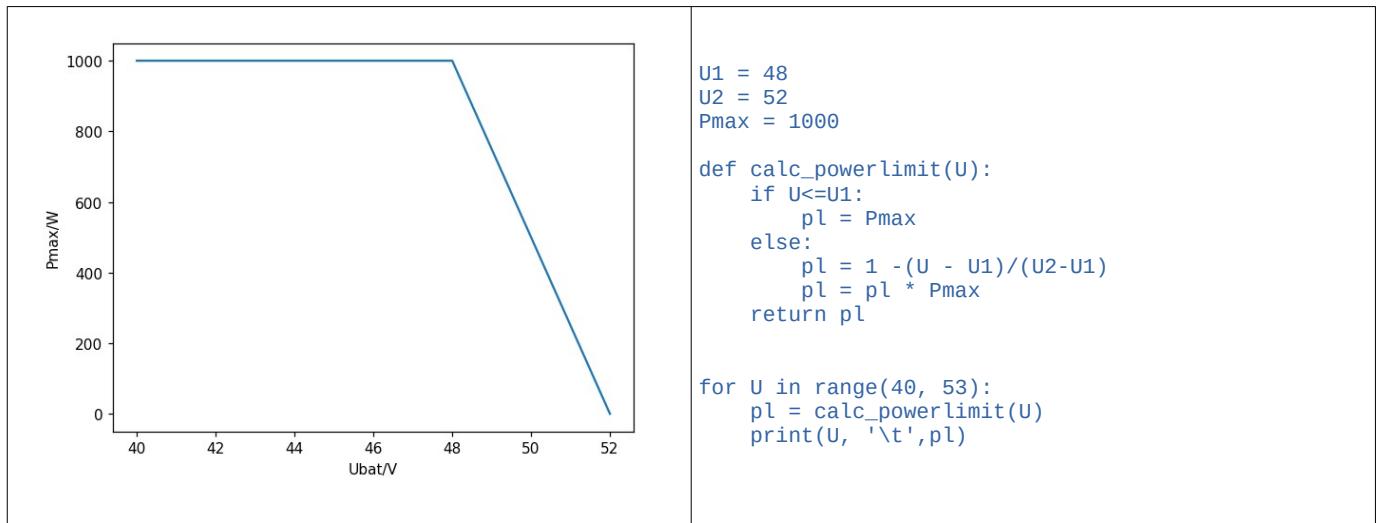
```

32. Leistungsbegrenzung bei fast voller Batterie

15.3.2024

Einige Tage mit etwas Sonne haben gezeigt, dass 1kW Leistung erreicht werden kann und dann die analoge Spannungsüberwachung den Ladevorgang bei relativ voller Batterie periodisch abschaltet.

Dies ist nicht so schön und Batterie-schonend wie ein sanfter Übergang, z.B. so:



33. Aufbau eines zweiten Leistungsteils

16.7.2024

Bei gutem Wetter wird die Leistung der Solarpanel nicht voll genutzt da der Wandler eine softwaremässige Begrenzung von 800W oder 1000W hat. Deswegen soll die Einspeisung auf 2 Wandler verteilt werden.

Der zweite Leistungsteil ist praktisch identisch mit dem ersten und funktioniert prinzipiell.

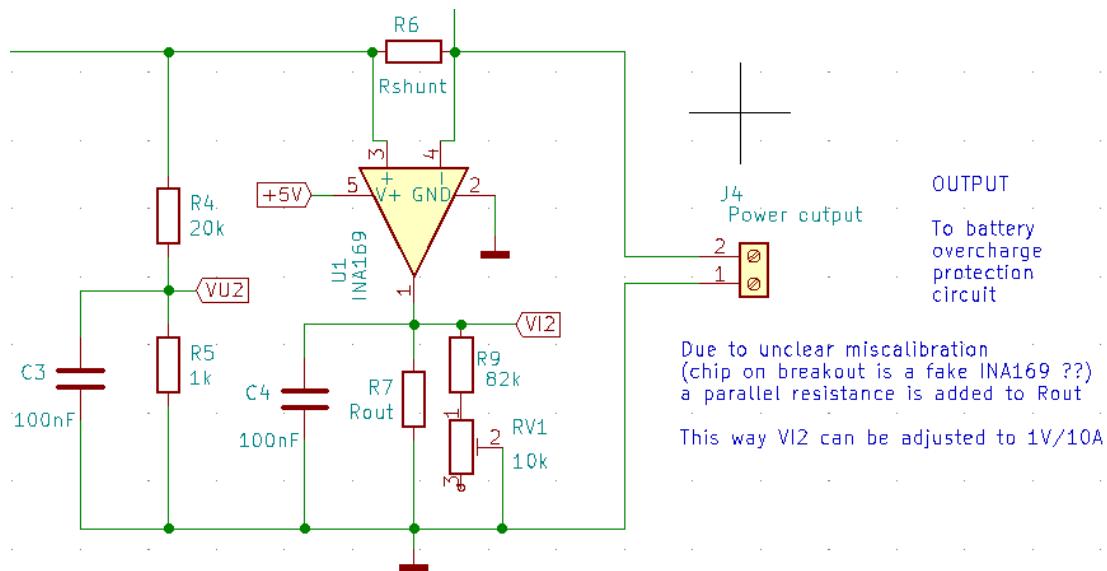
Nun haben sich aber bei genauer Betrachtung einige Tücken gezeigt:

Messwandler für den Strom

Dieser mit INA169 (angeblich!) bestückte Sensor auf einem Breakoutboard liefert ein falsches Signal. Die gemessenen Stromwerte sind wesentlich höher als die tatsächlichen.

Vielleicht ist der INA169 auf dem Board eine billige chinesische Kopie ?

Workaround:



Mit dem Trimmstabspotentiometer kann VI2 auf 1V / 10A eingestellt werden.

Die Korrektur hätte man auch softwaremässig machen können, aber so kann VI2 mit dem Voltmeter gemessen und leicht umgerechnet werden.

Störsignale

Die Temperatursensoren wurden über eine kleine Verteilerplatine angeschlossen, die direkt auf den MOSFET-Kühlkörper geklebt wurde. "Ob das gut geht?" hat sich schon eine kleine Stimme in meinem Hinterkopf gemeldet. "Wir werden sehen!" war die Antwort. Und wir sahen es. Nicht oft, aber hin und wieder verabschiedete der Pico sich ins Nirvana.

OK, da muss eine Abschirmung her!

Eine mit Masse verbundene Vollkupfer-Platine löste das Problem.

Achtung: Masse ist nicht gleich Masse! In diesem Fall: kurze Verbindung zur Masse des Störfrieds, nämlich des Leistungsteils.

34. Watchdog

23.7.2024

Aufgrund der Erfahrungen mit Störsignalen wurde die Idee des Watchdogs im Programm wieder aufgegriffen. Die Implementation ist einfach, aber tückisch. Ohne Vorsichtsmassnahmen landet man gerne in einer Reboot-Schleife wegen der man aus Thonny nicht mehr auf seinen Pico zugreifen kann.

Deswegen wurde ein Jumper an GP14 benutzt, der einen Programmstart ohne Watchdog erlaubt. So kann man ohne Stress das Programm entwickeln und erst im fertigen Zustand den Jumper setzen.

Am Anfang des Programms (solartrack_25.py):

```
67 # Watchdog only when WDT jumper is set
68 jp_WDT = Pin(14, Pin.IN, Pin.PULL_UP)
69 if jp_WDT.value() == 0:
70     from machine import WDT
71     wdt = WDT(timeout=6000) # enable it with a timeout of 6s, max. = ca. 8s
```

In der Schleife wird dann regelmässig der Watchdog “gefüttert”:

```
def main_loop():
    global i, interval, page, n_rows
    global V, I, P, mppt, EnergyWs, EnergyWh, maxpower, stemperatures

    last_t = time.ticks_ms()

    while True:

        # only when WDT is active:
        try:
            wdt.feed()
        except:
            pass
```

Das try – except ist notwendig da wdt ohne gesetzten Jumper nicht definiert ist.

Meine erste Idee war es, statt des Jumpers eine Taster zu benutzen, der ein sys.exit() und damit einen Programmstop bewirkt hätte. Da aber der Watchdog nicht zu stoppen ist wenn er einmal läuft funktioniert diese Idee nicht.

Vor 3 Jahre hatte ich schon eine diskussion im Forum zu diesem Thema:

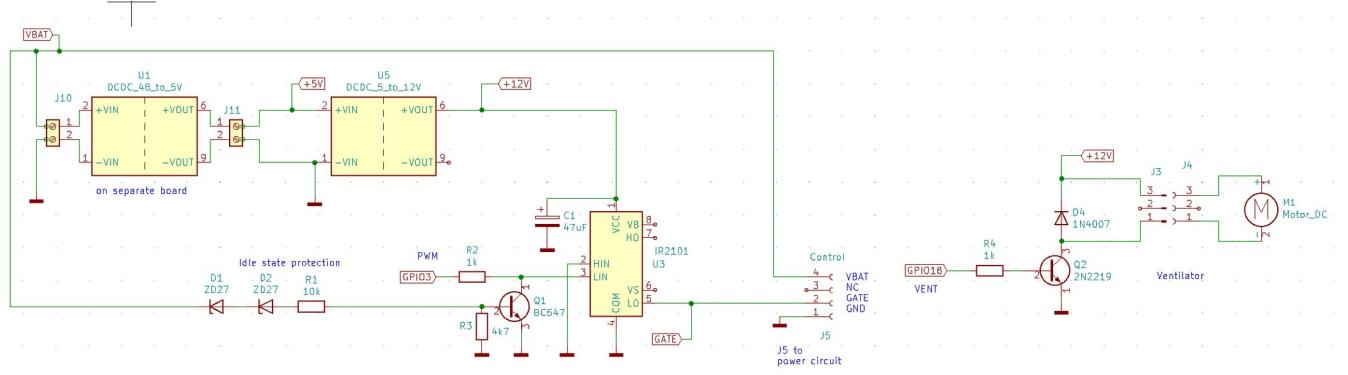
<https://forum.micropython.org/viewtopic.php?t=12751>

35. Weitere Änderungen an der Software

- Die Temperaturen werden statt der maximalen Leistung auf Seite 1 des Displays angezeigt.
- Die Daten für HomeAssistant werden über UART1 (TX = GPIO4, RX = GPIO5) geschickt.

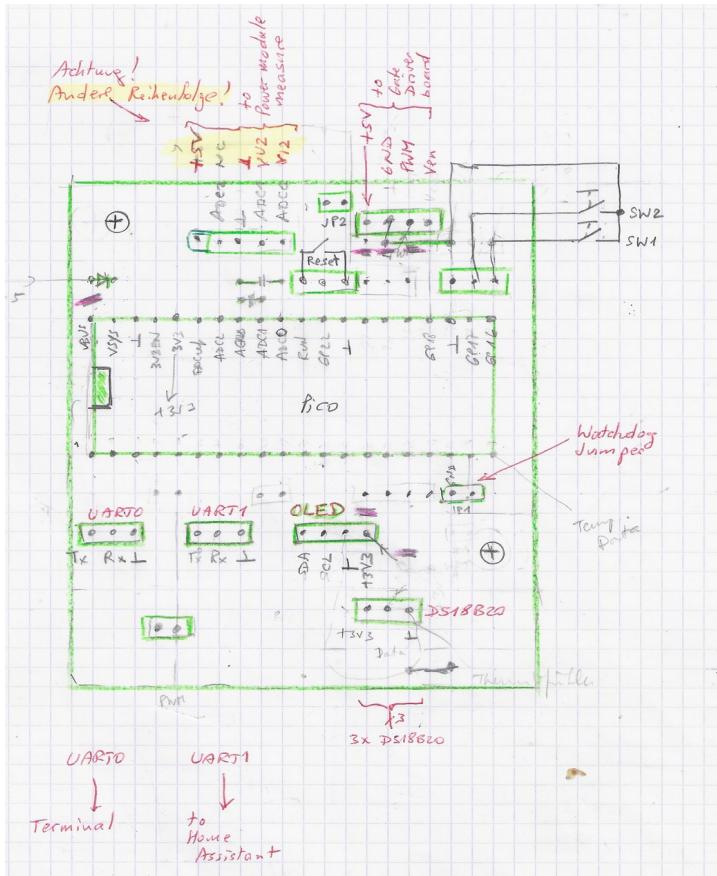
→ solartrack_26.py

36. Gate Driver Platine für den zweiten Aufbau



37. Testprozeduren

Controllerplatine



- main.py enthält zu Beginn am besten ein kleines Blinken für alle Tests.

Man sieht sofort ob der Controller startet:

```
import blink
blink.blink(3, dt = 0.1)
```

- Buttons (test_buttons.py)
- Jumper (test_jumper.py)
- Temperatursensoren (test_ds18b20_02.py)
- Serielle Schnittstellen (test_uart_repl.py)

Achtung: Micropython >= 1.23 für REPL über UART!

Achtung: in main.py das Laden des solartrack_xx.py verhindern.

Statt dessen nur: "import test_uart_repl", so dass das Testprogramm sofort gestartet wird.

Warum? Thonny stört eventuell die Funktion.

UART0 über USB-Adapter verbinden und Terminalprogramm starten (115200 baud).

Nun ist eine inkrementale Zahlenfolge zu sehen.

Mit Ctrl-C kann ein REPL erzwungen werden.

Mit Ctrl-D erfolgt ein Reset.

Mit "*" wird das laufende Programm gestoppt, es kann die Dummy-Funktion hi() aufgerufen werden, die "HELLO" ausgibt. Mit <Enter> wird das Programm fortgesetzt. Ebenso bei falschem Kommando.

Nun auch UART1 verbinden und zweites Terminalprogramm starten (9600 Baud).

Hier müssen nun die ausgegebenen Werte auch zu sehen sein.

Achtung: die Reihenfolge der Ports nach einer Unterbrechung ist willkürlich, wobei das erste angeschlossene zu Beginn die Nummer 0 hat.

- PWM (test_pwm.py)
- Ventilator (test_ventilate.py)
- OLED (test_oled.py)
- ADC (test_adc.py zeigt Spannungswerte 0...3.3V an)
- Autonomer Betrieb (main.py, solartrack_24.py)

Mit +5V extern eingespeist, ohne USB-Speisung soll die interne LED 3x blinken, danach jede Sekunde.

Steuerteil

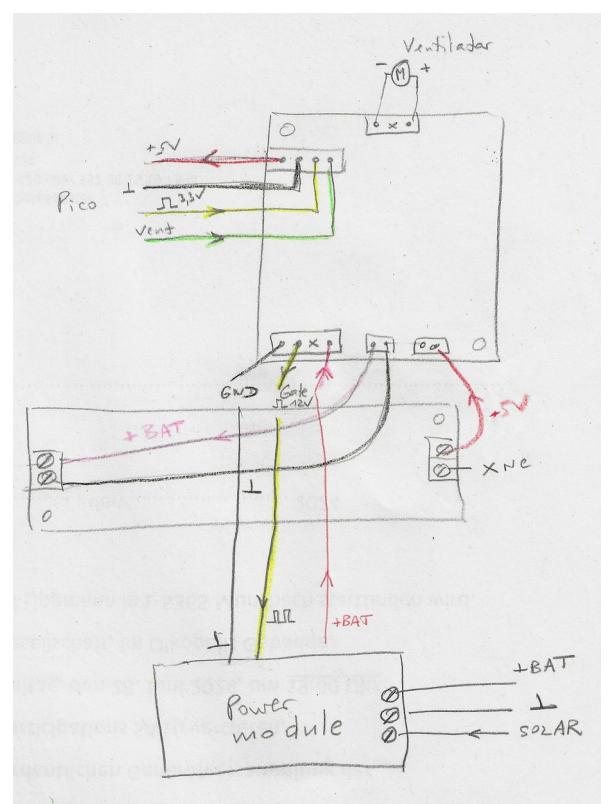
(Controller + Gatedriver)

Statt des Powermoduls wird ein Netzgerät an +BAT und GND angeschlossen (U = 48V)

Vor dem Verbinden des Picos:

- +5V des 48V/5V-Wandlers kontrollieren
- +12V des DC/DC-Wandlers auf der Gatedriverplatine kontrollieren.
- Pico anschliessen und test_ventilator.py test_pwm.py ausführen.

Kontrollieren ob der Gatedriver 12V-Impulse liefert.



38. Probleme mit einem Elko und noch ein Sensor

Beim Test des zweiten Leistungsteils explodierte irgendwann der Stützkondensator am Ausgang, mit der Folge dass der 48V/5V – DC/DC – Wandler auch den Geist aufgab. Sonst scheint alles unversehrt.

Der Elko war zwar neu, aber er schien mir schon optisch zu klein für diese Aufgabe zu sein.

Der Elko des ersten Leistungsteils wird auch warm, er scheint dies aber besser zu verkraften wegen seiner deutlich grösseren Bauform.

Hier bietet es sich an, auch die Temperatur des Elkos zu messen.

Es muss also noch ein DS1820 in die Schaltung integriert werden.

Hierbei stellt sich das Problem der Reihenfolge der Sensoren. Die Adresse sollte nicht zwischen, sondern hinter den anderen liegen.

Tests ergaben eine “quick and dirty” – Lösung, die vielleicht nicht optimal ist, aber den Zweck erst mal erfüllt. Während die anderen Sensoren DS18B20 sind (Family code 0x28), war der zugeschaltete Sensor ein DS1820 (Family code 0x10). Diese Sensoren scheinen automatisch hinter den DS18B20 eingesortiert zu werden. (Achtung: dies könnte sich natürlich mit einer neuen DS18x20-Bibliothek ändern).

Der Elko wurde durch ein voluminöseres Exemplar mit 450V Spannungsfestigkeit ersetzt.

Erste Messungen zeigen eine akzeptable Temperatur die immer etwas unter der Spulentemperatur bleibt (normalerweise unter 40°C).

39. Probleme mit Störspannungen

Bei zusammengebautem Leistungsmodul mit Steuermodul zeigte sich, dass der Pico öfter hängen blieb und dank des Watchdogs neu bootete.

Eine Analyse der Ausgabe auf UART0 zeigte dass es sich immer um einen CRC Error beim DS1820 handelte.

Hier ein Auszug aus dem LOG:

```
5      47.77    7.59    362      0.350    0      43.3    42.2    47.632.5 43.6    42.4    47.6    32.7    362    0
7      47.65    7.46    355      0.350    0      43.4    42.1    47.632.4 43.6    42.4    47.6    32.7    362    0
9      47.78    7.30    349      0.350    0      43.4    42.1    47.7    32.4    43.6    42.4    47.7    32.7    362
0
Traceback (most recent call last):
  File "main.py", line 4, in <module>
    File "solartrack_27.py", line 312, in <module>
      File "solartrack_27.py", line 298, in main_loop
        File "solartrack_27.py", line 145, in print_my_oled
          File "/lib/ds18x20_jc03.py", line 37, in get_as_string
            File "/lib/ds18x20_jc03.py", line 28, in get
              ds18x20.py", line 1, in read_temp
                File "ds18x20.py", line 1, in read_scratch
Exception:
MicroPython v1.23.0-preview.87.g0285cb2bf on 2024-02-01; Raspberry Pi Pico with RP2040
Type "help()" for more information.
# SOLAR TRACK
          # Temperature sensor addresses: (Family, SerialNb, CRC)
# 0x28  27F907D6013C  239
# 0x28  42FF75D0013C  37
# 0x28  A82D75D0013C  30
# 0x10  3064C3020800  144
```

Immer wieder gab es (alle paar Sekunden) ähnliche Vorfälle, alle durch den DS18x20 Sensor.

Die Zuleitungen zu den Sensoren waren zwar nicht geschirmt, aber mit einem geerdeten (mit Power-Masse verbundenen) Schaltdraht spiralförmig umwickelt.

Es steht noch eine Untersuchung aus, wieviel diese Massnahme im Vergleich zu abgeschirmtem Kabel hilft. Die Halterungsplatine für den Controllerteil (kupferkaschiert) war mit Masse verbunden.

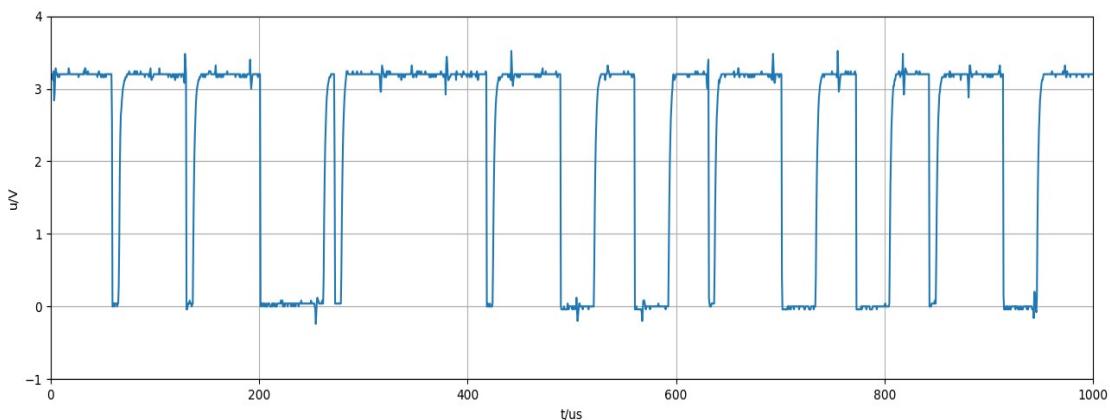
All dies reichte nicht aus für störungsfreien Betrieb.

Ein kleines Fachgespräch mit ChatGPT brachte nichts worauf ich nicht selbst hätte kommen können, dies aber in einer angenehmen wohlstrukturierten Form. Es hilft, und zwar mehr als eine frustrierende Google-Suche. Unter anderem brachte dies folgende Ideen:

- den Pullup – widerstand verkleinern (hier auf ca. 1.3k durch Parallelschalten von 1.8k)
- einen Entkoppel-C parallel zur Betriebsspannung
- einen Kondensator zwischen Datenleitung und Masse zur Tiefpassfilterung.

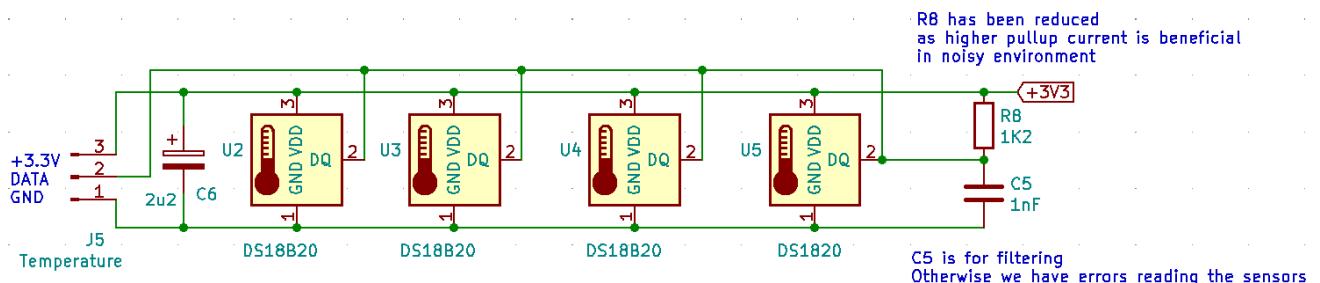
Interessanterweise wurde diese Massnahme erst auf meine Frage ("Would a small capacitor between Data and Ground line be helpful?") gutgeheissen. Der vorgeschlagene Wert (100nF ... 1uF) war aber viel zu hoch (er wäre für die Entkopplung der Betriebsspannung gut gewesen).

Ich wählte 1nF, was mit 1kΩ eine Zeitkonstante von 1µs ergibt, das schien mir sinnvoll, und ein Oszilloskop bestätigte diese Annahme:



Witzigerweise gab mir ChatGPT recht, als ich ihn (oder sie oder es ???) mit dem Fehler konfrontierte, und bestätigte dies durch Berechnung der Zeitkonstante.

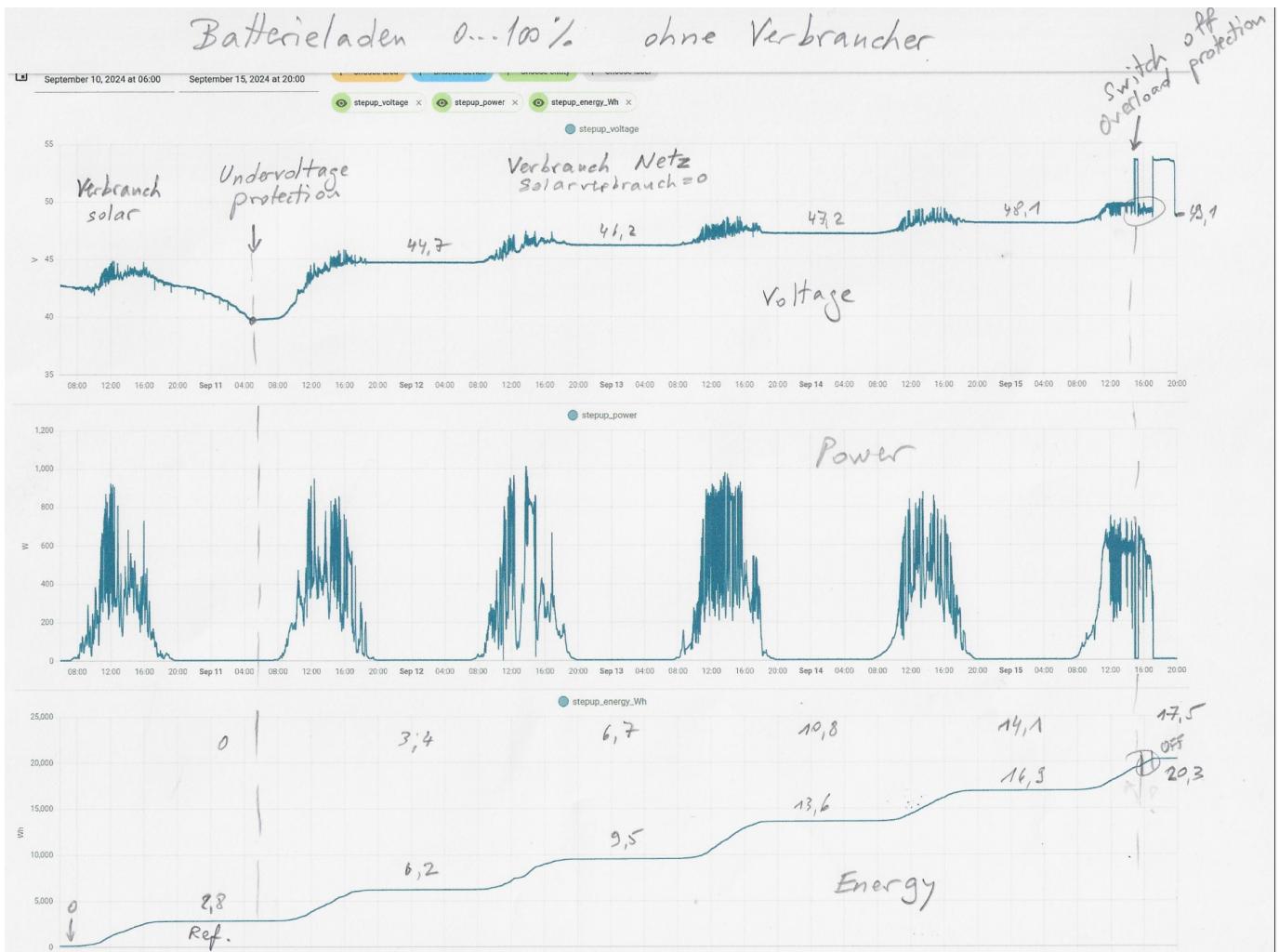
So ergab sich die modifizierte Schaltung für die Temperaturmessung:



40. Vollständige Ladekurve

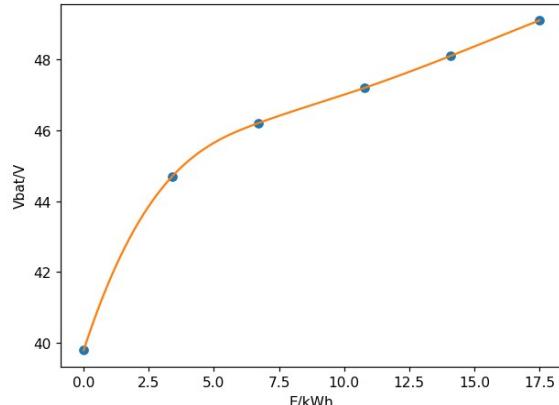
Nach einigen sehr trüben Tagen schaltete die Anlage auf Netzbetrieb, die Batterie war "leer".

Ich nutzte dies um eine vollständige Ladekurve ohne Verbraucher aufzunehmen:



Hieraus konnte ich den Zusammenhang zwischen aufgenommener Energie und Batteriespannung ermitteln:

E/kWh	Ubat/V
0	39.8
3.4	44.7
6.7	46.2
10.8	47.2
14.1	48.1
17.5	49.1



Der Zusammenhang ist weniger nichtlinear als ich erwartet hätte.

$$y=0.0023x^3 - 0.053x^2 + 0.5x + 40$$

mit $y = V_{bat}/V$ und $x = E/kWh$

Umgekehrt lassen sich die Punkte schlechter interpolieren, etwa mit

$$x = 0.2657 y^2 - 21.74 y + 444.6$$

41. Balancing ja oder nein?

Eine Kontrolle mit dem Voltmeter ergab wie bei den vorherigen Tests eine überall gleiche Zellspannung (von 4.07V nach dem Voll-Laden und einer Nacht Pause).

Also: ein Balancing scheint bis jetzt unnötig, möglicherweise aufgrund der Gruppenschaltung mit vielen parallelgeschalteten Zellen, deren Unterschiede sich von selbst ausbalancieren.

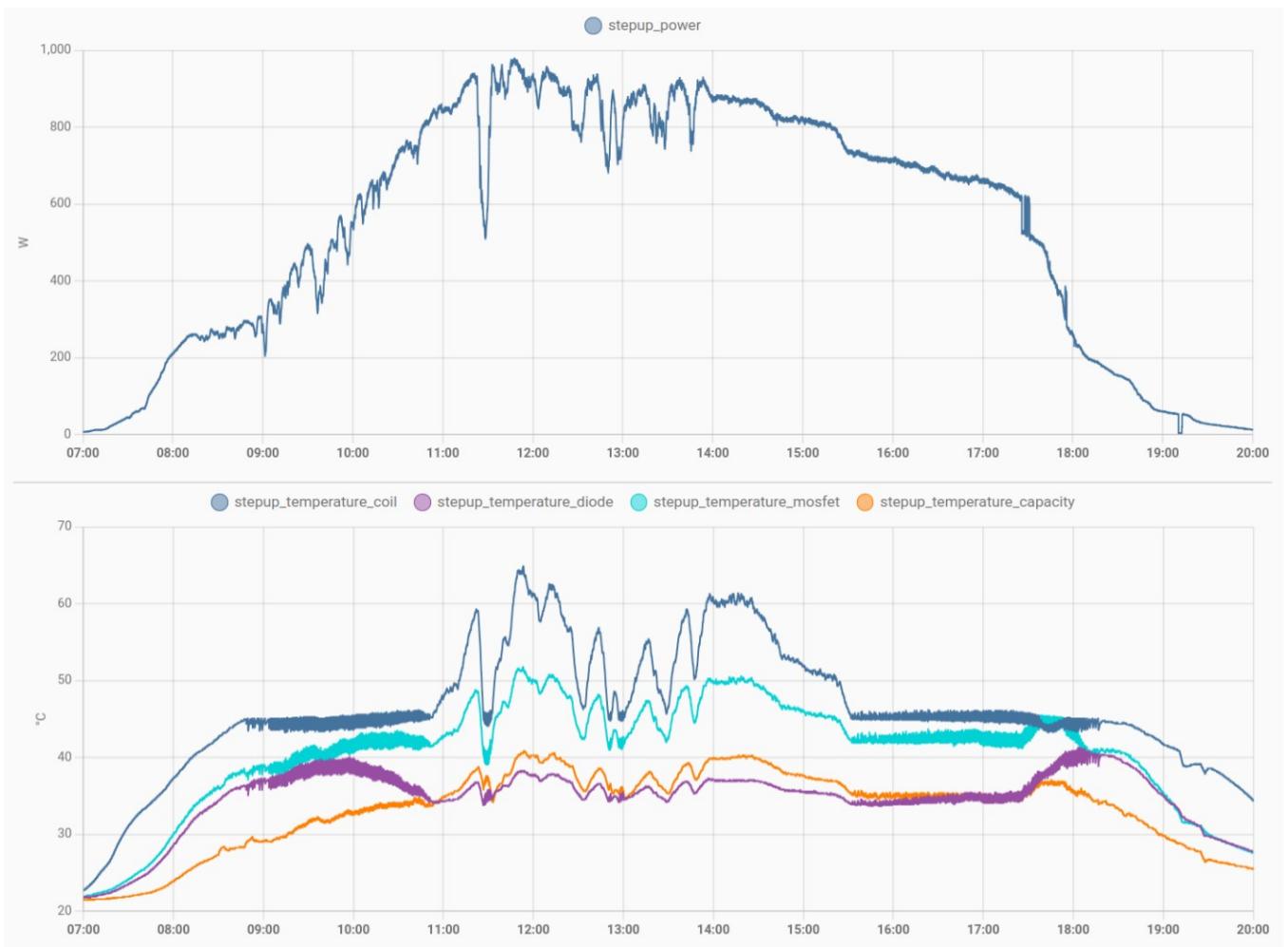
42. Erfahrungen mit richtig starker Sonneneinstrahlung

Der März 2025 ist ein trockener und sehr sonniger Monat.

Nach dem Winter mit wenig Sonne gibt es nun fast zuviel davon. Dies führt zu Leistungen von 800-950W über längere Zeit. Die Spulen werden bedenklich warm und der Lüfter schafft es nicht mehr ganz, die Temperatur im grünen Bereich zu halten:

Die Spulen sind hierbei die kritischsten Bauteile. Sie erwärmen sich auf bis zu 80°C.

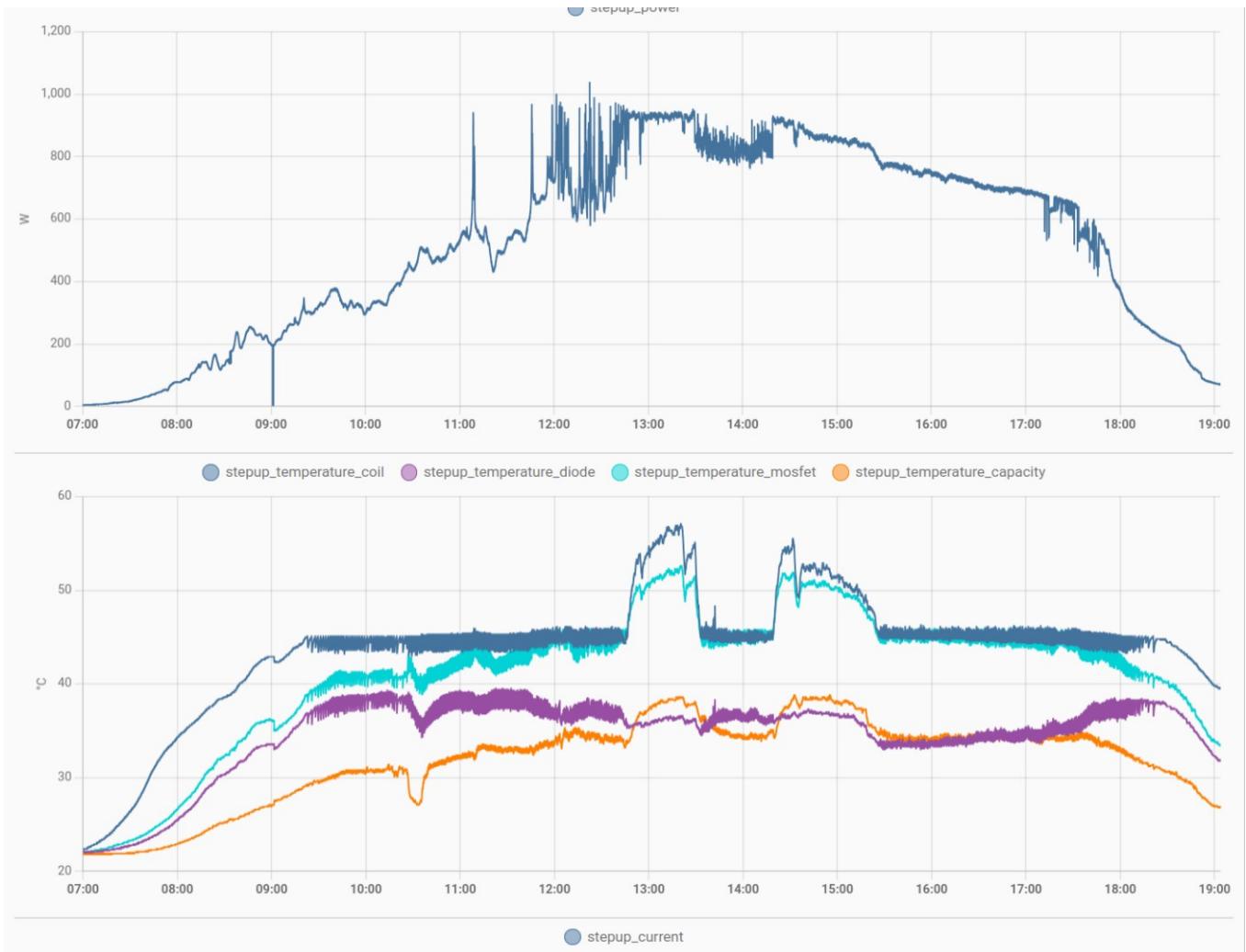
Ein Freiräumen des Luftwegs brachte schon eine grosse Verbesserung:



43. PWM-Frequenz 18kHz

Bis jetzt wurde die PWM-Frequenz auf 16kHz festgelegt um die Schaltverluste des MOSFET gering zu halten. Eine Erhöhung der Frequenz sollte die Schaltverluste erhöhen (der MOSFET wird wärmer) und die Spulenverluste durch geringere Welligkeit verringern, die Spule sollte weniger heiss werden.

Ein praktischer Versuch scheint dies zu bestätigen:

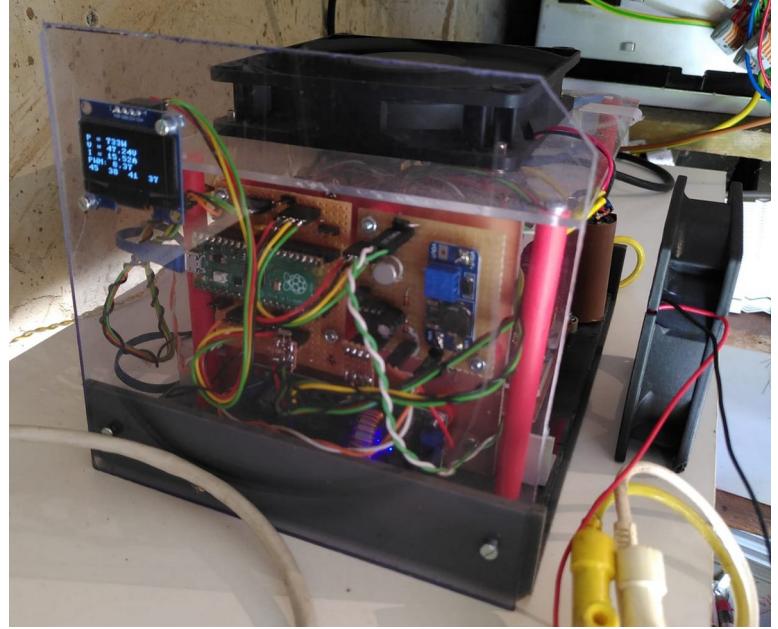
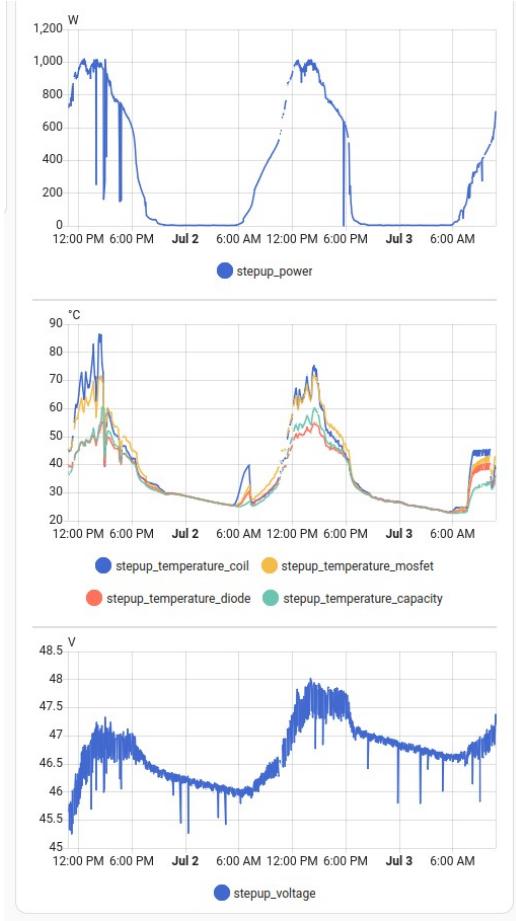


Nur bei ganz hohen Leistungen um 950W schafft der Ventilator es nicht, die Spule auf unter 50°C zu kühlen. Die erreichte Spulentemperatur liegt aber noch unter 60°C, was wesentlich besser ist.

44. Probleme mit extrem heissem Wetter

Anfang Juni 2025 stieg die Aussentemperatur tagsüber auf 35-38°C. Im “Batteriehaus” wurde es noch heißer. Die gute Nachricht: die schaltung überlebte ohne Ausfall, obwohl die Spulentemperatur zeitweise über 80°C lag.

Um für etwas zusätzliche Abkühlung zu sorgen, stellte ich einen zweiten Lüfter seitlich von den Spulen auf und betrieb ihn mit konstanten 12V aus einem Bleiakku, der noch an einem Solarpanel betrieben wird. Den Effekt sieht man im mittleren Diagramm (2. Juli). Die Temperatur fing schon an zu steigen, bis um 7:10 der Zusatzlüfter angeschlossen wurde. Sofort fällt die Temperatur wieder. Und auch später bleibt die Temperatur niedriger als am vorigen Tag, obwohl die Leistung um 1kW liegt.



45. Andere Lüfteranordnung

Aufgrund der vorigen Erfahrungen scheint es günstiger, 2 Lüfter zu benutzen, einen für die Spule und einen für die Halbleiter. Eventuell ist es auch sinnvoller, den Leistungsteil vertikal anzurichten so dass die warme Luft nach oben entweichen kann.

46. Nochmal: die Reihenfolge der Temperatursensoren

Die Hardware des ersten Leistungsteils soll nun für einen zweiten Wandler genutzt werden. Dabei muss ein Temperatursensor für den Ausgangselko hinzugefügt werden. Natürlich liegt der verwendete Sensor zufällig adressmäßig zwischen den schon vorhandenen. Anstatt zu fummeln soll eine brauchbare Lösung gefunden werden.

Dazu wird das Modul ds18x20_jc03 ge-updated als ds18x20_jc04 und mit einer Funktion versehen die die Reihenfolge der Adressen ändert:

```
import onewire, ds18x20
import time

class TemperatureSensors():
    def __init__(self, ds_pin):
        self.ds_sensor = ds18x20.DS18X20(ow.Onewire(ds_pin))
        self.addresses = self.ds_sensor.scan()
        self.nbsensors = len(self.addresses)
```

```

...
def re_sort_addr(self, indexes):
    # Reorder sensors: indexes e.g. [3, 1, 2, 0]
    print("# Reordering ", indexes)
    newaddresses = []
    for i in range(0, len(indexes)):
        n = indexes[i]
        a = self.addresses[n]
        newaddresses.append(a)
    self.addresses = newaddresses
    self.nbsensors = len(self.addresses)

...

```

In config_B.py wird die Reihenfolge festgelegt:

```

# User defined parameters (editable)
# This is for unit B

# new for unit B: (7.2025)
identity = "B"
ds1820_order = [0,2,3,1]
#-----
...
```

Damit funktioniert die Temperaturerfassung schon ohne Probleme.

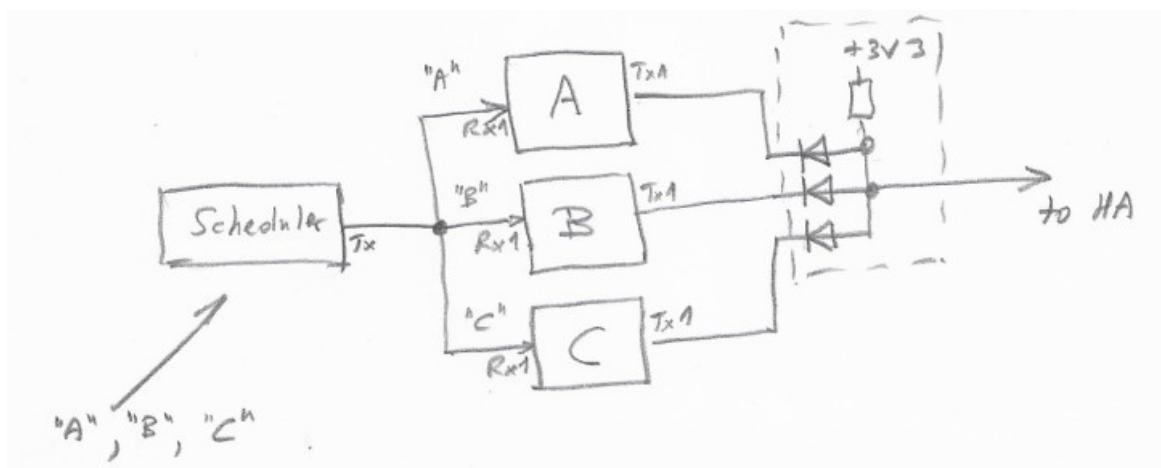
47. Überlegungen zur Datenübertragung für mehrere Module

Die Daten sollen weiter seriell mit 9600 baud übertragen werden.

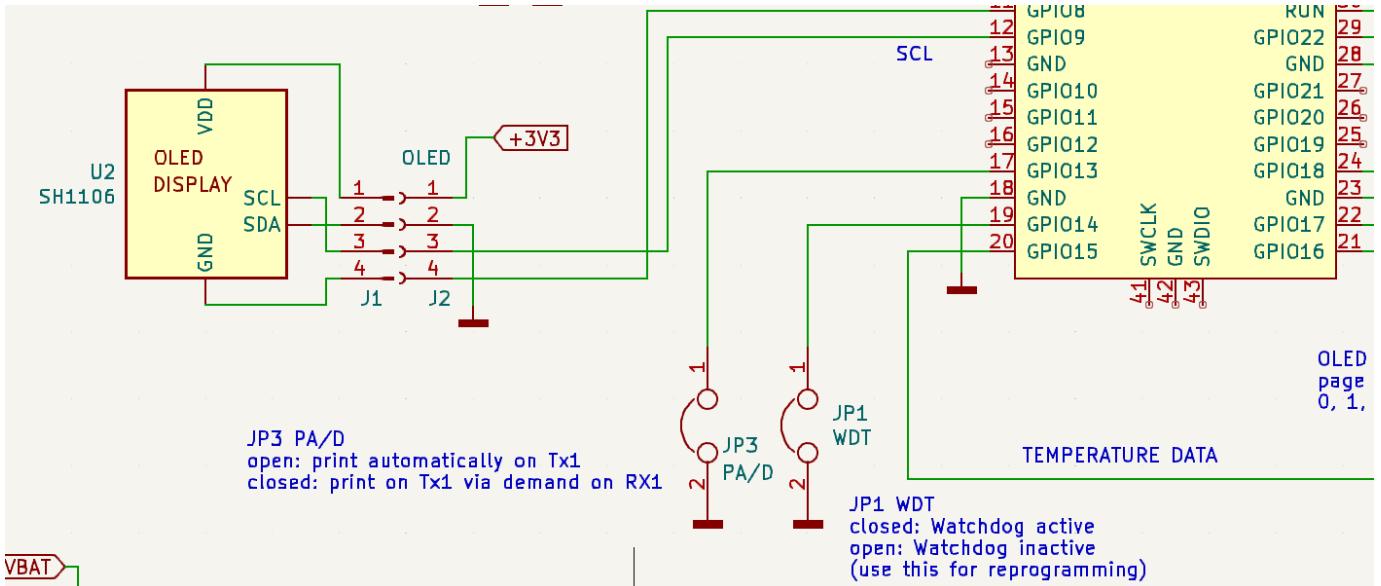
Es ist recht einfach, die Tx-Ausgänge (über Dioden zur Entkopplung) zu einem gemeinsamen Ausgang zusammenzufassen.

Es muss nur sichergestellt werden dass nicht 2 Module gleichzeitig senden.

Eine einfache Möglichkeit dazu wäre, dass ein Scheduler (Pico oder Signal von HomeAssistant) die Identität („A“, „B“ ...) über UART an alle Module schickt und diese nur reagieren wenn die Identität übereinander stimmt.



Es soll aber auch der automatische Modus beibehalten werden bei dem das Modul automatisch seine Werte schickt. Also wird ein Jumper zur Umschaltung eingeführt: JP3 PA/D (Print Auto / on demand) (Hierfür war im Schaltplan eigentlich schon JP2 Master / Slave vorgesehen, aber nun ist es einmal so, JP2 kann für was anderes benutzt werden.



Print Tx1:

JP3 closed: only on demand

JP3 open: automatically every printinterval and on demand

48. Änderungen im Hauptprogramm für Unit B

- Für die Temperaturmessung wird das neue Modul ds18x20_jc04 importiert.
Dies erlaubt es die Reihenfolge der Sensoren anzupassen.
- Weitere Hardware-Konfigurationen werden in config.py verlegt

49. Kommunikationsmöglichkeiten

- USB serial (→ Thonny):**

Programmierung, Datenausgabe

- UART0 115200 baud:**

Terminal, gibt die Daten in Tabellenform aus.

Mit <Ctrl-C> wird das Programm unterbrochen, es erscheint ein REPL-Prompt „>>>“

Mit <Ctrl-D> wird der Pico neu gebootet

Unterbrechen des Programms + Eingabe von Kommandos mit „*“, es erscheint „CMD“ Prompt.

Im Unterschied zu <Ctrl-C>, bei dem das Programm gestoppt wird, läuft es hier nach der

Befehlseingabe weiter. Dies ist wichtig, um im laufenden Programm Variablen ändern zu können oder Funktionen aufzurufen.

- **Variablen ändern**, z.B.:

```
print_interval = 10
```

```
track_interval = 30
```

```
print_header_every = 10
```

- **Funktionen aufrufen**, z.B.

```
track()
```

```
config_mppt(pwm_min, pwm_max, pwm_step)
```

```
set_freq(f)
```

```
ventilator.on()
```

```
ventilator.off()
```

- **UART1 9600 baud:**

- TX Ausgabe der Daten für HomeAssistant in Tabellenform

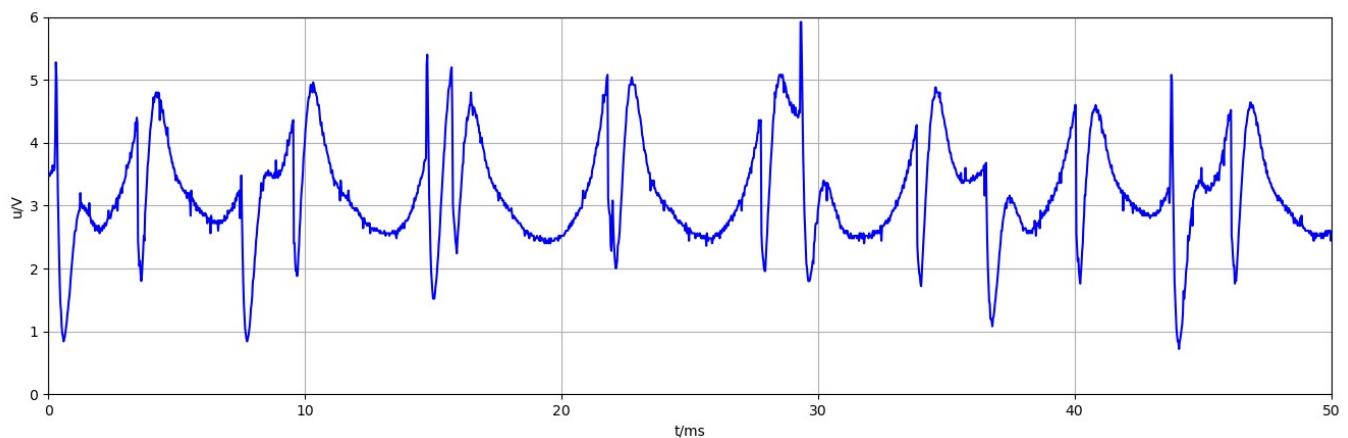
- RX: „B“ (für Modul B, „A“ für Modul A) → Ausgabe von Daten

(wird gebraucht wenn der Jumper J3 PA/D = Print Automatically or On Demand gesetzt ist)

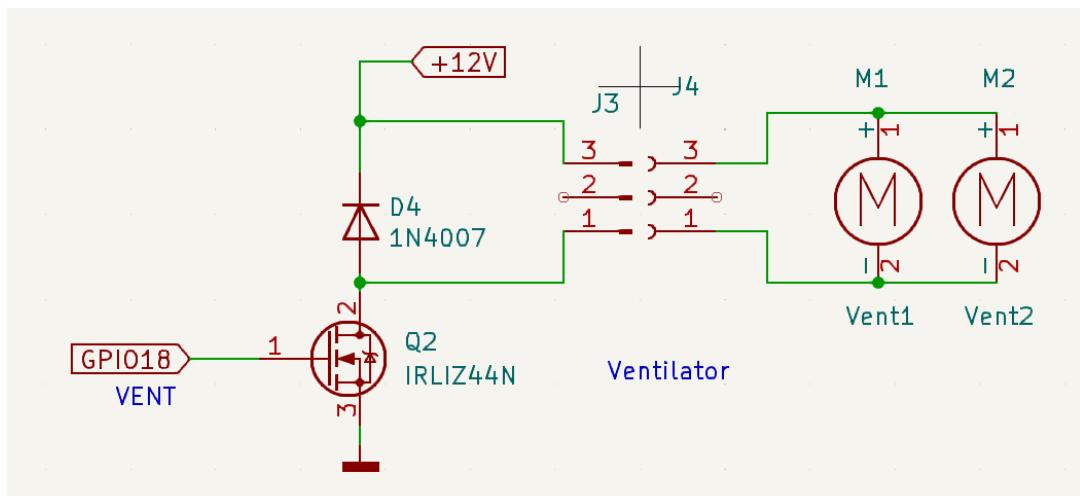
50. Hardware

Änderungen:

Zur besseren Kühlung zweiter Ventilator, Luftströmung von 2 Seiten in die Schaltung hinein, Abluft nach oben / unten wenn die Schaltung senkrecht hängt. Nun wird der Schalttransistor aber recht warm, das Oszilloskop zeigt dass er nicht ganz durchschaltet:



Mit einem Logic Level MOSFET IRLIZ44 ergibt sich eine drastische Verbesserung, nur noch ca. 50mV Spannungsabfall zwischen D und S.



51. Inbetriebnahme des neuen Wandlers Modul B

Probleme mit DS1820

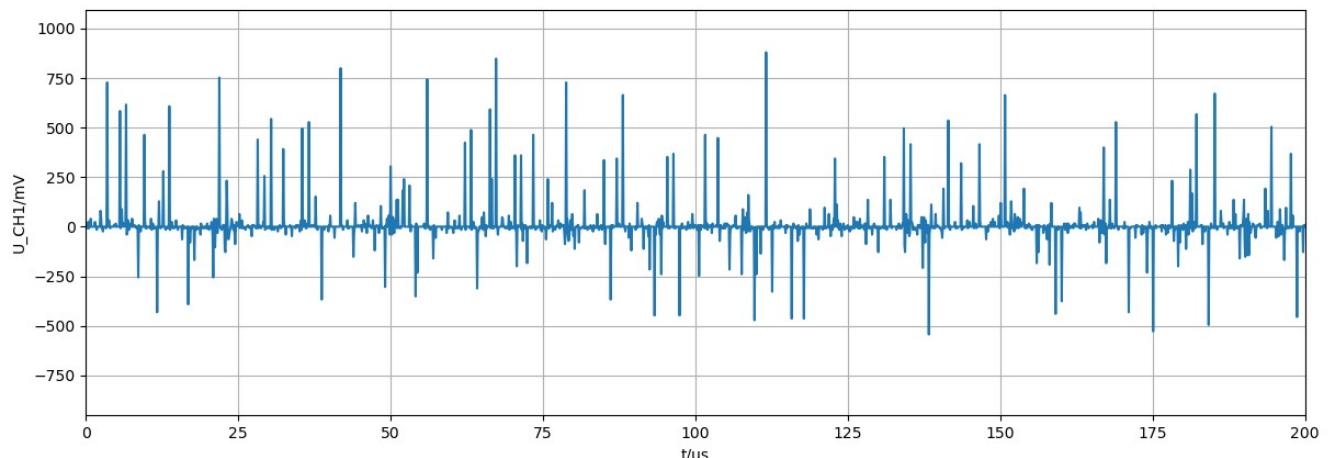
Wie beim Modul A sorgen Störungen der Übertragung für einen Programmausstieg, dies trotz Verringerung des Arbeitswiderstandes + Kondensator 1nF

```
Traceback (most recent call last):
  File "main.py", line 4, in <module>
    File "solartrack_29_B.py", line 357, in <module>
      File "solartrack_29_B.py", line 304, in main_loop
        File "solartrack_29_B.py", line 206, in check_temperature
          File "/lib/ds18x20_jc04.py", line 66, in get_as_string
            File "/lib/ds18x20_jc04.py", line 55, in get
              File "ds18x20.py", line 1, in read_temp
                File "", line 1, in read_scratch
Exception: CRC error
```

Der Watchdog Timer sorgt aber bei gesetztem Jumper für einen automatischen Neustart, was natürlich nicht die optimale Lösung ist.

Störfelder

Die Einstreuung, wahrscheinlich auch magnetisch, ist beträchtlich. Hier eine Aufnahme mit einer Leiterschleife von 5cm Durchmesser , ein Punkt an Masse, seitlich neben der Leiterplatte



Abschirmung

Die Datenleitung der DS1820 wird abgeschirmt, die Masse der Abschirmung nur Pico-seitig verbunden, GND und +3.3V normal verdrahtet.

Software

Das Modul ds18x20jc_04.py wird zu ds18x20jc_04.py, wobei die Methode get eine Fehlerbehandlung bekommt:

```

class TemperatureSensors():
    # ...

    def get(self):
        # get all temperatures as array and update maxtemps
        # to avoid problems with read errors that stop a program,
        # reading is done with try - except and on error, -300 is returned
        self.temps = []

        for i in range(0, self.nbsensors):
            a = self.addresses[i]

            try:
                t = self.ds_sensor.read_temp(a)
            except:
                t = -300

            self.temps.append(t)

            if t > self.maxtemps[i]:
                self.maxtemps[i] = t
        return self.temps
    
```

Im Fehlerfall wird die Temperatur auf -300 gesetzt, die Datenverarbeitung muss dann schauen ob sie die ganze Datenzeile ignoriert oder sonst was macht.

Interessanterweise zeigt die Log-Datei, dass nur eine Temperatur (hier mehrmals die des MOSFETs) das Problem verursacht:

#	i	V2/V	I2/A	P2/W	PWM	E/Wh	T_{MOS}	TD	TL	TC2	TMmax	TDmax	TLmax	TC2max
460	0	48.57	13.40	651	0.420	76	35.1	32.3	41.1	39.2	35.8	32.7	42.5	44.4
		0	856.8105											812
462	0	48.56	13.17	639	0.420	77	35.2	32.3	41.2	39.2	35.8	32.7	42.5	44.4
		0	859.7355											812
464	0	48.60	13.24	643	0.420	77	35.3	32.3	41.3	39.2	35.8	32.7	42.5	44.4
		0	850.543											812
466	0	48.57	13.19	641	0.420	77	35.2	32.4	41.3	39.2	35.8	32.7	42.5	44.4
		0	857.2283											812
468	0	48.58	13.11	637	0.420	78	35.2	32.4	41.3	39.3	35.8	32.7	42.5	44.4
		0	855.1407											812
470	0	48.56	12.98	630	0.420	78	-300.0	32.4	41.4	39.3	35.8	32.7	42.5	44.4
		0	858.901											812

Die anderen Temperaturen werden dann richtig gemessen.

Dies legt die Idee nahe, dass im Fehlerfall die eine Temperatur nochmal gemessen werden soll:

```

class TemperatureSensors():
    # ...

    def get(self, nbtries = 3):
        # get all temperatures as array and update maxtemps
        # to avoid problems with read errors that stop a program,
        # reading is done with try - except and on error, -300 is returned
        self.temps = []
    
```

```
for i in range(0, self.nbsensors):
    a = self.addresses[i]

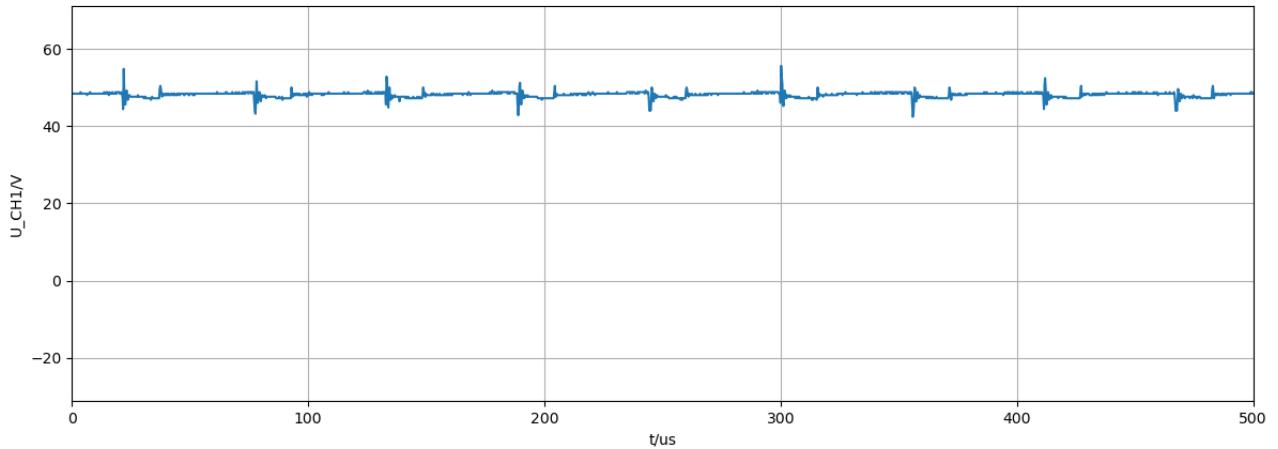
    for j in range(nbtries):
        try:
            t = self.ds_sensor.read_temp(a)
            break
        except:
            t = -300

    self.temps.append(t)

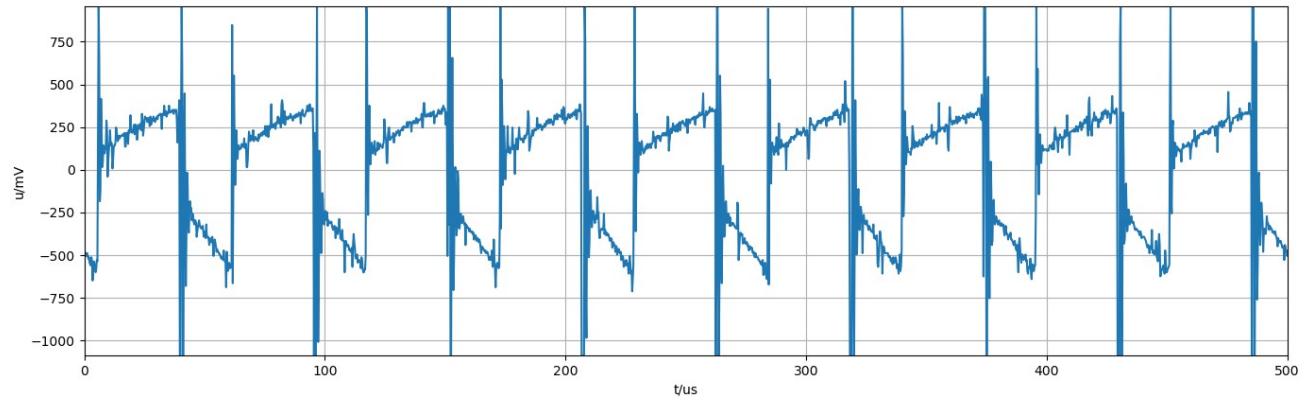
    if t > self.maxtemps[i]:
        self.maxtemps[i] = t
return self.temps
```

Welligkeit der Ausgangsspannung

Hier bei $P = 700W$:



Wechselspannungsanteil ca. 1V bei 300W:



•

52. Aufnahme einer Tracking-Kurve

Dies geht recht einfach über UART1 mit den Befehlen:

* = Programm unterbrechen

track()

Die Daten erscheinen in tabellarischer Form:

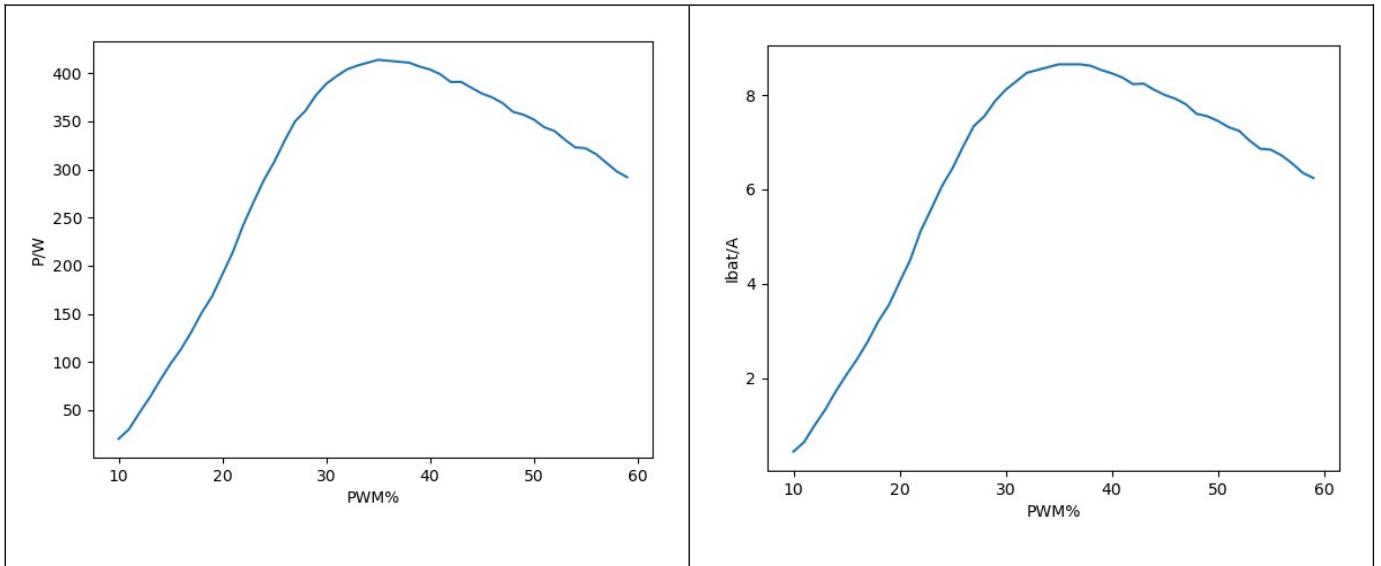
#	PWM	track	PWM%	V2/V	I2/A	P2/W	P2max
##	PWM	track	10	46.87	0.44	20	20
##	PWM	track	11	46.86	0.64	30	30

```

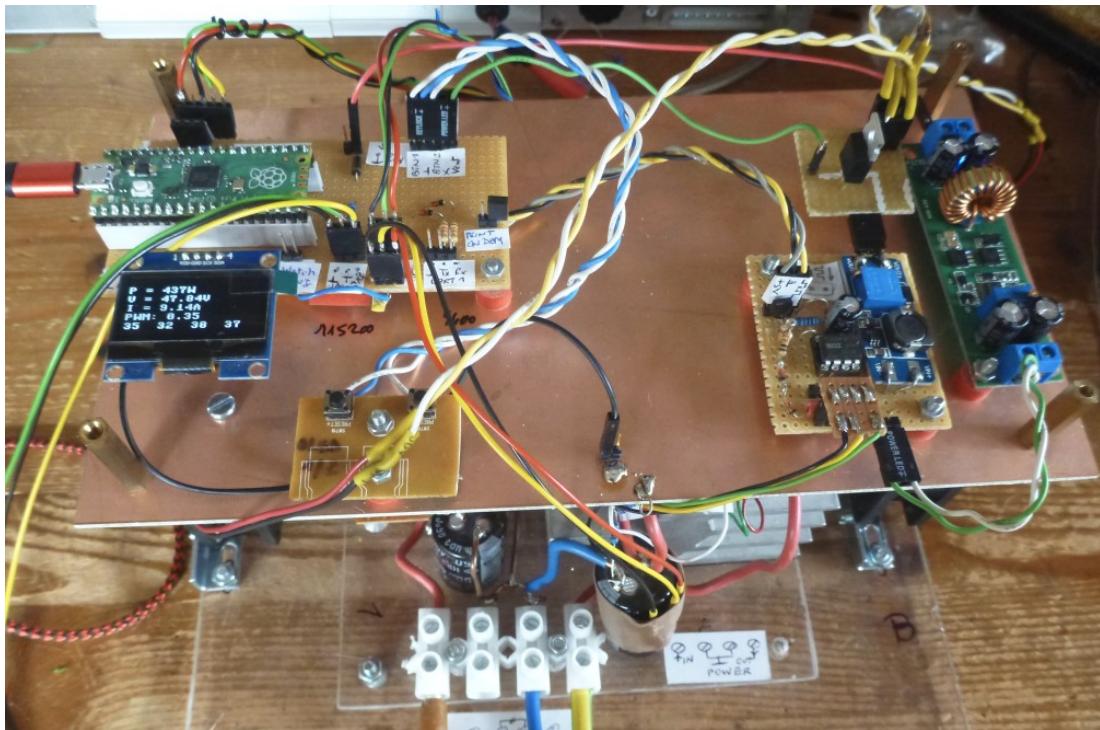
## PWM track    12    46.91      1.00      47    47
## PWM track    13    47.00      1.33      63    63

```

so dass man problemlos daraus Diagramme machen kann:



53. Die neue Ventilatorkonfiguration

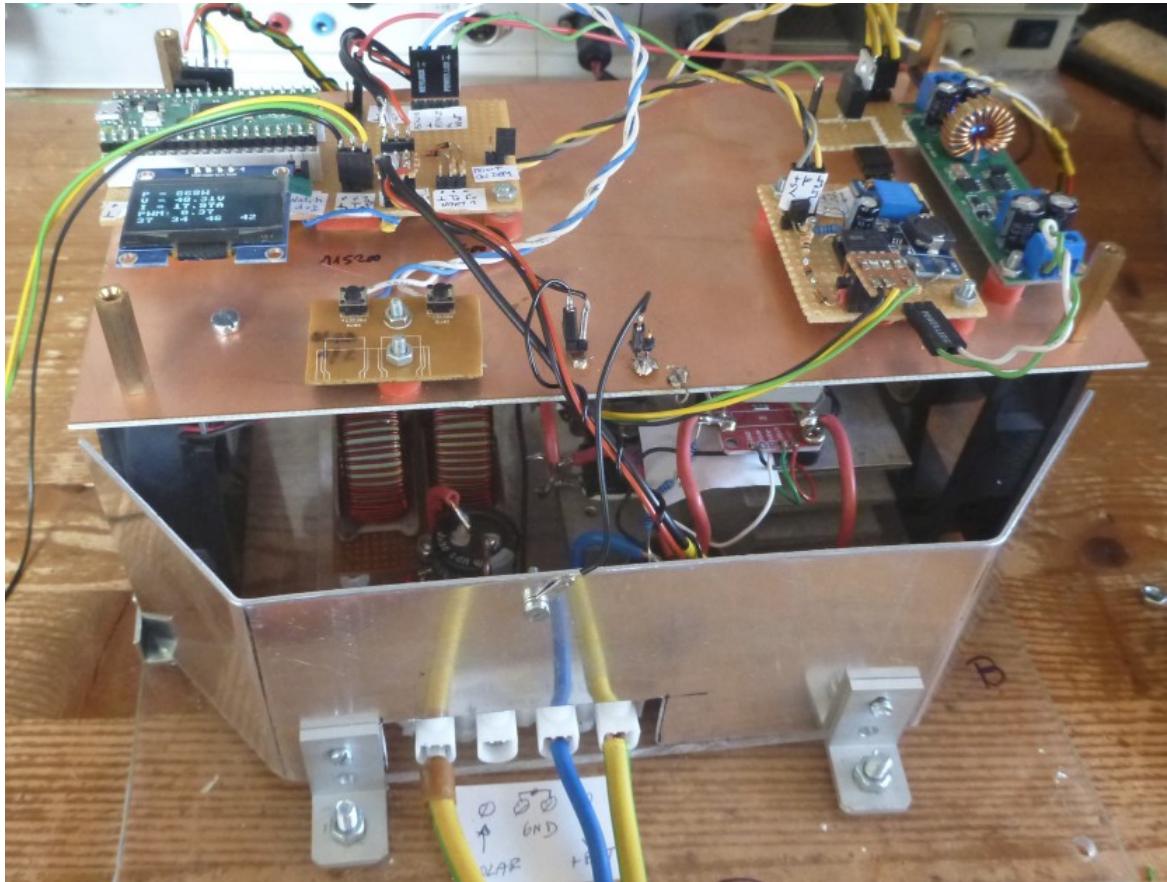


Im Bild durch die Controllerplatine verdeckt: links und rechts ein Lüfter. Diese blasen auf die Schaltung zu.

Die Konfiguration ist nicht optimal, die Temperatur steigt recht hoch an bei 700W...1000W.
Ursachen?

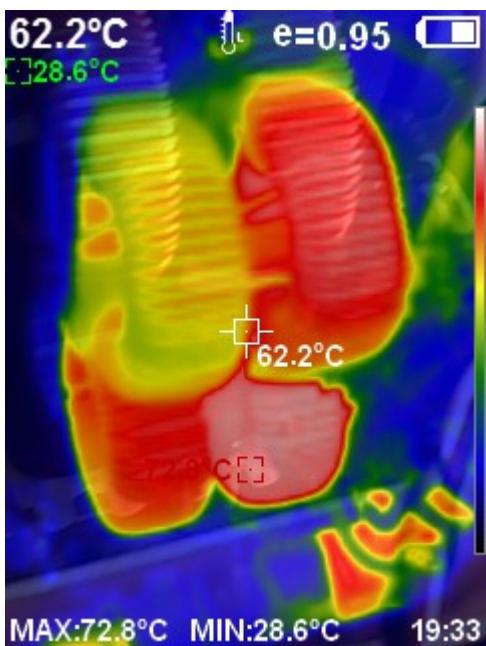
Luftwirbel? „Blinder Fleck“ in Achsenrichtung des Lüfters?

Idee: Blech zum Leiten der Luft in Richtung Komponenten



Auch diese Konfiguration löst das Problem nicht.

Wie die Wärmebildkamera zeigt, werden die Spulen auch nicht gleichmäßig gekühlt:



Stand 26.7.2025