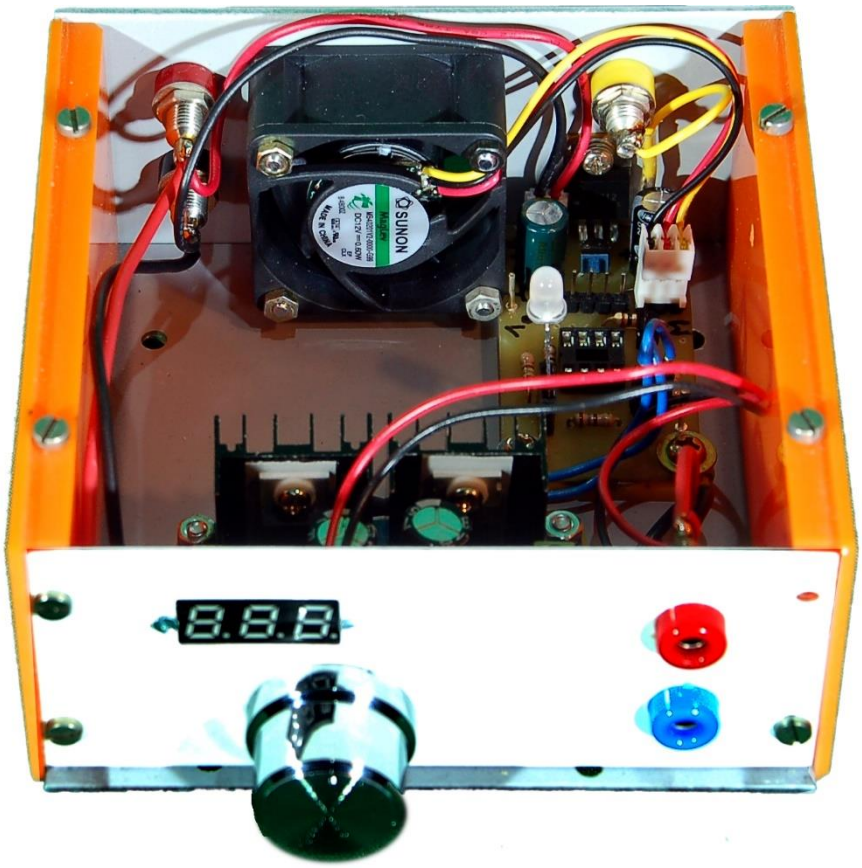


Abbildung 1: Buck-Converter_von vorne

Der Sommer steht vor der Tür und obwohl es in diesem Jahr nicht so tierisch heiß zu werden scheint, wie vor zwei Jahren, ist es dennoch schön, ein gekühltes Getränk griffbereit am Arbeitsplatz zu haben. Das Ziel ist also ein transportabler Getränkekühler. Als "Kühl-Aggregat" dafür habe ich Peltierelemente vorgesehen. Mit kleinen 4 x 4 cm in der Fläche und 3,7mm dünn kann man die Teile sowohl zum Kühlen aber auch zum Heizen (für was Heißes in der kalten Jahreszeit) verwenden. Die kleinen Tausendsassas haben nur einen kleinen Nachteil, sie müssen mit einigen Amperechen versorgt werden, damit sie ordentlich arbeiten. Die Stromstärke soll natürlich auch gemessen werden, autonom, vom Gerät. Dazu verwenden wir einen Halleffekt-Stromsensor, und für die Anzeige stellen wir einen ordentlich großen OLED-Zeichensatz aus einem Windowszeichensatz her. Messen und Anzeigen erledigt ein ESP8266 auf der Basis von MicroPython. Genau hier setzen wir im ersten Teil der neuen Reihe an. Und damit willkommen zu diesem Blog mit dem Thema

Eine Stromquelle für die Kaltmacher

Bevor wir loslegen, will ich Ihnen eine Vorschau darüber geben, was Sie insgesamt erwartet.

Im ersten Teil geht es um ein Netzteil, das genug Power für mehrere Thermoelemente mitbringt. Wir recyclen ein altes Computernetzteil und paaren es mit einer sehr effizienten Reglerschaltung, die bis zu 8 Ampere liefern kann. Der durch ein Poti steuerbare DC-DC-Wandler braucht bei so viel Power eine aktive

Kühlung, die wir aus einem 4 x 4 – cm Lüfter und einer Temperaturregelung aufbauen. Die Regelung sorgt dafür, dass der Lüfter nicht ständig saust.

In der nächsten Folge beschäftigen wir uns mit dem Peltierelement als Thermosensor. Aber wir werden damit nicht Temperaturen messen, sondern Wärmeströme. Damit kann man feststellen, welche Wärmemenge durch Fensterscheiben oder Wände hereinkommt oder den Raum durch dieselben verlässt. Als Messknecht wird ein ESP8266 dienen, den wir in MicroPython programmieren werden. So ein Teil ist nützlich für die Energieoptimierung von Zimmern, Heiz- und Kühlgeräten.

Schließlich verwenden wir die gleiche Art von Peltierelementen als Kühlaggregate. Der Getränkekühler kann mit unserer Reglerschaltung sowohl aus dem PC-Netzteil als auch aus dem Autobordnetz versorgt werden. Die Überwachung der Stromversorgung wird ein ESP32 übernehmen, der auch die Temperatur in der Kühlbox kontrolliert und über eine Relaiseinheit steuert.

Also lassen Sie uns beginnen. Diese Teile werden für die erste Folge benötigt.

	Stromversorgung
1	XH-M401 DC-DC Step Down Buck Converter XL4016E1,
1	0,28 Zoll Mini Digital Voltmeter Spannungsmesser mit 7-Segment Anzeige 2,5V - 30V
1	KY-013 Thermistor Temperatur Sensor Modul
1*	0,91 Zoll OLED I2C Display 128 x 32 Pixel
1*	NodeMCU Lua Amica Modul V2 ESP8266 ESP-12F für optionale Strommessung
1*	ACS712 20A Ampere Stromsensor Range Modul Current Sensor
1	Aktiver Miniventilator Kühl- Axial- Ventilator für Raspberry Pi oder Prozessor-Lüfter 4 x 4 cm
	Zweipunkt-Temperaturregler
1	PCB Board Set Lochrasterplatte oder eigenes PCB mit Herstellung nach Printvorlage
1	LM7812 / 1A
1*	LM78L05, falls ein Amica mit ins Rennen soll
1	BC337 NPN-Transistor mit Ic=500mA zur Schaltung des Lüfters
1	LM358 Operationsverstärker
1	IC-Fassung DIL8
1	Widerstand 1kΩ
1	Widerstand 1,5kΩ
1*	Widerstand 3,3kΩ
1	Widerstand 3,9kΩ
1	Widerstand 15kΩ
1	Widerstand 82kΩ
1	Widerstand 100kΩ
1	Trimmpoti 10-Gang 10kΩ*
5	Keramikkondensator 0,1µF
1	Elektrolytkondensator 470µF/16V
1*	Elektrolytkondensator 1000µF/6,3V, falls ein Amica mit ins Rennen soll

1	Diode 1N4148
1	Stiftleiste 10-polig
div.	Blech-, Kunststoff- oder Holzplatten für ein Gehäuse
5	4mm-Buchsen für Ein- und Ausgang (2 x rot, 2 x blau, 1 x gelb)
div	isolierte Litze für Verbindungen (>=1,5mm²)

* Diese Teile brauchen Sie für die optionale Strommessung

Der Buck Converter XL4016E1

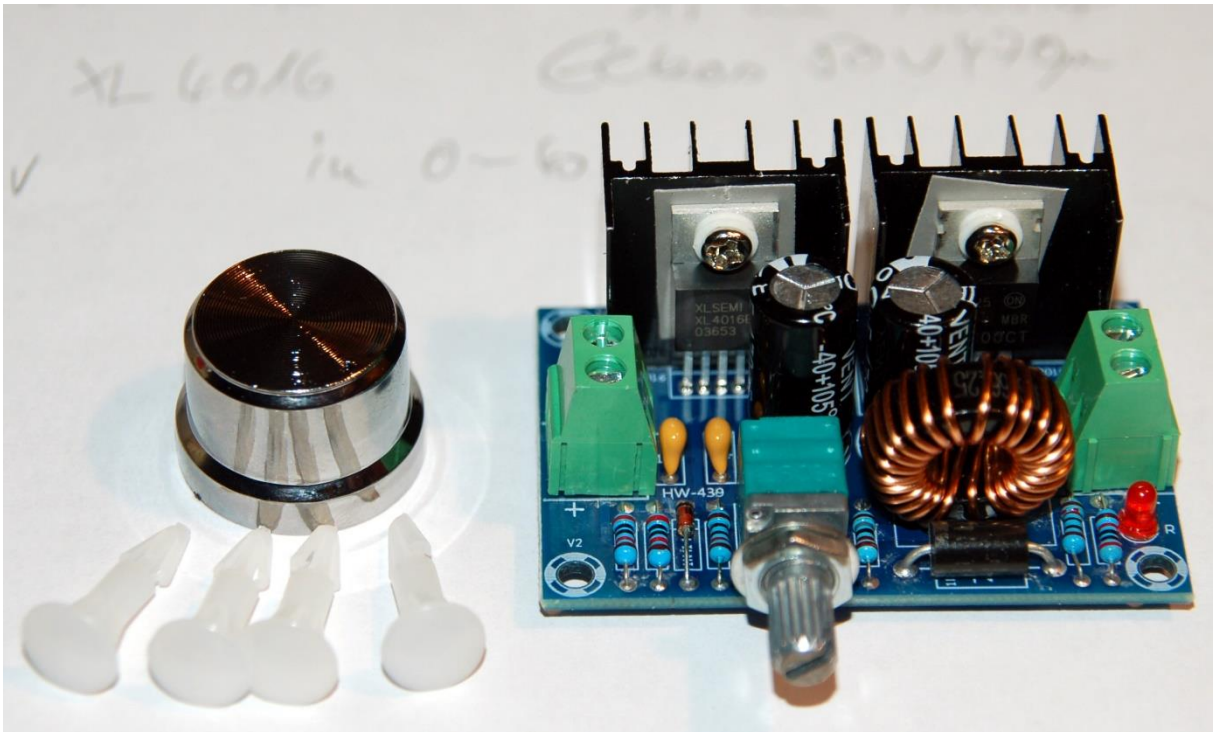


Abbildung 2: Buck-Converter

Der Converterchip kann laut Herstellerdatenblatt Eingangsspannungen bis 32V verdauen und bis zu 12A ausspucken. Eigene Messungen ergaben einen maximalen Wirkungsgrad der Schaltung von bis zu 88% bei höheren Stromstärken. Das kommt uns zugute, denn wir wollen das Teil ja gerade in diesem Bereich einsetzen. Auch der niedrige Dropout von 0,4V zwischen Ein- und Ausgangsspannung ist ideal. Beim Anschluss der Schaltung ist nur wenig zu beachten. Die Eingangsklemmen für Vcc und GND liegen in der Abbildung oben links, und der Plusanschluss ist bei Ein- und Ausgang nach vorne zum Betrachter.

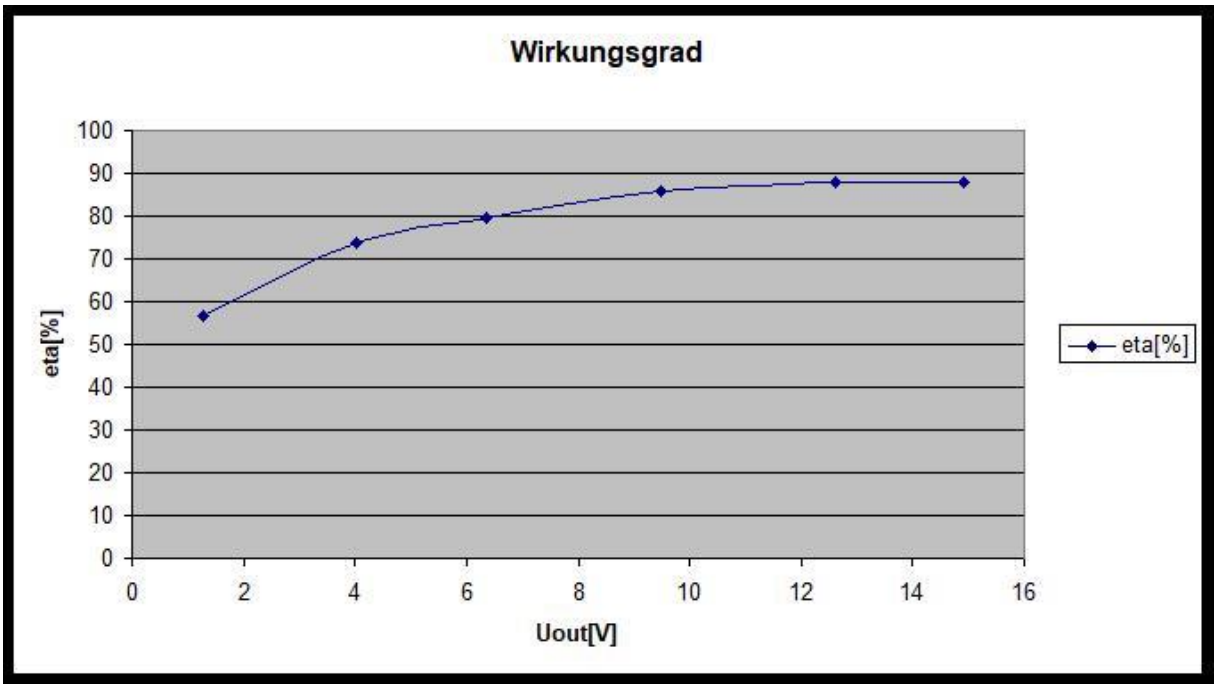


Abbildung 3: Wirkungsgrad bei 8A Output

Die minimal einstellbare Spannung beträgt 1,25V, bedingt durch die Referenzspannung des Schaltreglers. Unsere Spannungsanzeige setzt bei ca. 3,5V ein. Sie ist klein, arbeitet aber sehr genau. Damit möglichst viel Energie am Verbraucher ankommt, dürfen die Zuleitungen nicht zu dünn und zu lang gewählt werden. Das Poti hat zusätzlich einen Schalter, womit der Ausgang spannungslos geschaltet werden kann.

Die Ausgangsgleichspannung ist von einem leichten hochfrequenten Rauschsignal überlagert, was für unsere Anwendung aber belanglos ist. Lediglich für die direkte Versorgung von Controlleranwendungen ist eine zusätzliche Filterstufe hilfreich.

Der Energielieferant für den Regler

Nun ist das Modul ja nur ein Umsetzer für Gleichspannung, aber keine wirkliche Stromquelle. Wir brauchen also eine Quelle, die neben einer Spannung von 12V aufwärts auch satte Stromstärken von bis zu 12A liefern können sollte. Ohne große Veränderungen am Gerät kommen hier eine Autobatterie oder ein altes Computernetzteil in Frage. Erstere hat halt; außer bei laufendem Motor, nur begrenze Ressourcen, während das zweitere aus dem Netz versorgt wird. Nachdem wir für die Bedienung unserer Thermoelemente Spannungen bis ca. 12V (15V maximal) benötigen, sind beide Versionen ausreichend.

Wer höhere Spannungen, bis 32V, eingangsseitig verarbeiten möchte, braucht entsprechende Trafos, einen Gleichrichter und einen satten Siebelko. Soviel nur als Ausblick auf eventuelle Erweiterungen und die Anwendbarkeit des Reglermoduls für andere Zwecke. Echte Transformatoren für Halogenbeleuchtungen sind dafür ein guter Tipp. Zur Spannungserhöhung auf 24V AC kann man 2 gleiche Trafos in Reihe schalten und erreicht damit Ausgangsgleichspannungen von bis zu ca. 33V. Auch Trafos aus ausgedienten Overheadprojektoren liefern 24V/10A. Hier aber aufpassen,

dass die Sekundärspule wirklich **galvanisch von der Netzwicklung getrennt** ist. Das sollte unbedingt ein Elektrofachmann prüfen und bestätigen. **Wenn die beiden Wicklungen leitend verbunden sind (Spartrafo), besteht bei Berührung Lebensgefahr!**

Die aktive Kühlung des Reglers

Die maximal umgesetzte Leistung für unsere geplanten Anwendungen beträgt 12V • 8A = 96W. Das sind 88% der zugeführten Leistung, von der 12% als Wärme abgeführt werden müssen, und das sind ca. 13 Watt. Dass die kleinen Kühlkörper auf dem Board das, noch dazu in einem Gehäuse, nicht schaffen können, ist offensichtlich. Deshalb brauchen wir eine aktive Kühlung durch virtuellen Fahrtwind. Den erzeugt der Lüfter für den Raspi oder einfach ein kleiner PC-Lüfter aus einem "steinzeitlichen" PC aus der Zeit, als diese Teile grade einmal 4 mal 4 cm groß waren (der Lüfter natürlich, nicht der PC).

Ständig laufende Lüfter sorgen aber für Unruhe, akustisch und bei den Luftmassen am Arbeitstisch. Deshalb musste eine Temperaturregelung her. Von den drei Lösungen stelle ich Ihnen heute die beiden praktikabelsten vor.

A	Ein AT-Tiny13A dient als Regler. Die Schwellen sind per Programm voreinstellbar.
B	Ein Schmitt-Trigger mit dem dualen Operationsverstärker LM358 schaltet den Motor temperaturabhängig.
C	Die Nobellösung mit einem Amica V2 liefert den Wert der Stromstärke am Ausgang der Reglerschaltung und übernimmt optional das Digitalsignal der Lösung B.

Lösung A habe ich für den Blog verworfen, weil einfach zu viele Voraussetzungen erfüllt sein müssten, um sie umzusetzen. ATMEL-Software, Programmieren in AVR-Assembler, das Brennen des ATTiny13 weicht von der Arduinomethode ab, abweichender SPI-Stecker und ein passender USB-Umsetzer wären nötig und allein schon Thema für einige Blogs. Wer dennoch Interesse an der Lösung A hat, sei auf diese Adresse verwiesen, von welcher der Ansatz B schaltungstechnisch abgeleitet wurde.

Also Lösung B, als solide Basisfunktion erweiterbar durch einen ESP8266 in jedweder Ausführung als Nobelversion C. Weil dieser nur einen Analogeingang hat, brauchen wir den LM358 nach wie vor als Temperaturarbeiter, der ein fertiges Digitalsignal liefert. Natürlich gäbe es als Alternative für beide zusammen den ESP32 oder einen Analogmultiplexer, der über I2C an den ESP8266 angeschlossen würde. Der könnte dann Spannung, Stromstärke, Leistung, Energie und Temperatur... - OK, Schluss - bleiben wir auf dem Pflaster mit den Lösungen B und C.

Der Temperatursensor

Einen ESP32 für den Einsatz als Messknecht haben wir eben als überzogen ausgeschlossen. Ebenso sieht es mit dem Sensor aus. Ein DS18B20 von Dallas wäre zwar sogar vom Amica V2 noch zu bewältigen, aber wir wollen ja keine

Temperaturen messen, sondern lediglich feststellen, wann ein Motor ein und wieder ausgeschaltet werden muss. Dafür reicht ein einfacher NTC-Widerstand. **N**egative **T**emperature **C**oefficient bedeutet, dass der Widerstandswert mit steigender Temperatur sinkt und zwar annähernd exponentiell. In Reihe mit einem Festwiderstand R1 von 100 kΩ liefert uns diese Spannungsteilerschaltung eine temperaturabhängige Spannung am Punkt A. Sie finden die Stelle im Schaltplan des Reglers ein Stück weiter unten im Text. Ein nachfolgender Schmitt-Trigger, den wir aus der einen Hälfte des Operationsverstärkers LM358 aufgebaut haben, vergleicht diese Spannung mit einer Referenzspannung, die uns die andere Hälfte zusammen mit dem Trimmerpoti am Punkt B liefert.

Zwar gäbe es ein komplettes Modul, das bereits mit einem Comparator (Vergleicher) LM393 ausgestattet ist, das aber einen entscheidenden Nachteil hat. Das ist der LM393, der als empfindlicher Vergleicher ohne Hysterese arbeitet. Das würde hier dazu führen, dass der Lüfter schon bei kleinsten Temperaturschwankungen schnell hintereinander aus und eingeschaltet wird. Ein ordentlicher Kühleffekt könnte damit nicht zustande kommen. Die Abbildungen 4 und 5 demonstrieren das.

Abbildung 4: Schaltverhalten Comparator

Abbildung 5: Schaltverhalten Comparator Zoom

Weil die Spannungsänderung an A nur langsam erfolgt, wäre das Flattern um den Schaltpunkt Dauerzustand. So was können wir nicht brauchen.

Ich habe deshalb den KY-013 ausgewählt, der ohne Comparator auf seinem Platinchen wohnt. Dennoch müssen wir beim Anschluss trickreich vorgehen, denn auf dieser Platine ist ein Festwiderstand von 10kΩ verbaut, den wir, zumindest schaltungstechnisch, umgehen müssen.

Die Gesamtschaltung sehen Sie hier. Dargestellt sind die Originalteile und deren Verbindungen. Zum Temperaturregler kommen wir anschließend. Wie immer gibt es einen besser lesbaren Plan als PDF-Datei zum Download.

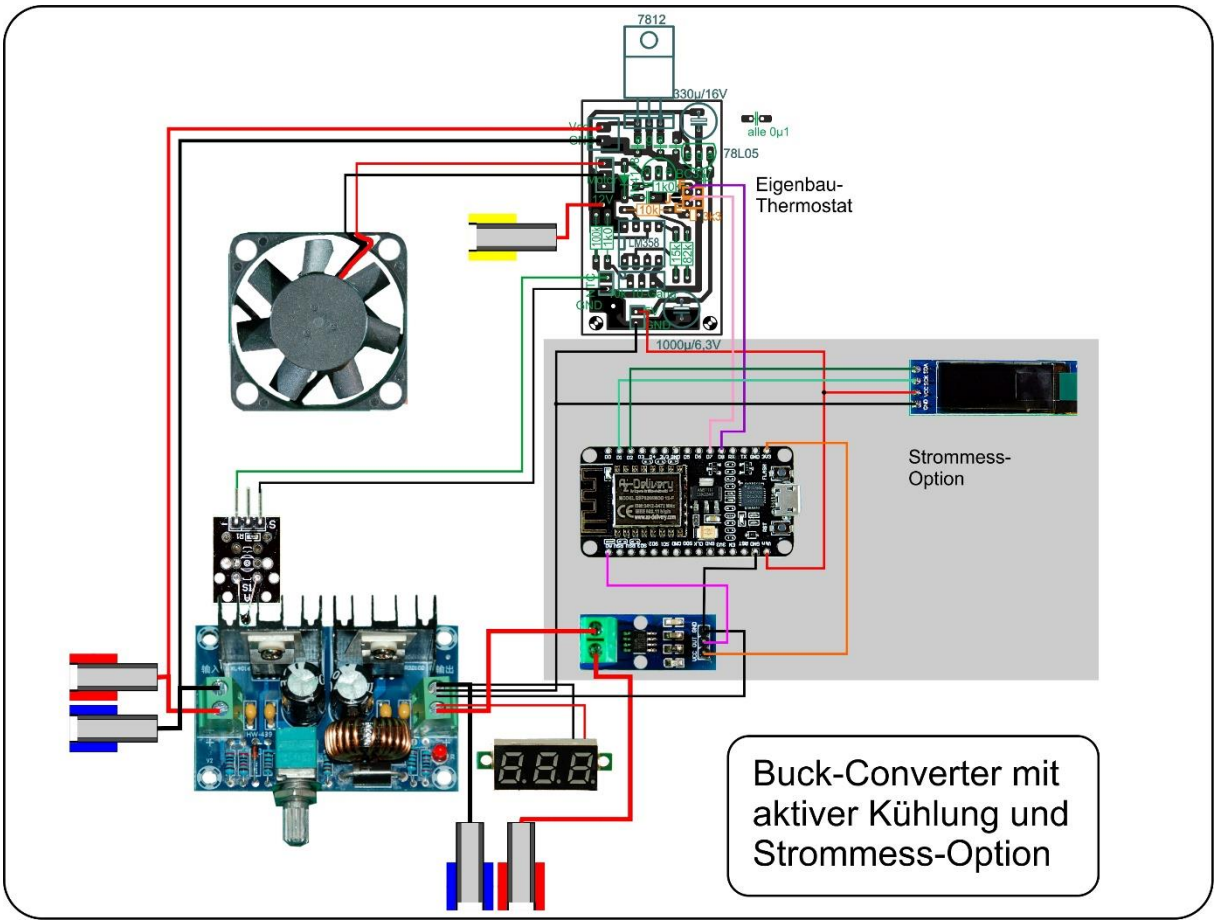


Abbildung 6: Schaltung_Converter

Unser Thermostat bezieht seine Energie auch vom Anschluss der externen Spannungsquelle. Die Schaltung liefert außer dem Temperatursignal an E noch +12V für die Versorgung externer Lüftermotoren und die +5V für einen optionalen ESP8266. Natürlich schaltet es auch den Lüftermotor. Das sehr genaue digitale Voltmeter zeigt die Ausgangsspannung des Aufbaus.

Hier kommt auch noch der Schaltplan für den Thermostaten. Auf meinem Platinenentwurf sind die Punkte E und F durch den Jumper J noch verbunden. Dadurch wird der Schalttransistor BC337 unmittelbar angesteuert. Wird J geöffnet, dann kann das Signal an E dem ESP8266 zugeführt werden. Das Schaltsignal des Controllers wird dann, wie in der obigen Gesamtansicht, am Punkt F eingespeist. Der ESP8266 entscheidet dann, wann der Lüfter wie lange (nach-) laufen soll. Der LM358 wird auch mit Vcc = 12V versorgt. Der LM78L05 kann 100 mA liefern.

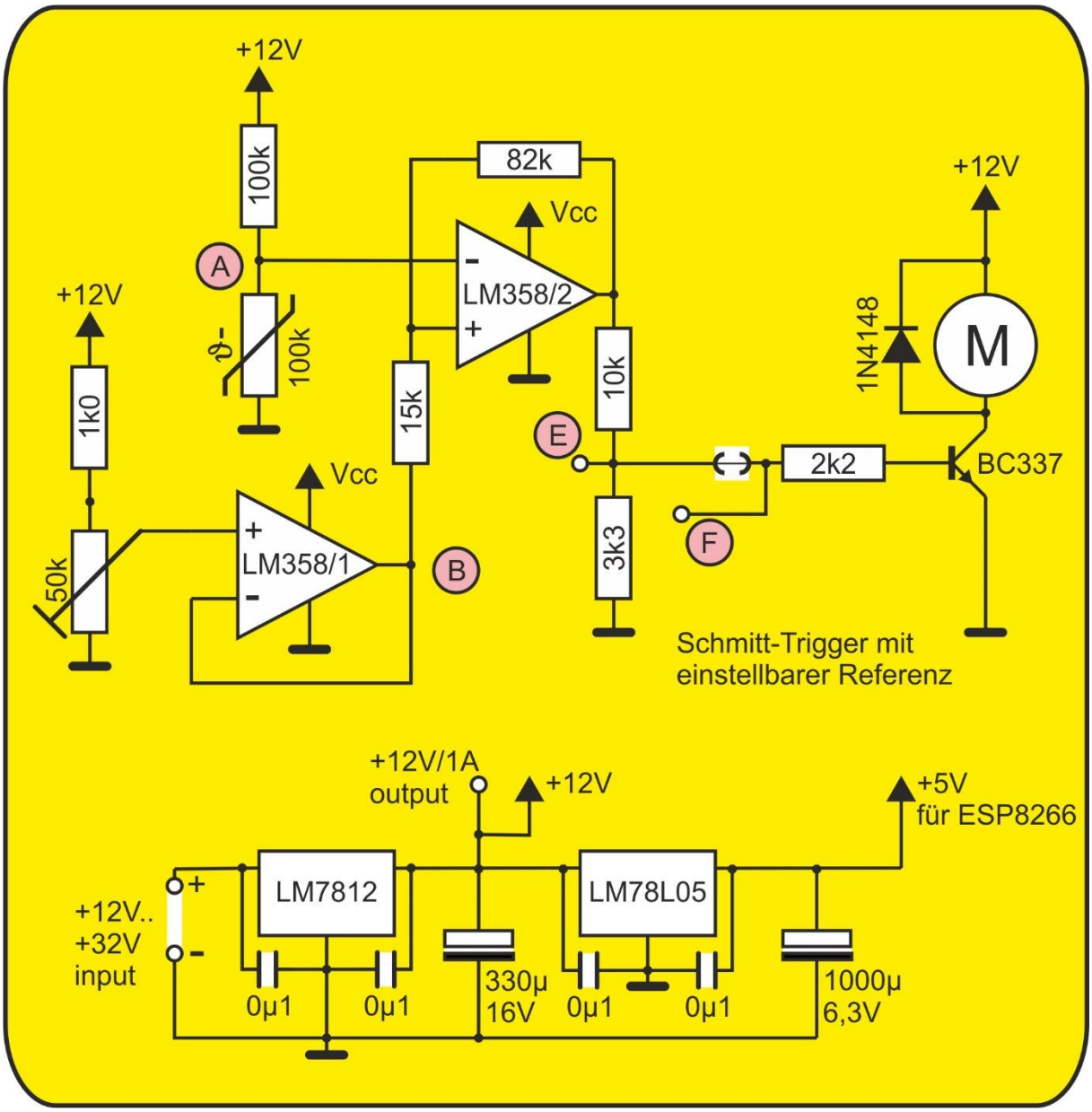
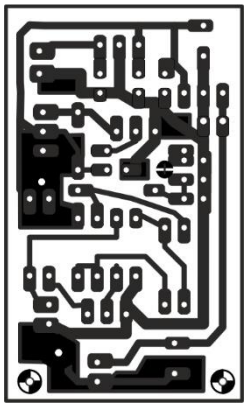


Abbildung 7: Thermostat mit LM358

Der NTC-Widerstand hat bei 25 °C einen Wert von um die 100kΩ. Mit einem Festwiderstand von ebenfalls 100kΩ legen wir die Spannung an A auf diese Temperatur fest. Um den unbrauchbaren Festwiderstand von 10kΩ auf der NTC-Platine zu umgehen, verwenden wir von der NTC-Platine nur die Anschlüsse S und GND, dazwischen befindet sich der NTC. Unser Festwiderstand von 100kΩ liegt auf der Reglerplatine. Über die Herstellung der Schaltung gibt es gleich Hilfen und Tipps. Wie das Ganze funktioniert besprechen wir danach.

Herstellen der Temperaturreglerschaltung

Die Schaltung kann man auf einer Lochrasterplatine herstellen. Die Leiterbahnen werden durch Lötbrücken zwischen den Pins oder durch kurze Drähtchen hergestellt.



Einfacher ist der Aufbau allerdings, wenn Sie sich eine gedruckte Schaltung herstellen. Das geht zum Beispiel mit Hilfe eines Laserausdrucks der Vorlage, den Sie dann auf eine einseitig kaschierte Platine legen und mit einem Bügeleisen auf höchster Stufe auf die Kupferschicht übertragen.

Abbildung 8:Thermoschalter_für_Lüftermotor_Printvorlage

Wie das genau geht, habe ich hier beschrieben. Danach wird die Platine geätzt, gebohrt, zugeschnitten und mit einem Lösemittel (z. B. Aceton) gereinigt. Abschließend folgt die Bestückung nach folgendem Plan.

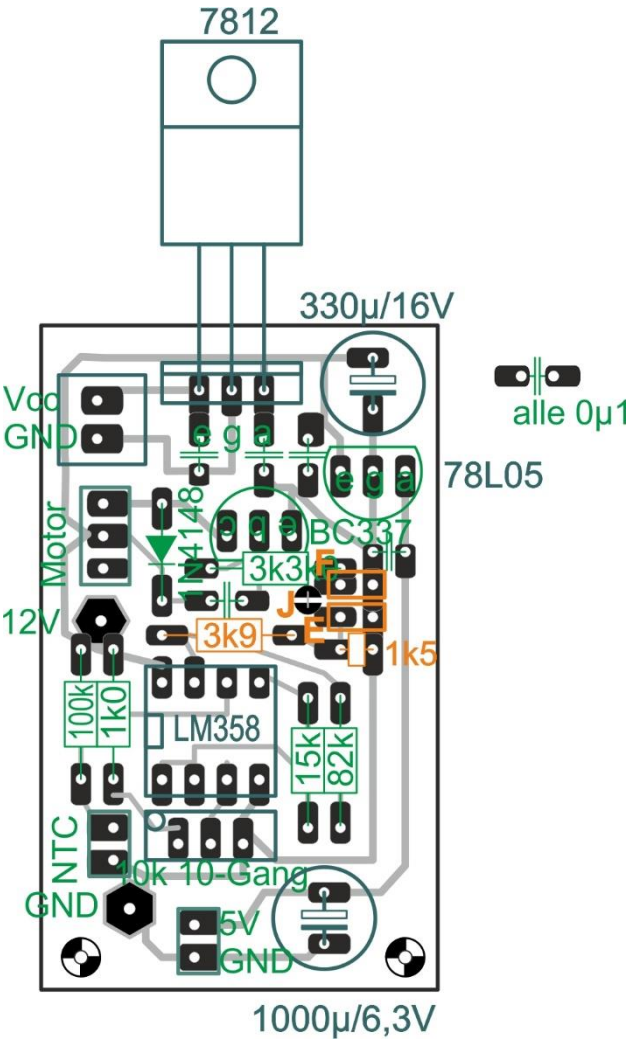


Abbildung 9: Thermoschalter_für_Lüftermotor_Bestückung

Wie arbeitet die Schaltung?

Mit dem Trimmpoti auf unserer Platine wird der Arbeitspunkt des Thermostaten eingestellt. Der 82kΩ-Widerstand sorgt durch die Mitkopplung des Ausgangspegels auf den positiven Eingang des Operationsverstärkers für eine Hysterese. Das bedeutet, dass bei einer niedrigeren Schwellspannung am Punkt A (BY=4,72V, gelbe Kurve) der Ausgang (blaue Kurve) auf einen Wert um die 10V springt und bei einem höheren Eingangswert (AY=6,74V) zurück auf 0V. Die folgende Grafik zeigt das. Zur Demonstration wurde an den Punkt A anstatt des NTC-Spannungsteilers eine Sinusspannung angelegt, um für die Darstellung ein periodisches Signal zu erhalten. Die Mitkopplung führt nicht nur zu einem verschobenen Schaltverhalten, sondern auch zu eindeutigen, scharfen Schaltflanken. Der Vergleich mit den Abbildungen 4 und 5 zeigt das eindeutig (Kurvenfarben sind vertauscht!)

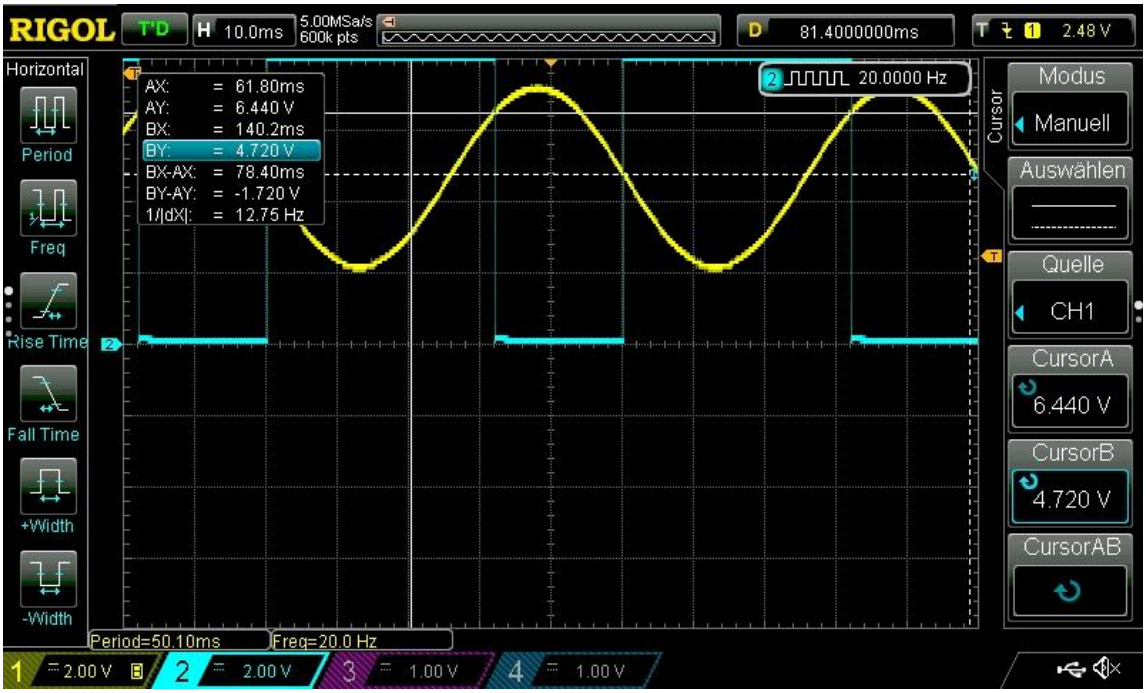


Abbildung 10: Schaltstufen und Hysterese2

Der NTC-Wert sinkt mit steigender Temperatur. Infolge sinkt auch die Spannung am Punkt A in der Schaltung. Wird die Schaltschwelle von 4,72V erreicht, kippt der Ausgang des Operationsverstärkers auf 10V, der Transistor wir leitend und der Motor läuft an. Durch die Kühlung sinkt die Temperatur der Kühlbleche, der NTC-Wert und die Spannung an A steigen. Mit Erreichen der oberen Schwellenspannung von 6,74V kippt der Ausgang auf 0V, der Transistor sperrt und der Motor stoppt. Durch die Hysterese wird der Kühlkörper quasi unterkühlt und damit die nächste aktive Phase des Lüfters hinausgeschoben. Weil die Schwellenspannungen weit auseinander liegen, stellt sich am Schaltpunkt auch kein Flattern ein, das auch hochfrequente Störungen verursachen kann.

Der Bereich der beiden Schwellenspannungen kann durch die Referenzspannung vom Trimmpoti verschoben werden. Der Temperaturbereich wird damit angepasst. Die eine Hälfte des Operationsverstärkers macht diese Referenzspannung vom Trimmer niederohmig am Punkt B verfügbar, sodass sie nicht rückwirkend durch die Schaltvorgänge beeinflusst werden kann.

Die optionale Stromanzeigefunktion mit dem ESP8266

Die Messung der Ausgangsstromstärke passiert mit dem Hall-Sensor Modul ACS712 dessen Stromsensibelchen ein ACS712T-ELC20A-Chip ist. Abweichend vom Datenblatt habe ich eine Empfindlichkeit von 60mV pro Ampere gemessen. Der Messbereich umfasst +/-20A.

Bei der Messung der Stromstärke mit einem Hallsensor wird der Strom nicht durch den Sensor geleitet, sondern an ihm vorbei. Das Magnetfeld des elektrischen Stroms durchsetzt den Sensor und bewirkt dort den Aufbau einer Spannung, die zur Stromstärke direkt proportional ist. Der "Hall" im Sensornamen hat übrigens nichts mit Echo zu tun, sondern geht auf Edwin Hall zurück, nach dem der Effekt benannt ist.

Unser Chip enthält außer dem Hallsensor selbst, noch einen Verstärker und eine Pegelanpassung, die den Nullpunkt der Strommessung auf den Wert der halben Versorgungsspannung des Chips legt. Das sind bei uns 3,30V : 2 = 1,65V. Pro Ampere wird diese Spannung um 60mV erhöht, oder, wenn der Strom in die Gegenrichtung fließt, erniedrigt. Das MicroPython-Programm im ESP8266 weiß das und rechnet die Spannungsdifferenz in die Stromstärke um. Leider ist die Linearität des ESP8266 nicht besonders gut, sodass trotz der Genauigkeit des ACS712, die Stromstärke mit einem Fehler von bis zu 10% belastet ist. Mit einem separaten Wandler, etwa vom Typ ADS1115, könnte man exaktere Ergebnisse erhalten. Allerdings stellt sich dann wieder die Frage der Verhältnismäßigkeit.

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
Links zu den Programmen und Modulen finden Sie im Text.

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.

Anzeige in großen Ziffern am OLED-Display.

Der Vorteil eines OLED-Displays ist eindeutig durch die flexible Darstellung von Text und Grafik gegeben. Wir nutzen das aus, um die Anzeige der Stromstärke nicht mit den popligen 10x8-Pixel-Zeichen, sondern mit ordentlich lesbaren Ziffern umzusetzen. Für diesen Zweck werden wir einen Windows-Zeichensatz in einen OLED-Zeichensatz portieren und daraus die benötigten Zeichen in ein Modul extrahieren. Beides passiert auf der Basis von CPython und MicroPython. CPython wurde zusammen mit Thonny installiert, denn Thonny ist in Python geschrieben und braucht die CPython-Umgebung, damit es überhaupt funktioniert. Natürlich können wir in Thonny (oder einem anderen Editor) auch CPython-Programme schreiben und dann von der Powershell aus starten. Genau das werden wir tun, sobald unser Zeichensatz umgewandelt ist.

Windows-TTF-Zeichensatz clonen

Um einen Vektorzeichensatz von Windows ins Pixelformat zu übertragen, benutzen wir eine Freeware von Peter Hinch, die der MIT-Licence unterliegt. Sie heißt micropython-font-to-py und wird als ZIP-Datei von Git-Hub heruntergeladen. Für die Installation habe ich das Verzeichnis _fonts im Rootverzeichnis meiner Festplatte angelegt. Das und alles Weitere funktioniert übrigens auch zusammen mit einem USB-Stick.

Die Datei **micropython-font-to-py-master.zip** habe ich in **F:_fonts** gespeichert und auch dorthin entpackt. Das entstandene Verzeichnis habe ich in **font2py** umgetauft, ich mag kürzere Pfadnamen. Wechseln wir jetzt in dieses Verzeichnis. Die beiden Pythonprogramme **font_to_py.py** und **font_test.py** werden wir in Kürze benutzen. Zuvor legen wir noch ein Verzeichnis **quellen** an.

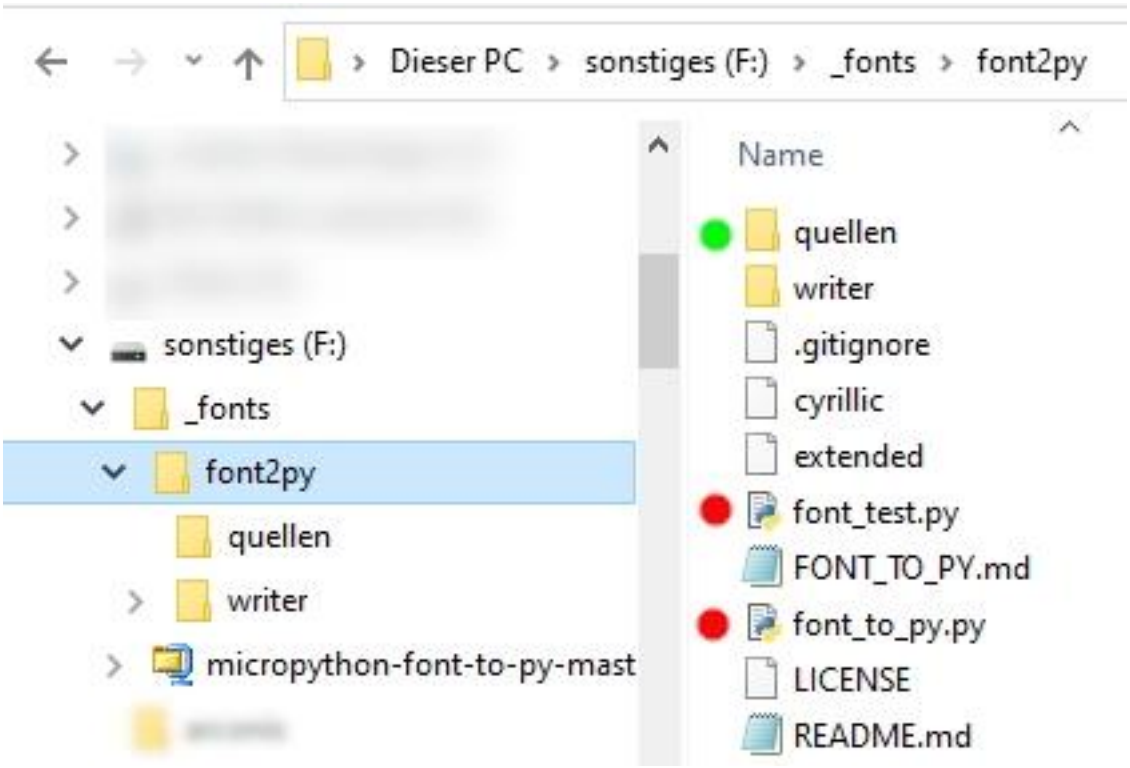


Abbildung 11: Charset01

Öffnen Sie nun den Schriftenordner von Windows, **C:\Windows\Fonts**. Suchen Sie nach einer möglichst klaren Schriftform und kopieren Sie die Datei in das Verzeichnis quellen. Ich habe hier **impact.ttf** gewählt.

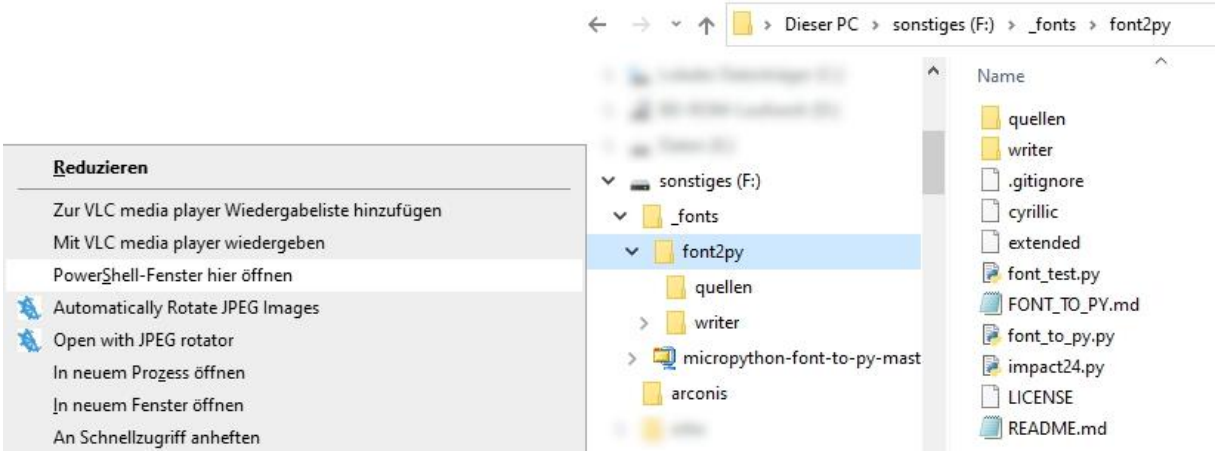


Abbildung 12: Charset02

Die nächsten beiden Schritte erfolgen in einem Powershell-Fenster. Wir öffnen das Contextmenü mit gedrückter **Umschalttaste** (Shift) und einem **Rechtsklick** auf den Ordner **font2py** und wählen **Powershell-Fenster hier öffnen**. Ein **dir** zeigt uns, dass wir im richtigen Verzeichnis sind. Dann setzen wir folgendes Kommando ab:

.\font_to_py.py F:_fonts\font2py\quellen\impact.ttf 24 -f impact24.py

Bitte beachten Sie, dass der Befehl mit einem ".\" eingeleitet werden muss, damit er in der gewünschten Pfadumgebung ausgeführt wird.

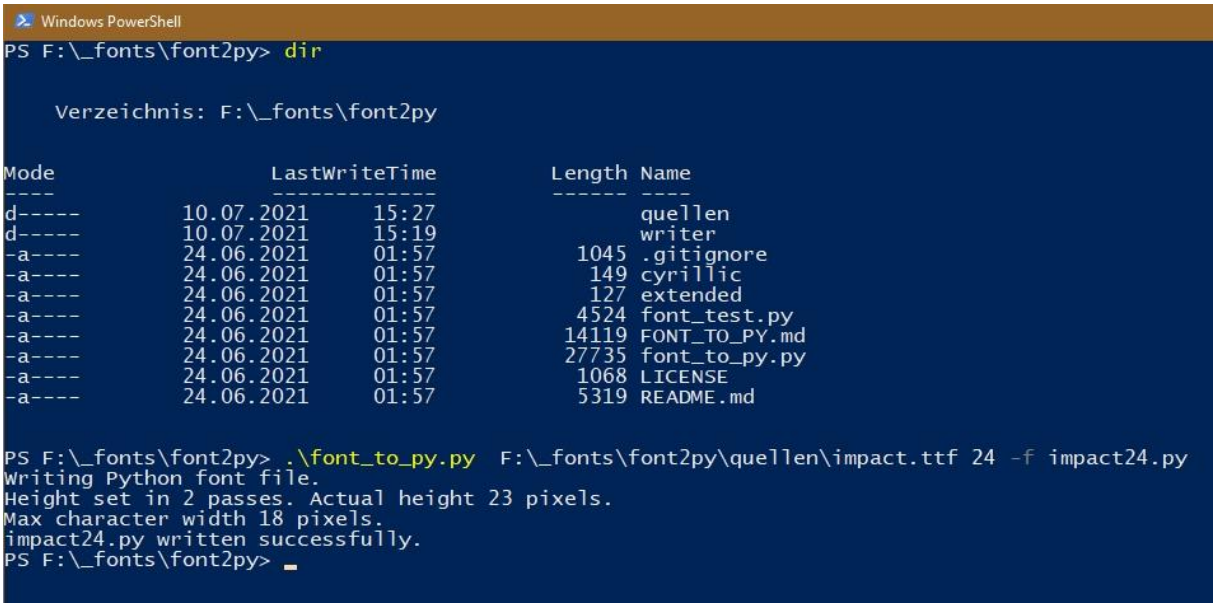


Abbildung 13: Charset03

Soeben haben wir den Zeichensatz Impact mit 24 Pixel Zeichenhöhe (inclusive Unterlängen) in eine Python-Datei umgewandelt. Der Schalter **-f** sorgt dafür, dass zunächst alle Zeichen auf gleiche Breite gebracht werden, auch die von

Proportionalzeichensätzen. Das ist wichtig, denn sonst gibt es beim Zerteilen der Strings im nächsten Schritt Zeichensalat.

Wir können Speicherplatz sparen, wenn wir vom gesamten Zeichensatz nur die Zeichen verfügbar machen, die wir wirklich brauchen. Diese Zeichen geben wir zwischen Anführungszeichen an und leiten die Ausgabe in die Datei **impact24.txt** um.

.\font_test.py impact24 "0123456789,-AVW" > impact24.txt

```
PS F:\_fonts\font2py> .\font_test.py impact24 "0123456789,AVW" > impact24.txt
>>
PS F:\_fonts\font2py> dir


    Verzeichnis: F:\_fonts\font2py


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        10.07.2021     15:27                quellen
d-----        10.07.2021     15:19                writer
d-----        10.07.2021     15:53                __pycache__
-a----        24.06.2021     01:57           1045 .gitignore
-a----        24.06.2021     01:57            149 cyrillic
-a----        24.06.2021     01:57            127 extended
-a----        24.06.2021     01:57           4524 font_test.py
-a----        24.06.2021     01:57          14119 FONT_TO_PY.md
-a----        24.06.2021     01:57          27735 font_to_py.py
-a----        10.07.2021     15:43          25211 impact24.py
-a----        10.07.2021     15:53          11950 impact24.txt
-a----        24.06.2021     01:57           1068 LICENSE
-a----        24.06.2021     01:57           5319 README.md
```

Abbildung 14: Charset04

Der Inhalt dieser Datei sieht etwa so aus.

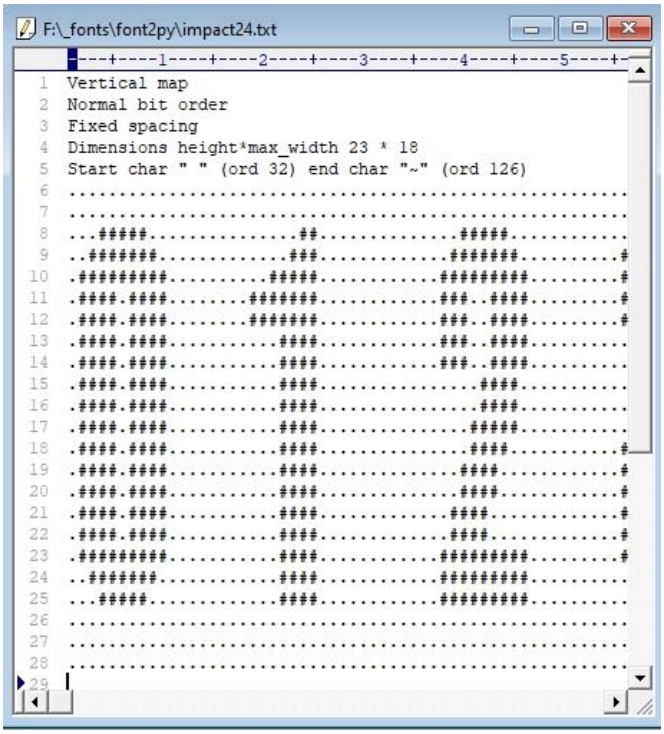
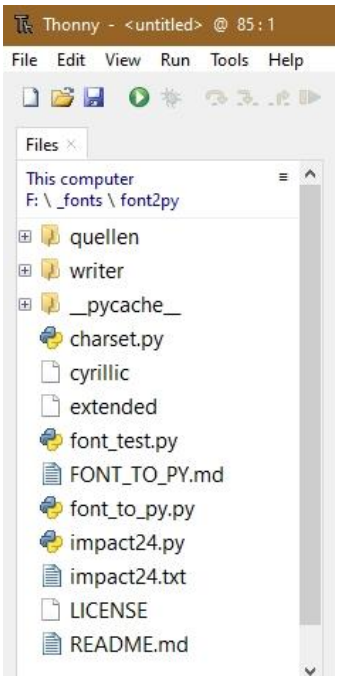


Abbildung 15: Charset-Text

Auf dieser Grundlage erzeugen wir nun abschließend ein Modul, das wir in Programme integrieren können, welche OLED-Displays nutzen.

Starten wir Thonny und wechseln wir in das Arbeitsverzeichnis **F:_fonts\font2py**.



Wir erzeugen ein neues Programmfenster via **File – New**, kopieren den Text des Programmlistings von text2py.py dort hin und speichern die Datei unter diesem Namen ab, nachdem die Variablen pfad, datei und chars angepasst sind. chars wird derselbe Stringinhalt zugewiesen, wie er beim Aufruf von font_test.py verwendet wurde.

Hinweis:
Bitte beachten Sie, dass der "\" in der Pfadangabe doppelt angegeben sein muss. Dadurch wird die spezielle Bedeutung des "\" als Escapezeichen in Zeichenketten aufgehoben.

```
# text2py.py

import sys

pfad="F:\\_fonts\\font2py\\"
datei="impact24"
chars="0123456789,-AVW"

def readln():
    s=""
    while 1:
        c=f.read(1)
        a=ord(c)
        if a>10 and a<126:
            s+=c
```

```
        elif a==10:
            for i in range(3):
                z=f.read(1)
            break
    return s



f=open(pfad+datei+".txt", "r")

# Kopfdaten einlesen und Zeichenhöhe und -breite holen
for j in range(3):
    zeile = readln()
    print(zeile)
zeile=readln()
pos1=zeile.find("width ")
pos2=zeile.find("*",pos1)
height=int(zeile[pos1+5:pos2])
width=int(zeile[pos2+1:])
print(height,width)
zeile=readln()

# jetzt die Zeicheninfo einlesen
a=[]
for z in range(height):
    a.append(readln())

f.close()

print("*****************************************")

f=open("charset.py","w")
print('chars="'+chars+'"')
print('height='+str(height))
print('width='+str(width))
print('number=[')
f.write('chars="'+chars+'"\n')
f.write('height='+str(height)+'\n')
f.write('width='+str(width)+'\n')
f.write('number=[\n')


for i in range(len(chars)):
    last=0
    block=[]
    b=len(a[1][i*width:(i+1)*width])
    for z in range(height):
        row=a[z][i*width:(i+1)*width]
        new=row.rfind("#")
        if new>last:
            last=new
    last=(last+2 if last < b else last)
```

```
    for z in range(height):
        row=a[z][i*width:(i+1)*width]
        row=row[:last]
        block.append(row)
    print('    ('+str(last)+',  #  Zeichen: '+chars[i])
    f.write('    ('+str(last)+',  #  Zeichen: '+chars[i]+'\n')
    for z in block:
        z=z.replace(".","0")
        z=z.replace("#","1")
        print('     0b'+z+',')
        f.write('     0b'+z+',\n')
    print('    ),')
    f.write('    ),\n')
print(']')
f.write(']\n')

f.close()
```

Das Programm starten wir nicht in Thonny, sondern auf der Powershell.

.\text2py.py

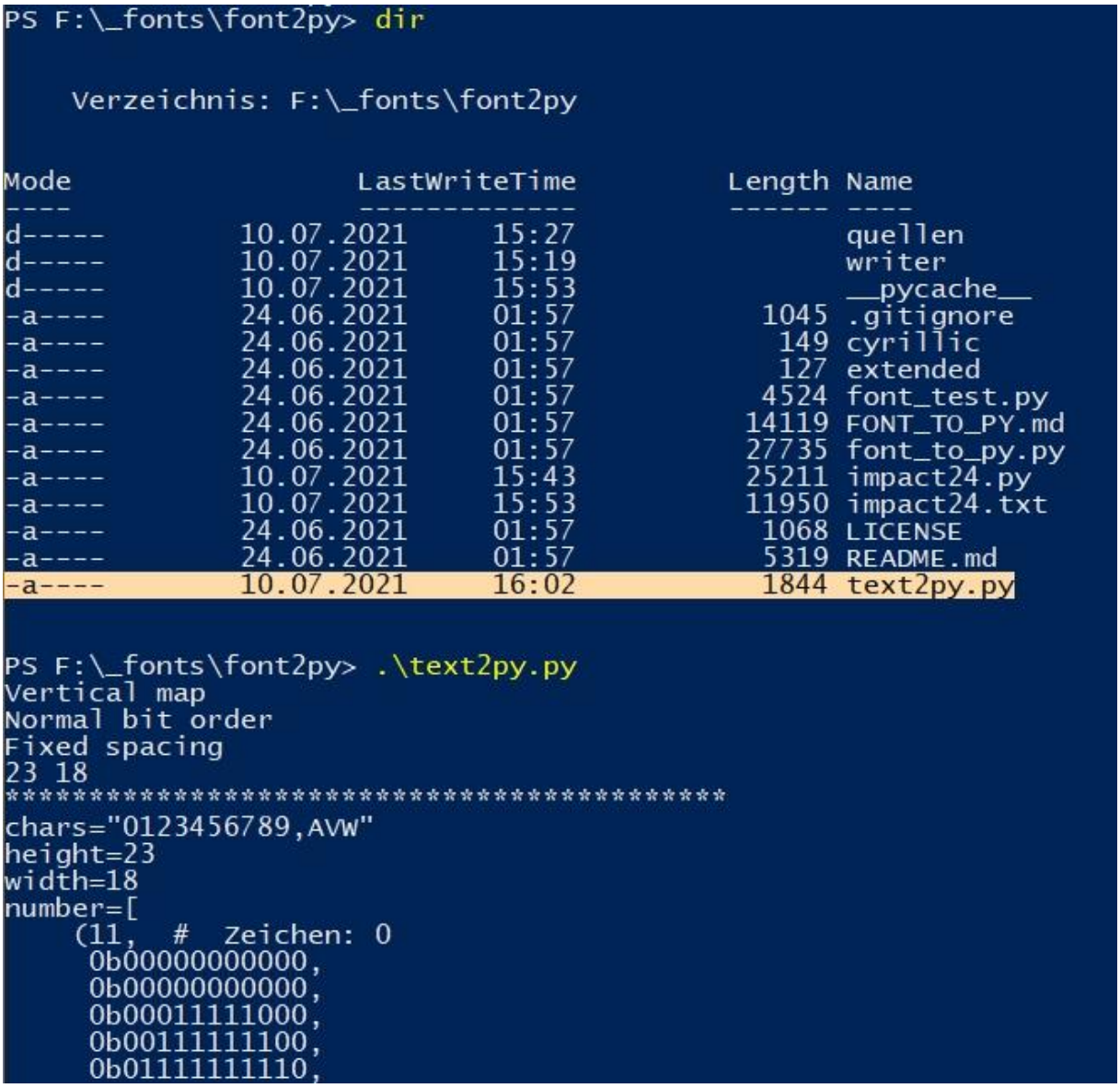
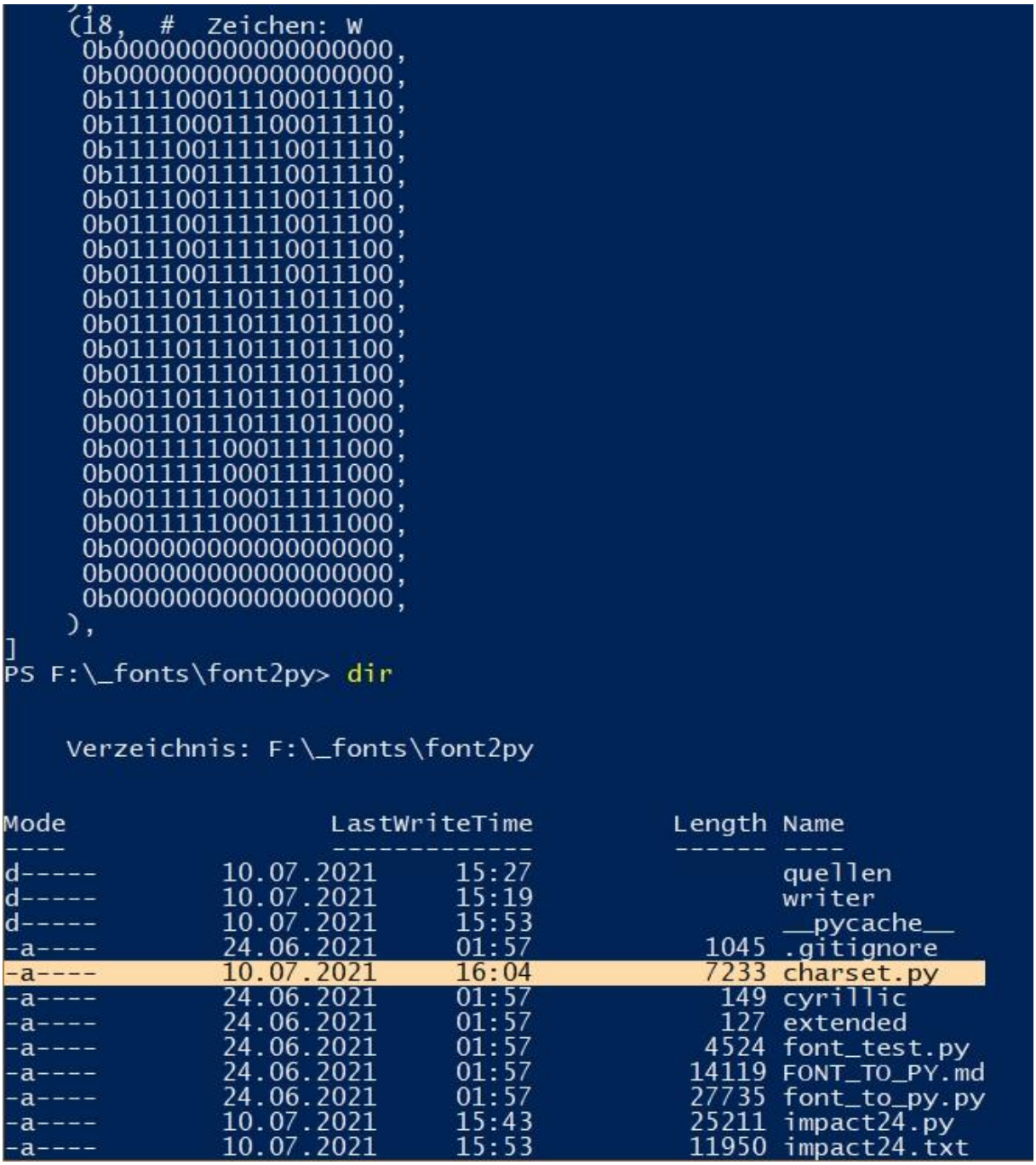


Abbildung 16: Charset05

Das Programm läuft auf dem PC, nicht im ESP8266. Es schneidet für jedes Zeichen von jeder Zeile der Zeichendefinition ein Stück mit der festgesetzten Zeichenbreite ab und sucht die Position des letzten "#". Zum Maximum dieser Positionen wird 1 addiert. Dann werden alle Zeilenstrings auf diese Länge verkürzt. So entsteht die ursprüngliche Gewichtung der Zeichen für ein ausgewogenes Schriftbild.

Jeder "." der Zeichendefinition wird nun als Nächstes in eine 0 und jedes # in eine 1 umgewandelt. Durch Voranstellen von "0b" entstehen im Listing aus dem Muster Ganzzahlen in Binärschreibweise. Pro Zeichen werden sie in einem Tupel zusammengefasst. Der erste Wert in jedem Tupel ist die Zeichenbreite. Die Tupel aller Zeichen werden in der Liste **number** gesammelt.

Das Ergebnis wird als Datei mit dem Namen **charset.py** abgespeichert, sie ist das angestrebte Modul, das jetzt in beliebige MicroPython-Programme importiert werden kann.



Das folgende Programm zeigt die Verwendung. **charset.py** wird in den Workspace des Projekts, hier **thermometer**, kopiert, in den Flash des ESP8266 geladen und als Modul im Programm **current**.py importiert. Wenn Sie bisher selbst noch keinen Zeichensatz erstellt haben, können Sie auch erst einmal meinen 30-er herunterladen.

```
# current.py
from machine import Pin, I2C, ADC, Timer
from time import sleep, ticks_ms,sleep_ms
from oled import OLED
import charset as cs

# 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)
UmidCnt=531

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

def putDigit(n,xpos,ypos,show=True):
    breite=cs.number[n][0]
    for row in range(1,cs.height):
        for col in range(breite-1,-1,-1):
            c=cs.number[n][row] & 1<<col
            d.setPixel(xpos+breite+3-col,ypos+row,c,False)
    if show:
        d.show()
    return xpos+breite+2

def putValue(wert,unit,xpos,ypos,show=True):
    d.clearAll(False)
    ws=str(wert).upper()
    pos=0
    ws=ws.replace(".",",")
    for z in ws:
        try:
            n=cs.chars.index(z)
            pos=putDigit(n,pos,0,False)
        except:
            print("unerlaubtes Zeichen:",z)
    try:
        n=cs.chars.index(unit)
        pos=putDigit(n,pos,0,True)
    except:
        print("unerlaubte Einheit:",unit)
    return pos

def getRawADC():
    s=0
    for i in range(10):
       s+=adc.read()
    return s//10

def calibrate():
    global UmidCnt
    UmidCnt=getRawADC()
    print(UmidCnt)
```

```
def getCurrent():
    cnts=getRawADC()-UmidCnt
    print(cnts)
    U=3000/1024*cnts
    I=int((U/58)*100)/100
    print(I)
    return I

calibrate()

while 1:
    putValue(getCurrent(),"A",0,0)
    sleep(1)
```

Das Programm für die Strommessung benötigt außer dem Modul charset.py noch oled.py und ssd1306.py. Alle wesentlichen Aktionen zur Stromerfassung und Darstellung mit dem vergrößerten Zeichensatz werden durch Funktionen erledigt. Die Darstellung erfolgt durch das Setzen derjenigen Pixel, die einer 1 in der Binärdarstellung einer Pixelzeile entsprechen. Das macht die Funktion putDigit(). Sie nimmt die Ziffer und die Koordinaten für die linke obere Ecke der Darstellung und gibt den x-Wert für die nächste Ausgabeposition zurück. Die Funktion putValue() nimmt einen Messwert und ein Einheitenzeichen. Der Wert, Integer oder Float, wird geparst. Die Funktion wandelt dabei evtl den Dezimalpunkt in ein Komma um, stellt die Digits, falls möglich dar und gibt ebenfalls die nächste freie x-Position zurück.

Beiden Funktionen ist der optionale Parameter show gemeinsam. Der Defaultwert True führt zur sofortigen Anzeige. False führt nur zu Änderungen des Framebufferinhalts, der erst mit d.show() zur Anzeige gebracht wird.

Die Applikation muss **stromlos** gestartet werden, damit die Calibrierung der Strommessung auf Null richtig funktioniert. Die Funktion **getCurrent**() holt einen Stromstärkewert und rechnet den Rohwert in die Einheit Ampere um.

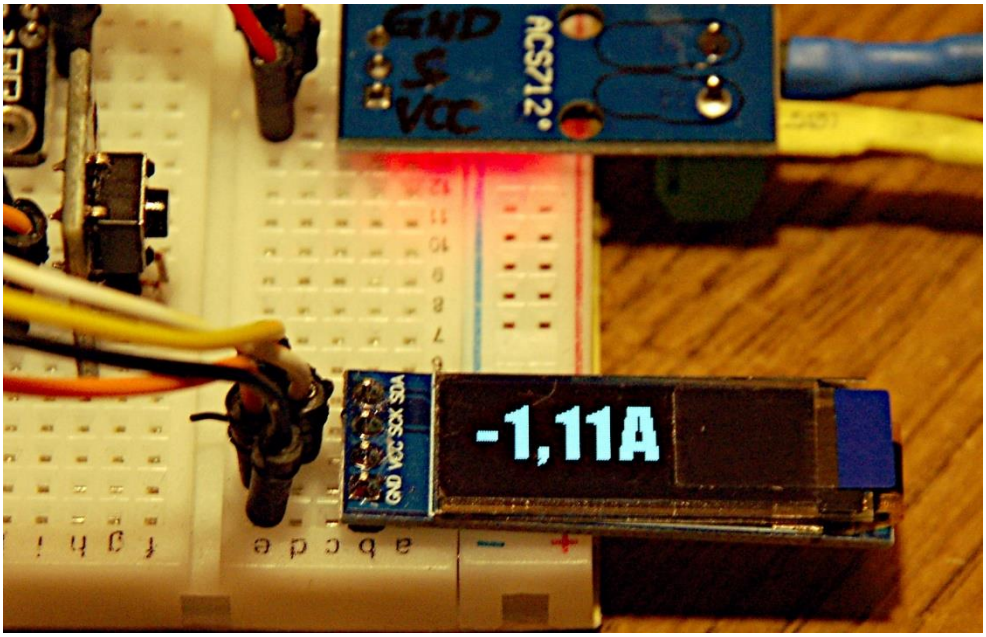


Abbildung 17: Ampere-Anzeige

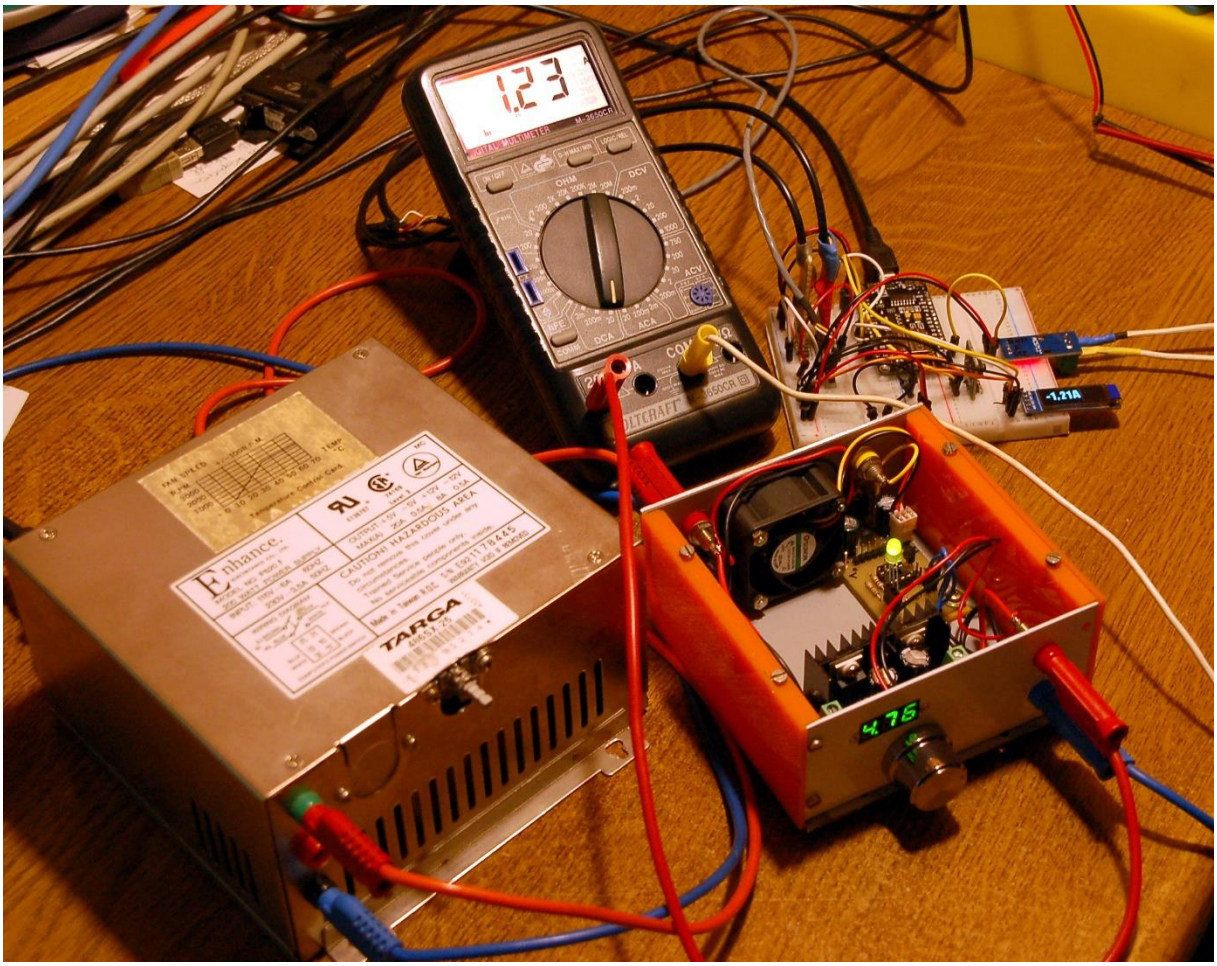


Abbildung 18: Testaufbau zur Strommessung

Mit den Fotos vom Testaufbau, bei dem ich gerade mit einem Thermoelement einen Aluquader aufheize und damit zur Messung der spezifischen Leistung des Elements vorbereite, verabschiede ich mich für heute. Mit genau diesem Thema beschäftigen wir uns im nächsten Beitrag. Dann behandeln wir die eher unbekannte, aber bestimmt nicht uninteressante Seite der Peltierelemente bei der Messung von Wärmeströmen.

Viel Vergnügen beim Bau und Betrieb des DC-Converters und natürlich der Programmierung des ESP8266.

Beitrag als PDF downloaden.