

Die Schaltung zum Projekt

Der Beitrag ist auch als [PDF-Dokument](#) verfügbar.

Was kann man tun, wenn das Projekt immer umfangreicher wird und die Programmteile schließlich nicht mehr auf dem ESP32/ESP8266 Platz haben? Nun, ganz einfach, man kann zum Beispiel die Module compilieren. Dadurch werden sie auf gut die Hälfte zusammengeschrumpft. Ein weiterer Aspekt könnte auch interessant sein, compilierte Programme liegen nicht mehr im Klartext vor und so kann man auch die ursprüngliche Programmstruktur nicht mehr erkennen. Die Ausgabe eines Listings ist nicht mehr möglich. Wie man ein Programm compiliert, wollen Sie wissen? In dieser Blogfolge zeige ich Ihnen in kleinen Schritten das Vorgehen auf Windows und Linux. Willkommen zu

MicroPython auf dem ESP32 und ESP8266

heute

mpy-cross – Compilereinsatz für die ESP-Family

Eines vorweg: bewahren Sie Ihre Originaldateien mit der Endung **.py** stets gut auf, denn wenn Sie nachträglich doch noch etwas ändern wollen, dann geht das nur in der Textdatei. Am Compilat können keine Änderungen mehr durchgeführt werden. Der Versuch, eine **.mpy**-Datei in Thonny zu öffnen führt zu folgender Ausgabe.



Abbildung 1: Thonny kann tcs3472.mpy nicht öffnen

Durch das Compilieren, kann ich also auch dafür sorgen, dass an meinem Programm keiner mehr rumpfriemeln kann.

Kommen wir zur Hardware. Als Chef des Projekts setze ich einen ESP32 ein, ein ESP8266 tut es aber auch, brauche ich doch für diesen Beitrag sonst nur noch mein Farberkennungsmodul und ein OLED-Display. Aufgebaut wird die Schaltung wie immer auf einem Breadboard mit etlichen Jumperkabeln.

Hardware

1	ESP32 Dev Kit C unverlötet oder ESP32 Dev Kit C V4 unverlötet oder ESP32 NodeMCU Module WLAN WiFi Development Board mit CP2102 oder NodeMCU-ESP-32S-Kit oder
1	NodeMCU Lua Amica Modul V2 ESP8266 ESP-12F WIFI oder D1 Mini NodeMcu mit ESP8266-12F WLAN Modul oder NodeMCU Lua Lolin V3 Module ESP8266 ESP-12F WIFI
1	0,96 Zoll OLED SSD1306 Display I2C 128 x 64 Pixel
1	TCS34725 RGB Farb Sensor mit Infrarot-Filter
1	Breadboard Kit - 3x Jumper Wire m2m/f2m/f2f + 3er Set MB102 Breadbord kompatibel mit Arduino und Raspberry Pi - 1x Set

Die Software

Fürs Flashen und die Programmierung des ESP32:

[Thonny](#) oder
[µPyCraft](#)

Verwendete Firmware für den ESP8266:

Unbedingt diese Version nehmen
[v1.19.1 \(2022-06-18\) .bin](#)

Verwendete Firmware für den ESP32:

Unbedingt diese Version nehmen
[v1.19.1 \(2022-06-18\) .bin](#)

Die MicroPython-Programme zum Projekt:

[ssd1306.py](#) Hardwaretreiber für das OLED-Display
[oled.py](#) API für das OLED-Display
[tcs3472.py](#) Hardwaretreiber für das Farbmodul
[tcs3472.mpy](#) kompilierter Hardwaretreiber für das Farbmodul
[colorcheck.py](#) Demo-Programm
mpy-cross MicroPython-Cross-Compiler (Windows und Linux)
Angaben zur Installation folgen weiter unten.

Hilfsprogramme

[HXD](#) Hexeditor zum Anzeigen des Inhalts von Binärdateien (Freeware)

MicroPython - Sprache - Module und Programme

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

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

Sobald die Firmware geflasht ist, können Sie sich zwanglos mit Ihrem Controller im Zwiegespräch unterhalten, einzelne Befehle testen und sofort die Antwort sehen, ohne vorher ein ganzes Programm kompilieren und übertragen zu müssen. Genau das stört mich nämlich an der Arduino-IDE. Man spart einfach enorm Zeit, wenn man einfache Tests der Syntax und der Hardware bis hin zum Ausprobieren und Verfeinern von Funktionen und ganzen Programmteilen über die Kommandozeile vorab prüfen kann, bevor man ein Programm daraus strickt. Zu diesem Zweck erstelle ich auch gerne immer wieder kleine Testprogramme. Als eine Art Makro fassen sie wiederkehrende Befehle zusammen. Aus solchen Programmfragmenten entwickeln sich dann mitunter ganze Anwendungen.

Autostart

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

Programme testen

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

Zwischendurch doch mal wieder Arduino-IDE?

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

Signale auf dem I2C-Bus

Wie eine Übertragung auf dem I2C-Bus abläuft und wie die Signalfolge aussieht, das können Sie in meinem Beitrag [mammutmatrix_2_ger.pdf](#) nachlesen. Ich verwende dort ein [interessantes kleines Tool](#), mit dem Sie die I2C-Bus-Signale auf Ihren PC holen und analysieren können.

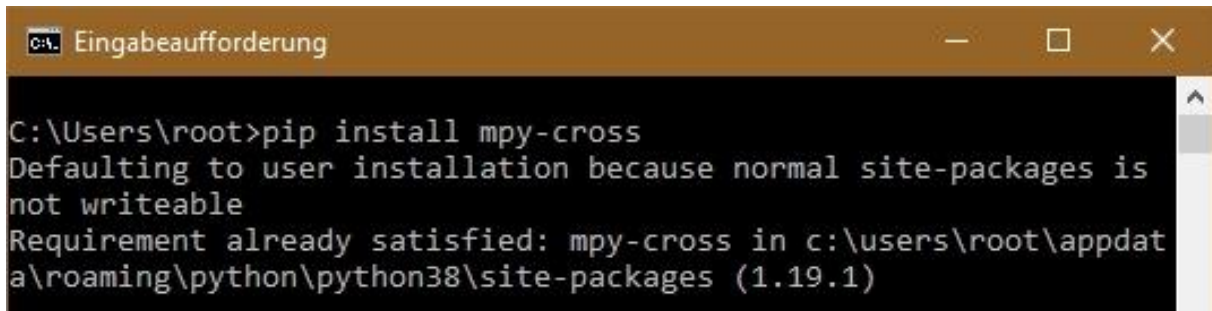
Compilieren auf der Windows-Kiste

Wenn Sie auf Ihrem Windows-Rechner Thonny installiert haben, muss wohl auch Python 3.8 oder höher bereits vorhanden sein, denn Thonny ist in Python geschrieben und daher benötigt es die Laufzeitumgebung von Python. Das ist vergleichbar mit der MicroPython-Firmware auf dem ESP32/ESP8266. In diesem Fall sollte sich auch bereits mpy-cross auf Ihrem Rechner befinden.

Öffnen Sie einfach einmal die Eingabeaufforderung, geben Sie die folgende Zeile ein und drücken Sie ENTER.

Pip install mpy-cross

Als Ergebnis erhalten Sie eine der beiden Rückmeldungen von pip.

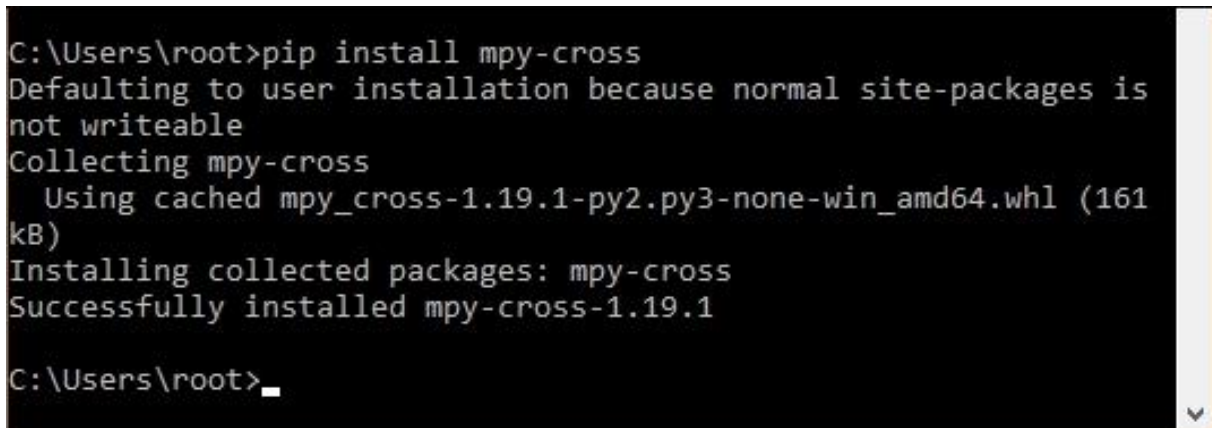


```
C:\Users\root>pip install mpy-cross
Defaulting to user installation because normal site-packages is
not writeable
Requirement already satisfied: mpy-cross in c:\users\root\appdat
a\roaming\python\python38\site-packages (1.19.1)
```

Abbildung 2: mpy-cross ist bereits installiert

So sieht es aus, wenn mpy-cross bereits auf Ihrem System installiert ist. Sie finden die exe-Datei im angegebenen Verzeichnis.

War mpy-cross vor der Ausführung des pip-Befehls noch nicht installiert, dann ist das jetzt der Fall.



```
C:\Users\root>pip install mpy-cross
Defaulting to user installation because normal site-packages is
not writeable
Collecting mpy-cross
  Using cached mpy_cross-1.19.1-py2.py3-none-win_amd64.whl (161
kB)
Installing collected packages: mpy-cross
Successfully installed mpy-cross-1.19.1

C:\Users\root>_
```

Abbildung 3: mpy-cross wurde installiert

Um zu erfahren, wo sich mpy-cross.exe befindet, geben Sie den pip-Befehl noch einmal ein, worauf Sie die obere Rückmeldung mit dem Zielverzeichnis erhalten.

Warum ist die Position der Datei so wichtig? Um ein mcp-File zu compilieren, also in Bytecode zu übersetzen, müssen Sie mpy-cross aufrufen und zwar mit dem Namen der MicroPython-Datei als Argument. Hierzu haben Sie drei verschiedene Optionen mit stark unterschiedlichem Komfort. Die ersten beiden stelle ich bereits teiloptimiert vor.

1. Sie befinden sich im Workspace ihres Projekts von Thonny und wollen die Datei painted_sky.py compilieren. Dann lautet der Aufruf etwa so.

C:\Users\root\AppData\Roaming\Python\Python38\site-packages\mpy_cross\mpy-cross.exe painted_sky.py

2. Sie befinden sich im Verzeichnis
C:\Users\root\AppData\Roaming\Python\Python38\site-

packages\mpy_cross\mpy-cross. Dann müssen Sie den Pfad zu Ihrer Quelldatei angeben und der kann auch eine etwas längliche Gestalt haben.

mpy-cross F:\P_programmieren\az-blog__Farberkennung mit TCS3472_workspace\painted_sky.py

3. Ich habe mich für den bequemsten Weg entschieden, und der geht so. Ich trage den Pfad zu mpy-cross.exe in die Pfadvariable PATH meines Windowssystems ein und kann dann das Programm aus jedem beliebigen Verzeichnis heraus starten, einfach nur mit **mpy-cross**, ohne Verzeichnisangabe. Den Eintrag in die Pfadvariable muss ich auch nur einmal durchführen. Dann wechsle ich auf meine Arbeitsfestplatte und gebe folgende Befehle ein, um ins Arbeitsverzeichnis zu kommen.

```
C:\Users\root>f:
F:\>cd \P_programmieren\az-blog\__Farberkennung mit TCS3472\_workspace\
F:\P_programmieren\az-blog\__Farberkennung mit TCS3472\_workspace>mpy-cross painted_sky.py
F:\P_programmieren\az-blog\__Farberkennung mit TCS3472\_workspace>
```

Abbildung 4: Compilieren im Arbeitsverzeichnis

Den Pfad zum Workspace kopiere ich mir aus dem Explorerfenster und füge ihn in die Eingabeaufforderung ein. Nach dem Wechsel in das Arbeitsverzeichnis kann ich alle dort befindlichen *.py – Dateien in Kurzform compilieren.

Zur Pfaderweiterung führen ein paar einfache Schritte.

1. Rechtsklick auf das Windows-Symbol – System.



Abbildung 5: Start - System

2. Im rechten Fenster ganz nach unten rollen - Erweiterte Systemeinstellungen

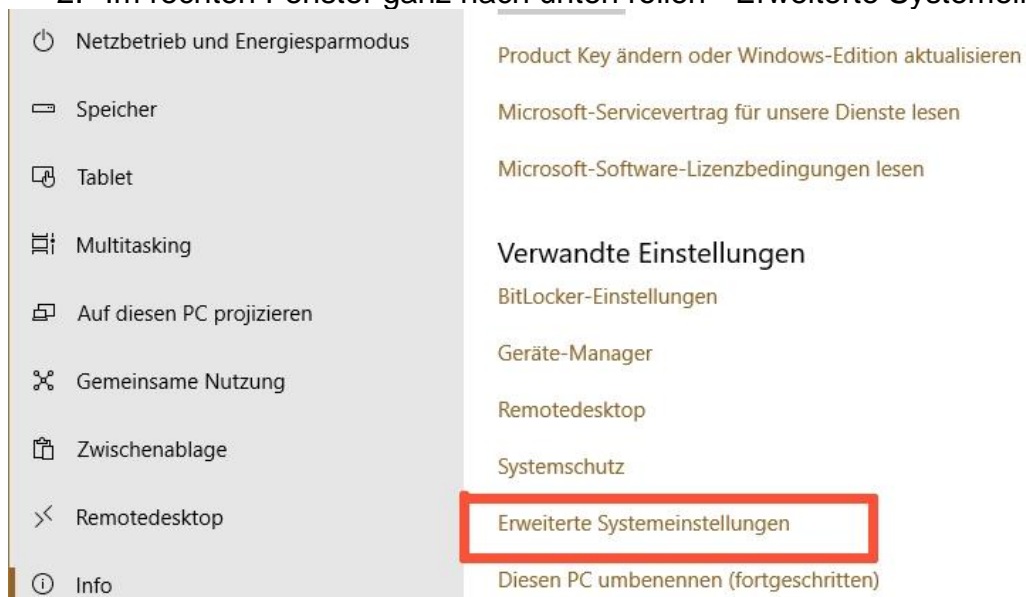


Abbildung 6: Erweiterte Systemeinstellungen

3. Umgebungsvariablen

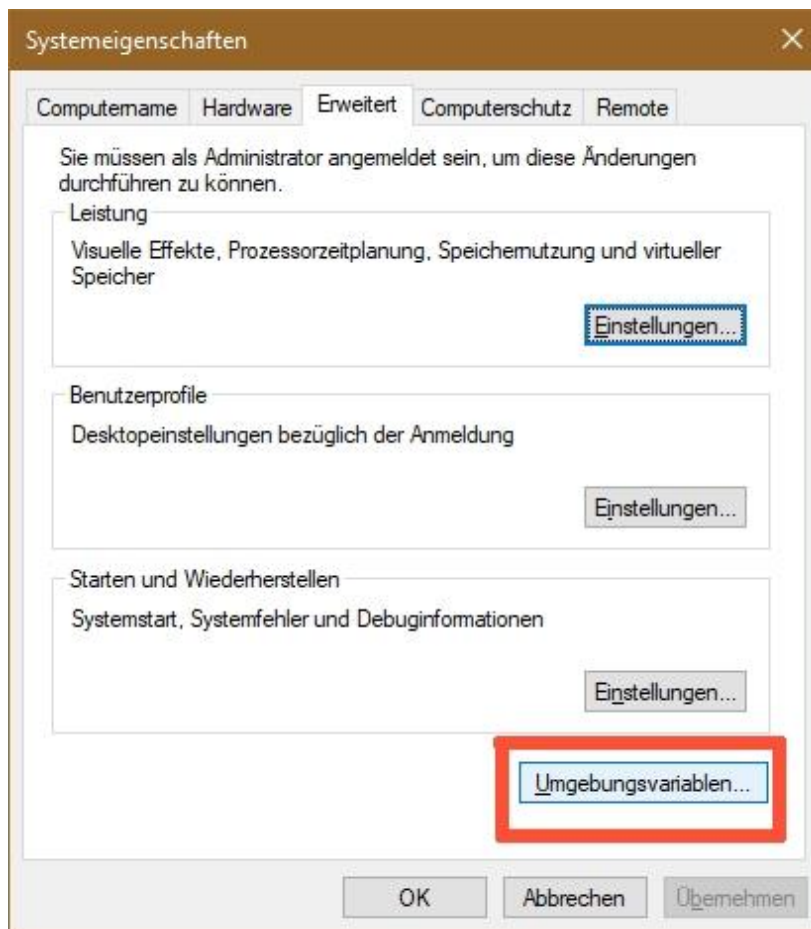


Abbildung 7: Umgebungsvariablen

4. Path markieren und Bearbeiten

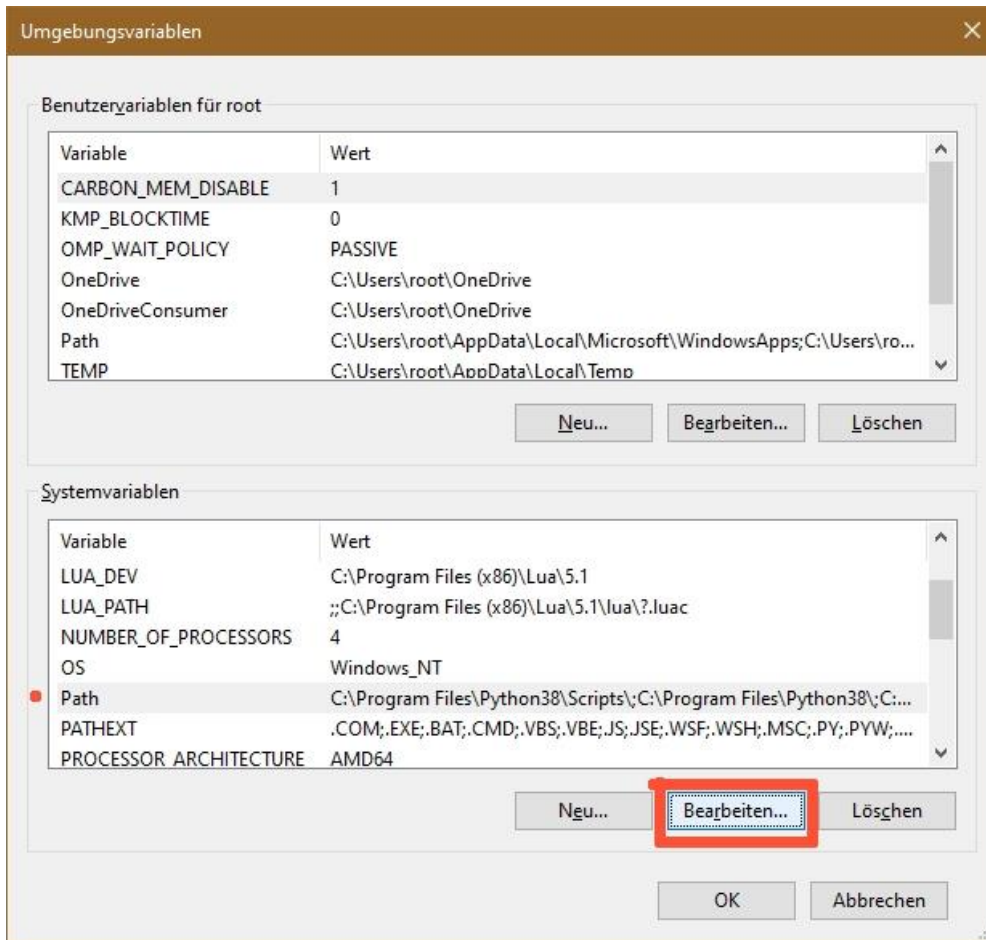


Abbildung 8: Systemvariablen - Pfad - Bearbeiten

5. Neu und in der Eingabezeile den Pfad zu mpy-cross.exe eintragen.

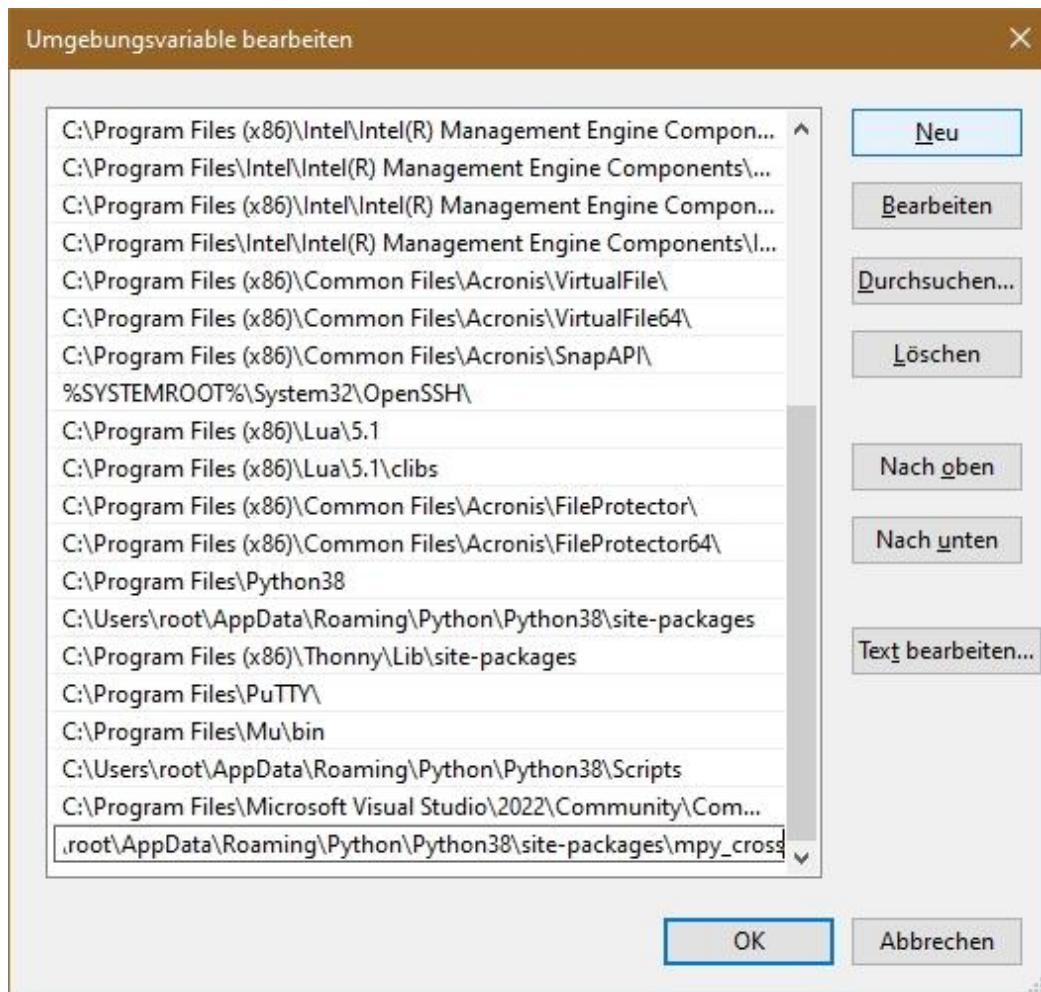


Abbildung 9: Neu - Pfad aus Explorer Kopfzeile einfügen

6. OK – OK – OK – fertig!

Da gibt es aber noch eine Kleinigkeit zu beachten, wenn Sie die erste Compilierung starten. Die Mainversionsnummer von **mpy-cross** muss der Version des MicroPython-Interpreters entsprechen. Weil **pip** immer die letzte Version installiert, muss auch die entsprechende MicroPython-Version installiert sein. Deswegen habe ich oben bei der Firmware bereits darauf hingewiesen, die dort angegebenen bin-Dateien zu brennen.

Die Syntax für den Compiler-Aufruf kennen Sie ja schon, es steht also nichts mehr im Wege, Ihre erste MicroPython-Datei zu übersetzen. Sie werden erstaunt sein, dass das Compilat nur noch halb so groß ist, wie die Ausgangsdatei.

Eine andere Möglichkeit, die Version von mpy-cross abzufragen ist

mpy-cross --version

MicroPython v1.19.1 on 2022-06-18; mpy-cross emitting mpy v6

Falls eine frühere Version gebraucht wird, weil die MicroPython-Firmware ein besonderes Release sein muss, dann kann man auf <https://pypi.org/project/mpy->

[cross/#history](#) nachschauen, die whl-Datei herunterladen und manuell installieren. Alternativ kann auch die URL des Repositories mit in die pip-Zeile geschrieben werden.

Pip install -i

https://files.pythonhosted.org/packages/42/61/b44c6e3802eb6898c836790cd69a1eb7f1a1d834b69d60dbdf4d5e5d23fd/mpy_cross-1.17-py2.py3-none-win_amd64.whl

Die URL-Link wurde direkt von der Seite <https://pypi.org/project/mpy-cross/1.17/#files> kopiert, das erspart langes Tippen und Fruckdehler.

Wie stelle ich aber die Versionsnummer einer bereits übersetzten MicroPython-Datei fest, wenn man die gar nicht auflisten kann? Nun, dafür gibt es Tools, mit denen ich Binärdateien darstellen und sogar editieren kann. [HXD](#) ist zum Beispiel ein solches Programm. Laden Sie die Freeware herunter und entpacken Sie diese. Dann Doppelklick auf **hxdsetup.exe**. Folgen Sie nun einfach dem Installationsassitenten

Starten Sie **HxD** und ziehen Sie eine *.mpy-Datei auf das Arbeitsfenster. Das erste Zeichen ist ein großes "M", das zweite Byte ist die mpy-Versionsnummer. Diese entspricht nun nicht der Version des MicroPython-Release, sondern ist aus folgender Tabelle zu entnehmen.

MicroPython release .mpy version

v1.19 and up	6
v1.12 - v1.18	5
v1.11	4
v1.9.3 - v1.10	3
v1.9 - v1.9.2	2
v1.5.1 - v1.8.7	0

Tabelle 1

So sieht das in HxD aus.

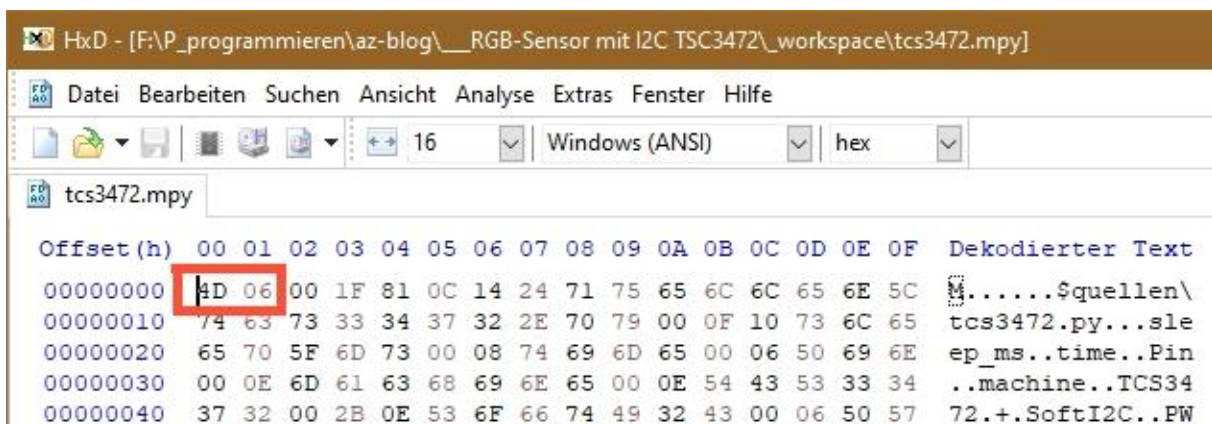


Abbildung 10: HxD-Plot

Kommen wir zu Linux

Auch der mpy-cross-Befehl in Linux arbeitet auf der Kommandozeile. Wir holen uns das ganze MicroPython-Paket von

<https://github.com/micropython/micropython/archive/refs/heads/micropython-master.zip>

in ein beliebiges Verzeichnis. Ich habe hierzu **/mcp** neu angelegt. Dann entpacke ich die Zip-Datei. Das von unzip angelegte Verzeichnis **/mcp/micropython-master** taufe ich um in **mcp** damit die Pfadnamen nicht so lang werden..

Gegebenenfalls müssen Sie dazu **unzip** installieren.

```
$ sudo apt-get install unzip
```

```
$ sudo unzip master.zip
```

```
$ sudo mv micropython-master mcp
```

Wechseln Sie jetzt in das Verzeichnis **mpy_cross**

```
$ cd mpy-cross
```

Geben Sie dort den Befehl

```
$ make
```

ein. Eine ganze Latte von Bildschirmausgaben rollt herunter und am Ende befindet sich im Verzeichnis **/mcp/mcp/mpy-cross/build** die lauffähige Datei **mpy-cross**. Verschieben Sie diese Datei nun ins Verzeichnis **/usr/bin**.

```
$ sudo cp mpy-cross/build/mpy-cross /usr/bin/mpc
```

Unter Linux müssen Sie den Programmsuchpfad nicht erweitern, weil der sowieso bereits den Eintrag **/usr/bin** enthält.

Wie Sie sehen, geht manches unter Linux bisweilen einfacher als in Windows.

Ein kleiner Wermutstropfen bleibt dennoch, Thonny müssen Sie mit Adminrechten starten, sonst kann es nicht auf das USB-Device zugreifen.

```
$ sudo thonny
```

Mit der Verwendung von mpy-cross können Sie jetzt größere Programme, gesichert auf einen ESP-Controller unterbringen. Viel Spaß damit!