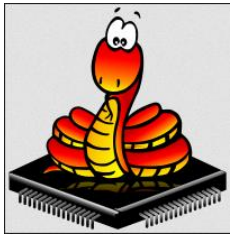
1. MicroPython auf ESP8266



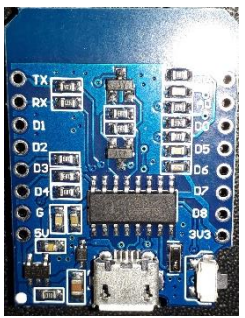
Das seit ca 5 Jahren bekannte Wifi Modul ESP01, basierend auf dem ESP8266 32Bit SoC uC der Firma Espressif, wird immer beliebter in der Maker Szene für Bastler, die kleine, WLAN-fähige Microcontroller verwenden wollen, die dazu noch einfach zu programmieren, preiswert und dennoch relativ schnell sind.
Der Einzelpreis liegt bei chinesischen Händlern ab ca. 1.50-2.00€ inkl. Versandkosten, größere Boards in DE bei ca. 5-10€.

Dies sind Eigenschaften, bei denen ein Arduino mit seinem kleinen Speicher, langsamen 16MHz Takt, 8Bit CPU und fehlender Konnektivität nicht mithalten kann. Dazu kommt, daß HW-Erweiterungen (Shields) relativ teuer sind und nicht jeder C lernen möchte ;-)

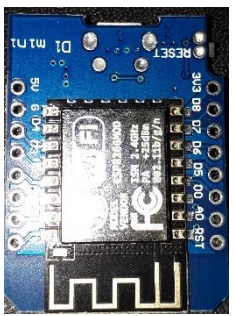
Nachdem die Comunity die beliebte Arduino Programmierumgebung um den ESP8266 erweitert hat, gibt es immer mehr Anwender, die auf dem ESP8266 programmieren, statt auf dem Atmega 328p des Arduino oder dem deutlich teureren und größeren Arduino Mega, die beide für WLAN Zusatz-Hardware benötigen, was den Preis nach oben treibt. Ein Raspberry Pi ist vielen eine Nummer zu groß und zu teuer.

Diverse ESP8266 Board Varianten

Wemos D1mini

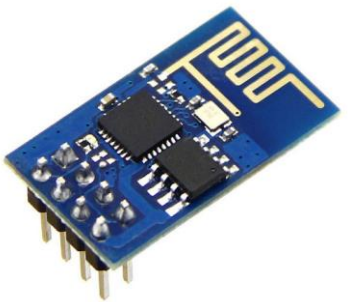


Unten mit 3,3V Regler und USB-UART-Chip



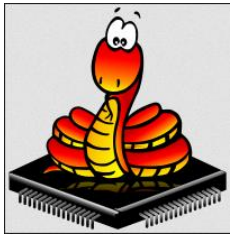
Oberseite mit ESP12-Modul, Micro-USB und Reset-Taster

ESP01 von Ai-Thinker



Das Ur-Modul, mit Mäander-Antenne 2,4GHz und externem SPI-Flash in SO8. Im Gegensatz zu anderen erhältlichen Modulen ohne Metalldeckel.

2. MicroPython



Was ist denn nun MicroPython?

MicroPython ist ein kleiner Open Source Interpreter in der Programmiersprache Python, der auf kleinen Entwicklerboards lauffähig ist. Mit MicroPython kann man einfach sauberen Python Code schreiben, um Hardware anzusteuern, ohne daß man komplizierte Low-Level Sprachen, wie C oder C++ anwenden muß. (Der Arduino wird in C programmiert).

Die Einfachheit der Python Programmiersprache macht MicroPython zu einer guten Wahl, wenn man unerfahren im Programmieren mit Hardware ist. In jedem Fall ist MicroPython recht gut ausgestattet und hat die wichtigsten Python Befehle so implementiert, daß sich selbst erfahrene Python Veteranen schnell in MicroPython zurecht finden werden.

Über die einfache Anwendung hinaus, hat MicroPython einige besondere Eigenschaften, die es von anderen Systemen unterscheidet:

- Interaktives REPL (Read-Evaluate-Print Loop): Dies erlaubt dem Anwender, sich mit dem Board zu verbinden und direkt darauf Programmcode auszuführen, ohne daß er compiliert oder hochgeladen werden muß. Perfekt um mal eben etwas zu probieren!

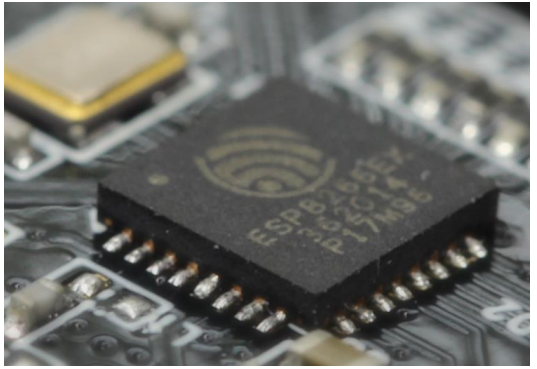
- Erweiterte Software Bibliothek. Wie die normale Python Programmiersprache, MicroPython ist “Ready-To-Fly” und hat die für die verschiedensten Anwendungen benötigten Librarys schon eingebaut. Z.B. um JSON Daten von einem Web Dienst zu parsen, Textsuche mit Regular Expressions, oder selbst Netzwerk-Sockets zu programmieren ist mit den Librarys von MicroPython einfach.

- Erweiterbarkeit. Für fortgeschrittene Anwender ist MicroPython erweiterbar mit Low-Level C/C++ Funktionen. Man kann so ausdrucksvollen High-Level MicroPython Code mit schnellem Low-Level Code mischen, wenn erforderlich.

3. Der ESP8266 von Innen

Während der ESP01 nur 2 GPIO Pins, sowie eine TX/RX Schnittstelle von Außen verfügbar hat, sind Boards mit aufgelöteten ESP-Modulen deutlich besser ausgestattet, sogar mit einem USB-UART-Wandler und 1MB, 4MB oder 16MB Flash, Spannungsregler und Batterie-Ladeschaltung für LiPos.

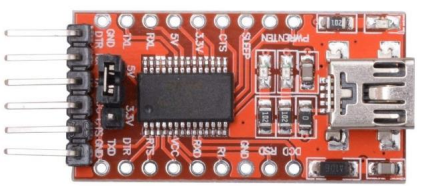
Spannungsversorgung: **2,5 bis 3,6 V, Typisch 3,3V**
Stromaufnahme: 15 µA (Sleep) bis 400 mA (Peak), durchschnittlich 80 mA



Schnittstellen:
- UART, SPI, I²C
- 11 programmierbare I/O Pins max. 12 mA
- 1 analoger Eingang 0 bis 1 V mit 10 Bit Auflösung
- **Alle digitalen Eingänge vertragen maximal 3,6 Volt**

Netzwerk:
- Wi-Fi 802.11 b/g/n 2,4 GHz mit WPA/WPA2 PSK
- TCP/IP v4, UDP und TCP, maximal 5 gleichzeitige Verbindungen
- Broadcast Nachrichten werden nicht unterstützt
- Durchsatz: 150 bis 300 kByte/s in beide Richtungen1

Das Besondere an allen ESP-Modulen ist jedoch, daß sie sich relativ einfach per UART Schnittstelle flashen lassen. Dabei läßt es sich nicht “kaputt” flashen, da das SoC ein internes Boot-ROM besitzt, das die Kommunikation nach außen und indirekt das Flashen der Anwendungs-Firmware in das externe SPI-Flash übernimmt.
Hierzu muß man nur das GPIO0 Pin auf GND ziehen, einen Reset auslösen, indem man das RST Pin kurz mit GND verbindet und schon kann man mit einem USB nach UART Wandler (siehe Bild) die neue Software vom PC aus flashen.
Hierzu im Detail später mehr.



4. Verwandtschaft des ESP8266, der ESP8285 SoC

ESP-8285 M3

1.CPU
ESP8285 integrates Tensilica L106 32-bit micro controller (MCU) and ultra-low-power 16-bit RSIC. The CPU clock speed is 80 MHz. It can also reach a maximum value of 160 MHz. Real Time Operation System (RTOS) is enabled. Currently, only 20% of MIPS has been occupied by the Wi-Fi stack, the rest can all be used for user application programming and development.

2.Memory
ESP8285 Wi-Fi SoC integrates memory controller and memory units including SRAM and ROM. MCU can access the memory units through iBus, dBus, and AHB interfaces.

3.Flash
ESP8285 has a built-in SPI flash to store user programs.
•Memory size: **1 MByte**
•SPI mode: Dual Out

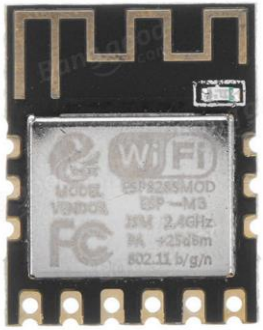
4.Radio
ESP8285 radio consists of the following blocks.
•2.4 GHz receiver and transmitter
•High speed clock generators and crystal oscillator
•Real time clock
•Bias and regulators
•Power management

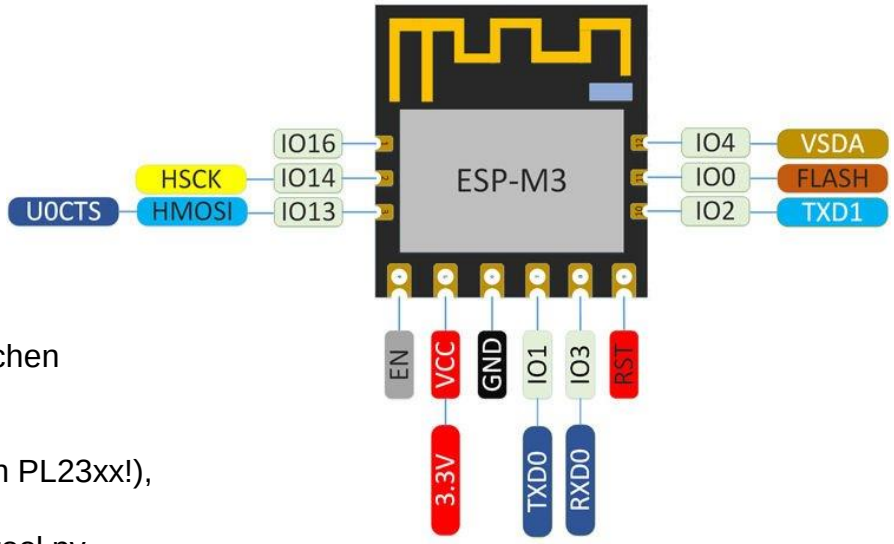
Operating temperature range: **-40 ℃ -125 ℃**

Link zum Block Diagramm:

http://linuxgizmos.com/files/espressif_esp8285_block.jpg

5. Die Anschlüsse des ESP8285 M3 Moduls

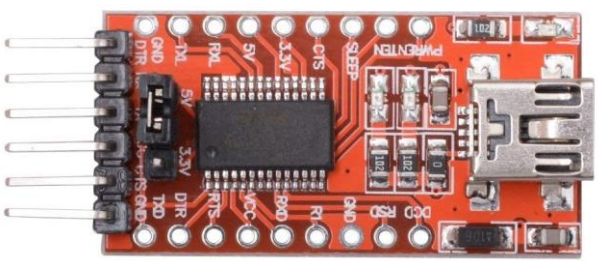




Egal welche Firmware man flashen möchte, man benötigt immer die gleichen Grundvoraussetzungen, die da sind:

Ein USB-UART Wandler mit **3,3V** (z.B. FT232RL, CH340G, CP2102, kein PL23xx!), zwei Taster, zwei Widerstände 2k2-10k, etwas Kabel und ein Elko 100uF.
Ein installiertes Python V3.5+ (V3.7.2 ist aktuell) und das Programm esptool.py

Für das Firmware-Flashen benutzt man das Tools esptool.py des Herstellers Espressif.
Man kann es hier finden: **https://github.com/espressif/esptool/**



Oder so in Python per **pip** nachladen:
pip install esptool

Der Standardpfad für die pip Installation ist hier zu finden:
C:\Users*UserName*\AppData\Local\Programs\Python\Python37-32\Scripts

<u>Wichtig:</u>UserName anpassen und den Pfad händisch eingeben, da **\AppData** versteckt it.

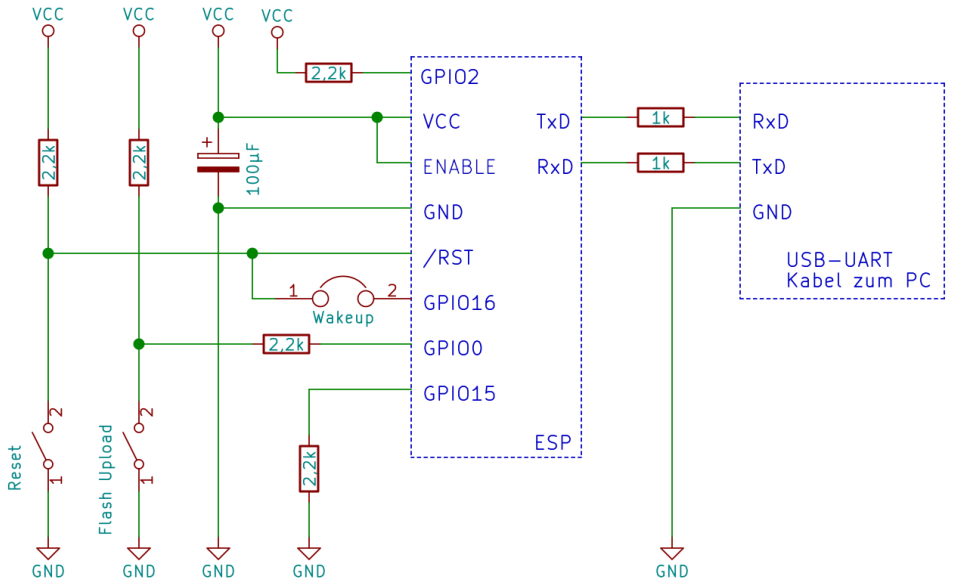
6. Die Schaltung zum Anschluß an den PC

Der ESP Chip enthält einen unveränderlichen Bootloader, welcher ein Firmware-Upgrade über den seriellen Port ermöglicht.
Beim Reset erwartet der Chip folgende Signale:

Firmware-Upload:
- GPIO0 = Low
- GPIO1 (TxD) = High über Widerstand oder offen
- GPIO2 = High über Widerstand oder offen
- GPIO15 = Low

Normaler Start:
- GPIO0 = High oder offen
- GPIO1 (TxD) = High über Widerstand oder offen
- GPIO2 = High über Widerstand oder offen
- GPIO15 = Low

Alle anderen Kombinationen an diesen vier Pins sind für andere Zwecke reserviert. Nach dem Start können diese Pins jedoch als frei programmierbare I/O Anschlüsse verwendet werden.
Daraus ergibt sich diese sinnvolle Grundschaltung:



Die Widerstände vor GPIO0, GPIO2 (und GPIO15, falls vorhanden) schützen vor Kurzschluss, falls man den Pin als Ausgang programmiert.
Der Wakeup-Jumper muss verbunden sein, wenn man den Deep-Sleep Modus mit Wakeup-Timer verwendet.

Um den ESP-Chip in den Firmware-Upload Modus zu versetzen, muss man beiden Taster gleichzeitig drücken und dann den Reset Taster zuerst loslassen. Der ESP Chip erwartet dann Kommandos und Daten vom PC.

7. Das Flashen der Firmware auf den ESP8285 (Teil1)

Nachdem die beiden Software-Komponenten Python >3.5 und esptool.py installiert sind, muss noch herausgefunden werden, auf welchen Com-Port Windows den USB-UART gelegt hat.

Hierzu unter Windows das Modul mit einem USB-Post verbinden, etwas warten, bis der Treiber installiert wurde.
Dann im Gerätemanager nachschauen, welche Nummer Windows dem Com-Port vergeben hat.
Ich verwende hier als Beispiel com3.
Entweder WIN+R oder in der Suchleiste der Windows-Oberfläche “cmd” <Enter> eingeben,
so daß sich ein Textfenster öffnet.

Dort diesen Befehl eingeben, **OHNE** <Enter> zu drücken:

esptool.py -p com3 erase_flash

(Bitte comX Port entsprechend anpassen!)

Jetzt Taste am GPIO0 **gedrückt halten** und **kurz** die Reset Taste drücken.

Dann den Befehl mit <Enter> abschicken.

Das Ergebnis sollte so aussehen wie rechts =>

```
C:\MicroPython>esptool.py -p com3 erase_flash
esptool.py v2.6
Serial port com3
Connecting....
Detecting chip type... ESP8266
Chip is ESP8285
Features: WiFi, Embedded Flash
MAC: 60:01:[illegible]
Uploading stub...
Running stub...
Stub running...
Erasing flash (this may take a while)...
Chip erase completed successfully in 3.7s
Hard resetting via RTS pin...

C:\MicroPython>
```

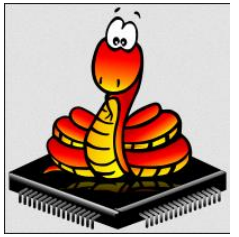
8. Das Flashen der Firmware auf den ESP8285 (Teil2)

Nach erfolgreichem Löschen, die gleiche Prozedur mit folgendem Befehl wiederholen:

esptool.py -p com3 -b 460800 write_flash -ff 40m -fs=detect -fm dout 0 esp8266-20jjmmdd-v1.X.Y.bin

```
C:\MicroPython>esptool.py -p com3 -b 460800 write_flash -ff 40m -fs=detect -fm dout 0 esp8266-20190113-v1.9.4-779-g5064df207.bin
esptool.py v2.6
Serial port com3
Connecting....
Detecting chip type... ESP8266
Chip is ESP8285
Features: WiFi, Embedded Flash
MAC: 60:01:[illegible]
Uploading stub...
Running stub...
Stub running...
Changing baud rate to 460800
Changed.
Configuring flash size...
Auto-detected Flash size: 1MB
Flash params set to 0x0320
Compressed 587560 bytes to 384343...
Wrote 587560 bytes (384343 compressed) at 0x00000000 in 9.3 seconds (effective 504.8 kbit/s)...
Hash of data verified.

Leaving...
Hard resetting via RTS pin...

C:\MicroPython>
```

9. Das Erwachen: MicroPython auf ESP8266



Mit einem Terminalprogramm, wie PuTTY sieht die erste Kontaktaufnahme so aus (115200 Baud,n,8,1):

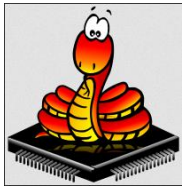
COM3 - PuTTY

```
OSError: [Errno 2] ENOENT

MicroPython v1.9.4-779-g5064df207 on 2019-01-13; ESP module with ESP8266
Type "help()" for more information.
>>> _
```

Hurra, geschafft!
Nach wunden Fingern vom Tippen kann es jetzt los gehen!

An seltsamen Zeichen und Meldungen vor dem Start von MicroPython darf man sich nicht stören, da der Bootloader nach dem Reset Status- und Debug-Meldungen ausgibt. Teilweise sogar in recht unüblichen Baudraten, wie z.B. 76800Baud. Die OSError-Meldung ist hier nicht wichtig.

10. Die MicroPython IDE EsPy



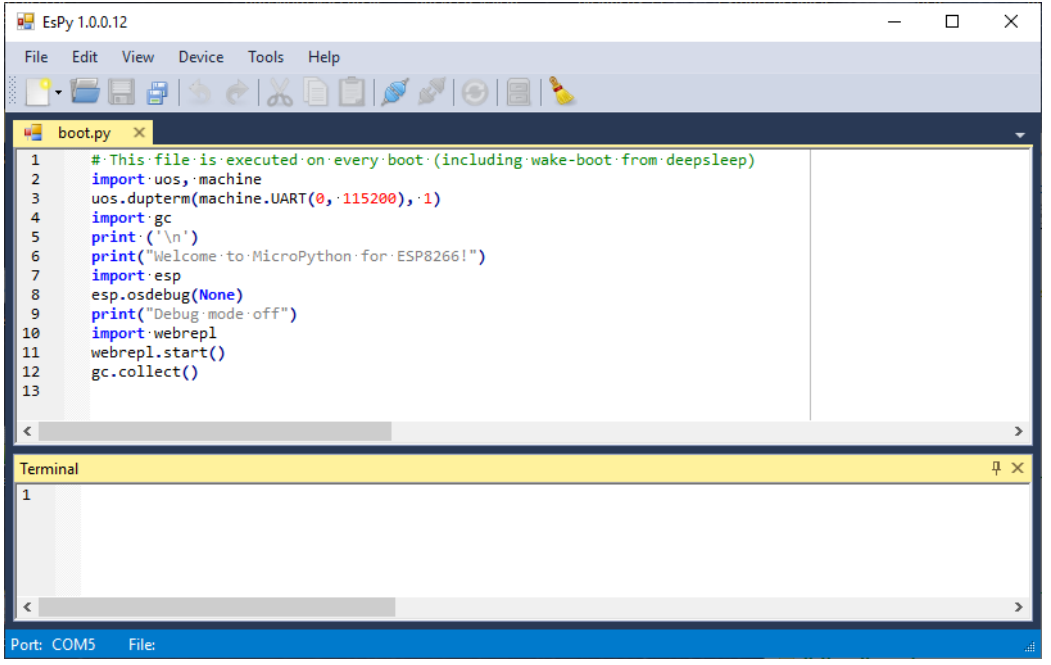
Obwohl nun mit einem einfachen Terminalprogramm und dem USB-UART-Modul die grundlegenden Voraussetzungen zum Programmieren unter MicroPython auf dem ESP-Baustein erfüllt sind, ist es dennoch etwas mühsam längere Programme einzutippen.

Das geht mit dem Lieblings-Editor, wie z.B. Notepad++, doch viel eleganter am PC.
Doch wie überträgt man den Code dann?

Eine offene und praktische IDE Oberfläche zur Entwicklung von eigenen Programmen, aber auch zum Flashen weiterer ESP Bausteine nennt sich EsPy.

Erhältlich unter **https://github.com/jungervin/EsPy**

Man kann komfortabel mehrere Dateien gleichzeitig öffnen und bearbeiten, wähend unten im Terminalfenster zum ESP-Baustein eine Verbindung besteht, so daß man dort in der REPL Eingaben machen kann.
Mit nur einem Maus-Klick überträgt man den Code aus dem Fenster in den Flash-Speicher des ESP.

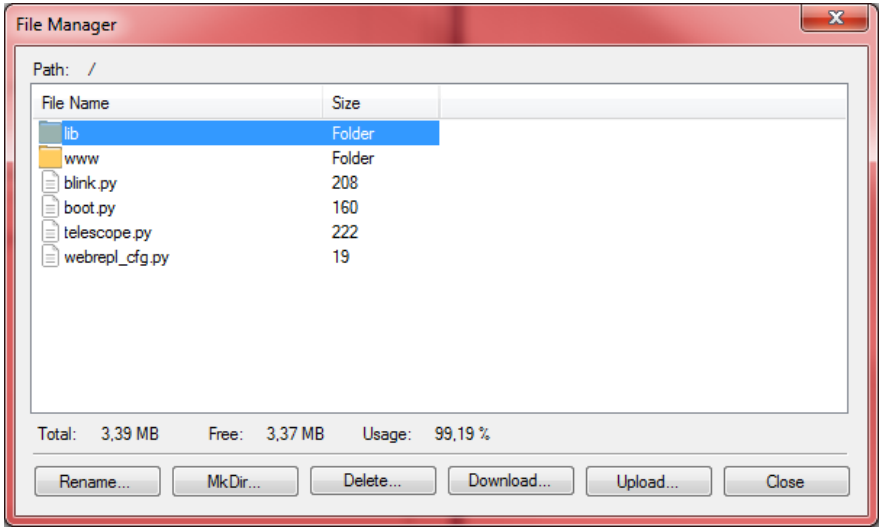


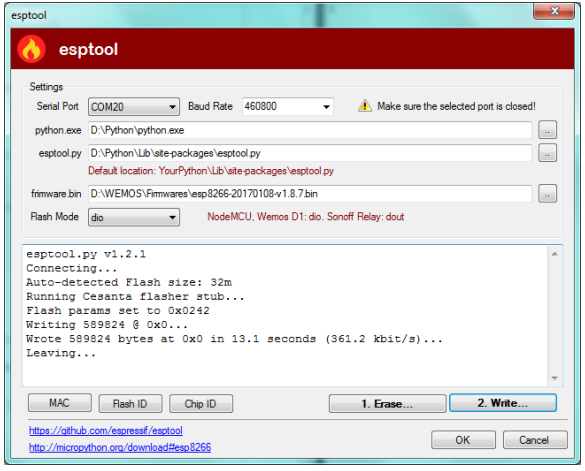
11. Die Ausstattung von EsPy

Mit dem integrierten Dateimanager kann man seine Dateien bequem uploaden oder downloaden, löschen oder Verzeichnisse erstellen.

Wenn auch typische Funktionen von bekannten IDEs (noch) fehlen, wie z.B Suchen und Ersetzen, so ermöglicht EsPy in der aktuellen Version immerhin schon recht komfortables Arbeiten. **==>**

Zum Code-Schreiben kann man immer noch seinen Lieblingseditor verwenden, zum Übertragen und Verwalten spart man sich viel Kommandozeilen-Tipperei.





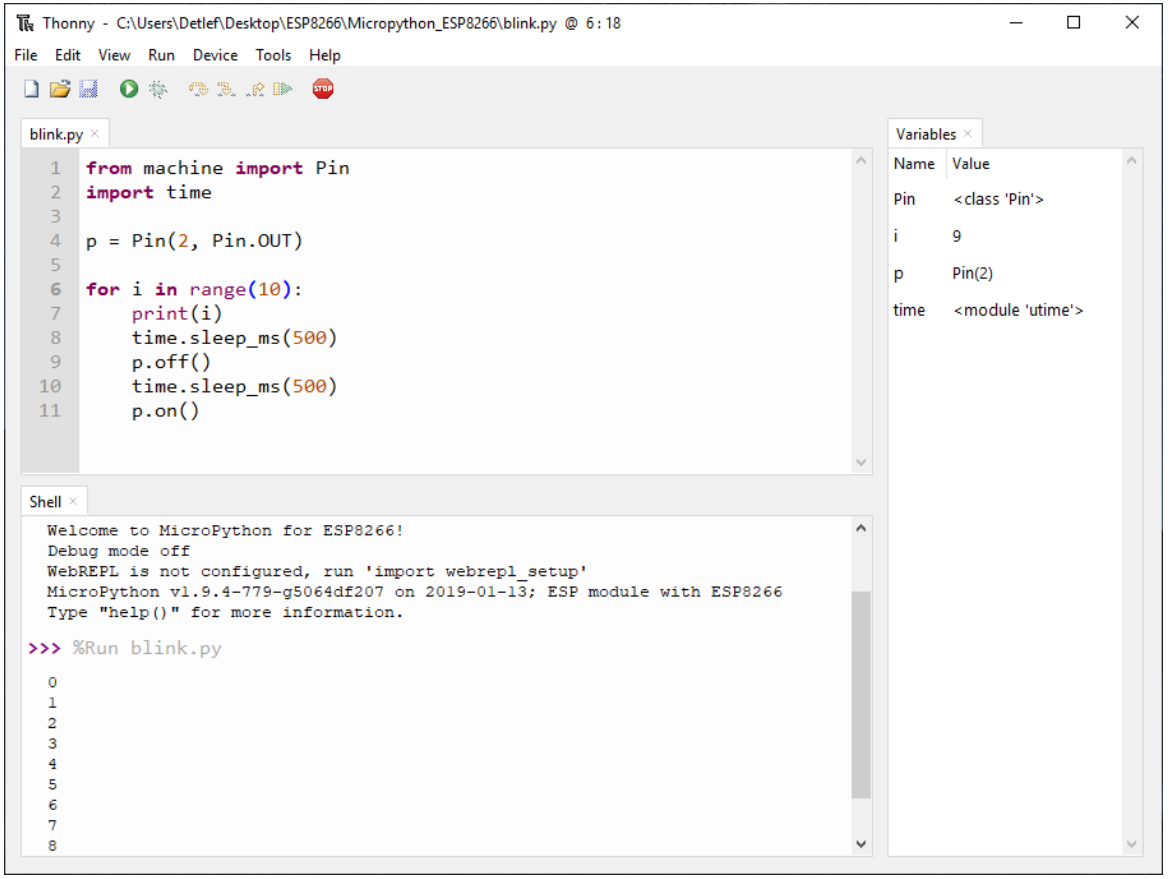
Wenn man sich die (kleine) Mühe macht und im vorgesehenen Reiter die Pfade zum **esptool.py**, sowie der MicroPython **.bin** Datei eingibt, so kann man auch hier beim Flashen weiterer ESP Bausteine unbequeme Kommandozeilen-Episoden vermeiden.

<==

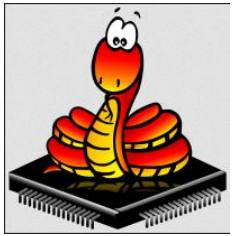
12. Die MicroPython IDE Thonny

Eine weitere kostenfreie interessante IDE Variante, die zu Lehrzwecken an der Universität von Taru (Estland) entwickelt wurde und dort eigesetzt wird.
(Siehe: https://thonny.org)

Primär zwar für das Coden auf dem PC (Win/Mac/Linux) gedacht, unterstützt es in der V3.0.8 auch die Online-Entwicklung auf einem per Com Port angeschlossenen MycroPython Modul. Es kann den geschrieben Code direkt übertragen, auslesen, zeigt Variablen an und im Terminalfenster unten gleich das Ergebnis.
Ein weiterer Vorteil für weniger erfahrene Programmierer ist, daß Python 3.7 gleich eingebaut ist.



13. Erste MicroPython Beispiele



Im REPL Terminalfenster können nun die ersten Gehversuche stattfinden.
Wenn wir z.B. wissen wollen, welche Dateien im Flash abgelegt sind, schauen wir einfach nach:

```
>>> import os
>>> os.listdir()
['boot.py', 'main.py']
>>>
```

Im nächsten Beispiel wollen wir die LED an GPIO2 auf dem ESP Board ein- und ausschalten.

```
>>> import machine
>>> pin=machine.Pin(2, machine.Pin.OUT)
```

Nun läßt sich die LED einschalten:

```
>>> pin.value(0)
```

und wieder ausschalten

```
>>> pin.value(1)
```

oder auch so:

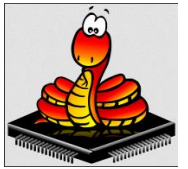
```
>>> pin.off()
>>> pin.on()
>>>
```

Wenn wir weiter eingeben**:**

```
>>> def toggle(p):
...     p.value(not p.value())
...
...
...
>>> import time
>>> while True:
...     toggle(pin)
...     time.sleep_ms(500)
...
>>>
```

Somit haben wir unsere erste Routine geschrieben.

14. Weitere MicroPython Beispiele



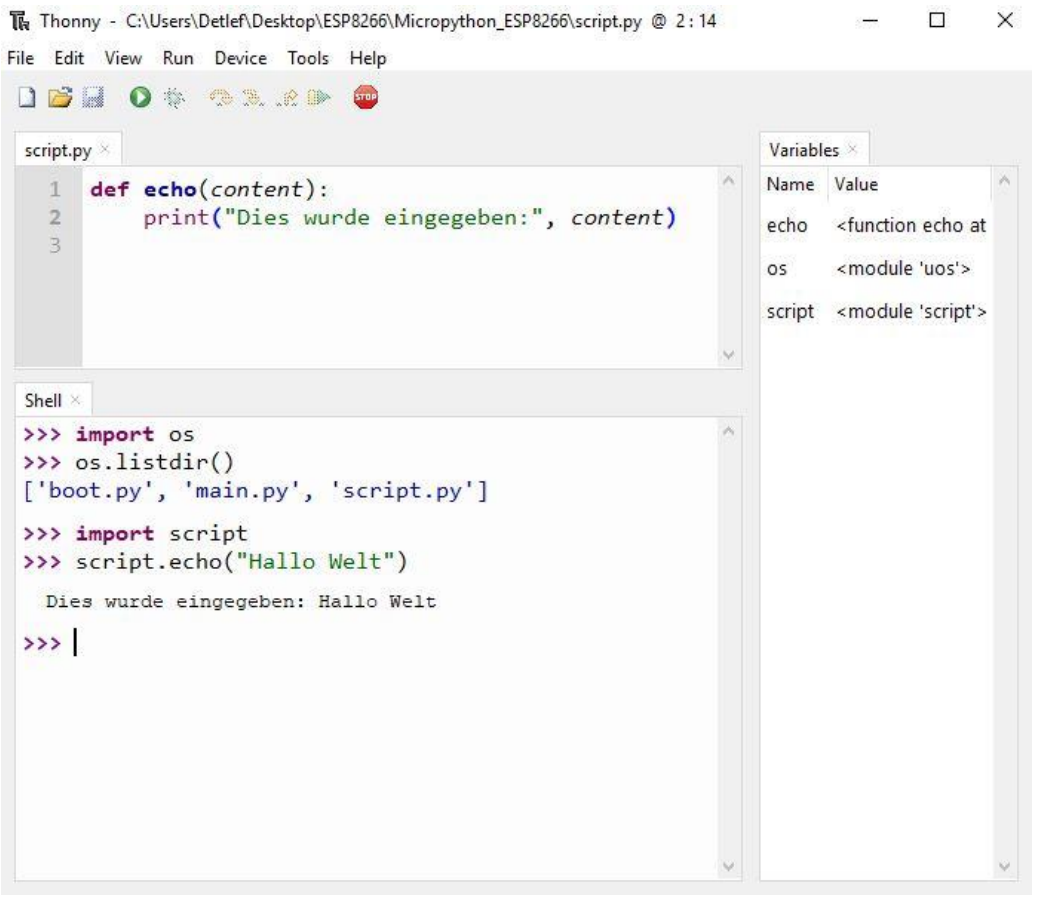
Wie werden die eigenen Programme nun am besten verwendet?

Den Programmcode schreiben und in einer oder mehreren Dateien mit .py Endung in den Flash-Spreicher übertragen.

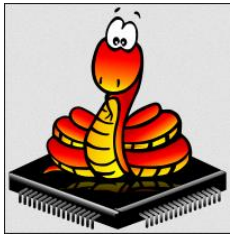
Wie werden die eigenen Scripte dann aufgerufen, wenn sie dort abgelegt sind?

Wir schauen zuerst nach, ob die Datei vorhanden ist.
So wie hier ==>

Dann einfach den Dateinamen **import**ieren und beim Aufrufen mit einem Punkt getrennt vor den Namen der Funktion setzen.
Siehe Beispiel rechts ==>



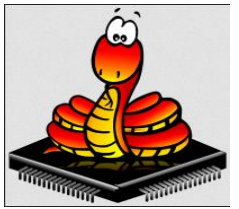
15. MicroPython Beispiele: Hilfe!



Ganz besonders nützlich ist die help() Funktion:

```
>>> help()
Welcome to MicroPython!

For online docs please visit http://docs.micropython.org/en/latest/esp8266/ .
For diagnostic information to include in bug reports execute 'import port_diag'.

Basic WiFi configuration:

import network
sta_if = network.WLAN(network.STA_IF); sta_if.active(True)
sta_if.scan()                             # Scan for available access points
sta_if.connect("<AP_name>", "<password>") # Connect to an AP
sta_if.isconnected()                      # Check for successful connection
# Change name/password of ESP8266's AP:
ap_if = network.WLAN(network.AP_IF)
ap_if.config(essid="<AP_NAME>", authmode=network.AUTH_WPA_WPA2_PSK, password="<password>")

Control commands:
  CTRL-A        -- on a blank line, enter raw REPL mode
  CTRL-B        -- on a blank line, enter normal REPL mode
  CTRL-C        -- interrupt a running program
  CTRL-D        -- on a blank line, do a soft reset of the board
  CTRL-E        -- on a blank line, enter paste mode

For further help on a specific object, type help(obj)
>>>
```

16. MicroPython-Oberfläche per WLAN: WebREPL



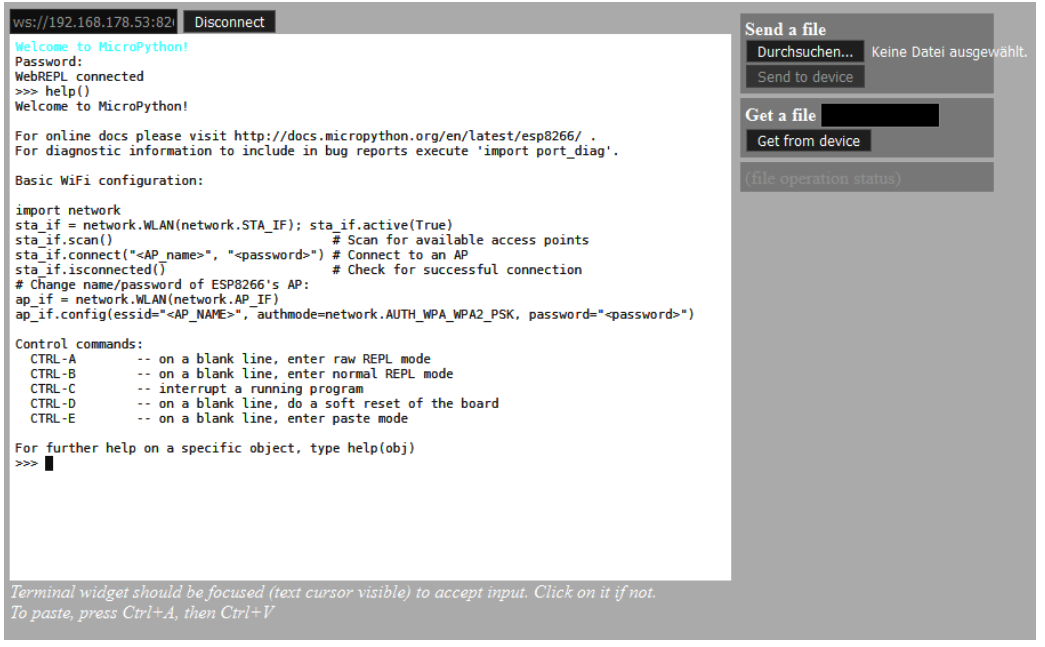
Mit einem normalem Web-Browser, wie Firefox, IE oder Chrome läßt sich über die WLAN Schnittstelle des ESP-Bausteins eine REPL Verbindung herstellen, die ein normales Arbeiten,
sowie Datei-Übertragungen auf einfache Art ermöglicht.

Hierzu ist ein einfaches Browser-Script zu starten: Das **WebREPL**

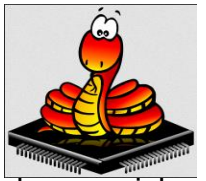
Eine gesonderte Version des Browser-Scripts für Android Geräte (Handy, Tablet) ist ebenso als **micropython_WEBREPL_Android.apk** Datei verfügbar. (nicht im Playstore)

Das passende Gegenstück im ESP Modul ist dort im MicroPython schon vorhanden und muß nur einmal konfiguriert werden, sofern es nicht anderweitig gestartet wird. (H33 WLAN-Manager)

import webrepl_setup



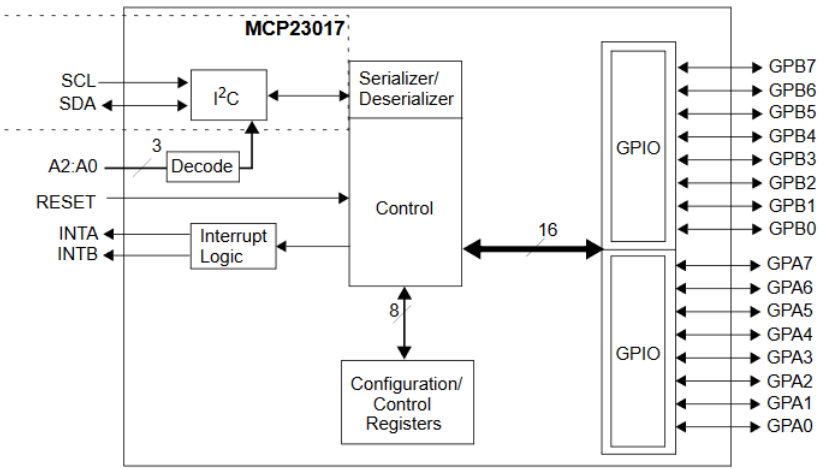
17. MicroPython GPIO-Anschlüsse

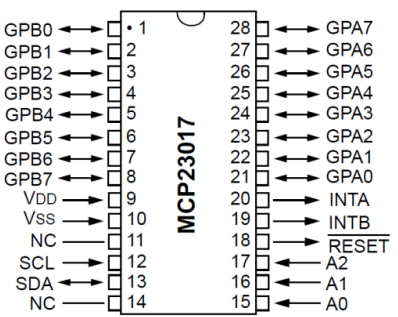


Recht einfach läßt sich Peripherie anschließen, auch wenn bei den meisten Modulen nur wenige Anschlüsse (GPIOs) herausgeführt sind und letztlich noch ungenutzt sind. Neben direkter Steuerung von LEDs oder Relais per Schalttransistor lassen sich auch Bausteine mit seriellen Schnittstellen I2C oder SPI anschließen. Hierzu zählen LCDs / OLEDs oder auch Port-Expander, die mit I2C oder SPI erhältlich sind.

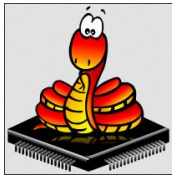
Hier der 2 x 8-Bit Port-Expander MCP23017 mit I2C Schnittstelle.
Es werden lediglich 2 Port-Pins benötigt (SCL, SDA) und durch einstellbare Adressen (A0-A2), sowie die Bus-Struktur des I2C, können damit bis zu 8 gleiche Expander benutzt werden, d.h. zusammen 128 GPIOs.
Die GPIOs lassen sich je einzeln als Eingang oder Ausgang konfigurieren. Jeder Ausgang kann High, Low, Open Drain, jeder Eingang mit einem Weak-Pull-Up programmiert werden.
Die I2C Geschwindigkeit bei 3,3V berträgt 400kHz, bei 5V (mit I2C Pegelwandler) bis zu 1,7MHz.

```
# I2C-Port-Expander.py
import time
from machine import Pin, I2C

slave_addr = 0x20       # MCP23017 address
IODIRA = 0x00           # IODIRA address
GPIOA = 0x12            # GPIOA address

def configure(PortDir, Data):
    global slave_addr
    buf = bytearray(2)
    buf = bytes([PortDir, Data])
    i2c.start()
    i2c.writeto(slave_addr, buf)
    i2c.stop()
#
def send(Port, Data):
    global slave_addri
    buf = bytearray(2)
    buf = bytes([Port, Data])
    i2c.start()
    i2c.writeto(slave_addr, buf)
    i2c.stop()
#
i2c = I2C(scl = Pin(2), sda = Pin(0), freq = 100000)
configure(IODIRA, 0xFE)
while True:
    send(GPIOA, 0)        # LED OFF
    time.sleep(1)         # Wait 1 second
    send(GPIOA, 1)        # LED ON
    time.sleep(1)         # Wait 1 second
```





MicroPython



Auf der Webseite ist alles Notwendige zu finden.
Man kann dort auch das Original-MicroPython-Board kaufen und
Infos über andere unterstützte Portierungen finden:
PyB
WiPy
ESP8266
ESP32
STM32F4
NUCLEO
Espruino Pico

https://micropython.org/

Auch der Source-Code ist dort verfügbar.
Einfach bei Adafruit oder Youtube nach Micropython suchen.

Das englischsprachige Forum unter
http://forum.micropython.org/
ist sehr aktiv, wird von den Entwicklern betrieben und unterstützt
auch gerne Anfänger.

https://forum.micropython.org/

Die Dokumentation und Infos zu neue Versionen zum Download:
http://docs.micropython.org/en/latest

Ein Wiki gibt es natürlich auch:
http://wiki.micropython.org/Home

Zusammenfassung

ESP Module für kleine Anwendungen:

Fernsteuerungen im WLAN
Datenauswertungen
Sensorik und Meldungen über WLAN

Abgrenzung zum RaspberryPi

Viel kleiner und kleinere Stromaufnahme
Weniger „build in“ Bibliotheken
Schnellere Einrichtung und Konfiguration

Abgrenzung zum Arduino

Kein C mehr nötig
Micropython einfache Sprache
Runtimekompilierung

Einfache Konfiguration mit „EsPy“ zum Beispiel

Thema Internetsicherheit:

Noch keinerlei Erfahrungen
Können die Module leicht gekapert werden, wenn diese von außen zugänglich im Internet sind?

Thema Amateurfunk

Eignen sich perfekt für Fernschaltungen, Rotorsteuerungen, Überwachungen

Der Ortsverband
Salzgitter-Lebenstedt (H33)
sagt:

Danke