

ESP8266 und ESP8285 Module Anleitung



Auf dieser Seite findest du alles was man für den Einstieg wissen muss, um mit diesen Chips eigene WLAN-fähige Geräte zu bauen und zu programmieren.

Let Google [translate this article](#).

Die [ESP8266 und ESP8285](#) Chips vom chinesischen Hersteller Espressif sind 32bit Mikrocontroller mit integrierter Wi-Fi Schnittstelle. Ihre Firmware basiert auf dem [Lightweight IP stack](#), der ursprünglich von Adam Dunkels entwickelt wurde. Es werden zahlreiche handliche Module mit diesen Mikrochips verkauft.

Der Unterschied zwischen den beiden Chips ist, dass der ESP8266 einen externen Flash Speicher benötigt, während der neuere ESP8285 über 1 MByte internem Flash Speicher verfügt.

Normalerweise werden die kleinen ESP-Module mit der sogenannten "AT-Firmware" verkauft, die es ermöglicht, sie so ähnlich wie analoge Modems oder [Bluetooth Module](#) zu benutzen. Der seriell angeschlossene Mikrocontroller sendet dann Befehle an das Modul, um Verbindungen aufzubauen und Daten zu übertragen. In meinem Buch [Einstieg in die Elektronik](#) zeigt, wie man auf diese Weise einen kleinen Webserver zum Fernsteuern einer LED aufbaut.

Man kann aber auch eigene Programme direkt durch den ESP Chip ausführen lassen. Siehe [hier](#) [unten](#).

Mit Preisen um 2 Euro sind die ESP Module eine sehr preisgünstige Netzwerk-Alternative.

Technische Daten

(gilt für beide Chips)

- Stromversorgung:
 - Spannung: 2,5 bis 3,6 V
 - Durchschnittlich: 80 mA
 - Beim senden: 400 mA
 - Tiefschlaf: ca. 25 μ A¹
- Schnittstellen:
 - SPI Slave, I²S, 2-UART (der zweite kann nur Ausgabe)
 - 11 oder 13 programmierbare I/O Pins max. 12 mA
 - 1 analoger Eingang 0 bis 1 V mit 10 Bit Auflösung
- Netzwerk:
 - Wi-Fi 802.11 b/g/n 2,4 GHz mit WPA/WPA2 PSK
 - IPv4, ab SDK 2.0 und Arduino Core 2.5.0 auch IPv6
 - UDP und TCP, maximal 5 gleichzeitige Verbindungen
 - Broadcasts werden nicht unterstützt
 - Durchsatz: 150 bis 300 kByte/s in beide Richtungen²
 - Latenz: 4 bis 10 ms²
 - Soft-AP für bis zu 4 Clients³

Gliederung

- Einleitung
- Technische Daten
- Module und Boards
 - ESP-1 und ESP-01 Modul
 - ESP-07 und ESP-12 Modul
 - NodeMCU Board
 - Wemos D1 Mini Board
 - WiFi Kit 8 Board
- Stromversorgung
 - Betrieb an Batterien/Akkus
 - Deep Sleep Modus
 - Power Down Modus
 - Stromaufnahme im Reset
- USB-UART Kabel
- Anschluss an 5 V Mikrocontroller
- Flash Speicher
 - Performance
- Firmware Hochladen
- Original AT-Firmware
 - AT Befehle
 - TCP Server
 - TCP Client
 - UDP Protokoll
 - Soft-AP Modus
- Programmierung
 - Erste Schritte mit Arduino
 - Beispiel Sketche
 - LED Blinker
 - UDP Server
 - TCP Server
 - WiFi Config Service
 - WLAN aus schalten
 - Fallstricke
- Andere IDE als besseren Text-Editor verwenden
 - Qt-Creator
 - Netbeans
- Kritik
- Vergleich mit ESP32

¹) Einschließlich Flash Speicher.

²) Wenn der Funk-Kanal frei ist. Auf stark genutzten Funk-Kanälen sind Latenzen bis 200 ms normal.

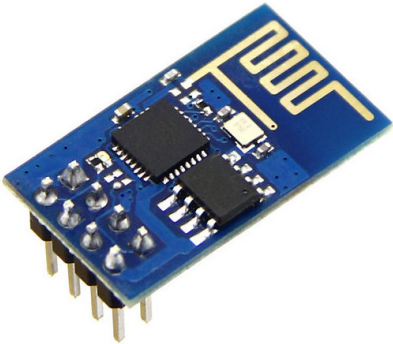

3) Im Soft-AP Modus können die WLAN Clients nur mit dem ESP kommunizieren, nicht miteinander.

Die Basis-Firmware belegt etwa 250 kByte vom Flash Speicher. Für eigene Anwendungen sind ca. 50 kByte RAM frei.

Module und Boards

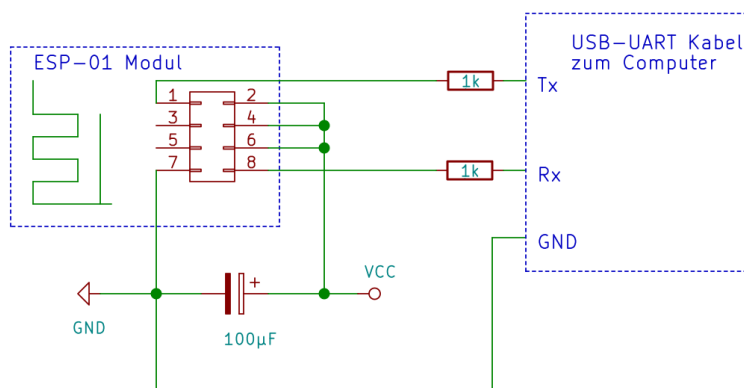
Der CHIP_EN Pin wird auch CHIP_PU, CH_PD, ENABLE oder kurz EN genannt.

ESP-1 und ESP-01 Modul

	
<p>Das ESP-01 Modul basiert auf dem ESP8266. Es wird mit 512 kByte (in blau) oder 1 MByte (in schwarz) Flash Speicher verkauft. Die rote Power-LED nimmt etwa 0,5 mA Strom auf, die blaue LED hängt an GPIO2.</p>	<p>Das ESP-1 Modul basiert auf dem ESP8285. Es hat 1 MByte Flash Speicher und einen Deckel zur Abschirmung. Das ESP-1 Modul hat nur eine blaue LED an GPIO2.</p>

Beide Module sind funktional identisch, das ESP-1 ist allerdings etwas langsamer. Die kleine 8-Polige Stiftleiste ist das auffälligste Merkmal dieser Module. Dadurch lassen sie sich leicht mit einem **kurzen** Flachkabel absetzen, um den Empfang zu optimieren.

Die Minimal-Beschaltung zur Nutzung der AT-Firmware sieht so aus:



Modul Pin	Arduino Pin	Name	Beschreibung
1	3	RxD (GPIO3)	serieller Eingang oder normaler I/O Pin
2		VCC	Spannungsversorgung 3,3 V

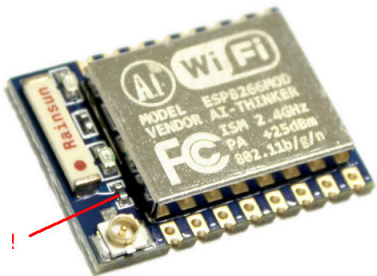

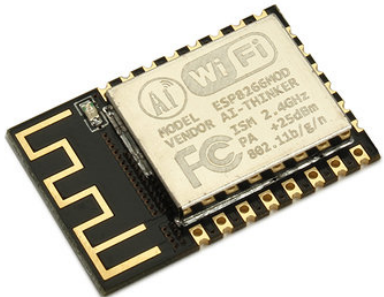
3	0	GPIO0	Low beim Start aktiviert Firmware-Upload, muss zum normalen Start offen sein oder auf High liegen
4		RESET	Low=Reset, hat schwachen Pull-Up Widerstand auf High
5	2	GPIO2 (TxD1)	Ist mit der blauen LED verbunden, die bei LOW Pegel leuchtet. Flackert beim Start. Darf beim Start nicht auf Low gezogen werden. Ausgang vom zweiten UART, der nur Ausgabe kann.
6		CHIP_EN	Muss auf High gezogen werden, damit der Chip arbeitet. Low=Chip Power Down, High=Chip Enabled
7		GND	Masse
8	1	TxD (GPIO1)	serieller Ausgang oder normaler I/O Pin, ist mit der blauen LED verbunden, flackert beim Start, darf beim Start nicht auf Low gezogen werden

Die Anschlussleisten beider Module haben 2,54 mm Raster.

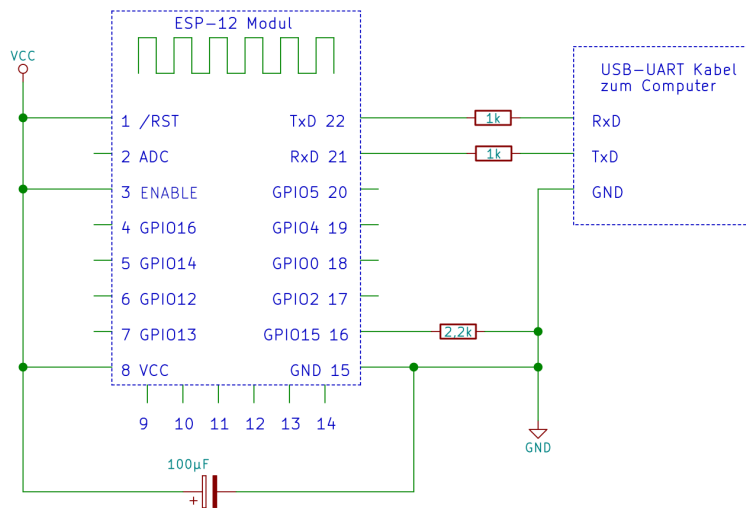
GPIO15 ist fest mit GND verbunden. Passe auf, dass du ihn nicht versehentlich als Ausgang mit High Pegel programmierst, denn der Kurzschluss könnte den Pin so zerstören, dass der Chip nicht mehr startet.

Die Reichweite der primitiven Antenne (Leiterschleife) ist mit Smartphones vergleichbar.

ESP-07 und ESP-12 Module

		
<p>Das ESP-07 Modul ist normalerweise mit einem 1 MByte Flash Chip, einer Keramikantenne und einer Steckbuchse für externe Antennen ausgestattet. Um die externe Antenne nutzen zu können, muss man den gezeigten Kondensator entfernen. Die rote Power-LED nimmt etwa 0,5 mA Strom auf. Die blaue LED hängt an GPIO2.</p>	<p>Das ESP-07s Modul hat ebenfalls 1 MByte Flash, aber keine LEDs und keine Antenne. Es kann nur mit externer Antenne betrieben werden.</p>	<p>Das ESP-12 Modul ist meistens mit 4 MByte Flash ausgestattet. Als Antenne dient hier eine aufgedruckte Leiterschleife. Die blaue LED hängt an GPIO2, das Modul hat keine Power-LED.</p> <p>Das abgebildete Modell ESP-12F hat eine etwas bessere Antenne, als das 12E Modell.</p>

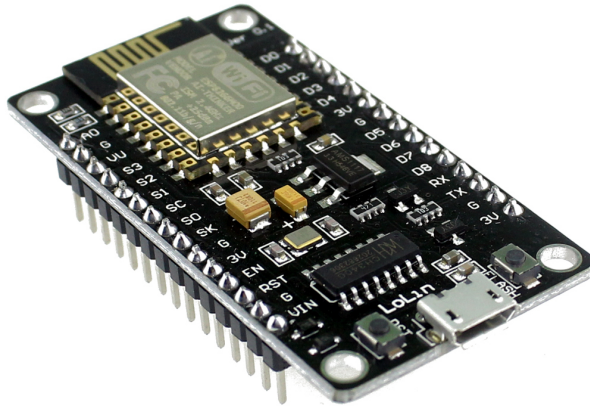
Die Minimal-Beschaltung zur Nutzung der AT-Firmware sieht für diese Module so aus:



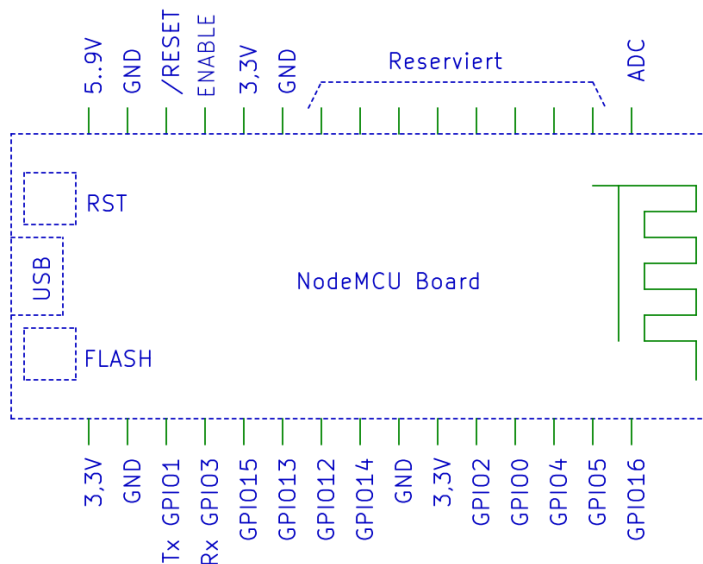
Modul Pin	Arduino Pin	ESP Name	Beschreibung
1		$\overline{\text{RESET}}$	Low=Reset, hat schwachen Pull-Up Widerstand auf High
2	17	ADC	Analoger Eingang 0 bis 1 V
3		CHIP_EN	Muss auf High gezogen werden, damit der Chip arbeitet. Low=Chip Power Down, High=Chip Enabled
4	16	GPIO16	Wenn mit $\overline{\text{RESET}}$ verbunden, kann ein Timer den μC aus dem Deep-Sleep aufwecken.
5	14	GPIO14 (SCK)	Normaler I/O Pin oder SPI Takt
6	12	GPIO12 (MISO)	Normaler I/O Pin oder SPI Daten
7	13	GPIO13 (MOSI)	Normaler I/O Pin oder SPI Daten
8		VCC	Spannungsversorgung 3,3 V
9			Durch den Flash Speicher belegt, nicht verwendbar*, nur beim ESP-12E und ESP-12F vorhanden. *) Wenn man den Flash Speicher im langsamen DIO Modus anspricht, kann man Pin 12 eingeschränkt als GPIO10 nutzen. Die Einschränkung ist, dass ein Low Pegel den Schreibschutz des Flash Speichers aktiviert. Pin 11 = GPIO9 kann man nicht verwenden, denn ein Low Pegel würde den Flash Speicher deaktivieren.
10			
11			
12			
13			
14			
15		GND	Gemeinsame Masse
16	15	GPIO15 (CS)	Normaler I/O Pin oder SPI Chip Select, muss beim Start auf Low gezogen werden, flackert beim Start
17	2	GPIO2 (TxD1)	Ist mit der blauen LED verbunden, die bei LOW Pegel leuchtet. Flackert beim Start. Darf beim Start nicht auf Low gezogen werden. Ausgang vom zweiten UART, der nur Ausgabe kann.
18	0	GPIO0	Low beim Start aktiviert Firmware-Upload, muss zum normalen Start offen sein oder auf High liegen
19	4	GPIO4	Normaler I/O Pin
20	5	GPIO5	Normaler I/O Pin
21	3	RxD (GPIO3)	serieller Eingang oder normaler I/O Pin
22	1	TxD (GPIO1)	serieller Ausgang oder normaler I/O Pin, flackert beim Start, darf beim

Die Anschlussleisten beider Module haben 2 mm Raster. Die Reichweite der on-board Antenne ist mit Smartphones vergleichbar. Mit einer guten externen Antenne soll das ESP-07 sogar weiter kommen. Der Antennen-Stecker heisst SMT, U.FL oder IPEX. Achtung: die gibt es in unterschiedlichen Größen!

NodeMCU Board



Das [NodeMCU](#) Board besteht aus einem ESP-12 Modul mit Spannungsregler und USB-UART Interface. Der Flash Speicher ist 4 Megabytes groß. Für die ersten Programmversuche ist das die ideale Ausstattung.



Board J2	NodeMCU Label	Arduino Pin	ESP Name	Beschreibung
1	VIN			Spannungsversorgung 5 bis 9 V
2	G		GND	Gemeinsame Masse
3	RST		$\overline{\text{RESET}}$	Hat 12 k Ω Pull-Up Widerstand und einen Taster mit 470 Ω nach GND
4	EN		CHIP_EN	Low=Chip Power Down, High=Chip Enabled, hat 12 k Ω Pull-Up
5	3V		VCC	Spannungsversorgung 3,3 V
6	G		GND	Gemeinsame Masse

7		Durch den Flash Speicher belegt, nicht verwendbar *		
8				
9				
10				
11				
12				
13		Reserviert		
14				
15	A0	17	ADC	Analoger Eingang für 0 bis 3,2 V weil mit Spannungsteiler 220 kΩ + 100 kΩ

Board J1	NodeMCU Label	Arduino Pin	ESP Name	Beschreibung
1	3V		VCC	Spannungsversorgung 3,3 V
2	G		GND	Gemeinsame Masse
3	TX (D10)	1	TxD (GPIO1)	Serieller Ausgang des ESP über 470 Ω mit dem USB-UART verbunden, flackert beim Start, darf beim Start nicht auf Low gezogen werden
4	RX (D9)	3	RxD (GPIO3)	Serieller Eingang des ESP über 470 Ω mit dem USB-UART verbunden
5	D8	15	GPIO15 (CS)	Normaler I/O Pin oder SPI Chip Select, muss beim Start Low sein, hat 12 kΩ Pull-Down Widerstand, flackert beim Start
6	D7	13	GPIO13 (MOSI)	Normaler I/O Pin oder SPI Daten
7	D6	12	GPIO12 (MISO)	Normaler I/O Pin oder SPI Daten
8	D5	14	GPIO14 (SCK)	Normaler I/O Pin oder SPI Takt
9	G		GND	Gemeinsame Masse
10	3V		VCC	Spannungsversorgung 3,3 V
11	D4	2	GPIO2 (TxD1)	Ist mit der blauen LED verbunden, die bei LOW Pegel leuchtet. Flackert beim Start. Muss beim Start high sein. Hat 12 kΩ Pull-Up Widerstand. Ausgang vom zweiten UART, der nur Ausgabe kann.
12	D3	0	GPIO0	Low beim Start aktiviert Firmware-Upload, hat 12 kΩ Pull-Up Widerstand und einen Taster mit 470 Ω nach GND, flackert beim Start
13	D2	4	GPIO4	Normaler I/O Pin
14	D1	5	GPIO5	Normaler I/O Pin
15	D0	16	GPIO16	Ist mit der blauen LED verbunden, die bei Low leuchtet. Die optionale Brücke R3 verbindet diesen Pin mit RESET, damit kann ein Timer den µC aus dem Deep-Sleep aufwecken.

In Arduino kann man die NodeMCU Labels nutzen, ich empfehle allerdings, die Arduino Pin Nummern zu verwenden da sie den GPIO Nummern des Chips entsprechen.

Man kann es wahlweise am 3,3V Eingang mit 2,5 bis 3,6 V betreiben, oder am 5 V Eingang mit 5 bis 9 V, oder über das USB Kabel mit 5 V. Eine Diode verhindert, dass Strom vom 5V Eingang zum USB Anschluss fließt. Der Eingebaute Spannungsregler hat für externe Erweiterungen maximal 300 mA Leistungsreserve bei 5 V Eingangsspannung. Für Batteriebetrieb eignet sich das Board eher nicht, da es eine Ruhestromaufnahme von etwa 15 mA hat.

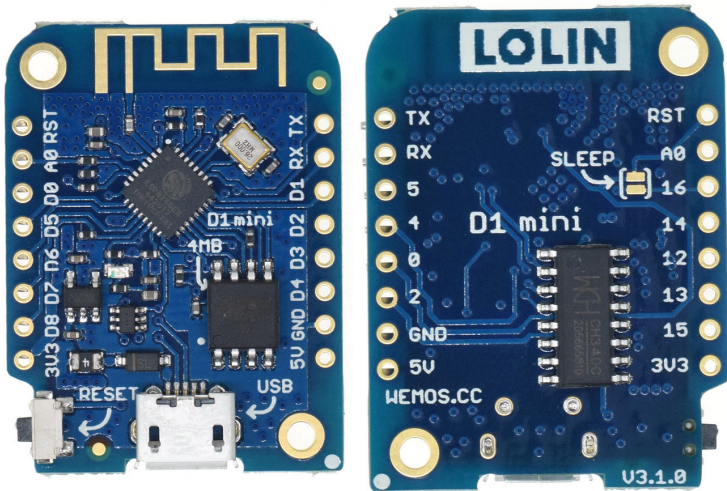
Auf einem Steckbrett läuft das Board zuverlässiger, wenn man die Stifte entfernt, die zum Flash Speicher führen.

Die Reset Leitung und GPIO0 können durch den USB-UART angesteuert werden. Damit aktiviert die Arduino IDE den Firmware-Upgrade Modus vollautomatisch:

DTR	RTS	RESET	GPIO0
Aus	Aus	High	High
Ein	Ein	High	High
Aus	Ein	Low	High
Ein	Aus	High	Low

Dafür sind die beiden über Kreuz verbundenen Transistoren im [Schaltplan des NodeMCU Boardes](#) zuständig. Alternativ kann man den Firmware-Upgrade Modus manuell aktivieren, indem man beide Tasten drückt und dann den Reset-Taster zuerst loslässt.

Wemos D1 Mini Board



Das Wemos D1 Mini Board gibt es in folgenden Versionen:

Version	ESP Chip	Flash MByte	LED an GPIO2	Antenne	Lithium Laderegler
Wemos D1 Mini Lite v1	ESP8285	1	ja	Leiterschleife	nein
Wemos D1 Mini v2	ESP-12 Modul (=ESP8266)	4	nein	Leiterschleife	nein
Wemos D1 Mini v3	ESP8266	4	ja	Leiterschleife	nein
Wemos D1 Mini Pro v1 (oder -16)	ESP8266	16	ja	Keramikantenne und ext. Antennenanschluss	nein
Wemos D1 Mini Pro	ESP8266	16	ja	Keramikantenne und ext.	ja

v2				Antennenanschluss	
----	--	--	--	-------------------	--

Die oberen vier Boards haben die gleichen Abmessungen, das letzte mit Laderegler ist etwas länger. Alle Versionen haben einen USB-UART (CH-340) und einem Low-Drop Spannungsregler. Alle freien I/O Pins des ESP Chips sind herausgeführt, aber nicht der CHIP_EN Pin.

Die Pinbelegung ist bei allen Wemos D1 Mini Boards gleich:

Board Pin	NodeMCU Label	Arduino Pin	ESP Name	Beschreibung
1	RST		$\overline{\text{RESET}}$	Hat 10 k Ω Pull-Up Widerstand und einen 100 nF Kondensator nach GND
2	A0	17	ADC	Analoger Eingang für 0 bis 3,2 V weil mit Spannungsteiler 220 k Ω + 100 k Ω
3	D0	16	GPIO16	Wenn mit $\overline{\text{RESET}}$ verbunden, kann ein Timer den μC aus dem Deep-Sleep aufwecken. Die Versionen Mini v3 und Pro v2 haben dazu auf der Rückseite eine "Sleep" Lötbrücke.
4	D5	14	GPIO14 (SCK)	Normaler I/O Pin oder SPI Takt
5	D6	12	GPIO12 (MISO)	Normaler I/O Pin oder SPI Daten
6	D7	13	GPIO13 (MOSI)	Normaler I/O Pin oder SPI Daten
7	D8	15	GPIO15 (CS)	Normaler I/O Pin oder SPI Chip Select, muss beim Start Low sein, hat 10 k Ω Pull-Down Widerstand, flackert beim Start
8	3V3		VCC	Spannungsversorgung 3,3 V
9	5V			Spannungsversorgung 4 bis 7 V
10	G		GND	Gemeinsame Masse
11	D4	2	GPIO2 (TxD1)	Alle außer Mini v2 haben eine blaue LED, die bei Low Pegel leuchtet. Flackert beim Start. Muss beim Start high sein. Hat 10 k Ω Pull-Up Widerstand. Ausgang vom zweiten UART, der nur Ausgabe kann.
12	D3	0	GPIO0	Low beim Start aktiviert Firmware-Upload, hat 10 k Ω Pull-Up Widerstand, flackert beim Start
13	D2	4	GPIO4	Normaler I/O Pin
14	D1	5	GPIO5	Normaler I/O Pin
15	RX (D9)	3	RxD (GPIO3)	Serieller Eingang des ESP über 470 Ω mit dem USB-UART verbunden
16	TX (D10)	1	TxD (GPIO1)	Serieller Ausgang des ESP über 470 Ω mit dem USB-UART verbunden, flackert beim Start, darf beim Start nicht auf Low gezogen werden

In Arduino kann man die NodeMCU Labels nutzen, ich empfehle allerdings, die Arduino Pin Nummern zu verwenden da sie den GPIO Nummern des Chips entsprechen.

Man kann es wahlweise am 3,3V Eingang mit 2,5 bis 3,6 V betreiben, oder am 5 V Eingang mit 3 bis 7 V, oder über das USB Kabel mit 5 V. Eine Diode verhindert, dass Strom vom 5V Eingang zum USB Anschluss fließt.

Die Ruhestromaufnahme des Boardes liegt je nach Variante zwischen 200 und 300 μA , so dass es für Batteriebetrieb bedingt geeignet ist. Der Spannungsregler reicht gerade für den WLAN Chip aus, er darf extern nur minimal belastet werden (50 mA bei 5 V).

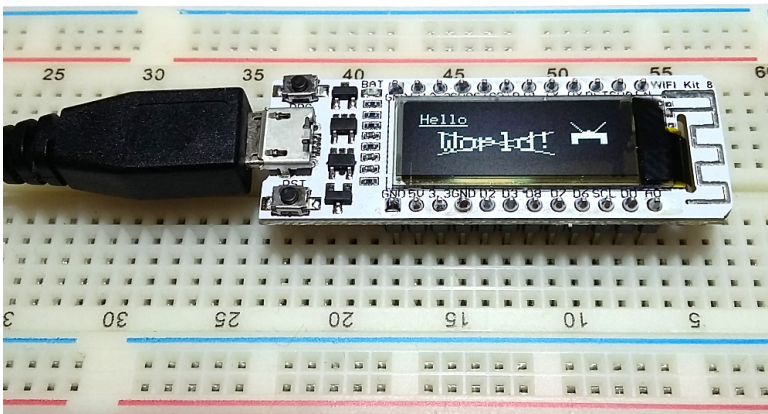
Der Reset Taster zieht den Reset-Pin ohne Schutzwiderstand direkt auf GND. Außerdem besitzt das Board die gleiche Reset Schaltung wie das [NodeMCU](#) Board, zum automatischen Aktivieren des Firmware-Upgrade Modus. Das Wemos D1 Mini hat keinen "Flash" Taster.

Der Analoge Eingang A0 ist mit einem Spannungsteiler (220kΩ + 100kΩ) ausgestattet, damit man Spannungen bis zu 3,2 V messen kann.

[Dokumentation des Herstellers.](#)

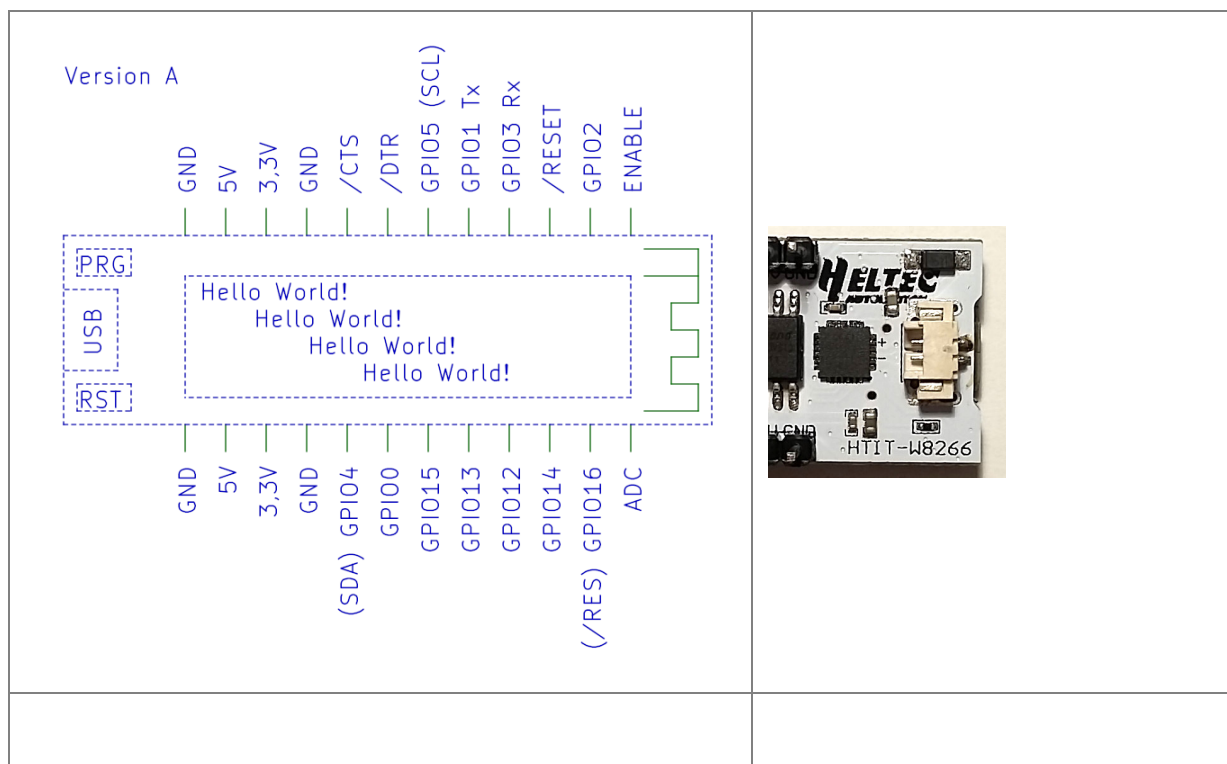
Achtung: Es sind [schlechte Nachbauten](#) mit zu schwachem Spannungsregler im Umlauf.

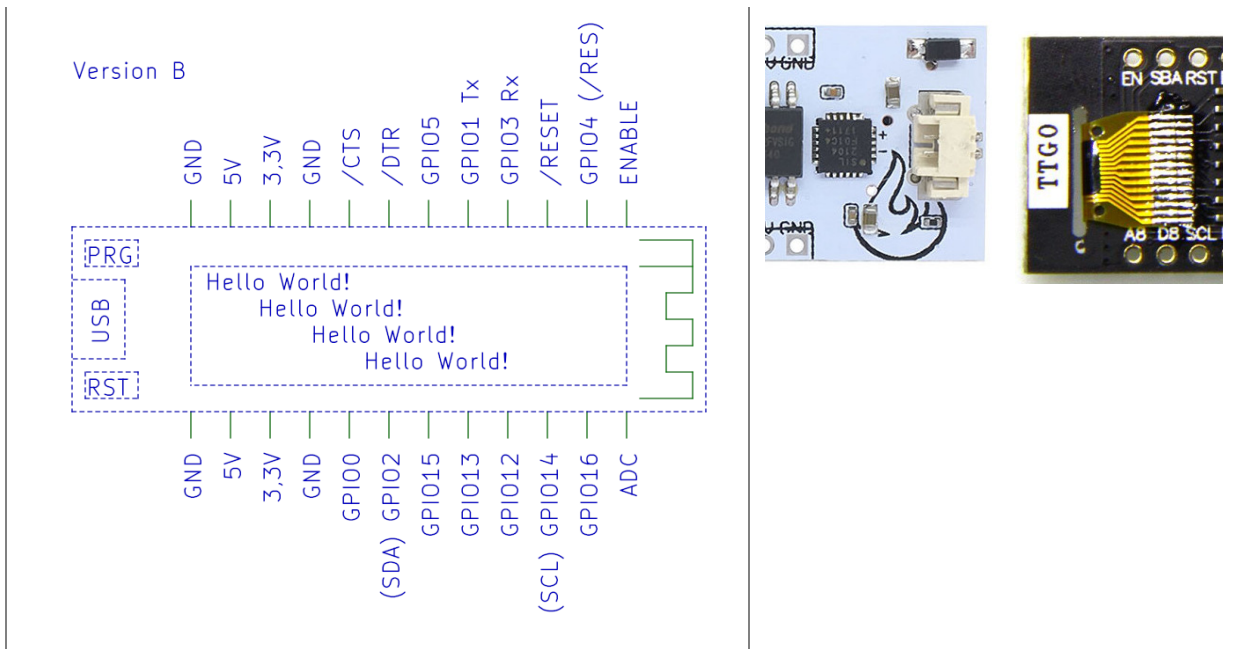
WIFI Kit 8 Board



Das WIFI Kit 8 (alias Heltec HTIT-W8266 oder Wemos TTGO ESP8266) aus China hat 4 MByte Flash Speicher, sowie ein winzig kleines 0,91" OLED Display mit 128x32 Pixeln. Darauf kann man Grafiken und 4x21 Zeichen Text ausgeben. Der Display Controller vom Typ SSD1306 ist über I²C mit dem ESP8266 verbunden. Für die USB Buchse wurde der Chip CP2104 von Silabs verwendet.

Viele Händler verkaufen diese Boards mit falsch beschrifteten Pins und falschen Anschlussplänen! Die folgenden Zeichnungen zeigen die richtige Pinbelegung, und in Klammern die Verbindungen zum Display:





Obere Stiftleiste:

Stift	Version A	Version B	Arduino Pin	Beschreibung
1	GND			
2	5 V			Spannungsversorgung 5 V 430 mA
3	3,3 V			Spannungsversorgung 3,3 V 430 mA
4	GND			Gemeinsame Masse
5	/CTS			Clear-To-Send Eingang vom USB-UART, Low aktiv
6	/DTR			Data-Terminal-Ready Ausgang vom USB-UART, Low aktiv
7	GPIO5		5	Version A: I ² C Takt zum Display mit 10 kΩ Pull-Up Widerstand Version B: Normaler I/O Pin.
8	TxD (GPIO1)		1	Serieller Ausgang des ESP oder normaler I/O Pin, direkt mit dem USB-UART verbunden, flackert beim Start, darf beim Start nicht auf Low gezogen werden
9	RxD (GPIO3)		3	Serieller Eingang des ESP, direkt mit dem Ausgang des USB-UART verbunden, daher nicht als GPIO verwendbar.
10	RESET			Hat 1,2 kΩ Pull-Up Widerstand und einen Taster nach GND
11	GPIO2 (TxD1)	GPIO4	2 bzw. 4	Version A: Normaler I/O Pin, flackert beim Start, darf beim Start nicht auf Low gezogen werden. Ausgang vom zweiten UART, der nur Ausgabe kann. Version B: Reset Signal zum Display, Low aktiv.
12	CHIP_EN			Muss auf High liegen, damit der Chip arbeitet. Hat 10 kΩ Pull-Up Widerstand. Low=Power Down, High=Enabled

Untere Stiftleiste:

Stift	Version A	Version B	Arduino Pin	Beschreibung
1	GND			Gemeinsame Masse

2	5 V		Spannungsversorgung 5 V 430 mA
3	3,3 V		Spannungsversorgung 3,3 V 430 mA
4	GND		Gemeinsame Masse
5	GPIO4	GPIO0	4 bzw. 0 Version A: I ² C Daten zum Display mit 10 kΩ Pull-Up Widerstand. Version B: Low beim Start aktiviert Firmware-Upload, hat 10 kΩ Pull-Up Widerstand und einen Taster nach GND, flackert beim Start
6	GPIO0	GPIO2 (TxD1)	0 bzw. 2 Version A: Low beim Start aktiviert Firmware-Upload, hat 10 kΩ Pull-Up Widerstand und einen Taster nach GND, flackert beim Start. Version B: Normaler I/O Pin, flackert beim Start, darf beim Start nicht auf Low gezogen werden. Ausgang vom zweiten UART, der nur Ausgabe kann.
7	GPIO15 (CS)		15 Normaler I/O Pin oder SPI Chip Select, muss beim Start Low sein, hat 10 kΩ Pull-Down Widerstand, flackert beim Start
8	GPIO13 (MOSI)		13 Normaler I/O Pin oder SPI Daten
9	GPIO12 (MISO)		12 Normaler I/O Pin oder SPI Daten
10	GPIO14 (SCK)		14 Version A: Normaler I/O Pin oder SPI Takt. Version B: I ² C Takt zum Display mit 10 kΩ Pull-Up Widerstand
11	GPIO16		16 Version A: Setzt das Display bei Low Pegel zurück. Beide Versionen: Wenn mit RESET verbunden, kann ein Timer den µC aus dem Deep-Sleep aufwecken.
12	ADC		17 Analoger Eingang für 0 bis 3,2 V weil mit Spannungsteiler 220 kΩ + 100 kΩ. In Arduino heißt der Pin A0 oder Nummer 17.

Die [Infos von Heltec](#) passen zur Version A, sind aber unvollständig und fehlerhaft.

Die Stromversorgung erfolgt wahlweise über USB, ein 3,3 V Netzteil oder ein 5 V Netzteil. Das Display nimmt zusätzlich zum ESP Chip ca. 30 mA auf. Der eingebaute Spannungsregler hat für Erweiterungen maximal 200 mA Leistungsreserve. Da der 5 V Anschluss direkt mit der USB Buchse verbunden ist, soll bei Nutzung von USB nicht gleichzeitig ein 5 V Netzteil verwendet werden.

Das Board eignet sich nur bedingt für Akku-Betrieb, da seine Ruhestromaufnahme nie unter 6 mA fällt. Unterhalb von 3,3 V Akku-Spannung fällt das Board aus und saugt dann den Akku mit 80 mA leer, bis er entweder kaputt geht oder seine integrierte Schutzschaltung (falls vorhanden) abschaltet. Beim Aufladen wird die Platine ziemlich warm, was für die Lebensdauer des Displays schlecht ist.

Die Reset Methode zum automatischen Aktivieren des Firmware-Upgrade Modus entspricht dem [NodeMCU](#) Board. Die beiden Taster ziehen die Leitungen RESET und GPIO0 direkt ohne Schutzwiderstände auf Low. Auch der USB-UART ist direkt mit den I/O Pins für TxD und RxD verbunden, deswegen kann man den RxD Pin (=GPIO3) nicht für andere Zwecke verwenden.

Ich hatte beim Experimentieren auf einem Steckbrett äußerst schlechten WLAN Empfang. Durch Verlängern der Anschlussbeine um 7 mm konnte ich das Problem beheben.

Zur Programmierung des Displays empfehle ich entweder meine [OLED Klasse](#) oder die Libraries [SSD1306](#) und [GFX](#) von Adafruit. Die Adafruit Library muss allerdings für Version B des Moduls angepasst werden (Pin Nummern als Parameter zu Wire.begin() hinzufügen).

Man sollte die geringe Lebensdauer des OLED beachten. Die häufiger benutzten Pixel werden schon nach einem Jahr deutlich sichtbar schwächer, das steht auch so im Datenblatt des Displays. Das Display hält länger, wenn man die Helligkeit reduziert.

Da ich öfters danach gefragt wurde: Hier ist ein [primitives Beispiel](#), wie man Datum und Uhrzeit von einem beliebigen Webserver abrufen kann.

Stromversorgung

Zu schwache Stromversorgung ist die häufigste Ursache von sporadischen Fehlfunktionen, sie kann sogar zur [Zerstörung des Chips](#) führen. Der ESP Chip benötigt für zuverlässigen Betrieb eine stabile Spannungsversorgung zwischen 2,8 und 3,6 Volt bei 10µA bis 400 mA. Innerhalb dieses Bereiches darf sich die Spannung nur langsam ändern. Es ist also ein starker Spannungsregler erforderlich, der [extreme Strom-Schwankungen](#) regelt, ohne dabei instabil zu werden.

Der oft anzutreffende Spannungsregler [AMS1117](#) ist zwar stark genug, aber er benötigt mindestens 11mA Belastung, was die Nutzung von Deep-Sleep und Power-Down Modi ausschließt. Ich bevorzuge daher den [LF33](#).

Egal woher die Stromversorgung kommt, empfehle ich dringend einen 100 µF Kondensator direkt an die Anschlüsse VCC und GND des ESP-Moduls zu löten. Beim NodeMCU Board ist dieser bereits vorhanden.

Die Leitungen der Stromversorgung sollen möglichst kurz sein und sternförmig an einem Verteil-Punkt zusammen kommen. Das gilt ganz besonders für GND. Steckbretter sind aufgrund der hohen Kontaktwiderstände schlecht zur Strom-Verteilung geeignet.

Die Widerstände um den Chip herum sollten maximal 2,2 kΩ haben, denn hochohmige Widerstände sind für die Funkwellen empfänglich. In der Nähe der Antenne sind Metallteile (auch Leiterbahnen) zu vermeiden, da sie den Empfang beeinträchtigen.

Betrieb an Batterien/Akkus

Alle fertigen Module mit USB Schnittstelle und Spannungsregler auf dem Board eignen sich eher nicht für Langzeit-Betrieb an Batterien, da sie mehr als 1 mA Ruhestrom aufnehmen. Hier sind minimale Module ohne Schnickschnack (wie das ESP-12F) die bessere Wahl.

Ich empfehle dazu folgende Batterien:

- Eine LiFePO4 Zelle, deren Spannung passt schon ohne Regler
- Eine Lilo oder LiPo Zelle mit 3,0 Volt Spannungsregler
- 4 NiMH Zellen mindestens in Größe AA mit 3,3 Volt Spannungsregler
- 4 alkalische Primärzellen mindestens in Größe AA mit 3,3 Volt Spannungsregler

Für Batterien bis maximal 6 Volt empfehle ich den Spannungsregler [HT7830 oder HT7833](#) mit 4 µA Ruhestrom. Gelegentlich werden [MCP1700](#) oder [MCP1702](#) empfohlen, aber die sind meiner Meinung nach zu knapp bemessen.

Wenn der ESP Chip wegen Unterspannung ausfällt, nimmt er ständig ca. 80 mA auf. Lithium- und Blei-Akkus müssen daher unbedingt mit einem Tiefentladeschutz ausgestattet werden. Manche Lithium-Akkus enthalten so eine Schutz-Schaltung bereits.

Tipp: Es gibt zahlreiche andere Funktechniken, die wesentlich weniger Energie benötigen, als WLAN.

Deep-Sleep Modus

Im Deep-Sleep Modus beträgt die Stromaufnahme des ESP Chips (incl. Flash) etwa 25 µA. Die WLAN Verbindung wird dabei unterbrochen. Die RTC (Uhr) läuft weiter, allerdings ungenauer als im Normalbetrieb (typisch 2 Minuten Abweichung pro Tag).

Man aktiviert den Deep-Sleep Modus durch Aufrufen der Funktion **ESP.deepSleep(µs)** gefolgt von delay(100). Als Parameter (µs) gibt man an, in wie vielen Mikrosekunden der Chip wieder aufwachen soll.

Maximal 71 Minuten sind möglich. Der Wert 0 lässt den Chip für immer schlafen, oder bis zum nächsten Hardware-Reset.

Da der Timer von einem temperaturabhängigen R/C Oszillator getaktet wird, kann die Zeit ein paar Prozent vom Soll abweichen.

Wenn man den Wakeup-Timer benutzt, muss der Timer-Ausgang GPIO16 mit dem $\overline{\text{RESET}}$ Eingang verbunden werden, sonst hängt der Chip nach dem Aufwachen. Da die Firmware beim Aufwachen neu startet, geht der Inhalt des RAM verloren. Die RTC enthält jedoch 512 Byte zusätzlichen Speicher, der den Deep-Sleep Modus (jedoch nicht Power-Down) überlebt.

Da beim Aufwachen eine neue WLAN Verbindung aufgebaut werden muss, dauert es typischerweise 3 Sekunden.

Power-Down Modus

Im Power-Down Modus ist der Chip komplett abgeschaltet. Es fließt nur noch ein kleiner Leckstrom von etwa 15 μA (incl. Flash).

Dieser Modus wird durch eine fallende Flanke am CHIP_EN Eingang ausgelöst, aber erst **nachdem** die Firmware gestartet ist. Wenn CHIP_EN zu früh auf Low gezogen wird, nimmt der Chip ca. 2,5 mA auf!

Die steigende Flanke am CHIP_EN Eingang bewirkt, dass der Power-Down Modus verlassen wird und die Firmware neu startet.

Stromaufnahme im Reset

Während der $\overline{\text{RESET}}$ Eingang auf Low gehalten wird nimmt der Chip ca. 23 mA auf. Der $\overline{\text{RESET}}$ Eingang eignet sich daher nicht zum Strom-sparen.

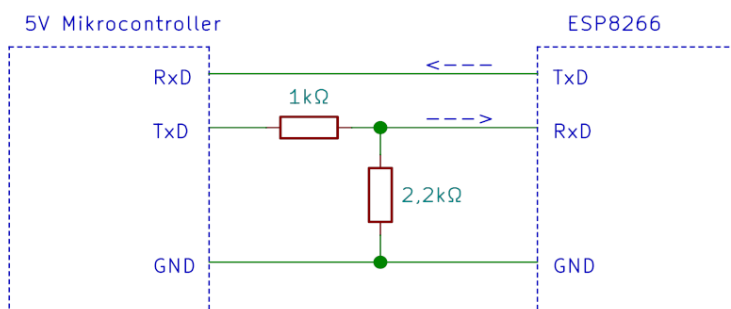
USB-UART Kabel

Für die Module ohne USB Anschluss, brauchst du ein USB-UART Kabel oder Adapter, um es mit dem PC zu verbinden. Dabei ist wichtig, dass das Ding 3,3 V Pegel hat. Ich schleife in die Signal-Leitungen immer 1 k Ω Widerstände zur Strombegrenzung ein, falls eine Seite mal keine Spannungsversorgung hat.

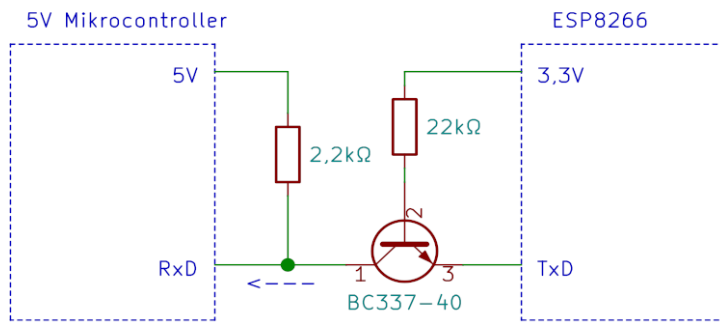
Für USB-UART Produkte mit altem oder gefälschtem PL2303 Chip habe ich [hier](#) einen passenden alten Windows Treiber, denn der aktuelle Treiber mag diese Chips nicht. Den Treiber für die chinesischen CH340 und CH341 Chips kann man direkt von der [Webseite des Herstellers](#) downloaden.

Anschluss an 5 V Mikrocontroller

Der ESP Chip verträgt maximal 3,6 V an allen Pins. Die Signale von 5 V Mikrocontrollern kann man mit einem Spannungsteiler herabsetzen.



Für die umgekehrte Richtung gilt: Die meisten 5 V Mikrocontroller (z.B. AVR) und alle 74HCT Logikgatter akzeptieren 3,3 V als gültigen High Pegel. Man kann sie also direkt verbinden. Falls du den Signalpegel dennoch auf 5V erhöhen musst, kannst du das so tun:



Diese Schaltung funktioniert mit jedem gewöhnlichen kleinen NPN Transistor.

Flash Speicher

Der ESP82xx Chip kopiert seine Firmware blockweise von einem externen Flash Speicher in sein internes RAM. Von dort aus wird der Code dann ausgeführt.

Der ESP8266 kann bis zu 16 MByte Flash Speicher adressieren, davon jedoch nur maximal 1 MByte als Programmspeicher. Ungefähr 250 kB davon belegt die Basis Firmware.

Der ESP8266 unterstützt folgende Zugriffsarten auf den Flash Speicher:

- **QIO** verwendet 4 Leitungen für Lesen und Schreiben
- **QOUT** verwendet 4 Leitungen für Lesen aber nur 1 Leitung für Schreiben
- **DIO** verwendet 2 Leitungen für Lesen und Schreiben
- **DOUT** verwendet 2 Leitungen für Lesen aber nur 1 Leitung für Schreiben

Welche der vier Varianten tatsächlich funktionieren, hängt vom eingebauten Speicherchip ab. Zum Ändern muss man den Quelltext der Firmware neu compilieren.

Der ESP8285 hat 1 MByte Flash intern, welcher nur **DOUT** mit 40 MHz unterstützt. Dafür hat der Chip zwei weitere I/O Pins frei, nämlich GPIO9 und GPIO10.

Beim Start gibt der Chip ein paar Diagnose-Meldungen mit 74880 Baud aus, noch bevor die Firmware gestartet wird. Im Fehlerfall lohnt es sich, diese Meldungen mit einem Terminal-Programm anzuzeigen.

Performance

Um einen Eindruck zu bekommen, wie stark sich die Varianten der Flash Anbindung auf die Gesamt-Performance auswirken, habe ich zwei Testreihen durchgeführt. Die erste Testreihe erfasste die beste Antwortzeit beim Empfangen+Senden von jeweils 1 kB. Die zweite Testreihe vergleicht die Anzahl von Multiplikationen in einem Zeitfenster.

CPU Freq.	Flash Freq.	Flash Mode	Antwortzeit	Rechenleistung
160 MHz	80 MHz	QIO und QOUT	4 ms	871
		DIO und DOUT	4 ms	870
	40 MHz	QIO und QOUT	4 ms	870
		DIO und DOUT	5 ms	870

80 MHz	80 MHz	QIO und QOUT	4 ms	435
		DIO und DOUT	4 ms	435
	40 MHz	QIO und QOUT	4 ms	434
		DIO und DOUT	5 ms	434

Firmware Hochladen

Der ESP Chip enthält einen unveränderlichen Bootloader, welcher ein Firmware-Upgrade über den seriellen Port ermöglicht. Beim Reset startet immer zuerst dieser Bootloader. Er erwartet folgende Steuersignale:

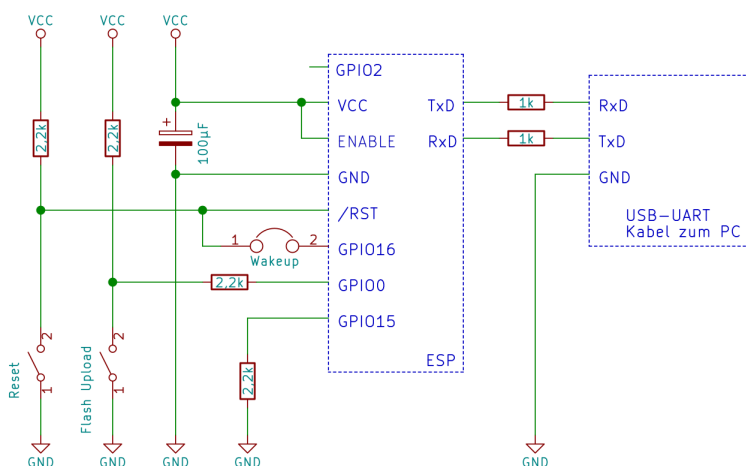
Firmware-Upload:

- GPIO0 = LOW
- GPIO15 = LOW
- GPIO1 (TxD) = nicht herunter ziehen
- GPIO2 (TxD1) = nicht herunter ziehen

Normaler Start:

- GPIO0 = HIGH
- GPIO15 = LOW
- GPIO1 (TxD) = nicht herunter ziehen
- GPIO2 (TxD1) = nicht herunter ziehen

GPIO1 und GPIO2 müssen beim Start auf HIGH Pegel liegen, was interne Pull-Up Widerstände erledigen. Der Bootloader gibt auf beiden seriellen Ports Meldungen mit 74880 Baud aus. Nach dem Start können alle vier Pins jedoch als frei programmierbare I/O Anschlüsse verwendet werden. Daraus ergibt sich die folgende sinnvolle Grundschialtung:



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

Der Widerstand vor GPIO0 schützt 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. Viele Boards haben die [Reset-Schaltung von NodeMCU](#) kopiert, mit der man sich das Drücken der Taster ersparen kann. Sie funktioniert allerdings nicht immer zuverlässig.

Es gibt zahlreiche grafische Programme zum hochladen der Firmware, aber die scheinen alle etwas zickig zu sein. Daher empfehle ich das Script [esptool.py](#), welches meine beiden weiter unten bereitgestellten Firmware-Pakete enthalten. Zum Ausführen musst Du vorher [Python](#) installieren. Achte beim Ausführen der Installation darauf dass es zur PATH Variable hinzugefügt wird. Anschließend installiere die Library für serielle Ports, indem du im CMD Fenster (Eingabeaufforderung) **pip install serial** eingibst.

Anmerkung zu Linux:

Bei Debian/Ubuntu Linux installiert man die Library hingegen mit dem befehl **sudo apt install python3-serial**. Außerdem heisst der bei Linux **python3**, nicht python.

Mit Hilfe des esptool kann man folgendermaßen die Größe des Flash Speichers anhand seiner ID Nummer ermitteln:

Windows: `python esptool.py --port COM6 flash_id`
Linux: `python3 esptool.py --port /dev/ttyUSB0 flash_id`

4013	512 kByte
4014	1 MByte
4015	2 MByte
4016	4 MByte

Der eigentliche Upload einer Firmware-Datei geschieht mit dem Befehl:

```
python esptool.py --port COM6 write_flash 0x000000 firmware.bin
```

Keine Angst: Wenn das Hochladen der Firmware (warum auch immer) fehlschlägt, kann man es einfach nochmal versuchen. Der Bootloader gibt im Fehlerfall eventuell hilfreiche Meldungen mit 74880 Baud aus.

Original AT-Firmware

Die AT-Firmware ermöglicht es, das Modul wie ein Modem zu benutzen. Durch das Absetzen von Befehlen auf der seriellen Schnittstelle konfiguriert man das Modul, baut Verbindungen auf und überträgt Daten. Das Modul dient also als Netzwerk-Adapter für einen angeschlossenen Mikrocontroller.

Auf der Seite [I/O Schnittstellen Module für WLAN](#) stelle ich eine solche Beispiel-Anwendung vor, und im Buch [Einstieg in die Elektronik mit Mikrocontrollern](#) erkläre ich detailliert, wie man so etwas macht.

Die AT Firmware sendet und empfängt im 100 ms Raster, maximal 2048 Bytes pro Intervall.

Für Module mit nur 512 kByte Flash Speicher muss man die [AT-Firmware 0.50.0.0 vom SDK 1.4.0](#) verwenden. Das ist die letzte Version, die noch dort hinein passt. Da die Firmware in mehrere Dateien aufgeteilt ist, muss man sie so installieren:

512 kByte	<code>python esptool.py --port COM6 write_flash 0x000000 noboot/eagle.flash.bin 0x040000 noboot/eagle.irom0text.bin 0x03e000 blank.bin 0x07e000 blank.bin 0x07c000 esp_init_data_default.bin</code>
------------------	---

Für alle größeren Module habe ich mir die zuverlässige [AT-Firmware 1.1.0.0 vom SDK 1.5.4](#) gesichert. Sie wird mit folgendem Befehl installiert:

1 MByte	<code>python esptool.py --port COM6 write_flash 0x000000 boot_v1.5.bin 0x001000 512+512/user1.1024.new.2.bin 0x0fc000 esp_init_data_default.bin 0x07e000 blank.bin 0x0fe000 blank.bin</code>
2 MByte	<code>python esptool.py --port COM6 write_flash 0x000000 boot_v1.5.bin 0x001000 512+512/user1.1024.new.2.bin 0x01fc000 esp_init_data_default.bin 0x07e000 blank.bin 0x1fe000 blank.bin</code>

4 MByte	<code>python3 esptool.py --port COM6 write_flash 0x000000 boot_v1.5.bin 0x001000 512+512/user1.1024.new.2.bin 0x03fc000 esp_init_data_default.bin 0x07e000 blank.bin 0x3fe000 blank.bin</code>
----------------	--

Falls deine AT-Firmware älter als 0.50.0.0 ist, rate ich zu einem Upgrade, da frühere Versionen noch ziemlich instabil liefen. Die Firma Espressif stellt ihre AT Firmware im **bin** Verzeichnis des [SDK](#) zur Verfügung. Dort befindet sich auch die Textdatei README.md, in der drin steht, welche Dateien abhängig von der Größe des Flash Speichers an welche Position geladen werden müssen.

AT Befehle

In diesem Kapitel beschreibe ich nur die Befehle, die mir wichtig sind. Sie funktionieren mindestens mit der AT-Firmware von Version 0.21.0.0 bis 3.0.2, vielleicht auch mit neueren. Beachte beim Programmieren, dass das Format der Antworten je nach Firmware Version etwas variiert.

Zum Testen von Netzwerkverbindungen empfehle ich das Kommandozeilen-Programm [Netcat](#). Außerdem solltest Du Dir unbedingt das Programm [Wireshark](#) anschauen, falls du es noch nicht kennst. Als Terminalprogramm zur manuellen Eingabe der AT Befehle empfehle das [Hammer Terminal](#), Putty oder Cutoffcom.

Die AT Firmware kommuniziert in der Regel mit 115200 Baud. Manche Module sind auf 57600 oder 9600 Baud vor-eingestellt. Das Format für Zeilenumbrüche ist CR+LF.

Nach dem Hardware-Reset zeigt das Terminal Programm einige unlesbare Zeilen an, und dann "ready". Danach kannst du Befehle eintippen. Sollten jedoch nur unlesbare Zeichen erscheinen, stelle die Baudrate auf 74880 um, dann kannst du die Fehlermeldungen des Bootloaders lesen.

Befehle:

AT

wird einfach mit "OK" beantwortet. Der Befehl hat keine weitere Funktion, er eignet sich prima, um die serielle Verbindung zu testen.

ATE0

schaltet das Echo der Befehle aus. Dieser Befehl kann die Programmierung vereinfachen und die Geschwindigkeit geringfügig verbessern. Mit der Zahl 1 schaltet man das Echo wieder an. Für manuelles Testen lässt man das Echo am besten eingeschaltet.

AT+GMR

Zeigt die Version der Firmware an, zum Beispiel:

```
AT version:1.1.0.0(May 11 2016 18:09:56)
SDK version:1.5.4(baaeeabb)
compile time:May 20 2016 15:06:44
OK
```

AT+CWMODE=1

stellt ein, dass das Modul sich in ein bestehendes WLAN Netz einbuchen soll.

AT+RST

löst einen Software-Neustart aus. Dies ist nach dem Wechsel von CWMODE notwendig.

AT+CWLAP

listet alle erreichbaren Access Points auf. Zum Beispiel:

```
+CWLAP: (4, "EasyBox-4D2D18", -72, "18:83:bf:4d:2d:b2", 2, -46)
+CWLAP: (4, "UPC2827302", -63, "88:f7:c7:52:40:9d", 6, -22)
```

```
+CWLAP:(0,"Unitymedia WifiSpot",-64,"8a:f7:c7:52:40:9f",6,-22)
+CWLAP:(3,"Pussycat",-45,"5c:49:79:2d:5b:cd",7,-4)
OK
```

AT+CWJAP="Pussycat","supersecret"

das Modul verbindet sich mit dem WLAN Netz "Muschikatz" und dem angegebenen WPA/WPA2 Passwort. Diese Einstellung wird automatisch dauerhaft gespeichert und beim nächsten Neustart aktiv. Das Passwort muss mindestens 8 Zeichen lang sein!

AT+CWJAP?

zeigt an, mit welchem Access Point das Modul gerade verbunden ist. Zum Beispiel:

```
+CWJAP:"Pussycat","5c:49:79:2d:5b:cd",7,-60
OK
```

AT+CIFSR

zeigt die IP-Adresse und MAC Adresse des Moduls an zum Beispiel:

```
+CIFSR:STAIP,"192.168.0.111"
+CIFSR:STAMAC,"5c:cf:7f:8b:a9:f1"
OK
```

AT+CIPMUX=1

Stellt ein, dass das Modul mehrere Verbindungen erlauben soll. In diesem Modus sind bis zu 5 gleichzeitige Verbindungen möglich. Die folgenden Beispiele setzen alle voraus, dass dieser Modus aktiviert wurde. Das Gegenteil wäre der AT+CIPMUX=0, wo das Modul nur als Client und mit nur einer Verbindung genutzt werden kann. In diesem Fall entfällt bei den drauf folgenden Befehlen die Kanalnummer.

AT Firmware als TCP Server benutzen

AT+CIPSERVER=1,5000

dieser Befehl schaltet den Server ein, so dass er Verbindungen auf Port 5000 annimmt. Er kann danach (je nach Firmware Version) maximal 4 oder 5 Verbindungen gleichzeitig annehmen. Danach könntest du zum Beispiel mit Netcat eine Netzwerkverbindung zu diesem Mini-Server aufbauen:

```
nc 192.168.0.111 5000
Hello
World!
```

Am seriellen Port des Moduls erscheinen dabei folgende Meldungen:

```
0,CONNECT
+IPD,0,7:Hello
+IPD,0,8:World!
```

"0,CONNECT" bedeutet, dass der TCP Server eine Verbindung auf Kanal 0 angenommen hat. "+IPD,0,7:" bedeutet, dass der Server auf Kanal 0 genau 7 Bytes (nämlich "Hello\r\n") empfangen hat. Nach dem Doppelpunkt folgen genau diese 7 Bytes. Achtung: Wenn der empfangene Text nicht mit einem Zeilenumbruch endet, dann kommt die nächste +IPD Meldung direkt dahinter in der selben Zeile!

AT+CIPSEND=0,9
> **Welcome!**

Mit diesem Befehl sendest du eine Antwort an den Computer. Die 0 ist wieder die Kanalnummer und die 9 sind die Anzahl der zu sendenden Bytes (maximal 2048). Nachdem das Modul mit ">" Antwortet, gibst du die zu sendenden Zeichen (oder Bytes) ein. Das Modul bestätigt den Sendevorgang nach kurzer Zeit mit

SEND OK

Laut Dokumentation soll man nach diesem Kommando mindestens 1 Sekunde warten. Bei meinen Tests funktionierten aber 4 Kommandos pro Sekunde weitgehend zuverlässig.

```
AT+CIPCLOSE=0
```

mit diesem Befehl schließt du die Verbindung auf Kanal 0. Das Modul antwortet mit

```
0,CLOSED  
OK
```

AT Firmware als TCP Client benutzen

```
AT+CIPSTART=0,"TCP","mail.stefanfrings.de",25
```

Kanal 0 baut eine TCP Verbindung zu dem angegebenen Server auf Port 25 auf. Anstelle des Hostnamens kannst du auch eine IP Adresse verwenden. Die Antwort lautet zum Beispiel:

```
0,CONNECT  
OK  
+IPD,0,96:220 wp039.webpack.hosteurope.de  
    ESMTP Host Europe Mail Service  
    Sat, 08 Apr 2017 23:37:19 +0200
```

Da dieser Mail-Server sofort antwortet, siehst du auch direkt eine +IPD Zeile. Der Text hinter dem Doppelpunkt ist die Antwort des Servers. Danach kann man mit AT+CIPSEND wie bereits oben beschrieben Daten an den Server senden. Mit AT+CIPCLOSE schließt man die Verbindung.

AT Firmware für UDP benutzen

Beim UDP Protokoll entfällt der Verbindungsaufbau, und damit auch die Begrenzung auf 5 gleichzeitige Verbindungen. Man sendet einzelne Nachrichten an die gewünschten Empfänger und hofft, dass sie zügig dort ankommen. Im Gegensatz zu TCP verzichtet das UDP Protokoll auf die Wiederholung von verlorenen Paketen. Es garantiert nicht einmal, dass alle Pakete in der richtigen Reihenfolge beim Empfänger ankommen (das passiert in lokalen Netzen allerdings praktisch nie).

UDP ist einfacher zu programmieren und nutzt die Übertragungsstrecke effizienter als TCP. Es eignet sich besonders gut zur regelmäßigen Übertragung von Messwerten, wenn einzelne Aussetzer tolerierbar sind (z.B. Temperaturfühler). Leider können Javascripte in Webseiten das UDP Protokoll nicht nutzen.

```
AT+CIPSTART=0,"UDP","0",0,3002,0
```

Nach diesem Befehl kann das Modul UDP Nachrichten auf Port 3002 von beliebigen Absendern empfangen. Es kann aber keine Nachrichten senden. Zum Test kannst du das Programm Netcat benutzen:

```
nc -u 192.168.0.111 3002  
Lets start!
```

Wobei du die IP-Adresse deines WLAN Moduls angeben musst. Wenn Du dann in Netcat etwas zum senden eintippst, wird das Modul den Empfang der Nachricht mit +IPD signalisieren.

```
AT+CIPSTART=0,"UDP","0",0,3002,2
```

Das Modul empfängt UDP Nachrichten auf Port 3002 von beliebigen Absendern, und es kann Antworten mit AT+CIPSEND zurück schicken. Die Antworten werden immer an den Computer gesendet, von dem zuletzt eine Nachricht empfangen wurde.

```
AT+CIPSTART=0,"UDP","192.168.0.5",3001,3002,0
```

Das Modul kann UDP Nachrichten auf Port 3002 von beliebigen Absendern empfangen. Mit dem AT+CIPSEND Befehl wird es Nachrichten an den Computer 192.168.0.5 Port 3001 senden.

Soft-AP Modus

Wo kein WLAN Netz existiert, kann das ESP-Modul selbst ein kleines provisorisches Netz aufspannen. Bis zu 4 andere Geräte können sich dann mit dem Modul verbinden, aber sie können sich nicht gegenseitig sehen. Der Soft-AP Modus eignet sich gut dazu, ESP Module untereinander zu verbinden. Die Nutzung mit Smartphones und Laptops kann ich nicht empfehlen, da sie bei vielen Geräten unzuverlässig funktioniert. Im Soft-AP Modus erreichst du das WLAN Modul mit der fest codierten IP-Adresse 192.168.4.1.

```
AT+CWMODE=2
AT+CWSAP="ESP-Modul","supersecret",5,3
AT+RST
```

Diese Befehlsfolge aktiviert den Soft-AP Modus, und stellt den Namen des Netzes und das Passwort ein. Die 5 gibt den WLAN Funk-Kanal an (0-13), und die 3 legt fest, dass WPA2 Verschlüsselung verwendet wird. Diese Einstellung wird automatisch dauerhaft gespeichert und beim nächsten Neustart aktiv. Das Passwort muss mindestens 8 Zeichen lang sein!

Es gibt auch einen Kombinierten Modus (AT+CWMODE=3), in welchem sich das Modul mit einem bestehenden WLAN Netz verbindet und außerdem ein eigenes zweites Netz aufspannt.

Möglichkeiten der Programmierung

Anstatt den ESP82xx wie ein Modem über AT-Befehle zu benutzen, kann man auch eigene Firmware schreiben, die direkt auf dem Mikrocontroller des ESP Moduls läuft. Der chinesische Hersteller Espressif stellt dazu ein Software development Kit ([SDK](#)) bereit. Dazu muss man sich allerdings gut mit Linux auskennen und mit unvollständiger Dokumentation arbeiten.

Das [NodeMCU](#) Projekt bemüht sich darum, die Programmierung stark zu vereinfachen, indem man auf [LUA](#) Scripte setzt. Allerdings leidet dieser Lösungsansatz an notorischem Speichermangel. Das eigene LUA Script muss samt Daten (Variablen) in ca. 40 kByte RAM gequetscht werden. Scrolle auf der [Projektseite](#) nach unten, dort gibt es einen Link zu dem "custom firmware build service", wo man sich eine individuell zusammengestellte Firmware erzeugen lässt.

Die Bastel-freundlichste Lösung ist meiner Meinung nach die [ESP8266 Erweiterung](#) für [Arduino](#). Man programmiert dort in C++. Die Installation der Arduino IDE ist kinderleicht - kein Vergleich zum Espressif SDK.

Alle drei Varianten haben eines gemeinsam: Das Debuggen mit einer IDE ist nur sehr eingeschränkt möglich. In der Praxis wird man eher mit seriellen Log-Meldungen arbeiten, wie das bei Arduino ohnehin üblich ist.

Erste Schritte mit Arduino

Der Arduino Core kombiniert das SDK des Herstellers mit dem Arduino Framework, um die Programmierung in C++ zu erleichtern. Das damit erzeugte Compilat enthält daher immer die originale Firmware des Herstellers plus dein eigenes Anwendungsprogramm.

Installation:

- Downloade die aktuelle [Arduino IDE](#) und starte sie.
- Gehe ins Menü "Datei/Voreinstellungen". Gebe ins Feld "Zusätzliche Boardverwalter-URLs" die Adresse http://arduino.esp8266.com/stable/package_esp8266com_index.json ein.
- Gehe ins Menü Werkzeuge/Board/Boardverwalter. Suche dort den ESP8266 Core und installiere ihn. Dort kannst du auch auf andere Versionen wechseln.

Falls deine Programme instabil laufen, versuche die Arduino IDE in Version 1.8.7 mit dem ESP8266 Core in Version 2.3.0.

Downloads: [Linux](#) [Windows](#)

Anwendung: