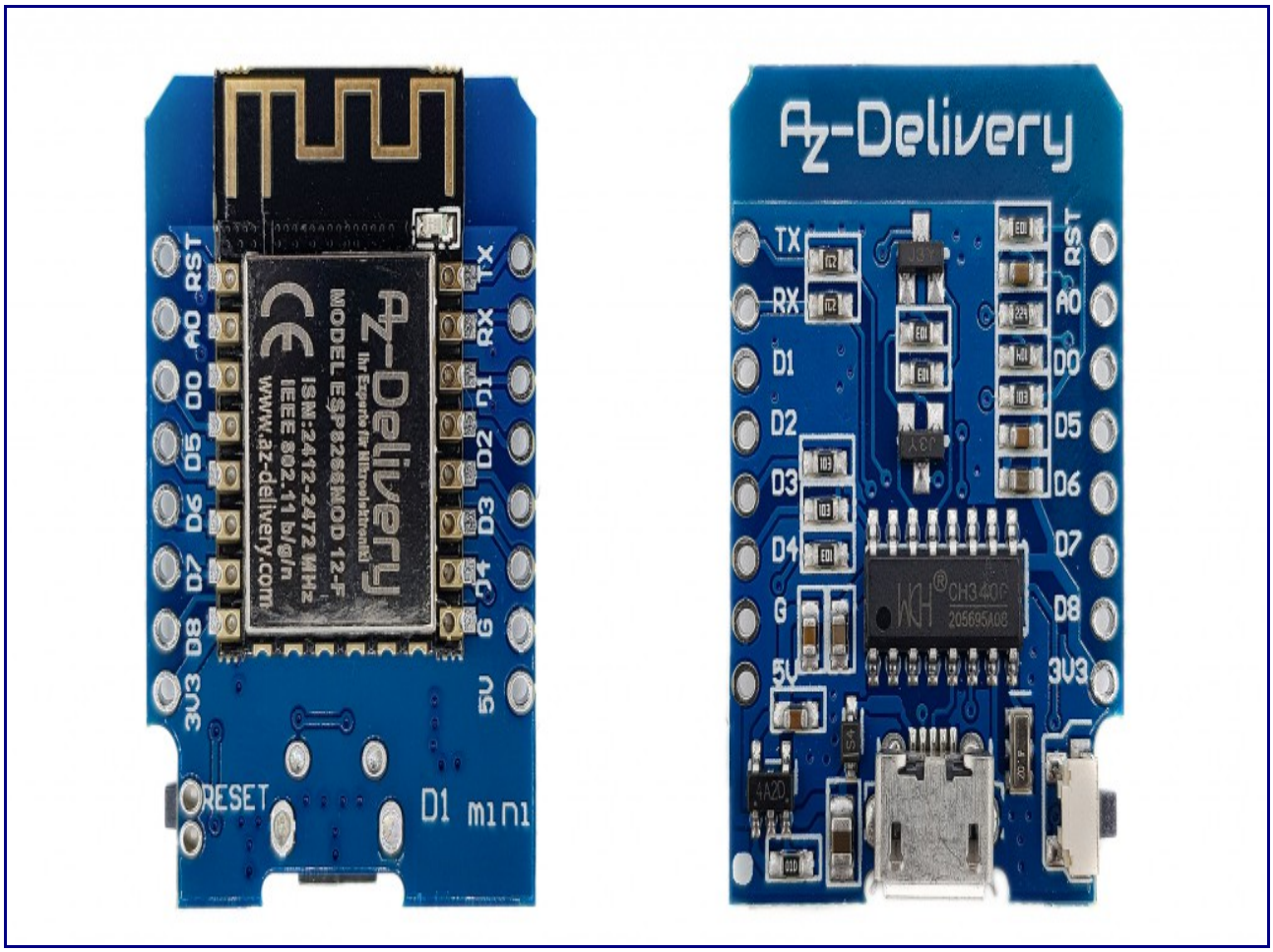
Wemos D1 Mini Boards

Über den Beitrag

In diesem Beitrag möchte ich das Wemos D1 Mini Board vorstellen. Genau genommen handelt es sich dabei um eine Board-Familie, deren Vertreter aber gar nicht so unterschiedlich sind. Wenn ich also vom Wemos D1 Mini spreche, meine ich grundsätzlich immer die ganze Familie.

Zunächst einmal gehe ich darauf ein, was das Wemos D1 Mini Board überhaupt ist, welche Eigenschaften es besitzt und wie sich die verschiedenen Varianten unterscheiden. Dann zeige ich, wie ihr das Board in die Arduino IDE integriert und welche Unterschiede es zum Arduino (UNO) Board gibt. Zum Schluss gibt es einen kleinen Geschwindigkeitstest.

Was ist das Wemos D1 Mini Board?



Wemos D1 Mini Board (hier von AZ-Delivery)

Das Herz eines Wemos D1 Mini Boards ist je nach Version ein ESP8266EX, ein ESP8266 12-F oder ein ESP8285 Chip. Dabei handelt es sich um leistungsstarke 32 Bit Microcontroller mit integrierter W-LAN Schnittstelle.

Das, was der ATmega328P für das Arduino UNO Board ist, ist also der ESP8266/ESP8285 für das Wemos Board. Und so wie der ATmega328P „stand-alone“ programmierbar ist (das habe ich hier behandelt), könnt ihr auch den ESP8266/ESP8285 als solches programmieren. Nur braucht ihr zu diesem Zweck einige zusätzliche Komponenten wie z.B. einen USB-zu-Seriell Adapter. Außerdem ist es nicht jedermanns Sache, ein SMD Bauteil wie den ESP8266EX zu verlöten. Da ist es einfacher, ein Entwicklungsboard wie den Wemos D1 Mini zu verwenden.

Namenswirrwarr

Wenn ihr nach dem Begriff „D1 Mini“ sucht, werdet ihr eine ganze Reihe von Boards mit unterschiedlichen Namen finden. Beispielsweise ist das der Lolin D1 Mini, der oben abgebildete AZ-Delivery D1 Mini (NodeMCU), aber natürlich auch der Wemos D1 Mini. Der „Vorname“ ist dabei nur ein Marken- oder Herstellername und spielt nach meiner Erfahrung keine Rolle. Im weiteren Verlauf des Beitrages verwende ich bevorzugt die Bezeichnung Wemos D1 Mini, denn das ist das Original.

Daneben haben diese Boards noch unterschiedliche „Nachnamen“, z. B. D1 Mini *Pro*, D1 Mini *V3* oder D1 Mini *Lite.* Hier gibt es dann tatsächlich auch Unterschiede.

Wesentliche technische Eigenschaften der Wemos D1 Mini Boards

Die folgenden technischen Eigenschaften haben alle Wemos D1 Mini Boards gemein:

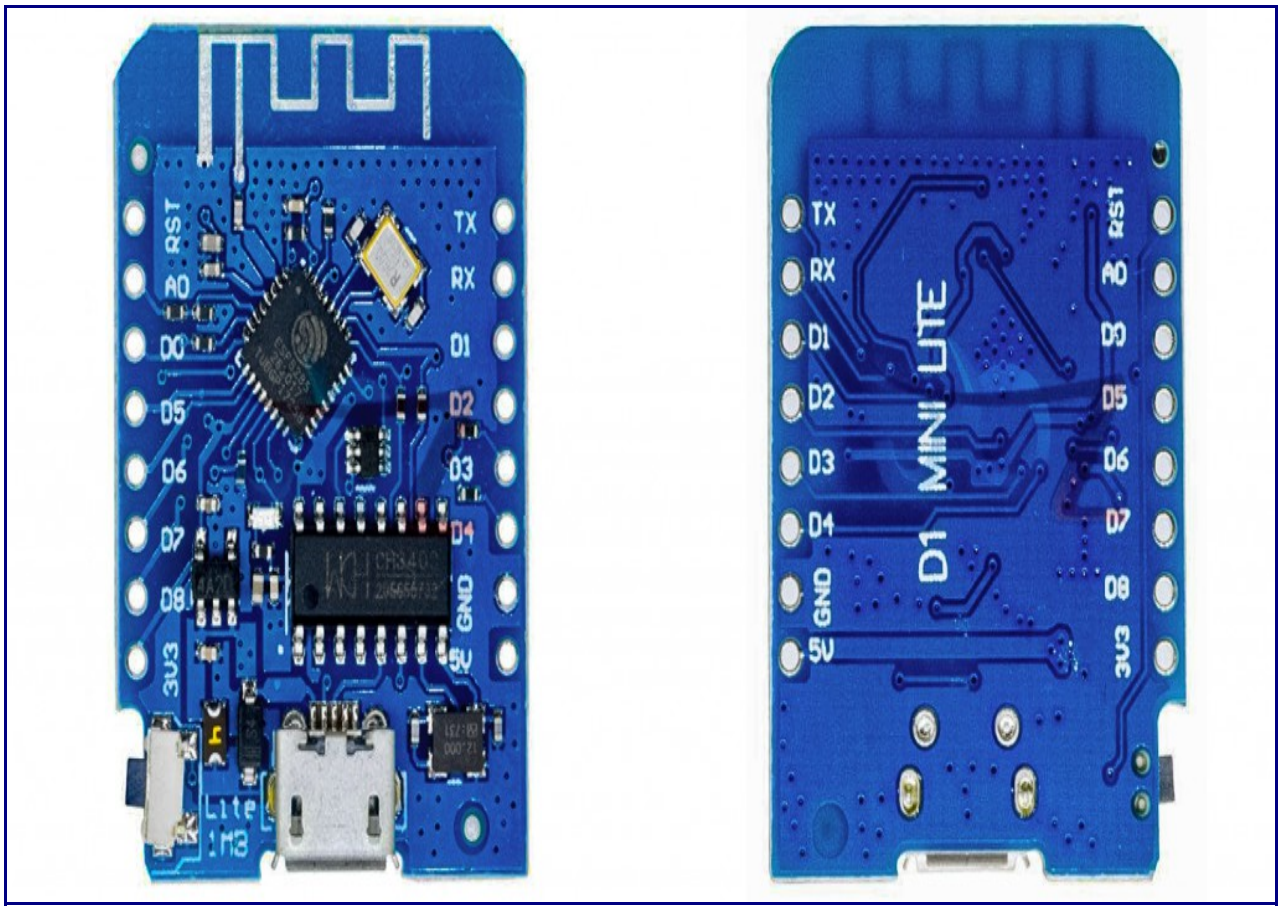
- Bei der **Spannungsversorgung** der Boards habt ihr verschiedene Optionen. Entweder ihr speist sie mit 2,5 bis 3,6 Volt am 3,3 Volt Pin, mit 3 bis 7 Volt am 5 Volt Pin oder über den USB-Mikro Anschluss. Die zugrundeliegenden Microcontroller laufen mit 2,5 – 3,6 Volt.
- Der **Strombedarf** der Wemos D1 Mini Boards ist mit ca. 70 Milliampere vergleichsweise hoch. Beim Senden steigt der Bedarf sogar auf mehrere hundert Milliampere.
- Die Wemos D1 Mini Boards haben die **Schnittstellen** I2C, SPI und UART.
- Das **WIFI** erfüllt die Spezifikationen 802.11 b/g/n (2,4 GHz) mit WPA/WPA2 PSK.
- Die **Taktrate** beträgt 80 oder 160 MHz.
- Es gibt 11 **I/O Pins,** alle bis auf D0 sind **PWM-, I2C- und One-Wire** fähig, I_{max} ist 12 mA.
- Die **Ausgangsspannung** an den I/O Pins beträgt maximal 3.3 Volt.
- 1 **analoger Eingang** mit 10 Bit Auflösung, **max. 3.2 Volt** (!)
- **Flashspeicher**: 1 – 16 MB.

Die verschiedenen Wemos D1 Mini Boards

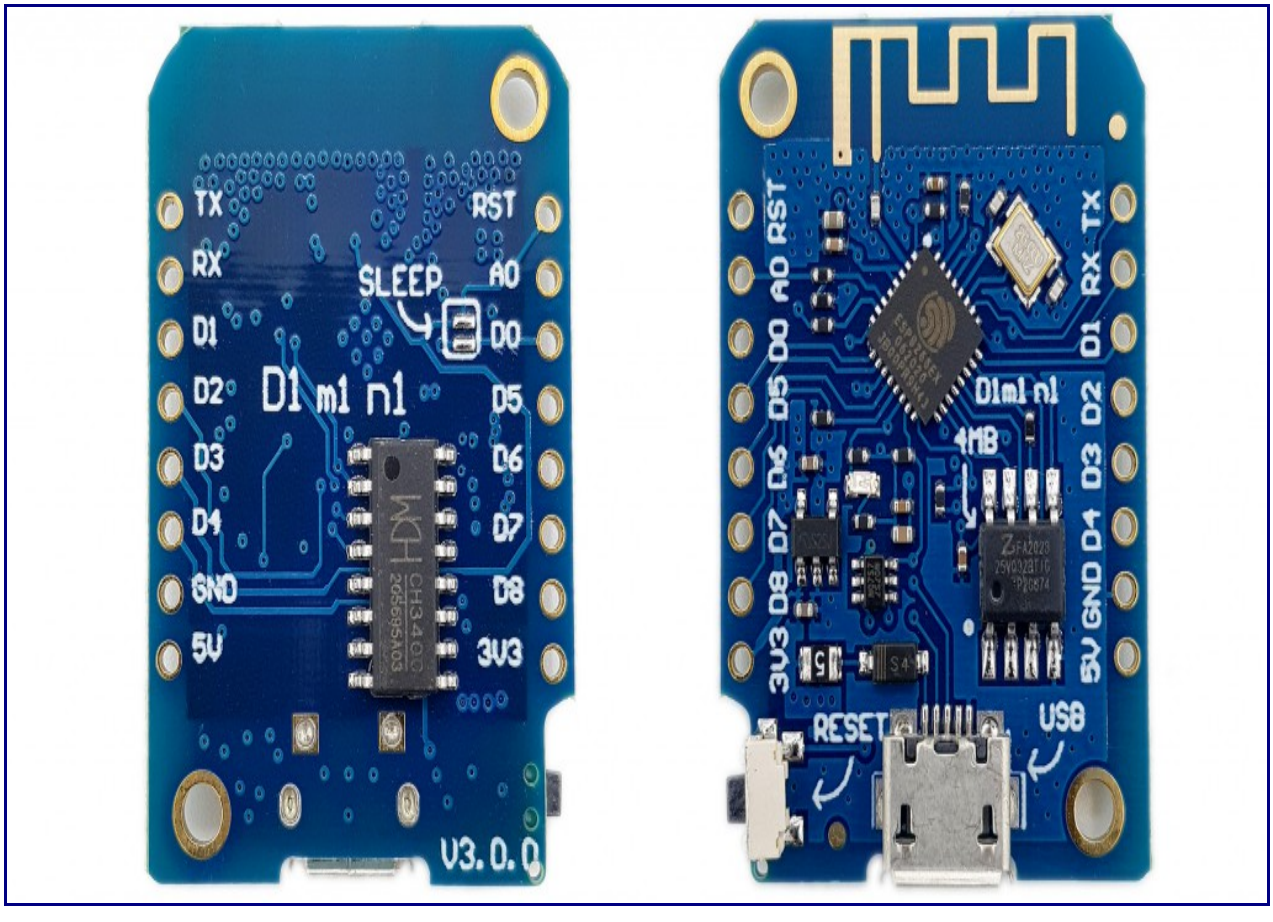
Board	Microcontroller	Flash Speicher	Antenne
D1 Mini Lite (V1)	ESP8285	1 MB	Leiterschleife
D1 Mini (V2)	ESP8266 12-F	4 MB	Leiterschleife
D1 Mini V3	ESP8266	4 MB	Leiterschleife
D1 Mini Pro	ESP8266	16 MB	Keramikantenne oder extern

Tabelle 1: Vertreter der Wemos D1 Mini Board Familie

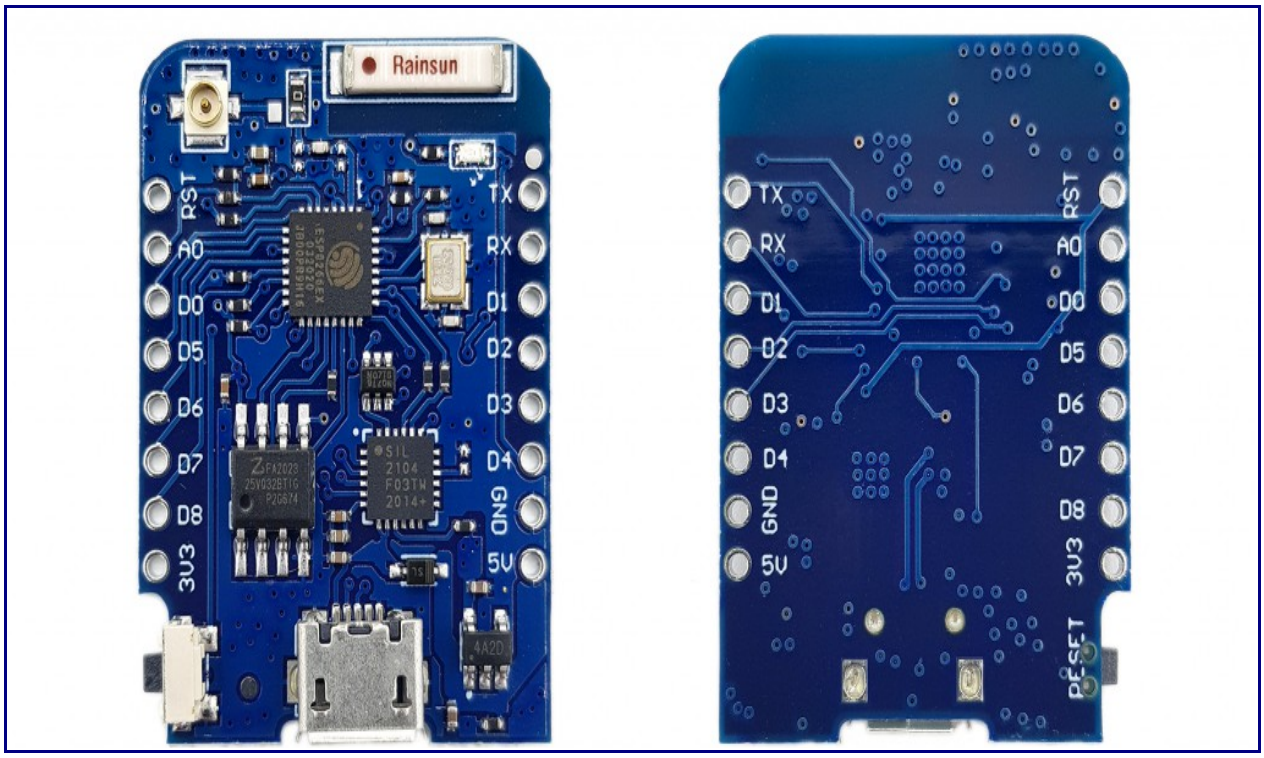
Die Boards unterscheiden sich im Wesentlichen durch den verwendeten Microcontroller, den Flash-Speicher und die Antenne. Daneben gibt es noch Unterschiede beim USB-zu-Seriell Adapter. Einige Boards verwenden den CH340G, andere den CP2104. Dieser Unterschied ist insofern relevant, als unterschiedliche Treiber benötigt werden. Aus der Bezeichnung des Boards lässt sich nicht sicher schließen, welcher USB-zu-Seriell Adapter verwendet wird.



Der kleinste Vertreter (hinsichtlich Flash-Speicher): das Wemos D1 Mini Lite Board



Wemos Mini V3 Board

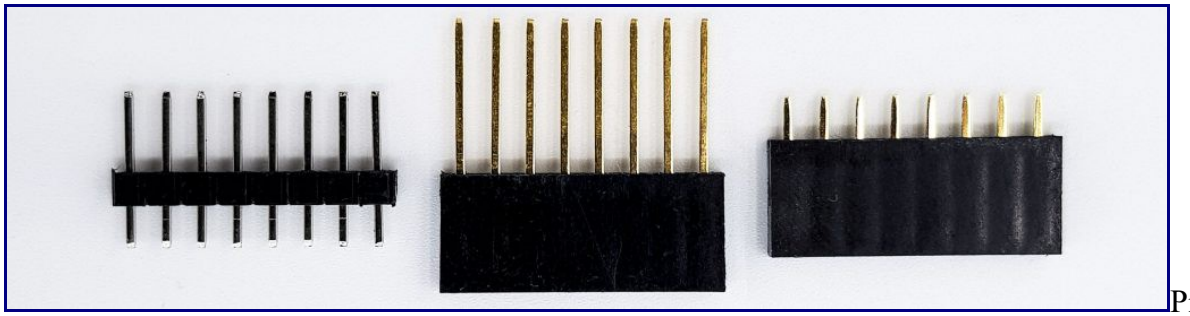


Mit externem Antennenanschluss: Wemos D1 Mini Pro Board

Vorbereitungen

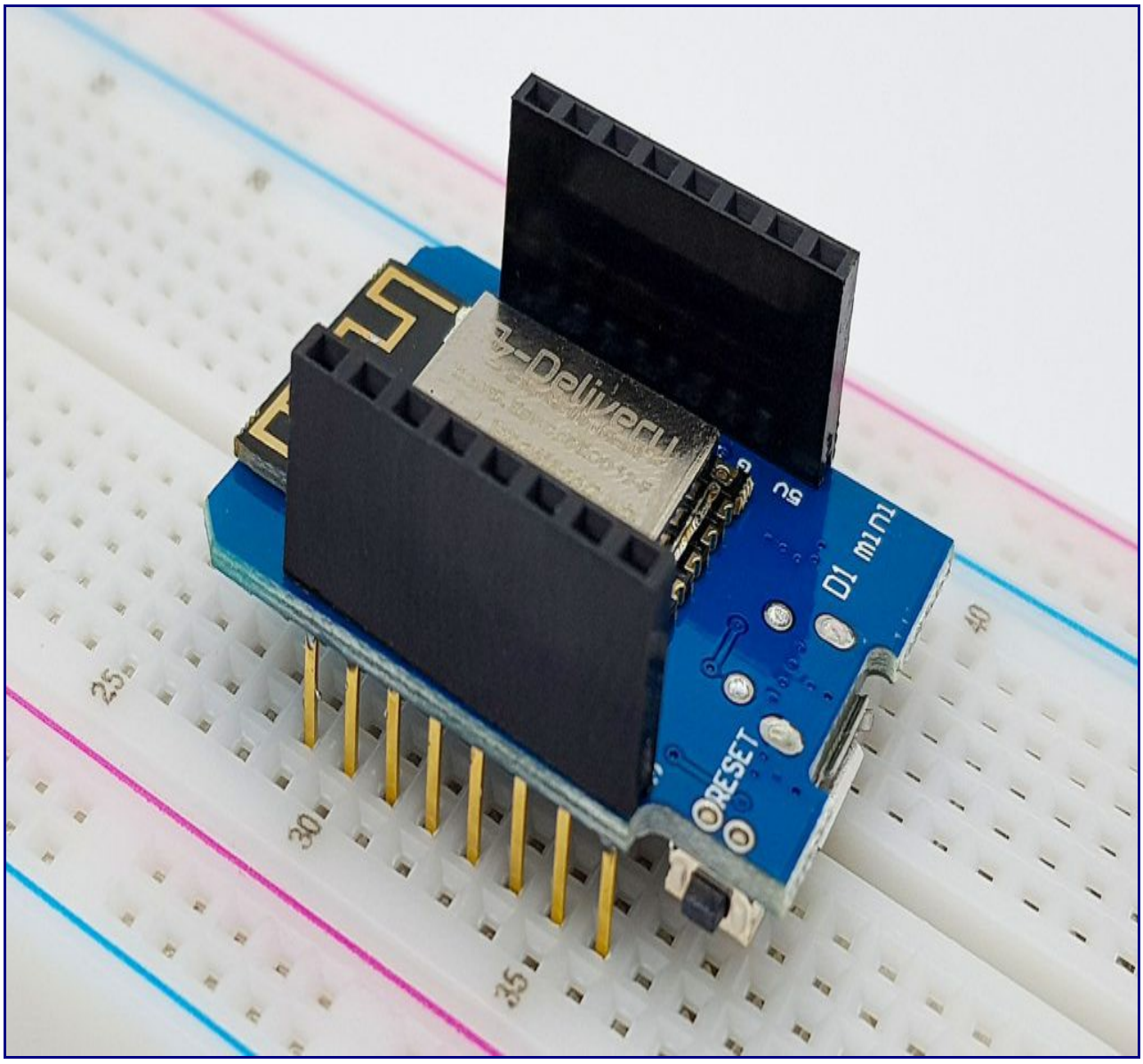
Pinleiste

Üblicherweise bekommt ihr mit eurem Board Pinleisten geliefert. Bei AZ-Delivery erhaltet ihr löblicherweise verschiedene Leisten mitgeliefert (nein, ich werde nicht von denen bezahlt!):



Pinleisten für das Wemos D1 Mini Board

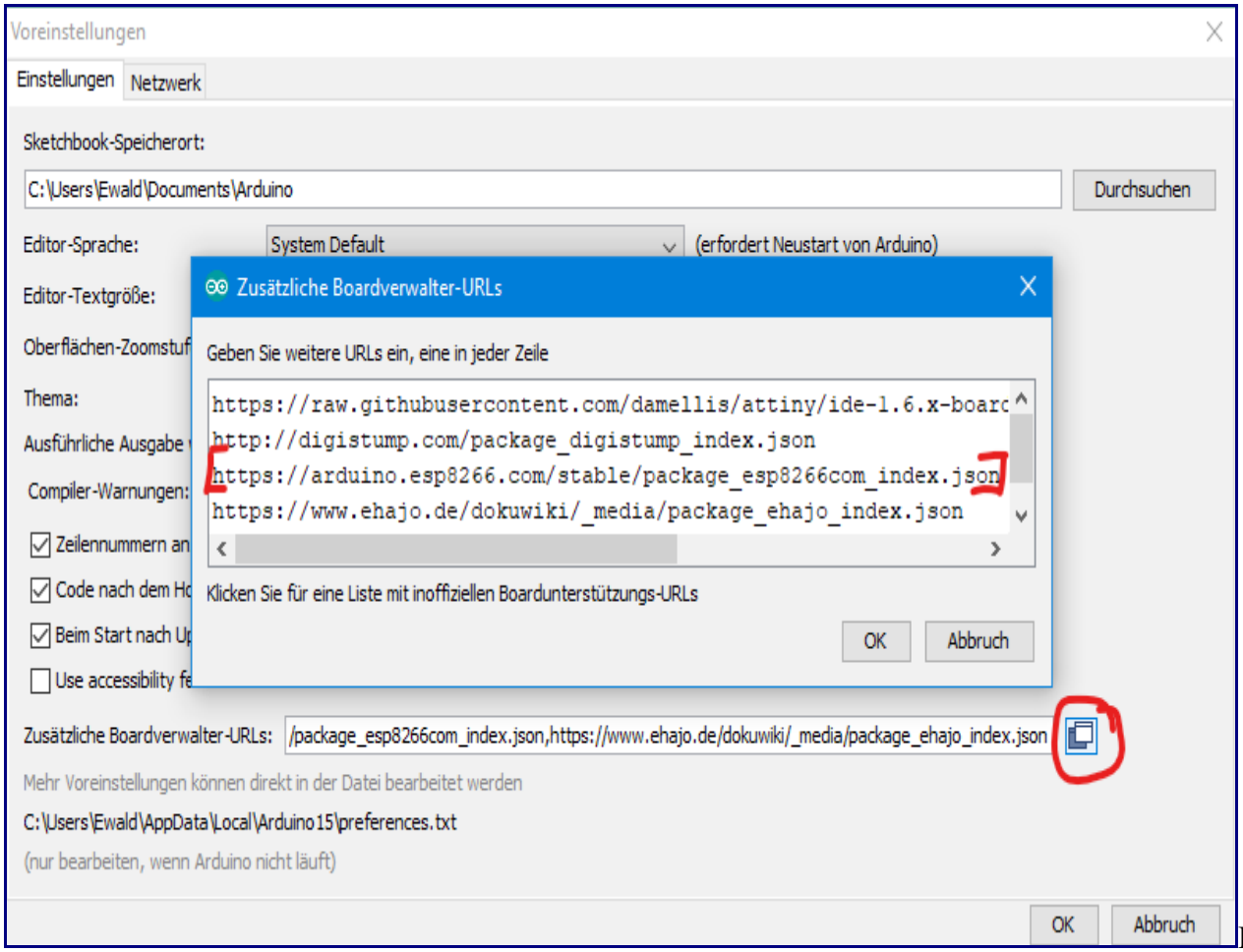
Ich persönlich finde die Buchsenleisten mit den langen Pins für das Experimentieren praktisch, weil sich das Board damit gut auf dem Breadboard fixieren lässt und man zusätzlich die Buchsen nutzen kann:



Das Wemos D1 Mini Board in die Arduino IDE einbinden

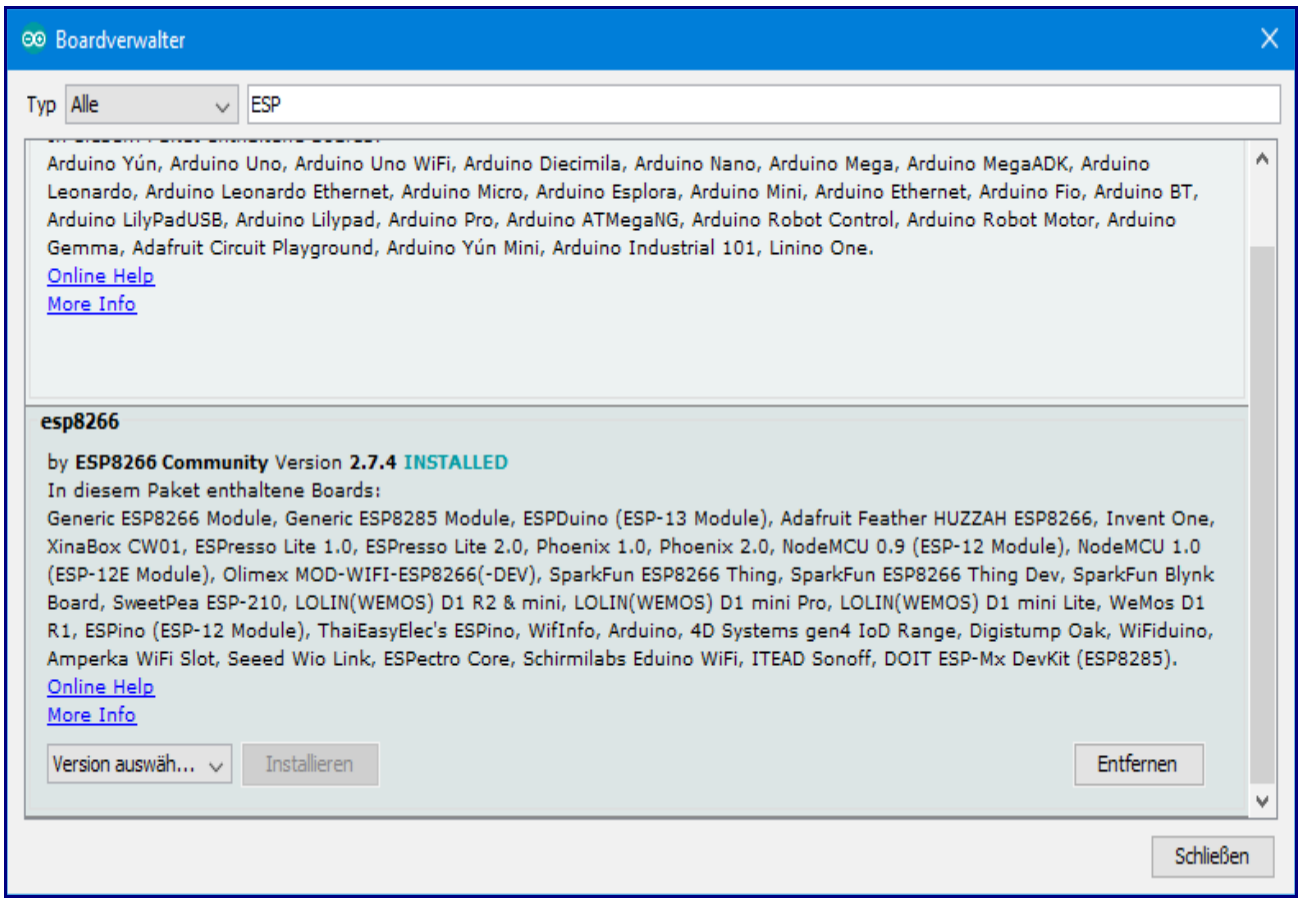
Der Arduino IDE müsst ihr zunächst noch beibringen, mit dem Wemos D1 Mini Board umzugehen. Als Erstes geht ihr dazu auf Datei -> Voreinstellungen und klickt auf das Symbol neben „Zusätzliche Boardverwalter-URLs“. In dem aufgehenden Fenster tragt ihr Folgendes in einer separaten Zeile ein:

https://arduino.esp8266.com/stable/package_esp8266com_index.json



Konfigurierung der Voreinstellungen

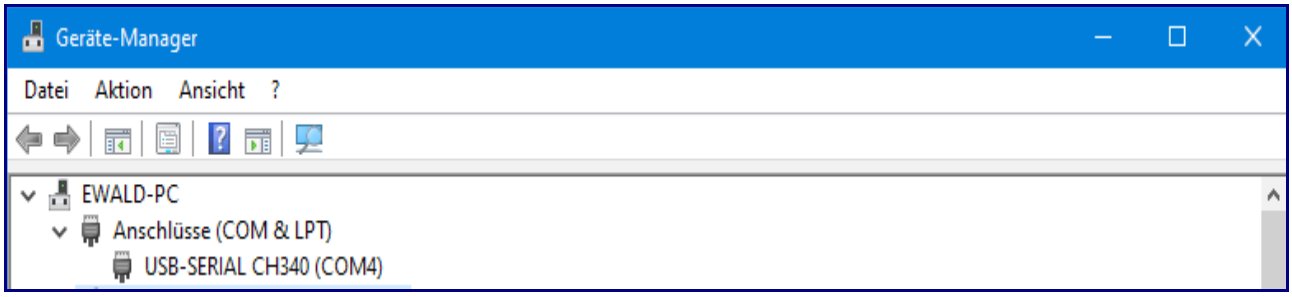
Dann geht ihr zu Werkzeuge -> Board -> Bordverwalter, sucht nach „esp8266“ und installiert das Paket:



Einrichtung im Boardverwalter

Dann startet ihr die Arduino IDE neu.

Wenn euer Board im Gerätemanager erkannt wird, sollte das so oder ähnlich aussehen:

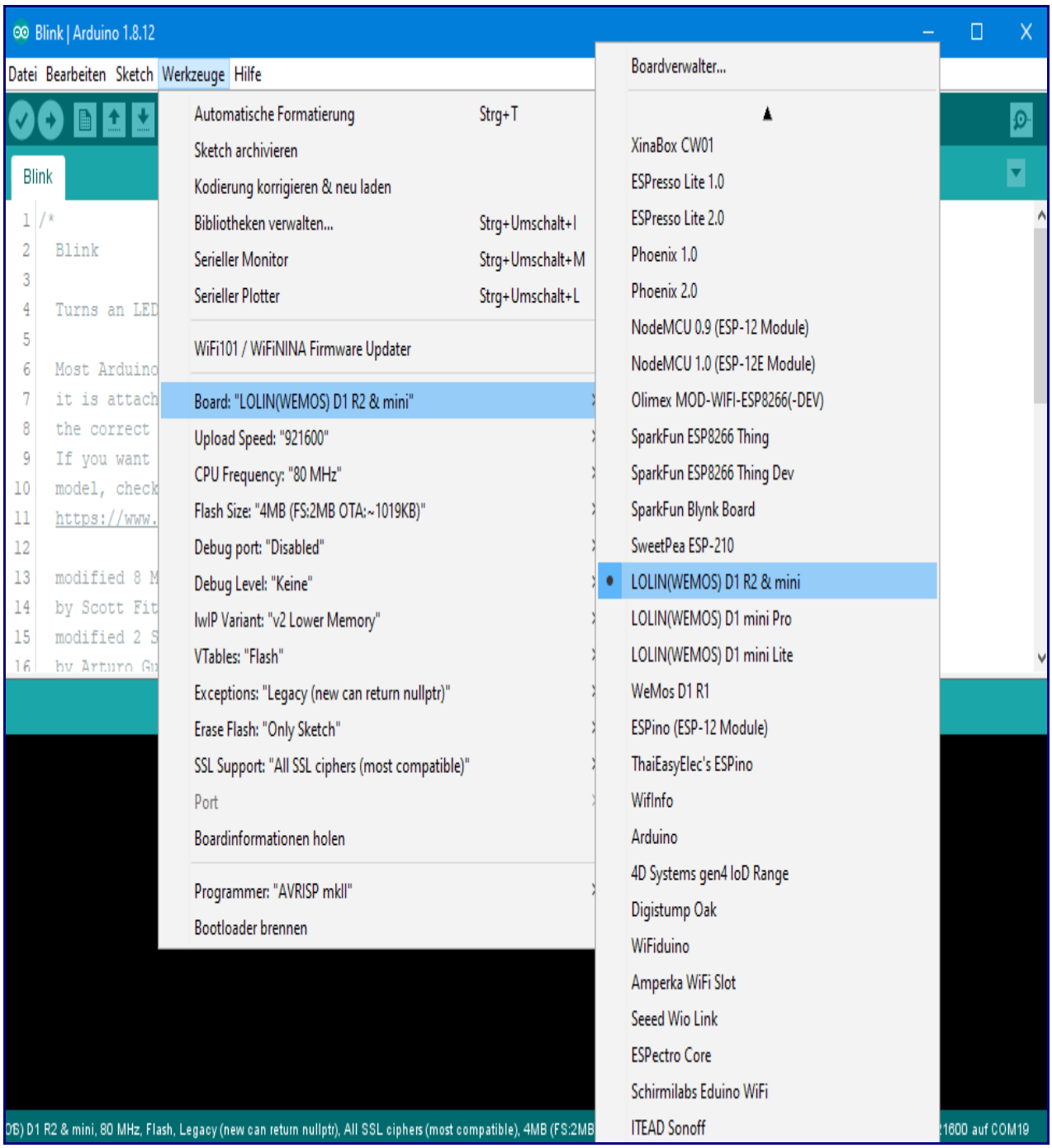


Wemos D1 Miniboard im Gerätemanager

Wenn euer Board nicht erkannt wird, dann fehlt euch wahrscheinlich der Treiber für den USB-zu-Seriell Adapter. Den Treiber für den CH340G erhaltet ihr z.B. hier oder hier (auf das Downloadsymbol klicken). Stört euch nicht an der Bezeichnung „CH341SER“. Den Treiber für den CP2104 bekommt ihr z.B. hier.

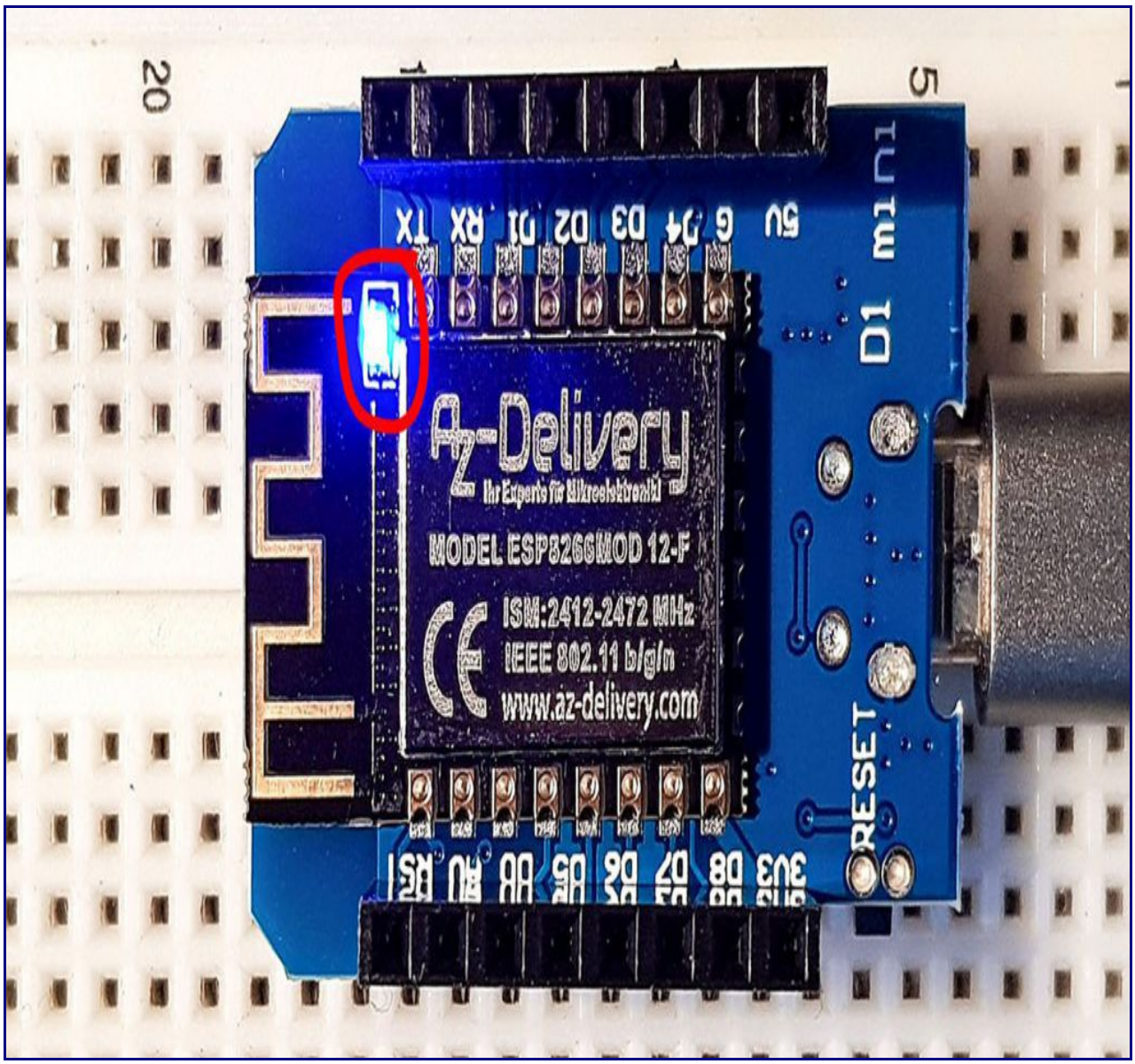
Ein kleiner Test

Öffnet den Blink Beispielsketch: Datei -> Beispiele -> Basics -> Blink. Dann wählt unter Werkzeuge -> Boards das richtige Board aus:



Boardauswahl in der Arduino IDE

Schließlich müsst ihr noch den richtigen Port auswählen und könnt den Sketch dann hochladen. Wenn alles klappt, sollte die Board LED jetzt blinken:



LED des Wemos D1 Mini Boards

Das Wemos D1 Board nutzen

Ich werde nun auf die wichtigsten Basisfunktionen eingehen und dabei besonderes Augenmerk auf die Unterschiede zum Arduino (Uno) legen.

Auch beim Wemos D1 Mini Board haben die Pins zum Teil mehrere Funktionen:

Bezeichnung	GPIO	PWM	Hauptfunktion	Serial	SPI	I2C	Sonstiges
D0	16	-	IO				Wake
D1	5	+	IO			SCL	
D2	4	+	IO			SDA	
D3	0	+	IO				Flash
D4	2	+	IO	TXD1			Board LED
D5	14	+	IO		CLK		
D6	12	+	IO		MISO		
D7	13	+	IO	RXD2	MOSI		
D8	15	+	IO	TXD2	CS		
TX	1	+	IO	TXD0			
RX	3	+	IO	RXD0			
A0	0	-	ADC				

Tabelle 2: Pinbezeichnungen und -funktionen der Wemos D1 Mini Boards

digitalWrite() / analogWrite()

Bei der `digitalWrite()` Funktion ist zunächst zu beachten, dass das HIGH Signal am entsprechenden Ausgang bei 3,3 Volt liegt. Der Strom darf 12 mA nicht überschreiten.

Ihr könnt die Pins D0 – D8 und RX / TX als Ein- bzw. Ausgangspins nutzen. Damit stehen euch also insgesamt 11 Pins zur Verfügung. Um sie anzusprechen, könnt ihr ihre Bezeichnung oder die GPIO Nummer verwenden. Die folgenden Anweisungen haben denselben Effekt:

`digitalWrite(D2, HIGH);` / `digitalWrite(4, HIGH);`

Bei Pin D4 ist zu beachten, dass er mit der Board LED verbunden ist. Allerdings in der Art, dass die Board LED leuchtet, wenn D4 LOW ist.

Der `pinMode` wird wie gewohnt eingestellt. Eine Besonderheit gibt es aber. Wenn ihr den Pull-Down Widerstand an D0 (GPIO 16) aktivieren wollt, dann müsst ihr das mit `pinMode(D0, INPUT_PULLDOWN_16)` tun.

`analogWrite()` steht für alle I/O Pins zur Verfügung. Das gilt auch für den Pin D0, der ansonsten eingeschränkte PWM-Fähigkeiten hat. Im Gegensatz zum Arduino UNO beträgt die Auflösung 10 Bit. Das heißt, dass `analogWrite(pin, 1023)` die volle Spannung liefert.

digitalRead() / analogRead()

`digitalRead()` funktioniert wie beim Arduino mit allen Pins D0 – D8. Die Pins RX und TX haben ihre Eigenheiten. Deshalb würde ich sie für diesen Zweck nicht verwenden.

Der Pin A0 ist der einzige analoge Eingang. Eigentlich verträgt der analoge Eingang des ESP8266 nur maximal 1 Volt. Allerdings ist auf den Wemos D1 Mini Boards ein Spannungsteiler (100 kOhm / 220 kOhm) davor gesetzt, sodass die maximale Spannung 3,2 Volt beträgt. Die Auflösung beträgt 10 Bit. Damit ergibt sich für die Spannung U:

U [V]=3.2·analogRead(A0)1024U\;[\text{V}]=3.2 \cdot \frac{analogRead(\text{A0})}{1024}U[V]=3.2·1024analogRead(A0)

Interessant ist, dass man bei „Vollausschlag“ tatsächlich 1024 erhält und nicht 1023 wie beim Arduino UNO. Bei diesem reite ich immer gerne darauf herum, dass man bei der Spannungsberechnung durch 1023 teilen muss. Das ist hier also anders.

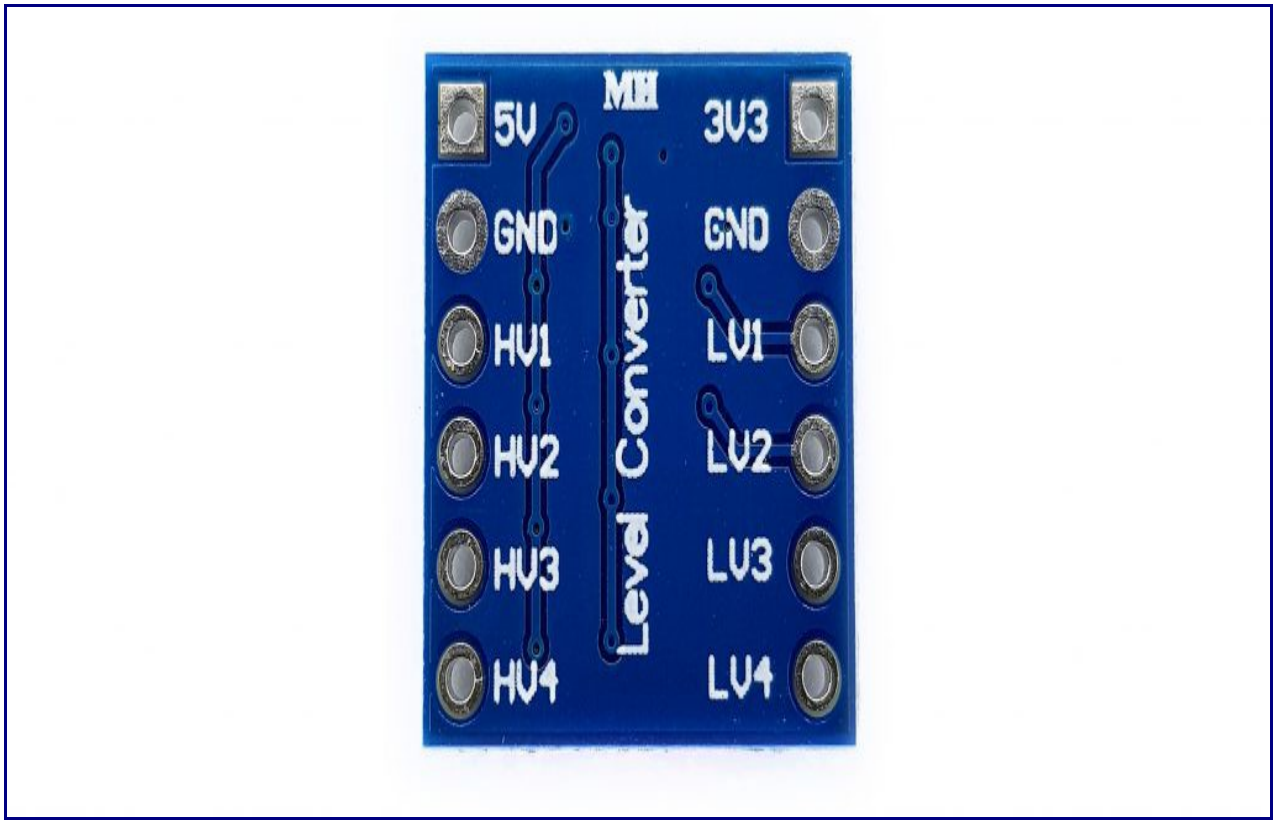
Die `analogRead()` Funktion der Wemos D1 Mini Boards gefällt mir erheblich besser als die des Arduino UNO. Mit dem Arduino UNO schwanken die analogRead Werte um +/- zwei Einheiten, selbst bei Anschluss einer stabilen Spannungsquelle. Die mit dem Wemos D1 Mini ermittelten Werten hingegen schwankten bei mir gar nicht.

I2C mit dem Wemos D1 Mini Board

Wenn ihr I2C mit `Wire.begin()` aktiviert, dann geht das Wemos D1 Mini Board davon aus, dass D1 als SCL und D2 als SDA dient. Es ist aber kein Problem, auch andere Pins nach dem Schema `Wire.begin(SDA-PIN,SCL-PIN)` auszuwählen.

Die Wemos D1 Mini Boards beherrschen den Fast Mode (400 kHz), darüber wird es eng.

Aufpassen müsst ihr bei den Spannungsleveln der I2C Verbindung. Verwendet ihr ein I2C Bauteil das auf 5 Volt läuft, müsst ihr einen Spannungsteiler in die Leitung setzen oder ihr nehmt einen Logik-Level Konverter wie diesen:



Logik-Level Konverter

SPI mit dem Wemos D1 Mini Board

Zum Thema SPI gibt es nicht so viel zu sagen. Die Anschlüsse findet ihr in Tabelle 2.

Serielle Schnittstelle

Die Pins TX und RX für die serielle Kommunikation benutzt ihr wie beim Arduino UNO. Ihr könnt

aber auch stattdessen die Pins D7 und D8 (= GPIO 13 / RXD2 bzw. GPIO 15 / TXD2) benutzen. Dazu müsst ihr lediglich im Setup die Anweisung `Serial.swap()` nach `Serial.begin()` einfügen.

Achtung – bissiger Wachhund

Nehmt mal den folgenden, an sich ziemlich sinnlosen Sketch und ladet ihn unverändert auf euer Wemos D1 Mini Board.

Im Setup blinkt die Board LED zehnmal in schneller Frequenz. Das soll einfach nur den Startpunkt des Sketches visualisieren. In der Hauptschleife wartet euer Wemos D1 Mini Board auf ein HIGH Signal an D1, z.B. durch einen Taster oder Sensor. Und falls keines kommt, sollte euer Board die `while()` Schleife ewig ausführen.

ESP8266_Watchdog_Demo.ino
void setup() {
pinMode(D1, INPUT); // Eigentlich redundant, da das der Standardzustand ist
pinMode(LED_BUILTIN, OUTPUT);
for(int i=0; i<10; i++){
digitalWrite(LED_BUILTIN, LOW);
delay(50);
digitalWrite(LED_BUILTIN, HIGH);
delay(50);
}
//ESP.wdtDisable();
}
void loop() {
while(!digitalRead(D1)){
//delay(0);
}
}
void setup() { pinMode(D1, INPUT); // Eigentlich redundant, da das der Standardzustand ist pinMode(LED_BUILTIN, OUTPUT); for(int i=0; i<10; i++){ digitalWrite(LED_BUILTIN, LOW); delay(50); digitalWrite(LED_BUILTIN, HIGH); delay(50); } //ESP.wdtDisable(); } void loop() { while(!digitalRead(D1)){ //delay(0); } }

```
void setup() {
  pinMode(D1, INPUT); // Eigentlich redundant, da das der Standardzustand ist
  pinMode(LED_BUILTIN, OUTPUT);
  for(int i=0; i<10; i++){
    digitalWrite(LED_BUILTIN, LOW);
    delay(50);
    digitalWrite(LED_BUILTIN, HIGH);
    delay(50);
  }
  //ESP.wdtDisable();
}

void loop() {
  while(!digitalRead(D1)){
    //delay(0);
  }
}
```

Wenn ihr den Sketch laufen lasst, dann werdet ihr aber an der blinkenden LED sehen, dass er ca.

alle drei Sekunden neu startet. Hier schlägt ein Sicherheitsmechanismus zu und der heißt Watchdog Timer. Über Watchdog Timer hatte ich schon mal einen ganzen Beitrag geschrieben (hier). An dieser Stelle nur so viel dazu: ein Watchdog Timer ist ein Zähler, der, wenn er nicht zurückgesetzt wird, überläuft und dann einen Reset auslöst. Er wacht also über den Microcontroller und sorgt so dafür, dass z.B. ein hängengebliebenes Programm neu startet.

Das Wemos D1 Mini Board, oder genauer gesagt der ESP8266, besitzt einen Software Watchdog und einen Hardware Watchdog Timer. Der Software Watchdog ist standardmäßig aktiviert und läuft ca. alle 3 Sekunden über. Bestimmte Funktionen setzen ihn zurück, z.B. wenn ihr die Hauptschleife (loop) neu startet oder ein `delay()` oder – ganz explizit – ein `ESP.wdtFeed()` ausführt (der Watchdog wird gefüttert). Entkommentiert mal die Zeile 15 und ihr werdet sehen, dass euer Board nicht mehr ständig neu startet.

Den Watchdog hinhalten

Ihr könnt die Zeit bis zum Reset verlängern, indem ihr den Software Watchdog abschaltet. Dazu entkommentiert die Zeile 10. Allerdings schlägt jetzt der Hardware Watchdog ca. alle 8 Sekunden zu. Der Hardware Watchdog lässt sich nicht abschalten und die Watchdog Zeiten lassen sich auch nicht variieren.

Was soll das?

Der ESP8266 führt regelmäßig Routinen aus, die sich um die Funktion des W-LANs kümmern. Das macht er beispielsweise, wenn die Hauptschleife neu startet oder bei einem `delay()`. Gibt man ihm dazu keine Gelegenheit, dann startet er lieber neu, um die korrekte W-LAN Funktion zu gewährleisten. Das wiederum kann aber zu Problemen führen wie bei dem Sketch oben.

W-LAN – Funktionen

Einer der Hauptgründe für die Wahl eines Wemos D1 Mini Boards ist natürlich seine W-LAN-Funktionalität. Allein darüber ließe sich ein eigener Beitrag schreiben. Als „Appetizer“ möchte ich hier ein einfaches Beispiel zeigen. Mit dem folgenden Sketch könnt ihr eine LED an D1 per Browser an- und ausschalten.

LED_per_WLAN_schalten.ino

```
#include "ESP8266WebServer.h"
#define LEDPIN D1
const char* ssid = "Deine SSID";
const char* pass = "Dein Passwort";
IPAddress ip(192,168,178,xxx); // xxx = wählt eine frei IP in eurem Heimnetz
IPAddress gateway(192,168,178,1);
IPAddress subnet(255,255,255,0);
ESP8266WebServer server(80);
String led1= "<a href=\"/led_an\">LED An</a>";
String led0= "<a href=\"/led_aus\">LED Aus</a>";
void handleRoot() {
String message="<h1>Testprogramm - Minimalprogramm ESP8266</h1>";
message += "Hallo ....., das ist ein Gru&szlig vom ESP8266 Server</BR></BR>";
message += led1;
server.send(200, "text/html", message);
}
void ledan(){
digitalWrite(LEDPIN, HIGH);
server.send(200, "text/html", led0);
```

```
}
void ledaus(){
digitalWrite(LEDPIN, LOW);
server.send(200, "text/html", led1);
}
void setup(){
pinMode(LEDPIN, OUTPUT);
digitalWrite(LEDPIN, LOW);
// Serial.begin(9600);
// Serial.println("Testprogramm - Minimalprogramm ESP8266");
// Serial.print("Verbinde mich mit Netz: ");
// Serial.println(ssid);
WiFi.begin(ssid, pass);
WiFi.config(ip, gateway, subnet);
// while(WiFi.status() != WL_CONNECTED){
// delay(500); Serial.print(".");
// }
// Serial.println("");
// Serial.println("WiFi Verbindung aufgebaut");
// Serial.print("Eigene IP des ESP-Modul: ");
// Serial.println(WiFi.localIP());
server.on("/",handleRoot);
server.on("/led_an", ledan);
server.on("/led_aus", ledaus);
server.begin();
// Serial.println("HTTP Server wurde gestartet!");
}
void loop(){
server.handleClient();
}
```

#include "ESP8266WebServer.h" #define LEDPIN D1 const char* ssid = "Deine SSID"; const char* pass = "Dein Passwort"; IPAddress ip(192,168,178,xxx); // xxx = wählt eine frei IP in eurem Heimnetz IPAddress gateway(192,168,178,1); IPAddress subnet(255,255,255,0); ESP8266WebServer server(80); String led1= "LED An"; String led0= "LED Aus"; void handleRoot() { String message="<h1>Testprogramm - Minimalprogramm ESP8266</h1>"; message += "Hallo ....., das ist ein Gru&szlig vom ESP8266 Server</BR></BR>"; message += led1; server.send(200, "text/html", message); } void ledan() { digitalWrite(LEDPIN, HIGH); server.send(200, "text/html", led0); } void ledaus() { digitalWrite(LEDPIN, LOW); server.send(200, "text/html", led1); } void setup() { pinMode(LEDPIN, OUTPUT); digitalWrite(LEDPIN, LOW); // Serial.begin(9600); // Serial.println("Testprogramm - Minimalprogramm ESP8266"); // Serial.print("Verbinde mich mit Netz: "); // Serial.println(ssid); WiFi.begin(ssid, pass); WiFi.config(ip, gateway, subnet); // while(WiFi.status() != WL_CONNECTED){ // delay(500); Serial.print("."); // } // Serial.println(""); // Serial.println("WiFi Verbindung aufgebaut"); // Serial.print("Eigene IP des ESP-Modul: "); // Serial.println(WiFi.localIP()); server.on("/",handleRoot); server.on("/led_an", ledan); server.on("/led_aus", ledaus); server.begin(); // Serial.println("HTTP Server wurde gestartet!"); } void loop(){ server.handleClient(); }

```
#include "ESP8266WebServer.h"
#define LEDPIN D1

const char* ssid = "Deine SSID";
const char* pass = "Dein Passwort";
IPAddress ip(192,168,178,xxx); // xxx = wählt eine frei IP in eurem Heimnetz
```

```
IPAddress gateway(192,168,178,1);
IPAddress subnet(255,255,255,0);

ESP8266WebServer server(80);

String led1= "<a href=\"/led_an\">LED An</a>";
String led0= "<a href=\"/led_aus\">LED Aus</a>";

void handleRoot() {
  String message="<h1>Testprogramm - Minimalprogramm ESP8266</h1>";
  message += "Hallo ......, das ist ein Gru&szlig vom ESP8266 Server</BR></BR>";
  message += led1;
  server.send(200, "text/html", message);
}

void ledan(){
  digitalWrite(LEDPIN, HIGH);
  server.send(200, "text/html", led0);
}

void ledaus(){
  digitalWrite(LEDPIN, LOW);
  server.send(200, "text/html", led1);
}

void setup(){
  pinMode(LEDPIN, OUTPUT);
  digitalWrite(LEDPIN, LOW);
//  Serial.begin(9600);
//  Serial.println("Testprogramm - Minimalprogramm ESP8266");
//  Serial.print("Verbinde mich mit Netz: ");
//  Serial.println(ssid);
  WiFi.begin(ssid, pass);
  WiFi.config(ip, gateway, subnet);

//  while(WiFi.status() != WL_CONNECTED){
//  delay(500); Serial.print(".");
//  }
//  Serial.println("");
//  Serial.println("WiFi Verbindung aufgebaut");
//  Serial.print("Eigene IP des ESP-Modul: ");
//  Serial.println(WiFi.localIP());

  server.on("/",handleRoot);
  server.on("/led_an", ledan);
  server.on("/led_aus", ledaus);
  server.begin();
//  Serial.println("HTTP Server wurde gestartet!");
}

void loop(){
  server.handleClient();
}
```

Erklärungen zu diesem Sketch und weitere Beispiele findet ihr in meinem Beitrag über den ESP8266 ESP-01. Oder ihr wollt euch auf eurem Smartphone informieren lassen, wenn bestimmte Sensordaten ein Limit überschreiten? Kein Problem mit IFTTT (if this then that) – das habe ich hier beschrieben.

Geschwindigkeitstest

Zum Schluss möchte ich mich noch der Frage widmen, wie schnell die Wemos D1 Mini Boards tatsächlich sind. 80 oder 160 MHz klingen ja recht verheißungsvoll gegenüber den 16 MHz des Arduino UNO.

digitalRead();

Als Erstes habe ich die Geschwindigkeit der `digitalRead()` Funktion ermittelt. Dazu habe ich das Board diese Funktion eine Million mal ausführen lassen. Die dafür benötigte Zeit habe ich über die `millis()` Funktion gemessen.

Geschwindigkeitstest.ino
void setup() {
Serial.begin(9600);
}
void loop() {
long startTime = millis();
for(long i=0; i<1000000; i++){
digitalRead(10);
}
long duration = millis() - startTime;
Serial.println(duration);
}
void setup() { Serial.begin(9600); } void loop() { long startTime = millis(); for(long i=0; i<1000000; i++){ digitalRead(10); } long duration = millis() - startTime; Serial.println(duration); }

```
void setup() {
  Serial.begin(9600);
}

void loop() {
  long startTime = millis();
  for(long i=0; i<1000000; i++){
    digitalRead(10);
  }
  long duration = millis() - startTime;
  Serial.println(duration);
}
```

Hier die Ergebnisse für 1 Mio. Ausführungen auf dem Arduino UNO und dem D1 Mini bei 80 bzw. 160 MHz:

- Arduino UNO : 3710 ms
- D1 Mini bei 80 MHz: 588 ms
- D1 Mini bei 160 MHz: 325 ms

In grober erster Näherung entspricht das tatsächlich dem Verhältnis der Taktraten.

Mathematische Operationen

Für die Geschwindigkeitsmessung mathematischer Operationen habe ich mit `pow(i,0.5)` eine Funktion ausgesucht, die relativ rechenintensiv ist:

void setup() {
Serial.begin(9600);

}
void loop() {
long startTime = millis();
float f = 0.0;
for(long i=0; i<50000; i++){
f = pow(i,0.5);
}
long duration = millis() - startTime;
Serial.println(duration);
Serial.println(f);
}
void setup() { Serial.begin(9600); } void loop() { long startTime = millis(); float f = 0.0; for(long i=0; i<50000; i++){ f = pow(i,0.5); } long duration = millis() - startTime; Serial.println(duration); Serial.println(f); }

```
void setup() {
  Serial.begin(9600);
}

void loop() {
  long startTime = millis();
  float f = 0.0;
  for(long i=0; i<50000; i++){
    f = pow(i,0.5);
  }
  long duration = millis() - startTime;
  Serial.println(duration);
  Serial.println(f);
}
```

Die Zeile `Serial.println(f)` ist übrigens wichtig, da der Compiler die Berechnung von f „wegoptimiert“, wenn f nicht verwendet wird. Ich hatte mich sehr gewundert, als ich immer „0“ als Ergebnis für „duration“ erhielt, bis ich darauf kam, dass f verwendet werden muss.

Hier nun die Ergebnisse:

- Arduino UNO : 17119 ms
- D1 Mini bei 80 MHz: 926 ms
- D1 Mini bei 160 MHz: 463 ms

Bei dieser Rechenoperation ist der ESP8266 Chip gegenüber dem ATmega328P noch schneller, als es durch die Taktrate erklärlich ist. Hier kommt zur Taktrate noch ein Architekturvorteil zum Tragen. Auf jeden Fall ist der Unterschied wirklich groß. Ob ihr den allerdings wirklich braucht, steht natürlich auf einem anderen Blatt Papier.