MIKROCONTROLLER · WLAN & BLUETOOTH

ESP32 DevKit

Dual-Core-Board mit WLAN und Bluetooth. Erster Upload in rund 20 Minuten, danach acht Beispiele vom Taster bis zum Webserver.



PROZESSOR 2 × 240 MHz	SPEICHER 4 MB Flash
FUNK WLAN + Bluetooth	USB USB-C · CH340C

01
Einrichten
IDE, Treiber, Board-Paket

02
Hochladen
Erster Sketch

03
Testen
WLAN-Scan im Monitor

04
Erweitern
8 Code-Beispiele

01 Auf einen Blick

Das ESP32 DevKit bringt alles mit, was ein UNO nicht hat: zwei Kerne mit bis zu 240 MHz, WLAN und Bluetooth auf demselben Chip. Programmiert wird es wie ein Arduino – mit der Arduino IDE und einem USB-C-Kabel.

Technische Daten

Modul	**ESP32-WROOM-32 (Chip ESP32-D0WD-V3)**
Prozessor	**Dual-Core Xtensa LX6, bis 240 MHz**
Speicher	**4 MB Flash, 520 kB SRAM**
Funk	**WLAN 2,4 GHz · Bluetooth Classic + BLE**
USB	**USB-C, Wandler CH340C**
Versorgung	**5 V über USB-C oder VIN**
Logikpegel	**3,3 V – Pins nicht 5-V-tolerant**
Anschlüsse	**30 Pins (2 × 15), 2,54-mm-Raster**

Lieferumfang

1 × **ESP32 CH340C Entwicklungsboard (30 Pins, USB-C)**

> ! **3,3 V an den Pins**
> Die GPIOs arbeiten mit 3,3 V und vertragen keine 5 V. 5-V-Signale nur über Spannungsteiler oder Pegelwandler anschließen.

> i **Strom: USB-C oder VIN**
> Versorge das Board über USB-C oder mit 5 V an VIN. Der 3V3-Pin ist ein *Ausgang* des Spannungsreglers – dort nichts einspeisen.

Das ist auf der Platine

Taste **EN**	**Neustart (Reset) – der Sketch beginnt von vorn.**
Taste **BOOT**	**Nur nötig, falls ein Upload hängen bleibt (Seite 5).**
LED „PWR“	**Leuchtet, sobald das Board Strom hat.**
LED „IO2“	**Blaue LED an GPIO2 – im ersten Sketch blinkt sie.**
Chip CH340C	**USB-Seriell-Wandler: macht aus USB-C einen COM-Port.**

Was du zusätzlich brauchst

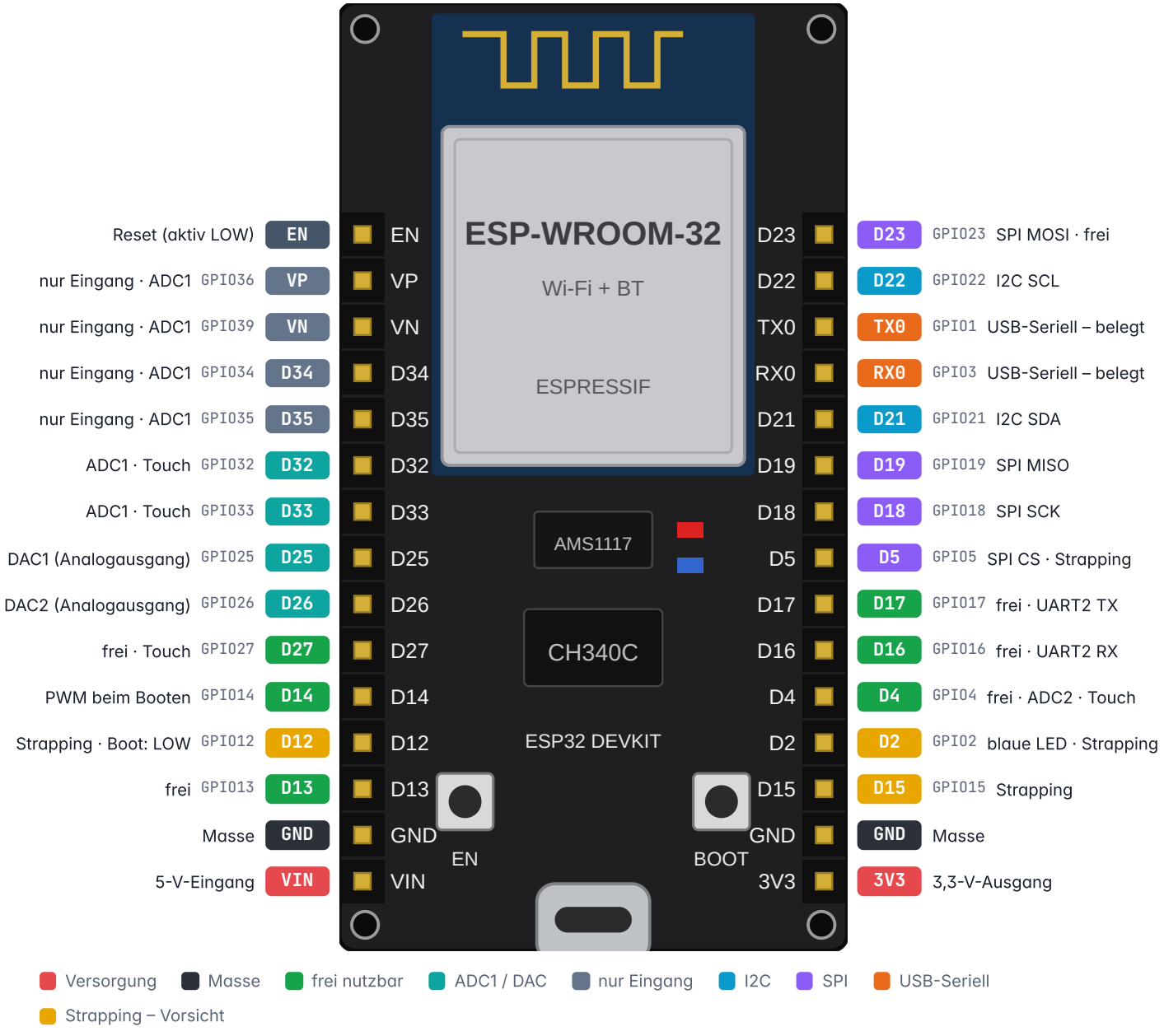
USB-C-Datenkabel
Kein reines Ladekabel – es muss Daten übertragen

PC mit Arduino IDE 2
Windows, macOS oder Linux, kostenlos

Für die Beispiele (optional)
Steckbrett, Jumper, LED + 220 Ω, Poti 10 kΩ, I2C-Sensor
`MKR-SET-BB400-JUMPER`

02 Pinbelegung

So, wie das Board vor dir liegt: Modul oben, USB-C unten. Die Namen entsprechen dem Aufdruck auf der Platine, im Code verwendest du die GPIO-Nummer.



Strapping-Pins: D2, D5, D12, D15

Ihr Pegel entscheidet beim Einschalten, wie der ESP32 startet. Schließe dort nichts an, was die Leitung beim Booten fest auf HIGH oder LOW zieht – sonst schlägt der Upload fehl. Bei aktivem WLAN funktionieren analoge Messungen nur an ADC1 (D32–D39).

03 Einrichten

Diese Schritte machst du nur ein einziges Mal. Die Bilder stammen aus der Arduino IDE 2.3 mit angeschlossenem Board.

1 Arduino IDE installieren

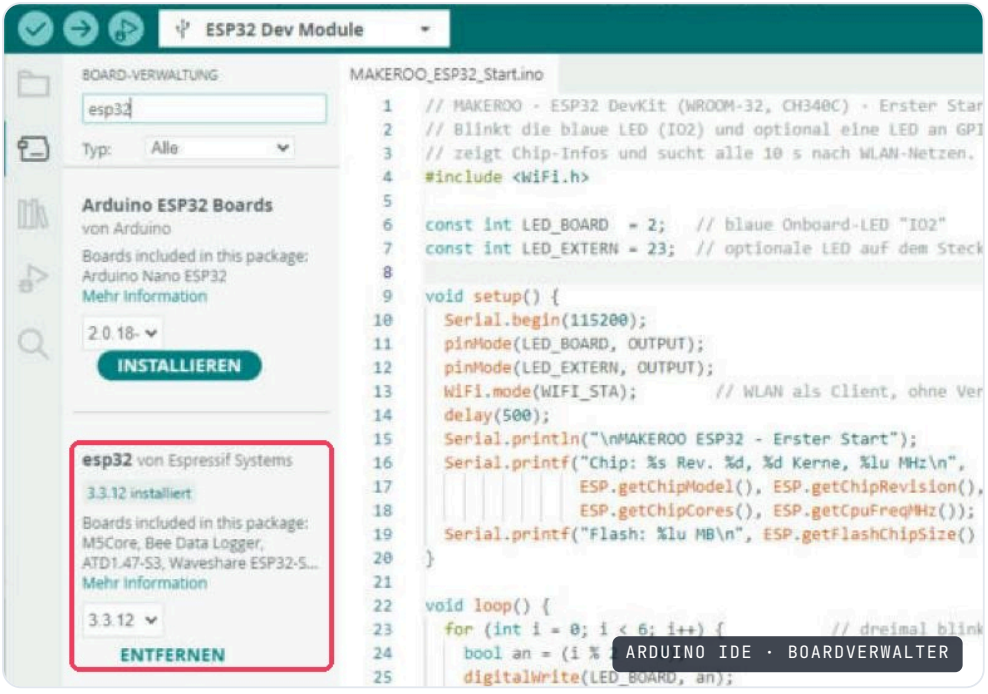
Lade die aktuelle Arduino IDE 2 von `arduino.cc/en/software` herunter und installiere sie.

2 Board anschließen, Treiber prüfen

Board per USB-C anstecken – die rote LED „PWR“ leuchtet. Windows 10/11 und macOS bringen den Treiber für den CH340C meist mit. Taucht später kein Port auf, installiere „CH341SER“ vom Chiphersteller WCH (`wch-ic.com`).

3 ESP32-Board-Paket installieren

Links auf das Board-Symbol klicken (Boardverwalter), **esp32** eingeben und beim Eintrag **esp32 von Espressif Systems** auf **Installieren** klicken. Der Download ist rund 300 MB groß und dauert ein paar Minuten. Nicht verwechseln mit „Arduino ESP32 Boards“ – das ist für den Arduino Nano ESP32.



4 Board und Port wählen

In der Board-Auswahl oben auf **Anderes Board und Port wählen** gehen, **ESP32 Dev Module** suchen und den neuen COM-Port wählen (Windows: `COMx`, macOS: `/dev/cu.wchusbserial…`, Linux: `/dev/ttyUSB0`). Unten rechts steht danach z. B. „ESP32 Dev Module an COM7“.



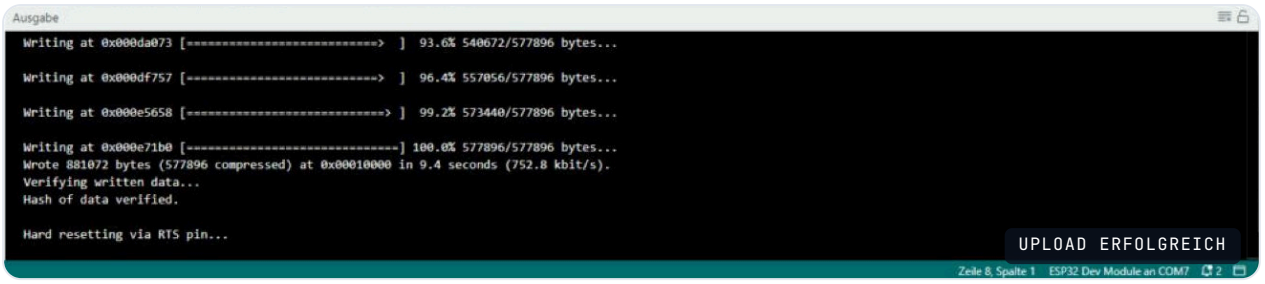


04 Hochladen & Testen

Jetzt kommt dein erster Sketch auf den ESP32: Er lässt die blaue LED blinken und zeigt dir alle WLAN-Netze in der Umgebung.

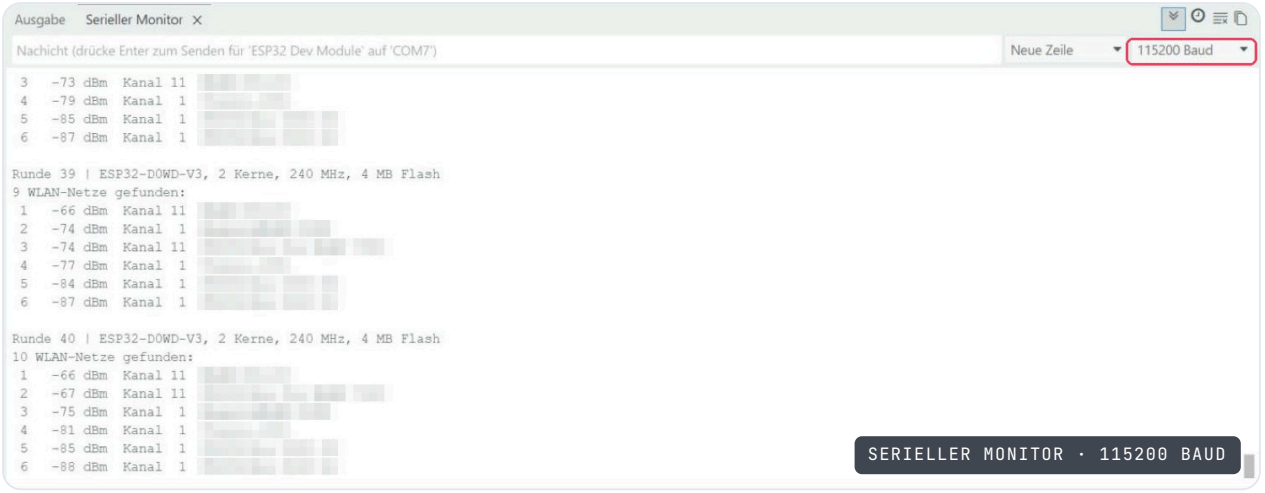
5 Sketch hochladen

Neuen Sketch anlegen, den Code von Seite 6 einfügen und mit dem Pfeil-Symbol hochladen (`Strg` + `U`). Das erste Kompilieren dauert etwa eine Minute. Unten erscheinen „Connecting…“, die Prozentanzeige und am Ende „Hard resetting via RTS pin…“.



6 Seriellen Monitor öffnen

`Strg` + `Umschalt` + `M` , unten rechts **115200 Baud** einstellen. Alle zehn Sekunden erscheint eine neue Runde mit Chip-Infos und den gefundenen Netzen. Die blaue LED „IO2“ blinkt vor jedem Scan dreimal.



i Upload hängt bei „Connecting…“?

Taste **BOOT** gedrückt halten, bis die Prozentanzeige startet, dann loslassen. Bei unserem Board war das im Test nicht nötig – der CH340C schaltet automatisch in den Upload-Modus.

05 Der erste Sketch

Kein Zubehör nötig: Der Sketch nutzt nur die blaue LED und das WLAN auf dem Board.

MAKEROO_ESP32_Start.ino

```
1  // MAKEROO · ESP32 DevKit (WROOM-32, CH340C) · Erster Start
2  // Blinkt die blaue LED (IO2) und optional eine LED an GPIO23,
3  // zeigt Chip-Infos und sucht alle 10 s nach WLAN-Netzen.
4  #include <WiFi.h>
5
6  const int LED_BOARD  = 2;   // blaue Onboard-LED "IO2"
7  const int LED_EXTERN = 23;  // optionale LED auf dem Steckbrett
8  int runde = 0;
9
10 void setup() {
11   Serial.begin(115200);
12   pinMode(LED_BOARD, OUTPUT);
13   pinMode(LED_EXTERN, OUTPUT);
14   WiFi.mode(WIFI_STA);        // WLAN als Client, ohne Verbindung
15   Serial.println("\nMAKEROO ESP32 - Erster Start");
16 }
17
18 void loop() {
19   for (int i = 0; i < 6; i++) {          // dreimal blinken
20     bool an = (i % 2 == 0);
21     digitalWrite(LED_BOARD, an);
22     digitalWrite(LED_EXTERN, an);
23     delay(250);
24   }
25   runde++;
26   Serial.printf("\nRunde %d | %s, %d Kerne, %lu MHz, %lu MB Flash\n",
27                 runde, ESP.getChipModel(), ESP.getChipCores(),
28                 ESP.getCpuFreqMHz(), ESP.getFlashChipSize() / 1048576);
29   int n = WiFi.scanNetworks();           // WLAN-Netze suchen
30   Serial.printf("%d WLAN-Netze gefunden:\n", n);
31   for (int i = 0; i < n && i < 6; i++) {
32     Serial.printf("%2d  %4d dBm  Kanal %2d  %s\n", i + 1,
33                   WiFi.RSSI(i), WiFi.channel(i), WiFi.SSID(i).c_str());
34   }
35   WiFi.scanDelete();
36   delay(10000);
37 }
```

Zeile 4
WLAN-Bibliothek
Gehört zum ESP32-Board-Paket – nichts extra installieren.

Zeile 6-7
Pins festlegen
GPIO2 ist die blaue LED „IO2“ auf dem Board, GPIO23 die optionale LED.

Zeile 14
WLAN einschalten
Als Client, ohne Verbindung – genug, um Netze zu suchen.

Zeile 19-24
Dreimal blinken
Zeigt sichtbar an, dass dein Sketch läuft.

Zeile 26-28
Chip-Infos
Modell, Kerne, Takt und Flash liest der ESP32 aus sich selbst aus.

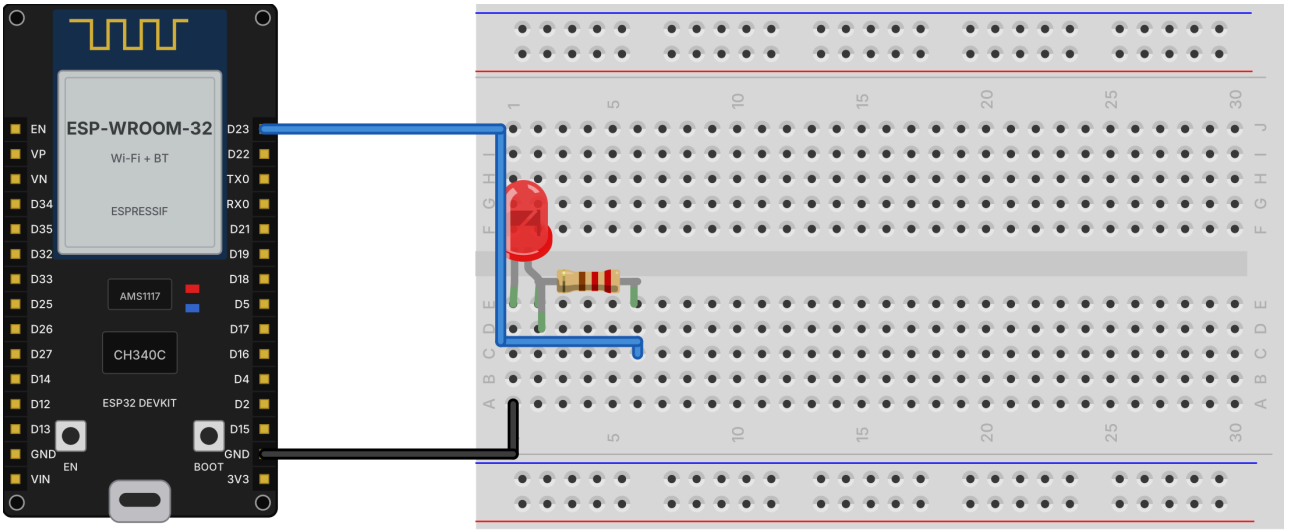
Zeile 29-34
WLAN-Scan
Listet bis zu sechs Netze mit Signalstärke (dBm) und Kanal.

Auf diesem Board getestet, Board-Paket esp32 3.3.12, „ESP32 Dev Module“: 860 kB Flash (67 %), 45 kB RAM (13 %).

06 Erweitern: LED am Steckbrett

Der Sketch schaltet GPIO23 schon mit. Mit einer LED und einem Widerstand siehst du, wie du eigene Bauteile an den ESP32 anschließt.

STECKPLATINEN-ANSICHT



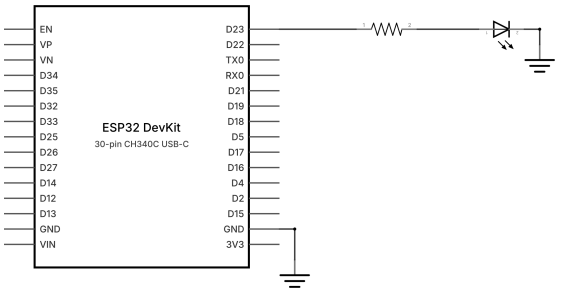
Verbindungen

ESP32		BAUTEIL
D23	→	220-Ω-Widerstand → LED-Anode (langes Bein)
GND	→	LED-Kathode (kurzes Bein)

Warum 220 Ω?
Bei 3,3 V und einer roten LED fließen damit rund 6 mA – hell genug und weit unter der Belastungsgrenze eines GPIO.

LED bleibt dunkel?
Langes Bein (Anode) zeigt zum Widerstand, kurzes Bein (Kathode) zu GND. Prüfe, ob das GND-Kabel wirklich auf der Reihe der Kathode steckt.

SCHALTPLAN



Verdrahtung erstellt mit Fritzing

Breites Board
Auf einem Steckbrett bleibt neben dem ESP32 nur eine Lochreihe je Seite frei. Für größere Aufbauten zwei Steckbretter nebeneinander nutzen – oder das Board wie hier daneben legen.

07 Grundlagen in Beispielen

Acht kurze Sketches zeigen, was im ESP32 steckt. Alle liegen auch als Dateien im Download – öffnen, hochladen, ausprobieren. Den Seriellen Monitor stellst du immer auf 115200 Baud.

Beispiel 1 Taster lesen

KEIN ZUBEHÖR

Die Taste **BOOT** hängt an GPIO0 und lässt sich nach dem Start wie ein normaler Taster abfragen. Jeder Druck schaltet die blaue LED um.

ESP32_01_Taster.ino

```ino
// Beispiel 1: Die BOOT-Taste schaltet die blaue LED um
const int TASTER = 0;    // BOOT-Taste liegt an GPIO0
const int LED = 2;       // blaue LED "IO2"
bool ledAn = false;
bool vorher = HIGH;

void setup() {
  Serial.begin(115200);
  pinMode(TASTER, INPUT_PULLUP);   // offen = HIGH, gedrückt = LOW
  pinMode(LED, OUTPUT);
}

void loop() {
  bool jetzt = digitalRead(TASTER);
  if (vorher == HIGH && jetzt == LOW) {   // gerade gedrückt
    ledAn = !ledAn;
    digitalWrite(LED, ledAn);
    Serial.println(ledAn ? "LED an" : "LED aus");
    delay(50);                            // Taster entprellen
  }
  vorher = jetzt;
}
```

Kompiliert für „ESP32 Dev Module“: 265 kB Flash (20 %), 22 kB RAM (6 %).

BEISPIELAUSGABE · 115200 BAUD

```
LED an
LED aus
LED an
```

Gut zu wissen

`INPUT_PULLUP` schaltet den internen Pull-up-Widerstand ein: Offen liest der Pin HIGH, gedrückt LOW. Einen externen Taster schließt du genauso zwischen einen GPIO (z. B. D27) und GND an.

Beispiel 2 LED dimmen mit PWM

KEIN ZUBEHÖR

PWM schaltet die LED tausende Male pro Sekunde ein und aus – je länger die Ein-Phase, desto heller. Der ESP32 kann das an fast jedem Ausgangspin.

ESP32_02_Dimmen.ino

```ino
// Beispiel 2: LED weich auf- und abdimmen (PWM)
const int LED = 2;

void setup() {
  ledcAttach(LED, 5000, 8);   // 5 kHz, 8 Bit = Helligkeit 0 bis 255
}

void loop() {
  for (int h = 0; h <= 255; h++) {   // heller werden
    ledcWrite(LED, h);
    delay(4);
  }
  for (int h = 255; h >= 0; h--) {   // dunkler werden
    ledcWrite(LED, h);
    delay(4);
  }
}
```

Kompiliert für „ESP32 Dev Module“: 257 kB Flash (20 %), 22 kB RAM (6 %).

Ergebnis

Die blaue LED „atmet“: rund eine Sekunde heller, eine Sekunde dunkler.

Gut zu wissen

`ledcAttach(pin, frequenz, bits)` ist die Schreibweise ab Board-Paket 3.x. `analogWrite(pin, 0…255)` funktioniert ebenfalls. Mit 50 Hz und 16 Bit steuerst du so auch Servos.

Beispiel 3 Spannung messen

POTENTIOMETER 10 KΩ

Der ESP32 misst Spannungen von 0 bis 3,3 V mit 12 Bit. `analogReadMilliVolts()` rechnet den Rohwert gleich mit den Kalibrierdaten des Chips in Millivolt um.

ESP32_03_Analog.ino

```ino
// Beispiel 3: Spannung messen – Potentiometer an D34 (ADC1)
const int POTI = 34;

void setup() {
  Serial.begin(115200);
  analogReadResolution(12);              // 12 Bit: 0 bis 4095
}

void loop() {
  int roh = analogRead(POTI);
  int mv = analogReadMilliVolts(POTI);   // kalibriert, in Millivolt
  Serial.printf("Rohwert %4d  ->  %4d mV\n", roh, mv);
  delay(500);
}
```

Kompiliert für „ESP32 Dev Module“: 271 kB Flash (21 %), 22 kB RAM (6 %).

Aufbau

3V3	→	Poti, äußeres Bein
GND	→	Poti, anderes äußeres Bein
D34	→	Poti, Mittelbein (Schleifer)

BEISPIELAUSGABE · 115200 BAUD

```
Rohwert    0  ->   142 mV
Rohwert 2051  ->  1712 mV
Rohwert 4095  ->  3138 mV
```

Gut zu wissen

Nimm für Analogwerte die Pins D32–D39 (ADC1). Die ADC2-Pins sind blockiert, sobald WLAN läuft. Mehr als 3,3 V nur über einen Spannungsteiler messen.

Beispiel 4 Berührung erkennen

DRAHT ODER ALUFOLIE

Zehn Pins des ESP32 sind kapazitive Touch-Sensoren. Ein Stück Draht oder Alufolie an D4 genügt – berührst du es, sinkt der Messwert deutlich.

ESP32_04_Touch.ino

```ino
// Beispiel 4: Berührung erkennen – Draht an D4 (Touch0)
const int TOUCH = 4;
const int LED = 2;
unsigned long ruhe;

void setup() {
  Serial.begin(115200);
  pinMode(LED, OUTPUT);
  delay(200);
  ruhe = touchRead(TOUCH);               // Wert ohne Berührung merken
  Serial.printf("Ruhewert: %lu\n", ruhe);
}

void loop() {
  unsigned long wert = touchRead(TOUCH);
  bool beruehrt = wert < ruhe * 7 / 10;  // Finger dran?
  digitalWrite(LED, beruehrt);
  Serial.printf("%3lu %s\n", wert, beruehrt ? "<- beruehrt" : "");
  delay(200);
}
```

Kompiliert für „ESP32 Dev Module“: 278 kB Flash (21 %), 22 kB RAM (6 %).

Aufbau

D4	→	Draht, Alufolie oder Münze

BEISPIELAUSGABE · 115200 BAUD

```
Ruhewert: 74
 73
 72
 15 <- beruehrt
 14 <- beruehrt
```

Gut zu wissen

Die Schwelle liegt bei 70 % des Ruhewerts, den der Sketch beim Start misst. Beim Einschalten das Touch-Feld also nicht berühren. Touch-fähig: D4, D2, D15, D13, D12, D14, D27, D33, D32.

08 WLAN & Internet

Beispiel 5 ## Webseite zum Schalten

SMARTPHONE ODER LAPTOP

Der ESP32 öffnet sein eigenes WLAN „MAKEROO-ESP32“ und liefert eine kleine Webseite aus. Damit schaltest du die blaue LED vom Handy – ganz ohne Router und Internet.

ESP32_05_Webserver.ino

```ino
// Beispiel 5: Eigener Hotspot mit Webseite – LED per Handy schalten
#include <WiFi.h>
#include <WebServer.h>

const char* NAME = "MAKEROO-ESP32";    // so heißt das WLAN
const char* PASSWORT = "makeroo123";   // mindestens 8 Zeichen
const int LED = 2;
bool ledAn = false;
WebServer server(80);

void seite() {
  String html = "<!doctype html><meta name='viewport' "
                "content='width=device-width'><style>body{font-family:"
                "sans-serif;text-align:center}button{font-size:2em;"
                "width:40%}</style><h1>MAKEROO ESP32</h1>";
  html += ledAn ? "<p>LED ist AN</p>" : "<p>LED ist AUS</p>";
  html += "<a href='/an'><button>An</button></a> "
          "<a href='/aus'><button>Aus</button></a>";
  server.send(200, "text/html", html);
}

void schalten(bool an) {
  ledAn = an;
  digitalWrite(LED, an);
  seite();
}

void setup() {
  Serial.begin(115200);
  pinMode(LED, OUTPUT);
  WiFi.softAP(NAME, PASSWORT);           // ESP32 wird zum Hotspot
  Serial.print("Hotspot aktiv: http://");
  Serial.println(WiFi.softAPIP());       // meist 192.168.4.1
  server.on("/", seite);
  server.on("/an", [] { schalten(true); });
  server.on("/aus", [] { schalten(false); });
  server.begin();
}

void loop() {
  server.handleClient();                 // Anfragen beantworten
}
```

Kompiliert für „ESP32 Dev Module“: 902 kB Flash (70 %), 46 kB RAM (14 %).



Die Seite aus dem Sketch im Browser

So testest du

1. Sketch hochladen.
2. Am Handy das WLAN **MAKEROO-ESP32** wählen, Passwort **makeroo123**.
3. Im Browser `192.168.4.1` öffnen.
4. Auf **An** und **Aus** tippen – die blaue LED folgt.

BEISPIELAUSGABE · 115200 BAUD

```
Hotspot aktiv: http://192.168.4.1
```

Gut zu wissen

Das Handy meldet evtl. „Kein Internet“ – einfach verbunden bleiben. Ändere das Passwort, bevor du das Board dauerhaft betreibst.

Beispiel 6 Heim-WLAN und Uhrzeit

DEIN WLAN (2,4 GHZ)

So verbindet sich der ESP32 mit deinem Router und holt die genaue Uhrzeit aus dem Internet – inklusive automatischer Sommerzeit. Trage in Zeile 4 und 5 deine WLAN-Daten ein.

ESP32_06_WLAN_Uhrzeit.ino

```
1  // Beispiel 6: Heim-WLAN und Uhrzeit aus dem Internet
2  #include <WiFi.h>
3
4  const char* WLAN_NAME = "DEIN-WLAN";          // anpassen
5  const char* WLAN_PASSWORT = "DEIN-PASSWORT";  // anpassen
6
7  void setup() {
8    Serial.begin(115200);
9    WiFi.begin(WLAN_NAME, WLAN_PASSWORT);
10   Serial.print("Verbinde");
11   while (WiFi.status() != WL_CONNECTED) {
12     delay(500);
13     Serial.print(".");
14   }
15   Serial.print("\nVerbunden, IP-Adresse: ");
16   Serial.println(WiFi.localIP());
17   // Zeitzone Deutschland inkl. Sommerzeit, Zeitserver pool.ntp.org
18   configTzTime("CET-1CEST,M3.5.0,M10.5.0/3", "pool.ntp.org");
19 }
20
21 void loop() {
22   struct tm zeit;
23   if (getLocalTime(&zeit)) {
24     Serial.println(&zeit, "%d.%m.%Y  %H:%M:%S");
25   }
26   delay(1000);
27 }
```

BEISPIELAUSGABE · 115200 BAUD

```
Verbinde.....
Verbunden, IP-Adresse: 192.168.178.42
20.10.2026  14:52:07
20.10.2026  14:52:08
```

Gut zu wissen

Die IP-Adresse vergibt dein Router. Mit ihr erreichst du später einen Webserver im Heimnetz – kombiniere dafür Beispiel 5 und 6: `WiFi.begin()` statt `WiFi.softAP()`.

Kompiliert für „ESP32 Dev Module“: 890 kB Flash (69 %), 47 kB RAM (14 %).

Beispiel 7 Strom sparen mit Deep Sleep

KEIN ZUBEHÖR

Im Tiefschlaf ist fast der ganze Chip aus. Ein Timer weckt ihn nach zehn Sekunden, dann beginnt der Sketch wieder bei `setup()`. Variablen mit `RTC_DATA_ATTR` bleiben erhalten.

ESP32_07_DeepSleep.ino

```
1  // Beispiel 7: Deep Sleep – aufwachen, messen, schlafen
2  RTC_DATA_ATTR int starts = 0;   // überlebt den Tiefschlaf
3
4  void setup() {
5    Serial.begin(115200);
6    delay(500);
7    starts++;
8    Serial.printf("Aufgewacht! Start Nr. %d\n", starts);
9    // ... hier messen und Werte senden ...
10   Serial.println("Schlafe 10 Sekunden.");
11   Serial.flush();
12   esp_sleep_enable_timer_wakeup(10ULL * 1000000);  // 10 s in µs
13   esp_deep_sleep_start();
14 }
15
16 void loop() {
17   // nie erreicht: nach dem Aufwachen startet setup() neu
18 }
```

BEISPIELAUSGABE · 115200 BAUD

```
Aufgewacht! Start Nr. 1
Schlafe 10 Sekunden.
…
Aufgewacht! Start Nr. 2
Schlafe 10 Sekunden.
```

Gut zu wissen

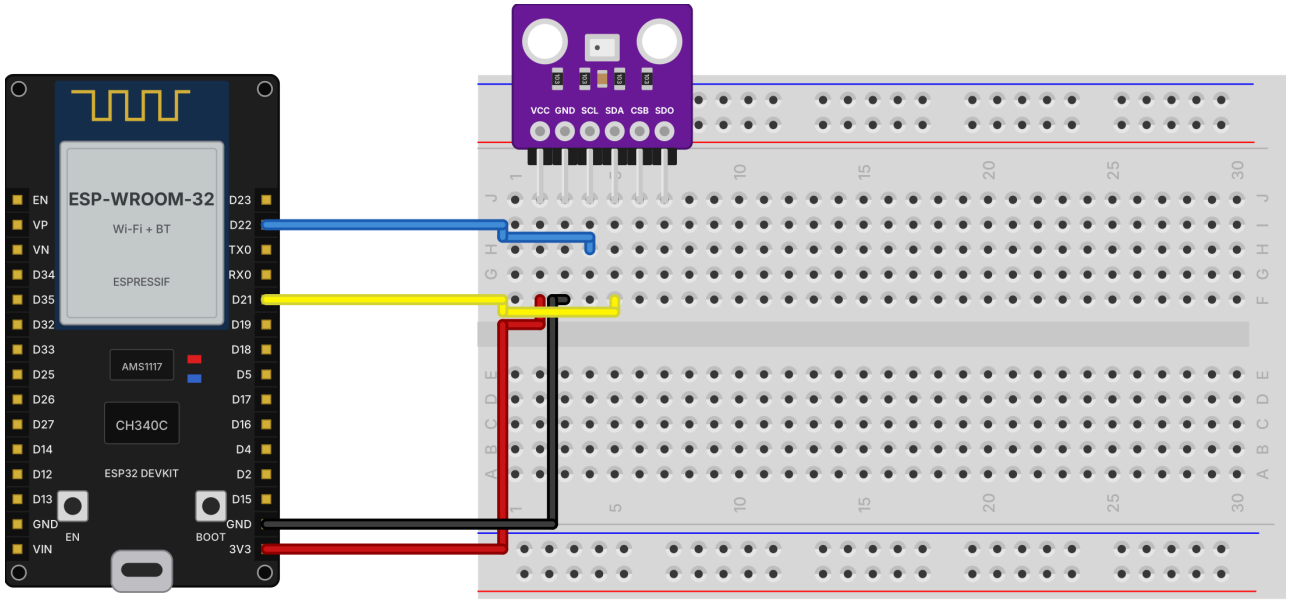
Zwischen den Meldungen gibt der Chip selbst ein paar Startzeilen aus (z. B. `rst:0x5 (DEEPSLEEP_RESET)`) – das ist normal. Auf dem DevKit verbrauchen USB-Wandler und LED weiter Strom; für Akkuprojekte zählt vor allem die lange Schlafzeit.

Kompiliert für „ESP32 Dev Module“: 269 kB Flash (21 %), 22 kB RAM (6 %).

09 Sensoren am I2C-Bus

Die meisten Sensoren und Displays sprechen I2C – zwei Leitungen für beliebig viele Geräte. Beim ESP32 liegt der Bus standardmäßig auf D21 (SDA) und D22 (SCL). Hier am Beispiel unseres BMP280, der ohnehin 3,3 V braucht.

STECKPLATINEN-ANSICHT · ESP32 + BMP280



Beispiel 8 I2C-Scanner

I2C-SENSOR, Z. B. BMP280

Der Scanner fragt alle 127 möglichen Adressen ab und listet, wer antwortet. So prüfst du in Sekunden, ob ein Sensor richtig verkabelt ist und welche Adresse er hat.

ESP32_08_I2C_Scanner.ino

```ino
// Beispiel 8: I2C-Scanner – welche Sensoren hängen am Bus?
#include <Wire.h>

void setup() {
  Serial.begin(115200);
  Wire.begin(21, 22);               // SDA = D21, SCL = D22
}

void loop() {
  int anzahl = 0;
  Serial.println("Suche I2C-Geraete ...");
  for (byte adr = 1; adr < 127; adr++) {
    Wire.beginTransmission(adr);
    if (Wire.endTransmission() == 0) {   // Geraet antwortet
      Serial.printf("  gefunden: 0x%02X\n", adr);
      anzahl++;
    }
  }
  Serial.printf("%d Geraet(e) gefunden\n\n", anzahl);
  delay(5000);
}
```

BEISPIELAUSGABE · 115200 BAUD

```
Suche I2C-Geraete ...
  gefunden: 0x76
1 Geraet(e) gefunden
```

Kompiliert für „ESP32 Dev Module“: 287 kB Flash (22 %), 23 kB RAM (7 %).

10 UNO-Anleitungen auf den ESP32 übertragen

Unsere Sensor-Guides nutzen den UNO R3. Mit dieser Tabelle baust du dieselben Schaltungen am ESP32 auf – die Sketches laufen meist unverändert, nur die Pinnummern ändern sich.

AM UNO R3	AM ESP32	WORAUF DU ACHTEN MUSST
`5V`	`3V3`	Nur wenn der Sensor 3,3 V verträgt (BMP280, BH1750, VL53L0X, MPU-6050 …). Braucht er 5 V, an **VIN** und seinen Ausgang per Spannungsteiler anpassen.
`GND`	`GND`	Immer verbinden.
`A4 (SDA)`	`D21`	I2C-Daten; `Wire.begin()` nutzt D21 automatisch.
`A5 (SCL)`	`D22`	I2C-Takt.
`A0 … A3`	`D32 … D35`	ADC1-Pins, funktionieren auch mit WLAN. Wertebereich 0–4095 statt 0–1023.
`D2, D3`	`D4, D27`	Freie digitale Pins ohne Sonderfunktion beim Booten.
`D9, D10`	`D16, D17`	Beliebige GPIOs für Trigger/Echo, Daten usw.
`SoftwareSerial`	`Serial2`	Echte zweite Schnittstelle: `Serial2.begin(9600, SERIAL_8N1, 16, 17)` – RX an D16, TX an D17.
`9600 Baud`	`115200 Baud`	Optional – der ESP32 schafft deutlich mehr. Im Monitor dieselbe Rate einstellen.

Kein 5-V-Signal an die GPIOs

Sensoren mit 5-V-Ausgang (z. B. HC-SR04-Echo, HC-SR501) über einen Spannungsteiler anschließen, etwa 1 kΩ in Reihe und 2 kΩ nach GND. Das senkt 5 V auf ungefähr 3,3 V.

Bibliotheken

Adafruit-, Pololu- und die meisten anderen Arduino-Bibliotheken unterstützen den ESP32. Du installierst sie wie gewohnt über den Bibliotheksverwalter.

11 Wenn es nicht klappt

SYMPTOM	LÖSUNG
Kein Port in der IDE	Anderes USB-C-Kabel probieren – viele Kabel laden nur. Danach den CH340-Treiber installieren (Schritt 2).
„Failed to connect to ESP32" bzw. „Timed out ... packet header"	Beim Hochladen die Taste **BOOT** gedrückt halten, bis „Connecting..." in Prozente übergeht. Hilft das nicht: `Werkzeuge → Upload Speed → 115200`.
Nur wirre Zeichen im Monitor	Baudrate unten rechts auf 115200 stellen.
Keine Startmeldung	Der Monitor war beim Start noch zu. Taste **EN** drücken – der ESP32 startet neu.
„Brownout detector was triggered", ständige Neustarts	Der USB-Port liefert zu wenig Strom. Anderen Port, aktiven Hub oder kürzeres Kabel nutzen.
0 WLAN-Netze gefunden	Der ESP32 sieht nur 2,4-GHz-Netze. Board nicht auf Metall legen, Antenne (Leiterbahn am Modulende) frei lassen.
Upload klappt nur ohne Schaltung	An D12, D2, D15 oder D5 hängt etwas, das den Boot-Pegel verändert. Diese Pins beim Hochladen frei lassen.

12 Wie geht's weiter?

Mit ESPHome & Home Assistant

Ohne eigenen Code: So erscheint die blaue LED als Schalter in Home Assistant.

```
esphome:
  name: makeroo-esp32
esp32:
  board: esp32dev
wifi:
  ssid: !secret wifi_ssid
  password: !secret wifi_password
api:
output:
  - {platform: gpio, pin: GPIO2, id: led}
light:
  - {platform: binary, name: LED, output: led}
```

Projektideen

WLAN-Thermometer
DHT22 an D4, Werte im Browser oder in Home Assistant

Handy-Schalter
Relais an D23, per Webseite vom Smartphone schalten

Messknoten mit Akku
Messen, senden, Deep Sleep – monatelang autark

Passt dazu

ESP32 Gehäuse aus Acryl
Schützt das Board
`MKR-ESP32-HOUSING-045`

0,96" OLED-Display
Werte direkt anzeigen
`MKR-0.96-OLED-015`

DHT22 Temperatur & Feuchte
Erster WLAN-Sensor
`MKR-DHT22-009`

Relais KY-019
Verbraucher per WLAN schalten
`MKR-KY019-034`

Sicherheitshinweise. Elektronikmodul für Ausbildungs-, Entwicklungs- und Experimentierzwecke. Nur mit Kleinspannung betreiben, niemals an Netzspannung. Nicht geeignet für Kinder unter 14 Jahren; Nutzung nur unter Aufsicht. Elektroaltgeräte gehören nicht in den Hausmüll – Rückgabe kostenlos bei kommunalen Sammelstellen.

Verantwortlicher Wirtschaftsakteur (GPSR): Alexander Prez · makeroo · Hinterm Bahnhof 4 · 90513 Zirndorf · support@makeroo.de
Fragen? Schreib uns mit einem Foto deines Aufbaus.
Arduino ist eine Marke der Arduino SA, ESP32 und ESPHome sind Marken ihrer Inhaber und dienen nur der Kompatibilitätsangabe.