Mikrocontroller ESP32

Das umfassende Handbuch

DIE LESEPROBE

Kapitel 1
Der Mikrocontroller ESP32

Dieses Kapitel soll Sie mit der Hardware ESP32 vertraut machen. In ihm stelle ich Ihnen die implementierten Funktionalitäten des Chips vor, die offiziellen auf dem Chip basierenden Module und verschiedene Boards, die diese Module enthalten. Außerdem erfolgt eine Festlegung auf das in diesem Buch verwendete Entwicklungsboard.

In diesem Kapitel möchte ich Ihnen zunächst einige Begriffe und Bezeichnungen erläutern, die Ihnen bei der Arbeit mit dem ESP32 oft begegnen werden. Nach einer kurzen Vorstellung der unterschiedlichen Boards, Chips und Module können Sie in Abschnitt 1.6 mit der Installation loslegen.

1.1 Ein kurzer Rückblick in die Entwicklung von Mikrocontrollern

Dieser kleine Rückblick zur Entwicklung von Mikrocontrollern und ihrem Weg in die Maker-Szene soll Ihnen ein besseres Gespür für das Umfeld vermitteln, in dem wir uns bewegen.

Als Mikrocontroller (auch µController, µC, MCU oder Einchipmikrorechner) werden Halbleiterchips bezeichnet, die einen Prozessor und zugleich auch Peripheriefunktionen enthalten. Sie sind Mitte der 1970er-Jahre aus dem Mikroprozessor hervorgegangen; auch heute noch ist der Übergang zwischen Mikrocontroller und Mikroprozessor fließend.

Die Verwendung von Mikrocontrollern war bis weit in die 2000er-Jahre professionellen Entwicklern und Nerds vorbehalten. Gamechanger war der Arduino, der 2005 basierend auf einem ATmega168 ein Development-Board mit einer leicht zu bedienenden und verständlichen Entwicklungsumgebung einführte. Darauf folgte eine sich durch Nachfrage und Angebot selbst befeuernde und unglaublich rasante Entwicklung, die ein nahezu unübersichtliches Angebotsspektrum zur Folge hatte. Und die Entwicklung ist sicherlich noch nicht beendet.

In diesem Umfeld gibt es Klassiker, wie z. B.:

- einen **ATiny** (z. B. ATtiny85): gut geeignet bei wenig Platz; geringer Stromverbrauch; allerdings nur beschränkte Anzahl GPIOs (*General Purpose Input Output Pins*), blanker IC
- einen **Arduino** auf ATmega168-Basis (z. B. Arduino Nano, Arduino Uno): maßgeschneiderte Entwicklungsumgebung; zwischenzeitlich recht teuer; kein WiFi
- den **ESP8266**: leistungsfähiger Mikrocontroller; viele Boards erhältlich; kein Bluetooth; im Vergleich zu einem ESP32 deutlich weniger GPIOs

Die Klassiker haben also durchaus noch ihre bevorzugten Einsatzgebiete. Das gilt insbesondere für den ESP8266, der in vielen industriellen Geräten wie WiFi-Steckdosen oder WiFi-Glühbirnen verbaut ist. Der ESP8266, der im August 2014 mit dem ESP-01-Modul Eingang in die Maker-Szene gefunden hat, wird von der chinesischen Firma *Espressif* hergestellt. Aus dem gleichen Hause stammen auch die ESP32-SoCs.

1.2 Die ESP32-SoCs

Unter *System-on-a-Chip* (SoC, dt. »Ein-Chip-System«), auch System-on-Chip, versteht man die Integration aller oder eines großen Teils der Funktionen eines programmierbaren elektronischen Systems auf einem Chip.

Hersteller der ESP32-SoCs ist die chinesische Firma *Espressif*. Bei diesen SoCs handelt es sich um verschiedene 32-Bit-Mikrocontroller. Die ersten Modelle kamen 2016 auf den Markt und zielen auf Anwendungen aus den Bereichen mobile Geräte, Wearables und *Internet of Things* (IoT). Espressif hat in den folgenden Jahren, beflügelt durch den Erfolg, seine Produktpalette dramatisch ausgebaut. Zwischenzeitlich umfasst das Portfolio unter dem Namen oder Überbegriff »ESP32« mehrere SoC-Serien:

- ESP32
- ESP32-S2
- ESP32-S3
- ESP32-C3
- ESP32-C6
- ESP32-H6

In Tabelle 1.2 finden Sie einen Überblick über die Fähigkeiten der verschiedenen Modelle anhand der Merkmale *Prozessor*, *SRAM* (statisches RAM), *ROM* (Festwertspeicher), *WiFi*, *BLE* (Blue Tooth Low Energy), *RTC* (Real Time Clock; dt.: Echtzeituhr) und *Flash Encryption*.

	ESP32	ESP32-S2	ESP32-S3	ESP32-C3	ESP32-C6	ESP32-H2
Prozessor	Tensilica Xtensa 32-bit 240 MHz dual core	Tensilica Xtensa 32-bit 240 MHz single-core	Tensilica Xtensa 32 bit 240 MHz dual-core	RISC V 32-bit 160 MHz	RISC V 32-bit 160 MHz	RISC V 32-bit 160 MHz
SRAM	520 KB	320 KB	520 KB	400 KB	512 KB	320 KB
ROM	448 KB	128 KB	384 KB	384 KB	320 KB	128 KB
WiFi	WiFi 4	WiFi 4	WiFi 4	WiFi 4	WiFi 6	WiFi 6
BLE	BLE 4.2	–	BLE 5.0	BLE 5.0	BLE 5.0	BLE 5.0
RTC	16 KB	16 KB	16 KB	8 KB	16 KB	–
Flash Encryption	×	XTS-AES-128/256	×	XTS-AES-128	XTS-AES-128	XTS-AES-128

Tabelle 1.1 Die Module der ESP32-Familie

Abbildung 1.1 zeigt – im Vorgriff – diverse ESP32-Boards.

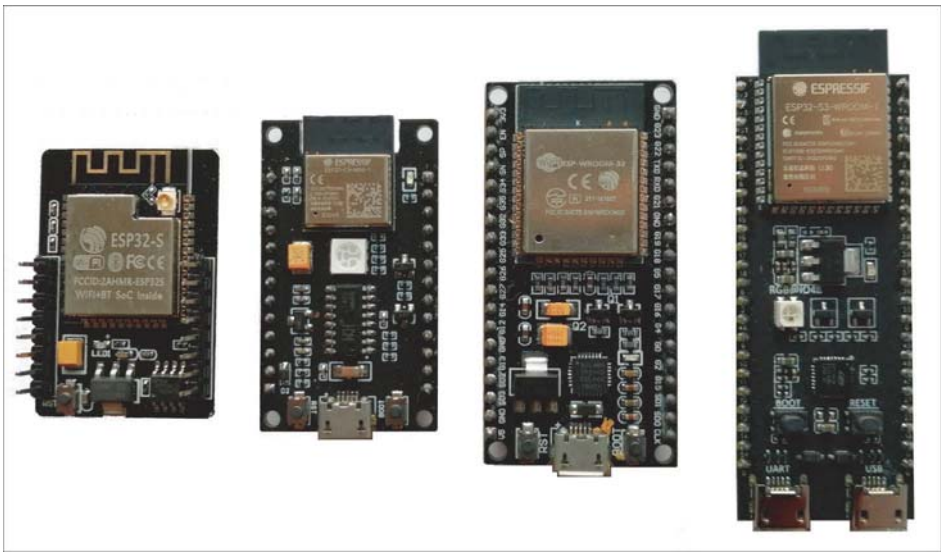


Abbildung 1.1 Boards mit diversen SoCs (ESP32-S, ESP32-C3-Mini, ESP32-WROOM, ESP32-S3

Nicht jeder SoC ist für jedes Projekt gleichermaßen geeignet. Die folgende Liste ist der Versuch einer kleinen Hilfestellung.

- **Rechenleistung und Prozessor**

  Bei der Auswahl eines ESP-Mikrocontrollers für Ihr IoT-Projekt ist die Frage von entscheidender Bedeutung, wie viel Rechenleistung Sie benötigen. Hohe Anforderungen an die Rechenleistung erfordern einen Dual-Core-Prozessor, wie ihn der ESP32 und der ESP32-S3 mitbbringen, während Single-Core-Prozessoren wie der ESP32-C3, ESP32-S2 oder ESP32-H2 zwar weniger Leistung bieten, dafür aber günstiger und stromsparender sind.

- **Memory**

  Ein nicht zu unterschätzender Faktor ist die Speicherkapazität. Der ESP32 bzw. der ESP32-S3 bieten sich für Projekte mit hohem Speicherbedarf an. Das kann auch das Programmieren einfacher machen, wenn Sie nicht unbedingt auf jedes Kilobyte achten müssen.

- **Energieverbrauch**

  Bei IoT-Projekten, die mit einer Batterie oder einem Akku betrieben werden sollen, ist Energieeffizienz von entscheidender Bedeutung. Der ESP32-C6, ESP32-C3 oder ESP32-H2 verwenden RISC-V-32-Bit-Prozessoren, die so konzipiert sind, dass sie weniger Strom verbrauchen und somit ideal für batteriebetriebene Geräte und Wearables sind.

- **Konnektivität**

  ESP-Produkte sind für ihre WiFi- und BLE-Verbindungen bekannt, was sie zur idealen Wahl für IoT-Anwendungen macht. Die Angaben in Tabelle 1.1 geben eine Orientierung.

- **Sicherheit**

  Die Sicherheit von IoT-Geräten gewinnt auch in der heimischen Umgebung zunehmend an Bedeutung. Die SoCs ESP32-S2, ESP32-C3 oder ESP32-C6 haben hier Vorteile, da sie mit der Flash-Verschlüsselung XTS-AES-128/256 ausgestattet sind.

- **Zigbee- und Openthread-Kompatibilität**

  SoCs der Serien ESP32-C6 und ESP32-H2 verfügen über integrierte Funksender für *Openthread*- und *Zigbee*-Protokolle. Darüber hinaus ist die ESP32-C-Serie *Matter*-zertifiziert – das ist ein neuer Standard, der besonders für den Aufbau eines Smarthomes wichtig ist. Er soll dafür sorgen, dass unterschiedliche Geräte Informationen austauschen können. ESP32-S und ESP32-C können als Board-Router für Thread-Anwendungen verwendet werden und der ESP32-H2 kann als Funk-Coprozessor zum Einsatz kommen, was die drei zur idealen Wahl für Thread-, Matter- und Zigbee-basierte Projekte macht.

- **Entwicklungsumgebung und Programmiersprache**

  Nicht alle SoCs lassen sich über die beliebte *Arduino IDE* programmieren. Dies gilt z. B. noch für die SoCs, die mit einem RISC-V-Prozessor ausgestattet sind. Wenn Sie diese Geräte einsetzen wollen, müssen Sie sich in die ESP-Tools einarbeiten, die etwas weniger komfortabel als die Arduino IDE sind.

  Dies gilt ähnlich auch für die Programmiersprache, denn nicht alle Sprachen stellen umfassend Bibliotheken bereit. Hier ist C bzw. C++ eindeutig breiter aufgestellt.

- **Preis**

  Schließlich ist der Preis ein wesentlicher Faktor bei der Auswahl eines ESP-Produkts für Ihr IoT-Projekt. ESP-Produkte sind für ihr gutes Preis-Leistungs-Verhältnis bekannt. Dennoch kann der Unterschied zwischen einem auf einem Board verbauten etablierten ESP32 und einem ESP32-S3 erheblich sein.

Ich habe die Beispiele dieses Buches mit SoCs der ESP32-Serie umgesetzt. Gleichwohl lassen sich die Programme auch auf andere Mikrocontroller übertragen, sofern deren technische Gegebenheiten (z. B. Bluetooth) dies zulassen.

Aber auch bei der ESP32-Serie handelt es sich genau genommen um eine ganze Chipfamilie, die aus verschiedenen Chips besteht:

- ESP32-DOWD (zwei Kerne, kein Embedded Flash),
- ESP32-DOWD-Q6 (wie ESP32-DOWD, andere Abmessungen),
- ESP32-D2WD (zwei Kerne, 16 MBit Embedded Flash) und
- ESP32-SOWD (einfacher Prozessorkern, kein Embedded Flash).

Die übrigen Eigenschaften sind identisch.

Der Prozessorkern ist vom Typ *Xtensa LX6* und arbeitet mit einem Systemtakt von 160 MHz bis 240 MHz. Das Blockdiagramm in Abbildung 1.2 zeigt die wesentlichen Teile des Systems.

Die ESP32-Mikrocontroller weisen einige Eigenschaften auf, die sie für eine Vielzahl von Projekten interessant machen:

- Der Chip bietet **internen Speicher** in unterschiedlichen Konfigurationen:
  - 520 KB SRAM für Daten und Befehle
  - 448 KB ROM für das Booten und Kernfunktionen
  - 8 KB SRAM im RTC für Daten (RTC-FAST-Memory); sie werden von der CPU während des RTC-Boots aus dem Tiefschlaf gelesen.
  - 8 KB SRAM im RTC für Daten (RTC-SLOW-Memory); sie werden vom Coprozessor während des Tiefschlafs genutzt.
- Es werden bis zu **4 × 16 MB externer Speicher** unterstützt.

- Beim **Systemtakt** besteht die Möglichkeit, den internen Takt (8 MHz) oder einen externen Quarztakt mit üblicherweise 160 MHz zu nutzen. Dieser übernimmt auch bei einem Zurücksetzen des Prozessors das System-Timing.
- Ein **interner Hall-Sensor** kann für die Messung von Magnetfeldschwankungen genutzt werden.
- Ein **interner Temperatursensor** ist für einen Messbereich von –40 bis 125 Grad ausgelegt. Wie bei dem Hall-Sensor werden die analogen Messwerte von einem internen Analog-Digital-Wandler digitalisiert. Bei neueren Modellen ist der Temperatursensor jedoch nicht mehr vorhanden.
- Es sind **34 GPIOs** (universelle Ein- und Ausgänge) vorhanden. Sie dienen der Ein- und Ausgabe analoger und digitaler Signale. Die nach außen geführten Pins sind bezüglich ihrer Funktion überwiegend mehrfach belegt. Mit internen Pull-down- und Pull-up-Widerständen können definierte Zustände herbeigeführt werden.
- Der ESP32 kann Signale von bis zu zehn unterschiedlichen **kapazitiven Touch-Sensoren** verarbeiten. Die Eingänge sind diversen GPIO-Pins zugewiesen.
- **Transceiver**: Für die WLAN- und Bluetooth-Kommunikation hat der ESP32 interne 2,4-GHz-Sende- und -Empfangsbausteine.
- Das **WLAN** arbeitet nach dem Standard IEE 802.11 b/g/n.
- Unterstützt wird Bluetooth 4.2 (sowohl in der herkömmlichen als auch in der Betriebsart *Bluetooth Low Energy*).
- **UART** (*Universal Asynchronous Receiver Transmitter*) bezeichnet eine Form der seriellen Kommunikation. Hier stehen drei Bausteine zur Verfügung.
- Impulse bzw. Impulsflanken können mit acht **Impulszählern** erfasst werden. Bei Erreichen eines bestimmten Werts kann ein Interrupt ausgelöst werden.
- Der ESP32 verfügt über **vier 64-Bit-Universal-Timer**. Sie können softwareseitig gesteuert werden und Interrupts erzeugen.
- Es stehen **drei Watchdog-Timer** zur Verfügung. Dabei wird zwischen *Main-Watchdog-Timern* (davon zwei im Part der Universal-Timer) und *RTC-Watchdog-Timern* (einer im RTC-Modul) unterschieden. Ein Interrupt, ein Core-Reset oder ein CPU-Reset kann durch ein Reset des Watchdog-Timers ausgelöst werden.
- Verbaut ist ein **12-Bit-A/D-Wandler** (*Analog-Digital-Wandler*) mit 18 Kanälen.
- Es können **zwei unabhängige 8-Bit-DAC** (Digital-Analog-Wandler) genutzt werden.
- Der ESP verfügt über **drei periphere SPI-Schnittstellen** (*SPI1*, *HSPI* und *VSPI*), die im Master- oder Slave-Modus betrieben werden können.
- Es werden **zwei I2C-Bus-Schnittstellen** vorgehalten, die im Master- oder Slave-Modus betrieben werden können.

- **Pulsweitenmodulation** (PWM) dient dazu, Geräte wie Motoren, elektrische Heizungen oder Ähnliches zu steuern. Der Chip verfügt über ein programmierbares Hardware-PWM-Modul und 16 unabhängige Software-PWM-Module. Die Software-PWMs können bis zu 16 unabhängige digitale Wellenformen erzeugen. Tastverhältnis und Perioden sind frei wählbar. Software-PWMs eignen sich besonders zur LED-Ansteuerung.
- Ein **Infrarot-Controller**, der bis zu acht Kanäle einer programmierbaren Infrarotfernbedienung bedienen kann, rundet das Angebot an Schnittstellen ab.

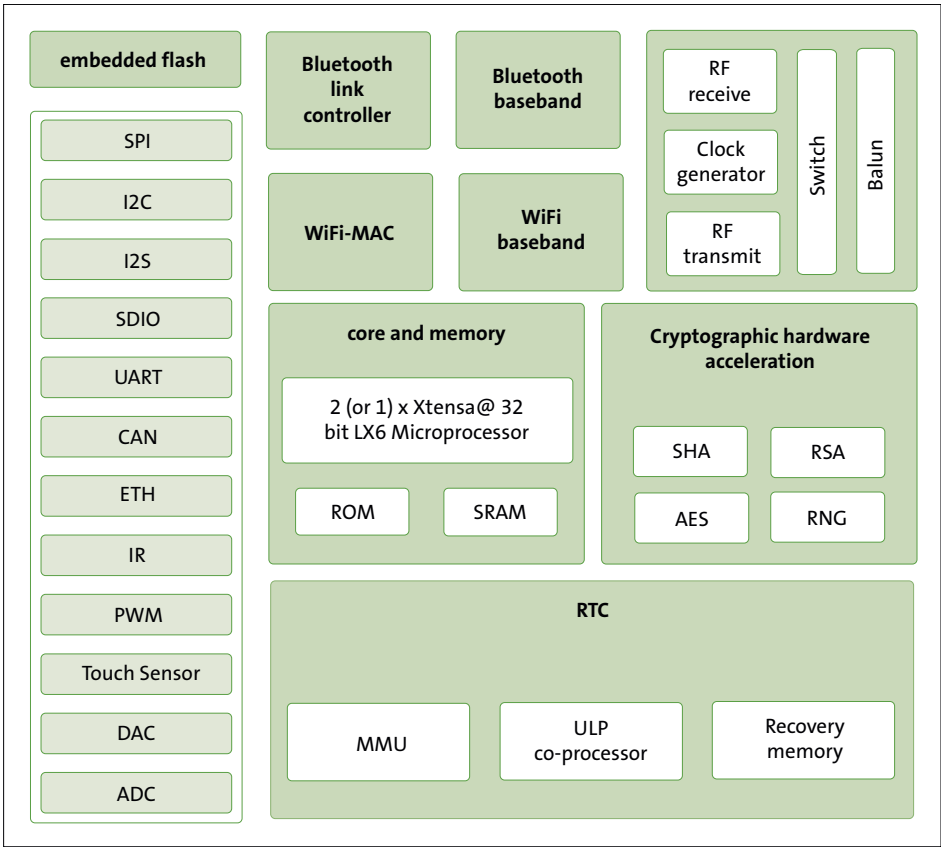


Abbildung 1.2 Blockschaltbild des ESP32

Weitere Einzelheiten können Sie dem Datenblatt entnehmen:

https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf

Darin wird Ihnen der Funktionsumfang detailliert erläutert, und auch bei allen Unklarheiten sollten Sie unbedingt dort nachsehen.

1.3 Die Module WROOM, SOLO, WROVER und PICO

Espressif entwickelt und fertigt verschiedene Module und Entwicklungsplatinen, um den Anwendern die Einschätzung des Potenzials der (reinen) ESP32-Serie zu erleichtern. Sie sind im Original oder auch als Klone bei den einschlägigen Bezugsquellen erhältlich.

Im Folgenden finden Sie eine kurze Beschreibung der wichtigsten Module, die von Espressif gefertigt werden. Weitere Informationen finden Sie unter *https://espressif.com/en/products/modules*.

Die ESP32-Modulfamilie basiert auf Modulen mit einigen integrierten Schlüsselkomponenten, einschließlich eines Quarzoszillators und einer Antennenanpassungsschaltung. Die Module sind vorgefertigte Lösungen für die Integration in Endprodukte. In Kombination mit einigen zusätzlichen Komponenten, wie einer Programmierschnittstelle, Pull-up-Widerständen und Stiftleisten, können diese Module auch zur Evaluierung der ESP32-Funktionalität verwendet werden.

Die wichtigsten Eigenschaften der verbreitesten Module sind in Tabelle 1.2 zusammengefasst. Einige zusätzliche Details werden in den folgenden Abschnitten behandelt. Vollständig kann diese Liste allerdings nicht sein – es werden ständig neue Modelle vorgestellt. Manche werden nur für den chinesischen Markt gefertigt, andere lassen sich nur in sehr kleinen Stückzahlen beziehen.

Modul	Chip	Flash (MB)	PSRAM (MB)	Antenne	Abmessungen (mm)
ESP32-WROOM-32	ESP32-D0W-DQ6	4	–	MIFA	18 × 25,5 × 3,1
ESP32-WROOM-32D	ESP32-D0WD	4, 8 oder 16	–	MIFA	18 × 25,5 × 3,1
ESP32-WROOM-32U	ESP32-D0WD	4, 8 oder 16	–	U.FL	18 × 25,5 × 3,1
ESP32-SOLO-1	ESP32-S0WD	4	–	MIFA	18 × 25,5 × 3,1
ESP32-WROVER (PCB)	ESP32-D0W-DQ6	4	8	MIFA	18 × 31,4 × 3,3
ESP32-WROVER (IPEX)	ESP32-D0W-DQ6	4	8	U.FL	18 × 31,4 × 3,3

Tabelle 1.2 Die Module der ESP32-Familie

Modul	Chip	Flash (MB)	PSRAM (MB)	Antenne	Abmessungen (mm)
ESP32-WROVER-B	ESP32-D0WD	4, 8 oder 16	8	MIFA	18 × 31,4 × 3,3
ESP32-WROVER-IB	ESP32-D0WD	4, 8 oder 16	8	U.FL	18 × 31,4 × 3,3

Tabelle 1.2 Die Module der ESP32-Familie (Forts.)

Zu den Angaben in der Tabelle gehören allerdings noch einige Erläuterungen:

- ESP32-**D...** kennzeichnet einen Dual-Core-Chip, ESP32-**S...** kennzeichnet einen Single-Core-Chip.
- MIFA steht für *Meandered Inverted-F Antenna*.
- U.FL und U.FL/IPEX sind Antennen-Konnektoren, die für Miniatur-HF-Steckverbinder ausgelegt sind.
- ESP32-WROOM-32x-, ESP32-WROVER-B- und ESP32-WROVER-IB-Module haben üblicherweise 4 MB Flash-Memory, sie sind aber auch mit Flash-Memory-Größen von 8 MB und 16 MB erhältlich.
- Die Erstversion des ESP32-WROVER-Moduls hatte 4 MB PSRAM.
- Der ESP32-WROOM-32 wurde früher als ESP-WROOM-32 bezeichnet.

ESP32-WROOM-32

Der ESP32-WROOM-32 ist ein grundlegendes und allgemein das gebräuchlichste ESP32-Modul mit integriertem ESP32-D0WDQ6-Chip. Es war das erste Modul der WROOM/WROVER-Familie, das auf den Markt kam (siehe Abbildung 1.3).

Abbildung 1.3 Das Modul »ESP-WROOM-32«, Vorder- und Rückseite

Dieses Modul wird mit den Pins direkt auf die Platine gelötet. Das Pinout des Moduls sehen Sie in Abbildung 1.4.

Das Datenblatt steht unter *https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf* zur Verfügung.

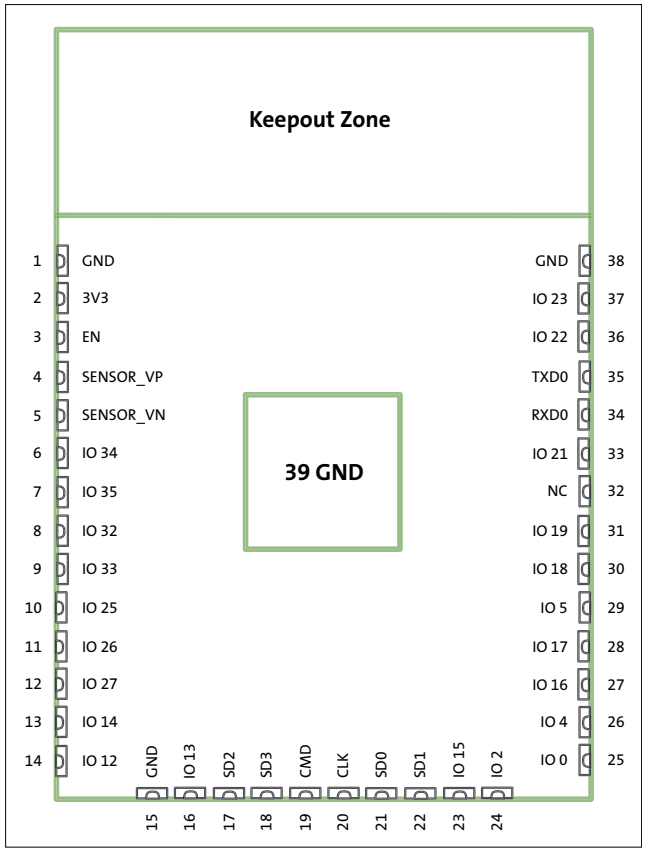


Abbildung 1.4 ESP32-WROOM-32, Pinout

ESP32-WROOM-32D und ESP32-WROOM-32U

Beide Module integrieren den ESP32-D0WD-Chip, der weniger Platz benötigt als der im ESP32-WROOM-32 installierte Chip ESP32-D0WDQ6. Der ESP32-WROOM-32U ist der kleinste Vertreter der gesamten WROOM/WROVER-Modulfamilie.

ESP32-SOLO-1

Der ESP32-SOLO-1 ist eine vereinfachte Version des ESP32-WROOM-32D-Moduls. Er enthält einen Single-Core-ESP32-Chip, der eine Taktfrequenz von bis zu 160 MHz unterstützt.

Die ESP32-WROVER-Serie

Die ESP32-WROVER-Serie besteht aus einigen Modifikationen der ESP32-WROOM-32x-Module, die unter anderem ein zusätzliches 8-MB-SPI-PSRAM (*Pseudo Static RAM*) enthalten.

- ESP32-WROVER (PCB) und ESP32-WROVER (IPEX) verfügen über ein PSRAM, das mit 1,8 V betrieben wird und eine Taktrate von bis zu 144 MHz unterstützt.
- ESP32-WROVER-B und ESP32-WROVER-IB verfügen über ein PSRAM, das mit 3,3 V betrieben wird und eine Taktrate von bis zu 133 MHz unterstützt.

1.4 Die Boards

Der ESP32-Chip wird auf einer Vielzahl von Boards verwendet. Der Hersteller Espressif führt selbst einige Module in seinem Programm auf.

- Das **ESP32-PICO-KIT** ist das kleinste Entwicklungsboard von Espressif. Es passt in ein Mini-Breadboard, ist aber mit der minimalen Anzahl diskreter Komponenten voll funktionsfähig, weil alle ESP32-Pins zugänglich sind.
- Das **ESP-WROVER-KIT** ist das vielseitige Entwicklungsboard von Espressif. Es hat eine lange Liste von Funktionen an Bord, z. B. LCD-Anzeige, JTAG, Kameraheader, RGB-LEDs usw. Es kann verschiedene Module aufnehmen (ESP32-WROVER, ESP-WROOM-32). Mehrere Stiftleisten und Steckbrücken machen die Anbindung und Konfiguration des ESP-WROVER-KIT sehr flexibel.
- Das **ESP32-Dev-KitC** ist eine Entwicklungsplattform für Einsteiger. Nahezu alle ESP32-Pins wurden herausgeführt und können einfach angeschlossen und verwendet werden.
- Das **ESP32-LyraTD-MSC** ist eines der Audio-Development-Boards von Espressif, die auf Spracherkennungsanwendungen ausgerichtet sind. Es unterstützt auch Nahfeld- und Fernfeld-Weckfunktionen.

Einen Überblick über die aktuellen Module und Boards von Espressif finden Sie auf der Seite *https://docs.espressif.com/projects/esp-idf/en/latest/hw-reference/modules-and-boards.html*.

Darüber hinaus wird der ESP32 auch von anderen Herstellern und Distributoren für eigene Boards eingesetzt:

- Das **LoLin32 ESP32 Development Board** ist in der Regel mit 4 MB Flash-Speicher und einer Lithium-Batterie ausgestattet.
- Das **ESP32-OLED Development Board** hat bereits ein OLED-Display.

- Das **SparkFun ESP32 Thing**:: Die Firma SparkFun bietet eine Fülle von Modulen, Sensoren und Platinen für eigene Entwicklungen an. Das von ihr vertriebene ESP32-Board hat bereits einige zusätzliche LEDs und Tasten.
- Das **ESP32-HiLetGo Development Board** kommt dem ESP32-Dev-KitC sehr nahe. Der Fritzing-Part wird zum Download angeboten.
- Das **Adafruit HUZZAH32 – ESP32 Feather Board** ist ein Development-Board von Adafruit. Auch hier wird der Fritzing-Part zum Download angeboten.

Es lässt sich im Vorhinein nur schwer einschätzen, ob Sie bei einer Bestellung dieser Geräte ein Original-Board von Espressif bekommen, einen mehr oder weniger gelungenen Klon erhalten oder stattdessen eine Eigenentwicklung eines Herstellers auspacken, die mehr oder weniger gut funktioniert. Sie müssen Vertrauen in die Bezugsquelle haben. Beim Erwerb preiswerter Ware aus Fernost lässt sich das Risiko etwas streuen, indem Sie das gleiche Bauteil bei unterschiedlichen Anbietern erwerben.

1.5 Das ESP32-Dev-KitC V4

Momentan zählt das ESP32-Dev-KitC V4 zu den gängigsten Entwicklungsboards auf Basis des ESP32. Aus diesem Grund verwende ich es für die Beispiele in diesem Buch. Die Kerninformationen zu diesem Modul lassen sich auch auf viele andere Varianten übertragen.

Abbildung 1.5 zeigt die wesentlichen Bauteile des ESP32-Development-Kit V4. Das Board selbst kann auf jedes passende Breadboard gesteckt werden. Durch seine Maße von ca. 53 mm × 25 mm lassen sich komfortabel sehr kompakte Schaltungen realisieren. Nur in den seltensten Fällen werden Sie auf das nackte Modul ausweichen müssen.

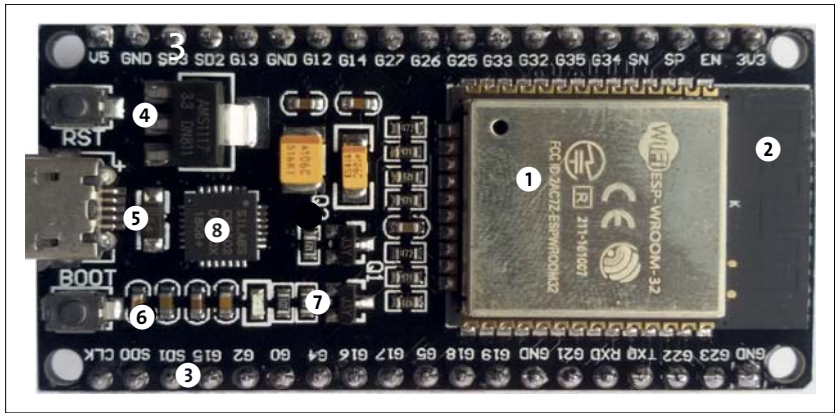


Abbildung 1.5 ESP32-Dev-KitC V4

❶ **ESP32-WROOM-32-Modul**: Bei manchen Boards kann hier auch das größere ESP32-WROVER-Modul vorhanden sein.

❷ **Antenne**

❸ **Header-Pins**: Die meisten Pins des ESP-Moduls sind auf die Pin-Header auf der Platine herausgeführt. Es gibt zwei Reihen zu je 19 Pins.

❹ **Reset-Knopf**: Bei manchen Boards wird diese Taste auch **EN** genannt.

❺ **USB-Anschluss**: Stromversorgung für das Board sowie die Kommunikationsschnittstelle zwischen einem Computer und dem ESP32-Modul.

❻ **Boot-Knopf** (Download-Knopf): Je nach gewählter Flash-Methode bzw. Entwicklungsumgebung ist diese Taste für das Aufspielen der Firmware zu drücken. Hier gibt es unterschiedliche Szenarien:

- **Arduino IDE**: Manchmal wird der Upload ohne zusätzliches Betätigen von Knöpfen initiiert. In der Regel müssen Sie aber den Boot-Knopf für das Initiieren des Uploads drücken.
- **Offiziell**: Halten Sie den Boot-Knopf gedrückt, um den Firmware-Download-Modus zum Herunterladen der Firmware über die serielle Schnittstelle zu starten, und drücken Sie gegebenenfalls anschließend EN für einen Reboot.

❼ **LED**: Sie leuchtet kurz auf, wenn an das Board Spannung angelegt wird. (Bei manchen alternativen Boards ist die LED auch mit GPIO2 verbunden.)

❽ **USB-UART-Bridge**: Verbaut ist eine CP2102-USB-to-UART-Bridge, die Transferraten bis zu 3 Mbps unterstützt.

Stromversorgung

Das Modul arbeitet mit einer Spannung von typischerweise +3,3 V. Die einzelnen Stromversorgungsmöglichkeiten sind:

- Micro-USB-Port: Dieser Port wird als hauptsächliche Stromversorgungsoption betrachtet. Da USB normalerweise mit +5 V arbeitet, wird die Spannung durch einen On-Board-Chip auf +3,3 V reduziert. Warnung: Periphere Geräte können nicht ausnahmslos und unbesehen an den 3V3-Ausgang angeschlossen werden, da der On-Board-Chip nur begrenzt belastbar ist.
- 5V/GND-Header-Pins
- 3V3/GND-Header-Pins

Je nach Einsatz (z. B. intensive WLAN-Nutzung) kann der ESP32 viel Strom ziehen. Laut Datenblatt können dies bis zu 500 mA sein.

1.5.1 Das Pinout

Das Pinout des ESP32-Dev-KitC V4 – soweit es in dem Buch von Bedeutung ist – finden Sie in Abbildung 1.6.

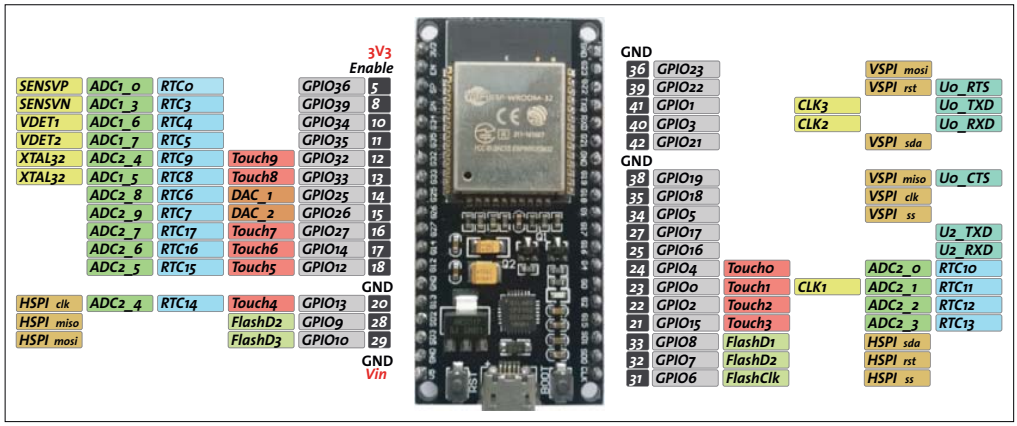


Abbildung 1.6 ESP32-Dev-KitC V4, Pinout

Die einzelnen Pins sind auf dem Board entsprechend bezeichnet. Dabei kann die aufgedruckte Beschriftung je nach Hersteller unterschiedlich sein. So bezeichnen manche Hersteller z. B. den Pin GPIO14 mit G14 (siehe Abbildung 1.6), manche mit D14 oder auch nur mit 14.

Auch ist es so, dass das Pinout nicht für alle Boards normiert ist; jeder Hersteller gönnt sich eigene Freiheiten. Im Extremfall kann sogar bei besonders einfach gehaltenen Klonen der Aufdruck bei einzelnen Pins gänzlich unzutreffend sein, obwohl die eigentliche Zuordnung richtig ist. Deshalb ist es besonders wichtig, sich vor jeder Inbetriebnahme mit dem Pinout zu befassen, und grundsätzlich ist natürlich etwas Vorsicht beim Anschluss geboten.

Im Schaubild aus Abbildung 1.6 ist den meisten Pins eine weiße Zahl in einem grauen Kasten zugeordnet. Sie verweist auf den entsprechenden Pin des Moduls.

Die GPIO-Pins sind spannungssensitiv. Es kann also zu Schädigungen oder einem Totalausfall des Mikrocontrollers führen, wenn die an einen Pin angelegte Spannung den Wert von 3,3 V überschreitet. Die Pins können nur mit maximal 40 mA belastet werden. Eine höhere Belastung kann ebenfalls zur Zerstörung des Boards führen.

Funktionen der GPIO-Pins

Das Schaubild zeigt auch, dass Pins durchaus mehrere Funktionen haben können. Darüber hinaus werden Sie bei akribischem Schauen feststellen, dass nicht alle GPIOs des Moduls auf dem Board nach außen geführt werden. Das ist dem Umstand geschuldet, dass manche Pins für interne Zwecke reserviert sind. Darüber hinaus gibt es

Pins mit spezifischen Merkmalen, die die allgemeine Funktion einschränken und sie nur für bestimmte Zwecke einsetzbar machen.

Eine Funktionsübersicht der Pins finden Sie in Tabelle 1.3.

GPIO	Input	Output	Bemerkungen
0	pulled up	OK	Gibt PWM-Signal beim Boot aus.
1	TX-Pin	OK	Debug-Output beim Boot
2	OK	OK	Verbunden mit Board-LED
3	OK	RX-Pin	HIGH beim Boot
4	OK	OK	
5	OK	OK	Gibt PWM-Signal beim Boot aus.
6	X	X	Verbunden mit internem SPI-Flash
7	X	X	Verbunden mit internem SPI-Flash
8	X	X	Verbunden mit internem SPI-Flash
9	X	X	Verbunden mit internem SPI-Flash
10	X	X	Verbunden mit internem SPI-Flash
11	X	X	Verbunden mit internem SPI-Flash
12	OK	OK	Wird bei einem Boot-Fehler auf HIGH gesetzt.
13	OK	OK	
14	OK	OK	Gibt PWM-Signal beim Boot aus.
15	OK	OK	Gibt PWM-Signal beim Boot aus.
16	OK	OK	
17	OK	OK	
18	OK	OK	
19	OK	OK	
20	OK	OK	
21	OK	OK	
22	OK	OK	

Tabelle 1.3 Funktionen der GPIO-Pins

GPIO	Input	Output	Bemerkungen
23	OK	OK	
24	OK	OK	
25	OK	OK	
26	OK	OK	
27	OK	OK	
28	OK	OK	
29	OK	OK	
30	OK	OK	
31	OK	OK	
32	OK	OK	
33	OK	OK	
34	OK		nur Input
35	OK		nur Input
36	OK		nur Input
39	OK		nur Input

Tabelle 1.3 Funktionen der GPIO-Pins (Forts.)

Interner SPI-Flash des ESP-WROOM-32

Die GPIOs 6 bis 11 können nicht genutzt werden, obwohl manche Boards sie nach außen führen. Der Grund ist, dass sie für den internen SPI reserviert sind.

Die GPIO-Pins 34 bis 39

Die GPIOs 34 bis 39 sind keine richtigen GPIOs, da sie nur als Input funktionieren (GPI). Sie haben keine internen Pull-up- oder Pull-down-Widerstände. Folgende GPIOs sind also nur Input-GPIOs:

- GPIO34
- GPIO35
- GPIO36
- GPIO39

PWM

Der ESP32-LED-PWM-Controller hat 16 unabhängige Kanäle, die mit unterschiedlichen Eigenschaften konfiguriert werden können. Alle richtigen Output-Pins können PWM generieren (also nicht die GPIOs 34 bis 39).

Interrupts

Alle GPIOs können für Interrupts konfiguriert werden.

Touch-GPIOs

Der ESP32 verfügt über zehn interne kapazitive Berührungssensoren. Diese können Schwankungen von elektrischen Ladungen registrieren, die z. B. durch Kontakt mit der menschlichen Haut entstehen (Berühren der GPIOs mit einem Finger). Ein Anwendungsfall ist unter anderem die Integration in kapazitive Touchpads, wo sie mechanische Tasten ersetzen können. Über die kapazitiven Touch-Pins kann der ESP32 auch aus dem Tiefschlaf erweckt werden. Die Touch-Pins sind folgendermaßen zugeordnet:

- T0 (GPIO4)
- T1 (GPIO0)
- T2 (GPIO2)
- T3 (GPIO15)
- T4 (GPIO13)
- T5 (GPIO12)
- T6 (GPIO14)
- T7 (GPIO27)
- T8 (GPIO33)
- T9 (GPIO32)

Analog to Digital Converter (ADC)

Der ESP32 hat 18 ADC-Input-Kanäle, die mit 12-Bit-Auflösung operieren. Das bedeutet, dass die Kanäle analoge Messwerte im Bereich von 0 bis 4095 aufnehmen können, wobei »0« 0 V und »4095« 3,3 V entspricht. Auflösung und ADC-Bereich können softwareseitig eingestellt werden.

- ADC1_CH0 (GPIO36)
- ADC1_CH1 (GPIO37)
- ADC1_CH2 (GPIO38)
- ADC1_CH3 (GPIO39)
- ADC1_CH4 (GPIO32)
- ADC1_CH5 (GPIO33)
- ADC1_CH6 (GPIO34)
- ADC1_CH7 (GPIO35)
- ADC2_CH0 (GPIO4)
- ADC2_CH1 (GPIO0)
- ADC2_CH2 (GPIO2)
- ADC2_CH3 (GPIO15)
- ADC2_CH4 (GPIO13)
- ADC2_CH5 (GPIO12)
- ADC2_CH6 (GPIO14)
- ADC2_CH7 (GPIO27)
- ADC2_CH8 (GPIO25)
- ADC2_CH9 (GPIO26)

ADC2-Pins und WLAN

Die ADC2-Pins können nicht gleichzeitig mit WLAN verwendet werden. Bei der Nutzung von WLAN muss auf die ADC1-Pins ausgewichen werden. Darüber hinaus ist der Verlauf nicht vollständig linear. So ist es nicht möglich, zwischen 0 und 0,1 V sowie zwischen 3,2 und 3,3 V zu unterscheiden.

Digital to Analog Converter (DAC)

Es gibt zwei DAC-Kanäle mit einer 8-Bit-Auflösung. Sie konvertieren digitale Signale oder analoge Werte in analoge Signale. Die Kanäle sind:

- DAC1 (GPIO25)
- DAC2 (GPIO26)

RTC-GPIOs

RTC steht für *Real Time Clock*. Der ESP32 bietet RTC-GPIO-Unterstützung. Das RTC-System selbst benötigt extrem wenig Strom und ist verantwortlich für die verschiedenen Energiesparmodi, die bis zum Tiefschlaf reichen. Mit den RTC-GPIOs kann der ESP32 geweckt werden. Die folgenden GPIOs können als externe Weckquelle verwendet werden:

- RTC_GPIO0 (GPIO36)
- RTC_GPIO3 (GPIO39)
- RTC_GPIO4 (GPIO34)
- RTC_GPIO5 (GPIO35)
- RTC_GPIO6 (GPIO25)
- RTC_GPIO7 (GPIO26)
- RTC_GPIO8 (GPIO33)
- RTC_GPIO9 (GPIO32)
- RTC_GPIO10 (GPIO4)
- RTC_GPIO11 (GPIO0)
- RTC_GPIO12 (GPIO2)
- RTC_GPIO13 (GPIO15)
- RTC_GPIO14 (GPIO13)
- RTC_GPIO15 (GPIO12)
- RTC_GPIO16 (GPIO14)
- RTC_GPIO17 (GPIO27)

I^2C

Der ESP32 hat zwei *I^2C*-Kanäle. Dabei kann jeder Pin als SDA oder SCL fungieren.

Die Arduino IDE hat die Pins für einen Kanal vorbelegt (SDA: GPIO21, SCL: GPIO22).

SPI

Es stehen zwei *SPI*-Kanäle zur Verfügung. Die Voreinstellung sehen Sie in Tabelle 1.4.

SPI	MOSI	MISO	CLK	CS
VSPI	GPIO23	GPIO19	GPIO18	GPIO5
HSPI	GPIO13	GPIO12	GPIO14	GPIO15

Tabelle 1.4 SPI-GPIOs

Strapping Pins

Ein IC kann eine oder mehrere Betriebsarten aufweisen. Festgelegt wird dies über einzelne Konfigurations-Pins. Sofern ein IC genügend Pins hat, können hierfür diese (Eingangs-)Pins auf LOW-Pegel (z. B. GND) bzw. HIGH-Pegel (z. B. 3,3 V) gelegt werden. Soll jedoch die Anzahl der Pins minimiert werden, z. B. für ein kleines Gehäuse, werden zum Konfigurieren (*Strappen*) merkwürdigerweise Output-Pins genutzt. Über diese wird dann direkt nach dem Einschalten oder nach einem Reset als Erstes die gewünschte Konfiguration eingelesen.

Der ESP32 hat folgende Strapping Pins:

- GPIO0
- GPIO2
- GPIO4
- GPIO5 (muss während des Boots HIGH sein)
- GPIO12 (muss während des Boots LOW sein)
- GPIO15 (muss während des Boots HIGH sein)

Diese GPIOs werden benötigt, um den ESP32 in den Bootloader- oder Flash-Modus zu versetzen. Allerdings ist es bei dem ESP32-Dev-KitC V4 wie übrigens auch bei den meisten anderen Entwicklungsboards so, dass das Board die richtigen Zustände für die verschiedenen Modi setzt. Weitere, recht ausführliche Informationen finden Sie in der Dokumentation *ESP32-Boot-Mode-Selection* (*https://github.com/espressif/esptool/wiki/ESP32-Boot-Mode-Selection*).

Probleme beim Hochladen

Es kann jedoch zu Problemen beim Hochladen von neuem Code, beim Flashen des ESP32 mit neuer Firmware oder beim Reset des Boards kommen, wenn diese Pins mit Peripheriegeräten belegt sind. Das kann darauf zurückzuführen sein, dass diese Geräte den ESP32 daran hindern, in den richtigen Modus zu wechseln. Auch hier kann die oben genannte Dokumentation weiterhelfen. Hilfreich ist in solchen Fällen auch, die gesamte Peripherie abzutrennen und schrittweise wieder anzuschließen, um so das störende Element zu isolieren.

Pins HIGH bei Boot

Einige GPIOs wechseln beim Booten oder beim Reset ihren Status zu HIGH oder geben PWM-Signale aus. Für möglicherweise angeschlossene Peripherie bedeutet dies, dass es bei ihr dann zu unerwartetem Verhalten kommen kann. Das betrifft:

- GPIO1
- GPIO3
- GPIO5
- GPIO6 bis GPIO11

  Diese GPIOs sind mit dem ESP32-integrierten SPI-Flash-Memory verbunden und sollten deshalb nicht benutzt werden.
- GPIO14 und GPIO15

Enable (EN)

Enable (EN) ist der Enable-Pin für den 3,3-V-Spannungsregler. Normalerweise liegt er per Widerstand auf HIGH (*pulled up*). Der Spannungsregler wird außer Betrieb gesetzt, wenn der Pin auf LOW (GND) gesetzt wird. Auf diese Weise lässt sich mit einem Button ein Restart des ESP32 erzwingen.

1.5.2 Höhere Eingangsspannung an GPIOs

Bedauerlicherweise ist es so, dass verschiedene im Mikrocontroller-Bereich eingesetzte Sensoren nur mit Spannungen über 3,3 V fehlerfrei betrieben werden können. Dementsprechend liegt ihr Ausgangspegel in der Regel ebenfalls über dieser Marke. Das führt gewissermaßen zu einem Dilemma, denn eine zu hohe Eingangsspannung an einem GPIO kann das Modul zerstören; auf der anderen Seite ist vielleicht gerade dieses Bauteil unverzichtbar. Die Eingangsspannung muss also abgesenkt werden. Hierzu gibt es zwei praktikable Möglichkeiten.

Spannungsteiler

Bei einem *Spannungsteiler* wird die Spannung durch eine Reihenschaltung passiver Elemente aufgeteilt. Einfache Spannungsteiler können bereits mit ein bzw. zwei Widerständen realisiert werden. Für die Absenkung der Eingangsspannung an einem GPIO könnte die Schaltung etwa so wie in Abbildung 1.7 aussehen.

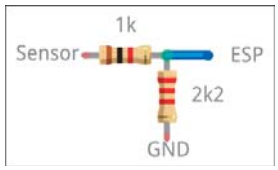


Abbildung 1.7 Sensor-Spannungsteiler mit Widerständen

Eine weitere Möglichkeit ist die Spannungsbegrenzung des Eingangssignals mit einer Zener-Diode (siehe Abbildung 1.8).

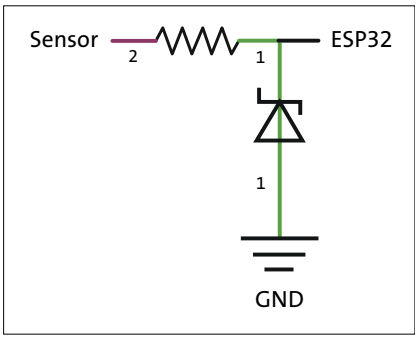


Abbildung 1.8 Sensor-Spannungsteiler mit Zener-Diode

Pegelwandler (Level Shifter)

Eleganter, aber auch etwas aufwendiger sind integrierte Pegelwandler (Pegelumsetzer, *Level Shifter*). Pegelwandler können unidirektional arbeiten, also nur in eine Richtung (z. B. bei Sensoren), oder bidirektional, also in zwei Richtungen für Bussysteme. Entsprechende Module sind auf dem Markt verfügbar (siehe Abbildung 1.9).

Abbildung 1.9 Pegelwandler

Die Anschlüsse sind:

- GND: für den gemeinsamen Ground-Anschluss
- HV: High-Voltage, das hohe VCC, z. B. 5 V
- LV: Low-Voltage, das niedrige VCC, z. B. 3,3 V
- HV1–HV4: die Ein-/Ausgänge Hochvolt
- LV1–LV4: die Ein-/Ausgänge Niedrigvolt

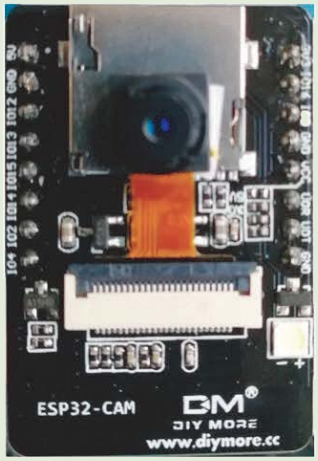
Der Einsatz von Pegelwandlern kann aber auch mit ein paar Nachteilen einhergehen:

- Überlastung einer oder beider Seiten bis hin zur Zerstörung
- inkompatible Logikpegel und daraus resultierendes Nichtfunktionieren der Schaltung oder, noch schlimmer, sporadische Fehlfunktionen
- Verzögerungen der Signale durch die Pegelwandlung und daraus resultierende maximale Signalfrequenzen

Das Board ESP32-CAM AI-Thinker

Das *ESP32-CAM AI-Thinker* ist ein sehr kleines ESP-Kameraboard mit dem ESP32-S-Modul (siehe Abbildung 1.10). Es verfügt über 420 KB SRAM und auf dem Board sind zusätzliche 4 MB PSRAM verbaut.

Neben der Kamera (meist einer *OV2640-Kamera* mit 2 Megapixel, die über einen FPC-Connector angeschlossen wird) und mehreren GPIOs für Peripheriegeräte verfügt das Board auf der Kameraseite auch über einen MicroSD-Kartensteckplatz zum Speichern von aufgenommenen Bildern oder Dateien für eine spätere Verarbeitung. Ein Blitzlicht, eine On-Board-LED und ein IPEX-Anschluss für eine externe Antenne runden das Bauteil ab.



Das Board ist nicht allzu teuer und recht leistungsfähig; unter optimalen Bedingungen ist sogar eine automatisierte Gesichtserkennung möglich. Entsprechende Software hat die Arduino IDE im Gepäck.

Dennoch bieten meiner Meinung nach komplette Geräte, z. B. eine Tapo-Kamera, ein besseres Preis-Leistungs-Verhältnis, da in sie meist Servos zum Schwenken und Neigen der Kamera integriert sind und auch das Kameramodul besser auf das Gerät abgestimmt ist.

1.6 Das ESP32-Dev-KitC V4 – Erstinbetriebnahme am PC

Wenden wir uns nun der Praxis zu: Verbinden Sie das ESP32-Dev-KitC V4 über ein Micro-USB-B-Kabel erstmalig mit Ihrem PC. Die Betriebszustands-LED sollte einige Male deutlich rot blinken, bei anderen Boards kann sie dauerhaft leuchten. Das bedeutet, es liegt Spannung an, und der On-Board-Spannungsregler arbeitet einwandfrei.

1.6.1 Windows

Das Board sollte von Windows selbstständig erkannt werden. Dies lässt sich einfach im Geräte-Manager überprüfen (Windows • Systemsteuerung • Hardware und Sound • Geräte-Manager, siehe Abbildung 1.10).

Hier ist das Modul an Port COM4 angeschlossen. Windows hat das neue USB-Gerät erkannt (hier die USB-to-UART-Bridge CP210x, in manchen Boards ist auch ein CH340-Chipsatz verbaut) und den nötigen Treiber installiert.

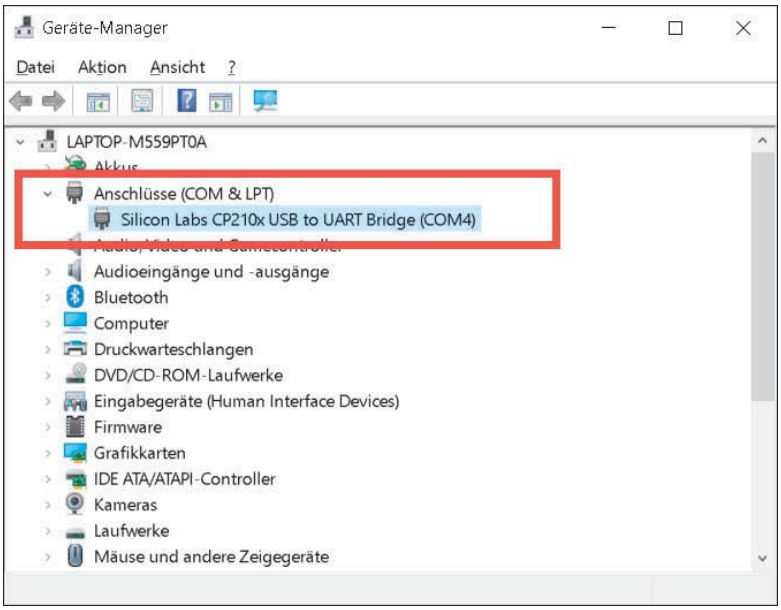


Abbildung 1.10 ESP32-Dev-KitC V4 im Geräte-Manager unter Windows

Sollte das Modul nicht identifiziert worden sein, kann das an fehlenden Datenleitungen im USB-Kabel liegen. Können Sie dies ausschließen bzw. hat das Betriebssystem einen unzutreffenden Treiber eingerichtet (z. B. bei Windows 11), dann hilft eventuell eine manuelle Installation des Treibers für die CP2102-Bridge (siehe Bauteil 8 des ESP32-Development-Kit V4).

Gehen Sie dann wie folgt vor:

1. Holen Sie sich das Treiberpaket von der Download-Seite des Herstellers – z. B. für die CP2102-Bridge von Silicon Labs von *https://www.silabs.com/developers/usb-to-uart-bridge-vcp-drivers?tab=downloads* und dort das Paket *CP210x Universal Windows Driver*.
2. Entpacken Sie die Datei *CP210x_Universal_Windows_Driver.zip*.
3. In dem entpackten Ordner befindet sich eine Datei *silabser.inf*. Gehen Sie mit einem rechten Mausklick auf die Datei, und wählen Sie aus der Dropdown-Liste INSTALLIEREN.

Je nach Ihrer Konfiguration müssen Sie noch eine Sicherheitsabfrage Ihrer Firewall bejahen. Eine erfolgreiche Installation erkennen Sie an der Meldung »The Operation Completed successfully«.

1.6.2 Linux

Linux erkennt üblicherweise das Board eigenständig – erfolgreich getestet wurde dies mit Ubuntu 22.04. Die Überprüfung erfolgt in einem Terminalfenster mit dem Befehl `lsusb` (siehe Abbildung 1.11).

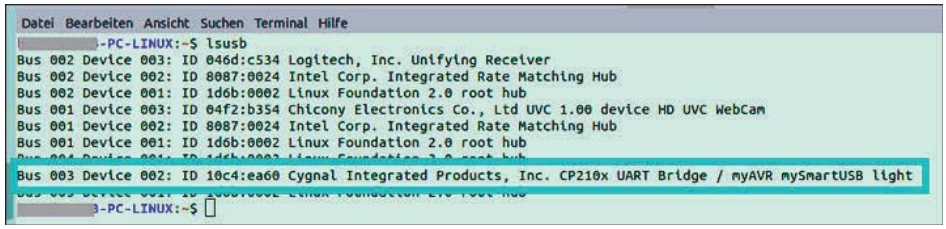


Abbildung 1.11 ESP32-Development-Kit V4 an Linux (Ubuntu)

Zusammenfassung

An dieser Stelle beende ich die Beschreibung der Hardware rund um den Chip ESP32, die von Espressif gefertigten Module und erhältlichen Boards. Das ESP32-Dev-KitC V4 hat sich als äußerst vielfältiger Baustein und dennoch recht einfach zu handhabende Komponente präsentiert, die dazu einlädt, es für eigene Projekte zu nutzen. Deshalb ist dieser Baustein unsere Basis für die folgenden Projekte.

3.3.7 Bauteile ändern und selbst erstellen

Nicht immer liegt ein Bauteil generell oder in der gewünschten Art und Weise in Form einer *.fzpz*-Datei vor. Dann führt kein Weg daran vorbei, ein vorhandenes Part zu ändern oder ein Part gänzlich neu zu erstellen. Zumindest die letztere Variante ist überaus umständlich, fehleranfällig und zeitaufwendig. Unter *https://www.heise.de/make/artikel/Neue-Bauteile-fuer-Fritzing-erstellen-so-geht-s-4072734.html* gibt es aber eine gute und ausführliche Anleitung.

Die meisten der in diesem Buch verwendeten Parts sind bereits im Auslieferungszustand von Fritzing enthalten. Weitere Teile finden Sie bei den Materialien zum Buch und in Abschnitt A.2.

3.4 Löttechnik

Löten ist ein thermisches Verfahren, um Werkstoffe zusammenzufügen. Die Verbindung wird durch Schmelzen eines Lots hergestellt. Löten ist eine sehr alte Technik, deren Ursprünge bereit mehr als 5.000 Jahren zurückliegen. Der Unterschied zum Schweißen besteht unter anderem darin, dass beim Schweißen wesentlich höhere Temperaturen (zum Teil über 3.000 °C) nötig sind und die Nahtstellen der zu verbindenden Materialien verflüssigt werden. Schweißverbindungen sind dadurch wesentlich stabiler und belastbarer.

Beim Löten wird wiederum zwischen *Hart-* und *Weichlöten* unterschieden. Hartlöten erfolgt mit Temperaturen oberhalb von 450 °C. Zum Einsatz kommen Lötbrenner. Die Temperatureinwirkung ist großflächig, sodass das Lot gut verlaufen kann. Ein verbreitetes Einsatzgebiet ist die Verbindung von Rohren für die Wasser- und Gasinstallation.

Das Weichlöten wird auch industriell in erster Linie in der Elektrotechnik angewendet, um Bauteile elektrisch leitend zu verbinden. Die Löttemperatur bewegt sich zum Schutz vor thermischer Überlastung der Bauteile in der Regel zwischen 180 und 250 °C. Auch erfolgt die Temperatureinwirkung nur punktuell. Maßnahmen des Gesundheits- oder Arbeitsschutzes sind nur hinsichtlich der entstehenden Dämpfe durch schmelzendes Lot oder verdampfendes Flussmittel erforderlich. Achten Sie auch im privaten Bereich darauf!

3.4.1 Bauteile auf Platinen löten

Der häufigste Anwendungsfall des Lötens ist vermutlich die Bestückung von Platinen. Dabei wird vorzugsweise Elektroniklot mit einer Flussmittelseele verwendet. Die Lötstelle selbst muss sauber und frei von Oxidschichten oder Flussmittelresten sein.

Erhitzen Sie zum Löten die Lötstelle zunächst mit der Lötspitze. Führen Sie das Lötzinn zu, wenn eine genügend hohe Temperatur erreicht ist. Stimmen die Bedingungen, verflüssigt sich das Lötzinn und umfließt Lötstelle und Bauteil. Beenden Sie die Wärmezufuhr, sodass die Lötstelle erkalten kann. Der Lötvorgang sollte zum Schutz vor einer thermischen Überlastung der Bauteile nicht mehr als fünf Sekunden in Anspruch nehmen. Im Ergebnis sollte das Lot das Anschlussbeinchen des Bauteils kegelförmig umschließen und mit der Platine verbinden (siehe Abbildung 3.24).

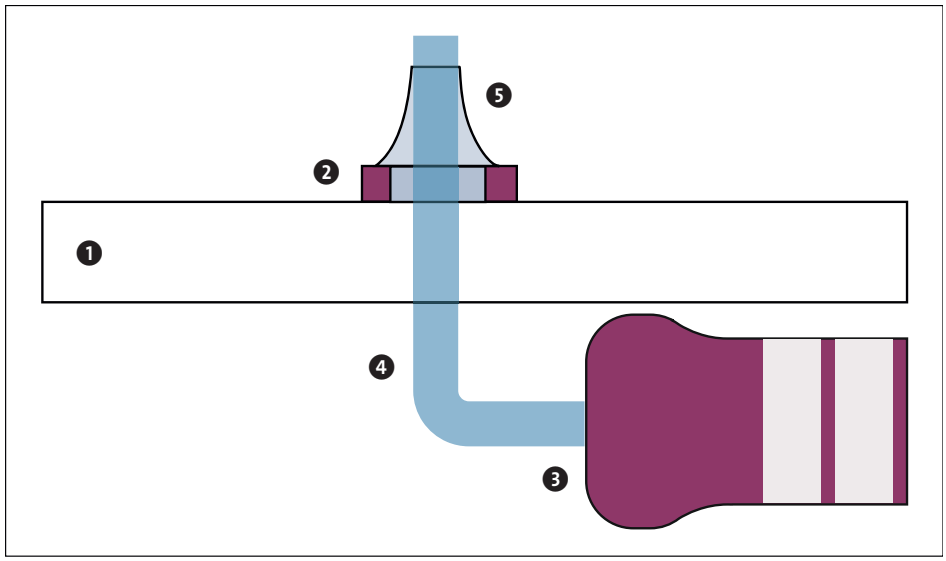


Abbildung 3.24 Optimale Lötstelle: Platine ❶, Leiterbahn ❷, Bauteil ❸, Beinchen ❹, Lötstelle ❺

Auch Kabel können miteinander verlötet werden. Isolieren Sie hierzu die Kabelenden ab und verzinnen Sie sie mit etwas Lötzinn. Anschließend verdrillen Sie die Enden leicht und verlöten sie. Zum Schutz gegen Kurzschlüsse und als mechanische Stütze ziehen Sie am besten noch einen Schrumpfschlauch über die blanke Lötstelle.

Allerdings können selbst bei scheinbar simplen Arbeiten offensichtliche und weniger offensichtliche Fehler unterlaufen:

- **Das Lot fließt nicht**: Der Grund dafür kann eine zu geringe Löttemperatur sein: Wählen Sie dann eine höhere Temperatur. Auch kann die Lötspitze korrodiert oder nicht sauber sein. Tauschen Sie die Lötspitze in diesem Fall aus oder säubern Sie sie.
- **Die Anschlussbeinchen des Bauteils sind nicht kegelförmig umschlossen**: Solche fehlerhaften Lötstellenausprägungen entstehen, wenn das Lot nicht richtig zum Fließen kommt, weil die Löttemperatur des Lötkolbens zu gering ist oder die

Lötstelle nicht lang genug erwärmt wurde (kraterförmige Lötstelle). Oder aber das Lötzinn war lediglich auf der Lötspitze und ist nur teilweise auf die Lötstelle aufgebracht worden (ringförmige Lötstelle, Lötzinnperlen). In beiden Fällen ist keine oder nur eine unzureichende Verbindung zwischen Bauteil und Platine zustande gekommen. Dies wird als *kalte Lötstelle* bezeichnet. Wiederholen Sie den Lötvorgang.

- **Das Bauteil wurde in der Abkühlphase bewegt**: Auch hier besteht die Gefahr, dass eine kalte Lötstelle entstanden ist.
- **Das Bauteil oder die Platine wurde überhitzt**: Dass das Bauteil oder die Platine beim Löten überhitzt wurde, lässt sich manchmal an Leiterbahnen erkennen, die sich vom Trägermaterial abgelöst haben. Wesentlich schwieriger ist es, die Auswirkungen bei den verlöteten Bauteilen zu verorten. Gerade Halbleiter wie Dioden, Transistoren und insbesondere ICs reagieren sehr empfindlich auf Überhitzung und quittieren dauerhaft und endgültig ihren Dienst. Hier hilft nur ein Austausch der Komponenten.

Header- oder GPIO-Pins nicht löten! [!]

Löten Sie nie auf Header- oder GPIO-Pins. Nach dem Trennen einer Lötverbindung bleiben immer Reste von Lötzinn am Bauteil. Sie erschweren es oder machen es sogar unmöglich, den Pin wieder mit einer üblichen Steckverbindung zu nutzen.

Löten ist Handwerk. Das allermeiste Handwerk lässt sich zumindest in seinen Grundfunktionen erlernen. Nach den ersten Versuchen mit vielleicht wenig überzeugenden Resultaten werden Sie durch Üben und Wiederholen schrittweise bessere Ergebnisse erzielen. Auch hier gilt: Lassen Sie sich nicht durch Misserfolge entmutigen, sondern bleiben Sie weiter am Ball.

3.4.2 Bauteile entlöten

Hin und wieder müssen Bauteile wieder von der Platine genommen oder fehlerhafte Lötverbindungen, die einen Kurzschluss bewirken, gelöst werden. Dafür gibt es zwei Hilfsmittel:

- *Entlötlitze* besteht aus feiner, mit Flussmittel getränkter Kupferlitze. Die Litze wird auf die zu entlötende Stelle aufgelegt und mit dem Lötkolben erhitzt. Durch die Kapillarwirkung zieht sich das Lötzinn in die Litze. Bevorzugtes Anwendungsgebiet sind kleine Lötstellen (ICs, SMDs). Allerdings ist etwas Vorsicht hinsichtlich der Überhitzungsgefahr geboten.
- Die *Entlötpumpe* arbeitet nach dem Vakuumprinzip. Sie wird durch eine Feder vorgespannt, die durch Knopfdruck ausgelöst wird und somit kurzzeitig einen Unter-

druck erzeugt. Der Unterdruck entfernt das verflüssigte Lötzinn. Auch hier besteht die Gefahr der thermischen Überlastung. Den Vorgang können Sie unterstützen, indem Sie auf die Lötstelle etwas Flussmittel (Lötfett, Lötwasser, Lötöl) aufbringen.

3.4.3 SMD löten

In der industriellen Fertigung werden viele Bauteile als *SMD* (*Surface Mounted Device*; dt. »oberflächenmontiertes Bauteil«) eingesetzt. SMD-Bauteile haben keine Anschlussdrähte und werden deshalb direkt auf die Platine aufgesetzt und verlötet. Diese Bauform erlaubt im automatisierten Bestückungsprozess einer Platine eine schnelle und auch platzsparende Fertigung. Für die heimische Realisierung von Schaltungen stellen sie jedoch eine Herausforderung dar, sodass selbst geübte Maker nur höchst ungern diese Bauteile verwenden.

Irgendwann wird aber der Punkt erreicht sein, an dem eine Komponente nur als SMD verfügbar ist. Dann führt kein Weg am ungeliebten Handlöten vorbei. Es ist zugegebenermaßen schwierig, doch nicht unmöglich. Und es erfordert Zeit, Geduld und Sorgfalt.

Spätestens jetzt benötigen Sie einen Lötkolben mit einstellbarer Temperatur. Der Lötkolben sollte leicht sein, denn je schwerer das Gerät ist, desto leichter zittert die Hand. Die Lötspitze muss bei dieser Arbeit kleiner als üblich sein. Sie sollte nur so dick sein, wie es für die anstehenden Lötarbeiten gerade noch vertretbar ist. Es besteht also ein Dilemma, denn je kleiner die Lötspitze ist, desto schlechter ist aufgrund der geringeren Fläche auch die Wärmeübertragung. Maker kommen an dieser Stelle häufig mit einer meißelförmigen Lötspitze besser zurecht als mit einer punktförmigen (siehe Abbildung 3.25).

Abbildung 3.25 Verschiedene Lötspitzen

Der Lötdraht sollte nicht stärker als 0,5 mm sein und idealerweise eine Flussmittelseele haben.

Ab jetzt zahlt sich der Erwerb von hochwertigerem Werkzeug aus, denn eine gute Pinzette und eine gute Lupe (unter Umständen mit zusätzlicher Beleuchtung und Dritter Hand) sind nahezu unverzichtbar. Fehlstellen oder Leitungsbrücken lassen sich vorzugsweise mit schmaler Entlötlitze mit einer Breite von ca. 1,5 mm beheben.

3.4.4 Ein ESP32-Modul löten

Damit steht das theoretische Gerüst, um auch den ESP32 in der SMD-Variante auf eine entwickelte oder erworbene Platine zu löten. Das unmittelbare Löten auf eine Lochstreifen- oder Lochrasterplatine scheidet aus, da hier der Rasterabstand nicht dem ESP32-WROOM-Modul entspricht.

Die Grundlage der folgenden Beschreibung ist eine ESP32-WROOM-Adapterkarte, wie sie auf den einschlägigen Plattformen angeboten wird. Im Lieferumfang sind meist zwei Microschalter und entsprechende Header-Pin-Leisten enthalten (siehe Abbildung 3.26 und Abbildung 3.27).

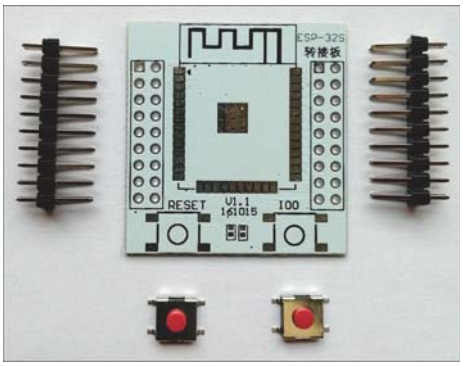


Abbildung 3.26 ESP32-WROOM-Adapterkarte – Vorderseite, Headerleiste, Taster

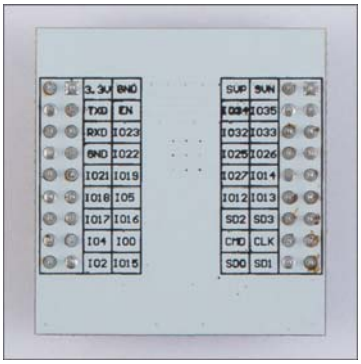


Abbildung 3.27 ESP32-WROOM-Adapterkarte – Rückseite

Pull-up-Widerstand

Im Handel sind verschiedene Adapter-Breakout-Boards für den ESP-WROOM-32 erhältlich. Bei den allermeisten Adapterboards (wie auch bei dem in diesem Abschnitt verwendeten) müssen Sie den EN-Pin und den 3V3-Pin mit einem 4,7-kΩ- bis 20-kΩ-Pull-up-Widerstand verbinden. Empfohlen wird ein Wert von 12 kΩ.

Das ESP32-WROOM-Modul auf der Adapterkarte platzieren und fixieren

Legen Sie das Modul auf die Vorderseite der Adapterplatte und schieben Sie es so lange, bis die Kontakte des Moduls passgenau mit den Kontakten der Platte überein-

stimmen. Das ist absolute Feinarbeit und dauert dementsprechend. Eine Pinzette hilft dabei, kleinste Korrekturen vorzunehmen.

Als Nächstes fixieren Sie das Modul mit einem Klebestreifen auf der Adapterplatte an einer oberen Ecke. Kleben Sie den Klebestreifen zuvor auf das Modul und drücken Sie ihn dann mit der Pinzette und nicht mit dem Finger fest. Häufig verschiebt sich das Modul dabei aber geringfügig; justieren Sie mit einer Pinzette etwas nach. Bringen Sie jetzt einen zweiten Klebestreifen auf, prüfen Sie erneut die richtige Platzierung und justieren Sie gegebenenfalls nach. Als Letztes bereiten Sie das Modul mit einem dritten Klebestreifen endgültig für das Löten vor. Überprüfen Sie zum Schluss sicherheitshalber nochmals die korrekte Lage des Moduls, vielleicht sogar unter Zuhilfenahme einer Lupe (siehe Abbildung 3.28).

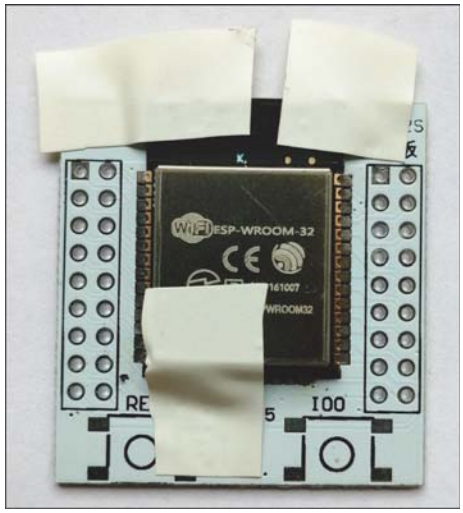


Abbildung 3.28 ESP32-WROOM, platziert auf der Adapterkarte

Dieser Schritt gehört fast zu den zeitaufwendigsten im gesamten Prozess, denn er ist Schlüssel und Grundlage für ein gutes Ergebnis.

Das ESP32-WROOM-Modul auf der Adapterkarte mit einem Kontakt fixieren

Nun beginnt der Lötprozess. Bringen Sie hierzu an einer Ecke, falls erforderlich, etwas Lötwasser auf und verlöten Sie einen Kontakt des Moduls fest mit der Adapterplatte. Es ist nicht schlimm, sollte die Lötstelle nicht 100%ig ausgebildet sein. Es kommt eher darauf an, dass eine erste feste Verbindung mit der Adapterplatte hergestellt wird (siehe Abbildung 3.29, Kontakt links unten).

Überprüfen Sie wie bereits zuvor die korrekte Ausrichtung nochmals genau, da jetzt noch Korrekturen möglich sind.

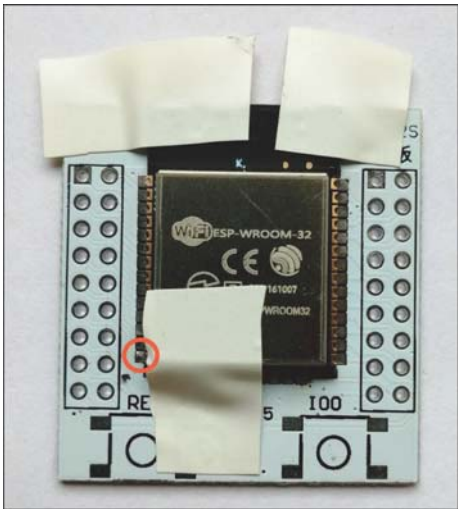


Abbildung 3.29 ESP32-WROOM, fest verlötet mit einem Kontakt

Das ESP32-WROOM-Modul auf der Adapterkarte mit einem zweiten Kontakt fixieren

Als Nächstes löten Sie das Modul an der gegenüberliegenden Ecke mit einem weiteren Kontakt auf. Dies ist dann so ziemlich der letzte Moment für Nachjustierungen (siehe Abbildung 3.30, Kontakt rechts oben).

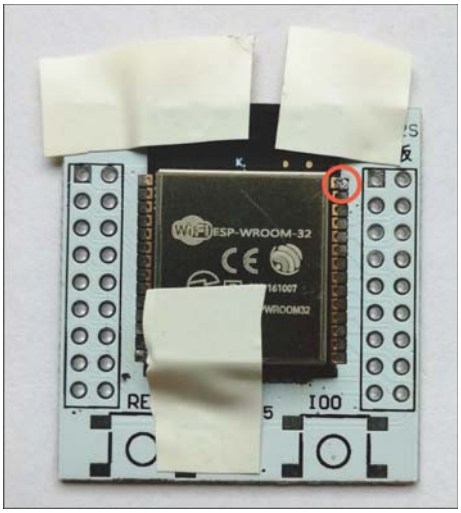


Abbildung 3.30 ESP32-WROOM, fest verlötet mit zwei Kontakten

Das ESP32-WROOM-Modul endgültig verlöten

Entfernen Sie nun die Klebestreifen, verlöten Sie alle Kontakte Reihe für Reihe fest und bessern Sie unzureichende Fixierungspunkte nach. Setzen Sie je nach Erforder-

nis Lötwasser ein. Oft werden die Augen durch das angestrengte Blicken auf kleinste Punkte sehr beansprucht. Legen Sie nach jeder Reihe eine Pause ein oder führen Sie die Lötarbeiten generell unter einer Lupe (z. B. der Dritten Hand) durch. Essenziell ist unabhängig davon eine sehr gute Ausleuchtung des punktuellen Arbeitsgebiets. Ein mögliches Verrutschen der kleinen Adapterplatte während des Lötens verhindern Klebestreifen, die die Platte auf der Arbeitsfläche fixieren. Flussmittelreste sollten mit Alkohol oder besser Isopropanol entfernt werden.

Lötverbindungen auf dem ESP32-WROOM-Modul überprüfen

Sind alle Kontakte verlötet, geht es an die Überprüfung der Lötverbindungen. Mit einem Multimeter erfolgt der Prüfvorgang in zweierlei Richtung:

- Haben sich unerwünschte Kontaktbrücken ergeben?
- Ist eine elektrische Verbindung zustande gekommen?

Die Taster verlöten

Benetzen Sie für das Verlöten der Taster zunächst eine Kontaktfläche auf der Adapterkarte mit Lot. Bringen Sie dann den Taster auf und richten Sie ihn aus. Stimmt die Position, verlöten Sie die restlichen drei Kontakte.

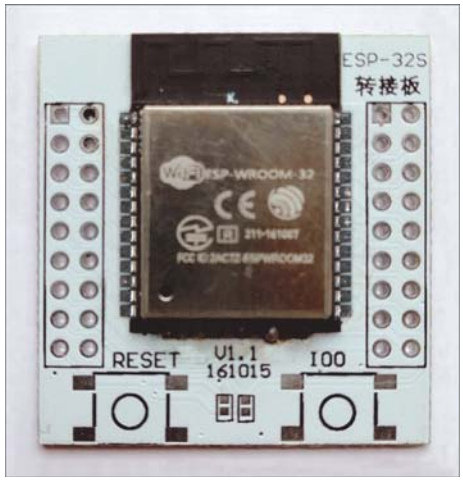


Abbildung 3.31 ESP32-WROOM – alle Kontakte sind fest verlötet.

Die Header-Pin-Leisten verlöten

Als Letztes fehlen noch die Header-Pin-Leisten. Sind sie montiert, kann die Adapterkarte ohne zusätzliche Koppelelemente (Jumperkabel) nicht mehr auf eine Steckplatte gebracht werden (siehe Abbildung 3.32).

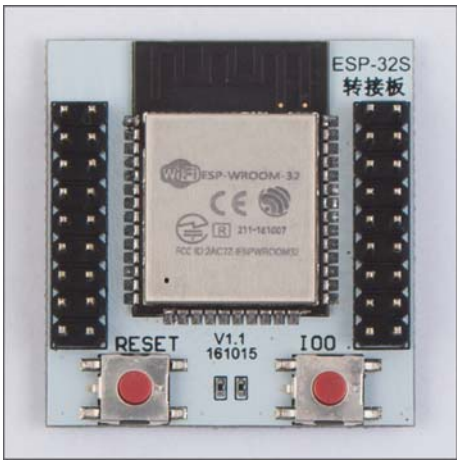


Abbildung 3.32 Die ESP32-WROOM-Adapterkarte ist nun fertig verlötet.

Das Pinout der Adapterplatte hat dann die Belegung von Abbildung 3.33.

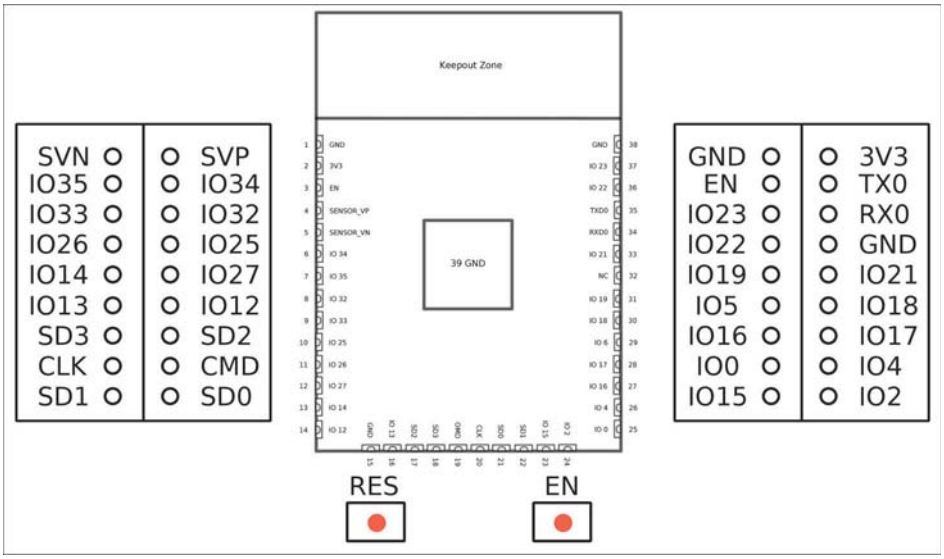


Abbildung 3.33 Pinout der ESP32-WROOM-Adapterplatte

Kapitel 10
Projektideen

Die bisherigen Kapitel haben das Fundament für das Arbeiten mit dem ESP32 gelegt. Dieses Kapitel baut auf dieser Basis auf, wandelt ab und ergänzt das Spektrum.

Die folgenden Abschnitte befassen sich mit ein paar Projektideen, die auch längeren Code enthalten – er kann tatsächlich ziemlich umfangreich werden und soll Ihnen als Vorlage für eigene Projekte dienen. Sie müssen aber natürlich nicht seitenweise Code abtippen! Alle Codebeispiele finden Sie unter *https://www.rheinwerk-verlag.de/5936* zum Download.

Die Ideen, die Sie in diesem Kapitel finden, bauen auf dem auf, was Sie bisher kennengelernt haben, beinhalten aber auch die eine oder andere Erweiterung bzw. Neuerung. Die Vorschläge verstehen sich als Anregung. Wandeln Sie sie ab, schneiden Sie sie auf Ihre persönlichen Erfordernisse zu, lassen Sie sich inspirieren und gehen Sie Ihren Weg.

Ich wünsche Ihnen viel Freude und Erfolg bei der Verwirklichung Ihrer Projekte!

10.1 The Evil Dice

Die kleine Sammlung von Projektideen soll mit einem Beispiel beginnen, das gut und gerne an einem verregneten Samstag umgesetzt werden kann: dem »Evil Dice«, einem smarten Würfel.

Was der smarte Würfel leisten soll

Für eine reine Würfelschaltung ist der ESP32 eigentlich überdimensioniert, denn ein rollender elektronischer Würfel lässt sich bereits mit deutlich kleineren Mikroprozessoren bauen, etwa mit dem ATTINY45.

Etwas Pfiff bekommt die Angelegenheit über verborgene Sonderfunktionen. So soll der Würfel alternativ über zwei nebeneinanderliegende Touch-Sensoren aktiviert werden können. Wird einer der beiden Sensoren berührt, ist das Würfelergebnis eine zufällige Zahl zwischen 1 und 6.

Wird aber der linke Sensor nur kurz berührt und anschließend mit einer Wischbewegung der rechte Sensor, ist das Ergebnis eine 1 bzw. mit einer Wischbewegung von rechts nach links eine 6. Eine solche Funktion kann ja bei spannenden Brettspielabenden im Familienkreis ganz nützlich sein ...

Die Schaltung

Für erste Gehversuche bietet es sich an, die Schaltung auf dem Steckbrett aufzubauen (siehe Abbildung 10.1).

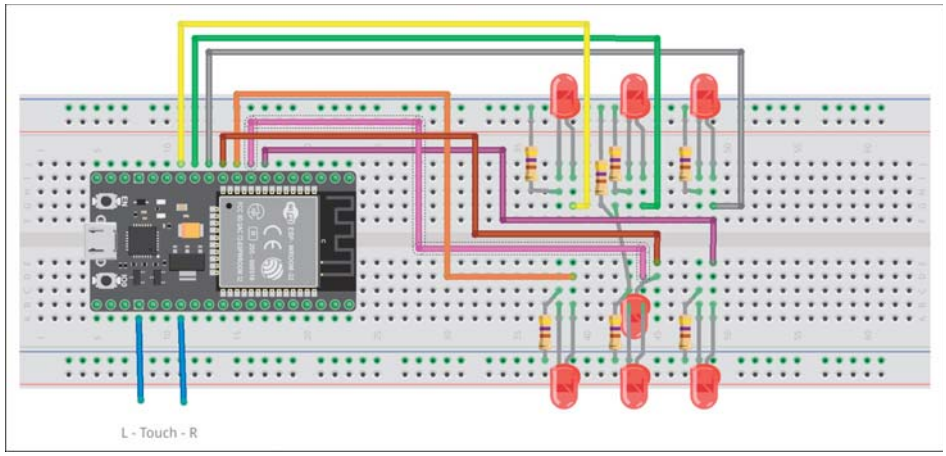


Abbildung 10.1 Schaltung für den smarten Würfel

Die Steckbrettschaltung können Sie dann in ein selbst gebasteltes Würfelgehäuse überführen.

Der Sketch

Der Sketch aus Listing 10.1 ist ziemlich umfangreich, die Komplexität ist aber noch überschaubar.

```
#define TOUTCH_PIN1 T3                    // ESP32-Pin D0
#define TOUTCH_PIN2 T0                    // ESP32-Pin D4
#define LED_PIN 2
#define LED_PIN1 12
#define LED_PIN2 14
#define LED_PIN3 27
#define LED_PIN4 25
#define LED_PIN5 33
#define LED_PIN6 32
#define LED_PIN7 26
```

```
unsigned long zeit1, zeit1Diff;         // Dauer Touch 1 berührt
unsigned long zeit2, zeit2Diff;         // Dauer Touch 2 berührt
unsigned long zeit3, zeit3Diff;         // Dauer Touch für Würfeln
int touchValue1, touchValue2;
int maxS = 30;

// LEDs ausschalten
void setLedLow() {
  digitalWrite (LED_PIN1, LOW);
  digitalWrite (LED_PIN2, LOW);
  digitalWrite (LED_PIN3, LOW);
  digitalWrite (LED_PIN4, LOW);
  digitalWrite (LED_PIN5, LOW);
  digitalWrite (LED_PIN6, LOW);
  digitalWrite (LED_PIN7, LOW);
  return;
}

// Ausgeben der gewürfelten Zahl
void displayZahl (int zahl) {
  switch (zahl) {
    case 1: digitalWrite (LED_PIN7, HIGH);
            break;
    case 2: digitalWrite (LED_PIN2, HIGH);
            digitalWrite (LED_PIN5, HIGH);
            break;
    case 3: digitalWrite (LED_PIN1, HIGH);
            digitalWrite (LED_PIN7, HIGH);
            digitalWrite (LED_PIN6, HIGH);
            break;
    case 4: digitalWrite (LED_PIN1, HIGH);
            digitalWrite (LED_PIN3, HIGH);
            digitalWrite (LED_PIN4, HIGH);
            digitalWrite (LED_PIN6, HIGH);
            break;
    case 5: digitalWrite (LED_PIN1, HIGH);
            digitalWrite (LED_PIN3, HIGH);
            digitalWrite (LED_PIN7, HIGH);
            digitalWrite (LED_PIN4, HIGH);
            digitalWrite (LED_PIN6, HIGH);
            break;
    case 6: digitalWrite (LED_PIN1, HIGH);
            digitalWrite (LED_PIN2, HIGH);
```

```
            digitalWrite (LED_PIN3, HIGH);
            digitalWrite (LED_PIN4, HIGH);
            digitalWrite (LED_PIN5, HIGH);
            digitalWrite (LED_PIN6, HIGH);
            break;
  }
  return;
}

void wuerfeln (int art) {
  int zahl;
  for (int i = 0; i < 30; i++) {        // Rollen des Würfels simulieren
    zahl = random (1, 6);
    displayZahl(zahl);
    delay(15 * i);                      // verlangsamen
    setLedLow();
  }
  if (art == 1)      zahl = 1;
  else if (art == 6) zahl = 6;
  displayZahl(zahl);
  delay (1000);
  setLedLow();
  return;
}

void setup() {
  pinMode(LED_PIN, OUTPUT);
  pinMode(LED_PIN1, OUTPUT);
  pinMode(LED_PIN2, OUTPUT);
  pinMode(LED_PIN3, OUTPUT);
  pinMode(LED_PIN4, OUTPUT);
  pinMode(LED_PIN5, OUTPUT);
  pinMode(LED_PIN6, OUTPUT);
  pinMode(LED_PIN7, OUTPUT);
  digitalWrite (LED_PIN, LOW);
  setLedLow();
  zeit1 = micros();
  zeit2 = micros();
  zeit3 = micros();
}
```

```
void loop() {
  touchValue1 = touchRead(TOUTCH_PIN1);
  touchValue2 = touchRead(TOUTCH_PIN2);

  if (touchValue1 < maxS ||
      touchValue2 < maxS) {
    if (touchValue1 > 5 && touchValue1 < maxS) {
      zeit1Diff = micros() - zeit1;
    }
    if (touchValue2 > 5 && touchValue2 < maxS) {
      zeit2Diff = micros() - zeit2;
    }

    int antT1, antT2;
    zeit3Diff = micros() - zeit3;
    if (zeit3Diff > 500000) {
      antT1 = zeit1Diff * 100 / (zeit1Diff+zeit2Diff);  // %-Anteil
      antT2 = zeit2Diff * 100 / (zeit1Diff+zeit2Diff);  // %-Anteil
      if (antT1 > 75 || antT2 > 75) {   // nur ein Touch angetippt
        wuerfeln (0);
      } else
      if (antT1 < 35) {                 // zuerst Tipp T1
        wuerfeln (1);
      } else
      if (antT2 < 35) {                 // zuerst Tipp T2
        wuerfeln (6);
      } else {                          // Tipp nicht eindeutig
        wuerfeln (0);
      }
      delay (1000);
      zeit1 = micros();
      zeit2 = micros();
      zeit3 = micros();
      zeit1Diff = 0;
      zeit2Diff = 0;
    }
    digitalWrite (LED_PIN, HIGH);
  } else {
    zeit1 = micros();
    zeit2 = micros();
    zeit3 = micros();
    zeit1Diff = 0;
    zeit2Diff = 0;
```

```
    digitalWrite (LED_PIN, LOW);
  }
  delay(10);
}
```

Listing 10.1 Sketch für »The Evil Dice«

Die meisten Programmbestandteile sind bereits in ähnlicher Form behandelt worden, sodass der Code keine dramatischen Neuerungen aufweist. Die eigentliche Schwierigkeit besteht darin, die Parameter der eigenen »Wischfertigkeit« anzupassen.

Die ersten Programmzeilen sind wie immer den notwendigen `#includes`, `#defines` und der Definition von globalen Variablen vorbehalten.

Für den Pegelwechsel aller LEDs auf `LOW` ist eine eigene Funktion `setLedLow()` vorgesehen, da dies von zwei Stellen des Programms nötig ist.

In der Funktion `displayZahl (int zahl)` werden die LEDs entsprechend dem Parameter `zahl`, der den Würfelwert enthält, über eine `switch`-Anweisung eingeschaltet.

Drei oder sieben Stränge?

Anstatt sieben Stränge für die LED-Steuerung zu verwenden, können Sie die gleiche Aufgabe auch mithilfe von nur drei Leitungen bewerkstelligen. Der Stromverbrauch kann dann aber in Abhängigkeit von den verwendeten LEDs die maximal zulässige GPIO-Belastung übersteigen, sodass in der Folge eine Transistorschaltung (siehe Abschnitt 7.2.1) nötig wäre.

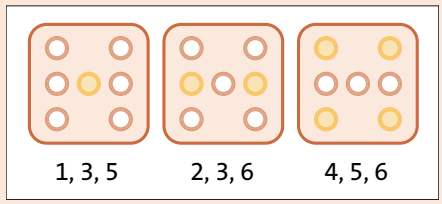


Abbildung 10.2 Smarter Würfel – Steuerung mit drei Leitungen

Die Funktion `wuerfeln (int art)` ermittelt das Ergebnis des Würfelvorgangs. Der Parameter `art` bestimmt, ob es sich um normales Würfeln oder um einen »Sondervorgang« handelt. Sie können die Anzahl der Rollvorgänge (hier 30) und die Verzögerungszeit beim Auslaufen des Würfels (hier `delay(15 * i)`) Ihren Anforderungen entsprechend ändern.

In der Hauptschleife befindet sich die eigentliche Verarbeitungslogik. Zu Beginn werden die Touch-Sensoren ausgelesen. In Abhängigkeit davon, ob der in der Variablen `maxS` festgelegte Schwellenwert für eine Berührung unterschritten ist, erfolgt dann die weitere Auswertung.

Die Abfrage `if (zeit3Diff > 500000)` überprüft, ob der Touch-Vorgang insgesamt lange genug gedauert hat, und zwar auf Basis von Mikrosekunden (1.000 Mikrosekunden = 1 Millisekunde, 1.000.000 Mikrosekunden = 1 Sekunde – hier also 0,5 Sekunden).

Bei einem positiven Ergebnis berechnet der Sketch im Anschluss den jeweiligen prozentualen Anteil der Berührung von Touch 1 und Touch 2 an der gesamten Zeit des Touch-Vorgangs. Dieser wird als Nächstes ausgewertet und die Funktion `wuerfeln()` wird entsprechend parametrisiert aufgerufen.

Der Sketch wird ausgeführt

Wie der fertige Würfel genau aussieht, hängt natürlich von Ihrem Bastelgeschick ab. Dass die LEDs leuchten, wie sie sollen, zeigt Abbildung 10.3.

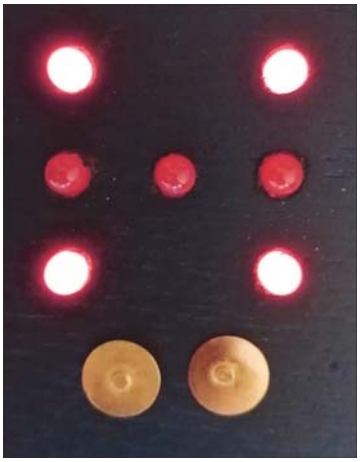


Abbildung 10.3 Ergebnis des smarten Würfels

Damit der Würfel auch seine Evil-Funktionen wunschgemäß ausführt, braucht es allerdings etwas Übung – die Funktion soll ja nicht zu einfach auszulösen sein. Passen Sie gegebenenfalls die Parameter an und probieren Sie unterschiedliche Werte aus.

10.2 Die Maker-Uhr

Maker realisieren gern eigene Lösungen – auch für gängige, alltägliche Güter. So liegt es nahe, diesen Ansatz auch auf eine individuelle Zeitanzeige, also eine unkonventionelle Digitaluhr, anzuwenden. In diesem Sinne soll der folgende Abschnitt eine Uhr präsentieren, die die Zeit in binärer Form anzeigt. Bezugsquelle für die aktuelle Zeit soll das Web sein. Außerdem sollen gewisse Anzeigeneinstellungen flexibel über eine App vorgenommen werden können.

Möglichkeiten einer Binäruhr

Bei einer Binäruhr wird die Uhrzeit mit binären Anzeigeelementen dargestellt. Dabei wird jede Dezimalziffer in das Dualsystem umgerechnet und angezeigt. Geläufig sind zwei Varianten:

- eine, in der jede Dezimalziffer (Stundenzehner, Stundeneiner usw.) in dualer Form dargestellt wird, und
- eine, in der die Darstellung ohne direkten Bezug zum Dezimalsystem als *BCD*-Codierung (*Binary Coded Decimal*) vorgenommen wird.

Ein Beispiel dazu sehen Sie links in Abbildung 10.4, die rechte Darstellung zeigt eine einfache Umwandlung ins Binärsystem (jede Stelle der Zahl hat den Wert der entsprechenden 2er-Potenz). Die der ersten Ziffer von rechts entsprechende Potenz ist $2^0 = 1$. Multiplizieren Sie jede Ziffer mit der entsprechenden Potenz und summieren Sie sie. Gehen Sie am besten von rechts nach links vor: Stundenzehner: 0001. Von rechts nach links bedeutet das $(1 \times 1) + (0 \times 2) + (0 \times 4) + (0 \times 8) = 1 + 0 + 0 + 0 = 1$.

Eine übersichtliche Darstellung der BCD-Codierung finden Sie in der Wikipedia unter *https://de.wikipedia.org/wiki/BCD-Code*.

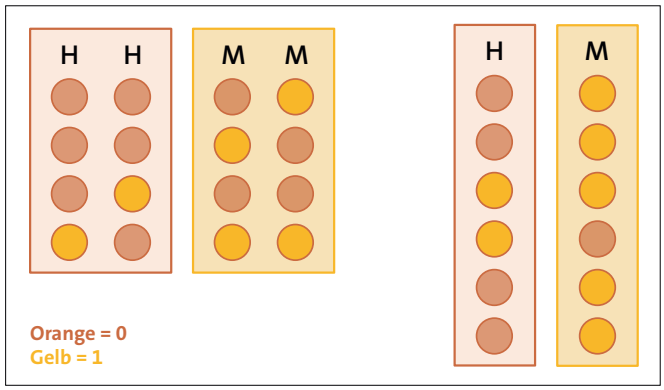


Abbildung 10.4 Prinzip einer Binäruhr

In beiden Fällen, die Sie in Abbildung 10.4 sehen, ist die Uhrzeit 12:59.

Die LED-Matrix

Für die Darstellung der einzelnen Binärwerte bietet sich eine LED-Matrix an. Diese Matrizes sind in den unterschiedlichsten Formaten erhältlich. Für dieses Beispiel kommt eine 4×5-Matrix zur Anwendung, die noch einen Bezug zum Dezimalsystem bietet. Die fünf Spalten sind für Stundenzehner, Stundeneiner, Minutenzehner, Minuteneiner und Sekunden vorgesehen. Die Sekunden sollen dabei in 15-Sekunden-Intervallen zusammengefasst werden, d. h., für die Sekundenwerte 0 bis 14 wird die unterste LED leuchten.

Die Matrix wird realisiert mithilfe von RGB-LEDs mit gemeinsamer Kathode. Dabei sind die Kathoden für Stundenzehner, Stundeneiner usw. miteinander verbunden und werden an einen GPIO-Pin des ESP32 geführt (siehe Abbildung 10.5).

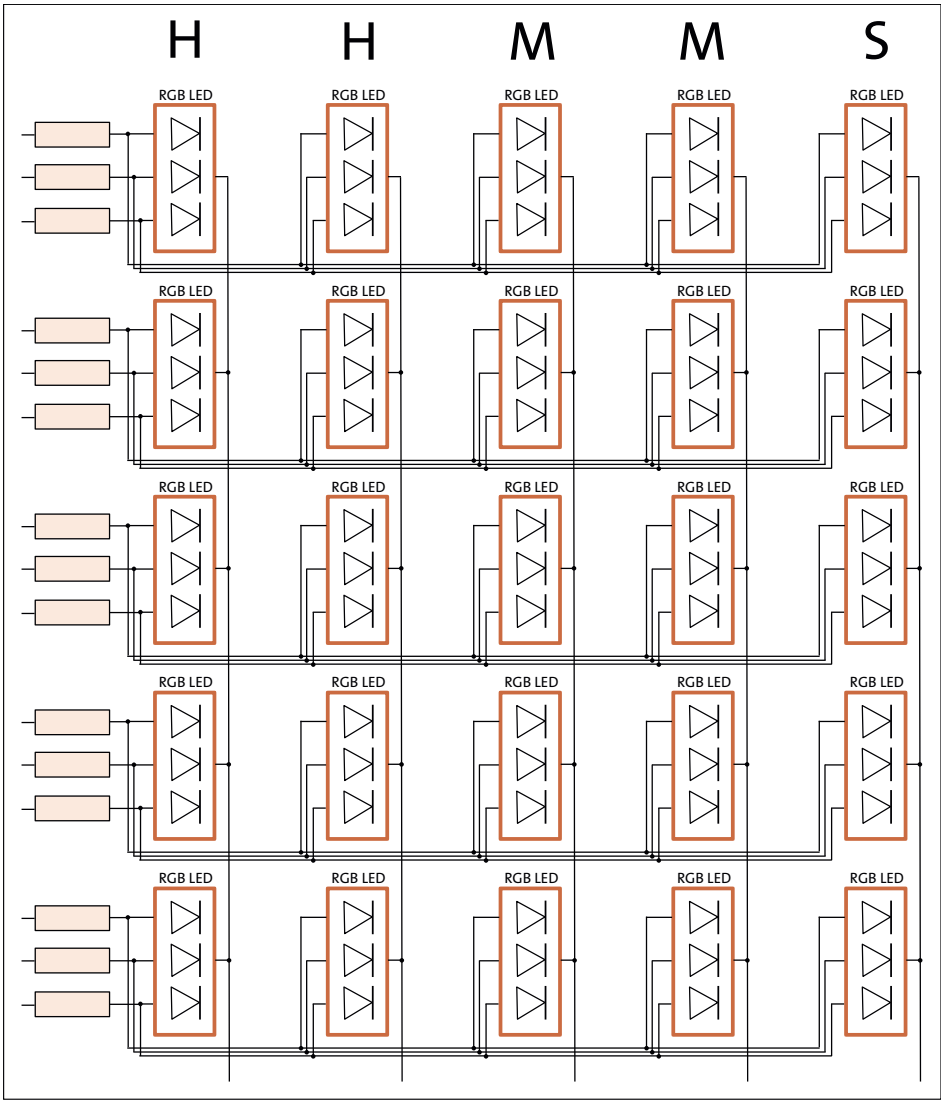


Abbildung 10.5 LED-Matrix – Schaltplan für eine Binäruhr

Miteinander verbunden sind auch die Anschlüsse für Rot, Grün und Blau der RGB-LED-Zeilen. Sie werden über einen Widerstand ebenfalls mit einem GPIO-Pin verbunden, wobei sich der Wert des Widerstands nach den Spezifikationen des Bauteils richtet. Jede Zeile repräsentiert einen Dualwert, unten beginnend mit 1, 2, 4 und 8.

Es bietet sich an, die LED-Matrix in Eigenentwicklung später als fertiges Bauteil auf einer Lochrasterplatine aufzubauen.

Auf der Oberseite der Lochrasterplatine werden die Verbindungen für die Kathoden geführt. Auf der Unterseite befinden sich die Stränge für die RGB-Anoden. Für den besseren Anschluss der Matrix an das ESP32-Dev-KitC V4 sind hinsichtlich der Verbindungen Headerstiftleisten vorgesehen (siehe Abbildung 10.6).

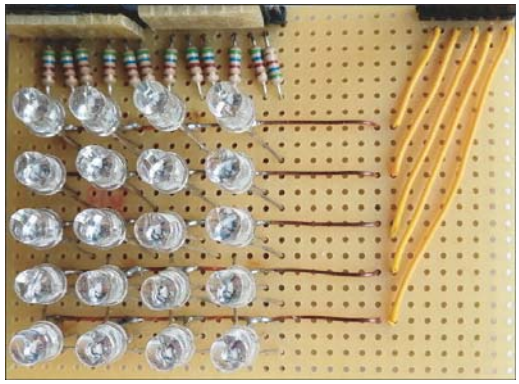


Abbildung 10.6 Bauteil für die Binäruhr

Was die Binäruhr leisten soll

Das Programm für die Binäruhr muss vor allem Folgendes leisten:

- dauerhaftes Speichern der Einstellungen, die hinsichtlich der RGB-Farbauswahl getroffen wurden
- Aufbauen einer WLAN-Verbindung
- Besorgen der aktuellen Zeitangabe
- Aufbereiten und Erzeugen der Daten für die LED-Matrix; die Ausgabe selbst erfolgt dann über PWM.

Der Code setzt sich also aus einigen Blöcken zusammen, die in ihren Grundzügen mit einer Ausnahme bereits angerissen wurden.

Neu und auf den ersten Blick etwas problematisch ist aber die aktuelle Zeitangabe. Anders als der PC oder ein Mikrocomputer verfügen Mikrocontroller von Haus aus nicht über die Möglichkeit, die aktuelle Zeit abzurufen. Da sich der ESP32 aber mit dem Internet verbinden kann, gibt es auch hierfür eine Lösung. Das Stichwort heißt *NTP* (*Network Time Protocol*).

NTP einrichten

Weltweit wird ein weitverzweigtes Netz von Rechnern betrieben, die die aktuelle Zeit auf der Basis von Atomuhren anbieten. Die Tiefe der Darstellung geht bis in den

Nanosekundenbereich, der aber eher für wissenschaftliche Zwecke von Interesse ist. Die Rechner sind in einem Pool organisiert, sodass beim Ausfall eines Zugangs auf einen anderen ausgewichen werden kann. In der Bundesrepublik sind Anbieter beispielsweise die Physikalisch-Technische Bundesanstalt in Braunschweig (drei Zeitserver: *ptbtime1.ptb.de*, *ptbtime2.ptb.de*, *ptbtime3.ptb.de*), Universitäten (z. B. Regensburg mit der URL *ntp.ur.de*) und andere.

Hilfreiche Funktionen für den Zugriff auf einen NTP-Server über den Arduino finden sich in der Bibliothek *NTPClient*, die Sie noch installieren müssen (siehe Abbildung 10.7).

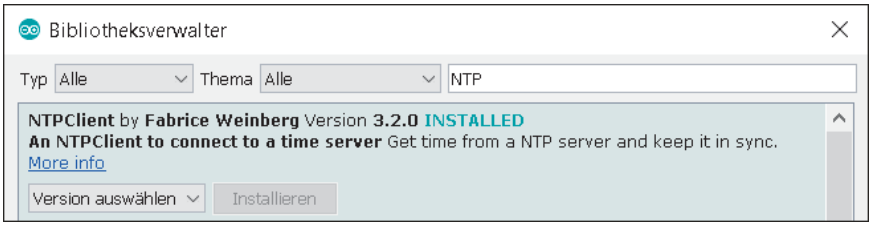


Abbildung 10.7 Die Bibliothek »NTPClient« einfügen

In der Bibliothek (Version: 3.2.0) enthaltene interessante Funktionen sind:

- `getFormattedTime()`: Gibt einen Wert vom Typ *String* zurück, der die Zeit im Format *hh:mm:ss* enthält.
- `getEpochTime()`: Gibt einen Wert vom Typ *Unsigned Long Integer* zurück, der die Zeit in Sekunden seit dem 01.01.1970 enthält.
- `getHours()`/`getMinutes()`/`getSeconds()`: Geben jeweils einen *Integer*-Wert mit der entsprechenden Zeitangabe zurück.
- Der Sketch

Der Sketch aus Listing 10.2 ist aufgrund der Vielzahl der Aufgaben etwas umfangreicher. Zahlreiche Kommentare helfen Ihnen aber beim Verständnis.

```
#include <EEPROM.h>                      // Bibliothek "Flash Memory"
#include <WiFi.h>                        // für WLAN
#include <WiFiUdp.h>
#include <NTPClient.h>                   // für NTP

#define EEPROM_SIZE 15                   // Speichergröße festlegen

#define NTP_OFFSET 2 * 60 * 60           // in Sekunden
#define NTP_INTERVAL 60 * 1000           // in Millisekunden
#define NTP_ADDRESS "3.de.pool.ntp.org"  // "ptbtime1.ptb.de"
```

```
const char* ssid     = "dieRouterSSID";
const char* password = "dasRouterPW";
IPAddress lclIP (192,168,2,207);
IPAddress gateway (192,168,2,1);            // IP-Adresse des Routers
IPAddress subnet (255,255,255,0);
IPAddress primaryDNS (8, 8, 8, 8);          // optional
IPAddress secondaryDNS (8, 8, 4, 4);        // optional

WiFiServer serverWiFi(80);                  // bietet Standardport 80
WiFiClient wifiClient;
WiFiUDP ntpUDP;
NTPClient timeClient(ntpUDP, NTP_ADDRESS, NTP_OFFSET, NTP_INTERVAL);

int eepromRGB [5][3];                       // Werte für [Zeitspalten]/[rgb]

// Pins für die Spalten
int pinGND1 = 18;                           // GPIO21, Stundenzehner
int pinGND2 = 19;                           // GPIO1, Stundeneiner
int pinGND3 = 21;                           // GPIO1, Minutenzehner
int pinGND4 = 22;                           // GPIO22, Minuteneiner
int pinGND5 = 23;                           // GPIO23, Block Sekunden
int pinGNDar [5] = {pinGND1, pinGND2, pinGND3, pinGND4, pinGND5};
// Pins für die Zeilen
// 1. Dimension = 1. (unterste) Reihe;  2. Dimension:
// GPIO-Pin rot, grün, blau
int RGBLedAr [4][3] = {{15,2,0}, {13,12,14}, {27,26,25}, {33,32,5}};

int stunden;
int minuten;
int sekunden;

// Lesen des RGB-Farbstatus
void lesenEeprom() {
  // gesicherten Status aus dem Flash-Memory auslesen
  int sp =0, rgb = 0, stat = 0;
  for (int i = 0; i < 15; i++) {
    stat = EEPROM.read(i);
    if (stat == 255) {
      eepromRGB [sp][rgb]= 1;
    } else {
      eepromRGB [sp][rgb]= 0;
    }
    rgb++;
```

```
    if (rgb > 2) {
      sp++;
      rgb = 0;
    }
  }
}

// Speichern des RGB-Farbstatus
// rx = gerade Zahl   => eeprom = 0   => aus
// rx = ungerade Zahl => eeprom = 255 => an
void speichernEeprom(int rx) {
  int i = rx / 2;
  int iRest = rx  % 2;
  if (iRest == 0) {
    EEPROM.write(i-1, 0);
  } else {
    EEPROM.write(i, 255);
  }
  EEPROM.commit();
  return;
}

// Holen der aktuellen Zeit
void holenNTP() {
  timeClient.update();                      // Aktualisieren des Zeitstempels
  stunden = timeClient.getHours();
  minuten = timeClient.getMinutes();
  sekunden = timeClient.getSeconds();
  return;
}

// Anzeige des Spaltenwerts
void displayWert(int w, int spalte){
  int r = eepromRGB [spalte][0];
  int g = eepromRGB [spalte][1];
  int b = eepromRGB [spalte][2];
  if (w & 1) {
    if (r) digitalWrite(RGBLedAr[0][0], HIGH);
    if (g) digitalWrite(RGBLedAr[0][1], HIGH);
    if (b) digitalWrite(RGBLedAr[0][2], HIGH);
    delay (2);
    digitalWrite(RGBLedAr[0][0], LOW);
    digitalWrite(RGBLedAr[0][1], LOW);
```

```
    digitalWrite(RGBLedAr[0][2], LOW);
  }
  if (w & 2) {
    if (r) digitalWrite(RGBLedAr[1][0], HIGH);
    if (g) digitalWrite(RGBLedAr[1][1], HIGH);
    if (b) digitalWrite(RGBLedAr[1][2], HIGH);
    delay (2);
    digitalWrite(RGBLedAr[1][0], LOW);
    digitalWrite(RGBLedAr[1][1], LOW);
    digitalWrite(RGBLedAr[1][2], LOW);
  }
  if (w & 4) {
    if (r) digitalWrite(RGBLedAr[2][0], HIGH);
    if (g) digitalWrite(RGBLedAr[2][1], HIGH);
    if (b) digitalWrite(RGBLedAr[2][2], HIGH);
    delay (2);
    digitalWrite(RGBLedAr[2][0], LOW);
    digitalWrite(RGBLedAr[2][1], LOW);
    digitalWrite(RGBLedAr[2][2], LOW);
  }
  if (w & 8) {
    if (r) digitalWrite(RGBLedAr[3][0], HIGH);
    if (g) digitalWrite(RGBLedAr[3][1], HIGH);
    if (b) digitalWrite(RGBLedAr[3][2], HIGH);
    delay (2);
    digitalWrite(RGBLedAr[3][0], LOW);
    digitalWrite(RGBLedAr[3][1], LOW);
    digitalWrite(RGBLedAr[3][2], LOW);
  }
  return;
}

// Ausgabe der Zeit
void displayZeit() {
  int s;
  switch (sekunden/15) {
    case 0: s = 1; break;
    case 1: s = 3; break;
    case 2: s = 7; break;
    default  : s = 15;
  }
```

```
  // Stundenzehner
  digitalWrite(pinGND1 , LOW);
  displayWert((stunden/10), 0);
  digitalWrite(pinGND1 , HIGH);
  // Stundeneiner
  digitalWrite(pinGND2 , LOW);
  displayWert((stunden%10), 1);
  digitalWrite(pinGND2 , HIGH);
  // Minutenzehner
  digitalWrite(pinGND3 , LOW);
  displayWert((minuten/10), 2);
  digitalWrite(pinGND3 , HIGH);
  // Minuteneiner
  digitalWrite(pinGND4 , LOW);
  displayWert((minuten%10), 3);
  digitalWrite(pinGND4 , HIGH);
  // Block Sekunden
  digitalWrite(pinGND5 , LOW);
  displayWert(s, 4);
  digitalWrite(pinGND5 , HIGH);
  return;
}

// sendet eine Antwort an den WLAN-Client
void wifiSend (WiFiClient client) {
  client.println("HTTP/1.1 200 OK");
  client.println("Content-type:text/html");
  client.println();
  // der Inhalt der HTTP-Antwort folgt auf den Header
  client.println();
}

// bearbeitet eine Anfrage von dem WLAN-Client
void wifiReceive (WiFiClient client) {
  //Serial.println("Neue Anfrage.");        // Ausgabe einer Meldung im
                                            // seriellen Monitor
  String currentLine = "";                  // String-Variable für
                                            // eingehende Daten
  int cnt = 0;                              // Zähler while client.connected
  while (client.connected()) {              // while-Shleife, solange Client
    cnt++;                                  // Schleifenzähler um 1 erhöhen
    if (client.available()) {               // hat Client Bytes?
      char c = client.read();               // Lesen 1 Byte in Variable c
```

```
      // Serial.print(c);                    // Ausgabe des Bytes im SM
      if (c == '\n') {                       // Ist Byte ein Newline-Zeichen?
         // das Ende der HTTP-Anfrage ist eine Blank-Zeile
         // und zwei Newline-Zeichen hintereinander
         if (currentLine.length() == 0) {     // Ende der HTTP-Anforderung
                                              // durch den Client
          wifiSend(client);
          break;                             // Beenden der while
                                             // client.connected-Schleife
        } else {                             // liegt ein Newline vor
          currentLine = "";                  // Variable currentLine löschen
        }
      } else if (c != '\r') {                // alles andere als ein
                                             // Wagenrücklaufzeichen
        currentLine += c;                    // Zeichen currentLine hinzufügen
      }
      // vollständiger Inhalt von currentLine z. B. GET /1 HTTP/1.1
      // war die Clientanfrage "GET /" gefolgt von einer Zahl?
      if (currentLine[0] == 'G' &&
         currentLine.length() == 7 &&
         isDigit (currentLine[5] )) {
        int n = 0;
        String s1 = "";
        if (isDigit (currentLine[6] )) {     //2-stellig?
          s1 += currentLine[5];
          s1 += currentLine[6];
        } else {
          s1 += currentLine[5];
        }
        n = s1.toInt();
        Serial.print("n = ");
        Serial.println(n);
        speichernEeprom(n);
      }
    }                                        // end if client.available

    // erzwingt disconnect bei fehlendem client.available
    if (cnt > 10000) {
      break;
    }
  }                                          // end while client.connected
  client.stop();                             // Verbindung schließen
```

```
  //Serial.println("Client Disconnected.");
  //Serial.println();
}

void setup() {
  Serial.begin(115200);
  // EEPROM mit festgelegter Größe initialisieren
  EEPROM.begin(EEPROM_SIZE);
  // gesicherten Status aus dem Flash-Memory auslesen
  lesenEeprom();

  // GPIO für die Spalten initialisieren
  for (int i = 0; i < 5; i++) {
    pinMode(pinGNDar [i], OUTPUT);
    digitalWrite(pinGNDar [i], HIGH);
  }

  // GPIO für die Zeilen initialisieren
  for (int i = 0; i < 4; i++) {
    pinMode(RGBLedAr[i][0], OUTPUT);
    digitalWrite(RGBLedAr[i][0], LOW);
    pinMode(RGBLedAr[i][1], OUTPUT);
    digitalWrite(RGBLedAr[i][1], LOW);
    pinMode(RGBLedAr[i][2], OUTPUT);
    digitalWrite(RGBLedAr[i][2], LOW);
  }

  if (!WiFi.config(lclIP, gateway, subnet, primaryDNS, secondaryDNS)) {
    Serial.println("STA failed to configure ");
  }
  Serial.print("Verbindungsaufbau zu ");    // Verbindungsaufbau
                                            // WLAN-Netzwerk
  Serial.println(ssid);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi verbunden");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
  serverWiFi.begin();
```

```
  Serial.println("Start NTP Client!");
  timeClient.begin();
  holenNTP();
  Serial.println("setup beendet");
}

void loop() {
  WiFiClient client = serverWiFi.available();  // horcht auf Clientanfragen
  if (client) {                                // Fragt ein Client an?
    wifiReceive (client);                      // Anfrage aufbereiten
  }
  for (int i = 0; i < 20; i++) {
    displayZeit();
  }
  holenNTP();
  delay(1);                          // warten
}
```

Listing 10.2 Sketch für die Maker-Uhr

Zu Beginn stehen die `#includes`, die wie gehabt die notwendigen Bibliotheken einbinden.

```
#include <EEPROM.h>                          // Bibliothek Flash-Memory
#include <WiFi.h>                            // für WLAN
#include <WiFiUdp.h>
#include <NTPClient.h>                       // für NTP
```

Die EEPROM-Speichergröße wird auf 15 festgelegt, d. h. je ein Byte für jede Spalte und dort für jede RGB-Farbe (5 × 3):

```
#define EEPROM_SIZE 15                       // Speichergröße festlegen
```

NTP verlangt noch einige Einstellungen. Basis für NTP ist die *GMT* (*Greenwich Mean Time*). Sie läuft der hiesigen Zeit (Berlin) um eine Stunde bzw. zwei Stunden hinterher, sodass die Basis um ein entsprechendes Offset zu erhöhen ist. Darüber hinaus werden ein Update-Intervall und der NTP-Time-Server festgelegt:

```
#define NTP_OFFSET 2 * 60 * 60               // in Sekunden
#define NTP_INTERVAL 60 * 1000               // in Millisesekunden
#define NTP_ADDRESS "3.de.pool.ntp.org"      // "ptbtime1.ptb.de"
```

Es folgen die Angaben für die WLAN-Verbindung. Zur Anwendung kommt an dieser Stelle eine feste IP-Adresse, damit diese in der Update-App für die Binäruhreinstellungen fest vorgegeben werden kann:

```
const char* ssid     = "dieRouterSSID";
const char* password = "dasRouterPW";
IPAddress lclIP (192,168,2,207);
IPAddress gateway (192,168,2,1);            //IP-Adresse des Routers
IPAddress subnet (255,255,255,0);
IPAddress primaryDNS (8, 8, 8, 8);          //optional
IPAddress secondaryDNS (8, 8, 4, 4);        //optional
```

Um die Funktionen der Bibliotheken nutzen zu können, sind einige Objekte nötig:

```
WiFiServer serverWiFi(80);                  // bietet Standardport 80
WiFiClient wifiClient;
WiFiUDP ntpUDP;
NTPClient timeClient(ntpUDP, NTP_ADDRESS, NTP_OFFSET, NTP_INTERVAL);
```

Danach werden einige globale Variablen initialisiert, insbesondere für die Zuordnung der GPIO-Pins zu den Anschlüssen der LED-Matrix. Passen Sie unter Umständen die GPIO-Belegung den eigenen Bedürfnissen an; dabei sollten Sie jedoch nicht aus den Augen verlieren, dass einige GPIOs nicht wahlfrei nutzbar sind!

Die Funktion `lesenEeprom()` liest die im EEPROM gespeicherten RGB-Farbwerte aus. Ist der eingelesene Wert 255, wird das korrespondierende Element der RGB-Farbtabelle auf 1 gesetzt, ansonsten auf 0. Auf Basis der Einzelwerte der RGB-Farbtabelle (1 = an, 0 = aus) entscheidet sich im weiteren Programmverlauf, ob die entsprechende Farbe der Binäruhrspalte angezeigt wird oder nicht. Programmtechnisch sind hier sicherlich auch andere Lösungen denkbar, hier soll das aber so genügen.

Die Funktion `speichernEeprom()` speichert den Status einer RGB-Farbe für eine Spalte im EEPROM. Der Status und der zugehörige Tabellenplatz (sprich RGB-Farbe/Spalte) ergeben sich aus dem übergebenen Parameter `rx`. Dieser kann einen Wert von 1 bis 30 annehmen. In der Programmlogik verbirgt sich dahinter ein Zahlenpaar: Eine ungerade Zahl bedeutet »an«, eine gerade »aus« (z. B. `1` / `2`: Dann bedeutet `1`: Farbe Rot der RGB-LEDs in Spalte 1 »an«; und `2` bedeutet: Farbe Rot der RGB-LEDs in Spalte 1 »aus«). Der Tabellenplatz der RGB-Farbtabelle und damit die Spalte der LED-Matrix wird durch eine Division durch 2 ermittelt:

```
int i = rx / 2;
```

Der Status berechnet sich mittels des Modulo-Operators. Ist der Rest der Division 0, also eine gerade Zahl, wird der Status auf 0 gesetzt, d. h. auf »aus«.

```
int iRest = rx  % 2;
if (iRest == 0) {
    EEPROM.write(i-1, 0);
```

```
  } else {
    EEPROM.write(i, 255);
  }
```

Die Funktion `lesenEeprom()` aktualisiert dann die RGB-Farbtabelle, die die Grundlage für die Darstellung der RGB-Farben der Matrix darstellt.

Die Funktion `holenNTP()` holt den aktuellen Zeitstempel und weist den Variablen `stunden`, `minuten` und `sekunden` die neuen Werte zu.

Die Funktionen `displayZeit()` und `displayWert(int w, int spalte)` sind gewissermaßen die Kernfunktionen des Sketchs. Sie sorgen dafür, dass die Zeit in binärer Form angezeigt wird, und steuern die LEDs der LED-Matrix an. Der Grundgedanke hierbei ist, die PWM-Funktionalität zu nutzen, und zwar dergestalt, dass in einem Zyklus eine jede Farbe aller RGB-LEDs für eine kurze Zeit leuchtet, sofern der in der RGB-Farbtabelle gesetzte Status eine »1«, also »an«, aufweist. Damit eine Farbe einer RGB-LED aber angeschaltet wird, muss deren Anode auf logisch HIGH und die Kathode auf logisch LOW gesetzt werden.

Die Funktion `displayZeit()` wertet den in den Variablen `stunden`, `minuten` und `sekunden` gespeicherten Zeitstempel aus. Für jede Spalte (Stundenzehner, Stundeneiner usw.) ermittelt sie den anzuzeigenden Wert und ruft die Funktion `displayWert(int w, int spalte)` auf. Der Parameter `w` enthält den Wert (z. B. 5), der Parameter `spalte` die Spalte. Berechnet werden die Werte durch eine einfache Division durch 10 für die Zehner und mit dem Modulo-Operator für die Einer. Vor dem Aufruf der Funktion wird die Spalte aktiviert, d. h., der GPIO, an den die Kathoden der RGB-LEDs in der Spalte angeschlossen sind, wird auf `LOW` gesetzt. Nach dem Aufruf wird der GPIO durch das Setzen auf `HIGH` wieder deaktiviert (z. B. für Stundenzehner):

```
// Stundenzehner
digitalWrite(pinGND1 , LOW);
displayWert((stunden/10), 0);
digitalWrite(pinGND1 , HIGH);
```

Die Ermittlung des Werts für die Sekunden weicht von diesem Schema aber ab. Da die Anzeige der Sekunden in 15er-Blöcken erfolgen soll (0 bis 14 stehen für die unterste LED der Sekundenspalte, 0 bis 30 für die untersten zwei LEDs der Sekundenspalte usw.), muss der Wert anderweitig ermittelt werden. Dies geschieht durch eine Division durch 15; das Ergebnis ist der Sekundenbereich, in dem die Zeitangabe liegt. Anschließend wird das Ergebnis in eine Zahl umgesetzt, die die Funktion `displayWert(int w, int spalte)` auswerten kann:

```
int s;
  switch (sekunden/15) {
    case 0: s = 1; break;
```

```
    case 1: s = 3; break;
    case 2: s = 7; break;
    default  : s = 15;
  }
```

Die Funktion `displayWert(int w, int spalte)` schaltet nun die Anoden der RGB-LEDs auf `HIGH` (an) bzw. `LOW` (aus). Welche Spalte angesteuert werden soll, ergibt sich aus dem Parameter `spalte`. Die RGB-Farbtabelle liefert die entsprechenden Statuswerte für die Variablen:

```
int r = eepromRGB [spalte][0];
int g = eepromRGB [spalte][1];
int b = eepromRGB [spalte][2];
```

Danach wird jede RGB-LED einer Spalte durch eine UND-Verknüpfung mit dem Parameter `w` daraufhin überprüft, ob sie angeschaltet werden soll. Beispiel: Ist der Wert 3 (also binär 0011), führt eine UND-Verknüpfung mit 1 und 2 zu *true*, mit 4 und 8 jedoch zu *false*.

```
if (w & 1) {
```

true hat den neuen GPIO-Status `HIGH` zur Folge, sofern die Angaben der RGB-Farbtabelle dies so vorsehen:

```
if (r) digitalWrite(RGBLedAr[0][0], HIGH);
if (g) digitalWrite(RGBLedAr[0][1], HIGH);
if (b) digitalWrite(RGBLedAr[0][2], HIGH);
```

Das anschließende `delay(2)` sorgt dafür, dass der Schaltzustand für eine kurze Zeit erhalten bleibt, ehe er in den Ausgangszustand zurückfällt.

Die Funktion `wifiSend (WiFiClient client)` gibt bei einer HTML-Anfrage lediglich eine leere Antwort.

Die Funktion `wifiReceive (WiFiClient client)` greift bei einem über das Netz angestoßenen Wechsel der RGB-Farbtabelle. Ihre wesentliche Aufgabe ist es, die maßgebliche Zeile mit dem neuen Status herauszufiltern und den Status selbst zu isolieren. Ihr Anknüpfungspunkt ist eine Zeile, deren vollständiger Inhalt z. B. `GET /1 HTTP/1.1` ist. An Position 5 beginnt der verschlüsselte Wert für den Status. Er kann einstellig oder zweistellig sein. Mit der String-Object-Funktion `s1.toInt()` wird der Wert in eine Integer-Variable überführt und mit der Funktion `speichernEeprom(n)` gespeichert:

```
if (currentLine[0] == 'G' &&
    currentLine.length() == 7 &&
    isDigit (currentLine[5] )) {
  int n = 0;
```

```
  String s1 = "";
  if (isDigit (currentLine[6] )) {    //2-stellig?
     s1 += currentLine[5];
     s1 += currentLine[6];
  } else {
     s1 += currentLine[5];
  }
  n = s1.toInt();
  Serial.print("n = ");
  Serial.println(n);
  speichernEeprom(n);
}
```

Die Funktion `setup()` weist prinzipiell mit einer kleinen Ausnahme keine Besonderheiten auf: Die Kathoden-GPIOs der LED-Matrix werden auf `HIGH`, die Anoden-GPIOs auf `LOW` gesetzt. So ist gewährleistet, dass alle LEDs initial ausgeschaltet sind.

Auch die `loop()`-Funktion ist wenig spektakulär. Einstellungsmöglichkeiten ergeben sich hinsichtlich der Anzahl der Schleifendurchläufe für die Zeitausgabe (hier 20):

```
for (int i = 0; i < 20; i++) {
  displayZeit();
}
```

Das `delay(1)` am Ende erlaubt dem ESP32, Selbstverwaltungsprozesse durchzuführen. Ob dies notwendig ist, kann experimentell verifiziert werden.

Der Sketch beginnt nach dem Hochladen mit ein paar Ausgaben im seriellen Monitor.

Beim ersten Start sollten alle EEPROM-Bytes den Wert 255 haben, sodass alle eingeschalteten RGB-LEDs einer Spalte weiß erscheinen.

Danach ist die aktuelle Uhrzeit 13:42 und 45–59 Sekunden (siehe Abbildung 10.8).

Abbildung 10.8 Binäruhranzeige

Nun können über eine Browsereingabe die RGB-Farben für die einzelnen Spalten aus- oder wieder eingeschaltet werden (siehe Abbildung 10.9 und Abbildung 10.10, z. B. *http://192.168.2.207/4* für grüne Stundenzehner aus).

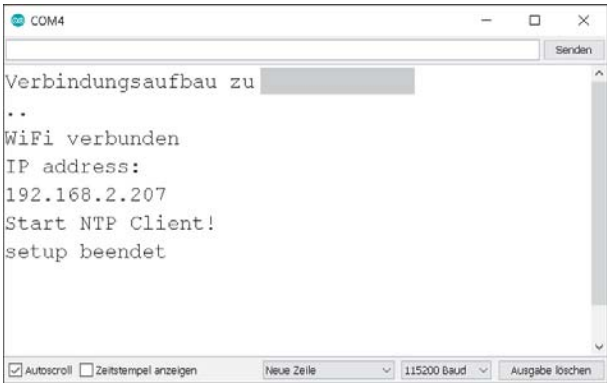


Abbildung 10.9 Startmeldungen der Binäruhr im seriellen Monitor

Abbildung 10.10 Binäruhranzeige mit geänderten RGB-Farben

RGB-Farben über eine App festlegen

Sie können die Farbeinstellungen auch smarter über eine App bestimmen. Da die Anzahl der kostenfreien Widgets bei *Blynk* beschränkt ist, kommt hierfür eher der *MIT App Inventor* infrage. Als Tool für den Verbindungsaufbau scheidet allerdings Bluetooth aus, da die Funktionen für den ESP32 mehr Programmspeicherplatz benötigen, als neben dem bereits existierenden Code noch zur Verfügung steht. Das stellt jedoch kein Problem dar, weil der Baukasten für die App auch über ein Tool verfügt, das eine Verbindung mittels WLAN ermöglicht.

Die App selbst zu erstellen, ist weniger schwierig als vielmehr eine zeitaufwendige Fleißarbeit. Dabei sind auch mehrere Lösungen denkbar, insbesondere ob nun das

Schalten der Farben über einen Schaltknopf oder einen Schalter realisiert wird. Sinnvollerweise werden diese in einer Tabelle mit entsprechenden Überschriften (hier für die Farben Rot, Grün und Blau) sowie Spaltenbezeichnungen (hier für die einzelnen Spalten der RGB-Matrix) angeordnet.

Aus optischen Gründen könnte das erste Element eine Komponente *Label* aus der Gruppe *User Interface* sein. Für die Tabelle wird die Komponente *TableArrangement* aus der Gruppe *Layout* benötigt. Die Eigenschaften werden mit sechs Spalten (vier Spalten mit Werten und je eine Trennspalte zwischen Rot/Grün sowie Grün/Blau) und sechs Reihen spezifiziert.

Als Nächstes gilt es, die einzelnen Tabellenfelder zu füllen. Für die Spaltenbezeichnungen sind das fünf *Label*-Komponenten, drei für die Farbbezeichnungen und zwei für die Zwischenräume zwischen den »Farbspalten«. Die Eigenschaften (siehe Abbildung 10.11) müssen aber noch angepasst werden:

- **»Farbspalten«**:
  - Width = 10 percent
  - Text = Rot/Grün/Blau
  - TextAlignment = center
- **Zwischenräume**:
  - Width = 5 percent
  - Text löschen
  - TextAlignment = center

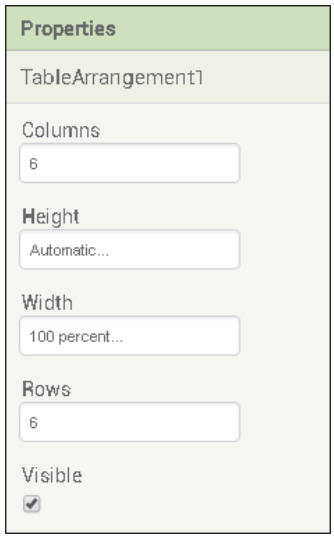


Abbildung 10.11 »TableArrangement« für die Binäruhr im »MIT App Inventor«

Für die Reihenbezeichnungen sind ebenfalls fünf *Label*-Komponenten nötig. Die Eigenschaften sind WIDTH = *30 percent* und TEXT = Stundenzehner usw.

Die Schaltvorgänge sollen an dieser Stelle mit Schaltern erfolgen. Benötigt wird also 15-mal die Komponente *switch* aus der Gruppe *User Interface*. Es empfiehlt sich, den einzelnen Komponenten sprechende Namen zu geben (so z. B. »hzR« für den Schalter, der die Farbe Rot für die Stundenzehner bestimmt). Bei den Eigenschaften wird WIDTH auf »10 percent« gesetzt und der Inhalt des Textfelds gelöscht.

Wenn Sie mögen, können Sie noch die Eigenschaft TRACKCOLOURACTIVE in Rot, Grün oder Blau ändern, sodass der aktivierte Schalter auch farblich ansprechend dargestellt wird.

Übertragen Sie schließlich noch die Komponente *WebViewer* aus der Gruppe *User Interface* in den Viewer.

Wechseln Sie nun in die Block-Ansicht. Ziehen Sie auch hier die benötigten Blöcke aus der Blockliste in den Viewer und setzen Sie diese dort zusammen. Erstellen Sie dabei für jede *switch*-Komponente eine Schaltstruktur (siehe Abbildung 10.12 und Abbildung 10.13).

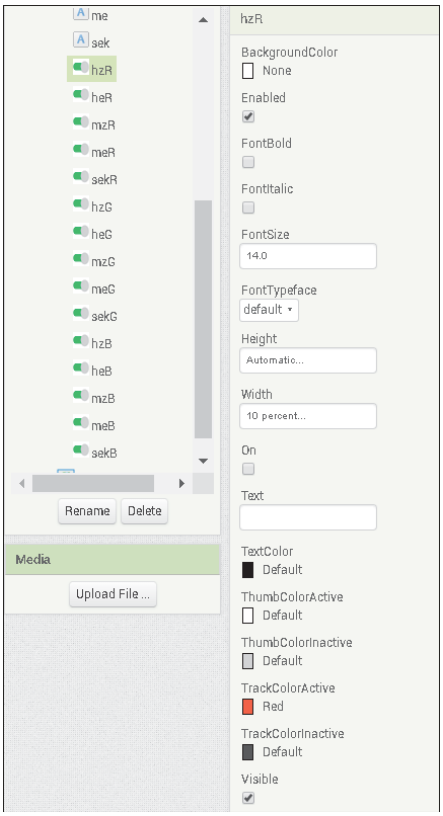


Abbildung 10.12 »switch«-Komponente im »MIT App Inventor«

Abbildung 10.13 Die App im Viewer

Bestimmendes Element ist der HTTP-Request. Er enthält die IP-Adresse des ESP32, die in dem Sketch festgelegt wurde (hier `192.168.2.207`), sowie den Inhalt der Anfrage (hier `1` für »ein« bzw. `2` für »aus«), getrennt durch ein /. Abbildung 10.14 können Sie als eine Art Schritt-für-Schritt-Anleitung betrachten, die Ihnen den Aufbau der Struktur verdeutlicht.

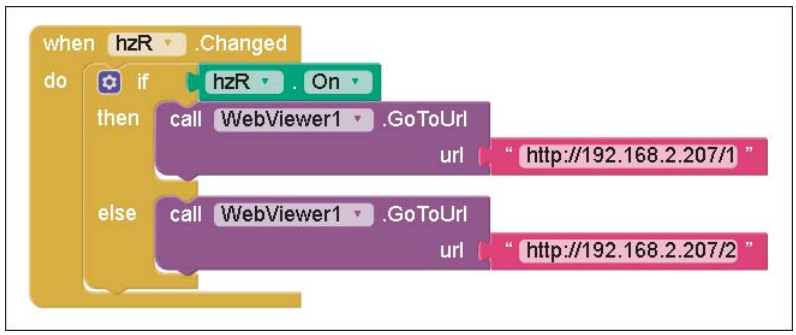


Abbildung 10.14 Blöcke im »MIT App Inventor«

Laden Sie die App nun auf Ihr Smartphone hoch (siehe Abbildung 10.15) und steuern Sie auf diese Weise Ihre Uhr.

Abbildung 10.15 Die fertige App: Binäruhr auf Basis des MIT App Inventor

10.3 Das Dateisystem einmal anders

Bereits in Kapitel 4 (Partitionstabelle), Kapitel 7 (Daten auf dem ESP speichern) und Kapitel 8 (SD-Karte) haben Sie das Dateisystem des ESP32 kennengelernt. Der Ansatz war, aus dem Arduino-Code Dateien zu erstellen und Daten in Dateien zu schreiben. Das ist ausreichend, wenn es darum geht, Messwerte zu protokollieren oder anderweitig sich ändernde Daten abzuspeichern. Häufig ergibt sich aber die Situation, dass große Datenmengen (z. B. Parameter) weitgehend unverändert bleiben. Diese im Sketch als Code zu implementieren, bläht den Code auf und macht ihn unter Umständen unübersichtlich. Eleganter ist es, solche Dateien im Dateisystem zu verankern.

In diesem Abschnitt geht es darum,

- den HTML-Code des Beispiels »ESP32 als ›Station‹« aus Abschnitt 8.5.2 auszulagern,
- die Einbindung von JavaScript zu zeigen,
- Beschriftungen variabel zu gestalten und
- einen *asynchronen Webserver* aufzusetzen.

Zunächst müssen Sie einige Vorbereitungen treffen.

Asynchrone Webserver

Ältere Webanwendungen sind synchroner Natur. Der Benutzer interagiert mit der im Browser dargestellten Weboberfläche, der Browser sendet dann Anforderungen an den Server zurück, und der Server antwortet auf diese Anforderungen mit einer neuen Darstellung der Weboberfläche. Das heißt, dass der Benutzer als Darstellung nur eine Momentaufnahme eines dynamischen Systems erhält. Dieser Schnappschuss ist zwischen Benutzerinteraktionen veraltet und spiegelt nicht immer den aktuellen Status des Systems. Daran ändert auch die Verwendung von XmlHttpRequest- und Ajax-Techniken im Kern nichts.

Das asynchrone Web unterscheidet sich grundlegend, da es die spontane Übermittlung von neuen Darstellungen an den Benutzer und somit jederzeit die Lieferung eines aktuellen Informationsstands ermöglicht.