



elektorMAG
design > share > earn

598
* SEIT 1961 *

DEZ. 2023 & JAN. 2024
ELEKTORMAGAZINE.DE

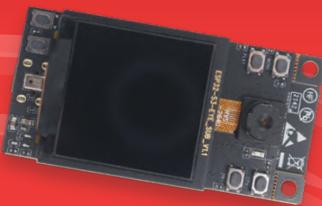
SONDERHEFT

140
Seiten

Gast-Ausgabe von



ESPRESSIF



Prototyping mit Espressif-Chips



ADF, IDF und andere SDKs



Einblicke von
**Espressif-
Ingenieuren**

Automatisierung
mit Rainmaker
und Matter

Ausprobiert:
die ESP32-S3-BOX-3



Ausgepackt: der ESP32-P4
Die nächste Ära der
Mikrocontroller

S. 59

Akustischer Fingerabdruck
Songerkennung mit dem ESP32

S. 80



AIoT-Chip-Innovation
Ein Interview mit Espressif-CEO
Teo Swee-Ann

S. 35





Design-In Expertise & Service

YOUR PARTNER FOR
 ESPRESSIF

-  Ineltek  Leading Espressif franchise distribution partner
-  Experience  36 years of experience in the semiconductor industry
-  Latest News  Ineltek is always up to date regarding Espressif's innovations – Contact us!
-  Innovation  Features are missing? We address your requests for next product generations
-  Application Support  Dedicated and focused in-house support offering
-  Time to Market  Espressif & Ineltek's FAE team are in close & regular contact to speed up your designs
-  Information  We inform you on Espressif's products & wireless trends in trainings & webinars
-  Supply  Popular Espressif SoCs, modules and EVKs available from stock

CONTACT US FOR PARTS & SUPPORT



www.ineltek.com

Ineltek is the worldwide independent distributor with a **Passion for Innovation** and a **Commitment to Service**.

Founded in 1987, Ineltek gained the trust of thousands industrial customers as a technical semiconductor and design-in service provider. We work with your team to ensure our mutual success by providing the highest level of technical and commercial services for your projects.



Follow us!

Start your career with us!

Apply now at personal@ineltek.com



- Field Sales Engineer (m/f/x)
- Field Application Engineer (m/f/x)
- Line Management / Marketing (m/f/x)

INELTEK GmbH
Heidenheim, Wien, Castelfranco, London
Hamburg, München, Frankfurt, Dresden



54. Jahrgang, Nr. 598
 Dezember 2023/Januar 2024
 ISSN 0932-5468

Das Elektor Magazin wird 8 mal im Jahr herausgegeben von
Elektor Verlag GmbH
 Lukasstraße 1, 52070 Aachen (Deutschland)
 Tel. +49 (0)241 95509190
www.elektor.de | www.elektormagazine.de

Für alle Ihre Fragen
service@elektor.de

Mitglied werden
www.elektormagazine.de/abo

Anzeigen
 Büsra Kas
 Tel. +49 (0)241 95509178
busra.kas@elektor.com
www.elektormagazine.de/mediadaten

Urheberrecht
 © Elektor International Media b.v. 2023
 Die in dieser Zeitschrift veröffentlichten Beiträge, insbesondere alle Aufsätze und Artikel sowie alle Entwürfe, Pläne, Zeichnungen einschließlich Platinen sind urheberrechtlich geschützt. Ihre auch teilweise Vervielfältigung und Verbreitung ist grundsätzlich nur mit vorheriger schriftlicher Zustimmung des Herausgebers gestattet. Die veröffentlichten Schaltungen können unter Patent- oder Gebrauchsmusterschutz stehen. Herstellen, Feilhalten, Inverkehrbringen und gewerblicher Gebrauch der Beiträge sind nur mit Zustimmung des Verlages und ggf. des Schutzrechtsinhabers zulässig. Nur der private Gebrauch ist frei. Bei den benutzten Warenbezeichnungen kann es sich um geschützte Warenzeichen handeln, die nur mit Zustimmung ihrer Inhaber warenzeichengemäß benutzt werden dürfen. Die geltenden gesetzlichen Bestimmungen hinsichtlich Bau, Erwerb und Betrieb von Sende- und Empfangseinrichtungen und der elektrischen Sicherheit sind unbedingt zu beachten. Eine Haftung des Herausgebers für die Richtigkeit und Brauchbarkeit der veröffentlichten Schaltungen und sonstigen Anordnungen sowie für die Richtigkeit des technischen Inhalts der veröffentlichten Aufsätze und sonstigen Beiträge ist ausgeschlossen.

Druck
 Senefelder Misset, Mercuriusstraat 35
 7006 RK Doetinchem (Niederlande)

Distribution
 IPS Pressevertrieb GmbH, Carl-Zeiss-Straße 5
 53340 Meckenheim (Deutschland)
 Tel. +49 (0)2225 88010



Beschleunigung der IoT-Innovation



C. J. Abate (Content Director, Elektor)
 Jens Nickel (Chefredakteur, Elektor)

Teo Swee Ann
 (Gründer/CEO, Espressif Systems)

Nach dem durchschlagenden Erfolg der Zusammenarbeit mit SparkFun (2021) und Arduino (2022) freuen wir uns, Espressif als Guest für 2023 an Bord zu haben. Vielen Dank, Espressif! Mit bahnbrechenden Lösungen wie dem ESP8266 und ESP32 hat das Espressif-Team den innovativsten professionellen Ingenieuren, ernsthaften Makern und zukunftsorientierten Studenten der Welt immer wieder Werkzeuge zur Verfügung gestellt. Mit dieser Gastausgabe von Elektor und der daraus resultierenden spannenden Zusammenarbeit mit den talentierten Ingenieuren von Espressif geben wir den Elektor-Lesern auf der ganzen Welt verschiedene Beispiele für die vielfältigen Lösungen und das Know-how des Unternehmens an die Hand.

Was erwartet Sie also bei der Lektüre dieser Gastausgabe von Elektor? In typischer Elektor-Manier haben wir diese Ausgabe mit einem breiten Spektrum an Inhalten gefüllt, von Projekten bis hin zu Tutorials zu Themen wie Liedererkennung auf einem ESP32, Embedded-Entwicklung mit Rust, ESP32 und ChatGPT, Gesichtserkennung mit dem ESP32-S3-EYE, praktische technische Einblicke und vieles mehr. Wir sind zuversichtlich, dass das Magazin - zusammen mit der kostenlosen Bonusausgabe, die wir auf unserer Website und über unseren Newsletter mit der Community teilen - monatlang Innovationen und Dutzende von spannenden neuen Elektronikprojekten und -anwendungen hervorbringen wird.

Unabhängig davon, ob Sie bereits Erfahrung mit Espressif-Lösungen haben oder ob Sie die Produkte des Unternehmens noch nicht kennen, sollten Sie alle Ihre Espressif-basierten Projekte auf der Online-Plattform Elektor Labs unter elektormagazine.de/labs veröffentlichen. Wir freuen uns darauf, mehr über Ihre Entwürfe zu erfahren. Viel Spaß mit dieser Ausgabe und viel Erfolg bei Ihrem nächsten Projekt!

Willkommen zu dieser speziellen Gastausgabe der Zeitschrift Elektor, die für ihr Engagement bei der Förderung von Innovation, Zusammenarbeit und Wissensaustausch bekannt ist und von Espressif Systems mitverfasst wurde. Wir möchten dem fantastischen Elektor-Team und den Redakteuren für ihre unglaubliche Arbeit und unermüdliche Unterstützung bei der Erstellung dieser Ausgabe herzlich danken. Es sind ihre Beiträge, die dies alles erst möglich gemacht haben.

Wir möchten auch der Espressif-Community und unseren geschätzten Partnern, die uns nicht nur unterstützen, sondern auch aktiv zu dieser Ausgabe beigetragen haben, unsere tiefste Anerkennung aussprechen. Ihr engagierter Einsatz und ihre wertvollen Erkenntnisse haben den Inhalt dieser Zeitschrift maßgeblich geprägt.

Diese Sonderausgabe ist ein Zeugnis für den kollaborativen Geist und die gemeinsame Leidenschaft für die Open-Source-Technologie, die unsere Community und unsere Partner verkörpern. In dieser Ausgabe tauchen wir tief in die Prinzipien ein, die Espressif Systems an die Spitze der IoT-Branche katapultiert haben. Sie erhalten einen Einblick in unsere Open-Source-Initiativen und erfahren, wie sie dazu beigetragen haben, eine robuste Entwickler-Community aufzubauen, die wiederum eine entscheidende Rolle auf unserem Weg der Innovation und der Stärkung unserer Position gespielt hat.

Wir blicken auch in die Zukunft und sehen das transformative Potenzial von generativen KI-Technologien wie ChatGPT. Wir werden erleben, wie sie ganze Branchen umgestalten - und wie Espressif Systems plant, sie für Innovationen zu nutzen, mit Einblicken aus unserer Community und von Partnern.

Begleiten Sie uns auf dieser Reise, auf der wir Technologie, Innovation und Gemeinschaft erkunden. Gemeinsam werden wir die erstaunlichen Inhalte dieser Gastausgabe entdecken, die Sie sicher für die grenzenlosen Möglichkeiten der modernen vernetzten Welt begeistern werden.

Bleiben Sie innovativ und teilen Sie weiter!



Unser Team

Chefredakteur: Jens Nickel (v.i.S.d.P.) | **Redaktion:** Asma Adhimi, Roberto Armani, Eric Bogers, Jan Buiting, Stuart Cording, Rolf Gerstendorf (RG), Ton Giesberts, Hedwig Hennekens, Saad Imtiaz, Alina Neacsu, Dr. Thomas Scherer, Clemens Valens, Brian T. Williams | **Regelmäßige Autoren:** David Ashton, Tam Hanna, Ilse Joostens, Prof. Dr. Martin Ossmann, Alfred Rosenkränzer | **Grafik & Layout:** Harmen Heida, Sylvia Sopamena, Patrick Wielders | **Herausgeber:** Erik Jansen | **Technische Fragen:** redaktion@elektor.de



Elektor ist Mitglied des 1929 gegründeten VDZ (Verband Deutscher Zeitschriftenverleger), der „die gemeinsamen Interessen von 500 deutschen Consumer- und B2B-Verlagen vertritt.“

ESP32 und ChatGPT

Auf dem Weg zum selbst-programmierenden System



AIoT-Chip-Innovation

Ein Interview mit dem Espressif-CEO Teo Swee-Ann

35



Rubriken

- 3 Impressum
- 50 Mein Elektronik-Arbeitsplatz Einblicke und Tipps von Espressif-Ingenieuren
- 106 Ihr Provider für AIoT-Lösungen Einblicke von Espressif
- 138 Tech the Future: Wohin steuert Smart Home IoT?



Hintergrund

- 13 Tutorial ESP Launchpad Von Null auf Flash in wenigen Minuten
- 28 Von der Idee zur Schaltung mit dem ESP32-S3 Ein Leitfaden für die Prototypenentwicklung mit Espressif-Chips
- 35 AIoT-Chip-Innovation Ein Interview mit Espressif-CEO Teo Swee-Ann
- 40 ESP32 mit Wokwi simulieren Der virtuelle Zwilling Ihres Projekts
- 46 Ausprobiert: die ESP32-S3-BOX-3 Eine umfassende AIoT-Entwicklungsplattform
- 53 Die ESP-RainMaker Story Wie wir „Ihre“ IoT-Cloud entwickelten
- 56 Zusammenbau des Elektor-Kits Cloc 2.0 Ein Elektor-Produkt, ausprobiert von Espressif
- 59 Ausgepackt: der ESP32-P4 Die nächste Ära der Mikrocontroller



- 64 Rust + Embedded Ein Power-Duo für die Entwicklung
- 70 Wer sind die Rust-begeisterten Embedded-Entwickler? Wie Espressif Embedded Rust für den ESP32 kultiviert
- 74 Die SoC-Reihe von Espressif auf dem Zeitstrahl
- 76 Aufbau einer SPS mit Espressif-Lösungen Mit den Fähigkeiten und Funktionen des ISOBUS-Protokolls
- 78 Das ESP32-S3-VGA-Board Bitlunis aufregende Reise in die Produktentwicklung
- 100 Interview mit Arduino über den Nano ESP32
- 110 Rationelle MCU-Programmierung dank ESP Privilege Separation
- 132 Matter – ein Meilenstein für das Smart Home IoT-Potenzial von Smart Home freigeschaltet

Industry

- 89 Für ein einfacheres und komfortableres Leben Ein Amateurprojekt basierend auf dem ESP8266-Modul von Espressif
- 90 Wie man IoT-Apps ohne Software-Expertise erstellt Mit der Blynk-IoT-Plattform und Espressif-Hardware
- 92 Eine intelligente Benutzeroberfläche auf dem ESP32
- 94 Schnelle und einfache IoT-Entwicklung mit M5Stack
- 99 Ein Distributor für IoT und mehr - mit Mehrwert



Knopfzellen-Sender mit ESP32-C2

5 Jahre Batterie-Lebensdauer

126



Projekte

6 Ein WLAN-Bilderrahmen mit Farb-E-Ink

16 ESP32 und ChatGPT

Auf dem Weg zu einem selbst-programmierenden System

24 Walkie-Talkie mit ESP-NOW

Nicht ganz WLAN, nicht ganz Bluetooth, aber...

80 Akustischer Fingerabdruck mit ESP32

Songerkennung mit dem Open-Source-Projekt Olaf

84 Zirkularer Weihnachtsbaum 2023

Weihnachtszeit, eine High-Tech-Annäherung

96 Prototyping eines Energiezählers mit ESP32

114 Ein Open-Source-Spracherkennungsserver

...und die ESP-S3-BOX-3

119 Das mitdenkende Auge

Gesichtserkennung und mehr mit ESP32-S3-EYE

126 Knopfzellen-Sender mit ESP32-C2

Entwurf und Leistungsbewertung



Vorschau

Elektor Januar/Februar 2024

Das Januarheft ist wie immer randvoll gefüllt mit Schaltungsprojekten, Grundlagen sowie Tipps und Tricks für Elektroniker. Schwerpunkt wird das Thema **Leistung und Energie** sein.

Aus dem Inhalt:

- ESP32-Energiemeter
- Balkonkraftwerke optimieren
- Variable lineare Spannungsversorgung
- Einfacher PV-Leistungsregler
- Waage für die smarte Küche
- Programmieren für PC, Tablet und Smartphone
- Projekt in Kürze: Lader/Entlader

Elektor Januar/Februar 2024 erscheint am 10. Januar 2024.

Änderungen vorbehalten!

BONUS-AUSGABE

Möchten Sie mehr Inhalte von Elektor und Espressif? In den kommenden Wochen werden wir eine Bonus-Ausgabe von Elektor veröffentlichen, die ebenfalls von Espressif gestaltet wurde und weitere Projekte, Tutorials und Hintergrundartikel enthält: ein Dekatron im Espressif-Stil, einen ESP32-basierten Authentifizierungs-Dongle, ein ESP-BOX-basiertes, sprechendes ChatGPT-System, ein Interview mit dem Entwickler des Home Assistant Paulus Schoutens und vieles mehr! Abonnieren Sie das E-Zine von Elektor (elektormagazine.de/ezine) und wir liefern Ihnen die Bonusausgabe direkt in Ihren Maileingang!



Ein WLAN-Bilderrahmen mit farbigem E-Ink-Display

Von Jeroen Domburg (Espressif)

E-Ink-Displays sind auf dem Markt weit verbreitet, aber es gibt sie überwiegend in schwarz-weiß, und die wenigen Farbdisplays sind ziemlich teuer und schwer an ein DIY-Projekt anzupassen. In diesem Artikel beschreiben wir den Einsatz eines preisgünstigen 7-Farben-Displays mit guter Farbtreue, welche durch Dithering-Techniken erreicht wird. Das Display kann sich über ein ESP32-C3-WROOM-02-Modul von Espressif mit einem drahtlosen Netzwerk verbinden.

Vor einiger Zeit wurde unsere Kleine geboren. Da ihre (Ur-)Großeltern über den ganzen Globus verstreut sind, können Real-Life-Besuche leider nicht allzu oft stattfinden. Natürlich gibt es heutzutage Videotelefonie, so dass wir in Kontakt bleiben, aber ich wollte, dass alle sie auch sehen können, wenn wir nicht gemeinsam online sind. Also dachte ich: Warum nicht einen Bilderrahmen mit einem Farbdisplay und einem WLAN-Chip bauen? Gerade weil das Kind so schnell wächst, wäre es doch schön, jeden Tag ein neues Foto zu verschicken und immer ein aktuelles Bild zu zeigen.

Wahrscheinlich haben Sie schon einmal E-Ink-Displays gesehen: wenn nicht in einem E-Book-Reader, dann sicher in einem Supermarkt oder einem anderen Geschäft, wo sie die alten Preisschilder aus Papier ersetzt haben. Sie eignen sich hervorragend für die Anzeige statischer Bilder, da sie den Anzeigehalt ohne Stromaufnahme beibehalten. Die meisten dieser Bildschirme sind schwarz-weiß, wobei manchmal entweder rot oder gelb als zusätzliche Farboption hinzugefügt ist. Diese Displays sind zwar nett, aber ein Schwarz-weiß-Bild wird den leuchtenden Farben eines glücklichen Babys, das mit seinem bunten Spielzeug herumtollt, nicht gerecht.



Aktuell auf dem Markt

Als ich Anfang 2023 diesen Artikel schrieb, erschienen endlich E-Book-Reader mit Farbbildschirmen im Handel. Ihre Qualität scheint gut zu sein, und die Farben kommen denen in einer Zeitung nahe. Leider sind sie teuer, und die Bildschirme selbst scheinen noch nicht als Ersatzteile erhältlich zu sein, so dass es keine Möglichkeit gibt, sie in einem Selbstbau-Projekt einfach zu verwenden. Es gibt jedoch eine Art von E-Ink-Farbdisplay, die für den Einsatz in DIY-Projekten erhältlich ist. Solche Displays werden hauptsächlich von Waveshare verkauft, einem chinesischen Unternehmen, das sich auf den Maker-Markt spezialisiert hat. Das gängigste Modell ist ein 5,65-Zoll-Display mit sieben Farben und 640x448 Pixeln (**Bild 1**). Mit nur sieben Farben und einer Aktualisierungszeit von etwa einer Minute scheint es der große Bruder der Preisschild-Varianten zu sein, der statische Informationen anzeigt, wobei die Farben zum Beispiel für monochrome Hintergrundfarben verwendet werden. Sie sind sicherlich nicht dazu gedacht, fotorealistische Bilder darzustellen.

Das ist ein wenig bedauerlich: Ein solches E-Ink-Display wäre hervorragend zur Darstellung eines Fotos geeignet, da es keine Hintergrundbeleuchtung benötigt und völlig statisch ist. Ich bin nicht der Einzige, der so denkt, und eine ganze Reihe von Leuten ([1] [2] [3]) haben bereits versucht, den Bildschirm mit Hilfe von Dithering so zu verwandeln, dass Bilder mit einer gewissen Wiedergabtreue dargestellt werden können.

Hardware-Anforderungen

Zunächst einmal musste ich die Hardware bauen. Ich begann meinen Entwurf mit einigen Anforderungen.



Bild 1. Das 5,65 Zoll große, siebenfarbige 640×448-Display von Waveshare. (Quelle: Waveshare)

Ich wollte, dass der Bilderrahmen einmal am Tag eine drahtlose Verbindung zu einem Server herstellt, um neue Bilder zu empfangen. Dann sollte er eines davon auf dem E-Ink-Display anzeigen. Wenn es gelingt, neue Bilder herunterzuladen, wird eines davon angezeigt, andernfalls wird weiterhin das alte Bild angezeigt. Danach geht es für 24 Stunden in den Ruhezustand. Die Auswahl eines Chips für diese Aufgabe war für mich nicht schwer, da ich zufällig für Espressif arbeite, ein Unternehmen, das großartige WLAN-fähige Mikrocontroller herstellt, die auch einen Tiefschlafmodus haben, und ziemlich gut funktionieren. Ich habe dafür einen ESP32C3 ausgewählt: Er ist kostengünstig und hat alle Funktionen, die ich brauche.

Außerdem hatte ich auch einige Anforderungen an die Stromversorgung. Ich wollte, dass das Ding einem normalen Bilderrahmen so ähnlich wie möglich ist (abgesehen von der Tatsache, dass die Bilder nächtens austauscht werden), also kam eine externe Stromversorgung nicht in Frage. Die Stromversorgung musste eine Art interne Batterie sein. Außerdem wollte ich das Gerät international verschicken, also wäre es am besten, wenn es ohne Batterie geliefert werden könnte. Diese Anforderung schloss

jede Art von wiederaufladbaren Li-Ionen-Zellen aus. Wie Sie im Schaltplan in **Bild 2** sehen können, habe ich mich für die Verwendung von zwei AA-Batterien entschieden, die überall erhältlich sind und selbst von den Großeltern mit wenig Problemen bei Bedarf ersetzt werden können. Da zwei AA-Batterien eine Gesamtspannung von anfangs etwa 3,2 V liefern, die während der Lebensdauer abnimmt, brauchte ich eine Boost-Schaltung, um die Spannung auf die 3,3 V zu erhöhen, die das E-Ink-Display und der ESP32C3 benötigen. Die Entladungsdiagramme [4] berücksichtigend, müsste die Boost-Schaltung mit einer Eingangsspannung von bis zu 2,0 V oder so arbeiten, wenn ich das letzte Quäntchen Energie aus einem Paar Alkalibatterien herausholen wollte. Da der ESP32C3 auch

Bild 2. Der komplette Schaltplan des Projekts, einschließlich des 3,3 V DC-DC-Wandlers, des ESP32-C3-WROOM-02-Moduls und der Stromversorgung für das Display.

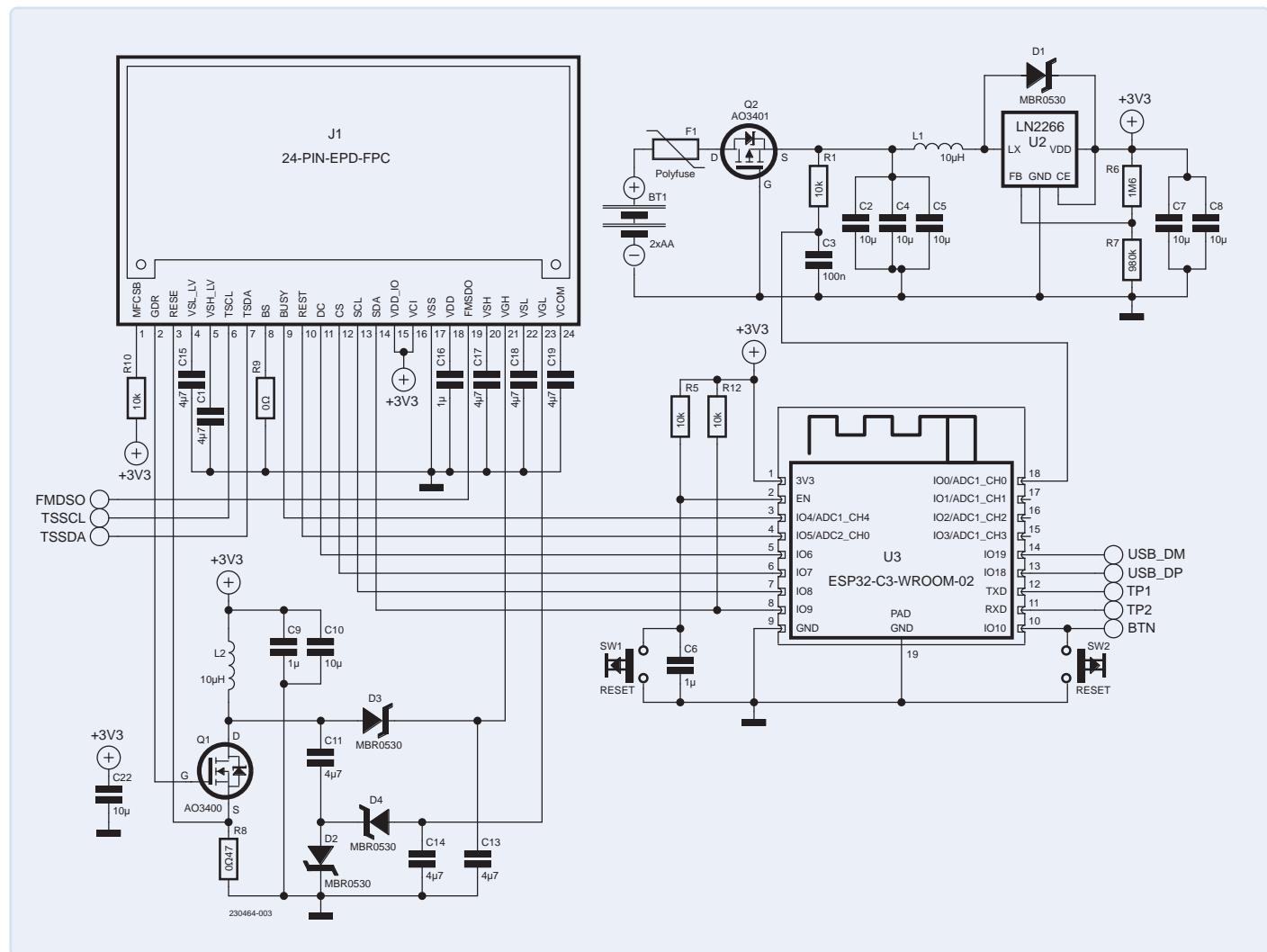
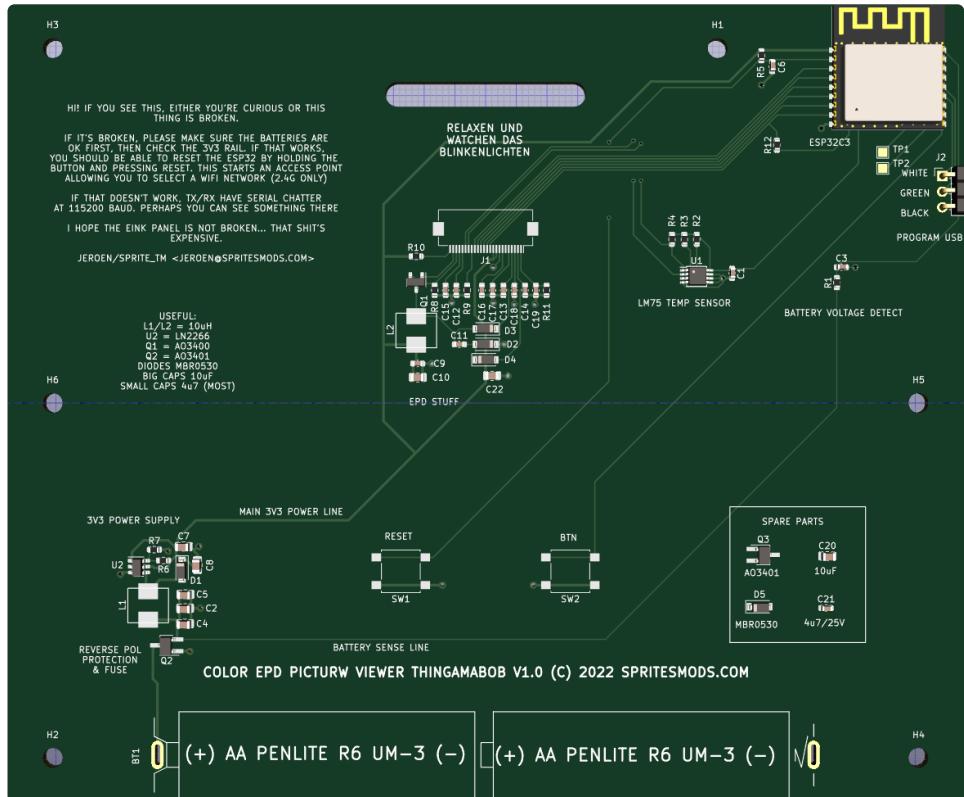


Bild 3. Lustiger Bestückungsaufdruck auf der geräumigen Platine.



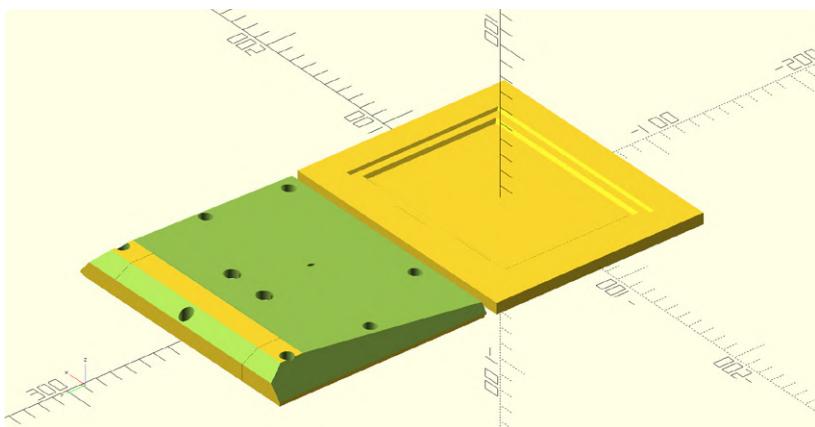
im Standby-Modus ordnungsgemäß mit Strom versorgt werden soll, muss der Ruhestrom entsprechend niedrig sein. Mit diesen Anforderungen machte ich mich auf die Suche nach einem brauchbaren Aufwärtswandler. Ich entschied mich für den Natalinear LN2266. Dieser winzige Chip wird von einem chinesischen Hersteller produziert, was bedeutet, dass er im Allgemeinen etwas weniger anfällig für die aktuellen IC-Lieferschwierigkeiten ist. Er funktioniert ab 2 V, hat einen Leerlaufstrom von 56 μ A und kann die etwa 500 mA WLAN-Startstrom liefern, die der ESP32-C3 benötigt. Ich habe die Schaltung so entworfen, dass der LN2266 seinen Strom von den beiden Batterien über einen P-Kanal-MOSFET bezieht, der so konfiguriert ist, dass er die Batterien und die Elektronik schützt, wenn sie verkehrt herum eingelegt werden. Damit der ESP32-C3 eine Vorstellung davon bekommt, wie viel Saft noch in den Zellen steckt, ist einer seiner ADC-Pins über ein RC-Filter mit der Batteriespannung verbunden. Ursprünglich hatte ich auch eine kleine Polysicherung in das Design eingebaut, aber wie sich herausstellte, konnte dies wegen des zu großen

Spannungsabfalls nicht funktionieren. Ich ersetzte sie durch einen 0-Ohm-Widerstand auf meinen Platinen und entfernte sie vollständig aus den Entwurfsdateien. Der ESP32-C3 wird in Form eines ESP32-C3-WROOM-02-Moduls geliefert, das in den Schaltplänen in Bild 2 rechts zu sehen ist. Dieses Modul enthält den WLAN-Mikroprozessor und 4 MB Flash sowie alle benötigten HF-Komponenten, einschließlich einer Antenne. Zur Programmierung habe ich eine Stiftleiste hinzugefügt, an die ich ein USB-Kabel anlöten oder aufstecken kann. Der interne USB-JTAG-Seriell-Konverter erledigt den Rest. Ich habe die Firmware mit einer OTA-Firmware-Update-Funktion ausgestattet, damit der Bilderrahmen neue Firmware von meinem Server herunterladen kann. So kann ich alle schon im Einsatz befindlichen Geräte mit neuer Firmware aus der Ferne versorgen.

Dann ist da noch das Wichtigste, das E-Ink-Display. Es ist über ein Flexkabel (Folienkabel) mit der Platine verbunden und benötigt einige Kleinigkeiten, um so zu funktionieren, wie es unten links in Bild 2 zu sehen ist: einen MOSFET, eine Spule, einige Kondensatoren und Dioden, um die benötigten Spannungen zu erzeugen, einige Entkopplungskondensatoren und ein oder zwei Widerstände. Das E-Ink-Display besitzt auch einen Anschluss für einen externen LM75-Temperatursensor. Ich habe einen für alle Fälle eingebaut, aber die aktuelle Firmware verwendet ihn nicht, da der Bildschirm auch ohne ihn funktioniert.

All dies befindet sich auf einer sehr geräumigen Platine, deren bauteilseitiger Bestückungsaufdruck in Bild 3 dargestellt ist. Da das nackte E-Ink-Display aus Glas besteht und etwas zerbrechlich ist, habe ich die Platine so gestaltet, dass sie auch als mechanische Stütze dient, um das Gesamtprodukt widerstandsfähiger zu machen. Die Leiterplatte ist etwas größer als das Display, da die Befestigungslöcher, alle SMDs und THT-Bauteile (zum

Bild 4. Rendering des Gehäuses, entworfen mit OpenSCAD.



Beispiel die Batteriekontakte) und die WLAN-Antenne außerhalb untergebracht werden müssen.

Gehäuse

Schließlich gibt es noch das Gehäuse, das ich in OpenSCAD entworfen habe, wie in **Bild 4** zu sehen. Ich habe einige Tricks angewandt, um den Bilderrahmen weniger sperrig aussehen zu lassen. Er muss ziemlich dick sein, weil die AA-Batterien hineinpassen müssen, und ich wollte keine große Ausbuchtung an der Rückseite gestaltet. Diese ist nicht zu sehen, wenn man den Bilderrahmen von vorne betrachtet, so dass der Bilderrahmen insgesamt viel schlanker aussieht. Der Bilderrahmen wirkt im Vergleich zum sichtbaren Bereich des E-Ink-Displays ein wenig klobig. Um dem abzuhelfen, habe ich ein weißes Passepartout hinzugefügt, das schön mit dem schwarzen Gehäuse kontrastiert.

Auf der Rückseite befindet sich das Fach für die Batterien. Ich wollte mich nicht auf zerbrechliche 3D-gedruckte Clips verlassen, um sie an Ort und Stelle zu halten. Das Fach wird durch das Lösen einer Schraube geöffnet. Der Bestückungsaufdruck zeigt deutlich die korrekte Ausrichtung der Batterien. So lassen sich die Batterien einfach und ohne Probleme wechseln.

Software/Firmware und WLAN-Verbindung

Da der ESP32C3 mit Batterien betrieben wird, habe ich versucht, ihn so wenig wie möglich tun zu lassen. Er sollte sich mit dem WLAN verbinden, mit meinem Server kommunizieren, schauen, ob es neue Bilder gibt, die er noch nicht heruntergeladen hat, und wenn ja, diese herunterladen, herausfinden, welches das beste Bild ist (das neueste oder das am seltenste gezeigte), dieses anzeigen und sich dann abschalten. Natürlich gibt es auch noch andere Dinge, die erledigt werden müssen, zum Beispiel die Behebung von Fehlern und der Umgang mit einem niedrigen Batteriestand.

Um eine Verbindung zum WLAN herzustellen, benötigt das Gerät eine SSID und ein Passwort. Ich wollte dies nicht fest in der Software codieren, falls es geändert werden muss. Daher umfasst die Firmware auch einen ESP32-WiFi-Manager [5], der modifiziert wurde, um einige Fehler zu beheben. Wenn man eine der Tasten auf der Rückseite des Bilderrahmens drückt, während man diesen mit der anderen Taste zurücksetzt, wird ein Zugangspunkt (Access Point, AP) gestartet, mit dem man sich verbinden kann. Ein eingebetteter Webserver zeigt dann eine Benutzeroberfläche an, über die man eine neue SSID auswählen und das Passwort dafür festlegen kann.

Nachdem der Bilderrahmen eine WLAN-Verbindung hergestellt hat, versucht er, Daten von einer fest codierten URL abzurufen, um zu sehen, was er tun soll. Die URL verschlüsselt auch einige Statusdaten, wie den MAC des Geräts, die Batteriespannung und die aktuelle Firmware-ID. Der Server gibt die ID der aktuellen Firmware für das betreffende Gerät sowie einen Index der zehn zuletzt



hochgeladenen Bilder zurück. Es werden auch einige Einstellungen gesendet, wie die Zeitzone und die Zeit, zu der der Bilderrahmen versuchen soll, aufzuwachen und ein Update durchzuführen.

Der Bilderrahmen hat in seinem Flash-Speicher Platz für zehn Bilder und ersetzt die ältesten durch neu heruntergeladene Bilder, falls der Server Bilder hat, die nicht schon im lokalen Speicher sind. Auf diese Weise verfügt er immer über einen Vorrat an relativ aktuellen Bildern, was gut ist, wenn die Verbindung aus irgendeinem Grund abbricht. So ist es sogar möglich, den Bilderrahmen an einen Offline-Ort zu betreiben: Während er ohne WLAN-Verbindung alte Bilder wiederverwendet, zeigt er trotzdem jeden Tag etwas anderes an.

Der Großteil der serverseitigen Software ist nicht besonders kompliziert: Es gibt eine einfache Webseite als Front-End, die mit *Cropper.js* [6] erstellt wurde und die man auf seinem Handy oder PC öffnen kann. Sie ermöglicht es, ein Bild auszuwählen und den Teil auszuschneiden, der auf dem Bilderrahmen gezeigt werden soll. Ein kleines Javascript beschneidet und skaliert das Bild dann auf der Client-Seite und sendet die resultierenden Daten an den Server. Der Server nimmt diese Daten entgegen, wandelt sie in Rohdaten für das E-Ink-Display um und speichert sie in einer MariaDB-Datenbank.

Wenn ein Bilderrahmen eine Verbindung herstellt, speichert der Server die gesendeten Daten, so dass ich ein Protokoll der Batteriespannungen habe und sehen kann, ob eine Firmware-Aktualisierung tatsächlich funktioniert hat. Der Server sucht dann in der MariaDB-Datenbank nach den letzten zehn Bildern sowie nach anderen Informationen wie der letzten Firmware-Version, kodiert diese Informationen in JSON und sendet die JSON-Daten zurück. Alles das ist ziemlich trivial.

Dithering

Der einzige wirklich komplizierte Teil ist die Umwandlung des RGB-Bildes in die sieben ziemlich spezifischen Farben, die das E-Ink-Display darstellen kann. Nehmen wir zum Beispiel das Foto in **Bild 5**.

Wenn wir es in schwarz-weiß umwandeln wollten, könnten wir einfach die Luminanz (Helligkeit) für jedes Pixel heranziehen und würden, wenn sie näher an

Bild 5. Das für Testzwecke verwendete Beispielbild.



Bild 6. Erster Konvertierungsversuch mit 100 % schwarz und 100 % weiß.



Bild 7. Schwarz-weiß-Konvertierung mit einem Diffusionsverfahren.



Bild 8. Die Konvertierung aus Bild 7 mit der Addition von RGB-Werten.



Bild 9. Das Beispielbild nach Anwendung des CIEDE2000-Standards.

schwarz liegt, das Ergebnis schwarz machen; wenn sie näher an weiß liegt, das Ergebnis weiß machen. Mit anderen Worten, wir nehmen die nächstgelegene „Farbe“ (wobei wir uns auf die „Farben“ 100 % schwarz und 100 % weiß beschränken) und ändern das resultierende Bild so, wie es in **Bild 6** dargestellt ist.

Dieses Bild hat natürlich keine große Ähnlichkeit mit dem Originalbild. Sogar bei schwarz und weiß könnten wir es besser machen, wenn wir etwas verwenden würden, das man *Fehlerdiffusion* nennt. Jedes Mal, wenn wir ein graues Pixel auf schwarz oder weiß setzen, nehmen wir den Unterschied zwischen der Luminanz des Pixels im Originalfoto und der Luminanz des Pixels, das wir tatsächlich auf dem E-Ink-Bildschirm zeigen (der „Fehler“ in „Fehlerdiffusion“) und addieren ihn teilweise zu den umliegenden Pixeln (die „Diffusion“). Der Diffusionsprozess kann auf verschiedene Arten durchgeführt werden, wobei der Floyd-Steinberg-Algorithmus die gebräuchlichste Art ist, und er ergibt ein ziemlich gutes schwarz-weißes Ditherbild (**Bild 7**).

Wir können dieses Verfahren auch für unser Sieben-Farben-Display verwenden. Das Problem ist, dass die Definition von „nächstliegender Farbe“ sehr viel komplizierter wird, ebenso wie die Definition von „Addieren“. Sogar die Definition von „Farbe“ ist schwierig, da ein

E-Ink-Display keine Hintergrundbeleuchtung hat und die wahrgenommenen Farben je nach Beleuchtung unterschiedlich sind: Wenn man es mit einer Kerzenflamme beleuchtet, sieht man andere Farben als wenn man das Display im hellen Sonnenlicht betrachtet.

Um die Farben zu ermitteln, habe ich ein temperaturregulierbares Licht genommen, es auf 4800 K eingestellt (die von mir geschätzte durchschnittliche Farbtemperatur, bei der das Display betrachtet wird), die sieben Farben als flache Rechtecke auf dem E-Ink-Display angezeigt und ein Foto davon gemacht. Ich importierte dieses Foto in meinen Computer und passte die Farben manuell an, bis die auf dem Bildschirm angezeigten Farben den E-Ink-Farben so nahe wie möglich kamen. Dann nahm ich die durchschnittlichen RGB-Werte der sieben Farben und gab sie in mein Programm ein.

Um für jedes beliebige Pixel des Quellbildes die von diesen sieben Farben „nächstgelegene“ Farbe zu ermitteln, müssen wir zwei Farben miteinander vergleichen, und da wir echte Farben und nicht nur Schwarz und Weiß verwenden, können wir uns nicht mehr mit der Luminanz begnügen. Eine schnelle und einfache Möglichkeit, zwei Farben zu vergleichen, besteht darin, den (linearisierten) RGB-Raum als dreidimensionalen Raum zu betrachten und den euklidischen Abstand zu verwenden, um zu

messen, wie nahe zwei Farben beieinander liegen. Mit diesem Modell können Farben durch einfaches Addieren der RGB-Werte hinzugefügt werden. Wenn wir das Floyd-Steinberg-Dithering so abändern, dass es nach dieser Methode die nächstgelegene Farbe übernimmt, erhalten wir ein einigermaßen akzeptables Bild, wie in **Bild 8** gezeigt.

Gar nicht so schlecht! Allerdings gibt es ein paar seltsame Fehlfarben. Der offensichtlichste Fehler ist, dass der Blumentopf nicht denselben Blauton hat. Das liegt daran, dass das E-Ink-Display einfach nicht über die nötigen Farben verfügt, um diesen Farbton wiederzugeben, und kein Algorithmus der Welt kann das ausgleichen. Aber es gibt noch mehr Ungereimtheiten in Form von seltsamen Farbverläufen (Banding), zum Beispiel im Schatten unter dem rechten Arm des Affen, und der Bauch des Affen hat einen höheren Orange-Anteil als im Originalbild.

Farträume

Das Problem bei der Berechnung von Farbunterschieden im RGB-Raum ist, dass unsere Augen nicht wirklich im linearen RGB-Raum arbeiten. Der Unterschied zwischen zwei Farben ist viel schwieriger zu definieren, und es gibt verschiedene Ansätze, dies zu erreichen. Einer der ersten Ansätze bestand darin, die RGB-Farben in den CIELAB-Farbraum zu konvertieren und die Differenz zu nehmen. Dieser Ansatz wird im Internet häufig empfohlen, aber es hat sich herausgestellt, dass er bei Farben, die nicht vollständig gesättigt sind, keine guten Ergebnisse liefert. Für mich erwies sich die Verwendung des CIEDE2000-Standards [7] als der beste Ansatz. Dies ist eines der aktuelleren Wahrnehmungsmodelle, und obwohl die Berechnung alles andere als trivial ist, liefert es die besten Ergebnisse, wie in **Bild 9** zu sehen ist. Es ist gut, dass ich mich bereits für eine serverseitige Berechnung entschieden habe, so dass ich die Batterien nicht mit dieser teuren Berechnung belasten muss.

Beachten Sie, dass das Banding in den Schatten verschwunden ist und der Bauch des Affen einen treffenderen Rotton aufweist. Es gibt immer noch Fehler im Bild, zum Beispiel die Farbe des Blumentopfs, aber wie ich bereits erwähnt habe, liegt das daran, dass das E-Papier einfach nicht die Farben zur Verfügung stellt, um diesen speziellen Farbton richtig darzustellen.

Um die Geschwindigkeit zu erhöhen, ist diese ganze Logik in einem einfachen C-Programm implementiert. Nachdem das Bild mit *Cropper.js* auf dem Client zugeschnitten wurde, wird es an den Server gesendet. Dort ruft ein PHP-Skript dieses C-Skript auf, um das Bild in E-Ink-Pixel umzuwandeln, die in einer MariaDB-Tabelle gespeichert werden, damit die Bilderrahmen sie bei jeder Verbindung abrufen können.

Die Webseite ist auch für die mobile Nutzung geeignet. Wenn jemand mit Zugang zur Seite ein besonders schönes Bild aufnimmt, kann er es sofort beschneiden und in eine Warteschlange stellen, um es an alle Bilderahmen da draußen zu verteilen (**Bild 10**).

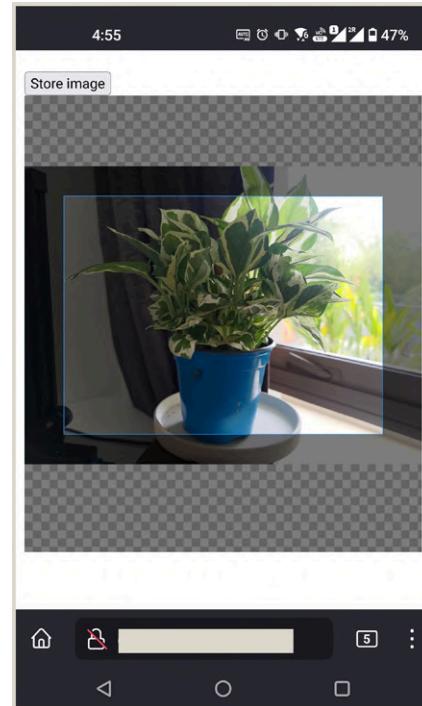


Bild 10. Schnelles Zuschneiden für die mobile Anwendung.

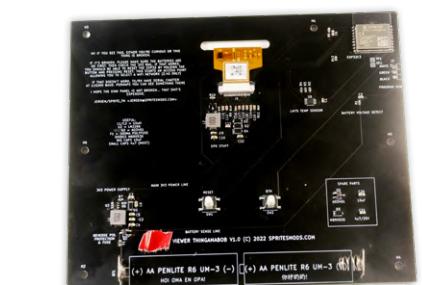


Bild 11. Die fertige bestückte Platine.



Bild 12. Die beiden Teile des für dieses Projekt entworfenen Gehäuses.

Ergebnis

Nachdem die Platine erstellt und die 3D-Teile gedruckt wurden, ist es an der Zeit, alles zusammenzubauen (**Bild 11**).

Auf der Rückseite der Platine befindet sich die gesamte Elektronik. Die Platine ist so konzipiert, dass sie repariert werden kann: Wenn sie kaputt geht und ich nicht da bin, um sie zu reparieren, habe ich vielleicht einen Freund oder zwei, die sich mit Elektronik auskennen und einen Blick darauf werfen können. Das bedeutet, dass sich auf der Rückseite einige Anweisungen zur Fehlersuche und sogar einige Ersatzkomponenten befinden, falls etwas kaputt geht. Ansonsten ist die Platine ziemlich spärlich bestückt: Ihre Abmessungen werden hauptsächlich durch die Größe des E-Ink-Displays auf der anderen Seite bestimmt. Ich hätte angesichts der Platinengröße auch größere Bauteile verwenden können, aber ich hatte jede Menge 0603-SMDs im Haus, also habe ich diese Mikroben verwendet.

Bild 13. Das Batteriefach mit seiner eigenen kleinen Klappe.



Bild 14. Das Endergebnis, dargestellt neben dem Original.



Mit einem guten Lötkolben und einem Binokularmikroskop konnte ich das alles problemlos von Hand löten. Das Gehäuse wird in **Bild 12** mit dem weißen Passepartout gezeigt. Dieses hat einen Ausschnitt für das E-Ink-Display: Selbst wenn sich der Klebstoff, mit dem es auf der Platine befestigt ist, lösen sollte, würde es dadurch an seinem Platz gehalten. Das Gehäuse wird mit Schrauben geschlossen, die in Gewindebuchsen aus Messing zum (vorsichtigen) Einschlagen geschraubt werden. Außer, man macht es wie vorgeschrieben, kann man die Gewindebuchsen auch fixieren, indem man sie mit einem Lötkolben erhitzt (mit einer alten, korrodierten Lötspitze) und in den Plastikrahmen einschmelzen lässt.

Die beiden Gehäuseteile werden mit einigen M3-Schrauben zusammengefügt. Das Batteriefach hat eine eigene kleine Klappe (**Bild 13**), so dass man nicht den ganzen Bilderrahmen auseinandernehmen muss, um die Batterien zu wechseln. Nach meinen Berechnungen sollte die Laufzeit der Batterien über ein Jahr betragen. Beachten Sie auch, dass es auf der Rückseite zwei Tasten gibt. Einer ist direkt mit dem ESP32-Reset verbunden, der andere mit einem Allzweck-GPIO. Mit dem Reset-Knopf kann man das Gerät dazu bringen, einen manuellen Verbindungs- und Aktualisierungszyklus durchzuführen, was den Nebeneffekt hat, dass das nächstfolgende Bild angezeigt wird. Wenn die andere Taste gedrückt und gehalten wird, während der Bilderrahmen zurückgesetzt wird, wird ein Zugangspunkt gestartet, über den Sie eine Verbindung zum Rahmen herstellen und die WLAN-Verbindung neu konfigurieren können.

In **Bild 14** schließlich können Sie das Endergebnis bewundern. Dieses Bild hat zwar nicht die beste Wiedergabetreue, aber die Tatsache, dass im Rahmen jeden Tag ein neues Foto vom anderen Ende der Welt zu sehen sein wird, macht dies mehr als wett.

Wie bei mir üblich, ist dieses Projekt Open Source: Mit einem 3D-Drucker, Zugang zu einem Server und etwas Geschick können Sie Ihre eigene Version dieses Bilderrahmens herstellen. Alle Quellen, Platinenvorlagen und weitere Unterlagen sind auf GitHub [8] zu finden. 

SG — 230464-02

Haben Sie Fragen oder Kommentare?

Wenn Sie technische Fragen oder Kommentare zu diesem Artikel haben, wenden Sie sich bitte an den Autor unter jeroen@spritesmods.com oder an das Elektor-Redaktionsteam unter redaktion@elektor.de.

Über den Autor

Jeroen Domburg ist leitender Software and Technical Marketing Manager bei Espressif Systems. Mit mehr als 20 Jahren Embedded-Erfahrung ist er sowohl am Software- als auch am Hardware-Designprozess von Espressifs-SoCs beteiligt. In seiner Freizeit tüftelt er gerne an Elektronik, um sowohl in der Praxis nützliche als auch weniger nützliche Geräte zu entwickeln.



Passende Produkte

- **Waveshare 5,65" ACeP 7-Farben E-Paper E-Ink Display Modul (600x448)**
www.elektor.de/19847
- **ESP32-DevKitC-32E**
www.elektor.de/20518
- **Dogan Ibrahim, *The Complete ESP32 Projects Guide*, Elektor 2019**
Buch, Paperback, englisch:
<https://www.elektor.de/the-complete-esp32-projects-guide>
E-Buch, PDF, englisch:
<https://www.elektor.de/the-complete-esp32-projects-guide-e-book>

WEBLINKS

- [1] 7-farbiger E-Paper Fotorahmen auf GitHub: <https://github.com/robertmoro/7ColorEPaperPhotoFrame>
- [2] Raspberry Pi E-Papier Bilderrahmen:
https://reddit.com/r/raspberry_pi/comments/10dbhnj/i_built_a_raspberry_pi_epaper_frame_that_shows_me
- [3] E-Paper-Fotoprojekt auf YouTube: <https://youtu.be/YawP9RjPcJA>
- [4] Entladungsdiagramme: <https://powerstream.com/AA-tests.htm>
- [5] ESP32-WiFi-Manager: <https://github.com/tonyp7/esp32-wifi-manager>
- [6] Cropper.js: <https://fengyuanchen.github.io/cropperjs>
- [7] CIEDE2000-Standard: https://en.wikipedia.org/wiki/Color_difference#CIEDE2000
- [8] GitHub-Repository: https://github.com/Spritetm/picframe_colepd



Tutorial ESP-Launchpad

Von Null auf Flash in wenigen Minuten

Von Dhaval Gujar, Espressif

Entwickeln Sie mit Espressif-Chips?
ESP-Launchpad ist ein webbasiertes Tool, das die Entwicklung und das Testen von Firmware vereinfacht. Werfen wir einen genaueren Blick darauf!

Dieser Artikel stellt ESP-Launchpad vor, ein webbasiertes Tool, das die Entwicklung und das Testen von Firmware für Espressif-Chips erleichtert. Es vereinfacht die Einrichtung der Entwicklungsumgebung, nutzt den Webbrowser zum Flashen der Firmware und ist für Entwickler und Nicht-Entwickler gleichermaßen von Vorteil.

Warum ESP-Launchpad?

Wenn Sie ein Open-Source-Projekt entwickeln, möchten Sie den Anwendern eine einfache Möglichkeit bieten, es zu untersuchen. Für Embedded-Entwickler bedeutet dieses Ziel das Einrichten des Entwicklungshosts und der Programmierwerkzeuge, das Klonen des Projekts, die Kompilierung und die anschließende Programmierung der Hardware.

Da Entwicklungshost und Softwareumgebung unterschiedlich sein können, gibt es viele Möglichkeiten, dass einer dieser Schritte schief geht. Und selbst wenn alles gut funktioniert, ist es für Benutzer, die keine Entwickler sind, sondern das Projekt nur ausprobieren wollen, zu viel Arbeit.

Wenn Sie mit einem der Espressif-Chips entwickeln, haben Sie jetzt eine bessere Möglichkeit, Ihr Projekt einfach zu entwickeln. ESP-Launchpad vereinfacht diesen Prozess und macht es schnell und problemlos möglich, für die ESP-Plattform entwickelte Firmware zu evaluieren und zu testen.

Auf die Plätze, fertig, ESP-Launchpad!

ESP-Launchpad ist eine webbasierte Anwendung, die die modernsten Browser-Funktionen nutzt, um auf einfache, schnörkellose Weise vorgefertigte Firmware auf ESP-Controller zu flashen. Traditionelle, dedizierte Software-Installationen für die Einrichtung von Entwickler-Hosts sind nicht mehr erforderlich; statt dessen wird die Zugänglichkeit einer Web-Plattform genutzt: über die serielle Webschnittstelle der Browser Chrome, Edge und Opera wird vom Browser aus mit der Entwicklungsplatine kommuniziert, programmiert und auf die serielle Konsole zugegriffen.

Schauen wir uns an, wie Sie Ihre Firmware mit ESP-Launchpad einfach starten können.

Vorbereiten der Firmware

Sobald Ihre Firmware fertig ist, besteht der nächste Schritt darin, Builds für alle Zielsysteme, die Sie unterstützen möchten, zu erstellen und sie in einzelne Binärdateien zu konvertieren. Keine Sorge, `esptool.py` (Espressifs zentrales Python-Dienstprogramm für alles, was mit dem Flashen zu tun hat) bietet eine praktische `merge_bin`-Option, mit der Sie eine solche Binärdatei ganz einfach erstellen können. Weitere Informationen dazu finden Sie auf unserer Dokumentationsseite [1]. Diese einzelnen Binärdateien sollten auf eine öffentlich zugängliche URL hochgeladen werden, die wir später verwenden werden.

Hinweis: In diesem Tutorial wird nicht näher darauf eingegangen, wie das erreicht werden kann.
Wissenswertes: Das Geheimnis hinter dem ESP-Launchpad ist eine JavaScript-Portierung von `esptool`, genannt `esptool-js`, die intern die WebSerial-API nutzt.



ESP-Launchpad ist eine webbasierte Anwendung, die die neuesten Browser-Funktionen zum einfachen, schnörkellosen Flashen vorgefertigter Firmware auf ESP-Controller nutzt.

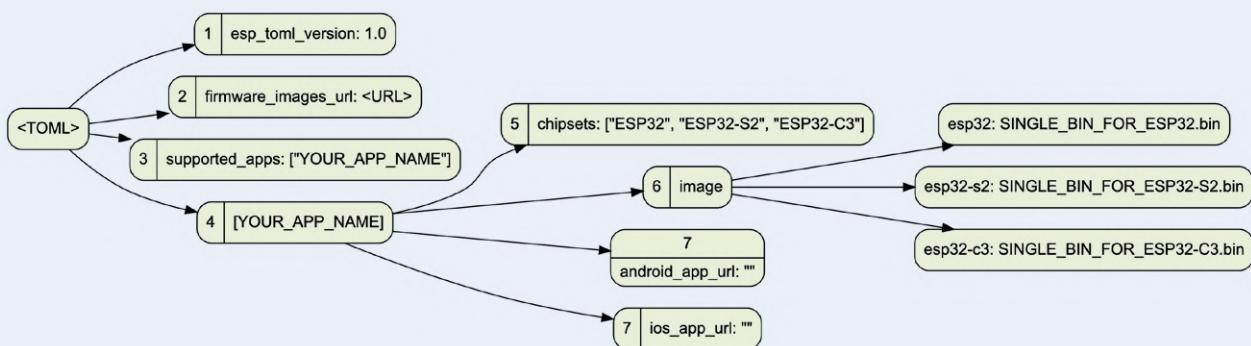


Bild 1. TOML-Strukturbau-Diagramm.

Verstehen der Grundlagen: Die Macht von TOML

Bevor wir in die Praxis eintauchen, sollten wir uns mit der TOML-Struktur der ESP-Launchpad-Konfiguration vertraut machen, einem zentralen Aspekt von ESP-Launchpad. TOML-Dateien dienen hierbei als einfache Konfigurationsentwürfe, die die Komponenten Ihrer Firmware, die unterstützte Hardware und die ergänzenden Apps detailliert beschreiben. Hier ist ein Beispiel:

```

esp_toml_version = 1.0
firmware_images_url = "<URL to your firmware images directory>"
supported_apps = ["YOUR_APP_NAME"]

[YOUR_APP_NAME]
chipsets = ["ESP32", "ESP32-S2", "ESP32-C3"]
image.esp32 = "SINGLE_BIN_FOR_ESP32.bin"
image.esp32-s2 = "SINGLE_BIN_FOR_ESP32-S2.bin"
image.esp32-c3 = "SINGLE_BIN_FOR_ESP32-C3.bin"
android_app_url = ""
ios_app_url = ""

```

Um dies besser zu verstehen, werfen Sie einen Blick auf Bild 1.

- `esp_toml_version` gibt die Version des TOML-Schemas an.
- `firmware_images_url` ist eine öffentlich zugängliche URL des Dateiservers, auf dem die Binärdateien Ihrer Firmware zum Download bereitstehen. Profi-Tipp: Wir hosten sowohl unsere TOML-Dateien und als auch Binärdateien auf GitHub, unter Verwendung von GitHub-Pages!
- `supported_apps` ist ein Array mit der Liste der Anwendungen, die Sie unterstützen und für die die Binärdateien verfügbar sind. Sie können mehrere Apps haben, die auf der Benutzeroberfläche von ESP-Launchpad in der Dropdown-Liste der verfügbaren Apps angezeigt werden. Die drei Werte, die wir uns gerade angesehen haben, waren die Schlüssel-Wert-Paare des Top-Levels.
- `[YOUR_APP_NAME]` - Dies ist eine Tabelle, die praktisch eine Sammlung von Schlüssel-Wert-Paaren ist. Sie enthält:
- `chipsets` - Ein Array mit einer Liste von ESP-Chipsätzen, für die Sie vorgefertigte Firmware bereitstellen werden. Beachten Sie die Namenskonvention hier, alles in Großbuchstaben, *ESP32-C3*.
- `image.<Chip-Name>` - Dies sind Schlüssel, die den Namen des Binärbildes mit jedem der unterstützten Zielen verbinden.

Dieser wird an die `firmware_images_url` angehängt, um die endgültige URL für die Binärdatei zu erhalten. Beachten Sie auch hier die Namenskonvention: Kleinschreibung, *esp32-c3*.

- `ios_app_url` und `android_app_url` sind optionale, aber spezielle Schlüssel-Wert-Paare unter der gleichen App-Tabelle, die es Ihnen ermöglichen, Links in Form von QR-Codes anzugeben, die die Nutzer einfach scannen können, nachdem Ihre App erfolgreich geflasht worden ist.

Schauen wir uns an, wie ESP-Launchpad aussieht (Bild 2), wenn wir die soeben erstellte Beispiel-TOML mit der folgenden imaginären URL bereitstellen:

https://espressif.github.io/esp-launchpad/?flashConfigURL=https://some-nice-url/my_app.toml

Okay, sieht gut aus! Wie geht es weiter?

Nur drei einfache Schritte für den Benutzer!

- Anschließen: Verbinden Sie den ESP mit dem seriellen USB-Anschluss Ihres Computers.
- Verbinden: Stellen Sie im Menü des Tools eine Verbindung mit dem Gerät her.
- Auswählen und Flashen: Wählen Sie die vorgefertigte Firmware aus und flashen Sie sie.

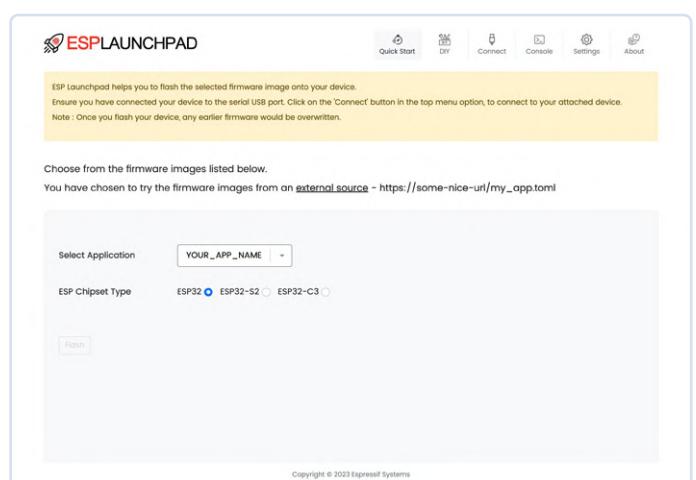


Bild 2. ESP-Launchpad mit geladener Beispielkonfiguration.

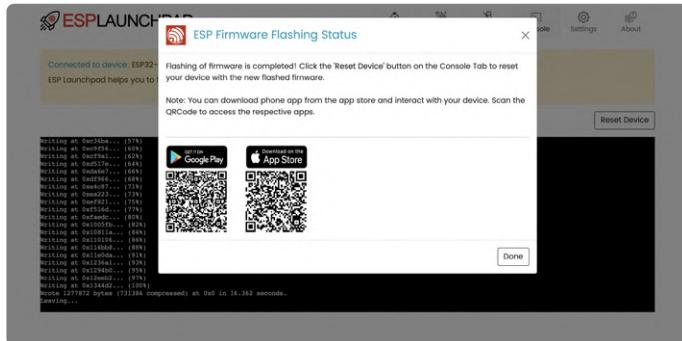


Bild 3. Abschied von ESP-Launchpad,

Nach dem Flashen wird der Benutzer von einer Konsole begrüßt, die das Flashprotokoll und ein Popup mit den QR-Codes der Handy-Apps anzeigt, sofern Sie diese zum TOML hinzugefügt haben (Bild 3). Herzlichen Glückwunsch, Sie haben soeben Ihre App mit ESP-Launchpad veröffentlicht! Profi-Tipp: Wir haben ein Badge (Bild 4) erstellt, das Sie in die *README* oder die Webseite Ihres Projekts einfügen können und das einen Hyperlink zur Flash-Konfigurations-URL Ihrer Anwendung enthält! Weitere Informationen finden Sie in der *README.md* des Projekts [2].

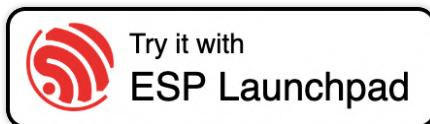


Bild 4: Ein Badge für Ihr Projekt!

ESP-Launchpad: Firmware trifft Community

Das Besondere am ESP-Launchpad ist die Möglichkeit, dass Nutzer ihre Firmware-Applikationen anderen zur Verfügung stellen können. In einer einfachen Konfigurationsdatei können Entwickler festlegen, woher die vorgefertigten Binärdateien ihrer Firmware stammen, welche Hardware unterstützt wird und sogar einen Link zu ergänzenden Apps

WEBLINKS

- [1] Grundlegende Befehle - Binärdateien zum Flashen:
merge_bin: <https://tinyurl.com/esp32mergebinaries>
- [2] Das GitHub-Repository des Projekts:
<https://github.com/espressif/esp-launchpad>



erstellen. Dieser ganzheitliche Ansatz stellt sicher, dass die Firmware nicht nur als Binärdatei weitergegeben wird, sondern als ein Erlebnis, das der Entwickler beabsichtigt.

Abschließende Überlegungen

Das wahre Potenzial eines jeden Tools kommt erst dann zum Tragen, wenn es in den Händen seiner Nutzer liegt. Erkunden Sie ESP-Launchpad, nehmen Sie es in Ihre Arbeitsabläufe auf und teilen Sie Ihre Erfahrungen! Ihre Erkenntnisse und Beiträge sind von unschätzbarem Wert, wenn wir unser Angebot weiter verbessern wollen. Lassen Sie uns gemeinsam die Erfahrung mit Open-Source-Evaluierungen neu definieren. 

RG-230596-02

Haben Sie Fragen oder Kommentare?

Wenn Sie technische Fragen oder Kommentare zu diesem Artikel haben, wenden Sie sich bitte an den Autor unter dhaval.gujar@espressif.com oder an die Elektor-Redaktion unter redaktion@elektor.de.

Über den Autor

Dhaval Gujar ist Ingenieur bei Espressif Systems, mit Schwerpunkt auf eingebetteten Systemen. Ihn treibt das dynamische Zusammenfließen von Technologien wie Cloud, IoT und mehr an. Dhaval interessiert sich leidenschaftlich für die sich entwickelnde Technologielandschaft und begeistert sich an moderne technologische Veränderungen.



Passende Produkte

- **ESP32-DevKitC-32E**
www.elektor.de/20518
- **LILYGO T-Display-S3 ESP32-S3 Entwicklungsboard (mit Headern)**
www.elektor.de/20299



Das ESP-NOW-Protokoll

Für flexible Kommunikation und Steuerung



Bei einigen Projekten möchten Sie vielleicht eine direkte Kommunikation von Gerät zu Gerät, ohne den Umweg über ein Infrastrukturnetz. Das ESP-NOW-Protokoll unterstützt eine Reichweite von mehr als 220 Metern in einer offenen Umgebung. ESP-NOW bietet eine Eins-zu-Eins- und Eins-zu-Viele-Gerätekommunikation und -steuerung. Dieses SDK enthält Beispiele dafür, wie das ESP-NOW-Protokoll für OTA, Gerätebereitstellung und -steuerung verwendet werden kann. Dieses SDK enthält auch eine Implementierung für einen mit Knopfzellen

betriebenen Schalter, der Geräte über das ESP-NOW-Protokoll steuern kann.

<https://github.com/espressif/esp-now>



ESP32 und ChatGPT



Auf dem Weg zu einem selbst-programmierenden System

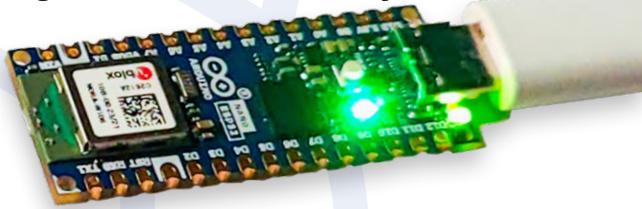
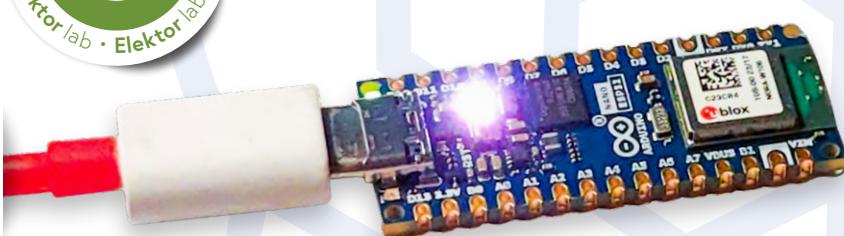


Bild 1. Die Arduino Nano ESP32 Boards, mit Arduino Framework (links) und MicroPython (rechts).

Von Saad Imtiaz (Elektor Lab)

Wir haben die Fähigkeiten von zwei Arduino-Nano-ESP32-Mikrocontrollern, drahtloser Kommunikation und der ChatGPT-API kombiniert, um ein intelligentes und interaktives Kommunikationssystem zu schaffen. Eines der Boards ist mit ChatGPT verbunden; Antworten und Code, die von diesem beliebten KI-Tool zurückgegeben werden, werden drahtlos an das andere Nano-Board übertragen. Wird sich dieses System aus zwei Boards mit Hilfe von ChatGPT selbst programmieren können? Folgen Sie der Schritt-für-Schritt-Anleitung und erfahren Sie, wie die Boards kommunizieren und wie die ChatGPT-API funktioniert.

Im Bereich der Embedded-Systeme und des Internets of Things (IoT) hat der ESP32-Mikrocontroller von Espressif aufgrund seiner Vielseitigkeit und robusten Fähigkeiten an Popularität gewonnen. Mit seiner integrierten WLAN-Konnektivität und leistungsstarken Verarbeitungsfunktionen eröffnet er eine Welt der Möglichkeiten für den Aufbau intelligenter und vernetzter Geräte. In diesem Artikel betrachten wir ein Demonstrationsprojekt, das die ChatGPT-API, ein von OpenAI entwickeltes KI-Sprachmodell, nutzt, um zwei Arduino-Nano-ESP32-Boards zu neuen Höchstleistungen zu bringen.

Das Ziel dieses Projekts ist es, ein nahtloses Kommunikationsverfahren zwischen den Nano-ESP32-Boards zu schaffen, von denen eines auf dem Arduino-IDE-Framework und das andere auf dem MicroPython-Framework läuft. Indem wir die ChatGPT-API nutzen, ermöglichen wir diesen Boards, geistreiche und interaktive Unterhaltungen

zu führen. Stellen Sie sich die Möglichkeiten vor, wenn Ihre Mikrocontroller Code-Snippets senden oder kreative Lösungen für Ihre Fragen anbieten.

Wir werden die technischen Details der Hardwareeinrichtung, der Softwarekonfiguration und der Kommunikation zwischen den Nano-ESP32-Boards erläutern. Außerdem werden wir die Vorteile und Grenzen der beiden Frameworks Arduino-IDE und MicroPython untersuchen und ihre einzigartigen Funktionen und Anwendungsfälle beleuchten. Darüber hinaus werden wir praktische Code-Snippets, Erklärungen und Einblicke zur Verfügung stellen, die Ihnen helfen, die Funktionsweise dieses Projekts zu verstehen.

Am Ende dieses Artikels wird deutlich, wie Sie Ihr eigenes Nano-ESP32-basiertes Kommunikationssystem bauen können, um die Macht von KI und IoT zu nutzen. Also, lassen Sie uns loslegen und die spannende Reise zur intelligenten und interaktiven Umgebung mit Nano-ESP32 und ChatGPT beginnen!

Die Hardware

Um dieses Projekt zu starten, benötigen wir einige Komponenten. Werfen wir einen genaueren Blick auf die Hardwareanforderungen:

1. Arduino Nano ESP32 Module (x2)

Das Herzstück unseres Projekts ist der ESP32-Mikrocontroller. Für die Kommunikation benötigen wir zwei Arduino-Nano-ESP32-Boards. Diese Boards sind gut verfügbar und bieten eine Reihe von Funktionen wie WLAN-Konnektivität, reichlich Rechenleistung und GPIO-Pins für die Verbindung mit externen Komponenten.

2. USB-Kabel Typ C (x2)

Die USB-Typ-C-Kabel werden für die Stromversorgung und Programmierung der Nano-ESP32-Boards benötigt. Mit diesen Kabeln können wir eine Verbindung zwischen den Boards und unserem Computer herstellen, was die Programmierung und Datenübertragung erleichtert.

3. WLAN

Ein stabiles WLAN mit Internetzugang ist für den Aufbau der Kommunikation zwischen den beiden Boards und der ChatGPT-API unerlässlich. Dies wird es dem ersten Nano-ESP32-Board (hier „Nano ESP32“)

genannt) ermöglichen, sich mit der ChatGPT-API zu verbinden und Nachrichten auszutauschen.

Wenn wir diese erforderlichen Hardwarekomponenten haben, können wir uns den Software- und Programmieraspekten des Projekts zuwenden.

Software und Programmierung

Um das Kommunikationssystem einzurichten und die Boards zu programmieren, benötigen wir die folgenden Software-Tools:

1. Arduino-IDE

Die integrierte Entwicklungsumgebung (IDE) von Arduino ist die am weitesten verbreitete IDE für die Programmierung von Mikrocontrollern mit Arduino-Firmware. Mit der neuen IDE 2.0 verfügt die Umgebung nun über eine benutzerfreundlichere Oberfläche, eine vereinfachte Programmiersprache und eine umfangreiche Bibliothek mit vorgefertigten Funktionen, die die Arbeit mit den Nano-ESP32-Boards erleichtern.

2. MicroPython Firmware

MicroPython ist eine abgespeckte Version der Programmiersprache Python, die für Mikrocontroller optimiert ist. Sie bietet im Vergleich zur Arduino-IDE eine flexiblere und interaktiveres Programmiererfahrung. Um den zweiten Nano ESP32 („ESP32-2“) mit MicroPython zu programmieren, müssen wir die MicroPython-Firmware auf dem Board installieren. In **Bild 1** sehen Sie die beiden angeschlossenen Boards. Das Board auf der linken Seite läuft mit dem Arduino-Framework und das Board auf der rechten Seite mit MicroPython.

3. Zugang zur ChatGPT-API

Um den ESP32-1 mit der ChatGPT-API zu verbinden, benötigen Sie Zugang zur API. Um Ihren API-Schlüssel zu erstellen, müssen Sie dem Link in [1] folgen und ein OpenAI-Konto erstellen. Nachdem Sie dem Link in [2] gefolgt sind, klicken Sie bitte *Create New Secret Key*. Sie können auch auf das Profilsymbol in der oberen rechten Ecke der Webseite klicken und *View API Keys* aus dem Dropdown-Menü auswählen. Sobald Sie den API-Schlüssel erstellt haben, bewahren Sie ihn an einem sicheren Ort auf, denn Sie können den Schlüssel nicht erneut kopieren.

4. Programmierung

Wir beginnen mit dem ESP32-1, der auf der Arduino-IDE läuft. Zuerst fügen wir die relevanten Bibliotheken unserem Code hinzu. Die Bibliothek *WiFi.h* wird benötigt, um eine Verbindung des Nano ESP32 mit dem WLAN-Netzwerk zu ermöglichen. Dann wird *HTTPClient.h* verwendet, um unsere Daten, also die ChatGPT-Antwort an das zweite Nano-ESP32-Board zu senden. Schließlich verwenden wir *Arduino-Json.h* (die Bibliothek, um JSON auf so ziemlich jedem Board am Markt einzusetzen). Sie ermöglicht uns die JSON-Serialisierung und JSON-Deserialisierung. Damit können wir auf die ChatGPT-API zugreifen, Aufforderungen senden und Antworten empfangen. Die Codes für beide Boards sind auch auf GitHub [3] verfügbar.

```
// Load Wi-Fi library
#include <WiFi.h>
#include <HTTPClient.h>
#include <ArduinoJson.h>
```

```
// Replace with your network credentials
const char* ssid = "WIFI_SSID";
const char* password = "WIFI_PWD";
//chatgpt api key
const char* apiKey = "sk-xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxx";
```

Jetzt fügen wir die IP-Adresse des ESP32-2 hinzu. Diese IP-Adresse wird im seriellen Monitor der IDE angezeigt, sobald das ESP32-2-Board mit dem WLAN verbunden ist.

```
// IP address of the second Nano ESP32
const char* serverIP = "192.168.1.82";
```



```
// Port number on the second Nano ESP32
const uint16_t serverPort = 80;
```

Listing 1 zeigt eine Funktion, die speziell für die Verbindung mit der ChatGPT-API und die Kommunikation mit ihr entwickelt wurde. Lassen Sie uns nun mit dem Code für das ESP32-2-Board beginnen. Zuvor wollen wir aber kurz erläutern, wie wir MicroPython auf dem Nano ESP32 ausführen. Wir werden *The Arduino Lab for MicroPython* verwenden. Um zu beginnen, gehen Sie zunächst zu [4] und laden das Arduino-MicroPython-Firmware-Tool herunter, um den Arduino-Nano-ESP32 zu flashen, wie in **Bild 2** unten gezeigt.

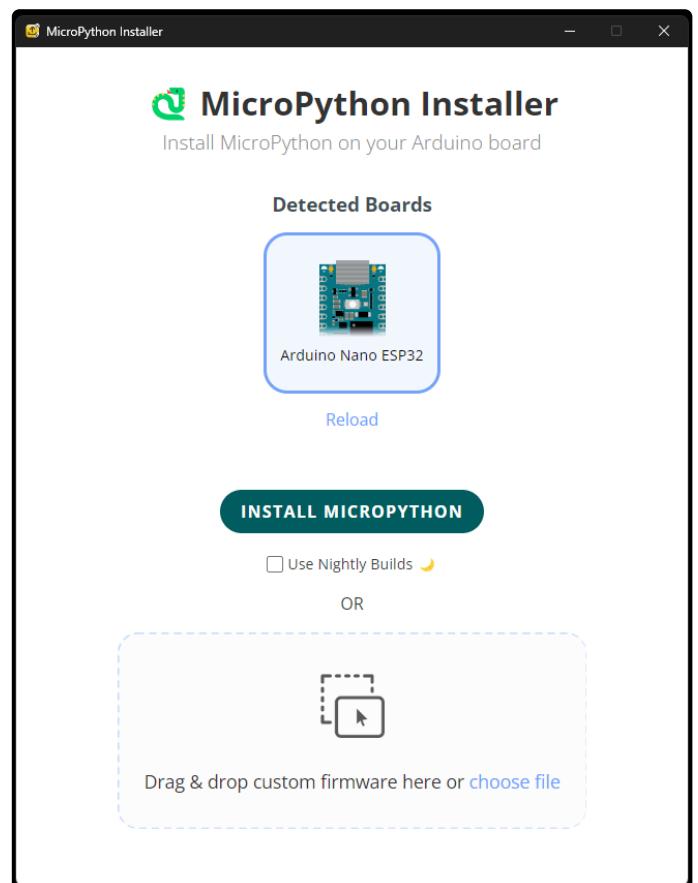


Bild 2. MicroPython Firmware Tool von Arduino Labs.



Listing 1. Arduino-Code, um Anfragen an ChatGPT zu senden

```
String sendChatGPTRequest(String prompt) {
    String received_response= "";
    String apiUrl = "https://api.openai.com/v1/completions";
    String payload = "{\"prompt\":\"" + prompt +
                    "\", \"max_tokens\":100, \"model\": \"text-davinci-003\"}";
    HttpClient http;
    http.begin(apiUrl);
    http.addHeader("Content-Type", "application/json");
    http.addHeader("Authorization", "Bearer " + String(apiKey));

    int httpResponseCode = http.POST(payload);
    if (httpResponseCode == 200) {
        String response = http.getString();

        // Parse JSON response
        DynamicJsonDocument jsonDoc(1024);
        deserializeJson(jsonDoc, response);
        String outputText = jsonDoc["choices"][0]["text"];
        received_response = outputText;
        //Serial.println(outputText);
    } else {
        Serial.printf("Error %i \n", httpResponseCode);
    }
    return received_response;
}
```

```
# Wait until connected to Wi-Fi
while not wifi.isconnected():
    pass
# Print the Wi-Fi connection details
print("Connected to Wi-Fi")
print("IP Address:", wifi.ifconfig()[0])
```

Schließen Sie einfach das Board an Ihren Computer an. Sobald es erkannt wird, klicken Sie auf *Install MicroPython*, und nach ein paar Sekunden ist Ihr Board mit der neuesten Firmware geflasht und Sie können MicroPython auf dem Nano ESP32 verwenden.

Jetzt werden wir die neueste IDE-Version von *Arduino Labs for MicroPython* von der offiziellen Website [5] herunterladen. Nach dem Entpacken der Datei starten Sie einfach die Software, indem Sie die Datei *Arduino Lab for MicroPython.exe* ausführen.

Auf dem ESP32-2 ist der Code schlicht und einfach. Wir importieren zunächst die Netzwerk- und Socket-Bibliotheken, um uns mit dem WLAN verbinden und dann unseren Server-Port erstellen zu können, an dem wir den Code oder die Antworten vom ESP32-1 empfangen können. Außerdem werden die Bibliotheken *machine* und *time* benötigt, um die GPIOs und die Timerfunktion des Nano ESP32 zu nutzen.

```
import network
import socket
import machine
import time

# Wi-Fi credentials
wifi_ssid = "WIFI_SSID"
wifi_password = "WIFI_PWD"

# Connect to Wi-Fi
wifi = network.WLAN(network.STA_IF)
wifi.active(True)
wifi.connect(wifi_ssid, wifi_password)
```

Dann wird ein Socket-Server erstellt, um Daten vom Nano ESP32-1 zu empfangen. Abschließend geht die MCU in eine while-Schleife, in der sie auf Antworten wartet. Wenn ein Code oder eine Antwort empfangen wird, führt der Nano ESP32 den Code aus und wartet dann erneut auf weitere Anweisungen (siehe Listing 2). Im Kasten **Arduino-IDE...** erörtern wir die Vor- und Nachteile der beiden Frameworks und erläutern, warum je nach Anwendungsfall ein Framework besser geeignet sein kann als das andere.

Kommunikation zwischen den ESP32-Boards

Jetzt wollen wir uns mit den technischen Details befassen, wie die beiden Nano ESP32-Boards in diesem Projekt über WLAN miteinander kommunizieren. Dieser Abschnitt enthält eine genauere Erklärung des Kommunikationsprozesses.

Nano ESP32-1 (mit Arduino-IDE-Framework)

Der Nano ESP32-1 mit dem Arduino-IDE-Framework stellt eine WLAN-Verbindung her, um mit externen Diensten und Geräten zu kommunizieren. Er verbindet sich mit einem bestimmten WLAN unter Verwendung der bereitgestellten Anmeldedaten. Sobald die Verbindung hergestellt ist, wartet Nano ESP32-1 auf Benutzereingaben auf dem seriellen Monitor der Arduino-IDE.

Wenn der Benutzer *GPT* eintippt und *Enter* drückt, fordert Nano ESP32-1 den Benutzer auf, eine Nachricht einzugeben. Diese wird dann vom Nano ESP32-1 zur Verarbeitung an die ChatGPT-API gesendet. Diese Übertragung erfolgt über die bestehende WLAN-Verbindung, wodurch ESP32-1 mit der externen API kommunizieren kann. In **Bild 3** sehen Sie die gesamte Kommunikations- und Verbindungsschleife zwischen den beiden Boards und der ChatGPT-API.

Die ChatGPT-API, eine Schnittstelle zum KI-Sprachmodell, empfängt die Nachricht vom ESP32-1. Mit hochentwickelten Techniken zur Verarbeitung natürlicher Sprache und maschinellen Lernalgorithmen analysiert die API die Eingabeaufforderung, um den Kontext zu verstehen und eine intelligente Antwort oder einen Code-Schnipsel zu generieren. Das ChatGPT-Modell innerhalb der API nutzt seine umfangreichen Trainingsdaten und Sprachverständnis-Fähigkeiten, um eine relevante und ansprechende Ausgabe zu liefern.

ESP32-1 empfängt den von der ChatGPT-API generierten Code-Snippet und legt ihn im Speicher ab. Dieses Code-Snippet ist die von der KI generierte Antwort auf die Eingabeaufforderung des Benutzers. ESP32-1 zeigt dann den empfangenen Code-Snippet auf dem seriellen Monitor an, so dass der Benutzer die von der KI generierten Anweisungen überprüfen kann. In **Bild 4** sehen Sie die Ausgabe des seriellen Monitors der beiden Boards. In diesem Fall sendet ESP32-1 den Code an ESP32-2, nachdem er die Antwort von ChatGPT erhalten hat.

Nano ESP32-2 (mit MicroPython-Framework)

ESP32-2 läuft mit dem MicroPython-Framework. ESP32-1 und ESP32-2 stellen mit den bereitgestellten Anmelddaten eine WLAN-Verbindung zum selben Netzwerk her. Dadurch kann ESP32-2 drahtlos Anweisungen von Nano ESP32-1 empfangen.

ESP32-2 stellt eine Socket-Verbindung her und lauscht auf einem bestimmten Port, normalerweise Port 80. Ein Socket ist ein Software-Endpunkt, der die Kommunikation zwischen zwei Geräten über ein Netzwerk ermöglicht. Indem ESP32-2 auf einem bestimmten Port lauscht, ist er bereit, eingehende Daten von ESP32-1 zu empfangen. Wenn ESP32-1 das generierte Code-Snippet senden möchte, stellt er über die Socket-Verbindung eine Verbindung zu ESP32-2 her. Das Code-Snippet wird dann an ESP32-2 übertragen, wo es empfangen und verarbeitet wird.

ESP32-2 verarbeitet das empfangene Code-Snippet und extrahiert die darin eingebetteten Anweisungen. Diese Anweisungen definieren dann spezifische Aufgaben, die von ESP32-2 ausgeführt werden sollen. Zum Beispiel könnte das Code-Snippet ESP32-2 anweisen, eine LED für eine bestimmte Dauer blinken zu lassen oder andere Aktionen (basierend auf der KI-generierten Antwort) auszuführen.



Listing 2. Empfang und Ausführung von Code-Snippets auf dem ESP32-2 (Python)

```
# Create a socket server
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.bind(('', 80))
server.listen(1)

# Accept and handle incoming connections
while True:
    print("Waiting for connection...")
    client, addr = server.accept()
    print("Client connected:", addr)

    # Receive the code snippet from the first ESP32
    code = ""
    while True:
        data = client.recv(1024)
        if not data:
            break
        code += data.decode()

    # Execute the received code
    try:
        exec(code)
        response = "Code executed successfully"
        print(response)
    except Exception as e:
        response = "Error executing code: " + str(e)

    # Send the response back to the first ESP32
    client.sendall(response.encode())
    # Close the connection
    client.close()
    print("Client disconnected")
```

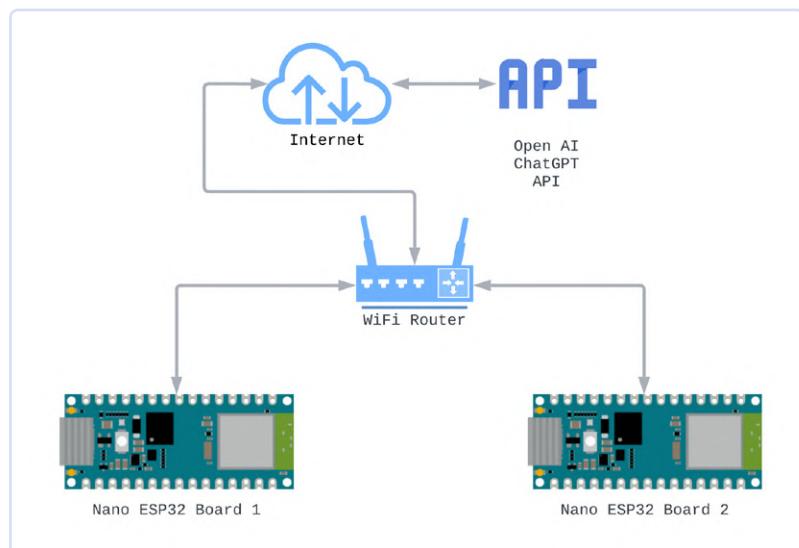


Bild 3. Die Kommunikation zwischen den zwei Arduino Nano ESP32 Boards und der API von ChatGPT.

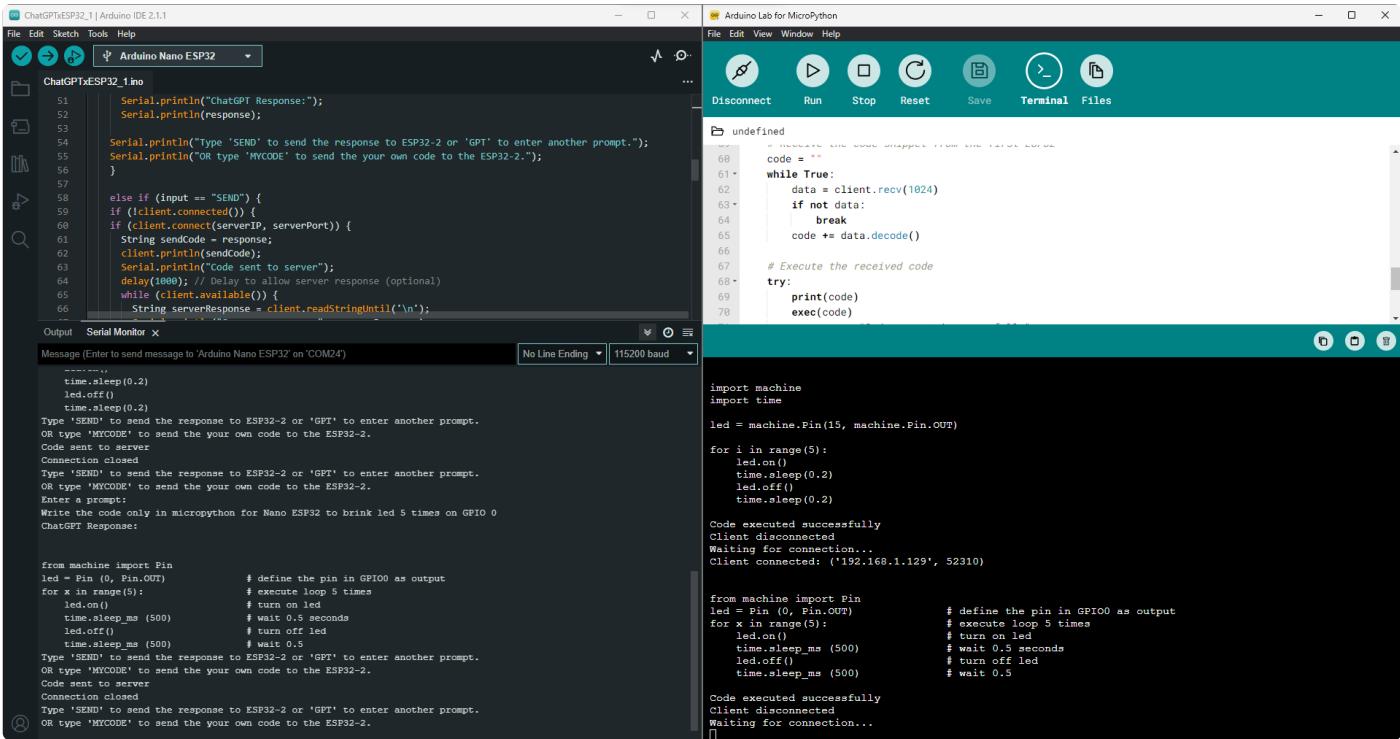


Bild 4. Die Kommunikation der beiden Nano ESP32 mit der Arduino-IDE (links) und Arduino Lab / MicroPython (rechts).

Nach der Ausführung sendet ESP32-2 über die aufgebaute Socket-Verbindung eine Antwort an ESP32-1 zurück. Diese Antwort dient als Bestätigung, dass das empfangene Code-Snippet erfolgreich ausgeführt wurde. So ist ESP32-1 über den Abschluss der angeforderten Aufgabe informiert.

Durch den hier beschriebenen Kommunikationsablauf können die beiden Nano-ESP32-Boards Nachrichten austauschen und effektiv zusammenarbeiten. ESP32-1 interagiert mit der ChatGPT-API und nutzt deren KI-Fähigkeiten, während ESP32-2 als Empfänger und Ausführer der KI-generierten Anweisungen agiert.

Die Grenzen von ChatGPT bei der Programmierung

ChatGPT ist zwar ein bemerkenswertes KI-Sprachmodell, aber es hat gewisse Einschränkungen, vor allem, wenn es darum geht, funktionalen Code in komplexeren Programmierszenarien zu erzeugen. Bei Tests wurde festgestellt, dass ChatGPT nur in etwa 60 % der Fälle funktionalen Code für einen einfachen Blink-Sketch liefern konnte. Dies

zeigt, dass die Fähigkeit des Modells, akkurate und funktionierende Code-Snippets zu erzeugen, nicht immer gewährleistet ist. Eine besondere Herausforderung kann sich ergeben, wenn Code-Snippets von ChatGPT angefordert werden. In einigen Fällen enthält die Antwort zusätzlichen Text vor und nach dem eigentlichen Code. Das kann zu Problemen führen, wenn Sie versuchen, die gesamte Antwort an das zweite Nano-ESP32-Board zu senden. In Bild 5 sehen Sie die Antwort von ChatGPT, nachdem Sie es aufgefordert haben, einen Code zur Überprüfung des Sensorwerts an Pin 36 des ESP32 zu liefern. Um diese Beschränkung zu umgehen, wurde eine zusätzliche Funktion in den Code implementiert, die es dem Benutzer ermöglicht, den Code selbst einzugeben und ChatGPT als Referenz oder Anleitung zur Optimierung und Reduzierung der Codezeilen zu verwenden. Dieser Ansatz ermöglicht eine größere Kontrolle über den generierten Code und behebt die Formatierungs- und Syntaxprobleme, die auftreten, wenn man sich ausschließlich auf die Ausgabe von ChatGPT verlässt. Im Zuge der kontinuierlichen Entwicklung von KI-Sprachmodellen können wir davon ausgehen, dass ihre Fähigkeit, präziseren und zuver-

```

python

from machine import Pin, ADC
import time

# Define the sensor pin
sensor_pin = ADC(Pin(36))

```

Bild 5. Antwort von ChatGPT nach der Aufforderung, ein Code-Snippet zu schreiben.

Anwendungen und Einsatzmöglichkeiten

Das Projekt der Integration von Nano ESP32 Boards mit der ChatGPT API eröffnet eine Reihe von spannenden Anwendungen. Hier sind ein paar bemerkenswerte Beispiele:

- Intelligente Hausautomation: Durch die Nutzung der ChatGPT-API können Benutzer mit ihren Smart-Home-Systemen über natürliche-sprachige Eingabeaufforderungen interagieren. Sie können Leuchten steuern, Temperatureinstellungen vornehmen oder sogar um Empfehlungen für energiesparende Maßnahmen bitten.
- Prototyping und schnelle Entwicklung: Die Kombination aus Nano-ESP32-Boards und KI-generierten Codeschnipseln ermöglicht ein schnelles Prototyping von IoT-Projekten. Benutzer können Ideen schnell testen und iterieren, indem sie Code-Vorschläge für verschiedene Funktionalitäten erhalten.
- Pädagogisches Werkzeug: Das Projekt ist ein wertvolles Lehrmittel für die kommende Programmierer-Generation. Die Schüler können interaktive Gespräche über das Programmieren mit der KI führen, Anleitungen erhalten und neue Programmietechniken erlernen.
- Persönlicher Assistent und Informationsbeschaffung: Die ChatGPT-Integration kann zur Entwicklung eines persönlichen Assistenten verwendet werden. Benutzer können Fragen stellen, Empfehlungen einholen oder Informationen zu verschiedenen Themen abrufen - alles mit Hilfe von KI-gesteuerten Antworten.
- Kreative Ideen für das Coding: Das Projekt dient als Inspirationsquelle für kreatives Programmieren. Es kann einzigartige und innovative Ideen für Animationen, Visualisierungen oder interaktive Projekte hervorbringen und so die Kreativität der Entwickler anregen.

lässigeren Code zu generieren, immer weiter verbessert wird. Diese Fortschritte werden neue Horizonte für die Nutzung von KI in der Programmierung eröffnen und eine noch engere Zusammenarbeit zwischen Menschen und Maschinen ermöglichen. Um das Beste aus ChatGPT herauszuholen, ist es wichtig, es als Werkzeug zu nutzen und sich nicht ausschließlich auf die Ergebnisse zu verlassen. Die Kombination von menschlichem Fachwissen und Urteilsvermögen mit den KI-generierten Antworten kann zu genaueren und zuverlässigeren Ergebnissen führen. Wenn die Benutzer diese Einschränkungen verstehen und berücksichtigen, können sie ChatGPT effektiv nutzen und gleichzeitig potenzielle Risiken und Herausforderungen abmildern.

Welche alternativen Ansätze gibt es?

Zusätzlich zum hier beschriebenen Projektaufbau gibt es alternative Methoden und Ansätze, die für die Nano-ESP32-Kommunikation und KI-Integration verwendet werden können. Lassen Sie uns ein paar andere Möglichkeiten erkunden, wie dieses Projekt auf der Nano-ESP32-Plattform implementiert werden kann:

1. MQTT-Protokoll

Das MQTT-Protokoll (Message Queuing Telemetry Transport) ist ein schlankes und effizientes Messaging-Protokoll, das häufig in IoT-Anwendungen verwendet wird. Anstatt sich auf WLAN und Socket-Verbindungen zu verlassen, können Nano-ESP32-Boards über das MQTT-Protokoll miteinander kommunizieren. MQTT-Broker können eingerichtet werden, um den Nachrichtenaustausch zwischen den Boards zu erleichtern, was wiederum einen nahtlosen Datenaustausch und eine KI-Integration ermöglicht.

2. Direct WebSocket Connection

Ein anderer Ansatz ist der Aufbau einer direkten WebSocket-Verbindung zwischen den Nano-ESP32-Boards. WebSocket ist ein Kommunikationsprotokoll, das Voll duplex-Kommunikation über eine einzige TCP-Verbindung ermöglicht. Durch die Implementierung von WebSocket auf den Nano-ESP32-Boards kann eine dauerhafte und bidirektionale Verbindung hergestellt werden, die Echtzeitkommunikation und KI-Integration ermöglicht.

3. ESP-NOW-Protokoll

ESP-NOW ist ein Kommunikationsprotokoll, das speziell für Low-Power-Geräte entwickelt wurde und eine schnelle und zuverlässige Daten-

übertragung zwischen Nano-ESP32-Boards ermöglicht. Dieses Protokoll kann genutzt werden, um eine direkte Kommunikation zwischen den Boards herzustellen und so die Notwendigkeit von WLAN-Verbindungen umgehen. Durch die Integration von KI-Fähigkeiten auf einem Board und die Verwendung von ESP-NOW für die Kommunikation können Echtzeitantworten und Code-Snippets effizient ausgetauscht werden.

4. KI-Integration auf dem Nano ESP32

Anstatt sich auf externe APIs zu verlassen, können KI-Modelle direkt auf den Nano-ESP32-Boards eingesetzt werden. Mit TensorFlow Lite zum Beispiel können KI-Modelle lokal auf Mikrocontrollern eingesetzt und ausgeführt werden. Durch das Training beziehungsweise die Feinabstimmung eines Modells für bestimmte Aufgaben wie die Generierung von Code-Snippets können die Nano-ESP32-Boards KI-gesteuerte Antworten liefern, ohne dass externe Verbindungen erforderlich sind.

Diese alternativen Ansätze bieten je nach Projektanforderungen (aber auch Beschränkungen) sowohl Vor- als auch Nachteile. Faktoren wie Stromaufnahme, Latenz, Komplexität und Skalierbarkeit sollten daher bei der Wahl des optimalen Ansatzes berücksichtigt werden.

Durch die Erforschung dieser alternativen Methoden können Entwickler die Fähigkeiten der Nano-ESP32-Boards erweitern, KI auf verschiedenen Ebenen integrieren und das jeweilige Kommunikationssystem an ihre Bedürfnisse anpassen.

Das Projekt der Kommunikation und KI-Integration des Nano ESP32 ist nicht auf einen einzigen Ansatz beschränkt. Durch die Berücksichtigung alternativer Methoden wie MQTT, WebSocket, ESP-NOW oder On-Device-KI-Integration können Entwickler das Projekt an ihre individuellen Anforderungen anpassen und das volle Potenzial der Nano-ESP32-Plattform nutzen. Lassen Sie Ihrer Kreativität freien Lauf, experimentieren Sie mit verschiedenen Ansätzen und entwickeln Sie innovative IoT-Systeme, die KI und der Nano ESP32 auf aufregende Weise kombinieren!

Die Schnittmenge von KI und IoT

Die Integration von Nano-ESP32-Boards mit der ChatGPT-API stellt eine faszinierende Schnittmenge von KI, IoT und Programmierung dar. Indem wir Nano-ESP32-Boards in die Lage versetzen, mit einem KI-Sprachmodell zu kommunizieren und zu interagieren, erschließen wir eine ganze Welt von neuen Möglichkeiten. Von der intelligenten

Arduino IDE, MicroPython und Andere

Arduino-IDE-Framework

Die Arduino-IDE ist aufgrund ihrer Einfachheit und Benutzerfreundlichkeit eine beliebte Wahl sowohl für Anfänger als auch für Maker. Sie bietet eine anfängerfreundliche Programmierumgebung mit einer vereinfachten Version der Programmiersprache C++. Hier sind einige Vor- und Nachteile der Verwendung des Arduino-IDE-Frameworks mit Nano ESP32:

Vorteile

- Leicht zu erlernen: Die Arduino-IDE bietet eine sanfte Lernkurve, so dass sie auch für Neulinge im Bereich Programmierung oder Mikrocontroller leicht zugänglich ist.
- Große Gemeinschaft und Bibliotheksunterstützung: Arduino verfügt über eine große Community von Enthusiasten, umfangreiche Online-Ressourcen und eine große Auswahl an Bibliotheken und Code-Beispielen, die die Entwicklung vereinfachen.
- Schnelles Prototyping: Die Arduino-IDE ermöglicht dank ihrer vereinfachten Syntax und Bibliotheksunterstützung ein schnelles Prototyping und damit schnellere Entwicklungszyklen.

Nachteile

- Eingeschränkte Sprachfunktionen: Die Arduino-IDE verfügt über eine vereinfachte Version von C++, was die Nutzung fortgeschrittenen Programmierungsfunktionen im Vergleich zu anderen Sprachen einschränken kann.
- Speichernutzung und Leistung: Die Arduino-IDE ist im Vergleich zu anderen Frameworks möglicherweise nicht so effizient in Bezug auf Speichernutzung und Leistung.
- Weniger Flexibilität: Die Arduino-IDE erlegt bestimmte Konventionen und Einschränkungen auf, die das volle Potenzial des Nano ESP32 einschränken können.

MicroPython-Framework

MicroPython ist eine schlankere Implementierung der Programmiersprache Python, die für Mikrocontroller entwickelt wurde. Sie bietet eine flexiblere und interaktive Programmiererfahrung. Hier sind einige Vor- und Nachteile bei der Verwendung des MicroPython Frameworks mit Nano ESP32:

Vorteile

- Python-Sprache: MicroPython ermöglicht es Entwicklern, die leistungs- und ausdrucksstarke Python-Sprache zu nutzen, was das Schreiben von komplexem Code und Algorithmen erleichtert.
- Interaktive REPL: MicroPython bietet eine REPL-Umgebung (Read-Eval-Print Loop), die es Entwicklern ermöglicht, Code direkt auf dem Nano-ESP32-Board interaktiv zu testen und damit zu experimentieren.
- Effiziente Speicherverwendung: MicroPython wurde im Hinblick auf Speichereffizienz entwickelt und eignet sich daher für ressourcenbeschränkte Geräte wie den Nano ESP32.

Nachteile

- Lernkurve: MicroPython kann eine steilere Lernkurve für Anfänger, die nicht mit der Sprache Python vertraut sind, haben.
- Begrenzte Bibliotheksunterstützung: Obwohl MicroPython über eine wachsende Sammlung von Bibliotheken verfügt, kann es im Vergleich zum umfangreichen Arduino-Bibliotheks-Ökosystem weniger Optionen bieten.
- Abstriche bei der Leistung: Während MicroPython Flexibilität bietet, kann die interpretierte Natur der Sprache zu einer etwas langsameren Ausführung im Vergleich zu kompilierten Sprachen wie C++ führen.

Hausautomatisierung über Rapid Prototyping bis hin zu Bildungswerkzeugen - die Anwendungsmöglichkeiten sind vielfältig (siehe Kasten **Anwendungen und Einsatzmöglichkeiten**).

Es ist jedoch wichtig, die Grenzen von ChatGPT im Auge zu behalten und bei der Arbeit mit KI-generierten Antworten Vorsicht walten zu lassen, insbesondere in Programmier- und Codierungsszenarien. Die Kombination von menschlichem Fachwissen und Urteilsvermögen mit KI-generierten Inhalten wird hier sicherlich zu den besten Ergebnissen führen.

Mit diesem Projekt können Benutzer ihre Kreativität entfesseln, neue Horizonte in der IoT-Entwicklung erkunden und ihre Programmierungsfähigkeiten verbessern. Schnappen Sie sich also Ihre Nano-ESP32-Bretter, tauchen Sie ein in die Welt der KI-gesteuerten Interaktionen und nutzen Sie das grenzenlose Potenzial dieser spannenden Integration! ↗

Übersetzung von Holger Neumann - 230485-02

Haben Sie Fragen oder Anmerkungen?

Wenn Sie technische Fragen oder Kommentare zu diesem Artikel haben, wenden Sie sich bitte an die Elektor-Redaktion unter redaktion@elektor.de.

Über den Autor

Saad Imtiaz ist Ingenieur mit Erfahrung in den Bereichen Embedded Systems, mechatronische Systeme und Produktentwicklung. Er hat mit mehr als 200 Unternehmen - von Start-ups bis hin zu Weltkonzernen - bei der Entwicklung von Produktprototypen zusammengearbeitet. Er war außerdem in der Luftfahrtindustrie tätig und hat ein Technologie-Startup-Unternehmen geleitet. Seit 2023 ist er bei Elektor tätig und treibt die Projektentwicklung in den Bereichen Software und Hardware voran.

Andere Frameworks

Einer der herausragenden Vorteile des Nano ESP32 ist seine Flexibilität, mit verschiedenen Frameworks zu arbeiten. Neben der Arduino-IDE und MicroPython gibt es PlatformIO, ESP-IDF, JavaScript (Node.js) und Lua-Frameworks. Der Nano ESP32 bietet also Kompatibilität mit mehreren anderen Frameworks, was seine Vielseitigkeit weiter erhöht. Diese Flexibilität ermöglicht es Entwicklern, das Framework zu wählen, das am besten zu ihren Projektanforderungen und ihrer Vertrautheit mit Programmiersprachen passt.

Lassen Sie uns ein paar andere Frameworks betrachten, die ebenfalls für die Verwendung mit dem Nano ESP32 in Betracht kommen:

Mongoose OS: Mongoose OS ist ein Open-Source-Betriebssystem für Mikrocontroller, einschließlich des Nano ESP32. Es bietet eine plattformunabhängige Umgebung mit integrierter Cloud-Konnektivität und unterstützt mehrere Programmiersprachen wie JavaScript, C und C++. Mongoose OS vereinfacht den Entwicklungsprozess durch eine einfach zu bedienende Befehlszeilenschnittstelle, umfangreiche Bibliotheken und Funktionen zur Fernverwaltung von Geräten.

Zephyr RTOS: Zephyr RTOS ist ein Echtzeit-Betriebssystem, das für ressourcenbeschränkte Systeme wie den Nano ESP32 entwickelt wurde. Es bietet eine skalierbare und modulare Architektur, die es für Projekte geeignet macht, die Multitasking, Echtzeitverarbeitung und fortschrittliches Gerätemanagement erfordern. Zephrys breite Hardware-Unterstützung und der umfangreiche Satz an Treibern und Bibliotheken ermöglichen es Entwicklern, komplexe IoT-Anwendungen mit geringem Aufwand zu erstellen.

ESPHome: ESPHome wurde für ESP32- und ESP8266 entwickelt und ist ein vielseitiges IoT-Framework mit einer modularen Architektur, die dem Zephyr RTOS ähnelt. Es bietet ebenfalls eine breite Hardware-Unterstützung, eine Reihe von Treibern und Bibliotheken für eine vereinfachte Entwicklung von Systemen mit eingeschränkten Ressourcen. Dieses Framework ermöglicht Echtzeitverarbeitung, Multitasking und fortschrittliches Gerätemanagement, was es zu einer ausgezeichneten Wahl für IoT-Projekte in der Heimautomatisierung macht.

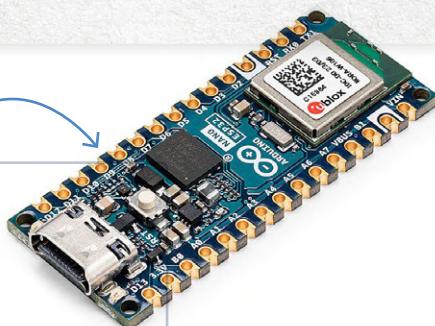
Die Wahl des richtigen Frameworks hängt von Faktoren wie den Projektanforderungen, der Kenntnis der Programmiersprache, den verfügbaren Bibliotheken und der Unterstützung durch die Community sowie dem angestrebten Grad der Kontrolle über Hardware und Leistung ab. Jedes Framework bringt eine Reihe von Vorteilen und Nachteilen mit sich, so dass die Entwickler das für ihre spezifischen Anwendungsfälle am besten geeignete auswählen können.

Dank der Kompatibilität des Nano ESP32 mit verschiedenen Frameworks haben Entwickler die Freiheit, verschiedene Programmiersprachen, Umgebungen und Entwicklungsansätze zu erkunden. Diese Flexibilität ermöglicht es ihnen, innovative IoT-Lösungen zu erstellen, bestehende Tools und Bibliotheken zu nutzen und die Fähigkeiten des Nano ESP32 effizient einzusetzen.



Passende Produkte:

- **Arduino Nano ESP32** www.elektor.de/20562
- **Arduino Nano ESP32 mit Headern** www.elektor.de/20529
- **Bartmann, Erik, Das ESP32-Praxisbuch (Elektor 2018)**
Buch, kartoniert, deutsch: www.elektor.de/18572
E-Buch, PDF, deutsch: www.elektor.de/das-esp32-praxisbuch-pdf
- **Günter Spanner, MicroPython für Mikrocontroller (Elektor 2019)**
Buch, kartoniert, deutsch: www.elektor.com/19412
E-Buch, PDF, deutsch: www.elektor.de/micropython-für-mikrocontroller-pdf



WEBLINKS

- [1] OpenAI: <https://platform.openai.com>
- [2] OpenAI - API-Keys:
<https://platform.openai.com/account/api-keys>
- [3] Projekt-Repository auf GitHub :
<https://github.com/elektor-labs/ChatGPTxESP32>
- [4] Arduino Labs - MicroPython-Installer:
<https://labs.arduino.cc/en/labs/micropython-installer>
- [5] Arduino Labs - MicroPython-IDE:
<https://labs.arduino.cc/en/labs/micropython>



Walkie-Talkie mit ESP-NOW

Nicht ganz WLAN, nicht ganz Bluetooth, aber...

Von Clemens Valens (Elektor)

Stellen Sie sich vor, Ihr drahtloses Projekt benötigt sowohl schnelle Reaktionszeiten als auch eine große Reichweite? WLAN und Bluetooth sind für solche Anwendungen ungeeignet. Vielleicht ist ESP-NOW eine gute Alternative? Verbindungen werden fast sofort hergestellt, und es sind Reichweiten von mehreren hundert Metern möglich. In diesem Artikel probieren wir es in einer einfachen drahtlosen Intercom-Anwendung vulgo Walkie-Talkie aus.

Der ESP32 von Espressif wird meist wegen seiner WLAN- und Bluetooth-Fähigkeiten verwendet. Wi-Fi und Bluetooth sind zwar großartige Protokolle für alle Arten von drahtlosen Anwendungen, aber sie haben doch ihre Grenzen.

Ein Nachteil von Wi-Fi ist die Zeit, die benötigt wird, um eine Verbindung herzustellen. Außerdem ist bei Wi-Fi keine direkte Kommunikation (Peer-to-Peer, **Bild 1**) zwischen Geräten möglich. Es ist immer ein Router beteiligt. Aus diesem Grund eignet sich Wi-Fi nicht wirklich für einfache Fernbedienungen mit geringer Latenzzeit, zum Beispiel,

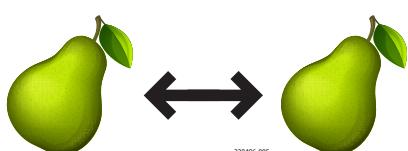


Bild 1. Eine Peer-to-Peer-Kommunikation, wie sie hier gezeigt wird, ist mit Wi-Fi nicht möglich, da immer ein Router zwischen den beiden Knotenpunkten benötigt wird.

um ein Garagentor zu öffnen oder eine Lampe ein- und auszuschalten. Solche Aufgaben erfordern nämlich eine sofortige Reaktion. Um diesen Makel zu umgehen, sind WLAN-Anwendungen in der Regel ständig eingeschaltet und verbunden. Doch dabei benötigen sie viel Energie, selbst wenn sie im Leerlauf sind.

Bluetooth hingegen zeichnet sich durch einen schnellen Verbindungsauflauf und Peer-to-Peer-Kommunikation aus. Es wäre damit hervorragend für Fernbedienungen mit geringer Latenz geeignet, wenn da nicht die geringe Reichweite wäre: Bluetooth funktioniert nur, wenn die kommunizierenden Geräte nicht weiter als etwa zehn Meter voneinander entfernt sind. Es gibt zwar Bluetooth für größere Entfernung, aber es ist noch nicht weit verbreitet.

Die Lösung: ESP-NOW

Das drahtlose Protokoll ESP-NOW [1] von Espressif ist eine Lösung für Situationen, in denen sowohl schnelle Reaktionszeiten als auch eine große Reichweite erforderlich sind und das gleiche Frequenzband wie bei Wi-Fi oder Bluetooth verwendet werden soll.

Das Protokoll vereint die Vorteile von Wi-Fi und Bluetooth. ESP-NOW zielt damit auf die Hausautomatisierung und das intelligente Haus ab. Da es One-to-Many- und Many-to-Many-Topologien ermöglicht

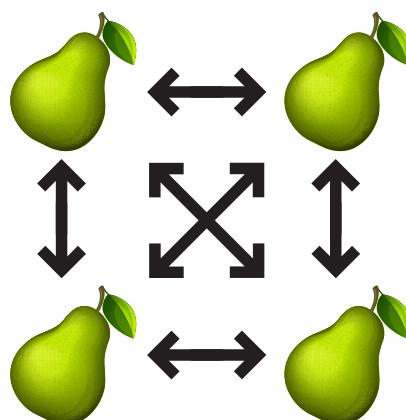


Bild 2. ESP-NOW unterstützt Many-to-Many-Netzwerke, in denen jeder Knoten direkt mit den anderen Knoten kommunizieren kann, ohne dass ein Router erforderlich wäre.

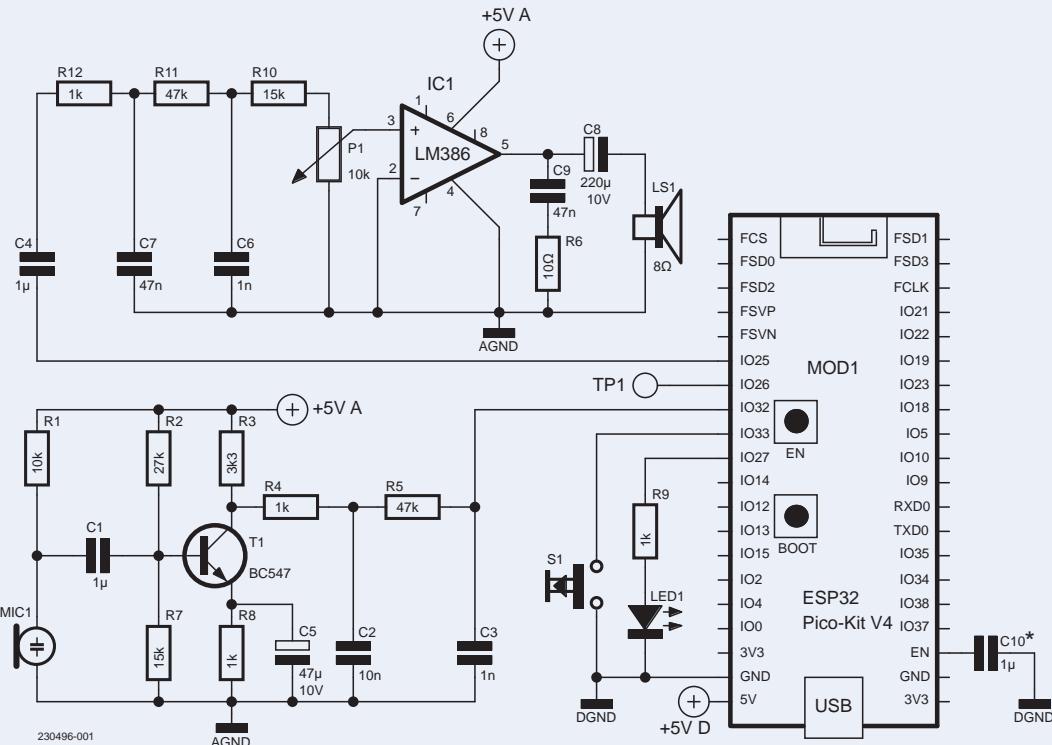


Bild 3. Ein einfacher Mikrofonverstärker am Eingang und ein klassischer LM386 als Leistungsverstärker am Ausgang. Die Stromversorgung für den analogen und den digitalen Teil sind voneinander getrennt.

(Bild 2), benötigt es keine Router, Gateways oder gar eine Cloud. ESP-NOW nutzt dafür keine ausgefallenen Verbindungs- oder High-Level-Kommunikationsprotokolle. Die Adressierung basiert auf der Ethernet-MAC-Adresse des Knotens, und es ist nur ein Pairing-Schritt erforderlich, damit die Knoten miteinander kommunizieren können. Außerdem wird nicht garantiert, dass die Datenpakete in der richtigen Reihenfolge ankommen. Für einfache Fernsteuerungsanwendungen ist das alles aber kein Problem.

Die Datenrate von ESP-NOW beträgt standardmäßig 1 Mbit/s (konfigurierbar), und ein Datenpaket kann eine Nutzlast von bis zu 250 Byte haben. Zusammen mit den Header- und Prüfsummenbytes und so weiter ergibt dies eine maximale Paketgröße von 255 Byte.

Bauen wir ein Walkie-Talkie!

Mein Ziel war es, eine Art Walkie-Talkie oder eine Gegensprechanlage auf der Grundlage von ESP-NOW zu bauen. Ein kurzer Blick auf die Spezifikationen des ESP32 zeigt, dass er über alles verfügt, was man dafür braucht: einen Analog-Digital-Wandler (ADC), einen Digital-Analog-Wandler (DAC), viel Rechenleistung und natürlich den ganzen Funkkram. In der Praxis sieht es allerdings nicht ganz so rosig aus. Der 12-Bit breite ADC erweist sich als ziemlich langsam, ich habe eine maximale Abtastrate von etwa 20 kHz gemessen. Irgendwo im Internet wurde erwähnt, dass seine analoge Bandbreite nur 6 kHz beträgt. Der DAC ist acht Bit breit (aber es gibt zwei davon), was die mögliche Audioqualität noch mehr einschränkt.

Ein Walkie-Talkie kann jedoch mit diesen Zahlen auskommen, wenn das Audiosignal auf die Standard-Telefonie-Bandbreite von 3,5 kHz gestutzt wird. Bei einer Abtastrate von 8 kHz ergibt sich eine Datenrate von $(8.000/250) \times 255 \times 8 = 65.280$ bit/s (zur Erinnerung: die maximale Payload beträgt 250 Byte). Dies liegt zwar weit unter der

Standardrate von 1 Mbit/s, so dass mit diesen Spezifikationen keine High-Fidelity-Audioqualität möglich ist, aber das ist ohnehin nicht unser Ziel: Für uns ist die Verständlichkeit wichtiger.

Die Schaltung

Der Einfachheit zuliebe habe ich einen bandbegrenzten Ein-Transistor-Kondensatormikrofon-Vorverstärker als Audioeingang verwendet und einen klassischen, auf dem LM386 basierenden Verstärker als Audioausgang hinzugefügt (Bild 3). Die Eingangsbandbreite wird am unteren Ende durch die etwas unterdimensionierten C1 und C5, am oberen Ende wird durch die Tiefpassfilter R4/C2 und R5/C3 begrenzt. Ähnliche Tiefpassfilter werden am Ausgang des DACs eingesetzt. Das Signal an der „heißen“ Seite von P1 sollte nicht größer als $400 \text{ mV}_{\text{SS}}$ sein. Als ESP32-Modul habe ich mich für das ESP32-PICO-KIT entschieden. Es gibt viele andere Module, aber nicht alle stellen die DAC-Ausgänge an GPIO25 und GPIO26 zur Verfügung. Außerdem brauchen wir einen ADC-Eingang. Ich habe dafür GPIO32 (ADC1, Kanal 4) verwendet. Der Testpunkt TP1 an GPIO26 (der zweite DAC-Ausgang) ist als Monitoreausgang für das Mikrofonsignal vorgesehen. Ein Drucktaster an GPIO33 bietet Push-to-Talk-Funktionalität (PTT), und die LED an GPIO27 ist die bei den meisten Mikrocontroller-Schaltungen übliche Multifunktions-LED.

Die Stromversorgung ist in einen analogen und einen digitalen Teil aufgeteilt ist. Der Grund dafür ist nicht, dass schnelle digitale Schaltgeräusche nicht in den Audioeingang eingekoppelt werden, sondern die Vermeidung eines bestimmten Klickgeräusches am Ausgang. Offenbar erzeugt eine auf dem ESP32 laufende Aufgabe periodische Stromstöße, die hörbar werden können, wenn die Schaltung nicht sorgfältig aufgebaut ist. Der beste Weg dazu ist die Verwendung von zwei separaten Stromversorgungen (Bild 4). Das ESP32-Modul muss in

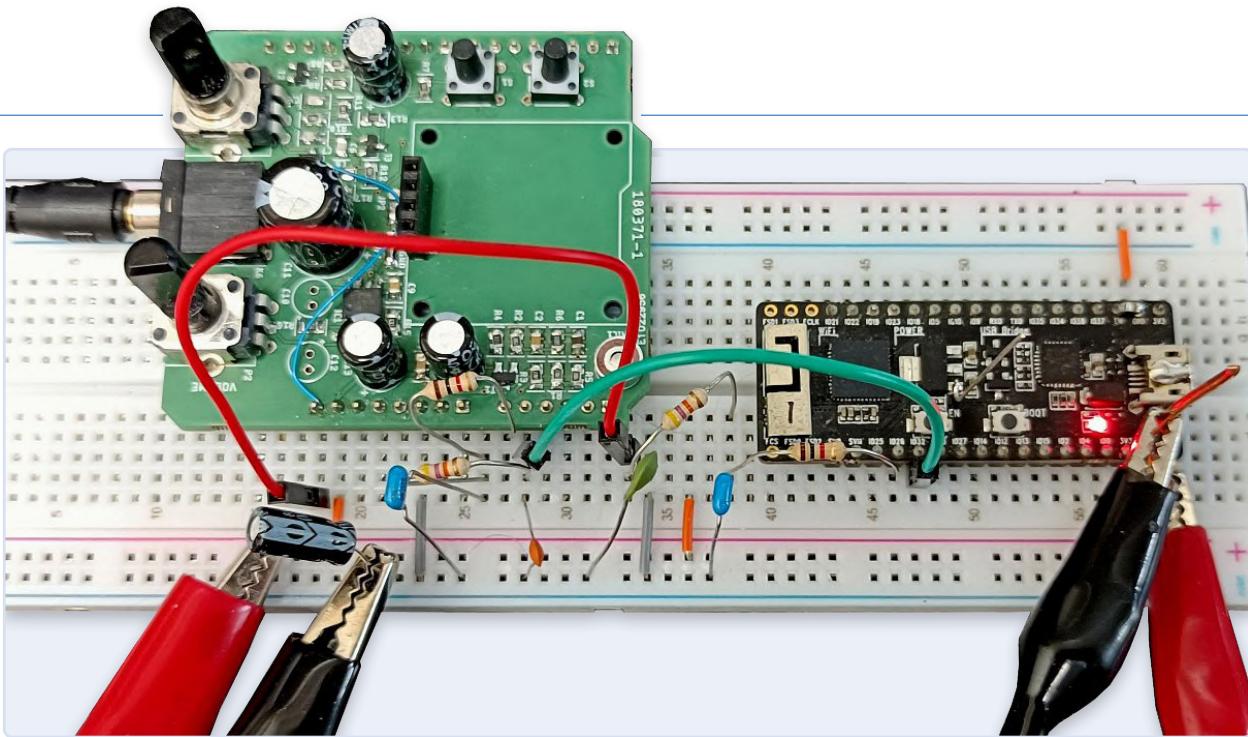


Bild 4. Ein Proof-of-Concept, aufgebaut auf einem Breadboard mit einem ESP32-PICO-KIT und einem leicht modifizierten Schnarchschutz-Shield [2] als Eingangs- und Ausgangsverstärker. Man beachte die beiden Krokodilklemmenpaare, die für die getrennte analoge und digitale Stromversorgung sorgen.

dieser Beziehung wie ein Bauteil behandelt werden, das eine Stromversorgung benötigt (wie der LM386), und nicht wie ein Modul, das auch den Rest der Schaltung mit Strom versorgen kann. Denken Sie daran, dass der LM386 einen Versorgungsspannungsbereich von 4..12 V hat. C10 ist optional und wird nur in einigen seltenen Fällen bei frühen ESP32-Modulen benötigt, die nicht richtig booten, wenn sie nicht an einen Computer angeschlossen sind (oder ähnliche Probleme haben). Zufälligerweise hatte ich noch einige dieser frühen Module, und so habe ich C10 in meinen Entwurf aufgenommen.

Die Software

Das Programm für das Walkie-Talkie basiert auf dem *ESPNow_Basic_Master*-Beispiel aus dem Arduino-ESP32-Boards-Paket von Espressif. Nachdem ich es an meine Bedürfnisse angepasst hatte, fügte ich Audio-Sampling und -Wiedergabe hinzu. Es gibt ein paar Dinge, die Sie vielleicht über das Programm wissen möchten und sollten. Audio-Sampling und Audio-Wiedergabe werden von einem Timer-Interrupt gesteuert, der mit 8 kHz läuft. Für das Sampling setzt die Interrupt-Service-Routine (ISR) des Sample-Rate-Timers nur ein Flag, um zu signalisieren, dass ein neues Sample erfasst werden soll. Die Funktion `loop()` fragt dieses Flag ab und führt die erforderlichen Maßnahmen durch. Dies liegt daran, dass der ADC nicht innerhalb einer ISR gelesen werden sollte, wenn die von Espressif bereitgestellte ADC-API verwendet wird. Die hier verwendete Funktion `adc1_get_raw()` ruft alle möglichen anderen Funktionen auf, die Dinge tun können, über die Sie keine Kontrolle haben. Da die ESP32-Software in einer Multitasking-Umgebung läuft, ist es wichtig, die Thread-Sicherheit zu gewährleisten. Wenn Sie Arduino für die ESP32-Programmierung verwenden, wird vieles davon automatisch erledigt, aber wenn Sie vorhaben, mein Programm auf ESP-IDF zu portieren, müssen Sie vielleicht etwas vorsichtiger sein.

Die Audiowiedergabe ist einfach, da die Sample-Rate-Timer-ISR ein Sample in den DAC schreibt, just wenn eines verfügbar ist. Ist dies nicht der Fall, wird der DAC-Ausgang auf die Hälfte der ESP32-Versorgung, also 1,65 V eingestellt. Das einzige Beachtenswerte ist ein so genannter Ping-Pong-Puffer, der den digitalen Audioempfang optimiert

(Bild 5). Ein solcher Puffer besteht eigentlich aus zwei Puffern, von denen einer gefüllt wird, während der andere gelesen wird. Dadurch kann es zu Überschneidungen kommen. Theoretisch sollte dies zwar nicht passieren, da Sender und Empfänger die gleiche Abtastrate und Timing-Logik verwenden, in der Realität ist dies jedoch aufgrund von Timing-Toleranzen manchmal der Fall. Ein Ping-Pong- oder Doppel-puffer hilft, störende Knackser bei der Wiedergabe zu vermeiden. Beachten Sie, dass ein ungeordneter Empfang von Datenpaketen nicht behandelt wird.

Pairing

Die Walkie-Talkie-Firmware ist ein Master-Slave-System. Der Master arbeitet im Wi-Fi-Station-Modus (STA), während ein Slave im Access-Point-Modus (AP) ist. Der Master stellt unmittelbar eine Verbindung zu einem Slave her, wenn er einen erkennt, und kann sofort mit dem Senden von Daten beginnen. Wenn der Master eine Verbindung zum Slave herstellt, wird jedoch nicht gleichzeitig der Slave mit dem Master verbunden. Der Slave kann keine Daten an den Master senden, und ein Zwei-Wege-Betrieb ist nicht möglich (zumindest ist es mir nicht gelungen; wenn Sie es besser wissen, lassen Sie es mich bitte wissen). Eine Möglichkeit, den Slave mit dem Master zu verbinden, ist die Verwendung von Datenempfangs-Callbacks. Wenn Daten empfangen

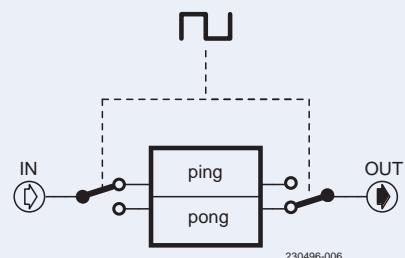


Bild 5. Doppel- oder Ping-Pong-Pufferung hilft, Diskontinuitäten in einem Datenstrom zu vermeiden.



werden, wird die Adresse des Absenders zusammen mit den Daten an diese Funktion übergeben. Sobald der Slave also etwas empfängt, kann er sich mit dem Absender der Daten verbinden. Dazu habe ich dieselben Funktionen und Verfahren wie der Master verwendet, um sich mit dem Slave zu verbinden. Es gibt jedoch eine Raffiniertheit, die nicht sehr gut (wenn überhaupt) dokumentiert ist: Der Slave muss sein Wi-Fi-Schnittstellenfeld auf `ESP_IF_WIFI_AP` setzen, sonst funktioniert er nicht. Dieses Feld ist standardmäßig auf `ESP_IF_WIFI_STA` voreingestellt, wie es vom Master benötigt wird, so dass das Programm es im Normalfall nicht explizit setzen muss. Daher erscheint das Feld nirgends in den Beispielprogrammen, so dass der Benutzer nichts von seiner Existenz weiß.

Push-to-Talk

Wenn ESP-NOW kontinuierlich sendet, wird die MCU ziemlich schnell heiß. In der Walkie-Talkie-Anwendung gibt es aber keinen Grund, ununterbrochen zu streamen, und so habe ich eine Push-to-Talk-Taste (auch bekannt als PTT) hinzugefügt (S1). Drücken Sie diese Taste und halten Sie sie gedrückt, während Sie sprechen. Wenn der Sender mit dem Empfänger gepaart ist, leuchtet die LED auf. Auf der Empfängerseite leuchtet die LED ebenfalls auf und zeigt damit an, dass ein Anruf eingeht. Um Rückkopplungen zu vermeiden, wird der Audioausgang auf der Senderseite stummgeschaltet, wenn die PTT-Taste gedrückt wird. Obwohl die Kommunikation im Prinzip vollduplex ist, sollten die beiden Gesprächspartner dennoch nicht gleichzeitig sprechen. Dies ist eine gute Gelegenheit, „Roger“ und „Over“ in Ihre Sätze einzubauen.

Ein Programm für alle Fälle

Das Programm besteht aus einer Arduino-.ino-Datei (Sketch). Außer dem Espressif-ESP32-Board-Paket werden keine weiteren Bibliotheken benötigt. Das Walkie-Talkie benötigt ein Master- und ein Slave-Gerät. Um das Programm für das Master-Gerät zu kompilieren, kommentieren Sie Zeile 12 aus, in der `NODE_TYPE_SLAVE` steht. Für das Slave-Gerät muss dieses Makro definiert werden. Sie können auch einige andere Einstellungen ändern, wenn Sie möchten. So ist es auch möglich, ohne die Unterstützung des Audioeingangs (`AUDIO_SOURCE`) und/oder -ausgangs (`AUDIO_SINK`) zu kompilieren. Dies ist praktisch für die Fehlersuche oder für eine Anwendung, die nur eine einseitige Kommunikation benötigt. Der Quellcode kann von [3] heruntergeladen werden.

Höhere Wiedergabebetreue?

Es sollte nicht allzu kompliziert sein, hochwertige Audiodaten über ESP-NOW zu übertragen, wenn man statt des einfachen Mikrofonverstärkers und des im ESP32 eingebauten ADC und DAC auf I^S umsteigt. Dies macht die Schaltung und das Programm etwas komplexer, würde aber - zumindest theoretisch - das Streaming von 16-Bit-Audiodaten mit einer Abtastrate von 48 kHz ermöglichen. Allerdings muss der möglicherweise nicht ordnungsgemäße Empfang von Paketen richtig gehandhabt werden. Aber hey, wurde Bluetooth nicht genau dafür entwickelt?

Reichweitentest

Um zu sehen, ob ESP-NOW eine Kommunikation über große Entfernnungen ermöglicht, habe ich ein einfaches Programm geschrieben, das einmal pro Sekunde eine Ping-Nachricht an den Slave sendet. Der Slave war nichts weiter als ein ESP32-PICO-KIT mit einer an GPIO27 angeschlossenen LED, die über eine USB-Powerbank mit Strom versorgt wurde. Jedes Mal, wenn ein Ping empfangen wird, blinkt die LED kurz auf (100 ms).

Mit dem Sender, der im Freien in 1 m Höhe über dem Boden platziert wurde, erreichte ich eine Sichtverbindung (LOS) von etwa 150 m. Ab dieser Entfernung wurde der Empfang unregelmäßig, und der Slave musste höher, etwa 2 m über dem Boden gehalten werden. Diese Situation kann wahrscheinlich durch eine sorgfältigere Positionierung der beiden Gegenstellen verbessert werden. ↗

RG – 230496-02

Haben Sie Fragen oder Kommentare?

Haben Sie technische Fragen oder Kommentare zu diesem Artikel? Schicken Sie eine E-Mail an den Autor unter clemens.valens@elektor.com oder kontaktieren Sie Elektor unter redaktion@elektor.de.

Über den Autor

Nach einer Karriere in der Marine- und Industrieelektronik begann Clemens Valens 2008 bei Elektor als Chefredakteur von Elektor Frankreich. Seitdem hatte er verschiedene Positionen inne und wechselte vor kurzem in die Abteilung Produktentwicklung. Seine Hauptinteressen sind Signalverarbeitung und Klangerzeugung.



Passende Produkte

- **ESP32-PICO-KIT V4**
www.elektor.de/18423
- **Elektor ESP32 Smart Kit Bundle**
www.elektor.de/19033



WEBLINKS

- [1] Mehr über ESP-NOW: <https://espressif.com/en/solutions/low-power-solutions/esp-now>
- [2] Schutzschild gegen Schnarchen: <https://www.elektormagazine.de/magazine/elektor-69/42260>
- [3] Downloads zu diesem Artikel: <https://elektormagazine.de/230496-02>

Von der Idee zur Schaltung mit dem ESP32-S3

Ein Leitfaden für die Prototypentwicklung mit Espressif-Chips

Von Liu Jing Hui, Espressif

Viele Produkte und Geräte mit Espressif-Chips, insbesondere Kleinserien und Einzelstücke, werden zunächst auf einer Entwicklungsplatine als Prototyp entwickelt. Ingenieure übertragen ihr Design später in der Regel auf ein Modul oder, wenn sie an Produkten mit höheren Stückzahlen arbeiten oder wenn sie ein schöneres Produkt benötigen, auf einen nackten Chip. Dabei gibt es einige Dinge zu beachten. Schauen wir uns die häufigsten Fallstricke an.

Am Beispiel des ESP32-S3 wollen wir zeigen, wie man mit Espressif-Chips entwickelt. Bei diesem Chip handelt es sich um einen hochintegrierten, stromsparenden 2,4-GHz-System-on-Chip (SoC) mit WLAN- und Bluetooth-LE-Funktionalität [5]. Er enthält zwei schnelle CPU-Kerne, extern erweiterbaren RAM und Peripherie, die für eine Vielzahl von Anwendungen ausreichen dürfte. Relevant ist hier, dass er auch fortschrittliche Kalibrierungsmethoden kennt, um etwaige Funkfehler zu kompensieren. Dies erleichtert die Inbetriebnahme des Designs und macht spezielle Prüfgeräte überflüssig. Der ESP32-S3 ist eine ideale Wahl für eine Vielzahl von Anwendungsszenarien rund um KI und Künstliche Intelligenz der Dinge (AloT). Beachten Sie, dass Sie ihn sowohl als „nackten“ Chip, als auch in Form eines Moduls wie den ESP32-S3-WROOM-1 erwerben können. Das Modul umgibt den ESP32-S3 mit den erforderlichen Quarz-, Flash-, HF-Schaltungen und besitzt entweder eine Trace-Antenne oder einem Anschluss für eine externe Antenne. Wir werden hier Details sowohl für die Verwendung des nackten Chips als auch für die Verwendung eines Moduls angeben. Wenn Sie diesen Chip in Ihren Entwurf verwenden möchten, gibt es vier Hauptaufgaben, auf die Sie achten sollten:

- Schaltungsentwurf und Schaltplanerstellung
- Platinenlayout-Entwurf
- HF- und Quarzabgleich
- Firmware-Download und Fehlerbehebung

Schaltungsentwurf und Schaltplanerstellung

Um loszulegen, sollten Sie die Dokumentation des betreffenden Chips und/oder Moduls durchgehen. Unter [1] stellen wir verschiedene Dokumente online zur Verfügung. Dokumente wie das Datenblatt,

Hardware-Design-Richtlinien und Modul-Referenzdesigns wurden erstellt, um zu zeigen, was der Chip benötigt, um eine optimale Leistung zu erbringen.

Wir zeigen in Bild 1 einen ESP32-S3-Chip sowie alle Komponenten, die er im Allgemeinen benötigt. Im Gegensatz zu Chips ohne Funkfunktion ist eine Entkopplung hier ziemlich wichtig. Insbesondere haben wir die Anzahl und die Werte der Entkopplungskondensatoren an jedem Versorgungspin überprüft. Es ist wichtig, genau diese zu verwenden und keine Kondensatoren einzusparen. Insbesondere an Pin 2 und Pi 3 muss eine CLC-Filterschaltung vorhanden sein (C8, L1 und C9 im Schaltplan). Im Allgemeinen gibt es für die meisten Espressif-Chips auch dedizierte Power-Pins für die HF-Versorgung, so dass dort immer auch ein CLC/CCL-Filter erforderlich ist, um Oberschwingungen zu unterdrücken.

Ein weiterer wichtiger Punkt in Bezug auf die Stromversorgung ist, dass für den ESP32-S3 mindestens 500 mA erforderlich sind. Dies gilt universell für alle bisher veröffentlichten Espressif-Chips. Einzige Ausnahme ist der ESP32-H2, der sich mit nur 350 mA zufriedengibt. Wenn Ihre Stromversorgung nicht genügend Strom zur Verfügung stellt, führt dies in der Regel zu Brownout-Resets und anderen Auffälligkeiten, meistens bei der Initialisierung des WLANs.

Der ESP32-S3 benötigt einen Hauptquarz als Taktquelle für das gesamte System. Bitte fügen Sie eine Reiheninduktivität im XTAL_P-Pfad hinzu, um Oberschwingungen des Quarzes zu unterdrücken. Die Werte der Lastkondensatoren C1 und C4 hängen vom Quarz sowie von der parasitären Impedanz der Leiterbahnen und der Pads ab. Adafruit hat ein gutes Tutorial zur Berechnung eines Anfangswerts für diese [2]. Für die beste Reichweite und Zertifizierungskonformität müssen Sie diese theoretischen Werte jedoch messen und optimieren, sobald Sie die Leiterplatte vor sich haben. Weitere Informationen finden Sie unten in diesem Artikel.

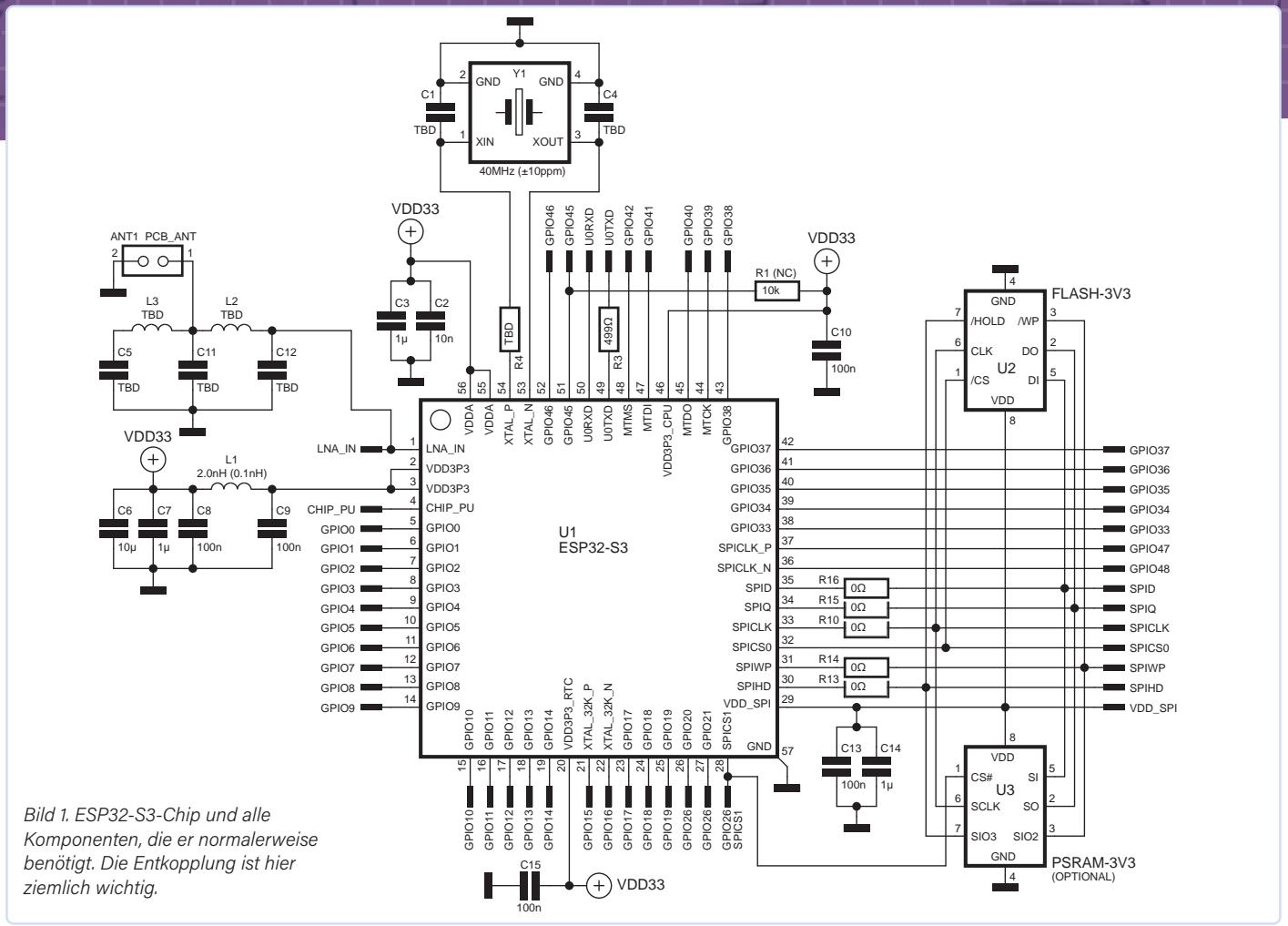


Bild 1. ESP32-S3-Chip und alle Komponenten, die er normalerweise benötigt. Die Entkopplung ist hier ziemlich wichtig.

Ein ESP32-S3 benötigt immer Flash-Speicher zur Speicherung seines Programms und kann zudem optional mit einem PSRAM verbunden werden, wenn zusätzlicher Speicher benötigt wird. Beachten Sie, dass einige Varianten des ESP32-S3 bereits Flash und/oder PSRAM enthalten. In diesem Fall können Sie U2 und/oder U3 weglassen. Dadurch sparen Sie den Platz für diese Chips und können die Platine schön klein halten. Lassen Sie die Entkopplungskondensatoren aber auf jeden Fall auf VDD_SPI bestehen, da sie weiterhin zur Entkopplung des internen Flash/PSRAM benötigt werden.

Um die beste Funkreichweite zu erzielen und gleichzeitig zu erreichen, dass das Gerät den Anforderungen der EMV-Zertifizierung entspricht, müssen Sie dafür sorgen, dass die Impedanzen auf dem Weg vom ESP32-S3 zur Antenne abgestimmt sind. Im Allgemeinen hat die Übertragungsleitung (Leiterbahn) zwischen dem ESP32 und der Antenne eine Impedanz von $50\ \Omega$, aber das LNA_IN-Pad des ESP32-S3 nicht und die von Ihnen verwendete Antenne möglicherweise auch nicht. Mit anderen Worten: Der Anschluss des ESP32 an die Übertragungsleitung erfordert eine π -CLC-Anpassungsschaltung in der Nähe des ESP32-S3, um die Impedanz auf $50\ \Omega$ anzupassen. Der Anschluss einer Antenne an die Leiterbahn kann auch eine CLC-Anpassungsschaltung an Ort und Stelle erfordern, die Sie nur weglassen können, wenn Sie zum Beispiel durch Simulation garantieren, dass die Antennenimpedanz $50\ \Omega$ beträgt. Beachten Sie, dass im Schaltplan dem CLC-Netzwerk (bestehend aus C11, L2 und C12) keine Werte zugewiesen sind. Dies liegt daran, dass die benötigten Werte vom Platinenlayout und -material abhängen und erst nach der Designphase festgelegt werden müssen. Weitere Informationen finden Sie weiter unten in diesem Artikel.

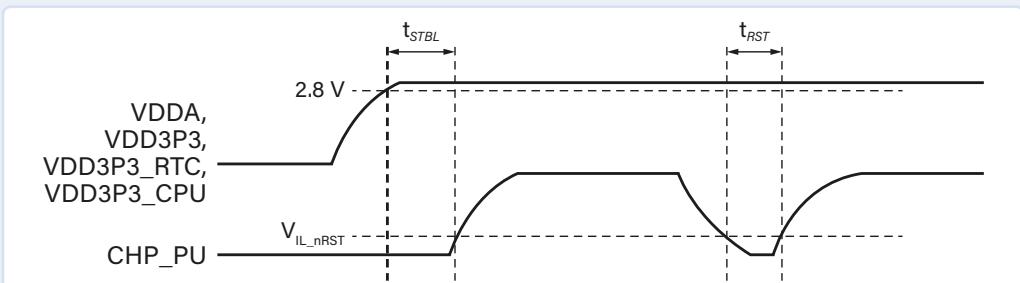
Wenn Sie ein Modul verwenden, müssen Sie sich – wie praktisch – um nichts kümmern, da das Modul bereits die richtigen Bauteile mit den richtigen Werten enthält.

Zu beachten ist, wenn Sie ein Modul oder einen Chip verwenden, dass der CHIP_PU-Pin (auf dem Modul als EN gekennzeichnet) sowohl als Freigabe- als auch als Reset-Pin fungiert. Er sollte keinesfalls freischwebend sein. Um den ESP32-S3-Chip zuverlässig zu starten, sollte der CHIP_PU-Pin erst aktiviert werden, wenn die Stromversorgung stabil ist, wie dies im Kasten „Timing-Parameter...“ erläutert wird. In der Regel wird diesem Pin eine RC-Schaltung hinzugefügt, um eine Verzögerung zu erzeugen. Einige Anwendungen müssen nämlich auch in einem Szenario zuverlässig funktionieren, in dem die Stromrampe sehr langsam ist, häufiges Ein- und Ausschalten erfordert oder die Stromversorgung instabil ist. Wenn beispielsweise ein ESP32-S3 über ein Solarpanel mit Strom versorgt wird, kann es sogar passieren, dass die ausschließliche Verwendung einer RC-Schaltung die Ein-/Ausschaltsequenz-Timings nicht einhält und der der Chip nicht erfolgreich hochfährt. In diesem Fall empfehlen wir den Einsatz eines externen Power-Supervisor- oder Watchdog-Chips. Wenn Ihr Design dies zulässt, können Sie den CHIP_PU-Pin auch über eine andere MCU steuern oder einfach eine Reset-Taste hinzufügen.

Der letzte Schritt, um den ESP32-S3 erfolgreich zu betreiben, besteht darin, auf die Konfigurationspins zu achten. Bei jedem Start oder Zurücksetzen benötigen Espressif-Chips einige initialen Parameter zur Konfiguration, etwa, in welchem Boot-Modus der Chip gestartet werden soll, die Spannungswerte des Flash-Speichers und so weiter. Diese Parameter werden über die Konfigurationspins ausgewählt. Nach dem Reset arbeiten die Konfigurationspins wie normale I/O-Pins. Die Details dazu finden Sie wie gewohnt im Chip-Datenblatt.

Von Entwicklern für Entwickler

Timing-Parameter für Power-up und Reset



Visualisierung von Timing-Parametern für Power-up und Reset. (Quelle [5])

Parameter	Beschreibung	Min. Zeit (µs)
t_{STBL}	Zeit, die für die Stabilisierung der 3,3-V-Versorgung reserviert ist, bevor der CHIP_PU Pin auf High gezogen wird, um den Chip zu aktivieren	50
t_{RST}	Zeit, die für CHIP_PU reserviert ist, um unter VL_nRST zu bleiben, um den Chip zurückzusetzen	50

Wenn all dies berücksichtigt ist, kann unser ESP32-S3 erfolgreich booten. Wahrscheinlich möchten Sie ihn jedoch auch mit anderen Chips, Sensoren und Aktuatoren verbinden. Welche GPIOs wären dafür geeignet? Die gute Nachricht: Alle Espressif-Chips verfügen über eine Funktion namens GPIO-Matrix, die es wirklich einfach macht, Peripherie an den ESP32-S3 anzuschließen. Sie ermöglicht Ihnen, JEDEN verfügbaren GPIO auf JEDES Signal der meisten Peripherieeinheiten wie I²C, SPI, SDIO abzubilden. Natürlich gibt es immer noch einige Peripherieeinheiten wie den ADC, die feste GPIOs verwenden. Weitere Informationen finden Sie bei Bedarf im Chip-Datenblatt. Denken Sie daran, zuallererst die technischen Beschreibungen zu lesen, auch wenn es schwerfällt. Es ist besser, Informationen, die Sie bereits kennen, noch einmal aufzufrischen, als eine Platine aufgrund eines vermeidbaren Fehlers neu routen zu müssen.

Entwurf des Platinenlayouts

Wenn Sie nur diesen Chip verwenden, sollten Sie nun einen perfekten Schaltplan für Ihr Gerät haben. Bei WLAN- und Bluetooth-Projekten ist jedoch auch das Platinenlayout entscheidend für eine qualifizierbare

HF-Performance. Selbst wenn Sie einen Espressif-Chip nur als MCU verwenden, ohne eine Antenne anzubinden oder das Funkmodul zu verwenden, sollten Sie die folgenden Richtlinien beachten, um ein stabiles System mit erhöhter Zuverlässigkeit zu erreichen. Beachten Sie, dass alle genauen Details in den *Hardware Design Guidelines* zu finden sind. Hier werden wir nur die wichtigsten Punkte ansprechen. Beginnen wir mit dem Lagenaufbau der Platine. Empfehlenswert sind vier (oder mehr) Lagen, da neben der Chiplage eine große Massefläche als Referenzmasse zur Verfügung steht. Wenn Sie wirklich nur zwei Lagen verwenden möchten, vergessen Sie nicht, auch eine möglichst gute Massefläche zu integrieren.

Der zweite wichtige Punkt ist die Verlegung der Versorgungsspannungen. Wir empfehlen, sie sternförmig auf einer inneren Lage zu verlegen. Jeder Entkopplungskondensator und jeder CLC-Filterkreis sollte sich so nah wie möglich an den Versorgungspins befinden (**Bild 2**). Der dritte Punkt betrifft Ihre HF-Leiterbahnen. Sie sollten den Platinenhersteller bitten (besser: beauftragen), für eine kontrollierte Impedanz von 50 Ω für alle HF-Leiterbahnen zu sorgen. Dies beginnt in der Phase des Layoutprozesses, in dem Sie die richtige Leiterbahnbreite und den Abstand zur Masse basierend auf Ihrem Lagenaufbau berechnen müssen. Der Hersteller kann diese Parameter dann leicht anpassen. Wenn Sie Bauteile in der SMD-Gehäusegröße 0201 einsetzen, verwenden Sie bitte eine Stichleitung (Stub) im Platinenlayout der HF-Anpassungsschaltung in der Nähe des Chips.

Als nächstes kommt es auf die Quarzbeschaltung an. Die Reiheninduktivität sollte sich sehr nahe am ESP32-S3-Chip befinden und die beiden Lastkondensatoren sollten auf beiden Seiten des Quarzes platziert werden (**Bild 3**). Ausreichende Masseanbindungen mit großer Maschendichte sind für die Quarzbeschaltung sehr wichtig.

Eine letzte Sache, auf die Sie achten sollten, sind die Flash- und UART-Leiterbahnen sowie die Leiterbahnen, die zu anderen peripheren GPIO-Pads führen. Diese Leiterbahnen können hochfrequente Oberschwingungen erzeugen, so dass wir empfehlen, sie auf den inneren Lagen zu verlegen und sie so weit wie möglich mit Massebahnen/flächen zu umgeben.

Wenn Sie ein Modul verwenden, ist es entscheidend, wie Sie das Modul platzieren, um die beste HF-Leistung zu erzielen. Die folgenden Richtlinien gelten nur für Module mit einer PCB-Antenne. Wenn Sie ein Modul mit einer externen Antenne verwenden, haben Sie mehr Flexibilität bei der Platzierung auf der Leiterplatte.

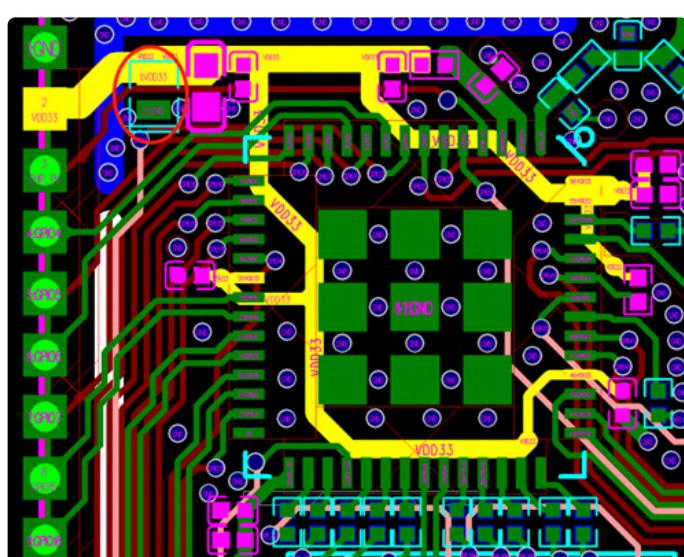


Bild 2. Jeder Entkopplungskondensator sollte sich so nah wie möglich an den Versorgungspins befinden.

Wie Sie in **Bild 4** sehen, ist es empfehlenswert, das Modul in einer Ecke der Platine so zu platzieren, dass der Einspeisepunkt auch einem Platinenrand am nächsten ist. Beachten Sie auch, dass nicht alle Modultypen den Einspeisepunkt auf der gleichen Seite haben; auch deshalb sollten Sie das Datenblatt überprüfen, bevor Sie die Platine entwerfen. Wir empfehlen, die Antenne komplett außerhalb der Platine anzubringen. Falls dies nicht möglich ist, sorgen Sie zumindest dafür, dass unter und um die Antenne herum ein Abstand von mindestens 15 mm vorhanden ist, in dem sich keine Leiterbahnen, Pads, Durchkontaktierungen oder Komponenten befinden sollten (**Bild 5**).

Während es bei den meisten Verbindungen unserer Wroom-Module ziemlich einfach, sie mit einem handelsüblichen Lötkolben zu löten, befindet sich darunter ein Masseanschluss, der normalerweise für das Löten von Hand unzugänglich ist. Wenn Sie dazu in der Lage sind (zum Beispiel mit einem Reflow-Ofen oder einer Heißluft-Rework-Station), empfehlen wir Ihnen, das Modul an eine Masseplatte anzuschließen, da dies hilft, Wärme abzuleiten und eine bessere Erdung für das Funkmodul zu gewährleisten. Unter normalen Umständen funktioniert das Modul aber auch ohne sie.

HF und Quarzabgleich

Wenn Sie einen Chip verwenden, ist es Voraussetzung, dass Sie alle oben genannten Punkte befolgt haben, um eine funktionierende Leiterplatte vom Platinenhersteller geliefert zu bekommen. Um eine optimale Zuverlässigkeit zu gewährleisten und die EMV-Zertifizierung zu bestehen, benötigen Sie aber leider noch einige ziemlich professionelle Instrumente, um die HF auf die passenden Werte und die Quarzkondensatoren abzustimmen. Für den Fall, dass Sie zu solchen Geräten Zugang haben (oder falls billige Geräte wie der MicroVNA gut genug sind), wollen wir Ihnen erklären, worauf es bei diesem Verfahren ankommt.

Wahrscheinlich haben Sie die Werte der Lastkondensatoren entsprechend der geschätzten parasitären Kapazität der Leiterbahnen ausgewählt. Jetzt ist es an der Zeit, diese Parameter zu verfeinern: Da die Hauptfrequenz von 2,4 GHz von diesem Quarzoszillator abgeleitet ist, erhalten Sie durch dessen Optimierung einen besseren Funktionsbereich und niedrigere Oberschwingungen. Wir werden dies optimieren, indem wir das *ESP RF Test Tool* verwenden, das Sie von unserer Website herunterladen können [3].

- Website herunterladen können [5].

 1. Wählen Sie den TX-Tone-Modus mit Hilfe dem Zertifizierungs- und Testtool aus.
 2. Beobachten Sie das 2,4-GHz-Signal mit einem Analysator für Funkfrequenzen oder einem Spektrumanalysator und demodulieren Sie es, um den tatsächlichen Frequenzoffset zu erhalten.
 3. Stellen Sie den Frequenzversatz auf ± 10 ppm (empfohlen) ein, indem Sie die externe Lastkapazität anpassen.

➤ Wenn der Mittenfrequenz-Offset positiv ist, bedeutet dies, dass die äquivalente Lastkapazität zu klein ist und die externe Lastkapazität erhöht werden muss.

➤ Wenn der Mittenfrequenz-Offset negativ ist, bedeutet dies, dass die äquivalente Lastkapazität zu groß ist und die externe Lastkapazität reduziert werden muss.

➤ Die externen Lastkapazitäten an den beiden Seiten sind in der Regel gleich, können aber in besonderen Fällen leicht unterschiedliche Werte haben.

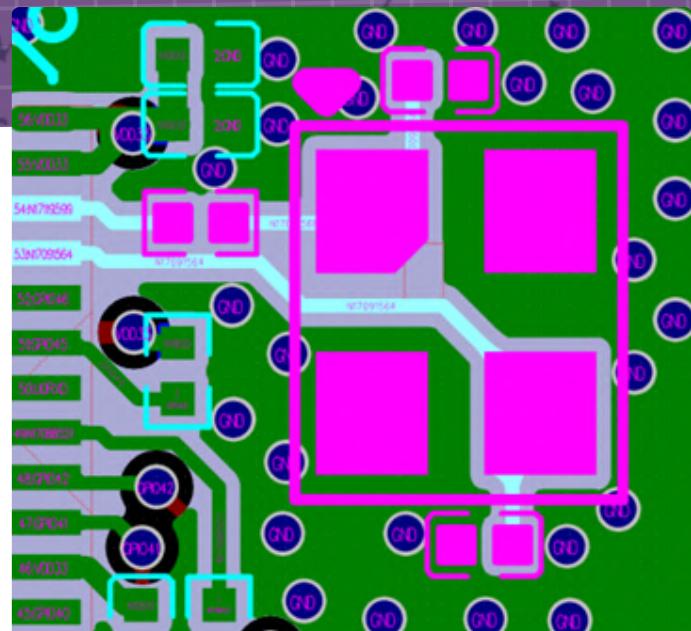


Bild 3 Die beiden Lastkondensatoren sollten dem Quarz zu Seiten stehen

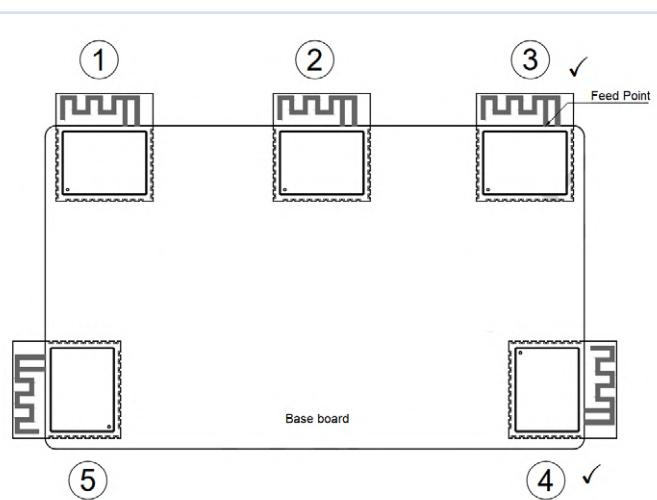


Bild 4. Optimale Position eines Moduls auf einer Basisplatine. (Quelle [6])

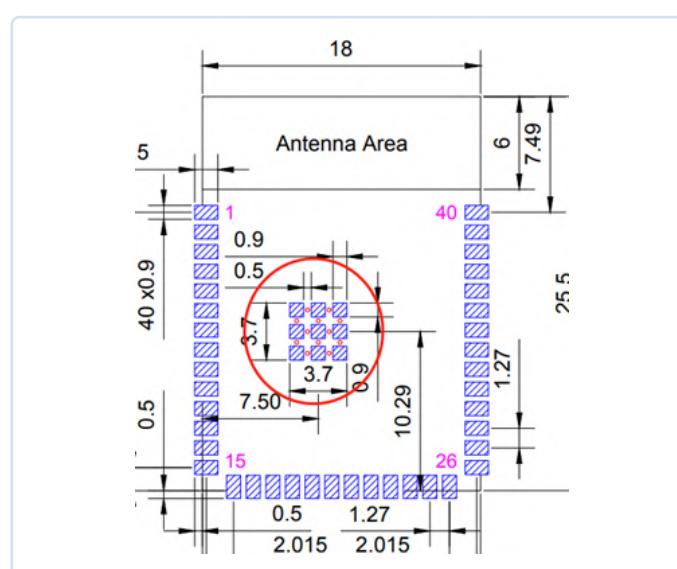


Bild 5. Empfohlener Leiterplatten-Footprint für den ESP32-S3-WROOM-1. (Quelle: [6])

Von Entwicklern für Entwickler

Nachdem die Lastkondensatoren des Quarzes eingestellt sind, gehen wir zum Antennenanpassungsnetzwerk über. In der Anpassungsschaltung definieren wir den Port in der Nähe des Chips als Port 1 und den Port in der Nähe der Antenne als Port 2. S11 beschreibt das Verhältnis der Signalleistung, die von Anschluss 1 reflektiert wird, zur Eingangssignalleistung. S21 beschreibt den Übertragungsverlust des Signals von Anschluss 1 zu Anschluss 2. Bei den Chips der ESP32-S3-Serie gilt: Wenn S11 bei der Übertragung von 4,8-GHz- und 7,2-GHz-Signalen kleiner oder gleich -10 dB und S21 kleiner oder gleich -35 dB ist, kann die Anpassungsschaltung die Übertragungsanforderungen erfüllen. Verbinden Sie die beiden Enden der Anpassungsschaltung mit dem Netzwerkanalysator (siehe Kasten „Anpassungsnetzwerk“) und testen Sie den Signalreflexionsparameter S11 und den Übertragungsparameter S21. Passen Sie die Werte der Bauteile in der Schaltung an, bis S11 und S21 den Anforderungen entsprechen. Wenn Ihr Platinenlayout des Chips strikt den PCB-Designrichtlinien entspricht und die Antenne gut gestaltet ist, können Sie die Wertebereiche im Kasten verwenden, um das Anpassungsnetzwerk einzurichten.

Was sollten Sie aber tun, wenn Sie einen ESP32-S3-Chip in Ihr Projekt einbauen möchten, aber nicht über solche ausgefallene Ausrüstung verfügen? Nun, wenn Sie alle anderen Punkte in diesem Artikel befolgt haben, können Sie möglicherweise mit den geschätzten Werten für die Lastkondensatoren und ohne CLC-Netzwerk davonkommen. Wenn Sie sowieso eines in Ihre Platine aufgenommen haben, können Sie C11/C12 weglassen und L2 durch eine Drahtbrücke oder einen Null-Ohm-Widerstand ersetzen. Auf diese Weise können Sie später immer noch ein CLC-Netzwerk hinzufügen, ohne das Platinenlayout anpassen zu müssen. Unserer Erfahrung nach wirkt sich das Fehlen eines abgestimmten Netzwerks zur Impedanzanpassung nicht allzu sehr auf die Reichweite aus. Sein Hauptzweck besteht vielmehr darin, den EMV (Error Magnitude Vector) der Geräte zu reduzieren, der erforderlich ist, um die Zertifizierungsanforderungen zu erfüllen.

Wenn Sie ein Modul verwenden: Vergessen Sie alles, was Sie gerade dankenswerterweise gelesen haben. Sie brauchen nämlich keine der oben genannten Anpassungen, denn Ihr Modul wird mit allen Komponenten geliefert, die für eine optimale Leistung voreingestellt sind.

Firmware-Download und Fehlerbehebung

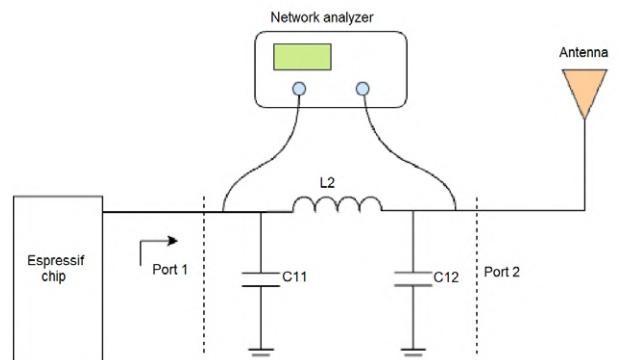
Ob Sie ein Modul oder einen Chip verwenden, bevor Ihr ESP32-S3 tatsächlich etwas tun kann, müssen Sie eine Firmware in den Flash-Speicher laden. Der Download-Prozess sieht bei allen Espressif-Chips wie folgt aus: Zurücksetzen des Chips in den Download-Modus – Download – Zurücksetzen und Ausführen des Chips im SPI-Boot-Modus.

Sie können Espressif-Chips im Allgemeinen auf zwei Arten programmieren: Alle Espressif-Chips unterstützen das Herunterladen von Firmware über den UART, aber die neueren Chips verfügen in der Regel auch über eine USB-Peripherie, über die Sie die Firmware direkt über eine USB-Verbindung Ihres Computers herunterladen können. Nachfolgend finden Sie die detaillierten Schritte für ESP32-S3.

So laden Sie über UART herunter:

1. Stellen Sie vor dem Download sicher, dass der Chip oder das Modul in den Download-Boot-Modus versetzt wird, das heißt, die Konfigurierpins GPIO0 (standardmäßig auf High) nach Low gezogen und GPIO46 (standardmäßig nach Low gezogen) wird frei schwebend oder auf Low gezogen sind. Konfigurieren Sie den GPIO45 entspre-

Anpassungsnetzwerk



Quelle [7]

Bauteil	Empfohlener Wert	Serien-Nr.
C11	1,2 ~ 1,8 pF	GRM0335C1H1RXBA01D
L2	2,4 ~ 3,0 nH	LQP03TN2NXPB02D
C12	1,8 ~ 1,2 pF	GRM0335C1H1RXBA01D

chend der Konfigurationstabelle oder lassen Sie ihn schweben, falls Sie ein Modul verwenden.

2. Schalten Sie den Chip oder das Modul ein und überprüfen Sie, ob er oder es über die serielle Schnittstelle UART0 in den *UART0-Download-Modus* gewechselt ist. Wenn im Protokoll „Waiting for Download“ angezeigt wird, ist der Chip oder das Modul in den *Download-Boot-Modus* gewechselt.
3. Laden Sie Ihre Firmware über den UART in den Flash. Sie können jede Programmieroption verwenden, die Ihnen Ihre Programmierumgebung (Arduino-IDE, ESP-IDF, VSCode, ...) bietet oder Sie können dafür das eigenständige Flash-Download-Tool verwenden.
4. Nachdem die Firmware heruntergeladen wurde, ziehen Sie IO0 auf High oder lassen Sie den Pin schweben, so dass der Chip oder das Modul in den *SPI-Boot-Modus* wechselt.
5. Schalten Sie das Modul wieder ein. Der Chip liest die neue Firmware während der Initialisierung und führt sie aus.

Beachten Sie, dass die meisten Entwicklungsbrettern über eine Möglichkeit im Schaltplan verfügen, die es der Software ermöglicht, den ESP32-Chip automatisch in den Download-Modus zu versetzen und wieder zu verlassen. Schauen Sie zum Beispiel auf den ESP32-S3-Devkit-C-1-Schaltplan in [4], wenn Sie wissen möchten, wie man so etwas realisiert.

Herunterladen über USB:

1. In den meisten Fällen, wenn der Chip über eine funktionierende Firmware verfügt, sollte das Flash-Tool in der Lage sein, den Chip über die USB-Verbindung in den *USB-Download-Modus* zu versetzen. In diesem Fall können Sie mit dem nächsten Schritt fortfahren. Wenn nicht, müssen Sie den Chip oder das Modul erst in den *Download-Boot-Modus* versetzen, also den Konfigurationspin GPIO0 (standardmäßig auf High) auf Low und den Pin GPIO46 (standardmäßig auf Low) schwebend oder auf Low zu ziehen. Konfigurieren Sie den Pin GPIO45 entsprechend seiner Konfigurationstabelle oder lassen Sie ihn schweben, wenn Sie ein Modul verwenden.
2. Schalten Sie den Chip oder das Modul ein und überprüfen Sie, ob er oder es über die serielle USB-Schnittstelle in den *UART-Download-Modus* gewechselt ist. Wenn im Protokoll angezeigt wird, dass auf *Download* gewartet wird, ist der Chip oder das Modul in den *Download-Boot-Modus* gewechselt.



3. Laden Sie Ihre Firmware über UART in den Flash. Sie können jede Programmieroption verwenden, die Ihnen Ihre Programmierumgebung (Arduino-IDE, ESP-IDF, VSCode, ...) bietet oder Sie können dafür das eigenständige Flash-Download-Tool verwenden.
4. Nachdem die Firmware heruntergeladen wurde und Sie automatisch in den *USB-Download*-Modus gewechselt sind, sind Sie fertig. Ist dies nicht der Fall, ziehen Sie IO0 auf High oder lassen Sie den Pin schweben, damit der Chip oder das Modul in den *SPI-Boot*-Modus wechselt. Wenn Sie den Chip oder das Modul zurücksetzen oder aus- und wieder einschalten, wird die neue Firmware während der Initialisierung gelesen und ausgeführt.

Wenn Sie die Firmware nicht über UART herunterladen können, überprüfen Sie das UART0-Protokoll über ein serielles Terminal. Die Startmeldung des ESP32-S3 informiert Sie über den tatsächlichen Speicherwert der Konfigurationspins, sodass Sie herausfinden können, welcher Konfigurationswert falsch ist. Der Wert nach dem **boot:** zeigt die Werte des Konfigurationspins in hexadezimaler Form an: Bit 2 = GPIO46, Bit 3 = GPIO0, Bit 4 = GPIO45, Bit 5 = GPIO3:

```
ets Jun 8 2016 00:22:57
rst:0x1 (POWERON_RESET),boot:0x3 (DOWNLOAD_BOOT(UART0/
UART1/SDIO_RESET,REO_V2))
```

Wenn Sie das USB-Gerät nicht erkennen oder die USB-Verbindung instabil ist, versuchen Sie zuerst, in den Download-Modus zu wechseln und dann herunterzuladen. Denken Sie auch daran, dass auf der Computerseite nur ein Programm die serielle Schnittstelle gleichzeitig geöffnet haben sollte: Der Versuch zu flashen schlägt fehl, wenn gleichzeitig ein serieller Anschluss auf demselben Port geöffnet ist. Beachten Sie, dass das Wechseln in den Download-Modus über USB zwar im Allgemeinen zuverlässig ist, es jedoch einige Szenarien gibt, in denen dies fehlschlagen kann:

- USB-PHY wird von der Anwendung deaktiviert
- Der USB-Anschluss wird für andere Aufgaben neu konfiguriert, zum Beispiel als USB-Host, USB-Standard-Device
- USB-GPIOs werden neu konfiguriert, zum Beispiel als Ausgänge für LEDC, SPI oder als generische GPIOs.

In diesem Fall ist es notwendig, manuell in den Download-Modus zu wechseln. Daher würden wir zumindest während der Firmware-Entwicklung vorschlagen, immer eine Möglichkeit (Tasten, Jumper, ...)

Über die Autorin

Liu Jinghui kam nach Abschluss ihres Studiums der Elektrotechnik zu Espressif und beschäftigte sich mit der Lösung von Hardwareproblemen. Seit fünf Jahren fungiert sie als Expertin für Espressif-Produkte und deren Nutzung durch die Kunden. Sie hat ein klares Verständnis für die Hardwaremerkmale und Probleme, die bei der Verwendung durch Kunden auftreten können.

bereitzustellen, um die Konfigurationspins zu setzen und den Start im Download-Modus zu erzwingen.

Eine weitere Sache, die Sie beachten sollten: Manchmal stellen Entwickler fest, dass ihr Board nicht richtig bootet (also ihr Programm nicht im Flash ausführt), nur wenn sie es manuell zurück setzen. Dies kann passieren, wenn ein Kondensator an einem Konfigurationspin platziert wird (zum Beispiel bei GPIO0 des ESP32-S3). Ein solcher Kondensator wird manchmal eingesetzt, um einen Taster zu entprellen, aber wenn das Gerät eingeschaltet wird, führt dies dazu, dass der Pegel auf GPIO0 nur langsam ansteigt, was der ESP32-S3 als Low-Pegel interpretiert.

Einfaches HF-Design mit Modulen

Wie Sie in diesem Artikel sehen können, ist es zwar möglich, etwas zu entwickeln, das „einfach funktioniert“, aber eine Platine mit einem ESP32-S3 so zu entwerfen, dass seine Leistungsfähigkeit voll genutzt und das Gerät erfolgreich zertifiziert werden kann, ist nicht trivial und erfordert einige fundierte Kenntnisse und Geräte. Wenn Sie also den Platz haben, um ein Modul in Ihrem Gerät unterzubringen, verwenden Sie eines. Die ganze knifflige HF-Magie ist bereits von Espressif erledigt, was die Integration in Ihr Design viel einfacher macht.

Wem das alles immer noch zu kompliziert klingt: Espressif bietet, wie zu Beginn des Artikels erwähnt, auch eine breite Palette von Entwicklungsbrettern an. Dieses Angebot reicht von einfachen Boards, die nur alle GPIOs zur Verfügung stellen, über DevBoards mit einfach zu handelnden Headern wie den ESP32-S3-DevKitC-1, bis hin zu vollständig integrierten Sprach-KI-Kraftpaketen wie der ESP32-S3-Box-3 und dem kamerafähigen ESP32-S3-Eye. Im Allgemeinen sind sogar Schaltpläne und Platinendesigns für diese Entwicklungsplatinen verfügbar, die, selbst wenn Sie es nicht vorhaben, ein solches DevBoard in Ihrem Produkt zu verwenden, eine großartige Starthilfe für Ihr Design sein können. 

Übersetzung von Jürgen Kellner (230563-02)

WEBLINKS

- [1] Technische Dokumente von Espressif:
https://espressif.com/en/support/documents/technical-documents?keys=&field_type_tid%5B%5D=842
- [2] Adafruit, „Choosing the Right Crystal and Caps for your Design“, 2012:
<https://blog.adafruit.com/2012/01/24/choosing-the-right-crystal-and-caps-for-your-design>
- [3] Test-Tool für ESP-RF : <https://espressif.com/en/support/download/other-tools>
- [4] Schaltung ESP32-S3-DevKitC-1: https://dl.espressif.com/dl/schematics/SCH_ESP32-S3-DevKitC-1_V11_20221130.pdf
- [5] Datenblatt ESP32-Serie: https://espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf
- [6] Richtlinien für das Hardware-Design der ESP32-H2-Serie:
https://espressif.com/sites/default/files/documentation/esp32-h2_hardware_design_guidelines_en.pdf
- [7] Datenblatt ESP32-S3-WROOM-1:
https://espressif.com/sites/default/files/documentation/esp32-s3-wroom-1_wroom-1u_datasheet_en.pdf

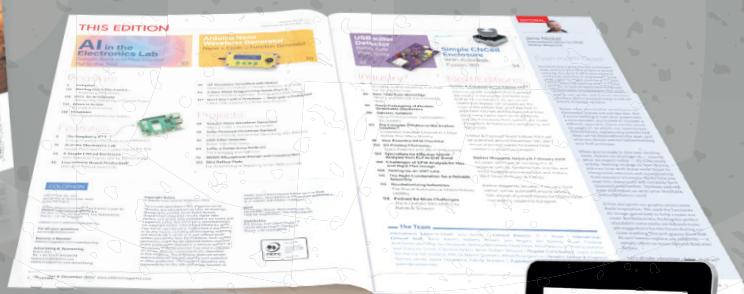
20%
discount
auf das erste Jahr Ihrer
Mitgliedschaft

Treten Sie jetzt der Elektor Community bei!

Jetzt



Mitglied werden!



- Komplettes Webarchiv ab 1970
- 8x Elektor Doppelheft (Print)
- 8x Digital (PDF)
- 10% Rabatt im Online-Shop und exklusive Angebote
- Zugriff auf über 5.000 Gerber Dateien aus Elektor Labs
- Kostenlose Lieferung innerhalb Deutschlands



www.elektormagazine.de/gold-member

Nutzen Sie den Gutscheincode:

ESPRESSIF20

ESPRESSIF

elektor

AIoT-Chip-Innovation



Ein Interview mit Espressif-CEO Teo Swee-Ann

Was braucht es, um eine innovative elektronische Lösung mit einer disruptiven Geschwindigkeit auf den Markt zu bringen? Teo Swee-Ann, CEO von Espressif, teilt seine Erkenntnisse. Wir erfahren etwas über seinen beruflichen Werdegang und wie er dazu kam, die beliebten Chipsätze, Module und AIoT-Lösungen zu entwickeln und einzusetzen.

Elektor: Wann haben Sie sich das erste Mal für Elektronik interessiert? Wurden Sie durch einen Verwandten oder einen Lehrer inspiriert? Oder wurden Sie vielleicht neugierig, als Sie Dinge nachbauten und mit elektronischen Produkten herumbastelten?

Teo Swee-Ann: Mein Interesse an der Elektronik begann, als ich noch sehr jung war. Als ich aufwuchs, war ich immer von Geräten und Maschinen fasziniert. Wir haben die üblichen Dinge gemacht, zum Beispiel Radios und Fernseher auseinandergenommen und versucht zu verstehen, wie sie funktionieren. Damals ging es nicht so sehr um Reverse Engineering, sondern eher um die reine Neugier, hinter die Kulissen zu schauen. Wie üblich hat das Auseinandernehmen von Dingen keine Fragen beantwortet, sondern uns nur noch neugieriger gemacht.

Niemand in meiner unmittelbaren Familie hatte mit Elektronik zu tun, also kam meine Inspiration nicht direkt von einem Verwandten. Aber ich scheine ein Händchen für Software und Mathematik zu haben. Meine Abschlussarbeit an der Universität befasste sich mit elektromagnetischen Berechnungen. Ich habe eine Reihe von Techniken erfunden, die für äußerst effiziente Berechnungen nützlich waren - Greensche Funktionen für geschichtete Medien.

Es stellte sich heraus, dass dies und eine Technik namens „partial elements equivalent circuit“ für die Charakterisierung von Induktivitäten in Chips nützlich sein könnte, insbesondere um Substratverluste zu erfassen. Ich glaube, so habe ich einen Job in der Chipindustrie gefunden. Als ich die Chance bekam, am Schaltungsentwurf zu arbeiten, besorgte ich mir sofort ein paar klassische Bücher zum Entwurf integrierter Schaltungen von Paul Gray, Philip Allen und Ken Martin und lernte sie praktisch auswendig. Danach gab es kein Zurück mehr.

Elektor: Waren Ingenieurwesen und Unternehmertum für Sie schon immer ein Thema? Als Sie zum ersten Mal die Universität besuchten, dachten Sie da schon daran, ein Unternehmen zu gründen und zu führen?

Teo Swee-Ann: Ich habe nicht über die geschäftliche Seite der Dinge nachgedacht. Ich bin eher ein Idealist; ich glaube, dass Wissen die Welt verändern wird - nicht die Wirtschaft. Aber meine späteren Erfahrungen haben mich natürlich gelehrt, dass man manchmal mehr tun, sich anpassen und auch eine Geschäftsplattform für die Technologie schaffen muss.

Elektor: Erzählen Sie uns von Ihren technischen Interessen. Worauf haben Sie sich während Ihres Masterstudiums an der National University of Singapore konzentriert?

Teo Swee-Ann: In meiner Studienzeit habe ich Mathematik und Software bevorzugt und alles gemieden, was mit Hardware zu tun hatte. Meine Masterarbeit befasste sich also mit der Charakterisierung von Mikrowellenpassiven in Schichtmedien unter Verwendung von berechnetem Elektromagnetismus. Ein Schichtmedium könnte ein Gegenstand wie eine Leiterplatte oder eine integrierte Schaltung sein, die Schicht für Schicht hergestellt wird, und deren Material zwischen den Schichten variiert. Ziel war es, die Hochfrequenzcharakteristiken von passiven Mikrowellenkomponenten wie Induktivitäten in solchen Medien zu berechnen.

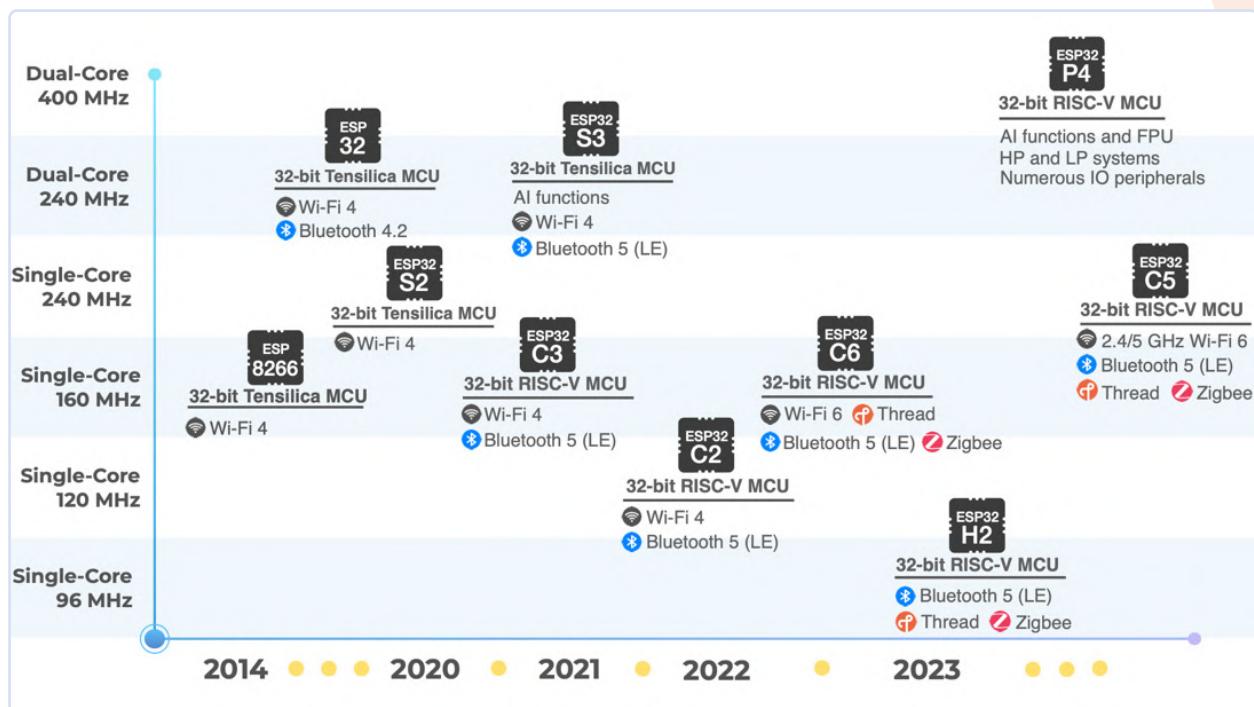
Wir sahen uns damals mit vielen Problemen konfrontiert: der Schwierigkeit, eine einzige Greensche Funktion zu erstellen, die sowohl das Nah- als auch das Fernfeld repräsentiert, wie man die Pole der Funktionen extrahiert und wie man die Sommerfeld-Transformationen der Funktionen vom Frequenzbereich in den räumlichen Bereich durchführt. Meine Arbeit war also in zwei Teile gegliedert. Der erste Teil bestand darin, zu zeigen, wie wir eine einzige Funktion erstellen können, um sowohl das Nah- als auch das Fernfeld darzustellen, und wie wir eine effiziente Berechnungsmethode für ihre Sommerfeld-Transformationen (so etwas wie die Laplace-Transformation, aber mit einem Bessel- oder Hankel-Kernel) entwickeln können. Wir wissen, dass diese Funktionen Singularitäten und Verzweigungsschnitte aufweisen. Es ist also, als würde man auf sich auf ein mathematisches Minenfeld mit einigen Zäunen begeben. Der Knackpunkt bestand darin, erstens die Singularitäten dieser Funktionen zu extrahieren und zweitens „synthetische Pole“ zu schaffen, die in der späteren Berechnung Lösungen in geschlossener Form haben, um die Konvergenz der Berechnung zu beschleunigen (ein Problem, das durch die Einbeziehung der Pole zur Singularitätsextraktion entstand). Der verbleibende kleine Teil der Funktionen, der nicht berücksichtigt wurde, wird dann berechenbar sein und schnell konvergieren.

Das letzte verbleibende Problem bestand darin, wie wir die Singularitäten im komplexen Bereich extrahieren können. Ich habe eine alternative Technik entwickelt, die ich „Cauchy-Tomographie“ nenne, um diese Singularitäten zu extrahieren, indem ich den Cauchy-Residuensatz verwende und ihre Lage mit Hilfe der „Generalized Pencil of Function“-Technik (GPOF) wiederherstelle.

Zufälligerweise verwende ich diese Technik auch heute noch manchmal beim Schaltungsentwurf, um die Übertragungsfunktion eines Systems auf der Grundlage seines Einschwingverhaltens zu berechnen. Es stellt sich heraus, dass das Einschwingverhalten eine Art Zeitbereichsintegrationsausgabe des Systemfrequenzgangs ist. Wenn man zwischen den beiden Bereichen umrechnet, kann man erkennen, wie die Systempole eine bestimmte Art von Einschwingverhalten erzeugen und umgekehrt.

Elektor: Ihre Antwort auf die Einladung von Elektor, die nächste Gastausgabe zu erstellen, war ein klares und sofortiges „Ja“. Wann haben Sie Elektor kennengelernt? Hat Elektor Sie und Espressif in irgendeiner Weise beeinflusst oder geprägt?

Teo Swee-Ann: Im Jahr 2015 sind wir zum ersten Mal auf Elektor gestoßen. Das Magazin zeichnete sich durch seine Fähigkeit aus, aufkommende Trends in der Branche zu erkennen und hervorzuheben. Ich glaube sogar, dass Elektor zu den Pionieren gehörte, die die Chips von Espressif vorstellten, vor allem den ESP8266, und damit eine wichtige Rolle bei der Popularisierung dieser Chips spielten. Wir wenden uns regelmäßig an Elektor als Inspirationsquelle und sind stets darauf bedacht, die sich entwickelnden Bedürfnisse und Interessen der Leserschaft zu erkennen. Darüber hinaus erfüllt Elektor eine unschätzbare Rolle als Umschlagplatz des Wissens für diejenigen, die in den Elektroniksektor einsteigen wollen. Elektor unterstreicht eine gemeinsame Verpflichtung, die wir in der Branche empfinden: Unser Wissen zu verbreiten und so dafür zu sorgen, dass die Elektronikbranche weiterhin neue Talente anzieht und fördert.



“

Eine unserer ersten Entscheidungen war, dass ein IoT-Chip idealerweise 600 DMIPS haben sollte. Dieses Leistungsniveau bietet ein ausgewogenes Verhältnis zwischen Leistung, die viele MCUs übertrifft, und Kosteneffizienz dank der Dual-Core-Architektur.

Elektor: Welche Vor- und möglichen Nachteile bringt Ihr technischer Hintergrund für Ihre derzeitige Rolle als CEO von Espressif mit sich?

Teo Swee-Ann: Ich sehe darin zumindest keinen großen Nachteil. Ich liebe es, an der Front zu sein, und ich arbeite immer noch an Schaltkreisen. Durch die Zusammenarbeit mit dem Team habe ich einen besseren Überblick über die Probleme, mit denen wir konfrontiert sind, und über die Schwierigkeiten bei der Entwicklung der einzelnen Funktionen. So kann ich die Ressourcen besser einteilen, die wichtigsten Probleme erkennen, mit dem Team kommunizieren und einen Konsens finden.

Elektor: Was sind Ihre Interessen außerhalb von Technik und Elektronik?

Teo Swee-Ann: Ich mache viel Musik, während ich über technische Probleme nachdenke. Irgendwie löst das Blockaden im Gehirn. Früher habe ich viel klassische Gitarre gespielt, und vor kurzem habe ich mit dem Cello angefangen. Ich gehe jetzt nach und nach Bachs Cello-Suiten durch und arbeite an der Verbesserung meiner Intonation.

Elektor: Sie haben Espressif im Jahr 2008 gegründet. Ihr erstes Produkt war der ESP8086, der 2013 auf den Markt kam. Lag der Hauptfokus des Unternehmens in diesen fünf Jahren auf diesem Produkt oder haben Sie auch andere Dienstleistungen angeboten?

Teo Swee-Ann: In den ersten Jahren war es ein sehr kleiner Betrieb. Ich arbeitete an Software, um eine Sprache zur Beschreibung von Schaltkreisen zu entwickeln. Daher auch der Name „Espressif“, der unsere Schaltkreisideen ausdrücken sollte. Software zu verkaufen ist jedoch schwierig, und wir wandelten uns zu einem Beratungsunternehmen, das analoge IPs für unsere Kunden entwickelt. Schließlich wurden wir von dem Konzept einer bevorstehenden technologischen Singularität inspiriert und beschlossen, IoT-Chips zu bauen, die als Nervensystem für ein solches intelligentes System fungieren würden.

Elektor: 2014 brachte Espressif seinen ersten IoT-SoC auf den Markt, den ESP8266EX. Auf welche Art von technischen Herausforderungen zielen Sie mit dieser Lösung ab?

Teo Swee-Ann: Zu dieser Zeit kursierten Ideen zur Entwicklung eines kostengünstigen WLAN-Chips. Wir beschlossen, dass wir diese Herausforderung annehmen könnten, indem wir alle externen HF-Komponenten in den Chip integrieren. In der Vergangenheit gab es Module, die mit 50...100 Komponenten überladen waren. Wir haben dieses Design revolutioniert, indem wir

den Balun, den Antennenschalter, den PA und den LNA direkt in den Chip integriert haben. Die größte Herausforderung bestand darin, die Spitzenleistung des Leistungsverstärkers von 27 dBm zu bewältigen. Ohne einen ausreichend robusten Schalter bestand die Gefahr, dass der LNA beschädigt wird.

Wir änderten auch die Art und Weise, wie die Leistungsverstärker intern angesteuert wurden. Anstelle von Induktivitäten verwendeten wir Baluns, was die Stabilität des Systems verbesserte. Außerdem luden die meisten WLAN-Chips in der Vergangenheit die Software-Images auf eine eher statische Weise in den Speicher. Wir entschieden uns für ein Cache-Design, so dass die Software stattdessen kontinuierlich aus dem Flash gelesen werden konnte. Das war zwar viel langsamer, aber es funktionierte. Außerdem haben wir einen Großteil der Logik des Systems in die Software und nicht in die Hardware gesteckt. Das hat uns bei der Entwicklung geholfen.

Der Höhepunkt dieser schrittweisen Innovationen führte zu einem äußerst kompakten Design. Unser Endprodukt kostete ein Viertel bis ein Drittel dessen, was unsere Konkurrenten anboten. Heute können wir mit Stolz feststellen, dass viele der von uns eingeführten Techniken heute zum Standard in der Branche gehören und erheblich zur Kostensenkung bei IoT-Chips beitragen.

Elektor: 2016 war ein wichtiges Jahr für Espressif. Erzählen Sie uns von der Entwicklung des ESP32 und seiner Veröffentlichung.

Teo Swee-Ann: Vor der Einführung des ESP32 war unser Ansatz eher spontan und weniger strukturiert. Der ESP32 markierte einen bedeutenden Wandel in unserer Denkweise, der uns dazu brachte, das Marketing systematischer anzugehen. Eine unserer ersten Entscheidungen war, dass ein IoT-Chip idealerweise 600 DMIPS haben sollte. Dieses Leistungsniveau bietet ein Gleichgewicht zwischen Leistung, die viele MCUs übertrifft, und Kosteneffizienz dank der Dual-Core-Architektur und dem effizienten Cache-System. Auf der Reise der Entwicklung des ESP32 gab es zahlreiche Diskussionen, Überlegungen und Überarbeitungen.

In dieser Phase haben wir unsere Teamstruktur formell festgelegt, unseren Arbeitsprozess verfeinert und unsere Arbeitsabläufe mit dem Design und den Spezifikationen des Produkts abgestimmt. Dies war eine Abkehr von unserem früheren, weniger formalen, „Garage Style“-Ansatz. Zeitgleich mit der Entwicklung des ESP32 startete unser Software-Team ESP-IDF. Entscheidungen, beispielsweise die über unsere Open-Source-Politik, wurden in dieser Zeit gefestigt und bildeten eine solide Grundlage für zukünftige Entwicklungen. Zusammenfassend lässt sich sagen, dass die Entwicklung des ESP32 die zunehmende Bedeutung der Software unterstreicht, anstatt sich



nur auf die Hardware zu konzentrieren. Dieser Übergang wurde durch das aktive Engagement und das unschätzbare Feedback unserer engagierten Entwickler-Community weiter bereichert.

Elektor: Erzählen Sie uns etwas über die Matter-kompatiblen Lösungen von Espressif. Matter scheint im Jahr 2023 und bis ins Jahr 2024 hinein eine Art Priorität zu sein, richtig?

Teo Swee-Ann: Bislang hat die Smart-Home-Branche ihr Potenzial noch nicht ausgeschöpft, und es ist für die Verbraucher immer noch schwierig, die Produkte effektiv zu nutzen. Die Anwendungsmöglichkeiten sind begrenzt und die Erfahrung, die sie bieten, ist fragmentiert. Matter versucht, das zu ändern, indem es die Geräte dazu bringt, dieselbe Sprache zu sprechen, und es ist sehr hoffnungsvoll zu sehen, dass die meisten der großen Akteure zusammenkommen, um diese Probleme zu lösen. Am wichtigsten ist, dass wir positive Reaktionen von den Kunden sehen. Daher ist dies eine natürliche Priorität für uns.

Der Ansatz von Espressif zur Entwicklung von Matter-Lösungen ist nicht auf Hardware beschränkt. Wir haben zwar verschiedene Chips, die Matter über Thread- und WLAN-Protokolle unterstützen, aber wir gehen darüber hinaus, indem wir spezifische Kundenprobleme bei der Matter-Einführung verstehen und versuchen, eine Lösung dafür zu entwickeln. Unsere ESP-ZeroCode-Module sowie unsere Dienste zur Zertifikatsbereitstellung und Zertifizierungshilfe sind gute Beispiele für diesen Ansatz.

Elektor: Was waren die Hintergründe für die Entwicklung von ESP RainMaker?

Teo Swee-Ann: Die Konzeption von ESP RainMaker zielte auch darauf ab, die Probleme der Kunden zu lösen. In der Vergangenheit haben wir erlebt, dass Kunden entweder das Rad neu erfunden haben, wenn es um den Aufbau einer IoT-Cloud-Plattform ging, oder Produkte auf der Grundlage von Cloud-Plattformen von Drittanbietern mit minimaler Differenzierung und Kontrolle über die Daten entwickelt haben. Andererseits war die serverlose Cloud-Architektur so vielversprechend, um eine solche Cloud aufzubauen, ohne sich um die Verwaltung der Infrastruktur kümmern zu müssen. Allerdings ist es für Gerätehersteller immer noch schwierig, eine solche Plattform von Grund auf aufzubauen. Deshalb haben wir ESP RainMaker mit der Vision entwickelt, eine Cloud-Plattform für Kunden zu schaffen, die ihnen die volle Kontrolle und Anpassungsfähigkeit gibt, um ihre eigene IoT-Cloud mit deutlich weniger technischen Investitionen aufzubauen.

Elektor: AIoT-Systeme und -Produkte übertragen viele Daten. Dies führt natürlich zu Bedenken hinsichtlich des Datenschutzes und der Privatsphäre. Glaubt Espressif, dass der Markt für AIoT-Produkte „boomen“ wird, sobald internationale Standards für Datenschutz und Privatsphäre vereinbart sind?

Teo Swee-Ann: In der gegenwärtigen AIoT-Landschaft sind Datenschutz und regulatorische Fragen zwar von Bedeutung, aber die dringendere Herausforderung, die wir festgestellt haben, ist der Mangel an Entwicklern von eingebetteter Software. Darüber hinaus wird die Situation durch die Fragmentierung des Marktes noch komplizierter, was zu einer erhöhten Komplexität führt. Bei Espres-

sif gehen wir diese Herausforderungen aktiv an. Wir haben viele Lösungen eingeführt. Zum Beispiel haben wir ESP-ZeroCode für die Entwicklung von Matter-Anwendungen, ESP-SkaiNet für die lokale KI-Sprachbefehlserkennung, ESP-RainMaker für die Erstellung Ihrer eigenen Cloud-Plattform, ESP-ADF für die Erstellung Ihres eigenen WLAN-Audiosystems und so weiter.

Unser Ziel ist es, unsere Kunden mit den Werkzeugen auszustatten, die sie benötigen, um Lösungen zu erstellen, ohne bei Null anfangen zu müssen. Das soll Effizienz und Geschwindigkeit in der Entwicklung gewährleisten. Für einen signifikanten positiven Wandel in der AIoT-Landschaft ist es unserer Meinung nach entscheidend, die Fragmentierung zu verringern. Es ist unerlässlich, dass verschiedene Plattformen zusammenarbeiten und einen einheitlichen Standard fördern. Der Matter-Standard ist ein vielversprechender Schritt in diese Richtung, doch sein volles Potenzial und seine langfristigen Auswirkungen müssen erst noch ermittelt werden.

Elektor: Hat die weltweite Chip-Knappheit die Verwendung von Espressif-basierter Technologie in kommerziellen Produkten gefördert? Wenn ja, wie können diese Produkte zur weiteren Etablierung der Marke Espressif beitragen?

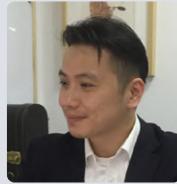
Teo Swee-Ann: Während der schwierigen Zeit der weltweiten Chip-Knappheit verfolgte Espressif einen strategischen Ansatz, indem wir die Chip-Verkäufe an unsere geschätzten Kunden rationierten. Diese Entscheidung wurde von vielen unserer Kunden positiv aufgenommen. Erstens konnten sie so sicherstellen, dass sie weiterhin eine kontinuierliche Versorgung mit Chips erhielten, auch wenn es sich um begrenzte Mengen handelte. Zweitens verhinderte diese Strategie, dass unsere Kunden als Reaktion auf die Verknappung ihre Lagerbestände vorschnell aufstockten. Wichtig ist auch, dass wir in dieser Zeit unsere Preise nicht erhöht haben, so dass sich die Marktvolatilität nicht negativ auf unsere Preisstruktur ausgewirkt hat. Ich glaube, dass unsere Stabilität und unser bescheidenes Wachstum im Jahr 2023 die Wirksamkeit und Voraussicht dieser Unternehmenspolitik widerspiegeln.

Elektor: In einem Interview mit Elektor vor einigen Jahren sagten Sie, dass KI das nächste große Ding sein würde, und das hat sich bewahrheitet. Was glauben Sie, was das nächste große Ding in der Elektronik sein wird?

Teo Swee-Ann: Im Bereich der Elektronik gibt es zwei verschiedene Entwicklungen, die wir in Betracht ziehen müssen. Der erste Weg führt zu einem immersiveren Ökosystem virtueller oder gemischter Realität. Stellen Sie sich eine Zukunft vor, in der wir in den nächsten Jahrzehnten Implantate besitzen könnten, die nicht nur die traditionellen Sinne wie Sehen und Hören erweitern, sondern auch neue sensorische Erfahrungen oder kognitive Wege eröffnen, die von der natürlichen Evolution noch nicht erschlossen wurden. Dies würde bedeuten, dass unsere Emotionen, kognitiven Fähigkeiten und Sinneserfahrungen durch KI erheblich verbessert oder moduliert werden könnten. Die Verwirklichung dieser Vision erfordert erhebliche technologische Innovationen, insbesondere im Hinblick auf höhere Rechenkapazitäten bei geringerem Stromverbrauch und niedrigeren Spannungen sowie kompakten Formfaktoren.

“

Mit Blick auf die Zukunft bleibt unser Hauptziel die Entwicklung bahnbrechender Lösungen, die Energieeffizienz und kleinere Formfaktoren in den Vordergrund stellen.



Über Teo Swee-Ann

Teo Swee-Ann ist der Gründer und CEO des in Shanghai ansässigen Halbleiterunternehmens Espressif Systems. Er hat einen Master-Abschluss in Elektrotechnik von der National University of Singapore. Teo Swee-Ann arbeitete bei den US-Chipherstellern Transilica und Marvell Technology Group sowie beim chinesischen Montage Technology, bevor er 2008 sein eigenes Unternehmen gründete.

Die zweite Richtung hingegen legt den Schwerpunkt auf die Anwendung von KI im Bereich des Elektronikdesigns. Um einen Vergleich zu ziehen: Wenn KI in der Lage ist, Fahrzeuge autonom zu steuern, dann hat sie sicherlich auch das Potenzial, anspruchsvolle Schaltungen zu entwerfen. Damit wird eine bemerkenswerte Herausforderung in unserem Bereich angegangen, nämlich die Tendenz zu Designs, die technisch zu komplex sind. Die Einbindung von KI in den Designprozess bietet die Möglichkeit, diese Designs zu verfeinern, sie rationeller und effektiver zu gestalten und so die Realisierung des ersten Entwicklungsansatzes zu beschleunigen.

Elektor: Was ist Ihre Vision für Espressif? Wo sehen Sie das Unternehmen in, sagen wir, fünf Jahren?

Teo Swee-Ann: Wir bei Espressif positionieren uns in erster Linie als ein führendes AIoT-Chip-Unternehmen. Allerdings

gehen unsere Fähigkeiten weit über die reine Hardware hinaus. Wir haben einen festen Stand in der Softwareentwicklung mit unserem proprietären Softwaresystem ESP-IDF und umfangreichen Lösungsframeworks für Cloud-Integration, Edge-KI, Mesh-Netzwerken, Audio- und Kameraanwendungen und vielem mehr. Was uns wirklich auszeichnet, ist unsere blühende Community und unser Ökosystem, in dem Profis und Amateure zusammenkommen, um bahnbrechende Ideen zu entwickeln und miteinander zu teilen. Mit Blick auf die Zukunft bleibt unser Hauptziel die Entwicklung bahnbrechender Lösungen, bei denen die Energieeffizienz und kleinere Formfaktoren im Vordergrund stehen. Im Wesentlichen hoffen wir, dass Espressif für ein unverwechselbares Ethos und eine fortschrittliche Einstellung zu Technik, Innovation und der Förderung eines Gemeinschaftsgeistes der Zusammenarbeit steht. ☈

Übersetzung von Matze Schrumpf -- 230614-02

WEBLINK

[1] C. Valens, „The Reason Behind the Hugely Popular ESP8266?: Interview with Espressif Founder and CEO Teo Swee Ann on the new ESP32“, Elektor Business, 1/2018.: <https://www.electormagazine.com/magazine/elektor-63>



Erste Schritte mit ESP-DL

Das Deep-Learning-SDK von Espressif



Der ESP32-S3 verfügt über KI-Beschleunigung und kann in verschiedenen Machine-Learning-Applikationen verwendet werden. ESP-DL ist das Deep-Learning-SDK von Espressif, mit dem Sie die Vorteile der KI-Beschleunigungsbefehle nutzen können, um optimierte ML-Modelle zu erstellen. Dieses SDK bietet Beispiele für die Erkennung von Menschen und Katzenköpfen sowie für die Gesichtserkennung. ESP-DL unterstützt jetzt auch den TVM-Compiler, mit dem Sie TensorFlow-, PyTorch- oder ONNX-Modelle verwenden können. Espressif bietet auch eine Portierung von Googles TensorFlow Lite Micro SDK mit ESP32-S3 Beschleunigungsunterstützung, mit der Sie gewöhnliche Modelle in TensorFlow Lite auf dem ESP32-S3 verwenden können.

<https://github.com/espressif/esp-dl>

<https://github.com/espressif/tflite-micro-esp-examples>



ESP32 mit Wokwi simulieren

Der virtuelle Zwilling Ihres Projekts

Von Uri Shaked, Wokwi

Wokwi ist ein Simulator für eingebettete Systeme und IoT-Geräte. Er bietet eine Online-Umgebung, in der Sie Ihre ESP32-Projekte testen, debuggen und mit anderen teilen können, ohne die physische Hardware zu benötigen. Der Simulator läuft direkt in Ihrem Webbrowser und kann alle Chips der ESP32-Familie simulieren: den klassischen ESP32 sowie die Varianten S2, S3, C3, C6 und H2. Weitere Mikrocontroller-Familien sind ebenfalls verfügbar.

Wokwi ermöglicht es Ihnen, einen virtuellen Zwilling Ihres Projekts zu erstellen: Sie können eine Vielzahl von Eingabe- und Ausgabegeräten [1] simulieren, zum Beispiel LCD-Bildschirme, Sensoren, Motoren, LEDs, Tasten, Lautsprecher und Potentiometer, und Sie können sogar Ihre eigenen Simulationsmodelle für neue Bauteile und -gruppen erstellen. Der Simulator ermöglicht es Ihnen, schneller zu iterieren, an Ihrem Firmware-Code zu arbeiten, auch wenn die Hardware

nicht verfügbar ist, leistungsstarke Debugging-Tools zu verwenden, Ihre Projekte zu teilen und mit anderen Ingenieuren auf einfache Weise zusammenzuarbeiten.

Beginnen Sie Ihre Wokwi-Reise

Sie können Ihre bevorzugte Programmiersprache mit Wokwi verwenden. Wenn Sie ein Anfänger sind, sind MicroPython oder Arduino Core wahrscheinlich eine gute Wahl. Für Profis und fortgeschrittene Benutzer können Sie das ESP-IDF direkt verwenden, die Programmiersprache Rust einsetzen oder ein eingebettetes RTOS wie Zephyr oder NuttX verwenden. Wir empfehlen, sich einige der vorhandenen Beispiele anzuschauen, um Ideen zu bekommen, was man mit Wokwi bauen könnte:

- MicroPython [2]
- ESP32 mit Arduino Core [3]
- Rust [4]
- DeviceScript (TypeScript für ESP32) [5]

Nachdem Sie ein Beispiel geöffnet haben, drücken Sie die grüne Play-Taste, um die Simulation zu starten und mit dem Projekt zu interagieren. Einige der Projekte haben auch eine README-Datei, in der Sie etwas über

```

WOKWI SAVE SHARE Simon Says ESP32-C3 by urish Docs 🌐

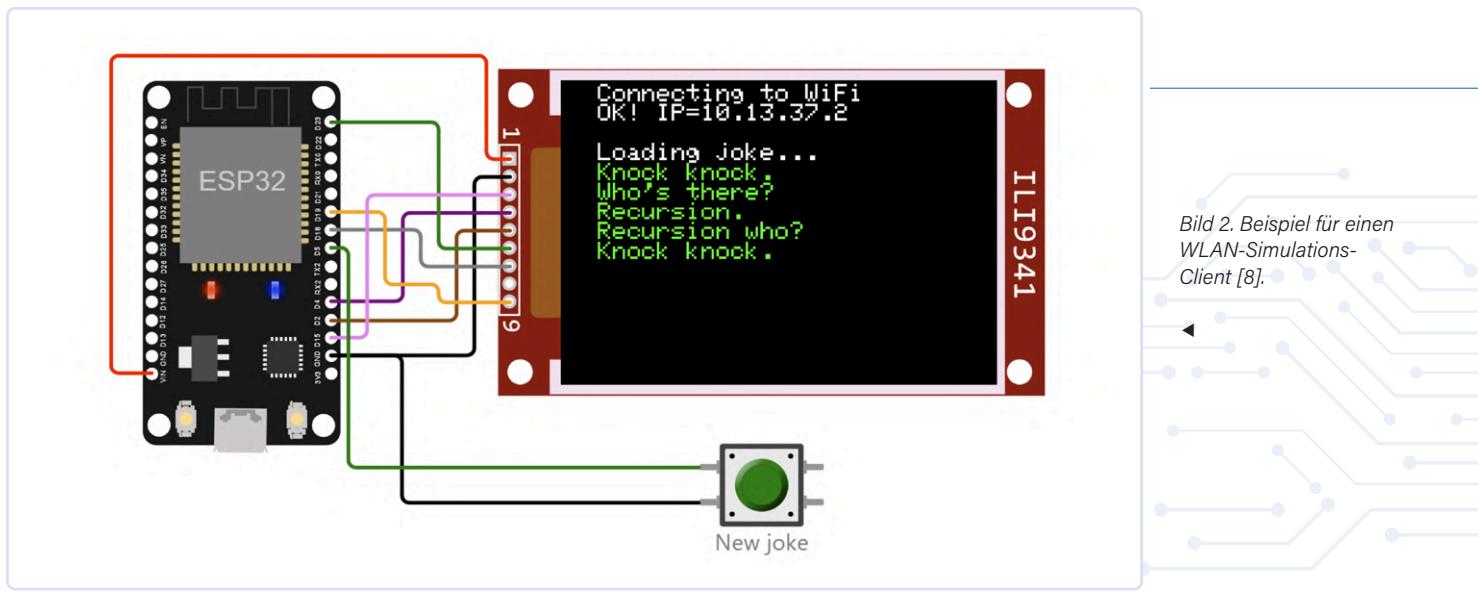
esp32-c3-simon-says.ino README.md pitches.h diagram.json Library Manager

Code Editor
Diagram Editor

/*
1  Simon Game for ESP32-C3 with Score display
2  Copyright (C) 2023, Uri Shaked
3
4  Released under the MIT License.
5
6
7 */
8
9 #include "pitches.h"
10
11 /* Define pin numbers for LEDs, buttons and speaker: */
12 const uint8_t buttonPins[] = {0, 1, 2, 3};
13 const uint8_t ledPins[] = {8, 7, 6, 5};
14 #define SPEAKER_PIN 10
15
16 // These are connected to 74HC595 shift register (used to show game score):
17 const int LATCH_PIN = 18; // 74HC595 pin 12
18 const int DATA_PIN = 19; // 74HC595 pin 14
19 const int CLOCK_PIN = 9; // 74HC595 pin 11
20
21 #define MAX_GAME_LENGTH 100
22
23 const int gameTones[] = { NOTE_G3, NOTE_C4, NOTE_E4, NOTE_G5 };
24
25 /* Global variables - store the game state */
26 uint8_t gameSequence[MAX_GAME_LENGTH] = {0};

```

Bild 1. Die Benutzeroberfläche von Wokwi, Spiel „Simon Says“.



das Projekt und seine Verwendung erfahren können. Um ein neues Projekt von Grund auf zu starten, gehen Sie zu [6], wählen Sie Ihren Mikrocontroller und Ihre Programmiersprache und klicken Sie auf Start.

Die Wokwi-Benutzeroberfläche

Die Web-Version von Wokwi ist in zwei Bereiche aufgeteilt: Im linken Bereich bearbeiten Sie den Quellcode Ihrer Firmware (*Code-Editor*), und im rechten Bereich zeichnen Sie das Projektdiagramm, indem Sie verschiedene Geräte hinzufügen und mit Ihrem Mikrocontroller verbinden (*Diagramm-Editor*), wie in **Bild 1** dargestellt.

Fügen Sie dem Diagramm neue Bauteile hinzu, indem Sie auf die runde blaue + -Schaltfläche klicken und das gewünschte Bauteil aus der Liste auswählen. Ziehen Sie die Teile an die gewünschte Position. Um einen Draht zu zeichnen, klicken Sie auf den Startpin und dann auf den Zielpin. Wenn Sie möchten, dass der Draht in eine bestimmte Richtung verlegt wird, können Sie ihn lenken, indem Sie nach der Auswahl des ersten Pins auf die gewünschte Stelle klicken. Wenn Sie ein saubereres Ergebnis wünschen, schalten Sie das Raster ein, indem Sie G drücken. Dadurch werden alle Teile und die Verdrahtung an einem 0,1"-Raster (2,54 mm), dem Standardabstand von Stiftleisten ausgerichtet. Einige nützliche Tastenfunktionen: D zum Duplizieren des ausgewählten Teils, R, um es zu drehen, die Ziffern 0...9, um bequem die Farbe eines Kabels, einer LED oder eines Druckknopfes festzulegen, und das Drücken der Umschalttaste beim Ziehen mit der Maus, um mehrere Teile auf einmal auszuwählen. Weitere Tastatur-Shortcuts finden Sie in der Anleitung zum Diagrammeditor [7].

Der Bibliotheksmanager

Verwenden Sie den Bibliotheksmanager, um Ihrem Projekt schnell eine beliebige Arduino-Standardbibliothek hinzuzufügen. Zahlende Benutzer von Wokwi können auch eine beliebige benutzerdefinierte Arduino-Bibliothek hochladen und in ihr Projekt einfügen.

Für andere Programmiersprachen können Sie Bibliotheken durch Bearbeiten der Projektkonfigurationsdateien (zum Beispiel *Cargo.toml* für Rust) oder durch direktes Hinzufügen des Quellcodes der Bibliothek zu Ihrem Projekt (zum Beispiel für MicroPython) einfügen.

WLAN-Simulation

Der ESP32 hat eine eingebaute WLAN-Funktionalität, was ihn zu einer beliebten Wahl für IoT-Projekte macht. Wokwi simuliert die WLAN-Funktionalität des Chips. Der Simulator (**Bild 2**) bietet einen virtuellen Zugangspunkt namens *Wokwi-GUEST*. Es ist ein offener Zugangspunkt (also kein Passwort erforderlich). Der Zugangspunkt ist mit dem Internet verbunden und ermöglicht Ihren simulierten Projekten die Verbindung zu HTTP-Servern (**Bild 3**), MQTT-Brokern und anderen Cloud-Diensten wie Firebase, Things-Speak, Blynk und Azure IoT.

Zahlende Nutzer können auch ein spezielles IoT-Gateway [10] auf ihren Computern laufen lassen, das es ihnen ermöglicht, von der Simulation aus eine Verbindung zu lokalen Servern herzustellen sowie von ihrem Computer aus eine Verbindung zu einem HTTP- oder einer anderen Art von Server herzustellen,

Bild 2. Beispiel für einen WLAN-Simulations-Client [8].

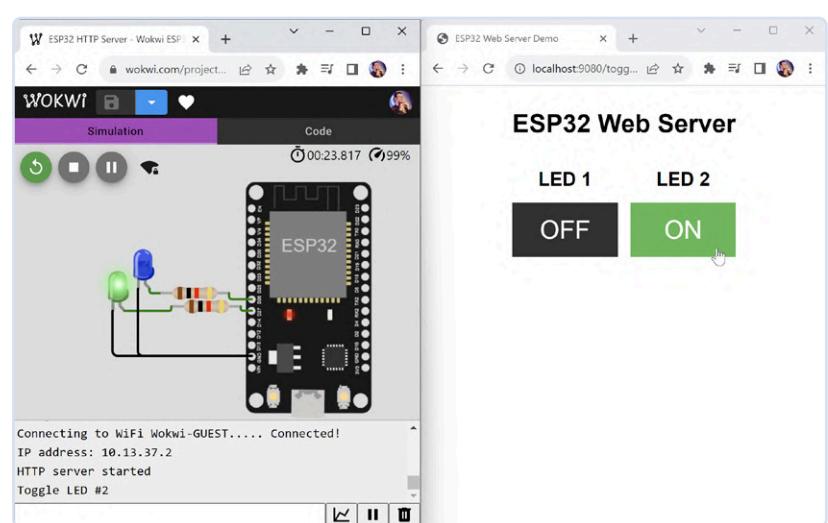


Bild 3. Beispiel für einen WLAN-Simulations-Webserver [9].

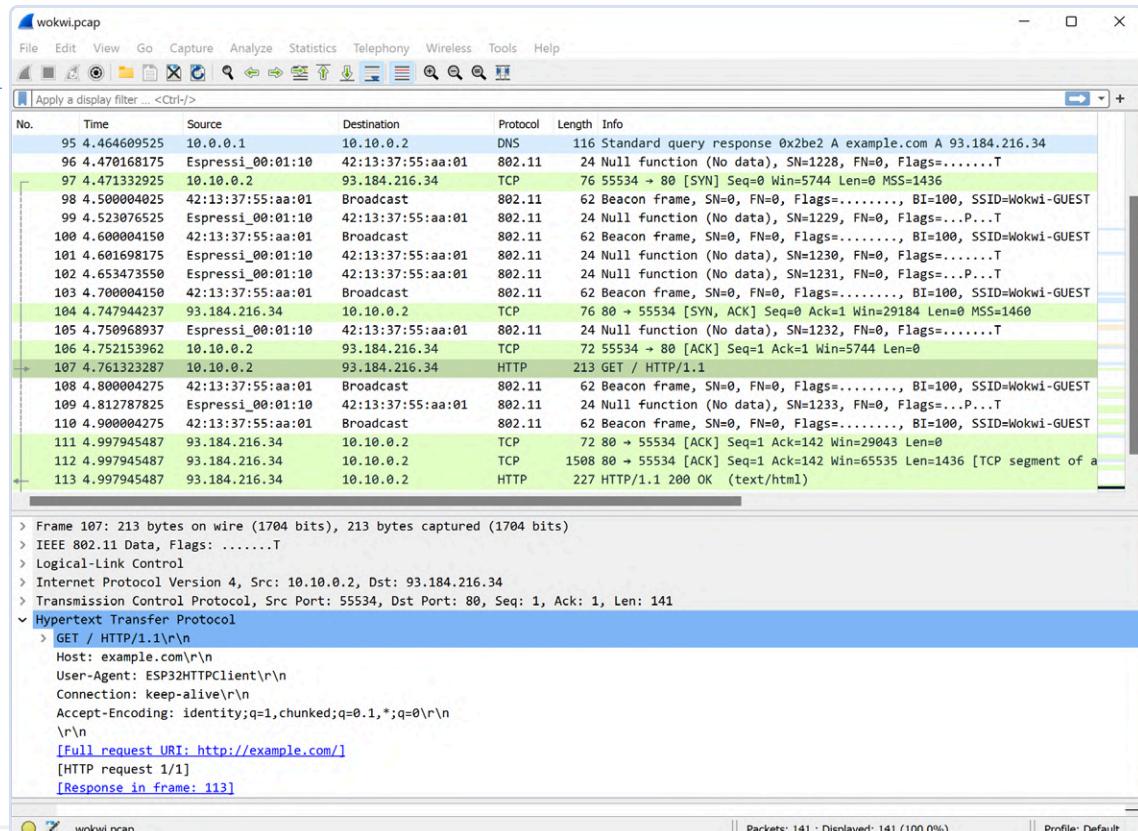


ESP32 Web Server

LED 1 LED 2

OFF ON

Bild 4. Ansicht der WLAN-Verkehrserfassung.



der innerhalb des Simulators läuft. Unter der Oberfläche simuliert Wokwi einen kompletten Netzwerkstack: angefangen bei der untersten 802.11-MAC-Schicht, über die IP- und TCP/UDP-Schichten bis hin zu Protokollen wie DNS, HTTP, MQTT, CoAP, und so weiter. Sie können den unbearbeiteten Netzwerkverkehr [11] aufzeichnen und in einem Netzwerkprotokoll-Analyseprogramm wie Wireshark anzeigen, wie in Bild 4 dargestellt.

Pin-Funktions-Debugger

Der ESP32 hat eine flexible Peripherie-Pinbelegung: Die Software kann definieren, welches Peripheriege-

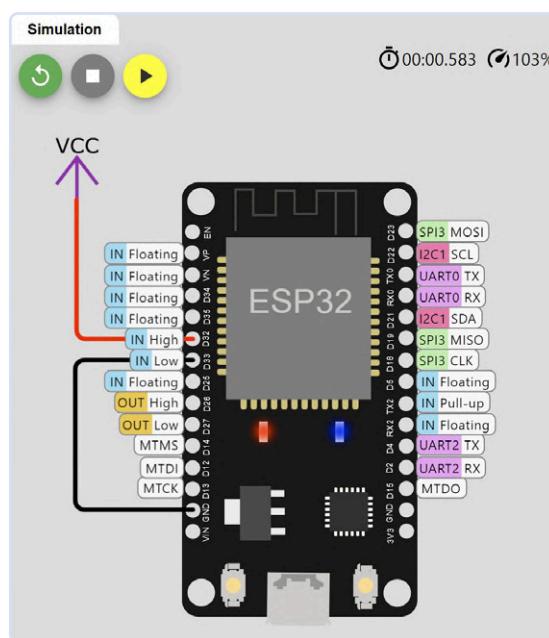
rät mit welchem der Pins verbunden ist, indem IO MUX, GPIO Matrix und die Low-Power/RTC-Peripherie konfiguriert werden. Die Definition der Pinbelegung in der Software kann sehr praktisch sein, macht es aber auch schwieriger, die tatsächliche Pinbelegung für Ihre Firmware herauszufinden. Glücklicherweise brauchen Sie mit Wokwi nur die Simulation anzuhalten, und Sie können sofort die aktuelle Funktion und den Zustand jedes Pins sehen, wie man in Bild 5 sehen kann.

Integration von Visual Studio Code

Die Verwendung von Wokwi in Ihrem Webbrowser eignet sich hervorragend zum Lernen, zum schnellen Prototyping und zum Teilen Ihrer Arbeit. Für komplexe Projekte empfehlen wir jedoch die Verwendung von Wokwi zusammen mit Ihrer Standard-IDE und Ihren Entwicklungswerkzeugen. Wokwi lässt sich nahtlos in Visual Studio Code integrieren. Alles, was Sie dazu tun müssen, ist es, die Erweiterung *Wokwi for VS Code* [12] zu installieren und eine einfache Konfigurationsdatei [13] zu erstellen, die Wokwi mitteilt, wo die kompilierte Firmware zu finden ist. Sie können Wokwi auch in den VS-Code-Debugger integrieren, indem Sie der Konfiguration einfach ein paar Zeilen hinzufügen [14]. Im Gegensatz zu echter Hardware verfügt Wokwi über eine unbegrenzte Anzahl von Haltepunkten, so dass Sie effizienter debuggen können (Bild 6).

Für IoT-Projekte können Sie TCP-Port-Weiterleitungen definieren [15]. Dadurch können Sie von Ihrem Computer aus eine Verbindung zu einem Webserver (oder einer anderen Art von TCP-Server) herstellen, der innerhalb des simulierten ESP32-Chips läuft.

Bild 5. Simulation pausiert, mit Statusanzeige für jeden Pin.



```

main.c
...
void led_task(void *pvParameter)
{
    while (1) {
        if (xEventGroupGetBits(wifi_event_group) & CONNECTED_BIT) {
            // We are connected - LED on
            gpio_set_level(LED_RED, 1);
            vTaskDelay(200 / portTICK_RATE_MS);
        } else {
            // We are connecting - blink fast
            gpio_set_level(LED_RED, 0);
            vTaskDelay(200 / portTICK_RATE_MS);
            gpio_set_level(LED_RED, 1);
            vTaskDelay(200 / portTICK_RATE_MS);
        }
    }
}

// Main task
void main_task(void *pvParameter)
{
    tcpip_adapter_ip_info_t ip_info;
    // wait for connection
    xEventGroupWaitBits(wifi_event_group, CONNECTED_BIT, false, true, portMAX_DELAY);
    printf("Connected!\n");

    // print the local IP address
    ESP_ERROR_CHECK(tcpip_adapter_get_ip_info(TCPIP_ADAPTER_IF_STA, &ip_info));
    printf("IP Address: %s\n", ip4addr_ntoa(&ip_info.ip));
    printf("Subnet mask: %s\n", ip4addr_ntoa(&ip_info.netmask));
    printf("Gateway: %s\n", ip4addr_ntoa(&ip_info.gw));
    printf("You can connect now to any webserver online! :-)\n");

    while (1) {
        vTaskDelay(1000 / portTICK_RATE_MS);
    }
}

```

Waiting for connection to the WiFi network...
Connecting to Wokwi-GUEST...
Connected!
IP Address: 10.13.37.2
Subnet mask: 255.255.255.0
Gateway: 10.13.37.1
You can connect now to any webserver online! :-)

Espressif-IDE-Integration

Diejenigen unter Ihnen, die am liebsten mit der Espressif-IDE arbeiten, können auch eine Startkonfiguration einrichten, um Ihr Projekt im Simulator zu starten. Die Ausgabe Ihres Programms am UART wird direkt in die IDE-Konsole geleitet. Eine vollständige Anleitung finden Sie unter *How to Use Wokwi Simulator with Espressif-IDE* [16].

Benutzerdefinierte Chips

Mit benutzerdefinierten Chips können Sie neue Bauteile in Wokwi erstellen und seine Funktionalität erweitern. Sie können neue Sensoren, Anzeigen, Speicher, Testinstrumente (**Bild 7**) erstellen und sogar Ihre eigene benutzerdefinierte Hardware simulieren. Um einen benutzerdefinierten Chip zu erstellen, müssen Sie die Pinbelegung in einer JSON-Datei definieren und dann Code schreiben, um die Logik des Chips zu implementieren. Der Code ist in der Regel in C geschrieben, und die API ähnelt den üblichen Hardware-Abstraktionsschichten (HAL) für eingebettete Chips, so dass sich Firmware-Entwickler wie zu Hause fühlen sollten. Es gibt auch experimentelle Unterstützung für andere Sprachen wie Rust, AssemblyScript und Verilog. Weitere Informationen und Beispiele finden Sie in der Chips-API-Dokumentation [17].

Wokwi für CI

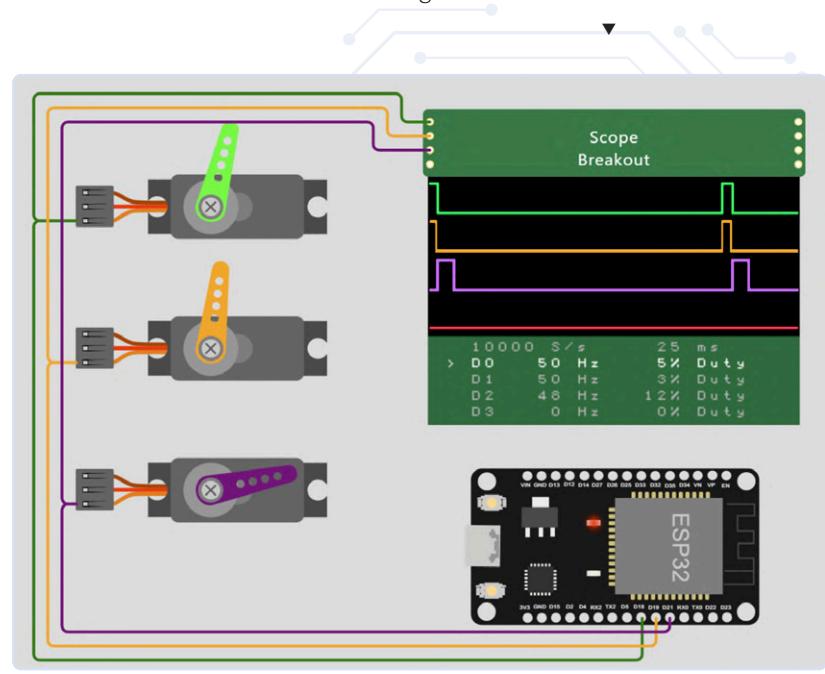
Die Simulation kann eine enorme Zeiterparnis beim Prototyping bedeuten, aber auch bei der weiteren Entwicklung Ihres Produkts helfen, Fehler zu finden. Kontinuierliche Integration (CI) ist eine moderne Softwareentwicklungsmethode: Erstellen und testen Sie Ihr Projekt, sobald Sie den Code ändern. Das Testen mit echter Hardware ist jedoch sehr zeitaufwändig und schwer zu automatisieren.

Noch komplizierter wird es, wenn Sie eine vollständige Systemintegration wünschen. Stellen Sie sich nur einmal vor, wie Sie automatisierte WLAN-Tests einrichten oder das Bild erfassen würden, das Ihr Projekt auf einem LCD-Bildschirm anzeigen. Wie viel Arbeit würde es machen, diese Einrichtung robust und zuverlässig zu machen?

Wokwi for CI nimmt Ihnen diese ganze Komplexität ab. Wenn Sie GitHub Actions verwenden, können Sie mit `wokwi-ci-action` [18] die Simulation mit nur wenigen Zeilen Code in Ihren bestehenden Workflow integrieren. Für andere CI-Umgebungen bietet `wokwi-cli` [19] eine einfache Befehlszeilenschnittstelle zum Ausführen und Testen Ihrer Firmware in der Simulation. Der CI Simulation Server ist als verwalteter Cloud-Service und auch für den Einsatz vor Ort verfügbar.

Bild 6. Fehlersuche in einem ESP-IDF-Projekt mit VS Code und Wokwi.

Bild 7. Ein digitales Oszilloskop, das ein Benutzer mit der Custom-Chips-API erstellt hat.



```

1 name: Pushbutton counter test
2 version: 1
3 author: Uri Shaked
4
5 steps:
6   - wait-serial: 'Pushbutton Counter' <-- Red arrow pointing to the screenshot
7
8   # Press once
9   - set-control:
10    part-id: btn1
11    control: pressed
12    value: 1
13   - delay: 100ms
14   - set-control:
15    part-id: btn1
16    control: pressed
17    value: 0
18   - delay: 200ms
19
20   # Press 2nd time
21   - set-control:
22    part-id: btn1
23    control: pressed
24    value: 1
25   - delay: 100ms
26   - set-control:
27    part-id: btn1
28    control: pressed
29    value: 0
30   - delay: 200ms
31
32   # Press for the 3rd time
33   - set-control:
34    part-id: btn1
35    control: pressed
36    value: 1
37   - wait-serial: 'Button pressed 3 times' <-- Red arrow pointing to the screenshot

```

build-and-test
succeeded 1 minute ago in 1m 15s

> Build PlatformIO Project 50s

< Test with Wokwi 5s

- 1 ► Run wokwi/wokwi-ci-action@v1
- 14 ► Run wget -q -O /usr/local/bin/wokwi-cli <https://github.com/wokwi/wokwi-cli/releases/latest/download/wokwi-cli-linuxstatic-x64>
- 25 ► Run wokwi-cli --timeout "10000" --expect-text "" \
- 38 Wokwi CLI v0.6.4 (8fa2d2b7b70f)
- 39 Connected to Wokwi Simulation API 1.0.0-20230804-gf0f4bfc7
- 40 Starting simulation...
- 41 ets Jul 29 2019 12:21:46
- 42
- 43 rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
- 44 configsip: 0, SPIWP:0xee
- 45 clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
- 46 mode:DIO, clock div:2
- 47 load:0x3fff0030,len:1156
- 48 load:0x40078000,len:11456
- 49 ho 0 tail 12 room 4
- 50 load:0x40080400,len:2972
- 51 entry 0x400805dc
- 52 Pushbutton Counter
- 53 [Pushbutton counter test] Expected text matched: "Pushbutton Counter"
- 54 [Pushbutton counter test] delay 100ms
- 55 Button pressed 1 times
- 56 [Pushbutton counter test] delay 200ms
- 57 [Pushbutton counter test] delay 100ms
- 58 Button pressed 2 times
- 59 [Pushbutton counter test] delay 200ms
- 60 Button pressed 3 times
- 61 [Pushbutton counter test] delay 200ms
- 62 [Pushbutton counter test] Expected text matched: "Button pressed 3 times"
- 63 [Pushbutton counter test] Scenario completed successfully

▲ Automatisierungsszenarien

Bild 8. Der Code des Automatisierungsszenarios [20] (links) mit der entsprechenden GitHub Action-Ausgabe (rechts).

Automatisierungsszenarien (**Bild 8**) ermöglichen Ihnen die Durchführung komplexer Tests und Assertionen für Ihre Firmware. Sie zu schreiben ist einfach: Jedes Szenario ist eine YAML-Datei, die eine Liste von Befehlen enthält, zum Beispiel das Drücken einer Taste, das Setzen eines Temperatursensorwerts oder die Suche nach einer Zeichenfolge in der seriellen Ausgabe der Firmware.

Beschränkungen

Wokwi hat viele Möglichkeiten, aber auch einige Einschränkungen, die Sie kennen sollten. Der Schwerpunkt des Simulators liegt auf der Ausführung von eingebettetem Firmware-Code. Die Simulations-Engine ist hauptsächlich digital, mit begrenzter Unterstützung für analoge Simulation. Das bedeutet, dass Wokwi Sie nicht ermahnen wird, wenn Sie einen strombegrenzenden Widerstand vergessen, und auch der „magische Rauch“ wird nicht aus der Schaltung freigesetzt (eines Tages vielleicht). ▶

RG – 230523-02

Über den Autor

Uri Shaked ist Maker, sein Leben lang. Derzeit arbeitet er an Wokwi, einer Online-Simulationsplattform für IoT und eingebettete Systeme, und an Tiny Tapeout, das die Herstellung kundenspezifischer ASICs erschwinglich und zugänglich macht.



Passende Produkte

- **ESP32-C3-DevKitM-1**
www.elektor.de/20324
- **Koen Vervloesem, Getting Started with ESPHome (Elektor, 2021)**
Buch, Paperback, englisch: www.elektor.de/19738
E-Buch, PDF, englisch: www.elektor.de/19739



Haben Sie Fragen oder Kommentare?

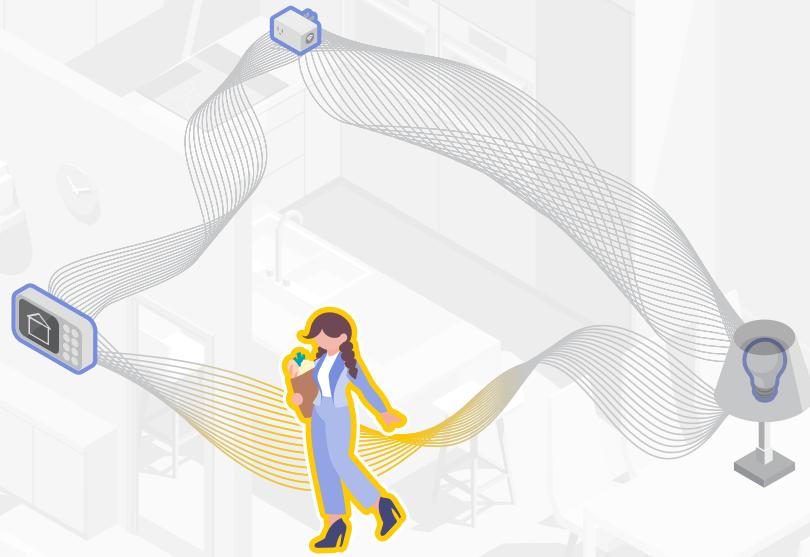
Haben Sie technische Fragen oder Kommentare zu diesem Artikel? Kontaktieren Sie Elektor unter redaktion@elektor.de.

WEBLINKS

- [1] Wokwi, unterstützte Hardware: <https://docs.wokwi.com/getting-started/supported-hardware>
- [2] Wokwi Online Embedded-Python-Simulator: <https://wokwi.com/micropython>
- [3] Wokwi Online ESP32-Simulator: <https://wokwi.com/esp32>
- [4] Wokwi Online Embedded-Rust-Simulator: <https://wokwi.com/rust>
- [5] Wokwi Online DeviceScript-Simulator: <https://wokwi.com/devicescript>
- [6] Neues Projekt mit dem Arduino Uno, Startseite: <https://wokwi.com/projects/new>
- [7] Diagrammeditor mit Tastaturkürzeln: <https://docs.wokwi.com/guides/diagram-editor#keyboard-shortcuts>
- [8] _esp32-jokes-api_ Beispiel einer WLAN-Simulation: <https://wokwi.com/projects/342032431249883731>
- [9] Beispiel eines ESP32-HTTP-Servers: <https://wokwi.com/projects/320964045035274834>
- [10] ESP32-WiFi-Netzwerk: The Private Gateway Application: <https://docs.wokwi.com/guides/esp32-wifi#the-private-gateway>
- [11] ESP32-WiFi-Netzwerk: Viewing Wi-Fi Traffic with Wireshark: <https://tinyurl.com/wsviewtraffic>
- [12] Wokwi-Simulator — Erweiterung Visual Studio Code: <https://tinyurl.com/wokwivsext>
- [13] Projekt konfigurieren (_wokwi.toml_): <https://docs.wokwi.com/vscode/project-config>
- [14] Wokwi — Debugging des Codes: <https://docs.wokwi.com/vscode/debugging>
- [15] Projekt konfigurieren — IoT-Gateway (Port-Weiterleitung): <https://docs.wokwi.com/vscode/project-config#iot-gateway-esp32-wifi>
- [16] Wokwi-Simulator nutzen mit der Espressif-IDE: <https://tinyurl.com/wokwiespide>
- [17] Einstieg in die Custom-Chips-C-API von Wokwi: <https://docs.wokwi.com/chips-api/getting-started>
- [18] _wokwi-ci-action_: <https://github.com/wokwi/wokwi-ci-action>
- [19] _wowkwi-cli_: <https://github.com/wokwi/wokwi-cli>
- [20] _platform-io-esp32-counter-ci_ — Code für ein Automation-Szenario: <https://tinyurl.com/pushcnnstt>

Revolutionieren Sie Ihre Produktpalette mit WiFi Motion™

– jetzt verfügbar auf Espressif WiFi-Chips!



Entdecken Sie die Zukunft der Bewegungssensorik

Kontaktieren Sie uns jetzt unter www.cognitivesystems.com

COGNITIVE 



Bild 1. Das Kit ESP32-S3-BOX-3 im ausgepackten Zustand. Nicht abgebildet sind das USB-Kabel und das winzige RGB-LED-Modul mit den vier Jumper-Drähten. Die Nektarine ist nicht im Kit enthalten.

Ausprobiert: die ESP32-S3-BOX-3

Eine umfassende AIoT-Entwicklungsplattform

Von Clemens Valens (Elektor)

Die ESP32-S3-BOX-3 von Espressif ist eine Plattform für die Entwicklung von Anwendungen wie persönliche Assistenten, intelligente Lautsprecher und andere sprachgesteuerte Geräte. Werfen wir einen genaueren Blick darauf!

Das ESP32-S3-BOX-3-Kit [1] basiert auf einem ESP32-S3 und wird als „Ihr nächstes AIoT-Entwicklungstool“ vermarktet. AIoT steht für *Artificial Intelligence of Things* (Künstliche Intelligenz der Dinge) und ist eng verwandt, aber nicht zu verwechseln mit IoT, was Internet of Things bedeutet (wie Sie sicher schon wussten). Das Kit hat die Form einer kleinen Konsole mit einem Touchscreen. Vorgeschlagene Anwendungen sind KI-Chatbots, Matter (ein einheitliches Protokoll für die Hausautomatisierung), Robotersteuerungen und intelligente Sensorgeräte.

Im Inneren der Box

Die ESP32-S3-BOX-3 (Bild 1) wird als eine dieser sich langsam öffnenden Boxen geliefert (um den Wow-Ef-

fekt zu verbessern, wie mir jemand sagte, der bei Apple gearbeitet hat). Wenn man die Box öffnet, sieht man ein Display-Modul (55 mm × 50 mm) mit einer Art Ständer dafür und einem etwa 30 cm kurzen USB-C-Kabel. Doch dies ist nur die oberste Schicht. Unter dem Ständer verbirgt sich ein zweiter, etwas kleinerer Ständer, unter dem Display-Modul befindet sich ein weiterer kleinerer Ständer und matroschka-artig ein noch viel kleinerer viarter, der auf ein Breadboard gesteckt werden kann. Neben dem USB-Kabel befindet sich eine kleine Tasche, die ein kleines RGB-LED-Modul und vier Jumper-Drähte enthält.

Display-Modul

Das Displaymodul ist erstaunlich schwer (60 g) für ein solch kleines Objekt (55 mm × 50 mm × 12 mm BHT), doch in dem Gehäuse verbirgt sich auch die ganze Kraft: ein Modul ESP32-S3-WROOM-1-N16R16V mit 2,4-GHz-WLAN (802.11b/g/n) und Bluetooth 5 (BLE). Das TFT-Display mit kapazitivem Touch besitzt eine Diagonale von 2,4" und löst 300 × 240 Pixel auf. Falls Sie sich gewundert haben, nein, es gibt keinen Akku im Inneren (Bild 2).

Auf der Vorderseite befinden sich neben dem Display zwei kleine Löcher (die Mikrofone) und ein roter Kreis (die Eingabetaste). Auf der linken Seite sieht man zwei Drucktasten (Boot und Reset) und einen USB-C-Anschluss; auf der rechten Seite etwas, das wie ein microSD-Kartenslot aussieht, aber in Wirklichkeit ein Lautsprecher ist. Auf der Oberseite befinden sich eine Stummschalttaste und zwei LEDs; an der Unterseite ragt ein PCIe-Anschluss heraus. Auf der Rückseite des Moduls befindet sich: nichts.

Ständer und Halterungen

Der PCIe-Anschluss wird mit einer Buchse an einem der Ständer oder Halterungen verbunden. Wie bereits erwähnt, gibt es vier davon, jeder mit unterschiedlichen Funktionen (**Bild 3**). Der oben links abgebildete Ständer ist der DOCK. Es hat zwei Pmod-Verlängerungsbuchsen (2x6-polige Buchsenleisten) auf der Rückseite, zusammen mit einer USB-C-Buchse für die Stromversorgung (Ein- und Ausgang). Eine USB-A-Host-Buchse ist auf der linken Seite vorhanden. Ein Etikett zeigt die Namen der Signale, die an den Pmod-Anschlüssen zugänglich sind. Der größte Ständer wird SENSOR genannt. Er hat einen Temperatur- und Luftfeuchtigkeitssensor mit winzigem Ein/Aus-Schalter auf der Rückseite, einen microSD-Kartenslot (diesmal kein Lautsprecher) auf der linken Seite und einen USB-C-Stromanschluss auf der rechten Seite. Auf der Vorderseite befinden sich ein IR-Sensor und eine LED, und ein wirklich cooles Feature, wenn Sie mich fragen: ein Radar. In dieser Halterung ist Platz für einen 18650er-Akku.

Der dritte Ständer ist der BRACKET, der dank der beiden Schrauben auf der Rückseite an etwas anderem befestigt werden kann. Dieser Ständer hat ebenfalls zwei Pmod-Header und eine USB-C-Buchse. Hier gibt es keine Beschriftungen, also nehmen wir an, dass die Pmod-Stecker auf die gleiche Weise verdrahtet sind wie beim DOCK.

Der kleinste Ständer, das BREAD, ist ein einfaches 24-Wege-Break-out-Board, das die PCIe-Anschlüsse auf ein Breadboard führen kann. Die Pins an der Breadboard-Seite sind deutlich beschriftet.

Erstes Einschalten

Wenn Sie das Anzeigemodul zum ersten Mal einschalten, bootet es in einen Modus, der sich später als Hilfs- oder Hinweismodus herausstellt. In diesem Modus werden Sie durch einige Hinweisbildschirme geführt, die einige grundlegende Funktionen des Geräts erklären. Danach wechselt es in den Normalmodus. In diesem Modus können Sie durch sechs Funktionen blättern: Sensorüberwachung, Gerätesteuerung, Netzwerk, Media Player, Hilfe und Über uns.

Sprachsteuerung

Laut den Hilfebildschirmen kann man einige Funktionen per Sprache aktivieren, indem man „Hai ie es pie“ sagt (ESP wird Buchstabe für Buchstabe ausgesprochen). Da bei meinem Versuch nichts passierte, habe ich das

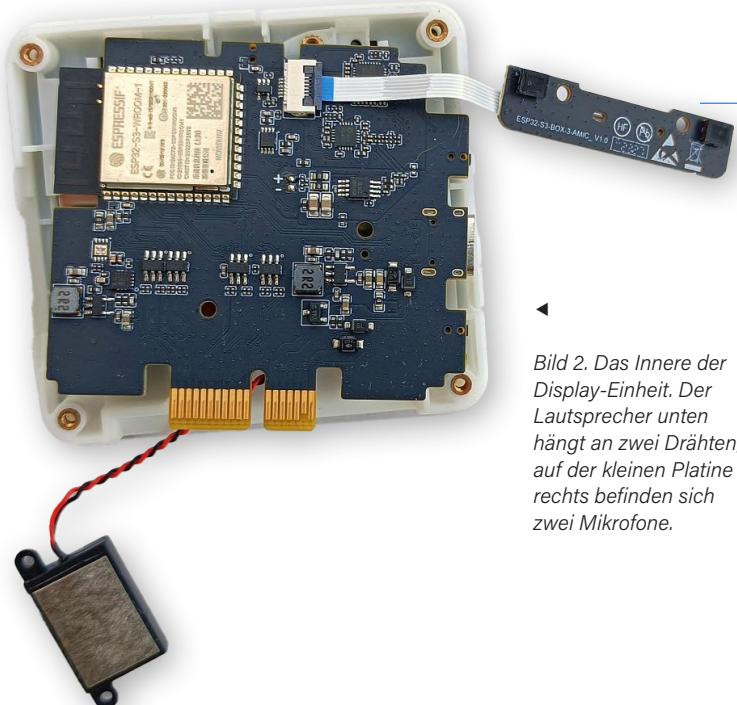


Bild 2. Das Innere der Display-Einheit. Der Lautsprecher unten hängt an zwei Drähten; auf der kleinen Platine rechts befinden sich zwei Mikrofone.

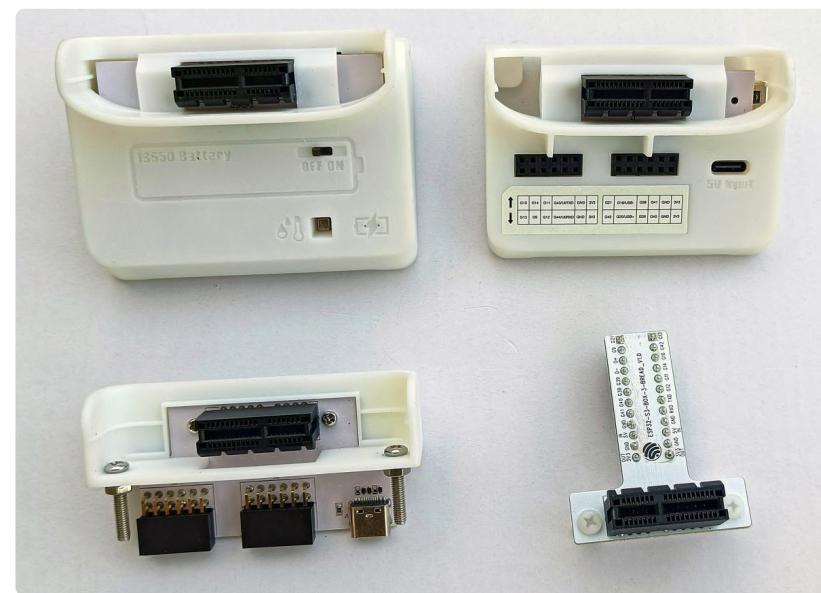
Benutzerhandbuch [2] heruntergeladen. Dazu ist auf der Unterseite der Verpackung ein QR-Code aufgedruckt. Laut Handbuch funktioniert die Sprachsteuerung in jedem Bildschirm (es sei denn, die Mute-LED leuchtet). Da ich das wusste, habe ich es noch einmal versucht und es ein paar Mal zum Laufen gebracht.

Wenn Sie Erfolg haben, hören Sie einen Ping-Ton und auf dem Display wird ein Mikrofon angezeigt. Sie haben nun sechs Sekunden Zeit, um einen Befehl auszusprechen. Nach sechs Sekunden Inaktivität sagt das Gerät „Timeout, see you next time“ und kehrt in den Modus zurück, in dem es sich vor der Aktivierung befand. Ich fand es sehr schwierig, das Gerät aufzuwecken, aber sobald es wach war, erkannte es meine Befehle problemlos. Seltsam.

Radar

Eine interessante Funktion entdeckte ich, als ich vom Nachfüllen meines Kaffeebechers zurückkam. Das Display hatte sich während meiner Abwesenheit ausgeschaltet und schaltete sich plötzlich wieder ein. Das muss die

Bild 3. Vier Halterungen ermöglichen alle möglichen Anwendungen.



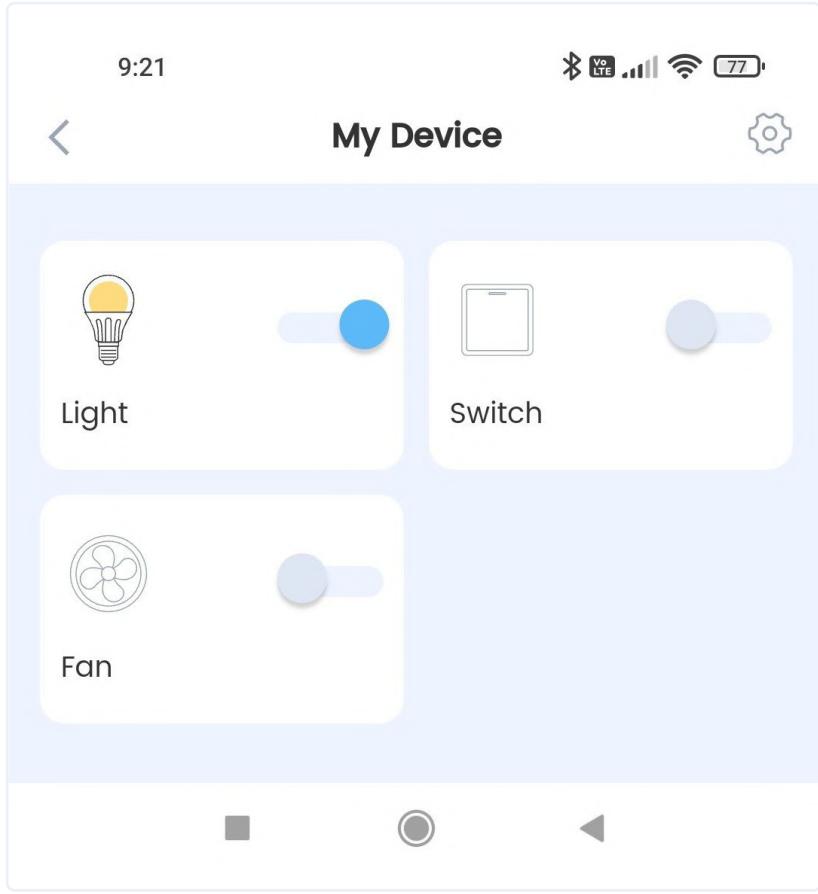


Bild 4. Die App *ESP BOX* stellt eine Verbindung zwischen dem Kit und dem Rainmaker-Cloud-Service von Espressif her.

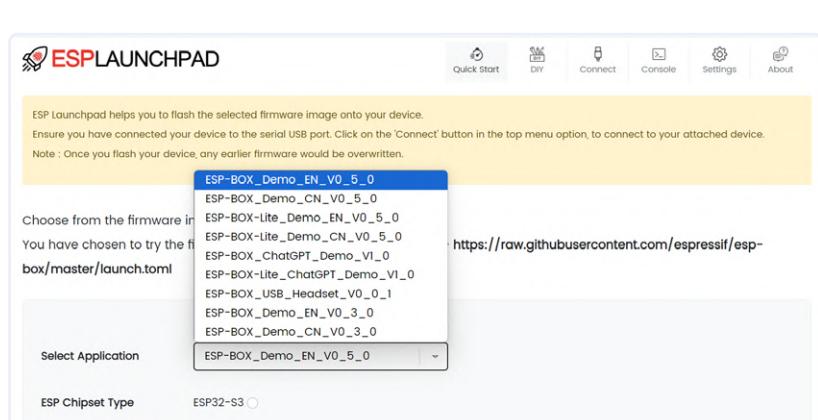
Arbeit des eingebauten Radars sein. Wenn man sich eine Weile nicht bewegt, schaltet sich das Gerät auch wieder aus.

Die Box in der Wolke

Nachdem ich ein wenig mit den eingebauten Apps herumgespielt hatte (schalten Sie das Display aus/ein, falls Sie es „hotplug“ an der Sensorhalterung angeschlossen haben, sonst funktioniert es nicht), ging ich zur ESP-BOX-App über. Sie können sie über die Netzwerksseite aufrufen, nachdem Sie auf die Schaltfläche *To install APP* getippt und den QR-Code gescannt haben.

Mit der App können Sie das Kit (und andere Geräte) über ein WLAN mit dem *Espressif Rainmaker Cloud-Service* verbinden. Nach dem Hinzufügen des Geräts zur App haben Sie Zugriff auf die gleichen Steuerelemente wie auf der Seite Gerätesteuerung (Licht, Schalter und Ventilator, aber nicht Air, **Bild 4**). Auf Ihrem Handy können Sie nun diese Funktionen umschalten; das Ergebnis wird auf dem Display der ESP32-S3-BOX-3 angezeigt. Allerdings

Bild 5. *ESP Launchpad* ist ein Online-Tool zur Firmware-Programmierung und ein serielles Terminal.



scheint dies nur eine einseitige Kommunikation zu sein, denn das Umschalten einer Taste auf dem Display aktualisiert die App nicht.

Tippt man in der App auf das Symbol in der Schaltfläche statt daneben auf den Schiebeschalter, öffnet sich eine Einstellungsseite. Hier können Sie festlegen, welche GPIO-Pins von dem Button gesteuert werden und was die Sprachsteuerungsbefehle bewirken. Sie können auch die Sprachbefehle neu definieren, indem Sie einfach eine neue Befehlsphrase eintippen. Einige Tipps für gute Befehle finden sich im Benutzerhandbuch. Da ich das Gerät nicht zum Aufwachen bringen konnte, war es mir leider nicht möglich, neue Befehle auszuprobieren.

Ich habe kein Dokument gefunden, das erklärt, wie man eigene Anwendungen für die ESP-BOX-App erstellt.

Serieller Anschluss

Sie können die Vorgänge im Gerät überwachen, wenn Sie die Anzeigeeinheit an einen Computer anschließen und ein serielles Terminal öffnen. Diese Funktion hat sich als sehr wertvoll für den nächsten Abschnitt erwiesen.

ChatGPT in ESP-Launchpad

Auf der Unterseite der ESP32-S3-BOX-3 befinden sich drei QR-Codes. Einer ist für *ESP Launchpad* [3]. Diese App funktioniert nur in Chrome (zumindest unter Windows) und ist dafür gedacht, das Kit mit anderer Firmware neu zu programmieren. Es scheint, dass man hier sogar seine eigene Firmware veröffentlichen kann, so dass auch andere Leute sie ausprobieren können (**Bild 5**). Zuerst wollte *ESP Launchpad* sich aus irgendinem Grund nicht mit meinem Gerät verbinden, aber nachdem ich es neu gestartet hatte, bekam ich eine Liste von Demoprogrammen.

Aus dieser Liste wählte ich (natürlich) *ESP-BOX_ChatGPT_Demo_V1_0* aus, lud es eifrig auf das Anzeigegerät hoch und... nichts. Der Bildschirm blieb schwarz. Dann versuchte ich es mit der nächstbesten Demo, *ESP-BOX-Lite_ChatGPT_Demo_V1_0*, aber ich erhielt ein ähnliches Ergebnis. Nachdem ich noch ein paar weitere Beispiele erfolglos ausprobiert hatte, dämmerte es mir plötzlich (danke, serielle Schnittstelle): Diese Dateien sind nicht für die ESP32-S3-BOX-3, sondern für frühere Versionen des Kits, die EPS32-S3-BOX (ohne -3) und die ESP32-S3-BOX-Lite gedacht. Hoffentlich ist dieses Problem gelöst, wenn Sie diese Zeilen lesen.

Eigene Anwendungen bauen

An diesem Punkt fand ich mich auf GitHub wieder, in dem Repository, das alle drei Versionen der ESP32-S3-BOX unterstützt [4]. Warum hat man nicht einen großen QR-Code für diese Seite auf die Box gedruckt? Das Repository wird auch nicht im Benutzerhandbuch erwähnt, obwohl es viele Informationen über das Kit und auch den Quellcode für die Beispielprogramme enthält. Es gibt keine vorkompilierten, leicht auszuprobierenden ausführbaren Dateien, zumindest habe ich keine gefunden, also müssen Sie sie selbst bauen. Dazu klonen Sie

zuerst das Repository. Sie benötigen außerdem ESP-IDF V5.1 oder höher.

Das Kompilieren und Hochladen auf das Gerät hat eine Weile gedauert, aber als es geschafft war, war ich endlich wieder am Anfang dieses Artikels, als die Anzeigeeinheit noch funktionierte. Zu meiner Überraschung schien die Sprachsteuerung besser zu reagieren, und das Aufwecken des Geräts war einfacher. Die Softwareversion war immer noch V1.1.1, aber meine EPS-IDF-Version war zu v5.11-dirty geworden (vorher v5.2-dev-2164-g3befd5fff7-dirty).

ChatGPT-Demo, Schritt 2

Da ich nun eine Entwicklungsumgebung für die ESP32-S3-BOX-3 installiert hatte, beschloss ich, die ChatGPT-Demo gemäß der mitgelieferten Anleitung zu erstellen. Das klappte gut.

Nachdem ich das Programm kompiliert und auf das Display hochgeladen hatte, startete die Demo normal und zeigte Anweisungen zur Konfiguration an [5]. Danach verband sie sich mit meinem WLAN und ich konnte mit dem Chatbot sprechen. Allerdings antwortete der Chatbot nicht auf meine Fragen, sondern zeigte stattdessen die Meldung *API Key is not valid* an. Im seriellen Terminal fand ich diese Zeile:

```
E (369916) OpenAI: You exceeded your current
quota, please check your plan and billing
details.
```

Es stellte sich heraus, dass man ein gewisses Guthaben auf seinem ChatGPT-Konto benötigt, um API-Schlüssel zu verwenden. Und mein Guthaben war erschöpft. Wenn Sie zum ersten Mal ein Konto erstellen (Sie benötigen dazu eine Telefonnummer), erhalten Sie ein kostenloses Guthaben zum Einstieg. Da ich keinen Zahlungsplan einrichten wollte, habe ich an dieser Stelle aufgehört.

Beispiel für die Bildanzeige

Schließlich beschloss ich, ein letztes Beispiel auszuprobieren, das Bild-Display. In der Beschreibung steht nicht, was es tut, das wäre also eine nette Überraschung. Dieses Beispiel ist etwas kleiner als die anderen, so dass es sich schneller kompilieren und hochladen lässt. Das Ergebnis? Eine Liste mit sechs Smileys, aus der Sie einen auswählen können, der angezeigt werden soll (**Bild 6**).

Komplex und viel zu bieten

Für dieses Review habe ich etwas mehr Zeit benötigt, als ich zu Beginn erwartet hatte. Der Grund dafür war die Komplexität des Produkts. Die ESP32-S3-BOX-3 hat sicherlich viel zu bieten, aber um etwas Nützliches aus ihr herauszuholen, muss man Zeit und Mühe investieren. Man sollte bei der Erkundung der ESP32-S3-BOX-3 mit dem GitHub-Repository beginnen und nicht mit den QR-Codes auf der Rückseite der Box. Auf GitHub finden Sie nicht nur das Benutzerhandbuch, sondern auch andere nützliche Dokumente und Beispielprogramme. Es hätte mir eine Menge Zeit erspart, wenn ich das von

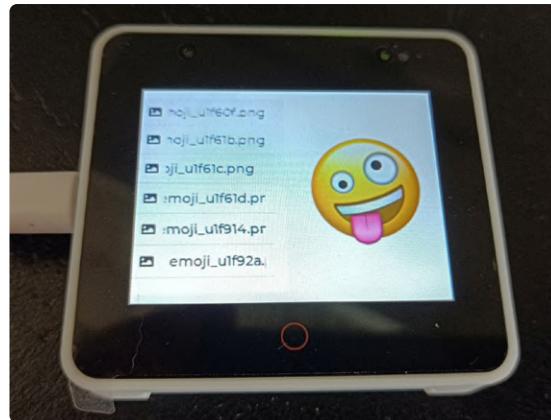


Bild 6. Noch ein weiter Weg zur Künstlichen Intelligenz der Dinge?

Anfang an gewusst hätte.

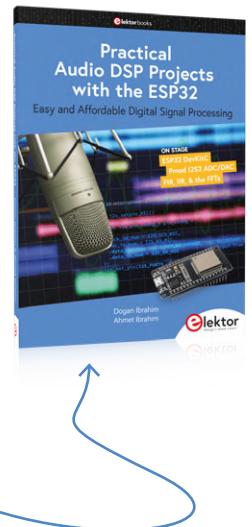
Obwohl es ziemlich viele Informationen gibt, fühlt sich die Unterstützung immer noch ein wenig unvollständig an. Ein Tutorial, wie man mit der ESP-BOX-App arbeitet oder wie man eine sprachgesteuerte Anwendung von Grund auf erstellt, wäre wünschenswert. Es gibt nicht viel Material zum Thema IoT, aber genau darum geht es doch bei diesem Produkt, oder?

Was die Hardware angeht, kann ich nur sagen, dass sie wirklich gut gemacht ist. Die Verarbeitungsqualität ist gut, die Halterungen passen perfekt, die Tasten funktionieren, das Display sieht schick aus, die Berührungs-empfindlichkeit ist gut, und alles ist in einem hochwertigen Karton verpackt. Ob mit oder ohne Halterung, das Display ist ein cooles Gadget auf dem Schreibtisch, im Schlafzimmer oder im Wohnzimmer. ↗

RG - 230555-02

Haben Sie Fragen oder Kommentare?

Haben Sie technische Fragen oder Kommentare zu diesem Artikel? Schicken Sie eine E-Mail an den Autor unter clemens.valens@elektor.com oder kontaktieren Sie Elektor unter redaktion@elektor.de.



Passende Produkte

- **ESP32-S3-BOX-3**
www.elektor.de/20627
- **Arduino Nano ESP32**
www.elektor.de/20562
- **A. und D. Ibrahim, Practical Audio DSP Projects with the ESP32 (Elektor 2020)**
Buch, Paperback, englisch:
www.elektor.de/20558
E-Buch, PDF, englisch: www.elektor.de/20559

WEBLINKS

- [1] **ESP32-S3-BOX-3:** <https://www.espressif.com/en/news/ESP32-S3-BOX-3>
- [2] **Benutzerhandbuch:** <https://qr10.cn/CoahPA>
- [3] **ESP-Launchpad:** <https://qr10.cn/DCgKrD>
- [4] **ESP32-S3-BOX-3 auf GitHub:** <https://github.com/espressif/esp-box>
- [5] **ChatGPT Secret Key:** <https://platform.openai.com/account/api-keys>

Mein ELEKTRONIK-ARBEITSPLATZ

Einblicke und Tipps von Espressif-Ingenieuren

Alle Magie entsteht im Elektronik-Arbeitsbereich. Sind Sie neugierig, was andere kreative Ingenieure in ihrem Arbeitsraum haben? Suchen Sie nach Tipps für die Einrichtung und Organisation Ihres Elektronik-Arbeitsplatzes? Einige Espressif-Ingenieure geben Ihnen einen Einblick.



Omar Chebib

Standort: Shanghai, China

Organisieren und Notizen machen

Mein Arbeitsplatz ist mein Schreibtisch zu Hause. Er ist zwar nicht groß, aber ich versuche, ihn immer sauber und organisiert zu halten. Ich bewahre nur die Werkzeuge, Geräte und Komponenten auf, die ich für meine aktuellen Aufgaben benötige. Indem ich sie konsequent an den dafür vorgesehenen Stellen platziere, kann ich die Effizienz meiner Arbeit aufrechterhalten. Dazu gehört natürlich auch ein gut organisiertes Regal mit Fächern, die sowohl reine Elektronik mit Durchsteck- und SMD-Bauteilen als auch Embedded-Werkzeuge wie einen Logikanalysator, einigen ESP32-Chips oder sogar I²C-Bauteilen enthalten. Dies ermöglichte mir die Arbeit an mehreren Projekten mit Logikgattern, einem 8-Bit-Z80-Prozessor, FPGA, für Durchsteck- und Oberflächenmontage, einschließlich der „schrecklichen“ BGA-Gehäusen. Ein guter Rat: Auf der Hardwareseite sollten Sie nur die Werkzeuge, Geräte und Kabel auf Ihrem Schreibtisch haben, die Sie für die aktuelle Aufgabe benötigen, neue Werkzeuge sollten Sie nur dann kaufen, wenn Sie sie wirklich brauchen, und nach der Arbeit aufräumen. Was die Software betrifft, so sollten Sie Ihre Arbeit immer versionskontrollieren, auch wenn sie lokal erfolgt. Es ist nie zu spät, mit der Versionierung eines bereits bestehenden Projekts zu beginnen. Ganz allgemein sollten Sie beim Wechsel zwischen Projekten eine Notiz anlegen, in der Sie erklären, was Sie bereits getan haben, wo Sie aufgehört haben und was der nächste Schritt ist. Das hilft sehr, mit dem alten Projekt wieder in Schwung zu kommen. Meine aktuellen Projekte: In meiner Freizeit habe ich einen Z80-basier-



ten Zeal-8-Bit-Computer von Grund auf neu entwickelt, vom Platinendesign bis zur Software. In meiner Arbeitszeit implementierte ich eine ESP32-C3-Emulation auf QEMU.

Unverzichtbar

- Logikanalysator
- Multimeter
- Oszilloskop
- Elektronisches Mikroskop
- Lötstation + Flussmittel
- Eine gute Dunstabzugshaube!
- Linux-Rechner

Wunschzettel

- Heizplatte zum Reflow-Löten
- Besseres Mikroskop
- Besserer Stuhl

